Combining the benefits of function approximation and trajectory optimization

Igor Mordatch
University of Washington
mordatch@cs.washington.edu

Emanuel Todorov
University of Washington
todorov@cs.washington.edu

Abstract—**Neural networks have recently solved many hard problems in Machine Learning, but their impact in control remains limited. Trajectory optimization has recently solved many hard problems in robotic control, but using it online remains challenging. Here we leverage the high-fidelity solutions obtained by trajectory optimization to speed up the training of neural network controllers. The two learning problems are coupled using the Alternating Direction Method of Multipliers (ADMM). This coupling enables the trajectory optimizer to act as a teacher, gradually guiding the network towards better solutions. We develop a new trajectory optimizer based on inverse contact dynamics, and provide not only the trajectories but also the feedback gains as training data to the network. The method is illustrated on rolling, reaching, swimming and walking tasks.**

I. Introduction and related work

While robots are able to perform increasingly impressive tasks, designing the underlying controllers remains tedious. Smooth and low-dimensional dynamical systems such as quadrotors admit principled solutions based on nonlinear control theory [8, 16]. But more complex domains such as legged locomotion or hand manipulation still require adhoc approaches, where the control problem is broken down into multiple subproblems whose solutions are manually adjusted and then pieced together.

Compare this to the rapid march towards automation taking place in Machine Learning and related areas. It is now possible to train neural networks to perform vision or speech recognition tasks with remarkable accuracy [10, 27, 6]. Equally remarkable is the level of automation and the simplicity of the training procedures. And this is not just a recent phenomenon; convolution networks have been among the top performers in character recognition for years [12].

So why are we not enjoying the same benefits in control? It is not because people have not tried. Indeed control was among the early applications of neural networks [17], and there is still intense interest in this topic in multiple communities. Reinforcement Learning and Approximate Dynamic Programming [4, 32, 28] have given rise to many techniques for function approximation in the context of control. These include indirect methods based on value function approximation [11] as well as direct policy search methods [31, 22]. Data-intensive methods have also been pursued in imitation learning [26]. Some of the most interesting robotic behaviors generated by neural networks are swimming and crawling [9] but they do not involve control hazards such as falling, dropping an object or hitting an obstacle. When such hazards are present, we doubt that any roboticist would leave their robot in the hands of a neural network trained with existing techniques.

Our goal is to make automatic controllers based on neural networks or other function approximators work better – particularly in complex domains that involve contact dynamics, underactuation, potential instabilities and a general need for precision. Is there any approach to automatic control that actually works in such domains? The only answer we are aware of is trajectory optimization [38]. We are not referring to optimization through reduced models (such as inverted pendulum models), but rather to full-state optimization taking into account the full dynamics. Such optimization is rarely used on real robots, even though model-free methods exist for adapting a given trajectory to unknown dynamics [34]. But at least in simulation, recent advances suggest that many problems in robotic control can be solved fully automatically [19, 18, 23, 2]. This is done by specifying a high-level cost (the same cost that would be used in policy search), and letting the optimizer compute a detailed control sequence and corresponding trajectory that accomplish the task optimally starting at a given state. The underlying numerical optimization is by no means easy, and only becomes feasible if contact dynamics are handled carefully and suitable continuation is used. Nevertheless these methods are now able to synthesize complex and physically realistic movements such as walking, running, climbing, acrobatics, dexterous hand manipulation as well as cooperative multi-character behaviors. The CPU time needed to perform the optimization (without good initialization) is between minutes and hours depending on which method is used. In some cases trajectory optimization is applicable online in model-predictive control [1, 33], and Moore's law will only make it more applicable in the future. But even if brute-force online computing is a viable option for control of complex robots, it is still desirable to somehow cache the solutions and train a function approximator that can operate at a fraction of the computational cost.

Combining function approximation with trajectory optimization should in principle be straightforward given that they both aim to solve the same optimal control problem. Indeed deterministic optimal control has two formulations [30], one based on dynamic programming leading to function approximation, and the other based on Pontryagin's maximum principle leading to trajectory optimization. In practice however it is not easy to get a trajectory optimizer to help a neural network. The obvious approach would be to generate many optimal

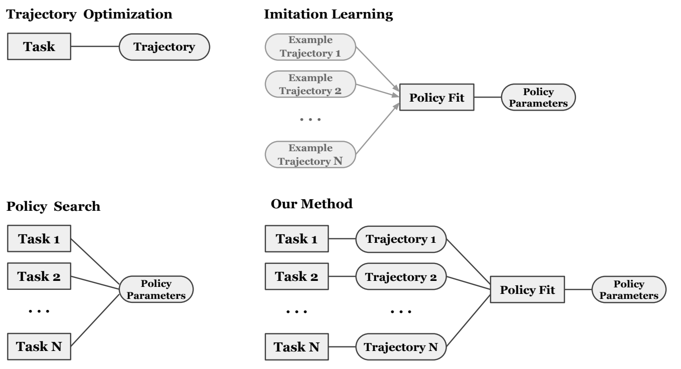

Fig. 1. Factor graphs of various optimal control algorithms compared to our method. Parameters being optimized are in circles and objective factors are rectangles.

trajectories (for different starting states) and use them as training data – similar to imitation learning but with synthetic instead of motion capture data. This is related to DAGGER [25], which in earlier comparisons [13] did not perform as well as the type of approach we advocate here. We will see below that straightforward learning from optimized trajectories is problematic because in the early phases of training the network is not sufficiently competent to utilize such complex data. In general, if traditional learning from trajectory data was going to produce good controllers for complex tasks, that would have already happened since human motion capture data has been abundant for a while. Alternatively one can use the local value function approximations generated by the trajectory optimizer to fit a more global approximator. This was attempted recently [39], and even though the fit was accurate the resulting control law was not; this is because value function approximation errors are not monotonically related to controller sub-optimality.

A. Our approach

The alternative which we consider is to use the trajectory optimizer as a teacher rather than a demonstrator. A good teacher adapts to the student, providing guidance in small steps which the student is able to assimilate. Here this will be done by modifying the cost for the trajectory optimizer, so that it still solves the control problem but in a way similar to how the student (fails to) solve it. On this level of abstraction our work is closely related to [14, 13]. The differences are in the technical approach. Here we use the Alternating Direction Method of Multipliers (ADMM) [5] to couple the network and trajectory optimizer, while in [13] the coupling was done by casting the control problem as a dual Bayesian inference problem [37] and exploiting variational approximations. Furthermore, here we utilize a recent method to trajectory optimization based on contact-invariant optimization [19], while [14, 13] used the iterative LQG method [35] through the forward dynamics. Another difference is that we use not only the trajectories but also the time-varying feedback gains produced by the optimizer as training data for the network. This is based on an older idea known as tangent propagation [29]. We will discuss the implications of these differences in V-E.

Beyond training the neural network to act as an approximately-optimal feedback controller, we aim to make this controller responsive to changes in the task itself. The structure of the task is of course fixed and is implicit in the training that the network received. But if the task has real-valued parameters – such as the location of a target, or the desired speed of movement – we encode these parameters as inputs to the network. In this way the same network can be used to solve a continuum of related tasks. Figure 1 illustrates how the proposed approach relates to more traditional optimization approaches in control.

II. Preliminary Description

A. Problem Formulation

In this paper we consider deterministic, discrete-time dynamical systems $\mathbf{x}^{t+1} = \mathbf{f}(\mathbf{x}^t, \mathbf{u}^t)$ and task cost functions $l(\mathbf{x}^t, \mathbf{u}^t)$. $\mathbf{x}^t$ and $\mathbf{u}^t$ are respectively the state of the system and controls applied at time t. Let X and U denote an entire trajectory of these quantities. In this work we assume the model is known and the state is fully observable. If necessary, we can encode any relevant task parameters into $\mathbf{x}$. For example, $\mathbf{x}$ could be joint angles of an arm and target location we want an arm to reach, while $\mathbf{u}$ are arm muscle activations. Given a specific initial state $\mathbf{x}^{\text{init}}$ and task (also encoded in $\mathbf{x}^{\text{init}}$), one can find an optimal trajectory $\left[\bar{X}\ \bar{U}\right]$ that achieves the task and satisfies the dynamics constraints:

$$\left[\bar{X}\ \bar{U}\right] = \underset{\mathbf{u}}{\operatorname{argmin}} \sum_{t=0}^{T} l(\mathbf{x}^t, \mathbf{u}^t),$$
$$\text{s.t.} \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t) \quad \text{and} \quad \mathbf{x}^0 = \mathbf{x}^{\text{init}} \tag{1}$$

This is the deterministic trajectory optimization problem, and several algorithms exist to solve it. But the solution is specific to only one situation. We wish to learn a control policy $\pi(\mathbf{x}|W) = \mathbf{u}$ parametrized by W, such that for any given initial state and any task it outputs controls achieving the task. The problem is then to find policy parameters that give good performance in expectation – which we approximate with sample average:

$$\bar{W} = \underset{W}{\operatorname{argmin}} \frac{1}{N} \sum_{i=0}^{N} \left[\sum_{t=0}^{T} l(\mathbf{x}_i^t, \pi(\mathbf{x}_i^t|W)) \right],$$
$$\text{s.t.} \quad \mathbf{x}_i^{t+1} = f(\mathbf{x}_i^t, \pi(\mathbf{x}_i^t|W)) \quad \text{and} \quad \mathbf{x}_i^0 = \mathbf{x}_i^{\text{init}} \tag{2}$$

It is possible to derive a gradient for the above objective and apply gradient-based optimization, but that presents a number of challenges as discussed in [13].

B. Desired Joint Optimization Problem

We instead propose to jointly optimize a collection of trajectories each achieving a particular task, and parameters of a single policy that can reconstruct all these trajectories.

$$
\begin{aligned}
\left[\bar{X}\ \bar{U}\ \bar{W}\right] = \underset{X,U,W}{\operatorname{argmin}} \frac{1}{N} \sum_{i=0}^{N} L(X_i, U_i) + R(X, U, W), \\
\text{s.t.} \quad \mathbf{x}_i^{t+1} = f(\mathbf{x}_i^t, \mathbf{u}_i^t) \quad \text{and} \quad \mathbf{x}_i^0 = \mathbf{x}_i^{\text{init}} \qquad (3)
\end{aligned}
$$

Where $L(X, U) = \sum_t l(\mathbf{x}^t, \mathbf{u}^t)$ is a total trajectory task cost and $R(X, U, W) = \sum_{i,t} \frac{1}{2} \|\pi(\mathbf{x}_i^t|W) - \mathbf{u}_i^t\|^2$ is the total policy reconstruction cost.

In policy search methods, this can be viewed as relaxing the constraint that controls can only come from a policy function. In trajectory optimization methods, this can be seen as adding an additional regularizer that trajectories have to be recreatable by a policy. This is a very large optimization problem and is intractable for anything but the simplest settings. Instead we will distribute this problem into a collection of independent subproblems that each have an effective, well-studied and readily available solution method. There are a number of ways to solve the resulting distributed optimization problem. In this work we use Alternating Direction Method of Multipliers (ADMM) [5].

C. Alternating Direction Method of Multipliers

ADMM is a method that can be used to efficiently optimize objective functions composed of terms that each independently have efficient solution methods. For our purposes, consider the following problem:

$$\min_{\mathbf{a}} f(\mathbf{a}) + g(\mathbf{a})$$

We can make two copies of the unknown variable and constrain the two to be equal:

$$\min_{\mathbf{a},\mathbf{b}} f(\mathbf{a}) + g(\mathbf{b}), \quad \mathbf{a} - \mathbf{b} = \mathbf{0}$$

We can solve this constrained problem by minimizing its augmented Lagrangian:

$$\mathcal{L}(\boldsymbol{\lambda}) = \underset{\mathbf{a},\mathbf{b}}{\operatorname{argmin}} f(\mathbf{a}) + g(\mathbf{b}) + \rho \boldsymbol{\lambda}^T (\mathbf{a} - \mathbf{b}) + \frac{\rho}{2} \|\mathbf{a} - \mathbf{b}\|^2$$

This Lagrangian can be minimized in an alternating Gauss-Seidel manner, optimizing each variable holding others fixed (updating $\boldsymbol{\lambda}$ by gradient descent). This gives the following iterative update scheme:

$$
\begin{aligned}
\bar{\mathbf{a}} &= \underset{\mathbf{a}}{\operatorname{argmin}} f(\mathbf{a}) + \frac{\rho}{2} \|\mathbf{a} - \bar{\mathbf{b}} + \boldsymbol{\lambda}\|^2 \\
\bar{\mathbf{b}} &= \underset{\mathbf{b}}{\operatorname{argmin}} g(\mathbf{b}) + \frac{\rho}{2} \|\bar{\mathbf{a}} - \mathbf{b} + \boldsymbol{\lambda}\|^2 \\
\boldsymbol{\lambda} &= \boldsymbol{\lambda} + (\bar{\mathbf{a}} - \bar{\mathbf{b}})
\end{aligned}
$$

The resulting scheme can be more efficient if minimizations of f and g independently have efficient (or closed-form) solution methods. If f and g are convex, the resulting scheme converges to a global optimum, however this method has also successfully been applied to non-convex problems [3].

III. Algorithm Description

We now describe the application of ADMM to solving problem (3). We introduce two versions of each state ($\mathbf{x}^L$ and $\mathbf{x}^R$) and control ($\mathbf{u}^L$ and $\mathbf{u}^R$). These correspond to states and controls minimizing trajectory cost and policy reconstruction cost, respectively. The analogue of augmented Lagrangian for this problem is:

$$
\begin{aligned}
\mathcal{L}(X^\lambda, U^\lambda) = \operatorname{argmin} \frac{1}{N} \sum_i L(X_i^L, U_i^L) + R(X^R, U^R, W) \\
+ \frac{\rho}{2} \|X^L - X^R + X^\lambda\|^2 + \frac{\rho}{2} \|U^L - U^R + U^\lambda\|^2 \\
\text{s.t.} \quad \mathbf{x}_{i,t+1}^L = f(\mathbf{x}_{i,t}^L, \mathbf{u}_{i,t}^L) \quad \text{and} \quad \mathbf{x}_{i,0}^L = \mathbf{x}_i^{\text{init}}
\end{aligned}
$$

We update each of the variables in a block manner as before.

The update for the trajectory variables is:

$$
\begin{aligned}
\left[\bar{X}_i^L\ \bar{U}_i^L\right] = \underset{X,U}{\operatorname{argmin}} \sum_t l(\mathbf{x}^t, \mathbf{u}^t) \\
+ \frac{\rho}{2} \|\mathbf{x}^t - \bar{X}_{i,t}^R + X_{i,t}^\lambda\|^2 + \frac{\rho}{2} \|\mathbf{u}^t - \bar{U}_{i,t}^R + U_{i,t}^\lambda\|^2 \\
\text{s.t.} \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t) \quad \text{and} \quad \mathbf{x}^0 = \mathbf{x}_i^{\text{init}} \qquad (4)
\end{aligned}
$$

This is simply a trajectory optimization problem with two additional quadratic terms in the cost function and can be solved with existing trajectory optimization methods described in III-A.

The update for the policy parameters is:

$$\bar{W} = \underset{W}{\operatorname{argmin}} R(\bar{X}^R, \bar{U}^R, W) \qquad (5)$$

This is simply a regression problem for function π with inputs $\bar{X}^R$ and target outputs $\bar{U}^R$ described in III-C.

The update for the reconstructed variables is:

$$
\begin{aligned}
\left[\bar{X}_t^R\ \bar{U}_t^R\right] = \underset{\mathbf{x},\mathbf{u}}{\operatorname{argmin}} \frac{1}{2} \|\pi(\mathbf{x}, \bar{W}) - \mathbf{u}\|^2 \\
+ \frac{\rho}{2} \|\bar{X}_t^L - \mathbf{x} + X_t^\lambda\|^2 + \frac{\rho}{2} \|\bar{U}_t^L - \mathbf{u} + U_t^\lambda\|^2 \qquad (6)
\end{aligned}
$$

This is a general but small optimization problem that can be performed independently and in parallel for each trajectory.

The updates for the Lagrange multipliers are:

$$
\begin{aligned}
X^\lambda &= X^\lambda + (\bar{X}^L - \bar{X}^R) \\
U^\lambda &= U^\lambda + (\bar{U}^L - \bar{U}^R) \qquad (7)
\end{aligned}
$$

We can think of $\boldsymbol{\lambda}$ as a running cost indicating where the policy function makes prediction errors. The above algorithm then changes the regression loss function *and* the trajectory cost function to give more weight to regions where errors are consistently made. Over time this has the effect of modifying the optimal trajectory solutions, to that they become less optimal at the task, but easier to learn a policy for.

Thus, we have reduced the policy learning problem to a sequence of trajectory optimization and regression problems, each of which are well-studied problems with efficient solution methods. In principle any trajectory optimizer, policy and regression method can be used. In practice however the selection of methods is important, especially in harder problems. We have used direct trajectory optimization and neural network policies, which offer a number of advantages. The complete algorithm is summarized below.

Algorithm 1: Distributed Policy Learning

```
1 generate $N$ samples $x_i^{\text{init}} \sim X$
2 initialize $\left[X^R U^R\right]$ with trajectory optimization solutions
3 while not converged do
4     update $\left[X^L U^L\right]$ by solving $N$ trajectory optimization problems (4) in parallel
5     update $W$ by solving regression problem (5)
6     update $\left[X^R U^R\right]$ by solving independent minimizations (6)
7     update $\left[X^\lambda U^\lambda\right]$ using (7)
8 end
```

A. Trajectory Optimization

Trajectory optimization problems i are independent and can be solved in parallel. While there exist several methods to find an optimal trajectory $[X\ U]$ for equation (4), we have used direct trajectory optimization because it was shown to solve very difficult optimization problems [19]. The optimization solves for a sequence of states, with controls recovered implicitly via an inverse dynamics function $\bar{\mathbf{u}}^t = g(\bar{\mathbf{x}}^t, \bar{\mathbf{x}}^{t+1})$.

We use Mujoco library [36] to compute inverse dynamics for all our experiments and contact-invariant optimization approach [20] to handle experiment with contact (such as walker). However, we stress that the only change to traditional problem is two additional quadratic terms, which can easily be incorporated into any existing trajectory optimization framework.

Additional quadratic objectives introduced as part of ADMM in (4) are easier to satisfy with direct optimization methods than with indirect methods, such as DDP or ILQG, because they can be used to directly warm-start the optimization procedure, while initial guess for a control sequence for DDP/ILQG may quickly diverge from its intended target. In each ADMM iteration, we must re-solve N trajectory optimization problems. Warm-starting the trajectory optimization with solution from previous ADMM iteration greatly speeds up convergence.

As we will see in the next section, we are also interested in recovering a locally optimal feedback control law for our optimal trajectories. While this a natural byproduct of DDP/ILQG, it is not immediately clear how to recover it from direct trajectory optimization.

B. Feedback Control Law from Direct Trajectory Optimization

We now show how a locally-optimal feedback control law can be obtained within direct trajectory optimization. This is needed here because our neural networks are trained to reproduce not only the controls but also the derivatives of the controls with respect to the states. Being able to obtain a feedback control law is of course valuable in itself, independent of neural network training. DDP and iLQG methods do this automatically – in fact computing the control law recursively is how they find an improved trajectory. But in the context of direct trajectory optimization we are not aware of prior work where this has been done.

Changing notation slightly, let $\mathbf{x}$ denote the stacked vector of states $\left[\mathbf{x}^t; \mathbf{x}^{t+1}; \cdots; \mathbf{x}^T\right]$ where $\mathbf{x}^t = \mathbf{c}$ is fixed and the remaining states are optimized. Let $\mathbf{x}^*(\mathbf{c})$ be the (possibly local) minimum of the trajectory cost. If we change the fixed state $\mathbf{x}^t$ the solution will change. For each solution we can apply inverse dynamics and recover the control sequence. Thus, in order to obtain a linear feedback control law, all we have to do is differentiate $\mathbf{x}^*(\mathbf{c})$ with respect to $\mathbf{c}$. The minimum is defined with respect to the local quadratic model of the trajectory cost:

$$\mathbf{x}^*(\mathbf{c}) = \arg\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{x}^T\mathbf{b} \quad \text{s.t.} \quad E\mathbf{x} = \mathbf{c}$$

The matrix E extracts the first state from the stacked vector $\mathbf{x}$. This equality-constrained quadratic program can be solved analytically using a Lagrange multiplier λ:

$$\begin{bmatrix} \mathbf{x}^* \\ \lambda \end{bmatrix} = \begin{bmatrix} A & E \\ E^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\mathbf{b} \\ \mathbf{c} \end{bmatrix}$$

Thus the derivative we seek is given by the second column of the above block-inverse:

$$\frac{\partial \mathbf{x}^*}{\partial \mathbf{c}} = \begin{bmatrix} A^{-1}E\left(E^TA^{-1}E\right)^{-1} \\ \left(E^TA^{-1}E\right)^{-1} \end{bmatrix}$$

The resulting changes in control trajectories at time t are:

$$\frac{\partial \mathbf{u}^*}{\partial \mathbf{c}} = \frac{\partial g}{\partial \mathbf{x}}\frac{\partial \mathbf{x}^*}{\partial \mathbf{c}}$$

Where $\frac{\partial g}{\partial \mathbf{x}}$ is a block Jacobian for the entire trajectory. Similar procedure can be repeated to find feedback gains for other timesteps. In the end, we gather all feedback gains in $\bar{U}_X^L$.

C. Policy Fitting

As mentioned, update to policy parameters W in (5) involves solving a regression problem for function π with inputs $\bar{X}^R$ and target outputs $\bar{U}^R$. As shown in the previous section, we also have optimal feedback gains $\bar{U}_X^L$, which provide additional information on how target outputs change wrt inputs. In this work, we make an approximation $\bar{U}_X^R \approx \bar{U}_X^L$. If the policy function we use are differentiable, we can incorporate this information as additional objective in our regression loss function:

$$\sum_{i,t} \frac{1}{2}\|\pi(\mathbf{x}_i^t|W) - \mathbf{u}_i^t\|^2 + \frac{\alpha}{2}\|\frac{\partial \pi}{\partial \mathbf{x}}(\mathbf{x}_i^t|W) - \frac{\partial {\mathbf{u}^*}_i^t}{\partial \mathbf{x}}\|^2 \qquad (8)$$

Where $\frac{\partial \mathbf{u}^*}{\partial \mathbf{x}}$ is an element in $\bar{U}_X^R$. In this work, we choose to neural networks, which can be differentiated wrt to inputs and minimizing the above loss function is known as *tangent propagation* [29]. This additional tangent data greatly helps in network generalization performance and prevents overfitting, while incurring only a modest computational overhead. Recently a special case of tangent propagation has also been reapplied for regularization in machine learning literature under the name of contractive autoencoders [24].

IV. Experiments

We have applied our method to learning control policies for four different dynamical systems and tasks: rolling ball, swimmer, arm, and planar walker.

A. Rolling Ball

The first experiment is a nonholonomic 3-dimensional ball that is in contact with the ground. The ball has two actuators, one along the spinning axis that can roll the ball in heading direction and one along the vertical axis that can change heading. The system has 5 degrees of freedom (3d position, heading and spin). These degrees of freedom and their velocities are the state input to the policy. The initial state is random for each trial. The cost is to reach the origin while minimizing actuation $l(\mathbf{x}, \mathbf{u}) = \|\mathrm{pos}(\mathbf{x})\|^2 + 0.01\|\mathbf{u}\|^2$.

B. Swimmer

The second experiment is a 2-dimensional two-link swimmer that starts in a stationary random pose and must reach and maintain a target forward velocity. The medium has water density, which can create drag force and there is a current force acting against the swimmer (so it cannot simply coast forward). The system has 5 degrees of freedom and two actuators. The cost is a quadratic around the target velocity, and also includes penalties for actuation and high drag forces.

In order to perform this task, the policy must implicitly learn two behaviors: discrete movement initiation to reach the target velocity, and well as cyclic behavior to maintain that velocity. In all cases, our method was able to produce these discrete and cyclic motions.

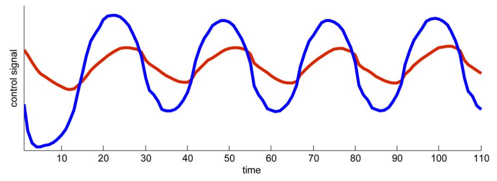

Fig. 2. Example of actuation patterns produced by rolling out a policy produced by our method. Note both movement initiation and cyclic maintenance

C. Arm Reaching

The third experiment is a 3d humanoid arm that starts in a random pose and must reach a spatial target with its end-effector while minimizing actuation. The system has 4 degrees of freedom (shoulder and elbow) and is fully actuated. Aside from the degrees of freedom and their velocities, the actual and target end-effector positions are also provided as inputs to the policy.

We successfully learned a policy that was able to generalize to new discrete reaching movements. The same policy (trained on discrete movements) can also generalize to a continuous movement task, where the target is dragged interactively. Additionally, we tested the policy's robustness against model error by increasing the stiffness of the elbow joint, making it very difficult to bend. The policy was still able to perform successful reaching movement when possible. See supplementary video for examples.

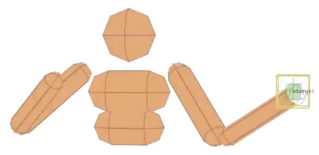

Fig. 3. Example of the arm model during an interactive reaching task

D. Planar Walker

The fourth experiment is a planar legs-and-trunk model with 7 degrees of freedom and 7 controls. The controls in this model are reference joint angles (rather than joint torques), which are then tracked with a PD controller in forward simulation. We found this control space to work better than torque space. Currently, there is a very small amount of root actuation (1e-3 the strength of leg actuators) in our model. Starting stationary, the task is for the trunk to be displaced up to 3 body heights in either direction. Our trajectory optimization is able to find solutions that discover stepping motions and the policy is trained to successfully recreate them. As with the swimmer, the policy neural network in this experiment must learn to capture both discrete movement initiation and termination, as well as cyclic gait maintenance. Figure 4 shows the model and a typical policy trajectory trail. As seen in the supplementary video, the resulting trajectory rollouts change smoothly with task parameters (target trunk location).

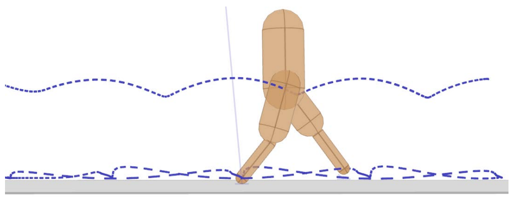

Fig. 4. Example of planar walker model and trajectory trail of a typical policy rollout.

E. Policy Neural Network Details

We used the same neural network for all our experiments except the walker. It consists of an input layer, two hidden layers of 10 units each and an output layer. The walker used three hidden layers with 25 units each. The hidden units use smoothly-rectified transfer functions and the output units are linear. We trained the network with LBFGS, but if the datasets were to become larger, it is possible to use training methods that scale better, such as stochastic gradient descent.

We trained the network on 100 trials, each of which lasted 2 seconds, consisting of 100 timesteps. Thus the training dataset size is 10000 examples.

V. Evaluation

A. Performance

We summarize the expected total cost for the different methods on the four problems we presented. The total cost over a trajectory is evaluated by running the forward dynamics under the different policies (trained with the different methods). The expected total cost is an average over 1000 test trials.

The first method is *Regression*, which trains the neural network on the training set once, using only controls as target outputs. The second method is *Regression+Gains*, which also trains on the dataset once, but includes optimal feedback gains as additional target outputs. The third method, *ADMM*, is the alternating algorithm we described above and also includes optimal feedback gain target outputs. We can see that we get a benefit out of using additional gain data and further improvement from using our ADMM scheme.

	Regression	Regression+Gains	ADMM
Ball	1.19e-2	1.92e-3	1.67e-3
Swimmer	1.03e+0	1.14e-1	7.43e-2
Arm	1.95e-2	9.03e-3	7.02e-3
Walker	9.92e-2	7.54e-3	5.24e-3

We further compare *Regression+Gains* and *ADMM* by looking at the two cost terms in the joint optimization: task and policy reconstruction. Both methods were given the same computation time budget. The results for walker are shown in figure 5. Adding a policy reconstruction term to trajectory optimization can in theory degrade task performance, but that does not happen here (both green curves are almost identical). On the other hand, we see that policy reconstruction greatly improves over ADMM iterations. However, these policy reconstruction improvements do not lead to drastic reduction in overall policy performance as seen in table above. Indeed, *Regression+Gains* is able to learn policies that function well, indicating that we need to explore more challenging tasks.

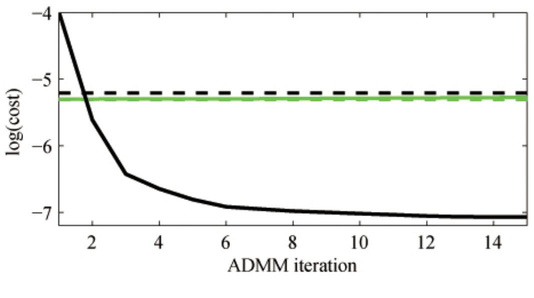

Fig. 5. Two cost terms in 3 plotted against ADMM iteration for walker experiment training. Green curves are task $L(X,U)$ cost and black curves are policy reconstruction cost $R(X,U,W)$. Solid curves are *ADMM* method and dashed curves are *Regression+Gains* method given the same computation time as *ADMM*.

We observed that the algorithm stops making significant changes to the parameters after at most 15 iterations in all our experiments. Below is a table of timing results (in seconds) for various portions of the algorithm along with total times. The hardware used for timing is a 24 core Intel Xeon 2.7GHz workstation. Note that the total time for ADMM is smaller than the time for Regression+Gains multiplied by the number of algorithm iterations. This is because warm-starts in trajectory optimization and network regression greatly reduce the number of iterations required by these methods to reach sufficient optimality.

	Ball	Arm	Swimmer	Walker
Trajectory optimization	51	37	233	316
Network regression (no gains)	56	89	75	121
Network regression (with gains)	73	125	92	143
X_R and U_R minimization	6	8	6	8
Total time for Regression	113	126	315	437
Total time for Regression+Gains	131	170	331	467
Total time for ADMM	680	701	1411	2011

B. Effect of Training Dataset Size

In order to explore how our method performs with much less training data, we revisited the reaching experiment and reduced the number of trials from 100 to 10. Results are shown in the table below. Performance of ordinary regression suffered significantly (by an order of magnitude). On the other hand, performance of our method degraded much more gracefully. This makes our method very promising for data-impoverished settings, which is often the case for complex, high-dimensional problems.

	Regression	Regression+Gains	ADMM
Normal Dataset	1.95e-2	9.03e-3	7.02e-3
Small Dataset	1.79e-1	1.45e-2	1.02e-2

C. Effect of Matching Feedback Gains

We further compared policies trained only on control target, and those trained with additional feedback gain targets, in the ball rolling task. In figure 6 we can see typical ball trajectories generated by policies trained by *Regression* and *Regression+Gains* respectively (results for *ADMM* were visually similar to the latter).

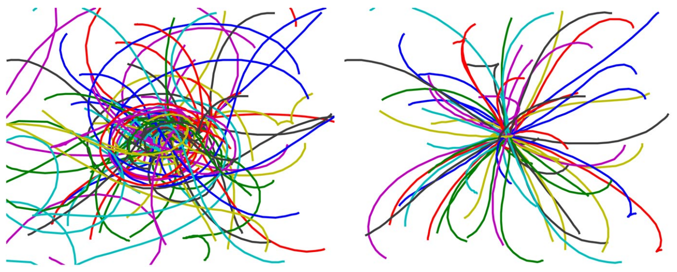

Fig. 6. Ball trajectories generated by training without (left) and with (right) feedback gain targets

We find that even though the first type of network matches the target controls very well, the derivatives of policy $\frac{\partial \pi}{\partial \mathbf{x}}$ behave very irregularly (red plot in 7). On the other hand, the feedback gains $\bar{U}_X$ (dashed green in figure 7) are a much more regular signal that our method matches in additional to the controls.

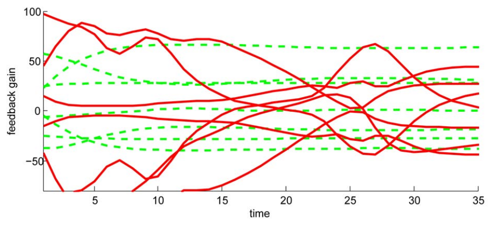

Fig. 7. Optimal feedback gains for a typical ball trajectory (dashed green) and the corresponding $\frac{\partial \pi}{\partial \mathbf{x}}$ produced by a standard neural network (red). Gains produced by our method are visually identical to dashed green curves.

D. *Effect of State and Control Dimensionality*

To investigate more systematically how the performance of our learning method scales with state and control dimensionality, we created an additional experiment where a planar N-link arm (where N is 2, 5 or 10) reached to spatial targets. Training and network size was similar to the other experiments. We observed that performance did not suffer significantly when the problem size increased. The task error averaged over 1000 test targets not seen in training was:

	N=2	N=5	N=10
Task Error	3.44e-2	3.84e-2	4.22e-2

E. *Comparison to Related Work*

As noted earlier, the recent approaches of [13] and concurrent [15] are related to ours. That approach used DPP for trajectory optimization. In problems involving contact DDP needs good initialization, which in [13] was provided via demonstration from a hand-crafted locomotion controller. We use the contact-invariant optimization method specifically designed for problems with contact, and are able to generate policies (albeit for a simpler model at the moment) without relying on prior controllers. Another important difference is the regularity of trajectories produced by the optimizers. In our experience, iLQG/DDP solutions may produce ill-conditioned and noisy trajectories when contact is involved, whereas direct methods tend to produce more regular and smoother trajectories. While this is usually not a problem for achieving the task, it makes training on this trajectory data more difficult.

Another difference is the way two methods modify the trajectory optimization cost. [13] adds a nonlinear term measuring divergence of controls from policy output under the current state. This term requires the policy to provide first and second derivatives with respect to the inputs, which in turn requires differentiable policies. Even if they exist, the policy derivatives may be sensitive quantities and take the trajectory states outside the space the network is trained on. This issue is ameliorated in [13] by averaging over multiple nearby state samples. On the other hand, our ADMM method adds only a quadratic objective on the states and controls and does not require policies to be differentiable or even continuous.

VI. Conclusion and Future Work

In this work we developed a new method for combining the benefits of local trajectory optimization and global policy (neural network) learning. The trajectory optimizer was given an augmented cost, making it find solutions that resemble the current output of the neural network – which in turn makes training the network easier. The method was illustrated on simple examples, focusing on assessment of the relative benefits of the different improvements we made.

The next step is to scale this method to harder and more interesting problems, which we have already been able to solve in the context of trajectory optimization. One exciting application is to recover movement policies that are interpretable. Several hand-crafted locomotion laws have been proposed for human walking and running that match human observations [7]. It would be interesting if our method finds sparse policies that are similar. This can be facilitated by introducing L1 regularization to the policy parameters, which can be easily integrated into our ADMM method (and in fact is one of the original applications of ADMM).

While conceptually policy search methods may be able to find the same policies our method does, the intermediate trajectories generated by policy search methods cannot be outside the policy function class. Our method allows arbitrary trajectories in the intermediate stages of optimization (because they are not generated by the policy, but trajectory optimization instead).

Currently, sampling a set of x^{init} (line 1 in the algorithm) is done once and they are fixed for the duration of the optimization. We would prefer to resample for new values of these quantities as the optimization progresses. It would be interesting to adapt recent work on stochastic ADMM [21] to this problem.

Acknowledgments

We would like to thank Zoran Popović and Dvijotham Krishnamurthy for inspiring technical discussions. This work was supported by the NSF. Igor Mordatch was supported by NSERC fellowship.

References

[1] P. Abbeel, A. Coates, M. Quigley, and A. Ng. An application of reinforcement learning to aerobatic helicopter flight. *NIPS*, 2006.

[2] M. Al Borno, M. de Lasa, and A. Hertzmann. Trajectory optimization for full-body movements with complex contacts. *IEEE Trans Visualization and Computer Graphics*, 2013.

[3] Jos Bento, Nate Derbinsky, Javier Alonso-Mora, and Jonathan S. Yedidia. A message-passing algorithm for multi-agent trajectory planning. In *NIPS*, pages 521–529, 2013.

[4] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1997.

[5] S. Boyd, N. Pariks, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 2011.

[6] L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. *ICASSP*, 2013.

[7] Hartmut Geyer and Hugh Herr. A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(3), 2010.

[8] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. *Proc AIAA*, 2007.

[9] A. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 2008.

[10] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[11] M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 2003.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc of the IEEE*, 1998.

[13] S. Levine and V. Koltun. Variational policy search via trajectory optimization. *NIPS*, 2013.

[14] S. Levine and V. Koltun. Guided policy search. *ICML*, 2013.

[15] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *ICML '14: Proceedings of the 31st International Conference on Machine Learning*, 2014.

[16] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *Internatiional Journal of Robotics Research*, 2012.

[17] W. Miller, R. Sutton, and P. Werbos. *Neural Networks for Control*. MIT Press, 1990.

[18] I. Mordatch, Z. Popovic, and E Todorov. Contact-invariant optimization for hand manipulation. *Symposium on Computer Animation*, 2012.

[19] I. Mordatch, E Todorov, and Z. Popovic. Discovery of complex behaviors through contact-invariant optimization. *SIGGRAPH*, 2012.

[20] I. Mordatch, J. Wang, E. Todorov, and V. Koltun. Animating human lower limbs using contact-invariant optimization. *SIGGRAPH Asia*, 2013.

[21] Hua Ouyang, Niao He, Long Tran, and Alexander G. Gray. Stochastic alternating direction method of multipliers. In *ICML (1)*, volume 28 of *JMLR Proceedings*, pages 80–88. JMLR.org, 2013.

[22] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

[23] M. Posa, C. Cantu, and R. Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *International Journal of Robotics Research*, 2014.

[24] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, pages 833–840, 2011.

[25] Stphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011.

[26] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Transactions of the Royal Society B*, 2003.

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint*, 2014.

[28] J. Si, A. Barto, W. Powell, and D. Wunsch. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, 2004.

[29] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition: Tangent distance and tangent propagation. *Lecture Notes in Computer Science*, 1998.

[30] R. Stengel. *Optimal Control and Estimation*. Dover, New York, 1994.

[31] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000.

[32] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA, 1998.

[33] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *IROS*, 2012.

[34] E. Theodorou, J. Buchli, and S. Schaal. Learning policy improvements with path integrals. *AISTATS*, 2010.

[35] E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. *American Control Conference*, pages 300–306, 2005.

[36] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. *IROS*, 2012.

[37] M. Toussaint. Robot trajectory optimization using approximate inference. *International Conference on Machine Learning*, 26:1049–1056, 2009.

[38] A. Witkin and M. Kass. Spacetime constraints. *SIGGRAPH*, 1988.

[39] M. Zhong, M. abd Johnson, Y. Tassa, T. Erez, and E Todorov. Value function approximation and model-predictive control. *IEEE ADPRL*, 2013.