1 Introduction

Inverse Reinforcement Learning (IRL) has been studied for more than 15 years and is of fundamental importance in robotics. It allows learning a utility function “explaining” the behavior of an agent, and can thus be used for imitation or prediction of a given behavior by having solely access to demonstrated optimal or near optimal solutions. When the reward function is assumed to be a linear combination of features, IRL has strong convergence properties \[AN04, KPRS13, BMZ+15, MHB15, MHB16\]. However, in the recent years, a lot of interest has been focused on using Deep Convolutional Neural Networks (CNNs) to encode the reward function [FLA16, WZWP16]. Using such powerful non-linear function approximators allows to learn from low level features directly, thus not requiring domain knowledge, which can potentially lead to learn higher fidelity behaviors. The LEARning to SearCH framework [RSB09] has introduced functional gradients as a powerful technique underlying the IRL loss optimization. This approach has shown to outperform subgradient methods when optimizing under a linear reward assumption, and was shown to efficiently optimize non-linear cost functions.

In this paper, we extend LEARCH to train CNNs using functional manifold projections, which we denote Deep-LEARCH. Earlier work on functional gradient approaches [RSB09] built large but flat additive models that continually grow in size. Our technique maintains the convergence advantage of functional gradient techniques (observed in linear spaces [MBVH09]) while generalizing to fixed sized deep parametric models (CNNs) by formally representing the function approximator as a non-linear sub-manifold of the space of all functions. We derive a simple step-project functional gradient descent method to walk across the manifold that is substantially more data efficient than traditional gradient steps consisting of a single back-propagation commonly used in Deep-IRL. We present preliminary experimental results showing higher-training rates on low-dimensional 2D synthetic data. We believe these ideas have broad implications for structured training beyond IRL as well as deep learning training in general.

2 Functional manifold projection

Defining \(\xi_i\) as the \(i\)th example trajectory and denoting a state along the trajectory by \(x_i \in \xi\), the LEARCH [RSB09] loss functional is

\[
L[c] = \sum_{i=1}^{N} \left[ \sum_{x_i \in \xi_i} c(x_i) - \min_{\xi \in \Xi} \sum_{x_i \in \xi_i} (c(x_i) - l_i(x_i)) \right],
\]

where \(\Xi\) is the set of all trajectories and \(l_i\) some loss defining the margin. For regularizalization, we often restrict the class of functions \(c\) to lie on the manifold of function approximators \(\mathcal{H}\), such as neural nets with a particular parameterization, and we may even restrict that class further by putting bounds on the size of the neural network weight vectors. In Deep-LEARCH both of these are handled by the
projection step in the inner loop where we train the neural network based on the data we’ve collected. The outline of the LEARCH algorithm that optimizes this loss functional based on its functional gradient is outlined in Section [5].

Intuitively, the negative gradient portion of the loss $\mathcal{L}[c]$ dealing only with the outputs of the cost function $y = c_t(x)$, i.e. $\frac{\partial}{\partial y} \mathcal{L}(y)$ defines the quickest way to decrease the loss function. Thus the (negative) functional gradient data set is just the (negative) loss partials of the loss function

$$\Delta c_t = \left. \frac{\partial}{\partial y} \mathcal{L}(y) \right|_{c_t(x_i)}.$$  

The functional gradient $\nabla f \mathcal{L}[c]$ defines a step off of the function manifold spanned by our hypothesis class $\mathcal{H} = \{ c = c(\cdot; w) | w \in \mathcal{W} \}$ [RSB09]. At each Deep-LEARCH iteration, the functional gradient is projected back ont the manifold by computing the direction $h^* \in \mathcal{H}$ that best correlates with the functional gradient. This results in training the CNN, solving a problem of the form

$$h^* = \arg\max_{h \in \mathcal{H}} < h, \nabla f \mathcal{L}[c] >.$$  

The space of all squared integrable $\mathcal{L}^2$ is big. Even if $\mathcal{H}$ is the space of all deep neural networks with 200 layers and 20 million weights (just a made up example), that space of function approximators can span a submanifold of at most 20 million dimensions, one for each weight. How do we know that? The set of all function approximators in that class is parameterized by its weight vector $w$. Changing the weight vector moves from point to point in the class of functions. The function approximators are differentiable w.r.t. $w$, so this movement between functions is smooth. Therefore, it creates a smooth submanifold in the space of functions. The Jacobian of the function approximator w.r.t. $w$ tells us, to the first order, how the function changes when we change the weights.

3 Results

In order to validate our approach we have implemented Deep-LEARCH with three types of environments. We then study the evolution of the validation loss in [RSB09] on a holdout set with increased numbers of CNN training steps (i.e., stochastic gradient descent steps) that intuitively relate to how precise the manifold projections are.

The environments are presented in Figure [1]. The occupancy maps (first row in the figure) are known and represent the agent sensory input. The three types of data set, from the left to right columns in the figure, are constructed to represent 1) a fully observable occupancy map, 2) lidar occupancy data, and 3) an object in motion, which direction of motion is represented by a few pixels. For each data set,
we randomly generated 800 environments composed of occupancy maps and synthetic costmaps on which we planned 20 example trajectories using the Field D* \cite{FS06} algorithm, resulting in 16,000 demonstrations. We then attempt to learn a CNN parameterization that can reproduce the behaviors using Deep-LEARCH. The CNN maps occupancy to cost (as shown in the third row of the figure), producing cost for each pixel in the image. The predicted behavior is obtained by path planning using the same Field D* algorithm. We show on the first data set that training the network with a larger number of stochastic gradient steps, in black in the figure, allows higher generalization performance, which makes our functional manifold projection view of Deep-IRL promising.

4 CNN models

We use a Convolutional Neural Network (CNN) as the function approximator to generate the cost function for a given scene. We have tested three specific architectures, all of which take an occupancy grid representation $\text{Occ}(\mathcal{X})$ of a scene $\mathcal{X}$ as the input and generate the corresponding cost function $c$:

- **Conv-Only**: This network applies a sequence of six convolutions, followed by Batch Normalization (BN) \cite{IS15} and PReLU \cite{HZRS15} non-linearities. We keep the resolution constant throughout the network.

- **Conv-Deconv**: This network first applies a sequence of four convolutions to the input. Each convolution layer is followed by a max-pooling operation that reduces resolution by a factor of 2. At the bottleneck, we apply a 1x1 convolution followed by four deconvolution layers which interpolate the output back to the full resolution. Each layer except the final layer is followed by a Batch Normalization and PReLU non-linearity.

- **Conv-Deconv-Linear**: This network architecture is similar to the Conv-Deconv network, but we replace the Conv + max-pooling operations by strided convolutions. We only have 3 strides convolution and deconvolutional layers. Also, we replace the 1x1 convolution at the bottleneck with two fully connected layers that allow us to propagate information throughout the entire image. As before, all layers except the final layer are followed by a Batch Normalization and PReLU non-linearity. We made use of this model in the results presented in Section 3.

We use the neural network package Torch \cite{tor} to implement our models. We use the ADAM optimization algorithm \cite{KB14} with default parameters, a step size of 1e-3 and a batch size of 32 for training all our networks.

5 Appendix: LEARCH algorithm

1. Initialize the data set to empty $\mathcal{D} = \emptyset$, and iterate the following across all examples $i$:
   
   (a) Solve the loss-augmented problem
   \[
   \xi_i^* = \underset{\xi \in \Xi}{\text{argmin}} \sum_{x_i \in \xi} (c(x_i) - l_i(x_i)).
   \] (2)

   (b) Add the functional gradient data from the loss-augmented problem
   \[
   \mathcal{D} = \mathcal{D} \cup \{ (x_i^*, c(x_i^*) + \eta_i) \mid x_i^* \in \xi_i^* \}. \] (3)

   These points suggest where to increase the cost function by $\eta_i$.

   (c) Add the functional gradient data from the example
   \[
   \mathcal{D} = \mathcal{D} \cup \{ (x_i^*, c(x_i^*) - \eta_i) \mid x_i \in \xi_i \}. \] (4)

   These points suggest where to decrease the cost function by $\eta_i$.

2. Solve the regression problem to improve the hypothesis:
   \[
   c_{t+1} = \underset{w}{\text{argmin}} \frac{1}{2} \sum_{(x,y) \in \mathcal{D}} (y - c(x; w))^2 + \frac{\lambda_1}{2} ||w - w_t||^2 + \frac{\lambda_2}{2} w. \] (5)
6 Acknowledgment

This research was supported in part by National Science Foundation grants IIS-1205249, IIS-1017134, EECS-0926052, the Office of Naval Research, the Okawa Foundation, and the Max-Planck-Society. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding organizations.

References


