Gradient-Free Supervised and Unsupervised Learning with Rewards

Sayna Ebrahimi
UC Berkeley
Berkeley, CA, 94720
sayna@eecs.berkeley.edu

Anna Rohrbach
UC Berkeley
Berkeley, CA, 94720
anna.rohrbach@berkeley.edu

Trevor Darrell
UC Berkeley
Berkeley, CA, 94720
trevor@eecs.berkeley.edu

Abstract

We develop a method for policy architecture search and adaptation via gradient-free optimization which can learn to perform autonomous driving tasks. By learning from both demonstration and environmental reward we develop a model that can learn with relatively few early catastrophic failures. We first learn an architecture of appropriate complexity to perceive aspects of world state relevant to the expert demonstration, and then mitigate the effect of domain-shift during deployment by adapting a policy demonstrated in a source domain to rewards obtained in a target environment. We show that our approach allows safer learning than baseline methods, offering a reduced cumulative crash metric over the agent's lifetime as it learns to drive in a realistic simulated environment.

1. Introduction

Deep architectures have become popular as function approximators to represent action-selection policies. Common approaches to learn the parameters of such models include reinforcement learning and/or learning from demonstration: both learn model parameters to maximize expected reward, mimic human behavior, and/or achieve implicit goals. However, the design of policy architectures, especially in a deep learning paradigm, remains relatively unexplored. Architectures are typically selected through a combination of intuition and/or trial and error.

Learning to learn, including the learning of learning architectures, is a long-articulated goal of AI, and many meta-learning and lifelong learning schemes have been proposed (e.g., [5] offered seminal views; see [4] for a survey).

We investigate policy architecture search using gradient-free optimization and learn optimal policy structure for autonomous driving tasks. We propose a model which learns jointly from demonstration and optimization, with the goal of safe training: minimizing the amount of damage a vehicle incurs to learn a threshold level of performance. We base our approach on exploration-based schemes due to their ability to optimize model weights and architecture hyperparameters, leverage expert demonstrations, and adapt to reward obtained in new domains. We believe that a model which can initialize from demonstration, and learn an optimal policy from that foundation, is likely to achieve higher performance while maintaining the constraint of safe training, compared to models which must randomly search through action space during initial learning, or which learn from a reasonably safe demonstration but cannot further optimize performance based on environmental reward.

Often, deep models which learn to perform in one domain fail to perform well when deployed in another setting, such as differing weather or lighting conditions. We show that our method can effectively and safely adapt a model demonstrated in one environment but deployed in a visually different environment based on the reward signal in the latter domain, even when the agent is initialized far from initial demonstrations. Our approach leverages only target domain reward, and makes no assumptions about domain alignment, explicit or implicit, nor assumes any demonstration supervision in the target domain.

To achieve these goals, we present a gradient-free optimization algorithm inspired by [3] with a modification in noise generation that results in estimating the gradients more efficiently and accurately (Sec. 2.1). We then apply this algorithm to search over variable length architectures Next, we combine our gradient-free policy search with demonstrations to learn a better policy that adapts to the new environment by receiving rewards as feedback (Sec. 2.3). We experimentally show that our architecture search model finds a policy on the GTA game environment that outperforms previously published methods (e.g., [1]) in end-to-end steering prediction from demonstrations, and that it can be efficiently adapted to learn to drive in previously unseen scenarios (Sec. 3). Our model reduces the number of crashes incurred while learning to drive, compared to baselines based only on reward or demonstration but not both, or compared to previously proposed fixed architectures that were not optimized for the domain.

2. Approach

We propose a learning-to-learn model which includes architecture optimization, parameter learning, and representation adaptation over different time scales. Our approach can be summarized by the following two steps. (1) Given expert demonstration, search over architectures and parameters to find a policy that best mimics performance by monitoring the obtained accuracy and number of parameters. (2) Having learned from demonstration, adapt the model to the reward provided by the target environment. In both steps, it is essential to derive a function approximator that optimizes an objective function. We use a gradient-free optimization algorithm [3] that maximizes a parametrized reward function using gradient estimation to perform architecture search (Sec. 2.2) and policy learning (Sec. 2.3).

2.1. Gradient-free optimization algorithm

Let $F(\cdot)$ be our objective function parametrized by θ which is an n-dimensional vector. F can be the reward that an environment provides for an agent when it executes a policy with parameters θ; our goal is to maximize the expected reward by perturbing the policy parameters, denoted as $\hat{\theta}$, by moving in particular directions. The parameter estimate update can be performed using a general stochastic form:

$$\hat{\theta}_{t+1} = \hat{\theta}_t + \alpha_t \, \nabla_{\hat{\theta}} y(\hat{\theta}) \tag{1}$$

where $y(\cdot)$ is an approximation of the objective function (i.e. $y(\cdot) = F(\cdot) + \text{noise}$) and $\nabla_{\hat{\theta}} y(\hat{\theta})$ is the gradient of objective estimate that can be approximated by any gradient estimator in the family of finite difference methods. The gradient is estimated in a randomly chosen direction by perturbing all the elements of $\hat{\theta}_t$ to obtain two measurements of $y(\cdot)$ as follows:

$$y_t^{(+)} = F(\hat{\theta}_t + c_t \Delta_t) + \epsilon_t^{(+)} \tag{2}$$

$$y_t^{(-)} = F(\hat{\theta}_t - c_t \Delta_t) + \epsilon_t^{(-)} \tag{3}$$

where $\Delta_t \in R^n$ is a vector of n mutually independent randomly perturbed variables taken from a zero-mean distribution. While there is no restriction for it to have a specific type of distribution, we use Laplace distribution, as it tends to choose orthogonal directions in the long run. Other recent efforts [6] utilized Gaussian noise to sample mirrored projections. c_t is a small positive number and $\epsilon_t^{(+)}$ and $\epsilon_t^{(-)}$ are the noise associated with evaluating $F(\cdot)$ such that: $E\big(\epsilon_t^{(+)} - \epsilon_t^{(-)} | \{\theta_1, \theta_2, \cdots \theta_t\}, \Delta_t\big) = 0$. The gradient estimate can then be computed as:

$$\nabla_{\hat{\theta}_t} y(\hat{\theta}_t) = \left[\frac{y_t^{(+)} - y_t^{(-)}}{2 \; c_t \; \Delta_{t1}} \quad \cdots \quad \frac{y_t^{(+)} - y_t^{(-)}}{2 \; c_t \; \Delta_{tn}} \right]^T \tag{4}$$

Parameter estimates can be updated by replacing the gradients in Eq. 1 with those found in Eq. 4.

2.2. Learning an optimal initial policy from demos

Inspired by [6] we have used a recurrent neural network to sequentially generate the description of layers of an architecture from a given design space defined by the user. The RNN acts as a controller which generates the architecture description defined by its hyper-parameters chosen from a pre-defined search space. Our model uses the demonstrations to provide the reward function, ((i.e., F above), to train the RNN. Unlike backpropagation which suffers from gradient vanishing while training RNNs, gradient-free algorithms do not have such an issue [3]. Our RNN controller specifies three types of layers: convolutional, fully connected, and max-pool which can have inter-layer dropouts. For the reward signal, we use the negative value of *total* loss function. At the last layer of the network, we regress to three real-valued numbers, each having a mean-squared loss. The total loss is the sum of all three losses. We use a novel reward function (Eq. 5) that not only results in the minimum total loss but also grows the architecture as long as the loss keeps decreasing.

$$R = R_1 - \lambda \left(R_1 \right) R_2 \tag{5}$$

where R_1 is the negative of the minimum loss (or maximum accuracy in a classification problem) on the validation set for the last 5 epochs and R_2 is the total number of parameters in the child network. λ is the Lagrange parameter defined as a function of the first sub-reward in a ReLU-based fashion.

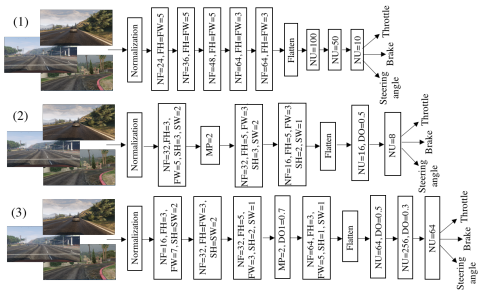
2.3. Adapting policy to a new domain

As confirmed experimentally below, it is well known that a policy learned from behavioral cloning can perform poorly when evaluated on inputs with a domain shift relative to the demonstration supervision. To overcome this, we further use the gradient-free search algorithm described in Sec. 2.1 to adapt a driving policy learned from demonstration in a source domain based on rewards in a target domain. We experiment with the setting where the driving domain attributes are substantially different than provided as demonstration.

In the driving scenario, we wish to learn to drive with optimal or near-optimal performance, using the reward function in target domain which is composed of two factors (we receive $+1$ if obeyed and 0 if violated): 1) No crashes with other objects 2) Staying within the lane lines if they are available in the driving scene. We have used the lane reward function and accident detection function defined in [2] source code as a *paths.xml* file.

3. Experimental evaluation

All the experiments are executed in the GTA game environment using a publicly available plugin [2]. We have



(a)

Figure 1. (a) Illustration of the baseline and learned architectures: (1) prior work [1], (2) our small network, (3) our large network

collected a dataset of an expert policy by playing GTA collecting $2,267,662$ images of size 66×200 similar to [1].

3.1. Learning policy architecture from GTA demos

We present our architectures and comparison to prior work in Fig. 1(a)(2). The corresponding performance comparison is shown in Tab. 1. The smallest architecture (in terms of the number of parameters) that we have built is shown in Fig. 1(a), which outperforms the baseline [1] with a smaller total loss. By not restricting our architecture search algorithm to be bounded by a number of parameters, we learn a larger network (1(a)(3)) that has over $2M$ parameters and obtains a minimum total loss of 0.085 on the training set and 0.088 on the validation set. We use this network as our initial driving policy in the next section and improve it further in the reward-providing GTA environment.

Table 1. Total MSE obtained using architecture proposed in prior work [1] and our models obtained by architecture search on demonstrations.

Model	# of params	Train loss	Val. loss	Test loss
Bojarski et al. [1]	252,241	0.098	0.11	0.212
Our small arch	228,227	0.093	0.096	0.197
Our large arch	2,198,723	0.085	0.088	0.185

3.2. Safe policy adaptation

Here we show the results for adapting the learned policy on demonstrations to the target domain using rewards. We use our large network and the baseline architecture [1] in Table 2. We start with an initial model, either using a behaviorally cloned or randomly initialized policy and gradually improve it by receiving rewards from the environment. We evaluate both models with and without being adapted to demonstration forming four cases: (1) the baseline network of without demonstration (i.e., with randomly initialized weights) and (2) with behaviorally cloned initial weights, (3) our larger architecture without demonstration and (4) with behaviorally cloned initial weights. Results averaged across several runs are presented in Tab. 2 where

Table 2. Comparison of two policies (our large network and [1]) learned based on target domain reward, with and without source-domain demonstrations.

Model	Wall-clock convergence	Total # of crashes	Total # of middle-lane keeping violations
(1) Bojarski et al. [1] w.o demo	154 hrs	15,565	18,662
(2) Bojarski et al. [1] w. demo	74 hrs	1,387	3,243
(3) Our large arch, w.o. demo	114 hrs	6,877	8781
(4) Our large arch w. demo	53 hrs	832	982

Table 3. Comparison of performance (average total loss) of two policies (our large network and [1]) at test time on target domain (T) when they are trained with rewards from target domain, source domain (S), and both (T+S).

	Behavioral cloning	Demo on S Rew on T+S Test on T	Demo on S Rew on S Test on T	Demo on S Rew on T Test on T
Baseline	0.212	1e-4	-	-
Our large arch.	0.185	1e-5	7e-5	3e-5

our model optimized with demonstration outperforms all other cases. In particular, our model has the least number of cumulative crash occurrence prior to converging to 100% of averaged reward. Supplementary video of results can be found in `https://saynaebrahimi.github.io/corl.html`

4. Conclusion

The goal of this work is to learn a policy for an autonomous driving task minimizing crashes and other safety violations while training. To this end we propose an algorithm which learns to generate an optimal network architecture from demonstration using a new reward function that optimizes accuracy and model size simultaneously. We show that our method can adapt the model learned by demonstration to a new domain relying on target environmental rewards. Experimental evaluation shows that our model achieves higher accuracy, fewer cumulative crashes, and higher target domain reward.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] A. Ruano. Deepgtav. `https://github.com/ai-tor/DeepGTAV`, 2017.

[3] T. Salimans, J. Ho, X. Chen, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[4] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[5] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[6] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.