Learning Rewards for Videos

Jonathan Booher
jaustinb@stanford.edu

Abstract

Sample efficient Reinforcement Learning requires meaningful reward functions to encourage desired behaviors. Often in robotics and complex manipulation, dense reward functions are difficult or impossible to create by hand. We propose learning a consistent reward function from large scale dataset of videos of expert demonstrations that are highly multi-modal and potentially sub-optimal. Learning such a reward function would admit the training of a RL agent directly from the learned reward solving an Inverse Reinforcement Learning problem without training RL agents within an optimization loop over reward functions.

1. Introduction

Deep reinforcement learning (RL) is commonly used in robotics to produce policies capable of accomplishing various tasks. Often, it is difficult or impossible to hand engineer reward functions that that the RL agent can learn to maximize. To address this issue, Imitation Learning seeks to leverage demonstrations from experts (like humans) to help guide the agent toward a good policy. Many of these algorithms use approaches like simply regressing expert actions (behavioral cloning) or learn to discriminate between expert actions and policy actions (generative adversarial imitation learning [1]). We seek, however, to learn a reward function from demonstrations that would allow for successful training of an RL agent to perform a chosen task.

2. Problem Statement

Specifically, we address the problem of learning a reward function from videos that accurately corresponds to progress toward accomplishing the task. This reward function should ideally be smooth to allow the agent a clear signal for learning. The reward function that we will learn takes as input two frames, the current frame (that corresponds to the current state of the environment) and a target frame (which corresponds to a target configuration of the environment) and outputs a single-dimensional real valued output (the reward).

3. Technical Approach and Background

3.1. Time Contrastive Networks

Time Contrastive Networks (TCNs) [5] a recently proposed method of learning reward functions from videos as well as aligning disparate videos from different angles and subjects. In order to learn a reward function from videos, TCNs learn a metric embedding of the images so that distances between a frame and a target frame can be interpreted as "progress" toward the target frame or as a measure of similarity. With the metric embedding, a reward function is straightforward

$$R(x, \text{target}) = \alpha||f(x) - f(\text{target})||_2 + \beta||f(x) - f(\text{target})||_1$$

where x is the current frame, α, β are hyperparameters, and f is the embedding network. This loss can be interpreted as a Huber loss on the distance between embeddings, but any loss function could be used on the distances between embeddings.

3.2. Triplet Networks

One of the more common approaches to learning metric embeddings of images is the triplet network [2]. The triplet network takes as input three images, an "anchor", and two other images, one from the same class as the anchor (a "positive" example) and the other from a different class (a "negative" example). Each of the images are passed through the embedding network and the loss function is designed to minimize the distance between the anchor and "positive" target while maximizing the distance between the anchor and "negative" target image. concretely, the triplet loss is

$$L = \max\Big(d\big(f(x), f(x_p)\big) - d\big(f(x), f(x_n)\big) + \Delta, 0\Big)$$

where x, x_p, x_n are the anchor, positive example, and negative examples respectively, f is an embedding network, d is an arbitrary distance metric, and Δ is the margin, in this work, d is chosen to the L2 distance. This loss is similar to that of the SVM loss. In the case of sequences of images, we can define the "positive" examples as images that are sampled from within a small window around the anchor image, and "negative" examples as those that are temporally distant

from the anchor. This creates the need for an additional two hyperparameters λ_1, λ_2 respectively where $\lambda_2 > \lambda_1$ are the sampling windows.

3.3. Semi-Hard Negative Mining

The use of triplet networks necessitates the selection of triplets from which to train. To make training as quickly as possible, it would be ideal to chose triplets such that

$$x_p = \text{argmax}||f(x)-f(x_p)||_2^2, \quad x_n = \text{argmin}||f(x)-f(x_n)||_2^2$$

where this allow us to calculate the ideal x_p, x_n for a given anchor image. However, this is likely infeasible for larger datasets and the argmax/min is expensive to compute in larger batches. Thus we seek a reasonable approximation of this ideal. Additionally, choosing examples that violate the margin can sometimes lead to local rather than global optimum. Rather than perform expensive operations, we sample triplets for which the following inequality holds

$$||f(x) - f(x_p)||_2^2 < ||f(x) - f(x_n)||_2^2$$

these are called semi-hard negatives as they are still within the margin, but are still hard as the negative is closer to the anchor than the positive example [4]. It is possible to efficiently calculate these triplets and they share the benefits of speeding up the training of triplet networks.

3.4. Our Approach

We train a TCN using the videos of the collected dataset by using semi-hard negative mining. However, the collected data is extremely multi-modal which poses a problem for learning a global reward function (one that accurately tracks task completion). The reward function proposed above relies on the presence of a goal frame which is not directly apparent for unstructured tasks like that one being examined in this work. To address this issue, we propose a further loss that encourages the start and end frames of an episode to be close together in the embedding space ie.

$$L_{end} = ||x_1 - x_2||_2^2$$

where x_1, x_2 are the start/end frames of a single trajectory. By encouraging the embedding of all terminal frames to be close together, we can directly interpret the loss function as progress toward *task completion* rather than progress toward achieving a given goal. This loss can also be applied inter trajectory by encouraging frames at the end of each window of length λ_2 to be close to the first frame of the next window.

3.5. Baselines

We will compare the above approach to several baselines. One set of baselines will be based on features extracted from the images by a ResNet that was pretrained on ImageNet, the baselines would then utilize cosine distance and l2 distance on these features. By using the pretrained ResNet, the features extracted should be meaningful to the scene. An additional baseline will be MSE on the pixels of the two images. All of these baselines provide a distance metric with respect to the input images that will be used in the same reward function (ie the same values of α and β) as the proposed method.

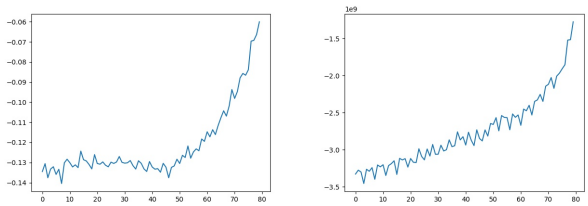

Figure 1. Examples of the learned reward function when using cosine distance and l2 distance on raw pixels. This notion of reward is also brittle to changes in lighting conditions as changes in shadows would lead to large differences in pixel distances (left cosine distance, right l2).

4. Dataset

The dataset that used in this work was collected using an extension of the RoboTurk platform [3] that applied the same method of teleoperation via iPhones to accomplishing tasks on real Sawyer robot arms. Specifically the task that is being examined is called *Laundry Layout* which is an unstructured task in which a cloth is to be unfolded by the user as efficiently as possible. This task inherently multi modal and difficult to replicate in simulation due to the deformable nature of the object being manipulated. The RoboTurk platform collects various different streams of data; however, in this work we limit ourselves to using sequences of images from the view that users use to accomplish the task. The collected data consists of over 300 trajectories from 15 different users and the camera angle and lighting is not consistent across trajectories.

5. Preliminary Results

5.1. Baseline experiments

We evaluate the different baselines and presenting graphs of the reward functions over periods of ≈3 seconds. Fig 1 shows results for reward functions using distances on raw pixels and Fig 2 shows results when using a pretrained encoder. It is clear that these approaches lead to noisy reward functions that would not be useful in an RL setting.

5.2. Proposed Method

We train the TCN model on the videos to show first that it is possible to learn reward functions from them without

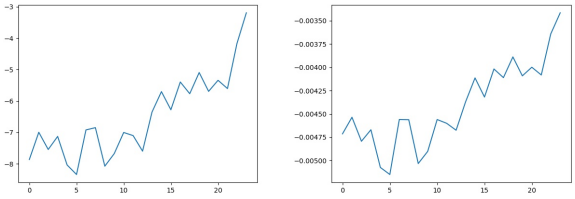

Figure 2. Examples of the learned reward function when using cosine distance and l2 distance on embeddings from a pretrained ResNet-18 on ImageNet. This reward is non-ideal as many of the classes in the scene are not in ImageNet (sawyer robot) and much of the scene remains constant. (left l2, right cosine)

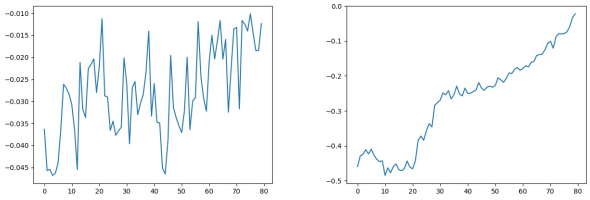

Figure 3. The graphs of the learnt reward function for two different segments of the same demonstration that highlight the difficulty of learning smooth and continuous reward functions.

the additional proposed constraints. Specifically, we train a TCN with the parameters $\lambda_1 = 20$ $\lambda_2 = 40$ and evaluate the reward with $\alpha = \beta = 1$ on video recorded at 30hz. Graphs (Fig 3) of the results over small horizons (≈3 seconds) show that the reward is generally increasing, but can contain significant levels of noise in the reward signal. It is also clear that there exists a difference in scale between different segments of the reward curve which could pose issues in RL without the usage of a staged reward function.

5.3. Analysis of Methods

To provide an analysis of the efficacy of these approaches, we run several different analyses to examine the smoothness of the learned reward functions. One potential method for evaluating the smoothness of these rewards is through a Fourier analysis. The learned rewards should smoothly vary over time which means there should be an absence of high frequency waves after the Fourier transform. Fig 4 shows examples of the results of such an analysis for the proposed method.

A quantitative analysis of the reward functions can be accomplished by using a metric based on the standard deviations of sequential differences in the sequence of rewards. This results in the following calculation:

$$d = \texttt{diff}(r), \quad s = \frac{\texttt{std}(d)}{\texttt{mean}(d)}$$

The differences in the sequential values of r (the reward) correspond to "jumps" in the function. Taking the standard deviation would then give a metric for the spread of these change over time, or how smooth the function is. Lower values, obviously, correspond to smoother functions. We report the average of this metric across the entire validation dataset.

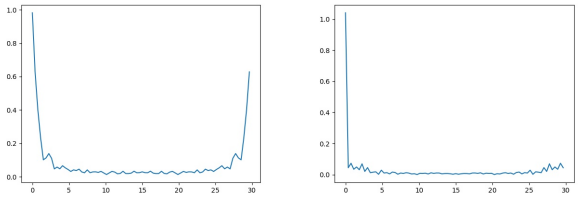

Figure 4. FFT decomposition of two pieces of the same trajectory. Left shows one with noisy reward signal exemplified by the large magnitude of a high frequency wave. Right shows a smoother curve that does not have comparably large magnitude at the high frequencies.

Method	Mean Smoothness	Std. Smoothness
Ours	5.922	33.56
Pixel l2	3.741	7.834
Pixel cosine	2.116	1.506
ImageNet l2	52.09	757.388
ImageNet cosine	157.38	2625.9

Table 1. Mean and std of the calculated smoothness metric across the dataset on the different methods.

While the results of the smoothness metric suggest that baseline methods are capable of providing a meaningful reward function, the smoothness metric currently used would be 0 when the provided sequence defines a line. We do not expect that the demonstrations provide a trajectory that should result in linear improvement in the reward. This is because the demonstrations themselves are noisy, but also the different strategies employed by users could be suboptimal in their progress toward the goal It is also worth noting that the standard deviation and mean of smoothness on the proposed method are skewed upward by some trajectories that exhibit large fluctuations like the segment in Fig. 3. More work needs to be done to reduce this variance of the proposed method and define a better metric of the quality of the reward functions.

References

[1] J. Ho and S. Ermon. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc., 2016. 1

[2] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *ICLR*, 2015. 1

[3] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and

L. Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018. 2

[4] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. 2

[5] P. Sermanet, C. Lynch, J. Hsu, and S. Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *CoRR*, abs/1704.06888, 2017. 1