Towards Decomposable Rewards through Generative Adversarial Inverse Reinforcement Learning

Peter Henderson
Department of Computer Science
McGill University
Montreal, QC H2X 3P8
`peter.henderson@mail.mcgill.ca`

Wei-Di Chang
Department of Electrical, Computer, and Software Engineering
McGill University
Montreal, QC H2X 3P8
`wei-di.chang@mail.mcgill.ca`

Abstract

Generative adversarial imitation learning examines the cost of learning a reward from state-action pairs. We examine the generative adversarial reinforcement learning framework for the task of indirect inverse reinforcement learning where only states of the expert agent are known. Additionally, from the idea of options in policy optimization as an intuitive way to break down multi-step tasks in a complex environment, we parallel this notion in the space of inverse reinforcement learning and introduce the notion of decomposable rewards or reward options. Using the framework of generative adversarial inverse reinforcement learning, we demonstrate that decomposable rewards perform as well as a non-decomposed reward approach in simple tasks. We further characterize some samples of the decomposed rewards and discuss possible stabilization methods which we discovered through qualitative examination of our methods.

1 Introduction

In the Reinforcement Learning setting, an agent attempts to solve a task through the maximization of a specific reward function. In recent years this framework has been successfully used to solve tasks in large and complex environments. Using reinforcement learning, agents have learned to play Atari games from raw pixels [1], beat human players at the game of Go [2], and driving cars [3]. However, specifying an efficient and optimal reward function can be difficult. This often requires manual manipulation and hand-coded features designed through trail-and-error. Furthermore, a significant amount of tasks outside of simulation can be hard to characterize well using a single reward function as they require multiple distinct parts.

To circumvent having to handcraft a reward function, the field of Inverse Reinforcement Learning (IRL) uses expert demonstrations of a task which the agent attempts to reproduce. IRL can be performed through direct learning of the policy, or indirectly by inference of the reward function the expert is maximizing.

The direct IRL framework thus learns a mapping from state-space features to actions through the expert's state-action trajectories, while indirect IRL first recovers a reward function using the expert state trajectories, and subsequently solves the task with the inferred reward structure.

We investigate the use of Generative Adversarial methods based on past work [4] for the task of indirect IRL. We further formulate a decomposable rewards framework in the context of Inverse Reinforcement Learning using Generative Adversarial learning, and benchmark it on small OpenAI Gym tasks to show that it works as well as Generative Adversarial Inverse Reinforcement Learning.

2 Background

2.1 Inverse Reinforcement Learning

Rooted in the control theory and initially formulated as "Inverse Optimal Control" in [5], Inverse Reinforcement Learning was formulated in the context of Markov Decision Processes (MDPs) in 2000 [6]. Different approaches to solving the indirect Inverse Reinforcement Learning problem have been formulated. Abbeel and Ng's approach [3] learns a parametrization of the reward function as a linear combination of the state feature expectation so that the hyperdistance between the expert and the novice's feature expectation is minimized. Ziebart in [7] formulates a solution using the maximum entropy principle, with the goal of matching feature expectation as well.
Recent advances in IRL make use of deep neural networks to learn the cost function without the need to handcraft features [8], Generative Adversarial Models to perform direct IRL [4], or domain confusion techniques combined with Generative Adversarial models to perform policy transfer learning between similar domains [9].

2.2 Mixture of experts

The idea of creating a mixture of experts (MoEs) was initially formalized to improve learning of neural networks by dividing the input space among several networks and then combining their outputs for the final result [10]. It has since come into prevalence for generating extremely large neural networks, as in [11]. This later work focuses on making MoEs a component which can be used throughout a larger neural network architecture. On a smaller scale, MoEs are beneficial when a state space can be divided into distinct processes. Generally, an overview of MoE models and training methods can be found in [12].

3 Generative Adversarial Inverse Reinforcement Learning

We formulate our IRL solution in a similar way to Generative Adversarial Imitation Learning [4]. We use a discriminator to learn a binary classification task to differentiate sample rollouts from our novice policy from expert rollouts (demonstrations). This label is then used as the reward for a policy optimization step (in this case, as in [4, 9], we use Trust Region Policy Optimization [13]). See Figure 1 for a general diagram of how this process flows. Generally, we perform sample rollouts using the current policy, perform a discriminator update step, and then proceed to a TRPO policy optimization step. Similarly to [9], we can formulate the problem as a generalized form of Generative Adversarial Imitation Learning such that we optimize the objective:

$$\max_{\pi_\theta} \min_{D_R} -\mathbb{E}_{\pi_\theta}[\log D_R(s)] - \mathbb{E}_{\pi_E}[\log(1 - D_R(s))] \tag{1}$$

Where π_θ and π_E are the policy of the novice and expert, respectively, and D_R is the discriminator or an approximator for the probability that a sample s belongs to the expert. Note that unlike [4], we do not assume knowledge of the expert actions. Our formulation is closer to [9] in nature which relies solely on observations in the discriminator problem. We therefore refer to our approach as Generative Adversarial Inverse Reinforcement Learning (GAIRL) as opposed to imitation learning.

3.1 Discriminator Architecture

For our discriminator, we concatenate a vector of n consecutive states (timesteps) and feed the output through two convolutional layers (5 filters each, kernel size 3, a stride of 1, with padding to match the previous layer) followed by a fully connected network with a single hidden layer (128 hidden units). We use a sigmoid activation for the final prediction and a sigmoid cross-entropy loss function, minimized using the Adam optimizer [14] with an initial learning rate of 0.001.

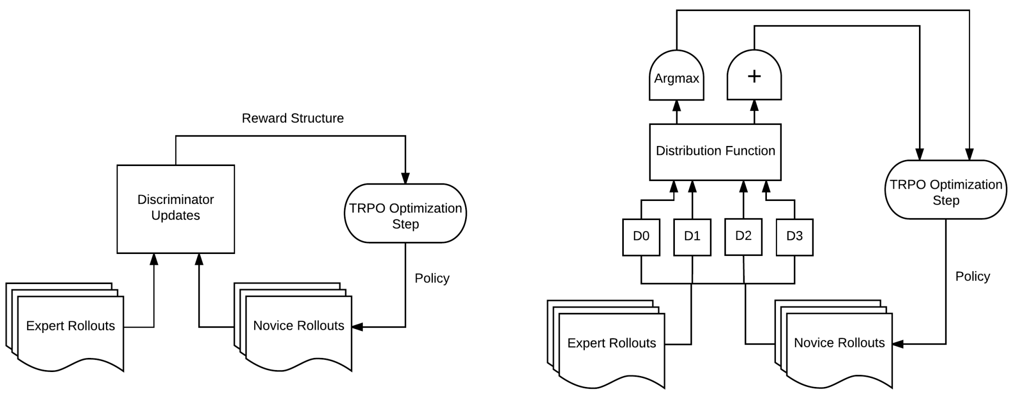

Figure 1: Generative Adversarial Inverse Reinforcement Learning Architecture

3.2 Policy and Generator

Similarly to [9], we use a two layers, fully connected neural network with 100 nodes per layer and tanh nonlinearities to represent our policy. We use Trust Region Policy Optimization (TRPO) [13] to improve the network by performing rollouts while using the reward function output by our discriminator.
While we chose TRPO for its robustness in the optimization of neural network based policies as well as its guarantee of monotonic improvements [13], desirable here for the stability of the discriminator network, any other existing policy representation and optimizer could have been used in its place.

3.3 Stabilization

In its standard form, our generative adversarial approach exhibits high variance as well as training instability. Several approaches have been experimented with in order to ensure convergence as well as stabilize the training of the discriminator.

3.3.1 Random Noise Injection

As performed in [15], we add noise to the input of the discriminator network, which has a regularizing effect. In our context this noise takes the form of trajectories obtained by rollouts performed by an agent with a random policy.

3.3.2 Class Weighting and Oversampling

As commonly done for classifiers, to handle our data class imbalance (less expert demonstrations than novice rollouts usually), we weight the loss incurred by a misclassification using the proportion of the full batch the misclassified class belongs to.
Additionally, another approach is oversampling which reuses existing data to fill in the lesser represented class.

3.3.3 Update Rule

While trying to stabilize the GAN framework for IRL, we find that the update rule is critical to prevent divergence and a timely climb to the optimal true reward. That is, a single update step of TRPO and a single Adam step for the discriminator was not enough for consistent convergence to a solution. We found that the best performance came by allowing TRPO to follow a gradient for a longer number of timesteps before performing a similar number of discriminator updates. While we choose the number of update steps qualitatively through experimentation (setting the value to 3 epochs of discriminator updates to 6 TRPO updates), we can gain a sense for why this is necessary and formalize this as follows. Let us first first begin by examining the TRPO optimization problem:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \mathbb{E}_{s\sim p_{\theta_{old}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{old}}(s,a)\right] \\ \text{subject to} \quad & \mathbb{E}_{s\sim p_{\theta_{old}}}\left[D_{\text{KL}}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdots|s))\right] \leq \delta \end{aligned} \tag{2}$$

Note here that $Q_{\theta_{old}}$ can be defined as:

$$Q_{\theta_{old}}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right] \tag{3}$$

In our adversarial game, r becomes a moving target. With every discriminator update the distribution of rewards shifts. For TRPO with the conjugate gradient estimation method to work successfully, the gradients need to follow a consistent direction, with a shifting target there may be too much noise to have a stable optimization problem. Let's say we denote the distribution of rewards for a given set of sample rollouts as $P_R(S)$. We can say that a TRPO update step will shift this distribution to $P_R(\hat{s})$. That is, the sample distribution will shift but the output of the reward function will not change. Similarly, for a discriminator update step, we call the new distribution $\hat{P}_R(S)$. In both cases, the distribution changes, but the cause of the distribution shift is different.

Indeed it is intuitive that if we would let the discriminator network overfit on the most recent rollout dataset batch, on the next iteration when provided with slightly different rollouts generated in the policy optimization step, the large difference in the presented data will lead to instability in training. Conversely we can apply the same principle to the policy learning step being presented with one reward structure, optimizing its policy with respect to this reward for multiple steps, and being presented with a drastically different reward structure in the next iteration. In both cases making too many updates without updating the adversary leads to instability.

However the magnitude of the updates isn't necessarily constant throughout the training process. Therefore the number of updates to carry out for each component (discriminator and policy) needs to be varied by observing the amount of progress the adversary is making in its update.

Using our earlier notation, we can loosely say that we want to continue TRPO updates while $D_{\text{KL}}(P_R(S)||\hat{P}_R(S)) \leq \xi$ and then continue discriminator updates until $D_{\text{KL}}(P_R(S)||P_R(\hat{S})) \leq \xi'$. We can define ξ, ξ' to be some constant (as we essentially do in our current implementation). More formally, however, we can say that the predicted distribution shift (from, say, previous sampling) due to a TRPO step can be defined as $P_R(\tilde{S})$. The distribution shift due to a discriminator update can be defined as $\tilde{P}_R(S)$. Thus we can replace ξ with $\hat{\xi} D_{\text{KL}}(P_R(S)||P_R(\tilde{S}))$ where $\hat{\xi}$ is some constant and replace ξ' with $\hat{\xi}' D_{\text{KL}}(P_R(S)||\tilde{P}_R(S)$ where $\hat{\xi}'$ is another constant. Essentially we want to continue updates of TRPO until the reward distribution shifts more than the predicted divergence of the next discriminator step and vice versa. We propose this as the beginning of a formulation for future experiments following our findings through qualitative experimentation with the number of TRPO/discriminator update steps.

4 Decomposable Rewards

In large multi-step tasks, it has been shown that training several different policies (options) across the state space can lead to improved results [16, 17]. Intuitively, learning the same policy across a widely varied domain would require the single policy to be overly complex and could result in over-generalization across the whole state space, while decomposing them leads to distinct actions to be taken across different steps in the task. A similar concept can be applied to reward structures in inverse reinforcement learning. In [18], a video demonstration is segmented in an unsupervised manner and distinct rewards are learned for each segmented step using a Maximum Entropy Inverse Reinforcement Learning (MaxEntIRL) method. We extend this idea further and more formally into the decomposable rewards framework. This framework directly parallels policy options in reinforcement learning.

4.1 Framework Formulation

In reinforcement learning, an option ($\omega \in \Omega$) can be defined with a triplet $(I_\omega, \pi_\omega, \beta_\omega)$. In this definition, π_ω is called an intra-policy option, $I_\omega \supseteq S$ is an initiation set, and $\beta_\omega : S \rightarrow [0, 1]$ is a termination function (that is the probability that an option ends at a given state) [17]. Furthermore, π_Ω is the policy over options. That is, π_Ω determines which option π_ω an agent picks to use until the termination function β_ω indicates that a new option should be chosen.

We directly parallel this formulation and define a reward option as follows. Instead of restricting ourselves to the call-and-return method of policy-options, we simply say that a reward option is defined as (R_ω, ζ_ω). That is, R_ω is the reward given a state input, paralleling a policy option, and $\zeta_\omega : S \rightarrow [0, 1]$, the selection approximator, is the probability that a given reward option should be used in a given state. Lastly, we learn a gating function (essentially a policy over reward options R_Ω) which simply chooses a reward option (or a mixture of reward options) given a state input. Intuitively, in the context of value functions in reinforcement learning, the optimal state value function over reward options would be written as:

$$V^*(s) = \max_{a \in A_s}[R_\Omega(s) + \gamma \sum_{s'} p^a_{ss'} V^*(s') \tag{4}$$

We apply our initial reward-options formulation in the context of GAIRL. Inherently, for IRL tasks, it becomes necessary to perform adversarial updates to the reward function and the policy. Here, we instead update our $\pi_R(s)$ and through it, each option and it's selection approximator. It is intuitive that in the context of policy options, these reward options can each be paired with policy options to learn different policies for multistep actions. We do not examine this here, but rather leave it for future work.

4.2 Formulation with Generative Adversarial Inverse Reinforcement Learning

To use decomposable rewards with GAIRL, we formulate decomposable rewards as follows.
First, we look at them as mixtures rather than options. Since we are learning rewards as part of the discriminator, we can define a decomposition of rewards similarly to the aforementioned mixtures of experts formulation found in training large decomposed neural networks [11].
We train Ω discriminators (decomposed rewards) with a learned composition function over the output of the discriminators. The composition function is another network of the exact same type as the discriminator, with Ω output units over a softmax function. These are then multiplied pointwise by the outputs of the discriminators to form a mixing function. This formulation results in the optimization problem:

$$L(R_\Omega) = \sum_\omega \zeta_\omega L(R_\omega) + w_{importance} \cdot CV(\sum_n D(S_n))^2 \tag{5}$$

with $L(R_\Omega)$ the cost over reward options, $w_{importance}$ a fixed weight and CV the coefficient of variance for the sum of all sample activations across discriminators. That is, the variance over mean of the summed activations between each discriminator.
We define the cost for a single discriminator to be the sigmoid cross-entropy:

$$L(R_\omega) = -\frac{1}{n} \sum_{n=1}^{N} [p_n \log \sigma(D_\omega(s_n)) + (1 - p_n) \log(1 - \sigma(D_\omega(s_n)))] \tag{6}$$

where $D_\omega(s_n)$ is the activation of a discriminator for a given sample s_n and p_n is the true label of that sample (1 for expert, 0 for novice).

We can then formulate a second way to generate more option-like decomposed rewards such that we take the top-1 activation rather than using a pointwise multiplication.

We thus generate two types of GAIRL decomposed rewards: MixGan (where R_Ω uses the summation over all weighted rewards as described above) and OptionGan (where we take the top-1 activation through a softmax, effectively choosing a reward-option for a given state). Figure 1 shows the architectural layout of both formulations.

As previously mentioned, the OptionGan formulation should partition the state space among the reward components with each component handling a mutually exclusive state space. This formulation is therefore fit for complex tasks which are decomposable into subtasks. This has a basis in policy options.
The MixGan additive reward formulation, on the other hand, finds a basis in the idea of biological motor primitives [19]. The theory of motor primitives suggests that muscle stimulation in animals is vectorially additive and originates from a lower dimensional combination of primitive activation functions (i.e. motor primitives).

Linking this idea with our GAIRL formulation above, each reward would be responsible for an individual actionable dimension of the agent, with the sum of the activations they incur accomplishing an action. This also reflects the aforementioned MoE models [10].

4.3 Implementation

Our implementation uses Tensorflow [20] for the neural network models and optimization functions, rllab [21] for its implementation of TRPO and general utility functions, and OpenAI Gym [22] for its defined benchmark environments. [1]

5 Experimental Results

To evaluate that our methods work, we run several experiments to attempt to characterize training behaviours. We then run all the algorithms until convergence to their best possible value and use the learned policy to generate rollouts to evaluate performance against the baselines.

5.1 Environments

We use two simple OpenAI Gym environments [22]. The first is the classic inverted pendulum control task, CartPole-v0 in OpenAI Gym, which uses the formulation from [23] and involves balancing a pole on a moving cart in a 2D plane with a +1 reward for every timestep that the pole is balanced within some angle range.
The second is MountainCar-v0 which uses the formulation found in [24]. This task gives a negative reward for every timestep that the car on the mountain has not reached a given goal.

5.2 Expert Policy

To generate the expert policies we train a policy using TRPO on the task with enough iterations for the expert to reach the OpenAI Gym threshold for "solving" the task, and record a set of 100 sample rollout trajectories with the resulting agent. We use the observations from these rollouts in our experiments (though we restrict the number of expert demonstrations used to 20 and even 5 in some cases to aim toward expert sample efficiency).

5.3 Baselines

We implement behavioural cloning as our baseline. While this may be an unfair comparison since behavioural cloning has access to the expert's actions, we wanted to see how close we could get to this without knowing the actions ahead of time. We also attempted to implement the algorithm presented in [3]. However, we found that it does not consistently converge to an optimal policy and often finds local solutions leading to poor results. In that work, the author suggests choosing the best one of a set of generated policies. It is unclear whether the authors simply chose the method with the best true reward. In any case, we do not show these results as we believe it is a bit unfair to choose the best policy if you assume you don't know the true rewards.

[1] We open source our code for easy reproduction of our obtained results or potential extensions: `https://github.com/Breakend/ExperimentsInIRL`

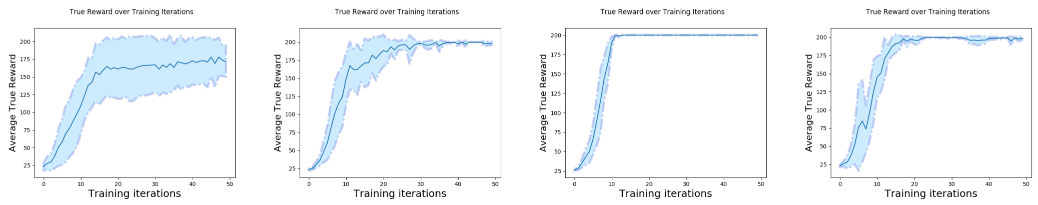

Figure 2: GAIRL, varying n=1, 2, 4, 8 from left to right respectively. We set importance weighting to .25 arbitrarily. Average is the solid line, variance is the highlighted region across five runs.

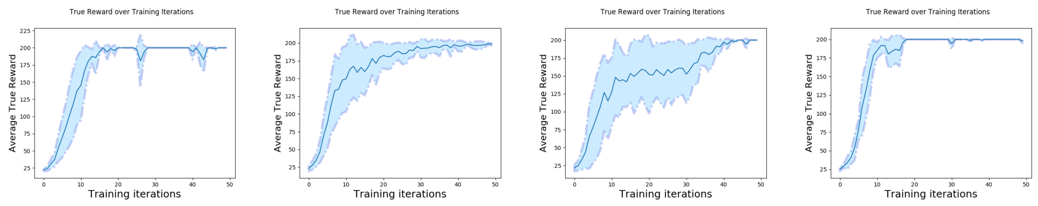

Figure 3: MixGan, varying n=1, 2, 4, 8 from left to right respectively. We set importance weighting to .25 arbitrarily. Average is the solid line, variance is the highlighted region across five runs.

5.4 Learning Performance

In the next sections we investigate learning performance on the simple CartPole task. We use 20 expert rollouts (across the entire training set) and 20 novice rollouts per iteration for all experiments.

5.4.1 Number of Frames Concatenated

As shown in figures 2, 3 and 4, we now experiment with the number of concatenated sequential states forming each training example. Each graph is generated by averaging the true reward over 5 runs. We can observe that GAIRL performs best when using 4 frames, while MixGan shows good performance when using 1 or 8 frames, and OptionGan perform best when using a single frame. However drawing conclusions from these results is difficult due to the high variance shown by the algorithm.

5.4.2 Importance Weighting

We also observe the effect of varying the importance weights used in MixGan and OptionGan. Each of the graphs is the result of averaging across 3 experiments on the Cartpole task. As we can see, both MixGan and OptionGan perform best with the lowest variance when not using any importance weighting.
We suspect the importance weighting formulation distributes the information in a poor way, leading the discriminators in both algorithms to see less specialized data and perform poorly overall. A better way to distribute the information would be to minimize the mutual information between the discriminators, as mentioned below in the Conclusion section.

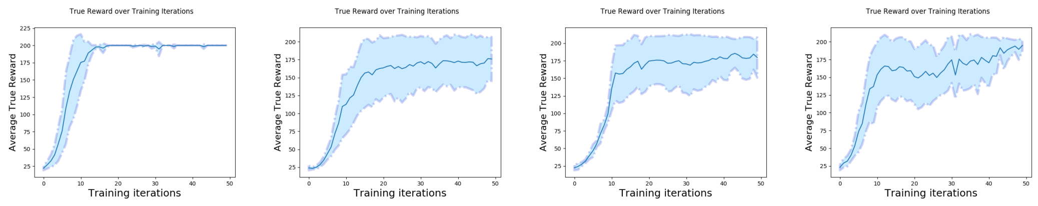

Figure 4: OptionGan, varying n=1, 2, 4, 8 from left to right respectively. We set importance weighting to .25 arbitrarily. Average is the solid line, variance is the highlighted region across five runs.

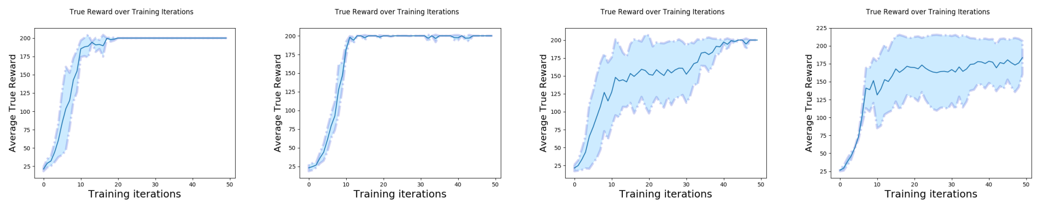

Figure 5: MixGan, holding n=4 constant and varying importance weighting from 0.0, 0.1, 0.25, 0.5 from left to right respectively. Average is the solid line, variance is the highlighted region across five runs.

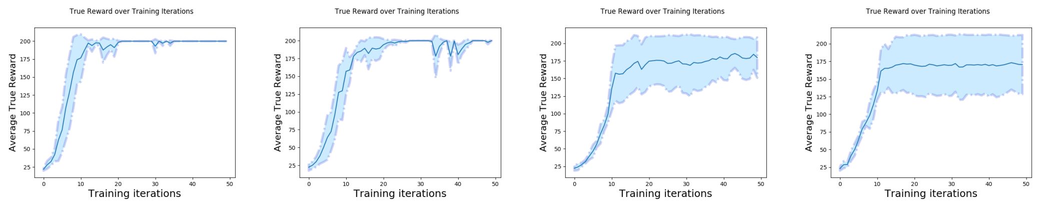

Figure 6: OptionGan, holding n=4 constant and varying importance weighting from 0.0, 0.1, 0.25, 0.5 from left to right respectively. Average is the solid line, variance is the highlighted region across five runs.

5.4.3 Mountain Car

We also investigate learning performance on the MountainCar task. Due to the high variance of the environment and the need for exploration, we found that stable learning required ≥ 100 sample rollouts per iteration. As a result, due to time constraints we only use 1 experiment for the graphing, while the true rewards shown in Table 1 were gathered on a longer experimental run (300 iterations). As presented in figure 7, all three algorithms obtain cumulative rewards close to the expert's, although with high variance.

5.5 Qualitative Analysis of Reward Options in Simple Settings

We can observe the effect of decomposing rewards using multiple discriminators. Figure 8 shows an example of how the state space gets decomposed along each dimension for the CartPole-v0 task. As can be seen here, there is not enough information to make use of all four options in the state space. Thus, two discriminators bear no information in this subsection of the state space (while it is possible for the state space to extend beyond, this is the region of interest for the CartPole task). We show that these two discriminators handle at least one distinct section of the state space along some dimensions and act as support in other dimensions. These graphs do not distinctly partition the hyperplane at which the mixtures may be divided, but they do give some insight into how information is distributed to each of the discriminators.

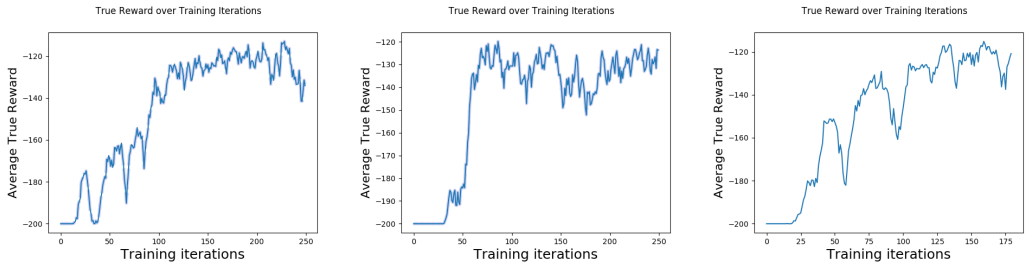

Figure 7: Average true reward for (left to right) GAIRL, MixGan and OptionGan. Note: OptionGan only had time to run 180 iterations versus 250.

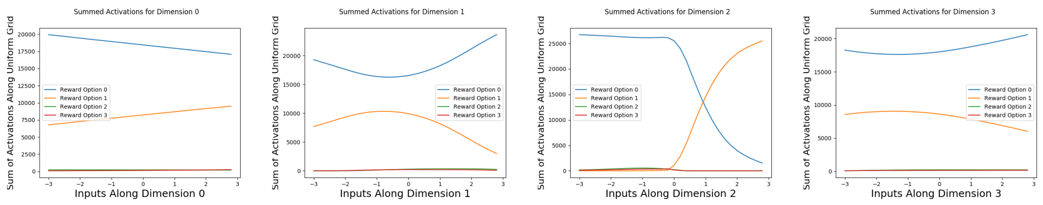

Figure 8: Activations across each reward-option for MixGan. This is just an example as the discriminators will converge to slightly different values every time in a non-deterministic fashion. We take an input dimension and sum all activations across the mesh of all possible values for the other dimensions to generate these graphs. This is run with only one frame in the observation set (i.e. no concatenation).

Task	Expert	Behavioural Cloning	GAIRL	MixGan	OptionGan
CartPole-v0	200 ± 0.0	200 ± 0.0	200 ± 0.0	200 ± 0.0	200 ± 0.0
MountainCar-v0	-100.5 ± 11.6	-111.95 ± 9.9	-121.68 ± 11.1	-124.13 ± 15.4	-121.38 ± 7.5

Table 1: True rewards across the final learned policies for sample rollouts (20 rollouts CartPole, 200 rollouts MountainCar). Parameters used for CartPole were: 10 expert rollouts, 20 sample rollouts per iteration, 30 iterations, importance coefficient 0.25. Parameters used for MountainCar were: 50 expert rollouts, 200 sample rollouts, 300 iterations, importance coefficient 0.25.
Note: MountainCar results are much more unstable and may vary across experiments.

5.6 Final Policy Evaluations

Table 1 shows the true environment reward obtained by the final learned policy for each method and comparison against our baseline of behavioural cloning and the expert policy trained using TRPO. Note that while behavioural cloning is an imitation learning task where the actions of the expert are known, we add it for comparison against such methods.

6 Conclusion

Overall, we find that the variance in the training for all our experiments was extremely high. We find that even if the algorithm reaches the maximum true reward, if left training for many more iterations, it is possible for the system to diverge from the optimal solution. This is the nature of the adversarial game being played by both the discriminator and the generator. From our qualitative improvements on the learning rate, we take a step towards formally addressing these instabilities. In the future, we wish to make this more principled than simply choosing values which seem to qualitatively work. As aforementioned, we derive a possible formulation to address this problem, but do not implement it in the context of this paper due to time constraints.

We believe that the decomposable rewards framework shows promise for several reasons. For small tasks it is comparable to GAIRL. It is decomposing the state space into several discriminators as previously discussed. Furthermore, at the same true reward, the accuracy of the optioned or mixing discriminator had a consistently higher classification accuracy than GAIRL. With stabilization, we believe the strength of this classifier can potentially lead the adversarial game down a more optimal trajectory. However, without stabilization, we find that this does not help significantly in the small tasks experimented with here.

We suggest several future directions and improvement to demonstrate the power of decomposable rewards. These generally fall under two categories: stability of the adversarial game and demonstration of the decomposable rewards framework.

For stabilizing the adversarial game, we suggest the following further improvements:

1. Improve stability with an entropy penalty [4]
2. Add temporal information to the discriminator [4]
3. Add a divergence constraint on the cost function [8]

4. Add our dynamic rate of training discriminator-generator updates (aforementioned here)

For demonstrating and improving decomposable rewards, we suggest:

1. Experiment with more complex multi-step tasks (e.g. Atari), this will likely require moving from state to pixel space inputs
2. Add a mutual information penalty between the decomposed rewards to try to force each reward option to classify a mutually exclusive state space region [25]
3. Pair reward options with policy options to improve learning multistep tasks.

We show that it is possible to learn decomposed rewards in simple settings, present framework for formulating reward options, investigate stability in the context of GAIRL models and propose solutions going forward.

Acknowledgments

We'd like to thank: Bradly Stadie for his rapid correspondence regarding his Third Person Imitation Learning paper and code; Pierre-Luc Bacon for his helpful suggestions in relevant works; and David Meger for his guidance. We'd also like to acknowledge OpenAI for their release of packages such as OpenAI Gym and RLLab which made building up our experiments easier. We use some snippets of the code released for [4, 9]. All code used is cited within the repository.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[4] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[5] R. E. Kalman. When is a linear control system optimal? *Journal of Basic Engineering*, 86(1):51–60, 1964.

[6] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[7] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1433–1438. AAAI Press, 2008.

[8] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, 2016.

[9] B. C. Stadie, P. Abbeel, and I. Sutskever. Third-Person Imitation Learning. *ArXiv e-prints*, March 2017.

[10] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, March 1991.

[11] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[12] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, pages 1–19, 2014.

[13] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[14] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, December 2014.

[15] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved Techniques for Training GANs. *ArXiv e-prints*, June 2016.

[16] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[17] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *arXiv preprint arXiv:1609.05140*, 2016.

[18] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016.

[19] P. S. Thomas and A. G. Barto. Motor primitive discovery. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–8, Nov 2012.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[21] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *ArXiv e-prints*, April 2016.

[22] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[23] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[24] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

[25] Yong Liu and Xin Yao. Learning and evolution by minimization of mutual information. In *International Conference on Parallel Problem Solving from Nature*, pages 495–504. Springer, 2002.