Deep Inverse Reinforcement Learning

Markus Wulfmeier
Mobile Robotics Group
Department of Engineering Science
University of Oxford
markus@robots.ox.ac.uk

Peter Ondrúška
Mobile Robotics Group
Department of Engineering Science
University of Oxford
ondruska@robots.ox.ac.uk

Ingmar Posner
Mobile Robotics Group
Department of Engineering Science
University of Oxford
ingmar@robots.ox.ac.uk

Abstract

This paper presents a simple yet general framework for employing deep architectures to solve the inverse reinforcement learning (IRL) problem. In particular, we propose to exploit the representational capacity and favourable computational complexity of deep networks to approximate complex, nonlinear reward functions in scenarios with large state spaces. This leads to a framework with the ability to make reward predictions in constant time rather than scaling cubically in the number of state rewards observed. Furthermore, we show that the maximum entropy paradigm for IRL lends itself naturally to the efficient training of deep architectures. The approach presented outperforms the state-of-the-art on a new benchmark with a complex underlying reward structure representing strong interactions between features while exhibiting performance commensurate to state-of-the-art methods on a number of established benchmarks of comparatively low complexity.

1 Introduction

The objective of inverse reinforcement learning (IRL) is to infer the underlying reward structure guiding an agent's behaviour based on observations and a model of the environment. This may be done either to learn the reward structure for modelling purposes (reward learning) or to provide a framework to allow the agent to imitate a demonstrator's specific behaviour (apprenticeship learning) [1]. While for small problems the complete set of rewards can be learned explicitly, many problems of realistic size require the application of function approximation.

Much of the prior art in this domain therefore relies on a specific parametrisation of the reward function, often based on features describing a state. The authors of [2, 3, 4], for example, express the reward function as a weighted linear combination of hand selected features. To overcome the inherent limitations of such a linear model [5, 6] extend this approach to a limited set of nonlinear rewards by learning a set of composites of logical conjunctions of atomic features. Non-parametric methods such as Gaussian Processes (GPs) have also been employed to cater for potentially complex, nonlinear reward functions [7]. While in principle this extends the IRL paradigm to the full gamut of nonlinear reward functions, the use of a kernel machine makes this approach prone to requiring a large number of reward samples in order to approximate complex reward functions [8]. This quickly renders the GP framework impractical due to the unfavourable computational complexity at query time ($\mathcal{O}(n^3)$ where n denotes the number of state-reward pairs in the active set).

In contrast to prior art, we explore the use of deep architectures to approximate the reward function. Deep networks (DNs) already achieve state-of-the-art performance across a variety of domains such as computer vision, natural language processing, speech recognition [9] and model-free deep reinforcement learning [10]. Their application in IRL is attractive due to their exceptional representational power as deep networks provide compact representations of highly nonlinear functions through the composition of many nonlinearities [8]. In addition, DNs provide favourable computational complexity ($\mathcal{O}(1)$) at query time, which provides for scaling to problems with large state spaces and complex reward structures – circumstances which might render the application of any of the existing prior art intractable or ineffective.

Our principal contribution is a framework for *deep inverse reinforcement learning* which brings to bear the significant representational power of deep architectures when learning reward structures from user demonstration. We show that the maximum entropy paradigm for IRL [2], which implicitly resolves expert suboptimality and stochasticity of the environment, lends itself naturally to use in a deep architecture and allows for efficient training. We demonstrate performance commensurate to state-of-the-art methods on a publicly available benchmark, while outperforming the state-of-the-art on a new benchmark where the true underlying reward has complex structure. We argue that these properties are important for practical large-scale applications of IRL with often complex reward functions requiring high capacity models and fast computational speeds.

2 Inverse Reinforcement Learning

This section presents a brief overview of IRL. For simplicity, we employ the notation introduced in [7]. Let a Markov Decision Process (MDP) be defined as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathbf{r}\}$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the set of possible actions, $\mathcal{T}$ denotes the transition model and $\mathbf{r}$ denotes the reward structure. Given an MDP, an optimal policy π^* is one which, when adhered to, maximizes the expected cumulative reward. In some cases an additional factor $\gamma \in [0, 1]$ may be considered in order to discount future rewards. In this work we focus on the undiscounted case $\gamma = 1$. The discounted case can be easily derived.

IRL considers the case where an MDP specification is available but the reward structure is unknown. Instead, a set of expert demonstrations $\mathcal{D} = \{\varsigma_1, \varsigma_2, ..., \varsigma_N\}$ are provided which are drawn from a user policy π. Each demonstration consists of a set of state-action pairs such that $\varsigma_i = \{(s_0, a_0), (s_1, a_1), ..., (s_K, a_K)\}$. The goal of IRL is to uncover the hidden reward $\mathbf{r}$ from the demonstrations.

A number of approaches have been proposed to tackle the IRL problem (see, for example, [3, 11, 4, 12]). An increasingly popular formulation is Maximum Entropy IRL (MaxEnt) [2], which was used to effectively model large-scale user driving behaviour. In this formulation the probability of user preference for any given trajectory between specified start and goal states is proportional to the exponential of the reward along the path

$$P(\varsigma|\mathbf{r}) \propto \exp\{\sum_{s,a \in \varsigma} r_{s,a}\}. \tag{1}$$

The principle of maximum entropy is employed in order to provide a model which does not require the demonstrations themselves to follow an optimal policy. Instead, observed behaviour can be randomly drawn from the above distribution, thus providing a likelihood for a user following any given trajectory. Moreover, MaxEnt leads to a likelihood which is differentiable with respect to the rewards $\mathbf{r}$. Learning is conducted by employing gradient descent in order to maximize the likelihood under the model of observing the expert demonstrations. As we will see in Section 3.1, in the context of our work this is an appealing property as this allows for the efficient optimisation of network weights using back-propagation.

2.1 Approximating the Reward Structure

In many real world applications of interest the MDP reward structure can not be observed (or stored) explicitly for every state. In these cases state rewards are not modelled directly but the reward structure is restricted by imposing that states with similar features, $\mathbf{x}$, should have similar rewards. To this end, function approximation is used in order to regress some representation of state onto a

real valued reward using a mapping $f : \mathcal{S} \rightarrow \mathbb{R}$ such that

$$r = f(\mathbf{x}, \theta). \tag{2}$$

The state representation, $\mathbf{x}$, is usually hand-crafted[1].

The choice of model used for function approximation has a dramatic impact on the ability of the algorithm to capture relationship between state feature vector $\mathbf{x}$ and user utility. Commonly (see, for example, [2] the mapping from state to reward is simply a weighted linear combination of feature values

$$f(\mathbf{x}, \theta) = \theta^\top \mathbf{x}. \tag{3}$$

This choice, while appropriate in some scenarios, is suboptimal if the true reward can not be accurately approximated by a linear model. In order to alleviate this limitation [5] extend the linear model by introducing a mapping $\mathbf{\Phi} : \mathbf{x} \rightarrow \{0, 1\}^N$ such that

$$f(\mathbf{x}, \theta, \Phi) = \theta^\top \mathbf{\Phi}(\mathbf{x}). \tag{4}$$

Here $\mathbf{\Phi}$ denotes a set of composite features which are jointly learned as part of the objective function. These composites are assumed to be the logical conjunctions of the predefined, atomic features $\mathbf{x}$. Due to the nature of the features used the representational power of this approach is limited to the family of piecewise constant functions.

In contrast, [7] employ a Gaussian Processes (GP) framework to capture the potentially unbounded complexity of any underlying reward structure. Here the set of expert demonstrations $\mathcal{D}$ is used to identify an active set of GP support points, $\mathbf{X_u}$, and associated rewards $\mathbf{u}$. The mean function is then used to represent the individual reward at a state described by $\mathbf{x}$

$$f(\mathbf{x}, \theta, \mathbf{X_u}, \mathbf{u}) = \mathbf{K}_{\mathbf{x},\mathbf{u}}^\top \mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1} \mathbf{u}. \tag{5}$$

Here $\mathbf{K}_{\mathbf{x},\mathbf{u}}$ denotes the covariance of the reward at $\mathbf{x}$ with the active set reward values $\mathbf{u}$ located at $\mathbf{X_u}$ and $\mathbf{K}_{\mathbf{u},\mathbf{u}}$ denotes the covariance matrix of the rewards in the active set computed via a covariance function $k_\theta(\mathbf{x}_i, \mathbf{x}_j)$ with hyperparameters θ. In practice, the entire reward structure $\mathbf{r}$ can be obtained from a single query. Nevertheless, a significant drawback of this GPIRL approach is a computational complexity cubic in the size of the active set for every query posed. While, therefore, the modelling of complex, nonlinear reward structures in problems with large state spaces is theoretically in the gift of the GPIRL approach, the cardinality of the active set will quickly become unwieldy, putting GPIRL at a significant computational disadvantage or, worse, rendering it entirely intractable. These shortcomings are remedied when using deep architectures for reward function approximation, as outlined in the next section.

3 Reward Function Approximation with Deep Architectures

We argue that IRL algorithms scalable to MDPs with large state spaces require models which are able to efficiently represent complex, nonlinear reward structures. In this context, deep architectures are a natural choice as they explicitly exploit the depth-breadth trade-off [8] for this purpose.

For the remainder of the paper, we consider deep neural networks (DNNs) which accept as input state features $\mathbf{x}$, map these to state reward r and are governed by network parameters $\theta_{1,2,..n}$. An example network is given in Figure 1. In the context of Section 2.1, the state reward is therefore obtained as

$$\begin{aligned} r &\approx f(\mathbf{x}, \theta_1, \theta_2, ..., \theta_n) && (6) \\ &= f_1(f_2(...(f_n(\mathbf{x}, \theta_n), ...), \theta_2), \theta_1). && (7) \end{aligned}$$

While many choices exist for the individual building blocks of a deep architecture it has been shown that a sufficiently large DNN with as little as two layers and sigmoid activation functions can represent any binary function [13] or any piecewise-linear function [14] and can therefore be regarded as a 'universal approximator'. Importantly, DNs also lend themselves naturally to training

[1]The state representation could be learned as part of the DeepIRL framework proposed here. However, in this paper we focus purely on the benefits of using DNs for function approximation in IRL and leave representation learning for future work

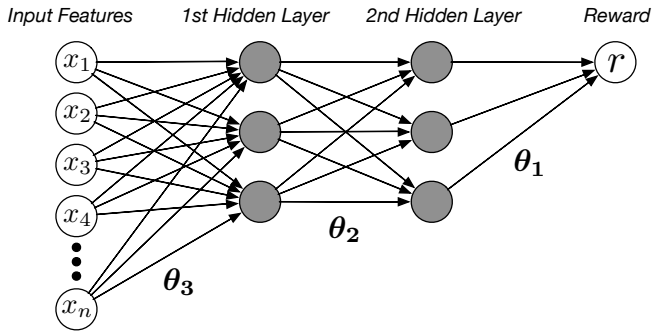


Figure 1: Example network structure for reward function approximation.

in the maximum entropy IRL framework (see Section 3.1). Therefore the network structure can be adapted to suit individual IRL tasks while maintaining the favourable properties of capability, scalability and adaptability and without invalidating the main IRL learning mechanism.

In the DeepIRL framework proposed here the full range of architecture choices thus becomes available. Different problem domains can utilise different network architectures. For example, it is interesting to note (and straightforward to show) that the linear maximum entropy IRL approach proposed in [2] can be implemented exactly by applying the rules of back-propagation to a network with a single linear output connected to all inputs with zero bias term.

In practice, several factors influence the appropriate structure of the network. While the complexity of the true underlying reward is often unobservable the amount of training data available is one such factor. As the number of training data increases, adding hidden layers and increasing the number of nodes will allow the network to fit more complex feature dependencies and thus model more complex rewards. However, increasing the capacity of the network excessively will result in over-fitting. This can be remedied via cross-validation, for example by evaluating the likelihood of observing a hold-out set of expert trajectories using Equation 1 given the learned rewards. This method was used to design the network in Section 4.

3.1 Learning

The task of solving the IRL problem can be framed in the context of Bayesian inference as MAP estimation maximizing the joint posterior distribution of observing expert demonstrations, $\mathcal{D}$, under a given reward structure and of the model parameters θ.

$$\log P(\mathcal{D}, \theta|\mathbf{x}) = \underbrace{\log P(\mathcal{D}|\mathbf{r})}_{\mathcal{L}_\mathcal{D}} + \underbrace{\log P(\theta)}_{\mathcal{L}_\theta}. \tag{8}$$

The reward structure can be approximated such that $r = f(\mathbf{x}, \theta))$ as described in Section 2.1.

We will now show that this joint log likelihood is differentiable with respect to the network parameters θ, which allows the application of gradient descent methods [15]. This makes the maximum entropy IRL formulation ideally suited for DNN training via back-propagation.

The gradient is given by the sum of the gradients with respect to θ of the data term $\mathcal{L}_\mathcal{D}$ and the model term $\mathcal{L}_\theta$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}_\mathcal{D}}{\partial \theta} + \frac{\partial \mathcal{L}_\theta}{\partial \theta}. \tag{9}$$

The gradient of data term can be expressed in terms of the derivative of the expert demonstration with respect to rewards as well as the derivative of these rewards with respect to network weights θ, such that

$$\frac{\partial \mathcal{L}_\mathcal{D}}{\partial \theta} = \frac{\partial \mathcal{L}_\mathcal{D}}{\partial \mathbf{r}} \cdot \frac{\partial \mathbf{r}}{\partial \theta} \tag{10}$$

$$= (\mu_\mathcal{D} - \mathbb{E}[\mu]) \cdot \frac{\partial}{\partial \theta} f(\mathbf{x}, \theta), \tag{11}$$

where $r = f(\mathbf{x}, \theta))$. As shown in [2], the gradient of the expert demonstration term $\mathcal{L}_{\mathcal{D}}$ with respect to state rewards $\mathbf{r}$ is equal to difference in the state visitation counts exhibited by the expert demonstrations and the expected visitation counts given behaviour exhibited by the probability of a user visiting a sequence of states as per Equation 1,

$$\mathbb{E}[\mu] = \sum_{\varsigma:\{s,a\}\in\varsigma} P(\varsigma|\mathbf{r}). \tag{12}$$

Computation of $\mathbb{E}[\mu]$ usually involves summation over exponentially many possible trajectories. In [2] an effective algorithm based on dynamic programming was given computing this quantity in polynomial-time. The effective computation of the gradient $\frac{\partial \mathcal{L}_{\mathcal{D}}}{\partial \theta}$ thus involves first computing the difference in visitation counts using this algorithm and then passing this as an error signal through the network using back-propagation.

The derivative of the model term $\mathcal{L}_\theta$ with respect to the network parameters is expressed as a regularisor. There exist a variety of choices to prevent over-fitting in deep neural networks. L_1 and L_2 regularisation are used as well as drop-out, which can be seen as a sub-model averaging approach, or methods corrupting the original training data, for example, by adding noise. In the case of L_2 regularization the gradient is given as

$$\frac{\partial \mathcal{L}_\theta}{\partial \theta} = \lambda\theta. \tag{13}$$

4 Experiments

We assess the performance of DeepIRL on an existing as well as a new benchmark task against current state-of-the-art approaches : GPIRL [7], BNP-FIRL [5] and MaxEntIRL [2].

All tests are run multiple times on training and transfer tests for the different scenarios, while learning is performed based on synthetically generated stochastic demonstrations based on the optimal policy to evaluate performance on suboptimal examples sets.

In our experiments we employ a fully connected feed-forward network with two hidden layers and sigmoid activation functions (see Figure 1). While a number of architectures were explored, three nodes in each hidden layer were found to be sufficient for the network to perform well across all tasks considered. Conjugate gradient descent was used for optimisation of Equation 9 with a small number of random restarts. For larger scenarios stochastic gradient descent accompanied with methods such as AdaGrad [16] can be used.

In line with related works (e.g. [7]), we use *expected value difference* as principal method of evaluation. It is a measure of the sub-optimality of the learned policy under the true reward. The score represents the difference between the value function obtained for the optimal policy given the true reward structure and the value function obtained for the optimal policy based on the learned reward model. Additionally to the evaluation on each specific training scenario, the trained models are evaluated on a number of randomly generated test environments. The test on these *transfer* examples serves to analyse an algorithm's ability to capture the true reward structure without over-fitting to the training data.

4.1 Objectworld Benchmark

The *Objectworld* scenario [7] consists of a map of $M \times M$ states for $M = 32$ where possible actions include motions in all four directions as well as staying in place. Two different sets of state features are implemented based on randomly placed colours to evaluate the algorithms. For the continuous set $\mathbf{x} \in \mathbb{R}^C$. Each feature dimension describes the minimum distance to an object of one of C colours. Building on the continuous representation the discrete set includes $C \times M$ binary features, where each dimension indicates whether an object of a given colour is closer than than a threshold $d \in \{1, ..., M\}$. The reward is positive for cells which are both within the distance 3 of color 1 and distance 2 of the color 2, negative if only within distance 3 of color 1 and zero otherwise. This is illustrated for a small subset of the state space in Figure 5. The algorithm is provided with a number of demonstrations sampled from an optimal policy based on the true reward structure. However, in 30% of the cases a random action is performed instead of following the policy.

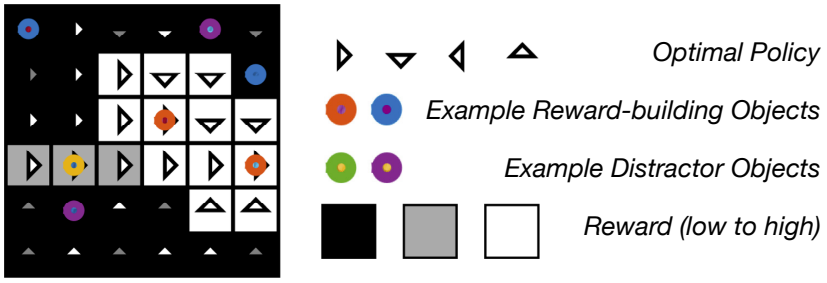


Figure 2: Objectworld benchmark. The true reward is displayed by the brightness of each cell and based on the surrounding object configuration. Only a subset of colors influences the reward, while the others serve to create distracting features.

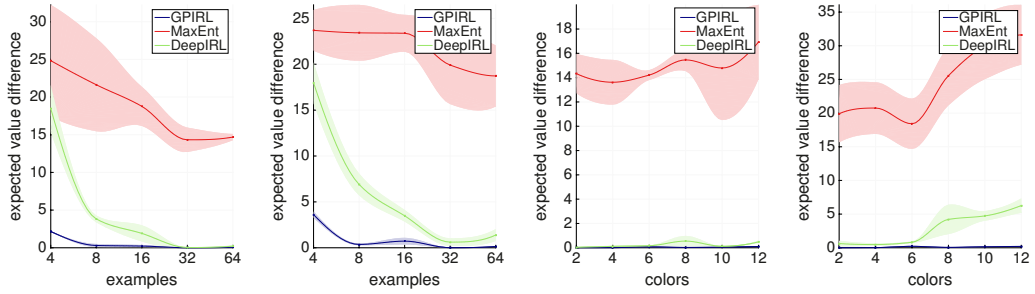


Figure 3: Objectworld benchmark with continuous features. From left to right: expected value difference (EVD) with $C = 2$ colours and varying number of demonstrations N for training (a) and transfer case (b); EVD for set number of demonstrations $N = 64$ and varying number of colors C for training (c) and transfer case (d). As the number of demonstrations grows DeepIRL is able to quickly match performance of GPIRL on the task.

All three algorithms were evaluated on the same scenarios in two different test cases: with a set number of colours and increasing demonstrations; and with a set number of demonstrations and increasing number of colours. The latter investigates the robustness of the algorithm to the presence of distractor variables. Additionally, the learned reward functions are deployed on randomly generated transfer scenarios to uncover any overfitting to the training data. As in [7], each test is repeated eight times.

While the original MaxEnt is unable to capture the nonlinear reward structure well, both DeepIRL and GPIRL provide significantly better approximations. Evaluation of BNP-FIRL on this benchmark was done in [5] where it showed the same level of performance as GPIRL. The GPIRL generates a good model already with few data whereas DeepIRL achieves commensurate performance when increasing the number of available expert demonstrations. The same behaviour is exhibited when using both continuous and discrete state features (Fig. 3 and 4, respectively) and is consistent with our conjecture regarding a need for more training data. Both DeepIRL and GPIRL show robustness to distractor variables, though DeepIRL shows signs of overfitting in the continuous feature transfer task as the number of distractor variables is increased. This is due to the DNN's capacity being brought to bear on the increasing noise introduces by the distractors.

4.2 Binaryworld Benchmark

In order to test the ability of all approaches to successfully approximate more complex reward structures, the *Binaryworld* benchmark is presented. This test scenario is similar to *Objectworld*, but in this problem every state is randomly assigned one of two colours (blue or red). The feature vector for each state consequently consists of a binary vector of length 9, encoding the colour of each cell

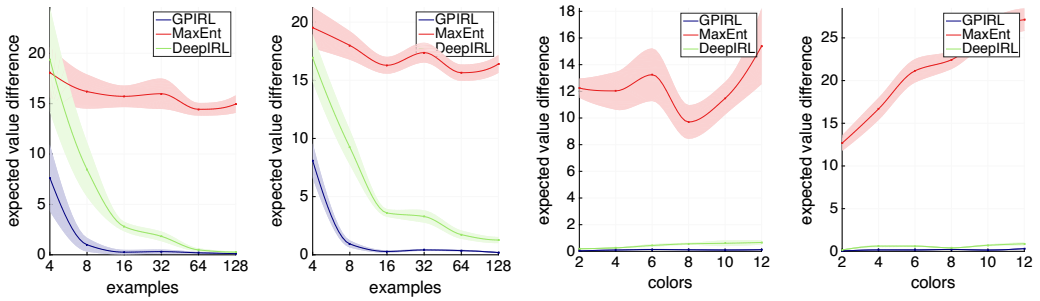


Figure 4: Objectworld benchmark with discrete features. From left to right: expected value difference (EVD) with $C = 2$ colours and varying number of demonstrations for training (a) and transfer case (b); EVD for set number of demonstrations $N = 64$ and varying number of colors C for training (c) and transfer case (d). Similar to the continuous case, as the number of demonstrations grows DeepIRL is able to quickly match performance of GPIRL on the task.

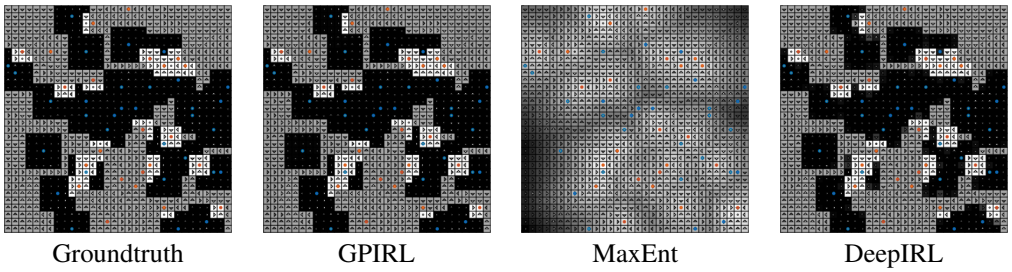


Figure 5: Reward reconstruction in Objectworld benchmark using GPIRL, MaxEnt and DeepIRL provided $N = 64$ examples and $C = 2$ colours with continuous features.

in its 3x3 neighbourhood. The true reward structure for a particular state is fully determined by the *number* of blue states in its local neighbourhood. It is positive if exactly four out of nine neighbouring states are blue, negative if exactly five are blue and zero otherwise. The main difference in here compared to the Objectworld scenario is that a single feature value does not carry much weight, but rather that higher-order relationships amongst the features determine the final reward.

The performance of DeepIRL compared to GPIRL, linear MaxEnt and BNP-FIRL is depicted in Fig. 6. In this increasingly more complex scenario DeepIRL is able to learn the higher-order dependencies between features, whereas GPIRL struggles as the inherent kernel measure can not correctly relate the reward of different examples with similarity in their state features. GPIRL needs a larger number of demonstrations to achieve better performance and to determine a good estimate on the reward for all 2^9 possible feature combinations.

Perhaps surprising is the comparatively low performance of the BNP-FIRL algorithm. This can be explained by the limitations of this framework. The true reward in this scenario can not be efficiently described by the logical conjunctions used. In fact, it would require 2^9 different logical conjunctions, each capturing all possible combinations of features, to accurately model the reward in this framework.

Fig. 7 shows the reconstruction of the reward structures estimated by DeepIRL, MaxEnt and GPIRL. While GPIRL was able to reconstruct the correct reward for some of the states having features it has encountered before it provides inaccurate rewards for states which were never encountered. On the other hand, DeepIRL is able to reconstruct it with high accuracy demonstrating the ability to effectively learn the complex structure of the underlying function.

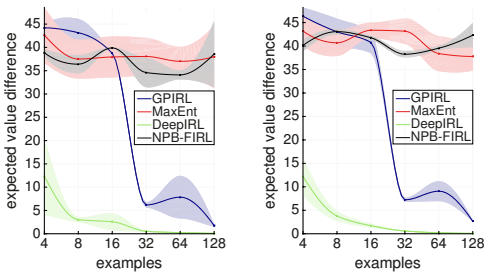
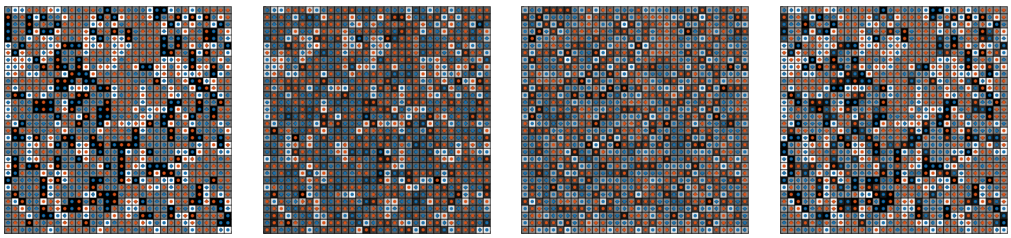


Figure 6: Value differences observed in the Binaryworld benchmark for GPIRL, MaxEnt and DeepIRL for the training scenario (left) and the transfer task (right).



Groundtruth GPIRL MaxEnt DeepIRL

Figure 7: Reward reconstruction for the Binaryworld benchmark using GPIRL, MaxEnt and DeepIRL provided $N = 128$ demonstrations.

5 Conclusion and Future Work

This paper presents DeepIRL, a framework exploiting DNNs for reward structure approximation in inverse reinforcement learning. DNNs lend themselves naturally to this task as they combine representational power with computational efficiency compared to state-of-the-art methods. Unlike prior art in this domain DeepIRL can therefore be applied in cases where complex reward structures need to be modelled for large state spaces. Moreover, DNN training can be achieved effectively and efficiently within the popular maximum entropy IRL framework. A further advantage of DeepIRL lies in its versatility. Custom network architectures and types can be developed for any given task while exploiting the same cost function in training.

Our experiments show that DeepIRL performance is commensurate to the state-of-the-art on a common benchmark. We also provide an alternative evaluation on a new benchmark with a significantly more complex reward structure. Here DeepIRL significantly outperforms the current state-of-the-art.

In future work we will explore the benefits of different network types in the DeepIRL context. For example, convolutional neural networks (CNNs) present an opportunity to incorporate state-space context directly into network training. Another, perhaps more apparent, area of interest is the use of CNNs to learn appropriate state representations directly from data.

References

[1] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51:61801, 2007.

[2] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438, 2008.

[3] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[4] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

[5] Jaedeug Choi and Kee-Eung Kim. Bayesian nonparametric feature construction for inverse reinforcement learning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1287–1293. AAAI Press, 2013.

[6] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1342–1350, 2010.

[7] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[8] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5), 2007.

[9] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

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

[11] Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. *CoRR*, abs/1206.5264, 2012.

[12] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2007.

[13] Mohamad H Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[15] Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer Science & Business Media, 2005.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.