Inverse Reinforcement Learning- Key Concepts, Fundamental Algorithms and Kernel based Methods

Pulkit Khandelwal (260717316)

April 22, 2017

Abstract

In this course project the author explores the concept of *Inverse Reinforcement Learning*. We will see what inspired the introduction of Inverse Reinforcement Learning, then see what early algorithms got introduced. We will then see a related concept of *Apprenticeship Learning*. Thereafter, we will encounter some entropy models and naturally extent to a deep neural network framework. Lastly, we will conclude with a non-linear kernel based method for IRL. The goal remains to understand IRL and its fundamentals.

1 Introduction

In the year 1998 [1], Russell introduced IRL in a talk and in an extended abstract as:

Given

- measurements of an agent's behavior over time, in a variety of circumstances
- if needed, measurements of the sensory inputs to that agent
- if available a model of the environment

Determine the reward function being optimized.

The problem of IRL in Markov Decision Processes is the problem of extracting a reward function given some observed optimal behavior. The motivation for IRL can be seen in animal and human learning. Such models can be seen in behavioral and neurophysiological studies. For example, models of bee foraging assume that the reward is a function of the nectar content of the flowers. Once can also add several attributes to this model such as flight distance,time, weather, predators, time of the day and the likes. Thus, one can use a certain type of empirical investigation to do this type of task.

IRL may also be useful to another closely related task of *Imitation Learning*. This is the problem an agent tries to learn to acquire skilled behavior and also recover a reward function. So, the agent can try to recover a reward function and thereby learn to generate

the desirable behavior. Some of the very early work in the field of imitation learning was done in economics for the task of multi attribute utility assessment.

Loosely speaking, the field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy, is the most succinct, robust and transferable definition of the task. Hence, it seems likely that IRL is an important area of spending some quality study time. Though recovering the reward function is not easy as it seems. We cannot recover a unique reward function due to the degeneracy issues. There exist a large set of reward functions for which the observed policy is optimal. For example if $\boldsymbol{R}$ is a reward function then some constant times $\boldsymbol{R}$ also satisfies the optimality conditions. And, so does $\boldsymbol{R}$=*const*.

Ng and Russell [2] tries to resolve this ambiguity by maximally differentiating the observed policy and other sub-optimal policies. This Linear Programming formulation is explained in Section 2. Four years later, Abbeel and Ng show a use case of IRL for Apprenticeship Learning where they try to match feature expectations and then recover the reward function. Their method is explained in Section 3. In section 4 we will see the method proposed by Riebart et. al [3] where they try to resolve an ambiguity in Abbeel and Ng's paper through an entropy based model. The ***ambiguity*** is that each policy can be optimal for many reward functions and many policies can lead to same feature counts. In section 5 we see Posner's method which extends the entropy method to a non-linear case with artificial neural networks. In section 6 we will see [6] use non-linear kernel based IRL.

We will also walk through how deep entropy model and kernel based IRL has been extended and implemented in this project report. These models provide and effective representation of the reward function in terms of non-linear state features. They automatically provide with good state feature construction and selection; and are learned using a *black-box* flavor of neural networks. Furthermore, we see how kernel based features provide good reward function approximation.

Note Throughout the report, the symbols and notations have the usual meaning as followed in the class and the text [7].

2 IRL: Linear Programming Formulation

Ng and Russell [2] proposed the first algorithm for IRL. They proposed algorithms for small and large state spaces. This was based on a simple linear programming formulation. Let's see how they came up with the method using a simple proof as follows,

They proved that there exists an optimal policy which is better than policies by maximally separating them. Let there be a finite state space S, a set of actions $A = a_1, a_2, ..., a_k$, transition probability matrices P_a, a discount factor $\gamma \in (0, 1)$ be given. Then the policy $\pi(s) = a_1$ is optimal if and only if, for all $a = a_2, a_3, ..., a_k$, the reward $\mathbf{R}$ satisfies

$$(P_{a_1}(i) - P_a(i))(I - \gamma P_{a_1})^{-1}\mathbf{R} \succeq 0$$

We know that,

$V^{\pi} = (I - \gamma P_{a_1})^{-1}\mathbf{R}$

$a_1 = \pi(s)$
Proof:

We see that $a_1 = \pi s$ is optimal if and only if,

$$a_1 = \pi(s) \in argmax_{a \in A} \sum_{s'} P_{sa}(s') \tag{1}$$

$\forall s \in S$
Implies that,

$\sum_{s' P_{sa_1}(s')} \geq \sum_{s' P_{sa}(s')}\ \forall s \in S and \forall a \in A \setminus a_1$

$\therefore P_{a_1} V^{\pi} \succeq P_a V^{\pi}$,

Hence,

$$P_{a_1}(I - \gamma P_{a_1})^{-1} R \succeq P_a (I - \gamma P_{a_1})^{-1} R \tag{2}$$

$\forall a \in A \setminus a_1$

This completes the proof and now we can recover the underlying reward function for the IRL problem in hand as the following ***LP formulation***,

maximize $\sum_{i=1:N} min_{\in \{a_2,.....,a_k\}} \{(P_{a_1}(i) - P_a(i))(I - \gamma P_{a_1})^{-1}\} - \lambda ||R||_1$

such that:
$(P_{a_1}(i) - P_a(i))(I - \gamma P_{a_1})^{-1} \succeq 0$
$\forall a \in A \setminus a_1$

with the condition that,
$|R_i| \leq R_{max}$, i=1,2,.....,N

For large state spaces, we represent the reward function as a combination of features of the states,

$$R(s) = \alpha_1 \phi_1(s) + \alpha_2 \phi_2(s) + ... + \alpha_d \phi_d(s) \tag{3}$$

Also,

$$V^{\pi} = \alpha_1 V_1^{\pi} + \alpha_2 V_2^{\pi}(s) + ... + \alpha_d V_d^{\pi} \tag{4}$$

$\therefore$ for large state spaces,

maximize $\sum_{s \in S_o} min_{\in \{a_1, a_2, ..., a_k\}} f(E_{s' \in P_{sa_1}}(V^{\pi}(s^{'})) - E_{s' \in P_{sa}}(V^{\pi}(s^{'})))$

such that:

$|\alpha_i| \leq 1$, i=1,2,.....,d.

A weight-decay term is incorporated into the formulation which helps in balancing the trade-off between having small reinforcements and maximize the criteria in the LP formulation.

3 Apprenticeship Learning via IRL

Abbeel and Ng [8] optimize the unknown reward function by matching state feature expectations and then taking a dot product of the weights αs and the feature expectation values to get the reward function. They assume that some expert trajectories have been provided ***The reward function obtained might not be the true reward function but can be a near-optimal one***. The performance guarantees depend ***approximately*** only on recovering the true underlying reward function.

$$E_{s_o \in D}[V^{\pi}(s^{'})] = E[\sum_{t=0}^{\infty} \gamma^t R(S_t) | \pi] \tag{5}$$

$$= E[\sum_{t=0}^{\infty} \gamma^t w.\phi(s_t) | \pi] \tag{6}$$

$$= w.E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi] \tag{7}$$

The feature expectation is then given by,

$\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi]$

We can then recover the reward function as,

$$R = w^T \phi \tag{8}$$

4 Maximum Entropy Model for IRL

Building upon the work of [8], Ziebart et. al [3] tries to solve the ambiguity based on an energy based formulation.

Let ζ be the set of path trajectories. The reward value of the trajectory is the sum of state rewards or put in other words, the reward weight applied to the path feature counts, $f_\zeta = \sum_{s_j \in \zeta} \mathbf{f}_{s_j}$.Hence, reward can be expressed as, Reward$(f_\zeta) = \theta^T f_\zeta$. Now, expected feature counts over all the trajectories $\overline{f} = (1/\mathrm{m}) \sum_i f_\zeta$. Matching feature expectations gives us the proposed formulation for an entropy based probability function$\overline{f} = \sum_{\zeta_i} P(\zeta_i)\mathbf{f}_{\zeta_i}$. $P(\zeta_i) = (1/Z(\theta))e^{\theta^T \mathbf{f}_{\zeta_i}}$ is the final form.

Under this model, plans, paths and states with higher rewards are exponentially more preferred. Given appropriate parameter weights, the partition function R(θ) always converges for finite horizon problems with discounted reward weights. [3] show an efficient state frequency expectation calculation as *Algorithm 1.* This method thus provides a convex, computationally efficient procedure for optimization with promising performance guarantees.

Ambiguity which is resolved

- Each policy can be optimal for many reward functions
- Many policies can lead to same feature counts

5 Deep non-linear Maximum Entropy Model for IRL

The Maximum Entropy model is extended to a deep learning framework. This leads to a much richer state feature representation with various activations. These activation functions can provide further insights into how good non-linearity and complex functions of the states and their features can lead to a very good approximation of the reward function in the IRL setting. Let,

$$R(s) = \alpha.\phi(s) \tag{9}$$
$$= g(W.\phi(s)) \tag{10}$$
$$\tag{11}$$

where, g is an activation function.

We can make this network deeper and deeper and use more relevant activation functions. More detailed explanation of the implementation is given in in Section 7. The derivatives are computed using simple backpropagation algorithm. [5] pose a similar approach.

6 IRL via Non-linear Gaussian Process

Levine et. al [6] proposed a non-linear feature approximation of the state spaces. They do so using a kernel based approach and gaussian priors. Some of the main equations from the paper are:

$$P(u,\theta|D,X_u) \propto P(D,u,\theta|X_u) = [\int_r P(D|r)P(r|u,\theta,X)dr]P(u,\theta|X_u) \tag{12}$$

Where, the posterior $P(r|u,\theta,X_u)$ is the probability of a reward function under the current values of u and θ, and $P(u,\theta|Xu)$ is the prior probability of particular assignment to u and θ. The kernel is then chosen. We can also calculate the covariance metric between the different features to find th relevance of each of the feature. In this report a RBF kerne has been used. More details follow in Section 7. RBF kernel,

$$K(x,x^{'}) = exp(\frac{-||x-x^{'}||^2}{2\sigma^2}) \tag{13}$$

Here, x is the input and $x^{'}$ is the mean of the different obtained features and σ is the standard deviation of the features.

7 Experiments

This section provides an in-depth account of the different experiments carried out as part of this study. Note all the variations, extensions, adaptations of the different algorithms, methodologies, hyperparameters, functions and other relevant details.
The environment is a simple GridWorld with wind influence at each state with a random probability of 0.3

- *Linear Programming* The original LP formulation is implemented for both small and large state spaces for a 5X5 and 10X10 grid with discount $\gamma = 0.5$ and $\gamma = 0.9$. This LP formulation is then extended to a non-linear setting where a Radial Basis Function is used.
- *Maximum entropy and its Deep extension* for a 10X10 grid at $\gamma = 0.9$. This is then compared to a deep learning extension with various activation functions such as identity mapping, coordinate mapping, exponential function, sigmoid and a RBF kernel. *Adam* is the optimization algorithm used because of its large popularity in the literature.
- *Gaussian based RBF kernel and its deep extension* for a 10X10 grid at $\gamma = 0.9$. The non-linear Gaussian priors have been used for non-linear state features to draw meaningful conclusions.

  Abbeel [8] uses the notion of Expected Value Difference for comparison of the optimal policies obtained using the recovered reward functions. Here, that metric has been used. Also, the accompanying figures show the values of the recovered reward functions as a heat map.

The author of this project report would like to thank M. Alger [4] for providing his open source implementation of some of the IRL algorithms. The project work adopts some of his codebase due to limitation of time and metric insights.

8 Results and Discussion

Expected value Difference and the heat maps for the recovered reward functions provide an excellent way to compare the results of various algorithms being studied. Thanks to the open source community. [4] This section details the results and give a discussion for each method.

In figure 1(a) one can see the reward function for a 10X10 grid. On the upper right corner, the reward value is 1 which is achieved when the agent reaches the goal state. Figure 1(b) shows the recovered reward function when the agent does not have any chance of blowing away to a random state. LP formulation fails to recover the reward function appropriately. The values of the rewards are very very low and of the order of e-12. This reward function is nowhere near to the optimal reward function, though it can be a reward function due to the degeneracy issue. This LP formulation though guesses the correct pattern of assigning the reward functions, in the sense that, the states which are closer to the terminal state has a higher reward value than other states as they try to tell the agent that you are near your desired goal. In figure (2) and (3) this experiment is carried for $\gamma = 0.5$ and $\gamma = 0.9$. $\gamma = 0.5$ gives us better convergence results for the reward function. This setting for the LP formulation seems to be optimal. In figure (4) we can see three linear feature representations for LP. *Sigmoid* does a better job as seen in figure (4:c) while an exponential weighting of the states can lead to bad results as apparent. Finally, we see the extension of LP to a non-linear case with RBF kernels. We see a remarkable improvement over other methods and settings. The order of the recovered reward functions is e-07 which is five fold improvement over other LP methods tried.

In figure 5 we can see the reward functions for (a)$\gamma = 0.5$ and (b)$\gamma = 0.9$. The former discount setting gives near optimal solution for the Max. Entropy algorithm. In figure 6 we can see the various hyperparamters and tuning for different Deep models for the Maximum Entropy model. The deeper the model the better it is. Notice how these methods are better than LP. The rewards are in between 0 and 9. See how good the network is at assigning just the terminal state very high value and rest of the states, a value where the reward is as desired for those states. The optimization procedure is *Adam*. The network has two settings for 5 and 10 hidden units and layers at $\gamma = 0.9$. *ReLU* has not been used as an activation function because the author believes that some states might blow up due to the sparsity created in the state representation.

The best results are obtained when a Gaussian based prior using an RBF kernel is used for a Deep Maximum Entropy Model. See figure (8). The reward values are between -1 and 2 which is really close to the optimal groundtruth reward function. Also, the environment lets the agent know what rewards to expect when a state is encountered.

Figures 9 through 13 shows the recovered state value functions. Deep models give very near-optimal policies. The table shows the Expected Difference Values for the different feature representations for the recovered optimal values. Though, the RBF kernel provides the best recovered reward function. It fails to give the best optimal values for the states. This might be because the features that are found using the RBF kernel might lie in some

other space/regions than the actual trajectories are followed at test time. This is another consideration to look after where we try to make sure that both the recovered functions and the states' value lie in the same space.

9 Conclusion

In this project we have seen how some of the algorithms have been proposed and evolved in Inverse Reinforcement Learning from Linear Programming, Entropy based and kernel based methods. We then extended these models to a non-linear setting with deep networks and Gaussian priors. We also saw related concept of Imitation Learning and how IRL can be used to solve that problem. We conclude that deep models provide a very efficient way of computing different meaningful feature representation for states. We have used both linear and non-linear combinations of them to estimate the unknown reward function. RBF kernels with deep model provide an additional benefit of first refining the features by taking the states to a *different dimension* and then use activation functions to get an enriched feature representation. In future, one can come with better ways to make sure that the obtained reward functions also give an optimal value and policy. Also, more kernel based methods can be developed because they seem to give promising results as seen in the report.

References

[1] Russell, Stuart. "Learning agents for uncertain environments." Proceedings of the eleventh annual conference on Computational learning theory. ACM, 1998.

[2] Ng, Andrew Y., and Stuart J. Russell. "Algorithms for inverse reinforcement learning." Icml. 2000.

[3] Ziebart, Brian D., et al. "Maximum Entropy Inverse Reinforcement Learning." AAAI. Vol. 8. 2008.

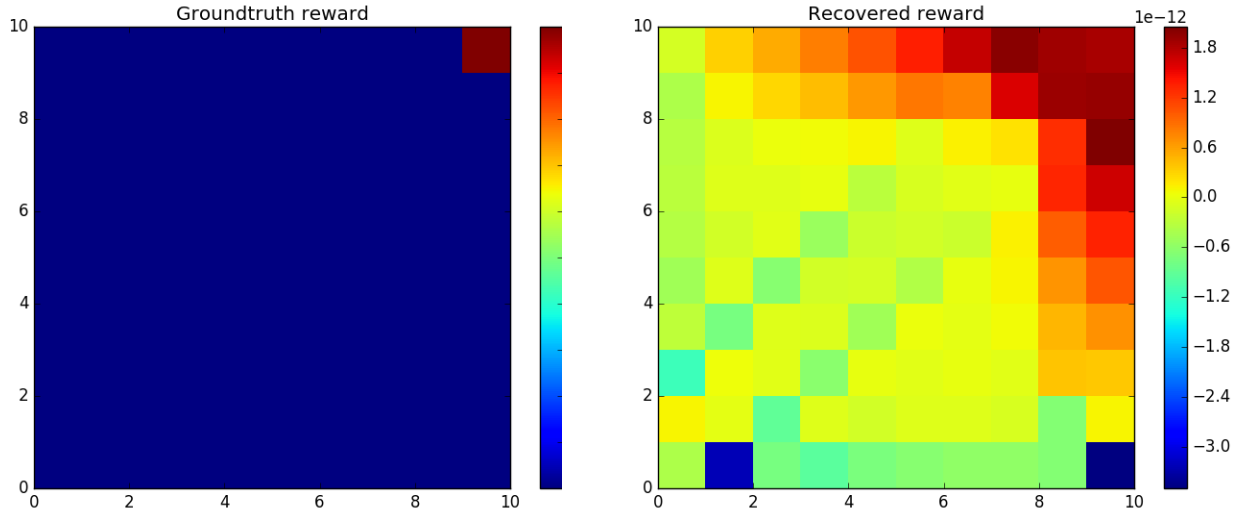
[4] Matthew. A, Australian National University

[5] Wulfmeier, Markus, Peter Ondruska, and Ingmar Posner. "Deep inverse reinforcement learning." CoRR (2015).

[6] Levine, Sergey, Zoran Popovic, and Vladlen Koltun. "Nonlinear inverse reinforcement learning with gaussian processes." Advances in Neural Information Processing Systems. 2011.

[7] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.

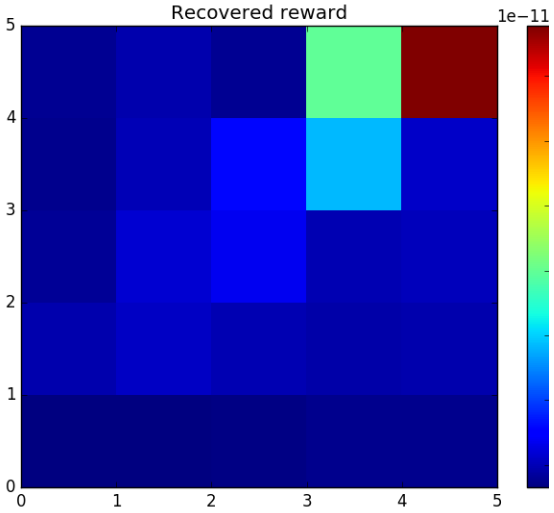
[8] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

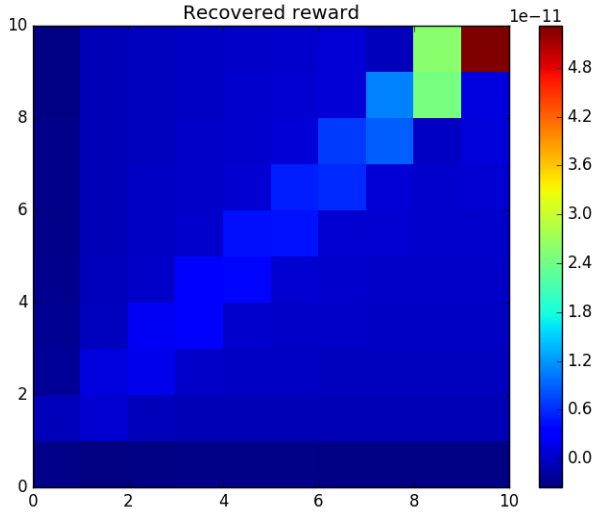


(a) (b)

Figure 1: Linear Programming for small state spaces(a) Groundtruth and (b) No wind $\gamma = 0.9$

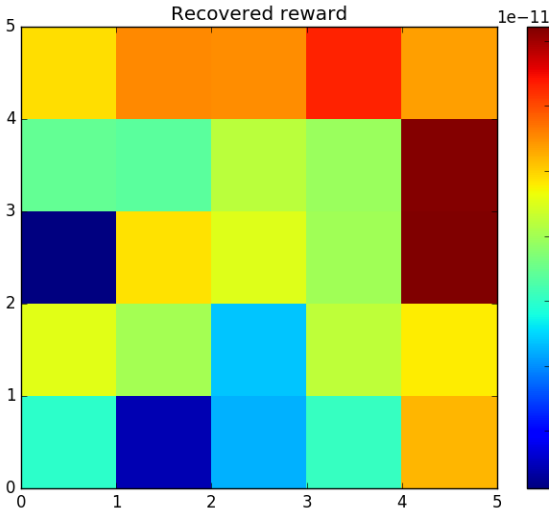
Feature Representation	**Max. Entropy**	**Deep Max. Entropy**
Sigmoid	0.73	0.49
Coordinate	0.01	0.01
Identity	0.87	0.01
Exponential	0.01	0.68
RBF kernel	0.63	0.60

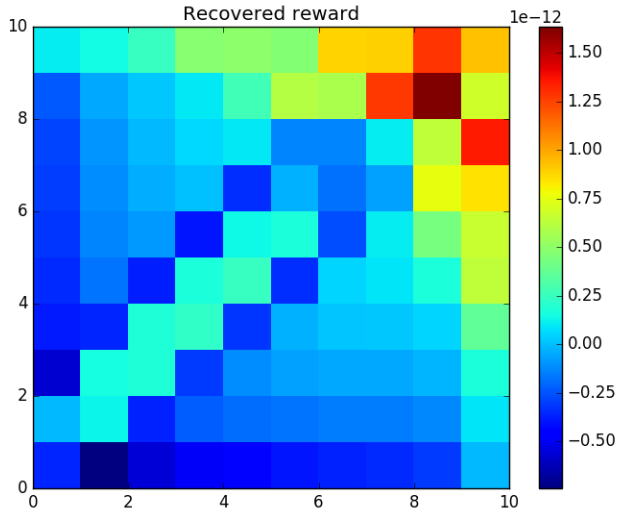




(a) (b)

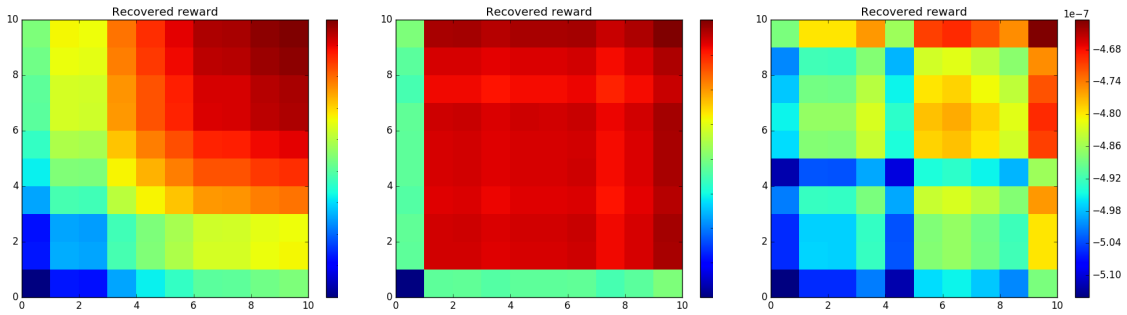
Figure 2: Linear Programming for small state spaces(a) 5X5 grid and (b) 10X10 grid $\gamma = 0.5$





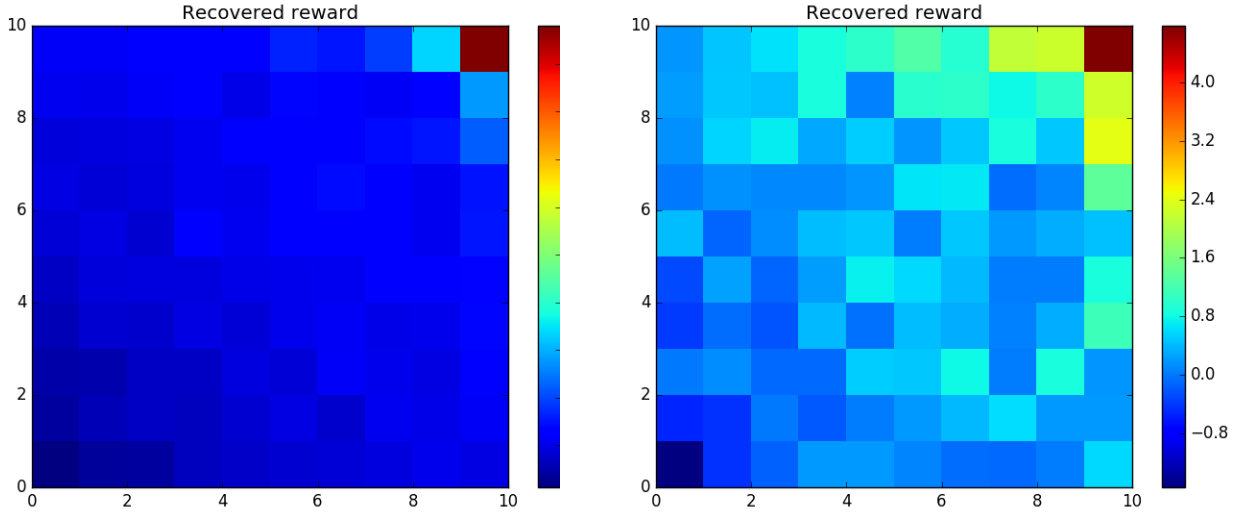
(a) (b)

Figure 3: Linear Programming for small state spaces(a) 5X5 grid and (b) 10X10 grid $\gamma = 0.9$



(a) (b) (c)

Figure 4: Linear Programming for large 10X10 grid at $\gamma = 0.9$ for different feature representation: (a) coordinate, (b) exponential, (c) sigmoid



(a) (b)

Figure 5: Maximum Entropy Model 10X10 grid $\gamma = 0.5$ and $\gamma = 0.9$

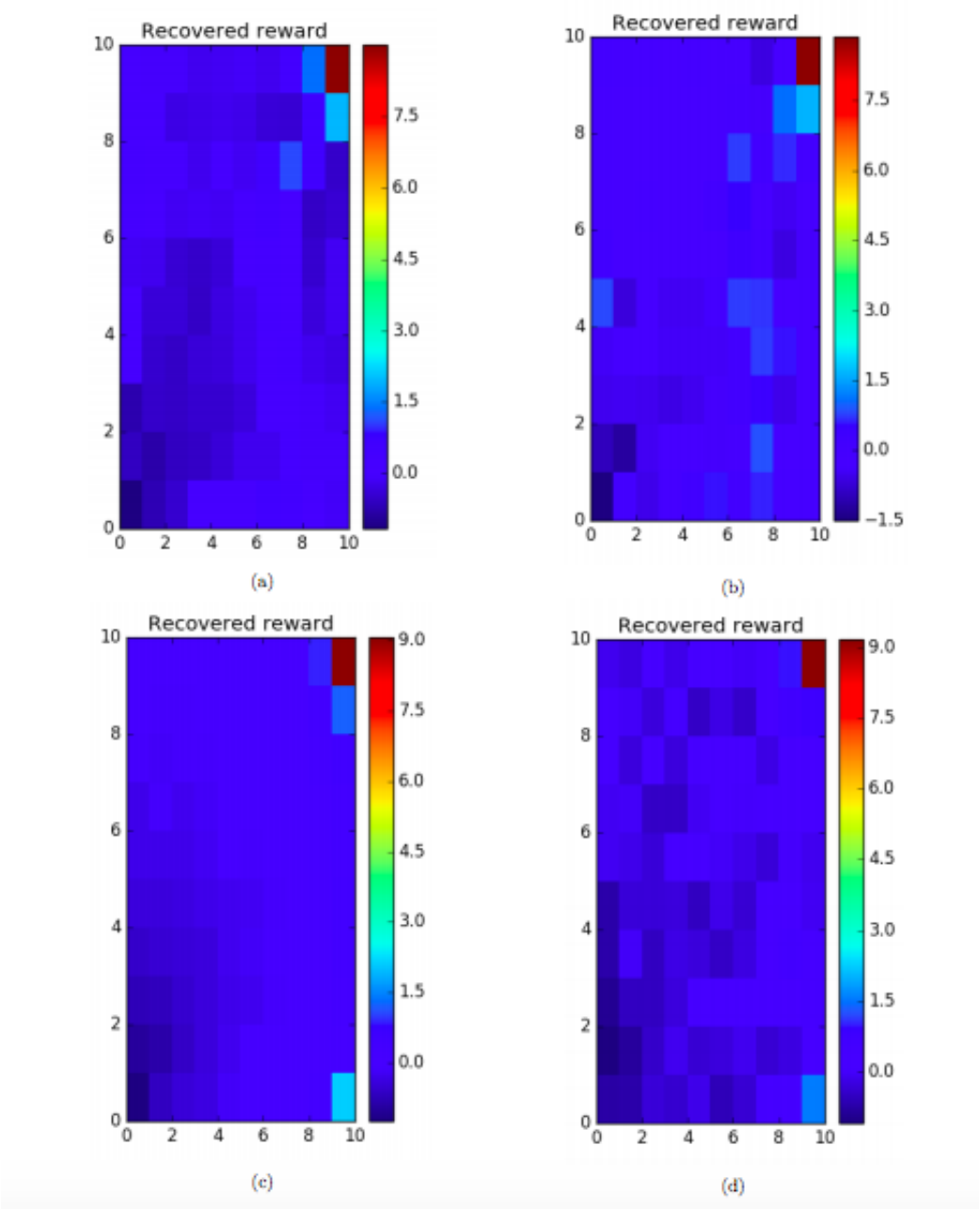
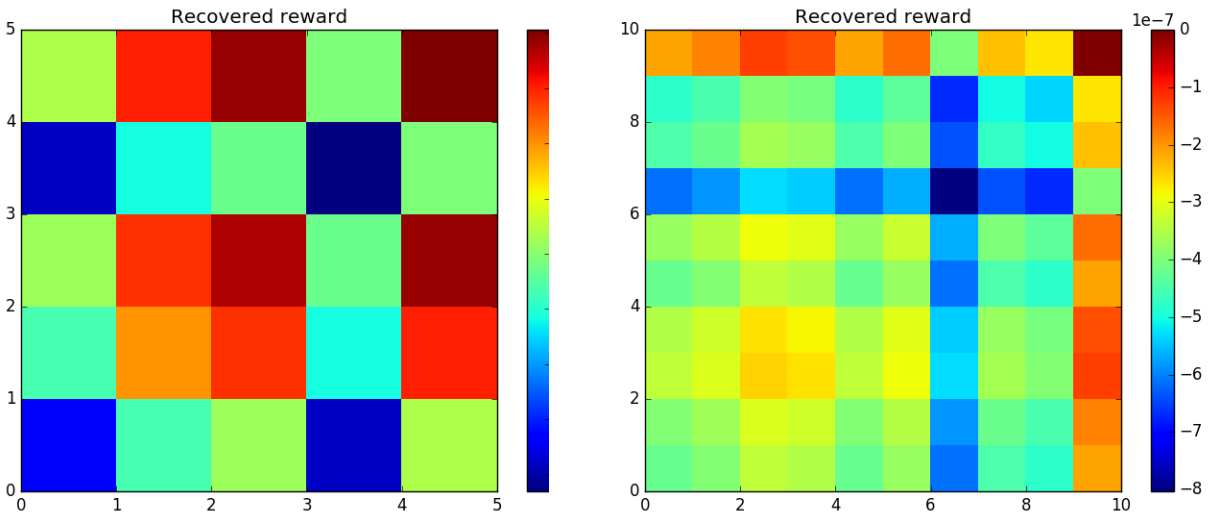


Figure 6: Deep Maximum Entropy (a) Adam at 10 units and 10 layers at $\gamma = 0.9$, (b)Adam at 5 units and 5 layers at $\gamma = 0.9$, (c) $\gamma = 0.5$ at 5 units and 5 layers and (d)shallow network



(a) (b)

Figure 7: Gaussian based non-linear LP 10X10 grid $\gamma = 0.9$

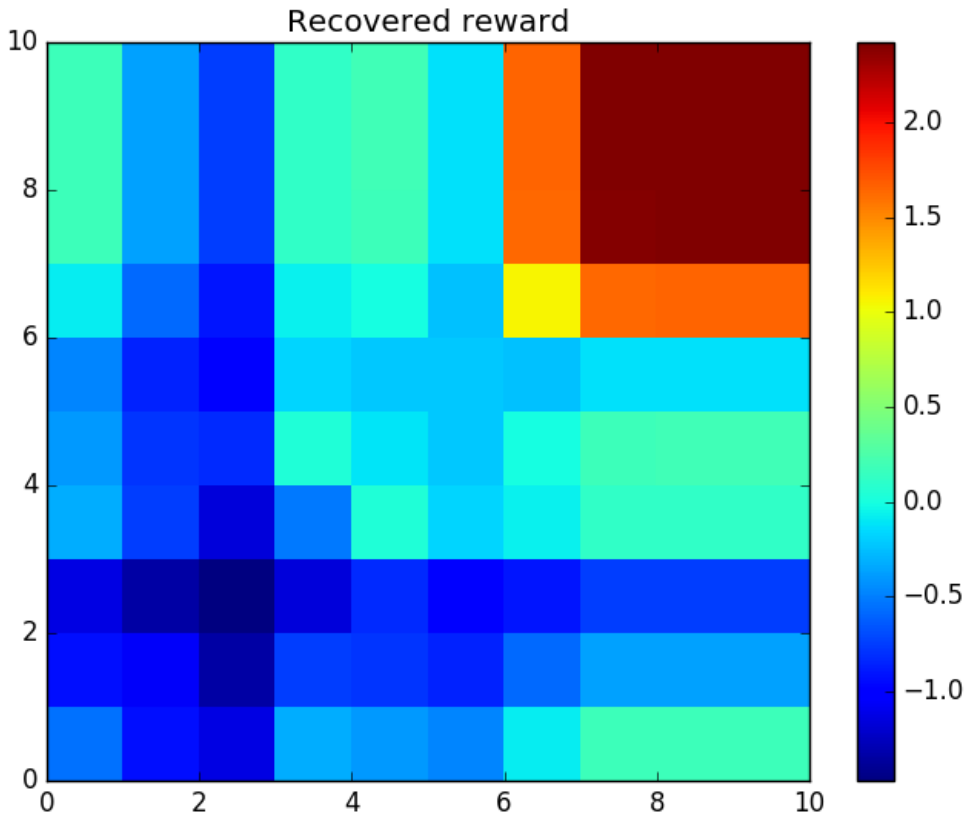


Figure 8: Deep Max Entropy with a Gaussian Kernel Prior

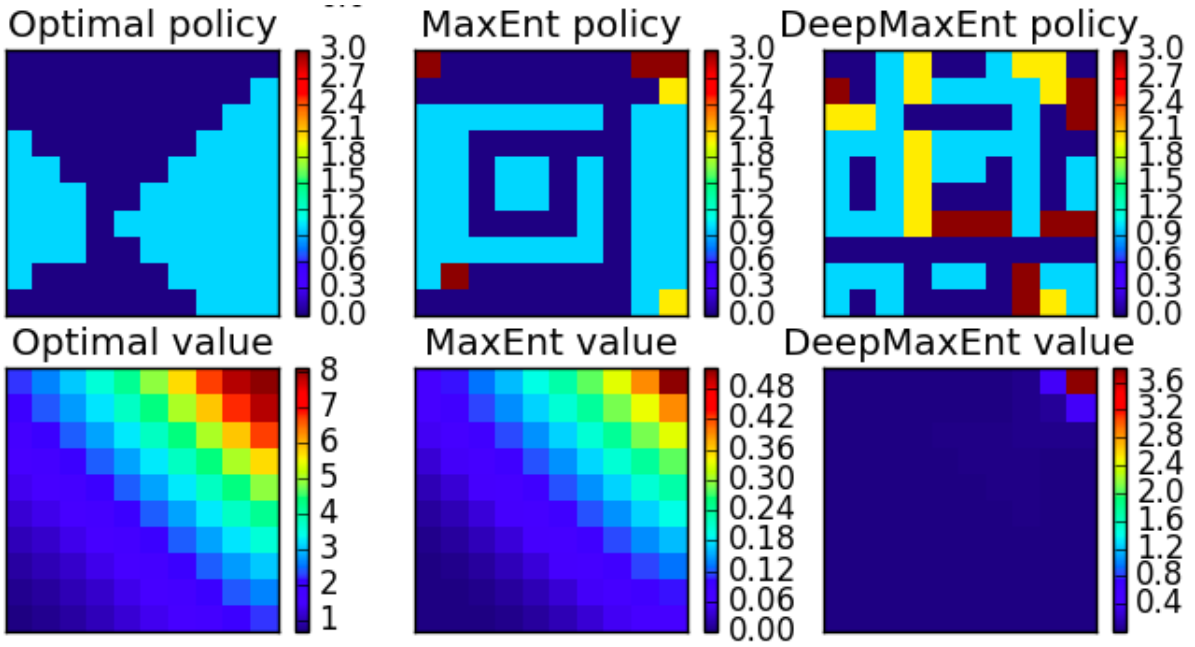


Figure 9: Deep Max Entropy for sigmoid feature

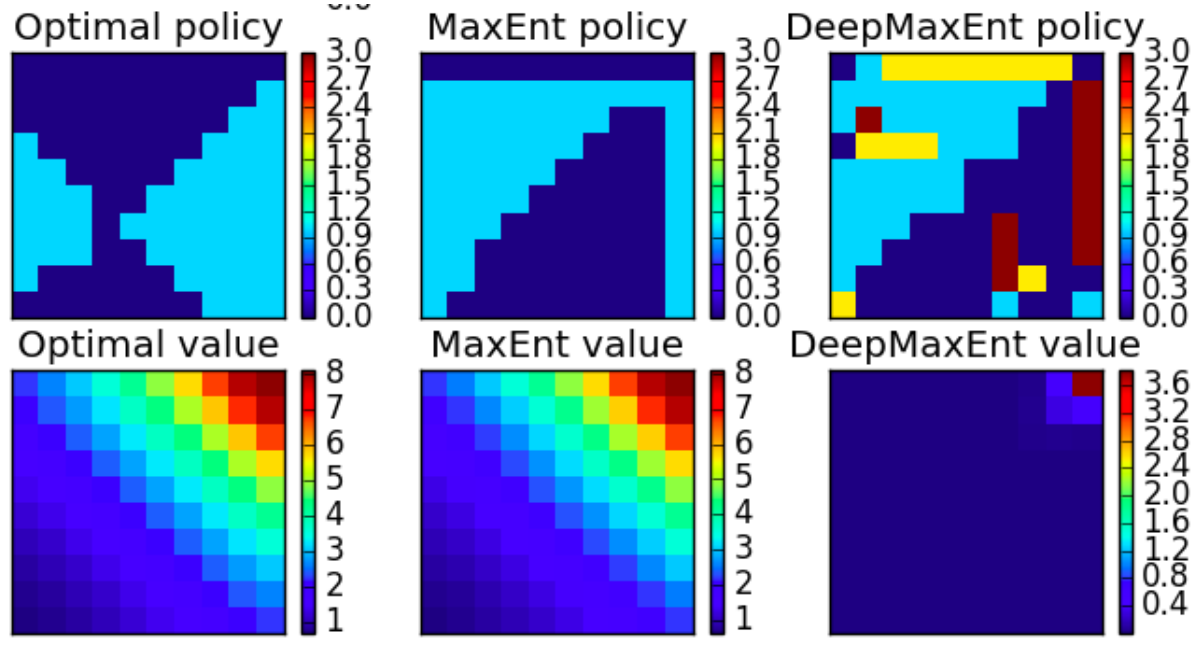


Figure 10: Deep Max Entropy for exponential feature

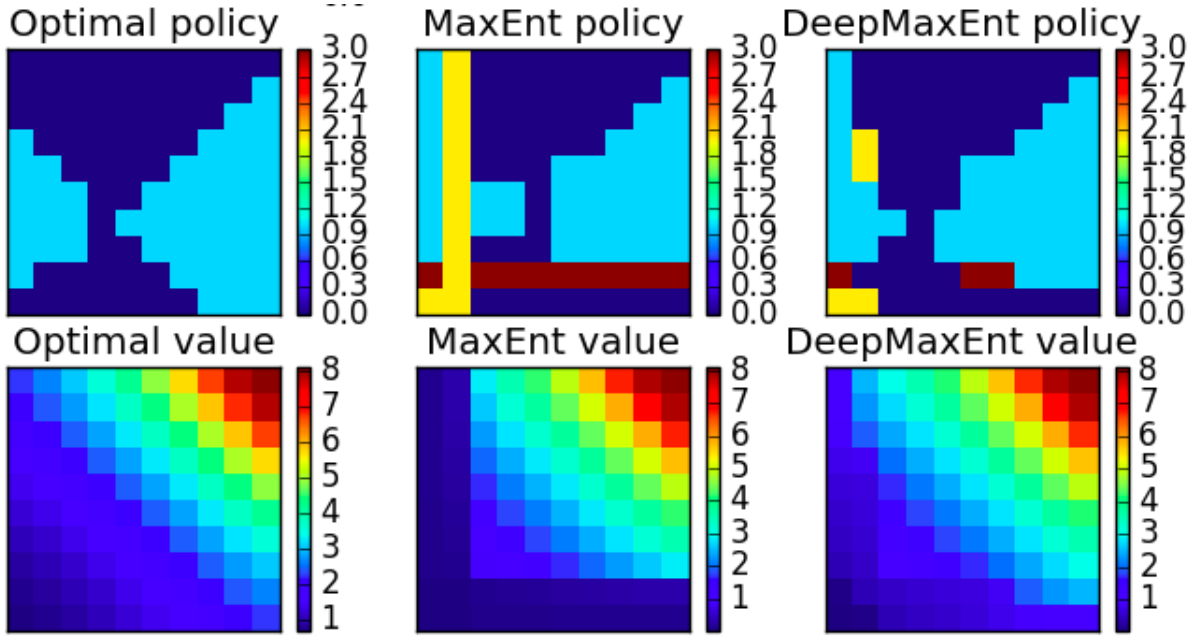


Figure 11: Deep Max Entropy for Gaussian based RBF kernel

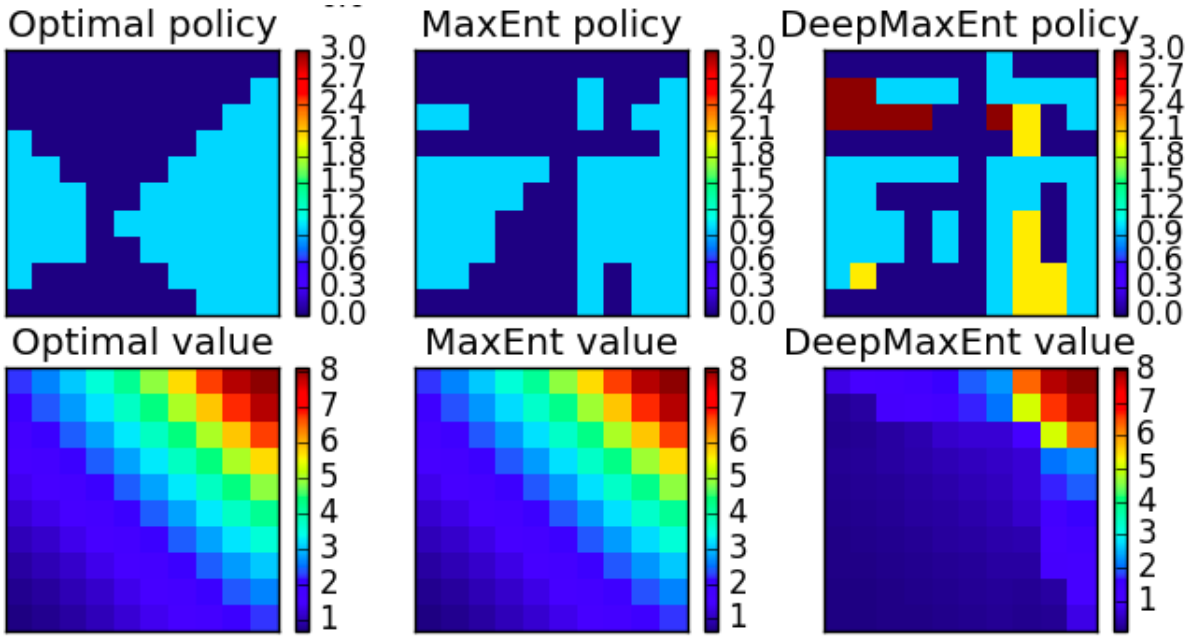


Figure 12: Deep Max Entropy for coordinate feature

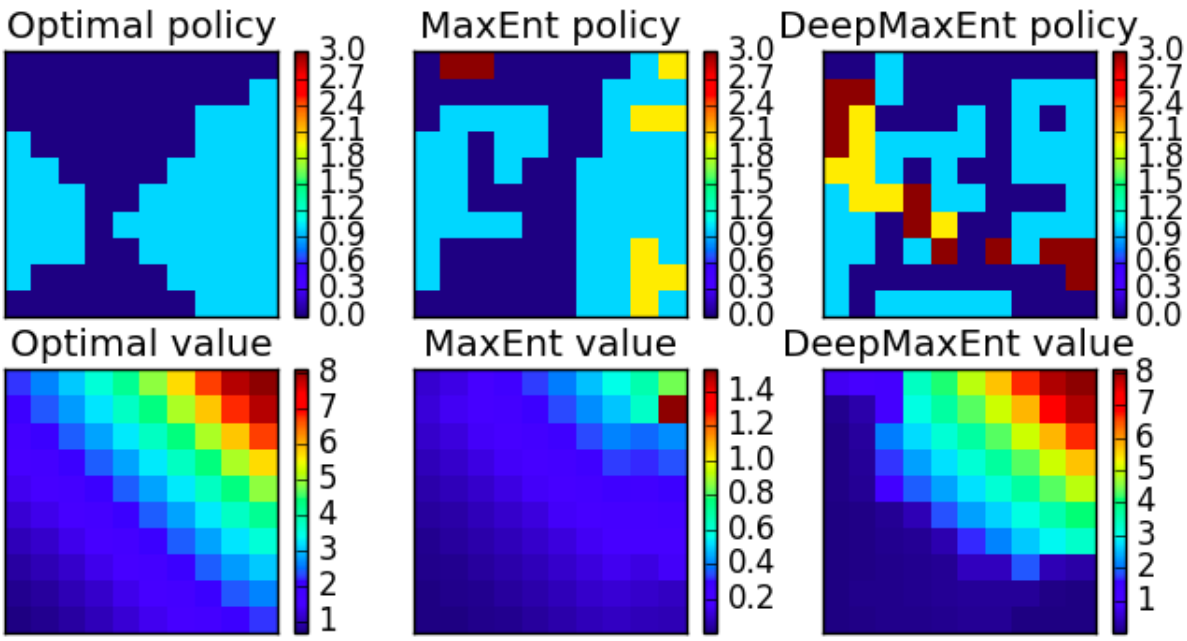


Figure 13: Deep Max Entropy for identity feature