Universität Hamburg
Department Informatik
Knowledge Technology, WTM

Explore Maximum Entropy Inverse Reinforcement Learning

Seminar Paper

Bio-inspired Artificial Inteligence

Mike Trzaska
mike.trzaska@studium.uni-hamburg.de

14.12.2023

Abstract

This paper investigates two actor model architectures employing Ziebart et. al.'s inverse reinforcement learning (IRL) algorithm based on the maximum entropy principle and expert state frequencies for reward estimation [12]. One architecture utilizes q-learning, while the other employs deep q-learning. A reinforcement learning (RL) algorithm with the same model update function is implemented to directly compare the differences between IRL and RL. The performances of the presented algorithms are analyzed for solving the Mountain Car problem. By using expert demonstrations and the maximum entropy principle it was possible to learn a complex spiral structure for the reward estimation. Without using neural networks (NN) for the reward estimation, fast running times are achieved.

Contents

1 Introduction

Supervised learning, i.e. "hard coded" rewards structures, reach their limits for complex applications, since the developer need to by able to hard code all future rewards and need to know the next correct action for every state [**?**]. However, this take a lot of development time [1]. Another possibility is to train the model using reinforcement learning (RL), which is able to learn optimal policies in for complex application Since a RL learner need to perform a large number of interactions with the environment, the training procedure takes a certain amount of time[1]. This disadvantage for RL, is alleviated by learning from expert demonstrations to build up the reward structure in inverse reinforcement learning (IRL)[1]. In the following, reinforcement learning, Markov decision processes (MDPs) and inverse reinforcement learning (IRL) are explained.

Reinforcement Learning In RL the actor learns to map states to actions s. t. a reward signal is maximized. The actor tries to discover the optimal policy π^* of the environment, i.e. which action leads to the most reward in the current state [9].

Markov Decision Process An idealized mathematically representation of the environment in reinforcement learning are MDPs [9]. MDPs are characterized by a 4-tuple of the form $M = \{S, A, T, r\}$, where S represents the state space, A represents the action space, T represents the transition model and r represents the reward function. Besides the MDP an additional factor gamma for discounting future rewards can be considered [11].

Inverse Reinforcement Learning The goal of IRL is to find the hidden reward function, which explains the observed behavior of an expert [8, 2], i.e. the definition of a MDP is known and the reward function is not accessible [11]. An expert provides a set of demonstrations $D = \{\xi_1, ..., \xi_n\}$ or can be sampled from the user policy π. Every $\xi_i \in D$ consists of a set of state and action tuples, i.e. $\xi_i = \{(s_0, a_0), ..., (s_K, a_K)\}$ [11]. Fig. 1 presents the whole workflow of IRL with provided expert demonstrations.

2 State-of-the-Art

This section shortly summarizes the maximum entropy IRL algorithm by Ziebart (MEIRLZ) et. al. and gives a short introduction to the maximum entropy deep IRL algorithm by Wulfmeier (MEDIRLW) et. al.

Maximum Entropy IRL by Ziebart (MEIRLZ) The MEIRLZ algorithm, described in [12] is an increasingly popular approach in the field of IRL [12, 11]. This algorithm is based on the maximum entropy principle [12]. It was initially developed for solving the problem of modeling real-world navigation and driving

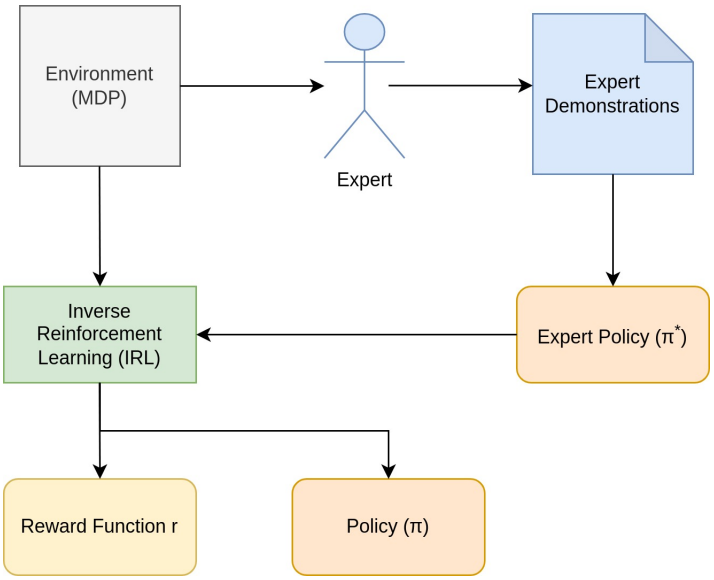


Figure 1: Inverse Reinforcement Learning with Expert Demonstrations.

behaviors [12]. The expert demonstrations used in [12] contains noisy and imperfect data. Ziebart et. al. show that their presented IRL model can handle such suboptimal expert data by using the maximum entropy principle [12, 11]. Eq. 1 was developed in [12] and shows, that for any path between start and goal state, the exponential of the reward along the path is proportional to the probability of expert preference for that path [12, 11]. This equation is the formal description, that state rewards are based on the probability of finding that state in the expert demonstrations.

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

The MEIRLZ algorithm is the basis for all three compared algorithms in this paper.

Maximum Entropy Deep Inverse Reinforcement Learning by Wulfmeier (MEDIRLW) Wulfmeier et. al. developed an IRL algorithm based on the maximum entropy principle from [12] with a reward estimation based on a neural network (NN) [11]. By deriving a gradient from Eq. 1 Wulfmeier et. al. achieved a reward estimation using a fully connected NN, where the running time is independent from the number of expert demonstrations. This algorithm is not further considered in this paper.

3 Implementation

In this paper the experiment is made with a maximum entropy IRL (MEIRL), which is the MEIRLZ algorithm using q-learning for the actor, with a maximum

entropy deep IRL algorithm, which is the MEIRLZ algorithm using deep q-learning and with a maximum entropy deep RL algorithm, which has the same update function for the actor as the MEDRIL algorithm. This section will shortly describe some implementation details of the considered IRL algorithms in this paper.

My project IRLwPytorch[1] contains all the implementations, used for the experiment in this paper. Alg. 1 describes the training procedure for the two IRL algorithms.

Algorithm 1: IRL Training Algorithm

$expert_f = get\_expert\_state\_frequencies()$;
$\theta = N\_states$ random values;
for $e = 0$ **to** $N\_episodes$ **do**
  $state = env\_reset()$
  for $s = 0$ **to** max_steps **do**
    $qvalues = model(state)$;
    $action = select\_action(state)$;
    $next\_state, real\_reward = env\_step(action)$;
    $next\_qvalues = model(next\_state)$;
    $irl\_reward = estimate\_reward(\theta, state)$;
    $update\_model(model, irl\_reward, qvalues, next\_qvalues)$;
  end
  if $e\%10 == 0$ **then**
    $learner_f = calc\_learner\_frequencies()$;
    $max\_ent\_irl(\theta, expert_f, learner_f)$;
    reset learner frequencies;
  end
end

The reward estimation is not done with a NN in this paper. Since the IRL reward estimation is based on the gradient between the state frequencies of an expert and the state frequencies of the learner, the states are discretized in Y different values for every dimension. This results in $\prod_{dim} Y$ different states, which are associated with a corresponding state index. The estimated reward for a state is saved in a vector θ of size $\prod_{dim} Y$ at position equal to the state index. Alg. 2 describes the IRL learning step for the reward estimation.

Algorithm 2: Maximum Entropy IRL Update

$gradient = expert_f - learner_f$;
$\theta = \theta + learning\_rate \cdot gradient$;
for $j = 0$ **to** $len(\theta) - 1$ **do**
  if $\theta[j]$ **then**
    $\theta[j] = 0$;
  end
end

[1] https://github.com/HokageM/IRLwPytorch

Maximum Entropy IRL (MEIRL) Implementation of the MEIRLZ algorithm from [12] is based on the implementation from [6]. It is an IRL algorithm using q-learning with a maximum entropy update function for the IRL reward estimation. In every step, the actor performs the action with the maximum q-value inside the q-table for the current state.

Maximum Entropy Deep Inverse Reinforcement Learning (MEDIRL) The MEDIRL is based on MEIRL but the q-table is replaced with a q-network. In deep q-learning the q-network maps a state to action-q-value pairs, instead of mapping a state-action pair to a q-value [10], i.e. the NN has a number of input nodes equal to the state dimension and a number of output nodes equal to the number of actions. All actions are numerated and the value of the first output node corresponds to the Q-value of the first action. The NN has two fully connected hidden layers, where the first layer contains 64 nodes and the second layer contains 32 nodes. Alg. 3 describes the algorithm, how the next action is selected based on an Epsilon-Greedy algorithm, which introduces randomness in the beginning of the learning process [5]. This randomness results in faster and better learning and prevents stucking in local minima [5]. The q-network takes continuous state values as input and the irl reward estimation takes discrete state values as input.

Algorithm 3: Action Selection MEDIRL and MEDRL

Result: Next Action

x = random value in $[0, 1]$;
if $x < \epsilon$ **then**
  return random Action;
else
  return Action corresponding to $\max\{q\_network.forward(state)\}$;
end
$\epsilon = \min\{\epsilon \cdot d, \epsilon_{min}\}$, with $d = 0.95$;

Maximum Entropy Deep Reinforcement Learning (MEDRL) MEDRL is a RL implementation of the MEDIRL algorithm. This algorithm gets the real rewards directly from the environment, instead of estimating IRL rewards, which is achieved by replacing *irl_reward* with *reward* in line 11 in Alg. 1. The NN architecture and action selection is the same as in MEDIRL.

4 Experiment

This section presents the experiment used for evaluating the performances of the algorithms, described in the sections above. Moreover, the evaluation is presented in the end of this section.

Mountain Car The deterministic MDP of the Mountain Car (MC) problem was first described in [7]. MC consists of a sinusoidal track where, a car is placed randomly at the bottom of this track. Possible actions for an actor are accelerate the car to the left, accelerate the car to the right and no acceleration. Reaching the top of the track on the right is the goal of this problem, therefore the car needs to be strategical accelerated [7, 4]. Fig. 2 describes the MC problem.

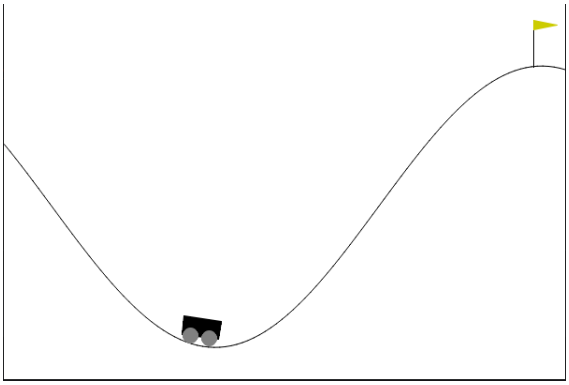


Figure 2: Mountain Car problem visualization.

This paper focuses on the discrete action version of the MC, which means that the action space is of dimension one and $\forall action \in [0, 1, 2]$, where $[0, 1, 2]$ represent the three actions mentioned above. MC has a two dimensional continuous state space consisting of the velocity of the car and its position on the x-axis. The transition from one state to the next state is using an action is done in two steps. First, the next velocity $v_{t+1} \in [-0.07, 0.07]$ is calculated with the formula in Eq. 2, where a denotes the action, f is 0.001 and g is 0.0025, after that the next position $p_{t+1} \in [-1.2, 0.6]$ is calculated with the formula in Eq. 3. Each time step gets an reward of -1, Since the goal is to reach the top as fast as possible. When the car reaches the goal an episode ends [7, 4].

$$v_{t+1} = v_t + (a - 1) \cdot f - cos(3 \cdot p_t) \cdot g \tag{2}$$

$$p_{t+1} = p_t + v_{t+1} \tag{3}$$

An environment of the Mountain Car is implemented by the open source python library Gym from OpenAI. Gym provides an API for evaluating RL algorithms [3] and can also be used in a similar way for benchmarking IRL algorithms.

Expert Demonstrations This paper uses the expert demonstrations for the MC provided by the open source project [6]. Continuous position and velocity tuples from the expert demonstrations are indexed s.t. there are X position indices and X velocity indices, i.e. continuous states are indexed to X^2 state indices. The expert state frequencies for $X = 20$ are presented by Fig. 3.

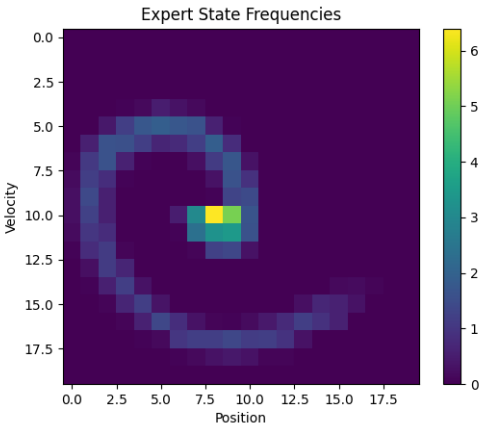


Figure 3: Heatmap of expert state frequencies with 400 states.

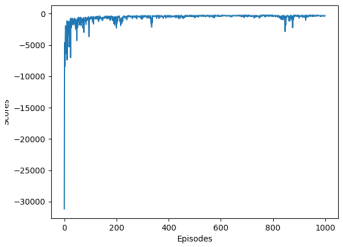
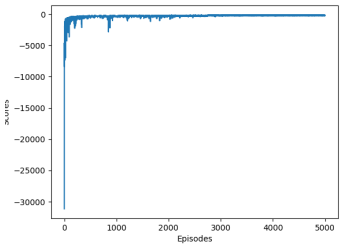
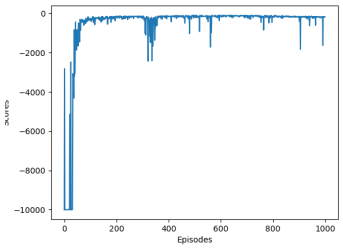
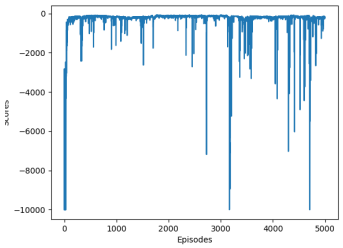
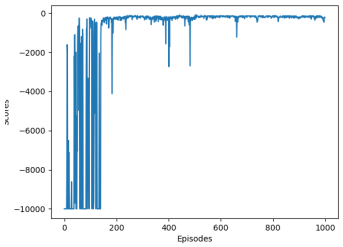
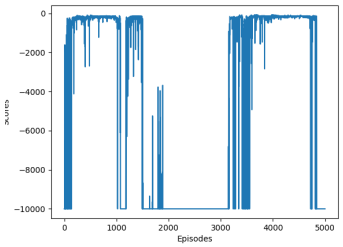
Setup Every algorithm trains its model for 5000 episodes, additionally the maximum entropy IRL algorithm is trained for 14000 to highlight the development of the estimated reward function. Because of long running times the maximum number of steps inside an episode is set to 10000 for the maximum entropy deep IRL algorithm and the maximum entropy deep RL algorithm. However, the maximum number of steps inside an episode is unlimited for the maximum entropy IRL algorithm.

For the two deep algorithms testing is done with the model that achieves the best score during training and the model that is build up after 5000 episodes for the maximum entropy IRL algorithm

Evaluation Table 1 shows the learning curves for the three algorithms after 1000 and 5000 episodes. The IRL algorithms MEIRL and MEDIRL learn much faster than the RL algorithm MEDRL, this results from the fact that the RL algorithm need to reach the goal a couple times at random before it can start learning. All three algorithms achieve a score in range of $[-100, -250]$ after 1000 episodes of training. The MEIRL algorithm converges to a score in range of $[-120, -200]$ for training more episodes. After training with the MEDIRL 5000 episodes a best model with a score of -83 is achieved, but stable good performing intervals are repeatedly followed by local swing outs to a bad score. Training 5000 episodes with the MEDRL algorithm results in the observation of short good performing intervals followed by long bad performing intervals, this highlights the instability of this algorithm. However, the best model trained with MEDRL achieves a score of -84.

Table 2 presents the learner state frequencies for the last 10 episodes after 1000, 2000 and 5000 episodes. Comparing those results with Fig. 3 shows, that all the two IRL algorithms try to approximate the spiral given by the expert demonstrations. Since, the MEIRL is based on a Q-table the distribution of the learner state frequencies is smaller than for MEDIRL, which is based on a Q-

Table 1: Learning Curves for different Algorithms

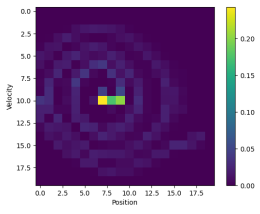
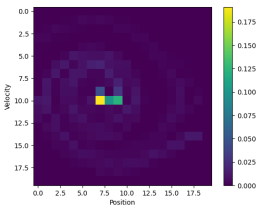
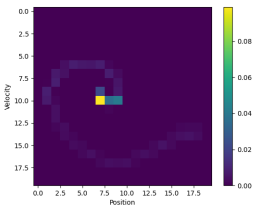
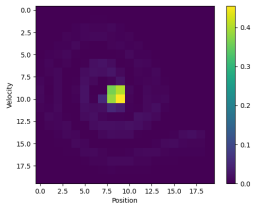
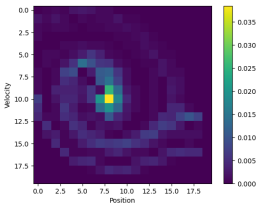
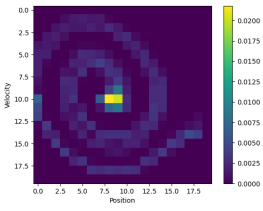
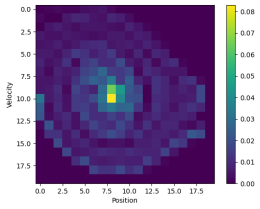
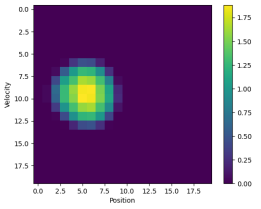
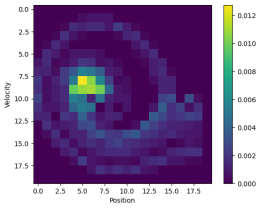
Algorithm	Learning Curve After 1000 Episodes	Learning Curve After 5000 Episodes
Maximum Entropy IRL		
Maximum Entropy Deep IRL		
Maximum Entropy Deep RL		

network. However, the learner state frequencies for the RL algorithm also result in a spiral structure, but is more instable over the training episodes.

Table 3 shows the estimated reward function for the two IRL algorithms. Both algorithms build up a spiral looking like reward structure. States, which are visited often by the learner than by the expert, are getting a negative reward. That is why the reward structure gets faster clearer for the MEDIRL, which comes from the fact that the MEDIRL explore the state space more in the beginning of the training process.

Table 4 presents the testing results for testing the resulting models for 100 test runs. Testing results are stable for the MEIRL, where the results are in the interval $[-125, -300]$ and MEDIRL, where the results are in the interval $[-85, -105]$. The

Table 2: Learner State Frequencies

Algorithm	Learner State Frequencies After 1000 Episodes	Learner State Frequencies After 2000 Episodes	Learner State Frequencies After 5000 Episodes
Maximum Entropy IRL			
Maximum Entropy Deep IRL			
Maximum Entropy Deep RL			

best model for the MERL algorithm is extreme instable. Some testing results are in the interval $[-85, 110]$ and every other testing result is < -10000.

5 Conclusion

The maximum entropy update function for the IRL reward estimation is stable and leads the learner's state frequencies to converge to an expert's state frequencies. A NN can learn the complex spiral structure with the MEDIRL algorithm. Using MEDIRL instead of MEIRL leads to more exploration of the state space, identifying states that an expert does not explore much faster. IRL directs the RL training method to a solution provided by an expert, removing the need for complete random searching for the goal. This results in much faster learning and shorter running times. That is why IRL could be interesting to consider for more

Table 3: Estimated Reward Functions

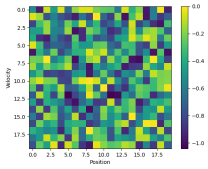
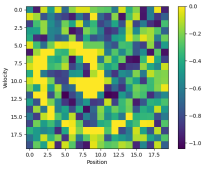
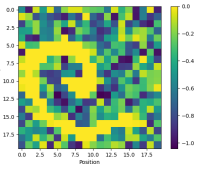
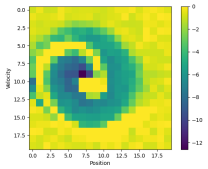
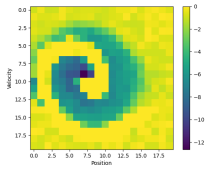
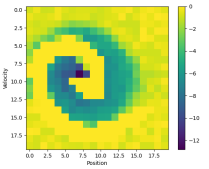
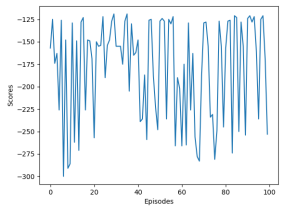
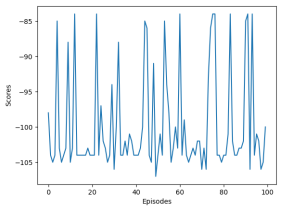
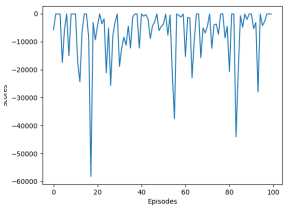
Algorithm	IRL Rewards After 1000 Episodes	IRL Rewards After 2000 Episodes	IRL Rewards After 5000 Episodes	IRL Rewards After 14000 Episodes
Maximum Entropy IRL		None		
Maximum Entropy Deep IRL				None
Maximum Entropy Deep RL	None	None	None	None

Table 4: Learning Curves for different Algorithms

	Maximum Entropy IRL	Maximum Entropy Deep IRL	Maximum Entropy Deep RL
Testing for 100 Runs			

complex experimental environments where RL has extremely high running times. However, one cannot completely generalize the performance of other RL algorithms based on MEDRL. In the future, exploring the performance of IRL algorithms on different experimental environments could reveal possible properties of the IRL algorithms. One possible environment could involve finding the right path inside a maze. Another interesting question is how IRL algorithms perform on expert demonstrations that are far from the optimal solution and how the algorithms could be modified to create much better solutions than the expert demonstrations.

References

[1] Stephen Adams, Tyler Cody, and Peter A Beling. A survey of inverse reinforcement learning. *Artificial Intelligence Review*, 55(6):4307–4346, 2022.

[2] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

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

[4] Gym. Mountain car. `https://www.gymlibrary.dev/environments/classic_control/mountain_car/`, 2021.

[5] N Hariharan and Anand G Paavai. A brief study of deep reinforcement learning with epsilon-greedy exploration. *International Journal of Computing and Digital Systems*, 11(1):541–552, 2022.

[6] Reinforcement Learning KR. lets-do-irl. `https://github.com/reinforcement-learning-kr/lets-do-irl`, 2021.

[7] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

[8] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[9] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[10] Mike Wang. Deep Q-Learning Tutorial: minDQN — towardsdatascience.com. `https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc`. [Accessed 07-12-2023].

[11] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning, 2016.

[12] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.