Inverse Reinforcement Learning Application for Discrete and Continuous Environments

Yeison Suarez[1], Carolina Higuera[1], and Edgar Camilo Camacho[1]

Universidad Santo Tomás, Colombia
{yeisonsuarez, carolinahiguera, edgarcamacho}@usantotomas.edu.co

Abstract. We show the application of inverse reinforcement learning (IRL) in discrete and continuous environments based on the apprenticeship focus. The objective is to learn a mathematical definition of a task based on trajectories made by an expert agent. To achieve this, there must be features functions that in some way describe abilities that the agent can learn. Therefore, the description of a task can be formulated as a linear combination of those functions. This learning method was applied in Open-AI gym environments to show the learning process of a task using demonstrations.

Keywords: inverse reinforcement learning, reinforcement learning, quadratic programming

1 Introduction

Reinforcement Learning (RL) is one of the best known algorithms for autonomous learning. It uses the agent's positive and negative experiences to learn how to behave in order to solve a defined task. However, the information of what is the task must be included in the rewards given to the agent.

Although RL has shown successful results, it highly depends on an established reward function. But, in some cases, designing the reward function can become more complex than the learning itself [1]. On the other hand, if we have an expert agent, it is possible to exploit its knowledge to infer the reward through an Inverse Reinforcement Learning (IRL).

Learning the reward function instead of the policy directly has some advantages, as shown in [7]. First, with the reward function it is possible to understand the purpose of the expert agent. Second, the reward is adapted to disturbances and is transferable to other environments, while with policies the agent can learn actions that are not relevant to the task, for instance, errors in the demonstrations. Third, the policy learned after inferring the reward function allows knowledge transfer for more complex tasks.

2 Theoretical background

2.1 Reinforcement Learning

The RL problem can be described as a Markov Decision Process (MDP), where for each time step t, the agent receives the state of its environment s_t and takes an actions in order to learn how to perform a task. As feedback, the agent receives a numeric reward r_{t+1} and moves to a new state s_{t+1} [9].

The agent should learn the best way to perform the task, which is reflected in the rewards that it gets every time it makes an action. For this reason, the agent has to learn to maximize the rewards in the long-term. To do this, the agent maps, with the function $Q(s_t, a_t)$, the expected discounted utility of taking action a_t in state s_t if he behaves optimally thereafter. From the Q-value, the agent can generate a policy $\pi_t(s, a)$ that indicates the probability of applying action a_t when it is in state s_t that assures the maximum long-term reward.

Q-learning is widely use as RL algorithm. It updates the Q-values according to greedy actions, without taking into account the current policy, which may include exploratory actions. The updating rule for each Q-value is:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[r + \gamma \max_{a \in A} Q(s_{t+1}, a)]$$

Where:

- r: reward at time step t ($r \in \mathbb{R}$)
- α: learning rate at time step t ($0 \leq \alpha \leq 1$)
- γ: discounting rate ($0 \leq \gamma < 1$)

2.2 Inverse Reinforcement Learning

According to Abbeel and Ng in [1], since the RL field is based on the fact that the reward function is the most clearly, robustly and transferable definition of the task, it makes sense to consider an RL approach in which the reward function is what has to be learned. The problem of deriving the reward function from observations is known as Inverse Reinforcement Learning - IRL [6].

IRL assumes that an expert already optimized a reward function $R(t)$ that is unknown for the learner agent, which can be expressed as a linear combination of known features ϕ_i describing possible subtasks or skills to be learned:

$$r(s) = w_1\phi_1(s) + w_2\phi_2(s) + \cdots + w_n\phi_n(s) = \boldsymbol{w^T \phi(s)}$$

Because the expert knows the truly reward function $\boldsymbol{w^{*T} \phi(s)}$, with $\|\boldsymbol{w^*}\|_1 \leq 1$ for rewards bounded by 1, the utility expectation of the expert policy is:

$$U^{\pi_E}(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s) \middle| \pi_E\right] = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{w^{*T} \phi(s)} \middle| \pi_E\right]$$
$$= \boldsymbol{w^{*T}} \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi(s)} \middle| \pi_E\right] = \boldsymbol{w^{*T}} \mu(\pi_E)$$

Where, $\mu(\pi_E)$ is known as *feature expectations* given policy π_E. For the case of the learning agent, its utility can be express as:

$$U^{\tilde{\pi}}(s) = \boldsymbol{w^T}\mu(\tilde{\pi})$$

The objective of IRL is to find values for the weights $w_1, w_2, \ldots, w_n$ such that the utility of the policy learned by the agent $\tilde{\pi}$ is close to the utility of the expert policy π_E:

$$\|\mu(\tilde{\pi}) - \mu(\pi_E)\|_2 \leq \epsilon$$

This means, IRL is seeking for a reward function $R = \boldsymbol{w^T\phi(s)}$, for which the expert policy performs better than the learner by a margin: $U^{\pi_E}(s) \geq U^{\tilde{\pi}}(s) + \Delta$. In terms of feature expectations:

$$|\boldsymbol{w^T}\mu(\tilde{\pi}) - \boldsymbol{w^{*^T}}\mu(\pi_E)| \leq \Delta$$

And, due to we want to approximate $\mathbf{w}$ to $\mathbf{w}^*$:

$$\mathbf{w^T}|\mu(\tilde{\pi}) - \mu(\pi_E)| \leq \Delta \tag{1}$$

Because we are looking for a margin Δ, the IRL problem is similar to the one solved in Support Vector Machines (SVMs), for finding the maximum margin hyperplane separating two set of points [1]. This similarity can be view taking the expert feature expectations as class +1 and the others as class -1, as show in Figure 1.

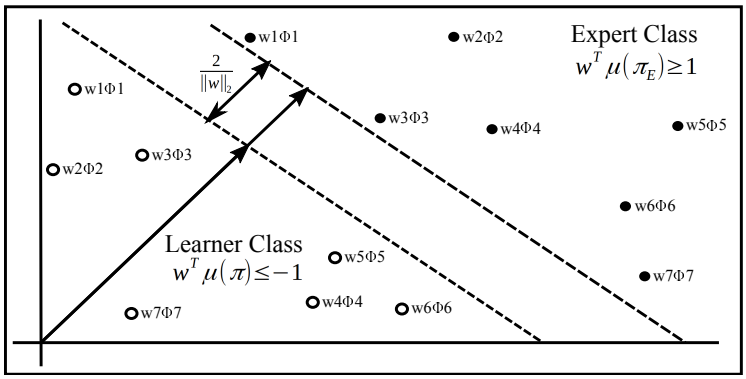

Fig. 1. IRL problem as SVM classification

The distance of the vector from the origin to the class +1 hyperplane is $1/\|\mathbf{w}\|_2$ and the distance of the vector from the origin to the class -1 hyperplane is $-1/\|\mathbf{w}\|_2$. Therefore, the margin size is:

$$\frac{1}{\|\mathbf{w}\|_2} - \frac{-1}{\|\mathbf{w}\|_2} = \frac{2}{\|\mathbf{w}\|_2}$$

To maximize the margin, we have to minimize the weights, resulting the following quadratic programming problem:

$$\min \frac{1}{2}\|\mathbf{w}\|_2^2$$

Regarding the constraints, they are related with equation 1. Due to the expert policy has to do it better that the learner by a margin:

$$\mathbf{w^T}|\mu(\tilde{\pi}) - \mu(\pi_E)| \leq \frac{2}{\|\mathbf{w}\|_2}$$

And, because bounded rewards by 1, we have $\|\mathbf{w}\|_2 \leq \|\mathbf{w}\|_1 \leq 1$.

The complete quadratic programming problem is formulated as:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & \mathbf{w^T}\left[\mu(\pi_E) - \mu(\tilde{\pi})\right] \geq -2 \end{aligned}$$

In Algorithm 1 we summarize the process for approximating $\mathbf{w}$ to $\mathbf{w}^*$ through quadratic programming and the feature expectations from both learner and expert.

Algorithm 1: weightsUpdate

Result: $\mathbf{w}$ and optimizer status:= {feasible, infeasible}

1 **Requirements:** *Learner* $\mu(\tilde{\pi})$ and *Expert* $\mu(\pi_E)$ feature expectations, convex optimization solver;

2 ObjectiveFunction = $\min \frac{1}{2}\|\mathbf{w}\|_2^2$;

3 Constraints = $\mathbf{w^T}\left[\mu(\pi_E) - \mu(\tilde{\pi})\right] \geq -2$;

4 $\mathbf{w}, status$ =Optimizer.solve(ObjectiveFunction, Constraints);

3 Framework: a discrete environment problem

We used the Open-AI gym suite due to it provides easy to use environments to test reinforcement learning algorithms. Specifically, we applied IRL to the Taxi-v2 task, which was introduced in [5] to illustrate some issues in hierarchical RL. The goal in the task is to pick-up and drop off a passenger in a specific location.

The environment is a 5×5 grid with five walls, five possible positions for the passenger (R,G,B,Y and inside taxi) and four possible positions for the destination (R,G,B,Y). The taxi, which is the learner agent, has 6 movements: up, down, right, left, pickup passenger and drop-off passenger. The total number of states for the environment is 500, divided in taxi row (5), taxi column (5), passenger location (5) and destination (4).

The episode starts with a random location for the agent (yellow if empty, green if full), and ends when the agent carries the passenger (blue) from his location until the desired destination (magenta). Figure 2 shows the graphical interface of the environment.

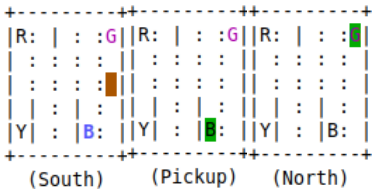

Fig. 2. Taxi-v2 Environment

The gym environment gives a reward of -1 for every time step. When the agent pickup correctly the passenger from his location to destination it gives a reward of +20 and, for every pickup and drop-off illegally, the reward is -10.

In order to apply IRL to learn the implicit reward function, we extracted 20 trajectories doing a manual control of the taxi. Furthermore, we proposed two types of convex functions as features, based on the distance between the taxi to the passenger when the taxi is free, the distance between the taxi to the destination when the taxi has a passenger and the time the route takes.

Features option 1:

Passenger: $\phi_1 = 1 - \left[\frac{1}{2}\exp\left(\frac{-(taxi_x - pass_x)^2}{2}\right) + \frac{1}{2}\exp\left(\frac{-(taxi_y - pass_y)^2}{2}\right)\right]$

Destination: $\phi_2 = 1 - \left[\frac{1}{2}\exp\left(\frac{-(taxi_x - dest_x)^2}{2}\right) + \frac{1}{2}\exp\left(\frac{-(taxi_y - dest_y)^2}{2}\right)\right]$

Time spent: $\phi_3 = 1 - \exp\left(\frac{-steps^2}{2}\right)$

Features option 2:

Passenger: $\phi_1 = 1 - \text{manhattanDistance}\,(taxi, pass)^2$

Destination: $\phi_2 = 1 - \text{manhattanDistance}\,(taxi, dest)^2$

Time spent: $\phi_3 = -steps^2$

To run IRL, we follow the work done by Abbeel and Ng in [1] called *Apprenticeship learning via IRL*. The procedure is shown in algorithm 2. We code it in Python and we use the package CVXPY as quadratic optimization solver [2, 4]. We train the IRL for the taxi environment for 60000 episodes, with a discounting rate of 0,99 and a learning rate of 0,03. Those values were set experimentally.

Figure 3 shows the cumulative reward during each episode, given by the gym environment during the IRL training. It shows that with features option 2 the apprenticeship is slower in comparison to features option 1. However, both options converge to the same maximum cumulative reward. This means that both make the agent to learn policies with the same long-term utility.

In addition, the weights given by the IRL algorithm to each feature in the reward function are:

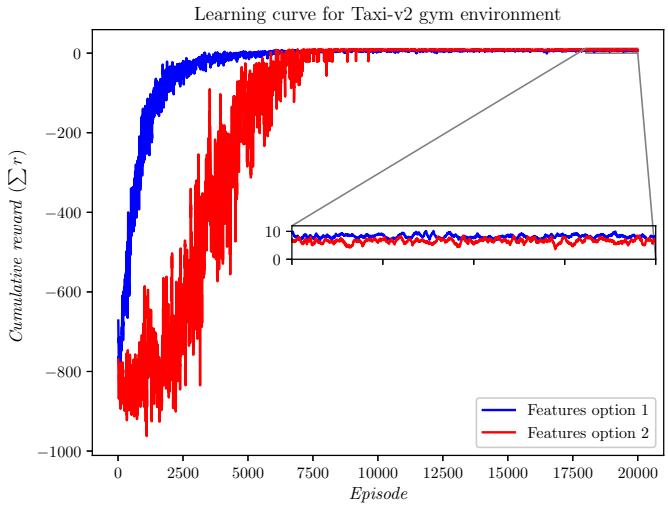

Fig. 3. Learning curve for the Taxi-v2 gym environment

Features option 1: $r(s) = -3,908 \times 10^{-3}\phi_1(s) - 3,908 \times 10^{-3}\phi_2(s) - 3,908 \times 10^{-3}\phi_3(s)$

Features option 2: $r(s) = -1,65 \times 10^{-3}\phi_1(s) - 35,1 \times 10^{-3}\phi_2(s) + 11,4 \times 10^{-3}\phi_3(s)$

Algorithm 2: IRL for discrete environment

Requirements: expert trajectories, features $\boldsymbol{\phi}$, discount factor γ, learning rate α, exploration rate ϵ, episodes;
Initialize weights $\mathbf{w}$;
Calculate feature expectations $\mu(\pi_E)$ from expert trajectories;
for *episode in episodes* **do**
  Initialize gym environment;
  Get current state s_t;
  while *not done* **do**
    Choose action a from policy $\tilde{\pi}$ derived from tabular $Q(s,)$ with ϵ-*greedy* exploration;
    Take action a and observe s_{t+1} ;
    Calculate reward $r = \mathbf{w^T}\boldsymbol{\phi}(\boldsymbol{s_{t+1}})$;
    Update policy with Q-Learning: $Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha[r + \gamma \max_{a' \in A} Q(s_{t+1}, a')]$;
    Update $s_t \leftarrow s_{t+1}$;
    if *is time to update weights* **then**
      Save and append learner trajectory;
      Calculate feature expectations $\mu(\tilde{\pi})$ from learner trajectories;
      do
        $\mathbf{w}, status$ = weightsUpdate($\mu(\tilde{\pi})$, $\mu(\pi_E)$);
        if *status=infeasible* **then**
          Remove last learner feature expectation;
      while *status=infeasible*;
    Check environment done;
  end
end

Notice that the definition for features option 1 rewards the agent if the taxi moves away from the passenger and the destination. However, the sign of the weights makes the inferred reward function to have the same purpose of the expert agent, which is to move closer to both, passenger and destination. At testing, the average cumulative reward of the learner is 7.4, while the expert wins, in average, 8.25 in each demonstration. Figure 4 shows followed by the expert and the learner agent with features option 1 in the same environment. Although both agents follows different trajectories, they receive the same reward at the end of the episode, which is 9, +20 for legal drop-off and -11 for the eleven steps.

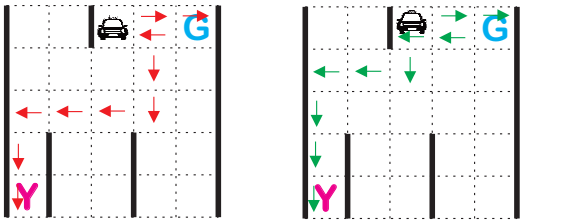

Fig. 4. Policies followed by expert and learner agents in he Taxi-v2 environment

4 Framework: a continuous environment problem

The cartpole [3] is a classic control problem in the Open-AI gym suite. It consists of a pole that is attached by a joint to a cart. The cart can move without friction while the pendulum remains upright. The episode ends when the pole angle exceeds ±12 degrees or the cart moves more than ±2.4 units from the center.

The state is conformed by four variables, as shown in Table 1. Because the ranges for each observation are continuous, we have to run Q-learning inside the IRL algorithm with a neural network instead of the tabular form. Also, the agent can apply two actions: 0 to push the car to the left and 1 to the right.

Table 1. States in the cartpole problem [3]

	Cart Position	*Cart Velocity*	*Pole Angle*	*Pole Velocity at Tip*
Min	-4,8	$-\infty$	$-24°$	$-\infty$
Max	+4,8	∞	$+24°$	∞

The expert trajectories and utilities were extracted from a policy π_E trained with neural Q-Learning. To infer the expert task, we proposed four features, one for each observation in the state.

Cart Position:	$\phi_1(s) = \exp\left(-\frac{pos^2}{2\times 1{,}5^2}\right)$	Pole Angle:	$\phi_3(s) = \exp\left(-\frac{angle^2}{2\times 6^2}\right)$
Cart Velocity:	$\phi_2(s) = \exp\left(-\frac{vel^2}{2\times 0{,}5^2}\right)$	Pole Velocity at Tip:	$\phi_4(s) = \exp\left(-\frac{tip^2}{2\times 0{,}5^2}\right)$

The IRL procedure for a continuous problem is presented in algorithm 3 and 4. Our Q neural network has two layers with 24 units and relu activations. Because we want to approximate the Q-values for each pair (s,a), we are facing a regression problem. Therefore, our loss function was Mean Squared Error. In order to help the neural network to remember earlier trajectories, we use a memory buffer with 50000 samples and a batch size of 64.

Algorithm 3: IRL for discrete environment

Requirements: expert trajectories, features $\boldsymbol{\phi}$, discount factor γ, learning rate α, exploration rate ϵ, episodes;
Initialize weights $\mathbf{w}$;
Initialize Q neural network and memory;
Calculate feature expectations $\mu(\pi_E)$ from expert trajectories;
for *episode in episodes* **do**
    Initialize gym environment;
    Get current state s_t;
    while *not done* **do**
        Choose action a from policy $\tilde{\pi}$ derived from tabular $Q(s,)$ with ϵ-*greedy* exploration;
        Take action a and observe s_{t+1} ;
        Calculate reward $r = \mathbf{w^T}\boldsymbol{\phi}(\boldsymbol{s_{t+1}})$;
        Add trajectory step to memory (s_t, a_t, r, s_{t+1});
        Call experienceReplay() to update policy $\tilde{\pi}$;
        Update $s_t \leftarrow s_{t+1}$;
        if *is time to update weights* **then**
            Save and append learner trajectory;
            Calculate feature expectations $\mu(\tilde{\pi})$ from learner trajectories;
            do
                $\mathbf{w}, status$ = weightsUpdate($\mu(\tilde{\pi})$, $\mu(\pi_E)$);
                if *status=infeasible* **then**
                    Remove last learner feature expectation;
            while *status=infeasible*;
        Check environment done;
    end
end

Figure 5 shows the evolution during training of the agent to keep the pole upright, satisfying the environment constraints. The training returns the weights

Algorithm 4: experienceReplay

Requirements: Batch size, Memory;
if *Memory size* $\geq$ *Batch size* **then**
  for $(s_t, a_t, s_{t+1}, r_t, done)$ *in random sample of memory* **do**
    if *done* **then**
      $QValue = r_t$;
    else
      $QValue = r_t + \gamma * \text{modelPredict}(s_{t+1})$;
    end
    QValues[a_t]=QValue;
    Fit model with new QValues;
  end
end

for each feature. The inferred reward function is:

$$r(s) = -988.583\text{e}{-6}\,[\phi_1(s) + \phi_2(s) + \phi_3(s) + \phi_4(s)]$$

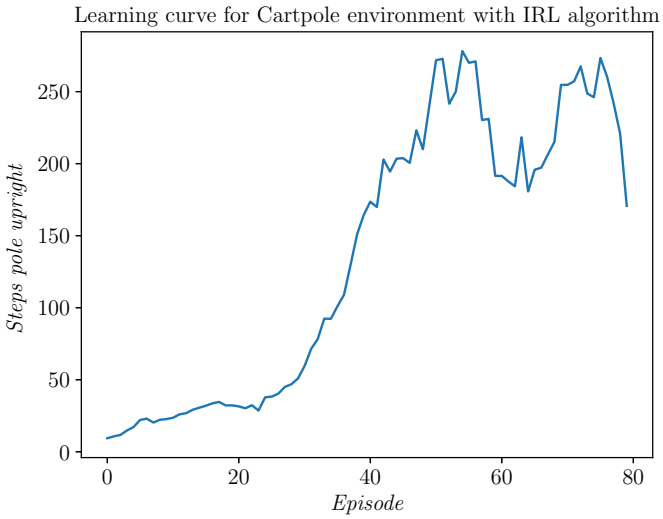

Fig. 5. Learning curve for the Cartpole-v1 Environment

A final remark with the continuous environment is that in some cases during training, when the agent almost know how to perform the task, the random samples of the memory result in very similar states. So, when carrying out the optimization process with neuronal network, the system forgets earlier stages in

the trajectory, causing a fall in the learning process and damaging the value of the weights that were obtained in previous episodes.

5 Conclusions

It was show that through the expert behavior on a certain task, an agent can learn it without necessarily following the same trajectory. For this, we need to define features that are significant for the reward function of the task. The weights of those features can be found establishing the problem as a SVM, looking for the maximum margin that separates the hyper plane from the expert and learner classes. We tested the IRL algorithm with discrete and continuous environments. We found that the learner accomplishes the task demonstrated by the expert, although following a different trajectory. However the utility gained by both agents were the same.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning, in Proceedings of the Twenty-first International Conference on Machine Learning, ICML '04, (New York, NY, USA) (2004)
2. Agrawal, A., Verschueren, R., Diamond, S., Boyd, S.: A rewriting system for convex optimization problems. Journal of Control and Decision 5(1), 42–60 (2018)
3. Barto, A.G., Sutton, R.S., Anderson, C.W.: Artificial neural networks. chap. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, pp. 81–93. IEEE Press, Piscataway, NJ, USA (1990), http://dl.acm.org/citation.cfm?id=104134.104143
4. Diamond, S., Boyd, S.: CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research 17(83), 1–5 (2016)
5. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. CoRR cs.LG/9905014 (1999), http://arxiv.org/abs/cs.LG/9905014
6. Ng, A.Y., Russell, S.J.: Algorithms for inverse reinforcement learning. In: Proceedings of the Seventeenth International Conference on Machine Learning. pp. 663–670. ICML '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000), http://dl.acm.org/citation.cfm?id=645529.657801
7. Piot, B., Geist, M., Pietquin, O.: Bridging the gap between imitation learning and inverse reinforcement learning. vol. 28, pp. 1814–1826 (Aug 2017)
8. Stoica, I., Song, D., Popa, R.A., Patterson, D.A., Mahoney, M.W., Katz, R.H., Joseph, A.D., Jordan, M., Hellerstein, J.M., Gonzalez, J., Goldberg, K., Ghodsi, A., Culler, D.E., Abbeel, P.: A berkeley view of systems challenges for ai Tech. Rep. UCB/EECS 2017-159 (Oct 2017)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (1998)