FASTER APPRENTICESHIP LEARNING THROUGH INVERSE OPTIMAL CONTROL

BY

ANDREY ZAYTSEV

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Assistant Professor Jian Peng

Abstract

One of the fundamental problems of artificial intelligence is learning how to behave optimally. With applications ranging from self-driving cars to medical devices, this task is vital to modern society. There are two complementary problems in this area – reinforcement learning and inverse reinforcement learning. While reinforcement learning tries to find an optimal strategy in a given environment with known rewards for each action, inverse reinforcement learning or inverse optimal control seeks to recover rewards associated with actions given the environment and an optimal policy. Typically, apprenticeship learning is approached as a combination of these two techniques. This is an iterative process – at each step inverse reinforcement learning is applied first to get the rewards, followed by reinforcement learning to produce a guess for an optimal policy. Each guess is used in the further iterations to come up with a more accurate estimate of the reward function. While this works for problems with a small number of discreet states, the approach scales poorly. In order to mitigate those limitations, this research proposes a robust approach based on recent advances in the field of deep learning. Using the matrix formulation of inverse reinforcement learning, a reward function and an optimal policy can be recovered without having to iteratively optimize both. The approach scales well for problems with very large and continuous state spaces such as autonomous vehicle navigation. An evaluation performed using OpenAI RLLab suggests that this method is robust and ready to be adopted for solving problems both in research and various industries.

Table of Contents

Chapter 1

Introduction

The problem of learning to perform various tasks in an optimal way is one of the fundamentals of the field of artificial intelligence. While this problem appears in different subfields of machine learning such as supervised and unsupervised learning, it is the central task of reinforcement and apprenticeship learning.

There is a multitude of applications related to optimal control. Autonomous vehicles require selecting an optimal direction and speed of driving given various sensor and location inputs. Robots need to continuously find optimal torques for their motors in order to walk or navigate through environments full of obstacles. Strategies for games such as Go and Poker involve finding an optimal move given the current state of the board. All these use cases make the problem a vital one.

There are three main problems related to optimal control: reinforcement learning, inverse reinforcement learning, and apprenticeship learning. The first, reinforcement learning, assumes that there is an environment with known states, actions, and rewards. One example would be a game of Pacman, where states are simply the states of the board, actions are directions of travel of Pacman, and rewards are the game scores in the upper right-hand corner. Therefore, the problem of optimal control (another name often used for reinforcement learning) is to find a formalized set of behaviors, known as a policy, that maximizes the total obtained reward. The second problem, inverse reinforcement learning (or inverse optimal control) assumes that there is an optimal policy and an environment with known states and actions but without the reward function. The goal is to recover the rewards of the original environment such that the optimal policy gets the highest total reward of all other possible policies. The third problem, apprenticeship learning, seeks to mimic a given optimal policy in an environment without rewards. The goal of apprenticeship learning is to come up with a model that represents a policy performing as optimally and close as possible to the given policy in the original environment. Due to the nature of the problem, it is often approached as a combination of inverse reinforcement learning and reinforcement learning. First, the environment is fully modeled by recovering the original reward function. Next, an optimal policy is found in the resulting environment.

This research focuses on apprenticeship learning. Traditionally, in order to solve the apprenticeship learning problem, an iterative approach is taken [Abbeel and Ng, 2004]. At the beginning of each step a reward function is found that separates the given optimal policy from all the other policies we currently have in our collection. Next, this reward function is used to find an optimal policy in the resulting environment. This policy is added to the current collection of policies to be used in further steps. The process terminates once the reward function and the derived policy converge to the optimality. While this process is fairly intuitive and effective for problems with small spaces of discreet states, it does not scale well. Applications such as self-driving cars have vast continuous state spaces, which make it infeasible to run this approach with today's computational capabilities.

While the amount of data and state spaces required to represent problems grew much faster than computational capabilities, the field of machine learning saw a significant breakthrough related to the scale of data in the form of deep neural networks [Krizhevsky et al., 2012]. Such neural networks are capable of representing complex nonlinear multidimensional functions, which allows them to solve tasks such as image recognition [He et al., 2016] with above human performance. Deep neural networks can be integrated with reinforcement learning in the form of deep Q learning [Mnih et al., 2013], which makes it possible to train a model that directly produces an optimal policy.

In order to make apprenticeship learning robust and scalable, this research employs deep neural networks alongside a novel matrix formulation of inverse reinforcement learning in order to quickly recover both the reward function and a model of an optimal policy. This approach generalizes to infinite continuous state and action spaces, and can be used with the previously mentioned applications.

Chapter 2

Background

While the applications of optimal control are very diverse, there is one widely recognized formalization called the Markov Decision Process (MDP) that is used across reinforcement, inverse reinforcement [Ng et al., 2000], and apprenticeship learning [Abbeel and Ng, 2004]. Typically, it is a model that consists of state space S, action space A, reward function R, transition model T, discount factor γ and initial state distribution D. T, the transition model is a set P_{sa}, which given a state and an action yields a probability distribution over the next states the system will end up in. R, the reward function maps a state (or a state and an action depending on the formulation) to a reward, which is a real number. This represents the reinforcement the agent receives after ending up in a particular state and/or picking a particular action. γ, the discount factor, ensures that the rewards obtained now are more valuable than the rewards obtained in the future. If the instantaneous reward is r, then after n steps, the reward will be $\gamma^n r$.

Given this model, we can define a policy π as a mapping of $S \mapsto A$. That is, given a state, the policy produces an action that it deems to be optimal. There are two functions related to the notion of a policy. The first one is the value function:

$$V^{\pi}(s_1) = E[R(s_1) + \gamma R(s_2) + \ldots | \pi] \tag{2.1}$$

This represents the total discounted reward the policy gets by starting in a given state. Similarly, the Q-function is:

$$Q^{\pi}(s,a) = R(s) + \gamma E_{s' \sim P_{sa}(.)}[V^{\pi}(s')] \tag{2.2}$$

This represents the total discounted reward the policy gets by taking a certain action in a given state.

Using these definitions, we can now formulate what it means for a policy to be optimal. The optimal value function is $V^* = \sup_\pi V^\pi(s)$. A policy π^* is considered optimal when for all s, $V^\pi(s) = V^{*\pi}(s)$. Thus, the value function at each state is optimal for such a policy. Finding an optimal policy is the main task of reinforcement learning. In contrast, inverse reinforcement learning finds R given an optimal policy and an MDP without the reward function. Similarly, apprenticeship learning seeks to find parameters θ for a policy π_θ, such that for all S, $V_{\pi_\theta}(s) = V_{\pi^*}(s)$ given a set of state-action pairs (s, a) for an optimal policy π^*.

Chapter 3

Related Work

There has been a substantial amount of research focusing on apprenticeship learning and related problems.

3.1 Max-margin Formulation

Apprenticeship learning is usually solved through inverse reinforcement learning. This approach was established by one of the foundational studies in the area [Abbeel and Ng, 2004]. The algorithm is based on the max-margin formulation of the problem. The target reward function is represented as a linear combination of the feature mapping ϕ and the optimal coefficients w^*, which are unknown:

$$R^* = w^{*T}\phi \tag{3.1}$$

In addition, we assume that we have some trajectories sampled from the expert policy π_E. This gives us the expert's feature expectations $\mu_E = \mu(\pi_E)$.

The algorithm is iterative and maintains a current collection of previously found policies $\pi^{(j)}$. At the beginning of each step, it gets the current policy's feature expectations $\mu(\pi^{(i)})$. Following that, it uses the max-margin formulation that requires the given expert policy to be the only optimal one and the rest of the policies in the collection to be separated from the expert one with the largest margin possible:

$$t^{(i)} = \max_{w:||w||_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)}) \tag{3.2}$$

The w at which this maximum is achieved gives us the linear features $w^{(i)}$ of the reward function $R^{(i)}$ – the estimate of our target function R^*. In addition, the maximum itself, $t^{(i)}$, is used as a termination or convergence criterion for the algorithm. If its value is less than some predefined ϵ, the algorithm terminates, and $R^{(i)}$ is declared to be the target reward function. This maximum margin problem is similar to that of a support vector machine (SVM). Thus, solving for $t^{(i)}$ and $w^{(i)}$ can be done using any SVM-solver.

In the event $t^{(i)}$ is still greater than ϵ, the obtained reward function is used to find an optimal policy $\pi^{(i)}$, which is added to the collection of obtained policies. This is done using any robust reinforcement learning algorithm such as REINFORCE [Williams, 1988]. This is the end of the step, and the method continues onto another iteration.

This method is fairly intuitive, since at each step it tries to find the reward function that separates optimal policy from non-optimal ones as much as possible. However, this algorithm does not scale well to be applicable to modern day complex multidimensional problems. In addition to the assumption that the reward function can be expressed as a linear combination of some predefined features, there is a need to do reinforcement learning repeatedly.

3.2 Maximum Entropy Formulation

The max-margin formulation is only one of the many approaches to inverse reinforcement learning. Some of the other approaches are based on the maximum entropy formulation of inverse optimal control [Ziebart et al., 2008]. This model assumes that the probability of user preference for a given trajectory is proportional to the exponential of the reward along that path. In other words, the following holds:

$$P(\varsigma|r) \propto exp\{\sum_{s,a \in \varsigma} r_{s,a}\} \tag{3.3}$$

In this formula, ς represents an expert demonstration, r represents the reward obtained on that demonstration. This formulation allows the model to handle expert sub-optimality as well as stochasticity by looking at a distribution.

There were developed some approaches using this formulation in combination

```
Algorithm 1 Maximum Entropy Deep IRL
Input: $\mu_D^a, f, S, A, T, \gamma$
Output: optimal weights $\theta^*$
 1: $\theta^1$ = initialise_weights()
    Iterative model refinement
 2: for n = 1 : N do
 3:     $r^n$ = nn_forward($f, \theta^n$)
        Solution of MDP with current reward
 4:     $\pi^n$ = approx_value_iteration($r^n, S, A, T, \gamma$)
 5:     $\mathbb{E}[\mu^n]$ = propagate_policy($\pi^n, S, A, T$)
        Determine Maximum Entropy loss and gradients
 6:     $\mathcal{L}_D^n = \log(\pi^n) \times \mu_D^a$
 7:     $\frac{\partial \mathcal{L}_D^n}{\partial r^n} = \mu_D - \mathbb{E}[\mu^n]$
        Compute network gradients
 8:     $\frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n}$ = nn_backprop($f, \theta^n, \frac{\partial \mathcal{L}_D^n}{\partial r^n}$)
 9:     $\theta^{n+1}$ = update_weights($\theta^n, \frac{\partial \mathcal{L}_D^n}{\partial \theta_D^n}$)
10: end for
```

Figure 3.1: Pseudocode for maximum entropy deep IRL [Wulfmeier et al., 2015]

with deep neural networks [Wulfmeier et al., 2015]. This approach uses the context of Bayesian inference to solve the problem using MAP estimation, by maximizing the probability to observe expert demonstrations under current parameters. Based on that, the gradients are calculated and back propagated through the deep neural network that constitutes a base of the model. A pseudocode of this algorithm is illustrated on figure 3.1.

Despite taking a different approach to the problem of inverse reinforcement learning, the maximum entropy based formulation still suffers from the drawback of having to iteratively run reinforcement learning algorithms. As mentioned earlier, this limits the scalability of an IRL algorithm, which is vital in modern complex multidimensional research problems.

```
Algorithm 1 Deep Q-learning with Experience Replay
  Initialize replay memory $\mathcal{D}$ to capacity $N$
  Initialize action-value function $Q$ with random weights
  for episode $= 1, M$ do
      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
      for $t = 1, T$ do
          With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
          Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
          Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
      end for
  end for
```

Figure 3.2: Pseudocode for deep Q-learning [Mnih et al., 2013]

3.3 Deep Q-Learning

In addition to various improvements to inverse optimal control algorithms, reinforcement learning has also been an active area of research. With the emergence of deep neural networks, the state-of-the-art reinforcement learning algorithms are largely based on deep Q-learning [Mnih et al., 2013]. This method employs deep convolutional neural networks to model the Q-function of the underlying MDP. The pseudocode of the approach is shown on figure 3.2. It takes advantage of a technique called experience replay by storing agent's experiences at each time-step pooled over many episodes into replay memory.

By using this technique, which is based on deep convolutional neural networks, it is possible to successfully perform reinforcement learning on complex multidimensional problems such as playing Atari games. While this technique improves the robustness of reinforcement, there is still a significant cost associated with iteratively running RL as a part of apprenticeship learning or inverse optimal control. Our method takes advantage of the breakthroughs in the area of deep neural networks, and we use a variation of deep Q-learning in the evaluation part of this thesis.

Chapter 4

Method and Implementation

We propose a method that takes advantage of the recent advances in deep learning by employing neural networks to model both the rewards function and the obtained optimal policy. In this section, we first describe the concept of reward shaping, which is heavily used by our method. We then describe our apprenticeship learning model, followed by a discussion of our implementation of the approach.

4.1 Reward Shaping

Certain reward functions allow a greedy policy, which chooses the action associated with the highest immediate reward, to be optimal. If one can transform a given reward function into this form without changing the value and Q-functions, the problem of reinforcement learning will be solved by a simple greedy policy without the need for expensive computation. An example of this is illustrated in figure 4.1.

This idea lies at the heart of reward shaping [Ng et al., 1999]. It turns out that it is always possible to transform the rewards function in an MDP without changing the optimal value and Q-functions. Specifically, the preferred transformation is such that the rewards associated with optimal actions are zero, and the rewards for suboptimal actions are negative.

The method modifies the original rewards function by adding a function that takes following parameters: s, the current state, a, the current action, and s', the next state. This function is the following:

$$F(s, a, s') = \gamma V^*(s') - V^*(s) \tag{4.1}$$

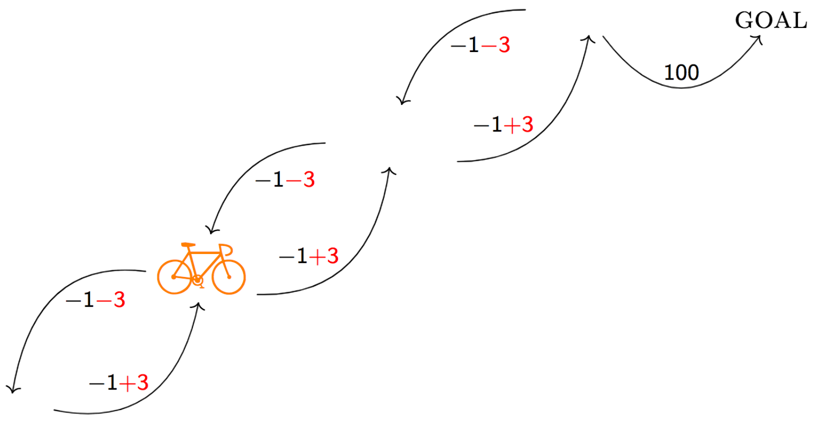


Figure 4.1: Example of reward shaping allowing for greedy policies to be optimal [Ng et al., 1999]

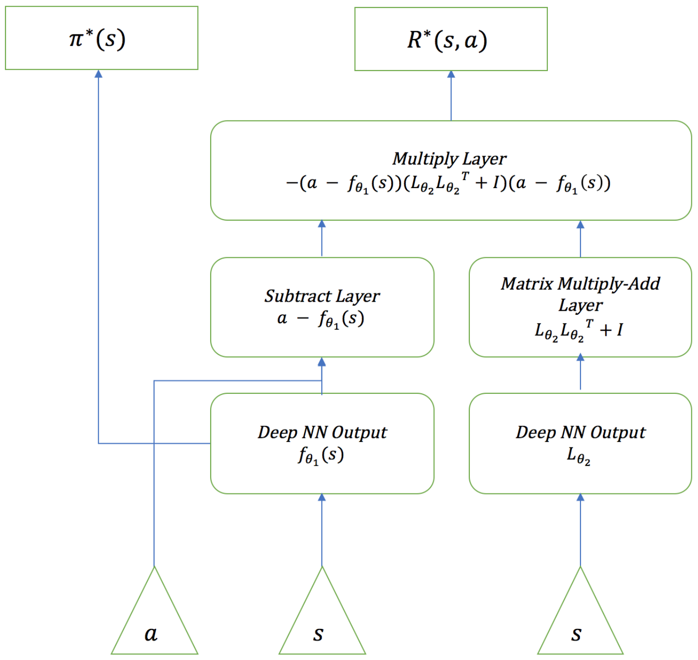
The new reward function is then the following:

$$R_{shaped}(s,a) = R(s,a) + F(s,a,s') \tag{4.2}$$

The optimal value function is necessary to do reward shaping. Thus, reward shaping does not help us solve the problem of reinforcement learning since it depends on having a solution to it – the optimal value function is sufficient by itself to construct an optimal policy without additional computation. However, if an algorithm for inverse reinforcement learning could recover a shaped reward function, the problem of apprenticeship learning would be solved as well. We use this observation in our method.

4.2 Model

Employing the concept of reward shaping, the problems of inverse optimal control and apprenticeship learning becomes simply the task of finding the shaped reward function for a given policy and environment. All values of the shaped reward func-



$\pi^*(s)$
$R^*(s,a)$
Multiply Layer
$-(a - f_{\theta_1}(s))(L_{\theta_2}L_{\theta_2}{}^T + I)(a - f_{\theta_1}(s))$
Subtract Layer
$a - f_{\theta_1}(s)$
Matrix Multiply-Add Layer
$L_{\theta_2}L_{\theta_2}{}^T + I$
Deep NN Output
$f_{\theta_1}(s)$
Deep NN Output
L_{θ_2}
a
s
s

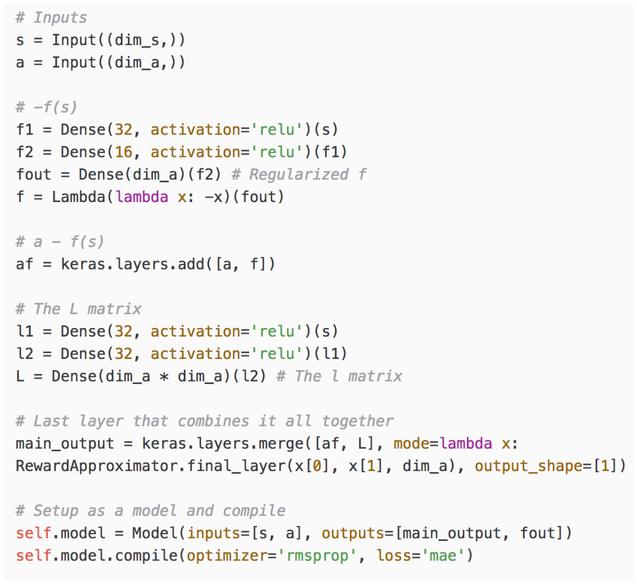
Figure 4.2: Diagram of the final modular neural network

tion are less than or equal to zero. This is one of the necessary conditions for the correctness of the found reward function.

Taking the above into account, it is also important to learn an optimal policy. The task of inverse optimal control can be formulated as a maximization problem. Thus, to include apprenticeship learning, $a - f_{\theta_1}(s)$ should be in the final product. We therefore develop the following equation to learn the shaped rewards function and the optimal policy simultaneously:

$$R(s,a) = -(a - f_{\theta_1}(s))^T (L_{\theta_2}(s)L_{\theta_2}(s)^T + I)(a - f_{\theta_1}(s)) \tag{4.3}$$

In this equation, $R(s,a)$ represents the resulting shaped reward function and $f_{\theta_1}(s)$ is the learned optimal policy, which is represented by a deep neural network. The other parametrized variable represented by a deep neural network $L_{\theta_2}(s)$ is a matrix that is directly related to representing the final reward function.



```
# Inputs
s = Input((dim_s,))
a = Input((dim_a,))

# -f(s)
f1 = Dense(32, activation='relu')(s)
f2 = Dense(16, activation='relu')(f1)
fout = Dense(dim_a)(f2) # Regularized f
f = Lambda(lambda x: -x)(fout)

# a - f(s)
af = keras.layers.add([a, f])

# The L matrix
l1 = Dense(32, activation='relu')(s)
l2 = Dense(32, activation='relu')(l1)
L = Dense(dim_a * dim_a)(l2) # The l matrix

# Last layer that combines it all together
main_output = keras.layers.merge([af, L], mode=lambda x:
RewardApproximator.final_layer(x[0], x[1], dim_a), output_shape=[1])

# Setup as a model and compile
self.model = Model(inputs=[s, a], outputs=[main_output, fout])
self.model.compile(optimizer='rmsprop', loss='mae')
```

Figure 4.3: Code snippet showing the implementation of the model using Keras

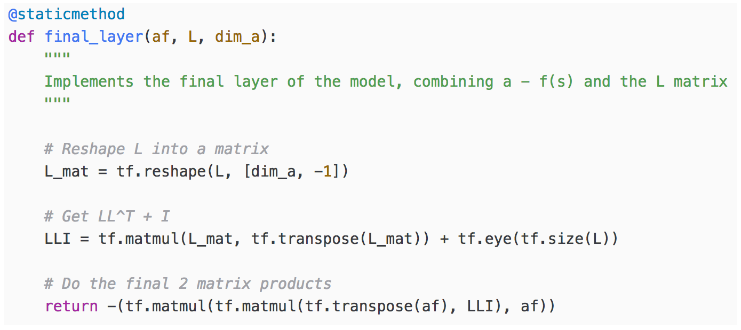
Note, that $L_{\theta_2}(s)L_{\theta_2}(s)^T + I$ is a positive-definite matrix. This allows us to ensure that the resulting reward function satisfies one of the criteria to be shaped by always being less than or equal to zero.

The term $a - f_{\theta_1}(s)$ becomes zero when a given action is optimal, which in turn, gives the overall reward of zero. When an action is suboptimal according to the learned policy, the overall result is strictly negative since we are multiplying by a positive definite matrix and inverting the sign.

The described model is one modular deep neural network with various layers corresponding to part of the equation. Figure 4.2 illustrates the layers of the resulting neural network. Note, that in addition to the main output, which is the rewards function for IRL, we also have an auxiliary output – an optimal policy for the task of apprenticeship learning.

In order to train the model, the following optimization is implemented:

$$R^* = \max_{\theta_1,\theta_2} R(s, a) \tag{4.4}$$


```python
@staticmethod
def final_layer(af, L, dim_a):
    """
    Implements the final layer of the model, combining a - f(s) and the L matrix
    """

    # Reshape L into a matrix
    L_mat = tf.reshape(L, [dim_a, -1])

    # Get LL^T + I
    LLI = tf.matmul(L_mat, tf.transpose(L_mat)) + tf.eye(tf.size(L))

    # Do the final 2 matrix products
    return -(tf.matmul(tf.matmul(tf.transpose(af), LLI), af))
```

Figure 4.4: Code snippet showing a TensorFlow implementation of the final layer

Note, that while there are separate coefficients θ_1 and θ_2 for parts of the model, the overall optimization is completed via back propagation through all the layers of the final model, and thus does not require any special optimization methods.

4.3 Implementation

In order to implement our proposed model, we chose to use a Python deep learning framework called Keras. This cross-platform library is compatible both with Theano and TensorFlow, allowing it to take advantage of the massive scalability potential offered by those platforms. In order to quickly train a deep neural network, Theano and TensorFlow use GPUs to massively parallelize the computation. That illustrates that it is equally easy to implement our model both for small toy examples and for complex multidimensional research problems such as autonomous vehicle navigation.

The main part of our code representing the model is shown on figure 4.3. We took advantage of neural network layers provided by Keras in order to achieve concise and understandable source code. In our code, Dense layers represent fully-connected layers, Lambda layers are used for simple operations such as multiplying the output by -1, and the Merge layer is used for the final operation involving matrix multiplication. One way to implement the final layer in TensorFlow is shown in figure 4.4. Another advantage of Keras is that back propagation is done automatically using the function associated with each layer. This further reduces the amount of code that needs to be written for the final model to work.

Chapter 5

Evaluation

Our goal is that the proposed method is able to run robustly and correctly on real tasks such as autonomous driving and playing complex games, in which a human has not been beaten using state-of-the-art research. In order to ensure that this is achievable, an evaluation must be done. The next three sections describe the evaluation approach, prior research, and the results, followed by a discussion.

5.1 Approach

Current research problems such as autonomous navigation and robotic control are complex multidimensional tasks. Obtaining meaningful results on such tasks requires access to either special hardware [Levine et al., 2016] or a cluster with immense computational power [Mnih et al., 2013]. In order to avoid investments both in terms of resources and time, we took the approach of training our algorithm to perform the simple and easily-reproducible task of Cart-Pole balancing.

Figure 5.1 illustrates the specifics of the task. There is a cart that is able to move left and right. On top of the cart, a pole is attached using two springs. Whenever some force is applied to the cart, it experiences acceleration, and the spring changes position. The goal of the task is to balance the pole such that it remains at a certain angle, usually 90 degrees, to the cart. A policy receives a reward at every timestep that the pole is within a certain ϵ of the target angle.

While one way to perform an evaluation of our model is to implement the Cart-Pole task from scratch, this is not the best approach. In addition to being prone to errors, such an implementation is not easily reproducible by future research in this area. In order to solve those problems, we used the state-of-the-art reinforcement

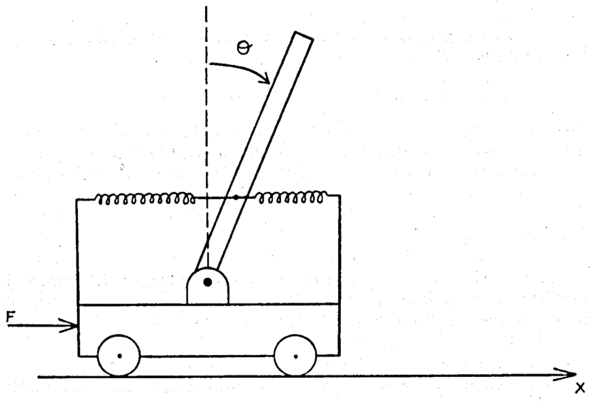


Figure 5.1: The Cart-Pole task illustration [Michie and Chambers, 1968]

learning benchmarking package called RLLab [Duan et al., 2016]. The package has an implementation of the Cart-Pole task and provides an API in Python to interact with the environment. In addition, this allows us to use our Keras implementation of the model described in the previous section without making any modifications to it.

After choosing the task and fine-tuning implementation specifics, we proceeded with our evaluation using the following approach. Each of the described steps are discussed in details below.

1. Train an optimal policy π_E in the original Cart-Pole environment to be used as the expert policy and as a benchmark
2. Sample trajectories form the expert policy π_E
3. Run our implementation using those trajectories and the original environment with rewards removed
4. Take the resulting learned policy π^* and evaluate it in the original environment comparing it to the benchmark π_E
5. Take the resulting rewards function R^* and use it to train an optimal policy π_R

6. Evaluate the learned policy π_R in the original environment and compare it to the benchmark π_E

The first step of the process uses reinforcement learning in the original environment to train an optimal policy π_E for the task of Cart-Pole. This is necessary in order to have an expert policy to perform apprenticeship learning. In addition, this policy's average discounted reward total is used to benchmark other policies trained in the further stages of this evaluation process.

The next step uses the expert policy π_E to obtain trajectories. Both inverse optimal control and apprenticeship learning problems take trajectories of the expert policy as an input. Thus, it is necessary to run our policy π_E and obtain sequences in the form (s, a) representing successive states and actions.

At the beginning of the third step, we took the original environment and removed the rewards, since both IRL and apprenticeship learning operate on MDPs without rewards. We then used the resulting environment without rewards and the trajectories obtained in the previous step to train our model. As a result of this process, the model gave us two outputs: the learned rewards function R^* and the learned optimal policy π^*. This represents a combination of the tasks of apprenticeship learning and inverse optimal control.

After this step, we obtained two outputs of our algorithm that need to be evaluated. We first evaluated the resulting policy π^* in the original environment and compare it to π_E. A similarly high average discounted reward total would signify that the task of apprenticeship learning was completed successfully.

As the fifth step, we evaluated the recovered shaped reward function R^*. Since we cannot directly compare it to the original R, we evaluated its potential to produce optimal policies. Thus, we trained an optimal policy π_R using a reinforcement learning algorithm on the environment R^*.

Finally, we compared the performance of the obtained policy π_R to the expert policy π_E on the original environment. If π_R performs just as well as π_E, that would indicate that the task of inverse reinforcement learning successfully produced a rewards function with the same optimal policy.

```
Algorithm 1 DDPG algorithm
  Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
  Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
  Initialize replay buffer $R$
  for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
      Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
      Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
      Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
      Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
      Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
      Update the actor policy using the sampled gradient:
          $\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
      Update the target networks:
          $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
          $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
  end for
```

Figure 5.2: Pseudocode for the DDPG algorithm [Lillicrap et al., 2015]

Overall, the goal of our approach was to evaluate the performance of both inverse optimal control and apprenticeship learning tasks embedded in our method.

5.2 Reinforcement Learning using DDPG

The evaluation process, mentioned in detail in the previous sections, has multiple instances where reinforcement learning is needed. In our implementation, we chose to use a state-of-the-art RL method called DDPG [Lillicrap et al., 2015].

DDPG combines the actor-critic approach with deep convolutional neural networks and replay buffers introduced in the deep Q-learning algorithm (DQN) [Mnih et al., 2013]. This allows it to overcome the limitations of DQN and robustly perform on tasks with continuous action spaces. The pseudocode for the algorithm is illustrated on figure 5.2. We believe that due to its advantages, this state-of-the-art

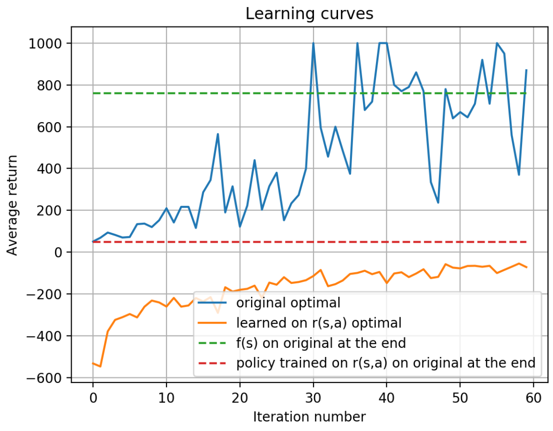


Figure 5.3: Cartpole environment evaluation

reinforcement learning algorithm best fits our purpose to training optimal policies on the Cart-Pole environment in this evaluation.

5.3 Results and Discussion

Figure 5.3 illustrates the results of our evaluation in the form of learning curves of the implementation on the Cart-Pole task. The evaluation began by first training an optimal policy in the original environment with known rewards (shown as the blue curve). As stated before, this was done using the DDPG reinforcement learning algorithm [Lillicrap et al., 2015]. This optimal policy was then used as the benchmark. Next, we sampled 100 trajectories from our optimal policy and used those to train our model. Each trajectory represents one complete execution of an optimal policy. The green dotted line represents the optimal policy learned from those trajectories using our implementation and evaluated in the original environment. In addition, we tracked the average discounted reward of the optimal policy on the learned shaped environment (the orange line). This was also done using DDPG and allows us to see how close the policy gets to being optimal under the shaped rewards assumption. Following this process, we took the optimal policy, which obtains the average discounted reward of about zero, and transferred it into the original environment (the red dotted line) to evaluate whether our learned environment could be used to train optimal policies.

As can be seen on the graph, the trained policy π^* achieves performance as high as the expert policy π_E. This indicates that the implementation of our approach completes the task of apprenticeship learning successfully. The average discounted reward of the optimal policy π_E obtained by DDPG on the original environment starts to fluctuate after iteration 30 (seen as the blue line on the graph). This happens because the algorithm converged to the optimal policy, and further optimization does not provided any added benefit. The green line representing the average discounted return on the trained policy π^* closely matches the level at which the return of π^E converged.

While the method produced a nearly-optimal policy in the Cart-Pole environment, the results in relation to the trained rewards function R^* were not as definite. The average discounted return of the optimal policy π_R trained and evaluated on R^* converged to zero, just as expected with a shaped rewards function. However, when this policy π_R was transferred to the original environment, it did not perform nearly as well as the trained policy π^*. This implies that at this stage, the obtained rewards function R^* is not a solution to the inverse optimal control problem.

We believe that this result in terms of the rewards function R^* may be caused by the overfitting of the model or insufficient trajectory sampling. This can be investigated in the future studies on this approach. We believe that evaluating the rewards function on different tasks such as balancing a pendulum or driving a car in a racing simulator will produce a conclusive result regarding the robustness of the inverse reinforcement learning part of our method. Despite these shortcomings of the produced rewards function, the main task of apprenticeship learning produces robust policies with high average returns.

Chapter 6

Conclusion

In this thesis, we presented a novel way to perform apprenticeship learning by simultaneously solving the task of inverse optimal control. Our evaluation of the method demonstrated that it is capable of producing optimal policies without knowing the original rewards function. These policies are able to perform just as well as those obtained by state-of-the-art reinforcement learning algorithms such as DDPG on the full environment with known rewards.

In addition to being robust, this method is simple to implement with an immense scalability potential. The entire system can be easily implemented using cutting-edge machine learning libraries such as TensorFlow and Theano. The use of those libraries and the representation of the learned policy and the learned environment as deep neural networks allow the system to be applied to complex multidimensional tasks such as autonomous vehicle control. While those tasks require significant investments in computing resources and time, TensorFlow and Theano utilize GPUs in order to parallelize the computations, giving the opportunity to solve tasks that were not feasible in the past.

Our method is completely task-agnostic, which means it can be applied just as easily to the problem of robotic navigation through a 3-dimensional terrain, as to the problem of balancing a Cart-Pole. In contrast, some of the previous works in this area made various simplifying assumptions about the environment or scaled poorly with the complexity and dimensionality of inputs. Our approach requires only a sample of trajectories of an optimal agent, which allows for use in continuous, partially-observable, and infinite environments.

Chapter 7

Future Work

In addition to improving the generalization of the rewards function in the Cartpole environment, there is a wide range of future research directions. The RLLab environment includes a variety of complex tasks ranging from overcoming 2-dimensional obstacles to a 3-dimensional model of a human walking through difficult terrain. Evaluating our proposed method on those tasks will make it possible to see how generalizable and robust it is.

In today's world, one of the most pervasive areas for apprenticeship learning is driving autonomous vehicles. Just as with the RLLab environment, our apprenticeship learning method can be applied to a 3D driving simulator to evaluate the performance and applicability of it to the task of driving. We believe that the use of deep neural networks in an efficient manner gives our method the potential to produce high performance on this task.

Our method is completely task-agnostic, which implies that the user is in charge of choosing an appropriate deep neural network structure. Integrating existing specialized deep neural networks trained on the inputs relevant to the given environment, such as those for image recognition, may prove beneficial to the performance of the overall system.

References

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

[Duan et al., 2016] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Michie and Chambers, 1968] Michie, D. and Chambers, R. A. (1968). Boxes: An experiment in adaptive control. *Machine intelligence*, 2(2):137–152.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[Ng et al., 1999] Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

[Ng et al., 2000] Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

[Williams, 1988] Williams, R. J. (1988). On the use of backpropagation in associative reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 263–270. San Diego, CA.

[Wulfmeier et al., 2015] Wulfmeier, M., Ondruska, P., and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

[Ziebart et al., 2008] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA.