# Behavioral Cloning in OpenAI using Case-Based Reasoning

**Chad Peters and Babak Esfandiari and Sacha Gunaratne and Robert West**
Carleton University, Ottawa ON, Canada

**Abstract**

When learning from observation, and expert agent can be used to model the desired behavior. In cases when an expert is not available, we can resort to near-optimal examples as the source of behavior policies, and judge the accuracy of an agent on how it learns from the example, regardless of the proficiency of the demonstration. In this paper we explore a basic Case-Based Reasoning approach to behavior cloning using the popular OpenAI Gym environment. We demonstrate a practical application of the jLOAF framework to multiple environments, and draw our conclusions from comparison and analysis of various learner and sampling strategies on the overall accuracy of behavior cloning. Our experimental results show how our approach can be used to provide a baseline for comparison in this domain, as well as identify the strengths and weaknesses when dealing with environmental complexity.

## Introduction

This paper describes our work with behavioral cloning in the OpenAI Gym environment. We approached this problem by using an agent already trained for near-optimal performance in various virtual environments, generating observable target behavior, and using a basic Case-Based Reasoning (CBR) approach to training a new agent with only the recordings of observable behavior as a reference.

To do this we used the *Java Learning from ObservAtion Framework* (jLOAF) to come up with similarity metrics that are widely known in the research community, easily repeatable in other domains, and can serve as a viable baseline for future performance comparison. Our work with jLOAF and Gym presented some technical challenges that we had to resolve to allow communication between the agent and environment.

Our research design involved modeling the inputs and the outputs between the environment and the CBR system, so that learning agents can use the observed behavior of other expert agents already proficient in the target virtual environment, modify their own behavior based on similarity of state-action pairs, and resulting with a close approximation in terms of behavioral cloning.

A cursory layout of the remainder of this paper is as follows; first we provide an overview of the related Learning from Observation (LfO) literature, and the cycle of working with the CBR system required for an agent's information knowledge management. Next, we introduce jLOAF, a framework that supports learning in autonomous agents without using direct control by a human supervisor, including some previous examples of implementation in a few problem domains, and discussion of this framework is designed to be extended in its current form to adapt to future problem domains. The following section provides an introduction to the OpenAI Gym environment, and an architectural overview of our interface layer between the Java Virtual Machine (JVM), and Python Interpreter used by jLOAF and Gym, respectively. The experimental methodology includes how the two environments interface, and generation of recorded observations used to build the base-cases. Last, we discuss the experimental results and performance evaluations of the selected learning algorithms.

## Background

There are a number of ways that intelligent agents can learn new concepts or ideas, or improve on existing ones. Machine Learning paradigms can be described in terms of how the reward function is delivered to the learning agent. In Reinforcement Learning, the feedback comes directly from the environment, and is used to tune a policy function to predict the expected utility across multiple actions (Sutton & Barto, 2012). In some cases, however, a reward function may not be directly available; it can be difficult to describe the proper function when either the human expert does not have personal experience beyond that of observation, or the problem domain reaches a level of complexity that is difficult to describe using a heuristic or programmatic approach. This can be seen in example of optimal behavior that can only be indirectly compared to the agent attempting to learn some function or trajectory given an observed state in the environment (Ontañón, Montaña, & Gonzalez, 2014). In these types of scenarios, we can resort to a form of learning commonly known as Learning from Observation (LfO), where the actions to be modeled become the examples provided to the learner.

## Learning from Observation

Humans and non-human mammals exhibit the ability to learn from reinforcement from birth (Friedenberg & Silverman, 2005), and as intelligent agents, still rely on some form of feedback (Russell & Norvig, 2009), whether supervised (by others), or unsupervised (by ourselves) using a memory-based system of recall.

The concept of Learning from Observation (also known as Learning from Demonstration), provides an affordance to teaching intelligent agents by providing sample behaviors to learn from, and removing the requirement for direct intervention by the researcher. This can be done by presenting *ideal examples* of desired behavior to the learning agent, and through some means of encoding the storage and retrieval of these examples, the learning agent has the opportunity to compare its behavior against the optimal example for the purpose of self correction.

Argall et al.(Argall, Chernova, Veloso, & Browning, 2009) argue that regardless if the environment is physical or virtual, one needs to encode the agent's observation before it can be compared algorithmically, regardless of how the algorithm is represented. With some careful creation of a method to transform observations into a *policy*, and compare that policy against future actions.

An example implementation that satisfies these requirements has been implemented by Floyd and Esfandiari (M W Floyd & Esfandiari, 2011) in the Java Learning from Observation Framework (jLOAF). The current[1] version of jLOAF implements LfO through indeterminate inputs, feature selection and filtering, Case Base creation and pruning, and time-sensitive representation through a method called temporal backtracking (Michael W. Floyd & Esfandiari, 2011). One well-known paradigm of LfO is through the creation of state-action pairs, or *cases*, for use in *case-based reasoning*, as described in the following section.

## Case Based Reasoning

CBR is an approach to problem solving that takes advantage of previously experienced situations in order to infer a probable solution to new experiences (Aamodt & Plaza, 1994). This paradigm differs from other approaches to problem solving that rely solely on a general understanding of the problem domain, and instead leverages specific *cases* that can be reused in new ways and applied to new experiences.

Aamodt and Plaza describe a number of methods(Aamodt & Plaza, 1994) that can be used to index, organize, retrieve, and utilize the information observed in the past, such as by *explar* (finding the right class for an unclassified problem), *instances* (combining cases to form concepts), *memory* (reasoning through search), *analogy* (using a different-but-similar domain), and *typical-case-base* (retrieving and adapting similar cases to new problems). Despite their subtle differences, all of these CBR methods can be represented as a cycle with four main phases; Retrieve, Reuse, Revise, and Retain.

## Research Methodology

The goal of this project was to evaluate a basic CBR approach to behavioral cloning to quickly establish comparative baselines across multiple environments, and which may later be improved upon with more sophisticated approaches. In order to test this approach, we selected frameworks purpose built to support Learning from Observation and Reinforcement Learning, the ability to establish measurements of state-action pairs through a sequence of time, and the ability to encode and transmit this information between the learning agent and each virtual environment.

We selected the OpenAI research platform as it provides a standardized interface for agents to observe, interact with, and receive feedback from a variety of virtual environments. Although the Gym testbed (an instance of the wider OpenAI platform) offers a reinforcement value for training agents, our approach was to train and test an agent using only the observable behavior of an example (near-)*optimal* agent already trained using standard Reinforcement Learning techniques.

We also selected the jLOAF framework to support and demonstrate the ability to both measure the overall performance of various Reasoner classes provided by the framework and use existing case bases generated by jLOAF to interact with the environments in real-time. One could argue that given the simplicity of the observation space in Gym that is used to calculate a similarity functions, we could have directly implemented a simple k-Nearest Neighbor (kNN) function instead of using a framework running in a separate environment. However, we decided to turn a potential obstacle into a challenge to overcome for two reasons: First, the jLOAF library provides an extensive framework that supports any combination of atomic and complex state-action pairs, and scales well to large and complicated environments. This is important for the support of future environments that may compound on existing observation spaces. Second, the jLOAF library provides a test suite to compare different learning strategies across a range of statistical measurements. This is important if we wish to run combinations of offline and batch experiments to compare the time and accuracy trade-offs between different learning agent strategies.

Our decision to couple jLOAF and Gym presented several architectural challenges due to the inherent differences between jLOAF and Gym, most notably the use of

[1] As of this writing, the jLOAF toolset can be accessed at https://github.com/NMAI-lab

incompatible runtime environments. jLOAF uses the Java Virtual Machine, whereas Gym only supports Python[2]. Despite these challenges, we created additional requirements to ensure the tests remain methodologically sound: First, it must provide an *extensible interface* between two disparate systems without sacrificing usability of either; Second, it must support both offline and online *learning and communication* between agent and environment; and finally, it must be *forwards-compatible* with future research projects based on OpenAI, such as the online Universe API used to interface with environments hosted remotely.

In order to properly evaluate agents operating in different environments with unique feature and action spaces, we designed a space-agnostic extension to abstract performance evaluation interface already available in the jLOAF framework so that each Case Base could be generated, stored, retrieved, and evaluated independent of the dimensionality of the observable environment space reported by Gym.

Our research project demonstrates the application of jLOAF to an open-ended Reinforcement Learning platform created by OpenAI (Brockman et al., 2016), and specifically the Gym environment that is used for tuning and testing reinforcement learning agents. The Gym project aims to provide a standard interface for agents to learn from and act upon a variety of virtual environments and problem domains. The main goal of the OpenAI project is to provide a common platform for researchers to compare and discuss novel reinforcement learning algorithms, and find a generalized solution to allow learning in a variety of domains. Our use of Gym for LfO is somewhat unique in that we are not trying to come up with an optimal agent that can beat humans at the task; instead, we wish to approximate and clone another agent, regardless of how good or bad the example may be.

## An Overview of jLOAF

The *Java Learning from ObservAtion Framework* (jLOAF) used in this study implements the general concepts of a CBR system through a collection of abstract classes that can be implemented by an agent to perform Learning by Observation in various environments. The overall architecture can be modified and extended, however all implementations have the following components in common:

**Agent**: represents to the central organizing construct that contains implementations of the others in order to observe the environment, reason about observations, select the most appropriate action, and perform those actions. The implemented agent class is what the main thread will instantiate and invoke.

**Input**: provides a way to encapsulate individual features that represent the environment. It supports both discrete and continuous variables, and can represent Atomic (single feature) and Complex (Atomic or Complex) representations in a recursive hierarchy.

**Action**: can also use Atomic and Complex representations, and represents the outcome of the agent reasoning process, and to interact with the environment.

**Similarity**: is the metric by which two Inputs are compared to each other. jLOAF provides similarity strategies for both Atomic and Complex inputs and actions.

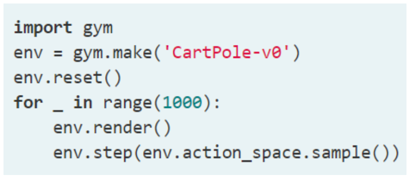
**Reasoning**: is the method by which the agent learns the behavior of the observed expert, and predicting the next action for a given input. There are a number of built-in reasoners that use Machine Learning techniques such as Bayesian, Neural Network, Temporal Backtracking, and K-Nearest Neighbour.

**Performance**: provides the template for evaluating how well an agent learns the target behavior, and performs in new situations through Cross Validation. Statistical libraries provide common measurements for comparison, such as Precision, Recall, and F-Score.

**Filters**: enable you to tune agent performance through both feature selection, and case-base optimization. Feature selection can apply weights to the feature space based on perceived utility, or ignore them outright. Clustering of the case-base allows you to reduce the overall size by combining like-cases, whereas Sampling provides over- and undersampling majority and minority classes, respectively.

## The OpenAI Gym API

The Gym environment is supported in Python[3] 2 and 3, works in both Linux and Windows (the authors have successfully used Gym on both platforms), and provides a standardized interface for each environment. A simple environment is instantiated in a python script[4] (as shown in Figure 1), and runs on the local machine of the Gym host.



```python
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample())
```

*Figure 1: Example Gym Script*

The Gym framework uses a standard agent-environment loop that steps through a new frame whenever the environment's *step* function is called, and returns a vector of four values:

***Observation** (object):* the state of the environment, represented as an array of double values. These values can represent any number of features, from the position or angle of

[2] Most of the OpenAI Gym environments are dependent on 3rd party libraries using Python 2.7

[3] At the time of this writing, Python 2.7 and 3.6 was supported by the OpenAI community.
[4] Examples from https://gym.openai.com/docs/

an object, to a pixel on the screen. This representation is left up to the environment creator.

***Reward*** (float): the reinforcement value used for an to learn in order to maximize the utility of each action. This value can take different ranges for the completion of each environment, as well as signal major events, such as entering a failed state, or achieving a checkpoint required for later success.

***Done*** (Boolean): returns true if the environment has finished one round, otherwise always false.

***Info*** (dictionary): a key-value collection that provides additional information about the state of the game. The OpenAI Gym standards do not allow agents to use this information in order to gain an advantage; rather it can be used by the researcher for development and debugging.

### Problem Classes in Gym

The Gym platform divides the environments into problems subtypes, depending on a number of factors such as the complexity of the representation, the possible feature-action space, and the increasing degree of overall difficulty to put the agent into a "solved" state. Example problem classes include *Search & Optimization* for text-based board games, *Classic Control* using a joystick or control pad, *Atari* games such as *Breakout*, and *Box2D* environments that can scale up for modern displays.

### Action and Observation Spaces in Gym

The Gym API also defines the concept of a *space*[5], that allows the calling agent to briefly interrogate the allowable actions for that environment, as well as the expected range for each feature in an observation. For example, the *Lunar Lander* environment will report a total of four allowable actions for each thruster, and the expected number range for features that describe the position, angle, and velocity of various dimensions. Feature spaces can be a standard unit vector represented as [*-1,* +1], or an infinite boundary represented as [*-inf, +inf*].

## Client-Server Model

The jLOAF-OpenAI interface uses a client-server paradigm to allow communication between the jLOAF agent running in a Java Virtual Machine, and the Gym Environment running in a Python Interpreter VM. We are using Py4J[6] to provide the underlying communication framework.

The Py4J package allows the jLOAF Agent and Gym Environment to communicate with each other through Inter-Process Communication (IPC) between the two virtual machines over TCP/IP, and an API for encapsulating the objects of the corresponding language constructs. This is accomplished by opening a socket and binding a port on each of the Client and Server Gateways for each call between Client and Server. The Py4J threading model[7] allocates a single thread for each call, and allows for event listeners and call backs if needed.

The jLOAF client implements a Py4j.ClientServer bound to the GymEnv Python server entry point, and the GymEnv class defines the interface functions and parameters by which the Java client can remotely create, interrogate, reset, and shutdown any Gym environment pre-installed on the listening server.

## Experimental Design

As defined in the Research Methodology, we wanted to capture near-optimal performance in a variety of environments to be used as the target behavior for observation and emulation. This observed behavior would then be used as the impetus for Case-Base creation, and training of a new agent.

The performance testing methodology comprises a number of scenarios to evaluate the overall accuracy between three elements: *Environments* in Gym to provide varying levels of complexity; *Reasoners* to train and predict the agent; and *Filtering* to optimize the size of the case base.

The following environments were tested to generate supporting enrichment data to describe how the agents learn and perform:

#### Classic Control Environments

- CartPole-v1 [4 features, 2 actions]
- MountainCar-v0 [2 features, 3 actions]

#### 2D Box Environments

- LunarLander-v2 [8 features, 4 actions]

The classic control environments were selected as they provide Complex Inputs and Atomic Actions in a real-time environment as well as providing a reasonable starting point for testing. A more advanced example is the "Box2D" environment, as it provides a considerably more complex observation space than Classic Control, and allows for higher difficulty by forcing agents to start in a randomly generated starting point.

[5] https://github.com/openai/gym/blob/master/gym/core.py
[6] https://www.py4j.org/
[7] https://www.py4j.org/advanced_topics.html#py4j-threading-and-connection-model

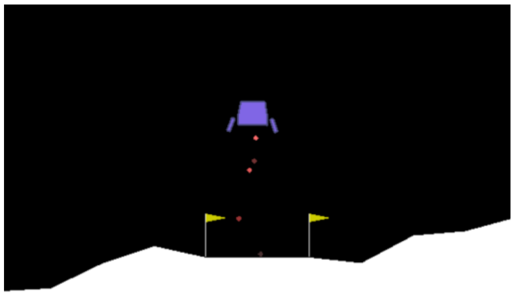


*Figure 2: Lunar Lander Gym Environment*

We selected the K-Nearest Neighbour family of similarity strategies for this project, since they map well to the state-based input-action pairs without adding temporal elements that may skew initial results. This is important as one of the main contributions of this research is providing a baseline that is both simple to measure, easy to replication, so future researchers using our approach can compare and contrast more sophisticated and effective LfO strategies.

Our experiments tested the following similarity strategies:

1. ONEKNN using the first nearest neighbour;
2. SimpleKNN using 5 nearest neighbours; and
3. WeightedKNN using weights for nearest neighbours by proximity to the query.

In order to compare the learning strategies against each other, we used Cross Validation with 10 slices of pre-generated results from expert agents in each domain, each slice containing 1000 cycles of state-action sets. We were also interested in measuring the effects of environmental complexity against each of the learning strategies as each environment varied in complexity, with different feature ranges and allowable actions.

Thus, each combination of environment, reasoner, and filter was run to calculate the mean and standard deviation of Accuracy, Recall, Precision, and F-Score. The F-Scores were then aggregated to provide a *Global F-Score* used to asses each agent configuration over the space of all possible actions given the feature representations provided.

## Experimental Results

The results were used to compare the change in accuracy for each of the Reasoning and Filter strategies selected, as well as their overall effect on the size of the combined Case Base used for training, and size of the slice being tested against.

### Reasoning and Filter Strategies

For the first two environments, being CartPole and MountainCar (*Figure 3*), we can see that the results are comparable between each Reasoning strategy, regardless of the filtering method used.

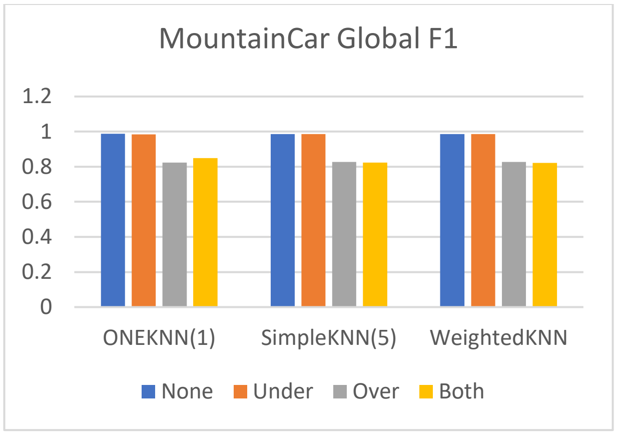


*Figure 3: Mountain Car F1 scores by Reasoner Strategy and Sampling Methodology*

The use of *Oversampling* (by itself, or combined with Undersampling the sparse Case space) has the greatest impact on both reducing the Case Base, and subsequently the accuracy as well. If the loading and seeking time of the Case Base was a great concern, the Oversampling filtering strategy may be worth the considerable reduction of the number of Cases.

### Case Size and Accuracy

The relationship between Case size for training and testing, and their impact on the accuracy, can be clearly seen in the Lunar Lander scenario (Figure 4); the added complexity of the environment posed a challenge for all of the Reasoning strategies, however, the impact on accuracy was not as great when using the Filtering methods.

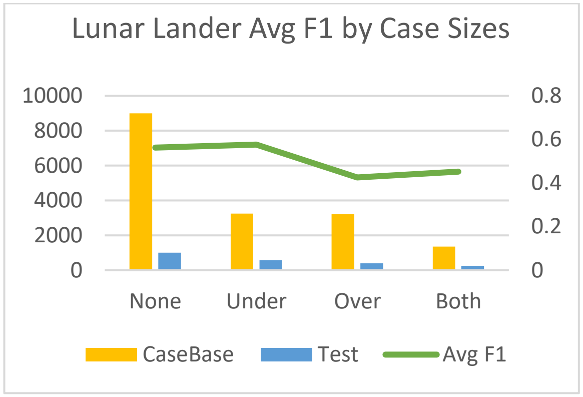


*Figure 4: Lunar Lander trade-off between Filtering and Accuracy*

In this scenario, using both Oversampling the "Support Vector" between similarity clusters, and Undersampling the overrepresented space improves both the case size requirements (by reducing them), but also slightly improves the overall accuracy. This may demonstrate the utility of these filters when dealing with problems in higher dimensions, and is an area we would like to explore in the future.

### Visual Observations

In addition to the qualitative methods as discussed, we want to judge the effectiveness of an agent based on approximation to the example (teacher) agent, as well as how human-like the learning behavior. Repeated observations of the agent operating in Gym environments produced some unusual results, all depending if the Gym environment uses a random seed to determine where the agent is situated when the environment is started.

For example, the cartpole and mountain car environments will start off in the same position, with only minor differences in whether the pole is leaning slightly to the left or right. In cases of Oversampling the range of observed teaching behavior, the agent was able to (eventually) self-correct. However, due to the near-optimal performance of the teaching agent, the learner did not have a chance to observe situations that forced the example agent to recover from a near-disaster; in these situations, a similarity function can be used to some degree of effectiveness to learn how to recover. Running the learner in simpler environments using a large case base size, with either no filter or Undersampling, produced Global F-Scores as high as 0.98. In this scenario, the learner might be able to fool a human observer into thinking it is the example agent.

When moving on to more complex environments with a random seed, the learner behavior diverges from the examples with a lower chance of recovery. For example, the Lunar Lander Environment not only randomizes the terrain and landing target, but also supports a difficulty rating; even the easiest default setting assigns the lander starting locations, cartesian velocities, and angular momentum (spin) that can be difficult for event human players to recover from. In situations when the agent started off in an ideal position and orientation as the example agent, the learner was able to land the craft in a similar fashion; however, when the learner had to recover from difficult scenarios, the outcome was almost always failure. This failure may be due to the lack of examples demonstrating how to recover from the variety of possible scenarios, or not enough simulation time for the learner to generate cases of successful recover. Additional work in this area is required to properly explain this phenomenon, and may include techniques such as Active Case Base generation (Michael W Floyd & Esfandiari, 2009) as a solution to these issues.

## Conclusion

This study provided a contribution to the study of Case-Based Reasoning and Learning from Observation by evaluating a basic CBR approach to behavioral cloning using the popular OpenAI Gym framework. We accomplished this by demonstrating a practical application of the jLOAF framework to an existing community-driven research platform, and the creation of a working interface that allows jLOAF users to leverage OpenAI technologies, while overcoming various technical hurdles to enable distributed communications. This research provided a comparison and analysis of the effects of Over- and Under-Sampling on the overall accuracy across various Reasoning strategies, as well as generating a series of baseline examples for future comparison.

Future work in this domain may involve an extension of the testing framework to encapsulate some of the environment-specific requirements such as 3$^{rd}$ party libraries for each environment, and possibly porting the jLOAF system to Python to natively work with OpenAI Universe proxies, and related research projects with DeepMind.

## References

Aamodt, A., & Plaza, E. 1994. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*. IOS Press, 7(1), 39–59.

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5), 469–483.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. 2016. OpenAI Gym. *ArXiv*, 1–4.

Floyd, M. W., & Esfandiari, B. 2009. An active approach to automatic case generation. In *Lecture Notes in Computer Science* (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 5650 LNAI, pp. 150–164).

Floyd, M. W., & Esfandiari, B. 2011. A Case-Based Reasoning Framework for Developing Agents Using Learning by Observation. *Tools with Artificial Intelligence* (ICTAI), 2011 23rd IEEE International Conference On, (September), 531–538.

Floyd, M. W., & Esfandiari, B. 2011. Learning state-based behaviour using temporally related cases. *CEUR Workshop Proceedings*, 829(August).

Friedenberg, J., & Silverman, G. 2005. *Cognitive Science: An Introduction to the Study of Mind*. Thousand Oaks, Cal.: Sage Publications.

Ontañón, S., Montaña, J. L., & Gonzalez, A. J. 2014. A Dynamic-Bayesian Network framework for modeling and evaluating learning from observation. *Expert Systems with Applications*, 41(11), 5212–5226.

Russell, S., & Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall.

Sutton, R. S., & Barto, A. G. 2012. *Reinforcement learning* (2nd ed., Vol. 2). Cambridge, Massachusetts: The MIT Press.