A User Study on Learning from Human Demonstration

Brandon Packard, Santiago Ontañón
Drexel University
Philadelphia, PA, USA
{btp36,so367}@drexel.edu

Abstract
A significant amount of work has advocated that Learning from Demonstration (LfD) is a promising approach to allow end-users to create behaviors for in-game characters without requiring programming. However, one major problem with this approach is that many LfD algorithms require large amounts of training data, and thus are not practical for learning from human demonstrators. In this paper, we focus on LfD with limited training data, and specifically on the problem of active LfD where the demonstrators are human. We present the results of a user study in comparing SALT, a new active LfD approach, versus a previous state-of-the-art Active LfD algorithm, showing that SALT significantly outperforms it when learning from a limited amount of data in the context of learning to play a puzzle video game.

1 Introduction
LfD has been proposed many times as a solution to the problem of behavior authoring (and often within the context of games, employing tactics such as Inverse Reinforcement Learning (Tastan and Sukthankar 2011), Neural Networks (Stanley et al. 2005), or C4.5 decision trees (Young and Hawes 2014), to name a few). However, most current LfD approaches assume access to a large amount of training data, which is not always feasible. If LfD is to be used to solve behavior authoring in games, this would imply the human author would have to demonstrate the desired behavior an unreasonable number of times in order to generate enough training data. This paper extends previous work on SALT (Packard and Ontañón 2017), a Learning from Demonstration (LfD) approach designed to investigate how to reduce the amount of training data required in LfD. The long term goal of our work is to design LfD approaches that can learn from human demonstrators, and thus require a limited amount of training data.

SALT has already been found to perform competitively with other state-of-the-art algorithms in multiple domains (Packard and Ontañón 2018) when using synthetic demonstrators, but the focus of this paper is an analysis of SALT with human demonstrators in learning how to play video games.

Specifically, we report on the results of a user study where different human subjects tried to teach a learning agent how to play a puzzle game called Thermometers. We compare SALT’s performance to a state-of-the-art LfD algorithm, DAgger (Ross, Gordon, and Bagnell 2010) on 6 factors: mental effort required during training, enjoyability of training, perceived learning performance, reason for ending training, length of training, and actual learning performance. Like recent LfD approaches such as DAgger, the key idea behind SALT is to use an active-learning (Krogh, Vedelsby, and others 1995) approach to LfD, where the learning agent initially learns from a small set of demonstrations, and then, as it performs the target task, it might request additional data from the demonstrator if the learning agent believes that the situation at hand is significantly different from any situation seen in the training data. The key problem to address is when to request more training data from the demonstrator in order to maximize learning performance while also minimizing the mental effort imposed on the demonstrator for providing the requested additional data. Our experimental results show that, at least in our target domain, in situations where training data is limited because the learning agent is trying to learn directly from a single human, SALT learns faster and imposes a lower cognitive burden on the demonstrator than DAgger. As not much work has been done in comparing active LfD algorithms on human demonstrators, DAgger was chosen for this study as it provides inspiration and building blocks for many other state-of-the-art active LfD methods (SALT (Packard and Ontañón 2017), SafeDAgger (Boularias, Kobel, and Peters 2011), RAIL (Judah et al. 2014), and SHIV (Laskey et al. 2016)) and is therefore a good place to start comparisons.

The rest of this paper is organized as follows. After describing some background in Section 2, our algorithm and experimental setup are described in Sections 3 and 4, and our experimental results in Section 5. The paper concludes with discussion/conclusions and future work in Section 6.

2 Background
LfD is very common in humans (Schaal 1997; Heyes and Foster 2002), who often look to a teacher for information on how to perform a task. The overall goal of LfD is, given training data consisting of a set of traces gotten from a demonstrator, derive a policy which allows the learner to
choose an action based on a current observed world state (and which will match the demonstrator’s policy as closely as possible). The reader is referred to Argall et al. (2009) for an overview and formal definition of LfD.

Many LfD approaches in the literature are based on supervised learning, often achieving low performance since they ignore the fact that LfD violates the i.i.d. assumption\(^1\) (Ross and Bagnell 2010). Some existing algorithms attempt to account for this, such as DAgger (Ross, Gordon, and Bagnell 2010) and SMILe (Ross and Bagnell 2010). However, these might require a large mental effort from the demonstrator, limiting their applicability when the demonstrator is human. DAgger, for example, is an active learning method which trains an initial learner based on a demonstrator performing the task, and then lets the learner collect additional training data by allowing the learner and the demonstrator to take turns in controlling the simulation. The demonstrator is later requested to provide correct actions for the states where the learner was in control (state re-labeling), and all the actions generated by the demonstrator are added to the training set. In the experiments reported by Ross, Gordon, and Bagnell (Ross, Gordon, and Bagnell 2010), DAgger required data from a demonstrator playing 660 Super Mario levels before the learner’s task performance plateaued.

This illustrates that one of the current problems in LfD is that existing algorithms often require too much training data to be practical for human demonstrators. Active-learning LfD approaches tend to exacerbate the problem—not only does the demonstrator need to provide demonstrations, but they also need to be able to respond to queries that the learner makes. While work on LfD can be traced back several decades, not as much work has been done on trying to reduce the amount of training data needed in the context of LfD. Some exceptions include SafeDAgger (Zhang and Cho 2016) or RAIL (Judah et al. 2014). SafeDAgger, for instance, attempts to reduce the mental effort placed on the demonstrator by attempting to learn a reference policy that states what actions the demonstrator would take without having to consult the demonstrator, and querying the demonstrator when the learner’s policy deviates too far from this reference policy. This reduces the amount of required training data but does not focus on being suitable for human demonstrators. This is because the amount of training data is not the only relevant factor when using a human demonstrator. For example, requesting the demonstrator to provide a large number of very short demonstrations can be more difficult for a human demonstrator than providing a single very long demonstration, since that would involve a large number of context switches (As shown by Rogers and Monsell (1995), introducing context switching increases reaction time and error rate in humans).

Concerning minimizing demonstrator mental effort, in addition to the previously discussed SafeDAgger, Boularias, Kober, and Peters (2011) tackle LfD with Inverse Reinforcement Learning (IRL). Specifically, they focus on when demonstrations only cover a small portion of a large state space. They propose a model-free algorithm based on Relative Entropy which is able to learn reward functions close to those of the demonstrator using a relatively small set of demonstrations. Another work with this focus is that of Floyd and Esfandiari (2009). Their goal is to create sequences of problems to show to the demonstrator. By giving the demonstrator an entire sequence of actions, they attempt to provide context to the demonstrator, allowing it to more easily provide new data. In their application domain of Robot Soccer, this also increases how accurately the learner selects actions.

Another active LfD algorithm which attempts to reduce the amount of needed training data is RAIL (Judah et al. 2014). Like SALT, RAIL seeks to help account for the i.i.d. violation via demonstrator queries after the initial learner has been training. However, RAIL assumes that the learner has access to a simulator of the domain, which SALT does not require. A similar approach that also requires a simulator is seen in SHIV (Laskey et al. 2016), only instead of using novelty or uncertainty calculations, a risk calculation is run on each state. If the state has too high of risk, then the demonstrator is queried to determine what to do. More general methods for reducing training data include novelty reduction and uncertainty reduction (Silver, Bagnell, and Stentz 2012), which seek to ask the demonstrator for more demonstrations using sampled problems that are considered to be too different from previously seen states or for which the learner is too uncertain about what to do in them. For their robot navigation domain, they find this requires less demonstrator interaction while getting improved results.

3 Selective Active Learning from Traces

Let us illustrate the overall idea behind SALT. Initially, the demonstrator provides a set of demonstrations that form the initial training set. A demonstration is a sequence of state/action pairs representing the state of the world at a given time and the action the demonstrator performed. Let \(D_t\) be the distribution of states in the initial set of demonstrations provided by the demonstrator, and \(D_t\) be the distribution of states the agent would encounter when executing the learned policy. Since the states that the agent will encounter depend on the learned policy, LfD violates the i.i.d. principle and \(D_t\) can potentially be very different from \(D_t\). As has been shown in the literature, this can cause the prediction error in LfD to compound when using standard supervised learning methods (Ross and Bagnell 2010). SALT is an iterative algorithm which attempts to make \(D_t\) and \(D_t\) as similar as possible. Like in DAgger, the learning agent and the demonstrator share control of the task. The main difference from DAgger is in how the learner and the demonstrator share control. SALT monitors whether the current state the learner is in is within \(D_t\). If the state is not in \(D_t\), the demonstrator is given control until the state is brought back to \(D_t\). Training data is generated only when the demonstrator is in control, so unlike DAgger, there is no stochastic mixing or state re-labeling required – which helps reduce human demonstrator mental effort. After each iteration, the new training data is added to the training set, and the learning agent is retrained.

\(^1\)That instances from the training and test set are independently and identically distributed.
The key problem of SALT is therefore determining when to give control to the demonstrator, and when to give it back to the learner. Three strategies are used for this:

- $\rho_s$: Determines when the learner has moved out of $D_i$.
- $\rho_b$: When control is given to the demonstrator, it might be interesting to back-up the world for a few time instants to collect training data on the sequence of states that led to the learner falling outside of $D_i$. This strategy determines how far to back up the world state.
- $\rho_d$: Determines when to give control back to the learner.

Algorithm 1 shows a detailed description of SALT. Specifically, the first of $N$ iterations has the demonstrator control, and those demonstrations become the initial training data. For the remaining $N-1$ iterations, the learner performs the task $C$ times (each of which is called a “trace”). Algorithm 2 shows how the three strategies are used to alternate control between the learning agent and the demonstrator when executing each trace. Specifically, the learner performs the task until strategy $\rho_s$ determines that the learner has moved out of $D_i$. When this happens, the state is backed up a certain number of ticks, as determined by $\rho_b$, and then the demonstrator is given control until the state goes back into $D_i$, as determined by $\rho_d$. The states that the demonstrator encounters and the actions it takes are added to the training data for the next iteration. For SALT, any supervised learning method could be used as the underlying learning method, but for our experiments we tested with J48, A modified version of C4.5 (Quinlan 1993) using WEKA (Witten et al. 2016).

In our empirical evaluation, we tested a single variant of each strategy (the SALT configuration shown to work the best according to our previous work (Packard and Ontañón 2018)). These are described below.

**Algorithm 1 SALT($\rho_s$, $\rho_b$, $\rho_d$, $C$, $N$)**

1: Sample $C$-step trajectories using $\pi^*$ (the demonstrator’s policy)
2: Initialize $D \leftarrow \{(s, \pi^*(s))\}$ – all states visited by the demonstrator and the actions it took
3: Train classifier $\pi_1$ on $D$
4: for $i = 1$ to $N$ do
5: Initialize $D_i \leftarrow \emptyset$
6: for $j = 1$ to $C$ do
7: $D_i = D_i \cup \text{runOneTrace}(\pi_i, \rho_s, \rho_b, \rho_d)$
8: end for
9: Aggregate datasets: $D \leftarrow D \cup D_i$
10: Train classifier $\pi_{i+1}$ on $D$
11: end for
12: **return** best $\pi_i$ on validation data (based on task reward)

### 3.1 SALT Strategies

We examined one possible variant of $\rho_s$, which determines when the learner has moved out of $D_i$:

- $\rho_s^{QBC-m}$ (Query By Committee): Train $m$ classifiers, each on 20% of the training data selected at random. Have each smaller learner predict an action; if those actions are not all the same, signal that the learner has exited $D_i$. For this work, we set $m = 5$.

We also examined one possible variant of $\rho_d$, which determines for how long to give control to the demonstrator:

- $\rho_d^{QBC-m}$ (Query By Committee): Train $m$ classifiers, each on 20% of the training data, selected at random (just as in $\rho_s^{QBC}$). Have each smaller learner predict an action, and signal to give control back to the learner if they all agree on a move. For this work, we set $m = 5$.

Finally, one variant of $\rho_b$ was used:

- $\rho_b^0$ (Back-0) – Does not back up the world state.

**Algorithm 2 runOneTrace($\pi$, $\rho_s$, $\rho_b$, $\rho_d$)**

if not outside of $D_i$ according to $\rho_s$ then
sample using $\pi$
else
back up world according to $\rho_b$
sample using $\pi^*$ according to $\rho_d$
end if
**return** $\{(s, \pi^*(s))|s \in S^*\}$, where $S^*$ is the set of all states where $\pi^*$ was used.

### 4 Experimental Setup

In order to evaluate the performance of the algorithms in terms of human demonstrators, we used a puzzle-game domain known as Thermometers (Figure 1), where the player sees a board with a collection of thermometers of different lengths and orientations, and needs to figure out how full each of the thermometers is based on a collection of row and column constraints. In Figure 1, the board has been filled in. Note how every tile has been marked as either full (red) or empty (bluish-white), and none of them are still marked as blank (black). Additionally, the number of filled pieces in every row or column matches the number above/beside that row or column, and every thermometer that is partially filled is filled starting from the circular bulb and working up towards the cap. Because all of the tiles are marked as filled or

Figure 1: A screenshot of the Thermometers puzzle game.
empty and all of the thermometers are filled from the bulb up, this is a valid solution to the board. For the Thermometers domain, we tested with boards of size 5x5.

Specifically, we use a simplified version of this domain known as Simple Thermometers, where only one row or column of the board can be seen at any given time. States in this domain are represented as a vector of 14 features representing a single row or column of the board, and the actions are also only related to the current row or column. There are 12 actions in total for a 5x5 board:

- fillTile(Tile): Fill a tile in the current row/column, where Tile can be any value 0-4 for 5x5 boards
- emptyTile(Tile): Empty a tile in the current row/column, where Tile can be any value 0-4 for 5x5 boards
- moveToNext: Move to the next row or column on the board.
- clear: Set all tiles in a row/column to undetermined and move to the first column (if previously looking at a row) or row (otherwise).

Therefore, the task of the underlying learner is to predict which of the 12 actions to perform given the values of the 14 features. Figure 2 shows and explains an example of a state/action pair that is recorded during training. Each iteration is a single board, and has a maximum move limit of 100 moves before the iteration forcibly ends. The reward function (used for validation of learning the task) for the Simple Thermometers domain is simply the overall percentage of constraints satisfied for the current board, where there are two kinds of constraints: The number of filled pieces in a row or column matches the number for that row or column, and that each thermometer has a legal configuration (meaning that it is filled starting from the circularly shaped bulb, and all filled pieces are adjacent).

The study itself has two phases (with a researcher overseeing both): In the first phase, the human demonstrator practices the puzzles without training a learner, to get used to the game and interface. Once both the demonstrator and the overseer feel confident in the demonstrator’s puzzle solving ability (in the case of the overseer, this means the demonstrator was able to solve 3 puzzles in a row during training without making any incorrect moves), they move onto phase two. In phase two, the human demonstrator trains a learner using SALT and a learner using DAgger, one after the other. The algorithms are anonymized, so they are not aware of which algorithm is which. A total of 21 participants were gathered: 10 trained SALT then DAgger, 10 trained DAgger then SALT, and one trained SALT then DAgger then SALT again (this user felt that they had made many more errors when training SALT than when training DAgger, and wished to train it again making less errors). Since this user was an outlier, being the only one to train a method twice, their results were removed from the final results. Therefore, the results shown are for 20 users (10 for each order of training). Of these 20 participants, 17 were Computer Science students in various levels of study, and 3 were people with basic knowledge of computers but no programming experience. This was done to make sure a wide range of programming experience levels were represented in the study. The minimum age was 18, and the maximum age was 62, with the median age being 20. 75% of the 20 included users identified as male, and 25% identified as female. Training for each algorithm continued until the user decided they were done - no minimum number of boards was enforced, but there was a maximum of 25 boards per algorithm. After training the first algorithm, the participant is asked to fill out a survey which asks for the following information:

- Programming Experience: The amount of programming experience they have, where 0 is none at all and 5 means they consider themselves an expert.
- Mental Effort: Participants were asked to rate the mental effort required to train each learner. Ratings were gathered on a scale of 1 to 5, where 1 was very little mental effort required and 5 was a lot of mental effort required.
- Perceived Learning: Participants were asked to rate how well they believed each algorithm learned from the training process and the final task reward on validation boards. Ratings were gathered on a scale of 1 to 5, where 1 means the algorithm learned to perform the task very poorly and 5 means the algorithm learned to perform the task very well.
- The Reason They Ended Training: Three default values were provided (“I got bored”, “I felt like it had learned the task”, and “I felt like it wasn’t learning”), but users could also enter their own response.
- Enjoyability: Participants were asked to rate how enjoyable the algorithm was to train comparing if they had to program their own puzzle solver by hand (since different levels of programming experience might favor one or the other, we attempted to sample users with a variety of levels of programming experience). Ratings were gathered on a scale of 1 to 5, where 1 was far less enjoyable and 5 was far more enjoyable.
- Any other comments they may have.

After training the second algorithm, the participant is asked to fill out a survey which contains the same information for the second algorithm, and also re-evaluate the first algorithm now that they have experienced both. The results reported use their final scores for each algorithm (the ones taken from the second survey).

In addition to this, we also captured two pieces of data from the training process:
5 Experimental Results

This section details the results of the study, examining the six pieces of data that help capture feasibility for human demonstrators: Mental Effort, Enjoyability, Perceived Learning, The Reason for Ending Training, Number of Boards Trained, and Actual Learning.

5.1 Mental Effort

First, let us examine the metric of Mental Effort. As can be seen from Table 1, users rated SALT a 2.75 on the 1 to 5 scale and DAgger a 4.65. This means that users placed SALT almost directly in the middle for this metric, but rated DAgger as requiring a very high amount of mental effort, providing strong evidence that SALT imposes less mental burden on human demonstrators than DAgger. From the comments made during the study, our hypothesis for this is that users found it jarring to make a move and then be in a different state than the one that they expected (this happens with DAgger since the user is not always in control, but never with SALT).

5.2 Enjoyability

Next, let us examine the second metric recorded directly from the survey: how enjoyable the algorithm was to train compared to if they had to program a puzzle solver by hand. As can be seen from Table 1, users rated SALT much higher in this metric than DAgger (a 3.75 compared to a 1.75 out of 5). This means that they rated SALT as somewhat enjoyable and DAgger as very unenjoyable, providing further evidence of SALT’s feasibility for human demonstrators. We believe this is also due to the users ending up in a state different than they expected. Because of this, many of them commented that they felt like DAgger wasn’t listening, even though they knew it was.

5.3 Perceived Learning

The third metric recorded directly from the survey was that of perceived learning. Table 1 shows us that users rated SALT as a 3.55 and DAgger as a 1.40. Therefore, although users didn’t find SALT to learn extremely well, they perceived DAgger to learn very poorly. We hypothesize that
this is because while sharing control with the learner during training, DAgger made more moves that seemed “odd” or “illogical” to the user than SALT. Since it doesn’t matter how effortless or enjoyable the users find an algorithm unless it can also learn to perform the necessary task, SALT being perceived to perform better provides evidence for its feasibility for learning from human demonstrators. Furthermore, given the very small amount of training data in this domain, SALT is not just perceived to learn better, but actually does learn better (this is discussed in detail in Section 5.6).

5.4 Reason for Ending Training

The fourth and final metric recorded directly from the survey was the reason the user stopped training each algorithm. Table 2 shows the aggregated results for this question. As can be readily seen, users stopped training SALT for a variety of reasons. Three people stopped because they believed the algorithm had learned the task and 3 people stopped because they felt that it wasn’t learning. The biggest reason users stopped training SALT was because they got bored, and 5 people only stopped training SALT because they either hit the cap of 25 training boards or ran out of time and had to leave. For DAgger, however, almost 75% of users stopped training because they believed that the algorithm wasn’t learning the task, and 0 stopped because they believed the algorithm successfully performed the task. It is also interesting to note that 4 users explicitly stated they stopped training DAgger because they got too frustrated with it, where none did so for SALT. This reasoning provides more evidence that users experienced less mental burden for and perceived better learning with SALT than with DAgger.

5.5 Boards Trained

This metric was not directly placed on the survey, but rather gathered from the users’ training data. Table 1 also shows us that users trained SALT for around 5 boards longer on average than DAgger (12.70 boards compared to 7.65 boards). This provides implicit evidence that SALT was more feasible for the users, as they were willing to train it for much longer than they were with DAgger.

5.6 Actual Learning

Finally, let us examine how well each algorithm actually learned from the users. Figure 3 shows the average task reward (with a 2-value simple moving average) for SALT and DAgger when learning off of the human demonstrators. It can readily be seen that SALT initially trains much faster than DAgger, and then continues to (slowly) grow overall. DAgger, however, stays pretty flat overall and even seems to lose performance as training progresses. In our previous work (Packard and Ontañón 2018) we have also found this to be the case with synthetic demonstrators, where SALT outperforms DAgger in this domain for significantly larger amounts of training data, and DAgger seems to lose performance. We believe this is because the current world state representation of the domain makes this a very hard problem for LfD to solve, which makes DAgger struggle. We also verified that in a simpler setting of the domain (using 3x3 boards instead of 5x5 boards) that DAgger’s performance eventually starts increasing after collecting enough training data from synthetic demonstrators (Packard and Ontañón 2017).

6 Conclusions

This paper presented SALT, an active LfD algorithm designed specifically to study active learning policies that can reduce the amount of training data required to learn as well as demonstrator burden. Specifically, we examined its performance versus DAgger when training an AI in a puzzle game with human demonstrators via a user study. The major result is that SALT performs more desirably than DAgger in six metrics, including amount of mental effort required to train and learning performance, providing strong evidence that it is more feasible for human demonstrators than DAgger. The insights uncovered through this study on why SALT appears to impose less mental effort on human demonstrators than DAgger can help improve SALT and other human-centric algorithms in the future (for example, ensuring that when a demonstrator provides an action, they do not end up in a state other than what they expect due to the learning algorithm). We also found that human demonstrators are not perfect (as is to be expected). Not only do human demonstrators make mistakes, but they also make moves that are logical but inconsistent (for example, filling an entire row from left to right one time, and then right to left another time). Therefore, learners on human demonstrators could be improved not only by reducing noise caused by errors, but also being able to take these inconsistencies and use them to learn a more general or higher level action (such as filling an entire row regardless of order).

In the future, we would like to continue exploring additional SALT strategies, focusing on variants which can account for these inconsistencies while learning and are able to boost learning while not imposing more mental effort on human demonstrators. We would also like to run more user studies to compare SALT against other state-of-the-art baselines or in other domains, to further explore its effectiveness for human demonstrators and gain further insights about the difference in SALT’s performance versus other state-of-the-art algorithms.

References


Heyes, C., and Foster, C. 2002. Motor learning by observation: Evidence from a serial reaction time task. The


