Towards Learning From Stories: An Approach to Interactive Machine Learning

Brent Harrison and Mark O. Riedl
School of Interactive Computing, Georgia Institute of Technology
Atlanta, Georgia, USA
{brent.harrison, riedl}@cc.gatech.edu

Abstract

In this work, we introduce a technique that uses stories to train virtual agents to exhibit believable behavior. This technique uses a compact representation of a story to define the space of acceptable behaviors and then uses this space to assign rewards to certain world states. We show the effectiveness of our technique with a case study in a modified grid-world environment called Pharmacy World. The results show that a reinforcement learning agent using Q-learning was able to learn a policy that results in believable behavior.

Introduction

One of the primary drawbacks of most machine learning algorithms is that users require in-depth knowledge about artificial intelligence and statistics to use them effectively. As such, one of the most sought after goals in machine learning is to allow non-experts to interactively train machine learning algorithms. *Interactive machine learning* (Chernova and Thomaz 2014) seeks to enable this by allowing humans to directly interact with machine learning algorithms by way of online feedback or demonstrations of behaviors. One issue with these techniques is that they can require humans to interact with these algorithms in ways that may be unnatural. A machine learning algorithm may require examples of failure states in order to execute properly, or a person may be forced to give feedback in unfamiliar states while training. By allowing for a more natural means of communication between humans and algorithms, we would further improve the ability of non-experts to train machine learning algorithms.

In this work we present the idea of using an unexplored source human communication to train machine learning algorithms: stories. Specifically, we will discuss how stories can be used to train agents to exhibit desirable human behavior. Stories have the potential to provide many advantages in training machine learning algorithms. Stories play a large role in how we as humans try to make sense of the world around us. They allow us to comprehend complex situations as well as explain complex ideas to others (Bruner 1991). If many of our interactions with other humans are by way of storytelling, then it would make communication between humans and machine learning algorithms more natural if the machine learning algorithms could use the stories told similarly to how they use demonstrations.

Our technique, which we call learning from stories (LfS) uses stories told by humans to help train virtual agents to exhibit specific behaviors. This task is similar to that of learning from demonstration (LfD) algorithms, except that instead of learning from explicit demonstrations of a task our technique will learn using stories told about said task. The primary difference between stories and demonstrations, as well as the primary difficulty in dealing with stories, is that stories are more unconstrained than demonstrations. Many different stories can be told about completing the same task and all be correct. Also, it is not uncommon for humans telling stories to skip steps in a story.

Our technique addresses this problem by first converting a set of exemplar stories to a *plot graph*, a compact and canonical representation of a space of stories. From there, our technique uses this plot graph to define the space of acceptable behaviors, which are then turned into reward functions that can be used to train reinforcement learning algorithms. This reward function offers guidance as to how the agent should behave while still allowing the agent to learn optimal behaviors for its environment. This allows the agent to deviate from the plot graph in extreme situations. To explore the effectiveness of our technique, we present a case study in which we train a reinforcement learning agent to exhibit believable behavior in Pharmacy World, a modified grid world in which the agent attempts to acquire drugs from a pharmacy.

Related Work

The type of interactive machine learning that is most closely related to our own work is a specific type of learning from demonstration known as *Inverse Reinforcement Learning* (IRL). In typical reinforcement learning, an agent attempts to compute a policy that describes the best action to take in any given state. IRL, on the other hand, attempts to learn the reward function that best describes a corpus of policy examples (Ng and Russell 2000) or policy trajectories (Abbeel and Ng 2004). Early work in this area required either complete policy examples or complete trajectories in order to learn. This requirement was relaxed in through the introduction of techniques such as Bayesian IRL (Ramachandran and Amir 2007) and maximum entropy IRL (Ziebart

et al. 2008). There has also been work on relaxing the assumption that all example policies or trajectories are correct (Grollman and Billard 2011). There has been an influx in work that seeks to derive reward functions or behaviors from natural language commands (Lignos et al. 2015; MacGlashan et al. 2015).

LfS is fundamentally different than the problem posed in IRL. While we are attempting to derive reward functions based on a corpus of examples, we make different assumptions about what these examples represent which leads to a different understanding of how to approach the problem. In this work, we assume that our example stories define a space of believable behaviors. Rather than design a reward function that can reproduce all of these examples, we are trying to create a reward function that produces behaviors that fall within this space.

Preliminaries

In this section, we review two concepts that are critical to our work: plot graphs and reinforcement learning.

Plot Graphs

In order to create an agent that exhibits believable behavior, we must first define what that behavior is. In this work, our goal is to use behaviors encoded in a set of stories to define the space of acceptable behaviors for our agent to follow. This presents a problem as it is likely that multiple different stories can be told all describing the same scenario. In addition, the set of stories initially used to define desirable behavior could be quite noisy (especially if different authors wrote different stories). Different people may skip events in the story or use different language to convey the same event. Some stories may contain errors or events that do not make sense given the scenario being described. To help mitigate these problems, we have chosen to first convert the set of exemplar stories into a plot graph as used in (Li et al. 2013).

In that work, the authors convert a set of crowdsourced stories about a scenario into a plot graph, which is a tuple $G =< E, P, M >$ where E is the set of events that occur in the story, $P \subseteq \{x \rightarrow y | x, y \in E\}$ is a set precedence relationships over pairs of events, and $M \subseteq \{(x, y) | x, y \in E\}$ is a set of mutual exclusion relationships. Also sometimes included is a set of optional events $O \subseteq E$.

Precedence relationships between events are a temporal ordering that defines when certain events are allowed to occur. Consider the plot graph in Figure 1(a). In this graph, a precedence relationship exists between event *c* and event *e*. This means that the stories that can be told in the space represented by this plot graph must have event *c* occur before event *e*. Note that this does not mean *c* must happen immediately before *e*. Mutual exclusion relationships exist between events that will never co-occur in the same story. Returning to the example plot graph in Figure 1(a), the mutual exclusion relation ship between event *a* and event *b* ensures that they will never occur in the same story. In the event that two mutually exclusive events share a precedence relationship, then only one of the preceding events need to occur in order for the following event to be executed. Optional events are

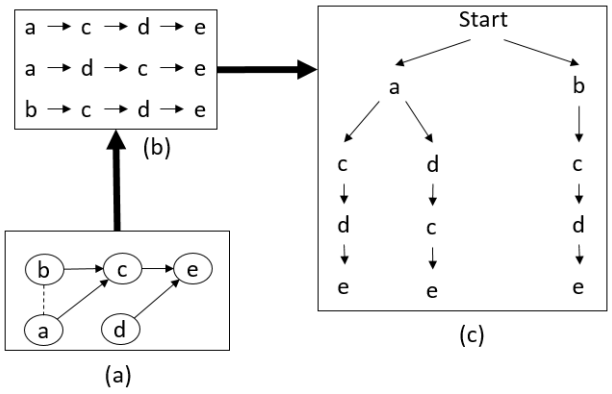

Figure 1: An example of how a plot graph (seen in a) produces a set of stories (seen in (b)) that can be represented by a trajectory tree (seen in (c)). In (a), ovals represent events, arrows between events are precedence relations, and dashed lines between events are mutual exclusion relations.

those that do not need to be executed in order to tell a legal story. Typically these are learned during construction of the plot graph.

Construction of the plot graphs involves a pairwise examination of events to determine if a precedence relationship or a mutual exclusion relationship needs to be inserted. Placement of either of these relationships is determined by examining the initial corpus of stories. Thus, this can be used to effectively filter out stories that contain noise due to an erroneous ordering of events or through ommitted events.

Reinforcement Learning

The goal of this work is to be able to use stories to define reward functions that can be used to train reinforcement learning agents. Reinforcement learning (Sutton and Barto 1998) is a technique that is used to solve a Markov decision process (MDP). A Markov decision process is a tuple $M =< S, A, T, R, \gamma >$ where S is the set of possible world states, A is the set of possible actions, T is a transition function $T : S \times A \rightarrow P(S)$, R is the reward function $R : S \times A \rightarrow \mathbb{R}$, and γ is a discount factor $0 \leq \gamma \leq 1$.

The goal of reinforcement learning is to find a policy $\pi : S \rightarrow A$, which defines which actions should be taken in each state. Ideally, the reinforcement learning agent will take actions that will maximize its reward as given by the reward function. In this work, we use Q-learning (Watkins and Dayan 1992), which uses a Q-value $Q(s, a)$ to estimate the expected future discounted rewards for taking action a in state s. There are several reasons that we use reinforcement learning for this task. First, it allows the agent to fill in any gaps that may exist in the stories that are used to train it. These gaps exist because the storytellers do not know the specific details of the environment. Therefore, it is possible that the agent will need to take several actions, such as those needed for navigating a virtual world, in between plot points. Reinforcement learning also allows the agent to deviate from the plot graph if doing so will allow it to more

efficiently reach a goal state. Reinforcement learning also allows the agent to learn in stochastic environments, which they are likely to exist in.

Methodology

Our technique for using stories to derive reward functions begins with a corpus of stories each describing the same scenario. This corpus, as it is typically crowdsourced, is likely very noisy and, thus, difficult to learn from. In order to reduce the noise in this corpus we convert the set of stories into a plot graph using the technique outlined in Li *et al.* (2013). This plot graph now encodes procedural knowledge about the desired behavior that our agent should exhibit. One assumption that we make about this plot graph is that each node in the plot graph refers to a state or action that exists in the MDP. We believe that this condition can be relaxed, but we leave that discussion as future work.

Once we have created this plot graph, we can then convert it into a *trajectory tree*. A trajectory tree is a tree structure that encodes every possible story that could be told using the plot graph. Figure 1(c) shows an example of a trajectory tree. This figure shows three of the stories that could be produced by the sample plot graph given in Figure 1(a) and how they would be represented as a trajectory tree.

We then incorporate this trajectory tree into the environment state to keep track of what the agent has done and what it still needs to do. The insight here is that each node in the trajectory encodes a unique story up until that point. In this way, we manage to encode action history into an environment that is still Markovian. This does, however, make the state space larger, which means it will take longer to train. This is acceptable as our primary objective is to produce believable behaviors.

We assign rewards based on what branches the story can take. Since the agent's location in the story is encoded as part of the state, we know how which unique story the agent is currently following. We also know the different ways that the story can unfold by examining the tree. Since we also know that each node in the plot graph (and thus each node in the trajectory tree) exists inside of the MDP, then we can assign rewards to states that would continue the story.

What is important to note is that we do not consider every story produced by the plot graphs to be optimal. Unlike IRL, we do not expect for our agent to necessarily recreate every story during normal execution. Instead, these stories define the *space* of acceptable behaviors. The agent is then allowed to reason about them and choose between behaviors based on the environment that it exists in.

To our knowledge, there is no definitive technique for assigning reward values. Agent behavior is highly sensitive to the environment that the agent exists in, making it difficult to determine a set of reward function that works for every environment. We can, however, weight rewards based on their relative importance compared to other events in the plot graph. Here, we define an event's importance as the percentage of stories that the event occurs in. The intuition behind this is that events that are more important to the success to the outcome of the story will be included more often. Therefore, the agent author must choose a base reward based on the current environment.

Case Study: Pharmacy World

To show the effectiveness of our technique, we have chosen to perform a case study in a modified gridworld called *Pharmacy World*. We will discuss this domain as well as the study performed in more detail below.

Pharmacy World Domain

The Pharmacy World domain is a modified gridworld that is loosely based on the innocuous activity of going to the pharmacy to purchase drugs.

Pharmacy World contains five different locations each located somewhere in the gridworld: a house, a bank, a doctor's office, a clinic, and a pharmacy. Each of these locations, except for the house, contains items that can be used to enable or disable certain actions. The bank contains money that can be used to purchase either weak or strong drugs from the Pharmacy. The doctor's office and the clinic both contain prescriptions that can be used in conjunction with money to purchase strong drugs.

The actions that the agent can take in Pharmacy World include simple movement actions, such as moving Left or Right, actions for entering/leaving a building, and actions that are used to retrieve objects. In order to receive a prescription, for example, the agent must first be examined by a doctor at either the Doctor's Office or the Clinic. There is a small amount of stochasticity in this environment in that this action is allowed to fail with a probability of 0.25. The remaining items can either be stolen directly from their respective locations or acquired through interacting with either the Bank Teller at the Bank or the Pharmacist at the Pharmacy.

The goal in Pharmacy World is to return to the house with either the strong or the weak drugs, with the strong drugs being preferred. The reason that the weak drugs is included is because it is possible to fail to acquire a prescription, which would make it impossible to reach a terminal state without stealing. The agent is always able to purchase the weak drugs in order to reach a terminal state.

The reason that we used this environment is because it is a relatively simple environment that highlights the differences between what a human would do and what a typical agent would do. One would expect a human to get a prescription, withdraw money from the bank, and then purchase the strong drugs; however, an agent that is only rewarded for retrieving the strong drugs would be inclined to steal since that takes the fewest number of actions.

Determining Rewards

In order to assign rewards in Pharmacy World, we first must have a plot graph defining acceptable behaviors. For this case study we chose to manually construct a plot graph and use that to create the acceptable stories. Our synthetic plot graph, seen in Figure 2, generates 213 stories, which can be used to generate a trajectory tree containing 827 nodes. As you can see, this plot graph manages to encapsulate the events that you would expect to see in Pharmacy World.

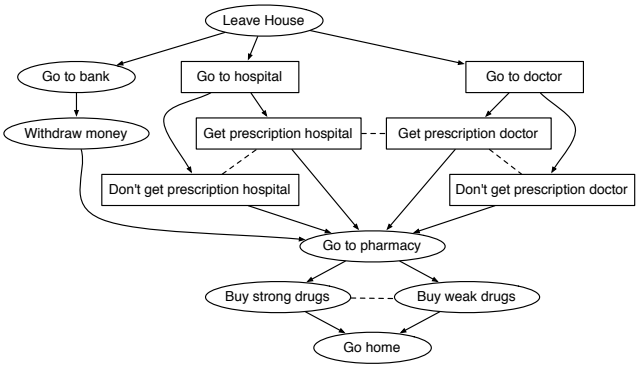

Figure 2: Plot graph for obtaining drugs. Ovals are events. Rectangles are optional events. Precedence relationships are represented by arrows between events. Mutual exclusions are represented by a dashed line between events.

Once the trajectory tree has been created, we manually convert the nodes in the tree into states that existed inside Pharmacy World. For the most part, nodes in the plot graph/trajectory tree directly correspond to actions available in Pharmacy World. The main exception is that *Get Prescription* and *Do Not Get Prescription* plot graph nodes do not correspond to any action in Pharmacy World. This node actually represents whether the *Get Examined* action fails or succeeds, so that is where rewards are assigned.

Since there is no definitive way to determine what the base reward value for this environment should be, we used the value 10 (which was then weighted by event importance) every time the agent moved to a new event in the trajectory tree. This, in practice, produced acceptable policies for Pharmacy World, but is likely domain specific. We leave the problem of automatically determining this value to future work. This base reward value was then further weighted by event importance to arrive at final reward values for each node in the plot graph. For each other possible state, we assigned a reward value of -1.0.

Training

We used Q-learning in conjunction with ϵ-greedy exploration for training. For this study we define ϵ to be 0.8 and then slowly decay it over $200,000$ learning episodes. This way the agent will prefer to randomly explore the environment early, but as training continues it will converge to taking the action it perceives as optimal more often than not. In addition, we use parameters $\gamma = 1.0$ and $\alpha = 0.5$. In order to evaluate the behavior that the agent learned, we examined the policy that the agent learned in order to verify that it existed within the space of accepted stories defined by the plot graph used to create the reward function.

Results

After training for $200,000$ episodes, we examined the policy that the agent learned. Since there is a source of stochasticity in Pharmacy World, there are 3 possible trajectories through the environment depending on the outcome of the *Get Examined* action.

The first case that the agent could encounter is the one in which the *Get Examined* action succeeds on the first attempt. In this case, the agent first navigates to the clinic and gets examined in order to recieve the prescription. Then, the agent navigates to the bank and requests and withdraws the money. Having obtained the money and the prescription, the agent then moves to the pharmacy and requests and purchases the strong drugs. The agent then finishes by returning to the house with the drugs.

The second case involves the first *Get Examined* action failing while the second one succeeds. In this case the agent's policy is the same as its policy in the first case except that instead of going to the pharmacy after retrieving the money, the agent goes to the doctor's office in order to get examined. This action ultimately succeeds, and then the agent navigates to the pharmacy to purchase the strong drugs and then returns home.

The final case is the one in which the agent is unable to obtain a prescription at all due to both *Get Examined* actions failing. In this case, the agent's policy the same as the policy in the second case, except that the agent then chooses to purchase the weak drugs and then return home.

Each of these policies fall within the realm of acceptable behavior as defined by the plot graph we used to generate the reward function. This means that the reward function generated using the plot graphs was successful in generating acceptable behavior in Pharmacy World. Thus, by introducing this reward function we were able to prevent the agent from exhibiting psychotic behaviors that may result from simpler reward functions (such as stealing the drugs rather than purchasing it).

Conclusions

In this work we introduce a technique for learning from stories. By using a plot graph to define a behavior space we can create reward functions that encourages reinforcement learning agents to exhibit behaviors that exist in that space. In doing so, we allow humans to interact with reinforcement learning algorithms using storytelling, which we posit is a more natural means of communication than demonstrations of tasks or giving positive or negative feedback. While we place many constraints on this technique in this paper, we believe that many of these can be relaxed and plan to explore this topic in more detail in future work. In addition to better enabling non-experts to interact with machine learning algorithms, it could also allow humans to convey more complex ideas to machines. It could allow humans to provide natural feedback and guidance during training. It could also possibly be used to enable machines to learn complex sociocultural values embedded in stories.

Acknowledegments

This material is based upon work supported by the U.S. Defense Advanced Research Projects Agency (DARPA) under Grant #D11AP00270 and the Office of Naval Research (ONR) under Grant #N00014-14-1-0003.

References

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

Bruner, J. 1991. The narrative construction of reality. *Critical inquiry* 1–21.

Chernova, S., and Thomaz, A. L. 2014. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(3):1–121.

Grollman, D. H., and Billard, A. 2011. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 3804–3809. IEEE.

Li, B.; Lee-Urban, S.; Johnston, G.; and Riedl, M. 2013. Story generation with crowdsourced plot graphs. In *AAAI*.

Lignos, C.; Raman, V.; Finucane, C.; Marcus, M.; and Kress-Gazit, H. 2015. Provably correct reactive control from natural language. *Autonomous Robots* 38(1):89–105.

MacGlashan, J.; Babes-Vroman, M.; desJardins, M.; Littman, M.; Muresan, S.; Squire, S.; Tellex, S.; Arumugam, D.; and Yang, L. 2015. Grounding english commands to reward functions. In *Proceedings of Robotics: Science and Systems*.

Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, 2586–2591. Morgan Kaufmann Publishers Inc.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Ziebart, B. D.; Maas, A.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*, 1433–1438. AAAI Press.