Learning a Visual State Representation for Generative Adversarial Imitation Learning

Boris Ivanovic, Yuke Zhu, Animesh Garg, Silvio Savarese
Department of Computer Science, Stanford University
{borisi, yukez, garg}@cs.stanford.edu, ssilvio@stanford.edu

Abstract

Imitation learning is a branch of reinforcement learning that aims to train an agent to imitate an expert’s behaviour, with no explicit reward signal or knowledge of the world. Generative Adversarial Imitation Learning (GAIL) is a recent model that performs this very well, in a data-efficient manner. However, it has only been used with low-level, low-dimensional state information, with few results on visual input. This work aims to expand the applicability of GAIL by enabling it to use visual input. To do this, we add a convolutional neural network to GAIL that learns a vector representation of images. We train the entire model on randomly-generated 2D “Grid World” environments with optimal experts. Further, we uncover that GAIL succumbs to the “DAgger problem” and analyze ways to overcome it.

1. Introduction

This work investigates the problem of generating a state vector representation of images of a simple “GridWorld” environment, commonly used in Reinforcement Learning (RL) research. Concretely stated, our problem is designing a function $f_{\text{rep}} : \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^{D}$ to generate a vector representation $v \in \mathbb{R}^{D}$ that encodes any information an agent needs to perform an action given a 3-dimensional state image $S \in \mathbb{R}^{H \times W \times 3}$ representing world state.

Two main methods for solving this task stand out. The first is treating this problem as a byproduct of a classification problem (similar to how semantic image descriptors are a byproduct of training neural networks on ImageNet [5]), where the classes are some aspect of the RL environment (like its type, e.g. hilly, obstacle-filled, etc.). This new problem would be trying to identify the environment type from either the current state, or more likely, a sequence of states and actions. The idea is that a Convolutional Neural Network (CNN) would learn a good state vector embedding along the way, and that we could extract it as our state vector representation.

The second, chosen, method is to train an agent to solve an RL environment from visual input, learning a vector representation of images along the way. Instead of just directly applying currently-used RL algorithms such as policy iteration [21], value iteration [3], Q-learning [20], or newer methods like A3C [12], we decided to explore GAIL. As can be inferred from the name, GAIL features a generative adversarial network (GAN) at its core, as described in [6].

We will append a CNN that learns a vector representation of images to GAIL. Then, we will train an agent with GAIL to solve RL environments and extract the CNN’s final fully-connected layer as our desired state vector representation.

2. Background

Learning a representation for images is a mature field, with machine learning models being applied to recognize documents [9], reduce data dimensionality [7], represent image patches generally [22], etc., with a depth of study on what properties of the learned latent space are desirable and how to obtain them [4].

Imitation learning, on the other hand, is a relatively new pursuit where two main methods stand out. The first is Behavioural Cloning [14], which formulates learning a policy as a supervised learning problem over state-action pairs (from expert trajectories), causing the agent to match those behaviours and literally act like the expert. The second main method is inverse reinforcement learning [13], which finds a cost function under which the expert is optimal, using that cost function to then train an agent with other RL methods. Generative Adversarial Imitation Learning (GAIL) [8] is a recently-released third method, which treats the imitation problem as one of matching agent and expert state-action distributions, without explicitly learning to copy it. This naturally encourages the use of a generative adversarial framework, which contains a generator that learns to match a ground-truth distribution [6].

In this work, we plan to merge the worlds of image representation and imitation learning in order to enable the
recently-released GAIL model to work on image-based environments, using raw pixels as input. Finding a good solution to the state representation problem is crucial since it affects fundamental properties of the underlying RL problem. For example, Maillard et al. (2013) showed that selecting the “right” state representation can lead to Markovian state dynamics, allowing much simpler inference methods to be applied as solutions [1].

3. Method

3.1. Dataset

The specific environment that we used is FaceBook’s MazeBase [17]. MazeBase is a flexible sandbox for creating and running simple 2D games. Further, it allows users to create their own games, with custom rules, reward functions, and goal conditions. The specific games that we implemented are random $10 \times 10$ versions of the following (along with a text description to aid in understanding the game):

1. Navigate to Object: "Go to Goal0"
2. Navigate to Ordered Sequence of Objects: "Go to Goal0, then Goal2, then Goal1"
3. Navigate to Conditional Goal: "If whatever indicator is red, go to Goal0, otherwise Goal1"
4. Navigate with Conditional Rules: "If whatever indicator is red, go to Goal0 via longest path, else go to Goal0 via shortest path"
5. Navigate to Adjacent Location: "Go to the tile left of Goal0"
6. Navigate to Relative Location: "Go to 3 tiles left, 2 tiles down from Goal0"
7. Navigate to Absolute Location (a): "Go to Top-Left"
8. Navigate to Absolute Location (b): "Go to 3,2"
9. Navigate to Specific Tile Type: "Go to a Water tile"
10. Navigate to Calculated Location: "Go to midway between Goal0 and Goal1"
11. Navigate to Abstract Location: "Go to the south half"

We generated 100 expert trajectories for each of these tasks. We only generated 100 because GAIL is very dataset-efficient, being able to solve difficult robotic locomotion tasks (e.g. on the MuJoCo simulator) with 30 - 50 expert trajectories [8]. The actual details of the expert are outlined in Section 3.5.

MazeBase’s internal world state is represented as a 3-dimensional matrix of shape $H \times W \times O$ where $H$ is the grid height, $W$ is the grid width, and $O$ is the total number of objects being considered. The world state is a “many-hot” matrix as each object in the game is represented with a one-hot vector, however, multiple objects can occupy the same space, meaning some positions can have multiple ones. We wrote a custom renderer on top of this world state which generates images representative of the world state. Fig. 1 shows an example of our rendered world, along with example of the extent of what MazeBase can create.

Inferring a state vector representation from this world information is interesting as it requires the encoding to be able to handle multiple objects being in the same place without losing information about the individual objects.

3.2. Network Architecture

Our architecture is depicted in Fig. 2. The input CNN is a two-layer architecture of the form\(^1\) Rendered $53 \times 53$ Image - Conv(32, 5, 0, 2) - LeakyReLU - Conv(64, 3, 0, 2) - LeakyReLU - Flatten - State Representation. The input image size is largely arbitrary, but $53 \times 53$ was chosen as it leads to reasonable activation and filter sizes while maintaining reasonable drawing capabilities (too small and circles would look like blocky diamonds, too large and the activation sizes get bloated with diminishing representational benefits).

The generator is a neural network with architecture\(^2\) State Representation - FC(256) - LeakyReLU - FC(128) - FC(2) - Tanh - Agent Actions. The generator can be viewed similar to a policy network, in that it takes the current state as input and outputs an action. The reason there is no nonlinearity between the FC(128) and FC(2) layers

\(^{1}\)Conv(F, N, P, S) denotes F N x N convolutional filters with zero-padding P and stride S
\(^{2}\)FC(n) denotes a fully-connected layer with n hidden units.
is that we want to allow negatively-valued actions (this will be explained more clearly in Section 3.4) and ReLU nonlinearities have a strong bias for positive activations (even with the LeakyReLU modification, since the negative part of the nonlinearity is often \( \leq 10\% \) of the positive part in magnitude). Further, we use a hyperbolic tangent nonlinearity before the output layer to ensure that the action output covers the same range as the allowed agent actions, keeping in line with a recent follow-up work to GAIL [10].

The discriminator is a similar neural network with architecture:

\[
\text{State-Action Pair} - \text{FC}(256) - \text{LeakyReLU} - \text{FC}(128) - \text{LeakyReLU} - \text{FC}(1) - \text{Output Score}
\]

Here, the output score is a measure of the probability of the input state-action pair coming from the expert policy. Ideally, we would like to have this value close to 0.5 as that means our generator’s actions are exactly matching those from the expert policy and are confusing the discriminator.

A noteworthy aspect of our model is that we do not exactly use GAIL. There was a very recent follow-up work which introduced a modification to GAIL, named InfoGAIL [10]. The key modification is replacing the original GAN with the more-stable Wasserstein GAN (WGAN), leading to an easier time training the model. Although the original authors used InfoGAIL to extract latent state information from an expert, we used it identically to GAIL as a method of training an agent from expert behaviour (we simply remove or zero out any part of the network dealing with latent state extraction). We found that training the original GAN-based model was too finicky and difficult, as a result we switched to this WGAN-based model, as indicated in the top-right of Fig. 2.

### 3.3. Network Training

Instead of the typical cross-entropy or mean-squared error loss metric, we backpropagate two GAIL losses through the CNN to update its weights. The first is the WGAN Discriminator Update that is used to update the InfoGAIL discriminator network:

\[
g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)})) \right]
\]

The weights are updated with RMSprop [13] as in the original WGAN paper [2].

The second is the Trust Region Policy Optimization (TRPO) Loss [16] that is used to update the InfoGAIL policy generator network. We did not reproduce the TRPO equations here as they are quite involved and outside of the scope of this course.

We opted to use these losses instead of an auxiliary measure because we believe that they will lead to a more useful state representation for reinforcement learning since the losses are based on the performance of the agent and its similarity to the expert.

### 3.4. Discrete vs. Continuous State-Action Space

By default, MazeBase is a discrete state-action space. While this is standard for simpler RL algorithms, GAIL and other modern deep RL methods usually work with a continuous state-action space. As a result, we have reimplemented parts of MazeBase to allow both discrete and continuous versions. Discrete versions feature a typical \{left, up,
right, down} action space and grid-based state space. Continuous versions feature a 2-dimensional \((x, y)\) action space (bounded within \([-1, 1]\) on both dimensions, which gets added to the agent’s current location) and a continuous state space allowing any real-valued position within the game bounds.

### 3.5. Expert Policy

GAIL, like all imitation learning, requires an expert to imitate. Instead of training a reinforcement learning agent to solve MazeBase tasks, we decided to hand-implement a heading-following agent.

The heading-following agent subtracts the goal position and agent position and takes a step along that vector. We added zero-mean Gaussian noise to the action to ensure the expert produces multiple different trajectories (for data variety and extra state-space cover).

### 4. Experiments

#### 4.1. Raw Many-Hot State Input

Since the input size is very small (the largest MazeBase grid size is \(10 \times 10\)), we believed that we could use a small CNN directly on its many-hot world state. This belief is in line with the approach that Oh et al. (2017) took in a recent unpublished work, and we settled on using a single-layer CNN with $32 \times 1 \times 1$ filters (no padding or stride). This was followed by a LeakyReLU nonlinearity \(^{11}\) and flattened into a vector to be passed into subsequent fully-connected layers. The reason this CNN does not contain any fully-connected layers is the same as in Section 3.2.

We implemented this simple one-layer CNN described in Section on a discrete \(10 \times 10 \times 3\) world. Sadly, we did not see satisfactory performance. We then tried a two-layer CNN with \(1 \times 1\) filters followed by \(3 \times 3\) filters (to hopefully capture local world information), but saw similar mediocre agent performance. The generator’s trajectories were poor for the trained tasks, frequently not reaching the goal condition, hitting the maximum allowed trajectory length, getting stuck in corners, etc.

Our belief in why this approach did not work is that CNNs are powerful because they exploit local spatial relationships in images. MazeBase’s many-hot world matrix does not have such spatial relationships because neighboring elements are entirely separate squares in the world.

Fig. 3 shows some of the strange losses we observed while training the modified InfoGAIL model with our two-layer CNN.

Fig. 4 shows a t-distributed Stochastic Neighbour Embedding (t-SNE) \(^{19}\) of the vector embedding learned by our model, showing why the discriminator performs so well after 75 training iterations.

#### 4.2. Model Generalization

In order to show that our image-based model generalizes to states that it has not seen before (even states that the expert has not seen before) we devised the following experiment on Game 1: Navigate to Object (“Go to Goal0”).

1. Train an agent with Behavioural Cloning on 1000 expert trajectories, with the idea to overfit and perform optimally (since the agent will just memorize all the optimal actions from most states). Force the goal to be at \((5, 5)\) and force the agent to only spawn within radius 3 of the goal (so that there are leftover unexplored states).

2. Train an agent with InfoGAIL with a warm-start from the 1000-expert-trajectory Behavioural Cloning model. Force the goal to be at \((5, 5)\), but allow the agent to spawn anywhere.

3. Verify that the actions on the states outside of the ones seen by the expert make sense and head toward the goal (as we would expect if the model was generalizing to states the expert has not been to).

Since MazeBase is a relatively simple “Grid World” environment, our learned agent’s performance can be visualized in a quiver plot, shown in Fig. 6. We have also provided the quiver plot for the Behavioural Cloning agent in Fig. 7.

Very surprisingly, we do not see generalization at all. We actually see the agent taking more-or-less random actions in the unexplored states.

We believe that this occurs not because of a fault in the GAIL or InfoGAIL algorithms, but because this “Grid World” environment and expert lend themselves to creating sharply-peaked state-action distributions. An example of how this could occur is shown in Fig. 5. Since the agent learns to match this state-action distribution, it will similarly always take actions to the right of the world. However, if it overshoots the goal (due to some noise in the agent’s output action), the agent will never make it back to the goal since its learned state-action distribution is completely skewed to taking actions to the right (to match the expert). Thus, it will never take actions to the left to correct its overshoot. We call this problem the “DAgger problem” as it is reminiscent of a previous RL method \(^{15}\) that suffered from the inability to generalize because it cannot work on states that an expert has not been to.

The reason this has not been noticed before in previous works is that previous works focus on robotic locomotion tasks, or tasks where the world resets to a constant initial state upon episode end. This means that repeated simulations of the world will not expose the agent to too many
Figure 3. (a) Our model’s discriminator accuracy in identifying generated policies from expert policies per training iteration. (b) A properly-training model’s discriminator accuracy in identifying generated policies from expert policies per training iteration (trained on a different continuous environment, OpenAI Gym’s CartPole). (c) Our model’s discriminator loss per iteration. (d) A properly-training model’s discriminator loss per iteration. The spikes are expected as those are points where the generator had a surge in performance, confusing the discriminator heavily.

Figure 4. A t-distributed Stochastic Neighbour Embedding visualization of expert and policy states, the expert states are all on the right and the policy states are all on the left, showcasing why the discriminator had perfect performance.

Figure 5. A depiction of an environment and expert that lend themselves to creating a sharply-peaked state-action distribution. Here, the expert always takes actions to the right, hitting the goal immediately.

states that the expert has not seen before, whereas our environments are randomly generated with random starting agent and goal positions.

In order to understand how much the “DAgger problem” problem affects our model, we perform a comparison between different policy models in terms of how long it takes, on average, to complete a random layout of Game 1: Navigate to Object (“Go to Goal0”).

1. Our expert agent takes $5.39 \pm 1.87$ actions to reach the goal.
2. A random agent (taking uniformly random actions) takes $155.18 \pm 90.90$ actions to reach the goal.
3. An agent trained only with Behavioural Cloning takes $64.70 \pm 117.68$ actions to reach the goal.
4. An agent trained with our model takes $172.00 \pm 146.42$ actions to reach the goal.

Understandably, our model performs the worst (even worse than random, but this is because it cannot recover from an overshoot). Further, interesting to see is that the agent trained on our model also has the highest variance, this is because sometimes the agent will reach the goal in a perfect, expert-length trajectory, whereas other times it will overshoot and hit the maximum episode length limit (300). This same behaviour can be seen with Behavioural Cloning, although to a lesser extent as its agent performs random actions when it is in a state the expert has not seen before, meaning it will eventually reach the goal and not get stuck in a corner like our model’s agent will.

5. Conclusion

Although we have implemented a model that enables InfoGAIL to use visual input and intended to show that it increases the applicability of InfoGAIL, we instead discovered that InfoGAIL (and by extension GAIL) suffer from the “DAgger problem.” This is not through a fault of the original algorithms, but it is a fact of using distribution matching (like with a generative adversarial framework) with experts that take only one type of action or with environments that lend themselves to these kinds of skewed state-action distributions.
Some future directions include using another imitation framework as well as looking at more explicit multi-task learning approaches to hopefully overcome this “DAgger problem.”

Since GAIL suffers from this “DAgger problem,” it may be beneficial to switch to a new imitation learning framework. However, since Behavioural Cloning suffers from this problem as well (it explicitly trains the agent to memorize the expert’s actions), inverse reinforcement learning [13] is the most likely candidate as it is the only other mainstream method that does not suffer from this problem.

Training an agent to solve more tasks is beneficial for generalization, preventing the agent from memorizing world layouts or exploiting non-optimal trajectories. This is feasible to do with MazeBase since it is a very flexible framework. The idea would be to explore if the “DAgger problem” could be overcome in GAIL from an explicit multi-task learning approach.

References


