Learning Visual Embeddings for Reinforcement Learning

Mihir Rana
Center for Data Science
New York University
ranamihir@nyu.edu

Kenil Tanna
Center for Data Science
New York University
kenil@nyu.edu

Ilya Kostrikov
Courant Institute
New York University
kostrikov@cs.nyu.edu

Rob Fergus
Courant Institute
New York University
fergus@cs.nyu.edu

Abstract

Reinforcement Learning (RL) tasks traditionally struggle with sparse reward environments. A possible solution, although potentially expensive, is to get labelled human expert demonstrations for the agent to imitate and learn from. Instead, we take the approach proposed in [1] and perform an initial self-supervised computer vision (CV) task before the RL one to get an understanding of the physical dynamics of the environment from the already existing limited number of expert demonstration videos, and use this prior knowledge to speed up training and guide the agent. We demonstrate the effectiveness of this method for an agent navigating through 2D grid mazes, and perform a thorough analysis of the learned representations, identifying potential problems with it, and proposing possible solutions.

We have made our code publicly available[1], and we highly encourage the reader to refer to it for viewing the results.

1 Introduction

Deep Reinforcement Learning (RL) tasks have traditionally struggled in environments where the rewards are particularly sparse. Typically, an agent receives a reward only at the very end of the task when it successfully completes it. Thus, it is extremely difficult for it to ascertain at what point it went wrong, especially since the number of possible trajectories grows exponentially with the number of steps it takes. One possible solution to this problem is collect a large number of human demonstrations and imitate them through a machine; needless to say, this is not feasible in cases where interactions with the environment are expensive, such as robots learning to pour water, self-driving vehicular applications, etc., where one might lose many such machines before the model learns anything meaningful.

In this paper, we take the approach proposed in [1], where we perform a *self-supervised* learning task before the main RL task in order to guide the agent better and speed up the training. Specifically, given demonstration videos of an expert solving a series of 2-dimensional mazes (see Figure 2), our problem is focused on training an agent to achieve the same using reinforcement learning. To alleviate the problem of sparse rewards, we first solve a self-supervised computer vision classification problem,

[1] https://github.com/ranamihir/visual_embeddings_for_rl

Figure 1: Dog Learns to Open Door by Watching Human

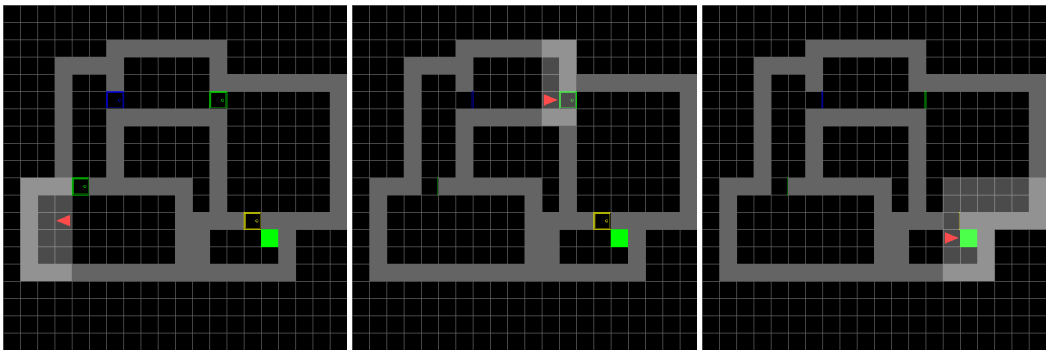


Figure 2: Different Agent Positions in 16×16 Maze

where the goal is to classify the temporal misalignment between pairs of frames that are sampled from this video. While solving this problem, we train an *encoder*, or an embedding network (see Figure 5), which learns continuous abstract representations of these frames in the form of embeddings. Finally, we make use of these embeddings to place checkpoints for the agent along the path of the expert trajectory, and give the agent rewards when it is close enough to these checkpoints, thereby incentivizing it, or guiding it, to clone the expert's behavior and making the rewards denser. In this manner, we train the agent using reinforcement learning via *Imitation Learning* (IL). As we demonstrate the effectiveness of this method, we also identify potential problems with the original paper, and propose possible solutions.

The intuition behind such a methodology is simple – living beings are able to perform complex tasks with ease just by watching, or *imitating*, other beings perform them. For instance, a dog can learn to open the door to a room by repeatedly watching humans do the same (see Figure 1); us humans learn to cook, use chopsticks, play sports, etc. to a great extent by watching others do it. In a similar way, *imitation learning* is a class of learning algorithms that aims to teach machines to perform complex tasks by watching expert demonstrators do them.

Background: Reinforcement Learning

Reinforcement Learning is traditionally seen as a finite-state Markov Decision Process (MDP), in which an *agent*, the learner, interacts with an *environment* (everything outside the agent, typically not in its control). Given its current situation, or state, s_t, the agent takes a specific action a_t, after which it receives some scalar reward signal r_t, and is transitioned into a new state s_{t+1}. To select, or *decide*, the next action a_t, it follows some *policy*, $\pi(a_t|s_t)$, which is the agent's behavior, and is conditioned only on the current state (following the 1$^{\text{st}}$ order *Markov* assumption). This can hence

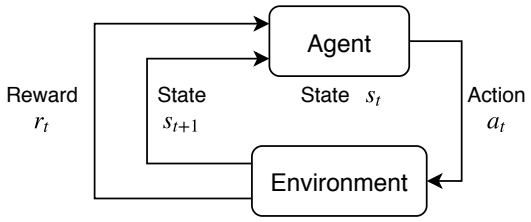


Figure 3: Reinforcement Learning as an MDP [2]

be seen as an MDP characterized by a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P})$, where $\mathcal{S}$ is the state space (the space of all possible states an agent can be in), $\mathcal{A}$ is the action space (the space of all possible actions the agent can take), $\mathcal{R}(s_t, a_t)$ is the reward function that maps any given state s_t to an action a_t, and $\mathbb{P}(s_{t+1} \mid s_t, a_t)$ is the transition probability that the environment will assume state s_{t+1} if the agent takes action a_t when in state s_t. The agent is put in the next state, s_{t+1}, according to the environment dynamics, or *model*, that is typically unknown, and is either estimated or learnt – as in the case of *model-based* RL algorithms, or is not learnt – as in the *model-free* RL case. In some environments, exact rewards can be directly obtained from it; otherwise, they must be defined explicitly as per the task that is to be achieved. This process continues until the agent reaches a terminal state, after which the reward is a constant at zero, and then it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the future discounted, accumulated reward with the discount factor $\gamma \in (0, 1]$, whose expectation the agent aims to maximize at each time step t. This entire process is depicted in Figure 3.

Background: Self-Supervised Learning

In many cases, it is extremely expensive, or sometimes not possible, to manually label the data. For example, if we want to make a robot learn how to pour water in a glass of water based on the camera images it takes as input using supervised learning, it is not feasible for humans to label each and every aspect of the images through the trajectory of the robot, especially considering the massive size of datasets used and required in practice today. A cheap, potentially effective, and important part of the solution, is *self-supervised learning*. In this setting, we perform an initial task prior to the primary problem, which is defined and constructed by us, and requires no additional data other than that required to solve the primary problem, in order to learn meaningful representations of the environment, so that they may be used as input in the second stage, i.e., the reinforcement learning task, to improve the training of the robot. This initial learning problem is *supervised*, i.e., we construct the problem in such a way that we already have, or can extract, the true target values for it from the existing data.

2 Related Work

Imitation learning methods have been widely used in robotics applications to make a robot learn to perform a particular task by watching human expert demonstrators [3] [4] [5]. Broadly speaking, imitation learning can be divided into two major fields: (1) Behavioral Cloning (BC), and (2) Inverse Reinforcement Learning (IRL). BC is the class of algorithms which considers this problem in a supervised setting, i.e., when we have state-action pairs (s_t, a_t), and learn to predict the action directly from the state [6] [7]. However, this method is not robust, and completely breaks when the slightest amount of stochasticity is introduced in the environment. If the agent deviates from the expert trajectory, it is not able to recover. The second class of algorithms, IRL, aims at learning a reward function that is used to optimize the agent's policy with reinforcement learning [8] from expert demonstrations. On a high level, our work is closely related to IRL. However, a key distinction is that our setting is more complex, since we do not feed expert action and reward sequences as input during training.

A related approach is the single-view Time Contrastive Network [9], which also makes use of a self-supervised task and does not require externally labelled data. The approach in this paper differs from this work by using temporal difference classification instead of triplet-based ranking, which removes the need to tune many sensitive hyperparameters.

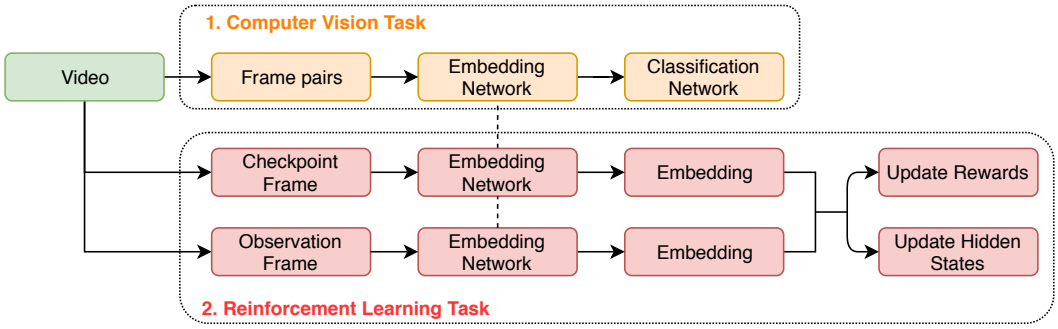


Figure 4: Overall Methodology

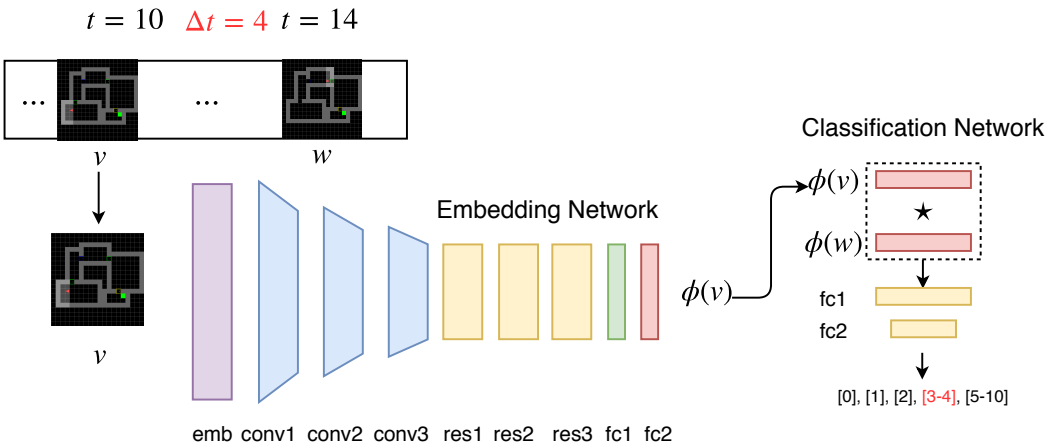


Figure 5: Embedding and Classification Networks.
Architecture from [1] with added initial Embedding (*emb*) layer

Finally, the paper which originally proposed a similar approach [1] combines multiple modalities (vision and sound) and works on Atari's MONTEZUMA'S REVENGE [10]. In this paper, we work on a different environment; importantly, our environment is irreversible, i.e., not all actions of an agent can be undone. Specifically, if our agent crosses a particular maze without passing through a particular checkpoint – which is possible since our environment is grid-based, and there are multiple optimal paths – it cannot go back to a previous maze to collect that checkpoint. As a result, the originally method proposed in [1] does not directly work in many cases, and we later provide a solution to this problem.

3 Methodology

We now concretely define the methodology used to solve the problem. For the purposes of our study, we consider a 2-dimensional grid-based maze environment built on top of the OpenAI Gym [11], which is shown in Figure 2. In this environment, the agent receives a reward only upon successful completion of the end goal, i.e., solving the maze by reaching the green cell. This is a hard exploration environment with extremely sparse rewards, since there are multiple levels of difficulty – (1) the number of trajectories increases exponentially with the steps – and becomes increasingly more complex as the grid size increases, (2) there are additional obstacles placed in the agent's way that it has to interact with, such as opening a door, etc., (3) the structure of and the number of rooms and doors in each maze is variable, and (4) being a grid-based structure, there are multiple optimal paths based on the Manhattan distance, and the agent must be aware of and robust to such paths.

To tackle this, we use the approach proposed in [1]. A high level overview of this methodology is shown in Figure 4. Specifically, we perform an initial auxiliary task before the main RL one, as follows:

1. **Self-Supervised Computer Vision Task**: The goal here is to classify the temporal misalignment between a pair of frames sampled from a video. We sample many pairs of frames from the given expert demonstration videos, and train a model to classify at what time-(frame-) difference the two frames are. The architecture used here is shown in Figure 5. We first pass each frame through an encoder, or an embedding network, which generates an embedding for each of them. These two are then passed to a classification network, which does an elementwise multiplication of the two embeddings, and passes them through a 2-layered multi-layered perceptron (MLP) which outputs probabilities for each of the temporal buckets.

   We define the data generation process in detail in section 4.1. To get the optimal path for the expert demonstrations, we implement bread-first search (BFS). The true temporal difference between a pair of frames, then, is just the difference in the indices of the stored maze video.

   For the classification task, we use temporal differences of upto 10, with 5 buckets defined as [[0], [1], [2], [3-4], [5-10]].

   The rationale behind this task is to learn meaningful embedding representations for each frame entirely from a small number of unlabelled expert demonstrations; we later show that these indeed capture the physical dynamics of the environment well, both, extrinsically through an independent analysis of the embedding space, and intrinsically, by showing that these embeddings help in improving the reinforcement learning task.

2. **Imitation Learning Task**: Use these representations to improve the training of the agent in the reinforcement learning task by making the rewards denser. To achieve this, while training the agent policy, we place checkpoints at fixed time intervals along the path of the expert trajectory. Then, at every time step, we compute the euclidean distance between the embedding of the current observation and the embedding of the next checkpoint. If the distance is below a certain pre-specified threshold (a hyperparameter), we give a reward of 1 to the agent, and increment the checkpoint index to the next one. We also set the current hidden state of the agent to be the next checkpoint (embedding).

   In this way, the agent is guided to (roughly) follow the trajectory of the expert. Importantly, the rewards in this case are much more dense – we, by construct, decide at what intervals the checkpoints (which we got entirely from a self-supervised task) must be placed along the way of the agent. If the interval is too short, the path will be extremely restrictive, and the agent will not be robust, since it will be forced to follow the *exact* path of the expert. If it is too long, the rewards will not be dense enough to effectively guide the agent.

Similar to the embedding network, to consider as a baseline, we also train a Variational Auto-Encoder (VAE) [12] tasked to reconstruct each frame in the video, and learn embeddings through this network. These embeddings are then used for placing checkpoints in the RL task as before.

An important **assumption** we make for this method to work is that the expert trajectory is *optimal*; hence, demonstrations encode distance between frames, i.e., closer frames have lesser distance in some abstract space (which we extract via the embeddings).

4 Experimental Setup

4.1 Data

The dataset is constructed using [13] (built on top of the OpenAI Gym [11]) and comprises unlabelled demonstration videos of an expert navigating through (i) 1000, and (ii) 10000 mazes of grid sizes 8×8 and 16×16 each, with obstacles such as closed doors of different colors, as shown in Figure 2.

Using this environment, the expert demonstration videos are constructed by implementing the bread-first search (BFS) algorithm between the agent's starting position and the target, i.e., the green cell. A subtle yet important point to keep in mind is that actions such as changing the orientation of the agent, opening or closing doors, etc., are additional steps in this search, and add extra temporal distance between any given pair of frames, which our model must be able to capture. Also, the observation and action spaces are discrete; the agent can only move from once cell to the next, not anywhere in between, and can similarly only take actions of UP, DOWN, LEFT, RIGHT, OPEN DOOR, and CLOSE DOOR – it cannot move diagonally, for instance.

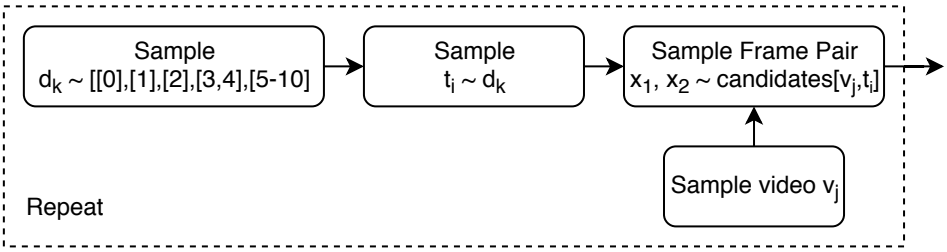


Figure 6: Data Generation Process
d_k = time bucket, t_i = time difference

The average sequence length of any given 8×8 maze is nearly 9 frames, while that of the 16×16 maze is nearly 21 frames; which means although we flatten the entire sequence into one video, the total video length is relative small.

It should be noted that this environment is *fully-observable*, i.e., the entire environment is visible to the agent at any given point in time; specifically, it can see where the target is.

We construct the data for the self-supervised task from this video (which, in our case, we consider to be one long sequence) of the agent solving mazes. This is the reason we do not require a big dataset – we keep sampling pairs of frames from the same video without requiring *any* additional external data.

The data for the self-supervised classification task, as mentioned before, is in the form of pairs of frames sampled from the original video. Specifically, the following procedure is repeated till 500,000 (train), 100,000 (validation), and 100,000 (test) frame pairs are obtained, as also shown in Figure 6:

1. Randomly sample a time difference bucket d_k from the list of buckets mentioned in section 1
2. Randomly sample a time difference t_i from bucket d_k
3. If multiple demonstration videos are available, randomly sample a video v_j
4. Out of pairs of frames in video v_j that are at a temporal difference of t_i, randomly sample a pair
5. Repeat steps 1-4 till required number of pairs are obtained

Note that, we do not use the raw pixels as input for either task. Instead, we use the observation space, which is a 3-dimensional matrix of spatial dimensions the same as that of the maze, but each channel representing a cleaner form of input. The first channel represents the objects in each cell, e.g., a label of 10 to denote the agent, 1 to denote empty cells (that the agent can move to), 2 for walls, etc. The second channel conveys the orientation information of the agent: 1 for up, 2 for right, 3 for down, and 4 for left. The last channel represents the interaction between various objects in the environment, such as information about whether a door is locked or not, etc.

4.2 Setup

As described in section 4.1, the dataset for the classification task is divided in the ratio 5(train) : 1 (validation) : 1 (test), with 500,000 pairs of frames sampled from a single demonstration video of an expert agent navigating through a variety of mazes for the training set.

To ensure robustness and add stochasticity, we make the doors of different colors. For the mazes of grid size 8×8, any given maze has either 1 or 2 rooms, with a door in between that must be opened first to pass through (and hence adds extra distance in the temporal space, which is not captured by BFS). For the mazes of size 16×16, any maze has between 3 and 6 rooms (inclusive); again, all adjacent rooms are connected by a door.

The embedding network is as shown in Figure 5. We implement a network similar to that described in [1], with 3 sets of convolutional – batch norm – ReLU layers, the last two of which use either spatial max-pooling and strided convolutions (we found both to perform similar) for downsampling. These are followed by 3 residual layers.

We make one important addition – since our observation space is discrete (as described in section 4.1), it is essential that we convert it into one that's continuous. To achieve this, we add an initial embedding layer for each channel in the input observation that projects each *cell* to an 8-dimensional continuous space. For the following (first) convolutional layer, we stack these 3 projections into 24 channels. The final embedding output size is 256.

Two such embeddings (for each frame) are fed to the classification network, which does an element-wise multiplication of the two and feeds the output to what is essentially a 2-layered multi-layered perceptron, all linear layers of which have a size of 256, producing the probabilities for each class (time difference bucket).

For the VAE, the architecture is almost exactly as that described for the embedding network (with the added embedding layer in the beginning), except we found an embedding size of 32 to work best in terms of reconstruction losses.

The networks are trained with a batch size of 128 split across 2 NVIDIA TITAN X (Pascal) GPUs, for 15 epochs, with early stopping based on the validation set accuracy with a patience of 10 epochs. We use the Adam [14] optimizer with a learning rate of 1e-4.

The two networks (embedding and classification) are trained jointly using a standard multi-class cross-entropy loss defined as follows:

$$\mathcal{L}(y, v, w) = -\sum_{j=1}^{K} y_j \log(\hat{y}_j) \tag{1}$$

where y is the true target class (time bucket), K is the number of classes (buckets), i.e., 5 in our case, and $\hat{y}_j$ is the probability of class j predicted by the model, defined as:

$$\hat{y}_j = \text{model}(\phi(v), \phi(w)) \tag{2}$$

where v and w are any two frames from the video.

For the reinforcement/imitation learning task, we use a standard Proximal Policy Optimization (PPO) [15] algorithm for training the agent. We also use Generalized Advantage Estimation (GAE) [16], a learning rate of 2.5e-4, gradient clipping at 0.1, and linear decay of the learning rate. To get stable results, we parallelize training across 16 processes. The training is run for 6,000,000 steps.

To update the rewards, the following protocol is used, based on the Euclidean distance between the embeddings of the agent (a) and the checkpoint (c):

$$r(a, c) = \begin{cases} 1, & \text{if } \|\phi(a) - \phi(c)\|_2 < \tau \\ 0, & \text{otherwise} \end{cases}$$

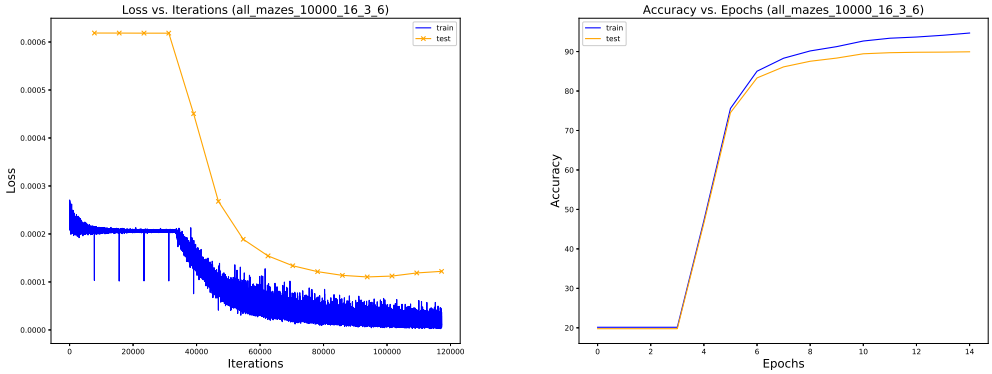
where τ is a distance threshold (hyperparameter); in our experiments, we found τ =1e-6 to work best.

Another important contribution we make to the original paper is to set time limits for reaching checkpoints. The original paper requires reversible environments, i.e., any action of an agent can be undone; in our context, that means the agent must be able to go back to any previous maze to collect a checkpoint reward even if it crosses that maze. However, to make the environment more difficult, we do not allow for that. Instead, we set a time limit of 100 steps. If the agent does not reach the next checkpoint within 100 steps of the previous one, we reset the environment to avoid wasting the whole episode. In our experiments, we found that without this fix, the agent skips many checkpoints and goes on to the next maze, and since it's not reversible, the system breaks and the training fails.

5 Results and Discussion

5.1 Classification Results

In Figure 7, we plot the training and validation set curves for loss and accuracy for the classification task on the 16×16 maze. An interesting observation we make is that the training is almost stagnant at 20% for the first 3 epochs, and then it boosts up suddenly in the following epochs, before saturating off at around 96%. This is likely because there is a lot of variation in our environment, and since we



(a) Loss vs. Iterations

(b) Accuracy vs. Epochs

Figure 7: Loss and Accuracy Curves for 16×16 Maze

Grid Size	Mazes	Accuracy
8×8	1000	84%
8×8	10000	88%
16×16	1000	90%
16×16	10000	96%

Table 1: Classification Test Accuracy for Different Configurations

flatten all maze demonstration videos as one long sequence, it is initially difficult for the model to ascertain the temporal difference in frames that are across two different mazes.

We now show time-misalignment classification accuracy results in Table 1. As can be seen, the accuracy scores obtained are extremely high. Notice that as the grid size increases, so does the accuracy – this is possibly because we treat all mazes as one long sequence (or video). Since time differences within a maze are more pronounced than that across different mazes, bigger grid sizes (with longer sequence lengths for each maze) are easier for the network to distinguish between.

Next, we analyze the embedding latent space obtained from the embedding network in Table 2.

1. We sample triplets of frames from the same maze and ascertain if the triangle inequality holds. The values shown in the table (k) are computed by taking the average across 100,000 such triplet samples for each maze, and 200 such mazes.

   As seen, the triangle inequality holds in almost all cases for both grid sizes, indicating the embedding space might have a linear relationship with the true time distance metric.

2. In the last two columns, we inspect the Pearson correlation coefficients between the embedding space distribution and the true (time) distance distribution. To achieve this, we sample 100,000 pairs of frames, and extract the true time difference between each pair, which is equivalent to the difference in indices of the two frames. For the second distribution to compute the correlation against, we take the following steps:

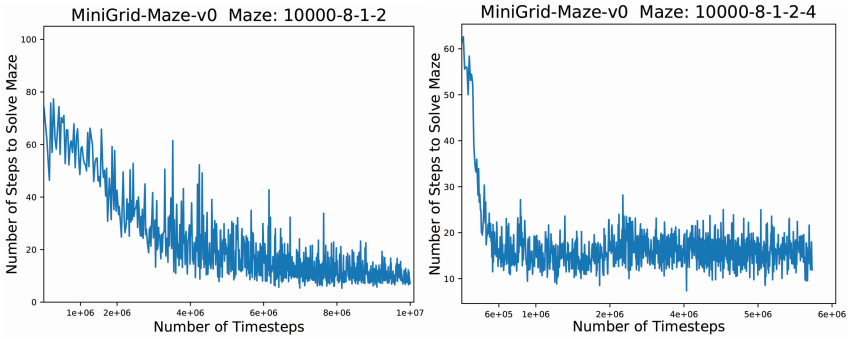
   (a) In the first case, for the 100,000 pairs of frames, we get the embeddings for each of them and compute the cosine distance, followed by the correlation coefficient $\rho_{emb,bfs}$ for it with the true temporal distance.

   (b) In the second case, we repeat the same procedure, but instead of the embeddings, we take as input the raw observations (which are fed to the embedding network), and obtain the correlation coefficient $\rho_{raw,bfs}$ for it with the temporal distance.

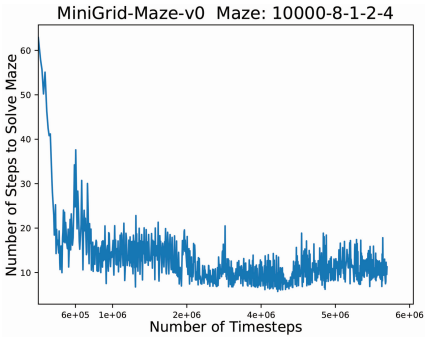
   The resulting correlations are shown in Table 2 for each grid size. As can be seen, the values are substantially higher in the embedding case, which further the indication that the embedding space might be linear (since correlation is a linear operator).

Grid Size	k	$\rho_{\text{emb,bfs}}$	$\rho_{\text{raw,bfs}}$
8×8	1.0	0.739	0.314
16×16	0.99	0.856	0.640

Table 2:
k: Fraction of Triplets with Valid Triangle Inequality
$\rho_{emb,bfs}$: Pearson Correlation of (BFS, Embeddings+Cosine)
$\rho_{raw,bfs}$: Pearson Correlation of (BFS, Raw Obs+Cosine)



(a) Pure RL (b) IL+VAE



(c) Our model

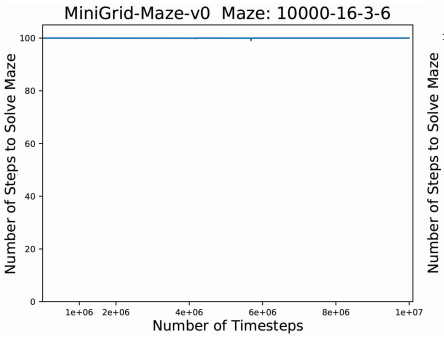
Figure 8: Avg. Steps to Solve 8×8 Maze vs. Timesteps (Checkpoints every 4 frames)

One interesting observation to note is that the correlation coefficient for the raw observations case is higher in the bigger grid size – this is expected, since more of the environment is the *same* (static); however, our embedding space is still able to achieve an extremely high correlation of 0.856 in this case.

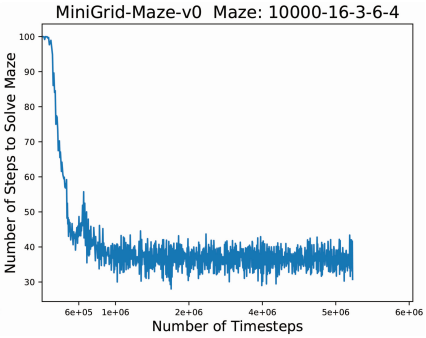
The implications of these results are quite essential to the reinforcement learning task: The embedding network is able to explain the time misalignment between a pair of frames by warping the raw observation space into a space that is (possibly) linear with the true spatio-temporal distance between the agent's position in the two frames; in other words, it is able to extract meaningful representations from these observations, and, in turn, understand the physical dynamics of the system. This is of paramount importance – with this prior knowledge about environment dynamics, our agent is well-equipped to tackle the RL task; unlike the case in which we train the policy from scratch, wherein it has to learn everything while learning the policy.

5.2 Imitation Results

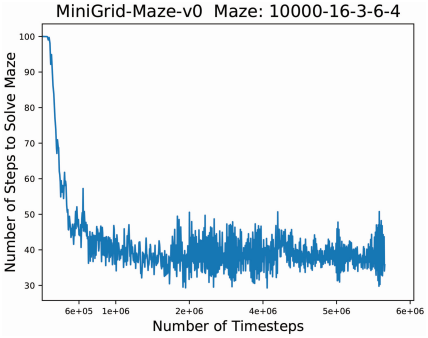
In this section, we show the results for the imitation learning task, where the agent is trained to solve the mazes described in section 4.1.



(a) Pure RL

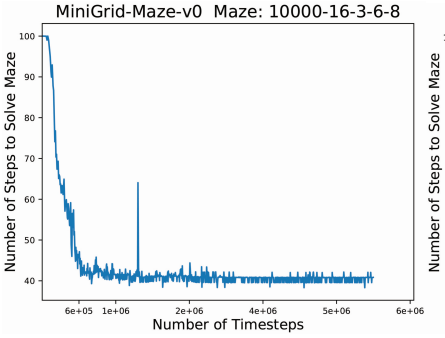


(b) IL+VAE

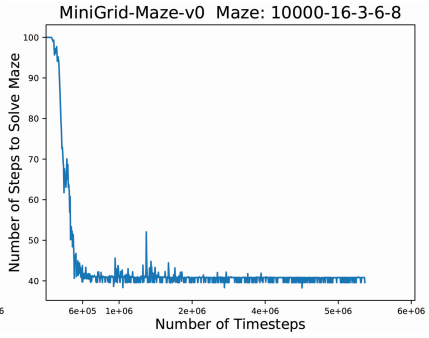


(c) Our model

Figure 9: Avg. Steps to Solve 16×16 Maze vs. Timesteps (Checkpoints every 4 frames)

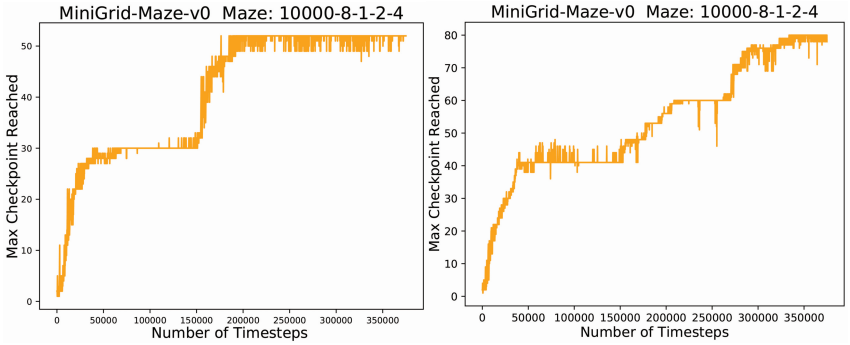


(a) VAE

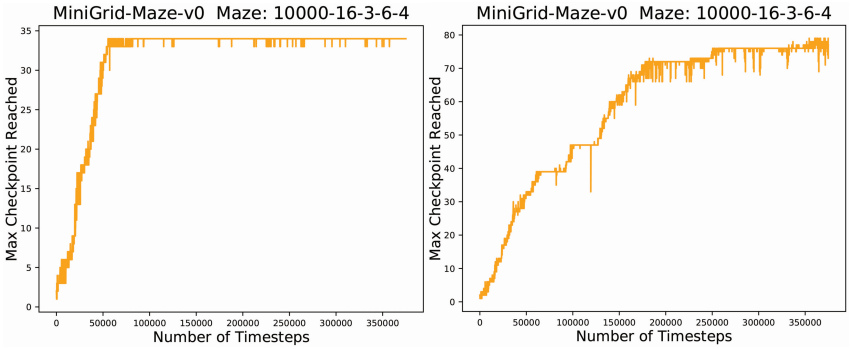


(b) Our model

Figure 10: Avg. Steps to Solve 16×16 Maze vs. Timesteps (Checkpoints every 8 frames)



(a) IL+VAE (8×8) (b) Our model (8×8)



(c) IL+VAE (16×16) (d) Our model (16×16)

Figure 11: Max. Checkpoint Crossed vs. Timesteps in Different Mazes (Checkpoints every 4 frames)

1. **Average steps taken to solve 8×8 maze with checkpoints every 4 frames**

First, we plot the average number of steps taken (across different processes) vs. the number of timesteps in Figure 8. In this case, the policy trained using pure RL (with sparse rewards) is effective, i.e., the agent is able to solve a given maze in nearly 15 steps on average.

However, the VAE and our model are able to solve them in less than 10 steps – which is essentially the optimal number of steps in the 8×8 maze.

Among these two, the latter converges faster, and also solves the mazes in fewer number of steps, although the difference is subtle.

2. **Average steps taken to solve 16×16 maze with checkpoints every 4 frames**

Next, we plot the same graphs for the 16×16 maze in Figure 9. In this case, the difference in the 3 training paradigms is more pronounced. Pure RL fails to solve any maze. As mentioned in section 4.2, we reset the environment after 100 steps if the agent fails to reach any checkpoint, and the average steps in this case taken never goes below 100. Comparing the VAE with our model, we see that both are able to solve the mazes in 30 steps on average.

3. **Average steps taken to solve 16×16 maze with checkpoints every 8 frames**

In Figure 10, we show the same results, but provide sparser rewards, i.e., with checkpoints every 8 frames instead of 4. In this case, we see that the agent is still able to solve the mazes in both cases (VAE and our model); however in 40 steps on average, instead of 30. It can also be seen that our model has slightly more stable and better convergence, although the difference is not very noticeable.

From this discussion, it seems that both VAE and our model outperform the pure RL training case, and that the former two are performing very similarly. However, we show in the next section that this does not tell the full story; that our model is indeed better than the VAE.

4. **Maximum checkpoint crossed with checkpoints every 4 frames**

In Figure 11, we show the maximum checkpoint (taken over the different parallel processes) that the agent crosses for both grid sizes with checkpoints every 4 frames, for both the VAE and for our model.

It should be noted that this is the *true* goal that we ultimately care about on this task – how many mazes the agent is able to solve.

For the 8×8 maze, the agent trained with VAE embeddings reaches at most the 50^{th} checkpoint on average, while the one with our model crosses the 80^{th} checkpoint.

In the 16×16 maze, our model's performance is not deteriorated; the agent still reaches the 80^{th} checkpoint, while that of the VAE drops down to the 35^{th} checkpoint.

Clearly, this a strong result: our learned embeddings capture the environment dynamics well enough for varying grid sizes and produce robust results, while the VAE, although performing better than training in the pure RL paradigm, does not generalize to environments where the number of possible trajectories increases beyond a point.

5. **Evaluation of Learned Policy**

As a final (and arguably the most important) step of evaluation of our methodology, we run the trained policy and record upto what checkpoint (maze) the agent reaches. As observed during the training, it reaches the 80^{th} checkpoint; a video demonstration of this agent can be found in our repository[2].

6 Conclusions and Our Contributions

We showed that pure reinforcement learning techniques fail under very sparse reward environments, wherein the number of possible trajectories (exponentially) increases beyond a point. To alleviate this issue, we constructed a self-supervised learning task to predict the temporal misalignment between a given pair of frames in a video, and generated embeddings using the encoder. These embeddings were then used to initialize the recurrent states of the agent, and for making the rewards denser by placing checkpoints along the path of the expert trajectory. Akin to this, we also performed a similar process with a variational autoencoder tasked to reconstruct any given input frame of a video, and used that to similarly extract the embeddings.

To independently analyze the quality of these embeddings and demonstrate that the embedding space indeed extracts an understanding of the physical dynamics of the environment, we performed an analysis of the embedding space by measuring its correlation with the true temporal difference, and compared that against the correlation with the raw state observations, which we show is weaker compared to the former case. Additionally, the embedding space also displays signs of linearity with the true distance metric when evaluated in terms of the fraction of triplets for which the triangle inequality holds true.

In the second stage of this study, we trained our RL policy using PPO for maze grids of sizes 8×8 and 16×16, and aided the training with embeddings learnt from the VAE and from our embedding network. We showed pure RL training is inferior to the VAE and our model, and in extreme cases, also fails completely. Our model performs the best, both in terms of the average number of steps taken to solve the maze, and the maximum checkpoint it reaches, or equivalently, the maximum number of mazes it solves.

We explained that reversible environments are necessary for the original method proposed in [1], otherwise some checkpoints may be skipped and the system breaks. To overcome this, we showed that setting time limits for reaching checkpoints is effective, as it avoids wasting the whole episode.

[1] also requires setting the first checkpoint as the initial state, and same training and test environments to allow for checkpoints during testing, since if we have different initial states it might

[2] https://github.com/ranamihir/visual_embeddings_for_rl

not be possible for the agent to even reach the first expert checkpoint.

Although we could not concretely test the solution due to time constraints, we propose the following solution: Assuming there is more than one expert demonstration for each initial state, or at least reasonably close enough to each one, set the closest starting state from other similar demonstrations to the current one, thereby allowing for different training and test environments.

7 Future Work

Possible extensions to work include, but are not limited to:

- Addressing *domain gap*: Make environments more complex, such as by starting agents from different initial positions, introducing more artifacts and factors of variation in the environment, e.g. with obstacles in the environment (keys to locked doors, etc.), different colored doors/walls, etc.
- Extending this framework to real-world settings, such as a robot learning to pour water, etc.
- Extending to partially instead of fully observable environments, where the agent can only see parts of the maze which are in its field of view

8 Acknowledgments

We are thankful to Dr. Rob Fergus (Associate Professor of Computer Science at Courant Institute, NYU and Director, Facebook AI Research, New York) and Ilya Kostrikov (PhD candidate, Computer Science at Courant Institute, NYU) for their guidance, valuable suggestions, and constructive feedback throughout this study. We are also grateful to Ilya for: (1) helping us with the code for the RL task, specifically, for setting up the environment and making it compatible with his prior work[3,4], and (2) generously sharing his GPU compute resources with us for making this work possible.

References

[1] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems 31*, pages 2935–2945. 2018.

[2] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[3] J. A. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation (ICRA2002)*, Washinton, May 11-15 2002, 2002. clmc.

[4] Nathan D. Ratliff, James A. Bagnell, and Siddhartha S. Srinivasa. Imitation learning for locomotion and manipulation. In *2007 7th IEEE-RAS International Conference on Humanoid Robots, November 29th - December 1st, Pittsburgh, PA, USA*, pages 392–397, 2007.

[5] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *Int. J. Rob. Res.*, 32(3):263–279, March 2013.

[6] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[7] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:97, 1991.

[3] Adapted code available here: https://github.com/ranamihir/visual_embeddings_for_rl

[4] Original code available here: https://github.com/ikostrikov/pytorch-a2c-ppo-acktr

[8] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[9] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-contrastive networks: Self-supervised learning from video. *Proceedings of International Conference in Robotics and Automation (ICRA)*.

[10] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 4148–4152. AAAI Press, 2015.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[13] Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[17] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems 30*, pages 4967–4976. 2017.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[19] Yurii Nesterov. *Introductory Lectures on Convex Optimization*. Springer US, 2004.

[20] Yuxi Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.