Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2022)

World Models with an Entity-Based Representation

Nazanin Yousefzadeh Khameneh and Matthew Guzdial

Department of Computing Science, Alberta Machine Intelligence Institute (Amii)
University of Alberta
Edmonton, Alberta, Canada
{nyousefz, guzdial}@ualberta.ca

Abstract

Reinforcement learning (RL) is a powerful way to solve sequential decision-making tasks. However training an RL agent in a complex environment requires a large amount of interactions, which is non-ideal when acting in an environment is costly or dangerous. One alternative is to learn an approximation of the real environment, referred to as a world model. This simulator can be used to train an agent and transfer the learned policy to the real environment. Unfortunately, training world models have traditionally required a significant number of interactions in the real environment. This brings us to the same problem when it is costly or dangerous to act in the real environment. To address this problem, we present an entity-based representation and corresponding architecture, which allows for greater data efficiency in world model training. Our approach outperforms other world model baselines in an initial application to the game Pong.

Introduction

Reinforcement Learning (RL) is a framework for defining and computationally solving sequential decision-making problems. Deep RL (DRL) has arisen as a combination of Deep Learning (DL) and RL to tackle more complex domains. Game-playing agents are among one of the most popular successes of DRL (Mnih et al. 2013). However, these approaches tend to require a significant amount of training data, often in excess of millions of interactions with an environment. This can be non-ideal, such as when interacting in the environment is dangerous or costly (Kiran et al. 2021).

As opposed to interacting in the real environment directly, there are alternative approaches to learn a simulation of the real environment, sometimes called a world model (Ha and Schmidhuber 2018). The RL agent can then be trained in this simulated environment, rather than the real one. There have been a number of these methods in recent years, some with very impressive results (Wang, Kosson, and Mu 2017). However, they have all needed a large amount of data from the real environment (Lotter, Kreiman, and Cox 2016). This means that these models still require a large number of interactions in the—potentially dangerous and costly—environment. One possible reason for this is that these recent world model approaches have relied on pixel input, a high variance representation (Halevy, Norvig, and Pereira 2009). If we had a simpler representation, we could potentially learn a helpful model in fewer interactions.

In an attempt to find a method to train a simulated environment with less training data, we propose a novel entity-based representation for game environments. This entity-based representation reduces the complexity of input, but can still be automatically extracted from a pixel-based representation for simple environments. For this reason, we focus on the Atari domain and specifically the game Pong. While the game Pong is simple, it is well-understood and appears in prior world model-learning work (Wang, Kosson, and Mu 2017), making it appropriate for an initial exploration.

We employ our entity-based representation as the basis for an action-conditioned model that predicts reward and the next state in order to serve as a world model. We use this model to train an RL agent, and compare it to an agent trained in the real environment, an agent trained in an existing World Model baseline (Ha and Schmidhuber 2018), and several variants of our approach. We find that our approach leads to a more accurate and useful model with much less training data compared to prior work.

Related Work

Game Representations

Automated game playing has traditionally been an important domain for artificial intelligence (Silver et al. 2018). One of the fundamental topics in intelligent game playing is the problem of efficient game representation (Mandziuk 2010). The majority of DRL approaches take video game frames, represented as matrices of pixels, as input (Oh et al. 2015). A single level of a game may consist of tens of thousands of frames, and each individual frame contains a great deal of information, much of it useless. Other approaches rely on more condensed, human-authored representations (Thielscher 2017), however, these representations take significant authoring effort and cannot represent many complex environments. We attempt to automatically extract a condensed representation from game frames.

There are other approaches to automatically extract or learn a representation related to game frames. For example, (Osborn, Summerville, and Mateas 2017) directly access the visual data, gaining information about what is present in

a frame directly from the game. While this is theoretically possible for all games, it might require specialized knowledge of the hardware or software in question. As an alternative approach, other work has attempted to automatically learn a representation from game frames or images (Jadhav and Guzdial 2021; Smirnov et al. 2021). There is also work combining both of these prior approaches, accessing a game's memory and then learning a usable representation from it (Karth et al. 2021; Mawhorter et al. 2021). We view all of these approaches as complimentary to our method, and consider learning forward models based on these representations an open area of future work. However, given our results with learned representations discussed below, we believe that our approach may be more general due to its simplicitly (Khameneh and Guzdial 2020).

Forward Models and World Models

In this paper we focus on the problem of training RL agents with fewer interactions in the real or target environment. One way to offset this problem is to derive an accurate simulated environment or forward model (Kober, Bagnell, and Peters 2013), a concept that was originated by Schmidhuber (Schmidhuber and Huber 1991). The idea is to train an agent in the simulated environment and then transfer the learned policy to the real environment. However, acquiring a sufficiently accurate model is a challenging problem and in many cases requires a large amount of data.

(Oh et al. 2015) proposed a highly accurate video frame action-prediction model using a combined Recurrent and Convolutional Neural Network architecture. Their approach required 500K training frames for each game. (Leibfried, Kushman, and Hofmann 2016) extended the work of Oh et al. by including reward prediction, with similar training data requirements. (Wang, Kosson, and Mu 2017) proposed a convolutional feed-forward model for predicting future frames and rewards, trained on roughly 600K frames of the game Pong. This work relied on a trained DQN to play Pong, which was used to collect the training data for the forward model. However, relying on a trained DQN to collect the training data is not ideal, since this contradicts the purpose of learning a simulated environment to train the agent in, in the first place. (Kaiser et al. 2019) proposed a complete model-based deep RL algorithm based on video prediction models called Simulated Policy Learning (SimPLe), which relied on considerably less training data compared with previous approaches. While we worked on a similar but distinct problem, our dataset is nearly an order of magnitude smaller. Outside of games, there are more recent approaches to learning to simulate complex physics and particles (Sanchez-Gonzalez et al. 2020; Duarte and Vlimant 2022). These approaches have not been tested in games and typically require direct access to the simulator.

Ha & Schmidhuber (Ha and Schmidhuber 2018) presented a predictive model that can be trained quickly in an unsupervised manner to learn a compressed spatial and temporal representation of a game. They coined the term "world model" for a forward model represented in a deep neural network architecture that attempted to fully approximate the real environment. Their approach derived world models training data via 10K random rollouts in the real environment. While this is an improvement over requiring a trained DQN agent to collect the training data, it still represents a significant amount of interactions in the real environment. In addition, their world model did not include reward, making it impossible to train an agent solely in this simulated environment.

Entity-Based Representation

In this section, we introduce our entity-based representation. We describe our forward model based on this representation in the next section. We use Pong from the Atari Learning Environment (Bellemare et al. 2013) as an initial domain to test our approach, due to its simplicity and its history with forward model learning (Wang, Kosson, and Mu 2017). We draw on the same approach as (Sieusahai and Guzdial 2021), to identify individual entities. At a high level, we run a depth-first search over the pixels of a frame to find all pixels of the same colour. This allows us to identify individual entities in a frame.

Note that we assume that entities have a constant pixel color, and the entire entities are structured so that every pixel in the entity is adjacent to at least one other pixel. This assumption holds for the majority of ALE games. This assumption breaks when a single game entity has multiple colors, which is common in more complex Atari games. For example, in Montezuma's Revenge the player character makes use of five distinct colors. However, even in cases where it does not hold, due to the fact that the different "pieces" of a single entity have highly correlated behavior, this type of representation has been shown to still capture important semantics (Sieusahai and Guzdial 2021).

After identifying an entity as a collection of pixels we extract the following features based on each entity, and a nearest matching between entities across pairs of frames.

- *SizeX* and *SizeY* are the sizes of the entity in the x and y dimensions, based on the differences between the furthest pixels in each dimension.
- *PositionX* and *PositionY* are the positions of the entity in the x and y dimensions of a frame, identified as the value of the leftmost and uppermost pixels of each entity, respectively.
- *VelocityX* and *VelocityY* are the the velocities in the x and y dimensions, identified by the difference between an entity and it's matching entity in the next frame.

This means each entity is re-represented as a vector of shape (1x6): $<$*SizeX, SizeY, VelocityX, VelocityY, PositionX, PositionY*$>$. This ignores variation in entity shape, and assumes all entities can be represented as rectangles. Because our unit of measurement is pixels, all the values are integers at this stage. All of the features are always greater than zero except *VelocityX* and *VelocityY* which can be negative. For uniformity, we append vectors of zeroes to each frame, such that each frame in our data has the same number of vectors. This allows us to represent when an entity disappears in the prior frame or is about to appear in the next frame, like the ball appearing and disappearing in Pong.

Forward Model Learning

Forward models can simulate how a real environment changes in response to an action from an agent, and can therefore be used to train agents. Prior approaches relied on high dimensional pixel input observations to train a predictive model. We present a method that is able to learn the temporal and spatial dependencies of entities, using an entity-based representation. This allows our approach to learn to make predictions with considerably less training data. In this section, we describe our forward model approach based on our our entity-based representation.

Methodology

The goal of our model is to learn a function $f : s_{t-k:t}, a_t \rightarrow s_{t+1}, R_{t+1}$, where s_t, a_t, and r_t are the entity-based representation of the state, action and reward at timestep t, respectively. $s_{t-k:t}$ indicates states from time $t-k$ to time t. Entity extraction converts the pixel inputs to entity vectors, and our predictive model takes these vectors and a representation of the action as input to predict the next state and the associated reward. The predictive model is composed of an LSTM to extract temporal features from the input data. We chose an LSTM due to it's common inclusion in similar prior work and to simplify comparisons (Ha and Schmidhuber 2018). The model also has an action-conditioned layer, which is concatenated with the output of the LSTM. Finally, the concatenation of the LSTM output and action is fed into dense layers to predict both the next state and reward. The major difference from this approach to existing, prior work is in the use of our entity-based representation instead of pixel input (Oh et al. 2015).

Model Architecture

Our network takes a fixed history of prior states as an input. We choose four for the history state time horizon as prior work also used it for the Atari domain (Oh et al. 2015). The network architecture consists of three LSTM layers each with 1024 hidden units. The action is represented as a one-hot encoded vector, which is concatenated to the output of the third LSTM layer. As shown in Figure 1, after incorporating the action, the network is divided into two parts. In the first part, the output is a real-valued approximation of the vectorized representation of the entities. The second part employs a fully connected softmax layer, which predicts the reward. We split the reward into three categories (-1,0,1) treating it as a classification problem. We discuss this further below. We used dropout equal to 0.5, the Adam optimizer, a batch size of 64, and trained the model for 200 epochs.

Loss Function

The model is trained by minimizing the weighted sum of two loss functions, mean squared error (MSE), and cross-entropy.

$$L_{total} = \lambda L_{mse} + L_{cross-entropy} \quad (1)$$

where: $\lambda = 10$

We measure the MSE loss between the predicted state and the next ground truth state, where each state is in our entity-based representation. In addition, we employ the cross-entropy loss function (Simard et al. 2003), to measure the difference between the predicted and true reward values. We weight the MSE loss ($\lambda = 10$) to encourage the model to not focus on the "easier" classification task.

Reward Prediction

Around 95% of the rewards in a typical game of Pong have a 0 value. This makes predicting non-zero rewards difficult. In our final model, rewards are represented as one-hot encoded vectors of size three (Leibfried, Kushman, and Hofmann 2016). Even with this, there was a large imbalance in the distribution of reward classes. As a result, the variations of our approach discussed below struggled to predict non-zero rewards.

World Model

We present our world model in Figure 2. When an agent acts in our world model it takes as input the state and outputs an action a_t. We take the current state s_t, in our entity-based representation, and feed it and the action into our forward model. Our forward model predicts the next state s_{t+1} and the associated reward r. We then convert the predicted state s_{t+1} back to a pixel representation to feed into our agent and the loop continues. Figure 3 demonstrates example predicted frames using this approach in comparison to the true next frames. All frames are represented in black and white as our entity-based representation does not track pixel colors.

Evaluation

The focus of this paper is to achieve a world model that can be used to train an RL agent with less training data than prior work. The reason for this is to minimize the amount of interactions in the environment, under the assumption that the environment is dangerous and/or costly to act in for a randomly initialized agent. Thus, if we can train our agent entirely with our world model or employ our world model to pretrain our agent to avoid the worst behaviors in the real environment, then we have achieved this goal.

For comparison, we include four variations of our approach. The first variation is our proposed entity-based approach ("Entity") described above, while the remaining three allow us to interrogate certain assumptions in our approach. All three are based around employing a Variational Autoencoder (VAE) as the basis of a representation, which is closer to the learned latent representations in other work (Oh et al. 2015; Ha and Schmidhuber 2018). All of these approaches differ only in the state representation input to and output from the forward model.

- **Entity**: Our approach. In this version the input size is 1×54 where 54 is number_of_entities (9) $\times$ entity_dimension (6).
- **Entity Embedding**: For this variant we employ a VAE trained on individual entities, and use it's latent dimension as the input representation to our forward model. We

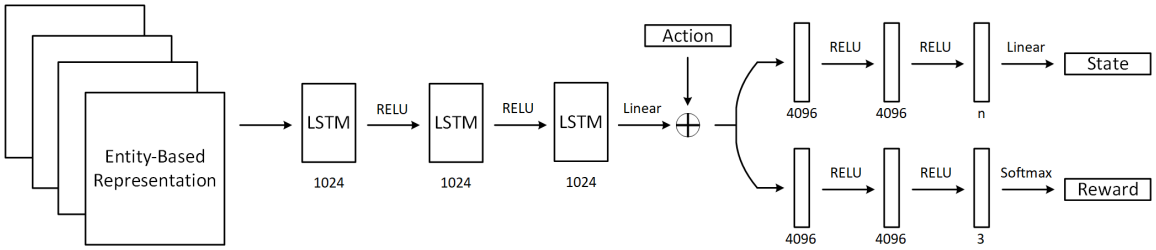

Figure 1: The complete architecture of our predictive model.

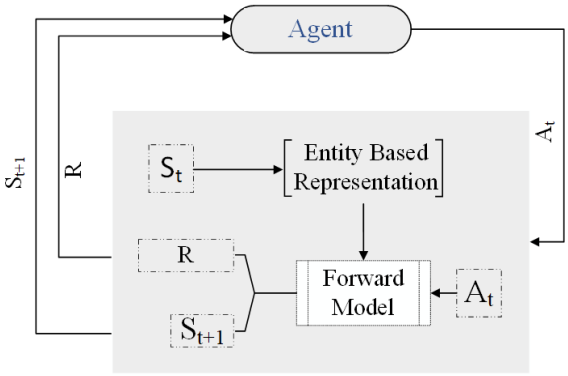

Figure 2: Visualization of our world model approach. An agent takes actions given the current state. The current state is transformed into our entity-based representation, and that and the action are passed to the forward model. The forward model then predicts the reward and the next state, and the cycle continues.

describe the VAE in further detail below. In this representation each input has a shape of 1×18 where 18 is number_of_entities (9) $\times$ latent_dimension_of_each_entity (2).

- **Output VAE**: We used the reconstructed output of the Entity VAE as the basis of another baseline. We included this as an in-between approach, potentially more general than the **Entity** representation, but with more detail than the **Entity Embedding** representation. In this approach each input has the same format as **Entity**.
- **Frame Embedding**: A variant embedding in which rather than individual entities we embed an entire frame's worth of entities. We employed this representation to explore if it is more helpful to represent all entities in a single embedding or model them individually. In this approach each frame has the shape of 1×12, where 12 is the latent_dimension_of_each_frame. We describe this model in more detail below.

We also employ the Ha & Schmidhuber world model work as a fourth baseline (Ha and Schmidhuber 2018), because it is the original world model paper and it had the smallest data requirement. We had to adapt this baseline to the Atari environment, thus we changed the input parameters to match Pong. Their model consists of a VAE to learn the compressed representation of the 2D image input, which inspired our embedding variant baselines. Rather than training a separate RL agent their model included a controller, responsible for choosing the action in order to maximize expected cumulative reward. We attempted to implement another forward model learning approach (Oh et al. 2015), however, their model had nearly 90 million parameters, and we found it to be untrainable with small datasets. We anticipate that the use of other, more complex forward model learning approaches would lead to similar results.

To evaluate our work we employ three experimental setups. To evaluate the quality of the predicted states, we directly compare the predicted and original frames. To evaluate the quality of the predicted reward function, we calculate the F1-score of the predicted rewards. Finally, to evaluate whether the forward models are helpful to an agent, we train a DQN agent in our world model.

Entity Embedding

For three of our baselines, we explore learning a latent representation based on our proposed entity-based approach. This allows us to verify whether our entity-based approach can be usefully compressed further. Employing a latent embedding in this way is the more typical solution to this sort of problem (Oh et al. 2015; Ha and Schmidhuber 2018). We train our VAEs on a dataset of entities extracted from our training dataset, which is described below. We explored a variety of VAE architectures and hyperparameters. Since accuracy of predicted entities using a VAE with low dimensional latent space is our main goal, we choose the model with the smallest possible latent space dimension with high reconstruction accuracy. Ultimately, we ended up with a model that offered high-quality reconstructions and several helpful characteristics, including the ability to differentiate between games while minimizing the difference between conceptually similar entities (Khameneh and Guzdial 2020). Our final architecture had only one fully connected hidden layer for the encoder and decoder, with a 2-dimensional embedding layer. The embedding layer used Sigmoid activation, with the two other layers using Relu. Further detail on this model can be found in (Khameneh and Guzdial 2020).

Frame Embedding

Our final variant baseline makes use of a VAE trained on all of the entities in frame instead of individual entities. In this model, each training instance is represented as a flattened vector of all of the entities in a frame. In Pong, we have

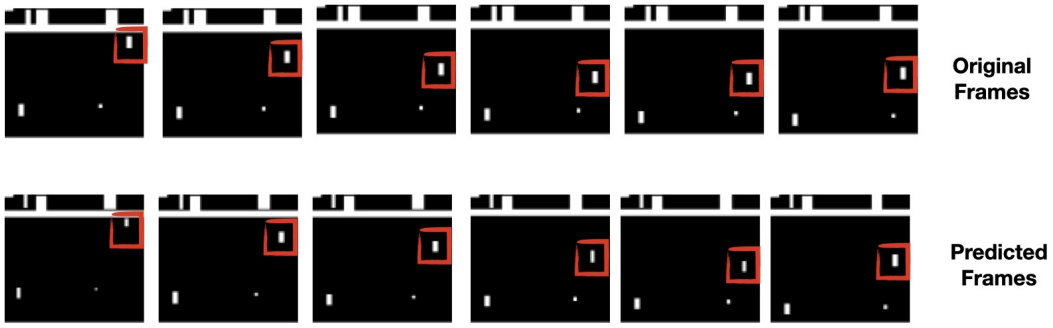

Figure 3: Consecutive predicted frames from our model. The right paddle is controlled by the agent in the original game.

at most 9 entities in each frame, and each entity has a shape 1×6, thus each input for this model has the shape 1×54. We employ a 12-dimensional latent space for this model, with all of the other layers and hyperparameters kept the same as the **Entity Embedding** VAE.

Dataset

For our dataset, we collect 15K frames in the Pong environment. We have an agent act randomly to explore the environment, and record the actions, states, and rewards. We use the first 14,000 frames for our training set, used to train our approach and the four baselines, and the last 1000 frames for our test set. This amount of training data is roughly an order of magnitude smaller than seen in prior work (Oh et al. 2015; Ha and Schmidhuber 2018).

Results

Predicted Frames

To evaluate the accuracy of the predicted frames, we use MSE and the Structural Similarity Index Measure (SSIM), which are two similarity metrics commonly used to compare images (Sara, Akter, and Uddin 2019). Given that our different baselines have different representations, we converted all of them to a grayscale frame for this analysis. In addition, since the goal of our model is to predict a frame in which each entity is in the right position with the right size, we also compare the predicted and ground truth for the entity representation of each frame. To this end, we calculate the MSE of each entity's position and size features between the predicted frames and the original frames. Lower values for MSE and higher values for SSIM are better. For this analysis, we feed the test data to each world model to predict the next frame given the prior frames.

As seen in Table 1, the **Entity** and **Frame Embedding** approaches outperform the others according to all metrics. We observed that, in both of these variants, our model is able to capture the movement of entities, particularly the player-controlled paddle. The models also predict the interactions between objects such as bouncing the ball off the paddles. The **Frame Embedding** approach performance makes sense, as it is closer to the more common wisdom of learning a single embedding for an entire state (frame) in forward model learning work. In comparison, the Ha & Schmidhuber baseline is unable to capture the important entities, such as the ball and paddles. One possible reason is that their model requires 10K rollouts while our training data was only the equivalent of 14 rollouts. The **Entity Embedding** approach does the worst of our variant approaches, even beaten out by **VAE Output**. This indicates that despite the model's high reconstruction accuracy, it lost important details in the latent space. Due to the importance of accuracy of the predicted states, we focus only on the **Entity**, **Frame Embedding**, and Ha & Schmidhuber models in our third experimental setup.

	SSIM ↑	MSE ↓	Entities ↓
Entity	0.9677	0.1644	10.1733
Entity Embed.	0.8442	0.4534	15.4121
VAE Output	0.9441	0.1997	11.1356
Frame Embed.	**0.9694**	**0.1587**	**8.0112**
Ha & Schmidhuber	-0.0951	2.4052	77.2876

Table 1: Results of our state comparison over our test data, comparing the output of the forward models to the original game states.

	Zero	Negative(-1)
Entity	**1**	**1**
Entity Embed.	0.99	0.76
VAE Output	0.99	0.79
Frame Embed.	0.99	0.77

Table 2: F1-score of the predicted test rewards

Predicted Rewards

To assess the validity of the reward prediction we calculate the F1_score, the harmonic mean of precision and recall. Since our training set is composed of 95% zero value rewards, it is important to investigate the validity of our model on non-zero rewards. In Table 2, we show the F1-score of our approach and the baselines over our test set for both 0 and -1 rewards. We did not include the +1 reward values in Table 2, as there were no examples of those values in our test set, as it is difficult for a random agent to score a point. Table 3 includes the F1-score of the predicted rewards over

	Zero	Negative(-1)	Positive(+1)
Entity	0.99	0.99	**1**
Entity Embed.	0.99	0.89	0
VAE Output	0.99	0.95	0
Frame Embed.	0.99	0.99	0

Table 3: F1-score of the predicted training rewards

	Real	Entity	Embed.	H & S	Rand.
Score	+1.89	-4.23	-6.15	-7.4	-10.4

Table 4: Average score of game playing agents over 100 game rounds in the real environment.

our training set. It shows that the **Entity** approach has an F1-score near or equal to 1 for all values, which shows the ability of the model to predict non-zero rewards. Our variant approaches could also predict both negative and zero rewards with a high F1-score. However, they could not predict positive rewards. This indicates that despite the **Frame Embedding** approach having the most accurate state predictions, it still may not represent the best world model. We did not include the Ha & Schmidhuber baseline in this section, as their approach does not predict rewards.

Training Agents

To evaluate the usefulness of the world models to our primary challenge we trained DQN agents (Mnih et al. 2013) using the **Entity** and **Frame Embedding** models. We employed the original DQN agent as Pong is a simple game and so we did not need a complex agent. We test the trained agents in the real environment to explore if they demonstrated positive transfer of the learned policy from the simulated environment to the real environment.

We use the agent from (Mnih et al. 2015) without alteration. In the original paper, the authors applied a simple frame-skipping technique from (Bellemare et al. 2013) to the frames. In frame-skipping the environment repeats the action for k frames and just returns the last frame to the agent instead of every frame. This technique allows the agent to play on k frames with much less computation. Since, we trained our forward models using the frame-skipping method they automatically predict the k^th frame. We choose $k = 4$ as the original paper suggests the same value for Pong. To train the agent, we use the decayed ϵ-greedy behaviour policy. ϵ decreases from 1 to 0.05 linearly over the first million frames. Episode lengths are 100 during training in all experiments. We choose 100 because our model can predict reasonable frames over around 100 time steps.

Figure 4, shows the average game score in the real environment per training episode, when training in both the real environment and simulated environments of the **Entity** and **Frame Embedding** approaches. We train three agents with different seeds in all experiments. This is a small number, but we found extremely consistent results in all cases, and so stopped after three. As is shown, the agent's score converges to -0.6 and -1 in the **Entity** and **Frame Embedding** simulated environments, respectively, while the agent's score converges to +0.5 in the real environment over the same period. The plot shows that agent learns by playing in the simulated environments, however with the same number of timesteps the agent learns more while playing in the real environment. This does not mean the simulated environments aren't useful, as they could potentially still help an agent learn to avoid catastrophic behavior in the real environment. This led us to try to pre-train an agent in the simulated environment, then transfer the learned policy to the real environment and continue to train the agent. Our goal was to see whether this would help us achieve a better score with fewer interactions in the real environment, while avoiding catastrophic behavior.

	Real	Pre-trained agent
Score	1.57	**3.65**

Table 5: Average game score for the agents over 100 game rounds in the real environment.

We choose the best performing **Entity** agent and transfer it to the real environment. We then trained the agent in the real environment with ϵ starting from 0.5 and decreasing to 0.05 in 500,000 steps. The agent trained for 1,200,000 steps or 12,000 episodes. We chose 0.5 as an initial value for ϵ to explore if we can train a DQN agent with less exploration that outperforms agents without pre-training. In Figure 5, the left plot shows the average score of this agent during training. As is shown the reward reaches +0.5 after around 5000 episodes. We also train the same agent in the real environment without pre-training it, with the pre-trained agent having a better jumpstart, and converging faster. Further, the pre-trained agent avoids the lowest average score achieved by the non-pre-trained agent. We take this as evidence of the utility of this approach, that it could be helpful to help avoid catastrophic behaviour from an agent in the real environment. This would be especially important in dangerous environments. We note that if we continue to train the two agents in the left side of Figure 5 they do eventually converge to the same policy.

The right plot of Figure 5, displays the cumulative reward of the Ha & Schmidhuber baseline during training. Since this approach does not predict the reward, the controller has access to the rewards from the real environment. As we mentioned the controller is a simple single layer linear model that maps the output of the VAE and LSTM directly to an action at each time step. To optimize the parameters of the controller, the authors employed Covariance Matrix Adaptation Evolution Strategy(CMA-ES) (Hansen 2006). We train the agent for 3000 generations. Since their predictive model is not able to predict appropriate future states, the controller converges to -1.5 after only a few generations.

We tested all the agents prior to the pre-training experiment in the real environment for 100 game rounds, each round with a length of 500 frames. As is shown in Table 4, the real environment (Real) agent outperforms all other agents. The **Entity** agent achieves an average score of -4.23, which is the second best agent. It beats out our **Frame Em-**

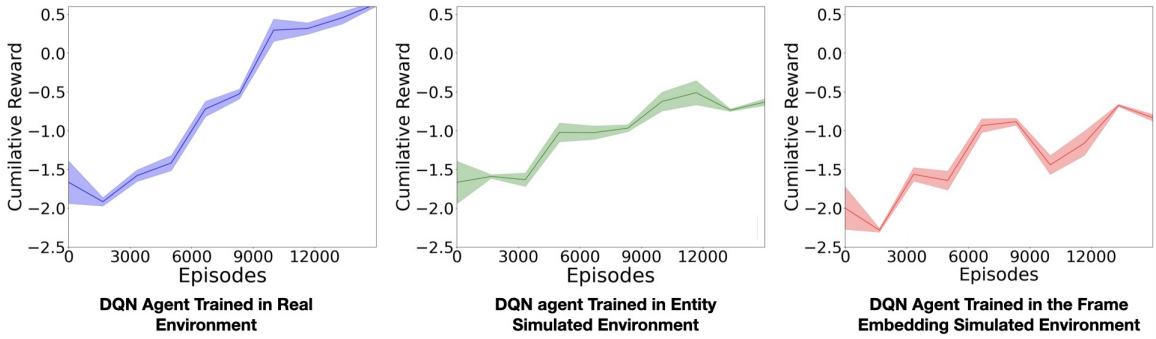

Figure 4: The average reward per episode on Pong during training in the real environment (left) and simulated environments (right).

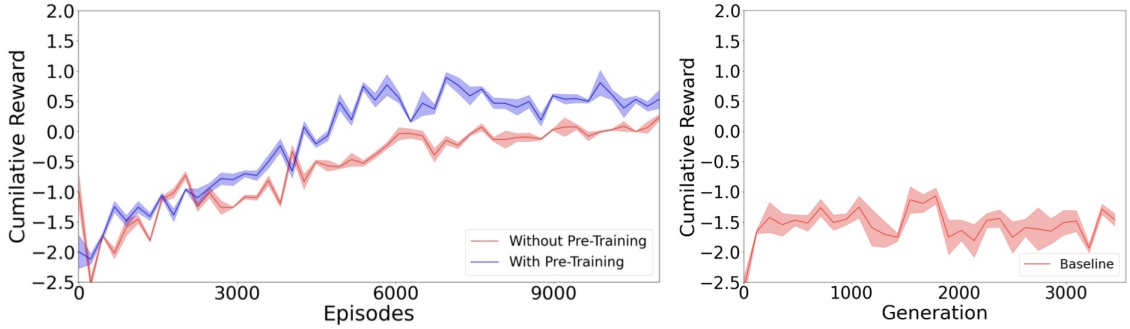

Figure 5: The left plot depicts the pre-trained agent (blue) and the agent trained only on the real environment (red). The right plot shows the Ha & Schmidhuber baseline.

bedding agent (Embed.), the Ha & Schmidhuber baseline, and the random agent used to collect training data. Thus the **Entity** world model leads to agents that avoid negative reward comparatively better than all our other world model baselines. One could be disappointed to see world models lead to worse outcomes than real environments, however, we note that using world models is less costly since they make it possible to train an agent with fewer interactions with the real environment. Further, we include the scores of the two agents from the pre-training experiment in Table 5. It shows that the pre-trained agent outperforms the agent trained only in the real environment.

Limitations

We focus on the domain of Pong in this paper, which is a major limitation. While focusing on 1-2 domains (including Pong) is not unusual for world model work (Oh et al. 2015; Ha and Schmidhuber 2018), we need to apply our approach to other domains to demonstrate generality. However, two problems must be solved first: the issue of a moving camera and a greater number of entities. We intend to address these issues with a change of representation and a change in entity identification process. In our Training Agents subsection, we converted the output of our predictive model to a pixel-based image to train DQN agents. This may lead to needless complexity and make the learning task harder for the DQN agent. We hope to train a DQN agent using the entity-based representation directly, which has a simpler structure than pixel-based image input.

Conclusions

Despite the huge success of modern RL algorithms, training them in a real environment can be dangerous or expensive. While approaches to learn simulated models of environments exist, they require a large amount of interactions with the real environment. We introduced a novel entity-based representation and world model as a means of beginning to address this issue. We demonstrated that our approach outperforms more standard solutions and an existing world model baseline.

Acknowledgements

This work was funded by the Canada CIFAR AI Chairs Program. We acknowledge the support of the Alberta Machine Intelligence Institute (Amii) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Duarte, J.; and Vlimant, J.-R. 2022. Graph neural networks for particle tracking and reconstruction. In *Artificial intelligence for high energy physics*, 387–436. World Scientific.

Ha, D.; and Schmidhuber, J. 2018. World models. *arXiv preprint arXiv:1803.10122*.

Halevy, A.; Norvig, P.; and Pereira, F. 2009. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2): 8–12.

Hansen, N. 2006. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, 75–102.

Jadhav, M.; and Guzdial, M. 2021. Tile embedding: a general representation for level generation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 17, 34–41.

Kaiser, L.; Babaeizadeh, M.; Milos, P.; Osinski, B.; Campbell, R. H.; Czechowski, K.; Erhan, D.; Finn, C.; Kozakowski, P.; Levine, S.; et al. 2019. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Karth, I.; Aytemiz, B.; Mawhorter, R.; and Smith, A. M. 2021. Neurosymbolic map generation with vq-vae and wfc. In *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, 1–6.

Khameneh, N. Y.; and Guzdial, M. 2020. Entity embedding as game representation. *arXiv preprint arXiv:2010.01685*.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A. A.; Yogamani, S.; and Pérez, P. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274.

Leibfried, F.; Kushman, N.; and Hofmann, K. 2016. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*.

Lotter, W.; Kreiman, G.; and Cox, D. 2016. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*.

Mandziuk, J. 2010. *Knowledge-free and learning-based methods in intelligent game playing*, volume 276. Springer.

Mawhorter, R.; Aytemiz, B.; Karth, I.; and Smith, A. 2021. Content reinjection for super metroid. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 17, 172–178.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Oh, J.; Guo, X.; Lee, H.; Lewis, R.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*.

Osborn, J.; Summerville, A.; and Mateas, M. 2017. Automatic mapping of NES games with mappy. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 1–9.

Sanchez-Gonzalez, A.; Godwin, J.; Pfaff, T.; Ying, R.; Leskovec, J.; and Battaglia, P. 2020. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, 8459–8468. PMLR.

Sara, U.; Akter, M.; and Uddin, M. S. 2019. Image quality assessment through FSIM, SSIM, MSE and PSNR—a comparative study. *Journal of Computer and Communications*, 7(3): 8–18.

Schmidhuber, J.; and Huber, R. 1991. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2(01n02): 125–134.

Sieusahai, A.; and Guzdial, M. 2021. Explaining Deep Reinforcement Learning Agents in the Atari Domain. In *Seventeenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Simard, P. Y.; Steinkraus, D.; Platt, J. C.; et al. 2003. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3. Citeseer.

Smirnov, D.; Gharbi, M.; Fisher, M.; Guizilini, V.; Efros, A.; and Solomon, J. M. 2021. Marionette: Self-supervised sprite learning. *Advances in Neural Information Processing Systems*, 34: 5494–5505.

Thielscher, M. 2017. GDL-III: A description language for epistemic general game playing. In *The IJCAI-16 workshop on general game playing*, 31.

Wang, E.; Kosson, A.; and Mu, T. 2017. Deep action conditional neural network for frame prediction in Atari games. Technical report, Technical Report, Stanford University.