Forward Model Learning with an Entity-Based Representation for Games

by

Nazanin Yousefzadeh Khameneh

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Nazanin Yousefzadeh Khameneh, 2021

Abstract

Reinforcement learning (RL) is a powerful way of solving sequential decision-making tasks in which the agent's goal is to learn how to maximize its reward. RL approaches can be divided into 2 different categories: model-based approaches that learn with the help of a model of the environment, and model-free approaches where the agent only learns by maximizing its reward. However, training an RL agent in a complex environment requires a vast amount of training data which can be expensive in tasks where the agent is expected to operate close to humans. In this work, we explore building a simulated environment, using a new representation of game frames. We use this representation to pre-train an agent and transfer the learned policy to the real environment. Our model can simulate the changes of a real environment in response to an agent's action and returns a reward value. Our major contribution is presenting a new entity-based representation for game frames and using the proposed representation as our baseline to train our virtual environment. Our work outperforms an existing method for learning a model of an environment with significantly less training data.

Preface

Chapter 3 of this thesis has been published at the Experimental AI in Games an AIIDE 2020 Workshop as: Khameneh, N. Y., Guzdial, M. (2020). Entity embedding as game representation. arXiv preprint arXiv:2010.01685.

Acknowledgements

Foremost, I wish to express my sincere appreciation to Professor Matthew Guzdial for generously sharing his wisdom with me throughout my research. He taught me to express my goals clearly and precisely. Without his persistent guidance, the goal of this thesis would not have been realized. I wholeheartedly appreciate his strong support and great advice which proved monumental towards the completion of this work. As a young researcher, I was fortunate to be in a place to learn my first steps from such amazing supervisor. I am indebted to my bigger family at the University of Alberta, specially Amii members whose inspirational discussions motivated my academic life. Last but not least, I would like to thank my lovely family back in my hometown and my supportive friends here, specially my dear husband Faraz, for all the passion and love they devoted to me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Imagination is the ability to produce and simulate novel objects, sensations, and ideas in the mind without any immediate input of the senses. Learning and understanding the internal representation of physics of the world gives a human the capability of predicting the future and imagining different real situations in their mind. This helps humans to learn quickly, plan, and react in critical situations in milliseconds [18].

Many state-of-the-art machine learning methods such as reinforcement learning (RL) and deep learning (DL) algorithms have been used to train agents that can accomplish human tasks. RL is a framework for defining and computationally solving sequential decision-making problems and DL models are powerful tools to handle large datasets by finding patterns among the data that can be used in decision making. Game-playing agents are among one of the most popular successes of both RL and DL [26].

Most of the impressive RL approaches are model-free. Model-free methods do not employ a model of the environment, they only focus on maximizing agents' future return by choosing the best action according to the given state, which is equivalent to finding the optimal behavior policy. With complex environments, model-free agents may require tens or hundreds of millions of interactions with an environment to learn the task. On the other hand, some other recent RL methods have been proposed to train agents with the ability to learn a model of the environment beside finding the best policy. These models are analogous to the internal models humans learn about their surrounding

environments. Knowing the environment can drastically reduce the number of interactions between the agent and the environment [15]. However, all these methods still need substantial amounts of training data through a direct interaction of the agent and environment and these interactions can be costly, such as when agents are expected to operate around humans, like self-driving cars [20].

Let's assume two different scenarios of a child who wants to learn how to play ping-pong: a sport in which two or four players hit a lightweight ball, known as the ping-pong ball, back and forth across a table using small rackets[38]:

In our first scenario, the child has the ability to learn the rules and the reward of an action at each time step. For example, if a child misses the ball then the child loses a point. After learning the rules, this child is able to play the game even in their own imagination without physically sitting and playing the game. In our second scenario, the child, instead of learning the rules ahead of time, just takes actions, and they are just told when they lose or gain points. This child is not able to play the game in their own imagination since they do not know anything about the game, they just learn how to map the current state to the best action in order to achieve the highest score.

Now, let's think of a robot instead of a child in our first scenario, where the robot will now attempt to learn the rules of the game. Unlike the child, the robot would likely need millions of interactions with the environment to learn the rules. One possible reason is that the robot does not know anything about the components of the environment, such as the ball, rackets, or even the table. It needs to play and collect large amounts of data to just understand that objects of the game differ from each other. Afterward, the robot can focus on the temporal features including ball movement, ball bouncing, and so on.

This is a metaphor for how an agent might learn to play the game. We include it to demonstrate the importance of how we think about the environment or represent the environment and how that impacts learning. This idea has become a key component in many recent research projects to develop a predictive model of the environment for the agent in order to interact with

a virtual version of the environment as their internal model. Over the years deep neural network methods have shown outstanding results in many visual problems such as object detection [39], image classification [30], video prediction [23], and so on. However, modeling video is still a very challenging issue due to the high dimensionality and complex temporal dynamics of the frames. Also, in some of these problems future frames are conditioned on both previous frames and control variables [28]. A few research projects have also focused on presenting the joint prediction of both future frame and reward for reinforcement learning applications [37]. To train an accurate predictive model we have traditionally needed a vast amount of data collected from the real environment [23]. One possible reason for this problem is that most DL methods require large amounts of training data when we use very complex and highly variable representations [11]. However, if we had a less variable representation, we might be able to learn a model of the environment in less time.

In the attempt to find a method to learn the internal model of an agent with less training data, We first proposed an entity-based representation for our game environment in a compressed space with less complexity compared with pixel-based images which contains the required information of each game state. We then use our proposed representation to train an action-conditioned predictive model as the agents internal model to predict both reward signals and the next frame. We also use this model to train an agent in its internal world model to investigate if this approach meet our goal which is obtaining a usefull forward model with less training data. While the high-level goal of the research projects in this area is just to train an accurate model, our aim is to train a model with significantly less training data.

Concretely, this thesis makes the following contributions:

- Proposes a new entity-based representation for game frames.
- Explores the entity-based representation with the use of Variational Autoencoder (VAE).
- Trains a forward model on the entity-based representation frames.

- Demonstrates that the forward model outperforms an existing forward model learning method.

- Demonstrates that an RL agent trained with our forward model as a virtual environment outperforms an agent based on this existing virtual environment work.

This thesis is structured in four more chapters. In Chapter 2, we provide preliminary background materials including RL, different DL models, and requirements for forward model training. Chapter 3 is dedicated to describing our entity-based representation approach as a game representation and how it can be applied to different problems. Further in Chapter 4, we provide details of our predictive model and evaluate our results and compare them with the state-of-the-art approaches. Finally, the last chapter is the conclusion, where we provide some limitations and possible future work.

Chapter 2

Background

This chapter introduces the preliminary knowledge for the following chapters. We provide background on Reinforcement Learning (RL), Deep Neural Networks (DNNs) such as Autoencoders, Variational Autoencoders (VAEs), and Long Short-Term Memory Recurrent Neural Netwoks (LSTM RNNs). We then discuss and introduce how to represent and learn game contents. Finally, we close the chapter with a discussion of Forward Model learning and its challenges.

2.1 Reinforcement Learning

An RL agent interacts with its surrounding environment by taking actions, and receiving rewards and observations in return. Unlike supervised learning that deals with a set of labeled examples, the RL agent is not provided with labels: it learns with trial and error which leads to learning which action gives rise to the maximum reward [2]. Markov decision processes (MDPs) are the standard mathematical formulation used for RL problems. An MDP is a mathematical framework used for modeling decision-making problems where the outcomes are partly random and partly deterministic. An MDP is described by the tuple $\langle S, A, T, \gamma, R \rangle$, where at each timestep, $s \in S$ indicates the current state, $a \in A$ is the choice that the agent makes at the current timestep, which causes the environment to transition to the new state $s' \in S$ with transition function $T(s, a, s') = Pr(s'|s, a)$ and returns reward $r : S \times A \rightarrow R$. The reward function indicates the 'pleasure' or 'pain' of taking particular actions

in particular states by emitting positive or negative reward. Finally, γ is the discount rate representing the importance of future reward compared to the immediate reward. The goal of the RL agent is to maximize discounted cumulative reward by finding the optimal policy $\pi : |S| \times |A| \rightarrow [0, 1]$. A Policy $\pi(a|s)$, returns the probability of taking an specific action a at state s.

As we discussed, the goal of the RL agent is to maximize the discounted cumulative reward over a long period of time. This weighted sum is called the return and we denote it as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \tag{2.1}$$

The expectation of the return for each state indicates the estimation the degree to which a specific state is good or bad. This expectation is called the value function. A value of a state, is the expectation of the return under the given the policy(π):

$$v_\pi(s) = E_\pi[G_t|S_t = s] \tag{2.2}$$

It is also useful to define a state-action value function which is denoted as $Q^\pi(s, a)$ is the expected sum of discounted reward at state s for a specific action a following policy(π).

$$q_\pi(a, s) = E_\pi[G_t|S_t = s, A_t = a] \tag{2.3}$$

The goal of most RL algorithms is to find the optimal state-action value function under the optimal policy.

In this thesis we do not focus on RL methods directly, we just evaluate our approach by training the RL agent on their internal model of the environment.

2.1.1 Q-Learning

To solve RL problems, we need to keep track of states, actions, and their expected rewards. One possible way to do this, is to save the required information in a Q-table. A Q-table is a data structure which maps state-action pairs to a Q-value (value function) which the agent will learn. Q-Learning is

an iterative algorithm to find the best state-action value function with given update rule:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t[R(s_t, a_t) + \gamma max_a q_t(s_{t+1}, a) - q_t(s_t, a_t)] \quad (2.4)$$

where s_t and a_t denotes the state and action at timestep t and q_t denotes the estimate of optimal state-action value function at timestep t. Finally, α_t is the learning rate at timestep t. Before learning begins, q is initialized to an arbitrary fixed value. Then, at each time t the agent selects an action a, observes a reward r, enters a new state s' that may depends on both the previous state s and action a, and q is updated.

We draw on Q-learning in this work. Our focus is not on optimal performance, and there are other RL algorithms that we could use. We focus on Q-learning due to its generality and popularity

2.1.2 Deep Q-Networks

In Q-learning, when the state and action space are discrete and relatively small, a Q-Table can be used to store the Q value of each state-action pair. However, when the state and action space are high-dimensional and continuous, a Q-Table is not appropriate. Deep Q-Networks (DQN) are possible solutions for dealing with high dimensional data in a continuous space. The Deep Neural Network (DNN) acts as a fuzzy Q-table, allowing the agent to approximate the Q values of unseen states and actions or generalize between them.

DQN is one of many algorithms that combines Deep Learning (DL) and RL to learn strategies directly from high-dimensional raw data. The state is generally the input to the network and the output is the state-action values for any action. In the original paper [26], the network architecture was designed to process image input from Arcade Learning Environment (ALE)[3]. The agent's experiences at each timestep, $e_t = (s_t, a_t, r_t, s_{t+1})$ were stored in a dataset and pooled over many episodes into a replay memory. To update the Q-values they applied Q-Leaning updates to random samples of experiences. After performing experience replay, the agent selected and executed an action according to an ξ-greedy policy. They also trained the Q-function on

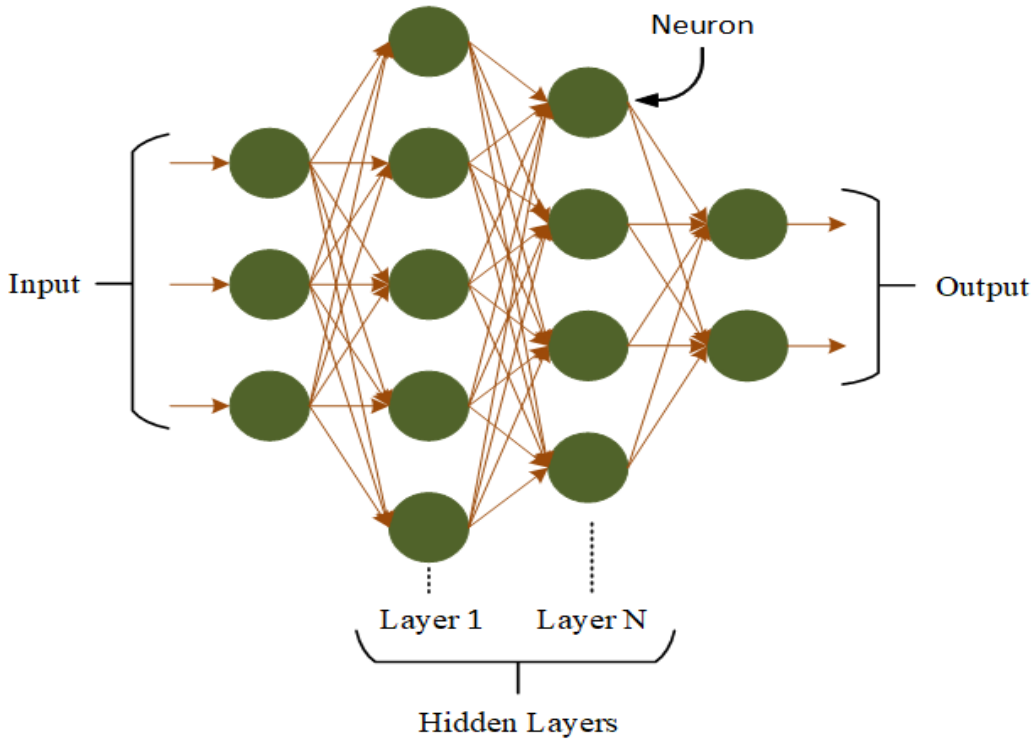


Figure 2.1: Deep Neural Network Architecture

a fixed length representation of histories. In this work we also use the same architecture on the Atari 2600 environment in order to evaluate our approach.

We conclude this section by explaining that DQN is a model-free RL algorithm, which estimates action values directly from experience. In other words, in model-free algorithms the agent does not learn how taking action a_t at state s_t changes the environment, it learns how good or bad taking action a_t at state s_t is. Training an accurate DQN may require millions of interactions between the agent and the environment in complex environments [27]. In comparison, model-based approaches achieve better performances with fewer environmental interactions [15]. Decreasing the number of interactions between the agent and the environment can improve data efficiency. This can be valuable in situations where the computational resources are significant. We can recall our earlier example here, self-driving cars which require significant resources.

2.2 Deep Neural Network(DNN)

DNNs have recently become a standard tool for many machine learning tasks. They are composed of artificial neurons, which are inspired by biological neurons.

DNNs have multiple hidden layers between input and output layers. Each layer consists of artificial neurons. The information from the input layer propagates through network to reach the output layer. Ideally, each hidden layer derives high-level information from the data.

DNNs process input into predicted output, and use the difference between the true output and the predicted output to update towards the final model The whole process occurs many times until it reaches local minimum in terms of loss. The learning problem can be represented as a minimization of the loss function. The loss function is a function that measures the performance of the model by how accurate the predicted result is.

In this thesis we apply a specific type of autoencoder called a variational autoencoder which is a type of DNN to explore the embedding of our proposed game representation. We then employ Long short-term memory (LSTM) Recurrent Neural Network for our predictive model.

AEs were first introduced by Hinton [31] to address the problem of backpropagation without a teacher by using the input data as the teacher. AEs are unsupervised machine learning methods where the input is the same as the output. They encode the input to a new space and then reconstruct the output from this representation. An AE consists of 3 components: encoder, latent space, and decoder. The encoder encodes the input to a latent space, the decoder then reconstructs the input from the latent space representation. Copying the input to the output may sound useless, but training the AE to perform the input copying task will result in taking on useful properties in the latent space. One way to obtain useful features from the AE is to constrain the latent space to have a smaller dimension than the input. These kinds of AEs are called undercomplete [7]. The latent space usually consists of most salient features of the input. We can describe the whole learning process as

below:

$$L(x, g(f(x)) \tag{2.5}$$

Where L is the loss function, x is the input, f tries to compress the data to a latent space and g reconstructs the input from the latent space. For example, if we use RGB images as inputs we can represent this as $x_i \in R^n$, where each x_i is a pixel. Also, mean pixel-wise squared error can be used as a loss function.

2.2.1 Variational Autoencoder (VAE)

In the previous section we mentioned that AEs use both an encoder and decoder to reconstruct the input data, however, they struggle to generate a new content from the encoded features. VAE can be defined as an AE in which instead of encoding an input to a single point, it is encoded to a distribution over the latent space [19]. In practice, the encoded distribution is chosen to be a normal distribution so that the VAE tries to approximate the mean and covariance matrix that describes the normal distribution. The distributions returned by the encoder are enforced to be close to a standard normal distribution. The latent space should have 2 important features:

- Decoded outputs from close points in the latent space should not be completely different
- For a chosen distribution, a sampled point from the latent space should give meaningful content once decoded.

The VAE loss function consists of two terms, the generative loss function to minimize the error between the input and output. The second term is the latent loss, which is the KL divergence that measures how closely the latent variables match a normal distribution. The loss function l_i for one data point x_i is:

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}[log\ p_\phi(x_i|z)] + KL(q_\theta(z|x_i)\ ||\ p(z)) \tag{2.6}$$

$$K_L(P||Q) = \int_{-\infty}^{\infty} p(x)\ log\left(\frac{p(x)}{q(x)}\right)\ dx \tag{2.7}$$

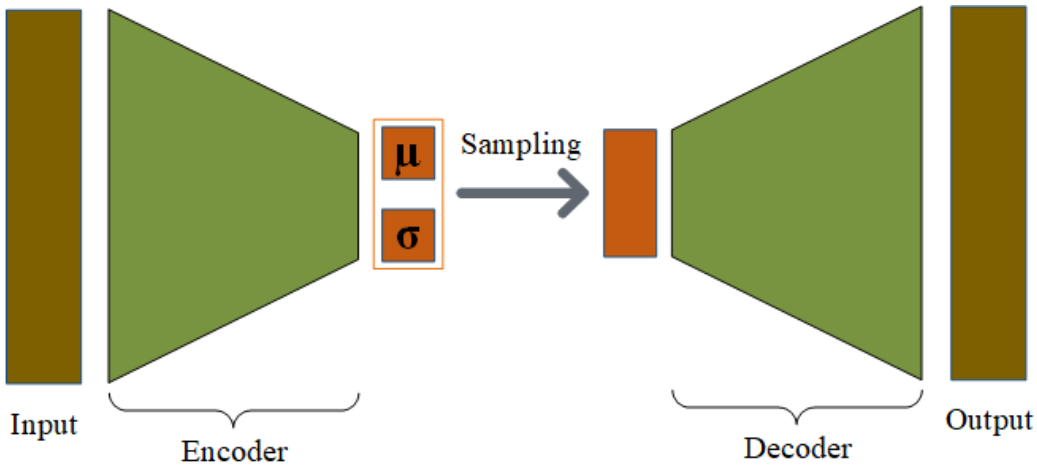


Figure 2.2: Variational Autoencoder Architecture

The total loss is $\sum_{i=0}^{N} l_i$ for N total data points. As we discussed above, the first term in the formula, is the expected negative log-likelihood of the i_{th} data point. This term encourages the decoder to reconstruct the input data from the latent space. The second term is the Kullback-Leiber (KL) divergence between the encoded distribution and $p(z)$ which is a unit Gaussian distribution. KL divergence measures how two different distributions over the same variable differ from each other

In this work, we have done experiments by training the VAE on an entity-based representation of Atari frame inputs which we will go through in more detail in the next chapter.

2.3 Recurrent Neural Networks(RNN)

RNNs are a type of deep neural network which use sequential data or time series data. These networks are used for temporal problems, such as language translation[1], natural language processing(NLP)[6], image captioning[14], next frame prediction[29] and so on. RNNs are distinguished from other types of DNN by memorizing information from previous inputs to affect the current output. They also share the same parameters across all timesteps. The RNN problem can be formalized as:

$$h_t = f(x_t, h_{t-1}) \tag{2.8}$$

where h_t is the current hidden state, x_t is the current input and h_{t-1} is the previous hidden state. f is usually the non-linear transformation which is applied to x_t and h_{t-1}. In RNNs, the hidden layer h takes part in memorizing the long term information.

Vanilla RNNs suffer from the problem of vanishing gradient when they want to learn and remember long-term temporal dependencies. LSTM networks are a type of RNN that can handle the problem of long-term temporal dependencies using frequent gates update on every timestep of the learning process. Gates are the part of LSTM's architecture which we will talk in more details in the next section.

2.3.1 Long Short-Term Memory (LSTM)

LSTM networks are capable of learning order dependencies in sequence prediction problems. They were introduced by Hochreiter & Schmidhuber [13]. Compared to standard RNNs, an LSTM unit has a memory cell c_t for memorizing information. Also, LSTMs can control information flow at each cell, by structures called gates. An input gate i_t controls how much information to keep and the forget gate f_t controls how much old information to forget. They are computed given the current state and previous hidden layer as follow:

$$f_t = \sigma(W_f \; . \; [h_{t-1}, x_t] + b_f) \tag{2.9}$$

$$i_t = \sigma(W_i \; . \; [h_{t-1}, x_t] + b_i) \tag{2.10}$$

where W_i and W_f are weight matrixes, σ is a Sigmoid function, b_i and b_f are biases.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

Similarly, an output gate $o_t$which is also called peehole connections is calculated as:

$$o_t = \sigma(W_o \; . \; [h_{t-1}, x_t] + b_o) \tag{2.12}$$

This gate decides what memory content to output.

$$h_t = o_t * tanh(c_t) \tag{2.13}$$

Where '*' is the element-wise product between two matrices. Now new c_t can be calculated as a weighted sum of the previous memory cell and candidate values, $c_t^{'}$, can be added to the state.

$$c_t = f_t * c_{t-1} + i_t * c_t^{'} \tag{2.14}$$

$$c_t^{'} = tanh(W_c \, . \, [h_t, x_t] + b_c) \tag{2.15}$$

With the help of this whole architecture the LSTM is able to memorize longer sequences.

LSTMs also were proven to be quite successful in video frame predictions. However, training an LSTM on high dimensional data such as images with lots of complexities, needs vast amount of training data. To tackle this problem, we proposed a new entity-based representation for game frames. We will go through more details in the upcoming chapters.

2.4 Game Representations

Game playing is an important domain of artificial intelligence. Video games provide a platform for DL models to learn how to interact with dynamic environments and solve complex problems, just like in real life. Video games have been utilized for decades to evaluate artificial intelligence (AI) agents performance [34]. One of the fundamental topics in intelligent game playing is the problem of efficient game representation [25]. Many DL algorithms are given each video game frame as an input. A single level played may consist of tens of thousands of frames, and each individual frame contains a lot of information. Game frames, take in all the necessary information of each game, like main characters and background objects. Also, by looking at the sequences of frames, the game rules can be captured [8]. For example, in Pong from Atari, by looking at each frame, we can see game entities like the ball, paddles, scores and so on. Additionally, to extract game rules such as, bouncing the ball and paddle movements, we need to consider sequences of frames.

There are different approaches to deal with sequential game frames. Many DL approaches have shown great success in many visual perception prob-

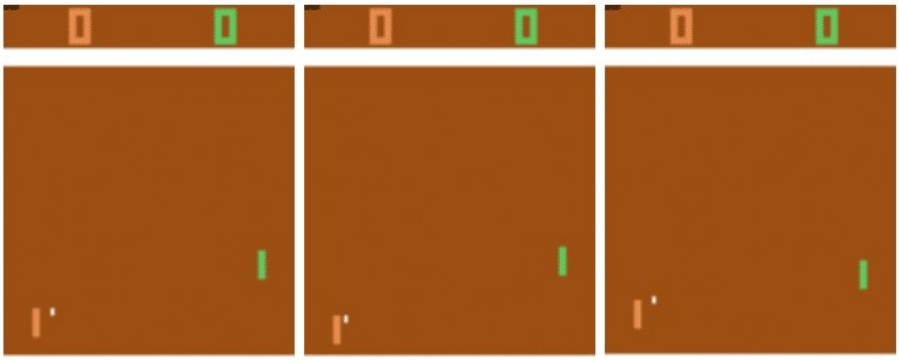


Figure 2.3: 3 Consecutive Pong Frames.

lems. Convolutional Neural Networks (CNNs) [32] and Recurrent Neural Networks(RNNs) [36] deal with both pixel input images and sequential data, respectively. Since images are high dimensional data structures, one major aspect to balance the training time and model performance is pre-processing the image inputs. However, different games can have totally different entities and rules, which means many of these DL models are game specific and can not be applied to different game domains.

In this project, we propose a new entity-based representation for frame inputs. We explore training a VAE on these frame entities. We then train an LSTM on this new representation as our predictive model.

2.5 Forward Model Learning

As we mentioned in the introduction, training RL agents in a real environment can be costly. One way to offset this problem is to derive an accurate simulated environment [21]. The initial idea was introduced by Schmidhuber & Huber [33]. With this idea, we can train an agent in the simulated environment and transfer the learned policy to the real environment. The simulated environment can be understood as an agent's internal model of the environment. However, acquiring a sufficiently accurate model is a challenging problem and

in many cases needs many data samples which is expensive when the agent operates close to humans. Oh et al. [28] proposed a highly accurate video frame action-prediction model using a combined RNN and CNN architecture. In this work they used a DQN agent to generate gameplay video frames using an $\epsilon-$greedy action selection policy. They also used 500,000 training game frames for each game. Later, Leibfried et al. [22] extended the work of Oh et al. by including reward prediction. Wang et al.[37] proposed a convolutional feed-forward model for predicting future frames and rewards. They also trained their model with around 600,000 frames of Pong from Atari. However, relying on a trained DQN to collect the training data frames is not ideal because it is not possible to have a trained DQN unless it is already trained in the real environment, which makes learning a simulated environment useless.

Ha & Schmidhuber[10] presented a predictive model that can be trained quickly in an unsupervised manner to learn a compressed spatial and temporal representation of the environment using two models: a VAE and LSTM. They evaluated their model on the Car Racing environment [5] and Doom [16]. They trained their model on a dataset of 10,000 random rollouts of the environment. They first trained their VAE which is called V model. Then they trained the LSTM (M model) with the latent representation of their V model. Their model also contains a controller, which is responsible for choosing the action in order to maximize the cumulative reward. Kaiser el al.[15] proposed a complete model-based deep RL algorithm based on video prediction models. They presented a predictive model, called Simulated Policy Learning (SimPLe), and used the model to train a policy to play the game within the learned model.

In this work, we propose a method in which by extracting the useful information from images we significantly decrease the required size of the training set. Additionally, we trained a VAE on the extracted information to investigate the embedding of our proposed representation. Finally, we train an LSTM to predict both next frame and the reward, given the current state and the action. We trained our LSTM with the latent representation and the original entity-based representation. In the upcoming chapters we will talk about the details of our approach.

Acknowledgements

Foremost, I wish to express my sincere appreciation to Professor Matthew Guzdial for generously sharing his wisdom with me throughout my research. He taught me to express my goals clearly and precisely. Without his persistent guidance, the goal of this thesis would not have been realized. I wholeheartedly appreciate his strong support and great advice which proved monumental towards the completion of this work. As a young researcher, I was fortunate to be in a place to learn my first steps from such amazing supervisor. I am indebted to my bigger family at the University of Alberta, specially Amii members whose inspirational discussions motivated my academic life. Last but not least, I would like to thank my lovely family back in my hometown and my supportive friends here, specially my dear husband Faraz, for all the passion and love they devoted to me.

Chapter 3

Entity-Based Representation for Game Frames

The goal of this work is to propose a method that can understand the physics of the environment and make predictions of future states. However, many previous works used pixel-based input images which are high dimensional input data. In this chapter, we introduce our entity-based representation, and we explore if we can develop a method for embedding entities from multiple games. The proposed representations will then be used in the next chapter as the basis for our forward model.

3.1 Entity-Based Representation

In this thesis we present a new representation for games that we call our "entity-based representation". We focus on the domain of Atari games to test this approach, as the games are relatively simple while still being more complex, which makes hand-authoring knowledge for them non-viable. We identify dynamic information (e.g. position) automatically from Atari games by running an algorithm introduced by Guzdial et al. [8]. We collect and vectorize these features for each entity, based on the extracted information from the frames. In this representation we can represent changes over entities as vectors (one entity at one point becoming another entity at another point), and whole games as graphs or point clouds (where each point is an entity in the game).

3.1.1 Feature Extraction

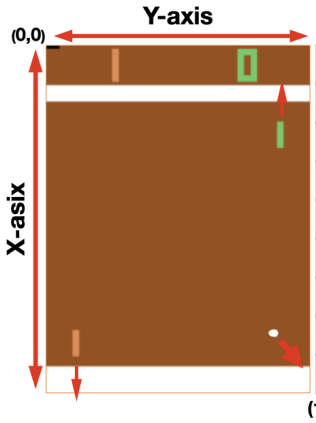
Our goal is to have a low dimensional representation in comparison to pixel-based images that reflects the semantics (mechanics) of the entities and maintains the needed information to train a predictive model. To obtain the required information, we first identify the background color for each frame, to avoid considering the background as an entity. We do this by finding the most common pixel color in a given frame. After obtaining the background color, we run a depth first search algorithm through the entire frame to identify every pixel that is not the background color. We also consider neighboring pixels to be neighbors in each frame if they have the same pixel color. Note that we assume that entities in an Atari games have a constant pixel color, and the entire entities are structured so that every pixel in the entity is adjacent to at least another pixel

Below is the list of the types of the entity information we use in this work:

- *Animation* contains $SizeX$ $SizeY$ of the entity.
- $VelocityX$ indicates the velocity of the entity horizontally.
- $VelocityY$ indicates the velocity of the entity vertically.
- $PositionX$ is the value of an entity in the x dimension of a frame.
- $PositionY$ is the value of an entity in the y dimension of a frame.

3.1.2 Entity Vectorization

As we mentioned in the previous section, we extract physical information from the frames to obtain: entity positions in both the x and y direction, the size of each entity (width and height), and the velocity of each entity in both the x and y directions. After obtaining the physical information of each entity we save the mechanical information of each game entity as an integer in an individual vector of shape (1x6). For example, entity 'A' is represented as a vector which contains,$SizeX$, $SizeY$, $VelocityX$, $VelocityY$, $PositionX$, $PositionY$. We note that different in-game entities would generate multiple instances of this



	sizeX	sizeY	posX	posY	velX	velY
Left Score	10	2	1	18	0	0
Right Score	10	6	1	55	0	0
White line	5	80	12	0	0	0
Left Paddle	8	2	87	8	-2	0
Right Paddle	8	2	23	66	2	0
Ball	2	1	87	66	2	2
Bottom line	8	80	97	0	0	0
----	0	0	0	0	0	0

Figure 3.1: This figure depicts the entity-based representation of a frame. Red arrows indicates the movement direction of entities.

representation across each frame. Further, all the values had to be integers as they were measured over the space of pixels. We also added a vector of zeroes to our dataset and use this vector to show entities that were removed in the previous frame or were about to be added in the next frame. All of the features are always greater than zero except *VelocityX* and *VelocityY* which can be negative.

Figure 3.1, is an example of an entity-based representation for a game frame. Each row in the table is how we represent entities in each frame.

We use this representation as a basis for our forward model in the next chapter.

3.2 VAE-Based Entity Representation

In this section, we explore developing a model to learn an abstract, compressed representation of our proposed entity-based approach. Plus, it helps validate that entity-based representation is itself reach enough for tasks like differentiating between games [17].

Further, we introduce some qualitative examples along with some evidence towards its quality and future utility.

As we mentioned in the previous chapter, autoencoders are efficient tools

for dimensionality reduction. These tools approximate a latent structure of a feature set. In this work, we aimed to explore if we can employ the embedded representation of our entity-based approach. To this end, we use a VAE to reduce the dimensionality of the entities in order to learn an entity representation with less variance. We decided to make use of a VAE, as it would allow us to learn a more consistent latent space and potentially to sample novel entities from this learned distribution. We train our VAE on a dataset of entities in our entity-based representation to learn the parameters of a probability distribution to represent our entities. Since it is a generative model, we expect the VAE can also be applied to game content generation tasks [9] like generating entities similar to the input and blending the entities in the latent space.

We tried various VAE architectures with different hyper parameters. Since accuracy of predicted entities using a VAE with low dimensional latent space is our main goal, we choose the model with the possible lowest latent space dimension that can predict highly accurate entities. Our final architecture has one fully connected hidden layer with Relu activation in the encoder, which are then fed into a n-dimensional embedding layer with Sigmoid activation. We use 2 and 12 as latent space dimensions in different experiments.

The decoder section architecture is an inverse of the encoder section, starting with a Sigmoid activated fully connected layer. We implemented this model in Keras using the Adam optimizer with a learning rate of 0.001. We employed the sum of mean-square error and kl-divergence as our loss function. We assigned 0.001, a low value to the weight of kl-divergence loss and one to the weight of the mean-square error, since we aimed to have more accurate reconstruction of frames.

3.2.1 Qualitative Examples

In this section we discuss our experiments and some qualitative examples to explore the latent space.

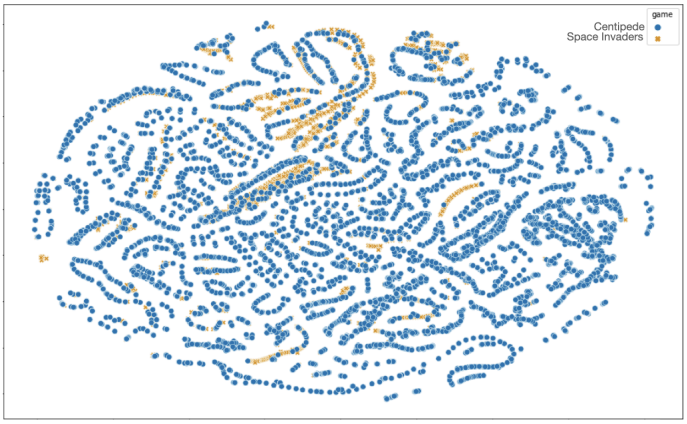


Figure 3.2: t-SNE Visualization of our latent space. Blue dots are entities from Centipede and yellow entities are from Space Invaders

3.2.2 Training VAE with Centipede and Space Invaders

In this example, we aimed to investigate how entities of different games are distributed in the latent space to explore if the latent space is really represents the semantics of each game.

We chose Space Invaders and Centipede. We trained the VAE with entities of 15,000 frames of both games. Since the initial goal of this thesis is to train a model with less training data, at first we use 10,000 frames we then gradually increase the number of frames to finally achieve the accurate VAE. We also choose Centipede and Space Invaders because both games have similar mechanics in which the player is a fighter who shoots at enemies. In Figure 3.2, we display the distribution of entities of both games in the latent space using the t-Distributed Stochastic Neighbor Embedding (t-SNE) technique. This technique is for dimensionality reduction and is well suited for the visualization of high-dimensional datasets [24]. Even though we use a 2-dimensional latent space. t-SNE is still helpful in terms of showing clear “clusters“ of points.

The representation depicts the distribution of entities in the projected 2D space. The reason that there are more Centipede entities than Space Invaders

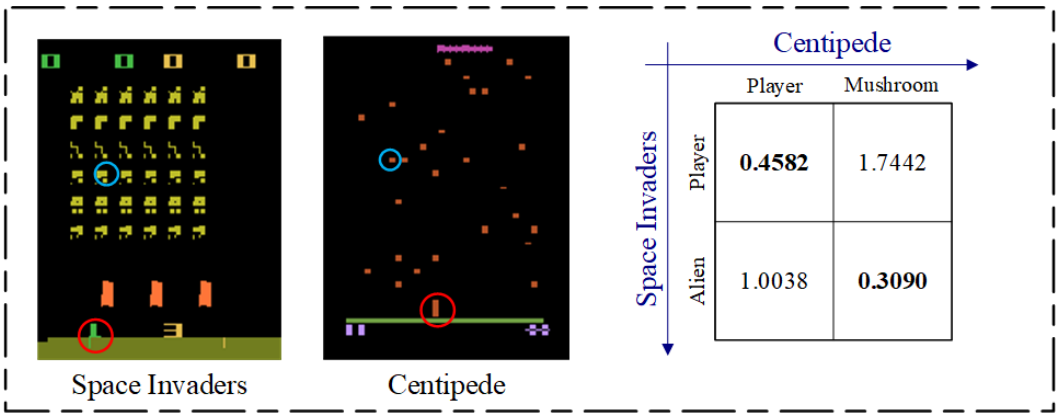


Figure 3.3: Two randomly selected frames of the games Centipede and Space Invaders. Entities in blue circles are the Alien and Mushroom from Space Invaders and Centipede respectively, both destroyable entities. Entities with red circles are player entities. The tables indicates the Euclidean distance of these entities in the latent space.

entities is that in Centipede each entity has many variations in different game frames. However, in Space Invaders, most entities do not vary substantially during the game round. As it is depicted, different groups of entities from Space Invaders are in the same cluster with entities of Centipede, this shows that different entities with similar physical features have less distance in the latent space. For example, 3.3, illustrates the Euclidean Distance between entities of two randomly selected frames from both Centipede and Space Invaders. As it is shown, the distance between players from both games is significantly less than the distance between the players and enemies of those games. This indicates that our latent space places mechanically similar entities closer to one another in its learned latent space. We hope to explore the possible research direction of this feature in the future.

3.2.3 Training Entity VAE with Pong

We also trained our VAE with entities extracted from 15,000 frames of the Atari game Pong, and used the trained model as a part of our forward model in some of our experiments in the next chapter. We again choose 15,000 since the one major goal of this thesis is to train an accurate model with less training data. In this section we will discuss some other qualitative examples. In this

Figure 3.4: Six random variations made to the right paddle in the latent space.

example we first pick a random entity from 5 consecutive frames and added a random vector in the range of (-0.5, 0.5) to that entity's embedding. As is shown in Figure3.4, our random entity is the right paddle which is surrounded by the red circle. The entity moves in different directions and has different sizes compared with the corresponding frame without the random variation. This example shows how original entities can be affected by small random variations in the latent space.

We also tested our VAE trained with Pong entities on Breakout from Atari. We chose breakout because in both Pong and Breakout some entities have similar mechanics, such as the ball and paddle. We show some generated frames in Figure 3.5. Although the generated frames are not playable frames and still have many differences compared with the original frames, we can see that the VAE is able to capture the ball movements, and the colorful lines still have the correct size, though they are in the wrong positions. In this example, we aimed to explore the possibility of using this approach as a shared representation for games with similar mechanics.

3.2.4 VAE-Based Entity Frame Representation

In this section, we discuss another experiment in which we train the VAE with entity-based frames instead of individual entities. In this model, each training instance is represented as a flatten vector of entities. For example, in Pong,

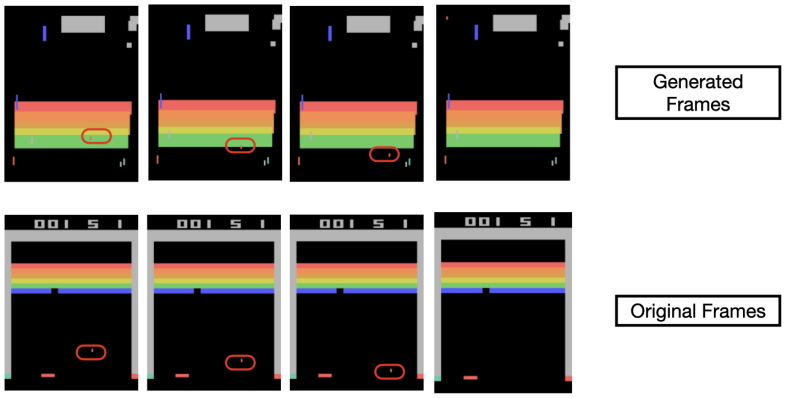


Figure 3.5: 4 consecutive original and generated frames of Breakout.

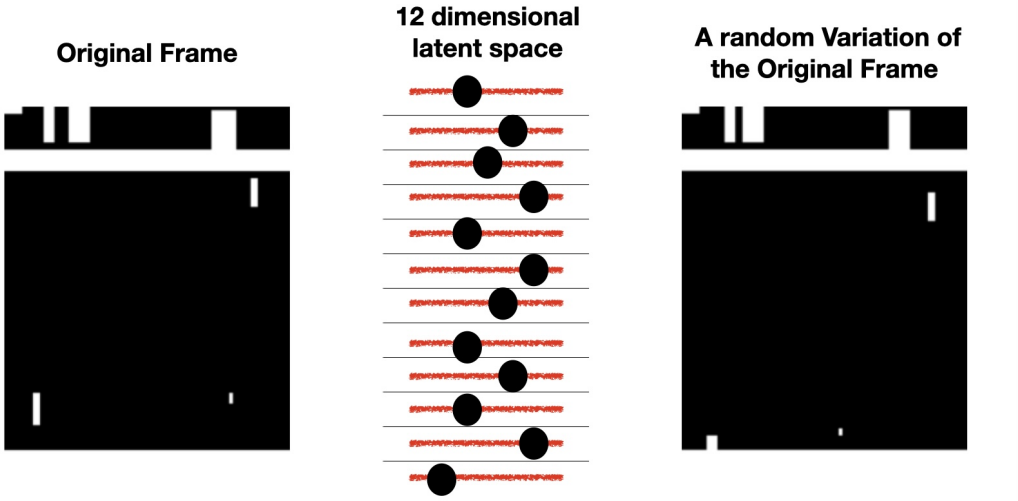


Figure 3.6: A random variation of a Pong frame generated from our VAE-based entity frames representation. As is shown, with a random variation of the frame in the latent space, the ball and paddles are in different position in the generated frame compared with the original frame

we have at most 9 entities in each frame, and each entity has a shape of 1×6, thus each frame has the shape of 1×54.

We train a highly accurate VAE with a low dimensional latent space (12-dimensional). We also use this model as a part of our forward model in one of our experiments in the next chapter.

3.2.5 Future Work

We first proposed a new representation for games based on mechanical features of game entities. We then trained different VAEs on mechanical features of entities in order to derive an embedding of entities and frames. We argue that this entity embedding can be a shared representation that enables various game content generation tasks. We list potential directions for future work below.

- *Entity Blending:* This representation can be used to generate new entities based on existing ones.

- *Speeding up Rule Learning*: As we discussed in the Ruleset section, the embedding is based on mechanical features of entities in each rule of a game. Each rule references a set of entities (group of conditional facts) together in a frame which causes an effect. We might expect similar mechanical effects if we have entities from another different game with a similar latent representation. We anticipate a need for another study to investigate this.

- *Extending the Dataset:* We trained this model on around 15,000 frames from three Atari games. Extending our dataset by adding other similar and dissimilar games is another potential future direction.

Chapter 4

Forward Model Learning

Predictive models can simulate how the real environment changes in response to an action from an agent. Unlike previous approaches which use high dimensional pixel input observations to make the prediction, we present a method that is able to learn the temporal and spatial dependencies of entities, using an entity-based representation, and make predictions with considerably less training data. In this chapter, we use our entity-based representation to learn the system dynamics of an environment and the reward structure jointly. We show that this approach can be used to improve performance by pre-training the RL agents in this simulated environment.

4.1 System Overview

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 frame, action and reward at timestep t, respectively. $s_{t-k:t}$ indicates frames from time $t-k$ to time t. Figure 4.1 demonstrates the whole structure of our approach which has 2 main components. Entity extraction converts the pixel inputs to entity vectors, and our predictive model to predict the next state and the reward. The predictive model is composed of LSTM layers that extract temporal and spatial features from the input data. 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 frame and reward. The major difference from this approach to existing, prior

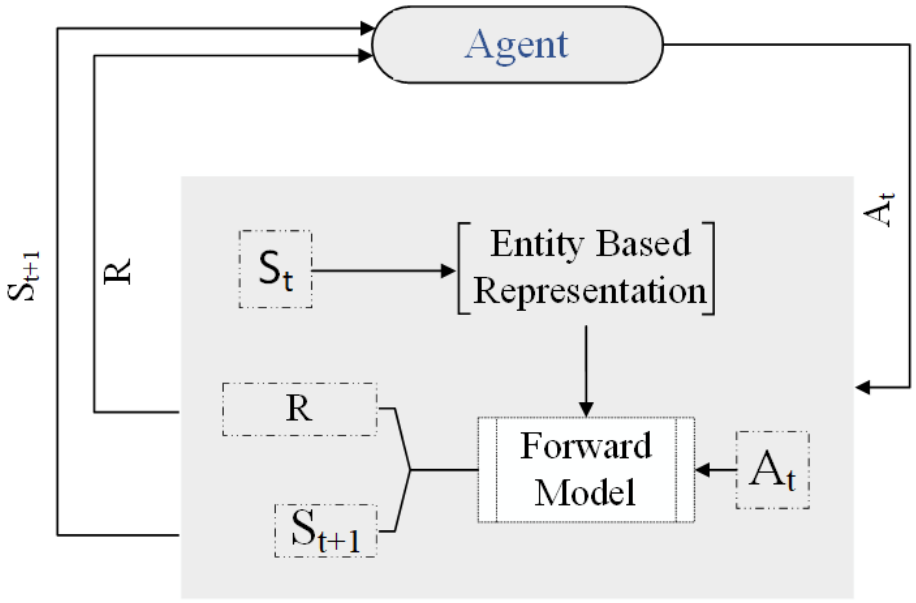


Figure 4.1: This figure demonstrates the whole structure of our approach. At each time step, the forward model takes the entity-based representation of the state and agent's action to predict the next state and possible reward value and send it back to the agent.

work is to use our entity-based representation instead of pixel input [28].

4.1.1 Dataset

We present an initial evaluation of our approach in the Pong game domain from ALE. We choose this game because it has a reasonable number of entities, with at most nine across all frames, including the ball, paddles, score and background items. We applied our model to Space Invaders which has around 80 entities, and Centipede with approximately 40 entities, nonetheless, the model could not perform sufficiently.

Pong is also a fixed camera game in which the scene does not change. We tried to evaluate our model in the Car Racing environment [5] in which the camera moves with the agent's action. However, the model had a poor performance in dealing with this kind of environment.

For our dataset, we collect 15,000 frames of the Pong environment. We have an agent act randomly to explore the environment, and record the random actions taken, the resulting observations, and the reward value from the

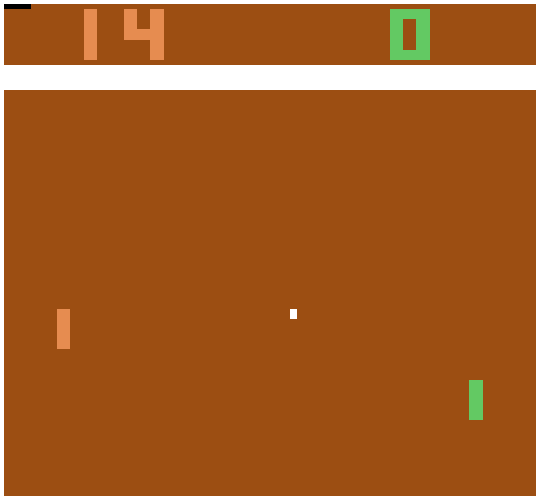


Figure 4.2: A frame example of Pong. There are two paddles, a ball, scores and some other background objects.

environment. The action is a one-hot encoded vector of shape 1×6. Possible actions in Pong are, moving down, moving up, and do nothing where

Reward values in many Atari games vary significantly between games. In Pong, the reward values are -1, 0, and 1. We extract the physical information of entities from frames as we mentioned in the previous chapter. The entity-based extracted features are used to train our forward model. In all experiments, each frame is represented as a fixed size vector of ordered entities. We have different numbers of entities in each frame, therefore we find all the possible kinds of entities and represent each frame with the maximum number of entities. In the case where there are fewer entities the remaining indices are filled with 0's, otherwise, they have a vectorized entity-based representation. We use the first 14,000 frames for our training set, and the last 1000 frames for our test set. We also train our forward model with e sequence of entity-based frames observed.

4.1.2 Model Architecture

Our network takes a fixed history of previous frames as an input. We choose 4 for the history frame time horizon as previous works also used this number for Pong [28]. The exact shape of the input varies depending on the entity-based

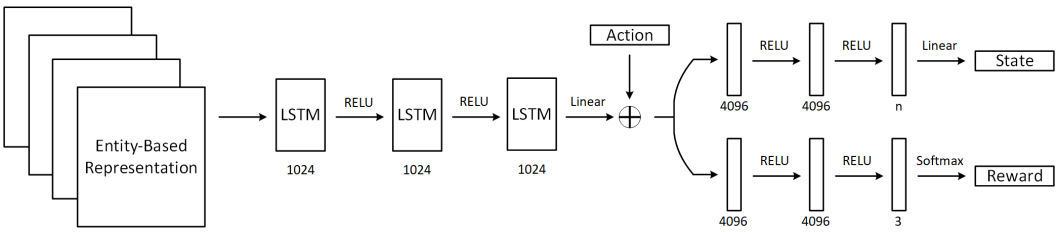


Figure 4.3: The Figure depicts the complete architecture of our predictive model.

representation, we will go through more details in the evaluation section. The network architecture consists of 3 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 4.3, after adding the action, the network is divided into two parts. Both parts are followed by three fully connected dense layers, which predict the next frame and the reward. In the first part, the output is a real-valued approximation of the vectorized representation of the entities. The second part employs a softmax-activated fully connected layer, which predicts the reward. We split the reward into three categories (-1,0,1) to make it a classification problem, we will go into more details on this below. We used dropout equal to 0.5, the Adam optimizer, a batch size of 64, and we trained the model for 200 epochs. We implemented this model in Keras.

4.1.3 Loss Function

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

$$Loss = (10 \times MSE) + (1 \times cross - entropy) \tag{4.1}$$

We exert the MSE loss function between the predicted frame and the next ground truth frame, where each frame is represented with a vectorized entity representation. In addition, we employ the cross-entropy loss function, which is popular for classification problems [35] to approximate the reward values.

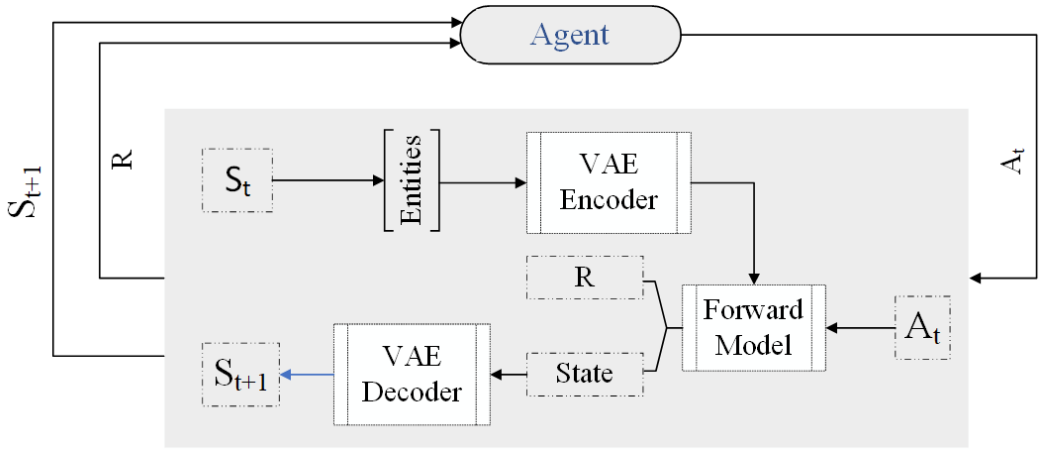


Figure 4.4: This figure illustrates our **Frame Embedding** and **Entity Embedding** experiment. Entity-based representation of state at time step t is first encoded to the latent space using the VAE. The encoded representation and agent's action then fed into the forward model to predict the next frame.

4.1.4 Reward Prediction

We tried different representations for reward values, such as considering the reward as a number. Since, around 95% of rewards in our dataset have a value of 0, the model was not able to predict non-zero rewards. In our final model, rewards are represented as one-hot encoding vectors of size 3 [22].

However, one major problem during reward prediction was the imbalanced data. 336 frames out of 15,000 have rewards of -1, 14,653 frames have 0 as reward and only 11 frames have +1 as their reward value. Although there are possible ways to deal with imbalance data such as sampling, we aim to train a model that can learn a sparse reward problem. As a result, many of our model variants struggled to predict positive rewards.

4.2 Evaluation

The entire purpose of this approach is to achieve a useful forward model that can predict accurate future frames and reward values. To find the best version of our method we employ four different versions of our entity-based representation.

- **Entity**: the original values of the physical information of entities are

used in the entity-based representation of frames. We wanted to investigate whether the direct entity-based representation can be employed in forward model learning. In this version the input size is 1×54 where 54 is number_of_entities $\times$ entity_dimension.

- **Entity Embedding**: the embedding of the entities are used in the entity-based representation of frames. Using this low-dimensional latent spaces enables us to increase the generalization. In this representation each frame has a shape of 1×18 where 18 is number_of_entities $\times$ latent_dimension_of_each_entity.

- **Output VAE**: the reconstructed output of the Entity VAE are used in the entity-based representation of frames. We included this because in-between approach, potentially is more general than the **Entity** representation, but has more detail than the **Entity Embedding** representation. In this approach each frame's shape is similar to frames in **Entity** representation.

- **Frame Embedding**: the embedded version of the entity-based frames. 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.

Further, to evaluate our work we use three different types of evaluations. To evaluate the quality of the predicted frames, we employ some similarity metrics between the predicted frames and the 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 pre-train a DQN agent in a subset of our simulated environments.

We also employ a previous attempt at learning a forward model as a baseline [10]. We chose this method as our baseline because it is the original world model paper. The world model uses a generative recurrent neural network, which can model RL environments through compressed spatio-temporal rep-

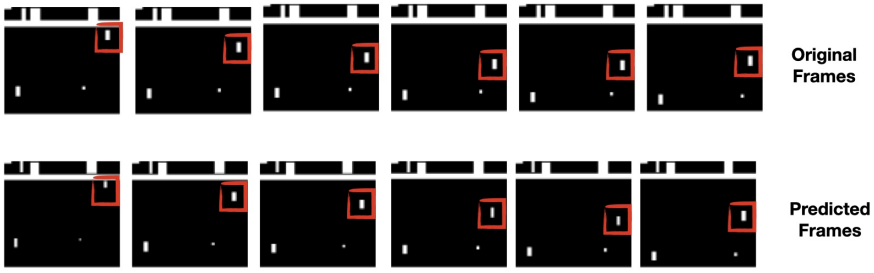


Figure 4.5: This figure depicts a consecutive predicted frames of our **Entity** predictive model. The right paddle which is surrounded by the red square is controlled by the agent.

resentations. This approach also works the best of subsequent world modeling papers with small datasets. Furthermore, we attempted to implement a model of [28], however, their model has around 90 million parameters, which means it would not be trainable with our small training set. We had to adapt our baseline to the Atari environment. Since the original paper evaluated their model using the Car Racing and Doom environments, we had to change the input parameters for Pong. Their model consists of a VAE to learn the compressed representation of the 2D image input. They also have an LSTM which learns to approximate the future latent vectors that their VAE is expected to produce. They also have a controller as a part of their model. This controller is responsible for choosing the action in order to maximize the expected cumulative reward. The controller is a single linear model that maps the latent space of the VAE and the output of the LSTM to the action a at timestep t. We trained the VAE and LSTM with our 14,000 training frames.

4.2.1 Accuracy of the 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 used to compare images. MSE calculates the average of the Euclidean distance of pixels. SSIM is another similarity metric that quantifies the image quality of the predicted frame compared to the original frame by looking for similari-

	SSIM	MSE	MSE of Entities
Entity	0.9677	0.1644	10.1733
Entity Embedding	0.8442	0.4534	15.4121
VAE Output	0.9441	0.1997	11.1356
Frame Embedding	**0.9694**	**0.1587**	**8.0112**
Baseline	-0.0951	2.4052	77.2876

Table 4.1: Comparison of our experiments over our test data, comparing the output of the forward models to the original game frame.

ties within pixels, such as, if the pixels in the two images have similar pixel density values. In addition, since the major 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 entity-based 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 both MSE between pixels and entity representation, and higher values for SSIM are better as they indicate fewer differences between the original image and the predicted one.

As seen in Table 4.1, the **Entity** and **Frames Embedding** representation of our proposed model outperform the others according to all similarity metrics. We observed that, in both of these versions, our model is able to capture the movement of entities, specifically the player-controlled paddle. The model also can predict the interactions between objects such as bouncing the ball from the paddle.

In comparison, our baseline is not able to capture the important entities of the frames, such as the ball and paddles. One possible reason for this problem is that their model requires 10,000 rollouts of frames for training while we just train the model with around 14 rollouts. Due to the importance of the accuracy of the predicted frames, we focus on the **Entity** and **Frame Embedding** forward models in our final evaluation.

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

Table 4.2: Comparison of our experiments over our test data, F1-score of the predicted rewards for negative rewards and zero value rewards.

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

Table 4.3: This table shows the F1-score of the predicted rewards over the training set.

4.2.2 Accuracy of the Predicted Rewards

To asses the validity of the reward prediction we calculate the F1_score, which is a measure of a model's accuracy on a dataset. The F1_score is defined as:

$$F1 = \frac{2 \times TruePositive}{TruePositive + \frac{1}{2}(FalsePositive + FalseNegative)} \tag{4.2}$$

Since our training set has around 95% zero value rewards, it is really important to us to check the validity of our model on non-zero rewards. In Table 4.2, we show the F1-score of all versions of our forward model over our test set for both 0 and -1 rewards. We did not include the +1 reward values in Table 4.2, as there were no examples of those values in our test set. Table 4.3 depicts the F1-score of the predicted rewards over our training set. As is shown, the **Entity** version has an F1-score near or equal to 1, which shows the ability of the model to predict both positive and negative rewards. Other versions also predict both negative and zero rewards with a high F1-score. However, they could not predict positive rewards. One possible reason is that we only had 11 positive rewards among our training set given that we used a random agent to collect the frames. We did not include our baseline in this section, since their approach does not predict rewards.

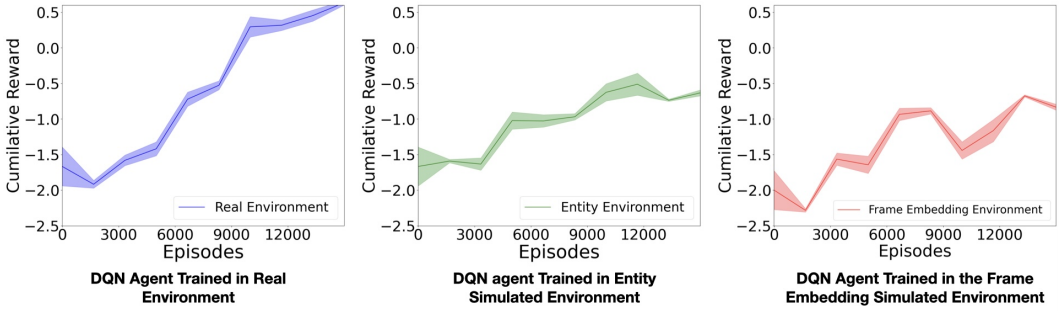


Figure 4.6: The plot shows the average reward per episode on Pong during training in simulated environment and real environment.

4.2.3 Usefulness of the Forward Models

To evaluate the usefulness of the trained forward models we trained DQN agents [26] using the **Entity** and **Frame Embedding** forward models. We then test the trained agents in the real environment to explore if they could transfer the learned policy from the simulated environment to the real environment.

The original DQN agent used 84×84 pixel inputs [27]. However, the output of our forward models is not a pixel-based image, therefore we convert the forward model's output to a gray-scale 210×160 frame, we then resize the image to 110×84 by down-sampling. Finally, we obtain the final input by cropping an 84×84 square area of the frame. Also, as is mentioned in the original paper, a single frame does not have all the necessary information for the agent, to tackle this issue, they stacked the last four frames to produce the input to the DQN agent. We also stack the last four predicted frames of our predictive model to create the input to the DQN agent. The final input shape for the DQN agent is $84 \times 84 \times 4$. In the original paper, the authors applied a simple frame-skipping technique from [4] 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 our models automatically predict the k^th frame. We choose $k = 4$ as the original paper suggests the same value for

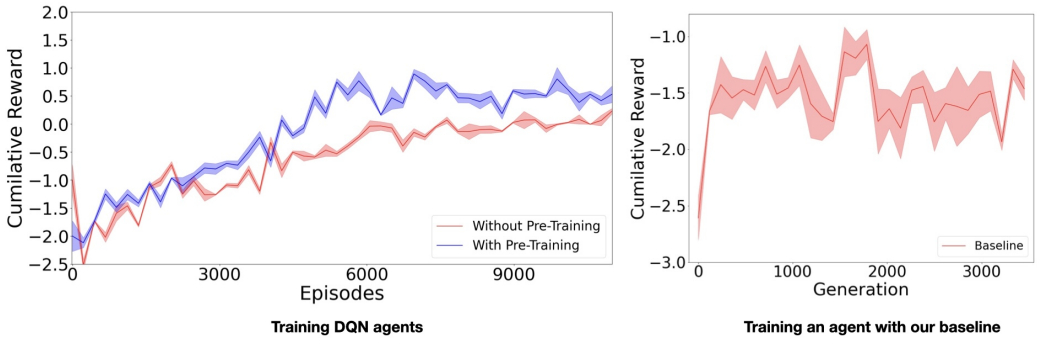


Figure 4.7: The left plot depicts the pre-trained agent and the agent just trained in the real environment. The right plot shows the average reward per generation on Pong during training with our baseline.

	RE	Entity	Frame Embedding	Baseline	Random
Score	+1.89	-4.23	-6.15	-7.4	-10.4

Table 4.4: Average game score of game playing agents over 100 game rounds in the real environment. 'RE' indicates DQN learned in the real environment.

Pong.

The DQN network's first hidden layer convolves 16 8×8 filters with stride 4. The second hidden layer convolves 32 4×4 filters with stride 2, both hidden layers are followed by a rectifier non-linearity layer. Finally, the last hidden layer is a fully-connected layer with 256 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions in Pong is 6. To train the agent, we use the ϵ-greedy behaviour policy. The 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 4.6, shows the visualization of the average game score per episode during training of the agent in both the real and simulated environment of our **Entity** and **Frame Embedding** representations, respectively. We train 3 agents with different seeds in all experiments. 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

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

Table 4.5: Average game score for our game playing agents over 100 game rounds in the real environment. 'RE' refers to the agent which trained in the real environment with ϵ starts from 0.5. The pre-trained agent, starts learning in the real environment with the same initial value of ϵ.

real environment. 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 led us to try to pre-train the agent in the simulated environment, then transfer the learned policy to the real environment and train the agent. Our goal was to see whether this would help us achieve a better score with fewer interactions in the real environment.

We choose the best **Entity** agent and transfer its learned policy to the real environment. We chose the **Entity** agent, since that agent outperforms the **Frame Embedding** agent. We then trained the agent in the real environment with ϵ starting from 0.5 and decreases 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 the ϵ to explore if we can train a DQN agent with less exploration that outperforms agents without pre-training. In Figure 4.7, the left plot shows the average score of this agent during training. As is shown the reward reaches to +0.5 after around 5000 episodes. We also train the same agent in the real environment without pre-training it, as is displayed, the pre-trained agent has higher scores from the initial steps.

The right plot in this Figure 4.7, displays the cumulative reward of our baseline during training. We train the agent in over 3000 generations. Since our baseline, 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 action at at each time step.

$$a_t = W_c[z_t h_t] + b_c \tag{4.3}$$

Where a_t is the action at timestep t, W_c and b_c are the weights and the bias that controller tries to learn, z_t and h_t are the latent space and the output of

the LSTM at timestep t. To optimize the parameters of the controller, the authors employed Covariance Matrix Adaptation Evolution Strategy(CMA-ES) [12]. Since their predictive model is not able to predict appropriate future states, the controller converges to -1.5 after only few generations.

We tested all the agents before the pre-training experiment in the real environment for 100 game rounds, each with a length of 500 frames. As is shown in table 4.4, the real environment (RE) agent outperforms all other agents. The Entity agent achieves an average score of -4.23 which is the second best agent compared to our **Frame Embedding** agent, random, and our baseline. Further, we include the two agents from the pre-training experiment in table 4.5. It is shown that, the pre-trained agent outperforms the other agent that only trained in the real environment and the RE agent from the first set of experiments. This indicates it has learned a more general final policy.

Chapter 5

Conclusion

5.1 Summary of Contributions

We started this thesis claiming that despite the huge success of modern RL algorithms, training them in the real world could be often be too time consuming, dangerous or expensive. To tackle this problem, many state-of-the-art approaches have been presented in which the agent learns a model of the environment alongside learning the optimal policy. However, many of these approaches still require a large amount of data from the environment.

In this thesis, we introduced a new representation for game frames in the Atari environment. This representation uses mechanical information of each entity at each frame, such as the position and size of each entity. We also explore how the entities are distributed in a lower dimensional space by training a VAE. We then use the proposed representations as basis to train different predictive models. These models predict future frames and the reward value based on agent actions and previous frames. We showed qualitatively and quantitatively that they are able to predict accurate rewards, future frames, and useful-for-control frames on Pong from Atari. To our knowledge, this is the first work that uses this kind of representation. This representation allows us to predict the joint probability of the next frame and the reward value with significantly less training data. We show that pre-training a DQN agent with our simulated environment can improve the agent's performance. We also demonstrated that our approach outperforms an existing state-of-the-art method [10] using significantly less training data.

5.2 Discussion and Future Works

This thesis opens up new interesting research directions. We discuss some of these directions in this section.

5.2.1 Other Game Domains

We only focused on the domain of Pong from Atari due to its simplicity and matching to the current version of our entity extraction algorithm, however for future work we can apply our approach to other domains to increase its generality. We anticipate that by including other mechanical information based on different games characteristics such as moving camera we may be able to extend this approach to domains where the camera moves.

5.2.2 Training Agents

In this work, we converted the output of our predictive model to a pixel-based image to train the DQN agent. We hope to train a DQN agent using the entity-based representation which has a simpler structure than pixel-based image input. Our simple structures could be helpful in terms of learning faster.

5.2.3 Using this predictive model in other RL Methods

Another possible area of future work is that we can employ our approach in different model-free and model-based algorithms to explore the usefulness of this representation in various RL algorithms.

References

[1] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks," 2013.

[2] P. Behboudian, Y. Satsangi, M. E. Taylor, A. Harutyunyan, and M. Bowling, "Useful policy invariant shaping from arbitrary advice," *arXiv preprint arXiv:2011.01297*, 2020.

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

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

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[6] Y. Goldberg, "Neural network methods for natural language processing," *Synthesis lectures on human language technologies*, vol. 10, no. 1, pp. 1–309, 2017.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[8] M. Guzdial, B. Li, and M. O. Riedl, "Game engine learning from video.," in *IJCAI*, 2017, pp. 3707–3713.

[9] M. Guzdial, J. Reno, J. Chen, G. Smith, and M. Riedl, "Explainable pcgml via game design patterns," *arXiv preprint arXiv:1809.09419*, 2018.

[10] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[11] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[12] N. Hansen, "The cma evolution strategy: A comparing review," *Towards a new evolutionary computation*, pp. 75–102, 2006.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] W. Jiang, L. Ma, Y.-G. Jiang, W. Liu, and T. Zhang, "Recurrent fusion network for image captioning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 499–515.

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

[16] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2016, pp. 1–8.

[17] N. Y. Khameneh and M. Guzdial, "Entity embedding as game representation," *arXiv preprint arXiv:2010.01685*, 2020.

[18] S. W. Kim, J. Philion, A. Torralba, and S. Fidler, "Drivegan: Towards a controllable high-quality neural simulation," *arXiv preprint arXiv:2104.15060*, 2021.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

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

[21] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

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

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

[24] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[25] J. Mandziuk, *Knowledge-free and learning-based methods in intelligent game playing*. Springer, 2010, vol. 276.

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

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[29] M. Oliu, J. Selva, and S. Escalera, "Folded recurrent neural networks for future video prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 716–731.

[30] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[32] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 8614–8618.

[33] J. Schmidhuber and R. Huber, "Learning to generate artificial fovea trajectories for target detection," *International Journal of Neural Systems*, vol. 2, no. 01n02, pp. 125–134, 1991.

[34] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[35] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *Icdar*, Citeseer, vol. 3, 2003.

[36] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International Conference on Machine Learning*, PMLR, 2016, pp. 1747–1756.

[37] E. Wang, A. Kosson, and T. Mu, "Deep action conditional neural network for frame prediction in atari games," Technical Report, Stanford University, Tech. Rep., 2017.

[38] Wikipedia contributors, *Table tennis — Wikipedia, the free encyclopedia*, [Online; accessed 8-June-2021], 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Table_tennis&oldid=1023261816`.

[39] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.