World Models

Gizem Aydın, Mick van Hulst, Lora Novaković & Jelle Piepenbrock

Abstract

In this work, we implemented World Models in order to train reinforcement learning agents in the simulated Neurosmash environment. We used background subtraction to enhance the performance of the variational autoencoder component, which allowed the recurrent world model to learn to predict plausible future states. We used a Deep Q Network as the controller component of the World Model setup. We found that both the standard DQN and the DQN with access to the World Model achieved average win rates of over 70%. However, we constructed several baselines, that had no access to any game-state information, and these performed similarly. These findings serve as an indication that the environment is not complex in its current form.

1 Introduction

In the original World Models paper written by Ha et al. [4], it is explained that humans develop a mental model based on what they are able to perceive with their limited senses. The decisions and actions that are made are based on this internal model. The paper goes on to divide such a model into three components, which can be divided in Vision, Memory and Controller components.

As our first contribution we explored the notion of this internal model by using it in a different environment. In previous work, it can be observed that the environments that were used are quite static in the sense that objects that are of importance, such as the car (i.e. in the Car Racing environment), do not move from their place while driving. As such, we were keen to observe if the Variational Autoencoder (VAE) was able to encode the objects of importance (i.e. the players) as they were non-static. For our second contribution, we explored the usage of a different type of controller, namely the Deep Q network (DQN) in a new environment.

2 Methods

The World Models paper describes three components that are essential to their work. These components are described as Vision (V), Memory (M), and Controller (C), which are embodied by the VAE (VAE), Recurrent Neural Net (RNN) and Covariance Matrix Adaptation Evolutionairy Strategies (CMA-ES) respectively. Due to computational limitations we replaced CMA-ES with a Deep Q Network (DQN) as our controller, after also briefly experimenting with Policy Gradient.

2.1 Vision - Variational Autoencoder

For each timestep t, the agent receives a large 2D matrix. Processing such a 2D matrix increases the complexity required for the agent to learn the environment correctly. To ease the burden of the agent, the Vision component is introduced, which maps the high-dimensional input of the environment to a low-dimensional space. As such, the role of the VAE [5] is to create a compressed representation of each input frame into the latent vector z [4]. The expectation is that this compressed representation helps the controller part of the system distinguish what states are beneficial to the agent and which are not.

Neural Information Processing Systems (2020), Nijmegen, The Netherlands.

2.2 Memory - Mixture Density Recurrent Neural Network

Based on the compressed state representation obtained from the VAE, we can train a RNN that predicts the future states. This network, the Mixture Density Recurrent Neural Network (MDN-RNN), predicts the future state of the world, given the current state and an action that influences the environment. The model parametrizes a distribution over the latent variables: $P(z_{t+1}|z_t, a_t)$. The environment contains random components, which we can handle by letting the RNN parametrize a mixture model [2]. Each mixture component conceptually represents a possible future state path. We will have a mixture of Gaussian distributions, so that the following holds:

$$p(z_{t+1}|z_t, a_t) = \sum_{k=1}^{K} \pi_k(z_t, a_t)\mathcal{N}(z_t|\mu_k(z_t, a_t), \sigma_k^2(z_t, a_t))$$

In practice, we will have a standard LSTM recurrent neural network with three densely connected neural networks that take the hidden state vector of this recurrent neural network and that predicts the mixture coefficients π_k, the means μ_k and the standard deviation σ_k for each Gaussian component of the mixture distribution (of which there are K in total). The π components have to sum to 1 for the mixture to qualify as a probability distribution, which is enforced through a softmax. The σ components have to be positive, which is enforced through an exponential transformation. The assumption is that the Gaussian distributions are isotropic, so we do not have to predict a full covariance matrix for each component distribution.

2.3 Controller - Deep Q Network

In the original work, CMA-ES is used, however, we found that this required a large amount of computational resources (as the results are most-optimal when ran in parallel fashion). As such, we deviated from the original setup and used a DQN as our controller. We briefly experimented with Policy Gradient, but it compared unfavourably to DQN.

The DQN, originally introduced by Mnih et al. [7] is a reinforcement learning network that was tested using the Atari environment. The interactions of the agent with an environment ϵ follow the notion of a Markov Decision Process, which in our setting resulted in transition tuples with values (s, a, r, s'), with a being the action that caused the transition of state s in state s' with resulting reward r. These transitions are stored in a replay buffer. During training, this allows for Experience Replay, which was introduced by Lin [6] and is characterised as randomly sampling a batch of transitions and updating the trainable network accordingly. The idea of the replay buffer is that, by default, sequences of transitions are highly correlated, resulting in less efficient optimisation of the DQN. By sampling from a variety of episodes, these sequences are decorrelated and learning becomes more optimal.

The goal of the agent is to select actions with the goal of maximising expected future reward. Future rewards are discounted by a factor of γ, and as such, the expected reward at a timestep t is defined as: $\sum_{t'=t}^{T} \gamma^{t'-t} r'_t$, where T is the time-step at which the game ends. The optimal action-value function is defined as $Q^*(s, a)$. This function maximises the expected return achievable by following any strategy and it obeys the identity known as the *Bellmann equation*. Maximising this expected return for sequences s' and all possible actions a' is formally defined as [7]:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \epsilon}[r + \gamma \max_{a'} Q^*(s', a')|s, a]$$

3 Experimental Setup

This section elaborates on the experimental setup of our work. For technical details, such as hyperparameter usage, reproducing our experiments or inspecting our code, we refer to our Github repository [1]. A general notion here is that we did not structurally perform hyperparameter optimization. As such, different hyperparameter usage might increase our final results. The reason for excluding this process was the lack of computational resources.

[1] https://github.com/JellePiepenbrock/neurosmash

3.1 Environment

The environment that was used during our experiments is a player versus computer game, where both agents are spawned in an arena and are tasked to push each other from the edges of this arena. If the players bump in to one another hard enough, the agents fall over and are unable to move for a set amount of time.

Our agent is tasked to push or lead the opponent from the edges of the arena. It does this by moving left, right or doing nothing. By default, the agent moves forward and is thus unable to move backwards. The opponent is programmed with a simple algorithm that follows the player around, with some noise in its movements. As this task is different from pushing the player off the edge, our player thus has to 'lead' the opponent player of the edge by moving around and moving near the edges such that the opposing player falls off.

Our agent receives a reward of ten if the opponent falls off the edge. This means that only the frame that leads to a win receives a reward, all other states such as simply running around or losing by falling of the edge, result in a reward of zero.

In the following sections various screenshots of the game will be shown, which illustrate the dynamics of the game. The agent is tasked to control the red player and our opponent is the blue player.

3.2 Variational Autoencoder

When reading the original World Models paper we noticed that the objects that were of importance were rather static. An example is the car in the car racing environment that does not move from its position when racing. We hypothesized that due to our environment being rather different, i.e. as the agents are non-static, the VAE would have difficulties generalizing to this new environment. We are reminded by a paper written by Van de Laar et al. [8]. In this paper, various forms of attention are mentioned and one that caught our attention is 'overt movement', which is one of the main forms of attention for us humans. This form of attention is based on reacting to moving objects. As such, if we were to observe this environment, then the agents would immediately catch our attention as they are the only moving objects.

We are keen to find if the VAE observes the environment in the same way as us humans would observe it. We setup two experiments, the *first* being the usage of the vanilla VAE without any adjustments. The *second* being the usage of a weighted loss function for the VAE, where the weights are based on a background subtraction method originating from the OpenCV package [1]. Essentially, the background subtraction model is a computer vision algorithm that attempts to distinguish background from foreground objects. This model is utilized to create weights to penalize the VAE in a heavier fashion when missclassifying pixels that are considered foreground. The resulting weights for the loss function distinguish moving objects from static objects and thus act as a form of attention.

3.3 Mixture Density Recurrent Neural Network

We performed two experiments with respect to the MDN-RNN. The first experiment being the case where the input was encoded by the vanilla VAE. The second experiment being the case where the input was encoded by the VAE that was enhanced with the background filter. In order to generate the input for the World-Model style controller, we concatenated the VAE state representation with a possible future per possible action. This means that as our total number of possible actions was three, three different possible futures were concatenated to the VAE's state representation.

3.4 Deep Q network

For our final experiments with the environment we wanted to make various comparisons. First of, we were curious to find out what the actual baseline of this environment was. As such, our first experiment consisted of giving the controller a constant input without the use of the RNN. This allowed us to test what the maximal performance of the agent would be given that the agent received no information about the state of the environment. Second, we were curious to measure differences

between the inclusion and exclusion of the RNN hidden state given that the VAE received a constant input (black screen). This would allow the agent to utilize some notion of time (through the RNN hidden state) to then perform the most suitable action given a timestep. Third, we wanted to compare the influence of the World Models environment to a traditional DQN environment. Our second experiment thus entailed the training of a vanilla DQN in the given environment. For our final experiment we trained the World Models approach using our weighted vision and its corresponding memory component.

During our experiments, success was measured by computing a rolling average win probability. After each episode, we evaluated if the episode resulted in a win, loss or draw. Draws were excluded from our average. The reason why draws were excluded was due to the sometimes buggy nature of the environment. At times, the opponent would be stuck outside of the arena. For each experiment, we trained the model for a total of 2000 episodes.

4 Results

First, we will describe results pertaining to only one component of the three-part system (VAE and RNN) and then we will move on to results involving the entire setup.

4.1 Variational Autoencoder (VAE)

As described in our experimental setup section, we performed two experiments for our vision component. Figure 1 shows an example of a first encoded and then decoded image of the vanilla VAE. We observe that the VAE completely ignores non-static objects and fails to encode the agent and its opponent. Figure 2 shows an example of a first encoded and then decoded image of the VAE with its respective weights that were based on background subtraction.

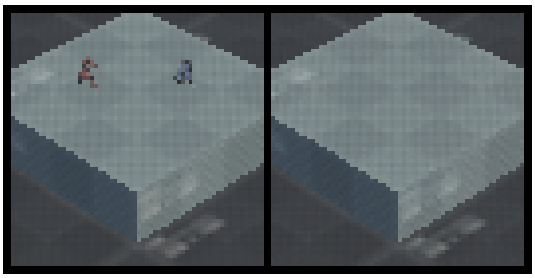

Figure 1: Example vanilla VAE. First image is the original image. Second image is the same image that is first encoded and then decoded.

Figure 2: Example weighted VAE. First image is the original image. Second image is the same image that is first encoded and then decoded. Third image being its respective filter that was used.

4.2 Memory - RNN

For the first experiment of the MDN-RNN, we quickly concluded that given an image such as depicted in the right image of Figure 1, the output was identical to the input image. In Figure 3, the

dynamics of the mixed density RNN trained on the outputs of the weighted variational autoencoder are shown. Clearly, the world model has learned to output plausible future states when given a state and an action.

As it was clear that the vanilla VAE with its respective RNN does not embed nor predict any any usable information and future states, we discarded this method and solely performed our final experiments with the weighted VAE and its respective RNN.

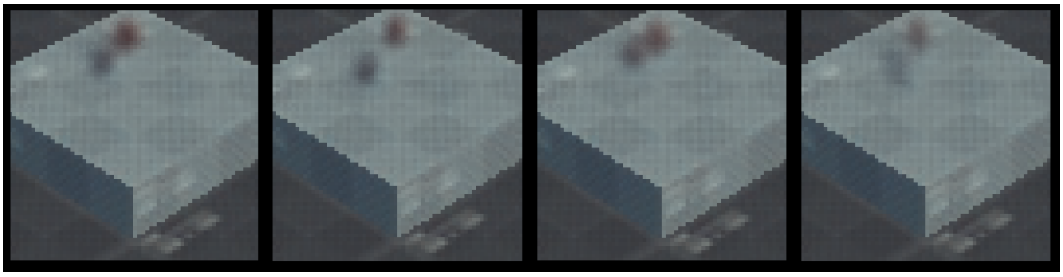

Figure 3: Example dynamics of the Mixture Density RNN when trained on the output of the weighted variational autoencoder. The leftmost image is the ‘real‘ image, the three images to the right are three of the RNNs possible futures (each sampled from a different Gaussian component) given an action and the state of the leftmost image.

4.3 Controller - DQN

We ran four experimental setups for the final evaluation. Two of them are no-visual baselines, one of them is vanilla DQN, while the last one is the full World Model setup.

In Figure 4, the performance of one of our baselines is shown, where the system receives a black screen as input to the VAE (the VAE then encodes this black screen). The input does not change during the episode, so the system can only learn what is the best possible move, regardless of the state of the game. Upon inspection, the baseline agent is only turning to the right. In Figure 5, another baseline is shown, where the agent gets a black screen as input to the VAE, but can still access RNN hidden states. The controller might learn some notion of time through the dynamics of the hidden state with this information.

Figure 6 and Figure 7 shows the performance of the vanilla DQN and the DQN as the controller of the World Model respectively. In Table 1, the final average win rate for each system can be observed. All of the systems perform similarly and the baselines seem to be hard to improve upon.

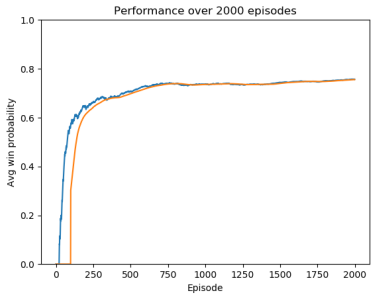

Figure 4: Average win probability of the baseline, which gets a black screen as input. Orange line is a rolling average over the last 100 episodes.

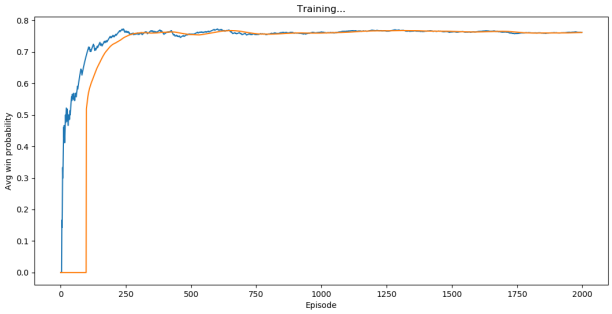

Figure 5: Average win probability of the baseline, which gets a black screen as input, but is able to use the RNN hidden state. Orange line is a rolling average over last 100 episodes.

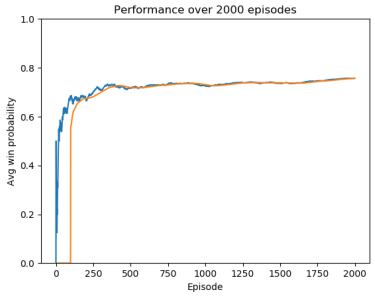

Figure 6: Average win probability of the vanilla DQN. Orange line is a rolling average over last 100 episodes.

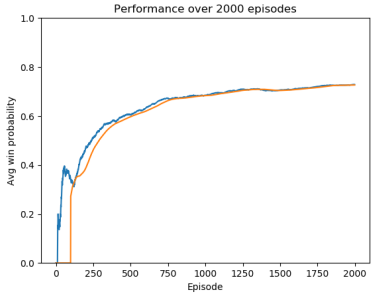

Figure 7: Average win probability of the DQN as a controller of the World Models architecture. Orange line is a rolling average over last 100 episodes.

Experiment	**Final Win Rate**
No input, no RNN	0.76
No input, RNN state	0.78
Vanilla DQN	0.76
Full World Model	0.73

Table 1: Final win averages after 2000 episodes.

5 Discussion & Future Work

In this section we reflect upon the contributions made during this project. First of, we conclude that the VAE in this particular environment is, by default, not able to encode objects that are most important (i.e. the players). We suspect that this is due to the players being moving objects, which makes it easy for the VAE to ignore and thus to focus on the remainder of the scene. By using a weighted loss function, we mimic an attentional-like mechanism, resulting in the encoding of the players. The results of these experiments expose a shortcoming of the VAE, namely that it is not able to differentiate between what is of importance in a scene. As such, it seems logical to explore this further in future work to experiment with the various properties of the VAE such that it can more closely resemble attention-like mechanisms.

Regarding the experimentation with a DQN as our controller, we found that all approaches perform similarly. As the baselines performed well, we did not expect that there was much more improvement to be gained by the more involved methods. This was unsatisfactory, as we did want to observe whether there was any gain from the World Models approach. Limited computational resources prevented us from rerunnning the algorithms to obtain stable estimates of the final performances, but we also did not judge this to be of very high priority, as the baseline seemed to be just as good as the advanced approaches. Anecdotally, the learning trajectory for the algorithms were always quite stable, and we did not experience any instance where learning did not occur with the DQN as the controller. This, in comparison to Policy Gradient, which caused us great difficulty. In general, the agents seem to learn to rotate in one direction and ignore most of the game state information. The noise in the opponent's movements caused the opponent to walk off the platform, making the game a win for our agent. We did not observe a more complex strategy, though it is hard to judge what the agent strategy is from observing its movement. We hypothesize that one of the reasons for this singular winning strategy, is the current physics system. Players are not able to actually push each other from the edge as bumping into each other results in both agents falling over, resulting in temporary immobility. As such, the player can only attempt to run close to the edge up until the opponent falls off. This is, obviously, a very tricky strategy as a single incorrect action causes the player to fall off itself.

During our project we had various ideas that we would like to explore in future work:

1. Train in dream environment as this would allow for faster training (sampling is more efficient due to reduced dimensionality VAE).
2. Explore Deep Recurrent Attentive Writer (DRAW) [3], which is a recurrent attentional model that uses a VAE for image generation. We are keen to explore if this approach can be integrated in World Models.
3. Alter the environment in various ways. For example, expand the environment for various types of attention. As bumping into a character leads immobility, this could be added as a negative reward, such that the agent learns to evade the opponent while trying to draw them towards the edges of the arena.
4. In general, increase the complexity of the environment such that the usage of the World Models approach can be justified.

References

[1] How to use background subtraction methods. url: `https://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html`.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[4] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[6] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] Pierre Van De Laar, Tom Heskes, and Stan Gielen. Task-dependent learning of attention. *Neural networks*, 10(6):981–992, 1997.