World Models

David Ha [1] **Jürgen Schmidhuber** [2] [3]

Abstract

We explore building generative neural network models of popular reinforcement learning environments. Our *world model* can be trained quickly in an unsupervised manner to learn a compressed spatial and temporal representation of the environment. By using features extracted from the world model as inputs to an agent, we can train a very compact and simple policy that can solve the required task. We can even train our agent entirely inside of its own hallucinated dream generated by its world model, and transfer this policy back into the actual environment.

An interactive version of this paper is available at https://worldmodels.github.io

1. Introduction

Humans develop a mental model of the world based on what they are able to perceive with their limited senses. The decisions and actions we make are based on this internal model. Jay Wright Forrester, the father of system dynamics, defined a mental model as:

The image of the world around us, which we carry in our head, is just a model. Nobody in his head imagines all the world, government or country. He has only selected concepts, and relationships between them, and uses those to represent the real system. (Forrester, 1971)

To handle the vast amount of information that flows through our daily lives, our brain learns an abstract representation of both spatial and temporal aspects of this information. We are able to observe a scene and remember an abstract description thereof (Cheang & Tsao, 2017; Quiroga et al., 2005). Evidence also suggests that what we perceive at any given moment is governed by our brains prediction of the future based on our internal model (Nortmann et al., 2015; Gerrit et al., 2013).

One way of understanding the predictive model in our brains is that it might not be about just predicting the future in general, but predicting future sensory data given current
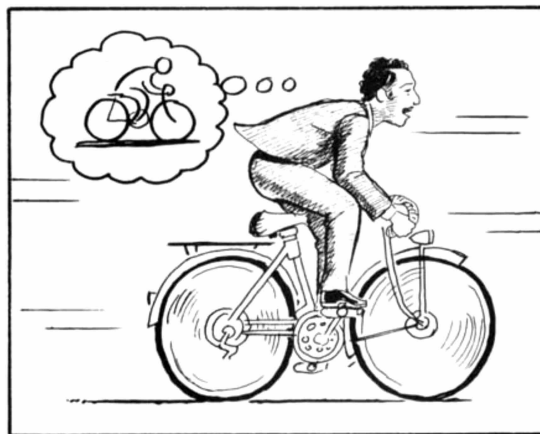
Figure 1. A World Model, from Scott McCloud's *Understanding Comics.* (McCloud, 1993; E, 2012)

motor actions (Keller et al., 2012; Leinweber et al., 2017). We are able to instinctively act on this predictive model and perform fast reflexive behaviours when we face danger (Mobbs et al., 2015), without the need to consciously plan out a course of action.

Take baseball for example. A batter has milliseconds to decide how they should swing the bat – shorter than the time it takes for visual signals to reach our brain. The reason they are able to hit a 115 mph fastball is due to our ability to instinctively predict when and where the ball will go. For professional players, this all happens subconsciously. Their muscles reflexively swing the bat at the right time and location in line with their internal models' prediction (Gerrit et al., 2013). They can quickly act on their predictions of the future without the need to consciously roll out possible future scenarios to form a plan (Hirshon, 2013).
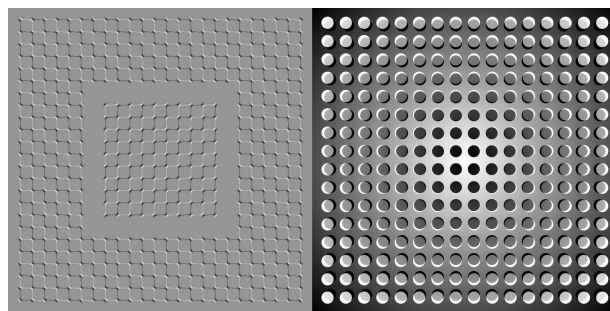

Figure 2. What we see is based on our brain's prediction of the future (Kitaoka, 2002; Watanabe et al., 2018).

In many reinforcement learning (RL) (Kaelbling et al., 1996; Sutton & Barto, 1998; Wiering & van Otterlo, 2012) problems, an artificial agent also benefits from having a good representation of past and present states, and a good predictive model of the future (Werbos, 1987; Silver, 2017), preferably a powerful predictive model implemented on a general purpose computer such as a recurrent neural network (RNN) (Schmidhuber, 1990a;b; 1991a).

Figure 3. In this work, we build probabilistic generative models of OpenAI Gym environments. The RNN-based world models are trained using collected observations recorded from the actual game environment. After training the world models, we can use them mimic the complete environment and train agents using them.

Large RNNs are highly expressive models that can learn rich spatial and temporal representations of data. However, many *model-free* RL methods in the literature often only use small neural networks with few parameters. The RL algorithm is often bottlenecked by the credit assignment problem, which makes it hard for traditional RL algorithms to learn millions of weights of a large model, hence in practice, smaller networks are used as they iterate faster to a good policy during training.

Ideally, we would like to be able to efficiently train large RNN-based agents. The backpropagation algorithm (Linnainmaa, 1970; Kelley, 1960; Werbos, 1982) can be used to train large neural networks efficiently. In this work we look at training a large neural network[1] to tackle RL tasks, by dividing the agent into a large world model and a small controller model. We first train a large neural network to learn a model of the agent's world in an unsupervised manner, and then train the smaller controller model to learn to perform a task using this world model. A small controller lets the training algorithm focus on the credit assignment problem on a small search space, while not sacrificing capacity and expressiveness via the larger world model. By training the agent through the lens of its world model, we show that it

---

[1]Typical model-free RL models have in the order of 10^3 to 10^6 model parameters. We look at training models in the order of 10^7 parameters, which is still rather small compared to state-of-the-art deep learning models with 10^8 to even 10^9 parameters. In principle, the procedure described in this article can take advantage of these larger networks if we wanted to use them.

can learn a highly compact policy to perform its task.

Although there is a large body of research relating to *model-based* reinforcement learning, this article is not meant to be a review (Arulkumaran et al., 2017; Schmidhuber, 2015b) of the current state of the field. Instead, the goal of this article is to distill several key concepts from a series of papers 1990–2015 on combinations of RNN-based world models and controllers (Schmidhuber, 1990a;b; 1991a; 1990c; 2015a). We will also discuss other related works in the literature that share similar ideas of learning a world model and training an agent using this model.

In this article, we present a simplified framework that we can use to experimentally demonstrate some of the key concepts from these papers, and also suggest further insights to effectively apply these ideas to various RL environments. We use similar terminology and notation as *On Learning to Think: Algorithmic Information Theory for Novel Combinations of RL Controllers and RNN World Models* (Schmidhuber, 2015a) when describing our methodology and experiments.

2. Agent Model

We present a simple model inspired by our own cognitive system. In this model, our agent has a visual sensory component that compresses what it sees into a small representative code. It also has a memory component that makes predictions about future codes based on historical information. Finally, our agent has a decision-making component that decides what actions to take based only on the representations created by its vision and memory components.
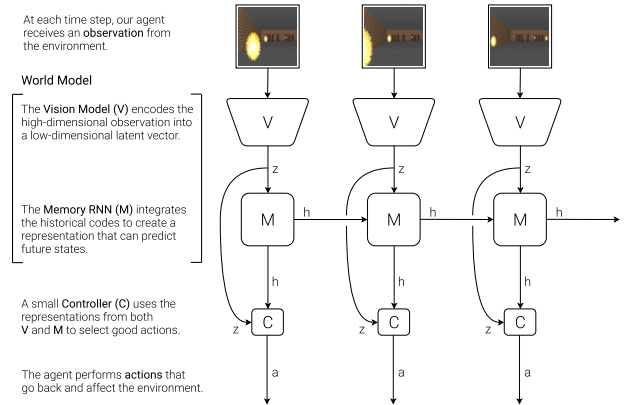

Figure 4. Our agent consists of three components that work closely together: **Vision (V)**, **Memory (M)**, and **Controller (C)**

2.1. VAE (V) Model

The environment provides our agent with a high dimensional input observation at each time step. This input is usually a 2D image frame in a video sequence. The role of the V model is to learn an abstract, compressed representation of each observed input frame.
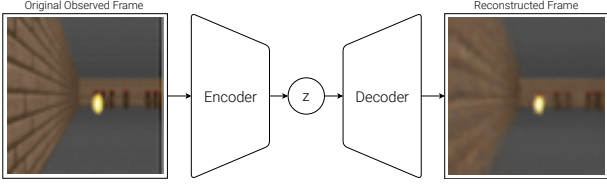
Figure 5. Flow diagram of a Variational Autoencoder.

In our experiments, we use a Variational Autoencoder (VAE) (Kingma & Welling, 2013; Jimenez Rezende et al., 2014) as the V model.

2.2. MDN-RNN (M) Model

While it is the role of the V model to compress what the agent sees at each time frame, we also want to compress what happens over time. For this purpose, the role of the M model is to predict the future. The M model serves as a predictive model of the future z vectors that V is expected to produce. Since many complex environments are stochastic in nature, we train our RNN to output a probability density function $p(z)$ instead of a deterministic prediction z.
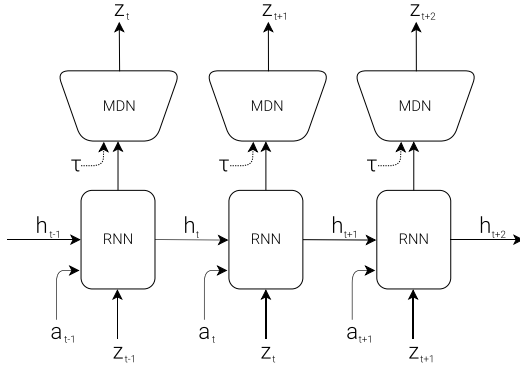

Figure 6. RNN with a Mixture Density Network output layer. The MDN outputs the parameters of a mixture of Gaussian distribution used to sample a prediction of the next latent vector z.

In our approach, we approximate $p(z)$ as a mixture of Gaussian distribution, and train the RNN to output the probability distribution of the next latent vector z_{t+1} given the current and past information made available to it.

More specifically, the RNN will model $P(z_{t+1} \mid a_t, z_t, h_t)$, where a_t is the action taken at time t and h_t is the *hidden state* of the RNN at time t. During sampling, we can adjust a *temperature* parameter τ to control model uncertainty, as done in (Ha & Eck, 2017) – we will find adjusting τ to be useful for training our controller later on.

This approach is known as a Mixture Density Network (Bishop, 1994) combined with a RNN (MDN-RNN) (Graves, 2013; Ha, 2017a), and has been used successfully in the past for sequence generation problems such as generating handwriting (Graves, 2013).
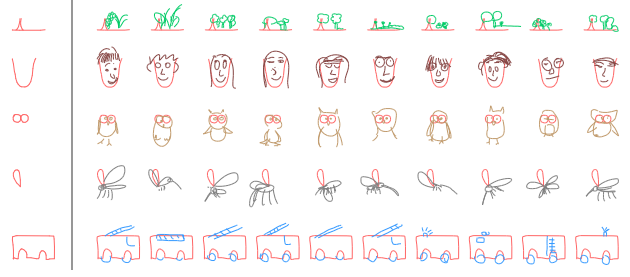

Figure 7. SketchRNN (Ha & Eck, 2017) is an example of a MDN-RNN used to predict the next pen strokes of a sketch drawing. We use a similar model to predict the next latent vector z_t.

2.3. Controller (C) Model

The Controller (C) model is responsible for determining the course of actions to take in order to maximize the expected cumulative reward of the agent during a rollout in the environment. In our experiments, we deliberately make C as simple and small as possible, and trained separately from V and M, so that most of our agent's complexity resides in the world model (V and M).

C is a simple single layer linear model that maps z_t and h_t directly to action a_t at each time step:

$$a_t = W_c \, [z_t \ h_t] \ + b_c \tag{1}$$

In this linear model, W_c and b_c are the weight matrix and bias vector that maps the concatenated input vector $[z_t \ h_t]$ to the output action vector a_t.

2.4. Putting V, M, and C Together

The following flow diagram illustrates how V, M, and C interacts with the environment:
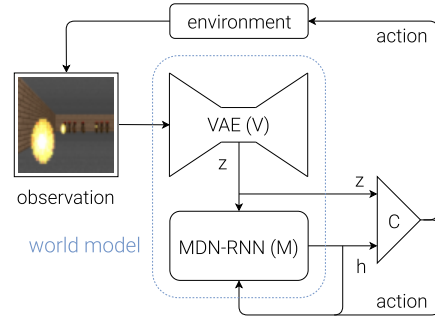

Figure 8. Flow diagram of our Agent model. The raw observation is first processed by V at each timestep t to produce z_t. The inputs into C is this latent vector z_t concatenated with M's hidden state h_t at each time step. C will then output an action vector a_t for motor control, and will affect the environment. M will then take the current z_t and action a_t as an input to update its own hidden state to produce h_{t+1} to be used at time $t + 1$.

Below is the pseudocode for how our agent model is used in the OpenAI Gym (Brockman et al., 2016) environment:

```
def rollout(controller):
  ''' env, rnn, vae are '''
  ''' global variables  '''
  obs = env.reset()
  h = rnn.initial_state()
  done = False
  cumulative_reward = 0
  while not done:
    z = vae.encode(obs)
    a = controller.action([z, h])
    obs, reward, done = env.step(a)
    cumulative_reward += reward
    h = rnn.forward([a, z, h])
  return cumulative_reward
```

Running this function on a given `controller` C will return the cumulative reward during a rollout.

This minimal design for C also offers important practical benefits. Advances in deep learning provided us with the tools to train large, sophisticated models efficiently, provided we can define a well-behaved, differentiable loss function. Our V and M models are designed to be trained efficiently with the backpropagation algorithm using modern GPU accelerators, so we would like most of the model's complexity, and model parameters to reside in V and M. The number of parameters of C, a linear model, is minimal in comparison. This choice allows us to explore more unconventional ways to train C – for example, even using evolution strategies (ES) (Rechenberg, 1973; Schwefel, 1977) to tackle more challenging RL tasks where the credit assignment problem is difficult.

To optimize the parameters of C, we chose the Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen, 2016; Hansen & Ostermeier, 2001) as our optimization algorithm since it is known to work well for solution spaces of up to a few thousand parameters. We evolved parameters of C on a single machine with multiple CPU cores running multiple rollouts of the environment in parallel.

For more specific information about the models, training procedures, and environments used in our experiments, please refer to the Appendix section.

3. Car Racing Experiment

In this section, we describe how we can train the Agent model described earlier to solve this car racing task. To our knowledge, our agent is the first known solution to achieve the required score to solve this task.[2]

---

[2] We find this task interesting because although it is not difficult to train an agent to wobble around randomly generated tracks

3.1. World Model for Feature Extraction

A predictive world model can help us extract useful representations of space and time. By using these features as inputs of a controller, we can train a compact and minimal controller to perform a continuous control task, such as learning to drive from pixel inputs for a top-down car racing environment called `CarRacing-v0` (Klimov, 2016).

Figure 9. Our agent learning to navigate in `CarRacing-v0`.

In this environment, the tracks are randomly generated for each trial, and our agent is rewarded for visiting as many tiles as possible in the least amount of time. The agent controls three continuous actions: steering left/right, acceleration, and brake.

To train our V model, we first collect a dataset of 10,000 random rollouts in the environment. We have first an agent acting randomly to explore the environment multiple times, and record the random actions a_t taken and the resulting observations from the environment. We use this dataset to train V to learn a latent space of each frame observed. We train our VAE to encode each frame into low dimensional latent vector z by minimizing the difference between a given frame and the reconstructed version of the frame produced by the decoder from z.

We can now use our trained V model to pre-process each frame at time t into z_t to train our M model. Using this pre-processed data, along with the recorded random actions a_t taken, our MDN-RNN can now be trained to model $P(z_{t+1} \mid a_t, z_t, h_t)$ as a mixture of Gaussians.[3]

---

and obtain a mediocre score, `CarRacing-v0` defines *solving* as getting average reward of 900 over 100 consecutive trials, which means the agent can only afford very few driving mistakes.

[3] In principle, we can train both models together in an end-to-end manner, although we found that training each separately is more practical, and also achieves satisfactory results. Training each model only required less than an hour of computation time on a single GPU. We can also train individual VAE and MDN-RNN models without having to exhaustively tune hyperparameters.

In this experiment, the world model (V and M) has no knowledge about the actual reward signals in the environment. Its task is simply to compress and predict the sequence of image frames observed. Only the Controller (C) Model has access to the reward information from the environment. Since the there are a mere 867 parameters inside the linear controller model, evolutionary algorithms such as CMA-ES are well suited for this optimization task.

We can use the VAE to reconstruct each frame using z_t at each time step to visualize the quality of the information the agent actually sees during a rollout. The figure below is a VAE model trained on screenshots from `CarRacing-v0`.
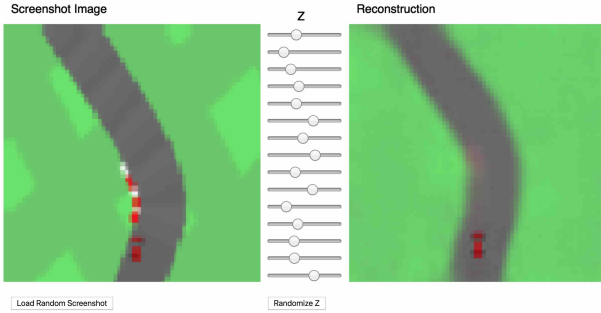

Screenshot Image   z   Reconstruction

Load Random Screenshot   Randomize Z

Figure 10. Despite losing details during this lossy compression process, latent vector z captures the essence of each image frame.

In the online version of this article, one can load randomly chosen screenshots to be encoded into a small latent vector z, which is used to reconstruct the original screenshot. One can also experiment with adjusting the values of the z vector using the slider bars to see how it affects the reconstruction, or randomize z to observe the space of possible screenshots.

3.2. Procedure

To summarize the Car Racing experiment, below are the steps taken:

1. Collect 10,000 rollouts from a random policy.

2. Train VAE (V) to encode frames to $z \in \mathcal{R}^{32}$.

3. Train MDN-RNN (M) to model $P(z_{t+1} \mid a_t, z_t, h_t)$.

4. Define Controller (C) as $a_t = W_c\,[z_t\ h_t]\ +\ b_c$.

5. Use CMA-ES to solve for a W_c and b_c that maximizes the expected cumulative reward.

MODEL	PARAMETER COUNT
VAE	4,348,547
MDN-RNN	422,368
CONTROLLER	867

3.3. Experiment Results

V Model Only

Training an agent to drive is not a difficult task if we have a good representation of the observation. Previous works (Hnermann, 2017; Bling, 2015; Lau, 2016) have shown that with a good set of hand-engineered information about the observation, such as LIDAR information, angles, positions and velocities, one can easily train a small feed-forward network to take this hand-engineered input and output a satisfactory navigation policy. For this reason, we first want to test our agent by handicapping C only to have access to V but not M, so we define our controller as $a_t = W_c\,z_t\ +\ b_c$.

Figure 11. Limiting our controller to see only z_t, but not h_t results in wobbly and unstable driving behaviours.

Although the agent is still able to navigate the race track in this setting, we notice it wobbles around and misses the tracks on sharper corners. This handicapped agent achieved an average score of 632 ± 251 over 100 random trials, in line with the performance of other agents on OpenAI Gym's leaderboard (Klimov, 2016) and traditional Deep RL methods such as A3C (Khan & Elibol, 2016; Jang et al., 2017). Adding a hidden layer to the C model's policy network improves the results to 788 ± 141, not enough to solve the environment.

Full World Model (V and M)

The representation z_t provided by our V model only captures a representation at a moment in time and doesn't have much predictive power. In contrast, M is trained to do one thing, and to do it really well, which is to predict z_{t+1}. Since M's prediction of z_{t+1} are produced from the RNN's hidden state h_t at time t, this vector is a good candidate for the set of learned features we can give to our agent. Combining z_t with h_t gives our controller C a good representation of both the current observation, and what to expect in the future.
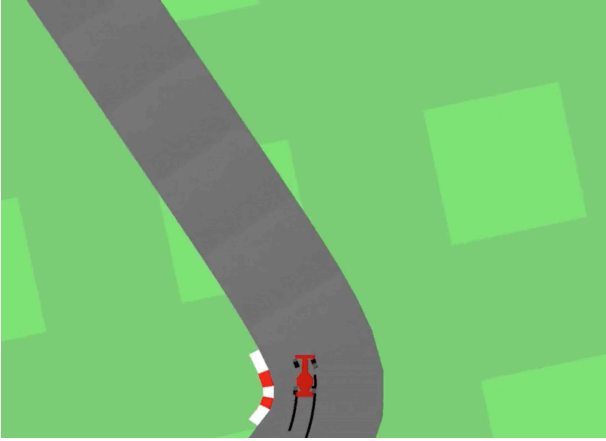
Figure 12. Driving is more stable if we give our controller access to both z_t and h_t.

We see that allowing the agent to access the both z_t and h_t greatly improves its driving capability. The driving is more stable, and the agent is able to seemingly attack the sharp corners effectively. Furthermore, we see that in making these fast reflexive driving decisions during a car race, the agent does not need to *plan ahead* and roll out hypothetical scenarios of the future. Since h_t contain information about the probability distribution of the future, the agent can just query the RNN instinctively to guide its action decisions. Like a seasoned Formula One driver or the baseball player discussed earlier, the agent can instinctively predict when and where to navigate in the heat of the moment.

METHOD	AVG. SCORE
DQN (PRIEUR, 2017)	343 ± 18
A3C (CONTINUOUS) (JANG ET AL., 2017)	591 ± 45
A3C (DISCRETE) (KHAN & ELIBOL, 2016)	652 ± 10
CEOBILLIONAIRE (GYM LEADERBOARD)	838 ± 11
V MODEL	632 ± 251
V MODEL WITH HIDDEN LAYER	788 ± 141
FULL WORLD MODEL	$\mathbf{906 \pm 21}$

Table 1. `CarRacing-v0` scores achieved using various methods.

Our agent is able to achieve a score of 906 ± 21 over 100 random trials, effectively solving the task and obtaining new state of the art results. Previous attempts (Khan & Elibol, 2016; Jang et al., 2017) using Deep RL methods obtained average scores of 591–652 range, and the best reported solution on the leaderboard obtained an average score of 838 ± 11 over 100 random trials. Traditional Deep RL methods often require pre-processing of each frame, such as employing edge-detection (Jang et al., 2017), in addition to stacking a few recent frames (Khan & Elibol, 2016; Jang et al., 2017) into the input. In contrast, our world model takes in a stream of raw RGB pixel images and directly learns a spatial-temporal representation. To our knowledge, our method is the first reported solution to solve this task.

3.4. Car Racing Dreams

Since our world model is able to model the future, we are able to have it hallucinate hypothetical car racing scenarios on its own. We can ask it to produce the probability distribution of z_{t+1} given the current states, *sample* a z_{t+1} and use this sample as the real observation. We can put our trained C back into this hallucinated environment generated by M. The following image from an interactive demo in the online version of this article shows how our world model can be used to hallucinate the car racing environment:
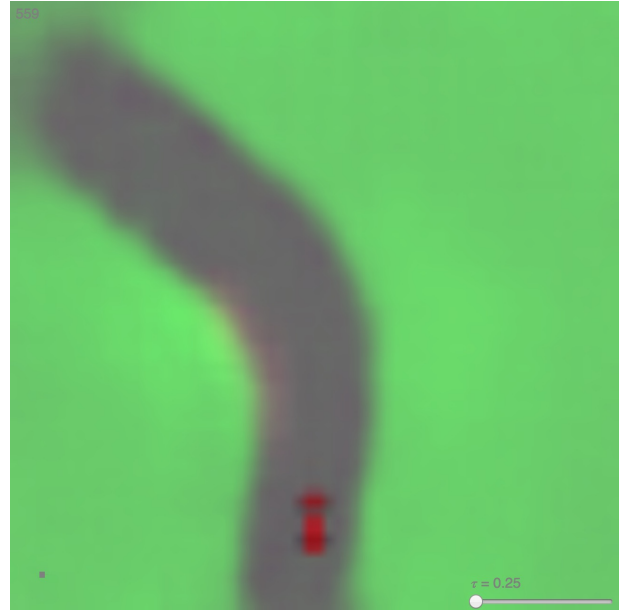

Figure 13. Our agent driving inside its own dream. Here, we deploy our trained policy into a fake environment generated by the MDN-RNN, and rendered using the VAE's decoder. In the demo, one can override the agent's actions as well as adjust τ to control the uncertainty of the environment generated by the M model.

4. VizDoom Experiment

4.1. Learning Inside of a Dream

We have just seen that a policy learned in the real environment appears to somewhat function inside of the dream environment. This begs the question – can we train our agent to learn inside of its own dream, and transfer this policy back to the actual environment?

If our world model is sufficiently accurate for its purpose, and complete enough for the problem at hand, we should be able to substitute the actual environment with this world model. After all, our agent does not directly observe the reality, but only sees what the world model lets it see. In this experiment, we train an agent inside the hallucination generated by its world model trained to mimic a Viz-Doom (Kempka et al., 2016) environment.

Figure 14. Our final agent solving *VizDoom: Take Cover*.

The agent must learn to avoid fireballs shot by monsters from the other side of the room with the sole intent of killing the agent. There are no explicit rewards in this environment, so to mimic natural selection, the cumulative reward can be defined to be the number of time steps the agent manages to stay alive in a rollout. Each rollout in the environment runs for a maximum of 2100 frames (~ 60 seconds), and the task is considered solved when the average survival time over 100 consecutive rollouts is greater than 750 frames (~ 20 seconds) (Paquette, 2016).

4.2. Procedure

Our VizDoom experiment is largely the same as the Car Racing task, except for a few key differences. In the Car Racing task, M is only trained to model the next z_t. Since we want to build a world model we can train our agent in, our M model here will also predict whether the agent dies in the next frame (as a binary event $done_t$), in addition to the next frame z_t.

Since the M model can predict the *done* state in addition to the next observation, we now have all of the ingredients needed to make a full RL environment. We first build an OpenAI Gym environment interface by wrapping a `gym.Env` interface over our M if it were a real Gym environment, and then train our agent in this virtual environment instead of using the actual environment.

In this simulation, we don't need the V model to encode any real pixel frames during the hallucination process, so our agent will therefore only train entirely in a latent space environment. This has many advantages as we will see.

This virtual environment has an identical interface to the real environment, so after the agent learns a satisfactory policy in the virtual environment, we can easily deploy this policy back into the actual environment to see how well the policy transfers over.

To summarize the *Take Cover* experiment, below are the steps taken:

1. Collect 10,000 rollouts from a random policy.

2. Train VAE (V) to encode frames to latent vector $z \in \mathcal{R}^{64}$, and use V to convert the images collected from (1) into latent space representation.

3. Train MDN-RNN (M) to model $P(z_{t+1}, done_{t+1} \mid a_t, z_t, h_t)$.

4. Define Controller (C) as $a_t = W_c \, [z_t \; h_t]$.

5. Use CMA-ES to solve for a W_c that maximizes the expected survival time inside the virtual environment.

6. Use learned policy from (5) on actual environment.

MODEL	PARAMETER COUNT
VAE	4,446,915
MDN-RNN	1,678,785
CONTROLLER	1,088

4.3. Training Inside of the Dream

After some training, our controller learns to navigate around the dream environment and escape from deadly fireballs shot by monsters generated by the M model. Our agent achieved a *score* in this virtual environment of ~ 900 frames.
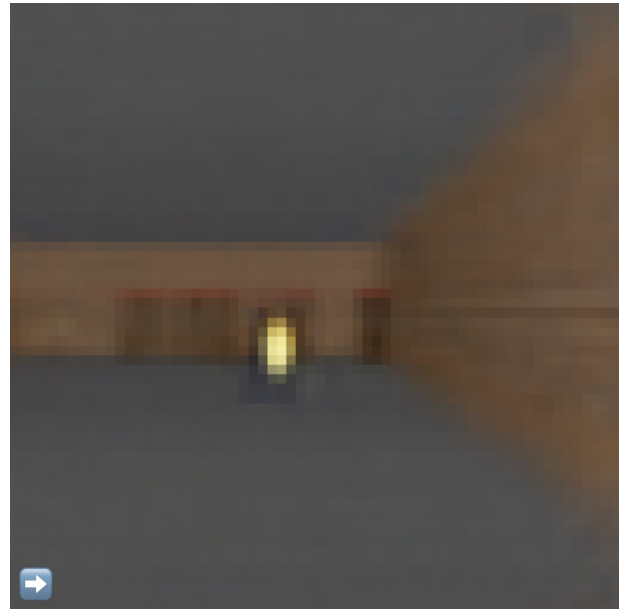

Figure 15. Our agent discovers a policy to avoid hallucinated fireballs. In the online version of this article, the reader can interact with the environment in this demo.

Here, our RNN-based world model is trained to mimic a complete game environment designed by human programmers. By learning only from raw image data collected from random episodes, it learns how to simulate the essential aspects of the game – such as the game logic, enemy behaviour, physics, and the 3D graphics rendering.

For instance, if the agent selects the left action, the M model learns to move the agent to the left and adjust its internal representation of the game states accordingly. It also learns to block the agent from moving beyond the walls on both sides of the level if the agent attempts to move too far in either direction. Occasionally, the M model needs to keep track of multiple fireballs being shot from several different monsters and coherently move them along in their intended directions. It must also detect whether the agent has been killed by one of these fireballs.

Unlike the actual game environment, however, we note that it is possible to add extra uncertainty into the virtual environment, thus making the game more challenging in the dream environment. We can do this by increasing the temperature τ parameter during the sampling process of z_{t+1}. By increasing the uncertainty, our dream environment becomes more difficult compared to the actual environment. The fireballs may move more randomly in a less predictable path compared to the actual game. Sometimes the agent may even die due to sheer misfortune, without explanation.

We find agents that perform well in higher temperature settings generally perform better in the normal setting. In fact, increasing τ helps prevent our controller from taking advantage of the imperfections of our world model – we will discuss this in more depth later on.

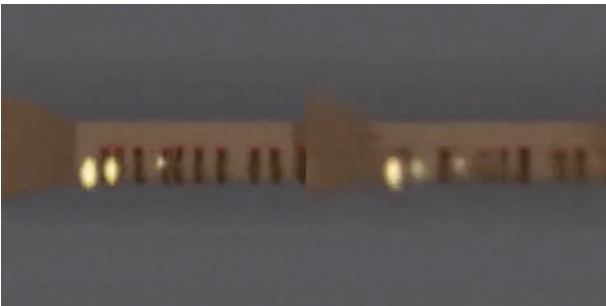### 4.4. Transfer Policy to Actual Environment

Figure 16. Deploying our policy learned in the dream RNN environment back into the actual VizDoom environment.

We took the agent trained in the virtual environment and tested its performance on the original VizDoom scenario. The score over 100 random consecutive trials is ~ 1100 frames, far beyond the required score of 750, and also much higher than the score obtained inside the more difficult virtual environment.
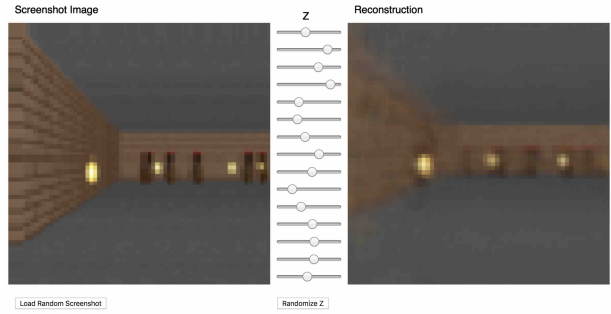

Figure 17. An interactive VAE of Doom in the online article.

We see that even though the V model is not able to capture all of the details of each frame correctly, for instance, getting the number of monsters correct, the agent is still able to use the learned policy to navigate in the real environment. The virtual environment also did not keep track of a clear number of monsters in the first place, and an agent that is able to survive the noisier and uncertain virtual nightmare environment will thrive in this clean, noiseless environment.

4.5. Cheating the World Model

In our childhood, we may have encountered ways to exploit video games in ways that were not intended by the original game designer (Wikipedia, 2017). Players discover ways to collect unlimited lives or health, and by taking advantage of these exploits, they can easily complete an otherwise difficult game. However, in the process of doing so, they may have forfeited the opportunity to learn the skill required to master the game as intended by the game designer.

For instance, in our initial experiments, we noticed that our agent discovered an *adversarial* policy to move around in such a way so that the monsters in this virtual environment governed by the M model never shoots a single fireball in some rollouts. Even when there are signs of a fireball forming, the agent will move in a way to extinguish the fireballs magically as if it has superpowers in the environment.

Because our world model is only an approximate probabilistic model of the environment, it will occasionally generate trajectories that do not follow the laws governing the actual environment. As we saw previously, even the number of monsters on the other side of the room in the actual environment is not exactly reproduced by the world model. Like a child who learns that objects in the air usually fall to the ground, the child might also imagine unrealistic superheroes who fly across the sky. For this reason, our world model will be exploitable by the controller, even if in the actual environment such exploits do not exist.

And since we are using the M model to generate a virtual dream environment for our agent, we are also giving the controller access to all of the hidden states of M. This is

essentially granting our agent access to all of the internal states and memory of the game engine of the game it is playing. Therefore our agent can efficiently explore ways to directly manipulate the hidden states of the game engine in its quest to maximize its expected cumulative reward. The weakness of this approach of learning a policy inside a learned dynamics model is that our agent can easily find an adversarial policy that can fool our dynamics model – it'll find a policy that looks good under our dynamics model, but will fail in the actual environment, usually because it visits states where the model is wrong because they are away from the training distribution.
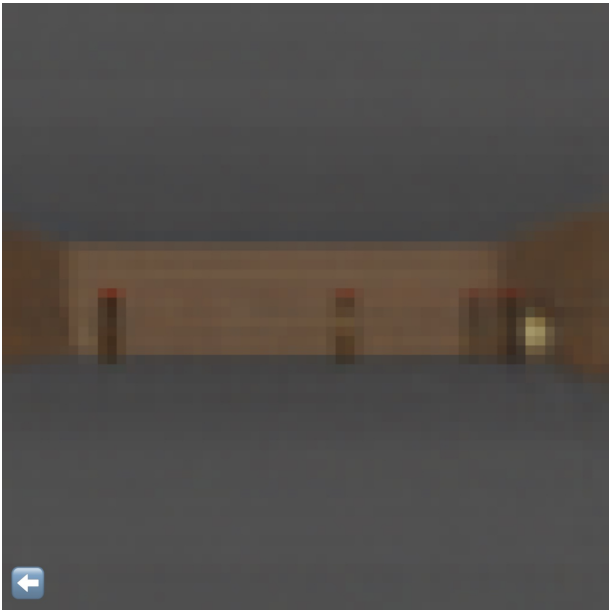

Figure 18. Agent discovers an adversarial policy to automatically extinguish fireballs after they are fired during some rollouts.

This weakness could be the reason that many previous works that learn dynamics models of RL environments but don't actually use those models to fully replace the actual environments (Oh et al., 2015; Chiappa et al., 2017). Like in the M model proposed in 1990 (Schmidhuber, 1990a;b; 1991a), the dynamics model is a deterministic model, making the model easy for the agent to exploit if it is not perfect. Using Bayesian models (as in PILCO (Deisenroth & Rasmussen, 2011)) helps to address this issue with the uncertainty estimates to some extent, however, they don't fully solve the problem. Recent work (Nagabandi et al., 2017) combines the model-based approach with traditional model-free RL training by first initializing the policy network with the learned policy, but must subsequently rely on model-free methods to fine-tune this policy in the actual environment.

In *Learning to Think* (Schmidhuber, 2015a), it is acceptable that the RNN M isn't always a reliable predictor. A (potentially evolution-based) RNN C can in principle learn to ignore a flawed M, or exploit certain useful parts of M for arbitrary computational purposes including hierarchical planning etc. This is not what we do here though – our present approach is still closer to some of the old systems, where a RNN M is used to predict and plan ahead step by step. Unlike this early work, however, we use evolution for C (like in *Learning to Think*) rather than traditional RL combined with RNNs, which has the advantage of both simplicity and generality.

To make it more difficult for our C model to exploit deficiencies in the M model, we chose to use the MDN-RNN as the dynamics model, which models the *distribution* of possible outcomes in the actual environment, rather than merely predicting a deterministic future. Even if the actual environment is deterministic, the MDN-RNN would in effect approximate it as a stochastic environment. This has the advantage of allowing us to train our C model inside a more stochastic version of any environment – we can simply adjust the temperature τ parameter to control the amount of randomness in the M model, hence controlling the tradeoff between realism and exploitability.

Using a mixture of Gaussian model may seem like overkill given that the latent space encoded with the VAE model is just a diagonal Gaussian. However, the discrete modes in a mixture density model is useful for environments with random discrete events, such as whether a monster decides to shoot a fireball or stay put. While a Gaussian might be sufficient to encode individual frames, a RNN with a mixture density output layer makes it easier to model the logic behind a more complicated environment with discrete random states.

For instance, if we set the temperature parameter to a very low value of $\tau = 0.1$, effectively training our C model with an M model that is almost identical to a deterministic LSTM, the monsters inside this dream environment fail to shoot fireballs, no matter what the agent does, due to mode collapse. The M model is not able to *jump* to another mode in the mixture of Gaussian model where fireballs are formed and shot. Whatever policy trained in this dream will get a perfect score of 2100 most of the time, but will obviously fail when unleashed into the harsh reality of the actual world, underperforming even a random policy.

Note again, however, that the simpler and more robust approach in *Learning to Think* does not insist on using M for step by step planning. Instead, C can learn to use M's *subroutines* (parts of M's weight matrix) for arbitrary computational purposes but can also learn to ignore M when M is useless and when ignoring M yields better performance. Nevertheless, at least in our present C–M variant, M's predictions are essential for teaching C, more like in some of the early C–M systems (Schmidhuber, 1990a;b; 1991a), but combined with evolution or black box optimization.

By making the temperature τ an adjustable parameter of the M model, we can see the effect of training the C model on hallucinated virtual environments with different levels of uncertainty, and see how well they transfer over to the actual environment. We experimented with varying the temperature in the virtual environment and observing the resulting average score over 100 random rollouts in the actual environment after training the agent inside the virtual environment with a given temperature:

TEMPERATURE τ	VIRTUAL SCORE	ACTUAL SCORE
0.10	2086 ± 140	193 ± 58
0.50	2060 ± 277	196 ± 50
1.00	1145 ± 690	868 ± 511
1.15	918 ± 546	1092 ± 556
1.30	732 ± 269	753 ± 139
RANDOM POLICY	N/A	210 ± 108
GYM LEADER	N/A	820 ± 58

Table 2. Take Cover scores at various temperature settings.

We see that while increasing the temperature of the M model makes it more difficult for the C model to find adversarial policies, increasing it too much will make the virtual environment too difficult for the agent to learn anything, hence in practice it is a hyperparameter we can tune. The temperature also affects the types of strategies the agent discovers. For example, although the best score obtained is 1092 ± 556 with $\tau = 1.15$, increasing τ a notch to 1.30 results in a lower score but at the same time a less risky strategy with a lower variance of returns. For comparison, the best score on the OpenAI Gym leaderboard (Paquette, 2016) is 820 ± 58.

5. Iterative Training Procedure

In our experiments, the tasks are relatively simple, so a reasonable world model can be trained using a dataset collected from a random policy. But what if our environments become more sophisticated? In any difficult environment, only parts of the world are made available to the agent only after it learns how to strategically navigate through its world.

For more complicated tasks, an iterative training procedure is required. We need our agent to be able to explore its world, and constantly collect new observations so that its world model can be improved and refined over time. An iterative training procedure (Schmidhuber, 2015a) is as follows:

1. Initialize M, C with random model parameters.

2. Rollout to actual environment N times. Agent may learn during rollouts. Save all actions a_t and observations x_t during rollouts to storage device.

3. Train M to model $P(x_{t+1}, r_{t+1}, a_{t+1}, d_{t+1}|x_t, a_t, h_t)$

4. Go back to (2) if task has not been completed.

We have shown that one iteration of this training loop was enough to solve simple tasks. For more difficult tasks, we need our controller in Step 2 to actively explore parts of the environment that is beneficial to improve its world model. An exciting research direction is to look at ways to incorporate artificial curiosity and intrinsic motivation (Schmidhuber, 2010; 2006; 1991b; Pathak et al., 2017; Oudeyer et al., 2007) and information seeking (Schmidhuber et al., 1994; Gottlieb et al., 2013) abilities in an agent to encourage novel exploration (Lehman & Stanley, 2011). In particular, we can augment the reward function based on improvement in compression quality (Schmidhuber, 2010; 2006; 1991b; 2015a).

In the present approach, since M is a MDN-RNN that models a probability distribution for the next frame, if it does a poor job, then it means the agent has encountered parts of the world that it is not familiar with. Therefore we can adapt and reuse M's training loss function to encourage curiosity. By flipping the sign of M's loss function in the actual environment, the agent will be encouraged to explore parts of the world that it is not familiar with. The new data it collects may improve the world model.

The iterative training procedure requires the M model to not only predict the next observation x and $done$, but also predict the action and reward for the next time step. This may be required for more difficult tasks. For instance, if our agent needs to learn complex motor skills to walk around its environment, the world model will learn to imitate its own C model that has already learned to walk. After difficult motor skills, such as walking, is absorbed into a large world model with lots of capacity, the smaller C model can rely on the motor skills already absorbed by the world model and focus on learning more higher level skills to navigate itself using the motor skills it had already learned.
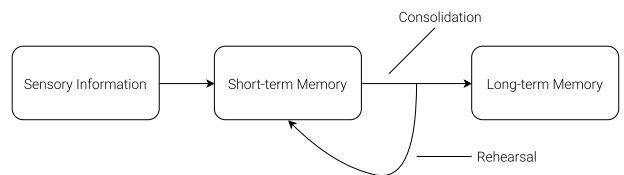

Figure 19. How information becomes memory.

An interesting connection to the neuroscience literature is the work on hippocampal replay that examines how the brain replays recent experiences when an animal rests or sleeps. Replaying recent experiences plays an important role in memory consolidation (Foster, 2017) – where hippocampus-dependent memories become independent of the hippocampus over a period of time. As (Foster, 2017) puts it, replay is *less like dreaming and more like thought*. We invite readers to read *Replay Comes of Age* (Foster, 2017) for a detailed overview of replay from a neuroscience perspective with connections to theoretical reinforcement learning.

Iterative training could allow the C–M model to develop a natural hierarchical way to learn. Recent works about self-play in RL (Sukhbaatar et al., 2017; Bansal et al., 2017; Al-Shedivat et al., 2017) and PowerPlay (Schmidhuber, 2013; Srivastava et al., 2012) also explores methods that lead to a natural curriculum learning (Schmidhuber, 2002), and we feel this is one of the more exciting research areas of reinforcement learning.

6. Related Work

There is extensive literature on learning a dynamics model, and using this model to train a policy. Many concepts first explored in the 1980s for feed-forward neural networks (FNNs) (Werbos, 1987; Munro, 1987; Robinson & Fallside, 1989; Werbos, 1989; Nguyen & Widrow, 1989) and in the 1990s for RNNs (Schmidhuber, 1990a;b; 1991a; 1990c) laid some of the groundwork for *Learning to Think* (Schmidhuber, 2015a). The more recent PILCO (Deisenroth & Rasmussen, 2011; Duvenaud, 2016; McAllister & Rasmussen, 2016) is a probabilistic model-based search policy method designed to solve difficult control problems. Using data collected from the environment, PILCO uses a Gaussian process (GP) model to learn the system dynamics, and then uses this model to sample many trajectories in order to train a controller to perform a desired task, such as swinging up a pendulum, or riding a unicycle.
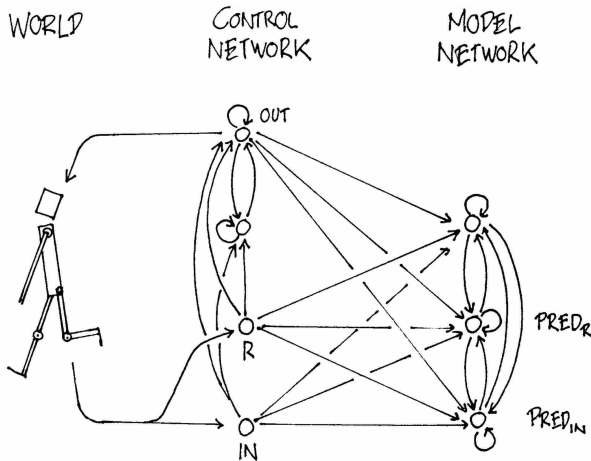

Figure 20. A controller with internal RNN model of the world (Schmidhuber, 1990a).

While Gaussian processes work well with a small set of low dimension data, their computational complexity makes them difficult to scale up to model a large history of high dimensional observations. Other recent works (Gal et al., 2016; Depeweg et al., 2016) use Bayesian neural networks instead of GPs to learn a dynamics model. These methods have demonstrated promising results on challenging control tasks (Hein et al., 2017), where the states are known and well defined, and the observation is relatively low dimensional.

Here we are interested in modelling dynamics observed from high dimensional visual data where our input is a sequence of raw pixel frames.

In robotic control applications, the ability to learn the dynamics of a system from observing only camera-based video inputs is a challenging but important problem. Early work on RL for active vision trained an FNN to take the current image frame of a video sequence to predict the next frame (Schmidhuber & Huber, 1991), and use this predictive model to train a fovea-shifting control network trying to find targets in a visual scene. To get around the difficulty of training a dynamical model to learn directly from high-dimensional pixel images, researchers explored using neural networks to first learn a compressed representation of the video frames. Recent work along these lines (Wahlstrm et al., 2014; 2015) was able to train controllers using the bottleneck hidden layer of an autoencoder as low-dimensional feature vectors to control a pendulum from pixel inputs. Learning a model of the dynamics from a compressed latent space enable RL algorithms to be much more data-efficient (Finn et al., 2015; Watter et al., 2015; Finn, 2017). We invite readers to watch Finn's lecture on Model-Based RL (Finn, 2017) to learn more.

Video game environments are also popular in model-based RL research as a testbed for new ideas. (Matthew Guzdial, 2017) used a feed-forward convolutional neural network (CNN) to learn a forward simulation model of a video game. Learning to predict how different actions affect future states in the environment is useful for game-play agents, since if our agent can predict what happens in the future given its current state and action, it can simply select the best action that suits its goal. This has been demonstrated not only in early work (Nguyen & Widrow, 1989; Schmidhuber & Huber, 1991) (when compute was a million times more expensive than today) but also in recent studies (Dosovitskiy & Koltun, 2016) on several competitive VizDoom environments.

The works mentioned above use FNNs to predict the next video frame. We may want to use models that can capture longer term time dependencies. RNNs are powerful models suitable for sequence modelling (Graves, 2013). In a lecture called *Hallucination with RNNs* (Graves, 2015), Graves demonstrated the ability of RNNs learn a probabilistic model of Atari game environments. He trained RNNs to learn the structure of such a game and then showed that they can hallucinate similar game levels on its own.

Using RNNs to develop internal models to reason about the future has been explored as early as 1990 in a paper called *Making the World Differentiable* (Schmidhuber, 1990a), and then further explored in (Schmidhuber, 1990b; 1991a; 1990c). A more recent paper called *Learning to Think* (Schmidhuber, 2015a) presented a unifying frame-

work for building a RNN-based general problem solver that can learn a world model of its environment and also learn to reason about the future using this model. Subsequent works have used RNN-based models to generate many frames into the future (Chiappa et al., 2017; Oh et al., 2015; Denton & Birodkar, 2017), and also as an internal model to reason about the future (Silver et al., 2016; Weber et al., 2017; Watters et al., 2017).

In this work, we used evolution strategies to train our controller, as it offers many benefits. For instance, we only need to provide the optimizer with the final cumulative reward, rather than the entire history. ES is also easy to parallelize – we can launch many instances of `rollout` with different solutions to many workers and quickly compute a set of cumulative rewards in parallel. Recent works (Fernando et al., 2017; Salimans et al., 2017; Ha, 2017b; Stanley & Clune, 2017) have confirmed that ES is a viable alternative to traditional Deep RL methods on many strong baselines.

Before the popularity of Deep RL methods (Mnih et al., 2013), evolution-based algorithms have been shown to be effective at solving RL tasks (Stanley & Miikkulainen, 2002; Gomez et al., 2008; Gomez & Schmidhuber, 2005; Gauci & Stanley, 2010; Sehnke et al., 2010; Miikkulainen, 2013). Evolution-based algorithms have even been able to solve difficult RL tasks from high dimensional pixel inputs (Koutnik et al., 2013; Hausknecht et al., 2013; Parker & Bryant, 2012). More recent works (Alvernaz & Togelius, 2017) combine VAE and ES, which is similar to our approach.

7. Discussion

We have demonstrated the possibility of training an agent to perform tasks entirely inside of its simulated latent space dream world. This approach offers many practical benefits. For instance, running computationally intensive game engines require using heavy compute resources for rendering the game states into image frames, or calculating physics not immediately relevant to the game. We may not want to waste cycles training an agent in the actual environment, but instead train the agent as many times as we want inside its simulated environment. Training agents in the real world is even more expensive, so world models that are trained incrementally to simulate reality will make it easier to experiment with different approaches for training our agents.

Furthermore, we can take advantage of deep learning frameworks to accelerate our world model simulations using GPUs in a distributed environment. The benefit of implementing the world model as a fully differentiable recurrent computation graph also means that we may be able to train our agents in the dream directly using the backpropagation algorithm to fine-tune its policy to maximize an objective function (Schmidhuber, 1990a;b; 1991a).

The choice of using a VAE for the V model and training it as a standalone model also has its limitations, since it may encode parts of the observations that are not relevant to a task. After all, unsupervised learning cannot, by definition, know what will be useful for the task at hand. For instance, it reproduced unimportant detailed brick tile patterns on the side walls in the Doom environment, but failed to reproduce task-relevant tiles on the road in the Car Racing environment. By training together with an M model that predicts rewards, the VAE may learn to focus on task-relevant areas of the image, but the tradeoff here is that we may not be able to reuse the VAE effectively for new tasks without retraining.

Learning task-relevant features has connections to neuroscience as well. Primary sensory neurons are released from inhibition when rewards are received, which suggests that they generally learn task-relevant features, rather than just any features, at least in adulthood (Pi et al., 2013).

Future work might explore the use of an unsupervised segmentation layer like in (Byravan et al., 2017) to extract better feature representations that might be more useful and interpretable compared to the VAE representations learned.

Another concern is the limited capacity of our world model. While modern storage devices can store large amounts of historical data generated using the iterative training procedure, our LSTM (Hochreiter & Schmidhuber, 1997; Gers et al., 2000)-based world model may not be able to store all of the recorded information inside its weight connections. While the human brain can hold decades and even centuries of memories to some resolution (Bartol et al., 2015), our neural networks trained with backpropagation have more limited capacity and suffer from issues such as catastrophic forgetting (Ratcliff, 1990; French, 1994; Kirkpatrick et al., 2016). Future work may explore replacing the small MDN-RNN network with higher capacity models (Shazeer et al., 2017; Ha et al., 2016; Suarez, 2017; van den Oord et al., 2016; Vaswani et al., 2017), or incorporating an external memory module (Gemici et al., 2017), if we want our agent to learn to explore more complicated worlds.

Like early RNN-based C–M systems (Schmidhuber, 1990a;b; 1991a; 1990c), ours simulates possible futures time step by time step, without profiting from human-like hierarchical planning or abstract reasoning, which often ignores irrelevant spatial-temporal details. However, the more general *Learning To Think* (Schmidhuber, 2015a) approach is not limited to this rather naive approach. Instead it allows a recurrent C to learn to address *subroutines* of the recurrent M, and reuse them for problem solving in arbitrary computable ways, e.g., through hierarchical planning or other kinds of exploiting parts of M's program-like weight matrix. A recent *One Big Net* (Schmidhuber, 2018) extension of the C–M approach collapses C and M into a single network, and uses PowerPlay-like (Schmidhuber, 2013; Srivastava et al.,
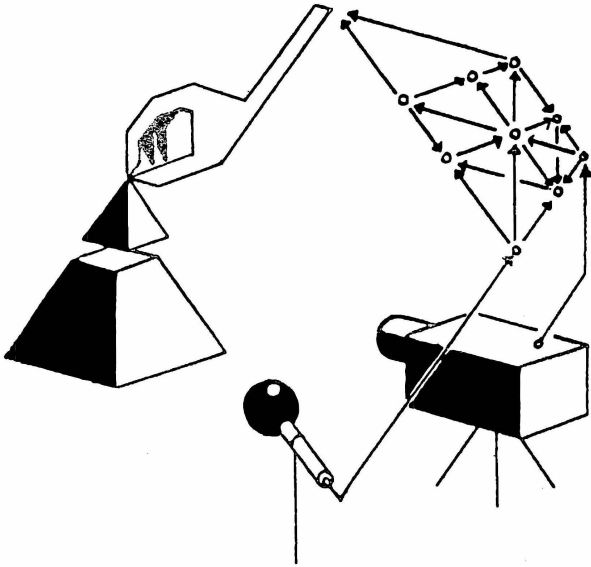
Figure 21. Ancient drawing (1990) of a RNN-based controller interacting with an environment (Schmidhuber, 1990a).

2012) behavioural replay (where the behaviour of a teacher net is compressed into a student net (Schmidhuber, 1992)) to avoid forgetting old prediction and control skills when learning new ones. Experiments with those more general approaches are left for future work.

Acknowledgements

Citation

For attribution in academic contexts, cite this work as:

```
Ha and Schmidhuber, World Models, 2018.
https://doi.org/10.5281/zenodo.1207631
```

BibTeX Citation:

```
@article{Ha2018WorldModels,
  author = {Ha, D. and Schmidhuber, J.},
  title = {World Models},
  eprint = {arXiv:1803.10122},
  doi = {10.5281/zenodo.1207631},
  url = {https://worldmodels.github.io},
  year = {2018}
}
```

A. Appendix

In this section we will describe in more details the models and training methods used in this work.

A.1. Variational Autoencoder

We trained a Convolutional Variational Autoencoder (ConvVAE) model as the V Model of our agent. Unlike vanilla autoencoders, enforcing a Gaussian prior over the latent vector z also limits the amount its information capacity for compressing each frame, but this Gaussian prior also makes the world model more robust to unrealistic z vectors generated by the M Model.

As the environment may give us observations as high dimensional pixel images, we first resize each image to 64x64 pixels before as use this resized image as the V Model's observation. Each pixel is stored as three floating point values between 0 and 1 to represent each of the RGB channels. The ConvVAE takes in this 64x64x3 input tensor and passes this data through 4 convolutional layers to encode it into low dimension vectors μ and σ, each of size N_z. The latent vector z is sampled from the Gaussian prior $N(\mu, \sigma I)$. In the Car Racing task, N_z is 32 while for the Doom task N_z is 64. The latent vector z is passed through 4 of deconvolution layers used to decode and reconstruct the image.

Each convolution and deconvolution layer uses a stride of 2. The layers are indicated in the diagram in *Italics* as *Activation-type Output Channels x Filter Size*. All convolu-
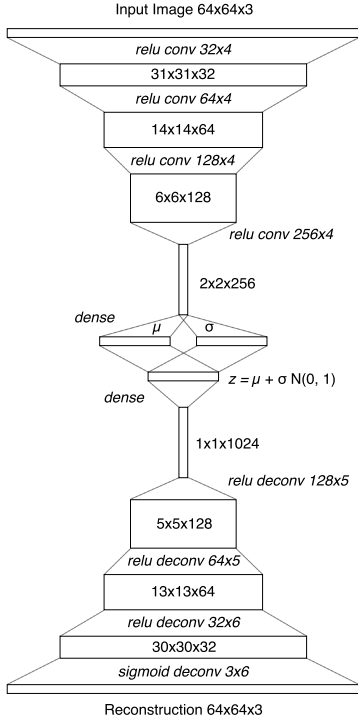
Figure 22. Description of the shape of our tensor at each layer of the ConvVAE.

tional and deconvolutional layers use relu activations except for the output layer as we need the output to be between 0 and 1. We trained the model for 1 epoch over the data collected from a random policy, using L^2 distance between the input image and the reconstruction to quantify the reconstruction loss we optimize for, in addition to KL loss.

A.2. Recurrent Neural Network

For the M Model, we use an LSTM (Hochreiter & Schmidhuber, 1997) recurrent neural network combined with a Mixture Density Network (Bishop, 1994) as the output layer. We use this network to model the probability distribution of the next z in the next time step as a Mixture of Gaussian distribution. This approach is very similar to (Graves, 2013) in the Unconditional Handwriting Generation section and also the decoder-only section of SketchRNN (Ha & Eck, 2017). The only difference in the approach used is that we did not model the correlation parameter between each element of z, and instead had the MDN-RNN output a diagonal covariance matrix of a factored Gaussian distribution.

Unlike the handwriting and sketch generation works, rather than using the MDN-RNN to model the pdf of the next pen stroke, we model instead the pdf of the next latent vector z. We would sample from this pdf at each timestep to generate the hallucinated environments. In the Doom task, we also also use the MDN-RNN to predict the probability of whether
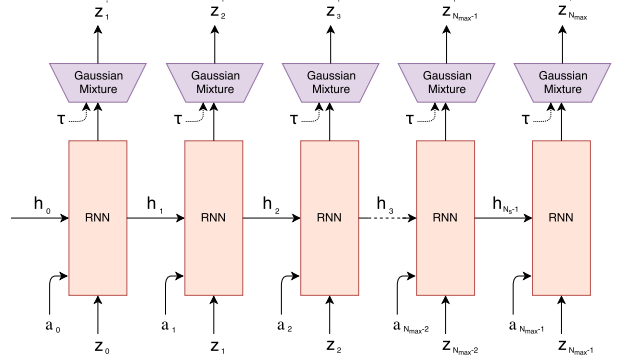

Figure 23. MDN-RNN decoder similar to (Graves, 2013; Ha & Eck, 2017)

the agent has died in this frame. If that probability is above 50%, then we set *done* to be *true* in the virtual environment. Given that death is a low probability event at each timestep, we find the cutoff approach to more stable compared to sampling from the Bernoulli distribution.

The MDN-RNNs were trained for 20 epochs on the data collected from a random policy agent. In the Car Racing task, the LSTM used 256 hidden units, while the Doom task used 512 hidden units. In both tasks, we used 5 Gaussian mixtures and did not model the correlation ρ parameter, hence z is sampled from a factored mixture of Gaussian distribution.

When training the MDN-RNN using teacher forcing from the recorded data, we store a pre-computed set of μ and σ for each of the frames, and sample an input $z \sim N(\mu, \sigma)$ each time we construct a training batch, to prevent overfitting our MDN-RNN to a specific sampled z.

A.3. Controller

For both environments, we applied $\tanh$ nonlinearities to clip and bound the action space to the appropriate ranges. For instance, in the Car Racing task, the steering wheel has a range from -1 to 1, the acceleration pedal from 0 to 1, and the brakes from 0 to 1. In the Doom environment, we converted the discrete actions into a continuous action space between -1 to 1, and divided this range into thirds to indicate whether the agent is moving left, staying where it is, or moving to the right. We would give the C Model a feature vector as its input, consisting of z and the hidden state of the MDN-RNN. In the Car Racing task, this hidden state is the output vector h of the LSTM, while for the Doom task it is both the cell vector c and the output vector h of the LSTM.

A.4. Evolution Strategies

We used *Covariance-Matrix Adaptation Evolution Strategy* (CMA-ES) (Hansen, 2016) to evolve the weights for our

C Model. Following the approach described in *Evolving Stable Strategies* (Ha, 2017b), we used a population size of 64, and had each agent perform the task 16 times with different initial random seeds. The fitness value for the agent is the *average cumulative reward* of the 16 random rollouts. The diagram below charts the best performer, worst performer, and mean fitness of the population of 64 agents at each generation:
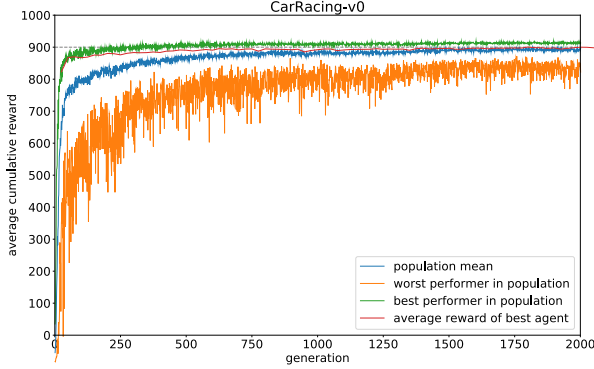

Figure 24. Training of `CarRacing-v0`

Since the requirement of this environment is to have an agent achieve an average score above 900 over 100 random rollouts, we took the best performing agent at the end of every 25 generations, and tested that agent over 1024 random rollout scenarios to record this average on the red line. After 1800 generations, an agent was able to achieve an average score of 900.46 over 1024 random rollouts. We used 1024 random rollouts rather than 100 because each process of the 64 core machine had been configured to run 16 times already, effectively using a full generation of compute after every 25 generations to evaluate the best agent 1024 times. Below, we plot the results of same agent evaluated over 100 rollouts:
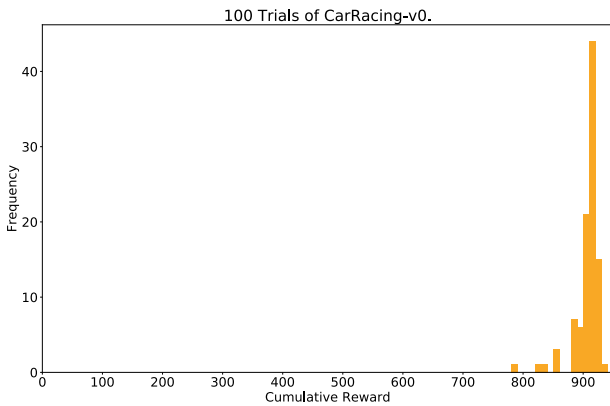

Figure 25. Histogram of cumulative rewards. Average score is 906 $\pm$ 21.

We also experimented with an agent that has access to only the z vector from the VAE, and not letting it see the RNN's hidden states. We tried 2 variations, where in the first variation, the C Model mapped z directly to the action space a. In second variation, we attempted to add a hidden layer with 40 $tanh$ activations between z and a, increasing the number of model parameters of the C Model to 1443, making it more comparable with the original setup.
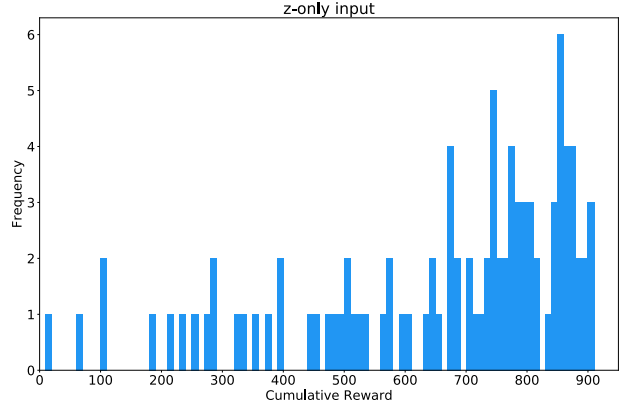

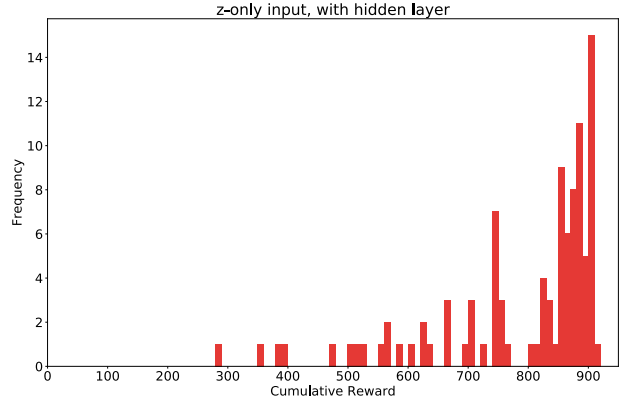*Figure 26.* When agent sees only z_t, average score is 632 $\pm$ 251.

Figure 27. When agent sees only z_t, with a hidden layer, average score is 788 $\pm$ 141.

A.5. DoomRNN

We conducted a similar experiment on the hallucinated Doom environment we called *DoomRNN*. Please note that we have not actually attempted to train our agent on the actual VizDoom environment, and had only used VizDoom for the purpose of collecting training data using a random policy. *DoomRNN* is more computationally efficient compared to VizDoom as it only operates in latent space without the need to render a screenshot at each timestep, and does not require running the actual Doom game engine.

In the virtual DoomRNN environment we constructed, we increased the temperature slightly and used $\tau = 1.15$ to make the agent learn in a more challenging environment. The best agent managed to obtain an average score of 959 over 1024 random rollouts (the highest score of the red
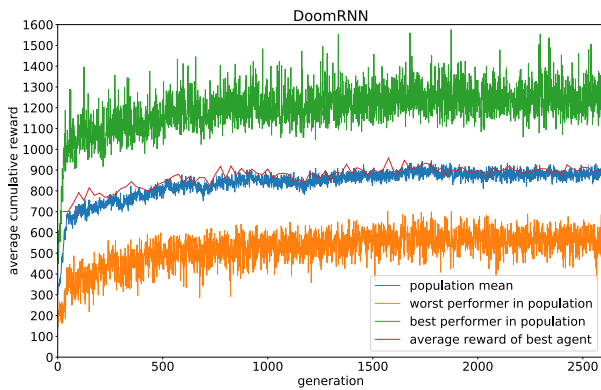
Figure 28. Training of DoomRNN.

line in the diagram). This same agent achieved an average score of 1092 ± 556 over 100 random rollouts when deployed to the actual *DoomTakeCover-v0* (Paquette, 2016) environment.
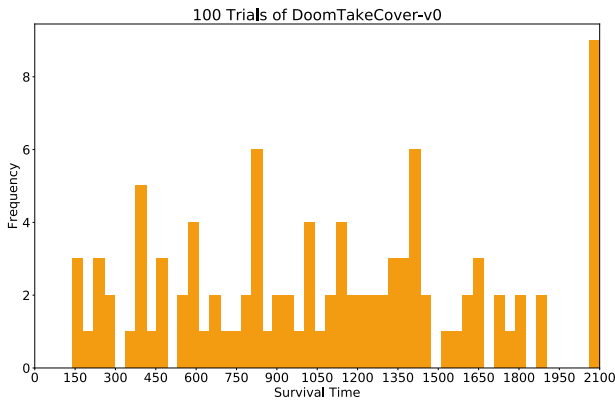

Figure 29. Histogram of timesteps survived in the actual VizDoom environment over 100 consecutive trials.

References

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. Continuous adaptation via meta-learning in nonstationary and competitive environments. *ArXiv preprint*, October 2017. URL https://arxiv.org/abs/1710.03641.

Alvernaz, S. and Togelius, J. Autoencoder-augmented neuroevolution for visual doom playing. *ArXiv preprint*, July 2017. URL https://arxiv.org/abs/1707.03902.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, Nov 2017. ISSN 1053-5888. doi: 10.1109/MSP.2017.2743240. URL https://arxiv.org/abs/1708.05866.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. Emergent complexity via multi-agent competition. *ArXiv preprint*, October 2017. URL https://arxiv.org/abs/1710.03748.

Bartol, Thomas M, Jr, Bromer, Cailey, Kinney, Justin, Chirillo, Michael A, Bourne, Jennifer N, Harris, Kristen M, and Sejnowski, Terrence J. Nanoconnectomic upper bound on the variability of synaptic plasticity. *eLife Sciences Publications, Ltd*, 2015. doi: 10.7554/eLife.10778. URL https://doi.org/10.7554/eLife.10778.

Bishop, Christopher M. Mixture density networks. *Technical Report*, 1994. URL http://publications.aston.ac.uk/373/.

Bling, Seth. Mar i/o kart, 2015. URL https://youtu.be/S9Y_I9vY8Qw.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *ArXiv preprint*, June 2016. URL https://arxiv.org/abs/1606.01540.

Byravan, A., Leeb, F., Meier, F., and Fox, D. Se3-posenets: Structured deep dynamics models for visuomotor planning and control. *ArXiv preprint*, October 2017. URL https://arxiv.org/abs/1710.00489.

Cheang, L. and Tsao, D. The code for facial identity in the primate brain. *Cell*, 2017. doi: 10.1016/j.cell.2017.05.011. URL http://www.cell.com/cell/fulltext/S0092-8674%2817%2930538-X.

Chiappa, S., Racaniere, S., Wierstra, D., and Mohamed, S. Recurrent environment simulators. *ArXiv preprint*, April 2017. URL https://arxiv.org/abs/1704.02254.

Deisenroth, M. and Rasmussen, C. Pilco: A model-based and data-efficient approach to policy search. 2011. URL http://mlg.eng.cam.ac.uk/pub/pdf/DeiRas11.pdf.

Denton, E. and Birodkar, V. Unsupervised learning of disentangled representations from video. *ArXiv preprint*, May 2017. URL https://arxiv.org/abs/1705.10915.

Depeweg, S., Hernandez-Lobato, J, Doshi-Velez, F., and Udluft, S. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *ArXiv preprint*, May 2016. URL https://arxiv.org/abs/1605.07127.

Dosovitskiy, A. and Koltun, V. Learning to act by predicting the future. *ArXiv preprint*, November 2016. URL https://arxiv.org/abs/1611.01779.

Duvenaud, David. Lecture slides on pilco. *CSC 2541 Course at University of Toronto*, 2016. URL https://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/pilco.pdf.

E, M. More thoughts from understanding comics by scott mccloud, 2012. URL https://goo.gl/5Tndi4.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *ArXiv preprint*, January 2017. URL https://arxiv.org/abs/1701.08734.

Finn, C., Tan, X., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. Deep spatial autoencoders for visuomotor learning. *ArXiv preprint*, September 2015. URL https://arxiv.org/abs/1509.06113.

Finn, Chelsea. Model-based rl lecture at deep rl bootcamp 2017, 2017. URL https://youtu.be/iC2a7M9voYU?t=44m35s.

Forrester, Jay Wright. Counterintuitive behavior of social systems, 1971. URL https://en.wikipedia.org/wiki/Mental_model. [Online; accessed 01-Nov-2017].

Foster, David J. Replay comes of age. *Annual Review of Neuroscience*, 40(1):581–602, 2017. doi: 10.1146/annurev-neuro-072116-031538. URL https://doi.org/10.1146/annurev-neuro-072116-031538.

French, Robert M. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In Cowan, J. D., Tesauro, G., and Alspector, J. (eds.), *Advances in Neural Information Processing Systems 6*, pp. 1176–1177. Morgan-Kaufmann, 1994. URL https://goo.gl/jwpLsk.

Gal, Y., McAllister, R., and Rasmussen, C. Improving pilco with bayesian neural network dynamics models. April 2016. URL http://mlg.eng.cam.ac.uk/yarin/PDFs/DeepPILCO.pdf.

Gauci, Jason and Stanley, Kenneth O. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 22(7):1860–1898, July 2010. ISSN 0899-7667. doi: 10.1162/neco.2010.06-09-1042. URL http://eplex.cs.ucf.edu/papers/gauci_nc10.pdf.

Gemici, M., Hung, C., Santoro, A., Wayne, G., Mohamed, S., Rezende, D., Amos, D., and Lillicrap, T. Generative temporal models with memory. *ArXiv preprint*, February 2017. URL https://arxiv.org/abs/1702.04649.

Gerrit, M., Fischer, J., and Whitney, D. Motion-dependent representation of space in area mt+. *Neuron*, 2013. doi: 10.1016/j.neuron.2013.03.010. URL http://dx.doi.org/10.1016/j.neuron.2013.03.010.

Gers, F., Schmidhuber, J., and Cummins, F. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL ftp://ftp.idsia.ch/pub/juergen/FgGates-NC.pdf.

Gomez, F. and Schmidhuber, J. Co-evolving recurrent neurons learn deep memory pomdps. *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pp. 491–498, 2005. doi: 10.1145/1068009.1068092. URL ftp://ftp.idsia.ch/pub/juergen/gecco05gomez.pdf.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9:937–965, June 2008. ISSN 1532-4435. URL http://people.idsia.ch/~juergen/gomez08a.pdf.

Gottlieb, J., Oudeyer, P., Lopes, M., and Baranes, A. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Cell*, September 2013. doi: 10.1016/j.tics.2013.09.001. URL http://www.pyoudeyer.com/TICSCuriosity2013.pdf.

Graves, Alex. Generating sequences with recurrent neural networks. *ArXiv preprint*, 2013. URL https://arxiv.org/abs/1308.0850.

Graves, Alex. Hallucination with recurrent neural networks, 2015. URL https://www.youtube.com/watch?v=-yX1SYeDHbg&t=49m33s.

Ha, D. Recurrent neural network tutorial for artists. *blog.otoro.net*, 2017a. URL http://blog.otoro.net/2017/01/01/recurrent-neural-network-artist/.

Ha, D. Evolving stable strategies. *blog.otoro.net*, 2017b. URL http://blog.otoro.net/2017/11/12/evolving-stable-strategies/.

Ha, D. and Eck, D. A neural representation of sketch drawings. *ArXiv preprint*, April 2017. URL https://magenta.tensorflow.org/sketch-rnn-demo.

Ha, D., Dai, A., and Le, Q. Hypernetworks. *ArXiv preprint*, September 2016. URL https://arxiv.org/abs/1609.09106.

Hansen, N. The cma evolution strategy: A tutorial. *ArXiv preprint*, 2016. URL https://arxiv.org/abs/1604.00772.

Hansen, Nikolaus and Ostermeier, Andreas. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, June 2001. ISSN 1063-6560. doi: 10.1162/106365601750190398. URL http://www.cmap.polytechnique.fr/~nikolaus.hansen/cmaartic.pdf.

Hausknecht, M., Lehman, J., Miikkulainen, R., and Stone, P. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2013. URL http://www.cs.utexas.edu/~ai-lab/?atari.

Hein, D., Depeweg, S., Tokic, M., Udluft, S., Hentschel, A., Runkler, T., and Sterzing, V. A benchmark environment motivated by industrial control problems. *ArXiv preprint*, September 2017. URL https://arxiv.org/abs/1709.09480.

Hirshon, B. Tracking fastballs, 2013. URL http://sciencenetlinks.com/science-news/science-updates/tracking-fastballs/.

Hochreiter, Sepp and Schmidhuber, Juergen. Long short-term memory. *Neural Computation*, 1997. URL ftp://ftp.idsia.ch/pub/juergen/lstm.pdf.

Hnermann, Jan. Self-driving cars in the browser, 2017. URL http://janhuenermann.com/projects/learning-to-drive.

Jang, S., Min, J., and Lee, C. Reinforcement car racing with a3c. 2017. URL https://www.scribd.com/document/358019044/Reinforcement-Car-Racing-with-A3C.

Jimenez Rezende, D., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *ArXiv preprint*, 2014. URL https://arxiv.org/abs/1401.4082.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: a survey. *Journal of AI research*, 4:237–285, 1996.

Keller, GeorgB., Bonhoeffer, Tobias, and Hbener, Mark. Sensorimotor mismatch signals in primary visual cortex of the behaving mouse. *Neuron*, 74(5):809 – 815, 2012. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2012.03.040. URL http://www.sciencedirect.com/science/article/pii/S0896627312003844.

Kelley, H. J. Gradient theory of optimal flight paths. *ARS Journal*, 30(10):947–954, 1960.

Kempka, Michael, Wydmuch, Marek, Runc, Grzegorz, Toczek, Jakub, and Jaskowski, Wojciech. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pp. 341–348, Santorini, Greece, Sep 2016. IEEE. URL http://arxiv.org/abs/1605.02097. The best paper award.

Khan, M. and Elibol, O. Car racing using reinforcement learning. 2016. URL https://web.stanford.edu/class/cs221/2017/restricted/p-final/elibol/final.pdf.

Kingma, D. and Welling, M. Auto-encoding variational bayes. *ArXiv preprint*, 2013. URL https://arxiv.org/abs/1312.6114.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.and Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *ArXiv preprint*, December 2016. URL https://arxiv.org/abs/1612.00796.

Kitaoka, Akiyoshi. Akiyoshi's illusion pages, 2002. URL http://www.ritsumei.ac.jp/~akitaoka/index-e.html.

Klimov, Oleg. Carracing-v0, 2016. URL https://gym.openai.com/envs/CarRacing-v0/.

Koutnik, J., Cuccu, G., Schmidhuber, J., and Gomez, F. Evolving large-scale neural networks for vision-based reinforcement learning. *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pp. 1061–1068, 2013. doi: 10.1145/2463372.2463509. URL http://people.idsia.ch/~juergen/compressednetworksearch.html.

Lau, Ben. Using keras and deep deterministic policy gradient to play torcs, 2016. URL https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html.

Lehman, Joel and Stanley, Kenneth. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. ISSN 1063-6560. URL http://eplex.cs.ucf.edu/noveltysearch/userspage/.

Leinweber, Marcus, Ward, Daniel R., Sobczak, Jan M., Attinger, Alexander, and Keller, Georg B. A sensorimotor circuit in mouse cortex for visual flow predictions. *Neuron*, 95(6):1420 – 1432.e5, 2017. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2017.08.036. URL http://www.sciencedirect.com/science/article/pii/S0896627317307791.

Linnainmaa, S. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki, 1970.

Matthew Guzdial, Boyang Li, Mark O. Riedl. Game engine learning from video. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3707–3713, 2017. doi: 10.24963/ijcai.2017/518. URL https://doi.org/10.24963/ijcai.2017/518.

McAllister, R. and Rasmussen, C. Data-efficient reinforcement learning in continuous-state pomdps. *ArXiv preprint*, February 2016. URL https://arxiv.org/abs/1602.02523.

McCloud, Scott. *Understanding Comics: The Invisible Art*. Tundra Publishing, 1993. URL https://en.wikipedia.org/wiki/Understanding_Comics.

Miikkulainen, R. Evolving neural networks. *IJCNN*, August 2013. URL http://nn.cs.utexas.edu/downloads/slides/miikkulainen.ijcnn13.pdf.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *ArXiv preprint*, December 2013. URL https://arxiv.org/abs/1312.5602.

Mobbs, Dean, Hagan, Cindy C., Dalgleish, Tim, Silston, Brian, and Prvost, Charlotte. The ecology of human fear: survival optimization and the nervous system., 2015. URL https://www.frontiersin.org/article/10.3389/fnins.2015.00055.

Munro, P. W. A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, pp. 165–176, 1987.

Nagabandi, A., Kahn, G., Fearing, R., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *ArXiv preprint*, August 2017. URL https://arxiv.org/abs/1708.02596.

Nguyen, N. and Widrow, B. The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 357–363. IEEE Press, 1989.

Nortmann, Nora, Rekauzke, Sascha, Onat, Selim, Knig, Peter, and Jancke, Dirk. Primary visual cortex represents the difference between past and present. *Cerebral Cortex*, 25(6):1427–1440, 2015. doi: 10.1093/cercor/bht318. URL http://dx.doi.org/10.1093/cercor/bht318.

Oh, J., Guo, X., Lee, H., Lewis, R., and Singh, S. Action-conditional video prediction using deep networks in atari games. *ArXiv preprint*, July 2015. URL https://arxiv.org/abs/1507.08750.

Oudeyer, P., Kaplan, F., and Hafner, V. Intrinsic motivation systems for autonomous mental development. *Trans. Evol. Comp*, apr 2007. doi: 10.1109/TEVC.2006.890271. URL http://www.pyoudeyer.com/ims.pdf.

Paquette, Philip. Doomtakecover-v0, 2016. URL https://gym.openai.com/envs/DoomTakeCover-v0/.

Parker, M. and Bryant, B. Neuro-visual control in the quake ii environment. *IEEE Transactions on Computational Intelligence and AI in Games*, 2012. URL https://www.cse.unr.edu/~bdbryant/papers/parker-2012-tciaig.pdf.

Pathak, D., Agrawal, P., A., Efros, and Darrell, T. Curiosity-driven exploration by self-supervised prediction. *ArXiv preprint*, May 2017. URL https://pathak22.github.io/noreward-rl/.

Pi, H., Hangya, B., Kvitsiani, D., Sanders, J., Huang, Z., and Kepecs, A. Cortical interneurons that specialize in disinhibitory control. *Nature*, November 2013. doi: 10.1038/nature12676. URL http://dx.doi.org/10.1038/nature12676.

Prieur, Luc. Deep-q learning for box2d racecar rl problem., 2017. URL https://goo.gl/VpDqSw.

Quiroga, R., Reddy, L., Kreiman, G., Koch, C., and Fried, I. Invariant visual representation by single neurons in the human brain. *Nature*, 2005. doi: 10.1038/nature03687. URL http://www.nature.com/nature/journal/v435/n7045/abs/nature03687.html.

Ratcliff, Rodney Mark. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97 2:285–308, 1990.

Rechenberg, I. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973. URL https://en.wikipedia.org/wiki/Ingo_Rechenberg.

Robinson, T. and Fallside, F. Dynamic reinforcement driven error propagation networks with application to game playing. In *CogSci 89*, 1989.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *ArXiv preprint*, 2017. URL https://arxiv.org/abs/1703.03864.

Schmidhuber, J. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. 1990a. URL http://people.idsia.ch/~juergen/FKI-126-90_(revised)bw_ocr.pdf.

Schmidhuber, J. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. *1990 IJCNN International Joint Conference on Neural Networks*, pp. 253–258 vol.2, June 1990b. doi: 10.1109/IJCNN.1990.137723. URL ftp://ftp.idsia.ch/pub/juergen/ijcnn90.ps.gz.

Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, pp. 222–227, 1990c. URL ftp://ftp.idsia.ch/pub/juergen/curiositysab.pdf.

Schmidhuber, J. Reinforcement learning in markovian and non-markovian environments. *Advances in Neural Information Processing Systems 3*, pp. 500–506, 1991a. URL https://goo.gl/ij1uYQ.

Schmidhuber, J. Curious model-building control systems. *In Proc. International Joint Conference on Neural Networks, Singapore*, pp. 1458–1463, 1991b.

Schmidhuber, J. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992. (Based on TR FKI-148-91, TUM, 1991).

Schmidhuber, J. Optimal ordered problem solver. *ArXiv preprint*, July 2002. URL https://arxiv.org/abs/cs/0207097.

Schmidhuber, J. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

Schmidhuber, J. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Trans. Autonomous Mental Development*, 2010. URL http://people.idsia.ch/~juergen/creativity.html.

Schmidhuber, J. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in Psychology*, 4:313, 2013. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00313. URL https://www.frontiersin.org/article/10.3389/fpsyg.2013.00313.

Schmidhuber, J. On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models.

ArXiv preprint, 2015a. URL https://arxiv.org/abs/1511.09249.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015b. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

Schmidhuber, J. One big net for everything. *Preprint arXiv:1802.08864 [cs.AI]*, February 2018. URL https://arxiv.org/abs/1802.08864.

Schmidhuber, J. and Huber, R. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2(1-2):125–134, 1991. doi: 10.1142/S012906579100011X. URL ftp://ftp.idsia.ch/pub/juergen/attention.pdf.

Schmidhuber, J., Storck, J., and Hochreiter, S. Reinforcement driven information acquisition in nondeterministic environments. Technical Report FKI- -94, TUM Department of Informatics, 1994.

Schwefel, H. *Numerical Optimization of Computer Models*. John Wiley and Sons, Inc., New York, NY, USA, 1977. ISBN 0471099880. URL https://en.wikipedia.org/wiki/Hans-Paul_Schwefel.

Sehnke, F., Osendorfer, C., Ruckstieb, T., Graves, A., Peters, J., and Schmidhuber, J. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010. doi: 10.1016/j.neunet.2009.12.004. URL http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=A64D1AE8313A364B814998E9E245B40A?doi=10.1.1.180.7104&rep=rep1&type=pdf.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ArXiv preprint*, January 2017. URL https://arxiv.org/abs/1701.06538.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. *ArXiv preprint*, December 2016. URL https://arxiv.org/abs/1612.08810.

Silver, David. David silver's lecture on integrating learning and planning, 2017. URL http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/dyna.pdf.

Srivastava, R., Steunebrink, B., and Schmidhuber, J. First experiments with powerplay. *ArXiv preprint*, October 2012. URL https://arxiv.org/abs/1210.8385.

Stanley, Kenneth and Clune, Jeff. Welcoming the era of deep neuroevolution, 2017. URL https://eng.uber.com/deep-neuroevolution/.

Stanley, Kenneth O. and Miikkulainen, Risto. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. URL http://nn.cs.utexas.edu/?stanley:ec02.

Suarez, Joseph. Language modeling with recurrent highway hypernetworks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 3269–3278. Curran Associates, Inc., 2017. URL https://goo.gl/4nqHXw.

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. *ArXiv preprint*, October 2017. URL https://arxiv.org/abs/1703.05407.

Sutton, Richard S. and Barto, Andrew G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981. URL http://ufal.mff.cuni.cz/~straka/courses/npfl114/2016/sutton-bookdraft2016sep.pdf.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *ArXiv preprint*, September 2016. URL https://arxiv.org/abs/1609.03499.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A, Kaiser, L., and Polosukhin, I. Attention is all you need. *ArXiv preprint*, June 2017. URL https://arxiv.org/abs/1706.03762.

Wahlstrm, N., Schn, T., and Deisenroth, M. Learning deep dynamical models from image pixels. *ArXiv preprint*, October 2014. URL https://arxiv.org/abs/1410.7550.

Wahlstrm, N., Schn, T., and Deisenroth, M. From pixels to torques: Policy learning with deep dynamical models. *ArXiv preprint*, June 2015. URL https://arxiv.org/abs/1502.02251.

Watanabe, Eiji, Kitaoka, Akiyoshi, Sakamoto, Kiwako, Yasugi, Masaki, and Tanaka, Kenta. Illusory motion reproduced by deep neural networks trained for prediction. *Frontiers in Psychology*, 9:345, 2018. ISSN 1664-1078. doi: 10.3389/fpsyg.2018.00345. URL https://www.frontiersin.org/article/10.3389/fpsyg.2018.00345.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. Embed to control: A locally linear latent dynamics model for control from raw images. *ArXiv preprint*, June 2015. URL https://arxiv.org/abs/1506.07365.

Watters, N., Tacchetti, A., Weber, T., Pascanu, R., Battaglia, P., and Zoran, D. Visual interaction networks. *ArXiv preprint*, June 2017. URL https://arxiv.org/abs/1706.01433.

Weber, T., Racanire, S., Reichert, D., Buesing, L., Guez, A., Rezende, D., Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Silver, D., and Wierstra, D. Imagination-augmented agents for deep reinforcement learning. *ArXiv preprint*, July 2017. URL https://arxiv.org/abs/1707.06203.

Werbos, P. J. Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, N.Y.*, 1987.

Werbos, P. J. Neural networks for control and system identification. In *Proceedings of IEEE/CDC Tampa, Florida*, 1989.

Werbos, Paul J. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pp. 762–770. Springer, 1982.

Wiering, Marco and van Otterlo, Martijn. *Reinforcement Learning*. Springer, 2012.

Wikipedia, Authors. Video game exploits, 2017. URL https://en.wikipedia.org/wiki/Video_game_exploits. [Online; accessed 01-Nov-2017].