CS-GY 6613 Final Project

Amber Barksdale azb9713
Vincent Haoyu Fang hf945
Russell Wustenberg rw2873

1 Task 1: Impact of a World Model on RL Learning

1.1 Introduction

The ultimate goal of reinforcement learning (RL) is the development of a general problem solving agent able to navigate a complex environment [26]. RL emulates the biological learning process of our own human brains. Through exploration and curiosity, the agent comes to better understand the rules of their environment and the behavior by which they may optimize their performance with respect to a given task [22]. These behaviors are encoded as a set of *policies* which govern how the agent reacts to observed environmental stimuli [24]. This is summarized in the definition of learning as expressed in [26], "*learning* or *credit assignment* is about finding real-valued weights that make the [neural network] exhibit desired behavior."

For simple task scenarios in a limited environment, these weights are easily learned. The agent performs a short sequence of actions in a highly defined space and easily prioritizes behavior patterns that result in success. This is exemplified in the Wumpus World experiments as shown in [24], where merely observing the current square and building up an observation history provides a deterministic method for success. At some point, however, the task or environment becomes too complex to make such direct correlations between behavior and success. This is referred to as the *credit assignment problem.*

In [12], authors Ha and Schmidhuber propose a methodology based on findings in neuroscience to improve the efficacy of RL agents. Specifically, their three-part *World Models* architecture overcomes the constraints which credit assignment imposes on the RL agent and makes possible previously impossible tasks.

1.2 Neuroscientific Background

The biological underpinnings of Ha and Schmidhuber's agents align well with the basal ganglia (BA) with dopamine (DA), prefrontal cortex (PFC), ventromedial prefrontal cortex (vmPFC), and the orbitofrontal cortex (OFC) [5] [15]. These together create the augmented model of reinforcement learning in the human brain. In humans, the temporal credit assignment problem is determining which actions lead to the best (maximized) outcome, which is largely dealt with by the augmented model previously mentioned [3][21].

The foundation of the augmented model is the primitive model, which involves the basal ganglia and dopamine (BG-DA). The BG-DA system runs an action controller called the cache system, learning motor habits and associating a reward with the stimuli that predicts it [3] [5] [6]. In humans, the response to making a correct or incorrect choice can be clearly measured through spikes or dips in DA; these dips in DA can occur when making a choice that does not lead to a reward or when receiving negative reinforcement [5][6]. Quite similarly to the World Models agents, the BG-DA system is limited: it cannot account for the magnitude of a gain or loss, so an augmented model is necessary for improved performance [5].

The addition of higher cortices such as the orbitofrontal cortex (OFC) and the ventromedial prefrontal cortex (vmPFC) allows for a “storage space” to encode what DA cannot - along with being more flexible as it updates itself with new information, which is necessary to guide behavior in rapidly-changing environments [5]. The vmPFC is suggested to encode the expected value of a stimulus, and together with the OFC create an augmented action controller called “tree search” [3][15]. The World Model agent components map well to the various portions of the augmented model: the M component takes on some responsibilities of the vmPFC, and the C component takes on the BG-DA system and the OFC.

1.3 The *World Models* Architecture

These observations about the human mind inspired Ha and Schmidhuber to create the *World Models* architecture for reinforcement learning [26]. The key observation they make is that the human body can be modeled as a system of specialized components with only a subset requiring input or output from the environment. This builds on Schmidhuber’s earlier work summarized in [26]. The output of each up-stream system provides only *essential* data to the downstream system. Specifically, the system attempts to do so in as succinct a form as possible. This has the benefit of reducing scope of learning at each successive step in the process, and sidesteps the bottleneck of credit assignment for the RL agent by distilling a complex environment to a simpler, more representative form [12].

Just as the human mind builds up an internal model of the real world, the *World*

Models (WM) agent builds a representative model of its environment. Taking in a full-resolution image, the WM agent learns to prioritize its most relevant aspects within the "world model." The output of this model is an approximation of the real environment, but has been shown to contain sufficient information for the RL agent to optimally perform the task assigned [1], [12]. This RL agent, called the "controller," only builds its policies based solely on the world model's output.

The world model is comprised of two sub-systems: the Visual (V) Model and the Memory (M) Model.[1] The V Model is responsible for taking in high-dimensional image data from the environment and summarizing its most important features. The M Model does its best to predict what the environment will look in the next time step given what is being observed in the V Model and the previous choices taken by the agent. The agent, responsible for selecting actions and receiving feedback from the environment, is the Controller (C) Model. The C Model takes in the heuristic of the World Model and is able to learn state-of-the-art capabilities. In the following sections, each model will be discussed in detail.

1.3.1 The Visual (V) Model

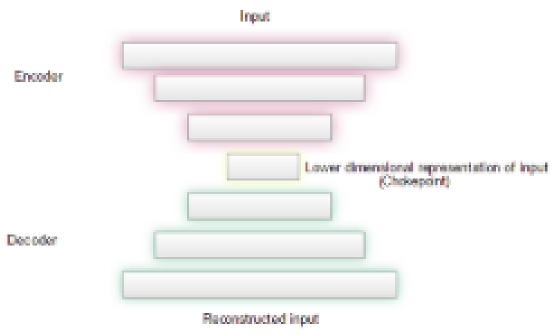
The task assigned to the V Model is to produce a low-dimensional representation of agent's complex environment. Auto-encoders are a natural choice for this task, as they have been an active area of neural network research since the 1980s [2], [20].

A Pure Auto-Encoder

[1]This section and that following is strongly based upon descriptions in [12].

An auto-encoder (AE) is composed of three basic parts: the *encoder*, the *chokepoint* and the *decoder* [1]. This structure can be seen in Figure 1. A full-resolution sample, usually an image, is provided to the encoder. The encoder compresses this sample into a latent vector z with greatly reduced dimensions. From this latent vector, the decoder reconstructs an approximation of the image. During training, the results of this reconstruction are compared with the original and the network adjusted until a certain similarity threshold is reached. The entire process hinges upon the capacity of the latent vector z to model features of the environment in summary.

Figure 1: *Generic AE structure. Figure from* [1].



Expressed mathematically, the auto-encoder first performs the action

$$z = enc(x)$$

where the latent vector z is the output of function $enc()$, the encoder, given input vector x. Then the decoding step takes place:

$$\tilde{x} \approx dec(z)$$

The model is trained to minimize the reconstructed error, which is traditionally calculated as

$$\mathcal{L}_{reconstruction} = \mathcal{L}_{MSE}(x, \tilde{x})$$

where $\mathcal{L}_{MSE}$ is the per-pixel mean squared error between the output $\tilde{x}$ and original input x. Section 3 covers another approach which addresses a fundamental flaw with this loss calculation.

The Variational Auto-Encoder

Pure AE models use a learned, linear mapping of x to z. This limits their usage and the results have many flaws. As a consequence, they were avoided by the deep learning community for many years[28]. Recent innovation has produced the variational auto-encoder (VAE) which has motivated further use of auto-encoders for feature extraction and data simplification [17]. Whereas the traditional AE is a *discriminative* model, modelling the latent vector z as a predictor of a *best fit* to input x, the VAE is a *generative* approach, modelling the latent *distribution* of z as a probabilistic model over all input variables[17].

The generative model in the VAE is a Bayesian network, meaning the encoder function $enc()$ now infers from the encoded data a mean, μ, and a covariance matrix, Σ. The covariance is an nxn matrix, whose diagonal is the latent vector z. The latent distribution $z \in \mathcal{R}^n$ is therefore drawn from the multivariate Gaussian distribution, $\mathcal{N}(\mu, \Sigma^2)$.

This is shown in Figure 2, which comes from a lecture presented by PhD candidate Mengwei Ren of New York University to the spring 2021 computer vision course taught by Professor James Fishbaugh [23]. In the figure, it can be seen that the encoder function,

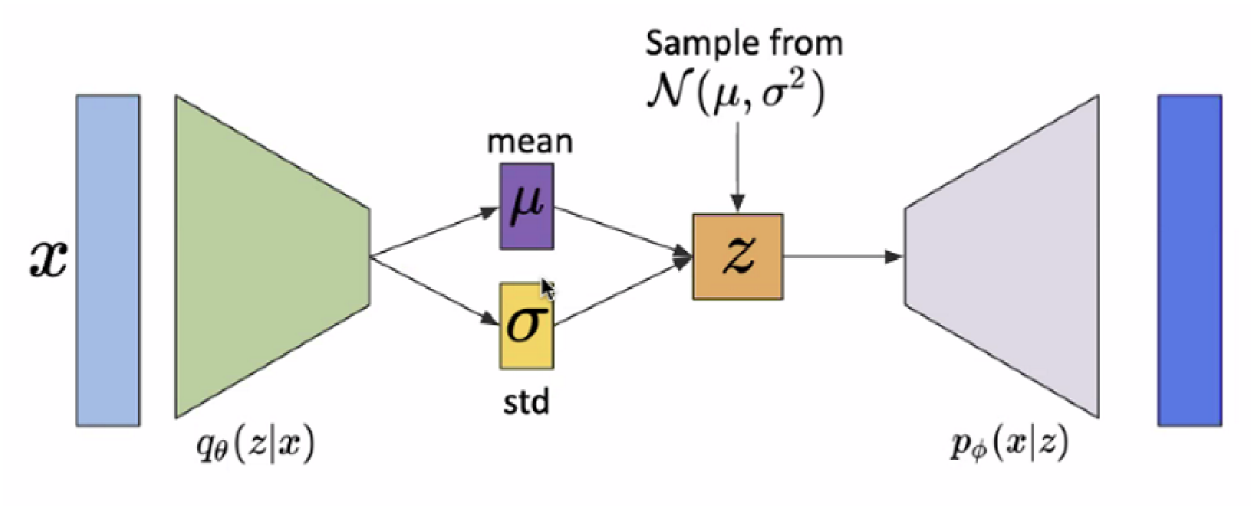


Figure 2: *Architecture of a variational auto-encoder. Figure from* [23].

here written as $q_\theta()$ compresses the input x to the distribution of z given x. Likewise, the decoding function p_ϕ samples from the *generative* distribution z and produces a sample, $\tilde{x}$. The similarity of the sample $\tilde{x}$ to input x is a result of the learned parameter sets θ and ϕ.[2]

As z is now a distribution, it may be sampled from at *any time* once the parameters have been tuned to the dataset. This is the primary mechanic by which the *World Models* architecture creates tokens of the information from the 64x64x3 input dimensions summarized in $z \in \mathcal{R}^{32}$. Figure 3 shows a side-by-side comparison of the *World Models* full-resolution frame (left) and the reconstructed output (right). The most prominent features are present in both images (the car placement and the road) but some details have been lost in the reconstruction (the red barrier). As discussed in [1], it is critical that the aspects of the environment which directly impact the score be represented in the latent compression. Here, the red boundary, indicating an upcoming turn, is less impactful than the contour of the road immediately preceding the car. In the experiments in [1], the mission-critical red

[2]There is a necessary reparameterization of the VAE in order to learn via backpropagation. This is covered in [17] and will not be discussed in depth at present.

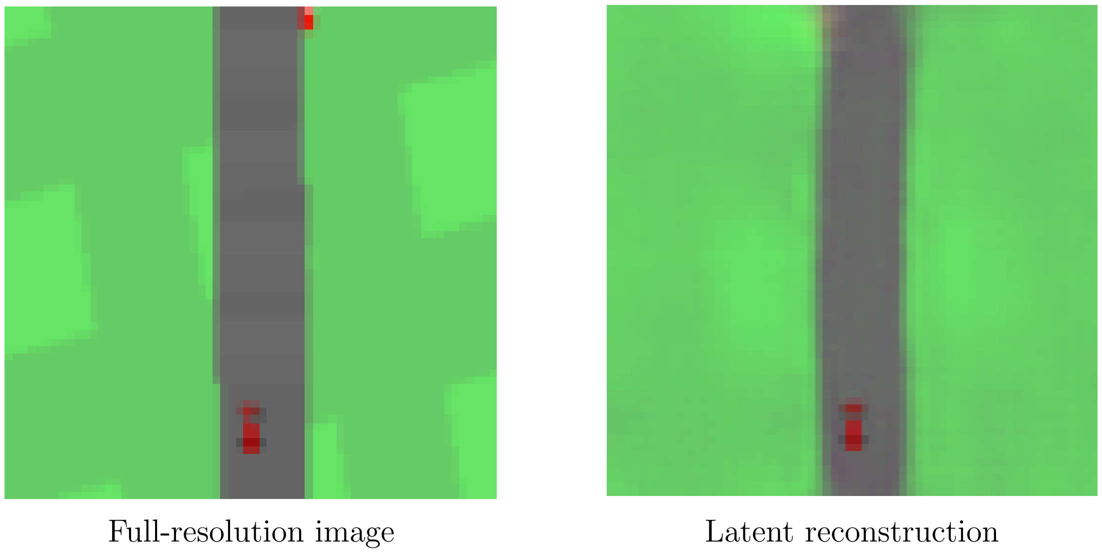


Figure 3: *Comparison of full-resolution and latent representation frames from the 'car racing' experiment. Images taken from the interactive version of* [12].

health packs were *not* captured by the latent variable in a given configuration, and the agent was unable to fully learn a policy that led to success.

The Convolution VAE or ConnVAE Finally, all of the elements are in place for a discussion of the *World Models* VAE, which is a particular kind of VAE called a convolution VAE or ConnVAE. The ConnVAE has been shown to be particularly capable at compressing image-based features by use of several convolution layers rather than the fully-connected layers used in early VAE architectures [17].

The *World Models* ConnVAE architecture, including all dimensions, is shown in Figure 4. As can be seen, the encoder takes the unraveled 64x64x3 input and passes it through four convolutional layers. The μ and Σ are then extracted from the resulting 2x2x256 layer by a dense (fully-connected) layer. These values are re-combined to produce the latent distribution z, which is here reparameterized by the equation $z = \mu + \sigma\mathcal{N}(0, I)$ in order for back propagation to traverse the entire length of the network [23].

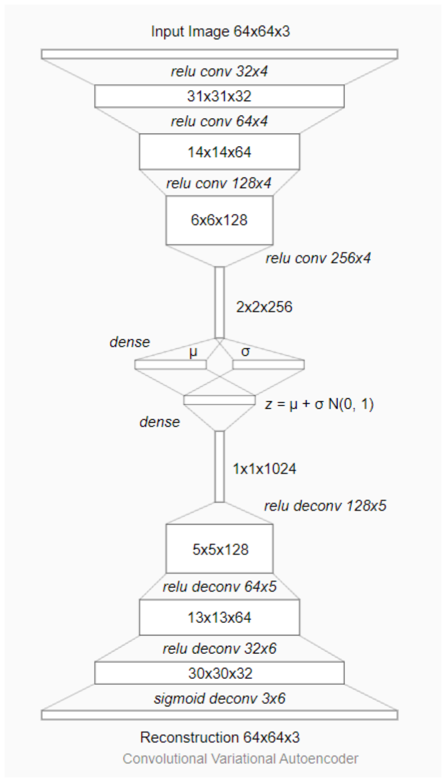


Figure 4: *World Models ConnVAE structure. Image from* [12].

As far as the M and C Models are concerned, it is only the encoding step of this V Model which is mission critical. The decoder, which reconstructs an approximation of the input by a similar convolution 'unpacking,' is only used during the training step.

1.3.2 The Memory (M) Model

The human brain is possibly capable of holding up to 100 years of historical sensory data [27]. By consequence, any learning agent attempting to emulate human capacity must treat "all data as holy" since what at first appear to be random noise may turn out to be a recurrent pattern in long term analysis [26]. Recurrent neural networks provide a means by which provide a means to encode past events into 'memories' that persist through time. The

M Model relies upon this mechanism for its predictive step given the sequence of previous environment states and agent actions.

If the V Model is the agent's eyes, the M Model is the agent's subconscious. Its job, and only job, is to make a prediction about the immediate future. To do this, the M Model is provided with the latent compression z for time-step t (denoted z_t). The M Model performs the operation generally summarized by the equation

$$z_{t+1} = f(z_t)$$

for some predictive function $f(x)$. This prediction is then fed forward to both the C Model, for immediate use, and back to the M Model as the network's recursive, hidden state, h.

The M Model is a built upon an MDN-RNN. This is a hybrid structure of two networks: a mixture density network (MDN) and a recurrent neural network (RNN). This type of model has seen great success in predicting probable next steps in complex systems such as predicting next pen strokes in drawings [11], making music [18], or even emulating the behavior of former presidents [25]. Figure 5 shows the summary of the M Model internal architecture 'unrolled' over several time-steps.

In Figure 5, at each time-step, a new latent vector z_t is fed in to the network along with an action a_t that has been chosen for that time-step. The reason that the action (a_t) and observation (z_t) have the same step count is that, in the sequence of the agent as a whole, it begins with no prior predictions about the environment state and the first action choice will be made 'blind' to the consequences.

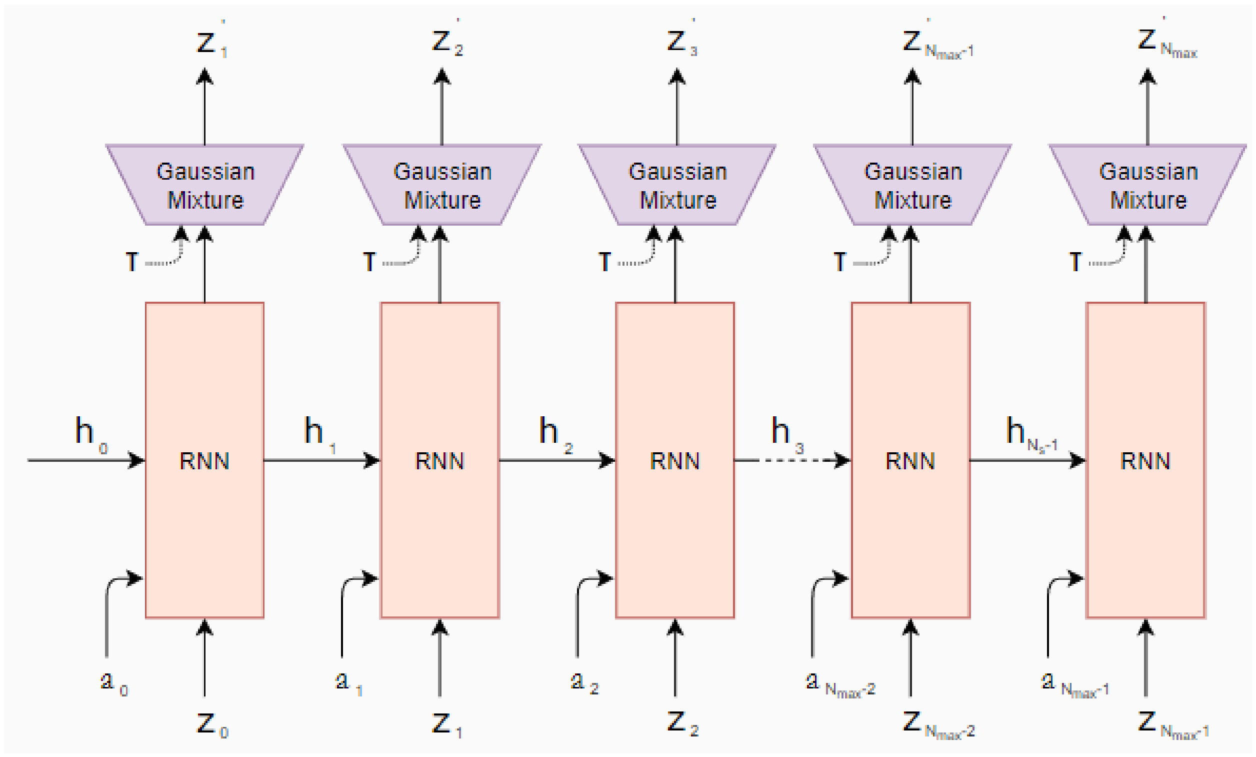


Figure 5: *Overview of the MDN-RNN architecture. Figure from* [12].

The variable h_t is the 'hidden state', or recurrent state, passed forward through time to future iterations of the network. As the RNN experiences the environment, the hidden state h will grow to represent, or "remember", more comprehensive conditions of the agent and the environment. It is by this mechanism that the RNN may be capable of developing a long-term *memory* as an entity in the environment. In [12], they even speculate that the RNN can grow to represent not only statistical facts about the environment, but eventually to subsume behaviors of the agent such as mobility.

The output of the RNN component of the M Model is then fed into a Gaussian Mixture Model of environment states along with the *temperature* hyperparameter (). Temperature plays a key role in keeping the agent from overfitting its policies to the rules of the environment.[3] The Gaussian Mixture Model (GMM) is a statistical explanation of discrete environment variables which makes the assumption that any point in the data set may be expressed as a *mixture* of k Gaussians. The data space is thereby divided into a complex, overlapping, model of states which may be sampled from given the prediction of the RNN [4].

An RNN Built on LSTM Units

The concept of the RNN, as visualized in the RNN portion of Figure 5, is of a network continually taking in new information and, based on previously input data, learning the probable contours of that data's sequence. Within this "black box" (or, here, "red box") diagram is in fact a complex network of interconnected neurons that not only propagate forward through the network but also laterally through time.

The basic structure of an RNN (again, unrolled) is shown in Figure 6. Here, x_t is an input x at time-step t. The input layer is fully connected to a set of hidden layers, h. These layers have a *recurrent connection* to their states at the previous time-step. [4] These hidden layers output in two directions: first, to the output layer, y_t and to itself at the next time-step, h_{t+1}. The output layer produces its proposed next-step prediction and, besides providing this to the user, also provides this in the form of a Bayesian state estimation to the input layer of the next time-step.

The performance of an RNN is driven by the internal architecture of the hidden

[3]An interesting anecdote is described in [12] where the agent learned a policy by which, within its "dream state" environment, it could magically dispel fireballs or manipulate the environment rules such that none were ever created.

[4]At the beginning of the sequence, $t = 0$, this recurrent state is initialized as a vector of all zeros.

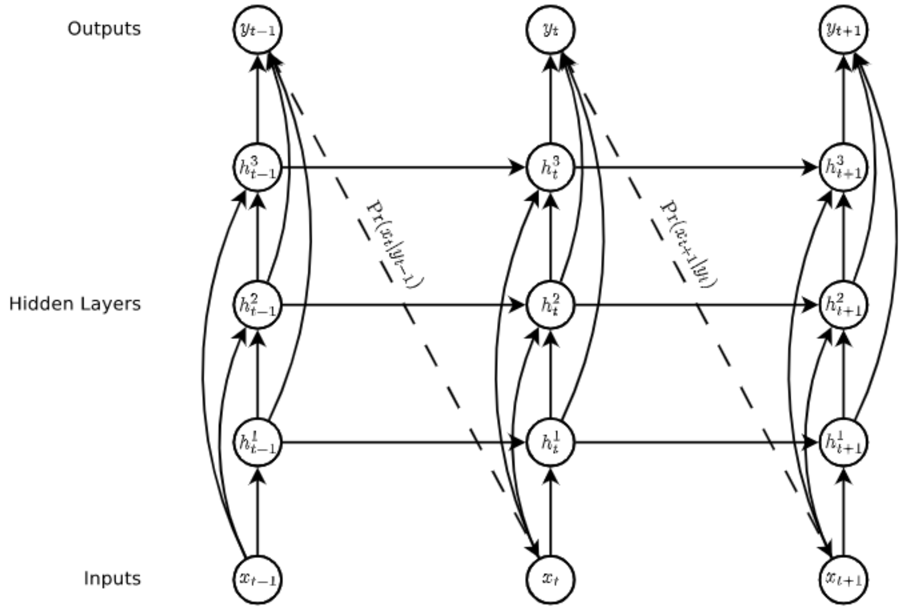


Figure 6: *Overview diagram of a recurrent neural network (RNN). Figure from* [10].

layer nodes. In early RNN architectures, the assumption was made that the known algorithms for building feed-forward networks would be effective. Each hidden unit was a linear function with a sigmoid activation [10]. This approach showed early promise, but had a major flaw in vanishing or exploding weight gradients as the error calculation propagated through the recurrent connections. It was necessary to rethink the hidden unit architecture to accommodate for this flaw.

The result of solving the gradient propagation issue with RNNs has led to the popular Long Short-Term Memory (LSTM) cell structure. Proposed originally in [14], the LSTM is a graphical model for hidden layer cells that introduces gates and directionality in such a way that the flows of error and information signals through the network are controlled. They call these gated blocks *memory cells* and have become fundamental to recurrent network design. Figure 7 shows the basic building blocks of the LSTM memory cell.

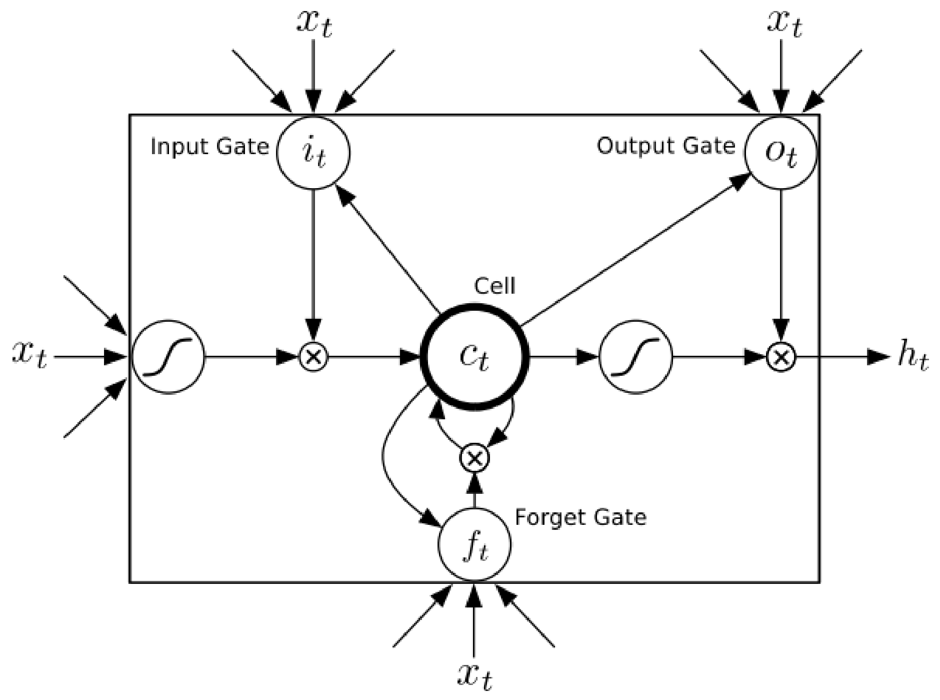


Figure 7: *Anatomy of the LSTM memory cell. Figure from* [10].

This is a complex design and a full account of its functionality is discussed in [10], from which the figure was taken. In brief, at time-step t, an input, x_t, is introduced to the network. This input if provided to *each* hidden layer at multiple points, resembling a skip connection used in the residual neural network (the other RNN). The cell ultimately will produce the output h_t, which is pushed forward through the network and laterally to itself in the next time step via the recurrent connection. Dividing the cell into three regions offers an *input* section, a *compute* section, and an *output* section.

The *input* action comprises two gates: an input gate, i_t, and an activation gate using the *sigmoid* non-linearity. Internally, these are multiplied by learned weight vectors and then multiplied together. The result of this input is fed to the *compute* section, which is a highly dense process involving two functions.

The “forget gate”, f_t, is arguably the cornerstone of the LSTM model. The weights within this gate are a combination of set and learned parameters which determine how strongly the recursive signal is 'present' in the memory cell. Information beyond a certain threshold in value or in time will be subject to forgetting. Conversely, information contained within the desired window will be utilized in the prediction and carried forward until forgotten. This is a broad summary of this technique, and the reader is encouraged to seek more information in [14] and [10].

The “cell”, c_t, is the computation of the unit and consists of several learned weight matrices applied to the input and recursive connections. The outcome of this calculation is fed forward into an *output* gate, o_t, which is symmetrical to the input gate in construction but produces the final memory cell output h_t. The activation in c_t and o_t is typically the *tanh* function.

The *World Models* architecture consists of several stacked LSTM units which are then fed into the subsequent output layer. In this particular RNN, that output layer is another neural network, the Gaussian Mixture Network (GMN).

The Gaussian Mixture Output Layer

Finite mixture models have increased in popularity as a tool for model-based clustering, as they can be used in multiple areas such as pattern recognition, image analysis, data mining, and other problems involving clustering and classification [30]. As a brief overview of the math: let y_i be a p-dimensional random vector. It contains p quantitative variables of interest for the i^{th} statistical unit, where $i = 1$ up to n. y_i is distributed as a Gaussian mixture model (GMM) with k components if

$$f(y_i;\theta) = \sum_{j=1}^{k} \pi_j \phi^{(p)}(y_i;\mu_j,\Sigma_j)$$

π_j are the positive weights subject to sum from $j = 1$ to k of $\pi_j = 1$, and μ_j, Σ_j are the parameters of the Gaussian components [30]. A Gaussian mixture model has a related factor-analytic representation via a linear model with a certain prior probability as $y_i = \mu_j + \Lambda_j z_i + u_i$ with probability π_j, where z_i is a p-dimensional latent variable with a multivariate standard Gaussian distribution and u_i is an independent vector of random errors [30].

The GMM layer is designed to be easily moved to any part of the structure, however, it is mainly used as the last layer [29]. Each node in the GMM sub-layer has g Gaussian components [29]. The number of nodes in the layer is equal to the number of classes (or states) [29]. For each state s, the corresponding node in the GMM layer outputs the negative log likelihood being produced by a mixture of Gaussians [29]. Additionally, there are four sub-layers: a *linear bottleneck sublayer* of d nodes that feeds its output to all nodes in the GMM layer as the input feature x (of dimensionality d), and three layers that store real-valued parameters for each node ($\mu - layer$, $\Sigma - layer$, and $\omega - layer$) [29].

GMMs primarily function on the classification side, and can be used as an alternative to the softmax layer [29]. Combining a GMM with something else - such as an RNN in the World Models paper - combines the advantages of the deeper neural network and GMM representation [29]. Using an RNN with a GMM output layer may be excessive, but the GMM has discrete modes, which is useful for environments with random discrete events

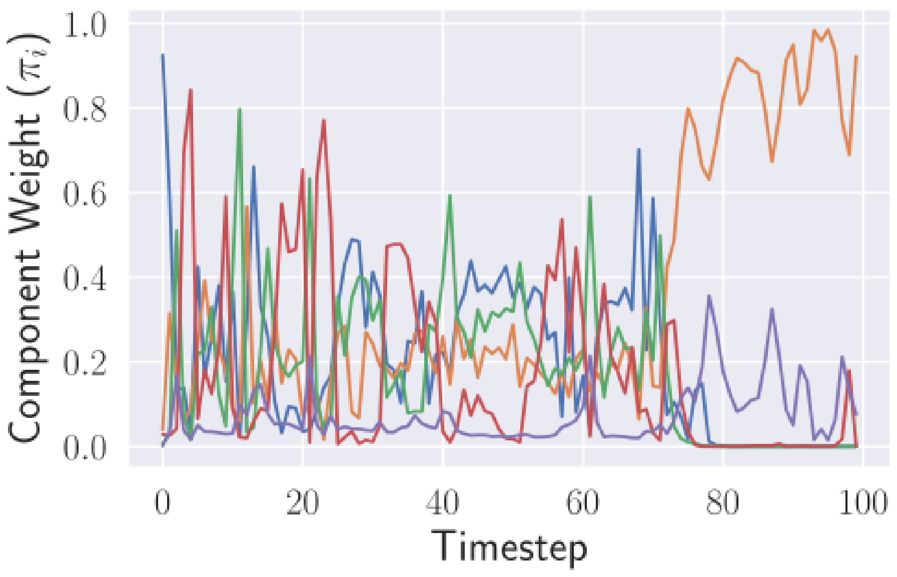


Figure 8: *Activations of sub-distributions with the GMM during 'VizDoom' experiment. Figure from* [4].

[12]. The RNN with a GMM output layer makes it easier to model the logic behind a more complicated environment with discrete random states [12].

The exact nature of this modelling has been explored deeply in the work done in [4]. Taking the MDN-RNN from the *World Models* architecture, they subjected the M Model to a variety of analytical tests to correlate the reconstructed 'dream state' it is predicting to neuron activations within the GMM topology. Figure 8 shows which internal Gaussian is dominant within the GMM as a single rollout of the *VisDoom* environment unfolds. The five colored lines represent the five internal Gaussians which segment the joint distribution of the GMM. Time is plotted along the x-axis as 100 time-steps fed into the MDN-RNN. The y-axis plots the π_i value for each distribution at that time-step. Recall, as π_i is the measure of the probability that a given state is generated by a given distribution, they will always sum to 1.0.

At the beginning of the rollout displayed in Figure 8, when the environment is in a neutral state, all the distributions are of equal probability to have generated the predicted frame. Towards the end of the sampled sequence, a fireball hits the character and detonates. As can be seen, the orange network takes dominance over all others. Two conclusions are reached in [4] on how to interpret this activation pattern. First, they conclude that different component distributions within the GMN model different possible 'directions,' or event sequences, for the future. Second, they conclude that the information contained within each component contains different environmental 'rules.' For example, when they run a prediction on the GMM on a model where the orange network has been deactivated, it will accurately predict the creation and trajectory of fireballs, but they will never impact and may even reverse course to their caster [4].

1.3.3 The Controller (C) Model

Finally, the C Model is the "conscious brain" of the agent, able to actively make goal-oriented decisions based on the unbiased observations from the previous sub-systems. Akin to how elements of the human body, such as the eye, function without regard to what is provided to them by the environment, the V and M models make no judgement upon the data which they are processing. It is only the C Model, tasked with selecting appropriate actions, that makes a utility-based judgement based on the given state information. For this reason, it is only the C Model that has access to the reward information from the environment. In effect, it is the only subsystem for which the reward mechanic impacts performance.

The C model was deliberately made as small and as simple as possible, even being trained separately from the V and M models to ensure that most of the agents' complexity would reside in the V and M models. It was a simple single-layer linear model with a relatively small number of parameters that mapped z_t and the M model's hidden state h_t directly to action a_t at each time step. Note that z_t and h_t are the concatenated input vector, and are multiplied by the weight matrix W_c with an addition of bias, b_c. The output is action vector a_t, which affects the environment: $a_t = W_c[z_t h_t] + b_c$. In both the car racing and Doom experiments, *tanh* nonlinearities were applied to bound the action space to the appropriate range - in the car racing experiment, the steering wheel had a range from -1 to 1, and the acceleration and brake pedals had ranges from 0 to 1.

The minimal design for C offered a handful of practical benefits, including allowing for exploration of unconventional methods of training. One such unconventional method includes evolution strategies (ES) to tackle more challenging RL tasks where the credit assignment problem is difficult. A covariance-matrix adaptation evolution strategy (CMA-ES) was chosen as an algorithm to optimize the parameters of C, which is known to work well for solution spaces of up to a few thousand parameters. The parameters of C could evolve on a single machine with multiple CPU cores, running multiple rollouts of the environment in parallel. Overall, the C model relies upon the validity of the output of its subsystems in order to evaluate the utility of its actions. Therefore, despite being simple, if the previously discussed models were trained well, then the C model could make good decisions that benefit the agent as a whole.

1.4 The Benefits of a Model-Based Approach

The above system has been shown to perform well on a number of challenging reinforcement learning tasks. It also comes with a number of architectural quirks which can be exploited to improve performance when not sampling directly from the actual training data via the generative distribution within the V Model. A number of these benefits will now be discussed.

First, the modularity of the agent architecture allows each sub-system to be trained independently. There is a sequential element of the training of the agent. The M Model samples from the latent distribution of the V Model, meaning that the V Model must first condition this distribution to the dataset which it is tasked to encode. As well, the C Model takes the outputs of both previous models as input. The consequence is that parallel training is not possible when building up a completely clean model.

However, since each sub-system is specialized to a task, training only needs to optimize the model to that specific task. The past few decades have given rise to optimal methods for training these various models in supervised and unsupervised contexts. As the reinforcement learning component of the agent is only relevant to the C Model, all associated sub-systems can be trained in whichever way desired by the trainer. In section 3, one such optimization will be applied to the V Model as an example of this particular architectural benefit.

As well, the volume of parameters to be learned by each model has been greatly reduced, speeding up the training in comparison to previous architectures [12]. Figure 9

Model	Parameter Count
VAE	4,348,547
MDN-RNN	422,368
CONTROLLER	867

Figure 9: *Parameters per model for the WM 'car racing' experiment. Figure from* [12].

shows the number of learned parameters for the entire WM architecture. As can be seen, only the V Model greatly scales down the parameter space of the learning problem by producing the latent vector $z \in R^{32}$. The result is that both the MDN-RNN and, particularly, the C Model have a much compressed data space to learn from. This is the primary reason the credit assignment problem is resolved with this architecture. The 867 parameters of the C Model permit the well-known CMA-ES strategy to be applied in developing suitable policies.

Finally, perhaps the most surreal benefit of the architecture, the WM agent is capable of learning without the aid of the training data. Once the initial exploration of the environment has been completed, the agent can be trained 'off-line' in a so-called "dream state" generated by the interplay between the controller and the distributions within the V and M models. For certain tasks, where training on actual data could prove prohibitive, the dream state provides a safe and effective means of improving results.

2 Task 2: Replicating the WM Car Racing Experiment

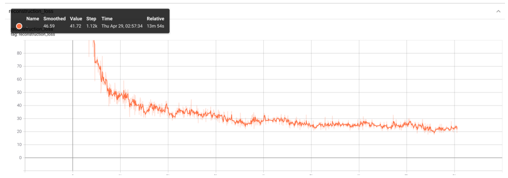
To better understand the results of the original paper [12], we attempted to reproduce one of the the original experiments using a public repository created by Zac Wellmer [31]. The repository was constructed in such a manner that replication should have been possible with a small set of prompts at the command line. However, the repository relies upon the Nvidia Docker tool to bundle several scripts into a large algorithm that trains the entire *World Models* agent in an end-to-end manner. Despite the recent date of Wellmer's work on the repository, a number of dependency issues complicated a theoretically simple replication. In the end, we chose not to utilize the training procedure outlined by Wellmer, but rather to return to the approach outlined in [12] and train the components independently.

To this end, we divided the algorithm into two main parts. First, we trained the VAE and RNN models sequentially. Then, we trained the Controller using the CMA-ES algorithm to optimize decision making. We chose to only replicate the '*car racing*' experiment from [12]. The training was done on two devices: one desktop had Nivida Titan X GPU device with 12 GB memory and 32 CPU cores that supported at most four workers for multi-thread processing, the other was a remote AWS server which provided CPU support only. The project performs in a virtual environment with Python 3.6 and Tensorflow 2.4. Other important dependencies are shown in the following chart:

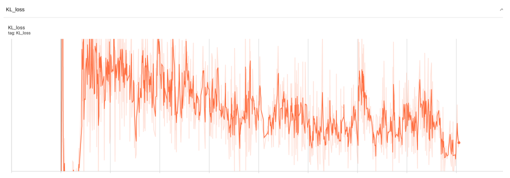
Module	**Version**
mpi4py	3.0.3
pyglet	1.3.2
tf-nightly	2.6.0
tensorflow-probability	0.10.1

Other common three-party dependencies were upgraded to the latest version.

Figure 10: Side-by-side comparison of VAE loss by MSE and KL Divergence.



(a) VAE loss calculated by MSE



(b) VAE loss calculated by KL Divergence

2.1 Results from Training the VAE

Ultimately, we were able to run the VAE training for a total of 9 epochs with 1000 iterations each. The loss for the model is calculated as the weighted sum of two terms,

$$\mathcal{L}_{\mathcal{VAE}} = \mathcal{L}_{MSE} + \mathcal{L}_{KL}$$

where $\mathcal{L}_{MSE}$ is the per-pixel mean squared error loss between the input image and the output reconstruction, and $\mathcal{L}_{KL}$ is the Kulbeck-Leibler Divergence score between the latent vector z and the input entropy given the current image. We find that the original implementation's score, without scaling, results in large loss values. We apply a weight factor of 0.5 to both $\mathcal{L}_{MSE}$ and $\mathcal{L}_{KL}$. Figure 10 shows plots of the $\mathcal{L}_{MSE}$ and $\mathcal{L}_{KL}$ components side by side. At a glance it is clear that the MSE plots a preferable loss shape, approaching a general value for convergence. KL Divergence, on the other hand, plots an erratic trajectory across a wide range of values. The VAE loss function of [12], it can be concluded, utilizes $\mathcal{L}_{MSE}$ as the driving loss function but attempts to "spice things up"[5] with the KL Divergence. A motivation for this approach may have been to avoid known issues with the VAE training model that will be discussed in Task 3.

[5]A phrase used several times in code comments within the repository, usually in reference to some innovation cooked up by Ha.

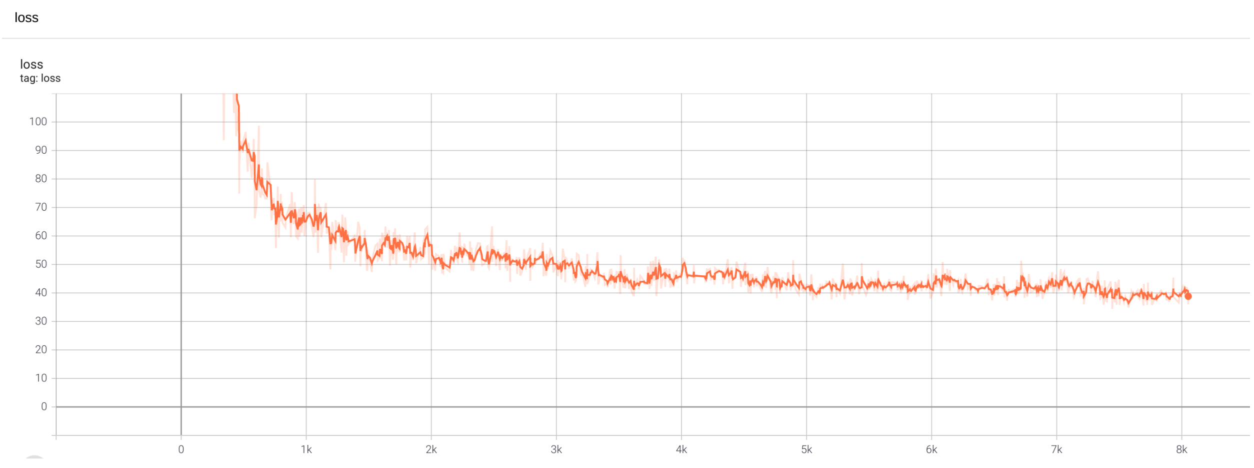


Figure 11: Total loss during VAE Training. The 9000 training iterations are plotted on the x-axis, indexed from [0, 8000]. Each vertical line represents the division of one epoch. Along the y-axis is plotted the log value of the loss equation.

The total loss over time is plotted in Figure 11. The overall shape is similar to that of $\mathcal{L}_{MSE}$ from Figure 10a, but with heightened variance from the $\mathcal{L}_{KL}$ component. The total $\mathcal{L}_{VAE}$ begins asymptotically large, enters within the plotting space of the graph, and converges towards a value of approximately 40.

The clearest evidence of the VAE's functionality is seen in Figure 12, which shows the input and output for four frames from the training set. It is evident that even with our short training schedule, the VAE has successfully captured salient aspects of the scene. The road silhouettes are accurate within a small degree of error, the car is accurately placed, and the red-and-white turn markers even appear (something not obtained by the model in [12]).

One interesting detail arising from this compression is in the reconstruction of the image in the left-most column. In the input image, the car is completely vertical, implying that (if it is moving at a fast speed) it will likely run off the road in the next few frames. The reconstruction, however, shows the car at a slight angle, implying that the model may

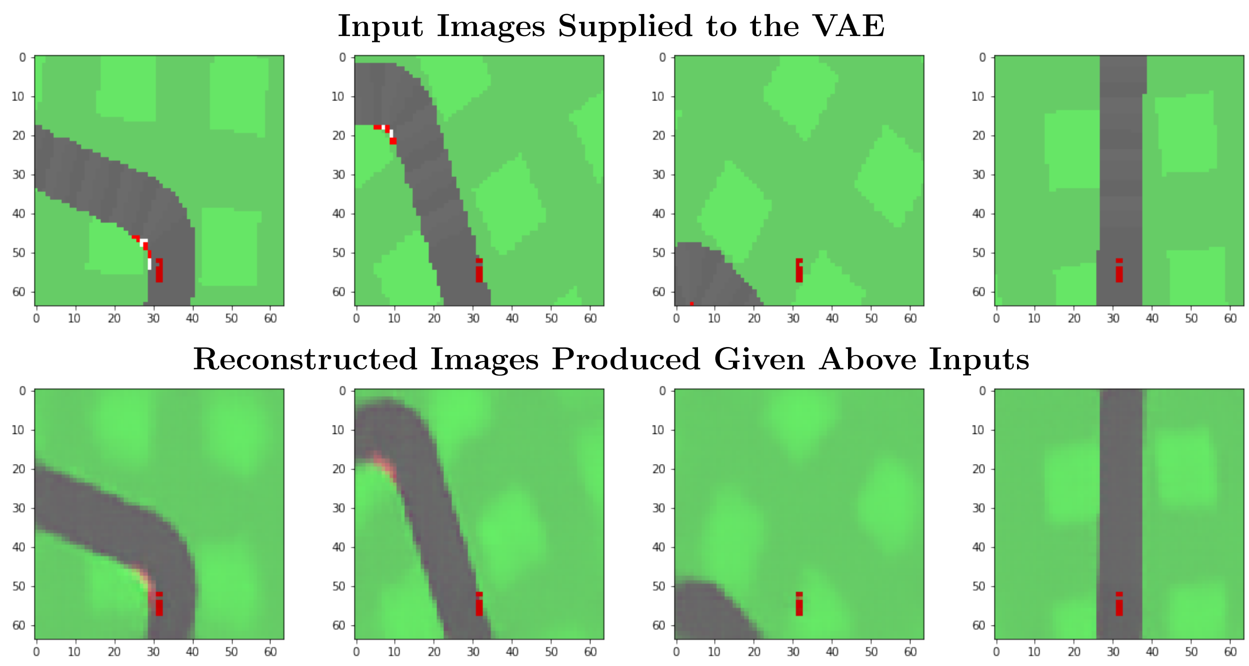


Figure 12: Input images and their latent reconstructions from the VAE.

already possess some level of encoding on proper driving behavior under those conditions.

2.2 Results from Training the RNN

We ran the RNN training for 5 epochs, each with 800 iterations for a total of 4000 training runs. The loss, as described above in section 1.3.2, was calculated using KL Divergence and backpropagation. Figure 13 shows the results of this training.

The success of this training can only be seen in the performance of the model as a whole, but it can be inferred that the model likely is undertrained given the lengths to which both the model in [12] and [31]. As noted in [4], the evaluation of the performance of an MDN-RNN in isolation is still a developing area of research. As such, we can only base the quality of its training by the performance of our agent.

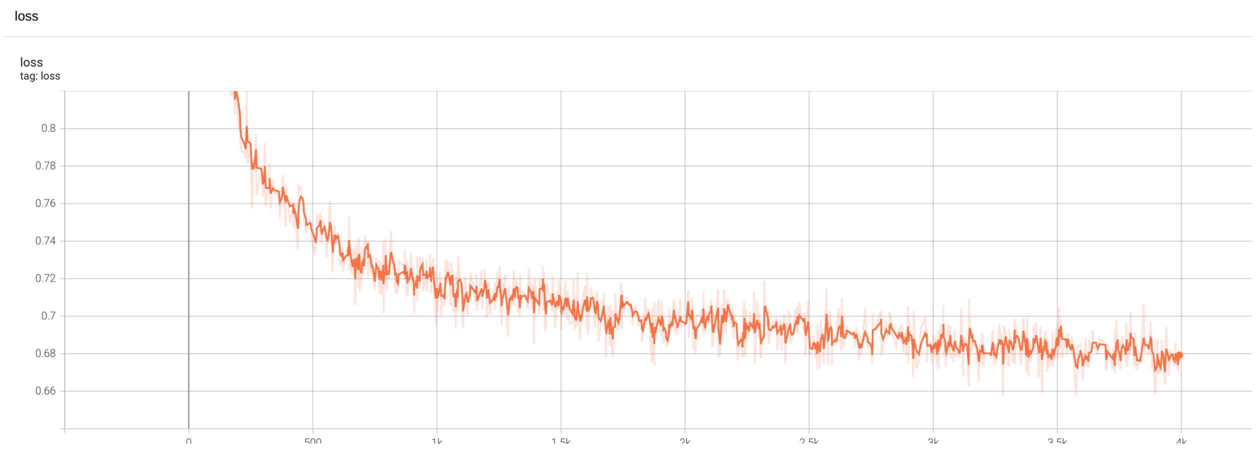


Figure 13: Total loss during RNN Training. The x-axis shows the iterations, with vertical lines representing the division of the epochs. The y-axis shows the log of the loss function over time within a range of approximately [0.60,0.86]. As seen with the VAE loss, the loss begins exponentially large and then converges upon a set value of approximately 0.68.

2.3 Results from Training the Controller

Figure 14 shows a comparison of our trained model, in orange, against those of [12], in green, and [31], in blue. While both earlier models rapidly rise to a high level of performance, ours shows a comparatively low-performing and erratic version of the model. It should be noted that the model does continue to improve its overall average performance, implying that better policies are being learned. Our best reward during the training was 273.

2.3.1 Discussion on Time to Train the Model

The likely cause of the gap in performance between our model and those of [12] and [31] is the limitation on device capability, and–by consequence–time. The hardware used for our replication can support 4 workers at most to train the agent. The original implementation

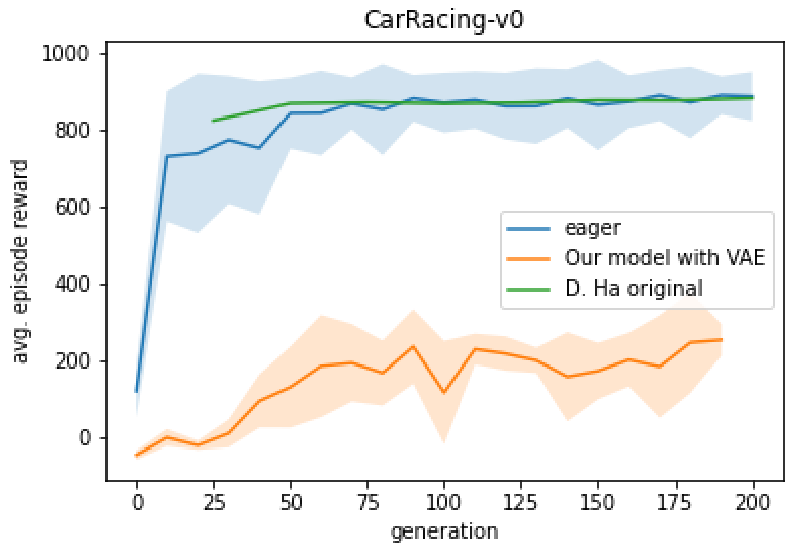


Figure 14: Final results of our implementation. The green line represents the original results from the World Models paper; the blue line represents Zac Wellmer's results; the orange line represents our results.

was built on a machine capable of supporting 64 workers. This means that for every iteration of the original machine, ours would have to run 16 more. The implementation in [31] utilized a similar hardware system, judging by the code inherited from that implementation.

Model	**Training Time**	**Workers**	**Clock Time**
VAE	≈60 hours	1	60 hours
MDN-RNN	≈10 hours	1	10 hours
Controller	47 hours	4	188 hours
Total			258 hours

Figure 15: Training time calculation for our replication attempt.

The impact of this is felt keenly in terms of linear training time (also called 'clock time'). Figure 15 shows the breakdown of training time for our attempt. In total, the training pipeline was allocated 258 hours to train. This is equivalent to $10\frac{2}{3}$ days. Conversely, the replication attempt in [31] quotes that the model was given 48 clock days of training and it was *still not enough* to achieve comparable scores to the original model. This leads to two avenues for future research with our model: attempting to let this model train with few workers and a longer time budget or increasing the hardware capacity of the unit.[6]

Given the above constraints, the results we obtained were promising. We predict that with more time and compute power, we could have been able to gain a more accurate reproduction of scores.

3 Task 3: Improving upon the VAE with a GAN

The biggest difference between the approach taken in World Models and Schmidhuber's earlier work [26] is the addition of the V model to the learning agent. The V model leverages an auto-encoder specialized in the task of compressing a high-dimensional image input down to the latent space vector, z. Some advantages of this approach were discussed in section 1.3.1.

Among the advantages of this approach is the fact that the V model is trained *in isolation* from the other components of the agent system [12]. This suggests that any approach may be used in training the VAE so long as it results in a latent distribution z suitable for sampling by the M and C models. One new approach was provided by [19], in which a generative adversarial network (GAN) was used to optimize the data distribution z. [19] recognized that "a small image translation might result in a large pixel-wise error [which] a human would barely notice" [19]. They observed that many autoencoders (AE) were previously trained using a mean squared error (MSE) loss calculation [19]; some metric

[6]Or, perhaps, both?

must be used to estimate encoder error that is invariant to certain aspects of the output, such as translation.

3.1 The VAE/GAN Architecture

The problem with traditional VAE training is the loss function. Since the goal of the VAE is to accurately reconstruct an image from its latent-space compression, the simplest way to calculate their dissimilarity is on a per-pixel basis. This is typically done using the *mean squared error* function (MSE), written as

$$\mathcal{L} = -MSE(x, \tilde{x}) + prior$$

This has the advantage of being simple to compute and quick to calculate but is not without flaw. In [19], it is mentioned that "element-wise metrics are simple but not very suitable for image data, as they do not model the properties of human visual perception." For example, translation by a single pixel may resut in two images between which the human eye cannot differentiate but yield a significant MSE metric. The GAN architecture provides a means by which to prioritize accurate reconstruction by application of a learned similarity measure and avoids the pitfall of element-wise calculation [19].

The General Adversarial Network

The GAN was first proposed in [9] as framework for training a model for realistic image generation. A GAN consists of two parts: a *generator* and a *discriminator*. The gen-

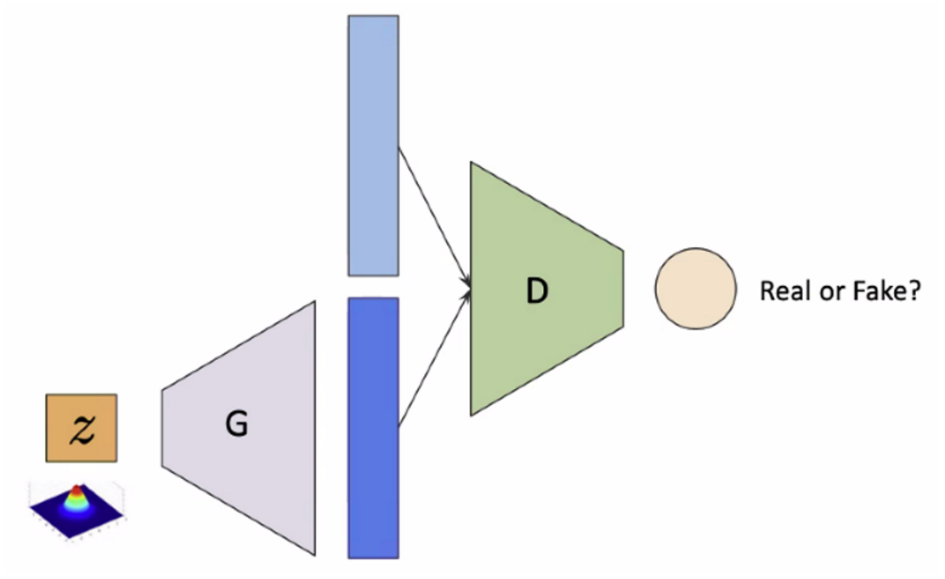


Figure 16: The basic GAN architecture. Figure from [23].

erator, $G()$ as a function, samples from a distribution, usually Gaussian noise, and generates an image. This image is given to the discriminator, $D()$ as a function, is provided with a real sample from the data set and the generated 'imitation' for comparison. The discriminator's task is to decide which image is more likely to have come from the data set [7]. This is shown in Figure 16.

In Figure 16, G represents the generator, which takes in a sampling from the distribution z and provides an image (dark blue) to the discriminator, D. D also takes in a real image sampled from the data set (light blue) and produces the output (yellow circle) of which image is likely to be real or "fake."

The GAN is essentially constructing a min-max game whereby the discrimator attempts to minimize its error in detecting false images and the generator is attempting to maximize the number of mistakes made by the discriminator. In short, we arrive at this

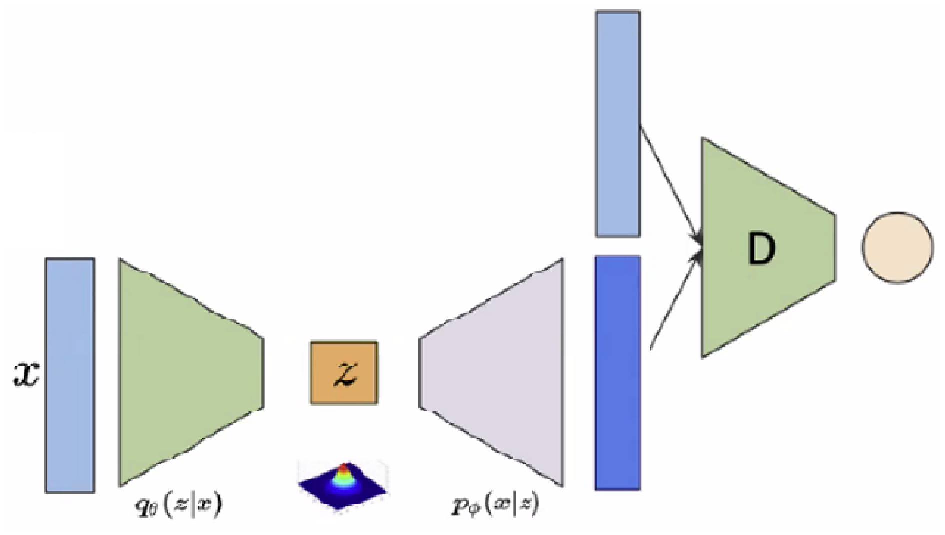


Figure 17: The VAE-GAN architecture. Figure adapted from [23].

equation from [9],

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

where the expected outcome of the model (V) is given by the set of arguments necessary to maximize the equation with respect to the generator, G, and minimize it with respect to the discriminator, D. [9] claims that this is equivalent to the expected value of the log of $D(x)$, where x is a random sample from the dataset p_{data}, plus the expected value of the log of 1 minus the discriminator's evaluation of G given z, a sampling from the noise distribution, p_z.

3.2 The VAE/GAN

Noting that the discriminator function of a GAN has to learn a "rich similarity metric for images", [19] proposes that a discriminator provides a better method for calculating error over the latent distribution z. The architecture they propose is summarized in Figure 17. The VAE (Figure 2, bottom) is now treated as the generator for the network. Provided input x, it produces the reconstructed estimation $\tilde{x}$. The two are compared by the discriminator function and the error propagates back through the the full VAE network [19].

The loss for the entire system is now calculated by a three-part equation

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN}$$

where the crucial term for training the VAE component is $\mathcal{L}_{GAN}$. This term unpacks into an equation reminiscent of Goodfellow's min-max game:

$$\mathcal{L}_{GAN} = \log(Dis(\mathbf{x})) + \log(1 - Dis(Dec(\mathbf{z}))) + \log(1 - Dis(Dec(Enc(\mathbf{x}))))$$

Dis = discriminator; Dec = decoder; Enc = encoder

In effect, we are now comparing *three* images: the original image, its reconstruction from the VAE, and a random sampling from the distribution z.

3.3 The Benefits Over A Basic VAE

As was previously mentioned, element-wise metrics are not suitable for image data - generative models trained with learned similarity measures will produce better image samples than those trained with element-wise error measures [19]. An appealing property of a GAN is that the discriminator network implicitly has to learn a rich similarity metric; this can be exploited to a more abstract reconstruction error for the VAE [19]. The end result is a method that combines the advantages of GAN as a high quality generative model and VAE as a method that produces an encoder of data into the latent space z. A plain VAE struggles when the data moves too far away from the front-facing alignments (specific to the experiments in [19]), because the pixel correspondence cannot be assumed. In comparison, a VAE/GAN produces sharper images with more natural textures [19]. Overall, compared to a plain VAE, the VAE/GAN model yields significantly better attributes that lead to smaller recognition error [19].

It may also be pointed out that the *World Model* architecture does not use a pure VAE, opting for a ConnVAE, incorporating several convolutional layers as the down-sampling method to construct the vector z. There is a mystique of *feature invariance* or *equivariance* around convolutional networks which imply that this model may already account for some failings of the VAE when it comes to transnational input properties. However, many of the architectures that have proven feature invariance have relied upon *pooling* [8], [32]. At first glance, as *World Models* does not utilize pooling, feature variance between frames may result in radically different outputs. This may explain some of the continuity jumps in contiguous images reconstructed in the dreamscape. As stated in [19], feature invariance was a motivating factor behind the VAE/GAN creation. Therefore, the output sequence is likely to have fewer continuity jumps (especially near curves).

3.4 Results from VAE-GAN on World Models '*carracing*'

With the above possible improvements upon the vanilla VAE in mind, we implemented a new training model for the V Model encoder using a VAE-GAN. To this end, we adapted a VAE-GAN model from [13]. One advantage of chosing this model is that it is already constructed to work with input images of the same dimension as the source frames in the '*car racing*' experiment (64x64x3). The model is in a well-known public repository, and has received sufficient traction to have several issues resolved from its initial release. This allowed us to spend less time fixing bugs and instead focus on optimizing the model via training.

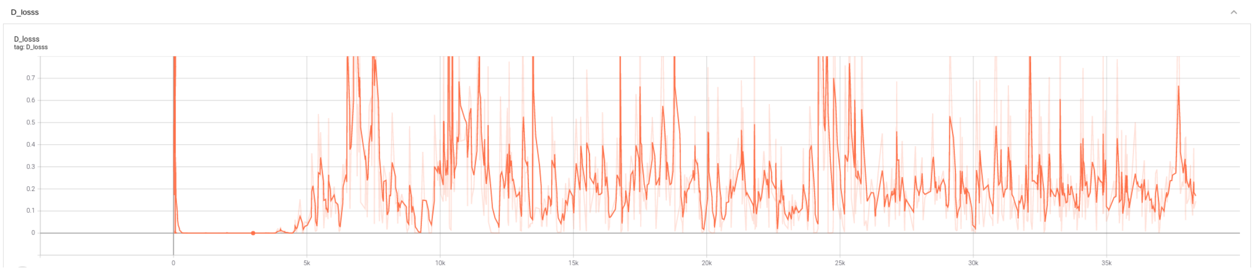


Figure 18: VAE/GAN loss with respect to the Discriminator.

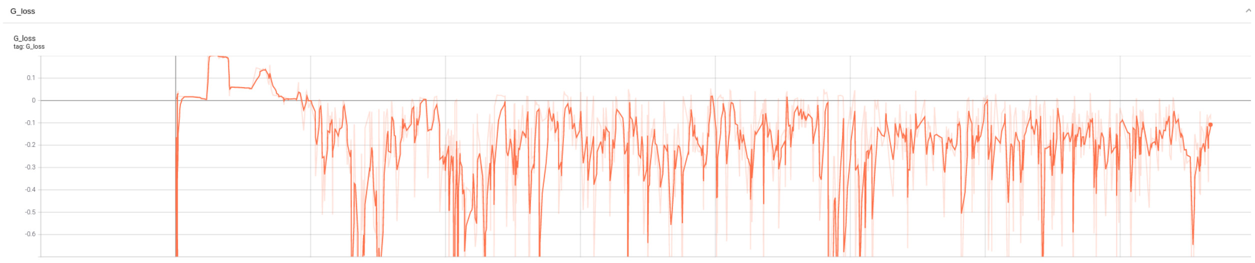


Figure 19: VAE/GAN loss with respect to the Generator.

The model was trained for 7 epochs with 5000 iterations each. This totals to 35,000 iterations, nearly 4 times as many as our plain VAE model. The loss is calculated here entirely by $\mathcal{L}_{KL}$ but propagates backwards with respect to three sources: the discriminator,

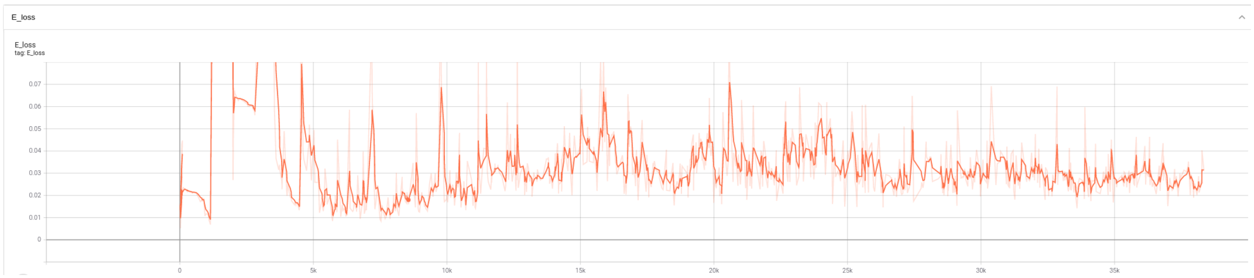


Figure 20: VAE/GAN loss with respect to the Encoder.

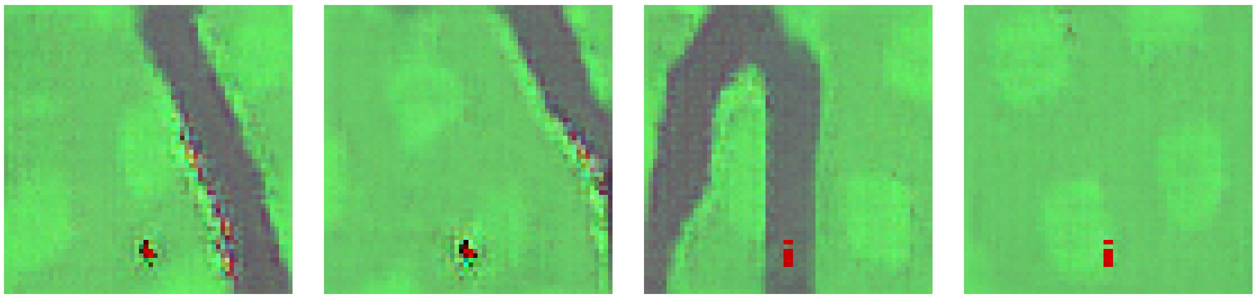


Figure 21: Latent reconstructions sampled from the latent distribution of the VAE-GAN

the generator and the encoder. These are shown in Figures 18, 19 and 20. In each figure, the loss is much noisier than in the original VAE. The loss seems to oscillate around a medial value without converging upon it over time.

As with the original VAE, we can compare the results of the encoding process visually examining the reconstructed images, now sampled from the latent distribution. Figure 21 shows the results of this operation on four random samplings from the training set.

As can be seen, the random frames contain all important aspects of the scene. One critical point raised in [19] is the shift from an "ego-centric" generative bias with a pure VAE that is avoided with the VAE-GAN approach. This is evidenced in the left two images as the car appears to be in the correct position within the scene, but does undergoes some shape decomposition. This shows that the frame as a whole is modeled by the distribution without a self-centered bias with the model. In comparison to the pure VAE trained for our replication, the road is less cleanly defined. This may be due to internal parameters which deserve further exploration. In a visual comparison with the output from the model trained in [12], the VAE-GAN output approximates some of the "noisier" samplings from their dream-state distribution.

3.4.1 Agent Performance with a VAE-GAN

Agent performance was expected to improve based on previous findings in [16], [19]. However, due to time constraints we were unable to resolve all of the issues in the code to effectively implement the VAE-GAN and produce sufficient results. That said, the current state of the code is promising and with extra time to work through bugs, we believe the VAE-GAN would indeed achieve scores equal to or better than the original implementation.

References

[1] Samuel Alvernaz and Julian Togelius. "Autoencoder-augmented neuroevolution for visual doom playing". In: *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017* (July 2017), pp. 1–8. DOI: 10.1109/CIG.2017.8080408. URL: http://arxiv.org/abs/1707.03902.

[2] H. Bourlard and Y. Kamp. *Auto-association by multilayer perceptrons and singular value decomposition.* Tech. rep. 4-5. 1988, pp. 291–294. DOI: 10.1007/BF00332918.

[3] Nathaniel D. Daw, Yael Niv, and Peter Dayan. "Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control". In: *Nature Neuroscience* 8.12 (Dec. 2005), pp. 1704–1711. ISSN: 10976256. DOI: `10.1038/nn1560`.

[4] Kai Olav Ellefsen, Charles Patrick Martin, and Jim Torresen. *How do mixture density RNNs predict the future?* 2019.

[5] Michael J. Frank and Eric D. Claus. "Anatomy of a decision: Striato-orbitofrontal interactions in reinforcement learning, decision making, and reversal". In: *Psychological Review* 113.2 (Apr. 2006), pp. 300–326. ISSN: 0033295X. DOI: `10.1037/0033-295X.113.2.300`.

[6] Michael J. Frank, Lauren C. Seeberger, and Randall C. O'Reilly. "By carrot or by stick: Cognitive reinforcement learning in Parkinsonism". In: *Science* 306.5703 (Dec. 2004), pp. 1940–1943. ISSN: 00368075. DOI: `10.1126/science.1102941`.

[7] Ian Goodfellow. *NIPS 2016 tutorial: Generative adversarial networks.* Tech. rep. 2017. URL: `http://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf`.

[8] Ian Goodfellow, Yo Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. ISBN: 3463353563306. URL: `http://www.deeplearningbook.org`.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2014), pp. 139–144. ISSN: 15577317. DOI: `10.1145/3422622`. URL: `http://www.github.com/goodfeli/adversarial`.

[10] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: (2014). URL: `http://arxiv.org/abs/1308.0850`.

[11] David Ha and Douglas Eck. *A neural representation of sketch drawings*. 2017. URL: `https://magenta.tensorflow.org/sketch_rnn.`.

[12] David Ha and Jurgen Schmidhuber. *World models*. 2018. DOI: `10.1016/b978-0-12-295180-0.50030-6`. URL: `https://worldmodels.github.io/%20https://worldmodels.github.io`.

[13] Leo Heidel. *leoHeidel/vae-gan-tf2: Tensorflow 2 implementation of VEA-GAN*. 2020. URL: `https://github.com/leoHeidel/vae-gan-tf2`.

[14] S Hochreiter and J Schmidhuber. *Long Short-Term Memory*. Tech. rep. 1997, pp. 1735–1780. URL: `http://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf`.

[15] Gerhard Jocham, Tilmann A. Klein, and Markus Ullsperger. "Dopamine-mediated reinforcement learning signals in the striatum and ventromedial prefrontal cortex underlie value-based choices". In: *Journal of Neuroscience* 31.5 (Feb. 2011), pp. 1606–1613. ISSN: 02706474. DOI: `10.1523/JNEUROSCI.3904-10.2011`.

[16] Diederik P. Kingma and Max Welling. *Auto-encoding variational bayes*. Tech. rep. 2014.

[17] Diederik P. Kingma and Max Welling. *An introduction to variational autoencoders*. 2019. DOI: `10.1561/2200000056`.

[18] Will Knight. *AI Songsmith Cranks Out Surprisingly Catchy Tunes*. 2016. URL: `https://www.technologyreview.com/s/603003/ai-songsmith-cranks-out-surprisingly-catchy-tunes/`.

[19] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, et al. *Autoencoding beyond pixels using a learned similarity metric*. Tech. rep. 2015. URL: http://arxiv.org/abs/1512.09300.

[20] Yann LeCun, Bernhard Boser, John Denker, et al. *Handwritten digit recognition with a back-propagation network*. Tech. rep. 1990, pp. 396–404.

[21] Marvin Minsky. *Steps Toward Artificial Intelligence*. Tech. rep. 1. 1961, pp. 8–30. DOI: 10.1109/JRPROC.1961.287775.

[22] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, et al. *Curiosity-driven exploration by self-supervised prediction*. Tech. rep. 2017, pp. 4261–4270. URL: http://pathak22..

[23] Mengwei Ren. "Deep Learning & Computer Vision: Deep Generative Models with emphasis on VAEs and GANs". In: *CS-GY 6643 Computer Vision Course Lecture*. New York, NY: Tandon School of Engineering, New York Unversity, 2021.

[24] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th editio. Boston: Pearson, 2020, p. 10.

[25] samim. *Obama-RNN — Machine generated political speeches*. 2015. URL: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0#.yrjj8mfy9%5Cnhttps://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0#.m5etwmf8q.

[26] Juergen Schmidhuber. *On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models*. Tech. rep. Manno-Lugano: The Swiss AI Lab, 2015, p. 36. URL: http://arxiv.org/abs/1511.09249.

[27] Jürgen Schmidhuber. "Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 5499 LNAI. 2009, pp. 48–76. ISBN: 3642025641. DOI: `10.1007/978-3-642-02565-5{\_}4`. URL: `http://www.idsia.ch/%E2%88%BCjuergen`.

[28] Jürgen Schmidhuber. *Deep Learning in neural networks: An overview.* Apr. 2015. DOI: `10.1016/j.neunet.2014.09.003`. URL: `http://arxiv.org/abs/1404.7828%20http://dx.doi.org/10.1016/j.neunet.2014.09.003`.

[29] Ehsan Variani, Erik McDermott, and Georg Heigold. *A Gaussian Mixture Model layer jointly optimized with discriminative features within a Deep Neural Network architecture.* Tech. rep. 2015, pp. 4270–4274. DOI: `10.1109/ICASSP.2015.7178776`.

[30] Cinzia Viroli and Geoffrey J. McLachlan. "Deep Gaussian mixture models". In: *Statistics and Computing* 29.1 (Nov. 2019), pp. 43–51. ISSN: 15731375. DOI: `10.1007/s11222-017-9793-z`. URL: `http://arxiv.org/abs/1711.06929`.

[31] Zac Wellmer. *GitHub - zacwellmer/WorldModels: World Models with TensorFlow 2.* 2020. URL: `https://github.com/zacwellmer/WorldModels`.

[32] Richard Zhang. *Making convolutional networks shift-invariant again.* 2019. URL: `https://richzhang.github.io/antialiased-cnns/.`.