Deep Neuroevolution of Recurrent and Discrete World Models

Sebastian Risi and Kenneth O. Stanley
Uber AI
San Francisco, CA 94103
sebastian.risi@gmail.com, kstanley@uber.com

ABSTRACT

Neural architectures inspired by our own human cognitive system, such as the recently introduced world models, have been shown to outperform traditional deep reinforcement learning (RL) methods in a variety of different domains. Instead of the relatively simple architectures employed in most RL experiments, world models rely on multiple different neural components that are responsible for visual information processing, memory, and decision-making. However, so far the components of these models have to be trained separately and through a variety of specialized training methods. This paper demonstrates the surprising finding that models with the same precise parts can be instead efficiently trained *end-to-end* through a genetic algorithm (GA), reaching a comparable performance to the original world model by solving a challenging car racing task. An analysis of the evolved visual and memory system indicates that they include a similar effective representation to the system trained through gradient descent. Additionally, in contrast to gradient descent methods that struggle with discrete variables, GAs also work directly with such representations, opening up opportunities for classical planning in latent space. This paper adds additional evidence on the effectiveness of deep neuroevolution for tasks that require the intricate orchestration of multiple components in complex heterogeneous architectures.

ACM Reference Format:
Sebastian Risi and Kenneth O. Stanley. 2019. Deep Neuroevolution of Recurrent and Discrete World Models. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3321707.3321817

1 INTRODUCTION

Neuroevolution, i.e. evolving neural networks through evolutionary algorithms, has long been applied to complex control problems [7, 32, 39] and has recently been shown to be a competitive alternative for reinforcement learning problems [34, 40]. Surprisingly, Such et al. [40] demonstrated that even a simple genetic algorithm (GA) is able to optimize a large-scale deep network to play various Atari games from raw pixels.

However, while the aforementioned deep networks have large numbers of parameters, their architectures are often relatively simple feed-forward, directly mapping high-dimensional inputs to the network's outputs [40]. It is therefore an open question how genetic algorithms would scale to problems that require more complex architectures with multiple different and interacting components.

One such neural network-based architecture, which is inspired by the human cognitive system, is the world model recently introduced by Ha and Schmidhuber [13]. This agent model contains three different components: a visual module that maps high-dimensional inputs to a lower-dimensional representative code, a memory component that tries to predict the future based on past experience, and a decision-making module that determines the action of the agent based on inputs from the visual and memory module.

The world model is motivated by the insight that our brains learn abstract representations of both spatial and temporal data, allowing us to generalize to different situations and to predict potential future sensory experiences. Because of its predictive abilities, the world model approach is able to find a solution for a challenging 2-D car racing task (defined as reaching a minimum average reward of 900 over 100 consecutive trials), a domain that other deep RL methods such as Q-Learning and A3C [16, 19] struggle with so far. However, the approach requires each of its three components to be trained separately and through specialized training methods. While the controller part is trained through an evolution strategy, both the visual and memory components are trained through stochastic gradient descent based on random rollouts. Given the surprising and competitive results of GAs on RL problems [40], the question in this paper is whether a simple GA might also be competitive with complex heterogeneous systems like world models, and if so, what type of representation would evolve.

As the results in this paper on a 2-D car racing domain demonstrate, it is in fact possible to train a complex multi-component system end-to-end with a simple genetic algorithm. Indeed, the GA performs comparably to the world model approach and finds a solution to the task, outperforming all of the other traditional deep RL methods. Surprisingly, even though the sensory component was not directly trained to compress similar sensory states to similar latent codes (as is the case in the training of the autoencoder in Ha and Schmidhuber [13]), the GA discovers such a representation by itself because it is beneficial for solving the task. Similarly, the emergent representation of the memory system is able to predict situations in which the agent needs to react quickly to changes in the environment, such as when taking sharp turns.

Additionally, this paper introduces a discrete world model approach, in which the VAE is restricted to binary outputs. While traditional machine learning techniques have focused on continuous representations because backpropagating through discrete variables is challenging [3, 30, 33, 45], evolutionary-based approaches

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321817

do not struggle with discrete representations. In the future, such representations could directly support classical planning approaches in latent space [2].

Overall, the performance of the GA for evolving the weights of more complex architectures suggests that it can be a competitive alternative in many tasks that were thought too high-dimensional for artificial evolution. In the future, it will be interesting to extend this approach to not only evolving the network's weights but also the architectures of the world models themselves.

2 RELATED WORK

A variety of different RL algorithms have recently been shown to work well on a diverse set of problems when combined with the representative power of deep neural networks [26, 36, 37]. While most approaches are based on variations of Q-learning [26] or policy gradient methods [36, 37], recently evolutionary-based methods have emerged as a competitive alternative [34, 40].

Salimans et al. [34] showed that a type of evolution strategy (ES) can reach competitive performance in the Atari benchmark and at controlling robots in MuJoCo. Additionally, Such et al. [40] demonstrated that a simple genetic algorithm is in fact able to reach similar performance to deep RL methods such as DQN or A3C. Earlier approaches that evolved neural networks for RL tasks worked well in complex RL tasks with lower-dimensional input spaces [7, 32, 39] and also showed promise in directly learning from high-dimensional input [21].

However, when trained end-to-end these networks are often still orders of magnitude simpler than networks employed for supervised learning problems [17] or depend on additional losses that are responsible for training certain parts of the network [48].

More complex agent models often require training different network components separately [13, 46]. For example, in the world model approach [13], the authors first train a variational autoencoder (VAE) on 10,000 rollouts from a random policy to compress the high-dimensional sensory data and then train a recurrent network to predict the next latent code. Only after this process is a smaller controller network trained to perform the actual task, taking information from both the VAE and recurrent network as input to determine the action the agent should perform.

In another earlier related approach the authors first train an autoencoder in an unsupervised way [1] or train an object recognizer in a supervised way [28] and then in a separate step evolve a controller module. The idea in the present paper is to explore whether a GA can optimize a multi-component system end-to-end without the need to separate training into different phases, which is explained in the next section.

Approaches to learning dynamical models have mainly focused on gradient descent-based methods, with early work on RNNs in the 1990s [35]. More recent work includes PILCO [6], which is a probabilistic model-based policy search method and Black-DROPS [4], which employs CMA-ES for data-efficient optimization of complex control problems. Additionally, interest has increased in learning dynamical models directly from high-dimensional pixel images for robotic tasks [47] and also video games [11]. Work on evolving forward models has mainly focused on neural networks that contain orders of magnitude fewer connections and lower-dimensional feature vectors [27] than the models in this paper.

3 END-TO-END TRAINING OF MULTI-COMPONENT NETWORKS

The agent model in this paper is based on the world model approach introduced by Ha and Schmidhuber [13]. The network includes a sensory component, implemented as a variational autoencoder (VAE) that compresses the high-dimensional sensory information into a smaller 32-dimensional representative code (Figure 1). This code is fed into a memory component based on a recurrent LSTM [15], which should predict future representative codes based on previous information. Both the output from the sensory component and the memory component are then fed into a controller that decides on the action the agent should take at each time step.

Following Such et al. [40], the deep neural networks are evolved with a simple genetic algorithm, in which mutations add Gaussian noise to the parameter vectors of the networks. Three different mutation operators are investigated:

- In the first approach (**MUT-ALL**), we apply additive Gaussian noise to the parameter vectors of all three modules (vision, memory, and controller) at the same time: $\theta' = \theta + \sigma\epsilon$, where $\epsilon \sim N(0, I)$ and σ was determined empirically and set to 0.01 for the experiments in this paper.
- In the second module mutation setup (**MUT-MOD**), a mutation has an equal probability to either mutate the visual, memory, or controller component of the network. The hypothesis is that this treatment should allow evolution to better fine-tune each component in the system than an approach that always adds Gaussian noise to all components.
- To elucidate the advantages of evolving both the VAE and memory component, their weights are randomly chosen in the third setup (**MUT-C**) and only the controller component is modified through evolution.

In the original world model approach the visual and memory component were trained separately and through unsupervised learning based on data from random rollouts. Here they are optimized through a simple genetic algorithm and the components are not evaluated individually. In other words, the VAE is not directly optimized to reconstruct the original input data and neither is the memory component optimized to predict the next time step; the whole network is trained in an end-to-end fashion and has to learn a representation by itself that allows it to solve the given task.

Another potential benefit of GAs, beyond being able to train the whole system end-to-end, is that training discrete VAEs, in which the latent code takes on only binary values, are seamlessly supported. While learning representations with continuous features have been the focus in machine learning, discrete VAEs can have benefits for domains that are composed of discrete elements (such as language) or can naturally support classical planning approach in latent space [2]. However, discrete VAEs have proven difficult to train through gradient descent-based methods [3, 30] or require a more complicated training procedure [33, 45], because backpropagating through discrete variables is not directly possible. This paper tests the idea of evolving discrete VAEs for the car racing domain. The **DISCRETE-MOD** approach feeds the original output of the

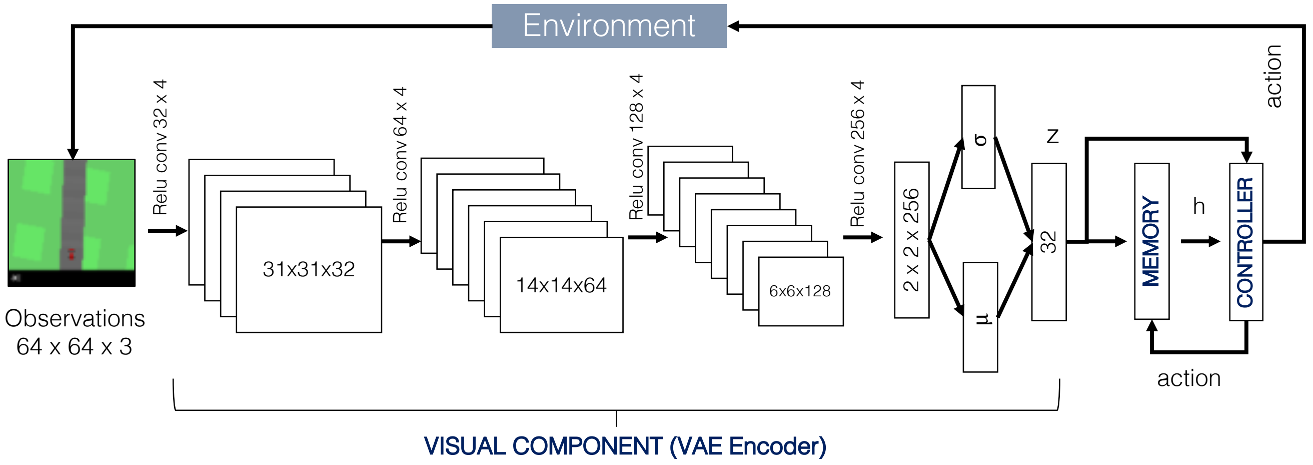
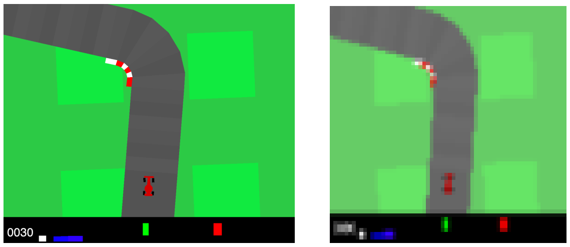


Figure 1: Agent Model. The agent model consists of three modules. A visual component (the encoder of a variational autoencoder) produces a latent code z_t at each time step t, which is concatenated with the hidden state h_t of the LSTM-based memory component that takes z_t and previous performed action a_{t-1} as input. The combined vector (z_t, h_t) is input into the controller component to determine the next action of the agent. In this paper, the agent model is trained end-to-end with a genetic algorithm.



(a) Original **(b) Scaled** ($64 \times 64 \times 3$)

Figure 2: Car Racing Domain. In the car racing domain the agent has to learn to drive across many procedurally generated tracks as fast as possible from 64×64 RGB color images.

VAE encoder through a step function that maps the continuous outputs to binary values.

4 EXPERIMENT

Following the original world model approach [13], in the experiments in this paper an agent is trained to solve a challenging 2-D car racing tasks from 64×64 RGB pixel inputs (Figure 2). In this continuous control task `CarRacing-v0` [20] the agent is presented with a new procedurally generated track every episode, receiving a reward of -0.1 every frame and a reward of +100/N for each visited track tile, where N is the total number of tiles in the track. The network controlling the agent (Figure 1) has three outputs to control left/right steering, acceleration and braking. Further details on the network model can be found in Section 4.1.

Training agents in procedurally generated environments [44], instead of only a particular one, can significantly increase their generality in a variety of different domains and prevent overfitting [5, 18, 49]. Because each agent is tested on a new randomly created track each evaluation, we evaluate the top three individuals of each generation 20 times and average the results to get a better estimate of the true elite (which gets assigned a fitness of ∞). Individuals for the next generation are composed of the 50% highest performing individuals in the current generation plus their offspring determined stochastically through 2-way tournament selection. No crossover operation was employed. To reduce the computational resources spent on non-promising individuals, an evaluation is terminated early in the experiments reported here if an agent is not able to reach an unvisited track tile in 20 time steps.

Following Ha and Schmidhuber [13], after training the champions found in each generation are evaluated on 100 randomly created tracks to estimate their generalization abilities. While it is not difficult to learn to drive slowly around a track, it is challenging to find a solution that can drive around any given track perfectly and as fast as possible. In fact, many traditional deep RL methods [16, 19], which additionally also require pre-processing such as edge detection [16] or stacking recent frames [16, 19], fail to reach high scores on this task (also see Table 2). Interestingly, Ha and Schmidhuber showed that a world model without the recurrent memory model receives a significantly lower score in this domain (decreasing from an average of 906±21 to 788±141), displaying more unstable driving behaviors. This result suggests the importance of a memory model in predicting potential futures that allow the agent to take sharp corners seamlessly. An interesting question is whether evolution will discover such dynamics by itself without being explicitly rewarded to doing so.

4.1 Experimental Setup and Model Details

The size of each population is 200 and evolutionary runs have a termination criterion of 1,000 generations. An overview of the agent model is shown in Figure 1, which employs the same architecture as the original world model approach [13]. The sensory model is implemented as a variational autoencoder that compresses the high-dimensional input to a latent vector z. The VAE takes as input

Table 1: Number of parameters and training procedures. The visual component of the agent (see Figure 1) is effectively only utilizing and evolving the encoder part of the VAE, which has 755,744 parameters. The decoder network is composed of four deconvolutional layers and has 3,592,803 parameters.

Model	**#Params**	**WM Training** [13]	**GA Training**
VAE	4,348,547	SGD - 1 epoch	Pop size 200 Rollouts 1 Solved: 1,200
MD-RNN	384,071	SGD - 20 epochs	
Controller	867	CMA-ES - Pop 64 Rollouts 16 Solved: 1,800 Gen.	

an RGB image of size 64 × 64 × 3, which is passed through four convolutional layers, all with stride 2. Details on the encoder are depicted in the visual component shown in Figure 1, where layer details are shown as: activation type (e.g. ReLU), number of output channels × filter size. The decoder, which is in effect only used to analyze the evolved visual representation in Section 5.1, takes as input a tensor of size 1 × 1 × 104 and processes it through four deconvolutional layers each with stride 2 and sizes of 128 × 5, 64 × 5, 32 × 6, and 32 × 6. The network's weights are set using the default PyTorch initilisation (He initialisation [14]), with the resulting tensor being sampled from $\mathcal{U}(-\text{bound}, \text{bound})$, where $\text{bound} = \sqrt{\frac{1}{\text{fan\_in}}}$.

The memory model [13] combines a recurrent LSTM network with a mixture density Gaussian model as network outputs, known as a MDN-RNN [9, 12]. The network has 256 hidden nodes and models $P(z_{t+1}|a_t, z_t, h_t)$, where a_t is the action taken by the agent at time t and h_t is the hidden state of the recurrent network. Similar models have previously been used for generating sequences of sketches [12] and handwriting [10]. The controller component is a simple linear model that directly maps z_t and h_t to actions: $a_t = W_c[z_t h_t] + b_c$, where W_c and b_c are weight matrix and bias vector. Table 1 summarizes the parameter counts of the different world model components and how they are trained here and in the world model paper. The code for the experiments in this paper can be found at: https://github.com/sebastianrisi/ga-world-models. It is build upon a PyTorch reimplementation of the world model paper by Tallec et al. [42].

5 RESULTS

Figure 3 shows the performance of each treatment for three independent evolutionary runs. Each evolutionary run took approximately two days to train on a 32-core CPU machine. Mutating either every parameter in the network or only targeting specific modules does not result in large changes although there are some notable differences. While MUT-ALL initially increases faster than MUT-MOD, all three runs of the latter ultimately reach a higher performance than any of the MUT-ALL runs. This result suggests that it can initially be beneficial to change many parameters at the same time to get a rudimentary behaviour but fine-tuning them is easier with a mutation operator that only changes one module at a time. The discrete VAE version DISCRETE-MOD also reaches a similar performance to the other methods, confirming the hypothesis that a GA can seamlessly learn a discrete representation that is useful for the task at hand. The results also demonstrate that only mutating the controller part of the neural architecture (MUT-C) and relying on the features produced by a randomly initialized VAE and MDN-RNN are not enough to allow the agent to learn to drive.

Table 2: `CarRacing-v0` scores of different approaches. Only the original world model and the GA approach introduced in this paper are able to solve the task (reaching an average score over 900).

Method	Average Score
DQN [29]	343 ± 18
DQN + Dropout [8]	893 ± 41
A3C (Continious) [16]	591± 45
A3C (Discrete) [19]	652 ± 10
CEOBILLIONAIRE (Gym leaderboard)	838 ± 11
World model [13]	**906** ± 21
World model with random MDN-RNN [43]	870 ± 120
GA (ours)	**903** ± 72

Interestingly, separate evolutionary runs often follow similar performance curves (which we also observed in other experiments). This behaviour appears very different from the training of networks with orders of magnitude fewer parameters traditionally studied in neuroevolution, which often have a higher variance across runs [7, 32, 39]. Analyzing this phenomenon in more detail is an interesting future research direction that we aim to investigate.

After 1,000 generations the MUT-MOD agents were getting very close to solving the domain, learning to drive around the track very effectively with few errors, with the best network reaching a generalization score of 888 ± 66. Therefore we continued evolution for another 200 generations with a lower mutation rate of 0.003 and evaluated each elite on 40 instead of 20 trials. This approach led to finding a solution to the task that reached a score of 903 ± 72. This average score is comparable to the original world model paper and higher than any traditional RL approaches, which reach scores of around 591 to 893 on average (Table 2). A video of the best agent driving around the track can be found at: https://youtu.be/a-tcsnZe-yE.

5.1 Learned Visual Encoder Representation

Because the visual component in our experiments is not specifically trained to reconstruct the given sensory input, it is interesting to analyze what information is contained in the learned latent vector representation. To analyze this question, the evolved VAE encoder weights of a champion network are kept fixed while a decoder is trained in a unsupervised way to reconstruct data collected from a random policy. The decoder is trained for 100 iterations with the Adam optimization algorithm, a learning rate of 0.0001, using the mean squared distance between the reconstructed image and the input image as loss, in addition to Kullback-Leibler (KL) loss.

Interestingly, while the initial random networks from the first generation do not allow the reconstruction of different track images (Figure 4a), which suggests that the initial random weights fail to capture some important information from the pixel inputs, the latent code of the evolved representation contains enough information for this task (Figure 4b). However, the low reconstruction error

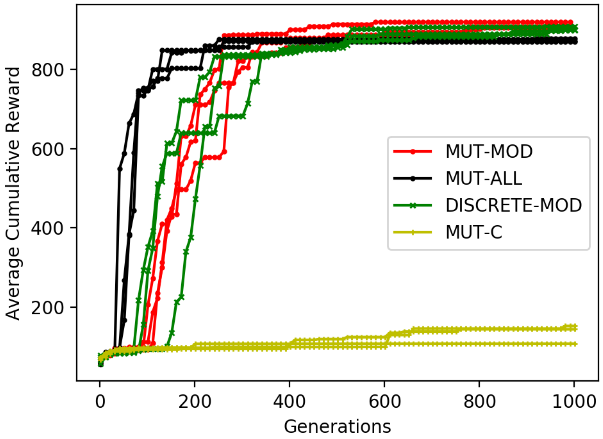
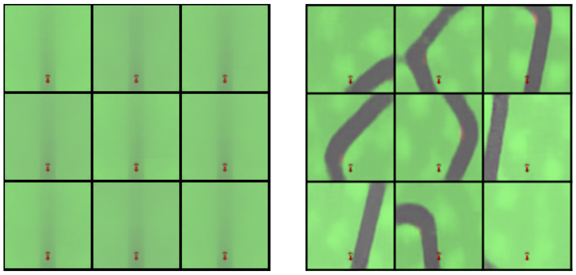


Figure 3: Training performance on `CarRacing-v0`. The score of the best individual is shown in each generation evaluated on 20 randomly created tracks. All approaches, except MUT-C, are able to evolve agent models that can drive around the track at high speed while making very few mistakes.



(a) Random Weights (b) Evolved Weights

Figure 4: VAE Reconstructions. While a network with random encoder weights does not allow the VAE to learn to reconstruct the given images (a), the evolved weights of a champion network encoder enabled different road images to be reconstructed (b). These results suggest that evolution discovered how to compress useful information in the latent code produced by the VAE's encoder.

does not directly explain how the evolved network is utilizing this compressed representation to drive efficiently around the track.

To analyze this question further, we employ the t-SNE dimensionality reduction technique [24] to map the sequence of 32-dimensional latent vectors collected during one car racing rollout to two dimensions. t-SNE has already been proven valuable for gaining insight into the workings of deep neural networks [26, 41].

The mapping (Figure 5) suggests that the agent learned to represent situations that require a similar action (e.g. turning sharp left or right) with a similar latent vector. For example, situations in which the agent needs to turn right are represented by similar latent codes, which are clustered together, while latent codes for situations in which the agent needs to drive straight or turn left are part of a different cluster. By learning an abstract, compressed representation of the higher-dimensional pixel inputs, it becomes easier for the controller module to learn the required behaviors.

5.2 Learned Forward Model Dynamics

In addition to the visual encoder representation, it is interesting to investigate the emergent dynamics of the evolved predictive memory component.

Figure 6 visualizes the activation levels of the MDN-RNN while the agent is driving around two tracks. To get a better sense of the dynamics of the system, we are interested in how much the average activation x_t of all 256 hidden nodes at time step t differs from the overall average across all time steps $\bar{X} = \frac{1}{N}\sum_1^N \bar{x_t}$. The variance of $\bar{x_t}$ is thus calculated as $\sigma_t = (\bar{X} - \bar{x_t})^2$, and normalized to the range $[0, 1]$ before plotting. Activation levels far from the mean should have a higher impact on the agent's controller component and can indicate situations in which the agent needs to pay particular attention to the predictions of the MDN-RNN.

The results show that the dynamics of the recurrent network are changing more drastically when the agent is near a corner and change less when the agent is driving on straight track segments. This effect confirms the hypothesis that predicting future sensory states is particularly important during situations in which the agent needs to react quickly to changes in the environment.

6 DISCUSSION AND FUTURE WORK

This paper demonstrated that genetic algorithms can not only train the weights of relatively simple network architectures but also complicated systems with over a million weights that include different components for sensory processing, memory, and decision making in an end-to-end fashion. The approach outperforms standard deep RL approaches and reaches a comparable performance to the recently-introduced world model approach that relies on a much more complex training regimen. Another surprising result is that the GA found a solution with a population size of only 200, compared to the much larger population sizes of 1,000 in the work by Such et al. [40] on Atari video game playing.

The difference between the two mutation treatments MUT-MOD and MUT-ALL also suggests a potentially useful hybrid approach, in which mutations sometimes affect all network layers and sometimes only one layer at a time. Such an approach could combine the better initial exploration of MUT-ALL with the ability of MUT-MOD to better fine-tune different parts of the network in later generations.

While the final average generalization score is slightly lower than the score in the original world model paper [13], the presented system does indeed solve the task while not relying on first learning from a large number of random rollouts; instead the system can learn directly in interaction with the environment. The slightly lower average score (903 compared to 906) with a higher standard deviation (72 compared to 21) could be explained by the fact that if random rollout data is available, training each component separately might produce slightly more robust solutions. However, especially in more complicated tasks for which data collected during random rollouts is insufficient (because a random rollout might not reach all relevant parts of the environment), the end-to-end learning approach could become more important.

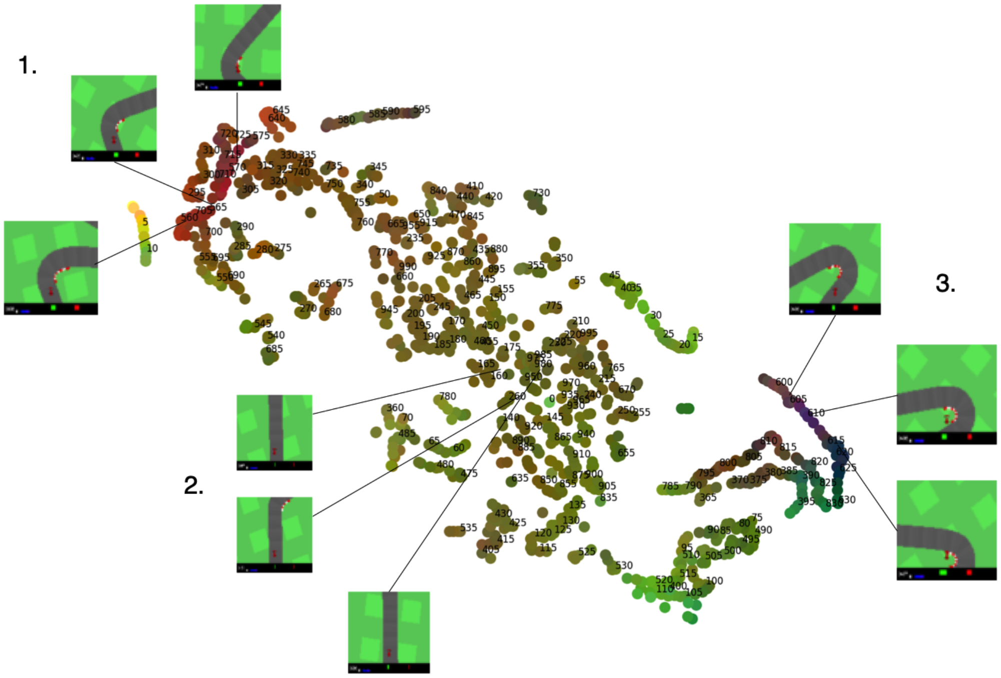


Figure 5: Evolved Visual Representation. t-SNE mapping of the 32-dimensional latent vectors onto two dimensions. The three action outputs of the agent are mapped to the RGB color values of each plot point (R=steer, G=gas, B=break). The GA successfully discovered a visual encoder that maps similar pixel inputs to similar latent codes. Similar latent codes in turn determine similar agent actions, such as turning right (1), driving straight (2), or turning left (3).

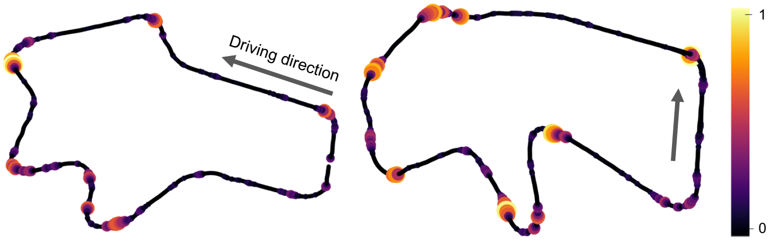


Figure 6: Dynamics of Evolved Forward Model. The size and color of each marker reflects the variance in activation levels of the MDN-RNN's hidden state while the agent is driving around the track (brighter colors and larger marker sizes indicate higher magnitudes). Variance levels are typically higher when the agent is near corners, while they are lower on straight road segments. These results suggest that the evolved agent model is most reliant on the memory component in situations that benefit from predicting future sensory states.

One advantage and indeed motivation of the original world model approach was the fact that agents can train and improve in the environments generated by the world model itself, without using the actual environment. Testing the evolved world model presented in this paper for the same purpose is an important next step. As noted by Ha and Schmidhuber [13], the discrete modes in the mixture density model can be beneficial in environments with random discrete events (e.g. firing of a weapon in an FPS game). They observed that if the temperature parameter that controls the model's uncertainty is set to a very low value, the enemies in the world model of their FPS environment never fire their weapons; the MDN-RNN is not able to reach a mode in the mixture of Gaussian models in which this event happens. In this context, we hypothesise that the ability of the GA to evolve discrete VAE representations (which are fed into the MDN-RNN) could make it even easier for the model to switch between different modes than the current continuous VAE version.

Another exciting prospect is not only to evolve the weights of such large-scale deep networks but also the neural architectures themselves. While evolutionary algorithms have allowed the architectures of relatively simple networks to be evolved for reinforcement learning problems, so far larger-scale architectures have mostly been evolved in combination with supervised learning [25] and not extended to very complex RL problems. Other promising extensions to the simple GA used in this paper could be additional crossover operators, indirect encodings such as HyperNEAT [31, 38], safe mutations [22], or more exploratory search methods such as novelty search [23].

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and David Ha for their insightful comments that very much improved the presentation of this paper. We would also like to thank all of the members of Uber AI for helpful discussions.

REFERENCES

[1] Samuel Alvernaz and Julian Togelius. 2017. Autoencoder-augmented neuroevolution for visual doom playing. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 1–8.

[2] Masataro Asai and Alex Fukunaga. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[4] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. 2017. Black-box data-efficient policy search for robotics. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 51–58.

[5] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2018. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341* (2018).

[6] Marc Deisenroth and Carl E Rasmussen. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 465–472.

[7] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (2008), 47–62.

[8] P. Gerber, J. Guan, E. Nunez, K. Phamdo, T. Monsoor, and N. Malaya. 2018. Solving OpenAI's Car Racing Environment with Deep Reinforcement Learning and Dropout. https://github.com/AMD-RIPS/RL-2018/blob/master/documents/nips/nips_2018.pdf

[9] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[10] Alex Graves. 2013. Hallucination with recurrent neural networks. https://www.youtube.com/watch?v=-yX1SYeDHbg&t=49m33s

[11] Matthew Guzdial, Boyang Li, and Mark O Riedl. 2017. Game Engine Learning from Video.. In *IJCAI*. 3707–3713.

[12] David Ha and Douglas Eck. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017).

[13] David Ha and Jürgen Schmidhuber. 2018. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*. 2455–2467.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] Min J. Jang, S. and C. Lee. 2017. Car racing with A3C. https://www.scribd.com/document/358019044/

[17] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. 2019. Deep learning for video game playing. *To appear in: IEEE Transactions on Games* (2019).

[18] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. 2018. Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation. *NeurIPS 2018 Workshop on Deep Reinforcement Learning* (2018).

[19] M. Khan and O. Elibol. 2018. Car racing using reinforcement learning. https://web.stanford.edu/class/cs221/2017/restricted/p-final/elibol/final.pdf.

[20] Oleg Klimov. 2016. Carracing-v0. https://gym.openai.com/envs/CarRacing-v0/

[21] Jan Koutník, Jürgen Schmidhuber, and Faustino Gomez. 2014. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 541–548.

[22] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O Stanley. 2017. Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients. *arXiv preprint arXiv:1712.06563* (2017).

[23] Joel Lehman and Kenneth O Stanley. 2008. Exploiting open-endedness to solve problems through the search for novelty.. In *ALIFE*. 329–336.

[24] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[25] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 293–312.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[27] Mohammad Sadegh Norouzzadeh and Jeff Clune. 2016. Neuromodulation improves the evolution of forward models. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 157–164.

[28] Andreas Precht Poulsen, Mark Thorhauge, Mikkel Hvilshj Funch, and Sebastian Risi. 2017. DLNE: A hybridization of deep learning and neuroevolution for visual control. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 256–263.

[29] Luc. Prieur. 2017. Deep-Q learning for Box2d racecar RL problem. https://goo.gl/VpDqSw

[30] Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. 2014. Techniques for learning binary stochastic feedforward neural networks. *arXiv preprint arXiv:1406.2989* (2014).

[31] Sebastian Risi and Kenneth O Stanley. 2012. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial life* 18, 4 (2012), 331–363.

[32] Sebastian Risi and Julian Togelius. 2017. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 1 (2017), 25–41.

[33] Jason Tyler Rolfe. 2016. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200* (2016).

[34] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[35] Jürgen Schmidhuber. 1990. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *1990 IJCNN international joint conference on neural networks*. IEEE, 253–258.

[36] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[38] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[39] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[40] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).

[41] Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Ludwig Schubert, Marc Bellemare, Jeff Clune, and Joel Lehman. 2018. An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents. *arXiv preprint arXiv:1812.07069* (2018).

[42] Corentin Tallec, Léonard Blier, and Diviyan Kalainathan. 2018. Reimplementation of World-Models (Ha and Schmidhuber 2018) in pytorch. https://github.com/ctallec/world-models

[43] Corentin Tallec, Léonard Blier, and Diviyan Kalainathan. 2018. Reproducing "World Models" Is training the recurrent network really needed ? https://ctallec.github.io/world-models/

[44] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[45] Aaron van den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*. 6306–6315.

[46] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. 2015. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251* (2015).

[47] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. 2015. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*. 2746–2754.

[48] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. 2018. Unsupervised Predictive Memory in a Goal-Directed Agent. *arXiv preprint arXiv:1803.10760* (2018).

[49] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. 2018. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893* (2018).