Revisiting "Recurrent World Models Facilitate Policy Evolution"[*]

Bernardo Esteves[‡] , **Francisco S. Melo**[†,‡]
[†]INESC-ID, Lisbon, Portugal
[‡]Instituto Superior Técnico
University of Lisbon, Portugal
bernardo.esteves@tecnico.ulisboa.pt, fmelo@inesc-id.pt

Abstract

This paper contributes a detailed analysis of the architecture of Ha and Schmidhuber (2018). The original paper proposes an architecture comprising 3 main components: a "visual" module, a "memory" module; and a controller. As a whole, such architecture performed well in challenging domains. We investigate how each of the aforementioned components contributes individually to the final performance of the system. Our results shed additional light on the role of the different components in the overall behavior of the agent, and illustrate how the different design options affect the behavior of the resulting agent. [1]

1 Introduction

In recent years, reinforcement learning (RL) has been the focus of increasing interest, largely due to the impressive successes in video games (Mnih et al., 2015), classical games (Silver et al., 2018) and control (Lillicrap et al., 2016), among others. RL algorithms can roughly be categorized as *model-free* and *model-based* (Sutton and Barto, 2018). And while most of the aforementioned success stories rely on model-free approaches, it is a well-known fact that model-based approaches typically exhibit better sample efficiency and are more transferable (Wang et al., 2019).

In a recent work, Ha and Schmidhuber propose a model-based approach inspired by human mental models that attained competitive performance in two difficult tasks (Ha and Schmidhuber, 2018). The proposed approach includes two key modules (see Fig. 1): a *visual processing module V* that, in abstract terms, translates the perceptual information of the agent to an internal representation; and a *memory module M* that corresponds to the agent's internal model of the world's dynamics. Based on the information from these two modules,

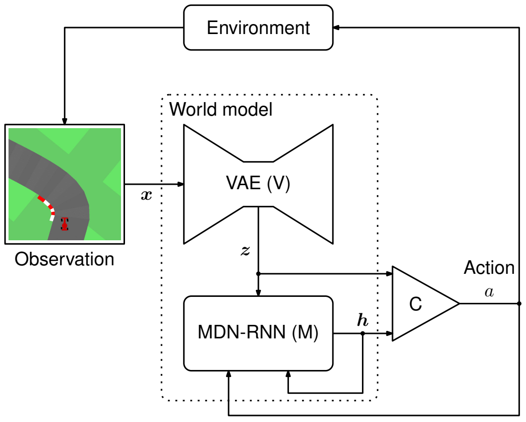

Figure 1: The model of Ha and Schmidhuber (2018) (adapted from the original). The model comprises a world model and a controller C. The world model, in turn, comprises a visual processing component (V) and a memory component (M).

the paper goes on to show that a simple controller is able to attain state-of-the-art results in these two tasks.

In this paper, we depart from the work of Ha and Schmidhuber (2018) and investigate the impact of the different model components in the observed performance of the agent. Our analysis focuses on one of the domains considered in the original paper—the carRacing-v0 domain from Open AI gym (Brockman et al., 2016), depicted in Fig. 1, on the left—and considers how different design choices contributed (positively or negatively) to the results from the original paper.

2 Background

This section provides an overview of the different modules used in the model of Ha and Schmidhuber (2018).

2.1 Variational autoencoders

Variational autoencoders (VAEs) are generative models originally proposed in the work of Kingma and Welling (2014). A VAE is a latent variable model, assuming that the data of interest, hereby denoted as $\mathbf{x}$, can be explained by some set of non-observed (latent) variables $\mathbf{z}$. In other words,

$$p(\mathbf{x}) = \int_{\mathcal{Z}} p_\phi(\mathbf{x} \mid \boldsymbol{z}) p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z},$$

[*]This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019. The first author acknowledges a research grant from Fundação Calouste Gulbenkian under program "Novos Talentos em IA."

[1]Source code is available at https://github.com/esteveste/World-Models-Experiments

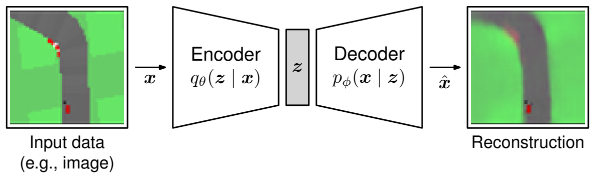

Figure 2: The variational autoencoder (Kingma and Welling, 2014).

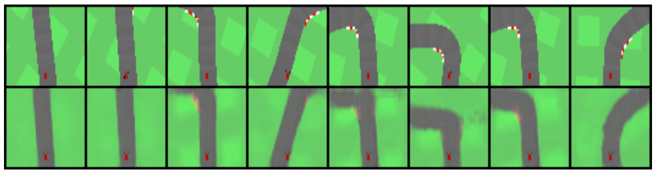

Figure 3: Example reconstruction results using the trained VAE.

where $p(\boldsymbol{z})$ is some prior distribution over the latent variables. Training a VAE thus consists in computing $p_\phi(\boldsymbol{x} \mid \boldsymbol{z})$.

A standard VAE has the structure depicted in Fig. 2 and includes two main parts: an *encoder* and a *decoder*. The encoder is a neural network that, given an input $\boldsymbol{x}$, outputs a distribution $q_\theta(\boldsymbol{z} \mid \boldsymbol{x})$ over the space of latent variables. In a sense, it "encodes" the input $\boldsymbol{x}$ as a low-dimensional representation $\boldsymbol{z}$. The decoder, in turn, is a neural network that, given a latent vector $\boldsymbol{z}$, outputs the distribution $p_\phi(\boldsymbol{x} \mid \boldsymbol{z})$. The, given a dataset $\{\boldsymbol{x}_n, n = 1, \ldots, N\}$, the whole encoder-decoder network is trained to minimize the loss

$$\mathcal{L}(\theta, \phi) = -\sum_{n=1}^{N} \mathbb{E}_{\mathbf{z} \sim q_\theta(\boldsymbol{x}_n)} [\log p_\phi(\boldsymbol{x}_n \mid \mathbf{z})] + \mathrm{KL}[q_\theta(\boldsymbol{x}_n) || p].$$

The first term measures the reconstruction error—the ability of the model to reconstruct $\boldsymbol{x}_n$ given a code $\boldsymbol{z}$ sampled from $q_\theta(\cdot \mid \boldsymbol{x}_n)$. The second works as a regularization term, keeping q_θ as close as possible to the prior p.

Figure 3 illustrates the reconstruction ability of the VAE trained with the data from the carRacing-v0 scenario. In the model of Ha and Schmidhuber (2018), the VAE is responsible for processing the visual inputs $\boldsymbol{x}$ and encoding them as a low-dimensional representation, $\boldsymbol{z}$. Specifically, the output of the encoder is an input-conditioned Gaussian distribution $q_\theta(\cdot \mid \boldsymbol{x})$ over latent variables, where the parameters θ correspond to the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$ of the distribution.

2.2 MDN-RNN

A *mixture density network* (MDN) is a probabilistic model originally proposed in the work of Bishop (1994). In an MDN, we model a conditional distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$ as a *Gaussian mixture*, i.e., we assume that

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{m=1}^{M} \pi_m(\boldsymbol{x}) \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\mu}_m(\boldsymbol{x}), \sigma_m(\boldsymbol{x})),$$

where the scalars π_m are known as the *mixture coefficients*, and $\boldsymbol{\mu}_m$ and σ_m are the parameters of the mth component of the mixture. Then, given a dataset $\{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, \ldots, N\}$, the MDN is trained to minimize the negative log-likelihood of the data.

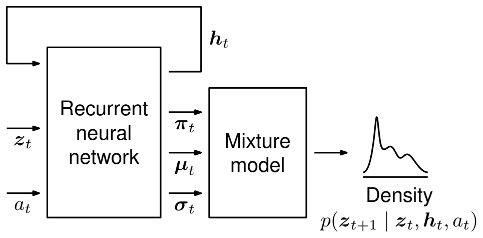

Figure 4: Pictorial representation of the mixture density network (Bishop, 1994) in the model of Ha and Schmidhuber (2018).

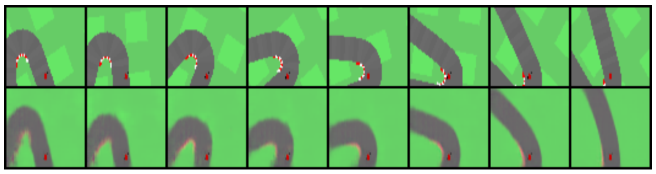

Figure 5: Predictions using the trained RNN. In each column, an input image $\boldsymbol{x}_t$ is passed through the VAE to get a code, $\boldsymbol{z}_t$. The pair $(\boldsymbol{z}_t, a_t)$ is used in the MDN-RNN to generate $\boldsymbol{z}_{t+1}$ which is then turned into an image $\hat{\boldsymbol{x}}_{t+1}$ using the VAE decoder. The top row shows the image $\boldsymbol{x}_{t+1}$, while the bottom row shows $\hat{\boldsymbol{x}}_{t+1}$.

In the model of Ha and Schmidhuber (2018), the MDN takes as input, at each time step, the latent vector $\boldsymbol{z}_t$ coming out of the VAE and the action a_t of the agent, and the output is the next latent vector, $\boldsymbol{z}_{t+1}$. In other words, the MDN learns the distribution $p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t, a_t)$. Since the role of the MDN in the overall model is to capture the temporal dynamics of $\boldsymbol{z}_t$, the neural network in Fig. 4 is actually a *recurrent neural network*—namely an LSTM (Hochreiter and Schmidhuber, 1997)—computing a prediction for the next latent vector, $\boldsymbol{z}_{t+1}$, as a function of the current latent vector, $\boldsymbol{z}_t$, the agent's current action, a_t, and the history up to time t, encoded in the RNNs hidden state $\boldsymbol{h}_t$. A similar MDN-RNN architecture has been previously used for successful sequence modeling and generation (Graves, 2013). Figure 5 illustrates the predictions from the trained MDN-RNN in the carRacing-v0 scenario.

2.3 Controller

The model comprising the VAE and the MDN-RNN network are used to determine the action to be executed by the agent. Namely, the VAE provides in $\boldsymbol{z}_t$ a compact representation of the present perception of the agent, while the MDN-RNN hidden state, $\boldsymbol{h}_t$, provides a compact representation of the *history* of the agent. The two are then used in a controller C to compute the action a_t as

$$a_t = \sigma(\boldsymbol{K} \boldsymbol{f}(\boldsymbol{z}_t, \boldsymbol{h}_t; \boldsymbol{w}) + b), \tag{1}$$

where σ is a squashing function, $\boldsymbol{f}$ is a non-linear function parameterized by $\boldsymbol{w}$, $\boldsymbol{K}$ is a gain matrix and b is a bias term. The parameters $\boldsymbol{w}$, gains and bias are determined using CMA-ES (Hansen and Ostermeier, 2001), an evolutionary approach widely used in RL and robotics (Stulp and Sigaud, 2012).

In the remainder of the paper we investigate the impact of each component discussed above in the overall performance of the agent, as well as that of several design choices:

- In the original paper, the visual model outputs a distribution $q_\theta(\cdot \mid \boldsymbol{x})$, and the latent vector $\boldsymbol{z}$ is sampled from this distribution. We also consider an alternative approach, in which the visual module returns the *mean* of the distribution $q_\theta(\cdot \mid \boldsymbol{x})$. We denote these two alternatives as V_{sampled} and V_{mean}, respectively.
- Tallec et al. (2018) provided empirical evidence that the performance of Ha and Schmidhuber (2018) can be replicated by using an *untrained* MDN-RNN. Therefore, in our analysis, we look at the performance of the agent with trained and untrained memory models (i.e., where the MDN-RNN weights are random). We denote these two alternatives as M_{trained} and $M_{\text{untrained}}$.
- Finally, in terms of the controller, we consider three distinct possibilities, corresponding to increasingly complex controllers. In the simplest controller, the action a_t considers that both σ and $\boldsymbol{f}$ correspond to the identity function. The action is, therefore, an affine function of $\boldsymbol{z}_t$ and $\boldsymbol{h}_t$. The second controller, corresponding to the one of Ha and Schmidhuber (2018), considers only $\boldsymbol{f}$ to be the identity function. Finally, the third controller considers only σ to be the identity function. We denote the three alternatives as C_{linear}, C_{squash} and $C_{\text{nonlinear}}$.

In the continuation, we report the results obtaining by considering different combinations of V, M and C, as described above. Due to space limitations we omit the description of the network architectures used and training methods, and refer to the supplementary material for details. In the upcoming discussion, HS denotes the model of Ha and Schmidhuber (2018), corresponding to $V_{\text{sampled}} + M_{\text{trained}} + C_{\text{squash}}$.

3 Comparative analysis

In this section we report the results of our empirical analysis of the model proposed by Ha and Schmidhuber (2018). We start, in Table 1, by presenting the results of HS and several variations thereof, as well as those of other approaches from the literature. For the sake of comparison, we also present the results of two baselines: the performances of an untrained HS agent and a controller optimized to work directly on images from the game. It is worth mentioning that variation $V_{\text{mean}} + M_{\text{untrained}} + C_{\text{linear}}$ was previously studied by Tallec et al. (2018), while $V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$ was investigated by (Risi and Stanley, 2019). In the continuation, we discuss some additional variations.

3.1 Replicating HS

We started by replicating the results of Ha and Schmidhuber (2018). In the original work, the model was trained for $1,800$ generations. In the results we report, the model was trained for only 140 generations. The differences between our and the original HS implementation are depicted in Fig. 6: our performance is similar to the one reported in the original paper, although exhibiting a larger variance.

3.2 Perceptual model

We compare in Table 2 the performance obtained when using the *mean* of q_θ against that obtained using *samples* of q_θ. Our results clearly show that the use of the mean leads to better performance of the agent, which is to be expected. In fact, considering the mean of q_θ—rather some potentially low-probability sample thereof—will generally improve the agent's ability to select a better action.

We also compare the performance obtained when we change the way the visual information is used—a linear vs a squashed controller C. Interestingly, if q_θ is sampled, C_{squash} leads to better performance, while in the case of the mean of q_θ, C_{linear} performs best. This suggests that the squashing

Table 1: Performance of different approaches in the carRacing-v0 domain. A policy is considered to clear the game if the average score is above 900 (entries in bold).

Method	Avg. Score
Prieur et al.[†]	343 ± 18
Guan et al.[††]	893 ± 41
Jang et al.[‡]	591 ± 45
Gaier and Ha (2019)	893 ± 74
Tang et al. (2020)	**914 ± 15**
CEOBillionaire (gym leaderboard)	838 ± 11
HS (Ha and Schmidhuber, 2018)	**906 ± 21**
$V_{\text{mean}} + M_{\text{untrained}} + C_{\text{linear}}$	852 ± 110
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$	**901 ± 43**
Untrained HS	95 ± 81
Controller on image	735 ± 139

[†] https://tinyurl.com/y73377bq.
[††] https://github.com/AMD-RIPS/RL-2018.
[‡] https://goo.gl/VpDqSw.

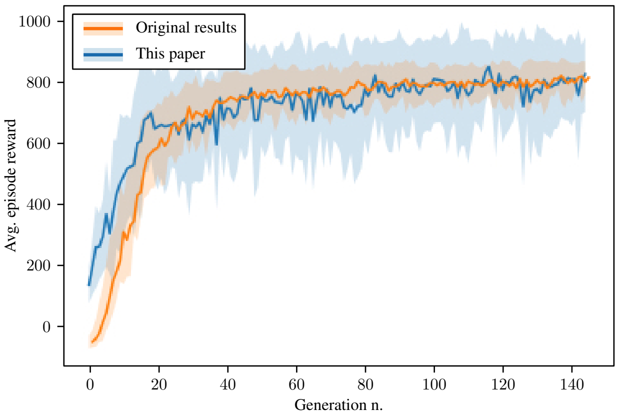

Figure 6: Comparison between the original results of HS and ours.

Table 2: Comparison of the average score of several variations of HS, differing in the visual information available to the agent.

Method	Cropped	Full
HS	853 ± 83	881 ± 40
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{squash}}$	869 ± 86	**900 ± 35**
$V_{\text{sampled}} + M_{\text{trained}} + C_{\text{linear}}$	801 ± 163	835 ± 93
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$	**901 ± 43**	**920 ± 29**

Table 3: Comparison of the average score obtained by removing components from HS.

Method	Avg. Score
$V_{\text{sampled}} + C_{\text{squash}}$	629 ± 102
$V_{\text{mean}} + C_{\text{linear}}$	826 ± 117
$V_{\text{mean}} + C_{\text{nonlinear}}$	872 ± 72
$V_{\text{mean}} + M_{\text{untrained}} + C_{\text{linear}}$	852 ± 110
$M_{\text{trained}} + C_{\text{linear}}$	852 ± 113
$M_{\text{untrained}} + C_{\text{linear}}$	754 ± 143

Table 4: Comparison of the average score obtained by considering a visual module that processes a stacked sequence of 4 frames, instead of a single frame. $V_{\text{mean, k-skip}}$ indicates the 4 stacked frames are obtained by skipping k frames in the real game.

Method	Avg. Score
$V_{\text{mean, 0-skip}} + C_{\text{linear}}$	$\mathbf{900 \pm 54}$
$V_{\text{mean, 1-skip}} + C_{\text{linear}}$	885 ± 48
$V_{\text{mean, 3-skip}} + C_{\text{linear}}$	734 ± 159
$V_{\text{mean, 0-skip}} + C_{\text{nonlinear}}$	$\mathbf{902 \pm 31}$

function in the output of the controller attenuates the noise coming out from sampling q_θ.

Finally, in the original work of Ha and Schmidhuber (2018), a bar at the bottom of the screen containing information about speed and acceleration is cropped from the model's visual input. We compare the performance obtained when using such cropped images against that obtained from the game's full image. As seen in our results, the use of the lower bar does improve the performance of the agent. Most approaches reported in the upper part of Table 1 use the full game image.

3.3 Ablation study

We now report in Table 3 the performance obtained by removing different components from the model in Fig. 1. We start by considering only the visual module. The first result ($V_{\text{sampled}} + C_{\text{squash}}$) corresponds to the configuration of HS without the memory module, and is the worst performing configuration. Following on our conclusions from Section 3.2, we can again observe that the use of the mean of q_θ or the use of a full game image both contribute to an improved performance. Finally, it is worth noting that none of the memoryless configurations is able to solve the game.

We also consider the impact of the memory module. Following Tallec et al. (2018), we analyze both the impact of considering only the memory module and the performance obtained with an *untrained* memory module. The results suggest that the memory module provides the agent with the temporal information that is lacking in the visual information.

To confirm such interpretation, we considered a variation of the original model, where the vision module is trained not with single frame but with *stacked sequence of frames*, thus (implicitly) including temporal information in the visual perception. The results are reported in Table 4.

Our results show that, in fact, by including the multiple frames in the visual module, the agent is again able to complete the game, confirming the intuition that, in fact, the role of the memory module is to complement the visual perception with temporal information.

Table 5: Comparison of the average score obtained when the model is trained with data from a good driving policy.

Method	Avg. Score
$V_{\text{mean}} + C_{\text{linear}}$	836 ± 78
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$	$\mathbf{910 \pm 34}$

3.4 Training policy

To conclude our analysis, we note that the HS model is trained with a batch of images from the game obtained with a random driving policy. We thus conducted an experiment to assess the impact that the policy used to sample the environment has in the model learned by the agent and, consequently, in the performance of the agent. The results are reported in Table 5. Our results clearly show that, by simply considering a better policy, the performance of the agent considerably improves.

4 Conclusion and Future Work

In this paper we conducted an empirical analysis of the architecture of Ha and Schmidhuber (2018). Our results highlight the role of each component of the model in the overall performance of the agent. One improvement suggested by our results is with respect with the policy used to sample the data used to train the model. We refer to the supplementary material for additional discussions.

References

C. Bishop. Mixture density networks. Technical Report NCRG/94/004, Neural Computing Research Group, Aston University, February 1994.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

A. Gaier and D. Ha. Weight agnostic neural networks. In *Advances in Neural Information Processing Systems 32*, pages 5365–5378, 2019.

A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2450–2462, 2018.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

D. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proc. 2nd International Conference on Learning Representations*, 2014.

T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *Proc. 4th Int. Conf. Learning Representations*, 2016.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

S. Risi and K. Stanley. Deep neuroevolution of recurrent and discrete world models. *Proc. 2019 Genetic and Evolutionary Computation Conference*, pages 456–462, 2019.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, Shogi, and Go through self-play. *Science*, 362:1140–1144, 2018.

F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proc. 29th Int. Conf. Machine Learning*, pages 1547–1554, 2012.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

C. Tallec, L. Blier, and D. Kalainathan. Reproducing "World Models": Is training the recurrent network really needed? Available at https://ctallec.github.io/world-models/, 2018.

Y. Tang, D. Nguyen, and D. Ha. Neuroevolution of self-interpretable agents. *CoRR*, abs/2003.08165, 2020.

T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019.

A Full comparative results

A.1 Ablation study additional results

Table 6: Comparison of average score of our ablation experiments on both the game's cropped and full image

Method	**Cropped**	**Full**
$V_{\text{mean}} + C_{\text{linear}}$	836 ± 78	877 ± 44
$V_{\text{sampled}} + C_{\text{squash}}$	629 ± 102	813 ± 102
$V_{\text{mean}} + C_{\text{nonlinear}}$	872 ± 72	881 ± 79
$M_{\text{trained}} + C_{\text{linear}}$	852 ± 113	894 ± 25
$M_{\text{untrained}} + C_{\text{linear}}$	754 ± 143	702 ± 121
$V_{\text{mean}} + M_{\text{untrained}} + C_{\text{linear}}$	852 ± 110	886 ± 55

A.2 Improved Sample Policy

The *VAE* and *MDN-RNN* components are trained with a batch of images from the game sampled with a random driving policy. We thus considered training these components with a mix of images sampled with both a random policy and an expert policy, by which it could help these components capture better features and dynamics of the environment. This improved sampling method we denoted as *Improved sample*.

Table 7: Comparison of average score obtained with a V_{mean} component trained with images with different sampling methods on the game's full image

Method	**Cropped**	**Full**
$V_{\text{mean}} + C_{\text{linear}}$	826 ± 117	877 ± 44
$V_{\text{mean}} + C_{\text{linear}}$ (*Improved sample*)	836 ± 78	$\mathbf{906 \pm 51}$

Table 8: Comparison of average score and steps to reach a score of 900 obtained by training the *VAE* and *MDN-RNN* components with images with different sampling methods

Method	**Avg. Score**	**900 in step**
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$ (Full)	920 ± 29	72
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$ (Crop)	901 ± 43	119
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$ (Full) (*Improved sample*)	913 ± 34	36
$V_{\text{mean}} + M_{\text{trained}} + C_{\text{linear}}$ (Crop) (*Improved sample*)	910 ± 34	35

We can notice that by using a better sampling method for training the HS ”visual” and ”memory” components tends to achieve better results, and reach them faster.