Visualizing MuZero Models

Joery A. de Vries [1 2] **Ken S. Voskuil** [1 2] **Thomas M. Moerland** [1] **Aske Plaat** [1]

Abstract

MuZero, a model-based reinforcement learning algorithm that uses a value equivalent dynamics model, achieved state-of-the-art performance in Chess, Shogi and the game of Go. In contrast to standard forward dynamics models that predict a full next state, value equivalent models are trained to predict a future value, thereby emphasizing value relevant information in the representations. While value equivalent models have shown strong empirical success, there is no research yet that visualizes and investigates what types of representations these models actually learn. Therefore, in this paper we visualize the latent representation of MuZero agents. We find that action trajectories may diverge between observation embeddings and internal state transition dynamics, which could lead to instability during planning. Based on this insight, we propose two regularization techniques to stabilize MuZero's performance. Additionally, we provide an open-source implementation of MuZero along with an interactive visualizer of learned representations, which may aid further investigation of value equivalent algorithms.

1. Introduction

Model-based reinforcement learning has shown strong empirical success in sequential decision making tasks, as illustrated by the AlphaZero (Silver et al., 2018) and MuZero algorithms (Schrittwieser et al., 2020). Both of these approaches nest a planning loop, based on Monte Carlo Tree Search (Kocsis & Szepesvári, 2006; Browne et al., 2012), inside a learning loop, where we approximate global value and policy functions. While AlphaZero used a known model of the environment, MuZero learned the model from sampled data. However, instead of a standard forward dynamics model, which learns to predict future states, the MuZero dynamics model is trained to predict *future values*, better known as a *value equivalent model* (Grimm et al., 2020).

A potential benefit of value equivalent models, compared to standard forward models, is that they will emphasize value and reward relevant characteristics in their representation and dynamics. This may be beneficial when the true dynamics are complicated, but the value relevant aspects of the dynamics are comparatively simple. As a second benefit, we train our model for its intended use: predicting value information during planning. Several papers have empirically investigated this principle in recent years (Tamar et al., 2016; Oh et al., 2017; Farquhar et al., 2018; Silver et al., 2017b; Schrittwieser et al., 2020), while (Grimm et al., 2020) provides a theoretical underpinning of this approach.

However, no literature has yet investigated what kind of representations these approaches actually learn, i.e., *how* the learned representations are organized. The goal of this paper is therefore to investigate and visualize environment models learned by MuZero. Most interestingly, we find that, after training, an action trajectory that follows the forward dynamics model usually departs from the learned embedding of the environment observations. In other words, MuZero is not enforced to keep the state encoding and forward state prediction congruent. Therefore, the second goal of this paper is to regularize MuZero's dynamics model to improve its structure. We propose two regularization objectives to add to the MuZero objective, and experimentally show that these may indeed provide benefit.

In short, after introducing related work (Sec. 2) and necessary background on the MuZero algorithm (Sec. 3), we discuss two research questions: 1) what type of representation do value equivalent models learn (Sec. 4), and 2) can we use regularization to better structure the value equivalent latent space (Sec. 5)? We experimentally validate the second question in Sec. 6 and 7. Moreover, apart from answering these two question, we also open source modular MuZero code including an interactive visualizer of the latent space based on principal component analysis (PCA), available from `www.anonymized.org`. We found the visualizer to greatly enhance our understanding of the algorithm, and believe visualization will be essential for deeper understanding of this class of algorithms.

[1]Leiden Institute of Advanced Computer Science, Leiden, The Netherlands [2]Authors contributed equally. Correspondence to: Joery A. de Vries <j.a.de.vries.3@umail.leidenuniv.nl>, Ken S. Voskuil <k.s.voskuil@umail.leidenuniv.nl>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

2. Related Work

Value equivalent models, a term introduced by (Grimm et al., 2020), are usually trained on end-to-end differentiable computation graphs, although the principle would be applicable to gradient-free optimization as well. Typically, the unrolled computation graph makes multiple passes through a dynamics model, and eventually predicts a value. Then, the dynamics model is trained through gradient descent on its ability to predict the correct value. The first value equivalent approach were Value Iteration Networks (VIN) (Tamar et al., 2016), where a differentiable form of value iteration was embedded to predict a value. Other variants of value equivalent approaches are Value Prediction Networks (VPN) (Oh et al., 2017), TreeQN and ATreeC (Farquhar et al., 2018), the Predictron (Silver et al., 2017b), and MuZero (Schrittwieser et al., 2020). These methods differ in the way they build their computation graph, where VINS and TreeQN embed entire policy improvement (planning) in the graph, VPNs, the Predictron and MuZero only perform policy evaluation. Therefore, the latter approaches combine explicit planning for policy improvement, which in the case of MuZero happens through MCTS. (Grimm et al., 2020) provides a theoretical analysis of value equivalent models, showing that two value equivalent models give the same Bellman back-up.

MuZero uses the learned value equivalent model to explicitly plan through Monte Carlo Tree Search (Kocsis & Szepesvári, 2006; Browne et al., 2012), and uses the output of the search as training targets for a learned policy network. This idea of iterated planning and learning dates back to Dyna-2 (Silver et al., 2008), while the particularly successful combination of MCTS and deep learning was introduced in AlphaGo Zero (Silver et al., 2017a) and Expert Iteration (ExIt) (Anthony et al., 2017). In general, planning may add to pure (model-free) reinforcement learning: 1) improved action selection, and 2) improved (more stable) training targets. On the other hand, learning adds to planning the ability to generalize information, and store global solutions in memory. For more detailed overviews of value equivalent models and iterated planning and learning we refer to the model-based RL surveys by (Moerland et al., 2020; Plaat et al., 2020).

Visualization is a common approach to better understand machine learning methods, and visualization of representations and loss landscapes of (deep) neural networks has a long history (Bischof et al., 1992; Yosinski et al., 2015; Karpathy et al., 2015; Li et al., 2018a). For example, (Li et al., 2018b) shows how the loss landscape of a neural network can indicate smoothness of the optimization criterion. Visualization is also important in other areas of machine learning, for example to illustrate how kernel-methods project low dimensional data to a high dimensional space for classification (Szymanski & McCane, 2011). Note that the most common approach to visualize neural network mappings is through *non-linear* dimensionality reduction techniques, such as Stochastic Neighbour Embedding (Hinton & Roweis, 2002). We instead focus on linear projections in low dimensional environments, as non-linear dimensionality reduction has the risk of altering the semantics of the MDP models.

3. The MuZero Algorithm

We briefly introduce the MuZero algorithm (Schrittwieser et al., 2020). We assume a Markov Decision Process (MDP) specification given by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, U, \gamma \rangle$, which respectively represent the set of states ($\mathcal{S}$), the set of actions ($\mathcal{A}$), the transition dynamics mapping state-action pairs to new states ($\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow p(\mathcal{S})$), the reward function mapping state-action pairs to rewards ($U : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$), and a discount parameter ($\gamma \in [0, 1]$) (Sutton & Barto, 2018). Internally, we define an abstract MDP $\langle \widetilde{\mathcal{S}}, \mathcal{A}, \widetilde{\mathcal{T}}, R, \gamma \rangle$, where $\widetilde{\mathcal{S}}$ denotes an abstract state space, with corresponding dynamics $\widetilde{\mathcal{T}} : \widetilde{\mathcal{S}} \times \mathcal{A} \rightarrow \widetilde{\mathcal{S}}$, and reward prediction $R : \widetilde{\mathcal{S}} \times \mathcal{A} \rightarrow \mathbb{R}$. Our goal is to find a policy $\pi : \mathcal{S} \rightarrow p(\mathcal{A})$ that maximizes the value $V(s)$ from the start state, where $V(s)$ is defined as the expected infinite-horizon cumulative return:

$$V(s) = \mathbb{E}_{\pi,T}\Big[\sum_{t=0}^{\infty} \gamma^t \cdot u_t | s_0 = s\Big]. \tag{1}$$

We define three distinct neural networks to approximate the above MDPs (Figure 1): the state encoding/embedding function h_θ, the dynamics function g_θ, and the prediction network f_θ, where θ denotes the joint set of parameters of the networks. The encoding function $h_\theta : \mathcal{S} \rightarrow \widetilde{\mathcal{S}}$ maps a (sequence of) real MDP observations to a latent MDP state. The dynamics function $g_\theta : \widetilde{\mathcal{S}} \times \mathcal{A} \rightarrow \widetilde{\mathcal{S}} \times \mathbb{R}$ predicts the next latent state and the associated reward of the transition. In practice, we slightly abuse notation and also write g_θ to only specify the next state prediction. Finally, the prediction network $f_\theta : \widetilde{\mathcal{S}} \rightarrow p(\mathcal{A}) \times \mathbb{R}$ predicts both the policy and value for some abstract state $\tilde{s}$. We will identify the separate predictions of $f_\theta(\tilde{s}_t^k)$ by $\mathbf{p}_t^k$ and V_t^k, respectively, where subscripts denote the time index in the true environment, and superscripts index the timestep in the latent environment. Also, we write $\mu_\theta = (h_\theta, g_\theta, f_\theta)$ for the joint model.

Together, these three networks can be chained to form a larger computation graph that follows a single trace, starting from state s_t following action sequence $(a_t, \ldots, a_{t+n})$. First, we use the embedding network to obtain the first latent state from the sequence of observations: $\tilde{s}_t^0 = h_\theta(s_0, \ldots, s_t)$. Then, we recursively transition forward in the latent space following $(\tilde{s}_t^{k+1}, r_t^{k+1}) = g_\theta(\tilde{s}_t^k, a_{t+k})$. Finally, for all generated abstract states $(\tilde{s}_t^0, \ldots, \tilde{s}_t^n)$ we use

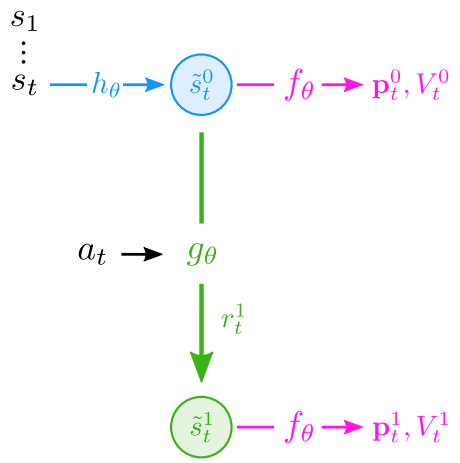

Figure 1. Illustration of MuZero's RNN unfolding. During training, actions are given by the data.

the prediction network to obtain $(\mathbf{p}_t^k, V_t^k) = f_\theta(\tilde{s}_t^k)$. As illustrated in Figure 1, the entire differentiable computation graph implements the mapping

$$(s_0, .., s_t, a_t, .., a_{t+n}) \rightarrow (\mathbf{p}_t^0, V_t^0, r_t^1, .., \mathbf{p}_t^{n+1}, V_t^{n+1}, r_t^{n+1}). \tag{2}$$

We will later use this graph to train our networks, but first discuss how they may be utilized to perform a search.

Given three individual neural networks for f, g and h as, we then use these networks to perform a MCTS search from state s_t in the abstract space. First, we embed the current state through h. Then, we can essentially follow the PUCT (Rosin, 2011) algorithm, a variant of MCTS that incorporates prior weights on the available actions, where the priors originate from the policy network, $\mathbf{p}_t^k$ (first proposed by (Silver et al., 2017a)). The state transitions and policy/value predictions within the search are governed through g and f, respectively. Eventually, this MCTS procedure outputs a policy $\pi_t = \pi(s_t)$ and value estimate $V_t = V(s_t)$ for the root node, i.e., $(\pi_t, V_t) \sim \text{MCTS}(s_0, \ldots, s_t | \mu_\theta)$. It then selects action $a_t \sim \pi_t$ in the real environment, transitions to the next state, and repeats the search. We refer to appendix B of the MuZero paper for full details on the MCTS search (Schrittwieser et al., 2020).

We still need to discuss how to train f_θ, g_θ, and h_θ. Through playing episodes, MuZero collects trajectories η of state, action, reward sequences, and the associated MCTS statistics obtained at each state, i.e., $\eta = (s_t, \pi_t, a_t, r_t, s_{t+1}, \ldots)$. As mentioned before, given a start state and a sequence of actions, we can unroll the differentiable graph that predicts rewards, values and policy parameters at each predicted state in the sequence (Eq. 2, Fig 1). Since the entire graph is end-to-end differentiable, we may train it on a joint loss (Schrittwieser et al., 2020):

$$\begin{aligned} l(\theta) = \sum_{k=0}^{n} & l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, V_t^k) \\ & + l^p(\pi_{t+k}, \mathbf{p}_t^k) + \lambda \|\theta\|_2^2, \end{aligned} \tag{3}$$

where $l^r = 0$ for $k = 0$, $\lambda \in \mathbb{R}$ is a constant that governs the L2 regularization, and z_t is a cumulative reward estimate obtained from an n-step target on the real trace η, $z_t = \sum_{i=0}^{n} \gamma^i u_{t+i} + \gamma^n V_{t+n+1}$. For $n \rightarrow \infty$ this of course becomes a Monte Carlo return estimate. Moreover, MuZero uses distributional losses, inspired by (Pohlen et al., 2018), for all three predictions (r, V, and $\mathbf{p}$).

Finally, MuZero employs two forms of normalization to stabilize learning. First, inside the MCTS selection rule, it min-max normalizes the value estimates based on the minimum and maximum estimate inside the current tree. This allows the algorithm to adapt to arbitrary reward scaling. Second, and more important for our paper, it also uses min-max normalization as the output activation of h_θ and g_θ. In particular, every abstract state is readjusted through

$$\tilde{s}_{\text{scaled}} = (\tilde{s} - \min(\tilde{s}))/(\max(\tilde{s}) - \min(\tilde{s})), \tag{4}$$

where the $\min$ and $\max$ are over the elements in the $\tilde{s}$ vector. This brings the range of $\tilde{s}$ in the same range as the discrete actions (one-hot encoded), but also introduces an interesting restriction, especially when the latent space is small. For example, when $|\tilde{s}| = 4$, then the min-max normalization will enforce two elements within $\tilde{s}$ to be one of 0 or 1. In practice, we observed that MuZero required at least a $|\tilde{s}| + 2$ latent space to effectively learn a $|\tilde{s}|$-dimensional problem given that the state context was non-redundant and approximately Markov — this restriction was most stringent for small $|\tilde{s}|$. Full details on the MuZero algorithm can be found in the Appendix of (Schrittwieser et al., 2020).

4. Visualizing MuZero's Latent Space

Value equivalent models, like MuZero, are solely trained on a low-dimensional, scalar reward signal, as the value and policy targets are also derived from the rewards. While the concept of value equivalent models is convincing, it remains unclear what types of representations these models actually achieve, and how they shape their internal space. Therefore, we first train MuZero agents to extract their learned embeddings/ representations and afterwards depict their state-space to latent-space mapping graphically in a 3D space. We create visualizations strictly on low-dimensional control environments as it quickly becomes infeasible and nonsensical to map higher dimensional representations linearly back to only 3 dimensions.

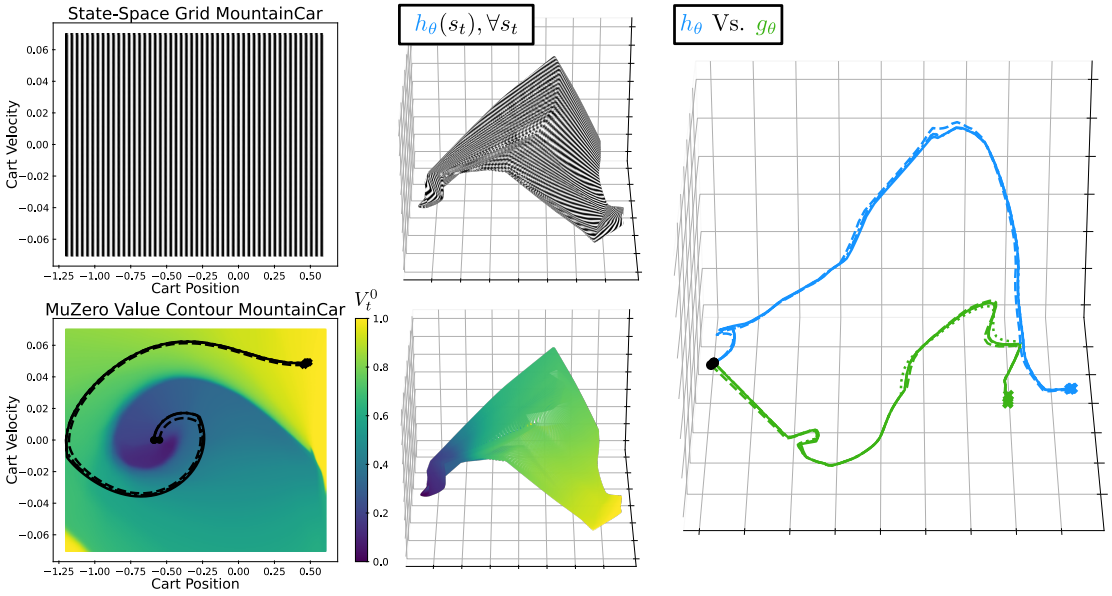

Figure 2. Left: Relation between true state space and latent state space on MountainCar. The top-left maps the true state space, indexed by vertical lines, into the latent space, retaining the vertical lines. The pattern indicates that the latent space gets curved, stretched and compressed. The bottom-left shows the same mapping, but this time indexes the states by their value estimate near convergence. We see that the latent embedding groups states that are dynamically close, which is visible in the grouping of states with similar values (in MountainCar, states that are dynamically close also have similar value, since there is only one start region and one sparse goal). Right: Latent embedding of a ground truth trace pushed through the encoder h (blue) and latent embedding of the same trace embedded at the first state and subsequently propagated at latent level through g. We clearly see that, although both traces should cover the same trace and cumulative reward, their latent trajectories diverge.

Figure 2 explores such a graphical depiction of MuZero's latent space near convergence (See also our supplementary material for additional visualizations during training and over ablations). The left part of the figure aims to relate the ground-truth state space to the latent space. On the top left, we visualize the ground-truth MountainCar state space (a 2D grid consisting of car position and car velocity) using vertical bars. Directly next to it, we visualize a Principal Component (PC) projection of the associated MuZero latent space when moving through the embedding function, i.e., the subspace $\{\tilde{s} : \tilde{s} = h_\theta(s), \forall s \in \mathcal{S}\}$ projected to its own 3-dimensional PC-space where $L = |\tilde{s}| = 4$. We retain the vertical bars, which thereby illustrates the way the true state space is curved into a high dimensional manifold. We do see that MuZero recovers the state space to a large extent, while it was never explicitly trained on it, although some parts are compressed or stretched.

The bottom-left of Figure 2 provides a similar illustration of the mapping between observed states and embedded states, but this time with accompanying value information. The bottom-left figure shows the learned value function of the state-space, where the black lines illustrate some successful final MuZero trajectories. Interestingly, when we map the value information into the embedding, we see that MuZero has actually strongly warped the original space. Relative to this projection, states with low value are grouped in the left-side of this space, while states with high value appear near the right-side. Effectively, MuZero has created a representation space in which it groups states that are dynamically close. For example, in the true state space (left), dark blue (low value) and yellow (high value) states are often adjacent. However, in the embedding (right), these states are strongly separated, because they are *dynamically distant* (we cannot directly move from blue to yellow, but need to follow the trajectory shown in black). This shows that MuZero indeed manages to retrieve essential parts of the dynamics, while only being trained on a scalar reward signal.

Our last visualization of the learned model is shown in the right panel of Figure 2. In this plot, we again visualize a PCA of MuZero fitted on the embedding near convergence. However, this time we fix a start state (s_0) and action sequence ($a_0, \ldots, a_n$), and visualize a successful trace passing through both the embedding and dynamics functions. For the embedding function, given a trace of environment data η, we plot $\langle h_\theta(s_0), h_\theta(s_1), \ldots, h_\theta(s_{n+1})\rangle$ as the *blue* trajectory. In contrast, in *green* we embed the first state and subsequently repeatedly transition through the learned dynamics function g_θ—i.e., we plot $\langle h_\theta(s_0), g_\theta(h_\theta(s_0), a_0), g_\theta(g_\theta(h_\theta(s_0), a_0), a_1), \ldots\rangle$. Interestingly, these two trajectories do not overlap. Instead,

the latent forward dynamics seem to occupy a completely different region of the latent space than the state encoder. We repeatedly make this observation on different experiments and repetitions. Although one would conceptually expect these trajectories to overlap, there is no explicit loss term that enforces these trajectories to stay congruent. When the latent capacity is large enough, MuZero may fail to identify this overlap, which hypothetically could waste generalization potential. In the next section we investigate two possible modifications to combat this phenomena.

5. Regularizing the Latent Space

Ideally, the dynamics function should stay relatively congruent to the embedding of observed environment states. Thereby, we can maximally profit from generalization in the latent space, and make optimal use of the available data. However, as we observed in the previous section, there is no explicit training objective that ensures this congruence. We therefore propose two types of regularization, i.e., differentiable penalty terms (added to the loss) that enforce the latent space to be more congruent.

The first solution is to add a *contrastive regularization*, which penalizes between the embedding of a state and the forward dynamics prediction towards the same abstract state. Given a trace η, we compute the loss

$$l^c(\theta) = \sum_{k=0}^{n} \|h_\theta(s_{t+k+1}) - \tilde{s}_t^{k+1}\|_2^2, \tag{5}$$

where $\tilde{s}_t^{k+1} = g(\ldots g(g(h_\theta(s_0), a_0), a_1), \ldots, a_k)$. This loss is added to the standard MuZero loss (Eq. 3), where its strength is governed by a scalar $\omega \in \mathbb{R}^+$. To prevent trivial solutions, we block the gradients of the above loss through h_θ for $k > 0$. This ensures that only the dynamics function should move towards the embeddings (and not the other way around), assuming that the embedding function is already relatively well adapted. This type of contrastive loss is often seen in supervised learning, for example in Siamese Networks (Chen & He, 2020; Koch et al., 2015). The process is visually illustrated in the top graph of Figure 3.

A second solution can be the use of a *decoding regularization*, which decodes latent predictions back to the true observation. This essentially adds a (multi-step) forward prediction loss to the MuZero model. We initialize an additional decoder network h_θ^{-1}, which maps latent states back to the true observations: $s'_{t+k} = h_\theta^{-1}(\tilde{s}_t^k)$. Then, given a trace of data η, we compute the loss

$$l^d(\theta) = \sum_{k=0}^{n} \|s_{t+k+1} - h_\theta^{-1}(\tilde{s}_t^{k+1})\|_2^2, \tag{6}$$

with $\tilde{s}_t^{k+1}$ defined as in the previous loss. Again, we can use a scalar $\omega \in \mathbb{R}^+$ to determine the relative contribution of the loss when added to Eq. 3. This approach combines the value equivalent loss model loss with a standard forward prediction loss. The additional penalty ties the latent predictions of g_θ back to the ground-truth observations. Although this mechanism is independent of the encoding h, it may potentially help to regularize the space of g_θ transitions. Moreover, as a second benefit, this regularization may be advantageous in initial stages of learning, as state transition are often dense in information compared to sparse reward signals. Thereby, it could be beneficial in early learning, but can be overtaken by the value equivalent objective when the model converges. The second type of regularization is visualized in the bottom graph of Figure 3.

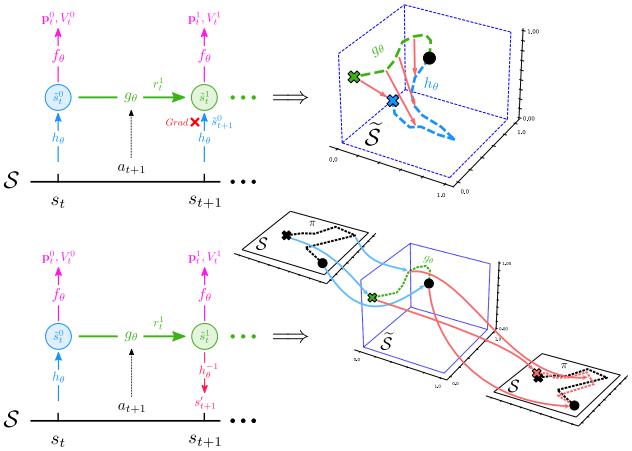

Figure 3. Two types of regularization of the MuZero objective. Top: *Contrastive regularization*, where we penalize the discrepancy between the embedding of a trace h (blue dashed line) and the forward predictions of the same action sequence through g (green dashed line). The contrastive loss pushes the green line towards the blue line (shown in red). Bottom: *Decoding regularization*, which adds an explicit decoding network h_θ^{-1}, which maps predictions of the latent model g (green dashed line) back to the observed state space (red lines). This enforces the learned latent model to retain some congruency with the true dynamics model.

6. Experimental Setup

We will now experimentally study our regularized versions of MuZero compared to standard MuZero. As an additional baseline, we also include an adaptation of AlphaZero for single-player domains (Silver et al., 2018), i.e., a similar agent with a perfect environment model. We provide a complete reimplementation of MuZero based on the original paper (Schrittwieser et al., 2020). All code is written in Python using Tensorflow 2.0 (Abadi et al., 2015). The code follows the AlphaZero-General framework (Nair et al., 2017) and is available at `https://anonymized.org`,

along with all resulting data.

The intention for our experiments was not to achieve maximal performance of MuZero, but to visualize the learned representations and study the effects of the different regularization terms. For this reason, we focus on low-dimensional, classic control environments, as it is hard to meaningfully visualize high-dimensional state spaces. In particular, we tested our work on CartPole and MountainCar, from OpenAI's Gym (Brockman et al., 2016). To aid our analysis, we developed an interactive visualization tool to for learned MuZero models on the MountainCar environment at `https://anonymized.org`. We encourage the reader to play around with this tool as we personally found this to provide more insight than static 2D images.

Table 1. Overview of factorial experimental design: 1) regularization type, 2) regularization strength (ω), 3) latent roll-out depth (K), and 4) latent dimensionality ($L = |\tilde{s}|$). Color annotations correspond to the loss regularization methods depicted in Figure 3.

Hyperpar.	CartPole	MountainCar
Regular.	{None, Contr., Dec.}	{None, Contr., Dec.}
ω	$\{0.01, 0.1, 1\}$	$\{1\}$
K	$\{1, 5, 10\}$	$\{3\}$
L	$\{4, 8\}$	$\{4, 8\}$

All hyperparameters can be found in Appendix A in supplemental material. For the results in this paper, we report on a factorial experiment design shown in Table 1, where we vary: 1) the regularization type, 2) the regularization strength (ω), 3) the number of latent unroll steps (K), and 4) the size of the latent representation ($L = |\tilde{s}|$). Of course, our main interest is in the ability of the regularizations to influence performance. All experiments were run on the CPU. The hyperparameter analysis experiment was run asynchronously on a computing cluster using 24 threads at 2GHz. On average it took about half a day to train an agent on any particular environment. The complete experiments depicted take about one week time to complete (after the separate lengthy tuning phase of the other hyperparameters).

7. Results

Figure 4 shows results of our CartPole experiments. In both plots we compare AlphaZero, MuZero, and our two regularized MuZero extensions. On the left, we investigated the effect of the regularization strength. AlphaZero consistently solves this environment, while MuZero has more trouble and solves it roughly 75% of the time. Both regularization methods do not seem to structurally improve MuZero performance on CartPole. Surprisingly, performance dips for intermediate regularization strength ($\omega = 0.1$). On the right of Figure 4, we display the average training curves for each of our four algorithms on CartPole. We see a similar pattern, where the regularization losses do not seem to provide benefit.

Since CartPole is a dense reward environment, in which MuZero quickly obtains gradients to structure the latent space, we also investigate our reguralization methods on MountainCar. Figure 5 (left) shows the training progression for AlphaZero, MuZero and the two regularization methods. Again, AlphaZero outperforms all other agents. However, this time, both our regularization methods outperform MuZero in learning speed, where especially the contrastive regularization leads to faster learning. All methods do reach the same optimal performance.

We further quantify the strength of the learned internal model by 'blindfolding' the agent (Freeman et al., 2019). A blindfolded agent only gets to see the true state of the environment ever x number of steps, where we call x the 'observation sparsity'. The right graph of Figure 5 displays the average reward obtained by all MuZero models when varying the observation sparsity (horizontal axis). All agents consistently keep solving the problem up to an observation sparsity of depth 10, which shows that their models do not compound errors too fast. Then, performance starts to deteriorate, although only the contrastively regularized, $L = 4$ MuZero variant keeps solving the problem, even up to 200 steps without a ground truth observation. In general, the blindfolded agents show that the model stays consistent over long planning horizons (even when trained on shorter roll-out depths), and that especially contrastive regularization further increases this ability. Please refer to the Appendix for additional visualization of the latent space in MountainCar experiments.

8. Discussion

We studied the latent space of the MuZero algorithm, a successful combination of value equivalent models and iterated planning and learning. In particular, we visualized the latent space, showing how the true state space is partially recovered and warped into a dynamically relevant space. Moreover, we identified a frequent mismatch between the latent trajectories and the embeddings of the ground truth observations. To alleviate this issue, we introduced two forms of regularization, which did not improve performance in CartPole, but did improve performance on MountainCar. Finally, we also found that these models can accurately predict correct actions over long planning horizons, up to 200 steps in some cases, while being trained on relatively short trajectories.

We hypothesize that the benefit of regularization is most prominent on sparse reward tasks like MountainCar. In CartPole, where we quickly observe rewards, the representation space of MuZero will quickly receive relevant gradients

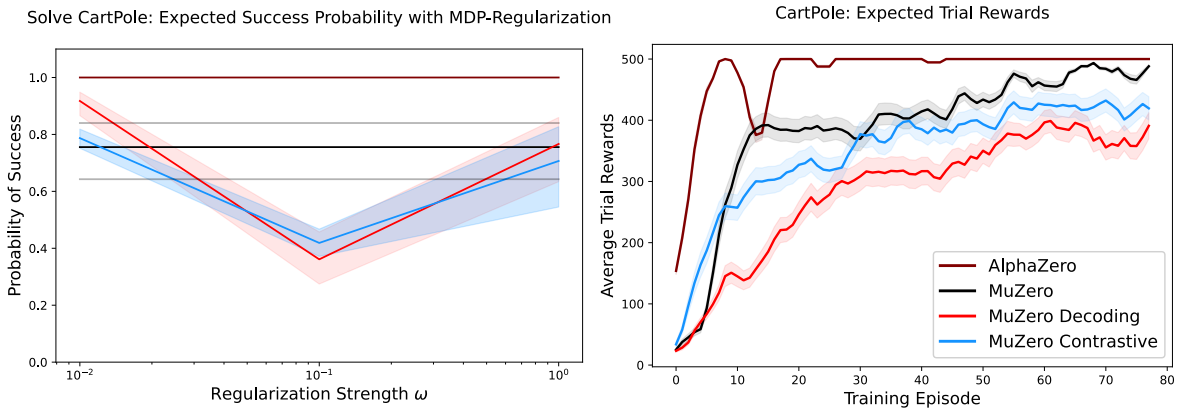

Figure 4. Performance of AlphaZero, MuZero and two regularized MuZero versions on CartPole. Results are averaged over all hyper-parameters K and L (Table 1). Left: Success percentage as a function of the regularization strength. 'Success' requires the maximum reward of 500 consistently over $n = 10$ test episodes. The error bars indicate a one standard error confidence interval of the mean success ratio. Right: Learning curves for AlphaZero, MuZero and two regularized MuZero versions. These curves are averaged over all values of K and L in Table 1, which acts as a sensitivity analysis.

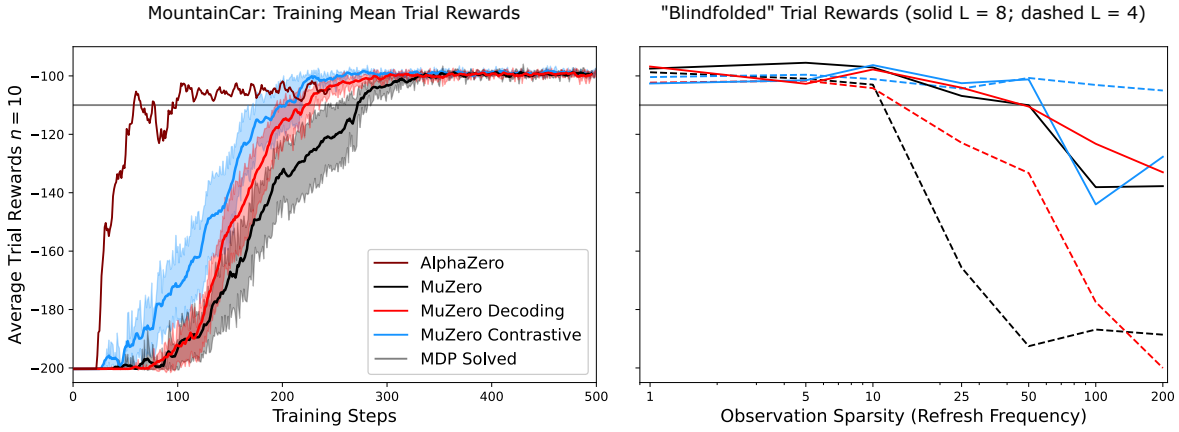

Figure 5. Performance of AlphaZero, MuZero and two regularized MuZero versions on MountainCar. Left: Reward progression during training. Results averaged over 10 repetitions. Right: Performance of blindfolded agents, where the agent only get to see the true environment observation every 'observation sparsity' number of steps. Solid curves for L=8, dashed curves for L=4. Results for a single repetition.

to structure itself. However, when reward information is sparse, we may benefit from additional regularization of the latent space, which might squeeze additional information out of the available data. Especially the contrastive regularization seems to show this effect, and could be a promising direction for future research.

A critique on our work could involve the dimensionality of the experiments. Deep reinforcement learning has started to focus more and more on high-dimensional problems. We opted for small scale experiments for two reasons. First of all, low-dimensional problems allow for better interpretation. We can only visualize a latent space in three dimensions, and it would not be feasible to map an Atari game representation back to 3D. Moreover, the general emphasis in RL has shifted towards to the challenge of dimensionality, while problems can be challenging in more ways than dimensionality only. A good illustration of the latter point is that such a high-impact algorithm as MuZero actually suffers to consistently solve CartPole and MountainCar. This recognition of the value of low dimensional experiments also starts to resurface in the RL community (Osband et al., 2019).

Furthermore, AlphaZero and MuZero are incredibly computationally expensive to run. While standard RL algorithms are already expensive to run, these iterated search and learning paradigms only exacerbate this limitation. When we use a small-scale MCTS of $n = 11$ at every timestep, it already takes at least 11 times as long to complete the same number of episodes as model-free RL. Compared to model-free RL, these iterated search and learning paradigms seem to benefit in long run performance, but they are computationally heavy

to train.

We observed that AlphaZero and, especially, MuZero are rather unstable to train. Their performance strongly depends on a variety of hyperparameters and network initialization, and we observe strong variation in the performance over different runs. A direction for future work is to further stabilize these algorithms. Our work on novel regularizations for value equivalent models like MuZero is a first step in this direction. Future work could investigate how to further stabilize MuZero through different regularizations, or further combinations with standard forward prediction or types of auxiliary losses (Jaderberg et al., 2016). Although value equivalency is a strong incentive for the final representation of the model, we may aid the model during learning with additional losses and regularizations, especially in sparse reward tasks.

9. Conclusion

The first part of this paper visualized the learned latent representations and model of MuZero, which we consider the main contribution of the paper. Our two most important observations are: 1) the learned latent space warps the original space into a 'dynamically close' space, in which states that are easy to reach from eachother are nearby, and 2) the embedding of an action trajectory through the latent dynamics often departs from the embedding of the ground-truth states, which suggest the latent space does not optimally generalize information. To alleviate the last problem, the second part of the paper investigated two new forms of regularization of value equivalent models: 1) a contrastive regularization, and 2) a decoder regularization. Experiments indicate that these approaches may aid MuZero in sparse reward tasks, and that these models in general manage to plan over long horizons (much longer than they were trained on). Our visualizations of the latent space show that the contrastive regularization term can force the dynamics function to predict values closer to the embedding, this in turn can help interpreting the look-ahead search of MuZero. All code and data to reproduce experiments is open-sourced at www.anonymized.com, including an interactive visualizer to further explore the MuZero latent space. We also hope our work further stimulates new emphasis on visualization of RL experiments, which may provide more intuitive insights into deep reinforcement learning.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30. NIPS Proceedings, 2017.

Bischof, H., Pinz, A., and Kropatsch, W. G. Visualization methods for neural networks. In *11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, volume 1, pp. 581–582. IEEE Computer Society, 1992.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Chen, X. and He, K. Exploring simple siamese representation learning. 2020. URL http://arxiv.org/abs/2011.10566.

Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. ICLR, 2018.

Freeman, C. D., Metz, L., and Ha, D. Learning to predict without looking ahead: World models without forward prediction. 2019. URL http://arxiv.org/abs/1910.13038.

Grimm, C., Barreto, A., Singh, S., and Silver, D. The value equivalence principle for model-based reinforcement learning. 2020. URL http://arxiv.org/abs/2011.03506.

Hinton, G. E. and Roweis, S. Stochastic neighbor embedding. 15:857–864, 2002. URL https://papers.nips.cc/paper/2002/hash/6150ccc6069bea6b5716254057a194ef-Abstract.html.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Karpathy, A., Johnson, J., and Fei-Fei, L. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. volume 37, pp. 8, 2015.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *Advances in neural information processing systems*, pp. 6389–6399, 2018a.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6389–6399. Curran Associates, Inc., 2018b.

Moerland, T. M., Broekens, J., and Jonker, C. M. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Nair, S., Thakoor, S., and Jhunjhunwala, M. suragnair/alpha-zero-general, 2017. URL https://github.com/suragnair/alpha-zero-general. original-date: 2017-12-01.

Oh, J., Singh, S., and Lee, H. Value prediction network. 2017. URL http://arxiv.org/abs/1707.03497.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvari, C., Singh, S., et al. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2019.

Plaat, A., Kosters, W., and Preuss, M. Model-based deep reinforcement learning for high-dimensional problems, a survey. *arXiv preprint arXiv:2008.05598*, 2020.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., Hessel, M., Munos, R., and Pietquin, O. Observe and look further: Achieving consistent performance on atari. 2018. URL http://arxiv.org/abs/1805.11593.

Rosin, C. D. Multi-armed bandits with episode context. 61(3):203–230, 2011. ISSN 1573-7470. doi: 10.1007/s10472-011-9258-6. URL https://doi.org/10.1007/s10472-011-9258-6.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. 2020. URL http://arxiv.org/abs/1911.08265.

Silver, D., Sutton, R. S., and Müller, M. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pp. 968–975, 2008.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. 550(7676):354–359, 2017a. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature24270. URL http://www.nature.com/articles/nature24270.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. 2017b. URL http://arxiv.org/abs/1612.08810.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. 362(6419): 1140–1144, 2018. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aar6404. URL https://science.sciencemag.org/content/362/6419/1140.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. Bradford Books, second edition edition, 2018. ISBN 978-0-262-03924-6.

Szymanski, L. and McCane, B. Visualising kernel spaces. pp. 449–452, 2011.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2154–2162, 2016.

van Seijen, H., Nekoei, H., Racah, E., and Chandar, S. The LoCA regret: A consistent metric to evaluate model-based behavior in reinforcement learning. 2020. URL http://arxiv.org/abs/2007.03158.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

A. Hyperparameters and Neural Architectures

Our neural network architectures for MuZero were partially inspired by the ones proposed by (van Seijen et al., 2020). We kept every constituent network within the larger MuZero model equivalent, that is: h_θ, g_θ, f_θ, and optionally h_θ^{-1} had the same *internal* architecture being a two-layered feed forward network with 32 hidden nodes using the Exponential Linear Unit activation function. The embedding function h_θ received only current observations, so no trajectories. Actions for the dynamics functions were one-hot-encoded and concatenated to the current latent state before being passed to g_θ; computed latent-states were also minmax-normalized within the neural architecture. The action policy, value function, and reward prediction were represented as distributions and trained using a cross-entropy loss, as was also done in the original paper (Schrittwieser et al., 2020). The AlphaZero neural network was simply f_θ that received state observations—i.e., when omitting h_θ and g_θ. All other unmentioned hyperparameters are shown in Table 2.

B. Additional Visualization of Latent Space

Figure 6 illustrates a typical training progression for MuZero on the MountainCar environment (hyperparameters can be found in the appendix), similarly to Figure 2 we used $L = |\tilde{s}| = 4$. The shapes shown in the top row portray the progression of the embedding $\{\tilde{s} : \tilde{s} = h_\theta(s), \forall s \in \mathcal{S}\}$, shown in a 3D space obtained through PCA. Upon initialization, the latent space is almost degenerate, and all latent traces occupy the same part of the latent space. The action selection strategy of MuZero is near uniform in this situation. Gradually during training, the agent starts to expand and eventually occupies the full latent space. In the top-right plot, we see that the latent spaces aggregate states with similar values. On the bottom, we display violin plots of the latent state distribution during training, in the true four dimensional latent space. We find that these distributions appear overinflated around 0 and 1, on all dimensions. Arguably, this is an artifact of the min-max normalization of the latent space (as discussed in Section 3), which always forces one element to be 0, and one element to be 1. We do see that MuZero spreads its assignment of min and max over all dimensions. Note that, although MuZero can theoretically still predict any point in the latent space $[0, 1]^4$, the normalization effectively reduces the degrees of freedom within the latent space.

Figure 7 provides additional illustration of the learned latent spaces for MuZero and our two regularized MuZero variants on MountainCar. The figure has a 3x2 design, where the rows depict different algorithms, and the columns different sizes of the latent space. The left of each cell contains a 3-dimensional PCA projection of the learned latent space. We again observe that the value equivalent objective nicely unfolds a latent space, in which states that are dynamically close in the true environment are also close in the latent space.

More interestingly, the right of each cell displays the two ways to embed a trajectory: 1) by passing each observed true state through h_θ (blue), or 2) by simulating forward through the latent dynamics g_θ (green). We again observe that these trajectories strongly depart for the original MuZero algorithm. In contrast, the two regularization methods force both trajectories to stay closer to eachother. Especially the contrastive loss manages to keep both trajectories close to consistent, which may help to obtain better generalization, and could explain the faster learning observed in Figure 5.

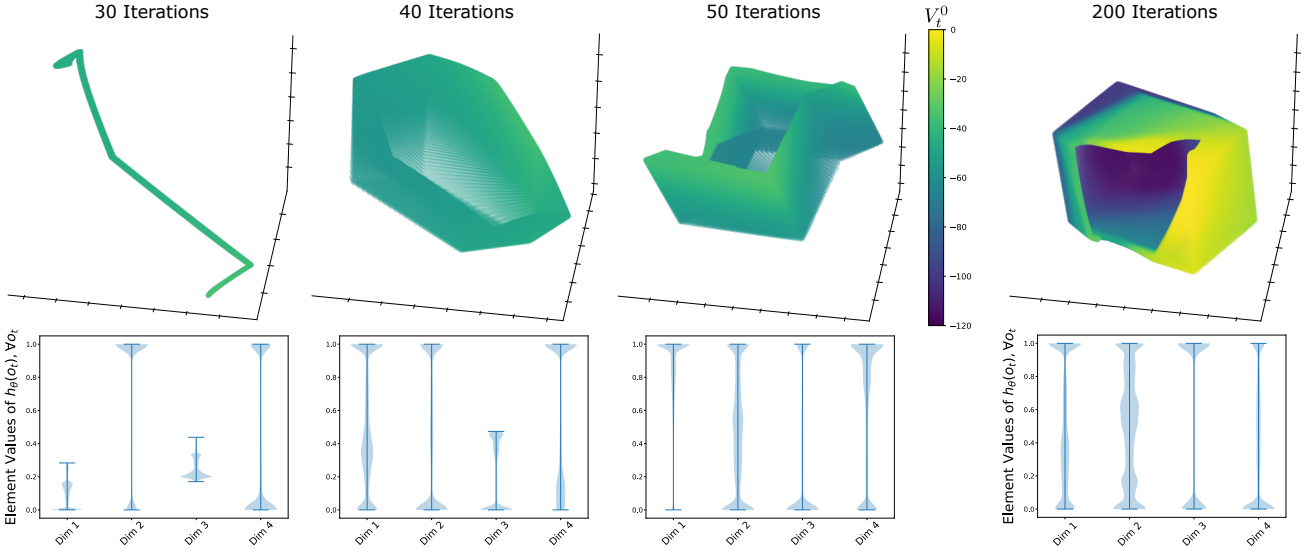

Figure 6. Development of the MuZero latent space over training. The progression on the top shows the 3D latent space, obtained by performing a principal component analysis on the agents true four-dimensional latent space. The color map indicates the learned value for each latent state. The bottom progression shows violin plots for the original 4D space, indicating which parts of the latent space are actually occupied during training.

Table 2. Overview of all constant hyperparameters used in our experiments for both AlphaZero and MuZero, see Table 1 for the non-constant hyperparameters.

Hyperparameter	Value (CP/ MC)	Description
Self-Play Iterations	80/ 1000	Number of data-gathering/ backpropagation loops to perform.
Episodes	20	Number of trajectories to generate within the environment for each Self-Play loop.
Epochs	40	Number of backprop steps to perform for each Self-Play loop.
Episode Length	500/ 200	Maximum number of steps per episode before termination.
α	0.25	Dirichlet alpha used for adding noise to the MCTS root prior.
$p_{explore}$	0.25	Exploration fraction for the MCTS root prior.
c_1	1.25	Exploration factor in MCTS's UCB formula.
c_2	19652	Exploration factor in MCTS's UCB formula.
τ	1	MCTS visitation count temperature.
N	11	Number of MCTS simulations to perform at each search.
Prioritization	False	Whether to use prioritized sampling for extracting training targets from the replay buffer.
Replay-Buffer window	10	Number of self-play loops from which data is retained, self-play iterations outside this window are discarded
TD-steps	10/ 50	Number of steps until bootstrapping when computing discounted returns.
γ	0.997	Discount factor for computing n-step/ TD-step returns.
Optimizer	Adam	Gradient optimizer for updating neural network weights.
η	$2 \cdot 10^{-2}$	Adam Learning Rate
Batch Size	128	Number of target samples to use for performing one back-propagation step.
λ	10^{-4}	L2-Regularization term for penalizing magnitude of the weights.
S	15/ 20	Number of distribution support points for representing the value and reward functions.

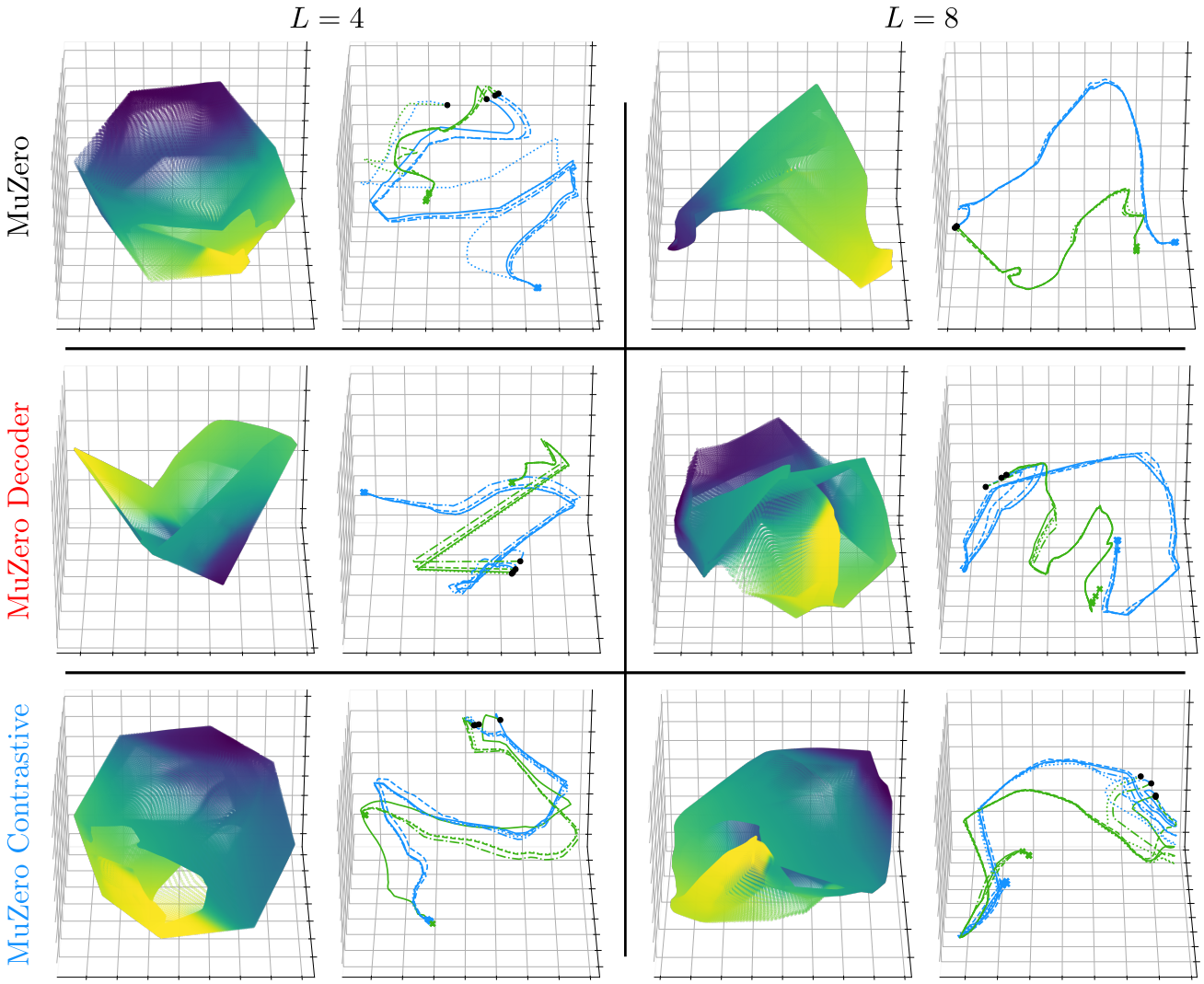

Figure 7. Rows: Different algorithms (MuZero, MuZero Decoder and MuZero Contrastive). Columns: Different latent sizes ($L = 4$, $L = 8$). For each combination we show two graphs: 1) (left) a 3-dimensional PCA projection of the learned embeddings in MountainCar and 2) (right) uniquely sampled trajectories embedded through h_θ (blue) or simulated forward through the latent dynamics g_θ (green). We see that both trajectories depart for standard MuZero, but especially contrastive regularization manages to keep these trajectories congruent.