A Study of the Weighted Multi-step Loss Impact on the Predictive Error and the Return in MBRL

Abdelhakim Benechehab
abdelhakim.benechehab1@huawei.com
Noah's Ark Lab, Huawei
Department of Data Science, EURECOM

Albert Thomas
albert.thomas@huawei.com
Noah's Ark Lab, Huawei

Giuseppe Paolo
giuseppe.paolo@huawei.com
Noah's Ark Lab, Huawei

Maurizio Filippone
maurizio.filippone@kaust.edu.sa
Statistics Program, KAUST

Balázs Kégl
balazs.kegl@huawei.com
Noah's Ark Lab, Huawei

Abstract

In model-based reinforcement learning, most algorithms rely on simulating trajectories from one-step models of the dynamics learned on data. A critical challenge of this approach is the compounding of one-step prediction errors as the length of the trajectory grows. In this paper we tackle this issue by using a multi-step objective to train one-step models. Our objective is a weighted sum of the mean squared error (MSE) loss at various future horizons. We find that this new loss is particularly useful when the data is noisy (additive Gaussian noise in the observations), which is often the case in real-life environments. We show in a variety of tasks (environments or datasets) that the models learned with this loss achieve a significant improvement in terms of the averaged R2-score on future prediction horizons. To our surprise, in the pure batch reinforcement learning setting, we find that the multi-step loss-based models perform only marginally better than the baseline. Furthermore, this improvement is only observed for small loss horizons, unlike the trend present with the R2-score on the respective datasets.

1 Introduction

In reinforcement learning (RL) we learn a control agent (or policy) by interacting with a dynamic system (or environment), receiving feedback in the form of rewards. Although successful in various applications (Silver et al., 2017; 2018; Mnih et al., 2015; Vinyals et al., 2019), RL remains largely confined to simulated environments and does not extend to real-world engineering systems. Model-based reinforcement learning (MBRL), can potentially narrow the gap between RL and applications thanks to a better sample efficiency.

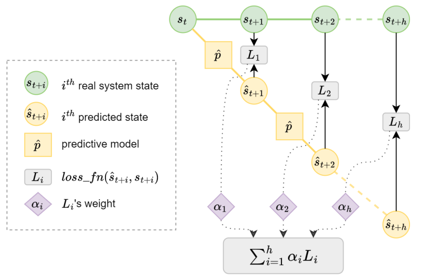

s_t
s_{t+1}
s_{t+2}
s_{t+h}
$\hat{p}$
L_1
L_2
L_h
$\hat{s}_{t+1}$
$\hat{s}_{t+2}$
$\hat{s}_{t+h}$
α_1
α_2
α_h
$\sum_{i=1}^h \alpha_i L_i$
s_{t+i} i^{th} real system state
$\hat{s}_{t+i}$ i^{th} predicted state
$\hat{p}$ predictive model
L_i $loss\_fn(\hat{s}_{t+i}, s_{t+i})$
α_i L_i's weight

Figure 1: Schematic representation of the multi-step prediction framework.

MBRL algorithms alternate between two steps: i) model learning, a supervised learning problem to learn the dynamics of the environment, and ii) policy optimization, where a policy and/or a value function is updated by sampling from the learned dynamics. MBRL is recognized for its sample efficiency, as policy/value learning is conducted from imaginary model rollouts, which are more cost-effective and readily available than rollouts in the true dynamics (Janner et al., 2019).

While MBRL algorithms have achieved significant success, they are prone to *compounding errors* when planning over extended horizons (Lambert et al., 2022). To tackle this problem, our approach involves adjusting the training objective of these models to focus on optimizing for long-horizon error (Fig. 1). This strategy is especially beneficial in presence of additive noise in observations, a context that mirrors real-world scenarios. Interestingly, we find that despite improving the predictive error of models, this approach does not necessarily result in significantly better model-based policies (especially for longer loss horizons).

The formal definition of the multi-step loss is given in Section 2. We then present the experimental setup and results in Section 3. In the appendices section, we provide a detailed related work (Appendix A), a theoretical analysis of the multi-step loss in tractable systems (Appendix B), and the details of our experiments (Appendix C, D, and E).

2 The multi-step loss

In MBRL, it is common to use a parametric model $\hat{p}_\theta$ that predicts the state[1] one-step ahead $\hat{s}_{t+1} \leadsto \hat{p}_\theta(s_t, a_t)$. We train this model to optimize the one-step predictive error $L\big(s_{t+1}, \hat{p}_\theta(s_t, a_t)\big)$ (MSE or NLL for stochastic modeling) in a supervised learning setting. To learn a policy, we use these models for predicting h steps ahead by applying a procedure called *rollout*.

In practice, a rollout corresponds to generating $\hat{s}_{t+j} \leadsto \hat{p}_\theta(\hat{s}_{t+j-1}, a_{t+j-1})$ recursively for $j = 1, \ldots, h$, to collect a trajectory $\tau = (s_0, a_0, \hat{s}_1, a_1, ..., \hat{s}_j, a_j, ...)$, where $(a_t, \ldots, a_{t+h-1}) = \boldsymbol{a}_{t:t+h-1} = \boldsymbol{a}_\tau$ is either a fixed action sequence generated by planning or sampled from a policy $a_{t+j} \leadsto \pi(s_{t+j})$ for $j = 1, \ldots, h$, on the fly. We denote $\hat{p}^j_\theta(s_t, \boldsymbol{a}_{t:t+j-1}) = \hat{p}_\theta\big(\hat{s}_{t+j-1}, a_{t+j-1}\big)$ for $j = 1, \ldots, h$, the prediction after j recursive applications of the model.

Using $\hat{p}^h_\theta(s_t, \boldsymbol{a}_{t:t+h-1})$ to estimate s_{t+h} is problematic for two reasons:

- A distribution mismatch occurs between the inputs that the model was trained on ($s_{t+1} \leadsto p(s_t, a_t)$) and the inputs the model is being evaluated on ($\hat{s}_{t+1} \leadsto \hat{p}_\theta(s_t, a_t)$) Talvitie (2014; 2017).
- The predictive error (and the modeled uncertainty in the case of stochastic models) will propagate through the successive model calls, leading to compounding errors (Lambert et al., 2022; Talvitie, 2014; Venkatraman et al., 2015).

To mitigate these issues, we study models that, given the full action sequence $\boldsymbol{a}_{t:t+h-1}$, learn to predict the state s_{t+h} by recursively predicting the intermediate states s_{t+j}, for $j = 1, \ldots, h$. We address this problem through the use of a weighted multi-step loss that accounts for the predictive error at different future horizons.

Definition (Weighted multi-step loss). Given horizon-dependent weights $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_h)$ with $\sum_{i=1}^h \alpha_i = 1$, a one-step loss function L, an initial state s_t, an action sequence $\boldsymbol{a}_\tau = \boldsymbol{a}_{t:t+h-1}$, and the real (ground truth) visited states $\boldsymbol{s}_\tau = \boldsymbol{s}_{t+1:t+h}$, we define the weighted multi-step loss as

$$L^h_{\boldsymbol{\alpha}}\big(\boldsymbol{s}_\tau, \hat{p}_\theta(s_t, \boldsymbol{a}_\tau)\big) = \sum_{j=1}^{h} \alpha_j L\big(s_{t+j}, \hat{p}^j_\theta(s_t, \boldsymbol{a}_{t:t+j-1})\big).$$

The dependency of $L^h_{\boldsymbol{\alpha}}$ on h is omitted in the rest of the paper when h is clear from the context. Furthermore, the loss L used in the multi-step loss $L_{\boldsymbol{\alpha}}$ will always be the MSE.

A multi-step loss with equal weights has been previously used in the literature (Lutter et al., 2021; Byravan et al., 2021; Xu et al., 2018). However, we are not aware of any other work that

[1] In this section, we do not make the distinction between states s and observations o as the definition of the multi-step loss is independent of the underlying MDP.

thoroughly study the more general weighted case of the multi-step loss, nor it's relevance for noisy observations (typically, this corresponds to homoscedastic Gaussian noise in the observations: $o_t = s_t + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ and $\sigma \in \mathbb{R}$). Indeed, predicting at different timesteps can be seen as a multi-task problem due to the system's periodicity and the growing scale of the long-horizon error. Therefore, we suggest the weighting mechanism as a mean to balance the training objective.

3 Experiments & results

In this section, we evaluate the performance of models trained with the multi-step loss $L_{\boldsymbol{\alpha}}$ for different values of the maximal horizon $h \in \{2, 3, 4, 10\}$ and different noise levels of the dynamics σ. For the weights $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_h)$, we consider an exponential parametrization (with a single parameter β) to reduce the size of the search space: $\alpha_i = \underbrace{\left(\frac{1-\beta}{1-\beta^{h+1}}\right)}_{\text{normalization constant}} \cdot \beta^i$ for $i \in \{1, \ldots, h\}$.

For the evaluation of the models we consider both a static and a dynamic setup. The *static evaluation* denotes the evaluation of dynamics models in terms of predictive error in held-out test datasets. For the *dynamic evaluation* we will consider the offline MBRL setting (Levine et al., 2020) where the goal is to learn a policy from a given dataset without interacting further with the environment. The static evaluation is done on three classical RL environments (*Cartpole swing-up*, *Swimmer* and *Halfcheetah*) and various datasets (eight in total) collected with different behavior policies (*random*, *medium*, *full_replay* and *mixed_replay*) on these environments. The dynamic evaluation is done on the *Cartpole swing-up* environment where the rewards are fully determined by the observations. The details of these tasks are provided in Appendix C. For all these tasks, we use the same neural network model. Implementation details for the model are given in Appendix D.

3.1 Static evaluation with the R2 metric

In this section, we consider a long-horizon aggregated R2 score $\overline{R2}(H)$ as our metric (details provided in appendix E.1.1). For the weights of the loss, we perform a grid search over β values, selecting the value giving the best $\overline{R2}(H)$ averaged over 3 cross-validation folds.

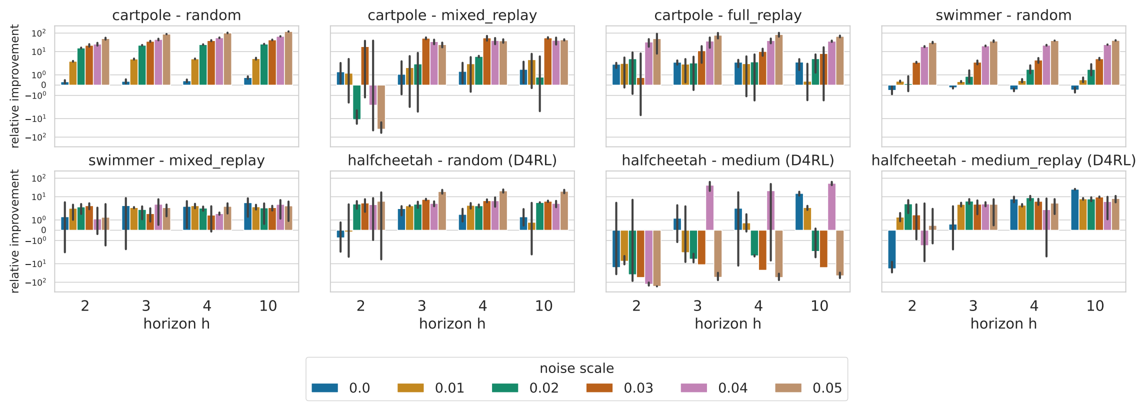


Figure 2: The series of bar plots display the relative improvement with respect to the $h = 1$ baseline, in the test $\overline{R2}(50)$ metric for various environments, and datasets. Performance is evaluated over loss horizons h with the relative improvement measured on a logarithmic scale. The relative noise scale, ranging from 0.0 to 0.05, is color-coded and represents a ratio of the range of the state space, for each dataset. The error bars indicate the 95% confidence intervals (mean $\pm$ 1.96 $\cdot$ standard error) obtained with the three cross-validation folds.

As can be seen from Fig. 2 For most of the datasets (all the *Cartpole* and *Swimmer* datasets, and *Halfcheetah random* and *medium_replay*) the benefit of using the multi-step loss when there is noise is significant and for most of them (all the *Cartpole* datasets, *Swimmer random* and *Halfcheetah random*) the larger the noise the higher the benefit. The impact of the horizon h of the loss is less clear although for some datasets increasing the horizon h as the noise increases also helps. This result is more mitigated on *Halfcheetah medium* which we suggest is due to the optimization process or the capacity of the model.

3.2 Dynamic evaluation: offline MBRL

We consider the offline setting where given a set of N trajectories $\mathcal{D} = \{(s_t^i, a_t^i, s_{t+1}^i, \ldots)\}_{i=1}^N$, the goal is to learn a policy maximizing the return in a single shot, without interacting with the environment.

We use a *Dyna*-style agent that learns a parametric model of the policy based on data generated from the learned dynamics model $\hat{p}_\theta$. Specifically, we train a Soft Actor-Critic (SAC) (Haarnoja et al., 2018) with short rollouts on the model à la MBPO Janner et al. (2019), a popular MBRL algorithm. We then rely on model predictive control (MPC) at decision-time where the action search is guided by the SAC policy. The details of this agent are given in Appendix C.3.

The experiments are run on the *Cartpole mixed_replay* dataset without and with small noise (scale of 1%). The reason is that the range of episode returns spanned by this dataset makes us hope that it is possible to learn a model that is good enough to learn a successful policy (for which episode returns are higher than 800). The distribution of the returns on the random and full replay datasets makes it more challenging to learn a successful policy. The goal here is not to study a new offline MBRL algorithm, avoiding unknown regions of the state-action space, but studying the improvement that can be obtained with the multi-step loss when varying the horizon h. Finally, in order to isolate the effect of dynamics learning, we assume that the reward function is a known deterministic function of the observations (which is another reason why we only consider cartpole for dynamic evaluation).

As it is acknowledged that static evaluation metrics may not always align with the final return of agents (Lambert et al., 2021), we include the performance of the models trained on the 10-step loss with weights tuned to maximize the return, using the same grid search as for the R2 (labeled *h=10 - return*).

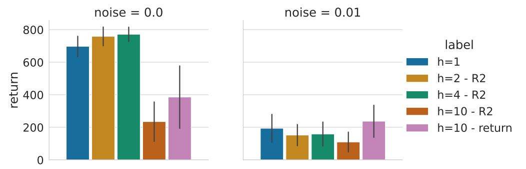


Figure 3: Returns of the agents trained on multi-step models in the *Cartpole mixed_replay* task. The error bars indicate the 95% confidence intervals (mean $\pm$ 1.96 $\cdot$ standard error).

Without noise. Fig. 3 shows the return obtained by different models on the *Cartpole mixed_replay* task. Against a strong baseline ($h = 1$), it is observed that the multi-step models exhibit marginally superior performance, especially for h values of 2 and 4. However, this trend is not maintained for $h = 10$, indicating that larger horizons may not be beneficial in this context, despite the improvement in the static metric (R2 score).

With noise. In the presence of noise, we observe that the performance of all models, including the baseline, significantly decreases. Notably, the optimal multi-step models do not demonstrate any improvement in this setting.

4 Discussion & Conclusion

In the previous experiments, we show that although $h = 10$-models improve over the smaller horizons in the R2 metric, this does not translate to better agents in the *Cartpole mixed_replay* task even when the weights are tuned on the return. Potential explanations of this finding include: the well-known objective mismatch in MBRL, the optimization difficulty induced by the autoregressive nature of the model, and the weights search space exponentially-growing with the horizon.

In this paper, we study a weighted multi-step loss that leads to models exhibiting significant improvements in the R2 score over popular RL environments with noisy dynamics. The insights from dynamic evaluation present a more complex picture. We found that longer horizons in the loss present a more challenging problem in the scope of Offline MBRL. To fully understand the observed behaviour, possible follow-up ideas include extending the dynamic evaluation to more RL environments, using a hyperparameters search engine for a more exhaustive weight search, and applying the multi-step prediction framework to other autoregressive models such as Recurrent Neural Networks or Transformers.

References

Pieter Abbeel, Varun Ganapathi, and Andrew Ng. Learning vehicular dynamics, with application to modeling helicopters. *Advances in Neural Information Processing Systems*, 18, 2005. URL https://proceedings.neurips.cc/paper/2005/hash/09b69adcd7cbae914c6204984097d2da-Abstract.html. Read.

Souhaib Ben Taieb and Gianluca Bontempi. Recursive Multi-step Time Series Forecasting by Perturbing Data, January 2012. URL https://ieeexplore.ieee.org/abstract/document/6137274.

Souhaib Ben Taieb, Gianluca Bontempi, Amir F. Atiya, and Antti Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Systems with Applications*, 39(8):7067–7083, June 2012. ISSN 0957-4174. doi: 10.1016/j.eswa.2012.01.039. URL https://www.sciencedirect.com/science/article/pii/S0957417412000528.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks, September 2015. URL http://arxiv.org/abs/1506.03099.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym, 2016.

Arunkumar Byravan, Leonard Hasenclever, Piotr Trochim, Mehdi Mirza, Alessandro Davide Ialongo, Yuval Tassa, Jost Tobias Springenberg, Abbas Abdolmaleki, Nicolas Heess, Josh Merel, and Martin A. Riedmiller. Evaluating model-based planning and planner amortization for continuous control. *CoRR*, abs/2110.03363, 2021. URL https://arxiv.org/abs/2110.03363.

Rohitash Chandra, Shaurya Goyal, and Rishabh Gupta. Evaluation of deep learning models for multi-step ahead time series prediction. *IEEE Access*, 9:83105–83123, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3085085. URL http://arxiv.org/abs/2103.14250. arXiv:2103.14250 [cs].

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31*, pp. 4754–4765. Curran Associates, Inc., 2018.

Klaus Fraedrich and Bernd Rückert. Metric adaption for analog forecasting. *Physica A: Statistical Mechanics and its Applications*, 253(1):379–393, 1998. ISSN 0378-4371. doi: https://doi.org/10.1016/S0378-4371(97)00668-7. URL https://www.sciencedirect.com/science/article/pii/S0378437197006687.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 10–15 Jul 2018.

Ferenc Huszár. How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?, November 2015. URL http://arxiv.org/abs/1511.05101.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Balázs Kégl, Alexandre Boucaud, Mehdi Cherti, Akin Kazakci, Alexandre Gramfort, Guillaume Lemaitre, Joris Van den Bossche, Djalel Benbouzid, and Camille Marini. The RAMP framework: from reproducibility to transparency in the design and optimization of scientific workflows. In *ICML workshop on Reproducibility in Machine Learning*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor Forcing: A New Algorithm for Training Recurrent Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/hash/16026d60ff9b54410b3435b403afd226-Abstract.html.

Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective mismatch in model-based reinforcement learning, 2021.

Nathan Lambert, Kristofer Pister, and Roberto Calandra. Investigating Compounding Prediction Errors in Learned Dynamics Models, March 2022. URL http://arxiv.org/abs/2203.09637. arXiv:2203.09637 [cs].

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Michael Lutter, Leonard Hasenclever, Arunkumar Byravan, Gabriel Dulac-Arnold, Piotr Trochim, Nicolas Heess, Josh Merel, and Yuval Tassa. Learning Dynamics Models for Model Predictive Agents, September 2021. URL http://arxiv.org/abs/2109.14311.

James McNames. Local averaging optimization for chaotic time series prediction. *Neurocomputing*, 48(1):279–297, 2002. ISSN 0925-2312. doi: https://doi.org/10.1016/S0925-2312(01)00647-6. URL https://www.sciencedirect.com/science/article/pii/S0925231201006476.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018*, pp. 7559–7566. IEEE, 2018.

Fernando J Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4(3):216–245, September 1988. ISSN 0885-064X. doi: 10.1016/0885-064X(88)90021-0. URL https://www.sciencedirect.com/science/article/pii/0885064X88900210.

Doina Precup and Richard S Sutton. Multi-time Models for Temporally Abstract Planning. In *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1997. URL https://proceedings.neurips.cc/paper_files/paper/1997/hash/a9be4c2a4041cadbf9d61ae16dd1389e-Abstract.html.

Doina Precup, Richard S. Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. In Claire Nédellec and Céline Rouveirol (eds.), *Machine Learning: ECML-98*, Lecture Notes in Computer Science, pp. 382–393, Berlin, Heidelberg, 1998. Springer. ISBN 978-3-540-69781-7. doi: 10.1007/BFb0026709.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404.

Satinder P. Singh. Scaling Reinforcement Learning Algorithms by Learning Variable Temporal Resolution Models. In Derek Sleeman and Peter Edwards (eds.), *Machine Learning Proceedings 1992*, pp. 406–415. Morgan Kaufmann, San Francisco (CA), January 1992. ISBN 978-1-55860-247-2. doi: 10.1016/B978-1-55860-247-2.50058-9. URL https://www.sciencedirect.com/science/article/pii/B9781558602472500589.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Richard S. Sutton. TD Models: Modeling the World at a Mixture of Time Scales. In Armand Prieditis and Stuart Russell (eds.), *Machine Learning Proceedings 1995*, pp. 531–539. Morgan Kaufmann, San Francisco (CA), January 1995. ISBN 978-1-55860-377-6. doi: 10.1016/B978-1-55860-377-6.50072-4. URL https://www.sciencedirect.com/science/article/pii/B9781558603776500724.

Richard S Sutton and Brian Pinette. The learning of world models by connectionist networks, 1985.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, August 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Erik Talvitie. Model Regularization for Stable Sample Rollouts. 2014.

Erik Talvitie. Self-Correcting Models for Model-Based Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), February 2017. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v31i1.10850. URL https://ojs.aaai.org/index.php/AAAI/article/view/10850.

Naoki Tanaka, Hiroshi Okamoto, and Masayoshi Naito Masayoshi Naito. An optimal metric for predicting chaotic time series. *Japanese Journal of Applied Physics*, 34(1R):388, jan 1995. doi: 10.1143/JJAP.34.388. URL https://dx.doi.org/10.1143/JJAP.34.388.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018. URL http://arxiv.org/abs/1801.00690.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Arun Venkatraman, Martial Hebert, and J.. Bagnell. Improving Multi-Step Prediction of Learned Time Series Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), February 2015. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v29i1.9590. URL https://ojs.aaai.org/index.php/AAAI/article/view/9590.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pp. 1–5, 2019.

Ronald J. Williams and David Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280, June 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270. URL https://ieeexplore.ieee.org/document/6795228. Conference Name: Neural Computation.

Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based reinforcement learning with theoretical guarantees. *CoRR*, abs/1807.03858, 2018. URL http://arxiv.org/abs/1807.03858.

Appendix

Table of Contents

A Related Work

The premises of multi-step dynamics modeling can be tracked back to early work about temporal abstraction (Sutton et al., 1999; Precup et al., 1998) and mixture of timescale models in tabular MDPs (Precup & Sutton, 1997; Singh, 1992; Sutton & Pinette, 1985; Sutton, 1995). These works study fixed-horizon models that learn an abstract dynamics mapping from initial states to the states j steps ahead. A different approach consists in optimizing the multi-step prediction error of single-step models that are used recursively, which we study here. This approach has been studied for recurrent neural networks (RNNs) and is referred to as *teacher forcing* (Lamb et al., 2016; Bengio et al., 2015; Huszár, 2015; Pineda, 1988; Williams & Zipser, 1989). The idea consists in augmenting the training data with predicted states. More recent works have built on this idea (Abbeel et al., 2005; Talvitie, 2014; 2017; Venkatraman et al., 2015). These methods, albeit optimizing for future prediction errors, assume the independence of the intermediate predictions on the model parameters, making them more of a data augmentation technique than a proper optimization of multi-step errors.

The closest works to ours which also consider the intermediate predictions to be dependent on the model parameters are Lutter et al. (2021), Byravan et al. (2021) and Xu et al. (2018). These works all use an equally-weighted multi-step loss whereas we emphasize on the need of having a weighted multi-step loss. Nagabandi et al. (2018) only use an equally-weighted multi-step loss for validation, which is also common in the time series literature (Tanaka et al., 1995; Fraedrich & Rückert, 1998; McNames, 2002; Ben Taieb & Bontempi, 2012; Ben Taieb et al., 2012; Chandra et al., 2021). Lutter et al. (2021) and Xu et al. (2018) find that only small horizons ($h = 2, 3, 5$) yield an improvement over the baseline, which we suggest is due to using equal weights in the multi-step loss. Byravan et al. (2021) successfully use $h = 10$ with equal weights in the context of model-predictive control (MPC). However, they consider it as a fixed design choice and might have tailored their approach accordingly.

B Understanding the multi-step loss: two case studies

B.1 Uni-dimensional linear system

The first case consists in studying the solutions of the multi-step loss for $h = 2$ in the case of an uncontrolled linear (discrete) dynamical system with additive Gaussian noise, and a linear model. With such a simple formulation, we benefit from the fact that the optimization is tractable and closed-form solutions can be obtained for each $\boldsymbol{\alpha} \in [0, 1]$. We start by defining the system and the model.

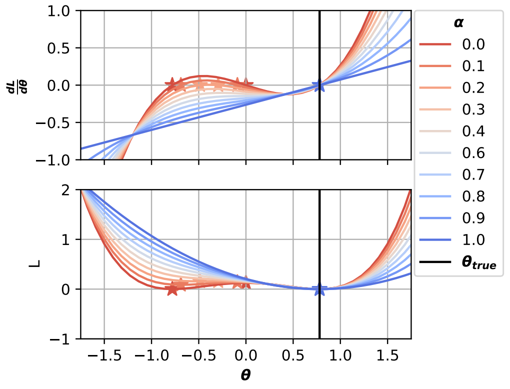


Figure 4: The loss function and its derivative for different values of θ and α, in absence of noise ($\sigma = 0$). In this figure, θ_{true} is fixed to a randomly selected value, $\theta_{true} = 0.78$. The roots of the derivative are highlighted with stars.

Definition (Uni-dimensional linear system with additive Gaussian noise). For an initial state $s_0 \in \mathbb{R}$ and an unknown parameter $\theta_{true} \in (-1, 1)$ (for stability) we define the transition function and observations as

$$s_{t+1} = \theta_{true} \cdot s_t$$
$$o_{t+1} = s_{t+1} + \epsilon_{t+1} \text{ with } \epsilon_{t+1} \sim \mathcal{N}(0, \sigma^2) \text{ and } \sigma \in \mathbb{R}$$

Definition (Linear model). For an initial state $s_0 \in \mathbb{R}$ and a parameter $\theta \in \mathbb{R}$ that we learn by minimizing the multi-step loss for $h = 2$, we define the linear model as $\hat{s}_{t+1} = \hat{p}_\theta(s_t) = \theta \cdot s_t$

In this setup, the multi-step loss for $h = 2$ boils down to a polynomial in the model's parameter θ:

$$L_\alpha\big(\boldsymbol{o}_\tau, \hat{p}_\theta(s_t)\big) = \alpha(\theta s_t - o_{t+1})^2 + (1 - \alpha)(\theta^2 s_t - o_{t+2})^2$$

where $\boldsymbol{o}_\tau = (o_{t+1}, o_{t+2})$. The aim of our study is to analyze the statistical properties of $\hat{\theta}(\alpha) \in \operatorname{argmin}_\theta L_\alpha\big(\boldsymbol{o}_\tau, \hat{p}_\theta(s_t)\big)$ for different values of α and different values of the observational noise scale[2] σ.

Fig. 4 shows the loss function curve and its critical points for different values of α. The minimizers $\hat{\theta}(\alpha)$ can be obtained by solving the polynomial equation $dL_\alpha/d\theta = 0$. When $\alpha \in (0, 1)$, we compute the roots of the cubic polynomial equation using Cardano's formulas. These latter include at least one real root ($\alpha \geq 0.3$ in Fig. 4) and two (potentially real) conjugate complex roots ($\alpha < 0.3$ in Fig. 4).

In the rest of the experiments, we fix a dataset of initial states $\mathcal{S}_0$ and sample K times a two-step transition, yielding a dataset $\mathcal{D} = (\mathcal{S}_0, \mathcal{O}_1^j, \mathcal{O}_2^j)_{j=1,\ldots,K}$. This dataset showcases different realizations of the observational noise, which is sampled i.i.d. from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

We then compare the distance between the minimizers of the loss function with different values of α and the true parameter θ_{true}. When there is more than one root, we assume access to the sign of θ_{true} so that we can choose the correct estimator $\hat{\theta}(\alpha)$. Fig. 5 shows that as the noise increases, the vanilla MSE loss estimator ($\alpha = 1$) is not the best estimator with respect to the distance to the true parameter θ_{true}. Interestingly, the best solution is obtained for $\alpha \in (0, 1)$.

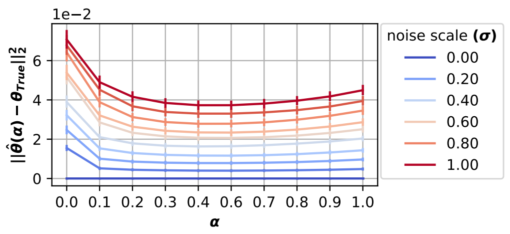


Figure 5: The impact of α and σ on the distance between θ_{true} and $\hat{\theta}$.

[2] *noise* and *noise scale* are used interchangeably. In practice, σ is computed as a percentage (e.g. 2%) of the state space width.

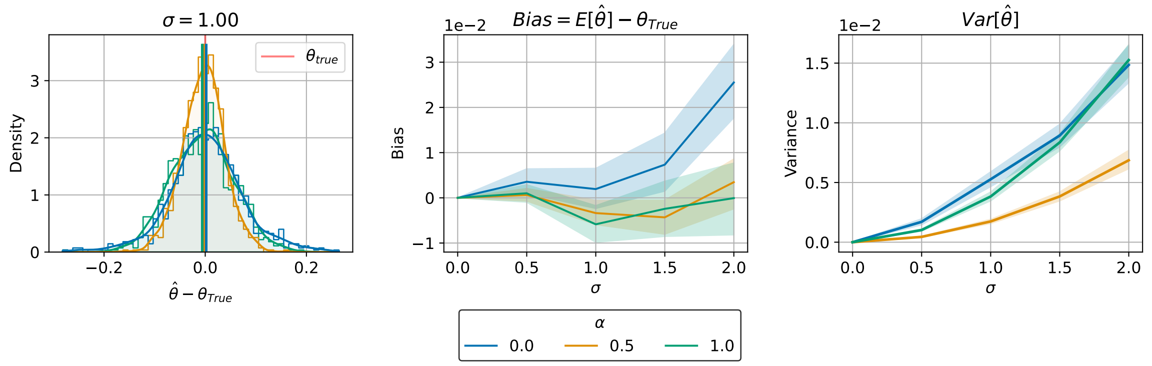


Figure 6: The left panel shows the density distribution of $\hat{\theta} - \theta_{true}$ for a fixed σ of 1.0. The middle panel delineates the bias of the estimator, defined as $E[\hat{\theta}] - \theta_{true}$, across varying levels of σ, and weights $\alpha \in \{0, 0.5, 1\}$ indicated by color. The right panel presents the variance of the estimator, $Var[\hat{\theta}]$, as a function of σ for the same set of α values. The shaded regions represent the 95% bootstrap confidence intervals across ten θ_{true} values and 100 Monte Carlo simulations.

To understand the observed results, we compute the
closed-form solutions of the multi-step loss in the case of $\alpha = 0$ and $\alpha = 1$:

Proposition ($\alpha = 1$). Given a transition ($s_t \neq 0, o_{t+1}$) from the linear system and a linear model with parameter θ, the minimizer of the $\alpha = 1$ multi-step loss can be computed as:

$$\hat{\theta}_1 = \frac{o_{t+1}}{s_t} = \theta_{true} + \frac{\epsilon_{t+1}}{s_t}$$

Proposition ($\alpha = 0$). Given a transition ($s_t \neq 0, o_{t+1}, o_{t+2}$) from the linear system, a linear model with parameter θ, the sign of the true parameter (for instance $\theta_{true} > 0$), and assuming its existence ($\frac{o_{t+2}}{s_t} > 0$), the minimizer of the $\alpha = 0$ multi-step MSE loss can be computed as:

$$\hat{\theta}_0 = \sqrt{\frac{o_{t+2}}{s_t}} = \sqrt{\theta_{true}^2 + \frac{\epsilon_{t+2}}{s_t}}$$

Remark For ease of notation in proposition ($\alpha = 0$) and proposition ($\alpha = 1$), we compute the solutions given only one transition (s_t, o_{t+1}, o_{t+2}). In practice, one minimizes the empirical risk based on a training dataset of size N: $\mathcal{D} = \{(s_{i,t}, o_{i,t+1}, o_{i,t+2})\}_{i=1:N}$, in which case the closed-form solutions become:

$$\begin{cases} \hat{\theta}_1 = \theta_{true} + \frac{\sum_{i=1}^N \epsilon_{i,t+1}}{\sum_{i=1}^N s_{i,t}} \\ \hat{\theta}_0 = \sqrt{\theta_{true}^2 + \frac{\sum_{i=1}^N \epsilon_{i,t+2} s_{i,t}}{\sum_{i=1}^N s_{i,t}^2}} \end{cases}$$

On the one hand, we observe that while $\hat{\theta}_1$ is an unbiased estimator of θ_{true} ($E_{\epsilon_{t+1} \sim \mathcal{N}(0,\sigma^2)}[\hat{\theta}_1] = \theta_{true}$), its variance grows linearly with the noise scale: $\text{Var}_{\epsilon_{t+1} \sim \mathcal{N}(0,\sigma^2)}[\hat{\theta}_1] = \frac{\sigma^2}{s_t^2}$. On the other hand, $\hat{\theta}_0$ is a potentially biased estimator, but with a smaller variance if $\theta_{true} \gg 1$ ($\frac{\epsilon_{t+2}}{4\theta_{true}^2 s_t^2} \approx 0$). In this case we can use a first-order Taylor expansion to approximate $\text{Var}[\hat{\theta}_0] = \text{Var}_{\epsilon_{t+2} \sim \mathcal{N}(0,\sigma^2)}[\hat{\theta}_0]$:

$$\begin{aligned} \text{Var}[\hat{\theta}_0] &= \text{Var}\Big[\sqrt{\theta_{true}^2 + \frac{\epsilon_{t+2}}{s_t}}\Big] \\ &\approx \theta_{true}^2 \text{Var}\Big[1 + \frac{\epsilon_{t+2}}{2\theta_{true}^2 s_t}\Big] = \frac{\sigma^2}{4\theta_{true}^2 s_t^2} \\ &\leq \text{Var}[\hat{\theta}_1] \end{aligned}$$

For intermediate models ($\alpha \in (0, 1)$), and in general when the conditions of the last result do not necessarily hold, we use Monte Carlo simulations to compare the variance of $\hat{\theta}_{\alpha\in\{0,0.5,1\}}$. Fig. 6 shows the variance reduction brought by the multi-step loss when $\alpha = 0.5$. It is also noticeable that, up to the noise scales considered in this experiment, $\alpha = 0$ generates a large bias (which matches the theoretical insights), while no significant bias is observed for the estimator with $\alpha = 0.5$. We conclude that in the case of a noisy linear system, the multi-step MSE loss minimizer with $\alpha = 0.5$ is a statistical estimator that (empirically) has a smaller variance and a comparable bias to the one-step loss minimizer. It is worth noticing that when the noise is non-zero the best solution for the one-step MSE is obtained for $\alpha \in (0, 1)$ and not $\alpha = 1$ (which corresponds to optimizing exactly the one-step MSE).

In this linear case, we had access to closed-form solutions. However, Fig. 4 shows that choosing the multi-step MSE loss as an optimization objective introduces additional critical points where a gradient-based optimization algorithm might get stuck. We now empirically study the optimization process in the case of a two-parameter neural network.

B.2 Two-parameter non-linear system

As an attempt to get closer to a realistic MBRL setup where neural networks are used for dynamics learning, we study a non-linear dynamical system using a two-parameters neural network model:

Definition (Two-parameter non-linear system with additive Gaussian noise). For an initial state $s_0 \in \mathbb{R}$ and unknown parameters $\boldsymbol{\theta}^{true} = (\theta_1^{true}, \theta_2^{true}) \in \mathbb{R}^2$ we define the transition function and observations as

$$s_{t+1} = \theta_1^{true} \cdot sigmoid(\theta_2^{true} \cdot s_t)$$
$$o_{t+1} = s_{t+1} + \epsilon_{t+1} \text{ with } \epsilon_{t+1} \backsim \mathcal{N}(0, \sigma^2) \text{ and } \sigma \in \mathbb{R}$$

Definition (Two-parameters neural network model). A single-neuron two-layer (without bias) neural network. We denote its parameters $\boldsymbol{\theta} = (\theta_1, \theta_2) \in \mathbb{R}^2$:

$$\hat{s}_{t+1} = \hat{p}(s_t) = \theta_1 \cdot sigmoid(\theta_2 \cdot s_t)$$

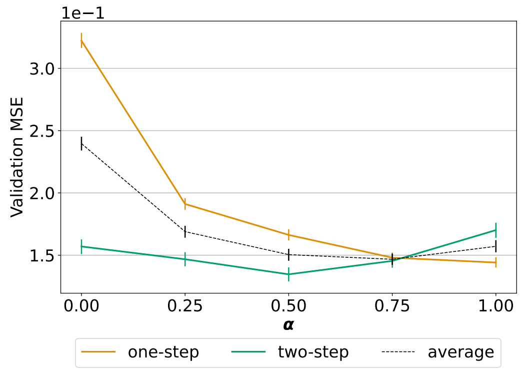
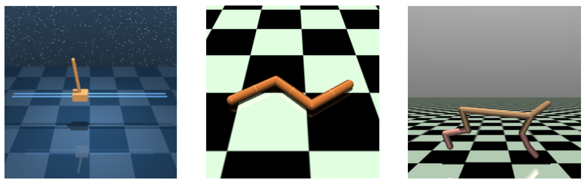


Figure 7: The validation one-step MSE L_1 (in yellow), the validation two-step MSE L_0 (in green) and the average of these two MSEs (dashed black line) for different values of α. The error bars represent the 95% bootstrap confidence intervals across 2 optimizers, 3 initialization distributions, 10 initial points, 3 noise levels, and 10 Monte Carlo simulations.

Fig. 7 shows that even in the presence of noise, compared to the linear case, the best value of α for the one-step MSE is $\alpha = 1$. This can be explained by the use of lower levels of noise in the simulations than the linear case. The intermediate models obtained for $\alpha \in \{0.25, 0.5, 0.75\}$ represent a trade-off between the one-step and two-step MSEs. The average of the two MSEs, which we use in the experiment section to assess the overall quality of our models over a range of horizons, achieves its minimum for $\alpha = 0.75$. Notice that the validation losses are only used for evaluation, and do not match the training loss which depends on α.

C The evaluation setup

C.1 Environments



(a) Cartpole (b) Swimmer (c) Halfcheetah

Figure 8: The environments: Cartpole swing-up, Swimmer and Halfcheetah.

In the present study, we examine three distinct environments within the scope of continuous control reinforcement learning, as delineated in Fig. 8, each exhibiting varying degrees of complexity. The complexity of a given environment is primarily determined by the dimension of the state space d_s, and the dimension of the action space d_a. Notably, *Cartpole swing-up* is a classic problem in the field where the task is to swing up a pole starting downwards, and balance it upright. The other considered tasks are *Swimmer* and *Halfcheetah*, which are two locomotion tasks, aiming at maximizing the velocity of a virtual robot along a given axis. *Swimmer* incorporates fluid dynamics with the goal of learning an agent that controls a multi-jointed snake moving through water. On the other hand, *Halfcheetah* simulates a two-leg Cheetah with the goal of making it run as fast as possible. For the latter environments, we use the implementation of OpenAI Gym (Brockman et al., 2016), and the implementation of Deepmind Control (Tassa et al., 2018) for *Cartpole*. Both libraries are based on the Mujoco physics simulator (Todorov et al., 2012). A detailed description of these environments is provided in Table 1.

Table 1: The environments characteristics. d_s: the dimension of the state space, d_a: the dimension of the action space, x_t: position along the x-axis, $\dot{x}_t$: velocity along the x-axis, $\|a_t\|_2^2$: the action a_t magnitude, θ: the angle of the pole (only for Cartpole).

environment	d_s	d_a	task horizon	reward function
Cartpole swing-up	5	1	1000	$\frac{1+\cos\theta_t}{2} \times \frac{1+e^{-0.25\log(10)x_t^2}}{2} \times \left(1 - \frac{a_t^2}{5}\right) \times \frac{1+e^{-0.04\log(10)\dot{x}_t^2}}{2}$
Swimmer	8	2	1000	$\dot{x}_t - 0.0001 \times \|a_t\|_2^2$
Halfcheetah	17	6	1000	$\dot{x}_t - 0.1 \times \|a_t\|_2^2$

C.2 Datasets

In this section, we introduce the different datasets that are used to evaluate the multi-step models. These datasets are collected using some *behavior policies* that are unknown to the models.

Table 2 illustrates the features of datasets across the three environments: Cartpole Swing-up, Swimmer, and Halfcheetah. These environments vary in dataset size and behavioral policies. In the Cartpole Swing-up setting, each of the three datasets (*random*, *mixed_replay*, and *full_replay*) includes 50 episodes, which are split into training, validation, and testing subsets. The and *full_replay* depict complete learning trajectories of an unstable model and a state-of-the-art (sota) model-based Soft Actor-Critic (SAC) respectively (Janner et al., 2019), and integrated with shooting-based planning Appendix C.3. For the Swimmer environment, both the random and *full_replay* datasets consist of 50 episodes each. The random dataset is derived from a random policy, whereas the *full_replay* dataset is generated using a model-based SAC with planning. For both the Cartpole Swing-up and Swimmer environments, the datasets were self-collected due to the absence of a unified benchmark that includes datasets from both tasks.

To enhance our understanding of the differences among these datasets, we present the distribution of returns for each dataset in Fig. 9. It is important to note that the variance in returns within a dataset serves as an indicator of the extent of the state space covered by that dataset. Specifically, datasets collected using a fixed policy exhibit a notably narrow distribution, predominantly concentrated around their mean values, as exemplified by the Halfcheetah *random* and *medium* datasets. This characteristic of the datasets significantly influences the out-of-distribution generalization error in offline MBRL, which represents a major challenge in this context.

Table 2: The datasets characteristics. *mf*: model-free, *mb*: model-based, random $\to \pi$: all episodes collected to learn the policy π. The datasets size is given in episodes (of 1000 steps each).

environment	dataset	size (train/valid/test)	behavior policy
Cartpole swing-up	random	50 (36/4/10)	random policy
	mixed_replay	50 (36/4/10)	random $\to$ unstable *mb* SAC + planning
	full_replay	50 (36/4/10)	random $\to$ *mb* SAC + planning
Swimmer	random	50 (36/4/10)	random policy
	mixed_replay	50 (36/4/10)	random $\to$ unstable *mb* SAC + planning
Halfcheetah	random (*D4RL*)	100 (76/4/20)	random policy
	medium (*D4RL*)	100 (76/4/20)	*mf* sac at half convergence
	medium_replay (*D4RL*)	200 (156/4/40)	random $\to$ *mf* sac at half convergence

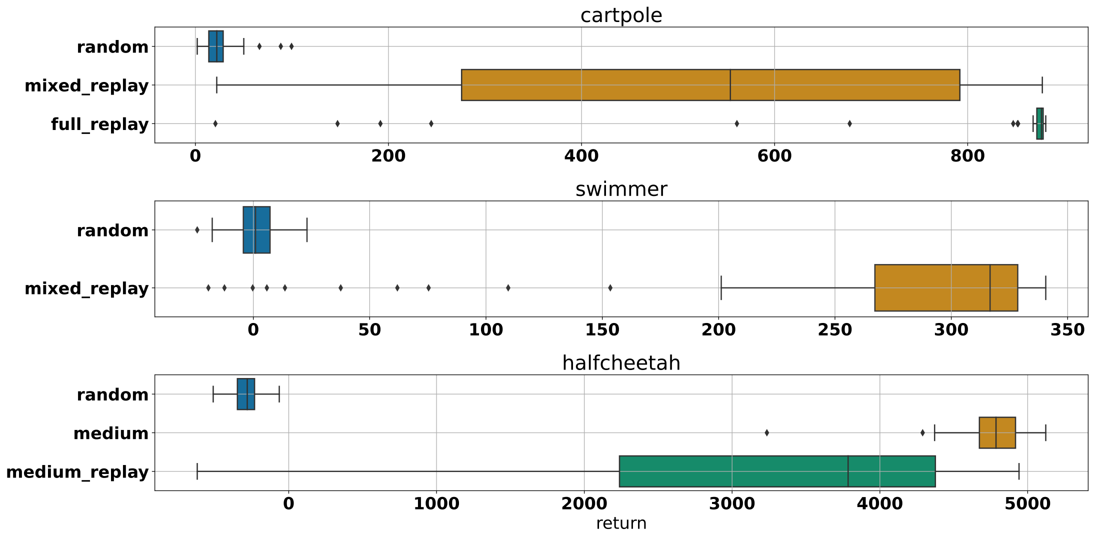


Figure 9: A comparison of the distribution of returns across the considered datasets.

C.3 Agent: SAC + planning

Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is an off-policy algorithm that incorporates the maximum entropy framework, which encourages exploration by seeking to maximize the entropy of the policy in addition to the expected return. SAC uses a deep neural network to approximate the policy (actor) and the value functions (critics), employing two Q-value functions to mitigate positive bias in the policy improvement step typical of off-policy algorithms. This approach helps in learning more stable and effective policies for complex environments, making SAC particularly suitable for tasks with high-dimensional, continuous action spaces.

In addition to *Dyna*-style training of the SAC agent on the learned model with short rollouts a la MBPO (Janner et al., 2019), we use Model Predictive Control (MPC). MPC is the process of using the model recursively to plan and select the action sequence that maximizes the expected cumulative reward over a planning horizon H. The set of K (population size) action sequences $\{(a^k_{t:t+H})_{k\in\{1,\dots,K\}}\}$ is usually generated by an evolutionary algorithm, e.g. Cross Entropy Method (CEM) (Chua et al., 2018). In this study, the pre-trained SAC guides the MPC process by generating candidate action sequences from the learned stochastic policy.

D Implementation details

For all the models, we use a neural network composed of a common number of hidden layers and two output heads (with *Tanh* activation functions) for the mean and standard deviation of the learned probabilistic dynamics (The standard deviation is fixed when we want to use the MSE loss). We use batch normalization (Ioffe & Szegedy, 2015), Dropout layers (Srivastava et al., 2014) ($p = 10\%$), and set the learning rate of the Adam optimizer (Kingma & Ba, 2015) to 0.001, the batch size to 64, the number of common layers to 2, and the number of hidden units to 256 based on a hyperparameter search executed using the RAMP framework (Kégl et al., 2018). The evaluation metric of the hyperparameter optimization is the aggregated one-step validation R2 score across all the offline datasets. The neural networks are trained to predict the difference between the next state and the current state $\Delta_{t+1} = s_{t+1} - s_t$. More precisely, the baseline consists in the single-step model trained to predict the difference Δ_{t+1} using the one-step MSE. The other multi-step models, take their own predictions as input to predict the difference $\Delta_{t+h} = s_{t+h} - \hat{s}_{t+h-1}$ at horizon h

For the offline RL experiments, we use SAC agents from the StableBaselines3 open-source library (Raffin et al., 2021) while keeping its default hyperparameters. In the offline setting, we train the SAC agents for $500,000$ steps on a fixed model by generating short rollouts of length 100 from states of the the dataset selected uniformly at random. At evaluation time, the MPC planning is done by sampling 500 action sequences from the SAC policy, and rolling out short rollouts of horizon 20 for return computation. This return is then bootstrapped with the value function learned by SAC.

E Details of the experiments

E.1 Static evaluation

E.1.1 The $\overline{R2}(H)$ metric

The commonly used metrics for the static evaluation are the standard mean squared error (MSE) or the explained variance (R2) which we prefer over the MSE because it is normalized and can be aggregated over multiple dimensions. In our attempt to reduce compounding errors in MBRL, we are especially interested in the long-term predictive error of models. For each horizon h, the error is computed by considering all the sub-trajectories of size h from the test dataset. The predictions are computed by calling the model recursively h times (using the ground truth actions of the sub-trajectories) and the average R2 score at horizon h (featuring the predictions $\hat{p}_\theta^h(s_t, \boldsymbol{a}_{t:t+h})$ and groundtruth states s_{t+h}), R2(h), averaged over the sub-trajectories is computed. We then report the average R2 score, $\overline{R2}(H)$, over all prediction horizons from 1 to H: $\overline{R2}(H) = \frac{1}{H}\sum_{h=1}^{H} \text{R2}(h)$.

E.1.2 $\overline{R2}(100)$ table

The static evaluation has been conducted by training models on the multi-step loss for different values of $h \in \{2, 3, 4, 10\}$ and $\beta \in \{0.1, 0.3, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 5.0, 20.0\}$. Table 3 shows the corresponding test $\overline{R2}(100)$ scores and their standard deviations after the symbol $\pm$. For each horizon h, we show the best value of the β parameter, as it's different across environments, datasets, and noise scales.

E.1.3 The weight profile and the effective horizon

We can define the effective horizon h_e that reflects the prediction horizon at which we effectively optimize the prediction error: given a weighted multi-step loss $L_{\boldsymbol{\alpha}}$ with nominal horizon h and weights $\boldsymbol{\alpha}$, the effective horizon h_e is defined as $h_e = \sum_{i=1}^{h} \alpha_i \cdot i$.

For a given model trained using the multi-step loss, the optimal effective horizon is an indication of the time scale needed for optimal performance. However, models that have a different nominal horizon h and the same effective horizon h_e do not necessarily have the same performance. Precisely, the loss with the larger nominal horizon is setting small weights on the furthest horizons, which has a direct impact on the loss landscape and consequently on the optimization process.

Table 4 shows the optimal effective horizon h_e, and the exponentially parametrized weight profiles (characterized by the decay parameter β), for different values of the horizon h and the noise scale σ.

Environment	Dataset	Noise scale	One-step	Multi-step			
				h=2 [β]	h=3 [β]	h=4 [β]	h=10 [β]
Cartpole swing-up	random	0	972 +- 4	975 +- 1 [0.1]	977 +- 2 [0.3]	980 +- 1 [0.1]	**986 +- 1 [0.75]**
		0.01	864 +- 5	930 +- 1 [2.0]	**950 +- 2 [2.0]**	946 +- 3 [1.0]	**954 +- 4 [0.75]**
		0.02	508 +- 16	690 +- 15 [2.0]	769 +- 7 [2.0]	812 +- 7 [1.5]	**836 +- 7 [0.75]**
	mixed_replay	0	812 +- 106	915 +- 22 [0.75]	921 +- 32 [2.0]	925 +- 24 [0.5]	931 +- 20 [0.75]
		0.01	541 +- 51	574 +- 26 [0.1]	659 +- 44 [3.0]	653 +- 51 [1.5]	679 +- 38 [0.75]
		0.02	428 +- 9	335 +- 4 [2.0]	459 +- 71 [0.75]	481 +- 12 [1.0]	457 +- 38 [0.75]
	full_replay	0	705 +- 52	828 +- 6 [2.0]	843 +- 42 [2.0]	858 +- 32 [0.75]	**880 +- 21 [0.5]**
		0.01	714 +- 7	709 +- 10 [0.1]	718 +- 6 [1.5]	722 +- 33 [0.75]	683 +- 32 [0.1]
		0.02	530 +- 14	551 +- 34 [0.1]	527 +- 21 [1.5]	530 +- 46 [0.1]	541 +- 24 [0.1]
Swimmer	random	0	**983 +- 1**	974 +- 3 [0.1]	978 +- 0 [0.1]	975 +- 1 [0.1]	974 +- 2 [0.1]
		0.01	934 +- 6	941 +- 0 [0.1]	**941 +- 2 [0.1]**	**942 +- 1 [0.5]**	**943 +- 2 [0.5]**
		0.02	865 +- 14	869 +- 17 [0.1]	880 +- 8 [0.5]	**892 +- 4 [0.5]**	**891 +- 4 [0.5]**
	mixed_replay	0	609 +- 46	680 +- 184 [20.0]	734 +- 18 [20.0]	**875 +- 25 [3.0]**	735 +- 120 [0.75]
		0.01	874 +- 14	904 +- 16 [0.5]	901 +- 16 [3.0]	**936 +- 5 [0.1]**	920 +- 8 [0.75]
		0.02	850 +- 13	893 +- 8 [1.0]	878 +- 6 [0.3]	886 +- 4 [1.0]	883 +- 8 [0.75]
Halfcheetah	random	0	755 +- 3	746 +- 7 [0.1]	**775 +- 3 [0.1]**	765 +- 9 [0.5]	766 +- 5 [0.3]
		0.01	737 +- 4	732 +- 27 [0.1]	756 +- 9 [0.1]	**763 +- 6 [0.1]**	725 +- 5 [0.5]
		0.02	704 +- 3	711 +- 17 [0.1]	709 +- 23 [0.1]	701 +- 3 [0.5]	**731 +- 0** [0.3]
	medium	0	516 +- 19	268 +- 51 [0.1]	547 +- 10 [0.3]	562 +- 63 [0.3]	**640 +- 13 [0.3]**
		0.01	691 +- 11	634 +- 20 [0.1]	674 +- 28 [0.1]	695 +- 3 [0.1]	**710 +- 4 [0.1]**
		0.02	**718 +- 4**	422 +- 242 [0.1]	675 +- 15 [0.1]	690 +- 1 [0.1]	702 +- 13 [0.1]
	medium_replay	0	541 +- 54	446 +- 70 [3.0]	589 +- 15 [0.75]	632 +- 7 [0.75]	**771 +- 12 [0.5]**
		0.01	709 +- 12	725 +- 4 [0.1]	737 +- 7 [0.1]	737 +- 16 [0.1]	**773 +- 9 [0.3]**
		0.02	687 +- 26	723 +- 16 [0.1]	731 +- 6 [0.1]	765 +- 18 [0.3]	**750 +- 7 [0.3]**

Table 3: $\overline{R2}(100)$ for different environments, datasets, and noise scales. We highlight entries that have significantly larger score. In addition, to the mean ^{p}m standard deviation of the reported metric, the table also shows the best β selected for each loss horizon h.

Table 4: Best $h_e(\beta)$ values found with a grid search for each horizon and each noise scale. The values are averaged over the eight datasets.

	horizon h			
	2	3	4	10
σ (%)	h_e (β)			
0.0	1.30 (0.81)	1.48 (0.45)	1.58 (0.41)	2.09 (0.46)
0.01	1.26 (0.56)	1.65 (0.76)	1.74 (0.51)	2.23 (0.47)
0.02	1.36 (0.86)	1.65 (0.72)	2.08 (0.78)	2.72 (0.54)
0.03	1.33 (0.67)	1.95 (1.21)	2.13 (0.81)	2.33 (0.50)
0.04	1.40 (0.74)	1.88 (1.01)	2.02 (0.74)	2.49 (0.55)
0.05	1.49 (1.33)	1.76 (0.83)	2.28 (0.97)	2.22 (0.51)

The main insight of Table 4 is that regardless of the nominal horizon h, the effective horizon h_e (and equivalently the decay parameter β) increases with the noise scale. This finding supports the idea that multi-step models are increasingly needed when incorporating information from the future is crucial to achieve noise reduction. As discussed in the previous experiment, the results are highly dependent on the task (environment/dataset), while in Table 4 we aggregate the results across tasks, and still observe the increasing trend.

Another important result highlighted in Table 4 is the upper bound on the effective horizon (h_e does not go beyond 2.52), even when the nominal horizon is large (e.g 10). This suggests that while putting weight on future horizons error does help the model, it is not beneficial to fully optimize for these horizons. Indeed, the additional components of the multi-step MSE loss act as a regularizer to the one-step loss, rather than a completely different training objective.