Bayesian Meta-Reinforcement Learning with Laplace Variational Recurrent Networks

Joery A. de Vries
Delft University of Technology
`J.A.deVries@tudelft.nl`

Jinke He
Delft University of Technology
`J.He-4@tudelft.nl`

Mathijs M. de Weerdt
Delft University of Technology
`M.M.deWeerdt@tudelft.nl`

Matthijs T. J. Spaan
Delft University of Technology
`M.T.J.Spaan@tudelft.nl`

Abstract

Meta-reinforcement learning trains a single reinforcement learning algorithm on a distribution of tasks to quickly generalize to new tasks outside of the training set at test time. From a Bayesian perspective, one can interpret this as performing amortized variational inference on the posterior distribution over training tasks. Among the various meta-reinforcement learning approaches, a common method is to represent this distribution with a point-estimate using a recurrent neural networks. We show how one can augment this point estimate to give full distributions through the Laplace approximation, either at the start of, during, or after learning, without modifying the base model architecture. With our approximation, we are able to estimate distributional statistics (e.g., the entropy) of non-Bayesian agents and observe that point-estimate based methods produce overconfident estimators while not satisfying consistency. Furthermore, when comparing our approach to full-distribution based learning of the task posterior, we found our method to perform on par with variational inference baselines despite being simpler to implement.

1 Introduction

Reinforcement Learning (RL) concerns itself with making optimal decisions from data (Sutton & Barto, 2018). This is typically achieved by letting an agent generate data in an environment and then optimizing a cost function of the agent's parameters given this data. In meta-RL, an agent is trained to optimize an expected cost over a prior distribution of environments (Finn et al., 2017; Chen et al., 2017; Beck et al., 2023). The idea is then that, given a trajectory of new data, an agent can infer latent environment parameters and successfully adapt its action policy online. This is known as zero-shot or few-shot adaptation or learning (Beck et al., 2023). In recent years, this paradigm has shown impressive results, for example, by the Capture the Flag agent (Jaderberg et al., 2019) or the Adaptive Agent (Bauer et al., 2023).

In any meta-RL algorithm, an accurate approximation of the latent parameter distribution given data, also known as the task posterior distribution, is useful to quantify the agent's uncertainty (Grant et al., 2018). Accurate quantification of uncertainty enables agents to detect distribution shifts (Daxberger et al., 2021) or guide exploration through novelty signals (Osband et al., 2013; Sekar et al., 2020). Importantly, on deployment, distribution shift detection is essential for timely human intervention or model-retraining. This ultimately improves the robustness and efficacy of our algorithms and allows us to more reliably inspect failure cases.

17th European Workshop on Reinforcement Learning (EWRL 2024).

A common approach in meta-RL is to model the task posterior with point estimates, e.g., using the hidden state of recurrent neural networks (RNN) (Chen et al., 2017). However, this prevents us from exploiting useful distributional statistics. Another downside of using point-estimates is the increased risk of overconfidence unless the true posterior is sharply peaked at that particular point. This would imply that there exists almost no uncertainty about the environment, which is typically a strong and unrealistic assumption. As a consequence, point-estimate based meta-RL has been known to overfit to its training distribution, leading to brittle downstream performance (Xiong et al., 2021; Greenberg et al., 2023).

Arguably, a better approach would be to explicitly parameterize some full distribution (e.g., a Gaussian) (Chung et al., 2015; Zintgraf et al., 2021). However, approximate Bayesian methods are slower to train and are still often outperformed by simple point-estimate methods in terms of expected returns (Greenberg et al., 2023) even though they model the posterior more accurately. This could be explained by the fact that non-Bayesian methods enjoy reduced sampling noise, easier numerical representation, and improved model capacity by not having to learn a complex posterior model (Goyal et al., 2017; Hafner et al., 2019).

To get benefits from Bayesian methods when using non-Bayesian models, we introduce the *Laplace variational recurrent neural network* (Laplace VRNN) which utilizes the Laplace approximation to extend RNN-based meta-reinforcement learning. Our method can perform uncertainty quantification for non-Bayesian meta-RL agents without modifying the model architecture or loss function, and without needing to retrain any parameters. In other words, the consequence of the Laplace approximation is that we can apply it at any point during model training. When applied after training, this is often referred to as a *post-hoc* posterior (Daxberger et al., 2021). This allows us to make use of deterministic pre-training schedules and benefit from their aforementioned advantages while also enjoying the benefits of Bayesian methods. Although the Laplace approximation has already been explored in meta-RL (Grant et al., 2018; Finn et al., 2018), it has not been applied in memory-based methods (Duan et al., 2016; Beck et al., 2023) which is what we explore.

The Laplace approximation is a simple method that only requires the *curvature* of a distribution's log-likelihood at a local maximum (Daxberger et al., 2021). For a Gaussian mean-field assumption on our posterior model (Bishop, 2007), we only require the Jacobian matrix of the RNN output with respect to its hidden state. This gives us a Gaussian distribution for the task posterior distribution centered at the RNN hidden state with inverse covariance equal to the sum of Jacobian outer products. This is a comparatively cheap approximation compared to typical methods that apply the Laplace approximation to the much higher dimensional neural network parameters (Grant et al., 2018; Daxberger et al., 2021; Martens & Grosse, 2015).

We empirically validate that our method can reliably estimate posterior statistics of our non-Bayesian baselines without degrading performance on supervised and reinforcement learning domains. Similarly to Xiong et al. (2021), our results show that non-Bayesian meta-RL agents do not learn consistent estimators, however, we also find that the learned representations are overconfident. This could be seen by inspecting the *post-hoc* posterior provided by the Laplace approximation, which showed low entropy while not converging to a stable distribution. Furthermore, when comparing our method against variational inference baselines, we find that our Laplace method performs at least as well in terms of mean returns. Ultimately, this shows that the Laplace approximation can be an useful alternative (or complement) to variational inference methods for uncertainty quantification, since we do not *learn* our local uncertainty but estimate this based on the model's fitted parameters.

2 Related Work

Meta-Reinforcement Learning. Meta-learning has been described from various viewpoints, ranging from contexts (Sodhani et al., 2022), latent-variable models (Garnelo et al., 2018; Wu et al., 2020; Gordon et al., 2018), or biological analogs of learning to learn (Beck et al., 2023; Wang et al., 2017; Hospedales et al., 2020). Applying these ideas to reinforcement learning has been gaining traction within the field recently, for example, learning maximum likelihood estimation algorithms (like our work) (Andrychowicz et al., 2016; Garnelo et al., 2018), probability density functions (Lu et al., 2022; Bechtle et al., 2021), or model exploration strategies (Gupta et al., 2018).

Related to our work is the neural process by Garnelo et al. (2018) which formalizes using meta-learning to infer a set of (global) latent variables for a generative distribution. Since we test on reinforcement learning problems, one could view our model as a type of non-stationary stochastic process or sequential neural process (Øksendal, 2003; Singh et al., 2019). This makes our model and optimization objective similar to the PlaNet model (Hafner et al., 2019), however, we do not condition our recurrent model on samples from the task posterior so we can obtain an analytical solution. This choice does reduce the non-linearity of our model, the topic of linear vs. non-linear state space models is still active research (Gu & Dao, 2023).

Bayesian Reinforcement Learning. Learning an optimal control policy conditional on a task-posterior amounts to approximating the Bayes-adaptive optimal policy (Duff, 2002). In this framework an agent is conditioned on its current state and a history of observations, the history can then be used to produce a belief distribution over latent variables. The Bayes-adaptive optimal policy maximizes the environment returns in expectation over this belief (Duff, 2002; Ghavamzadeh et al., 2015; Zintgraf et al., 2021; Mikulik et al., 2020). Although meta-learning induces uncertainty only over the reward and transition function, many have also successfully tackled the problem as a general partially observable Markov decision process (Chen et al., 2017; Bauer et al., 2023). Doing this is orthogonal to our method, however, we focus on the Bayes-adaptive framework for simplicity.

Laplace Approximation. The Laplace approximation has been explored for meta-learning in, for example, the model agnostic meta-learning algorithm (Finn et al., 2017, 2018) to achieve more accurate inference, or for continual learning (Kirkpatrick et al., 2017) as a regularizer for weight-updates. The main obstacle to using the Laplace approximation in practice is the computation of the inverse Hessian, which alone has quadratic memory scaling in the number of model parameters. For this reason, in Bayesian neural networks, the block-diagonal factorization has become quite popular (Martens & Grosse, 2015), as used by TRPO (Schulman et al., 2015) or second order optimizers (Botev et al., 2017). Our method bypasses the costly Hessian problem by only modeling a distribution on a small subset of the full model's parameter set. In contrast to doing Bayesian linear regression on the last layer of a neural network, our method can express multimodal distributions.

3 Preliminaries

We want to find an optimal policy π for a sequential decision-making problem, which we formalize as an episodic Markov decision process (Sutton & Barto, 2018). We define states $S \in \mathcal{S}$, actions $A \in \mathcal{A}$, and rewards $R \in \mathbb{R}$ as random variables that we sample in sequences. We write $H^i = \{S_t, A_t, R_t\}_{t=1}^T$ to abbreviate the joint random variable of episode $i \in \mathbb{N}$,

$$p(H^i) = \prod_{t=1}^{T} p(R_t|S_t, A_t)\pi(A_t|S_t)p(S_t|S_{t-1}, A_{t-1}),$$

where $p(S_1|A_0, S_0) \triangleq p(S_1)$ is the initial state distribution, $\pi(A_t|S_t)$ is the policy, $p(S_{t+1}|S_t, A_t)$ is the transition model, and $p(R_t|S_t, A_t)$ is the reward model. To avoid confusion, we denote episodes in the *superscript* from $i = 1, \ldots, n$ and time in the *subscripts* from $t = 1, \ldots, T$. For convenience, we subsume the common discount factors $\gamma \in [0, 1]$ into the transition probabilities as a global termination probability of $p_{\text{term}} = 1 - \gamma$ (Levine, 2018) assuming that the MDP will end up in an absorbing state with zero rewards. The objective is to find π^* such that $\mathbb{E}_{p(H)} \sum_t R_t$ is maximized.

3.1 Meta-Reinforcement Learning

In contrast to single-task reinforcement learning (RL), in meta-RL we want to find the optimal policy π^* to a distribution over different environments. The agent typically does not know which environment it is currently being deployed in and needs to adaptively switch strategies based on online feedback. We will assume that the agent can adapt over *multiple* episodes $H^{1:n}$, as opposed to only one episode H^1. This is also known as zero-shot or few-shot adaptation (Beck et al., 2023). We can formalize this idea using the concept of a global latent variable Z. Each trajectory H that we sample depends on a sampled latent variable $Z \sim p(Z)$, in its most simple form this could be interpreted as a unique identifier of the true environment. Our agent does not directly observe Z, but this variable influences the reward and the transition models of the environment.

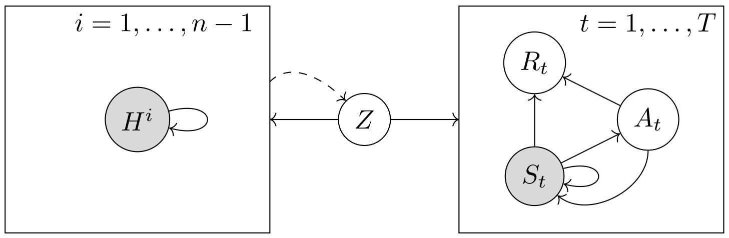

Figure 1: Simplified graphical model for the sampling distribution of a meta-reinforcement learning agent given policy π (not optimal per se). All trajectories H^i are sequentially dependent since the agent uses past data $H^{<n}$ to infer the latent variable Z to generate a new trajectory H^n. The arrows going in and from the squares indicate that all variables inside that box depend on Z.

So, in meta-RL, we can extend the true generative process according to,

$$p(H^{1:n}, Z^{1:n}) = \prod_{i=1}^{n} p(H^i|Z^i)p(Z^i|Z^{<i}, H^i), \tag{1}$$

where the first term on the right hand side indicates the sampling distribution of the environment under our current model for Z, and the second term denotes the (non-stationary) posterior distribution over latent variables Z given all the data we have observed so far. For brevity, we do not expand the sampling distribution $p(H^i|Z^i)$ here (see Appendix A.1), however, the above expression hides that we can update our posterior model for Z^i given data *inside* the episode H^i. Finally, note that this particular distribution is the most general form for meta-RL, resembling a continual-RL like objective (Khetarpal et al., 2022). In practice we often simplify the true distribution for $p(Z^i)$ such that Z^i is i.i.d. although our agent could still model this more generally.

In the stationary case $p(Z^i)$, we can sample data from the true process by sampling an initial Z_0, fixing this value, and then sequentially sampling the environment traces $H^{1:n}$ with a history-dependent agent (to model the belief over Z^i). This special case for the factorization is shown in the graphical model of Figure 1, where we have dropped the dependence on $Z^{<i}$ and, for brevity, inter-episodic data. In other words, this model assumes stationarity of Z and that inference on Z is done offline.

For our agent, the posterior $p(Z^i|Z^{<i}, H^i)$ is usually intractable. Some meta-learning frameworks deal with this by learning an approximate posterior and updating this model online using maximum a posteriori methods (Finn et al., 2017), instead we will amortize the computation by predicting a parametric variational distribution q_θ as typically done in memory-based meta-learning (Wu et al., 2020; Duan et al., 2016; Wang et al., 2017). This gives us a lower bound on the data log-likelihood through Jenssen's inequality (Appendix A.1),

$$\ln p(H^{1:n}) \geq \mathbb{E}_{q_\theta(Z^{1:n}|H^{1:n})} \sum_{i=1}^{n} \ln p(H^i|Z^i) - KL\left(q_\theta(Z^{\leq i}|H^{<i}) \| p(Z^{\leq i}|H^{<i})\right), \tag{2}$$

for some *fixed* policy π (not necessarily optimal).

3.2 Control as Inference

To define optimal behaviour in our generative model we require the control as inference framework (Levine, 2018). This framework redefines the classical RL description by defining a prior distribution over trajectories $p(H^i)$ and conditioning this distribution on a desired outcome $\mathcal{O}$. We will write this outcome $\mathcal{O}$ as a binary variable indicating whether a trajectory is optimal or not. Then, we can write the likelihood of being optimal under a trajectory H^i as the exponentiated sum of rewards, $p(\mathcal{O} = 1|H^i) \propto \exp(\sum_t R_t^i)$. Using Bayes rule, we can then infer the optimal policy either by directly estimating $p(H^i|\mathcal{O} = 1)$ or by maximizing an evidence lower bound for $\ln p(\mathcal{O} = 1)$.

Similarly to the previous section, we define a lower bound to the log-likelihood of the outcome variable $\ln p(\mathcal{O} = 1|H^{1:n})$. For this, we change the lower bound for $\ln p(H^{1:n})$ to include a parametric variational distribution for the prior policy π_θ. We then marginalize over the joint distribution of data and latent variables $p(H^{1:n}, Z^{1:n})$. Although this is enough to give us an objective for θ, we simplify the lower-bound to give us (Appendix A.1.2),

$$\mathcal{L}(\theta) = \mathbb{E}_{q_\theta(H^{1:n}, Z^{1:n})} \sum_{i=1}^{n} \sum_{t=1}^{T_i} R_t^i - KL\left(\pi_\theta(A_t^i | S_t^i, Z_t^i) \| \pi_{\theta_{old}}(A_t^i | S_t^i, Z_t^i)\right) \tag{3}$$
$$- \beta \cdot KL\left(q_\theta(Z_t^i | Z_{<t}^i, H_{<t}^i, Z^{<i}, H^{<i}) \| \texttt{stop\_grad}[q_\theta(Z_{t-1}^i | Z_{<t-1}^i, H_{<t-1}^i, Z^{<i}, H^{<i}]) \right)$$

which is an extension of the objective by Abdolmaleki et al. (2018) to include the task-posterior KL-penalty. Note that we substituted the KL-divergences to depend on a previous policy $\pi_{\theta_{old}}$ and the previous latent-variable posterior $q_\theta(Z_{t-1}| \dots)$ instead of their true likelihoods to make these terms more practical to compute. We also scale the posterior KL with the hyperparameter β to account for arbitrary differences in loss scale. The functional `stop_grad`$[\cdot]$ indicates an application of the stop-gradient operation, which treats the input as a *constant* such that the gradient $\nabla_\theta \mathcal{L}(\theta)$ with respect to this operator's input becomes zero. The effect is that our posterior q_θ is not optimized to fit data past its current timestep t. The final objective $\mathcal{L}(\theta)$ is easy to optimize using end-to-end differentiation and sampling, in practice one only needs to add the posterior KL-penalty over the trajectory to the loss of an existing recurrent policy search algorithm (Ni et al., 2022). Of course, for a point-estimate posterior, we must ignore this penalty term since it is not well defined.

4 Laplace Variational Recurrent Neural Networks

We introduce the Laplace variational recurrent neural network (Laplace VRNN) to make a relatively simple approximation to the variational task posterior $q_\theta \approx \hat{q}_\theta$, to be used in the lower-bounds of Eq. (2) and Eq. (3), using the Laplace approximation (Bishop, 2007; Daxberger et al., 2021). Although the posterior q_θ depends on the full history and past latent variables (c.f., Eq. (1)), we make the degree of non-stationarity for the approximate posterior model a tunable parameter. So, for the sake of presentation, we will introduce our approximation starting from a simpler variational distribution $q_\theta(Z_t | H_{<t})$, dropping superscripts. The complete derivation is given in Appendix A.3.

The Laplace VRNN first needs a deterministic helper variable ϕ such that our posterior is defined as $q_\theta(Z_t | H_{<t}) = q_\theta(Z | H_{<t}, \phi_t = f_\theta(H_{<t}))$. One can interpret the mapping $f_\theta : H \mapsto \phi$ as a learned *summary* statistic of $H_{<t}$ to more efficiently compute the distribution of Z_t. In principle, ϕ could be a quantile, mean, or higher moment estimate, but the posterior still depends on all the data despite this statistic. For practicality, we can compute ϕ autoregressively with a recurrent neural network (RNN) such that $\phi_{t+1} = f_\theta(S_t, A_t, R_t; \phi_t)$.

We then factorize this variational model using a type of mean-field assumption,

$$q_\theta(Z_t | H_{<t}, \phi_t) = \frac{1}{q_\theta(Z_t|\phi_t)^{t-2}} \prod_{i=1}^{t-1} q_\theta(Z_t | S_i, R_i, A_i, \phi_t), \qquad \text{(Lemma 1; Appendix A.2)}$$
$$= \exp\left[(2-t) \ln q_\theta(Z_t|\phi_t) + \sum_{i=1}^{t-1} \ln q_\theta(Z_t | S_i, R_i, A_i, \phi_t)\right]$$
$$= \exp h(Z_t; H_{<t}, \phi_t),$$

which gives us the target function h that we wish to approximate. Given data $H_{<t}$, this is done through the second order Taylor expansion of $h \approx \hat{h}$ linearized at $\phi = \phi_t$. Finally, we exponentiate $\hat{h}$ and renormalize it to integrate to 1. Assume that ϕ_t is the argument to a local mode of $q_\theta(Z_t | H_{<t}, \phi_t)$, then we recover the Laplace approximation (Daxberger et al., 2021),

$$q_\theta(Z_t | H_{<t}) \approx \frac{\exp \hat{h}(Z_t; H_{<t}, \phi_t)}{\int \exp \hat{h}(z; H_{<t}, \phi_t) dz} = \mathcal{N}(\mu_t = \phi_t, \Lambda_t = -\nabla_\phi^2 \ln q_\theta(Z_t | H_{<t}, \phi)|_{\phi=\phi_t}),$$

where $\nabla_\phi^2 \ln q_\theta$ is the Hessian of our log-posterior with respect to ϕ.

To complete our model, we continue with the lower bound of Eq. (2) from the previous section. We solve the expectation over $\mathbb{E}_{q_\theta(Z_{<t}|H_{<t})}$ for the posterior $q_\theta(Z_t | H_{<t}, Z_{<t})$, at each time-step t, by assuming a convolution of Gaussian densities. The result of this convolution is well-known to be another Gaussian with summed parameters (Bromiley, 2003),

$$q_\theta(Z_t | H_{<t}) = \mathcal{N}\left(\mu_t = \phi_t + \sum_{i=1}^{t-1} \mu_i, \Lambda_t = -\nabla_\phi^2 \ln q_\theta(Z | H_{<t}, \phi)|_{\phi=\phi_t} + \sum_{i=1}^{t-1} \Lambda_i\right)$$

In practice, however, we will heuristically drop the dependence on the complete past by only managing a smaller window $H_{k:t-1}$ and $Z_{k:t-1}$ for computational efficiency. In summary, this implements an RNN where we sum the last k hidden states and covariances for the output-Gaussian, and where the covariances are produced by the Hessian of the log-posterior w.r.t. the hidden state.

The assumption that ϕ_t is a mode is quite strict as this requires our mapping $f_\theta : \phi_{t-1} \mapsto \phi_t$ to be maximum a posteriori. If not, the first-order term in the Taylor expansion induces a location shift resulting in a worse approximation. However, since θ is optimized end-to-end to maximize the log-likelihood $p(\mathcal{O} = 1)$ through the lower bound (as given in Eq. (3)), the posterior q_θ is fitted such that Z indirectly maximizes this term. This makes the local maxima assumption at least reasonable.

Special Case: Gaussian Assumption. Our method obtains a particularly nice form if we choose $q_\theta(Z_t|S_i, A_i, R_i, \phi_t)$ to be standard Gaussian and use an uninformative prior for $q_\theta(Z_t|\phi_t)$ (an infinite variance-Gaussian). In that case, the Hessian of the log-posterior $q_\theta(Z_t|H_{<t}, \phi_t)$ becomes a sum of outer products of our RNN state Jacobians w.r.t. ϕ. This gives the inverse covariance,

$$\Sigma_t^{-1} = \sum_{i=1}^{t-1} (\nabla_\phi f_\theta(S_i, A_i, R_i; \phi)|_{\phi=\phi_t})(\nabla_\phi f_\theta(S_i, A_i, R_i; \phi)|_{\phi=\phi_t})^\top, \quad \text{(Prop. 2; Appendix A.3)}$$

which is relatively cheap to evaluate with forward-mode differentiation (Bradbury et al., 2018).

Posterior Predictive. If we would now use an policy $\pi(A_t|S_t, Z_t)$ that is linear in Z_t, then our full model would recover a type of Gaussian process (Immer et al., 2021; Rasmussen, 2004). However, we will model this term with another neural network $\pi_\theta(A_t|S_t, Z_t)$ to improve model expressiveness. Our policy conditional on an observed history $H_{<t}$ is then defined by the *posterior predictive* $\pi_\theta(A_t|S_t, H_{<t})$, which due to the use of neural networks must be approximated with Monte-Carlo,

$$\int \pi_\theta(A_t|S_t, z_t) q_\theta(z_t|H_{<t}) dz_t \approx \frac{1}{k} \sum_{i=1}^{k} \pi_\theta(A_t|S_t, Z_t = z^{(i)}), \quad z^{(i)} \sim q_\theta(Z_t|H_{<t}),$$

overloading the superscripts to index the Monte-Carlo samples. This induces a finite mixture for the model output where $k = 1$ corresponds to posterior sampling (Osband et al., 2013).

Interestingly, during training we found output aggregation to train much more stably in the loss when $k > 1$. Therefore, during optimization we chose to average the predicted logits of π_θ over samples $z^{(i)}$, or in the continuous case, we average over the parameters of a parametric distribution (Wang et al., 2020), e.g., the mean and variance of a Gaussian (see Appendix B.3 for a discussion).

5 Experimental Validation

We evaluated our proposed Laplace VRNN to answer the following questions:

1. **Utility:** Does the Laplace approximation offer useful posterior distribution statistics for our non-Bayesian baseline?
2. **Performance:** When used as an alternative to variational inference, does the Laplace VRNN perform at least on par with existing methods?
3. **Sensitivity:** What model assumptions for the Laplace VRNN are empirically effective?

Our point-estimate (RNN) baseline was implemented with a long-short term memory architecture (Hochreiter & Schmidhuber, 1997). The VRNN baseline (Chung et al., 2015) extends the RNN by predicting the mean and covariance for a Gaussian distribution as a transformation of the RNN output. For the RNN and VRNN we assumed a stationary factorization of the posterior $q_\theta(Z_t|H_{t-1})$, which is a simplification of the fully general posterior shown in Eq. (1) and most accurate to the true generative process. For all experiments, we intermittently created model snapshots of the point-estimate baseline (RNN) and finetuned these snapshots over our parameter grid for the Laplace VRNN and VRNN.

All experiments were repeated over $r = 30$ distinct seeds (number of network initializations), we tested intermediate model parameters by measuring their in-distribution performance and model statistics for $B = 128$ samples (number of test-tasks). We report 2-sided confidence intervals with a confidence level $\alpha = 0.99$ for each metric X aggregated over the seeds r and the test-tasks B. For the full details on the experiment and baseline setup, see Appendix B (code available at `https://github.com/joeryjoery/lvrnn`).

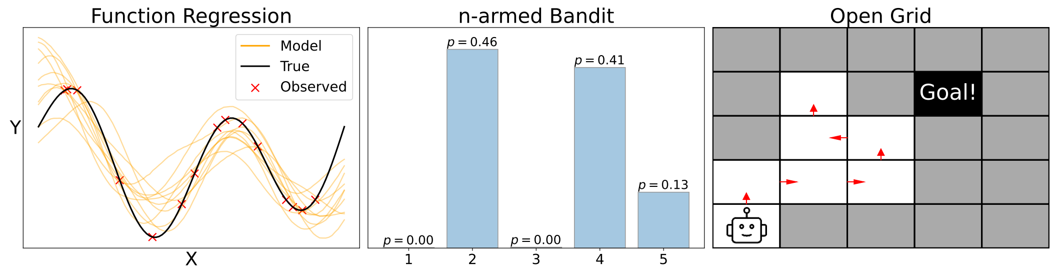

Figure 2: Visualization of sampled tasks we evaluated our method on. 1) Zero-shot learning of a function (left), 2) learning a stochastic best-arm selection algorithm (middle), and 3) learning a deterministic grid exploration agent (right).

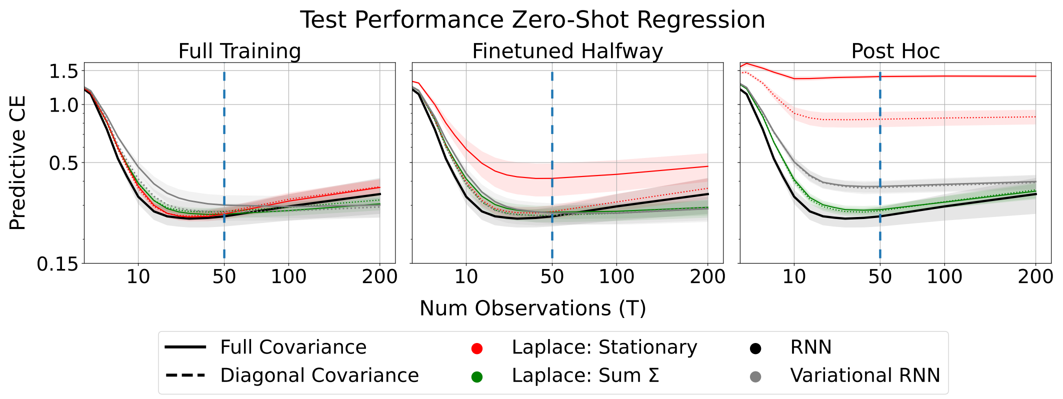

Figure 3: Final performance on the zero-shot regression task in terms of predictive cross-entropy, this should go down over time. The left column shows the results when doing complete model training, the middle and right columns perform model pre-training with the RNN (black). The middle column does finetuning with the variational architectures, the right column does not. The blue dashed line indicates the training cut-off ($T = 50$), past this point the model is generalizing.

5.1 Supervised Learning

Experimental Setup. As a didactic test-setup, we evaluated our method on noiseless 1D regression tasks. We generated data by sampling parameters to a Fourier expansion and then sampling datasets $\{\{(X_i, f_j(X_i))\}_{i=1}^{T}\}_{j=1}^{n}$ where each $X_i \sim \text{Unif}(-1, 1)$, $f_j \sim p_{\text{fourier}}(f)$, and $n = 256, T = 50$. During training, we optimized a lower bound for a supervised domain using a weight for the KL-term of $\beta = 10^{-2}$ (see Eq. (2); Appendix A.1.1). During testing, we computed the predictive cross-entropy (CE) with the true data-generating distribution and our model. So, at each step t, we used $H_t = \{X_i, f(X_i)\}_{i=1}^{t}$ to estimate the posterior predictive distribution $\mathbb{E}_{q_\theta(Z_t|H_t)} p_\theta(Y_i|X_i, Z_t)$ with Monte-Carlo using $m = 30$ samples. The predictive CE was then estimated by averaging the CE over a large i.i.d. test dataset.

Variations. Our Laplace VRNN used a stationary $q_\theta(Z_t|H_{<t})$ assumption (Laplace: Stationary; red) and a Markovian $q_\theta(Z_t|H_{t-1}, Z_{t-1})$ assumption (Laplace: Sum Σ; green) on the graphical model from Eq. (1). In practice, the stationary model computes the covariance for each datapoint in $H_{<t}$ at each t, whereas, the Markovian model sums the covariances for each pair (X_i, Y_i). To reduce clutter, we only show the Laplace VRNN ablation that sums the covariance, which also performed best among our variations (c.f., Appendix C.1). We test for both diagonal or full covariance matrices for the Gaussian distributions.

Results. As shown in Figure 3, across our comparisons the predictive CE goes down initially (except for the post-hoc stationary Laplace VRNN), however slightly increases again after the size of the dataset exceeds that seen during training $T > 50$. Notably, we see that our stationary Laplace VRNN strongly degrades performance when not used at the beginning (red), most notably the full-covariance variation. This could be an indication that the Laplace approximated posterior is too wide, whereas

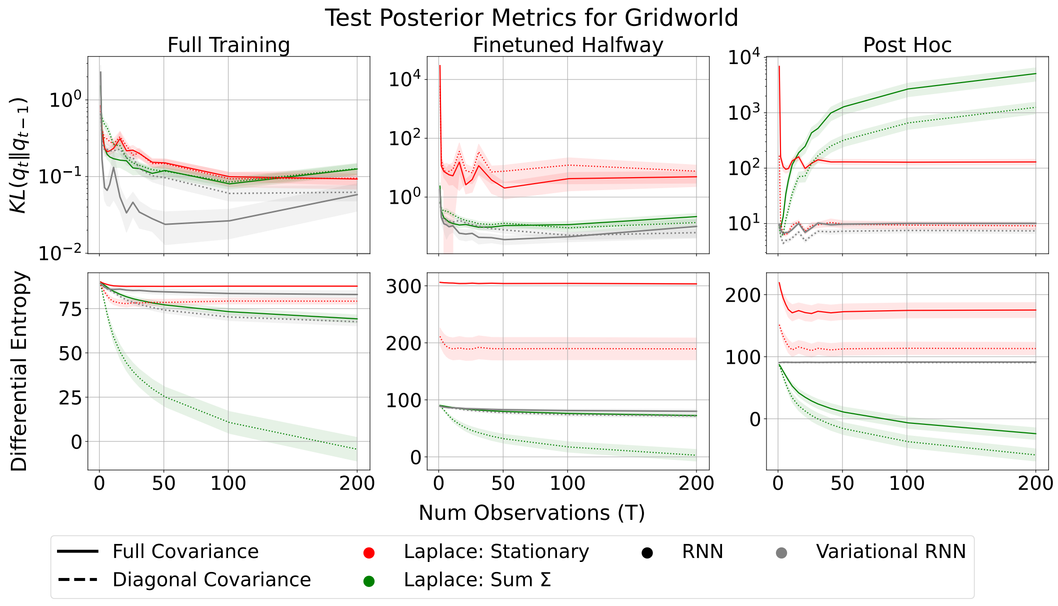

Figure 4: Evolution of summary statistics for the posterior model during testing. The top row shows the KL-divergences between consecutive posteriors q_t and q_{t+1}, and the bottom row shows the model entropy over time. In principle, we expect all lines to decrease gradually with more observations. When applying our Laplace approximation with summed covariances (green) after deterministic pre-training (right-column), we see that the posterior becomes more and more confident but does not converge to a stable distribution.

the point-estimate is extremely sharp, causing samples from our method to be out-of-distribution for the predictive model. In contrast, this result also shows that our method with the Markovian assumption (green) performs at least as well as the baselines in all cases while also providing a Bayesian posterior for the RNN after training.

5.2 Reinforcement Learning

Experimental Setup. To show that our method can perform uncertainty quantification while maintaining strong performance in reinforcement learning problems, we evaluated our method on a stochastic 5-armed bandit and a deterministic 5×5 gridworld with sparse rewards. We tested all models using a variant of recurrent PPO (Schulman et al., 2017) as a simple approximation for Eq. (3). During training, we used a batch size of $B = 256$ task samples and a sequence length of $T = 50$ interactions with the bandit and $T = 100$ for the gridworld environment.

For the bandit, we generated training tasks by sampling reward probabilities from a Dirichlet prior using $\vec{\alpha} = 0.2$. In this domain we only condition our policy on the sampled model hypotheses $z_t \sim q_\theta(Z_t|H_t)$, $a_t \sim \pi_\theta(Z = z_t)$ as is typical in Thompson sampling (Osband et al., 2013). This experiment also aimed to investigate robustness to model sampling noise. For the gridworld (Zintgraf et al., 2021) we sampled tasks by generating the agent's start- and goal-tile uniformly randomly across the grid. In contrast to the bandit problem, the gridworld agent modeled the task as a Bayes-adaptive Markov decision process (Duff, 2002). Meaning that the policy conditions on both the model samples Z and the current state, $a_t \sim \pi_\theta(Z = z_t, S = s_t)$.

Variations. Like before, we test different assumptions for our Laplace VRNN agent's model from Eq. (1). In this instance, we used a *windowed* version of the stationary Laplace VRNN $q_\theta(Z_t|H_{t-w-1:t-1})$ for $w = 10$ (red). In other words, this truncates the history up to a certain timestep to improve the runtime of the covariance computation, which otherwise scales in $\mathcal{O}(t)$-time. We also tested two variations of the Markovian $q_\theta(Z_t|H_{t-1}, Z_{t-1})$ factorization, which scaled in $\mathcal{O}(1)$-time. The proper-Markovian method (blue) sums the mean and covariance computed at each state-action (S_t, A_t) whereas the second variant only sums these covariance (green).

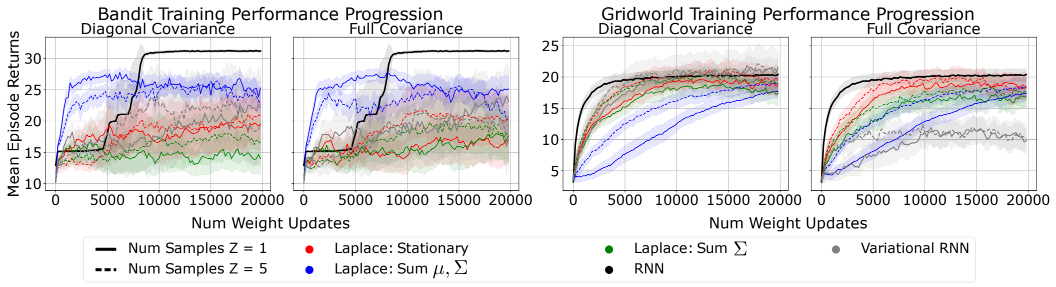

Figure 5: Average return curves during training for the Reinforcement Learning experiments. The dashed and solid lines (Num Z) indicate the number of Monte-Carlo samples used for the posterior model during training inside the loss of Eq. (3), to validate off-policy robustness in our loss.

Results. We visualize the evolution of estimated posterior statistics during testing in figure 4, where we removed the ablation that sums both the mean and covariance (blue) to reduce clutter (this ablation performed in between the other two, see Appendix C.3). We plotted the differential entropy of the posteriors q_θ and the consecutive KL-divergences $KL(q_\theta(Z_t|\dots)\|q_\theta(Z_{t-1}|\dots))$ between posteriors over time, to see whether their behaviour matches that of the true posterior. For the true posterior, we expect the entropy to decrease gradually with more observations T, which indicates that our model concentrates around some true value. Furthermore, we expect the KL-divergences to converge to zero, which implies that it converges to some true value.

As expected, we see that the posterior entropy of the Bayesian methods reliably goes down and the KL-divergences gets close to zero (left-column). We see a similar pattern when doing finetuning (middle-columns) except for our stationary Laplace variation (red). Most importantly, we see a strong effect of our accumulating covariances variation (green) when using a post-hoc posterior approximation. We see that the entropy steadily decreases while the KL-divergences between consecutive posteriors grows larger and larger. In contrast, the post-hoc VRNN (grey) and stationary Laplace (red) stay relatively constant, and are therefore non-informative. This result shows that the deterministic RNN does not converge to a stable hidden state when not explicitly regularized during training. This means that the learned estimator becomes more and more confident while not being consistent (Xiong et al., 2021).

The average training returns for our model ablations are shown in Figure 5. As argued in the introduction, we find that the deterministic method (black) has strong performance while also being the least noisy in the mean episode returns and being the fastest to train in terms of algorithm runtime. Interestingly, the proper-Markovian factorization (blue) of our Laplace VRNN showed faster learning in the Bandit up to a certain point, whereas it degraded training performance for the gridworld. All Bayesian methods tested on the bandit task were significantly noisy and only achieved sub-linear cumulative regret during test-time about 50% over all experiment repetitions. On the grid task, all methods achieved sub-linear cumulative regret during testing.

In summary, none of the ablations for our Bayesian methods degraded performance when applied after deterministic pre-training without finetuning (post-hoc). Our Laplace VRNN also typically performed on par with the VRNN in terms of training returns. However, only our Markovian Laplace variation that summed the covariances (green) could produce insightful posterior statistics of the pre-trained model. This confirms all our research questions of whether our proposed Laplace VRNN, and for what model assumptions, can we get useful posterior statistics while not degrading performance compared to the baselines.

6 Conclusions

We have described how the Laplace approximation can be applied to recurrent neural network models in a zero-shot meta-reinforcement learning context. Our method is a cheap transformation of an existing recurrent network to a Bayesian model. This enables trained agents to more accurately model their task-uncertainty which can be exploited to obtain a better or more robust method.

We tested our method on supervised and reinforcement learning tasks to investigate the utility of our approximation (the quality of posterior statistics), how it depends on model assumptions (ablations), and how it compares against variational inference or point-estimate baselines (no degradation in performance). Our results show that the proposed *Laplace variational recurrent neural network* can reliably transform existing non-Bayesian models to produce a Bayesian posterior, at any point during training without modifying the model or training procedure. In contrast, variational inference requires a completely different model architecture and training setup, even though our results show that the Laplace approximation performs on par or sometimes better.

One of the limitations of our work is the computation of the Jacobians and possible restrictiveness of the Gaussian distribution. Future work could explore improving this using, e.g., low-rank approximations (Lee et al., 2020; George et al., 2018). Future work could use our method to estimate online distribution shifts (Daxberger et al., 2021) or use exploration strategies like those in Bayesian optimization, e.g., entropy search (Hvarfner et al., 2022), which could lead to stronger algorithms.

7 Acknowledgements

This work was supported by the AI4B.io project funded by dsm-firmenich. We thank our colleagues who provided insight and expertise that greatly assisted the research.

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, 2016.

Jakob Bauer, Kate Baumli, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, Karol Gregor, Edward Hughes, Sheleem Kashem, Maria Loks-Thompson, Hannah Openshaw, Jack Parker-Holder, Shreya Pathak, Nicolas Perez-Nieves, Nemanja Rakicevic, Tim Rocktäschel, Yannick Schroecker, Satinder Singh, Jakub Sygnowski, Karl Tuyls, Sarah York, Alexander Zacherl, and Lei M Zhang. Human-timescale adaptation in an open-ended task space. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *25th International Conference on Pattern Recognition (ICPR)*, 2021.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. arXiv:2301.08028, 2023.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007. ISBN 0387310738.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Paul Bromiley. Products and convolutions of Gaussian probability density functions. *Tina-Vision Memo*, 2003.

Yutian Chen, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, 2015.

G. Cybenko. Approximation by superpositions of a sigmoidal function. In *Mathematics of Control, Signals and Systems*, 1989.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux - effortless Bayesian deep learning. In *Advances in Neural Information Processing Systems*, 2021.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. arXiv:1611.02779, 2016.

Michael O'Gordon Duff. *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, The University of Massachusetts Amherst, 2002.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, 2018.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. arXiv:1807.01622, 2018.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, 2018.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 2015.

Jonathan Gordon, John F. Bronskill, M. Bauer, Sebastian Nowozin, and Richard E. Turner. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2018.

Anirudh Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Rosemary Ke, and Yoshua Bengio. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, 2017.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical Bayes. In *International Conference on Learning Representations*, 2018.

Ido Greenberg, Shie Mannor, Gal Chechik, and Eli Meirom. Train hard, fight easy: Robust meta reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv:2312.00752, 2023.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.

Timothy M. Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Carl Hvarfner, Frank Hutter, and Luigi Nardi. Joint entropy search for maximally-informed Bayesian optimization. In *Advances in Neural Information Processing Systems*, 2022.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of Bayesian neural nets via local linearization. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, 2021.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 2019.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 2022.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

Jongseok Lee, Matthias Humt, Jianxiang Feng, and Rudolph Triebel. Estimating model uncertainty of neural networks in sparse information form. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv:1805.00909, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. In *Advances in Neural Information Processing Systems*, 2022.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Vladimir Mikulik, Grégoire Delétang, Tom McGrath, Tim Genewein, Miljan Martic, Shane Legg, and Pedro Ortega. Meta-trained agents implement Bayes-optimal agents. In *Advances in Neural Information Processing Systems*, 2020.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, 2013.

Carl Edward Rasmussen. Gaussian processes in machine learning. Springer, 2004. ISBN 978-3-540-28650-9.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured Laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems*, 2018.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv:1707.06347, 2017.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. The curse of recursion: Training on generated data makes models forget. arXiv:2306.17493, 2023.

Gautam Singh, Jaesik Yoon, Youngsung Son, and Sungjin Ahn. Sequential neural processes. In *Advances in Neural Information Processing Systems*, 2019.

Shagun Sodhani, Franziska Meier, Joelle Pineau, and Amy Zhang. Block contextual MDPs for continual learning. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2 edition, 2018. ISBN 978-0-262-03924-6.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Che Wang, Yanqiu Wu, Quan Vuong, and Keith Ross. Striving for simplicity and performance in off-policy DRL: Output normalization and non-uniform sampling. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. arXiv:1611.05763, 2017.

Mike Wu, Kristy Choi, Noah Goodman, and Stefano Ermon. Meta-amortized variational inference and learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Zheng Xiong, Luisa M Zintgraf, Jacob Austin Beck, Risto Vuorio, and Shimon Whiteson. On the practical consistency of meta-reinforcement learning algorithms. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: Variational Bayes-adaptive deep RL via meta-learning. *Journal of Machine Learning Research*, 2021.

Bernt Øksendal. *Stochastic Differential Equations*. Springer, 2003. ISBN 978-3-642-14394-6.

A Derivations

A.1 Lower Bounds

In this section, we derive the two lower bounds used for model training in the main paper. These lower bounds are not particularly new or special, they only show how one can derive a learning objective from a probabilistic graphical modeling perspective.

A.1.1 Supervised Learning

For the supervised learning domain we can derive an evidence lower bound on the data-marginal as a training objective for our neural network parameters in the following way. For all permutations of $H^{1:n} = \{X^i, Y^i\}_{i=1}^n$, where $X^i \in \mathcal{X}, Y^i \in \mathcal{Y}$, we have,

$$\begin{aligned} p(H^{1:n}) &= \int p(H^{1:n}|z)p(z)dz \\ &= \int p(X^n, Y^n|z, H^{<n})p(H^{<n}|z)p(z)dz \\ &= \int p(X^n, Y^n|z)p(H^{<n}|z)p(z)dz \\ &= p(H^{<n}) \int p(X^n, Y^n|z)p(z|H^{<n})dz \end{aligned}$$

if we complete the recursion for $p(H^{<n})$ and do importance sampling on the posterior with q, we get,

$$\begin{aligned} \ln p(H^{1:n}) &= \ln \prod_{i=1}^{n} \int p(X^i, Y^i|z^i)p(z^i|H^{1:i})dz^{1:n} \\ &= \sum_{i=1}^{n} \ln \int p(X^i, Y^i|z^i)\frac{q(z^i|H^{1:i})}{q(z^i|H^{1:i})}p(z^i|H^{1:i})dz^{1:n} \\ &\geq \sum_{i=1}^{n} \int q(z^i|H^{1:i}) \left[\ln p(X^i, Y^i|z^i) + \ln \frac{p(z^i|H^{1:i})}{q(z^i|H^{1:i})}\right] dz^{1:n} \\ &= \sum_{i=1}^{n} \mathbb{E}_{q(Z^i|H^{1:i})} \ln p(X^i, Y^i|Z^i) - KL(q(Z^i|H^{1:i}) \| p(Z^i|H^{1:i})), \end{aligned}$$

which gives us the objective for our approximate inference model when we use neural network parameters θ for the predictive and posterior models,

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \mathbb{E}_{q_\theta(Z^i|H^{1:i})} \underbrace{\ln p_\theta(X^i, Y^i|Z^i)}_{\text{Prediction Loss}} - \beta \cdot \underbrace{KL(q_\theta(Z^i|H^{1:i}) \| \texttt{stop\_grad}[q_\theta(Z^{i-1}|H^{1:i-1}))}_{\text{Complexity Penalty}},$$

where $\texttt{stop\_grad}[\cdot]$ indicates a stop-gradient operation and $\mathcal{L}$ should be *maximized* with respect to θ. The hyperparameter $\beta \in \mathbb{R}_+$ accounts for differences in scaling. The stop-gradient is necessary so that the posterior at time t does not depend on the future. During generation and training, we also assume a uniform prior over the inputs $p(X^i|Z) = \text{Unif}$. Of course, this is just one lower bound, the one we use in the main paper for the reinforcement learning tasks also assumes that each z^i is sequentially dependent. In this case, the product would appear inside the integral in the first line for $\ln p(H^{1:n})$, this is only relevant for the Laplace VRNN that accumulates the mean and covariances.

For simplicity, we only perform training on a single permutation of H^n (i.e., canonical order), as in expectation all permutations are covered anyway and this provides training batches with more diverse examples. Unfortunately, when amortizing the computation of this lower bound with recurrent models it can be difficult to properly distill this permutation invariance of the data into the model. Using a recurrent model that linearly transforms the state, like a transformer (Vaswani et al., 2017; Katharopoulos et al., 2020) or general state space model (Bishop, 2007), would prevent this. We leave this open for future work.

A.1.2 Reinforcement Learning

Consider the joint distribution over environment traces $H^i = \{S_t, R_t, A_t\}_{t=1}^T$ and latent variables Z, we'll write episode indices (*extra*-episodic) $i = 1 \dots, n$, in the superscript and time indices (*inter*-episodic) $t = 1, \dots, T$, in the subscript,

$$\begin{aligned}
p(H^{1:n}, Z^{1:n}) &= \prod_{i=1}^{n} p(H^i, Z^i | H^{<i}, Z^{<i}) \\
&= \prod_{i=1}^{n}\prod_{t=1}^{T_i} p(S_t^i, R_t^i, A_t^i, Z_t^i | S_{<t}^i, R_{<t}^i, A_{<t}^i, Z_{<t}^i, Z^{<i}, H^{<i}) \\
&= \prod_{i=1}^{n}\prod_{t=1}^{T_i} p(S_t^i, R_t^i, A_t^i | Z_t^i, H_{<t}^i) p(Z_t^i | \underbrace{Z_{<t}^i, H_{<t}^i}_{inter}, \underbrace{Z^{<i}, H^{<i}}_{extra}) \\
&= \prod_{i=1}^{n}\prod_{t=1}^{T_i} \underbrace{p(R_t^i | S_t^i, A_t^i, Z_t^i)}_{\text{Reward Model}} \underbrace{\pi(A_t^i | S_t^i, Z_t^i)}_{\text{Action Model}} \cdot \\
&\qquad \underbrace{p(S_t^i | S_{t-1}^i, A_{t-1}^i, Z_t^i)}_{\text{Transition Model}} \underbrace{p(Z_t^i | Z_{<t}^i, H_{<t}^i, Z^{<i}, H^{<i})}_{\text{Posterior Model}},
\end{aligned}$$

the lower-bound in Eq. 2 can then be easily derived by doing importance sampling on the posterior model with q, marginalizing out the latent variables, and assuming that H^i is independent of all other variables given the latent-variable Z^i,

$$\begin{aligned}
\ln p(H^{1:n}) &= \ln \int \prod_{i=1}^{n} p(H^i, z^i | H^{<i}, z^{<i}) dz^{1:n} \\
&= \ln \int \prod_{i=1}^{n} p(H^i | z^i) \frac{q(z^{1:i} | H^{<i})}{q(z^{1:i} | H^{<i})} p(z^i | H^{<i}, z^{<i}) dz^{1:n} \\
&\geq \int \sum_{i=1}^{n} q(z^{1:i} | H^{1:i}) \left(\ln p(H^i | z^i) - \ln \frac{q(z^{1:i} | H^{<i})}{p(z^i | H^{<i}, z^{<i})} \right) dz^{1:n} \\
&\propto \mathbb{E}_{q(Z^{1:n} | H^{1:n})} \sum_{i=1}^{n} \ln p(H^i | Z^i) - KL(q(Z^i | Z^{<i}, H^{<i}) \| p(Z^i | Z^{<i}, H^{<i})).
\end{aligned}$$

As stated in the paper, this lower bound only reproduces the data but does not maximize the rewards per se. So, using the control as inference framework (Levine, 2018), if we write the conditional that a given trajectory H is optimal as $p(\mathcal{O} = 1 | H) \propto \exp(\sum_{t=1}^{T} R_t)$, then we can derive a lower bound for the sampling distribution for a reinforcement learning agent as,

$$\begin{aligned}
\ln p(\mathcal{O} = 1) &= \ln \mathbb{E}_{q(H^{1:n}, Z^{1:n})} p(\mathcal{O} = 1 | H^{1:n}, Z^{1:n}) \frac{p(H^{1:n}, Z^{1:n})}{q(H^{1:n}, Z^{1:n})} \\
&\geq \mathbb{E}_{q(H^{1:n}, Z^{1:n})} \ln p(\mathcal{O} = 1 | H^{1:n}) - KL(q(H^{1:n}, Z^{1:n}) \| p(H^{1:n}, Z^{1:n})), \\
&= \mathcal{L}(q)
\end{aligned}$$

where we define the variational distribution $q(H^{1:n}, Z^{1:n})$ to factorize in exactly the same way as $p(H^{1:n}, Z^{1:n})$ where we fix the reward and transition models and then modify the action and posterior models. This choice of factorization cancels out the fixed terms in the KL-divergence, giving us the lower bound,

$$\begin{aligned}
\mathcal{L}(q) = \mathbb{E}_{q(H^{1:n}, Z^{1:n})} \sum_{i=1}^{n} \sum_{t=1}^{T_i} R_t^i &- KL\left(q(A_t^i | S_t^i, Z_t^i) \| \pi(A_t^i | S_t^i, Z_t^i)\right) \\
&- KL\left(q(Z_t^i | Z_{<t}^i, H_{<t}^i, Z^{<i}, H^{<i}) \| p(Z_t^i | Z_{<t}^i, H_{<t}^i, Z^{<i}, H^{<i})\right).
\end{aligned}$$

To amortize computation of this lower bound and make this practical to compute, we parametrize the variational posterior $q_\theta(Z | \dots)$ and action model $\pi_\theta(A | \dots)$ on the joint set θ. To then finally give us a practical optimization objective for the parameters θ, we substitute for the action model $\pi(A|\dots) = \pi_{\theta_{old}}$ and for the true posterior we simply use $q_\theta(Z|\dots)$ with a stop-gradient. We scale the KL-penalty with a hyperparameter $\beta \in \mathbb{R}_+$ to account for differences in scaling. This gives us our final objective,

$$\mathcal{L}(\theta) = \mathbb{E}_{q_\theta(H^{1:n}, Z^{1:n})} \sum_{i=1}^{n} \sum_{t=1}^{T_i} R_t^i - KL\left(\pi_\theta(A_t^i|S_t^i, Z_t^i) \| \pi_{\theta_{old}}(A_t^i|S_t^i, Z_t^i)\right)$$
$$- \beta \cdot KL\left(q_\theta(Z_t^i|Z_{<t}^i, H_{<t}^i, Z^{<i}, H^{<i}) \| \Box q_\theta(Z_{t-1}^i|Z_{<t-1}^i, H_{<t-1}^i, Z^{<i}, H^{<i})\right),$$

which we can optimize through sampling and end-to-end differentiation from the action model to the posterior. Although this is the objective we desire, we make further heuristic approximations through the use of the Proximal Policy Optimization algorithm (Schulman et al., 2017). This could roughly be interpreted as doing expectation-propagation (Bishop, 2007) on the action model. Further details on this approximation of the true graphical model are outside the scope of this paper.

Our lower bound is an extension of the one by Abdolmaleki et al. (2018), for standard Markov decision processes, to include the latent variable posterior for use in memory-based meta-reinforcement learning (Duan et al., 2016). When using an RNN to approximate the posterior, the KL-penalty for $q_\theta(Z|\dots)$ is typically ignored since this is undefined for point-estimates. Doing this would recover the RL^2 objective in combination with MPO (Abdolmaleki et al., 2018; Duan et al., 2016).

A.2 Posterior Factorization

To choose an efficient factorization for our variational model we need the following result,

Lemma 1. *We can write* $p(Z|\{X_i\}_{i=1}^n) = \frac{1}{p(Z)^{n-1}} \prod_{i=1}^n p(Z|X_i)$ *iff* $X_i \perp\!\!\!\perp X_j, \forall j \neq i$.

Proof. This result can be shown by applying Bayes rule then factorizing each X_i to be independent of $X_j, \forall j \neq i$ and then applying Bayes rule again,

$$\begin{aligned} p(Z|\{X_i\}_{i=1}^n) &= \frac{p(X_1, X_2, \dots, X_n|Z)p(Z)}{p(X_1, X_2, \dots, X_n)} \\ &= \frac{p(Z)}{\prod_{i=1}^n p(X_i)} \prod_{i=1}^n p(X_i|Z) && \text{(Independence)} \\ &= \frac{p(Z)}{\prod_{i=1}^n p(X_i)} \left[\prod_{i=1}^n p(Z|X_i)\frac{p(X_i)}{p(Z)}\right] \\ &= \frac{p(Z)}{\prod_{i=1}^n p(X_i)} \left[\frac{\prod_{i=1}^n p(X_i)}{p(Z)^n} \prod_{i=1}^n p(Z|X_i)\right] \\ &= \frac{1}{p(Z)^{n-1}} \prod_{i=1}^n p(Z|X_i) \end{aligned}$$

$\square$

A.3 Laplace Variational Recurrent Model

Proposition 1. *Given a mean-field assumption on the data for our posterior q_θ (Lemma 1). The second order Taylor Expansion of $\ln q_\theta(Z_t|H_{<t}, \phi_t)$ linearized at ϕ_t, where $\phi_t = \phi^*$ is a local maximizer of q_θ and occupies the same space as Z_t, yields the following Gaussian distribution,*

$$q_\theta(Z_t|H_{<t}, \phi_t) = \mathcal{N}\left(Z_t; \mu = \phi_t, \Sigma = (-\nabla_\phi^2 \ln q_\theta(Z_t|H_{<t}, \phi)|_{\phi=\phi_t})^{-1}\right).$$

Proof. Reiterating the results from the main paper, we choose to factorize our model as,

$$\begin{aligned}
q_\theta(Z_t|H_{<t},\phi_t) &= \frac{1}{q_\theta(Z_t|\phi_t)^{t-2}} \prod_{i=1}^{t-1} q_\theta(Z_t|S_i,R_i,A_i,\phi_t), && \text{(Lemma 1)}\\
&= \exp\left[(2-t)\ln q_\theta(Z_t|\phi_t) + \sum_{i=1}^{t-1} \ln q_\theta(Z_t|S_i,R_i,A_i,\phi_t)\right]\\
&= \exp h(Z_t;H_{<t},\phi_t),
\end{aligned}$$

where our aim is to make a local approximation to h, the rest of the proof follows Appendix A from Daxberger et al. (2021).

The second order Taylor expansion of $h(Z_t;H_{<t},\phi_t)$ where ϕ_t (locally) maximizes h keeping all other arguments fixed, gives us,

$$\begin{aligned}
\hat{h}(Z_t;H_{<t},\phi) &= h(Z_t;H_{<t},\phi_t) + \underbrace{\nabla_\phi h|_{\phi=\phi_t}(\phi-\phi_t)}_{=0} + \frac{1}{2}(\phi-\phi_t)^\top \nabla^2_\phi h|_{\phi=\phi_t}(\phi-\phi_t),\\
&= h(Z_t;H_{<t},\phi_t) - \frac{1}{2}(\phi-\phi_t)^\top \nabla^2_\phi h|_{\phi=\phi_t}(\phi-\phi_t),
\end{aligned}$$

dropping the function arguments to h for the higher-order terms for brevity. When exponentiating $\hat{h}$ and renormalizing it to integrate to 1, it is easy to show that we recover a Gaussian,

$$\begin{aligned}
h(Z_t;H_{<t},\phi_t) &\approx \frac{1}{\int_{\mathbb{R}} \exp \hat{h}(Z_t;H_{<t},\phi')d\phi'} \exp \hat{h}(Z_t;H_{<t},\phi)\\
&= \frac{\exp\{h(Z_t;H_{<t},\phi_t) - \frac{1}{2}(\phi-\phi_t)^\top(-\nabla^2_\phi h|_{\phi=\phi_t})(\phi-\phi_t)\}}{\int_{\mathbb{R}} \exp\{h(Z_t;H_{<t},\phi_t) - \frac{1}{2}(\phi'-\phi_t)^\top(-\nabla^2_\phi h|_{\phi=\phi_t})(\phi'-\phi_t)\}d\phi'}\\
&= \frac{\exp\{-\frac{1}{2}(\phi-\phi_t)^\top(-\nabla^2_\phi h|_{\phi=\phi_t})(\phi-\phi_t)\}}{\int_{\mathbb{R}} \exp\{-\frac{1}{2}(\phi'-\phi_t)^\top(-\nabla^2_\phi h|_{\phi=\phi_t})(\phi'-\phi_t)\}d\phi'}\\
&= \mathcal{N}\left(\phi;\mu=\phi_t,\Sigma=(-\nabla^2_\phi \ln q_\theta(Z_t|H_{<t},\phi)|_{\phi=\phi_t})^{-1}\right).
\end{aligned}$$

Perhaps surprisingly, this gives us a distribution over ϕ and not Z. However, we only introduced ϕ as a helper variable for Z and assumed that $\phi \equiv \mu_Z \equiv Z_{mode}$, i.e., the local maximizer for the log-posterior. Therefore, we can simply use this distribution over ϕ as if it were Z. $\square$

Proposition 2. *If we choose $q_\theta(Z_t|S_i,R_i,A_i,\phi) = \mathcal{N}(Z_t;\mu=f_\theta(S_i,R_i,A_i;\phi),\Sigma=I_n)$ and $q_\theta(Z_t|\phi) = \mathcal{N}(Z_t;\mu=\phi,\Sigma=\sigma^2_\phi I_n)$ where we take the limit for σ^2_ϕ to infinity, then our Laplace approximated posterior (Proposition 1) has an inverse covariance that is computed as,*

$$\Sigma_t^{-1} = \sum_{i=1}^{t-1}(\nabla_\phi f_\theta(S_i,R_i,A_i;\phi)|_{\phi=\phi_t})(\nabla_\phi f_\theta(S_i,R_i,A_i;\phi)|_{\phi=\phi_t})^\top.$$

Proof. To see this we only need to write down the Hessian under a local maximum assumption of $\phi = \phi^*$ (Laplace approximation) and substitute the chosen Gaussian distributions in for all terms.

$$\begin{aligned}
\nabla^2_\phi \ln q_\theta(Z_t|H_{<t}, \phi) &= \nabla^2_\phi \left[(2-t) \ln q_\theta(Z_t|\phi) + \sum_{i=1}^{t-1} \ln q_\theta(Z_t|S_i, R_i, A_i, \phi) \right] \\
&= \nabla^2_\phi \left[(2-t) \ln \mathcal{N}(Z_t; \phi, \sigma^2_\phi I_n) + \sum_{i=1}^{t-1} \ln \mathcal{N}(Z_t; f_\theta(S_i, R_i, A_i; \phi), I_n) \right] \\
&= \frac{t-2}{\sigma^2_\phi} I_n + \sum_{i=1}^{t-1} (J_\phi)_i \underbrace{\left(\nabla^2_\mu \ln \mathcal{N}(Z_t; (f_\theta)_i, I_n)\right)}_{=-1} (J_\phi)_i^\top \\
&\qquad \underbrace{+ \sum_{i=1}^{t-1} \nabla^2_\phi (f_\theta)_i|_{\phi=\phi_t} \nabla_\mu \ln \mathcal{N}(Z_t; (f_\theta)_i, I_n)}_{=0,\text{ when } \phi_t=\phi^* \text{ (Laplace approx.)}} \\
&= \frac{t-2}{\sigma^2_\phi} I_n - \sum_{i=1}^{t-1} (J_\phi)_i (J_\phi)_i^\top,
\end{aligned}$$

where we abbreviate $(J_\phi)_i = \nabla_\phi f_\theta(S_i, R_i, A_i; \phi)$ and $(f_\theta)_i = f_\theta(S_i, R_i, A_i; \phi)$. Then, in the case of using an infinite variance Gaussian for the prior, $\lim_{\sigma^2_\phi \to \infty} \nabla^2_\phi \ln q_\theta(Z_t|H_{<t}, \phi)$, we get,

$$\begin{aligned}
\Sigma_t^{-1} = \left(-\nabla^2_\phi \ln q_\theta(Z_t|H_{<t}, \phi)|_{\phi=\phi_t}\right) &= \sum_{i=1}^{t-1} (J_\phi)_i (J_\phi)_i^\top \\
&= \sum_{i=1}^{t-1} (\nabla_\phi f_\theta(S_i, R_i, A_i; \phi)|_{\phi=\phi_t})(\nabla_\phi f_\theta(S_i, R_i, A_i; \phi)|_{\phi=\phi_t})^\top.
\end{aligned}$$

□

We can also add a slight diagonal jitter or constant to the inverse covariance Σ_t^{-1}, this can be interpreted as imposing an isotropic Gaussian prior for ϕ. This has a slight ambiguity due to the other prior term $\frac{t-2}{\sigma^2_\phi} I_n$, which also induces a shift on the diagonal values. Hence, our reason for not directly including this in the derivation. In our implementation, adding a small diagonal prior was only really important for improved numerical stability during matrix inversion.

A.3.1 Final Model

To complete our fully general Laplace approximated variational recurrent neural network, we need to define the posterior over *all* previous latent variables, $q_\theta(Z_t|Z_{<t}, H_{<t})$. We can easily plug this dependency in for our Laplace approximation from Prop. 1 by assuming that each consecutive posterior has an additive effect on all future posteriors (i.e., a Gaussian convolution),

$$q_\theta(Z_t|Z_{<t}, H_{<t}) = \mathcal{N}\left(Z_t; \mu_t = \phi_t + \sum_{i=1}^{t-1} Z_i, \Lambda_t = -\nabla^2_\phi \ln q_\theta(Z_t|Z_{<t}, H_{<t}, \phi_t)|_{\phi=\phi_t} \right).$$

As long as we do not condition ϕ_t on $Z_{<t}$, the dependency on past latent variables becomes a constant w.r.t. ∇^2_ϕ, making the covariance independent of these terms. This is in contrast to a recurrent state-space model architecture which does condition on these values (Hafner et al., 2020), however, our choice permits an analytical solution. It is a known result that the expected posterior then becomes another Gaussian with the means and inverse covariances summed (Bromiley, 2003),

$$\mathbb{E}_{q_\theta(Z_{<t}|H_{<t})} q_\theta(Z_t|Z_{<t}, H_{<t}) = \mathcal{N}\left(\mu_t = \phi_t + \sum_{i=1}^{t-1} \mu_i, \Lambda_t = -\nabla^2_\phi \ln q_\theta(Z|H_{<t}, \phi)|_{\phi=\phi_t} + \sum_{i=1}^{t-1} \Lambda_i \right).$$

This particular form has also been described by Ritter et al. (2018) in the context of continual learning. It is easy to accumulate the mean and covariances terms sequentially over t. Depending on the assumptions one makes on the data-generating distribution, one can sum over fewer terms to make the calculation more efficient. The ones we ran experiments for in the main paper include:

1. **Stationary Posterior**: $q_\theta(Z_t|H_{<t})$. Full summation over $H_{<t}$ in the calculation of Λ_t. No summation over previous posteriors $Z_{<t}$.
2. **Markov Chain Posterior**: $q_\theta(Z_t|H_{t-1}, Z_{t-1})$. The inverse covariance is calculated only using the most recent observation $t-1$. Only the previous posterior mean and precision are summed with the current mean and precision.
3. **Windowed Markov Chain Posterior**: $q_\theta(Z_t|H_{k:t-1}, Z_{t-1})$. The inverse covariance is calculated using the k most recent observations. Only the previous posterior mean and precision are summed with the current mean and precision.

However, these are all simplified models, whereas the variational recurrent model (Chung et al., 2015) we discuss in the main paper is fully general.

B Implementation Details

B.1 Model Architecture and Optimization

Following the main text we can define our model according to the following components,

$$\begin{array}{lll}
\text{Embedding} & S_t^g, A_t^g, R_t^g & = g_\theta(S_t), h_\theta(A_t), w_\theta(R_t), \\
\text{Recurrent Model} & \phi_{t+1} & = f_\theta(S_t^g, A_t^g, R_t^g; \phi_t), \\
\text{Posterior Model} & Z_t & \sim q_\theta(Z_t|H_{<t}, \phi_t), \\
\text{Reward Model} & \hat{R} & \sim p_\theta(\hat{R}|Z_t, A^g, S_t^g), \\
\text{Action Model} & A_t & \sim \pi_\theta(A_t|Z_t, S_t^g),
\end{array}$$

note that we do not train an action model for the supervised experiments, and we do not use the reward model in the reinforcement learning experiments (i.e., we do not use it to select actions or to do planning). In the supervised case, the reward model simply learns a direct function prediction $\hat{R}$ where the state can be considered stationary. For brevity, we denoted the full set of parameters as θ, in practice each component has its parameters but we jointly optimize for these using end-to-end differentiation.

For the embedding model we used a multi-layered perceptron (MLP) (Cybenko, 1989) of two hidden layers of width 256 nodes, we used leaky-ReLU for the activation. The action and predictive model used a three hidden layer MLP with sizes (256, 256, 64), also with leaky-ReLU. For the recurrent model, we use a long-short-term memory module (Hochreiter & Schmidhuber, 1997) with $n = 128$ hidden nodes. We projected the outputs (not the carried state) of the LSTM to a smaller $n = 64$ feature output vector with a learned affine transform (i.e., a 1-layer MLP without an activation).

For discrete environments, the action model predicted the n logits for the full action space. For the regression task, the reward model outputs a Gaussian with a learned mean and input-independent variance.

As noted in the main paper, we apply stop-gradients on the prior term appearing in the KL-divergences (Eq. 2 and Eq. 3). Doing this prevents past posteriors from fitting to data beyond their respective timestep, while still constraining our current posterior to not deviate too much from these terms. We also apply stop-gradients to the *past* mean and covariance terms when *accumulating* these terms. This is only relevant for the Laplace VRNN ablations. The reasoning for this is the same as for the stop-gradient in the KL term, we want to use the past means and covariances as constants at timestep t, and not as another learnable parameter. As a side note, we also found that doing this sped up training orders of magnitude.

We developed everything discussed in this paper in Jax v.0.4.23 (Bradbury et al., 2018). For neural network design, we used the Flax library v0.8.1 (DeepMind et al., 2020). For optimization, we used the Adamw optimizer (Loshchilov & Hutter, 2019) implemented in Optax with learning-rate $= 10^{-3}$, weight-decay $= 10^{-6}$, and the rest on default settings at version v0.1.7. We used gradient-clipping to have a max global norm of 1.0 and a maximum individual gradient of [-5.0, 5.0]. We used our implementation for the PPO algorithm with help from the RLax library to compute the generalized advantage estimators. For a full list of dependencies, requirements, and program flags, our code will be made available upon publication.

B.2 Variational Posterior Baseline

As discussed in the main paper, typically in meta-reinforcement learning we see that the posterior $q_\theta(Z_t|H_{<t})$ is modeled with a point-estimate and we propose to use the Laplace approximation to convert this point-estimate into a Gaussian distribution. Instead of computing the covariance matrix using the Laplace approximation, we can also directly predict this covariance with another neural network.

So, our variational recurrent neural network (VRNN) baseline predicted the $m(m-1)$ lower triangular elements of the $m \times m$ dimensional covariance matrix (or just m for the diagonal ablations) and the m dimensional mean. We opted for a spectral decomposition $\Sigma = USV$ where S is diagonal and U was predicted through the Cayley map starting from a skew-symmetric matrix. First, we compute ϕ_t with our RNN, then we project ϕ_t to μ_t, S_t, L_t with a linear layer such that $\mu_t \in \mathbb{R}^m$, $S_t \in \mathbb{R}^{m \times m}$ is diagonal and $L_t \in \mathbb{R}^{m \times m}$ is lower-triangular, we then compute $U_t = (I - A_t)(I + A_t)^{-1}$ where $A_t = L_t - L_t^\top$ to get an orthogonal eigenbasis. We then construct the Gaussian as,

$$q_\theta(Z_t|H_{<t}, \phi_t) = \mathcal{N}(Z_t; \mu = \mu_t, \Sigma = U_t(\exp S_t)U_t^\top),$$

this representation also made matrix inversion incredibly easy as $(Ue^S V)^{-1} = Ue^{-S}V$ since $U = V^\top$ are orthogonal.

The reason for using the spectral decomposition was that we did not achieve stable training using any variant of the Cholesky factorization for the covariance matrix (LU or LDU decomposition), even if explicitly transforming the eigenvalues to be positive. We suspect that the spectral parameterization trained more stably than the LU or LDL parameterization as the basis matrices U or V are constrained to the group of orthogonal matrices. Therefore, each element in the predicted parameters L_n is also constrained. In contrast, the LDL parameterization leaves the triangular parameters in an unconstrained representation which might enable an unstable representation. However, we did not investigate this problem beyond what was needed to get our method working.

We also did not accumulate the mean or covariances for the VRNN, unlike for the Laplace VRNN, as this model parameterization could simply learn this function instead (or learn to undo this parameterization). The point of our experiments was not to squeeze performance out of our baseline but to have a strong reference for comparison. We also found that the VRNN was highly competitive with the Laplace VRNN.

B.3 Predictive Ensemble Averaging

Since we sample latent variables Z from our posterior q_θ, when we pass multiple samples through our predictive model or the action model, we obtain multiple distributions for the output modalities. Typically this induces a mixture distribution, however, we simply averaged out either the logits or the means and variances (Wang et al., 2020). This can be interpreted as a normalized Gaussian convolution over the output parameters or simply as a kind of bagging strategy over ensemble members $Z = z^{(i)}, i = 1, \dots, k$.

The reasoning for opting for ensemble averaging instead of mixture distributions is that this simply achieved more stable training in combination with PPO (Schulman et al., 2017). The main problem was caused by the penalty term of the policy entropy, our implementation did not achieve stable training when approximating this term in any way (so neither with mixtures nor with singular distributions), we only achieved stable training when the entropy term could be computed *exactly*.

We did not investigate in depth why an approximate entropy loss caused training divergence, but we speculate this is due to the problems of training on generated data as discussed in the context of language models by Shumailov et al. (2023). In this case, the tails of the action distribution slowly shrink over time as Monte-Carlo estimation of the entropy might not cover low-likelihood events sufficiently with small sample sizes. This phenomenon is referred to as *model collapse*, not to be confused with *posterior collapse* (Goyal et al., 2017). We suspect that this problem could be reduced by using a proper (approximate) Bayesian inference algorithm, like maximum a posteriori optimization (Abdolmaleki et al., 2018), which essentially removes the instability of reinforcement learning losses by casting policy learning as a supervised learning problem.

B.4 Environment Design

For our testing environments we implemented three problem domains, a 1D function regression problem (Finn et al., 2017), a discrete n-armed bandit problem (Duan et al., 2016), and an $n \times n$ open grid problem (Zintgraf et al., 2021), as shown in Figure 2 in the main paper. We generated many variations of these environments to learn from by sampling their dependent task parameters.

Supervised. For the supervised problem we generated noiseless 1D test-functions on the bounded domain $[-1, 1]$. We did this through a Fourier expansion of $n = 4$ components where we randomly generate amplitudes, phase shifts, and input shifts. We manually tuned the ranges for these parameters and sampled uniformly random within these ranges. In other words, we can define the joint distribution over a function dataset as,

$$p(X, Y) = \delta(Y = \text{FourierSum}(Y; X, \varphi_{1:n}, c_{\text{shift}}, A_{0:n})) \cdot \text{Unif}(X; -1.0, 1.0),$$
$$c_{\text{shift}} \sim \text{Unif}(0.0, \pi), \varphi_i \sim \text{Unif}(0.0, \pi), A_i \sim \text{Unif}(-1.0, 1.0)$$

where the 'FourierSum' is computed in its amplitude-phase form. The training was performed with 50 examples per sampled function.

Bandit To generate bandit problems we used a Dirichlet distribution with $\alpha = 0.2$ during training and $\alpha = 0.3$ during testing (higher α makes the problem more difficult), this gave us a normalized vector of probabilities, $p_n \sim \text{Dir}(\alpha = 1_n \cdot 0.2)$ for which the agent needed to find $\max_i p_i$. Observations (which are equivalent to the rewards) were generated by sampling Bernoulli outcomes given p_i. Since each interaction of the agent with the bandit is seen as an episode, the environment returned discount factors of $\gamma = 0$. The training was performed over 50 total interactions.

Gridworld For the discrete grid environment we closely match the implementation of (Zintgraf et al., 2021). We reimplemented this environment in Jax to benefit from GPU acceleration for the data-generation process. The environment constructs a $n \times n$ open grid for the agent and uniformly randomly initializes the start and goal state (such that they don't overlap). The agent can choose between moving up, down, left, or right, the environment transitions deterministically but does not move the agent if it moves outside the bounds. The agent is rewarded with +1 if it encounters the goal and 0 otherwise, the observations were constructed as two one-hot-encoded vectors for the row and column index. The goal tile is *not* observed. If the agent did not find the goal within $T = 15$ steps it would be reset to its starting state and the discount factor would be set to $\gamma = 0$ at that transition. The training was performed over 100 total interactions.

B.5 Experimental Design and Hyperparameters

The supervised experiments did not provide additional hyperparameters to set, all necessary parameters were learned using an empirical cross-entropy with the data (see the main paper). For the reinforcement learning experiments we needed to set several hyperparameters for PPO (Schulman et al., 2017), these are given in Table 1. We did not use mini batching for our version of recurrent PPO and accumulated the loss over the full trajectories and batches. We found that larger batch sizes gave us faster and more stable learning.

Then our experimental design implied running an exhaustive parameter grid over the domain presented in Table 2. This grid was adjusted over successive experiments to reduce the computational footprint, this was manually tuned to select the parameter values that performed the best for both the baseline and our method. The parameter grid was applied equivalently on all problem domains for $r = 30$ distinct seeds. Although this experiment design is still quite modest, running this full grid induces 144 distinct configurations times 30 repetitions for the Laplace VRNN alone. One single run took on average $1\frac{1}{2}$ hour to complete for the gridworld environment on an A100 80GB NVIDIA GPU. Although, a better learning algorithm other than PPO (e.g., MPO), and minibatch optimizations could probably get the wallclock time down drastically while still achieving similar performance.

For the finetuning experiments we essentially made model snapshots of the deterministic baselines (RNNs) and reran the ablations as shown in Table 2 using the snapshots as starting weights. For each

Table 1: Proximal Policy Optimization loss parameters. Note that we use our Recurrent implementation for this algorithm.

Name	**Symbol**	**Value**
Minibatches		Full-Batch
Batch-Size		256
TD-Lambda	λ	0.9
Discount	γ	0.9
Policy-Ratio clipping	ϵ	0.2
Standardize Advantages		False
Exact Policy Entropy		True
Value Loss Scale		1.0
Policy Loss Scale		1.0
Entropy Loss Scale		0.1

Table 2: Proximal Policy Optimization loss parameters. Note that we use our recurrent implementation for this algorithm. All configurations were repeated for $r = 30$ repetitions (distinct random seeds). We also drop configurations that do not induce a valid model, e.g., $k_Z = 0$ and accumulation of Σ is not a valid configuration since there is no window to accumulate over.

Name	**Symbol**	**Value**
Deterministic RNN		
Posterior Dimensionality	n	{32, 64}
Variational RNN		
Posterior Dimensionality	n	{32, 64}
Covariance Parameterization		{Full, Diagonal}
Posterior KL-Penalty	β	$\{1.0, 10^{-2}, 10^{-4}\}$
Number of Posterior Samples	n_Z	$\{1, 5\}$
Laplace VRNN		
Posterior Dimensionality	n	{32, 64}
Covariance Parameterization		{Full, Diagonal}
Posterior KL-Penalty	β	$\{1.0, 10^{-2}, 10^{-4}\}$
Number of Posterior Samples	n_Z	$\{1, 5\}$
History Buffer Window	k_H	$\{1, 10\}$
Latent Variable Window	k_Z	$\{0, 1\}$
Accumulation		$\{(\mu, \Sigma), \Sigma\}$

experiment, these snapshots were taken halfway, and three-quarters way during training in terms of the number of weight-updates.

To make some final informal notes on the choices for the parameters,

- We found that scaling the KL-penalties in the lower bound with hyperparameters that were slightly larger than $\beta > 10^{-2}$ caused posterior collapse (Goyal et al., 2017). Meaning, our posterior simply fitted its parameters to always match the prior despite accumulating more data.

- For the regression problem, using multiple samples for $z^{(i)}$ inside the lower bound decreased the predictive loss a lot. Showing that integration over the predictive is effective. Although, this did not result in better test-time performance.
- Using a small buffer size for the history window in the posterior $q_\theta(Z_t|H_{t-k:t-1})$ is more practical since the computation of the Jacobians is the main bottleneck of our method. We found that accumulation of only the covariance where each covariance is computed with a window of just 1, H_{t-1} results in the fastest method (in wallclock time) while being on par with many of the ablations.

C Supplementary Results

C.1 Supplementary Supervised Results

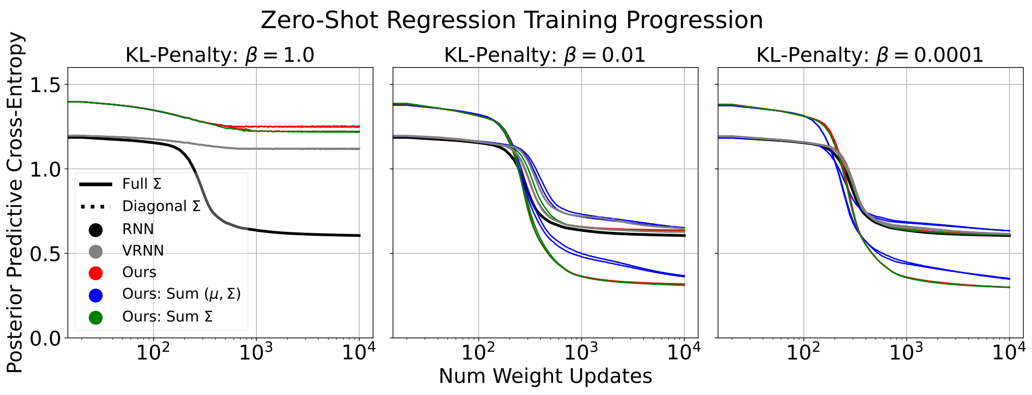

Figure 6: Progression of the predictive error during supervised model training of all ablations. Ablations are averaged over parameter groups as indicated by the legend. This figure does not show the finetuning results. To reduce computation time for subsequent results, we picked $\beta = 0.01$ for the reinforcement learning task ablations.

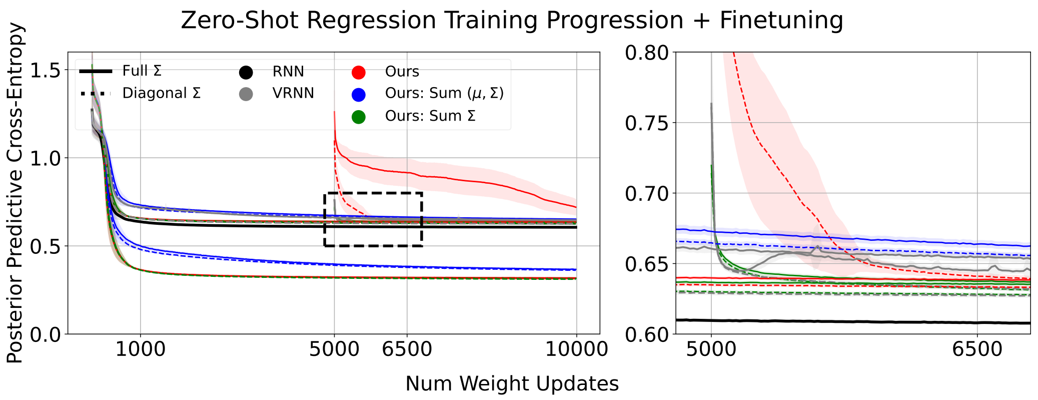

Figure 7: Zoom-in of Figure 6 for three finetune runs (left), for most ablations the predictive error quickly goes down to their full variational training error.

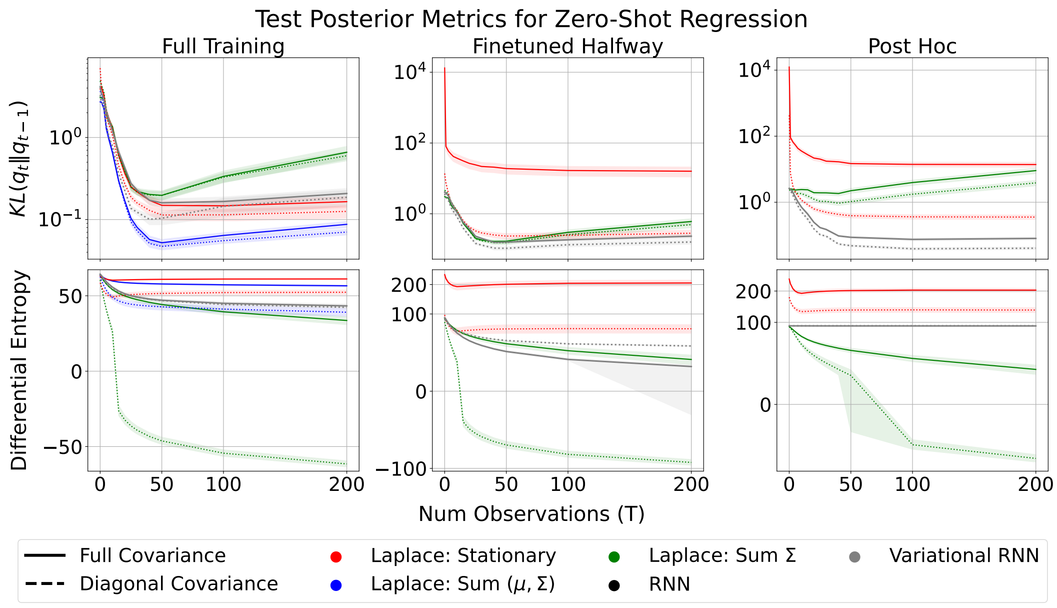

Figure 8: Posterior statistics during testing on the supervised task. The consecutive KL divergences (top) and entropy (bottom) should go down over time. The Laplace VRNN where we sum the covariances (green) is the only method that performs as expected for the entropy estimation but seems to grow more unstable for the KL-divergences. Results use $\beta = 10^{-2}$.

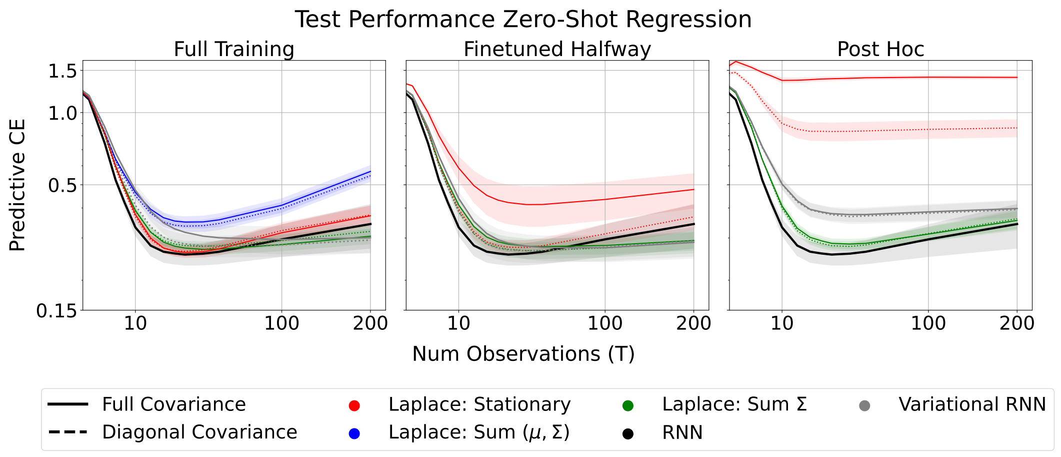

Figure 9: Complete plot of Figure 3 from the main paper to include the accumulation over means and covariances (blue) in the left plot. This was left out of the main paper to improve visibility. Results use $\beta = 10^{-2}$.

C.2 Supplementary Bandit Results

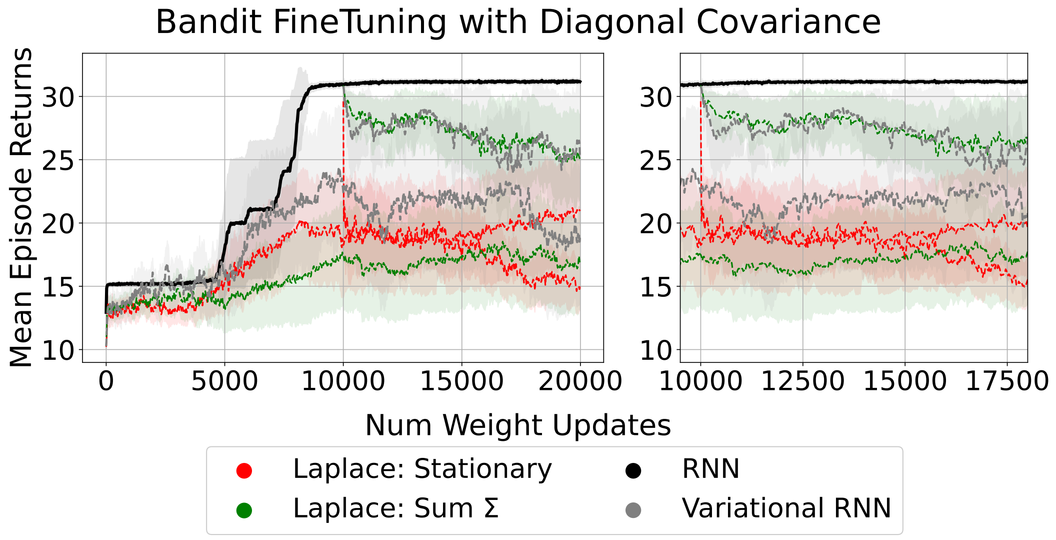

Figure 10: Zoom-in of Figure 5 for the bandit task when finetuning the deterministic model weights intermittently with a diagonal variational model. In this domain, it seems that finetuning slightly actually helps the expected training performance. Results use $\beta = 10^{-2}$.

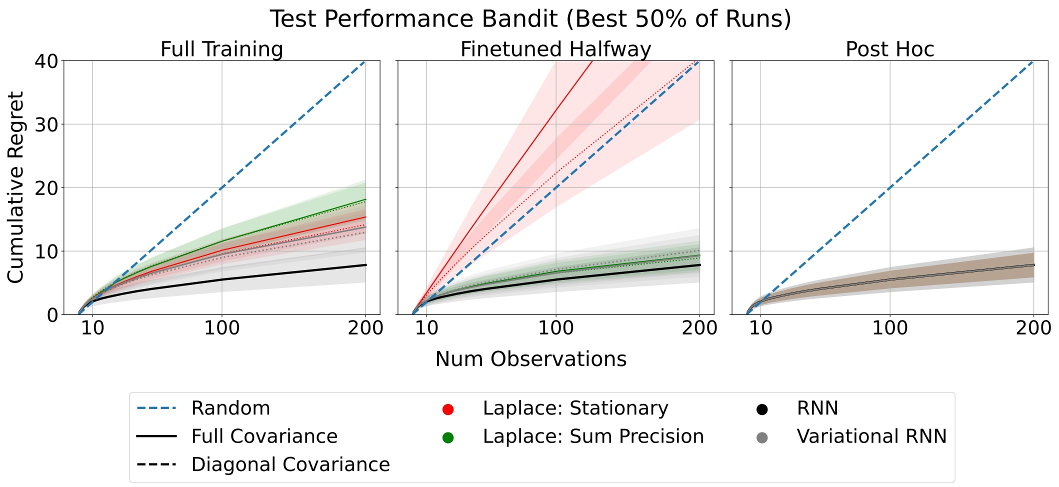

Figure 11: Cumulative regret of trained agents on the bandit task, lower is better. Since the training was quite unstable, about half of the repetitions did not find model weights with strong final performance. Only when we filtered out the better-than-median agents, did we find stronger than random performance. Results use $\beta = 10^{-2}$.

C.3 Supplementary Gridworld Results

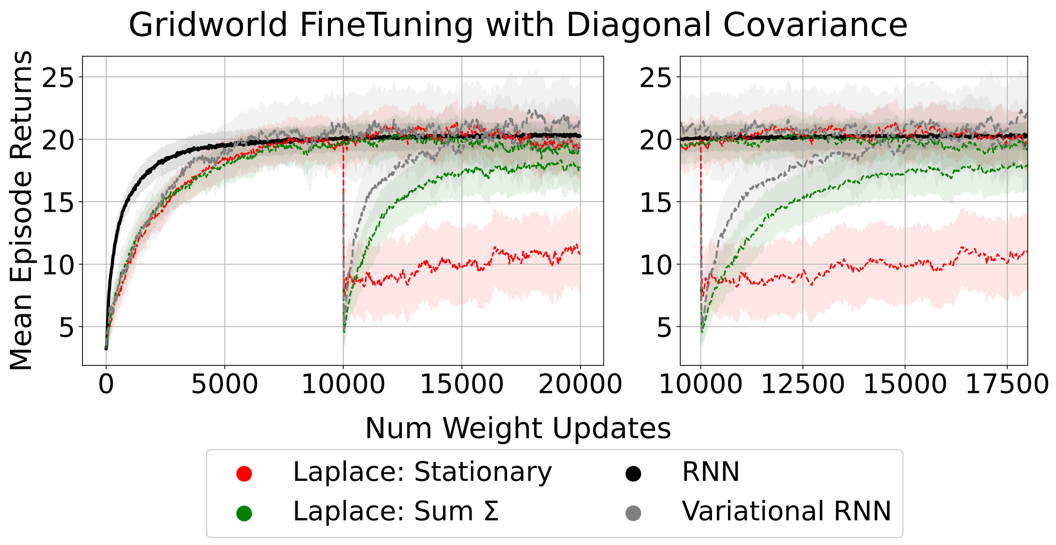

Figure 12: Zoom-in of Figure 5 for the gridworld task when finetuning the deterministic model weights intermittently with a diagonal variational model. In contrast to the bandit task, the agent needs to recover from this sudden change of additional model noise. Results use $\beta = 10^{-2}$.

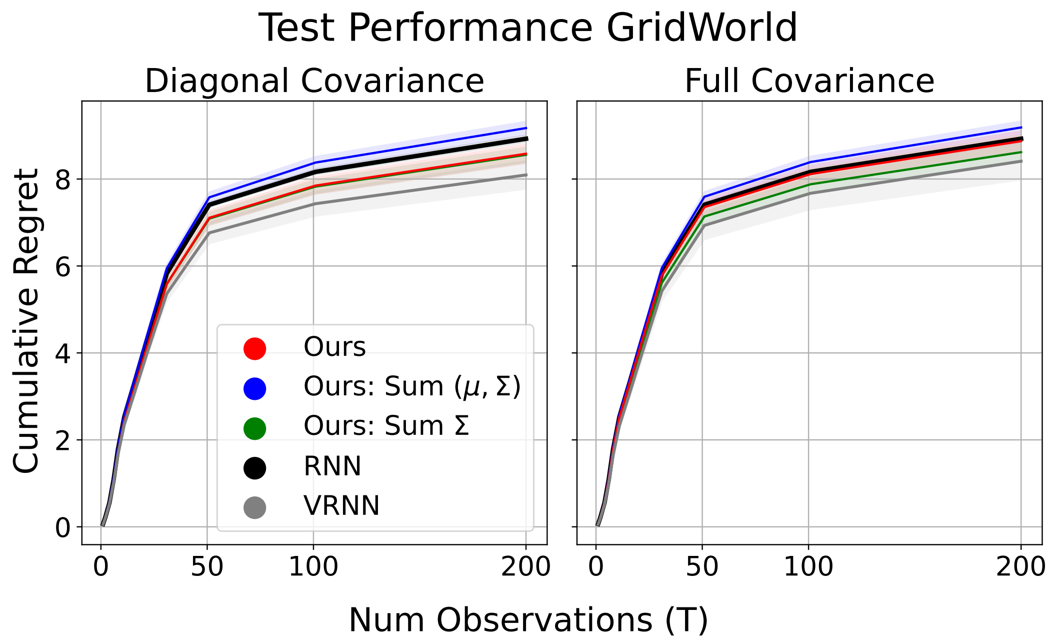

Figure 13: Cumulative regret of trained agents on the gridworld task, lower is better. All agents perform well on this task, the diagonal Variational RNN slightly outperforms all other agents.

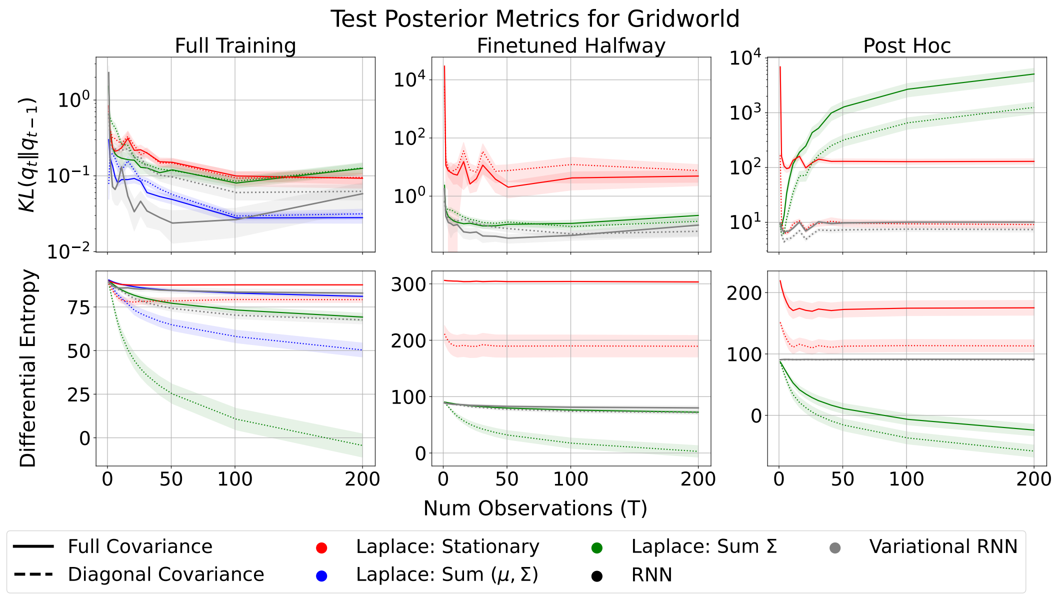

Figure 14: Complete plot of Figure 4 from the main paper to include the accumulation over mans and covariances (blue) in the left plot. Posterior statistics during testing on the gridworld task. The consecutive KL divergences (top) and entropy (bottom) should go down over time. Results use $\beta = 10^{-2}$.