Exchangeable Models in Meta Reinforcement Learning

Anonymous Authors[1]

Abstract

One recent approach to meta reinforcement learning (meta-RL) is to integrate models for task inference with models for control. The former component is often based on recurrent neural networks, which do not directly exploit the exchangeable structure of the inputs. We propose to use a lightweight, yet an expressive architecture that accounts for exchangeability. Combined with an off-policy reinforcement learning algorithm, it results in a meta-RL method that is sample-efficient, fast to train and able to quickly adapt to new test tasks as demonstrated on a couple of widely used benchmarks.

1. Introduction

The intent of meta-learning is to bridge the gap between machine and human learning by designing algorithms that acquire prior knowledge from a multitude of training tasks and use these priors for a rapid adaptation to new tasks. In reinforcement learning (RL), this goal is translated into building agents that are able to adjust their policies to new environment settings after a handful of environment interactions and with little or no retraining.

One popular group of methods in meta-RL separates the algorithm into inference and acting modules. The former is responsible for inferring the task from a sequence of interactions, while the latter needs to choose actions conditionally on the results of inference. This approach is motivated by the formulation of the meta-RL problem as solving a special type of a partially observed Markov decision process called Bayes-adaptive MDP (BAMDP): one in which the task specification is hidden from the agent (Duff & Barto, 2002). Each task on its own, however, can be described by a Markov decision process (MDP). As in most meta-RL problems, we assume that the MDPs have the same action and state spaces but different transition functions $\mathcal{T} = p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ and/or reward functions $\mathcal{R} = p(r|\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$.

As we mentioned, the goal of the inference module is to maintain a belief state over what the underlying MDP might be, or equivalently, the MDP's transition and reward functions. At any given step t, this belief is a posterior distribution $p(\mathcal{T}, \mathcal{R}|\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})$, where we condition on previously observed transitions within a trajectory. Alternatively, we can reason in terms of beliefs $p(\boldsymbol{\theta}|\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})$ over a latent variable $\boldsymbol{\theta}$ that encapsulates task specification and thus determines which MDP we are in, i.e. $\mathcal{T} = p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})$ and $\mathcal{R} = p(r|\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', \boldsymbol{\theta})$. In either case, the posterior is often intractable.

Variational inference as implemented in variational autoencoders (VAEs) (Kingma & Welling, 2014) has been previously used to approximate the belief state (Zintgraf et al., 2020). In this paper, we explore possibly better ways of doing so without the help of VAE's. A model called BRUNO (Korshunova et al., 2019), which was originally designed for few-shot image generation in a classical meta-learning scenario, is the alternative we are interested in. BRUNO has a number of appealing properties, among which is the exact computation of the posterior predictive distribution and the existence of recurrent updates for its parameters. In this model, the predictive distribution is constructed without a reference to the posterior, so we will derive it before BRUNO can become a full-fledged replacement to VAEs in meta-RL. With the addition of minor architectural changes, we use BRUNO in conjunction with a soft actor-critic (SAC) RL algorithm (Haarnoja et al., 2018) to get a competitive meta-RL algorithm, which we will refer to as *BrunoSAC*. Based on a couple of benchmarks, we show that our method is sample efficient, fast and easy to train, and it can adapt to the test tasks given a small number of observations. Our code will be released with the final version of this paper.

2. Method

2.1. Posterior computation

Before we focus on problem of computing the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})$, let us revise the concept of exchangeability – the main modelling assumption we are going to make. For some stochastic process of tu-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

ples $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$, where y_i depends on x_i, exchangeability amounts to:

$$p(y_1, \ldots, y_t | x_1, \ldots, x_t) = p\left(y_{\pi(1)}, \ldots, y_{\pi(t)} | x_{\pi(1)}, \ldots, x_{\pi(t)}\right), \tag{1}$$

which holds for any finite t and any permutation π of $\{1, \ldots, t\}$.

In MDPs, we deal with sequences of state-action-reward-state transitions $(\boldsymbol{s}_1, \boldsymbol{a}_1, r_1, \boldsymbol{s}'_1), (\boldsymbol{s}_2, \boldsymbol{a}_2, r_2, \boldsymbol{s}'_2), \ldots$, and it is reasonable to assume that they form a conditionally exchangeable process with $x_i = (\boldsymbol{s}_i, \boldsymbol{a}_i)$ and $y_i = (r_i, \boldsymbol{s}'_i)$ if we use the notation from Eq. 1.

How can exchangeability help us to find the posterior $p(\boldsymbol{\theta} | \boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})$ we are after? The answer is given by de Finetti's theorem, which relates exchangeability to Bayesian inference. It justifies the existence of a latent variable $\boldsymbol{\theta}$ underlying the exchangeable stochastic process, thus the following holds[1]:

$$p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\tau}_{1:t}) = \int p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \boldsymbol{\tau}_{1:t}) d\boldsymbol{\theta}, \tag{2}$$

with $\boldsymbol{\tau}_{1:t} = \{(\boldsymbol{s}_1, \boldsymbol{a}_1, r_1, \boldsymbol{s}'_1), \ldots, (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)\}$ denoting all observed transitions up to step $t \geq 0$ either in their natural or in a permuted order. The former assumes that $\boldsymbol{s}'_t = \boldsymbol{s}_{t+1}$, while this is not necessarily true for the latter.

Using de Finetti's theorem, we can reason out why conditional exchangeability of $(r, \boldsymbol{s}')$ given $(\boldsymbol{s}, \boldsymbol{a})$ is a valid assumption. This theorem allows to think about exchangeable processes as sequences of random variables that are independent and identically-distributed (i.i.d.) conditionally on an underlying latent factor, which we denoted by $\boldsymbol{\theta}$. In our case, $\boldsymbol{\theta}$ encapsulates the knowledge of the MDP with its reward and transition functions. Then, given $\boldsymbol{\theta}$, and conditionally on $(\boldsymbol{s}, \boldsymbol{a})$, $(r, \boldsymbol{s}')$ become i.i.d., while without $\boldsymbol{\theta}$, they are correlated. This conditional independence might still seem unintuitive, especially since we often think of $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ transitions in the order they appear within a trajectory. Therefore, it is important to additionally remember the Markov property and note that the current state $\boldsymbol{s}$ is always the conditioning event.

If we wish to model the distributions involved in Eq. 2 using VAE-based models, e.g. neural processes (Garnelo et al., 2018), we need to approximate the posterior $p(\boldsymbol{\theta} | \boldsymbol{\tau}_{1:t})$ and derive a lower bound on $\log p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\tau}_{1:t})$. BRUNO, on the other hand, can directly model this predictive distribution by constructing a suitable exchangeable process while ignoring the integral on the right hand side.

[1] de Finetti's theorem for conditional probabilities is commonly accepted as being true, however, it has not been formally proven.

We do, however, need the posterior $p(\boldsymbol{\theta} | \boldsymbol{\tau}_{1:t})$ for the meta-RL algorithm, so let us deal with it after a brief explanation of the BRUNO model.

BRUNO combines Gaussian processes (GPs) with a deep bijective Real NVP mapping (Dinh et al., 2017). GPs are defined in the feature space $\mathcal{Z}$ of the Real NVP, where dimensions $z^1, \ldots z^D$ are modelled independently. BRUNO uses the simplest type of GPs: for every finite n, the assumption is that $z^d_1, \ldots, z^d_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma}_{ii} = v^d$ and $\boldsymbol{\Sigma}_{ij} = \rho^d$. This simplicity allows to derive recurrent updates for parameters of $p(\boldsymbol{z}_{t+1} | \boldsymbol{z}_{1:t})$. Predictive distribution in the input space $\mathcal{Y}$ can then be evaluated using the change of variables formula, which gives $p(\boldsymbol{y}_{t+1} | \boldsymbol{y}_{1:t})$ or $p(\boldsymbol{y}_{t+1} | \boldsymbol{y}_{1:t}, \boldsymbol{x}_{1:t+1})$ if we condition Real NVP on some extra input $\boldsymbol{x}$. Training of BRUNO amounts to maximum likelihood estimation with optimizing variances and covariances of GPs and parameters of the Real NVP.

By setting $\boldsymbol{y}_t = (r_t, \boldsymbol{s}'_t)$ and $\boldsymbol{x}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t)$, we get BRUNO to model the predictive distribution in Eq. 2. However, we also need the posterior $p(\boldsymbol{\theta} | \boldsymbol{\tau}_{1:t})$ or, equivalently, $p(\boldsymbol{\theta} | \boldsymbol{y}_{1:t}, \boldsymbol{x}_{1:t})$. This posterior is not mentioned in the derivation of BRUNO, still it exists. The use of a bijective mapping and independent dimensions in $\mathcal{Z}$ implies that $p(\boldsymbol{\theta} | \boldsymbol{y}_{1:t}, \boldsymbol{x}_{1:t}) = p(\boldsymbol{\theta} | \boldsymbol{z}_{1:t}) = \prod_{d=1}^{D} p(\theta^d | z^d_{1:t})$. Thus, we only need an expression for the univariate posterior $p(\theta^d | z^d_{1:t})$. Further, we will drop the index d since the same equations hold for every dimension in $\mathcal{Z}$, but please note that a GP associated with a d-th dimension has its own values of v and ρ parameters. For the type of GPs used in BRUNO, the likelihood $p(z_i | \theta) = \mathcal{N}(\theta, v - \rho)$ and the prior $p(\theta) = \mathcal{N}(0, \rho)$. The conjugate analysis concludes that the posterior is also Gaussian (Murphy, 2007): $p(\theta | z_{1:t}) = \mathcal{N}(\mu_t, \sigma^2_t)$. Its parameters can be updated recursively starting from $\mu_0 = 0$ and $\sigma^2_0 = \rho$ as:

$$\begin{aligned} \mu_{t+1} &= (1 - b_t)\mu_t + b_t z_t \\ \sigma^2_{t+1} &= (1 - b_t)(\sigma^2_t - v + \rho), \end{aligned} \tag{3}$$

with $b_t = \frac{\rho}{v + \rho(t-1)}$.

2.2. The bottleneck problem

There remains one more obstacle before we can successfully use BRUNO to model $p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\tau}_{1:t})$ and $p(\boldsymbol{\theta} | \boldsymbol{\tau}_{1:t})$: its inefficiency when dealing with complex low-dimensional inputs and its apparent inability to handle 1D inputs. This is the result of having to use a bijective mapping such as Real NVP. An existing solution is to augment the inputs with extra dimensions whose values are drawn from $\mathcal{N}(0, 1)$. Introduction of these dimensions allows for more expressive models, however, we will no longer be able to compute the predictive posterior exactly during training. Instead, we will maximize its lower bound. Huang et al. (2020) refer to this as augmented maximum likeli-

hood (AMLE). While there is a connection to the VAE's objective, AMLE training is still neater as, for instance, we do not have to deal with a two-part loss function or the reparameterization trick.

In addition, we will replace Real NVP with a more flexible bijective architecture called masked autoregressive flow (MAF) (Papamakarios et al., 2017),

2.3. BrunoSAC

We can now combine the modified BRUNO model with a soft actor-critic into a meta-RL algorithm. Most of our choices resemble those used in VariBAD (Zintgraf et al., 2020), PEARL (Rakelly et al., 2019) and Belief (Humplik et al., 2019) – methods that will be discussed in the next section.

The crux of BrunoSAC is to have the SAC policy depend on parameters of the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{1:t})$ given by BRUNO. As we showed, this posterior is a multivariate Gaussian with independent dimensions, so it can be described by its mean and variance parameters: $\boldsymbol{m}_t^{\boldsymbol{\theta}} = \{\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2\}$. Having observed t transitions, the agent can sample an action from a stochastic policy conditioned on a state $\boldsymbol{s}$ and current posterior parameters $\boldsymbol{m}_t^{\boldsymbol{\theta}}$, i.e. $\boldsymbol{a} \sim \pi(\boldsymbol{s}, \boldsymbol{m}_t^{\boldsymbol{\theta}})$.

We train BRUNO separately from the actor and critics, although all these components use the same replay buffer with off-policy data. Training sequences $\boldsymbol{\tau}$ are constructed from $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ transitions that come from the same MDP, i.e. not necessarily the same trajectory. Knowledge of how to group transitions according to which MDP they belong is privileged information. However, since the design of the meta-training stage is under the researchers' control, this information comes for free.

Given a sequence $\boldsymbol{\tau}$ of T transitions, the objective of BRUNO is to maximize the likelihood of each observed $(r_t, \boldsymbol{s}'_t)$ given $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and $\boldsymbol{\tau}_{1:k}$ – a part of the sequence before step k. In other words, the goal is to 'decode' both past and future transitions with respect to k. In this case, our loss can be written as:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p(r_t, \boldsymbol{s}'_t|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\tau}_{1:k}). \tag{4}$$

Ideally, one would sample a random k for every training sequence, however, we found that sampling k per batch of sequences or even choosing a fixed k works well in practice. Moreover, there is flexibility in choosing what to model. For example, if we know that MDPs differ only with respect to their reward function $\mathcal{R}$, then we can safely withhold from modelling the next state distibution $\mathcal{T}$ since it does not contribute any relevant information to our posterior. However, once we are left with predicting scalar rewards, it becomes important to use the augmented input space as we described in the previous section.

Training of the actor and critics in BrunoSAC remains unchanged in comparison to the original SAC algorithm except for the following few modifications. Firstly, we adapted SAC to work with recurrent policies. Evidently, this is the case for us since $\pi(\boldsymbol{s}, \boldsymbol{m}_t^{\boldsymbol{\theta}})$ depends on $\boldsymbol{m}_t^{\boldsymbol{\theta}}$ that we compute using recurrent updates in Eq. 3 for every step t. Secondly, we condition the critics on the true task specification, e.g. the target direction or velocity of a robot. In all the problems we can think of, this information is available, so not using it makes the training needlessly harder. In either case, whether we condition on the true task specification or on $\boldsymbol{m}_t^{\boldsymbol{\theta}}$ parameters, the critics are discarded at test time.

3. Related work

In this section, we will not try to review the vast number of methods from areas of meta-learning, meta-RL, Bayesian RL, POMDPs, etc. that are related to BrunoSAC. Instead, we will focus on the most relevant ones: Belief (Humplik et al., 2019), PEARL (Rakelly et al., 2019) and VariBAD (Zintgraf et al., 2020). Like BrunoSAC, these three methods implement the approach of conditioning the policy on results of the task inference. Their main differences can be identified by asking the following: 1) how is privileged information, i.e. task IDs or specifications, used during meta-training? 2) what type of model is used to process sequences of transitions? 3) what information from the inference module is passed on to the policy and, optionally, to the value functions? 4) which RL algorithm is used? We will answer these questions next.

Belief uses a supervised approach to learn the belief state. Namely, the inference network is trained to directly predict the task description or its ID given a trajectory. To process the trajectories, Belief uses an LSTM-based architecture (Hochreiter & Schmidhuber, 1997). Features from the penultimate layer of this model are passed to the actor and critics of an off-policy SVG(0) (Heess et al., 2015) or an on-policy PPO (Schulman et al., 2017) RL algorithms. The off-policy method is concluded to be preferable for its sample efficiency.

PEARL is identical to BrunoSAC with respect to how it groups transitions according to their tasks. Moreover, it makes similar exchangeability assumptions and constructs a permutation-invariant inference network for $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{1:t})$, though based on VAEs. While PEARL allows for a decoder that could predict future states and rewards, the authors prefer to predict q-values instead. The policy is trained with SAC, where both policy and the critics are conditioned on samples from the VAE's posterior. Using samples is perhaps the reason why PEARL needs a lot more transitions before converging to a reasonable behaviour at test time, while

others adapt after very few steps.

VariBAD uses no privileged information during training and works based on trajectories. To process them, VariBAD applies VAEs with a recurrent neural network as an encoder and an MLP decoder that predicts past and future rewards r and states $\boldsymbol{s}'$. Parameters of the posterior distribution are supplied to the policy and critics of PPO. Since PPO is an on-policy method, VariBAD is relatively sample inefficient. Combined with a slow recurrent encoder, this increases the training time of VariBAD in comparison, for instance, to PEARL.

To conclude, each of these methods makes different design choices, whose compatibility, in our opinion, is sometimes unjustified. The reason is that they trade off some desirable properties such as simplicity of the implementation, sample efficiency, fast adaptation at test time or short training times. BrunoSAC is, therefore, our attempt in combining elements that we think are most sensible with respect to the listed criteria.

4. Experiments

We applied BrunoSAC to two popular meta-RL benchmarks introduced by Finn et al. (2017): Cheetah-Dir and Cheetah-Vel. In the first one, the simulated cheetah robot (Todorov et al., 2012) needs to run as fast as possible either forward or backwards. These two directions are the only tasks, so we use them both during training and testing. While such setup is not ideal in terms of estimating the agent's ability to adapt to the unseen tasks, Cheetah-Dir is still a difficult problem that cannot be solved by RL algorithms with non-recurrent policies. Cheetah-Vel, on the other hand, does have a different set of tasks for training and testing. Here, the task is to run with a certain velocity. There are 100 train and 30 test tasks with a target velocity sampled once from $\mathcal{U}(0, 3)$. Such settings were previously used in PEARL and VariBAD. However, we will not directly compare these methods to BrunoSAC since a fair comparison is only possible if all three methods run under the same conditions. In either case, we will try to relate our results to those of PEARL and VariBAD when it is meaningful to do so.

Before we look at the results of BrunoSAC, let us discuss several hyperparameter choices. During training, we roll out the trajectories of length 200, however, it does not oblige us to train BRUNO on sequences of the same length (parameter T in Eq. 4). In our experiments, we used sequences of 100 transitions, and for every batch of sequences, we sampled k from $\mathcal{U}(25, 75)$ to compute the loss in Eq. 4. Since in Cheetah-Dir and Cheetah-Vel the states transition function is the same across all tasks, we train BRUNO to predict only the rewards. In order to use MAF, we add 4 extra Gaussian noise dimensions, which results in having a 5-dimensional latent space for $\boldsymbol{\theta}$.

In Figures 1 and 2 we plot learning curves of BrunoSAC and the oracle SAC. The latter has its policy conditioned on the true task specification, i.e. one-hot encoding of the forward-backward direction for Cheetah-Dir and a scalar target velocity for Cheetah-Vel. For the latter problem, the oracle is trained on 30 test tasks. Figures 3 and 4 plot the rewards obtained by our trained models when we roll out policies on the test tasks. Similarly to VariBAD, but unlike PEARL, our model requires a handful of observations to infer what the task is and to adapt its behaviour accordingly.

One finding we would like to highlight is that we get similar performance regardless of whether we condition the policy on the posterior mean and variance or on the mean alone. The redundance of variance indicates that cheetah benchmarks are not suited for exploring the role of uncertainty over tasks. We also admit that the variance we have in our model is inadequate since it does not depend on the data. In future, Student-t processes should be used instead of GPs as suggested by Korshunova et al. (2018).

5. Discussion and conclusion

We presented a meta-RL method that relies on an exchangeable BRUNO architecture suitably repurposed for doing task inference. The latter ability stems from trying to model the reward and transition functions of multiple MDPs. This also makes BRUNO appropriate for model-based RL in meta-learning settings, similar to how GPs or neural processes are used (Sæmundsson et al., 2018; Galashov et al., 2019). For instance, one could use planning algorithms based on predictive probabilities of rewards and next states given by BRUNO. In this paper, however, we focused on a different approach in which both BRUNO and the policy are trained during the meta-training stage, and no changes are made at test time.

Our BrunoSAC advocates in favour of exchangeable architectures for task inference in meta-RL. Sometimes, they are seen as being more restrictive compared to recurrent neural networks (Zintgraf et al., 2020). However, this a sensible restriction since it directly encodes the basic property of the processes we wish to model, as we argued in Section 2.1. The preliminary results of BrunoSAC on the two common benchmarks support our claim, though we admit that any definite conclusions can only be made after an extensive ablation study.

An interesting application of BrunoSAC could be sim2real: a domain adaptation task addressing the simulation to reality gap (Higgins et al., 2017). In this problem, we want to find a policy to apply to a plant, for instance a robot, if we only have an imperfect model of the plant available in simulation. It is a common issue that policies which work

well on the simulator, fail to perform on the plant because of the imperfections in the simulator. In this case, the meta-RL setup is useful: we train BrunoSAC on a distribution of simulators, and then test its policy on the plant, where the agent autonomously narrows down its beliefs over $\boldsymbol{\theta}$ and adjusts its control accordingly.

From the sim2real perspective, we are skeptical that common meta-RL benchmarks give an indication of how performant the algorithms might be in reality. For instance, Cheetah-Dir and Cheetah-Vel use the same transition function $\mathcal{T}$, while only changing the reward function $\mathcal{R}$ across MDPs. This the less useful case for sim2real, where usually $\mathcal{T}$ varies and $\mathcal{R}$ is fixed. Moreover, the meta-RL problem with different reward functions is substantially simpler compared to when the MDP's state transition graph defined by $\mathcal{T}$ changes from one task to another. Only in this latter case, the set of possible trajectories in the environment changes.

BrunoSAC follows the approach of VariBAD and Belief, which use the BAMDPs theory to justify their methods. However, we are uncertain whether theoretical results extend to what is implemented in practice. Namely, BAMDPs assume there exists a Bayes-optimal policy that optimizes the expected return in an MDP whose original states are extended with the belief states. In meta-RL, we train on a number of tasks and, given a sufficiently powerful model, we can find an optimal policy for each of them. But what happens when test tasks are different from the ones we trained on? The theory does not help us here, so we can only hope that the policy makes good use of the unfamiliar belief states that it sees at test time. How far such policy is from the optimal is a difficult question to answer. This problem does not appear to be illustrated appropriately in the commonly used locomotion benchmarks.

There is another argument against the approach of separating inference and acting modules as BrunoSAC, VariBAD, PEARL and Belief do. While task inference is based on Bayesian principles, the policy that uses outputs of the Bayesian model to select actions is implemented as a black-box neural network. In many cases, we cannot guarantee a neural network to behave reasonably when presented with far-out inputs. This leads to a natural question whether the policy itself can be made Bayesian while encapsulating task inference procedures in a principled way. We consider this to be an interesting future line of research, where exchangeable models might play an important role.

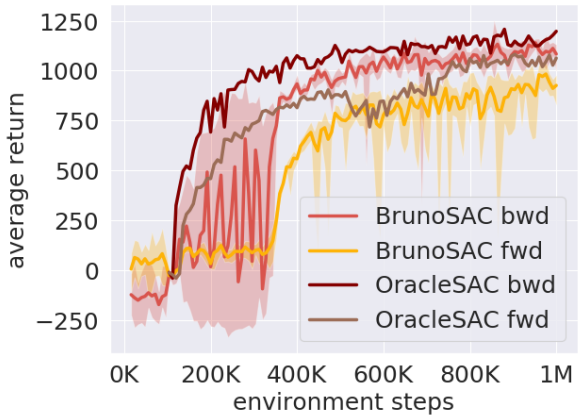

Figure 1. Cheetah-Dir test returns versus the number of environment interactions during training. Returns are averaged over 10 trajectories per task. Shaded areas represent minimum and maximum returns for BrunoSAC. Each trajectory is 200 steps long. Returns from the two tasks are plotted separately to illustrate the asymmetry between learning how to run forward and backwards. We see that after ∼500K collected transitions, BrunoSAC approaches the oracle's performance, which could only mean that tasks are inferred correctly. To our best estimate, this is at least an order of magnitude fewer steps than required for PEARL and VariBAD, which respectively need to train for 24 hours and 48 hours as reported by Zintgraf et al. (2020). For comparison, it takes about 8 hours to train BrunoSAC on a laptop CPU.

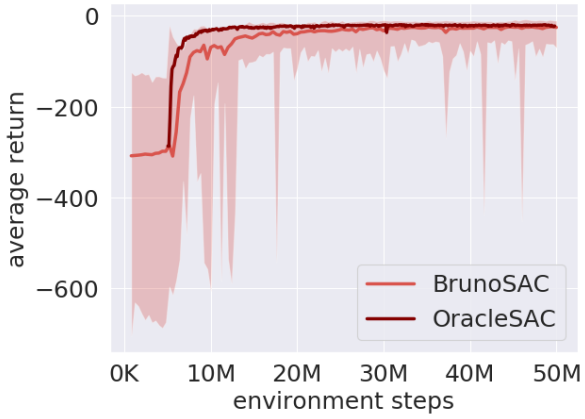

Figure 2. Cheetah-Vel test returns averaged over 150 trajectories (5 trajectories per test task) with each trajectory having 200 steps. We can see that BrunoSAC closely approaches the performance of the oracle which was trained directly on the test tasks. VariBAD requires twice as many environment steps to catch up with the oracle, while PEARL needs a bit more than a million steps, which is surprising since it amounts to learning only from 50 trajectories per training task.

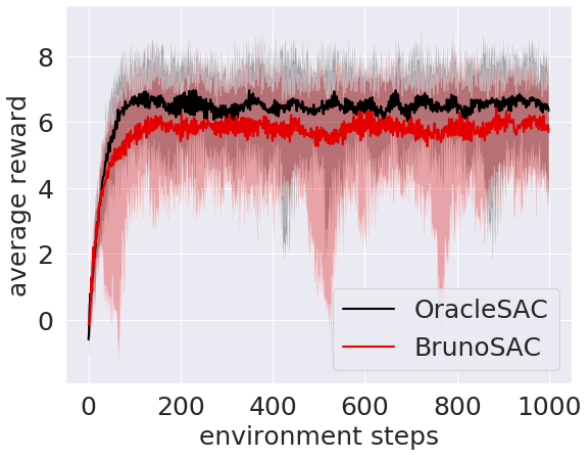

Figure 3. Cheetah-Dir test rewards for every environment step. Rewards are averaged over 20 trajectories (10 per task). Shaded regions represent minimum and maximum rewards. Average returns of BrunoSAC and the OracleSAC are 5645 and 6319 respectively. Our results are incomparable to those of VariBAD or PEARL, where the maximum return is ∼2000. This is likely due to using different versions of RL toolkits.

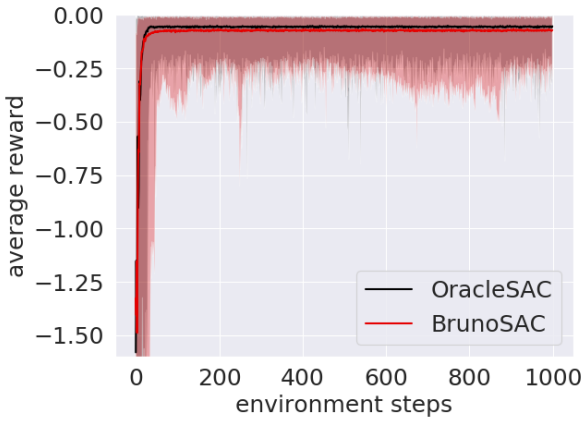

Figure 4. Cheetah-Vel average rewards for each step. Rewards are averaged over 300 trajectories (10 per test task). Average returns of BrunoSAC and the OracleSAC are -83 and -66 respectively.

References

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. In *International Conference on Learning Representations*, 2017.

Duff, M. O. and Barto, A. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, Univ of Massachusetts at Amherst, 2002.

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

Galashov, A., Schwarz, J., Kim, H., Garnelo, M., Saxton, D., Kohli, P., Eslami, S. M. A., and Teh, Y. W. Meta-learning surrogate models for sequential decision making. *ArXiv*, abs/1903.11907, 2019.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. Neural processes. *Theoretical Foundations and Applications of Deep Generative Models, ICML workshop*, 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems 28*. 2015.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1480–1490. JMLR. org, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. ISSN 0899-7667.

Huang, C., Dinh, L., and Courville, A. C. Augmented normalizing flows: Bridging the gap between generative flows and latent variable models. *ArXiv*, abs/2002.07101, 2020.

Humplik, J. F., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. Meta reinforcement learning as task inference. *ArXiv*, abs/1905.06424, 2019.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Korshunova, I., Degrave, J., Huszar, F., Gal, Y., Gretton, A., and Dambre, J. BRUNO: a deep recurrent model for exchangeable data. In *Advances in Neural Information Processing Systems 31*. 2018.

Korshunova, I., Gal, Y., Gretton, A., and Dambre, J. Conditional BRUNO: a neural process for exchangeable labelled data. In *Proceedings of the 27th European Symposium on Artificial Neural Networks*, 2019.

Murphy, K. P. Conjugate bayesian analysis of the gaussian distribution. Technical report, 2007.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems 30*. 2017.

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Sæmundsson, S., Hofmann, K., and Deisenroth, M. P. Meta reinforcement learning with latent variable gaussian processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2020.