Decision-Aware Model Learning for Actor-Critic Methods: When Theory Does Not Meet Practice

Anonymous Author(s)
Affiliation
Address
email

Abstract

Actor-Critic methods are a prominent class of modern reinforcement learning algorithms based on the classic Policy Iteration procedure. Despite many successful cases, Actor-Critic methods tend to require a gigantic number of experiences and can be very unstable. Recent approaches have advocated learning and using a world model to improve sample efficiency and reduce reliance on the value function estimate. However, learning an accurate dynamics model of the world remains challenging, often requiring computationally costly and data-hungry models. More recent work has shown that learning an everywhere accurate model is unnecessary and often detrimental to the overall task; instead, the agent should improve the world model on task-critical regions. For example, in Iterative Value-Aware Model Learning, the authors extend model-based value iteration by incorporating the value function (estimate) into the model loss function, showing the novel model objective reflects improved performance in the end task. Therefore, it seems natural to expect that model-based Actor-Critic methods can benefit equally from learning value-aware models, improving overall task performance, or reducing the need for large, expensive models. However, we show empirically that combining Actor-Critic and value-aware model learning can be quite difficult and that naive approaches such as maximum likelihood estimation often achieve superior performance with less computational cost. Our results suggest that, despite theoretical guarantees, learning a value-aware model in continuous domains does not ensure better performance on the overall task.

1 Introduction

Actor-Critic (AC) methods [17, 19, 26, 40, 41] are a prominent class of model-free reinforcement learning (RL) algorithms based on the classic Policy Iteration procedure. The method consists in iterating over two main steps: evaluating the agent's policy (updating the value function estimate of the expected policy returns) , and improving the current policy. Combined with recent advances in deep neural network function approximators [16], AC methods have obtained impressive results in many challenging sequential decision making problems such as playing Atari games from image pixel input [40] and learning complex robot gaits in simulation [22].

Recently, there has been a growing interest in extending AC methods by progressively learning and using a model of the environment dynamics [7, 8, 10, 15, 23]. The use of a dynamics model (a.k.a. *world model*) has been shown to greatly improve sample-efficiency [27, 32, 39], an important aspect in environments with costly actions (which are arguably more common when one ventures outside of simulation-based tasks, such as in robotics and industrial applications). Using a world model can also reduce the algorithm's reliance on the quality of the estimated value-function through mechanisms such as multistep bootstrapping. Moreover, recent work shows that learning a world model is often

Submitted to 1st I Can't Believe It's Not Better Workshop (ICBINB@NeurIPS 2020). Do not distribute.

easier than finding an accurate value function approximation [9], increasing the stability of RL methods. Finally, having access to a world model unlocks different learning modalities, such as learning value functions in Sobolev spaces [10].

Yet learning a world model introduces its own challenges. For instance, the model class may be *misspecified* [24], meaning that the function approximator used cannot accurately represent the true dynamics everywhere on the state space (or even in most of it). In such cases, by attempting to improve the average accuracy standard model learning approaches such as Maximum Likelihood Estimation (MLE) may sub-utilize the model capacity and fail at aiding policy learning.

Decision-aware model learning (DAML) [1, 11, 13, 14] tackles this problem by taking the decision making objective into account when optimizing for the world model. Iterative value-aware model learning [13] (IterVAML), for instance, introduces a novel model loss function which incorporates the current estimate of the value function. Under assumptions on the sample collection, model space, and value function space, the authors of IterVAML were able to theoretically link the performance of a model-based agent as a function of, among other things, the value-aware model-error (in particular they have shown that the lower that error, the closer the resulting policy is to the optimal one).[1] Although promising in theory, neither the IterVAML work or any subsequent work validated the result in practical deep RL settings.

Given the above discussion, it seems just natural to expect that *one can boost the performance of model-based Actor-Critic (*MBAC*) methods by incorporating value-awareness into model learning*. Guided by that reasoning, in this work we provide an empirical analysis of VAML in the deep RL setting, with neural network models in continuous control environments. We assess the core claims of VAML in carefully constructed artificial settings while varying the capacity of the neural network model. Then, we reformulate IterVAML, originally devised as an approximate value iteration method, as an approximate policy iteration algorithm, thus obtaining a value-aware MBAC method. To our surprise, we found that value-aware learning does not provide a clear and consistent advantage over MLE in our controlled artificial settings. Yet we find some evidence of VAML being advantageous in dynamical systems common to many deep RL benchmarks. However, upon combining value-aware learning and model-based policy improvement, we find that the resulting agent is too unstable and that switching to MLE provides comparable, and sometimes better, performance at lower computational cost. Our results suggest that, despite theoretical guarantees, learning a value-aware model in continuous domains does not ensure better performance on the overall task when compared with conventional likelihood-based model learning methods such as MLE. We speculate that some of the assumptions behind the IterVAML framework may not transfer to the deep RL setting and that poorly learned values may negatively impact model optimization.

The rest of this paper is organized as follows. Section 2 delineates related work on model-based actor-critic methods and decision-aware model learning. Section 3 lays out important background on Actor-Critic methods and value-aware model learning. Section 4 introduces our proposed adaptation of IterVAML to the Actor-Critic setting. Section 5 introduces our proposed experiments and discusses the positive and negative results from each. Finally, section 6 concludes with a discussion of the possible explanations for the gap between theory and practice, while also pointing to future research directions to expand our limited understanding of these issues.

2 Related work

Model-based Actor-Critic methods. There has been a growing interest in extending AC methods by progressively learning and using a model of the environment dynamics. The Dyna framework [36] lays the foundation many model-based methods by using the model to generate virtual experiences for use in model-free methods. One of the most successful instantiations of this paradigm in deep RL is the Model-Based Policy Optimization (MBPO) algorithm [23]. MBPO extends the Soft Actor-Critic (SAC) algorithm [19] (an off-policy, model-free AC method) by generating short model-based rollouts branched from real experiences that are then mixed together to augment the experience replay buffer. MBPO is successful in increasing the sample-efficiency of SAC, but learns its model via maximum likelihood, ignoring performance on the end task.

[1] The other relevant quantities in the performance bound are the number of samples and the regression error when learning the value function. Since these are inherent to any RL algorithm, we emphasize here the value-aware error.

Another prominent line of work in model-based AC methods leverages differentiable models to calculate the gradient of final performance in a policy gradient framework. Stochastic Value Gradients (SVG) uses reparameterizable models to calculate analytical policy gradients using real experiences (no virtual samples are generated). SAC-SVG [3] and Model-Augmented Actor-Critic [8] expand this concept by using model gradients and virtual experiences in the policy update of SAC. Dreamer [21] uses many of the same ideas, but its model operates in a compressed state space extracted from pixel observations using variational methods. Only Dreamer, however, incorporates some aspect of the decision-making task by optimizing its model to predict rewards, but not full returns.

Another way to exploit model derivatives is in learning action-values. In Model-based Action-Gradient-Estimator (MAGE) [10], the authors also exploit reparameterizable models to compute action-value gradients used to learn critics in Sobolev space. Built upon the TD3 [17] AC method, MAGE manages to learn better critics that increase sample-efficiency over the model-free TD3.

Decision-aware model learning. The authors of Value-aware Model Learning (VAML) [14] argue that optimizing a probabilistic model loss (as in MLE) is overkill since it does not take into account the end RL task. They show, theoretically and empirically, that incorporating the value function structure in the model cost produces better results when the model capacity is restricted. However, VAML's loss function uses a pessimistic formulation that requires an additional search over the value function space. IterVAML [13] relaxes this formulation by using the current value function estimate, produced along the course of the RL algorithm, in the model loss. The authors show, theoretically, that the value-aware model loss impacts the end task performance, thus optimizing it should lead to better results. Both VAML and IterVAML lack an in-depth empirical analysis with deep RL models and complex benchmarks.

In the policy gradient domain, Gradient-aware Model-based Policy Search (GAMPS) [11] optimizes its model to minimize an upper bound on the error between the true policy gradient and the model-based estimate. However, GAMPS does not build upon an AC method since it uses its model to approximate action-values via Monte Carlo methods using long virtual rollouts. In model-predictive control (MPC), End-to-end MPC [2] leverages learned models in online planning with receding horizon rollouts. The entire architecture is trained end-to-end, naturally incorporating the end task performance into model learning. However, the method does not learn a parameterized policy and has to replan for every action selection.

3 Background

We consider the agent-environment interaction modeled as a Markov Decision Process [38] $(\mathcal{S}, \mathcal{A}, \mathcal{P}^*, \mathcal{R}^*, \rho, \gamma)$. At every timestep $t \in \mathbb{N}$, the agent observes the current state $\mathbf{s}_t \in \mathcal{S}$ from the set of possible states. It may then select an action $\mathbf{a}_t \in \mathcal{A}$ from the set of possible actions to perform in the environment. The environment then transitions to the next state $\mathbf{s}_{t+1} \sim \mathcal{P}^*(\cdot \,|\, \mathbf{s}_t, \mathbf{a}_t)$, by sampling from the transition probability kernel, and emits a reward signal $r_{t+1} \sim \mathcal{R}^*(\cdot \,|\, \mathbf{s}_t, \mathbf{a}_t)$. We assume the initial state is sampled from some initial state distribution, $\mathbf{s}_0 \sim \rho$.

We overload notation to let $r(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{r \sim \mathcal{R}^*(\mathbf{s}, \mathbf{a})}[r]$ denote the expected immediate reward from $(\mathbf{s}, \mathbf{a})$. The agent's objective is to select actions that produce the highest cumulative discounted rewards, or *return*, from the initial state: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$, where $\gamma \in (0, 1)$ defines the agent's degree of preference for immediate against delayed rewards.

3.1 Actor-Critic methods

Actor-Critic methods approach the RL problem by searching for a *policy* $\pi(\mathbf{a} \,|\, \mathbf{s})$, an action distribution conditioned on states, that induces the highest return. The general approach starts with an initial policy estimate π_0 and then iterates over two main steps: *policy evaluation* and *policy improvement*.

Policy evaluation consists of computing the policy's *action-value function*: the expected return from each state when committing to an initial action, $Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \,|\, \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}]$. One can perform policy evaluation in theory by starting with an initial guess Q_0 and repeatedly applying to it the Bellman operator T^π,

$$(T^\pi Q)(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \, \mathbb{E}_{\mathbf{s}' \sim \mathcal{P}(\mathbf{s}, \mathbf{a})}[V(\mathbf{s}')] \,, \tag{1}$$

where $V(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$. The sequence $(Q_0, Q_1, Q_2, \dots)$ generated by this procedure is guaranteed to converge (in infinity norm) since the operator is a contraction and its unique fixed point is the policy's true value function, Q^π [38].

Policy improvement consists of computing a policy which maximizes the estimated action-value in each state. For the special case of deterministic policies, this step consists of setting $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}\in\mathcal{A}} Q^\pi(\mathbf{s},\mathbf{a})$ everywhere in the state space. This procedure is guaranteed to produce a policy as good as, or better than, the original [37]. In the general case of stochastic policies, policy improvement can be cast as an optimization problem in the space of policies

$$\pi_{k+1} \leftarrow \arg\min_{\pi\in\Pi} \mathop{\mathbb{E}}_{\mathbf{s}\sim\nu}\left[\mathop{\mathbb{E}}_{\mathbf{a}\sim\pi}[Q^{\pi_k}(\mathbf{s},\mathbf{a})]\right], \tag{2}$$

where $\nu(\mathbf{s})$ is some state distribution.

The names "actor" and "critic" come from the intuitive role that each component plays: the policy chooses actions, hence it's the "actor"; the value function evaluates the policy, hence it's known as the "critic".

Actor-Critic methods in deep RL [16] update neural network estimators π_θ and Q_ϕ for actor and critic respectively (where $\theta \in \mathbb{R}^n$ and $\phi \in \mathbb{R}^d$ denote the parameters). In this setting, the evaluation and improvement steps described above are casted as optimization problems w.r.t. the networks' parameters , which are updated via stochastic gradient descent (SGD) [33]. For instance, many methods performance policy evaluation by turning the value update into a regression problem:

$$\phi_{k+1} \leftarrow \arg\min_{\phi} \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a},r,\mathbf{s}'\sim\mathcal{D}}\left[\left(Q_\phi(\mathbf{s},\mathbf{a}) - (r + \gamma Q_\phi(\mathbf{s}', \pi_\theta(\mathbf{s}')))\right)^2\right], \tag{3}$$

where $\mathcal{D}$ is some dataset of collected experiences. Due to the nonlinear nature of neural networks, deep AC methods have no convergence guarantees to optimal policies and value functions. However, the representation capacity and flexibility of neural networks allowed tackling many new problems (e.g., continuous-action domains and learning from pixels) which were previously too challenging for classic methods.

3.2 Value-aware model-learning

Model-based RL adds one more function approximator to be estimated from the available data: a *model* of the transition dynamics, $\hat{\mathcal{P}} \approx \mathcal{P}^*$. In IterVAML [13], the model is used as a proxy for the real dynamics in the Bellman optimality operator

$$(T^*Q)(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}'\sim\mathcal{P}^*(\mathbf{s},\mathbf{a})}[V(\mathbf{s}')], \tag{4}$$

where $V(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$. Akin to the Bellman operator, the optimality operator is also a contraction, however its unique fixed point is the optimal action-value function, $Q^*(\mathbf{s},\mathbf{a}) = \arg\max_\pi Q^\pi(\mathbf{s},\mathbf{a})$. This operator is used in the *value iteration* procedure, which computes the value of the optimal policy directly instead of alternating between policy evaluation and policy improvement. Similar to actor-critic methods, approximate value iteration (AVI) algorithms use parametric estimators and sampling in practice.

Since the model is used for AVI, IterVAML optimizes for the error between T^*Q and $\hat{T}^*Q$,

$$\left\|(\mathcal{P}^* - \hat{\mathcal{P}})V\right\|_{2,\nu}^2 \doteq \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\nu}\left[\left|\int\Big(\mathcal{P}^*(\mathbf{s}' \mid \mathbf{s},\mathbf{a}) - \hat{\mathcal{P}}(\mathbf{s}' \mid \mathbf{s},\mathbf{a})\Big)V(\mathbf{s}')d\mathbf{s}'\right|\right], \tag{5}$$

where ν is some state-action distribution and V is defined as in eq. (4). In contrast, the model loss function used in MLE ignores the structure of the current value estimate:

$$\mathcal{L}_{\text{MLE}}(\mathcal{P}) \doteq \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\nu}\left[\mathop{\mathbb{E}}_{\mathbf{s}'\sim\mathcal{P}^*(\mathbf{s},\mathbf{a})}[-\log\mathcal{P}(\mathbf{s}' \mid \mathbf{s},\mathbf{a})]\right]. \tag{6}$$

The authors of IterVAML argue that minimizing the error in eq. (5) leads to better models for AVI, especially the when the model set $\mathcal{M}$ does not contain the true dynamics, $\mathcal{P}^*$. A shortcoming of this formulation is the difficulty of implementing it in continuous action environments, since the argmax underlying the state-value in eq. (5) is difficult to compute.

Algorithm 1: Value-Aware MBAC

Input: Initial parameters θ_0, ϕ_0, ψ_0 for actor, critic and model respectively; number of iterations K; model interval M

Output: Stochastic policy

$\mathcal{D}^{(0)} \leftarrow \emptyset$

for $k = 0, 1, \ldots, K-1$ **do**

  Generate training set $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1}^n$ by following π_{θ_k} in the environment.

  $\mathcal{D}^{(k+1)} \leftarrow \mathcal{D}^{(k)} \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1}^n$

  $\psi_{k+1} \leftarrow \psi_k$

  if $k \,\%\, M$ **then**

    Find ψ_{k+1} by minimizing eq. (9) w.r.t. model parameters

  Find ϕ_{k+1} by approximate policy evaluation via eq. (7) using $\mathcal{D}^{(k+1)}$ and $\mathcal{P}_{\psi_{k+1}}$

  Find θ_{k+1} by approximate policy improvement via eq. (2) using $Q^{\pi}_{\phi_{k+1}}$ and $\mathcal{D}^{(k+1)}$

return π_{θ_K}

4 Policy iteration with value-aware models

We now present our adaptation of IterVAML as a value-aware model-based AC method.

4.1 Model-augmented value learning

We adopt the *policy evaluation* version of IterVAML as a generic procedure for learning the value function in an Actor-Critic algorithm.[2] Given an initial guess Q_0, the algorithm generates a sequence $(Q_0, Q_1, \ldots)$ by repeatedly applying the update $Q_{k+1} \approx \hat{T}^{\pi} Q_k$. Here, $\hat{T}^{\pi}$ is the same as eq. (1) with the parametric model $\mathcal{P}_{\psi}$ substituted for $\mathcal{P}^*$. Similar to most AC methods, we approximate the theoretical formulation as a regression problem. Effectively, this is a modification of eq. (3) with virtual next-state samples

$$\phi_{k+1} \leftarrow \arg\min_{\phi} \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}} \left[\left(Q_{\phi}(\mathbf{s},\mathbf{a}) - \left(r(\mathbf{s},\mathbf{a}) + \gamma \mathop{\mathbb{E}}_{\mathbf{s}'\sim\mathcal{P}_{\psi}(\mathbf{s},\mathbf{a})} [Q_{\phi}(\mathbf{s}', \pi_{\theta}(\mathbf{s}'))] \right) \right)^2 \right], \tag{7}$$

assuming knowledge of the reward function (which we do in our experiments in the next section).

4.2 Value-aware model optimization

Considering the approximate model-based Policy Evaluation procedure described above, we optimize the model to minimize the difference between $T^{\pi} Q_k$ and $\hat{T}^{\pi} Q_k$, where Q_k is the current iterate of the API algorithm. Thus, we define the *value-aware error* incurred by a model $\mathcal{P}$ as

$$\begin{aligned} c^2_{2,\nu}(\mathcal{P}, V_k) &\doteq \|(\mathcal{P}^* - \mathcal{P}) V_k\|^2_{2,\nu} \\ &= \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\nu} \left[\left| \int (\mathcal{P}^*(\mathbf{s}' \,|\, \mathbf{s},\mathbf{a}) - \mathcal{P}(\mathbf{s}' \,|\, \mathbf{s},\mathbf{a})) V_k(\mathbf{s}') d\mathbf{s}' \right|^2 \right], \end{aligned} \tag{8}$$

where V_k is defined as in eq. (1) and ν is some state-action distribution. This is identical to the cost defined in IterVAML, except that the current policy is used in the computation of V_k.

We assume access to limited samples from the distribution of states and actions and its corresponding next-states (sampled from the transition dynamics $\mathcal{P}^*$) in the form of a replay buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)\}_{i=1}^N$. Thus, we minimize the following cost w.r.t. the model

$$\mathcal{L}_{\text{VAML}}(\mathcal{P}, V) = \frac{1}{N} \sum_{(\mathbf{s}_i,\mathbf{a}_i,\mathbf{s}'_i)\in\mathcal{D}} \left| V(\mathbf{s}'_i) - \mathop{\mathbb{E}}_{\mathbf{s}'\sim\mathcal{P}(\mathbf{s}_i,\mathbf{a}_i)} [V(\mathbf{s}')] \right|^2. \tag{9}$$

[2] See section 3 of IterVAML's supplementary material.

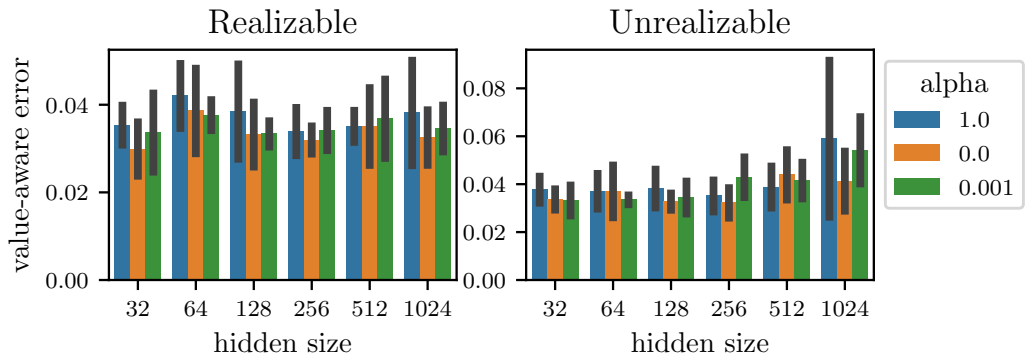

Figure 1: Results on the artificial environment.

4.3 Model-based Actor-Critic

Algorithm 1 summarizes the overall method, which essentially adds a model learning step (lines 5-8) and virtual samples for the critic update (line 9) to an otherwise model-free AC method. Our approach is similar to MBPO [23] with 1-step rollouts, except we don't store virtual experiences in the replay buffer and instead sample fresh ones in every critic update. We use Adam [25] as an SGD optimizer for lines 7, 8, and 9 to find model, critic and actor parameters respectively in each iteration.[3]

Since algorithm 1 is an off-policy method, we adopt the Soft Actor-Critic [19] formulation of API to have stable actor and critic updates and encourage exploration. This amounts to adding policy entropy coefficients to model, actor and critic loss functions (due to the definition of the state-value function in maximum entropy RL), but does not alter the underlying reasoning behind each update.[4]

5 Experiments

Here we perform an empirical analysis of value-aware model learning in the deep RL setting. Throughout all our experiments, we use simple Multilayer Perceptrons (MLPs) for the model parameterization, mapping state and action vectors to the means and variances of a diagonal Gaussian distribution. The distribution is used for sampling an estimated difference between the current state and the predicted next one (similar to residual connections in deep learning). For gradient estimation of the value-aware error defined in eq. (8), we use the pathwise derivative estimator w.r.t. the model's parameters [35]. Code for our experiments is freely available and open-source.[5]

5.1 Unrealizable vs. realizable domains

In this experiment we evaluate one of the underlying premises of VAML: that optimizing for value-aware error will yield better models when there is inherent model approximation error. To test this hypothesis, we took the following steps.

We defined an artificial environment where the transition dynamics are given by a randomly initialized 2-layer MLP (same architecture as described in the previous section). We initialized the model with the same architecture of the dynamics (but different parameters), for the realizable case, and with one less hidden layer for the unrealizable case (so that the model cannot fully capture the real dynamics). We then initialized a random uniform policy and collected two sets of data in the fake environment: one for training (50 thousand transitions) and one for testing (one thousand transitions). For the value function, we randomly initialized an MLP with one hidden layer mapping state vectors to scalar values. We then optimized the model via SGD for several epochs with early stopping on a validation set (taken as a subset of the training subset). Finally, we computed the empirical value-aware error of the learned models in the test set.

[3] Actor and critic updates use only one stochastic gradient step on each invocation of lines 9 and 10.

[4] See section 4.1 of the SAC paper for theoretical foundations and definition of the soft Bellman operator.

[5] A Git repository will be linked upon publication.

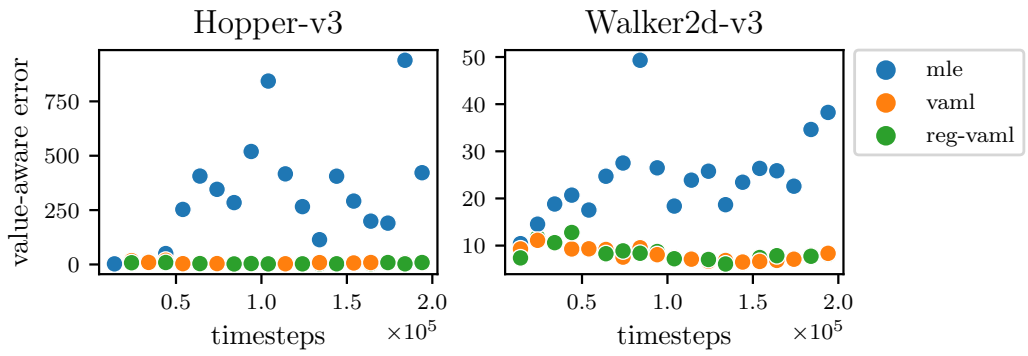

Figure 2: Value-aware error incurred by models trained via different methods in different environments.

In our experiments, we notice that the Gaussian variance learned by value-aware models tends to collapse towards zero. To counteract this, we used a model loss that combines the value-aware and maximum likelihood losses,

$$\mathcal{L}_{\text{MODEL}}(\mathcal{P}_\psi, V) = \alpha \mathcal{L}_{\text{MLE}}(\mathcal{P}_\psi) + (1-\alpha)\mathcal{L}_{\text{VAML}}(\mathcal{P}_\psi, V)\,, \tag{10}$$

where $\alpha \in [0, 1]$. In what follows, we also refer to α in eq. (10) as the MLE coefficient.

Figure 1 shows results across different sizes of the hidden layers in the dynamics MLP, where error bars denote one standard deviation across five repetitions of each experiment and 'alpha' is the MLE coefficient. The realizable setting results are somewhat surprising, with VAML beating MLE by a small margin even though the true dynamics are in the model class. As alluded to before, we notice, upon closer inspection, that the Gaussian variance learned by VAML tends to 0, therefore the slight advantage might be simply due to the value-aware model never sampling values too far from the mean. On the other hand, VAML provides a narrow and inconsistent advantage over MLE, with highly overlapping error bars in most cases. Therefore, we cannot ascertain that VAML is better than MLE in this artificial setting.

5.2 Value-aware error optimization in practice

Our intent in this experiment is to evaluate the feasibility of optimizing the value-aware error in unknown environments, without yet incorporating model-based policy improvements. Thus, we optimize models on data gathered during training runs of the SAC algorithm, without using the model to influence Actor-Critic updates. We consider two environments from the OpenAI Gym [6] toolkit for RL research: Hopper-v3 and Walker2d-v3. We run SAC on each environment for 200 thousand timesteps and save the policy, value function and replay buffer at different timesteps. We use these to check how well we can fit value-aware models at different training phases, since data collection and value function structure change over the course of the algorithm.

Figure 2 compares the value-aware error of models learned via different methods. We add a variation called "regularized VAML" (shortened to "reg-vam" in the plot legends) and used MLE coefficient of 0.001. The results show a trend of increasing value-aware error over the course of the algorithm when the model is trained via MLE. On the other hand, value-aware models keep the error under control throughout training; a promising sign that there's a qualitative difference between models learned via different methods.

5.3 Model-based Actor-Critic with value-awareness

In this experiment we deploy algorithm 1 in two continuous control environments from OpenAI Gym commonly used in deep RL: InvertedPendulum-v2 and Pusher-v2.[6] Our aim was to test if minimizing the model's value-aware error translated into improved performance in the end task. Thus, we fix

[6]We tried the Hopper-v3 and Walker2d-v3 tasks, but due to their higher dimensionality and computational cost, we chose simpler environments instead.

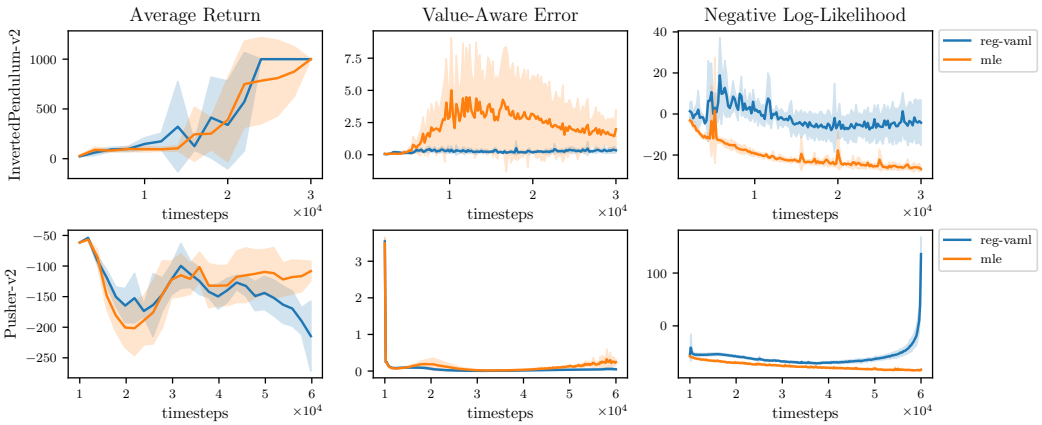

Figure 3: Model-based AC results on InvertedPendulum-v2 and Pusher-v2

all hyperparameters except for the MLE coefficient in the model loss, for which we considered two values: $\alpha = 1$ (model training via MLE) and $\alpha = 0.001$, linearly decaying towards 0 throughout training (regularized VAML). We chose to regularize VAML to keep the Gaussian variance from collapsing to zero, which rendered updates unstable.

Figure 3 shows the results in the InvertedPendulum-v2 (top) and Pusher-v2 (bottom) environments. Lines denote the average across 4 independent runs of the algorithm for each configuration. Shaded regions denote one standard deviation of results. We can see that VAML was able to successfully achieve a lower value-aware error compared to MLE throughout the majority of training. We can also see that the value-aware model is not too far behind in negative log-likelihood. However, the end task performance of the agent using MLE-based model learning is as good as (if not better, as in the case of Pusher-v2) the performance of the agent using VAML. Failing to establish a significant difference between the two model-learning methods in these simple environments, we cannot ascertain that VAML provides an advantage over conventional likelihood-based methods in a MBAC framework.

6 Conclusions and future work

In this work we adapted the ideas of VAML to the Actor-Critic framework. We showed that, despite promising theoretical results and preliminary experiments, VAML does not translate well to the deep RL setting, with conventional MLE-based approaches yielding comparable, or better, end task performance.

Despite these negative results, there are interesting research directions to further our understanding of the underlying issues with VAML in its current formulation. One concern with the approach in algorithm 1 is that it is too unstable as a model-based method (we observed that its performance is sometimes worse than SAC). Stabilizing this base algorithm by conservatively updating the policy or model [31], or incorporating epistemic uncertainty in model learning with randomized prior functions, [28] may provide a more solid foundation upon which to compare VAML and MLE-based approaches.

Our results call for a better understanding of the underlying assumptions underlying IterVAML.[7] The paper's theoretical analysis assumes data collected in each iteration is independent of all the others, which is clearly not the case in an approximate policy iteration setting. Moreover the analysis also assumes the value function is independent of the data and bounded. We also suspect that the interplay between model and critic may destabilize training (in an already unstable actor-critic framework). We suggest that the original VAML formulation [14] may be more promising since it decouples the value used in learning the model from the one used for policy improvement. Moreover, the VAML loss function may be amenable to optimization via adversarial methods [4] and has a connection to the Wasserstein distance between model and true dynamics [5].

[7] See Assumptions 1 and 3 of the IterVAML work.

References

[1] Romina Abachi, Mohammad Ghavamzadeh, and Amir-massoud Farahmand. Policy-aware model learning for policy gradient methods. *CoRR*, abs/2003.00030, 2020.

[2] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable MPC for End-to-end Planning and Control. In *NeurIPS*, pages 8299–8310, 2018.

[3] Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. On the model-based stochastic value gradient for continuous reinforcement learning. *CoRR*, abs/2008.1, 2020.

[4] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *CoRR*, abs/1701.0, 2017.

[5] Kavosh Asadi, Evan Cater, Dipendra Misra, and Michael L Littman. Equivalence Between Wasserstein and Value-Aware Model-based Reinforcement Learning. *CoRR*, abs/1806.0, 2018.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[7] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. In *NeurIPS*, pages 8234–8244, 2018.

[8] Ignasi Clavera, Yao Fu, and Pieter Abbeel. Model-Augmented Actor-Critic: Backpropagating through Paths. In *ICLR*. OpenReview.net, 2020.

[9] Kefan Dong, Yuping Luo, and Tengyu Ma. On the expressivity of neural networks for deep reinforcement learning, 2020.

[10] Pierluca D'Oro and Wojciech Jaskowski. How to Learn a Useful Critic? Model-based Action-Gradient-Estimator Policy Optimization. *CoRR*, abs/2004.1, 2020.

[11] Pierluca D'Oro, Alberto Maria Metelli, Andrea Tirinzoni, Matteo Papini, and Marcello Restelli. Gradient-Aware Model-Based Policy Search. In *AAAI*, pages 3801–3808. AAAI Press, 2020.

[12] WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[13] Amir-massoud Farahmand. Iterative Value-Aware Model Learning. In *NeurIPS*, pages 9090–9101, 2018. URL `http://papers.nips.cc/paper/8121-iterative-value-aware-model-learning`.

[14] Amir Massoud Farahmand, André Barreto, and Daniel Nikovski. Value-Aware Loss Function for Model-based Reinforcement Learning. In *AISTATS*, volume 54 of *Proceedings of Machine Learning Research*, pages 1486–1494. PMLR, 2017. URL `http://proceedings.mlr.press/v54/farahmand17a.html`.

[15] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. *CoRR*, abs/1803.00101, 2018.

[16] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018. URL `https://doi.org/10.1561/2200000071`.

[17] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR, 2018. URL `http://proceedings.mlr.press/v80/fujimoto18a.html`.

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.

[19] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Jennifer G Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

[20] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *CoRR*, abs/1812.05905, 2018. URL `http://arxiv.org/abs/1812.05905`.

[21] Danijar Hafner, Timothy P Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *ICLR*. OpenReview.net, 2020.

[22] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.

[23] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to Trust Your Model: Model-Based Policy Optimization. In *NeurIPS*, pages 12498–12509, 2019.

[24] Joshua Mason Joseph, Alborz Geramifard, John W Roberts, Jonathan P How, and Nicholas Roy. Reinforcement learning with misspecified model classes. In *ICRA*, pages 939–946. IEEE, 2013.

[25] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*, 2015.

[26] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1509.02971`.

[27] Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. Model-based Reinforcement Learning: A Survey. *Proceedings of the International Conference on Electronic Business (ICEB)*, 2018-Decem:421–429, 6 2020. URL `http://arxiv.org/abs/2006.16712`.

[28] Ian Osband, John Aslanides, and Albin Cassirer. Randomized Prior Functions for Deep Reinforcement Learning. In *NeurIPS*, pages 8626–8638, 2018.

[29] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time Limits in Reinforcement Learning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 4042–4051. PMLR, 2018.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[31] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A Game Theoretic Framework for Model Based Reinforcement Learning. *CoRR*, abs/2004.07804, 2020.

[32] Benjamin Recht. A Tour of Reinforcement Learning: The View from Continuous Control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):253–279, 5 2019. ISSN 2573-5144. doi: 10.1146/annurev-control-053018-023825. URL `http://arxiv.org/abs/1806.09460`.

[33] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[34] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014.

[35] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *NIPS*, pages 3528–3536, 2015.

[36] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, 1991.

[37] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[38] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010. doi: 10.2200/S00268ED1V01Y201005AIM009. URL `https://doi.org/10.2200/S00268ED1V01Y201005AIM009`.

[39] Istvan Szita and Csaba Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *ICML*, pages 1031–1038. Omnipress, 2010.

[40] Volodymyr Mnih. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1928–1937, 6 2013. ISBN 9781510829008. doi: 10.1177/0956797613514093. URL `http://proceedings.mlr.press/v48/mniha16.html`.

[41] Che Wang, Yanqiu Wu, Quan Vuong, and Keith W Ross. Towards Simplicity in Deep Reinforcement Learning: Streamlined Off-Policy Learning. *CoRR*, abs/1910.0, 2019.

A Algorithm hyperparameters

Hyperparameter	Value
n (line 3)	1
M	200
\SGD{} optimizer	Adam
actor/critic/entropy learning rate	$3 \cdot 10^{-4}$
model learning rate	10^{-4}
model L2 penalty	10^{-5}
initial entropy coeff	1.0
target entropy	$-dim(\mathcal{A})$
γ	0.99
polyak factor	0.995
replay size	1e5
replay batch size	256
actor/critic hidden layers	[256, 256]
actor/critic nonlinearity	ReLU
actor max logvar	2
actor min logvar	-20
model hidden layers	[128, 128]
model nonlinearity	Swish
model max logvar	2
model min logvar	-20

Table 1: Algorithm 1 hyperparameters

Table 1 lists the hyperparameters used in algorithm 1, our model-based augmentation of SAC with value-aware models.

We use Glorot initialization [18] for all networks with defaults parameters via the PyTorch machine learning framework [30].

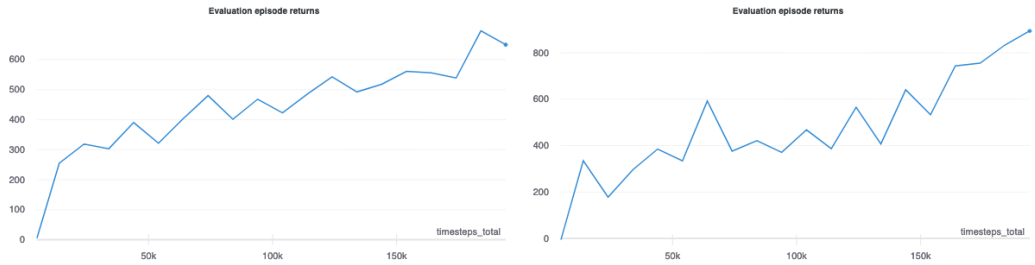

Figure 4: SAC learning curves on Hopper-v3 (left) and Walker2d-v3 (right) of the runs used in the second experiment.

Due to limited availability of computational resources, we use a single model network instead of an ensemble, which is a trend in recent model-based RL methods.

As mentioned in section 4.3, algorithm 1 uses only one gradient step on each call to lines 8 and 9 to update critic and actor respectively.

We use clipped double Q-learning [17] with target networks, updated after every critic update (line 8) with polyak averaging of the parameters. In eq. (7), we sample 10 future states from the model and average the clipped Q-value to approximate the inner expectation.

We use automatic temperature tuning for the entropy coefficient as described in the original paper [20]. We take one dual gradient step on the entropy coefficient's loss function right after the actor update in line 9 of algorithm 1.

We collect 10000 initial transitions by choosing from the action space uniformly at random. Only after this initial phase do we start acting according to the current policy.

We evaluate the current policy every 2000 timesteps by turning off exploration (sampling from the mean of the learned Gaussian policy) and computing the average return over 10 episodes.

In terminal transitions due to timeouts, we ignore the termination flag and instead bootstrap from the final state [29].

We use the Pytorch Lightning [12] framework for training the model. On every call to line 7 of algorithm 1, we subsample 20% of the current replay buffer as a holdout set, with a maximum of 5000 transitions. After the initial 10000 random uniform transitions, we train the model for a maximum of 1000 epochs. Otherwise, over the normal course of the algorithm, we train the model for a maximum of 20 epochs or 200 gradient steps. We calculate the validation loss on the holdout set at the end of every epoch. We early stop training if the validation loss hasn't improved over 5 consecutive epochs, where improvement is characterized as any decrease in loss.

B Supervised learning experiments details

Experiments in the artifical domain. We use the dynamics architecture described in section 5, a state space $\mathcal{S} \subseteq \mathbb{R}^{11}$, an action space $\mathcal{A} \subseteq [-1, 1]^3$, and a horizon of 200 timesteps. The state and action spaces were chosen to mimic the spaces of the Hopper-v3 task from OpenAI Gym.

We collect 50000 transitions for training (with 20% of them held out for validation) and 1000 transitions for testing. We use a batch size of 128 for training over a maximum of 1000 epochs. We early stop training if the validation loss (calculated at the end of every epoch) has not decreased for 3 consecutive epochs.

The value function architecture consists of a single-layer MLP with 64 hidden units and ReLU nonlinearities. We use orthogonal initialization [34] with default parameters via PyTorch.

Experiments on SAC data. Figure 4 shows the episode returns attained by the SAC agent from which we gathered data for the second experiment.

We take actor, critic, and replay data from different checkpoints of each SAC run to train the model with. The model architecture is as described in section 5, with each hidden layer having 32 units. We

split the replay data into 70% for training, 20% for validation, and 10% for testing. For each model training run, we use the Adam with learning rate 10^{-3} to optimize the model for a maximum of 1000 epochs, early stopping training if the validation loss, calculated once every epoch, does not decay for 10 consecutive epochs. We report in fig. 2 the test loss at the end of each training run.