A Supervised Formulation of Reinforcement Learning: with super linear convergence properties

Amit Parag, Nicolas Mansard

A supervised formulation of Reinforcement Learning: with superlinear convergence properties

Amit Parag[1,2,*], Nicolas Mansard[1,2]

***Abstract*— Deep reinforcement learning uses simulators as abstract oracles to interact with the environment. In continuous domains of multi body robotic systems, differentiable simulators have recently been proposed but are yet under utilized, even though we have the knowledge to make them produce richer information. This problem when juxtaposed with the usually high computational cost of exploration-exploitation in high dimensional state space can quickly render reinforcement learning algorithms impractical. In this paper, we propose to combine learning and simulators such that the quality of both increases, while the need to exhaustively search the state space decreases. We propose to learn value function and state, control trajectories through the locally optimal runs of model based trajectory optimizer. The learned value function, along with an estimate of optimal state and control policies, is subsequently used in the trajectory optimizer : the value function estimate serves as a proxy for shortening the preview horizon, while the state and control approximations serve as a guide in policy search for our trajectory optimizer. The proposed approach demonstrates a better symbiotic relation, with super linear convergence, between learning and simulators, that we need for end-to-end learning of complex poly articulated systems.**

I. Introduction

Reinforcement Learning (RL) [1] sets its goal as the search for an optimal policy to navigate its immediate environment. It does so by establishing its interactions with the environment as a Markov decision process where immediate action taken in the current state is driven to maximize an expected reward over a foreseeable future. In turn, it relies on an oracle to provide it with states, actions and expected rewards, with implicit assumptions on the overall efficiency of the oracle. In high dimensional robotic systems, the oracle itself is a differentiable simulator that can plan over long horizons. While an end to end RL framework can, in principle, be applied to robot learning, practical applications of RL on real world has been less successful.

Simulators, even though can handle non linearities, complex dynamic behaviour and constraints, can become severely limited by the corresponding computation time, in particular shooting methods like Differential Dynamic Programming (DDP) can require a large number of iterations to converge or fail to converge entirely. This is relatable to the application of RL solvers in continuous domain, where training can become prohibitively expensive and is strongly dependent on the sample efficiency of its exploration strategy.

We argue that in the complex domain of poly articulated systems, combinations of *model based* and *model free* methods are more suited to explore solutions of a Markov Decision Process (MDP). The overall goal is to find optimal solution of a MDP and in the general landscape of robot learning, this can be done such that either *model based* help *model free* methods find that solution or the opposite. This point is important. In our work, we attempt to combine learning and trajectory optimization in a manner such that the overall combination finds the optimal solution.

A. Our Contributions

We propose an actor-critic *esque* coupling as a solution to optimal control problem (ocp) that combines a trajectory optimizer (TO) with a reinforced learning loop: the reinforced learning loop itself is the actor, while the role of critic is played by the trajectory optimizer. We achieve this by setting an iterative loop in the backdrop of recursivity provided by Bellman's optimality principle [2], [3] such that learning depends on data provided by TO, while efficient computation of optimal trajectories over a preview horizon by the trajectory optimizer depends on accurate learning.

This synergistic coupling can alternately be viewed as a game between two players where the optimal outcome, i.e the optimal solution to the Markov Decision Problem, is contingent upon the strategy chosen by each player to be optimal, where the strategy of each player depends on the strategy of the other player. The purpose of utilizing Bellman's optimality principle is to ensure that the strategy of both players remain guided by optimality.

We explicitly focus on learning with high accuracy and with reduced rollouts. We use Differential Dynamic Programming (DDP) [4], [5], a particular class of (direct shooting) trajectory optimizer, to give us state-value pairs and state-control trajectories which we learn in the supervised phase using three neural networks. The estimates of value function is subsequently used inside DDP as an anchor at the terminal position, while the approximations of optimal policies serve as a guide for DDP. We iterate over this process until *convergence*.

As DDP also requires the 1^{st} and 2^{nd} order derivatives of the value function, we explicitly use gradients during training using a Sobolev Loss. 1^{st} order supervised training also has the benefit of reducing the dependence of learning on hyper parameters.

B. Related Work

The algorithmic formulation of RL often requires a large number of samples [6] followed by considerable tuning. To mitigate these problems, in [7] an extension to soft actor

[1]Artificial and Natural Intelligence Toulouse Institute, France
[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
*corresponding author: aparag@laas.fr

critic approach [8] was shown to counter over sensitivity to hyper-parameters [9]. The quality of predictions of the actor itself, was examined in [10] with the conclusion that the actor learns better when learning to act optimally over a horizon rather than learning the next optimal state.

Noting that value functions are unique fixed points of *Bellman operators* of the corresponding Markov decision process and govern interactions of RL agent with its environment, an extensive analysis in [11] showed that the mathematical foundations of RL are fully not realised in the algorithmic implementation of policy gradient methods [12] : value function estimates never match the true value function and only marginally guide the search for policy. The mathematical foundation is the minimization of some *stochastic objective* function based on the governing dynamics of the system and RL usually proceeds by estimating the 0^{th} order gradient of that objective [1], [13]. This is where differentiable simulators differ from RL and other derivative free methods [14], [15]. In robotic systems it is possible to compute either analytic [16] or approximate derivatives through automatic differentiation [17], [18].

However, planning over forecasts of process behaviour over long horizons by minimizing a cost function or learning optimal policies by maximizing a reward function are , in essence, complementary approaches to the underlying optimal control (Markov decision process) problem. This duality has spawned numerous approaches to combine them more rigorously, typically with the objective of making one benefit from the other. To overcome lack of consistency of RL, in [19] suitable guiding samples drawn from a trajectory optimizer are incorporated to assist direct policy search in high dimensional system. In [20], the sub optimal trajectories are refined using a trained policy which are then used as guiding samples. In [21] trajectory optimizers were used in exploration of the state space to minimize the risks associated by the RL agent acting *greedily*.

On the other hand, various approaches have tried to use learning to improve trajectory optimizers. This typically involves learning some quantity to improve the performance of trajectory optimizers. The learned quantity can be a dedicated dynamic model of the system [22] or a cost model for the task as in [23]. Much more closer to our work are [24], [25], where the authors learn value function : the learned value function is subsequently used inside trajectory optimizer either to serve as a stable anchor for the terminal position or for one step model predictive control.

In our earlier work in learning value functions [26], we build upon the idea of learning value function at the terminal state with high fidelity by coupling a trajectory optimizer with a *sobolev supervised* learning phase. Using value function as a terminal proxy rather than a running cost proxy also minimizes the risk associated with function approximation. Learning in Sobolev spaces, [27], [28], differs from classical regression in that it constrains learning to simultaneously match target derivatives with derivatives of the deep neural network while minimizing the error between predictions and target. This form of constrained learning has been shown to mitigate the additional cost of computing gradients of the neural network with respect to input, by being more data efficient and robust [29]–[31]. Supervised Sobolev training also opens the idea of learning control trajectories by forcing the learning agent to match the Riccati gains : these gradients (analytic or inexact) of control are provided by differentiable simulators.

II. Recursive optimality

In this section, we give a brief summary of the different foundations of our algorithm and establish notations. We choose to use the optimal control formulation of MDP.

A. Problem formulation and notations

We consider a time discrete formulation of (finite state) MDP for a system with autonomous dynamics in some environment. The evolution of such a system can then be written as

$$x_+ = f(x, u, \Omega) \tag{1}$$

where x_+, x are the next state and the current state respectively in n dimensional vector space such that $x \in \mathcal{X}$. f represents the time independent state transition function. Ω is a parametrization of the environment where the system evolves and $u \in \mathcal{U}$ denotes the controls applied to the system. For brevity, we will drop Ω from here on.

The solution to the corresponding optimal control problem would then be to find a pair $X : t \to x(t) \in \mathcal{X}$ and $U : t \to u(t) \in \mathcal{U}$ which minimizes a cost functional $L(X, U)$. In a discretized transcription, $L(X, U)$ can be written as an infinite sum of running cost, $l(x, u)$:

$$L(X, U) = \sum_{k=0}^{+\infty} l(x_k, u_k) \tag{2}$$

For finite time horizon problems of length T, the $L(X, U)$ is split in two parts:

$$L(X, U) = \sum_{k=0}^{T-1} l(x_k, u_k) + l^f(x_T) \tag{3}$$

where $l^f(x_T)$ is the cost at the terminal position. We denote X^*, U^* as the optimal solution to this minimization problem over a finite time horizon T and from the initial starting state x_0.

B. Value function

Assuming that the solution pairs to Eq 3 are optimal, then we can define a *value function*, $V : x \to V(x)$, as the optimal value of the cost functional when the system at the starting state x_0 moves along the optimal trajectory, $x \in X^*$, while following an optimal policy $u : x \to u(x) \in U^*$.

We can further define V_k as the cost-to-go over the horizon $T - k$ from any starting state x by rewriting Eq. 3 as:

$$V_{T-k}(x) = \min_U \sum_{j=k}^{T-1} l(x_j, u_j) + l^f(x_T) \tag{4}$$

If l^f can be replaced with value function, then the cost-to-go over the horizon $T-k$ can be reformulated to obtain recursive optimality. In that case, the minimization problem becomes:

$$V_{T-k}(x) = \min_u l(x,u) + V_{T-k-1}(f(x,u)) \tag{5}$$

If a fair estimate of the value function can be provided, them this problem transforms into an infinite horizon problem while remaining solvable with finite resources. Furthermore, the cost-to-go is now independent of timestep and is equal to value function: $V_T(x) = V(x) \quad \forall T > 0$.

C. Differential Dynamic Programming

DDP is a second order iterative algorithm [4], [32] that takes advantage of the recursivity of Bellman's Optimality Principle by adding the boundary condition, $V_T(x) = l^f(x_T)$ to Eq. 5. In each iteration, it numerically solves the optimal control problem described above by performing a backward and a forward pass on the current estimate of the state-control trajectories : a backward phase to estimate the value function as quadratic fit along the current candidate trajectory, a forward phase to refine the candidate trajectory based on the value function.

To construct a quadratic fit of the value function, DDP measures the deviations from the current candidate trajectory through Taylor's expansion [33], discards terms beyond second-order. It then returns an estimation of the cost-to-go and the hessian and gradient at every step along the preview horizon.

This implies that for the neural network that approximates the value function to be substituted as a proxy for $l_f(x_T)$, it has to be sufficiently accurate and twice differentiable, since, in the backward pass, DDP requires a estimate of value function along with its first and second order derivatives.

III. Differential Policy Value Programming

a) Algorithmic Principles: Eq 5 immediately shows that the global value function, V^*, can be approximated through the locally optimal rollouts of DDP if V is known. With this in view, we propose Differential Policy Value Programming, ∂PVP, that iteratively estimates the global time independent value function and state-control trajectories.

We use three neural networks : a residual network, V_α^n, to approximate the value function through Gauss Newton decomposition, a state network, X_β^n, to learn the state trajectory, and a final control network, U_γ^n, to estimate the control trajectory[1]. α, β, γ are the parameters of the respective neural networks at the n^{th} iteration.

At the beginning of ∂PVP, we generate a batch of trajectories for a predefined horizon length T, but without any terminal cost model, i.e $l^f = 0$. We use this for offline training of $V_\alpha^{n=1}$, $X_\beta^{n=1}$, $U_\gamma^{n=1}$. At the end of the first iteration, $V_\alpha^{n=1}$ approximates the cost-to-go for a horizon of length T, $X_\beta^{n=1}$ approximates the state trajectory and $U_\gamma^{n=1}$ approximates the control to be applied at each step along the preview horizon.

b) Iterative Episodic Learning: ∂PVP then proceeds to iteratively build upon its estimates of the value function, state and control trajectories. In subsequent iterations V_α^n acts as the proxy for terminal cost function. So at the end of every iteration, Eq. 4 is changed to:

$$V^n(x) = \min_u \sum_{k=0}^{T-1} l(x_k, u_k) + V_\alpha^{n-1}(x_T) \tag{6}$$

where n is the iteration number, $n \geq 2$, V_α^{n-1} is the value function approximated in the previous iteration, while X_β^n, U_γ^n are used online in DDP to provide candidate trajectories in the forward pass.

With each iteration n, the learning should approximate the cost-to-go, state-control trajectories over a horizon $(n+1).T$. This in turn should provide a stable anchor for DDP at the terminal position, and drive it toward steady state solutions. Simultaneously, X_β^n, U_γ^n provide precise enough guesses of the steady state solutions.

By setting a reinforced loop between V_α^n, X_β^n, U_γ^n and DDP, the convergence of the algorithm should depend on $(n+1)T$. Therefore as n increases, V_α^n, X_β^n, U_γ^n should tend toward global V^*, X^*, U^*. and the algorithm should achieve super linear convergence in the number of attempts required to find optimal trajectories.

IV. Supervised Training Procedure

A. Architecture of Trainable Models

V_α, X_β, U_γ were initially implemented as the outcome of a three headed feed forward network with common hidden layers. This was specifically done to test the trade-off between the training time of a multi headed network and the theoretical advantages of enabling the multiple heads to benefit from the rich information encoded in the common hidden layers. However, this resulted in bloated computation time of parameter updates during the training phase. The practical benefits of shorter training time far outweighed the theoretical advantages of common hidden layers. We do not include those results in this paper. For our experiments, we decided to learn the three different quantities - value function, state trajectory and control trajectory - on three different feed forward networks.

a) State-Control Approximators : X_β and U_γ are implemented with simple deep feed forward networks with 6 hidden layers with ReLU, ELU, Tanh alternately applied . The output and input shape depends on the dimensions of the optimal control problem. These two approximators are trained in the classical supervised manner as opposed to Sobolev Regression for learning the value function described next.

b) Value Function Approximator : DDP as mentioned in Sec II-C, is a second order algorithm. It computes the 2^{nd} and 1^{st} order derivatives of value function in the backward pass and therefore requires $V_\alpha^{'}$ and $V_\alpha^{''}$ in every iteration of our algorithm. Since computing $V_\alpha^{''}$ is not feasible, especially

[1]see Sec VI for use of Riccati Gains during training

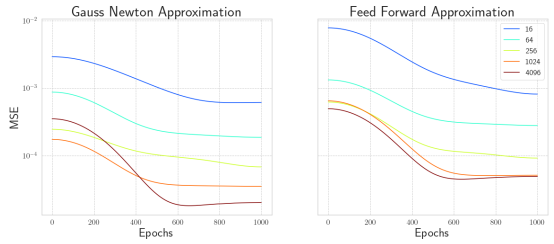

Fig. 1: Illustration of the relation between accuracy during training and size of dataset. The different colors of training curves represent different size of datasets.

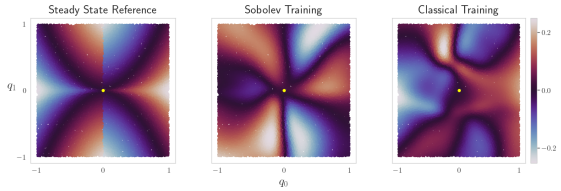

Fig. 2: Quality of gradients of V_α, with and without Sobolev Training.

in higher dimensional systems like manipulator arm, we use Gauss Newton approximation to design V_α as squared sum of residuals:

$$V(x|\alpha) = R(x|\alpha)^2 \tag{7}$$

This immediately allows us to write the 1^{st} and 2^{nd} order derivatives as :

$$V'(x|\alpha) = 2R'(x|\alpha)^T R(x|\alpha) \tag{8}$$

$$V''(x|\alpha) \approx 2R'(x|\alpha)^T R'(x|\alpha) \tag{9}$$

We implement this with a feed forward network with three hidden layers and tanh activation. The final layer outputs a three-vector residual. Modeling the value function as the output of Gauss Newton Approximation seemingly imparts a more physical interpretation to the hidden layers as compared to a simple feedforward network. We also find that $R(x|\alpha)^2$ performs better during training as compared to a one output feedforward network as shown in Fig 1.

c) Sobolev Regression for V_α : The trainable parameters of V_α during the learning phase are consequently changed in response to cumulative errors accrued by two losses - 0^{th} and 1^{st} order. The 0^{th} order loss is identical to canonical losses used in functional regression that penalizes the difference in observations and target with some norm λ_d, while the 1^{st} order loss forces the neural network to match its derivatives (w.r.t input) with the corresponding derivatives of the target. The corresponding difference between only 0^{th} order training and Sobolev training in seen quite clearly in Fig 2. Theoretically Sobolev regression can involve higher order derivatives, however in our experiments we choose to stay at 1^{st} order with norm $d = 2$ to offset the increased workload on the automatic differentiation engine.

V. Empirical Evaluations

We benchmark[2] our algorithm on three classical control problems: unicycle [34], cartpole [35], [36] and inverted pendulum [36], along with a torque-controlled 7 degrees of freedom (*dof*) manipulator arm[3] where the ocp is formulated as static end-effector pose reaching task, controlled in torque with additional regularization on both state and torque controls.

Since the precision of our algorithm increases with $(n+1).T$, the quality of V_α^n should depend on the n^{th} iteration or on T or both. We establish validation dataset for simple classical control systems by sampling for a large collection of locally long optimal trajectories of horizon length 1000. For the 7 *dof* manipulator, sampling for a similar validation dataset is not feasible.

In this section[4], we present certain results of interest.

A. Estimates of Quasi Steady State Value Function : V_α^n

a) Quality of predictions of V_α : To compare the differences in the predictions of the value function, we compute a *quasi-infinite* horizon value function validation dataset by sampling for locally long optimal horizons of length 1000. We use this dataset, $V(s, T = 1000)$, to establish distance to infinite horizon in value learning for systems like unicycle.

As we can see in Fig 3, the algorithm quickly learns the overall topology of value function across state space. As the number of iterations increase, ∂PVP seems to refine its understanding of the inherent symmetry in value function topology. Fig 4 shows the difference between quasi infinite horizon value function and predicted value functions at iterations 1, 5, 9. We observe that the iterative aspect of ∂PVP allows it to learn value function over long horizons quite well despite initialization in short horizon. However, there seems to be regions in configuration space difficult to handle (shown in white in Fig 4). We suspect that the non-holonomic constraints in the unicycle environment leads to singularities which in turn destabilize learning. With more learning, the algorithm overcomes the presence of singularities. By the 10^{th} iteration, the algorithm had narrowed the location of singularities to be symmetrically distributed around the goal, coincident with the q_1 axis in Fig 4

Fig 5 and Fig 6 shows the smoothness of predictions of $V_\alpha^{n=10}$ for the end-effector pose reaching task with the 7 *dof* manipulator arm.

B. Quality of Warmstarts : X_β, U_γ

Fig 7, Fig 8, Fig 9 and Fig 10 show the predicted state trajectories and control trajectories for the EE pose reaching task and the classical control problems with comparisons with a corresponding locally optimal trajectory computed by DDP. The advantage of learning state-control trajectories in a supervised setting can be seen in the smoothness of

[2]The source code is available at `https://gitlab.laas.fr/aparag/kuka-arm-dpvp`

[3]Kuka lwr iiwa R820 14

[4]The experiments were performed on core i9 processor with 32 Gb RAM

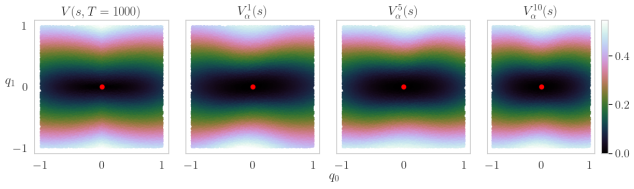

Fig. 3: Comparison of the topology of predicted value functions by $V_\alpha^1, V_\alpha^5, V_\alpha^{10}$ with value function at horizon length 1000: $V(s, T = 1000)$. The target position is shown in red. The unicycle can start anywhere in the configuration space and tries to reach for target

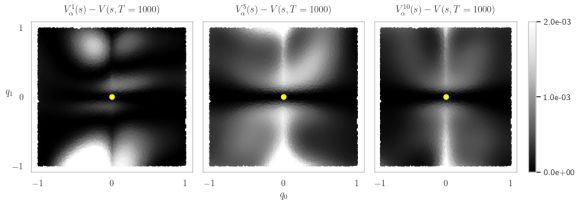

Fig. 4: Scatter plot of the distribution of mean squared errors between $V_\alpha^1, V_\alpha^5, V_\alpha^{10}$ and $V(s, T = 1000)$ for unicycle. The yellow dot is the goal position for the unicycle.

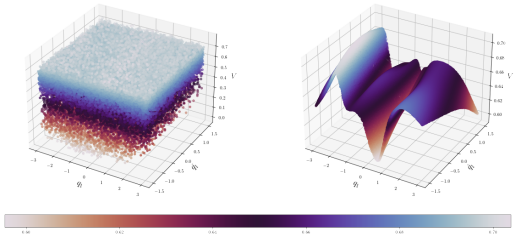

Fig. 5: The Figure on the left shows the predictions for value functions of $V_\alpha'^{n=10}$ for the 14 dimensional state space of the manipulator arm. The points were randomly sampled from 14 dimensional space. In the Figure on the right, only $q_1, \dot{q_1}$ were randomly sampled and the other 12 dimensions were set to 0.

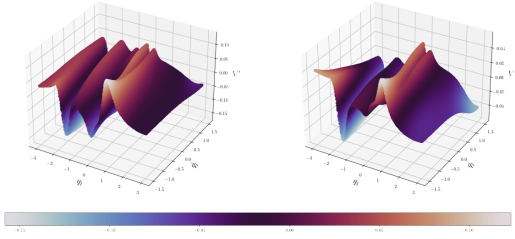

Fig. 6: Slices of the gradients of the neural network, i.e $V_\alpha'^{n=10}$ for the manipulator state space

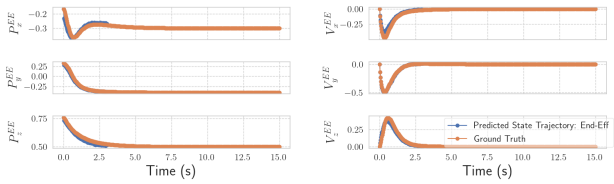

Fig. 7: Predictions of X_β for EE pose reaching task. During the training phase, 150 locally optimal samples of horizon length 200 were drawn the 14 dimensional configuration space in each of the 20 iterations. This resulted in our TO computing 10347 rollouts overall. The training phase lasted 61 minutes.

the predictions. The predicted state trajectories are nearly coincident with the corresponding long optimal trajectories. When these learned trajectories are used as a reference to guide policy search in DDP, the number of rollouts/attempts required by DDP reduces, as opposed to the number of rollouts required by DDP when not provided any intelligent guesses.

C. Super Linear Convergence

In Fig 11 and Fig 12, we compare the number of rollouts required DDP and ∂PVP to solve a problem for the 7 *dof* manipulator arm and unicycle respectively.

We use our trained networks inside our trajectory optimizer to solve for 500 uniformly sampled initial configurations from the configuration space. For those very same initial configurations, we also use trajectory optimizer but without any help from the trained networks. We then compare the number of rollouts needed in both cases. We see that with ∂PVP the number of attempts made to find optimal policies decrease drastically. For the manipulator arm the average rollouts required by ∂PVP is 2.71, while the average rollouts required by DDP is 9.3.

In Fig 13, we show the difference in solutions computed by DDP and ∂PVP at different horizons. Trajectories computed with ∂PVP at short preview horizons seem coincident with trajectories computed over longer horizon.

VI. Discussion and Conclusion

In this work, we showed that reformulating reinforcement learning as a combination of iterative supervised learning phase, with emphasis on value functions, and simulators allows for reduction in trials needed to find the (locally) optimal solution. The iterative supervised learning phase enforces the stability of predictions through Sobolev regression while learning in the backdrop of recursive optimality further reduces the dependence on hyper parameters.

There are a few points of interest in the way information from policy trials are used in the supervised learning phase. First, in the learning phase TO plays the role of an oracle to provide us for a predefined horizon for each ocp, the state trajectory, the control trajectory, the Riccati Gains, time dependent value function estimates of every node in the state trajectory (along with the corresponding gradients and hessians).

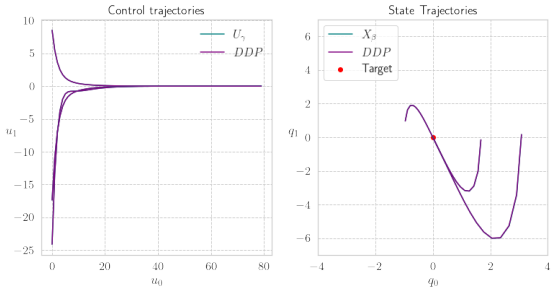

Fig. 8: Predictions of X_β and U_γ for Pendulum compared to the corresponding solutions by DDP.

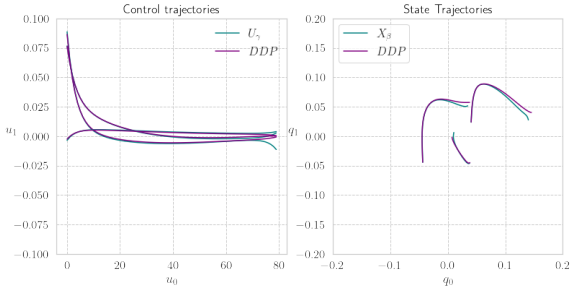

Fig. 9: Predictions of X_β and U_γ for Cartpole. vs DDP

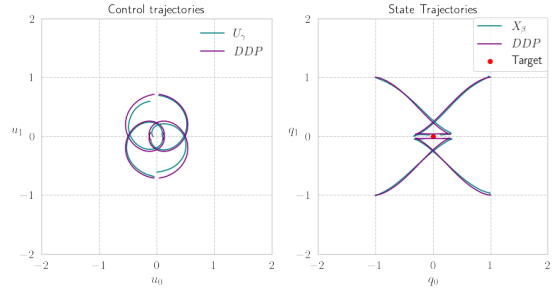

Fig. 10: Warmstarts provided by X_β and U_γ for Unicycle compared to the final solutions computed by DDP.

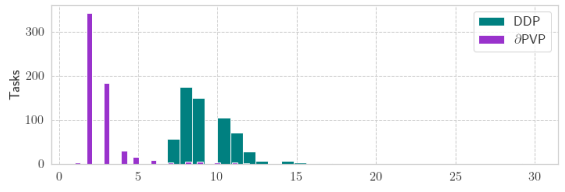

Fig. 11: Histogram of rollouts required for the 7 *dof* manipulator arm with DDP and ∂PVP across 500 tasks.

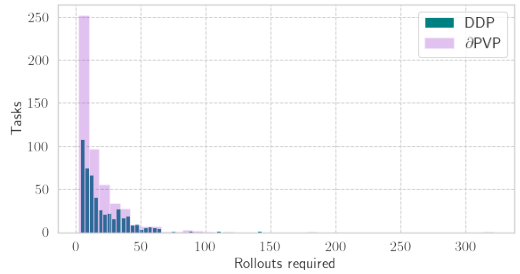

Fig. 12: Histogram of rollouts required for unicycle with DDP and ∂PVP

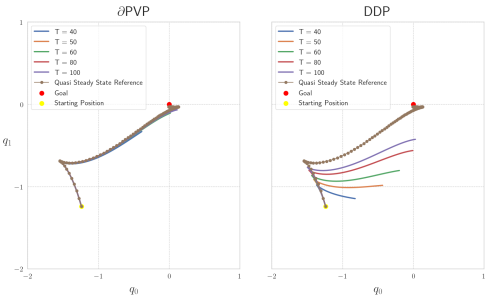

Fig. 13: State trajectories computed by DDP vs ∂PVP at different horizons for the unicycle problem.

Of this information, we choose to keep the node with the highest preview horizon for learning value function. This can be seen as an under utilization of the state-value information. On the one hand, states with value function estimations of smaller preview horizon can augment the dataset but on the other hand, this approach can lead to a noisier dataset which can effectively slow down convergence.

To enforce a shorter training time, we also discarded the use of hessians of value function during the training of V_α. This consideration also precluded the use of Riccati gains during training of U_γ.

Similarly we tested the feasibility of adding a kernel loss function based on Bellman' contraction operator on the learning of state and control trajectories. A kernel loss function that constrains trajectories to satisfy the Hamilton-Jacobi-Bellman criteria of optimal sub-structures: sub-solutions of an indefinite horizon optimal control problem should also be optimal solutions to the corresponding definite horizon sub problems, should in theory, elevate the quality of the predicted state and control trajectories. However implementing such a kernel loss function seemed to have little benefits at the expense of increased computation time. We leave it to a future work to explore these avenues.

The training time can be further reduced with the observation that the policies learned by the neural networks serve as a guide for DDP. In our experiments, we observe that the precision of X_β, U_γ increases only slightly as ∂PVP iterates. This allows us flexibility in defining the number of training epochs in every iteration, which we need if the dimensionality of the problem under consideration increases. For experiments with manipulator arm, we choose to learn and improve state-control trajectories only in the initial and final iterations. We found this training strategy, where estimations of value functions are refined at every iteration, whereas estimation of state-control trajectories are refined only at start and end to be good enough for our purposes.

In this paper, we presented the foundational aspects of our algorithm. We believe that coupling approach presented in this paper is mature enough to be tested on legged robots such as quadrupeds and mini cheetah in real time. We plan to present those results in a future work.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, 1966.

[3] S. Peng, "A generalized dynamic programming principle and hamilton-jacobi-bellman equation," *Stochastics: An International Journal of Probability and Stochastic Processes*, vol. 38, no. 2, pp. 119–134, 1992.

[4] D. Q. Mayne, "Differential dynamic programming–a unified approach to the optimization of dynamic systems," in *Control and Dynamic Systems*. Elsevier, 1973, vol. 10.

[5] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *International Conference on Robotics and Automation*. ICRA, 2020.

[6] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[7] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Robotics: Science and Systems XV*. RSS, 2019.

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[9] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[10] D. Hoeller, F. Farshidian, and M. Hutter, "Deep value model predictive control," in *Conference on Robot Learning*. CoRL, 2020.

[11] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "A closer look at deep policy gradients," in *International Conference on Learning Representations*. ICLR, 2020.

[12] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.

[15] H. Qian and Y. Yu, "Derivative-free reinforcement learning: a review," *Frontiers of Computer Science*, vol. 15, no. 6, pp. 1–19, 2021.

[16] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integration (SII)*. SICE, 2019.

[17] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," *ICLR*, 2020.

[18] R. Tedrake *et al.*, "Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems," 2014.

[19] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*. ICML, 2013.

[20] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *IEEE International Conference on Robotics and Automation*. ICRA, 2018.

[21] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," in *International Conference on Learning Representations*. ICLR, 2019.

[22] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *Robotics: Science and Systems*. RSS, 2015.

[23] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the hindsight plan—episodic mpc improvement," in *IEEE International Conference on Robotics and Automation*. ICRA, 2017.

[24] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 2013.

[25] J. Viereck, A. Meduri, and L. Righetti, "Valuenetqp: Learned one-step optimal control for legged locomotion," *arXiv preprint arXiv:2201.04090*, 2022.

[26] A. Parag, S. Kleff, L. Saci, N. Mansard, and O. Stasse, "Value learning from trajectory optimization and sobolev descent: A step toward reinforcement learning with superlinear convergence properties," in *International Conference on Robotics and Automation*, 2022.

[27] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Świrszcz, and R. Pascanu, "Sobolev training for neural networks," in *Advances in Neural Information Processing Systems*. Neural IPS, 2017.

[28] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, 1991.

[29] T. M. Mitchell, S. B. Thrun *et al.*, "Explanation-based neural network learning for robot control," in *Advances in Neural Information Processing Systems*. Neural IPS, 1993.

[30] J.-W. Lee and J.-H. Oh, "Hybrid learning of mapping and its jacobian in multilayer neural networks," *Neural computation*, vol. 9, no. 5, 1997.

[31] J. B. Witkoskie and D. J. Doren, "Neural network models of potential energy surfaces: Prototypical examples," *Journal of chemical theory and computation*, vol. 1, no. 1, 2005.

[32] L.-z. Liao and C. A. Shoemaker, "Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems," Cornell University, Tech. Rep., 1992.

[33] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 695–702.

[34] S. Fleury, P. Soueres, J.-P. Laumond, and R. Chatila, "Primitives for smoothing mobile robot trajectories," *Transactions on Robotics and Automation*, vol. 11, no. 3, 1995.

[35] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," *Center for Cognitive and Neural Studies*, 2007.

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