SAFE DEPLOYMENT OF REINFORCEMENT LEARNING USING DETERMINISTIC OPTIMIZATION OF TRAINED NEURAL NETWORKS

Radu-Alexandru Burtea and Calvin Tsay*
Department of Computing, Imperial College London
South Kensington SW7 2AZ

Abstract

Enabling reinforcement learning (RL) to explicitly consider constraints is important for safe deployment in real-world process systems. This work exploits recent developments in deep RL and optimization over trained neural networks to introduce algorithms for safe training and deployment of RL schemes. We show how optimization over trained neural-network state-action value functions (i.e., a critic function) can explicitly incorporate constraints, and we describe two corresponding RL algorithms. The first uses constrained optimization of the critic to give optimal actions on which an actor is trained, while the second guarantees constraint satisfaction during deployment by directly implementing actions from optimizing a trained critic model. The two algorithms are tested on a supply chain case study from OR-Gym and are compared against state-of-the-art algorithms TRPO, CPO, and RCPO.

Keywords

Constrained reinforcement learning, Optimization of trained neural networks, Supply chain optimization.

Introduction

Reinforcement learning (RL) has been central to many notable successes in machine learning, such as in self-driving cars, playing games, and operating data centers (Shin et al., 2019). However, in comparison to traditional model-based control strategies, e.g., model predictive control, RL does not typically consider state constraints explicitly. For many practical engineering applications, simply maximizing reward without considering the appropriateness of actions can lead to highly undesirable consequences. For instance, in supply chain applications, an RL algorithm may direct all goods to the cheapest warehouse, without considering that the warehouse will eventually reach capacity, creating massive backlogs.

Given the above, it is desirable to impose constraints on the range of behavior that can be explored in RL. This has inspired research into so-called *safe* RL (García & Fernández, 2015). Several methods for safe RL allow implicit consideration of state constraints using stage-wise reward or penalty functions. Alternatively, some safe RL techniques use external knowledge, e.g., imitation learning, and/or risk metrics during exploration.

This work takes advantage of two recent developments to enable explicit consideration of state constraints during deployment of RL: (i) incorporation of deep neural networks into RL schemes (Arulkumaran et al., 2018), known as *deep* RL, and (ii) techniques for deterministic optimization of trained neural networks (Grimstad & Andersson, 2019; Tsay et al., 2021). We present two algorithms based on this combination. The first, called OMLT-DDPG, comprises an actor-critic method, where optimization over a critic neural network gives optimal actions on which to train the actor. The second, called SAFE, is a strategy for deployment, wherein directly

* To whom all correspondence should be addressed: c.tsay@imperial.ac.uk

implementing actions from optimizing a trained critic network guarantees constraint satisfaction.

The two proposed algorithms are applied to a multi-level supply chain case study from OR-Gym (Hubbs et al., 2020). Computational results demonstrate that OMLT-DDPG is significantly more sample-efficient compared to other RL methods, owing to the use of deterministic optimization. The results further show the SAFE explicitly satisfies known constraints during RL deployment.

Safe Reinforcement Learning Background

Markov decision processes, or MDPs, are the foundation of RL problems. A given MDP is defined by a state space *S*, action space *A*, reward function *R*, and transition probability *P*. The goal of RL is to learn a policy π that maximizes a performance metric $J(\pi)$, usually defined as the total expected reward over an infinite time horizon, subject to a discount factor. Safe RL methods seek to also enforce constraints, typically by introducing some cost function(s), analogous to reward. The inclusion of cost functions results in *constrained* MDP, or CMDP.

Recent methods for safe RL such as constrained policy optimization (Achiam et al., 2017) and interior-point policy optimization (Liu et al., 2020) can provide safety guarantees for CMDPs, but only in the form of simple constraints on expected total discounted cost.

Trust-Region Policy Optimization (TRPO)

Although TRPO (Schulman et al., 2015) does not consider constraints, it serves as the basis for several safe RL algorithms and is briefly described here. TRPO is a policy iteration algorithm based on computing the *advantage* of one policy over another, i.e., the expected improvement in the performance metric by switching from one policy to another. Schulman et al. (2015) provide a method for approximating this. As these approximate updates resemble a first-order method when the policy is differentiable, the step size between policies should be constrained. This is typically done using a trust region method, such as constraining the KL divergence to the old policy when maximizing the advantage function.

Constrained Policy Optimization (CPO)

Achiam et al. (2017) extend TRPO to a constrained MDP and introduce an algorithm useful for both safe exploitation and exploration. The extension involves adding constraints on the auxiliary cost functions on top of the constraint on the distance metric between two successive policy updates, resulting in the following optimization problem to find policy π_{k+1}:

$$\max_{\pi} J(\pi) \tag{1}$$

$$\text{s.t. } J_{C_i}(\pi) \leq d_i, \forall i = 1, \dots, m \tag{2}$$

$$D_{KL}(\pi, \pi_k) \leq \delta \tag{3}$$

where $J_{C_i}(\pi)$ is the total expected cost of the i^{th} constraint over an infinite time horizon and δ is the step size for policy iteration. The objective and constraints are replaced with surrogate functions, for which worst-case bounds are computed, based on the hyperparameters of the algorithm. Again, this problem is computationally difficult and is solved using a primal-dual method after linearizing $J(\pi)$ and $J_{C_i}(\pi)$, and a second-order expansion for $D_{KL}(\pi, \pi_k)$. This approximation motivates a small step size δ.

Reward Constrained Policy Optimization (RCPO)

Reward Constrained Policy Optimization (Tessler et al., 2019) is similar to CPO but instead solves an unconstrained optimization problem with Lagrange multipliers instead of using a Primal-Dual method with hard constraints. The optimization objective of the problems then becomes:

$$\min_{\lambda \geq 0} \max_{\pi} \big(J(\pi) - \lambda(J_C(\pi) - d)\big) \tag{4}$$

Tessler et al. (2019) consider that this optimization problem can be viewed on two timescales: a faster one, where the policy is optimized, and a slower one, which involves gradually increasing λ until the constraint is satisfied. This is achieved by selection of different step sizes for the updates to the Lagrange multipliers and the policy.

Algorithm 1: OMLT-DDPG

Our approach incorporates deterministic optimization of neural networks using the Optimization and Machine Learning Toolkit (OMLT) into Deep Deterministic Policy Gradients (DDPG). This section first summarizes DDPG and OMLT and then presents our algorithm.

Deep Deterministic Policy Gradients (DDPG)

The Deep Deterministic Policy Gradients algorithm (Lillicrap et al., 2016) was conceived as a continuous-space extension to the popular deep Q-learning framework (DQN). DQN cannot be directly applied to continuous action spaces, as selecting the action with maximum Q-value at a given state becomes complex and inefficient for high-dimensional action spaces. DDPG is an off-policy, model-free algorithm that instead uses the Q-value function to estimate the policy gradient.

Specifically, DDPG keeps a parameterized policy network, known as the *actor*, which deterministically maps states to actions. The other component of the algorithm is a critic network, which behaves as the Q-value function used in DQN. The actor is then trained using gradients from the critic, and the critic is trained by minimizing the difference between the expected discounted rewards if the greedy actor policy is followed, and the current Q-value assigned by the critic to the state-action pair.

Most DDPG implementations maintain a replay buffer to avoid "catastrophic forgetting" of previous transitions.

Optimization and Machine Learning Toolkit (OMLT)

The challenge of selecting the action with maximum Q-value from a given neural network (NN) can be viewed as optimization over a trained neural network (the neural network parameters are fixed during an RL step). We propose to address this using the Optimization and Machine Learning Toolkit (OMLT), an open-source package for optimization over pre-trained machine learning models (Ceccon et al., 2022).

OMLT enables engineers and optimizers to easily translate learned machine learning models to optimization formulations. OMLT 1.0 supports GBTs through an ONNX (https://github.com/onnx/onnx) interface and NNs through both ONNX and Keras interfaces. OMLT transforms pre-trained machine learning models into the Python-based algebraic modeling language Pyomo (Bynum et al., 2021) to encode optimization formulations. The literature often presents different optimization formulations as competitors, but in OMLT, competing optimization formulations become alternative choices for users.

DDPG with Deterministic Optimization of NNs

Pseudocode of our proposed OMLT-DDPG algorithm is presented in Figure 1. OMLT-DDPG preserves properties of the DDPG algorithm, but is extended for CMDPs and benefits from deterministic optimization of trained NNs.

Algorithm 1 OMLT-DDPG (one episode)

Require: $|R|, \sigma \geq 0$ ▷ Replay buffer, warm-up

Randomly initialize critic $Q(s, a|\theta^Q)$, actor $\mu(s|\theta^\mu)$, and target NNs $Q'(s, a|\theta'^Q)$, $\mu'(s|\theta'^\mu)$.
Observe: s_1
for $t = 1, \ldots, T$ **do**
  Action: $a_t = \mu(s_t|\theta^\mu)$
  Observe: r_t, c_t, s_{t+1}
  Store: $R \leftarrow (s_t, a_t, r_t, c_t, s_{t+1})$
  Sample $\hat{R}$ from R, $|\hat{R}| = N$
  $y_i = r_i + \gamma Q'(s_{i+q}, \mu'(s|\theta'^\mu)|\theta'^Q), \forall i \in \hat{R}$
  Update $Q(s, a|\theta^Q)$ by minimizing MSE against $y_i, \forall i \in \hat{R}$
  if $t \geq \sigma$ **then**
    Optimize: $a_i^* = \text{argmax } Q(s_i, a_i|\theta^Q)$ s.t. constraints c_i ▷ OMLT Package
    Update $\mu(s|\theta^\mu)$ by minimizing MSE against $a_i^*, \forall i \in \hat{R}$
  end if
  $\theta'^Q \leftarrow \tau\theta^Q + (1-\tau)\theta'^Q$
  $\theta'^\mu \leftarrow \tau\theta^\mu + (1-\tau)\theta'^\mu$
end for

Figure 1. OMLT-DDPG Algorithm

The algorithm is initialized with actor and critic neural networks, as well as *target* actor and critic networks. For each episode the agent observes the initial state and executes the policy defined by the actor network, storing transitions in the replay buffer. The algorithm then samples a batch of transitions of length *N* and updates the critic network against the target network.

Up to this point, our algorithm closely resembles DDPG. However, the original DDPG uses the negative value of the critic for a given state-action pair as the loss for training the actor. DDPG-OMLT instead uses the optimal actions obtained by optimizing the critic network in OMLT, subject to the problem constraints. Interestingly, the algorithm preserves the theoretical properties of the policy gradients used in DDPG, if the optimal actions are obtained by optimizing for the gradients of the critic network (this is the case if a gradient-based optimizer such as *ipopt* is used).

Note that this comprises solving one constrained optimization problem for each sample from the replay buffer. Therefore, we use the predicted action from the actor as the initial guess to expedite optimization. We hypothesize that in the initial stages of training the actions chosen in this manner will be suboptimal, as the critic is not accurate enough to "critique" state-action pairs accurately. Nevertheless, this may prove beneficial for exploration. Given the above, we introduce a warm-up period σ, where only the actor is fixed and only the critic is updated.

Following the actor and critic updates, the target networks are also updated using *soft* updates, i.e., only a certain portion τ of the weights are updated. Lillicrap et al. (2016) found this to improve stability.

Algorithm 2: SAFE

The above DDPG-OMLT algorithm promotes safe exploration and exploitation by always incorporating environment constraints when optimizing over the critic network to select optimal actions. However, when used in deployment, the actor can still give an action that results in constraint violation. This section describes an algorithm, SAFE, that explicitly enforces constraints in deployment.

Usually, in an actor-critic setting the actor network gives the action to take (as suggested by the name). OMLT enables us to directly use the critic network to choose the optimal actions at each state, which guarantees constraint satisfaction and high rewards. The pseudocode that describes this can be found in Figure 2.

A key different between OMLT-DDPG and SAFE is that OMLT-DDPG uses the constraint values in the *previous* timestep to evaluate the actor (after the action has been taken), while SAFE uses the constraint values in the *current* timestep to only take an action that is feasible. If an actor network is available, this can be used to provide initial guesses to the optimization algorithm.

Algorithm 2 SAFE

Require: Trained critic $Q(s, a|\theta^Q)$

Observe: s_1, c_1

Optimize: $a_1^* = \text{argmax } Q(s_i, a_i|\theta^Q)$

Action: $a_1 = a_1^*$

s.t. constraints c_1 ▷ OMLT Package

for $t = 2, \ldots, T$ **do**

  Observe: s_t

  Optimize: $a_t^* = \text{argmax } Q(s_t, a_t|\theta^Q)$

  s.t. constraints c_{t+1} ▷ OMLT Package

  Action: $a_t = a_t^*$

end for

Figure 2. SAFE Algorithm

We note that SAFE is model-agnostic, meaning that it can use any state-action value neural network, as long as the activation functions used in the neural network are supported by OMLT. While we only test SAFE with the critic models from OMLT-DDPG, any Q-value neural network that uses a state-action pair as input can theoretically be used in this framework.

Computational Results

We employ the multilevel supply chain case study from OR-Gym to test OMLT-DDPG and SAFE. We further implement TRPO, CPO, and RCPO as baselines for comparison.

Description of Environment

We select a supply chain inventory management problem from OR-Gym (Hubbs et al., 2020) as the environment for our experiments. The model comprises a supply chain with multiple levels organized in a tree-like structure from production nodes to distribution and retail. The agent must place replenishment orders at nodes throughout multiple levels of a supply chain, subject to lead times and uncertain customer demand at retail nodes.

Each episode has a predetermined length, and there are no conditions for early termination. Inventories are subject to capacity constraints, with excess incurring a penalty cost. Likewise, penalties are incurred for unmet customer demand. Given the above, a single interaction between the agent and the environment comprises the following steps:

1. Each node gives replenishment orders to upstream nodes (constrained by available inventory).
2. Replenishment orders are shipped with lead times.
3. Demand is generated at retail nodes and is either met by available inventory or backlogged.
4. A holding cost is charged for surplus inventory.

The reward function at each step is computed as the sales revenue, less the procurement and operating costs, as well as costs associated with unfilled demand and holding inventory. Our constrained formulations enforce the capacity limits for the inventories at modeled nodes.

A three-level supply chain is selected, and algorithms are run for 150 episodes, except for RCPO, which required significantly more episodes to converge. Uncertainty is introduced by customer demand, which we model using a Poisson distribution with mean of 20. Each experiment is repeated with five random starts. Note that we have used a variant of the environment with backlog instead of lost sales. The latter would simply incur a penalty whenever demand is not met in a particular period.

DDPG is typically run with a lower learning rate for the actor than for the critic, allowing the two to converge on different "timescales." However, we found OMLT-DDPG to benefit from a faster learning rate for the actor, and we set the actor and critic learning rates to, respectively, 0.005 and 0.001. We used a batch size of 8, sampled from a replay buffer size of maximum 35000. We found a strong tradeoff between performance gain and computational time related to the batch size. We use a batch size of 40000 for CPO and TRPO.

Safe Reinforcement Learning Results

Figure 3 and Table 1 compare the performance of the various algorithms through 150 episodes. TRPO, CPO, and RCPO exhibit more stable behavior, while OMLT-DDPG seems to fluctuate. This can be attributed to the stability associated with enforcing constraints on maximal KL divergence between policy updates. However, OMLT with warm-up achieves rewards only 6% and 28% lower than CPO and TRPO, respectively, which is noteworthy as OMLT-DDPY uses 1334x fewer samples. This may be attributed to the information gain from the use of deterministic optimization. Indeed, OMLT-DDPG achieves a reward of 200 after only four episodes, and this steep learning curve is consistent in our experiments, suggesting behavior as a "few-shot" learner.

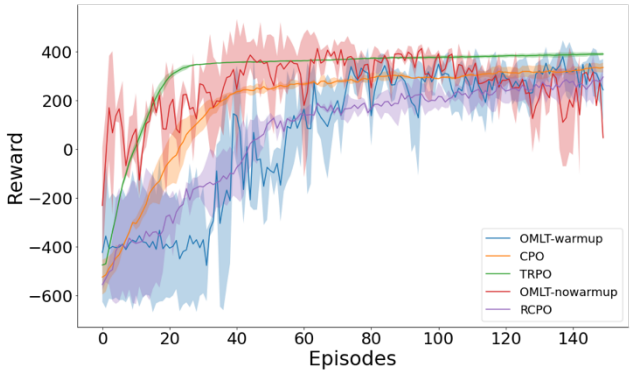

Figure 3. Training rewards in supply chain case study.

Table 1. Training performance for supply chain case study

	OMLT DDPG	OMLT DDPG (NW)	CPO	TRPO	RCPO
Reward ($\times 10^{-3}$ final 20 ep.)	316.1 ±70.0	223.6 ±129.1	323.7 ±17.7	386.4 ±8.6	288.5 ±34.5
Penalties (final 20 ep.)	3052 ±2507	212 ±301	2727 ±216	2023 ±676	47±67
CPU Time	~3h	~3.5h	~1h	~1h	~0.1h
Samples	4500	4500	6 mil	6 mil	45000

In this experiment, the incorporation of the warm-up period does not seem to help OMLT-DDPG; however, more experiments documenting when the warm-up period helps are provided in our presentation. In general, the lack of warm-up period results in more aggressive behavior by the agent. Figure 4 compares how close the orders at the first supply chain node are between the implementations of OMLT-DDPG with and without a warm-up period. Without a warm-up period, the actor initially takes more varied actions, but eventually settles to a more conservative operating regime. This is partially explained by the Q-value distributions of the critic: without warm-up, the distribution is relatively flat, but with warm-up, we found the critic to (at least initially) overvalue actions with high means.

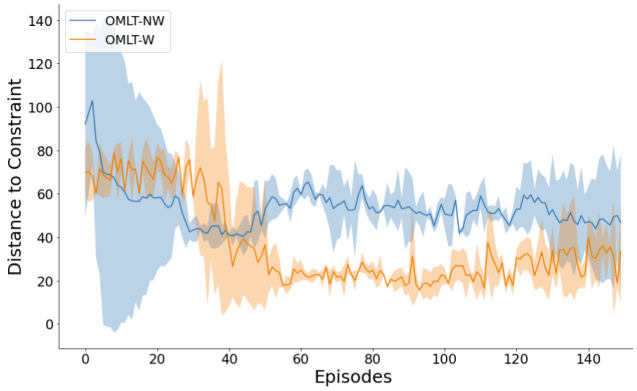

Figure 4. Distance to constraint for the first node for OMLT-DDPG with and without a warm-up period.

Safe Deployment Results

To simulate safe deployment in production, we deploy the models trained after the 150 episodes. For OMLT-DDPG without warm-up, we use a snapshot of the models from episode 65, after which the models seem to overfit the environment. Deployment of the algorithms in production is simulated by using the learned models subject to new random episodes without further training. The rewards and penalties incurred by using these trained models to operate the supply chain are shown in Figure 4 and Table 1.

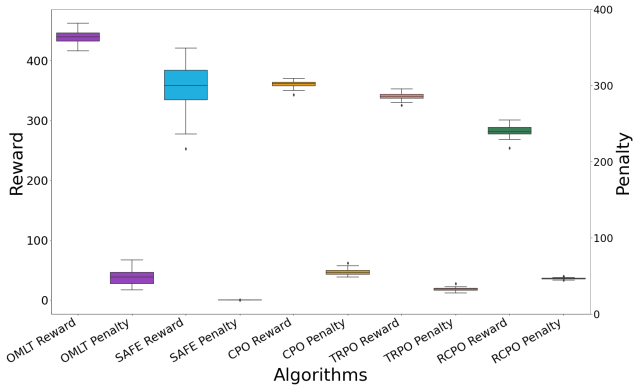

Figure 5. Rewards and penalties incurred during safe deployment in supply chain case study.

Table 2. Deployment performance for supply chain case study

	OMLT DDPG	SAFE	CPO	TRPO	RCPO
Reward ($\times 10^{-3}$)	433.9 ±23.9	365.0 ±35.4	359 ±10.1	340.7 ±9.3	280.8 ±19.3
Penalties ($\times 10^{-3}$)	40.3 ±21.8	0±0	48.0 ±8.8	19.2 ±5.8	36.3 ±2.5

Figure 4 and Table 2 show that OMLT-DDPG outperforms the other algorithms in terms of rewards; it also obtains smaller penalties. Compared to CPO, OMLT-DDPG achieves 21% higher rewards and 20% less penalties in deployment. The SAFE algorithm results in more unstable rewards, but is the only algorithm to not accumulate any penalties during deployment. This demonstrates the value of explicitly enforcing process constraints during replyment of RL.

Conclusions

This presentation introduces two algorithms, OMLT-DDPG and SAFE, for safe reinforcement learning and deployment, based on constrained optimization over trained critic networks. The first algorithm uses an actor-critic framework; constrained optimization over the critic network is used to provide targets on which the actor is trained. We show that this algorithm is very sample efficient, resembling behavior of a “few-shot” learner. The second algorithm uses constrained optimization over a pre-trained critic network to explicitly enforce process constraints during deployment.

Acknowledgments

The authors gratefully acknowledge support from the Engineering & Physical Sciences Research Council (EPSRC) through fellowship grant EP/T001577/1 and an Imperial College Research Fellowship to CT.

References

Achiam, J., Held, D., Tamar, A., Abbeel, P. (2017). Constrained policy optimization. *In Proceedings of the 34th International Conference on Machine Learning (ICML), PMLR*, 70, 22.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6), 26.

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., Watson, J. P., Woodruff, D. L. (2021). *Pyomo-optimization modeling in Python*. Springer, New York.

Ceccon, F., Jalving, J., Haddad, J., Thebelt, A., Tsay, C., Laird, C. D., Misener, R. (2022). OMLT: Optimization & Machine Learning Toolkit. *arXiv:2202.02414*

García, J., Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16, 1437.

Grimstad, B., Andersson, H. (2019). ReLU networks as surrogate models in mixed-integer linear programs. *Comput. Chem. Eng.*, 131, 106580.

Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., Wassick, J. M. (2020). OR-Gym: A reinforcement learning library for operations research problems. *arXiv 2008:063192*

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. (2016). Continuous control with deep reinforcement learning. *In Proceedings of the International Conference on Learning Representations*.

Liu, Y., Ding, J., Liu, X. (2020). IPO: Interior-point policy optimization under constraints. *In Proceedings of the AAAI Conference on Artificial Intelligence*, 34, 4940.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P. (2015). Trust region policy optimization. *In Proceedings of the 32nd International Conference on Machine Learning (ICML)*, *PMLR*, 37, 1889.

Shin, J., Badgwell, T. A., Liu, K. H., Lee, J. H. (2019). Reinforcement learning-overview of recent progress and implications for process control. *Comput. Chem. Eng.*, 127, 282.

Tessler, C., Mankowitz, D. J., Mannor, S. (2019). Reward constrained policy optimization. *In Proceedings of the International Conference on Learning Representations*.

Tsay, C., Kronqvist, J., Thebelt, A., Misener, R. (2021). Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. *In Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 34, 3068.