7-2019

Resource constrained deep reinforcement learning

Abhinav BHATIA

Pradeep VARAKANTHAM

Akshat KUMAR
Singapore Management University, akshatkumar@smu.edu.sg

Citation

BHATIA, Abhinav; VARAKANTHAM, Pradeep; and KUMAR, Akshat. Resource constrained deep reinforcement learning. (2019). *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019), Berkeley, CA, July 11-15*. 610-620.
Available at: https://ink.library.smu.edu.sg/sis_research/5059

Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Resource Constrained Deep Reinforcement Learning

Abhinav Bhatia, Pradeep Varakantham, Akshat Kumar
School of Information Systems
Singapore Management University
80 Stamford Rd, Singapore 178902
{abhinavb, pradeepv, akshatkumar}@smu.edu.sg

Abstract

In urban environments, resources have to be constantly matched to the "right" locations where customer demand is present. For instance, ambulances have to be matched to base stations regularly so as to reduce response time for emergency incidents in ERS (Emergency Response Systems); vehicles (cars, bikes among others) have to be matched to docking stations to reduce lost demand in shared mobility systems. Such problems are challenging owing to the demand uncertainty, combinatorial action spaces and constraints on allocation of resources (e.g., total resources, minimum and maximum number of resources at locations and regions).

Existing systems typically employ myopic and greedy optimization approaches to optimize resource allocation. Such approaches typically are unable to handle surges or variances in demand patterns well. Recent work has demonstrated the ability of Deep RL methods in adapting well to highly uncertain environments. However, existing Deep RL methods are unable to handle combinatorial action spaces and constraints on allocation of resources. To that end, we have developed three approaches on top of the well known actor-critic approach, DDPG (Deep Deterministic Policy Gradient) that are able to handle constraints on resource allocation. We also demonstrate that they are able to outperform leading approaches on simulators validated on semi-real and real data sets.

1 Introduction

This paper is motivated by aggregation systems that aggregate supply to improve efficiency of serving demand. Such systems have been employed in mobility systems, emergency response, logistics, food delivery, grocery delivery, and many others. There are multiple supply resources (e.g., ambulances, delivery/movement vehicles, taxis) controlled by a central agency that need to be continuously allocated to supply entities (e.g., base stations, docking stations) so as to improve service efficiency for customer demand. This sequential allocation problem becomes challenging due to combinatorial action space (allocating resources to entities), cost of reallocation, uncertainty in demand arrival, constraints on resource allocation and in some cases also due to uncertainty in resource movement.

Existing systems typically employ myopic (single or few time steps) and greedy optimization approaches (Yue, Marla, and Krishnan 2012; Ghosh et al. 2017; Powell 1996; Lowalekar et al. 2017) to optimize allocation of supply resources to locations. As we demonstrate in our experimental results, greedy approaches perform poorly when there are surges in demand or when variance in demand is high. Recent extension to employ Deep Learning with Reinforcement Learning, referred to as Deep RL, has significantly improved the scalability and effectiveness of RL in dealing with complex domains (Mnih et al. 2015; 2016; Lillicrap et al. 2015). In this paper, we propose the use of Reinforcement Learning (RL) approaches to learn decisions in aggregation systems that can better represent and account for the sequential nature of decision making and uncertainty associated with demand.

However, current Deep RL methods are not directly suitable for handling aggregation systems of interest due to two reasons: (i) Deep RL methods do not scale well in domains with discrete and combinatorial action space, more so in problems at the scale of a city; (ii) Due to resource allocation constraints, action space is constrained. There have been research works that have provided mechanisms for solving resource allocation problems with Deep RL (Dulac-Arnold et al. 2015; Mao et al. 2016). However, they do not consider constraints on resource allocation. (Amos and Kolter 2017) have integrated quadratic optimization problems as individual layers in end-to-end trainable deep learning networks. Such networks (OptNet) could potentially be integrated with RL to handle resource constraints. Unfortunately, as indicated in their paper, they can only solve small problems due to the computational complexity of training these networks. (Pham, De Magistris, and Tachibana 2018) proposed an architecture (OptLayer) building on ideas from OptNet for constrained RL in the context of robotics. They were able to demonstrate scaling to problems on a 6-DoF robot (6 dimensional action space). Unfortunately, their approach does not scale to problems of our interest where we have up to 95 dimensions.

DDPG (Deep Deterministic Policy Gradient) (Lillicrap et al. 2015) is an approach that has been applied to multi dimensional continuous control problems with great results. However, like the other Deep RL methods, it is also unable to handle constraints on resource allocation. We propose extensions to DDPG that are able to handle constraints on resource allocation. We make five key contributions in this paper. First, we formally define the Resource Constrained Reinforcement

Learning (ReCO-RL) model to represent problems of interest (decision making in aggregation systems). We are specially interested in hierarchical linear constraints, a useful subset of ReCO-RL problems. Second, we provide an extension to DDPG referred to as Constrained Projection (CP) that is generic (works for any kinds of resource constraints) but ensures constraints only approximately and has adhoc theoretical justifications. Third, we provide a novel, fast, scalable, easy-to-implement Constrained Softmax (CS) extension to DDPG that provably ensures constraints on resource allocation, but works only for a subset of hierarchical linear constraints. Next, we provide another novel, fast and scalable, Approximate OptLayer (ApprOpt) extension to DDPG that can provably handle any hierarchical linear constraints, while being orders of magnitude faster than OptLayer. Finally, we demonstrate that our extensions DDPG-CP, DDPG-CS and DDPG-ApprOpt provide either comparable or significantly better solutions than existing best approaches on two simulators for emergency response and bike sharing. Customer demand in these simulators was generated using real or semi-real datasets.

2 Background

In this section, we briefly describe the Reinforcement Learning (RL) problem (Sutton and Barto 1998) and the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al. 2015) that is used to learn in environments with continuous action spaces.

The RL problem to maximize the long term reward while operating in an environment can be represented as a Markov Decision Process (MDP). Formally, an MDP is represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $T(s, a, s')$ represents the stochasticity in the underlying environment and provides the probability of transitioning from state s to state s' on taking action a. $R(s, a)$ represents the reward obtained on taking action a in state s. The RL problem is to learn a policy that maximizes the long term reward from *experiences* without knowing the exact model of transitions and rewards. An experience is defined as a tuple (s, a, s', r), and typically learning happens over a batch of experiences (referred to as an episode) that ends when s' is a terminal state. Q-learning represents the value function for being in state s and taking action a:

$$Q(s, a) = \mathbb{E}_{s',r}[r + \gamma \cdot \max_{a'} Q(s', a')] \tag{1}$$

where the expectation, $\mathbb{E}$ is over the stochasticity in the environment with respect to transitions and also reward.

Since, we extend on Deep Deterministic Policy Gradient (DDPG) approach in this paper, we provide a brief description. DDPG works in domains with continuous action spaces. In DDPG, we have a *critic* function Q parameterized by θ^Q that approximates the state-action-value function. We also have an *actor* μ parameterized by θ^μ that outputs the deterministic action in a continuous space given the current state. Let N denote the size of the batch of total experiences $e_i = (s_i, a_i, s_{i+1}, r_i), i = 1..N$ collected in an episode. The critic is updated by minimizing the loss:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\theta^Q))^2, \text{where}$$

$$y_i = r_i + \gamma \cdot Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

where Q' and μ' are *target* networks parameterized by $\theta^{Q'}$ and $\theta^{\mu'}$ respectively. The parameters of these target networks are made to slowly track the parameters of the original networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. This is done to avoid making targets y_i non-stationary, and improve the stability of updates. Next, the actor policy μ is updated by using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

3 Resource Constrained Reinforcement Learning (ReCO-RL)

In our motivating problems of interest, supply resources have to be allocated online to the right entities to efficiently serve demand. There is uncertainty in demand (w.r.t. both serving time and arrival) and potentially also in resource movement. Since allocation decisions at one stage have an impact on the subsequent stages and there is transitional uncertainty, RL is an ideal model for problems of interest in this paper. However, a *key differentiating factor* from typical RL problems is that the action space in problems of interest is constrained due to resource allocation constraints.

We propose a modification to the RL model that can represent such constraints on resource allocation. We call the new model *Resource Constrained Reinforcement Learning* (ReCO-RL). To capture the domains of interest, we have n entities (e.g., base stations) where supply resources are situated and m zones that capture customer demand. Similar to the RL setting, the underlying tuple is $< S, A, T, R >$.

- **States, S:** Each state $s \in S$ is a tuple $\langle b_1, \ldots, b_n, d_1, \ldots, d_m, t \rangle$ where b_k is the number of resources assigned to entity k, d_l is the demand for resources in zone l, and t is the decision epoch.
- **Actions, A:** Each action $a \in A$ is a tuple $\langle a_1, \ldots, a_k, \ldots, a_n \rangle$ where a_k represents the number of resources to be assigned to entity k. Depending on the domain, there can be different allocation constraints on the action, including but not limited to:
  1. *Global sum constraint:* This enforces the global constraint on number of total resources available:
  $$\sum_k a_k = C \tag{2}$$
  2. *Local minimum and maximum bounds:* These constraints enforce the minimum and maximum number of resources to be allocated to an entity. For a given entity k,
  $$\check{C}_k \le a_k \le \widehat{C}_k$$
  $$\forall k, \check{C}_k, \widehat{C}_k \in [0, C]; \sum_k \check{C}_k \le C \le \sum_k \widehat{C}_k \tag{3}$$

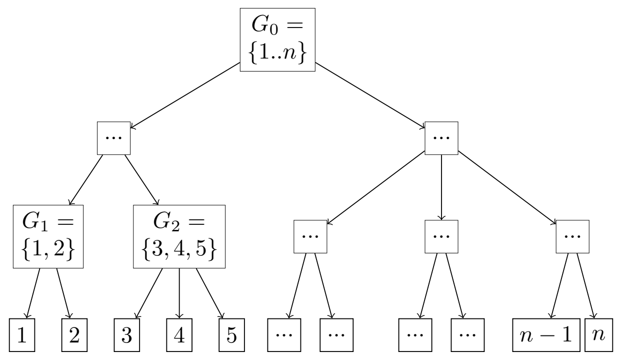

Figure 1: Region Hierarchy Example

3. *Regional minimum and maximum bounds:* These constraints enforce the minimum and maximum number of resources to be allocated to a region (a subset of entities). For a given region G_j:

$$\check{C}_{G_j} \leq \sum_{k \in G_j} a_k \leq \widehat{C}_{G_j} \tag{4}$$

It should be noted that $\check{C}_{\{1..n\}} = \widehat{C}_{\{1..n\}} = C$ due to the global sum constraint. Also, $\forall k : \check{C}_{\{k\}} = \check{C}_k$ and $\widehat{C}_{\{k\}} = \widehat{C}_k$, due to the local minimum and maximum bound constraints respectively. In the most general case, constraints can be on any subsets of regions. However, in practice and in problem domains of interest, there is a region hierarchy to ensure effective management. For instance, in emergency response domain, for allocating ambulances:
(a) The city is divided into multiple major regions (East, West, North, South, Centre);
(b) Each major region is divided into communities;
(c) Each community has some base stations;
The connection between regions and entities can in such cases be represented as a tree as shown in Figure 1.

- **Transitions and Rewards:** $T(s, a, s')$ captures uncertainty in demand and movement of resources between entities. $R(s, a, s')$ represents the demand served or the utility of serving the demand.

We now provide two examples of how ReCO-RL can represent the problems of interest:

Emergency Response as ReCO-RL: Emergency Response Systems (ERSs) are tasked with reducing the response times for emergencies in many cities by using resources like ambulances, fire trucks etc. There are n base stations (entities) where ambulances (or other resources) are placed and requests for ambulance can arise anywhere in the city that is divided into m zones. The goal is to place the right number of ambulances at the base stations, so as to optimize bounded time response (number of requests served within bounded time) (Yue, Marla, and Krishnan 2012). b_k represents the number of ambulances at kth base station; d_l represents the demand in lth zone. With respect to action, a_k represents the number of ambulances to be assigned to kth base station. For bounded time response, reward is 1 for every request that is served within bounded time, zero otherwise. Transitions between states are dependent on demand patterns and action taken with respect to movement of ambulances.

Bike Placement as ReCO-RL: In bike placement problem, there are n docking stations where bikes are placed and requests for bikes can arise at these docking stations (thus in this case $m = n$). The goal is to place the right number of bikes at the right docking stations at the right times, so as to reduce lost demand (Ghosh et al. 2017). With respect to state, b_k represents the number of bikes at kth docking station. d_l represents the demand in lth zone. With respect to action, a_k represents the number of bikes to be assigned to kth docking station. Reward is -1 for every lost customer due to lack of bikes at a docking station. Transitions between states are dependent on demand patterns and action taken with respect to movement of bikes.

3.1 Extensions to ReCO-RL

In the definition of ReCO-RL, we have considered a single type of resource and we do not distinguish between resources assigned to the same entity. However, it is easy to extend the model to consider multiple types of resources (e.g., multiple types of ambulances and bikes). We will have state features to be s_k^ζ indicating the number of resources of type ζ assigned to entity k. We will have a similar modification to action features, a_k^ζ indicating the number of resources of type ζ assigned to entity k. Constraints can then be defined on these new action features in a similar way. As we show in Footnote 1, our approaches can still be applied, as state and actions can be converted to continuous space in a similar way. For purposes of easy explainability and since there are many domains which operate with single resource types, we focus on single resource type in this paper.

4 Approaches

ReCO-RL problems have a discrete and combinatorial action space in problems of interest in this paper. For instance, even the simplest ambulance allocation problems considered in this paper have approximately 32^{25} possible actions. Due to the combinatorial action space and the presence of constraints on actions, existing approaches for Deep RL are not suitable. Deep Deterministic Policy Gradient (DDPG) approach is also not directly applicable. However, in this paper, we propose novel extensions on top of DDPG to solve ReCO-RL problems effectively and efficiently.

For DDPG to be applicable for solving ReCO-RL problems, there are two key challenges:

1. Action space should be continuous and not discrete.
2. Address constraints on actions. Such constraints imply not every action obtained using actor network is feasible and furthermore, unconstrained exploration strategies (like Ornstein-Uhlenbeck process) are not applicable (as they result typically in violation of action constraints).

First, we consider the easier challenge of dealing with discrete action space. Action space is discrete and combinatorial in domains of interest due to the need for allocation of resources at every decision epoch. However, it is easy to approximate such discrete and combinatorial resource allocation actions into continuous actions. For instance, consider a discrete action $a = (10, 20, 30, 40)$ that represents 10 resources assigned to entity 1, 20 resources assigned to entity 2 and so on. This is (approximately) equivalent to $\tilde{a} = (0.1, 0.2, 0.3, 0.4)$, where 0.1 refers to the fraction of resources assigned to entity 1, 0.2 refers to the fraction of resources assigned to entity 2 and so on.[1] In this paper, we employ such a conversion. Since we convert to a continuous action space, the constraints also get normalized to be between 0 and 1. We refer to $\check{\mathcal{C}}$ as the normalized lower bound of $\check{C}$ (i.e. $\check{\mathcal{C}} = \frac{\check{C}}{C}$) and $\hat{\mathcal{C}}$ ($= \frac{\hat{C}}{C}$) the normalized upper bound. In this converted continuous action space, the action components thus must sum to $\mathcal{C} = \frac{C}{C} = 1$.

Addressing the second challenge of handling constraints on actions within DDPG is one of the key contributions of this paper. We provide three methods in the context of DDPG:

1. Constrained Projection (CP): For Reco-RL problems, the actor network of DDPG generates infeasible actions. In this method, we employ penalties to train the actor network to generate feasible actions. Thus the policy gradient is computed at the infeasible output of the actor network. For the purpose of taking an action in the environment, whenever the actor generates an infeasible action, we use its nearest projection in the feasible action space, computed using a Quadratic Program (QP).
2. Constrained Softmax (CS): In this approach, we introduce modifications of the traditional softmax function as new layers in the actor network to ensure that it generates feasible actions. The layers are differentiable since they are essentially closed form expressions. These layers are part of the end to end backpropagation training of the actor i.e. the policy gradient is computed at the output of these layers. These layers can handle a subset of local and hierarchical regional constraints.
3. Approximate OptLayer (ApprOpt): In this approach, we introduce new differentiable layers based on ideas from OptLayer, but orders of magnitude faster. The speedup comes from solving the QP approximately in a semi closed-form semi-iterative fashion, which makes computing the gradients trivial. These layers are part of the end to end backpropagation training of the actor. These can handle the full breadth of local and hierarchical regional constraints.

The feasible action $\vec{z}$ computed using the above methods is continuous and its components sum to 1. As in the Wolpertinger approach (Dulac-Arnold et al. 2015), we round off $C \times \vec{z}$ to the nearest (by L_1 distance) discrete action a to act in the environment. $\tilde{a} = a/C$ is added to the experience buffer and thus used to train the critic.

As indicated earlier, unconstrained exploration strategies (like Ornstein-Uhlenbeck process, which adds noise to the generated action) cannot be used in ReCO-RL problems because they can result in violation of action constraints. Therefore, we employ adaptive parameter noise (Plappert et al. 2017) for exploration in all the approaches.

Now we will present each of the approaches in detail. For DDPG-CS and DDPG-ApprOpt, first we will show how to generate feasible actions $\vec{z}$ in presence of local constraints only. Later, we will show how these local-constraints handling layers (or LCHLs) can be nested to build a differentiable module to handle hierarchical regional constraints as well.

4.1 Constrained Projection

In this method, we employ penalties to train the actor network to generate feasible actions. In case the generated action is infeasible, then for the purpose of taking an action in the environment, we use the nearest projection of the infeasible action in the feasible action space, computed using a Quadratic Program (QP).

Let $x_k = \mu_k(s_t|\theta^\mu)$ refer to the actor output corresponding to resource allocation for entity k on observing state s_t at timestep t, with no activation function used in the final layer. Let the activated output be:

$$y_k = \frac{\tanh(x_k) + 1}{2} \tag{5}$$

The actor is trained to satisfy the allocation constraints by adding *violation* cost penalty terms to the policy gradient equation. The violation cost ν is given by:

$$\nu(\vec{y}) = |1 - \sum_k^n y_k| + \sum_{G_j} \max(0, \check{\mathcal{C}}_{G_j} - \sum_{k \in G_j} y_k) + \sum_{G_j} \max(0, \sum_{k \in G_j} y_k - \hat{\mathcal{C}}_{G_j}) \tag{6}$$

In DDPG-CP, the overall gradient is a linear combination of the sampled policy gradient taken at $\vec{y}$, and the negative of the gradient of $\nu(\vec{y})$.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \Big[\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\vec{y}(s_i)} \nabla_{\theta^\mu} \vec{y}(s|\theta^\mu)|_{s_i} - \lambda \cdot \nabla_{\theta^\mu} \nu(\vec{y})|_{\vec{y}=\vec{y}(s_i)} \Big] \tag{7}$$

where $\lambda > 0$ is tuned for each domain separately.[2]

For stepping in the environment, we identify the nearest L_2 projection $\vec{z}$ of $\vec{y}$ that satisfies all the given constraints using the following QP:

$$\text{Minimize} \|\vec{z} - \vec{y}\|_{L_2}, \text{subject to}$$
$$\sum_k^n z_k = 1; \quad \check{\mathcal{C}}_{G_j} \leq \sum_{k \in G_j} z_k \leq \hat{\mathcal{C}}_{G_j}, \forall G_j \tag{8}$$

This method has a very broad scope and can handle even non-hierarchical regional constraints. The main weakness of this approach is that the policy gradient is computed at $\vec{y}$, which need not be feasible. This can be a problem since the critic is trained on feasible actions $\tilde{a}$, making the Q function at $\vec{y}$ theoretically undefined and practically ill defined. Thus this method is theoretically adhoc.

[1]In case of multiple resource types, we will normalize each resource type separately. For instance, if there are two resource types, then an action $(10, 20, 5, 10)$ – which indicates 10 resources of type 1 to entity 1, 20 resources of type 1 to entity 1, 5 resources of type 2 to entity 1, 10 resources of type 2 to entity 2 – gets converted to $(\frac{10}{30}, \frac{20}{30}, \frac{5}{15}, \frac{10}{15})$.

[2]We use $\lambda = 10^3$ for emergency response domain and $\lambda = 10^5$ for bike sharing domain, as those values yield the best performance.

4.2 Constrained Softmax

In this method, we introduce new differentiable layers to the actor network that are dependent on the type of constraints present in the problem. These new layers compute a softmax output $\vec{z}$ over the actor network output while satisfying the allocation constraints, and hence we refer to these additional layers collectively as *constrained softmax* layers. These layers become part of the end to end backpropagation training of the actor i.e. the policy gradient is computed at $\vec{z}$. Since *constrained softmax* layers are dependent on the type of constraints, we describe the changes for each type of constraint separately.

Global Sum Constraint We will start with the simplest case of having just the global constraint i.e. $\sum_k a_k = 1$. This can be handled by having only one additional layer at the end of the actor network, i.e. the traditional softmax layer. Assume $\vec{x} = \mu(s_t|\theta^\mu)$ is any arbitrary vector output of the actor network, with no activation function used in the final layer. We compute softmax over $\vec{x}$ to give the feasible action $\vec{z}$ as follows:

$$z_k = \frac{y_k}{\sum_k y_k} \tag{9}$$

$$\text{where activation} \qquad y_k = e^{min(0,x_k)} \tag{10}$$

Local Minimum and Maximum Bounds We now consider the local minimum and maximum bounds for each entity k. Like in the previous case, we just need one extra layer that is a modification of the softmax layer to handle local bounds. The goal here is to identify a function $\vec{z}$ (with inputs $\vec{y}$ and bounds $\{\check{\mathcal{C}}_k\}$ and $\{\hat{\mathcal{C}}_k\}$) that satisfies the following properties:

$$\begin{aligned}
&\textit{Min, max bounds: } \check{\mathcal{C}}_k \leq z_k(\vec{y}, \{\check{\mathcal{C}}_i\}, \{\hat{\mathcal{C}}_i\}) \leq \hat{\mathcal{C}}_k \\
&\textit{Global sum constraint: } \sum_k z_k(\vec{y}, \{\check{\mathcal{C}}_k\}, \{\hat{\mathcal{C}}_k\}) = 1 \\
&\textit{Monotonicity: } \frac{\partial z_k}{\partial y_k} \geq 0 \textbf{ and } \forall i \neq k, \frac{\partial z_k}{\partial y_i} \leq 0
\end{aligned} \tag{11}$$

Monotonicity is required to identify the conditions where the functional form will yield a maximum. We have found one functional form that satisfies the properties above under some conditions. Proposition 1 provides this functional form (which is a minor modification to the traditional softmax layer) along with the conditions under which they are applicable.

Proposition 1 *If we only have the sum constraint and local maximum bounds $\{\hat{\mathcal{C}}_k\}$ for individual entities, then the feasible outputs $\{z_k\}$ are given by:*

$$z_k(\vec{y}, \vec{0}, \{\hat{\mathcal{C}}_k\}) = \frac{y_k + \epsilon_k}{\sum_i \left[y_i + \epsilon_i\right]} \textit{ with } \epsilon_k = \frac{\hat{\mathcal{C}}_k \cdot (n-1)}{\sum_i \hat{\mathcal{C}}_i - 1} - 1 \tag{12}$$

where $\forall k, \epsilon_k \geq 0$, $\sum_i \hat{\mathcal{C}}_i \neq 1$

Proof: There are three steps to the proof:

Step 1: z_k satisfies monotonicity properties of (11):

$$\frac{\partial z_k}{\partial y_k} = \frac{\sum_i \left[y_i + \epsilon_i\right] - (y_k + \epsilon_k)}{(\sum_i \left[y_i + \epsilon_i\right])^2} = \frac{\sum_{i \neq k} \left[y_i + \epsilon_i\right]}{(\sum_i \left[y_i + \epsilon_i\right])^2}$$

$$\frac{\partial z_k}{\partial y_i} = \frac{-(y_k + \epsilon_k)}{(\sum_i \left[y_i + \epsilon_i\right])^2}$$

Since $\forall i$, $y_i \geq 0$ (from Equation 10) , $\frac{dz_k}{dy_k} \geq 0$ and $\frac{dz_k}{dy_i} \leq 0$ in all cases if we have $\forall i, \epsilon_i \geq 0$

Step 2: Given the monotonicity properties of z_k, the maximum value for z_k in Equation (12) occurs when $y_k = 1$ and $\sum_{j \neq k} y_j = 0$. Therefore:

$$\frac{1 + \epsilon_k}{1 + \sum_k \epsilon_k} = \hat{\mathcal{C}}_k \implies 1 + \epsilon_k = \hat{\mathcal{C}}_k \cdot (1 + \sum_k \epsilon_k)$$

$$\implies \hat{\mathcal{C}}_k \cdot \epsilon_1 + \ldots + (\hat{\mathcal{C}}_k - 1) \cdot \epsilon_k + \ldots \hat{\mathcal{C}}_k \cdot \epsilon_n = 1 - \hat{\mathcal{C}}_k \tag{13}$$

The above set of equations (13) for all k in matrix form is: $\mathbf{C} \times \epsilon = \mathbb{C}$, which has a solution as long as determinant of $\mathbf{C}$ is not zero. We calculate the determinant by performing the following steps: (i) subtract first column values from all the other columns; and (ii) add all rows to the first row.

$$|\mathbf{C}| = (\sum_k \hat{\mathcal{C}}_k - 1) \cdot (\pm 1) \tag{14}$$

Therefore, determinant is zero only if $\sum_k \hat{\mathcal{C}}_k = 1$.[3]

Step 3: Closed form expression for ϵ_k can be verified by substituting its value from (12) in (13). The L.H.S. of (13) becomes:

$$\begin{aligned}
&\sum_{j \neq k} \hat{\mathcal{C}}_k \cdot \left[\frac{\hat{\mathcal{C}}_j \cdot (n-1)}{\sum_i \hat{\mathcal{C}}_i - 1} - 1\right] + (\hat{\mathcal{C}}_k - 1) \cdot \left[\frac{\hat{\mathcal{C}}_k \cdot (n-1)}{\sum_i \hat{\mathcal{C}}_i - 1} - 1\right] \\
&= \frac{\hat{\mathcal{C}}_k \cdot (n-1) \cdot [\sum_i \hat{\mathcal{C}}_i - 1]}{\sum_i \hat{\mathcal{C}}_i - 1} - n \cdot \hat{\mathcal{C}}_k + 1 \\
&= 1 - \hat{\mathcal{C}}_k
\end{aligned}$$

This is the R.H.S. of (13) and hence verified. ■

For local minimum bounds, we allocate each entity its minimum bounds and solve the problem for the remaining value (1 - sum of minimum bounds) by normalizing the bounds. For instance, in a problem with 3 entities with minimum bounds as (0.1, 0.1, 0.1) and maximum bounds as (0.4, 0.5, 0.6). We convert it to a problem with minimum bounds as (0, 0, 0) and maximum bounds as (0.3/0.7, 0.4/0.7, 0.5/0.7). When there are both local minimum and maximum bounds, the expression for z_k is:

$$z_k(\vec{y}, \{\check{\mathcal{C}}_i\}, \{\hat{\mathcal{C}}_i\}) = \check{\mathcal{C}}_k + (1 - \sum_j \check{\mathcal{C}}_j) \cdot z_k(\vec{y}, \vec{0}, \{\frac{\hat{\mathcal{C}}_i - \check{\mathcal{C}}_i}{1 - \sum_j \check{\mathcal{C}}_j}\}) \tag{15}$$

Therefore, we can use Proposition 1 to compute the expression for minimum bounds as well.

In general, if the sum constraint is $\mathcal{C}$, then the expression for z_k is:

$$\begin{aligned}
z_k(\vec{y}, \{\check{\mathcal{C}}_i\}, \{\hat{\mathcal{C}}_i\}, \mathcal{C}) &= \check{\mathcal{C}}_k \\
&+ (\mathcal{C} - \sum_j \check{\mathcal{C}}_j) \cdot z_k(\vec{y}, \vec{0}, \{\frac{\hat{\mathcal{C}}_i - \check{\mathcal{C}}_i}{\mathcal{C} - \sum_j \check{\mathcal{C}}_j}\})
\end{aligned} \tag{16}$$

[3]When $\sum_k \hat{\mathcal{C}}_k = 1$, then we assign $z_k = \hat{\mathcal{C}}_k$, as that is the limit of z_k in (12) as $\sum_k \hat{\mathcal{C}}_k \to 1^+$.

Limitation of Constrained Softmax The proof of proposition (1) requires that $\forall k : \epsilon_k \geq 0$. i.e.

$$\forall k : \frac{\widehat{\mathcal{C}}_k \cdot (n-1)}{\sum_i \widehat{\mathcal{C}}_i - 1} - 1 \geq 0 \tag{17}$$

An example when this condition will be violated is when $\{\widehat{\mathcal{C}}\} = (0.3, 0.5, 0.6)$. Therefore, the applicability of constrained softmax is limited to the cases where (17) is satisfied.

4.3 OptLayer

One way to address different constraints on the actions is to project the output of the actor neural network to the feasible space of actions. This can be done using OptLayer (Pham, De Magistris, and Tachibana 2018). Assume that $\vec{y}$ is the output of the actor network with some activation function, which may not satisfy all the constraints in our domain. We can project $\vec{y}$ to the feasible space and get the feasible action $\vec{z}$ by solving the following QP:

$$\begin{aligned} \min_{\vec{z}} \sum_{k=1}^{n} (z_k - y_k)^2 \quad & \text{subject to} \\ \sum_{k=1}^{n} z_k - \mathcal{C} = 0 \qquad & : \lambda \\ \forall k = 1..n : z_k - \widehat{\mathcal{C}}_k \leq 0 \qquad & : \alpha_i \\ \forall k = 1..n : \check{\mathcal{C}}_k - z_k \leq 0 \qquad & : \beta_i \end{aligned} \tag{18}$$

Here $\lambda, \alpha_i, \beta_i$ are the corresponding Lagrange multipliers. Since the objective function is *strictly* convex and the constraints are linear, there exists a unique solution to this QP. The Lagrangian function is (Bertsekas 1999):

$$\begin{aligned} L(\vec{z}, \vec{\alpha}, \vec{\beta}, \lambda) = & \sum_k (z_k - y_k)^2 + \lambda(\sum_k z_k - \mathcal{C}) \\ & + \sum_k \alpha_k (z_k - \widehat{\mathcal{C}}_k) + \sum_k \beta_k (\check{\mathcal{C}}_k - z_k) \end{aligned} \tag{19}$$

The KKT conditions (conditions satisfied by the optimal solution $\vec{z}^\star, \vec{\alpha}^\star, \vec{\beta}^\star, \lambda^\star$) of the QP (18) are given by $\nabla_{\vec{z},\lambda} L = \vec{0}$, along with the equations of complementary slackness:

$$\begin{cases} \sum_k^n z_k^\star - \mathcal{C} & = 0 \\ \forall k = 1..n : \quad 2(z_k^\star - y_k) + \lambda^\star + \alpha_k^\star - \beta_k^\star & = 0 \\ \forall k = 1..n : \quad \alpha_k^\star (z_k^\star - \widehat{\mathcal{C}}_k) & = 0 \\ \forall k = 1..n : \quad \beta_k^\star (\check{\mathcal{C}}_k - z_k^\star) & = 0 \end{cases} \tag{20}$$

To include this layer in the end-to-end backpropagation training of the actor, in the backward pass, we need to compute the gradients of the optimal solution of the QP w.r.t. the inputs $\vec{y}$ or $\mathbf{J}_{kj} = \frac{\partial z_k^\star}{\partial y_j}$. Such gradients can be computed using the implicit function theorem as shown in (Amos and Kolter 2017). The technique yields a system of linear equations in partials $\frac{\partial z_k^\star}{\partial y_j}$, which can be solved to find the Jacobian matrix $\mathbf{J}$. Thus in this method, the forward pass requires solving a QP using an optimizer and the backward pass involves solving the set of linear equations for each item in the sampled minibatch.

4.4 Approximate OptLayer algorithm

The main challenge in solving (18) in the forward pass is that it becomes computationally slow given the large number of iterations most RL approaches require. Therefore, we next propose an approximate algorithm to solving the QP (18), which apart from being very efficient, makes computing the gradients in the backward pass computationally much faster. The motivation for our proposed approach comes from similar iterative approaches that have been used to solve QPs, but in different contexts such as graphical models (Kumar and Zilberstein 2011; Duchi et al. 2008).

Before we describe the algorithm, we impose the condition that the output $y_i \forall i$ of the neural network always satisfies respective upper/lower bounds or $\check{\mathcal{C}}_k \leq y_k \leq \widehat{\mathcal{C}}_k \ \forall k$. This condition is easy to enforce in a neural network. Assume $\vec{x} = \mu(s_t|\theta^\mu)$ is any arbitrary vector output of the network, with no activation function used in the final layer. We use the activation function:

$$y_k = \check{\mathcal{C}}_k + (\widehat{\mathcal{C}}_k - \check{\mathcal{C}}_k) \frac{x_k - \min(\vec{x})}{\max(\vec{x}) - \min(\vec{x})} \tag{21}$$

where min, max provide minimum, maximum component of the vector respectively. We do this scaling only if any x_k violates its corresponding bounds. Else, we set $y_k = x_k \ \forall k$.

QP (18) computes the optimal L_2 projection of $\vec{y}$ onto the feasible space determined by the constraints. The main insight behind this approach is that we do not need the optimal L_2 projection of $\vec{y}$. We just need *any differentiable projection* function which can respect the constraints. Empirically, the projection derived by our proposed approach is not very far from the optimal L_2 projection.

Initially, we find a closed-form solution to the QP (18) by ignoring the inequality constraints. That is to say, we solve the KKT conditions (20) by setting $\vec{\alpha}^* = \vec{0}$ and $\vec{\beta}^* = \vec{0}$ and get:

$$\forall k : z_k = y_k + \frac{\mathcal{C} - \sum_{k=1}^n y_k}{n} \tag{22}$$

After applying this formula, if all the z_k are found to be satisfying the bound constraints $\check{\mathcal{C}}_k \leq z_k \leq \widehat{\mathcal{C}}_k$, then the algorithm terminates. We compute the gradients ($\frac{\partial z_k}{\partial y_j}$) by differentiating (22):

$$\frac{\partial z_k}{\partial y_j} = \delta_{kj} - \frac{1}{n} \tag{23}$$

where δ_{kj} is 1 if $k = j$, else 0.

If the bound constraints are not satisfied for some z_k, the algorithm proceeds to correct $\vec{z}$ in two phases. The first phase, referred to as LOWER, is to satisfy the min constraints, and the second phase, referred to as UPPER, is to satisfy max constraints. In the LOWER phase, all the outputs z_k which are *below* the respective min bounds are clamped to the respective min bound $\check{\mathcal{C}}_k$. The remaining value $\mathcal{C}'$ ($= \mathcal{C}$ minus sum of clamped outputs) is redistributed over the remaining n' ($= n$ minus number of clamped outputs) outputs using an expression similar to (22), but with new $\mathcal{C} = \mathcal{C}', n = n'$ and involving only the y_k corresponding to the unclamped z_k. The LOWER phase loops until there is no need for anymore clamping i.e. it ends when $\forall k : z_k \geq \check{\mathcal{C}}_k$.

Algorithm 1 Forward pass and gradient computation for ApprOpt layer

```
Require: $n \geq 2$
Require: $\forall k = 1..n : 0 \leq \check{\mathcal{C}}_k < \widehat{\mathcal{C}}_k \leq \mathcal{C}$
Require: $\sum \check{\mathcal{C}}_k < \mathcal{C} < \sum \widehat{\mathcal{C}}_k$
Require: $\forall k : \check{\mathcal{C}}_k \leq y_k \leq \widehat{\mathcal{C}}_k$
 1: $n' \leftarrow n$                    ▷ count of unclamped indices
 2: $\mathcal{C}' \leftarrow \mathcal{C}$          ▷ value to distribute to unclamped indices
 3: $\Omega \leftarrow \{1, 2, ..., n\}$            ▷ unclamped output indices
 4: phase ← LOWER                 ▷ possible values are: LOWER $= 0$; UPPER $= 1$; DONE $= 2$
 5: while phase $\neq$ DONE do
 6:     $\Omega' \leftarrow \phi$    ▷ indices clamped in this iteration of the while loop
 7:     $z_k \leftarrow y_k + (\mathcal{C}' - \sum_{j\in\Omega} y_j)/n'$ foreach $k \in \Omega$
 8:     $\mathbf{J}_{kj} \leftarrow \delta_{kj} - 1/n'$ foreach $j \in \Omega$, foreach $k \in \Omega$
 9:     $\mathbf{J}_{kj} \leftarrow 0$ foreach $j \notin \Omega$, foreach $k = 1..n$
10:     for $k \in \Omega$ do
11:         if $z_k < \check{\mathcal{C}}_k$ and phase = LOWER then
12:             $z_k \leftarrow \check{\mathcal{C}}_k$
13:             $\mathbf{J}_{kj} \leftarrow 0$ foreach $j = 1..n$
14:             $\Omega' \leftarrow \Omega' \cup \{k\}$
15:         else if $z_k > \widehat{\mathcal{C}}_k$ and phase = UPPER then
16:             $z_k \leftarrow \widehat{\mathcal{C}}_k$
17:             $\mathbf{J}_{kj} \leftarrow 0$ foreach $j = 1..n$
18:             $\Omega' \leftarrow \Omega' \cup \{k\}$
19:         end if
20:     end for
21:     $n' \leftarrow n' - |\Omega'|$
22:     $\mathcal{C}' \leftarrow \mathcal{C}' - \sum_{k\in\Omega'} z_k$
23:     $\Omega \leftarrow \Omega - \Omega'$
24:     if $\Omega' = \phi$ then
25:         phase = phase + 1
26:     end if
27: end while
28: return $\vec{z}, \mathbf{J}$
Ensure: $\forall k : \check{\mathcal{C}}_k \leq z_k \leq \widehat{\mathcal{C}}_k$
Ensure: $\sum_{k=1}^{n} z_k = \mathcal{C}$
Ensure: $z = y$ if $\sum_{k=1}^{n} y_k = \mathcal{C}$
```

After the LOWER phase is over, the UPPER phase begins, which similarly repeatedly clamps the outputs which violate the respective max constraints and redistributes the remaining value.

During the entire process, if a z_k is clamped either to its upper or lower bound, it remains clamped to the same value for all the future iterations and remains out of the clamp-and-redistribute loop along with the corresponding y_k. Thus z_k is assigned to a constant and not affected by any input, and y_k does not affect any output. Thus at the end of the algorithm, we have:

$$\mathbf{J}_{kj} = \frac{\partial z_k}{\partial y_j} = \begin{cases} 0 & \text{if } z_k \text{ or } z_j \text{ is clamped} \\ \delta_{kj} - \frac{1}{n'} & \text{otherwise} \end{cases} \tag{24}$$

Algorithm 1 provides the pseudocode.

Proof for Algorithm 1: There are five steps to proving that the algorithm terminates and always gives a feasible output.
Step 1: In LOWER phase, every iteration of the while loop leaves at least one output z_k unclamped. This can be proven by contradiction.

Assume that after line 7 we have: $\forall k \in \Omega : z_k < \check{\mathcal{C}}_k$

$$\implies \forall k \in \Omega : y_k + \frac{\mathcal{C}' - \sum_{j\in\Omega} y_j}{n'} < \check{\mathcal{C}}_k$$
$$\implies \sum_{k\in\Omega} y_k + \mathcal{C}' - \sum_{j\in\Omega} y_j < \sum_{k\in\Omega} \check{\mathcal{C}}_k$$
$$\implies \mathcal{C}' < \sum_{k\in\Omega} \check{\mathcal{C}}_k$$
$$\implies \mathcal{C} - \sum_{k\notin\Omega} \check{\mathcal{C}}_k < \sum_{k\in\Omega} \check{\mathcal{C}}_k$$
$$\implies \mathcal{C} < \sum_{k=1}^{n} \check{\mathcal{C}}_k$$

This contradicts with given precondition that $\mathcal{C} \geq \sum_{k=1}^{n} \check{\mathcal{C}}_k$ from (3). Thus, $\exists k$ s.t $z_k \geq \check{\mathcal{C}}_k$, which would not get clamped.
Step 2: LOWER phase always terminates with not having excluded all feasible solutions. With every iteration of the while loop in the phase, the count of unclamped outputs i.e. n' monotonically decreases, due to line 21. But it cannot decrease to less than 1 in any iteration, by step 1. Thus it must terminate with $n' \geq 1$. After the LOWER phase terminates, we also need to show that the remaining value to distribute, $\mathcal{C}'$, is less than or equal to the sum of upper bounds on the unclamped outputs i.e. $\mathcal{C}' \leq \sum_{j\in\Omega} \widehat{\mathcal{C}}_j$. If this condition is not maintained then the remaining value cannot be feasibly distributed over $z_k, k \in \Omega$. We will prove that the condition is maintained after every iteration in LOWER phase because clamping of any z_k preserves the condition. Output z_k is clamped if

$$z_k < \check{\mathcal{C}}_k$$

Also, we observe that before clamping, i.e. after line 7, z_k is minimum if y_k is minimum, i.e. $\check{\mathcal{C}}_k$, and y_j $\forall j \in \Omega - \{k\}$ are maximum, i.e. $\widehat{\mathcal{C}}_j$.[4] Thus,

$$\check{\mathcal{C}}_k + \frac{\mathcal{C}' - \check{\mathcal{C}}_k - \sum_{j\in\Omega-\{k\}} \widehat{\mathcal{C}}_j}{n'} \leq z_k < \check{\mathcal{C}}_k$$
$$\implies \check{\mathcal{C}}_k + \frac{\mathcal{C}' - \check{\mathcal{C}}_k - \sum_{j\in\Omega-\{k\}} \widehat{\mathcal{C}}_j}{n'} < \check{\mathcal{C}}_k$$
$$\implies \mathcal{C}' - \check{\mathcal{C}}_k < \sum_{j\in\Omega-\{k\}} \widehat{\mathcal{C}}_j$$

which means that the remaining value to distribute after clamping z_k, i.e. $\mathcal{C}' - \check{\mathcal{C}}_k$ will be less than the sum of remaining upper bounds, i.e. $\sum_{j\in\Omega-\{k\}} \widehat{\mathcal{C}}_j$.
Step 3: In UPPER phase, every iteration of the while loop always leaves at least one output z_k unclamped. This can be proven by contradiction using the same tactics as step 1.
Step 4: UPPER phase always terminates: With every iteration in the phase, the count of unclamped outputs i.e. n' monotonically decreases. But it cannot decrease to less than 1 in any iteration, by step 3. Thus it must terminate with $n' \geq 1$.

[4]This is true because $\frac{\partial z_k}{\partial y_k} > 0$ and $\frac{\partial z_k}{\partial y_{j\neq k}} < 0$ in line 7 of Algorithm 1

Step 5: By step 4, the while loop terminates with $n' \geq 1$, i.e. at least one unclamped output. On termination of the while loop, all the outputs must satisfy the min and max constraints, since the LOWER and UPPER phase have ended. The sum of the unclamped outputs is $\mathcal{C}'$ (by summing the expression in line 7 over $k \in \Omega$), and $\mathcal{C}' = \mathcal{C} - \sum_{k \notin \Omega} z_k =$ ($\mathcal{C}$ minus sum of clamped outputs). Thus clamped and unclamped outputs add up to $\mathcal{C}$.

Thus the algorithm provides a feasible solution that satisfies all the constraints. ■

Whenever the input is already feasible, i.e. $\sum_k y_k = \mathcal{C}$ and $\check{\mathcal{C}}_k \leq y_k \leq \hat{\mathcal{C}}_k$, then $z_k = y_k$ by equation (22) i.e. the algorithm behaves as an identity function for feasible inputs. Thus there exist a set of inputs such that the corresponding set of outputs occupies the entire feasible solution space. Thus this approach does not preclude any feasible solution.

Computing gradients: Notice that Algorithm 1 computes all the required gradients $\mathbf{J}_{kj} = \frac{\partial z_k}{\partial y_j}$ in lines 8, 9, 13, and 17. There is no expensive matrix inversion or solving of a system of linear equation required. This provides significant speedup over the exact solving of QP using CPLEX and then computing the gradients as in the standard OptLayer.

Practical Considerations: (Pham, De Magistris, and Tachibana 2018) found that using only the OptLayer gradient does not lead to efficient learning. We found the same problem with the ApprOptLayer gradient. We suspect that this is because when $\vec{x}$ is very far from feasible, it leads to many z_k getting clamped, resulting in many zero gradients. Conversely, if $\vec{x}$ is close to feasible, then the mapping from $\vec{x}$ to $\vec{z}$ is close to identity, resulting in near-unit gradients. Hence, to get $\vec{x}$ close to feasible, we penalize the actor output $\vec{x}$ for being infeasible. This is unlike the reward shaping used in (Pham, De Magistris, and Tachibana 2018). Our penalty term $\nu(\vec{x})$ (as defined in (6)) is added to the actor's training objective, like in DDPG-CP. This penalization method gives direct gradient information to the actor network, as opposed to the critic learning the penalty returns first and then passing the gradient information back to the actor network. Even though we use a penalty in the actor's objective, this approach is still superior to DDPG-CP, since it being an end to end learning approach, the policy gradient feedback from the critic is propagated via the ApprOpt layer, making it theoretically more justified.

4.5 Hierarchical Regional Constraints

For DDPG-CS and DDPG-ApprOpt, we have shown how to handle local constraints by adding a differentiable projection layer at the end of the actor network. We will now show how these local constraints handling layers (or LCHLs) can be nested to create a differentiable module for handling hierarchical regional constraints. For example, suppose there are two regions G_1 and G_2 and five entities. Say $G_1 = \{1, 2, 3\}$ and $G_2 = \{4, 5\}$. We approach this example problem as follows. Let the actor neural network output seven pins (instead of five): $x_1, x_2, ..., x_5, x_{G_1}, x_{G_2}$, activated to $y_1, y_2, ..., y_5, y_{G_1}, y_{G_2}$ using the appropriate activation function (10) or (21) depending on whether we are using CS or ApprOpt. Then we use a LCHL to determine the allocations for G_1, G_2 first i.e z_{G_1}, z_{G_2} using inputs y_{G_1}, y_{G_2} with sum constraint as 1 and (given) bound constraints as $\check{\mathcal{C}}_{G_1}, \check{\mathcal{C}}_{G_2}, \hat{\mathcal{C}}_{G_1}, \hat{\mathcal{C}}_{G_1}$. Then we use another LCHL to determine z_4, z_5 using inputs y_4, y_5 with sum constraint as z_{G_2} and bound constraints as $\check{\mathcal{C}}_4, \check{\mathcal{C}}_5, \hat{\mathcal{C}}_4, \hat{\mathcal{C}}_5$. Similarly, another LCHL would compute z_1, z_2, z_3 using inputs y_1, y_2, y_3 with sum constraint as z_{G_1}.

To make nested-LCHLs end-to-end differentiable, it is required that the LCHLs, for which the sum constraint input is an output of another LCHL, should compute the gradients of its outputs w.r.t the sum constraint as well.

For CS layer, this is handled automatically by auto-differentiating frameworks such as TensorFlow, as (16) is a closed form expression w.r.t. $\mathcal{C}$.

For ApprOpt layer, $\frac{\partial z_k}{\partial \mathcal{C}} = 0$ for clamped outputs ($k \notin \Omega$) because clamped outputs are assigned to constants. For unclamped outputs, i.e. $k \in \Omega$,

$$\begin{aligned}
\frac{\partial z_k}{\partial \mathcal{C}} &= \frac{\partial y_k}{\partial \mathcal{C}} + \frac{1}{n'} \cdot \Big(\frac{\partial \mathcal{C}'}{\partial \mathcal{C}} - \sum_{j \in \Omega} \frac{\partial y_j}{\partial \mathcal{C}}\Big) && \text{(From line 7 of Alg 1)} \\
&= 0 + \frac{1}{n'} \cdot \Big(\frac{\partial \mathcal{C}'}{\partial \mathcal{C}} - 0\Big) && \text{(input } \vec{y} \text{ independent of } \mathcal{C}) \\
&= \frac{1}{n'} \cdot \frac{\partial(\mathcal{C} - \sum_{k \notin \Omega} z_k)}{\partial \mathcal{C}} && \text{(by def of } \mathcal{C}') \\
&= \frac{1}{n'} \cdot \frac{\partial \mathcal{C}}{\partial \mathcal{C}} && \Big(\tfrac{\partial z_k}{\partial \mathcal{C}} = 0 \text{ for } k \notin \Omega\Big) \\
&= \frac{1}{n'} && (25)
\end{aligned}$$

5 Experiments

Our goal in the experiments is to evaluate the performance of our new approaches, DDPG-CP, DDPG-CS and DDPG-ApprOpt in comparison to baseline approaches. DDPG-OptLayer was at least an order of magnitude slower and on the limited computation resources available to us, we were unable to evaluate it completely. Within the limited evaluations, we observed that DDPG-ApprOpt was on par with DDPG-OptLayer.

We train and evaluate our approaches on two simulators[5] related to emergency response and bike sharing that have inherent constraints.

Emergency Response System (ERS): First, we consider a simulator for emergency response with the transitional dynamics inspired by (Yue, Marla, and Krishnan 2012). In our simulator, there are 32 ambulances to be distributed among 25 base stations. We define a zone (corresponding to a base station) as the points which are closest to that base than any other base. Thus $m = n$. All vehicles travel in straight lines at a uniform speed. As requests (incidents) arrive according to the patterns described in the next paragraph, they are added to a request queue, to be served on a first-come-first-serve basis. For a given request, an ambulance is dispatched from the nearest base station which has an idle ambulance. If

[5]The OpenAI Gym environments for the two simulators are available at: *https://github.com/bhatiaabhinav/gym-ERSLE* (ERS) and *https://github.com/bhatiaabhinav/gym-BSS* (BS)

there are no idle ambulances, the request is not served until an ambulance gets free. An ambulance is considered busy throughout its life cycle which consists of picking the patient, dropping them off to the nearest hospital, and coming back to the base assigned to it. It might come back to a different base than its starting base if it was reassigned to a new base before leaving from the hospital. When an idle ambulance is reassigned to a new base, it is considered unavailable to serve requests during its journey to the new base. Thus there is an implicit cost to reallocations.

One episode corresponds to one day. The episode starts with a uniform allocation. Reallocation is done every 30 minutes i.e. the MDP frame consists of running the simulator for 30 simulated minutes, and returning the new observation and a reward signal. The observation consists of: the allocation at the end of that frame, the number of new requests per zone during that frame, and the time of the day. The reward signal is the number of request sites at which an ambulance reached within 10 minutes during that frame. The RL system is expected to act by providing a 25-dimensional target-allocation. The simulator achieves the target allocation in a way such that it minimizes the number of ambulances which are assigned to a new base.

For our experiments, we consider 2 demand patterns: "Singapore Poisson" (in which the probability of an incident at a place is a function of the time of the day, inspired by actual demand statistics in Singapore 2011 (Ji et al. 2013)), and a "random surges" version in which we introduce unpredictable random surges in demand at random places at random times, once per episode. The surges are gaussian in nature. We consider uniform local constraints for this environment.

Bike Sharing (BS): We consider the simulator described in (Ghosh and Varakantham 2017) for Hubway bike sharing system. The bike sharing system consists of 95 base stations (zones) and 760 bikes. The planning horizon (episode) is of 6 hours in the morning peak (6AM-12PM) and the duration of each decision epoch is considered as 30 minutes. The observation consists of: the distribution of bikes at the end of the frame, the cumulative demand per zone during the 30 minute frame, and the time of the day. The RL agent is expected to act by providing a target allocation every 30 minutes. The simulator may not be able to achieve this allocation at the end of the next 30 minutes, since in this domain, the allocation is influenced also by the customers picking up bikes at one zone and leaving them at different zones.

For our experiments we consider the first demand data set from (Ghosh and Varakantham 2017), which corresponds to a slightly modified real demand data for 60 weekdays. Like in that work, the dataset is divided as 20 days for training and 40 days for testing. Corresponding to these, we create a training MDP and a testing MDP, which use training data and testing data respectively. There are non-uniform local constraints in this environment.

Baselines: For ERS domain, we compare our approaches against static allocation baselines, computed offline using a greedy allocation algorithm, described in (Yue, Marla, and Krishnan 2012). The optimization was done over a fixed set of 32 episodes, initialized with seeds 0 through 31.

Briefly, in this algorithm, the best base station is decided for each ambulance one by one. Each ambulance is tried on every base station for the entire episode, and finally the base station which results in maximum average gain (average over the 32 random seeds) due to that ambulance is chosen. Since the maximum average gain decreases as more ambulances are assigned an appropriate base, i.e. the domain is (close to) submodular (Manohar, Varakantham, and Lau 2018), this myopic algorithm yields an allocation close to optimal. We did not use a dynamic greedy repositioning baseline (i.e. use this algorithm to find an offline allocation for every frame, as opposed to the entire episode), partly due to the computational requirements (25x32x32x48 simulations), and partly due to the fact that any gains from dynamic repositioning are largely dampened by the cost of reallocation. Further, dynamic greedy repositioning would be greedy in time as well, i.e. it might do a re-allocation which is good in short term, but bad in long term. Static positioning does very well assuming an average demand statistics across the day for each place. As we will see later, RL too performs similar to static repositioning when the demand statistics are predictable/Poisson. When they are not predictable, i.e. in the domain instances with random surges at random places, greedy baselines are expected to not work well, since the Poisson assumption made in the (Yue, Marla, and Krishnan 2012) (that the probability of a request at a given place and time is independent of other requests) breaks. Greedy algorithm cannot consider constraints while calculating the allocation and hence the results from baselines might appear stronger. It should be noted that since we consider allocation problems, we cannot employ planning methods like the one by (Dayapule et al. 2018), where the focus is on planning the dispatch rather than allocation.

For bike sharing domain, we take the best results from (Ghosh and Varakantham 2017), which correspond to "RTrailer": a dynamic repositioning algorithm using 10 bike trailers, each having a capacity of 5. The work by (Ghosh and Varakantham 2017) has demonstrated that with small trailers, they get better performance than with 3 larger trucks with a capacity of 20. We took their best setting (10 trailers with 5 capacity) when comparing against our approach. Successive allocations recommended by our approach requires a maximum movement of bikes that is 10 (much lower than large vehicle capacity of 20).

In summary, the big advantage with our approach is that it is domain independent and we do not need a domain specific approach (like greedy or optimization). It could be used for replenishing vending machines serving fresh juices, or for food delivery etc. and can handle unpredictable events which can be handled only online.

Algorithmic Details: For the actor and critic deep learning networks, we largely retain the basic architecture used in (Lillicrap et al. 2015). Both the actor and the critic use two hidden layers with 128, 96 ReLU units for emergency response domain and 400,300 ReLU units for bike sharing domain. For the critic, actions are not included until the second hidden layer. Layer normalization (Ba, Kiros, and Hinton 2016) is applied before non-linearities to all hidden layers in both the actor and the critic. Weights are initialized using glorot uniform initializer (scale=1.0) (Glorot and

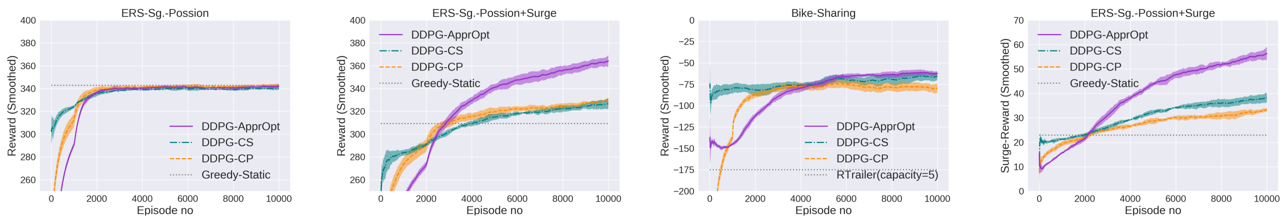

Figure 2: Learning curves comparing different approaches. Each curve shows average reward per episode $\pm$ standard deviation over different seeds. The last subfigure shows the average reward *due to the incidents which were part of the surges*.

Domain (Demand pattern)	Baseline	DDPG-CP	DDPG-CS	DDPG-ApprOpt
ERS Poisson	342.9	342.09 ± 0.35	339.42 ± 1.35	341.16 ± 0.84
ERS Poisson+Surge	309.3	316.46 ± 1.38	312.32 ± 3.59	353.20 ± 4.17
BS	-175	-92.86 ± 5.69	-77.64 ± 3.02	-90.3 ± 11.2

Table 1: Average evaluation scores $\pm$ standard deviation over different seeds. The baseline approaches are Greedy-static for Emergency Response System (ERS) domain and RTrailer for Bike Sharing (BS) domain.

Bengio 2010) for hidden layers and using orthogonal initializer (gain=1.0) (Saxe, McClelland, and Ganguli 2013) for final layers. The parameters of the target network are soft updated from those of the main network after every training step with $\tau = 10^{-3}$. The parameters of the main network are trained using Adam optimizer (Kingma and Ba 2014) at a learning rate of 10^{-3} for the critic and 10^{-4} for the actor. The parameters of the critic (except those of the final layer and the biases) are regularized with an L_2 norm penalty of 10^{-2} for emergency response domain and 10^{-1} for bike sharing domain. A gradient descent step is performed every 2 frames, in which a minibatch of size 128 is randomly sampled from an experience replay buffer of size 10^6. Adaptive parameter noise (Plappert et al. 2017) is used for exploration with a target divergence of $1/C$, and adaptation factor of 1.05.

To make the environment more observable, the most recent observation is modified to include the demand statistics from the previous two frames as well. The observation input to the actor and the critic are normalized with the running mean and standard deviation of the observations. Additionally, the action input to the critic is also normalized with a running mean and standard deviation of the actions.

For DDPG-CP, we use $\lambda = 10^3$ for emergency response domain and $\lambda = 10^5$ for bike sharing domain. For DDPG-ApprOpt, we use penalty term coefficient of 10^3 for emergency response Domain and 10^4 for bike sharing.

Training and Evaluation: Each experiment consisted of training on the corresponding environment 5 times for 10,000 episodes using random seeds=0..4 to initialize the environment and model parameters. During the training, every 4th episode was played without exploration. These exploitative episodes were used to generate the learning curves. Each learning curve shown in figure 2 shows the mean and standard deviation of the smoothed individual learning curves across the random seeds. For evaluating a trained model, its average score was taken across 100 test episodes without exploration (simulator initialized with random seed=42). The final evaluation score for an experiment was calculated as the average score of the 5 trained models corresponding to it.

5.1 Results and Discussion

A summary of the results is presented in Table 1. Figure 2 shows the learning curves for emergency response and bike sharing domains in different scenarios.

The scores for the emergency response domain instance with Poisson demand pattern reflect that all three of our approaches, DDPG-CS, DDPG-CP and DDPG-ApprOpt are competitive to the greedy baseline. This is in spite of greedy providing near optimal solutions in this domain as the environment is stationary. Additionally, greedy does not consider the bound constraints and hence the results are an upper bound on the actual solution that is achievable with constraints.

All three approaches outperform the baseline in the emergency response domain instance with random surges. As is clear from the last subfigure of Figure 2, the difference is mainly due to RL performing much better during the surges period in the episodes. This is owing to the unpredictability of the surges, which cannot be handled by an offline and static approach. DDPG-ApprOpt is particularly good in handling surges and is able to very significantly outperform all other approaches. DDPG-CS and DDPG-CP had roughly the same performance with DDPG-CP having the slight edge.

On the bike sharing domain with real data, there is considerable unpredictability in demand and so RL vastly outperforms the baseline RTrailer, which once again being an offline approach cannot handle unpredictability so well. Also, it needs to be noted that all the RL approaches generalized very well to the test environment, which uses different data than the training environment. While DDPG-ApprOpt performed best in the training environment, DDPG-CS generalized better and came out on top on in the test environment.

In summary, the results suggest that:

- DDPG-ApprOpt clearly produced the best learning curves on all the benchmark problems. However, on bike sharing domain, it did not generalize to test environment as well as DDPG-CS.
- DDPG-CS had the best test score on bike sharing domain.

But this approach has the disadvantage of being limited to the cases where $\forall k, \epsilon_k > 0$.

- Finally, while DDPG-CP never wins clearly over the other two approaches, it is very general and easy to implement, which might make it desirable in some scenarios.

6 Conclusion

In summary, we have shown how RL can be used in online resource allocation problems where traditionally offline approaches or heuristics were used due to exponential action spaces and inability of RL to handle constraints. We presented three novel approaches based on DDPG for the same. We showed that specially in settings with non-Poisson demand patterns, RL has an important value due to its ability to have a reactive policy based on the situation. We backed up our claims with empirical evidence gathered by testing our approaches on simulators based on emergency response and bike sharing domains, using real or semi-real data. Each approach seemed to have its own pros and cons in terms of generality, efficiency and effectiveness.

7 Acknowledgements

This research was supported by the Singapore Ministry of Education Academic Research Fund (AcRF) Tier 2 grant under research grant MOE2016-T2-1-174.

References

Amos, B., and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. *CoRR* abs/1703.00443.

Ba, L. J.; Kiros, R.; and Hinton, G. E. 2016. Layer normalization. *CoRR* abs/1607.06450.

Bertsekas, D. 1999. *Nonlinear Programming*. Athena Scientific.

Dayapule, D. H.; Raghavan, A.; Tadepalli, P.; and Fern, A. 2018. Emergency response optimization using online hybrid planning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 4722–4728. International Joint Conferences on Artificial Intelligence Organization.

Duchi, J. C.; Shalev-Shwartz, S.; Singer, Y.; and Chandra, T. 2008. Efficient projections onto the l_1-ball for learning in high dimensions. In *International Conference on Machine Learning*, 272–279.

Dulac-Arnold, G.; Evans, R.; Sunehag, P.; and Coppin, B. 2015. Reinforcement learning in large discrete action spaces. *CoRR* abs/1512.07679.

Ghosh, S., and Varakantham, P. 2017. Incentivizing the use of bike trailers for dynamic repositioning in bike sharing systems. In *ICAPS 2017*, 373–381.

Ghosh, S.; Varakantham, P.; Adulyasak, Y.; and Jaillet, P. 2017. Dynamic repositioning to reduce lost demand in bike sharing systems. *Journal of Artificial Intelligence Research* 58:387–430.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W., and Titterington, M., eds., *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, 249–256. PMLR.

Ji, Z.; Wei, S. L. S.; Yng, N. Y.; and Hock, M. O. E. 2013. Designing effective ambulance deployment strategies – a retrospective study. http://www.singaporehealthcaremanagement.sg/Abstracts/Documents/PDFs/OP0031%20-%20Zhang%20Ji.pdf.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Kumar, A., and Zilberstein, S. 2011. Message-passing algorithms for quadratic programming formulations of MAP estimation. In *International Conference on Uncertainty in Artificial Intelligence*, 428–435.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971.

Lowalekar, M.; Varakantham, P.; Ghosh, S.; JENA, S. D.; and Jaillet, P. 2017. Online repositioning in bike sharing systems. AAAI.

Manohar, P.; Varakantham, P.; and Lau, H. C. 2018. Bounded rank optimization for effective and efficient emergency response. In *ICAPS 2018*, 375–382.

Mao, H.; Alizadeh, M.; Menache, I.; and Kandula, S. 2016. Resource management with deep reinforcement learning.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Pham, T.-H.; De Magistris, G.; and Tachibana, R. 2018. Optlayer - practical constrained optimization for deep reinforcement learning in the real world. *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *CoRR* abs/1706.01905.

Powell, W. B. 1996. A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. *Transportation Science* 30(3):195–219.

Saxe, A. M.; McClelland, J. L.; and Ganguli, S. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR* abs/1312.6120.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Yue, Y.; Marla, L.; and Krishnan, R. 2012. An efficient simulation-based approach to ambulance fleet allocation and dynamic redeployment. In *AAAI Conference on Artificial Intelligence*, 398–405.