Learning Resource Scheduling with High Priority Users using Deep Deterministic Policy Gradients

Steffen Gracla ⓘ, Edgar Beck ⓘ, Carsten Bockelmann ⓘ and Armin Dekorsy ⓘ

Dept. of Communications Engineering, University of Bremen, Bremen, Germany

Email: {gracla, beck, bockelmann, dekorsy}@ant.uni-bremen.de

Abstract

Advances in mobile communication capabilities open the door for closer integration of pre-hospital and in-hospital care processes. For example, medical specialists can be enabled to guide on-site paramedics and can, in turn, be supplied with live vitals or visuals. Consolidating such performance-critical applications with the highly complex workings of mobile communications requires solutions both reliable and efficient, yet easy to integrate with existing systems. This paper explores the application of Deep Deterministic Policy Gradient (DDPG) methods for learning a communications resource scheduling algorithm with special regards to priority users. Unlike the popular Deep-Q-Network methods, the DDPG is able to produce continuous-valued output. With light post-processing, the resulting scheduler is able to achieve high performance on a flexible sum-utility goal.

Index Terms

Deep Learning, Reinforcement Learning, Resource Allocation, 6G, Scheduling, eHealth

I. Introduction

The increased capabilities of mobile data transfer have opened opportunities to more tightly connect the chain of emergency care, e.g., transmitting video, vitals or specialist input directly at an emergency site. When time is of the essence, [1] show that a patients recovery chances may be improved significantly. Pilot projects such as [2] have taken first steps in the direction of harnessing this potential of communications technology, but highly integrated solutions are only just emerging. This is in part due to the substantial performance demands that medical applications require in, e.g., throughput, latency, and mean failure time. These demands are outliers even in light of the 5G NR specifications of 3GPP [3]. Serving medical use cases therefore requires new, tailored solutions that are able to deal with complex optimization problems, balancing heterogeneous performance and reliability demands.

Deep Learning (DL) methods have shown promising results dealing with highly complex tasks in a variety of fields including computer vision [4] and human speech recognition [5]. In these applications a data-driven learning

approach has shown strengths where classic, model-based algorithms are struggling; Where an explicit, adjustable model is not required or feasible, DL offers algorithms that are approximately optimal for given resources and expected input data range [6], [7]. However, data-driven approaches must be applied in ways that are mindful of their limitations. Typical gradient-based DL methods result in approximate, hard-to-interpret algorithms in which undesired behavior patterns are hard or impossible to adjust. For this reason, the applicability of DL for applications with strict performance demands requires further research, shown by, e.g., [8].

We consider a scenario where an Emergency Vehicle (EV) shares a communication medium with normal users. Through the use of intelligent resource scheduling, we can influence service quality for all users and ensure the proper handling of EV communications, though the question of QoS-optimal resource allocation for multiple traffic classes remains an open topic in research [9]. Other works, such as [10], [11], have applied Deep Q-Networks (DQN) to the task of intelligent resource allocation in communications with some success. While DQN are able to leverage the function approximation capabilities of deep neural networks, the basic DQN method restricts itself, by design, to discrete decision spaces. Some problems, such as allocating available resources by proportion, lend themselves more easily to continuous-valued formulations. In this paper, we investigate a Deep Deterministic Policy Gradient (DDPG) [12], [13] based approach to learning a flexible resource allocation algorithm, directly outputting the proportion of resources to be allocated to each user. In this way, the optimization process can be melded with the given problem more freely and easily compared to discrete action spaces. Using these Reinforcement Learning (RL) methods, we form a scheduling algorithm that is able to drastically improve priority user performance with a minimal drop in the overall utility.

II. Setup & Notation

This section introduces the simulation environment that schedulers will interact with, characterized by a resource grid, a job queue, and the communication link to users. We then formulate the optimization objective that motivates our DL-approach.

A. Simulation Environment

As depicted in Fig. 1, we assume a scheduler at a base station that manages a limited number U of discrete resource blocks, such as in an OFDM-system. In each discrete time step t, the scheduler divides the available discrete resources U among N connected users. According to the fraction of resources that user n is granted, the resources are then filled with jobs j from a job queue. For performance metric calculation, we define the set $\mathbb{J}[t]$ of all jobs in queue at time step t, and the set $\mathbb{J}_n[t]$ of the jobs assigned to user n. At the beginning of each time step t, every user n generates a new job j with a probability c_j, collected in sets $\mathbb{J}_{n,\,\mathrm{new}}[t]$. Jobs j have two attributes: a remaining request size $u_{j,\,\mathrm{req}}[t]$ in discrete resource blocks, and a time-to-timeout $v_j[t]$ in discrete time steps t. The jobs' initial values are set at generation, with the initial request size $u_{j,\,\mathrm{req}}[t] \sim \mathbb{U}[1,\, u_{n,\,\mathrm{max}}]$ drawn from a discrete uniform distribution, and the initial time-to-timeout $v_j[t] \leftarrow v_{n,\,\mathrm{init}}$ set to a fixed value $v_{n,\,\mathrm{init}}$. Both $u_{n,\,\mathrm{max}}$ and $v_{n,\,\mathrm{init}}$ are defined by a user-specific profile.
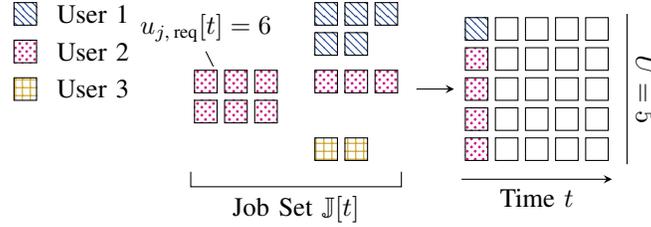
Fig. 1. At each time step t, a scheduler is tasked with distributing a number of U discrete resources among users. In a job queue, jobs j assigned to a user n are requesting a number $u_{j,\,\mathrm{req}}[t]$ of discrete resources to be completed. This example shows a system configuration with $U = 5$ managed resource blocks and $N = 3$ users.

Once resources the have been divided in a time step t, the appropriate amount of discrete blocks is deducted from the jobs' resource requests $u_{j,\,\mathrm{req}}[t]$ and all remaining jobs' time-to-timeout $v_j[t]$ is decremented. For record keeping, a lifetime count $u_{n,\,\mathrm{sx}}[t]$ of resources scheduled to a user n until time step t is increased by the appropriate amount. Should a job have remaining requests $u_{j,\,\mathrm{req}}[t] > 0$ when timing out, i.e., $v_j[t] = 0$, this job is removed from the queue and added to the set $\mathbb{J}_{\mathrm{fail}}[t]$ of jobs that timed out during time step t to be used in performance metric calculation.

At the beginning of a simulation episode, all user vehicles n are placed at a position drawn from a uniform random distribution centered on the base station position. The user distance $d_n[t]$ then varies over time due to vehicle movement, which is simulated akin to the Manhattan-model of movement [14]; At each time step t, user vehicles move a unit-size step in a random direction, with a $98\,\%$ probability of selecting their previous direction and otherwise uniform probability of turning left, right, or stopping. This results in a grid-like movement pattern. The communication channel between the base station and the users n is assumed as a Rayleigh-fading channel with additional distance-proportional path loss. Rayleigh-fading amplitudes $|\tilde{h}_n[t]| \sim \mathrm{Rayleigh}(\sigma_h)$ are drawn from a Rayleigh-distribution with a scale σ_h, while the path loss $\mathrm{PL}_n[t]$ is calculated according to the distance $d_n[t]$ between a user n and the base station at time step t,

$$\mathrm{PL}_n[t] = \min\left(1, (d_n[t])^{-1}\right). \tag{1}$$

While typical path loss models assume exponents of -2 or less, we opted for an exponent of -1 to reduce spread, thereby decreasing the computation required in the learning task ahead. Both path loss and Rayleigh-fading amplitude are combined into the power fading factor

$$h_n[t] = |\tilde{h}_n[t]| \cdot \mathrm{PL}_n[t]. \tag{2}$$

B. Problem Statement

The scheduler will be tasked with learning an allocation strategy that balances performance in three global metrics: (a) channel capacity, (b) timeouts, and (c) packet rate. These metrics will be defined subsequently. We also introduce a priority class of users, the EV-type, that requires preferential treatment by the scheduler.

(a) We define the *sum capacity* $r_C[t]$ at time step t for a given user power fading $h_n[t]$, signal power P and expected noise power σ_{noise}^2 as

$$r_C[t] = \sum_{n=1}^{N} \log\left(1 + h_n[t]\frac{P}{\sigma_{\text{noise}}^2}\right) = \sum_{n=1}^{N} \log\left(1 + \text{SNR}_n[t]\right). \tag{3}$$

Signal power P and expected noise power σ_{noise}^2 are assumed as fixed for all users.

(b) *Sum timeouts* $r_L[t]$ are defined as the sum of resources among the set $\mathbb{J}_{\text{fail}}[t]$ of all jobs that have timed-out at the end of time step t,

$$r_L[t] = \sum_{j \in \mathbb{J}_{\text{fail}}[t]} u_{j,\,\text{req}}[t]. \tag{4}$$

We also define the metric $r_{L,\text{EV}}[t]$ as the sum of resources lost from timeouts among EV-type users in time step t.

(c) The *sum packet rate* $r_P[t]$ is the ratio of lifetime resources scheduled to a user $u_{n,\,\text{sx}}[t]$, and the lifetime sum of resources requested by that user n. It is calculated as

$$r_P[t] = \sum_{n=1}^{N} \frac{u_{n,\,\text{sx}}[t]}{\sum_{\tau=1}^{t} \sum_{j \in \mathbb{J}_{n,\,\text{new}}[\tau]} u_{j,\,\text{req}}[\tau]} = \sum_{n=1}^{N} k_n[t]. \tag{5}$$

To judge a scheduler's overall performance, the aforementioned metrics are combined into a sum utility metric

$$r[t] = w_C r_C[t] - w_L r_L[t] - w_{L,\text{EV}} r_{L,\text{EV}}[t] + w_P r_P[t], \tag{6}$$

with individual tunable weights $w_C, w_L, w_{L,\text{EV}}, w_P$. The weights' purpose is two-fold: Primarily, they balance the submetrics against each other, as the individual submetrics are not normalized in regard to their expected magnitude. Secondarily, they allow us to communicate a preference or importance of the individual submetrics to the overall optimization process, such as in the case of EV-timeouts $r_{L,\text{EV}}[t]$.

III. DDPG Approach

We make use of deep RL in order to find an allocation algorithm that optimizes for a diverse set of goals, balancing multiple global metrics while also protecting priority users with its choice of allocation actions. An allocation action is a vector that specifies the percentage $A_n[t]$ of total available resources U that each user n should receive,

$$\mathbf{A}[t] = [A_1[t], \quad A_2[t], \quad \ldots, \quad A_N[t]]. \tag{7}$$

Central to this is a DDPG [12] actor-critic learning module. It consists of a *critic*, who learns to assess the quality of an allocation action according to the sum utility defined in (6), and an *actor* that learns to output allocation actions that satisfy the critic. Both actor and critic are modeled by standard fully connected deep neural networks, learning via variations on stochastic gradient descent from a memory buffer. During the learning process, an exploration module introduces noise to the allocation action to ensure that the action space is properly explored and the learning process has a rich and varied set of experiences to draw on. The learning module is bookended and assisted by pre- and post-processing blocks that embed the learning module input and output with the greater communication system, as well as ensuring valid allocation solutions. Fig. 2 displays the general process flow of the scheduler for one time step t, with the scheduling loop in solid lines on the outside, and the learning loop in thin blue lines on the inside. In the following, we will describe the modules in more detail.
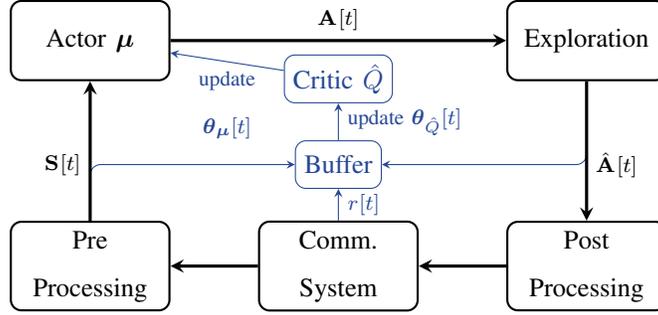
Fig. 2. The DDPG process flow consists of two main loops: The *scheduling loop* (outside, solid lines) handles the task of generating a valid allocation action, as well as gathering the system feedback to that allocation. Based on the experiences collected, the *learning loop* (blue, thin lines, on the inside) first tunes a critic $\hat{Q}$ that judges the goodness of an allocation action, and then uses the critic to update the actors behaviour.

A. The Scheduling Loop

We start with the preprocessor that summarizes the current state of the simulated communication system in a vector $\mathbf{S}[t]$ of fixed dimensionality, fit as an input to a neural network. For each user n, we extract 3 relevant features:

1) The current channel power fading

$$S_{n,1}[t] = h_n[t] \tag{8}$$

2) The sum of resource requests

$$S_{n,2}[t] = \sum_{j \in \mathbb{J}_n[t]} u_{j,\,\mathrm{req}}[t] \tag{9}$$

3) The inverse minimum time-to-timeout

$$S_{n,3}[t] = \left(\min_{j \in \mathbb{J}_n[t]} v_j[t] \right)^{-1} = (l_n[t])^{-1} \tag{10}$$

The resulting state vector $\mathbf{S}[t]$ is saved into a memory buffer as part of an experience, for later use in the learning loop.

For a given state $\mathbf{S}[t]$, the actor neural network is a function $\boldsymbol{\mu}(\mathbf{S}[t], \boldsymbol{\theta_\mu}[t])$ that outputs an allocation $\mathbf{A}[t]$ given its current parametrization $\boldsymbol{\theta_\mu}[t]$. To ensure that the output vector is normalized, i.e., $\sum_{n=1}^{N} A_n[t] = 1$, we use a softmax-activation for the last layer of the neural network. Initially, the parameters $\boldsymbol{\theta_\mu}[t]$ are randomized, and subsequently tuned to output "good" allocations during the learning loop. Next, an exploration module takes the network action $\mathbf{A}[t]$ and adds random distortion $\sigma_\mathrm{e} \cdot \tilde{A}_n$ to each entry $A_n[t]$, with $\tilde{A}_n \sim \mathrm{U}(-0.5,\, 0.5)$ drawn from a uniform distribution, and a noise magnitude multiplier σ_e that is decayed to 0 over the course of training. The resulting noisy allocation $\hat{\mathbf{A}}[t]$ is clipped to the valid range of $[0,\, 1]$, re-normalized, and added to the memory buffer to for learning.

Finally, the post-processing block represents a form of incorporating model-knowledge into the learning process by, e.g., limiting network output to plausible actions, filtering undesired behavior, or optimizing the output in some way that is known to be favorable. While this can relax the burden of the learning module to learn outputs that

are perfect, it also changes the learning objective from *finding the best allocation solution* to *finding the best post-processing input*. An overly zealous post-processor may lead to local optimum solutions that the DL process cannot escape from. Considering this, we opt for the following post-processing steps:

1) Zero out allocations to users with no requests (implausible)
2) Limit allocation to at most the amount requested per user (implausible)
3) Distribute at most one additional discrete resource to any user that had a fractional resource allocated, removing the need to learn perfect discrete mappings

During evaluation, when the learning module parameters are frozen and the scheduler is being tested for performance, we enable a further post-processing step:

4) Distribute remaining available resources by order of largest remaining request, to ensure no resource remains unused due to, e.g., rounding.

After processing, the action is forwarded to the communication system, where the available resources are distributed accordingly to jobs from the queue. One-by-one, requests from jobs are scheduled, starting with the jobs closest to timing out. The success in the metrics defined in Section II-B is calculated, and the simulation state updates to the next discrete time step t. The sum utility achieved by an allocation, according to (6), is also added to the memory buffer.

B. The Learning Loop

During each full scheduling loop iteration, the scheduler collects an experience tuple

$$\text{EXP} = (\mathbf{S}[t], \quad \mathbf{A}[t], \quad r[t]), \tag{11}$$

thereby building a data set of state-action combinations and their resulting benefit $r[t]$. The DDPG algorithm uses this data set to teach a critic neural network $\hat{Q}$, parametrized by $\boldsymbol{\theta}_{\hat{Q}}[t]$, to estimate the reward resulting from a state-action combination, i.e., $\hat{Q}(\mathbf{S}[t], \mathbf{A}[t], \boldsymbol{\theta}_{\hat{Q}}[t]) = \hat{r}[t]$. In each iteration of the learning loop, a mini-batch of B experiences is sampled from the experience buffer, and a mean squared estimation loss

$$\mathcal{L}_{\hat{Q}} = \sum_{b=1}^{B} (r_b[t] - \hat{r}_b[t])^2 \tag{12}$$

is evaluated on the samples b from this batch. Sampling is done prioritizing experiences that are new or diverge most from the estimation, as described in [15]. Using variants on mini-batch stochastic gradient descent, the critic parameters $\boldsymbol{\theta}_{\hat{Q}}[t]$ are then updated using the gradient $\nabla_{\boldsymbol{\theta}_{\hat{Q}}[t]} \mathcal{L}_{\hat{Q}}$ of this batch loss, thus nudging the critic estimate $\hat{r}[t]$ closer to the true sum utility $r[t]$.

The relevance of the critic lies in approximating the unknown system dynamics that lead from allocation action to metric success, making them tractable as the concrete parametrized function $\hat{Q}(\mathbf{S}[t], \mathbf{A}[t], \boldsymbol{\theta}_{\hat{Q}}[t])$. It is used to update the actors parameters $\boldsymbol{\theta}_{\boldsymbol{\mu}}[t]$ by the same principle. A loss function

$$\mathcal{L}_{\boldsymbol{\mu}} = -\sum_{b=1}^{B} \hat{Q}\left(\mathbf{S}_b[t], \ \boldsymbol{\mu}\left(\mathbf{S}_b[t], \boldsymbol{\theta}_{\boldsymbol{\mu}}[t]\right), \ \boldsymbol{\theta}_{\hat{Q}}[t]\right) \tag{13}$$

TABLE I

USER PROFILES

	Delay $v_j[t]$ in sim. steps	**Max Job Size** $u_{n,\,\mathrm{max}}$ in res. blocks
Normal	20	30
High Packet Rate	20	40
Low Latency	2	8
Emergency Vehicle	1	16

is evaluated on the same batch of experiences. The loss assesses the sum utility $\hat{r}[t]$ estimated on batch experiences given the actors current parametrization $\boldsymbol{\theta}_{\boldsymbol{\mu}}[t]$, as well as the critics current understanding of the system dynamics, i.e., $\boldsymbol{\theta}_{\hat{Q}}[t]$. Updating the actor parameters $\boldsymbol{\theta}_{\boldsymbol{\mu}}[t]$ using the gradient $\nabla_{\boldsymbol{\theta}_{\boldsymbol{\mu}}[t]}\mathcal{L}_{\boldsymbol{\mu}}$ aims to change the parameters to maximize the *estimated* rewards.

IV. PERFORMANCE EVALUATION

A. Implementation Details

We configure the simulation environment with $U = 16$ available resources and $N = 10$ connected users. According to the profiles listed in Table I, we assign a normal profile to five of the users, High Packet Rate and Low Latency to two users each, and the last user is assigned the priority EV-profile. TX-SNR $P/\sigma^2_{\mathrm{noise}}$ is set to $13\,\mathrm{dB}$ for all users, with a variance $\sigma_h = 1$ for the fading channel amplitudes. Both training and evaluation are carried out with 50 simulation steps t, repeated over $10\,000$ episodes. The job generation probability per step per user is set to $c_j = 0.2$, putting an expected load of about 1.6 requests per resource on the system. We set the reward weightings of (6) such that the submetrics have approximately equal influence on the sum metric according to their expected magnitudes, $w_C = w_P = 0.25$ for sum capacity $r_C[t]$ and sum packet rate $r_P[t]$, and $w_L = w_{L,\mathrm{EV}} = 1$ for sum timeouts $r_L[t]$ and EV-timeouts $r_{L,\mathrm{EV}}[t]$. As EV-timeouts are part of the global sum timeouts, this weighting means that EV-timeouts have double the weight of other timeouts.

The neural networks for actor and critic are implemented as standard, fully connected feed-forward neural networks with 6 layers each, and $[300, 300, 300, 300, 400, 300]$ nodes per layer. Optimization is done on batches of size $B = 128$ using the Adam optimizer [16] with default parameters and a learning rate of 1×10^{-4} and 1×10^{-5} for critic and actor, respectively. Exploration noise is multiplied by a factor $\sigma_{\mathrm{e}} = 1.5$ initially, decaying linearly to 0 after $50\,\%$ of episodes. The learning loop is enabled after a minimum of 100 experiences have been collected in the buffer.

We use Python to implement the simuation, using primarily the Tensorflow library. Simulations are run on generic hardware. For further implementation details, we refer to the full code, available at [17].

B. Benchmark Algorithms

Three basic model-based scheduling algorithms [18] are implemented to gauge the DL schedulers performance on the given application.

- A Max Throughput (MT) scheduler allocates as many resources as requested to users by order of channel quality.

- A second scheduler operates by the max-min-fair principle (MMF), allocating at most an equal share of available resources to each requesting user.

- A Delay Sensitive (DS) scheduler assigns a channel priority

$$\bar{h}_n[t] = \frac{k_n[t]}{\sum_{\tilde{n}=1}^{N} k_{\tilde{n}}[t]} \frac{h_n[t]}{\sum_{\tilde{n}=1}^{N} h_{\tilde{n}}[t]}$$

given each users relative channel quality $h_n[t]$ and packet rate $k_n[t]$ (see (5)). Using the lowest time-to-timeout $l_n[t]$, and the total timeouts $m_n[t] = \sum_{\tau=1}^{t} r_{L,n}[\tau]$ specific to user n, the scheduler also assigns a timeout urgency

$$\bar{m}_n[t] = \frac{m_n[t]/l_n[t]}{\sum_{\tilde{n}=1}^{N} m_{\tilde{n}}[t]/l_{\tilde{n}}[t]}.$$

Both priorities are weighted and normalized, after which the scheduler assigns a share of the available resources equal to the weighted combined priority vector. Uniquely, this scheduler is allowed to not allocate to a job if it cannot be completed before timeout.

Random scheduling by selecting a normalized action $\mathbf{A}[t]$ with entries drawn from $U(0, 1)$ serves as the baseline benchmark.

C. Numerical Results

The simulation setup inherently exhibits a variance in the performance boundaries achievable within a simulation run, e.g., in some runs random job creation leads to longer queues and subsequently more unpreventable timeouts. Therefore, we display results in a normalized cumulative histogram of performances achieved during all simulation runs. Fig. 4 shows performance on the weighted reward sum and Fig. 3a to Fig. 3d break down performance on each component metric.

The DDPG scheduler, with the help of post-processing, is able to find a scheduling approach that outperforms the simple model-based algorithms on the given reward function and weighting. Analysis of the submetrics shows the balanced profile that the given weighting would suggest, going head-to-head with the best model-based algorithms performance in each category. On EV timeouts specifically the DDPG scheduler is showing by far the best performance out of any given algorithm, outperforming even the general timeout-focused Delay Sensitive scheduler, achieved by the targeted addition of EV timeouts to the maximization goal.

Fig. 5 further compares the weighted sum metric against a random baseline and a DQN adaptive scheduler [19] that selects from a variety of models in order to maximize a flexible reward goal. While the DQN ensemble scheduler is able to lean on model-based design more easily without yielding the flexible goal setting that RL offers, it is ultimately constrained by its available ensemble of models. The DDPG scheduler on the other hand is capable of
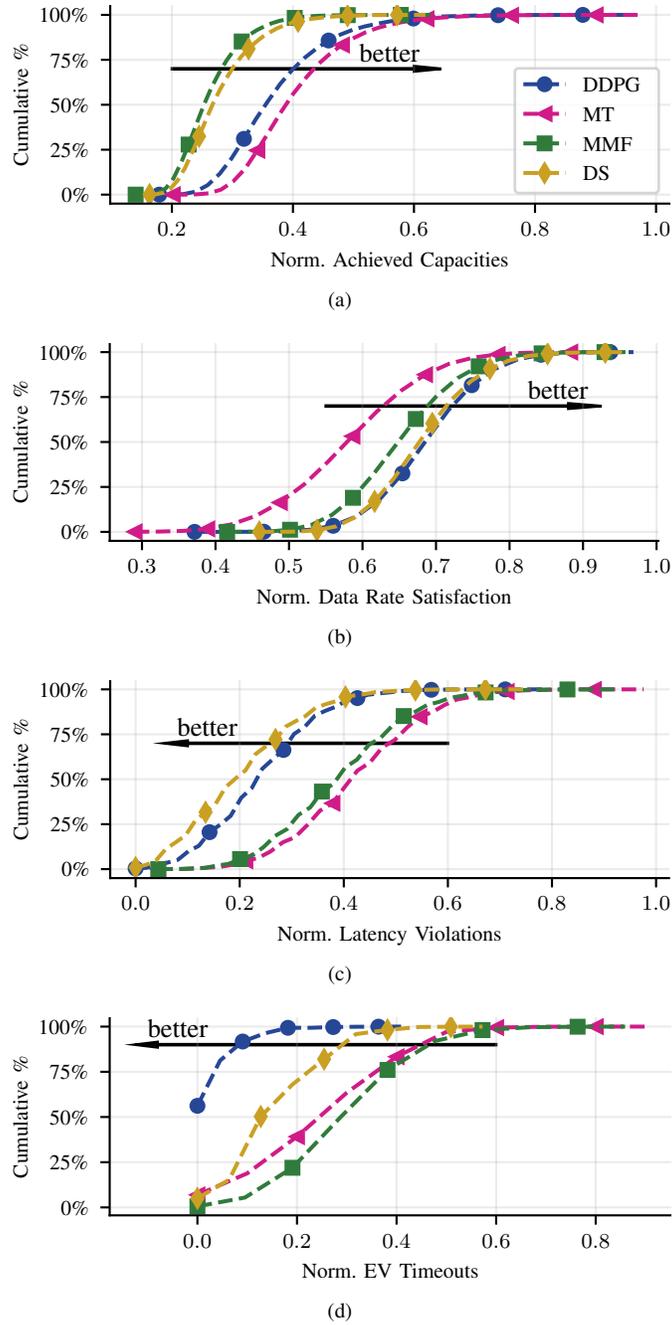
Fig. 3. Cumulative histograms of performance of Maximum Throughput (MT), Max-Min-Fair (MMF) and Delay Sensitive (DS) schedulers as well as DDPG scheduler on submetrics of capacity (a), packet rate (b), timeouts (c) and EV timeouts (d). The DDPG scheduler achieves strong performance on all submetrics, balancing them against each other in order to maximize their sum utility.

finding a scheduling solution better than any combination of model-based algorithms, if it exists, and therefore has a higher performance ceiling. Indeed, in the given simulation setup, the DDPG scheduler is able to outperform the DQN ensemble scheduler.
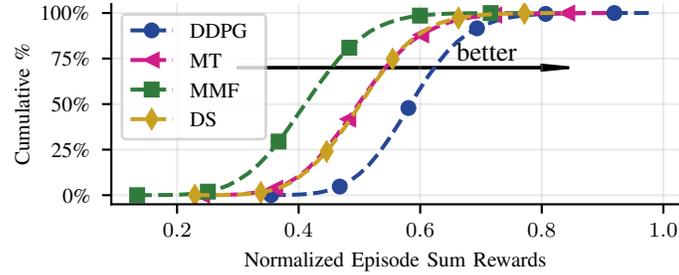
Fig. 4. Cumulative histogram of performance of Maximum Througput (MT), Max-Min-Fair (MMF), Delay Sensitive (DS) and DDPG scheduler on the weighted sum metric. For each episode, all achieved rewards $r[t]$ are summed. Achieved reward sums are normalized by the highest achieved reward sum.
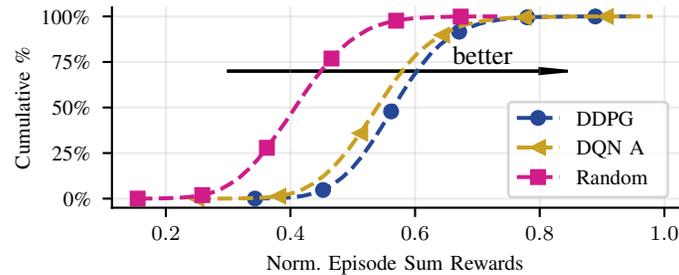
Fig. 5. Cumulative histogram of performance of DDPG, DQN adaptive and fully random scheduler on the weighted sum metric.

V. CONCLUSION

In this work, the DDPG principle of RL is used to learn a resource scheduling algorithm for communication system applications. Continuous-valued output allows DDPG to easily fit in with existing frameworks. The scheduling strategy learned by DDPG approximately maximizes a complex target utility, where special design boundary conditions, such as the protection of EV traffic, can be easily incorporated without minor impact on the overall performance. While neural-network-based learning algorithms are often regarded as black boxes, avenues for incorporating model-based expert inductive bias still exist. Most prominently, we make use of reward design and post-processing to guide and stabilize the learning process.

REFERENCES

[1] M. Zanatta, P. Benato, and V. Cianci, "Pre-Hospital Ultrasound: Current Indications and Future Perspectives," *Int. J. Crit. Care Emerg. Med.*, vol. 2, no. 2, p. 019, 2016.

[2] C. Büscher, J. Elsner, M.-T. Schneiders, S. Thelen, T. Brodziak, P. Seidenberg, D. Schilberg, M. Tobias, and S. Jeschke, "The Telemedical Rescue Assistance System "TemRas"–development, first results, and impact," *Biomed. Eng./Biomedizinische Technik*, vol. 59, no. 2, pp. 113–123, 2014.

[3] 3GPP, "Study on Communication Services for Critical Medical Applications," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 22.826, 2019, version 17.1.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails. aspx?specificationId=3546

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Proc. NeurIPS*, vol. 25, pp. 1097–1105, 2012.

[5] G. Dahl, M. Ranzato, A.-r. Mohamed, and G. E. Hinton, "Phone Recognition with the Mean-Covariance Restricted Boltzmann machine," *Proc. NeurIPS*, vol. 23, pp. 469–477, 2010.

[6] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," in *Proc. SPAWC*. IEEE, 2017, pp. 1–6.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] Z. Gu, C. She, W. Hardjawana, S. Lumb, D. McKechnie, T. Essery, and B. Vucetic, "Knowledge-Assisted Deep Reinforcement Learning in 5G Scheduler Design: From Theoretical Framework to Implementation," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2014–2028, 2021.

[9] M. M. Nasralla, "A Hybrid Downlink Scheduling Approach for Multi-Traffic Classes in LTE Wireless Systems," *IEEE Access*, vol. 8, pp. 82 173–82 186, 2020.

[10] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep Reinforcement Learning Based Resource Allocation for V2V Communications," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3163–3173, 2019.

[11] F. Al-Tam, N. Correia, and J. Rodriguez, "Learn to Schedule (LEASCH): A Deep Reinforcement Learning Approach for Radio Resource Scheduling in the 5G MAC Layer," *IEEE Access*, vol. 8, pp. 108 088–108 101, 2020.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[13] Y. Huang, S. Li, C. Li, Y. T. Hou, and W. Lou, "A Deep-Reinforcement-Learning-Based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR," *IEEE Internet Things J.*, vol. 7, pp. 6439–6456, 2020.

[14] N. Aschenbruck, E. Gerhards-Padilla, and P. Martini, "A survey on mobility models for performance analysis in tactical mobile networks," *J. Telecommun. Inf. Technol.*, pp. 54–61, 2008.

[15] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," in *Proc. AAAI-18*, vol. 32, no. 1, 2018.

[16] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980*, 2014.

[17] S. Gracla, "DDPG Scheduling," https://github.com/Steffengra/resourceallocation/tree/master/scheduling_policygradient, 2020.

[18] S. O. Schmidt, "Analyse von Latenz-empfindlichem Scheduling auf Systemebene für Anwendungen der Industrie 4.0," Master's thesis, Universität Bremen, Bremen, Germany, 2018.

[19] S. Gracla, E. Beck, C. Bockelmann, and A. Dekorsy, "Deep Reinforcement Model Selection for Communications Resource Allocation in On-Site Medical Care," *Submitted WCNC*, 2022.