On the Importance of Exploration for Real Life Learned Algorithms

Steffen Gracla, Carsten Bockelmann and Armin Dekorsy
Dept. of Communications Engineering, University of Bremen, Bremen, Germany
Email: {gracla, bockelmann, dekorsy}@ant.uni-bremen.de

Abstract

The quality of data driven learning algorithms scales significantly with the quality of data available. One of the most straight-forward ways to generate good data is to sample or explore the data source intelligently. Smart sampling can reduce the cost of gaining samples, reduce computation cost in learning, and enable the learning algorithm to adapt to unforeseen events. In this paper, we teach three Deep Q-Networks (DQN) with different exploration strategies to solve a problem of puncturing ongoing transmissions for URLLC messages. We demonstrate the efficiency of two adaptive exploration candidates, variance-based and Maximum Entropy-based exploration, compared to the standard, simple ϵ-greedy exploration approach.

Index Terms

5G, Machine Learning, URLLC, Puncturing, Exploration

I. Introduction

In recent years the topic of machine learning (ML) has reemerged with force due to breakthroughs in big data technology. Data driven learning methods boast a profile of strengths that complements the more traditional model-based design well, being able to extract approximate solutions from empirical data where models fail. As such, researchers from all fields of study are examining the feasibility of introducing ML into their areas of expertise, with promising results in communication technologies [1].

While the early findings for ML in communication systems are encouraging, some challenges lie ahead in transferring the theoretical findings to real life applications. Central to the strength of data driven technology is the quality of the learning data set itself; Although data collection has increased considerably in all areas of life, good real life data can remain costly and difficult to obtain. A variety of approaches have been proposed to alleviate this issue. For example, by generating data in a parameterized simulation environment, the data generation can be tuned to act as an inductive bias for optimal learning [2]. However, it has been observed that such data can suffer from a model gap where the simulation environment does not capture real life in sufficient detail to transfer learnings [3]. Further, approaches such as the Prioritized Experience Replay [4] aim to increase sample efficiency by extracting

This work was partly funded by the German Ministry of Education and Research (BMBF) under grant 16KIS1028 (MOMENTUM).
This work was accepted for presentation at IEEE SPAWC 2022.

as much information as possible from the available data. While this is valuable, these algorithms cannot increase the inherent quality of the data set. If information required for successful learning is not captured by the data set, the algorithm cannot learn it. Ultimately, the most straight-forward way to extract high quality samples from an environment is intelligent exploration.

This issue is of particular importance for the field of Reinforcement Learning (RL), where the algorithm generates its own data set as it learns. A common, simple approach to help RL-agents explore their environment is the ϵ-greedy exploration, in which the agents make a random decision at a probability ϵ instead of acting on their own. The parameter ϵ is annealed over the course of training as the agent gets ready to exploit what it has learned. While better than no exploration, this simple mechanism comes with a set of drawbacks: 1) In real life, high volumes of random actions can be costly; 2) The exploration is likely to generate redundant samples, exploring actions that the agent is already confident about; 3) If certain events are especially rare or only appear late during training, the random exploration probability ϵ may have already been annealed too much to discover them. Ideally, we would like to emulate curiosity during exploration, intelligently deciding to take a risk where uncertainty is high.

In this paper, we highlight some pitfalls of weak exploration. We examine a medium access control problem where an agent is tasked with scheduling URLLC messages. In order to do this, they may opt to puncture an existing transmission on orthogonal resource channels as proposed for 5G NR URLLC or wait to see whether a channel will become free in the near future. We train three simple Deep Q-Networks (DQN) [5] for this purpose: 1) A DQN with ϵ-greedy exploration; 2) A DQN with stochastic output; And 3) a DQN with an approximate Maximum-Entropy-Learning [6] constraint that imposes a uniform prior onto the output, in effect pulling the agent away from committing to just one course of action.

In the following, we will first introduce the setup and notations of our URLLC puncturing problem and review the design of our ML agents. We will then detail the differences in the employed exploration mechanisms and follow with practical examinations of their behavior in learning and in experiencing new situations. We conclude by summarizing our findings.

A. Related Work

The use of ML for QoS-optimal URLLC puncturing has been investigated by, e.g., [7], [8]. They show the general feasibility of using ML algorithms to learn efficient trade-off estimation in URLLC puncturing. Research on smart exploration or data generation strategies has a long history. The authors in [9] provide a rich survey of exploration mechanisms in the context of RL. More recently, methods such as SAC [3], Munchhausen DQN [10], and, in the context of communication technology, [11] have re-explored the concept of entropy-penalties to state-of-the-art RL. Our work brings the two topics together and highlights the importance of exploration in communication technology.

II. Preliminaries

This paper assumes knowledge of stochastic gradient descent learning and feed-forward neural networks (NN). Matrices and vectors are denoted in boldface.

III. Setup & Notations

In this section, we first describe mathematically the challenge of URLLC puncturing that we wish to learn a good strategy for, and secondly review the design and training of DQN for decision making with deterministic or aleatoric output.

A. URLLC Puncturing Simulation

Fig. 1 schematically displays the puncturing simulation. We assume our agent is a puncturing module that is part of a greater medium access protocol within a centralized communication traffic scheduler. As typical in OFDM, transmissions are scheduled by the greater protocol on discrete sub-frames of fixed length. Multiple orthogonal transmissions can be scheduled at the same time on N available resources. Each sub-frame is divided into 7 discrete blocks that we call mini-slots, akin to the numerologies specified in 5G NR. At the beginning of each sub-frame, the greater protocol fills each available resource n with a probability p_o for a length of $l_o \sim \mathrm{U}(5,\, 7)$ mini-slots drawn from a uniform random distribution. Further, at the beginning of each sub-frame, a power gain $h_{n,\,t} = |\tilde{h}|^2_{n,\,t}$ for each resource n with $|\tilde{h}|_{n,\,t} \sim \text{Rayleigh}(1)$ is drawn from a i.i.d. Rayleigh distribution for each resource. This introduces a measure of stochasticity into the simulation environment.

Time t moves discretely with the beginning of each mini-slot. On every t, with a probability p_p, a URLLC puncturing request may be posed to the puncturing agent. Puncturing requests occupy one mini-slot and come in two types: 1) The normal type has to be scheduled within the current sub-frame or else be discarded; 2) The critical type has to be scheduled within the next mini-slot or else be discarded. Only a low percentage $p_{p,\,c}$ of requests are critical. The agent can then select one out of $N+1$ options: Either schedule the request to one of the N available resource or do nothing. If the agent decides to schedule to a resource n, then any transmission ongoing in that resource is voided.

We therefore formulate three objectives for our puncturing agent:

1) Do not interrupt ongoing transmissions unnecessarily;
2) Puncture normal requests with as little influence on ongoing transmissions as possible;
3) Puncture critical requests immediately.

Mathematically, we formulate these objectives in a weighted reward sum, as is typical for RL. At the end of each mini-slot time step t, we determine three rewards $(r_{C,\,t},\, r_{d,\,t},\, r_{d,\,c,\,t}) \in \mathbb{R}$ for ongoing transmissions, normal requests and critical requests, respectively.

$$r_{C,\,t} = \sum_{n=1}^{N} \log(1 + h_{n,\,t}) \tag{1}$$

is the sum capacity achieved by ongoing transmissions within the mini-slot.

$$r_{d,\,t} = \begin{cases} -1 & \text{if normal request discarded} \\ 0 & \text{else} \end{cases}, \tag{2}$$

$$r_{d,\,c,\,t} = \begin{cases} -1 & \text{if critical request discarded} \\ 0 & \text{else} \end{cases} \tag{3}$$

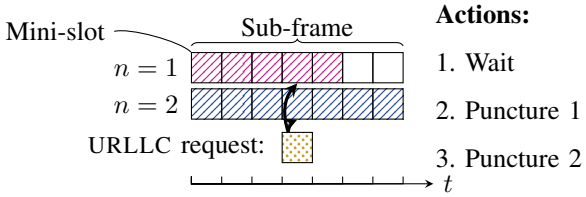

Fig. 1. URLLC Puncturing simulation for $N = 2$ Resources. Ongoing transmissions occupy at most one sub-frame of 7 discrete blocks, or mini-slots. At a time step t, there is a URLLC request. The puncturing agent can decide to wait, or puncture either of the mini-slots on the resources n.

are indicator functions for whether the objectives 2) and 3) were violated. All three rewards are collected with their respective weights in the reward sum

$$r_t = w_C r_{C,\,t} + w_d r_{d,\,t} + w_{d,\,c} r_{d,\,c,\,t} \tag{4}$$

for the learning agents to process.

B. Deep Q-Networks

In order to select the most beneficial out of a number of options, one might consider the long term benefit of selecting each option. In RL, the long term benefit is typically defined using the immediate reward r_t as defined in (4) as

$$R_{t,A_i} = \mathrm{E}_\pi \left[\sum_{\tau=t}^{\infty} \lambda^{\tau-t} r_\tau | A = A_i \right] \tag{5}$$

for an action A_i, given that we follow some policy π afterwards. Future rewards are typically scaled down exponentially by a discount factor $\lambda \in [0, 1]$ to devalue uncertain consequences in the far future. If this long term benefit R_{t,A_i} is fully known, optimal decisions may be chosen by greedily selecting whichever action A_i has the highest R_{t,A_i} in each time step t.

DQN attempt to approximate the long term benefit function (5) via neural networks. Feed-forward neural networks are parameterized mathematical functions with one or multiple inputs and outputs. The relation between input and output can be influenced by tuning the m network parameters $\boldsymbol{\theta} \in \mathbb{R}^m$. The parameters $\boldsymbol{\theta}$ are tuned automatically, typically using variants of stochastic gradient descent, such that the optimal parameters minimize an objective function. A DQN $Q(\mathbf{S}_t, \boldsymbol{\theta})$ specifically will take as an input a vector $\mathbf{S}_t$ that describes the current state of the system, to be described subsequently, and output an estimate for the long term benefit function R_{t,A_i} for all available actions A_i. When data points $(\mathbf{S}_t, A_i, r_t, \mathbf{S}_{t+1})$ are experienced, the network parameters can be tuned to improve the estimate by minimizing the temporal difference error

$$\begin{aligned} \mathcal{L}_{\mathrm{TD}} = &(Q(\mathbf{S}_t, A_i, \boldsymbol{\theta}_t) \\ &- (r_t + \max_i Q(\mathbf{S}_{t+1}, A_i, \boldsymbol{\theta}_t)))^2. \end{aligned} \tag{6}$$

This loss compares the networks current estimate $Q(\mathbf{S}_t, A_i, \boldsymbol{\theta})$ with an updated estimate that incorporates the experienced immediate reward r_t and following state $\mathbf{S}_{t+1}$.

In this paper, we consider two variations on DQN. The first has the same number of $N+1$ outputs as there are actions available, i.e., the decision to either do nothing or puncture one of the N available resources. The second type has twice the number of outputs, two for each action. Using the well-known reparameterization trick, these pairs of outputs are interpreted as mean and log-standard deviation for a Normal distribution from which the output estimates are subsequently sampled. This gives the second type of DQN an inherent variance in decision making; The standard deviation may also be interpreted as a measure of uncertainty in the network output.

We summarize the system state $\mathbf{S}_t$ in a simple vector with the entries

- $S_{1,\,t} \in [0, 1]$ is the current relative mini-slot position within the sub-frame;
- $S_{2,\,t} \in \{0, 1\}$ is 1 if there is a puncturing prompt, else 0;
- $S_{3,\,t} \in \{0, 1\}$ is 1 if there is a critical puncturing prompt, else 0;
- $S_{4:4+N,\,t} \in [0, 1]$ is each resources current relative remaining occupation.

IV. Exploration Mechanisms

In this paper, we compare three different learning agents. All three use DQN as described in the previous section, using three different exploration mechanisms: 1) ϵ-greedy exploration; 2) Variance based exploration; 3) Variance and approximate Maximum-Entropy based exploration. This section will introduce their workings and differences.

*1) ϵ-greedy exploration (*EG*)* is using a DQN with deterministic output. At each time step t, this exploration mechanism has two options. At a probability ϵ, a random action is selected with uniform probability from the set A, i.e., either do nothing or puncture a communication resource. Alternatively, at a probability $1-\epsilon$, the agent selects the action with the highest current DQN long term benefit estimate, $\max_{A_i} Q(\mathbf{S}_t,\, A_i,\, \boldsymbol{\theta}_t)$. Over the course of training, the probability ϵ is decayed, allowing the network to increasingly exploit the knowledge it has learned.

*2) Variance based exploration (*VB*)* is using a DQN with stochastic output. It introduces another term $\mathcal{L}_{\text{LP}}$ to the loss function (6) that is the sum of log probability densities $\text{lp}(Q(\mathbf{S}_t,\, A_i,\, \boldsymbol{\theta}_t))$ for the networks current sampled long term benefit estimates. Therefore,

$$\mathcal{L} = \mathcal{L}_{\text{TD}} + w_{\text{LP}}\mathcal{L}_{\text{LP}}, \tag{7}$$

$$\text{with } \mathcal{L}_{\text{LP}} = \sum_{i=1}^{N+1} \text{lp}(Q(\mathbf{S}_t,\, A_i,\, \boldsymbol{\theta}_t)). \tag{8}$$

Parameter w_{LP} can be tuned to scale the relative importance of each loss term. Low variances lead to high log probability densities; Therefore, in order to minimize this new loss, the DQN is enticed to keep variance high while still learning appropriate benefit estimates.

*3) Variance and approximate Maximum-Entropy based exploration (*ME*)* is using a DQN with stochastic output. Maximum Entropy Learning, known from algorithms such as Soft Actor-Critic [3], seeks to encourage exploration by rewarding a learning agent for keeping their decisions at high entropy, i.e., similar relative likelihood. As a proxy, we apply the softmax function to the networks long term benefit estimates (6), transforming the outputs into a pseudo-probability distribution to be able to calculate their entropy. This implies that $\text{sm}_i(Q(\mathbf{S}_t,\, A,\, \boldsymbol{\theta}_t)) \in [0, 1]$

TABLE I
TRAINING CONFIGURATION

Steps t per episode	3000	Episodes	30
Resources N	2	Rew. Discount λ	0.99
Prob. Tx p_o	70 %	Prob. URLLC p_p	10 %
ϵ Initial	0.99	Episodes $\epsilon \to 0.0$	50 %
Adam Learning Rate	$1e-4$	Target Update	$1e-4$
Hidden Layers $\times$ Nodes	2×128	VB Weight w_{LP}	$1e-2$
Cap. Weight w_C	1	URLLC Weight w_d	5
Critical Weight $w_{d,\,c}$	5	ME Weight w_{ME}	e

and $\sum_{i=1}^{N+1} \text{sm}_i(Q(\mathbf{S}_t,\, A,\, \boldsymbol{\theta}_t)) = 1$. We calculate the entropy of these new terms and add it as a second term to the temporal difference loss function (6),

$$\mathcal{L} = \mathcal{L}_{\text{TD}} + w_{\text{ME}}\mathcal{L}_{ME}, \tag{9}$$

$$\text{with } \mathcal{L}_{ME} = \sum_{i=1}^{N+1} \log[\text{sm}_i(Q(\mathbf{S}_t,\, A,\, \boldsymbol{\theta}_t))]. \tag{10}$$

Parameter w_{ME} is used to tune the relative importance of the loss terms. This new loss applies a uniform prior to the DQN output, encouraging the network output to be similar in magnitude. To promote the value of one action over the others, the parameter update needs to escape the pull of this new loss term.

V. EXPERIMENTS

This chapter investigates the impact of the three different choices in exploration strategy, ϵ-greedy (EG), Variance based (VB), and approximate Maximum Entropy (ME), detailed in the previous chapter. We first iterate specific implementation details used in this paper, then evaluate the success of each strategy in terms of initial learning sample efficiency and ability to explore new, unseen events. Finally, we break down the impact that these exploration strategies have on the overall puncturing performance.

A. Implementation Details

The used DQN are vanilla implementations with target networks [12] as the only addition, which we found required to reach acceptable learning stability. All simulation and DQN parameters can be found in Table I. We select reward weights w_C, w_d assuming they have been tuned by an expert to reach the desired balance of URLLC time outs and transmission interruptions for a given application. For the EG DQN we opted for a linear decay of the exploration probability ϵ to zero after half of the training episodes. For numerical stability, the softmax values in (10) are clipped to $[\text{sm}(\cdot)]^1_{1\times10^{-3}}$. The Adam optimizer [13] is used to perform gradient descent updates. We found most success using the penalized tanh activation function [14]. We focus on a small number of $N = 2$ resources to restrict computational and design complexity, though the methods and conclusions presented are expected to scale

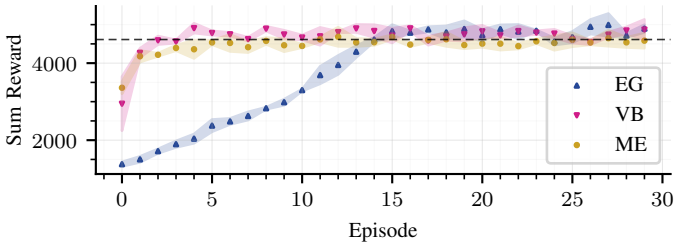

Fig. 2. The sum rewards achieved within a single training episode for ϵ-greedy (EG), Variance Based (VB), and Maximum Entropy (ME) Deep Q-Networks. The dashed black line shows the mean rewards achieved by manual scheduling. The EG DQN does not achieve strong results until its exploration parameter ϵ is annealed to zero. Both other exploration candidates are adaptive and achieve competitive results much more quickly.

to higher N. The design and tuning of stable NN learning for highly complex tasks represents its own challenge and would distract from the key research aim.

Simulation and learning are implemented in Python3.9 and TensorFlow. For further implementation details, the full code implementation is available online in [15].

B. Exploration Performance

For the initial training, we set the probability $p_{p,\,c}$ of encountering critical URLLC events to zero. Three networks EG, VB, ME, with exploration strategies as described in Section IV, are trained according to Table I. Training is repeated three times for each type, as the simulation and data generation have an inherent variance due to the simulation's stochastic setup. The mean rewards achieved during training by each variant are depicted in Fig. 2. The dark dotted line denotes the mean rewards achieved by manual scheduling, which all three networks are able to meet eventually. As expected, the EG DQN generates many more samples to reach comparable performance to the other two candidates, as the exploration strategy is in no way adaptive. This also results in many episodes with mediocre performance.

Next, we take the networks as trained in the previous step and confront them with a critical URLLC event in the first mini-slot of a sub-frame and both resources occupied. Unlike normal URLLC events, critical events must be scheduled immediately or else time out. We record the inferred results in Table II. It shows the magnitude difference (MD) in preference between action one compared to action two or three, calculated as

$$Q(\mathbf{S}_t,\, A_1,\, \boldsymbol{\theta}_t)/((Q(\mathbf{S}_t,\, A_2,\, \boldsymbol{\theta}_t) + Q(\mathbf{S}_t,\, A_3,\, \boldsymbol{\theta}_t))/2); \tag{11}$$

the log standard deviation of selecting action one; and the mean log standard deviation of selecting actions 2 and 3. We note that all three DQN show preference for action one, i.e., doing nothing, therefore letting the critical URLLC request time out. This is to be expected, as neither DQN have seen a critical event during their training thus far and therefore cannot have learned the requirement to schedule it immediately. In these results we can already spot the effects of our exploration strategies, where the EG DQN has by far the largest difference in magnitude between the preferred action and the other actions and the ME DQN has the smallest difference. Further, the VB DQN has considerably lower certainty in its decision to commit to doing nothing compared to the ME DQN.

TABLE II

MEAN (MD) AND LOG STD. (LS) REACTION TO NEW EVENT

	EG	**VB**	**ME**
MD	$(80.00 \pm 0.03)\,\%$	$(36.00 \pm 0.05)\,\%$	$2.00\,\%$
ls 1	-	-2.26 ± 0.28	-13.84 ± 0.88
ls 2 & 3	-	-0.27 ± 0.09	-13.96 ± 1.18

TABLE III

MEAN AND STD. TRAINING STEPS UNTIL EXPLORING NEW EVENT

EG	**VB**	**ME**
6800 ± 4525	96 ± 17	22 ± 2

For the final step, we again load the pre-trained DQN and again confront them with the new critical event situation from the previous step. This time, we train them on this experience, confront them again, and repeat until a DQN first decides to take a puncturing action, i.e., an action other than action 1. Due to the stochastic nature of the simulation we repeat this ten times for each pre-trained network. Table III displays the mean training steps required until each DQN decides to explore this new situation. On some training runs, the EG DQN is stopped early after not deciding to explore for $10\,000$ steps, while both other exploration strategies are able to start exploring massively more early in every training.

C. Puncturing Performance Impact

After examining how the adaptive mechanisms have improved their respective networks' exploration strategies, we next examine their impact on asymptotic reward sum performance. Fig. 3 breaks down the ratio of transmissions interrupted by puncturing over the course of training, while Fig. 4 shows the ratio of URLLC prompts missed over the course of training. Both VB and ME show slightly weaker asymptotic performance on the transmissions interrupted. We attribute this to three factors:

1) All networks converge to slightly different puncturing strategies with slightly different focus on each sub-task, for similar approximate overall performance on the optimization metric r_t. For example, the manual puncturing, represented by the black dotted line, puts a heavy focus on catching URLLC requests, leading to a slightly increased amount of transmissions interrupted;
2) Adding another term to the optimization function, in the form of exploration penalties, does represent a potential loss in optimality. The optimization focus is no longer to just optimize the target metric r_t, but to balance it with additional constraints. This effect is somewhat mitigated by VB and ME still being greedy schemes, i.e., selecting the highest estimate action, which leads to accuracy on the non-maximum estimates being less important. Further, this effect is controllable via the weighting factors $w_{\mathrm{LP}}, w_{\mathrm{ME}}$. In this paper,

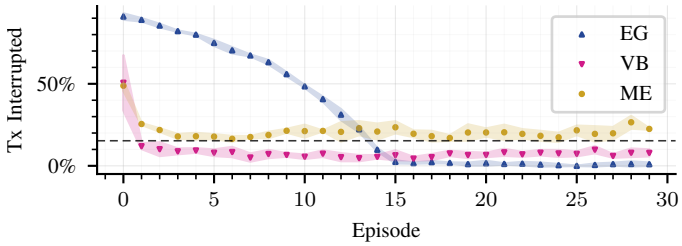

Fig. 3. Relative transmissions interrupted for puncturing for ϵ-greedy (EG), Variance Based (VB), and Maximum Entropy (ME) Deep Q-Networks over the course of training. The dashed line represents the mean result achieved by manual puncturing.

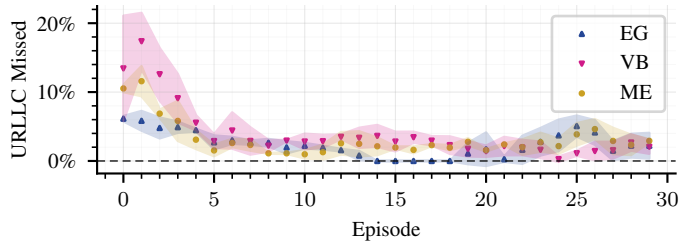

Fig. 4. Relative URLLC prompts missed over the course of training for ϵ-greedy (EG), Variance Based (VB), and Maximum Entropy (ME) Deep Q-Networks. The dashed line represents the mean result achieved by manual puncturing.

the weight for ME in particular was set to a value that leads to excellent exploration performance for this application, as expressed in Table II and Table III;

3) The training regimen used in this paper was not adjusted for each network's training and may therefore favor one variant over another.

VI. Conclusions

In this paper we examined an optimization problem in puncturing ongoing transmissions for URLLC messages of differing priority. We implemented three learning agents, one with a standard ϵ-greedy deterministic Deep Q-Network (DQN) and two DQN with stochastic output. For the stochastic DQN we implemented exploration strategies based on a variance penalty and Maximum Entropy Learning, respectively. Both stochastic DQN exploration strategies encourage the learning agent to explore when uncertain and to not commit too heavily onto a single course of action. While all three agents were able to learn to solve the optimization problem, we showed how the adaptive exploration strategies can lead to significant gains in learning sample efficiency and ability to adapt to unforeseen events, both of which we consider to be crucial for real life learned algorithms. However, no exploration algorithm is universally optimal, and therefore must be applied mindful of their limitations.

References

[1] H. Dahrouj, R. Alghamdi, H. Alwazani, S. Bahanshal, A. A. Ahmad, A. Faisal, R. Shalabi, R. Alhadrami, A. Subasi, M. T. Al-Nory, O. Kittaneh, and J. S. Shamma, "An Overview of Machine Learning-Based Techniques for Solving Optimization Problems in Communications and Signal Processing," *IEEE Access*, vol. 9, pp. 74 908–74 938, 2021.

[2] A. T. Z. Kasgari, W. Saad, M. Mozaffari, and H. V. Poor, "Experienced Deep Reinforcement Learning with Generative Adversarial Networks (GANs) for Model-Free Ultra Reliable Low Latency Communication," *arXiv:1911.03264*, Oct. 2020.

[3] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," *arXiv:1812.05905*, Jan. 2019.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv:1511.05952*, 2015.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602*, 2013.

[6] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum Entropy Inverse Reinforcement Learning," in *Proc. AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[7] 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, no. 7, pp. 6439–6456, Jul. 2020.

[8] J. Li and X. Zhang, "Deep Reinforcement Learning-Based Joint Scheduling of eMBB and URLLC in 5G Networks," *IEEE Wireless Commun. Lett.*, vol. 9, no. 9, pp. 1543–1546, 2020.

[9] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup, "A Survey of Exploration Methods in Reinforcement Learning," *arXiv:2109.00157*, 2021.

[10] N. Vieillard, O. Pietquin, and M. Geist, "Munchausen Reinforcement Learning," *arXiv:2007.14430*, 2020.

[11] A. Srivastava and S. M. Salapaka, "Parameterized MDPs and Reinforcement Learning Problems–A Maximum Entropy Principle-Based Framework," *IEEE Trans. Cybern.*, pp. 1–13, 2021.

[12] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep Exploration via Bootstrapped DQN," *Advances in neural information processing systems*, vol. 29, 2016.

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980*, 2015.

[14] S. Hayou, A. Doucet, and J. Rousseau, "On the Impact of the Activation Function on Deep Neural Networks Training," in *Proc. ICML*, 2019, p. 9.

[15] S. Gracla, "Adaptive Scheduling Model Selection," https://github.com/Steffengra/on_exploration, 2020.