Applications of Q-Learning to Network Optimization and Graph Problems

Marco Dollinger, Max Helm, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: dollingm@in.tum.de, helm@net.in.tum.de, jaeger@net.in.tum.de

Abstract—**This paper provides a theoretical overview of Markov Decision Processes (MDP), Reinforcement Learning (RL) in general, and (Deep) Q-Learning in particular. Furthermore, we examine the application of Deep Q-Learning in network optimization of Software-defined Satellite-terrestrial networks and in general graph problems like the traveling salesman problem (TSP) and a graph representation of the boolean satisfiability problem (SAT). Furthermore, we reference the results obtained by Deep Q-Learning approaches to the examined application areas. Moreover, we give an overview of recent research progress in the field of Reinforcement Learning and present open questions and challenges.**

Index Terms—**Q-Learning, Reinforcement Learning, Software-defined Satellite-terrestrial Networks, SAT, TSP**

1. Introduction

In the recent past, Reinforcement Learning has received extensive research interest from academia as well as industry. In particular, Reinforcement Learning promises meaningful advances in the field of robotics, control theory, statistics, and economics among others. The Google DeepMind application AlphaGo, which is based on Reinforcement Learning, was even covered by mainstream media for beating world-class players in the board game Go. This paper is structured as follows: Section 2 provides an introduction to the theoretical foundations of Reinforcement Learning, and in particular to Q-Learning which is a specific type of Reinforcement Learning. Section 3 analyzes and categorizes applications of Q-Learning to graph problems like the traveling salesman problem and the boolean satisfiability problem as well as applications to network optimization on the example of Software-defined Satellite-terrestrial networks. In Section 4, an overview of recent developments and future challenges of Reinforcement Learning research is given. Lastly, Section 5 concludes the paper and summarizes the results.

2. Theoretical Foundations

In this section, an overview of the theoretical foundations of Reinforcement Learning and specifically Q-Learning is presented.

2.1. Reinforcement Learning

Reinforcement Learning is one of the three main paradigms of modern machine learning, next to supervised and unsupervised learning. In Reinforcement Learning, an agent takes actions within an environment to maximize rewards. Usually, a Reinforcement Learning environment is modeled as a Markov Decision Process. In [1], Watkins defined an MDP as:

- a set of actions A
- a set of states S,
- $R_a(s, s')$: a reward function that rewards the agent when transitioning from state s to state s' using action a
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$: a probability function that expresses the probability that a certain action a which is taken at time t in state s will result in state s'.

It is important to note that Markov Decision Processes satisfy the Markov Property that transitions and rewards only depend on the current state and are independent of previous actions or states.
Further, we will only introduce finite Markov Decision Processes with finite sets of actions and states. The model of an MDP is the combination of transition function and reward function. When the model of an MDP is unknown, Reinforcement Learning is a possible technique to find an optimal policy ("when to choose which action"). As described by Kaelbling et al. in [2], we can divide RL techniques into model-free approaches and model-based approaches.

- Model-free: learn a policy without learning the model.
- Model-based: learn the model to derive a policy.

The goal for the RL agent is to find a policy that maximizes the acquired rewards until a sequence of actions leads to a final state, or the algorithm is aborted (e.g. by a time constraint). Figure 1 shows the general framework for Reinforcement Learning of a Markov Decision Process as illustrated by Wang et al. in [3].
In a Reinforcement Learning process, the term "regret" describes the performance differences between the actual agent and a (hypothetical) optimal agent which we aim to minimize. A common obstacle to minimizing regret is the tradeoff between long-term rewards (exploration) and short-term rewards (exploitation). Exploitation means learning from previous experiences and choosing the most optimal action as the next decision. In contrast, exploration is the process of trying new policies/decisions that can offer a smaller immediate reward but enables the agent to learn more information about the environment [2].

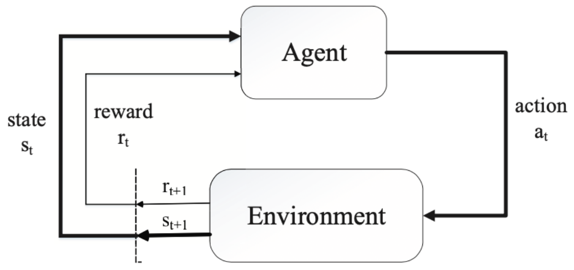

Figure 1: The Reinforcement Learning Framework [3]

2.2. Q-Learning

Q-Learning is a model-free Reinforcement Learning algorithm that learns a controller without learning transition probabilities or rewards from the environment. The Q-function calculates the quality of a state-action combination which is a measure of how good it is to take an action in a specific state.

$$Q : S \times A \rightarrow \mathbb{R} \tag{1}$$

The learning goal is to find an optimal Q-function for each state-action pair that can be used to achieve our goal of maximizing reinforcement rewards. At the beginning of learning, the Q-function might be randomly initialized. During learning, the agent will update the Q-function with each decision step with the following formula [1]:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2}$$

where:

- r_t is the reward that our agent receives when moving from state s to state s_{t+1} using action a
- the learning rate $\alpha : 0 < \alpha \leq 1$ defines how much we weight "new knowledge" compared to previous experiences. This is similar to many machine learning algorithms.
- the discount factor $\gamma : 0 < \gamma \leq 1$ weighs immediate rewards against future rewards
- $\max_a Q(s_{t+1}, a)$ is an estimate of the maximum reward that can be obtained from state s_{t+1}

When implementing the Q-Learning algorithm in an RL agent, it would be understandable that for every action decision, the agent should choose the action with the highest Q-value for the current state. However, with this approach, the agent is purely exploiting previous knowledge and neglects the exploration aspect of the RL task. This problem is solved by ϵ-greedy Q-Learning as proposed by Wunder et al. in [4]. In ϵ-greedy Q-Learning, the agent chooses the action with maximum Q-value with the probability of $(1 - (\epsilon(k-1)/k)$ and selects one of the remaining actions with a uniform probability distribution. With increasing ϵ, the agent increasingly explores the environment rather than exploiting its knowledge. However, ϵ needs to be sufficiently small to avoid unnecessarily increasing the learning duration.

Since in basic Q-Learning (2), we calculate the Q-function for the whole, (sparse) action-state-matrix, the algorithm can become very computationally expensive. To speed up our computations we can use Quantization which means grouping similar actions or states together at the cost of quantization errors. Quantization can discretize infinite spaces or decrease the cardinality of discrete state/action spaces. Another approach to handle large action and state spaces is function approximation. With function approximation, the agent does not compute the complete action/state matrix, but rather only "estimates" the Q-function values, for example by using neural networks. [3]

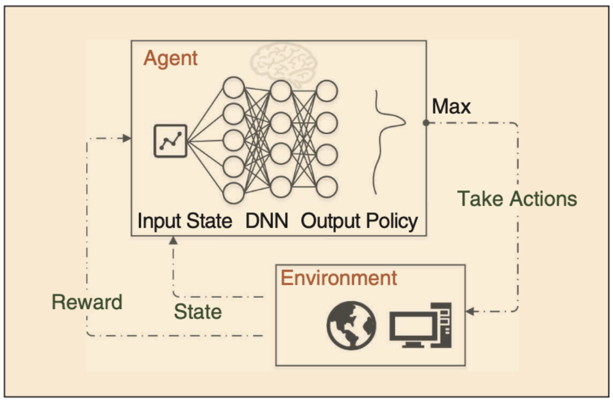

Figure 2: An illustration of DQL. DNN: deep neural network [5]

2.3. Deep Q-Learning

Deep Q-Learning (DQL) uses a neural network to realize non-linear function approximation which enables efficient Q-Learning in high dimensional action and state spaces. Figure 2 shows the general Reinforcement Learning framework with a neural network agent as illustrated by Tham et al. in [5]. The input of the neural network represents the current environment state, whereas the maximum value of the output layer encodes the next action to take by the agent.

3. Applications of (Deep) Q-Learning

This section introduces example applications of (Deep) Q-Learning to network optimization and graph problems. Since the examined problems are very high-dimensional and computationally expensive, (Deep) Q-learning promises great performance in their respective solutions.

3.1. Network Optimization of Software-Defined Satellite-Terrestrial Networks (SDSTN)

One goal of network optimization is the optimal network resource allocation to many different actors. This matching problem is a high-dimensional task that can be modeled as an MDP and therefore solved with Reinforcement Learning. In other words, network optimization aims to fulfill user requirements/requests while minimizing resource consumption.
As Chao et al. showed in [6], Deep Q-Learning can

improve network resource allocation in software-defined satellite-terrestrial networks. The physical resources of the network are categorized in the data layer as networking capabilities through low earth orbiters (LEO), caching/storage space in distributed infrastructure and computing capacity of distributed mobile edge computing (MEC servers). Software-defined satellite-terrestrial networks virtualize physical resources by a logically centralized control layer that allocates the optimal resources to user/application requests like communication or navigation tasks [7]. Since the physical network resources are distributed over many different entities, allocating specific devices to a user request is a high-dimensional matching problem that can be modeled by the general reinforcement framework in Figure 1 and thus solved by Deep Q-Learning. The data layer of the network represents the state $S(t)$ at time t of the RL task, in particular the position and availability of LEOs, the cached contents, and the idle/occupied computing capabilities. The RL agent is the control layer that decides for a given user u request at time t the optimal action $a_u(t)$ which includes assigning an LEO, deciding if the requested content should be cached, and allocating a MEC server to execute the computation task. According to He et al. in [8], the control layer needs to pay fees to consume the physical resources, whereas the user u pays the control layer for executing a request. The RL reward function calculates the ratio of fees paid by the control layer divided by the fees paid by the user. With increasing efficiency of physical resource consumption, while maintaining constant user fees, the respective fee ratio increases which rewards the agent to optimize its allocation policy [6].
To measure the performance/efficiency of the Deep Q-Learning approach to resource allocation, Chao et al. simulated an SDSTN with three LEOs, five MEC servers, and five content caches. Compared to a static resource-to-user allocation strategy, the DQL-based resource allocation achieved greater utility per resource and thus increased the efficiency of the network. Additionally, the authors showed that "with the increase of training episodes, the expected utility per resource increases" [6] which proves that the modeled reward function can be used to optimize the agent's policy, and therefore optimize the network's allocation strategy.

3.2. Graph Problems

3.2.1. Boolean Satisfiability Problem (SAT). Given a boolean formula, the Boolean Satisfiability Problem is the task of finding a satisfying variable configuration. Since SAT is NP-complete, commercial solvers rely on heuristics to speed up finding a satisfying configuration or proving unsatisfiability. As shown by Kurin et al. in [9], Deep Q-Learning can potentially be applied to commercial settings and reduce wall-clock time to solve SAT problems. In [9], Kurin et al. introduce Graph-Q-SAT that uses Q-Learning with graph neural networks as function approximation. Similarly to Selsam et al. in [10], boolean formulas in conjunctive normal form (CNF) are represented by bipartite graphs. Since boolean formulas vary in size and the graph changes during solving by setting variables, the DQL input must support dynamic dimensionality. Therefore, Graph-Q-SAT uses Graph Neural

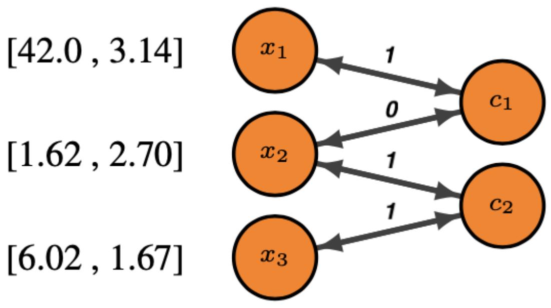

Figure 3: Graph representation of $(x_1 \vee x_2) \wedge (-x_2 \vee x_3)$. The annotations are Q-function values for setting variables to true or false respectively [9].

Networks as formalized by Battaglia et al. in [11]. Besides vertices and edges, Graph Neural Networks include annotations, which change by their operations. For example, Figure 3 shows a possible input/output of a Graph Neural Network of the formula $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. To decide the next action (setting one variable to True or False), the agent selects the highest annotation value of all variable nodes. In the example of Figure 3, the agent will set $x_1 = True$. The reward function punishes the agent for each non-terminating decision which incentivizes the agent to find a satisfying configuration as fast as possible. For unsatisfiable formulas, Graph-Q-SAT will try all configurations to prove unsatisfiability [9]. To evaluate Graph-Q-SAT, Kurin et al. trained the agent with Random 3-SAT instances from the SATLIB benchmark. It is demonstrated that Graph-Q-SAT outperforms Variable State Independent Decaying Sum (VSIDS) by reducing the required iterations to solve SAT problems by 2-3 times. VSIDS is a frequently used Conflict Driven Clause Learning (CDCL) branching heuristic which means that for each iteration the solver chooses a variable and assigns a binary value, similarly to Graph-Q-SAT. In particular, Graph-Q-SAT needed less than half the iterations of VSIDS for SAT-50-218 instances (50 variables, 218 clauses). Further important characteristics to evaluate are the generalization properties of Graph-Q-SAT. Compared to VSIDS, "Graph-Q-SAT has no difficulty generalizing to larger problems, showing almost 4X improvement in iterations for a dataset 5 times bigger than the training set" [9] which shows great generalization to other problem sizes of Graph-Q-SAT. Another important characteristic is the generalization to unSAT problems (unsatisfiable SAT instances) when trained only on SAT problems. While Graph-Q-SAT can solve unSAT problems, its performance is worse than on SAT problems relative to VSIDS. This is partly because unSAT differs from SAT problems, where proving unsatisfiability requires exhausting all possible assignments, whereas one satisfying assignment suffices to prove satisfiability. While Graph-Q-SAT achieved great performance overall, Kurin et al. noted that "more work is needed to apply Graph-Q-SAT to reduce wall clock time in modern SAT solving settings" [9].

3.2.2. Travelling Salesman Problem (TSP). Given a weighted graph where vertices represent cities, the traveling salesman problem is the task of finding the shortest

Hamiltonian circle. Since the TSP is NP-complete, we rely on heuristic algorithms to efficiently solve the problem for large graphs. Introduced by Gambardella et al. in [12], the Reinforcement Learning-based algorithm "Ant-Q" can be applied to the TSP and achieve competitive performance compared to other heuristic algorithms.
Ant-Q is inspired by the "ant system" (AS) as described by Colorni et al. in [13], and the Q-Learning algorithm [1]. Because Ant-Q is a distributed algorithm, the performance can be further increased by additional (distributed) computing capacity. Given an agent k located in city r, the agent moves to city s using the following formula:

$$s = \underset{u \in J_k(r)}{argmax} [AQ(r, u) \cdot HE(r, u)] \tag{3}$$

where:

- $J_k(r)$ is the list of cities that agent k has not yet visited.
- $AQ(r, u)$ is the equivalent to the Q-function (2) that expresses the usefulness of moving to city u when located in city r
- $HE(r, u)$ is a heuristical value that is used to prefer small distances. E.g. by multiplying the inverse of the distance between r and u

While Ant-Q was the best performing algorithm "compared to the elastic net, simulated annealing, the self-organizing map, and farthest insertion" [12] for specific standard sets of the symmetric TSP (symmetric weights), Ant-Q's polynomial time complexity makes it impossible to apply it to large TSPs. However, for asymmetric TSPs, which are harder than the symmetric case, Ant-Q delivered promising results that are usually only achieved by very specialized algorithms [12].

4. Recent Developments in Deep Reinforcement Learning Research

Hardware advances, particularly GPUs, have driven increased interest in Deep Learning over the last decade. As mentioned in Sections 2 and 3, Deep Neural Networks are used for function approximation in Deep Reinforcement Learning. Since function approximation made it feasible to apply Reinforcement Learning to increasingly large action and state spaces, DRL continues to be successfully applied to fields like robotics, control theory, and more. Recent examples were the super-human performance DRL agents achieved in playing Atari games [14] or the success of DeepMind's AlphaGo achieving world-class performance in the board game GO [15], which is computationally considerably more complex than chess. Another example is the OpenAI Gym Bipedal Robot, which successfully applies DRL to the control of robotic joint angles which has analog, or potentially very large, state and action spaces [15].
However, the increasing applications of DRL have also raised further problems and open questions which continue to drive research interest. For example, how to secure the stability of DRL, which refers to the stability of weights of the DNN, remains a mostly open question [16]. Another research area is the application of transfer learning to speed up the training process. When training a robotic DRL agent with visual input, transfer learning enables the agent to learn from simulated data before using real-world data which makes training more cost-efficient and speeds up development cycles [14].

5. Conclusion

This paper provides a theoretical introduction to Reinforcement Learning and in particular Q-Learning as well as the techniques to apply Q-Learning to high dimensional data through function approximation. Additionally, Section 3 summarized possible applications and results of Deep Q-Learning (function approximation with deep neural networks) to network optimization and graph problems. As shown in Section 3, Deep Q-Learning outperformed static resource allocation strategies in the simulation of Software-defined Satellite-terrestrial networks [6]. Additionally, Deep Q-Learning promises to reduce wall-clock time in SAT-solving [9] and asymmetric TSPs [12]. Section 4 outlined recent research activity in reinforcement learning and raised open questions/challenges to further apply Deep Q-Learning in fields like robotics.

References

[1] C. Watkins, "Learning From Delayed Rewards," 01 1989.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *CoRR*, vol. cs.AI/9605103, 1996. [Online]. Available: https://arxiv.org/abs/cs/9605103

[3] H. Wang, X. Chen, Q. Wu, Q. Yu, X. Hu, Z. Zheng, and A. Bouguettaya, "Integrating Reinforcement Learning with Multi-Agent Techniques for Adaptive Service Composition," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 12, pp. 1–42, 05 2017.

[4] M. Wunder, M. Littman, and M. Babes-Vroman, "Classes of Multiagent Q-learning Dynamics with ϵ-greedy Exploration," 08 2010, pp. 1167–1174.

[5] M.-L. Tham, A. Iqbal, and Y. Chang, "Deep Reinforcement Learning for Resource Allocation in 5G Communications," 11 2019, pp. 1852–1855.

[6] Q. Chao, H. Yao, F. Yu, F. Xu, and C. Zhao, "Deep Q-Learning Aided Networking, Caching, and Computing Resources Allocation in Software-Defined Satellite-Terrestrial Networks," *IEEE Transactions on Vehicular Technology*, 04 2019.

[7] B. Yang, Y. Wu, X. Chu, and G. Song, "Seamless Handover in Software-Defined Satellite Networking," *IEEE Communications Letters*, vol. 20, 06 2016.

[8] Y. He, F. Yu, N. Zhao, and H. Yin, "Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities: A Big Data Deep Reinforcement Learning Approach," *IEEE Communications Magazine*, vol. 55, pp. 31–37, 12 2017.

[9] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning," *CoRR*, vol. abs/1909.11830, 2019. [Online]. Available: http://arxiv.org/abs/1909.11830

[10] D. Selsam, M. Lamm, B. Bunz, P. Liang, L. Moura, and D. Dill, "Learning a SAT Solver from Single-Bit Supervision," 02 2018.

[11] P. W. Battaglia, J. B. Hamrick, V. Bapst, and ..., "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[12] L. M. Gambardella and M. Dorigo, "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem." 01 1995, pp. 252–260.

[13] A. Colorni, M. Dorigo, and V. Maniezzo, "An Investigation of some Properties of an "Ant Algorithm"." 01 1992, pp. 515–526.

[14] K. Arulkumaran, M. P. Deisenroth, and ..., “A Brief Survey of Deep Reinforcement Learning,” *CoRR*, vol. abs/1708.05866, 2017. [Online]. Available: http://arxiv.org/abs/1708.05866

[15] J. Shin, T. Badgwell, K.-H. Liu, and J. Lee, “Reinforcement Learning – Overview of Recent Progress and Implications for Process Control,” *Computers & Chemical Engineering*, vol. 127, 05 2019.

[16] L. Busoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, “Reinforcement learning for control: Performance, stability, and deep approximators,” *Annual Reviews in Control*, vol. 46, 10 2018.