ENHANCEMENT OF THE S-DALINAC CONTROL SYSTEM WITH MACHINE LEARNING METHODS*

J. Hanten†, M. Arnold, J. Birkhan, C. Caliari, N. Pietralla, M. Steinhorst
Institut für Kernphysik, TU Darmstadt, Darmstadt, Germany

Abstract

For the EPICS-based control system of the superconducting Darmstadt electron linear accelerator S-DALINAC, supporting structures based on machine learning are currently developed. The most important support for the operators is to assist the beam setup and controlling with reinforcement learning using artificial neural networks. A particle accelerator has a very large parameter space with often hidden relationships between them. Therefore neural networks are a suited instrument to use for approximating the needed value function which represents the value of a certain action in a certain state. Different neural network structures and their training with reinforcement learning are currently tested with simulations. Also there are different candidates for the reinforcement learning algorithms such as Deep-Q-Networks (DQN) or Deep-Deterministic-Policy-Gradient (DDPG). In this contribution the concept and first results will be presented.

INTRODUCTION

The S-DALINAC [1] is a superconducting, thrice recirculating electron linear accelerator at the institute for nuclear physics at the TU Darmstadt (see Fig. 1). It is used for investigation of nuclear structure physics and is operated since 1991 in recirculating mode. The design value of its energy is 130 MeV at a maximum current of 20 µA. The accelerator operates in a continuous wave mode with a frequency of 3 GHz. Its electrons are provided by a thermionic gun or a spin polarized source. The acceleration proceeds, after passing a copper based chopper prebunching system, in an up to 10 MeV superconducting injector and an up to 30 MeV main LINAC. The position and spot size of the electron beam within its pipe is currently controlled with scintillating BeO-Targets and can be manipulated with corrector dipoles and quadrupoles.

It has to be optimized in terms of position, dimension, transmission and energy resolution to suit the proposed experiment. However, due to partly unknown fluctuations, set points from previous beam times can not be used again without adaption. At the moment, beam setup is done manually by operators and can take up to several weeks for the most complex systems. To improve this situation, it is planed to use deep reinforcement learning algorithms to support this process.

* *Work supported by DFG through GRK 2128
† jhanten@ikp.tu-darmstadt.de

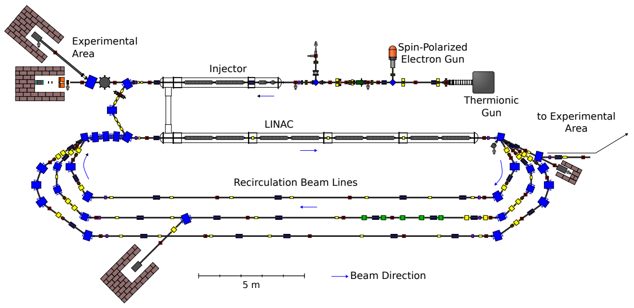

Figure 1: Accelerator hall floorplan of the thrice recirculating S-DALINAC.

DEEP REINFORCEMENT LEARNING

Deep Learning with the use of artificial neural networks (NN) as function approximators is used more frequently in the last years for sensory processing and computer vision [2]. There were also accelerator physics-related applications such as image based diagnostics of particle beam parameters [3]. More recently, there was a significant progress in combining deep learning with reinforcement learning, resulting in the "Deep Q Network" (DQN) algorithm [4]. This was capable to perform human level performance in many Atari games, having only the pixel data as its input. Because of the big parameter space and the nonlinear connection between these, machine learning and especially NNs are also proposed to be suited for modeling and control of particle accelerators [5].

At present, different algorithms are tested with a simple simulated pair of corrector magnets with the electron beam tracking algorithm *elegant* [6].

Parameters and Basic Functions

The standard reinforcement learning setup consists of an agent interacting with an environment E, in a sequence of actions a_t, observations of states s_t and rewards r_t, where $t = \{1, \dots, T\}$ is the index for one discrete time step. In general E can also be stochastic. The values s_t can be represented by the beam center coordinates, beam width, target position, magnet set points, etc. When used with the accelerator, it is planed to use the *areaDetector* [7], a plugin for the EPICS based control system [8] to obtain this information from the images produced by the scintillating targets. It is also possible to add additional environment parameters like temperature or vacuum pressure to s_t. At each time step the agent chose an action a_t from a set of set-point changes of corrector or quadruple magnets. Depending on the chosen algorithm, the action space can be discrete or continuous.

The goal is to steer the beam on a desired point of a target. In order to reach it, the algorithm must be able to learn from a scalar reward r. A positive reward is received if the chosen action will lead to a position change towards the destination while a negative reward is received if the beam will increase the distance to the destination or leaving the target. The objective for the agent is to select actions maximizing future rewards. The standard assumption is made that future rewards are discounted by a factor of $\gamma \in [0, 1]$ per time step so that the return at time t is defined as

$$R_t \coloneqq \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i) \tag{1}$$

where T is the time step of a terminate state (i. e. beam reaching the destination or leaving the target). With this, the optimal action-value function can be defined as

$$Q^*(s, a) \coloneqq \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right] \tag{2}$$

where π is the policy that maps the states to actions.

If the Q^* is known, the agent can chose the action a in the current state s with the maximum value of Q^*. This is the same as maximizing the value of $r + \gamma Q^*(s', a')$, with s' being the next state after executing action a and a' being the optimal action in the state s'. This leads to an identity known as *Bellman Equation*

$$Q^*(s, a) = \mathbb{E}_{s' \sim E}\left[r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right]. \tag{3}$$

The basic idea behind reinforcement learning algorithms (included the one described here), is to estimate the action-value function so the agent can chose the action with the biggest value of Q^*. This can be done by using the Bellman equation as an iterative update,

$$Q_{i+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s', a') \middle| s, a \right]. \tag{4}$$

For estimating the value of Q a function approximator is needed and for that a NN can be used. An NN with at least one hidden layer and a nonlinear activation function can be used as universal approximator [9]. The weights θ of the NN are updated by the algorithms for the objective $Q(s, a; \theta) \approx Q^*(s, a)$. This is referred as Q-Network in the following. A model of a Q-Network which could be used can be seen in Fig. 2.

DEEP-Q-NETWORK-ALGORITHM

One possible candidate for an automatic beam setup using reinforcement learning is the DQN-algorithm. The complete algorithm can be looked up in [4]. It is suited for an continuous state space but is limited to a discrete action space. The basic procedure is shown in Fig. 3. An important hyperparameter is the greediness ϵ which describes the possibility, that the agent chooses a random action instead of choosing the one with the maximum Q-Value according to the Q-Network. If ϵ is chosen to large, the agent tends to rarely exploit the state-action pairs with higher rewards and thus the NN learns less efficient in the interesting areas. However, if ϵ is chosen to small, the agent does not look for better strategies which is a problem especially in the start of training. Therefore it is often chosen as $\epsilon = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot e^{-t/\lambda}$ with λ as a decay hyperparameter and t represents the current time step. This ensures a balance of exploration and exploitation during the learning process.

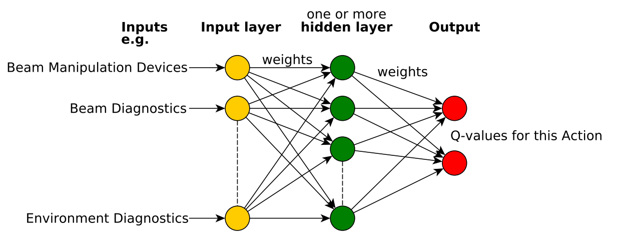

Figure 2: Illustration of an NN as function approximator for the Q-function.

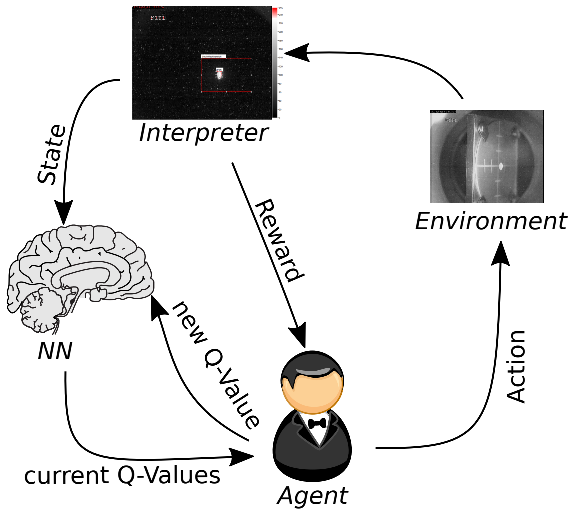

Figure 3: Basic visualized priciple of the DQN-algorithm. The agent selects an action which affects the environment. The input values for the NN have to be interpreted out of this environment. The output values of the NN representing the Q-values and the reward of the interpreter are used to calculate the new Q-value which is then used to train the NN.

After an action is chosen the reward r_t is given and and a new value for Q can be calculated according to Eq. (4). The value for $\max_{a'} Q_i(s', a')$ can be received from a second NN, the so called target net, which is a copy of the original NN (then called policy net) but is only updated in a certain number of episodes. This improves the stability of the training. To train the NN a Huber loss function is used (other loss functions are also possible)

$$L_t(\theta_t) = \begin{cases} \frac{1}{2}\left(y_t - Q(s_t, a_t; \theta_i)\right)^2 & \text{for } |y_t - Q(s_t, a_t; \theta_i)| < 1 \\ |y_t - Q(s_t, a_t; \theta_i)| - 0.5 & \text{otherwise} \end{cases} \tag{5}$$

where y_t is the calculated updated Q-value from the Bellman equation. The weights θ of the NN are then updated with a gradient descent method or similar algorithms like *Adam* [10] which was used in this application.

A useful technique to prevent overfitting is the replay memory. It is a cyclic buffer of size N where tuples of the experence at each time step, $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a data set $D = e_1, \dots e_N$. At each learning step a so called mini batch of this buffer is sampled. This minimizes the correlation between samples and makes the learning process independent of the agent's immediate actions (off policy). Equation (5) will then be changed to the expected value over a mini batch.

Test of a DQN-Algorthm in a Simulation

The DQN-algortihm was tested with the tracking code *elegant* and a simple beam line consisting of one pair of corrector magnets and a drift line with a length of 15 cm. The objective was to reach the destination point on the target within 50 episodes. The deflection angle of the magnets and the destination points of the virtual target which had a radius of 15 mm, were chosen randomly in every episode. The Q-Network consists of two hidden layers with each having 128 neurons. As activation function the exponential linear units

$$f(x) = \begin{cases} e^x - 1 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \qquad (6)$$

are used.

The input of the NN was a six dimensional state representation, consisting of the current defection angle of both magnets, the absolute coordinates on the target and the distance coordinates relative to the destination. These values are normalized by subtracting its mean value and dividing by there standard deviation assuming a uniform distribution.

Different action space consists of a discrete change of the magnet deflection (horizontal and vertical). Three different action spaces were tested, defined as

$$A_4 = \{-5\,\text{mrad}, 5\,\text{mrad}\}^2$$
$$A_9 = \{-5\,\text{mrad}, 0\,\text{mrad}, 5\,\text{mrad}\}^2$$
$$A_{25} = \{-10\,\text{mrad}, -1\,\text{mrad}, 0\,\text{mrad}, 1\,\text{mrad}, 10\,\text{mrad}\}^2\,.$$

An ϵ-greedy policy with an exponentially decaying ϵ was used to ensure exploration. The parameters were set to $\epsilon_{\text{start}} = 0.5$, $\epsilon_{\text{end}} = 0$ and $\lambda = 500$. Also a replay memory with the size 10,000 was applied. The algorithm was trained for 2000 episodes, with one episode ending with the beam in a certain range of a destination point or by reaching a critical distance to the center point of the virtual target representing the escape from the target. One episode was limited to 50 steps before it was aborted. The given reward was proportional with the distance reduction (1/mm) per step with an additional reward (10) when reaching the goal and negative reward (-10) when leaving the target. The discount factor was set to $\gamma = 0.25$. The selection of a rather low γ was reasonable because due to the linearity of the problem in this test case, the maximization of the current reward will usually lead to a higher long term return.

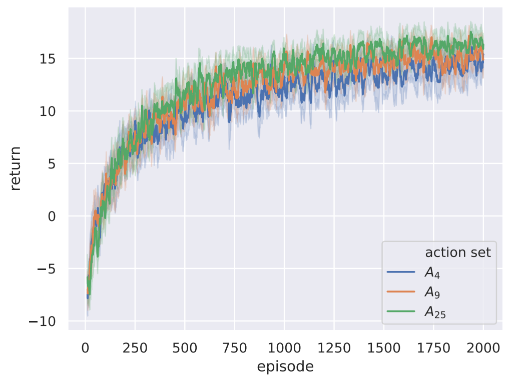

Figure 4: Average reward over the last 10 episodes during training with the reinforcement algorithm DQN with different action space sizes. The strong colored lines shows the mean value over 20 different training processes. The pale colored areas marks its 95 % confidence interval [11].

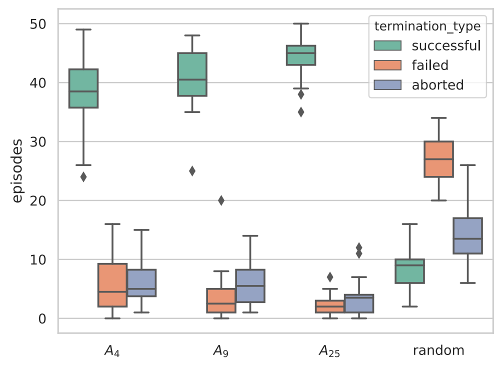

Figure 5: Boxplot to determine the performance of the 20 trained Q-Networks after 2000 training episodes in a test set of 50 episodes with different action space sizes and random chosen actions. Successful agents were able to reach the target. In failed tests the agent left the specified radius and in aborted cases the agent was not able to reach the goal within 50 episodes [11].

The average reward of the 20 training sessions in dependence of the episode for each action space is shown in Fig. 4. As one can see, the algorithm learns with all action spaces and it has a better performance with the bigger finer action space A_{25}. This approved by an additional test afterwards, where the trained NNs were tested with 50 random scenarios. In Fig. 5 it is seen, that the A_{25} has the biggest success rate. It is also possible that more training would lead to a higher performance.

CONCLUSION AND OUTLOOK

A first reinforcement learning algorithm working with the simulation code *elegant* was presented. It was shown that the algorithm based on DQN was able to learn the value function for a simple beam line consisting of two corrector magnets and a drift path. It was seen that the algorithm had a better performance with a larger action set and could perform potentially better with a more episodes of training. As a next step the algorithm has to be expanded to more complex beam lines with more demanding tasks so that it can be finally used in parts of the beam line during operation. Therefor the NN needs probably more neurons and potentially more layers.

As mentioned above, the drawback of using DQN-algorithms is the limitation to a discrete action space. Due to the discretization the dimension of the action space grows exponential with the number of elements and the number of dicretization steps. This limitation could be a bigger problem when applicating the algorithm to more complex beam lines with more target set points and quadrupole focusing. An alternative to DQN is the policy gradient algorithm "Deep Deterministic Policy Gradient" (DDPG) [12].

In contrast to DQN, DDPG learns a policy $\pi : S \rightarrow \mathcal{P}(A)$ where S represents the state space and $\mathcal{P}(A)$ the power set of the action space. This has the advantage that no discretization is needed. Instead two interleaved NNs are used: One with the (real numbered) action as output and the other for calculating the Q-value for this action. Both have the current state as input and the value network uses the chosen action additionally.

The DDPG algorithm looks like a promising alternative to tackle the problems of DQN and will be tested in the future.

REFERENCES

[1] N. Pietralla, *Nuclear Physics News*, vol. 28, no. 2, p. 4, 2018. DOI: 10.1080/10619127.2018.1463013

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, 6 2012. DOI: 10.1145/3065386

[3] A. L. Edelen, S. G. Biedron, S. V. Milton, and J. P. Edelen, "First steps toward incorporating image based diagnostics into particle accelerator control systems using convolutional neural networks," Dec. 16, 2016. arXiv: 1612.05662v1 [physics.acc-ph]

[4] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," Dec. 19, 2013. arXiv: 1312.5602v1 [cs.LG]

[5] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile, "Neural networks for modeling and control of particle accelerators," *IEEE Trans. Nucl. Sci.*, vol. 63, 2 2016. DOI: 10.1109/TNS.2016.2543203

[6] M. Borland, "Elegant: A flexible sdds-compliant code for accelerator simulation," Sep. 2000. DOI: 10.2172/761286.

[7] M. Rivers. (2019). Areadetector, https://areadetector.github.io/master/index.html

[8] C. Burandt *et al.*, "The epics-based accelerator control system of the s-dalinac," in *Proceedings of ICALEPCS 2013, San Francisco, CA, USA*, 2014, p. 82. http://tubiblio.ulb.tu-darmstadt.de/90003/

[9] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis, 43(2):233-268, 2017*, May 14, 2015. DOI: 10.1016/j.acha.2015.12.005. arXiv: 1505.03654v2 [cs.NE]

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 22, 2014. arXiv: 1412.6980v9 [cs.LG]

[11] C. Caliari, Bachelor's Thesis in preperation, TU Darmstadt, 2019.

[12] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," Sep. 9, 2015. arXiv: 1509.02971v6 [cs.LG]