








On Autonomous Drone Navigation Using Deep Learning and an Intelligent Rainbow DQN Agent

Andreas Karatzas¹, Aristeidis Karras¹ , Christos Karras¹ ,
Konstantinos C. Giotopoulos² , Konstantinos Oikonomou³ ,
and Spyros Sioutas¹ 

¹ Computer Engineering and Informatics Department,
University of Patras, Patras, Greece
{akaratzas, akarras, c.karras, sioutas}@ceid.upatras.gr

² Department of Management Science and Technology,
University of Patras, Patras, Greece
kgiotop@upatras.gr

³ Department of Informatics, Ionian University, Corfu, Greece
okon@ionio.gr

Abstract. Drones are intelligent devices that offer solutions for a continuously expanding variety of applications. Therefore, there would be a significant improvement if these systems could explore space automatically and without human-supervision. This work integrates cutting-edge artificial intelligence techniques that allow drones to travel independently. Following an overview of reinforcement learning methods built for discrete action space settings, a multilayer Perceptron model is constructed for feature extraction along with Hybrid neural networks. The agent employed in the experiments is a Rainbow DQN agent trained on the AirSim simulator. The experimental results are encouraging as the agent was tested for 16 missions and the accuracy was higher than 93%. In particular, the success for action selection was 97% and 93 for mission success. Finally, future work related to the navigation of autonomous drones is discussed including current concepts and methods of integration with more sophisticated algorithmic approaches.

Keywords: Drones · Autonomous drone navigation · Deep learning · Reinforcement learning · Intelligent agents · Swarm intelligence · UAV

1 Introduction

The area of autonomous robotics is rapidly growing due to the range of tasks that robots can perform [7]. Examining a number of missions done by such robots, it is discovered that there is an increase in risk. In many of these instances, the integrity of the human workers sent at the point of dispatch is at danger. As the operator is removed from the hazardous shipping point, robots arrive to handle

the situation. Because of their usefulness, robots have been mass-produced. In addition to ground robots, there are also flying robots which keep the scientific community busy. Researchers are concerned about the autonomy of such systems [6]. This specific difficulty is intriguing, as groups of unmanned aerial vehicles (UAVs) now take off frequently. Coordinating various systems with a shared goal might provide a solution in a variety of circumstances, such as forest patrols, identification of suspicious behaviour in stalking instances, etc.

When studying designs of autonomous computer systems for unmanned aerial vehicles, the subject of accessible information arises, i.e., what information is available that may be utilised by the system to solve the autonomy problem? The standard sensors of an unmanned aerial vehicle are: i) Camera(s) ii) Radar(s) iii) a Time-of-flight sensor and iv) a LiDar Sensor [8].

Other than the first sensor, the others contribute to obstacle detection in an unknown areas. The camera of the UAV gathers all required information. Using image processing techniques mixed with new approaches of artificial intelligence, there are presently systems that can do remarkably well in such tasks given the right circumstances [2]. Typically, the sole stipulation is that there must be sufficient light for the camera to capture the required information in fine detail. Smaller resolution might entail higher uncertainty about the solution the system software will compute at its output, and hence there might be an increase in risk.

2 Preliminaries

A challenge in reinforcement learning (RL) may be characterised as a Markov process decision consisting of four sets [1] $\langle S, A, R, T \rangle$. S is a limited collection of training circumstances experienced by the agent. The information provided by the environment at the input of each training cycle for the agent configures the current state of the agent. The A is a limited collection of responses to a circumstance from which an agent may pick. $R(s, a)$ computes the reward offered to a state s after an action a . $HT(s, a, \hat{s}) \rightarrow [0, 1]$ is a potential transfer function. It specifies the likelihood of transitioning from state s to state \hat{s} after action a . The functions $R(s, a)$ and $T(s, a, \hat{s})$ solely rely on the current values of s , a , and \hat{s} . The probability density of a Markov decision process is defined as follows:

$$\Pr \{s_{t+1} = \hat{s}, r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} \quad (1)$$

Equation (1) is a specific example of the more general situation in which the present state also relies on earlier states and actions (2).

$$\Pr \{s_{t+1} = \hat{s}, r_{t+1} = r \mid s_t, a_t\} \quad (2)$$

When Eq. (1) is equivalent to Eq. (2) for all \hat{s}, r and the sequence of form $s_t, a_t, r_t, s_1, a_1, r_1, s_0, a_0, r_0$ then the case holds the Markov property.

When investigating RL algorithms, the value of the Markovian property is substantial. However, in the majority of cases, agents do not have access to all of the data hence, there is concealed state data, preserving the Markov condition. Such data are described as Markov decision processes that are partly observable.

2.1 Value Function

Each training cycle is comprised of the agent observing a state and executing one action. The mapping between state and action is known as policy π . Specifically, the chance of selecting an action a in a given state s under policy is specified by the probability function V^π as $\pi(s, a) \rightarrow [0, 1]$ as defined by Eq. (3).

$$V^\pi(s) = E_\pi \{R_t \mid s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \quad (3)$$

In Eq. (3), E_π is the expected value received by agent if follow policy π for a state s_t . Equation (3) can be rewritten, as shown in Eq. (4), as the sum of the rewards presented in the training cycles performed by the agent.

$$V_\pi(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \implies V_\pi(s_t) \equiv \sum_{k=0}^{\infty} \gamma^k r_{t+i} \quad (4)$$

In Eq. (4), the factor $\gamma \in [0, 1]$ is a degradation value of this sum so that the most recent rewards gain more weight. An equally important concept is the function Q^π , also known as the energy function and payoff for policy π . Q^π is the expected payoff having chosen an action a in a state s and following policy π . The function $Q^\pi(s, a)$ is given by Eq. (5).

$$Q^\pi(s, a) = E_\pi \{R_t \mid s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \quad (5)$$

The goal when training a RL agent is to approximate the policy that maximizes the sum of rewards over time. The optimal policy is denoted by π^* and the value function which returns the maximum accumulated reward to the agent in a state s following the optimal policy π^* is denoted by $V^*(s)$ and is called the reward function. $Q^*(s, a)$ is the optimal function of reward, and means the expected reward of choosing an action a in a state s following the optimal policy.

2.2 Multilayer Perceptron Neural Networks

Perceptron multilayer neural networks are composed of many layers of perceptron-type neurons. Perceptron neurons receive input data through the forward propagation technique and are trained using the reverse propagation algorithm. Input vector is received by the first layer of Perceptron neurons. Subsequent layers change the input vector using two parameters: the numerical weight of synapses between neurons at the various levels and the non-linear activation function that follows the output of each neuron.

A multi-layer Perceptron neural network employs a collection of numerical samples created by certain controlling features for each class in the set of classes

throughout its training. The format of the produced sample set is X matrix, as shown in Eq. (6).

$$X = \begin{bmatrix} | & | & \cdots & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & \cdots & | \end{bmatrix} \quad (6)$$

In Eq. (6), each vector $x^{(i)}$ has dimension n , where n is the number of the attributes used to code the classes. Therefore, it holds that $X \in \mathbb{R}^{n \times m}$, where m is the number of training samples. Same as the registry X , there is also the Y register in which the desired output of the after model is stored from the transformation of the input data, which is shown in Eq. (7). Each element $y^{(i)}$ of register Y has a number representing the class it belongs to the sample i . Therefore, it holds that $Y \in \mathbb{R}^{i,m}$.

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}] \quad (7)$$

From the forward propagation of the X matrix in the model, intermediates arise transformations of the matrix X . The matrix A in each hidden level j of the neural network is defined as in Eq. (8).

$$A^{[j]} = \begin{bmatrix} | & | & \cdots & | \\ a^{(1)[j]} & a^{(2)[j]} & \cdots & x^{(m)[j]} \\ | & | & \cdots & | \end{bmatrix} \quad (8)$$

The synapse weights of a multi-layer Perceptron neural network for one layer j are described in Eq. (9).

$$W^{[j]} = \begin{bmatrix} w_{1,1}^{[j]} & w_{1,2}^{[j]} & \cdots & w_{1,m}^{[j]} \\ w_{2,1}^{[j]} & w_{2,2}^{[j]} & \cdots & w_{2,m}^{[j]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1}^{[j]} & w_{n,2}^{[j]} & \cdots & w_{n,m}^{[j]} \end{bmatrix} \quad (9)$$

In Eq. (9), the subscript n refers to the number of neurons in layer $j-1$, while the index m refers to the number of neurons in level j . At each level j , there is also a special neuron called a threshold. Threshold has no synapses with the previous level. A constant is stored in the threshold which is propagated only at the respective level. Considering b_j the threshold at a level j , then Eq. (10) describes the forward propagation result algorithm from a level $j-1$ to a level j .

$$Z^{[j]} = \left(W^{[j]}\right)^T \times A^{[j-1]} \quad (10)$$

Threshold $b^{[j]}$ is added to each element of register $Z^{[j]}$. Then, it takes effect in the register a non-linear function called the activation function. With the activation function, the model can categorize the samples of the set education. In case there is no trigger function, then the model performs a simple linear transformation of the input data and therefore will not learn complex patterns.

If $g(\cdot)$ is the activation function, then at the end of a cycle forward propagation of the neural network, the result shown in Eq. (11).

$$A^{[j]} = g\left(Z^{[j]} + b^{[j]}\right) \quad (11)$$

3 Methodology

3.1 Deep Q Networks

Deep Q-nets were developed as a result of the necessity for an algorithm capable of tackling a broad variety of issues. It integrates reinforcement learning concepts with artificial neural networks. The development of complex topologies for artificial neural networks, which use additional layers of neurons for more effective generalisation and pattern recognition, made it feasible for machines to learn categories from raw data. Deep Q networks primarily take tensors as input; hence, deep convolutional networks are used for the identification of needed standards. At the model's input, the tensor shells are pictures of a single channel.

Deep Q networks solve challenges involving agent-environment interaction. In particular, it posits that the agent may observe a circumstance, choose an action, and get a reward. The agent's objective is to choose the sequence of behaviours that accumulates the greatest potential amount of rewards over time. Specifically, a neural network with deep convolution is employed to approximate the ideal energy-reward function shown in Eq. (12).

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \mid s_t = s, a_t = a, \pi \right] \quad (12)$$

Equation (12) expresses the maximum sum of rewards r_t reduced by a factor γ in each training cycle t . This sum can be achieved following the behavior defined by the policy function $\pi = P(a \mid s)$ in a state s performing action a .

Neural networks like other non-linear methods until the discovery of deep Q networks were considered unsuitable for approximating the function energy-reward (also known as Q function). The difficulties are enough, such as for example small changes in the Q function that can produce large ones changes in agent policy. The concept of deep Q networks uses experience repetition mechanism that solves the problem of correlating similar data in a time sequence by reducing the rate of change of the data distribution. The second element of deep Q networks is the iterative update of the Q function in order to more accurately approximate the optimal energy-action function Q^* .

A tuple is stored to implement the iterative mechanism $e_t = (s_t, a_t, r_t, s_{t+1})$ at each training cycle t forming a data set $D_t = e_1, \dots, e_t$. During learning, refreshes are applied following the method of learning Q on samples (chunks) of experiences $(s, a, r, \hat{s}) \sim U(D)$, which are chosen in a random fashion following a uniform distribution from the set D_t . For implementation of the iterative renewal mechanism introduces a new neural network itself architecture with the model used to train the agent.

The second model \hat{Q} aims to estimate the optimal energy. Every t training cycles, \hat{Q} is synchronized with model Q . This change stabilizes \hat{Q} algorithm as it deals with small but often unwanted changes in agent policy that would occur at the end of each training cycle. Thus, based on [5], the refresh learning following the learning method Q in an iteration i is configured from the cost function described in Eq. (13).

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,\hat{s}) \sim U(D)} \left[\left(r + \gamma \max_a Q(\hat{s}, \hat{a}; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right] \quad (13)$$

In Eq. (13), the θ_i are the parameters of the neural network Q during iteration i and the θ_i^- are the parameters of the neural network \hat{Q} used for the estimation of the best possible energy during repetition i . Traffic information of the agent in the environment is formed by combining 4 consecutive single frames channel. To avoid overtraining, except for the conversion done in each frame from color to grayscale, the frame is also cropped from 210×160 to 84×84 . Thus, variable s , which is the convolutional neural network input is a $4 \times 84 \times 84$ tensor size.

3.2 Double Deep Q Networks

One of the new concepts that appeared in deep Q networks was the usage second neural network to estimate the optimal policy, in parallel with one neural network operating during agent training. In this way, the agent is not affected by small changes in policy and the training process converges as only after a number t training cycles does synchronization occur of the Q model with the \hat{Q} model. However, this can also lead to overestimation of the objective as the same maximization operator is used, as shown in Eq. (13).

With the algorithm of double deep Q networks a separate operator is introduced maximization for the \hat{Q} model. Therefore, there is a separate selection operator energy and a separate energy evaluation operator. Also, the policy that is modeled by the ϵ -greedy method is evaluated by the Q model, but the reward estimated by the model \hat{Q} . Based on [9], the policy update for the double deep algorithm Q of networks can be described as in Eq. (14).

$$Y_t^{\text{Double DQN}} \equiv R_{t+1} + \gamma Q \left(S_{t+1}, \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a; \theta_t), \theta^- \right) \quad (14)$$

3.3 Learning with Multiple Training Cycles

The method adopted for solving problems with RL methods uses the model of Markovian decision processes. According In this decision model, an agent interacts in an environment for a series training cycles t . In each training cycle, the agent receives information for the environment state $S_t \in \mathcal{S}$, where \mathcal{S} is the set of all possible situations. The agent uses this information to choose an action A_t from the set of all possible actions \mathcal{A} . Based on the agent's behavior in state $S_t \in \mathcal{S}$, a payoff $R_{t+1} \in \mathbb{R}$ is calculated and the agent moves to next state

$S_{t+1} \in \mathcal{S}$ with a state transfer probability $p(\hat{s} | s, a) = \hat{s} | S_t = s, A_t = a$, for $a \in \mathcal{A}$ and $S, \hat{s} \in \mathcal{S}$. The behavior of the agent is determined by policy $\pi(a | s)$ follows, which is a probability distribution over the set $\mathcal{S} \times \mathcal{A}$.

During the training of the agent, the optimal policy π^* is formulated that maximizes the estimated reduced total reward, as described in Eq. (15).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} \quad (15)$$

Algorithms following the time difference method aim to maximizing the amount of G_t . The state-value function describes the estimated performance when the agent is in a state s and follows policy π , as shown in Eq. (16).

$$u_\pi = \mathbb{E}[G_t | S_t = s] \quad (16)$$

At the center of the agent's training is also the energy function-value, which is the estimated payoff when the agent chooses an action a over a state s following policy π , as shown in Eq. (17).

$$q_\pi = \mathbb{E}_\pi [G_t | S_t = s, A_t = a] \quad (17)$$

Equation (17) can be calculated iteratively by observing new rewards based on previous estimates of q_π and using the renewal rule that shown in Eq. (18).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (18)$$

In Eq. (18), the constant $\alpha \in (0, 1]$ is the cycle length parameter education. The time difference method can be extended to more cycles education. By carefully choosing a parameter $n > 1$ improved results can be obtained results when training an agent. The result is Eq. (19), which shows the update rule used for learning with multiple cycles of training [3].

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1}^{t+n} [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \quad (19)$$

In Eq. (19), the estimated return $G_{t:t+n}$ for an agent using learning n training cycles is given by Eq. (20).

$$G_{t:t+n} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \quad (20)$$

In Eq. (19) and Eq. (20), the quantity Q_{t+n-1} is the estimate of the function q_π at time $t + n - 1$ and the subscript $t : t + n$ denotes the renewal duration. Also, in Eq. (19), the term ρ_{t+1}^{t+n} sets the sampling mode so that they are selected proportionally the most important samples and is described by Eq. (21).

$$\rho_t^{t+n} \prod_{k=t}^{\tau} \frac{\pi(A_k | S_k)}{\mu(A_k | S_k)} \quad (21)$$

In Eq. (21), the variable $\tau = \min(t + n - 1, T - 1)$ is the training cycle until the end of the refresh step.

3.4 Rainbow Agent

A Rainbow agent is a combination of all the previous methods (Sects. 3, 3.2, and 3.3). First, the learning duration is increased, as described in Sect. 3.3 using n training cycles for learning. The multiple cycle learning method training is also combined with the dual deep Q network method using the action obtained by following the ϵ -greedy method in the state S_{t+n} . Finally, the architecture of the model is conflicting, and on the pathways used to assess benefit and value noise is introduced from a NoisyNet model. Rainbow agent hyperparameters are given in Table 1, as they were optimized. The Rainbow agent [4] successfully combines all previous enhancement learning techniques, as shown in Fig. 1.

Table 1. Hyperparameter table of a Rainbow agent.

Hyperparameter	Value
Minimum training cycles before learning	80.000
Adam optimizer Training rate	0.0000625
Parameter ϵ	0.0
Parameter σ_0 for NoisyNet models	0.5
Synchronization interval between \hat{Q} and Q model	32.000
Numerical stability parameter ϵ for the Adam optimizer	0.00015
Precedence exponent ω	0.5
Importance parameter β for priority sampling	0.4 \rightarrow 1.0
Parameter n for learning with multiple training cycles	3
Sets of groups in the N value distribution	51
V_{MIN}, V_{MAX}	$[-10, 10]$

3.5 Problem Formulation

On the complex problem of autonomy in navigation for unmanned aerial vehicles a Rainbow agent was created [4]. Two different ones were created architectures at the Q model level. In the first case, the kernel that uses the agent to extract the necessary patterns from the environment is a multilayer Perceptron neural network, while the second kernel was hybrid. As a core characterizes the model that processes the input data before passing through the advantage and value paths of the overall Q model. The tasks of the agent are as follows:

1. The agent is asked to move from a reference point $\langle x_0 y_0 z_0 \rangle$ to a target point $\langle x_1 y_1 z_1 \rangle$, with $x_0 \neq x_1, y_0 \neq y_1$ and $z_0 \neq z_1$.
2. The agent shall not collide with any object during its transition in the target.
3. The agent should try to reach the shortest known¹ route.

¹ The agent is not aware of the map, i.e. the layout of the obstacles. However, he knows the direction where the target is located. Therefore, it can estimate the direction to shift it with a larger one pace to target.

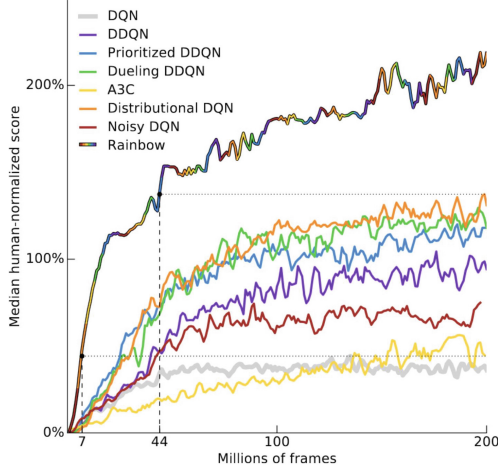


Fig. 1. Comparison between the rainbow agent and reinforcement learning methods.

To satisfy all 3 missions, the agent needs more information from the image given by the depth camera. Therefore the classical deep architecture Q network should be transformed according to the needs of the problem. As result, instead of convolutional neural network, tests were done using 2 different cores: using a) Multilayer Perceptron Neural Network and b) Hybrid neural network which consists of: Convolutional Neural Network Core, Core Perceptron Multilayer Neural Network, Fully cohesive layer for combining features from cores.

The idea behind the multi-layer Perceptron neural network is that it can process the information it receives from the depth camera along with the necessary information about its location in the environment and the location of the target. The state of S_t agent at time t is a vector of: i) The preprocessing image of depth D_t , ii) The position vector of the agent $P_t = \langle x_t \ y_t \ z_t \rangle$, iii) The position vector of the agent $P_{t-1} = \langle x_{t-1} \ y_{t-1} \ z_{t-1} \rangle$ in the previous time, iv) The position vector of the target $\mathcal{T} = \langle x_{\mathcal{T}} \ y_{\mathcal{T}} \ z_{\mathcal{T}} \rangle$ and v) A scalar l_t floating point that informed the agent how many actions he has done up to time t . The agent could do up to 4 times more actions than the optimal estimated number of actions. The optimal estimated number of actions was obtained by summing the number of steps that the agent pays if he always chooses a reducing action among its distance from the target.

Preprocessing on the depth image consists of a basic image transformation in gray scale followed by a transformation to smaller dimensions and finally a crop of the result to its center so that it has the same dimension the length by the width. Thus, the depth image is transformed from $3 \times 210 \times 160$ to 84×84 . Finally, the depth image is normalized to the range $[0, 1]$ by dividing it by the maximum possible value which is 255. The result is the matrix D_t .

The matrix D_t is transformed into a column vector D_t^{fl} and then joins with remaining input parameters $P_t, P_{t-1}, \mathcal{T}$ and l_t shaping its state agent S_t at time

t . The available actions are 6. In the experiments of the work, the displacement step was 1 m. The payoff at each step was given by Eq. (22).

$$R_t = \begin{cases} -100 & \text{in the event of a collision with an obstacle,} \\ 100 & \text{upon completion of the mission,} \\ -10 & \text{if } l_t \leq 0, \\ -10 & \text{in case of early landing} \\ \mathcal{D}_{t-1} - \mathcal{D}_t & \text{otherwise} \end{cases} \quad (22)$$

In Eq. (22), the quantity D_t is the distance of the agent from the target in time moment t . The same quantity D_{t-1} is the distance of the agent from the target in time $t - 1$. The distance is calculated according to Eq. (23).

$$\mathcal{D}_t = P_t - \mathcal{T} \quad (23)$$

4 Experimental Results

During the experiments the performance of the agent is evaluated across 16 different targets. To create a simulation interface, the software AirSim was used². The results for the Multilayer Perceptron Neural Network are shown in Table 2. Moreover, the training evaluation is shown in Fig. 2. The results for the Hybrid Neural Network are shown in Fig. 3.

Table 2. Percentage of success/failure for action selection and mission completion.

Evaluation	Result	Percentage
Percentage of action selection as per target	Success	97.3%
Percentage of action selection as per target	Failure	2.7%
Percentage of success across 16 missions	Success	93.8%
Percentage of success across 16 missions	Failure	6.2%

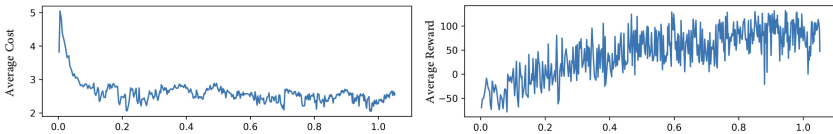


Fig. 2. Results of the rainbow agent as per average cost and average reward.

The Perceptron multi-layer neural network core agent has been shown to work very satisfactory. However, it showed inability to make a decision in cases where was faced with an obstacle of great dimensions. To solve the problem, a hybrid

² <https://microsoft.github.io/AirSim/>.

architecture was tried. The entry of the new Q model consists of the matrix D_t and the vector V_t which is the concatenation of P_t, P_{t-1}, T and l_t . The model Q consists of a deep convolutional neural network for efficient extraction features from the D_t matrix and from a deep multi-layer neural network Perceptron for feature extraction from vector V_t . The characteristics of 2 cores are joined at a common plane, called the union plane. The level union is an input layer for a deep multilayer Perceptron neural network, the association model.

The computational resources required to train the agent were very high more than available. Notable was the maximum possible memory size of repeating experiences according to the system memory used for the experiments. The maximum experience memory size was 50,000 samples. Also, for for speed reasons, prioritized replay memory was implemented using segment tree³. The partition tree belongs to the family of binary trees and therefore the agent instead of 50,000 items in memory could only load $2^{\lceil \log_2 50.000 \rceil} = 2^{15} = 32.768$ experiences. As a result, model training was not able to be completed fully.

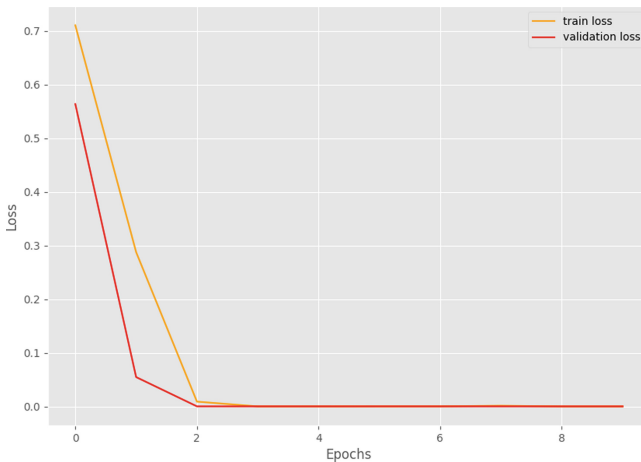


Fig. 3. Training of multi-layer neural network as per feature extraction from V_t .

After training the 2 kernels, it remained to train the union model. Keeping the parameters of the 2 cores fixed, during training they were adjusted only the union model parameters and the parameters in the value paths and advantage respectively. As in the previous experiments, it is stated that available computing resources were extremely limited for his needs problem and how the indicative number of training cycles, as can be seen in Fig. 1, is 44,000,000 cycles instead of the 450,000 training cycles run by model.

5 Conclusions and Future Work

Autonomous UAV navigation is an emerging concept in robotics where most environments are unknown. Therefore, techniques for automated navigation are

³ https://github.com/segment_tree.py.

increasing including modern AI systems that can analyse huge volumes of data to uncover patterns leading to successful solutions. Using reinforcement learning, the software learns a function that maximises the reward within an environment. Rainbow-type agents were chosen because of their exceptional performance in limited-action situations. As noted, an agent with high performance and optimum policy was built by using preprocessed image as input for pattern recognition from a tensor allowing a deep convolutional neural network to train using more data. Moreover, a better strategy may be established than the one utilised here to build the agent using a multi-layer neural network core like Perceptron as the current computing resources are insufficient to perform such tasks.

Future directions of this work include the construction of a model that utilizes a set of sequential images at its input, which contain all the information the agent needs in terms of the device, in terms of surroundings and in terms of the goal. Perhaps a convolutional model will emerge which can be designed to detect many different classes within an image to perform better than the convolutional core used for the experimental evaluation. Ultimately, fine-tuning optimizations include the use of more sophisticated algorithmic choices along with this work.

References

1. Karras, A., Karras, C., Giotopoulos, K.C., Giannoukou, I., Tsolis, D., Sioutas, S.: Download speed optimization in P2P networks using decision making and adaptive learning. In: Daimi, K., Al Sadoon, A. (eds.) *Proceedings of the ICR'22 International Conference on Innovations in Computing Research. ICR 2022. Advances in Intelligent Systems and Computing*, vol. 1431, pp. 225–238. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14054-9_22
2. Amer, K., Samy, M., Shaker, M., ElHelw, M.: Deep convolutional neural network based autonomous drone navigation. In: *Thirteenth International Conference on Machine Vision*, vol. 11605, pp. 16–24. SPIE (2021)
3. De Asis, K., Hernandez-Garcia, J., Holland, G., Sutton, R.: Multi-step reinforcement learning: a unifying algorithm. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32 (2018)
4. Hessel, M., et al.: Rainbow: combining improvements in deep reinforcement learning. In: *Thirty-Second AAAI Conference on Artificial Intelligence* (2018)
5. Mnih, V., et al.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533 (2015)
6. Roghair, J., Niaraki, A., Ko, K., Jannesari, A.: A vision based deep reinforcement learning algorithm for UAV obstacle avoidance. In: Arai, K. (ed.) *IntelliSys 2021. LNNS*, vol. 294, pp. 115–128. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-82193-7_8
7. Siciliano, B., Khatib, O., Kröger, T.: *Springer Handbook of Robotics*, vol. 200. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-30301-5>
8. Um, J.S.: *Drones as Cyber-Physical Systems*. Springer, Singapore (2019). <https://doi.org/10.1007/978-981-13-3741-3>
9. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30 (2016)