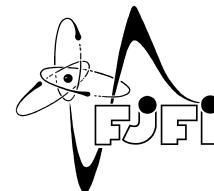




CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Nuclear Sciences and Physical Engineering



Exploration in Knowledge Transfer

Prozkoumávání prostoru v úloze přenosu znalostí

Master Thesis

Author: **Bc. Adam Jedlička**
Supervisor: **Dipl.-Eng. Tatiana V. Guy, Ph.D.**
Consultant: **Ing. Marko Ruman**
Academic year: **2022/2023**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Adam Jedlička
Studijní program: Aplikované matematicko-stochastické metody
Název práce (česky): Prozkoumávání prostoru v úloze přenosu znalostí
Název práce (anglicky): Exploration in Knowledge Transfer

Pokyny pro vypracování:

- 1) Seznamte se s existujícími přístupy k řešení problému „exploration – exploitation“ ve zpětnovazebním učení (RL).
- 2) V návaznosti na výzkumný úkol navrhnete vhodné řešení problému „exploration – exploitation“ pro úlohu přenosu znalostí.
- 3) Navržené řešení implementujte v MATLABu nebo Pythonu a ověřte na simulovaných datech.
- 4) Vyhodnoťte získané výsledky.

Doporučená literatura:

- 1) M. Gimelfarb, A. Barreto, S. Sanner, C.-G. Lee (2021). Risk-aware transfer in reinforcement learning using successor features, arXiv:2105.14127.
- 2) D. Pathak, D. Gandhi and A. Gupta (2019). Self-Supervised Exploration via Disagreement. In Proceedings of Machine Learning Research 97:5062-5071
- 3) I. Osband, B. Van Roy and Z. Wen (2016). Generalization and Exploration via Randomized Value Functions, ICML'16: Proceedings of the 33rd Int. Conference on International Conference on Machine Learning.

Jméno a pracoviště vedoucího diplomové práce:

Ing. Tatiana Valentine Guy, Ph.D.

Ústav teorie informace a automatizace AV ČR, v.v.i., Pod Vodárenskou věží 1143, 182 00
Praha 8-Libeň, ,

Jméno a pracoviště konzultanta:

Ing. Marko Ruman

Ústav teorie informace a automatizace AV ČR, v.v.i., Pod Vodárenskou věží 1143, 182 00
Praha 8-Libeň

Datum zadání diplomové práce: 28.2.2022

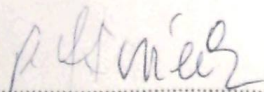
Datum odevzdání diplomové práce: 5.1.2023

Doba platnosti zadání je dva roky od data zadání.

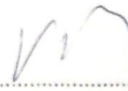
V Praze dne 28.02.2022



garant oboru



vedoucí katedry



děkan



Acknowledgment:

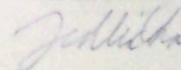
I would like to thank Dipl.-Eng. Tatiana V. Guy Ph.D. and Ing. Marko Ruman for their expert guidance. Partial support of MŠMT project LTC18075 is acknowledged.

Author's declaration:

I declare that this Master thesis is entirely my own work and I have listed all the used sources in the bibliography.

Prague, December 28, 2022

Adam Jedlička



Název práce:

Prozkoumávání prostoru v úloze přenosu znalostí

Autor: Bc. Adam Jedlička

Obor: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: Dipl.-Eng. Tatiana V. Guy Ph.D.

Konzultant: Ing. Marko Ruman

Abstrakt: Tato práce je zaměřená na prozkoumávání prostoru v přenosovém učení, i.e. přenos znalostí z úkolu naučeného v minulosti, použité pro nový úkol za účelem zrychlení učícího se procesu nebo nalezení lepšího řešení. Ojevování-využívání uvažuje novou zkušenost, kterou by agent mohl získat pomocí náhodné akce. Ojevování je obzvláště důležité v metodách zpětnovazebního učení, kdy může objevit výhodnější stavy. Cílem této práce je vybrat několik objevovacích metod a použít je pro přenos znalostí pro hluboké zpětnovazební učení. Konkrétně jsme použili algoritmus hlubokého cíleného přenosu Q-učení v kombinaci s ϵ -greedy, Boltzmannovým a UCB prozkoumávacím algoritmem a zjistili jsme, že fungují relativně dobře. Výsledky však zdůraznily nekonzistenci výkonu několika algoritmů ve srovnání mezi sebou v souvislosti se zadaným úkolem.

Klíčová slova: Přenos znalostí, Q-učení, Metody prozkoumávání prostoru, Hluboký cílený přenos Q-učení.

Title:

Exploration in Knowledge Transfer

Author: Bc. Adam Jedlička

Abstract: This thesis focuses on exploration in knowledge transfer, i.e. when knowledge learned in past task used for a new task with an aim to accelerate the learning process or find a better solution. The exploration-exploitation balances the novel experience that the agent may obtain by applying a random action. In particular the exploration is of high importance in reinforcement learning methods when it can discover more advantageous states. The goal of this thesis is to select several exploration methods and use them for knowledge transfer in deep reinforcement learning. We have used a deep target transfer Q-learning in combination with ϵ -greedy, Boltzmann and UCB exploration algorithms and have found out that they work relatively well. However, the results highlighted the inconsistency of several algorithms in comparison with each other with regard to different tasks.

Key words: Knowledge transfer, Q-learning, Exploration methods, Deep target transfer Q-learning

Contents

Introduction	7
1 Preliminaries	9
1.1 Notions and definitions	9
1.2 Markov Decision Process	9
1.3 Reinforcement Learning	11
1.3.1 Q-learning	11
1.3.2 Deep Q-learning	11
2 Neural Networks	14
2.1 Basic introduction	14
2.2 Activation functions	15
2.3 Convolution and Pooling Layers	15
2.4 Epochs, Batches and Dropout rate	16
2.5 Calculating Loss	16
2.6 Training	17
3 Knowledge Transfer and exploration methods	18
3.1 Knowledge transfer	18
3.2 Target transfer Q-learning (TTQL)	18
3.3 Deep TTQL	20
3.4 Exploration methods	20
3.5 ϵ -greedy method	20
3.6 Boltzmann exploration method	21
3.7 Upper confidence bound method	21
4 Implementation for a virtual drone	22
4.1 States and actions	22
4.2 Rewards	23
4.3 Deep Q-learning and knowledge transfer	25
4.4 Parameters used	26
4.5 ϵ and λ decrease	26
4.6 Overall algorithm	27
5 Experiments	28
5.1 Experiment 1	29
5.2 Experiment 2	31
5.3 Experiment 3	33
Conclusion	35

Introduction

In this section we will outline the basic ideas and motivations behind this thesis. This thesis will focus on exploration methods in knowledge transfer between tasks, that are modeled by Markov Decision Processes (MDPs). First let us clarify what we mean by knowledge. The amount of knowledge we have represents how much of a model (MDP) we know. For example an experienced driver going through the city knows, where to turn, in order to get to his final destination efficiently. Therefore he knows how to decide on each crossroad and we say, that he has certain amount of knowledge about the city (Model) for performing his task. We say that driver is an agent, but generally the agent can be anything - human, algorithm or machine solving some kind of task. Knowledge transfer or transfer learning means, that knowledge accumulated in past task (source task) will be partially used by the agent in solving a new task. Note that generally the transferred knowledge can be learned by other agent in the different, but similar task (target task). This is useful, because it can speed up the learning process in target task and the agent does not need to learn the target task from scratch. The agent solves the task with reinforcement learning and this requires exploration.

Thus the topic can be broken down into three parts. Reinforcement learning (RL) [7], knowledge transfer and exploration methods.

In RL the agent has neither the environmental model or the reward model. The agent is given reward (reinforcement) as a reaction of the environment to his actions. The advantage of this type of learning is that the agent can learn complex tasks by trial and error and given enough time he can even find nearly optimal solution. The example of task for RL would be teaching the agent to find a path through the maze. A basic RL algorithm is called Q-learning. If we come back to the driver example, we could call each crossroad the state in which our driver is. The town has finite number of such states and information about all of them can therefore be memorised. In a case where this number of states is very large (for instance image sequence) or infinite we would have to create a model to determine information about the state instead. In this case we could use more sophisticated version of Q-learning called deep Q-learning. The difference is that this algorithm uses neural network to determine in which state the agent is, or some other attribute connected to Q-learning. This type of RL algorithm will be used as basis for transfer learning (TL) algorithms in this thesis.

We have various types of TL options for Q-learning and Deep Q-learning. They can be divided into three groups - representation transfer, parameter transfer and instance transfer [3]. Instance transfer methods hopes to achieve speed-up of learning a target task by gathering samples of knowledge from a source task and using them in the target task. Representation transfer focuses on some general characteristics, that are preserved in both tasks (feature transfer) and parameter transfer focuses on changing the parameters of algorithm used for learning the target task based on knowledge of parameters of the source task.

Parameter connected with exploration is called an exploration rate. It is an important parameter of both RL and TL methods. This parameter determines how much agent is willing to try new unexplored states compared to going for the most beneficial states that he knows of. Generally we would want the agent to explore more new states when he has less knowledge and less new states when he has more knowledge and that is where the exploration algorithm comes in. These methods allow us to modify this parameter during the learning process. There are multiple approaches that we will consider. Namely ϵ -greedy method, Boltzmann exploration method and Upper confidence bound method.

The thesis implemented knowledge transfer method with previously mentioned exploration methods and compared it with Q-learning method in order to find out, if these methods are effective for knowledge transfer in specific type of task. The considered task is virtual drone in virtual environment. The drone aims to navigate

through an environment based on images in its camera. The main focus of this thesis is to examine various exploration methods in TL, and to find the most suitable and efficient one.

The thesis outline is as follows. Chapter 1 introduces necessary notions and definitions, recalls the theory of Markov decision processes, reinforcement learning, Q-learning and deep Q-learning. Chapter 2 describes details of neural networks. Chapter 3 briefly summarizes technique of knowledge transfer and the three most broadly used exploration methods. Chapter 4 focuses on describing implementation of the RL and TL on the model of the virtual drone. Chapter 6 summarizes the configuration and presents the obtained results of the experiments. Chapter Conclusions summarises main findings, analyses them and outlines open questions.

Chapter 1

Preliminaries

1.1 Notions and definitions

This section defines basic notions and definitions used in the thesis. We will work with discrete time t that will appear as sub index of respective variables. Calligraphic capital letters (\mathcal{A}) denote sets and calligraphic small letters (a) denote elements of those sets, unless stated otherwise. Letters in bold font (ϕ) denote vectors and matrices. In Table 1.1 we define terms, that are used throughout this text.

Name	Notation	Meaning
Set of possible actions	$\mathcal{A} \subset R$	A discrete finite set of actions
Set of possible states	$\mathcal{S} \subset R$	A discrete set of observable states.
Time	$t \in N$	Discrete time
Action	$a_t \in \mathcal{A}$	Action at time t
State	$s_t \in \mathcal{S}$	State of the environment at time t
Policy	$\pi : \mathcal{S} \rightarrow \mathcal{A}$	$\pi(a_t s_t)$ is a function assigning actions to states
Discount factor	$\gamma \in (0, 1]$	Determines the importance of future rewards.
Transition function	$p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$	$p(s_{t+1} s_t, a_t)$ is probability of state s_{t+1} when applying action a_t in state s_t , $\sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1} s_t, a_t) = 1, \forall s_t \in \mathcal{S}, \forall a_t \in \mathcal{A}$
Reward function	$r : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow R$	Reward $r(s_{t+1}, a_t, s_t)$ is received when transitioning from state s_t to state s_{t+1} by action a_t . Here, we distinguish $r(s_{t+1}, a_t, s_t)$ and r_t . The first one denotes reward function and the last denotes value of reward at time t .
Q - function	$Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow R$	Function $Q^\pi(s_t, a_t)$ determines value of state-action pair when policy π is executed.
Discrete mean value	$E[X] = \sum_{i \in I} p_i x_i$	Mean value of discrete random variable X, where p_i is probability of $X = x_i$ and x_i are values of X.

Table 1.1: Notations and definitions

Note, that states and actions do not necessarily have to be subsets of R .

1.2 Markov Decision Process

This section focuses on defining Markov Decision Process and terms associated with it. Firstly, let us define the general problem of decision making. We have an agent and an environment. The agent observes environment states and has a goal to influence them. Agent takes action in order to reach a goal specified by the task that is defined for him. The degree to which he reached his goal is measured by the reward agent obtains after each action. Basic scheme of this interaction is shown in Figure 1.1. At each time t the agent observes environment state s_t and chooses action a_t . The action changes environment state to state s_{t+1} and the agent receives reward r_{t+1} , expressing the value of this transition $(s_t, a_t) \rightarrow s_{t+1}$ for the agent.

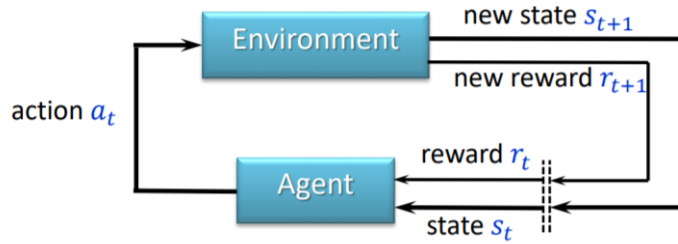


Figure 1.1: Interaction of the agent with environment in MDP

The agent and the environment obey Markov property. This property means, that each state at any time depends only on the previous state and action. Decision process with this property is called Markov decision process (Definition 1.2.1).

Definition 1.2.1. *Let us assume that environment follows Markov property. Then MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, p, \gamma, r)$, see Table 1.1.*

Policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ specifies action choice for each state. The agent has a goal to select policy that maximizes its expected reward. The following definition introduces an important function.

Definition 1.2.2. *The value function of state under policy π , $V^\pi(s)$, is defined as $V^\pi(s) = E^\pi[\sum_t \gamma^t r(s_{t+1}, a_t, s_t) | s = s_t]$, where $r(\cdot, \cdot, \cdot)$ is a reward function and γ is a discount factor.*

The value function shows how good is given state s for the agent. It represents the value agent could get if he uses policy π in states s . A finite MDP has a unique V^{opt} with regard to finding solution of (1.1) which is unknown and at least one deterministic optimal policy. Solving an MDP means to find an optimal value function V^{opt} and optimal policy π^{opt} . Another way to evaluate the state for a given policy is via so-called action-value function or Q-function $Q^\pi(s, a)$

Definition 1.2.3. *Q-function is defined as $Q^\pi(s, a) = E^\pi[\sum_t \gamma^t r(s_{t+1}, a_t, s_t) | s = s_t, a = a_t]$, where $r(\cdot, \cdot, \cdot)$ is the reward function and γ is a discount factor.*

Both, value function V^π and Q-function Q^π , obey Bellman equations (1.1) and (1.2) [6]. These equations separate value functions into immediate reward part and discounted future reward part [16] and express a relationship between the value of a state and the values of its successor states

$$V^{opt}(s) = \max_{a \in \mathcal{A}} \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1} | s_t, a) [r_t + \gamma V^{opt}(s_{t+1})] \quad (1.1)$$

$$Q^{opt}(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1} | s_t, a_t) (r_t + \gamma \max_{a \in \mathcal{A}} Q^{opt}(s_{t+1}, a)). \quad (1.2)$$

The optimal policy maximizes the overall accumulated reward and is derived by (1.3).

$$\pi^{opt}(s_t) = a_t \in \operatorname{argmax}_{a \in \mathcal{A}} Q^{opt}(s_t, a_t). \quad (1.3)$$

It is easy to see that both the value function and the Q functions represent the discounted cumulative reward the agent obtains if following policy π . Unlike the value function, the Q-function also considers action. Both functions are related as follows $V(s) = \max_{a \in \mathcal{A}} Q(s, a)$. Equations (1.1) and (1.2) can be solved by dynamic programming. Dynamic programming [19] is the optimisation framework for sequential decisions when transition model is known. It simplifies a large problem by dividing it into smaller subproblems that can be solved recursively. These subproblems need to be overlapping i.e. solution of one is used for solution of another subproblem. Another requirement is that optimal solution of the whole problem can be used to gain optimal solution of its subproblems [19].

1.3 Reinforcement Learning

In practice transition model and/or reward model might not be known. Reinforcement learning (RL) [7] solves MDP with unknown transition model p and reward model r (Definition 1.2.1). The basic idea of how reinforcement learning avoids the need for knowing transition and reward model is following. At each time t agent takes action a_t in a form of reward. Initially, he does not need to know how the action influences in which state he is going to end up he just measures the outcome of action a_t . i.e. he does not need to know transition model p but learns it over time. Similarly, the reward is given by the environment at each transition so the agent maps the reward given by each transition given by the environment. The higher the reward agent obtains the more beneficial action. Eventually, over time, agent approximates transition and reward models based on information gathered from exploring different transitions. Naturally, the richer the experience of agent the more precise model of the environment the agent obtains. In RL agent balances between two kinds of actions, exploiting and exploring. The first maximizes immediate reward and the second explores states in hope to find higher reward in later steps. If the agent explores all the time, there is a risk that the cumulative reward will be low. If the agent uses only greedy strategy, he might also get stuck on low reward. The key question of RL is what portion of time should be dedicated to exploration and what portion of time should be dedicated to exploitation (based on knowledge accumulated so far).

1.3.1 Q-learning

Q-learning is a model free off-policy RL algorithm and therefore is used when we do not have environment model at disposal. This algorithm works with Q-function (Definition 1.2.3). Q-function satisfies equation (1.3). The core of Q-learning is based on Bellman equation for Q-function (1.2). The Q-learning algorithm updates Q-function via the weighted average of the current value of Q-function, Q^{old} , and the maximum expected future reward

$$Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha[r_t + \gamma \max_{a \in \mathcal{A}} Q^{old}(s_{t+1}, a) - Q^{old}(s_t, a_t)], \quad (1.4)$$

where $\alpha \in (0, 1]$ is learning rate that determines weight of newly gained information and expression in brackets behind α is called temporal difference. If learning rate is close to 1, the agent takes into account only recently gained information, if it is close to 0, the agent ignores it i.e. learns nothing. Finally, when algorithm converges, we find optimal policy given the knowledge gathered using equation (1.3). An algorithm for Q-learning is presented below.

Algorithm 1 - Q-learning

input - initial Q-function Q^{old} , reward function r , learning rate α , factor γ , terminal state s_{term} , initial state s_{start} and max_iter as maximum number of iterations of t .
set initial state $s_0 = s_{start}$
set time $t = 0$
while $t < max_iter$
 if $s_t = s_{term}$
 set s_t to s_{start}
 end if
 choose action a_t and get following state s_{t+1} .
 $Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q^{old}(s_{t+1}, a_{t+1}) - Q^{old}(s_t, a_t))$
 $Q^{old}(s_t, a_t) = Q^{new}(s_t, a_t)$
 $t = t + 1$
end while

1.3.2 Deep Q-learning

In practical applications we often deal with large or continuous state space and Algorithm 1 is not viable due to computational complexity. In that case deep Q-learning may be of help. The algorithm has similar structure as simple Q-learning and update rule is also derived from Bellman equation.

In contrast with regular Q-learning, where we save Q^{old} values into a table, Deep Q-learning uses neural network to estimate them. Neural networks will be described in further chapters, however the basic idea is that they function as a model with which we assign each state its Q-values (Figure 1.2) [9]. The information about the Q-function is saved in parameters of neural network. Pseudocode is shown in Algorithm 2 [12]. The learning rate α mentioned in Q-learning is removed and replaced by similar parameter in neural network training process mentioned in the next chapter instead. The meaning of other parameters is the same as in Q-learning. Details of neural networks such as loss function and network training will be discussed in the next chapter. Deep Q-learning also includes several new parameters such as *train interval*, *batch* and *replay memory*.

Train interval is a number of iterations, after which training of the neural network happens i.e if *train interval* is equal to 10, training happens every 10 iterations. *Replay memory* is a set of transitions (s_t, a_t, s_{t+1}, r_t) recorded in a memory and *batch* is a randomly selected sample of transitions from *replay memory*. Size of this subset is denoted as *batch size* in pseudocode in Algorithm 2. The second index j in $(s_{t,j}, a_{t,j}, s_{t+1,j}, r_{t,j})$ refers to the j -th position of a transition in a batch chosen at time t .

The reason for using *replay memory* is, that if we used consecutive samples instead, these samples might be correlated. By using larger replay memory this problem is avoided.

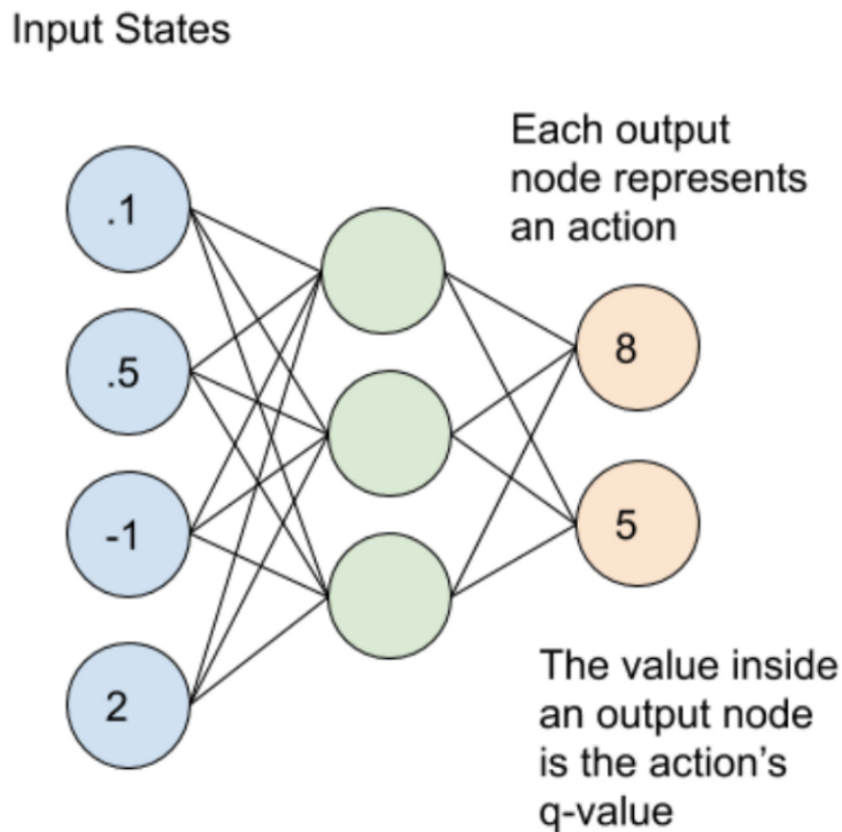


Figure 1.2: Neural network example mapping states to Q-values corresponding to each action [9]

Algorithm 2 - Deep Q-learning

input - initialize replay memory D , gamma factor γ , terminal state s_{term} , starting state s_{start} , *batch size*, *train interval* n , maximum number of iterations max_iter and neural network estimating Q-function Q^{old} with random network parameters. Set $t = 0$ and $s_0 = s_{start}$.

while $t < max_iter$

if $s_t = s_{term}$

set s_t to s_{start}

end if

 choose action a_t based on estimate given by neural network Q^{old} and get following state s_{t+1} and reward r_t .

 store transition (s_t, a_t, s_{t+1}, r_t) into D

if t is divisible by n

 sample random *batch* from D

for $j = 1:batch\ size$

$$Q^{new}(j) = \begin{cases} Q^{old}(s_{t,j}, a_{t,j}) + (r(s_{t+1,j}, a_{t,j}, s_{t,j}) + \gamma \max_{a_{t+1}} Q^{old}(s_{t+1,j}, a_{t+1}) - Q^{old}(s_{t,j}, a_{t,j})) & \text{if } s_{t,j} \text{ is not terminal} \\ r_{t,j} & \text{otherwise} \end{cases}$$

$Q^{old}(j) = Q^{old}(s_{t,j}, a_{t,j})$

end for

 Train network on Q^{new} and Q^{old} by minimizing appropriate loss function

end if

$t = t + 1$

end while

Chapter 2

Neural Networks

As mentioned in the previous chapter, the neural networks are necessary mathematical model for modelling the Q-function in Algorithm 2. This chapter is focused on theory behind the neural networks and their training process.

2.1 Basic introduction

We can describe a single layer neural network with Figure 2.1 [8]. The purpose of this section is to create predictive model. We have the collected data of input variables X_1, X_2, X_3, X_4 and we will use them to calculate function $f(\mathbf{X})$, that predicts response Y [8]. The system of predicting response is slightly similar to classical

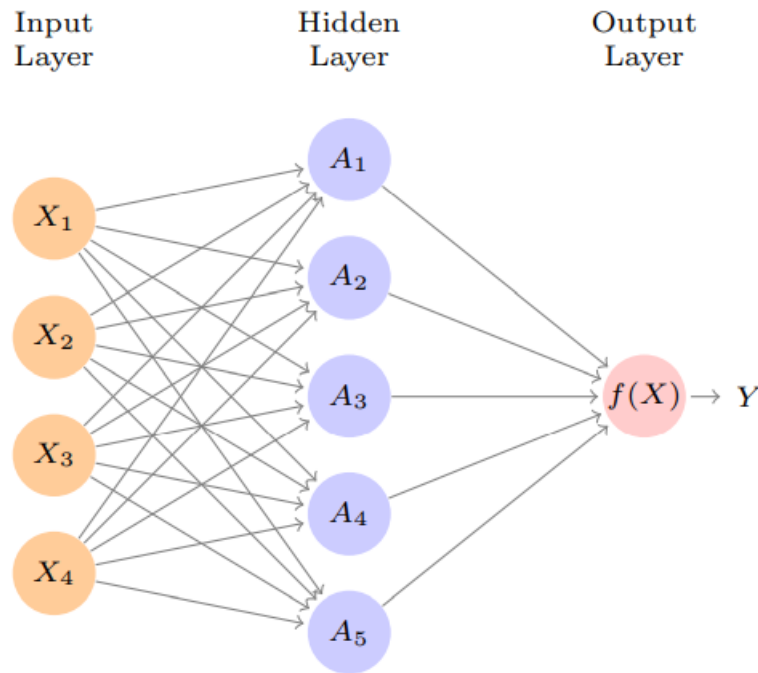


Figure 2.1: Single layer neural network

regression, however the function $f(\mathbf{X})$ might not be linear. We start the mathematical description with activation function. This function describes the value that comes to A_k neuron of hidden layer. The value of A_k is then given by equation (2.1)

$$A_k = g(\omega_{k,0} + \sum_{j=1}^c \omega_{k,j}X_j), \tag{2.1}$$

where c is a number of input variables (in case of Figure 2.1 $c=4$), $\omega_{k,j}$ network parameters associated with the k -th neuron and g is an activation function. Finally the value of $f(\mathbf{X})$ is described by equation (2.2)

$$f(\mathbf{X}) = b_0 + \sum_{k=1}^K b_k A_k = b_0 + \sum_{k=1}^K b_k g(\omega_{k,0} + \sum_{j=1}^p \omega_{k,j} X_j), \quad (2.2)$$

where K is number of hidden units (Figure 2.1), b_k are additional network parameters associated with these units and b_0 is parameter that plays the role of constant. Neural network, of course, does not have to have a single hidden layer but can have multiple hidden layers.

2.2 Activation functions

Nowadays the most commonly used function is called "Rectified linear unit - ReLU" given by equation (2.3) [8]

$$g(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases} \quad (2.3)$$

This activation function restricts the output to positive real numbers. Another option is simple linear function (2.4). The range of this activation function are the real numbers

$$q(z) = az + b. \quad (2.4)$$

In case we choose linear function we get classical regression model (2.5)

$$f(\mathbf{X}) = b_0 + \sum_{k=1}^K b_k \omega_{k,0} + \sum_{j=1}^p X_j \sum_{k=1}^K \omega_{k,j} b_k. \quad (2.5)$$

This means, that in case we would use Linear activation function for every layer in the network, we would get just a linear regression model with large amounts of parameters. These activation functions are usually the same for each layer of neurons. Therefore we call these layers as Relu or Linear layers.

2.3 Convolution and Pooling Layers

These layers are mainly utilized for classification of images. Therefore, instead of input variables X_i and output $f(\mathbf{X})$ used previously, the input is an image represented by tensor M and output is represented by tensor O . Convolution layers are used to find features in the image and pooling layers are used to condense large image into a smaller one [8].

Let the input colored image be represented by $m \times n \times 3$ (m and n represent the size of and image in this section) tensor M of pixels with values corresponding to brightness of each pixel (with 3 channels corresponding to red, green and blue channel). Now let us consider filter that is represented by a $k \times l \times 3$ tensor F , where the dimensions of F are lower or equal than the dimensions of M . For simplicity let us call three coordinates of the previously mentioned tensors x, y and z . Now let us pad the matrix M with additional matrix elements with value of 0 in such a way, that there are $\lfloor \frac{k}{2} \rfloor$ (where $\lfloor \cdot \rfloor$ is a floor function) additional elements on each side in the x coordinate and $\lfloor \frac{l}{2} \rfloor$ on the top and on the bottom in the y coordinate. This way we get a matrix M_2 and such matrix is called a padded matrix of M . Now we can define a convolution of $M_2 * F$ with the padding by equation (2.6)

$$O(i, j) = \sum_{k=1}^3 \sum_{s=1}^k \sum_{p=1}^l M_2(i - \lfloor \frac{k}{2} \rfloor + s, j - \lfloor \frac{l}{2} \rfloor + p, k) F(s, p, k). \quad (2.6)$$

We can see, that in this case we get a tensor that is of the same x and y coordinates as an input, but lacks the z coordinate. In case we do not use the padding. We use the same formula but we limit the i and j coordinates. The coordinates must satisfy conditions $k < i < m - k$ and $l < j < n - l$. Result then has dimensions $m - 2k \times n - 2l$. Another way we can use the convolution layers to reduce the dimensions is to apply a "strides" parameter to

it. This parameter chooses just certain pixels from the image to be presented in the output. As an example if our strides parameter would be $[a, b]$ we would only use every $a - th$ pixel in the x coordinate and every $b - th$ pixel in the y coordinate. This method significantly reduces the amount of model parameters and thus avoids overfitting. In the convolution layer we use pixels as input neurons and elements of filter are used as the parameters for each separate neuron in the output. This way we get an output image. However we can use more than one filter in the convolution layer. Output in this case will have an additional dimension representing filter used. For example, if we use c amount of filters in the convolution layer, the output O will have dimensions of $m \times n \times c$.

Convolution layer is often connected to the ReLu layer after convolution is done. Sometimes this is considered as a part of the convolution layer and sometimes it is taken as a separate layer, in which case it is called a detector layer [8].

Pooling layer provides a way to condense a larger image into a smaller summary image, while keeping the most important features of the image [8]. The type of the pooling layer that will be used is called a Max-pooling layer and compresses the image in a following way. For each pixel in the input we will find maximum in his surrounding $k \times l$ area as an output at this point. Next, we will apply similar strides method as in the convolution layer to compress the output [8].

2.4 Epochs, Batches and Dropout rate

Now, we will introduce epochs and batches. In practice we take a data set and split it into training and testing sets. Next we take training set and choose a batch subset of fixed size and train our network on it. This network has no information about the parameters in the beginning (they are set randomly). Once this happens, we call this episode an epoch and we are left with the parameters on certain values. For each epoch we choose a different batch and as the initial parameter values we use the parameter values that were gained from previous epoch. With increasing number of the epochs we should converge to certain values of network parameters. Next we will introduce layer dropout and dropout rate. This is a process during which we randomly dropout certain percentage of neurons in hidden layers and thus temporarily removing them from the network in beginning of each epoch. In the next epoch different neurons are chosen. The rate refers to percentage of neurons that we leave out in each hidden layer. This process helps with overfitting by adding some noise into data and improves the performance [8].

2.5 Calculating Loss

When fitting the network parameters for response we try to minimize a value of loss. This loss can take take for example following shapes - mean squared error loss [8] (2.7) and Huber loss (2.8)

$$L_{MSE} = (y - f(\mathbf{X}))^2 \quad (2.7)$$

$$L_{Huber} = \begin{cases} \frac{1}{2}(y - f(\mathbf{X}))^2 & \text{if } |y - f(\mathbf{X})| \leq \delta \\ \delta(|y - f(\mathbf{X})| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \quad (2.8)$$

In these equation y is datapoint of test set and $f(\mathbf{X})$ is the value neural network predicts in this datapoint. In case of having more input variables, the loss is averaged over the number of these variables.

Each of the losses have some advantages and disadvantages. Mean squared error loss has the advantage of having no local minima. However, it does not handle outliers well as it squares their error. On the other hand, Huber loss handles outliers well at the cost of a larger computational complexity [25].

2.6 Training

This section explains the process of optimization of neural network model. As we can see in (2.2) the model $f(\mathbf{X})$ has network parameters b_k, b_0 and $\omega_{k,j}$. Next, let us put all the network parameters from equation (2.2) into a vector \mathbf{w} . Let us denote the size of this vector as p . The purpose of training of neural network is to estimate the values of these network parameters to gain a mathematical model $f(\mathbf{X})$ that can predict \mathbf{Y} (Figure 2.1). In deep Q-learning we will use this model to predict a value of the Q function.

Next, we will introduce an algorithm [14] which will be used to update network parameters based on the test data, predicted responses and the network parameters learned in past epoch. This algorithm is called Adaptive moment estimation (ADAM). It calculates first moment \mathbf{m} and second central moment \mathbf{s} of the gradient of the activation function with respect to its parameters. Afterwards it calculates new values of the parameters. This cycle happens k times, where k is the size of the training batch. Let us denote vector of collected responses as \mathbf{Y} and vector of collected data for i -th input variable as \mathbf{X}_i (out of I input variables). The aforementioned algorithm is described in Algorithm 3. Recommended values for parameters β_1, β_2 and ρ are 0.9, 0.99 and 10^{-8} respectively. These values are going to be used everywhere, where this algorithm is utilized.

Algorithm 3 - Adaptive Moment Estimation

input - initialize network parameters of past epoch \mathbf{w} , Collected data of input variables \mathbf{X}_i for $\forall i \in I$, collected response data \mathbf{Y} , loss function $g(X_1, X_2, \dots, X_m, y, \mathbf{w})$, parameters ρ, β_1, β_2 and learning rate l .

initialize moments $\mathbf{m} = 0$ and $\mathbf{s} = 0$ (size of these vectors is p).

calculate $\mathbf{h}(X_1, X_2, \dots, X_m, y, \mathbf{w}) = \text{grad}(g(X_1, X_2, \dots, X_m, y, \mathbf{w}))$ with respect to \mathbf{w} .

for $n = 1 : k$

$x_{n_i} = X_{n_i}$ for $\forall i \in I$

$y_n = Y_n$

$\mathbf{m} = \beta_1 \mathbf{m} - (1 - \beta_1) \mathbf{h}(x_{n_1}, x_{n_2}, \dots, x_{n_m}, y_n, \mathbf{w})$

$\mathbf{s} = \beta_2 \mathbf{s} - (1 - \beta_2) \mathbf{h}^2(x_{n_1}, x_{n_2}, \dots, x_{n_m}, y_n, \mathbf{w})$

$\mathbf{m}_{\text{norm}} = \frac{\mathbf{m}}{(1 - \beta_1^n)}$

$\mathbf{s}_{\text{norm}} = \frac{\mathbf{s}}{(1 - \beta_2^n)}$

$\mathbf{h}_{\text{norm}} = l \frac{\mathbf{m}_{\text{norm}}}{\sqrt{\mathbf{s}_{\text{norm}} + \rho}}$

$\mathbf{w} = \mathbf{w} + \mathbf{h}_{\text{norm}}$

end for

Chapter 3

Knowledge Transfer and exploration methods

3.1 Knowledge transfer

Let us have the agent interacting with the environment solve an RL task. Let there be a source task for which the agent found a model and got nearly optimal policy. Now, the agent is faced with a new target task. The target task can have different domain (environment) or different goals. Knowledge transfer suggests that the relevant knowledge from the source task will be used for solving the target task. This is useful in case we managed to teach the agent the source task and expended computational power on it, however afterwards we want to modify the given source task in some way and gain new task (target task). This approach will let us get solution to this target task faster than if we were to start training the target task from the beginning without using any knowledge.

The computational complexity of Algorithm 1 is quadratic in number of states and linear in number of actions. The convergence slows as $\gamma \rightarrow 1$. We are interested in reusing past experience and improving the agent's performance in the target task. Let us have two sets of tasks C_1 and C_2 such that $C_1 \subset C_2$. Now we consider any task T . We say that there is a transfer if, after training the agent for task T on C_1 , it performs equally well or better than if only trained on C_2 [2].

The main tasks of TL algorithms are to properly define what to transfer, when to transfer and how to transfer. We want to transfer knowledge in form of Q-function from the source task to the target task. Naturally, knowledge transfer makes sense only when it is beneficial for the target task, so we need some condition that stops the most of the knowledge that is not beneficial for the target task from transferring. Lastly, we need to know how to transfer the knowledge from the source task to the target task. For that we need some type of update rule or equation, that inserts knowledge into our algorithm.

In following text we will describe TL via Q-function (TTQL) [1] as a temporal difference update of the target task. It is expected, that a similarity between the source task and the target task implies the similarity of optimal Q-functions [1].

3.2 Target transfer Q-learning (TTQL)

Let us assume that we have solved source task defined by MDP $M_1 = (\mathcal{S}, \mathcal{A}, r_1, p_1, \gamma_1)$ with known estimate of optimal $Q_{source}^\pi(s, a)$. Now let us consider similar target task defined by MDP $M_2 = (\mathcal{S}, \mathcal{A}, r_2, p_2, \gamma_2)$. TTQL algorithm allows us to use knowledge of $Q_{source}^\pi(s, a)$ to speed up the Q-learning algorithm for the target task (Knowledge transfer). To perform successful TL one has to make sure that the distance between M_1 and M_2 is smaller than error given by Q-function of the target task [1]. This knowledge transfer method uses Bellman error (MNBE) given by equation (3.1)

$$MNBE(Q(s_t, a_t)) = \max_{\substack{a_t \in \mathcal{A} \\ s_t \in \mathcal{S}}} |Q(s_t, a_t) - (r(s_{t+1}, a_t, s_t) + \gamma E^{s_{t+1}} \max_{a_{t+1} \in \mathcal{A}} Q(a_{t+1}, s_{t+1}))|, \quad (3.1)$$

as condition, If MNBE in the current state is lower for Q-function of the source task, than MNBE of Q-function that agent constructed so far for the target task, the agent updates Q-function with value from the source task. He also takes an action based on Q-function of the source task. TTQL algorithm can use a learning rate α , that descends with each timestep to lower value. The pseudocode is given below [1].

Algorithm 4 - TTQL

input - initiate Q-function of the target task Q^{old} , reward function r , learning rate α , gamma factor γ , terminal state s_{term} . Q function of source task Q_{source} and maximum number of iterations max_iter
set initial state $s_t = s_{start}$
set time $t = 0$
while $t < max_iter$
 if $s_t = s_{term}$
 set s_t to s_{start}
 end if
 $\alpha_t = \frac{1}{t}$ (optional)
 if(MNBE($Q_{source}(s_t, a_t)$) \leq MNBE($Q_t(s_t, a_t)$))
 $Q_{tt} = Q_{source}$
 else
 $Q_{tt} = Q^{old}$
 end if
 choose action a_t and get following state s_{t+1} .
 $Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha_i(r(s_{t+1}, a_t, s_t) + \gamma \max_{a_{t+1}} Q_{tt}(s_{t+1}, a_{t+1}) - Q^{old}(s_t, a_t))$
 $Q^{old}(s_t, a_t) = Q^{new}(s_t, a_t)$
 $t = t + 1$
end while

3.3 Deep TTQL

In case we are using Deep Q-learning, the algorithm works in similar way but applied to Algorithm 2. This means that we also compare Bellman errors to determine if to use Q-function of the source task in the update or Q-function of the target task. The main difference is, that Q_{source} is estimated by neural network trained on the source task and Q function of the target task is estimated by network we are currently in training. The training is done by Algorithm 3.

Algorithm 5 - Deep TTQL

input - initialize replay memory D , gamma factor γ , terminal states s_{term} , starting state s_{start} , batch size b , train interval n , maximum number of iterations max_iter , neural network estimating Q_{source} of the source task with learned network parameters and neural network estimating Q^{old} of the target task with random network parameters. Set s_0 to s_{start} and $t = 0$.

while $t < max_iter$

if $s_t = s_{term}$

set s_t to s_{start}

end if

 choose action a_t using probability distribution $\theta(a_t|s_t)$ of one of the exploration methods (Sections 3.5 to 3.7) estimate given by neural network Q^{old} and get following state s_{t+1} and reward r_t .

 store transition (s_t, a_t, s_{t+1}, r_t) into D

if t is divisible by n

 sample random batch of size b from D

for $j = 1:b$

if $MNBE(Q_{source}(s_{t,j}, a_{t,j}) \leq MNBE(Q^{old}(s_{t,j}, a_{t,j}))$

$Q_j(s_{t,j}, a_{t,j}) = Q_{source}(s_{t,j}, a_{t,j})$

else

$Q_j(s_{t,j}, a_{t,j}) = Q^{old}(s_{t,j}, a_{t,j})$

end if

$Q^{new}(j) = \begin{cases} Q^{old}(s_{t,j}, a_{t,j}) + (r(s_{t+1,j}, a_{t,j}, s_{t,j}) + \gamma \max_{a_{t+1}} Q_j(s_{t+1,j}, a_{t+1}) - Q^{old}(s_{t,j}, a_{t,j})) & \text{if } s_{t,j} \text{ is not terminal} \\ r_{t,j} & \text{otherwise} \end{cases}$

$Q^{old}(j) = Q^{old}(s_{t,j}, a_{t,j})$

end for

 Train network on Q^{new} and Q^{old} by minimizing the chosen loss function using Algorithm 3 (see Section 2.6).

end if

$t = t + 1$

end while

3.4 Exploration methods

We have established TL in the previous sections of this chapter. However, the key factor of both TL and RL is the method used for exploration. As we will discover in later chapters, the choice of such method has large impact on the performance of TL. These methods focus on describing exploration of state space. It will focus on defining probability distribution $\theta(a_t|s_t)$ that determines which action a_t should agent take from state s_t based on $Q(a_t, s_t)$ and other method-specific parameters. Ideally, we would want to have optimal widely applicable method. However even though few works focusing on exploration methods exists [22] [23] [24], none of them suggest any such solution.

3.5 ϵ -greedy method

This method is based on a simple idea that agent chooses action maximizing the expected reward with probability $1 - \epsilon$ and any other action with probability $\frac{\epsilon}{|A|-1}$, where $|A|$ denotes number of actions available in state

s_t . can be described by equation (3.2) [13]

$$\theta(a_t|s_t) = \begin{cases} 1 - \epsilon & \text{if } a_t = \operatorname{argmax}_{a \in \mathcal{A}} Q(s_t, a_t) \\ \frac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise.} \end{cases} \quad (3.2)$$

In practice we usually decrease the ϵ over time. The ϵ -greedy method was discovered to be quite effective in various machine learning problems, despite its simplicity. The problem of this method is that even with infinite time horizon and every action-state pair visited indefinitely often, it asymptotically prevents the agent from taking the best action. Another possible issue is for this method to converge too quickly on local optimum [18].

3.6 Boltzmann exploration method

Boltzmann method works with probability distribution that assigns each action probability dependant on expected reward. This probability distribution is given by equation (3.3) [13]

$$\theta(a_t|s_t) = \frac{\exp(\frac{Q(s_t, a_t)}{\lambda})}{\sum_{a_t \in \mathcal{A}} \exp(\frac{Q(s_t, a_t)}{\lambda})}. \quad (3.3)$$

Parameter λ is called temperature. If this parameter approaches infinity, then $\theta(a_t|s_t)$ approaches uniform distribution and if it approaches zero, then $\theta(a_t|s_t)$ will approach epsilon greedy method. In other words the method assigns the highest probability to choosing the best action based on the value of temperature λ . Therefore we will start out with large λ and decrease it over time which will cause increase in exploitation over time. The potential problem of this method is over-exploration, which causes algorithm to take longer than we would want to converge [18].

3.7 Upper confidence bound method

This method relies on counting number of times the action has been taken up until time t . Lets denote this number as $n_t(a_t)$ for $a_t \in \mathcal{A}$. We can see the probability distribution in equation (3.4) [13]

$$\theta(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \operatorname{argmax}_{a \in \mathcal{A}} (Q(s_t, a_t) + \sqrt{\frac{2 \ln(t)}{n_t(a_t)}}) \\ 0 & \text{otherwise.} \end{cases} \quad (3.4)$$

In this method the basic premise is that less utilized actions are advantaged by adding a bonus to their Q-value (as in (3.3)) over the more frequently utilized ones and thus utilizing actions that have not been as frequently utilized before.

Chapter 4

Implementation for a virtual drone

The task for implementation was chosen to be a virtual drone navigating through an environment based on images from its camera. The virtual environment is modeled in freely accessible Unreal Engine 4 [21] and connected via Airsim [20] package to Python coding language. The following subsections will describe how the MDP is implemented. The basic code for Deep Q-learning implemented for the drone was taken from [10] and complemented by deep TTQL algorithm and several exploration methods, see Chapter 3.

4.1 States and actions

As suggested in section 2.3, the states in this task are represented by images taken from RGB camera of the drone (see Figure 4.1). These images are then resized to match the pixel size that is later given by the parameter *input_size*. This means that each state s_t is represented by 3 matrices of $input_size \times input_size$ pixels, where each matrix describes red, green or blue color brightness. The considered terminal state in this implementation is any state, where the drone physically hits an obstacle by getting there. Once this happens, the drone returns to the initial state and another episode starts.



Figure 4.1: Image from RGB camera in virtual environment

Real drone would be controlled by actions "up", "down", "forward", "backward", "left" and "right" indicating preferred/optimal direction of the drone move. However, this would be a valid idea only if the state was represented by the spatial coordinates, which is not the case of virtual drone, where the state is given by a camera image (see Figure 4.1). Let us split the image by a grid with fixed amount of cells, then we can define an action as moving in the direction of centre of targeted grid cell of an image. It will correspond to the physical drone moving in the targeted direction with a constant speed. As a movement of the real drone is influenced by noise (for instance observation noise, imprecise control, unpredictable and not measured outer conditions), we shall model that by artificial noise with characteristics reflecting the modelled phenomena.

Thus in our case the number of possible actions in each state s_t is given by parameter $num_actions$ reflecting the number of grid cells. This parameter separates camera image into $\sqrt{num_actions} \times \sqrt{num_actions}$ cells. For instance, when num_action equals 4, as illustrated in Figure 4.2 i.e. The drone can move in the direction of the centre of one of four depicted cells (rectangles).



Figure 4.2: Example of an image segmented by a grid into cells corresponding to actions

4.2 Rewards

The virtual drone, as a real one could be, is equipped with a depth of field (DoF) camera. This camera uses a grayscale image, that shows closer objects darker and further objects brighter. The reward is obtained from this image as it holds information about distance to the obstacles. The example of such image can be seen in Figure 4.3. The image brightness is scaled between 0 and 1. The reward is calculated from certain rectangular cell of the image, which is different at every state. Dimensions of this cell are given by an empirical formula (4.1)

$$\begin{aligned} x_{size} &= \lceil x_m/d \rceil, \\ y_{size} &= \lceil y_m/d \rceil. \end{aligned} \quad (4.1)$$

where d is given by (4.2)

$$d = \begin{cases} \frac{50}{3n} \sum_{i=1}^n o(i, s_t) & \text{if } \frac{50}{3} \sum_{i=1}^n o(i, s_t) > 1 \\ 1 & \text{otherwise.} \end{cases} \quad (4.2)$$

In (4.1), (4.2) and (4.3) $o(i, s_t)$ is the i -th pixel in state s_t , n is a total number of pixels in the image, x_m is size of the image over x-axis in pixels and y_m is size over the y-axis. The centre of the location coordinates $[x_{location}, y_{location}]$ by (4.3)

$$\begin{aligned} x_{location} &= \begin{cases} \frac{0.9x_m(d-1)}{2d} & \text{if } \frac{0.9x_m(d-1)}{2d} > 0, \\ 0 & \text{otherwise} \end{cases} \\ y_{location} &= \begin{cases} \frac{y_m(d-1)}{2d} & \text{if } \frac{y_m(d-1)}{2d} > 0, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (4.3)$$

. These formulas are taken from source code [10], The reward in the terminal state is set as -1 to punish drone for hitting an obstacle.



Figure 4.3: Example of depth of field image for calculating reward

4.3 Deep Q-learning and knowledge transfer

The architecture of the neural network used is in Figure 4.4. The layers function is described in Chapter 2. From an input image we apply a set of five convolution layers with multiple filters and three max-pool layers. The architecture is taken from [17]. Next, scheme divides into two very similar neural networks with four ReLu and one linear layers each (see Chapter 2 for details). One network has a single dimensional output val and one has a $num_actions$ dimensional output adv . The network output is simplified in the scheme. It estimates the Q-function of the virtual drone and it has a shape of a vector of size $num_actions$. Each element of the vector corresponds to estimates $Q(s_t, a_t)$ for each action a_t , given image state s_t , see more details in equation (4.3) [17]. As loss we can use either Huber loss or mean squared error (MSE) loss

$$Q(s_t, a_t) = val + adv(a_t) - \frac{1}{num_actions} \sum_{a_t=1}^{num_actions} adv(a_t), \quad (4.4)$$

where val and adv are outputs called "Value" and "Advantage" of the two subbranches of the neural network in scheme in Figure 4.4.

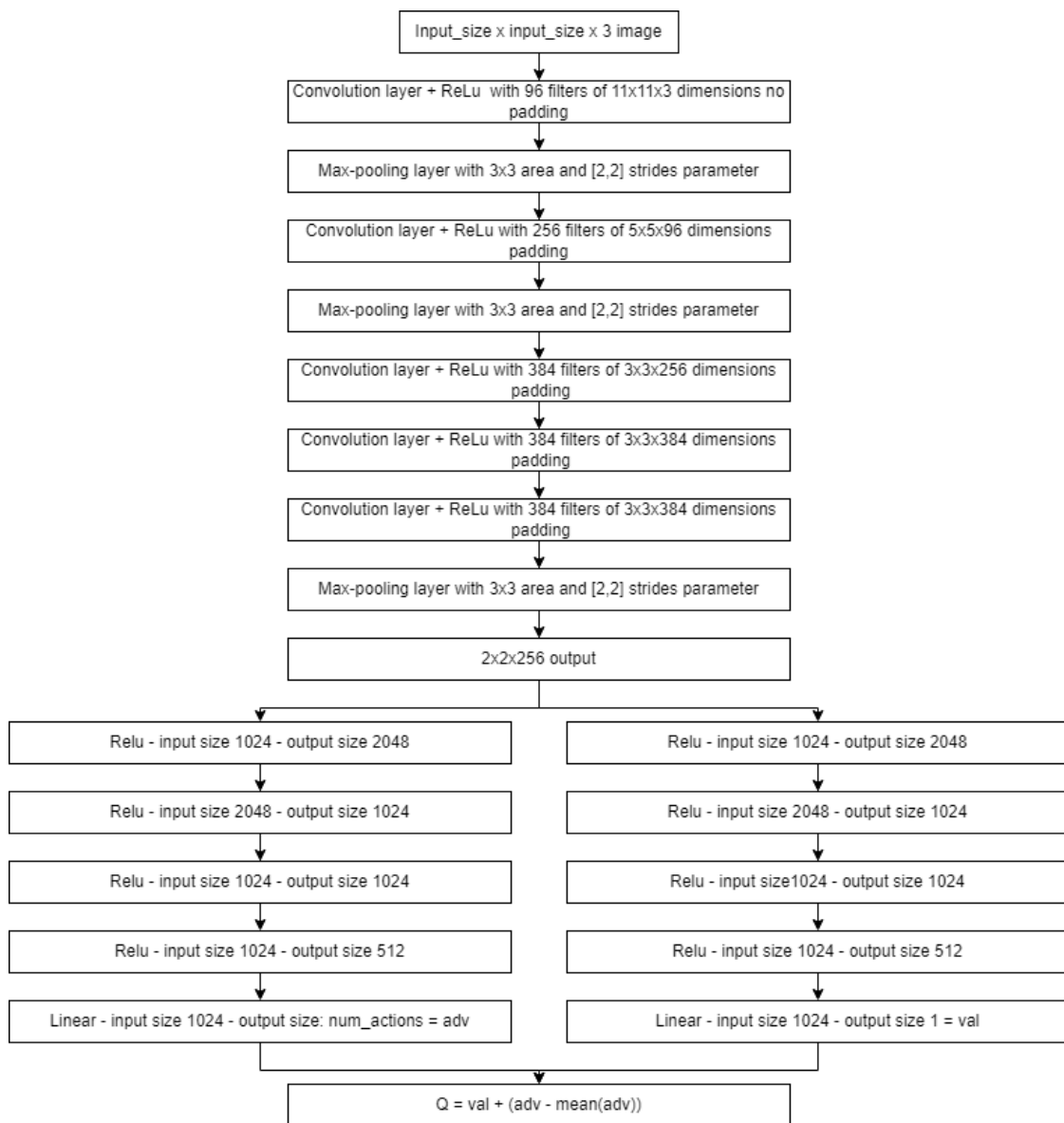


Figure 4.4: Network architecture

As knowledge transfer method we use deep TTQL (Algorithm 5). The architecture of networks for the source task and the target task is the same as well as most of the other parameters from the Section 4.6. The difference is that the source task is going to be trained on completely different map than the target task.

4.4 Parameters used

When setting up the experiments we can adjust parameters that are listed in Table. 4.1.

Parameter	Explanation
<i>input_size</i>	Size of the edge of the rectangle cell of camera image that is taken as state.
<i>num_actions</i>	Number of cells input image is divided into i.e. number of possible actions in each state
<i>wait_before_train</i>	Number of iterations where algorithm just collects data before start of neural network training
<i>max_iter</i>	Maximum number of iterations
<i>buffer_len</i>	Size of the replay memory
<i>batch_size</i>	Size of the batch taken from the replay memory
<i>epsilon_saturation</i>	Parameter used in exponential and linear decline of ϵ . Only used in ϵ -greedy exploration
<i>Q_clip</i>	True/False - if true temporal difference from equation (1.4) is clipped to interval [-1,1] i.e. values greater than 1 are rounded to 1 and values lower than -1 are rounded to -1
<i>train_interval</i>	Number of iterations after which training occurs
<i>gamma</i>	Discount factor
<i>dropout_rate</i>	Dropout rate as mentioned in Chapter 2 applied to neural networks
<i>learning_rate</i>	Neural network learning rate l (as mentioned in Algorithm 3)
<i>starting_temperature</i>	Start temperature λ in Boltzmann exploration
<i>end_temperature</i>	End temperature λ in Boltzmann exploration
<i>end_epsilon</i>	The value where ϵ stops descending in ϵ -greedy algorithms

Table 4.1: Table of parameters

Note, that the reason for long names of parameters is so that they have the same name as in the code used for implementation for a virtual drone.

4.5 ϵ and λ decrease

We have suggested a decrease of ϵ and temperature λ parameters for ϵ -greedy and Boltzmann algorithms from Chapter 3. Here we will show the used equations calculate either of those parameter for each iteration of t in each algorithm. For ϵ -greedy algorithm the equations were taken from [10].

ϵ -greedy algorithm with exponential decrease

For this method we will use (4.5)

$$\epsilon = \begin{cases} \exp\left(\frac{-2}{\epsilon_{saturation} - wait_before_train}(t - wait_before_train)\right) & \text{if } t > wait_before_train \\ 1 & \text{otherwise.} \end{cases} \quad (4.5)$$

If ϵ should be lower than *end_epsilon* in any iteration of t it is automatically set to *end_epsilon* parameter value.

ϵ -greedy algorithm with linear decrease

For this version of ϵ -greedy algorithm we will use (4.6)

$$\epsilon = \begin{cases} (1 - end_epsilon) \cdot \frac{(t - wait_before_train)}{(\epsilon_{saturation} - wait_before_train)} & \text{if } t > wait_before_train \\ 1 & \text{otherwise.} \end{cases} \quad (4.6)$$

Again, if ϵ should be lower than *end_epsilon* in any iteration it is automatically set to *end_epsilon* parameter value.

Boltzmann algorithm

For Boltzmann algorithm we will use (4.7)

$$\lambda = \begin{cases} \text{starting_temperature} \cdot (1 - \frac{t}{\text{max_iter}}) & \text{if } t > \text{wait_before_train} \\ \text{starting_temperature} & \text{otherwise.} \end{cases} \quad (4.7)$$

If temperature λ should ever be lower in any iteration than end_temperature it is set to end_temperature parameter value.

4.6 Overall algorithm

This section shows scheme of overall algorithm for better illustration of the final algorithm architecture (Figure 4.5). This scheme connects Deep TTQL in algorithm 5 and network training from algorithm 3.

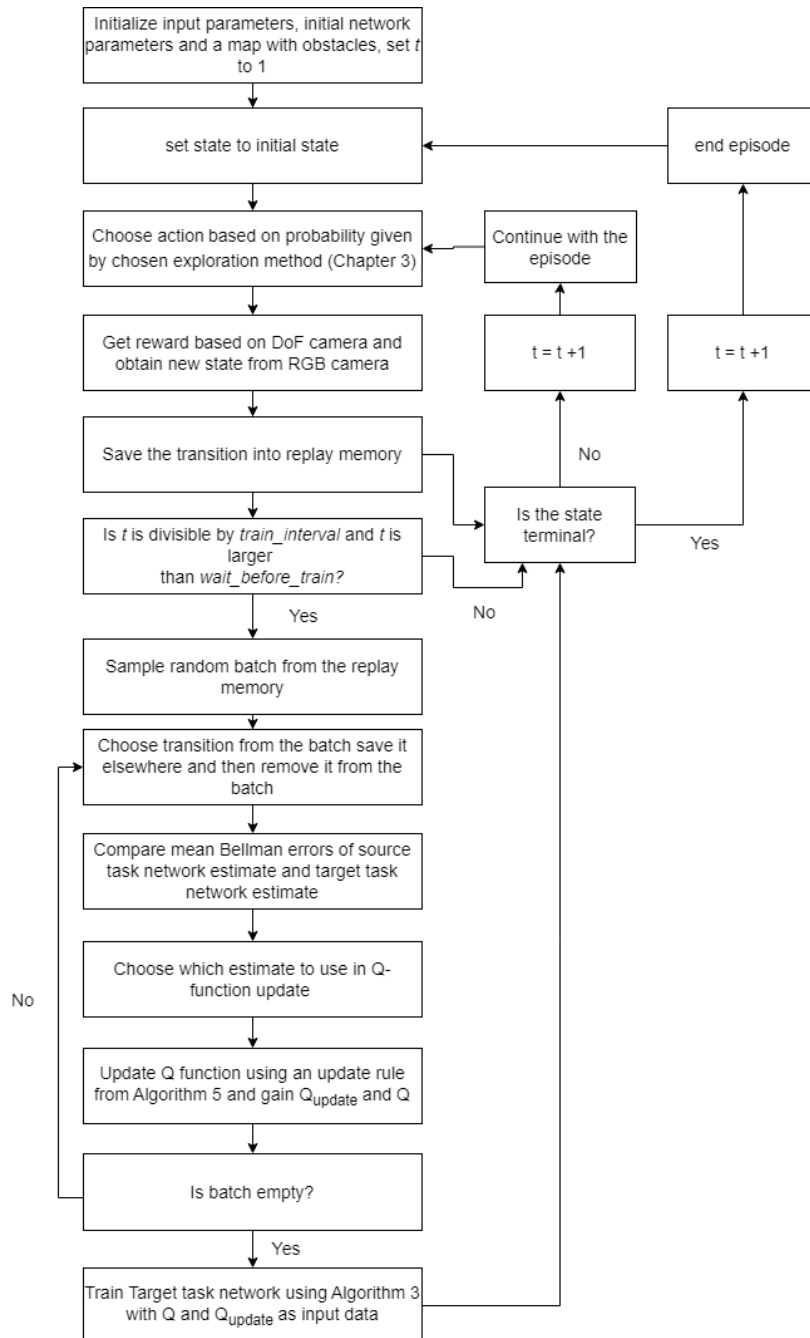


Figure 4.5: Final TTQL architecture

The code is available at: https://drive.google.com/file/d/1XAUx7PfTnY1hwAiSAbek28tlk2V6OfjK/view?usp=share_link

Chapter 5

Experiments

This section describes results of TL experiments. The goal of the experiments is to compare exploration methods in transfer learning. The source task is for the drone to move through an environment with a goal to fly for as long as possible before hitting an obstacle. The target task for the drone to fly through a different environment with the same goal i.e. to fly for as long as possible. By performing experiment on the various target tasks we tested the robustness of each exploration method on this particular family of tasks.

For the source task we use an environment called `indoor_pyramid` [11], part of which is shown in Figure 5.1, in every experiment. The knowledge about this environment will be learned using deep Q-learning with ϵ -greedy algorithm with exponential decline of ϵ . We will test all the exploration methods (see Chapter 3) in conjunction with Deep Q-learning and Target transfer deep Q-learning (see Chapter 1 and Chapter 3). Boltzmann and ϵ -greedy algorithms use parameter descents from Section 4.7. The environments generally differ by layout, thus RGB camera may have different images. Therefore, the environments are sufficiently different to verify transfer of knowledge.

The output of the experiments will be a graph showing reward accumulated over an episode. The reward is given by closeness to the obstacles or by hitting them. As mentioned in Section 4.1 episode is considered the period between drone starting in initial state and hitting an obstacle which is its terminal state. The total number of iterations (*max_iter*) will be the same for each exploration method and each single iteration corresponds to a single time step. However, since each episode may have different amount of steps, depending on time till the terminal state, drone takes different number of iterations per episode so graphs for each method will end on different number of episodes. This means that when looking at the graphs we need to realise that each single line symbolises the same total amount of iterations divided into different amount of episodes i.e. the graphs are comparable. Table 5.1 contains the exploration methods used in the following experiments.

Notation	Explanation
no transfer exponential epsilon greedy	Deep Q-learning (with no transfer) with ϵ -greedy exploration algorithm and exponential epsilon decrease (See Section 4.7)
transfer linear epsilon greedy	Target Transfer Q-learning with ϵ -greedy exploration algorithm and linear epsilon decrease (See Section 4.7)
transfer exponential epsilon greedy	Target Transfer Q-learning with ϵ -greedy exploration algorithm and exponential epsilon decrease (See Section 4.7)
transfer Boltzmann	Target Transfer Q-learning with Boltzmann exploration algorithm (See Section 4.7)
transfer UCB	Target Transfer Q-learning with UCB Algorithm

Table 5.1: Exploration methods used in experiments

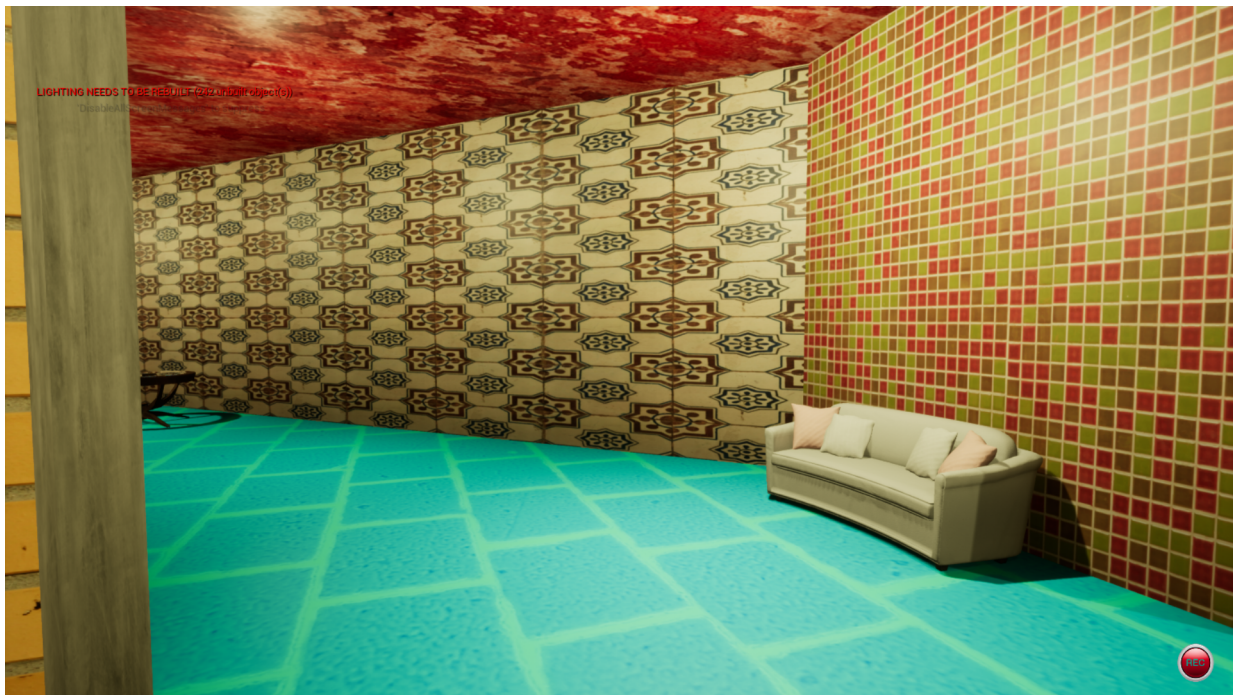


Figure 5.1: Environment Indoor_pyramid

5.1 Experiment 1

In this experiment we transfer knowledge from the source task in Indoor_pyramid environment seen in Figure 5.1. to the target task in Indoor techno [11] environment seen in Figure 5.2. The input parameters used for all exploration methods in the target task and for the source task are in Table 5.2. We used mean squared error loss (2.7) for training neural network.



Figure 5.2: Environment Indoor techno

Parameter	Value
<i>input_size</i>	103
<i>num_actions</i>	25
<i>wait_before_train</i>	5000
<i>max_iter</i>	150 000
<i>buffer_len</i>	10 000
<i>batch_size</i>	32
<i>epsilon_saturation</i>	100 000
<i>Q_clip</i>	True
<i>train_interval</i>	16
<i>gamma</i>	0.99
<i>dropout_rate</i>	0.1
<i>learning_rate</i>	0.000002
<i>starting_temperature</i>	1
<i>end_temperature</i>	0.05
<i>end_epsilon</i>	0.05

Table 5.2: Values of parameters in Experiment 1

The results of the experiment can be seen in Figure 5.3. The baseline **no transfer exponential epsilon greedy** algorithm, has the worst result as expected. It managed to reach cumulative reward over an episode of 35 after 150 000 iterations and 3 753 episodes.

The best performing algorithm in this experiment is **transfer UCB** algorithm. After 150 000 iterations divided into around 2000 episodes it managed to reach cumulative reward of 80. It is followed by **transfer Boltzmann** algorithm that reached cumulative reward over an episode of roughly 75.

The ϵ -greedy algorithms with transfer performed considerably worse. However we can see, that despite **transfer linear epsilon greedy** algorithm needing more episodes than **transfer exponential epsilon greedy** algorithm, it managed to reach higher cumulative reward over an episode of 55 compared to an exponential descent result of 38.

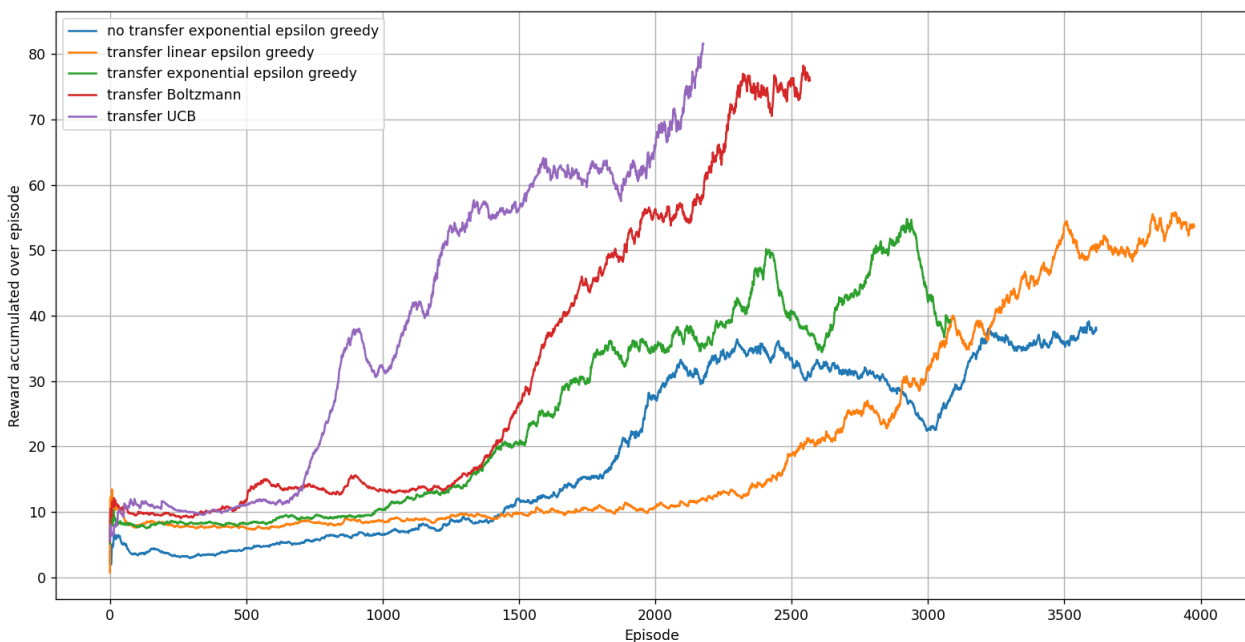


Figure 5.3: Results of Experiment 1 - Reward accumulated over episodes with different number of iterations

5.2 Experiment 2

In this experiment we transfer knowledge from the source task in Indoor_pyramid environment seen in Figure 5.1. to the target task in Indoor_twist [11] environment seen in Figure 5.4. The input parameters used for all exploration methods in the target task and for the source task are in Table 5.3. This time we have used different learning rate l used for training the Neural network (Algorithm 3), than in Experiment 1. We used mean squared error loss (2.7) for training neural network.



Figure 5.4: Environment Indoor_twist

Parameter	Value
<i>input_size</i>	103
<i>num_actions</i>	25
<i>wait_before_train</i>	5000
<i>max_iter</i>	150 000
<i>buffer_len</i>	10 000
<i>batch_size</i>	32
<i>epsilon_saturation</i>	100 000
<i>Q_clip</i>	True
<i>train_interval</i>	16
<i>gamma</i>	0.99
<i>dropout_rate</i>	0.1
<i>learning_rate</i>	0.00001
<i>starting_temperature</i>	1
<i>end_temperature</i>	0.05
<i>end_epsilon</i>	0.05

Table 5.3: Value of parameters for Experiment 2

The results of the experiment can be seen in Figure 5.5. The baseline **no transfer exponential epsilon greedy** algorithm, has the worst result as expected. It managed to reach a cumulative reward of 6 after 150 000 iterations and 11 654 episodes.

The results of other algorithms are very close to each other. The best performing algorithm in this experiment

is, as in Experiment 1, the **transfer UCB** algorithm. After 150 000 iterations divided into around 6700 episodes it managed to reach a cumulative reward of around 13.

The **transfer Boltzmann** algorithm did worse than **transfer exponential epsilon greedy** algorithm for most of the learning process but managed to reach a similar cumulative reward of around 11 in the end. **Transfer linear epsilon greedy** algorithm did best out of all the algorithms for part of the learning. However, the cumulative reward started to descend at around 6000-th episode and in the end this method gave the worst result out of all the exploration methods. This exploration method only reached a cumulative reward of 6.

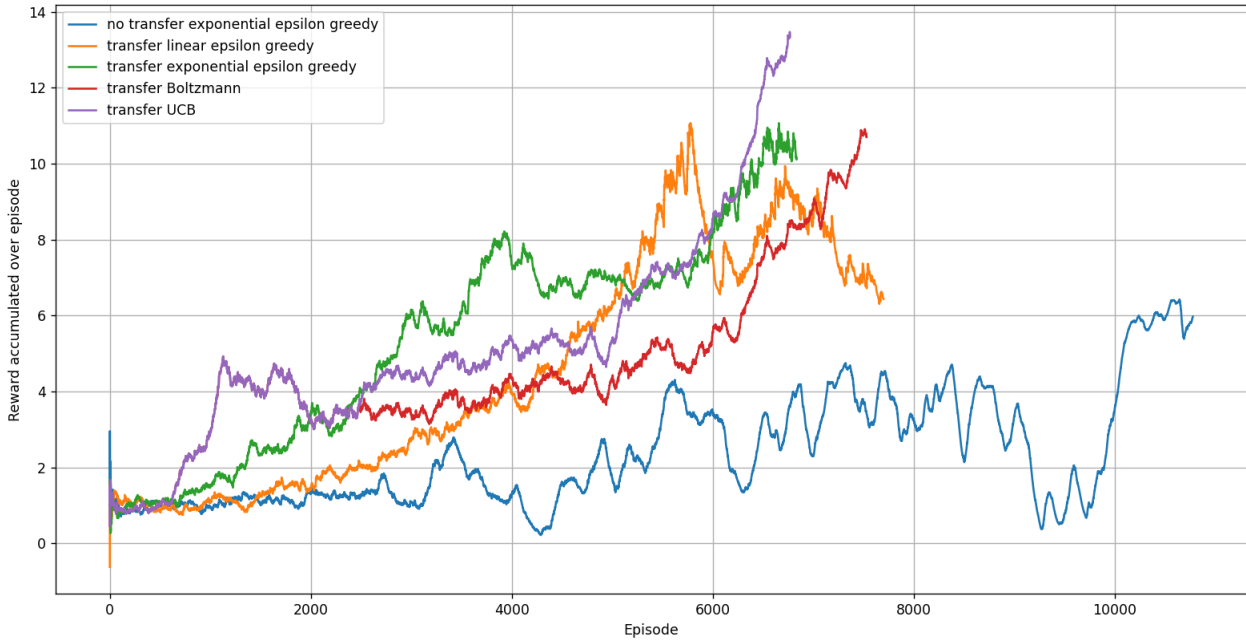


Figure 5.5: Results of Experiment 2 - Reward accumulated over episodes with different number of iterations

5.3 Experiment 3

In this experiment we transfer knowledge from the source task in Indoor_pyramid environment seen in Figure 5.1. to the target task in Indoor_complex [11] environment seen in Figure 5.6. The input parameters used for all exploration methods in the target task and for the source task are in Table 5.4. In comparison with previous experiments, we used Huber loss (2.8) for training neural network.



Figure 5.6: Environment Indoor_complex

Parameter	Value
<i>input_size</i>	103
<i>num_actions</i>	25
<i>wait_before_train</i>	5000
<i>max_iter</i>	150 000
<i>buffer_len</i>	10 000
<i>batch_size</i>	32
<i>epsilon_saturation</i>	100 000
<i>Q_clip</i>	True
<i>train_interval</i>	16
<i>gamma</i>	0.99
<i>dropout_rate</i>	0.1
<i>learning_rate</i>	0.000002
<i>starting_temperature</i>	1
<i>end_temperature</i>	0.05
<i>end_epsilon</i>	0.05

Table 5.4: Values of parameters in Experiment 3

The results of the experiment can be seen in Figure 5.5. The baseline algorithm, **no transfer exponential epsilon greedy** algorithm, has the worst result as expected. It managed to reach a cumulative reward of 5 after 150 000 iterations and 8000 episodes.

The best performing algorithm in this experiment is, as in Experiment 1 and 2, the **transfer UCB** algorithm. After 150 000 iterations divided into around 3000 episodes it managed to reach a cumulative reward of around 25. The second best is **transfer exponential epsilon greedy** algorithm. This algorithm managed to reach a cumulative reward of 22 after 4800 episodes.

Transfer Boltzmann and **Transfer linear epsilon greedy** algorithms both reached similar results. The first one reached a cumulative reward of 16 over 6000 episodes. The second one reached a cumulative reward and of 13 over 6300 episodes.



Figure 5.7: Results of Experiment 3 - Reward accumulated over episodes with different number of iterations

Conclusion

This thesis was focused on exploration methods in knowledge transfer in sequential decision making. In particular, it focuses on exploration methods for knowledge transfer using Q-learning. Target transfer Q-learning (TTQL) was used as knowledge transfer algorithm. Both Q-learning and TTQL were generalised into deep Q-learning and deep TTQL respectively. Furthermore, we have introduced various types of ϵ -greedy, Boltzmann and UCB exploration methods, that were used in deep TTQL.

These methods were implemented on a task with drone navigating through a room using its RGB and DoF cameras to avoid obstacles and walls. The benefits of exploration methods in knowledge transfer were evaluated based on comparison of results obtained using TTQL with different exploration methods and deep Q-learning using ϵ -greedy method with exponential decrease of parameter ϵ . To assess the knowledge accumulated we used an accumulated reward over an episode, that indicates how efficient a drone is at avoiding obstacles. We have conducted three experiments, which illustrate usefulness of exploration methods on different maps given the task mentioned. These experiments used knowledge of the source task used in an indoor_pyramid environment on the target tasks of avoiding an obstacle in the following environments. Each with different layout and sets of images camera can obtain. Furthermore, some parameters were changed in each experiments so the change in the source task and the target task was significant.

- Experiment 1 - indoor_techno, baseline experiment (Figure 5.2)
- Experiment 2 - indoor_twist (Figure 5.4), higher learning rate
- Experiment 3 - indoor_complex (Figure 5.6), different loss function (Huber loss)

The results for each experiment have confirmed the impact of knowledge transfer, with slightly weaker results in Experiment 1. As for exploration methods the experiment have consistently shown that UCB method performs the best, with other methods giving worse results that were not as consistent between the experiments. The issue of ϵ -greedy methods possibly converging too fast to local optimum has not been a problem in our experiments with possible exception of Experiment 1, where ϵ -greedy method with exponential decrease seemed to converge. As for Boltzmann method the over exploration has not been an issue in any of the experiments. Future work could focus on improving the parameters decrease equations for Boltzmann and ϵ -greedy exploration methods and involving more exploration methods into the analysis. Furthermore, the drone could, apart from avoiding obstacles, have an additional task of reaching some area of the map or taking trajectory of certain shape.

Bibliography

- [1] Y. Wang, et.al, (2020), Target transfer Q-learning and its convergence analysis, *Neurocomputing*, 392,11-22
- [2] A.Barreto, et.al, (2017), Successor Features for Transfer in Reinforcement Learning in *Advances in Neural Information Processing Systems 30*, NIPS 2017
- [3] A. Lazaric, (2012). *Transfer in Reinforcement Learning: A Framework and a Survey*.
- [4] L. P. Kaelbling, M. L. Littman, A. W. Moore, (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 237–285.
- [5] M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley 2005
- [6] R. Bellman, *Dynamic Programming* Princeton Univ.Press, 1957.
- [7] R.Sutton, A.Barto, (2018) *Reinforcement Learning: An Introduction*, March 1, Second Edition MIT Press, Cambridge, MA
- [8] G.James, D. Witten, T. Hastie, R. Tibshirani, (2021). *An Introduction to Statistical Learning with Applications in R*, Second Edition. Springer New York.
- [9] M. Wang, (2020). Deep Q-Learning Tutorial: minDQN. Towards Data Science. <https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc>
- [10] A. Anwar, (2020). Pedra - Programmable Engine for Drone Reinforcement Learning Applications. <https://towardsdatascience.com/pedra-programmable-engine-for-drone-reinforcement-learning-applications-5b04423a42dd>
- [11] A. Anwar, (2019). Deep Reinforcement Learning for Drones in 3D realistic environments. <https://drive.google.com/drive/folders/1w-Il1vHhAcSf9V75y5-dXPUF76-AgbpY>
- [12] R. Jafari, M. Javidi, (2020). Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning. *SN Applied Sciences*. 2. 10.1007/s42452-020-2012-0.
- [13] F. Hůla, (2018). Bc. [Master's Thesis].
- [14] S. Ruder, (2020, March 20). An overview of gradient descent optimization algorithms, <https://ruder.io/optimizing-gradient-descent/index.html#adam>
- [15] M. Otterlo, M. Wiering, (2012). Reinforcement Learning and Markov Decision Processes. *Reinforcement Learning: State of the Art*. 3-42. 10.1007/978-3-642-27645-3_1.
- [16] J.Torres, (2020, June 11). The Bellman Equation. <https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7>
- [17] A. Anwar, A. Raychowdhury, (2019). Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes using Transfer Learning, arXiv:1910.05547.
- [18] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, D. Precup, (2021). A Survey of Exploration Methods in Reinforcement Learning, arXiv:2109.00157.
- [19] R. Jagtap, (2020). Dynamic Programming in RL. <https://towardsdatascience.com/dynamic-programming-in-rl-52b44b3d4965>

- [20] Microsoft Research created simulation platform for AI research and experimentation. (2017).
- [21] 3D computer graphics game engine developed by Epic Games. (2012).
- [22] M. Gimelfarb, A. Barreto, S. Sanner, C.-G. Lee (2021). Risk-aware transfer in reinforcement learning using successor features, arXiv:2105.14127.
- [23] D. Pathak, D. Gandhi and A. Gupta (2019). Self-Supervised Exploration via Disagreement. In Proceedings of Machine Learning Research 97:5062-5071
- [24] I. Osband, B. Van Roy and Z. Wen (2016). Generalization and Exploration via Randomized Value Functions, ICML'16: Proceedings of the 33rd Int. Conference on International Conference on Machine Learning.
- [25] N.Kapoor (2020), Loss functions - when to use which one. <https://www.numpyninja.com/post/loss-functions-when-to-use-which-one>