Universiteit Leiden

Master Computer Science

Deep Q-learning for Matching Comprehensively Typed Red Blood Cell Units

Name:	Evani Lachmansingh
Student ID:	s1949624
Date:	23/1/2023
Specialisation:	Artificial Intelligence
1st supervisor:	Mart P. Janssen.
2nd supervisor:	Mike Preuss.
External superv.:	Merel Wemelsfelder.

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract. Comprehensive antigen typing of red blood cell (RBC) units increases the number of blood groups to consider for matching, which introduces various logistical challenges in the allocation policy. Since identical matching is not feasible with a large amount number of included antigens (up to 2^{14} when considering 14 antigens), the issuing policy needs to consider mismatch tolerance for all 11 minor antigens, while enforcing compatible matching on the 3 major antigens (A, B and D), minimizing shortages and outdating of RBC units. Currently, a linear programming method is available that applies a near-optimal policy for matching comprehensively typed RBC units. However, it is limited by the requirement of determining a matching policy beforehand by defining the model's objective and constraints. This study researched using Reinforcement Learning (RL), specifically Deep Q-learning (DQN), for finding optimal matching policies. Results showed that DQN can match up to 90% requests with an ABD-compatible product, with only an average of 0.15 - 0.18 antigen mismatches per request when including 2 antigens additionally to A, B, and D. This indicates that, with further research, DQN might be a suitable option for creating an allocation policy for comprehensively typed RBC units.

Keywords: RBC matching · Linear programming · Deep Q-learning

Table of Contents

1 Introduction

Medical conditions or acute trauma can cause individuals to require a red blood cell (RBC) transfusion to restore the blood's hemoglobin levels and oxygen supply. RBC transfusions are relatively safe and common [13]. In the Netherlands, around 400.000 RBC units are distributed annually [2]. Before performing an RBC transfusion, the recipient's and the donor's blood are matched based on the presence of antigens. This prevents the recipient from developing antibodies against antigens present on the donor's red blood cells, a process known as alloimmunization. In most first-time transfusions, matching based on the three 'major' antigens A, B, and Rhesus D is sufficient to prevent transfusion reactions from occurring. However, once patients require multiple transfusions, they are at an increased risk of alloimmunization against other minor antigens (up to 60% of patients develop antibodies after multiple transfusions) [3,4]. Should a patient develop antibodies against some antigen, for any subsequent transfusions, this patient can now only receive blood that is compatible with that antigen, to prevent transfusion reactions. Ideally, however, preventive comprehensive blood matching on clinically relevant 'minor' antigens is performed for all RBC transfusions to minimize alloimmunization rates and simplify subsequent RBC transfusions.

While comprehensive matching may have a positive impact on health outcomes and the efficiency of the healthcare system, it also introduces logistical challenges. Extending the matching strategy to include more relevant antigens means that the number of possible blood groups increases exponentially (2^n, where n is the number of clinically relevant antigens). Since it is not feasible to always identically match blood units when considering many antigens, the number of potentially transfusable blood groups increases, making the process of deciding which blood group to administer more difficult. When matching comprehensively, a policy is required to determine when to allow minor antigens mismatches, to minimize alloimunization risks, while also preventing shortages in the blood bank inventory. Additionally, the process is further complicated by the 35-day shelf-life of RBC units. Therefore, it is necessary to develop a well-suited issuing policy that assists in distributing comprehensively matched RBC units.

In 2022, van Weem et al. proposed a linear programming model (MINRAR) to develop an issuing policy for comprehensive RBC matching [29]. In linear programming, an objective function is minimized or maximized, bounded by a set of predefined constraints. In the context of our matching problem, the constraints include the weighted sum of minor antigen mismatches, the number of shortages, and the number of outdated RBC products. Compared to standard ABO-RhD matching, MINRAR could reduce alloimmunization rates by 78.2%. Hereby they show that, with a proper issuing policy, comprehensive matching has the potential to drastically reduce the risk of transfusion reactions. While MINRAR showed promising results, it is limited by the core concept of lin-

ear programming: objectives and constraints are set beforehand and a policy is created accordingly. Its performance may therefore be improved by discovering other policies that may lead to better outcomes.

Reinforcement Learning (RL), a specific branch of machine learning, might serve as a method for facilitating the blood-matching process. In RL, a so-called agent is tasked with developing an optimal policy for a sequential problem. The agent learns this policy by making decisions in an environment and learning from these decisions. By rewarding the agent for favorable actions, and penalizing it for undesirable actions, the policy can be steered in an optimal direction.

This project will aim to answer the following research question:

1. Can Reinforcement Learning, specifically Deep Q-learning, be used to develop an issuing policy for the distribution of comprehensively typed RBC units?

The RBC issuing process is translated to a Markov Decision Process (MDP) and Deep Q-learning is implemented to create an issuing policy. The goal is to create an issuing policy that minimizes alloimmunization risks, blood bank inventory shortages, and the outdating of RBC units. Antigen sets with 3, 5, and 7 antigens are included in this study. MINRAR will serve as the baseline model to which results will be compared.

2 Background

2.1 Blood Compatibility

Ensuring compatibility of the recipient's and the donor's blood types is essential before proceeding with a blood transfusion. Should the recipient's blood host any antibodies against antigens present on the donor's red blood cells, a transfusion reaction could occur [18]. During such a reaction, the recipient's antibodies will attack the donor's red blood cells. The severity of the immune response varies and depends on various factors, such as the type of antigen that started the reaction and the alloantibodies produced following the exposure. Still, the medical complications could be life-threatening [13][25].

An individual's main blood type is determined by the ABO blood group. Using the ABO antigens, blood types can be categorized into 4 types: A, B, AB, and O. An individual with blood type A has A antigens and anti-B antibodies, indicating they cannot be matched with blood that holds B antigens (blood type B or AB). Blood type O indicates the absence of A and B antigens. The ABO antibodies are naturally present and are highly immunogenic, indicating they have a high probability of inducing an immune response, which is why they are used as the primary determinants for blood types [7].

Additionally, the primary blood types can be further defined by the Rh(D) antigen or 'Rhesus factor'. The Rh(D) antigens are also highly immunogenic, although their antibodies generally do not occur naturally. This implies that the development of antibodies against Rh(D) antigens typically only occurs after exposure to blood with incompatible Rhesus factors [8]. The presence or absence of the Rhesus factor is denoted by + and -, respectively.

2.1.1 Matching Matching of the major blood types can be simply represented as matching binary vectors, also illustrated by van Sambeeck et al. [23]. Table 1 shows the vectors for the blood types including the ABO and Rh(D) antigens.

	$\mathbf{O}^-$	$\mathbf{O}^+$	$\mathbf{A}^-$	$\mathbf{A}^+$	$\mathbf{B}^-$	$\mathbf{B}^+$	$\mathbf{AB}^-$	$\mathbf{AB}^+$
vector	[0 0 0]	[0 0 1]	[1 0 0]	[1 0 1]	[0 1 0]	[0 1 1]	[1 1 0]	[1 1 1]

Table 1. Binary vectors for blood types when the ABO and Rhesus factor antigens are included. Bits correspond, from left to right, to ABD.

Blood types match when every antigen present on the donor's RBCs is also present on the recipient's RBCs. Let vector i denote the recipient's blood type and j denote the donor's blood type. i_k represents the k-th element of vector i and j_k represents the k-th element of vector j. The vectors i and j match under the following condition:

$$\forall k \in \{1, 2, ...|i|\} : (j_k = 1 \rightarrow i_k = 1) \tag{1}$$

Using this condition, compatibility can also be shown in a compatibility matrix of these vectors. The compatibility matrix of the vectors representing the major blood types is shown in Table 2.

		Donor i							
		[000]	[001]	[100]	[101]	[010]	[011]	[110]	[111]
Recipient j	[000]	1	0	0	0	1	0	0	0
	[001]	1	1	0	0	0	0	0	0
	[100]	1	0	1	0	0	0	0	0
	[101]	1	1	1	1	0	0	0	0
	[010]	1	0	0	0	1	0	0	0
	[011]	1	1	0	0	1	1	0	0
	[110]	1	0	1	0	1	0	1	0
	[111]	1	1	1	1	1	1	1	1

Table 2. Compatibility matrix of the major blood types. The vectors in the first column represent the recipient's blood type, the vectors in the first row represent the donor's blood type.

2.1.2 Comprehensive matching In addition to the three major antigens, The International Society of Blood Transfusion (ISBT) recognizes more than 300 different minor antigens [16]. The vast majority of their antibodies are not naturally occurring, meaning individuals must be exposed to them before they develop corresponding antibodies [5]. Consequently, patients who undergo multiple blood transfusions face an increased risk of developing antibodies against a variety of antigens. For these patients, it is important to conduct comprehensive matching of clinically relevant antigens, to prevent transfusion reactions from occurring. Table 3, adapted from van Sambeeck et al., summarizes the Dutch matching strategy for various patient groups [23].

Patient group	**Matching strategy**
Sickle cell anemia and thalassemia	Rh phenotype, K and Fy^a. (and if available, Jk^b., S and s)
Autoimmune hemolytic anemia	Rh phenotype and K
Myelodysplastic syndrome	Rh phenotype and K
Alloimmunized with clinically important antibodies	Rh phenotype and K
Woman of childbearing age	c, E and K

Table 3. Matching strategies for various patient groups according to the 2011 Duch Transfusion guideline. Adapted from van Sambeeck et al. The antigens present in the Rh blood group, are C, c, D, E, and e (referred to as the Rh phenotype).[23]

Ensuring comprehensive matching for all RBC transfusions would reduce overall alloimmunization rates and simplify the matching procedure for any patients that require multiple transfusions [11]. Additionally, the standard ABO-RhD matching reduces the availability of RBC units with rare blood types in the blood bank inventory, since units are distributed regardless of any minor antigens. Comprehensive matching does take all clinically relevant antigens into account, and could, therefore, be used to reduce shortages of RBC units with rare blood types.

Of all the known antigens, an additional 11 are deemed clinically relevant due to their prevalence and immunogenicity. Immunogenicity is formally defined as the probability of an antigen provoking an immune response. Evers et al. calculated the proportion of antigen-negative patients who developed antibodies after a transfusion with mismatched antigens in a cohort study [6]. These probabilities, normalized to sum up to a total of 100%, were utilized to indicate the severity of a mismatch on a certain antigen, shown in Table 6.

	Rel. immunogenicity
C	3.45%
c	7.06%
E	23.97%
e	8.37%
K	38.42%
Fy^a	4.43%
Fy^b	1.31%
Jk^a	8.37%
Jk^b	0.33%
S	1.31%
s	0.00%

Table 4. 11 minor clinically significant antigens and their relative immunogenicities.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a domain of artificial intelligence focused on developing algorithms that learn decision-making policies through trial-and-error [26]. Unlike supervised learning, where models are trained on labeled data, in RL, a so-called 'agent' learns by receiving feedback in the form of rewards based on their actions. RL consists of states, actions, and rewards, illustrated in Figure 1. At timestep t, the agent receives an observation from the environment, or the state S_t. The agent interacts with this state through action A_t which leads to a new state S_{t+1} and a corresponding reward R_{t+1}. The agent uses a policy to decide which actions to take. The received rewards for these actions represent their contribution to achieving some goal and are used to update the policy accordingly. As a result, actions that led to success become more likely to be executed again, while unfavorable actions are unlikely to be repeated.

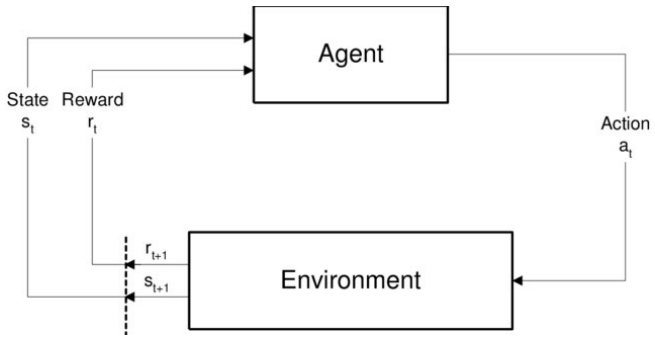

Fig. 1. Illustration of the workflow in Reinforcement Learning. Adapted from [26]

RL algorithms can be implemented in various ways. Generally speaking, RL algorithms can be defined by the following characteristics:

1. Model-based or model-free. In model-based RL, the agent has access to a model that predicts the outcome of actions [22]. In model-free RL, there is no such model and the agent has to explicitly try out actions to find out the outcome [26]. Model-based is very useful, but such a model is not always available.
2. Value-based or policy-based. In value-based RL, the estimations of the values of the actions are learned. The policy is implicit and can be derived from these values. An example is Q-learning [28]. Value-based RL works well in environments with discrete action spaces. In continuous action spaces, policy-based methods are better suited [20]. In policy-based RL, the actual policy itself is learned. This policy then specifies which action the agent needs to take. Trust Region Policy Optimization is an example of a policy-based algorithm [24]. Some algorithms, such as Actor-Critic, combine policy and value estimation [17].

2.3 Q-Learning

Q-learning is a model-free and value-based RL algorithm that aims to learn value estimations of actions, referred to as 'Q-values' [28]. The Q-value is calculated using the reward that the agent received by taking an action a in a state s. By continuously updating these Q-values for each state-action pair, the agent learns the value of actions over time. In tabular Q-learning, all Q-values are stored in a table or Q-matrix. The values are calculated using the Bellman equation [20]:

$$Q'(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma * maxQ'(s',a') - Q(s,a)] \tag{2}$$

where α indicates the learning rate, γ the discount value, $R(s,a)$ the reward for taking action a in state s, and Q' and Q are the Q-values of the next and the current state respectively. The discount factor γ is used to influence the impact of future rewards. A value close to 1 amplifies the influence of rewards in the distant future, while a value close to 0 prioritizes immediate rewards [20].

Over time, Q-values reflect the quality of actions and can be used to guide the agent's next moves. High Q-values represent the actions that have accumulated the most cumulative reward. However, it is not always useful to simply select the actions with the highest Q-values. In the beginning of a learning process, Q-values do not represent the true worth of state-action pairs yet and always choosing the action with the highest Q-value could result in the agent missing out on potentially better, but unexplored, actions. Therefore it is necessary to balance exploitation and exploration when performing action selection during the training of the agent. A common approach to balance exploitation and exploration is ϵ-greedy action selection [20]. In ϵ-greedy, the ϵ parameter represents the probability of the agent selecting the action with the highest Q-value versus choosing a random action. This way, we encourage the agent to also explore other options apart from just choosing the action with the highest value.

	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$	$\mathbf{A_4}$
$\mathbf{S_1}$	0.4	5.2	5.7	4.1
$\mathbf{S_2}$	0.8	0.4	6.3	0.4
$\mathbf{S_3}$	10.5	15.0	7.7	1.4
$\mathbf{S_4}$	0.5	12.0	3.0	2.4
$\mathbf{S_5}$	4.5	0.2	3.0	5.5
$\mathbf{S_6}$	5.5	12.0	0.9	0.5
$\mathbf{S_7}$	13.5	8.1	1.5	20.1
$\mathbf{S_8}$	7.5	18.0	3.0	0.9

Table 5. Example of a Q-matrix used in tabular Q-learning. In this example, there are 4 actions and 8 different states. The cell values represent the Q-values for each state-action combination.

2.4 Deep Q-learning

Tabular Q-learning falls short when the environment of the problem is characterized by a large number of states (or even an infinite number of states). It would then require impractical amounts of memory to store and update all Q-values in a look-up table. In such cases, a more efficient approach is to implement a neural network to approximate the Q-values, a method called Deep Q-learning [21]. This neural network, also referred to as the Q-network, takes the state as input and predicts Q-values for all available actions. Moreover, a technique called experience replay can be utilized, in which the agent's experiences at each time step are stored in memory [19]. An experience, represented as $e_t = (s_t, a_t, r_t, s_{t+1})$, is saved in this replay buffer, and random samples from the buffer are used to update the Q-network. When performing an update, the next state s_{t+1} is passed through the Q-network, which then outputs the predicted Q-values for this state. Updating the Q-value of the chosen action can be performed in one of two ways. In case the environment is ending, and an ending (or terminal) state is reached, the Q-value will be equal to the received reward. If the next state is not terminal, the Bellman equation is used to calculate the target Q-value. The current state s_1 is also passed through the network. The squared difference between the Q-value corresponding to action a_t and the target Q-value is then calculated, and the gradients of this loss w.r.t. the Q-network's weights are used to update these weights. Pseudocode for Deep Q-learning, adapted from Mnih et al, is provided in Algorithm 1 [19].

Algorithm 1: Deep Q-learning with experience replay, adapted from Mnih et al. [19]

Initialize experience replay buffer B
Initialize Q-network with random weights θ
for *timestep* $t = 1...T$ **do**
  With probability ϵ select action a_t,
  otherwise select $a_t = argmax_a Q((s_t), a; \theta)$
  Execute a_t and receive state s_{t+1} and reward r_t
  Store experience (s_t, a_t, r_t, s_{t+1}) in B
  Sample experience $(s_s, a_s, r_s, s_{s+1}) \sim B$
  if s_{j+1} *is terminal* **then**
    $y_s = r_s$;
  else
    $y_s = r_s + \gamma * max_{a_s+1} Q((s_{s+1}), a_{s+1}; \theta)$
  Gradient descent with $[y_s - Q(s_s, a_s, \theta)^2]$ as loss

3 Related work

3.1 Optimal blood issuing by comprehensive matching

In 2022, van Sambeeck et al. developed a mathematical framework for the distribution of comprehensively matched RBC units [23]. Their research investigated the feasibility of an issuing policy for RBC units typed with more than just the ABO-RhD antigens. The authors represented the blood types as binary vectors and formulated the RBC distribution as a minimum cost flow problem (MFCP), an optimization method that aims to find the cheapest flow through a directed graph. Three distinct issuing policies —FIFO (First-in-First-out), MROL (minimize relative opportunity loss), and a combined FIFO/MROL policy— were investigated. In the MROL policy, relative opportunity loss quantifies the relative loss of a match when an RBC unit with type i is issued to a request with type j.

The study investigated scenarios including 8 antigens and 14 antigens, conducting simulations with varying shelf lives of RBC units, inventory size in weeks, and the three issuing policies. In the 8-antigen setting, shortage and outdating rates remained below 1% in nearly all scenarios when a maximum shelf life of 21 days or longer was selected. When including 14 antigens and a maximum shelf life of 28 days or longer, outdating and shortage rates were kept under 5 % in almost all scenarios.

3.2 MINRAR

Weem et al. proposed a matching strategy, MINRAR (MINimize Relative Alloimmunization Risks), that utilizes linear programming for developing an issuing policy for the matching of comprehensively typed RBC units [29]. MINRAR

was compared to using the FIFO/MROL ABO-RhD strategy of van Sambeeck et al [23]. In linear programming, the aim is to minimize or maximize an objective, within the bounds of predefined constraints. In the MINRAR context, the objective function was designed to minimize shortages, outbreaks, and the cost associated with major and minor antigen substitutions. Penalty costs for major antigen substitutions were calculated based on antigen prevalence, while relative immunogenicity was used to calculate penalty costs for minor antigen substitutions. Evaluation of MINRAR involved multiple 1-year simulations using simulated request and inventory data across various inventory sizes. Results indicated a notable 78.2% reduction in alloimmunization risk compared to the FIFO/MROL ABOD policy, without an increase in shortages and outdated units. In the scenario with the largest inventory size, MINRAR surpassed the baseline even further, with a 93.7% decrease in alloimmunization risk.

MINRAR provides the basis of our research. While MINRAR's results are promising, it is limited by the prerequisite of defining constraints before optimization. In Reinforcement Learning, an agent can learn a policy without explicit constraints, meaning there is potential for finding a policy that leads to lower alloimmunization risks.

4 Methodology

4.1 Data simulation

The state is created from a combination of hospital request (demand) data and blood bank inventory (supply) data. This data is simulated according to the actual Dutch hospital demand statistics, blood bank inventory statistics, and antigen prevalence in the population.

4.1.1 Demand data In the demand data, daily RBC requests are described. Features of a request include the blood type (14 antigens), the number of requested units, the patient type, and the lead time of the request, which is the duration between the request being known and the actual transfusion. Overall, the lead time for all requests was limited to a maximum of 7 days. The patient type distribution is shown in Table 6. In the data simulated for this study, only one hospital is included. For this hospital, the average daily demand is assumed to be 50 RBC units.

4.1.2 Supply data Supply data represents the inventory of the blood bank. Like the demand data, it is assumed that each unit is typed for 14 antigens. The major antigen prevalence in the simulated supply data represents the actual antigen distribution in the donor population and is shown in Figure 2. The minor antigens are sampled according to their prevalence in the Caucasian population.

Patient group	Distribution
Allo	3.16%
SCD / Thal	1.72%
MDS	0.00%
AIHA	1.49%
Wu45	4.94%
Other	88.69%

Table 6. Patient group distribution used in the simulated data. The data originates from the patient distribution of a hospital in Amsterdam, OLVG East.

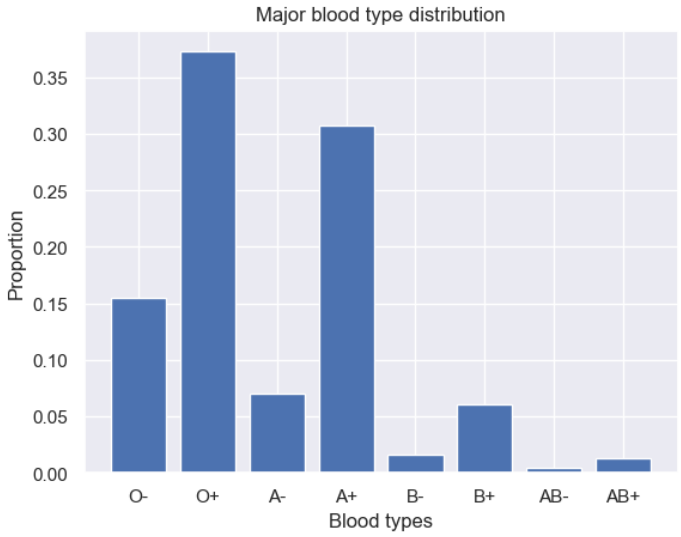

Fig. 2. Blood type distribution of the major blood groups (including A,B, and Rh(D) antigens).

4.2 Environment

4.2.1 State space Each state is a matrix representation of the inventory of the blood bank (supply data), along with information on the pending donor requests from the hospital (demand data). The matrix representing the state consists of 2 concatenated matrices: one for the inventory (I), and one for the requests (R).

Matrix I The row indices of the matrix I correspond to specific blood groups. The binary vector representation of a blood group, as introduced in Table 1, can be transformed into an integer number (the row indices) and these are ordered from 0 to $2^n - 1$, where n represents the number of included antigens. The columns in the matrix denote the age of blood products. In this study, the

maximum age of an RBC unit is set to 35 days, in line with the Dutch blood bank (Sanquin) policy [1]. Each cell in the matrix displays the number of blood units available for a specific blood group and age combination. A visual representation of matrix I is displayed in Figure 3.

$$
\boldsymbol{I} = \begin{array}{c} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \cdots & \mathbf{34} \end{array} \\ \begin{pmatrix} 0 & 2 & 5 & 3 & 0 & 0 & 4 & \cdots & 3 \\ 3 & 0 & 0 & 2 & 0 & 1 & 3 & \cdots & 5 \\ 0 & 5 & 2 & 0 & 4 & 1 & 4 & \cdots & 1 \\ 4 & 4 & 3 & 0 & 3 & 0 & 3 & \cdots & 0 \\ 0 & 5 & 1 & 0 & 0 & 3 & 4 & \cdots & 3 \\ 3 & 0 & 4 & 2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 3 & 4 & 0 & 0 & 4 & \cdots & 3 \\ 0 & 4 & 3 & 0 & 5 & 3 & 0 & \cdots & 0 \end{pmatrix} \end{array} \tag{3}
$$

Fig. 3. An example representation of matrix I in any state. This state is designed with only the A, B, and Rh(D) antigens, and the maximum age of a blood product is set to 35 days. Each row number corresponds to a blood group, while the columns correspond to the age of the blood unit. In this specific example, there are 5 remaining blood units with a blood group of O- and an age of 2 days (see red entry).

Matrix R In this matrix, data on the number of requests for each blood group is stored. The row numbers correspond to the same blood groups as in matrix I. Matrix R has 9 columns. The first 8 columns indicate the lead time: how many days there are left until the request must be matched. Each request is handled independently. The 9^{th} column is filled with 0's, except for one: this is the request that the agent is currently handling in this specific state.

$$
\boldsymbol{R} = \begin{array}{c} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathbf{8} & \mathbf{7} & \mathbf{6} & \mathbf{5} & \mathbf{4} & \mathbf{3} & \mathbf{2} & \mathbf{1} & \boldsymbol{HR} \end{array} \\ \begin{pmatrix} 0 & 2 & 5 & 3 & 0 & 0 & 4 & 3 & 1 \\ 3 & 0 & 0 & 2 & 0 & 1 & 3 & 5 & 0 \\ 0 & 5 & 2 & 0 & 4 & 1 & 4 & 1 & 0 \\ 4 & 0 & 3 & 0 & 3 & 0 & 3 & 0 & 0 \\ 0 & 5 & 1 & 0 & 0 & 3 & 4 & 3 & 0 \\ 3 & 0 & 4 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 4 & 0 & 0 & 4 & 3 & 0 \\ 0 & 4 & 3 & 0 & 5 & 3 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{4}
$$

Fig. 4. An example representation of matrix R in any state. In this state, only the A, B, and Rh(D) antigens are included and the maximum lead time of a blood unit is set to 7 days. The row numbers represent the same blood groups as matrix I. Column HR indicates the request that is currently being handled. In this specific example, the agent is handling the donor requests in row 0 (blood type O-, marked red).

The full state consists of the concatenation of matrices I and R: $[I, R]$. The matrix is flattened when used as input for the Q-network. Note that the dimension of the state matrix is dependent on the number of included antigens the maximum age of inventory products and the maximum lead time of requests. Including more antigens increases the number of possible blood types, which increases the number of rows in the matrix. The number of columns depends on the maximum age and lead time. The dimension of the state can therefore be formulated as $2^n * (A + LT)$, where n represents the number of included antigens, A the maximum age of an RBC unit, and LT the maximum lead time of a request.

4.2.2 Action space In a given state, the agent must choose a blood unit to issue according to the request at hand. The action space, therefore, consists of all the possible blood groups included in the environment configuration. In the same way, the blood groups are represented as integers in the state, the agent's action is simply an integer representing a blood group. The total number of possible actions equals 2^n, where n represents the number of antigens considered.

4.2.3 Step simulation Episodes in the simulation consist of days and each day has several requests that the agent must handle. The distribution process is handled request-by-request, so in a single time step, the agent is handling one request (indicated by the 9^{th} column in the state matrix) and will choose a single unit to match this request. When the agent chooses an action, the state will be updated accordingly. Matrix I is updated by removing the chosen blood product from inventory, if it is available, where the oldest product is removed first. The handled request is removed from matrix R. This way, the agent proceeds to handle all requests, until all requests on a single day have been handled. When a new day starts, the age of all products is increased, outdated products (with an age > 35) are removed, the lead time of requests is decreased, the inventory is filled up to its maximum capacity of 150 products, and new requests are sampled for the day. Filling the inventory up to its maximum capacity ensures that shortages and outdated units can be attributed to the issuing policy and not to the inventory levels of the blood bank.

4.2.4 Reward calculation In the reward system, the agent is only penalized with negative rewards, meaning the maximum (best) reward an agent can get is 0. A reward is calculated for each action the agent chooses so for each time step. Reward calculation is shown in Algorithm 2. When a unit is matched with a request, relative immunogenicity weights are used to calculate penalties for mismatched antigens. The weights used in this study are shown in Table 7.

2: Reward system

Input : Inventory I, issued product i, request r,
weights w for each antigen a

Initialize: Reward $R = 0$

```
if i ∈ I then
    // Compatible on the A, B, and Rh(D) antigens
    if i and r are not compatible then
        // Penalty of 10 for shortage and 1 for discarded product
        R -= 10 + 1
    else
        for i in v do
            if a ∉ r then
                R -= w[a]
else
    R -= 50
```

Antigen	Immunogenicity weights
Fy^a	0.0443
Fy^b	0.0131
Jk^a	0.0836
Jk^b	0.0033

Table 7. Relative immunogenicity weights for the antigens included in this study. Weights are derived from the relative immunogenicity values stated in Table 6.

5 Experimental analysis

Various experiments are performed to identify which methods and hyper-parameters would result in the best-performing Deep Q-learning model. Several different measurements are plotted and shown in this section:

1. Training reward per simulated day. Training is performed online, meaning the agent is exposed to new training data at each time step. Training reward shows how well the agent learns to choose actions that are valuable.
2. Number of matched requests per day. I.e. requests matched in lines 4 - 7 in Algorithm 2. This measurement provides more insight of DQN's performance in the actual problem.
3. Shortages of RBC units (issuing products that are not compatible and therefore discarded) and outdating of RBC units (units remaining in inventory with an age > 35 days). Goals of the distribution policy is to prevent shortages and outdated products as much as possible. Hence, these measurements are reported.

Performance measurements are plotted per simulation day. An episode has 800 simulation days. 800 days provides a balance between a reliable training time and the required computational time required. On average, a day will have 50 requests (or 50 time steps). Episodes are indicated in the plots by the blue vertical lines. Considering the time scope of this project, only 3 sets of antigens are researched: a 3-antigen setting with A, B, and Rh(D), a 5-antigen setting with A, B, Rh(D), Jk^a, and Jk^b, and a 7-antigen setting A, B, Rh(D), Jk^a, Jk^a, Fy^a, and Fy^b. The resulting size of the state matrices are 8 x 43, 32 x 43, and 128 x 43, for each setting respectively. The architecture and hyperparameters used for the Q-network and the DQN methods are shown in Table 8.

5.1 Learning rate tests

To ensure efficient updating of the Q-network weights, experiments with varying learning rates are performed. A high learning rate can provide fast convergence, but the model might never learn the optimal weights, since the weight adjustments are too large to move in the optimal direction. While a low learning rate could cause slower convergence, but a higher chance of finding the optimal weights. The network was tested with learning rates $1E-2$, $1E-3$, and $1E-4$. All experiments were performed for 30 episodes or $800 * 30 = 24.000$ simulation days.

Figure 5 depicts the daily total training reward and the proportion of matched requests for various learning rate tests. The left column illustrates results for the 3-antigen setting using antigens A, B, and Rh(D), while the right column displays outcomes for the 5-antigens setting, including A, B, Rh(D), Jka, and Jkb. Across all experiments, it is clear that adjusting the learning rate significantly impacts the performance and computational time.

Hyperparameter	Value
Optimizer	Adam
Input	8 x 43 = 344 (3 antigens) 32 x 43 = 1376 (5 antigens)
Batch size	64
Output	8 (3 antigens) 32 (5 antigens)
Layers	64, 32
Learning rate*	$1E-2/E-3/1E-4$
Activation	ReLu
Loss function	MSE
ϵ	0.1
γ	0.98
DDQN (f)*	100/1000

Table 8. Architecture and hyperparameters for the Q-network and DQN methods used in the experiments. Hyperparameters denoted with * vary depending on the experiment. The differences in NN architecture between the 3-antigen setting and the 5-antigen setting are also noted. ϵ is the probability of choosing a random action, γ is used to prioritize future rewards vs immediate rewards (see section 2.3)

The highest learning rate, $1E-2$, learns the quickest but reaches the lowest performance in most experiments. In the 3-antigen setting, the difference in performance is not significant compared to the other learning rates. After 30 episodes, the reward converges at slightly above -250 and the proportion of matched requests reaches 0.9. The other learning rates only slightly outperform $1e^{-2}$. In the 5-antigen setting, however, this learning rate does again learn the quickest but reaches a much worse performance compared to the other learning rates.

A learning rate of $1E-4$ is, as expected, the slowest learner in all experiments. In the 3-antigen setting, convergence is attained, but it takes slightly longer than with the other learning rates. Over time, this learning rate does achieve a proportion of matched requests above 0.9. In the 5-antigen setting (second column in the figure), it can be observed that the model has not reached convergence yet after 30 episodes of training, indicating that, with longer training time, a learning rate of $1E-4$ could achieve better results.

A learning rate of $1E-3$ provides our optimal balance between computational time and algorithm performance. In the 3-antigen setting, the agent consistently matches over 90% of requests regularly. In the 5-antigen setting, the agent's reward and proportion of matched requests are much higher after 30 episodes than in the other settings. Daily reward converges around -250, with the percentage of matched requests nearing 90%, approaching the performance level observed in the 3-antigen setting.

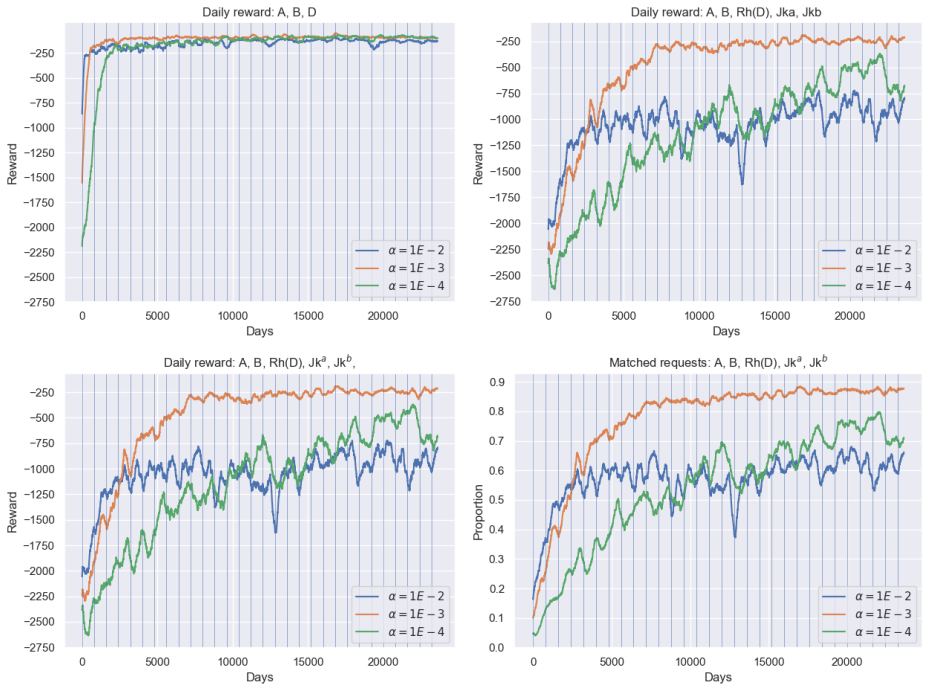

Fig. 5. On the left, the total daily training reward is plotted for the various learning rate tests. Experiments were performed with learning rates $1E-2$, $1E-3$, and $1E-4$. On the right, the proportion of matched requests is plotted. The total training time was 30 episodes for each setting.

5.2 Pre-training the Q-network

For this study, the MINRAR optimization code is available, which allows for using this near-optimal matching policy to train the Q-network in a supervised manner, before starting the Q-learning algorithm. This way, the Q-network will be pre-trained and the agent will have some prior knowledge to estimate the quality of the task. Which may increase algorithm performance and decrease computational time.

For a total of [x simulated days/episodes] MINRAR is used to assign inventory products to requests. For every issued product, the current inventory, known requests, and the match that is made are transformed into a state-action pair as described in Section 4.2. Each state-action pair is stored as data, and finally used to train the Q-network to predict the MINRAR-optimal action given a state. Subsequently, the weights of this neural network are saved and used to 'kick-start' the Q-learning process.

5.2.1 Supervised Learning The architecture of the neural network is shown in Table 9. In our previous learning rate tests, a learning rate of $1E-3$ provided the best trade-off between computational time and performance, hence we continue with this setting. The cross entropy loss is used, since this problem can be defined as a multi-class classification problem [9]. In order to simplify copying of the weights of the trained network to the Q-network in DQN, the rest of the architecture is kept the same.

Hyperparameter	**Value**
Optimizer	Adam
Input	$8 * 43 = 344$
Batch size	64
Output	8
Layers	64, 32
Learning rate	$1E-3$
Activation function	ReLu
Loss function	Cross Entropy Loss

Table 9. Architecture and hyperparameters used in pre-training the Q-network.

An analysis of the class distribution, which can be found in Appendix A.1, shows that the dataset for pre-training the Q-network is highly imbalanced. Actions 1 and 5, representing blood groups O+ and A+, are executed much more often than all other classes. Imbalances in the dataset could lead to poor performance of the network on minority classes. Therefore, a weighted sampler was implemented that, during training, uses the class frequencies to sample all classes evenly.

The data was split into a train set (70%) and a validation set (30%). The validation set is used to test the performance of the network on unseen data. The neural network was trained for 50 epochs. An epoch is a complete run through the available data. Each epoch consists of 250 training batches and 50 validation batches. Since the batch size is 64, a total of 16.000 training samples and 3.200 validation samples were used. The training and validation loss and accuracy are depicted in Figure 6.

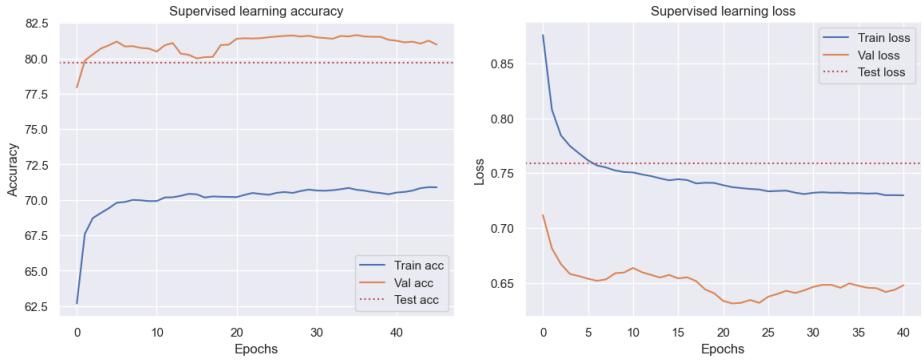

Fig. 6. The accuracy and loss plots over time for training a neural network on the 3-antigen setting MINRAR optimized data. The neural network architecture is described in Table 9. The model was trained for 50 epochs. 250 batches for training and 50 batches for validation. The model performs much better on the validation set compared to the training set.

The training graphs show that the model's performance increases quickly, plateauing at around 15-20 epochs, with the validation accuracy consistently reaching 80 - 82 %. Remarkably, the graphs also show that the model performs much better on the validation set then on the training set. The validation accuracy achieved is around 10 p.p. higher and the loss is 0.1 lower than the training loss. This can be attributed to the underlying distribution of the validation set compared to the test set. Due to the weighted sampler, the train set contains evenly distributed classes, whereas, in the validation set, classes 1 and 5 are over-represented. Even though a weighted sampler is used, the variety in classes 1 and 5 still results in better generalizability of these classes to new data. The weighted sampler does ensure greater performance overall, which is confirmed by re-running the experiments without sampling (results are depicted in Appendix A.1). On the test set, the network achieves a loss of 0.7592 and an accuracy of 79.67%.

The same supervised learning experiment was performed on the 5-antigen setting. A similar class imbalance was found and can be viewed in Appendix A.1. The model was again run for 50 epochs. The accuracy and loss plots are shown in Figure 7.

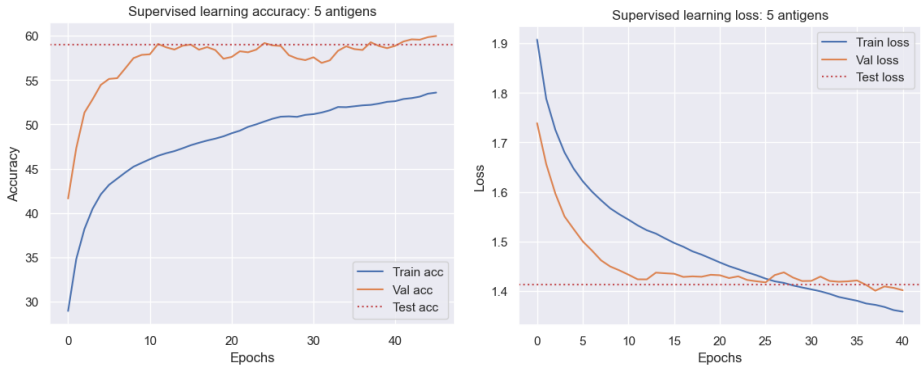

Fig. 7. Accuracy and loss over time for supervised training of the 5-antigen setting. Architecture can be found in Table 8. Training was done for 50 epochs.

Similarly to the 3-antigen setting, the model achieves better accuracy performance on the validation set compared to the train set. Accuracy reaches approximately 60% and 54%, for the validation and train set, respectively. Again, this can be attributed to the difference in class distribution and variety between the two sets. On the test set, an accuracy of 59% and a loss of 1.4124 is achieved.

5.2.2 Supervised learning: convolution The low performance of the NN on the 5-antigen setting, compared to the 3-antigen setting, could be due to the increased dimension of the state matrix (32 x 43, compared to 8 x 43). Flattening the matrix and feeding it into a fully connected NN might complicate finding relations within the data and result in a more complex task to learn. To investigate this, a convolutional neural network (CNN), that takes the complete unflattened matrix as input, was also implemented to investigate if higher accuracy rates could be achieved. A simple CNN was implemented, the architecture is described in Table 10. In the CNN architecture, feature maps refer to the output of a layer in a CNN. The kernel size is the size of the matrix used for convolution. The stride is the step size used for moving the kernel across the input and the padding refers to adding pixels on the edges of the input data to retain the original dimensions.

Accuracy and loss plots are displayed in Figure 8.

	Type	Feature maps	Kernel size	Stride	Padding
Layer 1	Conv2D	32	3 x 3	1	1
Layer 2	MaxPool	-	2 x 2	2	
Layer 3	Conv2D	64	3 x 3	1	1
Layer 4	Max Pool	-	2 x 2	2	
Layer 5	Conv2D	128	3 x 3	1	1
Layer 6	Linear	64	-	-	
Layer 7	Linear	32	-	-	

Table 10. Architecture used for the implemented CNN in the convolutional supervised learning experiment.

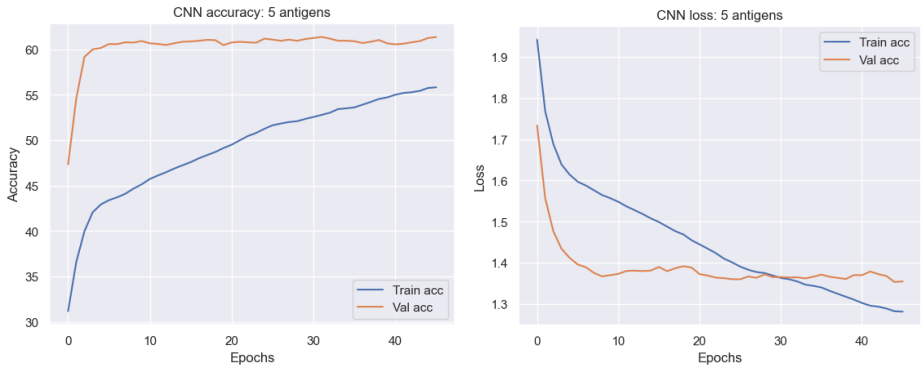

Fig. 8. Training accuracy and loss plots for the 5-antigen setting, using the CNN from Table 8.

As can be seen in the plots, convolution only improves accuracy by a small margin (61% and 55%) compared to flattening the matrix and using a feed-forward NN. Hence, further experiments are continued with the previous fully connected NN. It could also be argued that an accuracy of 60% is good enough since the aim is to simply kick-start the Q-learning, not to reach the most optimal NN.

5.2.3 Kick-starting Q-learning The trained weights from the supervised learning experiments are saved and used to 'kick-start' the Q-learning. Figure 9 shows the performance in the 3-antigen setting with and without a pre-trained Q-network.

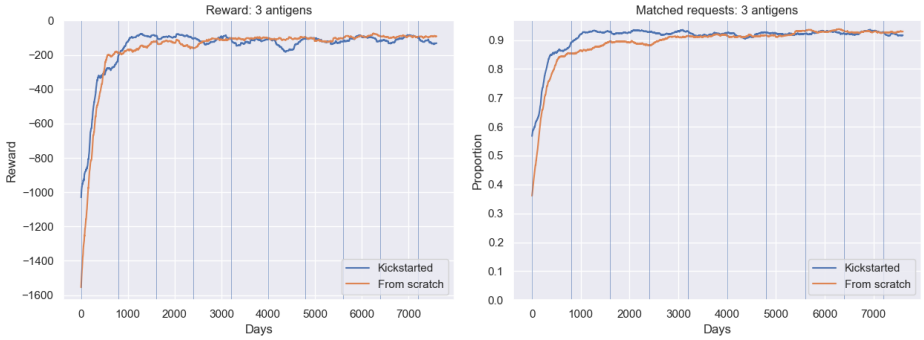

Fig. 9. The reward plot (left) and the proportion of matched requests (right) for kick-starting the Q-network versus training from scratch. Training was done for 10 episodes. The architecture of the NN is shown in Table 9.

Using a pre-trained network makes the agent start at a higher base reward, around -1,000 kick-started versus -1,500 from scratch, and the agent reaches higher rewards slightly faster. In the proportion of matched requests, a similar performance increase is observed: a higher start base (approx. 0.6 vs 0.4) and the proportion of matched requests reaches rates above 0.90 after 1,000 simulation days (kick-started), while training from scratch takes around 2,500 days to reach the same performance. Training from scratch does catch up with the kick-started network over time. This slight increase in performance and computational speed might be even greater in more complex settings, such as using 5 or 7 antigens. Hence, the same experiment was performed in the 5-antigen setting.

The results of the same experiments in the 5-antigen setting with the learning rate at $1 * E - 3$ and $1 * E - 4$ are shown in Figure 10.

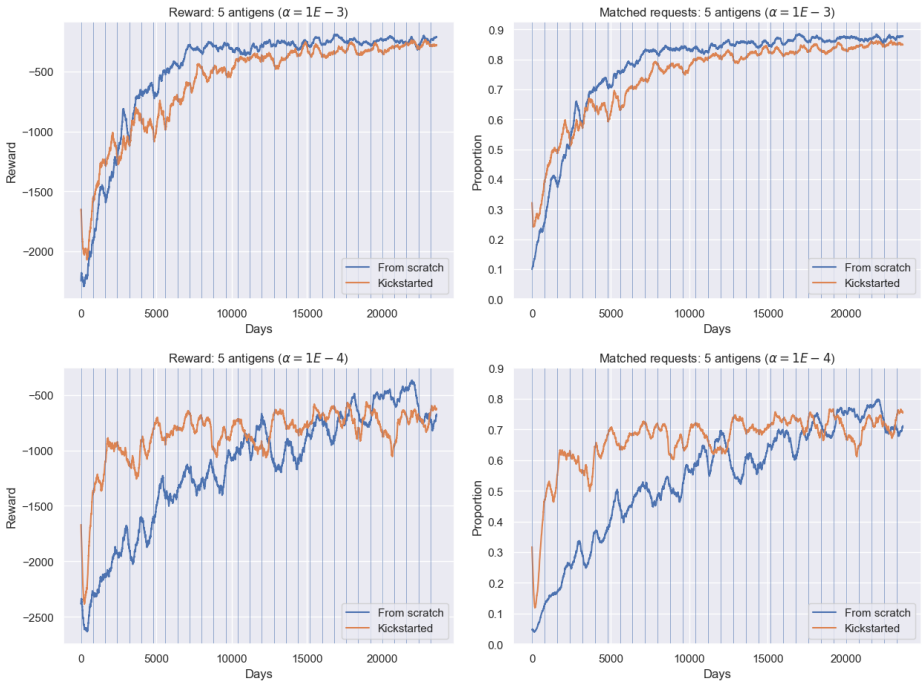

Fig. 10. Daily total training reward (left) and proportion matched requests (right) with 'kick-starting' the Q-network in the 5-antigen setting. Learning rates $1E-3$ and $1*E-4$ were tested. Training was performed for 50 episodes.

With the learning rate set at $1E-3$, the kick-started network learns slightly faster but is quickly caught up with by the network trained from scratch. With the learning rate set at $1E-4$, the kick-started network learns much faster. The model consistently matches over 60% of requests after 5,000 simulation days compared to 10,000 days when starting from scratch.

5.3 Double Deep Q-Learning

To investigate whether performance could be increased further, a variation of the Deep Q-learning algorithm was implemented: Double Deep Q-learning. In the original DQN algorithm, the *max* argument in the update function could lead to an overestimation of Q-values and poorer policies. A solution for this is the addition of a second Q-value estimator, first proposed in 2010 by van Hasselt et al [14]. In Deep Q-learning, this second estimator is implemented in the form of an additional Q-network [15]. The second estimator is used to evaluate the states to prevent overestimation of Q-values. Updates to the second estimator's weights are only performed at a predefined interval of time steps.

Action selection is performed in the same way as in the original DQN, using the Q-network with weights θ. Evaluation of states during the update of the Q-network, however, is done with the additional Q-network with weights θ'. This means the update function changes as follows:

$$y_s = r_s + \gamma * max_{a_s+1} Q((s_{s+1}), a_{s+1}; \theta') \tag{5}$$

An additional hyperparameter (f) is introduced that controls after how many updates the weights of the Q-network (θ) are copied to the second Q-network (θ').

Experiments are performed with varying values for f to investigate whether Double Deep Q-learning improves performance. In the 3-antigen setting, the same architecture as described in Table 5 is used, with the learning rate set at $1E-3$. Experiments were performed by updating the double Q-network each 100 time-steps and 1,000 time-steps ($f = 100$ and $f = 1000$). The algorithm was run for 10 episodes, the reward and number of matched requests are plotted in Figure 11.

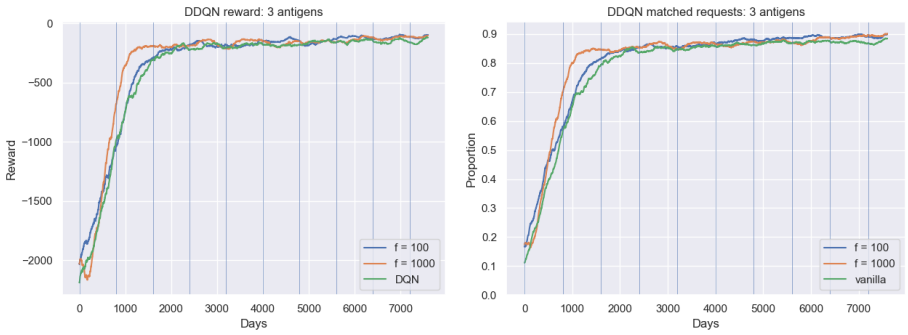

Fig. 11. On the left, the daily training reward for the various frequency updates is shown. On the right, the number of matched requests is plotted. The performance of the original Deep Q-learning algorithm (DQN) is also included.

Updating the second Q-network every 100 time-steps slightly improves speed and performance in the 3-antigen setting. A frequency of 1,000 results in the fastest learning: matching 80% of requests after 1,000 days versus around 1,500 days in the original DQN.

Similar experiments were conducted in the 5-antigen setting. Results are shown in Figure 12. Again, faster learning can be observed when the DDQN algorithm is implemented. A frequency of 100 does not differ much from a frequency of 1000.

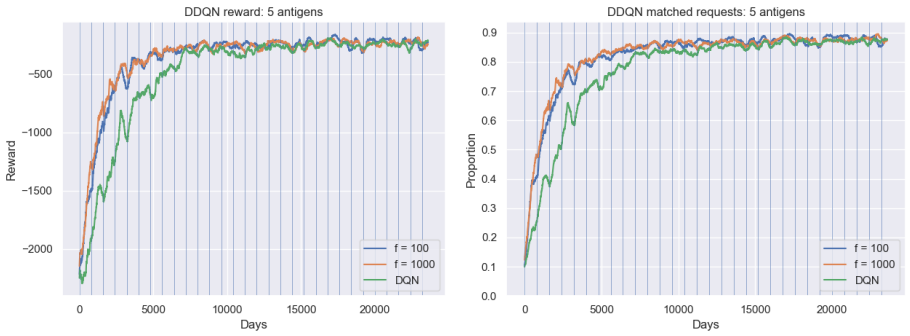

Fig. 12. Daily reward plots and number of matched requests in the DDQN algorithm for the 5-antigen setting.

5.4 Comparison to MINRAR

To evaluate the performance of the Q-learning models, the best-performing models are compared to the MINRAR model. For the 3-antigen setting, this means kick-starting the network before the Q-learning and using the DDQN algorithm with the frequency set at 1,000. For the 5-antigen setting, the same architecture is used, apart from kick-starting, as did not yield any improved results. The DQN algorithm was tested on the same test episodes in both the 3- and 5-antigen setting. The test set consisted of 2,400 simulation days in total. The number of matched requests, shortages, outdated units, and the number of antigen mismatches (5-antigen setting) are stated in Table 11.

	M. requests	Shortages	Outdates	Mism. Jka	Mism. Jkb
DQN 3 antigens	112,174 (92.98%)	8,473 (2.35%)	108 (0.003%)	-	-
MINRAR 3 antigens	**120,032 (99.49%)**	**48 (0.04%)**	**41(0.001%)**	-	-
DQN 5 antigens	104,490 (86.61%)	16,157 (4.48%)	2,571 (0.07%)	18,462 (0.15 p.r.)	21,134 (0.18 p.r.)
MINRAR 5 antigens	**120,129 (99.43%)**	**688 (0.19%)**	**123** (0.003%)	**8,568** (0.07 p.r.)	**7,992** (0.07 p.r.)

Table 11. Matched requests, shortages, outdated units, and the number of antigen mismatches for the DQN models compared to MINRAR. Total numbers for the entire test simulation (2,400 days) are given. For the matched requests, the proportion of the total number of requests (120.647) is given. For the shortages and outdated units, the proportion of the total number of products in inventory is given. The average number of mismatches per request (p.r.) is also stated.

Hyperparameter	**Value**
Optimizer	Adam
Input	$8 * 43 = 344$ (3a)
	$32 * 43 = 1376$ (5a)
Batch size	64
Output	8 (3a)
	32 (5a)
Layers	64, 32
Learning rate	$1E - 3$
Activation function	ReLu
DDQN	True, $f = 100$
Loss function	MSE

Table 12. Architecture and hyperparameters of the Q-networks used for testing. For the in- and output, the values for both the 3-antigen (3a) and the 5-antigen (5a) settings are shown.

Table 11 shows that DQN has not reached the performance of MINRAR yet. MINRAR matches considerably more requests with a compatible product in both the 3-antigen (99.49% vs 92.98%) and the 5-antigen setting (99.43% vs 86.61%). When considering 5 antigens, MINRAR has an average of 0.07 antigen mismatches per request for both Jka^a and Jka^b. More than half the amount of DQN: 0.15 and 0.18, for Jka^a and Jka^b, respectively.

MINRAR was also evaluated using the reward system in the DQN algorithm. This shows how MINRAR would perform if its unit distribution was evaluated using the same conditions as in the DQN model. As expected from the statistics in Table 11, MINRAR reaches higher rewards than DQN when evaluated with the same reward system. MINRAR achieved an average reward of -19.99 vs -41.12 for the DQN model in the 3-antigen setting. In the 5-antigen setting, MINRAR got a reward of -77.89 versus -269.09 for the DQN model.

5.5 Policy analysis

The reward system in the DQN algorithm is set to optimize the policy to achieve the following objectives:

1. Prevent discarded units by matching compatible RBC units.
2. Allocate RBC units that are existent in inventory.
3. Prevent discarding of RBC units due to outdating.

To investigate whether the implementation achieved the desired results, the number of units that are 1)issued but non-existent, 2)issued but discarded, and 3)outdated are plotted over the training time in Figure 13.

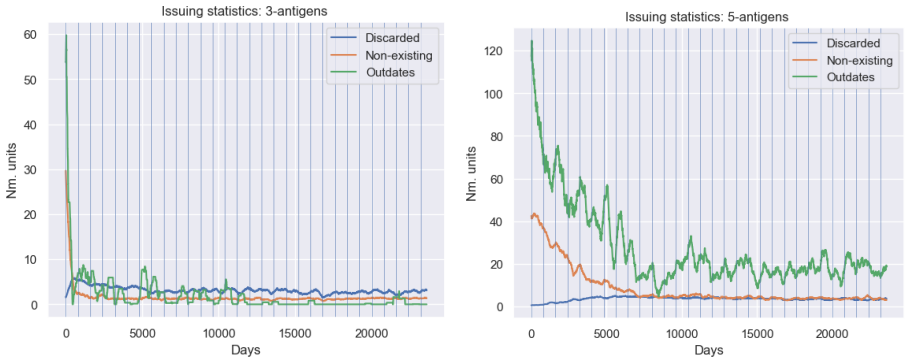

Fig. 13. Number of discarded, outdated, and non-existing issued units in the best-performing 3-antigen and 5-antigen setting.

As depicted in the plots, there is a noticeable decline over time in both the number of outdated units and non-existent units. This indicates that the policy is indeed optimized in the correct direction. Interestingly, although the number of units discarded remains low, it increases slightly over time.

For the 5-antigen setting, the policy is further optimized with an additional condition: minimizing the number of minor antigen mismatches. Figure 14 displays the number of relative mismatches for antigens Jk^{a} and Jk^{b} over time. Relative mismatches are used because the total mismatches are expected to rise as the agent learns to match more compatible units. For an antigen x and a total number of daily requests r, the relative mismatches per day are calculated as follows:

$$\text{Relative mismatches} = \frac{\sum_{j=0}^{r} m_{j,X}}{s} \tag{6}$$

Where M denotes the number of mismatching antigens for that specific match and s denotes the total number of matched requests.

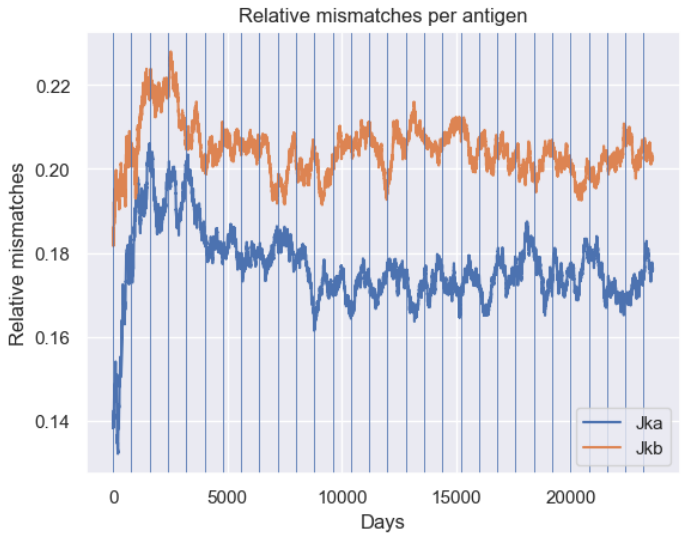

Fig. 14. Relative mismatches per antigen for the DDQN algorithm in the 5-antigen setting. Relative mismatches are calculated according to equation 6.

As the agent issues units, the relative number of mismatches for both antigens increases in the beginning. After some training, relative mismatches decrease slightly, however, a more or less stable value of around 0.17 for Jk^b and 0.21 for Jk^a is reached after approximately 10,000 training days. Mismatches for antigens Jk^a and Jk^b follow a similar trend.

5.6 7 antigens

Finally, experiments were performed including a total of 7 antigens: A, B, D, Jk^a, Jk^b, Fy^a, and Fy^b. Since this is a much more complicated task, with a state matrix of 128 x 43, the algorithm was run for 100 episodes or 8,000 simulation days. The same architecture was used, described in Table 5, with a learning rate of $1E-3$. Figure 15 shows the daily reward and the number of matched requests.

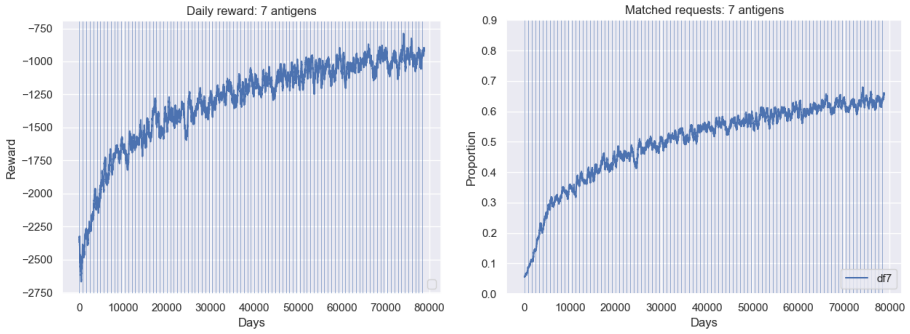

Fig. 15. The reward and number of matched requests for the 7 antigen-setting. The model was run for a total of 100 training episodes.

After 100 training episodes, the model reaches rewards of around -1000 and matches 60% of requests. The algorithm has not converged yet, indicating more training time is required, which expected considering the complexity of the 7-antigen setting.

6 Discussion

6.1 Overall performance

This study showed that DQN is suited for creating an issuing policy for the distribution of RBC units. In the 3-antigen setting, DQN could consistently match over 90% over requests and up to 90% in the 5-antigen setting. Shortages and outdated units were kept to 2.35% and 4.48%, for the 3- and 5-antigen settings respectively. In the 5-antigen setting, the average number of mismatches per minor antigen remains quite low: 0.15 and 0.18, for Jk^a and Jk^b respectively. The number of units issued and discarded, non-existent units, and outdated units were minimized over time, showing the policy was steered in the desired direction. While promising, the DQN's performance is not near that of MINRAR yet. MINRAR can match up to 99% of requests in the same scenarios while keeping shortages to 0.19% and achieving roughly half the amount of mismatches per request.

It must be noted, though, that the current implementation of DQN is fairly simple. A small NN (2 hidden layers: 64, 32) is already able to capture the tasks of the 3- and 5-antigen settings relatively well. In the 7-antigen setting, this small NN can achieve 60% matched requests. A larger NN might achieve better results since it is better able to capture the relations within the data.

6.2 Methods review

Kick-starting the network seems promising. In the 3-antigen setting, the agent was able to learn faster and started at higher initial rewards. In the 5-antigen setting, however, the best agent was not kick-started. While kick-starting may provide some knowledge to the agent and increase computational speed, it could inhibit the agent; the prior knowledge might cause the agent to over-exploit certain actions since these will have high Q-values from the beginning. This might be the case here since there was a clear difference in the performance of the 3-antigen NN (79% accuracy) and the 5-antigen NN (60% accuracy). With more time, a better network could be trained that may exhibit a performance increase in the 5-antigen scenario and other antigen scenarios as well.

The Double Deep Q-learning algorithm clearly increased performance. Faster learning and higher rewards were observed. More experiments regarding the frequency of updating the second network should be performed in further research.

6.3 Future work

Future work should focus on extending the model to include more relevant antigens and improve overall performance to approximate, or surpass, that of MINRAR. The results for the 3-antigen, 5-antigen, and 7-antigen settings show that increasing the number of antigens significantly increases learning complexity. 7 antigens take over 100 training episodes to only match around 60% of requests. Including up to 14 antigens, which is the number that is considered clinically relevant for matching, might be too complex for the DQN agent to learn all at once.

A possible approach could be to implement a form of incremental learning. Incremental RL involves an environment that is dynamic and adaptable, meaning the state and action spaces can change. In this context, incremental learning could be used to expose the Q-learning model to more antigens over time. Initiating learning with a low number of antigens, such as 5 or 7, and incorporating more antigens gradually whenever the model reaches convergence. The challenge lies in the implementation of the states. Adding in antigens not only increases the number of input and output neurons in the neural network but also alters the ordering of the blood groups in the state matrix. A possible solution would be to start with the entire 14-antigen state and apply masks to the non-included antigens. Over time, the masks can be progressively removed to include more antigens. Incremental learning has proved its worth already in various other domains, such as computer vision and robotics [10] [27].

While DQN has shown promising results, other RL methods could also be explored to improve results. Actor-critic methods provide more stable and sample-efficient learning since they combine value estimation (which offers the advantage of low variance) with policy estimation (low bias) [20]. When including

more antigens, more training samples are necessary to achieve decent results. A state-of-the-art method that could deal with this high sample complexity, is Soft Actor-Critic [12]. Future work could also focus on more extensive reward shaping, specifically for mismatching. In the current implementation, mismatching has not decreased significantly over time. This might be because the relative immunogenicity weights used are too small compared to the rewards for the minimizing of non-existent products, discarded products, and outdated products. The rewards could be shaped dynamically over time: increasing the penalties for mismatching relative to the distribution of units, to emphasize mismatching more once the agent has learned to issue mostly compatible units.

Finally, a larger Q-network combined with more extensive hyper-parameter tuning may also achieve better results.

6.4 Thesis obstacles

During this MSc. thesis, various obstacles along the way limited overall progression. For a long period (4/5 months), undiscovered bugs in the algorithm code led to the assumption that DQN was not suitable for the RBC distribution task. The first part of the project consisted mostly of trying out many different experiments to just get the algorithm working. The conclusion that this task is simply not learnable by RL was already drawn and this was going to be the main conclusion of the thesis. It was not until small bugs in the code, relating to the copying of variables between classes, that the DQN algorithm started working. Once bugs were discovered, little time was left to re-run experiments and get the most out of the algorithm (such as extending to more than 7 antigens).

Before the DQN algorithm started working, many approaches were tried to just get the algorithm running. These include hyper-parameter optimization, decreasing the number of antigens, decreasing the maximum age and lead times to simplify learning, also the idea of 'kick-starting' the network originated from this issue to see if this would get the code working. The scope and direction of the thesis were therefore different from how it turned out in the end. When the algorithm did start working, it turned out that even simple methods and very little tuning already provided decent results.

Additionally, complications arose with the use of LIACS virtual machines. Regular downtimes, user storage limits, and killing of processes after running for periods of time-limited thesis progression as well.

7 Conclusion

The goal of this study was to answer the following research question:

1. Can Reinforcement Learning, specifically Deep Q-learning, be used to develop an issuing policy for the distribution of comprehensively typed RBC units?

Results showed that DQN is indeed suited for this task. With DQN, it is possible to develop an issuing policy that can match up to 86% of requests, when considering 5 antigens, while keeping the proportion of discarded and outdated units below 5%. A simple DQN implementation was also able to match up 60% of requests when including 7 antigens. The initial results are promising, but further research is necessary to extend the model to include more antigens and improve performance. Further research could include more extensive hyper-parameter tuning, incremental learning, and the use of more state-of-the-art RL methods.

A Supervised Learning

A.1 Class imbalance and classification reports

In the supervised learning experiments, described in section 5.2.1, the MINRAR optimized data was used to pre-train the Q-network. Due to the distribution of blood groups among the patient population, MINRAR's policy leads to a larger proportion of the O+ (class 1) and A+ (class 5) classes in the dataset. Figure 16 shows the histogram of the class occurrence in the dataset.

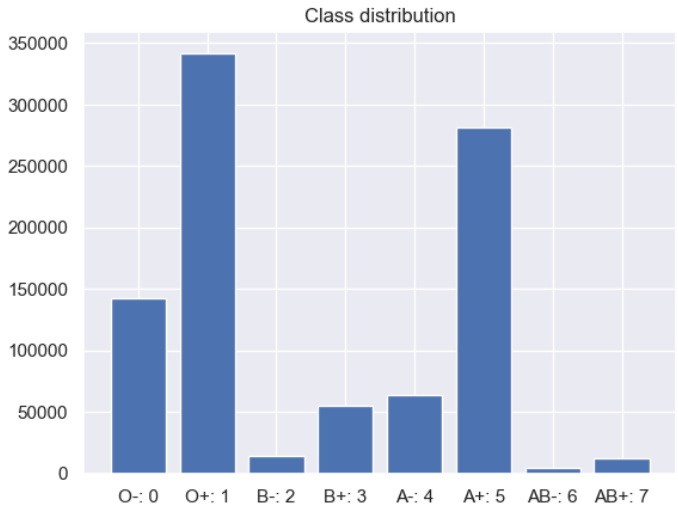

Fig. 16. The distribution of all classes in the 3-antigen MINRAR optimized dataset. Classes 1 (O+) and 5 (A+) are majority classes and represent the largest part of the dataset. The total number of data samples is 915.579.

Further analysis of the supervised training experiment includes a classification report, which shows the precision, recall, and F1-score per class.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

Where TP is the number of 'true positives' and FP the 'false positives'.

The recall calculates how many of the actual relevant classes were correctly predicted and is calculated as:

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

Where FN stands for 'false negatives'.

The F1-score is a combined metric of precision and recall and is calculated as follows:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{9}$$

The metrics for each of these measurements per class are shown in the classification reported depicted in Figure 17.

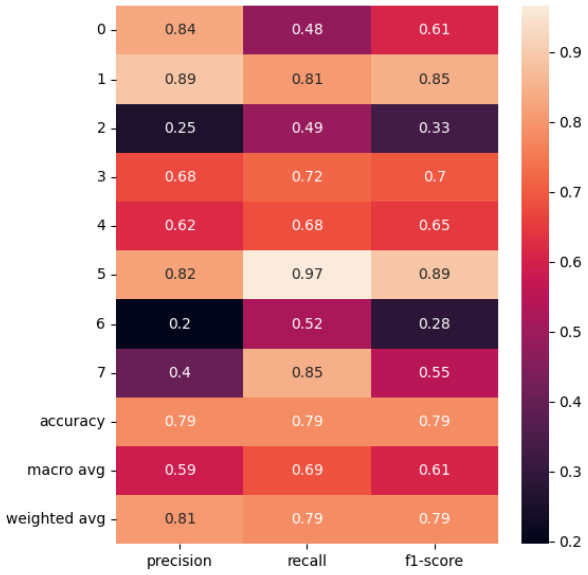

Fig. 17. The test set classification report for the 3-antigen setting. The y-axis labels (0-7) correspond to the integer mapping of the blood groups. The precision, recall, and F1-score are shown per class. The model achieves an accuracy of 0.79 on the test set. The model performs best in classes 1 and 5, with an F1-score of 0.85 and 0.89, respectively.

Fig. 18. Class distribution in the 5-antigen dataset. Similarly to the 3-antigen dataset, classes are highly imbalanced, with classes 7 and 23 being the biggest.

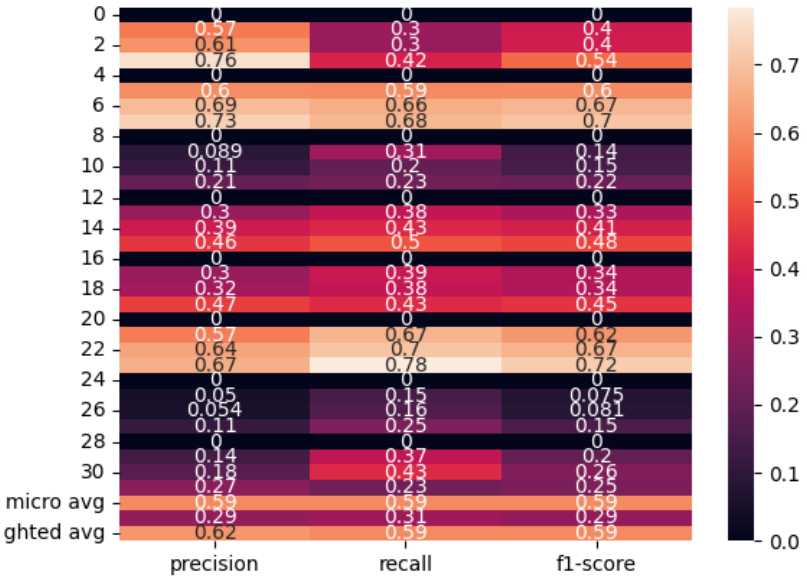

Fig. 19. Precision, recall, and f1-score for each class in the 5-antigen supervised learning task.

A.2 Supervised learning without weighted sampler

When using a weighted sampler, the NN performed better on the validation set than on the test set. To confirm this was due to the variation in distribution between the two tests, caused by the weighted sampler, the experiments were run again without a sampler. Results are depicted in Figure 20. The plots show that, when a weighted sampler was not used, the classifier overfits quickly on the training set.

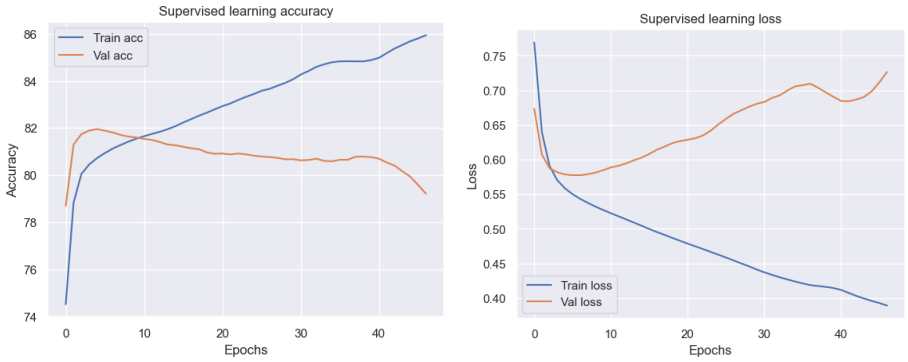

Fig. 20. Accuracy and loss when training an NN on the MINRAR optimized data without a weighted sampler.

References

1. Erytrocyten, https://www.sanquin.org/nl/producten-en-diensten/bloed-producten/voor-onderzoek/erytrocyten-in-sagm-niet-voor-therapeutisch-gebruik, accessed: 15-11-2023
2. Jaarverslag sanquin, https://www.sanquin.nl/binaries/content/assets/sanquinnl/over-sanquin/pers--actueel/jaarverslagen/jaarverslag-stichting-sanquin-bloedvoorziening-2022.pdf, accessed: 01-11-2023
3. Match study, https://www.sanquin.org/research/ctr-studies/match, accessed: 01-11-2023
4. Bhuva, D., Vachhani, J.: Red cell alloimmunization in repeatedly transfused patients. Asian Journal of Transfusion Science **11**(2), 115 (2017). https://doi.org/10.4103/0973-6247.214347
5. Chandra, T., Solanki, A., Singh, A.: Prevalence of red blood cell antibodies in whole blood donors: A single-centre experience in north india. Indian Journal of Medical Research **152**(3), 280 (2020). https://doi.org/10.4103/ijmr.ijmr$_2$96$_1$9
6. Chung, Y., Kim, J.S., Youk, H.J., Kim, H., Hwang, S.H., Oh, H.B., Ko, D.H.: Relative immunogenicity of blood group antigens: First report in a korean population. Transfusion and Apheresis Science **62**(2), 103585 (2023). https://doi.org/10.1016/j.transci.2022.103585
7. Dean, L.: Chapter 5 The ABO blood group. NCBI (2005)
8. Dean, L.: Chapter 7, The Rh blood group. NCBI (2005)
9. Demirkaya, A., Chen, J., Oymak, S.: Exploring the role of loss functions in multiclass classification pp. 1–5 (2020). https://doi.org/10.1109/CISS48834.2020.1570627167
10. Dou, J., Li, J., Qin, Q., Tu, Z.: Moving object detection based on incremental learning low rank representation and spatial constraint. Neurocomputing **168**, 382–400 (2015). https://doi.org/https://doi.org/10.1016/j.neucom.2015.05.088
11. Goodell, P.P., Uhl, L., Mohammed, M., Powers, A.A.: Risk of hemolytic transfusion reactions following emergency-release rbc transfusion. American Journal of Clinical Pathology **134**(2), 202–206 (2010). https://doi.org/10.1309/ajcp9ofjn7fltxdb

12. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. CoRR **abs/1801.01290** (2018), http://arxiv.org/abs/1801.01290
13. Harewood J, Ramsey A, M.S.: Hemolytic transfusion reaction. StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing (2022)
14. Hasselt, H.: Double q-learning. In: Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Advances in Neural Information Processing Systems. vol. 23. Curran Associates, Inc. (2010), https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf
15. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. CoRR **abs/1509.06461** (2015), http://arxiv.org/abs/1509.06461
16. Isbt: Red cell immunogenetics and blood group terminology: Isbt working party, https://www.isbtweb.org/isbt-working-parties/rcibgt.html#:~:text=There%20are%20currently%2045%20recognised,genetically%20determined%20by%2050%20genes.
17. Konda, V., Tsitsiklis, J.: Actor-critic algorithms. In: Solla, S., Leen, T., Müller, K. (eds.) Advances in Neural Information Processing Systems. vol. 12. MIT Press (1999), https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf
18. Li, H.Y., Guo, K.: Blood group testing. Frontiers in Medicine **9** (2022). https://doi.org/10.3389/fmed.2022.827619
19. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR **abs/1312.5602** (2013), http://arxiv.org/abs/1312.5602
20. Plaat, A.: Deep reinforcement learning. CoRR **abs/2201.02135** (2022), https://arxiv.org/abs/2201.02135
21. Plaat, A.: Deep Reinforcement Learning, chap. Deep Value-Based Reinforcement Learning. Springer Nature (2022)
22. Ray, S., Tadepalli, P.: Model-Based Reinforcement Learning, pp. 690–693. Springer US, Boston, MA (2010). https://doi.org/10.1007/978-0-387-30164-8$_5$56, https://doi.org/10.1007/978-0-387-30164-8_556
23. van Sambeeck, J.H., van Brummelen, S.P., van Dijk, N.M., Janssen, M.P.: Optimal blood issuing by comprehensive matching. European Journal of Operational Research **296**, 240–253 (1 2022). https://doi.org/10.1016/j.ejor.2021.02.054
24. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization. CoRR **abs/1502.05477** (2015), http://arxiv.org/abs/1502.05477
25. Suddock JT, C.K.: Transfusion reactions. StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing (2023)
26. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (2018), http://incompleteideas.net/book/the-book-2nd.html
27. Wang, M., Wang, C.: Learning from adaptive neural dynamic surface control of strict-feedback systems. IEEE Transactions on Neural Networks and Learning Systems **26**(6), 1247–1259 (2015). https://doi.org/10.1109/TNNLS.2014.2335749
28. Watkins, C.J.C.H., D.: Q-learning. mach learn. Springer (1992). https://doi.org/https://doi.org/10.1007/BF00992698
29. van de Weem, R.H., Wemelsfelder, M.L., Luken, J.S., de Haas, M., Niessen, R.W., van der Schoot, C.E., Hoogeveen, H., Janssen, M.P.: Preventing alloimmunization using a new model for matching extensively typed red blood cells. Vox Sanguinis **117**, 580–586 (4 2022). https://doi.org/10.1111/vox.13217