UiT The Arctic University of Norway

Faculty of Science and Technology, Department of Physics and Technology

Evaluating Deep Q-Learning Techniques for Controlling Type 1 Diabetes

Sigurd Thorvald Nordtveit Hjerde

FYS-3900 Master's thesis in physics – 60 ECTS – February 2020

Abstract

Patients with type 1 diabetes (T1D) must continually decide how much insulin to inject before each meal to maintain an acceptable level of blood glucose. Recent research has worked on a solution for this burden: the artificial pancreas (AP), which is a closed-loop system combining a continuous glucose monitor (CGM) and an insulin pump with a decision-making algorithm.

The goal of this thesis is to implement and evaluate several hybrid closed-loop deep Q-learning (DQL) algorithms for the task of regulating the blood glucose in T1D patients. Firstly, we will review the diabetes disease, its burdens and challenges, and existing treatment models. Secondly, we will study the foundations of reinforcement learning (RL) and deep reinforcement learning (DRL), with the emphasis on DQL techniques. Then we will merge the theories and implement DQL algorithms with the application of regulating blood glucose for T1D in-silico patients. Finally, we will test these algorithms on a T1D glucoregulatory simulator.

Acknowledgements

This thesis would not have been possible without the help from many people. First of all, I would like to thank my supervisor Professor Fred Godtliebsen for his guidance and counsel.

I would also like to thank my co-supervisor Jonas N. Myhre, for his patience and willingness in aiding me through endless questions while writing this thesis, and pointing me in the right directions. And thank you, Miguel, for your assistance and determination that helped me in all aspects of this thesis.

Furthermore, I would like to thank Phuong for the informative discussions, the additional supervision, and academic contributions.

I would also like to thank my fellow students. Thank you for the inspiration, the discussions we had, our collaboration and the support. It has proven to be an invaluable source of inspiration, motivation and knowledge.

Lastly, to my friends, my family, and to Nathalie, thank you for your love and support, and for giving me the strength to accomplish this work.

Sigurd Thorvald Nordtveit Hjerde,
Tromsø, February 2020.

Table of Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
AP	**A**rtificial **P**anreas
BG	**B**lood **G**lucose
CHO	Carbohydrate (**C**arbon **H**ydrogen **O**xygen)
CGM	**C**ontinuous **G**lucose **M**onitor
CNS	**C**entral **N**ervous **S**ystem
DRL	**D**eep **R**einforcement **L**earning
DQL	**D**eep **Q** **L**earning
DQN	**D**eep **Q** **N**etwork
MDP	**M**arkov **D**ecision **P**rocess
MESS	**M**eal intake, **E**xercise, **S**leep and **S**tress
MPC	**M**odel **P**redictive **C**ontrol
NN	**N**eural **N**etwork
RL	**R**einforcement **L**earning
T1D	**T**ype **1** **D**iabetes
T2D	**T**ype **2** **D**iabetes

Chapter 1

Introduction

1.1 Diabetes Mellitus

Diabetes is a chronic disease that occurs when the natural blood glucose (BG) dynamics are interrupted. This happens when pancreas is no longer able to make enough insulin - type 1 diabetes mellitus (T1D), or when the body cannot utilize the insulin it produces - type 2 diabetes mellitus (T2D). Insulin is a hormone made by the pancreas, that acts like a key to let glucose pass from the bloodstream into the cells in the body to produce energy. When insulin is present in the bloodstream, it stores the glucose into the cells by breaking down the carbohydrates, and then lowering the BG levels. In either case of diabetes, the consequences leads to raised BG levels (hyperglycemia) [1]. In the long-run, high glucose levels in the blood are associated with damage to the body and failure of various organs and tissues. Currently there is no existing cure for this disease [2].

According to the International Diabetes Federation (IDF), over 425 million people have diabetes worldwide, and there will be approximately 629 million people with diabetes in the world in 2045 [3]. Around 10% of all people with diabetes have T1D [4]. This type of the disease is caused by an autoimmune reaction where the patient's immune system attacks the β cells that produce insulin from the pancreas [5]. As a result, this body produces little to no insulin. Patients with T1D therefore need daily injections of insulin to maintain healthy glucose values. They administer three to five insulin injections per day to regulate their BG levels in the target range (70-180 mg/dL). The primary goal of this BG regulation is to maintain the BG levels within a narrow range. This tight regulation is referred to as glucose homeostasis, and when BG levels are normal we refer to it as **normoglycemia**.

Figure 1.1: Diabetes equipment, insulin pen and BG level test [6].

1.1.1 Major Burdens

Diabetes has emerged as a major health problem worldwide, with serious health-related and socioeconomic impacts on individuals and populations alike. Diabetes results in a range of distressing symptoms; altered daily functioning (requiring attentive monitoring and treatments), changed family roles, higher costs, lost productivity and premature mortality (which are felt by households, communities, and national economies) [7]. Furthermore, the pandemic growth of diabetes is being spurred on by transitioning demographic (e.g. population aging), socioeconomic, migratory, nutritional and lifestyle patterns, and an affiliated proliferation in overweight and obese adults and children [8, 9]. The majority of this escalation will be attributable to the growth of T2D [7].

The burden of diabetes, or any other disease, can be described by its health-related impacts, social implications and economic costs (Box 1) [7]. More specifically on the health complications, patients with diabetes have an increased risk of developing several serious health problems. Consistently high BG levels can lead to serious diseases affecting the heart and blood vessels, eyes, kidneys, nerves and teeth. Additionally, patients with diabetes also have a higher risk of developing infections. In almost all high-income countries, diabetes is a leading factor of cardiovascular disease, blindness, kidney failure, tooth loss and lower limb amputation [10].

Other burdens for diabetes patients, especially in the case of T1D, are the states in which the BG levels are very low/high, e.g. when glucose homeostasis is not maintained. The fear of hypoglycemia (low BG levels) is a major concern for most T1D patients, since it can be fatal if unnoticed. It causes a lack of energy in the brain, which compromises its operations and leads to dizziness, fainting, diabetic coma, or

even death. Counteractions for this can be as simple as drinking a glass of fruit juice or eating a sandwich. However, such involvements may be delayed if hypoglycemia occurs during sleep. Hence, many families of young children must often sacrifice their sleep to monitor and prevent hypoglycemia [4].

Hyperglycemia on the other hand, destroys the cardiovascular system, leading to a lack of nutrients to different organs. This is also the case of some T2D patients. The constant increase in BG levels will eventually exhaust the pancreas, resulting in the body producing less and less insulin. This will cause even higher BG levels [11].

Box 1 - Evaluation of burden

Health-related burdens

- Health seeking and utilization (frequency and costs)
- Disease events and/or ill health
- Morbidity (physical and psychological)
- Mortality

Social implications

- Disability
- Alteration of social roles or family structure
- Caregiver or intangible burdens

Economic costs

- Direct medical expenditure
- Ancillary expenses of health seeking and care
- Indirect costs (losses in productivity)

1.1.2 Artificial Pancreas

Diabetes, T1D in particular, provides a challenging control problem. The complexities in achieving glucose homeostasis, the occurrences of non-stationary daily disturbances, the time-varying changes of BG level dynamics, the time-varying delays in BG measurements and insulin infusion and the noisy data from the sensors, all of these problems must be addressed when developing both hypoglycemia alarm systems [12] and artificial pancreas (AP) systems [13, 14, 15]. The latter system, also known as a closed-loop BG controller, may prove to be capable of automating the BG control in T1D patients and reduce the burden of the disease. The required components of an AP

are:

1. Continuous glucose monitor (CGM) (sensor)

2. A decision-making algorithm or control algorithm (controller)

3. Insulin infusion mechanism (e.g., insulin pump)

The control algorithm takes as an input the BG level measured by the CGM and outputs the insulin amount to be injected via the pump. The insulin pump can administer different types of insulin: a basal insulin, which covers the everyday absence of insulin, and a bolus insulin, which covers for meal intakes [16].

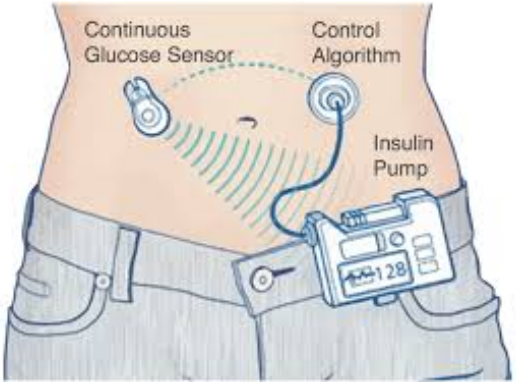

Figure 1.2: An illustration showcasing a simple AP system.

The first closed-loop controller for regulating BG concentration of T1D patients was proposed over 40 years ago [17]. The whole concept of AP systems was conceived to automate the information collection, decision making, and insulin management of a person with T1D to maintain euglycemia (the condition of having a normal concentration of glucose in the blood). In other words, a system that tries to mimic the functionality of the pancreas [15, 18, 19, 20]. Since the development of the first AP, major improvements have been made, though the system still needs development before it can be routinely used in clinical practice and everyday life [14]. One of the major limitations of the AP is the demand for an adaptive algorithm that individualizes the AP for each patient. The variability in BG concentration due to meal intake, exercise, sleep and stress (MESS) [13] are not yet modeled efficiently. Development for an adaptive AP then becomes a tedious task, which suggests moving the focus towards model-free approaches for automated insulin administration.

A model that is similar to closed-loop system is the **hybrid close-loop** [21], which fully automates the basal insulin deliveries only, whereas a standard closed-loop system covers both basal and bolus insulin rates. With this hybrid version the patient still needs to self-administer bolus insulin to account for meals. One benefit T1D patient might

find is that glucose fluctuations overnight can be controlled by the automated basal insulin rate. The system will suspend insulin delivery if the glucose sensor detects BG levels that are close to that of hypoglycemia [21].

1.2 Reinforcement Learning for Blood Glucose Regulation

Model-free methods for automating insulin administration, such as the hybrid close-loop system, is a well-suited machine learning (ML) task, or more specifically, a reinforcement learning (RL) task. RL is a ML technique that utilizes agents that learn by interacting with its environment. Algorithms that regulate the BG levels in T1D patiens has been done before [22, 23], and it has been shown that a RL algorihtm can improve BG control algorithms [23].

The purpose of this thesis is then to implement and evaluate deep Q-learning algorithms inspired by Fox and Wiens (2019) [23], to automatically regulate the BG levels in a T1D patient. A hybrid closed-loop T1D simulator will be implemented as well and represents the environment the RL agent interacts with. This system simulates the glucose-insulin dynamics of a T1D patient.

1.3 Notation

Unless otherwise is specified, the following notation will be used throughout this thesis:

- Scalars will be written in lowercase letters, e.g.,
  x, y, z
- Random variables will be written in uppercase letters, e.g., X, Y, Z
- Vectors will be written in lowercase bold letters, e.g.,
  $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$
- Matrices will be written in uppercase bold letters, e.g.,
  $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$
- Sets of numbers will be written in common blackboard bold notation, e.g., $\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$
- Other sets will be written in calligraphy notation, e.g.,
  $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$

1.4 Thesis Structure

This thesis is structured as follows. Chapter 2 covers the theory of RL. In Section 2.1 we will define basic elements of RL. We will then give a formal representation for RL in Section 2.2 in order to obtain useful Bellman equations, which are used to obtain RL algorithms. Then in Section 2.3 we will review Q-learning, that is the basic component of deep Q-learning, which is what we are going to use in our algorithms. Next some deep learning concepts will be covered in Section 2.4 in order for the reader to fully understand deep Q-learning algorithms in Section 2.5, which is of main interest in this thesis.

Chapter 3 will review our deep Q-learning algorithms, which aims at controlling the blood glucose. Section 3.1 is giving an in-depth description of the algorithms, including pseudocode, and Section 3.2 will go over the T1D simulator specifics.

In Chapter 4, the experimental setup is presented in Section 4.1. The results and analysis are given in Section 4.2.

Chapter 5 presents some concluding remarks of the results and observations.

Chapter 2

Preliminaries: Reinforcement Learning

Machine learning is a field of study that lies in the intersection between mathematics, statistics and computer science. It is an application of artificial intelligence (AI) since it provides computer systems the ability to automatically perform tasks, and improve from experience without being explicitly programmed. Machine learning algorithms focuses on constructing mathematical models based on observations, also known as training data, such that it can make predictions or decisions without human intervention or assistance. These systems are then capable of learning automatically from the training data, adjusting actions accordingly and will adapt to the current task at hand, even if the quantity of the data is massive. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. In conjunction with AI and cognitive technologies, it might become even more effective in processing large volumes of information. Machine learning is often used to solve problems that classical algorithms are unable to solve. Examples of such problems are: classification tasks, regression, dimensionality reduction, grouping or clustering of data points, autonomous vehicles and robot learning.

The foundations of machine learning [25]-[31] teach us that there are several ways for a machine to learn. The differences in the approach of the algorithms, the type of data they input and output, and the type of task is what helps us distinguish them. Machine learning techniques are often categorized into four different groups [32]:

- With **supervised learning**, the AI agent learns from a given set of labeled data consisting of input-output pairs and is trained such that it can map the correct output for each input. The outputs are often called ”labels” or ”ground truth”, and it is the algorithm’s task to apply what has been learned in the past to new data using the labels in order to predict future events. Algorithms of this category includes classification and regression.

- In contrast, **Unsupervised learning** is used when the given data is only input observations. The information used to train the agent is now ”unlabeled”. The purpose of the algorithm now is to identify hidden structures in the data, for

instance grouping or clustering of data points. The agent learns based on the presence or absence of such structures in each new piece of unlabeled data. Examples of this class would be pattern recognition and clustering.

- **Semi-supervised learning** fall somewhere between supervised and unsupervised learning. The training data consists of both labeled and unlabeled data - typically a small amount of labeled data and a large amount of unlabeled data [32, 33]. Systems that utilize this method can improve considerably in learning accuracy. Semi-supervised learning is of interest in the studies of human cognition and as a model for human learning [34].

- **Reinforcement learning** (RL) is characterized by the interactions between an agent and a given environment. The input to the system are observations regarding the environment which then produces actions the agent can take. Once some actions has been chosen, the agent will receive feedback from the environment in the form of either reward or punishment. The goal of the agent is to maximize the reward in the long run. To achieve this the agent must learn from the feedback it receives and adapt its behaviour with respect to its environment. Trial and error searching and delayed rewards are key components of RL. This allows agents to automatically develop a policy and determine the optimal behaviour in order to maximize its performance. Applications of RL algorithms are widely used in robotics, chemistry, autonomous vehicles and AI that learns to solve games or play against human opponents [35].

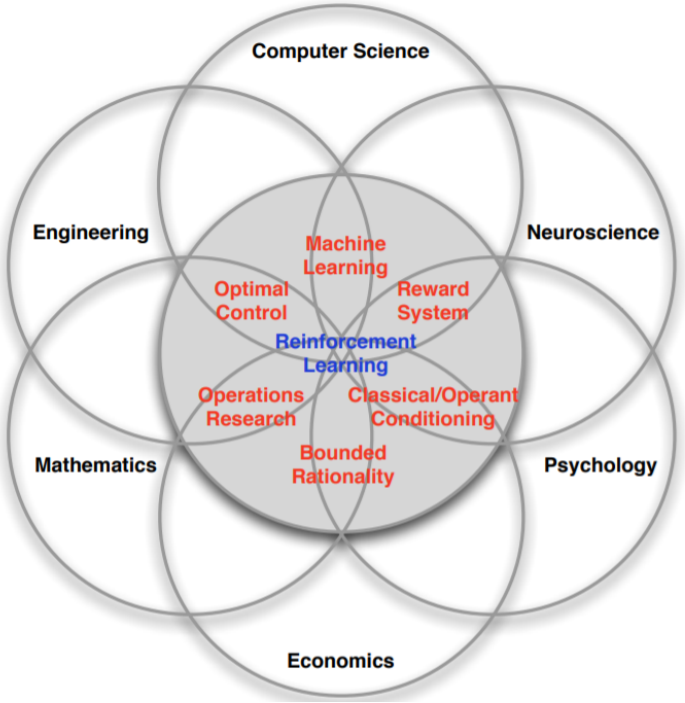

Figure 2.1: The many faces of RL [36].

The machine learning category considered in this thesis are RL techniques. Some characteristics that make RL different from other machine learning paradigms are [36]:

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d[1] data)
- Agent's actions affect the subsequent data it receives

Since RL is tailored to work well with sequential decision-making tasks, a closed-loop BG regulation problem seems like a great fit. No training data is required nor any model of the environment. In addition, RL can be applied on real data. In other words, interact with dynamical systems represented by mathematical models [37]. A RL agent can directly interact with an AP environment, get observations and learn how to control BG in a closed-loop manner. This is ideal for AP systems since there is a need to continuously observe the patients BG level and then determine the amount and time for insulin delivery. This takes the focus of this chapter towards existing RL techniques with extra weight on critic-only algorithms.

As opposed to supervised learning, where we have learning with a teacher, in RL we have a **critic** [38]. This type of learning differs from a teacher in the sense that it does not tell us what to do, but rather how well we have been doing in the past. The critic never informs the agent ahead of time, but when it does it receives a delayed sparse feedback. This introduces the **credit assignment problem**, or the **blame attribution problem** [38, 39]. In short, this is the problem of determining which action was responsible for a reward or punishment. This is one of the reasons why RL can be a difficult task, and we will review more in the sections to come.

The early years of RL had a long and rich history, which can be split into two threads. One thread concerned learning by trial and error and started in the psychology of animal learning. This thread had some of the earliest work in AI and led to the revival of RL in the early 1980s, which was mostly done by Harry Klopf [40, 41, 42]. Klopf's work influenced Sutton and Barto [37] which sparked their ideal of *maximizing the rewards* obtained once an action is taken. The second thread concerned around optimal control and its solutions using value functions and dynamic programming. This thread did not involve any learning for the most part and has its roots from Richard Bellman's [43, 44] concepts in the mid-1950s. Since then, the relationships between optimal control and dynamic programming have been extensively developed by many, particularly by Bertsekas and Tsitsiklis [45]. Their goal was to *minimize the costs* generated after taking an action. In the late 1980s the two independent threads intertwined together

[1] An abbreviation for independent and identically distributed.

with a third thread, which was involved with temporal-difference (TD) methods, that produced the modern field of RL as presented in the Sutton and Barto book [37]. In Sutton's Ph.D. [47], a method for using TD learning combined with trial-and-error learning was developed later known as *actor-critic architecture*. TD learning refers to the learning methods for value function evaluation discovered by Sutton in 1988 [48]. The optimal control and TD threads were fully brought together in 1989 with Chris Watkin's [46] development of Q-learning (QL). This finally enabled both the extension and integration of all prior research in all three threads of RL.

In this chapter we will delve into some of the foundations of RL and deep learning (DL), which is necessary to fully grasp the state-of-the-art deep reinforcement learning (DRL) algorithms. First we will introduce the definition of the elements of RL. Then a formulation of a Markov decision process (MDP) will be made in conjunction with the RL framework. Next we will derive the Bellman equations for the value function, which are central conceptions in RL that are often used to obtain certain RL methods, followed by definitions for optimal policies and optimal value functions.

In the next sections we will go through RL methods, which has been split into three sections. Each of these sections represents a class of RL techniques: critic-only, actor-only and actor-critic. This is then accompanied by a general overview of DL and well-known methods within the category. We then move on to the sections regarding DRL techniques. Although some of these algorithms are not relevant for the purpose of this thesis, they were still included to motivate and to show relevant knowledge for the field of DRL.

2.1 Reinforcement Learning Foundations

Previously reviewed, the main components of RL are the **agent** and the **environment**. The agent is the learner and decision-maker and the environment is the world the agent lives in and interacts with, encompassing everything outside the agent. At every time step of the interaction, the agent observes the state of the environment and determines an action to perform. For every action the agent takes, the environments responds by presenting new situations (states) and rewards to the agent. The goal of the agent is to maximize its cumulative reward or **return**. Discovering by itself, the agent must explore and learn the actions that yield the best rewards and reinforce its behaviour, either by trying different actions or by repeating them.

In the subsections to come, the basic RL terminologies will be defined. All interactions between the agent and the environment can be assumed to occur during discrete time steps, i.e., $t = 0, 1, 2, \ldots$. Additionally, since most states from real-life scenario experiments are restricted by natural boundaries, their respective state spaces are of finite size. In the upcoming subsections, continuous state spaces are allowed to not restrict the theory.

2.1.1 States

A state is defined as the information that describes an environment. This information may either be described by words or consist of values or numbers. For instance, in a game of coin tossing, the states could be represented as "heads" or "tails", or as 0 or 1. More generally, assigned values can be real-valued vectors, matrices or tensors. We can denote the state of the environment at time t by S_t, and its realizations by s_t.

In practice we say that the states are the observations that the agent receives from the environment. More specifically, the environment has a state S_t^E and the agent can observe a state from the environment S_t^A. There are two scenarios for the agent when observing [49]:

1. The agent directly observes the environment ($S_t^A = S_t^E$), meaning the complete description of the environment. We then say that the environment is **fully observed**.

2. The agent indirectly observes the environment ($S_t^A \neq S_t^E$), meaning a partial description of the environment. We then say that the environment is **partially observed**.

A good example of a fully observable environment would be a game of chess. The states are represented by the positions of the pieces on the board. The state $\mathbf{S}_t$ is then a 32×2 matrix that represent the 32 chess pieces in a 2D coordinate system, where the realized chess board is $\mathbf{s}_t$. However, in a poker game, the agent has access to all the cards on the table, but has no information about the next card from the deck. This means that only the cards on the table represent the state and we have a partially observable environment.

A **state space** is defined as the set of all possible states of an environment, denoted $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$. Generally one can define the state spaces $\mathcal{S}_t$ as the range of available states of an environment at time t. In the previous two examples, the state spaces are discrete meaning they consist of a finite set of states. For the application of BG regulation, the state space could be either discrete or continuous. Every state S_t could be represented by the BG level at time t, which is any non-negative real number. Additionally, the BG level and the insulin concentration could both be used to represent the state $\mathbf{S}_t$. In either case, the state space $\mathcal{S}_t$ should be the set $\mathbb{R}_+$[2], meaning that every state is observed from an infinite length space [50].

2.1.2 Actions

Similarly to state spaces, an **action space** $\mathcal{A}$ is the set of all valid actions in a given environment [49], i.e., $\mathcal{A} = \{a_1, a_2, \ldots a_{|\mathcal{A}|}\}$. Action spaces can either be discrete or continuous. With a discrete action space, the agent decides upon an action from a finite set. With a continuous action space, the actions are represented as real-valued vectors [51]. Environments like CartPole, Atari and Go have discrete action spaces, whilst an

[2]The set of non-negative real numbers.

environment from robotics could have continuous action spaces [49]. Additionally, in the case of BG regulation, the actions could be defined as the insulin amount to be injected. In that case the action space could be either discrete or continuous.

If the environment at time t is in the state s_t, realized by S_t, then the action space at time t is denoted by $\mathcal{A}(s_t)$. The current state s_t determines the available actions in $\mathcal{A}$. In the CartPole [52] environment, the actions available at all times is either moving the cart to the left or to the right. In either case, the action a_t taken by the agent will have an impact on the environment, and so the next state S_{t+1} is obtained. In the BG regulation problem, after an injection has been done there is a delayed response in that the action has both impacted the next state and the following one. This is also the case for CartPole. The goal of the agent is to pick actions that leads into preferred states, which is difficult because it may not know what actions leads into which states.

Note that in some environments, not all actions can be applied in every state. For simplicity, we will assume that [53]

$$\mathcal{A}(s_t) = \mathcal{A}, \quad \forall s_t \in \mathcal{S}.$$

The sequence of actions from the initial state to the end state (terminal state) is often defined as an **episode**, or a **trial**.

2.1.3 Policy

A policy is the agent's behaviour function at a given time and is one of the core elements of RL. It is a **mapping** from states to actions which dictates what action the agent selects in its current state. In a sense, the policy is the brain of the agent, defined around the goal of maximizing a long-term reward. Another objective of this brain is to manage the trade-off between exploration and exploitation. The former describes the agent exploring different states, seeking to find potential states that provide maximum reward. The latter portrays the agent maximizing its immediate gain, i.e., staying in its preferred state [50].

There are two types of policies [49]:

1. **Deterministic** policies, usually denoted by μ
2. **Stochastic** policies, usually denoted by π

A mapping from the state s_t to an action $a_t \in \mathcal{A}(s_t)$ can then be written as

$$a_t = \mu(s_t)$$
$$a_t \sim \pi(a_t|s_t)$$

In general, a deterministic policy μ is formally defined as a function $\mu : \mathcal{S} \rightarrow \mathcal{A}$ [53]. The probability of selecting an action $a_t \in \mathcal{A}(s_t)$ is 1 and the probability for selecting any other is zero. Likewise, a stochastic policy π is defined as $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. It can also be expressed as

$$\pi(a_t|s_t) = P(A_t = a_t|S_t = s_t). \tag{2.1}$$

This is the case for discrete action spaces and is only valid such that for each state $s_t \in \mathcal{S}$, it holds that

$$\pi(a_t|s_t) \geq 0 \tag{2.2}$$

and

$$\sum_{a_t \in \mathcal{A}(s_t)} \pi(a_t|s_t) = 1 \tag{2.3}$$

In the case of continuous action spaces, the π distribution becomes

$$\pi(a_t|s_t) = P(A_t \in \mathcal{A}|S_t = s_t) = \int_{\mathcal{A}} \pi(a|s_t)da. \tag{2.4}$$

If $\mathcal{A} = \mathcal{A}(s_t)$, the previous equation must integrate to 1 [50]. Two common stochastic policies (often used in DRL) are **categorical policies** and **diagonal Gaussian policies** [49]. The former policies are used in discrete action spaces, while the latter are used in continuous ones.

Notice that the deterministic definition is included in the stochastic one, which happens when the current state s_t personally influences the choice of a single action a_t. In this scenario we use a deterministic policy $\pi(a_t|s_t) = 1$, for a specific action a_t and zero otherwise [50].

Later in this chapter, we will review deep reinforcement learning (DRL), which uses **parameterized policies**. These policies output computable functions that depend on a set of parameters, e.g., weights and biases from a neural network, which can be adjusted to change the behaviour of an RL agent according to some optimization algorithm [49]. Such parameters are often denoted by θ or ϕ, as a subscript on the policy to emphasize the link:

$$a_t = \mu_\theta(s_t)$$
$$a_t \sim \pi_\phi(a_t|s_t)$$

2.1.4 Rewards

When a RL agent interacts with its environment it observes states and performs an action with respect to some policy. Every action the agent selects results in a response from the environment consisting of a new state and a reward. This happens every time step of the interaction and we denote the reward by $r_t \in \mathcal{R} \subset \mathbb{R}$. This is a scalar feedback indicating how good the agent did at time step t. It is the output of the reward function R, which can depend on the current state, the action performed and the next state [49]:

$$r_t = R(s_t, a_t, s_{t+1}). \tag{2.5}$$

In many scenarios, the reward function only depends on the state-action pair, $r_t = R(s_t, a_t)$, or even simpler, the current state, $r_t = R(s_t)$. Knowing this, the reward function can have 3 definitions [53]: one for the state reward $R : \mathcal{S} \to \mathcal{R}$, the state-action pair reward $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, or the transition between states $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$. These definitions are all interchangeable though the last one is well suited for model-free algorithms. This is because we usually need both the start state and the end state to influence the reward [53]. Though, this is the least restricted definition of the bunch, reward functions of the form $r_t = R(s_t, a_t)$ will be mostly assumed as the standard throughout this chapter and beyond. In figure 2.2, one can see a basic illustration of a RL setup.

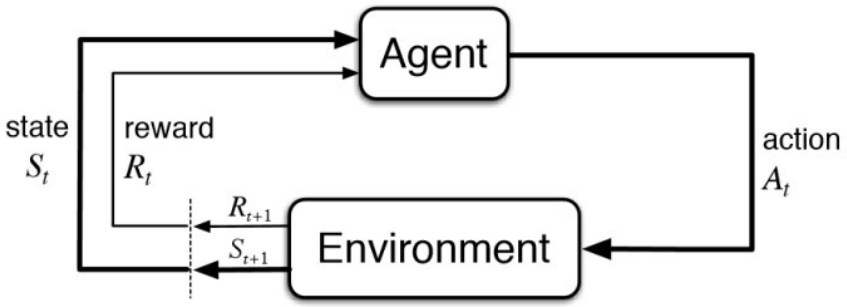

Figure 2.2: A RL agent-environment interaction. The agent chooses an action A_t based on current state S_t from the environment and on a policy π. The environment then outputs the next state S_{t+1} and the corresponding reward R_{t+1}, where the latter is used to update π [54].

The reward function defines the goal in a RL problem. The objective of all RL agents is to maximize the total reward in the long run [37]. A reward can be the added score in a game, winning a game or successfully completing a sub-task. Rewards may imply positive values only, since these things go hand in hand. However, negative rewards can be interpreted as punishment instead (similarly to a cost function), and the learning goal now is to minimize this punishment. As an example, combining the idea of reward and punishment, one could assign positive values to good transitions and negative values to bad transitions. In a game of chess, assign a positive reward when the agent wins the game, a negative reward when the agent loses the game and a zero reward

when the game is a draw.

When trying to accomplish a direct goal, like the one described in the previous paragraph, it is of paramount importance that the reward is reflecting only that goal and no other sub-goals. If the latter is the case, the agent could try to achieve these sub-goals, without advancing in the main one. Learning sub-goals is sometimes desirable, where we assign non-zero reward to non-goals.

Another remark is the emphasize on the relationship between the reward function and the policy. It is crucial that the policy does not directly influence the reward function. There should only be an indirect change in the reward mapping by the agent's action selection. Contrariwise, having rewards influence the policy can help achieve the agent's goal.

When dealing with BG regulation, the agent's goal is to attain normoglycemia. The reward assignment could be, for example, negative values assigned to transitions that yield in lower or higher BG level than the desired threshold, ultimately weakening the patient's condition. If the ending states receives a negative reward, the agent should strive to choose a better insulin amount for the next day [50]. For the task at hand, one could instead of maximizing the long-run cumulative reward, minimize the long-run total of negative rewards.

2.1.5 Trajectories

When one looks at what a RL agent has experienced, in terms of state-actions pairs, then we refer to the sequences called trajectories, often denoted by τ. Generally trajectories take the form $\tau = (s_0, a_0, s_1, a_1, \ldots, s_{N-1}, a_{N-1}, s_N)$, where $N \in \mathbb{N}^{+3}$. Such a sequence can be taken from a single episode, a single part of a continuous task, or from any other scenario [55]. The first state of the environment, $s_0 \in \mathcal{S}$, is randomly sampled from the **start-state distribution** [49], often denoted ρ_0:

$$s_0 \sim \rho_0(s_0)$$

The state transitions only depend on the most recent action a_t obtain from the agent's policy. They can either be deterministic,

$$s_{t+1} = f(s_t, a_t), \tag{2.6}$$

or stochastic

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t) \tag{2.7}$$

A more elegant definition of trajectories is

[3]The set of positive integers without zero.

$$\tau = \{s_t, a_t\}_{t\in[t_0,t_H]} \quad s_t \in \mathcal{S}, a_t \in \mathcal{A}, \tag{2.8}$$

where t_0 is the initial time and $t_H > t_0$ is the time associated with the horizon H. A horizon is a future point relative to a time step t. In other words; a trajectory τ is the path of the agent through $\mathcal{S}$ and $\mathcal{A}$ up until the horizon H [55]. In general, the goal of a RL agent is to maximize the long-time reward, which happens to be

$$\max_{\tau} R(\tau),$$

from time t up until $t + H$. In the next sub-section, this quantity known as the return will be described in more detail.

2.1.6 Returns

We defined the reward function R to only depend on the state-action pairs:

$$r_t = R(s_t, a_t).$$

The RL goal is to maximize some perception of cumulative reward over a trajectory. Cumulative reward is referred to as a **return** and is denoted by

$$G_t = g(r_t + r_{t+1} + r_{t+2} + \dots) = R(\tau), \tag{2.9}$$

where g is a function considered for two scenarios: the discounted return and the average return. Firstly, let us consider the most simple case, that is the **finite-horizon undiscounted return**:

$$G_t \doteq R(\tau) = \sum_{\ell=0}^{T} r_{t+\ell}, \tag{2.10}$$

where T is the final time step (horizon). This is also known as the *episodic model* [38]. For the case of continuous problems, the **infinite-horizon discounted return** will be used:

$$G_t \doteq R(\tau) = \sum_{\ell=0}^{\infty} \gamma^{\ell} R_{t+\ell}, \tag{2.11}$$

where $\gamma \in (0, 1)$ is the discount rate. This sum will only converge if the sequence of rewards $\{r_{t+\ell}\}_{\ell=0}^{\infty}$ is bounded and $\gamma < 1$ [50, 38]. Therefore, it might be better to define γ to be in $[0, 1)$. If $\gamma = 0$, only the immediate reward is given since $G_t = r_t$ and all the other terms has perished. When $T < \infty$ we obtain the episodic version of the above equation:

$$G_t \doteq R(\tau) = \sum_{\ell=0}^{T} \gamma^{\ell} R_{t+\ell}, \tag{2.12}$$

i.e., the **finite-horizon discounted return**. If $\gamma = 1$, all the rewards in the sum gets an equal amount of importance. So, the larger γ is, the more attentive the agent will be on

maximizing the rewards over the trajectory.

Lastly, the **finite-horizon average return** is defined as

$$G_t \doteq R(\tau) = \frac{1}{T}\sum_{k=0}^{T} R_{t+\ell}, \tag{2.13}$$

and the **infinite-horizon average return**,

$$G_t \doteq \lim_{T\to\infty} \frac{1}{T}\sum_{\ell=0}^{T} R_{t+\ell}. \tag{2.14}$$

Again, this sum will converge as long as $\{r_{t+\ell}\}_{\ell=0}^{\infty}$ is bounded. The average return also provides an equal split of importance to each reward the agent receives.

2.1.7 The Value Function

While a reward function indicates what is good in an immediate sense, a value function specifies what is good in the long run (the return) [37]. More specifically, a value function is a prediction of the total future reward [36]:

$$V_\pi(s_t) = E[r_t + r_{t+1} + \ldots r_T | S_t = s_t] = E\left[\sum_{\ell=0}^{T} r_{t+\ell} | s_t\right]. \tag{2.15}$$

It is often referred to as a state-value function. The value of the state s_t is a goodness/badness evaluation. It gives the agent an expectation of cumulative reward it can accumulate starting from that state. Similarly to the finite-horizon model, the infinite version is given by

$$V_\pi(s_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | S_t = s_t] = E\left[\sum_{\ell=0}^{\infty} < \gamma^\ell r_{t+\ell} | s_t\right]. \tag{2.16}$$

A more in-depth review about value functions will be done in the next section, as some statistical support is needed first.

Another remark: while the line between the two main definitions of the return is quite sharp in RL formalism, in practice, i.e., deep RL, tends to blur this line. This means that the algorithms optimize the undiscounted return, but use the discount factor in estimating value functions [49].

2.1.8 Agent and Environment

Sutton and Barto (2018) [37] define the **agent** as the decision-maker and the learner. The **environment** is what the agent interacts with, comprising everything outside the agent's control. More specifically, the environment includes the reward function since the agent should not dictate over it, as mentioned previously. In Figure 2.2, we can see that there is a boundary between the agent and the environment.

In our problem, the insulin pump is placed within the environment. This is because there might be differences between the dose selected by the agent and the actual amount of insulin administered. Therefore, the agent only has the controller included. The CGM, insulin pump and the body of the patient are part of environment [50]. This task and setup is more intricately explained in sections 3.1 and 3.2.

As mentioned earlier, RL is like trial-and-error learning. The agent should discover a good policy from its experiences of the environment, without losing too much reward along the way. Exploration yields more information about the environment, while exploitation exploits known information to maximize reward. It is usually important to explore as well as exploit.

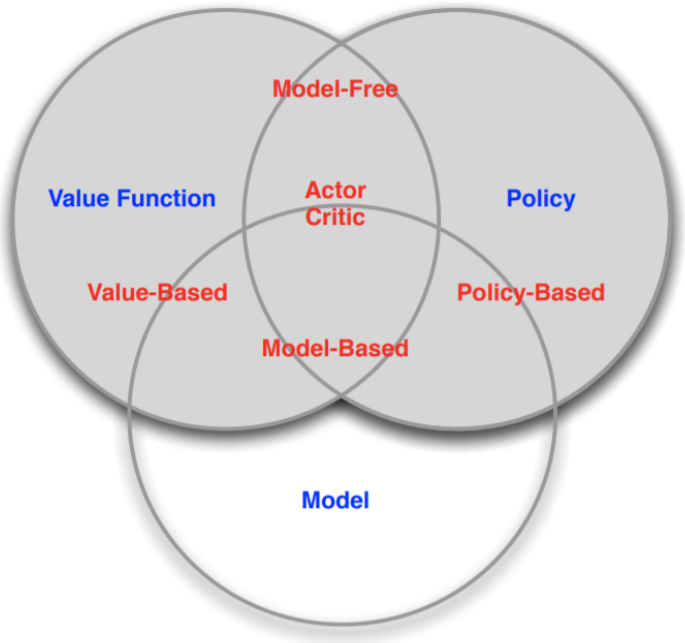

Figure 2.3: A RL agent taxonomy [36].

2.1.9 Model

For some RL systems there exists a **model**, which is the agent's representation of the environment. In practice this is a prediction that tries to mimic the behaviour of the environment, or more generally, that allows conjectures to be made about how the environment will act [37]. Models are used for planning, in the sense that they predict what the environment will do next. For example, given a current state and action, the model might predict the resultant next state and corresponding reward. RL tasks that require models and planning are called **model-based** methods. The opposite, **model-free** methods, do not use models nor planning and are explicitly trial-and-error learners [37]. Since this thesis focuses on deep Q-learning techniques, model-based methods will not be reviewed.

2.2 Markov Chain Formalism

2.2.1 Markov Decision Processes

Markov decision processes (MDP) are an intuitive and fundamental formalism for RL [37, 56]. Up until this point, we have only discussed the agent's environment in an informal manner. MDPs are the standard mathematical formalism for this setting [49].

A MDP (or Markov chain) is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, R, P \rangle$, where

- $\mathcal{S}$ is the set of all possible states,
- $\mathcal{A}$ is the set of all possible actions,
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function,
- $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition probability function, with $P(s'|s, a)$ being the probability of transitioning into state s' given that you start in state s and take action a [49].

More explicitly, the **state-transition probabilities** can be denoted

$$P(s'|s, a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} P(s', r|s, a) \tag{2.17}$$

where the sum expression is the *dynamics* of the MDP [37]

$$P(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a), \tag{2.18}$$

where $s', s \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$. It follows that equation 2.18 must satisfy

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r|s, a) = 1, \tag{2.19}$$

$\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$.

2.2.2 Markov Property

MDPs are discrete-time stochastic processes that satisfy the **Markov property** [57]. This property states that transitions only depend on the most recent state and action, and no prior history [49]. It is therefore often interpreted as a memoryless property. equation 2.18 for example, says that the environment's response at time step t depends only on the state and action at the previous time step.

A state S_t has the Markov property only if

$$P(S_{t+1}|S_t, A_t) \doteq P(S_{t+1}|S_0, A_0 \ldots, S_{t-1}, A_{t-1}, S_t, A_t), \tag{2.20}$$

like the state-transition probabilities in equation 2.17. This equation implies that [58]:

- The state S_t captures all relevant information from the history,
- Once this state is known, all prior history can be forgotten,
- The state is a sufficient statistic of the future.

The BG regulation problem is a difficult context. Once an insulin amount has been administered, its effect on the patient will be visible for several hours after injection. In addition, the upcoming BG level will most likely follow a trend from the previous BG measures. By this reasoning, it will not be adequate to only use the most recent BG level in S_t, since it actually depends on a trajectory of BG measures. This problem along with a state-space definition will be covered in greater detail in section 3.1.1.

The expected rewards for the state-action pairs is calculated by

$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} P(s', r|s, a), \tag{2.21}$$

where $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We can also compute the expected rewards for the state-action-next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ [37]

$$r(s, a, s') \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r\in\mathcal{R}} r\frac{P(s', r|s, a)}{P(s'|s, a)}. \tag{2.22}$$

These equations in the continuous case easily be obtained by replacing the sums with integrals:

$$\rho(s, a) \doteq \int_{\mathcal{R}} \int_{\mathcal{S}} rP(s', r|s, a)ds'dr \tag{2.23}$$

and

$$\rho(s, a, s') \doteq \int_{\mathcal{R}} r\frac{P(s', r|s, a)}{P(s'|s, a)}dr. \tag{2.24}$$

2.2.3 Value Functions and Bellman Equations

We reviewed the concept of value in the previous section, often referred to as the **state-value function** $V_\pi(s)$, which reflects the quality of states. Using MDPs, we can define the state-value function formally by [37]

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s], \tag{2.25}$$

where G_t is the return, τ is a trajectory of states and actions, R is the reward function and $S_0 = s$ is the initial state. This value function is also known as the **on-policy value function** because it gives the expected return given you start in state s, acting accordingly to some policy π. Another useful quantity is the **action-value function**, which gives a value to the state-action pairs and is defined as

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a], \tag{2.26}$$

where $A_0 = a$ is an arbitrary action. Similarly to the value function, this equation is also known as the **on-policy action-value function**.

The value function in equation 2.25 has an interesting recursive characteristic. For any policy π and any state s, the following condition holds between the value of s and the value of its succeeding states [37]:

$$\begin{aligned} V_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_{a\in\mathcal{A}(s)} \pi(a|s)\mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} P(s', r|s, a) \left[r(s, a) + \gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] \end{aligned}$$

Notice that the second term inside the square bracket is the value function, but for the next state s'. We can therefore rewrite the above as

$$V_\pi(s) = \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} P(s', r|s, a)[r(s, a) + \gamma V_\pi(s')]. \tag{2.27}$$

Note that we are using a discounted return with discount factor γ, and we have merged the two sums over s' and r to simplify the notation. This equation is known as the **Bellman equation** for V_π. The idea behind this equation is that it expresses a relationship between the value of a state and the values of its successor states [37]. The value of your starting state is the expected reward for being there, plus the value of wherever you go next [49]. For each triple (a, s', r) we compute its probability $\pi(a, s)P(s', r|s, a)$, then weight the expression inside the square bracket by that probability, and then sum over all possibilities to achieve an expected value. Similarly in the continuous case, the Bellman equation for the value function becomes

$$V_\pi(s) = \int_{\mathcal{A}(s)} \pi(a|s) \int_{\mathcal{S}} \int_{\mathcal{R}} P(s', r|s, a)[r(s, a) + \gamma V_\pi(s')]drds'da. \tag{2.28}$$

A similar Bellman equation can be derived for the action-value function. It is given by

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r|s, a)[r(s,a) + \gamma V_\pi(s')], \tag{2.29}$$

for the discrete case and

$$Q_\pi(s,a) = \int_{\mathcal{S}} \int_{\mathcal{R}} P(s', r|s, a)[r(s,a) + \gamma V_\pi(s')] dr ds', \tag{2.30}$$

for the continuous case. Both the Bellman equations for V_π and Q_π have a stationary property since they do not depend on time t anymore. We can also see from equation 2.27 that the relationship between V_π and Q_π can be expressed as

$$V_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) Q_\pi(s,a), \tag{2.31}$$

or in the continuous case

$$V_\pi(s) = \int_{\mathcal{A}(s)} \pi(a|s) Q_\pi(s,a) da. \tag{2.32}$$

A more compact version of the Bellman equation for the value functions may be written as

$$V_\pi(s) \doteq \mathbb{E}_{\pi, s' \sim \mathcal{P}}[r(s,a) + \gamma V_\pi(s')] \tag{2.33}$$
$$Q_\pi(s,a) \doteq \mathbb{E}_{s' \sim \mathcal{P}}[r(s,a) + \gamma \mathbb{E}_\pi[Q_\pi(s', a')]], \tag{2.34}$$

where $s' \sim \mathcal{P} = P(s'|s,a)$ is the successor state sampled from the state transition. Another way to write the connection between the value functions is

$$V_\pi(s) = \mathbb{E}_\pi[Q_\pi(s,a)]. \tag{2.35}$$

2.2.4 Optimal Policy

The goal of any given MDP is to find a *best* policy, that receives the highest amount of reward possible in the long-run. A policy π is defined to better than or equal to another policy π' if its expected return is greater than or equal to that of π' for all states [37]. In equation form this is

$$\pi \geq \pi', \quad \text{if and only if } V_\pi(s) \geq V_{\pi'}(s), \ \forall s \in \mathcal{S}. \tag{2.36}$$

The **optimal policy** is therefore the policy that is better than or equal to all other policies, and we denote this by π^*. Note that there may be more than one optimal policy.

We can now define **optimal state-value function** as

$$V^*(s) \doteq \max_\pi V_\pi(s) = \max_\pi \mathbb{E}_{\tau \sim \pi}[R(\tau)|S_0 = s], \quad \forall s \in \mathcal{S}. \tag{2.37}$$

Optimal policies also gives us the **optimal action-value function**, defined as

$$Q^*(s,a) \doteq \max_\pi Q_\pi(s,a) = \max_\pi E_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a], \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{2.38}$$

Using the identity from equation 2.35 we can obtain a similar connection for the optimal value functions:

$$V^*(s) = \max_a Q^*(s,a). \tag{2.39}$$

The Bellman equations for the optimal value functions can be obtained using the same derivation as in the last subsection. They are regularly called the **Bellman optimality equations** [37]. The Bellman equation for V^* is given by

$$V^*(s) = \max_{a\in\mathcal{A}(s)} Q^*(s,a) = \max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S},r\in\mathcal{R}} P(s',r|s,a)[r(s,a) + \gamma V^*(s')], \tag{2.40}$$

and in the continuous case

$$V^*(s) = \max_{a\in\mathcal{A}(s)} \int_{\mathcal{S}}\int_{\mathcal{R}} P(s',r|s,a)[r(s,a) + \gamma V^*(s')]drds'. \tag{2.41}$$

Once again we are using discounted return for all the value functions.

To arrive at an equivalent expression for Q^*, it would be beneficial to write Q^* in terms of V^*. Deriving from equation 2.38 we get:

$$\begin{aligned} Q^*(s,a) &= \max_\pi Q_\pi(s,a) \\ &= \max_\pi \mathbb{E}[r(s,a) + \gamma V_\pi(s')|S_t = s, A_t = a] \end{aligned}$$

Moving the $\max$ operator inside the expectation only affects the state-value function, which yields

$$Q^*(s,a) = \mathbb{E}[r(s,a) + \gamma V^*(s')|S_t = s, A_t = a]. \tag{2.42}$$

Substituting the expression for V^* in the above equation gives us

$$Q^*(s,a) = \mathbb{E}[r(s,a) + \gamma \max_{a'} Q^*(s',a')|S_t = s, A_t = a].$$

Similarly to the first Bellman equation derivation, the above equation can be rewritten as

$$Q^*(s,a) = \sum_{s'\in\mathcal{S},r\in\mathcal{R}} P(s',r|s,a)[r(s,a) + \gamma \max_{a'} Q^*(s',a')], \tag{2.43}$$

which is the Bellman equation for Q^*. The continuous version of this equation is correspondingly

$$Q^*(s,a) = \int_{\mathcal{S}}\int_{\mathcal{R}} P(s',r|s,a)[r(s,a) + \gamma Q^*(s',a')]drds'. \tag{2.44}$$

A more compact way to express these Bellman optimality equations is

$$V^*(s) \doteq \max_{a \in \mathcal{A}(s)} \mathbb{E}_{s' \sim \mathcal{P}}[r(s,a) + \gamma V^*(s')] \tag{2.45}$$

$$Q^*(s,a) \doteq \mathbb{E}_{s' \sim \mathcal{P}}[r(s,a) + \gamma \max_{a' \in \mathcal{A}(s)} Q^*(s',a')], \tag{2.46}$$

where where $s' \sim \mathcal{P} = P(s'|s,a)$ is the successor state sampled from the state transition.

Now that we have an equation for the optimal state-action values, we can then define our optimal policy π^* as taking the optimal action a^* starting in state s [38]:

$$\pi^*(a|s): \quad \text{Choose } a^*(s) = \arg\max_a Q^*(s,a). \tag{2.47}$$

Note that there may be multiple actions that maximize $Q^*(s,a)$, in which all of them are considered optimal. In this scenario, π^* may randomly select any of them [49]. By having the Q^* values, we perform a so-called **greedy search**, in which we choose the optimal sequence at each local step that maximizes the cumulative reward [38].

From here on out, it will be common to refer action-state values $Q(s,a)$ as Q-values.

2.2.5 The Advantage Function

Another quantity that is useful in RL is the **advantage function**. It is the difference between the Q-values for a given state-action pair and the state-value function:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.48}$$

This function describes how much better it is to take a specific action a in state s over any other action according to a policy $\pi(a|s)$ [49]. Another way to perceive the advantage is that it tells about the extra amount of reward that could have been obtained by the agent by taking the action a over any other action from its policy [59].

2.3 Reinforcement Learning Methods

With the now formalized RL problem using MDPs we can finally review methods for solving this problem. Sutton and Barto (2018) [37] states that "any method that is well suited to solving such problems we consider to be a **reinforcement learning method**". They also state that RL methods specify how the agent's policy has changed as a result of its experience. In this section some common RL methods will be presented.

2.3.1 Dynamic Programming

The main idea behind **dynamic programming** (DP) is the use of value functions in order to obtain good policies [37]. This method require a complete and accurate model of the environment which makes DP algorithms of limited use in RL. Other assumptions in DP is a finite MDP and that we have access to its dynamics given by the probabilities $P(s', r|s, a)$ [37].

Some common DP algorithms include [56, 60]:

- **Policy evaluation (prediction)**: Iterative computation of the value function for a given policy,
- **Policy improvement**: Computation of an improved policy for a given value function,
- **Policy iteration**: Iterates between the policy evaluation step and policy improvement step,
- **Value iteration**: Combines policy evaluation and policy improvement into one update rule.

The policy evaluation algorithm, also known as *the prediction problem*, is all about finding the value function V_π of a fixed policy π. From the Bellman equation in equation 2.27, we transform it into an update rule and obtain successive state-value approximations [37]:

$$V_{\pi,k+1}(s) = \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S},r\in\mathcal{R}} P(s', r|s, a)[r(s, a) + \gamma V_{\pi,k}(s')]. \tag{2.49}$$

Now the next is to improve the policy. Recall the Bellman equation for Q-values in equation 2.29. We can improve the current policy by using a *greedy* policy π^* instead. Also recall $\pi^* \geq \pi$ if and only if $V_{\pi^*}(s) \geq V_\pi(s), \forall s \in \mathcal{S}$. By combining equation 2.29 and equation 2.47 we obtain the optimal policy

$$\pi^*(s) = \arg\max_a \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} P(s', r|s, a)[r(s, a) + \gamma V_\pi(s')]. \tag{2.50}$$

If the new greedy policy π^* is as good as the old policy π, but not better, i.e., $\pi^* = \pi$, then from the above equation we have that [37, 56]

$$V_{\pi^*}(s) = \max_a \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} P(s', r|s, a)[r(s, a) + \gamma V_{\pi^*}(s')]. \tag{2.51}$$

In short, policy iteration starts with an arbitrary initial policy π_0. Then a sequence of iterations follows in which the current policy is evaluated according to equation 2.49, where we compute $V_{\pi,k}$, and then improved upon according to equation 2.50, where we compute π_k. After k steps, the policy iteration yields in a policy that is not worse than the greedy policy π^* [56, 60].

Value iteration is a DP algorithm where we take a truncated version of the policy evaluation step and combine it with the policy improvement step [56]:

$$V_{k+1}(s) = \max_a \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r|s, a)[r(s, a) + \gamma V_k(s')]. \tag{2.52}$$

With this algorithm we start with a arbitrary state-value V_0 and iterate towards V_k. This algorithm, as well policy iteration, can converge towards V^* [37, 56, 60].

As mentioned earlier, DP algorithms assumes finite MDPs as well as the access to the state transition probabilities and MDP dynamics. In practice, these requirements are rarely met and makes these methods not feasible for our case. This issue is solved when using classes of RL methods including: critic-only methods, actor-only methods and actor-critic methods. The focus of this thesis though is with **critic-only** methods, more specifically **Q-learning**, where we maximize an approximate value function to determine the policy.

2.3.2 Q-learning

With critic-only methods, or direct methods, we aim to approximate the optimal Q-value function Q^* [60], in order to select the best action and build a policy, without knowing the state transitions $P(s'|s, a)$ and environment dynamics $P(s', r|s, a)$. Critic-only methods is part of a class of RL methods known as **temporal-difference learning**, or TD learning [37]. These methods learn from experience: directly from interaction with the environment and does not require full models. They are therefore model-free RL methods.

One of the most basic and popular methods to estimate the Q-values is the **Q-learning** algorithm by Watkins and Dayan (1992) [61]. Assuming finite state and action spaces and discounted return, the Q-learning algorithm is given by

$$Q_{i+1}(s_t, a_t) = Q_i(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q_i(s_{t+1}, a) - Q_i(s_t, a_t)], \tag{2.53}$$

where $(s_t, a_t) \in \mathcal{S} \times \mathcal{A},\ \forall t, \gamma \in (0, 1)$ is the discount rate and $\alpha \in (0, 1]$ is the learning rate. The difference between the current Q-value estimate $Q_i(s_t, a_t)$ and the received reward r_{t+1} plus the discounted Q-value estimate for the next state $Q_i(s_{t+1}, a_t)$ is known as the **TD error** [37, 38]. We can define this error by

$$\delta(Q_i) = r_{t+1} + \gamma \max_a Q_i(s_{t+1}, a) - Q_i(s_t, a_t), \tag{2.54}$$

and use it in equation 2.53 to get

$$Q_{i+1}(s_t, a_t) = Q_i(s_t, a_t) + \alpha\delta(Q_i). \tag{2.55}$$

equation 2.55 is trying to reduce this TD error over some iterations. The agent goes from state s_t to s_{t+1} using action a_t and receiving a reward r_{t+1} [60]. We then update from the old Q-values Q_i to the new Q-values Q_{i+1}.

The Q-values are updated independently of any policy, and in such a way that the sequence of Q-values will converge to an optimal action-value function Q^* [61]. It is therefore considered as an **off-policy** RL method, where the value of the best action is used without sampling from a policy [38]. The fact that Q-learning allows for arbitrary sampling strategies for generating state-action pairs is a major attraction [60]. A common choice for action sampling is using the ε-greedy action selection, defined as

$$a_t = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s_t, a) & \text{with probability } 1 - \varepsilon, \\ a \sim \mathcal{A} & \text{otherwise,} \end{cases} \tag{2.56}$$

where $\varepsilon \in (0, 1]$. This is also known as the ε-greedy policy. What happens here is that the agent either selects the greedy action with a probability $1 - \varepsilon$, or otherwise it samples a random action a from the action space. The randomness ε brings is to ensure that the agent explores different actions from time-to-time, which may give better return in the end. In Figure 2.4 we can a simple illustration depicting how ε-greedy is used in Q-learning.

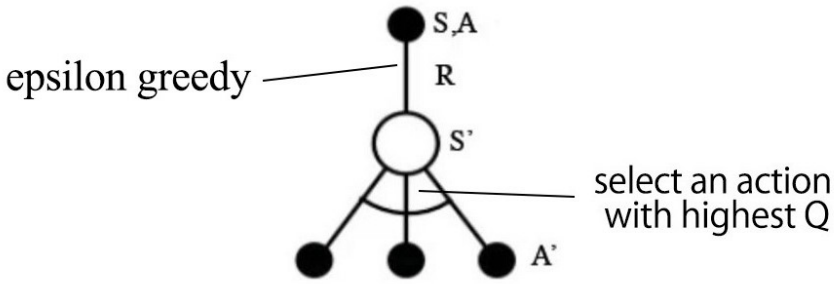

Figure 2.4: A simple Q-learning illustration. The agent performs an action under the ε-greedy policy, moves on to the next state and receives a reward. Then the agent selects the next action corresponding to the best Q-value [62].

Another attribute for Q-learning is that it's exploration-insensitive [45, 56, 61], meaning that the algorithm will converge to an optimal policy regardless of the action sampling scheme. This is under the assumption that each state-action pair is visited an infinite number of times and the learning rate α decreases with time in an appropriate way [45, 56].

2.3.3 Double Q-learning

In the Q-learning update in equation 2.53, the target policy $\max_a Q_i(s_{t+1}, a_t)$ is the greedy policy given the current action [37]. This value is an estimate for $\mathbb{E}[\max_a Q_i(s_{t+1}, a_t)]$, which in turn approximates $\max_a \mathbb{E}[Q_i(s_{t+1}, a_t)]$ [63]. If the last expectation is over-estimated, then by taking the maximum of that can lead to significant bias [37]. To see this more clearly, consider a state s with many actions a with true values $Q(s, a)$ all equal to zero. The estimated values $\hat{Q}(s, a)$ are all nonzero, which means when taking the maximum of the estimates we get positive values, but the maximum of the true values are zero. This positive bias is called **maximization bias** [37]. As a consequence, Q-learning suffers from large over-estimations of Q-values because they are

estimated from the maximum Q-value as an approximate for the maximum expected Q-value [63].

Van Hasselt (2010) [63] proposes a solution to this problem by applying a double estimator method to approximate the maximum expected Q-value. This algorithm is known **double Q-learning**. This algorithm considers two Q-value functions; $Q_1(s, a)$ and $Q_2(s, a)$, where each is an estimate of the true action-value $Q(s, a)$. Now Q_1 could be used estimate the maximal valued action $a^* = \arg\max_a Q_1(s, a)$, and Q_2 to obtain an estimate of this value, $Q_2(s, a^*) = Q_2(s, \arg\max_a Q_1(s, a))$. This estimate will be unbiased since $\mathbb{E}[Q_2(s, a^*)] = Q(s, a^*)$ [37]. This same method can be applied if we reversed the roles of Q_1 and Q_2, where the unbiased estimate is $Q_1(s, a^*) = Q_1(s, \arg\max_a Q_2(s, a))$. Note that even though we have two estimates that are learned, only one can be updated each transition.

In the double Q-learning update, we sample an action a in state s using a ε-greedy policy based in Q_1 and Q_2. This behaviour policy could be based on the average or sum of the Q-value functions. Then we select either of the double Q-learning update rules:

$$Q_{1,i+1}(s_t, a_t) = Q_{1,i}(s_t, a_t) + \alpha[r_{t+1} + \gamma Q_{2,i}(s_{t+1}, a^*) - Q_{1,i}(s_t, a_t)] \tag{2.57}$$

$$Q_{2,i+1}(s_t, a_t) = Q_{2,i}(s_t, a_t) + \alpha[r_{t+1} + \gamma Q_{1,i}(s_{t+1}, a^*) - Q_{2,i}(s_t, a_t)], \tag{2.58}$$

where a^* is the greedy action, $\alpha \in (0, 1]$ is the learning rate and $\gamma \in (0, 1)$ is the discount factor. These estimates are treated equally and the update we end up choosing is random, for example by using a coin flip (50%). Similarly to Q-learning, double Q-learning also converges to the optimal policy [63].

Before we can dive into deep Q-learning and its extensions, some deep learning background is necessary for the rest of this chapter. It is also required for the next chapter, where we review the algorithms used in our experiments.

2.4 Deep Learning

This section provides an overview of deep learning (DL) principles that are essential to comprehend the contributions of this thesis. The main DL components included in the algorithms in section 3.1 used for experiments are **neural networks** (NNs), and an **optimization algorithm**, which is used for optimizing the network parameters.

2.4.1 Neural Networks

Neural networks (NNs) is a very powerful and expressive machine learning architecture to learn arbitrary input-output functions, given that you have enough training data [64]. The goal of a NN is to define a mapping $\boldsymbol{y} = \hat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ and learn the parameters $\boldsymbol{\theta}$ that best approximates some function $\hat{f} : \boldsymbol{x} \rightarrow \boldsymbol{y}$ of the true function f [65]. The most

frequently used NN architectures are **multilayer perceptrons** (MLPs), **convolutional neural networks** (CNNs) and **recurrent neural networks** (RNNs) [66]. Since our algorithms only used MLPs, these will be emphasized in this section. These types of networks are also known as feedforward networks because there is no feedback, since the outputs of the different layers of network are fed back into itself. The input $\boldsymbol{x}$ flows only in one direction through the computational nodes $\{f_\ell\}_{\ell=1}^{L}$ used to define the NN f, and then comes out as the output $\boldsymbol{y}$. Here L denotes the number of layers in the network. Each layer node f_ℓ is a vector-to-vector function, consisting of **units** that work in parallel, each being a vector-to-scalar function [65]. Units are also referred to as **neurons**, where each one produces one element in the output vector. The layer nodes computation have the general form [65, 67, 68]:

$$f_\ell(\boldsymbol{x}_{\ell-1};\boldsymbol{\theta}_\ell) = \boldsymbol{W}\phi(\boldsymbol{x}_{\ell-1};\boldsymbol{\theta}_{\boldsymbol{\ell}}) + \boldsymbol{b}, \quad \ell = 1,\dots,L, \tag{2.59}$$

where

- $\boldsymbol{x}_{\ell-1}$ is the output vector of layer $\ell-1$, with $\boldsymbol{x_0}$ being the input vector,
- $\theta_\ell = \{\boldsymbol{W}, \boldsymbol{b}\}$ denotes the set of trainable parameters for layer ℓ,
- $\boldsymbol{W} \in \mathbb{R}_{k_\ell \times k_{\ell-1}}$ is the weight matrix,
- $\boldsymbol{b} \in \mathbb{R}_{k_\ell}$ is the bias vector, and
- ϕ denotes a nonlinear transformation (activation function) with respect to the layer parameters $\boldsymbol{\theta}_\ell$. These define the **hidden layers**.

Here we assume L number of layers, k_0 units in the input layer and k_ℓ units in the hidden layers. k_0 and k_ℓ are ruled by the input and output data respectively. Each layer can also represented by a weighted sum [67]

$$x_j^{(\ell)}(i) = \sum_{j=1}^{k_\ell}\sum_{k=1}^{k_\ell-1} W_{jk}^{(\ell)}\phi(x_j^{\ell-1}(i)) + b_j^{(\ell)}, \quad i = 1,\dots,N, \tag{2.60}$$

where N is the number of samples. The number of hidden layers L is known as the *depth* of the network, while the dimensionality of these layers k_ℓ determines the *width* of the model [65]. In Figure 2.5 one can see a generic MLP illustration.

Activation Function

The vector-valued function ϕ provides a set of features describing $\boldsymbol{x}$. The strategy in DL is to learn ϕ by parameterizing the representation as $\phi(\boldsymbol{x};\boldsymbol{\theta})$ and use some optimization algorithm to find the $\boldsymbol{\theta}$ yielding in the best representation [65]. This function is a non-linear transformation that gives the output of an unit, transforming linear features in the layers into non-linear ones. This gives the NN the ability to comprehend non-linear representations.

Typical choices for an activation function include:

- The sigmoid function

$$\phi(\boldsymbol{x};\sigma) = \frac{1}{1+e^{\sigma \boldsymbol{x}}}, \tag{2.61}$$

- The hyperbolic tangent function

$$\phi(\boldsymbol{x}) = \tanh(\boldsymbol{x}), \tag{2.62}$$

  and

- The rectified linear unit (ReLU)

$$\phi(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}, & x > 0 \\ 0, & x \leq 0. \end{cases} \tag{2.63}$$

In our implementations, the ReLU activation function was the one being utilized.

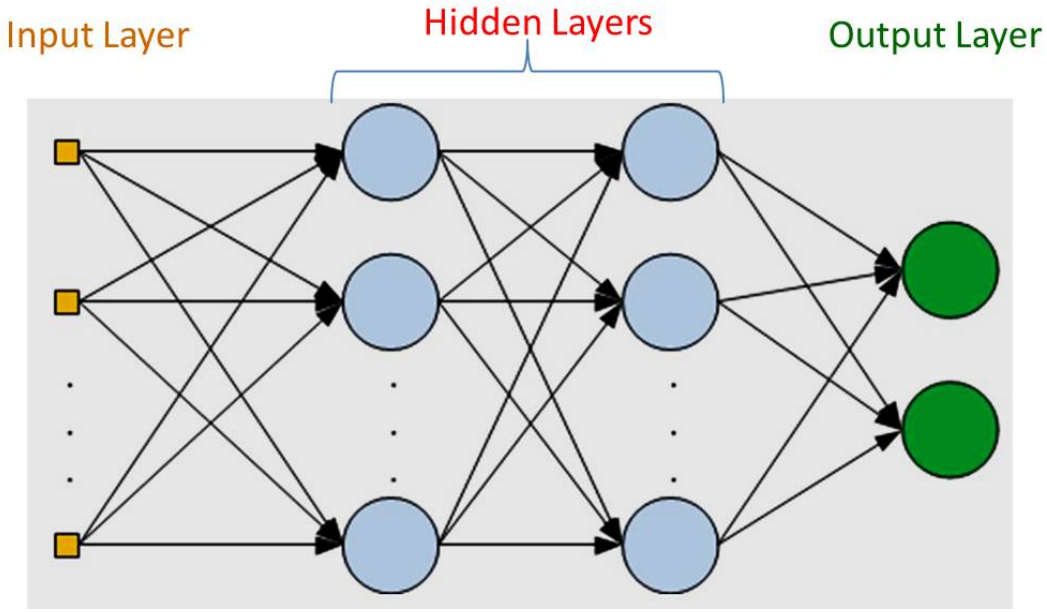

Figure 2.5: Figure illustrating a typical MLP with two hidden layers and two output layers [69].

Parameter Optimization and Loss Functions

In order to train the parameters $\boldsymbol{\theta}$ and reach the goal of the NN we need a measure for how well $f(\boldsymbol{x};\boldsymbol{\theta})$ is approximating $\hat{f}(\boldsymbol{x})$. This measure is known as the **loss function**, or **cost function**, and represents the problem the NN is striving to solve [65, 68], namely to optimize this function. Given some training data $\mathcal{X}$, with underlying ground-truths (desired outputs) $\boldsymbol{y}$, and the network outputs (predictions) $\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{\theta})$, a loss function can be formulated through some dissimilarity measure $\mathcal{E}$:

$$\mathcal{L}_i = \mathcal{E}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i). \tag{2.64}$$

The goal here is to minimize this loss with respect to the training data. We define the objective function $\mathcal{J}$ as

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{X}}[\mathcal{L}]. \tag{2.65}$$

The optimal network parameters are then obtained by minimizing this objective function [68]:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}). \tag{2.66}$$

Common loss functions include

- The mean square error (MSE) loss

$$\mathcal{L}_i = ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||_2^2, \tag{2.67}$$

  where $||\cdot||_2$ is the Euclidean norm, and

- The cross-entropy (CE) loss

$$\mathcal{L}_i = -(\boldsymbol{y}_i \ln(\hat{\boldsymbol{y}}_i) + (1 - \boldsymbol{y}_i)\ln(1 - \hat{\boldsymbol{y}}_i)). \tag{2.68}$$

The objective function for these loss functions become

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{X}}[\mathcal{L}_i] = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||_2^2, \tag{2.69}$$

for the MSE loss and

$$\mathcal{J}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{y}_i \ln(\hat{\boldsymbol{y}}_i) + (1 - \boldsymbol{y}_i)\ln(1 - \hat{\boldsymbol{y}}_i)), \tag{2.70}$$

for the CE loss. N is the number of samples in the data $\mathcal{X}$.

In our experiments, both loss functions were used. The MSE loss is the most common loss function to encounter in NN architectures, and the CE loss is used for categorical network outputs [70], or in our case, for categorical deep Q-learning.

2.4.2 Optimization Algorithms

There are many ways to solve the minimization task in equation 2.66, and the common intuition is to follow the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})$ towards its minimum [68]. This technique is referred to as the **backpropagation algorithm** [71], which allows the information from the loss function to flow backwards through the NN in order to then compute the gradient of the loss [65]. The details of this algorithm is out of the scope of this thesis, and we therefore encourage the reader to refer to Chapter 6.5 in Goodfellow et al. (2016) [65] for a deeper understanding of the subject.

In our experiments, the optimization algorithm that was incorporated was the **adaptive moment estimation** (Adam) [72], which is based off of the **stochastic gradient**

descent (SGD) algorithm [73].

The SGD algorithm computes the gradient of the loss function and then updates the parameters according to the estimated gradient:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla_{\boldsymbol{\theta}_k} \mathcal{J}(\boldsymbol{\theta}_k), \tag{2.71}$$

where α is the learning rate. Iteratively the weighted gradient estimations is subtracted from the current parameters to obtain the new parameters.

Sometimes SGD struggles to navigate through ravines [74], i.e., areas where the surface curves much more steeply in one dimension than in another [75]. Momentum SGD [76] is a modification that can accelerate SGD in the relevant directions [74]. It takes into account past gradients to smooth out the update. The update rule is given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \boldsymbol{v}_k, \tag{2.72}$$

where $\boldsymbol{v}_k$ is the exponentially weighted average of past gradients expressed as

$$\boldsymbol{v}_k = \gamma \boldsymbol{v}_{k-1} + (1-\gamma) \nabla_{\boldsymbol{\theta}_k} \mathcal{J}(\boldsymbol{\theta}_k), \tag{2.73}$$

where γ is the momentum term, which is usually set to 0.9 [74]. The momentum results in faster convergence and reduced oscillation in the SGD.

Adam [72] is the optimizer that was used in the experiments of this thesis. It is a combination of momentum SGD and RMSprop [77], in that it uses both exponentially weighted averages of past gradients $\boldsymbol{v}_k$ and exponentially weighted averages of past squared gradients $\boldsymbol{s}_k$. We get then get the following update equations:

$$\boldsymbol{v}_k = \gamma \boldsymbol{v}_{k-1} + (1-\gamma) \nabla_{\boldsymbol{\theta}_k} \mathcal{J}(\boldsymbol{\theta}_k) \tag{2.74}$$

$$\boldsymbol{s}_k = \eta \boldsymbol{s}_{k-1} + (1-\eta) [\nabla_{\boldsymbol{\theta}_k} \mathcal{J}(\boldsymbol{\theta}_k)]^2, \tag{2.75}$$

where γ and η are decay parameters. The authors of Adam [72] found that the estimates for $\boldsymbol{v}$ and $\boldsymbol{s}$ were biased towards zeros [74], especially during initial iterations and when the decay parameters were small. The bias-corrected estimates were then calculated through

$$\hat{\boldsymbol{v}}_k = \frac{\boldsymbol{v}_k}{1-\gamma^k} \tag{2.76}$$

$$\hat{\boldsymbol{s}}_k = \frac{\boldsymbol{s}_k}{1-\eta^k}. \tag{2.77}$$

The parameter update is then given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\hat{\boldsymbol{v}}_k}{\sqrt{\hat{\boldsymbol{s}}_k} + \epsilon}, \tag{2.78}$$

where ϵ is a very small number to avoid dividing by zero. The authors proposed hyperparameters default values as 0.99 for γ and η, and 10^{-8} for ϵ [72, 74].

2.5 Deep Q-Learning Extensions

In the previous sections we covered the foundations for deep RL, which combines NNs with a RL architecture [78]. NNs work as function approximators and could be used in RL to approximate a value function or a policy. With **deep Q-learning** (DQL) the Q-value function is approximated using a NN [79]. The input for the NN is the current state and the output is the Q-value of all the possible actions the agent can take. The comparison between Q-learning and DQL is illustrated in figure 2.6.

The first DQL algorithm, often referred to as the **deep Q-network** (DQN) algorithm, was proposed by Mnih et al. (2013) [80] from DeepMind Technologies. Since then many improvements to the original DQN algorithm has been made [81] - [86], and **Rainbow DQN**, which combines all DQN algorithms, is currently the state-of-the-art algorithm on ATARI games [87].

The purpose of this section is to get an understanding of the DQN extensions, which we then will put together in our own algorithms in section 3.1.5. Since there are many DQN extensions to cover in this section, a brief overview of each of them will be given. We encourage the inclined reader to refer to the original papers of the various DQN algorithms for a more in-depth review [80] - [86].

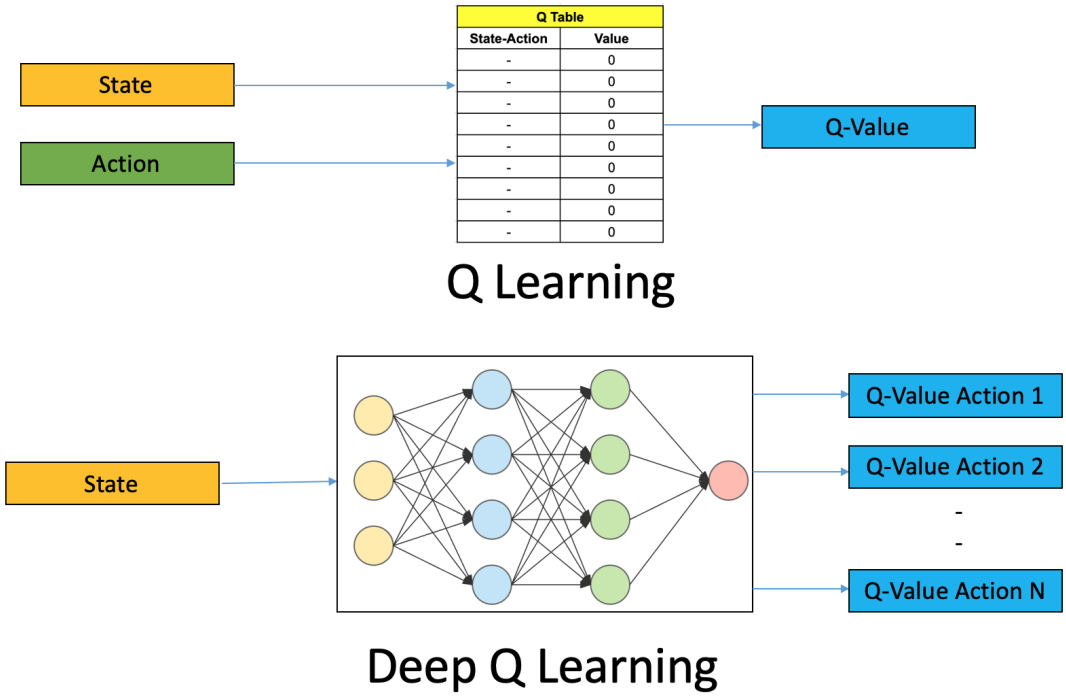

Figure 2.6: An illustration that compares Q-learning and deep Q-learning [79]. In Q-learning state-action pairs are used to estimate Q-values stored in a Q-table. With DQL a NN is approximating the Q-value function. The input is the state and the output are the Q-values of all possible actions.

2.5.1 Deep Q-learning

As mentioned earlier deep Q-learning, or DQN, was the first extension to Q-learning, where a NN $Q_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is used to approximate Q^* with network parameters $\boldsymbol{\theta}$ [88]. Recall from earlier that the optimal Q-value can be written as

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{S}}[r + \gamma \max_{a'} Q^*(s',a')|s,a], \tag{2.79}$$

where s' is the next state, a' is the next action, r is the reward and γ is the discount factor. To achieve an optimal Q-value we must iteratively converge from Q_i to Q^* as $i \to \infty$:

$$Q_{i+1}(s,a;\boldsymbol{\theta}) = \mathbb{E}_{s' \sim \mathcal{S}}[r + \gamma \max_{a'} Q_i(s',a';\boldsymbol{\theta})|s,a]. \tag{2.80}$$

To train the Q-network with parameters $\boldsymbol{\theta}$, we minimize a sequence of MSE loss functions for each iteration i [80]:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho}\left[(y_i - Q(s,a;\theta_i))^2\right], \tag{2.81}$$

where $\rho = \rho(s,a)$ is the *behaviour distribution* which is a probability distribution over state s and action a and $y_i = \mathbb{E}_{s' \sim \mathcal{S}}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target for iteration i. The next step is to differentiate the sequences of loss functions:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho, s' \sim \mathcal{S}}[(y_i - Q(s,a;\theta_i))\, \nabla_{\theta_i} Q(s,a;\theta_i)]. \tag{2.82}$$

We can then update the network parameters according to

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta_i} L_i(\theta_i), \tag{2.83}$$

where α is the learning rate.

The steps involved in the DQN algorithm can be summarized with the following procedure [79, 90]:

- Store all the past experiences of the agent in a memory, known as the **experience replay** [89], denoted $\mathcal{E}$. At each time step t, save the transition $e_t = (s_t, a_t, r_t, s_{t+1})$ and store it in the memory $\mathcal{E} = e_1, \ldots, e_N$, over N episodes.
- Choose an action according to the ε-greedy exploration of the output of the DQN.
- Estimate and minimize the MSE loss to then update the network parameters $\boldsymbol{\theta}$.
- Sample a random transition from $\mathcal{E}$ and repeat the training process until $Q_{\boldsymbol{\theta}} \to Q_{\boldsymbol{\theta}^*}$, or until specified.

This algorithm is included in Algorithm 1.

2.5.2 Double Q-Learning

For the same reasoning that was discussed in section 2.3.3, double Q-learning in the DQL setting is improving upon the overestimation of the Q-values under certain conditions [81]. The double DQN (DDQN) proposes two Q-value approximators, represented by two NNs, where each one gets updated by the other for the next state [91]. Originally, the DQN calculated both the predicted Q-value and the target Q-value. which could lead to divergence between the two [79]. So with DDQN, we have one of the Q-networks estimate the target Q-values, which then will be referred to as the **target network**. The other Q-network, known as the **local network** or **prediction network**, will be used to estimate the predicted Q-values. The target network will have the same architecture as the local network, and with the same parameters initialized [81]. To update the target network we copy the parameters from the local network $\boldsymbol{\theta}$, and this is done every τ iterations. The overall effect of DDQN reduces the influence of overestimation, which leads to better performance compared to standard DQN and more stable training due to keeping the target network somewhat fixed [79, 81, 87, 91].

The new target Q-values for DDQN can be written as [81]

$$y_i = r + \gamma Q(s, \arg\max_a Q(s, a; \theta_i); \omega_i), \tag{2.84}$$

where θ_i are the local network parameters and ω_i are the target network parameters.

This algorithm is included in Algorithm 2, that combines DDQN and dueling DQN, which is the next algorithm to be reviewed.

2.5.3 Dueling Network Architectures

The purpose of this DQN extension is to change the Q-network's architecture to represent two separate estimators [82]: one for the state-value function and one for the advantage function. We defined the advantage function in section 2.5.3 as the difference between the Q-value function and the state-value function: $A(s, a) = Q(s, a) - V(s)$.

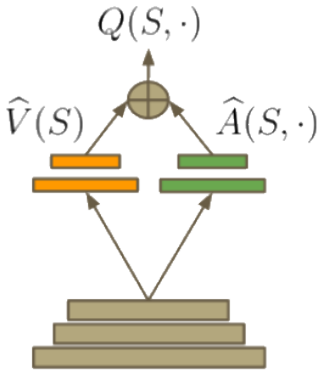

Figure 2.7: A simple illustration of the dueling network [92].

The dueling DQN splits the input data (states) into two branches, where each branch utilize as an estimator for either $V(s)$ or $A(s, a)$. At the end, these two streams will

add up in the sum operator, as depicted in figure 2.7. In this node we will also subtract either the maximum advantage of the next state, or the average advantage [82, 87].

Defining $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as the parameters of the advantage stream and the value stream respectively, we can express the Q-value function as

$$Q(s, a; \boldsymbol{\theta}, \alpha, \beta) = V(s; \boldsymbol{\theta}, \beta) + A(s, a; \boldsymbol{\theta}, \alpha). \tag{2.85}$$

The above equation evokes a problem in the sense that the two streams are unidentifiable when summed up to estimate Q [82]. The solution to this issue is to force the advantage stream to have zero advantage at the chosen action. To do that we simply do as we stated earlier with regards to the sum operator. We either subtract the maximum advantage of the next action a'

$$Q(s, a; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = V(s; \boldsymbol{\theta}, \boldsymbol{\beta}) + \left(A(s, a; \boldsymbol{\theta}, \boldsymbol{\alpha}) - \max_{a' \in |\mathcal{A}|} A(s, a'; \boldsymbol{\theta}, \boldsymbol{\alpha}) \right), \tag{2.86}$$

or we subtract the average advantage

$$Q(s, a; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = V(s; \boldsymbol{\theta}, \boldsymbol{\beta}) + \left(A(s, a; \boldsymbol{\theta}, \boldsymbol{\alpha}) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'; \boldsymbol{\theta}, \boldsymbol{\alpha}) \right). \tag{2.87}$$

If the agent now selected the greedy action a^*, from equation 2.86 we obtain

$$Q(s, a^*; \boldsymbol{\theta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = V(s; \boldsymbol{\theta}, \boldsymbol{\beta}). \tag{2.88}$$

This means that the V stream provides a true estimate for the state-value function [82]. Using equation 2.87 instead will always lead to an off-target between V and A by a constant, as a result from the mean advantage. This will result in unidentifiability between the streams, but increases the stability of the optimization [82].

2.5.4 Prioritized Experience Replay

As we saw in the DQN algorithm, it stores all transitions in a replay buffer $\mathcal{E}$ and then randomly samples from this buffer to train its network parameters. This method suffers from low efficiency since many of the stored transitions can be useless in the sense that they do not give the highest amount of reward in the long-run [83, 87]. With prioritized experience replay (PR) we want to sample important transitions more regularly by giving a priority measure to these transitions and then sample using importance-sampling (IS) [83, 93].

Transitions are sampled from the buffer according to a stochastic prioritization probability $P(i)$, where i is the transition index. This probability is defined as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{2.89}$$

where $p_i > 0$ is the priority of transition i and α is the prioritization parameter [83].

PR introduces bias in the high priority samples with stochastic updates, because these are more likely to be sampled in the DQN. To correct for this bias, PR DQN utilizes IS weights in the Q-learning update [83, 93]. These weights are given by

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}, \tag{2.90}$$

where β is the IS parameter.

The choices for α and β varies, but Schaul et al. (2015) [83] recommends $\alpha = 0.6$ and β as 0.4 initially and then linearly annealed to 1.

This algorithm is included in Algorithm 3.

2.5.5 Noisy Networks for Exploration

Another issue with the standard DQN is the exploration. DQN incorporate its exploration through ε-greedy which is inefficient because it relies on randomness in the policy [87]. Fortunato et al. (2017) [84] proposes a deep RL agent called NoisyNet, or **noisy DQN**, which uses parametric noise to its weights and can replace the previous exploration method.

To integrate the noise into the network architecture, a parametric function $y = f_\theta(x)$ of the noisy parameters $\boldsymbol{\theta}$ must unsettle the NN weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$. We can define the noisy parameters as

$$\theta \doteq \boldsymbol{\mu} + \boldsymbol{\Sigma} \odot \boldsymbol{\varepsilon}, \tag{2.91}$$

where $\boldsymbol{\zeta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ is a set of trainable parameters, $\odot$ denotes the element-wise multiplication and $\boldsymbol{\varepsilon}$ is a zero-mean noise vector [84]. Consider now a linear layer $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ with k inputs and ℓ outputs. The corresponding noisy layer is then defined as

$$\boldsymbol{y} \doteq (\boldsymbol{\mu}_W + \boldsymbol{\sigma}_W \odot \boldsymbol{\varepsilon}_W)\boldsymbol{x} + \boldsymbol{\mu}_b + \boldsymbol{\sigma}_b \odot \boldsymbol{\varepsilon}_b, \tag{2.92}$$

where $\boldsymbol{\mu}_W + \boldsymbol{\sigma}_W \odot \boldsymbol{\varepsilon}_W$ replace $\boldsymbol{W}$ and $\boldsymbol{\mu}_b + \boldsymbol{\sigma}_b \odot \boldsymbol{\varepsilon}_b$ replace $\boldsymbol{b}$. The learnable parameters are $\boldsymbol{\mu}_W \in \mathbb{R}_{\ell \times k}$, $\boldsymbol{\sigma}_W \in \mathbb{R}_\ell$, $\boldsymbol{\mu}_b \in \mathbb{R}_{\ell \times k}$ and $\boldsymbol{\sigma}_b \in \mathbb{R}_\ell$, and the parameters $\boldsymbol{\varepsilon}_W \in \mathbb{R}_{\ell \times k}$ and $\boldsymbol{\varepsilon}_b \in \mathbb{R}_\ell$ are random noise variables [84]. The learnable parameters are therefore subsets of $\boldsymbol{\mu} = \{\boldsymbol{\mu}_W, \boldsymbol{\mu}_b\}$ and $\boldsymbol{\Sigma} = \{\boldsymbol{\sigma}_W, \boldsymbol{\sigma}_b\}$, and the set $\boldsymbol{\zeta}$ are the parameters that optimize the noisy DQN. A graphical representation of the noisy layer is showcased in figure 2.8.

There are two choices proposed by Schaul et al. (2017) [84] for the noisy parameters:

- Independent Gaussian noise, where each element of random matrix $\boldsymbol{\varepsilon}_W$ and random vector $\boldsymbol{\varepsilon}_b$ are drawn from a unit Gaussian distribution.

- Factorized Gaussian noise, where we factorize the noise parameters using a real-valued function f:

$$\varepsilon_w[i,j] = f(\varepsilon_i) f(\varepsilon_j), \tag{2.93}$$

and

$$\varepsilon_b[j] = f(\varepsilon_j), \tag{2.94}$$

where ε_i is unit Gaussian noise for the k inputs and ε_j is the unit Gaussian noise for the ℓ outputs.

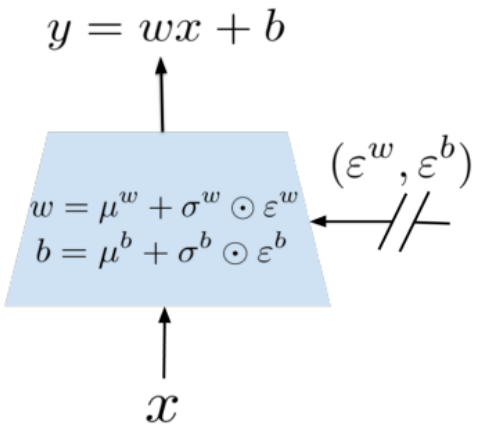

Figure 2.8: A noisy layer illustration. The parameters $\boldsymbol{\mu}_W$, $\boldsymbol{\sigma}_W$, $\boldsymbol{\mu}_b$ and $\boldsymbol{\sigma}_b$ are learnable of the network, while $\boldsymbol{\varepsilon}_W$ and $\boldsymbol{\varepsilon}_b$ are noise variables [84].

The loss of the noisy DQN is the expectation over the noisy parameters:

$$\mathcal{L}(\boldsymbol{\zeta}) = \mathbb{E}[L(\boldsymbol{\theta})] = \mathbb{E}[L(\boldsymbol{\mu} + \boldsymbol{\Sigma} \odot \boldsymbol{\varepsilon})], \tag{2.95}$$

where $L(\cdot)$ represents the TD-error. The gradients are then given by

$$\nabla \mathcal{L}(\boldsymbol{\zeta}) = \mathbb{E}[\nabla_{\boldsymbol{\mu},\boldsymbol{\Sigma}} L(\boldsymbol{\mu} + \boldsymbol{\Sigma} \odot \boldsymbol{\varepsilon})]. \tag{2.96}$$

In the paper by Schaul et al. (2017) [84], they used a Monte Carlo approximation of the above gradient, where they take a single sample $\boldsymbol{\xi}$ each step of the optimization:

$$\mathcal{L}(\boldsymbol{\zeta}) \approx \nabla_{\boldsymbol{\zeta}} L(\boldsymbol{\mu} + \boldsymbol{\Sigma} \odot \boldsymbol{\xi}). \tag{2.97}$$

Expanding the noisy DQN loss function into an update rule yields:

$$\mathcal{L}_i(\zeta_i) = \mathbb{E}\left[\mathbb{E}_{s,a,r,s' \sim \mathcal{E}}[(y_i - Q(s,a,\boldsymbol{\varepsilon};\zeta_i))^2]\right], \tag{2.98}$$

where ζ_i are the local network parameters, $\mathcal{E}$ is the replay buffer, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon}'$ are noisy parameters, $y_i = r + \gamma \max_{a'} Q(s', a', \boldsymbol{\varepsilon}'; \psi_i)$ are the noisy target values, and ψ_i are the target network parameters.

This algorithm is used in Algorithm 4, that combines both noisy DQN and categorical DQN, which is the next algorithm to be reviewed.

2.5.6 Categorical Deep Q-learning

The main idea behind **categorical DQN**, or **distributional DQN**, is that instead of learning an estimation of the Q-value function we learn a distribution of the Q-values [87].

Bellemare et al. (2017) [85] proposes the main object as the random return Z, whose expectation is the Q-value. We can represent Z as the **distributional Bellman equation**

$$Z(s,a) \doteq R(s,a) + \gamma Z(s',a'). \tag{2.99}$$

This is also known as the **value distribution**. The Q-values can then be estimated by

$$Q_\pi(s,a) = \mathbb{E}[Z_\pi(s,a)]. \tag{2.100}$$

They also define the **Bellman operator** $\mathcal{T}_\pi$ and **Bellman optimality operator** $\mathcal{T}$ as operators that describe Q-learning and they are both contraction mappings [45, 85]. These are expressed as

$$\mathcal{T}_\pi Q(s,a) = \mathbb{E}R(s,a) + \gamma \mathbb{E}_\pi Q(s',a') \tag{2.101}$$
$$\mathcal{T} Q(s,a) = \mathbb{E}R(s,a) + \gamma \mathbb{E} \max_{a' \in \mathcal{A}} Q(s',a'). \tag{2.102}$$

These operators converge some value Q_0 to either Q_π or Q^*, over repeated applications [85].

In the paper by Bellemare et al. (2017) [85], they propose an algorithm known as the **C-51**, which is based on the Bellman optimality operator. To get there they approximate the value distribution using the parametric distribution

$$Z_\theta(s,a) = z_i, \tag{2.103}$$

with probabilities

$$p_i(s,a) = \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}}. \tag{2.104}$$

Here z_i is known as the set of **atoms** defined as

$$z_i = \{V_{\text{MIN}} + i\Delta z : 0 \leq i < N\}, \tag{2.105}$$

where $\Delta z = (V_{\text{MAX}} - V_{\text{MIN}})/(N-1)$, $N \in \mathbb{N}$ is the number of atoms, $V_{\text{MAX}}, V_{\text{MIN}} \in \mathbb{R}$ are the value bounds and $\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^N$ are the model parameters. This discrete distribution has shown to being highly expressive and computation friendly [94].

In the algorithm, the Bellman update in DQL is replaced with a **projected Bellman update**, where we project the sample Bellman update $\hat{\mathcal{T}} Z_\theta$ onto the support Z_θ. The Bellman update $\hat{\mathcal{T}} z_j = r + \gamma z_j$, for each set of atoms z_j is calculated, then its probabilities p_j are distributed to the neighbours of $\hat{\mathcal{T}} z_j$ [85]. Then the i-th component of the projected Bellman update is given by

$$\Phi \mathcal{T} Z_\theta(s,a)_i = \sum_{j=0}^{N-1} \left[1 - \frac{|[\mathcal{T} z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}} - z_i|}{\Delta z} \right]_0^1 p_j(s', \pi(s')), \tag{2.106}$$

where $[\cdot]_a^b$ bounds its argument in the range $[a, b]$. The loss that was used in this algorithm were given by the CE term of the KL divergence

$$D_{\text{KL}}(\Phi \hat{\mathcal{T}} Z_\omega(s,a) || Z_\theta(s,a)), \tag{2.107}$$

where ω are the target network parameters. The KL divergence is known as the Kullbeck-Leibler divergence.

This algorithm was called C-51 because Bellemare et al. (2017) [**?**] found that the results using $N = 51$ atoms out-performed DQN in a striking way. It reached state-of-the-art performance in SEAQUEST and out-performed DQN in 5 different ATARI games.

2.5.7 Rainbow - Combining Deep Q-learning Algorithms

The **rainbow DQN**, or just **rainbow**, is a DQN extension that combines and improves with all the previous DQN algorithms we have covered in this section [86, 87]. It is currently the state-of-the-art algorithm on ATARI games, both in terms of data efficiency and final performance.

In the paper by Hessel et al. (2017) [86], they integrate all the DQN components into one agen: Rainbow. They then define the target distribution as

$$d_t^{(n)} = (r_t^{(n)} + \gamma_t^{(n)} \boldsymbol{z}, \boldsymbol{p}_\omega(s_{t+n}, a_{t+n}^*)), \tag{2.108}$$

and the loss as

$$D_{\text{KL}}(\Phi_z d_t^{(n)} || d_t), \tag{2.109}$$

where Φ_z is the projection onto z. In the experiments they prioritized transitions by the KL loss:

$$p_t \propto D_{\text{KL}}(\Phi_z d_t^{(n)} || d_t)^\kappa, \tag{2.110}$$

where $\propto$ is the proportional symbol and κ is the prioritization parameter. It is used because the KL loss is what the network is minimizing.

The parametric distribution $p_\theta(s,a)$, which is used to estimate the value distribution, is obtained through the combined streams of the value and the advantage, just like in dueling DQN, and then passed through a softmax layer [95]. These distributions can be calculated with the equation

$$p_\theta^i(s,a) = \frac{\exp(v_\eta^i(\phi)) + a_\psi^i(\phi,a) - a_\psi^{-i}(s)}{\sum_j \exp(v_\eta^j(\phi)) + a_\psi^j(\phi,a) - a_\psi^{-j}(s)}, \tag{2.111}$$

where $\psi = f_\xi(s)$ is the shared representation that is fed into the value stream v_η and the advantage stream a_ψ, $a^i_\xi(f_\xi(s), a)$ is the output corresponding to atom i and $a^{-i}_\psi = (1/|\mathcal{A}|) \sum_{a'} a^i_\psi(\phi, a')$. The parameters ξ, η, ψ are the shared encoder parameters, which make up the local network parameters $\theta = \{\xi, \eta, \psi\}$ [86].

Chapter 3

Deep Q Algorithms for Blood Glucose Control

In this chapter, we present our Deep Q algorithms whose goal is to control the BG level in T1D patients. These algorithms draw inspiration from the paper by Fox and Wiens (2019) [23].

The following sections will describe the elements of the algorithms and the simulations. In Section 3.1 we will define the state space, action space, reward function and policy, as well as provide some pseudocode for the algorithms. Then in Section 3.2, we will explain how environments are utilized and how we run simulations.

Much like Fox and Wiens (2019) [23], we frame the BG regulation problem as a MDP, consisting of a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$.

3.1 Algorithm Descriptions

As we reviewed in Chapter 2, RL is a machine learning technique characterized by an agent learning how to adapt its behaviour to a given environment. In the context of BG regulation, we need to set the foundations for this specific problem.

We reviewed in Section 1.1.2 what the required components are for an AP: a CGM, a control algorithm and an insulin infusion mechanism. Our environment consists of a CGM, an insulin pump, and the body of a patient to be simulated. Here, the controller is the only component that is part of the agent, which dictates the theoretical insulin amount to be injected. The pump controls the actual amount.

3.1.1 State and Action Space

There are some conditions that needs to be met when defining the states and actions for this environment. The CGM gives us information about the BG levels in a patient's body, which is the only data we receive from it. This implies that the states of the environment has to depend on the data from the CGM. We mentioned earlier that the agent only controls the theoretical insulin amount to be injected by the insulin pump. This also implies that there is a connection between this insulin amount and actions that can be performed.

Assuming that the above conditions are met, characterizing the states and actions of the environment can be done. As there are many different ways of doing this, a choice of design must be made.

The CGM measures the BG level of a patient every minute. An option would be to use the last BG measure as the state s_t. The agent would then have to update its policy every minute, and consequently, the action a_t is the amount of insulin to be injected by the insulin pump during the next minute, or until the next BG measure by the CGM [50]. Alternatively, one could use BG data from the last 30 minutes or hour. Then the policy update happens every 30/60 minutes.

Huyett et al. (2015) [96] states that many state-of-art AP designs utilize commercially available insulin pumps and CGMs that operate in the subcutaneous space, which leads to serious delays into the control loop. When using subcutaneous insulin pumps, the absorption peaks occur after 50-60 min [96, 97, 98]. With intraperitoneal insulin pumps, these delays are reduced to 20-25 minutes [96, 99]. Both of these devices have a insulin residence time of several hours, more specifically, 6-8 hours and 1-2 hours for subcutaneous and intraperitoneal insulin pumps respectively [96]. This implies that the actions from an agent would not be reflected immediately by the CGM measurements [50].

Our hybrid closed-loop system utilizes subcutaneous devices that use short-acting insulin (e.g., lispro), which can be read about in more detail in A.1.3. Short-acting insulin, or often referred as regular insulin, starts to work after 30-60 minutes and peaks after around 2-4 hours. This again implies that the BG data alone is not practicable as a state representation.

The above dilemma sparks a motivation to include insulin information in the state representation. Similarly to Fox and Wiens (2019) [23], the choices for the state and action spaces is defined below.

States

The states $\boldsymbol{s}_t \in \mathcal{S}$ consists of the previous 30 minutes of BG data, as well as the 4 last insulin actions (last 2 hours) at a time resolution of 1-minute: $\boldsymbol{s}_t = [\boldsymbol{G}_t, \boldsymbol{I}_t]$, where

$\boldsymbol{G}_t = [g_{t-29}, g_{t-28}, \ldots, g_t]$, $\boldsymbol{I}_t = [i_{t-3}, i_{t-2}, i_{t-1}, i_t]$, and

$g_t \in \mathbb{R}_{0:500}$, $i_t \in \mathbb{R}_+$ and $t \in \mathbb{N}_{0:72}$. Here g_t [mg/dL] is the BG data, i_t [mU/min] is the insulin data and t is the time index, where one time step is 30 minutes. The time step limit is at 72. This is because when we simulate a patient, an episode lasts 1.5 day (24h $+12$h $= 36$h). This is for taking into account the whole night before the next day. This is in total 2160 minutes, and divided by the 30 minute step we obtain $2160/30 = 72$.

Actions

Now that the definition of the state variable has been established, the agent perform an action $a_t \in \mathcal{A}$ every t steps, i.e., every 30 minutes. We define a_t as real positive numbers $\mathcal{R}_+$ in a discrete action space. In the experiments to come we used two action spaces: $\mathcal{A}_1 = \{b_0, b^*, 3b^*\}$ and $\mathcal{A}_2 = \{b_0, b^*/2, b^*, 2b^*, 3b^*\}$, where b_0 is 0 insulin (stop the insulin pump) and b^* [mU/min] is the optimal basal rate, which is set to 6.43. Note that both action spaces have the same minimum and maximum, the only difference is $\mathcal{A}_2$ has two more actions in-between.

3.1.2 Transition Function

The transition function $P \in \mathcal{P}$ consists of two elements [23]:

1. The meal schedule $M : t \to c_t$, where $c_t \in \mathbb{R}_+$ is the amount of carbohydrates [mmol].
2. The model of the glucoregulatory system $C : (c_t, a_t) \to (g_{t+1}, i_{t+1})$. C is defined in according to the Havorka model, which can be studied in greater detail in A.1.

3.1.3 Reward Function

We obtain a new state $\boldsymbol{s}_t$ every 30 minutes, and when the agent performs an action a_t we receive the next state $\boldsymbol{s}_{t+1}$ and a reward $r_t \in \mathcal{R} \subset \mathbb{R}$.

The reward function R is defined as Gaussian function:

$$R(g_t) = e^{-\frac{1}{2}(g_t - b_r)^2/900}, \tag{3.1}$$

where g_t is the BG level, b_r [mg/dL] is the BG reference, which is set to 108.

In addition to this, the simulator checks if the BG levels are within valid bounds, i.e., $[g_\ell, g_h]$ [mg/dL], where $g_\ell = 70$ is the lower bound and $g_h = 180$ is the higher bound. So, if $g_t \in [g_\ell, g_h]$, then $r_t > 0$. Otherwise, receive a reward of $r_t = -1000$, which can be interpreted as a punishment.

3.1.4 Policy

In the previous chapter we reviewed Q-learning, and for most of the DQL frameworks ε-greedy exploration is used. It is defined as

$$\pi(a_t|\boldsymbol{s}_t) = \begin{cases} 1-\varepsilon, & \text{if } a_t = \arg\max_a Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}) \\ \frac{\varepsilon}{|\mathcal{A}|-1} & \text{otherwise} \end{cases} \tag{3.2}$$

Here $\arg\max_a Q(\boldsymbol{s}_t, a; \boldsymbol{\theta}) = a^*$ is the optimal action and $\boldsymbol{\theta}$ are the NN weights that parameteritize the policy.

The purpose of ε-greedy exploration is to select a random action a_t with probability ε, and otherwise choose the optimal action a^* with probability $1-\varepsilon$. One main problem with this strategy is that the agent tends to explore too much, even though an action is the optimal one [100]. This can lead to missed opportunities and increase total regret. Given an optimal action a^* that yields the highest reward, the regret L_T over T attempts is defined as

$$L_T = TE[r|a^*] - \sum_{t=1}^{T} E[r|a_t]. \tag{3.3}$$

As $T \to \infty$, it can be proven that the regret tends to a lower bound [100]:

$$\lim_{T\to\infty} \geq k \log T, \tag{3.4}$$

where k is some constant.

Because of this increased regret generated by the ε-greedy exploration, a decaying ε-greedy exploration was used with decay function

$$\varepsilon(t) = \varepsilon_F + (\varepsilon_0 - \varepsilon_F)e^{-t/\eta}, \tag{3.5}$$

where ε_0 is the initial ε-value, ε_F is the final value and η is the decay. This method tries to decrease the percentage dedicated for exploration over time [100]. The challenge with this however, is to control the decaying process. If ε decayed too fast, then the agent will end up having no exploration and exploitation might be a difficult task. Too slow decay and the agent will face the same consequences as with ε-greedy, or worse.

3.1.5 Pseudocode

With every element of the simulator and algorithm defined, the pseudocode for the different algorithms can be described. In section 2.6, an overview of the most important DQN extensions was made. Note that some of the extensions are combined in our algorithm showcasing to prevent redundancies. Further details on specific extensions will be made as we go through this subsection.

The included algorithms are: DQN [80], DDQN [81], Dueling DQN and Dueling DDQN [82], Prioritized Replay DQN [83], Noisy Networks DQN [84], Categorical

DQN [85] and Rainbow DQN [86].

Algorithm 1 showcases the DQN algorithm, inspired by the works of Mnih et al. [80]. It is worth mentioning again that an episode is terminated after 72 time steps. Since one time step represents 30 minutes, 72 steps is equivalent to 1.5 days, where one day is 24 hours.

A general note for all the algorithms is that when meals and meal indicators are generated, we actually refer to the carbohydrate (CHO) intakes and estimated CHO intakes.

Algorithm 1 - DQN

Initialize experience replay memory $\mathcal{E}$ with capacity C
Initialize deep Q network with random weights θ_0
Generate meals and meal indicators
Initialize state $s_1 \sim \varsigma$ from environment reset
for time steps $t = 1, T$ **do**:
    Perform ε decay
    With a probability $1 - \varepsilon$ select the greedy action $a_t = \max_a Q^*(s_t, a; \theta)$
    otherwise select a random action a_t
    Perform action a_t in simulator and receive new state s_{t+1} and reward r_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{E}$ and set $s_{t+1} = s_t$
    if episode is done **do**:
        Reset meals, meal indicators and state
    end
    if $\mathcal{E}$ is greater than the batch size B **do**:
        Sample random mini-batch of transitions $(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{E}$
        Set $y_i = \begin{cases} r_i & \text{for terminal } s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta_{i-1}) & \text{for non-terminal } s_{i+1} \end{cases}$
        Calculate the loss $L_i(\theta_i) = (y_i - Q(s_i, a_i; \theta_i))^2$
        Perform gradient descent step on $L_i(\theta_i)$
    end
end

In Algorithm 2, we showcase both DDQN and dueling DQN. This is inspired by the algorithms stated in the papers by van Hasselt et al. [81] and Wang et al. [82]. In our experiments, a combination of these algorithms was tested, i.e., DDQN and dueling for both regular DQN and DDQN. The algorithms to come will not follow the same procedure, as this was simply an investigation.

Algorithm 2 - Double & Dueling DQN

Initialize experience replay memory $\mathcal{E}$ with capacity C
Initialize local weights θ_0 and set target weights ω_0 as copy
Generate meals and meal indicators
Initialize state $s_1 \sim \varsigma$ from environment reset
for time steps $t = 1, T$ **do**:
    Perform ε decay and choose action $a_t \sim \pi_\theta(s_t)$
    Execute a_t and receive new state s_{t+1} and reward r_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{E}$ and set $s_{t+1} = s_t$
    if episode is done **do**:
        Reset meals, meal indicators and state
    end
    if $\mathcal{E}$ is greater than the batch size B **do**:
        Sample random mini-batch of transitions $(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{E}$
        Define $a_{\max}(s'; \theta) = \arg\max_a Q(s', a'; \theta)$
        Set $y_i = \begin{cases} r_i & \text{for terminal } s_{i+1} \\ r_i + \gamma Q(s_{i+1}, a_{\max}(s_{i+1}; \theta_i); \omega_i) & \text{for non-terminal } s_{i+1} \end{cases}$
        Calculate the loss $L_i(\theta_i) = (y_i - Q(s_i, a_i; \theta_i))^2$
        Perform gradient descent step on $L_i(\theta_i)$
    end
    Copy network weights $\omega \leftarrow \theta$ every τ steps
end

A version of Schaul et al.'s prioritized experience replay DQN [83] is described in Algorithm 3. It is very similar to Algorithm 1 and Algorithm 2, with the only differences being priorities for sampling transitions and IS for weights used in the Q-learning update.

The β-annealing function that was used in the experiments is given by

$$\beta(t) = \min \left[1, \beta_0 + \frac{1 - \beta_0}{\eta} t \right], \tag{3.6}$$

where β_0 is the initial IS coefficient, t is time steps and η is the annealing steps. As suggested by Schaul et al. [83], α was set to 0.6 and β_0 was set to 0.4 and annealed to 1.

Algorithm 4 presents a combination of noisy networks and categorical DQN. This algorithm corresponds to the ideas of Fortunato et al. [84] and Bellemare et al. [85]. Instead of denoting the local weights θ and the target weights ω, we now use ζ and ψ respectively. This is to indicate that the networks has noisy parameters. Another change-up is that this algorithm has no decay or annealing function of sorts. Actions are selected by a policy π that maps each state $s_t \in \mathcal{S}$ to a probabilistic distribution over the action space $\mathcal{A}$. The loss is also changed from MSE to cross-entropy. Rainbow DQN is a combination of all the mentioned algorithms.

Algorithm 3 - Prioritized Replay DQN

Initialize replay memory $\mathcal{E}$ with capacity C
Initialize first priority transition $p_1 = 1$
Initialize local weights θ_0 and set target weights ω_0 as copy
Generate meals, meal indicators and state $s_1 \sim \varsigma$
for time steps $t = 1, T$ **do**:
    Perform ε decay and choose action $a_t \sim \pi_\theta(s_t)$
    Execute a_t and receive new state s_{t+1} and reward r_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{E}$ with maximum
    priority $p_t = \max_{i<t} p_i$ and set $s_{t+1} = s_t$
    if episode is done **do**:
        Reset meals, meal indicators and state
    end
    if $\mathcal{E}$ is greater than the batch size B **do**:
        Perform β annealing
        Sample transition $j \sim P(j)$
        Compute IS weights $w_j = (CP(j))^{-\beta} / \max_i w_i$
        Sample mini-batch $(s_j, a_j, r_j, s_{j+1}) \sim \mathcal{E}$
        Define $a_{\max}(s'; \theta) = \arg\max_a Q(s', a'; \theta)$
        Set $y_j = \begin{cases} r_j & \text{for terminals} s_{j+1} \\ r_j + \gamma Q(s_{j+1}, a_{\max}(s_{j+1}; \theta_j); \omega_j) & \text{for non-terminal } s_{j+1} \end{cases}$
        Calculate loss $L_j(\theta_j) = w_j(y_j - Q(s_j, a_j; \theta_j))^2$
        Update priority transition $p_j \leftarrow |L_j(\theta_j)| + \delta$
        Perform gradient descent step on $L_i(\theta_i)$
    end
    Copy network weights $\omega \leftarrow \theta$ every τ steps
end

Algorithm 4 - Categorical DQN with Noisy Networks

Initialize replay memory $\mathcal{E}$ with capacity C
Initialize local weights ζ_0 and set target weights ψ_0 as copy
Generate meals, meal indicators and state $s_1 \sim \varsigma$
for time steps $t = 1, T$ **do**:
    Calculate $Q(s_t, a) = \sum_i z_i p_i(s_t, a; \zeta)$
    Choose action $a_t = \arg\max_a Q(s_t, a)$
    Execute a_t and receive new state s_{t+1} and reward r_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{E}$
    if episode is done **do**:
        Reset meals, meal indicators and state
    end
    if $\mathcal{E}$ is greater than the batch size B **do**:
        Sample mini-batch $(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{E}$
        Compute the next distribution
        $p_{j+1} = \sum_i z_i p_i(s_{t+1}, a; \psi)$ for $i \in 1, \ldots, N$
        Compute the projection $\mathcal{T}_{z_j} = [r_t + \gamma z_j]$
        onto the support $\{z_i\}$ for $i \in 1, \ldots, N$
        Define $b_j = (\mathcal{T}_{z_j} - V_{\min})/\Delta z$
        Define bounds $\ell = \lfloor b_j \rfloor$ and $u = \lceil b_j \rceil$
        Initialize projection distribution $m_i = 0$ for $i \in 1, \ldots, N$
        Distribute the probability of $\mathcal{T}_{z_j}$
        $m_\ell \leftarrow m_\ell + p_{j+1}(s_{t+1}, a_{\max})(u - b_j)$
        $m_u \leftarrow m_u + p_{j+1}(s_{t+1}, a_{\max})(b_j - \ell)$
        Calculate loss $L = -\sum_i m_i \log p_i(s_t, a_t; \zeta)$
        Perform gradient descent on L
        Reset noise parameters in ζ and ψ
    end
    Copy network weights $\psi \leftarrow \zeta$ every τ steps
end

3.2 Glucose-Insulin Dynamics Simulator

Having a T1D patient's glucoregulatory system in a simulator is important and beneficial. It can let researchers test on imitates of patients in a uninvolved manner and is safe. This is also crucial to able to trial the AP algorithms *in silico*[1]. Glucoregulatory system simulators is therefore of high interest to researchers, and have been well developed over the last decade.

The most common simulator of such was made by Man et al. (2014) [102] from the University of Virginia and the University of Padova, known as the UVA/PADOVA Type 1 Diabetes Simulator. This is a closed-loop simulation software implemented in Simulink/MATLAB [50] and is based on the glucoregulatory system model formulated by Man et al. (2007) [103].

The first version of the UVA/PADOVA was accepted by the Food and Drug Administration (FDA) in 2008 [104] and had shown to represent sufficient glucose fluctuations in T1D observed during meal challenges. It could also simulate CGM sensor errors and subcutaneous insulin delivery. A later update, that was accepted by FDA in 2013, was presented by Man et al. (2014) [103] and showed better performance in describing hypoglycemic events. Other features that were improved was glucagon kinetics and secretion and action models. In addition to this, a refined statistical strategy for virtual T1D patient generation.

An upgrade of the previous UVA/PADOVA simulator was presented by Visentin et al. (2018) [105] and introduced an extended domain of validity from "single-meal" to "single-day" scenarios. This would allow, for instance, the CGM to describe the nocturnal BG increase. Consequently, the simulator has a more realistic framework that reflects more on real data.

As the UVA/PADOVA simulator is not openly available, and there are other alternatives, the simulator of choice is a modified version of the Hovorka model. This is presented in great detail in Appendix A.1. The Hovorka model was implemented in *Python* in *OpenAI Gym*, which is a toolkit for developing and comparing RL algorithms [106]. The implementation was done by Myhre et al. (2018) [107] in order to use the simulator in the gym, and then to regulate the BG concentration of T1D patients using fitted Q-iterations. For this research, some of that implementation was modified in order to fit the needs of the experiments.

The simulator functions as follows. Every minute m, the simulated patient resides in a state $\boldsymbol{S}_m$ [50]. This is a vector containing all the variables declared in equations A.1 to A.14, and can be shown in Table A.1. The variable $Q_{1,m}$ [mmol] represents the glucose in the main blood stream. It is used to derive the the glucose concentration $y_m = G_m = Q_{1,m}/V_G$ [mmol/L], where V_G [L] is the glucose distribution volume. For more details regarding this please consult equation A.7 and section A.1.2.

[1] In silico is an expression meaning "performed on computer or via computer simulation", in reference to biological experiments [101]

After the patient obtains the state $\boldsymbol{S}_m$, the simulator then receives two inputs. One of them is the amount of insulin to be injected, u_m [mU/min], during the the interval $[m, m + dm]$. Here, $dm = 1\,\text{min}$ is a discretization step [50]. The other input is D_m [mmol/min], which is the amount of CHO intake during the same time period. The CHO input rate is defined in equation A.1. Accordingly, the simulator solves the differential equations A.3, A.5, A.10, A.12 and A.13, which in turn gives the patient a new state $\boldsymbol{S}_{m+dm}$. Now that the simulated patient is in this state, the CGM outputs a new glucose concentration G_{m+dm}.

As times goes by the BG data monitored by the CGM is growing. In our experiments, we used 30 minutes of BG data, as well as the 4 last insulin actions at time t to be the environment state $\boldsymbol{s}_t$. One time step is set to be 30 minutes, meaning that the 4 last insulin actions were from the last 2 hours. We use this in algorithms 1 - 4 to obtain the insulin doses (actions) to compensate for the current BG. Then we update parameters accordingly.

Chapter 4

Experiments and Analysis

The main goal with the following experiments is to compare different DQN extensions for the BG simulations, and see if they can maintain BG values within the time-in-range (TIR) [111]. Different action space sizes will also be compared, as this might affect the algorithms. TIR is the percentage of time a person spends with their BG levels in a target range. General guidelines suggest a range of 70-180 mg/dL [111]. The TIR goal will be explained in more detail shortly, but first we will review the meal schedules.

For each simulation and experiment, an individual who weighs 70 kg is used. The meal generation is as follows: 4 meals are generated each day with a set schedule, and lasts up to 1 minute. To make the meals more random and realistic we added an uniformly distributed noise $\upsilon \sim \mathcal{U}(-20, 20)$ to each base meal, as well as ± 30 minutes to each meal time at random. This noise υ is affecting the amount of CHO in a meal. The daily meal schedule is then:

- **Breakfast**: $(40 + \upsilon_1)$ [g] of CHO at around 8:00 (8 am). The CHO rate is then $D_B = 3.701$ [mmol/min]
- **Lunch**: $(80 + \upsilon_2)$ [g] of CHO at around 12:00 (12 am). The CHO rate is then $D_L = 7.401$ [mmol/min]
- **Dinner**: $(60 + \upsilon_3)$ [g] of CHO at around 18:00 (6 pm). The CHO rate is then $D_D = 5.551$ [mmol/min]
- **Supper**: $(30 + \upsilon_4)$ [g] of CHO at around 22:00 (10 pm). The CHO rate is then $D_S = 2.775$ [mmol/min]

Here there are four noise variables υ_1, υ_2, υ_3 and υ_4, one for each meal. The CHO rate is calculated by equation A.1 from appendix A.1 and the base meals are taken from El Fathi et al.'s work [112]. Each meal that is generated consists of a CHO intake and an

estimated CHO intake. The latter is often referred to as the meal indicator, which is the guessed CHO amount by the patient. This quantity is estimated with CHO counting errors done by the patient that works as a measurement uncertainty.

TIR gives a measure of how the BG levels of a person might vary over time. It can also be perceived as the amount of hours per day spent in-range. As an example, 50% TIR is equivalent to 12 hours per day spent in-range. Now, consider an increase from 50% TIR to 55% TIR. This 5% increase translates to one more hour per day spent in-range, which is a significant increase considering the small change in TIR.

Goals with TIR vary from person to person and may depend on the type of medication they use, type of diabetes, diet, health, age, and risk of hypoglycemia [111]. Generally, any patient that suffers from diabetes should spend as much TIR as possible. Studies show that the TIR for an average person with diabetes is around 50%-60% [111], which is approximately 12 - 14.4 hours per day spent in-range.

TIR goals for patients with T1D and T2D were recently published in by researchers [113], where they recommended aiming for the following [111]:

- At least 70% of the day in the range 70-180 mg/dL
- Less than 4% of the day below 70 mg/dL
- Minimize the time each day above 180 mg/dL

The above goals are what we try to achieve when using different DQN extensions to regulate BG levels in simulated T1D patients. To calculate the TIR, it is recommended to use at least 14 days' worth of BG data [111]. This is exactly what we do in the following experiments. The TIR is estimated using the following equation:

$$\text{TIR} = \frac{N_{\text{in-range}}}{N_{\text{total}}} \cdot 100, \tag{4.1}$$

where $N_{\text{in-range}}$ is the number of in-range BG data points and N_{total} is the number of total BG data points. We also define the metrics time-above-range (TAR), the percentage of time a person spends with their BG levels above the target range, and time-below-range (TBR), which is the percentage of time below the target range. These metrics will be used later in results as a way of explaining how high or low the BG curve went over time. Also, they can be used to study how poorly certain algorithms did compared to others.

4.1 Experimental Setup

In this section we present the experiments, network architectures used in them, and how we tested the algorithms. The goal is to successfully regulate the BG values in a simulated T1D patient, while still meeting the TIR criteria. In order to achieve this, three experiments were conducted.

The first experiment aims at comparing all the DQN algorithms mentioned in section 3.1.5, while still using the same hyperparameters, training duration and batch size. For the second experiment, the same algorithms were used, but now with a larger action space. Here we attempt to explore what more actions can have an effect on the RL agent. Lastly, the third experiment was organized in order to test how well a trained agent would perform when skipping meal boluses at random. In the subsections to come, each experiment and their setups will be described in more detail.

Algorithms and neural network implementations were done in *Python* 3.8.1 using *PyTorch* 1.4 [108]. The full code is available at my Master's thesis repository[1], and the gym implementation can be viewed at my Gym repository[2]. As mentioned earlier in Section 3.2, the gym implementation was forked from Jonas N. Myhre's repository. Some algorithms were tested on both own implementations and *OpenAI Baselines* [109] implementations, which is a set of high-quality implementations of RL algorithms. This was done because some results came out better using one implementation over the other.

All data results were exported from *Python* into *Julia* [110] for data analysis and plotting.

4.1.1 Experiment 1 - Comparing Algorithms

In this experiment we compare all the DQN algorithms reviewed in Algorithms 1-4. The main goal here is to see which algorithm achieves the best TIR score. The comparison procedure is as follows:

- Plot a test episode and see how the BG curve evolves and what actions the agent performs
- Plot the learning curve and observe how the return evolves with episodes
- Plot the mean BG per minute over 100 succeeding episodes and calculate the TIR, mean BG per episode and standard variation BG per episode
- Calculate the mean TIR of 14 succeeding episodes

[1] `https://github.com/sigurdhjerde/Masters-Thesis`
[2] `https://github.com/sigurdhjerde/gym/tree/master_student_branch`

The first two points are more for visual analysis/inspection and to see what actions the trained agent selects, as well as how the learning process evolves over the training period. The last two points are tests for checking how well an agent does over many episodes. To calculate the mean BG per minute over 100 episodes allow us to accurately estimate the TIR over this period. On the other hand, the mean BG per episode and the variance in BG per episode is a way to look at the global mean and variance of the BG data over 100 episodes. Lastly, the mean TIR of 14 episodes is simply a mean value for the TIR instead of a TIR estimate of mean BG data. 14 episodes was selected since it was the suggested on the diaTribe website, as the minimum amount of days required to determine the TIR [111].

We used the state space as described in section 3.1.1 and we used the action space with 3 actions. The models were trained for 10^5 time steps (epochs), with a batch size of 128, an experience replay buffer size of 10^5, and a discount factor of $\gamma = 0.99$. Decaying ε-greedy exploration was used during training, according to equation 3.5, where the initial value was set to $\varepsilon_0 = 1.0$, the final value $\varepsilon_F = 0.01$ and the decay $\eta = 3 \cdot 10^4$. The exploration curve can be seen in figure 4.1, and is approximately equivalent to 50% exploration during training.

The MSE training loss of the TD errors was optimized using Adam, with a learning rate of 10^{-3}. Our neural network weights were initialized using *PyTorch* default settings. Four networks were used in experiment 1, and now we will describe these architectures and assign algorithms to them.

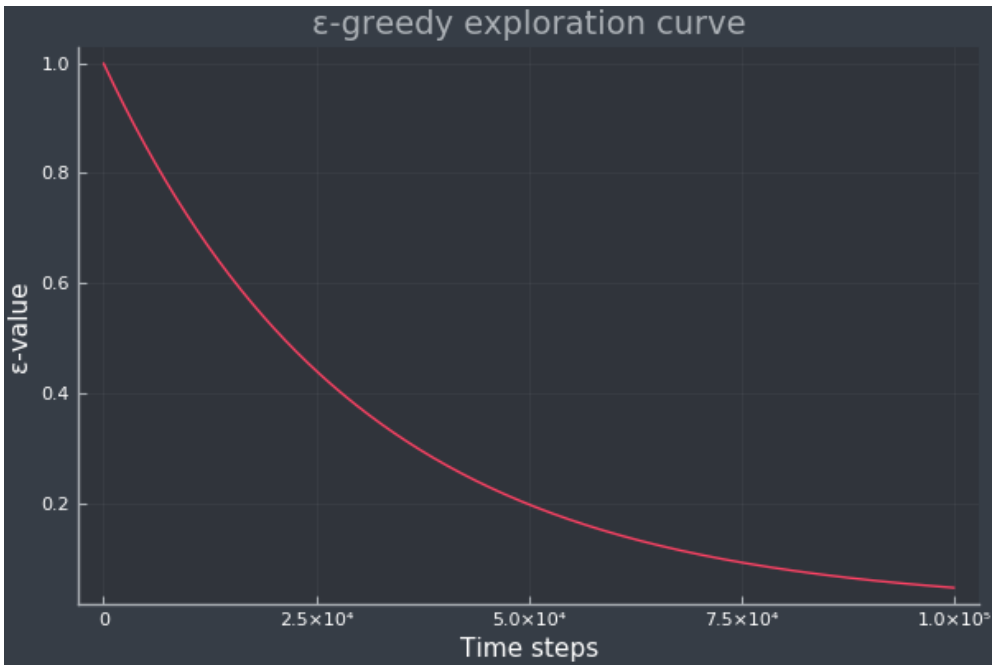

Figure 4.1: The ε-greedy exploration curve for all experiments. The ε-value shows the percentage of exploration at the current time step.

DQN

A 4 layer fully connected network with 64 hidden units each. ReLU nonlinearity was used across all layers. The output layer has a linear output. This network was used for the DQN and DDQN algorithm.

Dueling DQN

A fully connected network consisting of two blocks, each with a dense layer of 4 fully connected layers. The amount of hidden units is 64, and ReLU nonlinearity was used across all layers. Each output layer in the two blocks have linear outputs, representing the advantage and value streams. This network was used for the dueling DQN, prioritized replay DQN and noisy DQN algorithm. For the last mentioned algorithm, we simply added noise to the linear layers and reset the noise parameters after every training batch.

Categorical DQN

A 4 layer fully connected network with 64 hidden units each. The two first layers are without noise and the following is with noise. ReLU nonlinearity was used across all layers. The output layer has a linear output. This network was only used for the categorical DQN algorithm.

Rainbow DQN

A fully connected network consisting of two blocks, each with a dense layer of 3 fully connected noisy layers. The input layer consists of an additional linear layer before splitting into the two streams. The amount of hidden units is 64, and ReLU nonlinearity was used across all layers. Each output layer is linear and represents the advantage and value stream as in the dueling DQN case. This network was only used for the rainbow DQN algorithm.

4.1.2 Experiment 2 - Expanded Action Space

The goal of this experiment is to see what influence an increase in the action space has on the agent and the training. By increasing the action space, one would assume that learning the right action at a certain state could be more difficult. If learned right, more actions could prove to be more efficient for the agent and the BG regulation as there are more choices in insulin amounts. Similar to experiment 1, we compare all the DQN algorithms using the same procedure and metrics.

We used the same state space as in experiment 1, but the action space had now 5 actions in it, as described in section 3.1.1. The only hyperparameter that was changed from experiment 1 was the batch size of 512. The increase in batch size is to compensate for the fact that we could need more data, since more actions could complicate the learning process.

Our neural network weights were once again initialized using PyTorch default settings, and the same network architectures from experiment 1 were used in this experiment.

4.1.3 Experiment 3 - Meal Disturbances

Based on the results from experiment 1, a final experiment was conducted, namely to skip meal boluses at random for an already trained agent. The goal of this test is to gain a deeper understanding of which algorithm performs best when the meal schedule is more unstable.

After training the agent using an algorithm of choice, a similar comparison to that of experiment 1 and 2 would be done, only now we looked at the TIR of the mean BG per minute, as well as estimating the mean BG per episode and BG variance per episode. The BG data was obtained from 100 simulated episodes, in which meal boluses were skipped with a 10% probability. One last comparison was to calculate the mean action per episode and action variance per episode, i.e., the global mean insulin and global insulin variance.

The compared agents are trained according to the experimental setup from experiment 1. All the meals and meal schedules were generated with a set seed, so that the comparison would be honest.

4.2 Results and Analysis

A baseline BG curve is presented in figure 4.2, where only the optimal basal rate $b^* = 6.43$ mU/min was selected as the action. We will refer to this action as either action 1 in experiment, and action 2 in experiment, depending on the action space being used. The curve presents the mean BG per minute of 100 episodes, and the shaded area is the BG variance per minute. This baseline will serve as a guideline when comparing trained RL agents for different DQN extensions in experiments 1 and 2. The BG curve in figure 4.3 presents the same data as the previous curve, only now with meal bolus skips taken into account. These two BG curves are very similar, only noticeable difference is that figure 4.3 has larger confidence intervals. As stated in the previous section for experiment 3, the BG data is the mean BG per minute of 100 episodes and meal boluses are skipped with a 10% probability. This curve will serve as a baseline for the results in experiment 3.

We will observe the actions selected by a trained agent from an arbitrary test episode, and then discern if the mean BG curve is regulated better than the baseline. The TIR of the baseline curve is 95.41%, with the TAR being 4.59%. The global mean BG and standard deviation was estimated as $\mu^* = 124.00$ mg/dL and $\sigma^* = 33.84$ mg/dL respectively. These were calculated by taking the mean of the episodic mean BG and the episodic BG standard deviation. The baseline results will be taken into account when analyzing the results from experiments 1-3.

Figure 4.2: A baseline BG curve using only the optimal basal rate $b^* = 6.43$ mU/min as the selected action. The curve is the mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

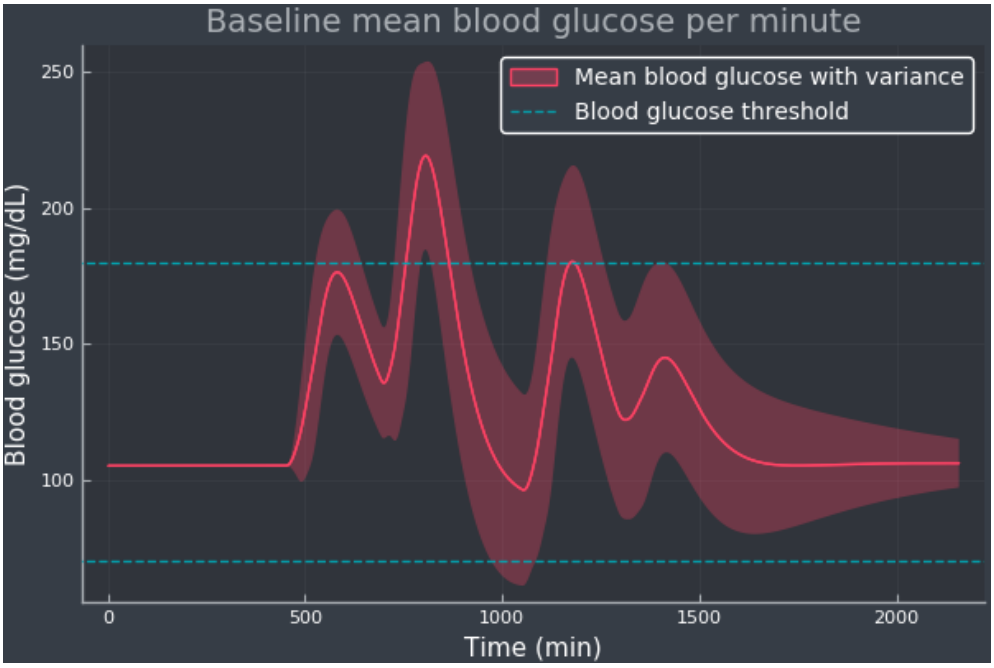

Figure 4.3: A baseline BG curve with meal disturbances using only the optimal basal rate $b^* = 6.43$ mU/min as the selected action. The curve is the mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

4.2.1 Experiment 1 - Comparing Algorithms

Starting with the most basic model, the DQN algorithm, we can see from the test episode in figure 4.4 that the actions taken are exactly the same as that of the baseline. When the agent only selects action 1, there seems to be stable BG values in the patient except for near the lunch meal, where there is a hyperglycemic event. Of course, this is only for one episode, but if we were to look at 100 subsequent episodes, then this hyperglycemic event near lunch time would happen every time. We can see in figure 4.5 that this assumption is true. Here the mean BG per minute is approximately equal to the baseline curve. If the meals in all 100 episodes were not somewhat randomized, and used a seed, then the obtained BG data would be identical to that of the baseline.

Looking at the results in table 4.1, we can confirm that the DQN algorithm is very similar to the baseline. The standard deviation of the BG per episode σ is slightly higher for DQN and scored the worst out of all the models, meaning that for some episodes the patient received larger meals at certain meal times, and the agent only selecting action 1 isn't always going to compensate the best for that. Comparing the σ values we see that they do not vary on a high scale. This can be perceived as that for each episode, the variation in BG do not differ that much from model to model. The TIR for the DQN agent is at 95.41%, while the baseline has 95.41%. Again, these results are so similar that we can practically say they are equals.

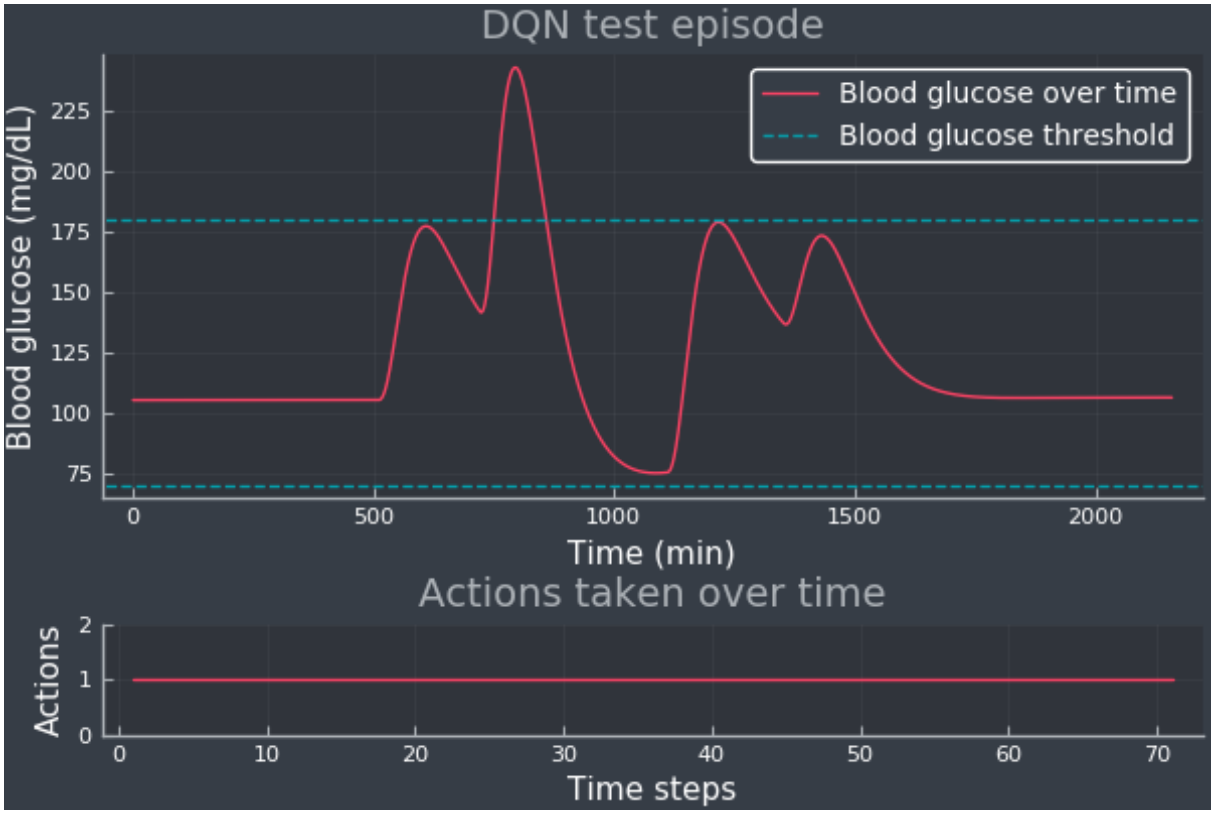

Figure 4.4: A DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

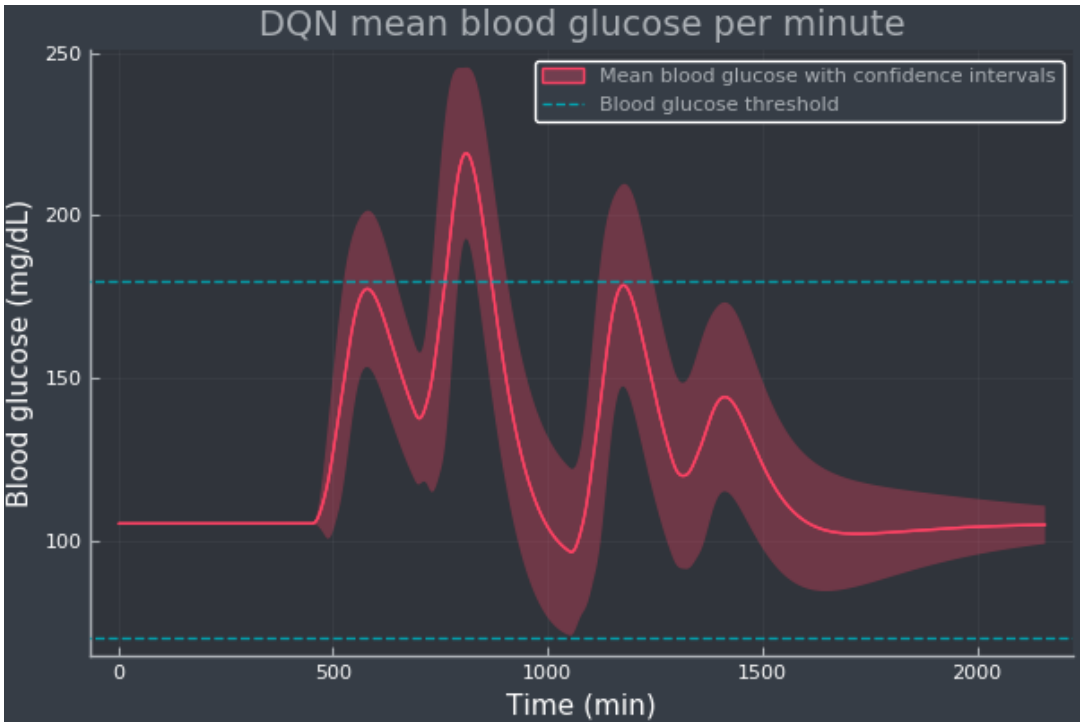

Figure 4.5: DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

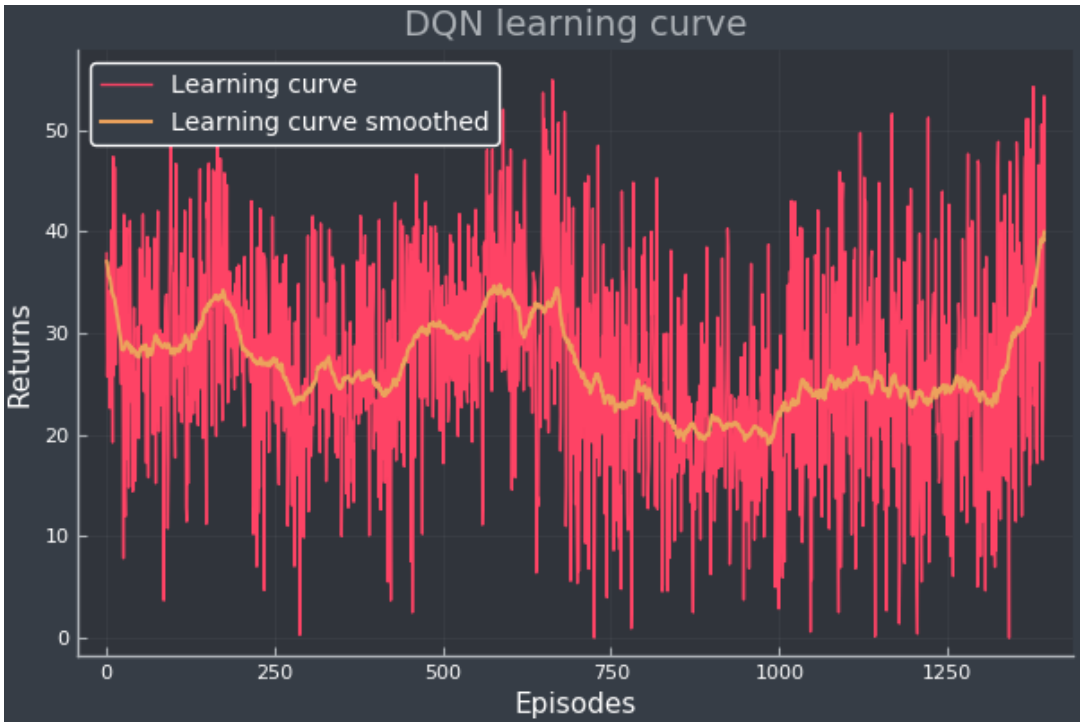

Figure 4.6: The DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

The DQN learning curve in figure 4.6 displays the evolution of return during the training period. At first glance it seems like the agent does not learn that much, except from the first few iterations and near the end. Knowing that the agent only performs action 1 after learning, it settled on this action early on in the training and the exploration was therefore limited.

Moving ahead with DDQN, we can see in figure 4.7 and figure 4.9 that the agent has learned to take different actions, and not just action 1. In the learning curve we can see that the return increases the first 200 episodes and has a sharp increase in the final episodes. Besides that this curve remains virtually flat. Analyzing the test episode and the mean BG curve in figure 4.8, we see that the TAR is very low, actually lower than the baseline. This is because the agent tends to choose higher basal rate actions, as well to stop the pump to compensate. As a consequence though, the TBR is higher and mean BG per episode is $\mu = 111.67$ mg/dL. The standard deviation in BG per episode is approximately the same as for the baseline, which makes sense because the BG spike near lunch has decreased resulting in a dip in BG levels between lunch and dinner time.

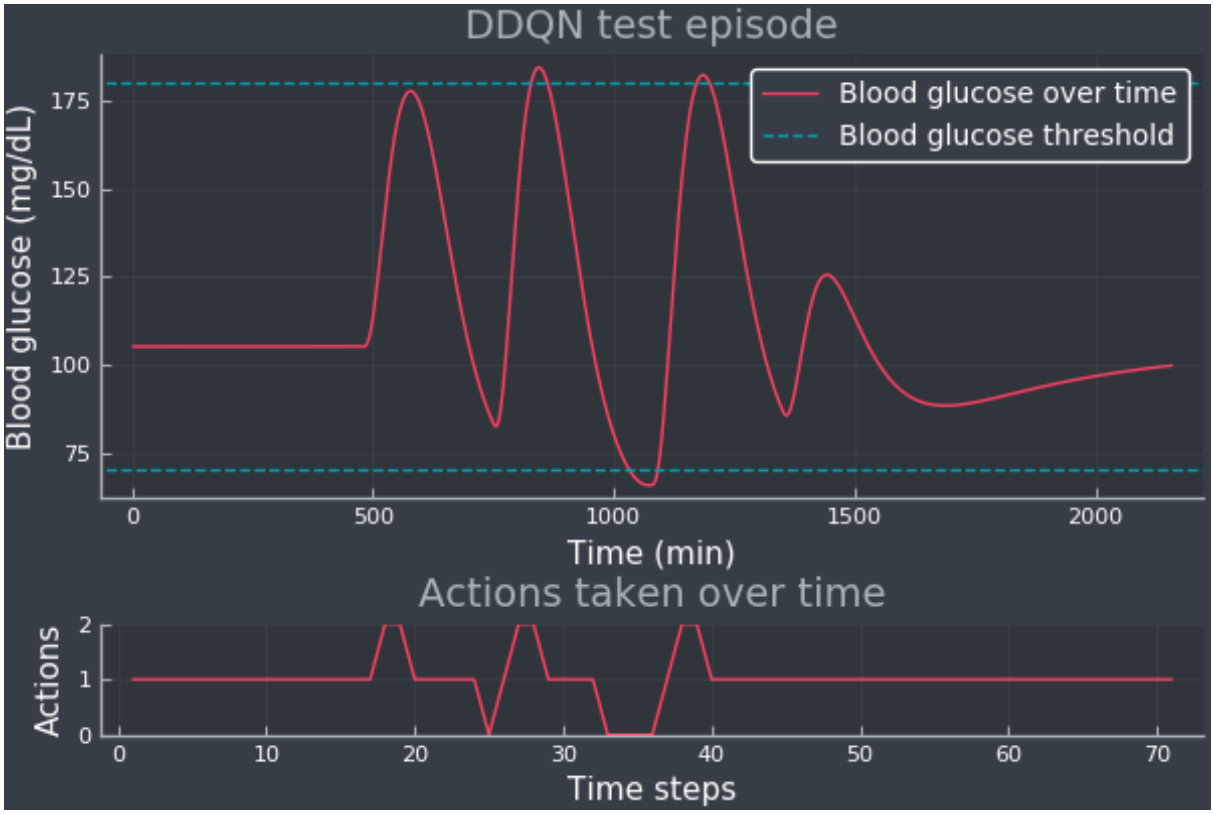

Figure 4.7: A DDQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

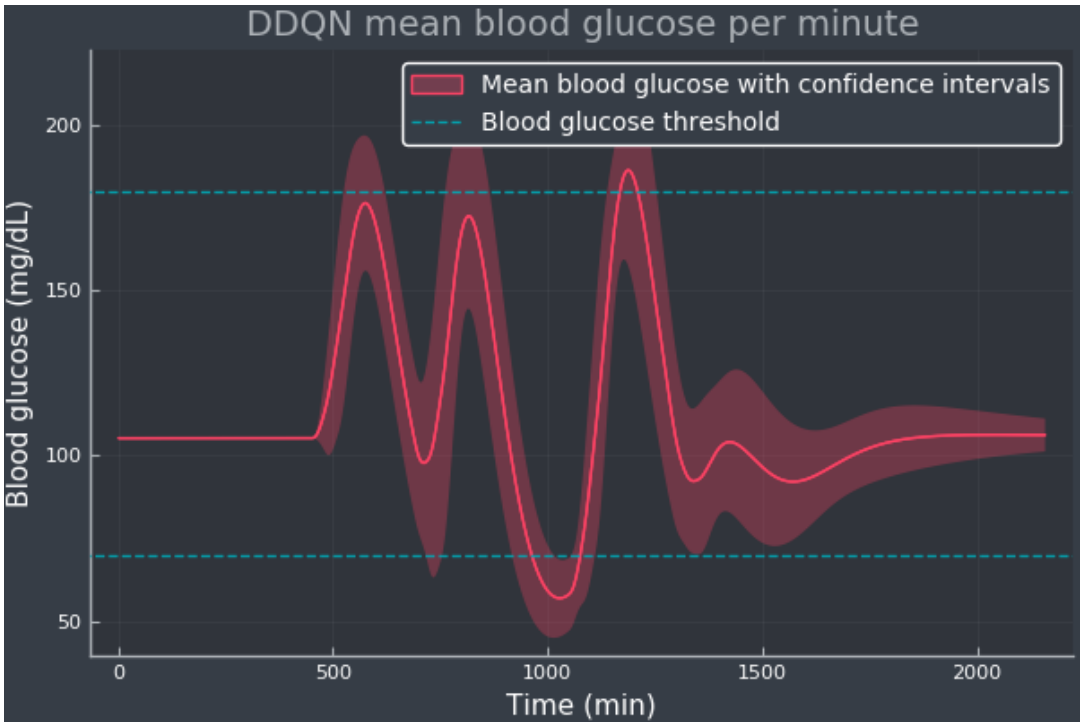

Figure 4.8: DDQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

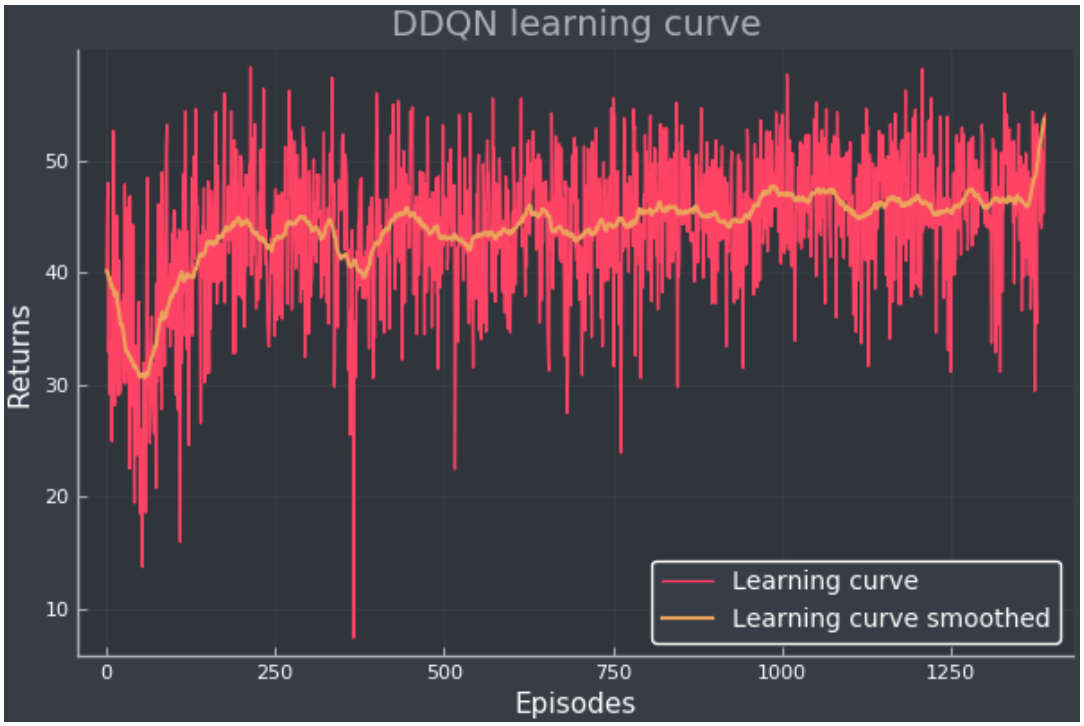

Figure 4.9: The DDQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Overall, the DDQN agent seems to learn more than the DQN agent, but fails to control the BG levels better. The TBR estimate is 5.28% and the mean TBR estimate is 7.22%, which does not meet the TIR criteria, where the TBR should be greater than 4%. In the end, the DDQN agent has the lowest TIR score of 92.82%, according to our specified criteria.

The dueling DQN agent has a learning curve that looks ideal. In figure 4.12 we observe that the return starts low and gradually increases until episode 1400 to around 55. The agent has clearly learned more than the previous agents, and we can confirm that by looking at the test episode in figure 4.10. The action selection is even more varied than previously and we can see that the BG curve is similar to the baseline curve. Also notice that the four BG peaks are lowered by a small amount. This is a consequence of the agent more frequently selecting the action 2, which is three times the optimal basal rate.

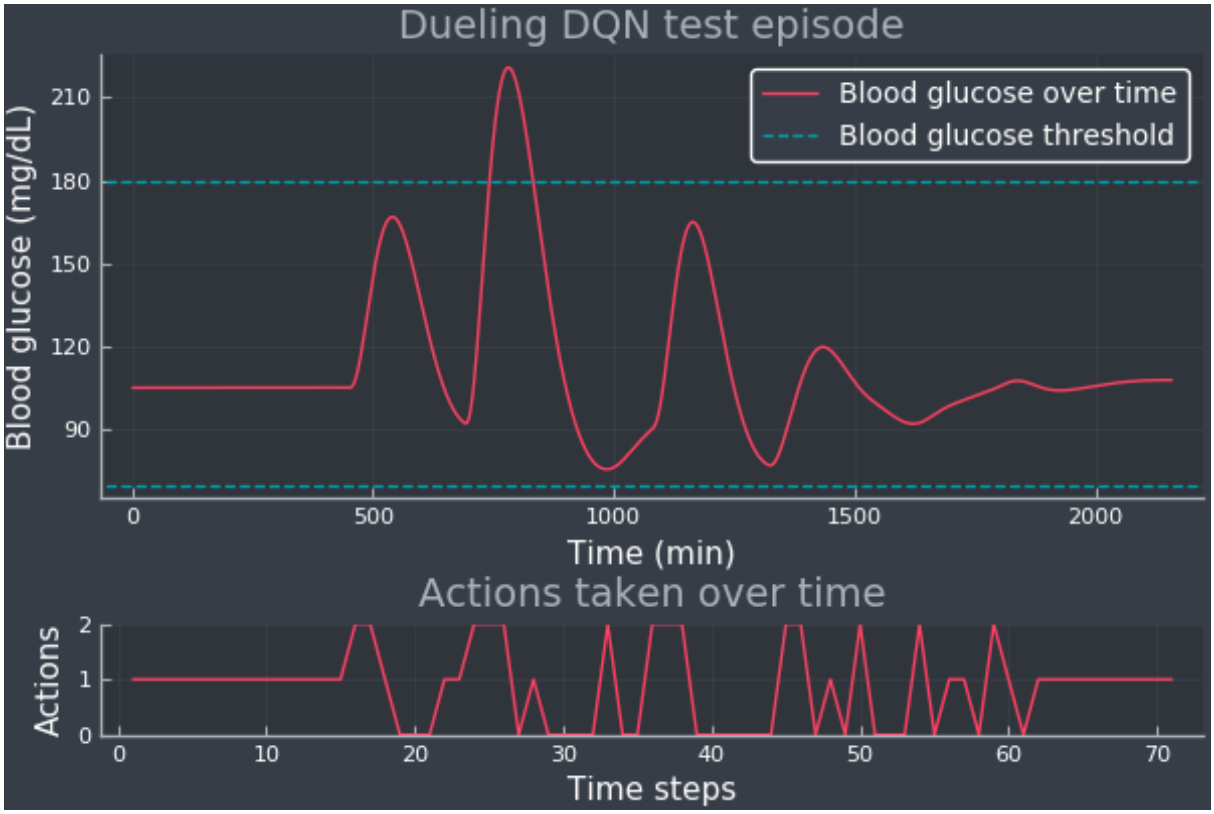

Figure 4.10: A dueling DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

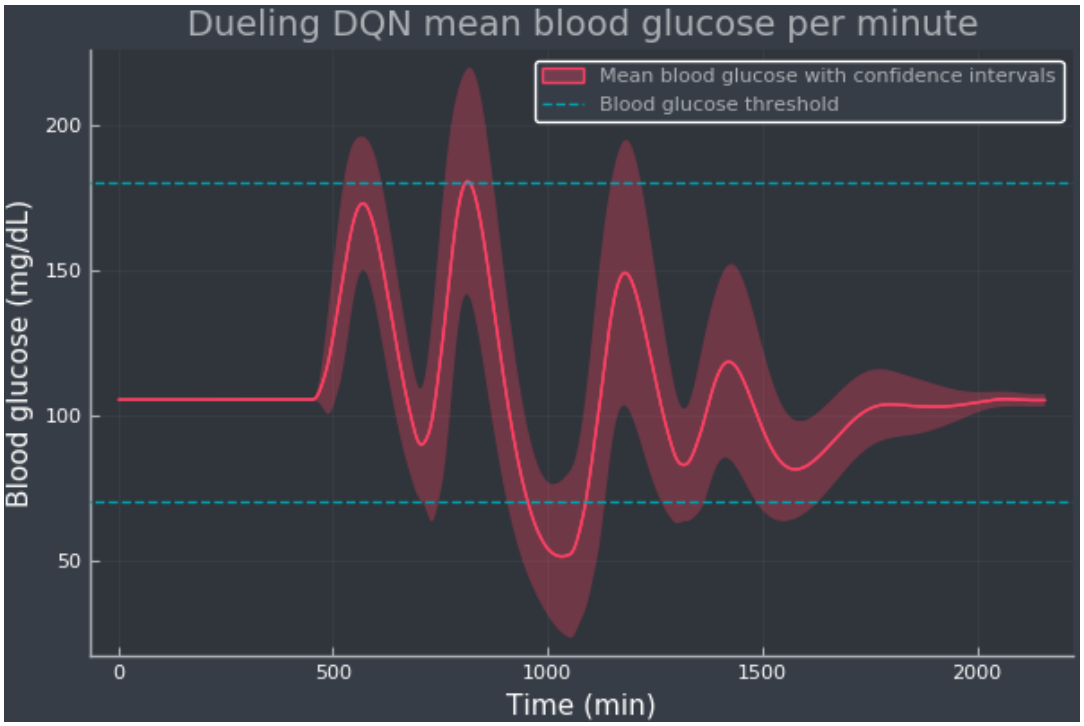

Figure 4.11: Dueling DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

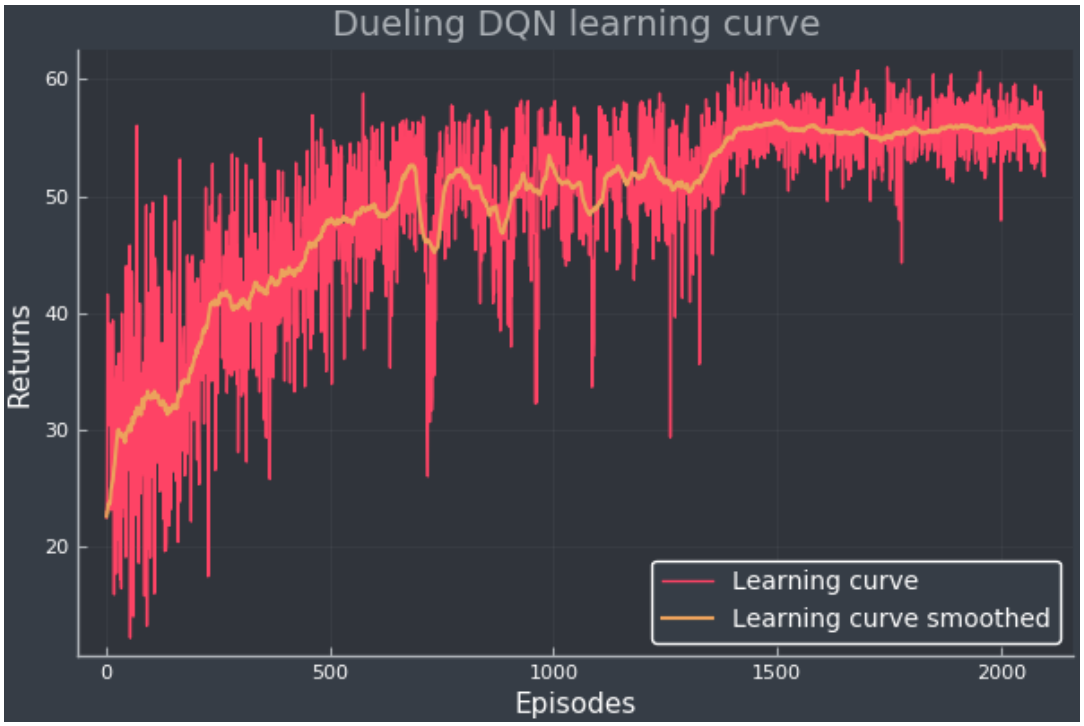

Figure 4.12: The dueling DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

The mean BG curve in figure 4.11 displays that over many episodes, the dip in BG levels after lunch is evident. Similarly to DDQN, this agent is controlling the other meal intakes just fine. In table 4.1 and table 4.2 we observe slightly higher TIR estimates, by 1% for the TIR of the mean BG curve and 3% for the mean TIR. The μ estimate tells us that the overall BG is higher per episode, but the TBR estimate is lower. Dueling DQN scored the lowest on the TBR estimate with a percentage of 6.25%. The particular episode in figure 4.10 is similar to the baseline, where there is no BG levels going below the lower bounds. The mean BG curve demonstrates that in the long-run, the agent's choice of actions will tend to lower BG levels too much near the lunch meal. However, the mean TAR is higher (7.19%) and the mean TBR is lower (1.11%) compared to the equivalent estimates from table 4.1. It is therefore evident that the episodic TAR score will be higher, and the episodic TBR will be lower.

For the dueling DQN agent we can temporarily deduce that it learns much more than the previous agents, but the experience it acquired is resulting in hypoglycemic events in the long-run for the simulated patient.

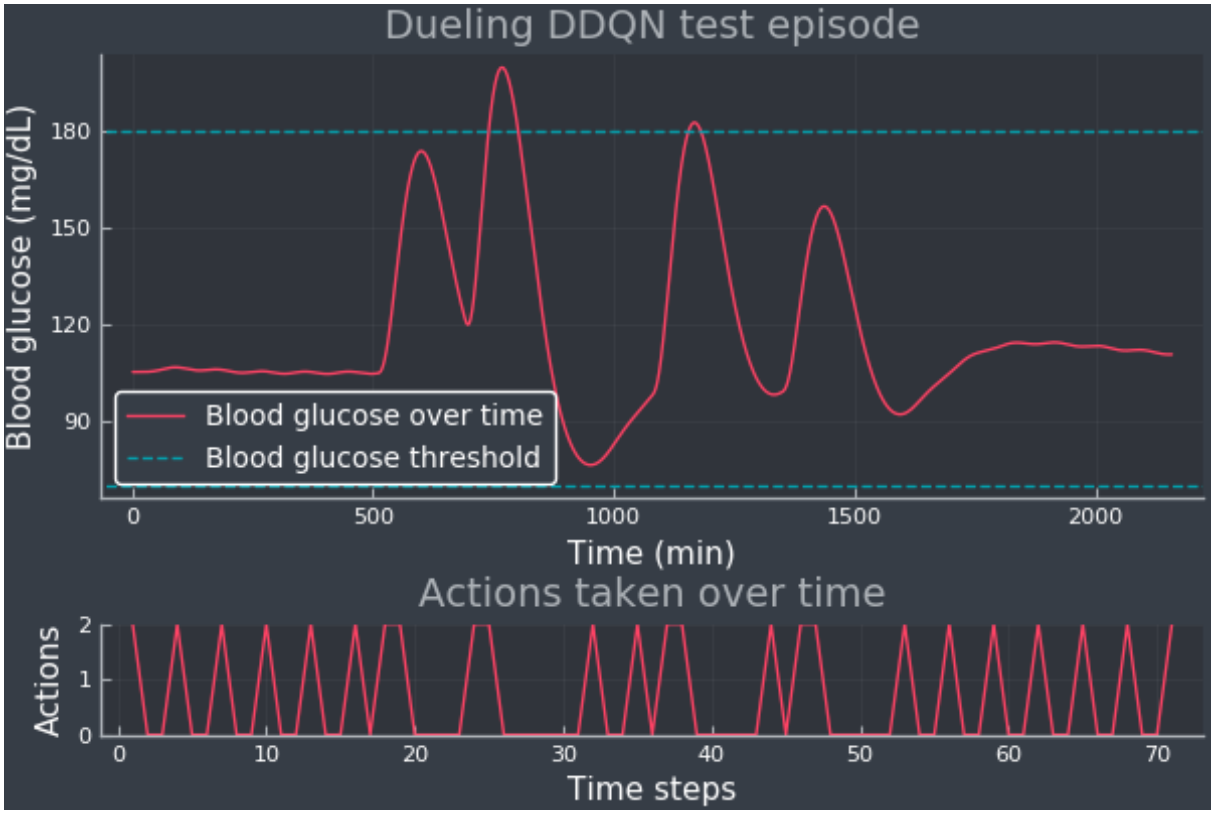

Figure 4.13: A dueling DDQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

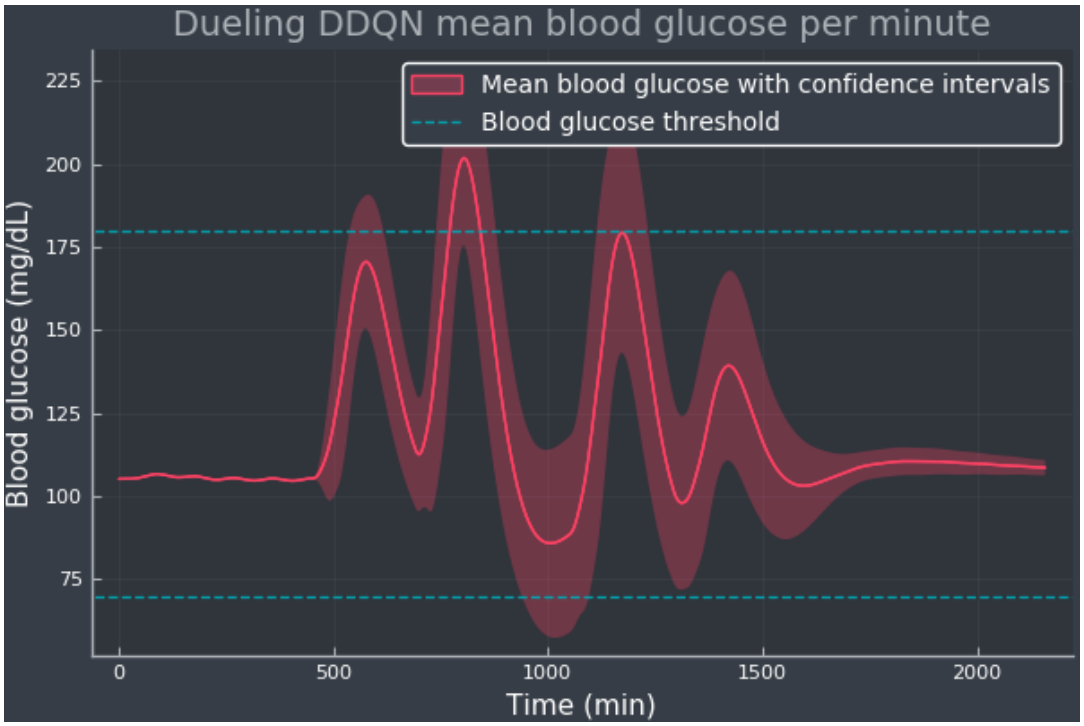

Figure 4.14: Dueling DDQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

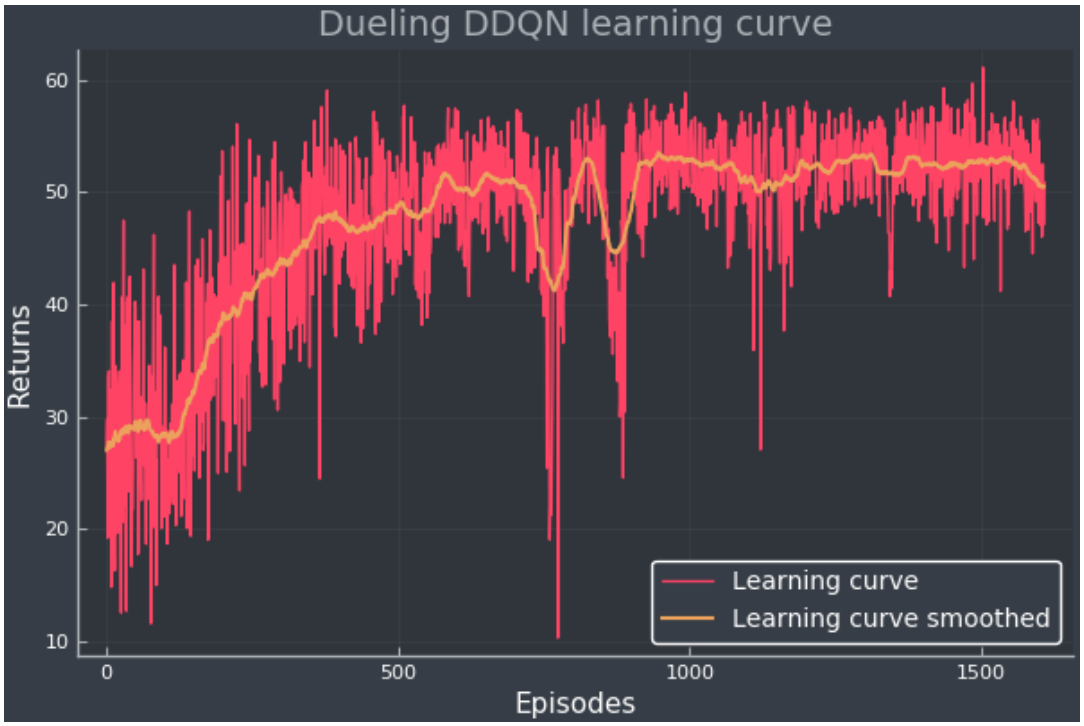

Figure 4.15: The dueling DDQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Turning over to dueling DDQN, we have a similar learning curve in figure 4.15. There are two dips in the learning process which symbolizes that the agent made some poor choices leading to less cumulative reward over the trajectory. The test episode in figure 4.13 displays a sawtooth-like pattern in the action selection. An episodic BG curve like this one is similar to the baseline curve. Also notice that this BG curve has some noise in the beginning, before the first meal. This phenomenon occurred occasionally even though the initial state was reset manually to the same state every time. This occurrence is within TIR threshold and has no effect on the first meal, since the agent is usually giving some basal amount to the patient. On behalf of this, the reader will find similar artifacts in test episodes of prioritized replay DQN and rainbow DQN.

Another point to investigate here is the mean BG curve in figure 4.14, which appears to be controlled better than the regular dueling DQN. With a TIR estimate of 96.71%, which is higher than the baseline, and 0% TBR, this algorithm has proved to be efficient at controlling the BG concentration in the patient. The mean TIR, TAR and TBR are all similar to the results of the previous algorithm, which makes it hard to compare using these metrics.

Overall the synergy between double Q-learning and dueling network has proved to be useful when managing the BG levels in a simulated patient.

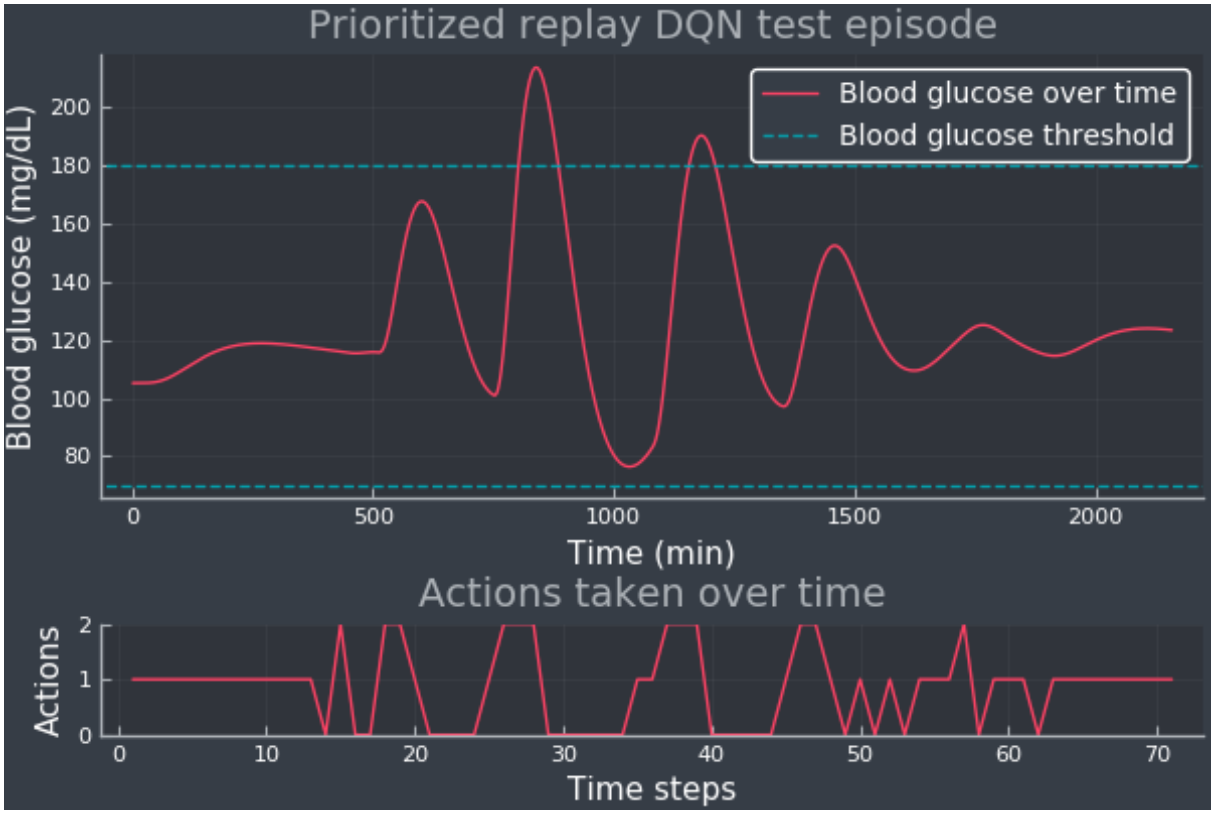

Figure 4.16: A prioritized replay DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

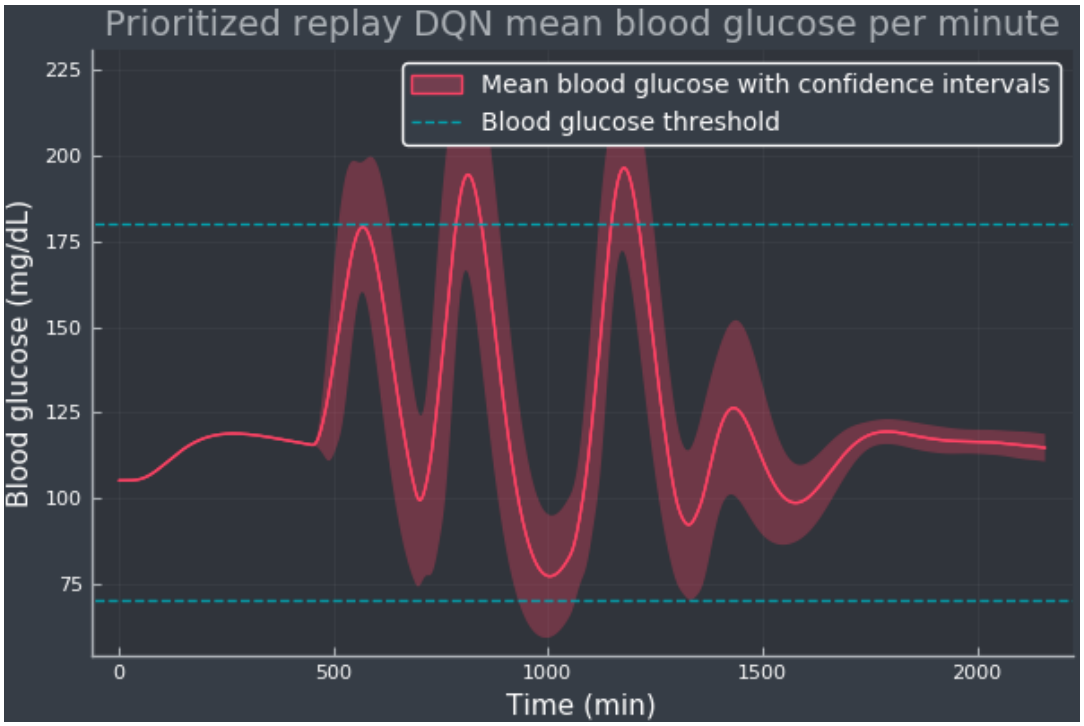

Figure 4.17: Prioritized replay DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

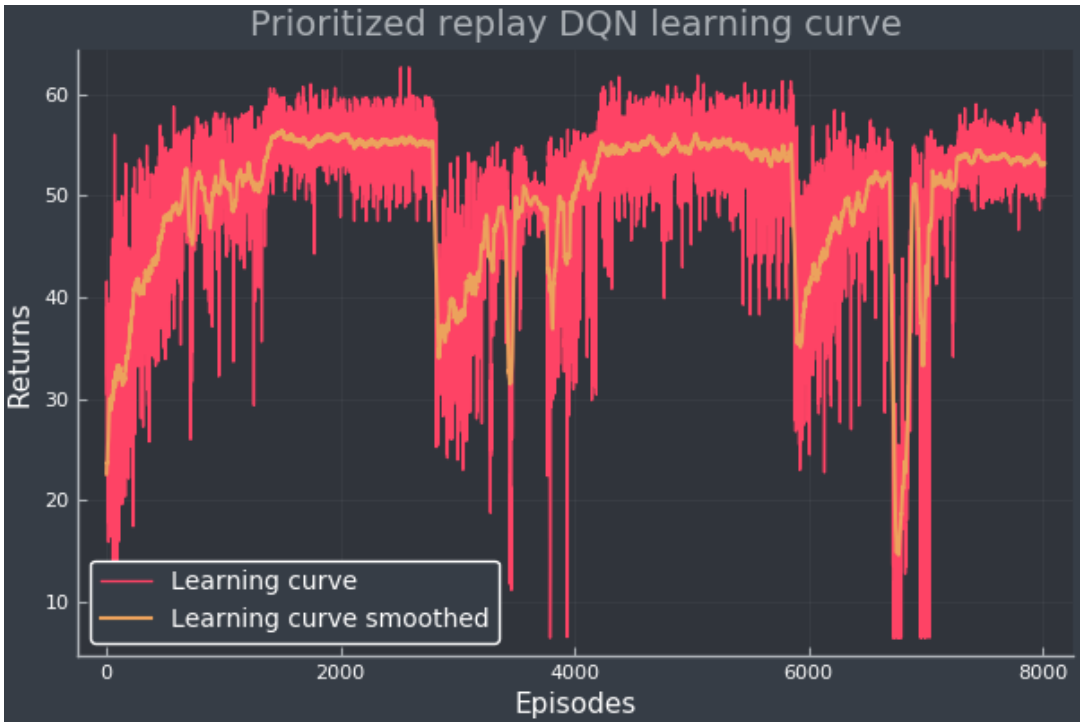

Figure 4.18: The prioritized replay DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Continuing on with the prioritized replay DQN (PR DQN) agent, there are some interesting results. Firstly, the BG curves in figure 4.16 and figure 4.17 show similar action selection to dueling DQN. However, the agent fails at controlling the dinner meal intake, which was the same case for the DDQN agent. For many time steps the agent turns off the basal pump and turns it back on whenever it registers an increase in BG levels. It seems like the agent is underestimating how much basal rate it should give the patient at times. The results in table 4.1 and table 4.2 validates these claims. The TAR estimate had the worst score of 5.65%, and a mean TAR of 8.48%, which is a bit higher than the baseline.

The most interesting point to investigate now is the phenomenon in this agent's learning curve. In figure 4.18, it appears that the agent is learning a bit until around episode 3000. Near this point in the training we can observe a huge decline in learning. Then the agent picks up the pace again and receives higher returns until the same situation occurs again. This incident appears to be the case of catastrophic interference, or catastrophic forgetting [114, 115, 116], which is the tendency to completely and abruptly forget previously learned information upon learning new information. In this case, the network trains well at first, but then it appears to overfit/forget what it has learned.

The reason for the catastrophic forgetting is worth examining. α, the prioritisation parameter, was held fixed at 0.6, and β, the IS parameter, was annealed from 0.4 to 1. This was the sweet-spot suggested by Schaul et al. [83]. The IS in this context, is meant to correct for the bias in high.priority samples, while leaving the low-priority samples unchanged. This is because samples with high priority are likely to be used in training more often. With this reasoning, using IS, the high-priority samples will indicate the network to train on them, but with much less emphasis. Contrariwise with low-priority samples, the IS weights will tell the network that there is not much to learn from them, since the TD error is low. The agent forgets previously learned information because it receives the same samples from the buffer over and over again. To fix this problem of preserving sufficient diversity and recycling, one possibility could be to try out other sweet-spots for α and β. Also, α could be annealed as well, leading to a even more aggressive prioritization sampling, while at the same time more strongly correcting the importance weights.

Schaul et al. [83] proposes a hybrid approach where each mini-batch are sampled according to one priority measure, and the rest according to another one. This introduces more diversity in the sampling and could prevent overfitting, premature convergence or poor representations.

The noisy DQN agent has proved to have some of the more robust results, with a TIR score of 97.04% being the highest compared to the rest. It also got the lowest standard deviation BG per episode of $\sigma = 31.74$ mg/dL. The μ estimate of this algorithm was marked in bold in table 4.1, because it is trusted to be the best result compared to the rest. The usual normoglycemic range is between 3.9-7.1 mmol/L, or 70-130 mg/dL [117], in which the BG reference was chosen to be 108 mg/dL (6.0 mmol/L). In table 4.1 there are three μ estimates that are the closest to the BG reference. The first one for DDQN (111.67) mg/dL, the second for noisy DQN (116.10) mg/dL, and the last

for rainbow DQN (100.66) mg/dL. We know from earlier that the DDQN agent scored poorly in TBR and mean TBR, so it's safe to assume that this μ estimate is not the best since the episodic BG levels tends to be low between lunch and dinner meals. Rainbow DQN, which we will come back to later, also scored poorly in TBR and mean TBR, which implies the same scenario here. The noisy DQN agent on the other hand, has low percentage in TBR and mean TBR estimates. It scores better than the baseline in all aspects, which signifies that the episodic BG levels are lower than the baseline's, but not in threatening manner that ultimately leads to hypoglycemic events. In that regard, a mean BG per episode of 116.10 mg/dL seems to be ideal for episodic BG levels.

We observe from figure 4.19 that the agent mostly picks action 1, and occasionally selects action 0 (no basal rate) or action 2 (three times the optimal basal rate). This test episode is similar to the one from dueling DQN, which is an overall lowered version of the baseline curve. The mean BG curve in figure 4.20 looks like an improved version of the same curve obtained using the PR DQN agent. This BG curve is very ideal for a T1D patient, since the TAR is very small in the long-run, and ultimately, there are no hypoglycemic events. The learning curve in figure 4.21 is also ideal, seeing that the agent is learning gradually and increasingly, much like dueling DQN.

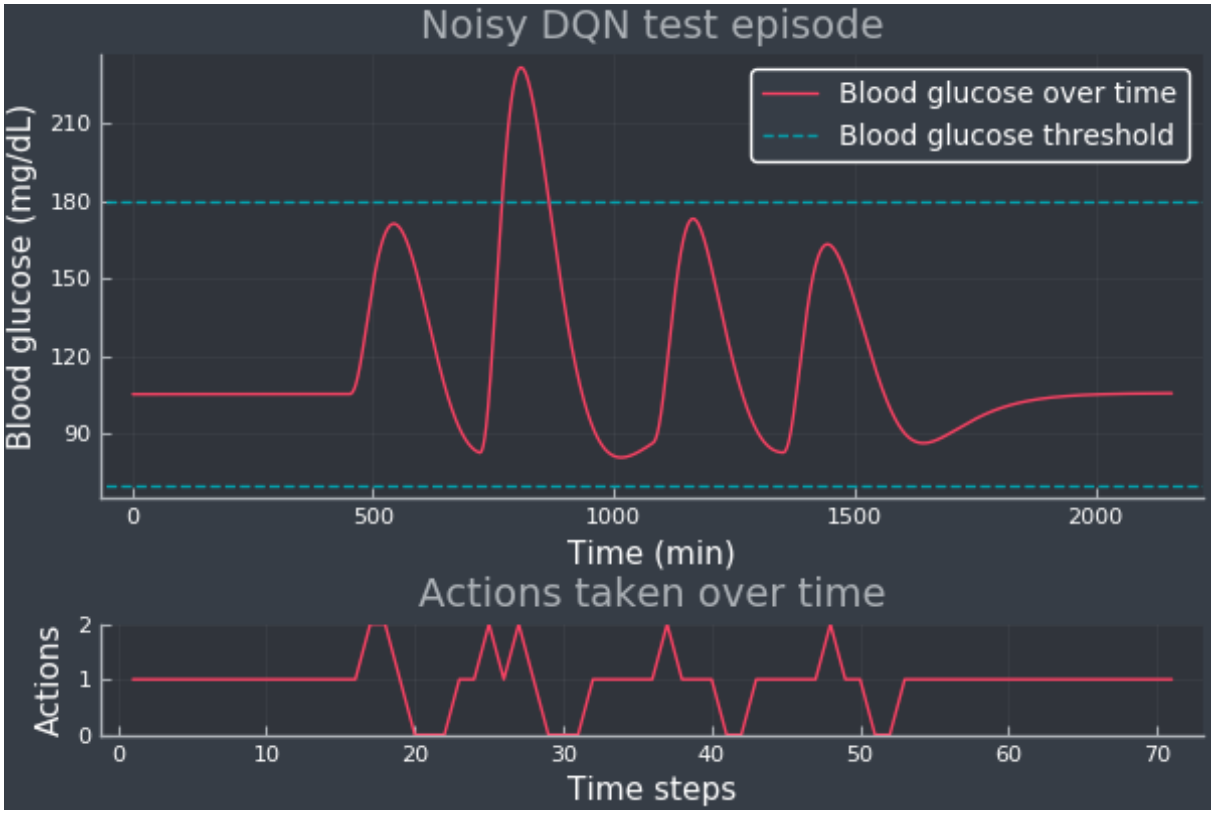

Figure 4.19: A noisy DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

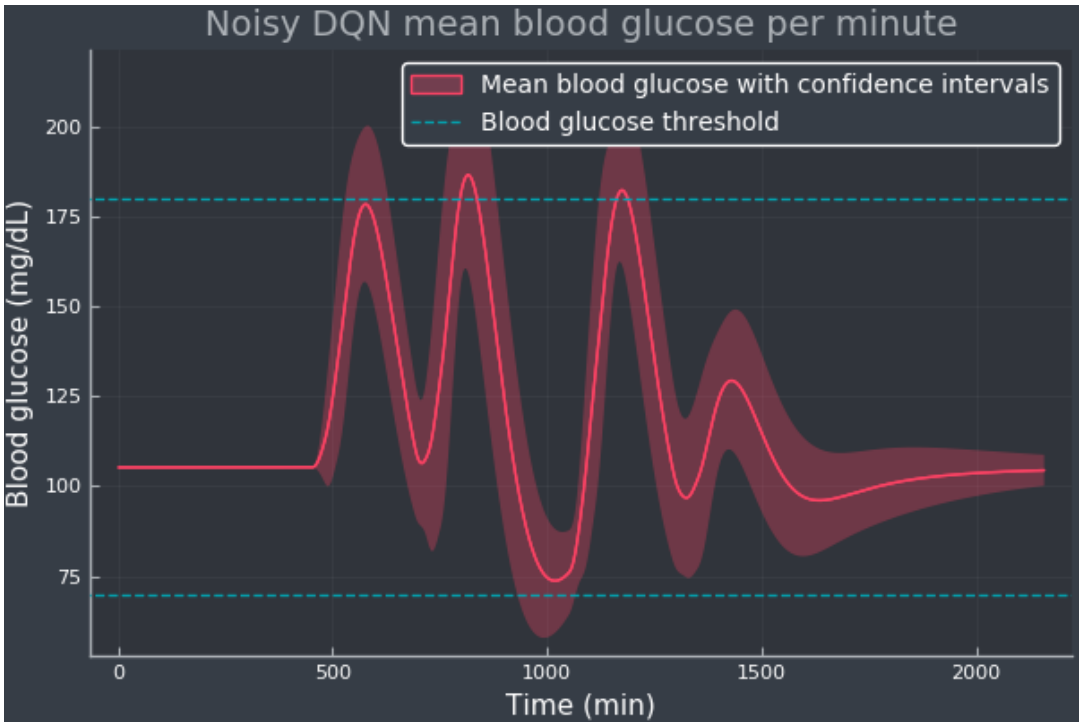

Figure 4.20: Noisy DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

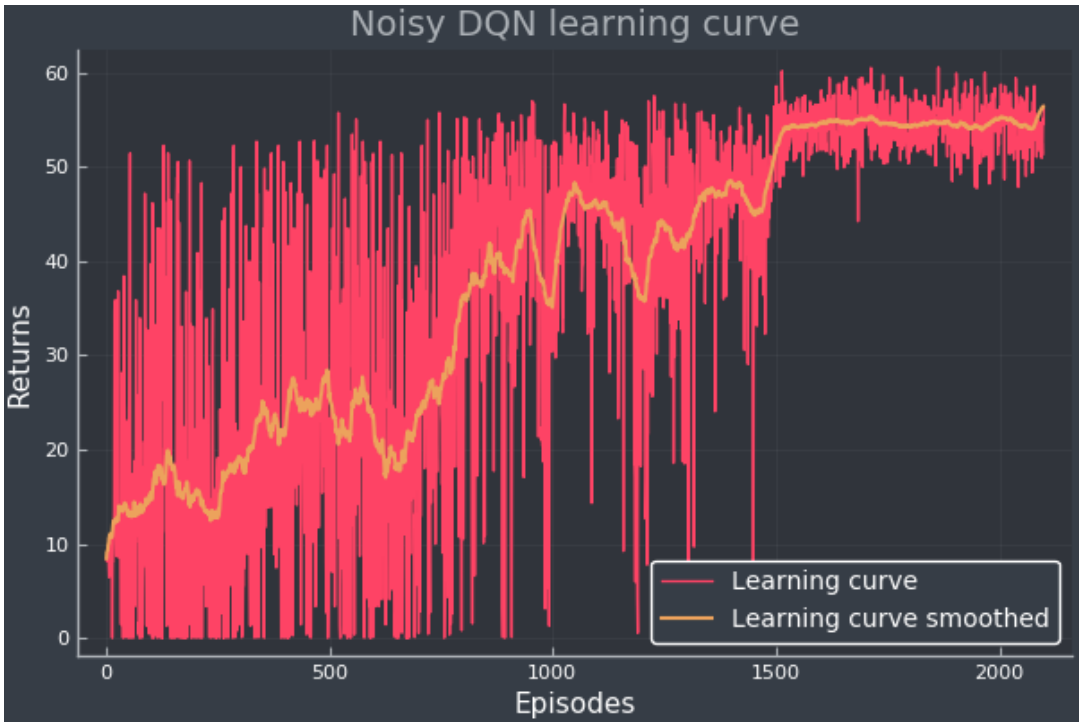

Figure 4.21: The noisy DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

We can see that the categorical DQN agent has learned similar strategy as the regular DQN, which is only giving the optimal basal rate to the patient. figure 4.22 gives an example of how an episode in this setting would like and we can agree that this is on par with the baseline curve. To strengthen that statement we can observe the mean BG curve in figure 4.23. The results in table 4.1 and table 4.2 provides more evidence that this is true. All metrics line up with the baseline, and it even got the highest outcome in mean TIR of 92.12%. Why this algorithm got the highest mean TIR compared to the baseline and DQN is because there are different meal amounts each episode, meaning that with the right CHO intake the constant flow of the optimal basal rate would yield in higher TIR than for other amounts. Categorical DQN happened to be lucky with the 14 episodes worth of BG data it received.

The learning process on the other hand seems stale, as seen in figure 4.24. After 50 episodes or so the agent seems to have learned all it can. The learning curve is virtually flat, and very early on the agent found out that giving the patient the optimal basal rate only is the best choice. For what it's worth, this strategy seems efficient enough to regulate the BG in the patient. In fact, it is effective enough to negate hypoglycemic events in the long-run.

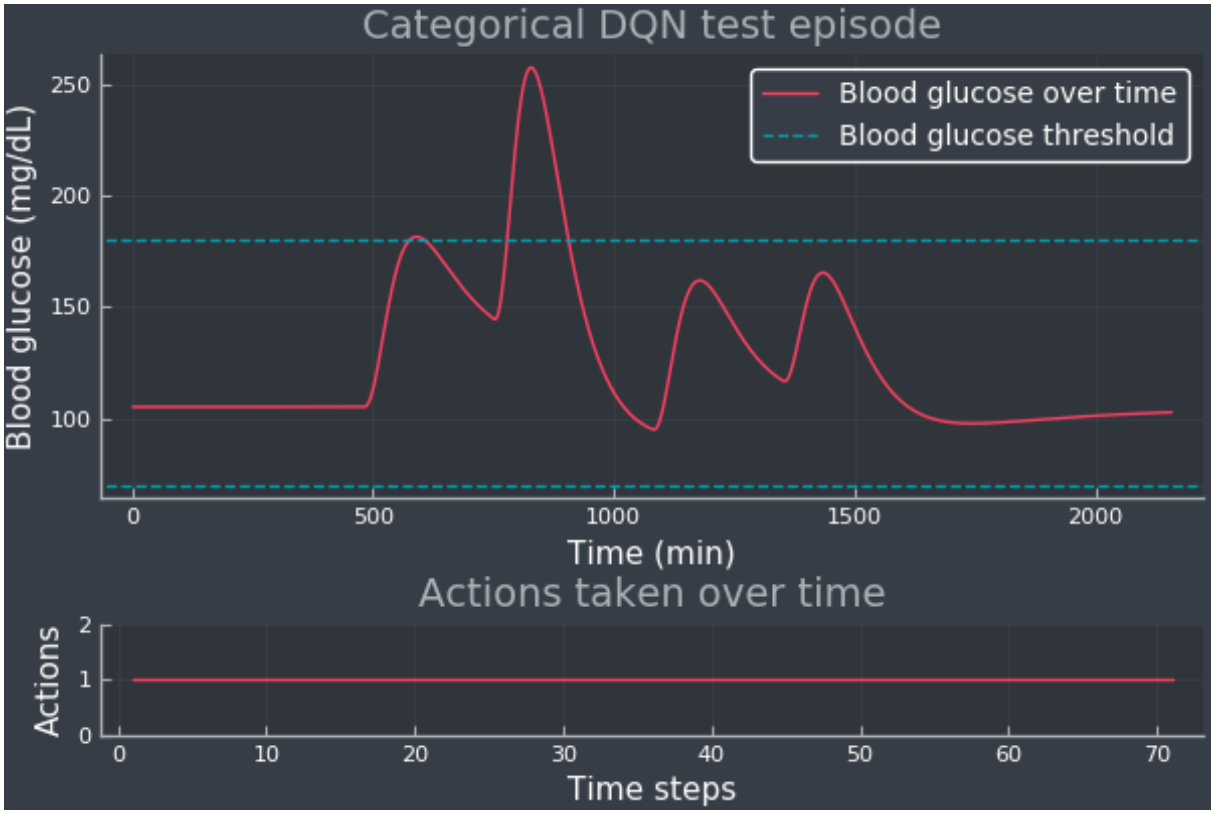

Figure 4.22: A categorical DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

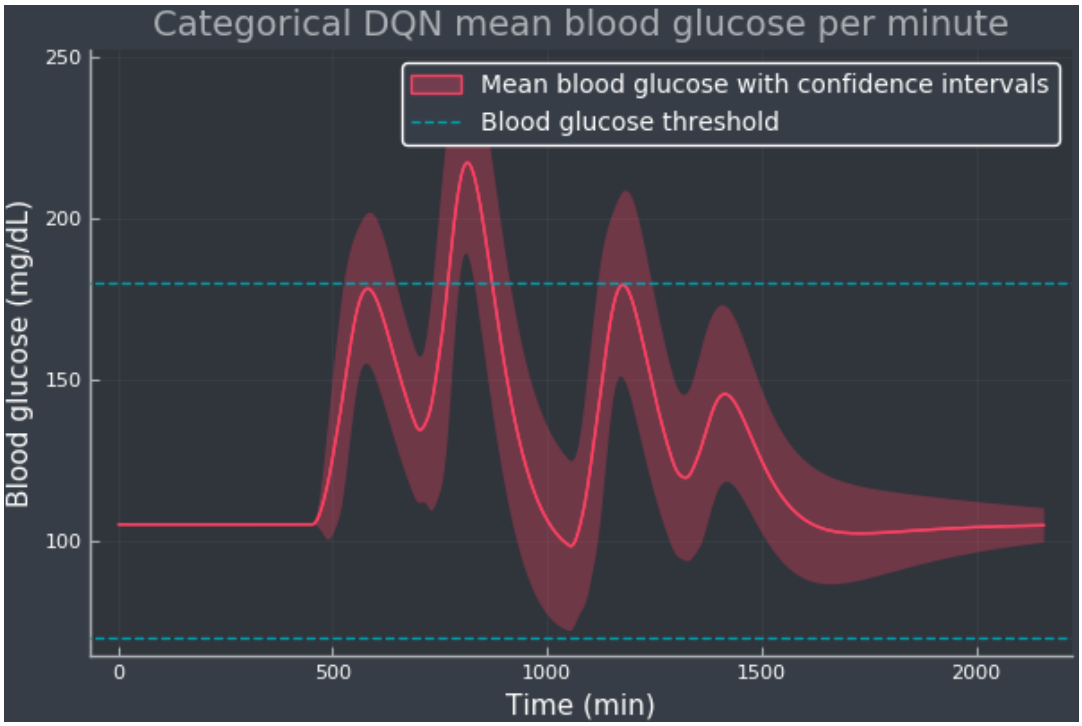

Figure 4.23: Categorical DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

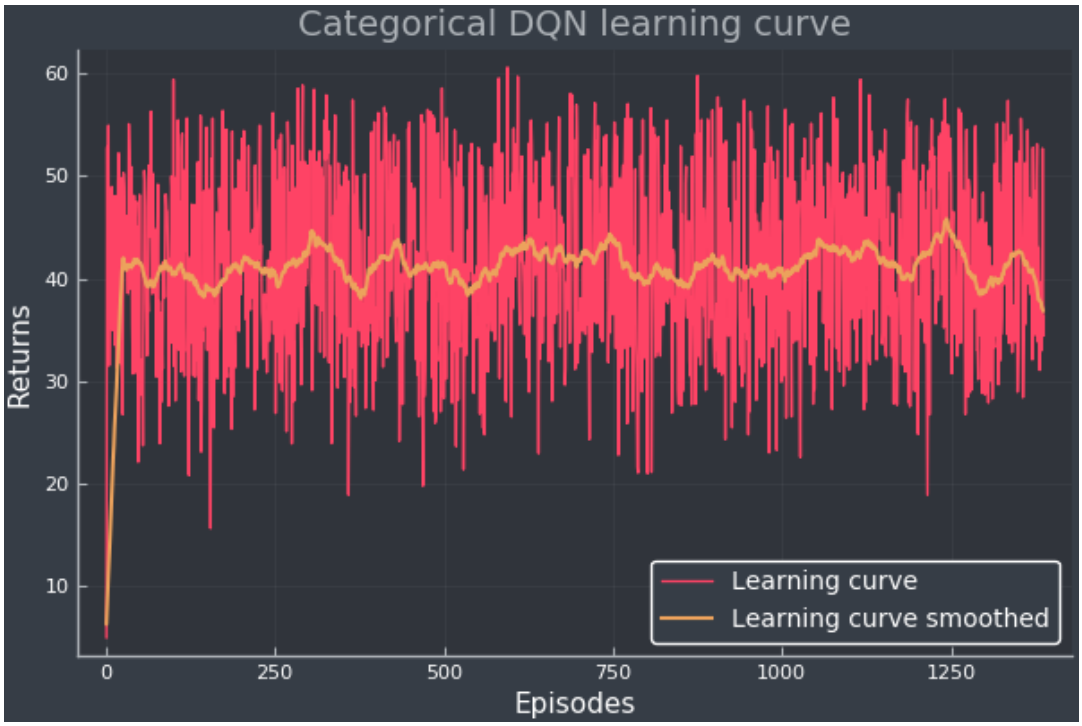

Figure 4.24: The categorical DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Why does the agent learn this strategy so fast then? The answer lies in the fact that in the categorical DQN algorithm, we use value distributions instead of value functions or expectations of the return, and a distributional Bellman operator. The latter preserves multimodality in value distributions, which can lead to more stable learning [85].

Finally we arrive at rainbow DQN, where all the methods in the previous algorithms have been implemented together. The plots in figure 4.25 and figure 4.26 shows similar character to DDQN and dueling DQN, where high basal rates are given to the patient with deficient cautiousness. The BG drop between lunch and dinner is on par with the mentioned algorithms as well. table 4.1 confirms that and convinces us that rainbow DQN have the lowest episodic BG. With zero percentage in TAR and the worst scores in mean TIR and TBR, as seen in table 4.2. As a consequence the mean TAR is lower compared to the rest.

It might have been the case that rainbow DQN required more samples to train on, or needed to adjust the network to be larger or have a different architecture. We can analyze from figure 4.27 that the agent struggles to learn anything from episode 200 and forward. There is a slope, but the return reaches only 40, which is lower compared to some of the better performing algorithms. Additionally the slope is slight and not very steep, which confirms that the agent has not learned enough information in order to efficiently exploit the action space.

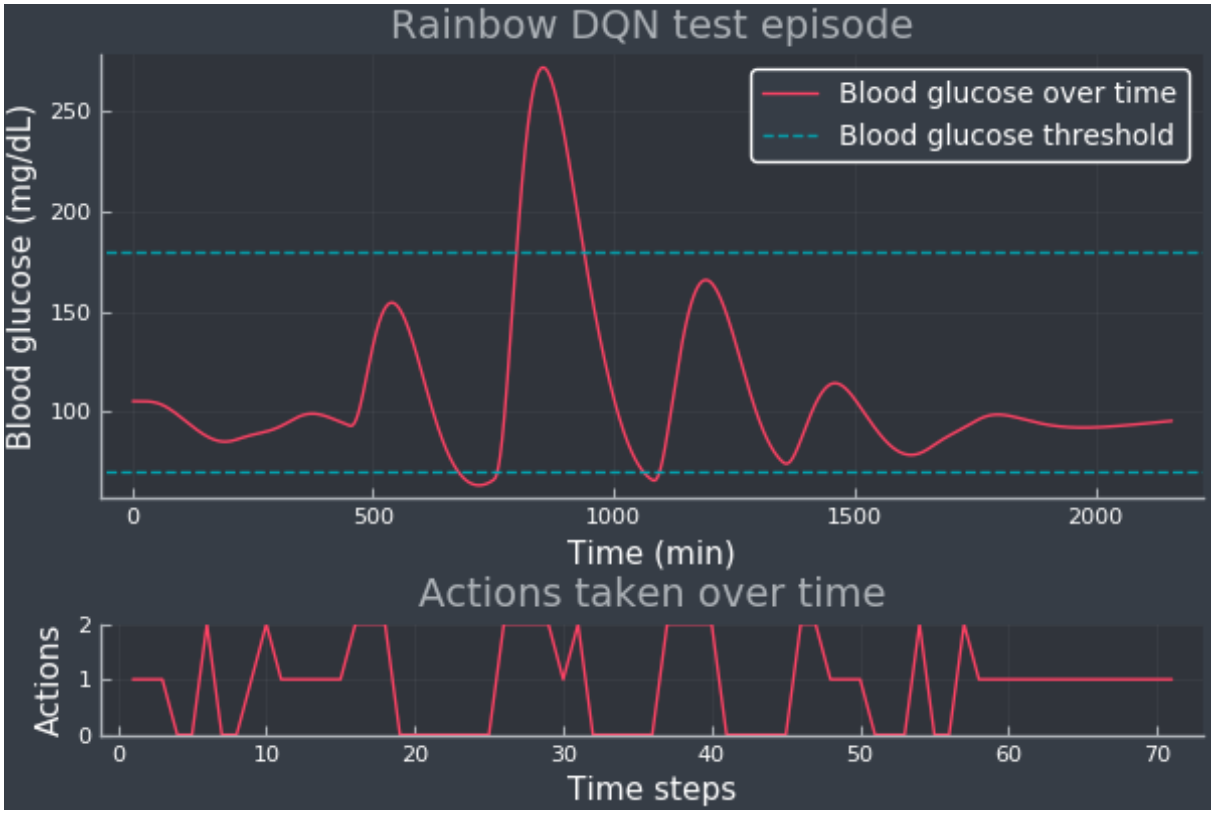

Figure 4.25: A rainbow DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

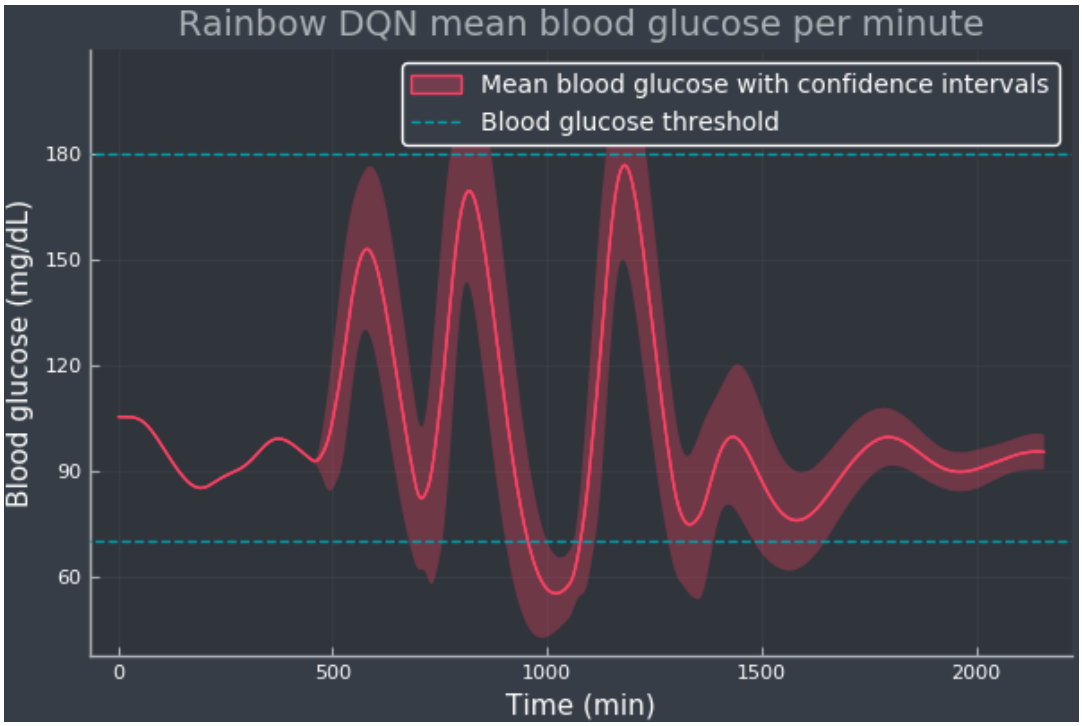

Figure 4.26: Rainbow DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

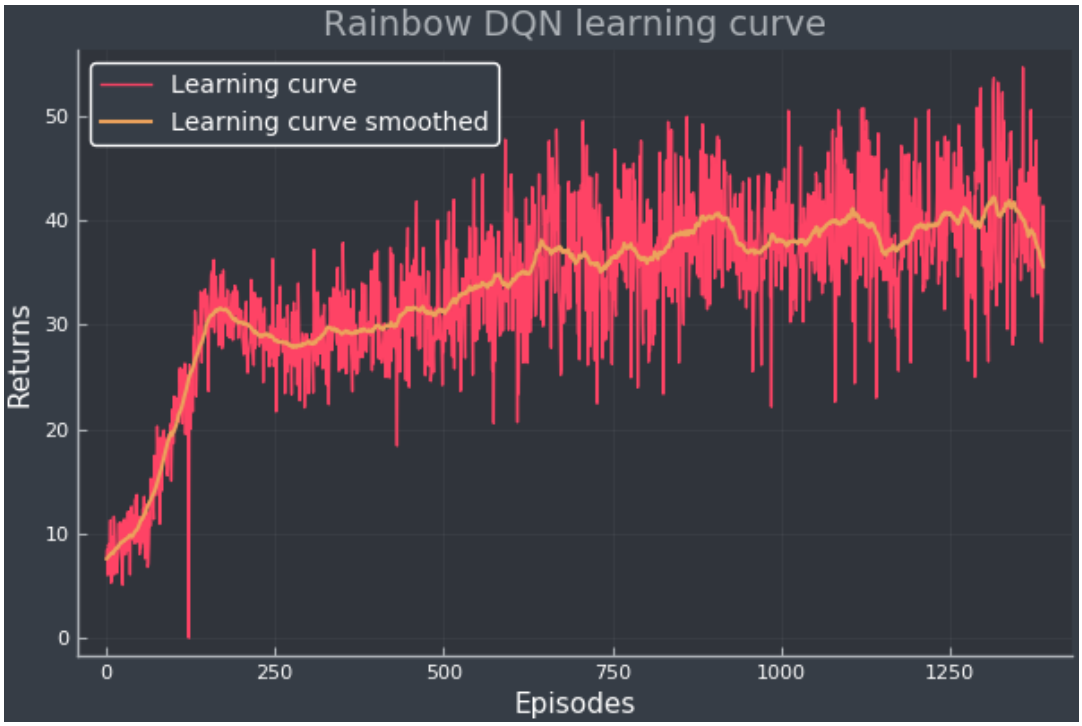

Figure 4.27: The rainbow DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

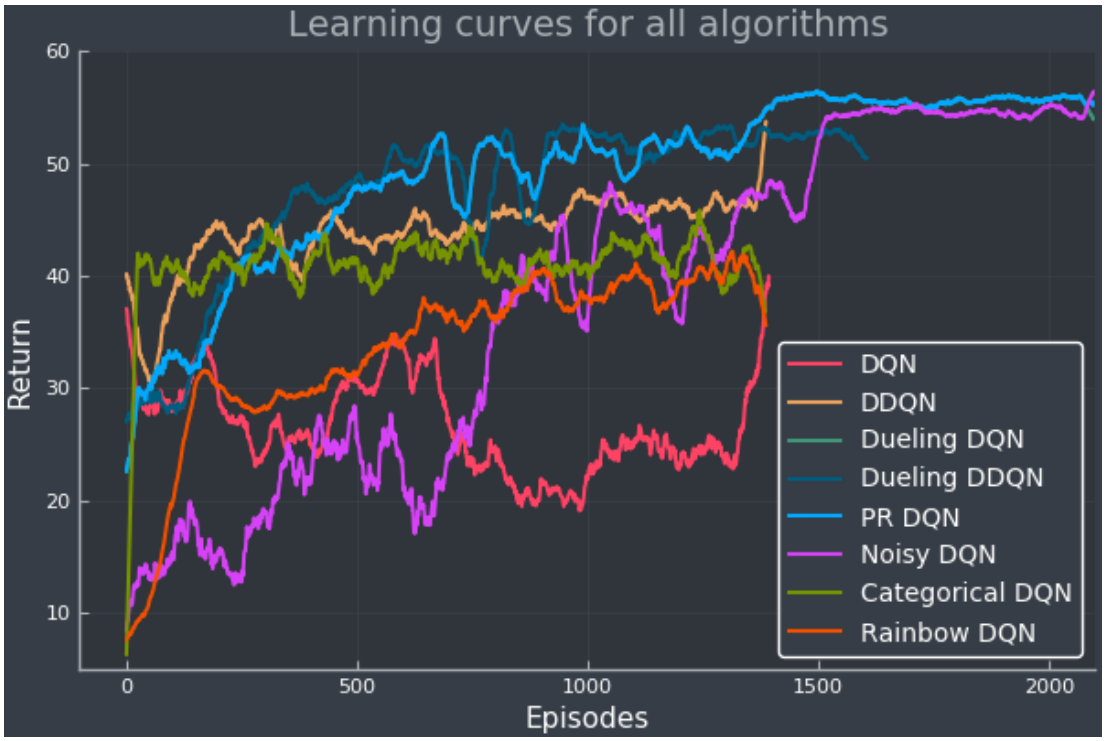

Figure 4.28: Experiment 1 - The learning curves for all the algorithms, showcasing how the return evolves with training episodes. All curves have been smoothed for easier interpretation and comparison. PR DQN have been cut off around the 2000 episodes mark, since it trained for a longer period compared to the other algorithms. The rest of the training was found to be redundant, which is discussed in this section. As a general rule; the longer the curve, the faster episodes are being terminated.

Through analyzing and comparing the different DQN extensions, a temporary conclusion is that it is possible to regulate BG in a simulated T1D patient with this experimental setup. Some algorithms performed better than others, in which the worse ones needs more development to be on par or surpass. Both the DQN agent and the categorical DQN agent learned a matching strategy to that of the baseline, in which they only give the patient the optimal basal rate. The performance of these are good enough, with a TIR score of $\approx 95\%$ and a mean TIR of $\approx 91 \pm 1\%$.

Dueling DDQN and noisy DQN performed slightly better than the baseline, with TIR scores of 96.71% and 97.04% respectively. The mean TIR of both is $\approx 92\%$ which is on par with the baseline. In figure 4.28 we see that these two algorithms surpass the others. They have learning curves with steep slopes from the start and have an average return of $50+$ near end episodes. Also notice that these are the only curves with gradual learning throughout the training. The other curves have plateaus much earlier on in the training, and less average return near the end. We also discussed the possible issue with PR DQN, hence the cutoff near episode 2000. The rest of the learning curve is not worth showing next to the others, since it is redundant and a great amount of information is lost by the agent during this period. Please refer to figure 4.18 for the complete curve.

Algorithm	TIR	TAR	TBR	μ	σ
Baseline	95.41%	4.59%	0.0%	124.00	33.84
DQN	95.05%	4.95%	0.0%	125.04	35.21
DDQN	92.82%	1.90%	5.28%	111.67	33.31
Dueling DQN	93.33%	0.45%	6.25%	124.32	32.24
Dueling DDQN	96.71%	3.29%	0.0%	126.92	32.32
Prioritized Replay DQN	94.35%	5.65%	0.0%	122.63	33.26
Noisy DQN	**97.04%**	2.96%	0.0%	**116.10**	**31.74**
Categorical DQN	95.23%	4.77%	0.0%	125.25	34.68
Rainbow DQN	94.35%	**0.0%**	5.65%	100.66	32.20

Table 4.1: Experiment 1 - TIR, TAR and TBR of the mean BG per minute of 100 episodes. μ is the mean BG per episode and σ is the standard deviation of the BG per episode. Estimated for different DQN extensions. The best results are written in **bold text**, while the worst results are written in red text. Note that in the TBR column there are multiples of the same result, hence they are not highlighted in bold.

Algorithm	TIR	TAR	TBR
Baseline	91.83%	7.80%	**0.37%**
DQN	90.01%	9.52%	0.47%
DDQN	88.44%	4.35%	7.22%
Dueling DQN	91.70%	7.19%	1.11%
Dueling DDQN	91.92%	6.13%	1.95%
Prioritized Replay DQN	89.31%	8.48%	2.22%
Noisy DQN	92.06%	5.73%	2.22%
Categorical DQN	**92.12%**	6.19%	1.69%
Rainbow DQN	85.53%	**2.63%**	11.84%

Table 4.2: Experiment 1 - Mean TIR, TAR, and TBR of 14 episodes of BG data. Estimated for different DQN extensions. The best results are written in **bold text**, while the worst results are written in red text.

4.2.2 Experiment 2 - Expanded Action Space

With this experiment, we anticipate either that the models will benefit from more actions to choose from, making the learning process better, or that more actions will complicate the training. It has been proved by Fox and Wiens [23] that discritizing the action space into three bins, similar to what we did in experiment 1, is sufficient to successfully control the BG in a simulated T1D patient. It will be educational to investigate what benefits/drawbacks an expanded version of the action space in experiment 1 can do.

For the fundamental DQN algorithm we observe similar results from experiment 1. The agent only gives the patient the optimal basal rate (figure 4.29) and the BG curve over time (figure 4.30) is akin to the baseline. In table 4.3, the σ estimate for this agent has been noticed as the worst result in this category, though it is insignificant since they are all so close-packed. The learning curve in figure 4.31 is somehow different. The agent seems to learn decently after 750 episodes and then the curves becomes flat, which means it cannot learn anything new. Before that, it struggles to maintain the learned information, which is evident from all the downfalls in the curve. This is truly a cause of overfitting. Near the last episodes, ε is reaching 0.01 meaning that the agent is more likely to select the greedy action. It chose to take action 2 and got decent returns. All in all, this agent is producing matched results to the baseline.

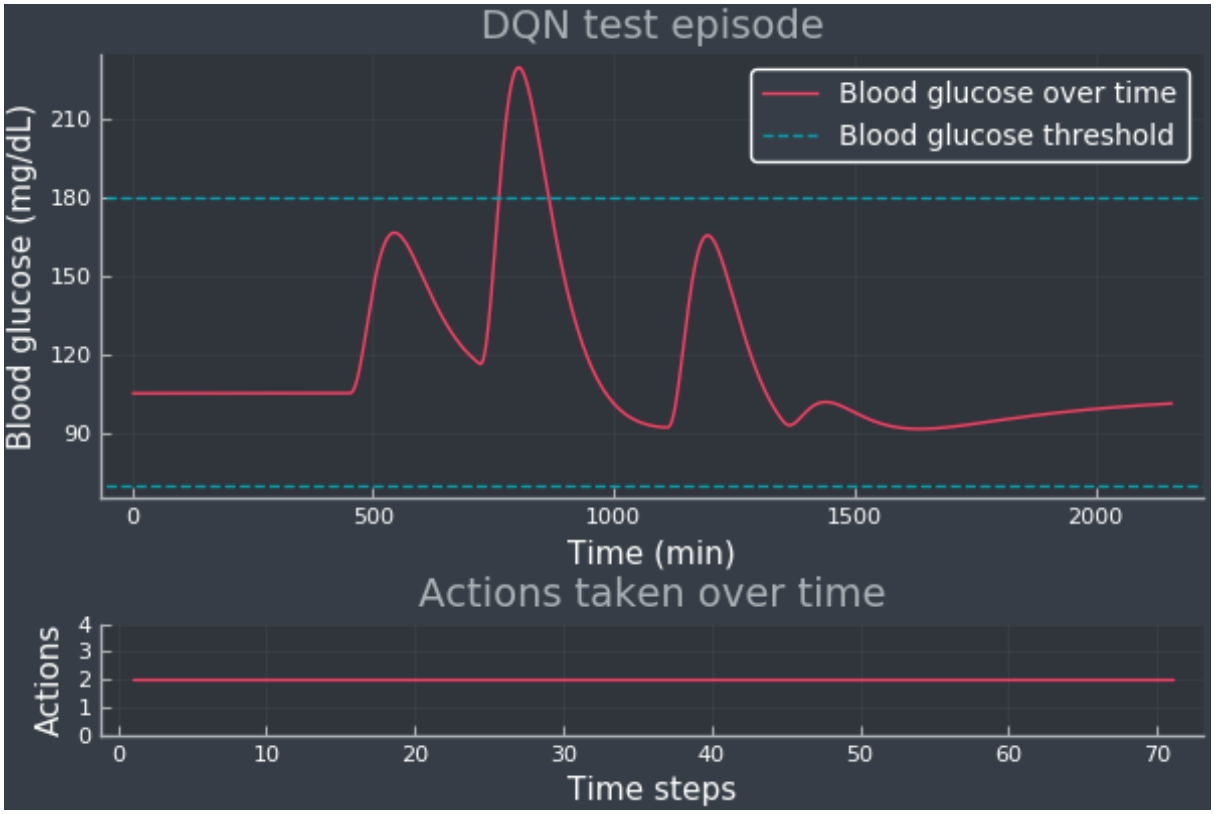

Figure 4.29: A DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

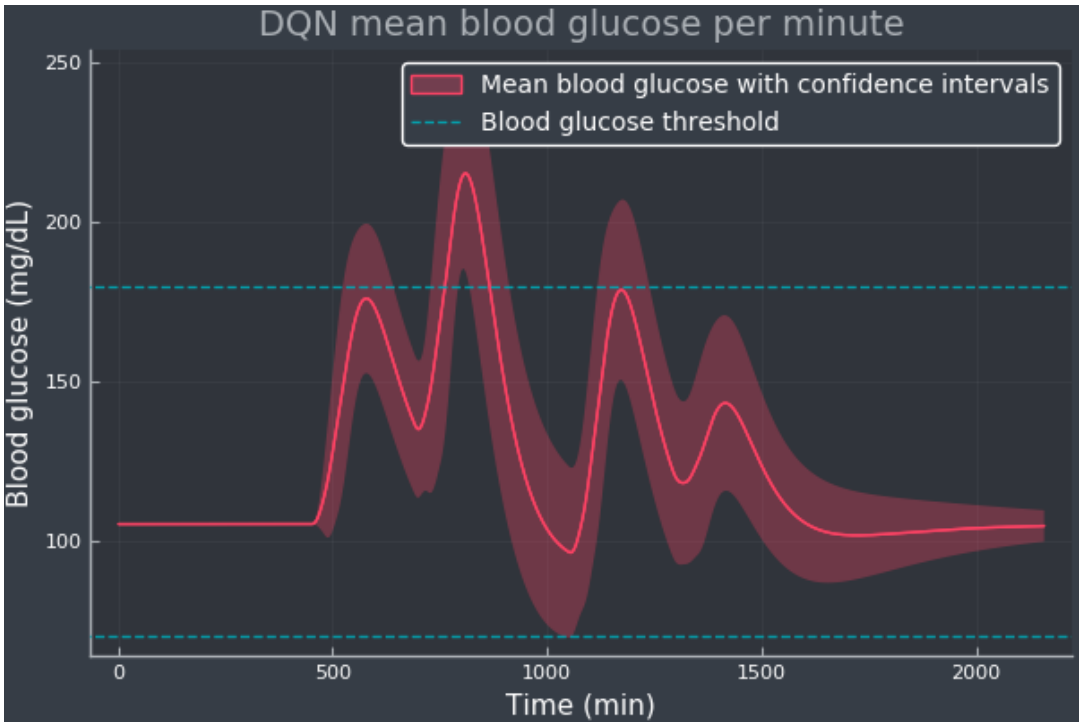

Figure 4.30: DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

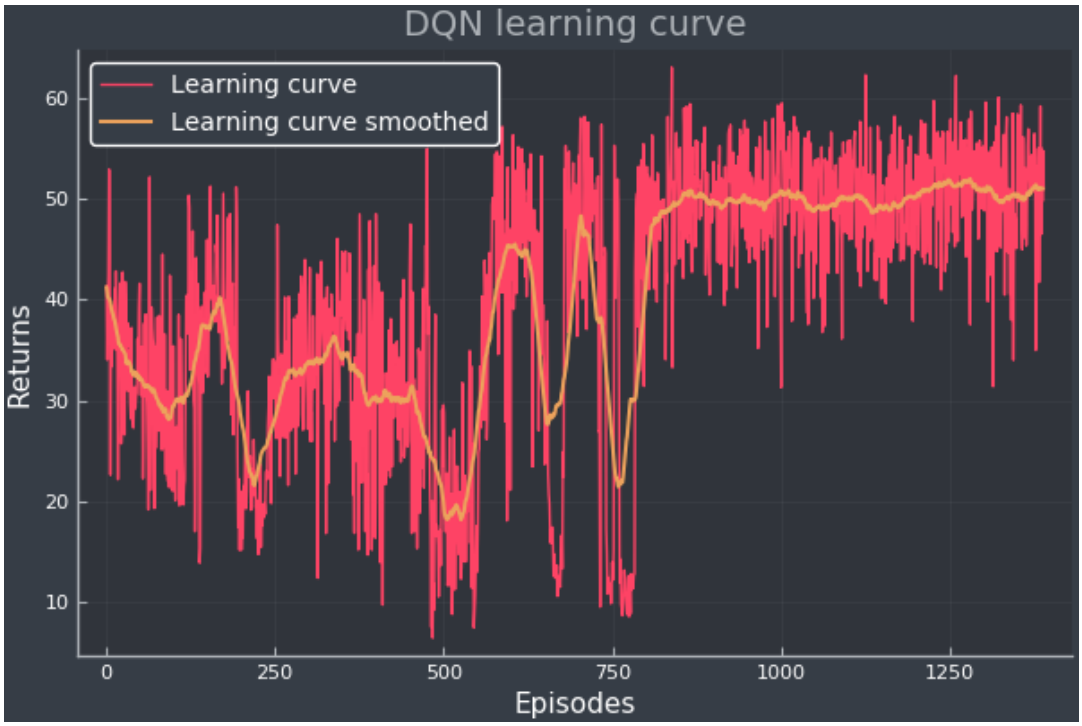

Figure 4.31: The DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

The DDQN agent is also similar to experiment 1 version. Notice that in figure 4.32, the agent picks frequently the high basal rate actions, which results in lower episodic BG levels, as we can see in table 4.3 and table 4.4. The μ estimate is at 103.29 mg/dL which symbolizes low BG values per episode. In addition, the TBR estimate of 6.20% scored the second lowest compared to other algorithms. The mean TIR and TBR estimates also scored low compared to the others, with a percentage of 85.29% and 11.94% respectively. Consequently the TAR and mean TAR scored the best, 0.0% and 2.77% respectively. We can confirm this by reviewing figure 4.33, where it's clear that there is a decrease in BG between lunch and dinner time. This decrease is similar to the results of the DDQN agent and dueling DQN agent from experiment 1.

In the learning curve (figure 4.34) of this algorithm, it is evident that the agent has not learned too much during training. We can draw this conclusion from the fact that the curve is not steep but slight and that the return starts around 35 and barely reaches 50 near the last iterations.

Comparing these outcomes with the baseline and DDQN results, it is comprehensible that more actions indeed complicated the learning for the agent. With more than 10% in hypoglycemic territory per day, this version of the algorithm is not feasible in controlling BG levels.

Figure 4.32: A DDQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

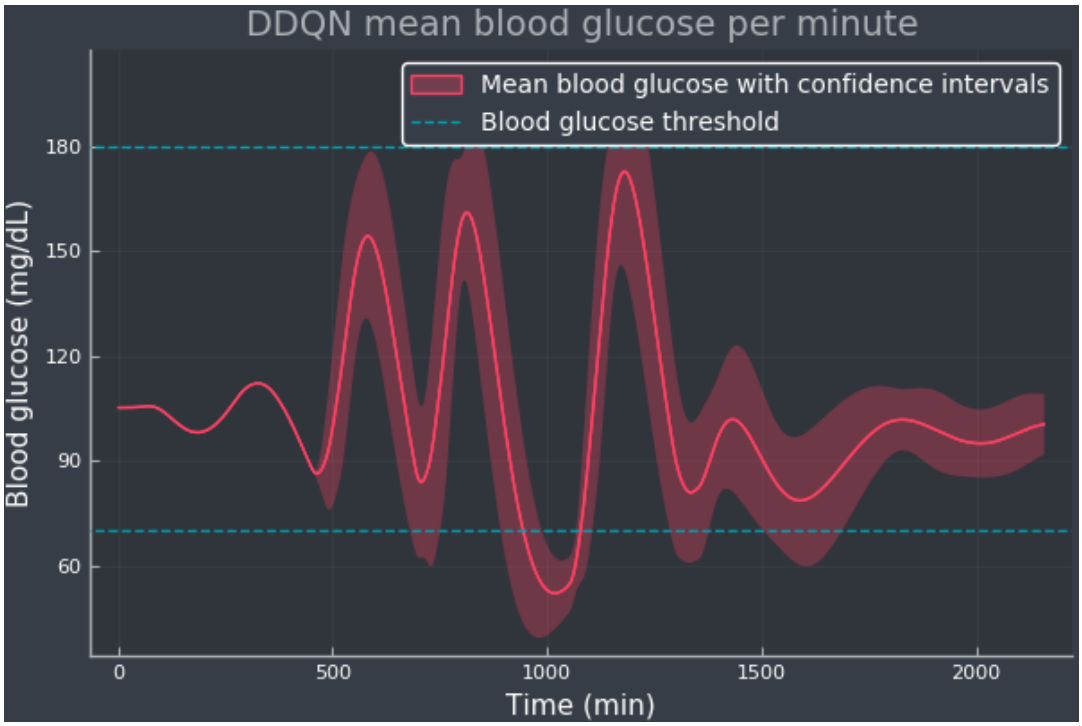

Figure 4.33: DDQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

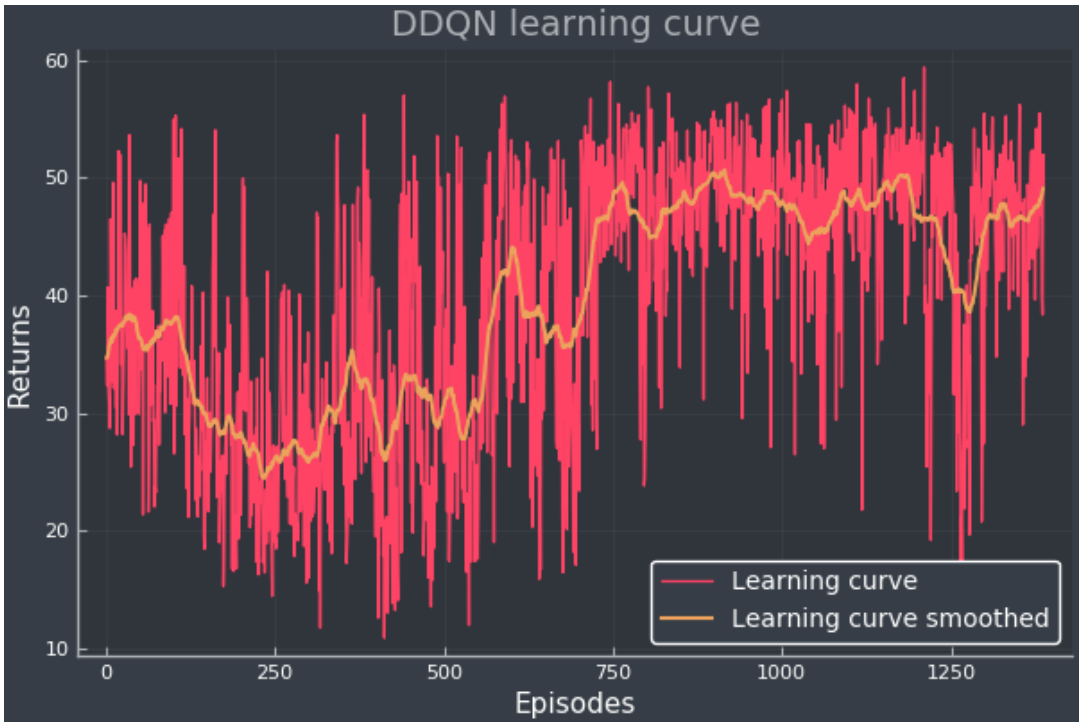

Figure 4.34: The DDQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

For the dueling DQN agent, it scored the best in TIR (97.04%) and in μ estimate (113.80 mg/dL), which is slightly better than the baseline, as seen in table 4.3. However, looking at the mean percentages in table 4.4, the episodic TAR were a bit higher than the rest, estimated as 8.30%. If we now analyze the graphs in figure 4.35 and figure 4.36, we see that test episode is similar to its experiment 1 version. The action selection is very diverse this time; the agent has learned to fine tune the usage of all basal rates from the action space. Also note that the least picked action is number 2, which is the optimal basal rate $b^* = 6.43$ mU/min. Specifically for this algorithm, the agent found it more efficient to control the BG concentration using multiples or parts of the optimal basal rate. For the mean BG curve we can see that the BG peaks in the curve has been attuned lower, and the tail of the curve is regulated back to the reference point.

Once again we see a decay in the learning curve (figure 4.37), which might hint to that the network has been overfitted, just like the PR DQN in experiment 1. The reason why might be as simple as that the same training samples was being used repeatedly. Although this is not a big deal at this point, since the agent has proved to successfully control the BG in the patient, but perhaps a modest improvement in the network or hyperparameters would have made the learning process even more efficient, possibly resulting in an agent near perfecting the usage of the action space in all scenarios.

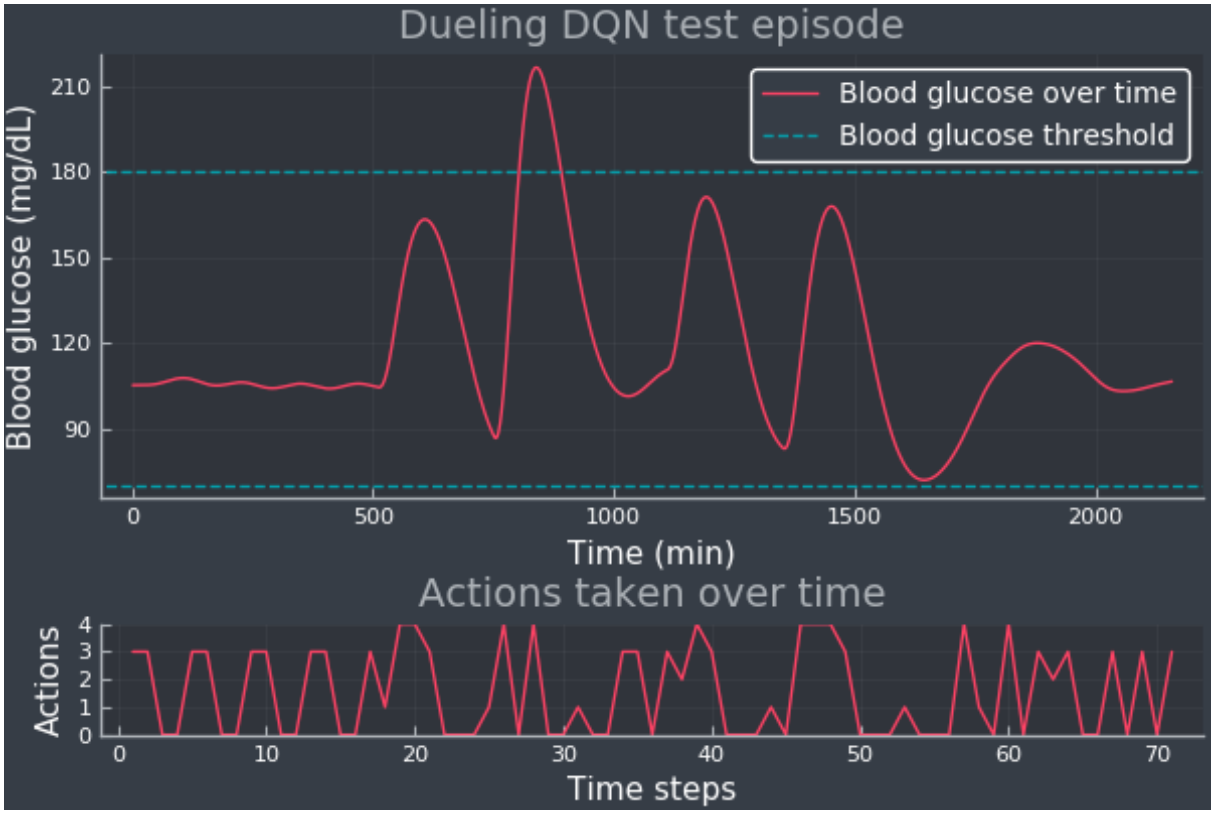

Figure 4.35: A dueling DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

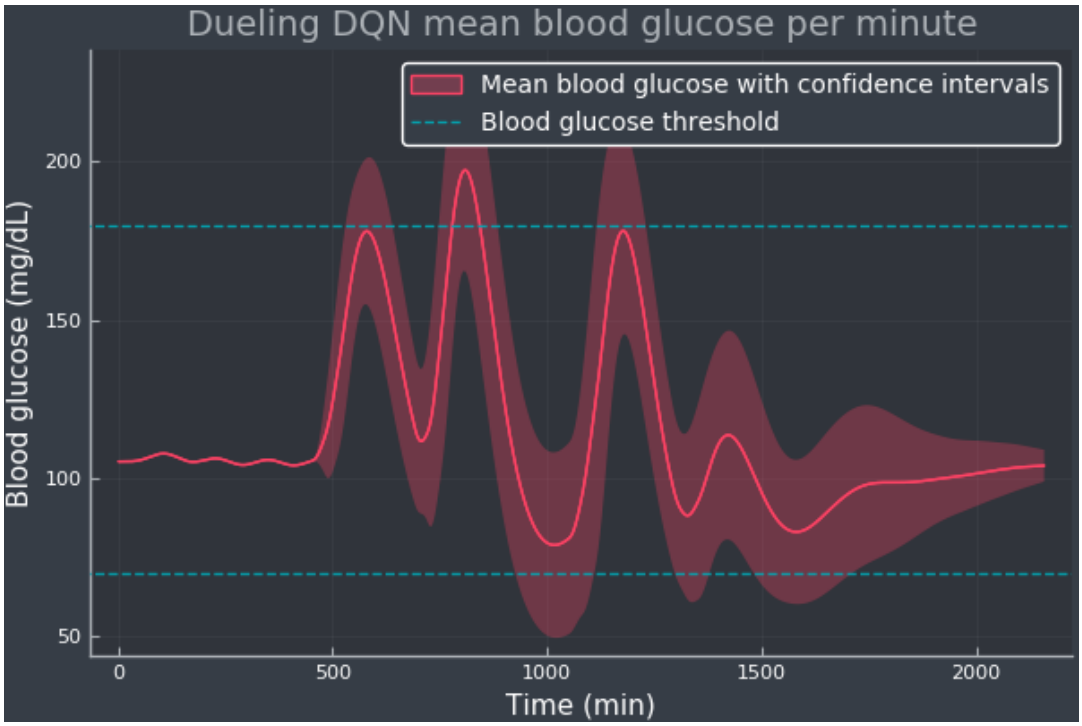

Figure 4.36: Dueling DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

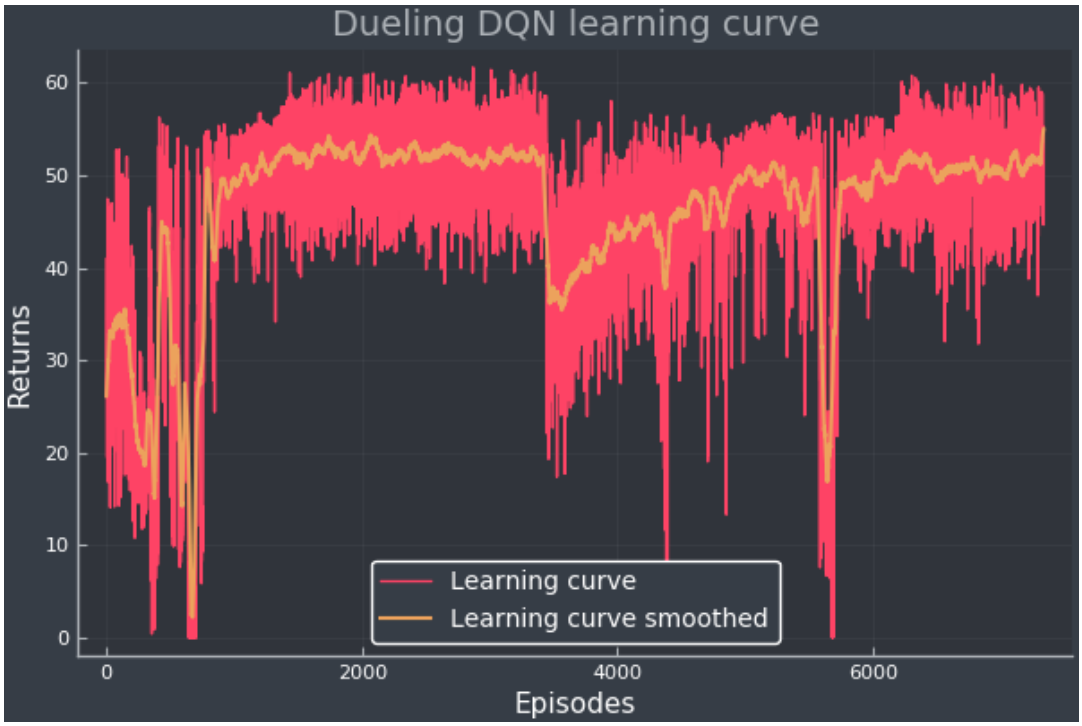

Figure 4.37: The dueling DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Turning over to the dueling DDQN case, the learning curve in figure 4.40 is very similar to the previous algorithm, only with a smaller decrease near the midpoint of the training. An interesting discovery is that the action selection pattern is very similar to the experiment 1 version, as seen in figure 4.38. This sawtooth way of selecting basal rates, that the agent has learned, seems like an alternate method of managing the BG levels corresponding to the dueling DQN agent's way. Comparing it with the mean BG curve in figure 4.39, the dinner meal has not been managed as good. The BG levels is slightly higher in this experiment compared to experiment 1. In table 4.3 the TAR estimate is 5.09%, which scored the worst, and the σ estimate is 31.60 mg/dL, which scored the best.

Comparing dueling DQN and dueling DDQN, we see that the former has more ideal BG levels over time. In terms of episodes, both have similar TIR means. The DDQN agent had the tendency to pick frequently high basal rates resulting in high TBR, while including double Q-learning in the dueling network fixed that issue and instead made the agent more carefully pick basal rates. This is an interesting found because in experiment 1, dueling DDQN had better results than the regular design. It appears that the target network in double Q-learning in synergy with extended action space somewhat complicate the learning process for the agent, developing lower TIR scores and faintly higher BG levels per episode. It also decreased the BG standard deviation per episode by $2 - 3\%$.

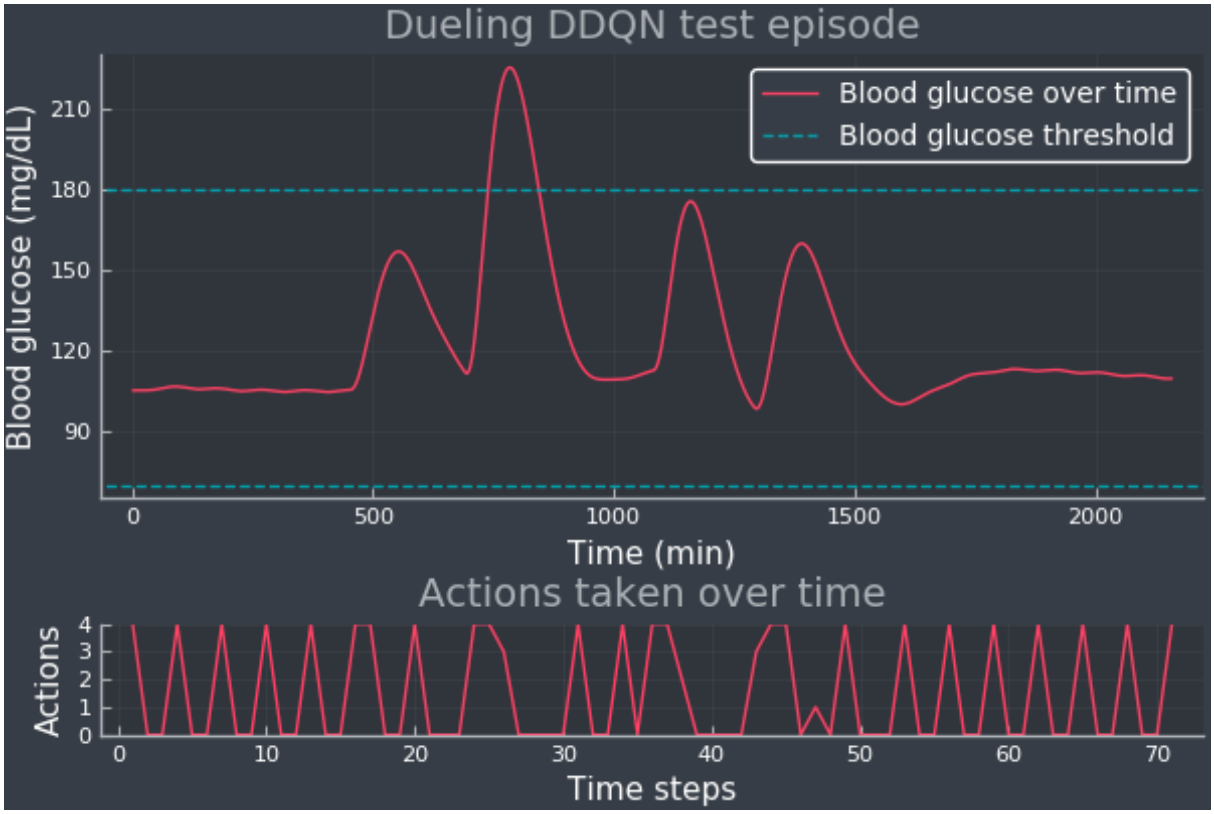

Figure 4.38: A dueling DDQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

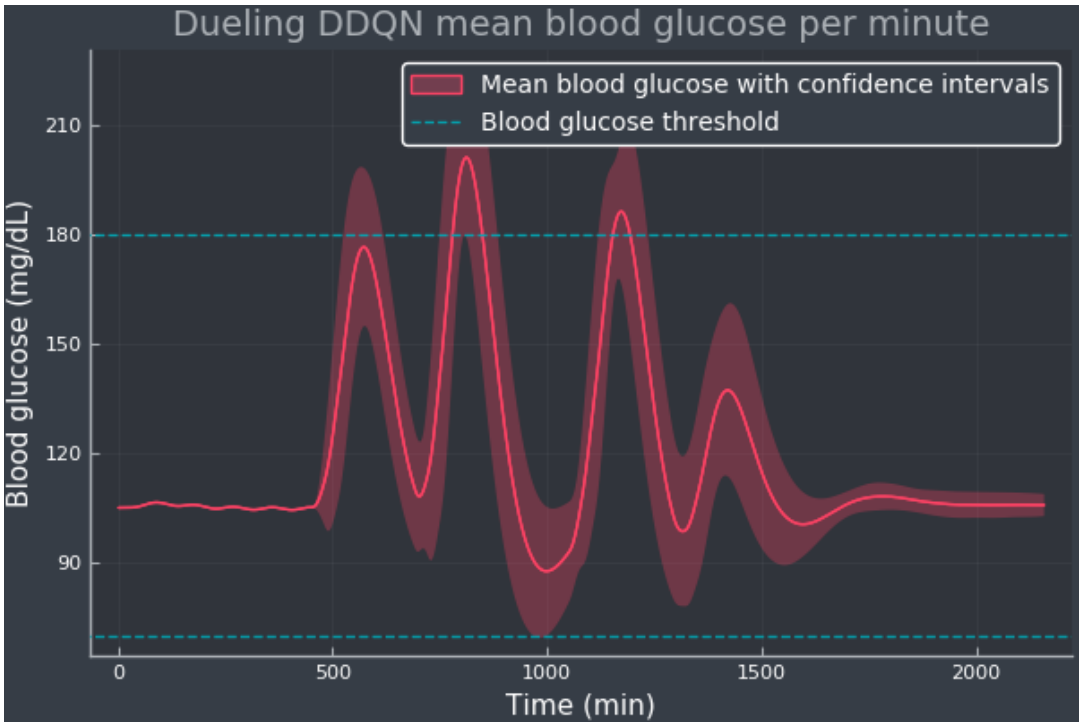

Figure 4.39: Dueling DDQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

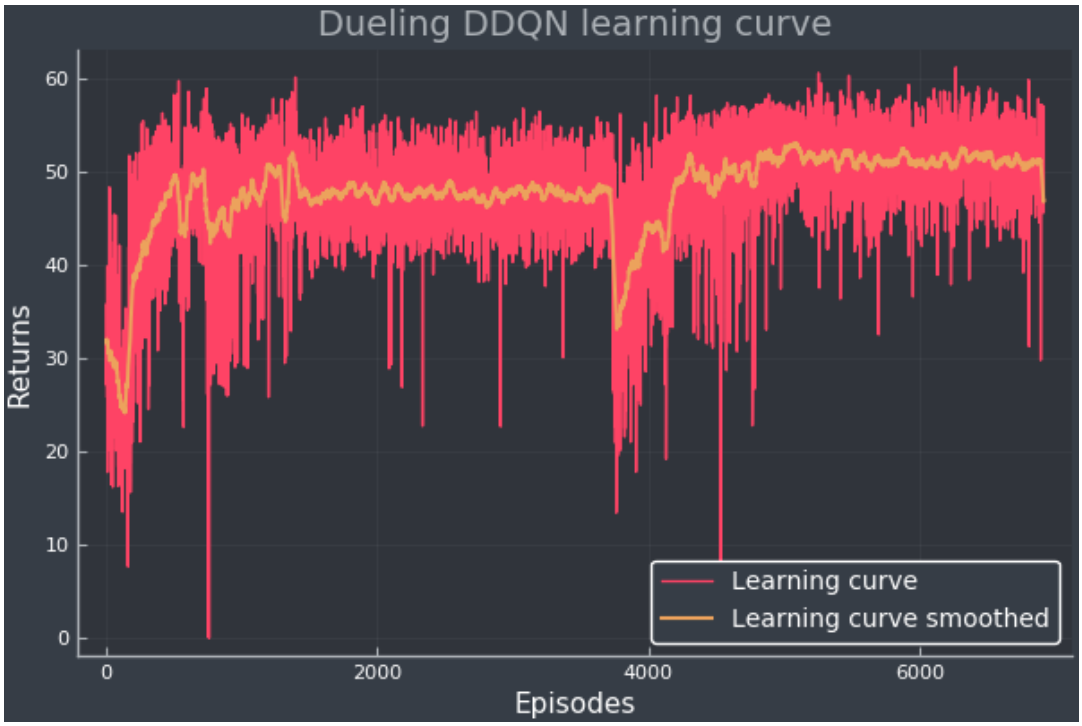

Figure 4.40: The dueling DDQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

The next algorithm is the PR DQN, and in figure 4.41 the last meal is flattened out completely by the insulin dosages. This might be a special case for this episode, so an investigation of the mean BG curve should be done. The BG peak near lunch time in figure 4.42 has been lowered, and the BG drops beyond the low bounds some hours after. We notice that the last meal has not been overrun by the given insulin, which is a sign that the agent is giving less insulin near the end of the day.

The results in table 4.3 and table 4.4 reveals that this algorithm is on par with DDQN, with slightly worse TIR and a bit better TBR. The episodic TBR is much better though, suggesting that the PR DQN agent is more careful giving too high basal rates compared to DDQN. Contrasting the two graphs in figure 4.32 and figure 4.41 it is clear that this is true.

This version of the algorithm is still not a candidate for regulating BG in the patient, as the TBR in the long run is 5.60%, which is not desired. In experiment 1, this scenario was the opposite, with zero TBR and the highest percentage in TAR. We can also observe that in figure 4.43 there are several declines in the learning suggesting catastrophic interference once again. The network trains alright in the first few episodes, and near the end iterations at approximately 2800 episodes. In the other parts of the training, the agent seems to forget previously learned information. Some of that information could have led to the agent being more careful after the lunch meal.

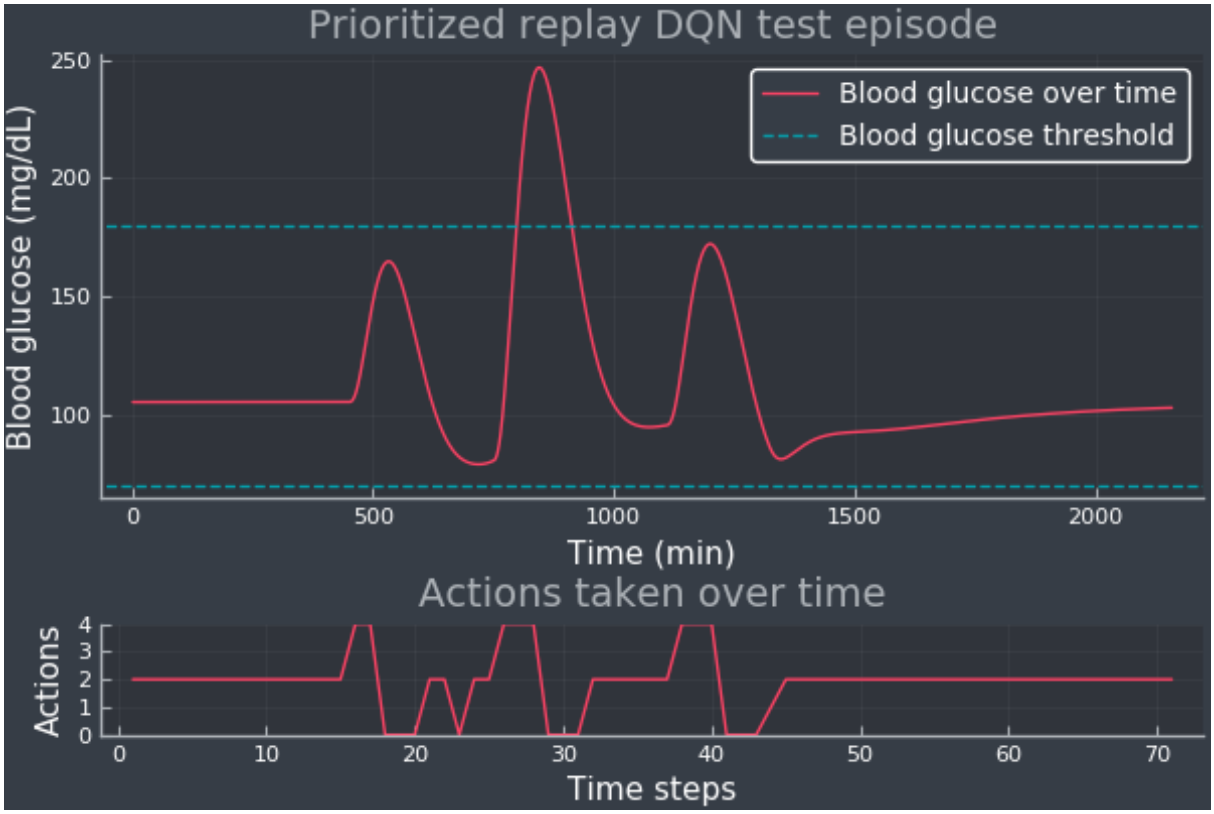

Figure 4.41: A prioritized replay DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

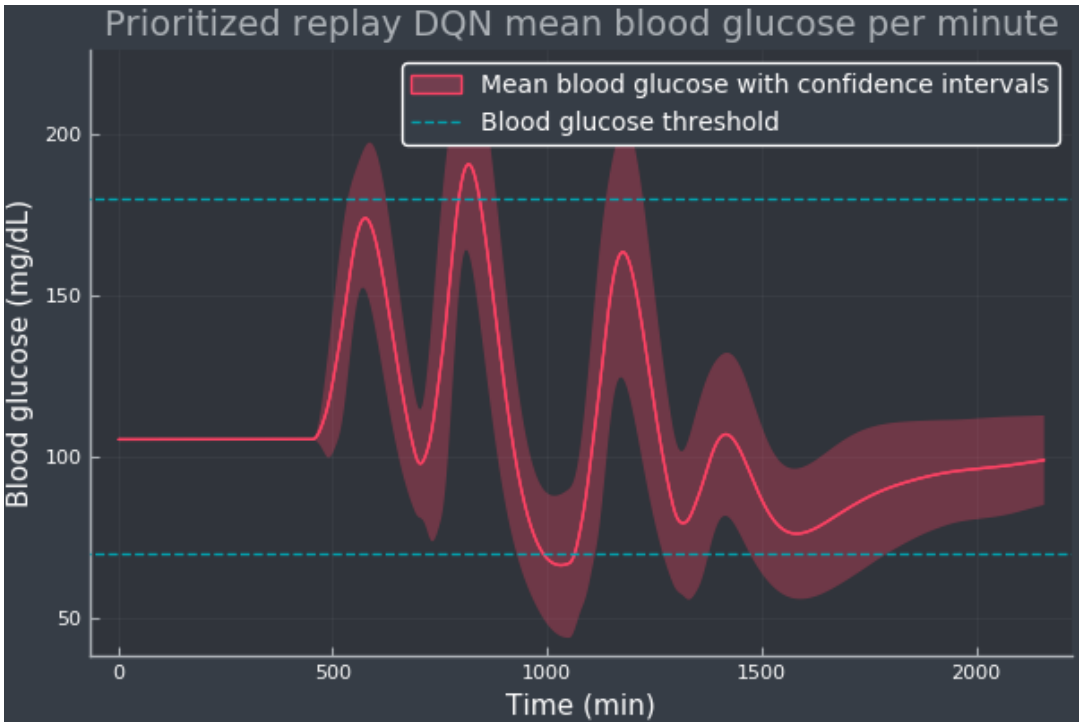

Figure 4.42: Prioritized replay DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

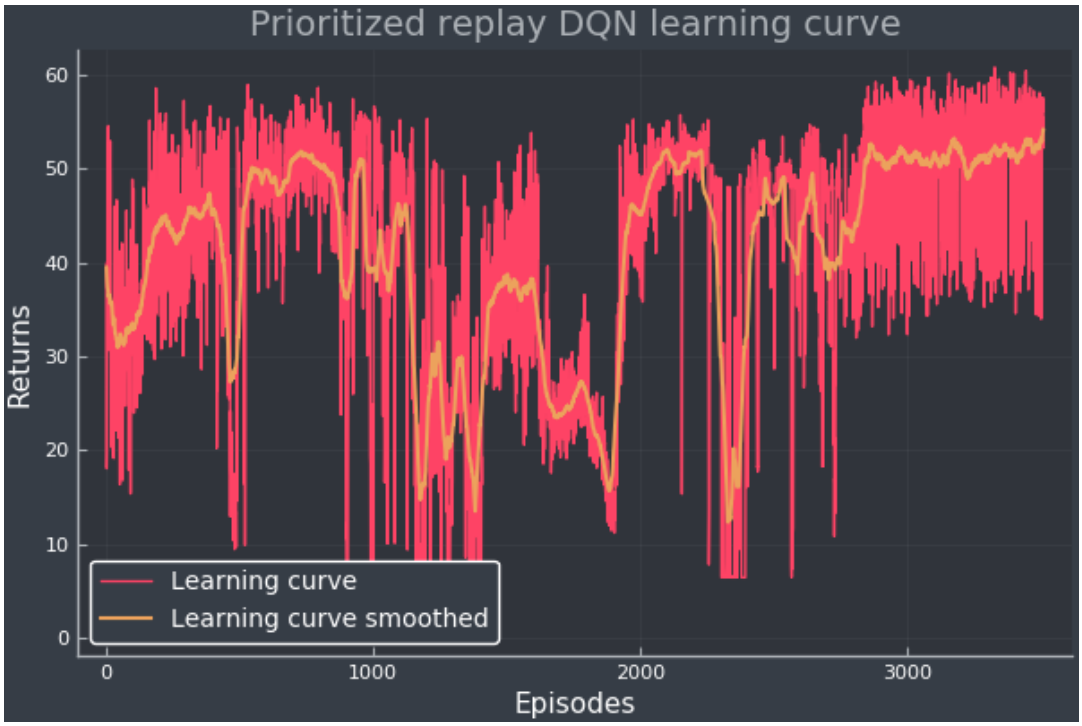

Figure 4.43: The prioritized replay DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Noisy DQN interestingly enough struggles with controlling the first and third meal, even though the basal rate choices are rather aggressive near the meal times. A side by side comparison between figure 4.45 and figure 4.45 we observe that the mean BG curve has reduced BG peaks, while getting a drop between lunch and dinner. This aggressiveness is a common mistake that the agent has acquired as a skill during training, and is something we have seen many times already. The results in table 4.3 are approximate to DDQN and PR DQN. The μ estimate also points out that the mean BG per episode is close to the reference point, but since the TBR is greater than 4% we know for a fact that the BG levels are lower than desired, each episode. In table 4.4 we see similar results and pointers.

The learning curve in figure 4.46 has many drops, which implies the same understanding as for PR DQN. In experiment 1, the synergy between PR buffer and noisy network had a great outcome. With more actions to choose from, the agent might be confused since the added noise in the layers encourages the agent to explore even more than before. A fix to this could be to have a steeper decay on the ε-greedy action selection. Perhaps a better solution might be to extend the network instead and make it deeper. Sometimes more neurons equals higher probability to learn useful information, and in conjunction with noisy layers, a deeper network might help out with exploration and exploitation.

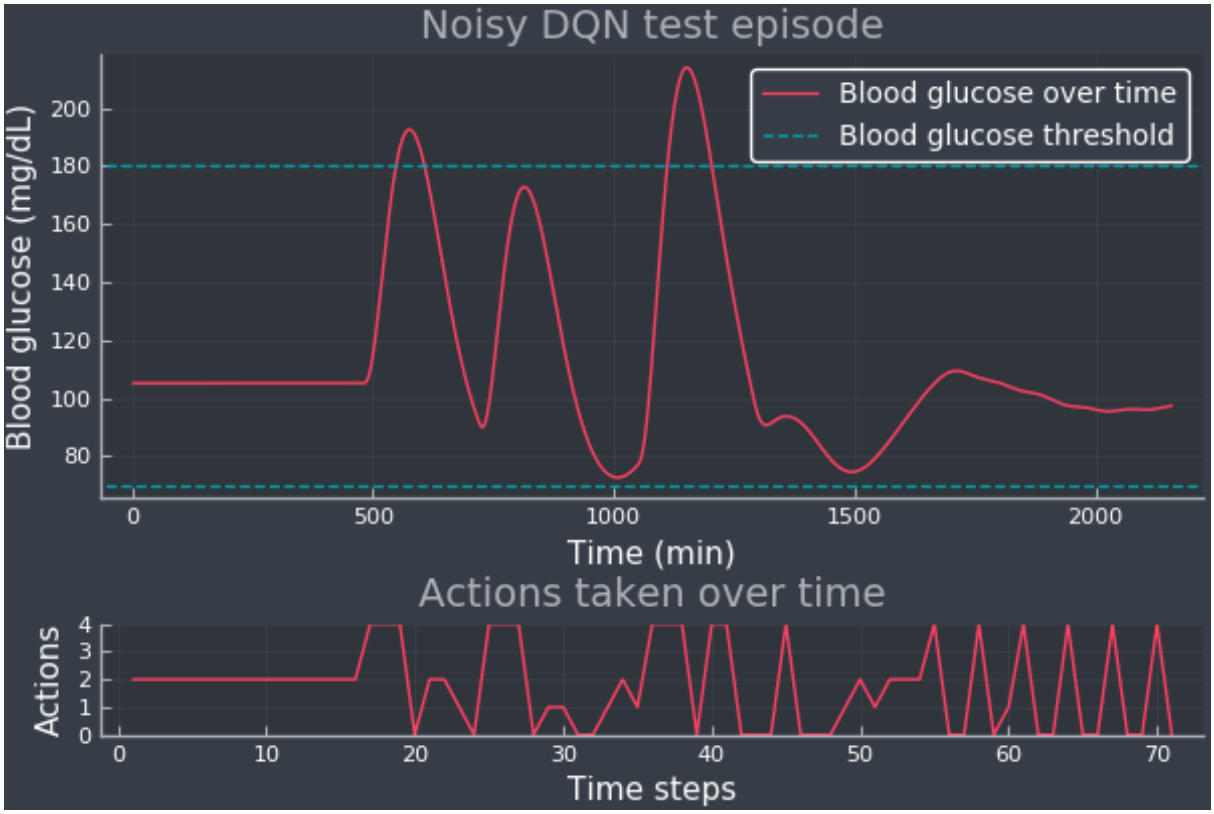

Figure 4.44: A noisy DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

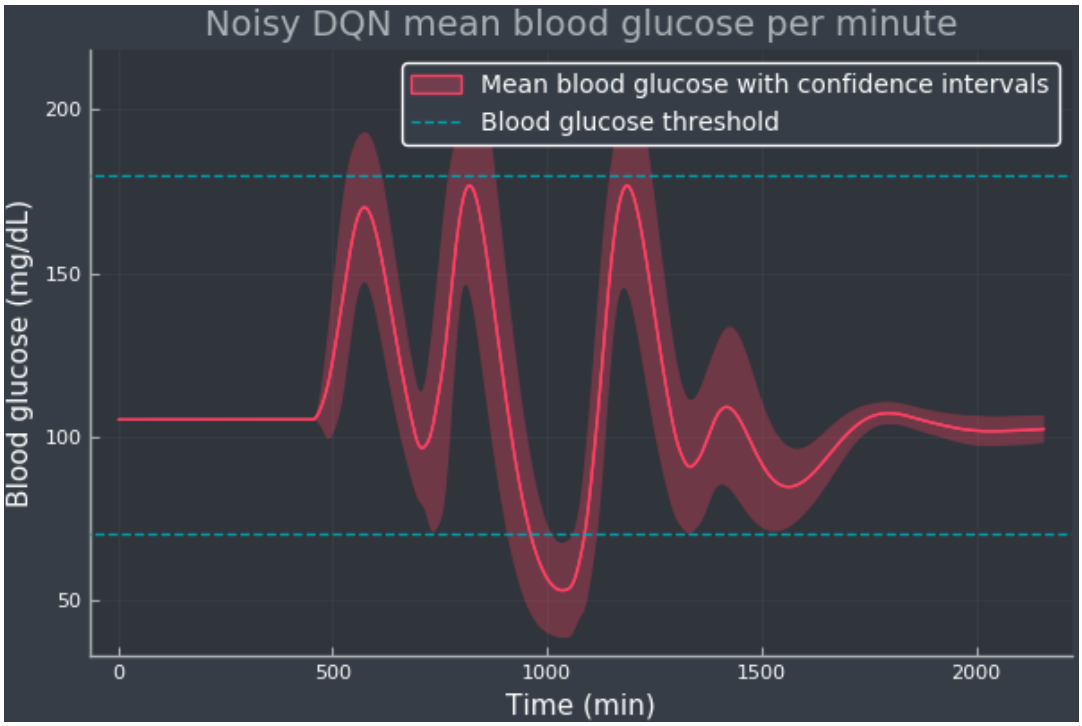

Figure 4.45: Noisy DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

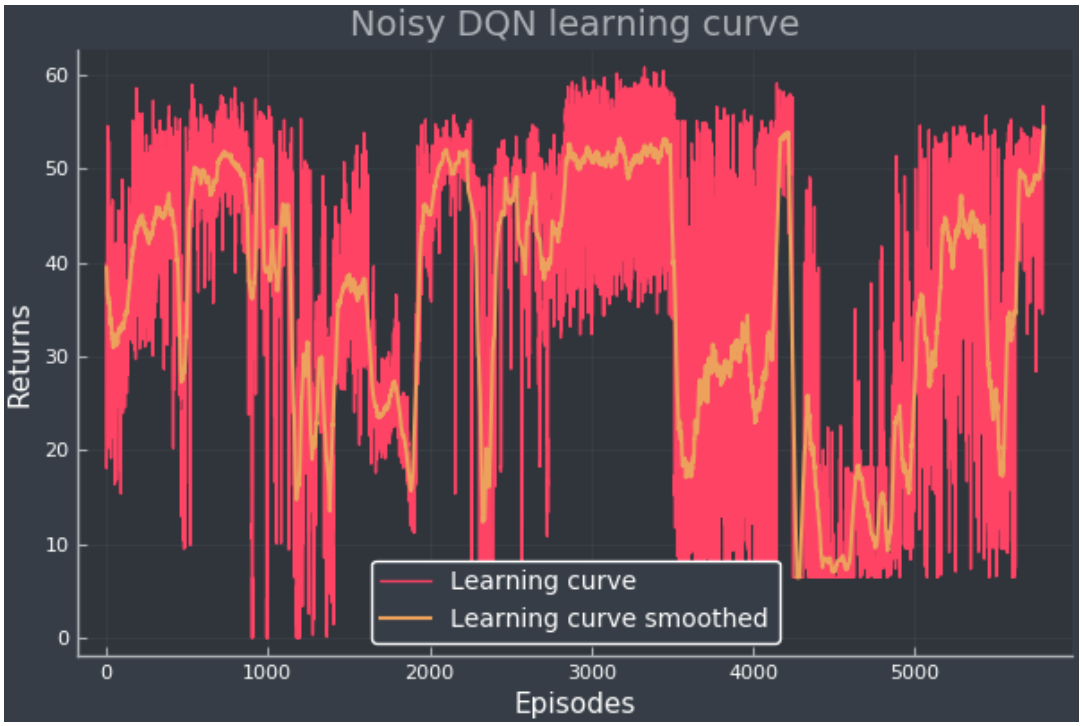

Figure 4.46: The noisy DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Once again, with categorical DQN, the agent has adapted to an aggressive style of giving basal rates. We can see this in figure 4.47 and figure 4.48 that the agent never stops the pump this time and ends up giving the patient a TBR of 6.11%. Reported from experiment 1, dueling DDQN and noisy DQN managed to regulate the drop in BG between lunch and dinner really well. The action selection pattern is usually to give the maximum amount of basal rate possible and turning off the pump right after, forming a sawtooth pattern. Back to experiment 2 with categorical DQN, since the agent never stops the pump there will be an imbalance between insulin and BG, which leads to hypoclycemia. Other than that, this agent has similar results compared to DDQN, PR DQN and noisy DQN.

Similarly to experiment 1, figure 4.49 shows a learning curve that has progress in the first 100 episodes or so. It then drops a bit and stays more or less flat for the rest of the training. The difference here is that agent learned a forceful approach in using high basal rates when needed, but never stopping the pump. The flat curve reminds us that the agent has accepted its past knowledge and is satisfied with its choices. This is a red flag because as we can see in the learning curve, the return is constant around 40, which means the agent receives a good amount of reward. Considering the outcome that was just discussed, this is not wanted. Perhaps in the future, modifications to the reward function or environment would be sufficient, implementing some kind of punishment that steers the agent further in the right direction.

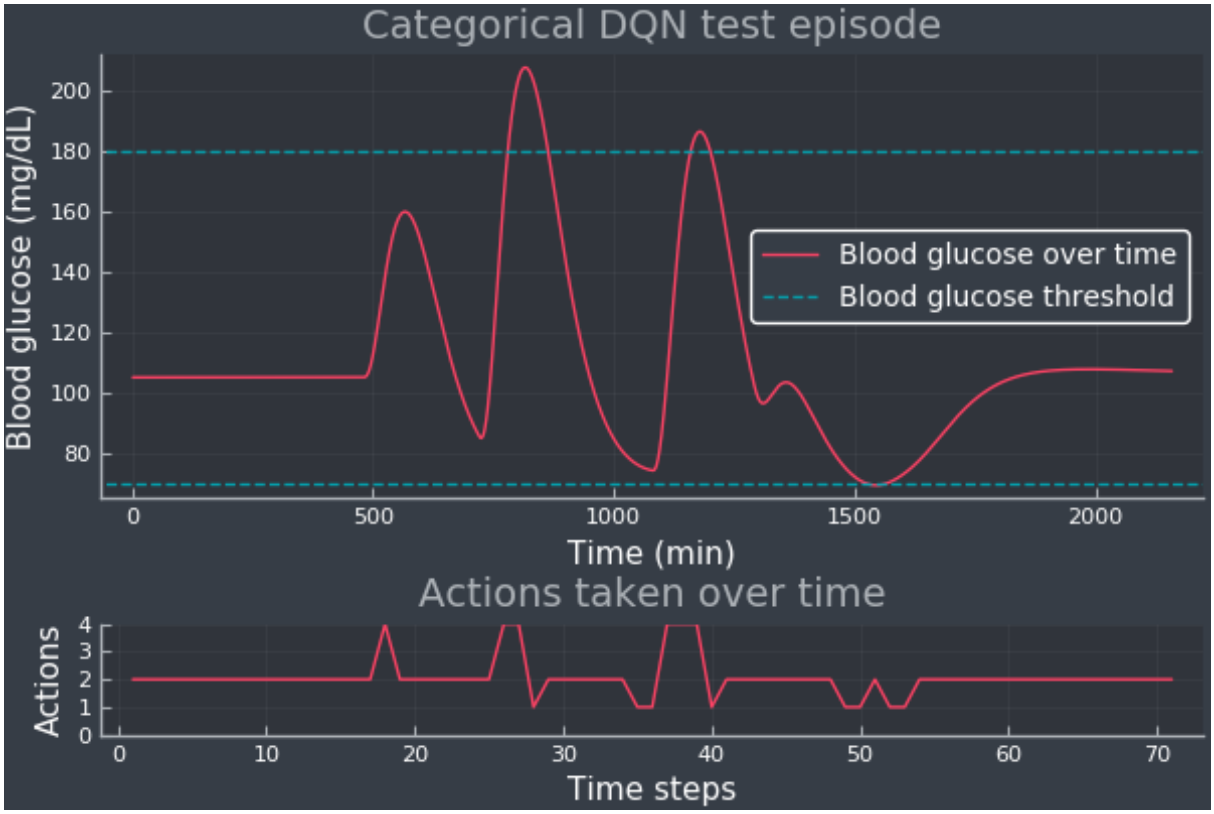

Figure 4.47: A categorical DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

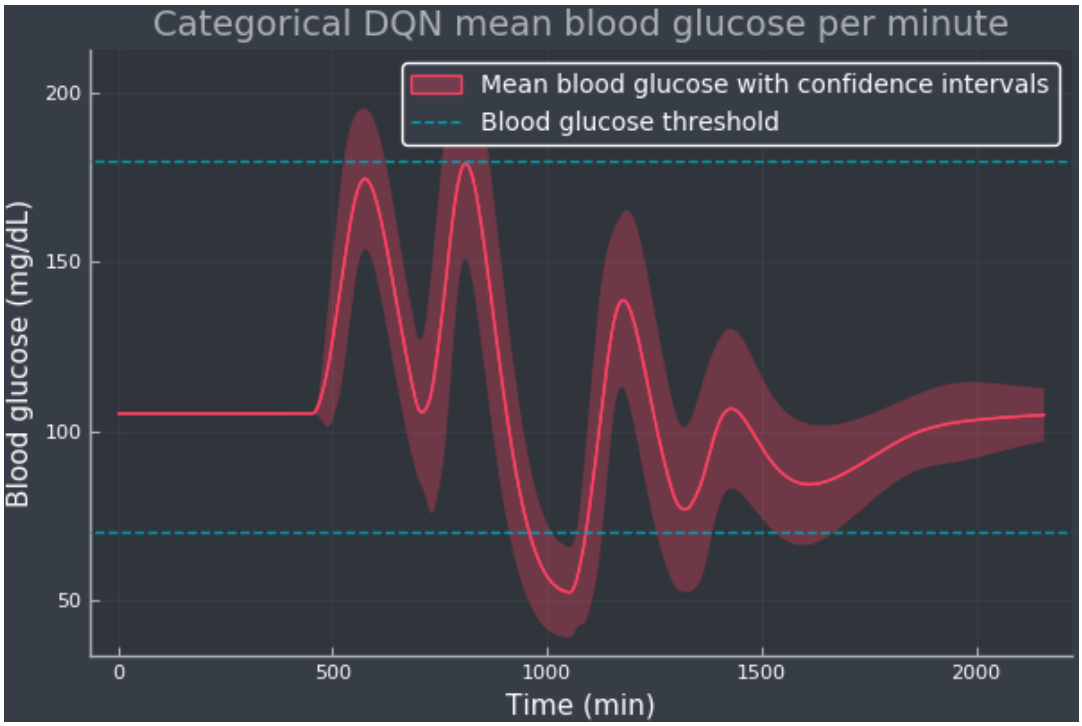

Figure 4.48: Categorical DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

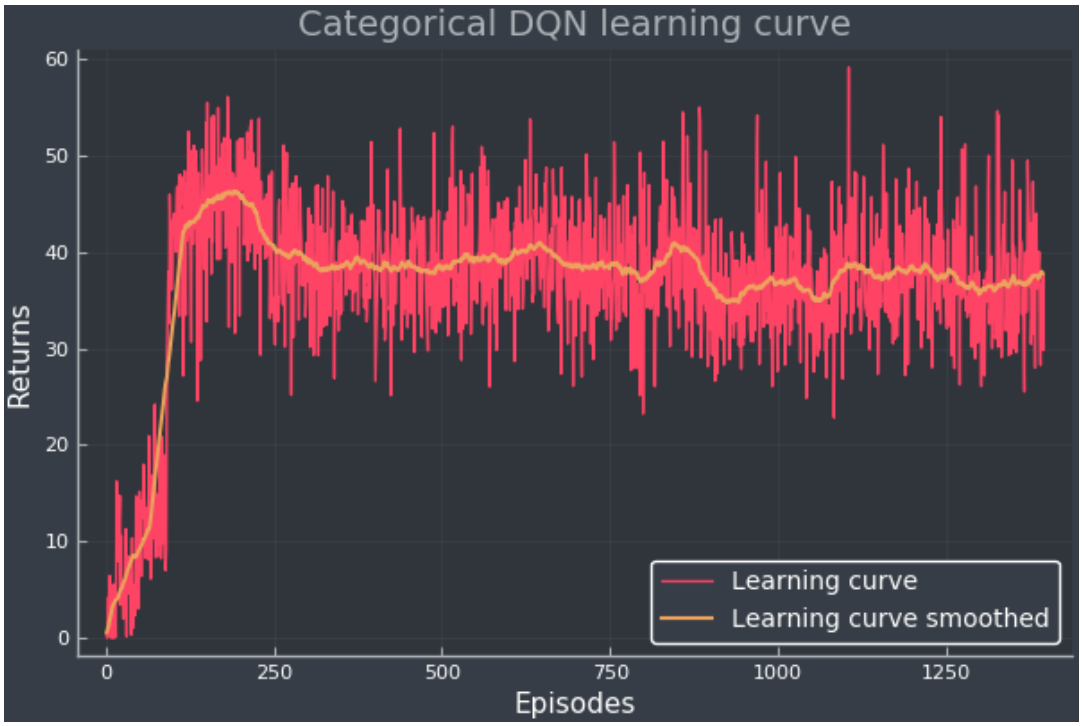

Figure 4.49: The categorical DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

Pretty much the same analysis can be applied to this version of the rainbow DQN like we did in experiment 1. The results displayed in Figures 5.50 - 5.52 are even worse than in experiment 1. First of all, in the test episode, it is clear that the agent is giving way too high basal rates at times, hence the low dip in the mean BG curve.

In both figure 4.51 and table 4.3 we see that σ is much larger than the other estimates. The TIR is also lower, with pretty high TBR and some percentage in the TAR as well. In an episodic setting, the TIR and TBR is worse than that. table 4.4 show that these estimates are 80.77% and 11.96% respectively, which is not wanted.

The learning curve in figure 4.52 is demonstrating both stale and poor learning capabilities by the agent. All the experience it got was from the first few iterations, and from there the agent didn't learn anything new, establishing that the algorithm is failing to converge to a better return near the end episodes.

Rainbow DQN in both experiment 1 and 2 has proved to not work as intended, and fails to achieve the goal. As discussed in the previous section, perhaps larger samples or a different network architecture would suffice. Again, there are many variables involved when designing a DRL algorithm, and rainbow DQN is especially comprehensive. In future experiments, more time and testing is required in order to succeed with such a broad-ranging algorithm.

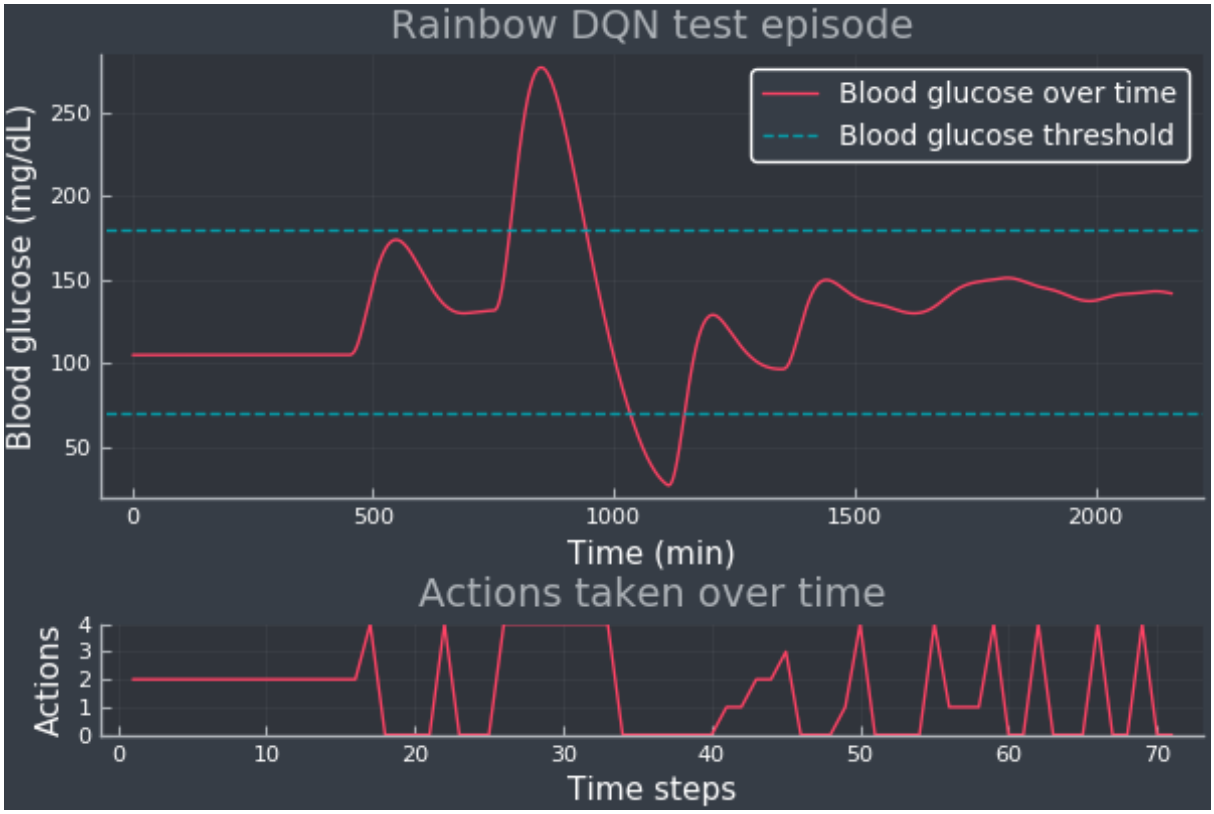

Figure 4.50: A rainbow DQN test episode showing the BG curve (upper panel) and the actions selected over time (lower panel). The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

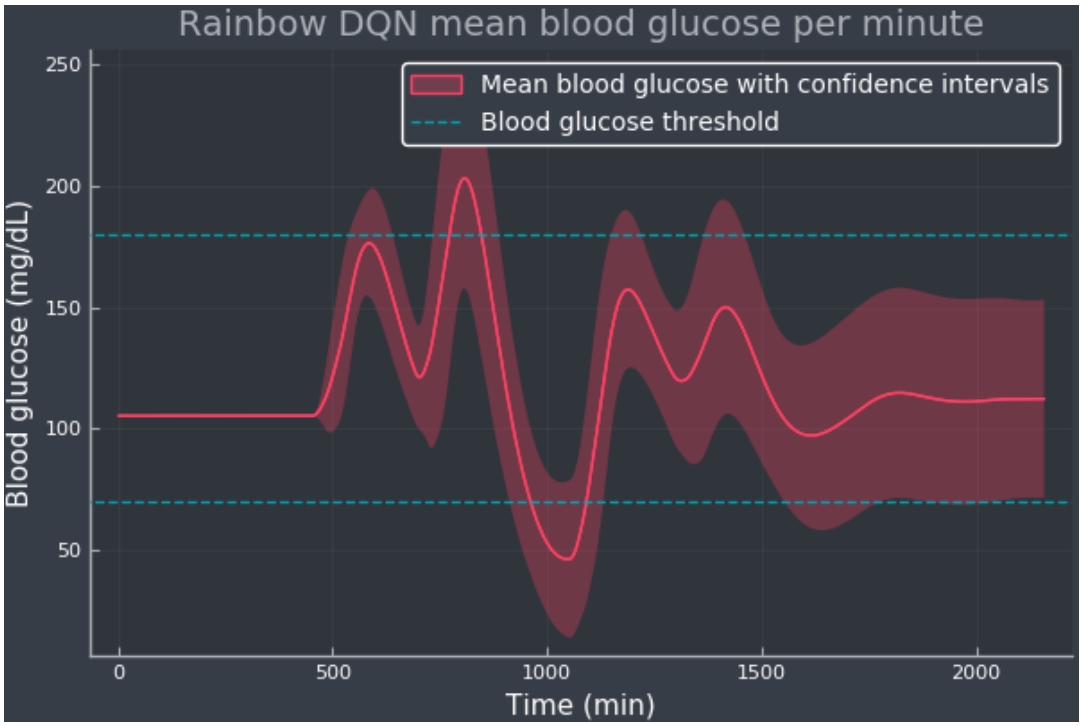

Figure 4.51: Rainbow DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

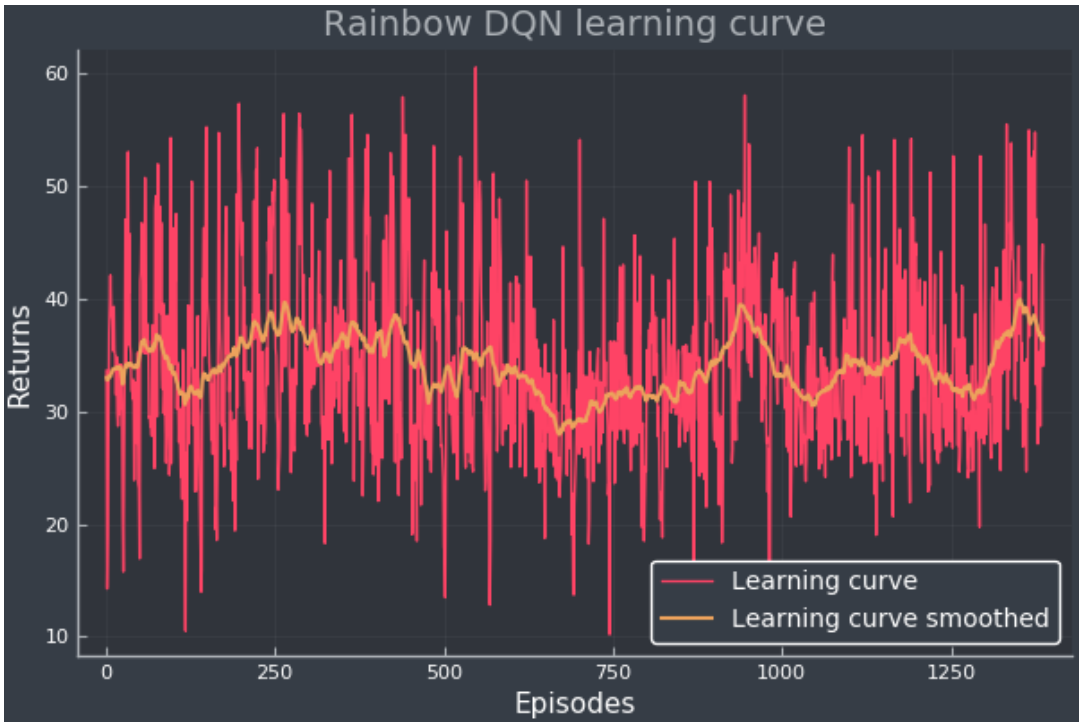

Figure 4.52: The rainbow DQN learning curve, displaying how the return evolves with training episodes. The red curve is the original return data and the orange curve is a smoothed version, which is easier to interpret.

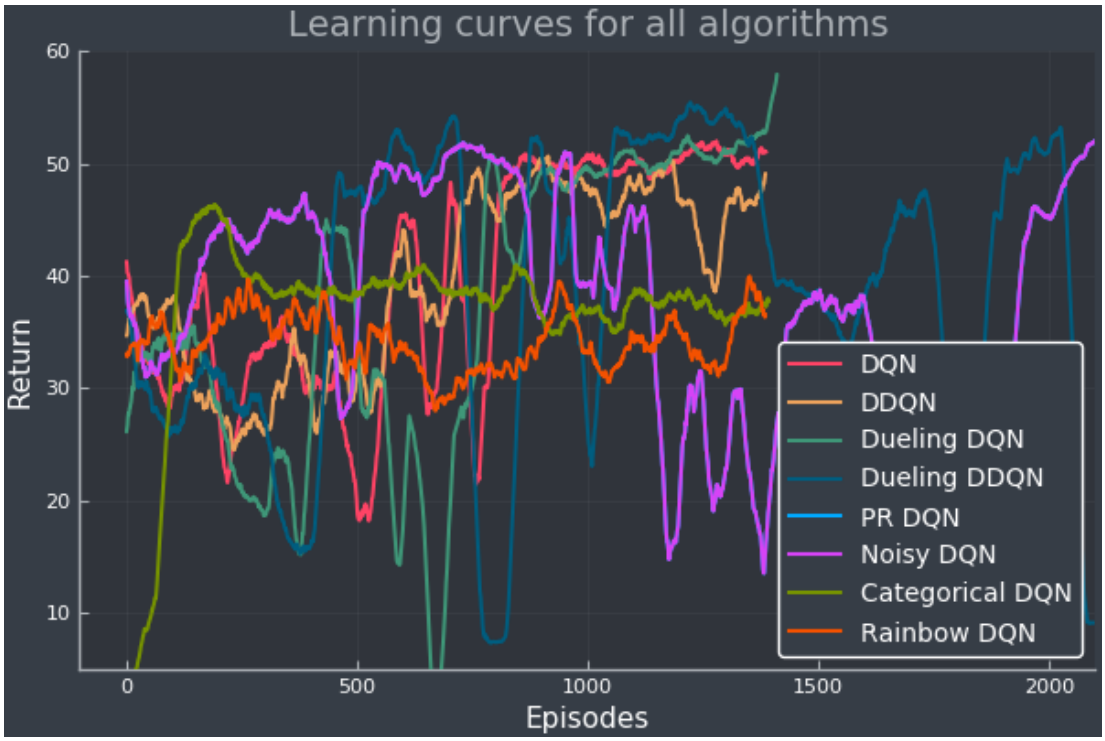

Figure 4.53: Experiment 2 - The learning curves for all the algorithms, showcasing how the return evolves with training episodes. All curves have been smoothed for easier interpretation and comparison. Dueling DDQN, PR DQN and noisy DQN have been cut off around the 2000 episodes mark, since it trained for a longer period compared to the other algorithms. The rest of the training was found to be redundant, which is discussed in this section. This is a similar reasoning as in experiment 1. As a general rule; the longer the curve, the faster episodes are being terminated.

As an arbitrary conclusion, this experiment has shown through its results that the extended version of the original action space isn't necessarily any better than just using three actions. In most cases it might have made the decisions harder for the agent and over-complicated the learning process. DDQN, PR DQN, noisy DQN, categorical DQN and rainbow DQN all failed the goal, that is to regulate the BG in the patient better than the baseline. Only dueling DQN managed to do so, with a TIR score of 97.04%, and DQN and dueling DDQN approximately matched the baseline's results. Same as in experiment 1, the DQN agent managed to learn the same strategy as the baseline, that is to only give the patient the optimal basal rate.

In the episodic setting, the baseline had the highest TIR score while DQN and dueling DQQN were close to this estimate. Since dueling DQN got the most optimal μ estimate, the mean estimate of TIR probably got affected by a larger meal than say the baseline. Since it was estimated over 2 weeks, the TIR estimated from the mean BG per minute is more worth in this case.

Analyzing the learning curves in figure 4.53 we can confirm that dueling DQN surpasses the rest. With most of the algorithms struggling to converge or keep the previously learned information intact, dueling DQN has a somewhat steep learning curve

with an average return of 50+ near the end episodes. Following the same reasoning as in experiment 1, some algorithms trained longer than others, meaning that many episodes were terminated early. The longer the learning curve, the faster episodes are being terminated. These curves belongs to dueling DDQN, PR DQN and noisy DQN, where only the last two had non-approved results.

Algorithm	TIR	TAR	TBR	μ	σ
Baseline	95.41%	4.59%	0.0%	124.00	33.84
DQN	95.23%	4.77%	0.0%	124.46	34.88
DDQN	93.80%	0.0%	6.20%	103.29	31.65
Dueling DQN	**97.04%**	2.96%	0.0%	**113.80**	34.03
Dueling DDQN	94.91%	5.09%	0.0%	119.61	**31.60**
Prioritized DQN	93.75%	0.65%	5.60%	107.04	33.70
Noisy DQN	94.17%	0.0%	5.83%	109.54	32.98
Categorical DQN	93.89%	0.0%	6.11%	106.28	32.28
Rainbow DQN	90.56%	3.43%	6.02%	118.35	42.50

Table 4.3: Experiment 2 - TIR, TAR and TBR of the mean BG per minute of 100 episodes. μ is the mean BG per episode and σ is the standard deviation of the BG per episode. Estimated for different DQN extensions. The best results are written in **bold text**, while the worst results are written in red text. Note that in the TAR and TBR column there are multiples of the same result, hence they are not highlighted in bold.

Algorithm	TIR	TAR	TBR
Baseline	**91.83%**	7.80%	**0.37%**
DQN	91.71%	6.92%	1.37%
DDQN	85.29%	**2.77%**	11.94%
Dueling DQN	90.45%	8.30%	1.25%
Dueling DDQN	91.64%	7.42%	0.94%
Prioritized DQN	90.24%	5.97%	3.80%
Noisy DQN	88.92%	4.24%	6.84%
Categorical DQN	89.75%	3.95%	6.30%
Rainbow DQN	80.77%	7.27%	11.96%

Table 4.4: Experiment 2 - Mean TIR, TAR, and TBR of 14 episodes of BG data. Estimated for different DQN extensions. The best results are written in **bold text**, while the worst results are written in red text.

4.2.3 Experiment 3 - Meal Disturbances

In this experiment we want to investigate the behaviour of trained agent's ability of dealing with imprecision in the meals. We would expect to see that the agent selects high insulin actions right after the meal where the bolus was skipped.

It is worth noting that all the mean BG curves in Figures 5.54 - 5.61 are so similar to the ones in experiment 1. One can hardly tell the difference, so table 4.5 is useful for checking scores and estimates to assist in comparing them.

Starting with DQN we immediately that the TIR has dropped by roughly 3% and the other estimates have all increased by a bit. The variance in the insulin action is very small (close to zero), which means that with high probability it went the same route as the baseline, only now meals were skipped with a 10% probability, leading to an increase in the standard deviation BG per episode.

With DDQN we can see that it is pretty much the same. Note that σ_A is larger than that of DQN, meaning it there is moderate variation in insulin actions. The reason why this algorithm does not change that drastically is because the agent does not care. From its learning curve we observed that the curve went to a plateau very early on in the training.

Changing focus over to dueling DQN we see that there is improvement in the TBR department, and instead an increase in TAR and slightly less TIR. This implies that the agent has improved its perception of insulin actions and stops the insulin pump when it needs to. We can see that in figure 4.56 that the usual decrease in BG between lunch and dinner has now been managed, at the cost of higher BG during lunch. The same can be said about dueling DDQN, except for the fact that this algorithm already knew how to regulate the BG in the patient just fine. Notice that algorithms have greater σ_A estimates compared to the previous ones. We know that from their test episodes, the action selection had a sawtooth-like pattern, meaning a more rapid variation in insulin actions.

For PR DQN, noisy DQN and categorical DQN we see that the TIR scores are lowered by $\approx 1\%$ and with a slight increase in TAR. The mean BG per episode has also increase as a consequence and all in all the results looks very similar. Noisy DQN also got the highest TIR score once again, and with a lower σ_A estimate perhaps implying that less insulin action variation can be just as beneficial as going from action 0 to 2 rapidly.

Rainbow DQN seems to be unaffected by the meal disturbances at first glance. With the lowest TBR score, it nearly has the same results from experiment 1.

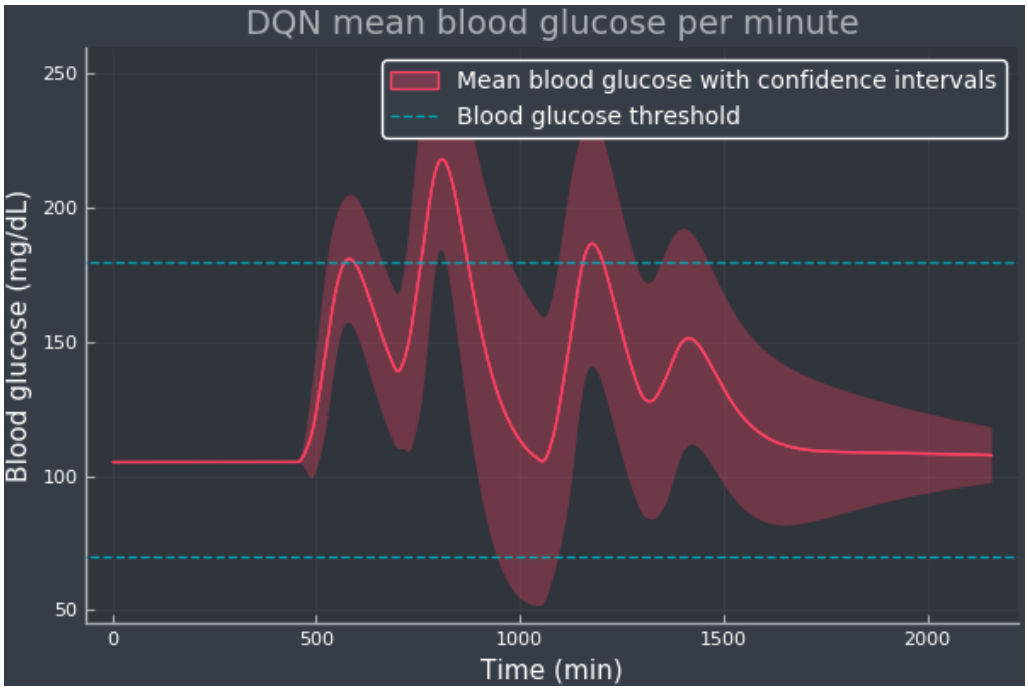

Figure 4.54: DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

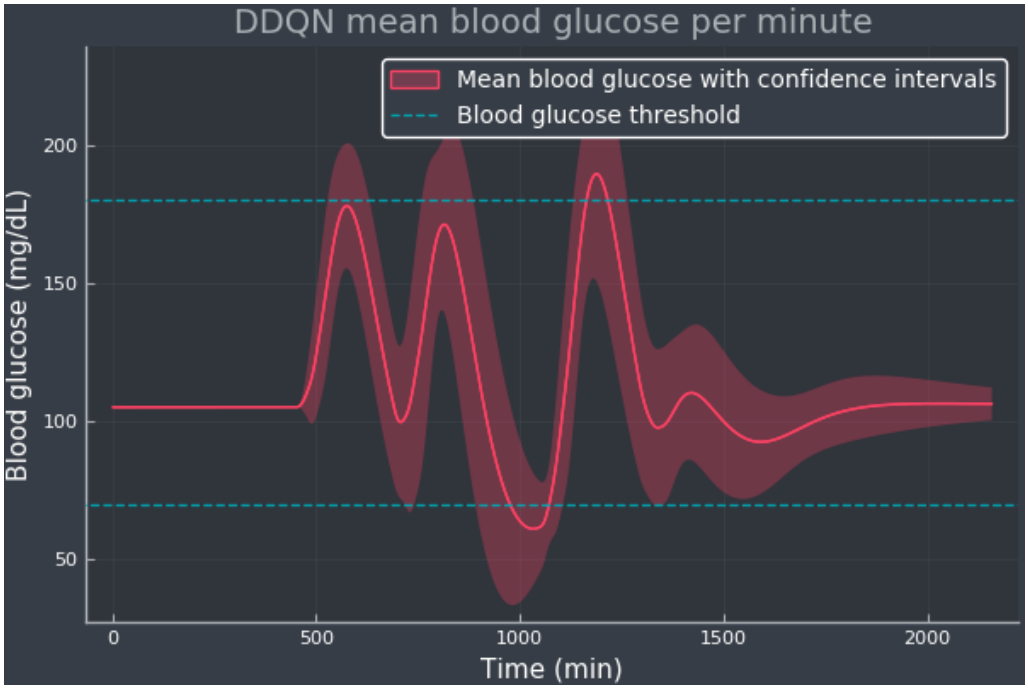

Figure 4.55: DDQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

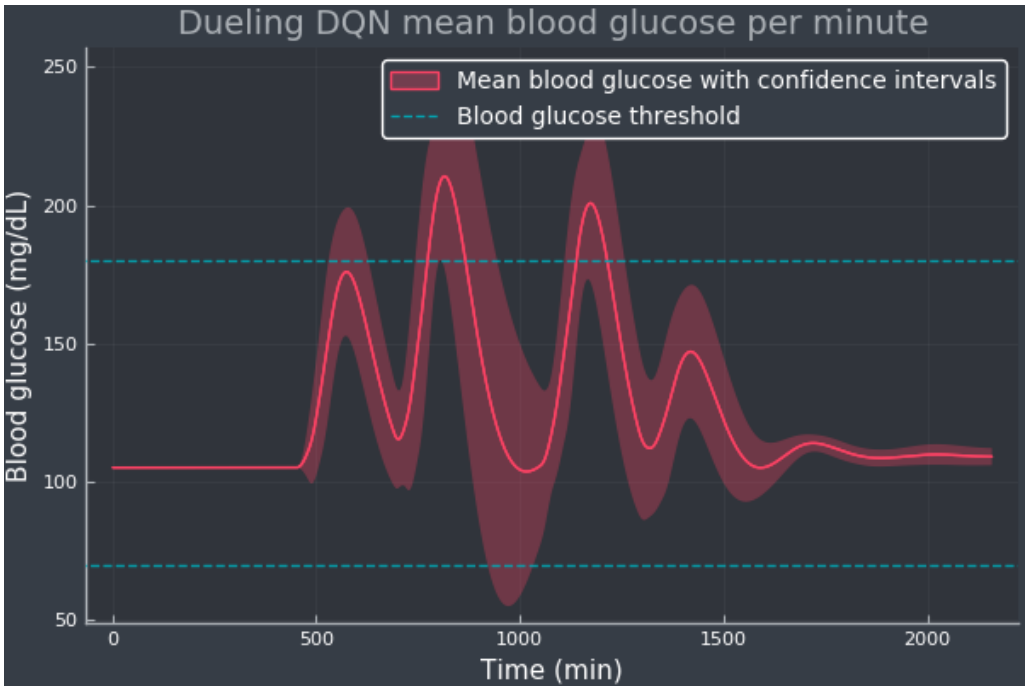

Figure 4.56: Dueling DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

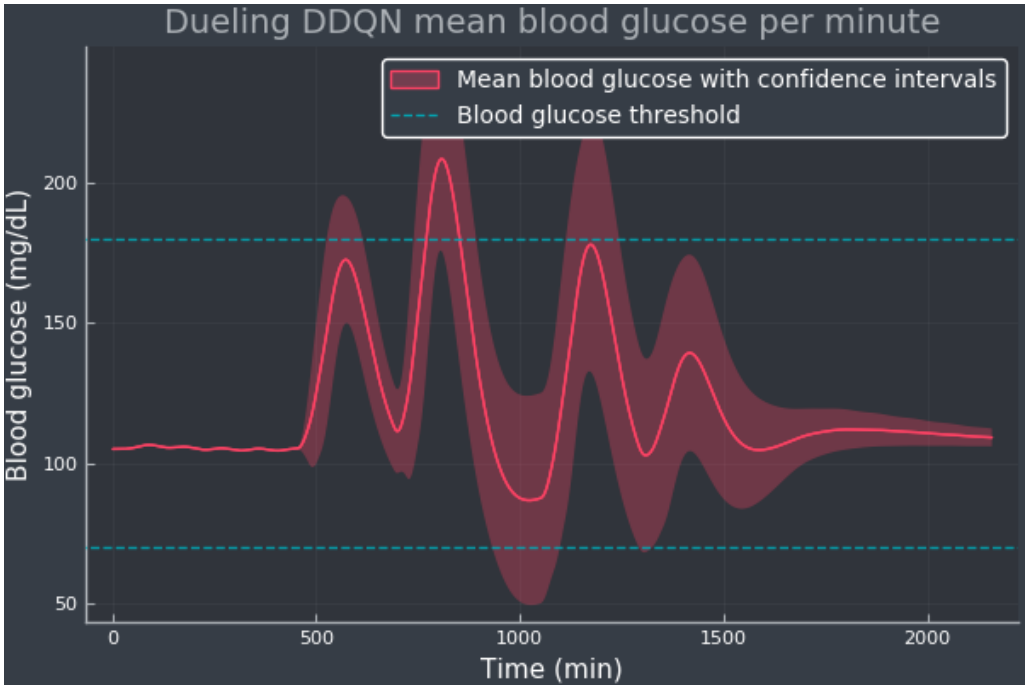

Figure 4.57: Dueling DDQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

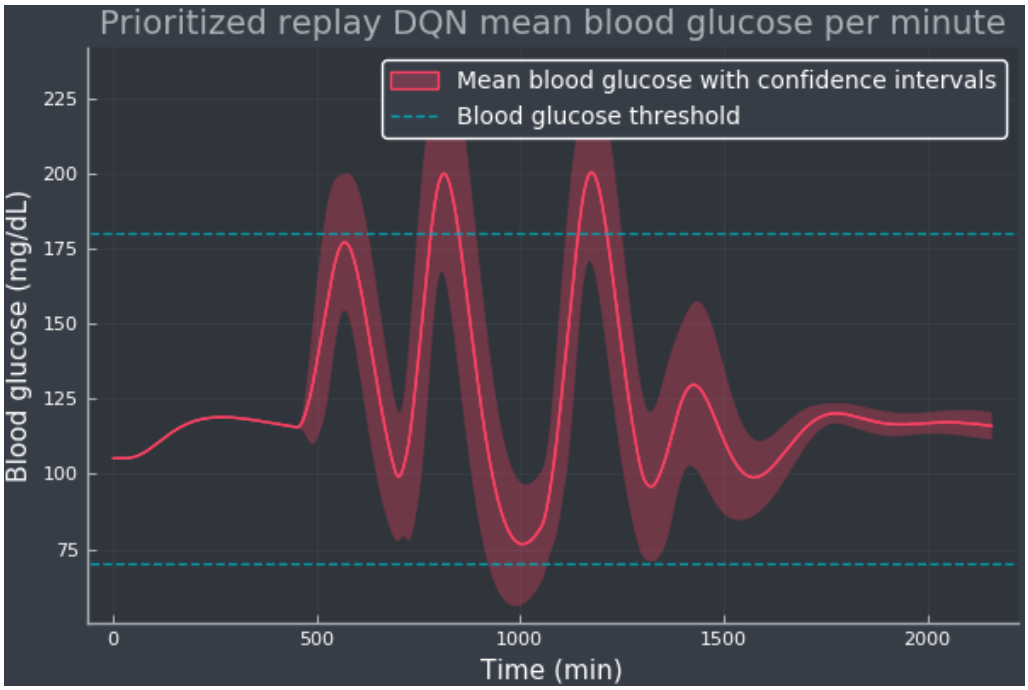

Figure 4.58: Prioritized replay DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

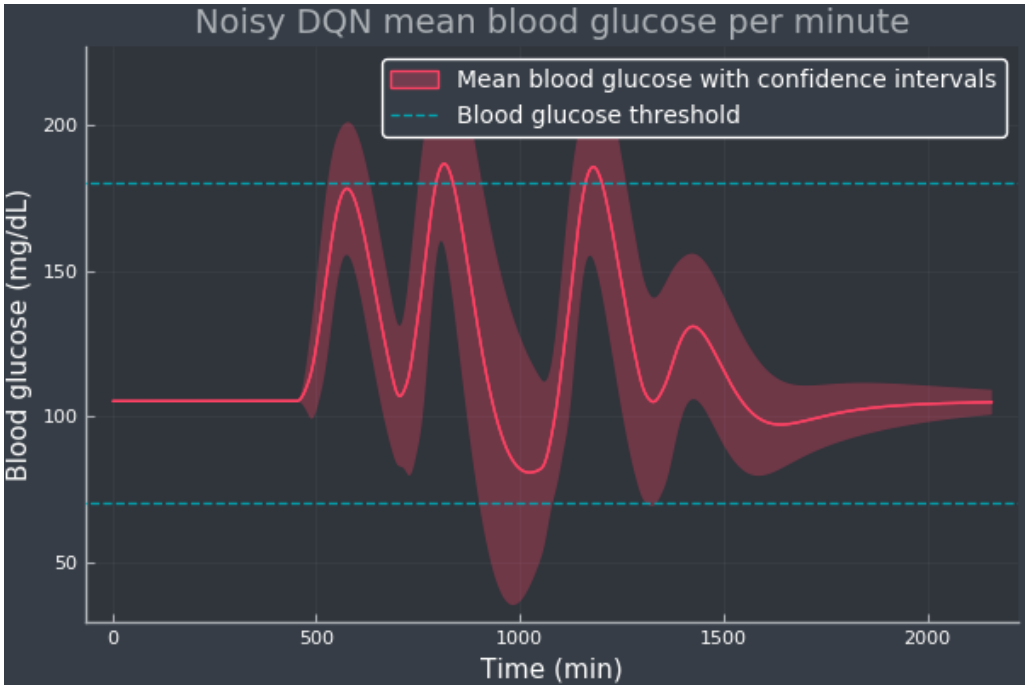

Figure 4.59: Noisy DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

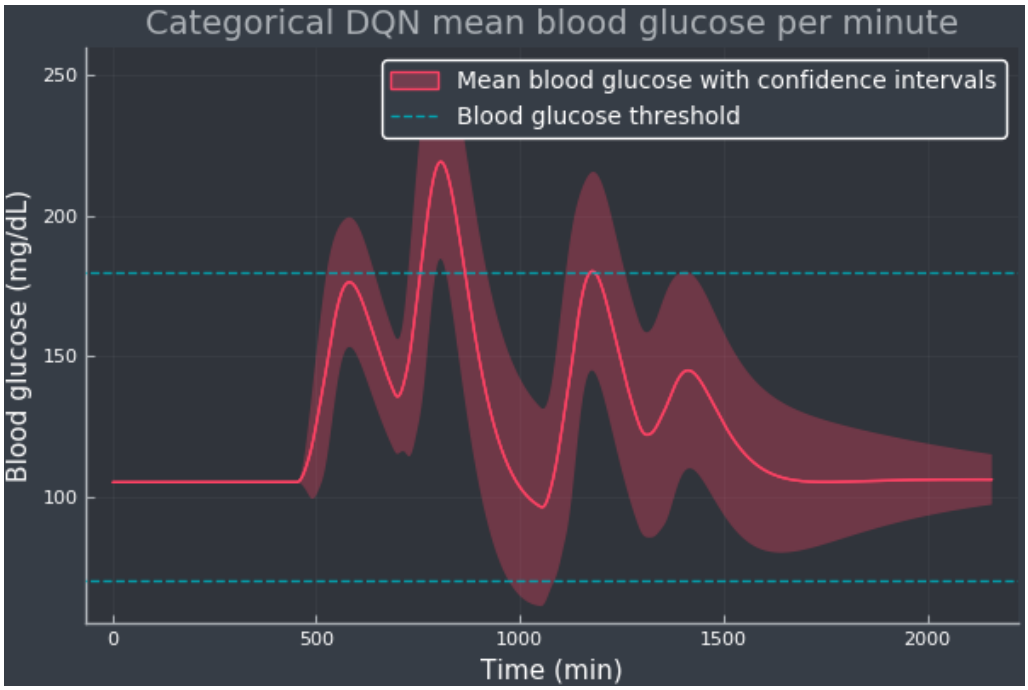

Figure 4.60: Categorical DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

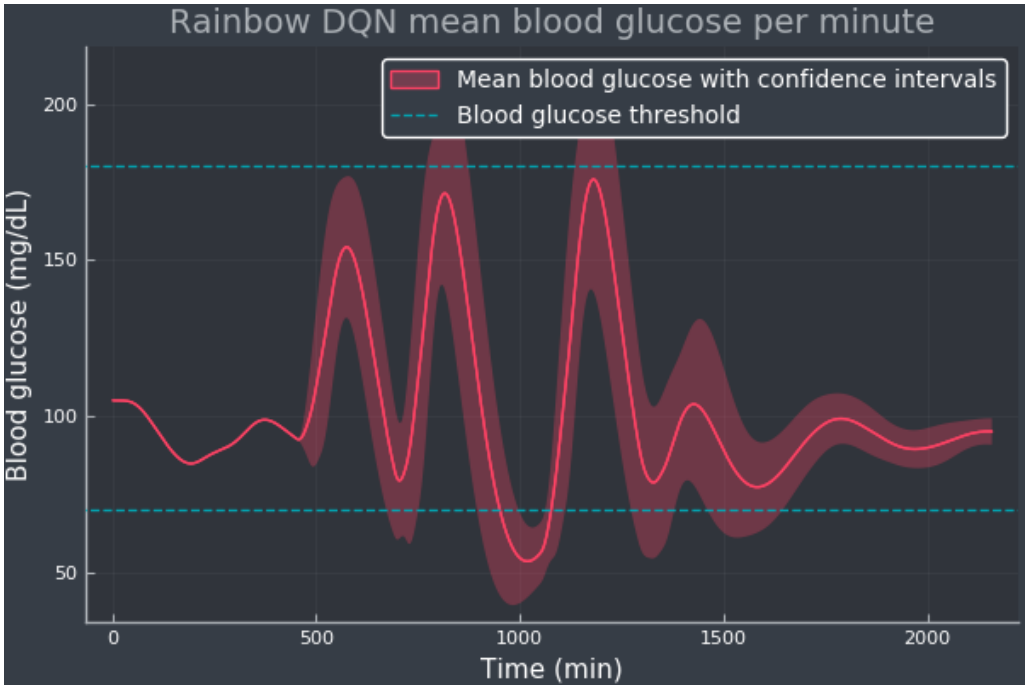

Figure 4.61: Rainbow DQN mean BG per minute with confidence bands (shaded red area), representing the standard deviation, simulated for 100 episodes and with a 10% probability to skip meal boluses. The blue dotted lines indicate the normoglycemic range and it is fixed at 70-180 mg/dL.

Algorithm	TIR	TAR	TBR	μ	σ	σ_A
Baseline	94.49%	5.51%	0.0%	125.95	36.47	0.0
DQN	91.57%	8.43%	0.0%	129.54	39.01	$8.88 \cdot 10^{-16}$
DDQN	93.15%	2.45%	4.40%	113.33	35.59	4.72
Dueling DQN	92.18%	7.82%	0.0%	126.11	35.65	5.85
Dueling DDQN	96.16%	3.84%	0.0%	123.41	34.43	9.06
Prioritized Replay DQN	93.61%	6.39%	0.0%	123.50	34.41	6.92
Noisy DQN	**96.20%**	3.8%	0.0%	117.73	34.97	4.50
Categorical DQN	94.49%	5.51%	0.0%	125.95	36.47	8.88
Rainbow DQN	94.21%	**0.0%**	5.79%	101.41	**33.57**	7.91

Table 4.5: Experiment 3 - TIR, TAR and TBR of the mean BG per minute of 100 episodes, and accounting for the meal bolus skips. μ is the mean BG per episode, σ is the standard deviation of the BG per episode and σ_A is the insulin action standard deviation per episode. Estimated for different DQN extensions. The best results are written in **bold text**, while the worst results are written in red text. Note that in the TBR column there are multiples of the result value, hence they are not highlighted in bold.

Chapter 5

Conclusion

This chapter aims to provide concluding remarks, that tie together all the results and discussions presented in the previous chapter, to round off this thesis.

5.1 Conclusion

In this thesis, we have implemented and evaluated several DQN algorithms for the task of regulating the BG in T1D patients. These seven methods were introduced in Chapter 4, where six of them aims to improve upon the original DQN algorithm. The methods were compared to a baseline, where only the optimal basal rate was given to the patient. Our results show that some DQN algorithms are capable of managing this task and out-performing the baseline results, where our best models were dueling DDQN and noisy DQN for experiments 1 and 3, and dueling DQN for experiment 2. However, many models in experiment 2 failed the BG regulation criteria, and generally some algorithms did not perform better than the baseline. These findings implies that there is still room for improvement, both in implementation of the methods and the environment setup.

Our experiments show that the DQN algorithms favor more an action-space with three actions, rather an action space with five. The extended action space indicate complications for the agents during the training process, resulting in undesirable learning and high insulin action tendencies. In this setting, DQN and dueling DDQN was found to be equal to the baseline, and dueling DQN were the only algorithm to out-perform it. Compared to the smaller action space, the experiments showed more promising results. DQN and categorical DQN learned a matching strategy to that of the baseline, and dueling DDQN and noisy DQN out-performed it.

When skipping meal boluses we found that the overall TIR estimates were lowered by a small percentage as a result of slightly higher BG (TAR). The TBR were virtually unchanged and the general conclusion is that this experiment led to more models being unsuccessful in meeting the BG regulation criteria. Generally, the models' ability to adapt to this situation were lacking and improvement in this territory is left to future work. Since these situations were not seen during training, we therefore assume that

the agents were not able generalize properly.

Given the current experimental setup, the best DQN models out-performed the baseline. Further experiments need to be carried out to fully validate DQN as a realistic algorithm for the AP. We do however, believe that this is a step in the right direction.

The algorithms DDQN, PR DQN and rainbow DQN under-performed in our experiments, and their suggested improvements from the discussion in the previous chapter is left to future work.

5.2 Future Research

Ideas for future work with the DQN algorithms include:

- Experiment with different types of NN architectures. In some instances of the experiments it was believed that a different type of NN and/or different types of layers/activation functions would lead to better learning, especially in the larger action space setting.

- Test the simulator on policy based RL algorithms. Here one should convert the action space from discrete to continuous.

- Change the simulation time and/or time resolution, which in turn changes the state space and the episode length.

- Change the size of the state and/or state space, for example, use the last 24-hours of BG data and insulin data, similar to the setup of Fox and Wiens (2019) [23].

- Develop and explore with quantum RL to implement a quantum DQN algorithm. Past contributions in this field has been occasionally done [118, 119, 120, 121] and holds exciting and new opportunities for DQN algorithms.

Appendix A

Glucoregulatory System Models

In appendix A.1 we describe the Hovorka model and its equations. Appendix A.2 states other relevant models and strategies used in MPC.

A.1 The Hovorka Model

In this section, the methodology of the Hovorka model [122]-[125] is described. It simulates a patient with T1D and consists of three two compartment models: Food absorption from meals, subcutaneous insulin injection and absorption, and the glucoregulatory system (glucose kinetics). The model has two inputs: Meal intakes and insulin infusions.

Figure A.1 summarizes the whole system in a diagram. For a definition of all the parameters and variables used in the model, please consult to Table A.1 and Table A.2.

A.1.1 Food Absorption

In this subsection we will consider the model describing food absorption [126]-[129] and its conversion to glucose. It is a subsystem consisting in a two compartment model.

Let $d(t)$ [g/min] be the meal input, which is the amount of carbohydrate (CHO) per minute a virtual patient at time t consumes. Then the CHO input rate, $D(t)$ [mmol/min], may be related to $d(t)$ by

$$D(t) = \frac{1000}{M_{wG}} d(t), \tag{A.1}$$

where M_{wG} [g/mmol] is the molecular weight of glucose.

Now, consider a small time window Δt. Then the amount of glucose that enters the first compartment $D_1(t + \Delta t)$ [mmol] is $A_G D(t)\Delta t$, where A_G is the CHO bioavailability. After the glucose in the first compartment is transferring over to the second, the output is $D_1(t)$ distributed over a time τ_D [min]

$$g_1(t) = \frac{D_1(t)}{\tau_D}\Delta t.$$

Once a time τ_D has passed, all the glucose will be out of the system. According to the mass balance equation by Himmelblau and Riggs (2004) [130], the accumulated mass in a system m_A during a time interval is equal to the mass that entered the system m_I, minus the mass that left the system m_O, plus the mass generated by the system m_G, minus the mass consumed by the system m_C:

$$m_A = m_I - m_O + m_G - m_C. \tag{A.2}$$

Since there is no generated or consumed glucose by the system here, equation A.2 gives us

$$m_A = m_I - m_O = D_1(t + \Delta t) - D_1(t) = A_G D_1(t)\Delta t - \frac{D_1(t)}{\tau_D}\Delta t.$$

If we divide by Δt and take the limit $\Delta t \to 0$, we can obtain a differential equation that describes the glucose dynamics in the first compartment of the food absorption subsystem $D_1(t)$. But first, let us find the equivalent equation for the second compartment.

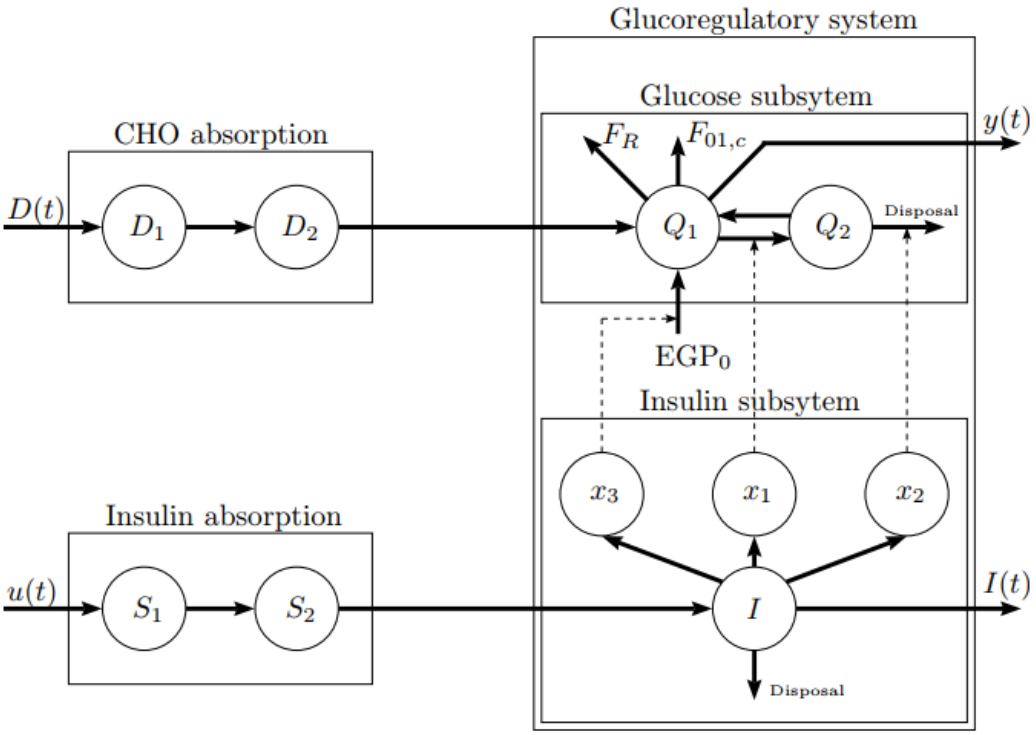

Figure A.1: A simple diagram of the Hovorka model. The diagram was obtained from Mosching's master's thesis (2016) [50], which was inspired by Hovorka et al. (2004) [122] and Boiroux et al. (2010) [124].

The mass input for the system in the second compartment is the same one that exited first compartment, $D_1(t)/\tau_D$. Likewise for the output mass it is $D_2(t)$ [mmol] distributed over τ_D, which is $D_2(t)/\tau_D$. The model describing the two compartments is then

$$\frac{dD_1(t)}{dt} = A_G D(t) - \frac{D_1(t)}{\tau_D} \tag{A.3a}$$

$$\frac{dD_2(t)}{dt} = \frac{D_1(t)}{\tau_D} - \frac{D_2(t)}{\tau_D} \tag{A.3b}$$

$A_G = 0.8$ is a constant describing the utilization of CHOs. The time constant $\tau_D = 40$ min is an indicator of how long it takes from the meal consumption until the glucose is present in the blood stream. The glucose absorption rate $U_G(t)$ [mmol/min] is then described by

$$U_G(t) = \frac{D_2(t)}{\tau_D}. \tag{A.4}$$

A.1.2 Glucose Subsystem

After the meal-related glucose has been absorbed, it enters the glucose subsystem, where we model the BG dynamics with two compartments, $Q_1(t)$ [mmol] and $Q_2(t)$ [mmol]. $Q_1(t)$ represents the glucose in the main blood stream, while $Q_2(t)$ represents glucose in the peripheral tissues.

The glucose subsystem model is represented by two equations:

$$\frac{dQ_1(t)}{dt} = U_G(t) - F_{01,c}(t) - F_R(t) - \Xi(t) + EGP_0(1 - x_3(t)) \tag{A.5a}$$

$$\frac{dQ_2(t)}{dt} = x_1(t)Q_1(t) - (k_{12} + x_2(t))Q_2(t) \tag{A.5b}$$

Firstly, looking at dependencies for equation A.5a we have $U_G(t)$, which as mentioned before is the glucose absorption from the meals. $F_{01,c}(t)$ [mmol/min] is the glucose consumption of the central nervous system (CNS) modelled as

$$F_{01,c}(t) = \begin{cases} F_{01}, & y(t) \geq 4.5\text{mmol/L}, \\ F_{01}y(t)/4.5, & \text{otherwise}, \end{cases} \tag{A.6}$$

where $y(t)$ [mmol/L] is the glucose concentration expressed by $Q_1(t)$ and the glucose distribution volume V_G [L]:

$$y(t) = G(t) = \frac{Q_1(t)}{V_G}. \tag{A.7}$$

Note that V_G depends on the patient's body weight, BW [kg]. $F_R(t)$ [mmol/min] is the glucose excretion rate modeled as

$$F_R(t) = \begin{cases} 0.003(y(t) - 9)V_G, & y(t) \geq 9\text{mmol/L}, \\ 0, & \text{otherwise.} \end{cases} \tag{A.8}$$

$\Xi(t)$ is a substitution for

$$\Xi(t) = x_1(t)Q_1(t) - k_{12}Q_2(t), \tag{A.9}$$

where $x_1(t)$ is the insulin action that has influence on glucose transport and distribution, and k_{12} [1/min] is the transfer rate from the blood to the tissues. Lastly, EGP_0 [mmol/min] is endogenous glucose production (EGP) by the liver at zero insulin level, and $x_3(t)$ is the insulin action that has influence on EGP.

For equation A.5b, the only new dependency is $x_2(t)$, which is the insulin effect on glucose disposal.

A.1.3 Insulin Absorption

This subsection covers a two compartment model describing the insulin absorption rate of subcutaneously administered short acting insulin [124]. To give the patient insulin subcutaneously means injecting it under the skin. Such models have been studied before [131], and its administration has proved to be much safer and more practical [132, 133], compared to more direct methods. This refers to injecting insulin intravenously, meaning directly into the blood stream [125].

Similar to meals, the insulin absorption model is given by

$$\frac{dS_1(t)}{dt} = u(t) - \frac{S_1(t)}{\tau_S} \tag{A.10a}$$

$$\frac{dS_2(t)}{dt} = \frac{S_1(t)}{\tau_S} - \frac{S_2(t)}{\tau_S} \tag{A.10b}$$

where $u(t)$ [mU/min] is the amount of insulin injected, $S_1(t)$ [mU] and $S_2(t)$ [mU] are the amounts of insulin in the two compartments, and τ_S [min] is the time constant associated with subcutaneous-to-intravenous absorption set to 55 min. The insulin absorption rate in the blood stream $U_I(t)$ [mU/min] is then

$$U_I(t) = \frac{S_2(t)}{\tau_S}. \tag{A.11}$$

The one drawback of this model is that $\tau_S > \tau_D$, which means that the insulin will reach the blood stream slower than the glucose. All in all, this results in a delayed effect when using a controller [125].

A.1.4 Insulin Subsystem

The absorption rate of insulin in the blood stream enters the insulin subsystem, which is part of the glucoregulatory system. Plasma insulin concentration $I(t)$ [mU/L] is the evolving according to the differential equation

$$\frac{dI(t)}{dt} = \frac{U_I(t)}{V_I} - k_e I(t). \tag{A.12}$$

Here V_I [L] is the insulin distribution volume and k_e [1/min] is the insulin elimination rate from plasma.

Previously, when looking at the glucose subsystem, we studied that the three insulin action states is ruled by: influence on transport and distribution $x_1(t)$, phosphorylation of glucose in adipose tissue $x_2(t)$ (glucose disposal), and EGP in the liver $x_3(t)$ [124, 125]. These quantities are estimated using the following differential equations:

$$\frac{dx_1(t)}{dt} = -k_{a1}x_1(t) + k_{b1}I(t), \tag{A.13a}$$
$$\frac{dx_2(t)}{dt} = -k_{a2}x_2(t) + k_{b2}I(t), \tag{A.13b}$$
$$\frac{dx_3(t)}{dt} = -k_{a3}x_3(t) + k_{b3}I(t) \tag{A.13c}$$

New dependencies here are the constants k_{ai} [1/min] and k_{bi} [L/(mU·min)], $i = 1, 2, 3$. k_{ai} are the deactivation rates and have been experimentally estimated [122, 125] to have a mean value of $k_{a1} = 0.006$, $k_{a2} = 0.06$ and $k_{a3} = 0.03$. The other constants k_{bi} can be calculated by using the three insulin sensitivities:

$$k_{b1} = S_{I,T}k_{a1}, \tag{A.14a}$$
$$k_{b2} = S_{I,D}k_{a2}, \tag{A.14b}$$
$$k_{b3} = S_{I,E}k_{a3} \tag{A.14c}$$

Here the insulin sensitivities are: $S_{I,T}$ of transport/distribution, $S_{I,D}$ of disposal and $S_{I,E}$ of EGP.

Factors like physical activity and MESS can influence the insulin sensitivities [134], and therefore these sensitivities will vary from person to person [125].

A.1.5 Parameters and Variables

The constants and variables reviewed in the previous sections are summarized in Table A.1 and Table A.2. Four of these parameters, EGP_0, F_{01}, V_G and V_I, depend on the body weight BW [kg] of a person. The mean values of these parameters is therefore listed only for one kg, and the actual values can be estimated by multiplying the table values with BW. For a 70 kg person, these parameters become:

$$EGP_0 = 0.0161 \ \frac{\text{mmol}}{\text{min}}/\text{kg} \cdot 70 \text{ kg} = 1.1270 \text{ mmol/min}$$
$$F_{01} = 0.0097 \ \frac{\text{mmol}}{\text{min}}/\text{kg} \cdot 70 \text{ kg} = 0.6790 \text{ mmol/min}$$
$$V_G = 0.16 \text{ L/kg} \cdot 70 \text{ kg} = 11.2 \text{ L}$$
$$V_I = 0.12 \text{ L/kg} \cdot 70 \text{ kg} = 8.4 \text{ L}$$

Variable	Definition	Unit
$D(t)$	Amount of CHO intake	mmol/min
$D_1(t)$	Amount of glucose in compartment 1	mmol
$D_2(t)$	Amount of glucose in compartment 2	mmol
$Q_1(t)$	Amount of glucose in the main blood stream	mmol
$Q_2(t)$	Amount of glucose in peripheral tissues	mmol
$y(t)$	Glucose concentration	mmol/L
$I(t)$	Plasma insulin concentration	mU/L
$S_1(t)$	Amount of insulin in compartment 1	mU
$S_2(t)$	Amount of insulin in compartment 2	mU
$u(t)$	Amount of insulin injected	mU/min
$x_1(t)$	Influence on glucose transport and distribution	-
$x_2(t)$	Influence on glucose disposal	-
$x_3(t)$	EGP	-

Table A.1: Model variables as summarized in [122] - [125], with inspiration from [50].

Parameter	Definition	Mean value	Unit
A_G	CHO bioavailability	0.8	-
k_{12}	Transfer rate	0.066	1/min
k_{a1}	Deactivation rate	0.006	1/min
k_{a2}	Deactivation rate	0.06	1/min
k_{a3}	Deactivation rate	0.03	1/min
k_e	Insulin elimination from plasma	0.138	1/min
τ_D	Glucose absorption constant	40	min
τ_S	Insulin absorption constant	55	min
EGP_0	Endogenous glucose production (EGP) at zero insulin	$0.0161 \cdot BW$	mmol/(kg·min)
F_{01}	CNS glucose consumption	$0.0097 \cdot BW$	mmol/(kg·min)
V_G	Glucose distribution volume	$0.16 \cdot BW$	L
V_I	Insulin distribution volume	$0.12 \cdot BW$	L
$S_{I,D}$	Insulin sensitivity of disposal	8.2×10^{-4}	L/mU
$S_{I,E}$	Insulin sensitivity of EGP	520×10^{-4}	L/mU
$S_{I,T}$	Insulin sensitivity of transport/distribution	51.2×10^{-4}	L/mU

Table A.2: Model constants and parameters as described in [122] - [125]. Four of these are dependent on the patient's weigth BW: EGP_0, F_{01}, V_G and V_I. One can obtain the actual value of a patient by multiplying the table mean value with BW.

In Europe, the unit for glucose concentration is mmol/L, while in America the unit is mg/dL [124]. One can convert between the two units by using the molecular weight of glucose ($C_6H_{12}O_6$), which is estimated to $M_{wG} = 180.1577$ g/mol:

$$G^*(t) = \frac{M_{wG}}{10} \cdot G(t), \tag{A.15}$$

where $G(t)$ has units mmol/dL and $G^*(t)$ has units mg/dL. Note that when converting

the units of M_{wG}, g/mol can automatically be changed to mg/mmol.

Hovorka et al. (2004) [122] re-estimated the parameters EGP_0, F_{01}, $S_{I,T}$, $S_{I,D}$, $S_{I,E}$ and τ_S using iterative Bayesian techniques. This is because they were considered to fluctuate between and within the patients [50].

Bibliography

[1] International Diabetes Federation, 2019. About Diabetes. URL: https://www.idf.org/aboutdiabetes/what-is-diabetes.html

[2] Holt, R. I., Cockram, C., Flybjerg, A., & Goldstein, B. J. (Eds.). (2017). Textbook of diabetes. John Wiley & Sons.

[3] International Diabetes Federation, 2017. http://www.diabetesatlas.org.

[4] Tuomilehto, J. (2013). The emerging global epidemic of type 1 diabetes. Current diabetes reports, 13(6), 795-804.

[5] You, W. P., & Henneberg, M. (2016). Type 1 diabetes prevalence increasing globally and regionally: the role of natural selection and life expectancy at birth. BMJ Open Diabetes Research and Care, 4(1), e000161.

[6] Paddock, Catharine. Diabetes: Synthetic beta cells could lead to skin patch treatment. https://www.medicalnewstoday.com/articles/319913.php.

[7] Ali, M. K., Weber, M. B., Narayan, K. V., & Cockram, C. (2010). The global burden of diabetes. Textbook of Diabetes, 4, 69-84.

[8] Narayan, K. V., Gregg, E. W., Fagot-Campagna, A., Engelgau, M. M., & Vinicor, F. (2000). Diabetesa common, growing, serious, costly, and potentially preventable public health problem. Diabetes research and clinical practice, 50, S77-S84.

[9] Wild, S., Roglic, G., Green, A., Sicree, R., & King, H. (2004). Global prevalence of diabetes: estimates for the year 2000 and projections for 2030. Diabetes care, 27(5), 1047-1053.

[10] International Diabetes Federation, 2019. About Diabetes - Diabetes complications. URL: https://www.idf.org/aboutdiabetes/complications.html

[11] International Diabetes Federation, 2019. About Diabetes - Type 2 diabetes. URL: https://www.idf.org/aboutdiabetes/type-2-diabetes.html

[12] Turksoy, K., Bayrak, E. S., Quinn, L., Littlejohn, E., Rollins, D., & Cinar, A. (2013). Hypoglycemia early alarm systems based on multivariable models. Industrial & engineering chemistry research, 52(35), 12329-12336.

[13] Cinar, A. (2018). Artificial pancreas systems: an introduction to the special issue. IEEE Control Systems Magazine, 38(1), 26-29.

[14] Bothe, M. K., Dickens, L., Reichel, K., Tellmann, A., Ellger, B., Westphal, M., & Faisal, A. A. (2013). The use of reinforcement learning algorithms to meet the challenges of an artificial pancreas. Expert review of medical devices, 10(5), 661-673.

[15] Bastani, M. (2014). Model-free intelligent diabetes management using machine learning.

[16] Cox, M. E., Feinglos, M. N., & Corsino, L. (2011). Glycemic control in the hospitalized patient. L. F. Lien (Ed.). Springer Science+ Business Media, LLC.

[17] Albisser, A. M., Leibel, B. S., Ewart, T. G., Davidovac, Z., Botz, C. K., & Zingg, W. (1974). An artificial endocrine pancreas. Diabetes, 23(5), 389-396.

[18] Farmer Jr, T. G., Edgar, T. F., & Peppas, N. A. (2008). The future of openand closedloop insulin delivery systems. Journal of Pharmacy and Pharmacology, 60(1), 1-13.

[19] Steil, G. M., Clark, B., Kanderian, S., & Rebrin, K. (2005). Modeling insulin action for development of a closed-loop artificial pancreas. Diabetes technology & therapeutics, 7(1), 94-108.

[20] Takahashi, D., Xiao, Y., & Hu, F. (2008). A survey of insulin-dependent diabetes-part II: control methods. International journal of telemedicine and applications, 2008, 4.

[21] Knebel, T., & Neumiller, J. J. (2019). Medtronic MiniMed 670G Hybrid Closed-Loop System. Clinical Diabetes, 37(1), 94-95.

[22] Daskalaki, E., Diem, P., & Mougiakakou, S. G. (2013). An ActorCritic based controller for glucose regulation in type 1 diabetes. Computer methods and programs in biomedicine, 109(2), 116-125.

[23] Fox, I., & Wiens, J. (2019). Reinforcement Learning for Blood Glucose Control: Challenges and Opportunities.

[24] Kong, J. D. (2012). Glucose control by means of an observer-based control law (Master's thesis, University of L'Aquila, L'Aquila, Italy). Retrieved from https://www.mathmods.eu/resources/downloads/master-theses/finish/37-master-s-theses/248-jude-kong-thesis

[25] Theodoridis, S. & Koutroumbas, K. (2009). Pattern Recognition. Elsevier Acad. Press, Amsterdam, 4

[26] Rusell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach Global Edition.

[27] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer Science+ Business Media.

[28] Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

[29] Alpaydin, E. (2014). Introduction to machine learning. MIT press.

[30] Mller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".

[31] Flach, P. (2012). Machine learning: the art and science of algorithms that make sense of data. Cambridge University Press.

[32] What is Machine Learning? A definition. (2017, March 17). Retrieved from https://expertsystem.com/machine-learning-definition/

[33] Castle, N. (2018, February 8). What is Semi-Supervised Learning? Retrieved from https://blogs.oracle.com/datascience/what-is-semi-supervised-learning

[34] Chapelle, O., Scholkopf, B., & Zien, A. (2009). Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. IEEE Transactions on Neural Networks, 20(3), 542-542.

[35] Gary (2018, August 2). Applications of Reinforcement Learning in Real World. Retrieved from https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

[36] Silver, D. (2015). Lecture 1: Introduction to Reinforcement Learning. Retrieved from http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf

[37] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[38] Alpaydin, E. (2014). Introduction to machine learning. MIT press.

[39] Poole, D. L., & Mackworth, A. K. (2010). Artificial Intelligence: foundations of computational agents. Cambridge University Press.

[40] Klopf, A. H. (1972). Brain function and adaptive systems: a heterostatic theory (No. AFCRL-SR-133). AIR FORCE CAMBRIDGE RESEARCH LABS HANSCOM AFB MA.

[41] Klopf, A. H. (1975). A comparison of natural and artificial intelligence. ACM SIGART Bulletin, (52), 11-13.

[42] Klopf, A. H. (1982). The hedonistic neuron: a theory of memory, learning, and intelligence. Toxicology-Sci.

[43] Bellman, R. E. (1957a). Dynamic Programming. Princeton University Press, Princeton.

[44] Bellman, R. E. (1957b). A Markov decision process. Journal of Mathematics and Mechanics, 6(5):679684.

[45] Bertsekas, D. P., & Tsitsiklis, J. N. (1996). Neuro-Dynamic Programming. Athena Scientific, Belmont, MA.

[46] Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. Ph.D. thesis, University of Cambridge.

[47] Sutton, R. S. (1984). Temporal Credit Assignment in Reinforcement Learning. Ph.D. thesis, University of Massachusetts, Amherst.

[48] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine learning, 3(1), 9-44.

[49] Part 1: Key Concepts in RL. (2018). Retrieved from https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[50] Mosching, A. (2016). Reinforcement Learning Methods for Glucose Regulation in Type 1 Diabetes (Unpublished master's thesis). Ecole Polytechnique Federale de Lausanne, Switzerland.

[51] Masson, W., Ranchod, P., & Konidaris, G. (2016, February). Reinforcement learning with parameterized actions. In Thirtieth AAAI Conference on Artificial Intelligence.

[52] CartPole-v1. (n.d.). Retrieved from http://gym.openai.com/envs/CartPole-v1/

[53] Wiering, M., & Van Otterlo, M. (2012). Reinforcement learning. Adaptation, learning, and optimization, 12, 3.

[54] Bhatt, S. (2018, March 28). 5 Things You Need to Know about Reinforcement Learning. Retrieved from https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

[55] What is a "trajectory" in reinforcement learning? (2018, July 31). Retrieved from https://ai.stackexchange.com/questions/7359/what-is-a-trajectory-in-reinforcement-learning

[56] Van Otterlo, M., & Wiering, M. (2012). Reinforcement learning and markov decision processes. In Reinforcement Learning (pp. 3-42). Springer, Berlin, Heidelberg.

[57] Markov property. (2019, October 6). Retrieved from https://en.wikipedia.org/wiki/Markov_property.

[58] Silver, D. (2015). Lecture 2: Markov Decision Processes. Retrieved from http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf.

[59] Vadali, G. (2019, March 7). Advantage function in Deep Reinforcement learning. Retrieved from https://mc.ai/advantage-function-in-deep-reinforcement-learning/.

[60] Szepesvri, C. (2010). Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning, 4(1), 1-103.

[61] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

[62] Hui, J. (2018, September 17). RL Reinforcement Learning Terms. Retrieved from https://medium.com/@jonathan_hui/rl-reinforcement-learning-terms-242baac11907.

[63] Hasselt, H. V. (2010). Double Q-learning. In Advances in neural information processing systems (pp. 2613-2621).

[64] Brunton, S. L., & Kutz, J. N. (2019). Data-driven science and engineering: Machine learning, dynamical systems, and control. Cambridge University Press.

[65] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

[66] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nat. 521, 436444.

[67] Wickstrøm, K. (2018). Uncertainty Modeling and Interpretability in Convolutional Neural Networks for Polyp Segmentation (Master's thesis, The University of Tromsø, Tromsø, Norway). Retrived from https://munin.uit.no/handle/10037/13552.

[68] Hansen, M. A. (2019). Affinity-Guided Image-to-Image Translation for Unsupervised Heterogeneous Change Detection (Unpublished master's thesis). University of Tromsø, Tromsø, Norway.

[69] Basics of Multilayer Perceptron A Simple Explanation of Multilayer Perceptron. (2018, January 7). Retrieved from https://kindsonthegenius.com/blog/2018/01/basics-of-multilayer-perceptron-a-simple-explanation-of-multilayer-perceptron.html.

[70] Gomez, R. (2018, May 23). Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. Retrieved from https://gombru.github.io/2018/05/23/cross_entropy_loss/.

[71] Ruhmelhart, D., Hinton, G., & Wiliams, R. (1986). Learning representations by back-propagation errors. Nature, 323(533-536), 10.

[72] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[73] Robbins, H., & Monro, S. (1951). A stochastic approximation method. The annals of mathematical statistics, 400-407.

[74] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[75] Sutton, R. (1986). Two problems with back propagation and other steepest descent learning procedures for networks. In Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986 (pp. 823-832).

[76] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural networks, 12(1), 145-151.

[77] Optimizers. (2017). Retrieved from https://ml-cheatsheet.readthedocs.io/en/latest/optimizers.html.

[78] Nicholson, C. (2019). A Beginner's Guide to Deep Reinforcement Learning. Retrieved from https://pathmind.com/wiki/deep-reinforcement-learning.

[79] Choudhary, A. (2019, April 18). A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python. Retrived from https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[80] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[81] van Hasselt, H., Guez, A., & Silver, D. (2015). Deep reinforcement learning with double Q-learning. CoRR abs/1509.06461 (2015). arXiv preprint arXiv:1509.06461.

[82] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.

[83] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952.

[84] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., ... & Blundell, C. (2017). Noisy networks for exploration. arXiv preprint arXiv:1706.10295.

[85] Bellemare, M. G., Dabney, W., & Munos, R. (2017, August). A distributional perspective on reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 449-458). JMLR. org.

[86] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... & Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. 2017. URL http://arxiv. org/abs/1710.02298.

[87] Sung, F. (2018, April 22). Conquering OpenAI Retro Contest 2: Demystifying Rainbow Baseline. Retrieved from https://medium.com/intelligentunit/conquering-openai-retro-contest-2-demystifying-rainbow-baseline-9d8dd258e74b.

[88] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[89] Lin, L. J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3-4), 293-321.

[90] Yang, Z., Xie, Y., & Wang, Z. (2019). A theoretical analysis of deep Q-learning. arXiv preprint arXiv:1901.00137.

[91] Salloum, Z. (2018, December 6). Double Q-Learning, the Easy Way. Retrieved from https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3.

[92] Arulkumaran, K. (2016, April 30). Dueling Deep Q-Networks. Retrieved from http://torch.ch/blog/2016/04/30/dueling_dqn.html.

[93] Neal, R. M. (2001). Annealed importance sampling. Statistics and computing, 11(2), 125-139.

[94] Oord, A. V. D., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759.

[95] Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144.

[96] Huyett, L. M., Dassau, E., Zisser, H. C., & Doyle III, F. J. (2015). Design and evaluation of a robust PID controller for a fully implantable artificial pancreas. Industrial & engineering chemistry research, 54(42), 10311-10321.

[97] Homko, C., Deluzio, A., Jimenez, C., Kolaczynski, J. W., & Boden, G. (2003). Comparison of insulin aspart and lispro: pharmacokinetic and metabolic effects. Diabetes care, 26(7), 2027-2031.

[98] Home, P. D. (2015). Plasma insulin profiles after subcutaneous injection: how close can we get to physiology in people with diabetes?. Diabetes, Obesity and Metabolism, 17(11), 1011-1020.

[99] Belicar, P. S., Vague, P., & Lassmann-Vague, V. (2003). Reproducibility of plasma insulin kinetics during intraperitoneal insulin treatment by programmable pumps. Diabetes & metabolism, 29(4), 344-348.

[100] Salloum, Z. (2019, April 24). Exploration in Reinforcement Learning. Retrieved from https://towardsdatascience.com

[101] In silico. (2020, January 5). Retrieved from https://en.wikipedia.org/wiki/In_silico

[102] Man, C. D., Micheletto, F., Lv, D., Breton, M., Kovatchev, B., & Cobelli, C. (2014). The UVA/PADOVA type 1 diabetes simulator: new features. Journal of diabetes science and technology, 8(1), 26-34.

[103] Dalla Man, C., Rizza, R. A., & Cobelli, C. (2007). Meal simulation model of the glucose-insulin system. IEEE Transactions on biomedical engineering, 54(10), 1740-1749.

[104] Kovatchev, B. P., Breton, M., Dalla Man, C., & Cobelli, C. (2009). In silico preclinical trials: a proof of concept in closed-loop control of type 1 diabetes.

[105] Visentin, R., Campos-Nez, E., Schiavon, M., Lv, D., Vettoretti, M., Breton, M., ... & Cobelli, C. (2018). The UVA/Padova type 1 diabetes simulator goes from single meal to single day. Journal of diabetes science and technology, 12(2), 273-281.

[106] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym.(2016). arXiv preprint arXiv:1606.01540.

[107] Myhre, J. N., Launonen, I. K., Wei, S., & Godtliebsen, F. (2018, September). CONTROLLING BLOOD GLUCOSE LEVELS IN PATIENTS WITH TYPE 1 DIABETES USING FITTED Q-ITERATIONS AND FUNCTIONAL FEATURES. In 2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP) (pp. 1-6). IEEE.

[108] Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. arXiv preprint arXiv:1903.02428.

[109] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., ... & Zhokhov, P. (2016). Openai baselines (2017). URL https://github. com/opfenai/baselines.

[110] Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language for technical computing. arXiv preprint arXiv:1209.5145.

[111] Time-In-Range. (2019, August 7). Retrieved from https://diatribe.org/time-range

[112] El Fathi, A., Smaoui, M. R., Gingras, V., Boulet, B., & Haidar, A. (2018). The artificial pancreas and meal control: an overview of postprandial glucose regulation in type 1 diabetes. IEEE Control Systems Magazine, 38(1), 67-85.

[113] Ryan, E. (2019, May 1). Time-in-Range Tips: Expert-Defined Goals, Plus Insights from Almost 500,000 FreeStyle Libre Users. Retrieved from https://diatribe.org/time-range-tips-expert-defined-goals-plus-insights-almost-500000-freestyle-libre-users

[114] Catastrophic interference. (2019, September 30). Retrieved from https://en.wikipedia.org/wiki/Catastrophic_interference

[115] Kemker, R., McClure, M., Abitino, A., Hayes, T. L., & Kanan, C. (2018, April). Measuring catastrophic forgetting in neural networks. In Thirty-second AAAI conference on artificial intelligence.

[116] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... & Hassabis, D. (2017). Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, 114(13), 3521-3526.

[117] Blood sugar level. (2020, January 19). Retrieved from https://en.wikipedia.org/wiki/Blood_sugar_level

[118] Dong, D., Chen, C., Li, H., & Tarn, T. J. (2008). Quantum reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 38(5), 1207-1220.

[119] Dunjko, V., Taylor, J. M., & Briegel, H. J. (2017, October). Advances in quantum reinforcement learning. In 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 282-287). IEEE.

[120] Hu, W., & Hu, J. (2019). Q Learning with Quantum Neural Networks. Natural Science, 11(01), 31.

[121] Ganger, M., & Hu, W. (2019). Quantum Multiple Q-Learning. International Journal of Intelligence Science, 9(01), 1.

[122] Hovorka, R., Canonico, V., Chassin, L. J., Haueter, U., Massi-Benedetti, M., Federici, M. O., ... & Wilinska, M. E. (2004). Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. Physiological measurement, 25(4), 905.

[123] Wilinska, M. E., Chassin, L. J., Acerini, C. L., Allen, J. M., Dunger, D. B., & Hovorka, R. (2010). Simulation environment to evaluate closed-loop insulin delivery systems in type 1 diabetes. Journal of diabetes science and technology, 4(1), 132-144.

[124] Boiroux, D., Finan, D. A., Jørgensen, J. B., Poulsen, N. K., & Madsen, H. (2010). Optimal insulin administration for people with type 1 diabetes. IFAC Proceedings Volumes, 43(5), 248-253.

[125] Nærum, M. (2010). Model predictive control for insulin administration in people with type 1 diabetes (Bachelor's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark).

[126] Elashoff, J. D., Reedy, T. J., & Meyer, J. H. (1982). Analysis of gastric emptying data. Gastroenterology, 83(6), 1306-1312.

[127] Lehmann, E. D., & Deutsch, T. (1992). A physiological model of glucose-insulin interaction in type 1 diabetes mellitus. Journal of biomedical engineering, 14(3), 235-242.

[128] Dalla Man, C., Camilleri, M., & Cobelli, C. (2006). A system model of oral glucose absorption: validation on gold standard data. IEEE Transactions on Biomedical Engineering, 53(12), 2472-2478.

[129] Goetze, O., Steingoetter, A., Menne, D., van der Voort, I. R., Kwiatek, M. A., Boesiger, P., ... & Schwizer, W. (2007). The effect of macronutrients on gastric volume responses and gastric emptying in humans: a magnetic resonance imaging study. American Journal of Physiology-Gastrointestinal and Liver Physiology, 292(1), G11-G17.

[130] Himmelblau, D. M., & Riggs, J. B. (2004). Basic principles and Calculations in Chemical Engineering, Prentice Hall, 2004: Basic principles and Calculations in Chemical Engineering (Vol. 1). Bukupedia.

[131] Wilinska, M. E., Chassin, L. J., Schaller, H. C., Schaupp, L., Pieber, T. R., & Hovorka, R. (2004). Insulin kinetics in type-1 diabetes: continuous and bolus delivery of rapid acting insulin. IEEE Transactions on Biomedical Engineering, 52(1), 3-12.

[132] Magni, L., Raimondo, D. M., Bossi, L., Dalla Man, C., De Nicolao, G., Kovatchev, B., & Cobelli, C. (2007). Model predictive control of type 1 diabetes: an in silico trial.

[133] Magni, L., Raimondo, D. M., Dalla Man, C., De Nicolao, G., Kovatchev, B., & Cobelli, C. (2009). Model predictive control of glucose concentration in type I diabetic patients: An in silico trial. Biomedical Signal Processing and Control, 4(4), 338-346.

[134] Eren-Oruklu, M., Cinar, A., Quinn, L., & Smith, D. (2009). Adaptive control strategy for regulation of blood glucose levels in patients with type 1 diabetes. Journal of process control, 19(8), 1333-1346.