Reinforcement Learning

An Insulin Bolus Advisor for Type 1 Diabetes Using Deep Reinforcement Learning

Yanis Schärer, 18-114-058
Raphael Joost, 18-109-504

26.01.2023

1 Introduction

1.1 Goal

The aim of this work is to reproduce the results of the bolus advisor learning algorithm published in [1] by using a deep deterministic policy gradient (DDPG, see 2.1 for further information) agent for blood glucose control in python. For this task, the `simglucose` simulator [3] is used.

1.2 Theory

Type 1 diabetes is an autoimmune disorder in which the body's immune system attacks and destroys the insulin-producing cells of the pancreas. This results in a lack of insulin, a hormone that regulates blood sugar levels. Without insulin, the body is unable to properly use glucose for energy, leading to high blood glucose (BG) levels and a variety of related health complications.

There are two types of insulin typically used to replace the natural insulin in people with type 1 diabetes: basal insulin and bolus insulin. Basal insulin is a long-acting insulin, also known as background insulin, that is typically delivered constantly to provide a steady background level of insulin throughout the day and night. Bolus insulin, on the other hand, is a rapid-acting insulin that is given to help control blood sugar levels after eating.

In comparison to a standard bolus calculator, the insulin bolus advisor implemented in [1] significantly improved the average percentage of time in target range (70-180 mg/dL) for adult and adolescent cohorts. Specifically, the average percentage of time in target range increased from 74.1% ± 8.4% to 80.9% ± 6.9% and 54.9% ± 12.4% to 61.6% ± 14.1% respectively, while also reducing hypoglycemia.

2 Methods

To reproduce the results of [1], the `simglucose` package [3] simulating the complex environment of glucose monitoring and insulin injection is used. `simglucose` also provides 10 different adolescent as well as 10 different adult patients. Furthermore, a deep deterministic policy gradient agent is implemented.

2.1 Deep Deterministic Policy Gradient (DDPG) Agent

The DDPG agent implemented for this task follows the algorithm found in [1]. The DDPG algorithm is explained in detail in [2] and consists of an actor and a critic modeled as a neural network.

Throughout this project, a simple basal calculator is used to simulate the basal rate of a patient. The basal rate is given by

$$Basal_t = \frac{u2ss \cdot BW}{6000} \tag{1}$$

where $u2ss$ is a steady state insulin rate per kg and BW is the body weight in kg. Both parameters are patient-specific.

Many standard bolus calculators use an empirical formula to derive the insulin dose:

$$Bolus_t^{standard} = \frac{CHO_t}{ICR} + \frac{G_t - G^T}{ISF} - IOB_t \tag{2}$$

where CHO_t is the amount of carbohydrate ingested in gram, G_t is the current BG level (ml/dL), G_T is the target BG level (set to 140 ml/dL in this project), IOB is the insulin on board and ICR as well as ISF are patient-specific values for insulin-to-carbohydrate ratio and insulin sensitivity factor, respectively. The index t is incremented after each bolus injection. The patient-specific values are given by the `simglucose` package. For general training of the agent, the values of one cohort are averaged to create an imaginary average patient. IOB_t can be estimated by

$$IOB_t = Bolus_{t-1} \cdot \max\left(0, \left(1 - \frac{ts_t - ts_{t-1}}{T_{IOB}}\right)\right) \tag{3}$$

where ts_t is the time of delivery of $Bolus_t$ and T_{IOB} denotes the active time of insulin (set to 5 hours in our case).

Given the state

$$s_t = \{CGM_{hist},\ CHO_{hist},\ Insulin_{hist}\} \tag{4}$$

where CGM_{hist}, CHO_{hist} and $Insulin_{hist}$ are a list of the last n ($n = 6$ in our case) respective values for CGM (BG measurements), CHO (meal ingestion) and $Insulin = Basal + Bolus$ (insulin dose).

The actor of the DDPG agent outputs a 3-dimensional vector $\mu(s_t|\theta^\mu)$ based on the state s_t given the actor weights θ^μ. The bolus action $Bolus_t$ then is

$$Bolus_t = \left[\frac{CHO_t}{ICR},\ \frac{G_t - G^T}{ISF},\ -IOB_t\right] \cdot \mu(s_t|\theta^\mu)^T. \tag{5}$$

Each entry of the action $\mu(s_t|\theta^\mu)$ is in the range [0.6, 1.4]. It therefore acts as gain to the standard bolus calculator in Eq. (2). The action of the DDPG bolus calculator (Eq. (5)) is the same as the action of the standard bolus calculator if $\mu(s_t|\theta^\mu) = [1, 1, 1]$. During general training, a noise drawn from $\mathcal{N}(0, 0.1)$ is added to $\mu(s_t|\theta^\mu)$. In [1] they use a different range and noise. The reasons for these changes are discussed in section 4.

The goal of the bolus calculator is to maximize postprandial BG levels in the target zone (70-180 ml/dL) while minimizing hypoglycemia occurrences. The reward function is chosen according to [1] as

$$r_t = \frac{1}{ts^* - ts_t} \sum_{k=ts_t}^{ts^*} f_R\left(G_k\right) \tag{6}$$

where $ts^* = \min\left(ts_{t+5h},\ ts_{t+1}\right)$ and

$$f_R\left(G_k\right) = \begin{cases} 0.5, & 70 \leq G_k \leq 180 \\ -0.8, & 180 < G_k \leq 300 \\ -1, & 300 < G_k \leq 350 \\ -1.5, & 30 \leq G_k < 70 \\ -200, & \text{else} \end{cases} \tag{7}$$

The factor preceding the sum in Eq. (6) serves a normalizing function to account for variations in episode duration. Unlike [1], a reward of -200 was assigned for blood glucose (BG) values outside of the range [30, 350]. In our implementation, the agent terminates when the BG value falls outside of this range, and a one-time reward of -200 is assigned. In the implementation of [1], the agent does not terminate and continues to sample rewards of -2 as long as it remains outside of the specified range. This modification was made to accelerate the training time.

2.2 Simglucose Environment

We use a version of the `simglucose` Python package [3], which was modified to suit the needs of this work. The `simglucose` package is based on Gymnasium, a standard API for reinforcement learning in Python. The most important functions of the specific modified `simglucose` are described below.

`reset`
The function named `reset` serves to reinitialize the current environment to its starting point. The starting point is defined as a randomly selected time on January 1st, 2018, and meal times and carbohydrate (CHO) amounts are generated based on the distributions outlined in previous study [1]. The meal times are established as 7:00, 14:00, and 19:00, each with a standard deviation of 30 minutes, while the CHO intake is set to 70g, 110g, and 90g (each with a standard deviation of 10%), respectively. The duration of the meal intake is fixed at 15 minutes. As a result, the CHO amount is distributed uniformly among all time steps within the 15-minute interval following the meal time. The `reset` function is invoked prior to the start of each episode.

`step`
The `step` function takes an action as input and returns a new state, a reward and some additional information. In the case of this work, the action input is the vector $\mu(s_t|\theta^\mu)$ given by the actor. Afterwards, $Bolus_t$ is computed through Eq. (5) where CHO_t is multiplied by a randomly chosen scalar between 0.7 and 1.1 to take the patient's under- or overestimation of the ingested amount of carbohydrate into account. Moreover, the new state in the form of (4) as well as the reward is returned. An additional output shows if an episode is finished. This happens when the patient's BG level is taking too extreme values (under 30 mg/dL or over 350 mg/dL) for the simulator to properly work.

2.3 Resources

The modified `simglucose` package can be found on GitHub: `https://github.com/Raphiraph7/simglucose` on branch `bolus_gym`.

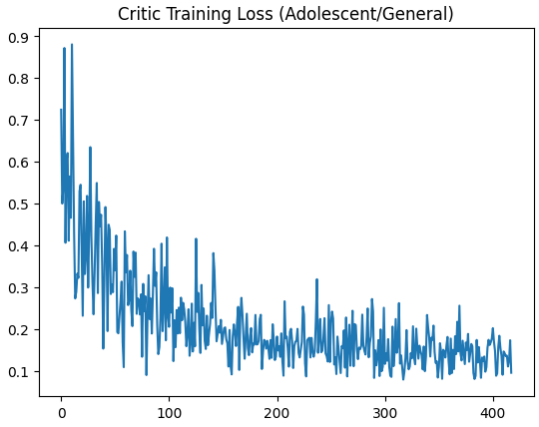
Our implementation of the deep deterministic policy gradient (DDPG) agent can be found in the file `Agent.py`, while the reward function and other utility functions are implemented in the file `utils.py`. The training process is conducted on UBELIX through various files located in the directory `training`. The training loss and the state of the agent are saved to their respective subfolders within the directory `agent_states`. To evaluate the performance of the trained agent, a Jupyter notebook named `results.ipynb` was developed, which can be utilized to replicate the evaluation of our results. To obtain the values presented in Section 3, the file located in the directory `evaluation` was run on UBELIX.

3 Results

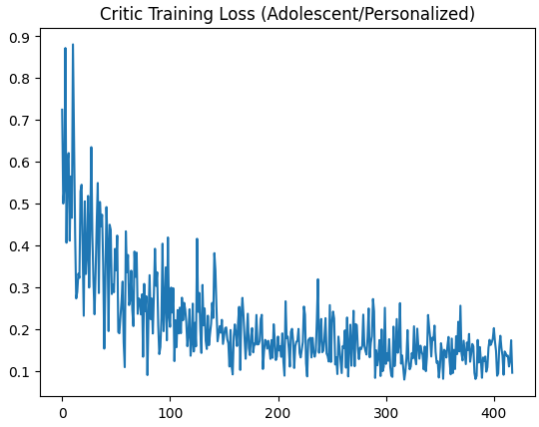
In Table 1, we present the results for the Time in Target Range (TIR) and Time Below Range (TBR). The TIR and TBR were evaluated for a sample of 50 patients, with 5 measurements taken for each patient over a period of four days, in the case of general training. The mean and standard deviation were calculated from these measurements. For personalized training, the TIR and TBR were also evaluated for a sample of 50 patients, with 50 measurements taken for the patient on which the agent was trained on.

Patient	**TIR**	**TBR**
Adolescent#0{00-10} after general training	$67.4\% \pm 18.0\%$	$0.1\% \pm 0.3\%$
Adolescent#001 after personalized training	$98.0\% \pm 1.2\%$	$0.1\% \pm 0.3\%$
Adult#0{00-10} after general training	$79.0\% \pm 11.3\%$	$1.1\% \pm 2.0\%$
Adult#001 after personalized training	$57.0\% \pm 3.3\%$	$0.2\% \pm 0.5\%$

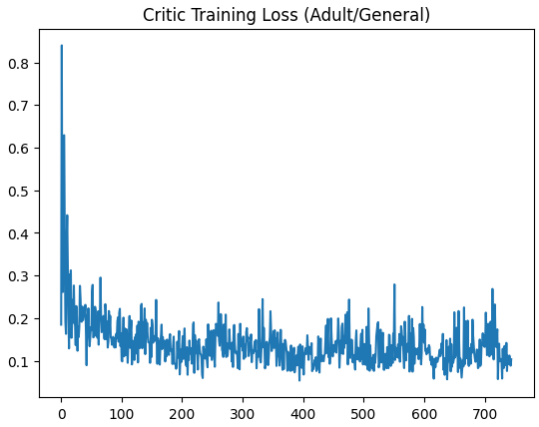
Table 1: time in target range (TIR) and time below range (TBR) values.



(a) General loss of average adolescent patient.



(b) Personalized loss of Adolescent#001.

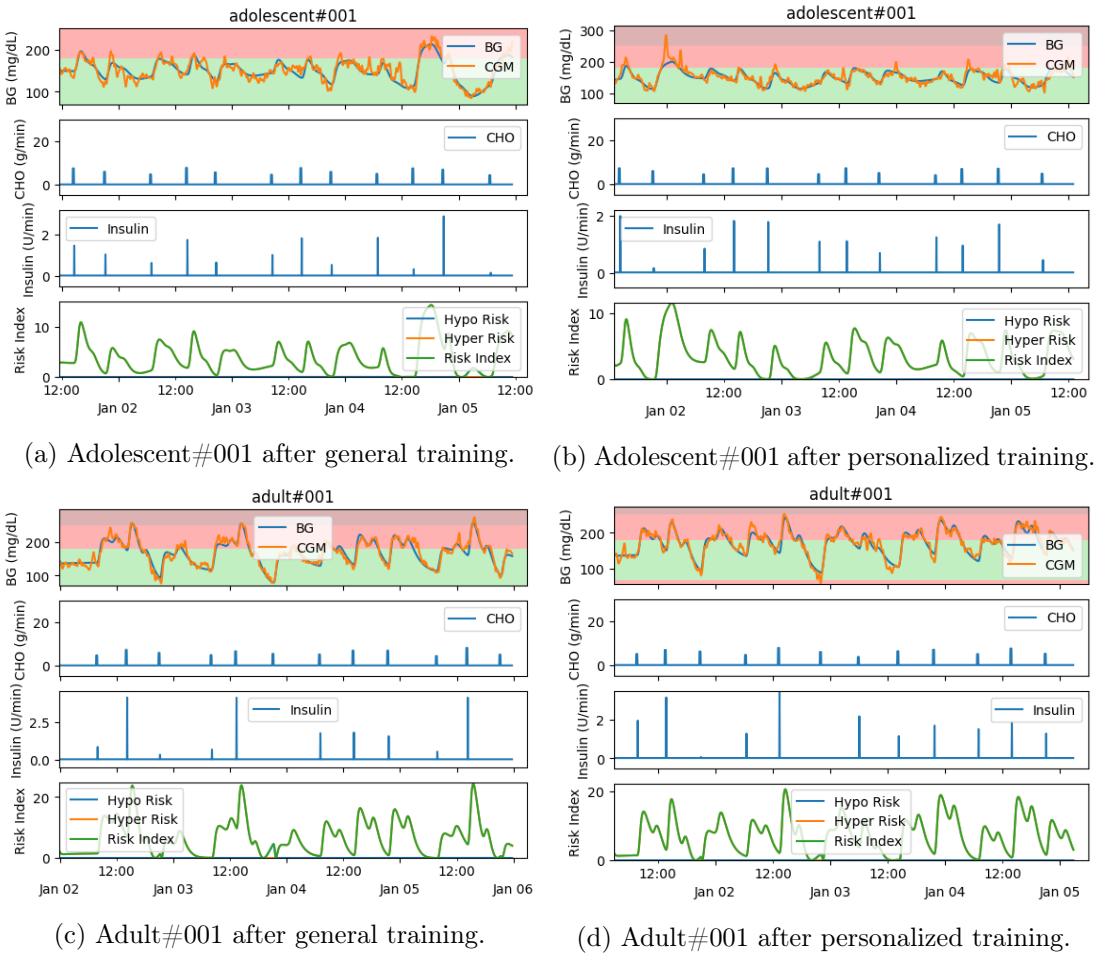


(c) General loss of average adult patient.

(d) Personalized loss of Adolescent#001.

Figure 1: Critic training losses.

The TIR value for Adult#001 after personalized training is notably low. A comparison of the training loss plots in Figure 1 reveals that the personalized training of the adult patient was unstable and the critic failed to converge within an acceptable time frame. The losses of the other training instances appear as expected. Figure 2 presents visual examples of four-day samples after each training instance.



(a) Adolescent#001 after general training.

(b) Adolescent#001 after personalized training.

(c) Adult#001 after general training.

(d) Adult#001 after personalized training.

Figure 2: Samples of blood glucose levels during four days after general and personalized training.

4 Discussion and Conclusion

Our agent demonstrated superior performance compared to the results reported in reference [1]. Specifically, in the case of general training on adults, the TIR value was slightly lower (79.0% vs 80.9%) while in the case of adolescents, the TIR value was higher (67.4% vs 61.6%). Additionally, both TBR values were significantly lower for our agent (1.1% vs 1.9% for adults and 0.1% vs 4.3% for adolescents). However, the results of the personalized agent did not align with expectations. The results for the adult patient appear to be reasonable, with a TIR value of 57.0% and a TBR value of 0.2%. Although the TIR value was expected to be higher. The TIR value for the adolescent patient at 98.0% is notably higher than anticipated and requires further investigation.

In the initial stages of this research, the same action space as in previous study [1] was adopted. The actions in that study were in the range of [0.2, 2] with a noise of $\mathcal{N}(0,\ 0.3)$ during general training. The action space used in the previous study was relatively large, making it difficult for the agent to learn optimal actions. When the agent was trained using the same range and noise, it exhibited variable performance, with some training instances resulting in good

actions and high TIR values, while others did not. The inconsistent performance was attributed to the randomness of the noise, which resulted in the agent never training in the same way and sometimes not learning optimal actions. To achieve consistent and adequate results with a larger action space, a significantly greater number of training iterations were required. To obtain reproducible results with a lower number of iterations, the action space was subsequently limited. Additionally, there were no scenarios where actions at the extremum (0.2 or 2) would be appropriate. Nevertheless, even with the adjusted range of [0.6, 1.4], the training process is not completely robust and sometimes still fails to produce good results (see Fig. 1d).

During the general training process, a fictional average patient was utilized, whose parameters were calculated by taking the mean of the parameters of the 10 available patients. It should be noted that the parameters exhibit non-linear behavior, and thus, simply taking the mean may not be an adequate method. This may have had an impact on the obtained results.

4.1 Outlook

In this study, the time interval for insulin-on-board (T_{IOB}) was set to five hours, which implies that the insulin-on-board (IOB) value is nullified if the subject does not consume food within five hours. The time periods between 7:00 and 12:00, and between 19:00, encompass seven hours. With a standard deviation of 30 minutes, it is highly improbable that the subject ingests two meals within five hours. Therefore, in the majority of cases during the training phase, the IOB value is zero, indicating that the third element in the vector in equation (5) is zero. As the dot product with the action is taken, the agent learns that the third element in the action vector has no significance and the output value is random. In a real-world scenario, it is likely that the subject consumes food multiple times within five hours. To achieve optimal results in these cases, the agent's training should incorporate a greater degree of variability in food intake.

Due to time constraints, several aspects of the agent were not optimized, such as the dimensions of the hidden layers of the actor and the critic. Throughout this project, the same architectures as in previous study [1] were utilized, which consist of multi-layer perceptrons with two hidden layers of size 200 and 10. Another feature that was not implemented is priority sampling, which is anticipated to significantly accelerate the training process.

In order to achieve the desired functionality, modifications to the code of the `simglucose` package were necessary at certain points. This approach had some negative consequences, such as faulty `info` returned from both the `reset` and "step" functions, which did not display accurate values for the meal intake. An alternative source was utilized to obtain the CHO_t value. Given additional time, efforts would be made to devise solutions for compatibility issues within our own code, rather than making changes to the provided `simglucose` code.

References

[1] Zhu T., Li K., Herrero P. & Georgiou P. (2020). "An Insulin Bolus Advisor for Type 1 Diabetes Using Deep Reinforcement Learning". *Sensors (Basel, Switzerland)* (18), pp. 1-15

[2] Lillicrap T. P. et al. (2015). "Continuous control with deep reinforcement learning". *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*

[3] Xie J. (2018). "Simglucose v0.2.1 (2018) [Online]". `https://github.com/jxx123/simglucose` (Accessed: 08/01/2023)