REINFORCEMENT LEARNING

Sara Silvestrelli
Dipartimento di Economia, Metodi Quantitativi e Strategie di Impresa
Università degli studi di Milano-Bicocca

ABSTRACT

Reinforcement Learning (RL) techniques work well in this field designing controllers for non-linear systems by training them according to a trial-error procedure: an agent is trained to provide decision recommending treatments. RL is a valid tool for sequential decision-making problems and suitable to describe medical environments. Q-learning is one of the main approaches for this purpose.

***K*eywords** Artificial Intelligence · Reinforcement Learning · Q-Learning

1 Introduction

ML is a branch of AI and computer science young enough to still be subject to continuous and rapid additions and modifications often driven by problems in practical applications. One major trend driving this expansion is a growing concern with the environment in which a machine-learning algorithm operates [1], which partially refers to to the computing architecture and also to the source of the data.

Depending on the nature of the signal available, there are three main categories in which ML algorithms can be divided into:

- **Supervised Learning**: a task-driven approach in which we define a model in order to predict a desired output.
- **Unsupervised Learning**: a data-driven approach to find structure in the data, like clustering of data points.
- **Reinforcement Learning (RL)**: an algorithm that can learn from its mistakes.

Reinforcement Learning [2] is the third ML paradigm chosen for this work as it is able to handle multi-stage problems and sequential decision making. It doesn't need a dataset to learn from, but rather an environment to interact with and from which it is received a feedback; RL is a computational approach that learns the optimal policy through interaction with the environment without the model of the environment; the training data in RL only provides an indication whether an action is correct or not. The aim of RL is to build an agent who automatically learns to take the optimal action, which is not known in advance, in order to maximize the expected cumulative return through a trial-and-error search over time using rewards and punishments as signals. This makes RL characteristics very similar to those of the human learning process. The idea behind RL can be enclosed in the *Reward Hypothesis*:

> All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The main difference from other algorithms is that a RL agent learns from the consequences of its own actions and experiences. In this scenario, time is a relevant component since the sequential nature of the data and the rewards. The quantities to be maximised are not necessarily instantaneous but delayed.

A number of difficulties may arise when working beyond benchmark environments, for example it can be complicated to understand the policy to be used, understand how to specify the reward function or define a representation of the state appropriate to the proposed study.

First of all some key terms can be summarised as follows:

- **Agent**: a program that makes intelligent decisions by interacting with the environment and taking actions. The action performed by the agent returns a cumulative return (CR) which is the quantity to maximize.

- **Environment**: the world with which the agent interacts. We can use both a real or a simulated environment.
- **History**: includes the information known up to that time on which the agent relies before acting.
- **State**: the information $s_t \in S$ used to determine what happens next. It can be anything useful to choose an action. Formally the state is a function of history and a sufficient statistic of the environment[1].
- **Action**: any decision $a_t \in A$ the agent can take.
- **Reward**: a numerical value, positive or negative, that indicates how well the agent is behaving. The immediate reward is expressed as r_{t+1}.
- **Episode**: a sequence of state-action-rewards where the number of step at each episode can be infinite or finite $(S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...)$.

In Figure 1 it is possible to observe more intuitively all these elements combined together. At the beginning, the RL agent interacts with the environment and, considering the information in its possession or in a completely random way as will be specified later, he takes an action *a*. The environment *E*, in response to that, returns a new state value s_{t+1} and a reward r_{t+1} and it starts all over again. It is an action-reward feedback loop.

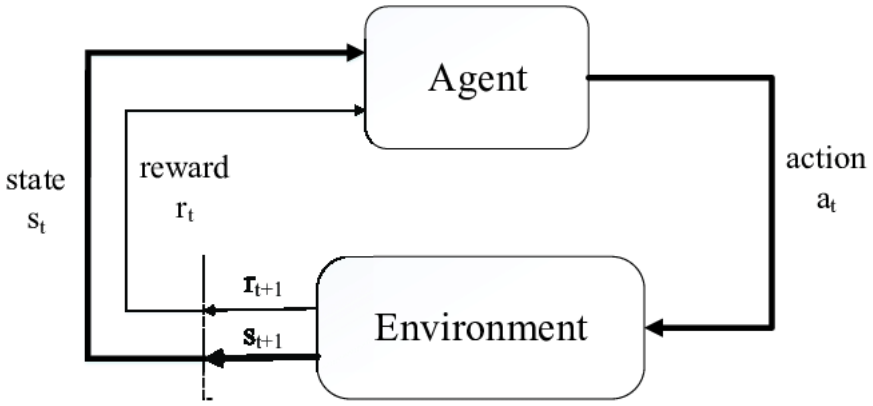

Figure 1: The agent–environment interaction in a Markov decision process [3].

2 Trade-off between Exploration and Exploitation

We want the agent to find the best solution as quickly as possible. However, choosing the right action to take too quickly can be counterproductive since usually at the beginning the environment is unknown and must be explored. This leads to the so called exploration-exploitation trade off: the exploration phase looks for more informations of the environment to make better decisions in the future, occasionally taking a random action instead of the supposed optimal one. The exploitation phase maximizes the reward with the known information.

Having the agent explore the environment too much can result in missed opportunities because even when an action appears to be the optimal one, wasting both time and compute resources, the agent continues to explore and may miss actions that lead to a better reward. In contrast, not exploring it enough can lead to not identifying new patterns or appropriate strategies. But exploitation is also risky. Exploiting only the information available can lead to the agent being stuck in a local optimum. Not exploiting it enough can have the opposite effect of not acting wisely when necessary.

3 Components of an RL Agent

RL's components are graphically explainable as shown in the Figure 1. Focusing on the RL agent, there are three major elements to be considered: the model, the policy, the value function.

[1]This means that all necessary information for the agent to perform the best actions is comprised in the state.

3.1 Model

The model allows us to make predictions about how the environment will change after an agent takes an action. It is something that emulates the behavior of the environment. The RL approaches that use models are called *model-based* methods, otherwise if the method concerns simple trial-and-error search they are called *model-free* methods.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \tag{1}$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] \tag{2}$$

The model 1 is the *Tansition Model* $\mathcal{P}$ and it predicts the next state, the model 2 is the *Reward Model* $\mathcal{R}$ and it predicts the immediate reward.

3.1.1 Policy

The policy π can be a treatment regime or a set of decision-making rules that determines how the agent selects its actions at a certain moment. It maps states to actions that promise the highest reward. In some scenarios the policy can be a simple function but more difficult tasks may require a more complex predictor such as a neural network.

The policy can be deterministic or stochastic: it is *deterministic* when each state corresponds to a single action ($\pi(s) = a$); it is *stochastic* when a state corresponds to a conditional probability distributions over actions given the state $S_t = s$ and we have to find the best one ($\pi(a|s) = \mathbb{P}[A_t = a|St = s]$).

3.1.2 Value function

The criterion that determines which policy is best is the value function. It is the prediction of future reward used to evaluate the goodness of states and/or actions defining what is best in the long run. The agent wants to maximize the *discounted expected return function* G_t described as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3}$$

where G_t is the expected discounted sum of future rewards under a particular policy π. The value of reward decreases over time introducing the concept of depreciation. Each reward is weighted by a value $\gamma \in [0, 1]$ called *discount factor* which is higher the more the reward it is associated with is recent. On the opposite, the lower the value of γ the faster future rewards are valued. When $\gamma = 0$ we are only interested in immediate reward which overshadows future rewards ("myopic" evaluation), otherwise when $\gamma = 1$ future or immediate reward are equally prioritized ("far-sighted" evaluation). This function allows us to define the importance of evaluating a reward delayed in time.

In order to calculate the reward, the RL agent must be able to assess the goodness of the chosen action. This is the task of the value function that can be split into two type: the *state* and *state-action* value functions. More specifically:

- State-value function (**V-function**)
  The letter "V" stands for the expected overall value of a state. It tells us how good a state $s \in S$ is considering the expected reward from that state s following policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \quad \forall s \in S \tag{4}$$

- State-action value function (**Q-function**[2])
  The letter "Q" stands for the "quality of the action". It tells us how much it pays the agent to take the action *a* starting from a state *s* following policy π. It is also used in the Q-Learning for the Q-value function where the optimal action maximizes this value:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \quad \forall s \in S, a \in A. \tag{5}$$

4 Markov Process

The agent at each time step receives some representation of the environment's state. A reinforcement learning agent can therefore be modelled as a Markov decision process. A Markov process (or Markov Chain), named after Russian mathematician A.A. Markov, is a "memoryless" stochastic process, i.e. a sequence of random states that satisfy the below *Markov Property*:

[2] Also called Q-value function.

the future is independent of the past given the present. A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1}|S_1, ..., S_t] = \mathbb{P}[S_{t+1}|S_t] \tag{6}$$

It is said to be "memoryless" because the state of the system is only influenced by what happens in the immediately preceding state. All states before the previous one can be thrown away. This means that the relevant information is enclosed in the recent state and the knowledge acquired further in the past may be excluded.

A MP can be defined as a tuple of states-probabilities $(\mathcal{S}, \mathcal{P})$ where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix s.t. $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$

4.1 Markov Decision Process

Let's consider the actions previously obtaining a MDP through which, mathematically speaking, it is possible to formulate a RL problem. Beyond that are considered reward functions (3) at each stage to evaluate the transition between different states. The sequential behavior decision problem of RL is defined by the MDP which has a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix s.t.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \tag{7}$$

- $\mathcal{R}$ is a reward function s.t.

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \tag{8}$$

- $\gamma \in [0, 1]$ is a discount factor

4.2 Bellman Equations

The goal of the agent is to maximise its expected future reward and this can be done by solving the Bellman optimality equations.

Intuitively speaking, Q-function and the value function are expressed as Bellman equations that, once written recursively in terms of other successive states, allow us to iteratively find the value function for a data policy.

An *Optimal Policy* π^* is a policy that always chooses the action that maximizes the expected discounted return (the utility) of the current state as follow:

$$V^*(s) \geq V^\pi(s), \quad \forall s \in S$$
$$Q^*(s, a) \geq Q^\pi(s, a), \quad \forall s \in S,\ a \in A.$$

Thus the *optimal value function* and *optimal action value function* can be formally explained as below:

$$V^*(s) = \max_\pi V^\pi(s), \quad \forall s \in S \tag{9}$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a), \quad \forall s \in S,\ a \in A. \tag{10}$$

Equation 9 and Equation 10 can be expanded in order to define what is known as the Bellman Optimality Equation in the form:

$$\begin{aligned} V^*(s) &= \max_\pi V^\pi(s) \\ &= \max_a Q^*(s, a) \\ &= \max_a \mathbb{E}_{\pi^*}[G_t|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi^*}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \\ &= \max_a \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'} V^*(s') \end{aligned} \tag{11}$$

$$
\begin{aligned}
Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a) \\
&= \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \ \max_{a'} Q^*(s',a')
\end{aligned}
\tag{12}
$$

5 MDP Algorithms

There are three main basic algorithms of RL to solve a MDP problem and compute optimal policies:

- **Dynamic Programming (DP)**: it can solve complex problems separating them into smaller sub-problems by later combining solutions.
- **Monte Carlo (MD)**: it is a method that learns directly from complete episodes of experience and does not require a complete understanding of the model.
- **Temporal Difference (TD)**: it is a model-free method that can learn directly from either complete or incomplete episodes of experience.

5.1 Dynamic Programming

In Dynamic Programming (DP) the key idea is to use value functions through Bellman equations to search good policies. The *dynamic* term refers to the sequential and temporal problem while the *programming* term refers to the optimization of a program, which is the policy. It is a useful strategy when the problem has an optimal substructure. DP can estimate the value by using only current estimates and involves *policy iteration* and *value iteration*: the first approach has a first step of iterative policy evaluation and a second step of greedy policy improvement; the second approach starts from the final reward and proceed backwards to find the optimal value functions. In this last case there is not an explicit policy to be considered.

5.2 Monte Carlo

MC is a method that uses the simple idea of *empirical mean return* instead of expected return. It does not need a complete knowledge of the environment but only requires complete episodes of experience.

The first *policy evaluation* step builds an approximation of the value based on the current policy. The second *improvement* step makes the policy better with respect to the current value function. This procedure is then iterated repeatedly converging to the optimal policy and value function but MC requires a large number of iterations to guarantee this result.

In conclusion, MC simply averages the returns observed after visiting that state and the average will converge to the expected value. It is a method of measuring value only through the experience without a model.

5.3 Temporal Difference

TD is a method codified by Sutton and his colleagues which combines the two approaches briefly mentioned above. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap). Like MC, TD methods can learn directly from raw experience without a model of the environment's dynamics and use experience to solve the prediction problem: both methods update the value function in order to find the optimal one under a policy π.

As opposed to MC methods, that need to get to the last state (only then G_t is known), TD proceeds step by step needing to know only the next time-step t+1. Reached the consecutive step it is possible to obtain an estimate of the value $V(S)$.

6 Temporal Difference Learning

As briefly mentioned before, TD is a model-free method that combines the ideas behind the MC and DP approaches. The general form of the update function is:

$$
NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate] \tag{13}
$$

The expression in square brackets indicates the error in estimation and the step-size parameter (StepSize) changes from time step to time step. The simplest TD method's update structure is:

$$
V(S_{t+1}) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{14}
$$

where $R_{t+1} + \gamma V(S_{t+1})$ is the TD-target which is a biased estimate of $V^\pi(S_t)$ (the MC target would have been G_t). The constant α is the step-size parameter. The difference in brackets is the TD error δ_t, it measures the distance between the estimated value of S_t and the better estimate given by the TD target:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{15}$$

This TD method is called TD(0), or *one-step* TD. It is a bootstrapping method and its complete algorithm in procedural form is explained below in the Algorithm 1.

Algorithm 1: One-Step Temporal-Difference for estimating V^π

Inputs: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in [0, 1]$
Initialization: $V(s)$, for all $s \in S^+$, arbitrarily except that $V(final) = 0$
for each episode **do**:
    Initialize S: $S \leftarrow state$
    for each step of episode **do**:
        $A \leftarrow$ new action following π
        $R, S' \leftarrow$ observed reward and next state
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    end for
end for
return $V^\pi \approx V^*$

The purpose remains to estimate the value function under policy π. TD principle advantage is that it can learn before or without knowing the final outcome. In fact, it does not need to reach the end of the episode but can learn after every step from incomplete episodes and working in non-terminating environments. TD has low variance and some bias while MC has high variance and zero bias.

The best-understood approach to delayed reinforcement learning is temporal difference learning. Two forms of TD learning have been studied in detail, the adaptive heuristic critic of Sutton (1988) and Q-learning, introduced by Watkins (1989). Several authors have compared these methods empirically and found Q-learning superior [4]. Although the focus for analysis will be on QL, the SARSA approach will also be described below. The first one compares the current state and the best possible next state, while the second one compares the current state with the actual next state. Unlike MC which needs to work with complete episode and so wait until the end of it to update the state-action value function $Q(s, a)$, SARSA and Q-learning make the update after each step. In both cases the policy followed by the agent is ϵ-greedy.

Depending on the type of study and the information available, it may be more or less appropriate to use one approach rather than the other. The main difference lies in the type of exploration strategy used that can be divided in *on-policy* or *off-policy*:

- On-policy: algorithms evaluate and improve the same policy which is then used to select a proper action. The agent use the same policy both to learn and to act.
- Off-policy: algorithms evaluate and improve a policy that is different from the one used to select the action. The agent uses two slightly different versions of the policy to learn and to act.

6.1 Q-Learning

Q-Learning [5] is a commonly used model-free off-policy TD learning algorithm with a trial-error approach. It operates in a world defined as a controlled Markov decision process with the agent as a controller. Letter "Q" in QL stands for "quality" and refers to the function computed by the algorithm that represents the discounted future reward when we perform action a in state s.

QL is off-policy since it separates the acting policy from the learning policy and updates the selection of action using the Bellman optimal equations applying the ϵ-greed policy. In addition to all the of RL characteristics, Q-Learning models has two more notes:

- The number of possible states is finite: the agent will be present in a fixed number of possible situations.

– The number of possible actions to be taken is finite: the agent may choose from a finite number of possible actions.

State spaces and action spaces must be modest in size other then discrete or the Q-table will become too large to fit into memory using a Q-table, which will be described later, to represent our Q-function.

The learned action-value function directly approximates the optimal action-value function Q^* irrelevant of the policy followed. What is required for correct convergence is that all pairs continue to be updated. QL is mostly used because it is simple and demonstrated an exceptional learning ability in single-agent environments. It directly estimates the Q-functions at each stage without the need to use state-transition probabilities (model-free). It is very useful since transition dynamics are generally unknown.

Recalling the definition 5, Q-value indicates the quality of a particular action a_t starting from a given state s evaluating the current estimates of the sum of future rewards. In other words it expresses how many additional rewards we can accumulate through the various remaining steps if the agent is at the state s and performs the action a. Q-value therefore increases as the highest reward approaches. Each state-action pair needs to be stored in memory, usually a table called Q-table. This approach leads to an enormous computational cost both in terms of space and time when dealing with real life tasks.

6.1.1 Q-table policy

In Q-Learning the Q-Values are inserted in a look-up Q-table that has a row for every possible state and a column for every possible action. An optimal Q-table contains values that allow the agent to choose the best possible action in each state and to follow the optimal path that results in the highest reward. The Q-table therefore represents the policy with which the agent moves in the environment.

Q-Learning method is based on the following upgrade equation:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)] \tag{16}$$

where the Q-function $Q(s_t, a_t)$ based on state s and action a at time t is updated adding some quantity to the Q-value $Q(s, a)$ of the action for the current state. Before the learning phase begins, Q is initialized to a possibly arbitrary fixed value which usually set at zero. The main elements of function 16 are:

- r_{t+1}: the immediate reward when transitioning from time t to the next turn
- $\gamma \in [0, 1]$: the discount rate to evaluate how much future rewards are worth. In the context of the study considered for the analyses, it may also be interpreted as the probability to survive at every step.
- $\max_a Q(s', a)$: the optimal Q-Value estimated to return the largest total future reward for the state we'll end up in. It is based on all possible actions from next state s'.
- $\alpha \in [0, 1]$: the learning rate (or step size) which controls the convergence. It must be a small value indicating how much we update the utility-value every time we choose an action. It is usually preferable to arrive gradually to the solution setting α less than one. If it is best to exclusively exploit prior knowledge, to set α at zero makes the agent learn nothing.

It emerges that TD provides a method for calculating how much Q-value, given the a action taken in the previous state, should be changed based on what the agent has learned about the current Q-value.

The utility represents what we expect to be the value of Q(s,a). If it happens that the TD error is zero, that would result in no change in our estimates. If this keeps happening, our policy converges and we do not need to update anything.

What QL does is to start from an arbitrary estimate of Q (usually assumed given or equal to 0) and iteratively improve its estimates by doing arbitrary actions. It observes rewards in the next state and updates the Q-values with the formula 16.

QL's general procedure is shown step by step in Algorithm 2:

Algorithm 2: Q-learning (off-policy TD control) for estimating $\pi \approx \pi^*$

Algorithm parameter: step size $\alpha \in [0, 1]$, small $\epsilon > 0$
Initialization: $Q(s, a)$, for all $s \in S^+$, $a \in A$ arbitrarily except that $Q(terminal, \cdot) = 0$
for each episode **do**:
    Initialize S: $S \leftarrow state$
    for each step of episode **do**:
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a)) - Q(S, A)]$
        $S \leftarrow S'$
    end for
end for
return $Q \approx Q^*$, $\pi \approx \pi^*$

7 RL in healthcare

Driven by the increasing availability of patient data, electronic health records (EHR) and the development in computational models and algorithms, the role of AI techniques in healthcare has grown in the past decade ([6] [7]) with a particular focus on RL approaches. RL is suited to systems with delayed feedbacks in which decisions are made at each step without knowing their actual effectiveness but evaluated by a long-term future reward.

Healthcare is a dynamic process that needs to continually observe the health state of a patient in order to adopt corresponding interventions in real time. Although ML methods have shown good results in assisting patients (e.g. using neural networks to advise on insulin regime and dose adjustment for type 1 diabetes patients [8]), the idea of RL methods allows action to be taken in response to changes in the environment.

Healthcare application domains can be grouped into three main types [9]: DTRs in chronic disease or critical care, automated medical diagnosis and other general domains (e.g. health resources allocation and scheduling and health management). For this work, the domain treated is of the first type and describes a generic clinical trial scenario in patients with cancer.

Precision medicine requires an individualised treatment regime and DTRs develop optimal treatments within groups of patients improving long-term benefits. Patients can drop out of the clinical trial and consequently treatment can be discontinued at any time for a variety of reasons causing a *data censoring problem*. The final outcome of the treatment thus remains undetected complicating the discovery of an optimal regimen. Finding valid methodologies for managing such flexible settings is currently a premier challenge together with the formulation of states, actions, rewards and evaluation of policies.

References

[1] Michael Jordan and T.M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science (New York, N.Y.)*, 349:255–60, 07 2015.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[3] Richard Sutton and AG Barto. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11:126–134, 01 1999.

[4] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *IJCAI*, 1991.

[5] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 2004.

[6] Jianxing He, Sally Baxter, Jie Xu, Jiming Xu, Xingtao Zhou, and Kang Zhang. The practical implementation of artificial intelligence technologies in medicine. *Nat Med*, 25:30–36, 12/2019 2019.

[7] Steven E Dilsizian and Eliot L. Siegel. Artificial intelligence in medicine and cardiac imaging: Harnessing big data and advanced computing to provide personalized medical diagnosis and treatment. *Current Cardiology Reports*, 16:1–8, 2013.

[8] Stavroula G. Mougiakakou and Konstantina S. Nikita. A neural network approach for insulin regime and dose adjustment in type 1 diabetes. *Diabetes technology & therapeutics*, 2 3:381–9, 2000.

[9] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Comput. Surv.*, 55(1), nov 2021.