The Pennsylvania State University
The Graduate School

EXPLORE THE POTENTIAL OF REINFORCEMENT LEARNING: DOSING CONTROL AND SKETCH GENERATION

A Thesis in
Industrial Engineering
by
Haizhou Wang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2020

The thesis of Haizhou Wang was reviewed and approved by the following:

Hui Yang
Associate Professor of Industrial and Manufacturing Engineering
Thesis Advisor

Hui Zhao
Associate Professor of Supply Chain Management

Peng Liu
Professor of Information Sciences and Technology

Ling Rothrock
Professor of Industrial and Manufacturing Engineering
Interim Head of the Department of Industrial and Manufacturing Engineering

Abstract

Reinforcement learning is a machine learning method that can learn directly from the environments. Unlike supervised and unsupervised learning, reinforcement learning does not need an explicit dataset so that it is widely used in control problems, where the amount of the data is limited. With the thrive of neural networks, reinforcement learning shows more and more potentials, as the neural network enables complex non-linear value function for reinforcement learning algorithms. On the other hand, the training methodologies of reinforcement learning also aid the power of neural networks, because the loss functions for many tasks can be designed by using updating methods in reinforcement learning. Most reinforcement learning researches focus more on playing video games and robotic controls, but the potential of reinforcement learning can be extended to many other fields. There are two major challenges when applying reinforcement learning algorithms to many fields: 1) limited environment simulation resources and 2) difficulty to design reward function. This research presented two uncommon applications of reinforcement learning with the challenges mentioned above: unfractionated heparin (UFH) dosing control and stroke-based sketch generation. The goal of UFH dosing control is to aid the existing UFH dosing protocol. The existing protocol is unable to provide suggestions frequently because the criterion, activated partial thromboplastin time, is measured infrequently. A trained reinforcement learning agent can use other frequently logged criteria so that the suggestion could be updated often. The goal of stroke-based sketch generation is to use standard pixel-based sketches as training data to produce stroke-by-stroke sketches. In the UFH dosing control problem, the environment, which is a human patient, cannot be explored fully due to ethical issues; whereas in the sketch generation problem, it is hard to assign the reward, because evaluating the quality of a sketch is challenging. These two applications indicate the potential of reinforcement learning could be extended to many fields where the training process could be challenging.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First, I wish to show my gratitude to my thesis advisor, Dr. Hui Yang, who had provided insightful advice on the dosing control project. The research of applying reinforcement learning on dosing control would not be completed without his expertise and experience in the domain of healthcare.

I also wish to show my sincere appreciation to my formal advisor, Dr. Conrad Tucker, who had supervised my first-year graduate study. Without his help, I would not be able to complete the sketch generation project, which is one of the most interesting projects I had ever done.

I am grateful to my thesis committee members, Dr. Hui Zhao and Dr. Peng Liu, who spend their time reading my thesis and giving comments. With their help, I am able to refine the thesis and finish my master's study.

The emotional supports from my family and girlfriend are the driving forces for me to complete this thesis. Without them, I would not have gone so far.

Lastly, I would like to acknowledge the financial supports from the Department of Industrial Engineering at Penn State and my parents, which enable me to continue my graduate study.

Chapter 1

Introduction and Background

1.1 The Power of Data-Driven Methods

In the past, engineering problems are solved mostly by rule-based methods. Rule-based methods are easy to interpret, modify and justify. In many fields where reliability is the priority, rule-based methods are dominating. However, in recent years, with the emergence of Big Data, data-driven methods have become one of the most popular topics in the world. Rule-based methods are designed by humans, and humans need to create the algorithms carefully to satisfy all the rules. On the contrary, data-driven methods are designed to learn all the detailed rules from data, so that humans only need to design the learning algorithms instead of cover all the details of the problem that needs to be solved. The subject of designing and implementing these learning algorithms is called machine learning. By using machine learning methods, people can 1) save the time to design very complicated rule-based algorithms and 2) solve those problems that cannot be tackled by rule-based methods.

1.1.1 Early Model of Classification, Regression and Clustering

Three most popular categories of early statistical learning are: classification, clustering, and regression. Both classification and regression are supervised learning, whereas clustering and is unsupervised learning.

Most of the early non-Bayesian machine learning models either use distance metrics as criteria or use linear mapping. Many early methods perform very well for linear problems if the features are well-selected and independent. However, whenever the problem is linear, it will be very hard to solve. Support vector machine (SVM), which is one of the most popular traditional machine learning methods, supports non-linear boundaries, but the selection of non-linear function (or kernel) could be challenging because the amount of non-linearity is hard to determine.

Bayesian methods use the Bayesian rule to approximate posterior probability. Even nowadays, many Bayesian methods are still considered high performance. The advantage of Bayesian methods is that the abstraction could be very general compare to non-Bayesian methods.

1.1.2 Generative Models and Control Methods

Recently, with the development of modern deep neural networks, generative models and control models become very popular. Essentially, a generative model can generate data that follows the same distribution as the training data, given different distribution parameters. Different from classification and regression, generative model is aiming to "create", so that there is no "correct answer". In other words, the motivation of a generative model is to generate different outputs, while they can achieve some criteria.

The relationship between a generative model and a control model could be very close because action sequences proposed by control models can be considered as generated data. The action sequence needs to meet some criteria that an environment or problem defined.

1.2 Reinforcement Learning

1.2.1 Overview

There are different kinds of machine learning methods. Categorized by learning mechanics, there are three categories: supervised learning, unsupervised learning and reinforcement learning. Both supervised and unsupervised learning models learn directly from the given data. Different from supervised learning and unsupervised learning models, a reinforcement learning agent does not directly use data to learn. Instead, a reinforcement learning agent learns from environments, which will change according to the agent's actions and give feedback to the agent. The data in the context of reinforcement learning are called experiences, which are acquired by repeatedly executing the environment. A reinforcement learning algorithm learns from the reward given by the environment. Essentially, after the agent performs a good action, then the environment should sooner or later give a positive reward to the agent; after the agent performs a bad action, then the environment should give a negative reward (or punishment) to the agent. Compare to a supervised learning algorithm, reinforcement learning is indirectly "supervised", as the agent is not informed with the exact correct answer. As a result, reinforcement learning relies on a very well-designed environment, if not already exist, and feedback. Such feedback is commonly called reward, and the environment is usually a simulated environment. The learning mechanism of reinforcement learning agent is somehow similar to how humans learn because humans also learn from the environment and rewards. However, reinforcement learning is not the mechanism that humans use, due to the fact that reinforcement learning agents require a significantly larger amount of experience to learn a good policy than humans do.

Modern reinforcement learning can be dated back to the 1980s, but reinforcement learning has not become a popular machine learning area until 2015 when Deep Q Network (DQN)[1] has introduced. Before the introduction of the DQN, although many previous reinforcement learning methods produced exciting results, they are not applicable to many real-world problems. For example, Q-learning is a very successful method and has shown a strong ability to solve the grid puzzles; but it cannot handle control problems such as playing video games. As an improvement, the DQN combined deep neural network and the Q-learning, which showed the ability to play

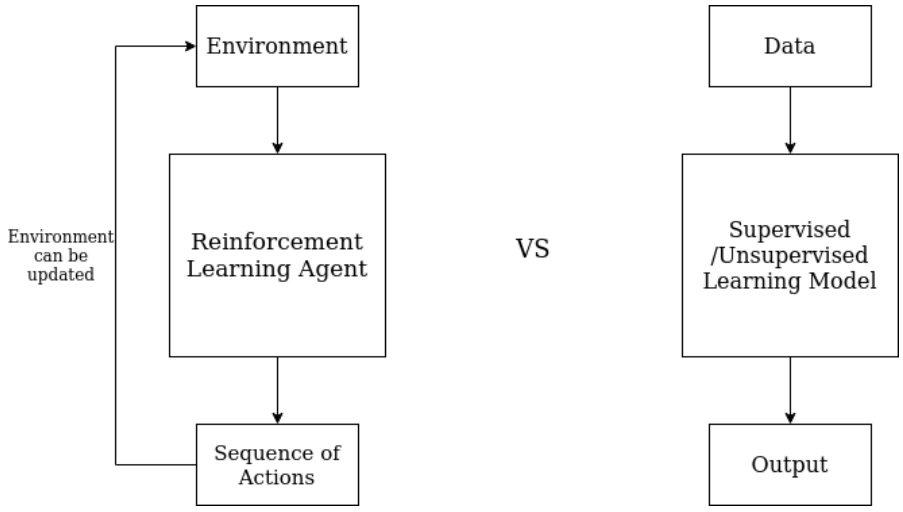


Figure 1.1: Difference between reinforcement learning and supervised/unsupervised learning

video games, only use the screen as the input. The difference between DQN and Q-learning is that a Q-learning agent has an explicit mapping that saves ALL state-optimal action pairs; while DQN uses a neural network to approximate the best action given an input state. As the number of possible inputs increases, the memory needed for Q-learning mapping will increase. If there are infinite numbers of possible input states, the memory will be infinitely large, which is not realistic.

After DQN, there are increasing number of reinforcement learning methods have been proposed. The state of the art methods can handle both inputs and actions with the infinite number of possibilities. Therefore, reinforcement learning could be a very powerful tool. Reinforcement learning is widely used in control problems, such as unmanned vehicle control and video game playing, but it can also be used in many other fields, such as generative task and tutoring systems.

1.2.2 Markov Decision Process and Temporal-Difference Learning

One assumption for reinforcement learning is that the control (or any other) problem needs to be solved could be modeled as Markov Decision Process (MDP). The major components of the MDP are the states and actions. In an MDP, by taking an action in a certain state, there would be probabilities to transit to other states. In real-world problems, the goal is usually to reach one or some states, from an initial state in MDP. Reinforcement learning is therefore to find the policy to complete the goal. In order to use reinforcement learning, algorithm designers need to carefully define the reward of each action given a state. The reward should be defined based on the problem and the environment where the MDP is formed.

The majority of reinforcement learning algorithms is so-called temporal-difference (TD) learning, which is a model-free method. In short, TD learning is to minimize the difference between estimated value and sampled value from the environment at a certain time step. The estimated value should be calculated from a value function, which takes states and actions as inputs. Note

that estimated value will take the next state as input, and this is why it is named as temporal difference. On the other hand, the sampled value should be acquired by summing up the rewards at different time steps. In order to bound the sum of rewards, there should also be a time-step discount factor.

One example of TD learning is Q-learning. When state space and action space are finite, Q-learning performs very well. Instead of finding the optimal policy directly, Q-learning is to estimate the Q-value (or discounted cumulative reward), which could be used to infer the optimal policy. Q-learning and its variants are the most widely used algorithms in the reinforcement learning community until 2014.

1.2.3 Monte Carlo Method and Policy Gradient Methods

Monte Carlo methods are to sample a combination of state, action, reward and next state each time when the policy needs an update. Policy gradient, which is a very popular category of reinforcement learning methods, is an example of Monte Carlo methods.

Policy gradient is referring to the computation of the gradient of the policy so that by updating the value function following the gradient, the performance will be maximized. Different from Q-learning, policy gradient methods are optimizing the policy directly.

Researchers did not pay much attention to policy gradient methods in the old days. The reason for the poor performance of policy gradient methods is the high variance generated by the Monte Carlo sampling process. Also, it is very hard to design the policy function before the use of neural networks. However, compare to Q-learning, policy gradient has the following advantages:

- Policy gradient directly update the policy function so that whenever Q-value is hard to estimate, policy gradient will have a better performance.
- Since Q-values needs to be calculated for each action, native Q-learning cannot handle continuous action space.

In conclusion, the development of reinforcement learning can be divided into following stages:

1. Early Monte Carlo methods
2. Q-learning and its variants
3. Policy gradient methods with Q-value (Deep Deterministic Policy Gradient)
4. Modified policy gradient methods (Trust-Region)

In the latter two stages, researchers are trying to reduce the variance of policy gradients calculated. Apparently, researchers are gradually shifting from Q-values to policy gradient. In essence, policy gradient methods are better in all perspectives once the sample variance could be reduced.

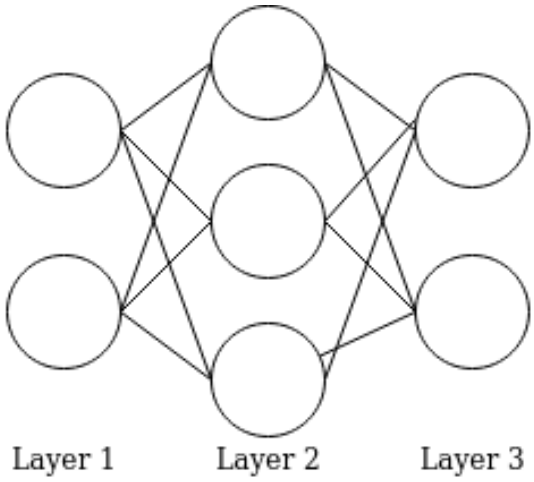


Figure 1.2: Neural network with 3 layers

1.2.4 Neural Network and Reinforcement Learning

The potential of reinforcement learning is not explored until the usage of neural networks. In the community, reinforcement learning with neural networks is called deep reinforcement learning.

Neural networks have many different names and abbreviations, including artificial neural network (ANN), deep neural network (DNN), multi-layer perceptron (MLP), etc. There are many discussions and arguments regarding the differences between the names, but generally speaking, all the names are referring to the same subject. Neural networks are often illustrated in multiple layers, where each layer is a n to m dimensional linear mapping with a non-linear function (activation function) applied after. Figure 1.2 shows the illustration of neural network with 3 layers. However, the essence of a neural network is a recursive mapping. Let σ to be the activation function and f to be the linear mapping, then a neural network with 3 layers is shown in Equation 1.1:

$$y = \sigma_3(f_3(\sigma_2(f_2(\sigma_1(f_1(x)))))) \tag{1.1}$$

Such recursive mapping is the reason why a neural network is so powerful and hard to optimize. Neural networks were not practical to use until 1970 when auto differentiation and back-propagation are introduced. Auto differentiation and back-propagation provide a numerical way to compute gradients of the parameters in neural networks. Even with these approaches, neural networks were not practical in production until 2010, when the computational power of CPUs and GPUs was sufficient enough to optimize a large neural network in a reasonable time.

The idea of using neural networks in reinforcement learning was realized in 2014 in deep Q network (DQN), where Q-values were approximated by a neural network instead of stored explicitly for every pair of states and actions. Since the Q-values could be estimated by a neural network, the state space could be continuous since then.

Later, neural networks are also used to represent the policy. Actor-critic structure is the most

widely used and the best structure for reinforcement learning. There are two neural networks, actor and critic, where the actor is the policy that can output action to maximize reward. Such actor neural network can be optimized by computing the policy gradient.

1.3 Heparin Dosing Control

Drugs that can have serious overdose consequences usually require strict dosing control, for example, insulin and heparin. Therefore, researchers and clinicians need to carefully design dosing protocols for these drugs. Tradition rule-based protocols take one major measurement, as the major criteria to decide the dose per unit weight for a patient. As an aid for traditional protocols, reinforcement learning has the potential to provide an automated dosing recommender system, which can reduce human errors and improve efficiency. One of the important goals of this research study is to design a model that can suggest when and how much heparin to inject, giving the vital sign and lab results of a patient.

Unfractionated heparin (UFH) is one drug that requires strict dosing control. UFH is usually used during medical operations and other treatments, where patients can be exposed to some medications or situations may help the formation of clots in the blood tube, to prevent blood clots. Essentially, the more UFH injected to patient per unit time per unit body weight, the long the clotting time. While having long clotting time can prevent the blood clots to form unexpectedly, it may also lead to excessive bleeding. As a result, all hospital has a very strict protocol to control the UFH injection rate. The majority of medical centers and clinics use activated partial thromboplastin time (aPTT) as criteria for their dosing protocol.

1.4 Image Generation

Image generation is the most popular topic of generative models. The causes behind the popularity of image generation include: 1) images or grid data are in a representation that neural networks handle really well; 2) the quality of generated data is very easy to verify; 3) the amount of available image data is huge.

Two most popular generative models are variational autoencoder (VAE) [2] and generative adversarial nets (GAN) [3].

1.4.1 Variational Autoencoder

Variational Autoencoder (VAE) is based on autoencoder, which is a very popular representation learning architecture. Autoencoder has two neural networks: an encoder and a decoder. In short, the data will be encoded to a latent vector by the encoder, and the latent vector will be decoded back to the original data. The latent vector fed into the decoder is considered to contain the parameters, or features, of the data distribution. The goal of an autoencoder is to train the networks so that the input of the encoder and the output of the decoder could be the same, and

then use the encoder to embed the data. However, in VAE, the goal is to use the decoder to generate new data, given a random latent vector.

The difference of VAE and autoencoder is that, the latent vector will be explicitly represented as parameter of a distribution (usually Gaussian). Therefore, the generated data from VAE will be different from the original ones.

1.4.2 Generative Adversarial Nets

GAN also has two neural networks, a discriminator and a generator. A generator takes a latent vector as input and output generated data, whereas discriminator takes the output data of generator (fake data) and real data as input, then classify whether the inputs are real or fake. One important characteristic of GAN is the discriminator will take the output of the generator as input, and therefore, the main contribution of GAN is the dedicated loss function to optimize the generator.

Although VAE can do generation tasks fairly well, GAN has one very important advantage against VAE. As VAE requires an assumption of the underlying distribution of data (usually Gaussian), GAN can fit the data without providing the detail of the distribution. This assumption can be a very huge disadvantage for complex generation tasks, as most of the data are apparently not normally distributed. As a result, all realistic image generation nowadays are all generated by GAN.

Chapter 2

Literature Review

2.1 Reinforcement Learning

Although reinforcement learning has been studied for several decades, it did not become popular until recently. The thrive of reinforcement learning recently is due to the implementation of artificial neural networks.

2.1.1 Development of Reinforcement Learning

The concept of reinforcement learning is introduced to solve learning problems whose environment can be modeled as Markov Decision Process (MDP), but the exact mathematical formulation is infeasible to obtain[4, 5]. In the early stages, reinforcement learning can only solve problems with discrete state space and action space[6, 7]. One of those early-stage popular reinforcement learning techniques is Q-learning [6]. Q-learning evaluates the weighted accumulative reward, also known as Q-value, of an action. In the machine learning community, Q-value is usually estimated by a function approximator, as there will be an infinite number of mathematical formulations if state space is continuous. With the advancement of neural networks, the Q-value estimation can be quite accurate by implementing neural networks as the approximator. However, Q-learning becomes extremely hard to train if the approximator is non-linear. This problem was not solved until the emergence of Deep Q Network (DQN)[1] introduced by Mnih et al as a variant of Q-learning. Although DQN uses the neural network as a Q-value approximator, the training is fairly stable due to the implementation of the following techniques: target network and experience replay. Both stabilizing techniques are widely used in the reinforcement learning community today.

However, in DQN, the action space is still discrete. Focusing on breaking the limitation of discrete action spaces in DQN, Timothy introduced Deep Deterministic Policy Gradients (DDPG)[8] based on Deterministic Policy Gradients (DPG) by Silver [9]. DDPG is a hybrid of policy gradient and Q-learning, where Q-value is used to calculate the policy gradient.

DDPG reduced the variance of policy gradient by using Q-values, but it may suffer from many drawbacks of Q-learning, including overestimation of Q-values, difficulty of calculating Q-values in some situations, etc. Therefore, pure policy gradient methods could be still better choices. Trust-region policy optimization (TRPO) [10] proposed an optimization method that has bounds on the policy gradient computed. However, TRPO is not practical for large scale problems, as optimizing objective function (loss function) with constraints is very hard and slow. However, TRPO shows the potential of policy gradient methods, when the variance can be controlled. Therefore, to accelerate and ease the training process of TRPO, proximal policy optimization (PPO) [11] is introduced. PPO proposed a clipped policy gradient to relax the constraints of TRPO.

Besides, recent advancement [12, 13, 14] on hierarchical reinforcement learning [15] shows great potential. Essentially, since hierarchical reinforcement learning architecture breaks the task into sub-tasks, it is ideal to perform complex tasks such as playing games [16] and controlling robots [17]. Many designing problems can be formulated into sub-problems as well, promising the possibility of implementing hierarchical structures.

2.1.2 Actor-Critic Structure

Most of the high-performance reinforcement learning algorithms use actor-critic structure, including DDPG, TRPO, PPO. However, the most widely known actor-critic algorithm is Advantage Actor Critic, which has two variants: Advantage Actor Critic (A2C) and Asynchronous Advantage Actor Critic (A3C) [18]. Actor-critic algorithms contain two neural networks, an actor and a critic. The actor is the policy, which outputs the actions; the critic output a value (Q-value or value function) that can evaluate the goodness of the action.

The major differences between all these algorithms are: 1) different operators (Q-value or advantage) to update policy, and 2) different strategies to reduce the variance of policy gradients. Here, only DDPG uses Q-value to update policy, and all others use advantage, which is Q-value subtracted by value function. TRPO and PPO enforce trust region and clipped loss to stabilize the policy gradient, whereas DDPG, A3C and A2C have no explicit stabilization. Thus, TRPO and PPO are essentially improved versions of A2C. For A3C, since it includes multiple workers training in a separate environment, A3C has the potential of parallel training.

2.2 Heparin Dosing

Unfractionated heparin (UFH) is widely used to prevent blood clots during medical treatment [19]. UFH is discovered by McLean [20] in 1916. In terms of pharmacokinetic, UFH interacts with many coagulation enzymes [21, 19], among which the factor Xa and thrombin are the most sensitive ones. Such interactions are the mechanics of the anti-coagulation effects caused by UFH.

Low and Medium Intensity (Arterial Thrombosis) Heparin Anticoagulation Dose Adjustments

aPTT (seconds)	Bolus/Hold	Infusion
<34	Give supplemental bolus if ordered & inform MD	↑ 100 units/hr = ↑ 2 mL/hr
34-37	Give ½ supplemental bolus if ordered & inform MD	↑ 100 units/hr = ↑ 2 mL/hr
38-44	0	↑ 50 units/hr = ↑ 1 mL/hr
45-54	0	NO CHANGE
55-64	0	↓ 50 units/hr = ↓ 1 mL/hr
65-84	0	↓ 100 units/hr = ↓ 2 mL/hr
85-100	Hold infusion 1 hour & inform MD	↓ 150 units/hr = ↓ 3 mL/hr
101-125	Hold infusion 1 hour & inform MD	↓ 200 units/hr = ↓ 4 mL/hr
>125	Hold infusion 1 ½ hour & inform MD	↓ 200 units/hr = ↓ 4 mL/hr

High Intensity (Venous Thromboembolism) Heparin Anticoagulation Dose Adjustments

aPTT (seconds)	Bolus/Hold	Infusion
<34	Give supplemental bolus if ordered & inform MD	↑ 100 units/hr = ↑ 2 mL/hr
34-44	Give ½ supplemental bolus if ordered & inform MD	↑ 100 units/hr = ↑ 2 mL/hr
45-54	0	↑ 50 units/hr = ↑ 1 mL/hr
55-70	0	NO CHANGE
71-85	0	↓ 100 units/hr = ↓ 2 mL/hr
86-100	Hold infusion 1 hour & inform MD	↓ 150 units/hr = ↓ 3 mL/hr
101-125	Hold infusion 1 hour & inform MD	↓ 200 units/hr = ↓ 4 mL/hr
>125	Hold infusion 1 ½ hour & inform MD	↓ 200 units/hr = ↓ 4 mL/hr

Figure 2.1: Example of a UFH dosing protocol.

2.2.1 Heparin Dosing Protocol in Practice

Most of the hospitals use activated partial thromboplastin time (aPTT) as major criteria to control UFH dosing, as suggested by many researchers [19, 22, 23]. Although enzymes such as factor Xa and thrombin could also be used as the criteria, it can cost less to measure aPPT. Therefore, the UFH dosing protocols are to maintain the aPTT level. Figure 2.1 shows an example of UFH protocol at a hospital [1].

2.3 Image Generative Model

2.3.1 Generative Sketching

Sketch generation can be divided into two categories, stroke-based and pixel-based. Although generating sketches using pixels has several disadvantages, there are still some works using the pixel-based method to generate sketches. One example is [24] proposed by Dering and Tucker in 2017, which is a GANs based model. Using the pixel-based method, [24] has impressive results, but with the problem of low fidelity. Another example is proposed by Burnap et al. mentioned before, which introduces a pixel-based model to generate product form design images based-on VAE. [25].

In the field of visual generation, although the majority of the previous works are pixel-based image generation [3, 2, 26], there are also some previous works focusing on sketch generation. [27, 28, 29, 30]. Sketch-RNN [27] proposed by Ha and Eck can generate vector images. The model

[1] https://www.uwhealth.org/files/uwhealth/docs/pdf2/Heparin_Infusion_Guideline.pdf

can be trained on vector images. Sketch-RNN is essentially a VAE[2] architecture, with RNN decoder and encoder. What neural networks are learning is the temporal relative position of the "pen" when a person is sketching. The advantage of Sketch-RNN is that the spacial features of the training image are not important; instead, Sketch-RNN extracts temporal information of the trajectory of the "pen". Other than Sketch-RNN, there are other methods focusing on generating sketches from reference images [31, 30, 28, 29], which are not quite generative methods. Zhou introduced a reinforcement learning technique named Doodle-SDQ [31] to doodle an image by sketching. One advantage which Doodle-SDQ poses in comparison to Sketch-RNN is that the training data can be the pixel-based image. However, Doodle-SDQ is not quite a generative model, since it requires an input reference image for each of its generations. Some other works such as [29, 30, 28] can also generate sketches using input image, but they are neither stroke-based nor generative models.

2.3.2 Realistic Image Generation

The advancement of neural networks drives many researchers to study realistic image generation. After the introduction of GAN [3] in 2014, high fidelity realistic image generation starts to thrive. Wasserstein GAN (WGAN) [32] introduced Wasserstein distance as the distance measure instead of Kullback-Leibler divergence in regular GAN, which could provide a stabler training and lower variance in the loss function. Deep Convolutional GAN [33] developed a general GAN architecture using deep convolutional neural networks. Style GAN [34] introduces a new concept, style, to improve the training of the GAN. Self-Attention GAN [35] improves the detail fidelity by incorporate spectral normalization to the generator.

Chapter 3

Reinforcement Learning in Unfractionated Heparin Dosing Control

In this chapter, the reinforcement learning application on dosing control is covered. The drug selected is unfractionated heparin (UFH), which is a very commonly used drug to prevent blood clots. UFH usually require strict dosing control, because the overdose of UFH can cause bleeding, which may delay other treatments and cause dangerous situations. In practice, UFH dosing protocols are based on the experience of the clinician and blood lab tests.

UFH is commonly used in the intensive care unit (ICU), where patients' vital signs, lab test results, and drug usage histories are very well recorded. Experiments in this chapter use the data extracted from MIMIC III (Medical Information Mart for Intensive Care III)[36].

In ICU, the vital signs are recorded very frequently (usually every 30 minutes), but lab tests are not frequently recorded. Therefore, the major challenge in applying reinforcement learning in ICU UFH dosing control is the difference in frequency of records collection, which may lead to null values in some attributes in some time steps. To reduce the number of null values, attribute selection will be conducted before applying the reinforcement learning algorithm.

The difference in frequency of records collection is also one of the motivations of using reinforcement learning. In practice, most of the UFH dosing protocols rely on the value of activated partial thromboplastin time (aPTT), which is an attribute with a low sample rate. Therefore, the change in the injection rate can only be made at a low frequency. One the other hand, reinforcement learning could also use attributes with high sample rates, and could provide suggestions of injection rate more often. Another motivation for using reinforcement learning for UFH dosing control is the fact that physician's suggestions could be improper due to the low sample rate of aPTT. This motivation will be explained in detail in Section 3.3.

The key challenges of this projects are:

- No actually or virtual environment available for the agent.
- Unsatisfying data quality in UFH related aspects.

3.1 Background of Unfractionated Heparin

Unfractionated heparin (UFH), or simply heparin, is injected intravenously (IV) in ICU to prevent blood clots during treatment. Due to the serious consequences caused by UFH overdose, UFH dosing is well studied since the 1990s [23, 37, 38]. In practice, clinicians in ICU inject IV UFH in a basal and bolus manner[39, 40]. Basal represent a continuous infusion, which is controlled by an infusion rate; while bolus is a one-time injection of UFH in a large amount.

In this research, the reinforcement learning agent uses attributes, including the vital signs and lab test results, as states, and output actions, which is the UFH injection. The reward of the reinforcement learning agent is determined by activated partial thromboplastin times (aPTT), which is a commonly used pharmacodynamic criterion for UFH dosing control. For a reinforcement learning agent, it is not ideal to consider both bolus and basal injection data, because: 1) bolus is usually not used for most of the patients and 2) basal has a dominating effect compare to bolus. Specifically, if a patient's aPTT is not too off the normal range, then bolus will not be applied so that most of the UFH injections of a patient is decided by using basal.

3.2 State Construction and Attributes Selection

Before applying reinforcement learning, a statistical analysis of attributes is conducted to determine which attributes should be used to construct state. During the analysis, injections of bolus and basal will both be considered. Bolus and basal are distinguished by whether a UFH injection records contain the injection rate in the MIMIC III database.

Medical records of 3211 patients with 22 attributes extracted from the MIMIC III database are used for the experiment. For reinforcement learning agents conducting UFH dosing control, in order to ensure the performance, the inter-dependency between attributes should be minimized, and the attributes should be highly related to the UFH injection. The following sections show the detail of the methodology and the results.

3.2.1 Methodology

To see whether an attribute is suitable for constructing state, three questions need to be evaluated:

1. Whether the attribute will change responding to the UFH injection?
2. What is the time delay of the change responding to the UFH injection?
3. Whether the attribute is dependant to other attributes?

The first two questions can be evaluated by using correlation coefficient and mutual information. Let an attribute forms a time series $\mathbf{s} = \{s_t\}$, and an action is also a time series $\mathbf{a} = \{a_t\}$,

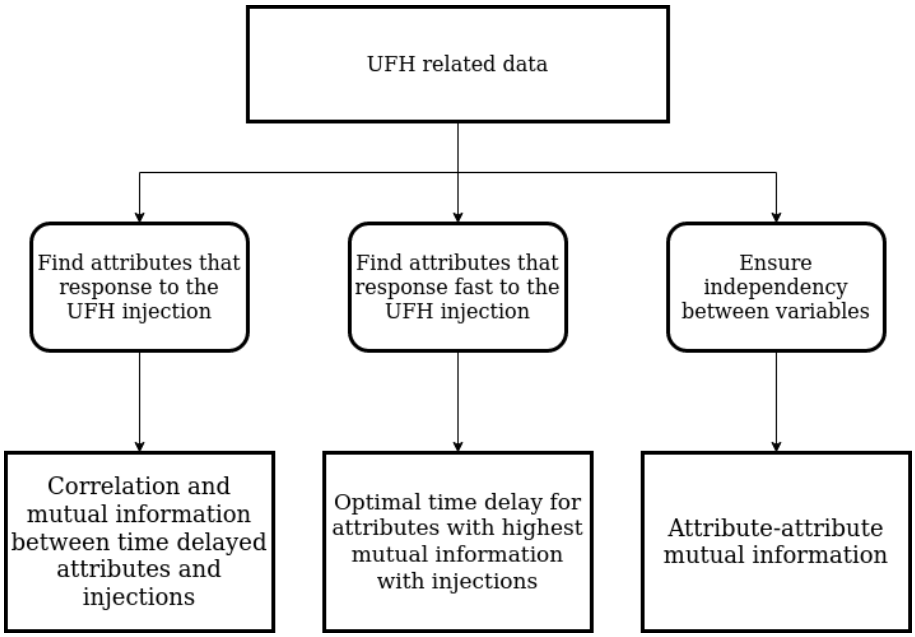


Figure 3.1: A flow diagram of the methodology for attributes analysis

where t is the time step. Those attributes with high correlation and/or mutual information are favored.

To address the third question, a mutual information matrix is constructed, where rows and columns are attributes. Those attributes that have high mutual information are redundant.

The summary of the methodology is illustrated in Figure 3.1.

3.2.1.1 Correlation Coefficient

The correlation coefficient is a widely used method to investigate linear relationships between two variables. Although not intended for time series analysis, the correlation coefficient can indicate the relationship between time series properly if the time series is without trends. Since most of the attribute time series are either vital signs or lab test measurements, there should not be trends. Let the correlation coefficient between attribute series and action series to be $\rho_{\mathbf{a},\mathbf{s}}$, then:

$$\rho_{\mathbf{a},\mathbf{s}} = \frac{\mathrm{E}\left[\left(a_t - \mu_{a_t}\right)\left(s_t - \mu_{s_t}\right)\right]}{\sigma_{a_t}\sigma_{s_t}} \tag{3.1}$$

where μ is the mean, and σ is the standard deviation. Note that correlation coefficient can be either positive or negative.

3.2.1.2 Mutual Information

In information theory, mutual information is a measure used to quantify the amount of information of a variable that could be obtained giving another variable. The advantage of mutual

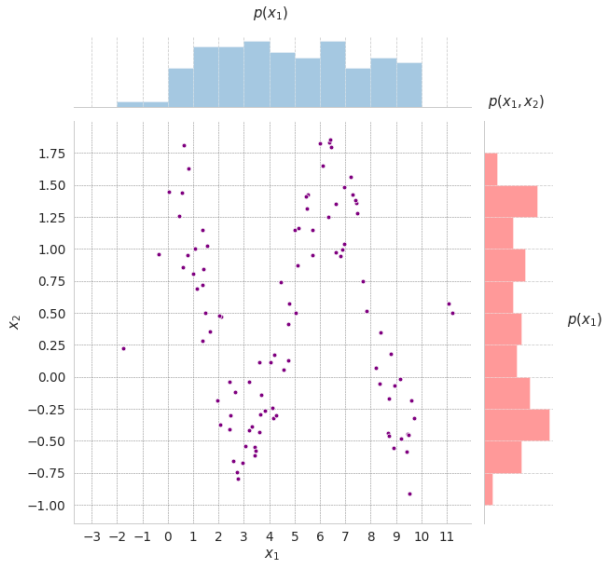


Figure 3.2: An illustration of calculation of mutual information

information is that it could measure the nonlinear relation between two variables, and the underlying transformation function between two variables will not cause a significant influence on the result. Mutual information $MI_{\mathbf{a},\mathbf{s}}$ between action and each attribute is:

$$MI_{\mathbf{a},\mathbf{s}} = \sum_{a_t \in \mathbf{a}} \sum_{s_t \in \mathbf{s}} p\left(a_t, s_t\right) \log \left(\frac{p\left(a_t, s_t\right)}{p\left(a_t\right) p\left(s_t\right)} \right) \tag{3.2}$$

The log base in Eq. 3.2 is 2, resulting in the unit of mutual information to be bit. In practice, the values of time series need to be binned to ensure finite number of unique values shown in the time series in order to calculate $p\left(a_t\right)$, $p\left(s_t\right)$ and $p\left(a_t, s_t\right)$. Fig 3.2 illustrates the calculation of mutual information. The maximum number of unique values in a time series used in this study is 16.

3.2.1.3 Optimal Time Delay

Mutual information with delayed attributes can be also used to determine the optimal time delay. Specifically, the amount of time delay on attributes that can result in the highest mutual information with the responses is desired. Let τ to be the number of time steps delayed, so that $\mathbf{s}(\tau)$ represent an attribute time series with τ steps delayed. The optimal time delay τ^* is:

$$\tau^* = \arg\max_{\tau} MI_{\mathbf{s}(\tau),\mathbf{a}} \tag{3.3}$$

3.2.2 Experiment Design

The three questions introduced in the previous section are addressed in the experiment. MIMIC (Medical Information Mart for Intensive Care) III database [36], which is a popular ICU database

created by Beth Israel Deaconess Medical Center, is used in experiment. Medical records, including at least 48 hours of patients' vital signs, lab test results, injection events, are saved in the MIMIC III database during patient's stay in ICU. Attributes frequently used in the literature related to UFH problems are selected in this study, including Albumin, Arterial CO2 Pressure, Arterial O2 Saturation, Blood Urea Nitrogen (BUN), Creatinine, Glasgow Coma Scale (GCS), Heart Rate, Hematocrit, Hemoglobin, Heparin Dose, International Normalized Ratio (INR), PH, aPTT, Platelet Count, Prothrombin Time (PT), Respiratory Rate, Temperature, Total Bilirubin, Troponin-T, and White Blood Cell (WBC). In terms of the injection, basal is the injection rate of heparin sodium, and the bolus is the injection amount of heparin sodium.

A total of 3211 patients who have UFH injected during the ICU stay have been selected. In terms of data processing, time series are firstly formed at a time interval of 1 hour. The time interval is decided based on the fact that the most frequently sampled lab measurements are sampled hourly [41]. Missing values are imputed by the sample-and-hold (zero-order hold) method. In case there is no initial value, average values are used.

It is very common to use sample-and-hold method in missing values for vital signs and lab tests in dosing problems, because normally clinicians will assume the attributes to be in normal range or not changed since last test.

All attributes will be delayed for 1 hour (1-time step) when they are used to calculate the correlation and the mutual information with the injections because it is intuitive that the effect will take place after the injection.

3.2.3 Results of Attributes Analysis

3.2.3.1 Relations between Attributes and Injections

Figure 3.3a shows the average correlations between 1-hour-delayed attributes and injections. The table in Figure 3.3a is ranked based on correlation with bolus.

Only aPTT has significant dependency with bolus. It not surprising that aPTT (or PTT) has negative correlations with the bolus, because the clinician will inject bolus UHF if the clotting time is low. However, another clotting factor, INR, have a small correlation with the bolus. This indicates that aPTT and INR/PT are fairly independent.

Different from bolus, the inter-dependencies between each attribute and basal injection are more informative. aPPT has a high correlation with the basal but not dominant anymore. Vital signs except for heart rate seems very not related to the basal injection. There are also many other attributes seems quite important to the UFH basal injection. However, many attributes that are negatively correlated with bolus become positively correlated with basal, or vice versa. This observation may be because the basal protocol is delayed, which means, even the aPPT is increasing after injection of the bolus, the basal may still be kept at a high level until later.

Figure 3.3b shows the mutual information of attributes with the basal and bolus. The table is also ranked based on the mutual information with bolus. Ranking slightly changed, which implies nonlinear relations exist.

Attribute	Cor with bolus	Cor with basal
Heart Rate	0.00125	-0.04673
WBC	0.00115	-0.02227
PTT	-0.00097	0.13479
GCS - Eye Opening	-0.00075	0.01926
Total Bilirubin	0.00072	-0.02524
Hemoglobin	0.00058	-0.01079
PH (Arterial)	0.00057	0.01268
Prothrombin time	0.00044	-0.03603
Troponin-T	-0.00041	-0.00086
INR	0.00034	-0.03831
Hematocrit (serum)	0.00033	-0.01944
Arterial CO2 Pressure	-0.00033	-0.00443
Respiratory Rate	0.00030	-0.01290
GCS - Verbal Response	-0.00030	0.02517
Platelet Count	0.00024	0.02324
Arterial O2 Saturation	0.00015	-0.00391
Creatinine	-0.00009	-0.02375
Albumin	-0.00006	0.00457
Temperature Fahrenheit	-0.00001	0.00589
BUN	-0.00000	-0.00690
GCS - Motor Response	-0.00000	0.03391
Heparin Dose (per hour)	0.00000	0.00196

(a) Average correlation between attributes and injections with descending order in the magnitude of correlation with bolus.

Attribute	MI with bolus	MI with basal
Respiratory Rate	0.00204	0.41974
Heart Rate	0.00186	0.48786
Temperature Fahrenheit	0.00153	0.44538
BUN	0.00091	0.22495
Platelet Count	0.00090	0.24784
WBC	0.00085	0.21558
PH (Arterial)	0.00081	0.14861
Arterial CO2 Pressure	0.00080	0.14231
PTT	0.00068	0.32972
Prothrombin time	0.00066	0.16116
INR	0.00066	0.13351
Hemoglobin	0.00065	0.18612
Creatinine	0.00063	0.14807
Hematocrit (serum)	0.00058	0.15731
GCS - Verbal Response	0.00044	0.08136
Total Bilirubin	0.00036	0.06237
GCS - Eye Opening	0.00028	0.09241
GCS - Motor Response	0.00025	0.07104
Troponin-T	0.00022	0.10896
Albumin	0.00005	0.05306
Arterial O2 Saturation	0.00005	0.06362
Heparin Dose (per hour)	0.00000	0.00407

(b) Average mutual information between attributes and injections with descending order in the magnitude of mutual information with bolus.

Figure 3.3: Results of relations between attributes and injections

Table 3.1 includes the ratings. Mutual information values are bold if they are greater than the median.

3.2.3.2 Optimal Time Delay

Optimal time delay for each attributes are shown in Table A.1 in the Appendix A. The optimal time delay is when the average correlation between bolus and injection is the greatest. Although bolus is not quite useful for learning model, since bolus is one-time injection of large amount, the effects of bolus will be less affected by other factors, and thus more suitable for time delay analysis.

From the result, only GCS-Eye Opening attribute has a significant optimal time delay of 7 hours. Arterial O2 Saturation, Heart Rate have about 4 hours. For many modern reinforcement learning methods, the policies can learn delayed reward. Therefore, as long as the optimal time delay is not massive, the learning model will be able to handle slight time delay on attributes.

Notice that all the highly relevant attributes shown in Table 3.1 have minimal optimal time delay. Therefore, the state for the reinforcement learning model could be high relevance attributes

Table 3.1: Rating for each attributes

Categories	Attributes	MI with Basal	MI with Bolus
High Relevance			
	Heart Rate	**0.48786**	**0.00186**
	Temperature Fahrenheit	**0.44538**	**0.00153**
	Respiratory Rate	**0.41974**	**0.00204**
	Platelet Count	**0.24784**	**0.0009**
	BUN	**0.22495**	**0.00091**
	WBC	**0.21558**	**0.00085**
	PH (Arterial)	**0.14861**	**0.00081**
	Prothrombin time	**0.16116**	**0.00066**
	PTT	**0.32972**	**0.00068**
Mid Relevance			
	INR	0.13351	**0.00066**
	Hemoglobin	**0.18612**	0.00065
	Hematocrit (serum)	**0.15731**	0.00058
	Arterial CO2 Pressure	0.14231	**0.0008**
Low Relevance			
	Troponin-T	0.10896	0.00022
	Creatinine	0.14807	0.00063
	GCS - Eye Opening	0.09241	0.00028
	GCS - Verbal Response	0.08136	0.00044
	GCS - Motor Response	0.07104	0.00025
	Arterial O2 Saturation	0.06362	0.00005
	Total Bilirubin	0.06237	0.00036
	Albumin	0.05306	0.00005
	Heparin Dose (per hour)	0.00407	0
Medians of MI		**0.14834**	**0.000655**

and mid relevance attributes with small optimal time delay. The result in next section, inter-attributes dependency, can indicate whether there is redundancies in the attributes.

3.2.3.3 Inter-attributes Mutual Information

Figure A.1 in Appendix A shows the inter-attribute mutual information. Most of the attributes are independent, but few are fairly correlated. For example, INR and Prothrombin Time (PT) are correlated, which both are used to measure one of the factors of clotting time. Since PT is among the high relevance attributes, INR here will not be used for constructing the state of the reinforcement learning agent. Another example is Hemoglobin and Hematocrit, and therefore the one with higher mutual information will be used for the state construction.

3.2.3.4 Summary

From the results of three aspects, only attributes with following properties will be used to construct the state:

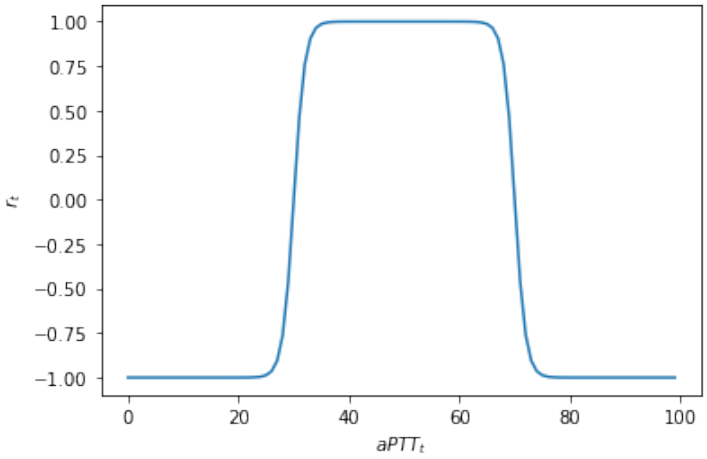


Figure 3.4: Illustration of reward function

- Among the categories of high relevance and mid relevance to UFH injection.
- Minimal (less than 4 hours) optimal time delay.
- Minimal inter-dependency

Therefore, the attributes selected for the state construction are: Heart Rate, Temperature Fahrenheit, Respiratory Rate, Platelet Count, BUN, WBC, PH, Prothrombin Time, aPPT (PTT), and Hemoglobin. Some of the lab tests are discarded because there are not enough tests.

3.3 Reward and Architecture

3.3.1 Reward Function

The reward is the most important component in reinforcement learning and it will directly influence the performance of the agent. In this UFH dosing control application, usage of the reward is one of the most important motivations of using reinforcement learning rather than supervised learning.

For a supervised learning model, the label is an important component. Labels in supervised learning are essentially the targeted value to output, given a set of attributes as input. To solve UFH dosing problem using supervised learning, the only information that can be used as a label is the UFH injection administrated by the clinician. However, as stated at the beginning of this chapter, some of the actions suggested by the clinician may not be proper. Specifically, because the aPTT value is not available at every hour, the injection rates suggested by the clinician could be unstable and conservative. As a result, if the actions suggested by the clinician are used as labels, the supervised learning model could learn an improper dosing policy.

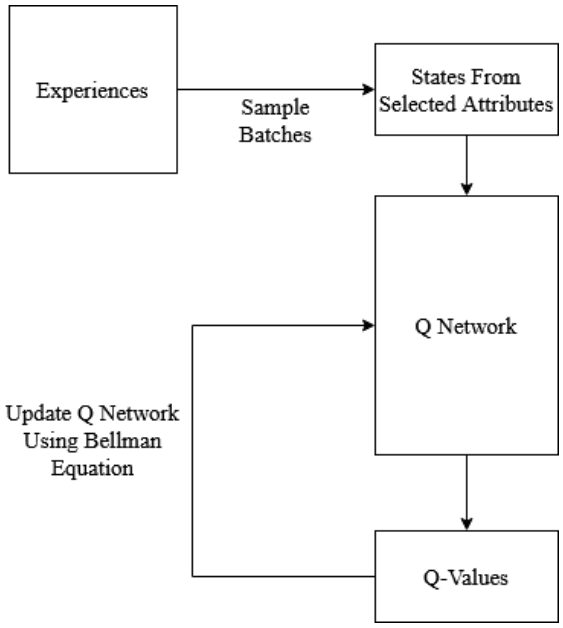


Figure 3.5: Flow diagram of the training

On the other hand, reinforcement learning can potentially learn a better policy. An indirect indication of the proneness of the injection rate is included in the data: aPTT value. Thus, a reward function could be designed based on the aPTT value and provide the rewards for the reinforcement learning agent to learn from.

The medical center where the MIMIC data collected from does not publish their protocol, but from many previous works [42, 43, 44], it is very common to set the target range of aPTT to be 60-100. Let the targe range of aPTT be 60-100 here for now. The reward function should give positive value when the aPTT is within the range, and give negative value when the aPTT is outside the range. Besides, the reward function should be a continuous and differentiable function to make the training stable. Let r_t to be the reward at time step t, then r_t can be calculated from aPTT value at time step t:

$$r_t = \frac{r_{max}+1}{1+e^{-(aPTT_t)-60}} - \frac{r_{max}+1}{1+e^{-(aPTT_t)-100}} - r_{min} \tag{3.4}$$

where r_{max} and r_{min} are the upper and lower bounds for the reward value. Let the reward value is between -1 and 1, then the reward function in Equation 3.4 can be illustrated in Figure 3.4.

3.3.2 Q Learning and Bellman Equation

Based on Q-learning and DQN, the reinforcement learning agent here uses a neural network to approximate Q-value q_a for each action a, which is the maximum expected cumulative reward R_t given a state-action pair. A important assumption to bound the cumulative reward is to have a discount factor γ, so that the cumulative reward at time t until terminate time T is:

$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$.

A neural network $f(s)$ will output a vector contains Q-values of each actions $\mathbf{q}$:

$$\mathbf{q} = f(s) \tag{3.5}$$

Therefore, Q-values for each action $q_a \in \mathbf{q}$ given state s is denote as $Q(s, a) = q_a$. The agent is to take the action with the highest Q-value q_a, so that the optimal action q^* at state s is:

$$q^* = \pi^*(s) = \arg\max_a q_a \tag{3.6}$$

The policy of Q-Learning is explicit. As shown in Equation 3.6, the optimal policy of Q-learning π^* is to select the action with the highest Q-value, and thus the approximation of Q-value needs to be proper.

To approximate the Q-values given a state, the most used method is a temporal-difference method, that can recursively estimate the Q-values of a state, given the optimal Q-value of the next state. This methodology is also known as dynamic programming. Let s' is next state, the recursive equation is called the Bellman Equation:

$$Q^\pi(s, a) = r + \gamma Q^\pi(s', \pi(s')) \tag{3.7}$$

and in here, the equation can be modified to:

$$Q(s, a) = r + \gamma \max_a(Q(s', a)) \tag{3.8}$$

by combining the explicit policy in Equation 3.6 and Bellman Equation shown in Equation 3.7.

Equation 3.8 can be used to estimate the Q-values and thus update the Q-value approximation. Since a neural network is used, Equation 3.8 is used to construct the loss function. By minimizing the mean squared error between Q-values $Q(s, a)$ estimated from Equation 3.8 and Q-values estimated from the neural network $f(s)$, the neural network can be trained. Formally, the loss function L of the neural network is:

$$L = (r + \gamma \max_a(Q(s', a)) - f(s, a))^2 \tag{3.9}$$

where s' is next state and $Q(s', a)$ is estimated by neural network: $Q(s', a) = f(s)'$.

3.3.3 Q-Network

The neural network used is a fully-connect neural network. For all layers except for output layer, a rectified linear unit (ReLU) activation function is used. ReLU is a very commonly used activation function, which is defined as:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.10}$$

Layer	Unit	Activation Function
Dense	512	ReLU
Dense	512	ReLU
Dense	512	ReLU
Dense	128	ReLU
Dense	64	ReLU
Dense	Action Dimension	None

Table 3.2: Q-Network Architecture

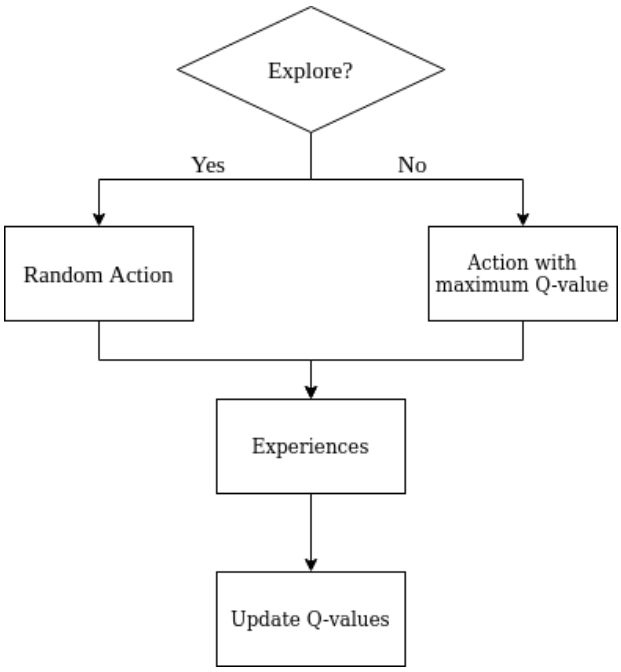


Figure 3.6: Flow diagram of policy used in standard training process for Q-learning

The architecture of the neural network is presented in Table 3.2: Note that the last layer output the Q-values of actions, so there is no activation function. The number of unit of output layer is depending on the number of bins to create for the continuous action, which will be discussed in Section 3.4.1.

3.3.4 Off-Policy Training

The off-policy training essentially means that the policies during training and executing are different. The policy used in a standard training process of Q-learning and DQN is shown in Figure 3.6. On the other hand, as shown in Equation 3.6, the policy of Q-learning should never explore, but the policy during training include exploration. The intuition behind exploration is to make sure all possible actions are tested, and the epsilon-greedy strategy is usually used for exploration.

Although the motivations of using reinforcement learning to solve the UFH dosing problem

[0,10)	[10,20)	[20,30)	[30,40)	[40,50)	[50,60)	[60,70)	[70,80)	[80,90)	[90,∞)

Figure 3.7: A illustration of bins of actions

are sufficient, a huge challenge is that the environment is not available. Without an environment, the agent will not able to execute the environment and gain their own experiences, which implies the reinforcement learning agent needs to be trained off-policy. This is one very strong motivation to use Q-learning and DQN here because Q-learning is an off-policy algorithm.

The agent will not execute the environment and thus the number of experiences is fixed. As shown in Figure 3.5, a memory buffer will store all the experiences available from the MIMIC database, and train the neural network using the loss function in Equation 3.9. The training time will be short because 1) there are limited experiences and 2) there is no executable environment.

3.4 Experiments

Two experiments are conducted to test the performance of the reinforcement learning model. The experiment results show that the clinician tends to use a relatively conservative dosage compare to the agent, and the agent tends to use a stabler dosage. By changing the target aPTT range and comparing the two experiments, the ability of agent understanding the relationship between action and states is illustrated.

3.4.1 Action Binning

Deep Q-learning is a reinforcement learning algorithm that only supports discrete action space. Therefore, it is necessary to bin the action into discrete values. The action is the amount of UFH injected per hour per unit body weight (e.g. units/hour/lb), which is divided into 10 bins, as shown in Figure 3.7. Note that the last bin is overflowed so that all values over 100 will be set to 100. Essentially, all action values over 90 will be categorized in one bin.

The distribution of action is skewed left, with the majority of actions are very small, implying the clinicians are giving conservative doses. The size of the bin and the distribution of the action values are shown in Figure 3.8

The size of the bin is determined by balancing the difficulty in training and level of precision. It is intuitive to have as many bins as possible to train an agent that can have precise control, but by making the action discrete, the neural network will not assume numerical relationships between bins, which can bring challenges in training. Introducing a large number of bins may result in the agent will not learn the numerical relationship between bins effectively. After several experiments, it comes out 10 bins is a good number to use.

Since the number of bins is 10, there are 10 Q-values need to be approximated. As a result, the dimension of the output layer (units in the layer) of Q-network will be 10, which will output 10 Q-values.

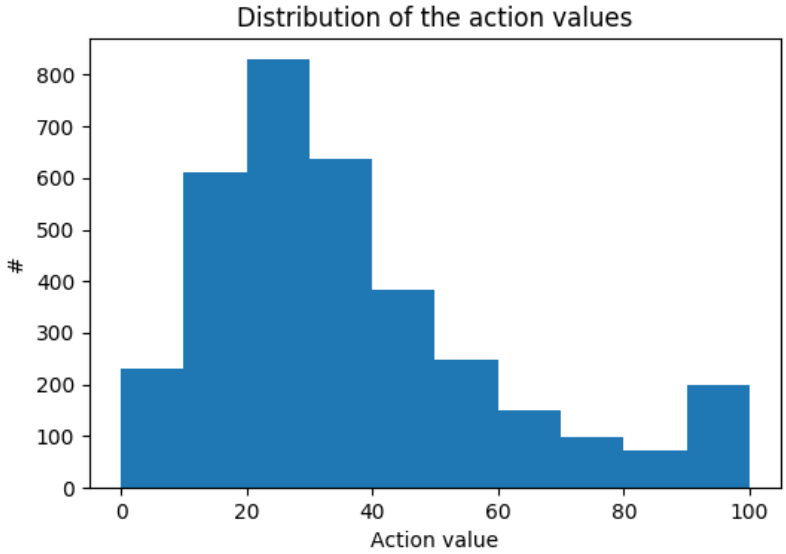


Figure 3.8: Histogram of action values with 10 bins.

Parameter	Experiment 1	Experiment 2
Target aPTT	60-100 sec	30-70 sec
Batch Size	30	30
Learning Rate	0.001	0.001
Training Steps	50000	50000
Discount Factor	0.95	0.95

Table 3.3: Two experiment set-ups

3.4.2 Two Experimental Set-Ups

To illustrate whether the agent can learn the numerical relationship between action bins, experiments are done in two different set-ups. The detail is shown in Table 3.3. Essentially, by controlling the target range of aPTT, the agent is expected to put in a different amount of injection. For example, if the target range of aPTT is low, then the agent should be putting less UFH, as aPTT will increase as the injection amount increase.

The target range of aPTT is controlled by adjusting the reward function shown in Equation 3.4. Equation 3.4 is an example of the reward function when the target range is 60-100, which is the range for Experiment 1. For Experiment 2, Equation 3.4 can be modified to:

$$r_t = \frac{r_{max}+1}{1+e^{-(aPTT_t)-30}} - \frac{r_{max}+1}{1+e^{-(aPTT_t)-70}} - r_{min} \tag{3.11}$$

and the comparison is illustrated in Figure 3.9.

Essentially, the interpretation of two different experimental set-ups is that the agent trained in Experiment 2 will be more conservative compared to the agent trained in Experiment 1.

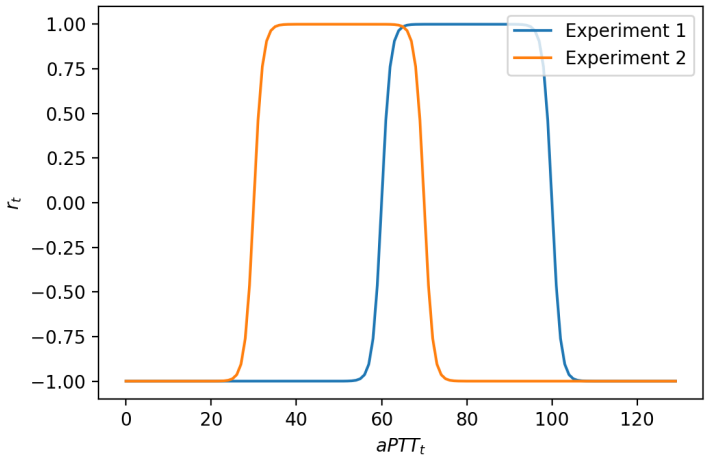
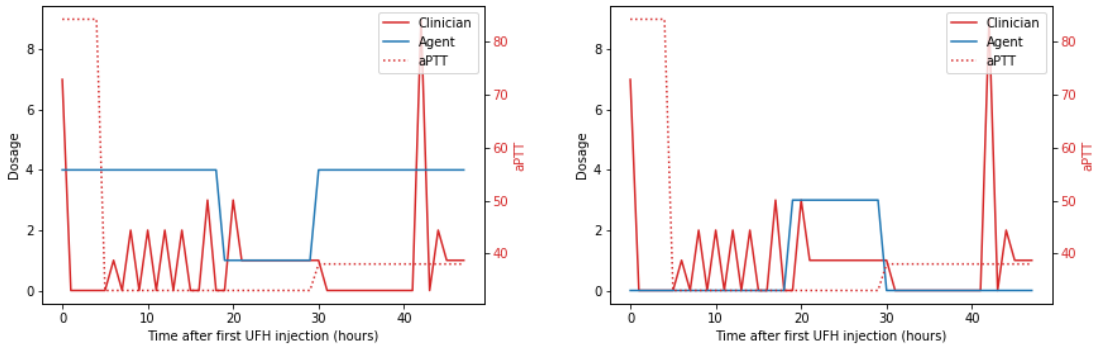


Figure 3.9: Comparison between reward functions



(a) Result of Experiment 1

(b) Result of Experiment 2

Figure 3.10: Comparisons between actions by agent and clinician on a patient

3.4.3 Results and Analysis

Since there is no executable environment, the trend of Q-value as the training steps increase becomes less important. Figure A.2 and Figure A.3 in the Appendix A show the trend of Q-values.

Figure 3.10 shows the comparison between the agent and clinicians. In Figure 3.10, a patient is used as an example to illustrate the difference between actions by agents and clinicians. The red line represents clinicians' actions, and the blue line represents agent's actions.

In Experiment 1, the agent suggested more injections than clinicians did. The aPTT values of the patient are mostly around 30 to 40, meaning that if the target range is 60-100, the patient will need more UFH injection. Essentially, the agent is trying to do so and adjust the aPTT value higher. In Experiment 2, the agent suggested a similar amount of injections as the clinicians did, which controls the aPTT at a low level.

	Mean Absolute Error
Experiment 1	24.4 unit/hr/lb
Experiment 2	15.3 unit/hr/lb

Table 3.4: Mean absolute error between actions by clinicians and actions by agents

The mean absolute error (MAE) between actions by clinicians and actions by agents are calculated for each experiment set-up, which is shown in Table 3.4. From the MAEs, it is intuitive to infer that agents in Experiment 2 set-up are closer to clinicians, which is conservative and trying to control the aPTT level at a low level. Also, from figure 3.10, clinicians tend to not maintaining the injection rate so that the variation of clinicians' actions is high.

Chapter 4

Reinforcement Learning in Stroke-based Sketch Generation

Many engineering design tasks involve creating early conceptual sketches, which do not require exact dimensions. Such sketches can be generated by computers today, and sketch generation can be a great inspiration for creative design during the early stages.

When human designers create sketches, they usually consider strokes as basic elements, which is very different from what computers consider. Despite the existence of vector images, the majority of digital images are saved in form of pixels, or grid data. It is hard for human to build intuition between images and pixels, especially in terms of sketches. Therefore, in order to provide better inspiration for human designers, the process of generating strokes will also be valuable.

The fact that most of the sketches are saved as a normal image (pixel-based grid data) creates the motivation of using reinforcement learning. Usually, a unsupervised generative model require the training data and generated data are in same form, because nearly all generative models are essentially try to approximate the distribution of the training data. However, by encoding the action space of reinforcement learning, the limitation mentioned above could be resolved. However, there are still challenges of using reinforcement learning for sketch generations:

- Difficulty to assign rewards due to the challenge in evaluating the quality of generated sketches.
- Lack of randomness may cause the generations to be tedious.
- Difficulty to ensure a smooth continuous action space.

(a) Pixel-based sketch of a tree

(b) Stroke-based sketch of a tree

Figure 4.1: Illustration of Stroke-based and Pixel-based

4.1 Stroke-based VS Pixel-based

A stroke-based sketch contains strokes such as lines and curves, and some areas on the canvas are completely blank. On the other hand, a pixel-based sketch, or image, is hard to break down into basic strokes, and most parts of the image are not blank. The key differences between a sketch and an image are shown in Figure 4.1.

Pixel-based generation methods such as regular GANs[3] and regular VAEs[2] generate images by assigning each of the pixels with its value, whereas stroke-based methods such as Sketch-RNN[27] generate sketches by outputting strokes. Pixel-based generation is frequently used to generate realistic color images such as photos, where the boundaries between subjects are blurry and there are almost no blank areas. However, for sketches, generating on the pixel level may cause reduced fidelity due to the noise that appears in blank areas. Besides, stroke-based generation also has the ability to scale on dimension. Thus, there is no necessity to re-train the model to generate larger sketches of the same class. The comparison of pixel-based generation and stroke-based generation is summarized in Table 4.1.

Sketch generation usually provide more information of generation process. For humans, stroke-based generation could be more intuitive, because pixel is more about a computer vision concept, where images need to be represented as 2-dimensional grid data. However, from machine learning perspective, generation of stroke usually requires stroke data. Unfortunately, many sketches, such as old engineering sketches, are saved as pixel images[45, 46]. This situation creates a major motivation for using reinforcement learning to generate sketches instead of VAE and GANs. One property of reinforcement learning is that an agent does not need to be directly trained based data. Instead, reinforcement learning is trained on environment, which provide the possibility of generate data that is different from the original data.

Table 4.1: Comparison between pixel generation and stroke generation

	Color Image	Noisy Blank Area	Dimension Scaling
Pixel Generation	Yes	No	No
Stroke Generation	No	Yes	Yes

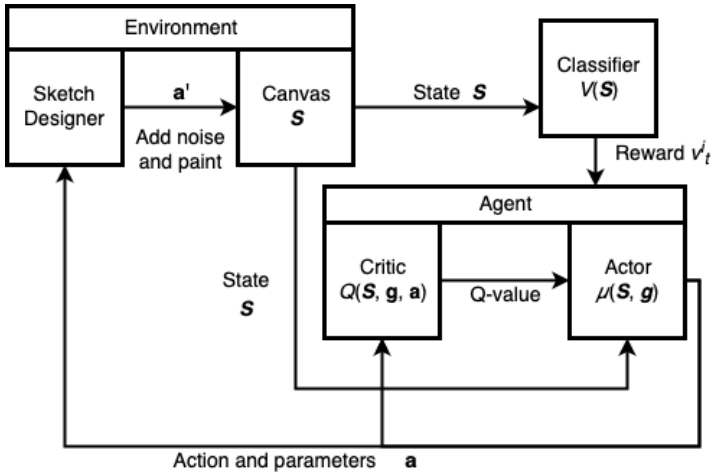


Figure 4.2: Flow diagram of the architecture

4.2 Model Architecture

In this section, the basic architecture of the agent and training techniques will be introduced. The process as a whole is illustrated in Figure 4.2. The architecture contains three major components: the environment, the classifier and the agent.

4.2.1 The Environment

Reinforcement learning relies on the environment to train, which can be modeled as a Markov Decision Process (MDP). The environment of the sketching agent consists of two components: the canvas and the sketch designer.

The Canvas.

The canvas is used to save and output the current state $\boldsymbol{S}$. Let the canvas be a $N \times N$ binary matrix $\boldsymbol{S}$ that contains only zeros and ones. The dimension of the $\boldsymbol{S}$ is matching the dimension of the dataset introduced in the case study, so that the classifier, denoted as V, that is trained based on the dataset can be used to evaluate the current state $\boldsymbol{S}_t$, where t denotes the time step. Since this is a stroke-based method, the dimension can be extended with no limitation after the agent is trained. Also, since the interested sketches are images with binary pixel values, so that the matrix $\boldsymbol{S}$ is also binary.

Table 4.2: The neural network classifier architecture

Type	Unit	Filter Size	Activation
Conv2d	32	4x4	ReLu
MaxPool	N/A	2x2	None
Conv2d	64	2x2	ReLu
FC	512	N/A	ReLu
FC	256	N/A	ReLu
FC	17	N/A	SoftMax

The Sketch Designer.

The sketch designer is another component of the environment. The major purpose of the sketch designer is to make the action noisy. The intuition behind this is that the environment for reinforcement learning is assumed to be a (MDP), which is a stochastic process. Methods in [16, 1, 47] show examples of playing games, whose environments are not predictable. In the sketching scenario, there will be no randomness in the environment, as the starting state $\boldsymbol{S}$ will always be a blank canvas, and the canvas varies only depending on the agent. Essentially, if the environment is not stochastic, the agent will generate the exactly same image over and over, which is against the purpose of generation.

$$\begin{aligned} x &= x + \text{Normal}(0,\, N/9) \\ y &= y + \text{Normal}(0,\, N/9) \\ c &= c + \text{Normal}(0,\, N/27) \end{aligned} \tag{4.1}$$

The Sketch Designer adds noise to the action $\mathbf{a}$ by making parameters noisy. The noisy action $\mathbf{a}'$ ensures that the stroke placed on the canvas is not identical but still similar to the one without the noise. Such kind of noise is very similar to human error, which is assumed to be normally distributed. Considering a person writing the alphabet repeatedly, none of the hand-written alphabet will be exactly the same. Shown in Equation 4.1, for the canvas size of $N \times N$, the coordinates x, y and the curvature parameter for a curve c is noised by Gaussian Distribution. The variance of the distribution is decided from multiple experiments. The detail of actions and parameters will be explained in later sections.

4.2.2 The Classifier

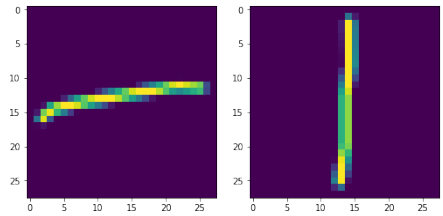
Let the neural network classifier be $V(\boldsymbol{S})$, which is pre-trained based on k classes. The architecture of the classifier is shown in Table 4.2. The classifier is a Convolutional Neural Network (CNN) classifier, which can extract features more efficiently than a fully-connected network. Note that there is only one pooling layer so that enough information is preserved. The output layer of the neural network contains k units, as there are k classes. For this study, the classifier is trained on 17 classes, so $k = 17$. The number of classes trained on the classifier is decided by

experiments. The output layer of the classifier has a softmax activation function. The softmax score for each class at time step t, $v_t^i = V(\boldsymbol{S}_t)$, where $i \in [1, k]$, representing the class, is used to reward the agent. Shown in Figure 4.4, the rewarding process is explained in the next subsection. The training data should be pixel-based images that contain only spacial features.

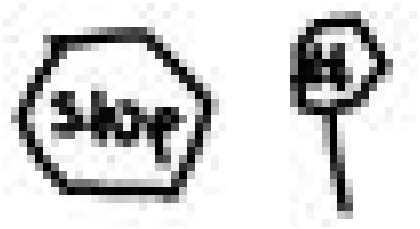
Shown in Figure 4.2, the classifier $V(\boldsymbol{S})$ is the most important component in this architecture, as it is playing the role of a judge for the agent. Unlike some real-world environment such as a game, there is no clear criteria for a good sketch or a bad sketch. There are two challenges in training the classifier. One challenge is that this pre-trained classifier should not only provide correct feedback of the finished sketch, but also provide a partially doodled sketch. The classifier also has to be very consistent on its evaluation.

Another challenge here is that human sketches normally contain large spacial variation, but relatively smaller temporal variation. Figure 4.3 shows examples of spacial variation. Most people sketch something by putting down strokes in a common order and relative position, but with a large variation in terms of size, orientation, and completeness. For example, in Figure 4.3, lines can be oriented in a different way, and the stop sign can either contain the rod support or not. In order to train the classifier, the sketches have to be transferred into regular black-and-white images. Then the neural network will be trained on the images with a large spacial variation, causing difficulties in training.

In terms of the training of the classifier, these two challenges are against each other. The number of classes should be maximized to ensure the correctness of the feedback, while the number of classes should be minimized to reduce the variation. Therefore, after experiments, 17 classes are picked here.



(a) Line 1 (b) Line 2



(c) Stop sign 1 (d) Stop sign 2

Figure 4.3: Example of large spacial variations of sketches based on stroke inputs

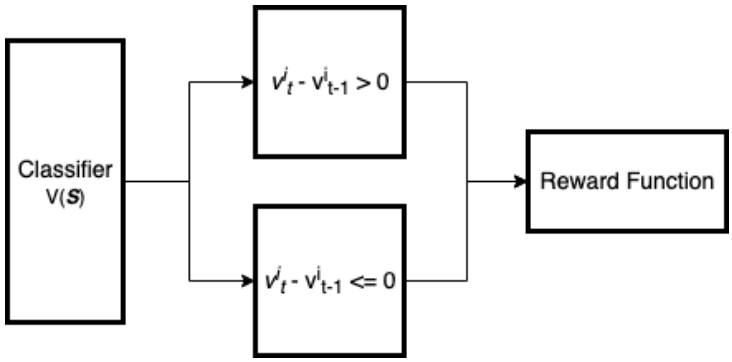


Figure 4.4: PROCESS OF ASSIGNING REWARDS

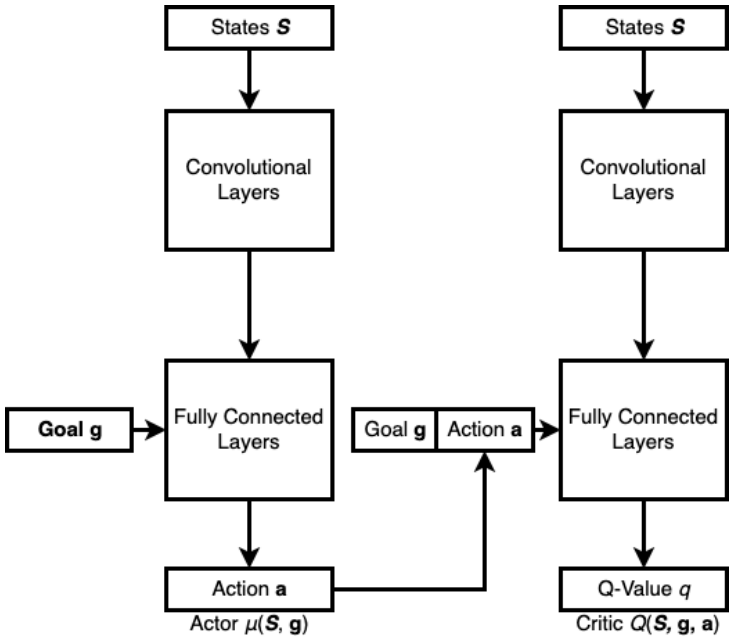


Figure 4.5: THE ARCHITECTURE OF ACTOR-CRITIC AGENT WITH SUB-GOAL

4.2.3 Reinforcement Learning Agent

A DDPG[8] actor-critic reinforcement learning architecture that is shown in Figure 4.5 is employed. This architecture enables a continuous action space so that the continuous parameters of the lines and curves can be generated. Both actor and critic are neural networks, whose architecture is shown in Table 4.3 and Table 4.4, respectively. The output of the actor is the action vector **a** with 14 elements, while the output of the critic is a scalar that is the Q-value. Note that both networks do not have an activation function at the output layer.

First, as shown in Figure 4.6, the action vector **a** contains:

1. Binary indicator variables that control which category (stop drawing, draw line, draw curve) of action should be taken. $(\delta_1, \delta_2, \delta_3)$

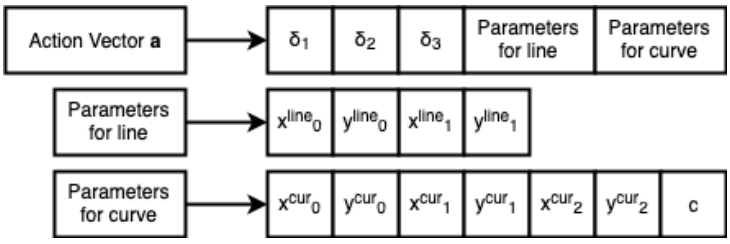


Figure 4.6: DECOMPOSITION OF ACTION VECTOR

Table 4.3: The neural network architecture for the actor

Type	Unit	Filter Size	Activation
Conv2d	32	7x7	ReLu
MaxPool	N/A	2x2	None
Conv2d	64	4x4	ReLu
FC	512	N/A	ReLu
FC	256	N/A	ReLu
FC	14	N/A	None

2. Coordinates of start point and end point of a line. $(x_0^{line}, y_0^{line}, x_1^{line}, y_1^{line})$
3. Coordinates of start point, mid point, and end point of a curve. $(x_0^{cur}, y_0^{cur}, x_1^{cur}, y_1^{cur}, x_2^{cur}, y_2^{cur})$
4. Curvature parameter for the curve. c

The environment starts with a blank canvas $\boldsymbol{S} = \left\{0\right\}$, and the actor will output actions $\mathbf{a}$ to sketch. Both networks take the canvas image matrix as the state $\boldsymbol{S}$.

Actor And Critic Networks

The actor network $\mu(\boldsymbol{S}, \mathbf{g})$ outputs an action vector that is 14 in length with 3 action indicators and 11 parameters of the actions, as shown in Figure 4.6. The indicator δ_j tells the environment which action to take (stop drawing, line, curve) and the rest of the 11 parameters are used to define a line and curve, as mentioned previously.

The agent shown at the bottom right part of Figure 4.2 consists of both actor network and critic network. The neural network structures for actor and critic are demonstrated in Figure 4.5, and the details of the layers of actor and critic are shown in Table 4.3 and Table 4.4, respectively. Note that both of the neural networks use convolutional layers before fully-connected layers to ensure the features are extracted correctly from the canvas.

The actor will take the state $\boldsymbol{S}$ as the input first. After the convolutional layers process the image input $\boldsymbol{S}$, the flattened tensor will be concatenated with the goal input $\mathbf{g}$, which is one-hot encoded; then the flattened tensor can be put into fully-connect layers. The fully-connected layer is used to interpret the relationship between state $\boldsymbol{S}$ and sub-goals, and reshape the output tensor to a desired shape. Then the action output $\mathbf{a}$ will be sent to the environment and the critic network. Note that there is no activation function for the last layer of the network, because an

Table 4.4: The neural network architecture for the critic

Type	Unit	Filter Size	Activation
Conv2d	32	7x7	ReLu
MaxPool	N/A	2x2	None
Conv2d	64	4x4	ReLu
FC	512	N/A	ReLu
FC	256	N/A	ReLu
FC	1	N/A	None

activation function for action output can cause saturation, as the activation functions usually have bounds. The parameters in Table 4.3 and Table 4.4 are tuned while conducting the experiments.

Also shown in Figure 4.5, the critic network Q takes image input $\boldsymbol{S}$ as the first step as well; then the flattened tensor will be concatenated with the goal $\mathbf{g}$ and the action $\mathbf{a}$ from the actor network. Note that the fully-connected layers in the critic network interpret the relationship between action, goal, and state. The output of the critic network is a scalar, which is the Q-value of an action $\mathbf{a}$ under an image input $\boldsymbol{S}$. Since the Q-value is the cumulative reward, the output layer for critic does not contain an activation function as well. The relationships described above are shown in Equation 4.2,4.3,4.4:

$$\mathbf{a} = \mu(\boldsymbol{S}, \mathbf{g}) \tag{4.2}$$

$$q = Q(\boldsymbol{S}, \mathbf{g}, \mathbf{a}) \tag{4.3}$$

$$\mathbf{v} = V(\boldsymbol{S}) \tag{4.4}$$

In Equation 4.2,4.3,4.4, $\mathbf{a}$ is the action vector, q is the Q-value, $\boldsymbol{S}$ is the state/canvas, $\mathbf{v}$ is a vector of softmax scores of different classes, g is the goal, Q is the critic network, μ is the actor network, and V is the classifier.

To summarize, both critic and actor have the same network structures except the output layer, so that the abilities of interpreting the states and goals are the same. The convolutional layers are used to extract spacial features, and the fully-connected layers are used to interpret the relation between goals, actions and states.

Training Process.

One challenge of training an actor network is that it is not possible to form a loss function without the output of a critic network for an actor network to update the parameters of an actor network. The authors employed the method introduced in DDPG [9]. Essentially, the update for an actor network will be conducted by applying gradient ascent using:

$$\bigtriangledown_{\theta_\mu} \mu(\boldsymbol{S}) = \bigtriangledown_{\mathbf{a}} Q(\boldsymbol{S}, \mathbf{a}|\theta_Q) \bigtriangledown_{\theta_\mu} \mu(\boldsymbol{S}|\theta_\mu) \tag{4.5}$$

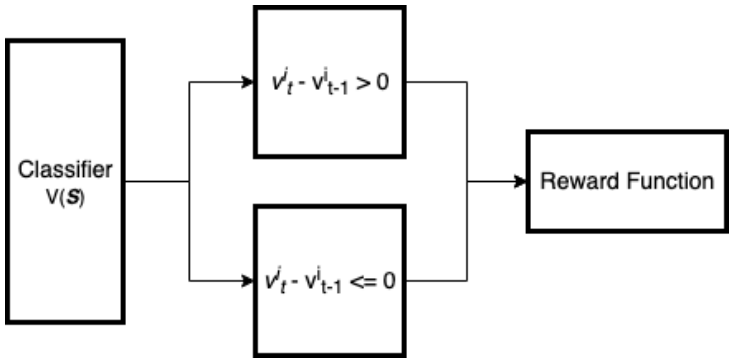


Figure 4.7: Process of assigning rewards

In Equation 4.5, $\boldsymbol{S}$ is the state, $\mathbf{a}$ is the action vector, θ_μ represents the parameters for actor network, and θ_Q represents the parameters for critic network.

An important component of the training process in reinforcement learning is the exploration. The authors develop a way to enforce exploration. The decision to explore is based on ϵ-greedy algorithm[48], since there is no prior domain knowledge in the model. First, a category of the action (stop drawing, draw line, draw curve) will be randomly selected; then the parameters will be sampled from the uniform distribution. The reason for choosing uniform distribution is because the upper bound and lower bound for the parameters are fixed, and all the points on the canvas should have an even chance to be selected. Equation 4.6 shows the detail of the exploration process. The bounds of the coordinates on the canvas x, y are determined from the size of canvas, and the bound for curvature c is determined empirically.

$$\text{If explore: } \begin{cases} x, y \sim \text{Uniform}(0, 27) \\ \\ c \sim \text{Uniform}(0, 6) \end{cases} \tag{4.6}$$

For critic network $Q(\boldsymbol{S}, \mathbf{g}, \mathbf{a})$, the loss function can be formulated using Bellman's equation with the reward. The challenge here is the assignment of the reward. Given softmax score of the goal class v^i_t, the reward r can be assigned. The authors develop the assignment of the reward that is shown in Equation 4.7. The whole process of assigning reward is shown in Figure 4.7.

$$r_t \in \begin{cases} 1 + v^i_t - v^i_{t-1} & \text{if } v^i_t - v^i_{t-1} > 0 \\ \\ -5 & \text{otherwise} \end{cases} \tag{4.7}$$

The final step of updating the networks is to use learning rate adaptive gradient descent methods such as ADAM[49] and RSMProp[50].

Stabilization.

The authors employ the stabilization techniques introduced in DQN, target networks and experience reply[1]. For DDPG, both actor and critic networks should have a separated target

network.

Another unstable issue is that the action output can be outside of the bounds, which will interrupt the training process. The authors employed an inverting gradient proposed by Hausknecht and Stone [47] to bound the parameters. Essentially, given a parameter p and the bounds $[p_{\min}, p_{\max}]$, the gradient of p, $\bigtriangledown_p$, the gradients are:

$$\bigtriangledown_p = \bigtriangledown_p * \begin{cases} (p_{\max} - p)/(p_{\max} - p_{\min}) & \text{if } \bigtriangledown_p \text{ suggest increase p} \\ (p - p_{\min})/(p_{\max} - p_{\min}) & \text{otherwise} \end{cases} \tag{4.8}$$

To summarize, in order to train a reinforcement learning agent to sketch, the action space has to be continuous and the environment has to be stochastic. The challenge is how the reward is evaluated for each action. If the reward can be properly assigned, a reinforcement learning algorithm can be conducted train the DDPG agent with all the stabilization techniques.

4.3 Experiment

In this section, an experiment is conducted to show the capability and limitation of the agent. The agent is compared with the Sketch-RNN[27] for benchmarking. Both the Sketch-RNN and the agent is trained on triangle class from QuickDraw dataset, which contains human sketches across hundreds of classes.

4.3.1 Results

The experiment is implemented by Python using TensorFlow as the automatic differentiation software. The dataset used to train classifier is Google Quick Draw, whose raw data contains the information of the trajectory of the strokes with the time stamp. The processed data used for training contains pixel-based 28x28 images. During the training, the batch size is 64 and the learning rate for ADAM [49] optimizer is 0.0001. The experiments are conducted on an NVIDIA GTX 1070 GPU on Ubuntu 16.04.

Figure 4.8: Random triangles generated by the agent

Figure B.1 and Figure B.2 in the Appendix show the statistics of the training process, and Figure 4.8 shows the samples of generating triangles. The generations are not quite recognizable, but the generated images contain features of triangles. The statistics shows that the convergence of both Q-value and cost function is very early at approximately 4000 steps, even though the generation is not satisfying. This implies the improper reward assignment.

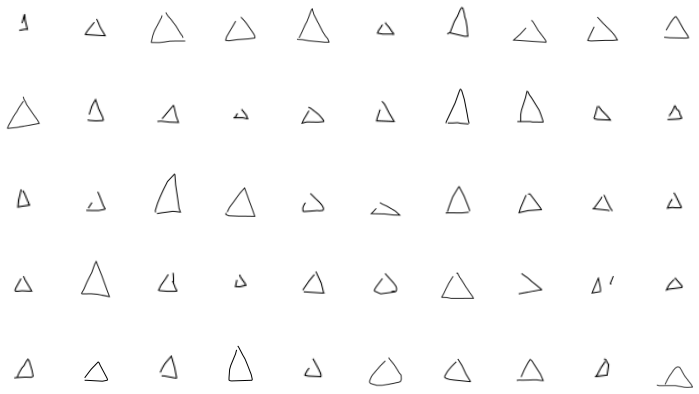


Figure 4.9: Random triangles generated by the Sketch-RNN

The Sketch-RNN is also trained on the same dataset. The configuration mostly follows the recommended configuration[1] from the authors, but with 512 units for decoder and 256 units for encoder. The Sketch-RNN is unable to converge at 30K steps. Figure B.3 shows the trend that the expected convergence of Sketch-RNN is most likely more than 45K steps. However, the generations are impressive and recognizable, with very limited variation. Figure 4.9 shows the generation of triangles by Sketch-RNN.

4.3.2 Analysis

Generally speaking, the generated samples are not quite satisfying. This section lists possible reasons that may cause poor results.

4.3.2.1 Reason 1: Classifier is Incapable

The fact that the cumulative reward (Q-Value) is converging even though the generation is poor quality implies problems in the process of assigning reward. Also, the increasing variation in the Q-Value as the time step increases implies an unstable training process.

It is clear that there are issues regarding the reward, but the modifications of reward function have very limited improvement. Taking a step back, the problem may occur before the reward function. As shown in Figure 4.4, the issues may happen in the classifier, since the modifications on the step after (changing reward function) seem not effective. The assumption of this reward assignment to be correct is that the classifier can evaluate if a stroke is good or poor accurately.

[1] https://github.com/tensorflow/magenta/blob/master/magenta/models/sketch_rnn

In the experiment, the agent appears to have learned incorrectly. Therefore, it is most likely that the classifier is unable to give a good evaluation of each stroke. Further improvements should be conducted on the classifier, such as replacing it with other kinds of methods of evaluation.

Another issue is that the growth of the Q-value is not as stable as expected. The sketching environment should be more stable than most of the tasks like playing video games, since there is very limited randomness. Therefore, the inconsistency of the evaluation from the classifier is also the reason for the unstable Q-values.

The issue with the classifier also causes the low fidelity of the generated images, shown in Figure 4.8. The only way of punishing strokes that add random noises is by decreasing the score, as shown in Figure 4.4 and Equation 4.7, and the classifier is not capable to do this.

4.3.2.2 Reason 2: Classifier is too Capable

The architecture introduced is very similar to the Generative Adversarial Nets (GAN) [3]. The classifier is essentially a discriminator in GAN, and therefore, the training process proposed could be suffered from the unbalanced capabilities of the classifier and the agent. In other words, the classifier is so capable that the agent cannot learn from the feedback of the classifier.

In order to solve the problem, one way to overcome this problem is to train both the classifier and the generator together. Specifically, the agent can be trained on policy gradients as normal, and the classifier can be trained using the discriminator loss in GAN.

By training the classifier and the agent together, they can learn from each other better. The agent will learn how the classifier determines whether a generated sample is similar to a real sample, and the classifier will learn less trivial features.

4.3.2.3 Reason 3: Inconsistent Paint Engine

One of the common problems of stroke based generation will be the difference between paint engines. Occasionally, it can be very hard to ensure that one paint engine could reconstruct a sketch that is created by another paint engine.

The experiment above is not using a very powerful paint engine. In fact, the paint engine used here is *scikit-image*, which is an open-source image library that can only draw simple strokes. Therefore, some of the features in the original data could be never represented correctly.

4.4 Conclusion

This chapter introduced an architecture that can produce stroke-based sketches using pixel-based training data. The architecture is based on DDPG reinforcement learning agent, which can operate in an environment with continuous state space and continuous action space.

The results from the case study are very preliminary. Compare to the state of the art, the major problem is the high variance in the generation. Future works can focus on improving the classifier structure, change different training strategies, and improving the paint engine capability.

Appendix A

Additional Results for Reinforcement Learning in Dosing Control

A.1 Additional Result in Attributes Analysis

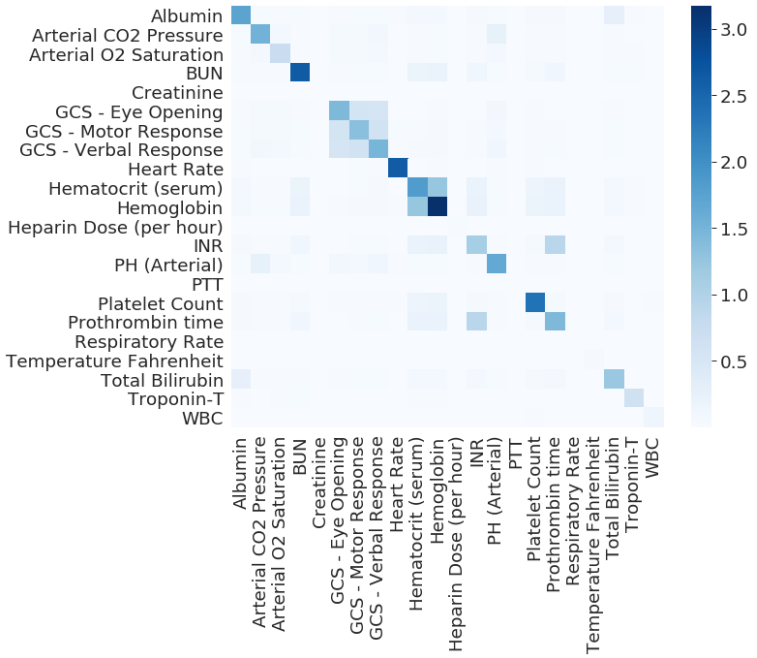


Figure A.1: Heatmap of average inter-attributes mutual information matrix.

Figure A.1 shows that most of the attributes are fairly independent. Only GCS families, INR-Prothrombin time, and Hematocrit-Hemoglobin correlated each other.

Table A.1 shows that most of the attributes has no delayed effect on injections.

Table A.1: Optimal time delay of each attributes based on mutual information with bolus

Attributes	Optimal time delay (hours)	Basal MI at optimal
Albumin	1	0.05306
Arterial CO2 Pressure	1	0.14231
Arterial O2 Saturation	4	0.06362
BUN	1	0.22495
Creatinine	1	0.14807
GCS - Eye Opening	7	0.09241
GCS - Motor Response	1	0.07104
GCS - Verbal Response	1	0.08136
Heart Rate	4	0.48786
Hematocrit (serum)	1	0.15731
Hemoglobin	1	0.18612
Heparin Dose (per hour)	1	0.00407
INR	1	0.13351
PH (Arterial)	1	0.14861
PTT	1	0.38117
Platelet Count	1	0.24784
Prothrombin time	1	0.16116
Respiratory Rate	1	0.41974
Temperature Fahrenheit	2	0.44538
Total Bilirubin	1	0.06237
Troponin-T	1	0.10896
WBC	1	0.21558

A.2 Additional Results for Agent Training

Figure A.2 and Figure A.3 shows how Q-value changes as the training proceed. Both curves are smoothed using exponential smoothing with $\alpha = 0.1$ Since we do not have environment executing, the experiment will not refresh. As a result, the graph is not very meaningful, but still worth to look at. The slight trend of increasing implying the agent is learning.

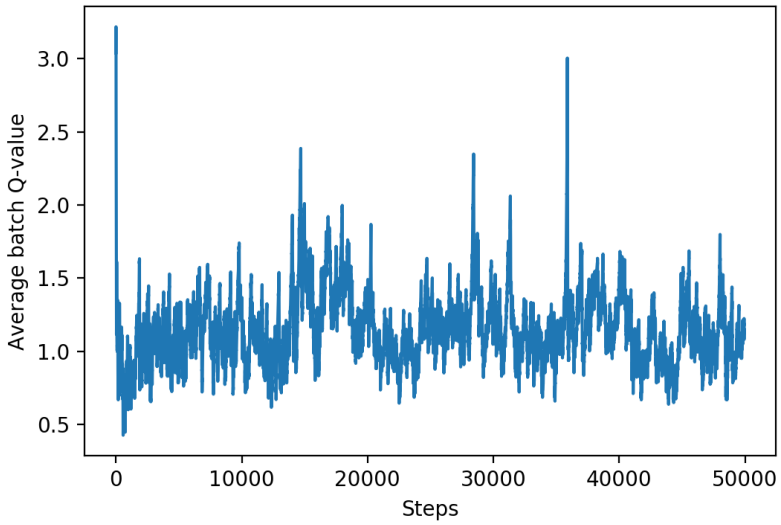


Figure A.2: Average batch Q-value VS. Training Steps for Experiment 1

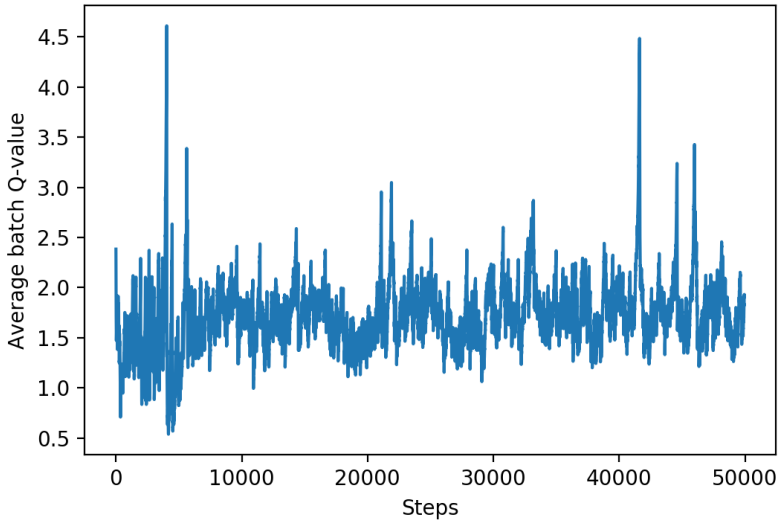


Figure A.3: Average batch Q-value VS. Training Steps for Experiment 2

Appendix B

Training Process for Sketch Generation Using Reinforcement Learning

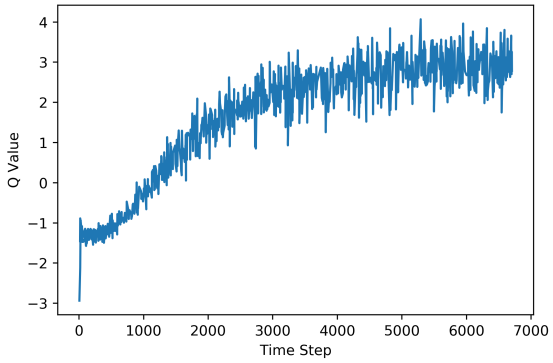


Figure B.1: Q-value VS time step

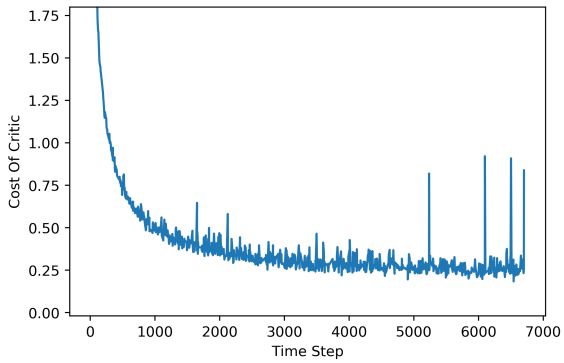


Figure B.2: Loss function of critic VS time step

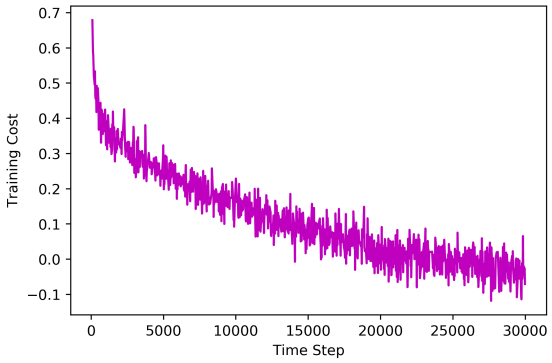


Figure B.3: Loss function of Sketch-RNN VS time step

Bibliography

[1] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015) "Human-level control through deep reinforcement learning," *Nature*, **518**(7540), p. 529.

[2] Kingma, D. P. and M. Welling (2013) "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*.

[3] Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014) "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680.

[4] White, D. J. (1969) *Dynamic programming*, vol. 1, Oliver & Boyd Edinburgh.

[5] van Otterlo, M. and M. Wiering (2012) *Reinforcement Learning and Markov Decision Processes*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 3–42.
URL `https://doi.org/10.1007/978-3-642-27645-3_1`

[6] Watkins, C. J. and P. Dayan (1992) "Q-learning," *Machine learning*, **8**(3-4), pp. 279–292.

[7] Rummery, G. A. and M. Niranjan (1994) *On-line Q-learning using connectionist systems*, vol. 37, University of Cambridge, Department of Engineering Cambridge, England.

[8] Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2015) "Continuous control with deep reinforcement learning," *arXiv e-prints*, arXiv:1509.02971, `1509.02971`.

[9] Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller (2014) "Deterministic policy gradient algorithms," in *ICML*.

[10] Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015) "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897.

[11] Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017) "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*.

[12] Kulkarni, T. D., K. Narasimhan, A. Saeedi, and J. Tenenbaum (2016) "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, pp. 3675–3683.

[13] Peng, X. B., G. Berseth, K. Yin, and M. Van De Panne (2017) "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, **36**(4), p. 41.

[14] Vezhnevets, A. S., S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu (2017) "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, pp. 3540–3549.

[15] Dayan, P. and G. E. Hinton (1993) "Feudal reinforcement learning," in *Advances in neural information processing systems*, pp. 271–278.

[16] Zhang, Z., H. Li, L. Zhang, T. Zheng, T. Zhang, X. Hao, X. Chen, M. Chen, F. Xiao, and W. Zhou (2019) "Hierarchical Reinforcement Learning for Multi-agent MOBA Game," *arXiv preprint arXiv:1901.08004*.

[17] Lee, J., J. Hwangbo, and M. Hutter (2019) "Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning," *arXiv preprint arXiv:1901.07517*.

[18] Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016) "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937.

[19] Hirsh, J. (1991) "Heparin," *New England Journal of Medicine*, **324**(22), pp. 1565–1574.

[20] McLean, J. (1916) "The thromboplastic action of cephalin," *American Journal of Physiology-Legacy Content*, **41**(2), pp. 250–257.

[21] Rosenberg, R. (1994) "The heparin-antithrombin system: a natural anticoagulant mechanism," *Hemostasis and Thrombosis, Basic Principles and Clinical Practice*, pp. 837–860.

[22] Reilly, B. M., R. Raschke, S. Srinivas, and T. Nieman (1993) "Intravenous heparin dosing," *Journal of general internal medicine*, **8**(10), pp. 536–542.

[23] Raschke, R. A., B. M. Reilly, J. R. Guidry, J. R. Fontana, and S. Srinivas (1993) "The weight-based heparin dosing nomogram compared with a standard care nomogram: a randomized controlled trial," *Annals of internal medicine*, **119**(9), pp. 874–881.

[24] Dering, M. L. and C. S. Tucker (2017) "Generative adversarial networks for increasing the veracity of big data," in *2017 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 2595–2602.

[25] Burnap, A., Y. Liu, Y. Pan, H. Lee, R. Gonzalez, and P. Y. Papalambros (2016) "Estimating and exploring the product form design space using deep generative models," in *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, pp. V02AT03A013–V02AT03A013.

[26] Sønderby, C. K., T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther (2016) "Ladder variational autoencoders," in *Advances in neural information processing systems*, pp. 3738–3746.

[27] Ha, D. and D. Eck (2017) "A Neural Representation of Sketch Drawings," *CoRR*, **abs/1704.03477**, 1704.03477.
URL http://arxiv.org/abs/1704.03477

[28] Zhang, L., L. Lin, X. Wu, S. Ding, and L. Zhang (2015) "End-to-end photo-sketch generation via fully convolutional representation learning," in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, ACM, pp. 627–634.

[29] Kang, H. W., W. He, C. K. Chui, and U. K. Chakraborty (2005) "Interactive sketch generation," *The Visual Computer*, **21**(8-10), pp. 821–830.

[30] Wen, F., Q. Luan, L. Liang, Y.-Q. Xu, and H.-Y. Shum (2006) "Color sketch generation," in *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM, pp. 47–54.

[31] Zhou, T., C. Fang, Z. Wang, J. Yang, B. Kim, Z. Chen, J. Brandt, and D. Terzopoulos (2018) "Learning to Sketch with Deep Q Networks and Demonstrated Strokes," *arXiv preprint arXiv:1810.05977*.

[32] Arjovsky, M., S. Chintala, and L. Bottou (2017) "Wasserstein gan," *arXiv preprint arXiv:1701.07875*.

[33] Radford, A., L. Metz, and S. Chintala (2015) "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*.

[34] Karras, T., S. Laine, and T. Aila (2019) "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410.

[35] Zhang, H., I. Goodfellow, D. Metaxas, and A. Odena (2018) "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*.

[36] Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark (2016) "MIMIC-III, a freely accessible critical care database," *Scientific data*, **3**, p. 160035.

[37] Gravlee, G. P., W. S. Haddon, H. K. Rothberger, S. A. Mills, A. T. Rogers, V. E. Bean, D. H. Buss, D. S. Prough, and A. R. Cordell (1990) "Heparin dosing and monitoring for cardiopulmonary bypass: a comparison of techniques with measurement of subclinical plasma coagulation," *The Journal of thoracic and cardiovascular surgery*, **99**(3), pp. 518–527.

[38] Despotis, G. J., J. H. Joist, C. W. Hogue Jr, A. Alsoufiev, D. Joiner-Maier, S. A. Santoro, E. Spitznagel, J. I. Weitz, and L. T. Goodnough (1996) "More effective suppression of hemostatic system activation in patients undergoing cardiac surgery by heparin dosing based on heparin blood concentrations rather than ACT," *Thrombosis and haemostasis*, **76**(06), pp. 0902–0908.

[39] Hurewitz, A. N., S. U. Khan, M. L. Groth, P. A. Patrick, and D. A. Brand (2011) "Dosing of unfractionated heparin in obese patients with venous thromboembolism," *Journal of general internal medicine*, **26**(5), pp. 487–491.

[40] Smythe, M. A., J. Priziola, P. P. Dobesh, D. Wirth, A. Cuker, and A. K. Wittkowsky (2016) "Guidance for the practical management of the heparin anticoagulants in the treatment of venous thromboembolism," *Journal of thrombosis and thrombolysis*, **41**(1), pp. 165–186.

[41] Chen, Y., F. Leonelli, and H. Yang (2016) "Heterogeneous sensing and predictive modeling of postoperative outcomes," .

[42] Ghassemi, M. M., S. E. Richter, I. M. Eche, T. W. Chen, J. Danziger, and L. A. Celi (2014) "A data-driven approach to optimized medication dosing: a focus on heparin," *Intensive care medicine*, **40**(9), pp. 1332–1339.

[43] Nemati, S., M. M. Ghassemi, and G. D. Clifford (2016) "Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, pp. 2978–2981.

[44] Lin, R., M. D. Stanley, M. M. Ghassemi, and S. Nemati (2018) "A Deep Deterministic Policy Gradient Approach to Medication Dosing and Surveillance in the ICU," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, pp. 4927–4931.

[45] Shah, J. J., N. Vargas-Hernandez, J. D. Summers, and S. Kulkarni (2001) "Collaborative Sketching (C-Sketch)—An idea generation technique for engineering design," *The Journal of Creative Behavior*, **35**(3), pp. 168–198.

[46] Yang, M. C. (2009) "Observations on concept generation and sketching in engineering design," *Research in Engineering Design*, **20**(1), pp. 1–11.

[47] Hausknecht, M. and P. Stone (2015) "Deep Reinforcement Learning in Parameterized Action Space," *arXiv e-prints*, arXiv:1511.04143, `1511.04143`.

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

[49] Kingma, D. P. and J. Ba (2014) "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*.

[50] Tieleman, T. and G. Hinton (2012) "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, **4**(2), pp. 26–31.