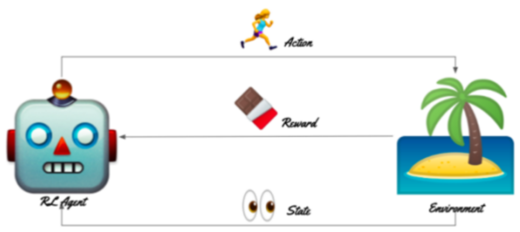
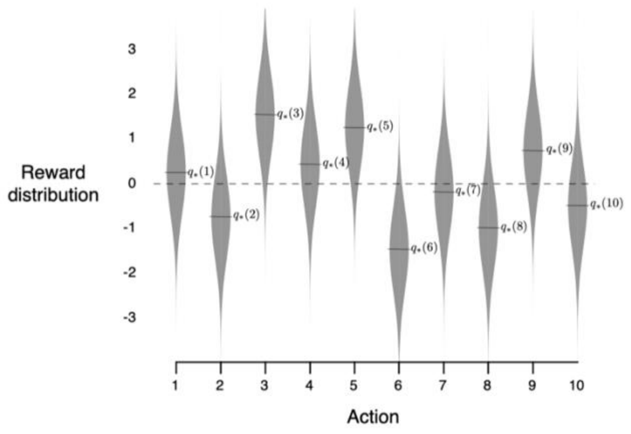
Final Written Assignment

Reinforcement Learning (RL) is one of the most advanced algorithms that people use in data analytics, data science, machine learning, and artificial intelligence. As technology has been developed drastically, RL is mentioned everywhere in the news, articles, and literature. I believe it is important to understand the fundamentals of RL so that I can get an idea of new variations of RL when they come out; more importantly, I can evaluate the algorithms and see if they can last over time. My career aspiration is purely based on my interest in the financial and technology industry as well as medical research. I would like to be a data scientist to help organizations to make high-impact decisions by using data-driven approaches. I strongly believe RL can be applied to the areas that I'm interested in.



The Bandit Algorithm

The K-armed Bandit algorithm gives me a nice introduction to the RL agent-environment interaction. The agent takes action based on the policy π, the action has an impact on the environment. The environment emits reward to the agent, the agent uses reward to update the action-value estimates q. Although the setup is relatively simple, using the correct algorithm can guarantee these action-value estimates q converge to the true action-values q^*.



I learned three policies to help our agent to take action, including Epsilon-Greedy Policy, Upper-Confidence-Bound Policy (UCB), and SoftMax

Policy. Understanding exploration vs exploitation is the key to get good results. Early exploration is critical for RL agents; once the agent gets enough information about each action (environment), it can greedily choose the best action available. Epsilon is the tuning parameter that controls the exploration (learning rate) in Epsilon-Greedy Policy. Similarly, UCB helps us explore actions that we have not chosen in a while. SoftMax function provides the probability of taking action at each timestep. It is powerful because the probability $\pi(a)$ is not affected by adding the same constant value to all action preferences.

Also, I learned three methods to update the action-value estimates q, including Sample Average Method, Recency Weighted Average Method, and Gradient Bandit. Sample Average updates the action-value estimates q by using the frequency of choosing action a as the learning rate, whereas Recency Weighted Average updates the action-value estimates q by using lagging indictor α as the learning rate. Large α allows quick adaptation, it is highly dependent on things that happened recently, small α deals with violent spikes that you don't want to react. Gradient Bandit allows the action you choose to have an impact on other actions.

I strongly believe the most complex model does not necessarily mean the best performance, a simpler

A Multi-Armed Bandit Framework for Recommendations at Netflix

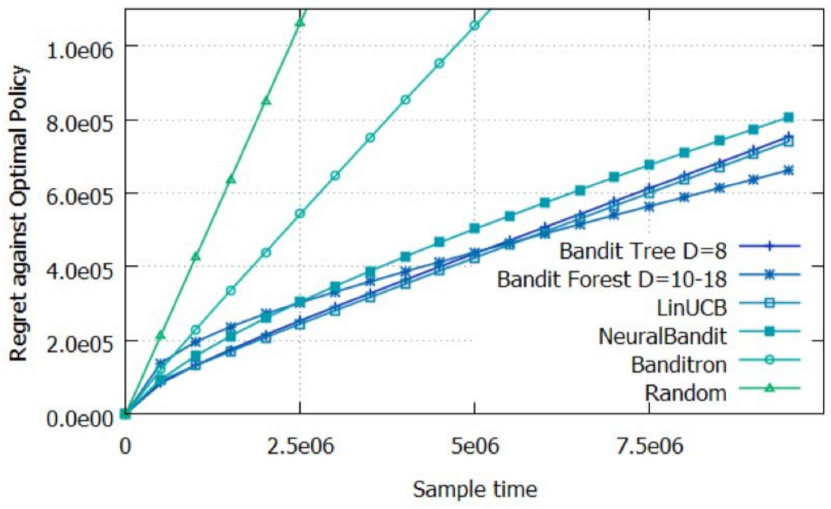
Jaya Kawale & Fernando Amat

PRS Workshop, June 2018

NETFLIX

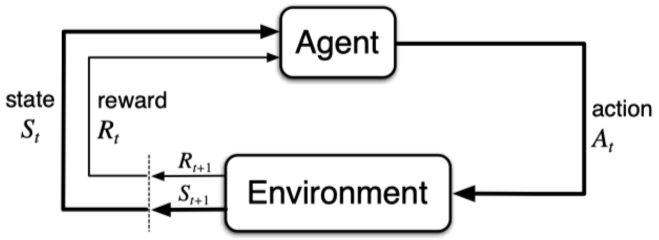
model like the Bandit Algorithm is more efficient and practical. Netflix is my favourite media consumption service. I used to think about the way they implement their recommendation system. Surprisingly, they use the Bandit Algorithm, and this is such an elegant way to implement the application. The challenge for the Netflix personalized recommendation system is finding the right titles (movies and TV shows); more importantly, displaying the right artwork/imagery that captures something compelling to the users. To customize the recommendation feature for every user, they should know the impact on changing artwork after the user watches a particular movie or TV show and have a diverse and comprehensive pool of artwork for each title. But traditional machine learning algorithms that run on batch data are impossible to handle a peak of over 20 million requests per second with low latency (Chandrashekar, 2018). So, they use contextual bandits, an online machine learning framework, to solve the issue. The contextual bandit algorithm is an extension of the k-armed bandit approach. It factors users' environment and context, as contexts change, the algorithm learns to adapt to the changes and find the maximum reward (Yang, 2019). For example, when you use a mobile device to log in to your Netflix account, you may find the artwork in your recommendation board is different than the one you use on your smart TV.

In addition, Feraud et al. (2016) has shown that the random forest algorithm can be implemented into the contextual bandits - Bandit Forest. It is well suited to deal with non-linear dependencies between contexts and rewards with low computational costs. The Bandit Forest clearly outperforms linear UCB, which is a strong baseline, and performs approximately the same as Neural Bandit. It is worth mentioning that its sample complexities have a logarithmic dependence on the number of contextual variables, which means that it can process a large number of contextual variables with a low impact on regret.

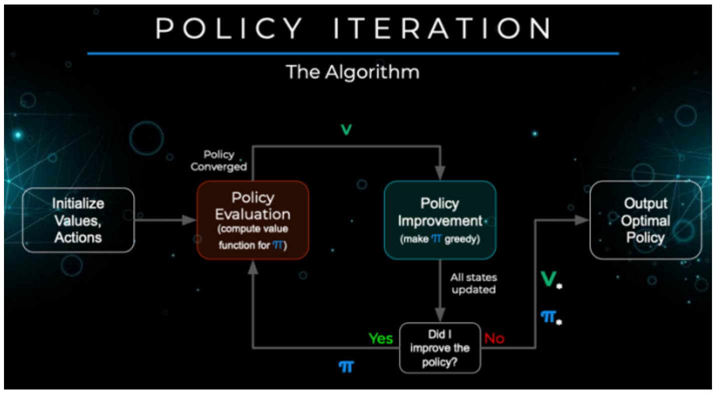


Optimality, Policy Improvement, Dynamic Programming

The finite Markov Decision Processes (MDPs) introduce the states into the agent-environment interface. In each discrete timestep, the agent takes action A_t based on the state S_t, then the environment emits reward $R_{t+1}$and transitions to a new state S_{t+1}. With the help of the Bellman



Equations, we are able to calculate state-value estimate $v_\pi(s)$ and action-value estimates $q_\pi(s, a)$. How can we improve our estimates? Simply "take the max": take the action that maximizes our expected


POLICY ITERATION
The Algorithm
Initialize Values, Actions
Policy Converged
Policy Evaluation (compute value function for π)
V
Policy Improvement (make π greedy)
All states updated
Did I improve the policy?
Yes
No
π
V*
π*
Output Optimal Policy

return. This is known as the Bellman Optimality Equations. Then the question is how we can improve our policy optimally. We can do it in two steps. The first step is policy evaluation (prediction), we fix the policy π and find state-value estimate v_π via many iterations until it converges ($v_o \rightarrow v_1 \rightarrow \cdots \rightarrow v_\pi$). The second step is policy improvement (control), we allow the agent to make greedy actions to see if the new policy π' is better than or equal to the old policy π in all states. We can actually merge these two steps into one policy iteration. MDP and π go in, π_* and v_* come out. Indeed, just like you said, "it is magic"! The idea of Dynamic Programming (DP) is when you break down a big problem into smaller problems and you solve those smaller problems optimally, those optimal solutions allow you to find the optimal solution for the big problem.

MDP and DP are widely used in the financial industry. They can be applied to optimize revenue and profit by providing an optimal pricing strategy. In the research from Krasheninnikova et al. (2019), their objective is to adjust renewal prices in the insurance industry. It is not a trivial problem because the organization needs to be aware of the trade-off between revenue and customer retention. If you increase the price, you will increase the revenue, but the customer may switch to another company. On the other hand, low renewal price means low revenue, there is a regret of revenue loss compared to what you could've made under the optimal pricing strategy. They decided to use RL agents to help to solve the problem. They define the reward as money that was earned from the renewal offer. The action space is defined as the percentage increase or decrease of the renewal offer. The state space is represented by a tuple, including revenue, retention, customer segment, and insurance price. There are a total of 16000 states/clusters in the paper. 16000 clusters are not ideal for supervised and unsupervised learning; however, the MDPs can handle them well. Since different type of customer has different behaviour of accepting offers, for instance, many rich people have a high tolerance of price increase while others may be very sensitive to price adjustment. Therefore, policy iteration and DP are critical to helping us find the optimal actions for each customer segment.

Monte Carlo Methods

Compare to supervised learning and unsupervised learning, DP is a great option to compute the value functions for a large state space, but it reaches the limitation when the environment is too complex or unknown. Monte Carlo methods are introduced to solve the issue. Monte Carlo methods don't assume complete knowledge of the environment; instead, the agents learn the environment from the actual experience by calculating the average sample returns. Since Monte Carlo (MC) agent improves iteratively through episodes, we assume the environment must be episodic, all episodes eventually terminate no matter what actions are chosen. To estimate state-value estimates $v_\pi(s)$ for policy evaluation (prediction), we can use the first-visit MC method or every-visit MC method. The first-visit MC method averages the return following the first visit to state s, whereas the every-visit MC method average the returns following all visits to state s. So far, we learned the on-policy approach for policy iteration, can we do better? Absolutely, the off-policy approach is more flexible, it enables learning from historical data, policies, agents, and much more. Off-policy has two separate policies, one for learning/exploring and one for improvement. The behavioural policy must ensure coverage that allows the agent to visit all states so that the target policy learns from it by acting optimally. Weighted importance sampling is applied to correct the difference between the target and behavioural policy. The benefit of the off-policy approach enables the agent to act fully greedily in the target policy, no longer need epsilon soft policy. Also, it allows transfer learning, we can apply certain skills in a new task.

Monte Carlo Methods are widely used in the stock market and portfolio management. MC agents can predict the future many times through episodes. We can feed historical data of the stock market to our agents so that we can make optimal decisions to maximize our profits. The best part of the off-policy approach is transfer learning. The agent can apply the model you have to better fit the new algorithm. For example, Geometric Brownian

Motion (GBM) is one of the most common models in finance, the MC agent can take advantage of the GBM model to learn the price fluctuation. GBM allows us to solve underlying stock price at time t (S_t) using initial stock price S_0, drift μ, volatility σ, and Brownian motion W_t.

$$GBM: dS_t = \mu S_t dt + \sigma S_t dW_t$$

$$Solution: S_t = S_0 e^{\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t}$$

Another example is in the options market, it is critical for our agent to learn the call and put option price via the famous Black Scholes model. Black Scholes model allows us to calculate the option price using the strike price K, the cumulative normal distribution Φ, drift μ and volatility σ coincide with the GBM model (Vittori et al., 2020).

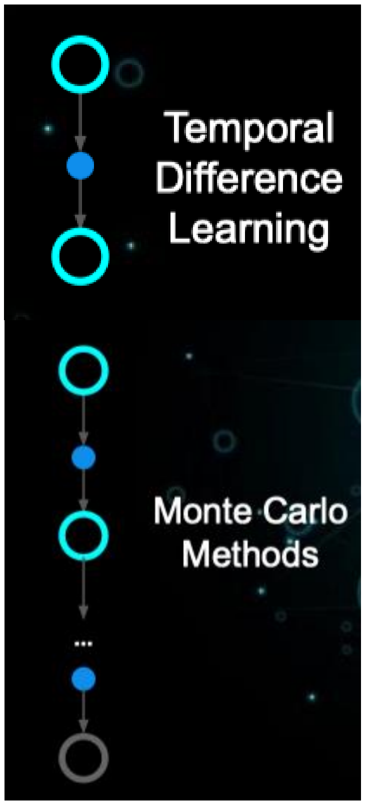
$$Call\ option: C_t = S_t \Phi(d_t) - K e^{\mu} \Phi(e_t)$$

$$Put\ option: P_t = K e^{\mu} \Phi(-e_t) - S_t \Phi(-d_t)$$

Vittori et al. (2020) explain the option hedging framework using the above two models. The action a_t is the current hedge portfolio, the state s_t consists of 4 items $(S_t, C_t, \frac{\partial C_t}{\partial S_t}, a_{t-1})$, the reward/investment return R is based on the state-action pair.

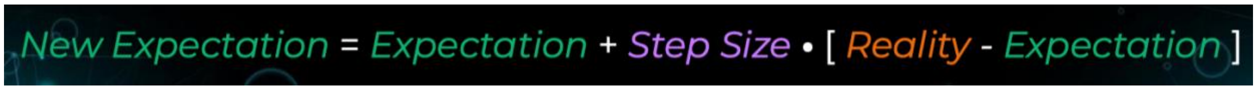
The off-policy approach gives our agent the learning ability by “looking over someone’s shoulder”. However, as powerful as off-policy learning is, it has drawbacks. Generally, it has a huge variance due to the difference between the likelihood of the action under the target policy and the behavioural policy. When you multiply the probabilities across the whole trajectory, it is almost guaranteed to have a variance exploding problem. Thus, we use MC as an on-policy method for many real-world applications. I realize that we should think twice before we apply different RL methods.

Temporal Difference Methods



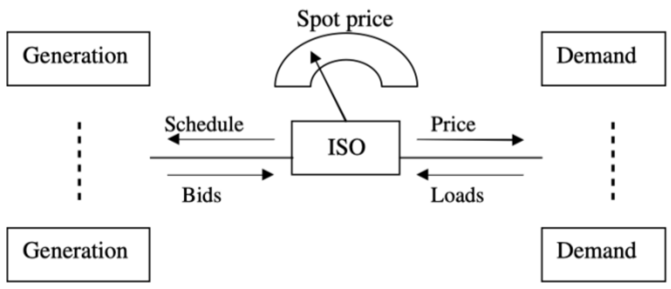
TD-learning is a combination of Monte Carlo (MC) and Dynamic Programming (DP). Just like MC methods, TD-learning agent learns directly from raw experience without knowing the environment. Just like DP, TD-learning agent updates estimates based on other learned estimates. The power of TD-learning is that it updates value estimates without waiting for the episode to end. TD-learning can learn bad things sooner. The analogy I like for TD-learning is that "you don't have to experience bad things, like jumping off a cliff, driving a car recklessly, doing bad drugs and have your entire life episode terminate, to learn." Compare to MC methods, TD-learning has low variance and some bias because it "learns a guess from a guess". Besides, TD-learning methods tend to be a lot faster and more efficient than the MC methods when the environment has very long episodes; TD-learning can handle continuing tasks while MC cannot. In certain applications, TD-learning is much preferred, like inventory management where you cannot fail - "you cannot back up death". Therefore, TD-learning gives us new opportunities to explore the more complex environment.

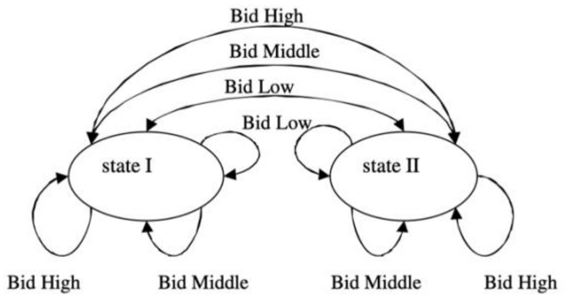
The methods we have seen so far all have the same form:


New Expectation = Expectation + Step Size • [Reality - Expectation]

The Temporal Difference (TD-learning) methods also share the exact form. There are 4 different methods covered in TD-learning, including TD(0), SARSA, Q-learning, and Expected SARSA. TD(0) is an on-policy prediction method that estimates state-value v_π under policy π. SARSA is an on-policy TD control method that estimates action-value q. Q-learning is an off-policy TD control that estimates action-value q_* and π_*. Expected SARSA is an on-policy or off-policy TD control method that estimates action-value q_* and π_*.

TD-learning is very useful in the electric power industry. According to the research from Naghibi-Sistani et al. (2006), choosing a suitable bidding strategy for trading electricity is a big issue in the real marketplace based on the limited information available to all participants. The Pool-Co model is one of the straightforward structures for implementation of competitive electric markets. Each generating company offers its bid and corresponding generation quantity to an independent system operator (ISO). The ISO computes the spot market prices and selects the lowest price until the forecasted demand is supplied.





The generation costs of any participating generator i belonging to each participant is

$$C_i = C(P_i) = a_i + b_i P_i + c_i P_i^2$$

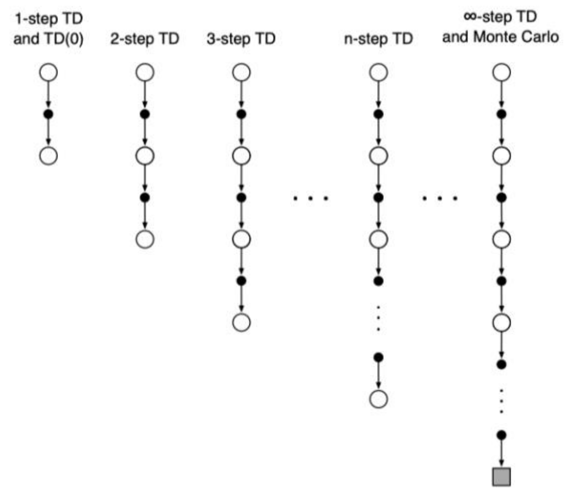
The participant's benefit can be calculated using the spot price ρ, the power transaction amount T_k.

$$G_k = \sum -\triangle\, C_i + \rho T_k$$

The Q-learning method is used in decision making for an agent in an electric power market. Since the electric power market is a multi-agent environment, the convergence of the Q-learning is not guaranteed. To solve the convergence issue, the authors modified the Boltzmann probability distribution by using a temperature parameter T that adjusts the randomness of the decisions. The action spaces (bidding strategies) are classified as high, low, and middle. Each participant has only two states, state I and II, which refer to low cost and high cost of production respectively, but the state is hidden from the competitors. The reward is the pay-off from the transaction of the competition. The main advantage of using Q-learning is that it enables participants to find optimal strategies using only a few information (e.g., spot price). With all the great benefits of Q-learning, we need to be aware of its large variance of the update towards the max of next actions experienced by the behavioural policy. When we are doing something that has real consequences in real-world applications, Expected SARSA is a more conservative approach to learn from its mistakes. We should always apply critical thinking skills to decide which algorithm to use.

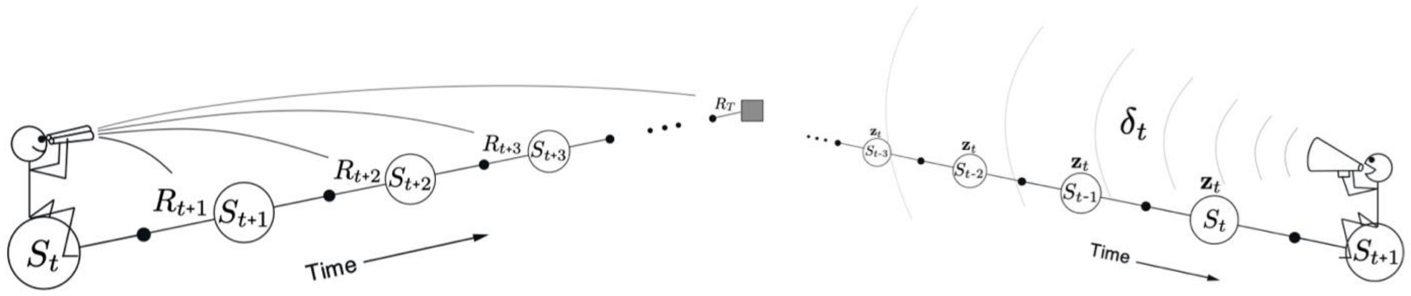
n-Step Methods, Planning, and Advanced Tabular Methods

Since one-step TD-learning has high bias and low variance, Monte Carlo is a bias-less method with high variance, we can find some methods in between with low bias and low variance. The n-step TD-learning methods are the optimal intermediate methods between the two extremes. When we increase the number of steps between each update, we get more accurate updates since we are collecting more pieces of reality before each update. The n-step methods are used for both prediction and control. For prediction, we can use the n-step TD method to estimate state-values v_π and adjust the number of steps. For example, a two-step update would be based on the first two rewards and the estimated value of the state two steps later. Similarly, we could have three-step updates, four-step updates, and so on. For control, we can use n-step SARSA/Expected SARSA to find the best action-value estimates q_*. Furthermore, off-policy learning is available to n-step methods. Just like it is

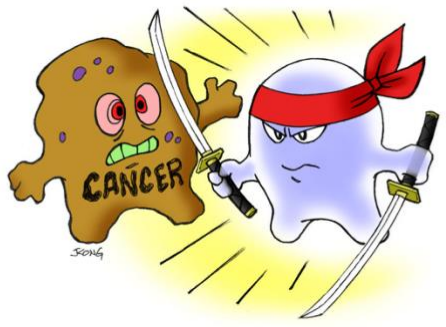


first introduced in Monte Carlo methods, we have a behaviour policy to explore the environment (e.g., $\epsilon - greedy$), and a target policy that acts greedily to select the best action available. The importance sampling ratio is applied to take into account the difference between the two policies.

Eligibility Traces unify and generalize TD and Monte Carlo methods. In earlier chapters, TD(0) assumes $\lambda = 0$ and Monte Carlo assumes $\lambda = 1$. The trace-decay parameter λ offers an elegant



mechanism with significant computational advantages since only a single trace vector is required to store in the learning process. We can decide how to update each state by looking forward to future rewards and state (forward view) or looking backward with eligibility traces (backward view).



According to the research (Noori et al., 2020), the Eligibility Traces algorithm with a backward view is used to determine the optimal drug dosage for decreasing the cancer cells in Chronic Myelogenous Leukaemia (CML) patients. Traditionally, physicians determine the optimal drug dosage by using their knowledge, experience, and clinical data of patients. Low drug dosage has little effect on reducing the growth rate of cancer cells; high drug dosage is harmful to the normal cells. But trial & error is a time-consuming task and can even be harmful to patients.

Extensive mathematical models represent the dynamics of the CML patient's body. The first equation describes the number of cancer cells, in which the cancer cells are generated and increased by the rate of $r_l \ln (\frac{l_a}{L(t)})$, and the minus sign indicates the eliminated cancer cells.

$$\frac{\partial L(t)}{\partial t} = r_l L(t) \ln\left(\frac{l_a}{L(t)}\right) - \gamma_l L(t) - f_l(h) L(t)$$

The second equation describes the number of normal cells in the patients' body, in which the normal cells are generated and increased by the rate of $r_n \ln\left(\frac{N_a}{N(t)}\right)$, and the minus sign indicates the eliminated normal cells due to the side effect of the drugs.

$$\frac{\partial N(t)}{\partial t} = r_n N(t) \ln\left(\frac{N_a}{N(t)}\right) - \gamma_n N(t) - cN(t) - f_n(h) N(t)$$

The third equation describes the number of the chemotherapy agent at each timestep, in which the agents are increased by injecting the drug $\mu(t)$.

$$\frac{\partial h(t)}{\partial t} = r_h h(t) + \mu(t)$$

Therefore, they build an RL environment where

- the states are considered as 2-dimensional states, including the cancer cells and normal cells
- the action increases or decreases the amount of drug dosage
- the reward is calculated by minimizing both cancer cells and drug dosage.

The eligibility trace can help to update the state-action pair more effectively; the more recent and frequent the state action pair you visit, the larger the weights. It will start decaying as you move to the next state. Researchers provide an optimal control method to compare the results with the RL approach. Although the control method can reduce the population of cancer cells faster, the side effects have not been considered and the drug dosage is higher than the RL method. Thus, the eligibility trace algorithm is a balanced method that performs well in both cancel cell reduction and normal cell protection.

References

Chandrashekar, A. (2018, September 12). *Netflix artwork personalization through multi-armed bandit testing*. Reforge. https://www.reforge.com/brief/netflix-artwork-personalization-through-multi-armed-bandit-testing#6tnTA-V1RNqadMkEeSEoVA

Feraud, R., Allesiardo, R., Urvoy, T., & Clerot, F. (2016). Random Forest for the Contextual Bandit Problem. *Proceedings of Machine Learning Research*, 93–101. http://proceedings.mlr.press/v51/feraud16.pdf

Krasheninnikova, E., García, J., Maestre, R., & Fernández, F. (2019). Reinforcement learning for pricing strategy optimization in the insurance industry. *Engineering Applications of Artificial Intelligence*, *80*, 8–19. https://doi.org/10.1016/j.engappai.2019.01.010

Naghibi-Sistani, M., Akbarzadeh-Tootoonchi, M., Javidi-Dashte Bayaz, M., & Rajabi-Mashhadi, H. (2006). Application of Q-learning with temperature variation for bidding strategies in market based power systems. *Energy Conversion and Management*, *47*(11–12), 1529–1538. https://doi.org/10.1016/j.enconman.2005.08.012

Noori, A., Alfi, A., & Noori, G. (2020). An intelligent control strategy for cancer cells reduction in patients with chronic myelogenous leukaemia using the reinforcement learning and considering side effects of the drug. *Expert Systems*, *38*(3), 1–13. https://doi.org/10.1111/exsy.12655

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, second edition: An Introduction (Adaptive Computation and Machine Learning series)* (second edition). Bradford Books.

Vittori, E., Trapletti, M., & Restelli, M. (2020). Option Hedging with Risk Averse Reinforcement Learning. *Option Hedging with Risk Averse Reinforcement Learning*, 1–8. https://arxiv.org/abs/2010.12245

Yang, Z. (2019, December 3). *Better bandit building: Advanced personalization the easy way with AutoML Tables*. Google Cloud Blog. https://cloud.google.com/blog/products/ai-machine-learning/how-to-build-better-contextual-bandits-machine-learning-models