First Try of Reinforcement Learning: A Tutorial

Final Project of STAT616, Fall 2021[*]

Dayu Yang

December 2021

1 Reinforcement Learning Basics

1.1 DQN

Given the reward of a game from the start to the end:

$$R_1, \cdots, R_t, \cdots, R_n$$

set a **hyperparameter** $\gamma \in [0,1]$, we have the Utility function U_t as:

Definition 1 (Definition of Discounted Reward)**.**

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \cdots + \gamma^{n-t} \cdot R_n$$

This idea is similar to the recent value of a continuous cash flow in finance.

While before we end the game, U_t is a random variable. If we are currently at time t, the randomness came from all the actions after t.

Definition 2 (Definition of Action-Value Function)**.**

$$Q_\pi(s_t, a_t) = E\left[U_t \mid S_t = s_t, A_t = a_t\right]$$

The E eliminate all the randomness after t (from all states $S_{t+1}, \cdots, S_n$, and all actions $A_{t+1}, \cdots, A_n$). Clearly, the objective of our learning problem is to find the maximum value of the action-value function:

Definition 3 (Definition of the best Action-Value Function)**.**

$$Q_\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t), \quad \forall s_t \in \mathcal{S}, \quad a_t \in \mathcal{A}.$$

This value $Q_\star(s_t$ means: no matter how you control the robot, the reward can never surpass this value (A upper bound).[1] If we know this $Q_\star$, at time t,

[*]Code Please see https://github.com/dayuyang1999/RL$_tutorial$

[1]Notice that "the best Action-Value Function" is still a function of Action and state.

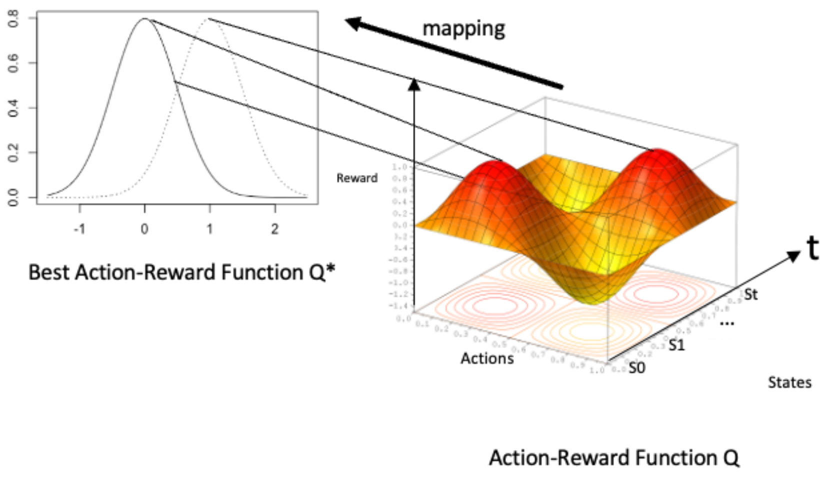

Figure 1: The relationship between Q and $Q_\star$

we can **forcast** what is the expectation of all future rewards after t. $Q_\star(s_t$ is the function we are trying to learn.

In DQN, $Q_\star(s_t$ is learned by CNN related model, denote as $Q(s, a; w)$, where w is the parameter in the network. The input of DQN is the current state S_t, while the output is an action that we react to the state. So basically, the whole learning process can be seen as **an image classification task**.

1.2 TD Algorithm

One of the learning algorithms of DQN is TD(Time Difference) Algorithm. The mean idea is that: Instead of taking a tons of true samples at the same time and estimate the parameters in a function, we can gradually learn a function. As the sample size gradually increase, our estimation is gradually becoming more precise. The idea is suitable for reinforcement learning task since the agent gradually knows the environment better and better during the exploration process.

We start from "the discounted reward" U_t:

$$U_t = \sum_{k=t}^{n} \gamma^{k-t} \cdot R_l = U_{t+1} = \sum_{k=t+1}^{n} \gamma^{k-t-1}$$

U_t can be gradually push forward as:

$$U_t = R_t + \gamma \cdot \underbrace{\sum_{k=t+1}^{n} \gamma^{k-t-1} \cdot R_k}_{=U_{t+1}} \tag{1}$$

Recall "the best action-value function" is:

$$Q_{\star}\left(s_t, a_t\right)=\max_{\pi} E\left[U_t \mid S_t=s_t, A_t=a_t\right] \tag{2}$$

According to (1) and (2), we can derive "optimal Bellman equations" as:

Definition 4 (Definition of optimal Bellman equations)**.**

$$\underbrace{Q_{\star}\left(s_t, a_t\right)}_{U_t's\ expectaion}=E_{S_{t+1} \sim p\left(\cdot \mid s_t, a_t\right)}[R_t+\gamma \underbrace{\max_{A \in \mathcal{A}} Q_{\star}\left(S_{t+1}, A\right)}_{U_{t+1}'s\ expectaion} \mid S_t=s_t, A_t=a_t] \tag{3}$$

The idea of "Bellman Equation" is: **we can learn "the best action-value function" step by step, by keeping interact with the environment.** (Gradually, we know the environment better.)

The right side of Bellman Equation is an expectation. Since $Q_\star$ is modeled by a network. We can only compute the integral numerically (using Monte Carlo Method to estimate). When the robot take action a_t, we use state-transition function $p\left(s_{t+1} \mid s_t, a_t\right)$ to compute s_{t+1}. Since R_t only relies on S_t, A_t, S_{t+1}. So after we observe s_t, a_t, s_{t+1}, we observe R_t as r_t[2].

This means after a_t is acted, feedback s_{t+1} is given, we have a 4-elements tuple as:

$$\left(s_t, a_t, r_t, s_{t+1}\right)$$

This tuple allows us to compute:

$$r_t+\gamma \cdot \max_{a \in \mathcal{A}} Q_{\star}\left(s_{t+1}, a\right) \tag{4}$$

Term (4) can be seen as the Monte Carlo estimation of

$$E_{S_{t+1} \sim p\left(\cdot s_t, a_t\right)}\left[R_t+\gamma \cdot \max_{A \in \mathcal{A}} Q_{\star}\left(S_{t+1}, A\right) \mid S_t=s_t, A_t=a_t\right]$$

So after t *th* exploration, we can estimate the best action-value function as:

$$Q_{\star}\left(s_t, a_t\right) \approx r_t+\gamma \cdot \max_{a \in \mathcal{A}} Q_{\star}\left(s_{t+1}, a\right)$$

By replacing $Q_\star(s_t$ with $Q(s, a; w)$(This is a DQN), we have:

$$\underbrace{Q\left(s_t, a_t ; w\right)}_{Estimate \widehat{q_t}} \approx \underbrace{r_t+\gamma \cdot \max_{a \in \mathcal{A}} Q\left(s_{t+1}, a ; w\right)}_{Objective\ of\ TD, \hat{y}_t} \tag{5}$$

Equation (5) is the key of TD algorithm. On the left side, we have the estimated Action value $Q_\star\left(s_t, a_t\right)$ before observing r_t. On the right side, we have a new estimate of Action value(**which is assumed to be more accurate**).

[2]A capital letter represents Random Variable, while a lowercase letter represents value(already observed)

So, TD-algorithm encourage $\widehat{q}_t Q(s_t, a_t; w)$ to approach $\hat{y}$, where we can define the loss function following this idea:

$$L(w) = \frac{1}{2}\left[Q(s_t, a_t; w) - \widehat{y}_t\right]^2$$

If we assume $\widehat{y}$ is a constant[3], we can compute the gradient of $\mathbf{w}$ on L:
$\nabla_w L(w) = \underbrace{(\widehat{q}_t - \widehat{y}_t)}_{\mathrm{TD}_{error\delta_t}} \cdot \nabla_w Q(s_t, a_t; w)$

Then, follow what deep learning always do – gradient descent:

$$w \leftarrow w - \alpha \cdot \delta_t \cdot \nabla_w Q(s_t, a_t; w)$$

We update parameter $\mathbf{w}$ in the Q(DQN).

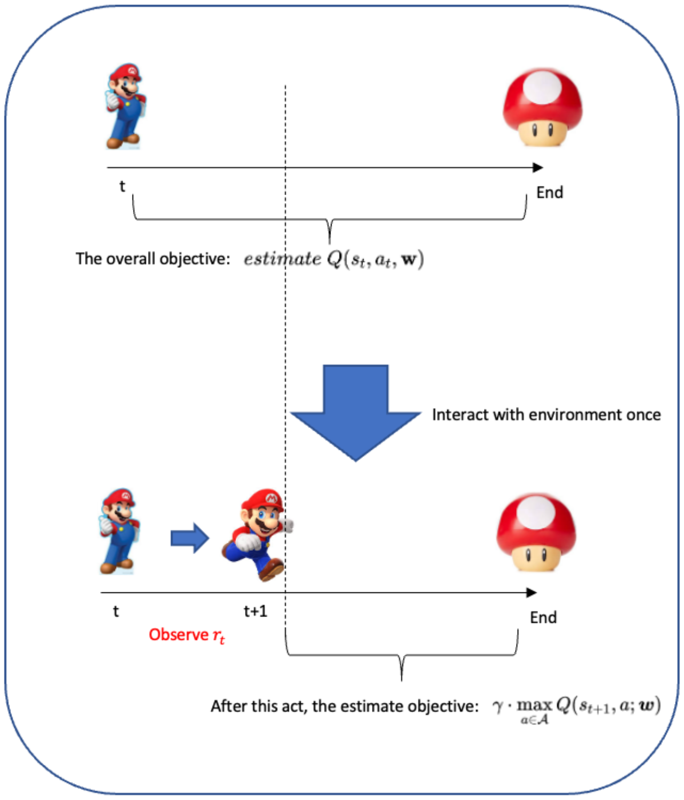
t
End
The overall objective: $estimate\ Q(s_t, a_t, \mathbf{w})$
Interact with environment once
t
t+1
Observe r_t
End
After this act, the estimate objective: $\gamma \cdot \max_{a\in\mathcal{A}} Q(s_{t+1}, a; \boldsymbol{w})$

Figure 2: Explain Bellman Equation

[3]actually it depends on Q, and Q is a DQN. So $\widehat{y}$ depends on $\mathbf{w}$

1.3 Training DQN following a behavior policy

Recall how we update the parameters of DQN:

- compute the estimation of action-value: $\widehat{q}_t = Q\left(s_t, a_t; w\right)$
- agent interact with the environment, yields a 4-elements tuple $\left(s_t, a_t, r_t, s_{t+1}\right)^4$.
- according to the new observed real-world data, compute TD-target: $\widehat{y}_t = r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q\left(s_{t+1}, a; w\right)$
- compute the differences between two estimations: $\delta_t = \widehat{q}_t - \widehat{y}_t$
- update parameters in DQN $w \leftarrow w - \alpha \cdot \delta_t \cdot \nabla_w Q\left(s_t, a_t; w\right)$

Here we arise a question: **Should I pick a choice that seem to be the best based on existing knowledge? Or should I pick a choice aiming to explore the environment better**. This is the well-known Exploitation and Exploration dilemma.

- Exploitation:
  - ”being greedy” and being ”short-sighted”
  - too much exploitation: Regret of missing unexplored “jade”
  - Go to your favorite restaurant
  - Show the most successful advertisement
  - Drill the crude oil at the best known location
  - Play the move you believe is best in a video game
- Exploration:
  - “gaining info” and being “long-sighted”
  - too much Exploration : Regret of wasting time on “garbage choice”
  - Try a new restaurant
  - Show a new advertisement
  - Drill the crude oil at a new location
  - Play an experimental move in a video game

How we decide the strategy of our agent to interact with the environment named: ”Behavior policy” π. There are many available policies. Here I will introduce three well-known strategies in multi-arm (agent) bandit algorithm: ETC, epsilon-greedy, UCB, and Thompson sampling.

[4] Notice that this 4-elements tuple is unrelated to the ”interacting strategy” π

1.4 Explore-Then-Commit (ETC)

1.4.1 Regret Guarantee

The following content aim to show the quantity of the upper bound of regret of ETC under stochastic bandit setting (Lattimore & Szepesvari, 2020).

We have an unknown distribution $p_i = u_i + \epsilon$ where ϵ is draw from i.i.d. standard gaussian. (Note that we only require ϵ to be sub-gaussian for the analysis to go through). At each time step, ETC algorithm selects 1 action A_t (also called "Arm" in Bandit algorithm), and receives a loss (called regret in bandit) g_{t,A_t} drawn i.i.d. from the distribution of the action distribution A_t. Here we focus on pseudo loss, means "the realized regret - the optimal regret"[5].

$$Regret_T := E\left[\sum_{t=1}^{T} g_{t,A_t}\right] - \min_{i=1,\ldots,d} E\left[\sum_{t=1}^{T} g_{t,i}\right] = E\left[\sum_{t=1}^{T} g_{t,A_t}\right] - \min_{i=1,\ldots,d} \mu_i$$

Before the proof of Upper bound, here we present some inequality about statistical estimations.

$X_1, X_2, \ldots, X_n$ is i.i.d. We have $\mu = E[X]$ and $\sigma = Var[X]$. Estimate the mean by the empirical mean:

$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i$

The estimation is unbiased, so mean of estimation is μ. While the variance is decreased:

$$Var(\hat{\mu}) = \frac{Var(X)}{n} = \frac{\sigma^2}{n}$$

The distance between the real number μ and estimation $\hat{\mu}$ is upper bounded by(Chebyshev's inequality):

$$Pr[|\hat{\mu} - \mu| \geq \epsilon] \leq \frac{Var[\hat{\mu}]}{\epsilon^2}$$

By plugging in the var of estimation, this is equal to:

$$Pr[|\hat{\mu} - \mu| \geq \epsilon] \leq \frac{\sigma^2}{n\epsilon^2}$$

Now using the formula above, we can give **the probability of having a bad estimation**. However, we can further decrease the upper bound by the central limit theorem.

We assume $S_n = \sum_{t=1}^{n}(X_i - \mu)$. With the central limit theorem, $\frac{S_n}{\sqrt{n\sigma^2}} N(0,1)$ (if n goes unlimited large). Then we can estimate $Pr[\hat{\mu} - \mu \geq \epsilon]$ by the integral of the density function of N(0,1).

$$Pr[\hat{\mu} - \mu \geq \epsilon] = Pr\left[S_n \geq n\epsilon\right] = Pr\left[\frac{S_n}{\sqrt{n\sigma^2}} \geq \sqrt{\frac{n}{\sigma^2}}\epsilon\right] \approx \int_{\epsilon\sqrt{\frac{n}{\sigma^2}}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx$$

[5]regret = loss = - reward.

while this also means:

$$Pr[\hat{\mu} - \mu \geq \epsilon] = \lim_{n \to \infty} \int_{\epsilon\sqrt{\frac{n}{\sigma^2}}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx$$

Here we have an integral in the right side(As a machine learning player, I think we all know integral is most likely to be incomputable). We don't need to compute it since our goal is to find a upper bound.

since a is the smallest number we can take (exp(-x) is a monotonous decaying function), we have:

$$\int_a^{\infty} \exp\left(-\frac{x^2}{2}\right) dx = \int_a^{\infty} \frac{x}{x} \exp\left(-\frac{x^2}{2}\right) dx \leq \frac{1}{a} \int_a^{\infty} x \exp\left(-\frac{x^2}{2}\right) dx = \frac{1}{a} \exp\left(-\frac{a^2}{2}\right)$$

set $a = \epsilon\sqrt{\frac{n}{a^2}}$, we get another upper bound which is tighter than Chebyshev's inequality:

$$Pr[\hat{\mu} - \mu \geq \epsilon] \sqrt{\frac{\sigma^2}{2\pi\epsilon^2 n}} \exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right)$$

The tiny modification of the equation above was called the Hoeffding's bound:

$$Pr[\hat{\mu} \geq \mu + \epsilon] \leq \exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right) \quad and \quad Pr[\hat{\mu} \leq \mu - \epsilon] \leq \exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right)$$

Accordingly, if we have a Gaussian random variable (We assume a subgaussian noise for convenience), we have:

$$P\left(\mu \in \left[\hat{\mu} - \sqrt{\frac{2\sigma^2 \ln\frac{1}{\delta}}{n}}, \hat{\mu} + \sqrt{\frac{2\sigma^2 \ln\frac{1}{\delta}}{n}}\right]\right) = 1 - 2\delta$$

Now we formally introduce the Explore-Then-Commit Algorithm, which is the most natural algorithm for stochastic bandit. The explore-then-commit strategy is characterized by a natural number m, which is the number of times each arm will be explored before committing. Thus the algorithm will explore for mK rounds before choosing a single action for the remaining $n-mK$ rounds. We can write this strategy formally as:

$$\mathrm{A}_t = \begin{cases} (t \; mod \; d) + 1, & t \leq mK \\ argmax_i \hat{\mu}_i(mK), & t > mK \end{cases}$$

where we denote $S_{t,i} = \sum_{j=1}^{t} 1\left[A_t = i\right]$, which is the number of times that the arm i was pulled in the first t rounds.

recall that $\hat{\mu}_i(t)$ is the average loss from arm i up to round t:

$$\hat{\mu}_i(t) = \frac{1}{T_i(t)} \sum_{s=1}^{t} I\left\{A_s = i\right\} X_s$$

where $T_i(t) = \sum_{s=1}^{t} I\left\{A_s = i\right\}$ is the number of times arm(action) i was chosen up to the end of round t. In the ETC, $T_i(t)$ is deterministic for $t \leq mK$(at the beginning). and later for$t > mK$, $\hat{\mu}_i(t)$ is taken into account.

Lemma 1. *UB for any policy*[6]

$$Regret_T = \sum_{i=1}^{d} E\left[S_{T,i}\right] \Delta_i$$

Now, we can obtain the regret guarantee of the ETC algorithm:

Theorem 1. *Assume that the losses of the arms minus their expectations are 1-subgaussian and $1 \leq m \leq T/d$, Then, ETC guarantees a regret of*[7]*:*

$$Regret_T \leq m \sum_{i=1}^{d} \Delta_i + (T - md) \sum_{i=1}^{d} \Delta_i \exp\left(-\frac{m\Delta_i^2}{4}\right)$$

The bound shows the trade-off between exploration and exploitation: if m is too big, we pay too much during the exploration phase (first term in the bound). On the other hand, if m is small, the probability to select a suboptimal arm increases (second term in the bound). Knowing all the gaps Δ_i, it is possible to choose m that minimizes the bound.

1.4.2 Experiment

Consider unstructural bandit problem. Suppose we have k arms, each with random rewards $p_i = u_i + \epsilon$ where ϵ is draw from i.i.d. standard gaussian. (Note that we only require ϵ to be sub-gaussian for the analysis to go through)

The hyperparameter of ETC is the exploration time mk. We design a experiment to implement the standard ETC and doubling trick. An advantage of the doubling trick is that it does not rely on knowing the time horizon. This comes at the cost of a constant factor increase of regret, its most obvious disadvantage. The algorithm still does require foreknowledge of the suboptimality gap which is likely not known.

> [H]Play each arm in the round-robin fashion until each arm are played m times Compute the empirical reward estimation for each arm Play the best arm (according to the empirical reward estimation) until the end of the game

[6]proof in Appendix
[7]Proof see Appendix

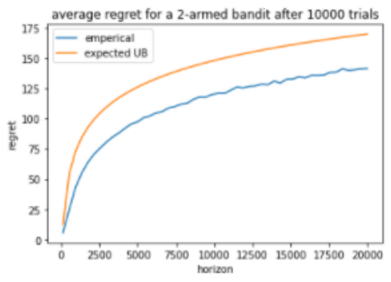

Figure 3: The expected regret of ETC(theoretical) VS Practical

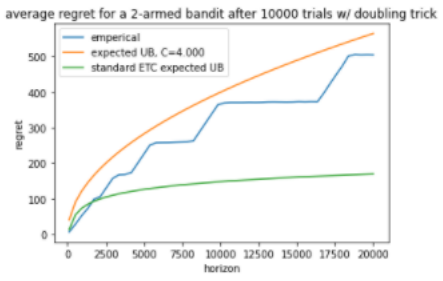

Figure 4: The expected regret of ETC with doubling trick

Final thoughts about ETC: With two arms of gaussian distributions, uniform sampling is an optimal exploration strategy. For multiple arms, the exploration strategy could be different. When confidence that an arm is suboptimal is sufficiently large it could be dropped from the exploration pool. Continually selecting the best arm is also optimal in the exploitation phase. Other than better exploration strategies, it is difficult to see where ETC could be improved given the information it has access to and how close it is to the optimal solution.

1.4.3 Other Algorithm

Due to the time limit, I only explore the experiment part of UCB, LinUCB, ETC, and Thompson Sampling.

First, I explore UCB vs LinUCB, and basic LinUCB algorithm implementation, for both Gaussian Reward and Bernoulli rewards in Figure 5 and 6.

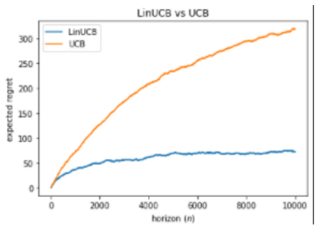

Figure 5: UCB vs LinUCB

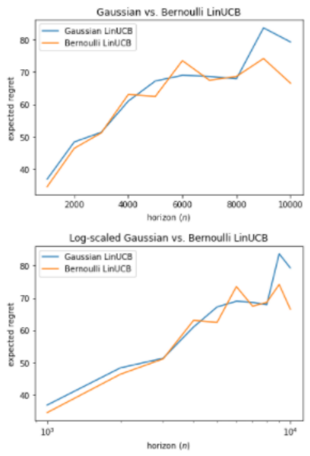

Figure 6: Gaussian vs Bernoulli reward under LinUCB

Second, I explore Thompson Sampling vs the standard UCB in Figure 7.

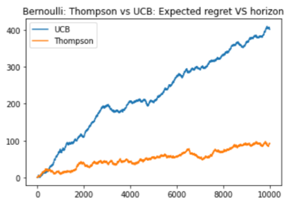

Figure 7: UCB vs Thompson Sampling under bernoulli reward

2 Experiments on CartPole

For the CartPole problem. I mainly did two things:

- Explore the influence of model design

- Explore the influence of parameter (Exploration and Exploitation trade-off(decaying factor), discount factor Γ)

2.1 Model Selection

For model selection part, I explore 4 setting:

- Vanilla Setting
- max-pooling in CNN
- going deeper in CNN
- going deeper in FCN

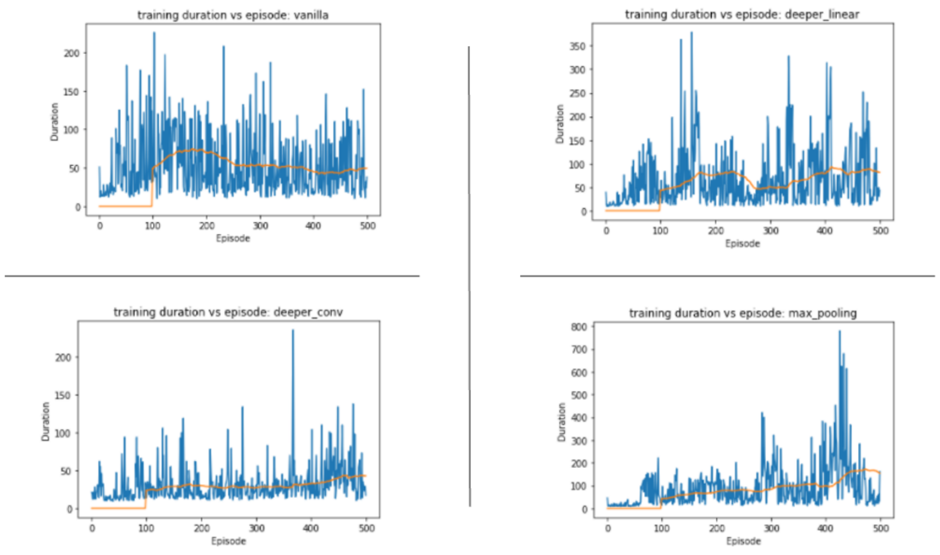

Figure 8: Robot Performance with different Model Setting

The results are shown in Figure 8. We can clearly observe that the model with max-pooling gives the best performance. The model with a deeper CNN structure compared with the vanilla model does not improve the performance of our robot at all. The average duration is even decrease from 50 (the vanilla model) to 30 (the model with a deeper CNN). This may make sense because after so many stacked convolutional layers, the receptive field becomes a 3 by 2 image with 64 channels. The kernel may no longer observe any useful pattern from such a small receptive field. In other words, the larger receptive field may keep the information of "how good the agent play" more precise and informative. For the model with deeper linear layer, we do see a significant increase of performance.

Until now, we can conclude our observation as: For a DQN, the receptive field should not be too small. And a deeper FCN do help our agent understand what behavior can obtain higher reward.

2.2 Parameter Tuning

For the setting of CartPole, there are mainly two parameters which are worth tuning: gamma and decaying factor of epsilon greedy algorithm.

2.2.1 Gamma

gamma is the discount factor for future reward $\gamma \in (0, 1)$. A larger gamma means we see the reward from the future as the same as the present. This encourage our model to explore the environment since better exploration allows us to get higher rewards in the future. While a smaller gamma encourage our model to be more greedy at present, or encourage exploitation of the current information we know about how to get the best reward.

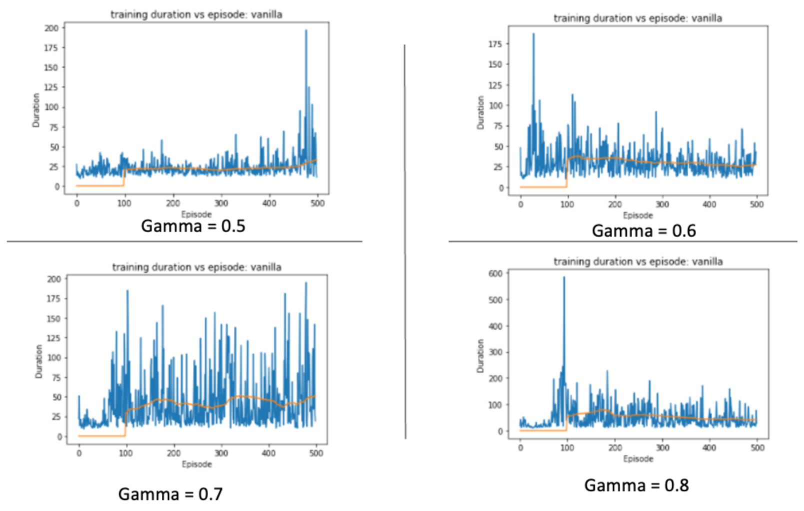

Figure 9: Robot Performance with different Gamma

From Figure 9, we can clearly observe that a higher gamma will eventually yields much higher rewards value, while lower gamma prohibit learning.

2.2.2 Decaying factor

Decaying factor also controls the tradeoff between exploration and exploitation. Since we set the probability of a random variable X $N(0, 1)$ larger than the value of the function as the probability to explore. So in Figure 10, $y = \exp\left(-\frac{x}{1}\right)$ is encourage exploitation compared with $y = \exp\left(-\frac{x}{10}\right)$ (which has larger decaying factor).

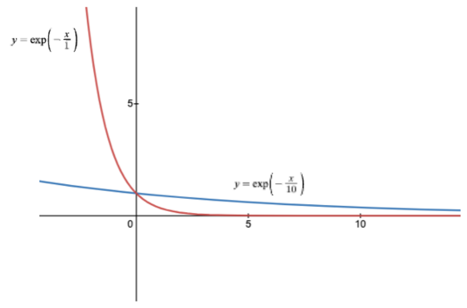

Figure 10: two decaying function

From Figure 11, we can observe that a smaller decaying factor (say, 1) will prohibit exploitation which eventually yield a worse performance overall compared to the learning strategy with higher decaying factor. However a very large decaying factor is also harmful to the overall performance since the agent is always unable to deploy the best play[8]. The best performance is located when $\text{eps}_decayisequalto100$.

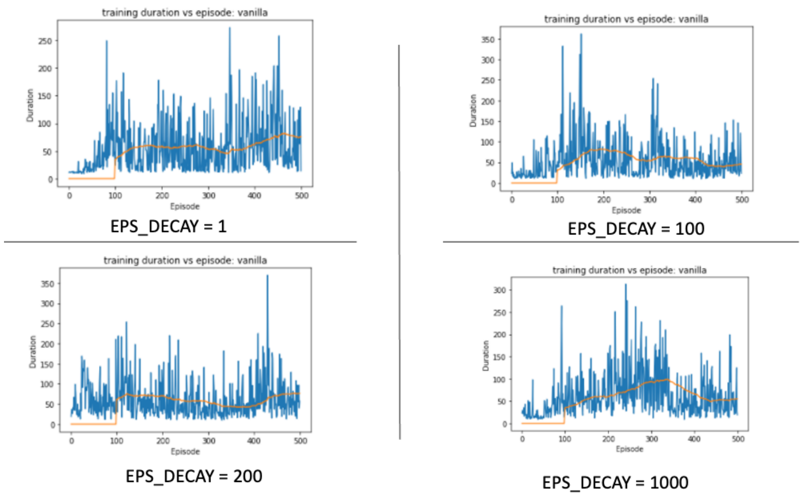

Figure 11: Experiments with different Decaying factor for epsilon-greedy algorithm

[8]Notice that the large or small is also related to the absolute value of number of episode

3 Conclusion

Reinforcement Learning is a power learning method, which is also an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. The most interesting part of RL is that we have strong statistical theory from the bandit algorithm as well as many useful practice in the real world.

Appendix

Appendix A: TD Algorithm

From the definition of discounted reward U_t, we can get a recursive equation:

$$\begin{aligned} U_t &= R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^4 \cdot R_{t+4} + \cdots \\ &= R_t + \gamma \cdot \underbrace{\left(R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+4} + \cdots\right)}_{= U_{t+1}} \end{aligned}$$

Lemma 2. *Reward recursive formula*

$$U_t = R_t + \gamma \cdot U_{t+1}$$

The lemma 1 told us the relationship between two consecutive rewards. And it could be used to derive TD-target.

Asumme that the reward at time t, R_t, depends on (S_t, A_t, S_{t+1}). According to the definition, Action-value function Q_π equals to the conditional expectation of the discounted reward(The Expectation is taken over all the future actions a_t and states s_t to eliminate randomness) :

$$Q_\pi(s_t, a_t) = E\left[U_t \mid s_t, a_t\right]$$

If we bring lemma 1 to the equation above(replacing the current discounted reward U_t by ...), we get:

$$Q_\pi(s_t, a_t) = E\left[U_t \mid s_t, a_t\right] = E\left[R_t + \gamma \cdot U_{t+1} \mid s_t, a_t\right] = E\left[R_t \mid s_t, a_t\right] + \gamma \cdot E\left[U_{t+1} \mid s_t, a_t\right] \tag{6}$$

For $E\left[U_{t+1} \mid s_t, a_t\right]$, this term is the expectation over $E\left[U_{t+1} \mid s_t, a_t\right]$(expectation over all future states and actions). Recall that $E(U_{t+1}) = E\left[U_{t+1} \mid s_t, a_t\right]$. So we can replace $E\left[U_{t+1} \mid s_t, a_t\right]$ to $E\left[Q_\pi(S_{t+1}, A_{t+1}) \mid s_t, a_t\right]$.

This allows us to move one step further:

$$Q_\pi(s_t, a_t) = E\left[R_t \mid s_t, a_t\right] + \gamma \cdot E\left[Q_\pi(S_{t+1}, A_{t+1}) \mid s_t, a_t\right]$$

In this equation, on the left side, $Q_\pi(s_t, a_t)$ is the action value at time t. It can be replaced by the terms on right side. The key is that: through this formula, we can estimate $Q_\pi(s_t, a_t)$ by interacting with the environment step by step. Besides, the expectation E is taken over the random variable of action A_{t+1} and state S_{t+1}.

In practice, it is usually pretty hard to compute estimation under deep learning methods, so we approximate $E\left[R_t \mid s_t, a_t\right] + \gamma \cdot E\left[Q_\pi(S_{t+1}, A_{t+1}) \mid s_t, a_t\right]$ by Monte Carlo method.

R_t is a random variable, we can approximate it by the observation r_t. Similarly, for S_t and A_{t+1}, we replace them by the observation s_t and a_t.

Sum over everything so far.

$$Q_\pi(s_t, a_t) \approx E\left[r_t + \gamma \cdot Q_\pi(s_{t+1}, a_{t+1})\right]$$

Initially, we want to update the action value at time t $Q_{\pi]}$. This can be estimated by a expectation on the right hand side. Further, we approximate the estimation by a combination of "real-world observations" (r_{t+1}, s_{t+1}, a_{t+1}) and the "not-ground-truth" function Q_π. In other words, $Q_\pi(s_t, a_t)$ Partially depends on the real-world observation r_{t+1}, partially depends on the forecast from $Q_\pi(s_{t+1}, a_{t+1})$.

This equation tells us what TD-algorithm is trying to do: **Encourage $Q_\pi(s_t, a_t)$ (purely forecast) to approach TD target($E\left[r_t + \gamma \cdot Q_\pi\left(s_{t+1}, a_{t+1}\right)\right]$), which is partially depends on the real-world observation.**

.1 Appendix B: proof of Lemma 1

Proof: Observe that

$$\sum_{t=1}^{T} g_{t,A_t} = \sum_{t=1}^{T}\sum_{i=1}^{d} g_{t,i}\mathbf{1}\left[A_t = i\right]$$

Hence,

$Regret_T = E\left[\sum_{t=1}^{T} g_{t,A_t}\right] - T\mu^\star = E\left[\sum_{t=1}^{T}\left(g_{t,A_t} - \mu^\star\right)\right] = \sum_{i=1}^{d}\sum_{t=1}^{T} E\left[\mathbf{1}\left[A_t = i\right]\left(g_{t,i} - \mu^\star\right)\right]$

considering the former observation, this is equal to:

$$\sum_{i=1}^{d}\sum_{t=1}^{T} E\left[E\left[\mathbf{1}\left[A_t = i\right]\left(g_{t,i} - \mu^\star\right) \mid A_t\right]\right]$$

.2 Appendix C: proof of Theorem 1

Start with Lemma 1,

$$\sum_{t=1}^{T} E\left[\mathbf{1}\left[A_t = i\right]\right] = m + (T - md)Pr\left[\hat{\mu}_{md,i} \le \min_{j \ge i} \hat{\mu}_{md,j}\right]$$

we assume the optimal arm is the first one(can be any one in the sequence, but result will be the same):

$$m + (T - md)Pr\left[\hat{\mu}_{md,i} \le \min_{j \ge i} \hat{\mu}_{md,j}\right] \le m + (T - md)Pr\left[\hat{\mu}_{md,i} \le \hat{\mu}_{md,1}\right]$$

And this is equal to the following with minor adjustment:

$$m + (T - md)Pr\left[\hat{\mu}_{md,1} - \mu_1 - \left(\hat{\mu}_{md,i} - \mu_i\right) \ge \Delta_i\right]$$

from the Hoeffding's bound, we get:

$$Pr\left[\hat{\mu}_{md,1} - \mu_1 - \left(\hat{\mu}_{md,i} - \mu_i\right) \ge \Delta_i\right] \le \exp\left(-\frac{m\Delta_i^2}{4}\right)$$