دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره دو
درس یادگیری تعاملی
پاییز ۱۴۰۰

نام و نام خانوادگی	محمد عظیم پور
شماره دانشجویی	۸۱۰۱۹۷۶۵۷

فهرست

چکیده ۴

سوال ۱ - سوال پیاده‌سازی ۵

هدف سوال ۵

قسمت اول ۵

الف) ۵

توضیح پیاده سازی ۵

نتایج ۵

ب) ۷

توضیح پیاده سازی ۷

نتایج ۷

قسمت دوم ۱۰

الف) ۱۰

توضیح پیاده سازی ۱۰

نتایج ۱۰

ب) ۱۱

سوال ۲ - سوال تئوری ۱۴

الف) ۱۴

ب) ۱۴

سوال ۳ - سوال پیاده‌سازی ۱۶

هدف سوال ۱۶

الف) ۱۶

توضیح پیاده سازی ۱۶

نتایج ۱۶

ب) ۱۷
ج) ۱۸
منابع ۱۹

چکیده

در این تمرین الگوریتم های مختلف یادگیری ماشین در محیط n-Armed Bandit پیاده سازی و بررسی می شوند. در سوال اول ابتدا سه الگوریتم مختلف برای Bandit با توزیع های داده شده پیاده سازی می شود و پس از آن تاثیر استفاده از تابع utility به جای reward برای این الگوریتم ها مورد بررسی قرار می گیرد. در ادامه سوال یک و همچنین در سوال دو تعدادی مساله که هرکدام شرایطی از بازی را تعیین می کنند باید به صورت تحلیلی بررسی شوند.

در سوال سوم نیز مساله دیگری آورده شده است که باید با مدل کردن آن روی یک مساله n-Armed Bandit و مقایسه روش های یادگیری در کمترین تعداد تلاش ممکن جواب بهینه را پیدا کنیم.

سوال ۱ - سوال پیاده‌سازی

هدف سوال

هدف از این سوال مقایسه سه الگوریتم یادگیری Epsilon-Greedy، Gradiant Based و UCB است و همچنین تاثیر پارامترهای تابع utility بر یادگیری بررسی می شود.

قسمت اول

الف)

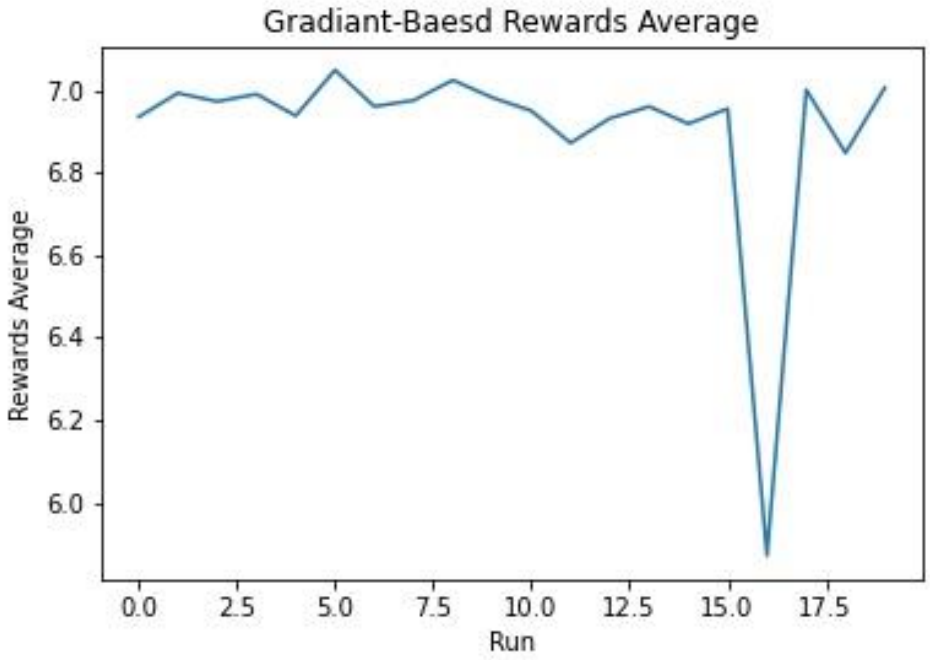
توضیح پیاده سازی

ابتدا با توجه به توزیع های گفته شده در صورت سوال یک نمونه از کلاس MutliArmedBanditEnvironment با ۴ بازو (که امید ریاضی پاداش های آن ها برابر ۷، ۵، ۴ و ۲ است) ایجاد شده است که به عنوان محیط به Agnetها داده می شود.

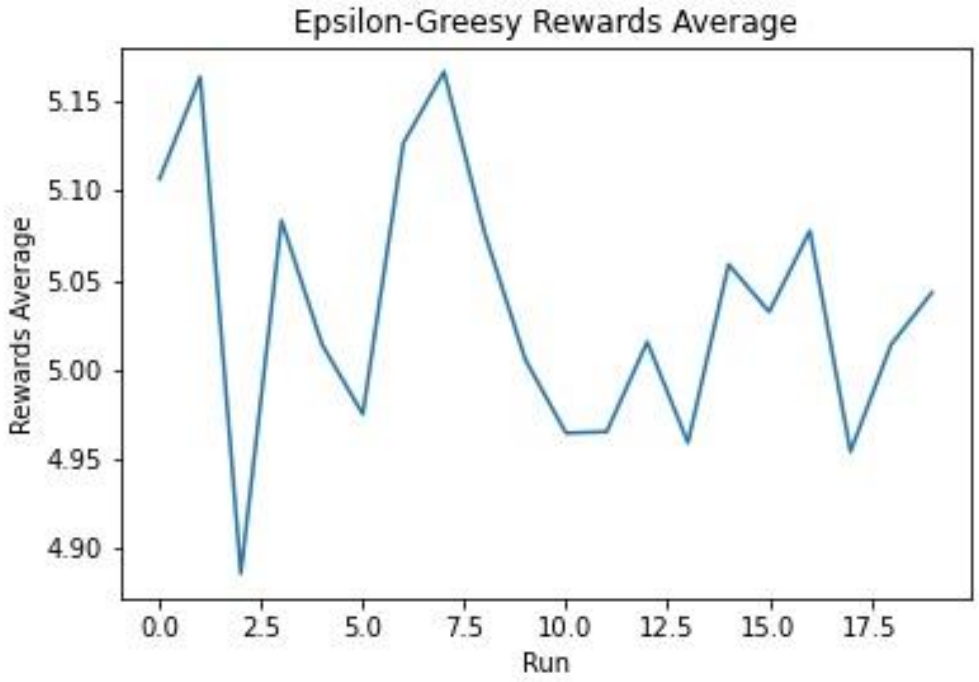
به ازای هر الگوریتم، کلاسی تحت عنوان سیاست پیاده سازی شده است که یکی از فیلدهای کلاس Agent متناظر آن الگوریتم است. این کلاس های سیاست وظیفه تصمیم گیری و انتخاب Action ها و همچنین به روزرسانی متغیرهایی که در تصمیم گیری دخیل هستند را بر عهده دارند. برای هر الگوریتم هم یک Agent که مشتق از کلاس AgentBase است پیاده سازی شده که سیاست آن در هنگام فراخوانی تابع سازنده ست می شود. Agent ها در هر گام در تابع take_action یک Action از کلاس سیاستشان دریافت و آن را اجرا می کنند و سپس با توجه به نتایج حاصل شده متغیرهای موجود را به تغییر می دهند. همچنین هر Agent یک تابع reset دارد که تمام متغیرها را به حالت اولیه شان برمی گرداند تا بتوانیم دوباره فرآیند یادگیری را اجرا کنیم.

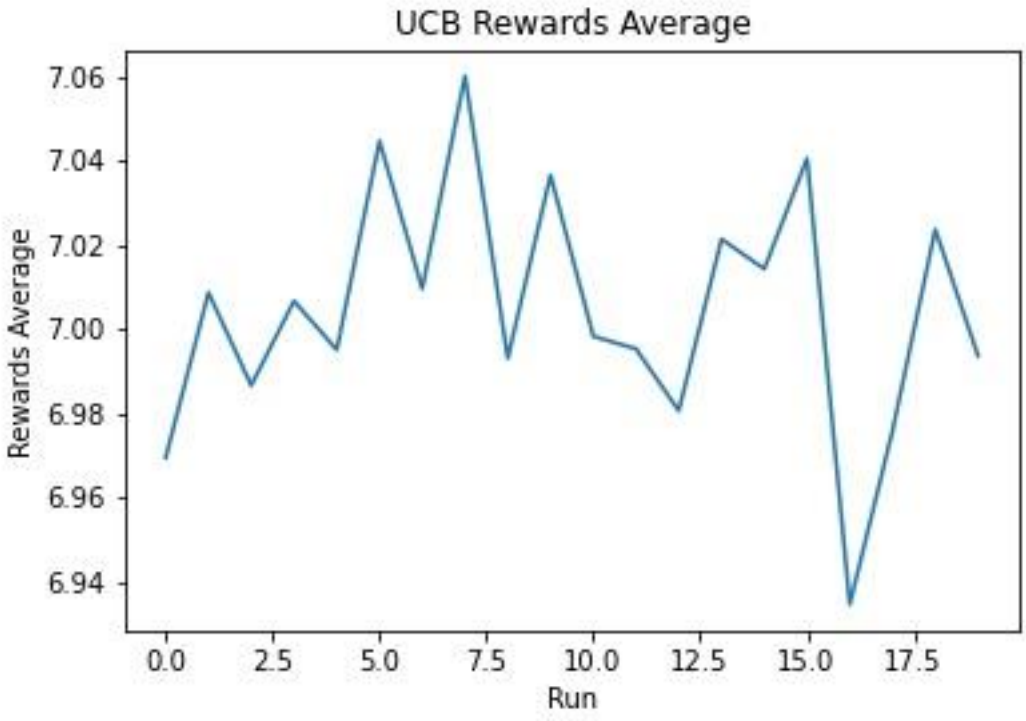
نتایج

در الگوریتم Gradiant Based مشاهده می شود که غیر از دو مرتبه در تمام اجراها Agent توانسته است Action بهینه را پیدا کند و میانگین پاداش دریافتی را به میانگین پاداش بهترین بازو یعنی عدد ۷ برساند. در دو اجرای دیگر به نظر می رسد که Agent در بیشینه موضعی گیر افتاده است و اشتباها بهترین بازو را بازو با میانگین پاداش ۵ تشخیص داده است.



مشاهده می شود که غیر از الگوریتم Gradiant Based، سایر الگوریتم ها صرفا چند بار به جواب بهینه همگرا شده اند و ۱۹ مرتبه دیگر هر بار عدد متفاوتی به عنوان میانگین پاداش حاصل شده است. در نتیجه می توان گفت که برای این الگوریتم ها ۱۰۰۰ trial برای یادگیری کم به نظر می رسد. با این حال به طور متوسط، سیاست UCB از سیاست Epsilon Greedy بهتر عمل می کند.





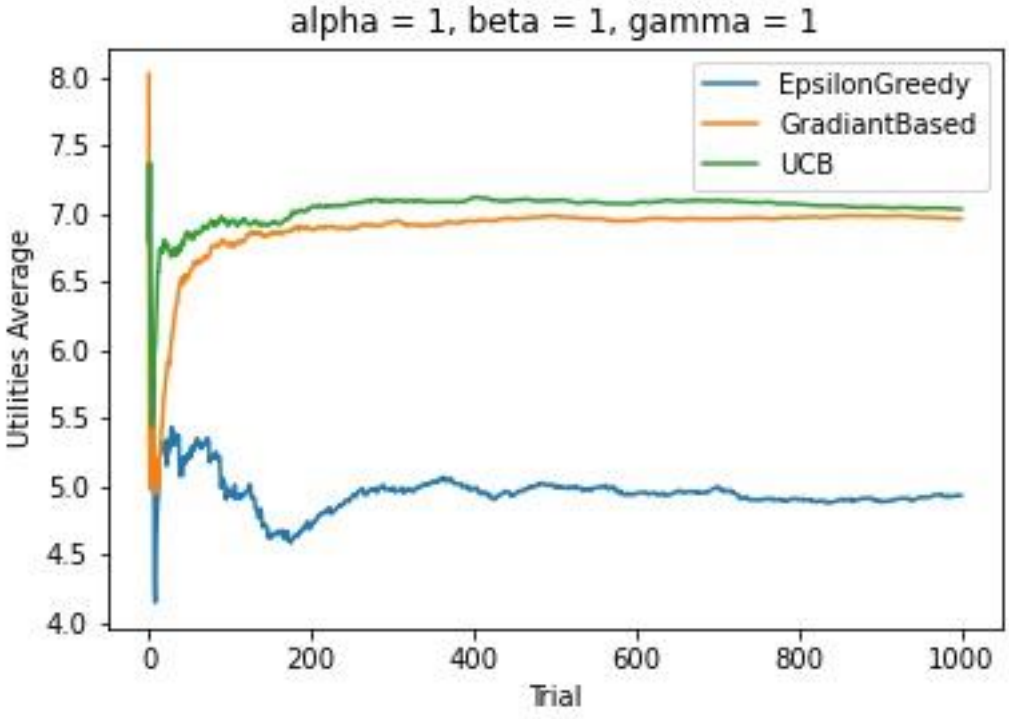
ب)

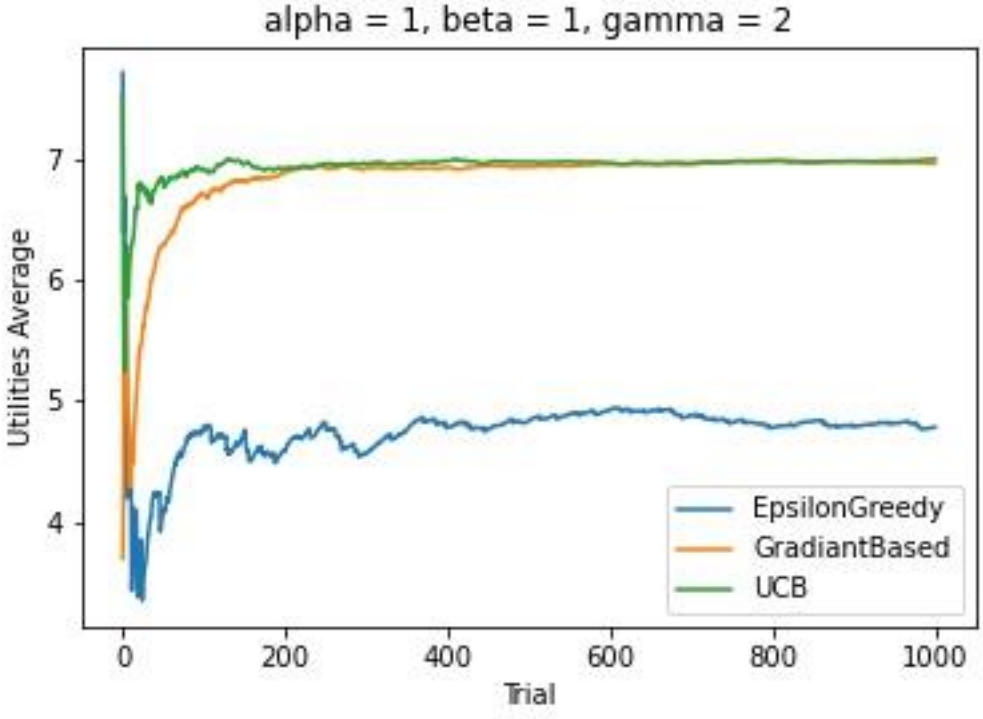
توضیح پیاده سازی

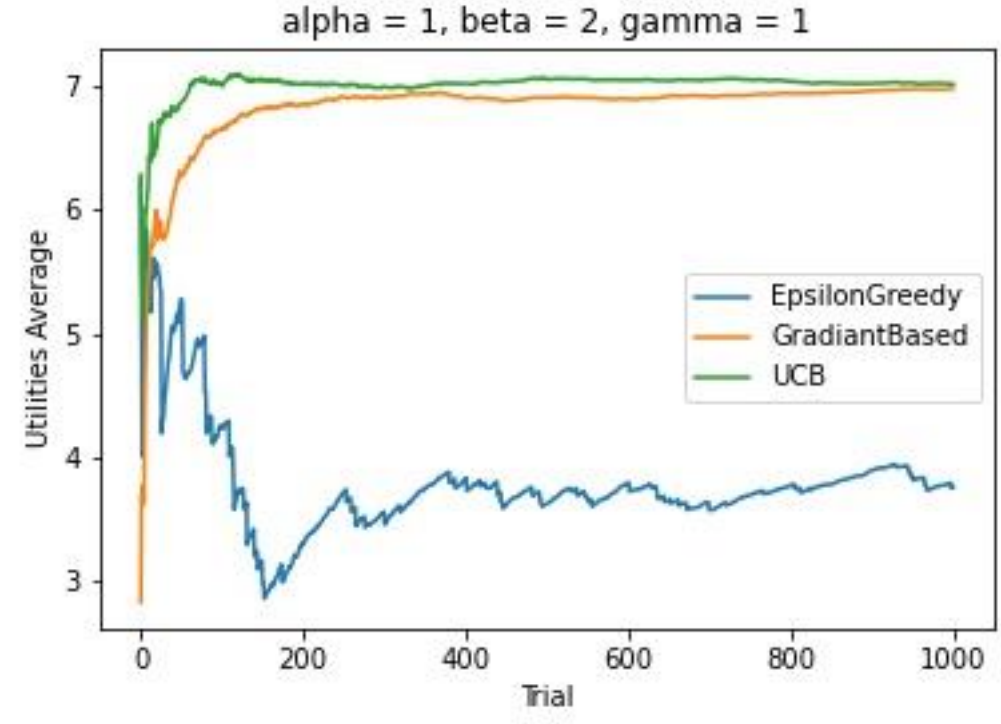
برای هر الگوریتم، یک Agent که به جای تابع پاداش تابع utility را ملاک قرار می دهد پیاده سازی شده است و این Agentها در ۴ حالت که مقادیر آلفا، بتا و گاما در آن ها تغییر می کند تمرین داده شده اند. برای هرکدام از این سه متغیر، دو مقدار یک و دو مورد بررسی قرار گرفته است.

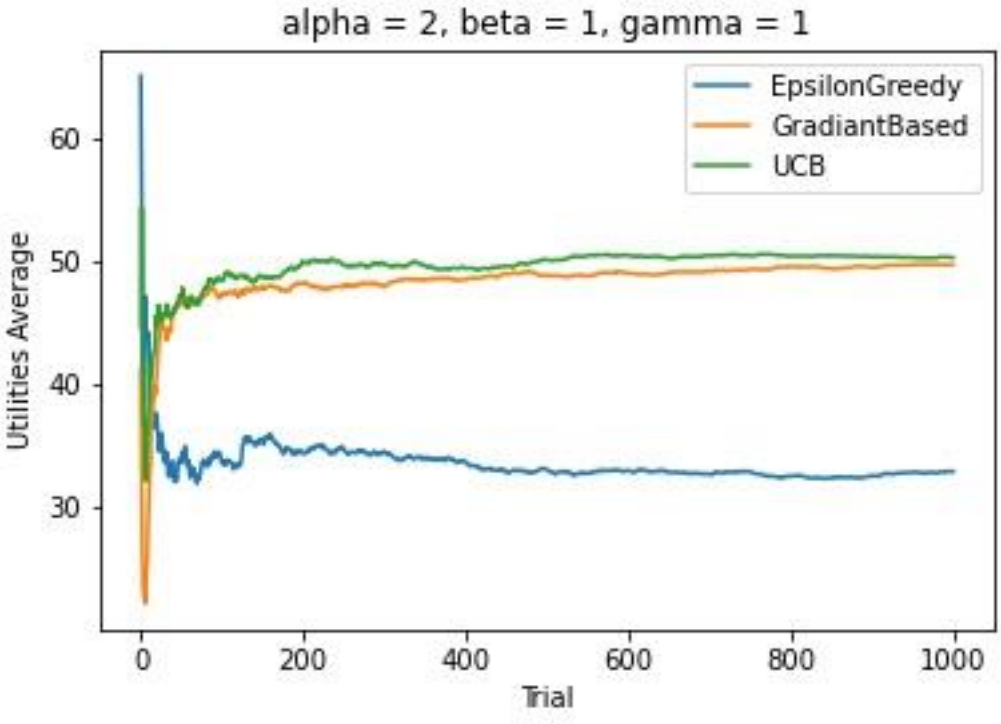
نتایج

با استفاده از تابع utility به جای reward، مشاهده می شود که سیاست های UCB و Gradiant Based همیشه به جواب بهینه همگرا شده اند حتی اگر در ابتدا با Actionای غیر بهینه شروع کرده باشد (سیاست UCB کمی بهتر از سیاست Gradiant Based عمل می کند)، و Epsilon Greedy تقریبا هیچ گاه به جواب بهینه همگرا نمی شود. حتی مشاهده می شود که گاهی اوقات که سیاست Epsilon Greedy در ابتدا تصادفا Action بهینه را انتخاب کرده بود، در ادامه به Action دیگری همگرا شده است. همچنین افزایش مقدار بتا یادگیری در سیاست Epsilon Greedy را شدیدا مختل می کند. (چندین بار این موارد بررسی شدند و همیشه نتایج همین بوده است)


alpha = 1, beta = 1, gamma = 1
EpsilonGreedy
GradiantBased
UCB
Utilities Average
Trial
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
0
200
400
600
800
1000


alpha = 1, beta = 1, gamma = 2
EpsilonGreedy
GradiantBased
UCB
Utilities Average
Trial
7
6
5
4
0
200
400
600
800
1000





از نظر سرعت همگرایی تفاوت معناداری در نمودارها قابل مشاهده نیست، ولی با افزایش مقدار آلفا و بتا می توان دید که Agentها دیرتر همگرا می شوند. احتمالا به این دلیل که بازه utilityهای ممکن را خیلی بازتر می کنند و همگرا شدن به میانگین این توزیع باز شده دیرتر انجام می شود. این مساله در حالت افزایش مقدار بتا نیز به میزانی کمتر (احتمالا چون Banditمان بیشتر پاداش مثبت تولید می کند

تا منفی) قابل مشاهده است (کمی شیب خطوط در ابتدای فرآیند افقی تر می شود). در الگوریتم UCB مشاهده می شود که با افزایش مقدار گاما یادگیری مقداری سریع تر انجام می شود.

قسمت دوم

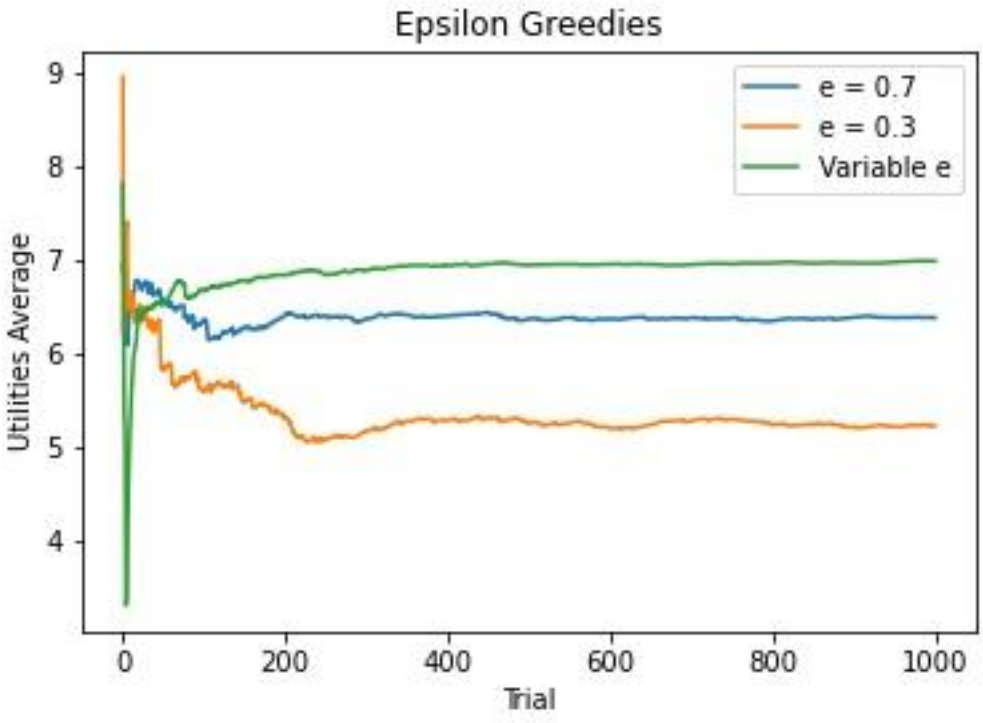
الف)

توضیح پیاده سازی

یک کلاس به نام VariableEpsilonGreedyPolicy پیاده سازی شده است که در آن اپسیلون در اولین trial برابر صفر است و با هر trial افزایش پیدا می کند تا در نهایت تقریبا برابر یک شود (فرمول محاسبه اپسیلون در آن به صورت $1 - \frac{1}{trial}$ است). متناظر با این کلاس، یک Agent که از این سیاست برای تصمیم گیری استفاده می کند نیز پیاده سازی شده است. در کنار این Agent، دو Agent دیگر با اپسیلون های ۰,۳ و ۰,۷ تمرین داده شده اند.

نتایج

مشاهده می شود که بهترین نتیجه مختص اپسیولن متغیر است و بدترین نتیجه مختص اپسیلون کوچک.



زمانی که اپسیلون کوچک باشد احتمال انتخاب Action هایی که تا اینجای یادگیری بهینه نبوده اند بیشتر می شود و این امر به Explore کرد بهتر فضا کمک می کند. ولی همین مساله در ادامه یادگیری باعث این می شود که این سیاست به جواب بهینه همگرا نشود و همیشه احتمال بالایی برای انتخاب Action های غیربهینه وجود داشته باشد. به همین دلیل میانگین پاداش دریافت شده در حالت اپسیلون کوچک کمتر از بقیه حالات است. در حالت اپسیلون بزرگ احتمال انتخاب Action غیربهینه کمتر است، در نتیجه میانگین پاداش دریافتی زیاد می شود ولی Exploration به نحو مناسبی صورت نمی پذیرد و لزوما جواب بهینه پیدا نمی شود. بنابراین بهتر است در ابتدای یادگیری که در فاز Exploration هستیم با اپسیلون کوچک شروع کنیم و تدریجا مقدار آن را افزایش دهیم.

ب)

۱) الگوریتم به این صورت است که ابتدا ضریبی برای آن که به چه کسی چه مقدار اطمینان کنیم قرار می دهیم شروع به بازی می کنیم و یک بار تمام Actionها را انتخاب می کنیم تا مقداری به عنوان Expected Reward آن Action لحاظ کنیم. از آن به بعد در هر مرحله، برای هر Action متغیری را با توجه به ضریب اطیمنان به باقی بازیکنان و تعداد دفعات انتخاب شدن Action توسط آن ها محاسبه می کنیم و با Expected Reward ای که از آن به دست آورده ایم جمع می کنیم. سپس Actionای که این مقدار برای آن بیشینه شود به عنوان Action بهینه انتخاب می کنیم[۱].

1: Initialization:

2: Set appropriate β, β_1, β_2, β_3. Set t = 1, $\bar{\mu}_t(i)=0$, $\forall i \in I$.

3: Repeat at the beginning of each time slot:

4: Let $I_t = t$, try action I_t, observe action I^{target}_t

5: Get reward $r_t(I_t)$, t = t + 1

6: Until t>NumberOfActions

7: Repeat at the beginning of each time slot

8: Update $N_t(i)$, $N_{1,t}(i)$, $N_{2,t}(i)$, $N_{3,t}(i)$

9: Update $\bar{\mu}_t(i)$ and $c^{OUCB}_t(i)$ as in (1)

10: Determine $I_t = \text{argmax}(\bar{\mu}_t(i) + c^{OUCB}_t(i))$, $i \in I$

11: Try action I_t, observe action I^{target}_t

12: Get reward $r_t(I_t)$, t = t + 1;

13: Until t>T;

در شبه کد بالا، β ضریب اطمینانمان به خودماناست و $\beta_1, \beta_2, \beta_3$ ضریب اطمینانمان به بازیکنان یک تا سه. $N_t(i)$ تعداد دفعاتیست که تا زمان t ما Action شماره i را انتخاب کرده ایم و تعداد دفعاتی که تا زمان t هر بازیکن دیگر Action شماره i را انتخاب کرده است $N_{1,t}(i), N_{2,t}(i), N_{3,t}(i)$ است که این مقادیر بعد از مشاهده اقدامات سایر بازیکنان به روز می شوند (خط ۸). $\bar{\mu}_t(i)$ متوسط پاداشی است که ما تا زمان t از انتخاب Action شماره i به دست آورده ایم. I^{target}_t نیز Actionای است که توسط بازیکنان دیگر انتخاب می شود (سه تا target داریم). نحوه محاسبه و به روز رسانی $\bar{\mu}_t(i)$ و $c^{OUCB}_t(i)$ به این صورت است:

$$\bar{\mu}_t = \frac{1}{N_t(i)} \sum_{j=1}^{t-1} r_j(i)(I_j == i)$$

$$c_t^{OUCB} = \sqrt{\frac{2\ln(t)}{N_t(i)}} (\beta + \beta_1 [\delta_{1,t}(i)]_+ + \beta_2 [\delta_{2,t}(i)]_+ + \beta_3 [\delta_{3,t}(i)]_+)$$

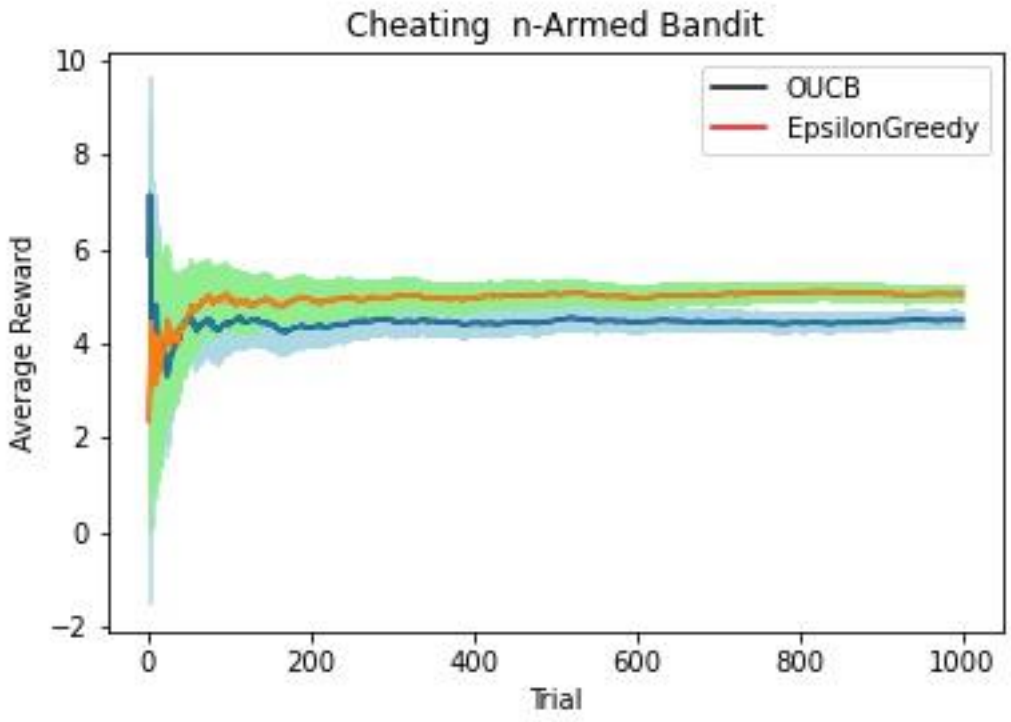
$$[\delta_{j,t}(i)]_+ = \left[\frac{N_{j,t}(i) - N_t(i)}{t}\right]_+$$

در شرایط این مساله، خوب است که با گذر زمان شخصی که سیاست تصادفی دارد را بیابیم و ضریب اطمینان او را صفر کنیم. احتمالا در طول بازی از بازیکنی که سیاست UCB دارد بیشتر بتوانیم استفاده کنیم، چون مطابق نتایجی که در قسمت های قبلی سوال به دست آمد این سیاست از سیاست Epsilon Greedy بهتر است.

۲) بهتر است مدتی زمان بگذرد. اگر مدتی زمان بگذرد بازیکنانی که سیاست های Epsilon Greedy و UCB دارند احتمالا توانسته اند میانگین پاداش دریافتی خودشان را بیشتر کنند به جواب بهینه نزدیکتر شده اند. همچنین ممکن است در صورت همگرا شدن این دو بازیکن به یک Action خاص بتوانیم تشخیص دهیم که کدام بازیکن بدون سیاست عمل می کند (از آنجا که به Action خاصی همگرا نمی شود و تقریبا تمام گزینه ها را انتخاب می کند.) و از دقت در رفتار او خودداری کنیم.

۳) در این حالت باید میزان اتکایی (ضرایب بتا) که به سایر بازیکنان داریم کمتر کنیم. می توان به مساله طوری نگاه کرد که انگار چشم های Mr.Nobody سالم است ولی دیگر بازیکنان Actionهایی انتخاب می کنند که نسبت به Action بهینه مقداری خطا دارد (لزوما همیشه Action بهینه را انتخاب نمی کنند).

۴) برای پیاده سازی این بخش، به کلاس های متناظر Agentهای UCB و EpsilonGreedy توابعی برای گرفتن جوایز دریافت شده اضافه شد. سپس کلاس جدیدی برای سیاست Random پیاده سازی شد. این کلاس، کلاس Policy متناظر ندارد چرا که منطق خاصی برای تصمیم گیری نیاز نیست. کلاس دیگری به نام OUCBPolicy برای الگوریتم توضیح داده شده پیاده سازی شد که کلاس دیگری به نام OUCBAgent از آن استفاده می کند. سپس، سه Agent برای بازیکنان دیگر و دو Agnet برای Mr.Nobody با شرایط و سیاست های تعریف شده در مساله تمرین داده شدند. که نتایج آن به این صورت است:



مشاهده می شود که الگوریتم پیاده سازی شده عملکرد ضعیف تری به نسبت سیاست Epsilon Greedy دارد. با توجه به نمودار و همچنین با توجه به اندازه بازه اطمینان ۹۵ درصد پاداش برای دو سیاست، می توان گفت اختلاف این دو سیاست نیز تا حدی معنادار است (بازه اطمینان برای سیاست OUCB بین ۴,۲۸ و ۴,۶۹ بود و برای EpsilonGreedy بین ۴,۸۴ و ۵,۲۴).

سوال ۲ - سوال تئوری

الف)

اگر واریانس پاداش بازوها را داشته باشیم می توانیم اندازه بازه Confidence Interval برای درصد اطمینان مشخص را محاسبه کنیم. در این صورت اگر متوسط پاداش یک بازو را داشته باشیم می توانیم بگوییم با احتمال مثلا ۹۵ درصد پاداش های این بازو در این بازه هستند. این عبارت همچنین این معنی را می دهد که اگر این بازو را انتخاب کنیم و پاداشی بگیریم، به احتمال ۹۵ درصد متوسط پاداش این بازو در بازه ای به همان اندازه (منظور اندازه بازه اطمینان است) و به مرکزیت آن مقدار پاداش است. بنابراین می توانیم شروع به بازی کنیم و از تمام بازو ها یک نمونه بگیریم و بازه هایی را حول پاداش های به دست آمده مشخص کنیم و بگوییم متوسط پاداش هر بازو با احتمال ۹۵ درصد در این بازه است. حال اگر دو بازه با هم اشتراک نداشته باشند، یعنی متوسط پاداش یکی شان با احتمال بسیار بالایی بیشتر از متوسط پاداش دیگریست. در این حالت بازوی با متوسط پاداش کمتر را حذف می کنیم و مجددا بازوهای دیگر را امتحان می کنیم. تدریجا و با دریافت نمونه های بیشتر از پاداش بازوها، مرکز بازه های اطمینان را به روز می کنیم و برابر متوسط پاداش به دست آمده قرار می دهیم. همچنین به این صورت احتمال این که متوسط پاداش در بازه هایی باشد که که چند بار داخل بازه های اطمینان نمونه های دریافت شده افتاده باشند افزایش پیدا می کنند می توانیم واریانس کمتری برای توزیع احتمال متوسط پاداش Action درنظر بگیریم (توجه شود که واریانس توزیع متوسط پاداش کم می شود و نه واریانس توزیع خود پاداش) که باعث کوچکتر شدن بازه های اطمینانِ گفته شده می شود. در نتیجه بازه های اطمینان متوسط پاداش Action هایی که متوسط پاداششان تفاوت معنایی با هم داشته باشد بیشتر از هم جدا می شود و می توانیم Action بهینه را انتخاب کنیم.

ب)

در صورتی که واریانس پاداش یک بازو با گذر زمان کاهش پیدا کند، تعداد دفعاتی که در آینده برای یاد گرفتن میانگین توزیع نیاز است تا آن بازو امتحان شود کمتر می شود. در نتیجه چه آن بازو Action بهینه باشد چه نباشد تکلیفش زود معلوم می شود و می توان گفت بسته به این که واریانس پاداش چند بازو کاهش پیدا کند سرعت یادگیری افزایش می یابد.

در صورتی که واریانس افزایش پیدا کند، می توان گفت یادگیری به مشکل می خورد چرا که با بازه بزرگتری به عنوان پاداش روبرو هستیم و همگرا شدن به مقداری به عنوان میانگین پاداش زمان بیشتری می طلبد. همچنین از آن جا که نرخ یادگیری تدریجا کم می شود احتمالا تاثیر خاصی از تغییرات و افزایش

واریانس نخواهیم گرفت، و در این صورت سیاستی مانند UCB برای تصمیم گیری به مشکل می خورد. چرا که با افزایش واریانس بازه اطمینان بزرگ می شود و مرز قبلی که برای UCB فرض کرده بودیم معتبر نیست و احتمال دریافت پاداشی که از مقدار خاصی بزرگتر باشد کمتر می شود. (با زیاد شدن واریانس توزیع بیشتر به خط راست نزدیک می شود. احتمال های بالای مقادیر نزدیک میانگین کم و احتمال های پایین مقادیر دور از میانگین زیاد می شوند)

سوال ۳ - سوال پیاده‌سازی

هدف سوال

هدف از این سوال پیاده سازی الگوریتم Learning Comparison برای پیدا کردن بهترین گزینه از بین گزینه های پیش رو برای لباس است.

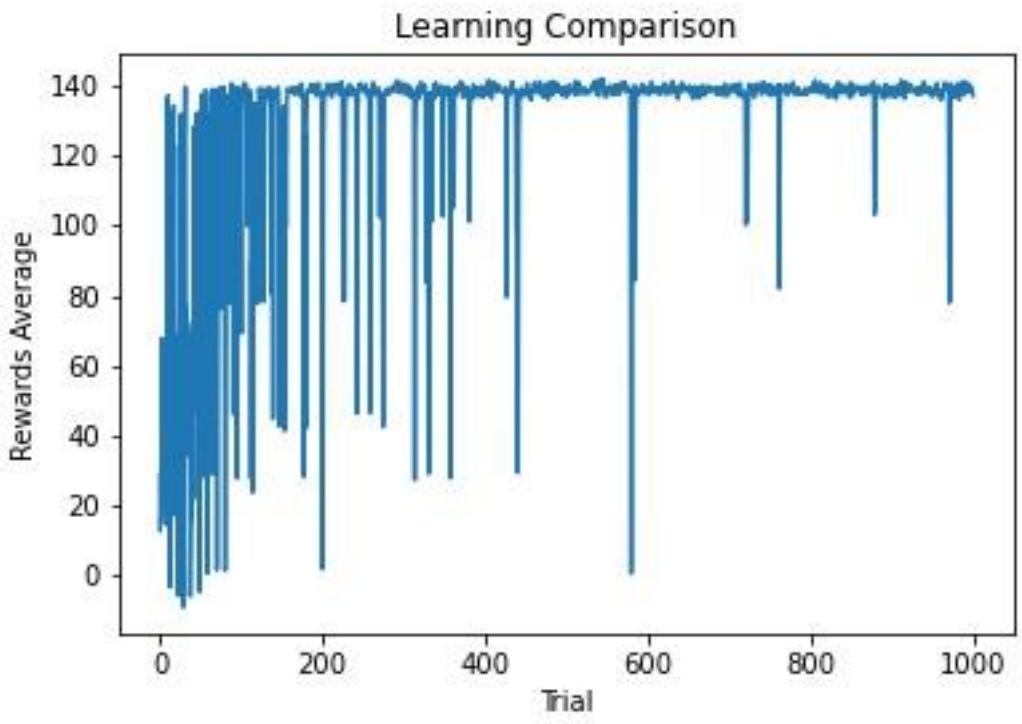
الف)

توضیح پیاده سازی

سیاست Learning Comparison داخل کلاسی به نام LearningComparisonPolicy پیاده سازی شده است و این کلاس در کلاس LearningComparisonAgent مورد استفاده قرار می گیرد.

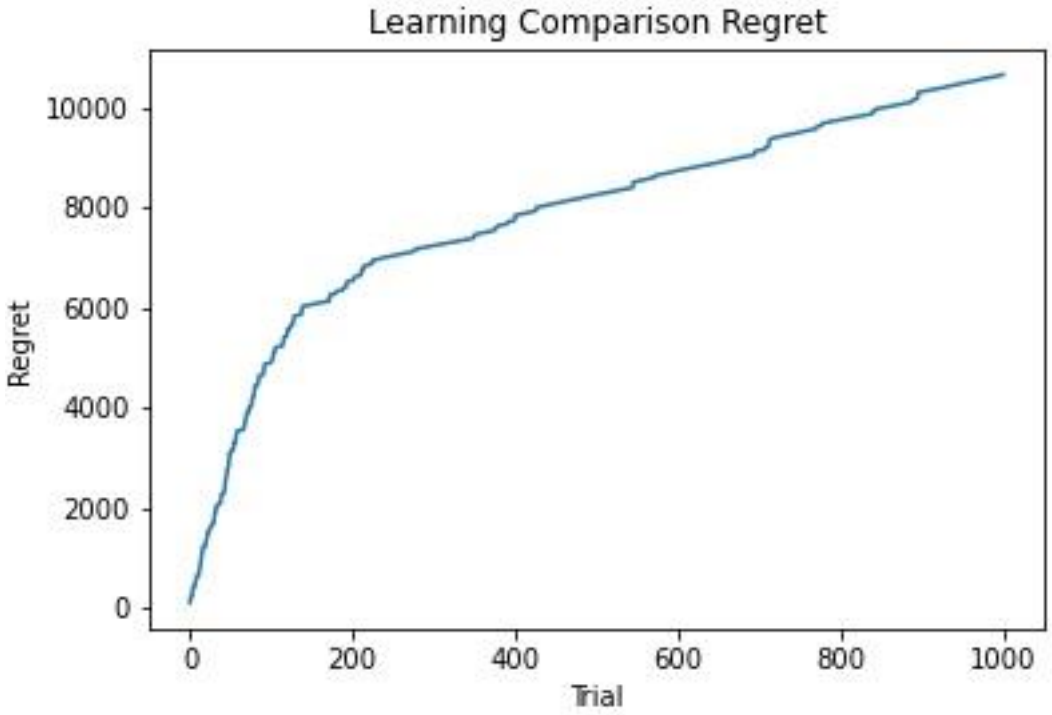
نتایج

مشاهده می شود که بهترین لباس لباس دوم آدیداس است که Action دوم (Actionها از صفر شمرده می شوند) n-Armed Bandit است. با توجه به نمودار به تدریج احتمال انتخاب Actionهای غیر بهینه کاهش پیدا می کند و Action بهینه بیشتر انتخاب می شود.

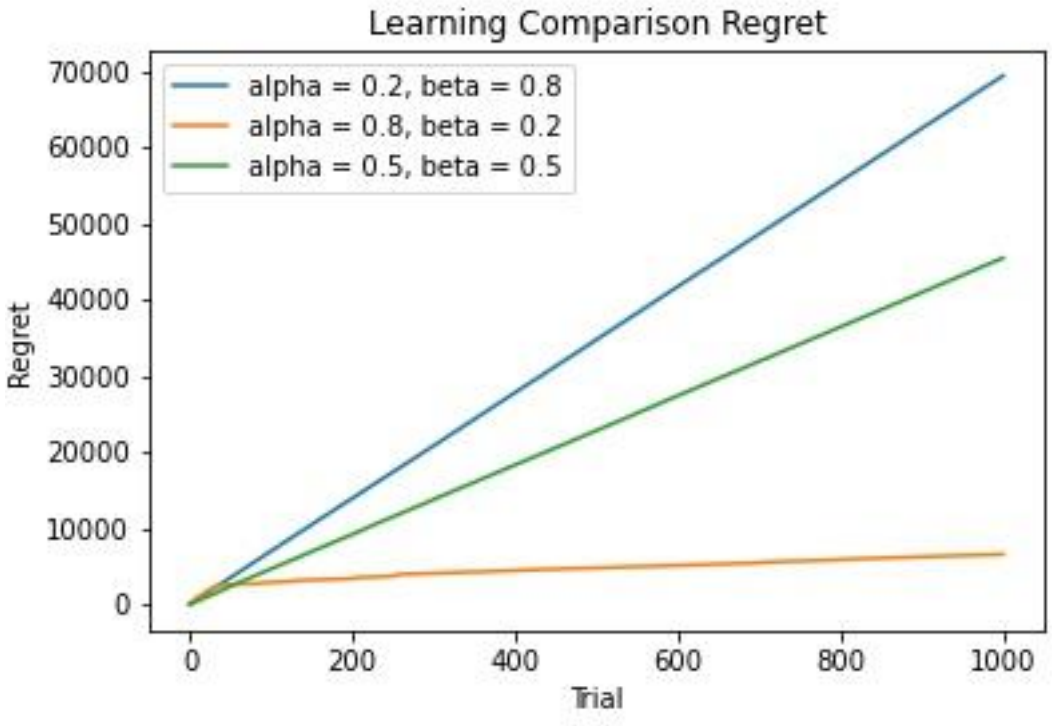


ب)

از آن جا که به تدریج تعداد دفعات Action بهینه زیاد می شود، پشیمانی نیز به همان شکل کاهش پیدا می کند (مقدار بیشینه پاداش را برابر بیشترین پاداش موجود برای لباس دوم آدیداس قرار دادیم. علت این که به جای میانگین از بیشینه استفاده کردیم این بود که در آن صورت در صورت دریافت پاداش بیش از میانگین مقدار پشیمانی منفی می شد). در نتیجه نمودار پشیمانی کل تدریجا افقی می شود.



برای مقایسه بین حالات استفاده از مقادیر مختلف آلفا و بتا، سه نمونه از Agent طراحی شده در این قسمت با مقادیر آلفا و بتا به ترتیب برابر ۰۲ و ۰٫۸، ۰٫۸ و ۰٫۲، و ۰٫۵ و ۰٫۵ تمرین داده شدند.



با توجه به نمودار، هرچه مقدار آلفا بیشتر و مقدار بتا کمتر شود، شیب نمودار پشیمانی افقی تر و مجموع مقدار پشیمانی کمتر است. این به آن معناست که با این مقادیر یادگیری بهتر و سریع تر انجام شده است.

ج)

با توجه به مساله پاسخ سوال تفاوت می کند. در صورتی که اختلاف Action بهینه با باقی Actionها خیلی زیاد باشد، سیاست های حریصانه (در صورتی که آنقدر حریصانه نباشند که صرفا روی همان Action اولی که انتخاب کردند بمانند و حداقل یک بار تمام Actionها را امتحان کنند) می توانند به جواب بهینه همگرا شوند و این همگرایی هم زودتر از سایر سیاست ها اتفاق می افتد. در غیر این صورت سیاست های حریصانه به دلیل این که خیلی خوب فضا را Explore نمی کنند ممکن است نتوانند بهترین Action را پیدا کنند.

در این مساله نیز اختلاف Action بهینه با باقی Actionها خیلی زیاد است. در صورتی که چند بار هر کدام را انتخاب کنیم و بفهمیم که کدام Action اختلاف فاحشی با بقیه دارد می توانیم از آنجا به بعد حریصانه عمل کنیم و بیشترین پاداش ممکن را دریافت کنیم.

منابع

1- Ieeexplore.ieee.org. 2021. *Social Bandit Learning: Strangers Can Help*. [online] Available at: <https://ieeexplore.ieee.org/document/9299725> [Accessed 9 November 2021].