پاسخ تکلیف دوم

"یادگیری تعاملی"

گروه دوم

نام استاد:

دکتر نیلی

تهیه کننده :

عرفان میرزایی ۸۱۰۱۹۹۲۸۹

تکلیف دوم- یادگیری تعاملی
عرفان میرزایی
۸۱۰۱۹۹۲۸۹

پرسش ۱: Thompson Sampling

ابتدا مانند شکل ۱-۲ کتاب ساتون-بارتو دسته های بندیت را پیاده سازی کردیم؛ بدین صورت که هر دسته از بندیت دارای پاداشی با توزیع گاوسی و با انحراف معیار یک است که میانگین توزیع هر دسته نیز توسط نمونه گیری از توزیع نرمال استاندارد بدست آمده است. که همان طور که از شکل زیر مشخص است عمل ۳ بهینه است.

```
[[-0.41675785],
 [-0.05626683],
 [-2.1361961 ],
 [ 1.64027081],
 [-1.79343559],
 [-0.84174737],
 [ 0.50288142],
 [-1.24528809],
 [-1.05795222],
 [-0.90900761]]
```

شکل ۱- مقدار متوسط پاداش هر دسته

سپس با استفاده از کلاس GaussianReward پکیج amalearn پاداش هر یک از ۱۰ دسته را تعریف کردیم. هم چنین محیط MultiArmedBanditEnvironment که در پکیجamalearn موجود است را به عنوان محیط در نظر گرفتیم.

در قسمت بعد ThompsonSamplingAgent را پیاده سازی کردیم که کد مربوط به آن در فایل Thompson_sampling_agent.py در پیوست موجود است.

به طور خلاصه این عامل در ابتدا برای هر دسته از بندیت یک توزیع گوسی تخمینی با میانگین و انحراف معیار دلخواه در نظر میگیرد و در هر مرحله با نمونه گیری از این توزیع های تخمینی، عمل با بزرگترین پاداش دریافتی از بین نمونه ها را انتخاب می کند و آن عمل را انجام می دهد و پاداش واقعی را از محیط دریافت می‌کند و توزیع تخمینی مربوط به آن عمل را با توجه به پاداش واقعی دریافتی به روز رسانی می‌کند.

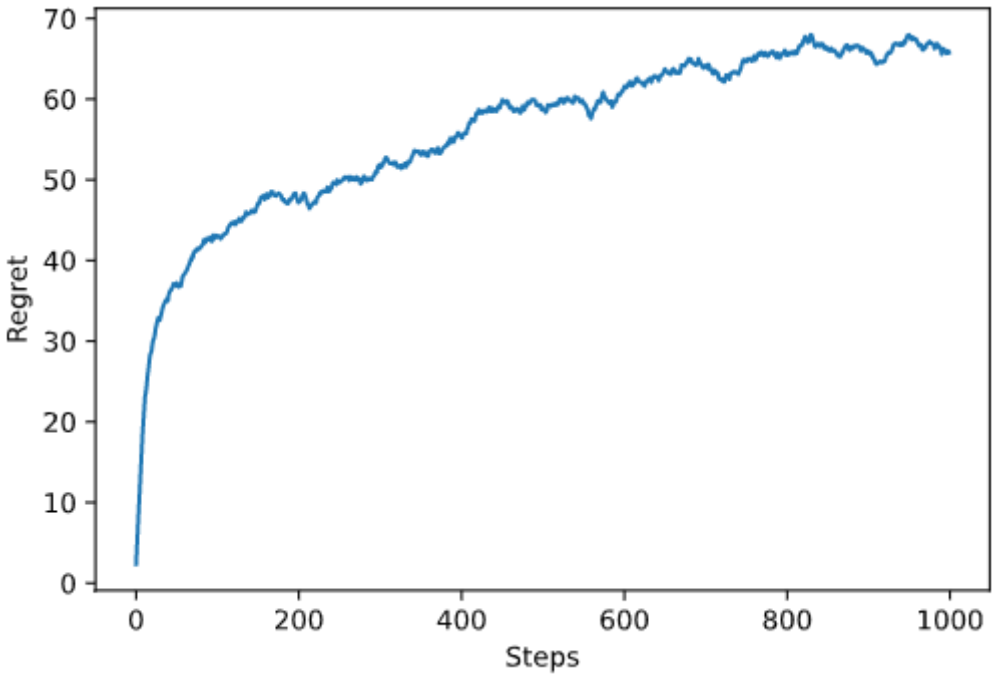
مقدار اولیه میانگین و انحراف معیار توزیع تخمینی برای هر عمل را به ترتیب صفر و یک میلیارد در نظر گرفتیم که در واقع این توزیع تقریبی از توزیع یکنواخت است و برای زمانی مناسب است که دانش پیشینی در مورد پاداش هر دسته نداریم.

یکی از نکات پیاده‌سازی در به روز رسانی توزیع عمل انتخاب شده توسط عامل در هر مرحله است. برای این کار با توجه به دانستن توزیع پاداش و هم چنین انحراف معیار آن، از روابط زیر[1] برای به روز رسانی میانگین و انحراف معیار توزیع تخمینی استفاده کردیم :

Likelihood	Model parameters	Conjugate prior distribution	Prior hyperparameters	Posterior hyperparameters[note 1]
Normal with known variance σ^2	μ (mean)	Normal	$\mu_0,\ \sigma_0^2$	$\frac{1}{\frac{1}{\sigma_0^2}+\frac{n}{\sigma^2}}\left(\frac{\mu_0}{\sigma_0^2}+\frac{\sum_{i=1}^{n} x_i}{\sigma^2}\right),\left(\frac{1}{\sigma_0^2}+\frac{n}{\sigma^2}\right)^{-1}$

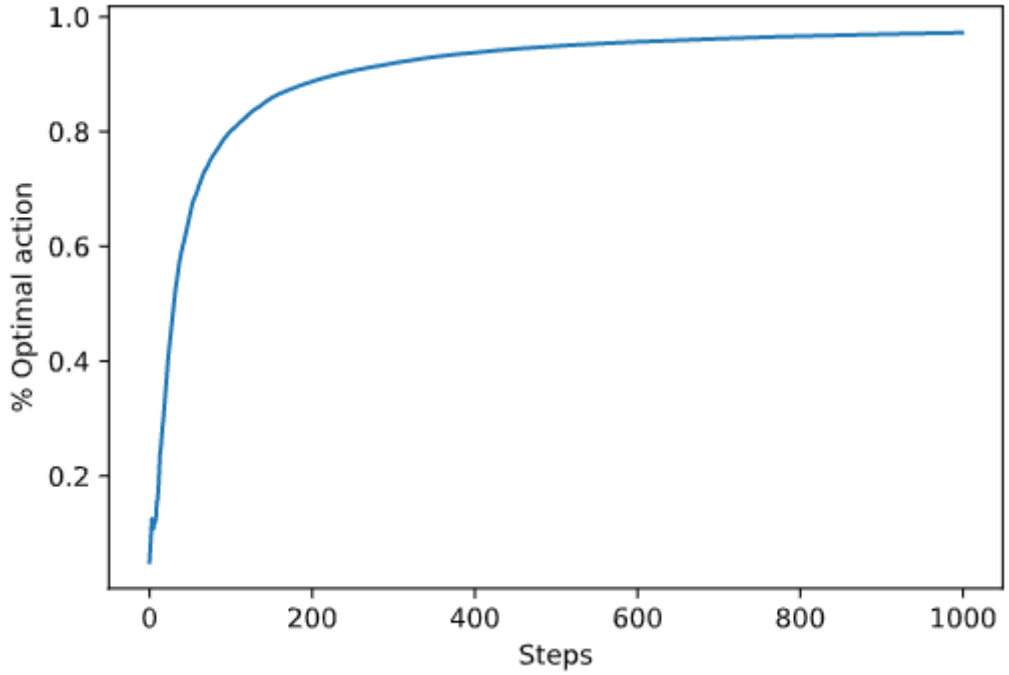
شکل ۲- به روز رسانی میانگین و انحراف معیار توزیع تخمینی عمل ها

الگوریتم را به اندازه ی ۱۰۰۰ گام برای ۲۰ بار انجام داده و نتایج بدست آمده را میانگین گرفته که حاصل آن ها به صورت زیر است.

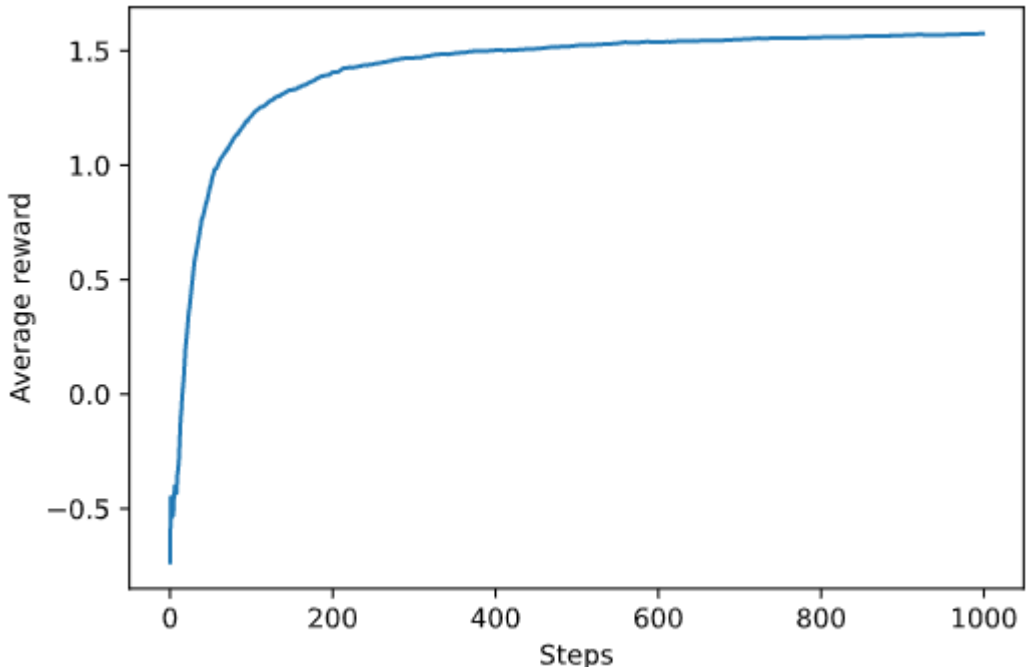


شکل ۳- نمودار میانگین حسرت بر اساس گام ها با اجرای الگوریتم Thompson Sampling

[1] .https://en.wikipedia.org/wiki/Conjugate_prior#When_likelihood_function_is_a_continuous_distribution



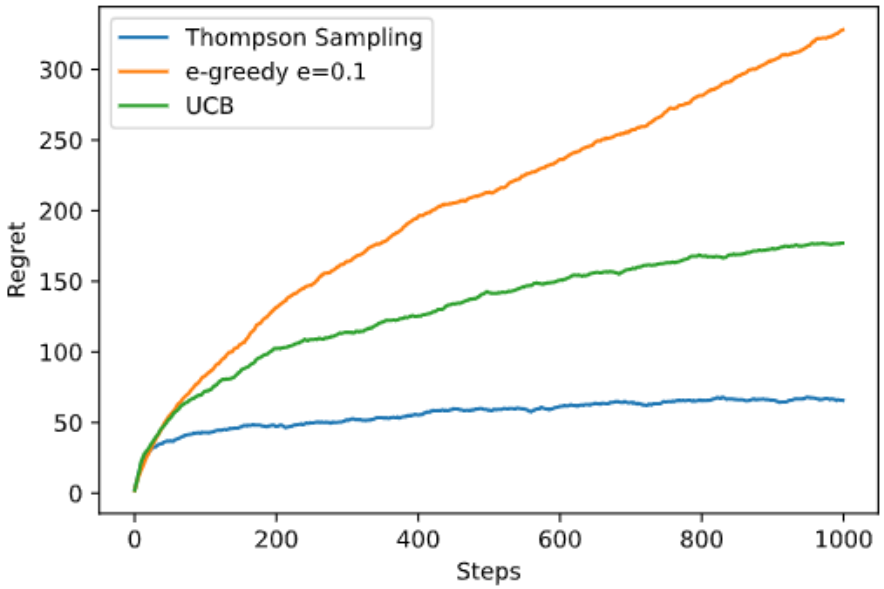
شکل ۴- نمودار میانگین درصد انجام عمل بهینه بر اساس گام ها با اجرای الگوریتم Thompson Sampling



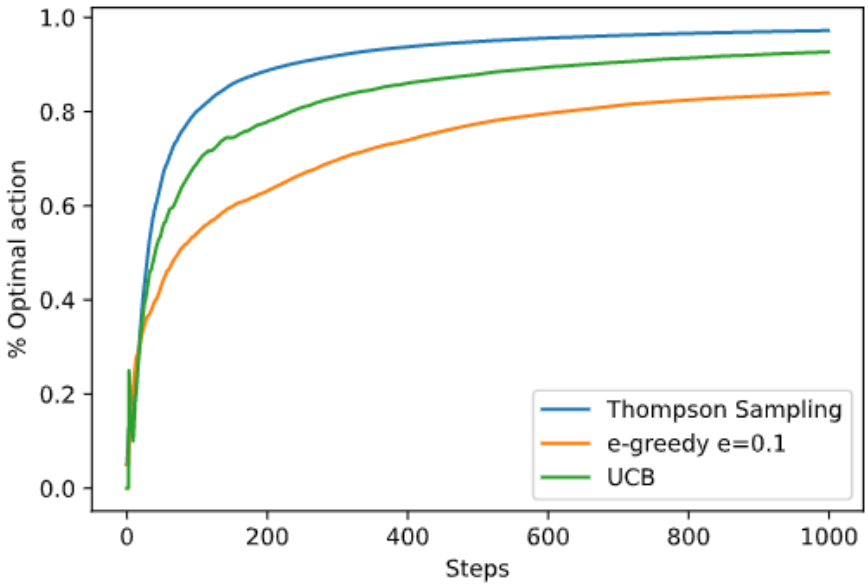
شکل ۵- نمودار میانگین پاداش بر اساس گام ها با اجرای الگوریتم Thompson Sampling

همان طور که از نمودار ها مشخص عامل عملکرد بسیار خوبی در این محیط داشته و به سرعت عمل بهینه را پیدا کرده و پس از آن، به طور مداوم آن را انجام می‌دهد.

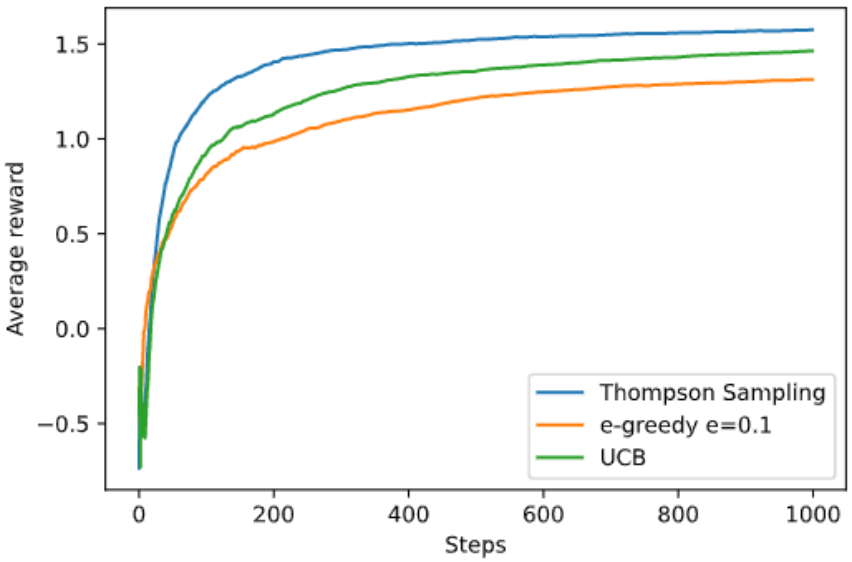
برای مقایسه عملکرد عامل با سایر عامل ها، عامل Epsilon-greedy e =0.1 , UCB (exploration degree = 2) در نظر گرفته شده است. که مقایسه آن ها به شرح زیر است :



شکل ۷ - نمودار میانگین حسرت بر اساس گام های سه الگوریتم مختلف



شکل ۶- نمودار میانگین انجام عمل بهینه بر اساس گام های سه الگوریتم مختلف



شکل ۸- نمودار میانگین پاداش متوسط بر اساس گام های سه الگوریتم مختلف

که همان طور که در نمودار ها نیز مشخص عامل Thompson Sampling بهترین عملکرد را داشته و زودتر از سایر عامل ها موفق شده عمل بهینه را پیدا کند و تعداد دفعات بیشتری آن را انجام دهد. هم چنین همان طور که از نمودار حسرت نیز مشخص دو عامل Thompson Sampling , UCB دارای حسرت به صورت لگاریتمی هستند که در این زمینه نیز برتری با Thompson Sampling است.

پاسخ پرسش ۲ :

الف) از آنجا که برای مدلسازی مسئله می‌بایست آن را در قالب n-armed bandit پیاده سازی کنیم بنابراین نمی‌توانستیم چند حالت را در نظر بگیریم و مسئله ما تنها یک حالت دارد. برای این کار، ما اعمال را طوری طراحی کردیم که پس از انتخاب هر عمل مسئله پایان یابد و بنابراین عامل بتواند زمان بهینه صبر کردن در صف را یاد بگیرد و هم چنین با محدودیت تک حالته بودن مسئله نیز سازگار باشد.

مجموعه اعمال ما شامل ۱۶ عضو است که هر کدام از این عمل ها بدین ترتیب است که به اندازه‌ی شماره‌ی عمل دقیقه برای رسیدن اتوبوس در صف منتظر می‌ماند اگر پیش از تمام شدن مدت انتظار اتوبوس آمد با اتوبوس به دانشکده می رود و متناسب با مدتی که حدس زده بود باید در صف بایستد پاداش منفی دریافت می‌کند و اگر اتوبوس تا زمان مقرر نیامد، عامل سوار تاکسی می‌شود و جریمه ای متناسب با کرایه تاکسی و هم‌چنین میزان زمان منتظر ماندن در صف دریافت می‌کند.

اعمال از شماره ۰ تا شماره ۱۵ هستند که برای مثال عمل شماره‌ی ۵ بدین معناست که عامل ۵ دقیقه در صف منتظر رسیدن اتوبوس می‌ماند اگر اتوبوس تا قبل از ۵ دقیقه رسید که سوار اتوبوس شده و با آن به دانشکده می‌رود و جریمه ای متناسب با ۵ دقیقه صبر کردن دریافت می‌کند و در غیر این‌صورت سوار تاکسی می‌شود و علاوه بر جریمه‌ی منتظر ماندن، جریمه‌ی هزینه‌ی تاکسی نیز به آن اضافه می شود.

توجه : در این مثال حتی اگر اتوبوس در دقیقه اول نیز برسد، عامل به اندازه‌ی ۵ دقیقه صبر کردن جریمه می‌شود که این امر بدان علت است که عامل سعی کند زمان بهینه صبر کردن را بیاموزد و هر دفعه حدس های دقیق تری را ارائه بدهد. در واقع اگر این کار صورت نمی‌گرفت عامل زمان بهینه صبر کردن را نمی‌آموخت و همواره عمل شماره ۱۵ را انتخاب می کرد.

اما در مورد تابع فایده، می‌دانیم که انسان ها دارای ویژگی های شناختی متفاوتی هستند و اتفاقات و پاداش های بیرونی ارزش های درونی متفاوتی برای هر شخص دارد که این ارزش ها متناسب با ویژگی های شناختی فرد هستند که بخشی از این ویژگی ها مشترک میان انسان هاست و بخشی دیگر از آن در بین انسان ها متفاوت است.

در این مثال، ما از مدل ارائه شده در مقاله‌ی Prospect Theory برای مدلسازی تابع فایده استفاده کردیم. که فرم کلی آن به صورت زیر است :

$$v(x) = \begin{cases} x^{\alpha} & x \geq 0 \\ -\lambda(-x)^{\beta} & x < 0 \end{cases}$$

که در این معادله x مقدار پاداشی است که از محیط دریافت می‌کنیم و آلفا، بتا و گاما پارامترهای مسئله‌ی ما هستند که متناسب با ویژگی های شناختی افراد می‌توانند متفاوت باشند.

برای مثال در این مسئله این پارامتر ها به طور مشخص با میزان درآمد هر فرد و هم‌چنین میزان صبور بودن آن شخص متناسب است.

ب) در این بخش برای پیاده‌سازی مدل تشریح شده در قسمت قبل، ابتدا پاداش هر عمل را پیاده سازی کردیم. بدین صورت که عامل به ازای هر دقیقه‌ای که پیش‌بینی کرده باید منتظر بماند، جریمه‌ای ثابت دریافت می‌کند(در هر دو صورت) و اگر این پیش‌بینی درست نباشد مقدار بزرگی جریمه می‌شود که این جریمه به علت اشتباه بودن پیش بینی و هم‌چنین هزینه‌ی اضافی تاکسی گرفتن است.

در قسمت بعدی از آنجا که توزیع زمانی رسیدن اتوبوس را می‌دانیم مقدار متوسط پاداش هر عمل را حساب می‌کنیم تا عمل بهینه را پیدا کنیم. همان طور که مشخص است نسبت جریمه‌ای که برای انتظار دریافت می‌کنیم به جریمه‌ی اشتباه بودن پیش‌بینی در تعیین عمل بهینه بسیار موثر است و همان طور که در قسمت پیش نیز در مورد آن بحث شد این نسبت می‌تواند به طور مستقیم به ویژگی های شناختی هر فرد وابسته باشد که برای سادگی کار از مدلسازی این امر به دلیل عدم وجود اطلاعات لازم صرف نظر شده است.

```
12

[-4980.848097162051,
 -5250.923356856773,
 -5486.249340259104,
 -5661.048238635926,
 -5743.943901370661,
 -5706.7237303427155,
 -5537.5373122653855,
 -5252.793299091182,
 -4900.0,
 -4547.206700908819,
 -4262.4626877346145,
 -4093.2762696572854,
 -4056.0560986293394,
 -4138.951761364074,
 -4313.7506597408965,
 -4549.076643143227]
```

شکل ۹- مقدار متوسط هر یک از عمل ها با توجه به توزیع رسیدن اتوبوس و میزان جریمه برای انتظار و تاکسی

در گام بعدی محیط EnghelabEnvironment پیاده سازی شده است که این کلاس از محیط EnvironmentBase پکیج amalearn ارث‌بری کرده است. کد آن تا حدی مشابه محیط MultiArmedBandit است که در پکیج amalearn پیاده سازی شده است. با هر بار اجرای این محیط مقدار تصادفی با توجه به توزیع آمدن اتوبوس در نظر گرفته می‌شود که با توجه به این عدد و مقایسه‌ی آن با عمل انتخاب شده توسط عامل پاداش متناظر به عامل داده می‌شود.(calculate_reward Method)

از آنجا که با انجام هر عمل مسئله پایان می‌یابد در متد terminated این نکته در نظر گرفته شده است. هم چنین برای اطلاع از آن که اتوبوس پیش از پایان مهلت انتظار رسیده است یا خیر نیز، در متد get_info در نظر گرفته شده است. هم چنین در متد reset نیز هر بار عدد تصادفی رسیدن اتوبوس تغییر می‌کند.

پ) در این بخش ابتدا دو عامل با سیاست های Epsilon Greedy و UCB پیاده سازی شد. هر کدام از این عامل ها مشخصات شناختی برای توصیف تابع فایده‌ی خود را علاوه بر سایر پارامترهای لازم برای تصمیم‌گیری دریافت می‌کنند.

عامل با سیاست epsilon-greedy مقدار epsilon و هم چنین epsilon_decay_rate را به عنوان هایپر پارامتر دریافت می‌کند. این عامل در هر مرحله، با احتمال epsilon عملی تصادفی را انتخاب می‌کند و در غیر این صورت عمل با بیشترین میانگین پاداش دریافتی را انتخاب می‌کند. هم چنین این عامل در هر مرحله با دریافت پاداش از محیط باور خود را نسبت به میانگین پاداش اعمال به روز رسانی می‌کند. مقدار epsilon نیز در هر مرحله به روز رسانی می‌شود.

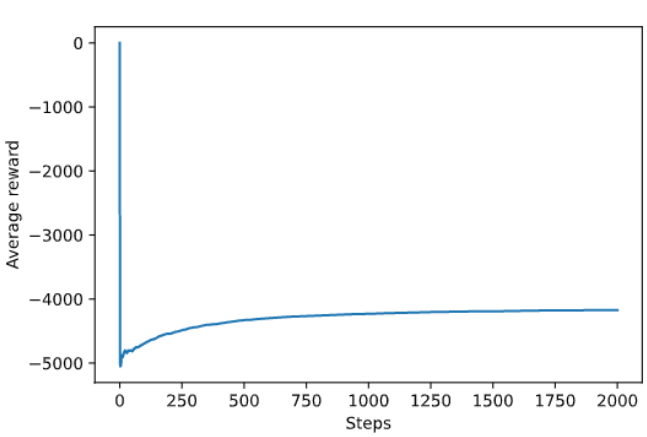
عامل با سیاست UCB مقدار exploration_degree را به عنوان هایپر پارامتر دریافت می‌کند و در هر مرحله با توجه به باور خود نسبت به میزان حد بالای بازه‌ی اطمینان هر عمل، عمل با بیشترین میزان حد بالای اطمینان را انتخاب می‌کند. پس از انجام هر عمل، با توجه به پاداش دریافتی باور خود را نسبت به حد بالای بازه‌ی اطمینان هر عمل به روز رسانی می‌کند که برای این کار از رابطه ی زیر استفاده می‌کند:

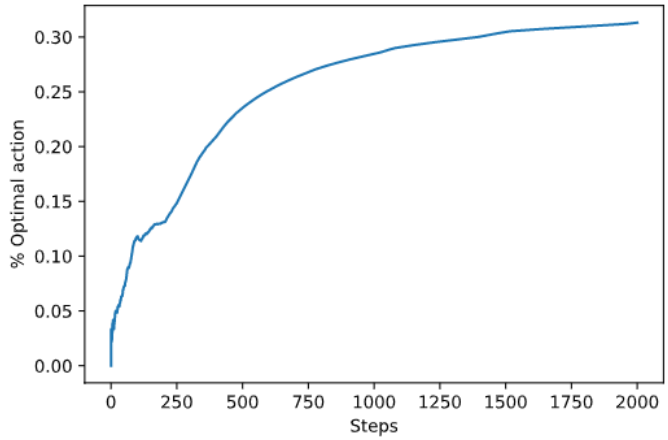
$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

پس از پیاده‌سازی عامل ها و هم‌چنین محیط، الگوریتم ها را اجرا می‌کنیم و نتایج حاصله را با هم مقایسه می‌کنیم.

در هر دو عامل مقدار آلفا و بتا برابر با ۰,۸۸ و مقدار گاما برابر با ۲,۲۵ در نظر گرفته شده است که این مقادیر از نتایج بدست آمده از مقاله Prospect Theory بدست آمده است.

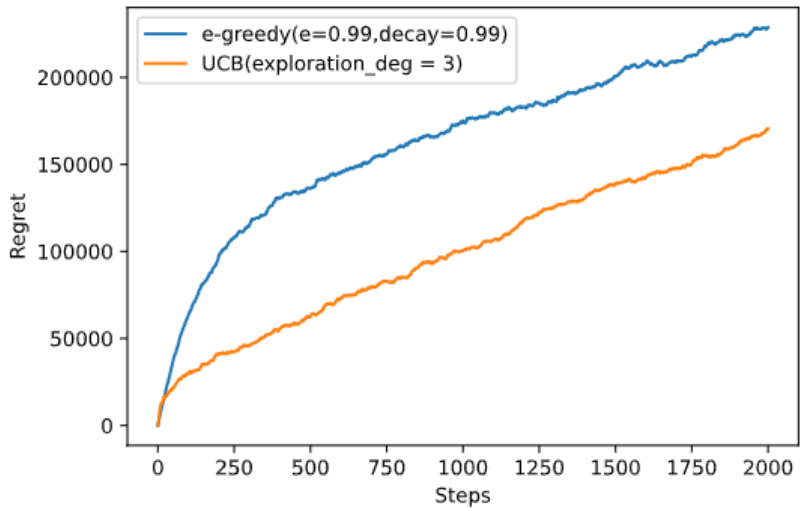
نتایج بدست آمده از میانگین ۳۰ بار اجرای ۲۰۰۰ گام برای عامل با سیاست epsilon greedy با مقدار epsilon = 0.99 و epsilon_decay_rate = 0.99 به شرح زیر است:

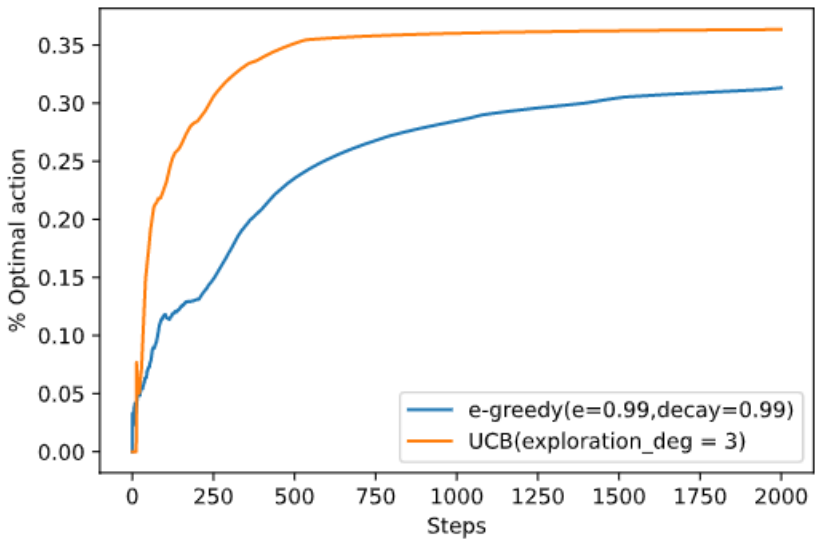


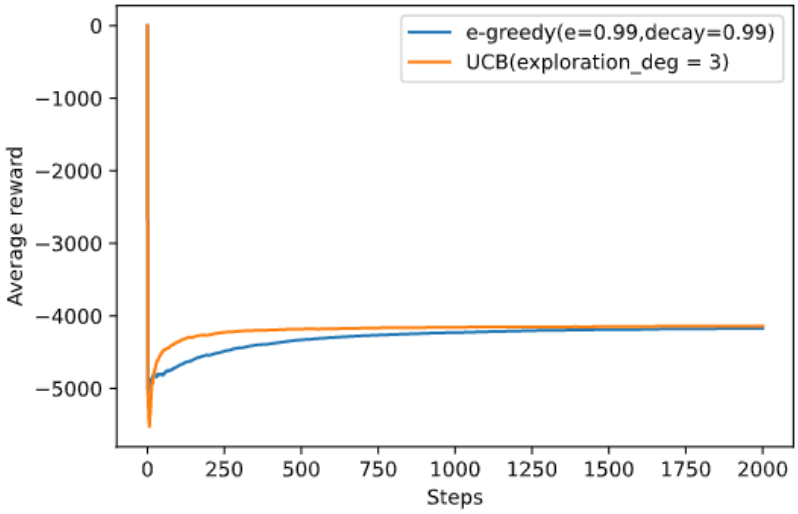


همان طور که از نمودار درصد انتخاب عمل بهینه نیز مشخص است، عامل در بیش از نیمی از موارد نتوانسته عمل بهینه را پیدا کند و با انجام بررسی ها مشخص شد که در این موارد عامل دومین و یا سومین عمل بهینه را انتخاب کرده که این اتفاق با توجه به مقدار پاداش ها که در شکل ۹ موجود است طبیعی به نظر می‌رسد، زیرا ما در واقعیت با یک مجموعه‌ی اعمال پیوسته روبرو بودیم که برای راحتی، آن را گسسته سازی کردیم و برای مثال اگر اتوبوس در زمان ۱۲ دقیقه و یک ثانیه بیاید مدل ما آن را در نظر نمی‌گیرد.

با توجه به تصادفی بودن الگوریتم ها و زمان رسیدن اتوبوس، با اجرای هر کدام از الگوریتم ها برای ۳۰ بار و میانگین گرفتن از نتایج بدست آمده، خواهیم داشت :







با توجه به نتایج بدست آمده مشهود است که مدل UCB تعداد دفعات بیشتری موفق شده است که عمل بهینه را انتخاب کند و هم چنین با توجه به نمودار حسرت و هم چنین میانگین پاداش دریافتی، مشهود است که عامل UCB عملکرد بهتری داشته است. که این بدین معناست که این عامل بسیار سریع اعمال با پاداش نزدیک به بهینه را پیدا می کند بنابراین حسرت کمتر و میانگین پاداش های دریافتی بیشتری دارد که این بدین خاطر است که در این عامل علاوه بر تخمین نقطه ای از متوسط پاداش ها به انحراف معیار پاداش ها نیز توجه می‌شود و با احتساب یک بازه‌ی اطمینان عمل مناسب را انتخاب می‌کند.

هم چنین از دیگر تفاوت های این دو مدل می‌توان به این مسئله اشاره کرد که مدل epsilon-greedy به دو هایپر پارامتر برای برقراری exploration-exploitation balance نیاز دارد اما در مدلی از UCB که ما پیاده سازی کرده ایم به تنها یک هایپر پارامتر احتیاج است.

پاسخ پرسش ۳:

الف) برای تبدیل این مسئله به فرم n-armed-bandit، با توجه به اینکه در این فرم از مسائل تنها یک حالت وجود دارد می‌بایست تمام مسیرهای ممکن را از حالت اولیه در نظر بگیریم و در واقع در همان شروع ماجرا در مورد کل مسیر تصمیم گیری کنیم. با توجه به شکل موجود در صورت سوال، برای انتقال پیام از گره صفر به گره دوازده ۴۸ مسیر وجود دارد بنابراین عامل ما ۴۸ عمل دارد.

در شکل زیر هر یک از این مسیر ها یا در واقع عمل ها آورده شده است :

```
actions = [ '0-1-5-8-12','0-1-5-9-12','0-1-5-10-12','0-1-5-11-12','0-1-6-8-12','0-1-6-9-12','0-1-6-10-12','0-1-6-11-12',
            '0-1-7-8-12','0-1-7-9-12','0-1-7-10-12','0-1-7-11-12','0-2-5-8-12','0-2-5-9-12','0-2-5-10-12','0-2-5-11-12',
            '0-2-6-8-12','0-2-6-9-12','0-2-6-10-12','0-2-6-11-12','0-2-7-8-12','0-2-7-9-12','0-2-7-10-12','0-2-7-11-12',
            '0-3-5-8-12','0-3-5-9-12','0-3-5-10-12','0-3-5-11-12','0-3-6-8-12','0-3-6-9-12','0-3-6-10-12','0-3-6-11-12',
            '0-3-7-8-12','0-3-7-9-12','0-3-7-10-12','0-3-7-11-12','0-4-5-8-12','0-4-5-9-12','0-4-5-10-12','0-4-5-11-12',
            '0-4-6-8-12','0-4-6-9-12','0-4-6-10-12','0-4-6-11-12','0-4-7-8-12','0-4-7-9-12','0-4-7-10-12','0-4-7-11-12']
```

پاداش های هر یک از مسیرها، از ترکیب چهار توزیع گوسی که میانگین و انحراف معیار آن متناسب با شماره دانشجویی و هم چنین رنگ هر لینک مشخص می شود. که شرح زیر است :

توزیع احتمالی	رنگ لینک
GaussianReward(9,8.2)	آبی
GaussianReward(8,3)	سبز
GaussianReward(2,9.5)	نارنجی

و البته سه توزیع برنولی که مقدار هر یک از این پاداش ها با توجه به شماره گره ها طبق جدول زیر محاسبه می‌شود :

پارامتر p	شماره گره
0.10	۱
0.06	۲
0.15	۳
0.50	۴
0.10	۵
0.15	۶
0.65	۷
0.12	۸
0.20	۹
0.05	۱۰
0.45	۱۱

برای پیاده‌سازی پاداش هر یک از مسیر ها ابتدا باید پاداشی با توزیع برنولی را پیاده‌سازی کنیم. برای این کار کلاس BernoulliReward را پیاده سازی کردیم که از RewardBase ارث بری می‌کند و در هر مرحله با توجه به پارامتر توزیع برنولی و هم چنین میزان پاداش، پاداش مورد انتظار را باز می‌گرداند.

در این مسئله به دلیل نداشتن تعریف صریحی از پاداش ها، پاداش هر یک از مسیرها به صورت دستی نوشته شده است. از آنجا که پاداش هر مسیر دارای چندین توزیع تصادفی با میانگین ها و انحراف معیارهای گوناگون است، مسیر بهینه برای ما نیز مشخص نیست. برای تعیین مسیر بهینه، باید تعداد زیادی نمونه هر یک از مسیرها بگیریم و میانگین این نمونه ها را به عنوان پاداش در نظر بگیریم. که برای این کار عاملی را پیاده سازی کردیم که در هر مرحله یک عمل مشخص و تکراری را انجام می‌دهد. این عامل به نام OneActionAgent پیاده سازی شده است که عملی که باید متوالیا آن را انجام دهد از ورودی دریافت و می‌کند و میانگین آن را نگه می‌دارد و هر بار به روز رسانی می‌کند.

پس از اجرای هر عمل برای ۱۰۰۰ مرحله و بدست آوردن میانگین تاخیر های هر مسیر یا به عبارتی دیگر میزان متوسط پاداش های دریافتی با انتخاب هر عمل به صورت زیر است :

```
[-30.41128373, -23.33668292, -21.74507382, -25.96180196]
[-31.88612693, -24.91844466, -24.43018288, -28.53970898]
[-36.50915233, -29.80285057, -28.8901718 , -32.97446578]
[-30.49857437, -24.2400073 , -22.96471099, -28.50221627]
[-32.13325546, -27.248522  , -25.0959228 , -28.20501476]
[-37.66590926, -31.18264038, -30.0384856 , -34.99571168]
[-29.97767965, -24.2504769 , -22.23753307, -26.81750236]
[-31.94576014, -26.33050348, -24.7184243 , -28.45596885]
[-37.87454666, -32.05474948, -29.32366626, -33.97261621]
[-28.88836321, -22.02099209, -21.14487831, -23.67592643]
[-29.71091626, -23.77491427, -22.22208717, -26.33003444]
[-34.96102828, -28.70579671, -27.21292231, -30.71392729]
```

که مسیر ۳۸ با متوسط ۲۱,۱۴۴- مسیر بهینه است که از گره های ۰-۴-۵-۹-۱۲ می‌گذرد. نکته ای که در شکل بالا نیز قابل توجه است نزدیک بودن فاصله‌ی میان تاخیر مسیر های مختلف است. که این نزدیک بودن پیدا کردن مسیر بهینه را دشوار می‌کند.

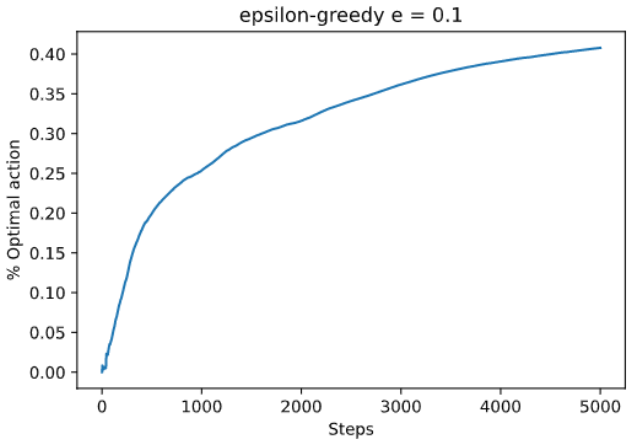
در این سوال نیز مانند سوال قبل از تابع فایده ای استفاده می‌کنیم که در مقاله‌ی Prospect Theory پیشنهاد شده است. به این ترتیب که برای آلفا و بتای کوچکتر از یک و هم چنین گامای بزرگتر از یک عامل های ما رفتاری loss averse دارند.

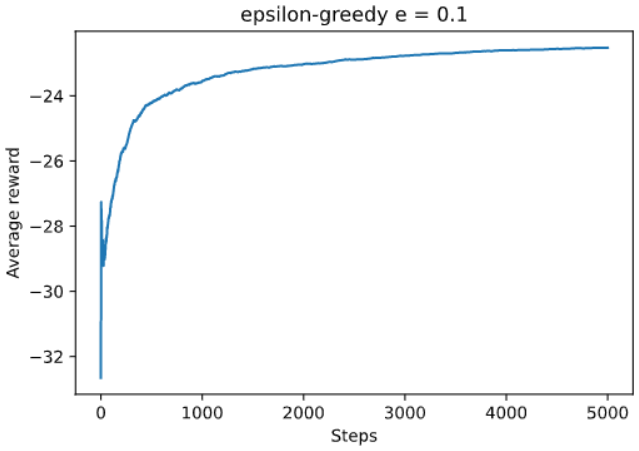
هم‌چنین برای این مسئله محیط NetworkEnvironment را پیاده سازی کردیم که از کلاس EnvironmentBase برگرفته شده است و کد آن تا حدی مشابه محیط MultiArmedBandit است که در پکیج amalearn پیاده سازی شده است.

در این مثال نیز با توجه به اینکه کل مسیر را به عنوان عمل در نظر گرفته ایم، محیط باید پس از انجام هر مرحله terminate بشود و برای این کار متد آن تغییر یافته است.

متد calculate_reward نیز متد دیگری است که باید آن را پیاده سازی کنیم که با توجه به آن هر پاداش از ۷ قسمت تشکیل شده است که هر کدام توزیع متفاوتی دارند. ابتدا هر کدام از آن ها را محاسبه کرده و حاصل جمع آن ها را با علامت منفی گزارش می‌کنیم. زیرا هر چه عدد بدست آمده بزرگتر باشد تاخیر ارسال پیام آن مسیر بیشتر است و باید جریمه بیشتری در نظر گرفته شود.

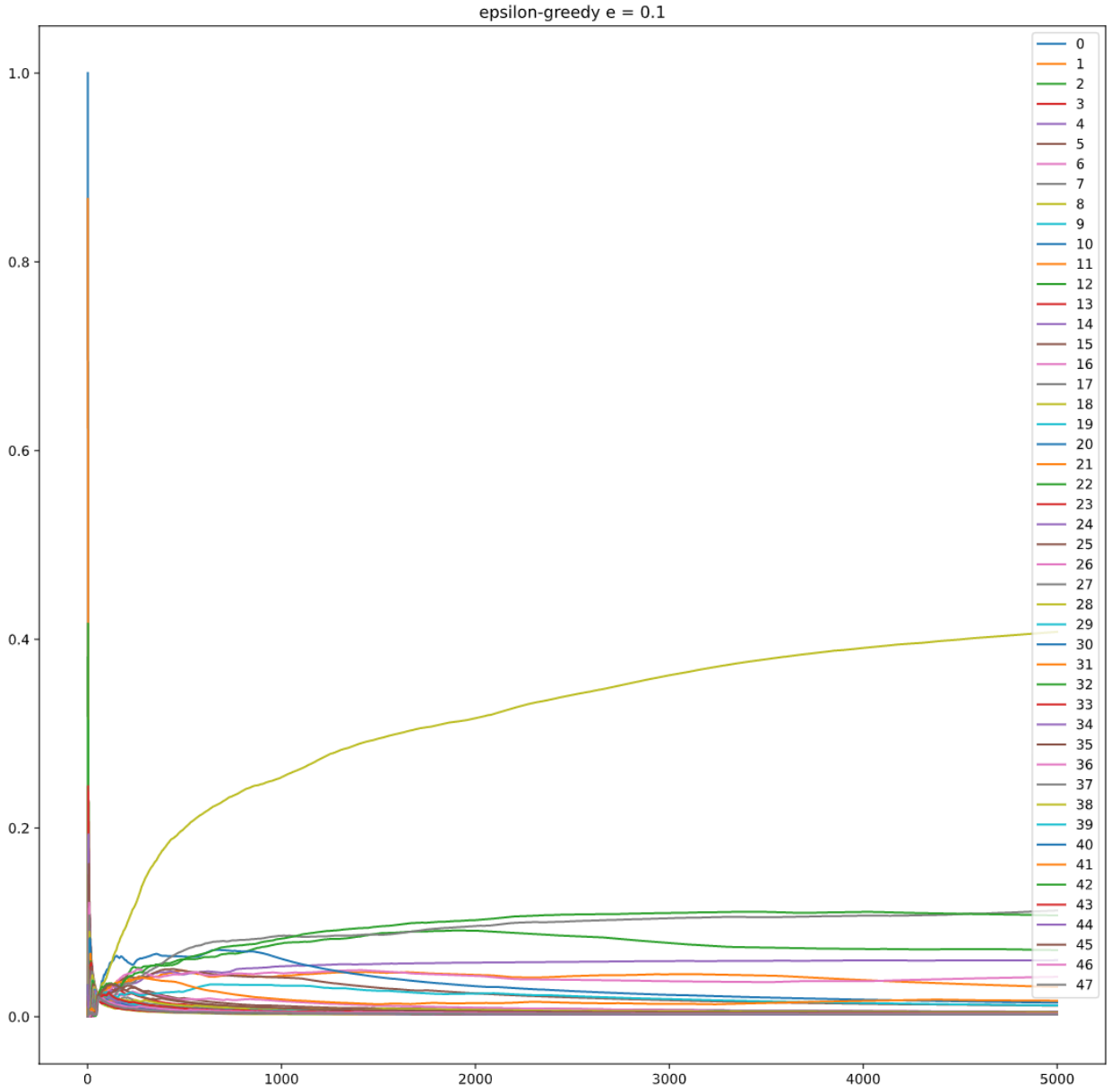
ب) ابتدا عاملی با سیاست epsilon-greedy با مقدار epsilon = 0.1 را درنظر گرفتیم. عامل را برای ۵۰۰۰ گام آموزش دادیم و با توجه به تصادفی بودن تاخیر هر لینک و هم‌چنین خود الگوریتم این کار را ۳۰ بار تکرار کرده و میانگین آن را گزارش می‌کنیم.





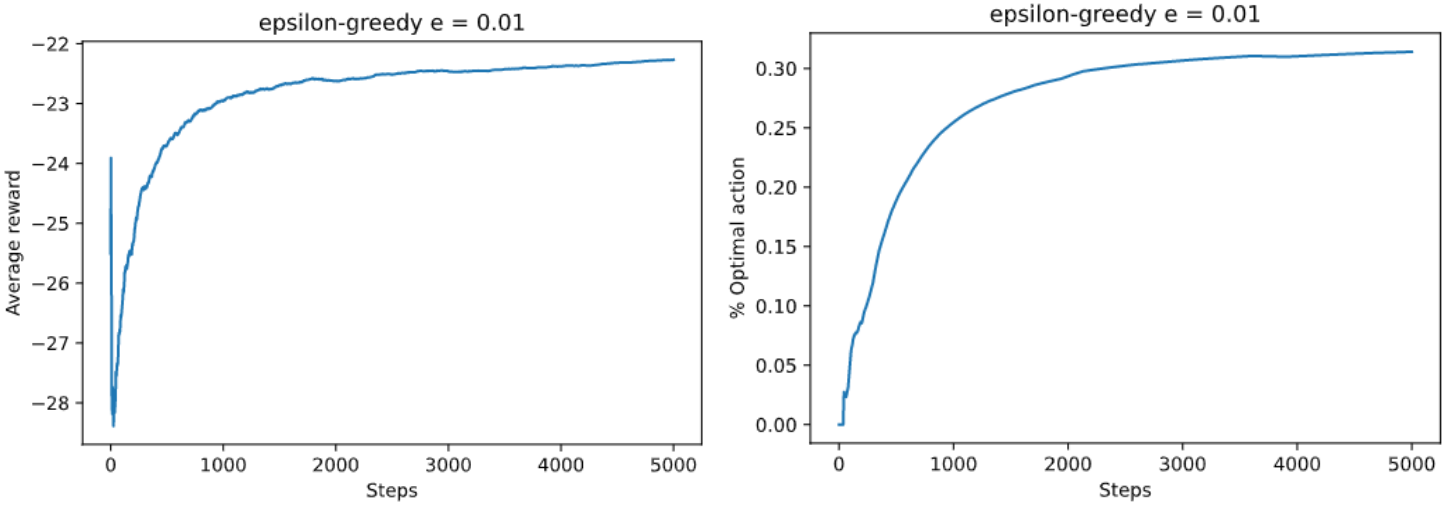
همان طور که از نمودار مشخص است عامل در کمتر از نیمی از مواقع می‌تواند عمل بهینه را پیدا کند که این مسئله با توجه به نزدیک بودن ریوارد بعضی از مسیر ها قابل حدس زدن بود.

اما برای پاسخ به این سوال که در هر بار که عامل موفق به یافتن عمل بهینه شده است عامل در چه زمانی توانسته است عمل بهینه را بیابد به این طریق عمل می‌کنیم که بعد از انجام ۵۰۰۰ گام برای هر بار اولین گامی که عامل در ۱۰۰ گام گذشته‌ی خود عمل بهینه را بیشتر از ۸۵ بار انجام داده را به عنوان زمان لازم برای پیدا کردن عمل بهینه به طور متوسط گزارش می‌کنیم، که این مقدار برای عامل با epsilon = 0.1 برابراست با ۳۶۵ گام.



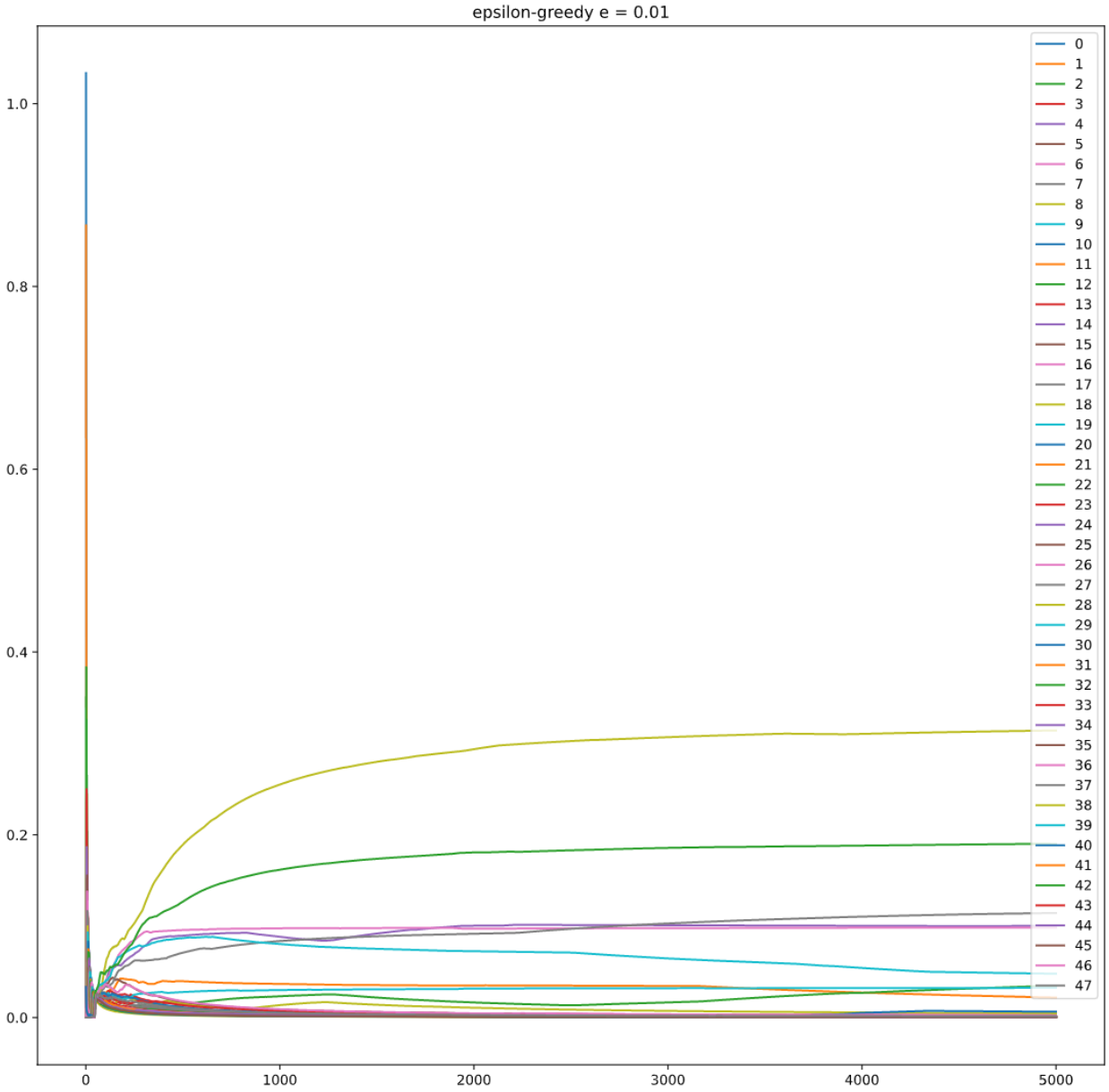
نمودار بالا نمودار تعداد دفعات انجام هر عمل به نسب کل اعمال است که از این نمودار نیز می‌توان دید که به طور متوسط پس از ۴۰۰ گام سایر عمل ها به صورت ثابت درآمده اند اما میزان انجام عمل بهینه درحال صعود است.

حال تمام مراحل بالا را برای epsilon = 0.01 نیز انجام می‌دهیم.



همان طور که از نمودار مشخص است عامل در کمتر از عامل قبلی توانسته است عمل بهینه را پیدا کند ولی میانگین پاداش دریافتی آن بیشتر از عامل قبلی است. این بدان معناست که هر چند این عامل نسبت به عامل گذشته کمتر توانسته عمل بهینه را پیدا کند اما عملکرد بهتری از خود نشان داده است که این امر با توجه به زیاد بودن تعداد اعمال و هم چنین نزدیک بودن پاداش عمل ها نسبت به عمل بهینه قابل توجیه است .

تعداد گام های لازم برای پیدا کردن عمل بهینه برای این عامل برابر است با :



نمودار بالا نمودار تعداد دفعات انجام هر عمل به نسب کل اعمال است که از این نمودار نیز می‌توان دید که به طور متوسط پس از ۴۰۰ گام سایر عمل ها به صورت ثابت درآمده اند اما میزان انجام عمل بهینه درحال صعود است.

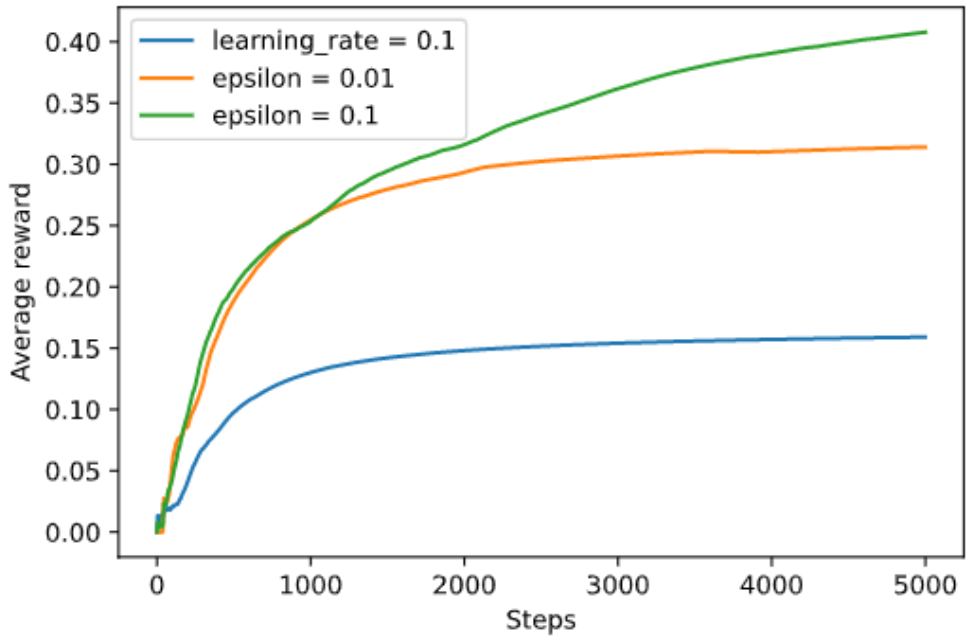
پ) در این قسمت عامل با سیاست GradientBandit را پیاده سازی کردیم که برای این کار از کلاس AgentBase ارث بری کردیم. در این عامل احتمال انتخاب هر عمل متناسب با softmax ترجیح های بدست آمده برای اعمال مختلف است. که این احتمال به صورت زیر محاسبه می شود :

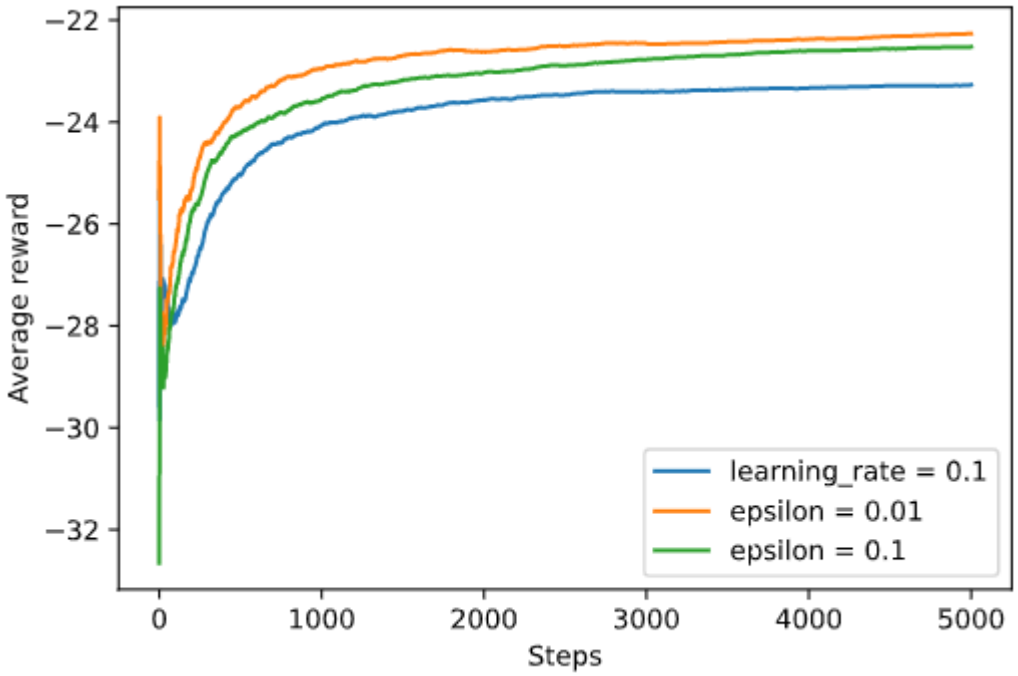
$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a),$$

هر یک از این ترجیح ها در هر مرحله با کمک گرادیان آن به روز رسانی می‌شود. که رابطه‌ی آن به روز رسانی آن از قرار زیر است بدست می‌آید:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

حال مانند دفعه های گذشته عامل را برای ۵۰۰۰ گام در محیط قرار می‌دهیم و این کار را ۳۰ بار تکرار کرده و میانگین نتیجه ها آن را گزارش می‌کنیم. که در این مرحله نتایج را با نتایج قبلی مقایسه می‌کنیم :





که با توجه به نتایج بدست آمده می‌توان گفت که عامل GradientBandit با Learning_rate = 0.1 عملکرد بدتری را نسبت به سایر مدل ها داشته است.

پ)با توجه به اینکه به طور کلی عامل هایی که بر اساس گرادیان کار می‌کنند به زمان کمتری جهت همگرایی نیازمند هستند با پیدا کردن نرخ یادگیری مناسب به نظر می‌رسد که این نوع عامل ها در این مسئله که یک مسئله **real-time** است بهتر به نظر می آیند. هم چنین از دیگر مزیت الگوریتم گرادیان نسبت به الگوریتم **epsilon-greedy** عدم نیاز به کاهش دادن هایپرپارامتر در طول زمان است.