به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

سیستم های هوشمند

تمرین شماره 6

نیما زمان پور

810198407

بهمن ماه 1401

فهرست سوالات

سوال 1

در الگوریتم policy-iteration ابتدا یک سری policy رندوم برای هر state که در اینجا خانه های جدول است؛ قرار می‌دهیم.

سپس value function را برای هر policy حساب کرده. و دوباره policy ها را بر اساس بهترین value function انتخاب می‌کنیم. این کار را تا میل کردن به policy بهینه انجام می‌دهیم.

$$V_\pi(s) = r_s^{\pi(s)} + \gamma \sum_{s'} P_{ss'}^{\pi(s)} V_\pi(s') \qquad (1)$$

$$\pi'(s) \coloneqq \arg\max r_s^a + \gamma \sum_{s'} P_{ss'}^{\pi(s)} V_\pi(s') \qquad (2)$$

با توجه به صورت سوال $\gamma = 0.2$ و با احتمال 0.6 به جهت دلخواه و 0.2 به طرفین حرکت می‌کنیم.(در شماره دانشجویی من b=0 بود. لذا برای صورت مسئله پویا تر اعداد رند و مناسب دلخواه انتخاب کردم)

برای سادگی کار، ابتدا مقدار همه $V_0 = 0$ در نظر گرفته(در غیر این صورت در هر مرحله باید 12 معادله 12 مجهول را حل می‌کردیم.) با و Policy رندوم مقدار می‌دهیم:

Iteration 1:

0	0	0	3
0	0	0	-2
0	■	0	0

→	→	→	3
→	→	→	-2
→	■	→	→

حال مقدار value function را بر اساس policy ها بروز می‌کنیم.

$$V_\pi((1,3)) = r_s^a + \gamma \left(P_{(1,3)\to(1,4)}^{right} V_\pi((1,4)) + P_{(1,3)\to(1,3)}^{up} V_\pi((1,3)) + P_{(1,3)\to(2,3)}^{down} V_\pi((2,3))\right)$$
$$= 0 + 0.2(0.6 * 3 + 0.2 * 0 + 0.2 * 0) = 0.36$$

بقیه خانه ها را به همین منوال آپدیت می‌کنیم:

$$V_\pi((2,3)) = 0 + 0.2(0.6 * -2 + 0.2 * 0 + 0.2 * 0) = -0.24$$

$$V_\pi((3,4)) = 0 + 0.2(0.6*0 + 0.2*-2 + 0.2*0) = -0.08$$

بقیه خانه ها همگی $V(s') = 0$ است. و $V(s)$ صفر می‌شود. حال بر اساس value function های بدست آمده. جهت policy بهینه، مسیر رفتن به خانه با ماکسیمم value function است.

$$\pi'((1,3)) = argmax(0.6*3 + 0.2*0.36 + 0.2*-0.24,$$
$$0.6*0.36 + 0.2*3 + 0.2*0, 0.6*0 + 0.2*0.36*0.2*-0.24,$$
$$0.6*-0.24 + 0.2*3 + 0.2*0) =\rightarrow$$

$$\pi'((2,3)) = argmax(0.6*0.36 + 0.2*-2 + 0.2*0,$$
$$0.6*-2 + 0.2*0.36 + 0.2*0, \quad 0.6*0 + 0.2*0.36*0.2*0,$$
$$0.6*0 + 0.2*0 + 0.2*-2) =\uparrow$$

Iteration 2:

0	0	0.36	3
0	0	-0.24	-2
0		0	-0.08

→	→	→	3
→	↑	↑	-2
→		←	←

حال مقدار value function را بر اساس policy ها بروز می‌کنیم.

$$V_\pi((1,3)) = 0 + 0.2(0.6*3 + 0.2*0.36 + 0.2*-0.24) = 0.37$$
$$V_\pi((2,3)) = 0 + 0.2(0.6*0.36 + 0.2*-2 + 0.2*0) = -0.04$$
$$V_\pi((3,3)) = 0 + 0.2(0.6*0 + 0.2*0 + 0.2*-0.24) = -0.01$$
$$V_\pi((3,4)) = 0 + 0.2(0.6*0 + 0.2*-0.08 + 0.2*-2) = -0.08$$
$$V_\pi((1,2)) = 0 + 0.2(0.6*0.36 + 0.2*0 + 0.2*0) = 0.04$$
$$V_\pi((2,2)) = 0 + 0.2(0.6*0 + 0.2*0 + 0.2*-0.24) = -0.01$$
$$V_\pi((1,1)) = V_\pi((1,2)) = V_\pi((1,3)) = 0$$

سپس policy ها را بروز می‌کنیم. (بر اساس فرمول (2) محاسبات را انجام می‌دهیم. که 2 نمونه از آن در iteration 1 آورده شده)

Iteration 3:

0	0.04	0.37	3
0	-0.01	-0.04	-2
0	■	-0.01	-0.08

→	→	→	3
→	↑	↑	-2
→	■	←	←

دوباره مقدار value function را بر اساس policy ها بروز می‌کنیم.

$$V_\pi\big((1,3)\big) = 0 + 0.2(0.6 * 3 + 0.2 * 0.37 + 0.2 * -0.04) = 0.37$$
$$V_\pi\big((2,3)\big) = 0 + 0.2(0.6 * 0.37 + 0.2 * -2 + 0.2 * -0.01) = -0.04$$
$$V_\pi\big((3,3)\big) = 0 + 0.2(0.6 * -0.01 + 0.2 * -0.01 + 0.2 * -0.04) = -0.003$$
$$V_\pi\big((3,4)\big) = 0 + 0.2(0.6 * -0.01 + 0.2 * -0.08 + 0.2 * -2) = -0.08$$
$$V_\pi\big((1,2)\big) = 0 + 0.2(0.6 * 0.37 + 0.2 * 0.04 + 0.2 * -0.01) = 0.05$$
$$V_\pi\big((2,2)\big) = 0 + 0.2(0.6 * 0.04 + 0.2 * 0 + 0.2 * -0.04) = -0.003$$
$$V_\pi\big((1,1)\big) = 0 + 0.2(0.6 * 0.04 + 0.2 * 0 + 0.2 * 0) = 0.005$$
$$V_\pi\big((2,1)\big) = 0 + 0.2(0.6 * -0.01 + 0.2 * 0 + 0.2 * 0) = -0.001$$
$$V_\pi\big((3,1)\big) = 0$$

جدول نهایی بصورت زیر درمی‌آید:

0.005	0.05	0.37	3
-0.001	-0.003	-0.04	-2
0	■	-0.003	-0.08

→	→	→	3
↑	↑	↑	-2
↓	■	←	←

سوال 2

در مسائل یادگیری تقویتی مبتنی بر مدل، با داشتن مدل MDP مسئله، می‌توان policy بهینه را بر اساس روش های مبتنی بر تکرار پیدا کرد.

حالا برای مدل کردن مسئله نگاه می‌کنیم که فرد هر بار تصمیم می‌گیرد که چه مقدار از موجودی خود را در شرط بندی بگذارد. سپس با احتمال خط آمدن به اندازه این مقدار، پول از دست داده و با احتمال شیر آمدن به دست می‌آورد. و موجودیش تغییر می‌کند. پس می‌توان گفت که فضای اقدام مقداری که فرد شرط میبندد است.(باتوجه به موجوی وی). و فضای حالت موجودی فرد است. یعنی به اندازه 0$ تا 100$ 101 حالت داریم. و برای هر حالت نیز برای مثال 40 دلار موجودی، از هیچی شرط بندی نکردن تا همه ی پول را شرط بندی کردن action داریم.

$$P_{ss'}^{a} = \begin{cases} p_h & s_{t+1} = s + a \\ 1 - p_h & s_{t+1} = s - a \end{cases}$$

$$R_{ss'}^{a} = 1 * p_h + 0 * (1 - p_h) = p_h$$

حالا الگوریتم value-iteration را پیاده سازی می‌کنیم. ابتدا باید V(s) را برای همه state ها مشخص کنیم. برای حالت 0 دلار مقدار -100 و برای حالت 100 دلار مقدار 100+ را قرار می‌دهیم. که هر کدام به معنای باخت و برد هستند.(این value ها آپدیت نمی‌شوند) سپس برای باقی حالات V(s)=0 را مقدار دهی اولیه می‌کنیم. و الگوریتم را تا رسیدن به یک policy بهینه اجرا می‌کنیم. با تمام شدن الگوریتم، هر policy بهینه مقدار پول شرط بندی را برای هر state نشان می‌دهد.

$$Q(s,a) = \sum_{s'} P_{ss'}^{a}\left[R_{ss'}^{a} + \gamma V(s')\right]$$

$$V(s) = \max Q(s,a)$$

$$\pi(s) = argmax\ Q^{*}(s,a)$$

با اجرای الگوریتم به میزان 100 بار نتایج زیر بدست می‌آید:

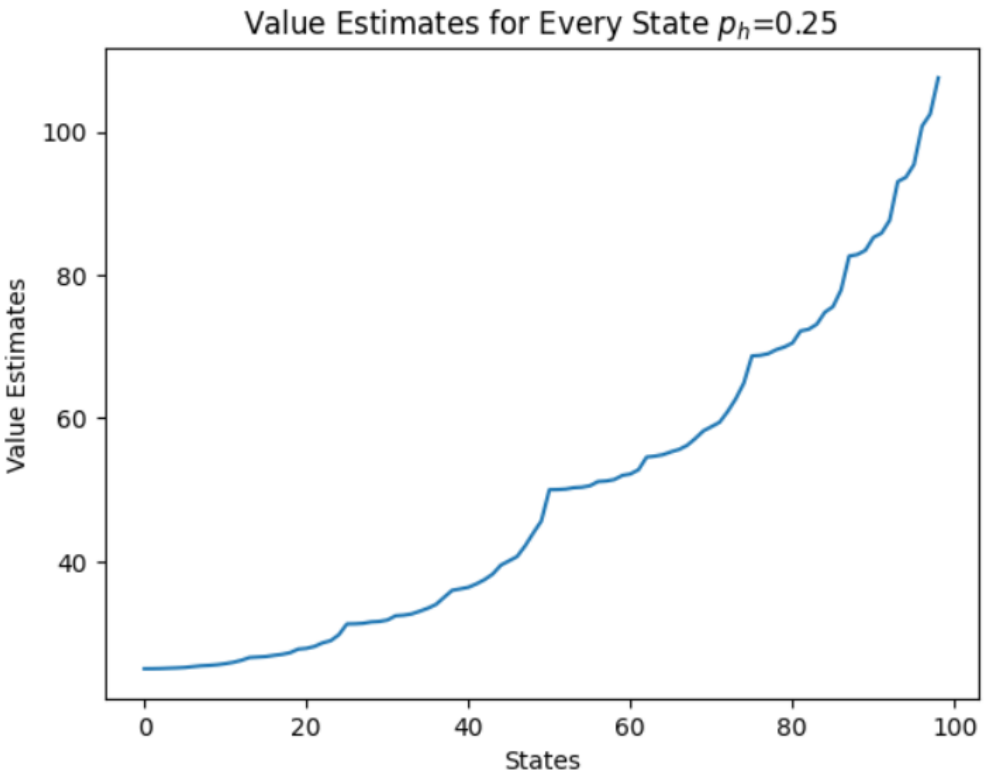

Figure 1 نمودار **value estimation** مدل برای **ph = 0.25**

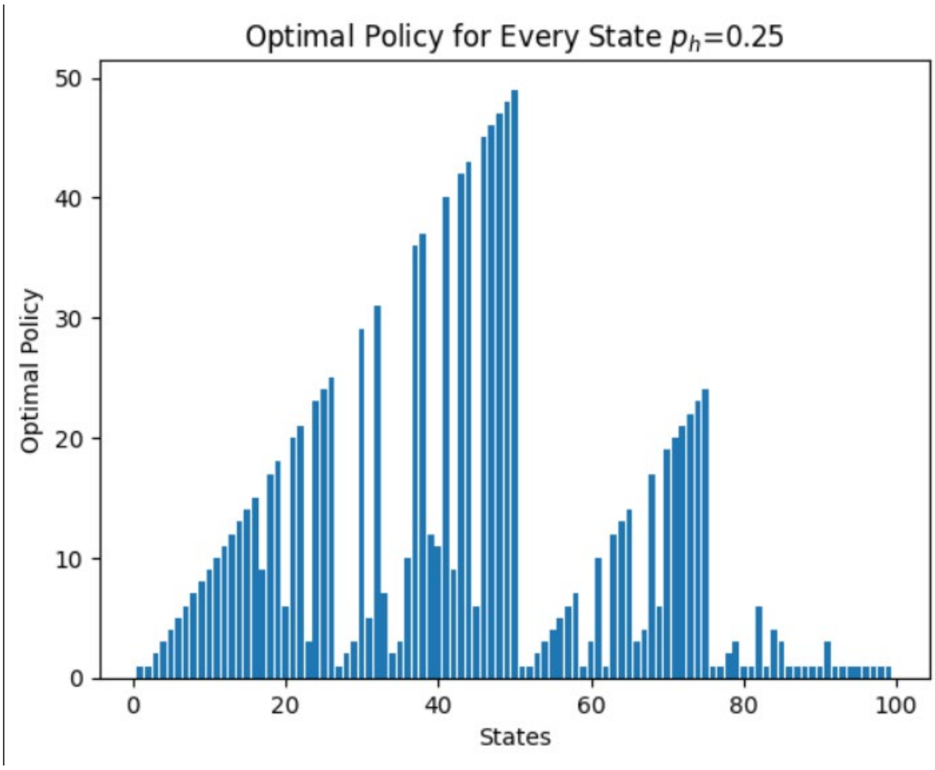

Figure 2 نمودار سیاست بهینه به ازای هر حالت در **ph = 0.25**

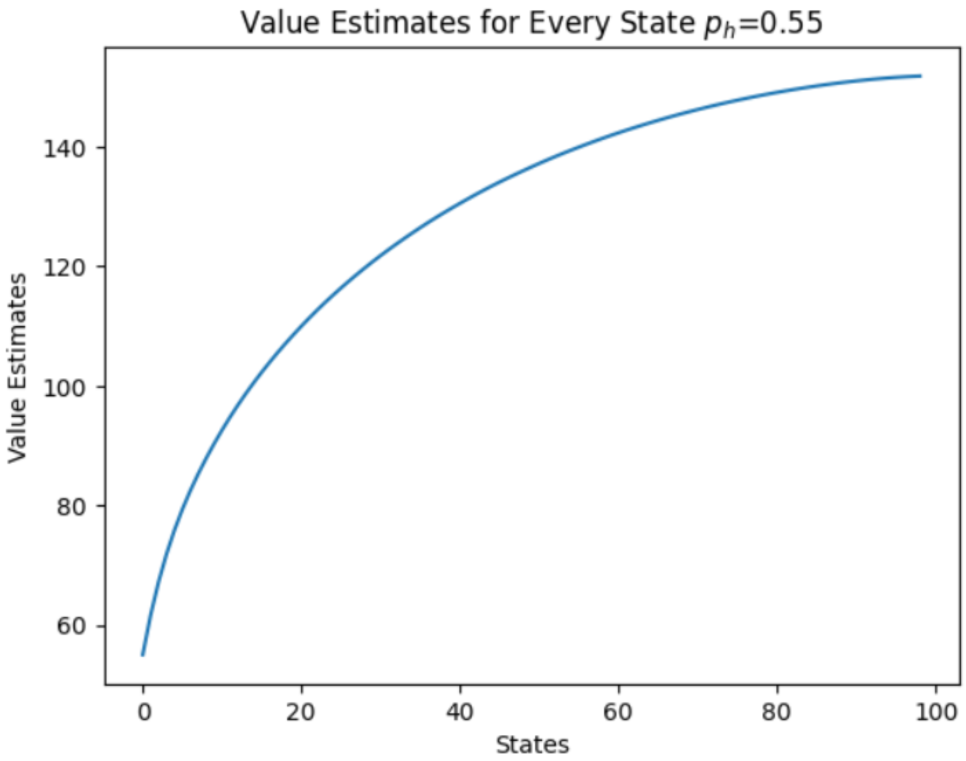

Figure 3 **value estimation** مدل برای**ph = 0.55**

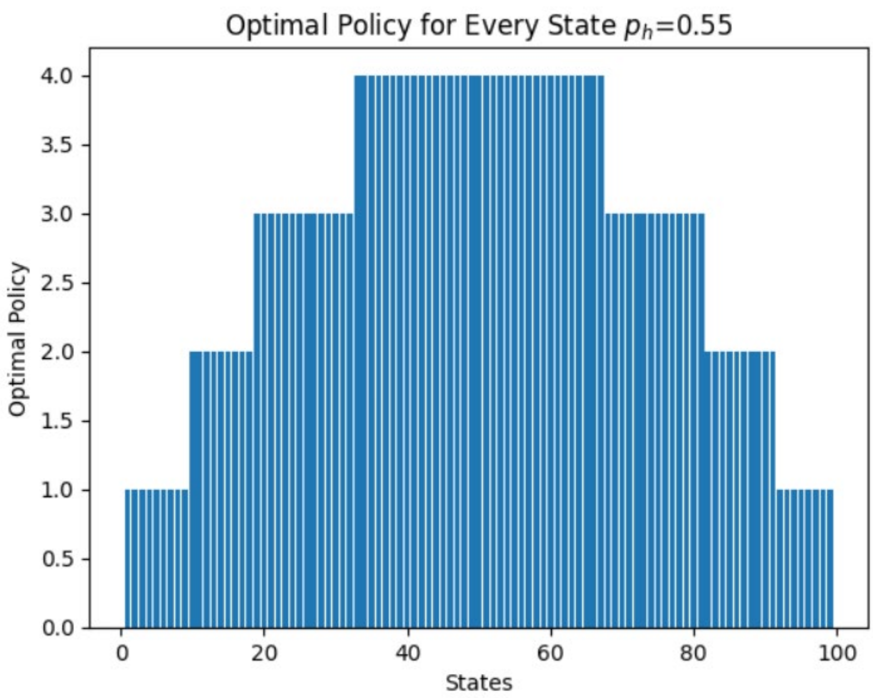

Figure 4 نمودار سیاست بهینه به ازای هر حالت در **ph = 0.55**

با نگاه به سیاست های بهینه هر دو حالت می‌توان متوجه شد. که در حالت $p_h = 0.25$ به ازای هر مقدار پولی که کمتر از 50 دلار است. بهترین کار این است که همه آن را شرط بندی کنیم. اما برای موجودی های بالاتر از 50 دلار مقدار کمتری در حدود 10 دلار شرط بندی می‌کنیم.

در حالت $p_h = 0.55$ به ازای مقادیر بین 30 تا 70 دلار 4 دلار شرط می‌بندیم. و در هر 10 دلار موجودی که کمتر یا بیشتر شود. 1 دلار میزان شرط بندی کاهش می‌یابد.(همانند شکل)

بطور کلی وقتی که شانس بردن زیر 0.5 باشد. در یک مدت طولانی بازی همیشه فرد همه پولش را از دست خواهد داد. پس بهتر است که شرطی نبندد. اما در تنظیم بازی می‌بینیم که مجازاتی برای باختن در نظر نگرفته شده و همنچین پاداشی برای ریسک های بزرگ نیز اعمال نشده.(خلاف معقول) به این ترتیب عامل وقتی که شانس بردن کم است. حالت(0.25) عامل شرط بندی های بزرگی انجام می‌دهد. و وقتی شانس بردن اندکی از شانس باختن کمتر است.(و در دراز مدت عامل پولش به بی‌نهایت میل می‌کند.) عامل شرط های بزرگی نمی‌بندد و معقول عمل می‌کند.

سوال 3

با نگاه به شکل سوال می‌توان دریافت که تاکسی در یک جدول 5*5 قرار دارد. در این جدول دو نقطه از 4 نقطه R, G, B, Y به عنوان مقصد و مبدا هستند و مسافر در هر لحظه یا منتظر سوار شدن است. یا سوار تاکسی است، یا پیاده شده. پس:

$$state\ space\ size = 5^2 * 4 * 5 = 500$$

500 حالت در بازی وجود دارد. که البته تعدادی از آنها در بازی نیستند مثلا وقتی که مبدا و مقصد یکی باشد. پس در کل:

$$reachable\ state\ size = 500 - 25 * 4 = 400$$

حالت درون بازی داریم و 4 حالت نیز وقتی داریم که بازی تمام شده و مسافر به مقصد رسیده(تاکسی نیز در همان مقصد است.) پس در کل 404 حالت در بازی داریم

برای هر state نیز 6 حرکت مطابق صورت سوال داریم که بالا، پایین، چپ، راست، سوار کردن و پیاده کردن است.

الف) پیمایش رندوم

برای این قسمت یک while ایجاد می‌کنیم. از دستور action_space.sample() یک action رندوم انتخاب کرده. و آن را به تابع step() می‌دهیم. این تابع state بعدی، reward حرکت فعلی و وضعیت تمام شدن این اپیزود(متغیر done) را برمی‌گرداند. و همینطور تکرار می‌کنیم. این حرکت در while تا جایی ادامه پیدا می‌کند. که متغیر done =True شود که به معنای تمام شدن بازی است. در طول بازی تمامی پاداش ها را باهم جمع کرده و همچنین تعداد گام ها نیز ذخیره می‌کنیم. حال خود این حلقه را نیز به تعداد زیادی اجرا کرده. و هر بار میزان پاداش کل و تعداد کل گام ها را ذخیره می‌کنیم.

با 1000 بار اجرای کد نمودار های زیر بدست آمد:

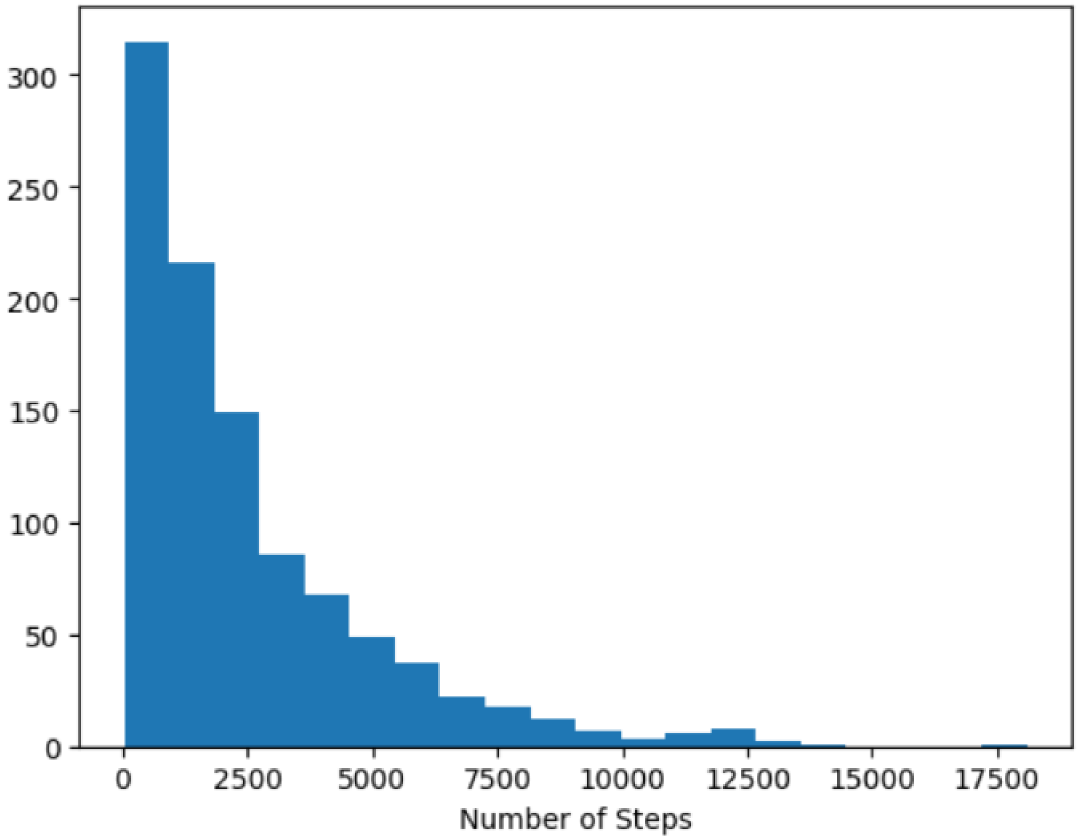

Figure 5 نمودار هیستوگرام تعداد گام های هر بار بازی

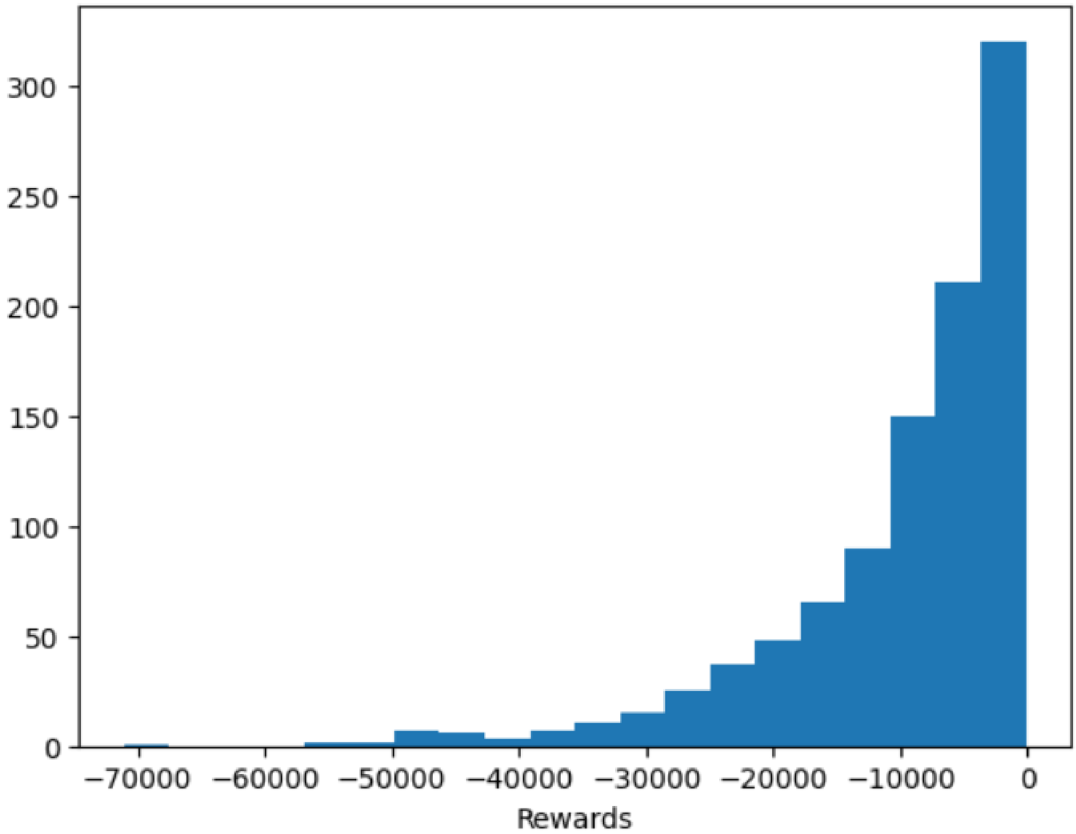

Figure 6 نمودار هیستوگرام مجموع پاداش ها در هر بار بازی

برای پیمایش رندوم میانگین تعداد گام ها 2352 و میانگین پاداش کل 9186- است. همچنین به ازای هر گام میانگین 4- پاداش می‌گیریم.(که با مجموع کل پاداش های هر action تقسیم بر تعداد اکشن ها برابر است. $\frac{-1-1-1-1-10-10}{6} = -4$)

همانطور که می‌بینیم پیمایش رندوم طول گام بسیار زیادی نسبت به اندازه کوچک جدول نیاز دارد. که نشان می‌دهد این حرکت اصلا بهینه نیست و نیاز به پیمایش هوشمندانه دارد.

ب) پیمایش هوشمندانه مبتنی بر Q-Learning

حال به پیاده سازی الگوریتم Q-Learning می‌پردازیم. الگوریتم بصورت زیر است:

Set values for learning rate α、 discount rate γ、 reward matrix R

Initialize $Q(s,a)$ to zeros

Repeat for each episode,do

  Select state s randomly

  Repeat for each step of episode,do

    Choose a from s using ε-greedy policy or Boltzmann policy

    Take action a obtain reward r from R, and next state s'

    Update $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

    Set $s = s'$

    Until s is the terminal state

  End do

End do

Figure 7 فلوچارت الگوریتم **Q-Learning** منبع

برای پیاده سازی الگوریتم بالا بر رو مدل بالا ابتدا یک جدول به ابعاد 500*6 که طول و عرض آن اندازه فضای حالت و فضای عمل مدل است. به عدد 0 مقدار دهی می‌کنیم. و مدل را reset کرده و به یک State رندوم آغاز می‌کنیم.

بعد در یک حلقه while همانند قسمت الف) که شرط آن done=Ture بود به فرم زیر action ها را انتخاب می‌کنیم.

بوسیله ε-greedy policy با احتمال ε یک action رندوم و به احتمال 1-ε در سطر مربوط به state حال حاضر action با بیشترین Q(s,a) را برمی‌گزینیم. آن را به مدل داده و به یک state جدید s' می‌رویم.

حال بر اساس پاداشی که در این state گرفتیم جدول را بصورت زیر آپدیت می‌کنیم:

$$Q(s,a) = Q(s,a) + lr[r + \gamma \max Q(s',a') - Q(s,a)]$$

پارامتر گاما که نشاندهنده اهمیت پاداش های دیرهنگام است. را نزدیک به یک برابر 0.99 قرار می‌دهیم چون تنها پاداش مثبتی که عامل می‌گیرد در پایان سفر نصیبش می‌شود. نرخ یادگیری را برابر 1 و اپسیلون را برابر 0.2 قرار میدهیم و هر 100 epoch آن ها را نصف می‌کنیم.

اساس کار الگوریتم Q-learning بشرح بالا بود. بقیه کد تابع همانند قسمت الف است. مجموع پاداش و طول گام سفر را در هر اپیزود ذخیره کرده و کل الگوریتم را به تعداد epoch اجرا می‌کنیم. تا لیستی از مجموع پاداش و طول گام سفر تشکیل شود.

با 1000 بار اجرای کد نمودار های زیر بدست آمد:

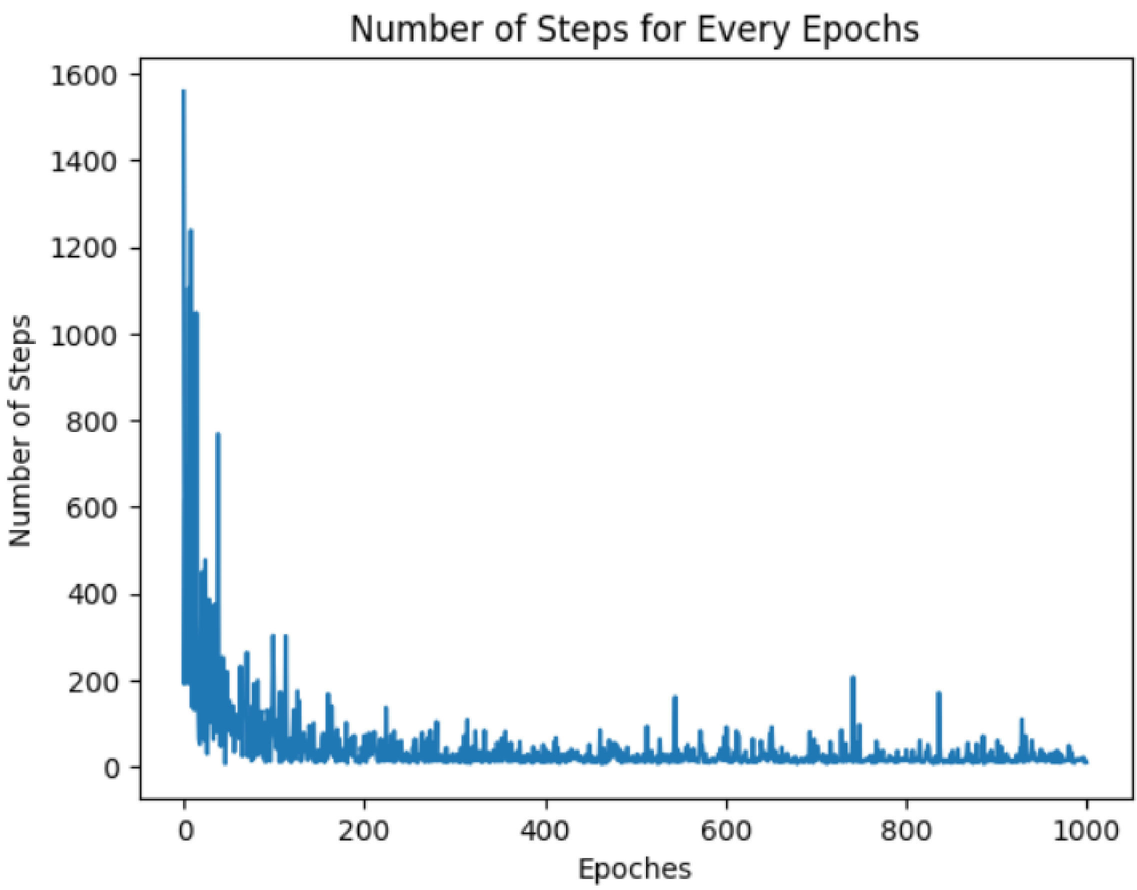

Figure 8 تعداد گام ها در هر تکرار در روش **Q-Learning**

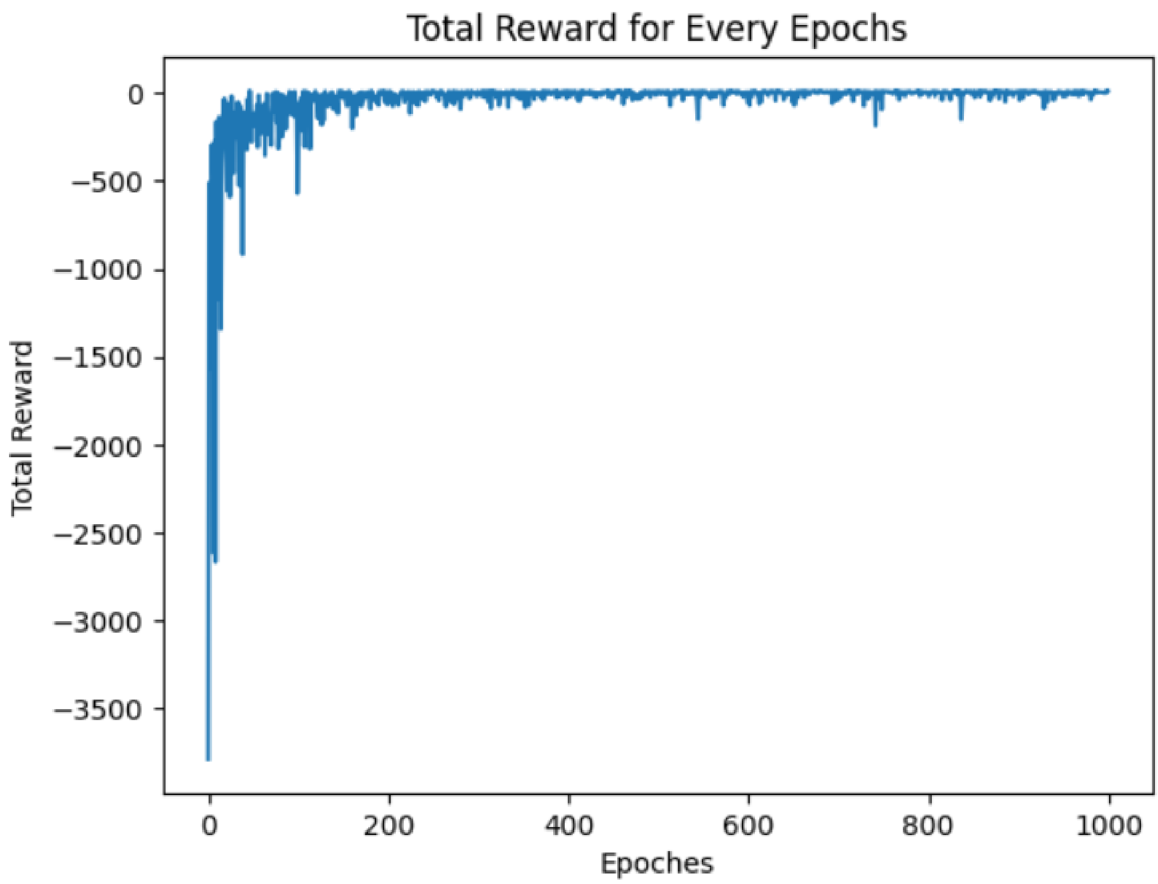

Figure 9 مجموع پاداش ها در هر تکرار در روش **Q-Learning**

برای پیمایش هوشمندانه در 100 تکرار آخر، میانگین تعداد گام ها 19.5 و میانگین پاداش کل 0.71 است. همچنین به ازای هر گام میانگین 0.036- پاداش می‌گیریم. که نسبت به پیمایش رندوم بسیار جریمه کمتر و تعداد گام کمتری است.

همانطور که می‌بینیم در پیمایش هوشمندانه، عامل در حین تعامل با محیط پاداش هایی که می‌گیرد. منجر به این می‌شود که انتخاب هایی در هر state انجام دهد که Expected Reward بیشتری نصیبش شود.(با توجه به اینکه گاما یک است. پاداش های بلندمدت را در نظر می‌گیرد). Q(s) هر حالت رفته آپدیت می‌شود. و مدل متوجه می‌شود که در هر حالتی باید چه action انجام دهد. البته این نوع از ایجاد جدول در حالتی ممکن است که تعداد state*action محدود باشد. در غیر این صورت نمی‌توان از جدول استفاده کرد و باید به سراغ روش های دیگری همانند Deep Q-Learning رفت.

برای این که به سرعت بیشتر همگرایی برسیم به این توجه می‌کنیم تنها پاداش مثبت بازی در آخر بازی نصیب عامل می‌شود که مسافر را به مقصد پیاده می‌کند. با زیاد کردن این مقدار، Q(s) آن بیشتر شده و عامل در انتخاب Policy به سمت انتخاب آن همگرا می‌شود. این مقدار را به 50 می‌رسانیم بقیه

موارد دست نخورده باقی می‌مانند.(توجه کنیم که مقدار مجازات اشتباهی پیاده و سوار کردن 10 است. یعنی اگر راننده مسافر را اشتباهی پیاده کند باید دوباره آن را سوار کند. در محیط 5*5 بطور میانگین 5 6 خانه را دوباره باید برگردد. پس میزان جریمه آن 11-=2*5.5- است. که تقریبا با مقدار پیش فرض برابر است. مقدار جریمه سوم که برای حرکت به اطراف است نیز بی تغییر باقی می‌ماند چون جریمه ها نسبت به هم سنیجده می‌شوند و اگر آن را زیاد کنیم همانند این است که 2 تای دیگر را کم کردیم)

با اعمال این تغییرات در 200 تکرار بازی با هایپر پارامتر های یکسان در 50 تکرار آخر میانگین طول کام بدون تغییرات برابر 46.4 و با تغییر برابر 36.6 شد که همگرایی سریعتری را رقم زد.

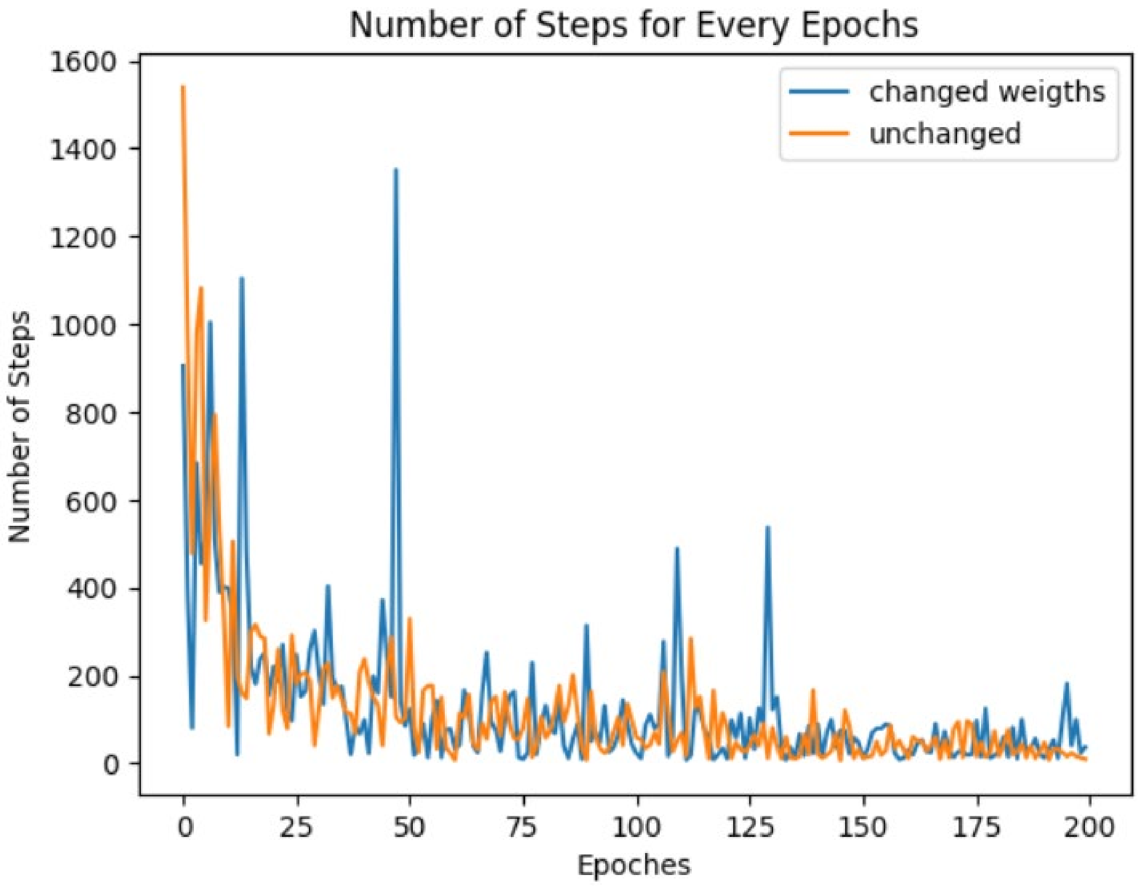

Figure 10 مقایسه مجموع طول گام بازی در 200 تکرار در حالت با تغییر و بدون تغییر

پیوست

سوال 2)

برای پیاده سازی کد این سوال ابتدا یک تابع valueIterationGambling() تعریف می‌کنیم. که درواقع تابع اصلی ما است. و خروجی آن V(s) های مدل است. درون آن ابتدا یک ارایه به اندازه 201 تعریف می‌کنیم. این آرایه همه موجودی های ممکن شرط بندی را داخل خود دارد.(مثلا وقتی فرد 100 دلار دارد و همه آن را شرط بندی می‌کند. اگر ببازد پولش 0 و اگر ببرد پولش 200 دلار می‌شود.) برای پول های بالاتر از 100 دلار V(s)=100 است. و برای V(s=0$)=-100 است.(منطق کار در سوال توضیح داده شد) سپس در هر مرحله از value-iteration یک کپی از لیست در می‌کنیم. تا در حین یک مرحله آپدیت مقداری تعییر نکند. و در آخر هر مرحله کپی را جایگزین مقدار اصلی می‌کنیم.

حال برای آپدیت مقدار هر state تابع getValueFunction() را تعریف می‌کنیم. این تابع Q(s) را برای همه‌ی action ها حساب می‌کند. و ماکسیمم آن ها را برمی‌گرداند.

تابع valueIterationGambling() الگوریتم بالا را به اندازه آرگومان ورودی epoch تکرار می‌کند. تا الگوریتم همگرا شود. در آخر نیز تابع getOptimalPolicy() همه Q(s,a) های ممکن برای هر state را در یک آرایه ریخته. و بیشترین آن را انتخاب کرده و آرگومان آن را بعلاوه یک برمی‌گرداند.(چون action ها از یک دلار تا sدلار هستند و اندیس آرایه از 0 شروع می‌شود. پس بعلاوه یک می‌کنیم تا همان مقدار دلاری برگردانده شود.)

توابع را برای ph=0.55,0.25, epoch=100 اجرا کرده. و نمودار های Value Estimate, Optimal Policy را به ازای هر state رسم می‌کنیم.

سوال 3)

به کمک فایل راهنمایی که در پوشه سوال قرار داده شد، محیط بازی را با مود ansi ایجاد می‌کنیم تا محیط بازی در کنسول چاپ شود و رندر زمان کمتری نسبت به حالت گرافیکی بگیرد.(قسمت زیادی از کد در جواب سوال 3 گنجانده شده.) برای حالت پیمایش رندوم تابع random_exploration() را تعریف می‌کنیم. این تابع بصورت کاملا کور عامل را در محیط چرخانده تا بالاخره به هدف خود و انتهای اپیزود برسد. ورودی آن تعداد epoch و خروجی آن لیستی از مجموع گام و پاداش هر epoch است. این تابع را برای epoch = 1000 اجرا کرده و خروجی های آن را رسم می‌کنیم.

برای قسمت Q-Learning تابع ()QLearningTaxi را تعریف می‌کنیم. محیط را هر reset epoch کرده و از یک state رندوم شروع می‌کنیم. الگوریتم را اجرا می‌کنیم. و در آخر محیط را می‌بندیم. درون حلقه while یک شرط وجود دارد که اگر تعداد گام ها از عدد 3000 بیشتر شد و مدل گیر کرده بود؛ حلقه را بشکند و محیط reset شود.

تابع تعداد epoch، اپسیلون، گاما و lr را می‌گیرد. و جدول Q و لیست مجموع پاداش ها و گام ها را بر می‌گرداند. تابع را برای epoch=1000, epsilon=0.2, gamma=0.99, lr=1 اجرا کرده و خروجی هارا رسم می‌کنیم.

برای قسمت تغییر پاداش یک تابع کمکی ()change_rewards تعریف می‌کنیم که صرفا مقدار های 20، 10- ،1- را به مقادیر پاداش دیگری نظیر می‌کند. و در آرگومان تابع اصلی یک بولین برای اعمال این تغییرات گنجانده می‌شود.

برای قسمت نمایش گرافیکی 3 حالت وجود دارد.

1. Ansi: که محیط را در کنسول چاپ می‌کند.
2. Rgb_array: که عکس از محیط بصورت آرایه نامپای می‌دهد که با دستور ()imshow نمایش داده می‌شود
3. Human: که در یک پنجره جدا همانند یک بازی محیط را نمایش می‌دهد.

برای حالت اول برای اینکه بتوان بخوبی انیمیشن بازی را نشان داد نیاز است که فریم به فریم بازی روی یک دیگر چاپ شود تا حس پویا بودن بدهد. متاسفانه دستوری برای پاک کردن کنسول در jupyter notebook یافت نشد تا این کار انجام شود

برای حالت دوم که از کتابخانه matplotlib استفاده می‌کند. باید همه کد سلول اجرا شود تا محیط نمایش داده شود و متاسفانه در این حالت نیز نمی‌توان انیمشن پویا استفاده کرد

برای حالت سوم نیز با اولین اجرای پنجره بازی، کد crash می‌کند.

لذا برای حل این مشکل 2 کد قسمت رندوم و Q-Learning را در 2 فایل جدای پایتون قرار می‌دهیم. در این محیط بخوبی انیمشین حالت human اجرا می‌شود. کد قسمت رندوم دقیقا همانند قسمت while کد تابع آن است. و کد قسمت Q-learning نیز از توابع یکسان استفاده می‌کند. با این تفاوت که خروجی تابع که جدول Q می‌باشد را یکبار در حالت ansi آموزش داده و یکبار در حالت human

در یک حلقه while جهت هدایت عامل استفاده می‌کند. برای واضح تر شدن حرکت تاکسی از تابع sleep کتابخانه time استفاده می‌کنیم. تا هر 0.1 ثانیه یک فریم رندر شود.

فایلی که برای راهنمایی قرار داده شده بود. متاسفانه دارای ارور بود.(به دلیل هماهنگ نکردن نسخه v3 با نسخه قدیمی v2) به همین دلیل در قسمت انکود کردن کدی که محیط را به یک state دلخواه می‌برد(env.s = state) کار نمی‌کرد.