یادگیری تقویت شده

گزارش پروژه پنجم

نام استاد : دکتر ستوده

نام دانشجو : حمزه قائدی

شماره دانشجویی : 9831419

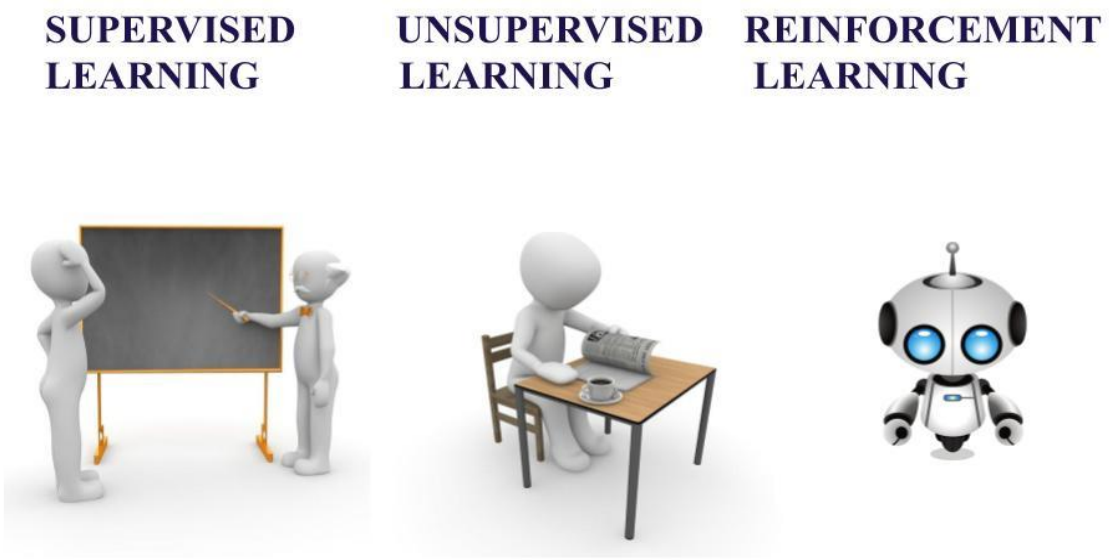
تاریخ :99/5/10

فهرست

بخش اول

مقدمه

تاکنون، با دو پارادایم کلی در مسائل یادگیری ماشین[1] آشنا شده ایم، یادگیری بانظارت[2] و یادگیری بدون نظارت[3]. در این پروژه، با سومین پارادایم در یادگیری ماشین،به نام یادگیری تقویت شده[4] آشنا میشویم. در زیر مختصرا به تشریح این سه دسته میپردازیم



یادگیری با نظارت :

متداول ترین پارادایم در یادگیری ماشـین میباشـد. در این حالت ما تعدادی ورودی[5] به همراه خروجی[6](یا برچسـب) نظیر مربوط به هر ورودی را در اختیار داریم و معنی یادگیری در این پارادایم، یافتن نوعی رابطه یا نگاشت بین ورودیها و خروجیهای نظیرشان است. مسائل این حوزه نیز خود به دو دسته رگرسیون[7] و طبقه بندی[8] تقسیم شده اند

یادگیری بدون نظارت:

در این روش، ورودی ها را در اختیار داریم اما خروجی نظیرشـان را نداریم (داده ها برچسـب ندارند) و معنی و هدف یادگیری در این حالت، یافتن الگوها، ساختارها و همبستگی بین این داده هاست (مسائل خوشه بندی[9]) و یا در حالت کلیتر یافتن توزیع احتمالی که داده ها از آن استخراج شده اند.

یادگیری تقویت شده:

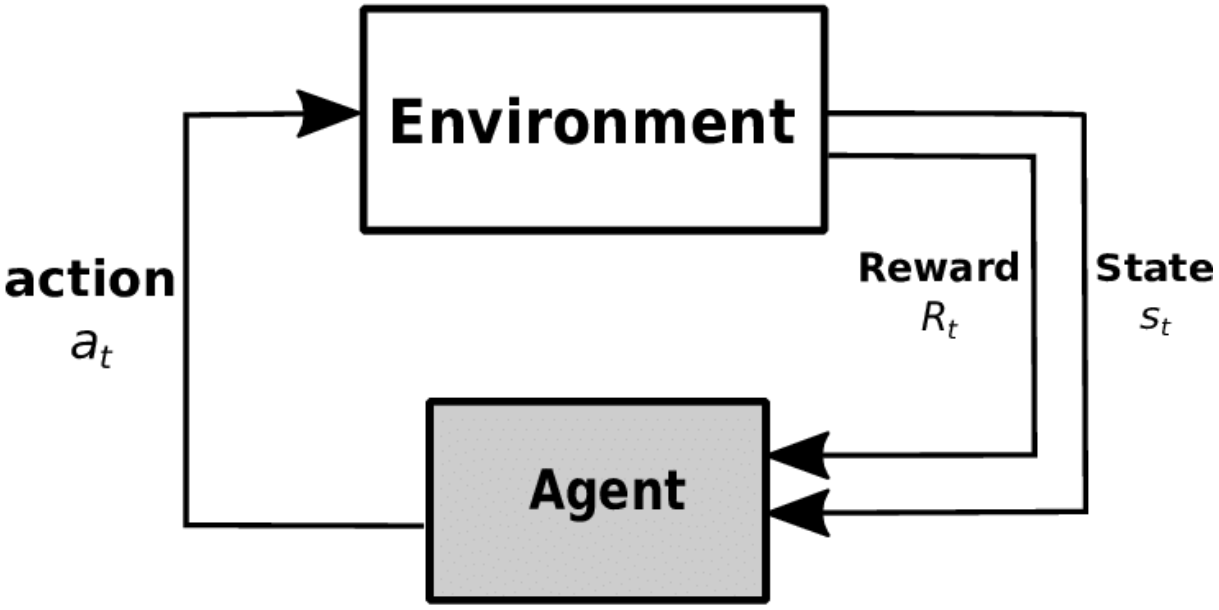
در این پارادایم، معمولا داده ای (چه به صـورت برچسـب دار و یا بدون برچسـب) نداریم و عامل یادگیر طی تعامل با محیط اطراف خود، اطلاعاتی را جمع آوری کرده و بر اسـاس این اطلاعات اقداماتی را در جهت نیل به هدف(معمولا کسـب بیشـترین جایزه) انجام میدهد در ادامه به تشریح این پارادایم یادگیری میپردازیم.

---

[1] Machine Learning
[2] Supervised
[3] Unsupervised
[4] Reinforcement Learning
[5] Input
[6] Output/Label
[7] Regression
[8] Classification
[9] Clustering

یادگیری تقویت شده

چنانچه گفته شد، در این پارادیم، معمولا داده ای برای آموزش نداریم (چه به صورت برچسب دار یا بدون برچسب). در عوض دو عنصر مهم با نام های عامل[10] و محیط[11] داریم که عامل در اثر تعامل با محیط اطلاعات لازم برای آموزش را جمع آوری میکند (اصطلاحا تجربه کسب میکند). در این پارادایم، ماشین یادگیر سعی دارد که بهترین روش[12] تعامل با محیط را بیابد.



با توجه به نمودار بالا، ماشین یادگیر بر اساس سیگنالهای جایزه و وضعیت، عملیات a_t را انجام میدهد(که این عملیات میتواند از بین تعداد معدودی عملیات انتخاب شود) و جایزه[13] و وضعیت[14](یا مشاهده[15] که میتواند گسسته یا پیوسته باشد) جدید را از طرف محیط دریافت میکند در ابتدای کار، عامل یادگیر، ممکن است محیط را کاملا بشناسد و توصیف یا مدلی از ان را داشته باشد (این توصیف مثلا میتواند به صورت یک مدل مارکوف باشد). در این حالت اصطلاحا محیط را کاملا قابل مشاهده[16] میگویند. عامل ممکن است تنها بخشی[17] از محیط را بشناسد و یا اساسا هیچ شناخت و مدلی از محیط نداشته باشد[18].

چنانچه مدل محیط در دسترس باشد، برای یافتن سیاست بهیه[19] میتوان از روشهای برنامه نویسی پویا، مونته کارلو، TD , و.. استفاده کرده و در صورتی که مدل محیط دردسترس نباشد از الگوریتم هایی نظیر SARSA و Q-Learning استفاده میشود

در این پروژه ما با یک مسئله بدون مدل[20] مواجهیم و میخواهیم با استفاده از یک الگوریتم بهبود یافته برپایه Q-Learning ، سیاست بهینه را به عامل یادگیر آموزش دهیم لذا در حین تشریح پروژه این الگوریتم و ورژن های بهبود یافته آن را بررسی میکنیم.

---

[10] agent
[11] environment
[12] Optimal policy
[13] reward
[14] state
[15] observation
[16] Fully observable
[17] Partially observable
[18] Model free
[19] Optimal policy
[20] Model free

Q-function:

چنانچه گفته شد هدف ما در این مسئله اینست که عامل یادگیر باتوجه به وضعتی که دارد (یعنی موقعیت و سرعتش) عملیاتی را انجام دهد (یکی از سه عملیات تعریف شده) که عامل را به هدفش (که خارج شدن از گوال است) نزدیکتر کند. بنابراین به معیاری برای توصیف کارایی و بازده عملیات انتخاب شده نیازمدیم. این معیار را با تابع ***Q(s,a)*** نشان میدهیم این تابع وضعیت عامل یادگیر (*s*) و نیز عملیاتی که در آن وضعیت انتخاب کرده است (*a*) را دریافت کرده و عددی را به نشانه میزان موثر بودن این عملیات در جهت رسیدن به هدف بازمیگرداند. حال با توجه به این معیار میتوان عملیاتها را مقایسه کرد. اما این معیار چگونه باید تعریف شود؟

قطعا عملیاتی که عامل در هر وضعیت انجام میدهد را باید بر اساس اینکه در نهایت، عامل به هدف میرسد یا نه، مورد سنجش و ارزیابی قرار گیرد. به عبارتی، در ارزیابی عملیات انجام شده در هر وضعیت باید به آینده نیز توجه داشت بر این اساس تابع *Q* به صورت زیر تعریف میشود:

$$Q(s,a) = E_\pi\{R_{t+1} + R_{t+2} + R_{t+3} + \cdots | S_t = s, A_t = a\}$$

که یعنی اگر در وضعیت *s* عملیات *a* انجام شود و پس از آن نیز عملیات ها بر اساس سیاست π انتخاب شود درنهایت (یا پس از طی تعداد مشخصی ***state***) مقدار مورد انتظار مجموع جایزه های دریافتی چقدر است

منظور از *Q-Learning* پیدا کردن تابع *Q* است!

Q-Learning:

گفتیم که تابع *Q* را نداریم و لذا باید آنرا تخمین بزنیم. همچنین برای ذخیره کردن این تابع نیز باید راه حلی داشته باشیم .مثلا اگر مسئله ساده باشد میتوان مقادیر *Q* را در جدول ذخیره کرد یا اگر برای آن شکل پارامتری در نظر بگیریم و پارامترهای آنرا تخمین بزنیم، ذخیره این پارامترها کافیست

Q-Learning الگوریتمیست که به منظور تخمین و یافتن تابع *Q* استفاده میشود. در این الگوریتم آپدیت کردن مقادیر *Q* به صورت زیر انجام میپذیرد:

$$\boldsymbol{Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma max_a Q(s', a) - Q(s,a)]}$$

در رابطه بالا α : نرخ ببیادگیری γ: ضریب ***discount***

R: جایزه ایست که عامل یادگیر با انجام عملیات *a* وقتی در وضعیت *S* قرار دارد دریافت میکند

γ ***maxQ(s',a)***: بیانگر حداکثر مجموع جایزه قابل انتظار است وقتی عامل از وضعیت *s'* تا وضعیت نهایی پیش رود. بنابراین جمع این دو ترم به نوعی بیانگر حد اکثر جایزه مورد انتظار است که عامل یادگیر میتواند با عبور از وضعیت جاری(*s*) به دست آورد. از طرفی تابع *Q* ، جایزه مورد انتظار را برای عملیات *a* در وضعیت *s* مشخص میکند. بنابراین اگر مقادیر *Q* به عبارت ***R+ γ*maxQ(s',a)*** میل کند(از اینرو به ترم مذکور هدف یا ***TD target*** میگویند). *Q* حاصله را میتوان بهترین تابع *Q* ممکن نامید.

Greedy action

حال با فرض داشتن تابع Q، *عامل یادگیر چگونه عملیات متناسب با وضعیت جاری را انتخاب کند؟ فرض میکنیم که روش انتخاب، یک تابع مثلا* π (s) *باشد یعنی این تابع وضعیت* s *را دریافت کرده وعملیات مناسب در وضعیت* s *را برگرداند.یک روش آنست که در هر وضعیت، عملیاتی انتخاب شود که بیشترین* Q-value *را داشته باشد دراین صورت داریم:*

$$\pi(s) = argmax_a Q(s,a)$$

این روش انتخاب را، حریصانه مینامند. اما اگر عملیاتی که بیشترین Q-value *را دارد لزوما بهترین عملیات نباشد چه؟ پیروی از سیاست حریصانه راه را برای آزمودن سیاستهای دیگر میبندد (در این حالت اصطلاحا مدل فقط در حال بهره برداریست!) در مقابل، عامل یادگیر میتواند به امید یافتن سیاستی بهتر، در فضای عملیاتها اقدام به کاوش کند هرچند، نتیجه این کاوش لزوما بهتر از سیاست حریصانه نخواهد بود.*

Є-Greedy action:

در انتخاب عملیات به صورت حریصانه، عملا احتمال انتخاب دیگر عملیات ها صفر است و لذا کاوشی نداریم. برای اینکه عامل یادگیر امکان کاوش در محیط را داشته باشد باید بخشی از احتمال انتخاب عملیات حریصانه را بین سایر عملیات ها تقسیم کنیم... ! Є-Greedy action *به صورت زیر، احتمال را بین عملیاتها تقسیم میکند:*

$$\pi(s)_{\epsilon-greedy} = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & if\ a = argmax_a Q(s,a) \\ \frac{\epsilon}{m} & others \end{cases}$$

یعنی با احتمال $(\frac{\epsilon}{m} + 1 - \epsilon)$ *عملیات حریصانه انتخاب میشود و با احتمال* $\frac{\epsilon}{m}$ هم بقیه عملیات ها

اما پیاده سازی روش فوق چگونه است؟

پیاده سازی انتخاب با احتمال دلخواه

ایده کلی بدین صورت است که، اگر متغیر تصادفی X دارای توزیع یکنواخت در بازه (0,1) باشد آنگاه داریم

$$P(X > \varepsilon) = \int_{\epsilon}^{1} 1dx = 1 - \varepsilon$$

به عبارت دیگر یک نمونه از این توزیع یکنواخت، با احتمال $1 - \varepsilon$ از ε بزرگتر است لذا برای پیاده سازی روش Є-Greedy action *به صورت زیر عمل میکنیم:*

1. ابتدا یک نمونه(عدد) از توزیع یکنواخت در بازه (0,1) تولید میکنیم
2. اگر نمونه تولید شده از ε بیشتر باشد آنگاه عملیات حریصانه را انتخاب کن
3. در غیر این صورت به صورت رندوم یک عملیات را انتخاب کن

Deep Q-Learning (DQN):

در الگوریتم *Q-Learning* تابع Q را میتوان با یک شبکه عصبی تقریب زد. این حالت را Deep Q-Learning *می گویند*

overestimation

عامل یادگیر در ابتدای کار هیچ شناختی از محیط ندارد لذا باید با استفاده از اطلاعاتی که در اثر تعامل با محیط جمع آوری میکند فرایند آموزش را پیش ببرد، فقدان شناخت اولیه، منجر به جمع آوری داده هایی میشود که عموما زائد و حاوی درصد زیادی نویز است. در الگوریتم Q-Learning *معمولی، عامل یادگیر به کمک تابع* Q *و بر اساس روش حریصانه، عملیات بهینه را انتخاب میکند و سپس ارزشیابی و آپدیت کردن تابع* Q *نیز بر اساس* TD target *اتفاق می افتد ولی* TD target *خود به* Q *وابسته است و با تغییر تابع* Q *تغییر میکند لذا همگرایی* Q *به حالت بهینه بسیار دشوار خواهد بود . از طرفی با توجه به نویز بالای داده های جمع آوری شده، عملیاتی که بیشترین* Q-value *را دارد لزوما*

عملیات بهینه نخواهد بود که مورد آخر در کنار سایر موارد میتواند منجر به پدیده ای موسوم به overestimation *شود. به طول خلاصه اگر مدل دچار* overestimation *شود، در این صورت در هر وضعیت عملیاتی انتخاب میشود که دارای بیشترین* Q-value *است ولی بهینه نیست!*

به منظور حل این مشکل راهکارهای مختلفی ارائه شده است.یکی از این راهکارها استفاده از دوتابع Q *و* Q' *یکی برای انتخاب عملیاتها و دیگری برای ارزشیابی عملیات این ایده راه* Double Q-Learning *گویند..*

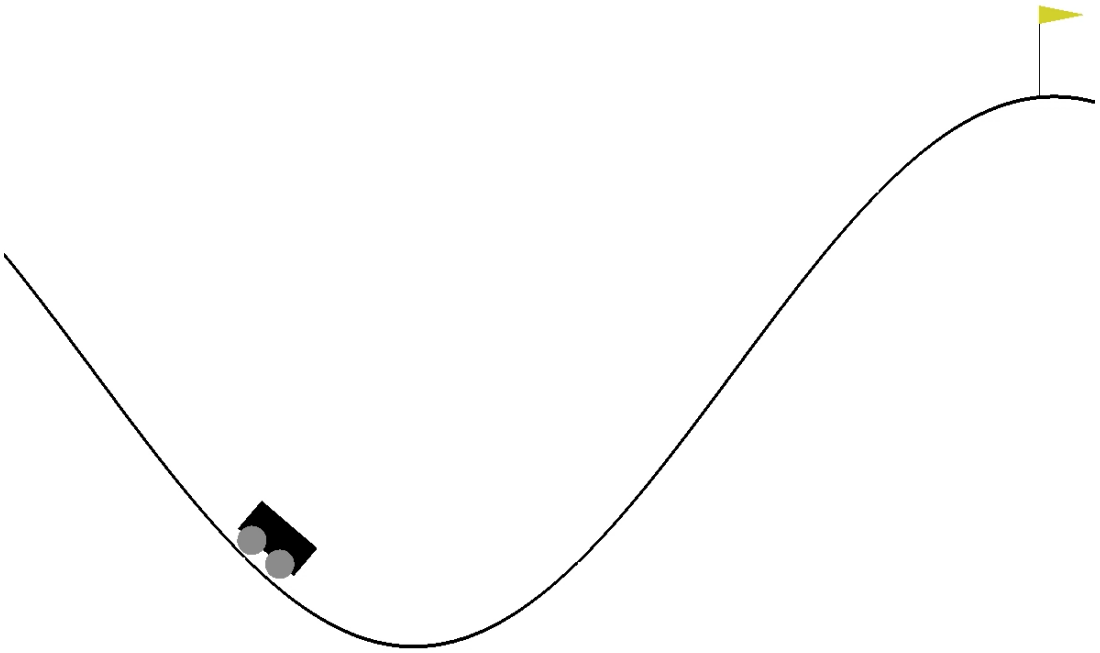
Double DQN(DDQN)

همان Double Q-Learning *است که توابع* Q *و* Q' *را با شبکه های عصبی تقریب زده میشود. بنابراین در این الگوریتم باید دوتا تابع* Q *و* Q' *را پیدا کنیم که این امر پیچیدگی محاسباتی مسئله را بیشتر میکند لذا به منظور کاهش بار محاسباتی از روشی که در (مقاله پیوست شده پروژه) تشریح شده است استفاده میکنیم در این روش، اگرچه دوتا تابع داریم (دوتا شبکه عصبی) ولی فقط یکی از آنها آموزش داده میشود(*train_newtwork*) و پس از تعداد مشخصی گام آموزشی، وزنهای شبکه آموزش دیده به شبکه دوم(*target_network*) منتقل میشود.*

بخش دوم

تشریح مسئله

مسئله Mountain Car یکی از اولین و مشهورترین مسائل در حوزه یادگیری تقویتی است. در این مسئله عامل، ماشینی است که در یک گودال (محیط مسئله) گیر افتاده و باید در طی تعامل با محیط خود را به نقطه هدف (نشانه داده شده با پرچم) برساند اما چون نیروی موتور این ماشین برای غلبه بر گرانش کافی نیست نمیتواند با حرکت مستقیم در جهت هدف ، خود را به هدف برساند. بنابراین ماشین باید ابتدا در جهت عکس هدف از شیب مقابل بالا رفته و پس از کسب انرژی پتانسیل لازم خود را رها کند و در نهایت در نقطه ای مشخص، با روشن کردن موتور خود را به هدف برساند!... هر اپیزود از این مسئله با انجام 200 عملیات و یا رسیدن به نقطه هدف، پایان میابد



بر اساس کدهای مربوطه به این مسئله در gym، گودال درواقع یک منحنی سینوسی به معادله زیر است:

$$y = 0.45 * \sin(3 * x) + 0.55 \qquad -1.2 < x < 0.6$$

و لذا با توجه به رابطه بالا که منحنی مسیر هم نامیده میشود رابطه سرعت باید کسینوسی باشد. در این مسئله، عامل یادگیر میتواند یکی از سه عملیات زیر را انجام دهد:

***0**. حرکت به چپ، **1**. حرکت نکند، **2**. حرکت به راست*

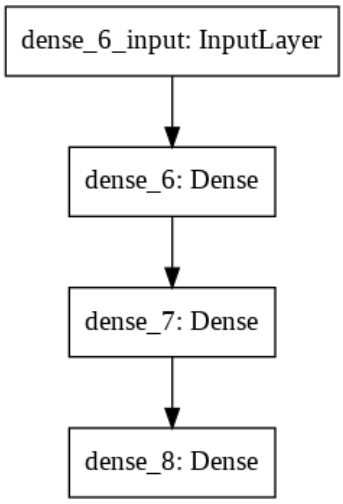
عامل یادگیر پس از انجام هر عملیاتی جایزه -1 دریافت میکند مگر زمانی که عملیات انجام شده منجر به رسیدن به هدف شود که در این حالت جایزه 0 را دریافت میکند همچنین وضعیت(State) نیز به ترتیب شامل دو مولفه مکان عامل و سرعت آن میباشد

$$state = (-1.2 < position < 0.6, 0.07 < velocity < 0.07)$$

هدف از حل مسئله، یافتن دنباله ای از عملیاتهای مجاز (عملیات های 0 و 1 و 2) است که دنبال کردن آن منجر به خروج ماشین از گودال شود (بهترین راه حل هم دنباله ایست که منجر به بیشترین جایزه شود). در ادامه با طراحی آموزش یک شبکه عصبی سعی میکنیم Q را برای این مسئله را بیابیم

شبکه های عصبی به عنوان تقریبی از تابع سیاستگذاری (Policy function)

در این بخش دو شبکه عصبی برای تقریب زدن توابع Q و Q' میسازیم. شبکه model_for_train معادل Q و شبکه model_for_predict معادل Q' است. ورودی این شبکه ها زوج مرتب (مکان, سرعت) است که هردو کمیتی پیوسته اند همچنین خروجی شبکه ها نیز یک بردار سه مولفه ای است که مولفه های آن به ترتیب ارزش عملیاتهای 0 1 2 را مشخص میکنند . چون ارزش هر عملیات، کمیتی پویسته است لذا با یک مسئله رگرسیون مواجهیم . ساختار کلی هر دو شبکه یکسان بوده و به صورت زیر است:




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 32)                96
_________________________________________________________________
dense_10 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 195
=================================================================
Total params: 2,403
Trainable params: 2,403
Non-trainable params: 0
_________________________________________________________________
```

چنانچه گفته شده تنها وزنهای شبکه model_for_train آپدیت شده و پس چند مرتبه آپدیت این وزنها به مدل دیگر انتقال میابد برای آپدیت کردن هم از رابطه زیر به عنوان target شبکه استفاده میشود:

$$Y^{DDQN} = R_{t+1} + \gamma * model\_for\_predict(S_{t+1}, argmax_a model\_for\_train(S_{t+1}, a))$$

چون مسئله ما از نوع رگرسیون است از معیار MSE به عنوان تابع هزینه استفاده شده است و الگوریتم بهینه سازی هم Adam است.

در کادر زیر تعدادی از هایپرپارمترهای مسئله آورده شده است

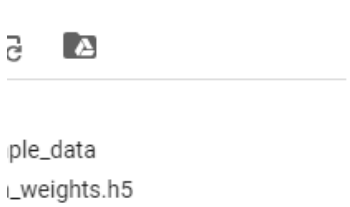
```
#optimizer : Adam
#loss function :MSE
eps = 0.99 #epsilon for greedy action
eps_decay = 0.005 #decay rate for epsilon per episode
gamma = 0.9 #discount factor
alpha = 0.01 #learning rate
n_episodes = 1500 #number of episodes
n_steps = 201 #max step to reach the goal
in_dim = 2 #input dim of the networks (position,velocity)
out_dim = 3 #output dim of the networks Q-values for (0,1,2) actions
batch_size = 32
```

چون عامل یادگیر در ابتدای کار هیچ شناختی از محیط ندارد لذا بهتر است در اپیزودهای اولیه حداکثر کاوش را داشته باشیم به همین دلیل مقدار اولیه اپسیلون 0.99 انتخاب شده است و در هر اپیزود 0.005 از آن کم میشود تا نهایتا با 0.01 برسد.

*) چون امکان استفاده از env.render() در گوگل کولب وجود نداشت! ، عامل یادگیر در گوگل کولب آموزش داده شده و سپس وزنهای حاصله در سیستم شخصی آزمایش شده است (فایل test.py)

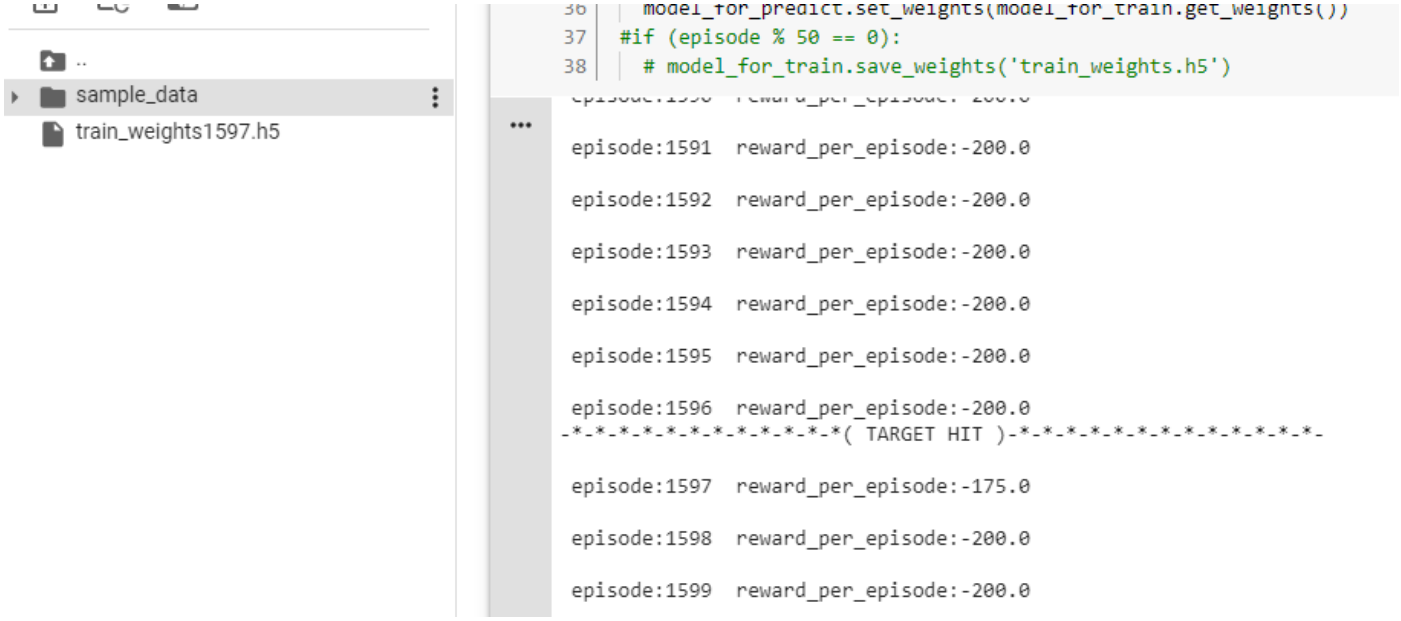
*) فرایند یادگیری چندین بار و به طور کامل از اول اجرا شده است (یعنی تمام متغییر ها در مقادیر پیشفرضشان بوده اند و memory هم خالی بوده است) و هربار پس از تعداد نامخشصی اپیزود، جواب مسئله به دست آمد. بنابراین میتوان نتیجه گرفت که الگوریتم اسفاده شده نسبت به شرایط اولیه بسیار حساس است.

*) در اولین اجرا، در اپیزود 1301 به جواب رسیدم (عبارت TARGET HIT قبل از اعلام شماره اپیزوده پرینت میشود!) و پس از ان در چندین اپیزود بعدش هم، عامل یادگیر به هدف رسیده است. اما با ادامه فرایند یادگیری، وضعیت بهتر نمیشود.




```
episode:1297  reward_per_episode:-200.0
episode:1298  reward_per_episode:-200.0
episode:1299  reward_per_episode:-200.0
episode:1300  reward_per_episode:-200.0
-*-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-
episode:1301  reward_per_episode:-124.0
-*-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-
episode:1302  reward_per_episode:-175.0
-*-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-
episode:1303  reward_per_episode:-176.0
episode:1304  reward_per_episode:-200.0
```

در اجرای دوم، پس از 1597 اپیزود هدف مشاهده شد. (و وزنهای آن ذخیره شد)



فرایند تا اپیزود 2631 ادامه داشـــته و در این اپیزود گوگل کولب قطع شـــد اجرای دوباره فرایند اما اینبار با بارگذاری وزنهای ذخیره شـــده اپیزود1597 منجر به نتایج زیر شد:

```
 episode:309  reward_per_episode:-190.0
-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-

 episode:310  reward_per_episode:-150.0
-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-

 episode:311  reward_per_episode:-121.0
-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-

 episode:312  reward_per_episode:-179.0

 episode:313  reward_per_episode:-200.0
-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-

 episode:314  reward_per_episode:-154.0
-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-

 episode:315  reward_per_episode:-157.0
-*-*-*-*-*-*-*-*-*-*-*( TARGET HIT )-*-*-*-*-*-*-*-*-*-*-*-*-

 episode:316  reward_per_episode:-162.0
```

یعنی در همان اپیزودهای ابتدایی (اپیزود چهار تا بیست و بست و دو) و سپس در اپیزودهای بالاتر از 300 به طور مداوم عامل به هدف میرسید و وزنهای مربوطه هم ثبت شده است(هرچند بازهم با ادامه فرایند، عامل از هدف دور شد). دلیل این امر احتمالا خالی شدن memory *باشد و به تعبیری فراموشی مدل است!...در ادامه به بررسی این اثر میپردازیم*

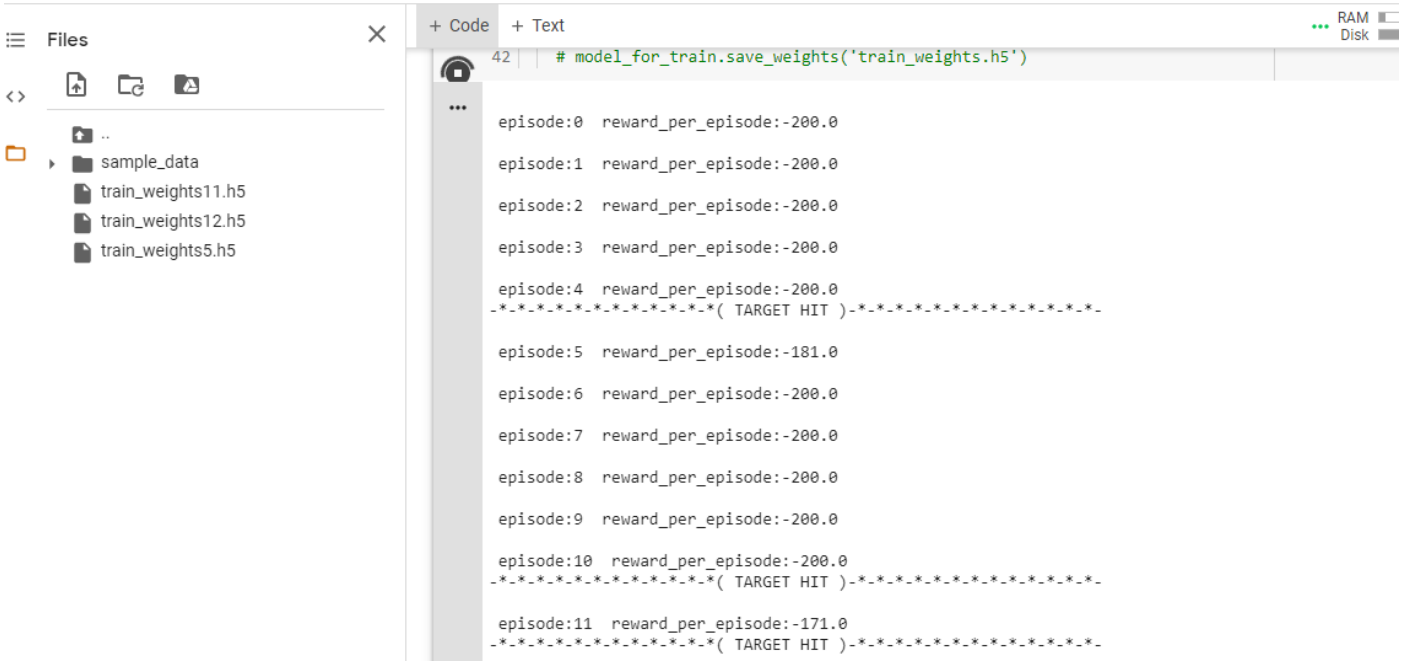
نقش مکانیزم فراموشی :

عامل یادگیر نتیجه تعاملات خود با محیط را در حافظه خود! (متغیر memory*) ذخیره میکند اما چنانچه گفته شد بخش قابل توجهی از این اطلاعات ذخیره شده زائد و همراه با نویز زیاد است لذا احتمالا فراموش کردن برخی از این اطلاعات منجر به بهبود وضعیت یادگیری شود.*

به منظور ایجاده مکانیزمی برای فراموشی از کد زیر در فرایند یادگیری استفاده شد. این کد، حافظه را پس از هر 512 داده ثبت شده پاک میکند

```
if(len(memory) == 512):
    memory.clear()
```

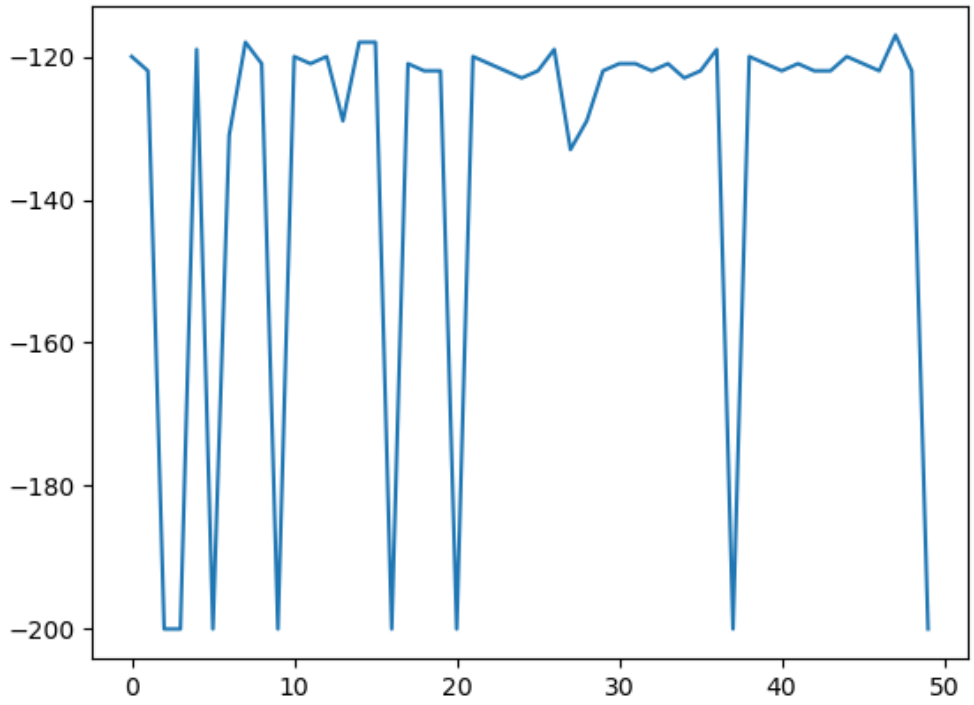
در اولین اجرا نتیجه زیر حاصل شد:



یعنی پس از پنج اپیزود جواب به دست آمد و سپس در اپیزدهای 11 و 12 هم عامل به هدف رسید! نتیجه شایسته توجه است!... لذا چندین بار و از نو الگوریتم اجرا شد اما متاسفانه همچین نتیجه ای به دست نیامد لذا احتمالا نتیجه اجرای اول به خاطر مقدار اولیه وزنها بوده است. ولی همگرایی مدل بهتر شده و در 1100 به هدف رسید (که این هم میتواند اتفاقی باشد!)..

نتیجه گیری

چنانچه گفته شد، فرایند آموزش شبکه و یافتن وزنها در گوگل کولب انجام شده است اما در کولب امکان استفاده از Render به صورت عادی وجود نداشت ! لذا نتیجه حاصله را در سیستم شخصی آزمایش کرده ام. بدین منظور از کدهای فایل (test.py) استفاده شده است. نمودار زیر،جایزه دریافتی به ازای پنجاه اپیزود را نشان میدهد. مشاهده میشود که وزنهای پیوست شده در اکثر مواقع منجر به نتیجه قابل قبول میشود!



بهترین نتیجه حاصله، جایزه حدود -120 *است.*

در این پروژه با برخی از الگوریتم های مطرح در یادگیری تقویتی آشـــنا شـــدیم و عوامل تاثیر گذار بر این الگوریتمها را شـــناختیم و همچنین با کتابخانه *gym* که ابزاری آماده برای شبیه سازی الگوریتمهای حوزه یادگیری تقویتی است نیز آشنا شدیم. سپس یکی از این الگوریتم ها را پیاده سازی کرده و مشکلات پیاده سازی عملی آنها را بررسی نمودیم