دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره ۵

درس یادگیری تعاملی

پاییز ۱۴۰۱

محمد سعادتی

۸۱۰۱۹۸۴۱۰

فهرست

چکیده

با توجه به پیشرفت روز افزون مدل های یادگیری عمیق و همچنین نتایج قابل توجه ادغام این مدل ها با کاربردهای یادیگیری تقویتی، در این تمرین به بررسی این الگوریتم ها و مدل ها خواهیم پرداخت. حل کردن مسائل یادگیری تقویتی با استفاده از شبکه های عصبی پیشینه ای قدیمی دارد و با توجه به پیشرفت سخت افزارهای محاسباتی در دو دهه اخیر، سرعت توسعه مدل های عمیق برای مسائل یادگیری تقویتی افزایش قابل ملاحظه ای داشته است. استفاده از شبکه های عصبی این امکان را به ما می دهد که از مسئله را با استفاده از یک مدل end to end حل کنیم. با توجه به این نکته کسب مهارت کارکردن با مدل های یادگیری عمیق و حل مسائل یادگیری تقویتی با استفاده از این مدل ها از مهارت های ضروری در زمینه یادگیری تقویتی می باشد. این تمرین مقدمه آشنایی ما با این مسائل را فراهم می کند.

سوالات تحلیلی

۱-۱

الگوریتم پالیسی گردینت یک تکنیک یادگیری تقویتی برای بهینه سازی سیاست های کنترلی در محیط های پیچیده است. از بازخورد محیط برای بهبود یک سیاست با تنظیم آن استفاده می کند تا اکشن ها با expected reward بالاتر احتمال انتخاب شدن بیشتری داشته باشند. هدف، به حداکثر رساندن expected reward به دست آمده در هنگام پیروی از سیاست آموخته شده است.

الگوریتم پالیسی گردینت یک رویکرد تکرار سیاست است که در آن سیاست مستقیماً مدل‌سازی و بهینه‌سازی می شود تا به سیاست بهینه ای برسد که expected return را به حداکثر می رساند. این سیاست معمولاً با یک تابع پارامتر شده با توجه به θ , $\pi_\theta(a|s)$ مدل می شود. مقدار تابع پاداش (هدف) به این سیاست بستگی دارد و سپس می توان الگوریتم های مختلفی را برای بهینه سازی برای بهترین پاداش اعمال کرد. این نوع الگوریتم‌ها یادگیری تقویتی model-free هستند. [3] [1]

روش‌های پالیسی گردینت نوعی از تکنیک‌های یادگیری تقویتی هستند که بر بهینه‌سازی سیاست‌های پارامتری با توجه expected return (پاداش تجمعی بلندمدت) با نزول گرادیان تکیه دارند. آنها از بسیاری از مشکلاتی که رویکردهای یادگیری تقویتی سنتی را مخدوش کرده اند، مانند فقدان تضمین یک value function ، مشکل حل نشدنی ناشی از اطلاعات استیت نامشخص و پیچیدگی ناشی از استیت ها و اکشن های مستمر، رنج نمی برند. روش‌های پالیسی گردینت پارامتر سیاست را در هر مرحله در جهت برآورد گرادیان عملکرد با توجه به پارامتر سیاست به‌روزرسانی می‌کنند. این روش ها یک سیاست پارامتری را یاد می‌گیرند که امکان انجام اکشن ها را بدون مشورت با تخمین‌های action-value می‌دهد. روش پالیسی گردینت به طور مکرر وزن‌های شبکه سیاست را (با به‌روزرسانی‌های نرم) اصلاح می‌کند تا جفت‌های اکشن-استیت که منجر به بازده مثبت شده‌اند، و جفت‌های اکشن-استیت که منجر به بازده منفی شده‌اند، کمتر محتمل‌تر شوند. [4] [2]

۲-۱

- مزایا:
  - الگوریتم‌های یادگیری تقویتی عمیق مزایای زیادی نسبت به الگوریتم‌های یادگیری تقویتی سنتی دارند، مانند یادگیری از ورودی‌های high-dimensional sensory ، تعمیم بهتر به موقعیت‌های جدید و توانایی تصمیم‌گیری طولانی‌مدت. روش‌های یادگیری عمیق

همچنین مقیاس‌پذیری بهبود یافته و پایداری بهتر را هنگام برخورد با مجموعه داده‌های بزرگ ارائه می‌دهند. در نهایت، الگوریتم‌های یادگیری تقویت عمیق می‌توانند فضاهای حالت پیچیده و عملی را با دقت بیشتری نسبت به همتایان کم عمق خود نشان دهند.

- استفاده از یادگیری عمیق در مسائل مربوط به high-dimensional state space بیشترین کاربرد را دارد. این بدان معناست که با یادگیری عمیق، یادگیری تقویتی به دلیل توانایی خود در یادگیری سطوح مختلف abstractions از داده ها، قادر به حل وظایف پیچیده تر با دانش قبلی پایین تر است. در الگوریتم های Deep RL پارامترها در طول زمان تنظیم می شوند و تعداد بهترین پارامترهای ممکن به سرعت با همگرا شدن مدل کاهش می یابد. [7] [5]

- معایب:
  - عیب اصلی الگوریتم‌های RL عمیق این است که برای درست کار کردن به مقدار زیادی داده و تجربه نیاز دارند. این می تواند پرهزینه و وقت گیر باشد، به خصوص زمانی که با وظایفی با پیچیدگی بالا سر و کار دارید. علاوه بر این، الگوریتم‌های RL عمیق هنوز هم مستعد بیش از حد overfitting هستند، که ممکن است منجر به عملکرد تعمیم ضعیف در نقاط داده نادیده شود. در نهایت، الگوریتم‌های RL عمیق نیز می‌توانند در همگرایی‌های خود ناپایدار باشند به دلیل مسائل چالش برانگیز بهینه‌سازی مرتبط با آنها.
  - Deep RL از نظر تئوری تضمین نمی شود که کار کند. می توان نشان داد که واگرا می شود، به این معنی که وزن ها تا بی نهایت منفجر می شوند. هرگز نمی توان تضمین داد که اوزان در یک کار جدید تا بی نهایت از هم جدا نمی شوند. این واگرایی در تنظیمات خطی زمانی اتفاق نمی‌افتد که از شبکه عمیق استفاده نمی‌شود، بلکه از بردار وزن‌ها استفاده می‌کند. [6]
  - Deep RL به زمان بسیار زیادی برای آموزش نیاز دارد. [6]
  - برای اینکه نسبت به سایر تکنیک ها بهتر عمل کند به حجم بسیار زیادی داده نیاز دارد. [8]
  - به دلیل مدل های پیچیده داده، آموزش بسیار گران است. [8]

۳-۱

بافرهای تجربه در یادگیری تقویتی عمیق (RL) برای ذخیره و پخش مجدد تجربیات گذشته استفاده می شوند. این تکنیک در خدمت این است که یک عامل را قادر می سازد تا با استفاده از داده های جمع

آوری شده از تعاملات قبلی با محیط، کارآمدتر و سریع تر یاد بگیرد. بافرهای تجربه فرآیند آموزش را با کاهش میزان تعاملات زمان واقعی که یک عامل باید انجام دهد، تسریع می‌کند، زیرا می‌تواند از حافظه استخراج کند و تجربیات گذشته را دوباره پخش کند. این روش همچنین sample efficiency را با اجازه دادن به آموزش مجدد مکرر بر اساس اطلاعاتی که قبلاً به دست آورده‌اید، به جای جمع‌آوری مداوم داده‌های جدید، بهبود می‌بخشد.

ایده اصلی پشت experience replay، ذخیره تجربیات گذشته و سپس استفاده از زیرمجموعه‌ای تصادفی از این تجربیات برای به‌روزرسانی Q-network است، نه تنها استفاده از آخرین تجربه. ما می توانیم با استفاده از یک بافر بزرگ از تجربیات گذشته خود و نمونه داده های آموزشی از آن، به جای استفاده از آخرین تجربه خود، از نوسان یا واگرایی فاجعه بار مقادیر عمل جلوگیری کنیم. این تکنیک بافر تکرار یا بافر تجربه نامیده می شود. بافر پخش مجدد شامل مجموعه ای از تاپل های تجربه (S، A، R، S′) است. تاپل ها به تدریج به بافر اضافه می شوند که ما در حال تعامل با محیط هستیم. ساده ترین پیاده سازی یک بافر با اندازه ثابت است که داده های جدیدی به انتهای بافر اضافه می شود تا قدیمی ترین تجربه را از آن خارج کند. بافرهای بازپخش تجربه معمولاً برای الگوریتم‌های یادگیری تقویتی off-policy با گرفتن تمام نمونه‌های تولید شده توسط یک عامل در تعامل با محیطش و سپس ذخیره آنها برای استفاده مجدد بعدی اعمال می‌شوند و به روش یادگیری ما اجازه می دهد چندین بار روی آنها به روز شود. [10] [9]

مسئله پیاده سازی

در این تمرین ما با محیط معروف highway کار خواهیم کرد. این محیط در کتابخانه gym پیاده سازی شده است و تسک های مختلفی از جمله تسک پارکینگ، عبور از چهار راه و ... روی این محیط تعریف شده است. مسائل موجود در این محیط مربوط به تسک autonomous drive در یک محیط دو بعدی می باشند.

بخش اول

زیر بخش ۱

- استیت: استیت به عنوان یک شبکه ۴×۴ تعریف می شود که موقعیت فعلی ماشین ها و جاده خالی را نشان می دهد. هر خودرو موقعیت شروع منحصر به فرد خود را دارد که با عدد صحیح بین ۰ تا ۱۵ نشان داده می شود که نشان می دهد در حال حاضر در کدام مربع است. مربع ها خالی هستند مگر اینکه ماشینی آن را اشغال کند. استیت عامل نشان دهنده وضعیت Observations محیط، سرعت، وضعیت تصادف، اکشن انتخاب شده، پاداش ها، وضعیت خاتمه برنامه و وضعیت truncated می باشد.
- اکشن: شامل ۵ عمل تغییر لاین به چپ، انجام یک اکشن غیرقابل دسترس (عمل غیر مجاز) (انجام هیچ کاری) (IDLE) ، تغییر لاین به راست، افزایش سرعت و کاهش سرعت می باشد.
- پاداش: به طور کلی بر دو ویژگی متمرکز شده است: یک وسیله نقلیه باید به سرعت در جاده پیشرفت کند. از برخورد اجتناب کنید. بنابراین، تابع پاداش اغلب از یک جمله سرعت و یک جمله برخورد تشکیل شده است:

$$R(s,a) = a\frac{v - v_{\min}}{v_{\max} - v_{\min}} - b\,\text{collision}$$

که v Vmin Vmax به ترتیب سرعت فعلی، حداقل و حداکثر سرعت خودروی ego هستند و a b دو ضریب هستند. در برخی محیط ها، وزن جریمه برخورد را می توان از طریق پارامتر collision_penalty پیکربندی کرد.

زیر بخش ۲

- اکشن: اکشن ها در این تسک در حالت پیش فرض از نوع DiscreteMetaAction هستند.
  - اکشن پیوسته: نوع ContinuousAction به عامل اجازه می دهد تا به طور مستقیم کنترل های سطح پایین سینماتیک خودرو، یعنی دریچه گاز a و زاویه فرمان δ را تنظیم کند. (کنترل دریچه گاز و فرمان را می توان به ترتیب از طریق تنظیمات طولی و جانبی فعال یا غیرفعال کرد. بنابراین، فضای عمل می تواند یک بعدی یا دو بعدی باشد.)
  - اکشن گسسته: DiscreteAction یک کوانتیزاسیون یکنواخت از ContinuousAction فوق است. پارامتر actions_per_axis اجازه می دهد تا مرحله کوانتیزاسیون را تنظیم کرد.

مشابه اکشن های پیوسته، محور طولی و جانبی را می توان به طور جداگانه فعال یا غیرفعال کرد.

- Meta-Actions گسسته: نوع DiscreteMetaAction لایه‌ای از کنترل‌کننده‌های سرعت و فرمان را در بالای کنترل سطح پایین پیوسته اضافه می‌کند، به طوری که خودروی ego می‌تواند به طور خودکار جاده را با سرعت دلخواه دنبال کند. سپس، متا اکشن های موجود شامل تغییر خط هدف و سرعت است که به عنوان نقطه تنظیم برای کنترل کننده های سطح پایین استفاده می شود.

- استیت: استیت ها در این محیط در حالت پیش فرض گسسته هستند. برای observations در این محیط در حالت پیش فرض از Kinematics استفاده شده است.KinematicObservation یک آرایه V × F است که فهرستی از V خودروهای مجاور را با مجموعه‌ای از ویژگی‌های اندازه F، که در قسمت پیکربندی «features» فهرست شده‌اند، توصیف می‌کند. برای مثال:

Vehicle	x	y	v_x	v_y
ego-vehicle	5.0	4.0	15.0	0
vehicle 1	-10.0	4.0	12.0	0
vehicle 2	13.0	8.0	13.5	0
...	...	...	...	...
vehicle V	22.2	10.5	18.0	0.5

بخش دوم

هدف سوال

در این بخش قرار است مسئله را با استفاده از الگوریتم Deep Q-Learning حل کنیم. برای پیاده سازی الگوریتم مطابق شبه کد آن در مقاله داده شده عمل می کنیم: [11]

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

توضیح پیاده سازی

کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت **Part 2 – Deep Q-Learning** قرار دارد.

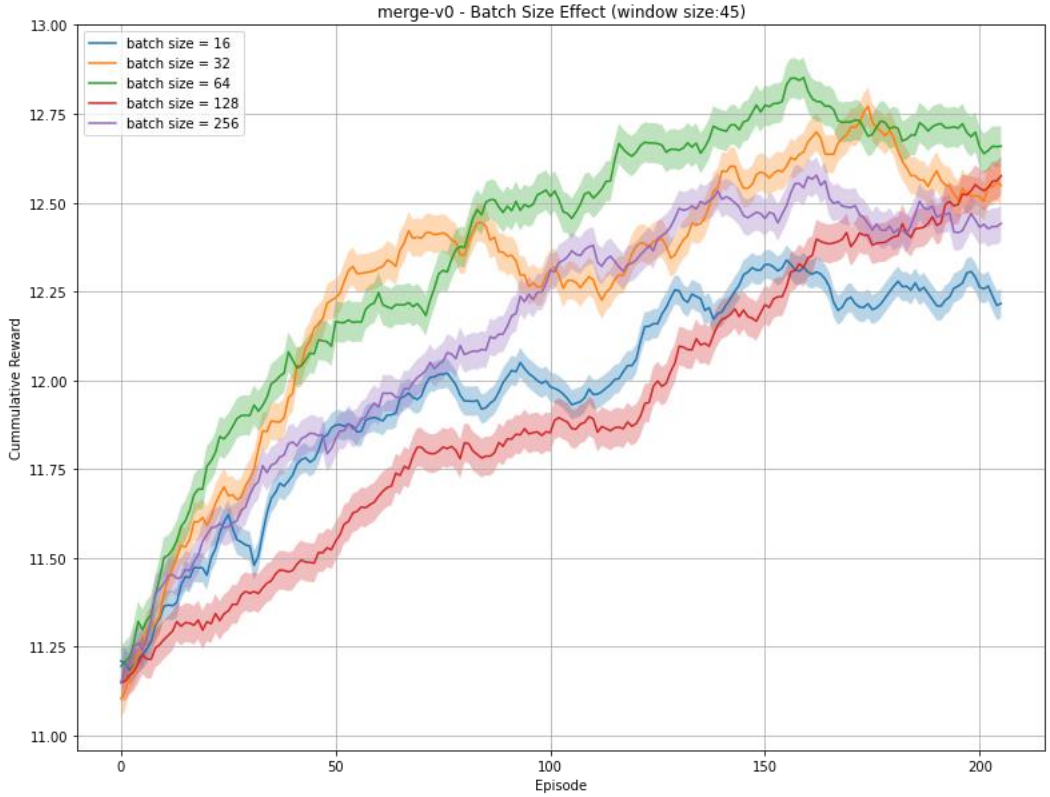
- در کلاس **ExperienceBuffer** بافر تجارب پیاده سازی شده است.
- در کلاس **Network** قسمت شبکه عصبی الگوریتم پیاده سازی شده است.
- در کلاس **DeepQLearning** الگوریتم Deep Q-Learning پیاده سازی شده است.
- تابع **run_ DeepQLearning _agent** الگوریتم Deep Q-Learning را با ۱۰ بار تکرار و به تعداد ۲۵۰ ایپزود اجرا می کند و پاداش های دریافت شده را بر می گرداند.

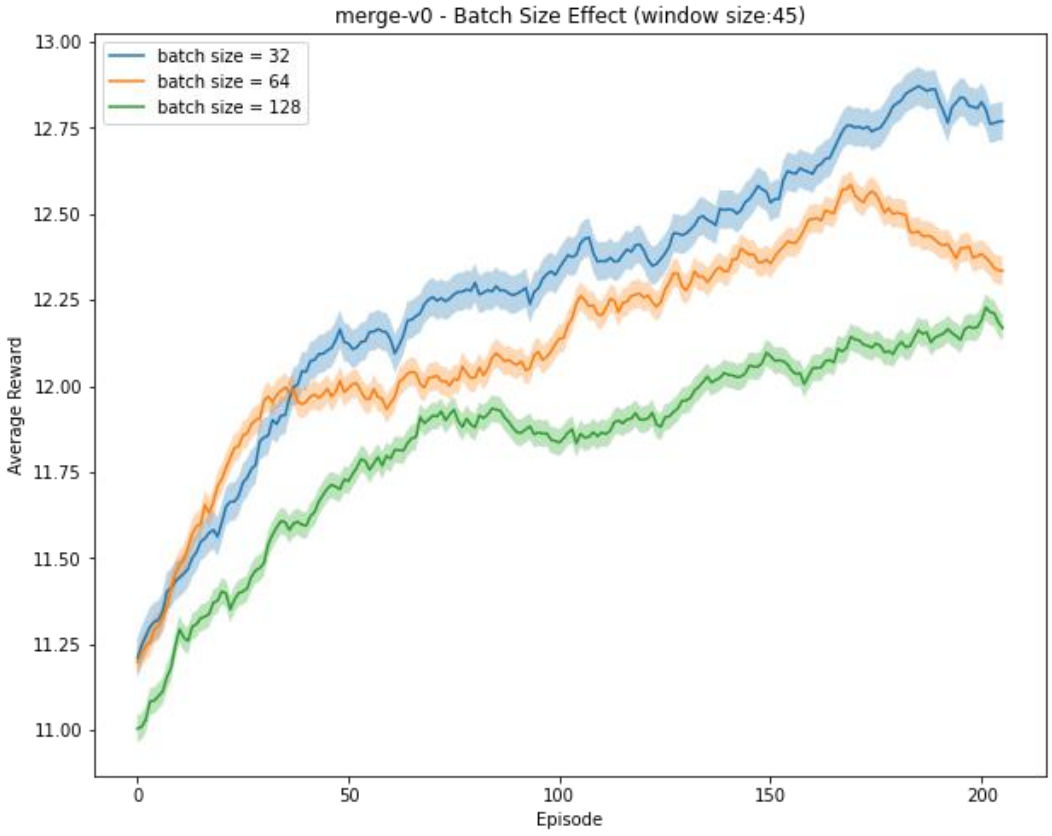
برای اجرا کد های این بخش سوال است تا کد های بخش ابتدایی و کد های این بخش را اجرا کنید.

نتایج و پاداش

کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت **Part 2 – Reward** قرار دارد.

نمودار پاداش کسب شده توسط عامل در طول یادگیری برای ۱۰ بار اجرا الگوریتم به ازای ۲۵۰ ایپزود به همراه بازه اطمینان ۹۵ درصد برای مقادیر مختلف به عنوان batch_size به شکل زیر است:

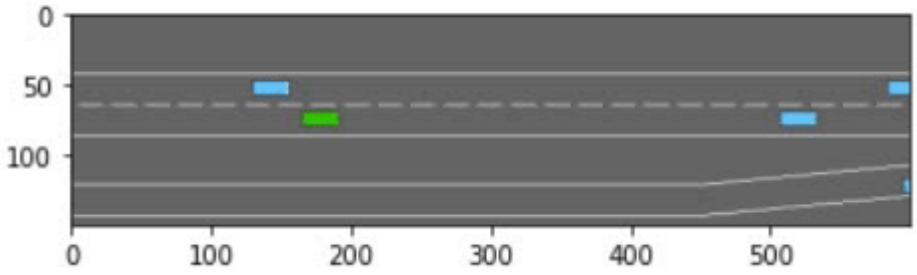




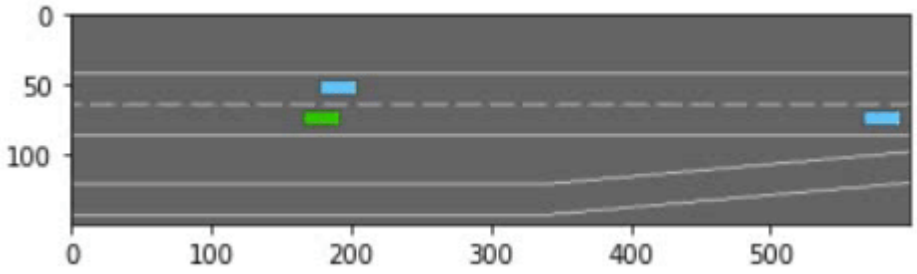
ارائه ویدیو (render)

کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت **Part 2 – Render** قرار دارد.

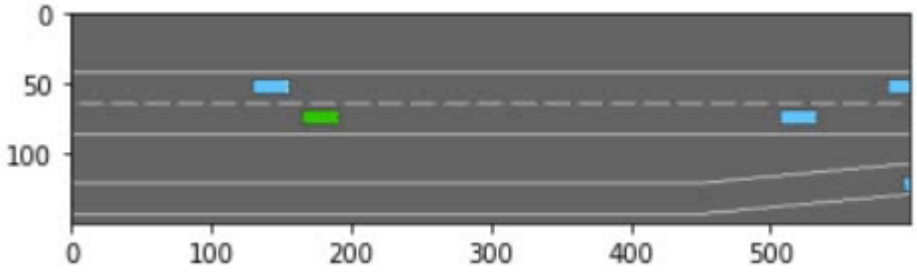
- epsilon = 0.01



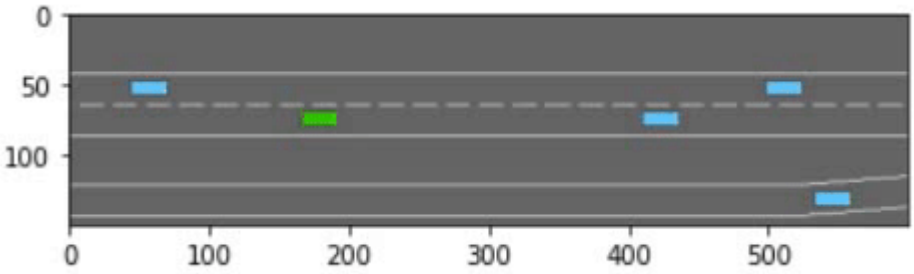
- epsilon = 0.1



- epsilon = 0.2



- epsilon = 0.5



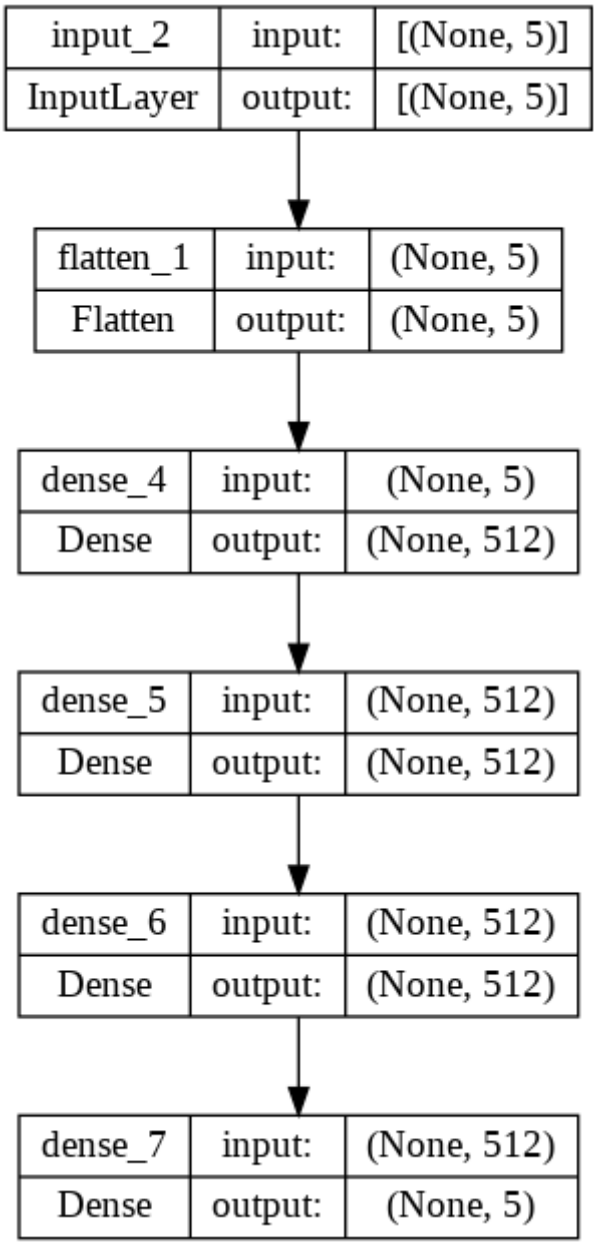
پارامتر ها

کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت – **Part 2** **Hyperparameters** قرار دارد.

مقادیر پارامتر ها به شرح زیر است:

Hyperparameters	Value
Optimizer	Adam
Optimizer learning rate	1e-4
Discount factor	0.99
Epsilon decay rate	0.97
Epsilon start	1
Epsilon minimum	0.01
Number of episodes	250
Number of replies	10

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_2 (InputLayer)         [(None, 5)]               0

 flatten_1 (Flatten)          (None, 5)                 0

 dense_4 (Dense)              (None, 512)               3072

 dense_5 (Dense)              (None, 512)               262656

 dense_6 (Dense)              (None, 512)               262656

 dense_7 (Dense)              (None, 5)                 2565

=================================================================
Total params: 530,949
Trainable params: 530,949
Non-trainable params: 0
_________________________________________________________________
```


input_2 | input: | [(None, 5)]
InputLayer | output: | [(None, 5)]
flatten_1 | input: | (None, 5)
Flatten | output: | (None, 5)
dense_4 | input: | (None, 5)
Dense | output: | (None, 512)
dense_5 | input: | (None, 512)
Dense | output: | (None, 512)
dense_6 | input: | (None, 512)
Dense | output: | (None, 512)
dense_7 | input: | (None, 512)
Dense | output: | (None, 5)

بخش سوم

هدف سوال

در این بخش قرار که تاثیر انتقال تجربه یاد گرفته شده در یک محیط به محیط دیگر را بررسی کنیم. برای این منظور ابتدا عامل را در محیط merge-v0 آموزش می دهیم. سپس در محیط highway-v0 در دو حالت عامل با وزن دهی رندم و عامل train شده، به بررسی سرعت یادگیری در محیط جدید می پردازیم.

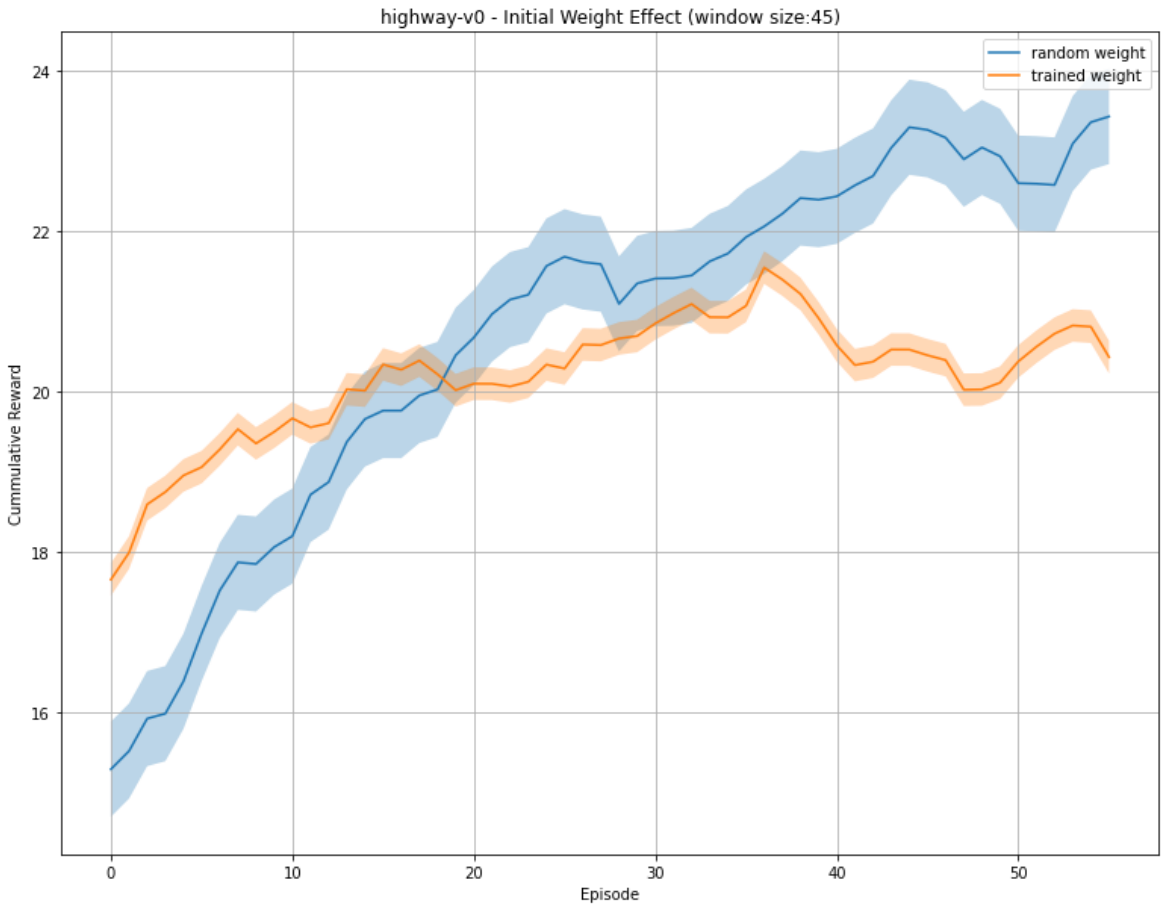
توضیح پیاده سازی

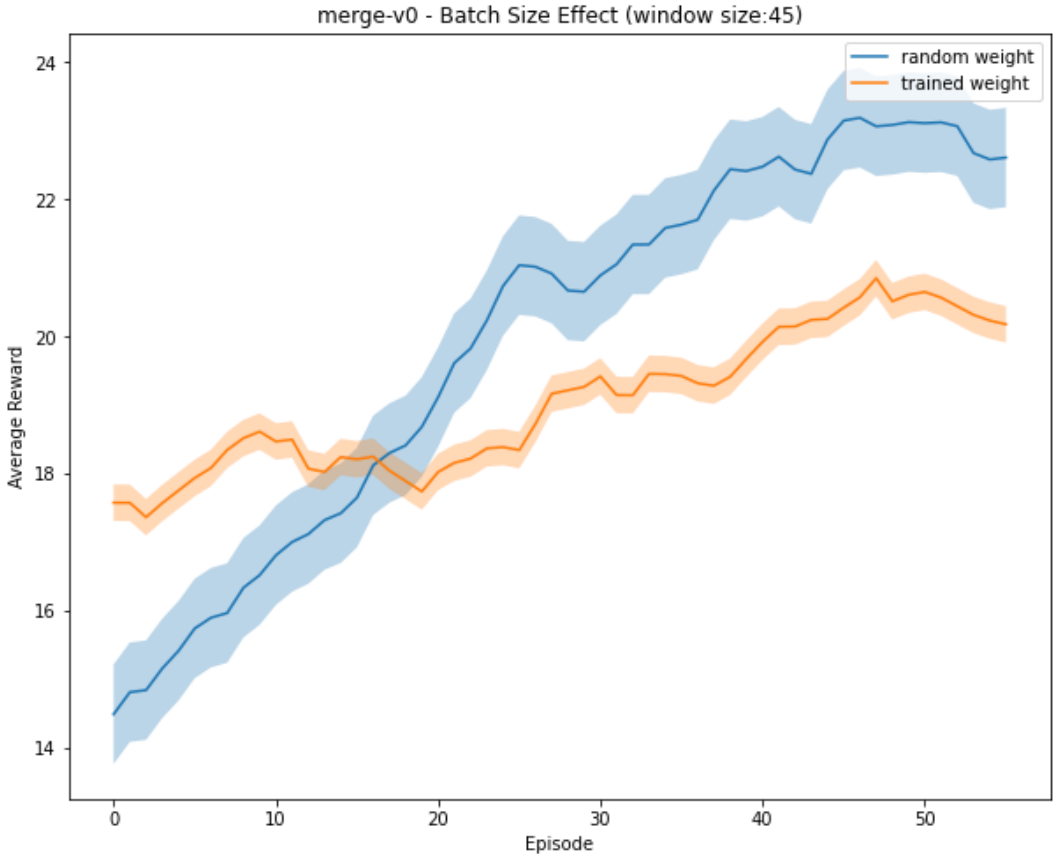
کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت **Part 3** قرار دارد.

برای اجرا کد های این بخش سوال است تا کد های بخش ابتدایی و کد های این بخش را اجرا کنید.

نتایج

نتایج حاصل از اجرا این بخش به شکل زیر است:





مشاهده می شود که سرعت یادگیری افزایش می یابد و در حالتی که از وزن های تسک قبلی استفاده کردیم، نسبت به حالت که عامل را با وزن دهی رندم اجرا کرده ایم، پاداش در اپیزود های اولیه بیشتر است. علت دریافت پاداش بیشتر در اپیزود های اولیه این است که عامل قبلا یک یادگیری ای انجام داده است. در نتیجه در تسک جدید دیگر از حالت رندم یادگیری را آغاز نمی کند و استفاده از تجربه کسب شده در تسکی دیگر، منجر به افزایش پاداش در اپیزود های اولیه در تسک جدید می شود. همچنین سرعت یادگیری نیز افزایش می یابد زیرا به دلیل استفاده از تجربه یادگیری شده، نرخ همگرایی کاهش می یابد در نتیجه عامل در تعداد گام های کمتری در تسک جدید آموزش می بیند. استفاده از وزن دهی تسک قبلی آموزش داده شده در وزن دهی شبکه برای تسک جدید، باعث می شود تا دیگر از یک حالت رندم یادگیری در تسک جدید را آغاز نکنیم و با استفاده از تجربه یادگیری شده در تسک قبل، یادگیری در تسک جدید را با سرعت بیشتری انجام دهیم و روند یادگیری هدفمند تر و مبتنی بر تجربه کسب شده انجام می شود. اما در نهایت، پاداش عاملی که از وزن دهی رندم استفاده کرده است بهتر از پاداش عاملی است که از وزن های یک تسک دیگر استفاده کرده است. علت آن است که در حالت وزن دهی رندم، عامل چون تجربه ای ندارد، بدون تبعیض بین وضعیت های مختلف یادگیری را انجام می دهد و امکان بیشتری در انتخاب وزن دارد. در نتیجه در پایان یادگیری پاداش بیشتری دریافت می کند. ولی در حالت استفاده

از وزن های تسک دیگر، عامل در ابتدای یادگیری پاداش بیشتری نسبت به حالت وزن دهی رندم دریافت می کند اما چون تلاش می کند تا همانند تسک قبلی در محیط تسک جدید یادگیری را انجام بدهد، به دلیل تفاوت های محیط دو تسک ممکن است برای عملیلت به روز رسانی وزن ها در مینیمم محلی گیر کند. در نتیجه حسرت بیشتری در اپیزود های پایانی خواهد داشت.

استفاده از وزن های آموزش‌دیده برای عامل در RL عمیق، سرعت یادگیری را افزایش می‌دهد، زیرا شبکه قبلاً با اطلاعات آزمایش قبلی seed شده است. این به عامل اجازه می دهد تا به جای شروع هر بار از ابتدا، بر آنچه قبلاً آموخته است، بسازد. علاوه بر این، عاملی که از وزن‌های train شده استفاده می‌کنند، با داشتن تنوع بیشتر شرایط اولیه در مقایسه با عواملی که training را از یک مجموعه وزن‌های اولیه به‌طور تصادفی شروع می‌کنند، شانس بیشتری برای اجتناب از local optima دارند. در نهایت، این بدان معنی است که exploration کمتری مورد نیاز است و عامل می تواند سریعتر به سیاست های بهینه خود برسد.

استفاده از وزن های از پیش آموزش دیده در یادگیری تقویتی عمیق به عامل اجازه می دهد تا از نقطه شروع بهتری تعامل با محیط را آغاز کند. این «warm start» یادگیری را برای عامل آسان‌تر می‌کند، زیرا از قبل نمایش خوبی از ویژگی‌های خاصی مانند وظایف تشخیص شی یا پیش‌بینی حالت‌های مفید در یک محیط دارد. این امر میزان زمان آموزش و منابع محاسباتی مورد نیاز برای یادگیری کار را کاهش می دهد و در نتیجه سرعت یادگیری در RL عمیق را افزایش می دهد.

بخش چهارم - امتیازی

هدف سوال

در این بخش به پیاده سازی الگوریتم DQN با حالت Prioritized Experience Replay می پردازیم و مسئله بخش دوم را مجدد با این الگوریتم بررسی و حل می کنیم. برای پیاده سازی الگوریتم مطابق توضحیات و شبه کد آن در مقاله داده شده عمل می کنیم: [12]

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

افزودن Prioritized Experience Replay به الگوریتم DQN می‌تواند به عامل کمک کند تا سریع‌تر و کارآمدتر یاد بگیرد و به او اجازه می‌دهد به طور انتخابی بر روی تجربیات دشواری تمرکز کند که به احتمال زیاد عملکردش را بهبود می‌بخشد. این را می توان با تعیین اولویت نمونه های با تجربه بر اساس میزان سودمندی پاداش بلندمدت مورد انتظار آنها برای عامل انجام داد. از آنجایی که عامل ها را قادر می‌سازد به جای تمرکز بر پاداش‌های کوتاه‌مدت، از پاداش‌های بلندمدت بیاموزند، که می‌تواند منجر به سیاست‌های sub-optimal شود، این منجر به یادگیری بهتر و قابلیت‌های تعمیم بهتر برای الگوریتم‌های DQN می‌شود.

توضیح پیاده سازی

کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت **Part 4 – DQN with Prioritized Experience Replay** قرار دارد.

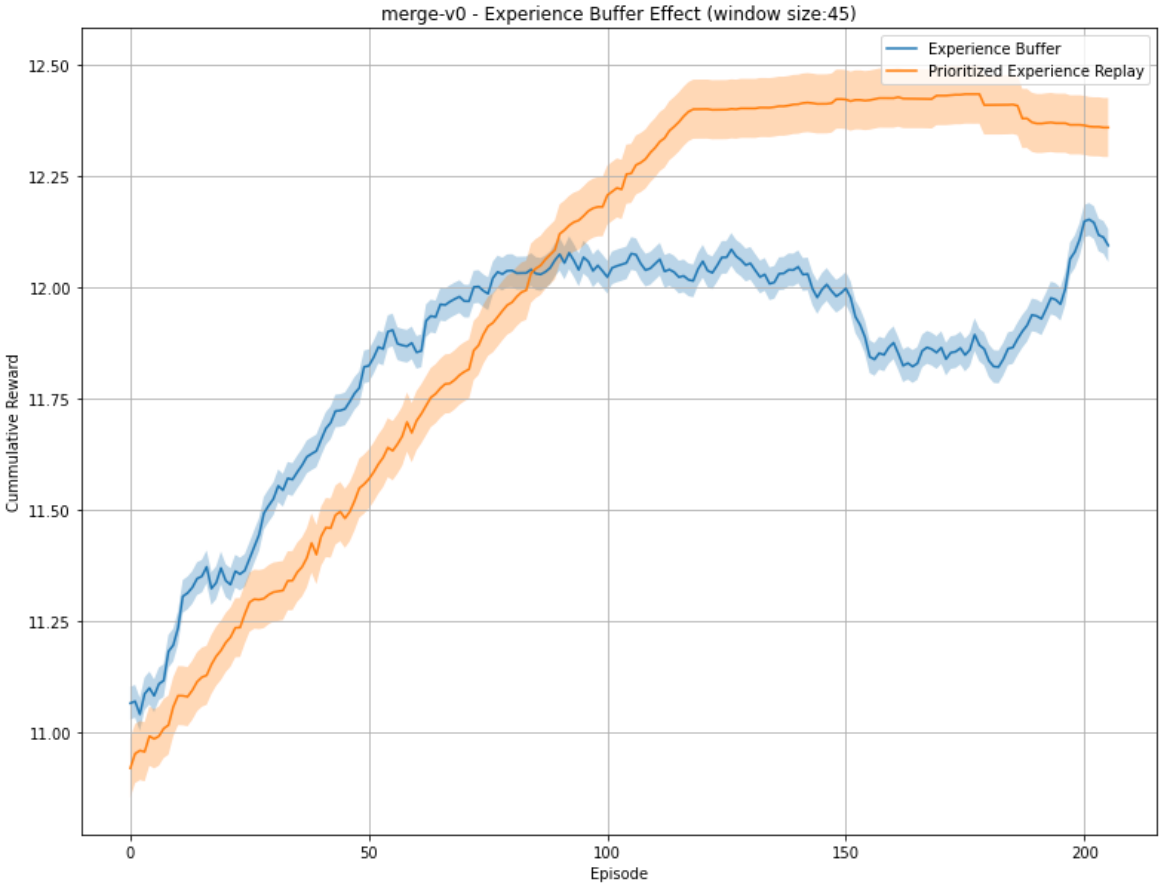
- در کلاس **SumTree** درخت مربوطه پیاده سازی شده است. SumTree یک درخت دودویی است که در آن ارزش یک گره برابر است با مجموع گره های موجود در زیر درخت سمت چپ و زیر درخت سمت راست. یک درخت خالی SumTree است و مجموع یک درخت خالی را می توان ۰ در نظر گرفت. یک گره برگ نیز به عنوان SumTree در نظر گرفته می شود.
- در کلاس **PrioritizedExperienceReplay** بافر تجارب اولویت دار پیاده سازی شده است.
- در کلاس **DQN_PER** الگوریتم DQN با حالت Prioritized Experience Replay پیاده سازی شده است.
- تابع **run_ DQN_PER _agent** الگوریتم DQN با حالت Prioritized Experience Replay را با ۱۰ بار تکرار و به تعداد ۲۵۰ ایپزود اجرا می کند و پاداش های دریافت شده را بر می گرداند.

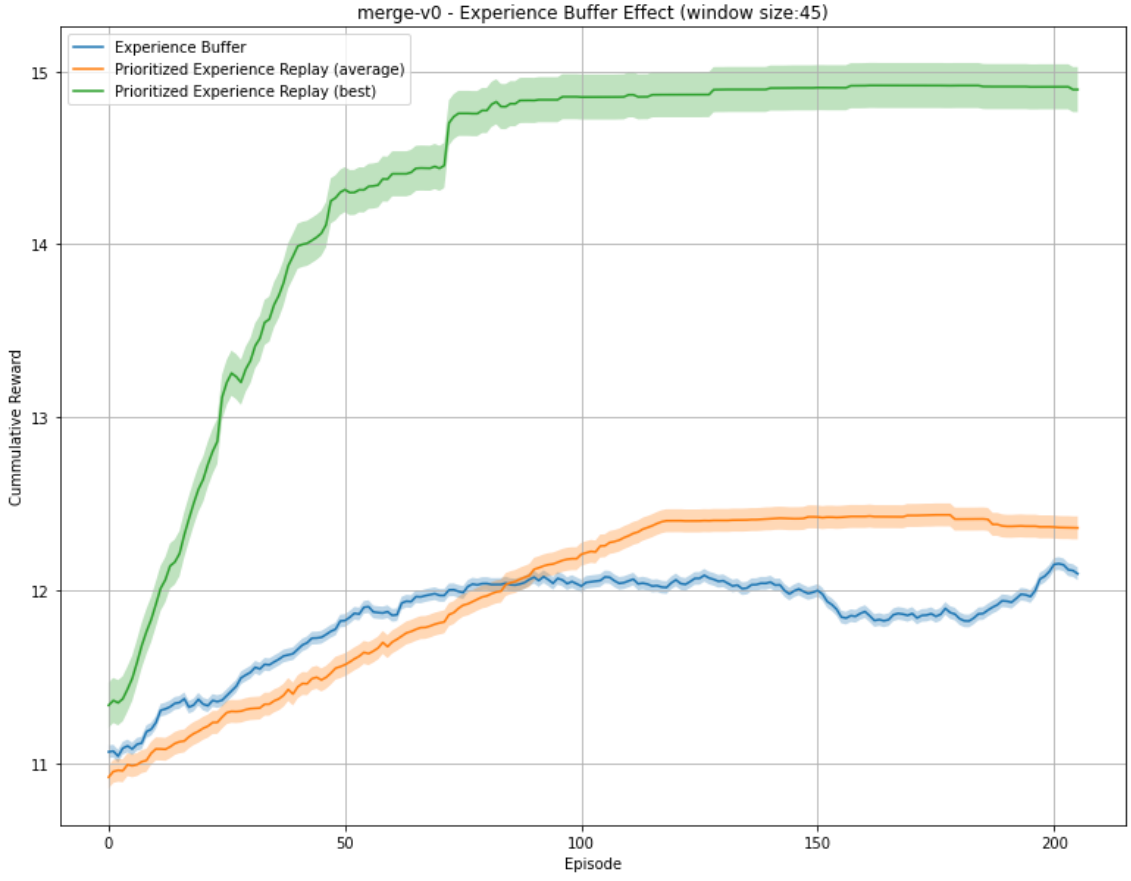
برای اجرا کد های این بخش سوال است تا کد های بخش ابتدایی و کد های این بخش را اجرا کنید.

نتایج

کد های مربوط به پیاده سازی این قسمت در فایل ***IL_HW5.ipynb/html*** قسمت **Part 4 – Result** قرار دارد.

نمودار پاداش کسب شده توسط عامل در طول یادگیری برای ۱۰ بار اجرا الگوریتم DQN یکبار با بافر تجارب عادی (Deep Q-Learning) و بار دیگر با حالت Prioritized Experience Replay به ازای ۲۵۰ ایپزود به همراه بازه اطمینان ۹۵ درصد به شکل زیر است:


merge-v0 - Experience Buffer Effect (window size:45)
Experience Buffer
Prioritized Experience Replay
Cummulative Reward
Episode
12.50
12.25
12.00
11.75
11.50
11.25
11.00
0
50
100
150
200


merge-v0 - Experience Buffer Effect (window size:45)
Experience Buffer
Prioritized Experience Replay (average)
Prioritized Experience Replay (best)
Cummulative Reward
Episode
15
14
13
12
11
0
50
100
150
200

Prioritized Experience Replay به الگوریتم DQN کمک می‌کند تا تجربیات مهم‌تری را در حافظه تکراری(replay memory) خود اولویت‌بندی کند تا بتواند روی مناطقی که نیاز به بهبود دارد تمرکز بهتری داشته باشد. این می تواند به زمان همگرایی کمک کند و همچنین به طور بالقوه منجر به عملکرد کلی یادگیری بهتر در مقایسه با الگوریتم استاندارد DQN بدون مولفه های تکراری تجربه اولویت بندی شده(prioritized experience replay) شود.

Prioritized Experience Replay ، کارایی داده را بهبود می‌بخشد، با پخش بیشتر انتقال‌هایی که از آنها چیزهای بیشتری برای یادگیری وجود دارد. DQN به طور یکنواخت از replay buffer نمونه برداری می کند. در حالت ایده‌آل، ما می‌خواهیم بیشتر از آن انتقال‌هایی که چیزهای زیادی برای یادگیری وجود دارد نمونه‌برداری کنیم. به عنوان یک پروکسی برای پتانسیل یادگیری، انتقال نمونه های تکراری تجربه اولویت بندی شده با احتمال p_t نسبت به آخرین *TD error* مطلق مواجه شده است: [13]

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t) \right|^\omega$$

منابع

1- https://medium.com/intro-to-artificial-intelligence/reinforce-a-policy-gradient-based-reinforcement-learning-algorithm-84bde440c816
2- http://www.scholarpedia.org/article/Policy_gradient_methods
3- https://lilianweng.github.io/posts/2018-04-08-policy-gradient/#policy-gradient
4- https://towardsdatascience.com/policy-gradient-methods-104c783251e0
5- https://viso.ai/deep-learning/deep-reinforcement-learning/#:~:text=The%20use%20of%20deep%20learning,levels%20of%20abstractions%20from%20data.
6- https://www.quora.com/What-is-the-downside-of-deep-reinforcement-learning-When-shouldnt-it-be-used
7- https://www.unite.ai/what-is-deep-reinforcement-learning/
8- https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Deep-Learning.html#:~:text=Drawbacks%20or%20disadvantages%20of%20Deep%20Learning&text=%E2%9E%A8It%20requires%20very%20large,increases%20cost%20to%20the%20users.
9- https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c
10- https://towardsdatascience.com/a-technical-introduction-to-experience-replay-for-off-policy-deep-reinforcement-learning-9812bc920a96
11- https://arxiv.org/pdf/1312.5602.pdf
12- https://arxiv.org/pdf/1511.05952.pdf
13- https://arxiv.org/pdf/1710.02298.pdf