دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره 5
گروه ...
درس یادگیری تعاملی
پاییز 1400

نام و نام خانوادگی	مهسا تاجیک
شماره دانشجویی	810198126

فهرست

چکیده

در این تمرین به حل مساله mountain car در فضای پیوسته با اعمال گسسته می پردازیم و این مساله را با روش tile coding و الگوریتم SARSA پیاده سازی میکنیم. سپس یک یادگیر برمبنای الگوریتم deep Q-learning در محیط lunerLander توسعه میدهیم و پاداش دریافتی در طول یادگیری را بررسی میکنیم.

سوال 1 - سوال پیاده‌سازی

هدف سوال

هدف از سوال بررسی مساله mountain car با استفاده از الگوریتم SARSA است که فضای حالت پیوسته و شامل 2 متغیر مکان فعلی ماشین و سرعت فعلی است و فضای عمل گسسته شامل حرکت به عقب، ماندن در موقعیت فعلی و حرکت به جلو است.

توضیح پیاده سازی

در ابتدا استیت ها و اکشن ها را دقیق مشخص میکنیم. استیت ها پیوسته شامل 2 متغیر مکان فعلی ماشین و سرعت فعلی است.

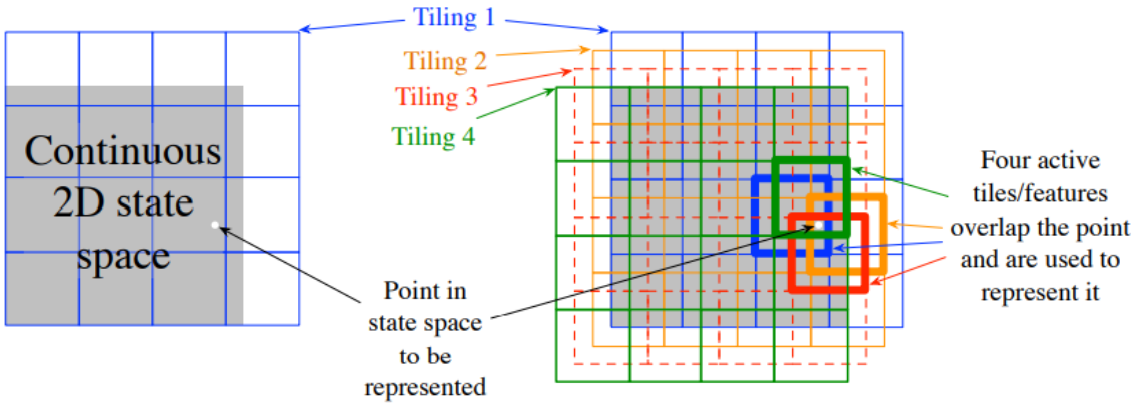
فضای اکشن ها گسسته و شامل 3 اکشن است که بصورت [-1, 0, 1] است که -1 معادل اکشن حرکت رو به عقب، 0 معادل اکشن ماندن در موقعیت فعلی و 1 معادل اکشن حرکت رو به جلو است.

متغیر مکان فعلی در بازه ی (-1.2, 0.5) و سرعت فعلی در بازه ی (-0.07, 0.07) قرار دارد. از آنجایی که فضای حالت پیوسته است نیاز یک function approximator داریم:

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s)$$

برای این که این تابع نسبت به پارامتر w خطی باشد، غیر خطی بودن را در تابع x(s) به روش tile coding کد میکنیم. در tile coding ، receptive field های ویژگی ها در پارتیشن های فضای حالت گروه بندی می شوند .هر یک از این پارتیشن ها را tiling و هر عنصر پارتیشن را tile می نامند .به عنوان مثال، ساده ترین tile coding یک فضای حالت دو بعدی، یک شبکه یکنواخت مانند آنچه در سمت چپ شکل زیر نشان داده شده است .کاشی ها یا receptive field در اینجا مربع هستند .

یک مورد ساده با چهار کاشی در سمت راست شکل نشان داده شده است . هر حالتی، مانند حالتی که با لکه سفید نشان داده می شود، در هر چهار کاشی دقیقاً در یک کاشی قرار می گیرد .این چهار کاشی با چهار ویژگی مطابقت دارند که با رخ دادن حالت، فعال می شوند.



برای پیاده سازی tile coding ابتدا باید تعداد tiling ها و ابعاد هر tiling و همچنین بیشترین سایز hash_table را مشخص کنیم که سه حالت مختلف خواسته شده است:

۱. num_tiles: 16, num_tilings: 2, iht_size: 4096

۲. num_tiles: 4, num_tilings: 32, iht_size: 4096

۳. num_tiles: 8, num_tilings: 8, iht_size: 4096

برای اینکه در هر لحظه کاشی های فعال را پیدا کنیم، تابع get_tiles را فراخوانی میکنیم که با توجه به استیتی که در آن هستیم، کاشی های فعال را برمیگرداند. انتخاب اکشن در الگوریتم sarsa با سیاست اپسیلون گریدی انجام شده است و در تابع choose_action در کلاس SarsaAgent پیاده سازی شده است.

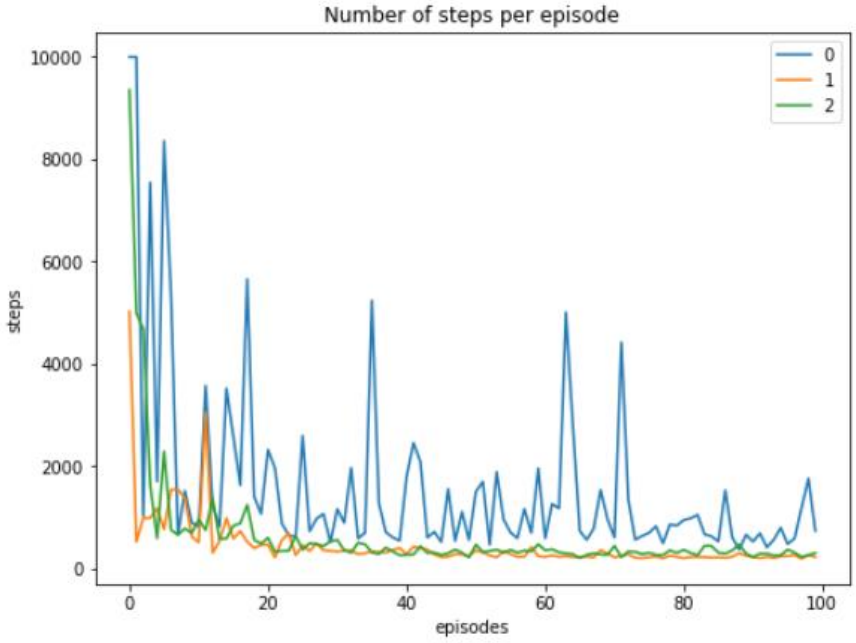
جمع وزن کاشی ها در هر استیت، value آن استیت را میسازد که اگر به موقعیت مکانی 0.5 که ترمینال استیت محسوب میشود، برسیم، value برابر صفر میشود. وزن کاشی ها در زمان اجرای الگوریتم sarsa با توجه به کاشی های فعال، اپدیت میشود که دو تابع اپدیت مختلف برای زمانیکه در ترمینال استیت هستیم و زمانیکه در ترمینال استیت نیستیم پیاده سازی شده است.

نتایج

با توجه به نمودار زیر بهترین حالت کدینگ از بین سه حالت، به ازای 100 اپیزود حالت سوم است یعنی زمانیکه :

Iht_size = 4096, num_tilings = 8, num_tiles = 8

البته حالت دوم هم بسیار نزدیک به حالت دوم است یعنی زیاد بودن تعداد تایلینگ ها به نسبت زیاد بودن تعداد تایل ها تاثیر کمتری روی تعداد استپ های هر اپیزود و نرخ کاهش آن و سرعت همگرایی الگوریتم دارد دلیلش هم آن است که با زیاد شدن تعداد تایلینگ ها تعداد المان هایی از w که اپدیت میشوند بیشتر میشود و یک استیت در تعداد تایلینگ های بیشتری کد میشود.



روند اجرای کد پیاده‌سازی

کدهای این سوال در نوتبوک Question1 قابل اجرا است.

سوال 2 - سوال پیاده‌سازی

هدف سوال

هدف از سوال بررسی مساله کنترل فرود فضاپیما در محیط LunarLander از پکیج gym می باشد. در این مساله فضای حالت پیوسته و شامل 6 متغیر حالت است و اکشن ها گسسته و شامل حرکت به چپ، راست، بالا و ماندن در وضعیت فعلی است. می خواهیم یک یادگیر بر مبنای الگوریتم DQN به منظور بیشینه کردن میانگین پاداش دریافتی توسعه دهیم.

توضیح پیاده سازی

همانطور که در صورت سوال خواسته شده یک کلاس ReplayExperienceBuffer تعریف میکنیم که در تابع add_experience تجربمون از حضور در یک استیت و انجام یک اکشن و پاداش دریافتی و رفتن به استیت بعدی را در آن ذخیره کنیم. در تابع sample به اندازه ی batch_size از این تجربه ها را سمپل برمیداریم.

یک کلاس DeepNetwork تعریف می کنیم که در constructor کلاس معماری شبکه را مشخص میکنیم:

```
def __init__(self, n_state, n_action, seed)

    super(DeepNetwork, self).__init__()
    self.seed = torch.manual_seed(seed)
    self.l1 = nn.Linear(n_state, 32)
    self.l2 = nn.Linear(32, 32)
    self.l3 = nn.Linear(32, n_action)
```

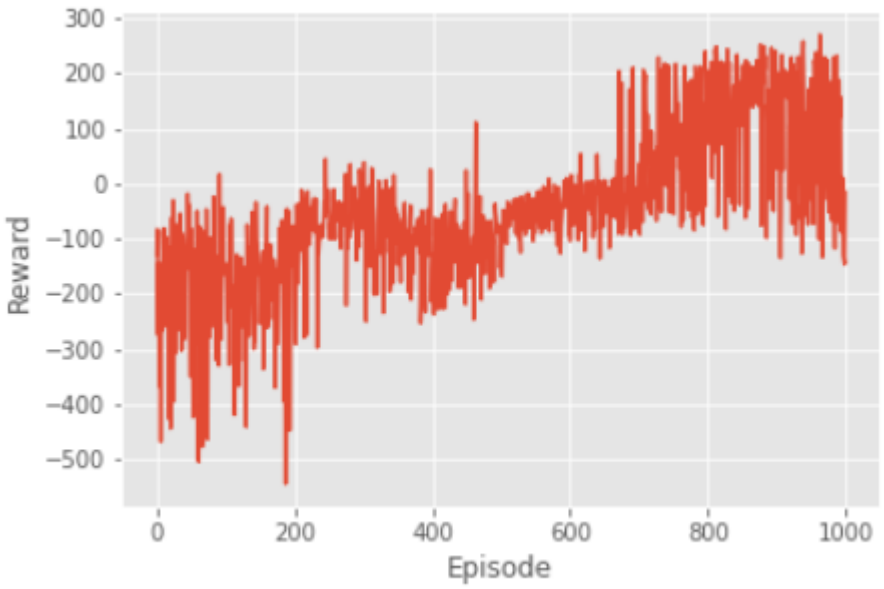
و یک متد forward که ورودی آن استیت بوده و خروجی شبکه را که Q(s,a) است، برمیگرداند:

```
def forward(self, state):
    x = self.l1(state)
    x = functional.relu(x)
    x = self.l2(x)
    x = functional.relu(x)
    return self.l3(x)
```

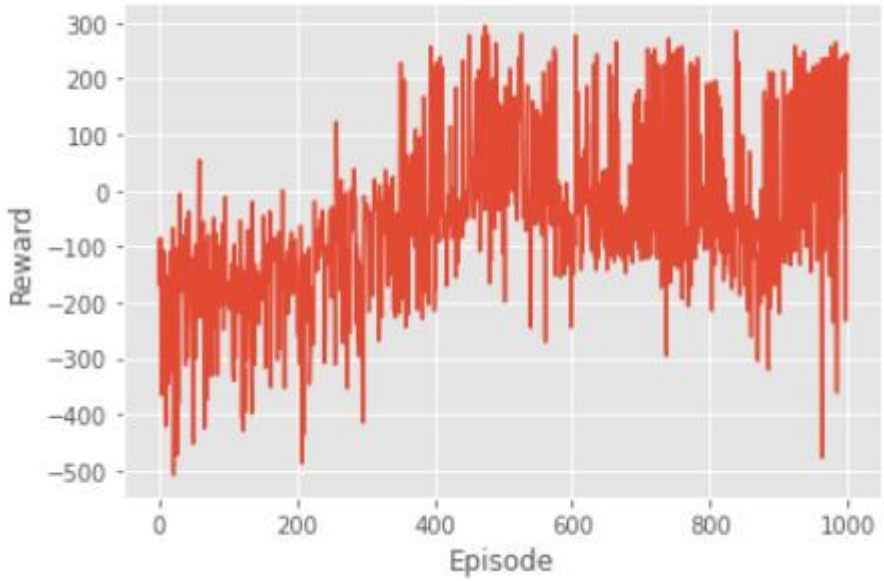
یک کلاس Agent تعریف کردیم که در متد choose_action یک اکشن را براساس سیاست اپسیلون گریدی انتخاب میکند و نسبت به Qvalue ها گریدی عمل میکند. در متد step ایجنت تجربه اش از انجام یک عمل در استیت فعلی و دریافت پاداش و رفتن به استیت بعدی را در بافر ذخیره می کند و اگر تعداد تجربه های داخل بافر به حد کافی رسیده باشد یعنی از اندازه ی داده ی batch که 32 درنظر گرفتیم، بزرگتر شده باشد، یک زیرمجموعه از تجربه ها را بصورت رندوم بعنوان سمپل برداشته و روی آن ها یادگیری انجام میدهد. در متد learn روی تجربه هایی که سمپل گرفتیم عمل یادگیری را انجام میدهد و مقدار loss را با استفاده از روش MSE محاسبه و کمینه میکند.

نتایج

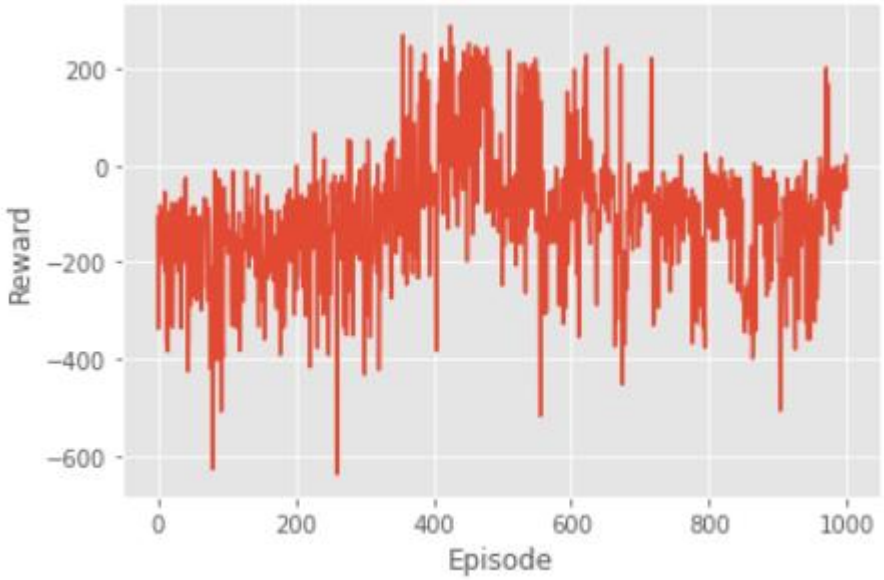
نمودار پاداش در طول یادگیری به ازای اندازه بافر e8 و 1000 اپیزود و batchsize = 32:



نمودار پاداش در طول یادگیری به ازای اندازه بافر e4 و 2000 اپیزود و batchsize = 64:



نمودار پاداش در طول یادگیری به ازای اندازه بافر 2e و 2000 اپیزود و batchsize = 64:



هرچقدر اندازه ی بافر بزرگتر است پاداش ها در طول اپیزود ها بیشتر افزایش می یابند. تجربه های بیشتری را ذخیره در بافر میتوانیم داشته باشیم که موقع سمپل برداشتن به ما کمک میکند.

روند اجرای کد پیاده‌سازی

کدهای این سوال در نوتبوک Question2 قابل اجرا است.