دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره پنجم

درس یادگیری تعاملی

پاییز 1400

نام و نام خانوادگی	علی ساعی زاده
شماره دانشجویی	810196477

فهرست

چکیده

در این پروژه قصد داریم موارد کاربرد یادگیری تقویتی در مسائل با فضای پیوسته بررسی کنیم. به صورت کلی و ساده دو رویکرد در مواجهه با مسائل پیوسته داریم. اول اینکه فضای پیوسته خود به روش های که در درس بررسی شد گسسته سازی کنیم و روش دوم آن که با استفاده از توابع تخمین با مسئله بصورت پیوسته برخورد کنیم. در این پروژه سعی می‌شود هر دو نگاه در قالب دو سوال بررسی شود.

سوال 1 – گسسته سازی به روش Tile Coding

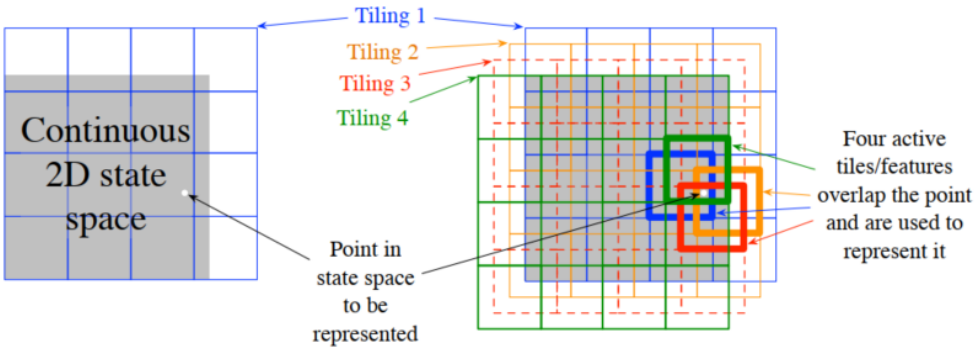
هدف سوال

در این مسئله سعی داریم با ماشینی که نیروی کافی برای بالا رفتن از صخره را ندارد به کمک اینرسی آن، به آن آموزش دهیم تا با حرکت گهواره ای اینرسی لازم را برای بالا رفت از صخره بدست آورد. حالت این مسپله شامل سرعت و موقعیت عامل است و اعمال آن شمانل سه عمل جلو و عقب و ایستادن است.

شکل 1 محیط ماشین صخره نورد

در این مسئله با فضای حالت پیوسته روبرو هستیم. طبق خواسته های سوال قصد داریم از روش Tile Coding که به طور مفصل در کلاس بحث شده است استفاده کنیم این روش صفحه حالت مارا قسمت بندی کرده و سپس با شیفت دادن این صفحه کاشی های بیشتری تولید می شود که این کاشی ها همپوشانی دارند و به Generalization ما کمک می‌کنند.

همچنین در این سوال حالت های مختلف Tile Coding پیاده شده تا بهترین حالت Coding برای محیط ارائه شده بدست آید.



شکل 2 شمایی از روش **Tile Coding**

توضیح پیاده سازی

ابتدا کد مربوط به الگوریتم Tile Coding از لینک داده شده برداشته شد و در فایل tiles3.py ذخیره شد. با استفاده از این الگوریتم می‌توان محیط را گسسته سازی کرد (state coding) و سپس با دادن ورودی حالت (پیسوته) به آن اندیس کاشی های فعال را دریافت کرد (گسسته).

سپس برای تعریف محیط و گسسته سازی آن از کلاس MountainCarTileCoder استفاده شده است که در فایل tile_coder.py موجود است. در این قست به کمک لینک داده شده محیط گسسته سازی می‌شود. برای اینکار مرز های پیوسته سرعت و موقعیت ماشین را نیاز داریم که از محیط gym گرفته شده است. همچنین تابع get_tiles با دریافت ورودی سرعت و موقعیت، کاشی های فعال را خروجی می دهد.

عامل محیط از روش SARSA برای حل محیط استفاده می‌کند ایده لی این الگوریتم در شکل 3 آمده است.[1]

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

شکل 3 الگوریتم **SARSA** برای محیط پیوسته

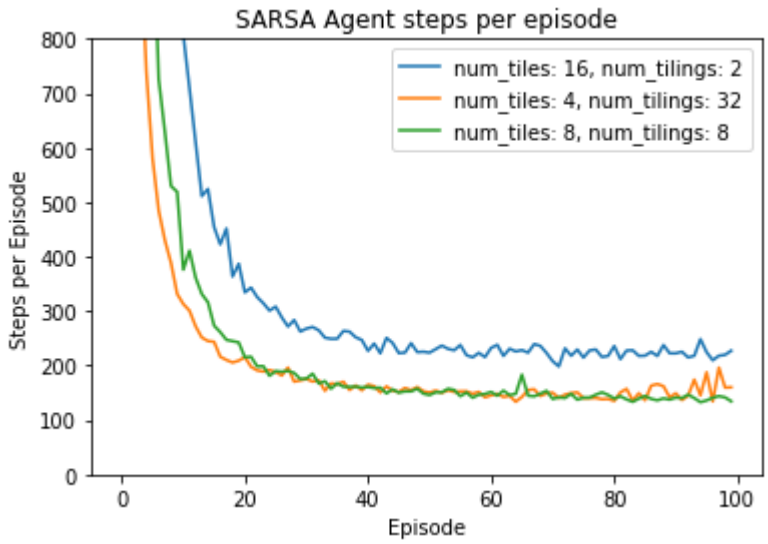
سیاست رفتاری عامل ما e-greedy خواهد بود. همچنین اپسیلون و نرخ یادگیری ما کاهشی خواهد بود. وزن ما در این پیاده سازی برداری 4096×3 خواهد بود که شامل عمل و کاشی های وجود (استیت) است. با توجه به اینکه تابع تخمین q-value ما تابعی خطی و با مقدار 1 است (همان کاشی های فعال) بنابراین نیازی به محاسبه گرادیان نیست.

پیاده سازی این عامل در فایل sarsa_agent.py موجود است که پیاده سازی الگوریتم موجود در شکل 3 است. قسمت های مختلف این الگوریتم بصورت comment در کد مشخص شده است.

نتایج

در 30 بار اجرای برنامه با افق 100 تایی (episode 100) و متوسط‌گیری برای Tile Coding با مقادیر خواسته شده شکل 4 حاصل شده است.

توجه شود که ضریب یادگیری یکسان باعث می‌شد تا بعضی عامل‌ها همگرا نشوند. بنابراین برای هر عامل سعی شد با آزمون خطا، ضریب یادگیری بهینه پیدا شود.



شکل 4 تعداد توسط گام هر عامل برای رسیدن به قله

num_tiles: 16, num_tilings: 2

این روش به تعداد گام بهینه یعنی حدود 160 همگرا نشده است و همچنین سرعت همگرایی پایینی دارد. علت آن هم این است که در این حالت تعداد کاشی ها برای حل مسئله کافی نیست و به اصطلاح به دچار under fit می‌شود و به نقطه بهینه نمی‌رسد. بنابراین استفاده از این ارقام برای مسئله ما مناسب نیست.

num_tiles: 4, num_tilings: 32

این روش بیشترین تعداد کاشی را در اختیار ما می‌گذارد و انتظار داریم که بهترین عملکرد را داشته باشد. اما در عمل بنظر می‌آید کمی این ارقام باعث overfitting می‌شوند. و بسیار تنظیم پارامتر های آزاد آن پیچیده است (بدلیل حساسیت زیاد همگرایی آن) اما پس از تلاش بسیار موفق شدیم این روش را به حدود 160 گام همگرا کنیم. با توجه به اینکه سرعت همگرایی این روش از همه روش های دیگر بهتر است و همچنین به مقدار بهینه همگرا می‌شود بنظر گزینه مناسبی برای بهترین کاندید می‌آید. اما پیچیدگی های تعیین پارامتر های آزاد و حساسیت بیش از حد آن به نرخ یادگیری کمی مارا به در مورد بهینه بودن این روش به شک می اندازد.

num_tiles: 8, num_tilings: 8

این روش به نرخ بهینه همگرا می‌شود اما سرعت همگرایی آن پایین تر از روش قبل است بنابراین استفاده از روش قبل ارجحیت دارد. اما مزیت این روش این است که حساسیت کمی به پارامتر های آزاد دارد و معولا همگرا می‌شود.

نتیجه گیری

همان‌طور که انتظار داشتیم با تعیین مناسب نرخ یادگیری بهترین روش روشی است که بیشتری تعداد کاشی را ایجاد کند (فضای گسسته بزرگتر) بنابراین روش دوم که سرعت همگرایی بییشتری دارد و به مقدار بهینه همگرا می‌شود بهترین حالت Coding است.

توضیحات پیاده سازی و اجرا

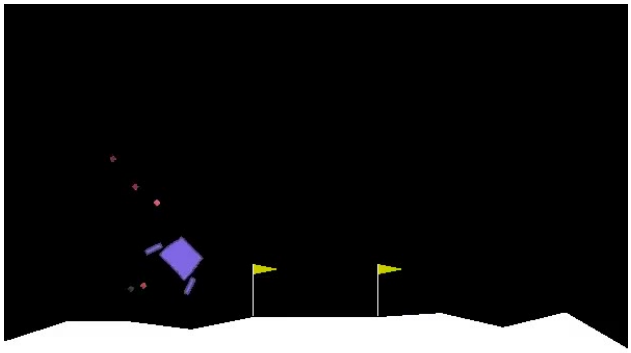
در محیط gym استفاده شده دو حالت برای رسیده به انتهای episode در نظر گرفته شده است. اول اینکه عامل به بالای صخره برسد دوم اینکه تعداد گام های عامل بیشتر از 200 شود. با توجه به اینکه بهترین تعداد گام ما در حدود 170 است یادگیری با این شرایط عملا قابل اجرا نبود نبود. بنابراین تغییراتی در محیط اجرا شد و تعداد محدودیت گام های آن به 10000 تغییر دادیم.

سوال 2 - کنترل فرود فضاپیما

هدف سوال

در این سوال سعی داریم توابع تخیمن خود را برای محیط پیوسته به کمک شبکه عصبی پیاده سازی کنیم. مسئله شامل یک فضاپیما با چهار عمل حرکت به سمت راست، چپ، بالا و بی‌حرکت است. سعی داریم این فضاپیما را در یک محدوده مشخص فرود آوریم.

پاداش ها و فضای حالت این مسوله در محیطgym تعریف شده و از آن استفاده می‌کنیم.



شکل 5 محیط مسئله کنترل فرود فضا پیما

توضیحات پیاده سازی

در این پیاده سازی سعی شده طبق فایل هایی که داده شده عمل شود. شبکه عصبی به کمک کتابخانه torch پیاده سازی شده است که شامل یک لایه است همچنین تعداد نورون ها 64 تا انتخاب شده است. ورودی این شبکه استیت خواهد بود و انتظار داریم به ازای 4 اکشن موجود خروجی ارزش آن هارا در استیت داده شده به ما بدهد. این شبکه در فایل network.py پیاده سازی شده است.

همچنین یک بافر برای ذخیره سازی و نمونه گیری از تجربیات عامل در فایل experience_replay.py پیاده شده است که دارای سه تابع اصلی اضافه کردن تجربه، نمونه گیری و مقدار طول بافر است.

Algorithmus 1 : Deep Q-Learning mit Experience Replay Memory

Result : a nearly optimal policy π
Initialize replay memory $\mathcal{D}$ that has a certain length ;
Initialize parameter $\mathbf{w}$;
Initialize parameter $\mathbf{v}$ that calculate TD-Target by $\mathbf{v} \leftarrow \mathbf{w}$;
for *episode*$= 1$ *to* M **do**
  Observe initial state s_0 from environment ;
  for $t= 1$ *to* T **do**
    Select a random action with probability ϵ/m otherwise
    select action $a_t = \arg\max_a \hat{q}(s, \mathbf{a}, \mathbf{w}^t)$;
    Observe reward r_t and next state s_{t+1} from environment ;
    Store (s_t, a_t, r_t, s_{t+1}) tupel in $\mathcal{D}$;
    Sample random batch from $\mathcal{D}$;
    $\mathbf{y} \leftarrow r_t + \gamma \max_{a^\circ} \hat{q}(s_{t+1}, \mathbf{a}^\circ, \mathbf{v})$;
    $\hat{\mathbf{y}} \leftarrow \hat{q}(s_t, \mathbf{a}, \mathbf{w}^t)$;
    $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \nabla \frac{1}{2}(\mathbf{y} - \hat{\mathbf{y}})^2$;
    Every c time steps, set $\mathbf{v} \leftarrow \mathbf{w}^t$;
  end
end

شکل 6 الگوریتم**DQN**

در مرحله آخر عامل DQN پیاده شده است. این عامل شامل دو شبکه عصبی عصبی برای محاسبه و بروزرسانی ارزش اعمال در هر استیت است. همچنینی مقادیر آزاد این عامل با توجه بهترین حالت بهینه شده است. سایز بافر هم برای مقایسه بصورت متغیر به عامل داده می‌شود.

تابع step مانند وظیفه ذخیر حرکت در بافر را دارد همچنین با توجه به فرکانس آپدیت، تجربه ها را استفاده می کند و شبکه را آموزش می دهد.

تابع act عمل بهینه را به کمک سیاست e-greedy و همچنین شبکه محلی آموزش دیده انتخاب می‌کند.

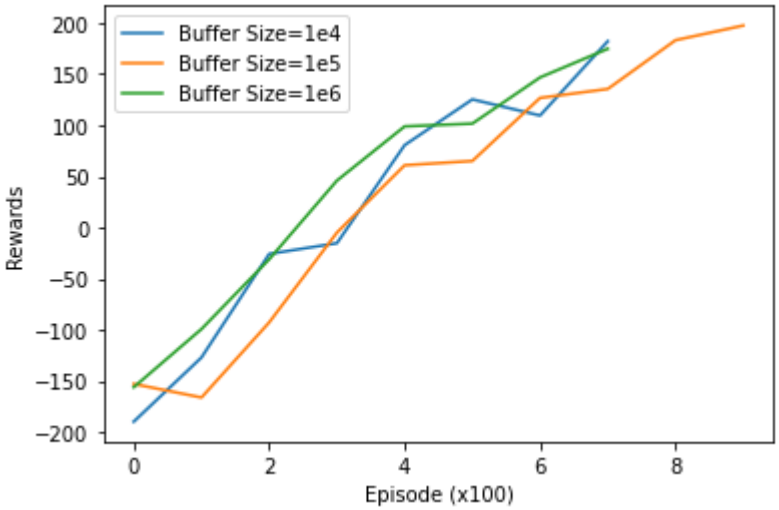
تابع learn وظیفه آموزش شبکه هارا به عهده دارد این اموزش با نمونه برداری از بافر انجام می شود. مقادیر q با توجه به عمل انجام شده محاسبه می‌شود و با مقایسه با خروجی شبکه، هر دو شبکه آموزش می‌بینند. این قسمت به کمک کتباخانه ها و توابع موجود در اینترنت پیاده سازی شده است.

برای محیط این مسئله از کتابخانه gym بدون تاخیر استفاده شده است. 2000 اپیزود اجرا شده است. و پایان کار زمانی است که عامل ما بیشتر از 200 امتیاز کسب کند سپس شبکه ذخیره می شود که در فایل optimal_network موجود است.

نتایج

در پایان برای 4 بافر با سایز های 1e4 1e5 1e6 عملکرد مقایسه شده است.انتظار داشتیم هر چه طول بافر بیشتر باشد (تجربه های قبلی بیشتری ذخیره شده باشند) یادگیری سریع تری را تجربه خواهیم کرد

اما در واقعیت اینگونه نیست. باید این را در نظر گرفت که طول بیشتر بافر به معنای اختصاص حافظه بیشتر است که در بعضی موادر ممکن است محدودیت باشد.



شکل 7 نمودار پاداش با سایز بافر مختلف

بادی طول بافر را نه خیلی زیاد درنظر گرفت نه خیلی کوتاه. کوتاهی بافر باعث می‌شود تجربه هایی که correlation بیشتری با هم دیگر دارند انتخاب شوند و یادگیری کند یا عملا غیرممکن شود از طرفی طول بافر زیاد یادگیری را در طولانی مدت به شدت کند می‌کند. [2]

فیلم کوتاه در پوشه video قابل دسترسی است.

منابع

[1] R. S. Sutton, Introduction to reinforcement learning, Cambridge: MIT press, 2018.

[2] R. a. J. Z. Liu, "The effects of memory replay in reinforcement learning," *56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2018.