

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

حل کردن مسئله یادگیری تقویتی با استفاده از الگوریتم

MDP

( به کمک رابط گرافیکی ماژول Gymnasium )

آرین جعفری (4003613014)

محمد‎حسن حیدری (4003613025)

* توضیحات کلی

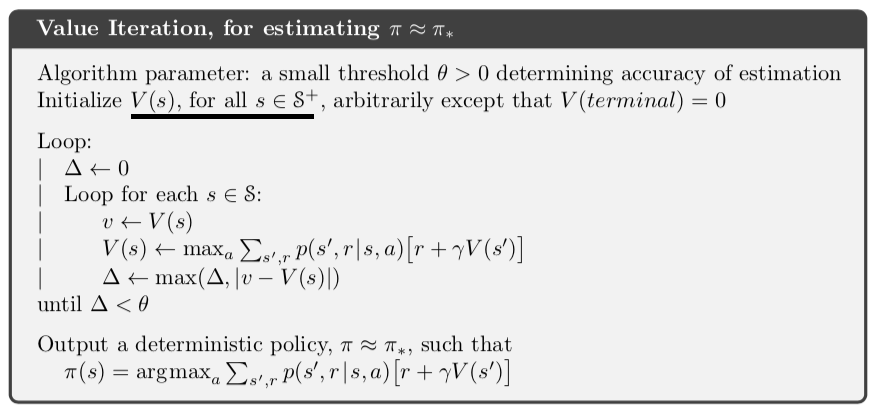
در فاز قبل ( پیدا کردن بهترین مسیر پروازی ) , محیطی که با آن کار می‌کردیم محیطی قطعی بود. اما محیطی که در این فاز پیش روی ماست محیطی غیر قطعی است. در این پروژه ما از کتابخانه Gymnasium استفاده میکنیم . با استفاده از Gymnasium علاوه بر پیاده سازی الگوریتم , میتوانیم به صورت گرافیکی نحوه اجرای الگوریتم را به صورت بصری ببینیم . محیطی که این پروژه در آن پیاده سازی شده و عامل قصد یادگیری آن را دارد ، محیط Cliff Walking است . این محیط امکان فعالیت عامل در فضای دو بعدی را فراهم می سازد

* جزئیات محیط و پیاده سازی

محیط Cliff Walking محیطی است برای یادگیری تقویتی در فضایی دو بعدی که در این پروژه به گونه‌ای شخصی‌سازی شده که تعداد 48 خانه مشخص داریم و در هر بار اجرادر آن تعداد صخره‌ی مشخصی به صورت رندوم در آن قرار می‎گیرند . خانه‌ ای که عامل از آن جست و جو را شروع میکند [3, 0] و خانه‌ی هدف [3, 11] است . عامل ما می‎تواند با کلید‎های 0، 1، 2و 3 به ترتیب به ترتیب و در صورت امکان به بالا، راست، پایین و چپ می‌رود . در ضمن عامل به صورت غیر قطعی عمل می‌کند و در صورت قصد برای یک حرکت تنها با احتمال 1/3 به آن سمت می‎رود و با احتمال 1/3 به دو سمت مجاور آن خواهد رفت . بازی در در صورت ورود به خانه‌ی صخره یا رسیدن به حالت هدف ، پایان می‌یابد و محیط دوباره بارگذاری میشود . امتیاز هر حرکت در این محیط برای خانه های غیر صخره -1 و امتیاز خانه‌ی صخره -100 است .

* الگوریتم حل مسئله

مسائل یادگیری تقویتی عموما با الگوریتم هایی حل میشوند که پیاده سازی آنها باید به گونه ای باشد که عامل بتواند در محیطی با ویژگی مشاهده پذیری جزئی به خوبی فعالیت کند . با توجه به محتوای درس , الگوریتم MDP ( Markov Decision Process ) برای حل این مسئله از یادگیری تقویتی در محیط دو بعدی انتخاب شد . فرایند تصمیم مارکوف الگوریتم و مدلی ریاضی برای حل مسائل ترتیبی است . در این الگوریتم یک عامل در هر مرحله با مجموعه ای از احتمالات و حالات روبرو است و کنشی را انتخاب میکند . انتخاب کنش موجب تغییر حالت و دریافت پاداش میشود . هدف عامل انتخاب سیاستی است که پاداش کل را به حداکثر برساند .



* پیاده سازی

*# Import nessary libraries  
import* numpy *as* np  
*import* gymnasium *as* gym  
*from* gymnasium.envs.toy\_text.cliffwalking *import* CliffWalkingEnv  
*from* gymnasium.error *import* DependencyNotInstalled  
*from* os *import* path

در ابتدای پیاده سازی این فاز به زبان پایتون , ماژول ها و کتابخانه های مورد نیاز را دخیل میکنیم . استفاده از کتابخانه Numpy روشی مرسوم و کارآمد برای انجام عملیات های عددی و ماتریسی است . به کمک این کتابخانه پیاده سازی الگوریتم های مختلف آسان میشود و در عین حال به دلیل استفاده از تکنیک های چند پردازشی , سرعت اجرای دستورات و به تبع الگوریتم ها بالاتر خواهد رفت .

استفاده از کتابخانه Gymnasium همانطور که قبلا گفته شد , پیاده سازی الگوریتم های یادگیری تقویتی در قالبی آماده و سهل انجام میشود . توابع آماده این کتابخانه به توسعه دهنده کمک میکند تمرکز خود را روی انتخاب سیاست بهینه برای حل مسئله بگذارد و وقت زیادی جهت تعریف محیط و پاداش از دست ندهد .

UP = 0  
RIGHT = 1  
DOWN = 2  
LEFT = 3  
image\_path = path.join(path.dirname(gym.\_\_file\_\_), "envs", "toy\_text")

کنش های ممکن برای عامل به صورت حرکات در دو بعد ممکن هستند . بالا , راست , پایین و چپ

*class* CliffWalking(CliffWalkingEnv):  
 *def* \_\_init\_\_(*self*, is\_hardmode=*True*, num\_cliffs=10, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 *self*.is\_hardmode = is\_hardmode  
  
 *# Generate random cliff positions  
 if self*.is\_hardmode:  
 *self*.num\_cliffs = num\_cliffs  
 *self*.\_cliff = np.zeros(*self*.shape, dtype=bool)  
 *self*.start\_state = (3, 0)  
 *self*.terminal\_state = (*self*.shape[0] - 1, *self*.shape[1] - 1)  
  
 *self*.cliff\_positions = []  
 *while* len(*self*.cliff\_positions) < *self*.num\_cliffs:  
 new\_row = np.random.randint(0, 4)  
 new\_col = np.random.randint(0, 11)  
 state = (new\_row, new\_col)  
 *if* (  
 (state *not in self*.cliff\_positions)  
 *and* (state != *self*.start\_state)  
 *and* (state != *self*.terminal\_state)  
 ):  
 *self*.\_cliff[new\_row, new\_col] = *True  
 if not self*.is\_valid():  
 *self*.\_cliff[new\_row, new\_col] = *False  
 continue  
 self*.cliff\_positions.append(state)  
  
 *# Calculate transition probabilities and rewards  
 self*.P = {}  
 *for* s *in* range(*self*.nS):  
 position = np.unravel\_index(s, *self*.shape)  
 *self*.P[s] = {a: [] *for* a *in* range(*self*.nA)}  
 *self*.P[s][UP] = *self*.\_calculate\_transition\_prob(position, [-1, 0])  
 *self*.P[s][RIGHT] = *self*.\_calculate\_transition\_prob(position, [0, 1])  
 *self*.P[s][DOWN] = *self*.\_calculate\_transition\_prob(position, [1, 0])  
 *self*.P[s][LEFT] = *self*.\_calculate\_transition\_prob(position, [0, -1])

کلاس Cliff Walking از قبل پیاده شده و شروط پاداش ها و خانه ها و شروع و پایان در آن مشخض شده اند . محیط 48 خانه دارد که تعداد 10 خانه از آن به صورت صخره یا Cliff و به صورت تصادفی از بین این 48 خانه انتخاب میشوند .

*def* step(*self*, action):  
 *if* action *not in* [0, 1, 2, 3]:  
 *raise* ValueError(f"Invalid action {action} must be in [0, 1, 2, 3]")  
  
 *if self*.is\_hardmode:  
 *match* action:  
 *case* 0:  
 action = np.random.choice([0, 1, 3], p=[1 / 3, 1 / 3, 1 / 3])  
 *case* 1:  
 action = np.random.choice([0, 1, 2], p=[1 / 3, 1 / 3, 1 / 3])  
 *case* 2:  
 action = np.random.choice([1, 2, 3], p=[1 / 3, 1 / 3, 1 / 3])  
 *case* 3:  
 action = np.random.choice([0, 2, 3], p=[1 / 3, 1 / 3, 1 / 3])  
  
 *return* super().step(action)

با توجه به محیط غیر قطعی تعریف شده , هر کنش عامل به احتمال 3/1 انجام خواهد شد . این ویژگی محیط به ویژگی های تابع step از کلاس پدر Cliff Walking اضافه شده تا به محیط چالش بیشتری ببخشد .

سایر توابع مربوط به پیاده سازی رابط گرافیکی را از این [لینک](https://mega.nz/file/oehyhRqS#euW3FMEck3mvnog2w2Nbwo0J83glyMUYNUP0Fa2waIQ) میتوان دنبال کرد .