تمرین ششم

(یادگیری تقویتی)

آرین محمدخانی- ۸۱۰۶۰۳۱۳۶

هوش مصنوعی

۱. مقدمه

این تمرین با هدف آشنایی با الگوریتم‌های مختلف یادگیری تقویتی و مقایسه‌ی عملکرد آن‌ها در محیط MountainCar-v0 طراحی شده است. پیاده‌سازی تمرین در دو بخش اصلی شامل اقدام‌های گسسته و اقدام‌های پیوسته انجام گرفته و هر بخش در دو قالب مختلف ارائه شده است؛ فایل‌های py. برای اجرای مستقیم و فایل‌های ipynb. جهت نمایش تعاملی کدها و نتایج در محیط Jupyter Notebook. نسخه‌ی PDF فایل‌های نوت‌بوک نیز به منظور سهولت مرور در بسته‌ی ارائه‌شده قرار گرفته است. برای کسب امتیاز اضافی، بخش‌های امتیازی تمرین شامل تحلیل‌های تکمیلی عملکرد مدل‌ها و مقایسه‌ی آن‌ها بر اساس معیارهای مختلف نیز پیاده‌سازی شده است. کدهای تمرین در یک مخزن GitHub بارگذاری شده تا دسترسی به آن‌ها آسان‌تر باشد. همچنین، به منظور نمایش بهتر نتایج آموزش، خروجی رندرهای محیط MountainCar-v0 برای تمام الگوریتم‌ها تولید و در پوشه‌ی gif ذخیره شده است.

ساختار کلی گزارش شامل مقدمه، بخش توضیح کد و بخش نتایج و بحث است. در بخش توضیح کد، ساختار کلی برنامه و الگوریتم‌های مورد استفاده به اختصار بیان شده و در بخش نتایج و بحث، نمودارها و خروجی‌ها تحلیل شده و علل عملکرد بهتر یا ضعیف‌تر مدل‌ها بر اساس معیارهایی همچون پاداش بر حسب تعداد اپیزود، زمان آموزش، دقت مدل‌ها و عملکرد در ده گام آزمایشی بررسی شده است. در این بخش، مدل‌های گسسته و پیوسته ابتدا به صورت جداگانه ارزیابی شده و در پایان با یکدیگر مقایسه شده‌اند.

جدول ۱، آدرس گیت‌هاب

لینک مخزن GitHub	https://github.com/ArianAZH/AI-Reinforcement-learning

۲. توضیح کد

در این بخش محیط‌های استفاده شده در تمرین بصورت جداگانه به همراه الگوریتم‌های یادگیری خود به‌طور مختصر، مورد بررسی قرار داده می‌شود. محیط‌ها با استفاده از شئ‌گرایی و بصورت Classساخته شده‌اند. در مرحله یادگیری، آموزش بر روی کلاس‌های ساخته شده انجام می‌شود و برای آزمایش و رندر محیط گرافیکی، از محیط MountainCar کتابخانه gym استفاده می‌شود.

۱/۲. محیط گسسته

بخش محیط گسسته در این کد یک نسخه ساده و سفارشی از محیط MountainCar را پیاده‌سازی می‌کند که در آن فضای حالت به‌صورت پیوسته تعریف شده، اما با استفاده از تابع discretize_state به تعداد مشخصی سطل (bucket) برای موقعیت و سرعت تقسیم می‌شود تا الگوریتم‌های جدولی مانند Q-learning و Double Q-learning بتوانند روی آن کار کنند. این محیط شامل پارامترهایی مثل محدودیت‌های موقعیت (۱.۲- تا ۰.۶)، محدودیت‌های سرعت (۰.۰۷- تا ۰.۰۷)، نیرو و گرانش برای شبیه‌سازی دینامیک ماشین است. عامل (agent) در هر مرحله با انتخاب یکی از سه عمل (حرکت به چپ، توقف، حرکت به راست) باعث تغییر سرعت و موقعیت می‌شود. اگر ماشین به قله سمت راست (موقعیت $\geq$ ۰.۵) برسد، پاداش مثبت دریافت می‌کند و اپیزود تمام می‌شود، وگرنه در هر مرحله پاداش منفی دریافت کرده و تلاش می‌کند در حداکثر ۲۰۰ گام به هدف برسد. این طراحی ساده اما گسسته‌سازی شده، امکان یادگیری و ارزیابی کارایی الگوریتم‌های مختلف را در شرایط کنترل شده و بدون نیاز به اجرای مستقیم محیط استاندارد Gym فراهم می‌کند. بر این محیط، سه الگوریتم یادگیری تقویتی مختلف پیاده‌سازی و آزمایش شده‌اند. که در ادامه توضیح داده شده است.

```python
class SimpleMountainCarEnv:
    def __init__(self):
        self.position_bounds = [-1.2, 0.6]
        self.velocity_bounds = [-0.07, 0.07]
        self.force = 0.001
        self.gravity = 0.0025
        self.max_steps = 200
        self.reset()

    def reset(self):
        self.position = np.random.uniform(-0.6, -0.4)
        self.velocity = 0.0
        self.steps = 0
        return np.array([self.position, self.velocity])

    def step(self, action):
        force_effect = (action - 1) * self.force
        gravity_effect = -np.cos(3 * self.position) * self.gravity

        self.velocity += force_effect + gravity_effect
        self.velocity = np.clip(self.velocity, *self.velocity_bounds)

        self.position += self.velocity
        self.position = np.clip(self.position, *self.position_bounds)

        if self.position in self.position_bounds:
            self.velocity = 0.0

        self.steps += 1
        done = False
        reward = -1

        if self.position >= 0.5:
            done = True
            reward = 1

        if self.steps >= self.max_steps:
            done = True

        return np.array([self.position, self.velocity]), reward, done
```

تصویر ۱، کلاس محیط گسسته

Q-learning

Q-leaning یکی از ساده‌ترین و پرکاربردترین الگوریتم‌های یادگیری تقویتی مبتنی بر جدول است که از معادله به‌روزرسانی Bellman استفاده می‌کند. در این روش، عامل یک جدول Q با ابعاد (تعداد سطل‌های موقعیت × تعداد سطل‌های سرعت × تعداد اعمال) نگهداری می‌کند که هر خانه نمایانگر ارزش تقریبی انجام یک عمل خاص در یک حالت خاص است. عامل با استفاده از یک سیاست ε-greedy بین جستجو (انتخاب تصادفی عمل) و بهره‌برداری (انتخاب بهترین عمل موجود در جدول) تعادل برقرار می‌کند. با گذشت زمان و به‌روزرسانی Q-table، مقدار ε کاهش می‌یابد تا عامل به‌تدریج کمتر به جستجو و بیشتر به استفاده از تجربه‌هایش بپردازد. این روش گرچه ساده و مؤثر است، اما در برخی شرایط دچار بیش‌براوردی (Overestimation) ارزش حالت-عمل‌ها می‌شود.

Double Q-learning

Double Q-learning برای رفع مشکل بیش‌براوردی در Q-learning معرفی شده است. ایده اصلی آن استفاده از دو جدول Q مجزا است. در هر به‌روزرسانی، یکی از جداول به‌صورت تصادفی انتخاب شده و برای محاسبه عمل بهینه از جدول دیگر استفاده می‌شود. این کار باعث می‌شود برآورد ارزش‌ها متعادل‌تر شود و عامل کمتر در اثر نویز یا نوسانات تصادفی به سمت انتخاب‌های اشتباه سوق پیدا کند. به‌عبارت دیگر، این الگوریتم برآورد ارزش و انتخاب عمل را از هم جدا می‌کند تا خطای سیستماتیک کاهش یابد. در محیط‌هایی مثل MountainCar که پاداش‌ها دیر دریافت می‌شوند، این ویژگی می‌تواند پایداری یادگیری را افزایش دهد.

DQN (Deep Q-Network)

DQN یک گام فراتر از Q-learning و Double Q-learning است و از شبکه‌های عصبی عمیق برای تقریب تابع Q استفاده می‌کند تا بتواند با فضاهای حالت پیوسته یا بسیار بزرگ کار کند. در این کد، شبکه عصبی با دو لایه مخفی (۱۲۸ و ۶۴ نورون) تعریف شده است که ورودی آن بردار حالت (موقعیت، سرعت) و خروجی آن سه مقدار Q برای سه عمل ممکن است.

برای پایداری بیشتر، از دو تکنیک کلیدی استفاده می‌شود:

- **Replay Buffer**: یک حافظه چرخشی که تجربه‌های عامل را ذخیره می‌کند و در هر به‌روزرسانی یک دسته تصادفی از این داده‌ها انتخاب می‌شود تا همبستگی زمانی داده‌ها کاهش یابد.
- **Target Network**: یک کپی از شبکه اصلی که به‌صورت دوره‌ای به‌روزرسانی می‌شود تا هدف یادگیری پایدارتر شود.

DQN برخلاف دو روش قبلی، نیازی به گسسته‌سازی فضای حالت ندارد و می‌تواند مستقیماً با مقادیر پیوسته کار کند، هرچند در این پیاده‌سازی همچنان از محیط ساده‌سازی‌شده استفاده شده است.

۲/۲. محیط پیوسته

بخش **محیط پیوسته** در این کد نیز، نسخه‌ای از محیط MountainCar را مدل‌سازی می‌کند که برخلاف نسخه گسسته، عامل می‌تواند نیرویی با مقدار پیوسته در بازه ۱- تا ۱ اعمال کند. این محیط در کلاسی با نام ContinuousMountainCar پیاده‌سازی شده و دارای ویژگی‌های اصلی زیر است:

- فضای حالت شامل دو متغیر پیوسته است: موقعیت در بازه ۱.۲- تا ۰.۶ و سرعت در بازه ۰.۰۷- تا ۰.۰۷.
- **کنترل پیوسته**: عامل می‌تواند در هر گام یک مقدار نیروی پیوسته بدهد که باعث تغییر سرعت و موقعیت بر اساس قوانین دینامیکی (نیروی محرک و نیروی گرانش) می‌شود.
- **پاداش و پایان اپیزود**: هر گام پاداش ۱- دارد تا عامل را به سمت حل سریع‌تر سوق دهد. اگر عامل به قله سمت راست (موقعیت $\geq$ ۰.۵) برسد یا ۲۰۰ گام تمام شود، اپیزود پایان یافته و پاداش بزرگ ۱۰۰ داده می‌شود.

این محیط فضای جستجوی بسیار بزرگ‌تری نسبت به نسخه گسسته ایجاد می‌کند، چون تعداد حالت‌ها و اعمال نامتناهی است، بنابراین روش‌های مبتنی بر جدول جوابگو نیستند و باید از الگوریتم‌های یادگیری تقویتی عمیق برای کنترل پیوسته استفاده شود. در این بخش نیز از دو الگوریتم برای آموزش استفاده شده است که در ادامه توضیح داده می‌شود.

```python
class SimpleMountainCarEnv:
    def __init__(self):
        self.position_bounds = [-1.2, 0.6]
        self.velocity_bounds = [-0.07, 0.07]
        self.force = 0.001
        self.gravity = 0.0025
        self.max_steps = 200
        self.reset()

    def reset(self):
        self.position = np.random.uniform(-0.6, -0.4)
        self.velocity = 0.0
        self.steps = 0
        return np.array([self.position, self.velocity])

    def step(self, action):
        force_effect = (action - 1) * self.force
        gravity_effect = -np.cos(3 * self.position) * self.gravity

        self.velocity += force_effect + gravity_effect
        self.velocity = np.clip(self.velocity, *self.velocity_bounds)

        self.position += self.velocity
        self.position = np.clip(self.position, *self.position_bounds)

        if self.position in self.position_bounds:
            self.velocity = 0.0

        self.steps += 1
        done = False
        reward = -1

        if self.position >= 0.5:
            done = True
            reward = 1

        if self.steps >= self.max_steps:
            done = True

        return np.array([self.position, self.velocity]), reward, done
```

تصویر ۲، کلاس محیط پیوسته

DQN با اعمال گسسته‌سازی‌شده

در نسخه پیوسته محیط، چون DQN ذاتاً برای اعمال گسسته طراحی شده است، ابتدا فضای عمل پیوسته به چند مقدار ثابت بین ۱- و ۱ تقسیم می‌شود. سپس شبکه عصبی، همانند نسخه استاندارد DQN، مقدار Q هر یک از این اعمال گسسته را پیش‌بینی می‌کند. عامل همچنان با سیاست ε-greedy بین جستجو و بهره‌برداری تعادل برقرار می‌کند. این رویکرد ساده و قابل‌اجراست، اما به دلیل گسسته‌سازی اجباری، قادر به استفاده کامل از پتانسیل فضای عمل پیوسته نیست و عملکرد آن معمولاً پایین‌تر از الگوریتم‌های ویژه کنترل پیوسته است.

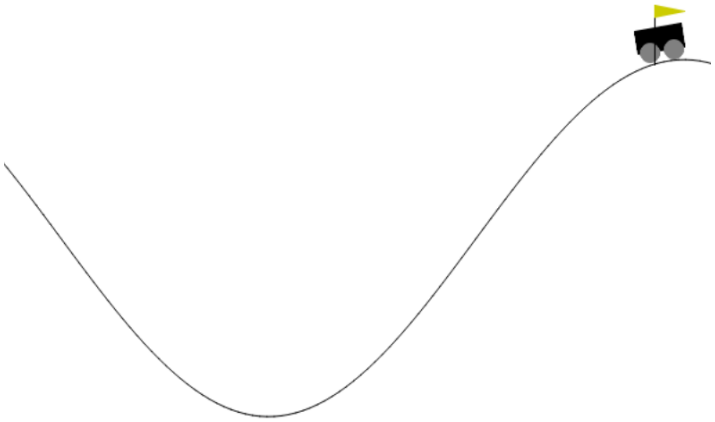
DDPG (Deep Deterministic Policy Gradient)

DDPG یک الگوریتم actor-critic آف‌پالیسی برای کنترل پیوسته است که هم‌زمان یک شبکه Actor برای تولید عمل بهینه و یک شبکه Critic برای برآورد ارزش Q حالت-عمل را آموزش می‌دهد. Actor به‌صورت مستقیم یک مقدار عمل پیوسته پیشنهاد می‌دهد و Critic این عمل را ارزیابی می‌کند. از Replay Buffer برای کاهش همبستگی داده‌ها و از Target

Networks با به‌روزرسانی نرم (Polyak averaging) برای پایداری بیشتر استفاده می‌شود. برای اکتشاف در فضای عمل، از نویز فرآیند Ornstein–Uhlenbeck استفاده می‌شود که به عامل کمک می‌کند اعمال متنوع و پیوسته را امتحان کند. مزیت اصلی DDPG این است که بدون نیاز به گسسته‌سازی می‌تواند سیاست بهینه را برای فضای عمل کاملاً پیوسته بیاموزد و بنابراین برای محیط‌هایی مثل MountainCarContinuous بسیار مناسب است.

۳. نتایج و بحث

در این بخش نتایج محیط‌های گسسته و پیوسته جدا از هم مورد بررسی و در پایان مورد مقایسه قرار می‌گیرند. در تمامی نتایج همانطور که در تصویر ۳ نمایش داده شده است، به هدف خواهند رسید. بدلیل عدم توانایی نمایش رندرها در گزارش، تمامی آن‌ها در پوشه‌ای جداگانه ارائه شده است.

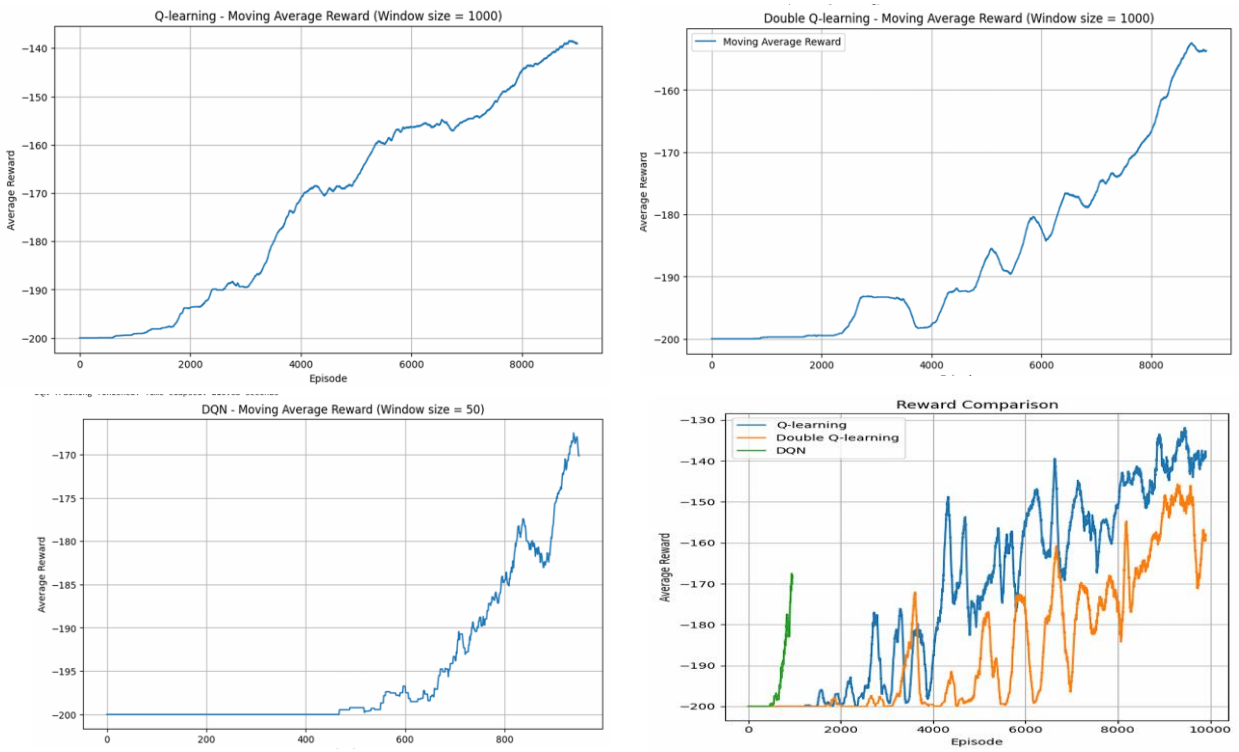


تصویر ۳، نمونه‌ای از نمایش رندر محیط گرافیکی، گیف‌های تمامی الگوریتم‌ها داخل **فایل** موجود می‌باشد.

در این تمرین، چندین بار حالت‌های مختلف بررسی شده است و بدلیل وجود عناصر تصادفی در فرآیند یادگیری مانند مقداردهی اولیه وزن‌های شبکه، انتخاب تصادفی اکشن‌ها در چارچوب سیاست ε-greedy، نمونه‌گیری تصادفی از حافظه تجربه و همچنین شرایط اولیه متغیر در محیط، نتایج هر بار اجرا دقیقاً یکسان به دست نمی‌آیند و اختلاف‌های جزئی در میانگین پاداش و سرعت همگرایی مشاهده می‌شود.

۱/۳. نتایج محیط گسسته

نتایج آموزش Q-learning نشان می‌دهد که با ۱۰,۰۰۰ اپیزود، پاداش‌ها در ابتدا منفی و نسبتا ثابت بودند (۲۰۰-) و به تدریج با افزایش اپیزودها بهبود یافتند و به حدود ۱۵۷- رسیدند. مقدار Epsilon نیز از ۰.۳۶۸ به ۰.۰۵ کاهش یافت که نشان‌دهنده کاهش تدریجی کاوش و تمرکز روی بهره‌برداری از دانش آموخته‌شده است. زمان آموزش این الگوریتم ۲۴.۷۳ ثانیه بود. در Double Q-learning، با همان تعداد اپیزود، مانند Q-learning پاداش‌ها در ابتدا ثابت منفی (۲۰۰-) بودند و پس از همان تعداد اپیزود بهبود یافتند و حداکثر به ۱۵۳- رسیدند. مقدار Epsilon در این الگوریتم ثابت روی ۰.۱ نگه داشته شد که باعث شد کاوش به صورت تدریجی و پایدار انجام شود و زمان آموزش آن ۲۹.۰۵ ثانیه بود. در مقابل، DQN با تنها ۱۰۰۰ اپیزود توانست پاداش‌ها را از ۲۰۰- به ۱۶۰- بهبود دهد، هرچند زمان آموزش آن به دلیل محاسبات سنگین شبکه عصبی بسیار بیشتر و برابر با ۲۱۶.۶۸ ثانیه بود.



تصویر ۴، نتایج آموزش بر روی الگوریتم‌های مختلف

بررسی جدول مقایسه مدل‌ها بعد از ۱۰ بار آموزش مجدد نشان می‌دهد که DQN برای رسیدن به هدف کمترین میانگین تعداد گام‌ها (۱۳۹) را دارد، در حالی که Q-learning و Double Q-learning با تعداد گام‌های ۱۴۴.۴و ۱۵۲.۹ به نرخ موفقیت یکسان با DQN رسیده‌اند.

جدول 2 ،جدول مقایسه

Model	Avg Steps	Success Rate	Episodes	Train Time (s)
Q-learning	144.4	1.00	10000	24.7
Double Q-learning	152.9	1.00	10000	29.0
DQN	139.4	1.00	1000	217.7

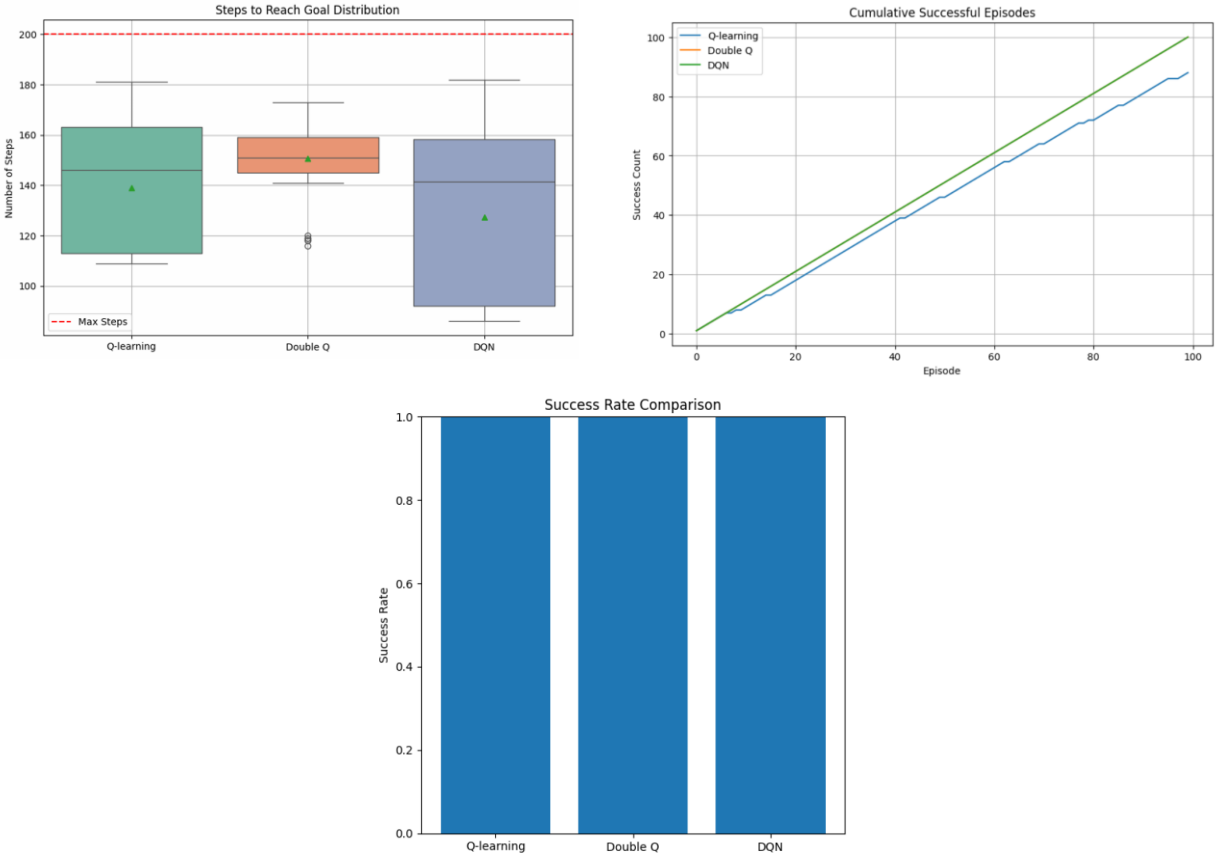
جهت کسب نمره امتیازی، دوباره مدل‌ها یک در طی یک آموزش با تعداد اپیزود ۱۰۰ قرار گرفتند. تحلیل دوباره عملکرد زمان-تا-هدف نیز نشان می‌دهد که DQN همچنان سریع‌ترین الگوریتم با انحراف معیار ۳۳.۳ است. نکته جالب در این تست، کاهش تعداد گام‌های Q-Learning نسبت به Double Q-learning می‌باشد.

جدول 3،جدول مقایسه

Model	Mean Steps	Std Dev	Success Rate
Q-learning	139.0	24.9	1.00
Double Q-learning	150.6	11.5	1.00
DQN	127.4	33.3	1.00

در مجموع، DQN بهترین عملکرد را از نظر موفقیت و میانگین تعداد گام‌ها ارائه می‌دهد اما زمان آموزش طولانی دارد. Double Q-learning با پایداری بالاتر نسبت به Q-learning، عملکرد موفق‌تری دارد اما تعداد گام‌ها بیشتر است. Q-learning سریع‌ترین الگوریتم از نظر زمان

آموزش است اما در پایداری و عملکرد کلی ضعیف‌تر عمل می‌کند. بر این اساس، اگر سرعت آموزش و منابع محاسباتی محدود اهمیت دارند، Q-learning یا Double Q-learning انتخاب مناسبی هستند و اگر دقت و موفقیت بالا در اولویت باشد و منابع کافی موجود باشد، DQN بهترین انتخاب خواهد بود. در تصاویر زیر سایر نمودارها از جمله نمودار جعبه‌ای گام‌ها برای رسیدن به هدف و نمودارها موفقیت الگوریتم‌ها نمایش داده شده است.



تصویر ۵، سایر نمودارهای بررسی نتایج

نمودار جعبه‌ای (Boxplot) در سمت چپ نشان می‌دهد که الگوریتم DQN در مقایسه با Q-learning و Double Q-learning توزیع تعداد گام‌های کمتری برای رسیدن به هدف دارد، به‌طوری‌که میانه (Median) آن پایین‌تر است و دامنه تغییرات آن نسبتاً گسترده‌تر می‌باشد. این گستردگی بیشتر به دلیل استفاده از شبکه عصبی و حساسیت آن به شرایط اولیه و نوسان در

فرایند یادگیری است. در مقابل، Q-learning توزیع نسبتاً فشرده‌تری دارد اما میانگین آن بالاتر از DQN است و نشان می‌دهد که به طور پایدار اما با گام‌های بیشتر به هدف می‌رسد. Double Q-learning هم پایداری بهتری نسبت به Q-learning نشان داده، اما میانگین گام‌هایش همچنان بیشتر از DQN است. و اما در سمت راست، نمودار سمت راست (Cumulative Successful Episodes) نیز نشان می‌دهد که الگوریتم DQN در تمامی ۱۰۰ اپیزود آزمایش موفق به رسیدن به هدف شده و نرخ موفقیت تجمعی آن خطی و کامل است.

و اما در ادامه به پاسخ سوالات مطرح شده، در فایل سوال پرداخته می‌شود.

در تحلیل **بررسی تأثیر تابع هزینه، الگوریتم بهینه‌سازی و تابع پاداش،** مجموعه‌ای از آزمایش‌ها در محیط MountainCar انجام شد (جدول4). هدف این آزمایش‌ها، بررسی اثر سه عامل نوع تابع خطا(Loss Function) ، نوع بهینه‌ساز (Optimizer) و نوع تابع پاداش (Reward Function) بر عملکرد الگوریتم DQN بود. بر اساس نتایج، بهترین عملکرد مربوط به استفاده از MSELoss همراه با Adam و تابع پاداش اصلی (original_reward) در آزمایشExp1 بوده که میانگین پاداش را به حدود ۱۲۳.۵۴- رسانده است. این نتیجه نشان می‌دهد که در این مسئله، MSELoss توانسته پایداری بیشتری در به‌روزرسانی وزن‌ها ایجاد کند، به‌خصوص وقتی با Adam ترکیب شده است. بهینه‌ساز Adam نیز در مقایسه با RMSprop، در اکثر سناریوها عملکرد بهتری ارائه داده و توانسته عامل را سریع‌تر به سمت سیاست‌های کارآمدتر هدایت کند. با این حال، استفاده از تابع پاداش تغییریافته (modified_reward) در اغلب موارد منجر به افت عملکرد شده است. برای مثال، در آزمایش Exp4 (MSELoss + Adam + modified_reward) میانگین پاداش به ۱۳۹.۱۰- کاهش یافته و در آزمایش Exp6 (SmoothL1Loss + Adam + modified_reward) این مقدار به ۱۶۰.۹۴- رسیده است. این افت می‌تواند ناشی از این باشد که پاداش اضافی برای حرکت به سمت راست، عامل را به رفتارهایی هدایت کرده که الزاماً منجر به رسیدن به قله نمی‌شوند. از نظر زمان آموزش، همه‌ی ترکیب‌ها بین ۲۲۱ تا ۲۵۳ ثانیه زمان برده‌اند، که نشان می‌دهد اختلاف عملکرد بیشتر ناشی از کیفیت یادگیری است تا سرعت اجرا. در مجموع،

MSELoss + Adam + original_reward بهترین ترکیب و SmoothL1Loss + RMSprop + original_reward (Exp5) با میانگین پاداش ۲۰۰- ضعیف‌ترین ترکیب بوده است.

جدول4 ، تاثیر هایپرپارامترها بر DQN

Experiment	Loss	Optimizer	Reward Function	Average Reward	Training Time (s)
Exp1	MSELoss	Adam	original_reward	-123.54	229.10
Exp2	SmoothL1Loss	Adam	original_reward	-155.94	237.32
Exp3	MSELoss	RMSprop	original_reward	-139.84	221.76
Exp4	MSELoss	Adam	modified_reward	-139.10	241.67
Exp5	SmoothL1Loss	RMSprop	original_reward	-200.00	252.04
Exp6	SmoothL1Loss	Adam	modified_reward	-160.94	253.72

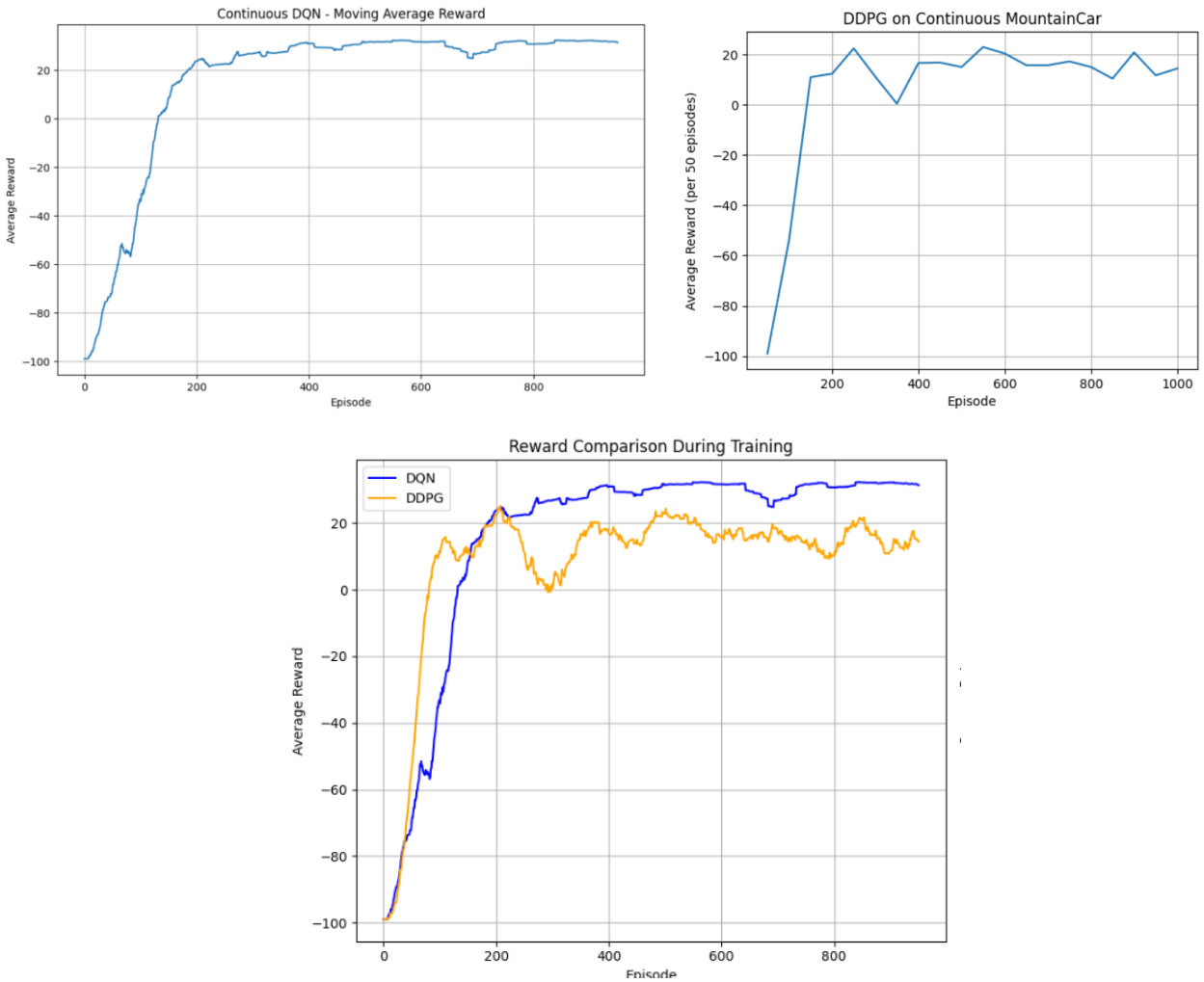
کدام مدل پاسخ دقیق‌تری دارد و دلیل آن چیست؟ مدلی که بیشترین دقت را دارد، DQN است. دلیل این دقت بالا این است که DQN توانسته در تمامی اپیزودها به هدف برسد و نرخ موفقیت آن برابر با ٪۱۰۰ است، یعنی همه مسیرها و تصمیمات آن درست بوده‌اند. همچنین میانگین تعداد گام‌ها در DQN کمتر از دو الگوریتم دیگر است (۱۳۲.۱ گام) که نشان‌دهنده انتخاب مسیرهای بهینه و کاهش گام‌های غیرضروری است. استفاده از شبکه عصبی عمیق در DQN باعث شده الگوریتم بتواند الگوهای پیچیده محیط را بهتر یاد بگیرد و تصمیم‌گیری‌های دقیق‌تری انجام دهد، در حالی که Q-learning و Double Q-learning به دلیل ساده‌تر بودن و جدول‌بندی محدودیت‌های محیط، دقت کمتری دارند.

درباره رابطه بین دقت و زمان آموزش، رابطه واضحی بین دقت و زمان آموزش مشاهده می‌شود. DQN با دقت بالا و میانگین گام کمتر، زمان آموزش طولانی‌تری دارد (۲۱۶ ثانیه) ؛ این به دلیل محاسبات سنگین شبکه عصبی و یادگیری ویژگی‌های پیچیده محیط است. در مقابل، Q-learning سریع‌ترین الگوریتم از نظر زمان آموزش (۲۴.۷۳ ثانیه) است اما دقت آن پایین‌تر است، زیرا جدول Q قادر به مدل‌سازی پیچیدگی محیط به اندازه شبکه عصبی نیست. Double Q-

learning تعادلی بین این دو ارائه می‌دهد؛ زمان آموزش آن کمی بیشتر از Q-learning است (۲۹.۰۵ ثانیه) و دقت و پایداری نسبتاً خوبی دارد (با انحراف معیار کمتر). بنابراین در این حالت، افزایش دقت معمولاً با افزایش زمان آموزش همراه است و انتخاب الگوریتم مناسب بستگی به محدودیت‌های زمانی و نیاز به دقت دارد. این استدلال، تنها با بررسی نتایج ارائه داده بدست نیامده است. در چندین بار آموزش‌های مختلف، اکثرا Double Q-learning نتایج بهتری از Q-learning ارائه داده است و به همین دلیل می‌توان گفت در اکثر مواقع دقت نسبتا بهتری نسبت به DQN دارد.

۲/۳. نتایج محیط پیوسته

DQN در اصل برای محیط‌های گسسته طراحی شده، ولی در این مورد به احتمال زیاد با تبدیل action پیوسته به گسسته اجرا شده است، بنابراین یادگیری آن نسبتاً سریع و پایدار بوده است و پاداش‌ها بعد از چند صد اپیزود تقریباً ثابت و بالا شده‌اند. این نشان می‌دهد که شبکه Q توانسته به سرعت یک policy مناسب روی actionهای گسسته پیدا کند و نوسانات کمی دارد. در مقابل، DDPG یک الگوریتم actor-critic برای محیط‌های پیوسته است و از exploration مبتنی بر نویز Ornstein–Uhlenbeck استفاده می‌کند. روند یادگیری آن در ابتدا کند است و پاداش‌ها بسیار منفی بوده‌اند، زیرا شبکه actor هنوز نمی‌داند چه actionهایی به پاداش مثبت منجر می‌شوند و نویز exploration باعث نوسانات زیادی در عملکرد می‌شود. با افزایش تعداد اپیزودها، actor و critic به تدریج یاد می‌گیرند و میانگین پاداش مثبت و بالاتر می‌رود، ولی هنوز نوسانات بیشتری نسبت به DQN دیده می‌شود. این تفاوت‌ها نشان می‌دهد که در محیط‌های پیوسته، DQN با گسسته‌سازی سریع‌تر به یک رفتار خوب می‌رسد ولی محدودیت‌های actionگسسته را دارد، در حالی که DDPG به دلیل استفاده از actionهای پیوسته، توانایی یادگیری policy دقیق‌تر و بهینه‌تر را دارد، ولی نیازمند تعداد اپیزودهای بیشتر و صبر طولانی‌تر برای کاهش نوسانات و رسیدن به پاداش پایدار است.



تصویر ۶، نمودارهای آموزش

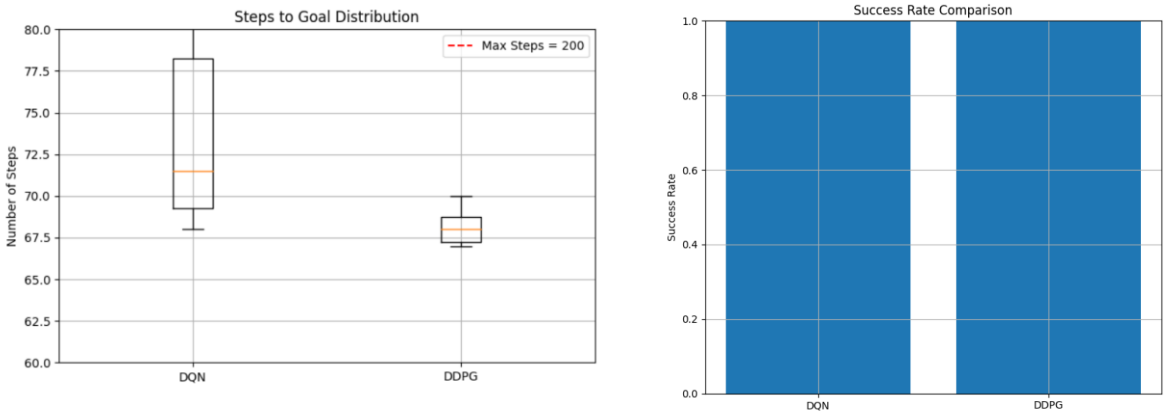
با توجه به جدول مقایسه در تست با آموزش ۱۰ اپیزود، دو الگوریتم DQN و DDPG را در یک محیط پیوسته مقایسه می‌کند. میانگین تعداد گام‌ها تا رسیدن به هدف نشان می‌دهد که DDPGبا ۶۸.۱ گام کمی سریع‌تر از DQN با ۷۳.۴ گام عمل می‌کند، که بیانگر توانایی DDPG در یافتن مسیرهای بهینه‌تر است. نرخ موفقیت هر دو الگوریتم برابر با ۱ است، یعنی هر دو به‌طور مداوم به هدف می‌رسند و از نظر قابلیت اطمینان عملکرد مشابهی دارند. با این حال، زمان آموزش DDPGبه‌طور قابل توجهی بیشتر است (۳۷۶.۷ ثانیه در مقابل ۱۰۶.۵ ثانیه برای DQN)، که ناشی از پیچیدگی شبکه‌های actor و critic و یادگیری در فضای عمل پیوسته است. به طور

کلی، DQN گزینه‌ای سریع و قابل اعتماد است، در حالی که DDPG می‌تواند سیاست‌های دقیق‌تر و بهینه‌تر ارائه دهد، اما با هزینه زمان آموزش بیشتر.

جدول 5، جدول مقایسه

Model	Avg Steps	Success Rate	Training Time (s)
DQN	73.4	1.00	106.5
DDPG	68.1	1.00	376.7

و در ادامه سایر نمودارها از جمله میزان موفقیت و نمودارجعبه‌ای برای تعداد گام‌های نشان داده شده است.



تصویر ۷، نمودارهای مقایسه

نمودار جعبه‌ای سمت چپ نشان می‌دهد که الگوریتم DDPG توزیع تعداد گام‌های کمتری برای رسیدن به هدف دارد و میانه (Median) آن پایین‌تر از DQN است، به‌طوری که میانگین گام‌های آن در حدود ۶۸ است در حالی که DQN حدود ۷۳ گام نیاز داشته است. همچنین پراکندگی مقادیر (Variance) در DDPG کمتر است که نشان می‌دهد این الگوریتم عملکرد باثبات‌تری در محیط پیوسته ارائه می‌دهد. در مقابل، DQN اگرچه کمی بیشتر گام برمی‌دارد، اما به دلیل طراحی ساده‌تر، نوسانات بیشتری در تعداد گام‌ها دارد.

و در ادامه به سوالات مطرح شده در فایل پروژه پرداخته می‌شود.

کدام مدل پاسخ دقیق‌تری دارد و دلیل آن چیست؟ هر دو مدل DQN و DDPG در محیط پیوسته نرخ موفقیت ٪۱۰۰ دارند، اما DDPG، میانگین گام کمتری (۶۸ در مقابل ۷۳.۴) دارد. این نشان می‌دهد که DDPG مسیر بهینه‌تری برای رسیدن به هدف انتخاب می‌کند و بنابراین پاسخ دقیق‌تری ارائه می‌دهد. دلیل این دقت، استفاده DDPG از شبکه‌های پیوسته Actor-Critic است که به جای انتخاب گسسته عمل‌ها، امکان تولید اعمال پیوسته و دقیق‌تر را فراهم می‌کند و در محیط‌هایی با فضای اقدام پیوسته، باعث عملکرد بهینه‌تر می‌شود. DQN به دلیل محدودیت گسسته بودن اقدامات، مسیر کمی طولانی‌تر دارد ولی هنوز عملکرد موفقیت کامل دارد.

سؤال ۲: درباره رابطه بین دقت و زمان آموزش بحث کنید از جدول مقایسه مشخص است که DDPG برای رسیدن به عملکرد دقیق‌تر و میانگین گام کمتر، زمان آموزش طولانی‌تری نیاز دارد (۳۷۶.۷ ثانیه در مقابل ۱۰۶.۵ ثانیه برای DQN). این نشان می‌دهد که افزایش دقت و بهینه‌سازی مسیر در محیط‌های پیچیده‌تر، هزینه محاسباتی و زمان آموزش بالاتری دارد. بنابراین همیشه باید بین دقت و زمان آموزش یک تعادل برقرار کرد: اگر منابع محاسباتی محدود و زمان آموزش اهمیت دارد، DQN انتخاب مناسبی است، اما اگر هدف دقت بالا و مسیر بهینه‌تر است و زمان کافی برای آموزش وجود دارد، DDPG مدل بهتری است.

۳/۳. مقایسه و جمع‌بندی

با توجه به نتایج گزارش، می‌توان یک مقایسه کلی و پاراگرافی بین محیط‌های گسسته و پیوسته ارائه داد. در محیط‌های گسسته، الگوریتم‌های Q-learning، Double Q-learning وDQN عملکرد متفاوتی داشتند. Q-learning با ۱۰,۰۰۰ اپیزود سریع‌ترین زمان آموزش را داشت. Double Q-learning با زمان آموزش کمی بیشتر عملکرد پایدارتر و نرخ موفقیت بالاتر ارائه کرد، ولی میانگین تعداد گام‌های آن بیشتر بود. DQN با وجود زمان آموزش طولانی‌تر دقیق‌ترین عملکرد را داشت، تمامی اپیزودها موفق بودند و میانگین گام‌ها کمترین مقدار را نشان می‌داد، زیرا شبکه عصبی آن توانست الگوهای پیچیده محیط را بهتر یاد بگیرد و مسیرهای

بهینه‌تری انتخاب کند. بنابراین در محیط گسسته، بین دقت و زمان آموزش یک رابطه مستقیم وجود دارد: افزایش دقت معمولاً با افزایش زمان آموزش همراه است، و انتخاب الگوریتم مناسب بستگی به محدودیت‌های زمانی و منابع محاسباتی دارد.

در محیط‌های پیوسته، نتایج کمی متفاوت است. هر دو الگوریتم DQN و DDPG توانستند به موفقیت کامل دست یابند و نرخ موفقیت ٪۱۰۰ داشتند، اما میانگین تعداد گام‌ها کمی متفاوت بود؛ DDPG مسیر بهینه‌تر و میانگین گام کمتر (۶۸) نسبت به DQN (۷۳.۴) داشت. با این حال، زمان آموزش DDPG بسیار طولانی‌تر بود. این نشان می‌دهد که در محیط‌های پیوسته، استفاده از الگوریتم‌های پیچیده‌تر Actor-Critic برای رسیدن به دقت بالا و مسیر بهینه، هزینه محاسباتی و زمان آموزش بیشتری می‌طلبد، در حالی که DQN می‌تواند با زمان آموزش کمتر عملکرد موفقیت مناسبی ارائه دهد، اما مسیر کمی طولانی‌تر است و ممکن است در تعمیم به شرایط پیچیده‌تر محدودیت داشته باشد.

به طور کلی، مقایسه بین محیط‌های گسسته و پیوسته نشان می‌دهد که محیط‌های پیوسته پیچیده‌تر هستند و نیازمند الگوریتم‌های قدرتمندتری برای دستیابی به دقت بالا و مسیر بهینه هستند اما در تعداد گام‌های کمتر می‌توانند به هدف برسند. در محیط‌های گسسته، الگوریتم‌های ساده‌تر می‌توانند با زمان آموزش کمتر عملکرد قابل قبولی ارائه دهند. در هر دو نوع محیط، همواره بین دقت و زمان آموزش یک رابطه وجود دارد: مدل‌های دقیق‌تر و بهینه‌تر معمولاً به زمان آموزش بیشتری نیاز دارند و انتخاب بهترین الگوریتم باید بر اساس محدودیت‌های زمانی، منابع محاسباتی و اهمیت دقت و بهینه‌سازی مسیر انجام شود.