



پروژه کارشناسی

فرود خودکار سازه پرنده با استفاده از یادگیری تقویتی و شبکه‌های عمیق Q

استاد راهنما

دکتر جمشید باقرزاده

نویسنده

پارسا محمودی

974421040

تابستان 1401

فهرست

۳	مقدمه
۳	مسئله
۳	یادگیری تقویتی Reinforcement Learning
۴	Q-Learning
۵	پیاده سازی Q-Learning
۶	تخمین Q با رابطه بلمن
۶	Deep Q-Learning
۷	بروزرسانی وزن شبکه با استفاده از معادله بلمن
۷	نحوه نگاشت حالت ها به جفت (Action, Q-value)
۸	دلیل استفاده از DQN
۸	A2C
۸	A3C
۱۰	OpenAI Gym
۱۰	Box2D
۱۰	Lunar Lander
۱۲	پیاده سازی برنامه
۲۱	نتیجه
۲۲	References

مقدمه

هوش مصنوعی امروزه پیشرفت بسیاری کرده است. بسیاری از وظایف تکراری، زمان گیر و یا مسائل پیچیده را به راحتی و با سرعت بیشتری می‌تواند حل کند و بار بسیار سنگینی از بارهای انسان‌ها را کم کرده است.

امروزه از این تکنولوژی در زمینه‌های بسیاری از قبیل پزشکی، اقتصادی، کشاورزی، تحصیلی و ... استفاده می‌شود و بشر همواره به دنبال آن است تا با استفاده از هوش مصنوعی در زمینه‌های جدید مسائل جدید را به راحتی و با سرعت بیشتری حل کند یا راه حلی برای آن‌ها بیابد.

مسئله

امروزه با پیشرفت صنعت هوا و فضا و کامپیوترها سازمان‌هایی از قبیل SpaceX و یا NASA به دنبال آن هستند تا در حد امکان فرآیند فرود سازه‌ها و فضاپیماهای خود را بدلیل تاخیر زیاد ارتباط از راه دور و احتمال خطای بالای انسان، کاملاً خودکار کنند. برای این منظور می‌توان از هوش مصنوعی و دقیق‌تر، از یادگیری ماشین کمک گرفت.

هدف ما در این پروژه طراحی یک سامانه هوشمند برای فرود یک فضاپیما بر سطح ماه به صورت خودکار با استفاده از شبکه‌های عمیق Q-Learning می‌باشد.

یادگیری تقویتی Reinforcement Learning

یادگیری تقویتی یکی زمینه‌های یادگیری ماشین است که توجه و محبوبیت زیادی را به خود جلب کرده است. یکی از دلایل مهم این محبوبیت به دلیل پیشرفت در یادگیری تقویتی است که در آن الگوریتم‌های کامپیوتری مانند Alpha Go و OpenAI Five توانسته‌اند در بعضی از مسائل عملکردی در سطح انسانی داشته باشند.

یادگیری تقویتی به نوعی علم گرفتن تصمیمات بهینه با استفاده از تجربیات است. مراحل این فرایند به صورت زیر است:

- رصد کردن محیط
- تصمیم گرفتن در مورد نحوه عمل با استفاده از برخی استراتژی ها
- شروع به کار می کند
- دریافت پاداش یا جریمه
- از تجربه بدست آمده یاد می گیرد و استراتژی را تصحیح می کند
- تا زمانی که یک استراتژی درست بدست نیامده است تکرار می کند

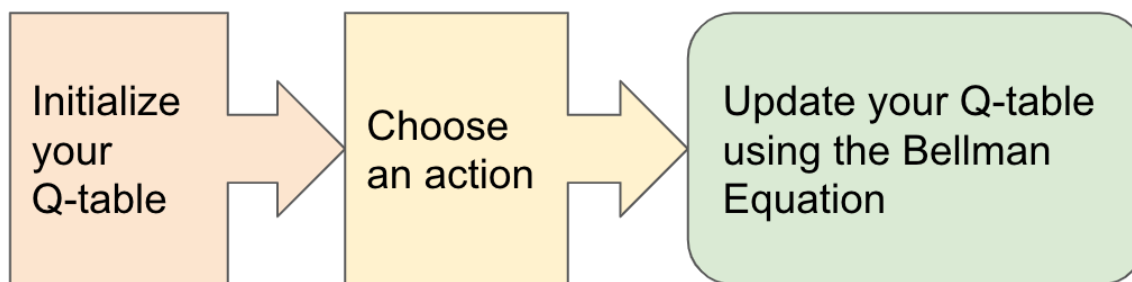
یادگیری تقویتی را می توان به طور کلی به دو گروه تقسیم کرد: الگوریتم های بدون مدل و الگوریتم های مبتنی بر مدل.

الگوریتم های بدون مدل، مدلی از تابع انتقال محیط خود را برای پیش بینی وضعیت ها و پاداش های آینده نمی آموزند. روش های Q-Learning، Deep Q-Networks و Policy Gradient الگوریتم های بدون مدل هستند زیرا مدلی از تابع انتقال محیط ایجاد نمی کنند.

Q-Learning

Q-Learning یک الگوریتم بدون مدل است. همچنین این الگوریتم مبتنی بر مقدار (Value-based) نیز می باشد. الگوریتم Value based، ارزش یک تابع را بر پایه یک معادله بروزرسانی می کنند، به عنوان مثال با معادله Bellman در حالی که نوع دیگر آن، الگوریتم مبتنی بر سیاست (policy-based) ارزش یک تابع را با یک سیاست حریصانه بهبود می بخشد.

Q-Learning یک یادگیرنده بدون سیاست است. به این معنا که ارزش های سیاست بهینه شده را سوا از انتخاب های عامل یاد می گیرد. به طوری که، یک یادگیرنده با سیاست، ارزش های سیاست گذاری شده را با کمک عامل می آموزد.



۱. مقداردهی اولیه Q-table

۲. انتخاب حرکت با استفاده از استراتژی اکتشاف اپسیلون-گریدی

۳. بروزرسانی جدول Q را با استفاده از معادله بلمن

Q-table یک جدول ساده است که ما از آن برای پیگیری وضعیت ها، اقدامات و پاداش های مورد انتظار آنها استفاده می کنیم. به طور خاص، جدول Q یک جفت حالت-عمل را به یک Q-value (مقدار بهینه تخمینی آینده) که عامل یاد خواهد گرفت، نگاشت می کند. در شروع الگوریتم Q-Learning، جدول Q به تمام صفرها مقداردهی می شود که نشان می دهد عامل چیزی در مورد جهان نمی داند. همانطور که عامل از طریق آزمون و خطا، اقدامات مختلف را در حالت های مختلف امتحان می کند، عامل پاداش مورد انتظار هر جفت حالت-عمل را می آموزد و جدول Q را با Q-value جدید به روز می کند. استفاده از آزمون و خطا برای یادگیری جهان را اکتشاف یا Exploration می نامند.

یکی از اهداف الگوریتم Q-Learning یادگیری Q-Value برای یک محیط جدید است. Q-Value حداکثر پاداش مورد انتظاری است که یک عامل با انجام یک عمل A از حالت S می تواند به آن دست یابد. پس از اینکه یک عامل ارزش Q هر جفت حالت-عمل را یاد گرفت، عامل در حالت S پاداش مورد انتظار خود را به حداکثر می رساند. انتخاب عمل A با بالاترین پاداش مورد انتظار.

تخمین Q با رابطه بلمن

معادله بلمن به ما می گوید که چگونه جدول Q خود را بعد از هر مرحله به روز کنیم. برای خلاصه کردن این معادله، عامل مقدار درک شده فعلی را با پاداش تخمینی بهینه آینده به روزرسانی می کند که فرض می کند عامل بهترین اقدام فعلی شناخته شده را انجام می دهد. در یک پیاده سازی، عامل در تمام اقدامات برای یک وضعیت خاص جستجو می کند و جفت حالت-عمل با بالاترین مقدار Q مربوطه را انتخاب می کند.

$$Q(S_t, A_t) = (1 - \alpha) Q(S_t, A_t) + \alpha * (R_t + \lambda * \max_a Q(S_{t+1}, a))$$

S = وضعیت یا حالت

A = اقدامی که عامل انجام می دهد یا حرکت عامل

R = پاداش انجام یک اقدام

t = مرحله زمانی

α = نرخ یادگیری

λ = ضریب کاهش که باعث می شود پاداش ها ارزش خود را در طول زمان از دست بدهند.

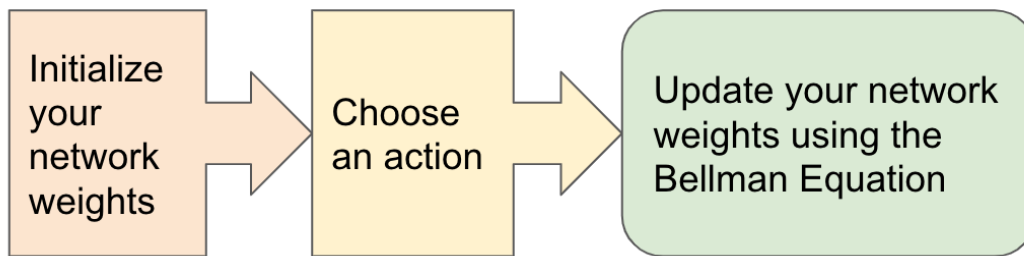
Deep Q-Learning

یکی از مفاهیم اصلی در یادگیری تقویتی، الگوریتم Deep Q-Learning.

در این روش بجای استفاده از جدولی برای نگهداری و به روزرسانی مقادیر Q، از شبکه های عصبی استفاده می شود. بجای تخصیص هر (S,a) به یک مقدار Q، در شبکه های عصبی هر زوج (Q, a) به یک حالت ورودی تخصیص داده می شود.

Q-Learning: یک جدول هر جفت حالت-عمل را به Q-مقدار مربوطه خود ترسیم می کند

Deep Q-Learning: یک شبکه عصبی حالت های ورودی را به جفت (عمل، Q-value) ترسیم می کند.



۱. شبکه های عصبی اصلی و هدف خود را راه اندازی کنید
۲. انتخاب عمل با استفاده از استراتژی اکتشاف اپسیلون-گریدی
۳. بروزرسانی وزن شبکه با استفاده از معادله بلمن

تفاوت اصلی بین Deep Q-Learning و Q-Learning اجرای جدول Q است. به طور دقیق، Deep Q-Learning جدول Q معمولی را با یک شبکه عصبی جایگزین می کند. به جای نگاشت یک جفت حالت-عمل به یک مقدار q ، یک شبکه عصبی حالت های ورودی را به جفت (عمل، Q-Value) نگاشت می کند.

بروزرسانی وزن شبکه با استفاده از معادله بلمن

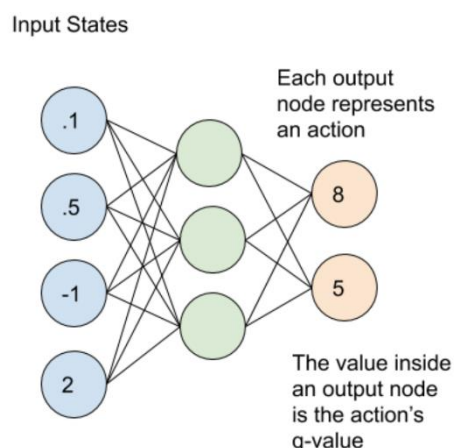
پس از انتخاب یک اقدام، زمان آن است که عامل عمل را انجام دهد و طبق معادله بلمن به روز کند. عوامل Deep Q-Learning از Experience Replay برای اطلاع از محیط خود و به روز رسانی شبکه های اصلی و هدف استفاده می کنند.

به طور خلاصه، شبکه اصلی هر ۴ مرحله نمونه برداری می کند و مجموعه ای از تجربیات گذشته را آموزش می دهد. سپس وزن شبکه اصلی هر ۱۰۰ مرحله به وزن شبکه هدف کپی می شود.

نحوه نگاشت حالت ها به جفت (Action, Q-value)

شبکه های عصبی اصلی و هدف، حالت های ورودی را به یک جفت (عمل، q -value) ترسیم می کنند. در این مورد، هر گره خروجی (نماینده یک عمل) حاوی مقدار q عمل به عنوان یک عدد ممیز شناور است. توجه داشته

باشید که گره‌های خروجی توزیع احتمال را نشان نمی‌دهند، بنابراین به عدد ۱ نخواهند رسید. برای مثال بالا، یک عمل دارای 8 Q-value و عمل دیگر دارای 5 Q-value است.



دلیل استفاده از DQN

Deep Q-Learning از Experience Replay برای یادگیری در دسته‌های کوچک استفاده می‌کند تا از انحراف توزیع داده‌ها در حالت‌های مختلف، اقدامات، پاداش‌ها و حالت‌های بعدی که شبکه عصبی خواهد دید جلوگیری کند. نکته مهم این است که نماینده بعد از هر مرحله نیازی به تمرین ندارد.

A2C

این الگوریتم یک نوع قطعی و همگام (Async) از الگوریتم A3C می‌باشد که از replay buffer استفاده نمی‌کند.

A3C

الگوریتم Asynchronous Advantage Actor Critic (A3C) یکی از جدیدترین الگوریتم‌هایی است که در زمینه الگوریتم‌های یادگیری تقویتی عمیق توسعه یافته است. این الگوریتم توسط DeepMind گوگل که بخش هوش مصنوعی گوگل است توسعه یافته است. این الگوریتم برای اولین بار در سال ۲۰۱۶ در یک مقاله تحقیقاتی ذکر شد.

این یک الگوریتم با سیاست گرادیان در یادگیری تقویتی است که سیاست

$$\pi(a_t | s_t; \theta)$$

و تخمینی از تابع مقدار

$$V(s_t; \theta_v)$$

را حفظ می کند. در نمای رو به جلو عمل می کند و از ترکیبی از بازگشت های n مرحله ای برای به روز رسانی سیاست و تابع مقدار استفاده می کند. سیاست و تابع مقدار پس از هر اقدام t -max یا زمانی که به حالت ترمینال می رسد به روز می شود. بروز رسانی انجام شده توسط الگوریتم زیر صورت می گیرد

$$\nabla_{\theta} \log \pi(a_t | s_t; \theta') A(s_t, a_t; \theta, \theta_v)$$

که

$$A(s_t, a_t; \theta, \theta_v)$$

تخمینی از تابع مزیت است که توسط

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$$

محاسبه می شود. که در آن k می تواند از حالتی به حالت دیگر متفاوت باشد و حد بالای t -max را دارد.

منتقدان، Critic، در A3C تابع مقدار را یاد می گیرند در حالی که چندین عامل به طور موازی آموزش می بینند و هر چند وقت یکبار با پارامترهای گلوبال همگام می شوند.

OpenAI Gym

Gym یک API استاندارد برای یادگیری تقویتی و مجموعه ای متنوع از محیط های مرجع است.

رابط Gym ساده، پایتونیک است و می تواند مشکلات عمومی RL را نشان دهد.

Box2D

این محیط ها همگی شامل بازی های اسباب بازی مبتنی بر کنترل فیزیک، با استفاده از فیزیک مبتنی بر box2d و رندر مبتنی بر PyGame هستند. این محیط ها در روزهای اولیه Gym توسط اولگ کلیموف ساخته شدند و از آن زمان به معیارهای محبوب اسباب بازی تبدیل شده اند. همه محیط ها از طریق آرگومان های مشخص شده در مستندات هر محیط بسیار قابل تنظیم هستند.

Lunar Lander

این محیط یک مسئله کلاسیک بهینه سازی مسیر موشک است. طبق اصل حداکثر پونتریاگین، بهینه است که موتور را با دریچه گاز کامل روشن کنید یا آن را خاموش کنید. به همین دلیل است که این محیط دارای اقدامات مجزا است: موتور روشن یا خاموش.

دو نسخه محیطی وجود دارد: گسسته یا پیوسته. سکوی فرود همیشه در مختصات (۰,۰) قرار دارد. مختصات دو عدد اول در بردار حالت هستند. فرود خارج از سکوی فرود امکان پذیر است. سوخت بی نهایت است، بنابراین یک مامور می تواند پرواز را یاد بگیرد و سپس در اولین تلاش خود فرود بیاید.

فضای اکشن

چهار عمل مجزا در دسترس است: هیچ کاری نکنید، موتور جهت چپ را آتش بزنید، موتور اصلی آتش، موتور جهت راست را آتش بزنید.

فضای رصد

۸ حالت وجود دارد: مختصات فرودگر در x & y ، سرعت های خطی آن در x & y ، زاویه آن، سرعت زاویه ای آن، و دو بولی که نشان می دهد هر پا در تماس با زمین است یا خیر.

پاداش

پاداش حرکت از بالای صفحه به سکوی فرود و استراحت حدود ۱۰۰-۱۴۰ امتیاز است. اگر فرودگر از سکوی فرود دور شود، پاداش را از دست می دهد. اگر فرودگر سقوط کند، ۱۰۰- امتیاز اضافی دریافت می کند. اگر به هدف بیاید، ۱۰۰+ امتیاز اضافی دریافت می کند. هر پا با تماس با زمین ۱۰+ امتیاز است. شلیک موتور اصلی ۰.۳- امتیاز در هر فریم است. شلیک موتور جانبی ۰.۰۳- امتیاز در هر فریم است. حل شده ۲۰۰ امتیاز است.

حالت شروع

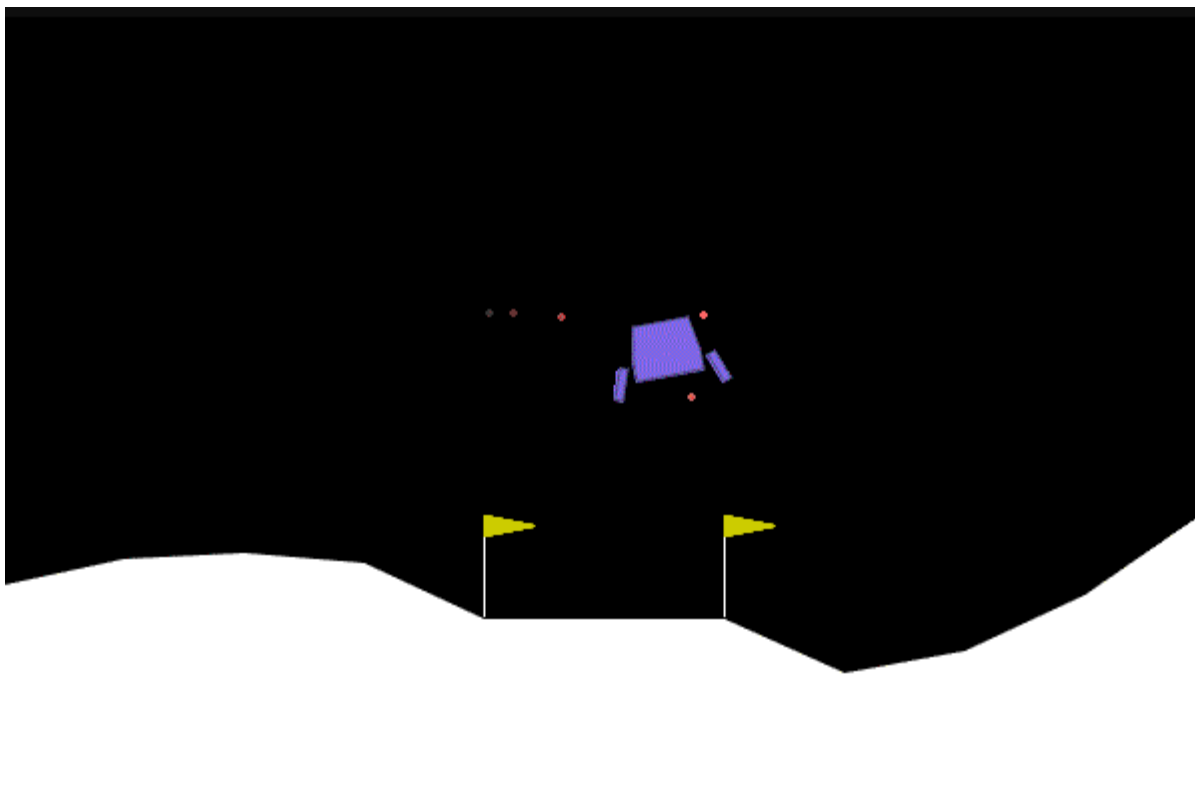
فرودگر با یک نیروی اولیه تصادفی که به مرکز جرم آن وارد می شود، از مرکز بالای نمای دید شروع می شود.

پایان /پیزود

اپیزود تمام می شود اگر:

فرودگر سقوط می کند (جسد فرودگر با ماه در تماس است).

فرودگر خارج از نما می شود (مختصات x بزرگتر از ۱ است).



پیاده سازی برنامه

برای پیاده سازی این پروژه از محیط توسعه Anaconda و Jupyter NoteBook استفاده شده است. بعد از نصب Anaconda کتابخانه‌های زیر نصب شده که به ترتیب وظایف زیر را بر عهده دارند.

- **Io** : این ماژول امکان کار با انواع مختلف I/O را برای پایتون فراهم می کند.
- **Os** : این ماژول یک روش پرتابل برای استفاده از عملکردهای وابسته به سیستم عامل را ارائه می دهد.
- **Glob** : این ماژول تمام نام مسیرهایی را که یک الگوی مشخص دارند پیدا می کند.
- **Torch** : این ماژول برای محاسبه محاسبات تنسور مانند **numpy** استفاده می شود.
- **Stable_baselines3** : مجموعه ای از پیاده سازی الگوریتم‌های یادگیری تقویتی است.
- **Numpy** : این ماژول برای کار با آرایه‌ها در پایتون است.
- **Matplot** : یک کتابخانه جامع برای ایجاد تجسم‌های ثابت، متحرک و تعاملی در پایتون است.

- Gym : یک کتابخانه منبع باز برای توسعه و مقایسه الگوریتم‌های یادگیری تقویتی با ارائه یک API استاندارد برای برقراری ارتباط بین الگوریتم‌های یادگیری و محیط‌ها است.

سپس یک تابع برای نمایش ویدیو در محیط برنامه نویسی تعریف شده.

```
# @title Plotting/Video functions
from IPython.display import HTML
from pyvirtualdisplay import Display
from IPython import display as ipythondisplay

display = Display(visible=0, size=(1400, 900))
display.start()

"""
Utility functions to enable video recording of gym environment
and displaying it.
To enable video, just do "env = wrap_env(env)"
"""

def show_video():
    mp4list = glob.glob('video/*.mp4')
    if len(mp4list) > 0:
        mp4 = mp4list[0]
        video = io.open(mp4, 'r+b').read()
        encoded = base64.b64encode(video)
        ipythondisplay.display(HTML(data='''<video alt="test" autoplay
loop controls style="height: 400px;"
<source src="data:video/mp4;base64,{0}" type="video/mp4" />
</video>'''.format(encoded.decode('ascii')))))
    else:
        print("Could not find video")

def wrap_env(env):
    env = Monitor(env, './video', force=True)
    return env
```

سپس تعداد نوروها در هر لایه را در یک لیست مشخص کرده‌ایم. برای شبکه عمیق Q خود ۳ لایه در نظر گرفته شده. سپس برای شبکه ضریب یادگیری نیز تعریف می‌کنیم تا بروزرسانی وزن‌ها با ضریب دلخواه انجام شود.

تعداد لایه‌ها و گره‌های هر لایه و همچنین نرخ یادگیری بعد از تعدادی بسیار زیادی آزمون و خطا بدست آمده است. در ابتدا ۲ لایه با تعداد گره ۶۴ در هر لایه با نرخ یادگیری ۰.۰۰۱ در نظر گرفته شده بود اما برای دقت بیشتر نیاز به افزایش لایه‌ها بود و با افزایش لایه‌ها نرخ یادگیری نیز نیاز به کاهش داشت و در نتیجه نصف شد تا جواب بهینه تری مدل بدست آورد.

```
nn_layers = [64,64,64] #This is the configuration of your neural network. Currently, we have three layers with 64 neurons each
learning_rate = 0.0005 #This is the step-size with which the gradient descent is carried out.
```

سپس یک دایرکتوری برای نگهداری Log های عامل و محیط ایجاد می‌کنیم که اطلاعات مورد نظر را در آن ذخیره کنیم تا از این داده‌ها برای ارزیابی کارایی سیستم استفاده کنیم.

سپس محیط خود را ایجاد می‌کنیم.

سپس با استفاده از تابع Monitor از کتابخانه `sable_baselines3` می‌توانیم این محیط‌های Gym را بررسی کنیم و مقادیر مورد نیاز از قبیل Reward و زمان هر Episode ویا اطلاعات دیگر را بدست آوریم.

سپس یک دیکشنری برای مشخص کردن activation function و تعداد لایه‌ها و نورون‌ها استفاده شده. تعداد لایه‌ها و نورون‌ها بالاتر در لیستی تعریفی شده بودند.

در مرحله بعد مدل DQN خود را با استفاده از کتابخانه `table_baselines3` تعریف کردیم. در این مرحله ابتدا سیاست مورد استفاده در مدل، `MLpPolicy` که همان سیاست DQN است را مشخص کرده‌ایم. سپس محیط خود را برای مدل و آرگومان‌هایی که بالاتر تعریف شده‌اند را مشخص کرده‌ایم. سپس نرخ یادگیری و باقی اطلاعات مورد نیاز از قبیل اندازه batch یا نمونه خود و گاما که در فرمول بلمن استفاده می‌شود تا وزن‌های شبکه آپدیت بشوند را برای مدل مشخص کرده‌ایم.

```
log_dir = "/tmp/gym/"
os.makedirs(log_dir, exist_ok=True)

# Create environment
env = gym.make('LunarLander-v2')

env = stable_baselines3.common.monitor.Monitor(env, log_dir)

callback = EvalCallback(env, log_path = log_dir, deterministic=True) #For evaluating the performance of the agent periodically and

policy_kwargs = dict(activation_fn=torch.nn.ReLU, net_arch=nn_layers)

model = DQN("MlpPolicy", env, policy_kwargs = policy_kwargs,
            learning_rate=learning_rate,
            batch_size=1, #for simplicity, we are not doing batch update.
            buffer_size=1, #size of experience of replay buffer. Set to 1 as batch update is not done
            learning_starts=1, #Learning starts immediately!
            gamma=0.99, #discount facto. range is between 0 and 1.
            tau = 1, #the soft update coefficient for updating the target network
            target_update_interval=1, #update the target network immediately.
            train_freq=(1,"step"), #train the network at every step.
            max_grad_norm = 10, #the maximum value for the gradient clipping
            exploration_initial_eps = 1, #initial value of random action probability
            exploration_fraction = 0.5, #fraction of entire training period over which the exploration rate is reduced
            gradient_steps = 1, #number of gradient steps
            seed = 1, #seed for the pseudo random generators
            verbose=0) #Set verbose to 1 to observe training logs.
```

سپس در ابتدا برای تست مدل یک نمونه از اجرای محیط و انتخاب‌های مدل را شبیه سازی کرده‌ایم.

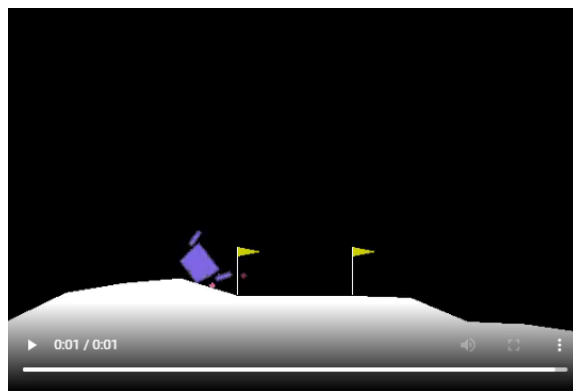
ابتدا در یکی حلقه با استفاده از تابع `render()` خروجی محیط را نمایش می‌دهیم که مشاهده می‌شود حالت آغازین ما به چه نحوی است. سپس با استفاده از تابع `predict` مدل تعریف شده یک عمل و یک حالت انتخاب می‌شود.

سپس با تابع `step()` محیط یک حرکت، عملی که توسط مدل انتخاب شده، انجام می‌دهیم. سپس مقدار امتیاز را به مقادیر قبلی اضافه می‌کنیم. در نهایت اگر عامل به حالت پایانی خود رسیده بود، از حلقه خارج می‌شویم.

```
test_env = wrap_env(gym.make("LunarLander-v2"))
observation = test_env.reset()
total_reward = 0
while True:
    test_env.render()
    action, states = model.predict(observation, deterministic=True)
    observation, reward, done, info = test_env.step(action)
    total_reward += reward
    if done:
        break;

# print(total_reward)
test_env.close()
show_video()
```

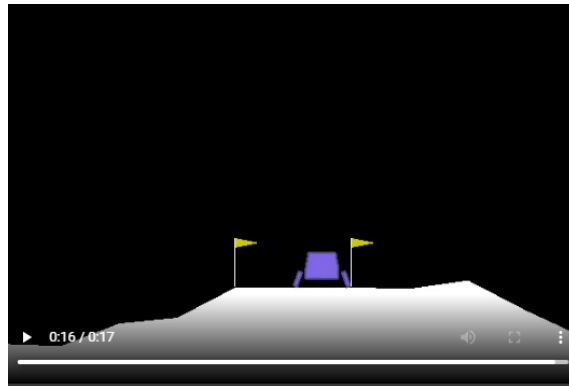
در این مرحله می‌توان مشاهده کرد که عامل به درستی حرکتهای خود را انتخاب نمی‌کند و به درستی فرود نمی‌آید.



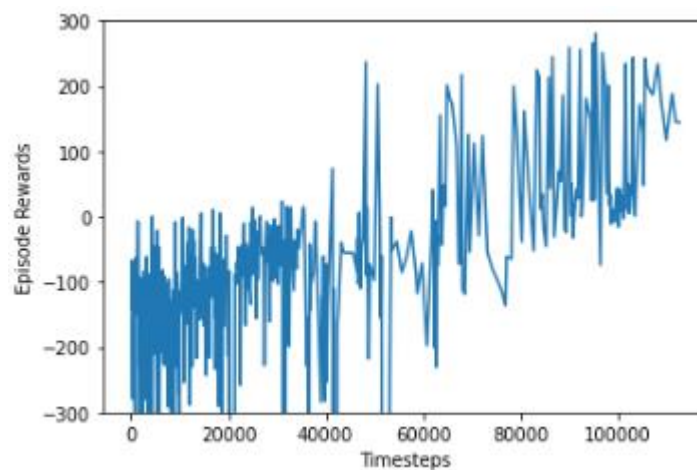
به همین دلیل مدل باید تعلیم داده شود. با استفاده از تابع `learn` در مدل این کار صورت می‌گیرد. همچنین در این تابع می‌توان تعداد مرحله‌ها و تعداد دفعاتی که خروجی فرآیند یادگیری نمایش داده شود را مشخص کنیم.

```
model.learn(total_timesteps=100000, log_interval=10, callback=callback)
# The performance of the training will be printed every 10 episodes.
```

بعد از پایان فرآیند یادگیری همانند قبل عامل می‌تواند در محیط با استفاده از مدل جدید حرکت کند. این بار مشاهده می‌شود که عامل با دقت بیشتری حرکت کرده و در محل مورد نظر فرود می‌آید.



سپس با استفاده از کتابخانه `matplotlib` داده‌های `reward` و `timesteps` را بر روی نمودار نمایش می‌دهیم. همانطور که مشاهده می‌شود مدل با گذشت مراحل بیشتر میانگین امتیاز بیشتری بدست آورده که بیانگر آن است که مدل رفته رفته بهتر شده است و انتخاب‌های خود را برای افزایش امتیاز انتخاب کرده است.



در ادامه همین فرآیند برای الگوریتم `A2C` نیز تکرار شده.

ابتدا تعداد نورون‌ها در هر لایه و ضریب یادگیری نیز تعریف می‌کنیم تا بروزرسانی وزن‌ها با ضریب دلخواه انجام شود.

تعداد لایه‌ها و گره‌های هر لایه و همچنین نرخ یادگیری با مدل `DQN` یکسان در نظر گرفته شده تا مقایسه به درستی صورت گیرد.

سپس یک دایرکتوری برای نگهداری Log های عامل و محیط ایجاد شده که اطلاعات مورد نظر را در آن ذخیره شده تا از این داده‌ها برای ارزیابی کارایی سیستم استفاده شود.

سپس محیط خود را ایجاد شده که با محیط قبلی یکسان در نظر گرفته شده است.

در مرحله بعد مدل با استفاده از کتابخانه table_baselines3 تعریف شده و محیط را برای مدل و آرگومان‌هایی که تعریف شده‌اند را مشخص کرده‌ایم. سپس نرخ یادگیری و باقی اطلاعات مورد نیاز را مشخص کرده‌ایم.

```
#This is the configuration of your neural network. Currently, we have 3 layers with 64 neurons each
nn_layers_a2c = [64,64,64]
#This is the step-size with which the gradient descent is carried out.
learning_rate_a2c = 0.0005

log_dir_a2c = "/tmp/gym/a2c"
os.makedirs(log_dir_a2c, exist_ok=True)

# Create environment
# We use same environment as the dqn
env_a2c = env

env_a2c = stable_baselines3.common.monitor.Monitor(env_a2c, log_dir_a2c )

#For evaluating the performance of the agent periodically and logging the results.
callback_a2c = EvalCallback(env_a2c,log_path = log_dir_a2c, deterministic=True)

policy_kwargs_a2c = dict(activation_fn=torch.nn.ReLU, net_arch=nn_layers_a2c)

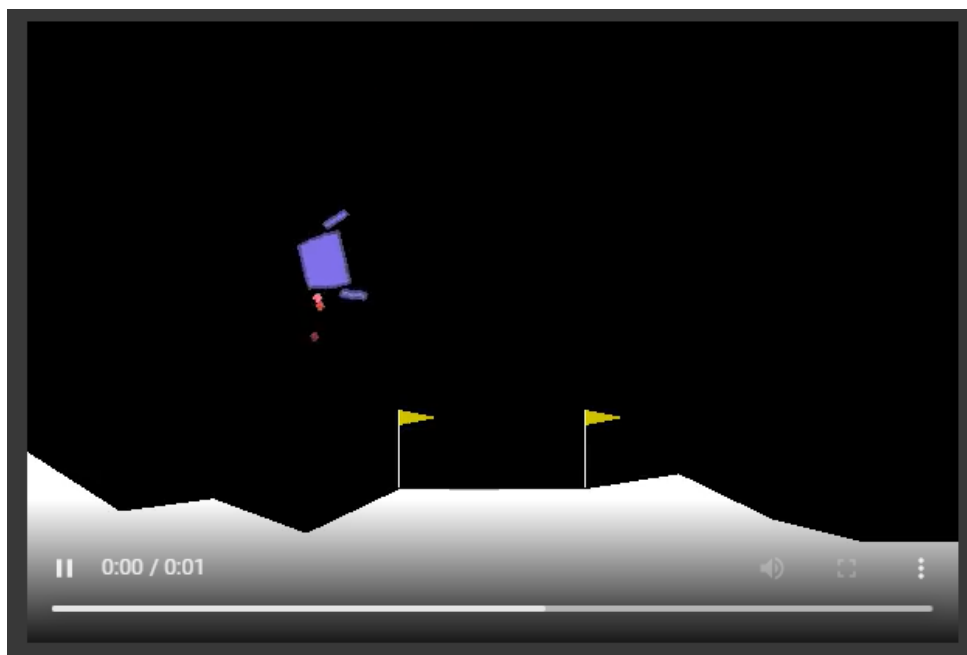
model_a2c = A2C("MlpPolicy", env,policy_kwargs = policy_kwargs_a2c,
               learning_rate=learning_rate_a2c,
               gamma=0.99, #discount facto. range is between 0 and 1.
               max_grad_norm = 10, #the maximum value for the gradient clipping
               seed = 1, #seed for the pseudo random generators
               verbose=0) #Set verbose to 1 to observe training logs.
```

سپس در ابتدا برای تست مدل یک نمونه از اجرای محیط و انتخاب‌های مدل را شبیه سازی کرده‌ایم.

```
test_env = wrap_env(gym.make("LunarLander-v2"))
observation = test_env.reset()
total_reward = 0
while True:
    test_env.render()
    action, states = model_a2c.predict(observation, deterministic=True)
    observation, reward, done, info = test_env.step(action)
    total_reward += reward
    if done:
        break;

# print(total_reward)
test_env.close()
show_video()
```

در این مرحله می‌توان مشاهده کرد که عامل به درستی حرکتهای خود را انتخاب نمی‌کند و به درستی فرود نمی‌آید.



به همین دلیل مدل باید تعلیم داده شود. با استفاده از تابع `learn` در مدل این کار صورت می‌گیرد. همچنین در این تابع می‌توان تعداد مرحله‌ها و تعداد دفعاتی که خروجی فرآیند یادگیری نمایش داده شود را مشخص کنیم.

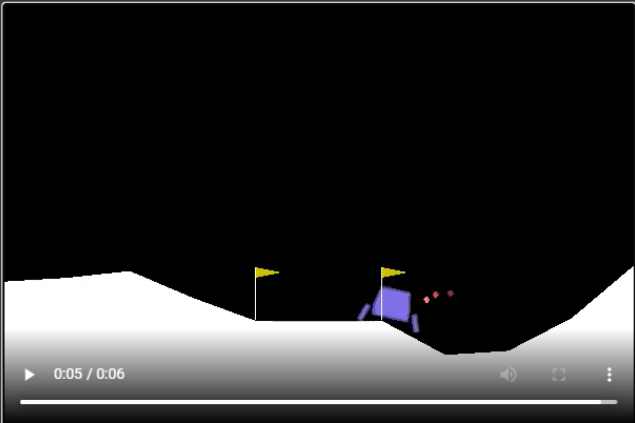
```
model_a2c.learn(total_timesteps=200000, log_interval=10, callback=callback_a2c)
```

```
Eval num_timesteps=10000, episode_reward=-785.24 +/- 351.29
Episode length: 117.80 +/- 33.77
New best mean reward!
Eval num_timesteps=20000, episode_reward=-638.58 +/- 144.66
Episode length: 105.00 +/- 12.41
New best mean reward!
Eval num_timesteps=30000, episode_reward=-218.44 +/- 28.01
Episode length: 194.00 +/- 65.25
New best mean reward!
Eval num_timesteps=40000, episode_reward=-304.44 +/- 95.90
Episode length: 117.60 +/- 17.50
Eval num_timesteps=50000, episode_reward=-91.24 +/- 37.93
Episode length: 883.60 +/- 99.49
New best mean reward!
Eval num_timesteps=60000, episode_reward=-286.22 +/- 90.84
Episode length: 1000.00 +/- 0.00
Eval num_timesteps=70000, episode_reward=-3.88 +/- 107.48
Episode length: 899.40 +/- 125.84
New best mean reward!
Eval num_timesteps=80000, episode_reward=-58.05 +/- 124.82
```

بعد از پایان فرآیند یادگیری همانند قبل عامل می‌تواند در محیط با استفاده از مدل جدید حرکت کند.

```
env_a2c = wrap_env(gym.make("LunarLander-v2"))
observation = env_a2c.reset()
while True:
    env_a2c.render()
    action, _states = model_a2c.predict(observation, deterministic=True)
    observation, reward, done, info = env_a2c.step(action)
    if done:
        break;

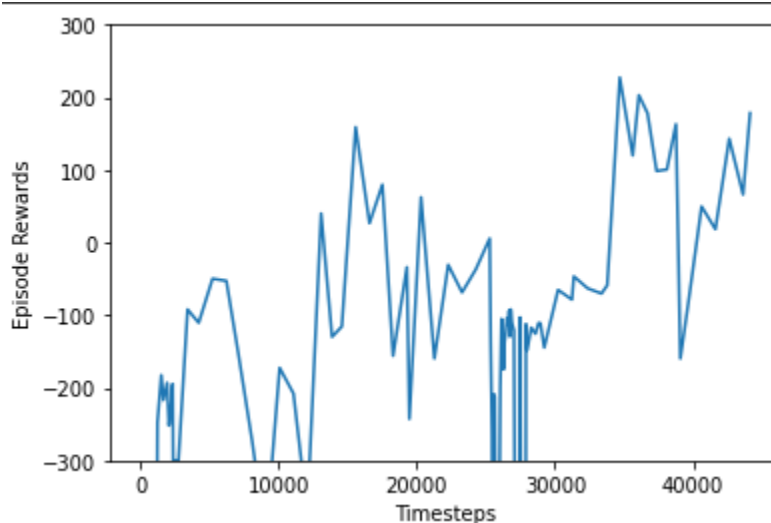
env_a2c.close()
show_video()
```



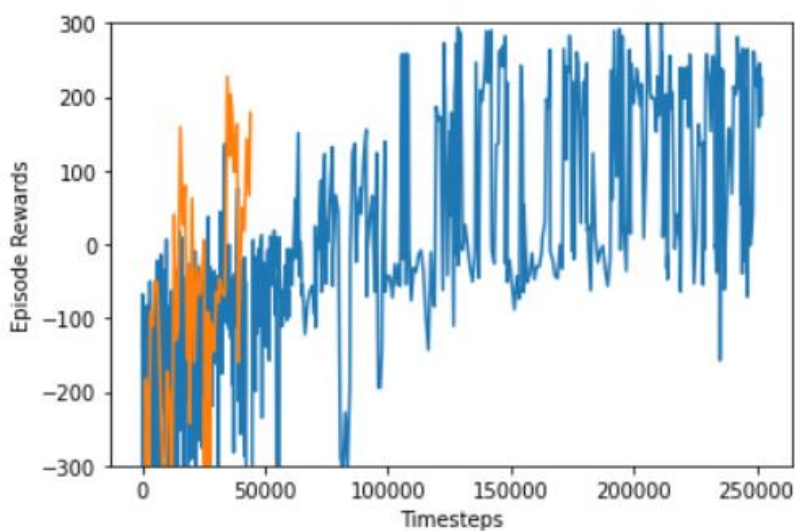
سپس با استفاده از کتابخانه matplotlib داده‌های reward و timesteps را بر روی نمودار نمایش می‌دهیم.

همانطور که مشاهده می‌شود مدل با گذشت مراحل بیشتر میانگین امتیاز بیشتری بدست آورده که بیانگر آن است که مدل رفته رفته بهتر شده است و انتخاب‌های خود را برای افزایش امتیاز انتخاب کرده است.

```
▶ x_a2c, y_a2c = ts2xy(load_results(log_dir_a2c), 'timesteps') # Organising the logged results in to a clean format for plotting.  
plt.plot(x_a2c, y_a2c)  
plt.ylim([-300, 300])  
plt.xlabel('Timesteps')  
plt.ylabel('Episode Rewards')
```



در نهایت داده‌های دو مدل را روی یک نمودار نمایش می‌دهیم تا مقایسه دو روش به راحتی صورت گیرد.



نتیجه

با توجه به نتایج می‌توان پی برد که می‌توان با استفاده از شبکه‌های عصبی عمیق Q مدل‌هایی هوشمند ساخت تا در محیط‌های شبیه سازی شده وظیفه فرود یک سازه را بر عهده گیرند. همچنین در طی این پروژه یک محیط بسیار ساده توسط یک مدل ساده با پیچیدگی بسیار کمی بررسی شد که طبیعتاً در دنیای واقعی این سادگی دور از انتظار است.

در مقایسه با الگوریتم A2C الگوریتم DQN، در تعداد مرحله‌های ثابت بهتر و سریع‌تر به یک رفتار بهینه نزدیک می‌شود. با توجه به این موضوع می‌توان نتیجه گرفت که در این سناریو خاص الگوریتم DQN به طور کلی گزینه برتری نسبت به الگوریتم A2C است.

در دنیای واقعی علاوه بر پیچیدگی بسیار زیاد مدل‌ها، محیط‌ها نیز بسیار متغیر و غیرقابل پیش بینی می‌باشد. به همین دلیل همواره تیم ناظری وجود دارد تا رفتار مدل را زیر نظر گرفته و در صورت خطا بلافاصله کنترل را بدست بگیرند.

در نتیجه می‌توان گفت که هرچند مسئله در طول این پروژه حل شده اما در واقعیت هنوز این مسئله به وضوح پایدار است.

References

1. [Stable Baselines Framework](#)
2. [Lunar Lander Environment](#)
3. [OpenAI gym environments](#)
4. [A good reference for introduction to RL](#)
5. [A3C](#)

با تشکر

پارسا محمودی کشتی‌بان

دانشجوی مهندسی کامپیوتر دانشگاه ارومیه