

دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق - کروه مهندسی کنترل

# درس یادگیری ماشین

## مینی پروژه سوم

نام و نام خانوادگی	علیرضا رضایی
شماره دانشجویی	۴۰۲۰۶۱۰۴
تاریخ	۱۴۰۳ تیر
استاد درس	دکتر علیاری



## فهرست مطالب

۴	۱	لینک های مربوطه
۴	۱.۱	لینک مربوط به گیت هاب
۴	۲.۱	لینک مربوط به گوگل کلب
۵	۲	سوال دوم
۵	۱.۲	قسمت اول
۶	۲.۲	قسمت دوم
۱۹	۳.۲	قسمت سوم



## فهرست تصاویر

۵	.....	Lunar Lander محیط	۱
۷	.....	import Lunar Lander from Gym محیط	۲
۱۶	.....	نمودار پاداش تجمعی برای batch size = 32	۳
۱۶	.....	نمودار پاداش تجمعی برای batch size = 64	۴
۱۷	.....	نمودار پاداش تجمعی برای batch size = 128	۵
۱۷	.....	پاداش تجمعی برای batch size = 32	۶
۱۸	.....	پاداش تجمعی برای batch size = 64	۷
۱۸	.....	پاداش تجمعی برای batch size = 128	۸
۱۹	.....	ویدیویی از ایجنت در محیط به ازای ۲۵۰ batch size = 128 در اپیزود	۹
۱۹	.....	ساختن ویدیو از ایجنت در محیط به ازای ۲۵۰ batch size = 128 در اپیزود	۱۰
۲۰	.....	نمودار پاداش تجمعی به کمک DDQN در حالت batch size = 128	۱۱
۲۰	.....	ویدیویی از ایجنت در محیط به ازای ۲۵۰ batch size = 128 در اپیزود	۱۲



## فهرست برنامه‌ها

۶	.....	swig installing	۱
۷	.....	Xvbf installing	۲
۷	.....	GPU using	۳
۸	.....	display virtual a initializing	۴
۹	.....	replay Experience	۵
۱۱	.....	class DeepQNetwork	۶
۱۲	.....	action take	۷
۱۳	.....	params update	۸
۱۴	.....	weights the load and save	۹
۱۴	.....	۳۲ size batch	۱۰
۱۴	.....	agent initialize	۱۱
۱۵	.....	loop train	۱۲
۱۵	.....	loop episode	۱۳
۱۵	.....	Decay Epsilon	۱۴
۱۹	.....	Agent DQN	۱۵



## ۱ لینک های مربوطه

### ۱.۱ لینک مربوط به گیت هاب

از این لینک گیت هاب ([Github](#)) می توانید برای دسترسی به صفحه Github مربوط به این پروژه استفاده کنید.

### ۲.۱ لینک مربوط به گوگل کلب

از این لینک گوگل کلب ([Google Colab](#)) می توانید برای دسترسی به notebook نوشته شده دسترسی پیدا کنید.

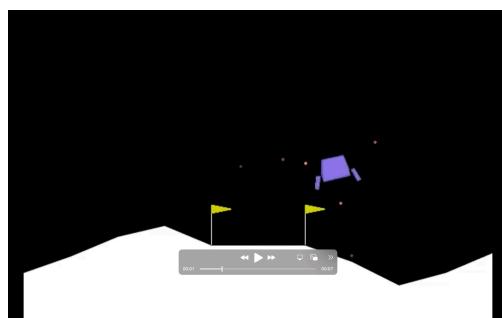


## ۲ سوال دوم

### ۱.۲ قسمت اول

محیط Open AI Lunar Lander در قالب یک کتابخانه متن باز به نام Gym توسعه یافته است. این محیط برای شبیه‌سازی‌ها و اهداف آموزشی در زمینه Reinforcement Learning به کار می‌رود. این کتابخانه شامل محیط‌های متعددی است که هر یک جنبه‌ای از RL را پوشش می‌دهند؛ از مسائل کنترل کلاسیک مانند Cart Pole تا بازی‌های آتاری و شبیه‌سازی‌های پیچیده‌تر مانند MuJoCo.

یکی از ویژگی‌های مثبت این منبع متن باز، یکپارچگی آن در تمامی محیط‌ها است، به گونه‌ای که تمامی محیط‌ها و نتایج مرتبط با آن‌ها قابل مقایسه هستند. متدهایی نظیر reset() و close() از جمله این متدها می‌باشند. اکنون که با فضای کلی، توسعه‌دهنده و اهداف این کتابخانه متن باز آشنا شدیم، به سراغ یکی از محیط‌های آن یعنی Lunar Lander می‌رویم. **شکل ۱** تصویری از این محیط را نشان می‌دهد که در آن یک فضای پیما قصد دارد روی یک سطح مشخص فرود بیاید. در واقع این مسئله بهینه‌سازی مسیر راکت یا فضای پیما است که یک مسئله کلاسیک به شمار می‌رود. بر اساس اصل Pontryagin's maximum، حالت بهینه زمانی است که موتور در حداقل توان خود روشن یا خاموش باشد. به همین دلیل، هر موتور دو حالت دارد؛ یا روشن است یا خاموش.



شکل ۱: محیط Lunar Lander

در این بازی شامل چهار اقدام گسسته است:

- عدم انجام کار
- روشن کردن موتور جهت چپ
- روشن کردن موتور اصلی
- روشن کردن موتور جهت راست

در این بازی شامل یک بردار ۸ بعدی است:

- موقعیت X که می‌تواند بین ۱.۵ تا ۱.۵ باشد.
- موقعیت Y که می‌تواند بین ۱.۵ تا ۱.۵ باشد.



- سرعت خطی در جهت X که می‌تواند بین ۵–تا ۵ باشد.
- سرعت خطی در جهت Y که می‌تواند بین ۵–تا ۵ باشد.
- زاویه که می‌تواند بین ۳.۱۴–تا ۳.۱۴ (به رادیان) باشد.
- سرعت زاویه‌ای که می‌تواند بین ۵–تا ۵ باشد.
- یک متغیر بولین که نشان می‌دهد پای ۱ با زمین تماس دارد یا نه.
- یک متغیر بولین که نشان می‌دهد پای ۲ با زمین تماس دارد یا نه.

Reward System به صورتی طراحی شده که بازیگر یا فضایپیما را ترغیب به فرود دقیق و نرم در محل موردنظر می‌کند. پاداش برای حرکت از بالا و فرود آمدن روی سکوی مشخص شده بین ۱۰۰ تا ۱۴۰ امتیاز است. اگر فضایپیما از سکوی هدف فاصله بگیرد، پاداش از دست می‌دهد. اگر تصادف کند، ۱۰۰–امتیاز از دست می‌دهد. اگر روی سکو نشسته و بی‌حرکت بماند یا reset انجام دهد، ۱۰۰ امتیاز دریافت می‌کند. هر پا که با زمین برخورد کند، ۱۰ امتیاز می‌گیرد. روشن شدن موتور اصلی ۰.۰۳–امتیاز در هر فریم را در بر دارد و برای موتورهای کناری نیز همین مقدار است. این کار به منظور تشویق به استفاده کمتر از سوخت انجام می‌شود. رسیدن به حالت solved دارای ۲۰۰ امتیاز است. موقعیت ابتدایی فضایپیما در مرکز و بالای تصویر قرار دارد و در ابتدا یک نیروی تصادفی به مرکز جرم آن اعمال می‌شود.

هر اپیزود بر اساس یکی از دلایل زیر به پایان می‌رسد:

- فضایپیما تصادف کند، یعنی بدن آن با ماه برخورد کند.
- فضایپیما از فضای دید خارج شود، مثلاً حرکت افقی آن بیشتر از ۱ شود.
- اگر فضایپیما بیدار نباشد، یعنی حرکتی نداشته و با چیز دیگری برخورد نکند.

## ۲.۲ قسمت دوم

برای حل این قسمت از کد قرار گرفته به عنوان نمونه استفاده شده و تغییراتی در آن اعمال شده تا ارورها برطرف شوند. در مرحله اول باید موارد و کتابخانه‌های لازم را نصب کنیم. دو نکته وجود دارد که قابل اشاره هستند. اول اینکه باید دو دستور برای نگرفتن ارور به کد اضافه کنیم که در برنامه ۱ آمده‌اند.

```

1 !apt-get update
2 !apt-get install -y swig

```

Code 1: installing swig

دوم اینکه برای نشان دادن ویدیوها به صورت مجازی باید از Xvbf استفاده کنیم که نیاز است آن را در محیط google collab نصب کنیم که به وسیله برنامه ۲ این کار انجام شده است.



```
1 !sudo apt-get update
2 !sudo apt-get install xvfb
```

Code 2: installing Xvbf

```
1 import torch
2 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
3 device
```

Code 3: using GPU

سایر دستورات در بخش نصب ثابت باقی مانده است و به سراغ مرحله بعد می‌رویم.  
در مرحله بعد دستور مورد نیاز برای استفاده از GPU در محیط google collab را به صورت برنامه<sup>۳</sup> وارد می‌کنیم که نتیجه آن به ما می‌گوید در حال استفاده از Coda و ران کردن GPU هستیم.  
در ادامه به صورت شکل<sup>۲</sup> محیط مورد نظر که همان Lunar Lander است را از Gym وارد می‌کنیم. همانطور که مشاهده می‌شود، ما state داریم و می‌توانیم action<sup>۴</sup> را از observation<sup>۸</sup> یا داریم توانیم که همگی در بخش<sup>۱.۲</sup> توضیح داده شدند.

```
# We should import gym library here
import gym

# Initialize Lunar Lander environment with RGB rendering mode
env = gym.make('LunarLander-v2', render_mode="rgb_array")

# Find observation size (state size)
state_size = env.observation_space.shape[0]

# Find action size, we can see it blew:
# 0: Do nothing
# 1: Fire left engine
# 2: Fire down engine
# 3: Fire right engine
action_size = env.action_space.n

state_size, action_size
```

شکل ۲: محیط Lunar Lander from Gym

در مرحله بعد ابتدا یک صفحه نمایش مجازی تعریف می‌کنیم و سایز آن را مشخص می‌کنیم و سپس یک تابع تعریف می‌کنیم تا بتوانیم ویدیوهایی که در آینده قصد ضبط آنها را داریم در محیط Jupiter notebook ببینیم. انجام این کار در برنامه<sup>۴</sup> آمده است.

در مرحله بعد برنامه<sup>۵</sup> را داریم. در ابتدای آن یک تاپل می‌سازیم که حالت یک experience را در خود ذخیره کند. این موارد شامل حالتی که فضای پیما در آن وجود دارد (قبل از انجام اکشن)، اکشنی که انجام می‌دهد، حالت بعدی که با انجام اکشن به آن می‌رود، پاداشی که دریافت می‌کند و یک متغیر بولین که نشان می‌دهد آیا عمل تمام شده یا نه. سپس کلاس ExperienceReplay یک reply buffer را پیاده می‌کند. در واقع تمامی transitions را در خود ذخیره می‌کند تا در مرحله آموزش از آنها استفاده شود. برای این کار از یک deque یا double-ended queue استفاده می‌شود تا memory ساخته شود. reply buffer به agent اجازه می‌دهد تا از تجربیات گذشته خود با نمونه‌برداری تصادفی از transitions یاد بگیرد. این تصادفی‌سازی به پایداری یادگیری کمک کرده و مشکل correlation بین store trans و correlation را بهبود می‌دهد. reply buffer جدید را به اضافه می‌کند.



```

1 !pip install pyvirtualdisplay # Install the missing module
2 from pyvirtualdisplay import Display
3 import glob
4 import io
5 import base64
6 from IPython.display import HTML, display as ipythondisplay
7
8 # Initialize virtual display
9 display = Display(visible=0, size=(1400, 900))
10 display.start()
11
12 """
13 Utility functions to enable video recording of gym environment
14 and displaying it.
15 To enable video, just do "env = wrap_env(env)"
16 """
17
18 def show_video():
19     mp4list = glob.glob('video/*.mp4')
20     if len(mp4list) > 0:
21         mp4 = mp4list[0]
22         video = io.open(mp4, 'r+b').read()
23         encoded = base64.b64encode(video)
24         ipythondisplay(HTML(data=f'''<video alt="test" autoplay loop controls style="height: 400
25             px;">
26                 <source src="data:video/mp4;base64,{encoded.decode('ascii')}" type="video/mp4"
27             />
28         </video>'''))
29     else:
30         print("Could not find video")

```

Code 4: initializing a virtual display



```
1 import random
2 from collections import namedtuple, deque
3
4 # Define the transition tuple
5 Transition = namedtuple('Transition', ('state', 'action', 'next_state', 'reward', 'done'))
6
7 class ExperienceReplay:
8     def __init__(self, capacity):
9         self.memory = deque(maxlen=capacity)
10
11     def store_transition(self, state, action, next_state, reward, done):
12         """Store a transition in the replay buffer."""
13         transition = Transition(state, action, next_state, reward, done)
14         self.memory.append(transition)
15
16     def sample(self, batch_size):
17         """Sample a batch of transitions from the replay buffer."""
18         return random.sample(self.memory, batch_size)
19
20     def __len__(self):
21         """Return the current size of internal memory."""
22         return len(self.memory)
```

Code 5: Experience replay

در مرحله بعد شبکه عصبی عمیق مربوط به الگوریتم DQN را تشکیل می‌دهیم. [برنامه ۶](#) این شبکه را نشان می‌دهد که در کلاس DeepQNetwork تعریف شده است. همانطور که مشخص است این یک شبکه sequential است که نسبت به کد اصلی به صورت مستقیم پیاده‌سازی شده و برگرددانده می‌شود. ورودی این شبکه به اندازه state‌ها نورون دارد و در نهایت خروجی این شبکه همان  $Q$ ‌ها هستند که مقادیر انتظاری از پاداش آینده را برای هر اکشن به ما می‌دهند. پس خروجی این شبکه به اندازه اکشن‌ها است و مقادیر  $Q$  برای آن‌ها تخمین زده می‌شود. در لایه اول یک لایه تمام‌اً متصل باتابع فعال‌ساز ReLU استفاده شده است. سپس از Layer normalization استفاده شده تا روند یادگیری را پایدار سازد و سرعت بیخشد، همچنین برای جلوگیری از overfitting از روش Dropout استفاده شده است و نرخ آن ۱۰ درصد است. لایه دوم نیز یک لایه خطی به صورت تمام‌اً متصل است و دوباره از نرمالیزه کردن و Dropout استفاده شده و تابع فعال‌ساز ReLU است. لایه سوم نیز به همین صورت. متند forward ورودی تنسور  $x$  را به شبکه می‌دهد و در خروجی  $Q$ ‌های مربوط به هر اکشن بازگرددانده می‌شود.

کلاس بعدی DQNAgent است که Agent را پیاده‌سازی می‌کند. اینجنتی که با محیط interact می‌کند،  $Q$  value را ذخیره می‌کند و  $Q$  value را به روز رسانی می‌کند. ابتدا نیاز است تا متغیرها را در کلاس initialize کنیم. این موارد شامل ابعاد state و action و یا batch size است. همچنین نیاز به تعریف gamma داریم که همان discount factor است.  $\tau$  پارامتر soft update برای شبکه هدف است. همچنین نیاز به experience replay و یک بهینه‌ساز و چند مورد دیگر نیز داریم.

متند بعدی در این کلاس take action است که در [برنامه ۷](#) آمده است. که بر اساس epsilon greedy policy از حالت فعلی یک اکشن انتخاب می‌کند. اگر یک عدد تصادفی از  $eps$  بزرگ‌تر باشد، اینجنت اکشن را انتخاب می‌کند که بیشترین  $Q$ -value که در شبکه اصلی پیش‌بینی شده دارد. در غیر این صورت اینجنت یک اکشن تصادفی انتخاب می‌کند و این تفاوت بین exploration و exploitation است. این روش مطمئن می‌شود که تعادل بین این دو روش در آموزش رعایت می‌شود.

متند بعدی update params است که در [برنامه ۸](#) آمده است. این متند پارامترهای شبکه اصلی را به کمک تجربه‌های گرفته شده از replay buffer به روز رسانی می‌کند و یک soft update sample(batch size) یک بچ از تجربه را از replay buffer بر می‌داریم و با فرمول بلمن مقدار  $Q$  value را حساب می‌کند. سپس اختلاف بین مقدار پیش‌بینی شده و اصلی را می‌یابد و یک گرادیان کاهشی برای کمینه‌سازی این خطای اجرا می‌کند.

در نهایت به کمک [برنامه ۹](#) این وزن‌ها را ذخیره و بارگذاری می‌کنیم. حالا باید برای سه batch مختلف، عمل آموزش را انجام دهیم. توجه شود که تمامی مراحل توضیح داده شده و همچنین آموزش برای سه حالت یکسان است و فقط batch size تغییر می‌کند. برای مثال برای بچ ۳۲، به صورت [برنامه ۱۰](#) خواهد بود.

حال به توضیح فرآیند آموزش می‌پردازیم. ابتدا به صورت [برنامه ۱۱](#) اینجنت را تعریف می‌کنیم و همچنین پاداش تجمعی را initialize می‌کنیم. در این قسمت ما یک اینجنت از DQNAgent با حالت و اکشن خاص درست می‌کنیم که بچ سایز آن را مشخص کردیم. همچنین صفحه‌ای ۲۵ تایی تشکیل می‌دهیم که در آینده از آنها میانگین می‌گیریم. در [برنامه ۱۲](#) حلقه آموزش را به تعداد مشخصی از اپیزود که در این حالت ۲۵۰ است تشکیل می‌دهیم. در این کد از هر ۵۰ امین اپیزود فیلم‌برداری می‌شود یعنی اپیزود ۵۰، ۱۰۰، ۱۵۰، ۲۰۰ و ۲۵۰ ام. سپس محیط را ریست می‌کند و مقادیر اولیه را می‌گیرد. سپس done را که قبل تر توضیح داده شده بود به حالت false در می‌آورد تا اپیزود بعدی استارت شود. سپس مقدار پاداش برای اپیزود فعلی را صفر می‌گذارد.



```

1 import torch.nn as nn
2 import torch.nn.functional as F
3
4 class DeepQNetwork(nn.Module):
5     def __init__(self, state_size, action_size):
6         """
7             Initialize the Deep Q-Network.
8
9         Args:
10            state_size (int): The size of the input state.
11            action_size (int): The size of the action space.
12        """
13        super(DeepQNetwork, self).__init__()
14
15        self.net = nn.Sequential(
16            nn.Linear(state_size, 512), # First hidden layer
17            nn.ReLU(), # Activation function
18            nn.LayerNorm(512), # Layer normalization
19            nn.Dropout(0.1), # Dropout for regularization
20            nn.Linear(512, 512), # Second hidden layer
21            nn.ReLU(), # Activation function
22            nn.LayerNorm(512), # Layer normalization
23            nn.Dropout(0.1), # Dropout for regularization
24            nn.Linear(512, 512), # Third hidden layer
25            nn.ReLU(), # Activation function
26            nn.Linear(512, action_size) # Output layer
27        )
28
29    def forward(self, x):
30        """
31            Perform a forward pass through the network.
32
33        Args:
34            x (torch.Tensor): The input state tensor.
35
36        Returns:
37            torch.Tensor: The output Q-values for each action.
38        """
39        return self.net(x)

```

Code 6: DeepQNetwork class



```

1   def take_action(self, state, eps=0.0):
2       """
3           Select an action using an epsilon-greedy policy.
4
5       Args:
6           state (array): Current state.
7           eps (float): Epsilon, for exploration-exploitation trade-off.
8
9       Returns:
10          int: Selected action.
11          """
12
13         self.value_net.eval()
14         if random.random() > eps:
15             with torch.no_grad():
16                 state_tensor = torch.tensor(state).float().unsqueeze(0).to(device)
17                 action = torch.argmax(self.value_net(state_tensor)).item()
18             else:
19                 action = np.random.randint(0, self.action_size)
20             self.value_net.train()
21         return action

```

Code 7: take action

بخش برنامه ۱۳ تا زمانی که اپیزود تمام شود با محیط ارتباط می‌گیرد، تجربه کسب می‌کند و  $Q$  value ها را به روز می‌کند. در واقع ابتدا یک اکشن انتخاب می‌شود، حالت بعدی، فلگ و پاداش را می‌گیرد و تجربه کسب شده را در  $Q$  value function ذخیره می‌کند. سپس  $Q$  را به روز رسانی می‌کند (به کمک نمونه‌گیری گرفته شده از reply buffer) سپس حالت را از حالت فعلی به حالت بعدی تغییر می‌دهد و پاداش را به پاداش‌های قبلی در آن اپیزود اضافه می‌کند.

بخش بعدی اپسیلون را کاهش می‌دهد و مقدار پاداش تجمعی در این اپیزود را ذخیره می‌کند. اگر اپیزود مضربی از ۵۰ بود آن را ذخیره می‌کند و اگر مضرب ۲۵ بود، پاداش میانگین آن اپیزود فعلی و مقدار اپسیلون را پرینت می‌کند. این بخش در برنامه ۱۴ آمده است.

در نهایت پاداش تجمعی برای batch size = 32 در شکل ۳ آمده است.

همین نمودار برای batch size = 64 در شکل ۴ آمده است. همانطور که مشاهده می‌شود، این حالت پایدارتری دارد اما در انتهای پاداش batch size = 32 بهتر است و نشان می‌دهد اینجنت بهتر عمل کرده است.

همین نمودار برای batch size = 128 در شکل ۵ آمده است. همانطور که مشاهده می‌شود این حالت ناپایدارتری دارد.

مطابق شکل ۵ در مراحل اولیه مقدار پاداش نوسان زیادی دارد که نشان دهنده فاز exploration اینجنت است.

مطابق شکل ۴ و ۶۴ در حدود اپیزود ۲۰ یک پاداش منفی بسیار زیاد می‌گیرد که نشان دهنده یک حادثه خیلی بد برای فضای پیما است. بعد از اپیزود ۵۰ دیده می‌شود که پاداش‌ها شروع به صعودی شدن می‌کنند و البته کمی



```

1
2     def update_params(self):
3         """
4             Update the parameters of the value network using experience replay and
5             perform a soft update of the target network parameters.
6         """
7
8         if len(self.experience_replay) < self.batch_size:
9             return
10
11        batch = Transition(*zip(*self.experience_replay.sample(self.batch_size)))
12
13        state_batch = torch.tensor(np.array(batch.state), dtype=torch.float32).to(device)
14        action_batch = torch.tensor(batch.action, dtype=torch.int64).unsqueeze(1).to(device)
15        next_state_batch = torch.tensor(np.array(batch.next_state), dtype=torch.float32).to(
16            device)
17        reward_batch = torch.tensor(batch.reward, dtype=torch.float32).unsqueeze(1).to(device)
18        done_batch = torch.tensor(batch.done, dtype=torch.float32).unsqueeze(1).to(device)
19
20        q_expected = self.value_net(state_batch).gather(1, action_batch)
21        q_targets_next = self.target_net(next_state_batch).detach().max(1)[0].unsqueeze(1)
22        q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))
23        loss = F.mse_loss(q_expected, q_targets)
24
25
26        self.optimizer.zero_grad()
27        loss.backward()
28        self.optimizer.step()
29
30
31        # Soft update target network parameters
32        for target_param, local_param in zip(self.target_net.parameters(), self.value_net.
33            parameters()):
34            target_param.data.copy_(self.tau * local_param.data + (1.0 - self.tau) * target_param
35            .data)

```

Code 8: update params



```

1 def save(self, fname):
2     """
3         Save the model parameters.
4
5     Args:
6         fname (str): Filename to save the model.
7     """
8     torch.save(self.value_net.state_dict(), fname)
9
10    def load(self, fname):
11        """
12            Load the model parameters.
13
14        Args:
15            fname (str): Filename to load the model.
16        """
17        self.value_net.load_state_dict(torch.load(fname, map_location=device))
18        self.target_net.load_state_dict(torch.load(fname, map_location=device))

```

Code 9: save and load the weights

```

1 import torch
2 import torch.optim as optim
3 import numpy as np
4
5 # Define constants
6 n_episodes = 250
7 eps = 1.0
8 eps_decay_rate = 0.97
9 eps_end = 0.01
10 BATCH_SIZE = 32

```

Code 10: batch size 32

```

1 agent = DQNAgent(state_size, action_size, batch_size=BATCH_SIZE)
2 crs = np.zeros(n_episodes)
3 crs_recent = deque(maxlen=25)

```

Code 11: initialize agent



```

1 for i_episode in range(1, n_episodes + 1):
2     if i_episode % 50 == 0:
3         env = RecordVideo(gym.make("LunarLander-v2", render_mode="rgb_array"), f"./DQN/batch{
4             ↪ BATCH_SIZE}/eps{i_episode}", step_trigger=lambda step: step % 1 == 0, video_length=0)
5     else:
6         env = gym.make("LunarLander-v2", render_mode="rgb_array")
7
8     state, info = env.reset()
9     done = False
10    cr = 0

```

Code 12: train loop

```

1 while not done:
2     action = agent.take_action(state, eps)
3     next_state, reward, done, truncated, info = env.step(action)
4     agent.experience_replay.store_transition(state, action, next_state, reward, done or
5         ↪ truncated)
6     agent.update_params()
7     state = next_state
8     cr += reward

```

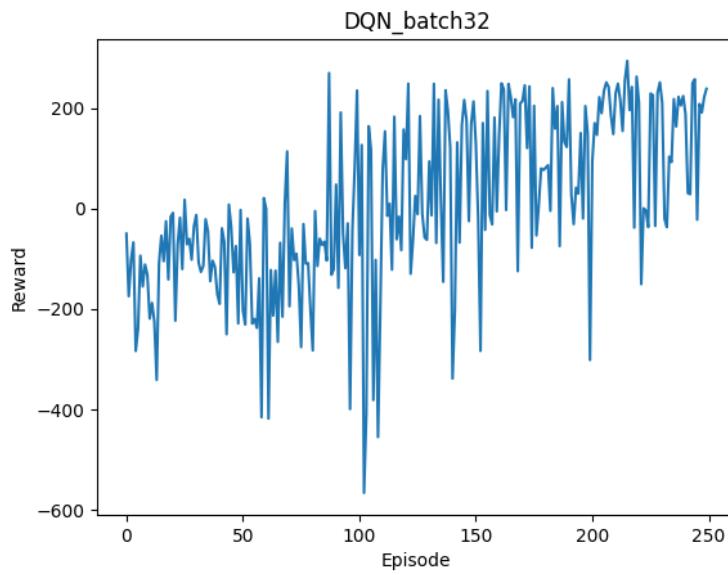
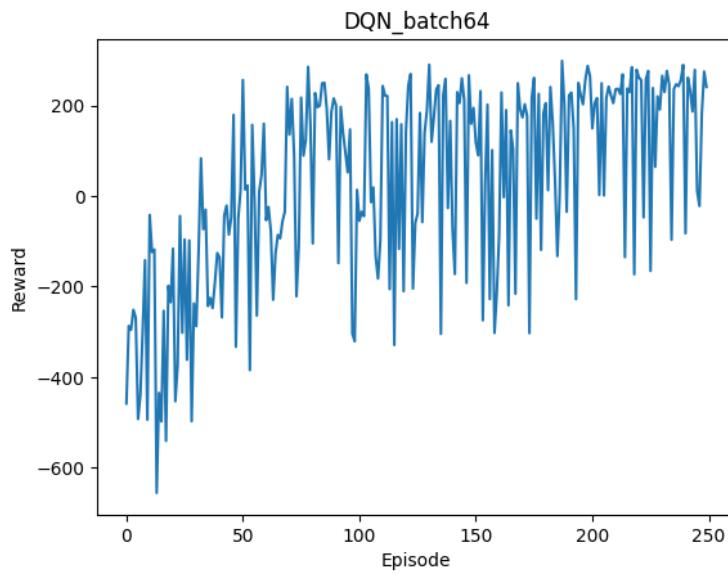
Code 13: episode loop

```

1 # Decay epsilon
2 eps = max(eps * eps_decay_rate, eps_end)
3 crs[i_episode - 1] = cr
4 crs_recent.append(cr)
5
6 # Save model every 50 episodes
7 if i_episode % 50 == 0:
8     agent.save(f"q_net_batch{BATCH_SIZE}_eps{i_episode}.pt")
9
10 # Print average reward every 25 episodes
11 print(f'\rEpisode {i_episode}\tAverage Reward: {np.mean(crs_recent):.2f}\tEpsilon: {eps:.2f}'
12     ↪ , end="")
13 if i_episode % 25 == 0:
14     print(f'\rEpisode {i_episode}\tAverage Reward: {np.mean(crs_recent):.2f}\tEpsilon: {eps
15     ↪ :.2f}')
16
# Close the environment
env.close()

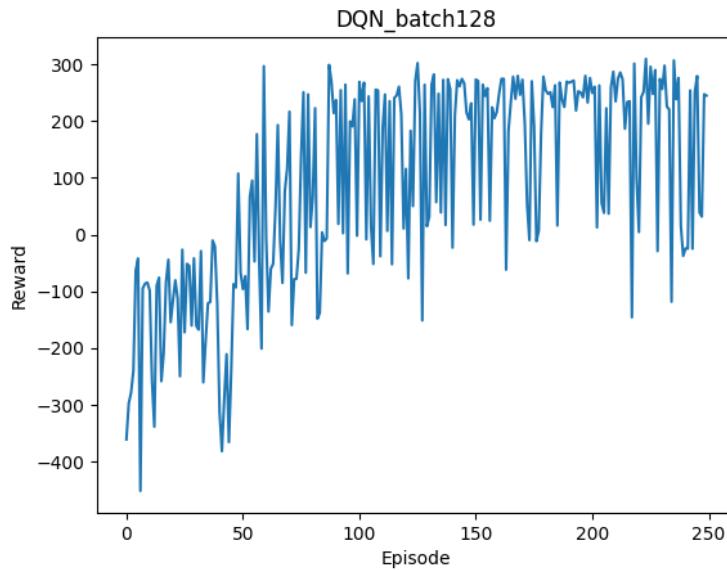
```

Code 14: Epsilon Decay

شکل ۳: نمودار پاداش تجمعی برای  $batch\ size = 32$ شکل ۴: نمودار پاداش تجمعی برای  $batch\ size = 64$ 

نوسانات وجود دارد ولی با جلو رفتن بهتر می شود. از اپیزود ۱۰۰ پاداش ها حدود ۱۰۰ یا مثبت هستند که نشان می دهد ایجنت در حال پایدارسازی عملکرد و رفتار خود است. می توان گفت در هر سه حالت عملکرد ایجنت با مرور زمان و زیادتر شدن تعداد episode بهبود می یابد و نشان می دهد که الگوریتم درست کار می کند.

می توان میانگین هر ۲۵ اپیزود را برای بچه های متفاوت در [شکل ۶](#) تا [شکل ۸](#) مشاهده کرد. مطابق این نمودارها و به صورت شهودی و با توجه به [شکل ۳](#) تا [شکل ۵](#) می توان گفت که بهترین عملکرد مدل مربوط به حالت  $batch\ size = 128$  است. این حالت بهترین بالانس را بین دو فاز exploration و exploitation ارائه می دهد که در نهایت به بیشترین



شکل ۵: نمودار پاداش تجمعی برای batch size = 128

پاداش‌ها منجر می‌شود. در جایگاه‌های بعدی به ترتیب batch size = 32 و batch size = 64 قرار دارند که روند بهبود پاداش در آن‌ها با سرعت کمتری دنبال می‌شود.

```

Episode 25      Average Reward: -108.03 Epsilon: 0.47
Episode 49      Average Reward: -99.23 Epsilon: 0.22Moviepy - Building video /content/DQN/batch32/eps50/r1-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps50/r1-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps50/r1-video-episode-0.mp4
Episode 50      Average Reward: -98.22 Epsilon: 0.22
Episode 75      Average Reward: -83.11 Epsilon: 0.10
Episode 99      Average Reward: -6.59 Epsilon: 0.05Moviepy - Building video /content/DQN/batch32/eps100/r1-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps100/r1-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps100/r1-video-episode-0.mp4
Episode 100     Average Reward: 8.46 Epsilon: 0.05
Episode 125     Average Reward: -61.39 Epsilon: 0.02
Episode 149     Average Reward: 31.99 Epsilon: 0.01Moviepy - Building video /content/DQN/batch32/eps150/r1-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps150/r1-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps150/r1-video-episode-0.mp4
Episode 175     Average Reward: 40.33 Epsilon: 0.01
Episode 175     Average Reward: 142.82 Epsilon: 0.01
Episode 199     Average Reward: 134.62 Epsilon: 0.01Moviepy - Building video /content/DQN/batch32/eps200/r1-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps200/r1-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps200/r1-video-episode-0.mp4
Episode 200     Average Reward: 145.22 Epsilon: 0.01
Episode 225     Average Reward: 214.53 Epsilon: 0.01
Episode 249     Average Reward: 174.53 Epsilon: 0.01Moviepy - Building video /content/DQN/batch32/eps250/r1-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps250/r1-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps250/r1-video-episode-0.mp4
Episode 250     Average Reward: 175.79 Epsilon: 0.01

```

شکل ۶: پاداش تجمعی برای batch size = 32

حال به سراغ ارزیابی این سه حالت به کمک متریک regret می‌رویم. ابتدا تعریفی از این معیار را با هم مرور کنیم.

این معیار برای اندازه‌گیری اوضاع در شرایط تصمیم‌گیری و در فضای یادگیری تقویتی به کار می‌رود. این معیار تفاوت بین پاداشی که توسط ایجنت دریافت شده و پاداشی که دریافت می‌شد اگر ایجنت بهترین تصمیم را در هر اپیزود می‌گرفت، ارزیابی می‌کند. پس می‌توان آن را به صورت رابطه ۱ فرموله کرد.

$$R_t = \mu^* - \mu_a \quad (1)$$



```

Episode 25   Average Reward: -157.51 Epsilon: 0.47
Episode 49   Average Reward: -178.48 Epsilon: 0.22Movieipy - Building video /content/DQN/batch64/eps50/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch64/eps50/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch64/eps100/r1-video-episode-0.mp4
Episode 50   Average Reward: -175.24 Epsilon: 0.22
Episode 75   Average Reward: 6.67 Epsilon: 0.10
Episode 99   Average Reward: 37.77 Epsilon: 0.05Movieipy - Building video /content/DQN/batch64/eps100/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch64/eps100/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch64/eps150/r1-video-episode-0.mp4
Episode 125  Average Reward: 113.18 Epsilon: 0.02
Episode 149  Average Reward: 170.41 Epsilon: 0.01Movieipy - Building video /content/DQN/batch64/eps150/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch64/eps150/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch64/eps200/r1-video-episode-0.mp4
Episode 150  Average Reward: 162.35 Epsilon: 0.01
Episode 175  Average Reward: 92.24 Epsilon: 0.01
Episode 199  Average Reward: 169.26 Epsilon: 0.01Movieipy - Building video /content/DQN/batch64/eps200/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch64/eps200/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch64/eps250/r1-video-episode-0.mp4
Episode 200  Average Reward: 177.33 Epsilon: 0.01
Episode 225  Average Reward: 144.90 Epsilon: 0.01
Episode 249  Average Reward: 75.09 Epsilon: 0.01Movieipy - Building video /content/DQN/batch64/eps250/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch64/eps250/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps50/r1-video-episode-0.mp4
Episode 50   Average Reward: -144.98 Epsilon: 0.22
Episode 75   Average Reward: -1.75 Epsilon: 0.10
Episode 99   Average Reward: 106.53 Epsilon: 0.05Movieipy - Building video /content/DQN/batch128/eps100/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps100/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps150/r1-video-episode-0.mp4
Episode 100  Average Reward: 107.44 Epsilon: 0.05
Episode 125  Average Reward: 142.99 Epsilon: 0.02
Episode 149  Average Reward: 182.61 Epsilon: 0.01Movieipy - Building video /content/DQN/batch128/eps150/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps150/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps200/r1-video-episode-0.mp4
Episode 150  Average Reward: 172.46 Epsilon: 0.01
Episode 175  Average Reward: 199.31 Epsilon: 0.01
Episode 199  Average Reward: 218.05 Epsilon: 0.01Movieipy - Building video /content/DQN/batch128/eps200/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps200/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps250/r1-video-episode-0.mp4
Episode 200  Average Reward: 218.28 Epsilon: 0.01
Episode 225  Average Reward: 186.54 Epsilon: 0.01
Episode 249  Average Reward: 159.23 Epsilon: 0.01Movieipy - Building video /content/DQN/batch128/eps250/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps250/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps50/r1-video-episode-0.mp4
Episode 250   Average Reward: 161.21 Epsilon: 0.01

```

شکل ۷: پاداش تجمعی برای batch size = 64

```

Episode 25   Average Reward: 168.00 Epsilon: 0.47
Episode 49   Average Reward: 143.30 Epsilon: 0.22Movieipy - Building video /content/DQN/batch128/eps50/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps50/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps50/r1-video-episode-0.mp4
Episode 50   Average Reward: -144.98 Epsilon: 0.22
Episode 75   Average Reward: -1.75 Epsilon: 0.10
Episode 99   Average Reward: 106.53 Epsilon: 0.05Movieipy - Building video /content/DQN/batch128/eps100/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps100/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps150/r1-video-episode-0.mp4
Episode 100  Average Reward: 107.44 Epsilon: 0.05
Episode 125  Average Reward: 142.99 Epsilon: 0.02
Episode 149  Average Reward: 182.61 Epsilon: 0.01Movieipy - Building video /content/DQN/batch128/eps150/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps150/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps200/r1-video-episode-0.mp4
Episode 150  Average Reward: 172.46 Epsilon: 0.01
Episode 175  Average Reward: 199.31 Epsilon: 0.01
Episode 199  Average Reward: 218.05 Epsilon: 0.01Movieipy - Building video /content/DQN/batch128/eps200/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps200/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps250/r1-video-episode-0.mp4
Episode 200  Average Reward: 218.28 Epsilon: 0.01
Episode 225  Average Reward: 186.54 Epsilon: 0.01
Episode 249  Average Reward: 159.23 Epsilon: 0.01Movieipy - Building video /content/DQN/batch128/eps250/r1-video-episode-0.mp4.
Movieipy - Writing video /content/DQN/batch128/eps250/r1-video-episode-0.mp4

Movieipy - Done !
Movieipy - video ready /content/DQN/batch128/eps50/r1-video-episode-0.mp4
Episode 250   Average Reward: 161.21 Epsilon: 0.01

```

شکل ۸: پاداش تجمعی برای batch size = 128

در رابطه ۱<sup>\*</sup>  $\mu_a$  پاداشی است که اگر ایجنت بهترین اکشن را انتخاب می‌کرد. در طرف دیگر  $\mu_a$  پاداشی است که ایجنت با انتخاب کردن اکشن  $a$  در زمان  $t$  آن را دریافت کرده است.

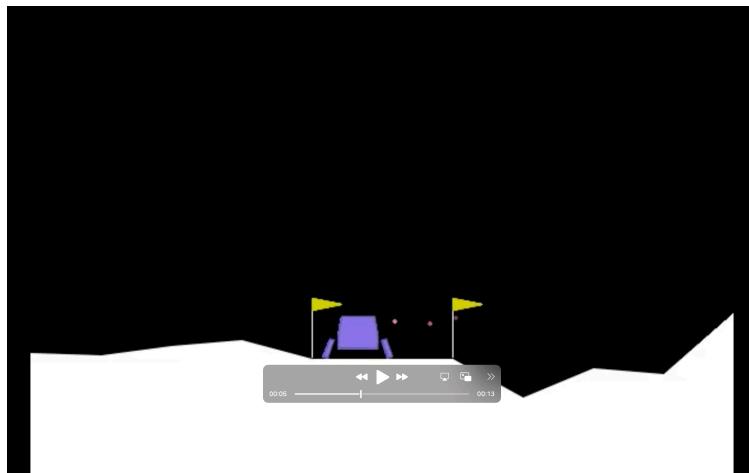
حال اگر ما رابطه ۱ را در تمامی اپیزودها حساب کنیم می‌توانیم regret تجمعی برای ایجنت را بدست آوریم.

در واقع می‌توان گفت regret هزینه انتخاب نکردن بهترین تصمیم در هر اپیزود را به ما نشان می‌دهد.

مطابق نتایج حالت batch size = 32 در ابتدا regret متوسط است، سپس این معیار به صورت واضحی کم می‌شود چون reward ها در حال افزایش هستند (حدود اپیزود ۵۰) و در نهایت در اپیزودهای بالاتر این معیار به کمترین حالت خود می‌رسد از آنجایی که پاداش ها در حال افزایش هستند.

برای حالت batch size = 128 ما کمترین میزان regret را در فازهای مختلف شاهد هستیم چون بیشترین پاداش ها را برای این حالت دریافت می‌کنیم و می‌توان اینطور توصیف کرد که در این حالت ایجنت به بهترین عملکرد خود نزدیک است. پس با توجه به این معیار نیز بهترین گزینه ما حالت batch size = 128 است.

در نهایت به سراغ تهیه فیلم ها می‌رویم که در روند آموزش آن ها را ذخیره کرده ایم. توجه شود که تمامی فیلم های مربوط به هر سه بچ در فایل ارائه آمده است.



شکل ۹: ویدیویی از ایجنت در محیط به ازای  $batch\ size = 128$  در اپیزود ۲۵۰

```

pip install pyvirtualdisplay # Install the missing module
from pyvirtualdisplay import Display
import glob
import os
import base64
from IPython.display import HTML, display as ipythondisplay

# Initialize virtual display
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 = open(mp4, 'rb').read()
        encoded = base64.b64encode(video)
        ipythondisplay.HTML(data=f'''<video alt="test" autoplay loop controls style="height: 400px;">
            <source src="data:video/mp4;{encoded.decode('ascii')}" type="video/mp4" />
        </video>''')
    else:
        print("Could not find video")

```

شکل ۱۰: ساختن ویدیو از ایجنت در محیط  $batch\ size = 128$  در اپیزود ۲۵۰

## ۳.۲ قسمت سوم

باید در این قسمت مدل DDQN را به جای مدل DQN جاگذاری کنیم. می‌توان گفت تفاوت اصلی این دو مدل در نحوه بروزرسانی Q value ها و شبکه هدف می‌باشد. به طور دقیق‌تر، DQN از شبکه استفاده می‌کند تا Q value ها را تخمین بزند اما DDQN از شبکه موجود اکشن را انتخاب می‌کند و Q value های مرحله بعد را به کمک شبکه هدف تخمین می‌زند. این تفاوت در [برنامه ۱۵](#) و [۱۶](#) قابل مشاهده است.

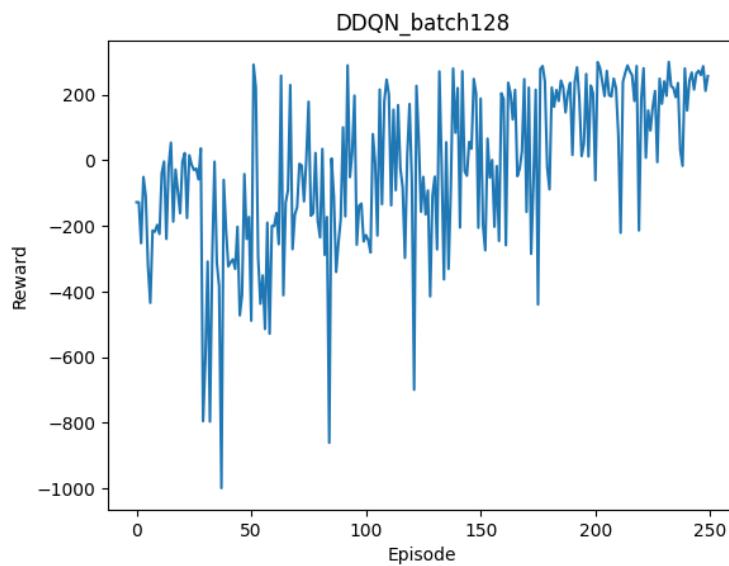
```

1     q_targets_next = self.target_net(next_state_batch).detach().max(1)[0].unsqueeze(1)
2     q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))

```

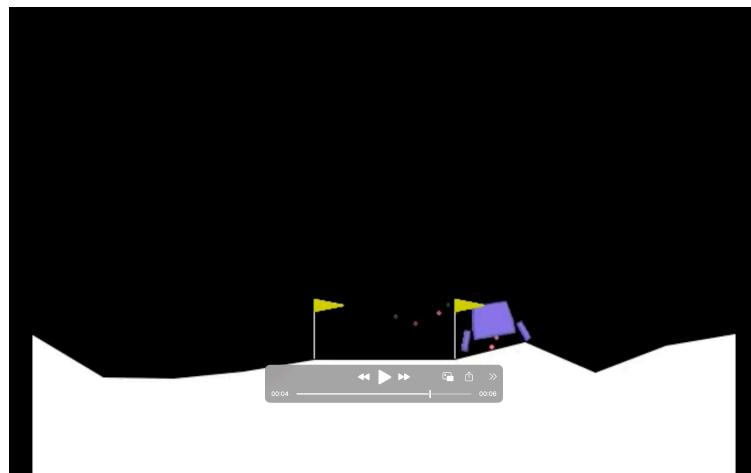
Code 15: DQN Agent

در این حالت دیده می‌شود که پاداش‌ها بسیار پایدارتر از حالت قبل هستند، یعنی پایداری ایجنت در دریافت پاداش‌های بیشتر بهتر شده است. برای تهیه ویدیو مانند بخش قبل از دستورات مربوط به نمایش ویدیو در محیط google collab استفاده می‌کنیم و به



شکل ۱۱: نمودار پاداش تجمعی به کمک DDQN در حالت batch size = 128

صورت گفته شده برای دو حالت اپیزود ۱۰۰ و ۲۵۰ ویدیو را نشان می‌دهیم. قابل ذکر است این ویدیوها در فایل ارسالی قرار دارند.



شکل ۱۲: ویدیویی از ایجنت در محیط به ازای batch size = 128 در اپیزود ۲۵۰