دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکدهٔ مهندسی برق - گروه مهندسی کنترل

درس یادگیری ماشین

مینی پروژه سوم

نام و نام خانوادگی	علی مهرابی
شمارهٔ دانشجویی	۴۰۲۲۲۳۸۵۴
تاریخ	اردیبهشت۱۴۰۳
استاد درس	دکتر علیاری

فهرست مطالب

۱ لینک های مربوطه ۴
۱.۱ لینک مربوط به گیت هاب ۴
۲.۱ لینک مربوط به گوگل کلب ۴

۲ سوال دوم ۵
۱.۲ قسمت اول ۵
۲.۲ قسمت دوم ۶
۳.۲ قسمت سوم ۱۶

فهرست تصاویر

۱ محیط Lunar Lander ۵
۲ محیط import Lunar Lander from Gym ۷
۳ نمودار پاداش تجمعی برای batch size = 32 . . . . . ۱۳
۴ نمودار پاداش تجمعی برای batch size = 64 . . . . . ۱۳
۵ نمودار پاداش تجمعی برای batch size = 128 . . . . . ۱۴
۶ پاداش تجمعی برای batch size = 32 . . . . . ۱۴
۷ پاداش تجمعی برای batch size = 64 . . . . . ۱۵
۸ پاداش تجمعی برای batch size = 128 . . . . . ۱۵
۹ ویدیویی از ایجنت در محیط به ازای batch size = 128 در اپیزود ۲۵۰ . . . . . ۱۶
۱۰ ساختن ویدیو از ایجنت در محیط batch size = 128 در اپیزود ۲۵۰ . . . . . ۱۶
۱۱ نمودار پاداش تجمعی به کمک DDQN در حالت batch size = 128 . . . . . ۱۷
۱۲ ویدیویی از ایجنت در محیط به ازای batch size = 128 در اپیزود ۲۵۰ . . . . . ۱۸

فهرست برنامه‌ها

۱ swig installing ۶
۲ Xvbf installing ۷
۳ GPU using ۷
۴ display virtual a initializing ۸
۵ replay Experience ۸
۶ class DeepQNetwork ۹
۷ action take ۱۰
۸ params update ۱۱
۹ weights the load and save ۱۱
۱۰ ۳۲ size batch ۱۱
۱۱ agent initialize ۱۲
۱۲ loop train ۱۲
۱۳ loop episode ۱۲
۱۴ Decay Epsilon ۱۲
۱۵ Agent DQN ۱۷
۱۶ Agent DDQN ۱۷

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.

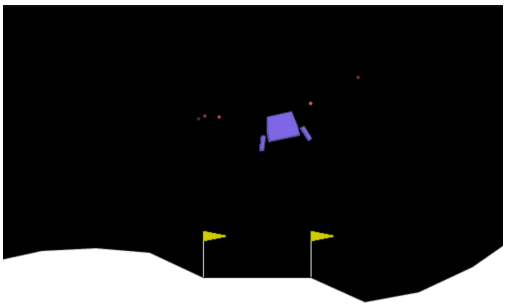
۲.۱ لینک مربوط به گوگل کلب

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 Lunar Lander یک محیط توسعه یافته توسط Open AI در قالب یک کتابخانه متن باز به نام Gym است. از این قالب برای شبیه سازی ها و اهداف آموزشی برای آموزش Reinforcement Learning استفاده می شود. این کتابخانه شامل محیط های زیادی است که هریک مارا با جنبه ای از RL آشنا می کند.مسائل برگرفته از کنترل کلاسیک مانند Cart Pole تا بازی آتاری و حتی شبیه سازی های پیچیده تر مانند MuJoCo. از خوبی های این منبع متن باز؛ یکپارچه بودن آن در تمامی فضا هاست به طوری که تمامی فضاها و نتایج مربوط به آن با هم قابل مقایسه هستند. متد هایی مانند close()، reset()و یا step(action) از جمله این متد ها هستند. حالا که با فضای کلی، توسعه دهنده و اهداف این کتابخانه متن باز آشنا شدیم به سراغ یکی از محیط های آن یعنی Lunar Lander می رویم. شکل ۱ تصویری از این محیط نشان می دهد که در ان یک فضاپیما قصد دارد روی یک مساحت مشخص فرود بیاید. در واقع این یک مسئله بهینه سازی مسیر این راکت یا فضاپیما است که یک مساله کلاسیک است. با توجه به اصل Pontryagin's maximum حالتی بهینه است که موتور در حالت حداکثری خود روشن باشد یا خاموش شود. به همین علت هر موتور دو حالت دارد؛ روشن باشد یا خاموش باشد.



شکل ۱: محیط Lunar Lander

Action Space در این بازی شامل چهار اکشن گسسته است:

- کاری انجام نشود
- موتور جهت چپ روشن شود
- موتور اصلی روشن شود
- موتور جهت راست روشن شود

State Space در این بازی شامل یک بردار ۸ بعدی است.

- پوزیشن X که می تواند بین -1.5 تا 1.5 باشد.
- پوزیشن Y که می تواند بین -1.5 تا 1.5 باشد.

- سرعت خطی در جهت X که می تواند بین -5 تا 5 باشد.
- سرعت خطی در جهت Y که می تواند بین -5 تا 5 باشد.
- در جه که می تواند بین -3.14 تا 3.14 (به رادیان) باشد.
- سرعت زاویه ای که می تواند بین -5 تا 5 باشد.
- یک متغیر بولین که نشان می دهد پای ۱ با زمین ارتباط دارد یا نه
- یک متغیر بولین که نشان می دهد پای ۲ با زمین ارتباط دارد یا نه

Reward System به این صورت است که بازیگر یا فضاپیما را ترغیب می کند تا به صورت دقیق و نرم در محل موردنظر فرود بیاید. پاداش برای حرکت از بالا و فرود آمدن روی سکوی مورد نظر ۱۰۰ تا ۱۴۰ امتیاز است. اگر فضاپیما از سکویی که باید روی آن فرود بیاید فاصله بگیرد پاداش از دست می دهد. اگر تصادف بکند -۱۰۰ امتیاز از دست می دهد. اگر روی سکو بنشیند و بی حرکت بماند یا rest انجام دهد ۱۰۰ امتیاز دریافت می کند. اگر هر پا با زمین برخورد کند ۱۰ امتیاز می گیرد. راه افتادن موتور اصلی -0.03 امتیاز در هر فریم را در بر دارد و برای موتور های کناری نیز همین مقدار است.اینکار برای تشویق به استفاده کمتر از سوخت انجام می گیرد. رسیدن به حالت solved ۲۰۰ امتیاز دارد.
پوزیشن ابتدایی فضاپیما در مرکز و بالای تصویر است و در ابتدا یک نیروی رندوم به مرکز جرم آن اعمال می شود.
هر اپیزود بنابر یکی از دلایل زیر تمام می شود:

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

۲.۲ قسمت دوم

برای حل این قمست از کد قرار گرفته به عنوان نمونه استفاده شده و تغییراتی در آن اعمال شده تا ارور های آن برطرف شود. در مرحله اول ابتدا باید موارد و کتابخانه های 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 برنامهٔ ۱ آمده اند.

```
!apt-get update
!apt-get install -y swig
```

Code 1: installing swig

دوم برای نشان دادن ویدیو ها به صورت مجازی باید از Xvbf استفاده کنیم که نیاز است تا آن را در محیط google collab نصب کنیم که به وسیله برنامهٔ ۲ اینکار انجام شده است.
سایر دستورات در بخش نصب ثابت باقی مانده است و به سراغ مرحله بعد می رویم.
در مرحله بعد دستور مورد نیاز برای استفاده از GPU در محیط google collab را به صورت برنامهٔ ۳ وارد می کنیم که در نتیجه به ما می گوید که ما در حال استفاده از Coda و در حال ران کردن GPU هستیم.

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

Code 2: installing Xvbf

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

Code 3: using GPU

در ادامه به صورت شکل ۲ محیط مورد نظر که همان Lunar Lander است را از Gym وارد می کنیم. همانطور که مشاهده می شود ما ۸ observation یا state داریم و می توانیم ۴ action انجام دهیم که همگی در بخش ۱.۲ توضیح داده شد. در مرحله بعد ابتدا یک صفحه نمایش مجازی تعریف می کنیم و سایز آن را مشخص می کنیم و سپس یک

```
[5] # enviroment
    import gym
    env = gym.make('LunarLander-v2', render_mode="rgb_array")
    #TODO: find observation size: 8
    state_size = env.observation_space.shape[0]
    #TODO: find action size: 4: 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

    (8, 4)
```

شکل ۲: محیط import Lunar Lander from Gym

تابع تعریف می کنیم تا بتوانیم ویدیو هارا که در آینده قصد ضبط آن هارا داریم در محیط Jupiter notebook ببینیم. انجام اینکار در برنامهٔ ۴ آمده است.

در مرحله بعد برنامهٔ ۵ را داریم. در ابتدای آن یک تاپل می سازیم که حالت یک experience را در خود ذخیره کند. این موارد شامل حالتی که فضاپیما در آن وجود دارد (قبل از انجام اکشن)، اکشنی که انجام می دهد، حالت بعدی که با انجام اکشن به آن می رود، پاداشی که دریافت می کند و یک متغیر بولین که نشان می دهد آیا عمل تمام شده یا نه است. سپس کلاس ExperienceReplay یک reply buffer را پیاده می کند. در واقع تمامی experience های گذشته یا همان transitions را در خود ذخیره می کند تا در مرحله آموزش از آنها استفاده شود. برای اینکار از یک deque یا double-ended queue استفاده می شود تا memory ساخته شود. reply buffer به agent اجازه می دهد تا از تجربیات گذشته خود با سمپل کردن به صورت رندوم از batch های transition یاد بگیرد و این رندوم سازی باید پایداری یادگیری می شود و موضوع correlation بین experience های متوالی را برای ما بهبود می دهد. store trans ترنزیشن های جدید را به reply buffer اضافه می کند.

در مرحله بعد شبکه عصبی عمیق مربوط به الگوریتم DQN را تشکیل می دهیم. برنامهٔ ۶ این شبکه را نشان می دهد که در کلاس DeepQNetwork تعریف شده است. همانطور که مشخص است این یک شبکه sequential است که نسبت به کد اصلی به صورت مستقیم پیاده سازی شده و برگردانده می شود. ورودی این شبکه به اندازه state ها نرون دارد و در نهایت خروجی این شبکه همان Q value ها هستند که مقادیر انتظاری از پاداش آینده را برای هر اکشن به ما می دهد. پس خروجی این شبکه به اندازه اکشن ها است و مقادیر Q برای آن ها تخمین زده می شود. در لایه اول یک

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

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")
```

Code 4: initializing a virtual display

```
Transition = namedtuple('Transition', ('state', 'action', 'next_state', 'reward', 'done'))

class ExperienceReplay():
    def __init__(self, capacity):
        self.memory = deque([], maxlen=capacity)

    def store_trans(self, s, a, sp, r, done):
        transition = Transition(s, a, sp, r, done)
        self.memory.append(transition)

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

Code 5: Experience replay

لایه تماما متصل با تابع فعال ساز ReLU استفاده شده است. سپس از Layer normalizaiton استفاده شده تا روند یادگیری را پایدار سازد و سرعت ببخشد، 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 تماما متصل است و دوباره از نرمالیزه کردن و Dropout استفاده شده و تابع فعال ساز ReLU است. لایه سوم نیز به همین صورت. متد forward ورودی تنسور x را به شبکه می دهد و در خروجی Q value های مربوط به هر اکشن بازگردانده می شود.

```python
class DeepQNetwork(nn.Module):
    def __init__(self, state_size, action_size):
        super(DeepQNetwork, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(state_size, 512),
            nn.ReLU(),
            nn.LayerNorm(512),
            nn.Dropout(0.1),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.LayerNorm(512),
            nn.Dropout(0.1),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, action_size)
        )

    def forward(self, x):
        return self.net(x)
```

Code 6: DeepQNetwork class

کلاس بعدی DQNAgent است که Agent مارا پیاده سازی می کند. ایجنتی که با محیط interact می کند، experience هارا ذخیره می کند و Q value هارا بروز رسانی می کند. ابتدا نیاز است تا متغیر هارا 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 بهینه ساز و چند مورد دیگر نیز داریم.

متد بعدی در این کلاس take action است که در برنامهٔ ۷ آمده است. که بر اساس epsilon greedy policy از حالت فعلی یک اکشن انتخاب می کند. اگر یک عدد رندوم از eps بزرگتر باشد، ایجنت اکشنی را انتخاب می کند که بیشترین Q-value که در شبکه اصلی پیش بینی شده دارد. در غیر این صورت ایجنت یک اکشن رندوم انتخاب می کند و این تفاوت بین exploitation و exploration است. این روش مطمعن می شود که بالانس بین این دو روش در آموزش رعایت می شود.

متد بعدی update params است که در برنامهٔ ۸ آمده است. این متد پارامتر های شبکه اصلی را به کمک تجربه های گرفته شده از replay buffer بروزرسانی می کند و یک soft update را روی شبکه هدف انجام می دهد. در واقع به کمک sample(batch size) یک بچ از تجربه را از reply buffer برمیدارم و با فرمول بلمن مقدار Q value را حساب

```
    def take_action(self, state, eps=0.0):
        self.value_net.eval()
        if random.random() > eps:
            with torch.no_grad():
                return torch.argmax(self.value_net(torch.tensor(state).float().unsqueeze(0).to(
    ↪ device))).item()
        else:
            return np.random.randint(0, self.action_size)
```

Code 7: take action

میکند. سپس اختلاف بین مقدار پیش بینی شده و اصلی را می یابد و یک گرادیان کاهشی برای مینیمایز کردن این خطا اجرا می کند.

در نهایت به کمک برنامهٔ ۹ این وزن هارا ذخیره و بارگزاری می کند.

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 می کند. برای مثال برای بچ ۳۲، به صورت برنامهٔ ۱۰ خواهد بود.

حال به توضیح فرآیند آموزش می پیردازیم. ابتدا به صورت برنامهٔ ۱۱ ایجنت را تعریف می کنیم و همچنین پاداش تجمعی را initialize می کنیم. در این قسمت ما یک ایجنت از DQNAgent با حالت و اکشن خاص درست می کنیم که بچ سایز اش را مشخص کرده ایم. همچنین صف های ۲۵ تایی تشکیل می دهیم که در آینده از آنها میانگین می گیریم.

در برنامهٔ ۱۲ حلقه آموزش را به تعداد مشخصی از اپیزود که در این حالت ۲۵۰ است تشکیل می دهیم. در این کد از هر ۵۰ امین اپیزود فیلم برداری می شود یعنی اپیزود ۵۰ ام ۱۰۰ ام ۱۵۰ ام ۲۰۰ ام و ۲۵۰ ام. سپس محیط را ریست می کند و مقادیر اولیه را میگیرد. سپس done را که قبل تر توضیح داده شدذه بود به حالت false در می آورد تا اپیزود بعدی استارت شود. سپس مقدار پاداش برای اپیزود فعلی را صفر می گذارد.

بخش برنامهٔ ۱۳ تا زمانی که اپیزود تمام شود با محیط ارتباط می گیرد، تجربه کسب می کند و Q value هارا به روز می کند. در واقع ابتدا یک اکشن انتخاب می شود، حالت بعدی، فلگ و پاداش را میگیرد و تجربه کسب شده را در reply buffer ذخیره می کند. سپس Q value function را به روز رسانی می کند (به کمک سمپل گرفته از reply buffer) سپس حالت را زا حالت فعلی به حالت بعدی تغییر میدهد و پاداش را به پاداش های قبلی در آن اپیزود اضافه میکند.

بخش بعدی اپسیلون را کاهش می دهد و مقدار پاداش تجمعی در این اپیزود را ذخیره می کند. اگر اپیزود مضربی از ۵۰ بود آن را ذخیره می کند و اگر مضرب ۲۵ بود، پاداش میانگین آم و اپیزود فعلی و مقدار اپسیلون را پرینت می کند. این بخش در برنامهٔ 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. که همانگونه که دیده میشود وضعیت پایدار تری را دارد ولی در انتها پاداش batch size = 32 بهتر است و نشان می دهد ایجنت بهتر عمل کرده است.

همین نمودار برای batch size = 128 در شکل ۵ آمده است. که همانگونه که دیده میشود وضعیت ناپایدار تری را دارد . مطابق شکل ۳ تا شکل ۵ در مراحل اولیه مقدار پاداش نوسان زیادی دارد که نشان دهنده فاز exploration ایجنت

```python
def update_params(self):
        if len(self.experience_replay) < self.batch_size:
            return
        batch = Transition(*zip(*self.experience_replay.sample(self.batch_size)))

        state_batch = torch.tensor(np.array(batch.state), dtype=torch.float32).to(device)
        action_batch = torch.tensor(batch.action, dtype=torch.int64).unsqueeze(1).to(device)
        next_state_batch = torch.tensor(np.array(batch.next_state), dtype=torch.float32).to(
    ↪ device)
        reward_batch = torch.tensor(batch.reward, dtype=torch.float32).unsqueeze(1).to(device)
        done_batch = torch.tensor(batch.done, dtype=torch.float32).unsqueeze(1).to(device)

        q_expected = self.value_net(state_batch).gather(1, action_batch)
        q_targets_next = self.target_net(next_state_batch).detach().max(1)[0].unsqueeze(1)
        q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))
        loss = F.mse_loss(q_expected, q_targets)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # Soft update target network parameters
        for target_param, local_param in zip(self.target_net.parameters(), self.value_net.
    ↪ parameters()):
            target_param.data.copy_(self.tau * local_param.data + (1.0 - self.tau) * target_param
    ↪ .data)
```

Code 8: update params

```python
    def save(self, fname):
        torch.save(self.value_net.state_dict(), fname)

    def load(self, fname):
        self.value_net.load_state_dict(torch.load(fname, map_location=device))
```

Code 9: save and load the weights

```python
# NOTE: DON'T change values
n_episodes = 250
eps = 1.0
eps_decay_rate = 0.97
eps_end = 0.01
BATCH_SIZE = 32
```

Code 10: batch size 32

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

Code 11: initialize agent

```
for i_episode in range(1, n_episodes + 1):
    env = RecordVideo(gym.make("LunarLander-v2", render_mode="rgb_array"), f"./DQN/batch{BATCH_SIZE}/eps{i_episode}") if i_episode  50 == 0 else gym.make("LunarLander-v2", render_mode="rgb_array")
    state, info = env.reset()
    done = False
    cr = 0
```

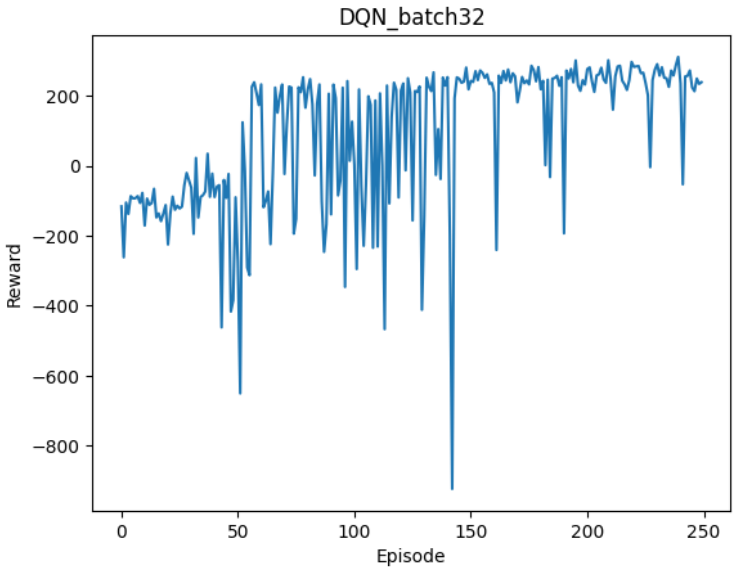
Code 12: train loop

```
    while not done:
        action = agent.take_action(state, eps)
        next_state, reward, done, truncated, info = env.step(action)
        agent.experience_replay.store_trans(state, action, next_state, reward, done or truncated)
        agent.update_params()
        state = next_state
        cr += reward
```

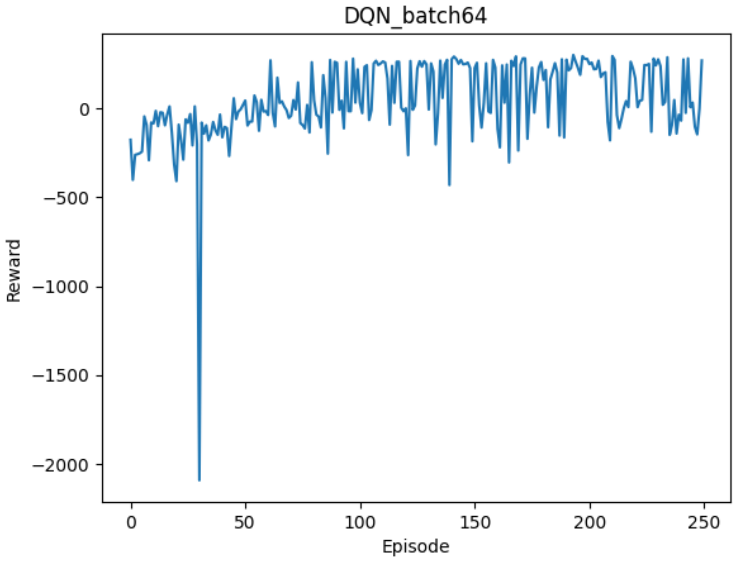
Code 13: episode loop

```
    eps = max(eps * eps_decay_rate, eps_end)
    crs[i_episode - 1] = cr
    crs_recent.append(cr)
    if i_episode % 50 == 0:
        agent.save(f"q_net_batch{BATCH_SIZE}_eps{i_episode}.pt")

    print(f'\rEpisode {i_episode}\tAverage Reward: {np.mean(crs_recent):.2f}\tEpsilon: {eps:.2f}', end="")
    if i_episode % 25 == 0:
        print(f'\rEpisode {i_episode}\tAverage Reward: {np.mean(crs_recent):.2f}\tEpsilon: {eps:.2f}')
```

Code 14: Epsilon Decay



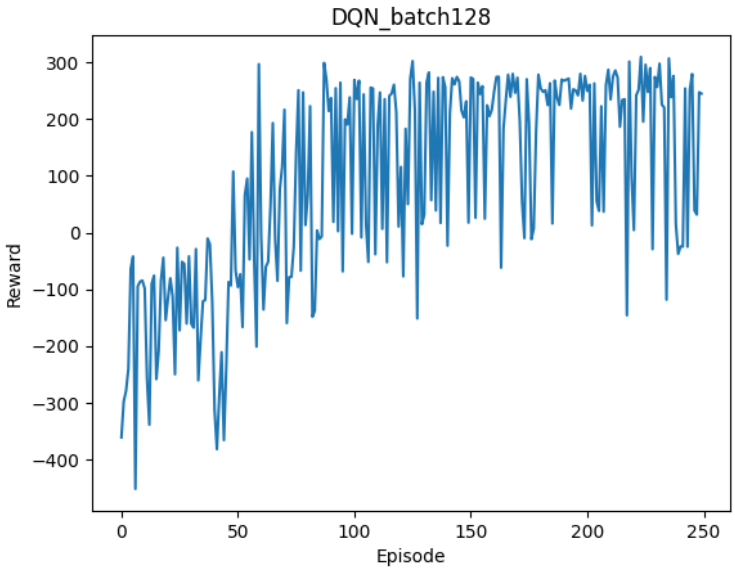
شکل ۳: نمودار پاداش تجمعی برای batch size = 32



شکل ۴: نمودار پاداش تجمعی برای batch size = 64

است.

مطابق شکل ۴ و batch size = 64 در حدود اپیزود ۲۰ یک پاداش منفی بسیار زیاد میگیرد که نشان دهنده یک حادثه خیلی بد برای فضاپیما است. بعد از اپیزود ۵۰ دیده می شود که پاداش ها شروع به صعودی شدن می کنند و البته کمی نوسانات وجود دارد ولی با جلو رفتن بهتر وضعیت 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.



شکل ۵: نمودار پاداش تجمعی برای batch size = 128

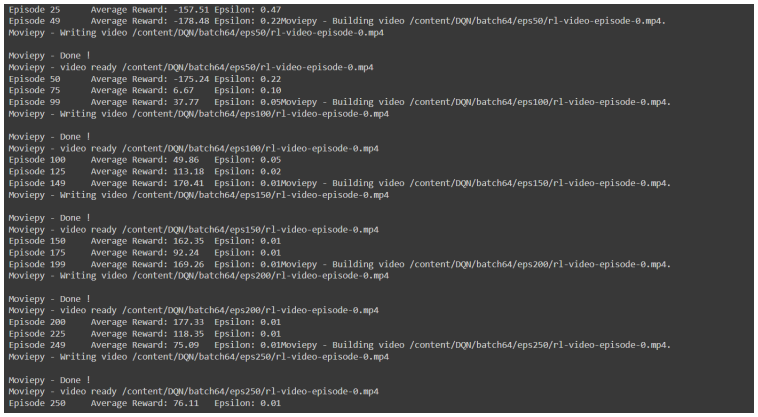
می توان میانگین هر ۲۵ اپیزود را برای بچ های متفاوت در شکل ۶ تا شکل ۸ مشاهده کرد. مطابق این نمودار ها و به صورت شهودی و با توجه به شکل ۳ تا شکل ۵ می توان گفت که بهترین عملکرد مدل مربوط به حالت batch size = 128 است. این حالت بهترین بالانس را بین دو فاز exploraiton و exploatation ارائه می دهدکه در نهایت به بیشترین پاداش ها منجر می شود. در جایگاه ها بعدی به ترتیب batch size = 64 و batch size = 32 قرار دارند که روند بهبود پاداش در آن ها با سرعت کمتری دنبال می شود.

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

/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.py:233: DeprecationWarning: `np.bool8` is a deprecated alias for `
  if not isinstance(terminated, (bool, np.bool8)):
Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps50/rl-video-episode-0.mp4
Episode 50      Average Reward: -98.75  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/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps100/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps100/rl-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/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps150/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps150/rl-video-episode-0.mp4
Episode 150     Average Reward: 48.33   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/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps200/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch32/eps200/rl-video-episode-0.mp4
Episode 200     Average Reward: 145.22  Epsilon: 0.01
Episode 225     Average Reward: 214.59  Epsilon: 0.01
Episode 249     Average Reward: 174.53  Epsilon: 0.01Moviepy - Building video /content/DQN/batch32/eps250/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch32/eps250/rl-video-episode-0.mp4

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

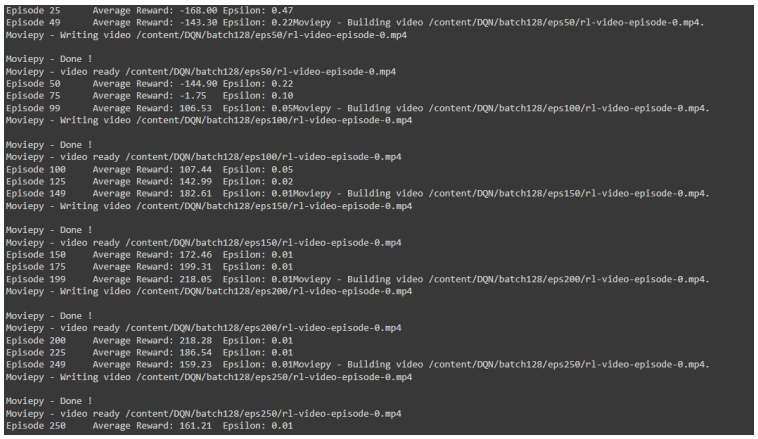
شکل ۶: پاداش تجمعی برای batch size = 32

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

این معیار برای اندازی گیری اوضاع در شرایط تصمیم گیری و در فضای یاگیری تقویتی به کار می رود. این معیار



شکل ۷: پاداش تجمعی برای batch size = 64



شکل ۸: پاداش تجمعی برای batch size = 128

تفاوت بین پاداشی که توسط ایجنت دریافت شده و پاداشی که دریافت می شد، اگر ایجنت بهترین تصمیم را در هر اپیزود میگرفت ارزیابی می کند. پس می توان آن را به صورت رابطهٔ ۱ فرموله کرد.

$$R_t = \mu^\star - \mu_a \qquad (۱)$$

در رابطهٔ ۱ $\mu^\star$ پاداشی است که اگر ایجنت بهترین اکشن را انتخاب می کرد دریافت می کرد. در طرف دیگر μ_a پاداشی است که ایجنت با انتخاب کردن اکشن a در زمان t آن را دریافت کرده است.

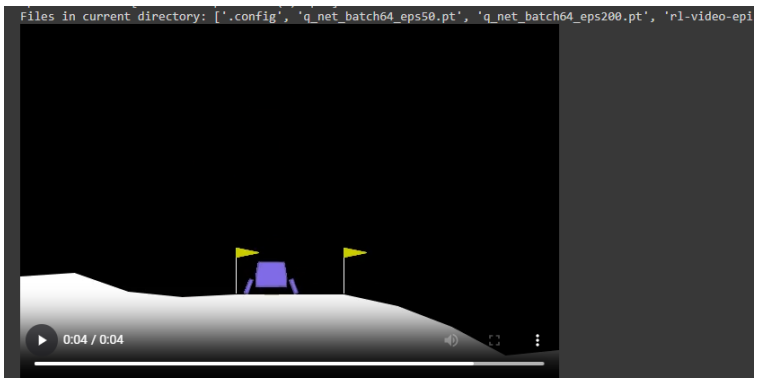
حال اگر ما رابطهٔ ۱ را در تمامی اپیزود ها حساب کنیم می توانیم regret تجمعی برای ایجنت را بدست آوریم.

در واقع می توان گفت regret هزینه انتخاب نکردن بهترین تصمیم در هر اپیزود را به ما نشان می دهد.

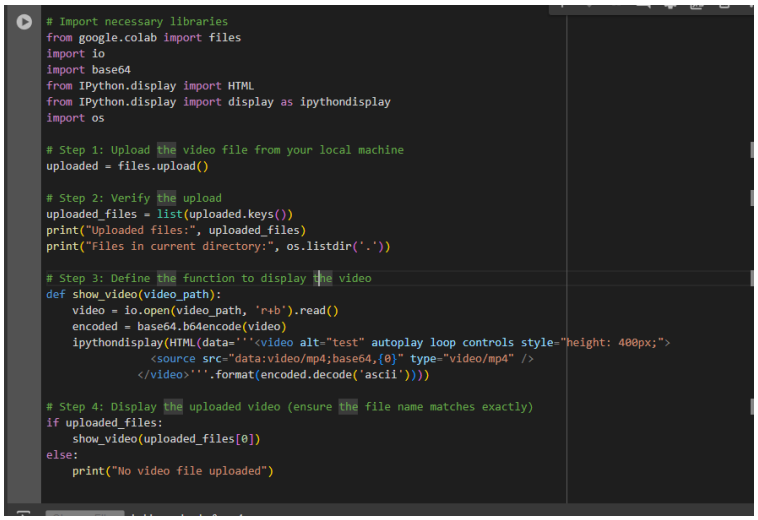
مطابق نتایج حالت batch size = 32 در ابتدا regret متوسط است، سپس این معیار به صورت واضحی کم میشود چون reward ها د رحال افزایش هستند (حدود اپیزود ۵۰) و در نهایت در اپیزود های بالاتر این معیار به کمترین حالت خود میرسد از آنجایی که پاداش ها در حال افزایش هستند.

برای حالت batch size = 128 ما کمترین میزان regret را در فاز های مختلف شاهد هستیم چون بیشتری پاداش هارا برای این حالت دریافت می کنیم و می توان 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 این بخش به سراغ تهیه فیلم ها می رویم که در روند آموزش آن هارا ذخیره کرده ایم. توجه شود که تمامی فیلم های مربوط به هر سه بچ در فایل ارائه آمده است.
برای حالت batch size = 128 این ویدیو ها روی فایل google collab به صورت شکل ۹ قرار گرفته اند.



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



```
# Import necessary libraries
from google.colab import files
import io
import base64
from IPython.display import HTML
from IPython.display import display as ipythondisplay
import os

# Step 1: Upload the video file from your local machine
uploaded = files.upload()

# Step 2: Verify the upload
uploaded_files = list(uploaded.keys())
print("Uploaded files:", uploaded_files)
print("Files in current directory:", os.listdir('.'))

# Step 3: Define the function to display the video
def show_video(video_path):
    video = io.open(video_path, 'r+b').read()
    encoded = base64.b64encode(video)
    ipythondisplay(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'))))

# Step 4: Display the uploaded video (ensure the file name matches exactly)
if uploaded_files:
    show_video(uploaded_files[0])
else:
    print("No video file uploaded")
```

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

این ویدیو ها به ازای تمامی اپیزود ها در فایل google collab قرار گرفته اند.

۳.۲ قسمت سوم

باید در این قسمت مدل DDQN را به جای مدل DQN جاگذاری کنیم. می توان گفت تفاوت اصلی این دو مدل در نحوه بروز کرد Q value ها و شبکه هدف می باشد. به طور دقیق تر DQN از شبکه استفاده می کند تا Q value هارا تخمین بزند اما DDQN از شبکه موجود اکشن را انتخاب می کند و Q value های مرحله بعد را به کمک شبکه هدف تخمین می زند. این تفاوت در برنامهٔ ۱۵ و برنامهٔ ۱۶ قابل مشاهده است.

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

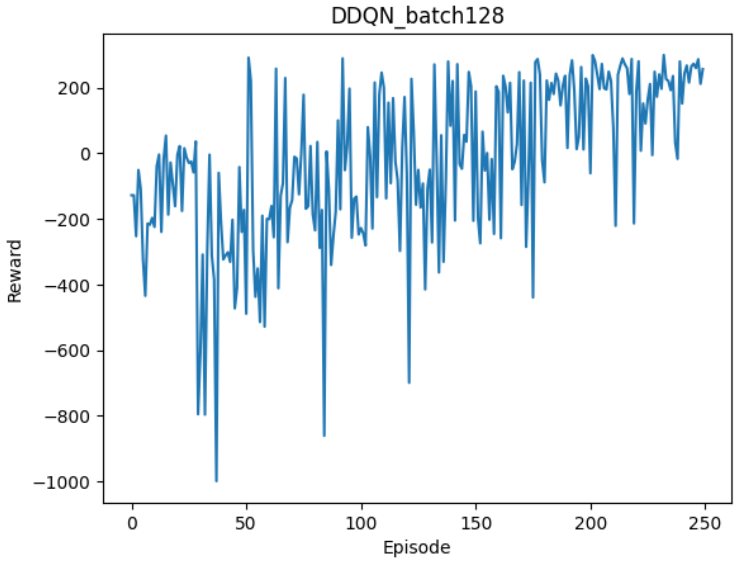
Code 15: DQN Agent

```
next_action_batch = torch.argmax(self.value_net(next_state_batch), dim=1, keepdim=True)
q_targets_next = self.target_net(next_state_batch).gather(1, next_action_batch).detach()
q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))
```

Code 16: DDQN Agent

مشاهده می شود که DDQN یک next action batch دارد که انتخاب اکشن با کمک شبکه فعلی را انجام می دهد که در DQN وجود ندارد. خوبی این روش این است که از بایاس های احتمالی در شبکه DQN جلوگیری می کند چون گام های انتخاب اکشن و تخمین Q value با هم تفاوت دارد. در نهایت این روش می تواند به پاسخ هایی پایدارتر و با اتکاتری را ارائه دهد.

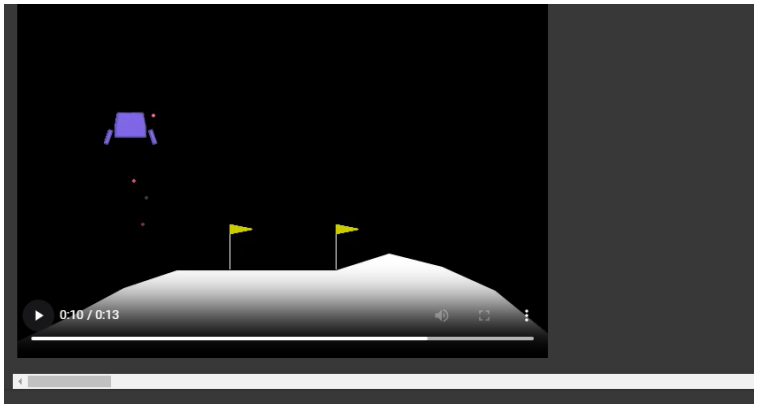
نتایج شبیه سازی با کمک DDQN در حالت batch size = 128 در شکل ۱۱ آمده است. در این حالت دیده می شود



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

که پاداش ها بسیار پایدار تر از حالت قبل هستند، یعنی پایداری ایجنت در دریافت پاداش های بیشتر بهتر شده است.

برای تهیه ویدیو مانند بخش قبل از دستورات مربوط به نمایش ویدیو در محیط google collab استفاده می کنیم و به صورت گفته شده برای دو حالت اپیزود ۱۰۰ و ۲۵۰ ویدیو را نشان می دهیم. قابل ذکر است این ویدیو ها در فایل ارسالی قرار دارند.



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