# مستندات پروژه Reinforcement Learning

## محیط: LunarLander-v۳

*الگوریتم‌های پیاده‌سازی شده:*

*Q-Learning, SARSA, Approximate Q-Learning*

**محمدامین رضاپور**

**امیرمحمد کمیجانی**

# فهرست مطالب

# ۱. معرفی پروژه و اهداف

این پروژه یک پیاده‌سازی کامل از الگوریتم‌های یادگیری تقویتی (Reinforcement Learning) برای حل مسئله فرود سفینه فضایی بر روی ماه در محیط **LunarLander-v3** است. هدف اصلی این پروژه آموزش یک عامل هوشمند (Agent) است که بتواند سفینه فضایی را به‌صورت ایمن و با حداقل مصرف سوخت بر روی سطح ماه فرود آورد.

## اهداف پروژه

- پیاده‌سازی سه الگوریتم اصلی RL: Q-Learning، SARSA و Approximate Q-Learning
- آموزش عامل‌ها برای فرود ایمن سفینه فضایی
- مقایسه عملکرد الگوریتم‌های مختلف از طریق نمودارها و متریک‌ها
- ذخیره و بازیابی مدل‌های آموزش دیده برای استفاده مجدد

# ۲. مفاهیم پایه Reinforcement Learning

## State (حالت)

در محیط LunarLander، هر حالت (State) یک بردار ۸ بعدی است که شامل اطلاعات زیر می‌باشد:

| توضیحات | شاخص |
|---|---|
| موقعیت افقی (x-coordinate) | 0 |
| موقعیت عمودی (y-coordinate) | 1 |
| سرعت افقی (horizontal velocity) | 2 |
| سرعت عمودی (vertical velocity) | 3 |
| زاویه سفینه (angle) | 4 |
| سرعت زاویه‌ای (angular velocity) | 5 |
| پای چپ تماس با زمین (۰ یا ۱) | 6 |
| پای راست تماس با زمین (۰ یا ۱) | 7 |

## Action (عمل)

عامل می‌تواند یکی از ۴ عمل زیر را انتخاب کند:

| توضیحات | Action |
|---|---|
| هیچ کاری انجام نده (Do Nothing) | 0 |

| | |
|---|---|
| موتور چپ را روشن کن (Fire Left Engine) | 1 |
| موتور اصلی را روشن کن (Fire Main Engine) | 2 |
| موتور راست را روشن کن (Fire Right Engine) | 3 |

## Reward (پاداش)

پاداش در هر مرحله بر اساس عملکرد سفینه محاسبه می‌شود:

- فرود موفق: پاداش بین +۱۰۰ تا +۱۴۰
- سقوط یا خروج از محدوده: پاداش منفی -۱۰۰
- استفاده از موتور اصلی: جریمه -۰.۳
- استفاده از موتورهای کناری: جریمه -۰.۰۳
- هر پا که با زمین تماس داشته باشد: +۱۰

# ۳. ساختار پروژه و فایل‌های کد

## ۳.۱ فایل rlagents.py

این فایل حاوی پیاده‌سازی تمامی الگوریتم‌های یادگیری تقویتی و توابع کمکی است. شامل:

### کلاس‌ها و توابع اصلی:

- Qtabel: کلاس دیکشنری سفارشی برای ذخیره مقادیر Q
- Qlearning: کلاس اصلی الگوریتم Q-Learning
- SarsaQlearning: کلاس الگوریتم SARSA (وارث Qlearning)
- approximateQlearning: کلاس الگوریتم Approximate Q-Learning
- discretize_state(): تابع گسسته‌سازی حالت‌های پیوسته
- draw_۳D_chart() و draw_۲D_chart(): توابع رسم نمودار
- save_qtable() و load_qtable(): ذخیره و بارگذاری Q-table

## ۳.۲ فایل main.py

فایل اصلی برای اجرای پروژه. این فایل محیط را ایجاد کرده و الگوریتم مورد نظر را اجرا می‌کند.

```
env = gym.make("LunarLander-v3", render_mode="human") sarsaagent =
SarsaQlearning(10000, flag=True) sarsaagent.train(env)
```

## ۳.۳ فایل random_agent.py

یک عامل ساده که به‌صورت تصادفی عمل انتخاب می‌کند. برای مقایسه با عامل‌های هوشمند استفاده می‌شود.

# ۴. الگوریتم‌های پیاده‌سازی شده

## ۴.۱ الگوریتم Q-Learning

Q-Learning یک الگوریتم یادگیری تقویتی off-policy است که بهینه‌ترین سیاست را بدون نیاز به مدل محیط یاد می‌گیرد.

### فرمول به‌روزرسانی:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)]$$

### پارامترهای کلیدی:

- نرخ یادگیری ۰.۱ :(α) - تعیین می‌کند چقدر اطلاعات جدید جایگزین اطلاعات قدیمی شود
- ضریب تخفیف ۰.۹۹ :(γ) - اهمیت پاداش‌های آینده
- Epsilon (ε): کاهشی از ۱ به ۰ - احتمال انتخاب عمل تصادفی (exploration)

### ویژگی‌های خاص:

- استفاده از Q-table برای ذخیره مقادیر Q برای هر جفت (state, action)
- گسسته‌سازی فضای حالت پیوسته به حالت‌های گسسته
- استراتژی epsilon-greedy برای تعادل بین exploration و exploitation
- به‌روزرسانی بر اساس بهترین عمل ممکن در حالت بعدی (off-policy)

## ۴.۲ الگوریتم SARSA

SARSA (State-Action-Reward-State-Action) یک الگوریتم on-policy است که سیاست فعلی را بهبود می‌بخشد.

### فرمول به‌روزرسانی:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

### تفاوت با Q-Learning:

- Q-Learning: از بهترین عمل ممکن (max) استفاده می‌کند - off-policy
- SARSA: از عملی که واقعاً انتخاب شده استفاده می‌کند - on-policy
- SARSA محافظه‌کارتر است و در محیط‌های خطرناک عملکرد بهتری دارد
- Q-Learning معمولاً سریع‌تر همگرا می‌شود اما ممکن است ریسک‌پذیرتر باشد

## ۴.۳ الگوریتم Approximate Q-Learning

این الگوریتم به‌جای ذخیره تمام مقادیر Q در یک جدول، از یک تقریب خطی استفاده می‌کند.

### روش کار:

به‌جای ذخیره Q(s,a) برای هر حالت، مقدار Q را به‌صورت زیر تقریب می‌زند:

$$Q(s,a) = \Sigma\ w_i \times f_i(s) + w_9 \times a$$

### مزایا:

- حافظه کمتر: فقط ۹ وزن به‌جای میلیون‌ها ورودی Q-table
- تعمیم بهتر: می‌تواند برای حالت‌های دیده‌نشده نیز پیش‌بینی کند
- عدم نیاز به گسسته‌سازی: مستقیماً با حالت‌های پیوسته کار می‌کند
- همگرایی سریع‌تر در فضاهای حالت بزرگ

# ۵. نحوه اجرا و استفاده

## ۵.۱ نیازمندی‌های نصب

قبل از اجرای پروژه، کتابخانه‌های زیر را نصب کنید:

```
pip install gymnasium
pip install numpy
pip install matplotlib
pip install gymnasium[box2d]
```

## ۵.۲ حالت‌های مختلف اجرا

### حالت ۱: آموزش مدل جدید Q-Learning

```
env = gym.make("LunarLander-v3", render_mode="human") agent = Qlearning(10000)  # 10000 اپیزود agent.train(env) env.close()
```

### حالت ۲: آموزش مدل جدید SARSA

```
env = gym.make("LunarLander-v3", render_mode="human") sarsaagent = SarsaQlearning(10000) sarsaagent.train(env) env.close()
```

### حالت ۳: بارگذاری و استفاده از مدل ذخیره شده

```
env = gym.make("LunarLander-v3", render_mode="human") sarsaagent = SarsaQlearning(1000, flag=True)  # flag=True برای بارگذاری sarsaagent.train(env)  # ادامه آموزش یا ارزیابی env.close()
```

### حالت ۴: آموزش Approximate Q-Learning

```
env = gym.make("LunarLander-v3", render_mode="human") appxagent =
approximateQlearning(5000) appxagent.train(env) env.close()
```

### حالت ۵: آموزش بدون نمایش گرافیکی (سریع‌تر)

```
env = gym.make("LunarLander-v3")  # بـدون render_mode agent = Qlearning(10000)
agent.train(env) env.close()
```

### حالت ۶: اجرای عامل تصادفی (Baseline)

```
python random_agent.py
```

## ۵.۳ پارامترهای قابل تنظیم

| پارامتر | مقدار پیش‌فرض | توضیحات |
|---|---|---|
| **niteration** | 10000 | تعداد اپیزودهای آموزش |
| **alpha (α)** | 0.1 | نرخ یادگیری |
| **gamma (γ)** | 0.99 | ضریب تخفیف |
| **epsilon (ε)** | کـا هشی | احتمال exploration |
| **flag** | False | True برای بارگذاری مدل ذخیره شده |
| **render_mode** | human | نمایش گرافیکی یا None |

## ۵.۴ فرایند اجرا گام‌به‌گام

۱. ایجاد محیط: محیط LunarLander با تنظیمات مورد نظر ایجاد می‌شود
۲. ساخت عامل: یکی از کلاس‌های Qlearning، SarsaQlearning یا approximateQlearning نمونه‌سازی می‌شود
۳. شروع آموزش: متد train() فراخوانی می‌شود و حلقه آموزش شروع می‌گردد
۴. در هر اپیزود: عامل حالت را مشاهده کرده، عمل انتخاب می‌کند، پاداش دریافت کرده و Q-values را به‌روز می‌کند
۵. نمایش پیشرفت: هر ۱۰۰ اپیزود، شماره iteration چاپ می‌شود
۶. ذخیره نتایج: پس از اتمام، نمودارها رسم و ذخیره می‌شوند
۷. بستن محیط: محیط شبیه‌سازی بسته می‌شود

# ۶. خروجی‌ها و نمودارها

## ۶.۱ خروجی‌های کنسول

در طول اجرا، اطلاعات زیر در کنسول نمایش داده می‌شود:

- پاداش کل هر اپیزود (Total Reward): پس از پایان هر اپیزود
- شماره اپیزود: هر ۱۰۰ اپیزود یک بار
- پیام ذخیره‌سازی: پس از ذخیره Q-table یا وزن‌ها

## ۶.۲ فایل‌های خروجی

| فایل | توضیحات |
|---|---|
| `SavedQtable.txt` | Q-table ذخیره شده برای Q-Learning و SARSA |
| `SavedApproximateWeights.txt` | وزن‌های ذخیره شده برای Approximate Q-Learning |
| `QLearning_3D_chart.png` | نمودار سه‌بعدی Q-Learning |
| `QLearning_2D_chart.png` | نمودار دوبعدی Q-Learning |
| `SarsaQLearning_3D_chart.png` | نمودار سه‌بعدی SARSA |
| `SarsaQLearning_2D_chart.png` | نمودار دوبعدی SARSA |

## ۶.۳ نمودارها

### نمودار دوبعدی (۲D Chart)

- محور افقی (X): زمان (شماره اپیزود)
- محور عمودی (Y): پاداش کل دریافتی در هر اپیزود
- کاربرد: نمایش روند بهبود عملکرد عامل در طول زمان

### نمودار سه‌بعدی (3D Chart)

- محور X: زمان (شماره اپیزود)
- محور Y: پاداش کل
- محور Z: آخرین عمل انتخاب شده
- کاربرد: بررسی رابطه بین زمان، پاداش و اعمال انتخاب شده

## ۶.۴ تفسیر نتایج

نحوه ارزیابی عملکرد الگوریتم‌ها:

- پاداش مثبت و رو به رشد: نشان‌دهنده یادگیری موفق
- پاداش بالاتر از ۲۰۰: فرود بسیار موفق و بهینه
- کاهش نوسانات در نمودار: نشان‌دهنده ثبات در سیاست یادگرفته شده
- پاداش منفی مداوم: عدم یادگیری یا نیاز به تنظیم پارامترها

# ۷. جزئیات فنی پیاده‌سازی

## ۷.۱ گسسته‌سازی فضای حالت

چون Q-Learning و SARSA نیاز به فضای حالت گسسته دارند، تابع discretize_state() هر مؤلفه را به بازه‌های گسسته تبدیل می‌کند:

```
discrete_state = (     min(2, max(-2, int((state[0]) / 0.05))),     min(2, max(-2,
int((state[1]) / 0.1))),     ...     int(state[6]),     int(state[7]) )
```

- مقادیر پیوسته به بازه‌های گسسته [-۲, ۲] تقسیم می‌شوند
- ضرایب تقسیم (۰.۰۵, ۰.۱) برای کنترل دقت انتخاب شده‌اند

## ۷.۲ استراتژی Epsilon-Greedy

برای تعادل بین Exploration و Exploitation:

```
epsilon = (niteration - iternumber) / niteration if flipcoin(epsilon):     action =
random.choice(actions) else:     action = best_action
```

- در ابتدای آموزش: epsilon بالا ← بیشتر Explore می‌کند
- در انتهای آموزش: epsilon پایین ← بیشتر Exploit می‌کند

## ۷.۳ ذخیره‌سازی و بارگذاری مدل

برای استفاده مجدد از مدل‌های آموزش دیده:

- Q-table و وزن‌ها در فایل‌های متنی ذخیره می‌شوند
- با تنظیم flag=True در constructor، مدل قبلی بارگذاری می‌شود
- امکان ادامه آموزش یا استفاده برای ارزیابی فراهم است

## ۸. مقایسه الگوریتم‌ها

| ویژگی | Q-Learning | SARSA | Approximate |
|---|---|---|---|
| **نوع** | Off-Policy | On-Policy | Off-Policy |
| **حافظه** | بالا (Q-table) | بالا (Q-table) | کم (۹ وزن) |
| **سرعت همگرایی** | متوسط | کند | سریع |
| **تعمیم** | ضعیف | ضعیف | عالی |
| **ریسک‌پذیری** | بالا | پایین | متوسط |
| **دقت** | بالا (با گسسته‌سازی) | بالا (با گسسته‌سازی) | متوسط |
| **مناسب برای** | فضای کوچک | محیط خطرناک | فضای بزرگ |

### نتیجه‌گیری

انتخاب الگوریتم بستگی به نیازهای پروژه دارد:

- برای محیط‌های خطرناک و نیاز به امنیت بیشتر: SARSA
- برای بهینه‌سازی سریع و فضای کوچک: Q-Learning
- برای فضاهای بزرگ و نیاز به تعمیم: Approximate Q-Learning

## ۹. رفع مشکلات متداول

### مشکل ۱: محیط اجرا نمی‌شود

- راه‌حل: نصب کتابخانه Box۲D با دستور pip install gymnasium[box۲d]

### مشکل ۲: عامل یاد نمی‌گیرد

- افزایش تعداد اپیزودها (niteration را به ۲۰۰۰۰ یا بیشتر تغییر دهید)
- تنظیم نرخ یادگیری alpha (مقادیر بین ۰.۰۵ تا ۰.۳ را امتحان کنید)
- بررسی گسسته‌سازی (ضرایب تقسیم را تغییر دهید)

### مشکل ۳: خطای حافظه

- استفاده از Approximate Q-Learning به‌جای Q-Learning/SARSA
- افزایش ضرایب گسسته‌سازی برای کاهش تعداد حالت‌ها

### مشکل ۴: نمودارها ذخیره نمی‌شوند

- اطمینان از نصب matplotlib
- بررسی دسترسی نوشتن در پوشه اجرا

### مشکل ۵: اجرا بسیار کند است

- حذف render_mode="human" برای افزایش سرعت

- کاهش تعداد اپیزودها برای تست اولیه
- استفاده از Approximate Q-Learning که معمولاً سریع‌تر همگرا می‌شود

## ۱۰. خلاصه و جمع‌بندی

این پروژه یک پیاده‌سازی کامل و جامع از الگوریتم‌های یادگیری تقویتی است که شامل موارد زیر می‌باشد:

- سه الگوریتم اصلی: SARSA (On-Policy)، Q-Learning (Off-Policy)، و Approximate Q-Learning (Function Approximation)
- پیاده‌سازی کامل مفاهیم: State، Action، Reward، Q-Value، Policy
- قابلیت ذخیره و بارگذاری مدل‌های آموزش دیده
- تولید نمودارهای دوبعدی و سه‌بعدی برای تحلیل عملکرد
- حالت‌های مختلف اجرا برای آموزش، ارزیابی و مقایسه

### نکات کلیدی برای موفقیت

۸. صبور باشید: یادگیری تقویتی زمان‌بر است و ممکن است هزاران اپیزود طول بکشد

۹. پارامترها را تنظیم کنید: alpha، gamma و epsilon تأثیر زیادی بر سرعت و کیفیت یادگیری دارند

۱۰. الگوریتم مناسب را انتخاب کنید: بسته به محدودیت‌های حافظه، سرعت و دقت مورد نیاز

۱۱. از نمودارها استفاده کنید: برای درک بهتر روند یادگیری و شناسایی مشکلات

۱۲. مدل‌ها را ذخیره کنید: برای استفاده مجدد و جلوگیری از اتلاف زمان آموزش

---

## پایان مستندات

*موفق باشید!*