



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی صنایع و سیستم‌های مدیریت

پایان‌نامه کارشناسی

رشته مهندسی صنایع

معاملات الگوریتمی در بازارهای مالی با استفاده از روش‌های یادگیری
تقویتی عمیق

نگارش

ابراهیم پیچکا

استاد راهنما

دکتر مسعود ماهوتچی

شهریور 1401



به نام خدا

تعهدنامه اصالت اثر

تاریخ:

اینجانب ابراهیم پیچکا متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

ابراهیم پیچکا

امضا

تقدیر و تشکر

از زحمات استار گرانقدر، دکتر ماهوتچی، که با راهنمایی ها رهنمودهای ارزشمند خود در مراحل مختلف انجام این پروژه، مرا یاری نمودند، بسیار سپاسگزارم. همچنین از خانواده و دوستان گرامی که در تمامی مراحل زندگی همراه و همیار اینجانب بوده اند تشکر و تقدیر می نمایم.

چکیده

امروزه با افزایش سرعت و قدرت منابع محاسباتی و پردازشی، معاملات الگوریتمی در بازارهای مالی به یک عنصر اصلی در مالی کمی^۱ تبدیل شده است. استراتژی های موفق برای معاملات خودکار سهام برای صندوق های پوشش ریسک و شرکت های سرمایه گذاری ضروری است. این استراتژی ها شامل بهینه سازی تخصیص سرمایه و به حداکثر رساندن بازدهی سرمایه گذاری است. در معاملات الگوریتمی، استخراج ویژگی و طراحی استراتژی معاملاتی دو چالش برجسته برای کسب سود بلندمدت هستند.

در مالی کمی، معاملات سهام اساساً مسئله ای از جنس تصمیم گیری پویا است، یعنی تصمیم گیری در مورد اینکه کجا، با چه قیمتی و چه مقداری در یک بازار سهام بسیار تصادفی، پویا و پیچیده معامله شود. با پیشرفت های اخیر متدهای یادگیری تقویتی عمیق (DRL)^۲، مسائل تصمیم گیری پویای متوالی را می توان با رویکردی شبیه به انسان مدل سازی و حل کرد.

در این پژوهش پتانسیل و عملکرد یادگیری تقویتی عمیق را برای بهینه سازی استراتژی معاملات سهام و در نتیجه به حداکثر رساندن بازده سرمایه گذاری بررسی می کنیم. سهام شرکت گوگل به عنوان سهام معاملاتی ما انتخاب شده و قیمت روزانه بازگشایی و پایانی به همراه حجم معاملات و چند شاخص تکنیکال به عنوان محیط آموزشی و بازار معاملات مورد استفاده قرار می گیرد.

در این پژوهش به صورت جداگانه دو روش معاملاتی بر اساس یادگیری تقویتی عمیق ارائه شده تا به طور مستقل تصمیمات معاملاتی اتخاذ و در بازارهای مالی پویا بازدهی ایجاد کند. عملکرد این عامل های هوشمند با عملکرد استراتژی خرید و نگهداری مقایسه می گردد. و در انتها نشان داده می شود رویکرد یادگیری تقویتی عمیق پیشنهادی از روش پایه خرید و نگهداری از نظر معیارهای ارزیابی ریسک و بازده سبب بهتر عمل می کند.

¹ Quantitative Finance

² Deep Reinforcement Learning

واژه‌های کلیدی:

یادگیری ماشین، یادگیری تقویتی، معاملات الگوریتمی، شبکه‌های عصبی مصنوعی

صفحه	فهرست مطالب
1	تقدیر و تشکر.....
أ	چکیده.....
1	1- فصل اول مقدمه.....
2	مقدمه.....
2	1-1- بیان مسئله.....
4	1-2- کاربرد و اهمیت تحقیق.....
4	1-3- اهداف تحقیق.....
6	1-4- روش تحقیق.....
7	2- فصل دوم پیشینه‌ی تحقیق و مفاهیم.....
8	پیشینه‌ی تحقیق و مفاهیم.....
8	1-2- فرآیند تصمیم‌گیری مارکوف.....
10	1-2-1- پاداش و بازده.....
12	1-2-2- توابع ارزش.....
13	2-2- یادگیری تقویتی.....
14	1-2-2- یادگیری تقویتی عمیق.....
20	2-2-2- یادگیری تقویتی مبتنی بر تابع ارزش (روش منتقد).....
22	2-2-3- یادگیری تقویتی مبتنی بر تابع سیاست (روش عملگر).....
23	2-2-4- روش عملگر-منتقد (Actor-Critic).....
25	2-3- معاملات کمی الگوریتمی.....
28	3- فصل سوم مدل‌سازی و روند پژوهش.....
29	مدل‌سازی و روند پژوهش.....
29	1-3- محیط، فضای حالت و داده‌ها.....
32	2-3- فضای عمل.....
33	3-3- الگوریتم‌ها و معماری شبکه‌های عصبی عاملها.....
33	1-3-3- Vanilla Policy Gradient (PG).....
36	2-3-3- Proximal Policy Optimization (PPO).....
38	3-3-3- Deep Q-Learning (DQL).....

38	3-4- مکانیزم پاداش
39	3-5- فرآیند آموزش عامل
41	4- فصل چهارم ارزیابی عملکرد
42	ارزیابی عملکرد
42	4-1- معیارهای سنجش
46	4-2- نتایج ارزیابی
48	4-2-1- بررسی عملکرد عامل PPO
52	4-2-2- بررسی عملکرد عامل DQL
56	4-2-3- مقایسه‌ی عملکرد استراتژی‌ها
60	5- فصل پنجم نتیجه‌گیری و پیشنهادات آتی
61	نتیجه‌گیری و پیشنهادات آتی
61	5-1- پیشنهادات آتی
63	منابع و مراجع
68	پیوست‌ها
83	Abstract

صفحه

فهرست اشکال

- شکل 1-1-2: دیاگرام تعامل عامل با محیط در فرآیند تصمیم مارکوف 9
- شکل 2-1-2: نمونه ای از یک MDP ساده با سه حالت (دایره های سبز) و دو عمل (دایره های نارنجی)، با دو مقدار پاداش (فلش های نارنجی). 10
- شکل 2-2-1: نمایش محاسبات یک واحد (نورون) در یک شبکه عصبی. 15
- شکل 2-2-2: معمول ترین توابع فعالسازی به همراه مشتق های آنها. 16
- شکل 2-2-3: شمای کلی از معماری شبکه های پرسپترون چندلایه (MLP). 18
- شکل 2-2-4: شمای کلی از نحوه استفاده از شبکه عصبی در یادگیری تقویتی 19
- شکل 2-2-5: نقشه کلی الگوریتم های یادگیری تقویتی عمیق بر اساس نوع تابع تقرب زده شده با شبکه های عصبی. 20
- شکل 2-2-6: نحوه استفاده از شبکه های عصبی برای تقرب تابع Q در روش های بر پایه منتقد. 21
- شکل 2-2-7: دو نحوه اصلی پیاده سازی شبکه Q. 22
- شکل 2-2-8: نحوه استفاده از شبکه های عصبی برای تقرب تابع سیاست در روش های بر پایه عملگر. 23
- شکل 2-2-9: دیاگرام یک سیستم یادگیری تقویتی ACTOR-CRITIC. 24
- شکل 2-2-10: نمودار ون روش های یادگیری تقویتی عمیق 25
- شکل 2-2-11: نحوه تقرب دو تابع ارزش و سیاست در روش های ACTOR-CRITIC با استفاده از شبکه عصبی 25
- شکل 2-3-1: سیکل ایجاد و نگهداری یک سیستم معاملاتی هوشمند کمی. (ALPHABOT, N.D.). 26
- شکل 2-3-2: اجزای اصلی یک سیستم معاملاتی کمی خودکار. (ALPHABOT, N.D.). 27
- شکل 3-1-2: تحلیل همبستگی ویژگی های مورد بررسی. 30
- شکل 3-3-1: الگوریتم آموزش یک عامل با روش گرادین سیاست ساده. (OPENAI, VANILLA POLICY GRADIENT, N.D.). 35
- شکل 3-3-2: دیاگرام نحوه تعامل و آموزش در روش گرادین سیاست. (HUI, 2018). 35
- شکل 3-3-3: محدودسازی تابع هزینه در الگوریتم PPO. (SCHULMAN, WOLSKI, DHARIWA, RADFORD, & KLIMOV, 2017). 37
- شکل 3-3-4: آموزش عامل با الگوریتم PPO. (OPENAI, N.D.). 37
- شکل 4-2-1: عملکرد و بازده تجمعی عامل PPO در بازه ی تست. 49
- شکل 4-2-2: تصمیمات معاملاتی PPO با توجه به قیمت سهام در یک تکرار از مرحله ی ارزیابی. 50
- شکل 4-2-3: موقعیت های معاملاتی اتخاذ شده توسط PPO در یک تکرار از مرحله ی ارزیابی. 50
- شکل 4-2-4: توزیع ارزش موقعیت های اتخاذ شده توسط عامل PPO در یک تکرار از ارزیابی. 51
- شکل 4-2-5: توزیع بازده روزانه عامل PPO در مرحله ی ارزیابی. 52
- شکل 4-2-6: عملکرد و بازده تجمعی عامل DQL در بازه ی تست. 53
- شکل 4-2-7: مصورسازی تصمیمات معاملاتی DQL با توجه به قیمت سهام در یک تکرار از مرحله ی ارزیابی. 54
- شکل 4-2-8: ارزش موقعیت های اتخاذ شده توسط DQL در یک تکرار از مرحله ی ارزیابی. 54

- شکل 4-2-9: توزیع و هیستوگرام ارزش موقعیت‌های اتخاذ شده توسط عامل DQL در یک تکرار از ارزیابی. 55
- شکل 4-2-10: توزیع بازده روزانه عامل DQL در مرحله‌ی ارزیابی. 56
- شکل 4-2-11: مقایسه توزیع بازده روزانه عامل‌های PPO و DQL و استراتژی خرید و نگهداری در مرحله‌ی ارزیابی. 57
- شکل 4-2-12: مقایسه نمودار جعبه‌ای بازده روزانه عامل‌های PPO و DQL و استراتژی خرید و نگهداری در مرحله‌ی ارزیابی. 57
- شکل 4-2-13: مقایسه‌ی میانگین عملکرد هر سه استراتژی در کنارهم. 58
- شکل 4-2-14: مقایسه‌ی بدترین سناریو (صدک پنجم) عملکرد هر دو عامل در کنارهم. 58
- شکل 4-2-15: مقایسه‌ی بدترین سناریو (صدک بیستم) عملکرد هر دو عامل در کنارهم. 59

صفحه

فهرست جداول

جدول 1-3	مقادیر انتخاب شده برای هایپرپارامترهای مدل‌ها.	39
جدول 1-4	نتایج بکتست الگوریتم‌ها در بازه‌ی تست.	47
جدول 2-4	مقایسه‌ی نتایج مدل‌ها با عملکرد پایه‌ی خرید و نگهداری.	48

فهرست علائم

علائم لاتین	حالت‌ها
$s \in \mathcal{S}$	حالت‌ها
$a \in \mathcal{A}$	عمل‌ها
$r \in \mathcal{R}$	پاداش لحظه‌ای
S_t, A_t, R_t	حالت، عمل، و پاداش در مرحله زمانی t از یک مسیر از تعاملات (همچنین به صورت s_t, a_t, r_t نیز نمایش داده می‌شوند).
G_t	بازده؛ یا پاداش تجمعی همراه یا بدون تنزیل؛ $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$.
$V(s)$	تابع ارزش حالت؛ بازده مورد انتظار حالت s را اندازه‌گیری می‌کند. $V_{\phi}(\cdot)$ تابع ارزشی است که با ϕ پارامتری و تقریب زده شده است.
$V^{\pi}(s)$	مقدار حالت s وقتی از خط مشی π پیروی می‌کنیم؛ $V^{\pi}(s) = \mathbb{E}_{a \sim \pi}[G_t \mid s_t = s]$.
$Q(s, a)$	تابع ارزش حالت-عمل مشابه $V(s)$ است، اما بازگشت مورد انتظار یک جفت حالت و عمل (s, a) را ارزیابی می‌کند. $Q_w(\cdot)$ یک تابع مقدار عمل است که با w پارامتری شده است.
$Q^{\pi}(s, a)$	مشابه V^{π} ، ارزش جفت حالت-عمل زمانی که از سیاست مشخص π پیروی می‌کنیم؛ $Q^{\pi}(s, a) = \mathbb{E}_{a \sim \pi}[G_t \mid s_t = s, a_t = a]$.
$A(s, a)$	تابع مزیت، $A(s, a) = Q(s, a) - V(s)$ ؛ می‌توان آن را به عنوان نسخه دیگری از Q با واریانس کمتر با حذف مقدار حالت به عنوان خط پایه در نظر گرفت.
L	تابع هزینه؛ تابع هدف مورد بهینه‌سازی که نشان‌دهنده‌ی مقدار اختلاف خروجی تابع برآوردگر با مقدار حاصل شده حاصل از تعامل یا محیط است.

علائم یونانی

γ	عامل تنزیل؛ ضریبی برای دخیل کردن عدم قطعیت پاداش های آینده و کمک یه همگرایی پاداش های تجمعی؛ $0 < \gamma \leq 1$.
$\pi(a s)$	تابع سیاست تصادفی (استراتژی رفتار عامل)؛ $\pi_\theta(\cdot)$ تابع سیاستی است که با θ پارامتری و تقریب زده شده است.
$\mathcal{T}(s', r s, a)$	احتمال انتقال به حالت بعدی s' از حالت فعلی s با عمل a و پاداش r .
∇_θ	گرادیان بر حسب θ

زیرنویس ها

t	مرحله ی زمانی از مسیر تعامل عامل و محیط.
-----	--

1- فصل اول

مقدمه

مقدمه

معاملات مالی موضوعی است که امروزه به طور گسترده ای مورد پژوهش قرار گرفته است و روش های مختلفی برای این گونه معاملات در عرصه ی آکادمیک ارائه شده است. از جمله این روش ها می توان به تحلیل بنیادی (Graham, Dodd, & Cottle, 1934)، تحلیل تکنیکال (Murphy, 1999) و معاملات الگوریتمی (Chan, 2009) اشاره کرد. در واقع، بسیاری از متخصصان حوزه مالی از ترکیبی از این تکنیک ها برای انجام معاملات استفاده می کنند. از این بین معاملات الگوریتمی اخیراً بیشترین توجه را به سمت خود جلب کرده به طوری که حدود 75٪ از حجم معاملات در بورس های ایالات متحده را تشکیل می دهد (Chan, 2009). از نقاط قوت معاملات الگوریتمی می توان به منابع محاسباتی قوی، اجرای سریع تر و کاهش و توزیع ریسک اشاره کرد. یکی از اجزای کلیدی چنین سیستم معاملاتی، سیگنال های پیش بینی است که می تواند منجر به ایجاد بازده اضافی معنادار نسبت به بازدهی بازار (که در ادبیات مالب به عنوان آلفا شناخته می شود و برای ارزیابی عملکرد مدیران سرمایه استفاده می شود) شود و برای این منظور، روش های ریاضی و آماری ای به طور گسترده اعمال می شود. با این حال، به دلیل پایین بودن نسبت سیگنال به نویز داده های مالی (به معنی تعداد مقادیر داده ی مفید در مقابل مقادیر داده ی مزاحم) و ماهیت پویای بازارها، طراحی این روش ها بی اثر است و اثربخشی سیگنال های به دست آمده در طول زمان متفاوت است و کارایی خود را از دست می دهد. در نتیجه به رویکرد های موثرتر و کاراتری در طراحی این سیستم های معاملاتی نیاز است.

1-1- بیان مسئله

در سال های اخیر، الگوریتم های یادگیری ماشین در بسیاری از زمینه ها با موفقیت های قابل توجهی در حوزه های کاربردی متنوع از جمله طبقه بندی تصاویر (Krizhevsky, Sutskever, & Hinton, 2012) و پردازش زبان طبیعی (Collobert, et al., 2011) محبوبیت زیادی به دست آورده اند. تکنیک های مشابهی نیز در بازارهای مالی در تلاش برای یافتن بازده اضافی معنادار بیشتر نسبت به بازدهی بازار به کار رفته است (برای چند نمونه با استفاده از چنین تکنیک هایی در زمینه داده های با بسامد بالا، به (Zhang, Zohren, & Roberts, 2019)، (Tsantekidis, et al., 2017) و (Sirignano & Cont, 2019) مراجعه شود). بیشتر پژوهش های انجام شده در این حوزه بر روی متدهای رگرسیون و طبقه بندی

تمرکز می‌کنند که در آن بازده اضافی، قیمت یا حرکت بازار در برخی افق‌های زمانی (ثابت) پیش‌بینی می‌شود. با این حال، بحث کمی در مورد تبدیل این پیش‌بینی‌ها به موقعیت‌های خرید یا فروش معاملاتی واقعی وجود دارد (برای مشاهده یک نمونه تلاش در جهت آموزش یک مدل یادگیری عمیق برای یادگیری مستقیم موقعیت‌ها، به (Lim, Zohren, & Roberts, 2019) مراجعه شود). در واقع، چنین نگرانی از اطلاعات بازار به موقعیت‌های معاملاتی در افق ثابت بی‌اهمیت است. به عنوان مثال، افق‌های پیش‌بینی اغلب نسبتاً کوتاه هستند (در صورت استفاده از داده‌های روزانه یک روز یا چند روز جلوتر)، با این حال، روندهای بزرگ می‌توانند برای هفته‌ها یا ماه‌ها با برخی دوره‌های تثبیت متوسط ادامه داشته باشند. بنابراین ما تنها به سیگنالی با قدرت پیش‌بینی خوب نیاز داریم، بلکه به سیگنالی نیز نیاز داریم که بتواند به طور مداوم موقعیت‌های معاملاتی جهت‌دار مناسبی در نظر بگیرد.

با توجه به موارد ذکر شده، تصمیم‌گیری در رابطه با گرفتن موقعیت‌های معاملاتی مناسب در محیط بازارهای مالی را می‌توان به صورت یک مسئله‌ی تصمیم‌گیری متوالی^۳، و به تبع، تحت شرایط فرآیندهای تصمیم مارکوف (MDP)^۴ (Bellman, 1957) بررسی نمود. در این پژوهش، ما الگوریتم‌های یادگیری تقویتی (RL) (Sutton & Barto, 2018) را که ابزاری قوی برای استفاده در برخورد با شرایط MDP است، در کنار متدهای یادگیری عمیق، برای حل این مسئله با شرایط و مشکلات فوق مورد بررسی و پیاده‌سازی قرار می‌دهیم.

در این پروژه، ما سهام شرکت گوگل با نماد GOOG در بازار سهام ایالات متحده را به عنوان سهام مورد مطالعه و آزمایش در نظر گرفتیم. داده‌های استفاده‌شده در این پروژه داده‌های قیمتی در چهارچوب زمانی روزانه، شامل قیمت بازگشایی، پایانی و حجم معاملات در هر روز است. محیط بازار در این مسئله دوطرفه در نظر گرفته شده، به این معنی که امکان فروش استقراضی سهام نیز وجود دارد و از امکان کسب بازدهی از کاهش قیمت نیز برقرار است.

فرض ما بر این است، که کلیه‌ی سفارشات، در ابتدای هر روز و با قیمت بازگشایی همان روز گذاشته می‌شود و با توجه به این که سهام انتخاب شده جزء سهام شناور و دارای حجم معاملات بالا در بازار

³ Sequential Decision-making

⁴ Markov Decision Processes

مربوطه است، فرض بر این است که عامل با مشکل شناوری در معاملات مواجه نخواهد شد. برای ساده سازی مسئله هزینه‌ی کارمزد معاملات برابر صفر در نظر گرفته شده، هرچند امکان در نظر گرفتن این عامل در پژوهش‌های آتی برای نزدیک‌تر کردن شرایط به شرایط و محدودیت‌های واقعی وجود دارد. همچنین و بازده دارایی نقد نیز در بازار ایالات متحده در زمان این پژوهش حدود 3.5٪ سالانه بوده که در برای ساده سازی در مرحله‌ی آموزش مدل برابر صفر در نظر گرفته شده اما در مرحله‌ی ارزیابی برای محاسبه‌ی معیارهایی مانند نسبت شارپ و سورینتو لحاظ شده است. و عمل‌ها به صورت تک‌دوره‌ای اتخاذ می‌شوند. همچنین فرض عدم امکان فروش اعماری از سهام نیز رعایت شده و تمام سفارشات بر روی تعداد گسسته‌ای از سهام انجام می‌شود.

2-1- کاربرد و اهمیت تحقیق

معاملات الگوریتمی یک موضوع پراهمیت در بازارهای مالی است و به طور گسترده در هوش مصنوعی مدرن مورد بحث قرار گرفته است. هم از سوی سرمایه‌گذاران شرکتی و هم از سوی سرمایه‌گذاران خرد، تقاضای بسیار زیادی برای الگوریتم‌های معاملاتی خودکار سودده و سازگار با بازارهای مالی پویا وجود دارد. از این رو دستیابی به یک سیستم معاملاتی خودکار که چالش‌های سیستم‌های معاملاتی پیشین را پشت سر بگذارد می‌تواند دارای ارزش بسیار زیادی باشد و با تقاضای قابل توجهی از سوی فعالان حوزه بازارهای مالی روبه‌رو شود.

روش‌های اصلی برای معاملات مبتنی بر هوش مصنوعی چالش‌های دیرینه‌ای به شرح زیر دارند: (1) دشواری در استخراج شاخصه‌ها و بازنمایی‌های مؤثر از بازار، و (2) تفاوت بین طبقه‌بندی (پیش‌بینی جهت حرکت بازار) و یادگیری مستقیم استراتژی‌های معاملاتی (معامله مستقیم). که در این پژوهش سعی بر این داریم که با استفاده از متدهای یادگیری تقویتی عمیق به این چالش‌ها پاسخ دهیم.

3-1- اهداف تحقیق

اخیراً، یادگیری تقویتی عمیق به موفقیت‌های چشمگیری در حل مسائل پیچیده تصمیم‌گیری متوالی دست یافته است (Mnih, et al., 2015)، (Silver, et al., 2016). مزیت ذاتی الگوریتم‌های یادگیری تقویتی (RL) (Sutton & Barto, 2018) یادگیری مستقیم یک استراتژی عملی، در فرآیند تعامل یک

عامل^۵ هوشمند با یک محیط^۶ پویا است. به طور خاص، رویکرد RL به صورت "در لحظه" کار می کند، به طوری که یک محیط ناشناخته را اکتشاف می کند و همزمان سعی در اتخاذ تصمیم (عمل^۷) بهینه در هر پله‌ی زمانی خاص دارد به طوری که پاداشی که از محیط دریافت خواهد کرد را حداکثر سازد. قابلیت بهینه‌سازی تابع سیاست^۸ در طول زمان از طریق خودآموزی، استفاده از رویکرد RL را ذاتاً برای استراتژی معاملات الگوریتمی مناسب می‌کند. در مرجع (Deng, Bao, Kong, Ren, & Dai, 2017) یادگیری تقویتی مستقیم عمیق، برای دریافت سیگنال مالی و اجرای معاملات ارائه شده است. گرچه در آن از معماری‌های محبوب و پیشرفته‌ای مانند DQN^۹ (Mnih, et al., 2015) (که از روش‌های موسوم به یادگیری تقویتی بر پایه ارزش می‌باشد) و شبکه A3C^{۱۰} (Mnih, et al., 2016) (که از متدهای موسوم به Actor-Critic می‌باشد) استفاده نمی‌شود، اما به طور قابل ملاحظه‌ای از روش‌های RL در مسائل مختلف کنترل بهتر عمل می‌کند. مهم‌تر اینکه، در مقایسه با مسائل رایج RL، چالش دیگری وجود دارد که طراحی یک سیستم DRL برای معاملات را بسیار دشوارتر می‌کند. و آن این است که به منظور کاربردی‌تر کردن مدل ارائه شده، وضعیت بازار (states)، عمل‌های معاملاتی (actions)، تابع پاداش یا بازخورد (reward function) و مدیریت موقعیت‌های معاملاتی باید به طور جدی مورد توجه قرار گیرد.

در این پژوهش، برای پرداختن به چالش‌ها و مسائل فوق، یک چارچوب یادگیری تقویتی عمیق جدید برای معاملات الگوریتمی عملی پیشنهاد می‌کنیم که قادر به معامله خودکار در بازارهای مالی است. مدل پیشنهادی از دو جزء اصلی محیط و عامل تشکیل شده است. محیط، داده‌های تاریخی بازار را مدیریت می‌کند و داده‌های جدید را از منابع اطلاعاتی دریافت می‌کند. عامل، از یک ماژول پیش پردازش داده و یک عامل معامله‌گر تشکیل شده است که توسط الگوریتم‌های DRL (مبتنی بر Actor-Critic) با یک حالت، عمل، پاداش و ساختار شبکه‌ی طراحی شده و آموزش داده شده است.

⁵ Agent

⁶ Environment

⁷ Action

⁸ Policy Function

⁹ Deep Q Network

¹⁰ Asynchronous Advantage Actor Critic

4-1- روش تحقیق

در این پژوهش ابتدا داده‌های روزانه‌ی بازگشایی و بسته‌شدن قیمت‌های تاریخی سهام مورد نظر برای آموزش عامل را از منابع رسمی دریافت، پاکسازی و پیش‌پردازش کرده و به دو قسمت داده‌های آموزش و داده‌های ارزیابی تقسیم می‌کنیم. سپس محیط شبیه‌ساز تولید داده‌های روزانه‌ی سهام را برای تعامل عامل طراحی و مکانیزم پاداش لحظه‌ای برای این تعامل را تعیین می‌کنیم. بعد از طراحی محیط، مدل‌های مورد مطالعه را مشخص کرده و سپس عامل‌های مختلفی را با این مدل‌ها طراحی کرده و با تعامل با محیط برای تعداد مشخصی از تعاملات آموزش می‌دهیم. در مرحله‌ی آموزش فضای هایپرپارامترهای مدل‌های تعیین شده را نیز جست‌وجو و بهترین مقدار ممکن را انتخاب می‌کنیم. در نهایت، برای ارزیابی مدل‌ها، عملکرد عامل‌ها را با عملکردهای پایه‌ی تعیین شده در داده‌های تعیین شده برای داده‌های ارزیابی مقایسه کرده و شاخصه‌ی های عملکردی آن‌ها را بررسی می‌کنیم.

2- فصل دوم

پیشینه‌ی تحقیق و مفاهیم

پیشینه‌ی تحقیق و مفاهیم

در این بخش به مرور مفاهیم اصلی و پژوهش‌های پیشین انجام شده حول موضوع این پروژه می‌پردازیم. ابتدا به معرفی فرآیند تصمیم‌گیری مارکوف^{۱۱} (MDP)، که پایه‌ی اصلی مفاهیم یادگیری تقویتی است، پرداخته و پس از آن، به اختصار، یادگیری تقویتی عمیق، یادگیری تقویتی ارزش‌محور^{۱۲} و یادگیری تقویتی مبتنی بر سیاست^{۱۳} و روش‌های ترکیبی یادگیری تقویتی (AC)^{۱۴} را معرفی می‌کنیم. نهایتاً به چگونگی ایفای نقش یادگیری تقویتی عمیق و نحوه‌ی کاربرد آن در معاملات الگوریتمی می‌پردازیم.

2-1- فرآیند تصمیم‌گیری مارکوف

یادگیری تقویتی (Sutton & Barto, 2018) را می‌توان به عنوان فرآیندی در نظر گرفت که یک عامل هوشمند با تعامل پیوسته و متوالی با یک محیط ناشناخته، یاد می‌گیرد تا اعمال خود را برای تعامل با این محیط تنظیم و بهینه کند. در این مسئله یک محیط ناشناخته عموماً به صورت یک فرآیند تصمیم‌گیری مارکوف (Bellman, 1957) در نظر گرفته می‌شود که با یک چندتایی از مجموعه‌های $M = (\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma)$ فرمول‌سازی و مشخص می‌شود، که به ترتیب، \mathcal{S} مجموعه حالات محیط، \mathcal{A} مجموعه عمل‌ها، T تابع احتمال انتقال حالت، \mathcal{R} تابع پاداش و γ نرخ تنزیل پاداش می‌باشند، که همگی در ادامه به تفصیل شرح داده می‌شوند. در این تعریف، فرض بر این است که محیط دارای ویژگی مارکوف می‌باشد، به این معنی که در هر مرحله از تعامل عامل و محیط $t = 1, 2, \dots, n$ ، انتقال به حالت بعدی s_{t+1} فقط مشروط و وابسته به حالت فعلی s_t و عمل اتخاذ شده در آن حالت a_t است. به طور خاص، پس از اینکه عامل، عمل $a_t \in \mathcal{A}$ را انجام دهد و پاداش $r_t \in \mathcal{R}$ را دریافت کند، محیط با توجه به تابع احتمال انتقال حالت $T(s_{t+1}|s_t, a_t)$ از حالت $s_t \in \mathcal{S}$ به حالت $s_{t+1} \in \mathcal{S}$ گذار می‌کند. همچنین

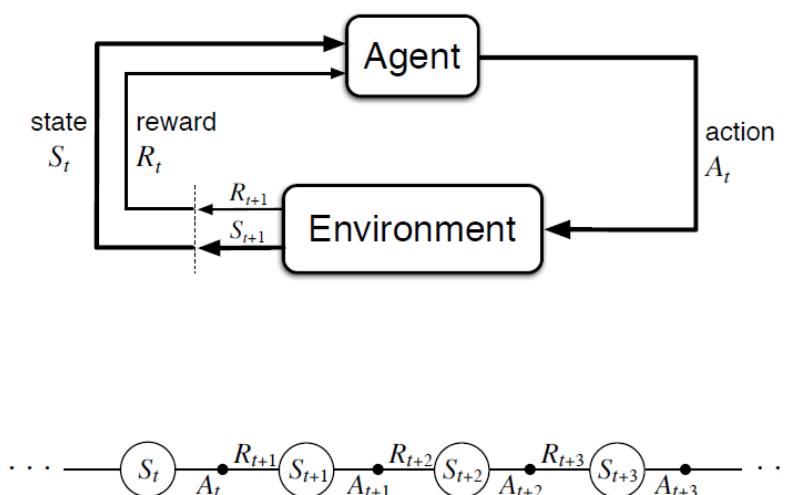
¹¹ Markov Decision Process

¹² Value-based

¹³ Policy-based

¹⁴ Actor-Critic Methods

در این چارچوب، مفهوم بازده نیز به صورت حاصل جمع پاداش های تنزیل شده‌ی حالت فعلی و تمامی حالات پیش رو با نرخ تنزیل $\gamma \in (0,1]$ تعریف می‌شود.



شکل 1-1-2: دیاگرام تعامل عامل با محیط در فرآیند تصمیم مارکوف

با این حال، منطقی نیست که عامل بتواند به طور کامل به حالت های محیط در دنیای واقعی دسترسی داشته باشد، به این معنی که ویژگی مارکوف به ندرت برای محیط برقرار می‌شود. در یک حالت جامع‌تر، فرآیند تصمیم‌گیری مارکوف نیمه‌مشهود¹⁵ (POMDP) (Williams & Young, 2007)، می‌تواند پویایی بسیاری از محیط‌های دنیای واقعی را با اذعان این موضوع که عامل فقط اشراف جزئی به حالت فعلی محیط دارد، به تصویر بکشد. به طور رسمی، یک POMDP با چندتایی $(S, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O})$ توصیف می‌شود. تفاوت آن با MDP این است که عامل به جای حالت واقعی $s \in S$ یک مشاهده o دریافت می‌کند. مشاهده‌ی o از حالت فعلی سیستم مطابق با توزیع احتمال $\mathcal{O}(s) = P(o | s)$ تولید می‌شود.

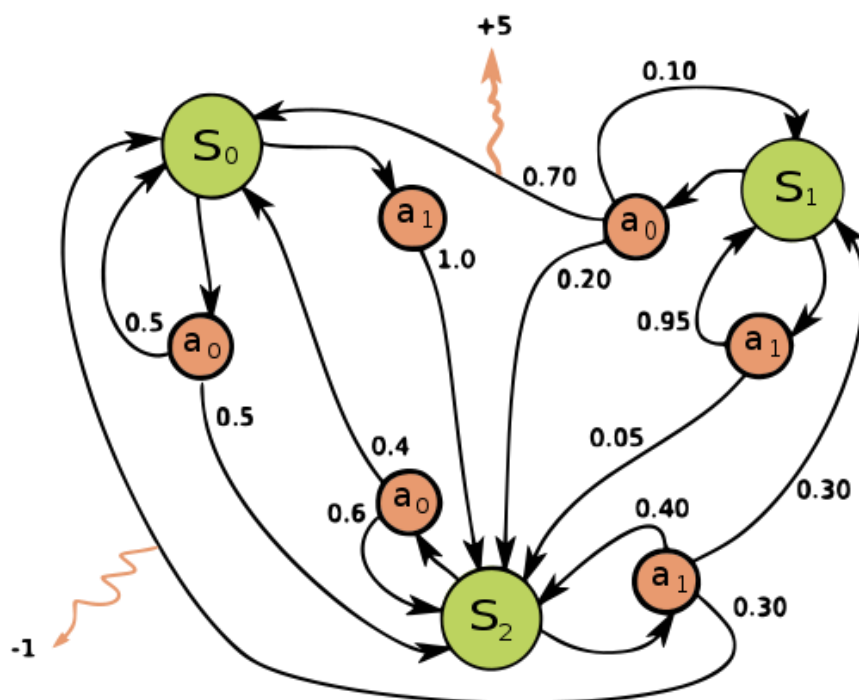
¹⁵ Partially-observable Markov Decision Process

2-1-1- پاداش و بازده

تابع پاداش R یکی از مهم‌ترین مفاهیم در یادگیری تقویتی است. خروجی این تابع که یک مقدار عددی به عنوان پاداش دریافت شده توسط عامل در یک مرحله‌ی زمانی است، به حالت فعلی محیط، عملی که اخیراً انجام شده و حالت بعدی محیط بستگی دارد:

$$r_t = R(s_t, a_t, s_{t+1})$$

اگرچه این رابطه اغلب به صورتی ساده می‌شود که فقط به حالت فعلی محیط، $r_t = R(s_t)$ یا جفت حالت-عمل $r_t = R(s_t, a_t)$ وابسته باشد.



شکل 2-1-2: نمونه‌ای از یک MDP ساده با سه حالت (دایره‌های سبز) و دو عمل (دایره‌های نارنجی)، با دو مقدار پاداش (فلش‌های نارنجی).

هدف عامل، به حداکثر رساندن پاداش انباشته در طول یک مسیر از تعاملات $\tau = (s_0, a_0, s_1, a_1, \dots)$ است، که به آن بازده $G(\tau)$ گفته می‌شود. به طور کلی، یک نوع از بازده، بازده تنزیل نشده افق محدود^{۱۶} است، که فقط مجموع پاداش‌های به دست آمده در یک تعداد ثابت و محدود از مراحل است:

$$G(\tau) = \sum_{t=0}^T r_t \quad (1-2)$$

نوع دیگر بازده، بازده تنزیل شده افق نامتناهی^{۱۷} است، که مجموع تمام پاداش‌هایی است که در طی زمان نامحدود توسط عامل به دست آمده است، اما با توجه به فاصله زمانی که در آینده این پاداش‌ها به دست می‌آیند، تنزیل می‌یابد. این فرمول بازده شامل نرخ تنزیل $\gamma \in (0,1)$ است:

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \quad (2-2)$$

استفاده از نرخ تنزیل در محاسبه‌ی بازده با دو دلیل توجیه می‌شود. دلیل اول برگرفته از مفهومی در علوم مالی است، و آن این است که پول نقد (پاداش) در حال حاضر و زمان نزدیک بهتر و با ارزش تر از پول نقد در آینده است (به دلایلی همچون عدم قطعیت در آینده و هزینه فرصت). دلیل دوم، که بیشتر بُعد ریاضی مسئله را در نظر دارد، این است که حاصل جمع یک افق نامتناهی از پاداش‌ها ممکن است به یک مقدار محدود همگرا نشوند، و همین باعث ایجاد دشواری‌هایی در معادلات می‌شود. اما با استفاده از نرخ تنزیل و در شرایط معقول، این مجموع بی نهایت همگرا می‌شود.

^{۱۶} Finite-horizon undiscounted return

^{۱۷} Infinite-horizon discounted return

2-1-2- توابع ارزش^{۱۸}

در یک MDP، دانستن ارزش یک حالت یا جفت حالت-عمل بسیار مفید خواهد بود. در اینجا منظور از ارزش، بازده مورد انتظار (امید ریاضی بازده) تحت یک سیاست است در صورتی که از یک حالت یا جفت حالت-عمل شروع کرده و طبق آن سیاست تا انتها عمل کنیم. توابع ارزش در الگوریتم‌های RL به اشکال مختلفی استفاده می‌شود. چهار نوع اصلی تابع مورد توجه ما وجود دارد:

1- تابع ارزش On-Policy، $V^\pi(s)$ ، که بازده مورد انتظار را در صورتی که از عامل حالت s شروع کرده و سپس برای همیشه طبق سیاست π عمل کند، می‌دهد:

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [G(\tau) \mid s_0 = s] \quad (3-2)$$

2- تابع ارزش عمل On-Policy، $Q^\pi(s, a)$ ، که بازده مورد انتظار را در صورتی که عامل در حالت s شروع کرده، عمل دلخواه a را انجام دهد (که ممکن است از سیاست اصلی عامل خارج باشد)، و سپس برای همیشه طبق سیاست π عمل کند، می‌دهد:

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi} [G(\tau) \mid s_0 = s, a_0 = a] \quad (4-2)$$

3- تابع ارزش بهینه، $V^*(s)$ ، که بازده مورد انتظار را اگر در حالت s شروع کنید و همیشه مطابق سیاست بهینه (سیاستی که بیشترین ارزش ممکن را در حالات ایجاد کند) در محیط عمل کنید، می‌دهد:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau \sim \pi} [G(\tau) \mid s_0 = s] \quad (5-2)$$

4- تابع ارزش عمل بهینه، $Q^*(s, a)$ ، که بازده مورد انتظار را اگر در حالت s شروع کنید، یک اقدام دلخواه a را انجام دهید و سپس برای همیشه پس از آن مطابق خط مشی بهینه در محیط عمل کنید، می‌دهد:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\tau \sim \pi} [G(\tau) \mid s_0 = s, a_0 = a] \quad (6-2)$$

¹⁸ Value Functions

ارتباط مهمی بین تابع ارزش عمل بهینه $Q^*(s, a)$ و عمل انتخاب شده توسط سیاست بهینه وجود دارد. سیاست بهینه در حالت s هر عملی را انتخاب می‌کند که بازده مورد انتظار را از شروع در s به حداکثر برساند. در نتیجه، اگر Q^* را داشته باشیم، می‌توانیم مستقیماً عمل بهینه، $a^*(s)$ را از طریق زیر بدست آوریم:

$$a^*(s) = \arg \max_a Q^*(s, a) \quad (7-2)$$

2-2- یادگیری تقویتی

یادگیری تقویتی (Sutton & Barto, 2018) به عنوان سومین پارادایم الگوریتم‌های یادگیری در کنار یادگیری تحت نظارت^{۱۹} و بدون نظارت^{۲۰} (Mohri, Rostamizadeh, & Talwalkar, 2012) در نظر گرفته می‌شود. همانطور که در بخش قبل ذکر شد، برخلاف دو نوع یادگیری دیگر، یادگیری تقویتی برای به حداکثر رساندن پاداش مورد انتظار در محیطی که معمولاً به صورت فرآیند تصمیم‌گیری مارکوف مدل‌سازی می‌شود، استفاده می‌شود. به منظور تخمین بازده مورد انتظار در یک حالت s (یا اصطلاحاً ارزش^{۲۱} یک حالت)، می‌توان از تابع ارزش عمل^{۲۲} (Q) استفاده کرد (Watkins & Dayan, 1992). تابع Q ، در هر مرحله‌ی زمانی t یک حالت s_t و یک عمل a_t را به عنوان ورودی می‌گیرد و مقدار عددی Q مورد انتظار (بازدهی که می‌توانیم از آن حالت و با انجام آن عمل به بعد در محیط انتظار داشته باشیم) را خروجی می‌دهد. در ادبیات این حوزه، تخمینگر تابع ارزش عمل به عنوان Critic یا همان منتقد شناخته می‌شود، به این دلیل که عمل انجام شده در یک حالت را با یک مقدار عددی «نقد» می‌کند. از سوی دیگر، در RL می‌توان مستقیماً سیاستی را که عامل برای به حداکثر رساندن پاداش اتخاذ می‌کند، بهینه کرد. اگر یک سیاست را به عنوان یک تابع توزیع احتمال π (چه پیوسته و چه گسسته) بر روی کل فضای عمل‌ها \mathcal{A} در نظر بگیریم، می‌توان از روش‌های گرادینان سیاست^{۲۳} (PG)

¹⁹ Supervised Learning

²⁰ Unsupervised Learning

²¹ Value

²² Action-value Function (Q Function)

²³ Policy Gradients

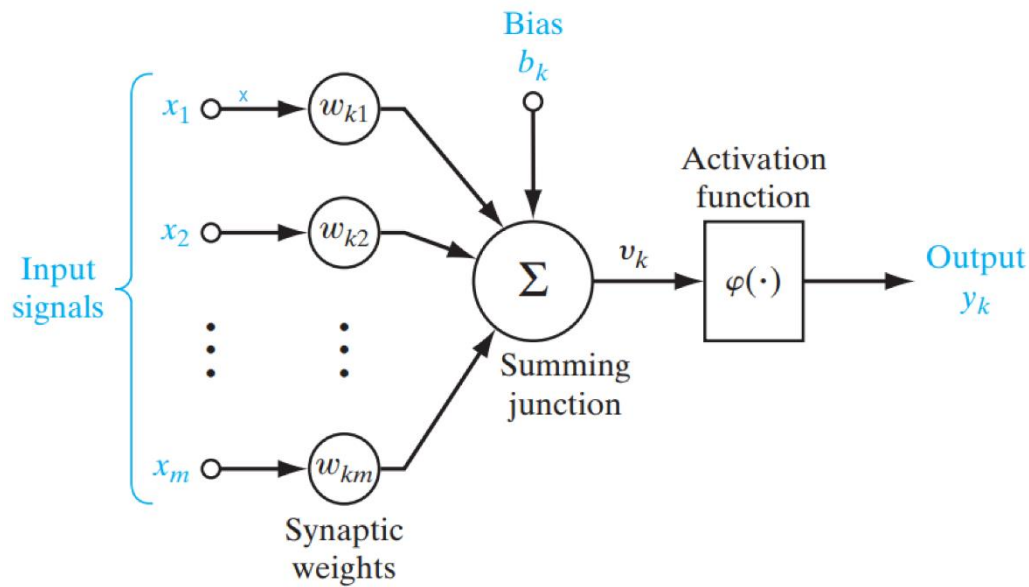
(Silver, et al., 2014) (Sutton, McAllester, Singh, & Mansour, 2000) برای بهینه سازی این توزیع استفاده کرد به طوری که عامل، عملی را که با بالاترین احتمال بیشترین بازده را می‌دهد، از بین تمام اعمال ممکن انتخاب کند. با توجه به اینکه این روش مستقیماً سیاست را کنترل می‌کند، در ادبیات این حوزه از آن به عنوان روش Actor یا همان **عملگر** یاد می‌شود.

مطالعات در مورد یادگیری تقویتی عمیق عمدتاً به دو دسته تقسیم می‌شوند: (1) رویکردهای یادگیری تقویتی مبتنی بر ارزش (روش منتقد) و (2) رویکردهای یادگیری تقویتی مبتنی بر سیاست (روش عملگر). علاوه بر این، رویکردهای یادگیری تقویتی Actor-Critic یا همان **عملگر-منتقد** ترکیبی از روش‌های یادگیری تقویتی مبتنی بر ارزش و رویکردهای مبتنی بر سیاست هستند (Sutton & Barto, 2018).

1-2-2- یادگیری تقویتی عمیق

یادگیری عمیق (LeCun, Bengio, & Hinton, 2015) یکی از روش‌های یادگیری ماشین مبتنی بر شبکه‌های عصبی مصنوعی است و در دهه‌ی اخیر به دلیل عملکرد فوق العاده‌ای که در زمینه‌هایی همچون پردازش تصویر، متن و گفتار داشته، بسیار مورد توجه قرار گرفته است.










واحد اصلی محاسبات در یک شبکه عصبی، نورون نام دارد. که اصطلاحاً به آن گره (node) یا واحد (unit) نیز گفته می‌شود. نورون، از سایر گره‌ها (لایه قبل) یا یک منبع خارجی، ورودی دریافت می‌کند و خروجی را محاسبه می‌کند. به هر ورودی، یک وزن اختصاص داده می‌شود که بیانگر میزان اهمیت آن ورودی نسبت به سایر ورودی‌هاست. در نهایت، جمع وزن‌دار این ورودی‌ها به یک تابع اعمال می‌شود و خروجی نورون بدست می‌آید.



شکل 1-2-2: نمایش محاسبات یک واحد (نورون) در یک شبکه عصبی.

ایده اصلی شبکه های عصبی مصنوعی از اینجاست که وزن های اختصاص یافته به هر ورودی قابل یادگیری هستند. و شدت تاثیر و جهت آن را کنترل می کنند. به این معنی که وزن های مثبت، تحریک کننده هستند و وزن های منفی، بازدارنده (از یک نورون به نورون دیگر). در مدل محاسباتی، سیگنال های وزن دار به واحد محاسباتی می رسند و در آنجا، تمامی این سیگنال ها با هم جمع می شوند و به عنوان ورودی به یک تابع غیرخطی (مشتق پذیر) مانند تابع سیگموئید یا ReLU^{24} (Nair & Hinton, 2010) که به آن تابع فعالسازی یا تابع انتقال گفته می شود داده می شوند. نتیجه نهایی این تابع به نورون بعدی انتقال می یابد.

²⁴ Rectified Linear Unit

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

شکل 2-2-2: معمول ترین توابع فعالسازی به همراه مشتق های آن‌ها.

شبکه های عصبی، در دسته های مختلفی قابل تقسیم بندی هستند. مانند شبکه های عصبی پیشخور^{۲۵} (Haykin, 1994)، شبکه های بازگشتی^{۲۶} (Williams, Hinton, & Rumelhart, 1986)، شبکه های پیچشی^{۲۷} (LeCun, et al.)، شبکه های مبدل^{۲۸} (Hochreiter & Schmidhuber, 1997) و بسیاری از انواع دیگر که از ترکیب یا بهبود این موارد حاصل شده اند. در (Vaswani, et al., 2017)

²⁵ Feed-Forward Neural Networks²⁶ Recurrent Neural Networks²⁷ Convolutional Neural Networks²⁸ Transformers (use attention mechanism)

شبکه‌های پیشخور که ساده‌ترین و ابتدایی‌ترین نوع شبکه‌های عصبی است، اتصالات بین واحدها، تشکیل حلقه نمی‌دهند. در این شبکه، اطلاعات تنها در یک جهت جابجا می‌شود: رو به جلو. یعنی از گره‌های ورودی به سمت گروه‌های خروجی و با عبور از گره‌های پنهان (در صورت وجود). هیچ حلقه یا چرخه‌ای در این نوع شبکه وجود ندارد.

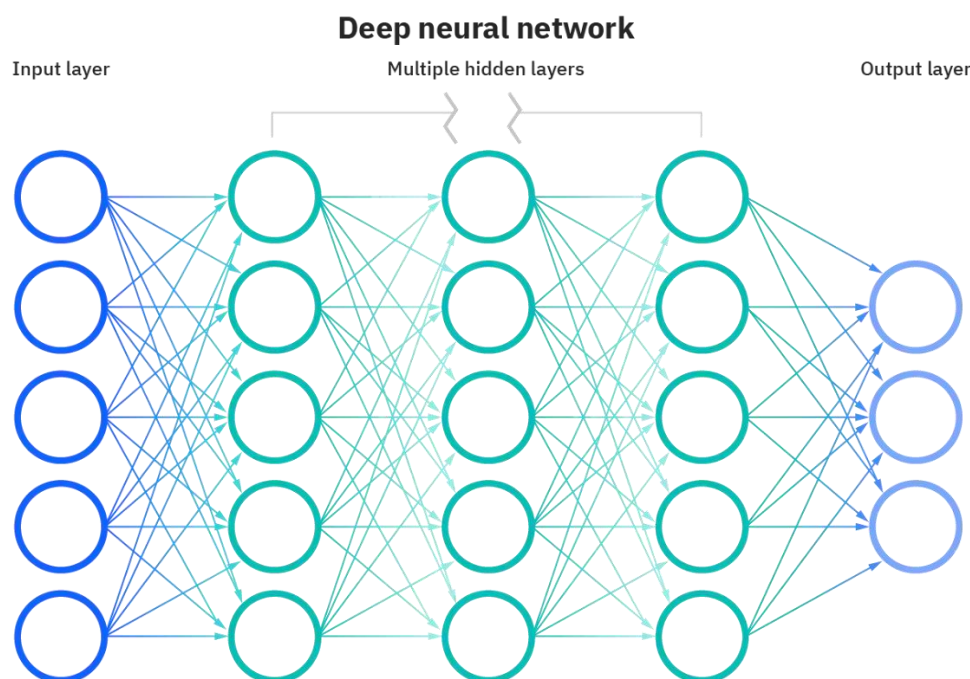
به طور کل می‌توان دو نوع از شبکه‌های عصبی پیشخور را به صورت زیر برشمرد:

1. پرسپترون تک لایه: ساده‌ترین نوع از شبکه‌های عصبی پیشخور است که هیچ لایه پنهانی ندارد. در واقع این شبکه، تنها از یک لایه تشکیل شده است. آن هم، لایه خروجی (گره‌های خروجی) است. نکته قابل توجه این است که در شمارش لایه‌های شبکه، لایه ورودی را حساب نمی‌کنیم. چرا که در لایه ورودی، محاسباتی انجام نمی‌گیرد. و ورودی‌ها، به صورت مستقیم و از طریق یکسری وزن‌ها، به خروجی‌ها پاس داده می‌شوند. به همین دلیل، به این شبکه، پرسپترون تک لایه گفته می‌شود.

2. پرسپترون چند لایه²⁹ (MLP): این دسته از شبکه‌ها، که به شبکه‌های عصبی عمیق نیز معروف هستند، از چندین لایه واحدهای محاسباتی تشکیل شده‌اند که عمدتاً به صورت پیشخور (در جهت رو به جلو) به یکدیگر متصل هستند. هر نورون در یک لایه، مستقیماً به تمامی نورون‌های لایه بعدی متصل است پرسپترون چندلایه در بسیاری از زمینه‌ها مفید واقع شده است و یکی از

²⁹ Multi-Layer Perceptron

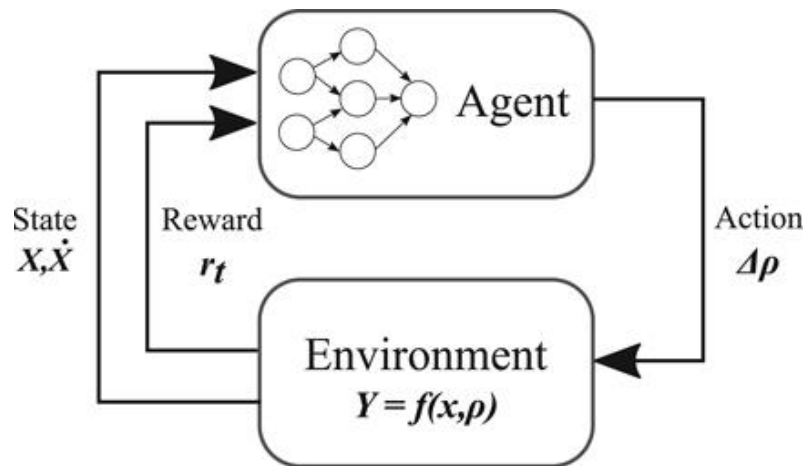
دلایل آن، این است که این دسته از شبکه‌ها قادر هستند توصیف‌های غیرخطی را یاد بگیرند (در بسیاری از موارد، داده‌هایی که با آنها سروکار داریم به صورت خطی قابل جداسازی نیستند).



شکل 2-3-2: شماتیک کلی از معماری شبکه‌های پرسپترون چندلایه (MLP)

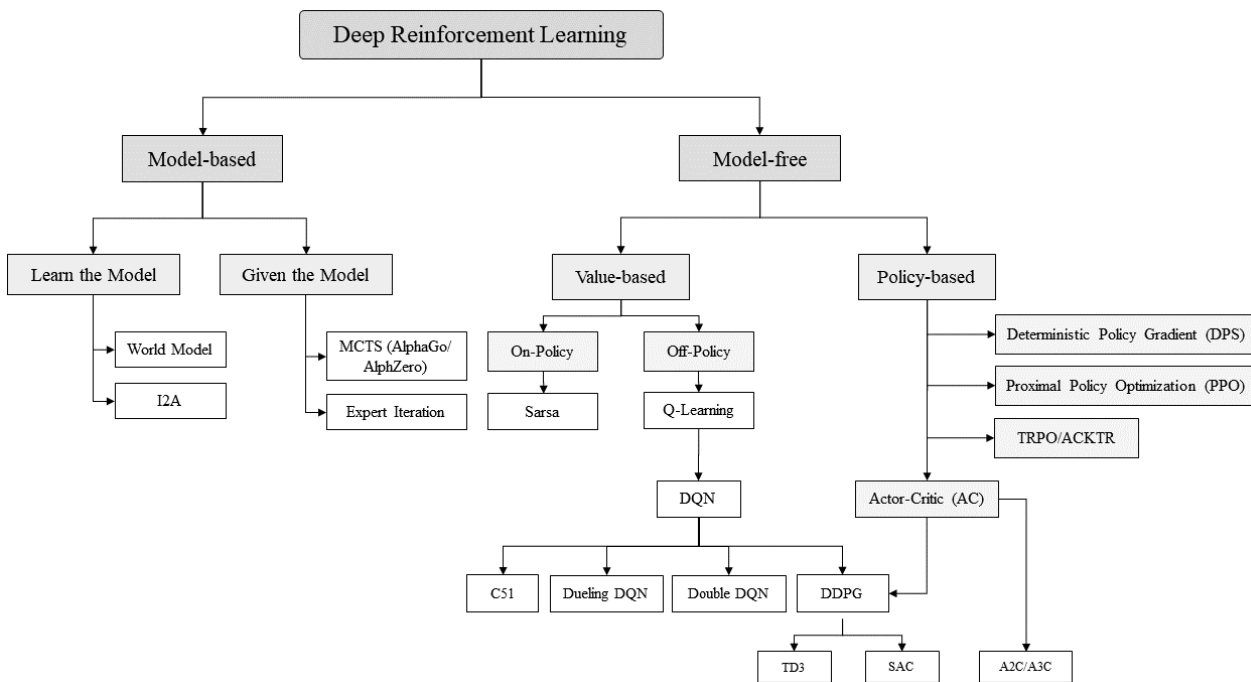
یادگیری عمیق یک چارچوب قدرتمند برای یادگیری تحت نظارت و بدون نظارت است. با افزودن لایه‌های بیشتر و واحدهای محاسباتی بیشتر در هر لایه، یک شبکه عصبی عمیق می‌تواند توانایی پیچیدگی بسیار بالا را بازسازی و بازنمایی کند. اکثر وظایفی که شامل نگاشت یک بردار عددی ورودی به یک بردار عددی خروجی با یک تابع نگاشت ناشناخته یا اصطلاحاً black-box است، می‌توانند از طریق یادگیری عمیق، با توجه به مدل‌ها و مجموعه داده‌های آموزشی برچسب‌گذاری شده کافی و بزرگ انجام شوند (LeCun, Bengio, & Hinton, 2015). از این رو می‌توان از شبکه‌های عصبی برای بازنمایی توانایی که در یادگیری تقویتی استفاده می‌شوند استفاده کرد و از یک رویکرد تصادفی برای آموزش این شبکه‌ها در حین برخورد عامل با محیط بهره برد. استفاده از شبکه‌های عصبی مصنوعی برای بازنمایی توانایی در یادگیری تقویتی چالش‌هایی همچون برخورد با فضاها یا حالت بسیار بزرگ یا

پیوسته، فضاهاى عمل پیوسته و محیط‌هاى بسیار پویا را حل کرده و برخورد با این قبیل از مسائل که بیشتر مسائل دنیای واقعی را شامل می‌شود را بسیار ساده‌تر می‌کند (Sutton & Barto, 2018).



شکل 2-2-4: شمای کلی از نحوه استفاده از شبکه عصبی در یادگیری تقویتی

در یادگیری تقویتی، توابعی همچون تابع ارزش V ، تابع ارزش عمل Q ، تابع سیاست π و تابع انتقال حالت (یا اصطلاحاً مدل محیط) \mathcal{T} قابل بازنمایی توسط شبکه‌های مصنوعی و قابل یادگیری هستند (Schulman, Wolski, Dhariwa, (Schrittwieser, et al., 2020) (Mnih, et al., 2015) (Radford, & Klimov, 2017). روش‌هایی که در آن‌ها هرکدام از این توابع با استفاده از یک شبکه عصبی عمیق نمایش داده شده باشند و یادگیری شوند به عنوان یادگیری تقویتی عمیق شناخته می‌شوند این گروه از زیرروش‌های هوش مصنوعی، به تازگی نتایج موفقیت آمیز چشمگیری در زمینه‌های پژوهشی مختلف داشته‌اند. از پژوهش‌های پیشین انجام شده در کاربرد یادگیری تقویتی عمیق در حوزه‌ی مالی و مدیریت سرمایه می‌توان به (Deng, Bao, Kong, Ren, & Dai, 2017), (Li, (Zheng, & Zheng, 2019), (Yang, Liu, Zhong, & Walid, 2020), (Théate & Ernst, 2021) و (Liu, Xiong, Zhong, Yang, & Walid) اشاره کرد.



شکل 5-2-2: نقشه کلی الگوریتم‌های یادگیری تقویتی عمیق بر اساس نوع تابع تقریب زده شده با شبکه‌های عصبی

2-2-2- یادگیری تقویتی مبتنی بر تابع ارزش (روش منتقد)

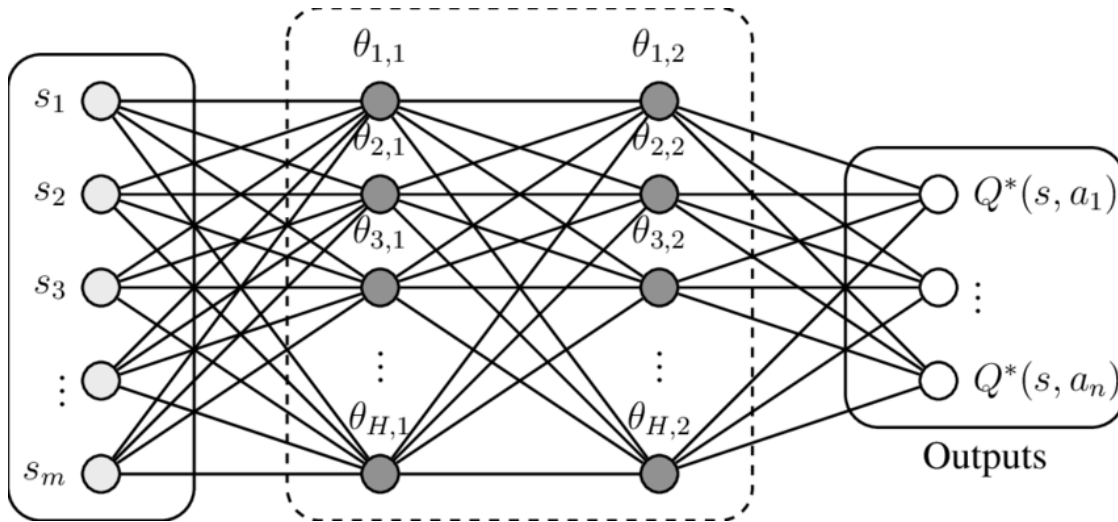
در روش‌های مبتنی بر منتقد، فقط از تابع ارزش عمل Q برای تصمیم‌گیری و انتخاب عمل استفاده می‌شود. در این روش از رویکرد حریصانه³⁰ استفاده می‌شود - یعنی در هر حالت، همیشه عملی انتخاب می‌شود که حداکثر پاداش ممکن را در افق زمانی آینده به ارمغان می‌آورد، مگر اینکه در حالاتی بخواهیم به کاوش بیشتر محیط و بررسی حالات جدید بپردازیم و عملی تصادفی بدون توجه به مقدار Q انتخاب کنیم. توجه داشته باشید که تابع Q را می‌توان به روش‌های مختلفی برآورد کرد، اما محبوب‌ترین و موفق‌ترین رویکرد تاکنون استفاده از شبکه‌های عصبی بوده است (شبکه Q عمیق - Mnih, DQN (2015) et al., و معماری‌های مشتق شده از آن نظیر DoubleDQN (Hasselt & Guez, 2016) و DDPG (Lillicrap, et al., 2016)).

³⁰ Greedy

در این خانواده از روش‌های یادگیری تقویتی، تخمینگر $Q_\theta(s, a)$ که با پارامترهای θ تقریب زده می‌شود برای تخمین تابع ارزش عمل بهینه، $Q^*(s, a)$ با استفاده از نمونه‌های جمع‌آوری شده طی تعامل با محیط یادگیری می‌شود. برای رسیدن به این مقدار بهینه از یک تابع هدف بر اساس معادله بلمن³¹ استفاده می‌شود. معادله‌ی بلمن برای تابع Q به صورت زیر می‌باشد:

$$Q^\pi(s, a) = E_{s' \sim P} \left[r(s, a) + \gamma E_{a' \sim \pi} [Q^\pi(s', a')] \right] \quad (8-2)$$

در این رابطه، s' حالت بعدی محیط طبق تابع احتمال گذار P و a' عمل بعدی عامل در آن حالت طبق سیاست π می‌باشد. $r(s, a)$ نیز پاداش دریافتی عامل در حالت s بعد از عمل a است.



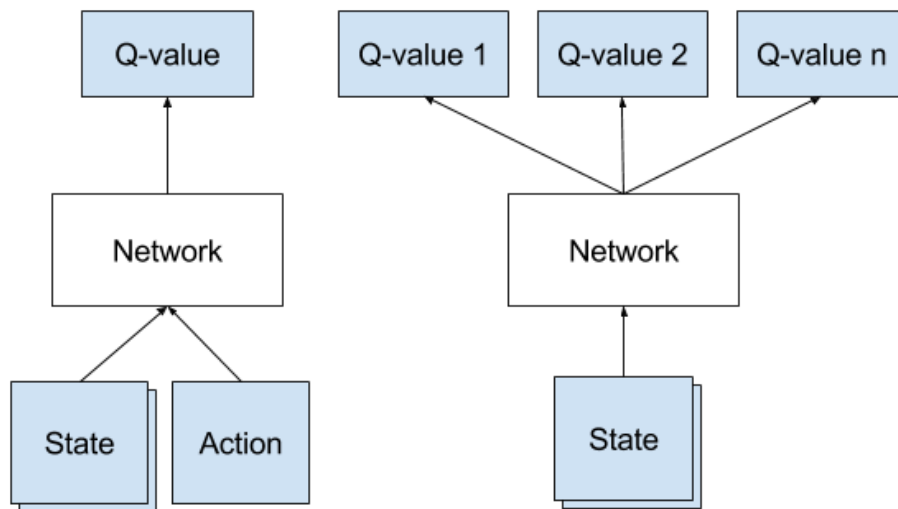
شکل 6-2-2: نحوه‌ی استفاده از شبکه‌های عصبی برای تقریب تابع Q در روش‌های بر پایه منتقد.

این بهینه‌سازی تقریباً همیشه به صورت off-policy انجام می‌شود، به این معنی که هر مرحله از به‌روزرسانی پارامترها می‌تواند از داده‌های جمع‌آوری‌شده در هر نقطه در طول آموزش استفاده کند، صرف نظر از اینکه عامل چگونه هنگام به‌دست‌آوردن داده‌ها با محیط تعامل داشته و این داده‌ها از چه مرحله‌ای

³¹ Bellman equation

به دست آمده‌اند. سیاست مربوطه از طریق ارتباط بین Q^* و π^* بدست می‌آید (Sutton & Barto, 2018)، به این صورت که عمل‌های انجام شده توسط عامل از طریق زیر به دست می‌آید:

$$a(s) = \arg \max_a Q_\theta(s, a)$$



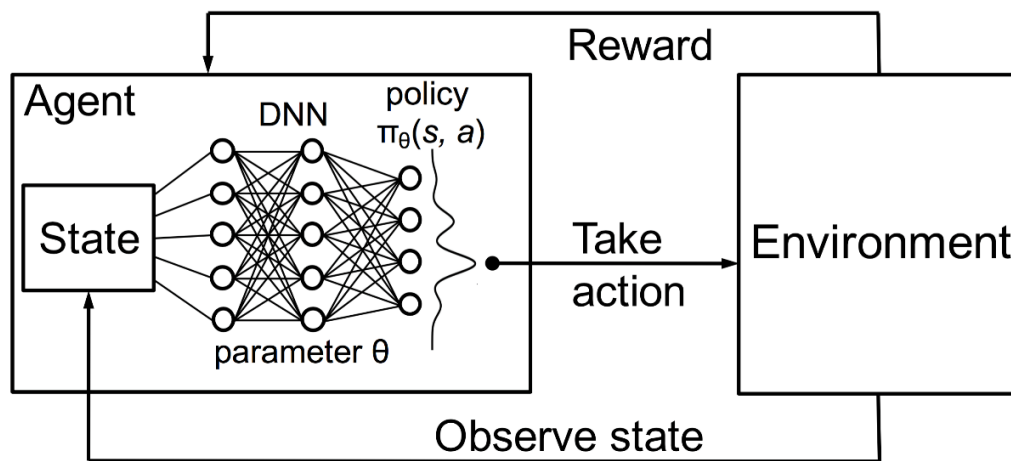
شکل 2-2-7: دو نحوه اصلی پیاده سازی شبکه Q.

3-2-2- یادگیری تقویتی مبتنی بر تابع سیاست (روش عملگر)

در رویکرد مبتنی بر عملگر، منحصراً یک توزیع احتمال بر روی فضای عمل‌ها در هر حالت مدل می‌کنیم. به طور دقیق‌تر، در این روش‌ها یک تابع سیاست به صورت $\pi_\theta(a | s)$ نشان داده می‌شود، که با استفاده از مجموعه پارامترهای θ و با عموماً با استفاده از یک شبکه عصبی تقریب زده می‌شود. و پارامترهای θ نیز یا مستقیماً با الگوریتم گرادیان صعودی^{۳۲} روی تابع هدف $L(\pi_\theta)$ (براساس عملکرد عامل)، و یا به‌طور غیرمستقیم با حداکثر کردن تقریب‌های محلی $L(\pi_\theta)$ بهینه می‌شوند (Sutton, McAllester, Singh, & Mansour, 2000).

³² Gradient Ascent

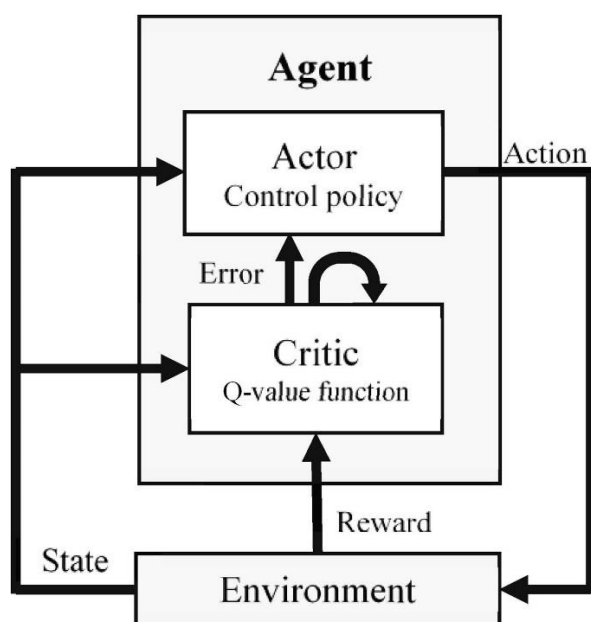
این بهینه‌سازی تقریباً همیشه به صورت on-policy انجام می‌شود، به این معنی که هر در مرحله از به‌روزرسانی پارامترها فقط از داده‌های جمع‌آوری شده از تعامل عامل و محیط طبق آخرین نسخه از تابع سیاست استفاده می‌کند. همچنین برای بهینه‌سازی تابع سیاست معمولاً نیاز به یادگیری یک تخمینگر $V_\phi(s)$ برای تابع ارزش می‌باشد (با مجموعه پارامترهای ϕ)، که از آن برای تعیین جهت بهینه‌سازی تابع سیاست استفاده می‌شود.



شکل 2-2-8: نحوه‌ی استفاده از شبکه‌های عصبی برای تقریب تابع سیاست در روش‌های بر پایه عملگر.

2-2-4- روش عملگر-منتقد (Actor-Critic)

در یادگیری تقویتی عملگر-منتقد، عملگر (طبق توزیع احتمال) با دریافت یک حالت از محیط به عنوان ورودی، یک عمل را به عنوان خروجی می‌دهد و منتقد (ما می‌توانیم به این به عنوان بازخوردی نگاه کنیم که آن عمل را «نقد می‌کند») با گرفتن خروجی عملگر در یک حالت یک مقدار Q به عنوان ارزیابی جفت حالت و عمل انتخاب شده خروجی می‌دهد (Sutton & Barto, 2018).



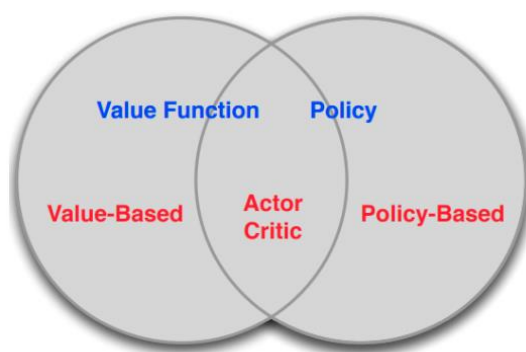
شکل 9-2-2: دیاگرام یک سیستم یادگیری تقویتی Actor-Critic

به طور کل در روش‌های عملگر-منتقد:

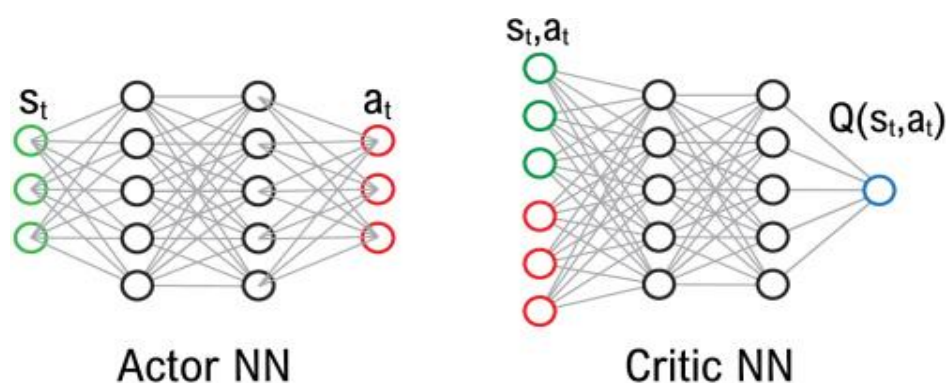
1. منتقد تابع ارزش را تخمین می‌زند. این مقدار می‌تواند مقدار ارزش عمل (مقدار Q) یا مقدار ارزش حالت (مقدار V) باشد.
2. عملگر تابع توزیع سیاست را در جهت پیشنهادی منتقد (مانند روش‌های گرادیان سیاست) به روز می‌کند.

و هر دو توابع منتقد و عملگر با شبکه‌های عصبی پارامترسازی می‌شوند.

الگوریتم‌های اخیر محبوب و پرکاربرد یادگیری تقویتی عمیق مانند Advantage Actor-Critic (A2C) (Mnih, et al., 2016)، Deep Deterministic Policy Gradient (DDPG) (Lillicrap, et al., 2016)، Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, et al., 2016) و Actor-Critic (Radford, & Klimov, 2017)، همگی در این دسته قرار می‌گیرند.



شکل 2-10: نمودار ون روش های یادگیری تقویتی عمیق

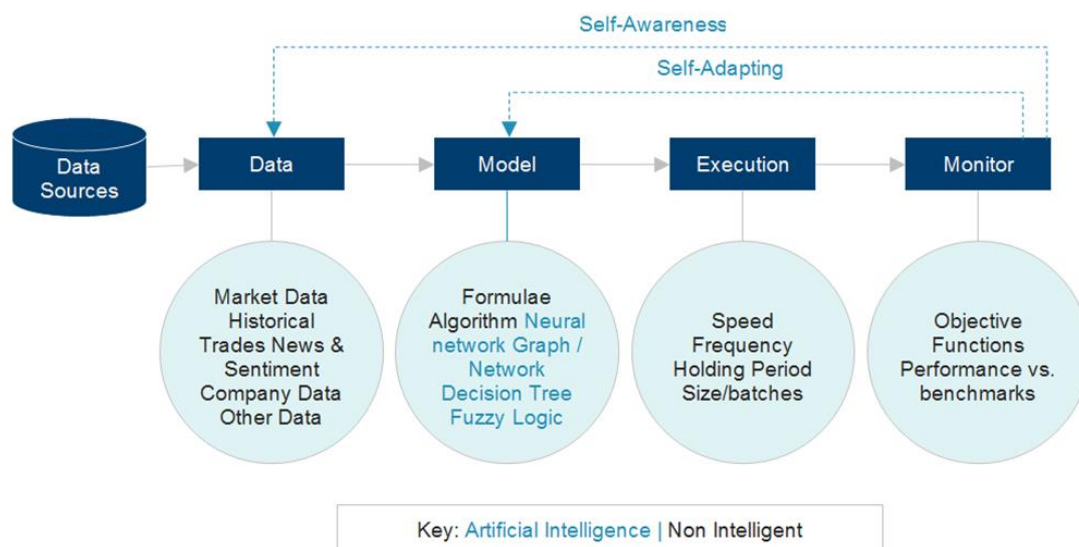


شکل 2-11: نحوه‌ی تقریب دو تابع ارزش و سیاست در روش های Actor-Critic با استفاده از شبکه عصبی

3-2- معاملات کمی الگوریتمی

معاملات کمی سهام به این مضمون اشاره دارد که یک عامل معاملاتی توسعه یافته بتواند تصمیمات خود را بر اساس داده های تاریخی و تجزیه و تحلیل کمی که برگرفته از شاخص های آماری و ابتکاری است برای تشخیص الگوهای موجود در بازار و انجام اقدامات معاملاتی مناسب بهینه کند. معاملات با فرکانس پایین گروهی از این راه حل ها را شامل می شود که در آن، عامل ها بتوانند مانند یک انسان از بازه زمانی ۱ دقیقه ای تا چند روزه معاملات را انجام دهند. از سوی دیگر، سیستم های معاملاتی با فرکانس بالا وجود دارند در سطح پیکو یا میلی ثانیه عمل می کنند و مطالعات در این حوزه معمولاً متعلق به شرکت های برجسته است که سعی می کنند تحقیقات خود را به صورت غیر عمومی نگه دارند.

همانطور که در شکل 1-3-2 نمایش داده شده، طراحی و ایجاد یک سیستم معاملاتی خودکار هوشمند شامل چهار مرحله‌ی اصلی است، آماده‌سازی داده، طراحی مدل، پیاده‌سازی و نظارت عملکرد مدل. که مثال‌های هر مرحله در شکل ذکر شده.

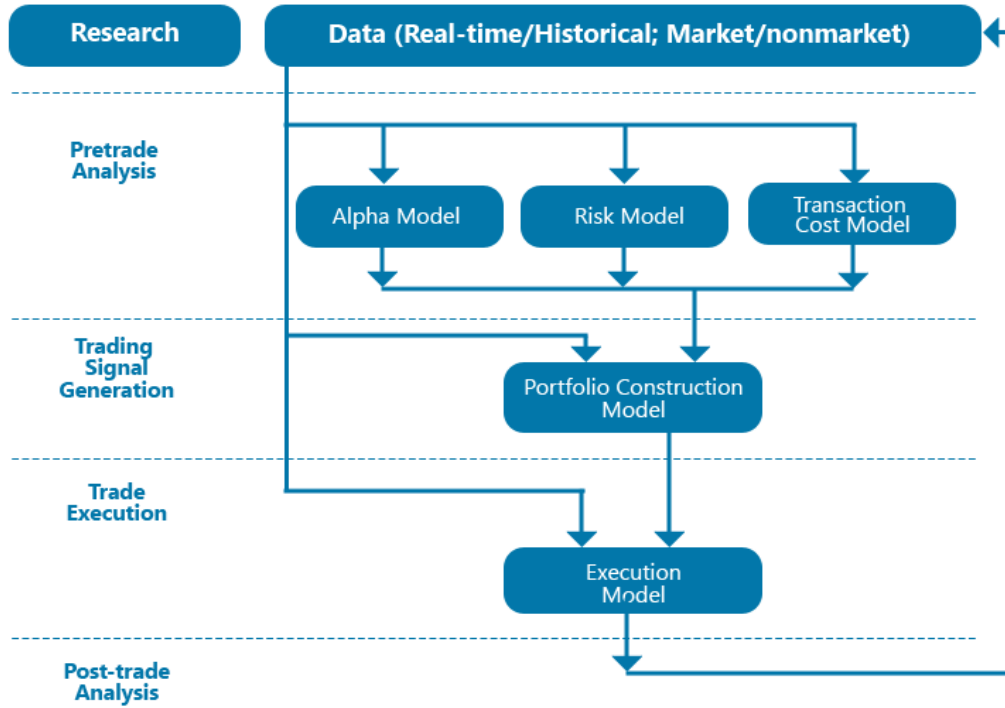


شکل 1-3-2: سیکل ایجاد و نگهداری یک سیستم معاملاتی هوشمند کمی. (AlphaBot, n.d.)

سخت افزار چنین مطالعاتی نیز بسیار محدود است که این خود دلیلی برای کمبود چنین مطالعاتی در معاملات با فرکانس بالا است، زیرا برای اتصال به مراکز داده زنده، به چندین دستگاه پردازش گرافیکی بسیار قوی و سامانه‌هایی با تأخیر ارتباطی بسیار کم نیز نیاز است. از این رو این دسته از معاملات الگوریتمی در این پژوهش مورد بررسی قرار نمی‌گیرند. در این پروژه فقط به استفاده از تکنیک‌های جدید یادگیری تقویتی عمیق در توسعه‌ی سیستم‌های معاملاتی کمی الگوریتمی می‌پردازیم.

همانطور که در شکل 2-3-2 نمایش داده شده، یک سامانه معاملاتی از پنج سطح تشکیل می‌شود. ابتدایی ترین سطح این سامانه، بخش انجام پژوهش‌ها و آزمایش‌های متعدد و جمع آوری انواع داده‌های مورد نیاز است. سطح دوم، ایجاد اجزای مختلف مدل معاملاتی شامل مدل ایجاد بازدهی (alpha model)، مدل مدیریت ریسک (risk model) و مدل هزینه تراکنش و اجرا (transaction cost) می‌باشد. سطح سوم، بخش ایجاد سیگنال معاملاتی بر اساس مدل‌های طراحی شده و ایجاد سبد سرمایه‌گذاری است. سطح چهارم نیز مربوط به پیاده‌سازی و اجرای معاملات بر اساس سیگنال‌ها و

سطح آخر نیز بررسی مداوم عملکرد سامانه برای جلوگیری از خطاهای احتمالی و بهبود هر یک از بخش‌ها در سطوح قبلی سامانه است.



شکل 2-3-2: اجزای اصلی یک سیستم معاملاتی کمی خودکار. (AlphaBot, n.d.)

3- فصل سوم

مدلسازی و روند پژوهش

مدلسازی و روند پژوهش

در این بخش، جزئیات مدل‌سازی و پیاده‌سازی این پژوهش را شامل تنظیمات محیط، داده‌ها و پیش پردازش‌ها، فضای حالت، فضاها، عمل و توابع پاداش را معرفی می‌کنیم. همچنین دو الگوریتم RL مورد استفاده در این پروژه، یعنی روش‌های شبکه‌های عصبی Q عمیق (DQN) و بهینه‌سازی کم‌فاصله‌ی سیاست (PPO) را شرح می‌دهیم. این دو روش به این دلیل انتخاب شدند تا از هر کدام از گروه الگوریتم‌های یادگیری تقویتی (ارزش محور و سیاست محور) یک روش را به عنوان نماینده مورد بررسی قرار دهیم. به طور معمول روش‌های زیر مجموعه‌ی هر کدام از این گروه‌ها شباهت‌های بسیار زیادی از نظر جزئیات عملکرد دارند، از این رو از هر کدام یک روش یک انتخاب شد تا تفاوت‌های عملکردی این دو گروه مورد بررسی قرار گیرد. همچنین در این پژوهش به دلیل تک سهمیه بودن و بی‌تاثیر بودن معامله با درصدی از سرمایه، تنها فضاها، عمل گسسته مورد بررسی قرار گرفته‌اند.

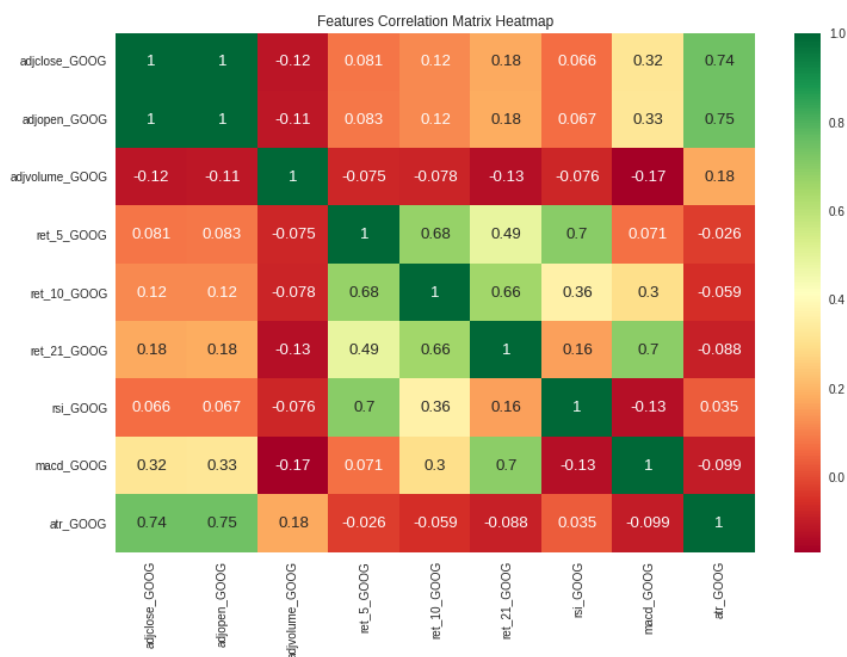
3-1- محیط، فضای حالت و داده‌ها

همانطور که پیش‌تر بحث شد، ما می‌توانیم برای این مسئله محیط معاملات را به عنوان یک فرآیند تصمیم‌گیری مارکوف (MDP) (Bellman, 1957) فرموله کنیم که در آن یک عامل در مراحل زمانی گسسته با محیط تعامل دارد. در هر مرحله زمانی t ، عامل یک بردار بازنمایی از حالت محیط را دریافت می‌کند که به عنوان حالت s_t نشان داده شده است و با توجه به این حالت، یک عامل یک عمل a_t را انتخاب می‌کند و بر اساس این عمل، یک پاداش عددی r_t به عامل در حالت داده می‌شود. مرحله بعدی، و عامل خود را در وضعیت جدید s_{t+1} می‌یابد. تعامل بین عامل و محیط یک مسیر از تعاملات $[s_1, a_1, r_1, s_2, a_2, r_2, \dots]$ را ایجاد می‌کند. در هر مرحله زمانی t ، هدف RL به حداکثر رساندن بازده مورد انتظار (پاداش تجمعی تنزیل شده) است که در زمان t با عنوان G_t نشان داده می‌شود:

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} r_k \quad (1-3)$$

که در این رابطه γ ضریب تنزیل پاداش است.

در ادبیات این حوزه، از ویژگی‌های مختلفی برای نشان دادن فضاها استفاده شده است. در میان این ویژگی‌ها، قیمت گذشته دارای مورد معامله تقریباً همیشه لحاظ می‌شود و اغلب از اندیکاتورهای تکنیکال مرتبط نیز استفاده می‌شود (Fischer, 2018). در این پژوهش، با استناد به (Li, Zheng, 2019 & Zheng, 2019) و (Zhang, Zohren, & Roberts, Deep Reinforcement Learning for Trading, 2019) از قیمت‌های گذشته، بازده R_t در افق‌های مختلف، و شاخص‌های فنی از جمله واگرایی میانگین متحرک همگرایی (MACD) (Baz, Granger, Harvey, Roux, & Rattray, 2015) و شاخص قدرت نسبی (RSI) (Wilder, 1978) برای نمایش حالت‌ها استفاده کرده ایم. ما این داده‌های روزانه را از تاریخ 01-01-2014 تا 01-08-2022 در اختیار داریم، 80٪ از نمونه‌های ابتدایی داده‌ها را به عنوان داده‌های آموزش مدل و 20٪ دیگر را به عنوان داده‌های تست و ارزیابی عملکرد در نظر گرفتیم. همچنین بر اساس شکل 1-1-3 تحلیل همبستگی بر روی داده‌ها انجام شده و پس از بررسی قیمت پایانی روزانه از ویژگی‌ها حذف شد. در هر مرحله زمانی معین، 10 مشاهده گذشته از هر شاخص را در نظر می‌گیریم تا بردار حالت را تشکیل دهیم.



شکل 2-1-3: تحلیل همبستگی ویژگی‌های مورد بررسی.

شاخصه‌های استفاده شده به صورت زیر می‌باشد:

- قیمت های تاریخی بازگشایی روزانه (adjusted open) دارایی مورد معامله برای 10 روز گذشته در هر مرحله ی زمانی.
- حجم تعدیل شده ی معاملات روزانه ی سهام برای 10 روز گذشته در هر مرحله ی زمانی.
- بازده های 5، 10 و 21 روزه ی دارایی مورد معامله برای 10 روز گذشته در هر مرحله ی زمانی.
- شاخص (MACD) Moving Average Convergence Divergence:

$$MACD_t = EMA_t(12) - EMA_t(26) \quad (2-3)$$

که در آن $EMA_t(S)$ میانگین متحرک وزنی نمایی قیمت‌ها با مقیاس زمانی S روزه است. دلیل انتخاب دوره‌های 12 و 26 به عنوان پنجره‌های زمانی کوتاه و بلند مدت MACD، مقادیر پیشفرض استفاده شده در عموم نرم افزارها و پژوهش هاست که عملکرد قابل قبولی در تشخیص روند بازار داشته است.

- شاخص RSI یک اندیکاتور نوسانی است که بین 0 تا 100 حرکت می کند. این شاخص با اندازه گیری میزان تغییرات اخیر قیمت، شرایط فروش بیش از حد (مقدار زیر 20) یا خرید بیش از حد (بالای 80) دارایی را نشان می دهد. ما این شاخص را با یک پنجره نگاه به عقب 10 روزه در نمایش حالت محیط خود لحاظ می کنیم.
- میانگین دامنه واقعی (ATR) یک شاخص نوسان است (Wilder, 1978) که بصورت زیر محاسبه می شود.

$$ATR_t = \frac{ATR_{t-1} \times (n - 1) + TR_t}{n} \quad (3-3)$$

که در این رابطه n تعداد نمونه‌های استفاده شده در میانگین نمایی تا آن مرحله ی زمانی می - باشد؛ همچنین:

$$TR = \max[(high - low), \text{abs}(high - \text{close}_{prev}), \text{abs}(low - \text{close}_{prev})]$$

$$ATR_0 = \frac{1}{n} \sum_{i=1}^n TR_i$$

ما این شاخص را با نیز یک پنجره 10 روزه در نمایش حالت خود لحاظ می‌کنیم.

در نهایت با در نظر گرفتن این 8 شاخصه در یک پنجره‌ی زمانی 10 روزه، محیط را با استفاده از 80 شاخصه (یک بردار 80 مولفه‌ای) در هر مرحله‌ی زمانی بازنمایی می‌کنیم. همچنین از آنجا که برای در دسترس بودن همه‌ی شاخصه‌ها در هر مرحله‌ی زمانی حداقل به داده‌های قیمتی 26 روز گذشته برای محاسبه‌ی MACD نیاز داریم و همچنین برای بازنمایی حالت هر مرحله‌ی زمانی به 10 روز گذشته نیاز داریم، داده‌های قابل استفاده از داده‌ی 37 ام مجموعه داده‌های در دسترس در هر دو بخش یادگیری و ارزیابی شروع می‌شود و 36 روز اول از هر بخش از مجموعه داده‌ها غیرقابل استفاده خواهد بود. در ابتدای شروع تعاملات یک عامل با محیط، مبلغ 100,000 دلار به عنوان سرمایه اولیه به صورت نقد در اختیار عامل قرار می‌گیرد.

3-2- فضای عمل

ما در این پژوهش گروه از فضاها‌ی عمل گسسته را مورد مطالعه و بررسی قرار می‌دهیم. برای فضاها‌ی کنش گسسته، از یک مجموعه عمل با مقادیر گسسته $a_t \in \{-1, 0, 1\}$ استفاده می‌شود و هر مقدار به طور مستقیم درصد دارایی در یک موقعیت معامله را نشان می‌دهد، یعنی 1- مربوط به موقعیت فروش استقراضی کامل به اندازه معادل ارزش کل دارایی‌های در اختیار عامل، 0 بدون موقعیت (دارایی نقد) و 1 مربوط به یک موقعیت خرید کامل است. باید توجه داشته باشیم که اگر عمل فعلی و عمل بعدی یکسان باشد، هیچ تراکنشی رخ نخواهد داد و ما فقط موقعیت‌های خود را حفظ می‌کنیم، در نتیجه دو عمل 1 در دو روز متوالی به منزله‌ی دو روز متوالی در موقعیت خرید و در تبعاً به معنای نگهداری (Hold) خواهد بود. همانطور که در بخش تعریف مسئله ذکر شد، به منظور ساده‌سازی، هزینه‌ی تراکنش در این مطالعه برابر صفر در نظر گرفته شده و سفارش گذاری هزینه‌ای نخواهد داشت. لازم به ذکر است، که در واقعیت، از این جهت که برای مقدار خرید استقراضی محدودیتی وجود ندارد و می‌توان هر مقدار در دسترسی از سهام را استقراض کرد و به فروش رساند، مقادیر قابل قبول برای عمل‌های متناظر با فروش استقراضی بی کران بوده و تمام مقادیر $a_t < 0$ را شامل می‌شود. اما به دلایلی همچون، ساده‌سازی مسئله، محدودیت مدل‌های سیاست در نوع خروجی (تابع \tanh) و محدود کردن ریسک عامل، حد این مقدار 1- در نظر گرفته شده و مقداری کمتر از آن برای اعمال اتخاذ نخواهد کرد.

طراحی فضاهای عمل پیوسته که در این پژوهش مورد مطالعه قرار نمی‌گیرد نیز بسیار شبیه به حالت گسسته است با این تفاوت که در آن اعمال ما درصدی از کل سرمایه خواهد بود که درگیر موقعیت معاملاتی می‌شود، درصدی از کل سرمایه که باید با آن موقعیت معامله تعیین شود با مقداری بین 1- و 1 به عنوان عمل در ابتدای روز و با قیمت بازگشایی آن روز موقعیت گیری می‌شود ($a_t \in [-1,1]$).

3-3- الگوریتم‌ها و معماری شبکه‌های عصبی عامل‌ها

در این پژوهش از الگوریتم‌های Deep Q Learning (DQN) (Mnih, et al., 2015) و Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwa, Radford, & Klimov, 2017) استفاده شده که در ادامه‌ی این بخش، ابتدا به عنوان پیش نیاز، به توضیح روش‌های گرادیان سیاست ساده می‌پردازیم سپس به تشریح جزئیات و معماری هر کدام از الگوریتم‌ها می‌پردازیم.

1-3-3 Vanilla Policy Gradient (PG)

در روش‌های گرادیان سیاست، برخلاف روش‌های مبتنی بر ارزش (مانند Q-Learning) که تابع سیاست از پیش تعیین شده‌ای (مانند سیاست حریم‌انه¹) دارند، تابع سیاست در ابتدا به صورت رندوم انتخاب شده (که توسط یک تخمینگر مانند شبکه عصبی بازنمایی می‌شود) و سپس به یافتن مقدار بهینه‌ی پارامترهای آن می‌پردازد. این روش با محاسبه یک تخمینگر برای گرادیان تابع سیاست و استفاده از آن در الگوریتم گرادیان صعودی تصادفی SGA^2 برای یافتن بهترین پارامترها برای تابع سیاست کار می‌کند.

به صورت دقیق‌تر، در این روش، هدف، بهینه کردن مقدار تابع هدف $L^{PG}(\theta)$ از طریق یافتن پارامترهای بهینه تابع سیاست $\pi_\theta(a_t | s_t)$ می‌باشد. به صورت کلی بهینه سازی تابع $L^{PG}(\theta)$ به این معنی است که عمل‌هایی که در هر حالت ارزش و پاداش بیشتری ایجاد می‌کنند، احتمال بیشتری برای انتخاب شدن در آن حالت داشته باشند. در پیاده‌سازی‌هایی که از نرم‌افزار مشتق‌گیری خودکار استفاده

¹ Greedy Policy: سیاستی که در هر حالت، عملی که بیشترین مقدار ارزش حالت-عمل (Q) را دارد انتخاب می‌کند.

² Stochastic Gradient Ascent

می‌کنند، با ساخت یک تابع هدف $L^{PG}(\theta)$ و محاسبه‌ی گرادیان آن به تخمینگر گرادیان تابع سیاست دست می‌یابند. تخمینگر \hat{g} با مشتق گرفتن جزئی از تابع هدف $L^{PG}(\theta)$ به دست می‌آید:

$$L^{PG}(\theta) = \mathbb{E}_t[\log \pi_\theta(a_t | s_t) \hat{A}_t] \quad (4-3)$$

$$\hat{g} = \nabla_\theta L^{PG} = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t | s_t) \hat{A}_t] \quad (5-3)$$

که در این رابطه π_θ یک تخمینگر تابع سیاست تصادفی پارامتری شده با مجموعه پارامتر θ (مانند یک شبکه عصبی مصنوعی یا تابع خطی) و \hat{A}_t تخمینگر تابع مزیت¹ در مرحله‌ی زمانی t است که به صورت زیر محاسبه می‌شود:

$$\hat{A}_t = G_t - \hat{V}_\phi(s_t) \quad (6-3)$$

در این رابطه، G_t بازده (پاداش تجمعی تنزیل شده) در هر مرحله‌ی زمانی است که از طریق مسیر تعامل طی شده توسط عامل و ثبت پاداش‌های حاصل شده محاسبه می‌شود. و \hat{V}_ϕ نیز یک تخمینگر تابع ارزش حالت پارامتری شده با مجموعه پارامتر ϕ (مانند یک شبکه عصبی مصنوعی یا تابع خطی) می‌باشد.

پارامترهای تابع سیاست $\pi_\theta(a_t | s_t)$ نیز به در هر مرحله به این صورت به روز رسانی و بهینه می‌شوند:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta L^{PG} \quad (7-3)$$

همچنین پارامترهای تخمینگر تابع ارزش حالت $\hat{V}_\phi(s)$ نیز در هر مرحله به این صورت به روز رسانی می‌شوند:

$$\phi_{k+1} = \phi_k - \alpha' \nabla_\phi L^V \quad (8-3)$$

که در این رابطه $L^V = (\hat{V}_\phi - V_{target})^2$ تابع هزینه‌ی ارزش حالت، برای استفاده در آموزش تخمینگر تابع ارزش حالت می‌باشد.

¹ Advantage Function

در هر دو رابطه، α و α' هایپرپارامتر بوده که عددی در بازه‌ی $(0,1)$ را دارا می‌پذیرند و به صورت تجربی، با جست و جو یا آزمون و خطا مشخص می‌شود.

در نهایت عامل به صورت زیر با الگوریتم PG آموزش داده می‌شود (خلاصه مراحل بالا):

Algorithm 1 Vanilla Policy Gradient Algorithm

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t.$$

- 7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.

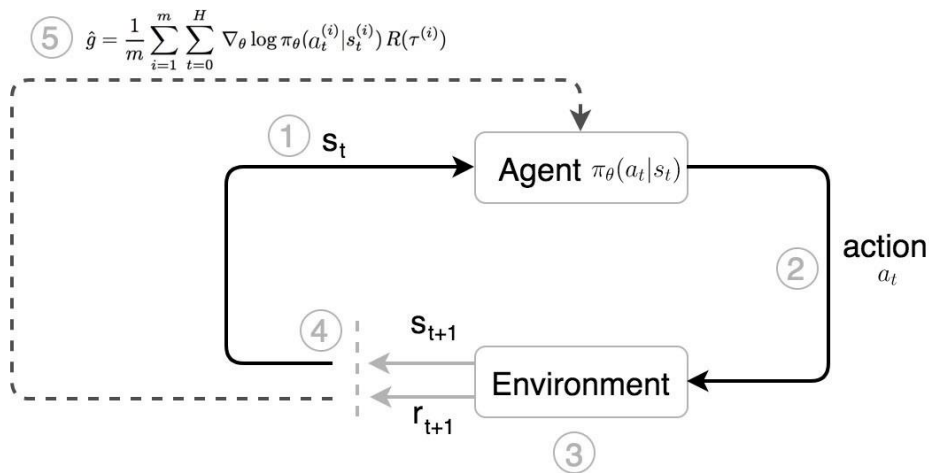
- 8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.

- 9: **end for**
-

شکل 1-3-3: الگوریتم آموزش یک عامل با روش گرادیان سیاست ساده. (OpenAI, Vanilla Policy Gradient, n.d.)



شکل 2-3-3: دیباگرام نحوه‌ی تعامل و آموزش در روش گرادیان سیاست. (Hui, 2018)

معماری در نظر گرفته شده برای شبکه‌ی عملگر و منتقد در این پژوهش، شامل دو لایه‌ی میانی (پنهان) که لایه اول شامل 128 نورون و لایه دوم شامل 64 واحد (نورون) با تابع فعال‌سازی ReLU می‌باشد و در انتها برای منتقد یک لایه‌ی خروجی با یک نورون برای تخمین مقدار ارزش حالت V و برای عامل دو واحد خروجی برای تخمین میانگین و واریانس تابع توزیع سیاست در هر حالت است.

Proximal Policy Optimization (PPO) -2-3-3

الگوریتم PPO (Schulman, Wolski, Dhariwa, Radford, & Klimov, 2017) یک خانواده از روش‌های گرادیان سیاست (PG) برای یادگیری تقویتی است که همانطور که در بخش قبل بحث شد، به این معنی است که به جای تخصیص مقدار ارزش بهینه به جفت‌های حالت-عمل، و انتخاب عملی که بیشترین ارزش را دارد، فضای پارامتری تابع سیاست را جستجو و مستقیماً پارامترهای تابع سیاست را بهینه می‌کند. این الگوریتم همزمان از نمونه‌گیری داده‌ها از طریق تعامل با محیط، و بهینه‌سازی یک تابع هدف «جانشین» با استفاده از SGA برای بهینه‌سازی پارامترهای تابع سیاست استفاده می‌کند.

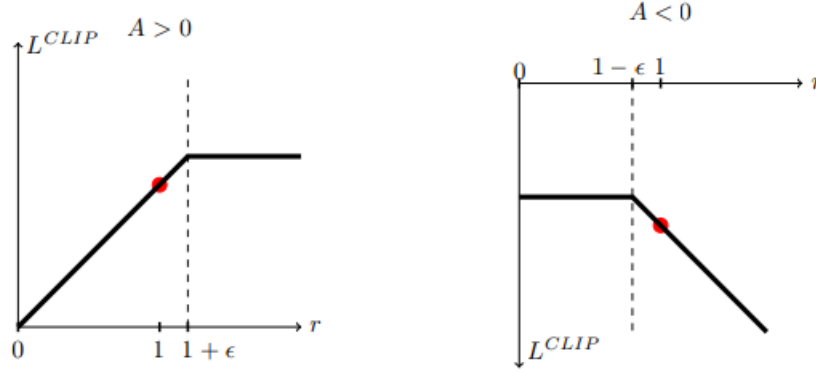
اما مشکل اینجاست که روش‌های گرادیان سیاست ساده بسیار و حساس هستند و ممکن است در حین یادگیری، با حاصل شدن یک مقدار گرادیان بزرگ، به سمت یک سیاست غیر بهینه سقوط کنند. الگوریتم PPO برای رفع این مشکل ارائه شد، و با محدود کردن تابع هدف، از ایجاد گرادیان‌های بزرگ جلوگیری کرده و باعث مستحکم‌تر شدن الگوریتم می‌شوند.

فرض کنید $r_t(\theta)$ نشان دهنده نسبت احتمال $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ باشد، بنابراین $r(\theta_{old}) = 1$ خواهد بود. در این رابطه $\pi_{\theta_{old}}$ تابع سیاست با پارامترهای مرحله‌ی قبلی بهینه‌سازی است و نسبت $r_t(\theta)$ میزان تغییر نسبی احتمال یک عمل را قبل و بعد از یک مرحله از بهینه‌سازی با هدف پیگیری تغییرات زیاد آن نشان می‌دهد. تابع هدف محدود شده در PPO بدین صورت خواهد بود:

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (9-3)$$

در این رابطه، نسبت r بسته به مثبت یا منفی بودن A در $1 - \epsilon$ یا $1 + \epsilon$ قطع و محدود می‌شود. تابع هدف در شکل 3-3-3 نشان داده شده.

لازم به ذکر است که در باقی موارد الگوریتم PPO مانند الگوریتم‌های گرادیان سیاست ساده عمل می‌کنند.



شکل 3-3-3: محدودسازی تابع هزینه در الگوریتم PPO (Schulman, Wolski, Dhariwa, Radford, & Klimov, 2017)

نهایتاً عامل به نحو زیر با الگوریتم PPO آموزش داده می‌شود:

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

شکل 3-3-4: آموزش عامل با الگوریتم PPO (OpenAI, Proximal Policy Optimization, n.d.)

3-3-3 Deep Q-Learning (DQL)

در الگوریتم DQL تابع ارزش حالت-عمل (تابع Q) با استفاده از یک شبکه عصبی تقریب زده می‌شود (اصطلاحاً شبکه‌ی Q عمیق یا DQN) تا میزان مطلوبیت اتخاذ یک عمل معین در یک حالت معین مشخص شود. در این الگوریتم فرض می‌کنیم تابع Q_w با یک شبکه عصبی با پارامترهای w پارامتری شده است. ما میانگین مربعات خطای بین خروجی‌های شبکه و مقدار هدف Q را به عنوان تابع هزینه با استفاده از الگوریتم گرادیان نزولی به حداقل می‌رسانیم تا مقادیر بهینه پارامترهای تابع Q (w) را بدست آوریم:

$$L^{DQN}(w) = \mathbb{E} \left[(Q_w(S, A) - Q'_w(S, A))^2 \right] \quad (10-3)$$

$$Q'_w(S_t, A_t) = r + \gamma \operatorname{argmax}_{A'} Q_w(S_{t+1}, A_{t+1}) \quad (11-3)$$

که در این رابطه $L^{DQN}(\theta)$ تابع هدف بهینه‌سازی (تابع هزینه) است. یکی از مشکلات DQN ساده این است که حین آموزش پایدار نیست و از تنوع رنج می‌برد. پیشرفت‌های زیادی برای تثبیت فرآیند آموزش انجام شده است مانند، Q هدف ثابت، Double DQN (Hasselt & Guez, 2016) و Dueling DQN.

معماری در نظر گرفته شده برای شبکه‌ی Q در این پژوهش، شامل دو لایه‌ی میانی (پنهان) هر کدام شامل 64 واحد (نورون) با تابع فعال‌سازی ReLU می‌باشد و در انتها یک لایه‌ی خروجی با سه نورون متناظر با هر یک از عمل‌ها (1، 0 و -1) می‌باشد برای تخمین مقدار ارزش آن عمل در هر حالت است.

3-4 مکانیزم پاداش

طراحی تابع پاداش برای محیط به میزان بازده روزانه‌ی عامل بستگی دارد. در این پژوهش، پاداش هر مرحله‌ی زمانی را برابر با فاصله‌ی بازده روزانه‌ی عامل با بازده روزانه‌ی خود دارایی (درصد اختلاف قیمت روزانه‌ی دارایی) در نظر می‌گیریم. هدف از این طراحی تابع پاداش، ترغیب عامل به عملکرد بهتر نسبت به خرید کامل و نگهداری دارایی است. پاداش r_t در زمان t برابر است با:

$$r_t = \frac{W_t}{W_{t-1}} - 1 \quad (12-3)$$

که در آن، W_t مقدار کل دارایی در دسترس عامل در مرحله ی زمانی t (مجموع دارایی در موقعیت و دارایی نقد خارج از موقعیت) بعد از اجرای عمل مربوطه در همان روز می باشد. هدف نهایی عامل حداکثر ساختن مجموع تنزیل شده ی این مقدار عددی در طول رشته تعامل با محیط خواهد بود.

5-3- فرآیند آموزش عامل

برای پیاده سازی، آموزش و ارزیابی روش ها از زبان برنامه نویسی پایتون در این پروژه استفاده شده است. برای طراحی مدل ها از کتابخانه ی مرسوم و مورد تایید 3 stable baselines استفاده شده، که به ما اجازه و انعطاف کامل در طراحی اجزای مختلف مدل ها را به علاوه ی مقادیر و معماری های اولیه می دهد. محیط های شبیه سازی بازار برای آموزش و تعامل عامل ها نیز به طور کامل از صفر طراحی شده. برای ارزیابی نتایج روش ها نیز از کتابخانه ی pyfolio بهره گرفته شده است. کد پروژه نیز به صورت کامل در بخش پیوست گزارش قرار داده شده. به منظور آموزش عامل ها، هر یک از مدل ها را به صورت جداگانه برای 500 هزار مرحله ی زمانی، در تعامل با محیط قرار دادیم و شبکه های عصبی را با استفاده از داده های تعاملات به دست آمده آموزش دادیم و بهترین عملکرد در دوره ی تست را حین پروسه ی آموزش ذخیره کردیم. برای هر کدام از مدل ها مقادیری به عنوان هایپر پارامترها در نظر گرفته شده که به صورت زیر می باشد:

جدول 1-3 مقادیر انتخاب شده برای هایپر پارامترهای مدل ها.

نرخ یادگیری (lr)	گاما	تعداد ایپاک	حد قطع گرادیان	تاو	بازه ی اپسیلون (نرخ کاوش)	
0.0001	0.99	15	0.2	-	-	PPO
0.0001	0.99	1	-	0.99	[1, 0.02]	DQN

هایپرپارامترها و معماری‌های انتخاب شده برای مدل‌ها (تعداد لایه، تعداد نرون، تابع فعال‌سازی و ...) از طریق جست و جوی دستی و آزمون خطای کاندیدهای متعددی به دست آمده. لازم به ذکر است، الگوریتم‌های یادگیری تقویتی بسیار به انتخاب هایپرپارامتر حساس بوده و به دلیل تعداد بالای هایپرپارامترها فضای پارامتری بسیار بزرگی را نیز برای انتخاب هایپرپارامتر شامل می‌شوند. از این رو پروسه‌ی انتخاب بهینه این هایپرپارامترها بسیار دشوار و زمان‌بر بوده و نتیجه‌ی نهایی را تحت تاثیر قرار می‌دهد. با این وجود سعی شده با جست و جوی کافی این فضا بهترین هایپرپارامترهای ممکن در مرحله‌ی cross-validation¹ انتخاب شود.

¹ به پروسه‌ی جداسازی داده‌های آموزش از داده‌های تست و انجام ارزیابی روی داده‌های تست گفته می‌شود.

4- فصل چهارم

ارزیابی عملکرد

ارزیابی عملکرد

برای ارزیابی عملکرد عامل‌های آموزش داده شده، نتایج بک‌تست^۱ هریک از عامل‌ها به صورت جداگانه را بر روی سهام شرکت گوگل در طول دوره‌ی تست (از تاریخ 19-11-2020 تا 01-08-2022) در مقابل عملکرد پایه‌ی خرید و نگهداری^۲ سهام در بک‌تست در همان دوره، از معیارهای تشریح شده در این بخش استفاده می‌کنیم. لازم به ذکر است، برای دستیابی به برآوردهای بهتر از عملکرد روش‌ها می‌توان از K-fold cross-validation^۳ استفاده کرد اما با توجه به این که آموزش مدل‌های یادگیری تقویتی عمیق بسیار زمانبر بوده و نیاز به منابع پردازشی قوی دارد، اجرای این نوع از ارزیابی بسیار هزینه‌بر بوده و ارزیابی روش‌ها فقط در یک دوره از داده‌های تست انجام شده. در ادامه به تشریح ارزیابی می‌پردازیم:

1-4- معیارهای سنجش

- بازده متوسط سالانه مرکب^۴: بازده متوسط مرکب سالانه نرخ بازگشت سرمایه را با در نظر گرفتن اثر مرکب سرمایه گذاری برای هر سال است. این معیار بسیار دقیق‌تری برای عملکرد نسبت به میانگین حسابی بازده سالانه است زیرا میانگین هندسی اثر ترکیبی بازده را مورد توجه قرار می‌دهد و مقدار بازده متوسط موثر را به ما خواهد داد. این معیار به صورت زیر محاسبه می‌شود:

$$R_{ann} = \left[\prod_{i=1}^n (1 + R_i) \right]^{252/n} - 1 \quad (1-4)$$

که در این رابطه، R_i بازده روزانه در روز i ام می‌باشد.

¹ Backtest

² Buy and Hold

³ پروسه‌ای که طی آن مجموعه داده‌ها به K قسمت مساوی تقسیم شده و برای K بار مدل آموزش داده می‌شود که در هر بار از آن، مجموعه $K-1$ قسمت مشخص از داده‌ها به عنوان داده‌ی آموزش در نظر گرفته شده و یک قسمت باقی مانده به عنوان داده‌های تست. و در نهایت میانگین K حاصل ارزیابی روی داده‌های تست گزارش می‌شود.

⁴ Compounded Annual Return

- بازده تجمعی^۱: بازده تجمعی یا انباشته تغییر کل در قیمت سرمایه گذاری در یک مدت زمان معین است.

$$R_{cum} = \left[\prod_{i=1}^n (1 + R_i) \right] - 1 \quad (2-4)$$

- نوسان سالانه^۲ (σ_{ann}): برای محاسبه نوسانات روزانه انحراف استاندارد بازده‌های روزانه را می‌توان محاسبه کرد. و برای محاسبه‌ی نوسان سالانه، با فرض وجود 252 روز معاملاتی در طی یک سال، می‌توان نوسان روزانه‌ی محاسبه شده را در جذر 252 ضرب کرد.

$$\sigma_{daily} = \sqrt{\frac{\sum_{i=1}^n (R_i - \bar{R})^2}{n - 1}}$$

$$\sigma_{ann} = \sigma_{daily} \sqrt{252} \quad (3-4)$$

- نسبت شارپ^۳ (S): نسبت شارپ (Sharpe, 1994) بازده سرمایه گذاری را با توجه به ریسک آن مقایسه می‌کند. در صورت کسر نسبت شارپ، تفاوت بین بازده تحقق یافته یا مورد انتظار و بازده معیاری پایه مانند نرخ بازده بدون ریسک (نرخ سود بانکی) است. و مخرج آن انحراف استاندارد بازده در بازه زمانی یکسان است. این نسبت معیاری برای بازده تعدیل شده با ریسک است و نشان می‌دهد که چه مقدار بازده اضافی برای نوسانات نگهداری یک دارایی دارای ریسک دریافت می‌کنید. نرخ بازده دارایی نقد در این نسبت برابر 3.5٪ سالانه، معادل نرخ اوراق قرضه‌ی شش ماهه‌ی خزانه داری دولت ایالات متحده، در زمان این پژوهش لحاظ شده. در مقایسه‌ی پورتفولیوها و الگوریتم‌های معاملاتی، نسبت شارپ بالاتر مطلوب‌تر است. این نسبت به صورت زیر محاسبه می‌شود:

¹ Cumulative Return

² Annual Volatility

³ Sharpe ratio

$$S = \frac{\bar{R} - R_f}{\sigma_{daily}}$$

$$S_{ann} = S\sqrt{252} \quad (4-4)$$

برای محاسبه‌ی نسبت شارپ سالانه، با فرض وجود 252 روز معاملاتی در طی یک سال، می‌توان نسبت شارپ روزانه‌ی محاسبه شده را در جذر 252 ضرب کرد.

- نسبت کالمار¹ (C): نسبت کالمار معیاری از بازده تعدیل شده با ریسک برای سبدها و الگوریتم‌های سرمایه گذاری است. نسبت کالمار از بیشینه افت به عنوان معیار ریسک خود استفاده می‌کند که آن را منحصر به فرد می‌کند.

$$C = \frac{R_{ann}}{Max DrawDown} \quad (5-4)$$

- بیشینه افت²: درصد اختلاف بالاترین ارزش دارایی (یا پورتفولیو) با پایین ترین ارزش بعد از آن است.

$$Max DrawDown = \frac{H - L}{L} \quad (6-4)$$

در این رابطه H بالاترین ارزش دارایی‌های الگوریتم، و L پایین ترین ارزش ثبت شده بعد از آن تا قبل از اوج بعدی است.

- نسبت سورینتو³: نسبت سورینتو مدل تغییر یافته‌ای از نسبت شارپ است و با نسبت شارپ از این جهت متفاوت است که فقط انحراف استاندارد نزولی (انحراف استاندارد از بازده‌های منفی) را در نظر می‌گیرد نه کل ریسک (صعودی و نزولی). از آنجایی که نسبت سورینتو تنها بر انحراف منفی بازده، دید بهتری از عملکرد تعدیل شده با ریسک (نسبت شارپ) ارائه می‌دهد. نرخ بازده

¹ Calmar ratio

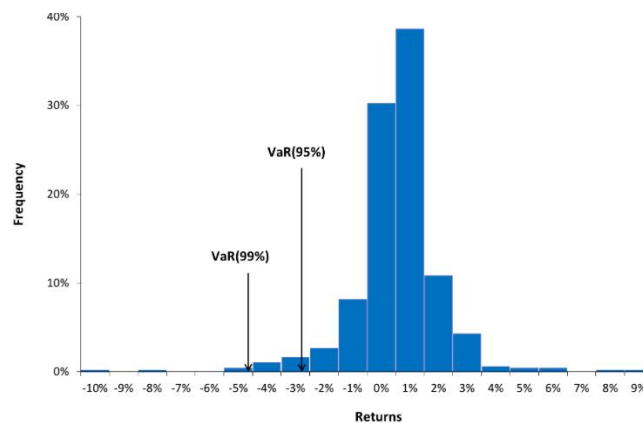
² Maximum drawdown

³ Sortino ratio

دارایی نقد در این نسبت نیز برابر 3.5٪ سالانه، معادل نرخ اوراق قرضه‌ی شش ماهه‌ی خزانه داری دولت ایالات متحده، در زمان این پژوهش لحاظ شده.

$$Sr = \frac{\bar{R} - R_f}{\sigma_{down}} \quad (7-4)$$

- در این رابطه σ_{down} انحراف معیار روبه پایین (فقط مربوط به مقادیر منفی) بازده‌ها است.
- ارزش روزانه در معرض خطر¹ (VAR): ارزش در معرض خطر آماری است که در حوزه مدیریت ریسک برای پیش‌بینی بیشترین زیان ممکن در یک بازه زمانی خاص استفاده می‌شود. VAR توسط سه متغیر تعیین می‌شود: یک دوره زمانی خاص، یک سطح اطمینان و اندازه ضرر احتمالی. یکی از روش‌های محاسبه VAR بر اساس داده‌های تاریخی است، با این فرض که بازده‌های آینده تقریباً مشابه خواهند بود. روش واریانس-کوواریانس فرض می‌کند که بازده از توزیع آماری نرمال پیروی می‌کند. ارزش در معرض خطر هم می‌تواند به صورت مقدار و هم به صورت درصد گزارش شود، در این پژوهش از حالت درصدی این گزارش استفاده می‌کنیم.
- به طور خاص، به این پرسش پاسخ می‌دهیم که حداکثر درصدی که می‌توانیم انتظار از دست دادن آن را - عموماً با اطمینان 95 یا 99 درصد - در یک سال (یا هر بازه زمانی دیگری) داشته باشیم چقدر است. به صورت دقیق تر، مقدار بازده ای است که به ازای آن سطح زیر نمودار توزیع بازده‌های روزانه، از سمت چپ نمودار به 5 یا 1 درصد برسد.



شکل 1-4: نحوه محاسبه ارزش در معرض ریسک با داده‌های تاریخی بازده‌ها.

¹ Daily value at risk (VAR)

4-2- نتایج ارزیابی

از این جهت که الگوریتم‌های آموزش داده شده از سیاست‌های احتمالی برای تعامل یا محیط خود استفاده می‌کنند - به این معنی که از یک تابع توزیع احتمال به عنوان تابع سیاست استفاده می‌کنند و اعمال خود را از این توابع نمونه گیری می‌کنند - در برخورد های متفاوت، نتایج یکسان و قطعی نخواهند داشت. از این رو در فاز ارزیابی مدل‌ها، هر یک از الگوریتم‌ها را به صورت جدا به تعداد 500 تکرار در بازه‌ی تاریخی ارزیابی اجرا می‌کنیم، نتایج بک‌تست هر یک از عامل‌ها را در هر تکرار ثبت و برآورد نقطه‌ای و بازه‌ی اطمینان آن‌ها به همراه بازه‌ی نوسان 3σ شاخص‌ها را به عنوان نتایج نهایی ارزیابی گزارش می‌کنیم.

پس از اجرای بک تست عامل‌ها بر روی سهام شرکت گوگل در بازه‌ی تاریخی تست از تاریخ 11-19-2020 تا 01-08-2022، نتایج به صورت گزارش شده به صورت زیر بوده:

جدول 1-4 نتایج بک تست الگوریتم‌ها در بازه‌ی تست.

مدل	معیار	برآورد نقطه‌ای	حد پایین اطمینان (95٪)	حد بالای اطمینان (95٪)	حد پایین 3σ	حد بالای 3σ	صدک 5 (بدترین سناریو)
PPO	بازده متوسط سالانه مرکب	0.457	0.435	0.48	0.077	0.838	0.079
	بازده تجمعی	0.853	0.807	0.898	0.071	1.635	0.129
	نوسان سالانه	0.292	0.292	0.292	0.288	0.296	0.287
	نسبت شارپ	1.269	1.216	1.321	0.375	2.161	0.286
	نسبت کالمار	2.706	2.527	2.886	-0.361	5.774	0.315
	بیشینه افت	-0.203	-0.208	-0.197	-0.296	-0.109	-0.33
	نسبت سورینتو	1.958	1.870	2.045	0.483	3.458	0.396
	VAR	-0.035	-0.035	-0.035	-0.036	-0.034	-0.036
DQL	بازده متوسط سالانه مرکب	0.525	0.521	0.530	0.447	0.603	0.441
	بازده تجمعی	0.965	0.956	0.975	0.805	1.125	0.794
	نوسان سالانه	0.219	0.218	0.219	0.215	0.222	0.216
	نسبت شارپ	1.881	1.867	1.895	1.638	2.124	1.607
	نسبت کالمار	4.531	4.473	4.589	3.539	5.523	3.388
	بیشینه افت	-0.117	-0.118	-0.116	-0.132	-0.102	-0.135
	نسبت سورینتو	3.060	3.032	3.087	2.595	3.525	2.539
	VAR	-0.026	-0.026	-0.026	-0.026	-0.025	-0.026

مقایسه‌ی برآورد نقطه‌ای نتایج بک تست عامل‌ها با عملکرد پلایه‌ی خرید و نگهداری نیز در جدول زیر نمایش داده شده:

جدول 4-2 مقایسه‌ی نتایج مدل‌ها با عملکرد پایه‌ی خرید و نگهداری.

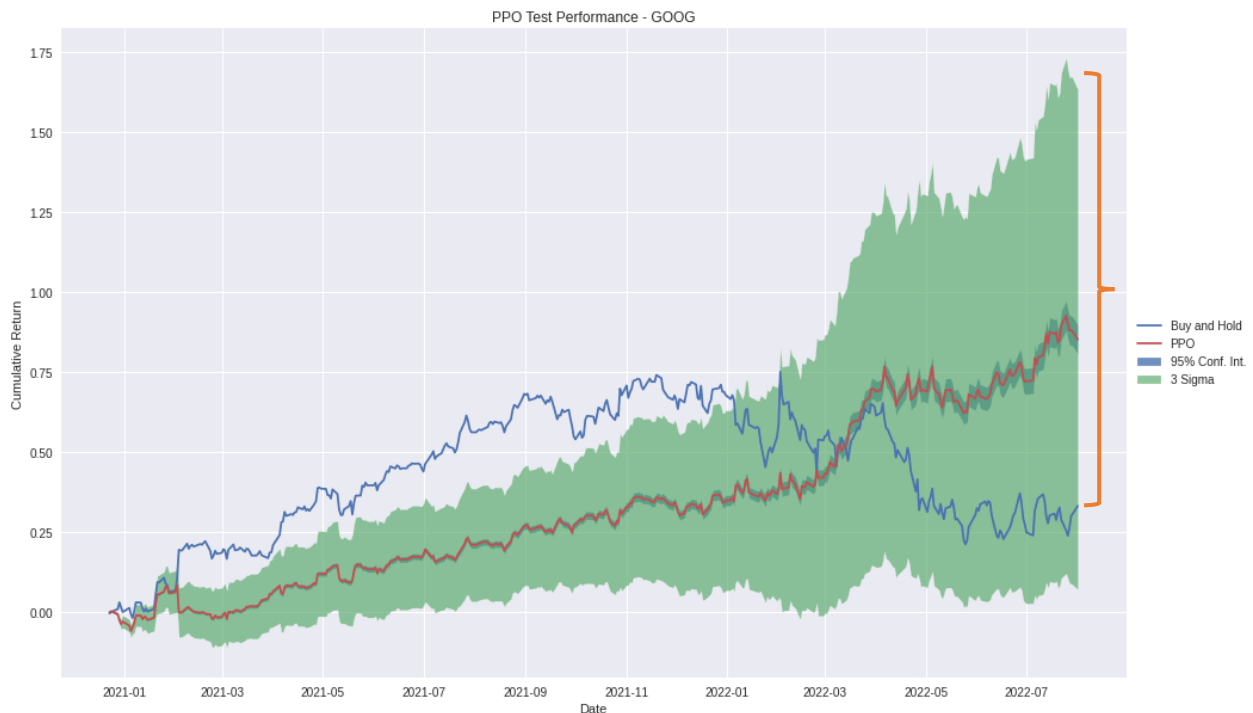
معیار	PPO	DQL	Buy & Hold
بازده متوسط سالانه مرکب	0.457	0.525	0.195
بازده تجمعی	0.853	0.965	0.332
نوسان سالانه	0.292	0.219	0.297
نسبت شارپ	1.269	1.881	0.749
نسبت کالمار	2.706	4.531	0.636
بیشینه افت	-0.203	-0.117	-0.307
نسبت سورینتو	1.958	3.060	1.123
VAR	-0.035	-0.026	-0.365

پس از بررسی نتایج حاصل شده از بک‌تست مدل‌ها و مقایسه‌ی آن‌ها با عملکرد پایه، عملکرد بهتر الگوریتم DQL نسبت به PPO و عملکرد پایه در تمامی معیارها کاملاً مشهود است. با این وجود PPO همچنان توانسته عملکرد بهتری نسبت به خرید و نگهداری داشته باشد.

4-2-1- بررسی عملکرد عامل PPO

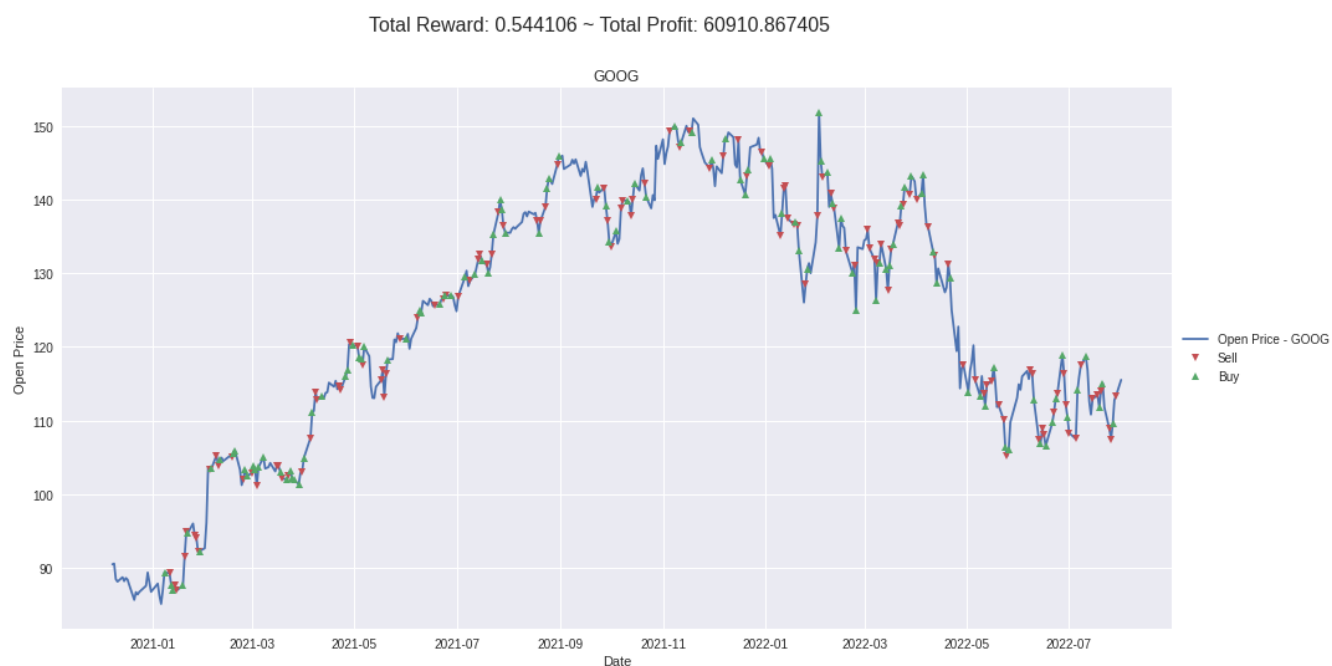
نمودار عملکرد بازده تجمعی PPO در دوره‌ی تست شکل 4-2-1 در ادامه قرار داده شده، با توجه به شکل 4-2-1 و همچنین جدول 4-1 واریانس و عدم قطعیت عملکرد بالایی در ارزیابی عامل PPO مشاهده می‌شود و مستحکم و قابل اطمینان نبوده، با این حال همانطور که در شکل نشان داده شده، بازده تجمعی PPO در انتهای بازه‌ی تست توانسته بیشتر از 80 درصد مواقع عملکرد بهتری از خرید و نگهداری داشته باشد. البته لازم به ذکر است که عملکرد PPO در نیمه‌ی ابتدایی بازه‌ی تست بسیار پایینتر از خرید و نگهداری است که هم می‌تواند ناشی از ضعف عملکردی و هم محدودیت‌های آموزش مدل (زمان، منابع پردازشی و غیره) باشد.

یکی از راه‌های پیشنهادی برای برخورد با واریانس بالای نتایج در حین استنتاج و بهره‌گیری از مدل در دنیای واقعی، استفاده از چندین نمونه‌ی یکسان از عامل به صورت موازی و میانگین‌گیری از تصمیمات و اعمال این عامل‌ها در هر مرحله است.

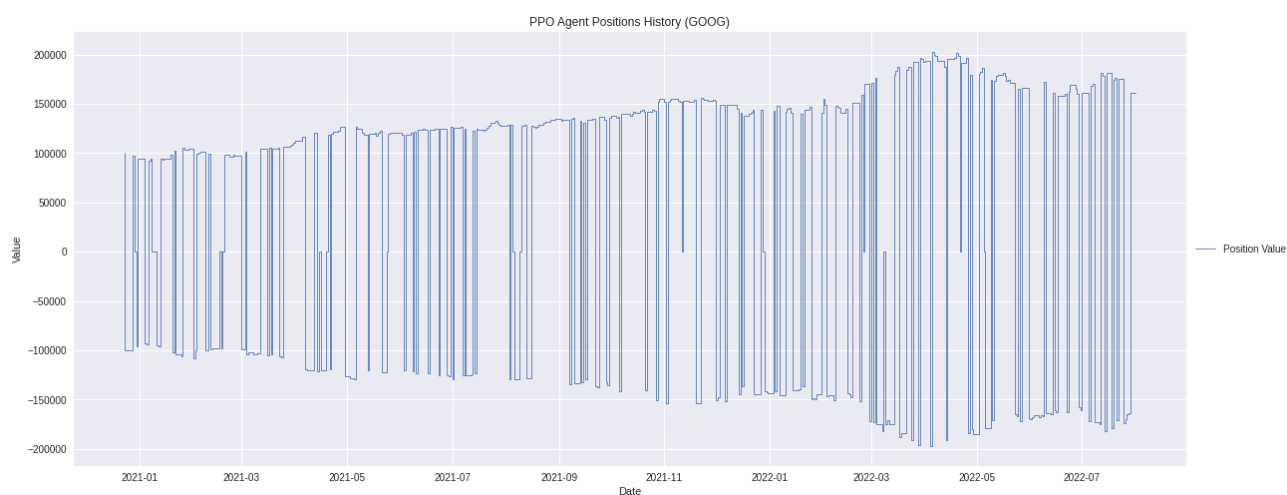


شکل 1-2-4: عملکرد و بازده تجمعی عامل PPO در بازه‌ی تست.

در شکل 2-2-4 تصمیمات معاملاتی عامل PPO در سطوح قیمتی مختلف در حین یک تکرار از 500 تکرار از ارزیابی در دوره‌ی تست مصور شده است و همچنین در شکل 3-2-4 مقادیر ارزش موقعیت‌های اتخاذ شده‌ی متناظر با آن نیز رسم شده است. با بررسی این دو نمودار می‌توان مشاهده کرد که عامل تمایل کمتری به نگهداری وجه نقد داشته و در بیش از 90 درصد مواقع در موقعیت خرید یا فروش استقرازی کامل قرار گرفته.

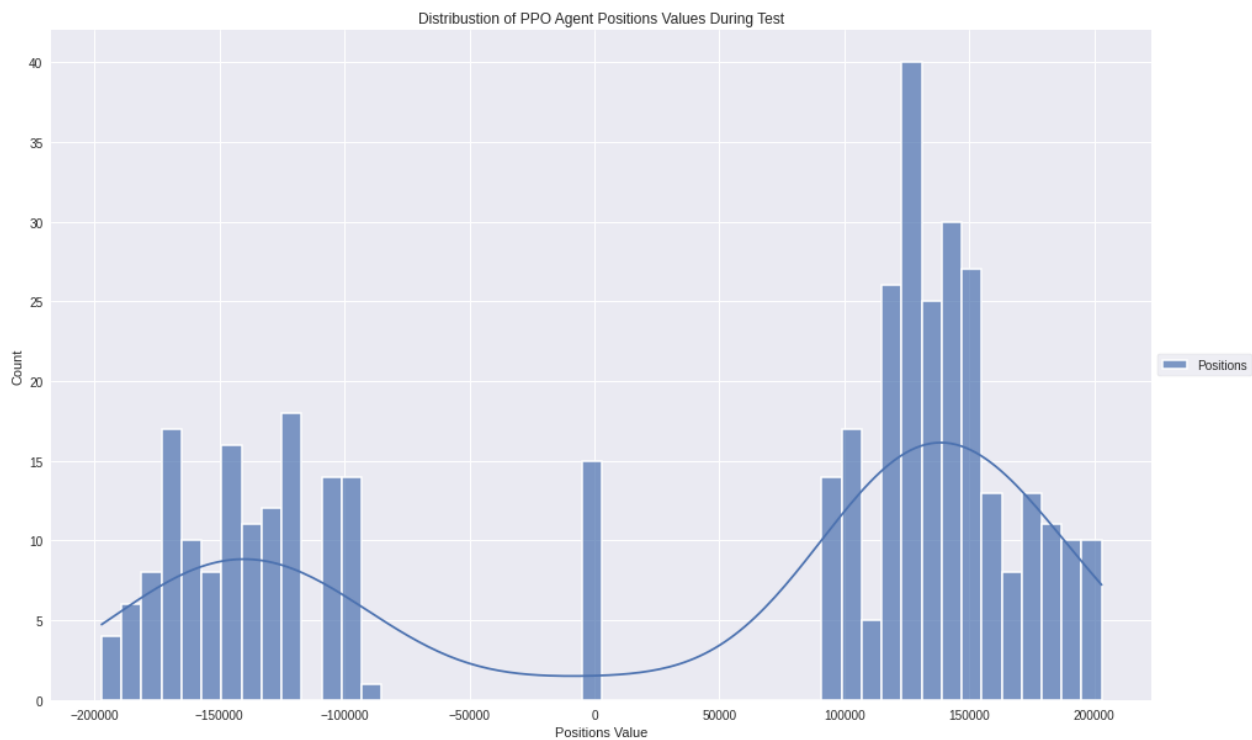


شکل 2-2: تصمیمات معاملاتی PPO با توجه به قیمت سهام در یک تکرار از مرحله‌ی ارزیابی.



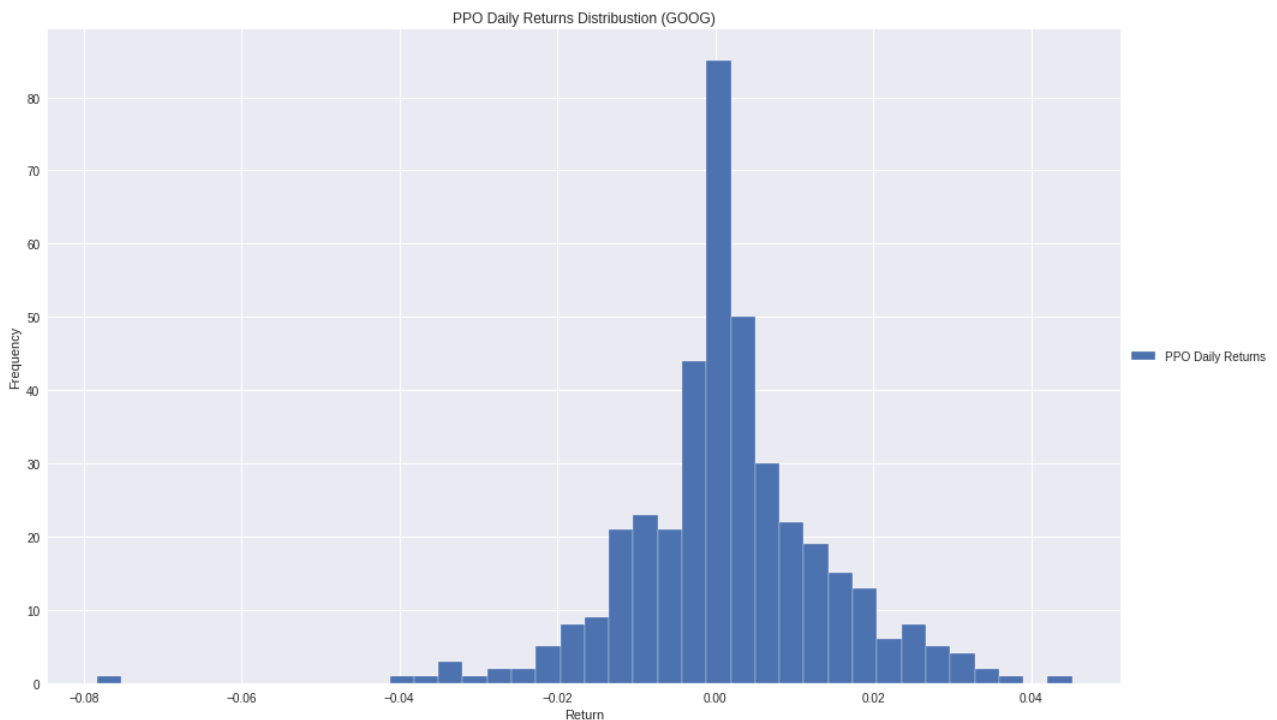
شکل 2-3: موقعیت‌های معاملاتی اتخاذ شده توسط PPO در یک تکرار از مرحله‌ی ارزیابی.

همچنین در شکل 4-2-4 هیستوگرام توزیع ارزش موقعیت‌های اتخاذ شده توسط عامل PPO در یک تکرار از ارزیابی نیز تصویر سازی شده، در این نمودار مقادیر منفی موقعیت‌ها نشان دهنده‌ی موقعیت فروش اسقراضی سهام است. این نمودار نشان می‌دهد تعداد موقعیت‌های خرید بسیار بیشتر از موقعیت‌های فروش اسقراضی و دارای نقد بوده.



شکل 4-2-4: توزیع ارزش موقعیت‌های اتخاذ شده توسط عامل PPO در یک تکرار از ارزیابی.

در شکل 4-2-5 نیز هیستوگرام توزیع بازده‌های روزانه‌ی عامل PPO به تصویر درآمده. قابل مشاهده است که بازده‌های روزانه‌ی عامل به طور شهودی از توزیع گوسی نرمال پیروی می‌کند. می‌توان در این نمودار دید که بازده‌های روزانه‌ی مثبت بیشتری نسبت به بازده‌های منفی در عملکرد عامل وجود داشته. همچنین تعدادی مقادیر پرت به خصوص در بازه‌ی منفی مشاهده می‌شود که می‌تواند نشان دهنده‌ی استحکام کم عملکرد عامل باشد.



شکل 4-2-5: توزیع بازده روزانه عامل PPO در مرحله‌ی ارزیابی.

4-2-2-2- بررسی عملکرد عامل DQL

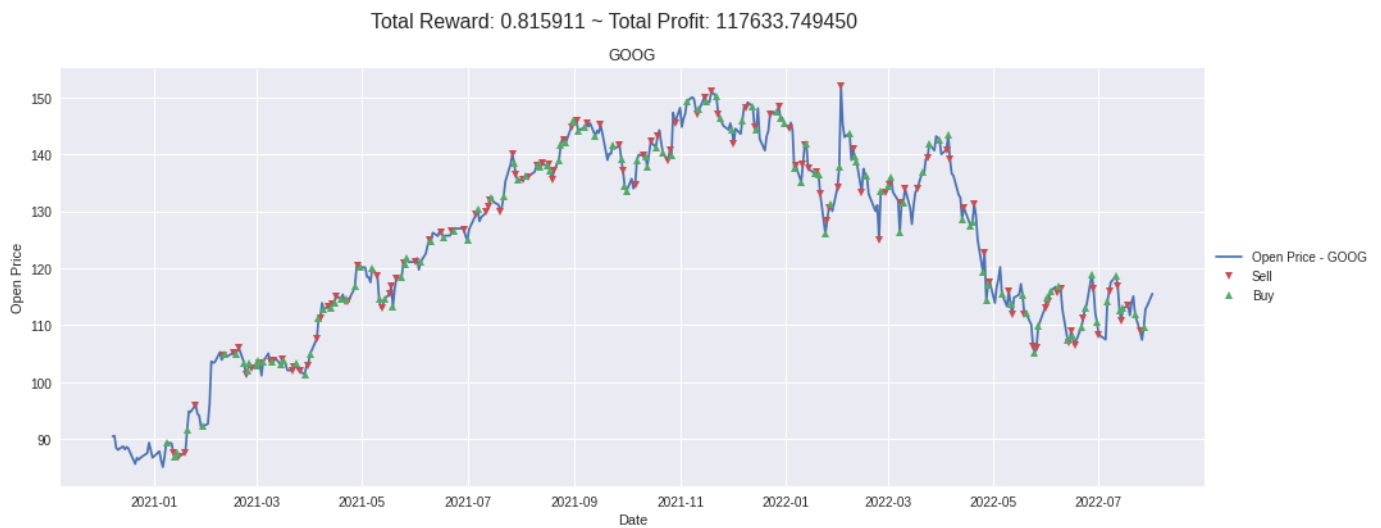
نمودار عملکرد بازده تجمعی عامل DQL در دوره‌ی تست نیز در ادامه قرار داده شده، با توجه به شکل 4-2-6 و جدول 4-1 می‌توان دریافت که این عامل نسبت به عامل PPO واریانس و عدم قطعیت کمتری در عملکرد دارد و نسبت به PPO بسیار قابل اطمینان‌تر و مستحکم‌تر بوده. در نمودار 4-2-6 دیده می‌شود که DQL در بدترین حالت از دامنه‌ی عملکرد خود نیز بازده تجمعی بسیار بهتری (نزدیک به 50 درصد بالاتر) از خرید و نگهداری داشته و همچنین در کل طول دوره‌ی تست نیز عملکرد قابل اطمینانی از منظر بازده تجمعی در مقایسه با خرید و نگهداری داشته.

اما همچنان می‌توان همانند PPO برای این عامل نیز در حین استنتاج و بهره‌گیری از مدل در دنیای واقعی از چندین نمونه از عامل به صورت موازی برای کاهش واریانس استفاده کرد و یا از سیاست حریصانه برای استنتاج استفاده کرد.



شکل 4-2-6: عملکرد و بازده تجمعی عامل DQL در بازه‌ی تست.

در شکل 4-2-7 تصمیمات معاملاتی عامل DQL در سطوح قیمتی مختلف در حین یک تکرار از 500 تکرار از ارزیابی در دوره‌ی تست مصور شده است و همچنین در شکل 4-2-8 مقادیر ارزش موقعیت‌های اتخاذ شده‌ی متناظر با آن نیز رسم شده است. با بررسی این دو نمودار نیز می‌توان مشاهده کرد که عامل DQL بسیار تمایل بیشتری به نگهداری وجه نقد داشته و در بیش از 60 درصد مواقع در موقعیت نگهداری دارایی نقد قرار گرفته. و همچنین در باقی موارد نیز تعداد موقعیت‌های خرید با نسبت چند برابری، بسیار بیشتر از تعداد موقعیت‌های فروش استقرازی بوده.



شکل 4-2-7: مصورسازی تصمیمات معاملاتی DQL با توجه به قیمت سهام در یک تکرار از مرحله‌ی ارزیابی.

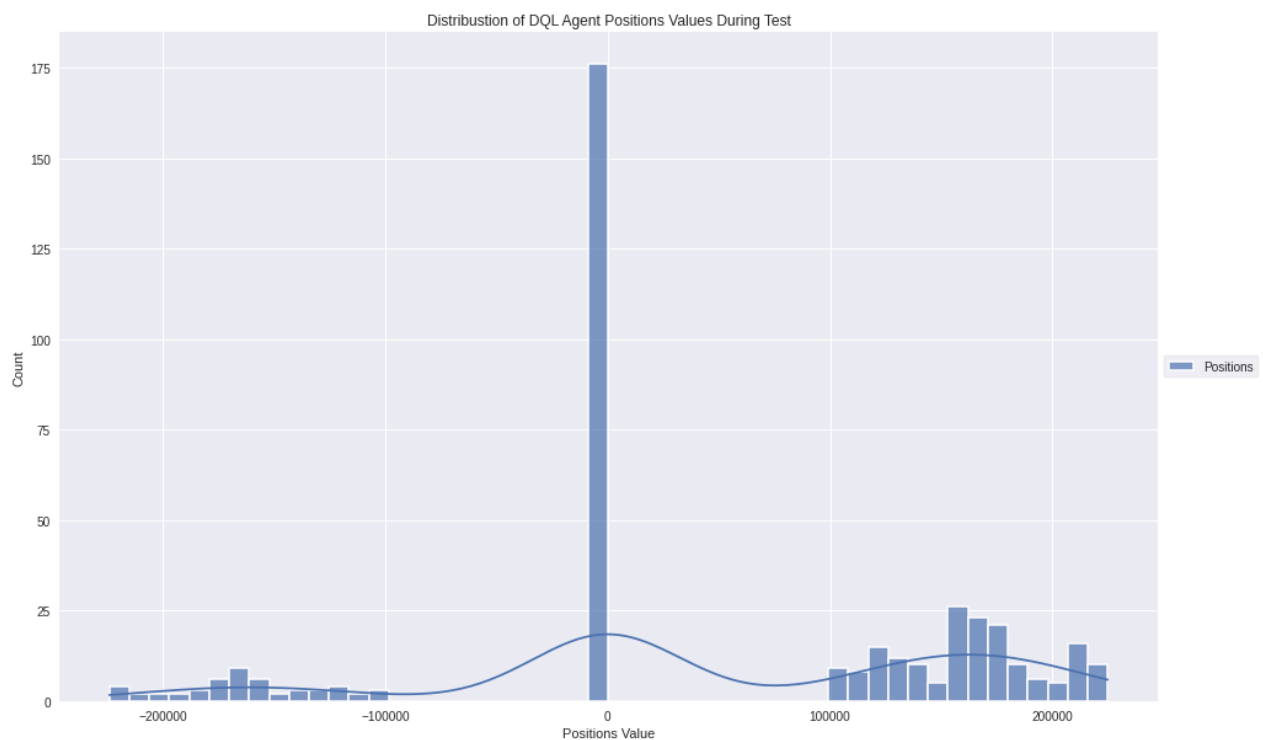


شکل 4-2-8: ارزش موقعیت‌های اتخاذ شده توسط DQL در یک تکرار از مرحله‌ی ارزیابی.

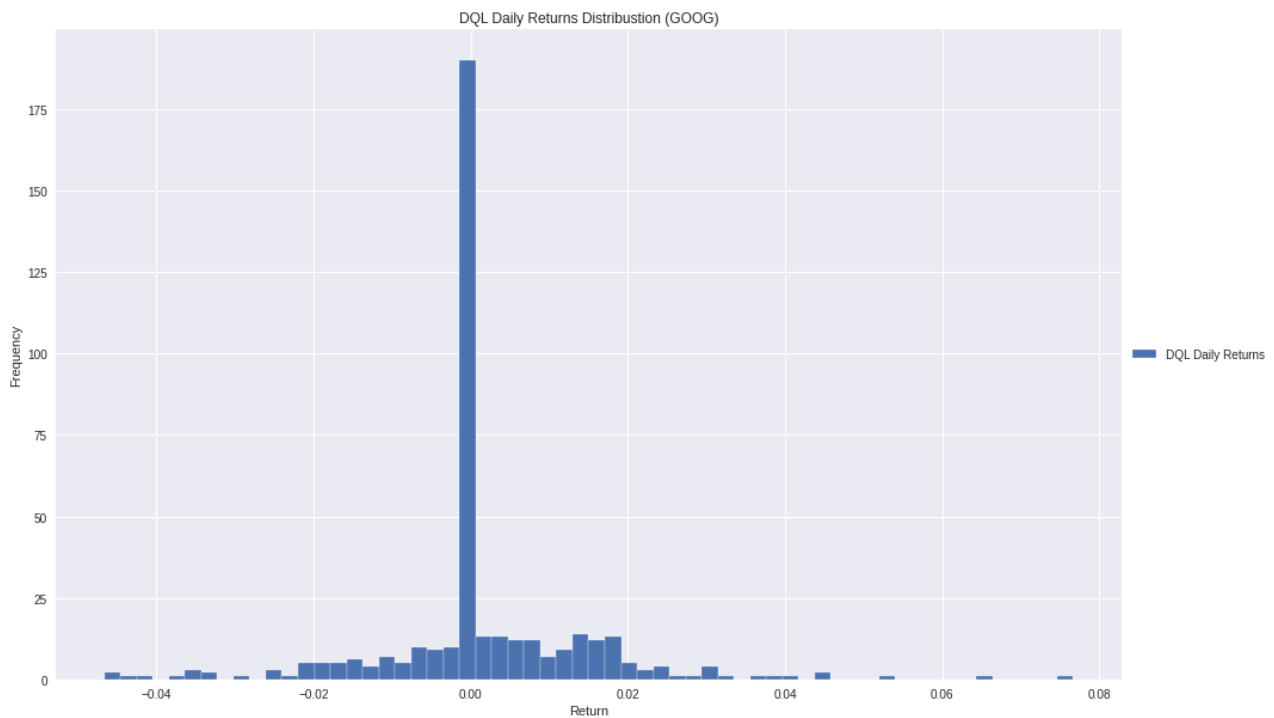
در شکل 4-2-9 توزیع ارزش موقعیت‌های اتخاذ شده توسط عامل DQL در یک تکرار از ارزیابی تصویر سازی شده، در این نمودار مقادیر منفی موقعیت‌ها نشان دهنده‌ی موقعیت فروش اسقراضی سهام است. در این شکل نیز نسبت بسیار بالای موقعیت‌های دارایی نقد نسبت به موقعیت‌های خرید و فروش و همچنین بیشتر بودن موقعیت‌های خرید نسبت به فروش مشهود است. از این شکل می‌توان برداشت کرد

که عامل DQL تمایل بسیار کمتری در معاملات پرریسک داشته و با وجود عدم تعریف معیار ریسک در تابع پاداش، نسبت به PPO بسیار کم ریسک تر عمل می‌کند.

در شکل 4-2-10 نیز هیستوگرام توزیع بازده‌های روزانه‌ی عامل DQL به تصویر درآمده. در این تصویر نیز قابل مشاهده است به طور قابل توجهی بازده‌های روزانه‌ی DQL حول صفر متمرکز بوده و دارای نوسان (واریانس) بسیار کمی می‌باشد. با این وجود فراوانی بازده‌های مثبت از بازده‌های روزانه‌ی منفی بیشتر بوده و همین باعث عملکرد نهایی بسیار بهتری نسبت به خرید و نگهداری شده.



شکل 4-2-9: توزیع و هیستوگرام ارزش موقعیت‌های اتخاذ شده توسط عامل DQL در یک تکرار از ارزیابی.

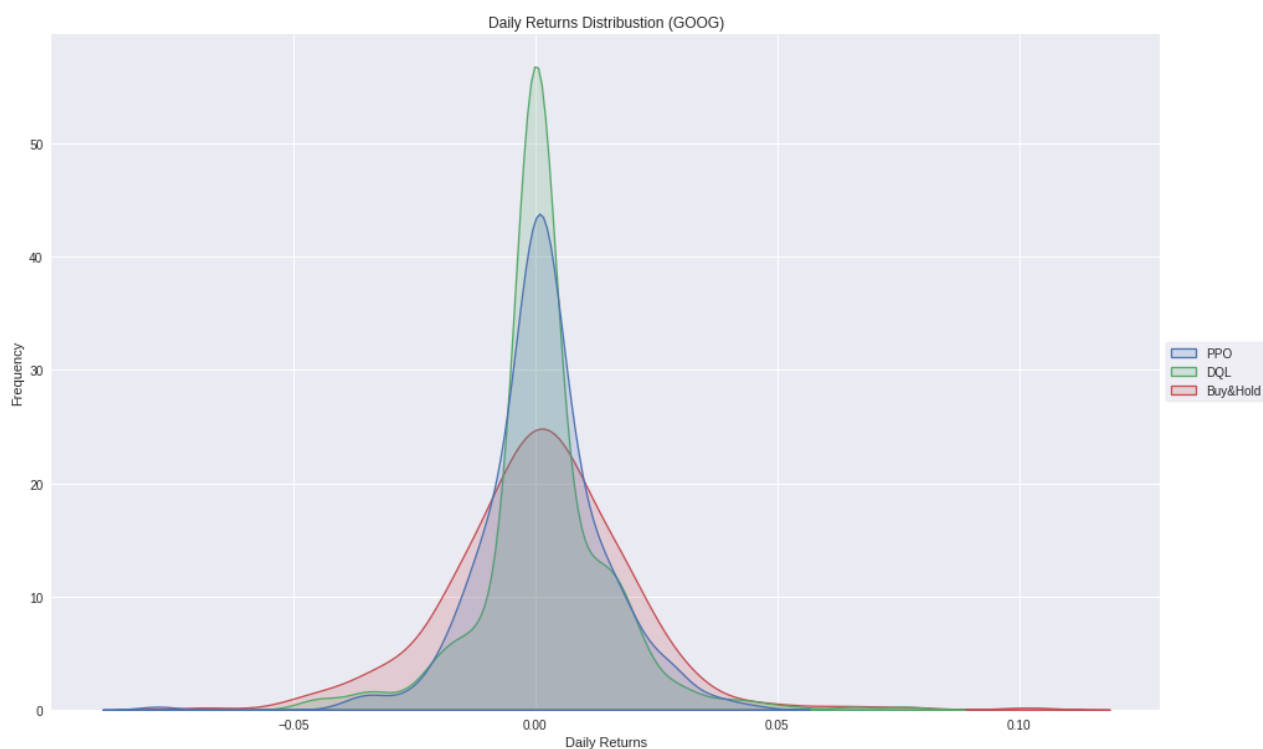


شکل 10-2-4: توزیع بازده روزانه عامل DQL در مرحله‌ی ارزیابی.

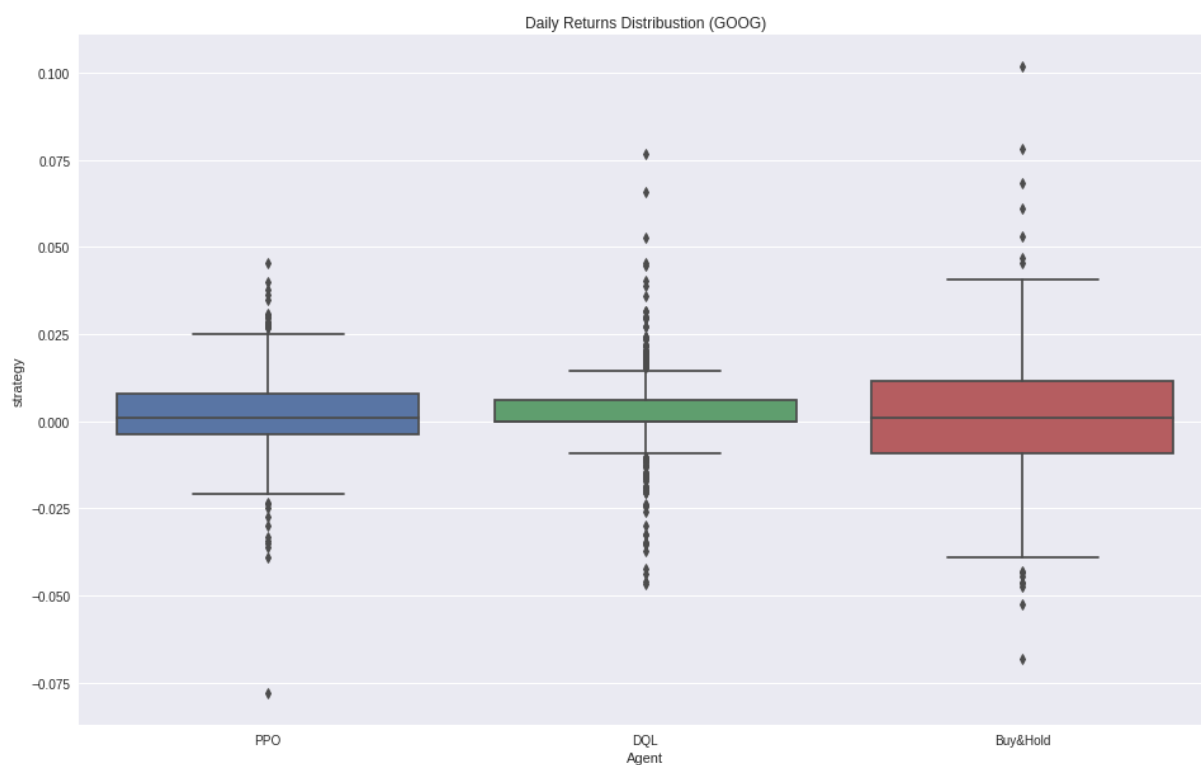
4-2-3- مقایسه‌ی عملکرد استراتژی‌ها

در شکل‌های 4-2-11 تخمین توزیع بازده روزانه‌ی دو روش PPO و DQL و همچنین استراتژی خرید و نگهداری به تصویر درآمده. از این نمودار برداشت می‌شود که عامل PPO بازده روزانه‌ی با واریانس کمتری نسبت به عامل DQL و استراتژی خرید و نگهداری داشته، و توزیع بازده روزانه DQL و استراتژی خرید و نگهداری بسیار نزدیک به هم بوده. می‌توان کم‌ریسک‌تر بودن و بازده بالاتر DQL را در این نمودار نیز مشاهده کرد.

در شکل‌های 4-2-12 نیز به ترتیب نمودار جعبه‌ای بازده‌های روزانه در مرحله‌ی ارزیابی برای سه استراتژی رسم شده. در این نمودارها مشاهده می‌شود که الگوریتم‌های یادگیری تقویتی عمیق بازده‌های پرخطر روزانه‌ی بیشتری نسبت به خرید و نگهداری داشته اند اما واریانس بازده‌های روزانه‌ی PPO و DQL بسیار کمتر بوده. همچنین بازده‌های روزانه‌ی DQL در نزدیکی صفر بسیار متمرکزتر هستند. در نهایت، بازده تجمعی هر سه استراتژی در کنار یکدیگر در شکل 4-2-13 مصور شده. با توجه به این نمودار عامل DQL در شش ماه انتهای سال 2022 عملکرد بهتری نسبت به دو استراتژی دیگر داشته.



شکل 4-2-11: مقایسه توزیع بازده روزانه عامل‌های PPO و DQL و استراتژی خرید و نگهداری در مرحله‌ی ارزیابی.

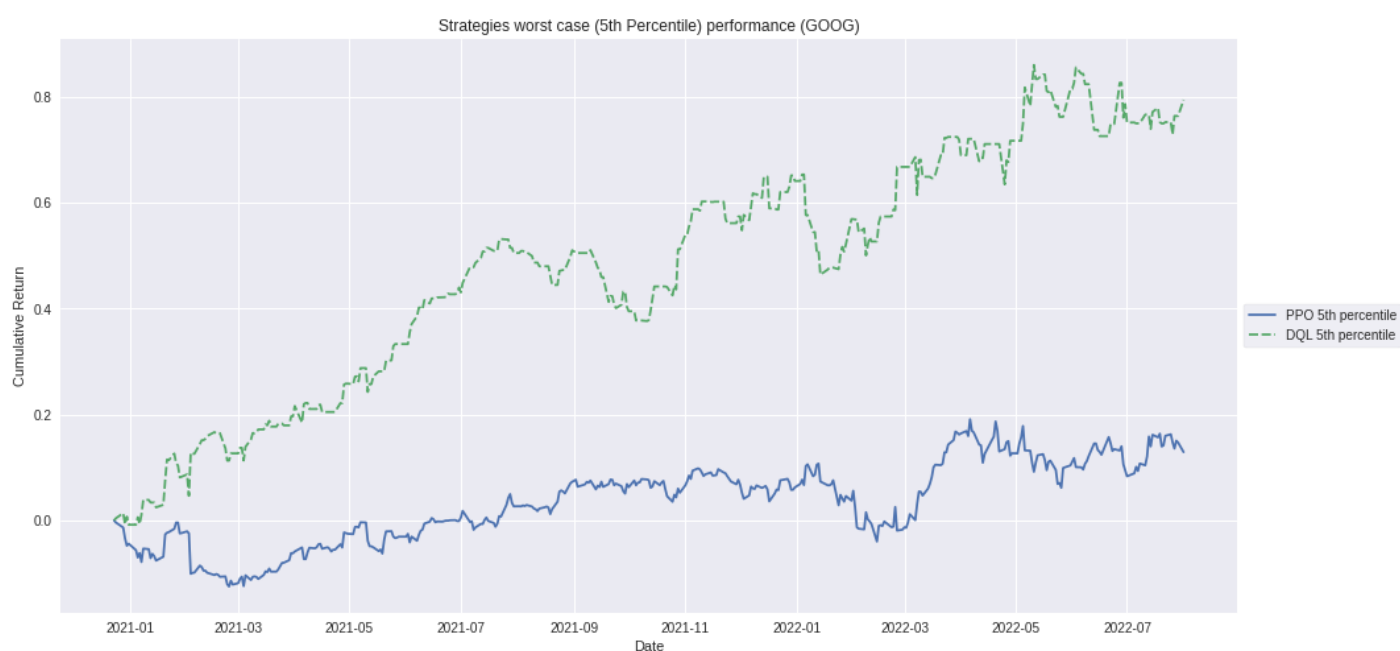


شکل 4-2-12: مقایسه نمودار جعبه‌ای بازده روزانه عامل‌های PPO و DQL و استراتژی خرید و نگهداری در مرحله‌ی ارزیابی.

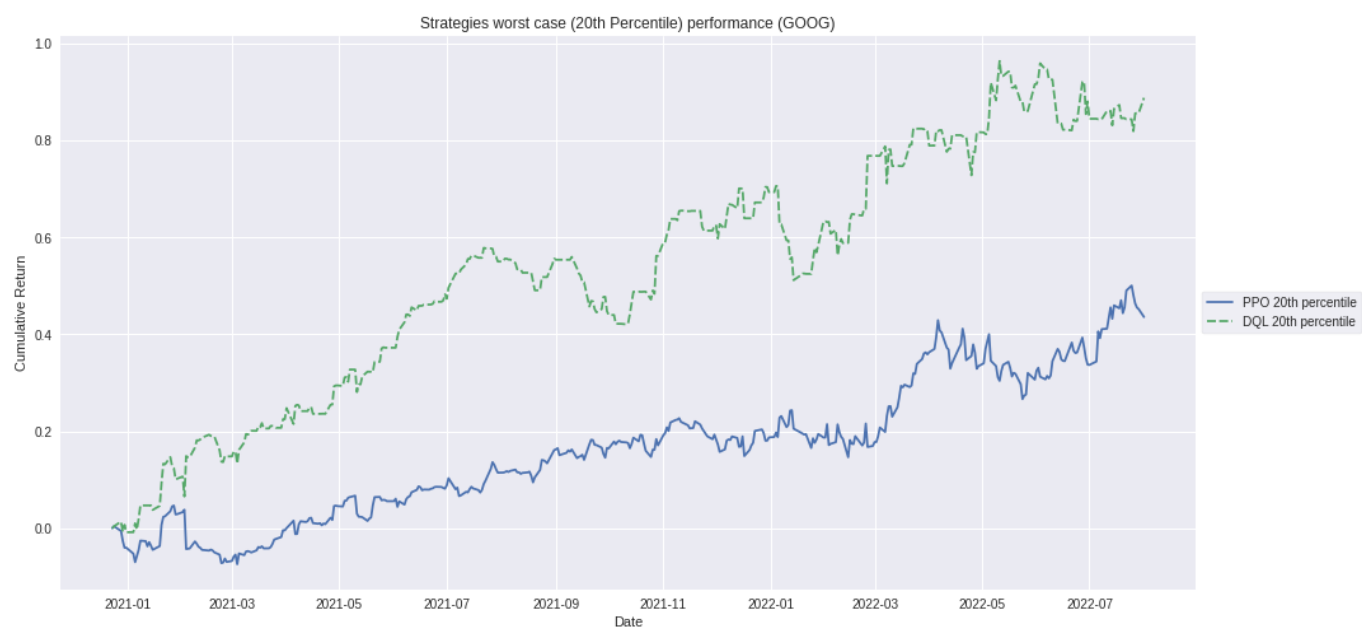


شکل 4-2-13: مقایسه‌ی میانگین عملکرد هر سه استراتژی در کنارهم.

در نهایت در شکل‌های 4-2-14 و 4-2-15 عملکرد دو مدل PPO و DQL در بدترین سناریوها، با استفاده از صدک‌های پنجم و بیستم عملکرد مدل‌ها در 500 تکرار از ارزیابی انجام شده مقایسه شدند. با این بررسی نشان داده می‌شود که روش DQL بسیار قابل اطمینان‌تر از PPO عمل می‌کند. و در بدترین حالات نیز عملکرد بهتری از خرید و نگهداری خواهد داشت.



شکل 4-2-14: مقایسه‌ی بدترین سناریو (صدک پنجم) عملکرد هر دو عامل در کنارهم.



شکل 15-2-4: مقایسه‌ی بدترین سناریو (صدک بیستم) عملکرد هر دو عامل در کنارهم.

5- فصل پنجم

نتیجه‌گیری و پیشنهادات آتی

نتیجه‌گیری و پیشنهادات آتی

در این پژوهش به بررسی، پیاده‌سازی و ارزیابی دو نوع از الگوریتم‌های یادگیری تقویتی عمیق به طور جداگانه که از پارادایم‌های اخیر هوش مصنوعی است، بر روی داده‌های بازار سهام با هدف آموزش یک عامل هوشمند با قابلیت اتخاذ تصمیم‌ها و موقعیت‌های معاملاتی به صورت مستقیم در سیستم‌های معاملات الگوریتمی پرداخته شده است. در طی روند این پژوهش ابتدا دو عامل با الگوریتم‌های PPO و DQL با داده‌های قیمت روزانه سهام شرکت گوگل به همراه داده‌های بازده روزانه و تعدادی از شاخص‌های تکنیکال آموزش داده شدند تا در ابتدای هرروز موقعیت معاملاتی مربوطه را اتخاذ کنند. پس از آموزش، عملکرد این عامل‌ها بر روی داده‌های بازه‌ی تست مورد بررسی قرار گرفت و مشاهده شد که این عامل‌ها قادر به این هستند که در شاخص‌های عملکردی در نظر گرفته شده نسبت به استراتژی خرید و نگهداری عملکرد بهتری داشته باشد. در نهایت نیز عملکرد این عامل‌ها به صورت دقیق مورد بررسی و مقایسه قرار گرفت. به طور خاص، مدل DQL توانست با اختلاف بسیاری هم از مدل PPO و هم از استراتژی خرید و نگهداری در همه‌ی معیارها بهتر عمل کند و همچنین نتایج قابل اطمینان‌تر و مستحکم‌تری داشته باشد و با ریسک بسیار کمتری به معامله بپردازد. همچنین عامل PPO نیز توانست به طور کلی عملکرد بهتری از خرید و نگهداری داشته باشد اما با این وجود نتایج با قابلیت اطمینان و استحکام کمتری از آن حاصل شد.

1-5- پیشنهادات آتی

امروزه یادگیری تقویتی عمیق، بیشتر و بیشتر مورد توجه محققان و شرکت‌های صنعتی قرار می‌گیرد. این گروه از روش‌های زیرمجموعه‌ی هوش مصنوعی پتانسیل بسیار زیادی در مسائل حوزه‌های بهینه‌سازی، کنترل و برنامه‌ریزی در حوزه‌های مختلف، به خصوص در حوضه‌ی مالی دارند. از آنجا که روش‌های یادگیری تقویتی قابلیت انعطاف زیادی از جنبه‌ی طراحی مکانیزم پاداش و سیگنال مورد بهینه‌سازی دارند، یکی از مهم‌ترین پیشنهادات برای پژوهش‌های آتی می‌تواند طراحی مکانیزم‌های پاداش بر اساس معیارهای مختلف سرمایه‌گذاری باشد. برای مثال مطالعه در زمینه‌ی آگاه‌سازی عامل‌های هوشمند از جنبه‌ی ریسک تصمیمات و طراحی عامل‌های آگاه به ریسک از طریق طراحی

مکانیزم پاداش صحیح برای کنترل ریسک سرمایه‌گذاری در کنار بیشینه کردن بازدهی سرمایه‌گذاری یکی از این موارد خواهد بود.

از پیشنهادات آتی دیگر نیز می‌توان به پژوهش در زمینه‌ی طراحی عامل‌هایی برای مدیریت سبدهایی متشکل از چندین دارایی به جای مدیریت یک دارایی خاص اشاره کرد. یکی دیگر از چالش‌های این روش‌ها نیز یافتن شاخصه‌های مناسب برای بازنمایی حالت محیط است. به تبع پژوهش در زمینه‌ی شاخصه‌های جاگزین، مانند بار احساسی فضای خبری و سیاسی و داده‌های مکمل دیگر از جمله داده‌های بنیادی و شرکتی سهام مورد معامله و یا داده‌های اقتصادی کلان همچون نرخ بهره و ارز نیز از دیگر زمینه‌های رشد این پژوهش می‌باشد.

و در آخر نیز، مطالعه و آزمایش روش‌ها گروهی و چند عامله (استفاده از چندین عامل مختلف به جای یک عامل خاص، چه به صورت موازی، تعاملی و یا به صورت رقابتی) در طراحی سیستم‌های معاملاتی، می‌تواند مسیری برای توسعه‌ی این پژوهش باشد.

منابع و مراجع

- AlphaBot. (n.d.). *Trading Bots*. Retrieved from alphabottrading: <https://www.alphabottrading.com/trading-bots>
- Baz, J., Granger, N., Harvey, C. R., Roux, N. L., & Rattray, S. (2015). Dissecting investment strategies in the cross section and time series.
- Bellman, R. E. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 679-684. Retrieved from <http://www.jstor.org/stable/24900506>
- Cao, L. J., & Tay, F. E. (2003). Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Trans. Neural Netw*, 1506-1518.
- Chan, E. (2009). *Quantitative trading: How to build your own algorithmic trading business* (Vol. 430). John Wiley & Sons.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 2493-2537.
- Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2017). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems*, 653-664.
- Fischer, T. G. (2018). *Reinforcement learning in financial markets - A survey*. FAU Discussion Papers in Economics.
- Graham, B., Dodd, D. L., & Cottle, S. (1934). Security analysis. *McGraw-Hill*.
- Hasselt, H. v., & Guez, A. (2016). Deep Reinforcement Learning with Double Q-Learning; David Silver;. *Proceedings of the AAAI Conference on Artificial Intelligence*. doi:<https://doi.org/10.1609/aaai.v30i1.10295>
- Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 1735-1780. doi:<https://doi.org/10.1162%2Fneco.1997.9.8.1735>

- Hui, J. (2018, 9 12). *RL — Policy Gradient Explained*. Retrieved from Medium: <https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146>
- Khaidem, L., Saha, S., & Dey, S. R. (2016). *Predicting the direction of stock market prices using random forest*. arXiv:1605.00003. Retrieved from <https://arxiv.org/abs/1605.00003>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097-1105.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature* 521, 436–444.
- LeCun, Y., Boser, B., Denke, J. S., D. H., Howard, R. E., Hubbard, W., & Jackel, L. D. (n.d.). *Backpropagation Applied to Handwritten Zip Code Recognition*. AT&T Bell Laboratories.
- Deep Robust Reinforcement Learning for (2019) Zheng, Z & ,Li, Y., Zheng, W .108014–108022 ,*IEEE Access* .Practical Algorithmic Trading
doi:10.1109/ACCESS.2019.2932789
- Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*. Retrieved from <https://arxiv.org/abs/1509.02971>
- Lim, B., Zohren, S., & Roberts, S. (2019). Enhancing time-series momentum strategies using deep neural networks. *The Journal of Financial Data Science*.
- Practical Deep* (بدون تاریخ). Walid, A & ,Liu, X.-Y., Xiong, Z., Zhong, S., Yang, H .arXiv:1811.07522 *Reinforcement Learning Approach for Stock Trading*
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of The 33rd International Conference on Machine Learning* (pp. 1928-1937). PMLR.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 529-533.

- Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.
- Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.
- Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *International Conference on International Conference on Machine Learning (ICML)* (pp. 807-814). Omnipress.
- OpenAI. (n.d.). *Proximal Policy Optimization*. Retrieved from openai spinning up: <https://spinningup.openai.com/en/latest/algorithms/ppo.html>
- OpenAI. (n.d.). *Vanilla Policy Gradient*. Retrieved from OpenAI Spinning Up: <https://spinningup.openai.com/en/latest/algorithms/vpg.html>
- Sámek, D., & Varacha, P. (2013). Time series prediction using artificial neural networks: Single and multi-dimensional data. *Int. J. Math. Models Methods Appl. Sci.*, 38-46.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., . . . Silver, D. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588, 604-609.
- Schulman, J. (2016). *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. UC Berkeley.
- Schulman, J., Wolski, F., Dhariwa, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*.
- Sharpe, W. F. (1994). The Sharpe Ratio. *The Journal of Portfolio Management*, 21, 49-58. doi:<https://doi.org/10.3905%2Fjpm.1994.409501>
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. v., . . . Kavukcuoglu, K. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 484.
- Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. *Proceedings of the 31st International Conference on International Conference on Machine Learning* (pp. 387-395). ICML.
- Sirignano, J., & Cont, R. (2019). Universal features of price formation in financial markets: Perspectives from deep learning. *Quantitative Finance*, 1449-1459.

- Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Proc. Adv. Neural Inf. Process. Syst*, (pp. 1057-1063).
- Sutton, R., & Barto, A. (2018). *Reinforcement learning: An introduction*. Cambridge, MA: MIT press.
- An application of deep reinforcement learning to algorithmic trading .(2021) Ernst, D & ,Théate, T *Expert Systems with Applications*
- Tsantekidis, A., N. Passalis, A. T., Kannianen, J., Gabbouj, M., & Iosifidis, A. (2017). Forecasting stock prices from the limit order book using convolutional neural networks. *IEEE 19th Conf. Bus. Inform. (CBI)*, 7-12.
- Tsantekidis, A., Passalis, N., Tefas, A., Kannianen, J., Gabbouj, M., & Iosifidis, A. (2017). Forecasting stock prices from the limit order book using convolutional neural networks. *IEEE 19th Conference on Business Informatics (CBI)* (pp. 7-12). IEEE.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). *Attention Is All You Need*. arXiv:1706.03762. Retrieved from <https://arxiv.org/abs/1706.03762>
- Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, 279-292. doi:<https://doi.org/10.1007%2F323533a0>
- Wilder, J. W. (1978). *New concepts in technical trading systems*. Trend Research.
- Williams, J. D., & Young, S. (2007). Partially observable Markov decision processes for spoken dialog systems. *Comput. Speech Lang*, 393-422.
- Williams, R. J., Hinton, G. E., & Rumelhart, D. E. (1986). Learning representations by back-propagating errors. *Nature*, 533-536. doi:<https://doi.org/10.1038%2F323533a0>
- Deep Reinforcement Learning .(2020) Walid, A & ,Yang, H., Liu, X.-Y., Zhong, S Association for for Automated Stock Trading: An Ensemble Strategy doi:10.1145/3383455.3422540 .Computing Machinery
- Zhang, Z., Zohren, S., & Roberts, S. (2019). Deep Reinforcement Learning for Trading. *The Journal of Financial Data Science*, 25-40.

Zhang, Z., Zohren, S., & Roberts, S. (2019). Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 3001-3012.

پیوست ها

جدول پ-1: شرح کد منبع بدنه اصلی کد طراحی و آموزش عامل ها.

```
import os
import math
import talib
import numpy as np
import pandas as pd
import logging
import datetime
from scipy.stats import t
from typing import Callable, Dict, List, Optional, Tuple, Type, Union
from tqdm import tqdm
from pandas_datareader import data as web
import pandas_datareader as pdr
from dateutil.relativedelta import relativedelta

import pyfolio.timeseries as ts
import scipy.stats as st

import matplotlib.pyplot as plt
import seaborn as sns
import quantstats as qs

import gym
from gym import spaces
from gym.utils import seeding
from gym_anytrading.envs import TradingEnv, ForexEnv, StocksEnv, Actions, Positions
from stable_baselines3 import A2C, DDPG, DQN, PPO, TD3, SAC
from stable_baselines3.common.policies import ActorCriticPolicy
from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3.common.noise import NormalActionNoise, OrnsteinUhlenbeckActionNoise

import torch as th
from torch import nn
from sklearn.preprocessing import scale

#####
# Data Source:

class DataSource(object):
    def __init__(self, data_path, start_date, end_date,
                 time_frame, tickers, window_size,
                 ):
```



```

self.tickers = tickers

if end_date == None:
    end_date = datetime.datetime.now()
if start_date == None:
    start_date = end_date - relativedelta(years=2)

self.data = pd.DataFrame()
for ticker in self.tickers:
    csv_name = os.path.join(
        data_path,
        ticker+start_date.strftime('_from_%Y%m%d') +
        end_date.strftime('_to_%Y%m%d')+".csv"
    )
    ticker_data = self._load_data(
        csv_name=csv_name, time_frame=time_frame,
        start_date=start_date, end_date=end_date, ticker=ticker
    )
    self.data = pd.concat(
        [self.data, ticker_data],
    )
self.date_time = self.data.index
self.count = self.data.shape[0]
self.window_size = window_size
self.states = self.data.values

@staticmethod
def seed(seed):
    np.random.seed(seed)

def _load_data(self, csv_name, time_frame, start_date, end_date, ticker):
    log.info('loading data for {}'.format(ticker))

    if os.path.exists(csv_name):
        df = pd.read_csv(csv_name)
    else:
        with open("./tiingo_api_key.txt") as file:
            key = file.readline()

        df = pdr.get_data_tingo(ticker,
                                start=start_date,
                                end=end_date,
                                api_key=key
                                ).dropna()

        df.columns = [col.lower() for col in df.columns]
        df['ret_5'] = df.adjopen.pct_change(5)
        df['ret_10'] = df.adjopen.pct_change(10)
        df['ret_21'] = df.adjopen.pct_change(21)
        df['rsi'] = talib.STOCHRSI(df.adjopen)[1]
        df['macd'] = talib.MACD(df.adjopen)[1]
        df['atr'] = talib.ATR(df.adjhigh, df.adjlow, df.adjopen)
        df = df.replace((np.inf, -np.inf), np.nan).drop(
            ['volume', 'high', 'low', 'close', 'open', 'adjhigh', 'adjlow', 'adjclose', 'divcash',
            'splitfactor'], axis=1).dropna()

```

```

        df.columns = [col+'_'+ticker for col in df.columns]
        df.to_csv(csv_name, index_label="date_time")

    log.info('got data for {}'.format(ticker))
    return df

config = {
    "data": {
        "time_frame": "tiingo",
        "data_path": os.path.join(os.getcwd(), "data"),
        "tickers": ["GOOG"],
        "episode_duration": 480
    },
    "seed": 42,
    "model": {
        "window_size": 10,
        "initial_cash": 1_000_000,
        "commission_rate": 0,
        "start_date": datetime.datetime(2014, 1, 1),
        "end_date": datetime.datetime(2022, 8, 1),
        "stat_save_folder": None,
        "agent_save_folder": None
    }
}

data_config = config['data']
seed = config.get("seed", 42)

stat_save_path = os.path.join(os.getcwd(), "saved_stats")
if not os.path.exists(stat_save_path):
    os.mkdir(stat_save_path)
stat_save_folder = os.path.join(
    stat_save_path, datetime.datetime.now().strftime('%Y%m%d'))
if not os.path.exists(stat_save_folder):
    os.mkdir(stat_save_folder)

agent_save_path = os.path.join(os.getcwd(), "saved_agents")
if not os.path.exists(agent_save_path):
    os.mkdir(agent_save_path)
agent_save_folder = os.path.join(
    agent_save_path, datetime.datetime.now().strftime('%Y%m%d'))
if not os.path.exists(agent_save_folder):
    os.mkdir(agent_save_folder)

np.random.seed(seed)
window_size = config["model"]["window_size"]
data_path = config["data"]["data_path"]
tickers = config["data"]["tickers"]
time_frame = config["data"]["time_frame"]
start_date, end_date = config["model"]["start_date"], config["model"]["end_date"]

data_source_goog = DataSource(
    data_path=data_path,

```

```

        start_date=start_date,
        end_date=end_date,
        time_frame=time_frame,
        tickers=tickers,
        window_size=window_size,
    )
goog = data_source_goog.data

def preprocess(df, ticker):
    df.set_index("date_time", inplace=True)
    df_train = df.iloc[:int(0.8*len(df)), :].copy()
    df_test = df.iloc[int(0.8*len(df)):, :].copy()

    sig_train = df_train.rolling(30).std()
    sig_test = df_test.rolling(30).std()

    mu_train = df_train.rolling(30).mean()
    mu_test = df_test.rolling(30).mean()

    eps = np.finfo(np.float32).eps

    df_train_norm = ((df_train - mu_train.shift())/(sig_train + eps)).dropna()
    df_test_norm = ((df_test - mu_test.shift())/(sig_test + eps)).dropna()

    df_train_norm.columns = [col+"_norm" for col in df_train_norm.columns]
    df_test_norm.columns = [col+"_norm" for col in df_test_norm.columns]

    df_train_norm["adjclose_" +
                  ticker] = df_train["adjclose_" + ticker][30:].values
    df_train_norm["adjopen_" + ticker] = df_train["adjopen_" + ticker][30:].values
    df_test_norm["adjclose_" + ticker] = df_test["adjclose_" + ticker][30:].values
    df_test_norm["adjopen_" + ticker] = df_test["adjopen_" + ticker][30:].values

    return(df_train, df_test, df_train_norm, df_test_norm)

df_train, df_test, df_train_norm, df_test_norm = preprocess(
    goog.copy(), "GOOG")

#####
# Env:

class MyTradingEnv(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self, df, window_size, frame_bound, init_cash=100_000, symbol="", IS_DQN=False):
        assert df.ndim == 2
        self.seed()
        self.df = df

        self.symbol = symbol
        self.IS_DQN = IS_DQN

```

```

self.frame_bound = frame_bound
self.window_size = window_size
self.dates = df.index
self.open_prices, self.close_prices, self.signal_features = self._process_data()
self.shape = (window_size * self.signal_features.shape[1],)

# spaces
self.action_space = spaces.Discrete(n=3)
self.observation_space = spaces.Box(
    Low=-np.inf, high=np.inf, shape=self.shape, dtype=np.float64)

# episode
self._start_tick = self.window_size
self._end_tick = len(self.open_prices) - 1
self.starting_price = self.open_prices[self._start_tick]
self._done = None
self._current_tick = None
self._last_trade_tick = None
self.init_cash = init_cash
self._cash = init_cash
self._position_value = 0
self._total_assets = self._cash + self._position_value
self._init_total_assets = self._total_assets
self._last_trade = 0
self._position_history = None
self._actions_history = None
self._total_reward = None
self._total_profit = None
self._first_rendering = None
self.history = None

def seed(self, seed=None):
    self.np_random, seed = seeding.np_random(seed)
    return [seed]

def reset(self, rand_trail=False):
    self._done = False
    if rand_trail:
        self._start_tick, _ = get_start_end_index(
            self.frame_bound[0], self.frame_bound[1])
    else:
        self._start_tick = self.frame_bound[0]
    self._end_tick = self.frame_bound[1] - 1
    self._current_tick = self._start_tick
    self.starting_price = self.open_prices[self._start_tick]
    self.curr_open_price = self.open_prices[self._start_tick]
    self._last_trade_tick = self._current_tick - 1
    self._cash = self.init_cash
    self._position_value = 0
    self._total_assets = self._cash + self._position_value
    self._init_total_assets = self._total_assets
    self._last_trade = 0
    self._position_history = (
        self.window_size * [0]) + [self._position_value]

```

```

self._trade_history = (self.window_size * [0]) + [self._last_trade]
self._actions_history = ((1+self.window_size) * [0])
self._total_reward = 0.
self._total_profit = 0. # unit
self._total_pct_profit = 0
self._first_rendering = True
self.history = {}
return self._get_observation()

def step(self, action):

    self._done = False
    self._current_tick += 1
    self.curr_open_price = self.open_prices[self._current_tick]

    action = action-1
    if self._current_tick == self._end_tick:
        self._done = True
    if -0.01 < action < 0.01:
        action = np.zeros(1)
    action = action.item()
    last_day_total_assets = self._total_assets
    step_reward = self._calculate_reward(action)
    self._actions_history.append(action)
    self._total_reward += step_reward
    self._total_profit = self._total_assets - self.init_cash
    self._total_pct_profit = self._total_profit/self.init_cash

    # self._position_history.append(self._position)
    observation = self._get_observation()
    info = dict(
        date=self.dates[self._current_tick],
        total_reward=rnd(self._total_reward),
        total_profit=rnd(self._total_profit),
        total_profit_percentage=rnd(self._total_pct_profit),
        buy_and_hold=rnd((self.curr_open_price/self.starting_price)-1),
        daily_price_return=rnd(
            (self.curr_open_price/self.open_prices[self._current_tick-1])-1),
        daily_return=rnd((self._total_assets/last_day_total_assets)-1),
        total_assets=rnd(self._total_assets)
    )
    self._update_history(info)
    return observation, step_reward, self._done, info

def _get_observation(self):
    sig = self.signal_features[(
        self._current_tick-self.window_size):self._current_tick]
    return(sig.reshape(-1))

def _update_history(self, info):
    if not self.history:
        self.history = {key: [] for key in info.keys()}

    for key, value in info.items():

```

```

        self.history[key].append(value)

def render(self, mode='human'):

    def _plot_position(position, tick):
        color = None
        if position > 0:
            color = 'red'
        elif position < 0:
            color = 'green'
        if color:
            plt.scatter(tick, self.prices[tick], color=color)

    if self._first_rendering:
        self._first_rendering = False
        plt.cla()
        plt.plot(self.open_prices)
        start_position = self._position_history[self._start_tick]
        _plot_position(start_position, self._start_tick)

    _plot_position(self._position, self._current_tick)

    plt.suptitle(
        "Total Reward: %.6f" % self._total_reward + ' ~ ' +
        "Total Profit: %.6f" % self._total_profit
    )
    plt.pause(0.01)

def render_all(self, mode='human'):
    window_ticks = np.arange(self._start_tick, self._end_tick)
    plt.plot(
        pd.to_datetime(self.dates),
        self.open_prices,
        label=f"Open Price - {self.symbol}"
    )

    short_ticks = []
    short_ticks_dates = []
    long_ticks = []
    long_ticks_dates = []
    for i, tick in enumerate(window_ticks):
        if self._trade_history[i] < 0:
            short_ticks.append(tick)
            short_ticks_dates.append(
                self.dates[tick]
            )
        elif self._trade_history[i] > 0:
            long_ticks.append(tick)
            long_ticks_dates.append(
                self.dates[tick]
            )

    plt.plot(
        pd.to_datetime(short_ticks_dates),

```

```

        self.open_prices[short_ticks],
        'rv',
        markersize=6,
        Label="Sell"
    )
    plt.plot(
        pd.to_datetime(long_ticks_dates),
        self.open_prices[long_ticks],
        'g^',
        markersize=6,
        Label="Buy"
    )

    plt.suptitle(
        "Total Reward: %.6f" % self._total_reward + ' ~ ' +
        "Total Profit: %.6f" % self._total_profit
    )

def close(self):
    plt.close()

def save_rendering(self, filepath):
    plt.savefig(filepath)

def pause_rendering(self):
    plt.show()

def _process_data(self):
    close_prices = self.df.loc[:, 'adjclose'+self.symbol].to_numpy()
    open_prices = self.df.loc[:, 'adjopen'+self.symbol].to_numpy()
    # validate index (TODO: Improve validation)
    open_prices[self.frame_bound[0] - self.window_size]

    signal_features = self.df.values

    return(open_prices, close_prices, signal_features)

def _calculate_reward(self, action):
    step_reward = 0
    if -0.01 < action < 0.01:
        trade = False
    else:
        trade = True

    current_price = self.open_prices[self._current_tick]
    last_day_price = self.open_prices[self._current_tick-1]
    pct_change = (current_price)/last_day_price

    before_position_value = self._position_value
    before_total_assets = self._total_assets
    self._position_value *= pct_change
    self._total_assets = self._cash + self._position_value

    if trade:

```

```

total_assets = self._total_assets
new_position_value = action * total_assets
position_value_diff = new_position_value - self._position_value

num_assets_to_trade = position_value_diff // current_price

actual_position_value_diff = num_assets_to_trade*current_price
actual_new_position_value = actual_position_value_diff + self._position_value

self._position_value = actual_new_position_value
self._cash = total_assets - actual_new_position_value
self._total_assets = self._cash + self._position_value

self._last_trade = actual_position_value_diff
self._trade_history.append(num_assets_to_trade)
self._position_history.append(actual_new_position_value)
self._last_trade_tick = self._current_tick
else:
    self._trade_history.append(0)
    self._position_history.append(self._position_value)

step_reward = ((self._total_assets - before_total_assets) /
               before_total_assets)

return step_reward

def _update_profit(self, action):
    raise NotImplementedError

def max_possible_profit(self): # trade fees are ignored
    raise NotImplementedError
#####
# utils:

def test_model(model, test_env, rounds=500):
    res = dict(
        date=[],
        total_reward=[],
        total_profit=[],
        total_profit_percentage=[],
        buy_and_hold=[],
        daily_price_return=[],
        daily_return=[],
        total_assets=[],
        perf_stat=pd.DataFrame(),
        bnh_perf_stat=pd.DataFrame()
    )

    for itr in tqdm(range(rounds)):
        observation = test_env.reset(False)

        while True:
            action, _states = model.predict(observation)

```



```

observation, reward, done, info = test_env.step(action)

# env.render()
if done:
    # print("info:", info)
    break

for key in test_env.history.keys():
    res[key].append(test_env.history[key])

res["perf_stat"] = pd.concat(
    [
        res["perf_stat"],
        ts.perf_stats(np.array(test_env.history['daily_return']))
    ],
    axis=1)

for key, val in res.items():
    if key not in ['date', 'perf_stat']:
        res[key] = np.array(val)

mean_res = {}
for k, v in res.items():
    if k == 'date':
        mean_res[k] = v[0]
    elif k == 'perf_stat':
        m = v.mean(1)
        s = v.std(1)
        dof = rounds-1
        se = st.sem(v.values, axis=1)

        lower95, upper95 = st.t.interval(
            alpha=0.95, df=dof, loc=m, scale=se)
        lower99, upper99 = st.t.interval(
            alpha=0.99, df=dof, loc=m, scale=se)

        lower95 = pd.Series(lower95)
        upper95 = pd.Series(upper95)
        lower99 = pd.Series(lower99)
        upper99 = pd.Series(upper99)

        lower95.index = v.index
        upper95.index = v.index
        lower99.index = v.index
        upper99.index = v.index

        lower_3s = m - 1.5*s
        upper_3s = m + 1.5*s

        mean_res[k] = pd.concat([m,
                                lower95, upper95,
                                lower99, upper99,
                                lower_3s, upper_3s
                                ], axis=1)

```

```

    )

    mean_res[k].columns = ["point_est (mean)",
                           "%95 conf. lower bound",
                           "%95 conf. upper bound",
                           "%99 conf. lower bound",
                           "%99 conf. upper bound",
                           "3sigma lower",
                           "3sigma upper"
                           ]

    elif k == "bnh_perf_stat":
        mean_res[k] = ts.perf_stats(
            test_env.df["adjopen_"+test_env.symbol][test_env.frame_bound[0]
                    :test_env.frame_bound[1]].pct_change().fillna(0).values
        )

    elif k in ["buy_and_hold", "daily_price_return"]:
        mean_res[k] = v.mean(0)

    else:
        mean_res[k+"_point"] = v.mean(0)
        m = v.mean(0)
        s = v.std(0)
        dof = rounds-1

        se = st.sem(v, axis=0)
        lower95, upper95 = st.t.interval(
            alpha=0.95, df=dof, loc=m, scale=se)
        lower99, upper99 = st.t.interval(
            alpha=0.99, df=dof, loc=m, scale=se)

        mean_res[k+"_lower95"] = lower95
        mean_res[k+"_upper95"] = upper95

        mean_res[k+"_lower99"] = lower99
        mean_res[k+"_upper99"] = upper99

        mean_res[k+"_min"] = v.min(0)
        mean_res[k+"_max"] = v.max(0)

        mean_res[k+"_lower3S"] = m - 1.5*s
        mean_res[k+"_upper3S"] = m + 1.5*s

    return(mean_res)

def plot_test_result(result_dict, model_name, symbol, split="Test"):
    plt.style.use("seaborn")
    plt.figure(figsize=(16, 10))
    plt.title(f"{model_name} {split} Performance - {symbol}")
    plt.xlabel("Date")
    plt.ylabel("Cumulative Return")
    plt.title(f"{model_name} {split} Performance - {symbol}")
    plt.plot(pd.to_datetime(
        result_dict['date']), result_dict['buy_and_hold'], Label="Buy and Hold")
    plt.plot(pd.to_datetime(

```

```

        result_dict['date']], result_dict['total_profit_percentage_point'], label=model_name,
        color="r")
    plt.fill_between(
        pd.to_datetime(result_dict['date']),
        result_dict['total_profit_percentage_upper95'],
        result_dict['total_profit_percentage_lower95'],
        label="95% Conf. Int.",
        alpha=0.8
    )

    plt.fill_between(
        pd.to_datetime(result_dict['date']),
        result_dict['total_profit_percentage_upper3S'],
        result_dict['total_profit_percentage_lower3S'],
        label="3 Sigma",
        alpha=0.4
    )

    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
#####
# Model and Training:

TRAIN = True

env = MyTradingEnv(
    df=df_train,
    window_size=window_size,
    frame_bound=(window_size, len(df_train)),
    symbol="GOOG"
)

test_env = MyTradingEnv(
    df=df_test,
    window_size=window_size,
    frame_bound=(window_size, len(df_test)),
    symbol="GOOG"
)

dqn_env = MyTradingEnv(
    df=df_train,
    window_size=window_size,
    frame_bound=(window_size,
                  len(df_train)),
    symbol="GOOG",
    IS_DQN=True
)

dqn_test_env = MyTradingEnv(
    df=df_test,
    window_size=window_size,
    frame_bound=(window_size, len(df_test)),
    symbol="GOOG",
    IS_DQN=True
)

```

```

)

class CustomNetwork(nn.Module):
    """
    Custom network for policy and value function.
    It receives as input the features extracted by the feature extractor.

    :param feature_dim: dimension of the features extracted with the features_extractor (e.g.
    features from a CNN)
    :param last_layer_dim_pi: (int) number of units for the last layer of the policy network
    :param last_layer_dim_vf: (int) number of units for the last layer of the value network
    """

    def __init__(
        self,
        feature_dim: int,
        last_layer_dim_pi: int = 64,
        last_layer_dim_vf: int = 64,
    ):
        super(CustomNetwork, self).__init__()

        # IMPORTANT:
        # Save output dimensions, used to create the distributions
        self.latent_dim_pi = last_layer_dim_pi
        self.latent_dim_vf = last_layer_dim_vf

        # Policy network
        self.policy_net = nn.Sequential(
            nn.Linear(feature_dim, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.07),
            nn.Linear(256, last_layer_dim_pi),
            nn.BatchNorm1d(last_layer_dim_pi),
            nn.ReLU(),
            nn.Dropout(0.07)
        )

        # Value network
        self.value_net = nn.Sequential(
            nn.Linear(feature_dim, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.07),
            nn.Linear(256, last_layer_dim_vf),
            nn.BatchNorm1d(last_layer_dim_vf),
            nn.ReLU(),
            nn.Dropout(0.07)
        )

    def forward(self, features: th.Tensor) -> Tuple[th.Tensor, th.Tensor]:
        """
        :return: (th.Tensor, th.Tensor) latent_policy, latent_value of the specified network.
        If all layers are shared, then ``latent_policy == latent_value``
        """

```

```

    """
    return self.policy_net(features), self.value_net(features)

def forward_actor(self, features: th.Tensor) -> th.Tensor:
    return self.policy_net(features)

def forward_critic(self, features: th.Tensor) -> th.Tensor:
    return self.value_net(features)

class CustomActorCriticPolicy(ActorCriticPolicy):
    def __init__(
        self,
        observation_space: gym.spaces.Space,
        action_space: gym.spaces.Space,
        lr_schedule: Callable[[float], float],
        net_arch: Optional[List[Union[int, Dict[str, List[int]]]]] = None,
        activation_fn: Type[nn.Module] = nn.Tanh,
        *args,
        **kwargs,
    ):
        super(CustomActorCriticPolicy, self).__init__(
            observation_space,
            action_space,
            lr_schedule,
            net_arch,
            activation_fn,
            # Pass remaining arguments to base class
            *args,
            **kwargs,
        )
        # Disable orthogonal initialization
        self.ortho_init = False

    def _build_mlp_extractor(self) -> None:
        self.mlp_extractor = CustomNetwork(self.features_dim)

ppo_model = PPO(
    CustomActorCriticPolicy,
    env,
    Learning_rate=0.0001,
    n_steps=1024,
    batch_size=128,
    n_epochs=15,
    gamma=0.99,
    gae_lambda=0.95,

```

```

clip_range=0.2,
clip_range_vf=None,
normalize_advantage=True,
verbose=1
)
dqn_model = DQN("MlpPolicy", dqn_env, verbose=1,
                learning_rate=0.0001,
                buffer_size=1000000,
                learning_starts=50000,
                batch_size=512,
                tau=0.99,
                gamma=0.99,
                train_freq=(1000, "step"),
                gradient_steps=1,
                target_update_interval=5000,
                exploration_fraction=0.2,
                exploration_initial_eps=1.0,
                exploration_final_eps=0.02
                )

if TRAIN:
    ppo_model.learn(total_timesteps=200_000)
else:
    ppo_model = PPO.load("/20220908_ppo_G00G_v4.zip")
    ppo_model.set_env(env)

dqn_eval_callback = EvalCallback(dqn_test_env, best_model_save_path='/dqn_goog_logs',
                                log_path='/dqn_goog_logs', eval_freq=2000,
                                deterministic=True, render=False)

if TRAIN:
    best_dqn.learn(total_timesteps=200000, callback=dqn_eval_callback)
else:
    best_dqn = DQN.load("/20220908_DQN_G00G_v2.zip")
    best_dqn.set_env(dqn_env)

ppo_model.set_random_seed(seed)
ppo_res = test_model(ppo_model, test_env, 500)
best_dqn.set_random_seed(seed)
best_dqn_res = test_model(best_dqn, dqn_test_env, 500)

ppo_res['perf_stat'].to_csv("PP0v4_200kitr_G00G_500r_stats.csv")
dqn_res['perf_stat'].to_csv("DQNV2_200kitr_G00G_500r_stats.csv")

plot_test_result(result_dict=ppo_res, model_name="PPO",
                 symbol="G00G", split="Test")
plot_test_result(result_dict=best_dqn_res, model_name="DQN",
                 symbol="G00G", split="Test")

```

Abstract

Today, with the increase in speed and power of computing and processing resources, algorithmic trading in financial markets has become a main element in quantitative finance. Successful strategies for automated stock trading are essential for hedge funds and investment firms. These strategies include optimizing capital allocation and maximizing investment returns. In algorithmic trading, feature extraction and trading strategy design are two prominent challenges for long-term profit.

In quantitative finance, stock trading is essentially a dynamic decision problem, that is, deciding where, at what price, and how much to trade in a highly stochastic, dynamic, and complex stock market. With recent advances in deep reinforcement learning (DRL) methods, sequential dynamic decision problems can be modeled and solved with a human-like approach.

In this project, we examine the potential and performance of deep reinforcement learning to optimize stock trading strategies and thus maximize investment returns. Google stock is selected as our trading stock and the daily opening and closing price along with trading volume and several technical indicators are used as a training environment and trading market.

We present two trading agents based on deep reinforcement learning to autonomously make trading decisions and generate returns in dynamic financial markets. The performance of these intelligent agents is compared with the performance of the buy and hold strategy. And at the end, it is shown that the proposed deep reinforcement learning approach performs better than the buy and hold benchmark in terms of risk assessment criteria and portfolio return.

Key Words: Machine Learning, Reinforcement Learning, Algorithmic Trading, Artificial Neural Networks



Amirkabir University of Technology
(Tehran Polytechnic)

Department of Industrial Engineering and Management Systems

MSc Thesis

**Algorithmic Trading in Financial Markets Using
Deep Reinforcement Learning Methods**

By
Ebrahim Pichka

Supervisor
Dr. Masoud Mahootchi

Advisor
Dr. Akbar Esfahanipour

September 2022