دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی

استفاده از یادگیری تقویتی برای گل زدن در موقعیت تک به تک در فوتبال

نگارش
آراد فیروزکوهی

استاد راهنما
دکتر احسان ناظرفرد

فروردین ۱۴۰۳


بسم الله الرحمن الرحيم

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی

استفاده از یادگیری تقویتی برای گل زدن در موقعیت تک به تک در فوتبال

نگارش
آراد فیروزکوهی

استاد راهنما
دکتر احسان ناظرفرد

فروردین ۱۴۰۲

صفحه فرم ارزیابی و تصویب پایان نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تایید و تصویب پایان نامه موسوم به فرم کمیته دفاع- موجود در پرونده آموزشی- را قرار دهید.

نکات مهم:

- نگارش پایان نامه/رساله باید به زبان فارسی و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان نامه های دانشگاه صنعتی امیرکبیر باشد.(دستورالعمل و راهنمای حاضر)
- رنگ جلد پایان نامه/رساله چاپی کارشناسی، کارشناسی ارشد و دکترا باید به ترتیب مشکی، طوسی و سفید رنگ باشد.
- چاپ و صحافی پایان نامه/رساله بصورت پشت و رو(دورو) بلامانع است و انجام آن توصیه می شود.

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

به نام خدا

تعهدنامه اصالت اثر

تاریخ: فروردین ۱۴۰۲

اینجانب **آراد فیروزکوهی** متعهد می‌شوم که مطالب مندرج در این پایان‌نامه 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 پایان‌نامه قبلا 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 پایان‌نامه 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.

آراد فیروزکوهی

امضا

تقدیم به پدر بزرگوار و مادر عزیز و مهربانم

که در سختی‌ها و دشواری‌های زندگی، همواره یاوری دلسوز و فداکار و پشتیبانی

محکم و مطمئن برایم بوده‌اند.

سپاس‌گزاری

از استاد دلسوز و محترم؛ جناب دکتر ناظرفرد که با صبر و حوصله، از هیچ کمکی در مسیر انجام این پروژه و نوشتن این پایان‌نامه از من دریغ ننمودند؛ کمال تشکر و قدرانی را دارم.

آراد فیروزکوهی
فروردین ۱۴۰۲

چکیده

این پایان‌نامه به اکشتاف عمیق یادگیری تقویتی و نحوه عملکرد آن در گل زدن در تک به تک فوتبال می‌پردازد. ابتدا، ما به بررسی مفاهیم یادگیری تقویتی و اصول اساسی آن می‌پردازیم تا درکی جامع از چگونگی تصمیم‌گیری و یادگیری ماشین در محیط‌های پویا ارائه شود. سپس پلتفرم شبیه‌ساز دو بعدی فوتبال ربوکاپ که از آن قرار است استفاده کنیم، معرفی می‌شود. در این پژوهش، الگوریتم‌های مختلف یادگیری تقویتی از جمله شبکه Q عمیق، یادگیری تقویتی مبتنی بر سیاست، و دیگر رویکردهای پیشرفته بررسی شده‌اند تا تأثیر آن‌ها بر بهبود عملکرد بازیکنان در زدن گل‌ها ارزیابی گردد. بخش مهمی از این تحقیق به ایجاد یک روش استاندارد برای انجام یادگیری تقویتی در شبیه‌سازی فوتبال ربوکاپ با استفاده از چارچوب جیم اختصاص یافته است، که نوید بخش یکپارچه‌سازی و استانداردسازی روش‌های یادگیری تقویتی در این حوزه است. در نهایت خواهیم دید که چگونه استفاده از این الگوریتم‌ها می‌تواند به طور موثری به بازیکنان کمک کنند تا در موقعیت‌های تک به تک بهترین تصمیم‌ها را بگیرند و گل‌های بیشتری بزنند. علاوه بر این، پایان‌نامه به بررسی تأثیر پارامترها و تنظیمات مختلف الگوریتمی بر عملکرد یادگیری تقویتی می‌پردازد و راهکارهایی برای بهبود عملکرد ارائه می‌دهد.

واژه‌های کلیدی:

یادگیری تقویتی، یادگیری تقویتی عمیق، شبیه‌ساز دوبعدی فوتبال، جیم، تک به تک با دروازه‌بان

فهرست مطالب

عنوان صفحه

فهرست تصاویر

شکل صفحه

فهرست جداول

جدول صفحه

فصل اول

مقدمه

۱-۱ مقدمه

۱-۲ تعریف مساله

۱-۳ اهداف

۱-۴ کارهای مشابه

۱-۵ ابزارها و نرم‌افزارهای مورد استفاده

۱-۶ ساختار پایان‌نامه

فصل دوم

یادگیری تقویتی

۲-۱ مقدمه

به طور عمومی، یادگیری ماشین به دسته‌ای از الگوریتم‌ها و روش‌های محاسباتی گفته می‌شود که به ماشین‌ها امکان یادگیری از داده‌ها و تجربه‌های خود را می‌دهند. یادگیری ماشین به دو دسته‌ی اصلی تقسیم می‌شود: یادگیری نظارت‌شده و یادگیری بدون نظارت. در یادگیری نظارت‌شده، مدل به کمک داده‌های برچسب‌خورده آموزش داده می‌شود، و سپس برای پیش‌بینی خروجی‌های جدید از این مدل استفاده می‌شود. در یادگیری بدون نظارت، مدل بدون داده‌های برچسب‌خورده آموزش داده می‌شود، و باید خودش مفاهیم و الگوهای موجود در داده‌ها را کشف کند. یادگیری تقویتی در این دو دسته قرار نمی‌گیرد، و به عنوان یک دسته جداگانه در نظر گرفته می‌شود.

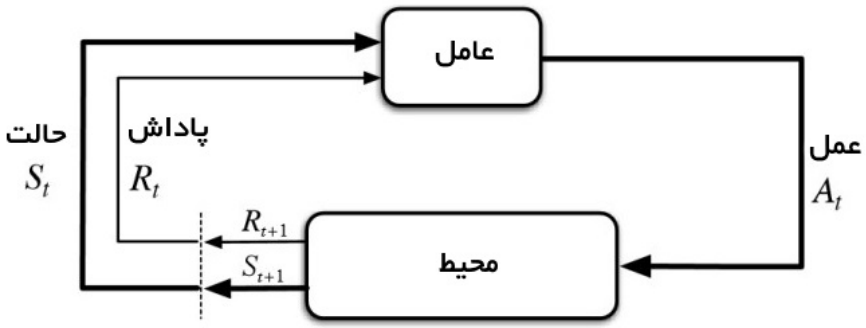
یادگیری تقویتی یک روش یادگیری ماشین است که به عامل اجازه می‌دهد تا رفتار بهینه را از طریق تعامل و آزمون و خطا با محیط یاد بگیرد. در این روش، عامل مشاهدات خود را از محیط (به کمک سنسور) دریافت کرده، و بر اساس آن تصمیم خود را اخذ می‌کند. پس از انجام هر عمل، محیط به عامل پاداشی می‌دهد که نشان‌دهنده‌ی عملکرد عامل در آن حالت است. هدف این است که عامل مجموع پاداش‌های دریافتی خود را بیشینه کند، که در صورت صحیح بودن تعریف سیگنال پاداش، معمولا به معنای رسیدن به هدف مطلوب است.

یادگیری ماشین با سایر رویکردهای یادگیری ماشین مانند یادگیری نظارت‌شده و یادگیری بدون نظارت تفاوت‌های زیادی دارد، و معمولا بر مسائلی به کار می‌رود که تمرکز بر تصمیم‌گیری یا پیدا کردن سیاست بهینه بدون نیاز به داده‌های برچسب‌خورده است. در عوض، عامل با اکتشاف و بهره‌برداری ابتدا انتقالات بین حالت‌ها و پاداش‌های مرتبط با آن‌ها را یاد می‌گیرد، و سپس سیاستی را یاد می‌گیرد که مجموع پاداش‌ها را بیشینه کند.

۲-۲ اصول یادگیری تقویتی

۲-۲-۱ عامل، محیط، حالت، عمل و پاداش

در یادگیری تقویتی، عامل [۱] تصمیم‌گیرنده برای رسیدن به هدف خود با محیط [۲] تعامل دارد. محیط معمولا به صورت مجموعه‌ای از حالت‌ها [۳] و عمل‌هایی [۴] که عامل می‌تواند انجام دهد مدل می‌شود. در هر گام، عامل مشاهده‌ای از حالت محیط را دریافت می‌کند و بر اساس آن تصمیمی اتخاذ می‌کند. پس از انجام عمل، محیط به عامل پاداش [۵] می‌دهد که نشان‌دهنده‌ی عملکرد عامل در آن حالت است.



شکل ۲-۱: تعامل عامل و محیط

در اکثر مواقع، یادگیری تقویتی در مسائلی استفاده می‌شود، که بتوان آن را به دنباله‌ای از گام‌ها تقسیم کرد که قطعا به یک حالت پایانی می‌رسد. هر یک از این دنباله‌ها را قسمت [۶] می‌نامند.

۲-۲-۲ فرایند تصمیم‌گیری مارکوف

فرایند تصمیم‌گیری مارکوف [۷] مدلی است برای تصمیم‌گیری در محیط‌هایی که به صورت مارکوف هستند. محیط‌های دارای خاصیت مارکوف، محیط‌هایی هستند که حالت بعدی به صورت کامل به حالت فعلی و عمل انجام شده وابسته است.

یک فرایند تصمیم‌گیری مارکوف، چهارتایی (S, A, P, R) است که در آن:

[1] Agent
[2] Environment
[3] State
[4] Action
[5] Reward
[6] Episode
[7] Markov Decision Process (MDP)

- S مجموعه‌ی تمام حالت‌های ممکن محیط است.
- A مجموعه‌ی تمام عمل‌های ممکن است.
- P تابع انتقال [8] است که به ازای هر حالت و عمل، توزیع احتمال حالت بعدی را مشخص می‌کند.
- R تابع پاداش [9] است که به ازای هر حالت و عمل، پاداش مورد انتظار را مشخص می‌کند.
- فاکتور تخفیف γ نیز معمولا به عنوان یک پارامتر دیگر در نظر گرفته می‌شود که نشان‌دهنده‌ی اهمیت پاداش‌های آینده نسبت به پاداش‌های فعلی است.

همانطور که گفته شد، در هر گام عامل با اخذ تصمیم خود، محیط را به حالت جدیدی می‌برد و پاداشی دریافت می‌کند. به مجموع کل پاداش‌هایی که عامل در یک قسمت دریافت می‌کند، خروجی [10] گفته می‌شود.

$$G_t = R_{t+1} + \gamma \times R_{t+2} + \gamma^2 \times R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k \times R_{t+k+1} \tag{۱-۲}$$

۲-۲-۳ سیاست، تابع ارزش، و تابع ارزش عمل

سیاست [11] یک تابع از حالت‌ها به عمل‌ها است که نشان‌دهنده‌ی رفتار عامل در هر حالت محیط است. در واقع مسئله یادگیری تقویتی را می‌توان به ((یافتن سیاستی که مجموع پاداش‌ها را بیشینه می‌کند)) تعبیر کرد.

$$\pi(a|s) = \mathbb{P}\{A_t = a | S_t = s\} \tag{۲-۲}$$

تابع ارزش [12] $V(s)$ یک تابع از حالت‌ها است که نشان‌دهنده‌ی میزان پاداش مورد انتظار از یک

[8] Transition Function
[9] Reward Function
[10] Return
[11] Policy
[12] Value Function

حالت تا به انتهای قسمت است.

$$v_\pi(s) = \mathbb{E}_\pi\{G_t|S_t = s\} = \sum_{a\in A}\pi(a|s) \times \{R_s^a + \gamma \times \sum_{s'\in S} p(s'|s,a) \times v_\pi(s')\} \qquad (۲\text{-}۳)$$

تابع ارزش عمل [۱۳] $Q(s,a)$ نیز مشابه تابع ارزش است، با این تفاوت که به جای حالت، از یک حالت و یک عمل مشخص محاسبه می‌شود. رایج است که به این تابع، تابع کیو گفته شود.

$$q_\pi(s,a) = \mathbb{E}_\pi\{G_t|S_t = s, A_t = a\} = R_s^a + \gamma \times \sum_{s'\in S} p(s'|s,a) \times v_\pi(s') \qquad (۲\text{-}۴)$$

به این معادلات، که از کلیدی‌ترین روابط در یادگیری تقویتی هستند، معادله بلمن [۱۴] گفته می‌شود.

۲-۳ الگوریتم‌های پایه یادگیری تقویتی

۲-۳-۱ برنامه‌نویسی پویا

همان‌طور که در معادله ۲-۳ مشخص است، می‌توان تابع ارزش را به صورت بازگشتی محاسبه کرد. این روش برنامه‌نویسی پویا [۱۵] نام دارد.

یادگیری به کمک تکرار ارزش

در روش تکرار ارزش [۱۶]، ، ابتدا تابع ارزش را به صورت تصادفی مقداردهی می‌کنیم و سپس آن را به صورت بازگشتی به روزرسانی می‌کنیم. قابل اثبات است که این روش به تابع ارزش بهینه همگرا می‌شود. به صورت شهودی نیز، می‌توان دید که ارزش از سمت حالت‌های پایانی به سمت حالت‌های ابتدایی به روزرسانی می‌شود.

[13] Action Value Function
[14] Bellman Equation
[15] Dynamic Programming
[16] Value Iteration

یادگیری به کمک تکرار سیاست

در روش تکرار سیاست [17]، ابتدا سیاست را به صورت تصادفی مقداردهی می‌کنیم و سپس تابع ارزش را برای آن محاسبه می‌کنیم. سپس سیاست را به صورت بازگشتی به روزرسانی می‌کنیم. این فرآیند را تا زمانی که سیاست تغییر نکند ادامه می‌دهیم. قابل اثبات است که این روش به سیاست بهینه همگرا می‌شود.

در عمل، با توجه به اینکه معمولا به تابع انتقال دسترسی نداریم، نمی‌توانیم از روش‌های برنامه‌نویسی پویا به صورت مستقیم استفاده کنیم. در واقع نیاز به راه حل‌های مستقل از مدل داریم که به کمک نمونه‌برداری و کاوش محیط، سیاست بهینه را یاد بگیرند.

۲-۳-۲ یادگیری به کمک نمونه‌برداری مونته کارلو

در روش‌های یادگیری به کمک نمونه‌برداری مونته کارلو [18]، به جای استفاده از مدل، از نمونه‌برداری برای تخمین ارزش استفاده می‌شود. کافی‌ست ابتدا یک قسمت را به طور کامل اجرا کنیم، و سپس ارزش هر حالت را، به سمت خروجی قسمت، به روزرسانی کنیم:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times (G_t - Q(s_t, a_t)) \quad (۵-۲)$$

که در این فرمول، α نرخ یادگیری [19] است.

لازم به ذکر است که در حین اجرای قسمت، از سیاست اپسیلون-حریصانه [20] استفاده می‌شود. در این سیاست، با احتمال ϵ عملی تصادفی انجام می‌شود و با احتمال $1 - \epsilon$ عملی که ارزش بیشتری دارد انجام می‌شود. دلیل استفاده از این سیاست، نیاز به کاوش محیط و جلوگیری از گیر کردن در حالت‌های محلی است.

از معایب این روش، می‌توان به نیاز به اجرای کامل قسمت‌ها و نیاز به زمان برای یادگیری اشاره کرد. به همین دلیل، این روش برای مسائلی که قسمت‌های طولانی دارند، مناسب نیست. از مشکلات دیگر این روش، عدم استفاده از ویژگی مارکوف محیط است.

[17] Policy Iteration
[18] Monte Carlo Sampling
[19] Learning Rate
[20] ϵ-greedy

۲-۳-۳ یادگیری به کمک تفاوت زمانی

همان‌طور که گفته شد، استفاده از یادگیری مونته‌کارلو باعث می‌شود که عامل در حین انجام قسمت، از تجربه‌ی قبلی خود استفاده نکند و به روز رسانی ارزش‌ها فقط پس از اتمام قسمت انجام شود. به این روش، یادگیری آفلاین [۲۱] گفته می‌شود. در روش یادگیری به کمک تفاوت زمانی [۲۲]، عامل در حین انجام قسمت، از تجربه‌ی خود استفاده می‌کند و ارزش‌ها را به صورت آنلاین به روزرسانی می‌کند. در واقع به کمک معادله بلمن، مقادیر کیو به سمت مقدار کیو بعدی به روزرسانی می‌شوند.

پیاده‌سازی این دو الگوریتم، به دو روش دید رو به جلو و دید رو به عقب انجام می‌شود. در حالت دید رو به جلو، مشابه با یادگیری مونته کارلو، پس از رسیدن به پایان قسمت، ارزش‌ها به روزرسانی می‌شوند. در حالت دید رو به عقب، ارزش‌ها به صورت آنلاین به روزرسانی می‌شوند. به این صورت که پس از دریافت یک پاداش، عامل مقادیر کیو n حالت قبلی خود را به روزرسانی می‌کند.

تی‌دی صفر

در ساده‌ترین حالت، عامل در حین انجام عمل، با دیدن یک گام در آینده یا گذشته، ارزش عمل فعلی را به روزرسانی می‌کند. به این روش تی‌دی صفر [۲۳] گفته می‌شود.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times (R_{t+1} + \gamma \times Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \qquad (۶-۲)$$

در واقع، به کمک رابطه بلمن (که فرم ارزش عمل آن در رابطه ۲-۴ دیده می‌شود) میزان صحیح بودن ارزش عمل فعلی، با ارزش عمل بعدی و پاداش فعلی مقایسه می‌شود و ارزش عمل فعلی به روزرسانی می‌شود.

[21] Offline Learning
[22] Temporal Difference Learning
[23] TD(0)

تی‌دی لامبدا رو به جلو

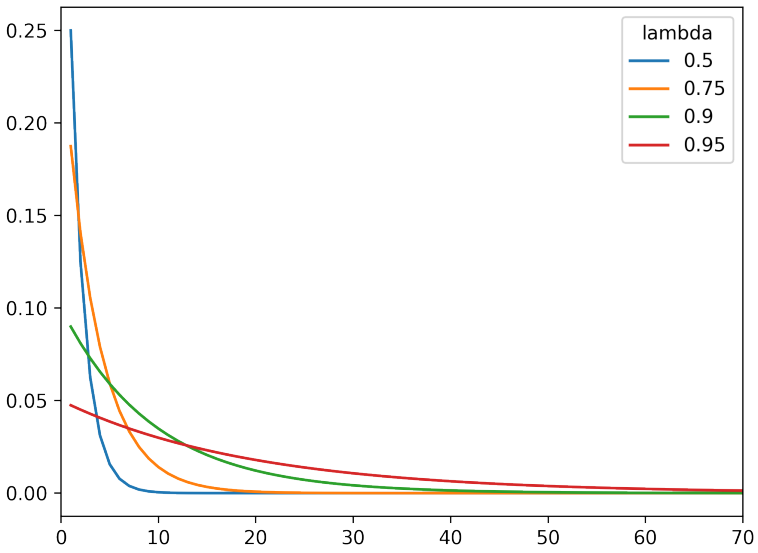
در روش تی‌دی λ، [۲۴] با دید رو به جلو به جای استفاده از یک گام در آینده ، از یک ترکیب خطی از پاداش‌ها و بازگشت چندین گام استفاده می‌شود. به این ترکیب خطی، λ-بازگشت [۲۵] گفته می‌شود.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times (G_t^{\lambda} - Q(s_t, a_t)) \tag{۷-۲}$$

که در این رابطه، G_t^{λ} به صورت زیر محاسبه می‌شود:

$$G_t^{\lambda} = (1 - \lambda) \times \sum_{n=1}^{\infty} \lambda^{n-1} \times G_t^{(n)} \tag{۸-۲}$$

که در آن $G_t^{(n)}$، مقدار بازگشتی بعد از n گام است، و λ یک پارامتر بین صفر و یک است که نشان‌دهنده‌ی اهمیت پاداش‌های آینده نسبت به پاداش‌های فعلی است. می‌توان میزان اهمیت پاداش‌های را به ازای مقادیر مختلف λ مشاهده کرد.



شکل ۲-۲: تاثیر پارامتر λ بر اهمیت پاداش‌های آینده. محور افقی نماینده تعداد گام‌ها، و محور عمودی نماینده وزن این بازگشت متناظر با آن است.

[24] TD(λ)

[25] λ-return

تی‌دی لامبدا رو به عقب

در روش تی‌دی λ با دید رو به عقب، از مفهومی به نام آثار شایستگی [۲۶] استفاده می‌شود. این مفهوم، نشان‌دهنده‌ی این است که هر عملی که در گذشته انجام شده و موجب تغییر ارزش عملی شده است، به چه میزان روی این تغییر سهیم بوده‌است.

ابتدا به ازای هر حالت و عمل، یک متغیر آثار شایستگی $E(s,a)$ صفر تعیین می‌شود و سپس در هر گام، این متغیر به صورت زیر به روزرسانی می‌شود:

$$E(s,a) = \gamma \times \lambda \times E(s,a) + 1(s = s_t, a = a_t) \quad (۹-۲)$$

در واقع کل آثار شایستگی در لامبدا ضرب شده، و یکی به آثار شایستگی عمل و حالت فعلی اضافه می‌شود. در این حالت، آثار شایستگی معادل با میزان نزدیکی زمانی هر حالت و عمل، به حالت و عمل فعلی است.

سپس ارزش عمل به صورت زیر به روزرسانی می‌شود:

$$Q(s,a) = Q(s,a) + \alpha \times \delta \times E(s,a) \quad (۱۰-۲)$$

که در این رابطه، δ خطای تخمین است که به صورت زیر محاسبه می‌شود:

$$\delta = R_{t+1} + \gamma \times (Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (۱۱-۲)$$

می‌توان نشان داد که پس از انجام یک قسمت، به‌روزرسانی‌های انجام شده توسط این روش، معادل با به‌روزرسانی‌های انجام شده توسط تی‌دی λ با دید رو به جلو است.

با ترکیب این روش و سیاست اپسیلون-حریصانه، می‌توان به یک روش یادگیری تقویتی کارا دست یافت که سارسا-لامبدا [۲۷] نام دارد.

[26] Eligibility Traces

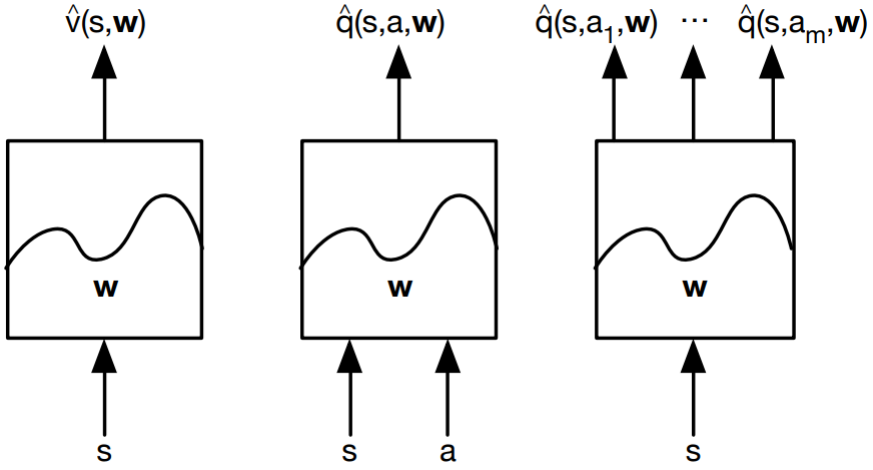
[27] SARSA(λ)

۲-۴ استفاده از تقریب‌گر تابع

با توجه به زیاد بودن تعداد حالت‌های ممکن در اکثر مسائل، یادگیری با روش‌های توضیح‌داده‌شده بسیار کند و حتی غیرمعقول می‌باشد. از این رو، به جای نگه‌داری مقادیر Q در یک جدول به ازای هر زوج حالت-عمل از تقریب‌گرهای تابع[۲۸] برای تخمین این مقادیر استفاده می‌کنیم.

از تقریب‌گرهای تابع مختلفی می‌توان برای این کار استفاده کرد. از جمله‌ی این تقریب‌گرها می‌توان به شبکه‌های عصبی، درخت‌های تصمیم، و توابع پایه‌ای مثل چندجمله‌ای‌ها اشاره کرد.

در واقع ابتدا از حالت، برخی ویژگی‌های مهم را استخراج کرده، سپس از این ویژگی‌ها برای تخمین ارزش‌ها به کمک پارامترهایی که قرار است یاد گرفته شوند استفاده می‌کنیم. رایج است که در روابط، به این پارامترها با θ یا w اشاره شود.



شکل ۲-۳: روش‌های مختلف استفاده از تقریب‌گر تابع

حال کافی‌ست که معادلات ۲-۶، ۲-۷، و ۲-۱۰ را به صورت تقریبی برای تقریب‌گر تابع بنویسیم، و به‌جای به روزرسانی مقادیر Q، پارامترهای تقریب‌گر تابع را به‌روز کنیم. با توجه به نیاز به به‌روزرسانی، لازم است از تقریب‌گرهای مشتق‌پذیر استفاده کنیم. همانطور که در عکس ۲-۳ می‌توان دید، در حالتی که از تقریب‌گرهای تابع برای پیش‌بینی مقادیر Q استفاده می‌شود، رایج است از عبارت $Q(s,a;\theta)$ یا $Q(s,a;w)$ برای نشان دادن این تقریب‌گرها استفاده کرد.

[28] Function Approximators

۲-۵ الگوریتم یادگیری Q عمیق

این الگوریتم، یکی از مستقیم‌ترین راه‌های استفاده از تقریب‌گرهای توابع، برای ترکیب یادگیری عمیق و تقویتی است. در این الگوریتم، هدف این است که مشابه با حالت سوم شکل ۲-۳، پارامترهای یک شبکه عصبی را به‌گونه‌ای تنظیم کنیم که مقادیر ارزش-عمل را بتواند تخمین بزند. از دستاوردهای این الگوریتم، عملکرد در حد انسان در چندین بازی آتاری [۲۹] است که توسط تیم دیپ‌مایند [۳۰] در سال ۲۰۱۵ به‌وقوع پیوست.

شبکه‌های عصبی الهام گرفته از ساختار و عملکرد مغز انسان هستند که برای یادگیری از داده‌ها و تصمیم‌گیری‌های پیچیده استفاده می‌شوند. این شبکه‌ها از واحدهای پردازشی به نام پرسپترون‌ها تشکیل شده‌اند که در لایه‌های مختلف قرار گرفته‌اند و از طریق وزن‌هایی به هم متصل می‌شوند.

یادگیری در شبکه‌های عصبی اغلب از طریق فرایندی به نام نزول گرادیان انجام می‌گیرد که در آن وزن‌های شبکه به صورت تکراری تنظیم می‌شوند تا خطا بین پیش‌بینی‌های شبکه و داده‌های واقعی به حداقل برسد. این فرایند شامل محاسبه گرادیان یا شیب تابع خطا نسبت به وزن‌ها و به‌روزرسانی وزن‌ها در جهت مخالف گرادیان برای کاهش خطا است.

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) \qquad (۱۲-۲)$$

در این فرمول، θ پارامترهای شبکه، α نرخ یادگیری، J تابع خطا، و ∇J گرادیان تابع خطا نسبت به پارامترها هستند.

برای اینکه یادگیری شبکه عصبی با ثبات بالاتری رخ دهد، این الگوریتم از دو تکنیک **بازیابی تجربه** و **استفاده از شبکه هدف** بهره می‌برد.

۲-۵-۱ بازیابی تجربه

برای رفع مشکلات داده‌های همبسته و توزیع‌های غیر ایستا در یادگیری آنلاین، DQN از یک مکانیزم بازیابی تجربه استفاده می‌کند. این شامل ذخیره تجربیات عامل در هر گام زمانی در یک بافر بازیابی و سپس نمونه‌برداری تصادفی ریزدسته‌ها [۳۱] از این بافر برای آموزش شبکه است. این رویکرد به شکستن

[29] Atari
[30] DeepMind
[31] Mini-batches

همبستگی بین نمونه‌های پیاپی کمک می‌کند و فرآیند یادگیری را پایدار می‌سازد.

۲-۵-۲ استفاده از شبکه هدف

DQN یک شبکه دوم به نام شبکه هدف را برای استقرار بیشتر آموزش معرفی می‌کند. شبکه هدف یک کپی از شبکه Q است، اما وزن‌های آن کمتر به‌روز می‌شوند. این جداسازی نوسان ارزش‌های هدف در به‌روزرسانی یادگیری Q را کاهش می‌دهد و خطر حلقه‌های بازخورد خودتقویتی را کاهش می‌دهد.

۲-۵-۳ فرآیند یادگیری

در ابتدا وزن‌های شبکه ارزش-عمل را به صورت تصادفی تنظیم می‌کنیم و آن را θ می‌نامیم. شبکه هدف را با وزن‌های یکسان مقداردهی می‌کنیم و آن را θ^- می‌نامیم. و بافر بازیابی را به طول N که یک ابرپارامتر است، مقداردهی می‌کنیم.

سپس در هر حالت، پاداش‌های خروجی شبکه اصلی را می‌گیریم و به صورت اپسیلون-حریصانه عمل را انتخاب می‌کنیم؛ یعنی به احتمال ϵ رفتار تصادفی داریم، و با احتمال $1-\epsilon$ رفتار با بالاترین ارزش-عمل پیش‌بینی شده را برمیداریم، یعنی: $a_t = \underset{a}{\operatorname{argmax}} Q(s_t, a; \theta)$. ترکیب چهارتایی حالت قبل عمل، عمل انتخاب شده، پاداش دریافتی، و حالت بعد عمل (s_t, a_t, r_t, s_{t+1}) را به بافر اضافه می‌کنیم. یک گروه با اندازه مشخص را با نمونه‌برداری تصادفی از بافر انتخاب می‌کنیم، و به صورت زیر یادگیری را انجام می‌دهیم:

$$y_j = \begin{cases} r_j & \text{اگر } s_{j+1} \text{ حالت پایانی قسمت باشد} \\ r_j + \gamma \times \max_{a'} Q(s_{j+1}, a'; \theta^-) & \text{در غیر این صورت} \end{cases} \quad (۱۳-۲)$$

در این معادله y_j تخمینی از میزان پاداش دریافتی واقعی است، که مشابه با روش‌های TD، از ترکیب پاداش دریافتی و ارزش-عمل بعدی به دست می‌آید. بنابرین می‌توان از این مقدار، مشابه با برچسب در یادگیری نظارت‌شده، برای آموزش شبکه استفاده کرد. میزان خطای شبکه به صورت زیر تعریف می‌شود:

$$L(\theta) = \mathbb{E}[(y_j - Q(s_j, a_j; \theta))^2] \quad (۱۴-۲)$$

که با استفاده از این خطا، می‌توان گرادیان تابع خطا نسبت به وزن‌های شبکه را به دست آورد و با استفاده از این گرادیان، می‌توان وزن‌های شبکه را به‌روز کرد:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta) \quad (۱۵-۲)$$

در نهایت کافی‌ست به صورت تناوبی و هر چند گام یک بار وزن‌های شبکه هدف را برابر با کپی وزن‌های شبکه اصلی قرار دهیم. با تکرار این مراحل، به شبکه عصبیی دست می‌یابیم که قدرت پیش‌بینی ارزش‌های عمل را دارد.

حال می‌توان اهمیت دو تکنیک گفته شده را بهتر فهمید: در صورت عدم استفاده از شبکه هدف، باید همزمان از شبکه اصلی برای پیش‌بینی آینده (در فرمول ۲-۱۳) و انتخاب عمل استفاده کنیم، که می‌تواند به مشکلاتی مانند نوسانات در آموزش و حلقه‌های بازخورد خودتقویتی منجر شود. همچنین در صورت عدم استفاده از بافر تجربه، باید صرفا از تجارب اخیر خود استفاده کنیم، که در این صورت هبستگی بین نمونه‌ها و توزیع‌های غیر ایستا می‌تواند مشکل‌ساز شود.

در بسیاری از پیاده‌سازی‌ها، مقدار اپسیلون که به آن نرخ کاوش هم گفته می‌شود، متغییر است و ابتدا زیاد بوده و در طول آموزش به تدریج کاهش می‌یابد و به مقدار ثابتی می‌رسد.

۲-۵-۴ اثرات برخی از ابرپارامترها

در این بخش به بررسی تاثیر تغییر ((اندازه ریزدسته)) و ((اندازه بافر تجربه)) بر عملکرد الگوریتم DQN می‌پردازیم.

اندازه ریزدسته

اندازه ریزدسته تعداد نمونه‌هایی است که از بافر تجربه برای آموزش شبکه استفاده می‌شود. با افزایش این مقدار، تنوع نمونه‌ها افزایش می‌یابد و از طرف دیگر، باعث افزایش پیچیدگی محاسباتی می‌شود. بنابرین باید یک تعادل بین زمان لازم برای هر به‌روزرسانی شبکه، و کیفیت آموزش شبکه داشته باشیم.

اندازه بافر تجربه

اندازه بافر تجربه تعداد نمونه‌هایی است که برای آموزش شبکه در طول یادگیری ذخیره می‌شود. با افزایش این تعداد، تنوع این تجربه‌ها افزایش می‌یابد و از طرف دیگر، باعث افزایش حافظه مصرفی می‌شود.

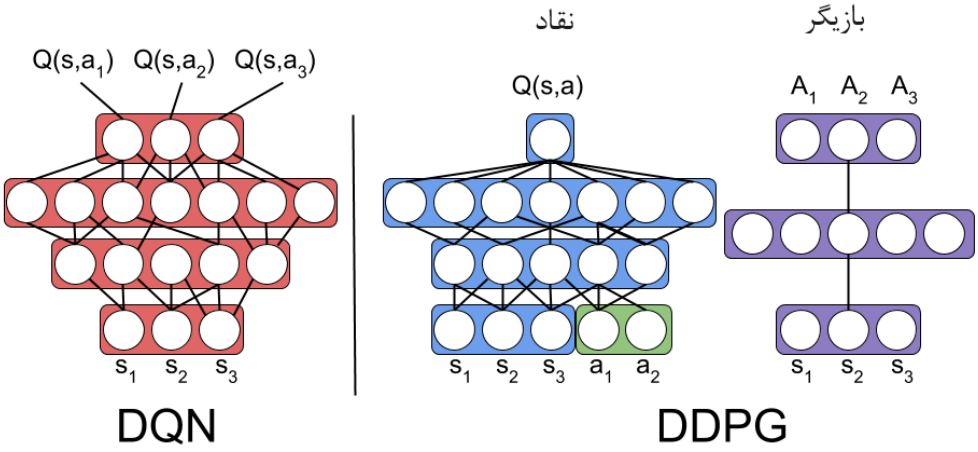
در صورت زیادی کوچک بودن این پارامتر، فقط تجارب اخیر در بافر ذخیره می‌شوند و از تجربیات گذشته استفاده نمی‌شود. بنابرین عامل ممکن است فراموشی تجربیات گذشته را تجربه کند و یادگیری آنلاین آن را مختل کند. از سوی دیگر، در صورت کوچک بودن تجربیات، احتمال وجود تجارب اخیر درون ریزدسته کاهش می‌یابد که باعث کند شدن فرآیند آموزش و عدم دریافت واکنش نسبت به امتحان کردن حالت‌های جدید می‌شود. بنابرین باید اندازه بافر نسبت به اندازه ریزدسته و همچنین طول کل آموزش به درستی انتخاب شود.

۲-۶ الگوریتم بهبود گرادیان سیاست معین عمیق

الگوریتم بهبود گرادیان سیاست معین عمیق [۳۲] یا به اختصار DDPG، یک الگوریتم یادگیری تقویتی است که برای حل مسائل پیوسته و فضای عمل پیوسته طراحی شده است. همانطور که در بخش‌های قبل دیدیم، برای انتخاب عمل معمولا از سیاست اپسیلون-حریصانه استفاده می‌شود، اما این روش برای فضای عمل پیوسته مناسب نیست، چرا که به‌دست‌آوردن بیشینه در فضای پیوسته به مراتب دشوارتر از حالت گسسته است. این الگوریتم از نوع یادگیری خارج از سیاست است؛ به این معنا که عامل با دنبال کردن سیاستی که سیاست انتخابی خود نیست به بهبود عملکرد خود می‌پردازد.

DDPG از دو شبکه عصبی استفاده می‌کند: یک شبکه برای تخمین ارزش عمل که به آن نقاد می‌گوییم، و یک شبکه برای تخمین سیاست و انتخاب عمل که به آن بازیگر می‌گوییم. شبکه نقاد مشابه با شبکه Q در DQN است، با این تفاوت که حالت و عمل را ورودی گرفته و ارزش عمل را خروجی می‌دهد. و شبکه سیاست یک شبکه عصبی است که خروجی آن عمل است. این الگوریتم مشابه با DQN از بافر تجربه برای ذخیره تجربیات عامل، و از شبکه‌های هدف برای هر دو شبکه نقاد و بازیگر استفاده می‌کند. البته در این الگوریتم، به‌جای کپی وزن‌های شبکه به شبکه هدف، از میانگین وزن‌دار استفاده می‌شود تا به روزرسانی آرام‌تر انجام شود و ثبات بیشتری داشته باشد.

[32] Deep Deterministic Policy Gradient



شکل ۲-۴: شبکه‌های عصبی بازیگر و نقاد در الگوریتم DDPG و مقایسه با DQN

رایج است که شبکه ارزش عمل را با $Q(s,a;\phi)$ و شبکه سیاست را با $\mu(s;\theta)$ نشان دهیم. فرآیند یادگیری این الگوریتم به صورت زیر است:

۱. ابتدا وزن‌های شبکه‌های نقاد و بازیگر را به صورت تصادفی مقداردهی می‌کنیم و آن‌ها را به ترتیب ϕ و θ می‌نامیم.

۲. بافر تجربه را به طول N مقداردهی می‌کنیم.

۳. در هر گام زمانی، حالت را به شبکه بازیگر می‌دهیم و عملی که این شبکه پیشنهاد می‌دهد را انجام می‌دهیم.

۴. پاداش دریافتی و حالت بعد عمل را به بافر تجربه اضافه می‌کنیم.

۵. یک ریزدسته از بافر تجربه را انتخاب کرده و با استفاده از فرمول زیر، مقدار واقعی ارزش حالت را (طبق الگوریتم TD) محاسبه می‌کنیم:

$$y_j = \begin{cases} r_j & \text{اگر } s_{j+1} \text{ حالت پایانی قسمت باشد} \\ r_j + \gamma \times Q(s_{j+1}, \mu(s_{j+1};\theta^-);\phi^-) & \text{در غیر این صورت} \end{cases} \quad (۱۶-۲)$$

۶. خطای شبکه نقاد را به صورت زیر محاسبه می‌کنیم:

$$L(\phi) = \mathbb{E}[(y_j - Q(s_j, a_j;\phi))^2] \quad (۱۷-۲)$$

و وزن‌های شبکه نقاد را به‌روز می‌کنیم:

$$\phi_{t+1} = \phi_t - \alpha \nabla_\phi L(\phi) \tag{۱۸-۲}$$

۷. خطای شبکه بازیگر را به صورت زیر محاسبه می‌کنیم:

$$L(\theta) = -\mathbb{E}[Q(s, \mu(s;\theta);\phi)] \tag{۱۹-۲}$$

و وزن‌های شبکه بازیگر را به‌روز می‌کنیم:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta L(\theta) \tag{۲۰-۲}$$

۸. در نهایت هر چند گام یک بار، وزن‌های شبکه‌های هدف را به‌روز می‌کنیم:

$$\theta^-_{t+1} = \tau\theta_t + (1-\tau)\theta^-_t \tag{۲۱-۲}$$

$$\phi^-_{t+1} = \tau\phi_t + (1-\tau)\phi^-_t \tag{۲۲-۲}$$

همانطور که مشاهده‌شد، این الگوریتم تعداد زیادی پارامتر دارد که باید به‌درستی تنظیم شوند تا به عملکرد بهینه برسد. از جمله این پارامترها می‌توان به نرخ یادگیری (α که می‌تواند برای شبکه‌های نقاد و بازیگر متفاوت باشد)، ، نرخ تخفیف (γ)، فرکانس به‌روزرسانی شبکه‌ها (هر چند گام یک بار ریزدسته انتخاب می‌کنیم و مراحل ۵ تا ۸ را طی می‌کنیم)، فرکانس به‌روزرسانی وزن‌های هدف (هر چند گام یک بار وزن‌های شبکه‌های هدف را به‌روزرسانی می‌کنیم) ، اندازه ریزدسته، طول بافر تجربه، و ضریب میانگین‌گیری برای به‌روزرسانی شبکه‌های هدف (τ) اشاره کرد.

۲-۷ جمع‌بندی

در این فصل، ابتدا به معنا و اهمیت یادگیری تقویتی پرداختیم و سپس به معرفی مفاهیم اصلی این حوزه پرداختیم. سپس به معرفی الگوریتم‌های یادگیری تقویتی کلاسیک بر مبنای مدل همچون برنامه‌نویسی پویا و روش‌های مبتنی بر ارزش و سیاست پرداختیم. در ادامه به معرفی الگوریتم‌های یادگیری تقویتی بدون مدل پرداختیم که از تقریب‌گرهای تابع برای تخمین ارزش‌ها استفاده می‌کنند. در این بخش، به معرفی دو الگوریتم معروف یادگیری تقویتی بدون مدل، یعنی DQN و DDPG پرداختیم. در ادامه، در فصل بعد به معرفی یک مسئله و کاربرد این الگوریتم‌ها در حل آن خواهیم پرداخت.

فصل سوم

محیط فوتبال ربوکاپ

در این بخش به معرفی مفاهیم پایه محیط فوتبال ربوکاپ و توصیف چگونگی عملکرد آن می‌پردازیم. این مفاهیم شامل قوانین بازی، رفتارهای ممکن، کد پایه ایجنت، حالت پنالتی، و روش اجرای بازی می‌باشند.

۳-۱ معرفی لیگ

۳-۱-۱ اهداف لیگ

لیگ ربوکاپ مجموعه مسابقاتی سالانه است که قصد دارد با کمک فوتبال، به پیشرفت زمینه‌های رباتیک و هوش مصنوعی کمک کند. علت انتخاب فوتبال به عنوان محیط مسابقه، این است که فوتبال یکی از محیط‌هایی است که می‌تواند مسائل مختلفی از جمله تصمیم‌گیری، هماهنگی، بینایی ربات، و ارتباط را در بر داشته باشد. یکی از اهداف بلندمدت لیگ، ساختن ربات‌هایی است که بتوانند تا سال ۲۰۵۰، تیمی از انسان‌ها را به چالش بکشند.

یکی از لیگ‌های این مسابقات، لیگ شبیه‌ساز فوتبال دو بعدی[1] است. همانطور که از اسم لیگ پیداست، مسابقه حالت دو بعدی دارد، به این منظور که فضا حالت دید از بالا دارد، و همه حرکات بازیکنان و توپ روی سطح زمین انجام می‌شود. تمرکز اصلی این لیگ، تصمیم‌گیری و استراتژی، و ساختن الگوریتم‌های مناسب با محیط‌های چندعامله با دید ناقص است.

۳-۱-۲ ویژگی‌های محیط و قوانین بازی

مشابه با فوتبال واقعی، هر بازی متشکل از دو تیم ۱۱ نفره است که هر کدام از این نفرات یک عامل مستقل می‌باشد. بازی در یک مستطیل به ابعاد ۱۱۵ در ۶۸ انجام می‌شود، و هر تیم یک دروازه‌بان دارد که در هر طرف زمین قرار دارد. بازی در ۶۰۰۰ گام[2] انجام می‌شود که به دو نیمه تقسیم شده، و هر گام ۱۰۰ میلی‌ثانیه طول می‌کشد. سرور مسابقات در هر گام، اطلاعات مربوط به وضعیت بازی را به تمامی عامل‌ها می‌فرستد، و عامل‌ها باید تصمیم‌گیری خود را بر اساس این اطلاعات انجام دهند.

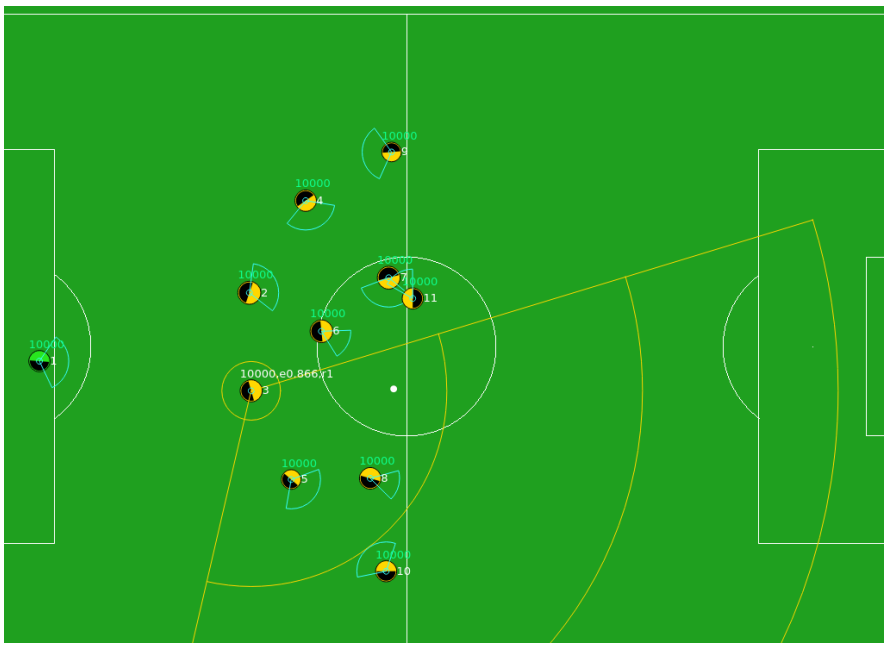
بازیکنان دید محدودی دارند. با توجه به زاویه گردنی که تنظیم کرده‌اند، یک قطاع از زمین را مشاهده می‌کنند و فقط اطلاعات مربوط به بازیکنان و اشیاء داخل این قطاع را دریافت می‌کنند در شکل ۳-۱ نمونه دید یک بازیکن نشان داده شده است. اطلاعات بازیکنانی که داخل دید نیستند، باید از حافظه بازیکن و یا از ارتباط با سایر بازیکنان به دست آید. شایان به ذکر است که مشاهدات هر بازیکن، دارای

[1] 2D Soccer Simulation

[2] Cycle

کمی خطای اندازه‌گیری است و به طور کامل دقیق نیست. به طور مشابه، ضربات بازیکن به توپ نیز کمی خطا دارند و کامل دقیق نیستند.

هر عامل باید به صورت مستقل تصمیم‌گیری کند و ارتباط بسیار ناچیزی با سایر عاملین دارد. در صورتی که دو عامل بخواهند ارتباط برقرار کنند، گیرنده باید از قبل توجه خود را به فرستنده تنظیم کرده‌باشد، و فرستنده حداکثر ۸ بایت می‌تواند ارسال کند.



شکل ۳-۱: نمونه دید یک بازیکن. در این مثال، بازیکن ۳، بازیکن ۵ و ۸ و ۱۰ و توپ را می‌بینند و از موقعیت سایر بازیکنان اطلاعی ندارد.

هر عامل یک رزرو انرژی و یک منبع انرژی دارد. در صورت پر نبودن انرژی بازیکن، انرژی به نرخ ثابتی از رزرو به منبع منتقل می‌شود. در صورت اتمام انرژی منبع، رفتارهای بازیکن از جمله حرکت‌کردن کندتر می‌شوند. رفتارهای بیشتری (مانند تکل زدن، خطا و ضربه آزاد، کارت زرد و قرمز و ...) نیز در محیط به کمک مدل‌های ریاضیاتی پیاده‌سازی شده‌است که خارج از دامنه این پروژه می‌باشد.

۳-۲ کد پایه ایجنت

در سال‌های اول مسابقات، هر تیمی کد عامل خود را از صفر می‌نوشت، که میزان در دسترس بودن لیگ را بسیار پایین آورده‌بود. با توجه به یکسان بودن بخش عمده‌ای از نیازمندی‌های تیم‌ها، همچون نیاز به اتصال به سرور، نیاز به تفکیک وظایف بازیکنان به دروازه‌بان و مدافع و مهاجم، نیاز به موقعیت‌یابی اشیا

در زمین، توابع هندسی، و ...، هیدهیسا آکیام از تیم هلیوس در سال ۲۰۱۲ تصمیم به ساختن یک کد پایه به صورت متن‌باز محض استفاده سایر تیم‌ها گرفت. این کد پایه، که کد پایه ایجنت نام دارد، زیربنای ۱۳ تیم از ۱۵ شرکت‌کننده سال اخیر ربوکاپ بوده، و نقطه شروع اکثر کسانی است که قصد فعالیت در این فضا را دارند. این کد پایه همچنان در حال به‌روزرسانی و تقویت‌شدن است [۳] ، و خود منشا سایر کد پایه‌های به اشتراک گذاشته‌شده همچون کد پایه گلایدر و کد پایه سایرس است.

با استفاده از این کد پایه، توسعه‌دهندگان می‌توانند سطح کدزدن خود را از سطوح پایین، به سطح استراتژی و تاکتیک و تصمیم‌گیری منتقل کنند. به طور مثال بدون استفاده از یک کد پایه، برای حرکت عامل به مرکز زمین باید کد اتصال به سرور و موقعیت‌یابی را پیاده‌سازی کنیم. سپس درگیر مسائلی همچون محاسبه شتاب بازیکن، نیرویی که باید اعمال شود، بهینه‌ترین مسیر حرکت (چرخیدن و دویدن یا دویدن مورب) و ... شوند. در حالی که این عمل، به صورت یک دستور سطح بالا در کد پایه ایجنت قابل اجراست. در بخش بعدی به تمام رفتارهای ممکن در این کد پایه و سطح‌بندی رفتارها می‌پردازیم.

۳-۳ معرفی رفتارهای ممکن

هر عامل در هر لحظه، باید تصمیم‌گیری خود را به سرور بفرستد. این تصمیم‌گیری می‌تواند شامل انجام یکی از رفتارهای ممکن باشد. در هر لحظه، عامل می‌تواند گردن خود را بچرخاند تا اطراف خود را ببیند، و همزمان یکی از پنج رفتار ضربه به توپ، حرکت بدن، چرخش بدن، تکل، یا گرفتن توپ را انجام دهد. رایج است که رفتارهای ممکن و پیاده‌سازی شده را به سه طبقه تقسیم‌بندی کنیم که در پایین‌ترین سطح، رفتارهای سطح سرور، و در بالاترین سطح، رفتارهای سطح استراتژیک و فکری قرار می‌گیرند.

۳-۳-۱ رفتارهای سطح پایین

در پایین‌ترین سطح، رفتارها مستقیما معادل با رفتارهای مورد پذیرش سرور می‌باشند. این رفتارها شامل اعمال نیرو روی توپ در یک راستا (نسبت به بدن بازیکن) و نیروی خاص، چرخاندن بازیکن به یک راستا، اعمال نیروی حرکت بازیکن در راستا و نیرو، تکل زدن در یک راستا، و گرفتن توپ (در صورتی که بازیکن دروازه‌بان باشد) می‌باشند.

[3] https://github.com/helios-base/helios-base

۳-۳-۲ رفتارهای سطح متوسط

در این سطح، رفتارها ساده‌سازی شده‌اند تا استفاده از آن‌ها برای تصمیم‌گیری راحت‌تر باشد. به طور مثال رفتار حرکت به سمت یک نقطه خاص از زمین، رفتار ضربه زدن به توپ با سرعت دلخواه در راستای غیرنسبی، رفتار ضربه زدن توپ به صورت چندضرب به یک نقطه خاص، و... را می‌توان از رفتارهای این سطح معرفی کرد.

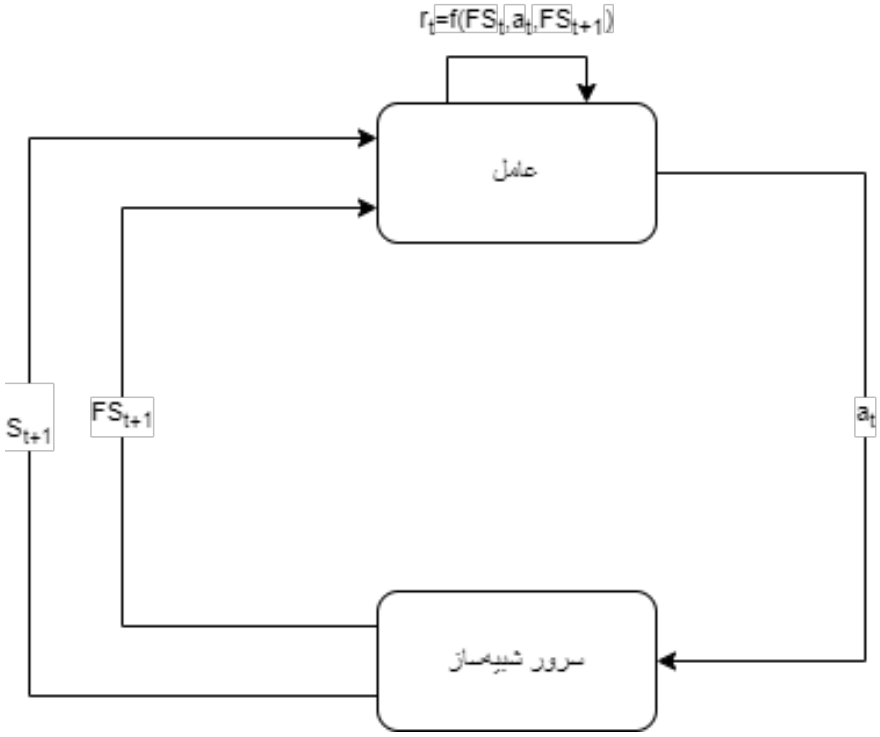
۳-۳-۳ رفتارهای سطح بالا

در این سطح، رفتارها حالت استراتژی و فکر کردن دارند، و به صورت انتظاعی و مشابه با فوتبال واقعی می‌باشند. به طور مثال، حرکت به سمت نقطه‌ی فرمیشن، حرکت برای قطع توپ، زدن شوت (در صورت ممکن بودن شوت موفق)، و... از رفتارهای سطح بالا می‌باشند.

۳-۴ حالت دید کامل

همانطور که گفته‌شد، محیط اطلاعات کامل را در اختیار عامل قرار نمی‌دهد. با توجه به اینکه قصد تغییر کد سرور مسابقات و افزودن پاداش به سرور را نداریم، این محاسبات درون خود عامل باید صورت بگیرند. خوش‌بختانه سرور مسابقات قابلیت ارسال محیط در حالت دید کامل [۴] را دارد که در این حالت، همه اطلاعات بدون نویز به عامل ارسال می‌شوند. داخل کد ایجنت می‌توان انتخاب کرد که در صورت دریافت حالت دید کامل آن‌را جایگزین دید ناقص کند، یا اینکه هر دو را کنار هم نگه‌دارد. ما در فرآیند یادگیری تقویتی از حالت دوم استفاده می‌کنیم تا با کمک دید کامل محاسبات پاداش را انجام دهیم، و با حالت دید ناقص تصمیم‌گیری کنیم. در عکس ۳-۲ می‌توان نحوه مشاهده کرد که پاداش داخل عامل محاسبه‌شده، و تابعی از حالت دید کامل (Fullstate یا به مختصر FS) است، و محیط هر دو حالت دید را برای عامل ارسال می‌کند. این نمودار در واقع حالت اصلاح‌شده ۲-۱ است که خاص‌منظوره این پروژه می‌باشد.

[4] Fullstate WorldModel



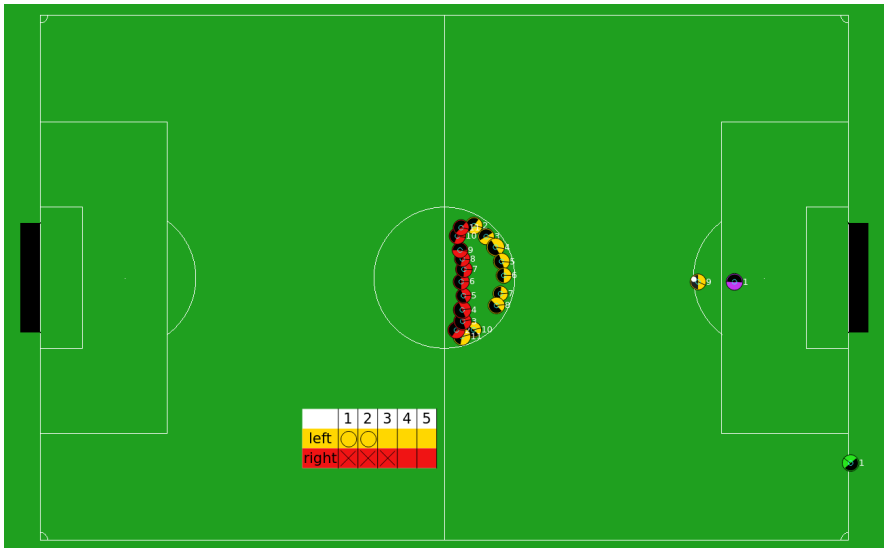
شکل ۳-۲: ارتباط بین محیط و عامل، و روش محاسبه پاداش

۳-۵ معرفی حالت پنالتی

در صورت تساوی بازی در ده دقیقه زمان عادی، و تداوم تساوی پس از وقت اضافی، بازی به حالت پنالتی می‌رود. با توجه به دو بعدی بودن محیط، در صورتی که مشابه با فوتبال واقعی، پنالتی به صورت تک ضرب باشد، می‌توان برای آن استراتژی قطعی ارائه داد. به همین منظور، پنالتی در فضای لیگ دوبعدی، به صورت تک به تک با دروازه‌بان است.

بازیکن مهاجم، ۱۵ ثانیه فرصت دارد تا توپ را به گل تبدیل کند. دروازه‌بان باید در این مدت تلاش کند جوری موقعیت‌گیری کند که حریف نتواند شوت منجر به گل داشته‌باشد. در صورتی که زمان مهاجم تمام شود، توپ به بیرون محیط بازی برود، یا توپ توسط دروازه‌بان گرفته‌شود، دروازه‌بان برنده می‌شود. در صورتی که توپ وارد گل شود، مهاجم برنده می‌شود.

با توجه به تک‌عامله بودن، و محدود بودن زمان محیط، می‌توان از روش‌های یادگیری تقویتی برای یادگیری استراتژی‌های بهینه برای پنالتی استفاده کرد. در شکل ۳-۳ یک نمونه از حالت پنالتی را مشاهده می‌کنید.



شکل ۳-۳: نمونه‌ای از بازی در حالت پنالتی

۳-۶ کار با مربی تمرینی برای تولید محیط قابل تکرار

با توجه به اینکه برای یادگیری تقویتی، نیاز به بازگرداندن محیط به حالت اولیه را داریم، می‌توانیم از مربی تمرینی [۵] استفاده کنیم. در صورتی که در تنظیمات سرور، این حالت فعال شده باشد، می‌توان کدی نوشت که در شرایط دلخواه، توپ و بازیکنان را جابه‌جا کنیم، حالت بازی را تنظیم کنیم، انرژی بازیکنان را پر کنیم و ... . در فصل‌های آینده از این امکان، برای ساختن یک رابط استاندارد یادگیری تقویتی برای محیط فوتبال استفاده خواهیم کرد.

۳-۷ جمع‌بندی

در این فصل، محیط فوتبال ربوکاپ دوبعدی به عنوان یک ابزار مطالعاتی برای تقویت پیشرفت‌ها در زمینه هوش مصنوعی و رباتیک معرفی شد. این محیط، با ارائه یک پلتفرم مسابقه‌ای مبتنی بر قوانین فوتبال واقعی، فرصت‌هایی برای توسعه و آزمایش الگوریتم‌های تصمیم‌گیری، هماهنگی، و استراتژی در محیط‌های چندعامله فراهم می‌کند.

معرفی این محیط شامل توضیح قوانین بازی، نحوه ارتباط و تصمیم‌گیری عامل‌ها، و همچنین ساختار کد پایه ایجنت بود که توسعه‌دهندگان را قادر می‌سازد تا تمرکز خود را بر روی تاکتیک و تصمیم‌گیری‌های

[5] Trainer

سطح بالای مشابه با فوتبال معطوف دارند.

یکی از نوآوری‌های کلیدی در این فصل، استفاده از حالت دید کامل در کنار دید ناقص است که به عامل‌ها امکان می‌دهد تا پاداش‌ها را بر اساس اطلاعات کامل محیط محاسبه کنند، در حالی که تصمیم‌گیری بر اساس دید ناقص انجام می‌شود. این رویکرد یک قدم مهم در راستای افزایش قابلیت‌های یادگیری تقویتی در محیط‌هایی با دید ناقص است.

همچنین، با معرفی حالت پنالتی و استفاده از مربی تمرینی برای تولید محیط‌های قابل تکرار، زمینه‌های لازم برای پیاده‌سازی الگوریتم‌های یادگیری تقویتی فراهم شده است. این امکانات به پژوهشگران اجازه می‌دهد تا در یک محیط مسابقه‌ای استاندارد، استراتژی‌های بهینه‌سازی شده را آزمایش و توسعه دهند.

فصل چهارم

پیاده‌سازی و آماده‌سازی محیط استاندارد

۴-۱ زیرساخت پایتونی تهاجم نصف زمین

در سال ۲۰۱۵، پروژه تهاجم نصف زمین [۱] یا همان اچ‌اف‌او تلاش کرد محیط پایتونیی برای یادگیری تقویتی در فوتبال دو بعدی ایجاد کند. کد سرور به گونه‌ای تغییر یافته‌بود، که بازیکن بتواند به سرور دستور اعمال رفتارها و رفتن به گام بعدی را بدهد.

این روش متکی بر قابلیت‌های لیب‌سی [۲] بود، که از آن برای ارتباط با کد cpp استفاده می‌کرد. لیب‌سی اجازه می‌دهد که اگر کلاس معادل پایتونی و cpp وجود داشته باشد، از سمت پایتون می‌توان به آن دسترسی داشت. در این پروژه، عامل پایتونی به عنوان ایجنت دسترسی به محیط دارد، که با آن از کد cpp محیط را درخواست می‌کند. محیط در دو سطح بالا و پایین قابل درخواست است؛ حالت سطح پایین ۶۰ ویژگی محیط، و حالت سطح بالا ۹ ویژگی را به صورت یک آرایه صفر تا یک به عامل پس می‌دهد. عامل سپس تصمیم خود را اخذ کرده، و به سرور دستور اجرای آن را می‌دهد. معایب استفاده از این محیط عبارت بود از:

- دشواری در تغییر محیط عامل به صورت دلخواه
- تفاوت زیاد بین محیط آماده‌شده در ایجنت، که مورد استفاده اکثر تیم‌هاست، و محیط اچ‌اف‌او
- عقب‌ماندگی محیط به علت تغییر کدهای سرور و به روز نبودن با تغییرات شبیه‌ساز
- دشواری نصب، به علت پیش‌نیازهای قدیمی همچون Qt4
- عدم امکان استفاده از تیم‌های جدید برای حریف یادگیری به علت نسخه قدیمی سرور

۴-۲ کد پایه پایرس

با توجه به دشواری استفاده از یادگیری ماشین در CPP، ما در تیم سایرس تصمیم به پیاده‌سازی یک کد پایه معادل با کد پایه Agent2D در پایتون گرفتیم، که نام آن پایرس [۳] است.[؟] این تلاش‌ها از سال ۲۰۱۹ آغاز شد و در نهایت بعد از سه سال و حدود ۲۵ هزار خط کد، پروژه به حالت پایدار و قابل استفاده رسیده‌است.[۴]

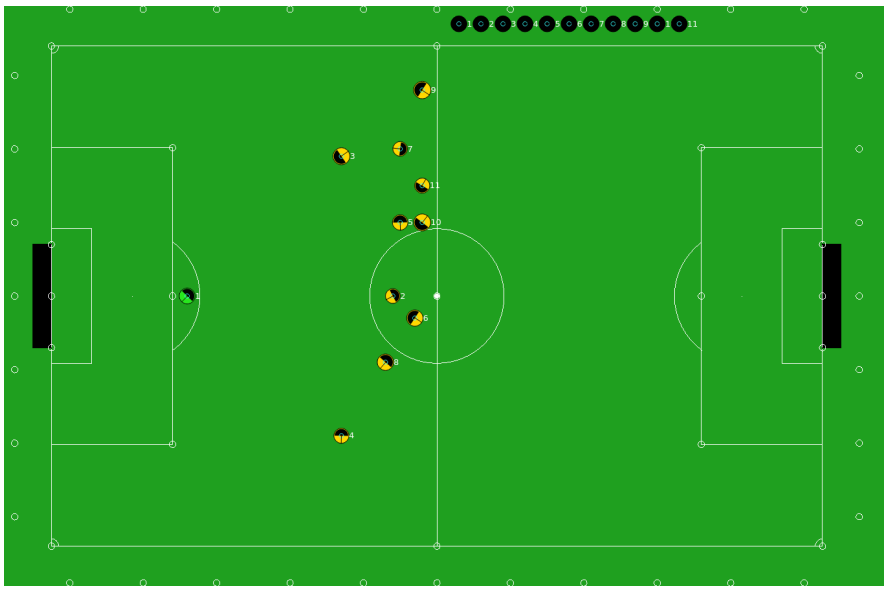
[1] Half Field Offense (HFO)

[2] LibC

[3] Pyrus

[4] https://github.com/Cyrus2D/Pyrus2D

امید بر آن بود که با توجه به اتکا بر الگوریتم‌های یادگیری ماشین به‌جای الگوریتم‌های درخت و گراف، بتوان کندی زبان پایتون را جایگزین کرد. در کد Agent2D، قبل از رسیدن به گام تصمیم‌گیری، محاسبات زیادی رخ می‌دهد تا بازیکن موقعیت خود و سایر بازیکنان را بیابد، سریعترین بازیکنان به توپ، موقعیت آفساید و... را به دست آورد. با توجه به کند بودن پایتون در مقایسه با cpp، همین محاسبات مقدماتی نیز بسیار زمان‌بر بودند و بخش بزرگی از ۱۰۰ میلی‌ثانیه موجود برای هر گام بازی را اشغال می‌کردند. به همین منظور، پایرس جایگزین مناسبی برای پایتونی کردن کد نبود.



شکل ۴-۱: پرچم‌های کنار زمین، که برای موقعیت‌یابی استفاده می‌شوند.

۴-۳ کد پایه جی‌آرپی‌سی

با توجه به سریع بودن cpp برای پیش‌پردازش، و کاربردی و راحت بودن پایتون برای یادگیری تقویتی، تصمیم گرفتیم به کمک پروتکل فراخوانی تابع از راه دور [۵]، این دو زبان را به هم متصل کنیم. در این پروژه از چارچوب فراخوانی تابع از راه دور گوگل، یا همان جی‌آرپی‌سی [۶] استنفاده می‌کنیم.

در این چارچوب، ابتدا یک فایل پروتو [۷] باید تعریف کرد، که حاوی مشخصات اشیاء و امضای توابع است. سپس با استفاده از کامپایلر پروتو می‌توان کدهای سرور و کلاینت را به زبان‌های دلخواه تبدیل کرد. سپس با اضافه کردن کد تولیدشده توسط کامپایلر پروتو می‌توان درون کلاینت از توابع به‌گونه‌ای

[5] Remote Procedure Call (RPC)

[6] gRPC (google Remote Procedure Call)

[7] Proto

استفاده کرد که گویا داخل خود کد قرار دارند. از مهم‌ترین مزایای این پروتکل ارتباطی، مستقل از زبان بودن آن، و نوشتن بسته‌های پیام به صورت صفر و یکی (در مقابل متنی) است که سربار زمانی نوشتن و خواندن پیام را ناچیز می‌کند. همچنین به علت استفاده از پروتکل اچ‌تی‌تی‌پی [8] برای ارسال بسته‌ها روی شبکه، می‌توان از ضمانت‌های دریافت پیام نیز استفاده کرد.

```
message Vector2D {
  float x = 1;
  float y = 2;
  float dist = 3;
  float angle = 4;
}
```

شکل ۴-۲: نحوه تعریف اشیا در جی‌آرپی‌سی

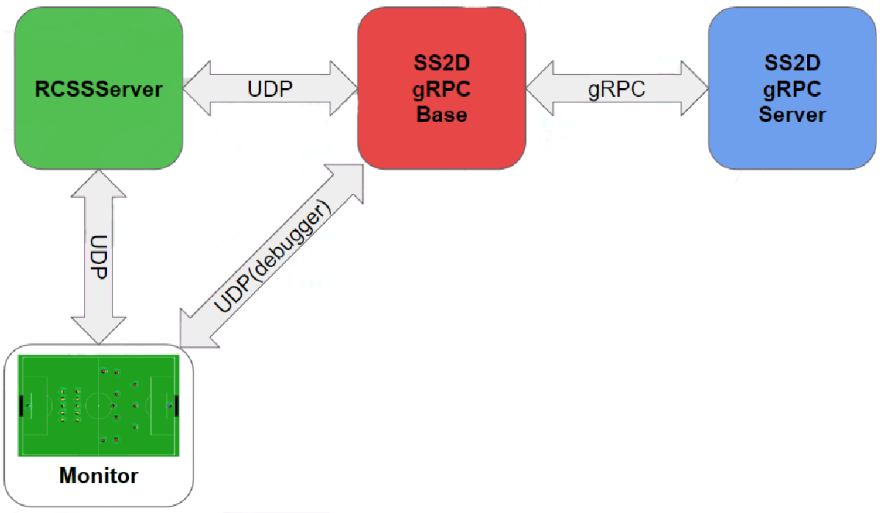
```
service Game {
  rpc GetPlayerActions(State) returns (PlayerActions) {}
  rpc GetCoachActions(State) returns (CoachActions) {}
  rpc GetTrainerActions(State) returns (TrainerActions) {}
  rpc SendInitMessage(InitMessage) returns (Empty) {}
  rpc SendServerParams(ServerParam) returns (Empty) {}
  rpc SendPlayerParams(PlayerParam) returns (Empty) {}
  rpc SendPlayerType(PlayerType) returns (Empty) {}
}
```

شکل ۴-۳: نحوه تعریف توابع در جی‌آرپی‌سی

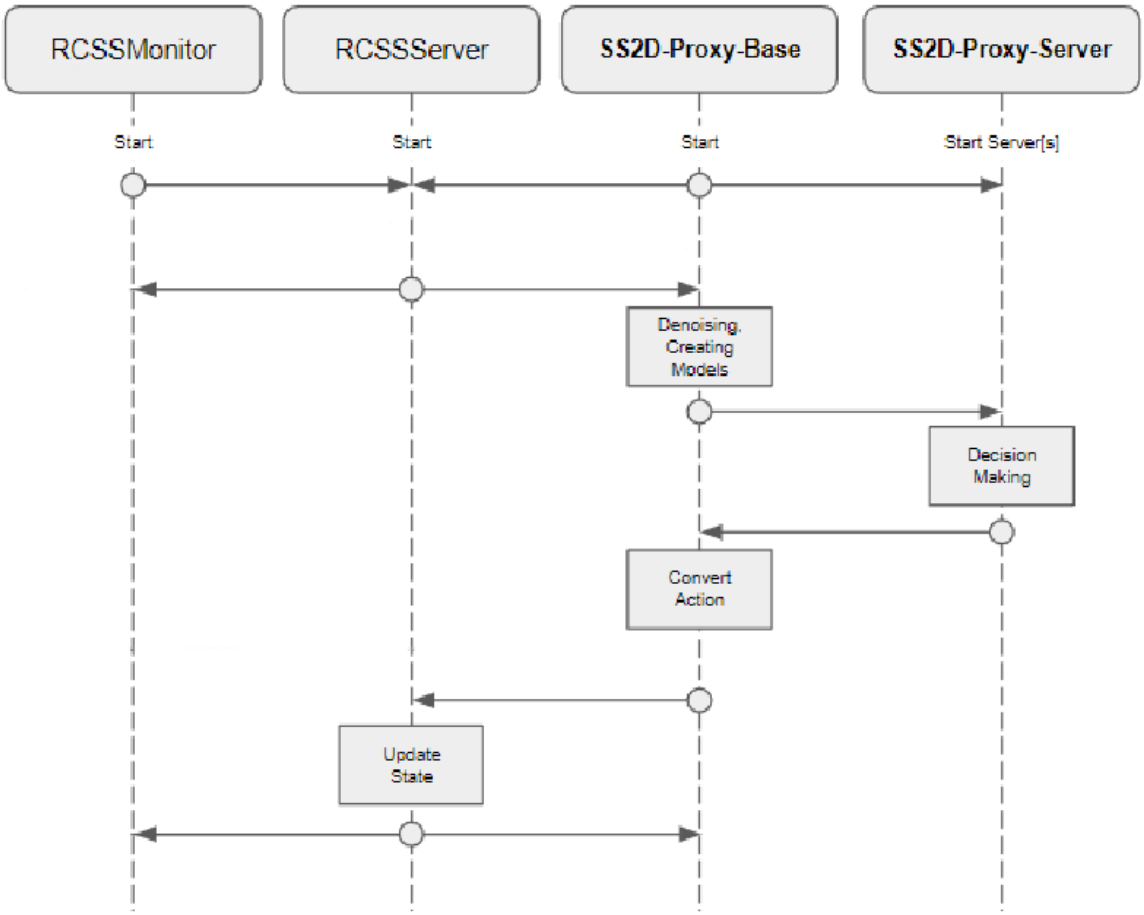
برای تشکیل اتصال بین این دو زبان کد پایه Agent2D را به‌گونه‌ای تغییر دادیم که پس از اتمام مراحل پیش‌پردازش و آماده شدن مدل دنیای بازیکن [9] در cpp، اطلاعات آن را با یک درخواست جی‌آرپی‌سی به یک سرور تصمیم‌گیری پایتونی بفرستد. در شکل ۴-۴ و ۴-۵ نحوه‌ی اتصال اجزای مسابقه، و روند عمومی تصمیم‌گیری نمایش داده شده‌است.

[8] HTTP

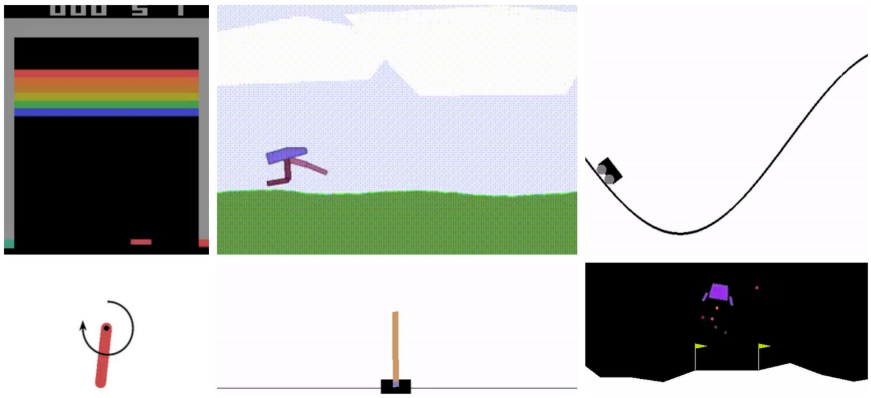
[9] WorldModel



شکل ۴-۴: پروتکل‌های استفاده‌شده برای ارتباط بین اجزای مسابقه



شکل ۴-۵: نحوه کارکرد و اتصال کد پایه جی‌آرپی‌سی به سرور مسابقات و نمایشگر بازی



شکل ۴-۶: نمونه‌ای از محیط‌های آماده شده در پلتفرم جیم

۴-۴ محیط استاندارد جیم

پلتفرم جیم [۱۰] که توسط گروه اپن‌ای‌ای [۱۱] توسعه داده شده‌است، یک محیط استاندارد برای یادگیری تقویتی است. این پلتفرم شامل محیط‌های گوناگون از پیش آماده شده است که می‌توانند برای یادگیری تقویتی استفاده شود. این محیط همچنین به ما قابلیت تعریف محیط‌های جدید را می‌دهد. در سال ۲۰۲۳، مدیریت و نگه‌داری این پلتفرم به شرکت فاراما [۱۲] واگذار شد و از آن زمان، این پلتفرم به نام پلتفرم جیمنازیوم [۱۳] شناخته می‌شود. تبدیل محیط یادگیری به این نوع محیط، نه تنها به ما قابلیت استفاده از کتاب‌خانه‌های پیشین و کمک گرفتن از ابزارهای از پیش تعبیه شده را می‌دهد، بلکه امکان به اشتراک‌گذاری راحت این محیط به سایر محققان را نیز فراهم می‌کند.

۴-۴-۱ توصیف رابط و توابع موجود

برای تعریف یک محیط جدید در جیم، باید توابع استاندارد آن را پیاده‌سازی کنیم. به این منظور، لازم است با روش کارکرد هر جزء محیط آشنا شویم. لازم به ذکر است که تفاوت‌های جزئیی بین نسخه اوپن‌ای‌ای و فاراما وجود دارد، که در اینجا به توصیف نسخه فاراما می‌پردازیم.

[10] Gym
[11] OpenAI
[12] Farama
[13] Gymnasium

تابع گام برداشتن

این تابع، یک عمل را به عنوان ورودی می‌گیرد، و وضعیت جدید، پاداش این عمل، و اطلاعاتی همچون اتمام بازی، یا تمام شدن وقت عامل را به عنوان خروجی برمی‌گرداند.

تابع شروع مجدد

این تابع، محیط را به حالت اولیه باز می‌گرداند، و حالت جدید را به عامل برمی‌گرداند. در صورتی که شروع بازی شامل المان‌های تصادفی باشد، ورودی این تابع باید کاوش تصادفی [۱۴] را نیز به عنوان ورودی بگیرد.

انواع فضاهای حالت و عمل

در حین تعریف محیط، باید فضاهای حالت و عمل را نیز تعریف کنیم. هر فضا می‌تواند از انواع زیر باشد:

- گسسته [۱۵]: به ازای ورودی n، در این فضا می‌توانیم از اعداد صحیح از ۰ تا n-1 استفاده کنیم. این حالت بیشتر برای فضای خروجی استفاده می‌شود.
- گسسته چندتایی [۱۶]: این فضا مشابه فضای گسسته است، با این تفاوت که می‌توانیم چند عدد از این فضا را به عنوان خروجی بگیریم. مشابه با گسسته است، ولی جای ورودی یک عدد، یک آرایه از اعداد است؛ هر عدد نشان‌دهنده تعداد حالت‌های ممکن در هر بعد است.
- فضای بسته [۱۷]: نشان‌دهنده یک فضای چندبعدی پیوسته است، که باید حد پایین و بالای هر بعد را مشخص کنیم.
- دوتایی چندبعدی [۱۸]: حالت چندبعدی است، که هر بعد مقدار صفر یا یک می‌تواند به‌خود بگیرد.

به کمک فضاهای فوق، می‌توان فضاهایی از ترکیب حالت‌های فوق تعریف کرد، که به انواع زیر است:

- دیکشنری [۱۹]: یک دیکشنری پایتونی از فضاهای مختلف است.

[14] Random Seed
[15] Discrete
[16] MultiDiscrete
[17] Box
[18] MultiBinary
[19] Dict

- چندتایی (توپل) [20]: یک چندتایی مرتب از فضاهای مختلف است.
- دنباله [21]: یک دنباله از فضاهای مختلف است.

تابع پایان

در اتمام کار با محیط، این تابع فراخوانی می‌شود تا محیط بسته شود، و منابع آن آزاد شوند.

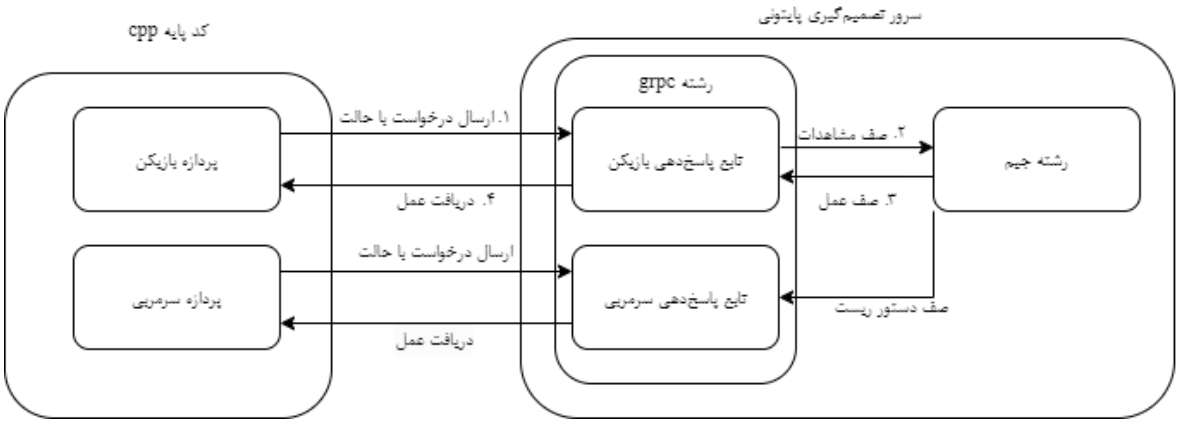
تابع ترسیم

با توجه به اینکه اکثر محیط‌های جیم، به کمک کتاب‌خانه‌هایی مانند PyGame ترسیم می‌شوند، این تابع برای ترسیم محیط در هر گام استفاده می‌شود. همچنین به عنوان ورودی این تابع می‌توان معین کرد که ترسیم به چه گونه‌ای انجام شود، چرا که ممکن است در حین آموزش برای تسریع عملیات، نیازی به ترسیم نباشد. از آن‌جا که در محیط ما، وظیفه ترسیم با نمایش‌گر خارجی انجام می‌شود، این تابع برای ما معنی ندارد.

۴-۴-۲ نحوه ادقام با فضای ربوکاپ

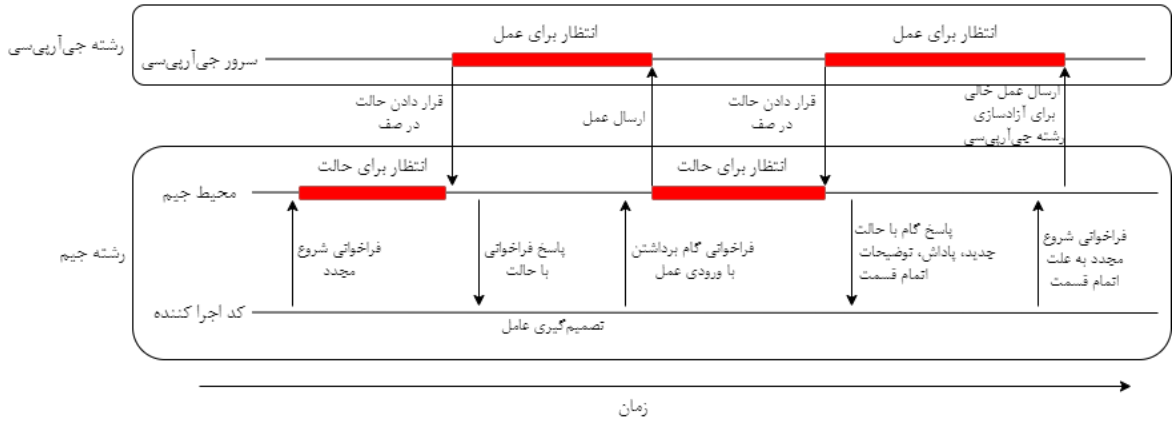
برای پیاده‌سازی محیط جیم در فضای ربوکاپ، از کد پایه gRPC استفاده می‌کنیم. فرآیند سرو کردن درخواست‌های این پروتکل را با کمک کتاب‌خانه‌های استاندارد پایتون به یک رشته [22] جدا منتقل می‌کنیم، که با کمک دو صف با رشته محیط جیم ارتباط دارد. در هنگام دریافت حالت از بخش پایه cpp، رشته سرویس‌دهنده حالت را داخل صف مشاهدات می‌گذارد، و منتظر می‌شود که رشته جیم، تصمیم عامل را بر اساس مشاهده ثبت‌شده در صف عمل‌ها قرار دهد. صف سومی نیز برای سرمربی وجود دارد، که در صورت نیاز به شروع مجدد محیط، از سمت جیم پر می‌شود. کد سرمربی تمرینی در هر لحظه از اجرایش چک می‌کند و در صورت خالی نبودن صف، دستورات لازم برای از نو کردن محیط را می‌فرستد. در صورت خالی بودن صف نیز عمل خالی به کد پایه پس می‌فرستد.

[20] Tuple
[21] Sequence
[22] Thread



شکل ۴-۷: نحوه کلی ارتباط کد پایه، تصمیم‌گیرنده جی‌آرپی‌سی و جیم

در شکل ۴-۸ به طور دقیق می‌توان ترتیب فراخوانی توابع، و نحوه استفاده از محیط جیم را مشاهده کرد. کد اجرا کننده [۲۳] دقیقا فرم مشابه سایر محیط‌های جیم را دارد و از این رابط به طور کامل طبعیت می‌کند. بخش‌های قرمز رنگ به معنی قفل بودن رشته‌ی اجرایی و در انتظار ماندن برای محتویات صف است.



شکل ۴-۸: ترتیب فراخوانی توابع برای اجرای جیم با سرور ربوکاپ

داخل شکل، یک قسمت که فقط یک گام تصمیم‌گیری دارد را می‌توان دید. در واقعیت مراحل ((گام برداشتن)) تا قبل از ((فراخوانی شروع مجدد)) به تعداد گام‌های قسمت، تکرار می‌شوند.

برای محیط پنالتی، در صورتی که توپ قابل ضربه‌زدن نباشد، حرکت صحیح همیشه قطع توپ با حداکثر سرعت برای ضربه مجدد است. بنابرین فقط حالت را در شرایطی به جیم ارسال می‌کنیم که توپ قابل ضربه‌زدن باشد.

[23] Driver Code

۴-۴-۳ فضای حالت و فضای عمل عامل

فضای حالت

فضای حالت انتخاب‌شده یک بسته ۹ بعدی است، که شامل ویژگی‌های زیر می‌باشد: موقعیت قطبی بازیکن نسبت به مرکز دروازه (اندازه و زاویه)، موقعیت دکارتی توپ (ایکس و ایگرگ)، موقعیت دکارتی دروازه‌بان (ایکس و ایگرگ)، زاویه نسبی حریف نسبت به توپ، موقعیت قطبی دروازه‌بان نسبت به بازیکن (اندازه و زاویه).

علت اهمیت عاملی مثل زاویه بدن دروازه‌بان، قابلیت حرکت سریع‌تر بازیکنان در راستای مستقیم است. می‌توان عواملی هم‌چون سرعت بازیکنان را نیز اثر داد، اما برای همگرایی بهتر و سریع‌تر از این عوامل صرف نظر شده.

فضای عمل

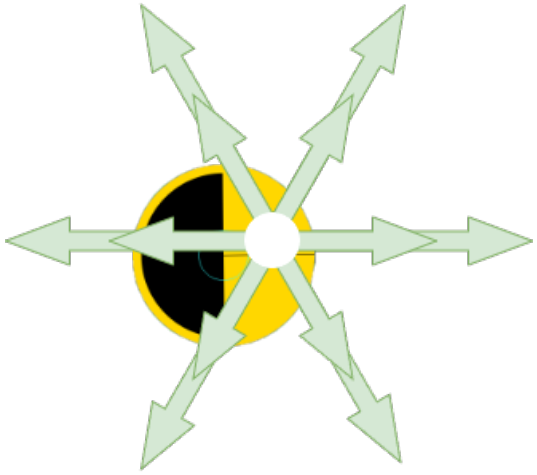
همان‌طور که گفته‌شد، عامل به صورت خودکار به دنبال توپ می‌رود و ما با کمک یادگیری تقویتی قرار است این عامل را به یادگیری ضربه‌زدن به دروازه برسانیم. برای این منظور، از رفتار سطح متوسط ضربه با سرعت دلخواه در راستای دلخواه (KickOneStep) استفاده می‌کنیم تا محاسبات مستقل از سرعت توپ قبلی و شتاب توپ شوند.

به این منظور، فضای عمل ما نیاز است سرعت و زاویه ضربه را تعیین کند. با توجه به اینکه برخی الگوریتم‌ها مانند DQN بهترین عمل را از بین اعمال ممکن انتخاب می‌کنند و نیاز به فضای عمل گسسته دارند، و سایر الگوریتم‌ها مانند DDPG نیاز به فضای عمل پیوسته دارند، محیط را در دو حالت گسسته و پیوسته پیاده‌سازی کردیم.

در فضای پیوسته، دو مقدار سرعت توپ و زاویه ضربه به عامل داده می‌شود. سرعت توپ بین صفر تا یک می‌باشد که نشان‌دهنده سرعت توپ نسبت به حداکثر سرعت ممکن است. زاویه ضربه بین منفی یک تا یک می‌باشد که نرمال‌شده زاویه مطلق ضربه بین ۱۸۰- تا ۱۸۰ درجه است.

در حالت گسسته نیز، خروجی ما سرعت توپ و زاویه‌است، اما به‌جای مقادیر پیوسته، زوایای ضربه ممکن را به ۱۲ قسمت مساوی تقسیم کرده‌ایم و سرعت توپ را به ۱۰ قسمت مساوی تقسیم کرده‌ایم. می‌توان این تقسیم‌بندی را به راحتی تغییر داد و تاثیرات آن روی سرعت یادگیری و نتیجه نهایی آن را بررسی کرد. در شکل ۴-۹ می‌توان یک نمونه از تقسیم‌بندی فضای عمل را مشاهده کرد. از آنجا که سرعت به ۲ درجه تقسیم شده‌است، حالت‌های ممکن آن معادل با نصف حداکثر سرعت توپ و حداکثر

سرعت توپ می‌باشد.



شکل ۴-۹: تقسیم‌بندی فضای عمل به ۶ زاویه و ۲ سرعت

۴-۴-۴ طراحی پاداش

همان‌طور که گفته شد، محیط جیم در حالت‌هایی که توپ قابل ضربه‌زدن است، یا توپ به بیرون رفته و یا توسط دروازه‌بان گرفته‌شده اجرا می‌شود. در این حالت‌ها باید بر اساس شرایط قبلی، عمل و شرایط جدید پاداشی محاسبه کنیم که به عامل کمک می‌کند تا راحت‌تر به هدف خود نزدیک شود.

پاداش‌های پایانی

همانطور که گفته‌شد، چهار حالت پایانی داریم:

۱. حالت گل زدن: با توجه به رسیدن به هدف، پاداش بزرگی به عامل می‌دهیم که این مقدار، ۱۵۰۰ فرض شده‌است.

۲. حالت بیرون رفتن توپ: از آن‌جا که تنها با ضربات بد این سناریو رخ می‌دهد، پاداش بسیار بزرگ منفی ۵۰۰- دارد.

۳. گرفتن توپ توسط دروازه‌بان: این حالت از آنجا که بهتر از بیرون رفتن توپ است، پاداش منفی کوچک‌تری دارد و مقدار ۲۰۰- دارد.

۴. حالت اتمام زمان: به این حالت، پاداش ۱۵۰- تخصیص داده‌شده، تا عامل در حالتی که هنوز به گل‌زدن نرسیده‌است، مستقیما توپ را به دروازه‌بان ندهد. در واقع با کوچک‌تر در نظر گرفتن این

پاداش، عامل را به کاوش بیشتر وامی‌داریم.

پاداش در حالت عادی

در حالت بین دو ضربه، پاداش را بر اساس سه فاکتور حساب می‌کنیم:

- می‌خواهیم عامل را به حرکت به سوی دروازه وادار‌یم، پس از تفاضل فاصله توپ با دروازه در حالت جدید و حالت قبلی به عنوان معیار اصلی استفاده می‌کنیم. در واقع از عکس این فاکتور استفاده می‌کنیم، چرا که کاهش فاصله ویژگی مثبتی است.
- نزدیک شدن زیادی به دروازه‌بان ویژگی خوبی نیست، پس از ضریبی از تفاضل فاصله توپ با دروازه‌بان در بین دو حالت استفاده می‌کنیم.
- پاداش ثابت ۱۰- که برای تشویق عامل برای سریع‌تر رسیدن به حالت گل می‌باشد. از این رو که عامل این پاداش را فقط در لحظات ضربه زدن می‌بیند، این فاکتور عامل را به زدن ضربات با قدرت بیشتر تشویق می‌کند.

۴-۵ پیاده‌سازی یادگیری تقویتی

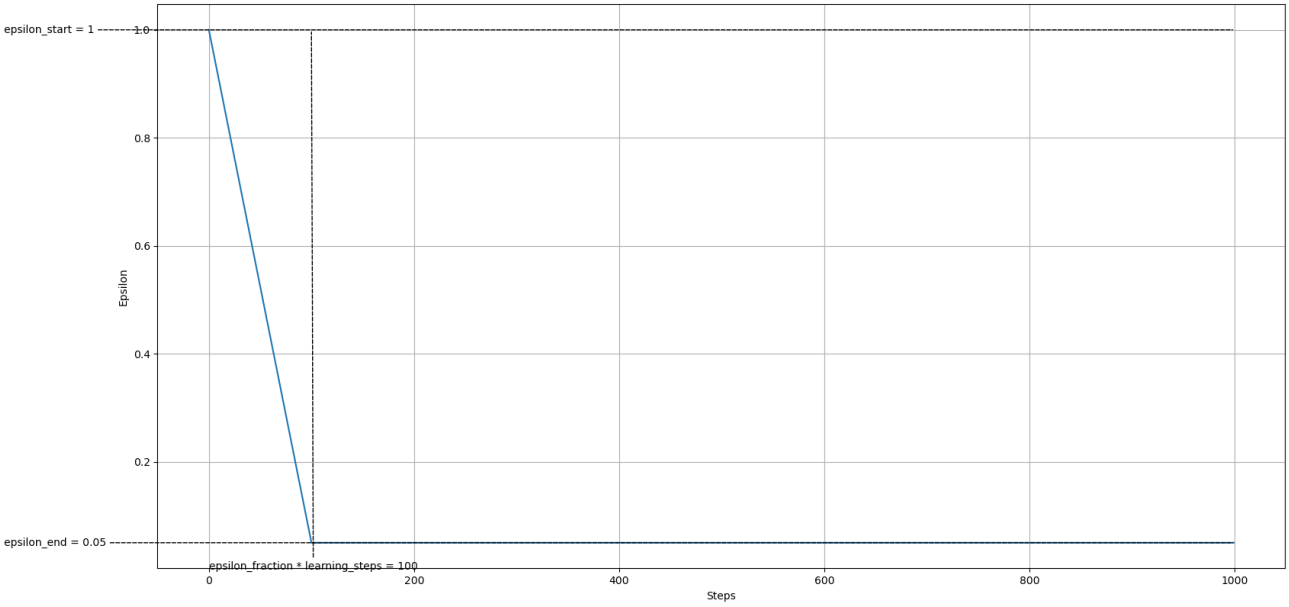
۴-۵-۱ پیاده‌سازی شبکه کیو عمیق

با کمک کتاب‌خانه PyTorch، یک شبکه عصبی با یک لایه پنهان و تابع فعال‌سازی ReLU پیاده‌سازی شد. این شبکه، ورودی به ابعاد فضای حالت محیط دارد و خروجی به اندازه فضای عمل است. تمامی ابرپارامترهای ما به عنوان ورودی تابع سازنده مدل داده می‌شوند.

سپس تابع یادگیری فراخوانی می‌شود، که ورودی آن مشخص می‌کند که چند گام از محیط برای آموزش استفاده شود. این تابع، با استفاده از الگوریتم DQN، به تعداد مراتب در محیط گام بر می‌دارد و تجارب را به بافر اضافه می‌کند. در گام‌هایی که شماره آن ضریبی از ابرپارامتر update_freq است، شبکه را به کمک گزینش ریزدسته‌ای از تجربیات و الگوریتم گرادیان، به‌روز می‌کند. همچنین در گام‌هایی که شماره آن ضریبی از ابرپارامتر target_update_freq است، شبکه هدف را به‌روز می‌کند.

همانطور که در فصل‌های قبل اشاره شد، نرخ کاوش تصادفی به مرور زمان کاهش می‌یابد. در ابرپارامترها، نرخ اولیه، نرخ نهایی، و درصد گام‌هایی که این کاهش نرخ در آن رخ می‌دهد با پارامترهای

epsilon_start، epsilon_end و epsilon_fraction مشخص می‌شود. پارامتر epsilon_fraction در واقع نشان می‌دهد که در چند درصد اول گام‌های آموزشی، نرخ کاوش تصادفی به نرخ نهایی می‌رسد. در شکل ۴-۱۰ می‌توان نمودار کاهش نرخ کاوش تصادفی را مشاهده کرد. در این مثال آموزش برای ۱۰۰۰ گام رخ داده، نرخ اولیه ۱ و نرخ نهایی ۵.۰ انتخاب شده‌است، و کسر اپسیلون [۲۴] ۱.۰ است.



شکل ۴-۱۰: کاهش نرخ کاوش تصادفی در طی آموزش

۴-۵-۲ پیاده‌سازی سایر الگوریتم‌ها به کمک کتابخانه

یکی از مهم‌ترین مزایای پیاده‌سازی رابط استاندارد جیم، ایجاد امکان استفاده از پیاده‌سازی‌ها و کتاب‌خانه‌های ازپیش‌موجودی اند که وجود دارند. با توجه به اینکه اکثر الگوریتم‌های یادگیری تقویتی در کتاب‌خانه‌هایی مانند Stable Baselines و OpenAI Baselines پیاده‌سازی شده‌اند، می‌توان از این کتاب‌خانه‌ها برای آموزش عامل‌ها استفاده کرد. در این پروژه، از کتاب‌خانه Stable Baselines برای آموزش عامل‌ها استفاده شده‌است. کافی‌ست ابتدا یک محیط جدید از جیم تعریف کنیم، مدلی از الگوریتم دلخواه بسازیم، و سپس با استفاده از تابع learn مدل را به تعداد گام‌های دلخواه آموزش دهیم.

```
model = DDPG('MlpPolicy', gym_env)
model = model.learn(1_000_000, progress_bar=True)
model.save("DDPG_model_1M")
```

شکل ۴-۱۱: استفاده از کتاب‌خانه Stable Baselines 3

[24] Epsilon fraction

طبق مستندات این کتاب‌خانه، می‌توان به مدل توابعی را به عنوان ورودی داد تا در حین آموزش به صورت مکرر و پس از تعداد گام دلخواه به صورت تناوبی صدا زده‌شوند. از این توابع می‌توان برای ارزیابی مدل به صورت تناوبی، و ذخیره‌سازی در حین آموزش استفاده کرد که برای مقایسات فصل بعدی بسیار کاربردی است.

۴-۶ جمع‌بندی

در این فصل، مروری بر راه‌های مختلفی که می‌توان از آن‌ها برای پیاده‌سازی یادگیری تقویتی استفاده کرد، داشتیم. از جمله این روش‌ها، استفاده از HFO و Pyrus بودند. سپس با معرفی پلتفرم جیم، نحوه پیاده‌سازی محیط‌های جدید و تعریف فضای حالت و عمل را مورد بررسی قرار دادیم.

در ادامه، نحوه ادغام این محیط‌ها با فضای ربوکاپ و پیاده‌سازی یک محیط جدید به نام پنالتی را مورد بررسی قرار دادیم. سپس نحوه پیاده‌سازی یک عامل یادگیری تقویتی با استفاده از شبکه کیو عمیق و کتاب‌خانه Stable Baselines را بررسی کردیم. در فصل بعدی، نتایج این پیاده‌سازی‌ها را مورد بررسی قرار می‌دهیم.

فصل پنجم

مقایسه، آزمایش‌ها، و نتایج

در این فصل به بررسی نتایج حاصل از پیاده‌سازی الگوریتم‌ها و مقایسه آن‌ها می‌پردازیم.

۵-۱ فرآیند آزمایش

برای آزمایش، ابتدا الگوریتم‌های معرفی شده در فصل‌های قبل را با یک میلیون گام علیه دروازه‌بان کد پایه Agent آموزش می‌دهیم، و نمودارهای فرآیند یادگیری را بررسی می‌کنیم. در هر ۵۰۰۰ گام از آموزش مدل به صورت تناوبی ذخیره می‌شود، و در نهایت بهترین مدل‌ها را انتخاب می‌کنیم. لازم به ذکر است که منظور از گام، گام‌های یادگیری تقویتی است، که هرکدام متناظر با یک ضربه به توپ اند و منظور گام‌های شبیه‌ساز فوتبال نیست.

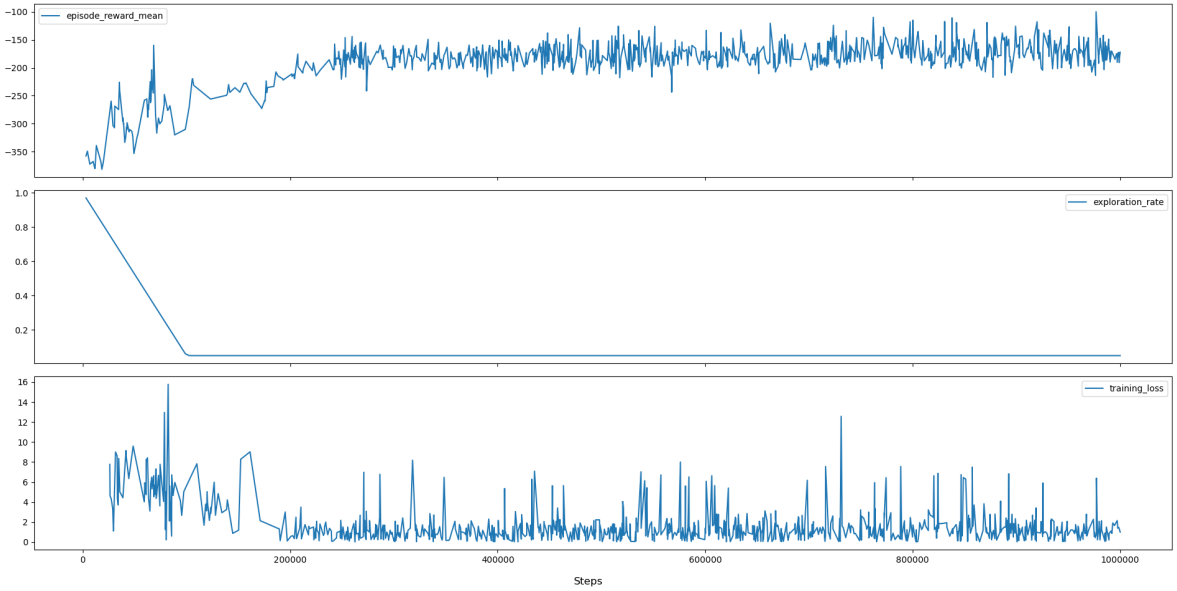
در نهایت مدل انتخاب شده را برای ۱۰۰ ضربه پنالتی علیه کد پایه قرار می‌دهیم تا با سیاست حریصانه و بدون تصمیم‌گیری تصادفی سنجیده شود، و درصد وقوع هر یک از حالت‌های پایانی بازی را بررسی می‌کنیم. در نهایت راهکارهای متفاوتی را برای بهبود نتایج یادگیری امتحان می‌کنیم.

تمامی آزمایش‌ها روی یک سیستم با پردازنده‌ی AMD Ryzen 7 6800H، حافظه ۳۲ گیگابایت، و کارت گرافیک NVIDIA RTX 3070Ti انجام شده است، همراه با کتاب‌خانه‌های کودا [1] CUDA 12.4، cudnn 8.9.2، و cublas 12.1.3 انجام شده‌است.

۵-۲ ارزیابی الگوریتم‌ها

در ابتدا الگوریتم DQN را برای یادگیری از صفر امتحان می‌کنیم. بعد از یک میلیون گام آموزش که حدود ۱۲ ساعت طول کشید، عامل موفق به کشف تکنیک گل زدن نشد. فرآیند آموزش ۵ بار و با ابرپارامترهای متفاوت آزمایش شد، اما در هیچ یک از اجراها عامل موفق به یادگیری نشد. یکی از دلایل ممکن این موضوع، می‌توانست تفاوت زیاد بین پاداش‌های موفق و ناموفق باشد، چرا که این موضوع ابعاد گرادیان را بزرگ کرده و باعث عدم ثبات می‌شود. به همین منظور یک ضریب مقیاس‌گر برای پاداش‌ها اضافه شد، که باز هم موثر در موفقیت نبود.

[1] CUDA

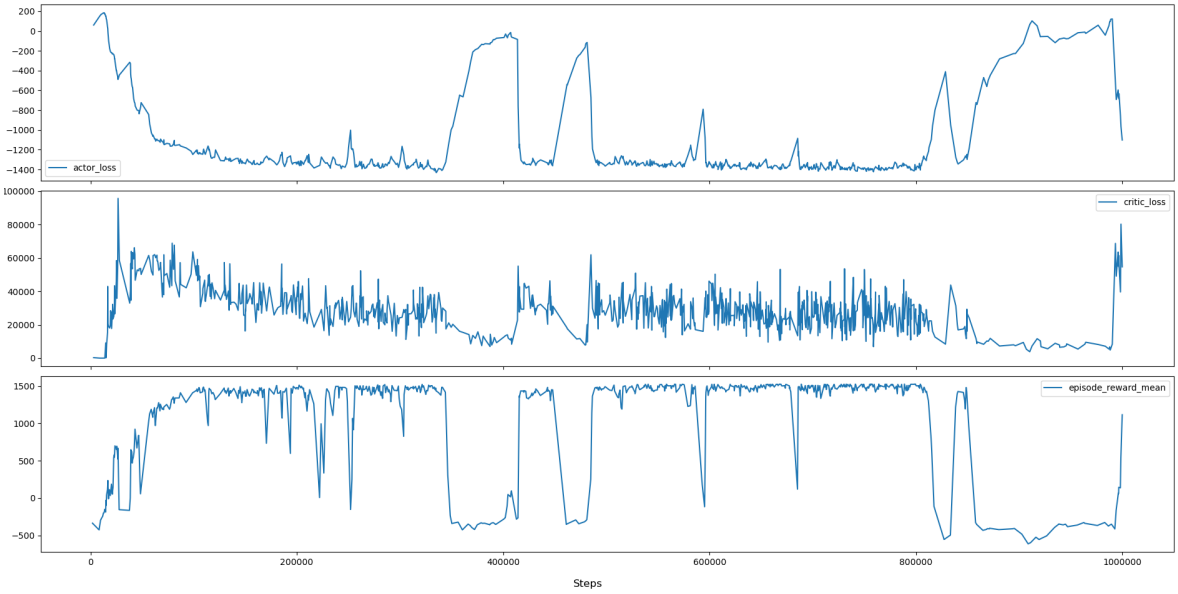


شکل ۵-۱: نمودار سه پارامتر متفاوت برای یک اجرای الگوریتم DQN

در شکل ۵-۱ می‌توان به ترتیب میانگین پاداش عامل، نرخ کاوش، و خطای شبکه عصبی را در طی زمان آموزش دید. محور افقی این نمودار گام‌های آموزش است، که در اینجا ۱ میلیون گام است. مقیاس پاداش‌ها برای این اجرا، ۲.۰ بوده‌است و برای رسم نمودار، به مقیاس ۱ تبدیل شده‌است. همانطور که می‌توان مشاهده کرد، نمودار خطای شبکه از گام ۵۰۰۰۰ آغاز شده. این به این علت است که آموزش شبکه این گام آغاز می‌شود، تا پیش از آن بافر با حالت‌های متفاوت پر شود.

همانطور که قابل مشاهده است، پاداش‌ها نسبت به حالت آغازین افزایش می‌یابند اما عامل موفق به کشف حالت گل زدن نشده‌است. میزان خطای شبکه عصبی نیز در طول زمان کاهش یافته و نزدیک به ۱ شده، اما این هم نشان‌دهنده‌ی یادگیری موفق نیست، بلکه شبکه همواره (به درستی) میزان ارزش حالت شکست را خروجی می‌دهد.

همین فرآیند را با الگوریتم DDPG تکرار می‌کنیم. می‌توان مشاهده کرد که عامل در زیر ۱۰۰ هزار گام موفق به کشف تکنیک گل زدن شده‌است. همچنین همانطور که در فصل‌های قبلی ذکر شد، به علت محدود بودن اندازه بافر، عامل ممکن است یادگیری خود را فراموش کند.



شکل ۵-۲: نمودار سه پارامتر متفاوت برای یک اجرای الگوریتم DDPG

در شکل ۵-۲ می‌توان به ترتیب خطای شبکه بازیگر، خطای شبکه منتقد، و میانگین پاداش عامل را در طی زمان آموزش دید. همانطور که می‌توان مشاهده کرد، در حین موفقیت عامل خطای شبکه‌ها و به ویژه خطای بازیگر کاهش یافته‌است.

جدول ۵-۱: بهترین نتایج الگوریتم‌ها مقابل کد پایه Agent2D برای صد پنالتی.

خروجی	DQN	DDPG
گل زدن	۲	۹۹
بیرون رفتن توپ	۱۸	۰
گرفتن توسط دروازه‌بان	۷۴	۱
اتمام زمان	۶	۰

با توجه به اینکه الگوریتم DDPG موفق بود، می‌توان حدس‌هایی راجع به عدم موفقیت DQN زد. یکی از دلایل، این است که به علت گسسته بودن اعمال، تعداد گام‌های لازم برای رسیدن به گل بیشتر است، و از آنجا که عامل باید به صورت تصادفی گل زدن را کشف کند، عامل به تعداد دفعات کافی به این حالت نمی‌رسد. همچنین همانطور که در شکل ۲-۴ می‌توان دید، شبکه در الگوریتم DQN به ازای هر عمل، یک خروجی دارد و به همین علت، فضای عمل بزرگ یادگیری را به مراتب دشوار می‌کند. از این رو می‌توان سه راه را برای بهبود یادگیری این الگوریتم پیشنهاد داد:

۱. بهبود گسسته‌سازی عمل‌ها.

۲. استفاده از پاداش های بهتر برای حالت‌های غیرنهایی که عامل را به سمت حالت‌های مفیدتر هدایت کند.

۳. تغییر فضای عمل و جداسازی اعمال به دریبل زدن و شوت زدن.

۵-۳ بهبود تابع پاداش

برای بهبود تابع پاداش، به روش گل‌زنی الگوریتم DDPG نگاه می‌کنیم. به نظر می‌آید که نزدیک شدن دروازه‌بان به توپ عامل منفیی نیست، چرا که این حالت ممکن است به عامل کمک کند تا با یک ضربه به توپ، زاویه دروازه را باز کند. همچنین در صورت نزدیکی بیش از حد دروازه‌بان به توپ در حین ضربه، عامل می‌تواند از قابلیت گسسته بودن زمان شبیه‌ساز استفاده کند و با انجام قوی‌ترین ضربه ممکن، توپ را از روی دروازه‌بان بگذراند. بنابرین برای گام‌های بعدی، عامل فاصله با دروازه‌بان را از محاسبات پاداش حذف می‌کنیم.

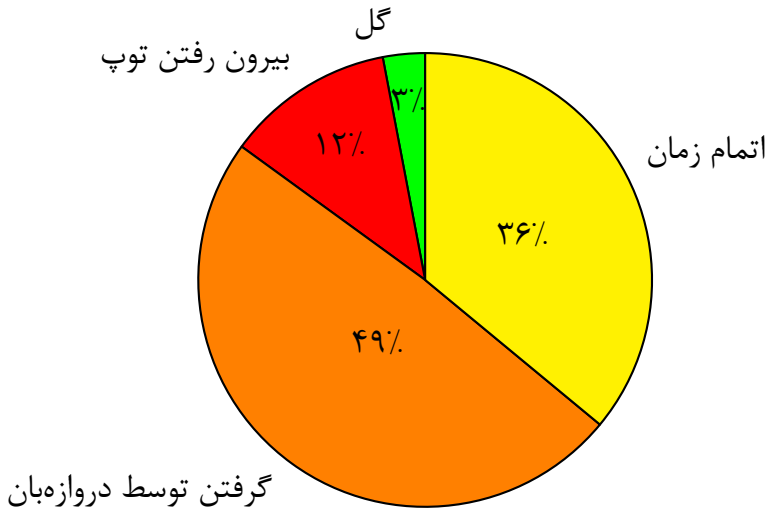
۵-۴ جداسازی عمل شوت

عامل در ابتدای یادگیری، هنوز به درک اینکه در چه حالتی می‌تواند شوت منجر به گل داشته باشد نرسیده است. از آنجا که این عمل با عمل رسیدن به موقعیت شوت‌زنی متفاوت است، عامل ممکن است دیر به دیر به حالت‌هایی که شوت زدن در آن ممکن است برسد. بنابرین در حین یادگیری نسبت به ضربه با سرعت‌های بالا بدبین می‌شود.

برای حل این مشکل، فضای عمل را به گونه‌ای اصلاح می‌کنیم که شوت زدن جدا باشد.

۵-۴-۱ شوت با رفتار سطح بالای کد پایه

یک عمل خاص مدنظر می‌گیریم که در صورت انتخاب شدن توسط عامل، شوت کد پایه Agent2D انجام می‌شود. این کد در صورت وجود شوت ممکن، آن را انجام می‌دهد و در غیر این صورت، هیچ عملی انجام نمی‌شود. در صورتی که عامل این عمل را انتخاب کند ولی به گل نرسد، پاداش منفیی به آن داده می‌شود، تا عامل استفاده صحیح از این ابزار را یاد بگیرد.

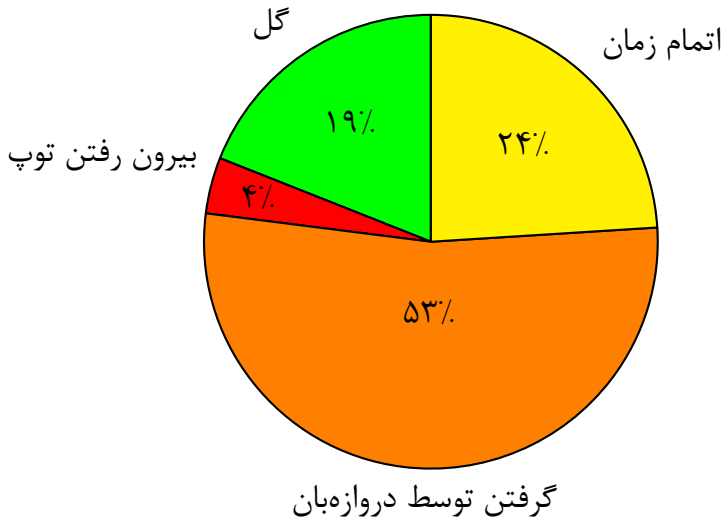


شکل ۵-۳: نتیجه ۱۰۰ ضربه پنالتی با شوت سطح بالا مقابل کد پایه Agent2D برای عامل یادگیری‌شده

طبق تجربه پیش از این پروژه، این رفتار سطح بالا بسیار محافظ‌کارانه تصمیم می‌گیرد. از این رو شاید کمک چندانی به عامل برای یادگیری نکند. همچنین از آنجا که استفاده از این قابلیت بسیار وابسته به شبیه‌سازی ضربه به توپ و بررسی امکان قطع توپ توسط دروازه‌بان است، خلاف ماهیت انجام این پروژه است.

۵-۴-۲ عمل شوت به نقاط ثابت دروازه

فرض کنید تعدادی نقطه ثابت دروازه را انتخاب کنیم. هر یک از این نقاط متناظر با یک عمل شوت است. در این عمل، به توپ با حداکثر سرعت ممکن در راستای این عمل ضربه می‌زنیم. برای انتخاب نقاط، دو تیرک دروازه را به عنوان نقاط ثابت در نظر گرفته، و فضای بین این نقطه را به $n-1$ نقطه تقسیم می‌کنیم.



شکل ۵-۴: نتیجه تست ۱۰۰ پنالتی مقابل کد پایه Agent2D با ۵ نقطه شوت.

۵-۵ تغییر گسسته‌سازی

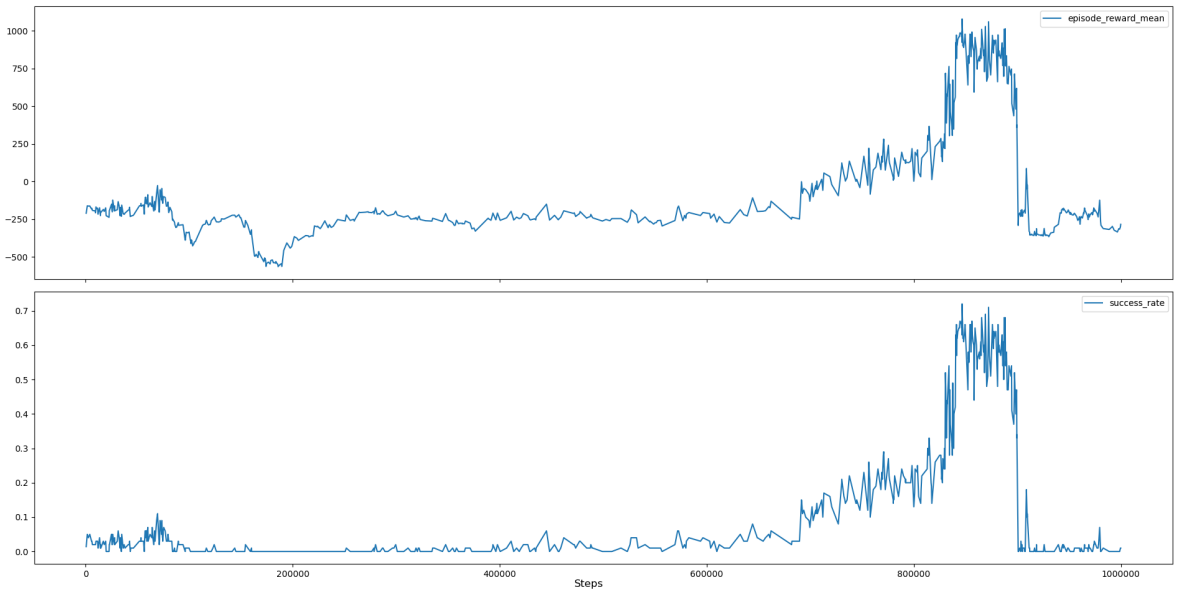
حال که عامل دارای عمل مستقیم شوت است، می‌توانیم گسسته‌سازی قدرت را به گونه‌ای تغییر دهیم که سرعت‌های بالا را نداشته باشد. برای این منظور ضربه به توپ را به سه سرعت تقسیم می‌کنیم: ۱۵ درصد حداکثر سرعت توپ، ۲۵ درصد حداکثر سرعت توپ، و ۵۰ درصد حداکثر سرعت توپ. در واقع اعمال ممکن را به شوت به نقطه یا دریبل با سرعت و زاویه معین تقسیم می‌کنیم.

همچنین به صورت شهودی می‌توان برداشت کرد که تقسیم زوایای جلوی بازیکن به شدت مهم‌تر از زوایای پشت بازیکن است، چرا که به ندرت نیاز می‌شود بازیکن با ضربه به عقب پیشرفت کند. زوایای جلوی بازیکن را به ۹ قسمت تقسیم می‌کنیم، و دو راستا برای دریبل به عقب (بالا عقب و پایین عقب) اضافه می‌کنیم.

با ترکیب این دو تکنیک، فضای حالت از ۱۲۰ عمل، به ۳۸ عمل کاهش می‌یابد. این کاهش از دو راستا به یادگیری ما کمک می‌کند:

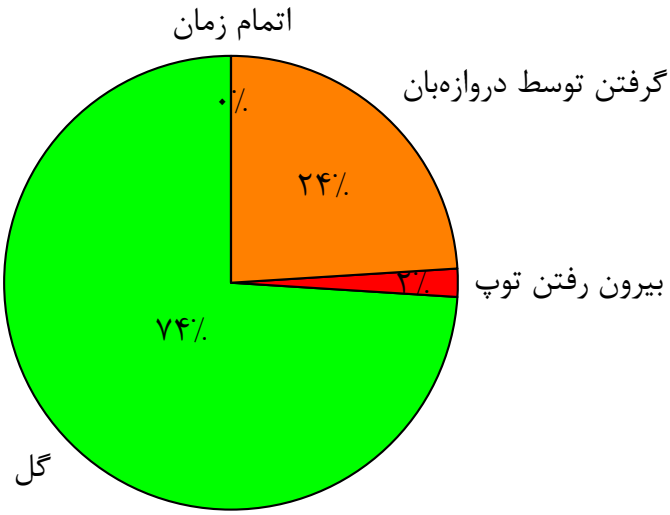
- کاهش ابعاد گرادیان و افزایش سرعت یادگیری
- افزایش احتمال انجام عمل مفید به کمک کاوش، به علت حذف رفتارهایی که به احتمال زیاد مفید نیستند

پس از یک میلیون گام آموزش، نمودار پاداش‌ها و درصد موفقیت در حین آموزش را مشاهده می‌کنیم.



شکل ۵-۵: نمودار میانگین نرخ گل‌زنی و میانگین پاداش DQN با تغییرات گسسته‌سازی

همانطور که در تصویر مشاهده می‌کنید، با این تغییرات عامل موفق به کشف روش گل‌زنی و موفقیت ۷۲ درصد شده‌است. همچنین می‌توان دید که نمودار پاداش و نرخ گل‌زنی هم‌رفتار و تقریبا هم‌شکل هستند.



شکل ۵-۶: نتیجه تست ۱۰۰ پنالتی مقابل کد پایه Agent2D با ۵ نقطه شوت.

۵-۶ بررسی تعمیم پذیری

در این بخش، با تست علیه دروازه‌بان تیم‌هایی که مقابل آن‌ها یادگیری رخ نداده، می‌بینیم که آیا این یادگیری تعمیم‌پذیر بوده یا خیر. با توجه به اینکه پس از مسابقات، فایل‌های اجرایی تیم‌ها در اختیار

سایر شرکت‌کنندگان قرار می‌گیرد، می‌توانیم این تست را انجام دهیم. از ۳ تیم برای این منظور استفاده می‌کنیم: تیم Helios2023 که قهرمان پیاپی چندین دوره است، تیم کد پایه Agent2D که مبنای بسیاری از سایر تیم‌ها است، تیم YuShan2023 که از تیم‌های موفق دیگر است و استراتژی متفاوتی نسبت به دو تیم قبلی دارد.

جدول ۵-۲: نتایج تست الگوریتم DDPG علیه تیم‌های مختلف

خروجی	Agent2D	Helios2023	YuShan2023
گل زدن	۹۹	۴۰	۲۷
بیرون رفتن توپ	۰	۴	۱۱
گرفتن توسط دروازه‌بان	۱	۴۰	۵۷
اتمام زمان	۰	۱۶	۴

به صورتی شهودی و دیدن نتایج تست، می‌توان دید که نتایج تا حدودی قابل تعمیم است، اما شوت زدن عامل بسیار وابسته به مختصات دروازه‌بان است و از این رو، شوت‌های به بیرون و یا گرفته شده توسط دروازه‌بان افزایش یافته‌اند. همچنین دروازه‌بان سایر تیم‌ها مانند Helios2023، در شرایطی که نتوانند به توپ برسند از دستور تکل استفاده می‌کنند، که برد بیشتری نسبت به گرفتن توپ دارد ولی به صورت احتمالاتی است. در صورت تکل موفق حالت گرفتن توسط دروازه‌بان ثبت می‌شود.

جدول ۵-۳: نتایج تست الگوریتم DQN بهبود یافته مقابل تیم‌های مختلف

خروجی	Agent2D	Helios2023	YuShan2023
گل زدن	۷۱	۰	۱۰
بیرون رفتن توپ	۰	۰	۱۵
گرفتن توسط دروازه‌بان	۲۹	۱۰۰	۷۰
اتمام زمان	۰	۰	۵

با توجه به گسسته‌سازی، هر گونه تغییر در موقعیت دروازه‌بان عمل بعدی عامل را به شدت تحت تاثیر قرار می‌دهد و عامل به شدت با حالت‌های از پیش دیده فاصله می‌گیرد. از این رو همانطور که پیش‌بینی می‌شد، تعمیم‌پذیری این الگوریتم به تیم‌های دیگر بسیار پایین است.

در نهایت یادگیری الگوریتم DDPG را تکرار می‌کنیم، با این تفاوت که آموزش را از ابتدا و مقابل تیم‌های مختلف انجام می‌دهیم. این آزمایش از این رو است که ببینیم در صورتی که آموزش روی تیم‌های قوی‌تر انجام شود، آیا عامل موفق به یادگیری تکنیک‌های مفیدتری می‌شود یا خیر.

فرآیند آموزش تا زمانی ادامه می‌یابد که عامل به تکنیک گل‌زنی مقابل تیم آموزشی برسد، و تا یک میلیون گام ادامه نخواهد یافت. برای یادگیری مقابل تیم Helios2023 فقط ۱۵۰ هزار گام کافی بود تا

نرخ موفقیت به بالا ۹۰ درصد برسد که حدود یک ساعت زمان برد. یادگیری مقابل تیم YuShan2023 حدود ۷۰ هزار گام طول کشید که ۲۰ دقیقه زمان برد.

جدول ۵-۴: نتایج تست الگوریتم DDPG علیه تیم‌های مختلف با آموزش مقابل تیم Helios2023

خروجی	Agent2D	Helios2023	YuShan2023
گل زدن	۴۷	۹۰	۲۳
بیرون رفتن توپ	۱۶	۰	۵
گرفتن توسط دروازه‌بان	۳۷	۷	۶۵
اتمام زمان	۰	۳	۷

جدول ۵-۵: نتایج تست الگوریتم DDPG علیه تیم‌های مختلف با آموزش مقابل تیم YuShan2023

خروجی	Agent2D	Helios2023	YuShan2023
گل زدن	۰	۰	۹۹
بیرون رفتن توپ	۰	۰	۰
گرفتن توسط دروازه‌بان	۸۳	۱۰۰	۱
اتمام زمان	۱۷	۰	۰

به نظر می‌رسد که استراتژی متفاوت YuShan2023 باعث می‌شود عامل‌هایی که روی سایر تیم‌ها آموزش دیده‌اند گیج شوند و نتوانند به موفقیت برسند. اما آموزش روی YuShan2023 که به دور زدن دروازه‌بان در آن آسان‌تر است، باعث می‌شود که عامل به موفقیت جلوی سایر تیم‌ها نرسد، چرا که تاکتیک موفق جلوی YuShan2023 منجر به گرفتن توسط دروازه‌بان می‌شود.

از سوی دیگر، با توجه به این که استراتژی Helios2023 مشابه ولی قوی‌تر نسبت به Agent2D است، آموزش مقابل آن باعث می‌شود که عامل به موفقیت نسبی نزدیک شود.

لازم به ذکر است که حتی اگر عامل در دو دور آموزش مقابل تیم آموزشی به نتایج یکسان برسد، نمی‌توان انتظار داشت که نتایج تست مقابل سایر تیم‌ها یکسان باشد. به این منظور که اگر آموزش را دو بار از صفر آغاز کنیم و تا صد درصد موفقیت پیش ببریم، ممکن است نتایج تست مقابل تیم‌های دیگر بین دو حالت نزدیک نباشد. از این رو این آزمایش معیار خیلی دقیقی برای بررسی کمی تعمیم‌پذیری نیست.

در آینده می‌توان راه‌های تولید مدل‌های تعمیم‌پذیرتر را بیشتر بررسی کرد. یکی از این راه‌ها استفاده از حریف‌های متفاوت در حین یادگیری، و روش‌های یادگیری تقویتی مداوم[۲] است.

[2]Continual Reinforcement Learning

۵-۷ جمع‌بندی

در این فصل، ما الگوریتم‌های یادگیری تقویتی DQN و DDPG را برای یادگیری تکنیک گل‌زنی در فوتبال تک به تک مورد بررسی قرار دادیم. همچنین چند راه حل برای بهبود یادگیری این الگوریتم‌ها ارائه دادیم. در نهایت، تعمیم‌پذیری این الگوریتم‌ها را بررسی کردیم و دیدیم که طبق توقع، عامل در مقابل حریف‌هایی که از پیش ندیده، قدرت یادگیری کمتری دارد.

فصل ششم

جمع‌بندی، نتیجه‌گیری و پیشنهادات

در این فصل مروری بر سیر مطالب عنوان شده خواهیم‌داشت. سپس مروری بر نتایج و دستاوردهای این پروژه خواهیم داشت و در نهایت پیشنهاداتی برای ادامه‌ی کار در این موضوع ارائه خواهیم کرد.

۶-۱ جمع‌بندی و نتیجه‌گیری

در این پروژه قصد داشتیم تا با استفاده از روش‌های یادگیری ماشینی، یک محیط مجازی برای آموزش و تقویت بازیکنان فوتبال ایجاد کنیم. به این منظور ابتدا با مفاهیم اساسی یادگیری تقویتی آشنا شدیم، با مهم‌ترین روش‌های پیاده‌سازی کلاسیک آن آشنا شدیم و در نهایت با روش‌های پیشرفته‌تری همچون یادگیری تقویتی عمیق و برخی الگوریتم‌های آن آشنا شدیم. در دو الگوریتم یادگیری کیو عمیق و یادگیری گرادیان سیاست قطعی عمیق به طور دقیق با فرآیند یادگیری و عوامل موثر در آن آشنا شدیم.

در ادامه محیط شبیه‌ساز دو بعدی فوتبال را معرفی کردیم و برخی از ویژگی‌های آن را توضیح دادیم. رفتارهای مجاز بازیکنان را بررسی کردیم، و کمی با تاریخچه ربوکاپ آشنا شدیم. سپس حالت پنالتی را به عنوان یک محیط مناسب برای یادگیری تقویتی معرفی کردیم. در بخش بعدی تلاش‌های متفاوتی که برای پیاده‌سازی یادگیری تقویتی در شبیه‌سازی فوتبال انجام شده‌بود را بررسی کردیم. در نهایت رابط استاندارد جیم را معرفی کردیم و یک پیاده‌سازی برای اتصال این رابط با کد پایه شبیه‌ساز به کمک جی‌آرپی‌سی انجام دادیم.

در نهایت با استفاده از الگوریتم‌های یادگیری تقویتی عمیق، یک مدل برای آموزش سمت مهاجم در تک به تک با دروازه‌بان ایجاد کردیم و الگوریتم‌های متفاوت را مقایسه کرده، و پیشنهاداتی برای بهبود عملکرد آن‌ها ارائه کردیم.

۶-۲ کارهای آتی

در آینده می‌توان از زیرساخت ایجاد شده در سه راستا استفاده کرد: حل سایر مسائل در لیگ شبیه‌ساز دو بعدی، استفاده از محیط برای ارزیابی الگوریتم‌ها و روش‌های یادگیری، و بررسی تاثیر ویژگی‌های مختلف بر عملکرد مدل‌های یادگیری تقویتی. برخی از پیشنهاداتی که در این زمینه‌ها می‌توان ارائه کرد عبارتند از:

۱. بررسی یادگیری تقویتی در حالت‌های چند عامله: یکی از بزرگترین چالش‌های یادگیری تقویتی، یادگیری در حالت‌های چند عامله‌ای است که نیاز به هماهنگی بین عامل‌ها دارد. این زیرساخت

و محیط فوتبال می‌تواند محل مناسبی برای توسعه ایده‌های جدید در این راستا باشد.

یک نمونه از این مسائل انتخاب حریف و یارگیری به صورت مجزا برای هر بازیکن است. از چالش‌های این مسئله می‌توان به مشکل همگام بودن عوامل در تصمیم‌گیری، بزرگی فضای حالت، و تفاوت میان اطلاعات عامل‌ها به دلیل دید ناقص اشاره کرد.

۲. بررسی تغییر روش نمایش ویژگی‌ها و تاثیر آن بر عملکرد مدل‌های یادگیری تقویتی: یکی از مهم‌ترین مسائل در یادگیری تقویتی، انتخاب نمایش مناسب برای ویژگی‌ها است. انتخاب نمایش مناسب می‌تواند تاثیر بزرگی بر عملکرد مدل‌ها داشته باشد. به طور مثال، موقعیت بازیکن را می‌توان به صورت دکارتی یا قطبی از مبدا مرکز زمین، مرکز دروازه، دروازه‌بان یا توپ نسبت به بازیکن دیگر نمایش داد. بررسی تاثیر و اهمیت نحوه نمایش ویژگی‌ها بر عملکرد مدل‌ها می‌تواند موضوعی جالب برای تحقیقات آینده باشد.

همچنین افزودن برخی ویژگی‌ها مانند سرعت بازیکنان یا انرژی بازیکن می‌تواند تاثیر مهمی روی عملکرد تیم داشته‌باشد، اگرچه ممکن است به طلسم ابعاد[1] بر بخوریم.

۳. استفاده از روش‌هایی همچون آموزش انتقالی[2] برای افزایش سرعت آموزش: همانطور که در نمودارهای یادگیری مانند شکل ۵-۵ می‌توان دید، عامل زمان زیادی را در ابتدای آموزش صرف جستجوی بخش‌های بیهوده فضای حالت می‌کند. استفاده از یادگیری انتقالی می‌تواند کمک کند از عامل‌های از پیش نوشته‌شده استفاده کنیم تا روند آموزش را سریع‌تر کنیم.

۴. استفاده از روش‌های یادگیری متقابل برای آموزش دروازه‌بان و مهاجم به صورت همزمان: می‌توان از روش‌های یادگیری متقابل برای آموزش دو بازیکن به صورت همزمان استفاده کرد. این روش‌ها می‌توانند بهبود عملکرد هر دو بازیکن را به صورت همزمان و بهینه کنند. به کمک این روش‌ها می‌توان همزمان دروازه‌بان و مهاجمی ساخت که هر دو از یادگیری دیگری بهره‌مند شوند و از بهترین تیم‌ها هم قوی‌ترند.

[1] Curse of Dimensionality

[2] Transfer Learning

منابع و مراجع