به نام خدا

کتابچه

مقدمه‌ای بر یادگیری تقویتی

میثم میرزائی

علی ژاله کریمی

شهریور ۱۴۰۳

فهرست مطالب

۱- مقدمه

این ایده که ما از طریق تعامل با محیط خود یاد می‌گیریم احتمالاً اولین چیزی است که وقتی به ماهیت یادگیری فکر می‌کنیم به ذهنمان خطور می‌کند. وقتی نوزادی بازی می‌کند، دستانش را تکان می‌دهد یا به اطراف نگاه می‌کند، معلم صریحی ندارد، اما ارتباط حسی - حرکتی مستقیمی با محیط خود دارد. اعمال این ارتباط اطلاعات فراوانی در مورد علت و نتیجه، در مورد پیامدهای اعمال و اقداماتی که برای دستیابی به اهداف باید انجام داد، تولید می‌کند. در طول زندگی ما، چنین تعاملاتی بدون شک منبع اصلی دانش در مورد محیط و خودمان است. چه در حال یادگیری رانندگی با ماشین باشیم و چه در حال انجام یک مکالمه، ما کاملاً از واکنش محیط خود به کاری که انجام می‌دهیم آگاه هستیم و به دنبال تأثیرگذاری بر اتفاقات از طریق رفتار خود هستیم. یادگیری از تعامل یک ایده اساسی است که زیربنای تقریباً تمام نظریه‌های یادگیری و هوش است (Richard S. Sutton and Andrew G. Barto).

این کتابچه حاصل یک کار تحقیقاتی است که تمام تلاش در آن ارائه یک دانش حداقلی نسبت به مباحث یادگیری تقویتی و پیشرفت‌ها در این حوزه است. در این کتابچه سعی شده است حجم مطالب ارائه شده حد الامکان به‌صورت مختصر ارائه شود و تا حد ممکن مباحث اصلی یادگیری تقویتی مورد بررسی قرار گیرد. این کتابچه دریچه ای برای ورود به دنیای یادگیری تقویتی است و طبیعتا تمام نیاز های خواننده برای تسلط بر این حوزه را تامین نمی کند. مخاطب این کتاب دانشجویان و علاقه مندان به حوزه هوش مصنوعی و یادگیری تقویتی هستند که می خواهند یک دانش اولیه از مباحث یادگیری تقویتی به دست آورند.

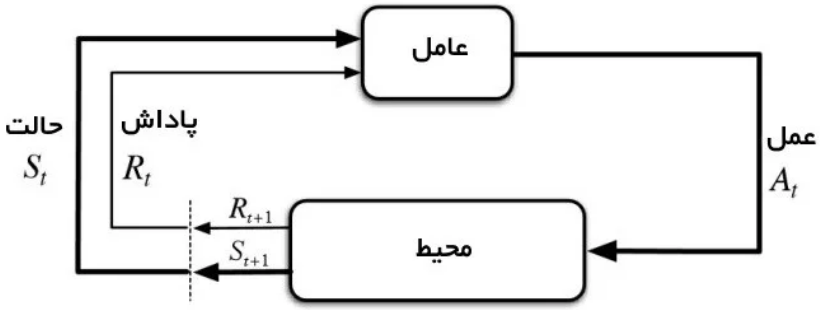
در بخش اول این کتابچه ابتدا مفاهیم یادگیری تقویتی پرداخته می‌شود سپس الگوریتم‌های پایه یادگیری تقویتی معرفی می‌شوند. در ادامه مباحث پیشرفته در یادگیری تقویتی بررسی می‌شود و در انتها منابع مورداستفاده معرفی می‌شوند و چند نمونه از پیاده‌سازی‌های الگوریتم‌های یادگیری تقویتی ارائه می‌شود.

۱-۱- یادگیری تقویتی چیست؟

یادگیری تقویتی[1] زیرشاخه‌ای از یادگیری ماشین[2] است که در آن یک عامل[3] هوش مصنوعی[4] با محیط[5] اطراف خود با کمک روش آزمون‌وخطا تعامل می‌کند و یک استراتژی رفتاری بهینه را بر اساس سیگنال‌های پاداش[6] دریافتی از تعاملات قبلی می‌آموزد. به یادگیرنده گفته نمی‌شود که چه اقداماتی را انجام دهد (مانند یادگیری با

[1] Reinforcement learning
[2] Machine learning
[3] Agent
[4] Artificial intelligence
[5] Environment
[6] Reward

نظارت)، بلکه باید کشف[1] کند که کدام اقدامات با امتحان‌کردن آنها بیشترین پاداش را به همراه دارد. دو ویژگی **جستجوی آزمون‌وخطا**[2] و **پاداش با تأخیر**[3] دو ویژگی متمایزکننده مهم یادگیری تقویتی هستند. **عامل** و **محیط** اجزای اساسی یادگیری تقویتی هستند، همان‌طور که در شکل ۱ نشان‌داده‌شده است. محیط موجودی است که عامل می‌تواند با آن تعامل داشته باشد.



شکل ۱ - تعامل عامل و محیط

از بین تمام اشکال یادگیری ماشینی، یادگیری تقویتی نزدیک‌ترین نوع یادگیری است که انسان و سایر حیوانات انجام می‌دهند، و بسیاری از الگوریتم‌های اصلی یادگیری تقویتی در اصل از سیستم‌های یادگیری بیولوژیکی الهام گرفته شده‌اند. یادگیری تقویتی هم از طریق یک مدل روان‌شناختی از یادگیری حیوانات که با برخی از داده‌های تجربی مطابقت دارد و هم از طریق یک مدل تأثیرگذار از بخش‌هایی از سیستم پاداش مغز، نتیجه داده است.

۲-۱- تفاوت یادگیری تقویتی با یادگیری با نظارت و بدون نظارت

یادگیری تقویتی با **یادگیری با نظارت**[4] متفاوت است، نوعی یادگیری که در بیشتر تحقیقات فعلی در زمینه یادگیری ماشین موردمطالعه قرار می‌گیرد. یادگیری تحت نظارت، یادگیری از مجموعه آموزشی از نمونه‌های برچسب‌دار ارائه شده توسط یک ناظر خارجی[5] آگاه است. هر مثال توصیفی از یک موقعیت همراه با یک مشخصه «**برچسب**» از اقدام صحیحی است که سیستم باید در آن موقعیت انجام دهد که اغلب برای شناسایی دسته‌ای است که موقعیت به آن تعلق دارد. هدف این نوع یادگیری این است که سیستم پاسخ‌های خود را برون‌یابی یا تعمیم دهد[6] تا در موقعیت‌هایی که در مجموعه آموزشی وجود ندارد به‌درستی عمل کند. این نوع مهمی از

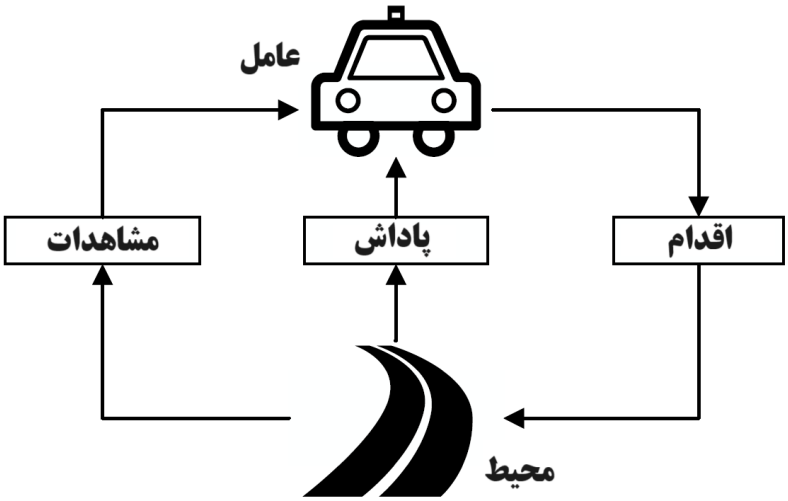
[1] Discover
[2] trial-and-error search
[3] delayed reward
[4] supervised learning
[5] external supervisor
[6] generalize

یادگیری اســت، اما به‌تنهایی برای یادگیری از تعامل کافی نیســت. در مســائل تعاملی، اغلب غیرعملی اســت که نمونه‌هایی از رفتار مطلوب به دســت آوریم که هم درســت و هم معرف همه موقعیت‌هایی باشــد که عامل باید در آنها عمل کند. در قلمروی ناشــناخته (جایی که انتظار می‌رود یادگیری ســودمندترین باشــد) یک عامل باید بتواند از تجربه خود بیاموزد.

یادگیری تقویتی همچنین با آنچه محققان یادگیری ماشـــینی آن را**یادگیری بدون نظارت**[1] می‌نامند، متفاوت است که معمولاً در مورد **یافتن ساختار پنهان** در مجموعه‌داده‌های **بدون برچسب** است. اگرچه ممکن است کســی وســوســه شــود که یادگیری تقویتی را نوعی یادگیری بدون نظارت بداند، زیرا بر نمونه‌هایی از رفتار صحیح تکیه نمی‌کند، اما یادگیری تقویتی به‌جای تلاش برای یافتن ســـاختار پنهان، ســـعی در **به حداکثر رســـاندن سیگنال پاداش** دارد. آشکارسازی ساختار در تجربه یک عامل مطمئناً می‌تواند در یادگیری تقویتی مفید باشد، اما به‌خودی‌خود به مشکل یادگیری تقویتی در به حداکثر رساندن سیگنال پاداش رسیدگی نمی‌کند.

بنابراین، یادگیری تقویتی در کنار یادگیری نظارت شـــده و یادگیری بدون نظارت و شـــاید پارادایم‌های دیگر، به‌عنوان سومین پارادایم یادگیری ماشینی شناخته می‌شود.

۳-۱- مفاهیم کلیدی یادگیری تقویتی



شکل ۲-اجزاء یادگیری تقویتی در رانندگی در جاده

دو جزء اصلی در یادگیری تقویتی عامل و محیط هستند. عامل تصمیم‌گیرنده است و راه‌حل یک مشکل است. محیط بیانگر یک مشکل است. یکی از تمایزات اساسی یادگیری تقویتی از سایر رویکردهای یادگیری ماشین

[1] unsupervised learning

این اســـت که عامل و محیط با هم تعامل دارند (شـــکل ۲). عامل تلاش می‌کند از طریق اعمال بر محیط تأثیر بگذارد و محیط به اقدامات عامل واکنش نشان می‌دهد.

۱-۳-۱- عامل(**agent**)

عامل **یادگیرنده یا تصـمیم‌گیرنده‌ای** اسـت که با انجام اقدامات خاص، انجام مشـاهدات و دریافت پاداش‌های نهایی **با محیط تعامل دارد**. در اکثر ســناریوهای عملی یادگیری تقویتی، عامل **قطعه نرم‌افزار** اســت که قرار است برخی از مشکلات را به روشی کم‌وبیش کارآمد حل کند.

منظور از یک عامل کامل، تعاملی و هدف‌جو همیشـــه چیزی مانند یک ارگانیســـم کامل یا ربات نیســـت. اینها نمونه‌هایی واضح هستند، اما یک عامل کامل، تعاملی و هدف جویی نیز می‌تواند جزء یک سیستم رفتاری بزرگ‌تر باشد. در این حالت، عامل به طور مستقیم با بقیه سیستم بزرگ‌تر و به طور غیرمستقیم با محیط سیستم بزرگ‌تر تعامل دارد. یک مثال ساده، عاملی است که سطح شارژ باتری ربات را کنترل می‌کند و دستوراتی را به معماری کنترل ربات ارسال می‌کند. محیط این عامل بقیه ربات، همراه با محیط ربات است.

۱-۳-۲- محیط(**environment**)

محیط، محدوده فیزیکی یا مجازی اســـت که عامل در آن قرار می‌گیرد و با آن تعامل دارد. محیط می‌تولند هر چیزی باشد که عامل با آن در تعامل است محیط **همه چیز خارج از یک عامل** است. در کلی‌ترین مفهوم، بقیه جهان است، اما این کمی بیش از حد است و حتی از ظرفیت رایانه‌های آینده فراتر می‌رود.

ارتباط عامل با محیط به پاداش (به‌دست‌آمده از محیط)، اقدامات (اجرا شده توسط عامل و داده شده به محیط) و مشاهدات (برخی اطلاعات علاوه بر پاداشی است که عامل از محیط دریافت می‌کند) محدود می‌شود.

۱-۳-۳- حالت و مشاهدات(**state & observations**)

حالت(وضعیت)، وضعیت فعلی محیط را نشان می‌دهد. این وضعیت ممکن است بر اساس داده‌های حسی که عامل دریافت می‌کند، تعریف شود. حالت می‌تواند اطلاعاتی مانند موقعیت، سرعت، شرایط محیطی و غیره را شامل شود. به‌عبارت‌دیگر محیط، با مجموعه‌ای از متغیرهای مرتبط با مسئله نمایش داده می‌شود. ترکیبی از تمام مقادیر ممکن که این مجموعه از متغیرها می‌تواند بگیرد، **فضای حالت** نامیده می‌شود. **حالت[1] مجموعه خاصی از مقادیر است که متغیرها در هر زمان معین می‌گیرند.**

[1] state

عامل‌ها ممکن است به وضعیت واقعی محیط دسترسی داشته باشند یا نداشته باشند. بااین‌حال، به‌هرحال، عامل‌ها می‌توانند چیزی را از محیط مشاهده کنند. مجموعه‌ای از متغیرهایی که عامل در هر زمان معین درک می‌کند، **مشاهده**[1] نامیده می‌شود.

ترکیبی از تمام مقادیر ممکن که این متغیرها می‌توانند بگیرند **فضای مشاهده** است. بدانید که حلت و مشاهده عباراتی هستند که به‌جای یکدیگر در جامعه یادگیری تقویتی استفاده می‌شوند. این به این دلیل است که اغلب عوامل اجازه دیدن وضعیت داخلی محیط را دارند، اما همیشه این‌طور نیست. ممکن است تفاوتی بین حالت‌ها و مشاهدات وجود داشته باشد.

۴-۳-۱- اقدام(action)

اقدامات یا کنش‌ها یا عمل‌ها[2] کارهایی هستند که یک عامل می‌تواند در محیط انجام دهد. برای مثال، اقدامات می‌توانند حرکاتی باشند که طبق قوانین بازی مجاز است (اگر بازی باشد). در هر حلت، محیط مجموعه‌ای از اقدامات را در دسترس می‌گذارد که عامل می‌تواند از بین آنها انتخاب کند. اغلب مجموعه اقدامات برای همه حالت‌ها یکسان است. مجموعه همه اقدامات در همه حالت‌ها **فضای عمل** نامیده می‌شود.

در زیر نمونه‌ای مختصر از مسئله، عامل، محیط، اقدامات احتمالی و مشاهدات یادگیری تقویتی آمده است:

مسئله: شما در حال رانندگی با ماشین خود هستید. **عامل**: بخشی از مغز شما که تصمیم می گیرد. **محیط**: ساخت و مدل ماشین شما، ماشین های دیگر، سایر رانندگان، آب و هوا، جاده ها، لاستیک ها و غیره. **اقدامات(عمل)**: هدایت با x، شتاب با y. ترمز توسط z، روشن کردن چراغ های جلو. مه شکن، پنجره ها، پخش موسیقی. **مشاهدات**: شما به مقصد خود نزدیک می شوید. یک راه بندان در خیابان اصلی وجود دارد. ماشین کنار شما بی احتیاط رانندگی می کند. باران شروع به باریدن می کند. یک افسر پلیس جلوی شما رانندگی می کند.

۵-۳-۱- خط‌مشی(policy)

یک **خط‌مشی یا سیاست**[3] نحوه رفتار عامل یادگیری را در یک زمان معین تعریف می‌کند. به‌طورکلی، یک خط‌مشی، **نگاشتی**[4] است از حالت‌های درک شده از محیط به اقداماتی است که باید در آن حالت‌ها انجام شود.

[1] observation
[2] actions
[3] policy
[4] mapping

این با چیزی مطابقت دارد که در روان‌شناسی مجموعه‌ای از **قوانین یا ارتباط‌های محرک-پاسخ**[1] نامیده می‌شود. در برخی موارد این خط‌مشی ممکن است **یک تابع ساده یا جدول جستجو**[2] باشد، درحالی‌که در موارد دیگر ممکن است شامل محاسبات گسترده مانند فرایند جستجو باشد. خط‌مشی هسته اصلی یک عامل یادگیری تقویتی است به این معنا که به‌تنهایی برای تعیین رفتار کافی است. به‌طورکلی، خط‌مشی‌ها ممکن است تصادفی باشند و احتمالات را برای هر عمل مشخص کنند.

۱-۳-۶- پاداش(**reward**)

سیگنال پاداش[3]، هدف یک مسئله یادگیری تقویتی را مشخص می‌کند. در هر مرحله زمانی[4]، محیط یک عدد واحد به نام پاداش را برای عامل یادگیری تقویتی ارسال می‌کند. تنها هدف عامل **به حداکثر رساندن کل پاداشی** است که در درازمدت دریافت می‌کند؛ بنابراین **سیگنال پاداش، رویدادهای خوب و بد را برای عامل مشخص می‌کند**. در یک سیستم بیولوژیکی، ممکن است پاداش‌ها را **مشابه تجربه لذت یا درد** بدانیم. آنها ویژگی‌های فوری و تعیین‌کننده مسئله‌ای هستند که عامل با آن مواجه است. سیگنال پاداش مبنای اولیه برای تغییر خط‌مشی است. اگر اقدامی که توسط خط‌مشی انتخاب می‌شود با پاداش کم همراه باشد، ممکن است این خط‌مشی برای انتخاب اقدام دیگری در آن موقعیت در آینده تغییر کند. به‌طورکلی، سیگنال‌های پاداش ممکن است توابع تصادفی از وضعیت محیط و اقدامات انجام شده باشد.

۱-۳-۷- تابع ارزش(**value function**)

درحالی‌که سیگنال پاداش نشان می‌دهد که چه چیزی در **معنای فوری خوب است**، یک **تابع ارزش**[5] مشخص می‌کند که **چه چیزی در بلندمدت خوب است**. به‌طورکلی، ارزش یک حالت، مقدار کل پاداشی است که یک عامل می‌تواند انتظار داشته باشد در آینده جمع‌آوری کند. درحالی‌که پاداش‌ها، **مطلوبیت ذاتی و فوری حالات** محیطی را تعیین می‌کنند، ارزش‌ها پس از درنظرگرفتن حالت‌هایی که احتمالاً دنبال می‌شوند و پاداش‌های موجود در آن حالت‌ها، **مطلوبیت بلندمدت حالت‌ها** را نشان می‌دهند. به‌عنوان مثال، یک حالت ممکن است همیشه یک پاداش فوری کم داشته باشد، اما همچنان ارزش بالایی داشته باشد، زیرا به طور منظم توسط سایر حالت‌هایی که پاداش‌های بالایی ارائه می‌کنند، دنبال می‌شود یا برعکس آن نیز می‌تواند درست باشد. برای انجام یک تشبیه انسانی، پاداش‌ها تا حدودی مانند لذت (اگر زیاد) و درد (اگر کم باشد) هستند، درحالی‌که ارزش‌ها با

[1] stimulus–response rules or associations
[2] lookup table
[3] reward signal
[4] time step
[5] value function

یک قضاوت دقیق‌تر و دوراندیشانه‌تر از اینکه ما چقدر از محیطمان در حالت خاصی خشنود یا ناراضی هستیم مطابقت دارد.

پاداش‌ها به یک معنا اولیه[1] هستند، درحالی‌که ارزش‌ها، به‌عنوان پیش‌بینی پاداش، ثانویه هستند. **بدون پاداش هیچ ارزشی وجود نخواهد داشت** و تنها هدف از تخمین ارزش‌ها دستیابی به پاداش بیشتر است. بااین‌وجود، این ارزش‌ها هست که هنگام تصمیم‌گیری و ارزیابی بیشتر به آن‌ها توجه می‌کنیم. انتخاب عمل بر اساس قضاوت‌های ارزشی انجام می‌شود. ما به دنبال اقداماتی هستیم که بالاترین ارزش را به همراه داشته باشد، نه بالاترین پاداش، زیرا این اقدامات در درازمدت بیشترین پاداش را برای ما به همراه دارد. متأسفانه، تعیین ارزش‌ها بسیار دشوارتر از تعیین پاداش است. جوایز اساساً مستقیماً توسط محیط داده می‌شوند، اما ارزش‌ها باید از روی توالی مشاهداتی که یک عامل در طول عمر خود انجام می‌دهد، تخمین زده و مجدداً تخمین زده شوند. در واقع، مهم‌ترین مؤلفه تقریباً همه الگوریتم‌های یادگیری تقویتی که در نظر می‌گیریم، روشی برای تخمین دقیق ارزش‌ها است. نقش اصلی برآورد ارزش مسلماً مهم‌ترین چیزی است که در مورد یادگیری تقویتی طی شش دهه گذشته آموخته شده است.

۸-۳-۱- مدل(**model**)

برخی از سیستم‌های یادگیری تقویتی از **مدلی[2] از محیط** استفاده می‌کنند. چیزی که **رفتار محیط را تقلید** می‌کند، یا به‌طورکلی تر، اجازه می‌دهد تا استنتاج‌هایی در مورد **نحوه رفتار محیط** انجام شود. برای مثال، باتوجه‌به یک حالت و عمل، مدل ممکن است حالت بعدی و پاداش بعدی را پیش‌بینی کند. مدل‌ها برای برنامه‌ریزی استفاده می‌شوند که به معنی استفاده از هر راهی برای تصمیم‌گیری در مورد یک مسیر عملی، با درنظرگرفتن موقعیت‌های احتمالی آینده، قبل از تجربه واقعی است. روش‌هایی برای حل مسائل یادگیری تقویتی که از مدل‌ها و برنامه‌ریزی استفاده می‌کنند، **روش‌های مبتنی بر مدل**[3] نامیده می‌شوند، در مقابل روش‌های بدون مدل[4] که به‌صراحت یادگیرندگان آزمون‌وخطا هستند و تقریباً برعکس برنامه‌ریزی تلقی می‌شوند. برخی از سیستم‌های یادگیری تقویتی به طور هم‌زمان هم با آزمون‌وخطا یاد می‌گیرند، هم مدلی از محیط را یاد می‌گیرند و از مدل برای برنامه‌ریزی استفاده می‌کنند. یادگیری تقویتی مدرن طیفی از یادگیری سطح پایین، آزمون‌وخطا تا برنامه‌ریزی سطح بالا و مشورتی را در بر می‌گیرد.

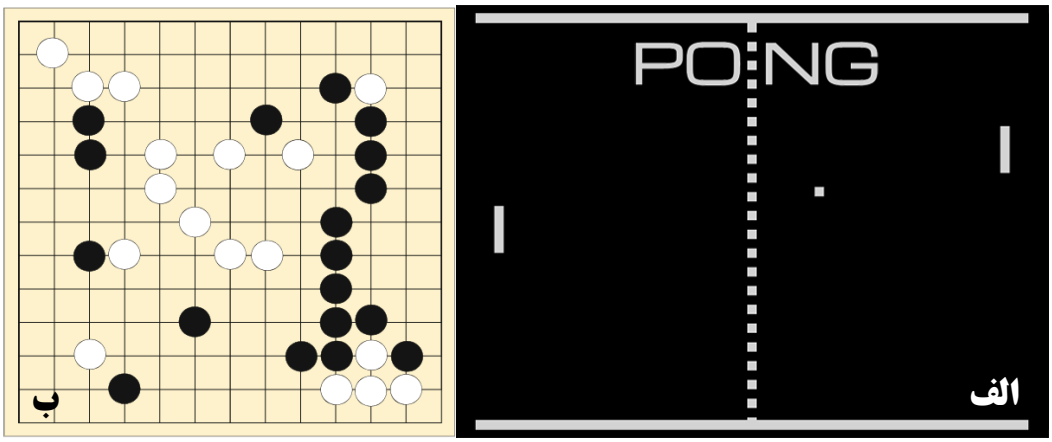
[1] primary
[2] model
[3] model-based
[4] model-free

۱-۴- چالش‌ها و محدودیت‌های یادگیری تقویتی

درحالی‌که یادگیری تقویتی قابلیت‌های قدرتمندی را ارائه می‌کند، با محدودیت‌های خاصی نیز همراه است. در حال حاضر چندین چالش و محدودیت عمده در الگوریتم‌های یادگیری تقویتی وجود دارد:

۱-۴-۱- کارایی نمونه

یکی چالش‌های یادگیری تقویتی **کارایی نمونه**[1] است، به این معنی که **یادگیری به داده‌های زیادی نیاز دارد**. در کارهایی که شبیه‌سازها در دسترس هستند، یک عامل کامپیوتری می‌تواند تعاملات بی حد و حصری با یک محیط ثابت داشته باشد. موفقیت AlphaGo (اولین برنامه رایانه‌ای است که می‌تواند یک بازیکن حرفه‌ای بازی Go (شکل ۳، ب) را شکست دهد) بر اساس تعداد زیادی بازی است که هر بازیکن انسانی نمی‌تواند در طول زندگی آن‌ها را تجربه کند. در محیط‌های فیزیکی واقعی، مانند کنترل ربات یا تعامل انسانی، تجربه واقعی می‌تواند زمان‌بر یا پرهزینه باشد و محیط می‌تواند مدام در حال تغییر باشد به‌طوری‌که یادگیرندگان کند نتوانند از پس آن بر بیایند.



شکل ۳- (الف) محیط بازی Pong و (ب) محیط بازی Go

به‌عنوان مثال بازی Pong را در نظر بگیرید (شکل ۳، الف): یک انسان عادی فقط به ده‌ها آزمایش نیاز دارد تا اساساً در بازی تسلط یابد و به امتیاز نسبتاً خوبی دست یابد. بااین‌حال، برای الگوریتم‌های یادگیری تقویتی موجود (به‌ویژه با روش‌های بدون مدل)، ممکن است حداقل به ده‌ها هزار نمونه نیاز داشته باشد تا به‌تدریج برخی از سیاست‌های مفید را یاد بگیرد. این یک مشکل اساسی در یادگیری تقویتی ایجاد می‌کند: چگونه می‌توانیم الگوریتم یادگیری تقویتی کارآمدتری طراحی کنیم تا عاملی با مثال‌های کمتر سریع‌تر یاد بگیرد؟

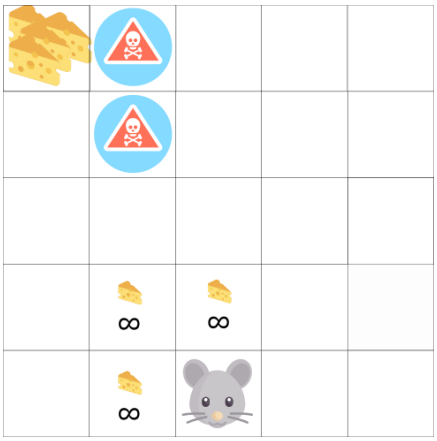
[1] sample efficiency

اهمیت این مشـــکل بیشـــتر به دلیل هزینه تعاملات زمان واقعی[1] یا دنیای واقعی[2] بین عامل و محیط و یا حتی زمان و انرژی مصـرفی تعاملات در محیط‌های شـبیه‌سـازی‌شـده موجود اسـت. بسـیاری از الگوریتم‌های یادگیری تقویتی کنونی در یک مسـئله در **مقیاس بزرگ یا فضـای پیوسـته آن‌قدر بازده یادگیری[3] پایینی دارند** که یک فرایند آموزشی معمولی حتی با شبیه‌سازی سریع هنوز به **زمان انتظار غیرقابل‌تحمل با توان محاسباتی فعلی** نیاز دارد. این می‌تواند برای تعاملات دنیای واقعی بدتر باشـد. مشـکلات بالقوه مصـرف زمان، فرسـودگی تجهیزات، ایمنی در حین اکتشـاف یادگیری تقویتی و خطرات موارد شـکست، همگی الزامات سـخت‌گیرانه‌تری را در مورد بهره‌وری یادگیری روش‌های یادگیری تقویتی در عمل ایجاد می‌کنند.

۲-۴-۱- اکتشاف در مقابل بهره‌برداری

یکی از چالش‌هایی که در یادگیری تقویتی مطرح می‌شـود، توازن[4] بین **اکتشـاف**[5] (انتخاب اقدامات تصـادفی تا بتوانیم بیشـــتر بیاموزیم) و **بهره‌برداری**[6](انتخاب بهترین اقدام بر اســـاس آنچه تاکنون می‌دانیم) اســـت. برای به‌دسـت‌آوردن پاداش زیاد، یک عامل یادگیری تقویتی باید اقداماتی را ترجیح دهد که درگذشـته امتحان کرده و در تولید پاداش مؤثر بوده اســـت. اما برای کشـــف چنین اقداماتی، باید اقداماتی را امتحان کند که قبلاً انتخاب نکرده اسـت. عامل باید از آنچه قبلاً تجربه کرده اسـت برای به‌دسـت‌آوردن پاداش بهره‌برداری کند، اما همچنین باید برای انتخاب اقدامات بهتر در آینده، کاوش کند. معضل این است که نه اکتشاف و نه بهره‌برداری را نمی‌توان منحصراً بدون شکست در کار دنبال کرد. عامل باید انواع مختلفی از اقدامات را امتحان کند و به‌تدریج از اقداماتی که به نظر بهترین هســـتند، حمایت کند. در یک کار تصـــادفی، هر عمل باید بارها امتحان شـــود تا تخمین قابل‌اعتمادی از پاداش مورد انتظار آن به دست آید. در ادامه مثالی از این چالش ارائه شده است.

[1] real-time
[2] real-world
[3] learning efficiency
[4] trade-off
[5] exploration
[6] exploitation



شکل ۴- محیط بازی موش و پنیر

به‌خاطر داشته باشید که هدف عامل در یادگیری تقویتی بیشینه‌سازی پاداش است. بااین‌حال می‌توان در یک تله متداول افتاد. برای مثال در بازی شکل ۴، موش می‌تواند میزان نامتناهی پنیرهای کوچک داشته باشد (۱+ برای هر کدام). اما در بالای هزارتو، حجم زیادی پنیر وجود دارد (۱۰۰۰+). اگر فقط روی پاداش تمرکز شود، عامل هیچ‌وقت به آن کوه عظیم پنیر نمی‌رسد. در عوض، تنها نزدیک‌ترین منابع پاداش را اکتشاف(جستجو) می‌کند، حتی اگر این منابع کوچک باشند. اگر عامل بیشتر جستجو کند و کمتر بهره‌برداری کند، می‌تواند پاداش بزرگی پیدا کند. ایجاد تعادل میان اکتشاف اقدامات جدید و بهره‌برداری از دانش آموخته‌شده چالش‌برانگیز است؛ زیرا اکتشاف بیش از حد ممکن است تصمیم‌گیری بهینه را به تأخیر بیندازد و بهره‌برداری بیش از حد ممکن است به راه‌حل‌های غیربهینه بینجامد. معضل اکتشاف و بهره‌برداری برای چندین دهه توسط ریاضی‌دانان به‌شدت موردمطالعه قرار گرفته است، اما هنوز حل نشده باقی‌مانده است. به‌هرحال باید قوانینی را تعیین کرد تا به برقراری توازن و مدیریت آن کمک کند.

۳-۴-۱- پاداش با تأخیر

یکی از اصول بنیادی یادگیری تقویتی این است که **پاداش می‌تواند به‌طورجدی پس از اعمال به تعویق بیفتد**[1]. عامل یادگیری می‌تواند پاداش‌های کوتاه‌مدت را با دستاوردهای بلندمدت معاوضه کند. درحالی‌که این اصل بنیادی یادگیری تقویتی را مفید می‌کند، همچنین **کشف سیاست بهینه را برای عامل دشوار می‌کند**. این مشکل به‌ویژه در محیط‌هایی که تا زمانی که تعداد زیادی از اقدامات متوالی انجام نشود، نتیجه ناشناخته است، وجود دارد. در این سناریو، اختصاص اعتبار به یک اقدام قبلی برای نتیجه نهایی چالش‌برانگیز است و می‌تواند مغایرت زیادی را در طول آموزش ایجاد کند. بازی شطرنج یک مثال مرتبط در اینجا است، جایی که

[1] delayed

نتیجه بازی تا زمانی که هر دو بازیکن تمام حرکات خود را انجام ندهند، نامعلوم است به‌طوری‌که یک حرکت قوی در میانه بازی می‌تواند نتایج را تغییر دهد.

۱-۴-۴- مهندسی پاداش

مشکل دیگر یادگیری تقویتی مربوط به **توابع پاداش و درک معنای پاداش** است. طراحی توابع پاداش مناسب که با رفتار مدنظر هماهنگ باشد می‌تواند پیچیده باشد و تعریف پاداش‌هایی که به طور دقیق هدف‌های عامل را نشان می‌دهند، یک کار غیرضروری است. اگر یک متخصص انسانی پاداش‌هایی را که عامل سعی در به حداکثر رساندن آن دارد تعریف کند، آیا به این معنی است که ما تا حدودی بر این عامل "نظارت" داریم؟ و آیا این چیز خوبی است؟ آیا پاداش باید تاحدامکان متراکم باشد که یادگیری را سریع‌تر می‌کند، یا تاحدامکان پراکنده که راه‌حل‌ها را هیجان‌انگیزتر و منحصربه‌فردتر می‌کند؟

ما به‌عنوان انسان، به نظر نمی‌رسد که پاداش‌های مشخصی داشته باشیم. اغلب، همان فرد می‌تواند یک رویداد را به‌سادگی با تغییر دیدگاه خود مثبت یا منفی ببیند. علاوه بر این، طراحی تابع پاداش برای کاری مانند راه‌رفتن، ساده نیست. آیا این حرکت روبه‌جلو است که باید هدف قرار دهیم یا سقوط‌نکردن؟ عملکرد پاداش "کامل" برای پیاده‌روی انسان چیست؟

۱-۴-۵- عدم تفسیرپذیری

هنگامی که یک عامل یادگیری تقویتی سیاستی بهینه را آموخته و در محیط مستقر می‌شود، بر اساس تجربه خود اقداماتی را انجام می‌دهد. برای یک ناظر خارجی، دلیل این اقدامات ممکن است واضح نباشد. این عدم تفسیرپذیری در ایجاد اعتماد بین عامل و ناظر اختلال ایجاد می‌کند. اگر یک ناظر بتواند اقداماتی را که عامل یادگیری تقویتی انجام می‌دهد توضیح دهد، به او در درک بهتر مسئله و کشف محدودیت‌های مدل، به‌ویژه در محیط‌های پرخطر کمک می‌کند.

۱-۴-۶- ملاحظه‌های اخلاقی

الگوریتم‌های یادگیری تقویتی می‌توانند رفتارهای نامطلوب یا مضر را بیاموزند، اگر به‌دقت طراحی نشده باشند، به طور بالقوه نگرانی‌های اخلاقی و نیاز به نظارت دقیق را افزایش می‌دهند.

۱-۴-۷- سایر چالش‌ها

مشکل دیگری که سر راه یادگیری تقویتی وجود دارد، مدت‌زمان و منابع محاسباتی‌ای است که لازم است تا اطمینان حاصل کنیم یادگیری به‌درستی انجام شده است. از طرفی، هرچه محیط آموزشی بزرگ‌تر باشد به زمان و منابع بیشتری برای فرایند آموزش الگوریتم نیاز است.

جدای از چالش‌های ذکر شـــده در بالا برای یادگیری تقویتی چالش‌های دیگری مانند فرا یادگیری و یادگیری بازنمایی برای **تعمیم‌پذیری**[1] روش‌های یادگیری تقویتی در بین وظایف (این که عامل چگونه می‌تواند بر اســاس آنچه که از یک کار قدیمی آموخته اســـت، ســـریع‌تر در مورد یک کار جدید یاد بگیرید؟)، یادگیری تقویتی چندعاملی[2] با ســایر عوامل به‌عنوان بخشــی از محیط، انتقال شــبیه‌ســازی به واقعیت[3] برای پر کردن شــکاف بین محیط‌های شــبیه‌ســازی‌شــده و دنیای واقعی، یادگیری تقویتی در مقیاس بزرگ[4] با چارچوب‌های آموزشــی موازی برای کوتاه‌کردن زمان آموزش و غیره.

۵-۱- کاربردها

یادگیری تقویتی کاربردهای متعددی در حوزه‌های مختلف پیدا کرده است، از جمله:

رباتیک: یادگیری تقویتی ربات‌ها را قادر می‌کند تا عمل‌ها و حرکت‌های خود را بر اساس آزمون‌وخطا یاد بگیرند و بهبود بخشند و به آن‌ها اجازه می‌دهد در محیط‌های پیچیده حرکت کنند یا اشیا را دست‌کاری کنند.

بازی: الگوریتم‌های یادگیری تقویتی در انجام‌دادن بازی‌های پیچیده، مانند شـطرنج، Go و بازی‌های ویدئویی، به موفقیت چشمگیری دست یافته‌اند و در برخی موارد از عملکرد انسان پیشی گرفته‌اند.

وســـایل نقلیه خودمختار: تکنیک‌های یادگیری تقویتی را می‌توان برای آموزش خودروهای خودران برای تصمیم‌گیری بهینه در زمان واقعی به کار برد که به حمل‌ونقل ایمن‌تر و کارآمدتر می‌انجامد.

مدیریت منابع: یادگیری تقویتی می‌تواند برای بهینه‌ســـازی تخصـــیص منابع، زمان‌بندی و تصـــمیم‌گیری در حوزه‌هایی مانند مدیریت انرژی، لجستیک و ارتباطات استفاده شود.

مراقبت‌های بهداشــتی :یکی از راه‌های مهم برای اســتقرار یادگیری تقویتی، رژیم‌های درمان پویا اســت. برای تهیه یک رژیم‌درمانی پویا شـخصـی باید مجموعه‌ای از مشـاهدات بالینی بیمار و ارزیابی‌های پزشـکی بیمار را وارد کند و ســپس با اســتفاده از نتایج قبلی و ســابقه پزشــکی بیمار، عامل پیشــنهاداتی درباره نوع درمان، دوز دارو و جدول زمان‌بندی بیمار ارائه می‌دهد. این کار باعث سریع‌تر شدن روند درمان بیمار می‌شود.

[1] generality
[2] multi-agent reinforcement learning
[3] sim-to-real transfer
[4] large-scale

۶-۱- فرایند تصمیم‌گیری مارکوف(MDPs)

فرایندهای تصمیم‌گیری مارکوف رسمی‌سازی کلاسیک تصمیم‌گیری متوالی هستند که در آن اقدامات نه‌تنها بر پاداش‌های فوری، بلکه بر موقعیت‌ها یا حالت‌های بعدی و از طریق آن پاداش‌های آینده تأثیر می‌گذارد؛ بنابراین MDPها شامل پاداش تأخیری و نیازمند توازن پاداش فوری و تسخیری هستند. فرایند تصمیم‌گیری مارکوف یک شکل ریاضی ایده‌آل از مسئله یادگیری تقویتی هستند که می‌توان برای آن گزاره‌هایی نظری دقیقی ارائه داد. فرایند مارکوف و فرایند تصمیم‌گیری مارکوف به طور گسترده در علوم کامپیوتر و سایر زمینه‌های مهندسی استفاده می‌شود.

۱-۶-۱- فرایند مارکوف(MP)

فرایند مارکوف ساده‌ترین فرزند از خانواده مارکوف است که به **زنجیره مارکوف** نیز معروف است. تصور کنید که سیستمی در مقابل خود دارید که فقط می‌توانید آن را مشاهده کنید. آنچه مشاهده می‌کنید **حالت‌ها** نامیده می‌شود و سیستم می‌تواند طبق برخی از قوانین دینامیک(حرکت) [1] بین حالت‌ها جابه‌جا شود. باز هم شما نمی‌توانید سیستم را تحت‌تأثیر قرار دهید، بلکه فقط می‌توانید شاهد تغییر حالت‌ها باشید.

همان‌طور که گفته شد تمام حالت‌های ممکن برای یک سیستم مجموعه‌ای به نام **فضای حالت** را تشکیل می‌دهند. برای فرایندهای مارکوف، ما نیاز داریم که این مجموعه حالت‌ها محدود باشد. مشاهدات شما دنباله‌ای از حالت‌ها یا **زنجیره‌ای** را تشکیل می‌دهد (به همین دلیل است که فرایند مارکوف، زنجیره مارکوف نیز نامیده می‌شود). به‌عنوان مثال، با نگاهی به ساده‌ترین مدل آب‌وهوا در برخی از شهرها، می‌توان روز جاری را به‌صورت آفتابی یا بارانی مشاهده کرد که فضای حالت ماست. دنباله‌ای از مشاهدات در طول زمان زنجیره‌ای از حالت‌ها را تشکیل می‌دهد، مانند [آفتابی، آفتابی، بارانی، آفتابی و... و به این **تاریخ**[2] می‌گویند.

فرایند مارکوف، باید **ویژگی مارکوف**[3] را برآورده کند، به این معنی که دینامیک سیستم آینده از هر حالتی، باید فقط به این حالت بستگی داشته باشد. نکته اصلی ویژگی مارکوف این است که هر حالت قابل‌مشاهده را برای توصیف آینده سیستم خودکفا کند. به‌عبارت‌دیگر، **ویژگی مارکوف مستلزم آن است که حالات سیستم از یکدیگر متمایز و منحصربه‌فرد باشند**. در این حالت، تنها یک حالت برای مدل‌سازی حرکت‌های آینده سیستم موردنیاز است و نه کل تاریخچه یا مثلاً آخرین N حالت.

[1] dynamics
[2] history
[3] markov property

در مورد مثال آب‌وهوا، ویژگی مارکوف مدل ما را محدود می‌کند تا فقط مواردی را نشان دهد که یک روز آفتابی را می‌توان با یک روز بارانی با احتمال یکسان دنبال کرد، صرف‌نظر از تعداد روزهای آفتابی که درگذشته دیده‌ایم. این یک مدل خیلی واقعی[1] نیست، زیرا از روی عقل سلیم می‌دانیم که احتمال بارندگی فردا نه‌تنها به شرایط فعلی بلکه به تعداد زیادی از عوامل دیگر مانند فصل، عرض جغرافیایی و وجود کوه‌ها و دریا بستگی دارد؛ بنابراین، مثال ما واقعاً ساده‌لوحانه است، اما درک محدودیت‌ها و تصمیم‌گیری آگاهانه در مورد آنها مهم است.

البته، اگر بخواهیم مدل خود را پیچیده‌تر کنیم، همیشه می‌توانیم این کار را با گسترش فضای حالت خود انجام دهیم که به ما امکان می‌دهد وابستگی‌های بیشتری را در مدل به قیمت یک فضای حالت بزرگ‌تر ثبت کنیم. به‌عنوان مثال، اگر می‌خواهید به طور جداگانه احتمال روزهای بارانی در تابستان و زمستان را ثبت کنید، می‌توانید فصل را در حالت‌های خود قرار دهید. در این صورت فضای حالت شما [آفتابی+تابستان، آفتابی+زمستان، بارانی+تابستان، بارانی+زمستان] خواهد بود.

ازآنجایی‌که مدل سیستم شما با ویژگی Markov مطابقت دارد، می‌توانید احتمالات انتقال را با یک **ماتریس انتقال**[2] که یک ماتریس مربع به اندازه N×N است که N تعداد حالت‌های مدل ما است، ثبت کنید. هر سلول در یک ردیف، i، و یک ستون، j، در ماتریس حاوی احتمال انتقال سیستم از حالت i به حالت j است.
به‌عنوان مثال، در مثال آفتابی/بارانی ما، ماتریس انتقال می‌تواند به‌صورت زیر باشد:

جدول ۱ - ماتریس انتقال فضای حالت (آفتابی/بارانی)

	آفتابی	بارانی
آفتابی	۰.۸	۰.۲
بارانی	۰.۱	۰.۹

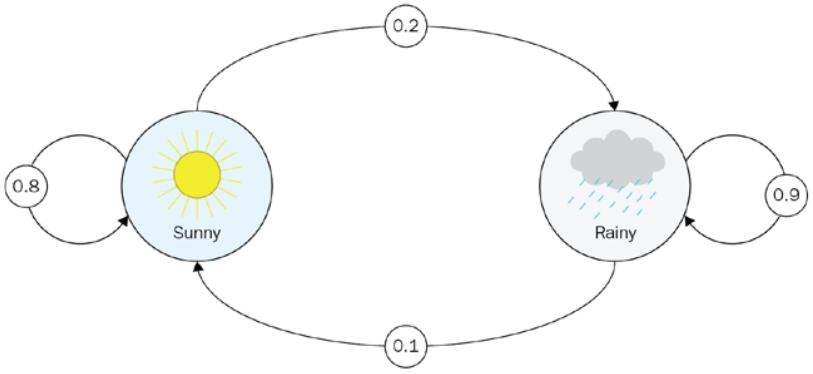
در این صورت، اگر روز آفتابی داشته باشیم، ۸۰ درصد احتمال آفتابی بودن روز بعد و ۲۰ درصد احتمال بارانی بودن روز بعد وجود دارد. اگر یک روز بارانی را مشاهده کنیم، به احتمال ۱۰٪ هوا بهتر می‌شود و به احتمال ۹۰٪ روز بعد بارانی است.

بنابراین، تعریف رسمی فرایند مارکوف به شرح زیر است:

- مجموعه‌ای از حالات (S) که یک سیستم می‌تواند در آن باشد
- یک ماتریس انتقال (T)، با احتمالات انتقال که دینامیک سیستم را تعریف می‌کند

[1] realistic
[2] transition matrix

شکل ۵ یک نمایش بصری مفید از فرایند مارکوف یک نمودار با گره‌های مربوط به حالت‌ها و یال‌های سیستم است که با احتمالاتی برچسب‌گذاری شده است که انتقال احتمالی از یک حالت به حالت دیگر را نشان می‌دهد. اگر احتمال انتقال ۰ باشد، یال نمی‌کشیم (راهی برای رفتن از یک حالت به حالت دیگر وجود ندارد).



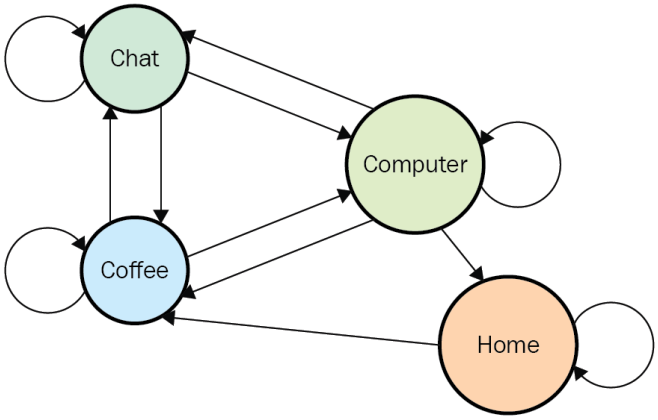
شکل ۵ – مدل آب و هوای آفتابی/بارانی

باز هم ما فقط در مورد مشاهده صحبت می‌کنیم. هیچ راهی برای تأثیرگذاری بر آب‌وهوا وجود ندارد، بنابراین فقط آن را مشاهده می‌کنیم و مشاهدات خود را ثبت می‌کنیم.

برای مثال پیچیده‌تر، بیایید مدل دیگری به نام کارمند اداری را در نظر بگیریم. فضای حالت او در مثال ما دارای حالات زیر است:

- **خانه(Home)**: او در اداره نیست
- **رایانه(Computer)**: او در اداره کار روی رایانه خود دارد
- **قهوه(Coffee)**: او در اداره مشغول نوشیدن قهوه است
- **گفتگوکردن(Chat)**: او در حال گفتگو با همکارانش در اداره است

نمودار انتقال حالت در شکل صفحه بعد نشان داده شده است:



شکل ۶ -نمودار انتقال حالت برای کارمند اداره

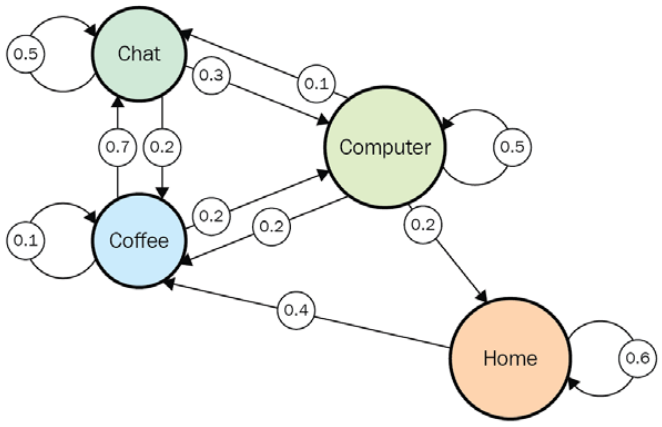
ما فرض می‌کنیم که روز کاری کارمند اداری ما معمولاً از حالت خانه شروع می‌شود و او بدون استثنا روز خود را با قهوه شـروع می‌کند (بدون یال **خانه** ← **کامپیوتر** و بدون یال **خانه** ← **گفتگو**). نمودار بالا همچنین نشـان می‌دهد که روزهای کاری همیشه از حالت **رایانه** به پایان می‌رسد (رفتن به حالت خانه)

ماتریس انتقال برای نمودار قبل به شرح زیر است:

جدول ۲ – ماتریس انتقال برای کارمند اداره

	خانه	قهوه	گفتگو	کامپیوتر
خانه	۶۰٪	۴۰٪	۰٪	۰٪
قهوه	۰٪	۱۰٪	۷۰٪	۲۰٪
گفتگو	۰٪	۲۰٪	۵۰٪	۳۰٪
کامپیوتر	۲۰٪	۲۰٪	۱۰٪	۵۰٪

همان‌طور که در اینجا نشان‌داده‌شده است، احتمالات انتقال را می‌توان مستقیماً در نمودار انتقال حالت قرار داد:



شکل ۷ – نمودار انتقال حالت با احتمالات انتقال

در عمل، ما به‌ندرت ماتریس انتقال دقیق را می‌دانیم. یک موقعیت بسیار واقعی‌تر در دنیای واقعی، زمانی است که ما فقط مشاهداتی از حالات سیستم خود داریم که به آنها **قسمت(اپیزود)** [1] نیز می‌گویند:

- خانه ← قهوه ← قهوه ← گفتگو ← گفتگو ← قهوه ← کامپیوتر ← کامپیوتر ← خانه
- رایانه ← رایانه ← گفتگو ← گفتگو ← قهوه ← رایانه ← رایانه ← رایانه
- خانه ← خانه ← قهوه ← گفتگو ← رایانه ← قهوه ← قهوه

تخمین ماتریس انتقال از روی مشاهداتمان پیچیده نیست - ما فقط همه انتقال‌ها را از هر حالتی می‌شماریم و آنها را تا مجموع ۱ نرمال می‌کنیم. هر چه داده‌های مشاهدات بیشتری داشته باشیم، تخمین ما به مدل اصلی واقعی نزدیک‌تر خواهد بود.

همچنین شایان‌ذکر است که ویژگی مارکوف بر ایستایی[2] دلالت دارد (توزیع انتقال برای هیچ حالتی در طول زمان تغییر نمی‌کند). غیرایستایی به این معنی است که یک عامل[3] پنهان وجود دارد که بر دینامیک سیستم ما تأثیر می‌گذارد و این عامل در مشاهدات گنجانده نمی‌شود. بااین‌حال، این در تضاد با ویژگی مارکوف است که مستلزم آن است که توزیع احتمال اساسی برای همان حالت بدون درنظرگرفتن تاریخچه انتقال یکسان باشد.

درک تفاوت بین انتقال‌های واقعی مشاهده شده در یک قسمت و توزیع اساسی ارائه شده در ماتریس انتقال بسیار مهم است. قسمت‌های مشخصی که مشاهده می‌کنیم به طور تصادفی از توزیع مدل نمونه‌برداری می‌شوند، بنابراین می‌توانند از قسمتی به قسمت دیگر متفاوت باشند. بااین‌حال، احتمال انتقال واقعی برای نمونه‌برداری یکسان باقی می‌ماند. اگر این‌طور نباشد، اصالت شکلی[4] زنجیره مارکوف غیرقابل‌اجرا می‌شود.

اکنون می‌توانیم جلوتر برویم و مدل فرایند مارکوف را گسترش دهیم تا آن را به مشکلات یادگیری تقویتی نزدیک‌تر کنیم. بیایید به نمودار جایزه پاداش کنیم!

۲-۶-۱- فرایند پاداش مارکوف

برای معرفی پاداش، باید مدل فرایند مارکوف خود را کمی گســـترش دهیم. اول، ما باید برای انتقال خود از حالتی به حالت دیگر ارزش بیفزاییم. ما در حال حاضر احتمال داریم، اما از احتمال برای ثبت دینامیک سیستم استفاده می‌شود، بنابراین اکنون یک عدد اسکالر اضافی بدون بار اضافی داریم.

[1] episodes
[2] stationarity
[3] factor
[4] formalism

پاداش را می‌توان به اشـــکال مختلف نشـــان داد. کلی‌ترین راه این اســـت که یک ماتریس مربع دیگر، شـــبیه به ماتریس انتقال، با پاداش برای انتقال از حالت i به حالت j که در ردیف i و ستون j قرار دارد، داشته باشید.

همان‌طور که گفته شد، پاداش می‌تواند مثبت یا منفی، بزرگ یا کوچک باشد. در برخی موارد، این نمایش اضافی است و می‌توان آن را ساده کرد. به‌عنوان مثال، اگر برای رسیدن به حالتی بدون توجه به حالت قبلی، پاداش داده شود، می‌توانیم فقط جفت‌های **حالت←پاداش** را نگه داریم که نمایش فشرده‌تری هستند. بااین‌حال، این فقط در صورتی قابل‌اعمال است که ارزش پاداش صرفاً به حالت هدف بستگی داشته باشد که همیشه این‌طور نیست.

دومین موردی که ما به مدل اضافه می‌کنیم ضریب تخفیف[1] γ (گاما) است که یک عدد واحد از ۰ تا ۱ (شامل) است. معنای این پس از تعریف ویژگی‌های اضافی فرایند پاداش مارکوف توضیح داده می‌شود.

همان‌طور که به یاد دارید، ما زنجیره‌ای از انتقال حلت را در یک فرایند مارکوف مشـــاهده می‌کنیم. این مورد همچنان در مورد فرایند پاداش مارکوف صـادق اسـت، اما برای هر انتقال، ما مقدار اضـافی خود یعنی پاداش را داریم؛ بنابراین اکنون، همه مشاهدات ما دارای ارزش پاداشی هستند که به هر انتقال سیستم متصل است.

برای هر قسمت، **بازده**[2] در زمان t را به‌عنوان کمیت تعریف می‌کنیم:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

برای هر نقطه زمانی،**بازده** را به‌عنوان مجموع پاداش‌های بعدی محاســـبه می‌کنیم، اما پاداش‌های دورتر در ضـــریب تخفیفγ افزایش‌یافته به توان تعداد قدم[3]هایی که از نقطه شـــروع در t فاصـــله داریم(همان k) ضـــرب می‌شوند. ضریب تخفیف به معنای آینده‌نگری عامل است. اگر گاما(γ) برابر با ۱ باشد، بازده، G_t، فقط برابر است با مجموع تمام پاداش‌های بعدی و مربوط به عاملی است که هر پاداش بعدی را کاملاً مشاهده می‌کند. اگر گاما برابر با ۰ باشـــد، G_t فقط یک پاداش فوری و بدون هیچ حالت بعدی خواهد بود و با کوته بینی[4](عدم مشـــاهده) مطلق مطابقت دارد.

این مقادیر مفرط[5] فقط در موارد گوشه‌ای مفید هستند و اغلب اوقات، گاما روی چیزی دراین‌بین تنظیم می‌شود، مانند ۰.۹ یا ۰.۹۹. در این مقادیر، ما به پاداش‌های آینده نگاه خواهیم کرد، اما نه خیلی دور. مقدار $\gamma = 1$ ممکن است در موقعیت‌های قسمت‌های محدود کوتاه، قابل استفاده باشد. پارامتر گاما در یادگیری تقویتی مهم است و

[1] discount factor
[2] return
[3] steps
[4] short-sightedness
[5] extreme

در حال حاضـــر، در مورد آن به‌عنوان معیاری برای تخمین بازده آینده در نظر بگیرید. هر چه به عدد ۱ نزدیکتر باشد، قدم های بیشتری را در پیش خواهیم داشت.

این مقدار بازده در عمل چندان مفید نیســـت، زیرا برای هر زنجیره خاصـــی که از فرایند پاداش مارکوف خود مشـاهده کردیم، تعریف شـده اسـت، بنابراین می‌تواند به طور گسـترده‌ای متفاوت باشـد، حتی برای همان حالت. بااین‌حال، اگر ما امید ریاضـــی[1] بازده را برای هر حالت محاســـبه کنیم (با میانگین تعداد زیادی زنجیره)، مقدار بسیار مفیدتری به دست خواهیم آورد که به آن **مقدار(ارزش) حالت** می‌گویند.

$$V(s) = \mathbb{E}[G|\ S_t = s]$$

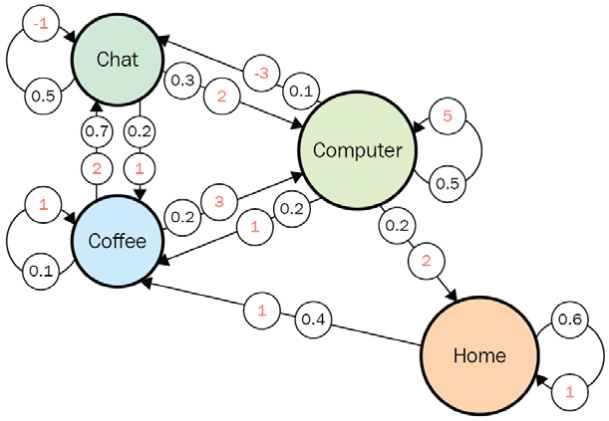
برای هر حللت، s، مقدار V(s)، میانگین (مورد انتظار) بازدهی اســـت که با دنبال‌کردن فرایند پاداش مارکوف به دست می‌آوریم.

برای نشـــان‌دادن این موارد نظری در عمل، بیایید فرایند کارمند اداره را با پاداش گســـترش دهیم و آن را به **فرایند پاداش** تبدیل کنیم. مقادیر پاداش ما به شرح زیر خواهد بود:

- **خانه ← خانه: ۱** (چون خانه بودن خوب است)
- **خانه ← قهوه: ۱**
- **کامپیوتر ← کامپیوتر: ۵** (سخت کار کردن چیز خوبی است)
- **کامپیوتر ← گفتگو: ۳-** (پرت شدن حواس خوب نیست)
- **گفتگو ← کامپیوتر: ۲**
- **کامپیوتر ← قهوه: ۱**
- **قهوه ← کامپیوتر: ۳**
- **قهوه ← قهوه: ۱**
- **قهوه ← گفتگو: ۲**
- **گفتگو ← قهوه: ۱**
- **گفتگو ← گفتگو: ۱-** (مکالمات طولانی خسته کننده می‌شود)

[1] mathematical expectation

نموداری از این مقادیر در اینجا نشان‌داده‌شده است:



شکل ۸-نمودار انتقال حالت با احتمالات انتقال (مشکی) و پاداش (روشن)

بیایید به پارامتر گامای خود برگردیم و در مورد مقادیر(ارزش) حلت‌هایی با مقادیر مختلف گاما فکر کنیم. ما با یک حالت ساده شروع می‌کنیم: گاما = ۰. چگونه مقادیر حالت‌ها را در اینجا محاسبه می‌کنید؟ برای پاسخ به این سوال، بیایید حالت خود را در **گفتگو** درست کنیم. انتقال بعدی چه می‌تواند باشد؟ پاسخ این است که بستگی به شانس دارد. باتوجه‌به ماتریس انتقال ما برای فرایند مارکوف، احتمال ۵۰ درصد این است که حالت بعدی دوباره **گفتگو**، ۲۰ درصد **قهوه** و ۳۰ درصد **رایانه** باشد. وقتی گاما = ۰ باشد، بازده ما فقط با مقداری از حالت فوری بعدی برابر است؛ بنابراین، اگر بخواهیم مقدار حالت **گفتگو** را محاسبه کنیم، باید همه مقادیر انتقال را جمع کنیم و آن را در احتمالات آنها ضرب کنیم:

$$V(\text{گفتگو}) = -1 \times 0.5 + 2 \times 0.3 + 1 \times 0.2 = 0.3$$
$$V(\text{قهوه}) = 2 \times 0.7 + 1 \times 0.1 + 3 \times 0.2 = 2.1$$
$$V(\text{خانه}) = 1 \times 0.6 + 1 \times 0.4 = 1.0$$
$$V(\text{رایانه}) = 5 \times 0.5 + (-3) \times 0.1 + 1 \times 0.2 + 2 \times 0.2 = 2.8$$

بنابراین، رایانه باارزش‌ترین حالتی است که می‌توان در آن قرار گرفت (اگر فقط به پاداش فوری اهمیت دهیم) که جای تعجب نیست؛ زیرا رایانه← **رایانه** مکرر است، پاداش زیادی دارد و نسبت وقفه‌ها خیلی زیاد نیست.

اکنون یک سوال پیچیده‌تر، ارزش زمانی که گاما = ۱ است چیست؟ در این مورد با دقت فکر کنید. پاسخ این است که مقدار برای همه حالت‌ها بی‌نهایت است. نمودار ما شامل حالات فرورفتگی (حالت‌های بدون انتقال خروجی) نیست، و وقتی تخفیف ما برابر با ۱ باشد، ما به تعداد بالقوه بی‌نهایتی از انتقال‌ها در آینده اهمیت

می‌دهیم. همان‌طور که در مورد گاما = ۰ مشاهده کردید، تمام مقادیر ما در کوتاه مدت مثبت هستند، بنابراین مجموع تعداد نامتناهی مقادیر مثبت، صرف‌نظر از حالت شروع، یک مقدار بی‌نهایت به ما می‌دهد.

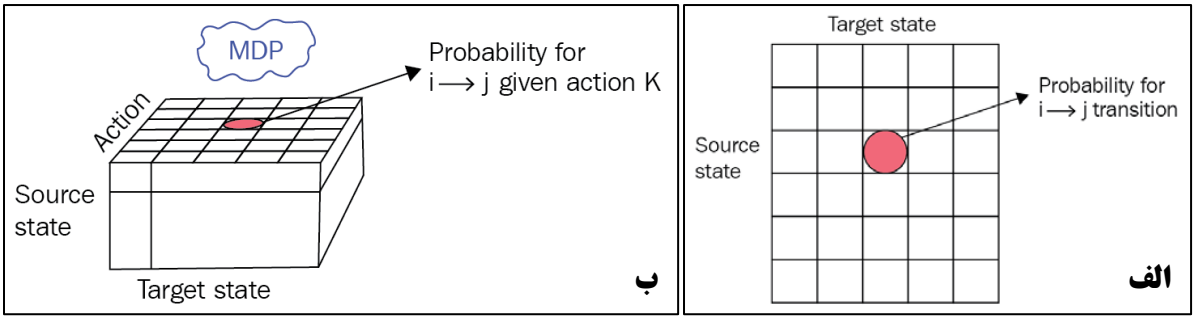
این نتیجه بی‌نهایت یکی از دلایلی را به ما نشان می‌دهد که به‌جای جمع‌کردن تمام پاداش‌های آینده، گاما را در فرایند پاداش مارکوف وارد کنیم. در بیشتر موارد، فرایند می‌تواند بی‌نهایت (زیاد) انتقال داشته باشد. ازآنجایی‌که پرداختن به مقادیر نامتناهی چندان عملی نیست، می‌خواهیم افقی را که مقادیر را برای آن محاسبه می‌کنیم محدود کنیم. گامای با مقدار کمتر از ۱ چنین محدودیتی را فراهم می‌کند. از طرف دیگر، اگر با محیط‌های افق محدود سر و کار دارید (مثلاً بازی tic-tac-toe که حداکثر با ۹ مرحله محدود شده است)، استفاده از گاما = ۱ خوب است.

در حال حاضر، اجازه دهید لایه دیگری از پیچیدگی را در اطراف فرایندهای پاداش مارکوف خود قرار دهیم و آخرین قطعه گمشده را معرفی کنیم: اقدامات.

۳-۶-۱- افزودن اقدامات

ممکن است در حال حاضر ایده‌هایی در مورد چگونگی گسترش فرایند پاداش مارکوف برای شامل اقدامات داشته باشید. ابتدا باید مجموعه‌ای از اعمال (A) را اضافه کنیم که باید متناهی باشد. این **فضای اقدامات** عامل ماست. ثانیاً، ما باید ماتریس انتقال خود را با اقدامات شرطی کنیم که اساساً به این معنی است که ماتریس ما به یک بعد اقدام اضافی نیاز دارد که آن را به یک مکعب تبدیل می‌کند.

اگر به‌خاطر داشته باشید، در مورد فرایند مارکوف و فرایندهای مارکوف، ماتریس انتقال شکل مربعی داشت(شکل ۹) که حالت مبدأ در ردیف‌ها و حالت هدف در ستون‌ها قرار داشت؛ بنابراین، هر ردیف i، حاوی لیستی از احتمالات برای پرش به هر حالت بود.



شکل ۹-(الف) ماتریس انتقال به شکل مربع و (ب) احتمالات انتقال برای فرایند تصمیم‌گیری مارکوف

اکنون عامل دیگر به‌صورت غیرفعال انتقال حالت را مشاهده نمی‌کند، بلکه می‌تواند به طور فعال اقدامی را برای انجام در هر انتقال حالت انتخاب کند؛ بنابراین، برای هر حالت منبع، ما فهرستی از اعداد نداریم، اما ماتریسی داریم که در آن بعد عمق شامل اقداماتی است که عامل می‌تواند انجام دهد و بعد دیگر آن چیزی است که سیستم حالت هدف پس از انجام اقدامات توسط عامل به آن خواهد پرید. نمودار زیر جدول انتقال جدید ما را نشان می‌دهد که به یک مکعب با حالت مبدأ به‌عنوان بعد ارتفاع (i) و حالت هدف به‌عنوان عرض (j) تبدیل شد و عملی که عامل می‌تواند به‌عنوان عمق (k) جدول انتقال انجام دهد.

بنابراین، به‌طورکلی با انتخاب یک عمل عامل می‌تواند بر احتمالات حالت‌های هدف تأثیر بگذارد که این یک توانایی مفید است.

در نهایت، برای تبدیل فرایند پاداش مارکوف به یک فرایند تصمیم‌گیری مارکوف، باید اقداماتی را به ماتریس پاداش خود اضافه کنیم، همان‌طور که با ماتریس انتقال انجام دادیم. ماتریس پاداش ما نه‌تنها به حالت بلکه به عمل نیز بستگی دارد. به‌عبارت‌دیگر، پاداشی که عامل به دست می‌آورد، نه‌تنها به حالتی که در نهایت به آن می‌رسد، بلکه به عملی که منجر به این حالت می‌شود نیز بستگی دارد.

۴-۶-۱- خط‌مشی (سیاست)

ممکن است به یاد داشته باشید که هدف اصلی عامل در یادگیری تقویتی این است که تاحدامکان بازدهی بیشتری را جمع‌آوری کند؛ بنابراین، دوباره، سیاست‌های مختلف می‌توانند مقادیر متفاوتی بازدهی را به ما بدهند که پیداکردن یک سیاست خوب را مهم می‌کند. به همین دلیل است که مفهوم سیاست مهم است.

به طور رسمی، خط‌مشی به‌عنوان توزیع احتمال بر روی اقدامات برای هر حالت ممکن تعریف می‌شود:

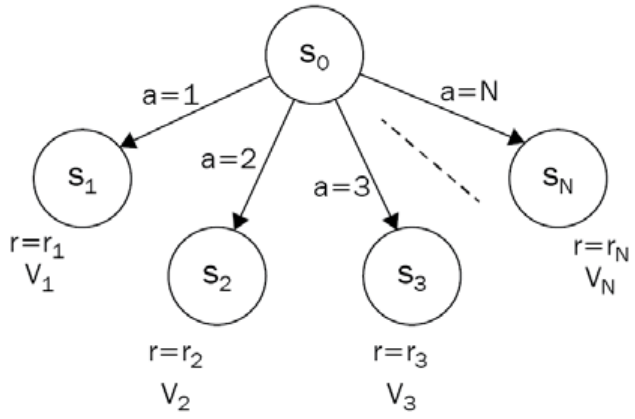
$$\pi(a \mid s) = P[A_t = a \mid S_t = s]$$

این به‌عنوان احتمال تعریف می‌شود و نه به‌عنوان یک اقدام مشخص برای واردکردن تصادفی بودن به رفتار(رفتار تصادفی) یک عامل. خط‌مشی قطعی یک مورد خاص از احتمال است که احتمال عمل موردنیاز برابر با ۱ است.

مفهوم مفید دیگر این است که اگر خط‌مشی ما ثابت باشد و تغییر نکند، فرایند تصمیم‌گیری مارکوف ما به یک فرایند پاداش مارکوف تبدیل می‌شود، زیرا می‌توانیم ماتریس‌های انتقال و پاداش را با احتمالات یک سیاست کاهش دهیم و از ابعاد عمل خلاص شویم.

۵-۶-۱- معادله بهینگی بلمن

برای توضیح معادله بلمن[1] بهتر است کمی انتزاعی پیش برویم(شکل ۱۰). بیایید با یک مورد قطعی[2] شروع کنیم، زمانی که همه اقدامات ما یک نتیجه ۱۰۰٪ تضمین شده دارند. تصور کنید که عامل ما حالت s_0 را مشاهده می‌کند و N عمل در دسترس دارد. هر عملی به حالت دیگری منتهی می‌شود، $s_1 \dots s_N$، با پاداش مربوطه، $r_1 \dots r_N$. همچنین، فرض کنید که مقادیر V_i، همه حالت‌های متصل به حالت s_0 را می‌دانیم. بهترین اقدامی که عامل می‌تواند در چنین حالتی انجام دهد خواهد بود؟



شکل ۱۰ -یک محیط انتزاعی با N حالت قابل‌دسترسی از حالت اولیه

اگر عمل مشخص، a_i را انتخاب کنیم و مقدار داده شده به این عمل را محاسبه کنیم، مقدار $V_0(a=a_i) = r_i + V_i$ خواهد بود. بنابراین، برای انتخاب بهترین اقدام ممکن، عامل باید مقادیر حاصل از هر اقدام را محاسبه کند و حداکثر نتیجه ممکن را انتخاب کند. به‌عبارت‌دیگر، $V_0 = max_{a \in 1 \dots N}(r_a + V_a)$. اگر از ضریب تخفیف γ استفاده می‌کنیم، باید مقدار حالت بعدی را در گاما ضرب کنیم:

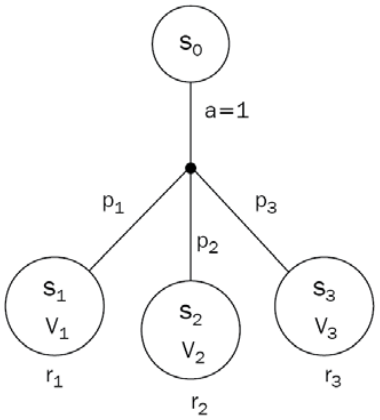
$$V_0 = max_{a \in 1 \dots N}(r_a + \gamma V_a)$$

این ممکن است بسیار شبیه به روش‌های حریصانه[3] باشد و در واقع همین‌طور است. بااین‌حال، یک تفاوت وجود دارد: در این مورد وقتی حریصانه عمل می‌کنیم، فقط به پاداش آنی عمل نگاه نمی‌کنیم، بلکه به پاداش فوری به‌اضافه ارزش بلندمدت دولت نگاه می‌کنیم.

[1] Bellman equation
[2] deterministic
[3] greedy

بلمن ثلبت کرد که با این بســـط، رفتار ما بهترین نتیجه ممکن را خواهد گرفت. به‌عبارت‌دیگر بهینه خواهد بود؛ بنابراین، معادله قبلی معادله ارزش بلمن نامیده می‌شود (برای یک حالت قطعی).

گســـترش این ایده برای یک مورد تصـــادفی[1]، زمانی که اقدامات ما این شـــانس را دارند که به حالت‌های مختلف ختم شـــوند، چندان پیچیده نیســـت. کاری که ما باید انجام دهیم این اســـت که به‌جای اینکه فقط مقدار حالت بعدی را بگیریم، مقدار مورد انتظار را برای هر اقدام محاسبه کنیم. برای نشان‌دادن این موضوع، بیایید یک اقدام واحد را که از حالت s_0 در دسترس است، با سه نتیجه ممکن در نظر بگیریم.



شکل ۱۱- مثالی از انتقال از حالت در یک مورد تصادفی

در اینجا ما یک عمل داریم که می‌تواند منجر به سه حالت مختلف با احتمالات متفاوت شود. با احتمال p_1، عمل می‌تواند به حالت s_1، با p_2 در حالت s_2، و با p_3 در حالت s_3 ختم شـود (البته $p_1 + p_2 + p_3 = 1$). هر حالت هدف پاداش مخصـوص به خود را دارد (r_1، r_2 یا r_3). برای محاســبه مقدار مورد انتظار پس از صـدور عمل ۱، باید همه مقادیر را ضرب در احتمالات آنها جمع کنیم:

$$V_0(a=1) = p_1(r_1 + \gamma V_1) + p_2(r_2 + \gamma V_2) + p_3(r_3 + \gamma V_3)$$

یا به‌صورت رسمی‌تر:

$$V_0(a) = \mathbb{E}_{s\sim S}[r_{s,a} + \gamma V_s] = \sum_{s\in S} p_{a,0\to s}(r_{s,a} + \gamma V_s)$$

[1] stochastic

با ترکیب معادله بلمن، برای یک حالت قطعی، با مقداری برای اقدامات تصـــادفی، معادله بهینگی بلمن برای یک حالت کلی به دست می‌آید:

$$V_0 = \max_{a \in A} \mathbb{E}_{s \sim S} \left[r_{s,a} + \gamma V_s \right] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s} \left(r_{s,a} + \gamma V_s \right)$$

توجه داشته باشید که $p_{a,\mathrm{i}\to\mathrm{j}}$ به معنای احتمال عمل a است که در حالت i صادر شده و به حالت j ختم می‌شود.

مقدار بهینه حللت برابر اســـت با عمل که حداکثر پاداش احتمالی فوری مورد انتظار را به ما می‌دهد، به‌علاوه پاداش بلندمدت با تخفیف برای حالت بعدی. همچنین ممکن است متوجه شوید که این تعریف بازگشتی است: مقدار(ارزش) حالت، از طریق مقادیر حالت‌های بلافاصله قابل‌دسترسی تعریف می‌شود.

این ارزش‌ها نه‌تنها بهترین پاداشی را که می‌توانیم به دست آوریم، به ما می‌دهند، بلکه اساساً خط‌مشی بهینه را برای به‌دست‌آوردن آن پاداش به ما می‌دهند: اگر عامل ما ارزش هر حالت را بداند، به طور خودکار می‌داند چگونه همه این پاداش را جمع‌آوری کند. به لطف اثبات بهینه‌ســازی بلمن، در هر حالتی که عامل به آن می‌رســد، باید اقدامی را با حداکثر پاداش مورد انتظار انتخاب کند که مجموع پاداش فوری و پاداش بلندمدت با تخفیف یک‌مرحله‌ای است؛ بنابراین، دانستن این مقادیر واقعاً مفید است.

۲- الگوریتم‌های پایه یادگیری تقویتی

در این بخش به بررسی الگوریتم‌های پایه در یادگیری تقویتی خواهیم پرداخت که محیط‌های مختلف بسیار پر کاربرد هستند و الگوریتم‌های پیشرفته بر پایه آن‌ها شکل‌گرفته‌اند. چندین الگوریتم و روش تصمیم‌گیری در یادگیری تقویتی وجود دارد که تفاوت آن‌ها عمدتاً به دلیل استراتژی‌های مختلفی است که برای کشف محیط خود استفاده می‌کنند. در ادامه به برخی از پرکاربردترین روش‌های تصمیم‌گیری در یادگیری تقویتی اشاره خواهیم کرد.

روش‌های مبتنی بر مدل:

- ساخت مدلی از محیط
- برنامه‌ریزی[1] با استفاده از مدل (مثلاً با نگاه پیش رو[2]).

روش‌های مبتنی بر ارزش:

- تخمین تابع مقدار بهینه $Q*(s, a)$
- این حداکثر مقدار قابل دستیابی تحت هر سیاستی است

روش‌های مبتنی بر سیاست:

- مستقیماً سیاست بهینه $\pi*$ را جستجو می‌کند
- این سیاست دستیابی به حداکثر پاداش آینده است

روش‌های تقریب توابع:

- استفاده از شبکه‌های عصبی عمیق برای تقریب توابع ارزش
- استفاده از توابع پایه برای تقریب توابع

۲-۱- روش‌های مبتنی بر مدل (برنامه‌نویسی پویا)

یادگیری تقویتی مبتنی بر مدل مانند پسرعموی باهوش خانواده یادگیری تقویتی است؛ جایی که عامل به شکل کورکورانه به محیط واکنش نشان نمی‌دهد. در عوض با ایجاد نسخه کوچکی از خود تلاش دارد رویدادهای آینده را پیش‌بینی کند. این مدل حرکات مختلف را امتحان می‌کند تا ببیند در آینده چه اتفاقی می‌افتد. در این شیوه،

[1] Plan
[2] lookahead

عامل می‌تواند سناریوهایی را در مدل خود اجرا کند، بدون آنکه نتیجه منفی بگیرد؛ درست مانند چیدن استراتژی حرکت در بازی شطرنج پیش از لمس مهره.

اصطلاح برنامه‌نویسی پویا[1] به مجموعه‌ای از الگوریتم‌ها اشاره دارد که می‌توانند برای محاسبه سیاست‌های بهینه باتوجه‌به یک مدل کامل از محیط به‌عنوان فرایند تصمیم مارکوف استفاده شوند. الگوریتم‌های برنامه‌نویسی پویا کلاسیک هم به دلیل فرض یک مدل کامل و هم به دلیل هزینه محاسباتی زیادشان در یادگیری تقویتی کاربرد محدودی دارند، اما از نظر تئوری همچنان مهم هستند. برنامه‌نویسی پویا یک‌پایه اساسی برای درک اکثر روش‌های ارائه شده در یادگیری تقویتی را فراهم می‌کند. در واقع، همه این روش‌ها را می‌توان به‌عنوان تلاش‌هایی برای دستیابی به نتایج مشابه برنامه‌نویسی پویا، تنها با محاسبات کمتر و بدون فرض یک مدل کامل از محیط مشاهده کرد.

- ایده کلیدی DP استفاده از توابع ارزش برای سازماندهی و ساختار جستجوی سیاست‌های خوب است
- هنگامی که توابع مقدار بهینه را یافتیم که معادلات بهینه بلمن را برآورده می‌کند، به‌راحتی می‌توانیم سیاست‌های بهینه را به دست. آوریم:

$$V^*(s) = \max_a \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right] = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right].$$

- ارزش اقدام a در حالت s تحت یک خط‌مشی π ، بازده مورد انتظار هنگام شروع در s، انجام اقدام a و دنبال‌کردن π پس از آن است.

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a \right] = \sum_{s'} P^a_{ss'} \left[\mathcal{R}_{ss'}{}^a + \gamma \max_{a'} Q^*(s', a') \right].$$

که در آن $V^*(s)$: تابع مقدار بهینه در حالت s. s: متغیر حالت. a: متغیر اقدام/کنترل. r_{t+1}: پاداش در زمان t+1. γ: ضریب تخفیف. s_t: حالت در زمان t. a_t: اقدام در زمان t. E: عملگر امید ریاضی. $P_{ss'}{}^a$: احتمال انتقال از حالت s به حالت s تحت عمل a. $R_{ss'}{}^a$: پاداش مرتبط با انتقال از حالت s به حالت s تحت عمل a.

تکرار سیاست:

- تکرار خط‌مشی یک فرایند تکراری است

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots\dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

[1] dynamic programming

- تکرار خط‌مشی دو مرحله دارد: ارزیابی و بهبود سیاست.
- در ارزیابی خط‌مشی، ما توابع مقدار حالت یا حالت-عمل را محاسبه می‌کنیم:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\pi}\left[R_t \mid s_t = s\right] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right] = \sum_{\pi}\pi(s,a)\sum_{s'}P_{ss'}{}^{a}[R_{ss'}{}^{a} + \gamma V^{\pi}(s')]$$

- در بهبود خط‌مشی، سیاست را تغییر می‌دهیم تا خط‌مشی بهتری به دست آوریم

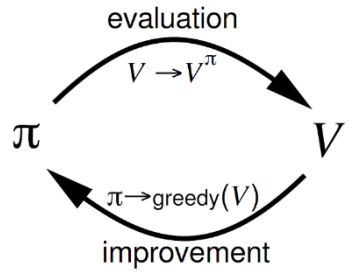
$$\pi'(s) = \operatorname*{argmax}_a Q^{\pi}(s,a) = \operatorname*{argmax}_a \sum_{s'}P_{ss'}{}^{a}[R_{ss'}{}^{a} + \gamma V^{\pi}(s')].$$

ارزش و تکرار خط‌مشی تعمیم‌یافته:

- در تکرار ارزش داریم:

$$V_{k+1}(s) = \max_a \mathbb{E}\left[r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\right] = \max_a \sum_{s'}\mathcal{P}_{ss'}{}^{a}\left[\mathcal{R}_{ss'}{}^{a} + \gamma V_k(s')\right]$$

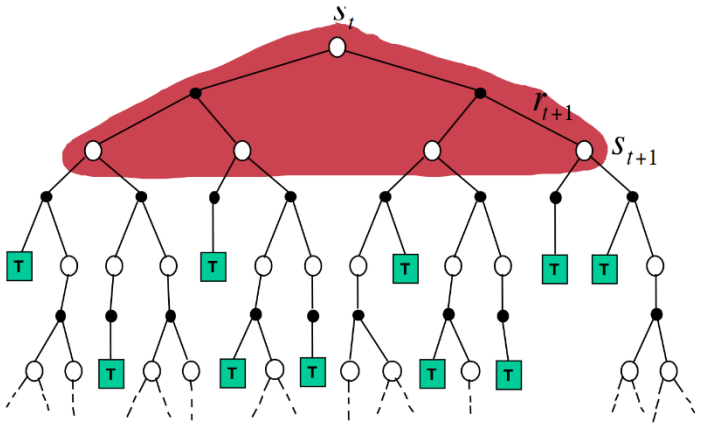
- تکرار سیاست تعمیم‌یافته



شکل ۱۲ - چرخه ارزیابی و بهبود در برنامه‌نویسی پویا

نمودار پشتیبان[1]:

$$V(S_t) \leq \mathop{\mathbb{E}}_{\pi}\left[R_{t+1} + \gamma V(S_{t+1})\right]$$



شکل ۱۳- نمودار پشتیبان روش برنامه‌نویسی پویا

۲-۲- روش‌های مبتنی بر ارزش (مونت‌کارلو و یادگیری تفاوت زمانی)

در این روش، تمرکز بر یادگیری تابع مقدار بهینه است که با Q^* یا V^* نشان داده می‌شود. تابع ارزش پاداش‌های تجمعی آینده مورد انتظار را برای قرارگرفتن در یک حالت معین و پیروی از سیاست فعلی پس از آن تخمین می‌زند.

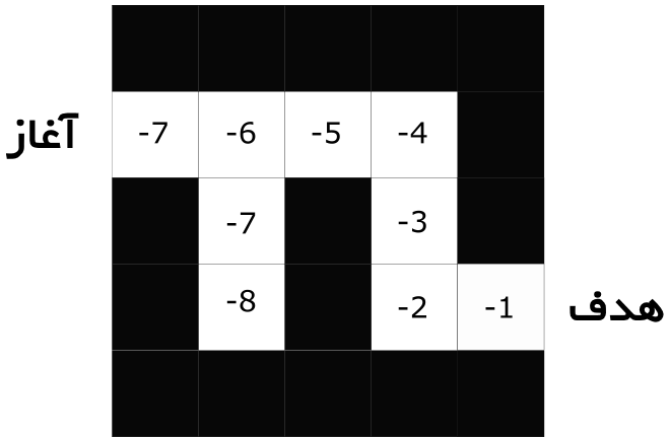
در رویکرد مبتنی بر ارزش، هدف بهینه‌سازی تابع ارزش V(s) است. تابع ارزش، تابعی است که پاداش بیشینه آینده را مشخص می‌کند که عامل در هر حالت دریافت می‌کند. ارزش هر حالت برابر است باارزش کل پاداشی که عامل می‌تواند انتظار داشته باشد در آینده با آغاز از آن حالت جمع‌آوری کند.

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$$

با توجه به حالت — پاداش تنزیل یافته — مورد انتظار

عامل از این تابع ارزش برای انتخاب آنکه کدام حالت در هر گام انتخاب شود، استفاده می‌کند. عامل حالتی با بیشترین ارزش را انتخاب می‌کند.

[1] نمودار پشتیبان(backup diagram) یک نمایش بصری است که در یادگیری تقویتی برای نشان دادن نحوه انتشار یا "پشتیبان گیری" مقادیر از حالت های آینده به حالت فعلی استفاده می شود.



شکل ۱۴-محاسبه ارزش همه حالت‌ها در مثال هزارتو

در مثال هزارتو، در هر گام بیشترین ارزش، یعنی ۷-، ۶- و سپس ۵- (و به همین ترتیب) برای رسیدن به هدف اتخاذ می‌شود.

- این روش‌ها به طور ضمنی بر تابع سیاست تکیه دارند.
- این روش‌ها ابتدا تابع مقدار Q(s, a) را یاد می‌گیرند.
- سپس خط‌مشی π(s, a) را از Q(s, a) استنتاج کنید.
- نمونه‌ها:
  ۱. روش‌های مونت‌کارلو[1]
  ۲. روش‌های تفاوت زمانی[2]
  ۳. Q-learning
  ۴. SARSA

۱-۲-۲- روش‌های مونت‌کارلو

روش مونت‌کارلو برخلاف روش برنامه‌ریزی پویا که نیازمند اطلاعات کامل در مورد توزیع احتمالات همه انتقال‌های ممکن بود؛ تنها به نمونه‌ای از توزیع‌های احتمال بسنده می‌کند. به بیانی دیگر، در روش مونت‌کارلو شناخت کامل محیط لازم نیست و با برقراری تعامل واقعی یا شبیه‌سازی شده با یک محیط می‌توان به توالی نمونه‌ای از حالت‌ها، اقدامات و پاداش‌های دست‌یافت. به همین دلیل است که در این روش، پاداش‌ها در انتهای دوره حساب می‌شود تا بتوان از دانش کسب شده برای دوره جدید استفاده نمود.

[1] Monte Carlo
[2] Temporal-difference

- روش‌های مونت‌کارلو به طور مستقیم از قسمت(اپیزود) های تجربه یاد می‌گیرند.
- مونت‌کارلو بدون مدل است: بدون آگاهی از انتقال / پاداش MDP
- مونت‌کارلو از قسمت‌های کامل درس می‌گیرد
- مونت‌کارلو از ساده‌ترین ایده ممکن استفاده می‌کند: ارزش = میانگین بازده
- هدف مونت‌کارلو: یادگیری $V\pi$ از قسمت‌های تجربه تحت خط‌مشی π

$$[S_1 \xrightarrow[R_1]{\alpha_1} S_2 \xrightarrow[R_2]{\alpha_2} S_3 \xrightarrow[R_3]{\alpha_3} S_4 \cdots \xrightarrow[R_{k-1}]{\alpha_{k-1}} S_k]$$

- بازده کل پاداش با تخفیف است:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- تابع مقدار بازده مورد انتظار است:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | \, S_t = s \right]$$

- ارزیابی خط‌مشی مونت‌کارلو از میانگین بازده تجربی به‌جای بازده مورد انتظار استفاده می‌کند.

ارزیابی سیاست هر بازدید از مونت‌کارلو:

- برای ارزیابی حالت s
- **هر** مرحله زمانی t که حالت s را در یک قسمت مشاهده می‌کند، شمارنده افزایش می‌یابد

$$N(s) \leftarrow N(s) + 1$$

- بازده کل را افزایش می‌دهد:

$$S(s) \leftarrow S(s) + G_t$$

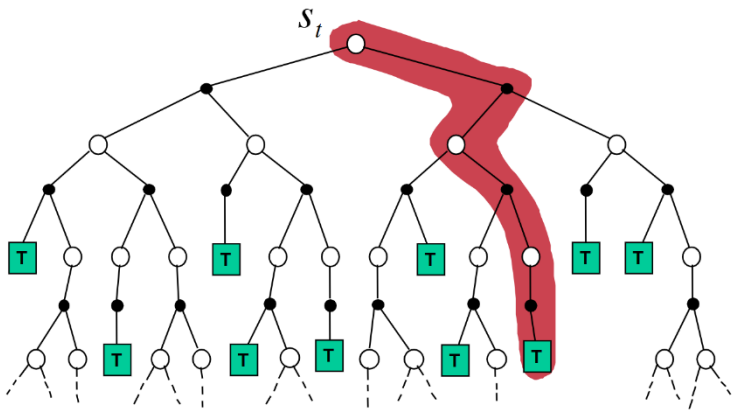
- ارزش با بازده متوسط تخمین زده می‌شود

$$V(s) = \frac{S(s)}{N(s)}$$

- طبق قانون اعداد زیاد، $V(s) \to V_\pi(s)$ به‌عنوان $N(s) \to \infty$

نمودار پشتیبان:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



شکل ۱۵- نمودار پشتیبان روش مونت‌کارلو

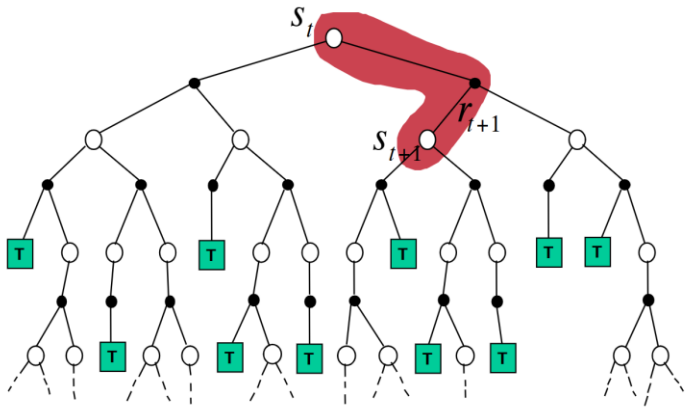
۲-۲-۲- روش‌های تفاوت زمانی

روش‌های تفاوت زمانی (TD) یک نوع روش پیش‌بینی است که از ایده‌های مونت‌کارلو و برنامه‌نویسی پویا برای حل مسائل تقویتی استفاده می‌کند. روش‌های تفاوت زمانی مانند روش‌های مونت‌کارلو، مستقیماً از تجربه خام بدون نیاز به مدلی پویا از محیط، می‌آموزند و مانند برنامه‌ریزی پویا، تخمین‌ها را تا حدی بر اساس سایر تخمین‌های آموخته شده به‌روز می‌کنند، بدون اینکه منتظر نتیجه نهایی باشند.

- تفاوت زمانی ترکیبی از ایده‌های مونت‌کارلو و ایده‌های برنامه‌نویسی پویا است.
- مانند روش‌های مونت‌کارلو، روش‌های تفاوت زمانی می‌توانند مستقیماً از تجربه خام بدون مدلی از پویایی محیط بیاموزند.
- مانند برنامه‌نویسی پویا، روش‌های تفاوت زمانی تخمین‌ها را تا حدی بر اساس سایر تخمین‌های آموخته شده به‌روزرسانی می‌کنند، بدون اینکه منتظر نتیجه نهایی باشند (آنها راه‌اندازی[1] می‌کنند).
- روش‌های مونت‌کارلو منتظر می‌مانند تا بازده پس از بازدید مشخص شود، سپس از آن بازده به‌عنوان یک هدف برای $V(s_t)$ استفاده می‌کند درحالی‌که روش‌های تفاوت زمانی فقط تا مرحله زمانی بعدی باید منتظر بمانند.
- ساده‌ترین روش تفاوت زمانی که با نام TD(۰) شناخته می‌شود، است

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

[1] bootstrap



شکل ۱۶- بردار پشتیبان روش تفاوت زمانی

الگوریتم TD(0):

```
Initialize V(s) arbitrarily, π to the policy to be evaluated

Repeat (for each episode):
    Initialize s

    Repeat (for each step of episode):
        a ← action given by π for s
        Take action a; observe reward, r, and next state, s'
        V(s) ← V(s) + α [r + γV(s') - V(s)]
        s ← s'
    until s is terminal
```

۳-۲-۲- SARSA

اکنون به اســـتفاده از روش‌های پیش‌بینی TD برای مســـئله کنترل می‌پردازیم. طبق معمول، ما از الگوی تکرار خط‌مشـــی تعمیم‌یافته[1] پیروی می‌کنیم، فقط این بار از روش‌های TD برای بخش ارزیابی یا پیش‌بینی اســـتفاده می‌کنیم. همانند روش‌های مونت‌کارلو، ما با نیاز به موازنه اکتشاف و بهره‌برداری مواجه هستیم، و دوباره رویکردها به دو دسته اصلی تقسیم می‌شوند: مبتنی بر سیاست[2] و خارج از سیاست[3]. در این بخش ما یک روش کنترل TD مبتنی بر سیاست را ارائه می‌کنیم.

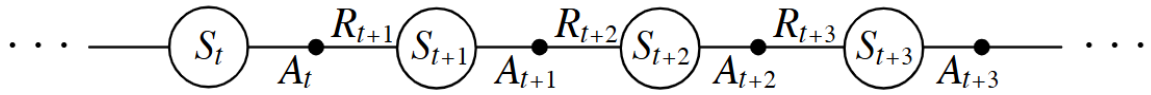
اولین گام این است که به‌جای یک تابع ارزش-حالت، یک تابع ارزش-عمل را یاد بگیرید. به طور خاص، برای یک روش مبتنی بر ســـیاســـت، باید $q_\pi(s, a)$ را برای خط‌مشـــی رفتار فعلی π و برای همه حالت‌ها s و اقدامات a

[1] generalized policy iteration
[2] on-policy
[3] off-policy

تخمین بزنیم. این را می‌توان با استفاده از همان روش TD که در زیر برای یادگیری v_π توضیح داده شد انجام داد:

$$v_\pi(s) \overset{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_t + 1 + \gamma G_{t+1} | S_t = s] = \mathbb{E}_\pi[R_t + 1 + \gamma v_\pi(S_{t+1}) | S_t = s]$$

به یاد بیاورید که یک قسمت(اپیزود) از یک توالی متناوب از حالت‌ها و جفت‌های حالت-عمل تشکیل شده است:



در بخش قبل انتقال از حالتی به حلت دیگر را در نظر گرفتیم و مقادیر حلت‌ها را یاد گرفتیم. اکنون انتقال از جفت حلت-عمل به جفت حلت-عمل را در نظر می‌گیریم و مقادیر جفت حلت-عمل را یاد می‌گیریم. به طور رسمی، این موارد یکسان هستند: هر دو زنجیره مارکوف با فرایند پاداش هستند. قضایایی که همگرایی مقادیر حالت را تحت TD(0) تضمین می‌کنند، در الگوریتم مربوطه برای مقادیر عمل نیز اعمال می‌شوند:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)].$$

Sarsa

این به‌روزرسانی پس از هر انتقال از حالت غیر پایانی S_t انجام می‌شود. اگر S_{t+1} ترمینال باشد، $Q(S_{t+1}, A_{t+1})$ به‌عنوان صفر تعریف می‌شود. این قانون از هر عنصر از پنج‌گانه رویدادها (S_t، A_t، R_{t+1}، S_{t+1}، A_{t+1}) استفاده می‌کند که انتقال از یک جفت حالت-عمل به جفت بعدی را تشکیل می‌دهند. این پنج‌گانه باعث ایجاد نام Sarsa(State-Action-Reward-State-Action) برای الگوریتم می‌شود. نمودار پشتیبان برای Sarsa در سمت چپ نشان‌داده‌شده است.

طراحی یک الگوریتم کنترل روی خط‌مشی بر اساس روش پیش‌بینی Sarsa ساده است. مانند همه روش‌های مبتنی بر سیاست، ما به طور مداوم q_π را برای خط‌مشی رفتاری π تخمین می‌زنیم، و درعین‌حال π را نسبت به q_π به سمت حریصانه[1] تغییر می‌دهیم. شکل کلی الگوریتم کنترل Sarsa در کادر صفحه بعد آورده شده است.

ویژگی‌های همگرایی[2] الگوریتم Sarsa به ماهیت وابستگی خط‌مشی به Q بستگی دارد. برای مثال، می‌توان از سیاست‌های «طمع آمیز(ε-greedy)» یا «نرم(ε-soft)» استفاده کرد. Sarsa با احتمال ۱، به یک خط‌مشی بهینه و تابع ارزش-عمل همگرا می‌شود تا زمانی که همه جفت‌های حالت-عمل بی‌نهایت بار بازدید شوند و خط‌مشی در حد خط‌مشی حریصانه همگرا شود.

[1] greediness
[2] convergence properties

الگوریتم Sarsa (کنترل TD مبتنی بر سیاست) برای تخمین $Q \approx q_*$:

```
Algorithm parameters: step size α ∈ (0, 1], small ε > 0
Initialize Q(s, a) for all s ∈ S+, a ∈ A(s), arbitrarily except that Q(terminal, ·) = 0

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        Q(S, A) ← Q(S, A) + α [R + γ Q(S', A') - Q(S, A)]
        S ← S'; A ← A';
    until S is terminal
```

۴-۲-۲- Q-learning

یکی از پیشرفت‌های اولیه در یادگیری تقویتی، توسعه یک الگوریتم کنترل TD خارج از سیاست(off-policy) معروف به یادگیری Q بود که به‌صورت زیر تعریف می‌شود:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

الگوریتم یادگیری Q شباهت زیادی با الگوریتم یادگیری SARSA دارد. تفاوت کلیدی این دو الگوریتم در این است که SARSA برخلاف الگوریتم یادگیری Q ، در دسته الگوریتم‌های مبتنی بر سیاست قرار می‌گیرد؛ بنابراین، الگوریتم SARSA مقدار Q-value را باتوجه‌به اقدامی که ناشی از سیاست فعلی است محاسبه می‌کند نه اقدام ناشی از سیاست حریصانه.

در این مورد، تابع ارزش-عمل آموخته شده، Q، به طور مستقیم به q_*، تابع ارزش-عمل بهینه، مستقل از سیاست مورداستفاده، تقریب می‌زند. این به طور چشمگیری تجزیه‌وتحلیل الگوریتم را ساده می‌کند و اثبات همگرایی[1] اولیه را فعال می‌کند. این خطمشی همچنان دارای اثری است که تعیین می‌کند کدام جفت‌های حالت-عمل بازدید و به‌روزرسانی می‌شوند. بااین‌حال، تنها چیزی که برای همگرایی صحیح موردنیاز است این است که همه جفت‌ها به‌روز شوند. این یک حداقل نیاز است به این معنا که هر روشی که تضمین شده است رفتار بهینه را در حالت کلی پیدا کند، باید آن را داشته باشد. با این فرض و نوعی از شرایط تقریب تصادفی معمول در دنباله پارامترهای اندازه-گام، نشان‌داده‌شده است که Q با احتمال ۱ به q_* همگرا می‌شود.

[1] convergence proofs

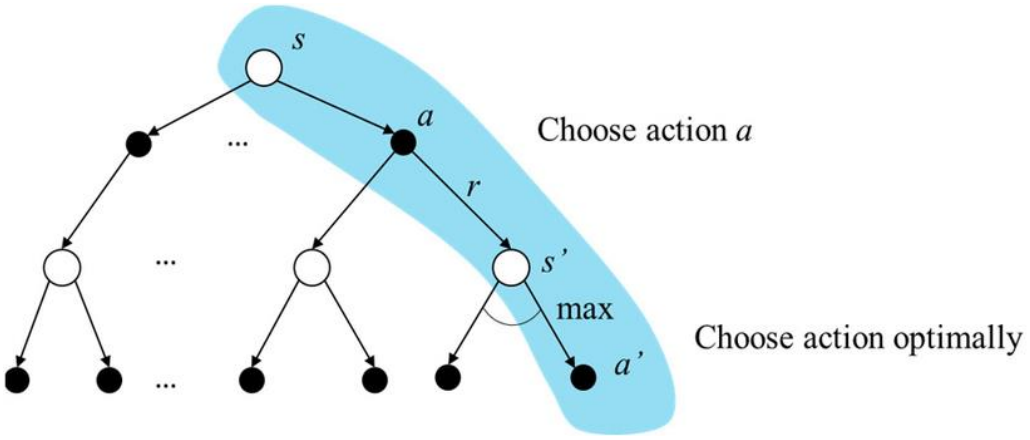
الگوریتم یادگیری Q (کنترل TD خارج از سیاست) برای تخمین $\pi \approx \pi_*$:

```
Algorithm parameters: step size α ∈ (0, 1], small ε > 0

Initialize Q(s, a), for all s ∈ S⁺, a ∈ A(s), arbitrarily except that
Q(terminal, ·) = 0

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    Q(S, A) ← Q(S, A) + α [R + γ max_a Q(S', a) - Q(S, A)]
    S ← S'
until S is terminal
```

در مورد نمودار پشـــتیبان برای یادگیری Q، قانون یادگیری Q یک جفت حالت-عمل را به‌روز می‌کند، بنابراین گره بالایی، ریشـه به‌روزرسـانی، باید یک گره اقدام کوچک و پر باشـد. به‌روزرسـانی نیز از گره‌های اقدام اسـت که بیش از همه آن اقدامات ممکن در حالت بعدی به حداکثر می‌رسد؛ بنابراین گره‌های پایینی نمودار پشـتیبان باید تمام این گره‌های عمل باشند(شکل ۱۷).



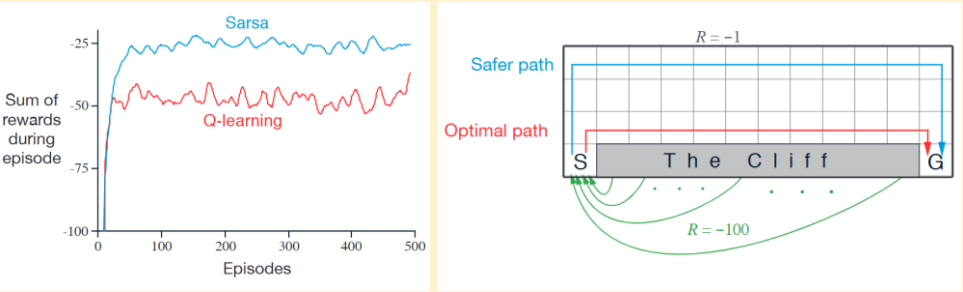
شکل ۱۷-بردار پشتیبان روش تفاوت زمانی

یکی از مزایای یادگیری خارج از سـیاسـت این اسـت که تابع مقدار بهینه با یک خط‌مشـی قطعی را می‌توان در حین پیروی از یک خط‌مشـی اکتشـافی تصـادفی یاد گرفت. اشـکالات یادگیری خارج از سـیاسـت این اسـت که عملکرد در طول یادگیری می‌تواند با نادیده‌گرفتن تأثیر اکتشـــاف به خطر بیفتد و یادگیری می‌تواند در صـــورت ترکیب با یک تقریب تابع، ناپایدار باشد.

مثال پیاده روی صخره(Cliff Walking): این مثال دنیای شبکه‌ای(gridworld) Sarsa و یادگیری Q را مقایسه می کند و تفاوت بین روش های مبتنی بر سیاست (Sarsa) و روش های خارج از سیاست (یادگیری Q) را برجسته می کند.

دنیای شبکه ای را که در زیر نشان داده شده است در نظر بگیرید. این یک کار اپیزودیک و بدون تخفیف استاندارد است، با حالت های شروع(start) و هدف(goal)، و اعمال معمولی که باعث حرکت به سمت بالا، پایین، راست و چپ می شود. پاداش در همه انتقال‌ها ۱- است به جز مواردی که در منطقه با علامت "The Cliff" مشخص شده است. قدم گذاشتن در این منطقه دارای پاداش ۱۰۰- است و عامل را فوراً به شروع باز می‌گرداند.

نمودار زیر عملکرد روش‌های Sarsa و Q-learning را با انتخاب عمل ε-greedy نشان می‌دهد، $\varepsilon = 0.1$. پس از یک دوره گذرا اولیه، Q-learning مقادیر سیاست بهینه را می آموزد، چیزی که درست در امتداد لبه صخره حرکت می کند. متأسفانه، به دلیل انتخاب اکشن ε-greedy، گاهی اوقات از صخره سقوط می کند. از سوی دیگر، سارسا، انتخاب عمل را در نظر می گیرد و مسیر طولانی تر اما امن تر را از طریق قسمت بالای شبکه یاد می گیرد. اگرچه Q-learning در واقع مقادیر خط مشی بهینه را می آموزد، عملکرد آنلاین آن بدتر از Sarsa است که خط مشی دورگرد را می آموزد. البته، اگر ε به تدریج کاهش یابد، هر دو روش به طور مجانبی به خط مشی بهینه همگرا می شوند.



شکل ۱۸-دنیای شبکه ای پیاده روی صخره(راست) و عملکرد روش های Sarsa و Q-learning

۳-۲- روش‌های مبتنی بر خط‌مشی(گرادیان خط‌مشی)

روش‌های یادگیری تقویتی مبتنی بر خط‌مشی[1](سیاست) به طور مستقیم خط‌مشی مدل را بهینه می‌کنند؛ درحالی‌که تکنیک‌های مبتنی بر ارزش به دنبال یافتن تابع ارزش بهینه هستند که به نوبه خود بهترین خط‌مشی را ارائه می‌دهد.

در روش‌های مبتنی بر خط‌مشی، قصد بهینه‌سازی تابع سیاست $\pi(s)$ بدون استفاده از تابع ارزش است. سیاست چیزی است که رفتار عامل را در یک زمان داده شده، تعیین می‌کند. عامل یک تابع سیاست را می‌آموزد. این امر به او کمک می‌کند تا هر حالت را به بهترین عمل ممکن نگاشت کند.

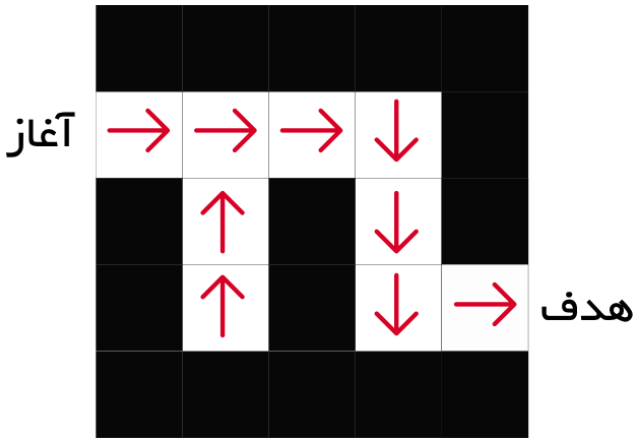
دو دسته از سیاست‌ها وجود دارند.

- قطعی: سیاست برای یک حالت داده شده همیشه عمل مشابهی را باز می‌گرداند.
- تصادفی: برای هر یک از اعمال یک توزیع احتمالی در نظر می‌گیرد.

سیاست تصادفی: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

انجام یک عمل مشخص

مشروط شده به یک حالت



شکل ۱۹ -محاسبه بهترین عمل بر اساس سیاست در مثال هزارتو

همان‌طور که مشهود است، سیاست مستقیماً بهترین عمل برای هر حالت را بیان می‌کند.

[1] Policy-based

در این نوع الگوریتم‌ها، عامل مستقیماً یک سیاست (تابعی که اقدام مناسب در هر حالت را مشخص می‌کند) را یاد می‌گیرد، بدون اینکه نیاز به تخمین تابع ارزش داشته باشد. این الگوریتم‌ها به‌ویژه در مسائلی که فضای عمل پیوسته است، مؤثر هستند.

۱-۳-۲- روش‌های گرادیان خط‌مشی (Policy Gradient)

نوعی دیگر از الگوریتم یادگیری تقویتی که یک خط‌مشی را یاد می‌گیرند. خط‌مشی تابعی است که از حالت‌ها به اقدامات صحیح می‌رسد. این شیوه با تخمین پاداش مورد انتظار باتوجه‌به هر خط‌مشی، گرادیان را به‌روز می‌کند.

به طور مثال، در آموزش سگ، تصور کنید که به او یاد می‌دهید که بنشیند، اما گاهی اوقات او خیلی آهسته می‌نشیند یا اصلاً نمی‌نشیند. با استفاده از گرادیان خط‌مشی[1]، می‌توانید بر اساس میزان پیروی از خط‌مشی، به سگ پاداش‌های مختلفی بدهید. به‌عنوان مثال، اگر سگ سریع بنشیند، می‌توانید غذای زیادی به او بدهید، اما اگر آرام بنشیند، می‌توانید غذای کمی در اختیارش بگذارید. این تفاوت در پاداش‌ها به سگ کمک می‌کند تا بفهمد کدام اقدامات بهتر است و او را تشویق می‌کند تا رفتار مورد انتظار را به طور مؤثرتری انجام دهد. با تکرار مکرر این فرایند، سگ به‌تدریج یاد می‌گیرد که این سیاست را بهتر و بیشتر دنبال کند. به طور مشابه، در هوش مصنوعی، ما مدل را با مثال‌های زیادی آموزش می‌دهیم و زمانی که تصمیم‌های خوب می‌گیرد به آن پاداش می‌دهیم و وقتی تصمیمات بد می‌گیرد، جریمه می‌کنیم. این کار به هوش مصنوعی اجازه می‌دهد تا سیاست خود را بهبود بخشد و در موقعیت‌های مختلف تصمیمات بهتری بگیرد.

هر کدام از این روش‌ها از نظر پیچیدگی متفاوت‌اند و برای انواع مختلفی از مشکلات بر اساس ماهیت محیط و اطلاعات موجود مناسب هستند. علاوه بر روش‌های فوق، تعدادی تکنیک دیگر وجود دارد که می‌توان از آن‌ها برای بهبود فرایند تصمیم‌گیری در یادگیری تقویتی استفاده کرد. این تکنیک‌ها شامل موارد زیر می‌شوند:

اکتشاف حریصانه اپسیلون: تکنیکی است که به عامل اجازه می‌دهد درعین‌حال که پاداش‌های بهینه را یاد می‌گیرد، محیط را هم کاوش کند. عامل با کاوش تصادفی محیط یادگیری را شروع می‌کند؛ اما با یادگیری بیشتر در مورد محیط حریص‌تر می‌شود و اقدامی را با بالاترین ارزش تخمینی انتخاب می‌کند.

پاسخ بر اساس تجربه: تکنیکی که تجربیات عامل را در یک بافر ذخیره می‌کند. این تجربیات ذخیره‌شده به عامل اجازه می‌دهد تا با یادگیری از تعداد بیشتری از تجربیات به طور کارآمدتری فرایند آموزش را طی کند.

[1] Policy Gradient

شکل‌دهی پاداش: این تکنیک پاداش‌های موجود در محیط را تغییر می‌دهد تا مشکل را آسان‌تر حل کند. این کار را می‌توان با افزودن پاداش برای رفتارهای دلخواه یا با حذف پاداش برای رفتارهای ناخواسته انجام داد.

- در یادگیری مبتنی بر سیاست، هیچ تابع ارزشی وجود ندارد.
- خط‌مشی $\pi(s, a)$ توسط بردار θ ($\pi(s, a; \theta)$) پارامتری می‌شود.
- به طور صریح خط‌مشی $\pi(s, a; \theta)$ را یاد بگیرید که به طور ضمنی پاداش را در تمام خط‌مشی‌ها به حداکثر می‌رساند.
- باتوجه‌به سیاست $\pi(s, a; \theta)$ با پارامترهای θ، بهترین θ را پیدا کنید.
- چگونه کیفیت یک خط‌مشی $\pi(s, a; \theta)$ را اندازه‌گیری کنیم؟
- فرض کنید تابع هدف $J(\theta)$ باشد.
- پارامترهای خط‌مشی θ را که $J(\theta)$ را به حداکثر می‌رسانند، پیدا کنید.
- یکی از الگوریتم‌های معروف در این دسته، الگوریتم REINFORCE است که با استفاده از نمونه‌گیری از توزیع احتمال اقدامات، سعی در بهینه‌سازی سیاست عامل دارد.

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \theta) = \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a \mid S_t, \theta) \right].$$

$$\theta_{t+1} \doteq \theta_t + \alpha \sum_a \hat{q}(S_t, a, \mathbf{w}) \nabla \pi(a \mid S_t, \theta)$$

الگوریتم نمونه REINFORCE:

```
Input: a differentiable policy parametrization π(a|s, θ)
Algorithm parameter: step size α > 0
Initialize policy parameter θ ∈ ℝ^d' (e.g., to 0)

Loop forever (for each episode):
  Generate an episode S₀, A₀, R₁, ..., S_{t-1}, A_{t-1}, R_t, following π(·|·, θ)
  Loop for each step of the episode t = 0, 1, ..., T - 1:
    G ← Σ_{k=t+1} γ^{k-t-1} R_k
    θ ← θ + αγ^t G∇ln π(A_t|S_t, θ)
```

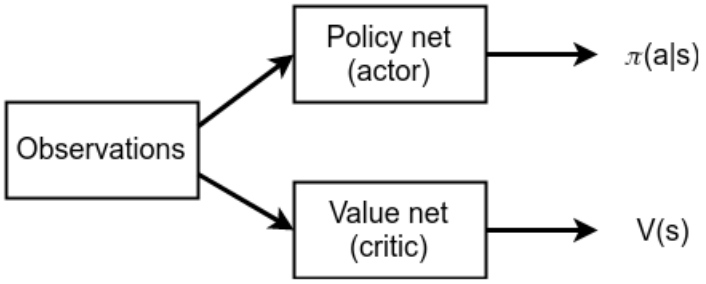
روش‌های عملگر-منتقد

روش عملگر-منتقد[1] به‌طورکلی به یادگیری هم‌زمان یک خط‌مشی و یک تابع ارزش اشاره دارد، جایی که تابع ارزش برای ارزیابی خط‌مشی استفاده می‌شود. عملگر مسئول ایجاد خط‌مشی‌ها، انتخاب کنش‌ها و تعامل با

[1] Actor-Critic

محیط است. عملگر با گرادیان‌های محاسبه شده از توابع زیر به‌روز می‌شود. منتقد کارکرد ارزشی سیاست عملگر را در هر مرحله زمانی ارزیابی می‌کند. معیارهای مختلفی را می‌توان برای ارزیابی خطمشی عملگر استفاده کرد، مانند تابع ارزش عمل Q(s, a)، تابع ارزش حالت V (s) یا تابع مزیت A(s, a).

$$\nabla_\theta J(\theta) = E_{\pi_\theta(s)}\left[\sum_a Q(s,a)\nabla_\theta \pi_\theta(a\,|\,s)\right]$$
$$\theta = \theta + \alpha\nabla_\theta J$$



شکل ۲۰ - عملکرد عملگر-منتقد

الگوریتم عملگر-منتقد

```
Hyperparameters: step size η_θ and η_ψ, reward discount factor γ

Input: initial policy parameters θ_0, initial value function parameters ψ_0

Initialize θ = θ_0 and ψ = ψ_0

for t = 0, 1, 2, ... do
    Run policy π_θ for one step, collection {S_t, A_t, R_t, S_t+1}
    Estimate advantages Â_t = R_t + γV_ψ^πθ(S_t+1) - V_ψ^πθ(S_t)
    J(θ) = Σ_t log π_θ(A_t|S_t)Â_t
    J_V^πθ(ψ) = Σ_t Â_t^2
    ψ = ψ + η_ψ ∇J_V^πθ(ψ), θ = θ + η_θ ∇J(θ)
end for
Return (θ, ψ)
```

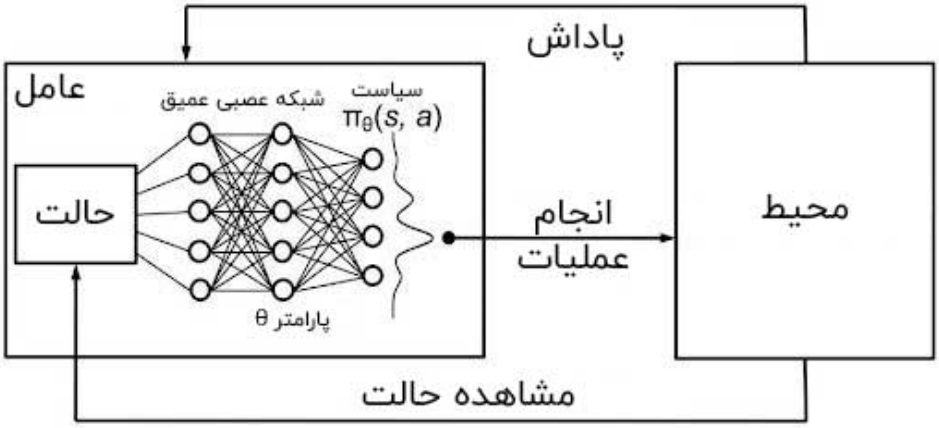
۴-۲- تقریب توابع و یادگیری عمیق در یادگیری تقویتی

عامل یادگیرنده در فرایند یادگیری تقویتی نیاز دارد تا تابع ارزش را محاسبه و ذخیره نماید. اما در مواقعی که ابعاد مسئله بزرگ باشد، ذخیره تمامی ارزش‌های حالت‌های مختلف سیستم عملاً غیرممکن خواهد بود. در این‌گونه مواقع به سراغ روش‌های تخمین تابع ارزش[1] خواهیم رفت. روش‌های تقریب مختلفی وجود دارد که برخی از آنها عبارت‌اند از:

[1] Approximate value function

- تقریب زننده خطی(Linear Approximator)
- تقریب زننده غیرخطی(Non-linear approximator)
- شبکه عصبی(Neural Network)
- و...

درصورتی‌که در روش یادگیری تقویتی، برای تخمین تابع ارزش از شبکه عصبی استفاده شود، شاخه جدیدی به نام **یادگیری تقویتی عمیق**[1] شکل می‌گیرد که در سال‌های اخیر **کاربردهای یادگیری تقویتی** در طیف وسیعی از حوزه‌ها آزمایش شده است.



شکل ۲۱-تعامل عامل و محیط در یادگیری تقویتی عمیق

یکی از چالش‌های موجود در پیاده‌سازی روش‌های یادگیری تقویتی در دنیای واقعی نیاز به حجم زیادی از داده جهت دستیابی به نتایج مطلوب است. در روش‌های کلاسیک یادگیری تقویتی، این داده‌ها که به واسطه تعامل با محیط و انجام آزمایش‌های سعی و خطا تولید می‌گردند، قابلیت تعمیم‌دهی کمی دارند. به بیان دیگر برای مسئله جدید مجدداً باید فرایند جمع‌آوری داده طی شود که می‌تواند بسیار دشوار و زمان‌بر باشد، به‌ویژه در مواردی که تعداد ترکیبی از اقدامات و حالت‌ها بسیار زیاد است و یا جایی که محیط غیرقطعی بوده و می‌تواند حالت‌های تقریباً نامحدودی داشته باشد.

بدین منظور مسائل یادگیری تقویتی عمیق که استفاده از یادگیری عمیق در بستر یادگیری تقویتی را فراهم کرده، مطرح شده است تا بتواند بسیاری از چالش‌های موجود در این زمینه را رفع نماید. باتوجه‌به ساختار و تابع بهینه‌سازی مناسب، یک شبکه عصبی عمیق می‌تواند یک خط‌مشی بهینه را بدون گذر از تمام حالت‌های ممکن

[1] Deep Reinforcement Learning

یک سیستم، یاد بگیرد. هرچند عوامل یادگیری تقویتی عمیق همچنان به حجم عظیمی از داده‌ها نیاز دارند، اما می‌توانند مشکلاتی را که حل آنها با سیستم‌های کلاسیک یادگیری تقویتی غیرممکن بود، حل کنند.

یادگیری عمیق، شامل استفاده از تقریب توابع غیرخطی چندلایه، معمولاً شبکه‌های عصبی است. یادگیری عمیق کار متفاوتی با آنچه قبلاً توضیح داده شد نیست. یادگیری عمیق مجموعه‌ای از تکنیک‌ها و روش‌ها برای استفاده از شبکه‌های عصبی برای حل وظایف یادگیری ماشین، اعم از یادگیری با نظارت، یادگیری بدون نظارت، یا یادگیری تقویتی است. یادگیری تقویتی عمیق استفاده از یادگیری عمیق برای حل وظایف یادگیری تقویتی است.

منظور از تقریب توابع[1] چیست؟ در واقع منظور از تقریب توابع غیرخطی همان یادگیری عمیق است. در این روش ما روش‌های ارائه‌شده را گسترش می‌دهیم تا برای مسائل با فضاهای حالت بزرگ دلخواه اعمال شود. در بسیاری از وظایفی که ما می‌خواهیم یادگیری تقویتی را برای آنها اعمال کنیم، فضای حالت ترکیبی و عظیم است. در چنین مواردی ما نمی‌توانیم انتظار داشته باشیم که یک خط‌مشی بهینه یا تابع مقدار بهینه را حتی در محدوده زمان و داده بی‌نهایت پیدا کنیم. هدف ما در عوض یافتن یک راه‌حل تقریبی خوب با استفاده از منابع محاسباتی محدود است.

مشکل فضاهای حالت بزرگ فقط حافظه موردنیاز برای جداول بزرگ نیست، بلکه زمان و داده‌های موردنیاز برای پر کردن دقیق آنهاست. در بسیاری از وظایف هدف ما، تقریباً هر حالتی که با آن مواجه می‌شویم هرگز قبلاً دیده نشده است. برای تصمیم‌گیری معقول در چنین حالت‌هایی، لازم است از رویارویی‌های قبلی با حالت‌های متفاوتی که به‌نوعی شبیه به حالت فعلی هستند، تعمیم دهیم. به‌عبارت‌دیگر، موضوع کلیدی بحث تعمیم است.

مزایای استفاده از تقریب تابع مقدار نه‌تنها شامل مقیاس‌پذیری برای وظایف در مقیاس بزرگ، بلکه سهولت تعمیم به حالت‌های نادیده از حالت‌های دیده شده با فضاهای حالت پیوسته است. علاوه بر این، تقریب تابع مبتنی بر شبکه‌های عصبی نیز نیاز به طراحی دستی ویژگی‌ها برای نمایش حالت‌ها را کاهش می‌دهد یا حذف می‌کند.

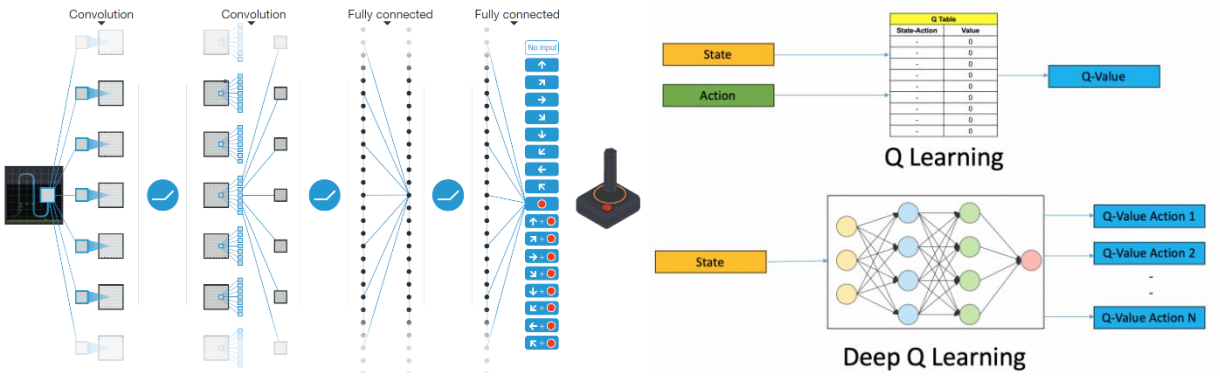
انگیزه ما برای استفاده از تقریب تابع نه‌تنها حل مسائلی است که در غیر این صورت قابل‌حل نیستند، بلکه برای حل کارآمدتر مسائل است.

[1] function approximation

۱-۴-۲- Deep Q-Networks (DQN)

الگوریتم یادگیری Q الگوریتم قدرتمندی است، اما قابلیت تعمیم‌پذیری ندارد و همین مسئله را می‌توان بزرگ‌ترین نقطه‌ضعف آن دانست. اگر الگوریتم یادگیری Q را به‌روزرسانی اعداد موجود در یک آرایه دوبعدی (شامل: فضای اقدام×فضای حالت) در نظر بگیرید، متوجه شباهت آن با برنامه‌نویسی پویا خواهید شد. این موضوع برای ما روشن می‌سازد که وقتی عامل تصمیم‌گیرنده در الگوریتم یادگیری Q با وضعیتی کاملاً جدید روبه‌رو شود، هیچ راهی برای شناسایی و انتخاب اقدام مناسب نخواهد داشت. به‌عبارت‌دیگر، عامل تصمیم‌گیرنده الگوریتم یادگیری Q توانایی تخمین ارزش حالت‌های ناشناخته را ندارد. برای حل این مشکل، شبکه DQN آرایه دوبعدی را حذف و شبکه عصبی را جایگزین آن می‌کند.

شبکه DQN به کمک یک شبکه عصبی، تابع Q-value را تخمین می‌زند. حالت فعلی به‌عنوان ورودی به این شبکه داده می‌شود، سپس مقدار Q-value متناظر با هر اقدام به‌عنوان خروجی از شبکه دریافت خواهد شد.



شکل ۲۲-تفاوت یادگیری Q با یادگیری Q عمیق (راست) و مثالی از شبکه DQN در آتاری (چپ)

شرکت **دیپ مایند** در سال ۲۰۱۳، شبکه DQN را همان‌طور که در تصویر بالا ملاحظه می‌کنید، در بازی آتاری به کار گرفت. ورودی که به این شبکه داده می‌شد یک تصویر خام از حالت جاری بازی بود. این ورودی از چندین لایه مختلف از جمله لایه‌های پیچشی و تماماً متصل عبور می‌کند و خروجی نهایی شامل مقادیرQ-value های مربوط به تمام اقدامات احتمالی عامل تصمیم‌گیرنده است.

دو تکنیک دیگر نیز برای آموزش شبکه DNQ ضروری است:

۱. **تکرار تجربه**: در سازوکار متداول یادگیری تقویتی، نمونه‌های آموزشی همبستگی بالایی دارند و ازنظر مقدار داده موردنیاز نیز کارآمد نیستند. به همین دلیل، سخت است که مدل به مرحله همگرایی برسد. یک راه برای حل مسئله توزیع نمونه، به‌کارگیری تکنیک تکرار تجربه است. در این روش، تابع انتقال

نمونه‌ها ذخیره می‌شـــود و ســـپس الگوریتم این تجربیات را از محلی به نام «مجموعه انتقال» به طور تصادفی انتخاب می‌کند تا دانش خود را بر اساس آن به‌روزرسانی نماید.

۲. **شبکه هدف مجزا**: ساختار شبکه هدف[1] Q مشابه ساختار شبکه‌ای است که ارزش را تخمین می‌زند. همان‌طور در نمونه کد بالا ملاحظه کردید، بعد از C مرحله، در شبکه هدف جدیدی تعریف می‌شود تا از شدت نوسانات کاسته شود و فرایند آموزش ثبات بیشتری داشته باشد.

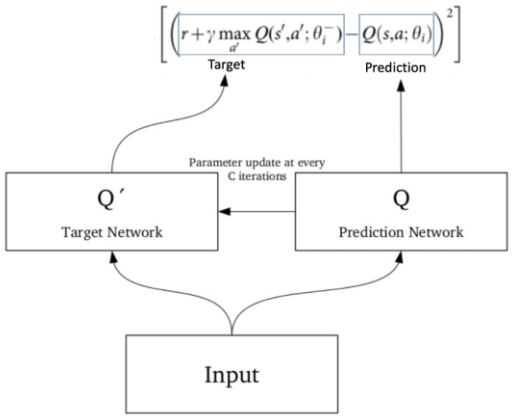
- شبکه Q عمیق شامل
  - شبکه Q که Q-value را پیش‌بینی می‌کند
  - شبکه هدف که ساختاری مشابه شبکه Q دارد
  - مؤلفه تکرار تجربه
- تکرار تجربه یک عمل حریصـــانه(ε-greedy) را از حالت فعلی انتخاب می‌کند، آن را در محیط اجرا می‌کند و یک پاداش و حالت بعدی را پس می‌گیرد. این مشاهده را به‌عنوان نمونه‌ای از داده‌های آموزشی ذخیره می‌کند
- دسته‌ای از داده‌های آموزشی به هر دو شبکه داده می‌شود.
  - شـــبکه Q حالت فعلی و عمل را از هر نمونه داده می‌گیرد و Q-value را برای آن عمل خاص پیش‌بینی می‌کند.
  - شـــبکه Target(Q′) حالت بعدی را از هر نمونه داده می‌گیرد و بهترین Q-value را از بین تمام اقداماتی که می‌توان از آن حالت انجام داد، پیش‌بینی می‌کند.
- تابع ضرر[2] در تکرار i به‌صورت تعریف شده است:

$$J_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim U(S)}\left[\left[\left((r+\gamma \max_{a'} Q(s',a';\theta^{-}) - Q(s,a;\theta_i)\right)^2\right]\right]$$

که در آن U(S) توزیع یکنواخت از مجموعه آموزشی S و θ^{-}_i پارامترهای شبکه هدف است.

- فقط شبکه Q آموزش‌داده‌شده و شبکه هدف ثابت است.
- در هر مرحله C، وزن شبکه Q در شبکه هدف کپی می‌شود.

[1] Target Network
[2] loss function



شکل ۲۳-عملکرد DQN

الگوریتم DQN همراه با تکرار تجربه:

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function Q^ with weights θ⁻=θ
For episode = 1, M do
  Initialize sequence s₁ = {x₁} and preprocessed sequence ϕ₁ = ϕ(s₁)
  For t = 1,T do
    With probability ε, select a random action aₜ
    Otherwise select aₜ = argmaxₐQ(ϕ(sₜ),aₜ; θ)
    Execute action aₜ in emulator and observe reward rₜ and image xₜ₊₁
    Set sₜ₊₁ = sₜ,aₜ,xₜ₊₁ and preprocesses ϕₜ₊₁ = ϕ(sₜ₊₁)
    Store transition (ϕₜ₊₁,aₜ,rₜ,ϕₜ₊₁) in D
    Sample random minibatch of transitions (ϕⱼ,aⱼ,rⱼ;ϕⱼ₊₁) from D
    Set yj = {
    rⱼ                              if episode terminates at step j+1
    rⱼ + γ maxₐ' Q^(ϕⱼ₊₁, a'; θ⁻)   otherwise
    }
     Perform a gradient descent step on (yⱼ-Q(ϕⱼ,aⱼ; θ))² with respect to
    the network parameters θ
     Every C steps reset Q^ = Q
  End For
End For
```

کد ساختار شبکه DQN:

```
class DQN(nn.Module):

    def __init__(self, n_observations, n_actions):
        super(DQN, self).__init__()
        self.layer1 = nn.Linear(n_observations, 128)
        self.layer2 = nn.Linear(128, 128)
        self.layer3 = nn.Linear(128, n_actions)

    # Called with either one element to determine next action, or a batch
    # during optimization. Returns tensor([[left0exp,right0exp]...]).
    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
     return self.layer3(x)
```

۲-۴-۲- گرادیان خط‌مشی قطعی عمیق(**DDPG**)

اگرچه شبکه DQN در مسائلی که ابعاد زیادی دارند (همچون بازی آتاری)، بسیار موفق عمل کرده است، اما فضای اقدام در این **شبکه گسسته** است و ازآنجاکه بسیاری از مسائل موردعلاقه و حائز اهمیت برای ما از جمله مسائل مربوط به کنترل فیزیکی، دارای فضای اقدام پیوسته هستند، گسسته بودن فضا در شبکه DQN یک نقطه‌ضعف به شمار می‌آید.

ما الگوریتمی به نام گرادیان خط‌مشی قطعی عمیق[1] (DDPG) را بررسی می‌کنیم. DDPG را می‌توان به‌عنوان یک DQN برای **فضاهای عمل پیوسته** در نظر گرفت. DDPG از بسیاری از تکنیک‌های مشابه موجود در DQN استفاده می‌کند. بااین‌حال، DDPG همچنین سیاستی را آموزش می‌دهد که عمل بهینه را تقریب می‌کند. به همین دلیل، DDPG یک روش گرادیان خط‌مشی قطعی است که به فضاهای کنش پیوسته محدود شده است.

با تبدیل یک فضای اقدام پیوسته به یک فضای گسسته به‌صورت دقیق و جزئی، یک فضای اقدام بسیار گسترده به دست می‌آید. برای مثال، فرض کنید درجه آزادی سیستم تصادفی ۱۰ باشد. به‌ازای هر درجه، باید فضا را به ۴ قسمت تقسیم کنیم. به‌این‌ترتیب، در آخر ۴^{10}=۱۰۴۸۵۷۶ عدد اقدام خواهیم داشت. سخت است که در چنین فضای اقدام بزرگی به همگرایی برسید.

الگوریتم DDPG مبتنی بر **معماری عملگر منتقد** عمل می‌کند. این معماری دو عنصر اصلی دارد: عملگر و منتقد. عنصر عملگر، تنظیم پارامتر θ با تابع سیاست را برعهده دارد، به‌این‌ترتیب، عملگر تصمیم می‌گیرد که بهترین اقدام ممکن برای هر حالت چیست.

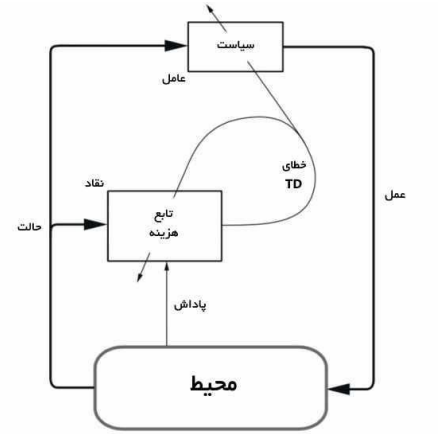
$$\pi_\theta(s,a) = \mathbb{P}[a|\ s,\theta]$$

وظیفه منتقد نیز ارزیابی تابع سیاستی است که عملگر بر اساس تابع خطای **تفاوت موقتی** تخمین زده است.

$$r_{t+1} + \gamma V^v(s_{t+1}) - V^v(s_t)$$

در این تابع حرف v کوچک نمایانگر سیاستی است که عملگر برمی‌گزیند. این فرمول کمی آشنا به نظر نمی‌رسد؟ بله، درست است. این فرمول دقیقاً مشابه معادله به‌روزرسانی الگوریتم یادگیری Q است. یادگیری تفاوت زمانی روشی است که الگوریتم به کمک آن می‌تواند نحوه پیش‌بینی یک تابع ارزش را بر پایه ارزش‌های آتی یک حالت معین بیاموزد. الگوریتم یادگیری Q نوعی خاصی از یادگیری تفاوت زمانی در حوزه یادگیری Q-value است.

[1] deep deterministic policy gradient



شکل ۲۴-معماری عملگر-منتقد

الگوریتم DDPG تکنیک‌های تکرار تجربه و شـــبکه هدف مجزا در شبکه DQN را نیز به کار می‌گیرد. اما یکی از مشکلات DDPG این است که به‌ندرت اقدامات را جست‌وجو می‌کند. یک راه‌حل برای این مشکل، **ایجاد اختلال** در فضای پارامترها یا فضای اقدام است.

البته محققین OpenAI در مقاله خود ادعا کرده‌اند که ایجاد اختلال در فضای پارامترها بهتر از ایجاد اختلال در فضای اقدام است. یکی از رایج‌ترین و پرکاربردترین اختلالات در این زمینه فرایند تصادفی **اورنستین-یولنبک** نام دارد.

الگوریتم DDPG:

```
Randomly initialize critic network Q(s, a | θ^Q) and actor μ(s | θ^μ) with weights
θ^Q and θ^μ
Initialize target network Q' and μ' with weights θ^Q' ← θ^Q, θ^μ' ← θ^μ
Initialize replay buffer R
for episode = 1, M do
    Initialize a random process N for action exploration
    Receive initial observation state s_1
    for t = 1, T do
        Select action a_t = μ(s_t | θ^μ) + N_t according to the current policy and
        exploration noise
        Execute action at and observe reward r_t and observe new state s_t+1
        Store transition (s_t, a_t, r_t, s_t+1) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_i+1) from R
        Set y_i = r_i + γ Q'(s_i+1, μ'(s_i+1 | θ^μ') | θ^Q')
        Update critic by minimizing the loss: L = 1/N Σ_i (y_i - Q(s_i, a_i | θ^Q))^2
        Update the actor policy using the sampled policy gradient:
        ∇_θμ J ≈ 1/N Σ_i ∇_a Q(s, a | θ^Q) |_s=si, a=μ(si) ∇_θμ μ(s | θ^μ) |_si
        Update the target networks:
        θ^Q' ← τ θ^Q + (1 - τ) θ^Q'
        θ^μ' ← τ θ^μ + (1 - τ) θ^μ'
    end for
end for
```

کد ساختار شبکه DDPG:

```
class DDPGActor(nn.Module): def __init__(self, obs_size, act_size):
super(DDPGActor, self).__init__()
self.net = nn.Sequential( nn.Linear(obs_size, 400), nn.ReLU(), nn.Linear(400,
300), nn.ReLU(), nn.Linear(300, act_size), nn.Tanh() )
def forward(self, x): return self.net(x)

class DDPGCritic(nn.Module): def __init__(self, obs_size, act_size):
super(DDPGCritic, self).__init__()
self.obs_net = nn.Sequential( nn.Linear(obs_size, 400), nn.ReLU(), )
self.out_net = nn.Sequential( nn.Linear(400 + act_size, 300), nn.ReLU(),
nn.Linear(300, 1) )
def forward(self, x, a): obs = self.obs_net(x) return
self.out_net(torch.cat([obs, a], dim=1))
```

۳- مباحث پیشرفته در یادگیری تقویتی

پس از بررسی الگوریتم‌های یادگیری تقویتی در بخش قبل، در این بخش از دنیای شبیه‌سازی و بازی فاصله گرفته و به دنیای واقعی و محیط‌های پیچیده‌تر پرداخته می‌شود و انواع تکنیک‌های پیشرفته‌تر و نحوه مقابله با محیط‌های پیچیده‌تر بررسی می‌شود.

۳-۱- انواع Deep Q-Networks

از زمانی که مدل شبکه Q عمیق (DQN) در سال ۲۰۱۵ معرفی شد، پیشرفت‌های زیادی، همراه با تغییراتی در معماری پایه، ارائه شده است که به طور قابل توجهی همگرایی، پایداری و کارایی نمونه پایه DQN را بهبود بخشیده است. . در این قسمت نگاهی به برخی از این ایده ها خواهیم داشت.

۳-۱-۱- N-step DQN

اولین پیشرفتی که بررسی خواهیم کرد، یک پیشرفت کاملاً قدیمی است. اولین‌بار در سال ۱۹۸۸ معرفی شد. برای درک ایده، اجازه دهید یک‌بار دیگر به آپدیت بلمن مورداستفاده در یادگیری Q نگاه کنیم:

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a_{t+1})$$

این معادله بازگشتی است، به این معنی که می‌توانیم $Q(s_{t+1}, a_{t+1})$ را بر حسب خودش بیان کنیم که این نتیجه را به ما می‌دهد:

$$Q(s_t, a_t) = r_t + \gamma \max_a \left[r_{a,t+1} + \gamma \max_{a'} Q(s_{t+2}, a') \right]$$

مقدار $r_{a,t+1}$ به معنای پاداش محلی در زمان t+1، پس از صدور اقدام a است. بااین‌حال، اگر فرض کنیم که عمل a در مرحله t+1 به طور بهینه یا نزدیک به بهینه انتخاب شده است، می‌توانیم عملیات $\max_a$ را حذف کنیم و این را به دست آوریم:

$$Q(s_t, a_t) = r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s_{t+2}, a')$$

برای سرعت‌بخشیدن به همگرایی(convergence) این مقدار می‌تولند بارهاوبارها باز شود. این بازکردن[1] را می‌توان به‌راحتی با جایگزین‌کردن نمونه انتقال یک‌مرحله‌ای(1-Step) با دنباله‌های انتقال طولانی‌تر از n مرحله(n-Step)، به به‌روزرسانی DQN ما اعمال کرد. این باز کردن به ما کمک می‌کند تا سرعت آموزش را افزایش دهیم.

[1] unrolling

الگوریتم یادگیری N-step Q به جز تغییرات زیر به روشی مشابه DQN عمل می‌کند:

- هیچ بافر اجرا مجدد استفاده نمی‌شود. به‌جای نمونه‌برداری از دسته‌های تصادفی انتقال، شبکه هر N مرحله با استفاده از آخرین N مرحله‌ای که توسط عامل اجرا می‌شود، آموزش داده می‌شود.
- به‌منظور تثبیت یادگیری، چندین کارگر با هم برای به‌روزرسانی شبکه کار می‌کنند. این همان اثر عدم همبستگی نمونه‌های مورداستفاده برای آموزش را ایجاد می‌کند.
- به‌جای استفاده از اهداف Q تک‌مرحله‌ای برای شبکه، پاداش‌های حاصل از مراحل بعدی N جمع‌آوری می‌شوند تا اهداف N مرحله Q را تشکیل دهند.

۲-۱-۳- Double DQN

ایده ثمربخش بعدی در مورد چگونگی بهبود DQN پایه توسط محققان DeepMind در مقاله‌ای ارائه شد. در این مقاله، نویسندگان نشان دادند که DQN اصلی تمایل دارد مقادیر Q را بیش از حد تخمین بزند که ممکن است برای عملکرد آموزشی مضر باشد و گاهی اوقات می‌تواند منجر به سیاست‌های غیربهینه شود. علت اصلی آن، عملیات max در معادله بلمن است. به‌عنوان راه‌حلی برای این مشکل، نویسندگان پیشنهاد کردند که آپدیت بلمن را کمی تغییر یابد:

$$Q(s_t,\ a_t)\ =\ r_t\ +\ \gamma\ \max_a Q'(s_{t+1},\ a)$$

$Q'(s_{t+1}, a)$ مقادیر Q بود که با استفاده از شبکه هدف ما محاسبه شد، بنابراین ما با شبکه آموزش‌دیده هر n مرحله را به‌روز می‌کنیم. نویسندگان مقاله پیشنهاد کردند اقداماتی را برای حالت بعدی با استفاده از شبکه آموزش‌دیده انتخاب کنند، اما مقادیر Q را از شبکه هدف بگیرند؛ بنابراین، عبارت جدید برای Q-values هدف به شکل زیر خواهد بود:

$$Q(s_t,\ a_t)\ =\ r_t\ +\ \gamma\ \max_a Q'\Big(s_{t+1},\ arg\ \max_a Q(s_{t+1},\ a)\Big)$$

نویسندگان ثابت کردند که این پیچ‌دادن ساده[1] تخمین بیش از حد را کاملاً برطرف می‌کند و آنها این معماری جدید را DQN دوگانه(Double DQN) نامیدند.

پیاده‌سازی آن بسیار ساده است. کاری که ما باید انجام دهیم این است که تابع ضرر خود را کمی اصلاح کنیم.

[1] simple tweak fixes

۳-۱-۳- Noisy networks

پیشرفت بعدی که می‌خواهیم به آن بپردازیم به یکی دیگر از مشکلات یادگیری تقویتی می‌پردازد: اکتشاف در محیط. شبکه‌های نویزی[1] یک ایده بسیار ساده برای یادگیری ویژگی‌های اکتشاف در طول آموزش به‌جای داشتن یک برنامه زمانی جداگانه مرتبط با اکتشاف است.

DQNکلاسیک با انتخاب اقدامات تصادفی با اپسیلون هایپرپارامتر ویژه تعریف شده به اکتشاف دست می‌یابد که به‌آرامی در طول زمان از ۱ (عملکردهای کاملاً تصادفی) به نسبت کوچکی ۰.۱ یا ۰.۰۲ کاهش می‌یابد. این فرایند برای محیط‌های ساده به خوبی کار می‌کند. اما حتی در چنین موارد ساده ای، برای کارآمد کردن فرایندهای آموزشی نیاز به تنظیم دارد .

در روش Noisy Networks نویز را به وزن لایه‌های کاملاً متصل شبکه اضافه می‌کنند و پارامترهای این نویز را در حین تمرین با استفاده از پس انتشار تنظیم می‌کنند. نویز توسط یک جریان نویز اضافی به لایه خطی $y = (W x + b)$ اضافه می‌شود:

$$y = (Wx + b) + ((W_{noisy} \odot \epsilon_w)x + b_{noisy} \odot \epsilon_b)$$

که در آن $\odot$ به ضرب از نظر عنصر اشاره دارد، هر دو W_{noisy} و b_{noisy} پارامترهای قابل‌آموزش هستند، درحالی‌که ε_w و ε_b مقیاس‌های تصادفی هستند که به صفر می‌رسند. آزمایش‌ها نشان می‌دهد که شبکه‌های نویزی برای طیف وسیعی از بازی‌های آتاری در چندین خط پایه امتیازات بالاتری به دست می‌آورند.

۳-۱-۴- Prioritized replay buffer

یکی از زمینه‌های بهبود در DQN پایه، استراتژی نمونه‌گیری بهتر برای تکرار تجربه است. تکرار تجربه اولویت‌دار[2] (PER) تکنیکی برای اولویت‌بندی تجربه است، به‌گونه‌ای که انتقال‌های مهم را بیشتر تکرار کنیم. ایده اصلی تکرار تجربه اولویت‌دار درنظرگرفتن اهمیت انتقال با خطای تفاوت زمانی δ است که می‌تواند به‌عنوان یک اندازه‌گیری شگفت‌انگیز در نظر گرفته شود.

دلیل این که تکرار تجربه اولویت‌دار می‌تواند کمک‌کننده باشد این است که برخی از تجربیات ممکن است حاوی اطلاعات بیشتری برای یادگیری در مقایسه با دیگران باشد. دادن شانس بیشتری برای بازپخش شدن به آن تجربه غنی از اطلاعات بیشتر به سریع‌تر و کارآمدتر شدن کل فرایند یادگیری کمک می‌کند.

[1] Noisy Networks
[2] Prioritized experience replay

تکرار تجربه اولویت‌دار از خطای تفاوت زمانی(TD) هر انتقال به‌عنوان معیار ارزیابی اولویت اســـتفاده می‌کند. شکل خطای TD به‌صورت زیر است:

$$\delta_t = R_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1})$$

هر چه‌قدر مطلق δ_t بزرگ‌تر باشـد و احتمال اینکه انتقال مربوطه انتخاب شـود بیشـتر باشـد، سـهم بیشـتری در بهبود سـیاسـت خواهد داشـت. در فرایند نمونه‌گیری، روش‌های **اولویت‌بندی تصـادفی[1] و نمونه‌گیری اهمیت[2]** به کار گرفته می‌شـود. عملیات اولویت‌بندی تصـادفی نه‌تنها می‌تواند از انتقال‌ها اسـتفاده کامل کند، بلکه تنوع را نیز تضمین می‌کند. نمونه‌برداری اهمیت سرعت به‌روزرسانی پارامتر را کاهش می‌دهد و ثبات یادگیری را تضمین می‌کند.

برای پیاده‌ســـازی کارآمد، تابع چگالی تجمعی[3] احتمال نمونه‌برداری با یک تابع خطی تکه‌ای[4] با k بخش تقریب زده می‌شود. به طور دقیق‌تر، اولویت‌ها در یک ساختار داده کارآمد به نام درخت قطعه[5] ذخیره می‌شوند. در طول زمان اجرا، تکرار تجربه اولویت‌دار ابتدا از یک قطعه نمونه‌برداری می‌کند، و ســـپس به طور یکنواخت از بین انتقال‌های درون آن نمونه‌برداری می‌کند.

۵-۱-۳- Dueling DQN

برای برخی از حالت‌ها، اقدامات مختلف به مقدار مورد انتظار مربوط نمی‌شـــود و ما نیازی به یادگیری تأثیر هر عمل برای چنین حالت‌هایی نداریم. برای مثال تصـــور کنید روی کوه ایســـتاده‌اید و طلوع خورشـــید را تماشـــا می‌کنید. منظره دلپذیر به شـــما آرامش زیادی می‌دهد که پاداش بالایی را به همراه دارد. شـــما می‌توانید اینجا بمانید و Q-value اقدامات مختلف مهم نیســـت؛ بنابراین جداکردن ارزش مســـتقل از عمل حالت و Q-value ممکن است به یادگیری قوی‌تری منجر شود.

مدل شبکه‌های عصبی عمیق مورداستفاده در DQN دارای سه لایه کانولوشن و به دنبال آن دو لایه کاملاً متصل است و خروجی مقادیر Q هر عمل است. متفاوت از معماری DQN، معماری دوئل ویژگی‌های استخراج شده از لایه‌های کانولوشـن را به دو جریان تقسـیم می‌کند. به طور دقیق‌تر، Q-value را می‌توان به بخش مقدار حالت و بخش مزیت عمل به‌صورت زیر تقسیم کرد:

$$Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a)$$

[1] stochastic prioritization
[2] Importance-sampling
[3] cumulative density function
[4] piece-wise linear function
[5] segment tree

و دوئل DQN نمایش(representations) های این دو بخش را به‌صورت زیر جدا می‌کند:

$$Q(s,a;\theta,\theta_v,\theta_a)=V(s;\theta,\theta_v)+(A(s,a';\theta,\theta_a)-\max_{a'}A(s,a';\theta,\theta_a))$$

که در آن θ_v و θ_a پارامترهای دو جریان لایه‌های کاملاً متصل[1] هستند، θ پارامترهای لایه‌های کانولوشن را نشان می‌دهد. توجه داشته باشید که عملگر max در معادله قابلیت شناسایی[2] را تضمین می‌کند که Q-value مقدار حالت و مزیت عمل را به طور منحصربه‌فرد بازیابی کند. در غیر این صورت، آموزش ممکن است عبارت مقدار حالت را نادیده بگیرد و تابع مزیت را فقط به Q-value همگرا کند. پیشنهاد شده است که برای پایداری بهتر، max با میانگین به‌صورت زیر جایگزین شود:

$$Q(s,a;\theta;\theta_v,\theta_a)=V(s;\theta,\theta_v)+(A(s,a;\theta,\theta_a)-\frac{1}{|A|}\sum_{a'}A(s,a';\theta,\theta_a))$$

که توسط آن تابع مزیت به‌جای دنبال‌کردن مزیت بهینه، فقط باید با مزیت مستقیم میانگین تطبیق یابد.

آموزش معماری‌های دوئل، مانند DQN استاندارد، تنها به لایه‌های بیشتری نیاز دارد. آزمایش‌ها نشان می‌دهند که معماری‌های دوئل منجر به ارزیابی بهتر سیاست در حضور بسیاری از اقدامات باارزش مشابه می‌شوند.

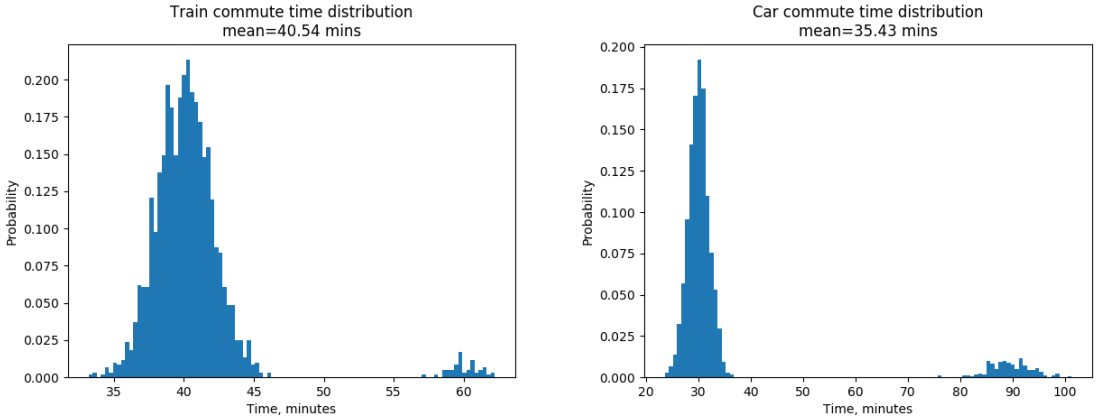
۶-۱-۳- Categorical(Distributional) DQN

آخرین و پیچیده‌ترین روش در جعبه‌ابزار بهبود DQN مربوط به یک مقاله است که توسط DeepMind در ژوئن ۲۰۱۷ منتشر شده است، به نام دیدگاه توزیعی[3] در یادگیری تقویتی. در مقاله، نویسندگان بخش اساسی یادگیری Q که همان Q-values است را زیر سوال بردند و سعی کردند آنها را با توزیع احتمال Q-value عمومی تر جایگزین کنند. بیایید سعی کنیم ایده را درک کنیم. هر دو روش یادگیری Q و تکرار ارزش با مقادیر اعمال یا حالت‌هایی که به‌صورت اعداد ساده نشان داده شده‌اند کار می‌کنند و نشان می‌دهند که ما چقدر می‌توانیم از یک حالت، یا یک عمل و یک حالت به پاداش کلی برسیم. بااین‌حال، آیا فشرده کردن همه پاداش‌های احتمالی آینده در یک عدد عملی است؟ در محیط‌های پیچیده، آینده می‌تواند تصادفی باشد و مقادیر متفاوتی با احتمالات متفاوت به ما بدهد.

به‌عنوان مثال، سناریوی رفت‌وآمد را زمانی که به طور منظم از خانه به محل کار رانندگی می‌کنید، تصور کنید. بیشتر اوقات، ترافیک آن‌چنان سنگین نیست و حدود ۳۰ دقیقه طول می‌کشد تا به مقصد برسید. دقیقاً ۳۰ دقیقه نیست، اما به طور متوسط ۳۰ است. هرازگاهی اتفاقی می‌افتد، مانند تعمیر جاده یا تصادف، و به دلیل

[1] fully connected layers
[2] identifiability
[3] Distributional

ترافیک، سه برابر بیشتر طول می‌کشد تا به سر کار برسید. احتمال زمان رفت‌وآمد شما را می‌توان به‌عنوان توزیعی از متغیر تصادفی "زمان رفت‌وآمد" نشان داد و در نمودار زیر نشان‌داده‌شده است.



شکل ۲۵-توزیع احتمال زمان رفت‌وآمد با ماشین(راست) و قطار(چپ)

حالا تصور کنید که یک راه جایگزین برای رسیدن به محل کار دارید: قطار. کمی بیشتر طول می‌کشد، زیرا باید از خانه به ایستگاه قطار و از ایستگاه به دفتر بروید، اما بسیار قابل‌اعتمادتر است. مثلاً بگویید که زمان رفت‌وآمد قطار به طور متوسط ۴۰ دقیقه است، با احتمال کمی اختلال در قطار که ۲۰ دقیقه زمان اضافی به سفر اضافه می‌کند. توزیع رفت‌وآمد قطار در نمودار بالا نشان‌داده‌شده است.

تصور کنید که اکنون می‌خواهیم در مورد نحوه رفت‌وآمد تصمیم‌گیری کنیم. اگر فقط میانگین زمان ماشین و قطار را بدانیم، ماشین جذاب‌تر به نظر می‌رسد، زیرا به طور متوسط ۳۵:۴۳ دقیقه طول می‌کشد که بهتر از قطار ۴۰:۵۴ دقیقه‌ای است.

بااین‌حال، اگر به توزیع کامل نگاه کنیم، ممکن است تصمیم بگیریم با قطار برویم، زیرا حتی در بدترین سناریو، یک ساعت رفت‌وآمد در مقابل یک ساعت و ۳۰ دقیقه خواهد بود. با تغییر به زبان آماری، توزیع خودرو **واریانس** بسیار بالاتری دارد، بنابراین در شرایطی که واقعاً باید حداکثر در ۶۰ دقیقه در دفتر باشید، قطار بهتر است.

وضعیت در سناریوی فرایند تصمیم مارکوف پیچیده‌تر می‌شود، زمانی که توالی تصمیم‌ها باید گرفته شود و هر تصمیمی ممکن است بر وضعیت آینده تأثیر بگذارد. در مثال رفت‌وآمد، ممکن است زمان یک جلسه مهم باشد که باتوجه‌به راهی که می‌خواهید رفت‌وآمد کنید، باید آن را ترتیب دهید. در این صورت، کار با مقادیر میانگین پاداش ممکن است به معنای ازدست‌دادن اطلاعات زیادی در مورد دینامیک‌های اساسی باشد.

دقیقاً همین ایده توسط نویسندگان دیدگاه توزیعی در یادگیری تقویتی ارائه شد. چرا با تلاش برای پیش‌بینی یک مقدار متوسط برای یک اقدام، خودمان را محدود می‌کنیم، درحالی‌که ارزش اساسی ممکن است توزیع زیربنایی پیچیده‌ای داشته باشد؟ شاید به ما کمک کند که مستقیماً با توزیع‌ها کار کنیم.

نتایج ارائه شده در مقاله نشان می‌دهد که در واقع، این ایده می‌تواند مفید باشد، اما به قیمت معرفی یک روش پیچیده‌تر. ایده کلی این است که توزیع ارزش را برای هر عمل، شبیه به توزیع‌های مثال ماشین/قطار پیش‌بینی کنیم. در مرحله بعد، نویسندگان نشان دادند که معادله بلمن را می‌توان برای یک حالت توزیع تعمیم داد، و به شکل $Z(x,a)\overset{D}{=}R(x,a)+\gamma Z(x',a')$ که بسیار شبیه به معادله بلمن آشنا است، اما اکنون Z(x, a) و R(x, a) **توزیع‌های احتمال** هستند و نه اعداد.

توزیع به‌دست‌آمده می‌تواند برای آموزش شبکه ما استفاده شود تا پیش‌بینی‌های بهتری از توزیع ارزش برای هر عمل در حالت داده شده ارائه دهد، دقیقاً به همان روشی که با یادگیری Q انجام می‌شود. تنها تفاوت در تابع ضرر خواهد بود که اکنون باید با چیزی مناسب برای مقایسه توزیع جایگزین شود. چندین گزینه در دسترس وجود دارد، به‌عنوان مثال، واگرایی Kullback-Leibler (KL) (ازدست‌دادن آنتروپی متقابل) که در مسائل طبقه‌بندی استفاده می‌شود، یا معیار Wasserstein. در مقاله، نویسندگان برای معیار Wasserstein توجیه نظری ارائه کردند، اما زمانی که سعی کردند آن را در عمل اعمال کنند، با محدودیت‌هایی مواجه شدند. بنابراین، در نهایت از واگرایی KL استفاده شد.

۲-۳- یادگیری انتقالی و فرا یادگیری

در اکثر الگوریتم‌های یادگیری تقویتی، تغییرات کوچک حالت باعث شکست استراتژی‌های آموزش‌دیده قبلی می‌شود و آموزش از ابتدا گران است. یادگیری انتقالی[1] (TL) می‌تواند با به‌کارگیری تجربه‌ای که از کار منبع به دست می‌آید، کارایی آموزشی وظیفهٔ هدف را بهبود بخشد؛ بنابراین، برای بهبود سرعت یادگیری در وظایف جدید، یادگیری انتقالی برای یادگیری تقویتی تبدیل به یک جهت تحقیقاتی جذاب شده است. باتوجه‌به این که چه دانش منتقل می‌شود، یادگیری انتقالی می‌تواند به الگوریتم یادگیری تقویتی عمیق در جنبه های زیر کمک کند: شکل‌دادن به پاداش[2]، یادگیری از نمایش‌ها[3]، انتقال خط‌مشی[4]، نگاشت بین وظایف[5]، و انتقال نمایش[6]. لان و همکاران یک رویکرد یادگیری تقویتی انتقال جدید را از طریق استخراج فرا دانش با استفاده از درختان تصمیم

[1] Transfer learning
[2] reward shaping
[3] learning from demonstrations
[4] policy transfer
[5] intertask mapping
[6] representation transfer

هرس شده پیشنهاد کرده‌اند. ژانگ و همکاران روش DQDR[1] را پیشنهاد کردند که دانش دامنه را از انسان استخراج می‌کند و آن را به‌عنوان مجموعه‌ای از قوانین بیان می‌کند و آن را با یک DQN جفت می‌کند تا قابلیت انتقال الگوریتم‌های یادگیری تقویتی را بهبود بخشد.

هدف فرا یادگیری تعمیم دانش به وظایف جدید با یادگیری تجربیات از چندین کار دیگر است. ایده اصلی یادگیری فرا-تقویتی[2] این است که دانش قبلی آموخته شده از تعداد زیادی از وظایف یادگیری تقویتی را در وظایف یادگیری تقویتی جدید به کار گیرد و سرعت یادگیری و توانایی تعمیم عامل را بهبود بخشد. یادگیری فرا تقویتی عمدتاً کاستی‌های زیر یادگیری تقویتی عمیق را حل و بهینه می‌کند: استفاده کم از نمونه، مشکل در طراحی توابع پاداش، استراتژی‌های کاوش در وظایف ناشناخته، و فقدان توانایی تعمیم.

۳-۳- یادگیری تقویتی سلسله‌مراتبی

اغلب، ما خود را درحال‌توسعه محیط‌هایی می‌بینیم که با افق‌های متعدد مشکل دارند. به‌عنوان مثال، اگر می‌خواهیم یک عامل بهترین استراتژی سطح بالا را پیدا کند، اما فقط دستورات کنترل سطح پایین را برای اقدامات به او بدهد، در این صورت عامل باید یاد بگیرد که از فضای عمل سطح پایین به سطح بالا برود. به طور شهودی، یک سلسله‌مراتب در سیاست‌ها برای اکثر عامل‌ها وجود دارد. وقتی برنامه‌ریزی می‌کنم این کار را در فضای عمل سطح بالاتر انجام می‌دهم من به رفتن به فروشگاه فکر می‌کنم نه اینکه دستانم را برای رسیدن به فروشگاه تکان دهم. یادگیری تقویتی سلسله‌مراتبی، عوامل را قادر می‌سازد تا سلسله‌مراتبی از اقدامات را در داخل برای مقابله با مشکلات افق بلند ایجاد کنند. عوامل، دیگر در مورد دستورات چپ-راست استدلال نمی‌کنند، بلکه بیشتر در مورد رفتن به اینجا یا آنجا استدلال می‌کنند.

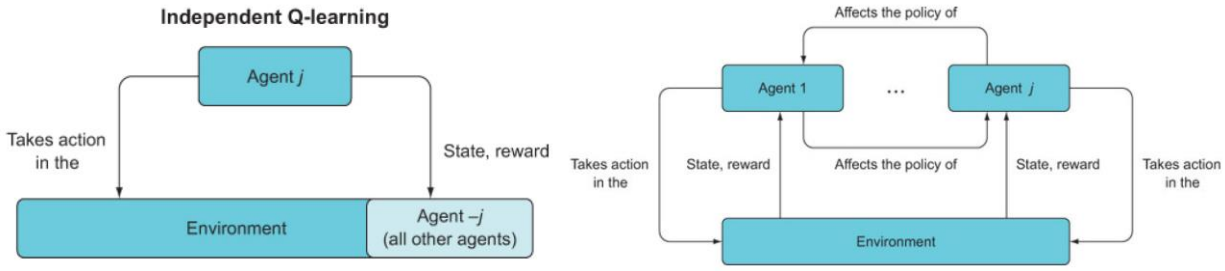
با کمک یادگیری عمیق، HRL می‌تواند مسائل مربوط به فضاهای حالت بزرگ و فضاهای عمل را حل کند و می‌تواند مستقیماً ورودی‌های با ابعاد بالا را پردازش کند. علاوه بر این، توانایی بازنمایی یادگیری عمیق، انتخاب‌های بیشتری را برای شکل اهداف فرعی فراهم می‌کند. یادگیری تقویتی عمیق سلسله‌مراتبی (HDRL) به یک جهت تحقیقاتی مهم یادگیری تقویتی عمیق تبدیل شده است.

الگوریتم‌های مطرح: HIRO و STRAW

[1] Deep Q-learning with transferable Domain Rules
[2] Meta-Reinforcement Learning

۴-۳- یادگیری تقویتی چندعاملی

یادگیری تقویتی چندعاملی[1] (MARL) یکی از مهم‌ترین زمینه‌های تحقیقاتی در یادگیری تقویتی است زیرا می‌تواند بسیاری از مشکلات تصمیم‌گیری مشارکتی پیچیده دنیای واقعی را برطرف کند. در بحث‌های قبلی، تنها الگوریتم‌های تک عامل یادگیری تقویتی موردبحث قرار گرفته‌لند. یک عامل یادگیری تقویتی منفرد که با محیط تعامل دارد نمی‌تواند تعداد زیادی از مشکلات کنترل را به طور کامل حل کند. بااین‌حال، چندین عامل یادگیری تقویتی که هم‌زمان با یک محیط در تعامل هستند، می‌توانند به یادگیری کارآمدتری دست یابند. MARL بر تجزیه‌وتحلیل و کنترل رفتارهای چندین عامل یادگیری تقویتی که در یک محیط مشترک وجود دارند متمرکز است. هر عاملی از عملکرد پاداش خود می‌آموزد و بر اساس منافع خود عمل می‌کند؛ این منافع فردی ممکن است با منافع برخی از عوامل دیگر در تضاد باشد و در نتیجه تعامل گروهی پیچیده شود. MARL یک پیچیدگی دیگر به مسئله یادگیری تقویتی اضافه می‌کند و آن معرفی غیرایستایی به دلیل تغییر در رفتار سایر عوامل در محیط پس از یادگیری است. یک شبکه ارتباطی کارآمد بین عوامل می‌تولند یک رویکرد مؤثر برای بهبود همکاری و یادگیری در MARL باشد.



شکل ۲۶-تعامل در محیط‌های چندعاملی(راست) و تعامل در یادگیری Q مستقل(چپ)

MARL یک موضوع مطالعاتی جذاب بوده است که برای برنامه‌های کاربردی دربازی‌های رایانه‌ای و روباتیک موردتوجه زیادی قرار گرفته است. محققان الگوریتم‌های MARL فوق‌العاده‌ای را در سال‌های اخیر پیشنهاد کرده‌اند که کاربرد خود را در زمینه‌های مختلف کاربردی یافته‌اند. تیم تحقیقاتی OpenAI از طریق یک مشکل پنهان و جستجوی چندعاملی و الگوریتم‌های یادگیری استاندارد RL دریافتند که عوامل می‌توانند یک استراتژی اقدام در حال تکامل خود نظارتی ایجاد کنند(محتوای پیشنهادی-ویدیو). در محیط، آنها شش مرحله در حال تکامل را در استراتژی مقابله با حریفان یافتند. عوامل در بهره‌برداری از کاستی های محیط برای دستیابی به پاداش‌های بهتر کاملاً موفق بودند.

[1] Multi agent reinforcement learning

یکی دیگر از افکار جذاب در هنگام بررسی یادگیری تقویتی چندعاملی این است که یادگیری تقویتی سلسله‌مراتبی را می‌توان به‌عنوان مورد دیگری از یادگیری تقویتی چندعاملی در نظر گرفت. چطور؟ به چندین عامل فکر کنید که در افق‌های مختلف تصمیم می‌گیرند. ساختار چند افق مشابه روشی است که اکثر شرکت‌ها در تجارت انجام می‌دهند. افرادی که در بالا هستند، اهداف سطح بالاتری را برای چند سال آینده برنامه‌ریزی می‌کنند و افراد دیگر تصمیم می‌گیرند که چگونه به‌صورت ماهانه و روزانه به این هدف برسند. آنهایی که در بالا هستند اهداف را برای کسانی که در پایین هستند تعیین می‌کنند. کل سیستم برای عملکرد همه عوامل پاداش می‌گیرد.

الگوریتم‌های مطرح: DDPG، MADDPG و Neighborhood Q-learning

۴- منابع

R. S. Sutton and A. Barto, *Reinforcement learning : an introduction(2nd ed.)*. Cambridge, Ma ; Lodon: The Mit Press, 2018.

Maxim Lapan, *Deep reinforcement learning hands-on : apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*. Birmingham ; Mumbai Packt January, 2020.

H. Dong, Z. Ding, and S. Zhang, *Deep Reinforcement Learning Fundamentals, Research and Applications*. Singapore: Springer Singapore : Imprint: Springer, 2020.

K. Doya, “Reinforcement Learning,” in *The Cambridge Handbook of Computational Cognitive Sciences*, R. Sun, Ed. Cambridge: Cambridge University Press, 2023, pp. 350–370

X. Wang *et al*., "Deep Reinforcement Learning: A Survey," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 5064-5078, April 2024, doi: 10.1109/TNNLS.2022.3207346

M. Morales and C. Isbell, *Grokking deep reinforcement learning*. Shelter Island, Ny: Manning Publications, 2020.

Alexander Alexander Zai, *Deep Reinforcement Learning in Action.* Manning Publications Company, 2020.

اسلایدهای درس یادگیری تقویتی دانشگاه صنعتی شریف

https://blog.faradars.org/an-introduction-to-reinforcement-learning/

https://parsinfotech.com/reinforcement-learning/

https://hooshio.com/?p=5198

https://intellabs.github.io/coach/components/agents/index.html

یادگیری تقویتی یا Reinforcement Learning و کاربردهایش چیست؟ | کافه‌تدریس(cafetadris.com)

محتواهای پیشنهادی:

https://www.youtube.com/@aiwarehouse

https://www.youtube.com/watch?v=kopoLzvh5jY

https://cs.stanford.edu/people/karpathy/reinforcejs/

پیوست: پیاده‌سازی‌ها

در این بخش چند نمونه از پیاده‌سازی‌های الگوریتم‌های یادگیری تقویتی ارائه می‌شود(به‌صورت فایل).