فصل سیزدهم: یادگیری تقویتی

یادگیری تقویتی به مسئله‌هایی می‌پردازد که در آن یک عامل مستقل حالت‌هایی را درک کرده و مطابق با آن ادراک اعمال بهینه‌ای را برای رسیدن به اهدافش را انجام می‌دهد. این مسئله‌ی بسیار جامع است و شامل مسائل یادگیری کنترل ربات‌های متحرک، یادگیری بهینه‌سازی کارخانه‌ها، و یادگیری بازی‌های صفحه‌ای[۱] می‌شود. هر گاه که عامل عملی را در محیطش انجام می‌دهد، یک معلم متناسب با حالت و عمل انجام شده به وی پاداش می‌دهد یا وی را تنبیه می‌کند (پاداش منفی). برای مثال، معلم ممکن است در یک بازی برای برد پاداش مثبت و برای باخت پاداش منفی و برای اعمال دیگر پاداش صفر را در نظر بگیرد. کار عامل، یادگیری از این پاداش‌ها (که گاهی تأخیر نیز دارد) است تا در اعمال بعدی بیشترین میزان تابع تجمعی پاداش را بگیرد. در این فصل بیشتر بر روی الگوریتمی به نام یادگیری Q که پاداش‌های تأخیری را نیز در نظر می‌گیرد تمرکز می‌کنیم. این الگوریتم حتی زمانی که عامل هیچ اطلاعاتی در مورد محیط ندارد درست کار می‌کند. الگوریتم‌های یادگیری تقویتی رابطه‌ی نزدیکی با الگوریتم‌های برنامه‌نویسی پویا[۲] دارند، که کاربرد بسیاری در مسائل بهینه‌سازی دارد.

۱۳,۱ معرفی

فرض کنید که رباتی یادگیر داریم. این ربات، یا عامل[۳]، دسته حسگری برای مشاهده‌ی حالت[۴] محیط دارد، و دسته‌ای از اعمال[۵] را می‌تواند برای تغییر حالت انجام دهد. برای مثال، یک ربات متحرک ممکن است حسگرهایی چون دوربین و اعمالی چون "حرکت به سمت جلو" و "چرخش"

---

[۱] board games
[۲] dynamic programming
[۳] agent
[۴] state
[۵] action

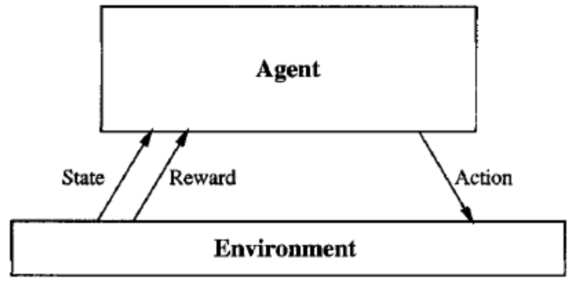
داشته باشد. هدف این ربات یادگیری خطمشی[۱] یا متدی برای کنترل اعمال است که بتواند با استفاده از آن‌ها به اهداف خود برسد. برای مثـال، ممکن است هدف ربات اتصال به شارژر در هنگام کمبود شارژ باشد.

در این فصل به چگونگی یادگیری استراتژی کنترل بهینه‌این عامل‌ها می‌پردازیم. فرض می‌کنیم که می‌توان اهداف عامـل را بـا تـابعی حقیقـی مقدار به نام "تابع پاداش"[۲] مشخص کرد. این تابع به هر عمل در هر حالت عددی را نسبت می‌دهد. مثلاً برای مثال، برای اتصال بـه شـارژر در حالتی شارژ باتری کم می‌تواند مقدار مثبتی (مثلاً 100+) و در برای اتصال به شارژر و در بقیه‌ی حالت‌ها مقدار صفر داشته باشد. تابع پـاداش را می‌توان به در قسمتی از ربات و یا عاملی خارجی مثل معلمی که برای اعمال پاداش می‌دهـد تعریـف کـرد. کـار ربـات انجـام سـری اعمـالی و مشاهده‌ی نتیجه‌ی آن‌ها در محیط و یادگیری استراتژی کنترل است. استراتژی کنترل این است که در هر حالت اولیـه ربـات بتوانـد اعمـالی را انتخاب کند تا به بیشترین پاداش برسد. این تعریف به طور خلاصه در شکل ۱۳,۱ آورده شده است.

همان‌طور که در شکل ۱۳,۱ نیز پیداست، مسئله‌ی یادگیری خطمشی‌ای برای حداکثر کردن پاداش تجمعی[۳]، خیلی کلی است و مسائلی خارج از محدوده‌ی یادگیری ربات را نیز در بر می‌گیرد. در کل، مسئله یادگیری کنترل سری اعمال، در هر حالت است. این مسـئله ممکـن اسـت، بـرای مثال، مسئله‌ی بهینه کردن اعمال یک تولیدکننده برای حداکثر کردن سود کارخانه باشد، در این مثال تابع پاداش می‌تواند قیمت کالاهای تولیـد شده منهای مبالغ مصرفی باشد. یا ممکن است مسئله‌این باشد که شرکت تاکسی بی‌سیم برنامه‌ی تاکسی‌های خود را چگونه در یک شهر بزرگ برنامه‌ریزی کند و تابع پاداش کم شدن زمان منتظر ماندن مسافرین و سوخت مصرفی تاکسی‌ها است. در کل هدف یادگیری خطمشی‌ای بـرای هر نوع عاملی است که اعمالش بر روی محیط تأثیر می‌گذارد و تابعی تجمعی برای کیفیت هر عمل در دسترس دارد. به همراه این نوع مسـائل معمولاً شرایطی در نظر گرفته می‌شود، مثلاً اینکه اعمال همیشه قطعی[۴] هستند یا نه، یا اینکه عامل از قبل اطلاعاتی را درباره‌ی اعمـالش دارد یـا خیر.

---

[۱] policy
[۲] reward function
[۳] cumulative
[۴] deterministic



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \text{ , where } 0 \leq \gamma < 1$$

شکل ۱۳،۱ عاملی در ارتباط با محیطش. عاملی در محیطی است که حالتش یکی از اعضای S است.
عامل می‌تواند اعمال مجموعه‌ی A را انجام دهد. هر بار که عملی مثل a_t را در حالت s_t انجام می‌دهد مقدار حقیقی‌ای r_t را از تابع پاداش دریافت می‌کند که حاصل لحظه‌ای این عمل است. این فرایند سری‌ای از s_i و a_iها و r_iها را ایجاد می‌کند. هدف عامل این است که خط‌مشی‌ای به فرم $\pi: S \rightarrow A$ را یاد بگیرد که مجموع این پاداش با پاداش‌های آینده در سری توانی حداکثر شود.

توجه دارید که قبلاً نیز مسائل یادگیری سری اعمال را در این کتاب دیده بودیم. در بخش ۱۱،۴ درباره‌ی یادگیری توضیحی دسته قوانین برای کنترل جستجو را در حین حل مسئله بحث کردیم. مسئله‌ی مطرح برای عامل در آن بخش انتخاب بین گزینه‌های مختلف عمل در هر مرحله از جستجو برای یافتن حالت هدف بود. تفاوت تکنیک‌هایی که در اینجا به کار می‌بریم با تکنیک‌های بخش ۱۱،۴ در این است که مسائلی را در نظر می‌گیریم که اعمال ممکن است نتایج غیرقطعی داشته باشند و یادگیر نیز تئوری قلمروی‌ای درباره‌ی غیرقطعی بودن نتایج اعمال ندارد. در فصل ۱ مسئله‌ی یادگیری انتخاب‌های اعمال بازی چکرز را مورد بحث قرار دادیم. در آنجا طراحی متد یادگیری‌ای بسیار مشابه متدهای این فصل را طراحی کردیم. در واقع، یکی از موفق‌ترین کاربردهای الگوریتم‌های یادگیری تقویتی این فصل برای مسئله‌ی طرز–بازی[۱]، بازی مشابهی است. (Tesauro 1995) برنامه‌ی TD-Gammon را که با روش یادگیری تقویتی برای رسیدن به بازیکن جهانی برای بازی تخته‌نرد آموزش دیده را معرفی می‌کند. این برنامه بعد ۱،۵ میلیون بازی خودساخته[۲]، حال در حد بهترین بازیکن‌های جهانی شمرده می‌شود و در مقابل بسیاری از بازیکنان سطح بالا در مسابقات جهانی تخته‌نرد بازی می‌کند.

مسئله‌ی یادگیری خط‌مشی انتخاب اعمال از جنبه‌هایی شبیه مسائل تخمین توابع که در فصل‌های گذشته بررسی کرده‌ایم است. در این تشابه تابع هدفی که باید یاد گرفته شود، تابع $\pi: S \rightarrow A$ است که برای هر حالت s از مجموعه‌ی S، یک عمل مثل a از مجموعه‌ی A را نظیر می‌کند. با وجود این تشابه مسئله‌ی یادگیری تقویتی با مسائل تخمین توابع در بسیاری از جنبه‌ها متفاوت است:

[۱] game-playing
[۲] self-generated

- پاداش تأخیری[۱]. هدف یادگیری، یادگیری تابع هدفی مثل π که هر حالت s را به عملی ربط دهد $a= \pi(s)$. در فصول قبلی، همیشه فرض بر این بوده که تابع هدفی چون π را از نمونه‌های آموزشی‌ای به فرم $<s, \pi(s)>$ یاد می‌گیریم. در حالی که در یادگیری تقویتی اطلاعات آموزشی به این فرم موجود نیست و به جای آن مربی[۲] سری پاداش‌هایی لحظه‌ای نظیر حرکات را به عامل می‌دهد. پس عامل با مسئله‌ی نسبت دادن ارزش موقتی[۳] مواجه است: تصمیم‌گیری برای اینکه کدام یک از اعمال در سری اعمال احتمالاً بعداً باعث پاداش خواهند داد.
- جستجوی محیط[۴]. در یادگیری تقویتی، عامل با انتخاب اعمال بر توزیع نمونه‌های آموزشی تأثیر می‌گذارد. همین تأثیر سؤالی دیگر را مطرح می‌کند، کدام استراتژی در انتخاب آزمایش‌ها[۵] آموزش بهینه‌تری را فراهم می‌کند؟ یادگیر دو انتخاب خواهد داشت، یکی اینکه حالت‌ها و اعمال جدید را جستجو کند (تا اطلاعات جدیدی به دست آورد) و دیگر اینکه به اطلاعاتی که تا به حال پیدا کرده اکتفا کند و از آن‌ها برای رسیدن به بیشترین پاداش استفاده کند[۶].
- حالت‌های نیمه معلوم. با وجود اینکه راحت‌تر است فرض کنیم حسگرهای عامل می‌توانند تمام شرایط محیط را در یک حالت را معلوم کنند، اما با این حال در بسیاری از مثال‌های کاربردی حسگرها اطلاعات کاملی در مورد محیط به ما نمی‌دهند. مثلاً، در مثال ربات زمانی که از یک دوربین روبه‌جلو استفاده می‌شود اجسام پشت سر ربات توسط دوربین مشخص نمی‌شوند. در چنین شرایط بهتر است که در هنگام انتخاب عمل، عامل شرایط مشاهده شده‌ی قبلی را نیز در نظر بگیرد، شاید بهترین خط‌مشی انتخاب اعمالی باشد که میزان مشاهده‌ی ربات را از محیط معلوم کند.
- یادگیری مادام‌العمر[۷]. بر خلاف تخمین توابع ، در یادگیری رباتیک گاهی نیاز است که ربات چندین کار مرتبط با هم را در همان محیط و با همان سنسورها یاد بگیرد. برای مثال ممکن است نیاز باشد که یک ربات متحرک علاوه بر وصل شدن به باتری، نحوه‌ی حرکت در راهروهای باریک و نحوه‌ی برداشتن خروجی پرینتر را نیز یاد بگیرد. این باعث می‌شود که آزمایش‌ها قبلی و دانش به دست آمده در حالت‌های قبلی را بتواند در یادگیری کارهای جدید به کار ببرد.

۱۳,۲ یادگیری کارها

در این قسمت دقیق‌تر و با دید ریاضی به مسئله‌ی یادگیری سری استراتژی‌های کنترل می‌پردازیم. توجه داشته باشید که راه‌های بسیاری برای این بررسی وجود دارد. برای مثال، ممکن است فرض کنیم که اعمال قطعی یا غیرقطعی هستند. یا ممکن است فرض کنیم که عامل می‌تواند حالت بعد از هر عمل را پیش‌بینی کند و یا در نقطه‌ی مقابل حالت‌ها غیرقابل‌پیش‌بینی هستند. یا حتی ممکن است فرض کنیم که عامل توسط یک معلم آموزش داده می‌شود که تمامی راه‌های بهینه را برای انجام اعمال نشان می‌دهد، یا در نقطه‌ی مقابل خود ربات باید با انجام اعمال

---

۱ delayed reward
۲ trainer
۳ temporal credit assignment
۴ exploration
۵ experimentation strategy
۶ exploitation
۷ life-long learning

کارها را یاد بگیرد. در اینجا ما فرمولی کلی را که از فرایندهای تصمیم‌گیری مارکوف[۱] به دست آمده ارائه می‌کنیم. ایـن فرمـول در شـکل ۱۳,۱ آورده شده است.

در یک فرایند تصمیم‌گیری مارکوف (MDP) عامل مجموعه‌ای از حالت‌ها به نام S و مجموعه‌ای از اعمال به نـام A را در اختیـار دارد. در هـر لحظه‌ی t، حسگرهای عامل حالت s_t را مشخص می‌کنند و عامل عمل a_t را انجام می‌دهد. محیط نیز در مقابل پـاداش $r_t = r(s_t, a_t)$ را به عامل می‌دهد و حالت $s_{t+1} = \delta(s_t, a_t)$ را ایجاد می‌کند. در اینجا دو تابع r و δ جزو محیط هستند و الزاماً بـرای عامـل مشـخص نیستند. در یک MDP توابع $r(s_t, a_t)$ و $\delta(s_t, a_t)$ فقط به حالت فعلی و عمل وابسته‌اند و به حالت‌ها و اعمال قبلـی هـیچ وابسـتگی‌ای ندارند. در این فصل فقط حالت‌هایی را که در آن S و A متناهی هستند را بررسی می‌کنیم. در کل، دو تابع r و δ ممکن اسـت توابـع قطعـی‌ای نباشند اما در ابتدا فرض می‌کنیم که این توابع قطعی‌اند.

کار عامل یادگیری خطمشی‌ای چون π: S → A است که بتواند حرکت بعدی a_t را بر اساس حالت فعلـی s_t بـه دسـت بیـاورد؛ داریـم کـه $\pi(s_t) = a_t$. اما چگونه می‌توانیم دقیقاً مشخص کنیم که عامل کدام خطمشی را برای π یاد بگیرد؟ یکی از راه‌حل‌های بسیار سـاده تعیـین خطمشی بهینه، تعریف آن به صورتی است که تابع تجمعی پاداش در طول زمان را حداکثر کند. برای تعریف دقیق‌تر مقدار تجمعی $V^\pi(s_t)$ را که توسط خطمشی π از حالت اولیه‌ی s_t به دست می‌آید را به فرم زیر تعریف می‌کنیم:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^{\mathrm{i}} \mathrm{r}_{\mathrm{t+i}} \qquad (13.1)$$

در این رابطه سری ی $\mathrm{r}_{\mathrm{t+i}}$ پاداش‌هایی هستند که از خطمشی π و شروع از s_t به دست می‌آید (($a_t = (s_t)$و $a_{t+1} = (s_{t+1})$ و...). در این رابطه $0 \leq \gamma < 1$ ثابتی برای پاداش‌های تأخیری است. در کل، تأثیر پاداشی که i مرحله بعد از انجام عمل داده می‌شود توسـط ضـریب توانی γ^i کوچک می‌شود. توجه داشته باشید که اگر γ =0، فقط پاداش لحظه‌ای در نظر گرفته خواهد شد. با بیشتر کردن مقدار γ تأثیر نسـبی پاداش‌های تأخیری در تابع بیشتر می‌شود.

کمیت $V^\pi(s_t)$ که در رابطه‌ی ۱۳,۱ تعریف شده گاهی پاداش تجمعی تخفیفی[۲] برای خطمشی π در حالت s نیز نامیده می‌شود. منطقی است که پاداش‌های آینده را تخفیف دهیم، زیرا که در بسیاری از موارد هدف ما بر سریع‌تر رسیدن به پاداش است. با این وجود، در بعضی موارد از کل پاداش دریافتی نیز استفاده می‌شود. برای مثال، finite horizon reward به فرم $\sum_{i=0}^{\mathrm{h}} \mathrm{r}_{\mathrm{t+i}}$ تعریف می‌شود که پاداش‌ها را تا h پلـه‌ی بعدی موردنظر قرار می‌دهد. یا متوسط پاداش[۳] به صورت $\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$ تعریف می‌شود که متوسط کل پاداش دریـافتی در طـول عمـر ربات را در نظر می‌گیرد. در این فصل ما خود را محدود به رابطه‌ی 13.1 می‌کنیم. برای پاداش متوسط به (Mahadevan 1996) مراجعه کنید.

---

[۱] Markov decition processes
[۲] discounted cumulative reward
[۳] average reward

حال به جایی رسیده‌ایم که کار یادگیری عامل را دقیق مشخص کنیم. باید خط‌مشی‌ای را پیدا کنیم تا برای تمامی حالت‌های s مقدار $V^{\pi}(s_t)$ ماکزیمم شود. چنین خط‌مشی‌ای را خط‌مشی بهینه[۱] می‌نامیم و با π^* نشان می‌دهیم.

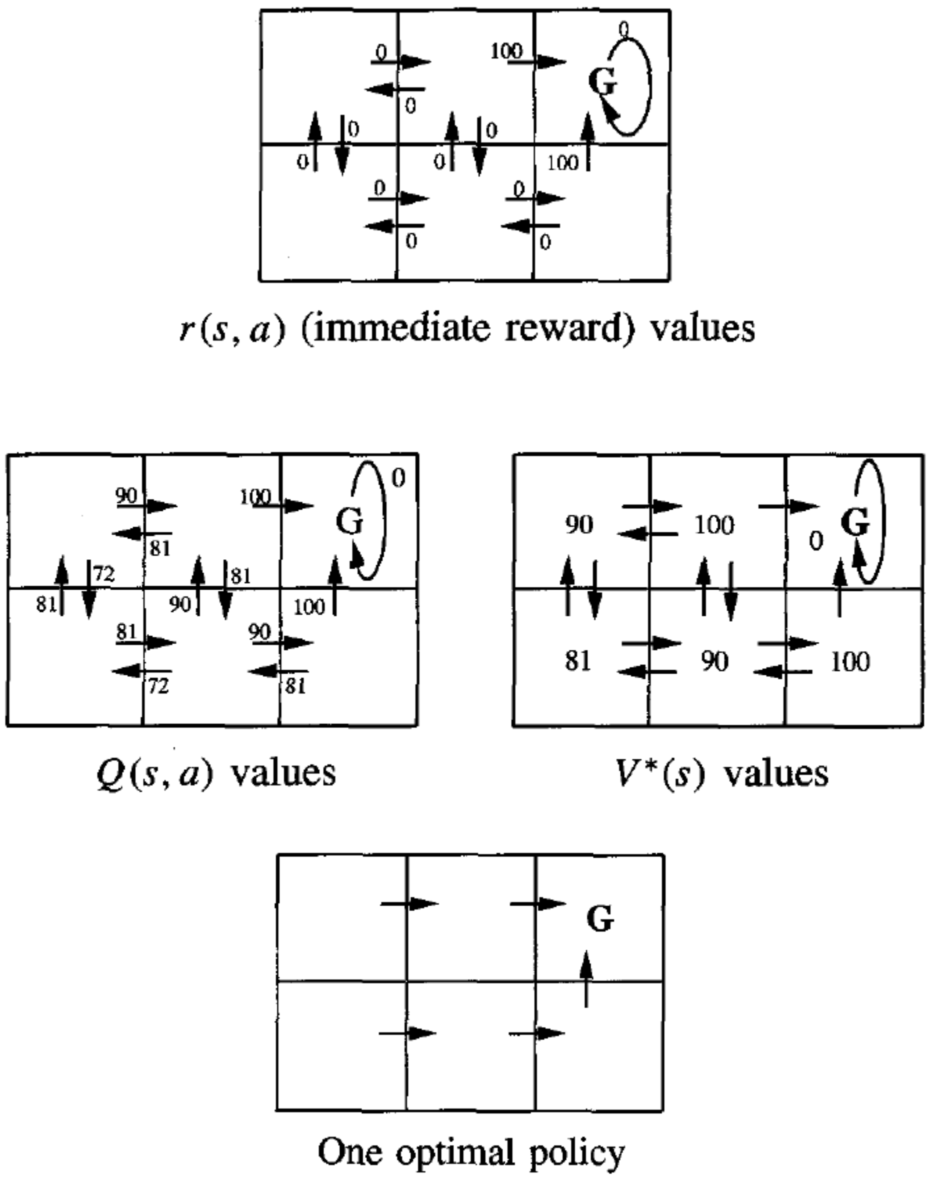
$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

برای ساده‌سازی نمایش، تابع $V^{\pi^*}(s)$ را به صورت $V^*(s)$ نمایش می‌دهیم. $V^*(s)$ بیشترین مقدار ممکن پاداش تجمعی تخفیفی را برای هر حالت s می‌دهد؛ به عبارت دیگر، این مقدار، مقدار خط‌مشی بهینه برای حالت s است.

برای درک این مفاهیم، قسمت اول شکل ۱۳٫۲ را که محیطی گسسته را نشان داده در نظر بگیرید. شش مربع نشان داده شده در شکل نشان‌دهنده‌ی شش حالت یا موقعیت برای عامل هستند. هر فلش در شکل یک عمل ممکن که عامل می‌تواند برای تغییر حالت خود انجام دهد را نشان می‌دهد. هر عدد روی فلش مقدار پاداش r(s,a) را که عامل پس از انجام هر عمل می‌گیرد را نشان می‌دهد. توجه داشته باشید که تمامی پاداش‌های لحظه‌ای در این مثال جز برای عملی که به خانه‌ی G ختم می‌شود صفر است. می‌توان حالت G را به عنوان حالت هدف در نظر گرفت زیرا که فقط با ورود به خانه‌ی G عامل پاداش دریافت می‌کند. توجه داشته باشید که در این محیط خاص تنها اعمالی که برای حالت G در نظر گرفته شده باقی ماندن در همان حالت است. به همین دلیل حالت G را حالت جاذب[۲] می‌نامیم.

با معلوم بودن تمامی حالت‌ها، اعمال و پاداش‌ها می‌توان به راحتی با تعیین مقدار γ خط‌مشی بهینه π^* و تابع $V^*(s)$ آن را مشخص کرد. در این حالت بیایید فرض کنیم که γ=0.9. نمودار وسطی یکی از خط‌مشی‌های بهینه را برای این شرایط مشخص می‌کند. مثل تمامی خط‌مشی‌ها، این خط‌مشی نیز در هر حالت فقط یک عمل را پیشنهاد می‌کند. جای تعجب ندارد که خط‌مشی بهینه کوتاه‌ترین راه را به سمت حالت G نشان می‌دهد.

[۱] optimal policy
[۲] absorbing state



شکل ۱۳,۲ یک محیط قطعی برای تصور مفاهیم اولیه‌ی یادگیری Q.

هر مربع یک حالت و هر فلش یک عمل را نشان می‌دهند. تابع پاداش، r(s,a) در ورود به حالت G 100+ و در بقیه‌ی موارد صفر است. مقادیر $V^*(s)$ و Q(s,a) که از r(s,a) با $\gamma = 0.9$ به دست آمده است. خط‌مشی بهینه‌ای نیز با مقادیر Q ماکزیمم نیز در شکل آمده است.

شکل سمت راست در شکل ۱۳,۲ مقادیر V^* را برای هر حالت نشان می‌دهد. برای مثال، مربع گوشه‌ی پایین و راست شکل را در نظر بگیرید. مقدار V^* برای این مربع ۱۰۰ است زیرا که خط‌مشی برای این مربع فلش رو به بالا را انتخاب می‌کند و عامل پاداش لحظه‌ای ۱۰۰ را می‌گیرد و پس از آن نیز عامل در حالت جاذب می‌ماند و هیچ پاداش دیگری نیز نمی‌گیرد. مشابهاً، مقدار V^* برای مربع پایین و وسط ۹۰ است. این به خاطر این است که خط‌مشی بهینه ابتدا عمل به سمت راست رفتن را با پاداش لحظه‌ای صفر و سپس عمل بالا رفتن را با پاداش لحظه‌ای ۱۰۰ انجام می‌دهد. پس به سادگی برای این مربع با توجه به رابطه‌ی V^* می‌توان نوشت:

$$0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \cdots = 90$$

در این محیط خاص بعد از رسیدن به حالت جاذب G پاداش اعمال بعدی صفر خواهد بود و عامل دیگر پاداشی دریافت نمی‌کند.

۱۳,۳ یادگیری Q

چگونه عامل می‌تواند برای محیطی دلخواه خطمشی بهینه π^* را یاد بگیرد؟ یادگیری مستقیم $\pi^*: S \to A$ ممکن نیست زیرا که نمونه‌هایی آموزشی به شکل <s,a> نداریم و بجای آن تنها مقادیر پاداش‌ها را به فرم $r(s_i, a_i)$ برای i=0,1,2,3,... داریم. همان‌طور که بعداً نیز خواهیم دید با داشتن چنین اطلاعاتی یادگیری تابع تخمینی عددی بر روی حالت‌ها و اعمال ساده‌تر است. بعد از پیدا کردن تابع تخمین عددی، با استفاده از آن خطمشی بهینه را پیدا می‌کنیم.

چه تابع تخمینی را باید عامل یاد بگیرد؟ یکی از انتخاب‌های آشکار خود V^* است. عامل باید حالت s_1 را هر گاه که $V^*(s_1) > V^*(s_2)$ به حالت s_2 ترجیح دهد. البته خطمشی بین اعمال حق انتخاب دارد نه بین حالت‌ها. با این وجود می‌توان برای انتخاب بین اعمال نیز از V^* استفاده کرد. عمل بهینه در حالت s عملی مثل a است که مجموع پاداش‌های لحظه‌ای و V^* حالت پایانی‌اش ضرب در γ ماکزیمم باشد:

$$\pi^*(s) = arg \max_a \left[r(s,a) + \gamma V^*\big(\delta(s,a)\big)\right] \quad (13.3)$$

(توجه داشته باشید که $\delta(s,a)$ نماد حالت بعدی برای عمل a از حالت s تعریف شد). بنابراین عامل با استفاده از V^* و اطلاعات کافی در مورد تابع پاداش‌های لحظه‌ای r و تابع حالت‌های پایانی δ می‌تواند خطمشی بهینه را مشخص کند. پس زمانی که عامل دو تابع δ و r را که محیط برای تغییرات استفاده می‌کند را بداند می‌تواند با استفاده از رابطه‌ی ۱۳,۳ عمل بهینه را برای حالت دلخواه s مشخص کند.

متأسفانه، با یادگیری V^* فقط زمانی می‌توان خطمشی بهینه را مشخص کرد که عامل دو تابع δ و r را بداند. چنین چیزی معادل دانستن نتایج لحظه‌ای (هم پاداش‌های لحظه‌ای و هم حالت‌های پایانی) برای هر زوج مرتب حالت عمل است. این فرض مشابه داشتن تئوری قلمروی کامل در یادگیری توضیحی (فصل ۱۱) است. در بسیاری از مسائل کاربردی، مثل کنترل ربات، پیش‌بینی دقیق نتیجه‌ی هر حرکت در هر حالت برای عامل و یا برنامه‌نویسش غیرممکن است. برای مثال، فرض کنید که بازوی بیل شکل رباتی می‌خواهد مقدار خاک‌برداری کند، و حالت نهایی مکان ذرات خاک بعد از خاک‌برداری است. در چنین شرایطی هم δ و هم r نامعلوم‌اند، و متأسفانه یادگیری V^* مفید نخواهد بود زیرا که عامل نمی‌تواند رابطه‌ی ۱۳,۳ را کامل کند. عامل باید از چه تابع تخمینی برای این تعریف مسئله‌ی کلی‌تر استفاده کند؟ تابع تخمین Q، که در بخش بعدی تعریف خواهد شد، جوابی برای این مسئله ارائه می‌کند.

۱۳,۳,۱ تابع Q

فرض کنید که تابع تخمینی Q(s,a) را به صورتی تعریف کرده‌ایم که مقدارش همیشه ماکزیمم ممکن تابع پاداش تجمعی تخفیفی برای هر حالت s و عمل a در گام اول است. به عبارت دیگر، مقدار عبارت Q همیشه جمع مقدار پاداش لحظه‌ای عمل a و پاداش خطمشی بهینه‌ی بعد از آن (با تخفیف γ) است:

$$Q(s,a) \equiv r(s,a) + \gamma V^*\big(\delta(s,a)\big) \quad (13.4)$$

توجه داشته باشید که Q(s,a) دقیقاً همان کمیتی است که در رابطه‌ی ۱۳,۳ برای انتخاب a در حالت s ماکزیمم شده است. بنابراین با بازنویسی رابطه خواهیم داشت:

$$\pi^*(s) = arg\max_a Q(s,a) \tag{13.3}$$

اهمیت این بازنویسی در چیست؟ این رابطه نشان می‌دهد که اگر عامل تابع Q را به جای V^* یاد بگیرد بدون نیاز به دو تـابع r و δ مـی‌توانـد اعمال بهینه را پیدا کند. همان‌طور که در رابطه‌ی ۱۳،۵ نیز آمده است، کافی است با در نظر گرفتن حالت‌های مختلف برای a در حالت s عملی را انتخاب کنیم که Q(s,a) را ماکزیمم می‌کند.

در ابتدا ممکن است عجیب به نظر برسد که با ماکزیمم کردن مقداری موضعی مثل Q می‌توان با تکرار اعمال به بیشترین پاداش کلی رسـید. و این بدین معناست که عامل بدون اینکه حتی در نظر بگیرد که بعد از این عمل چه حالت رخ خواهد داد می‌تواند عمل بهینه را پیدا کنـد. زیبـایی یادگیری Q[۱] در این است که تابع Q طوری تعریف شده که تمامی اطلاعات لازم درباره‌ی تابع پاداش تجمعی تخفیفی در آینده با انتخـاب عمـل a در حالت s را در خود داراست.

برای درک بهتر، شکل ۱۳،۲ مقادیر تابع Q را برای هر حالت در محیط مربعی تعریف شده نشان می‌دهد. توجه داشته باشید که مقـدار Q بـرای هر زوج مرتب حالت عمل مجموع مقدار پاداش لحظه‌ای و مقدار تابع V^* را با تخفیف γ نشـان مـی‌دهـد. همچنـین توجـه داشـته باشـید کـه خط‌مشی بهینه نیز با مقادیر Q برای حالت‌های مختلف متناسب است.

۱۳،۳،۲ الگوریتمی برای یادگیری Q

یادگیری Q ارتباطی مستقیم با پیدا کردن خط‌مشی بهینه دارد. اما چگونه می‌توان Q را یاد گرفت؟

کلید حل این مشکل پیدا کردن راهی قابل‌اطمینان برای تخمین مقادیر آموزشی برای Q، از طریق تنها داده‌های موجود یعنی سری پاداش‌هـای r در طول زمان است. می‌توان با تخمین تکراری‌ای به چنین چیزی دست یافت. برای پی بردن به چگونگی این امر، بیایید به رابطه‌ی بین Q و V^* نگاهی دقیق‌تر بیندازیم،

$$V^*(s) = \max_{a'} Q(s,a')$$

که با توجه به رابطه‌ی ۱۳،۴ خواهیم داشت:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a') \tag{13.6}$$

این تعریف بازگشتی Q پایه‌ی الگوریتم‌های حلقه‌ای برای تخمین Q است (Watkins 1989). بـرای توضـیح الگـوریتم، از نمـاد $\hat{Q}$ بـرای نمایش تخمین یا همان فرضیه‌ی یادگیر استفاده می‌کنیم، و نماد Q را برای تابع اصلی به کار می‌بریم. در این الگوریتم یادگیر فرضیه‌ی $\hat{Q}$ را با جدولی بزرگ که برای هر جفت حالت و عمل عددی آورده شده نشان می‌دهـد. جـدول مقـادیر <s,a> مقـادیر را بـرای $\hat{Q}(s,a)$ نگـه‌داری می‌کند، فرضیه‌ی فعلی یادگیر که تخمینی از Q اصلی است. جدول در گام اول با مقادیر تصادفی پر می‌شود (اگر فرض کنیم جدول در گـام اول با صفر پر می‌شود درک الگوریتم راحت‌تر خواهد بود). در هر مرحله عامل حالت s را مشاهده می‌کند و عملی چون a را انجام مـی‌دهـد، سـپس

---

[۱] Q learning

پاداش ناشی از عمل r=r(s,a) و حالت پایانی s′=δ(s,a) را مشاهده می‌کند. سپس مقدار $\hat{Q}(s,a)$ داخل جدول را برای چنین حالتی به فرم زیر تغییر می دهد:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \qquad (13.7)$$

توجه دارید که عامل از مقدار فعلی $\hat{Q}$ در s′ برای تخمین $\hat{Q}$ در مرحله‌ی قبلی استفاده می‌کند. با وجود اینکه این رابطه به تخمین قبلی عامل از $\hat{Q}$ نیز وابسته است اما می‌توان گفت این قانون آموزش از رابطه‌ی ۱۳٫۶ نشأت گرفته است. توجه داشته باشید که با وجود اینکه رابطه‌ی ۱۳٫۶ Q را بر اساس دو تابع δ(s,a) و r(s,a) توصیف می‌کند، عامل برای پیاده‌سازی رابطه‌ی ۱۳٫۷ هیچ نیازی به دانستن این دو رابطه ندارد. به جای آن عامل عمل مذکور را انجام می‌دهد و حالت جدید s′ و پاداش r را مشاهده می‌کند. پس می‌توان گفت که در این رابطه دو مقدار s′ و r این دو تابع را مدل‌سازی می‌کنند.

الگوریتمی که در بالا توضیح داده شد برای سیستم‌های تصمیم‌گیری مارکوف دقیق‌تر در جدول ۱۳٫۱ آورده شده است. با استفاده از این الگوریتم تخمین عامل $\hat{Q}$ به مقدار واقعی Q میل می‌کند. با توجه به اینکه فرض کردیم سیستم یک مدل تصمیم‌گیری مارکوف است، تابع پاداش کران‌دار است و اعمال طوری انتخاب می‌شوند که هر جفت حالت و عمل چند دفعه یک بار مشاهده شود.

الگوریتم یادگیری Q

برای هر جفت s وa مقدار اولیه‌ی جدول $\hat{Q}(s,a)$ را صفر قرار بده

حالت فعلی s را مشاهده کن

حلقه‌ی زیر را تا بی‌نهایت ادامه بده:

- عملی مثل a را انتخاب کن و آن را انجام بده.
- مقدار پاداش لحظه‌ای r را دریافت کن
- حالت جدید s′ را مشاهده کن
- مقدار جدول را با رابطه‌ی زیر تغییر بده

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

- $s \leftarrow s'$

جدول ۱۳٫۱

الگوریتم یادگیری Q با فرض اینکه پاداش‌ها و اعمال قطعی هستند. ثابت تخفیف $\mathbf{0 \le \gamma < 1}$ در نظر گرفته شده است.

۱۳٫۳٫۳ مثالی توضیحی

برای تصور بهتر یادگیری Q، عمل و حالت‌های نشان داده شده در شکل ۱۳٫۳ را در نظر بگیرید. در اینجا یک عامل عملی را انجام داده و $\hat{Q}$ را با توجه به این عمل تغییر می‌دهد. عامل در این مثال به سمت راست حرکت کرده (عمل) و برای این عمل پاداش صفر را دریافت می‌کند.

سپس با توجه به رابطه‌ی ۱۳،۷ مقدار تخمینی خود $\hat{Q}$ را برای زوج حالت و عملی که انجام داده تغییر می‌دهد. با توجه به رابطه مقدار جدید $\hat{Q}$ جمع پاداش لحظه‌ای عمل (صفر) و بیشترین مقدار $\hat{Q}$ برای حالت پایانی (۱۰۰) با تخفیف γ، ۰،۹ خواهد بود.

در هر تکرار عامل عملی را انجام داده و از حالت قدیمی به حالت جدید می‌رود، و یادگیری Q نیز در هر مرحله $\hat{Q}$ را از مرحله‌ی جدید به مرحله‌ی قبلی گسترش می‌دهد به طور همزمان میزان پاداش لحظه‌ای توسط عامل دریافت شده و در تغییر $\hat{Q}$ تأثیر خود را می‌گذارد.

فرض کنید که این الگوریتم را برای محیط مربعی و تابع پاداش نشان داده شده در شکل ۱۳،۲ به کار ببریم. از آنجایی که این محیط یک حالت جاذب دارد باید آموزش را در چند قسمت انجام دهیم. در هر قسمت عامل در حالتی تصادفی قرار می‌گیرد و تا رسیدن به حالت جاذب حق انتخاب اعمال را خواهد داشت. زمانی که عامل به حالت جاذب می‌رسد این قسمت از آموزش پایان می‌یابد و عامل دوباره به حالتی تصادفی فرستاده می‌شود تا قسمت بعدی آموزش انجام شود.

مقادیر $\hat{Q}$ چگونه در این مثال با استفاده از یادگیری Q تغییر می‌کنند؟ در ابتدای آموزش همه‌ی مقادیر $\hat{Q}$ صفر قرار داده می‌شوند و تا زمانی که عامل به حالت جاذب نرسیده و پاداشی غیر صفر دریافت نکرده این مقادیر صفر می‌مانند. با ورود به حالت جاذب، حالتی که عامل قبلاً در آن بوده مقدار $\hat{Q}$ خود را تغییر می‌دهد. در قسمت بعدی آموزش اگر عامل از حالتی که گفته شد رد شود مقدار غیر صفر $\hat{Q}$ در آن خانه باعث می‌شود تا خانه‌ای با فاصله‌ی دو از G نیز مقدار $\hat{Q}$ خود را پیدا کند و به همین ترتیب. اگر تعداد قسمت‌های آموزشی کافی باشد، اطلاعات لازم از طریق پاداش‌های غیر صفر در میان فضای جفت حالت عمل پخش خواهد شد و در آخر نیز به جدولی که در شکل ۱۳،۲ نیز نشان داده شده است ختم می‌شود.

در قسمت بعدی ثابت خواهیم کرد که الگوریتم ارائه شده در جدول ۱۳،۱ به سمت تابع Q میل خواهد کرد. اما ابتدا دو خاصیت کلی الگوریتم یادگیری Q را که در مورد تمامی MDP های قطعی با پاداش‌های غیر منفی صادق است را بررسی می‌کنیم. فرض کنید که تمامی مقادیر اولیه‌ی $\hat{Q}$ را صفر قرار داده‌ایم. اولین خاصیت این است که با این شرایط مقادیر $\hat{Q}$ هیچ‌وقت در طول آموزش کاهش نمی‌یابد. به عبارت دیگر اگر $\hat{Q}_n(s,a)$ نماد مقدار یاد گرفته شده‌ی $\hat{Q}(s,a)$ بعد از n تکرار فرایند (مثلاً بعد از اینکه عامل n جفت حالت و عمل را بررسی کرد) باشد، خواهیم داشت:

$$(\forall s,a,n)\ \hat{Q}_{n+1}(s,a) \geq \hat{Q}_n(s,a)$$

خاصیت کلی دوم این است که در طول آموزش همیشه مقدار $\hat{Q}$ بین دو مقدار صفر و مقدار حقیقی Q خواهد بود.

$$(\forall s,a,n)\ 0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$$

۱۳،۳،٤ همگرایی

آیا تقریب $\hat{Q}$ که در الگوریتم جدول ۱۳،۱ آورده شده به مقدار حقیقی Q میل خواهد کرد؟ جواب مثبت است، اما با شرایطی. ابتدا باید فرض کنیم که سیستم یک سیستم قطعی MDP است. دوم اینکه باید فرض کنیم که پاداش‌های لحظه‌ای کران‌دارند؛ یعنی اینکه مقداری ثابت مثل c موجود است که برای تمامی حالت‌های s و تمامی اعمال a داشته باشیم $|r(s,a)|<c$. سوم اینکه باید فرض کنیم که عامل هر جفت حالت و عمل را حداقل هر چند وقت یک بار بررسی می‌کند. به عبارت دیگر اگر عمل a در حالت s یک قانونی بود، در طول زمان عامل باید تا زمانی که طول سری اعمال به بی‌نهایت برسد عمل a از حالت s با تناوبی غیر صفر انجام دهد. توجه دارید که این شرایط از جنبه‌ای محدود و از

جنبه‌ای بسیار کلی هستند. نسبت به مثال‌های بررسی شده این شرایط بسیار کلی هستند زیرا کـه بـا چنـین شـرایطی محیط‌هـای دلخـواهی را می‌توان در نظر گرفت که پاداش‌های صفر یا مثبت یا منفی و تعداد دلخواهی جفت حالت و عمل داشته باشند. اما از طرفی دیگر این محـدودیت وجود دارد که باید عامل بتواند در محیط هر جفت حالت و عمل را چند وقت یک بار بررسی کنـد. در کـل ایـن شـرط، در محیط‌هـای بـزرگ و پیوسته شرط بسیار قوی‌ای است. در آینده به نتایج قوی‌تر همگرایی خواهیم پرداخت. با این وجود شروط بررسی شده در ایـن قسـمت پایـه‌های درک درستی یادگیری Q است.

نکته‌ی کلیدی اثبات همگرایی این است که پر خطاترین مقدار جدول $\hat{Q}$ هر زمان که بررسی می‌شود باید به نسـبت ضـریب γ خطـای خـود را تصحیح کند. زیرا که قسمتی از مقدار جدید $\hat{Q}$ به مقدار خطادار $\hat{Q}$ و بقیه‌ی آن به مقدار بدون خطای پاداش r بستگی دارد.

قضیه‌ی ۱۳٫۱. همگرایی یادگیری Q برای فرایندهای تصمیم‌گیری مارکوف در حالت قطعـی. یـک عامـل یـادگیر را در یـک MDP قطعی با پاداش‌های کران‌دار در نظر بگیرید: $(\forall s,a)|r(s,a)| \leq c$. که این عامل از قـانون یـادگیری Q کـه در رابطـه‌ی ۱۳٫۷ استفاده می‌کند. پس عامل مقادیر $\hat{Q}(s,a)$ را با مقادیر دلخواه کران‌دار مقداردهی اولیه می‌کند و عامل تخفیف نیز γ است کـه $0 \leq \gamma < 1$. اگر $\hat{Q}_n(s,a)$ نماد فرضیه‌ی $\hat{Q}(s,a)$ پس از n بار تغییر باشد و هر جفت حالت و عمل نیز چند وقت یک بار بررسی شود، آنگاه زمـانی که n→∞ برای تمامی مقادیر a و s، $\hat{Q}_n(s,a)$ به Q(s, a) میل خواهد کرد.

اثبات. از آنجایی که هر زوج حالت و عمل چند وقت یک بار بررسی می‌شوند، بازه‌های پیاپی‌ای را در نظر بگیرید که در آن تمامی زوج حالت و عمل چند وقت یک بار بررسی می‌شوند. در اینجا ثابت می‌کنیم که در چنین بازه‌ای خطـای مـاکزیمم بـرای تمـامی مقـادیر جـدول $\hat{Q}$ حـداقل متناسب با γ کاهش می‌یابد. اگر $\hat{Q}_n(s,a)$ جدول مقادیر فرضیه بعد از n بار تغییر باشد و Δ_n نیز حداکثر خطای این جدول باشـد (بـه فـرم زیر):

$$\Delta_n \equiv \max_{s,a} |\hat{Q}(s,a) - Q(s,a)|$$

در زیر از نماد s′ برای نمایش δ(s,a) استفاده خواهیم کرد. حال برای هر مقدار جدول $\hat{Q}_n(s,a)$ که در حلقه‌ی n+1 ام عـوض مـی‌شـود، اندازه‌ی خطا در $\hat{Q}_{n+1}(s,a)$ خواهد بود:

$$\begin{aligned}
\left|\hat{Q}_{n+1}(s,a) - Q(s,a)\right| &= |(r + \gamma \max_{a'} \hat{Q}_n(s',a')) - (r + \gamma \max_{a'} Q(s,a)| \\
&= \gamma |\max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a')| \\
&\leq \gamma \max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\
&\leq \gamma \max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a')|
\end{aligned}$$

$$\left|\hat{Q}_{n+1}(s,a) - Q(s,a)\right| \leq \gamma \Delta_n$$

خط دوم به سوم در روابط بالا از رابطه‌ی کلی‌ای زیر برای دو تابع دلخواه f_1 و f_2 ناشی شده:

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

در خط سوم به چهارم نیز توجه کنید که متغیر جدیدی به نام s'' را که متغیر ماکزیمم کننده است اضافه کرده‌ایم. این کار از این رو درست است که مقدار ماکزیمم بعد از اضافه کردن متغیر جدید بزرگ‌تر یا مساوی خواهد شد. توجه داشته باشید که با اضافه کردن این متغیـر جدیـد تعریـف Δ_n را در نامعادله می‌سازیم.

بنابراین تغییر $\hat{Q}_{n+1}$ برای هر مقدار s و a حداقل به نسبت γ از ماکزیمم خطای جدول $\hat{Q}_n$، Δ_n، کوچک‌تر است. از طرفی بزرگ‌ترین خطای اولین جدول یا همان Δ_0 نیز کران‌دار است زیرا تمامی مقادیر $\hat{Q}_0$ و Q(s,a) به ازای تمامی مقادیر a و s کران‌دارند. حال بعد از اولین بـازه‌ای که تمامی s,a ها را بررسی می‌کند، بزرگ‌ترین خطای موجود در جدول $\gamma\Delta_0$ خواهد بود. بعد از k بار تکرار حداکثر خطا به $\gamma^k\Delta_0$ تقلیل خواهد یافت. از آنجایی که هر حالت هر چند وقت یک بار بررسی شده زمانی که $n\to\infty$ تعداد چنین تکرارهایی نیز به سمت بی‌نهایت میل خواهد کرد و متعاقباً خواهیم داشت که $\Delta_n \to 0$.

۱۳،۳،۵ استراتژی‌های آزمایش

توجه دارید که الگوریتم جدول ۱۳،۱ نحوه‌ی انتخاب اعمال توسط عامل را مشخص نکرده. یکی از استراتژی‌های ساده‌ی موجود انتخاب عملـی است که مقدار $\hat{Q}(s,a)$ را ماکزیمم کند و از اطلاعاتی که تا به حال به دست آورده برای به دست آوردن پاداش استفاده کند. با این وجـود بـا انتخاب چنین استراتژی‌ای عامل ریسک اعتماد[۲] را افزایش می‌دهد. یعنی اینکه در مراحل اولیه‌ی تخمین $\hat{Q}$ عملی مقـدار زیـادی در $\hat{Q}$ را پیـدا کرده در حالی که Q همان عمل ماکزیمم نیست. از طرف دیگر طبق قضیه‌ی بالا برای اینکه $\hat{Q}$ به Q همگرا شود باید تمامی اعمـال هـر چنـد وقت یک بار انجام شوند. واضح است که با انتخاب ماکزیمم در هر حالت چنین شرطی برآورده نخواهد شد. به همین دلیل، معمول است کـه در یادگیری Q برای انتخاب اعمال از روشی احتمالی استفاده می‌کنند. مسلماً به اعمالی که $\hat{Q}$ بیشتر دارند احتمال بیشـتری داده مـی‌شـود امـا بـه تمامی اعمال احتمالی غیر صفر باید داده شود. یکی از راه‌های انتخاب این احتمالات به شکل زیر است:

$$P(a_i|s) = \frac{k^{\hat{Q}(s,a)}}{\sum_j k^{\hat{Q}(s,a)}}$$

در این رابطه $P(a_i|s)$ احتمال انتخاب عمل a_i در حالت s است و k نیز ثابتی مثبت است که قدرت انتخاب گزینه‌ی بهتر را نشان می‌دهد. با افزایش مقدار k، اعمالی که $\hat{Q}$ بیشتر دارند احتمال بیشتری خواهند داشت، و باعث می‌شود تا عامل از دانسته‌هایش استفاده[۳] کنـد تـا اینکـه بـه جستجوی[۴] محیط بپردازد. در مقابل با کم شدن k احتمال اعمال دیگر افزایش می‌یابد و عامل بیشتر به سمت جستجو تمایل پیـدا مـی‌کنـد. در بعضی موارد بد نیست تا مقدار k را متناسب با تعداد تکرارهای الگوریتم عوض کنیم تا در مراحل اولیه‌یادگیری عامـل بـه جسـتجو بپـردازد و در مراحل آخر نیز از اطلاعات استفاده کند.

[۲] overcommit
[۳] expoit
[۴] explore

۱۳,۳,٦ سری تغییرها

یکی از اهمیت‌های قضیه‌ی همگرایی بالا این نکته است که یادگیری Q نیازی به دنبال کردن خط‌مشی بهینه برای یادگیری آن ندارد. در واقع می‌توان تابع Q (و متعاقباً خط‌مشی بهینه) را با دنبال کردن یک خط‌مشی کاملاً تصادفی در هر مرحله و بررسی تمامی جفت حالت و عمل‌ها هر چند وقت پیدا کرد. همین نکته به ما اجازه می‌دهد که بدون از بین بردن همگرایی الگوریتم بتوانیم نحوه‌ی انتخاب اعمال را برای بهینه‌سازی یادگیری تغییر دهیم. برای تصور، دوباره یادگیری MDP را با همان حالت جاذب شکل ۱۳,۱ را در نظر بگیرید. فرض کنید که هنوز عامل را با قسمت‌های یادگیری آموزش نداده‌ایم. برای هر قسمت عامل را در حالتی تصادفی قرار می‌دهیم تا اعمالی را انجام دهد و جدول $\hat{Q}$ را تغییر دهد تا به حالت جاذب برسد. و بعد از آن نیز دوباره عامل در حالتی تصادفی قرار می‌گیرد. همان‌طور که قبلاً نیز گفته شد اگر تمامی مقادیر جدول $\hat{Q}$ را صفر قرار دهیم بعد از مرحله‌ی اول تنها یک مقدار جدول تغییر کرده و آن نیز مقدار حالت قبلی حالت جاذب است. توجه داشته باشید که اگر عامل همان سری اعمال را طی کند یکی دیگر از مقادیر جدول $\hat{Q}$ نیز غیر صفر خواهد شد. حال اگر در تمامی قسمت‌ها همین اعمال را تکرار کنیم مقادیر حالت‌ها به ترتیب از حالت جاذب به سمت حالت اولیه تک‌تک با سرعت یک حالت در قسمت غیر صفر خواهند شد. حال همان آموزش را با ترتیب برعکس برای هر قسمت در نظر بگیرید. یعنی اینکه از همان رابطه‌ی ۱۳,۷ برای محاسبه‌ی $\hat{Q}$ استفاده می‌کنیم اما این ترتیب را برعکس انجام می‌دهیم. با چنین شرایطی در یک قسمت می‌توان تمامی مقادیر تخمینی $\hat{Q}$ را در طول این مسیر دلخواه به سمت حالت جاذب پیدا کرد. چنین فرایندی مسلماً در تکرارهای کمتری همگرا خواهد شد اما با این وجود حافظه‌ی بیشتری برای ذخیره‌ی کل قسمت قبل از آموزش آن قسمت لازم است.

استراتژی دوم مطرح برای بهبود سرعت همگرایی ذخیره‌سازی جفت حالت و عمل‌های قبلی با پاداش‌های لحظه‌هایشان تکرار تناوبی عملیات مربوطه است. با وجود اینکه در نگاه اول چنین کاری هدر دادن زمان به نظر می‌رسد اما این کار باعث تصحیح $\hat{Q}(s,a)$ می‌شود زیرا که در اجرای اول این مقدار از $\hat{Q}(s',a)$ نشأت گرفته (s'=δ(s,a)). در حالی که خود مقدار $\hat{Q}(s',a)$ بعد از آن دچار تغییرات می‌شود، پس تکرار دوباره‌ی جفت <s,a> باعث بهبود مقدار $\hat{Q}(s,a)$ خواهد شد. در کل، میزانی که به مسیرهای قدیمی می‌پردازیم در مقابل میزانی که مسیرهای جدید ایجاد می‌کنیم بستگی به هزینه‌ی نسبی این دو عملیات در قلمرو[۵] مسئله دارد. مثلاً در قلمرو رباتی که حرکت‌هایش بسیار کند است، تأخیر ایجاد شده در پیدا کردن جفت حالت و عمل‌های جدید در جهان واقعی هزینه‌ی چندین برابر تکرار همان مسیرهای قبلی دارد. این تفاوت باعث می‌شود که گاهی یادگیری Q بعد از چندین هزار تکرار حلقه همگرا شود.

توجه داشته باشید که در تمام طول بحث بالا فرضمان بر این بود که توابع δ(s,a) و r(s,a) برای عامل مجهول است. اگر این دو تابع برای عامل معلوم باشند متدهای مؤثرتری را می‌توان بکار برد. برای مثال اگر اجرای اعمال خارجی هزینه‌بر است عامل می‌تواند از اثر خارجی آن چشم‌پوشی کرده و آن را شبیه‌سازی کند و در این محیط شبیه‌سازی شده با انجام اعمال و پاداشی که خود با استفاده از r در نظر می‌گیرد خود را آموزش دهد. (Sutton 1991) ساختار Dyna را معرفی می‌کند که بعد از انجام هر مرحله از آموزش در جهان واقعی تعدادی عمل را در جهان شبیه‌سازی شده انجام می‌دهد. (Moore and Atkeson 1993) متدی به نام prioritized sweeping را معرفی می‌کنند که زمانی که در ادامه فرایند به حالت‌های امیدوارکننده جدید می‌پردازد و فقط زمانی که تغییر بزرگی رخ داد به حالت‌های قبلی باز می‌گردد. (Peng and Williams 1994) نیز روشی مشابه را توصیف می‌کنند. زمانی که توابع δ و r معلوم‌اند، تعداد زیادی از الگوریتم‌های کارآمد را می‌توان برای مبحث برنامه‌نویسی پویا به کاربرد. (Kaelbling 1996) تعدادی از این الگوریتم‌ها را بررسی می‌کند.

---

[۵] Domain

۱۳,٤ اعمال و پاداش‌های غیرقطعی

در قسمت‌های قبلی یادگیری Q را برای محیط‌های قطعی بررسی کردیم. حال می‌خواهیم فرض کنیم که محیط غیرقطعی است، یعنی تابع r(s,a) و تابع δ(s,a) ممکن است خروجی‌های احتمالی داشته باشند. برای مثال در برنامه‌ی بازی تخته‌نردی که Tesauro (1995) طراحی کرده، خروجی اعمال ذاتاً احتمالی است زیرا که هر حرکت به پرتاب تاس وابسته است. همچنین در مسئله‌ی رباتی که حسگرها و اعمال نویز دار هستند لازم است که رابطه‌ی بین اعمال و پاداش‌ها را غیرقطعی فرض کنیم. در چنین شرایطی، دو تابع r(s,a) و δ(s,a) را می‌توان به صورت توزیع‌های احتمالی روی دو فضای s و a در نظر گرفت و سپس خروجی تصادفی‌ای با توجه به این توزیع‌ها به دست آورد. زمانی که این توزیع‌های احتمال منحصراً به s و a وابسته باشند (مثلاً به حالت قبلی یا عمل قبلی وابسته نباشند)، آنگاه به کل سیستم، سیستم تصمیم‌گیری غیرقطعی مارکوف[۶] می‌گوییم.

در این قسمت الگوریتم یادگیری Q را برای حالت غیرقطعی در محیط‌های غیرقطعی MDP تامیم می‌دهیم. برای این کار، دوباره الگوریتم را برای حالت قطعی بررسی کرده و در موارد لازم آن را عوض می‌کنیم.

در حالت غیرقطعی، در ابتدا باید برای عامل در نظر بگیریم که دیگر خروجی‌ها قطعی نیستند. پس بدیهی است که رابطه‌ی V^π را برای خطمشی π به صورت امید پاداش تجمعی تخفیفی بیان کنیم (زیرا که دیگر هیچ‌چیز قطعی نیست). پس داریم:

$$V^\pi(s_t) \equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

که در این رابطه، همان‌طور که قبلاً نیز گفته شد، سری r_{t+i} پاداش‌های دریافتی خطمشی π با شروع از حالت s است. توجه داشته باشید که این همان تامیم رابطه‌ی ۱۳,۱ برای حالت غیرقطعی است.

همان‌طور که قبلاً نیز گفته شد، خطمشی بهینه π^* خطمشی‌ای مثل π است که مقدار $V^\pi(s)$ را برای تمامی حالت‌های s ماکزیمم می‌کند. در گام بعدی تعریف Q را که در رابطه‌ی ۱۳,۴ آمده تامیم می‌دهیم:

$$\begin{aligned} Q(s,a) &\equiv E[r(s,a) + \gamma V^*(\delta(s,a))] \\ &= E[r(s,a)] + \gamma E[V^*(\delta(s,a))] \\ &= E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) V^*(s') \end{aligned} \tag{13.8}$$

که در آن $P(s'|s,a)$ احتمال رسیدن به حالت s' پس از انجام عمل a در حالت s است. توجه داشته باشید که ما در اینجا از $P(s'|s,a)$ برای بازنویسی امید مقدار $V^*(\delta(s,a))$ استفاده کرده ایم.

درست مثل روابط قبلی برای تعریف Q نیز خواهیم داشت:

---

[۶] nondeterministic Markov decision process

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \qquad (13.9)$$

که این رابطه نیز تامیم یافته‌ی رابطه‌ی ۱۳٫۶ است. به طور خلاصه، تعریف جدید Q(s,a) امید مقدار تعریف شده‌ی حالت قطعی است.

حال که تعریف Q را بازنویسی کردیم، باید به سراغ بازنویسی قانون آموزش برویم. قانون آموزش قبلی‌ای که از بازنویسی قانون قبلی (رابطه‌ی 13.7) به دست می‌آید برای همگرایی در شرایط غیرقطعی دچار مشکل می‌شود. فرض کنید، برای مثال، که r(s,a) تابعی غیرقطعی است که هر بار پاداش‌های متفاوتی برای زوج <s,a> می‌دهد. در چنین شرایطی، رابطه‌ی قبلی متناوباً مقدار $\hat{Q}(s,a)$ را حتی اگر مقدار $\hat{Q}(s,a)$ مقدار درست تابع Q باشد تغییر خواهد داد. به طور خلاصه، قانون آموزش قبلی در چنین شرایطی همگرا نمی‌شود. برای حل چنین مشکلی باید در قانون قدیمی تغییراتی را انجام داد، قانون جدید در هر مرحله باید به جای از بین بردن تخمین قبلی میانگینی وزن‌دار بین مقدار تخمینی جدید و مقادیر قبلی محاسبه کند. با تغییر رابطه به فرم زیر شرایط لازم برای همگرایی $\hat{Q}$ به Q فراهم می‌شود:

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')] \qquad (13.10)$$

که

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \qquad (13.11)$$

در رابطه‌ی بالا s و a حالت و عملی هستند که در تکرار n ام حلقه رخ می‌دهند و $visits_n(s,a)$ تعداد تمامی جفت حالت و عمل‌هایی است که تا تکرار n ام بررسی شده‌اند.

نکته‌ی کلیدی در این قانون این است که $\hat{Q}$ در این رابطه کندتر از قانون قبلی تغییر می‌کند. توجه داشته باشید که اگر مقدار α_n در رابطه‌ی ۱۳٫۱۰ یک قرار داده شود به همان رابطه‌ی آموزش قدیمی می‌رسیم. با کمتر کردن مقدار α، این جمله میانگین $\hat{Q}$(s,a) و جدیدتر خواهد بود. توجه دارید که با افزایش n در رابطه‌ی ۱۳٫۱۱ α کاهش می‌یابد، پس تغییرات با پیشرفت آموزش به تدریج کمتر خواهد شد. با کم کردن α با سرعت مناسب در طول آموزش می‌توان به تابع Q میل کرد. انتخاب α_n ی که در بالا آمده با توجه به قضیه‌ی زیر فقط یکی از چندین شرط همگرایی است (Watkins and Dayan 1992).

قضیه‌ی ۱۳٫۲. همگرایی یادگیری Q برای فرایند تصمیم‌گیری غیرقطعی مارکوف. فرض کنید که عامل یادگیری Q در یک محیط غیرقطعی MDP قرار گرفته است که پاداش‌ها نیز کران‌دارند $(\forall s,a)|r(s,a)| \le c$. عامل یادگیری Q با رابطه‌ی 13.10 و مقدار اولیه‌ی دلخواه کران‌دار اولیه‌ی جدول $\hat{Q}(s,a)$ و عامل تخفیف $0 \le \gamma < 1$ سعی می‌کند تا Q را تخمین بزند. اگر n(I,s,a) شماره‌ی تکراری باشد که عمل a از حالت s برای i امین بار اجرا می‌شود و اگر تمامی جفت حالت و عمل‌ها هر چند وقت یک بار بررسی شوند و نیز $0 \le \alpha_n < 1$ و داشته باشیم

$$\sum_{i=1}^{\infty} \alpha_{n(i,s,t)} = \infty, \sum_{i=1}^{\infty} [\alpha_{n(i,s,t)}]^2 < \infty$$

برای تمامی s و a ها اگر n→∞ با احتمال ۱ خواهیم داشت که $\hat{Q}(s,a) \to Q(s,a)$.

با وجود اینکه ثابت می‌شود که یادگیری Q و دیگر الگوریتم‌های یادگیری تقویتی تحت شرایطی همگرا می‌شوند، اما در عمل چند هزار تکرار حلقه‌ی اصلی لازم است تا به میزان مطلوب همگرا شوند. برای مثال در بازی TD-Gammon که توسط (Tesauro) مطرح شده، که قبلاً به آن اشاره کردیم، ۱,۵ میلیون بازی مختلف وجود دارند که هر کدام نیز ده‌ها جفت حالت و عمل دارند.

۱۳,۵ یادگیری اختلاف ارزش‌ها

یادگیری Q با کم کردن تفاوت مقدار تخمینی و مقدار اصلی Q در هر تکرار حلقه، Q را یاد می‌گیرد. از این نظر یادگیری Q یک حالت خاص از دسته الگوریتم‌های کاهش اختلاف ارزش‌ها[۷] است که از طریق کم کردن تفاوت بین تخمین عامل و تابع اصلی در چندین مرحله تابع هدف را تخمین می‌زنند. همان‌طور که قانون آموزش ۱۳,۱۰ تفاوت بین مقدار تخمینی $\hat{Q}$ برای حالت ابتدایی و حالت پایانی یک عمل را کم و زیاد می‌کند می‌توانیم الگوریتم‌هایی را طراحی کنیم تا اختلاف بین حالت‌های چند عمل قبل و حالت‌های چند عمل بعد را نیز کم و یا زیاد کند.

برای بررسی بیشتر، محاسبات رابطه‌ی یادگیری Q را که برای محاسبه‌ی مقدار $\hat{Q}(s_t, a_t)$ با استفاده از مقدار $\hat{Q}(s_{t+1}, a_{t+1})$ آورده شده را در نظر بگیرید (که در آن s_{t+1} نتیجه‌ی انجام عمل a_t در حالت s_t است). این رابطه را با نگاه یک مرحله‌ای را با $\hat{Q}^{(1)}(s_t, a_t)$ نیز نشان می دهند:

$$\hat{Q}^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a_t)$$

می‌توان بجای این نگاه یک مرحله‌ای به پاداش‌ها نگاه دو مرحله‌ای داشت:

$$\hat{Q}^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a_t)$$

یا در حالت کلی نگاهی n مرحله‌ای به پاداش‌ها داشت:

$$\hat{Q}^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a_t)$$

(Sutton 1988) متدی کلی برای ترکیب این روابط تخمینی جایگزین به نام TD(λ) معرفی می‌کند. ایده‌ی اصلی TD(λ)، استفاده از ثابت $0 \le \lambda \le 1$ برای ترکیب این جایگزین‌هاست:

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda)\left[\hat{Q}^{(1)}(s_t, a_t) + \lambda \hat{Q}^{(2)}(s_t, a_t) + \lambda^2 \hat{Q}^{(3)}(s_t, a_t) + \cdots\right]$$

به صورت بازگشتی خواهیم داشت که :

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma[(1-\lambda) \max_a \hat{Q}(s_t, a_t) + \lambda Q^{\lambda}(s_{t+1}, a_{t+1})]$$

حال اگر λ=0 بگیریم به همان رابطه‌ی $\hat{Q}^{(1)}$ خواهیم رسید که فقط یک مرحله پاداش را در تخمین نظر می‌گیرد. با افزایش مقدار λ الگوریتم به سمتی می‌رود تا اختلاف تخمین را نیز در مراحل بعدی کم کند. در نهایت امر زمانی که λ=1 است فقط مقادیر r_{t+i} در نظر گرفته می‌شوند

[۷] temporal difference algorithms

و تخمین فعلی نیز تأثیری در رابطه نخواهد داشت. توجه داشته باشید که اگر $\hat{Q} = Q$ باشد قانون ذکر شده برای Q^{λ} برای تمامی مقادیر $0 \leq \lambda \leq 1$ ایده آل خواهد بود.

انگیزه‌ی ایجاد TD(λ) این است که در بعضی شرایط آموزش اگر با عمق بیشتر نگاه کنیم آموزش مؤثرتر خواهد شد. برای مثال، زمانی که عامل از خطمشی بهینه پی روی می‌کند، با مقدار λ=1 تخمین Q^{λ}بدون توجه به اشتباهات موجود در $\hat{Q}$ عالی‌ای از مقادیر واقعی تابع Q به ما می‌دهد. از طرف دیگر اگر اعمال به صورت ناحیه‌ای بهینه انتخاب شوند، r_{t+i} های به دست آمده ممکن است با توجه به آینده گمراه‌کننده باشند.

(Peng and Williams 1994) بحثی گسترده‌تر را به همراه نتایج آزمایشی که کارایی Q^{λ} را در قلمروی مسئله‌ی خاصی نشان می‌دهد ارائه می‌کنند. (Dayan 1992) نیز نشان می‌دهد که تحت شرایط خاصی روشی مشابه TD(λ) برای تمامی مقادیر $0 \leq \lambda \leq 1$ به تابع V^* میل می‌کند. (Tesauro 1995) از روش TD(λ) در برنامه‌ی TD-Gammon برای بازی تخته‌نرد[۸] استفاده می‌کند.

۱۳,۶ تعمیم روی نمونه‌ها

شاید یکی از محدودکننده‌ترین فرض‌هایی که تا به حال در مورد یادگیری Q کردیم این بود که Q به صورت تابعی جدولی در نظر گرفته می‌شد که برای هر ورودی خاص (مثلاً یک زوج حالت و عمل) یک خروجی خاص داشت. پس الگوریتم‌هایی که تا به حال به کار بردیم چیزی شبیه یک حافظه معمولی[۹] انجام می‌دهند و هیچ تلاشی برای تخمین مقادیر دیگر Q نمی‌کنند. این فرض در اثبات همگرایی نیز خود را نشان داده است، تنها زمانی الگوریتم همگرا خواهد شد که هر جفت حالت و عمل بررسی شوند (آن هم هر چند وقت یک بار!). چنین فرضی در فضاهای بزرگ و نامحدود فرضی کاملاً غیرواقعی است، یا حداقل، اجرای آن هزینه و وقت زیادی می‌برد. همین باعث شده تا در اکثر سیستم‌های کاربردی ترکیبی از یادگیری Q و تخمین توابع در فصل‌های گذشته شد مورد استفاده قرار می‌گیرد.

ترکیب الگوریتم‌های تخمین توابع مثل Backpropagation و یادگیری Q بسیار ساده است، به راحتی می‌توان از مقادیر جدول $\hat{Q}$ را برای آموزش شبکه‌ای عصبی استفاده کرد. برای مثال، می‌توان با کد کردن حالت s و عمل a و ورود آن‌ها به شبکه و گرفتن مقدار $\hat{Q}$ از خروجی آن و آموزش شبکه با استفاده از نمونه‌های جدول $\hat{Q}$ و روابط ۱۳,۷ و ۱۳,۱۰ شبکه‌ای برای تخمین $\hat{Q}$ ساخت. روش دیگر که در کاربرد موفق‌تر بوده، آموزش شبکه‌های مجزایی برای هر عمل با ورودی حالت و خروجی $\hat{Q}$ است. یکی دیگر از راه‌های مرسوم آموزش شبکه‌ای است که حالت را به عنوان ورودی می‌گیرد و برای هر عمل نیز یک مقدار $\hat{Q}$ خروجی می‌دهد. توجه دارید که در فصل اول در صفحه‌ی بازی checkers تابع ارزیابی را با استفاده از LMS با تابعی خطی تخمین زدیم.

در عمل، سیستم‌های یادگیری تقویتی موفقی را می‌توان با جایگزینی الگوریتم‌های تخمین به جای جدول ایجاد کرد. در برنامه‌ی TD-Gammon که ساخته‌ی Tesauro است برای بازی تخت نرد از شبکه‌ای عصبی و الگوریتم Backpropagation و قانون TD(λ) استفاده شده است. (Zhang and Dietterich 1996) از ترکیبی مشابه از Backpropagation و TD(λ) برای برنامه‌ریزی

---

[۸] backgammon

[۹] Rote learning

مغازه‌ای[۱۰] استفاده کرده‌اند. (Crites and Barto 1996) روشی تقویتی با شبکه‌ی عصبی برای برنامه‌ریزی یک آسانسور را مورد استفاده قرار دادند. (Thrun 1996) نیز از یادگیری Q بر اساس خوشه یابی دسته حالت‌ها برای مسئله کنترل ساده‌ی ربات استفاده کرده است.

با وجود موفقیت در این سیستم‌ها، یادگیری تقویتی در کارهایی دیگر با شکست روبرو شده است، تابع همگرا نشده است. مثال‌هایی از این شکست‌ها در (Boyan and Moore 1995) و (Baird 1995) و (Gordon 1995) آمده است. توجه دارید که قضایای همگرایی مطرح شده در این فصل فقط برای زمانی برقرار است که $\hat{Q}$ با جدولی از داده‌ها نمایش داده شود. برای درک بهتر، فرض کنید به جای جدول از شبکه‌ای عصبی بجای جدول $\hat{Q}$ استفاده شود. توجه دارید که اگر یادگیر شبکه را برای یادگیری بهتر Q برای نمونه‌ی خاصی چون < s_i, a_i > تغییر دهد، ممکن است تخمین‌های $\hat{Q}$ دیگر زوج مرتب‌ها را نیز دستخوش تغییر کند. زیرا که ممکن است این تغییرات خطای $\hat{Q}$ را در تخمین دیگر زوج مرتب‌ها افزایش دهد، و ویژگی لازم بنا قضیه‌ی اثبات همگرایی دیگر تضمین شده نیست. بررسی‌های تئوری یادگیری تقویتی با تعمیم توابع تخمینی در (Gordon 1995) و (Tsitsiklis 1994) آورده شده است. (Baird 1995) متدهای شیب نزول را که خطای کل نمونه‌های آموزشی را در نظر می‌گیرند را برای رفع این مشکلات (عدم همگرایی) پیشنهاد می‌کند (این روش خطای باقیمانده‌ی بلمن[۱۱] نیز نامیده می‌شود).

۱۳٫۷ رابطه با برنامه‌نویسی پویا

متدهای یادگیری تقویتی همچون یادگیری Q رابطه‌ای بسیار نزدیک با تحقیقاتی در برنامه‌نویسی پویا برای حل مسائل تصمیم‌گیری مارکوف دارند. در چنین مسائلی فرض می‌کنند که عامل اطلاعات کاملی درباره‌ی توابع δ(s,a) و r(s,a) که محیط را تعریف می‌کنند دارد. بنابراین، با فرض اینکه محیط کاملاً قابل شبیه‌سازی است و نیازی به تعامل مستقیم با محیط نیست اصولاً این سؤال مطرح می‌شود چگونه با کمترین تلاش محاسباتی به خطمشی بهینه برسیم؟ ویژگی طولانی کننده‌ی یادگیری Q این بود که فرض می‌کرد عامل هیچ علمی درباره‌ی توابع δ(s,a) و r(s,a) ندارد و باید به جای حرکت در شبیه‌سازی باید در محیط واقعی به حرکت و مشاهده عکس‌العمل محیط بپردازد. در شرایطی که ذکر شد اولویت اول ما معمولاً کم کردن تعداد حرکاتی است که عامل در جهان واقعی انجام می‌دهد تا به خطمشی قابل قبولی همگرا شود، کم کردن تعداد تکرار حلقه‌ی محاسبات در اولویت‌های بعدی است. دلیل چنین اولویت‌بندی‌ای این است که در بسیاری از زمینه‌های کاربردی مثل مسائل تولید، هزینه‌ی اعمال در جهان واقعی پول و زمان است که از هزینه‌ی مصرفی برای انجام محاسبات بسیار بیشتر است. سیستم‌هایی که در محیط واقعی عمل می‌کنند و نتایج اعمال را مشاهده می‌کنند اصطلاحاً سیستم‌های online نامیده می‌شوند در حالی که سیستم‌هایی که در محیط‌های شبیه‌سازی عمل می‌کنند سیستم‌های offline نامیده می‌شوند.

ارتباط بین روش‌های قبلی و یادگیری تقویتی که در اینجا ذکر شد توسط رابطه‌ی بلمن (Bellman) آشکار می‌گردد، این رابطه پایه‌ی بسیاری از روش‌های برنامه‌نویسی پویا برای حل MDP هاست. رابطه‌ی بلمن در زیر نوشته شده:

$$(\forall s \in S) V^*(s) = E[r\big(s,\pi(s)\big) + \gamma V^*(\delta\big(s,\pi(s)\big))]$$

[۱۰] job-shop schedualing

[۱۱] Bellman residual errors

به رابطه‌ی نزدیک بین رابطه‌ی بلمن و رابطه‌ای که قبلاً برای خط‌مشی بهینه تعریف کردیم (رابطه‌ی ۱۳٫۲) توجه داشته باشید. بلمن (Bellman 1957) نشان داد که خط‌مشی بهینه‌ی π^* در رابطه‌ی بالا صدق می‌کند و هر رابطه‌ای که در رابطه‌ی بالا صدق کند نیز خط‌مشی ی بهینه است. از جمله اولین بررسی‌ها درباره‌ی برنامه‌نویسی پویا می‌توان الگوریتم کوتاه‌ترین مسیر بلمن–فورد (Bellman 1958; Ford and Fulkerson 1962) را نام برد که یاد می‌گرفت تا کوتاه‌ترین مسیر به سمت هدف در یک گراف را برای هر گره گراف بر اساس طول مسیرهای گره‌های همسایه‌اش تخمین می‌زد. در این الگوریتم این فرض که یال‌های گراف و گره هدف معلوم‌اند معادل فرض ما در معلوم بودن دو تابع δ(s,a) و r(s,a) است. (Barto 1995) و مقالات دیگر نیز درباره‌ی رابطه‌ی نزدیک برنامه‌نویسی پویا و یادگیری تقویتی بحث کرده‌اند.

۱۳٫۸ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی مطرح شده در این فصل به شرح زیر است:

- یادگیری تقویتی مسئله‌ی یادگیری استراتژی کنترل برای عامل‌های خودکار[۱۲] را حل می‌کند. این نوع یادگیری فرض می‌کند که داده‌های آموزشی به فرم سیگنال پاداش حقیقی مقداری به ازای هر جفت حالت و عمل به یادگیر داده می‌شود. هدف عامل یادگیری خط‌مشی‌ای که کل پاداش دریافتی را مستقل از نقطه‌ی شروع حداکثر کند.
- الگوریتم‌های یادگیری تقویتی‌ای که در این فصل آورده شده‌اند، برای تعریف مسئله‌ی معروفی به نام فرایند تصمیم‌گیری مارکوف ارائه شده است. در فرایند تصمیم‌گیری مارکوف، نتیجه‌ی حاصل از اعمال یک عمل به یک حالت فقط به حالت و عمل انجام شده وابسته است (و به اعمال قبلی و حالات قبلی وابستگی ندارد). تعریف مسئله‌ی فرایند تصمیم‌گیری مارکوف مسائل زیادی را از جمله بسیاری از مسائل کنترل ربات، اتوماسیون کارخانه و مسائل برنامه‌ریزی را در بر می‌گیرد.
- یادگیری Q یکی از فرم‌های یادگیری تقویتی است که عامل در آن تابعی بر روی حالات و اعمال را یاد می‌گیرد. در کل، تابع ارزیابی Q(s,a)، امید ماکزیمم پاداش تخفیفی تجمعی قابل دریافت برای عامل با اعمال عمل a به حالت s تعریف می‌شود. الگوریتم یادگیری Q این مزیت را دارد که حتی زمانی که عامل دانش قبلی‌ای در مورد تأثیر عملش بر محیط ندارد هم قابل‌استفاده است.
- ثابت می‌شود که یادگیری Q به شرط آنکه فرضیه یادگیر از $\hat{Q}(s,a)$ جدولی از داده‌های خام <s,a> باشد به تابع Q درست میل می‌کند. این قضیه هم در حالت قطعی و هم در حالت احتمالی MDP درست است. در عمل یادگیری Q حتی در مسائلی با معتدل‌ترین اندازه ممکن است پس از هزاران حلقه همگرا شود.
- یادگیری Q عضوی از کلاس گسترده‌تری از الگوریتم‌ها به نام الگوریتم‌های کاهش اختلاف ارزش‌ها محسوب می‌گردد. در کل، الگوریتم‌های کاهش اختلاف ارزش‌ها با کاهش تناوبی اختلافات بین تخمین عامل و واقعیت یاد می‌گیرند.
- یادگیری تقویتی بسیار نزدیک به برنامه‌نویسی پویا در حل فرایند تصمیم‌گیری مارکوف است. تفاوت کلیدی در این است که برنامه‌نویسی پویا فرض می‌کند که اطلاعات لازم از $\delta(s,a)$ و $r(s,a)$ را دارد. در مقابل یادگیری تقویتی در کل فرض می‌کند که یادگیر از داشتن چنین موهبتی محروم است.

---

[۱۲] autonomous

موضوع متداولی که زیرساخت اکثر کارهای یادگیری تقویتی است، کم کردن اختلاف بین ارزیابی‌های حالت‌های موفق در هر حلقه است. بعضی از تلاش‌های اولیه روی چنین متدی توسط (Samuel 1959) انجام گرفت. برنامه‌ی یادگیری بازی چکرز وی سعی دارد تا تابع ارزیابی‌ای با استفاده از ارزیابی حالت‌های انتهایی برای ایجاد مقادیر آموزشی برای حالت‌های اولیه ایجاد کند. تقریباً در همان موقع، الگوریتم‌های Bellman-Ford، برای کوتاه‌ترین مسیر با هدف معلوم توسط (Bellman 1958; Ford and Fulkerson 1962) ایجاد گشتند، این الگوریتم‌ها مقادیر متغیرهای فاصله تا هدف را از گره‌ها به همسایه‌هایشان سرایت می‌دهند. تحقیق روی کنترل بهینه برای حل فرایندهای مارکوف به متدهای مشابه ای رسید (Bellman 1961; Blackwell 1965). متد bucket brigade (Holland 1986) نیز برای یادگیری سیستم‌های دسته‌بندی از متدی مشابه که امتیاز را در بین پاداش‌های تأخیری پخش می‌کند استفاده می‌کند. (Barto et al. 1983) روشی برای اختصاص امتیاز زمانی ارائه کرده که در نهایت به مقاله (Sutton 1988) ختم شد که TD(λ) را معرفی کرد و همگرایی آن را برای λ=0 اثبات کرد. (Dayan 1992) نیز این نتیجه را به مقادیر دلخواه λ تعمیم داد. (Watkins 1989) یادگیری Q را برای رسیدن به خط‌مشی بهینه زمانی که توابع پاداش و انتقال نامعلوم‌اند ارائه داد. اثبات همگرایی برای بسیاری از حالات مختلف این متدها ارائه گردیده است. علاوه بر اثباتی که در این فصل ارائه گردید، می‌توانید به (Baird 1995; Bersekas 1987; Tsitsiklis 1994; Singh and Sutton 1996) مراجعه کنید.

یادگیری تقویتی همچنان زمینه‌ی تحقیق فعالی است. برای مثال، (McCallum 1995) و (Littman 1996) تعمیم یادگیری تقویتی را به تعریف مسئله‌ای با متغیرهای حالت غیرقابل مشاهده را بحث کردند، این تعریف مسئله خارج از تعریف مسئله‌ی فرایند تصمیم مارکوف است. تحقیق زیادی برای تعمیم این متدها روی مسائل بزرگ‌تر و کاربردی‌تر انجام می‌شود. (Maclin and Shavlik 1996) روشی ارائه کردند که یادگیر تقویتی بتواند از مربی توصیه‌های ناکامل دریافت کند، این روش بر اساس تعمیمی از KBANN است (فصل ۱۲). (Lin 1992) نیز نقش آموزش را با سری اعمال پیشنهادی بررسی می‌کند. متدهایی نیز برای تعمیم و اعمال سلسله مراتبی اعمال توسط (Singh 1993) و (Lin 1993) ارائه شده است. (Dietterich and Flann 1995) انتگرال‌گیری متدهای مبتنی بر توضیحات را با یادگیری تقویتی بحث کرده و (Mitchell and Thrun 1993) کارایی الگوریتم EBNN (فصل ۱۲) را بر روی یادگیری Q بحث می‌کنند. (Ring 1994) یادگیری پیوسته یادگیر را روی کارهای چندگانه بررسی کرده است.

تحقیقات جدید روی یادگیری تقویتی در (Kaelbling et al. 1996)؛ (Barto 1992)؛(Barto et al. 1995) و (Dean et al. 1993) آمده است.

تمرینات

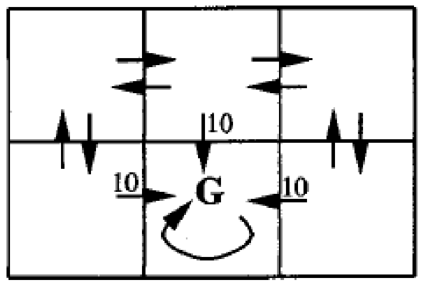
۱۳٫۱ خط‌مشی بهینه‌ی دیگری برای مسئله‌ی نشان داده شده در شکل ۱۳٫۲ بیابید.

۱۳٫۲ جهان قطعی نشان داده شده زیر را با حالت جاذب هدف G در نظر بگیرید. در این جا تمامی فلش‌هایی که عدد ۱۰ دارند پاداش ۱۰ و بقیه اعمال پاداش ۰ دارند.

(a) V^* را برای تمامی حالات شکل مشخص کنید. برای تمامی حرکت‌های ممکن Q(s,a) را محاسبه کنید. در انتها نیز خط‌مشی بهینه‌ای را ارائه کنید. فرض کنید که γ=0.8.

(b) تغییری در r(s,a) پیشنهاد کنید که مقادیر Q(s,a) را تغییر دهد اما در خطمشی بهینه تأثیری نداشته باشد. تغییری در r(s,a) پیشنهاد کنید که Q(s,a) را تغییر داده اما تأثیری بر $V^*(s,a)$ نداشته باشد.

(c) حال می‌خواهیم یادگیری Q را به این جهان با فرض اینکه مقادیر اولیه‌ی $\hat{Q}$ صفرند اعمال کنیم. فرض کنید که عامل از گوشه پایین سمت چپ شروع و ساعت‌گرد به حرکت خود ادامه می‌دهد تا به حالت جاذب برسد. چه مقادیری از $\hat{Q}$ تغییر خواهد کرد و این تغییرات چه هستند. حال همین سؤال را برای تکرار همین دور برای بار دوم و سوم جواب دهید.



۱۳٫۳ بازی دوز[۱۳] را در مقابل بازیکنی که به صورت تصادفی بازی می‌کند را در نظر بگیرید. در عمل فرض کنید که حریف با توزیع یکنواخت یکی از خانه‌های خالی را انتخاب می‌کند، مگر اینکه مجبور به انتخاب خانه‌ی دیگر باشد (که به وضوح خانه درست را انتخاب خواهد کرد).

(a) مسئله یادگیری استراتژی بهینه‌ی بازی دوز را در این شرایط برای یادگیری Q را دقیق بیان کنید. در این فرایند تصمیم‌گیری غیرقطعی مارکوف مجموعه‌های حالات اعمال و پاداش چه هستند؟

(b) آیا برنامه‌ی شما در صورتی که حریف بهینه بازی کند نیز موفق خواهد بود؟

۱۳٫۴ توجه داشته باشید که در بسیاری از مسائل MDP به سادگی می‌توان دو خطمشی مختلف مثل π_1 و π_2 را پیدا کرد که اگر عامل از حالت s_1 شروع به حرکت کند خطمشی π_1 بهینه‌تر باشد و اگر از حالت s_2 شروع کند π_2 بهینه‌تر باشد. به عبارت دیگر، $V^{\pi_1}(s_1) >$ $V^{\pi_2}(s_1)$ اما $V^{\pi_2}(s_2) > V^{\pi_1}(s_2)$. توضیح دهید که چرا همیشه خطمشی‌ای (مثل π^*) وجود دارد که برای تمامی حالات اولیه‌ی s $V^{\pi}(s)$ ماکزیمم است. به عبارت دیگر، توضیح دهید که چرا یک MDP همیشه اجازه می‌دهد که خطمشی‌ای مثل π^* باشد که $(\forall \pi, s) V^{\pi^*}(s) \geq V^{\pi}(s)$.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

استفاده از دانسته‌ها برای به دست آوردن پاداش	Expoit
اعمال	Action
بازی‌های صفحه‌ای	board games
برنامه‌نویسی پویا	dynamic programming

[۱۳] tic-tac-toe

پاداش تأخیری	delayed reward
پاداش تجمعی تخفیفی	discounted cumulative reward
پاداش لحظه‌ای	imidiate reward
تابع پاداش	reward function
تجمعی	cumulative
ثابت تخفیف	discount factor
جستجوی محیط	Exploration
جستجوی محیط	Explore
خطامشی	Policy
خطامشی بهینه	optimal policy
فرایند تصمیم‌گیری غیرقطعی مارکوف	nondeterministic Markov decision process
عامل	Agent
غیرقطعی	Nondeterministic
قلمرو	Domain
قطعی	Deterministic
الگوریتم‌های کاهش اختلاف ارزش‌ها	temporal difference algorithms
متوسط پاداش	average reward
محیط	Envirement
مربی	Trainer
نسبت دادن ارزش موقتی	temporal credit assignment
حالت	State
حالت جاذب	absorbing state