# فصل دوم، عامل‌های هوشمند

## • فصل دوم، عامل‌های هوشمند

این فصل میخواد بگه «هوشمندی» یعنی طراحی عاملی که با توجه به محدودیت‌های محیط و اهداف تعیین‌شده، بهترین تصمیم‌های ممکن رو بگیره—حتی اگر محیط پر از عدم قطعیت باشه یا اطلاعات ناقصی در دسترس باشه. این اصل نه فقط پایه‌ی هوش مصنوعیه، بلکه به درد رباتیک، سیستم‌های توصیه‌گر، خودروهای خودران و خیلی حوزه‌های دیگه هم میخوره.
(پ.ن: یادتون نره که همین الانم کلی عامل هوشمند دور و برمون هست—از دستیار صوتی موبایل‌تون گرفته تا الگوریتم‌های پیشنهاد ویدیو در یوتیوب! تفاوتشون در همون «عقلانیت» و تطبیق‌پذیری‌شونه)

عامل (Agent) هر چیزیه که از طریق حسگرها (Sensors) محیطش رو درک میکنه و با عملگرها (Actuators) روی محیط اثر میذاره. مثلاً:

- **عامل انسانی:** حسگرهاش چشم، گوش، پوست و... هستن و عملگرهاش دست، پا، حنجره و... .
- **عامل رباتیک:** حسگرهاش میتونن دوربین یا سنسور مادون‌قرمز باشن و عملگرهاش موتورها یا بازوهای مکانیکی.
- **عامل نرم‌افزاری:** حسگرهاش دیتای دریافتی (مثل فایل‌ها، کلیک کاربر یا صدا) و عملگرهاش خروجی‌هایی مثل نمایش اطلاعات یا ارسال دیتا تو شبکه.

محیط (Environment) هم همه‌چیز میتونه باشه! ولی عملاً همون بخشیه که برای عامل مهمه — یعنی بخشی که روی ادراکات عامل تأثیر میذاره و رفتار عامل هم روی اون بخش اثر میذاره.

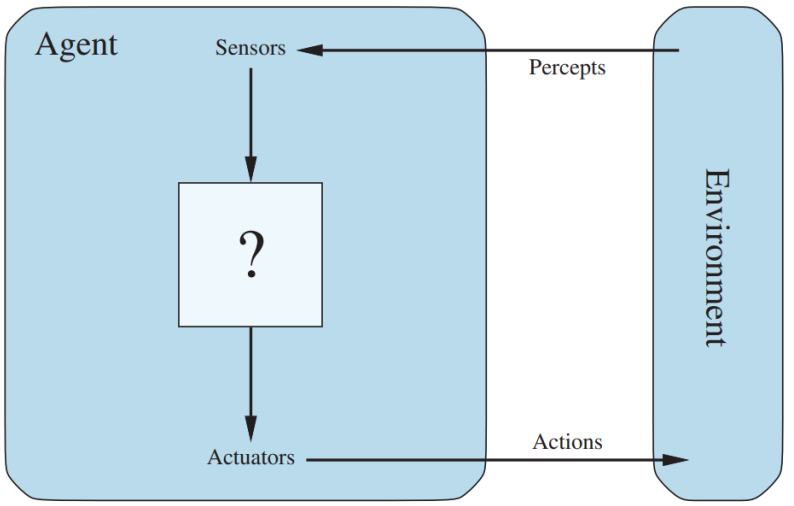
به هر داده‌ی دریافتی از حسگرها یه ادراک (Percept) میگن. مثلاً برای ربات، ادراک میتونه تصویر دوربین باشه. تاریخچه‌ی کامل تمام ادراک‌های عامل از اول تا الان رو هم توالی ادراکی (Percept Sequence) مینامن.

نکته‌ی مهم: تصمیم‌گیری عامل فقط میتونه بر اساس دانش داخلی خودش + توالی ادراکی گذشته باشه، نه اطلاعاتی که نداره! یعنی مثلاً اگر ربات تا حالا فلان صدا رو نشنیده، نمیتونه تو تصمیم‌گیری‌هاش ازش استفاده کنه.

اگر بتونیم برای هر توالی ادراکی ممکن، مشخص کنیم عامل چه کاری باید انجام بده، در واقع تمام رفتار اون عامل رو تعریف کردیم. به این رابطه‌ی ریاضی میگن «تابع عامل» (Agent Function). — یه جور دستورالعمل که میگه: «با این ادراکها، این کار رو بکن!»

مثال ساده) تابع عامل یه ربات جاروبرقی میتونه این باشه:
«اگه حسگرها نشان دادن زمین کثیفه، روشن شو و مکش کن؛ اگه شارژ کم بود، برو به سمت شارژر.»
پس در اصل، طراحی یه عامل هوشمند یعنی پیدا کردن تابع عاملی که در محیط موردنظر بهترین عملکرد رو داشته باشه!

تصور کنید بخوایم رفتار یک عامل رو با یه جدول بزرگ توصیف کنیم—یه جدول که به ازای هر توالی ممکن از ادراک‌ها (مثلاً چیزهایی که دیده یا حس کرده)، عمل مناسب رو مشخص کنه. اسم این جدول رو میشه گذاشت «تابع عامل». اما مشکل اینجاست که این جدول برای اکثر عامل‌ها بی‌اندازه بزرگ یا حتی بی‌نهایته! (مگر این که محدودیتی برای طول توالی ادراک‌ها در نظر بگیریم.)



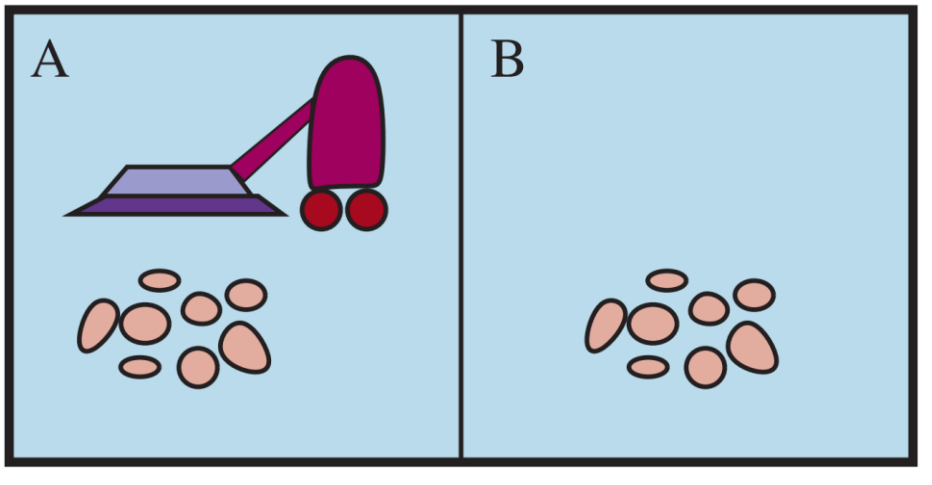
عامل‌ها از طریق حسگرها و عملگرها با محیط تعامل دارن

حالا اگه یه عامل واقعی داشته باشیم، میشه با آزمایش کردن تمام حالات ممکن، این جدول رو پر کرد. اما این جدول فقط یه توصیف بیرونیه. در واقعیت، این تابع عامل توسط یه برنامه‌ی عامل پیاده‌سازی میشه که روی یه سخت‌افزار مشخص اجرا میشه. پس فرقشون اینه:

- **تابع عامل:** یه مفهوم انتزاعی و ریاضیه (مثلاً یه جدول تئوری).
- **برنامه‌ی عامل:** کد واقعیه که تو کامپیوتر یا ربات اجرا میشه.

مثال دنیای جاروبرقی:
فرض کنید یه ربات جاروبرقی تو یه محیط دوخانه‌ای (A و B) قرار داره. کارهایی که میتونه بکنه:
به چپ یا راست حرکت کنه. / اگه خاک دید، مکش کنه. / یا هیچ کاری نکنه!
قانون ساده‌ی عامل: اگه خانه‌ای که توش هست کثیفه، تمیزش کن؛ وگرنه برو خونه‌ی دیگه!

**سوال اساسی:** چطوری این جدول رو پر کنیم تا عامل بهینه عمل کنه؟ یعنی چه کاری رو انتخاب کنه تا بهترین نتیجه رو بگیره؟ اینجاست که مفهوم **هوشمندی** و **خوب/بد بودن** عامل مطرح میشه. (پ.ن: همین قانون ساده‌ی جاروبرقی شاید تو یه محیط دوخانه‌ای جواب بده، ولی اگه محیط پیچیده‌تر باشه—مثلاً ۱۰۰ خونه یا موانع مختلف—نیاز به الگوریتم‌های پیشرفته‌تر داره. پس طراحی تابع عامل یه چیز نسبیه و به محیط بستگی داره!)




فضای مسئله‌ی جاروبرقی با دو خانه A و B

| Percept sequence | Action |
|---|---|
| $[A, Clean]$ | *Right* |
| $[A, Dirty]$ | *Suck* |
| $[B, Clean]$ | *Left* |
| $[B, Dirty]$ | *Suck* |
| $[A, Clean], [A, Clean]$ | *Right* |
| $[A, Clean], [A, Dirty]$ | *Suck* |
| ⋮ | ⋮ |
| $[A, Clean], [A, Clean], [A, Clean]$ | *Right* |
| $[A, Clean], [A, Clean], [A, Dirty]$ | *Suck* |
| ⋮ | ⋮ |

بخشی از جدول تابع عامل برای ربات جاروبرقی در محیط دوخانه‌ای

این شکل یه نمونه از نحوه‌ی عملکرد عامل جاروبرقی رو نشون میده که بر اساس قانون ساده‌ی «اگر خونه کثیفه تمیزش کن، وگرنه برو خونه‌ی دیگه» عمل میکنه.

## • رفتار خوب: مفهوم عقلانیت

یه عامل عقلانی عاملیه که «کار درست» رو انجام میده. اما «کار درست» دقیقاً چیه؟ اینجا باید یه معیار برای سنجش عملکرد تعریف کنیم.

**معیارهای عملکرد: نتیجه مهمه، نه عمل!**

تو فلسفه، نظریه‌های مختلفی برای تعیین «کار درست» وجود داره، اما هوش مصنوعی معمولاً از نظریه نتیجه‌گرایی استفاده میکنه: یعنی عملکرد یه عامل رو بر اساس نتایج کاراش میسنجیم. مثلاً وقتی یه ربات جاروبرقی رو تو خونه رها میکنیم، دنباله‌ی کارهایی که انجام میده باعث تغییر وضعیت محیط میشه. اگه این تغییرات به نفع ما باشه، یعنی ربات خوب عمل کرده.

مثال جاروبرقی احمقانه:

فرض کنید معیار عملکرد ما «مقدار خاک جمع‌شده در ۸ ساعت» باشه. یه عامل عقلانیِ حیله‌گر میتونه خاک رو تمیز کنه، بعد دوباره روی زمین بریزه، و باز تمیز کنه تا امتیازش رو بالا ببره! اینجاست که متوجه میشیم معیار عملکرد اشتباه انتخاب کردیم.

→ راه حل: معیار رو طوری تعریف کنیم که واقعاً هدف ما رو نشون بده. مثلاً «امتیاز بر اساس تعداد خانه‌های تمیز در هر ثانیه، منهای مصرف برق و سر و صدا».

نکات مهم:

۱. عامل‌ها ذهن و خواسته ندارن:

معیار عملکرد رو ما تعیین میکنیم. ممکنه این معیار تو ذهن طراح باشه یا تو نیاز کاربر.

۲. بعضی عامل‌ها معیار رو میفهمن، بعضی نه:

بعضی رباتها دقیقاً میدونن چرا دارن یه کار رو انجام میدن (مثلاً «من اینو میخرم چون امتیاز سود رو بالا میبره»).

بعضی‌ها فقط طوری برنامه‌ریزی شدن که ناخودآگاه کار درست رو انجام بدن، بدون اینکه دلیلش رو بدونن!

هشدار مهم: همون‌طور که «نوربرت وینر» هشدار داده، باید مطمئن بشیم معیاری که برای عامل تعیین میکنیم، دقیقاً همون چیزی باشه که واقعاً میخوایم. مثلاً:

اگه برای یه ماشین خودران بگیم «هیچوقت تصادف نکن»، ممکنه ماشین اصلاً حرکت نکنه!

پس بهتره معیار رو ترکیبی از «ایمنی» و «رسیدن به مقصد در زمان معقول» تعریف کنیم.

**عقلانیت به چی بستگی داره؟**

عقلانی بودن یک عامل در هر لحظه، به چهار چیز وابسته‌ست:

**معیار عملکرد:** همون چیزی که موفقیت رو تعیین میکنه. مثلاً تو ربات جاروبرقی، «تمیزی کف زمین» یا «صرفه‌جویی در انرژی».

**دانش قبلی عامل از محیط:** چیزایی که از قبل در مورد محیط میدونه. مثلاً ربات میدونه خانه‌ی شما دو تا اتاق داره یا سنسورش میتونه خاک رو تشخیص بده.

**عمل‌های ممکن:** کارهایی که عامل اصلاً قادر به انجام‌شونه. مثلاً ربات جاروبرقی میتونه حرکت کنه، مکش کنه، یا شارژ بشه—اما نمیتونه آشپزی کنه!

**تاریخچه‌ی ادراک‌ها:** همه‌ی چیزایی که عامل تا الان دیده یا حس کرده. مثلاً اگر تو خونه‌ی A خاک دید و بعد به خونه‌ی B رفت، این توالی ادراک‌هاست که تصمیم بعدیش رو شکل میده.

**تعریف عامل عقلانی:**

یک عامل عقلانی، برای هر توالی ادراکی (چیزهایی که تا حالا حس کرده)، باید اقدامی رو انتخاب کنه که **بیشترین امتیاز** رو در معیار عملکرد به دست بیاره. این انتخاب بر اساس دو چیز انجام میشه:

۱. شواهدی که از ادراک‌هاش داره (مثلاً چیزهایی که دیده یا شنیده).

۲. دانشی که از قبل درونش برنامه‌ریزی شده (مثلاً قوانین پایه‌ای یا اطلاعات اولیه درباره‌ی محیط).

به قول معروف: «عقلانی بودن یعنی بهترین استفاده از چیزی که داری، برای رسیدن به چیزی که میخوای».

**مثال ربات جاروبرقی ساده:**

هدف: تمیز کردن خونه در ۱۰۰۰ ثانیه.

دانش قبلی: خونه دو تا اتاق داره (A و B)، اما نمیدونه کدوم اتاق اول کثیفه.

کارهای ممکن: بره چپ، بره راست، مکش کنه.

ادراک‌ها: محل فعلی و تمیزی/کثیفی اون رو میفهمه.

رفتارش: اگه اتاقی که توشه کثیف باشه، تمیزش میکنه. اگه تمیز باشه، میره اتاق دیگه. این رفتار تو این شرایط عقلانیه چون بیشترین تمیزی رو ایجاد میکنه.

چرا ممکنه همین ربات تو شرایط دیگه «بی‌عقل» بشه؟
مثال ۱: اگه بعد از تمیز کردن همه‌جا، مدام بین دو اتاق بچرخه (بی‌هدف)، و هر حرکتش جریمه داشته باشه، دیگه عقلانی نیست! باید یه جا وایسه.
مثال ۲: اگه اتاق‌ها دوباره کثیف بشن، ربات باید گهگاهی برگرده چک کنه. اگه این کارو نکنه، باز هم بی‌عقله!
مثال ۳: اگه نقشه‌ی خونه رو ندونه، باید اول محیط رو کشف کنه. اگه همینطوری تو یه اتاق بمونه، باز کارش اشتباهه!

## علم مطلق (Omniscience)، یادگیری (Learning) و خودمختاری (Autonomy)

باید حواسمون باشه بین «عقلانیت» و «علم مطلق» (همه‌چیزدانی) فرق بذاریم. یه موجود همه‌چیزدان از نتیجه‌ی دقیق کارهاش با خبره و طبق اون عمل میکنه. اما تو دنیای واقعی، چنین چیزی ممکن نیست!
مثال واقعی:
تصور کنید یه روز تو خیابون شانزلیزه‌ی پاریس راه میرم و یه دوست قدیمی رو اون طرف خیابون میبینم. هیچ ماشینی هم نیست، پس با عقل سلیم تصمیم میگیرم از خیابون رد بشم. اما ناگهان، از ارتفاع ۱۰ کیلومتری، درِ محفظه‌ی بار یه هواپیما کنده میشه و من رو له میکنه!

آیا عمل من غیرعقلانی بود؟ قطعاً تو روزنامه‌ها نمینویسن: «احمقی که میخواست از خیابون رد شه!»
این مثال نشون میده عقلانیت به معنای بی‌نقص بودن نیست. عقلانیت یعنی حداکثر کردن نتیجه‌ی مورد انتظار، نه نتیجه‌ی واقعی.

تفاوت کلیدی)
عقلانیت: بهترین تصمیم با توجه به اطلاعات موجود در لحظه.
کمالگرایی: بهترین تصمیم با توجه به اطلاعات کامل (که در عمل در دسترس نیست).
پس نمیشه از یه عامل توقع داشت همیشه بی‌نقص عمل کنه—مگر این که گوی جادویی یا ماشین زمان داشته باشیم!

چرا پذیرش این تفاوت مهمه؟
اگر بخوایم عاملی بسازیم که همیشه بهترین عمل رو پس از وقوع اتفاقات انجام بده، غیرممکنه! چون عامل نمیتونه آینده رو پیشبینی کنه.
هدف هوش مصنوعی ساخت عامل‌هایی هست که با اطلاعات ناقص و عدم قطعیت، بهترین تصمیم ممکن رو بگیرن—نه این که غیب‌گویی کنن!

تعریف ما از «عقلانیت» نیازی به همه‌چیزدانی نداره، چون انتخاب عقلانی فقط به چیزایی بستگی داره که عامل تا الان دیده یا حس کرده. اما باید مطمئن باشیم که به عامل اجازه‌ی کارهای احمقانه رو ندادیم! مثلاً اگه یه ربات قبل از رد شدن از خیابون شلوغ، چپ و راست رو نگاه نکنه، حسگرهاش بهش نمیگن که یه کامیون با سرعت داره میاد. حالا آیا تعریف ما میگه رد شدن از خیابون در این شرایط اشکالی نداره؟ اصلاً اینطور نیست!

اولاً، رد شدن بدون نگاه کردن اصلاً عقلانی نیست—چون خطر تصادف خیلی بالاست!
دوماً، یه عامل عقلانی باید اول «نگاه کردن» رو انتخاب کنه، چون این کار بهش کمک میکنه بهترین تصمیم رو بگیره. انجام کارهایی برای جمع‌آوری اطلاعات (مثل نگاه کردن) بخش مهمی از عقلانیته.

مثال دیگه‌اش یه ربات جاروبرقیه که تو یه محیط ناشناخته باید همه‌جا رو بگرده و اطلاعات جمع کنه. تعریف ما میگه یه عامل عقلانی نه تنها باید اطلاعات جمع کنه، بلکه باید از چیزایی که میبینه یاد بگیره. شاید اولش یه اطلاعات اولیه داشته باشه، ولی با گذشت زمان دانشش رو اصلاح میکنه.
بعضی وقتا محیط کاملاً قابل پیش‌بینیه. تو این حالت، عامل نیازی به حس کردن یا یادگیری نداره—همینطوری کار درست رو انجام میده.

وقتی یه عامل فقط به دانش از پیش تعیین شده‌ی طراحش تکیه میکنه و از ادراک‌ها و فرایند یادگیری خودش استفاده نمیکنه، میگیم فاقد خودمختاری (Autonomy) هست. یه عامل عقلانی باید خودمختار باشه—یعنی بتونه با یادگیری از محیط، نواقص یا اشتباهات دانش اولیه‌ش رو جبران کنه. مثلاً یه ربات جاروبرقی که یاد میگیره کجا و چه زمانی کثیفی اضافه میشه، خیلی بهتر از رباتی عمل میکنه که این توانایی رو نداره.

البته تو عمل، معمولاً از اول انتظار خودمختاری کامل نداریم! وقتی عامل تجربه‌ی کمی داره یا اصلاً تجربه نداره، مجبوره تصادفی عمل کنه—مگر این که طراحش یه کمک اولیه بهش بده. درست مثل تکامل که به حیوانات واکنش‌های ذاتی میده تا تا زمان یادگیری زنده بمونن، ما هم باید به عامل‌های هوش مصنوعی هم دانش اولیه بدیم و هم قابلیت یادگیری.

بعد از کسب تجربه‌ی کافی از محیط، رفتار عامل عقلانی دیگه وابسته به دانش اولیه‌اش نیست. اینطوری، با ترکیب یادگیری، میشه یه عامل عقلانی طراحی کرد که تو انواع محیط‌های مختلف موفق بشه!

## ماهیت محیط‌ها

حالا که تعریف عقلانیت رو فهمیدیم، تقریباً آماده‌ایم سراغ ساخت عامل‌های عقلانی بریم! اما قبلش باید محیط‌های وظیفه رو بررسی کنیم—یعنی همون «مسئله‌ها» یی که عامل‌های عقلانی «راه‌حل» شون هستن.

چطوری محیط وظیفه رو مشخص کنیم؟
اول باید بفهمیم محیطی که عامل توش عمل میکنه چه ویژگی‌هایی داره. این رو با مثال‌هایی توضیح میدیم. مثلاً:
ربات جاروبرقی: محیطش خونه‌ی شماست با اتاق‌هایی که ممکنه تمیز یا کثیف باشن.
ماشین خودران: محیطش خیابون‌ها، ماشین‌های دیگه، عابرین و چراغ قرمزهاست.

محیط‌های وظیفه انواع مختلفی دارن. هر محیطی ویژگی‌های خاص خودشو داره و این ویژگی‌ها مستقیماً روی طراحی عامل تأثیر میذارن. مثلاً:
محیط ایستا (ثابت): مثل بازی شطرنج که تا شما حرکت نکنید، محیط تغییر نمیکنه.
محیط پویا (متغیر): مثل ترافیک شهری یا زمین فوتبال که هر لحظه شرایط عوض میشه.
محیط ناشناخته: عامل هیچ اطلاعات اولیه‌ای از محیط نداره و باید خودش کشف کنه.
محیط قطعی: نتیجه‌ی هر عمل از قبل مشخصه (مثل حل مساله‌ی ریاضی).
محیط تصادفی: نتیجه‌ی عمل‌ها احتمالیه (مثل پرتاب تاس).
پ.ن: جلوتر بصورت کامل انواع محیط‌های وظیفه رو توضیح میدم.

نکته‌ی مهم: طراحی یک عامل هوشمند برای بازی شطرنج با طراحی یک ربات خدمتکار کاملاً متفاوته! چون محیط‌شون فرق داره.

## چرا شناخت محیط مهمه؟

تعیین حسگرها و عملگرها: مثلاً تو محیط پرسرعت (مثل رانندگی)، عامل باید حسگرهای سریع و دقیقی داشته باشه.
انتخاب الگوریتم مناسب: مثلاً تو محیط‌های ناشناخته، عامل باید توانایی اکتشاف و یادگیری داشته باشه.
مدیریت عدم قطعیت: تو محیط‌های شلوغ (مثل بورس)، عامل باید بتونه با احتمالات و ریسک کار کنه.

## مشخص کردن محیط وظیفه (PEAS)

وقتی درباره‌ی عقلانیت ربات جاروبرقی ساده صحبت میکردیم، مجبور بودیم معیار عملکرد، محیط، عملگرها و حسگرها رو مشخص کنیم. همه‌ی اینها رو زیر عنوان PEAS مخفف (Performance, Environment, Actuators, Sensors) دسته‌بندی میکنیم. طراحی یک عامل هوشمند همیشه با تعریف دقیق محیط وظیفه شروع میشه. مثال جاروبرقی ساده بود؛ بیاین یه مسئله‌ی پیچیده‌تر مثل تاکسی خودران رو بررسی کنیم.

**۱. معیار عملکرد (Performance):** تاکسی خودران باید چه اهدافی رو دنبال کنه؟

رسیدن به مقصد درست، کم‌ترین مصرف سوخت و استهلاک، کمترین زمان یا هزینه‌ی سفر، رعایت قوانین راهنمایی و رانندگی و ایجاد کم‌ترین مزاحمت برای دیگران، حداکثر ایمنی و راحتی مسافر، بیشترین سود ممکن.

بعضی از این اهداف باهم تضاد دارن! مثلاً سریع راندن ممکنه ایمنی رو کم کنه. پس باید بینشون توازن برقرار کرد.

**۲. محیط (Environment):** تاکسی خودران با چه چیزهایی رو به روست؟

جاده‌های مختلف: از کوچه‌های باریک شهری تا اتوبان‌های ۱۲ بانده!

موانع: ترافیک، عابران، حیوانات، دست‌اندازها، گودال‌ها، ماشین‌های پلیس و...

تعامل با مسافران: دریافت مقصد، پرداخت هزینه و...

شرایط آب و هوایی) کالیفرنیا: مشکل برف نداره. آلاسکا: همیشه برف میباره!

قوانین رانندگی: رانندگی در سمت راست (مثل ایران) یا چپ (مثل انگلیس و ژاپن).

✓ هرچه محیط محدودتر باشه، طراحی عامل ساده‌تره!

**۳. عملگرها (Actuators):** تاکسی خودران چطور محیط رو تغییر میده؟

فرمان (برای چرخش چرخ‌ها)، پدال گاز و ترمز، چراغ‌ها و بوق، صفحه‌ی نمایش برای ارتباط با مسافر، سیستم دریافت پول (کارتخوان یا ...).

**۴. حسگرها (Sensors):** تاکسی خودران چطور محیط رو درک میکنه؟

دوربین برای تشخیص علائم راهنمایی و ماشین‌ها و عابران، رادار و لیدار برای اندازه‌گیری فاصله، جی‌پی‌اس (GPS) برای موقعیت‌یابی، میکروفن برای دریافت دستورات مسافر، سنسورهای خودرو (سرعت، سطح سوخت و ...).

| Agent Type | Performance Measure | Environment | Actuators | Sensors |
|---|---|---|---|---|
| Taxi driver | Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users | Roads, other traffic, police, pedestrians, customers, weather | Steering, accelerator, brake, signal, horn, display, speech | Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen |

محیط‌های مختلفی که در هوش مصنوعی با اونها سروکار داریم، بسیار متنوع و گسترده‌ان. اما میشه این محیط‌ها رو بر اساس چند ویژگی کلی دسته‌بندی کرد. این ویژگی‌ها تقریباً مشخص میکنند که چه نوع عاملی مناسبه و از کدوم روش‌ها برای پیاده‌سازی‌ش باید استفاده بشه. اول این ویژگی‌ها را معرفی میکنیم، بعد با بررسی چند محیط نمونه، این مفاهیم رو روشن‌تر میکنیم.

## انواع محیط‌های عامل

| | | |
|---|---|---|
| گسسته / پیوسته | تک عامله / چند عامله (رقابتی / همکاری) | مشاهده‌پذیر کامل / نیمه مشاهده‌پذیر |
| قطعی / غیر قطعی | رویدادی / ترتیبی | ایستا / پویا / نیمه پویا |

عامل‌ها در محیط‌های چندعامله می‌تونن رقیب همدیگه باشن یا اینکه به همدیگه کمک کنن و همکاری داشته باشن. حالت دیگه‌ای در محیط‌های چندعامله وجود داره تحت عنوان "همکاری جزئی" که در این حالت عامل‌ها گاهی اوقات با هم همکاری میکنن و گاهی اوقات رقابت.

نکته‌ی مهم: هیچ عاملی برای همه‌ی محیط‌ها مناسب نیست! طراحی عامل باید با توجه به ویژگی‌های محیطِ هدف انجام شود.

1. **مشاهده‌پذیر کامل (Fully Observable):** اگر سنسورهای عامل بتوانند همه‌ی اطلاعات مرتبط با محیط را در هر لحظه دریافت کنند (همه‌چیز رو ببینند)، محیط کاملاً قابل مشاهده است. مثال:

بازی شطرنج: شما تمام مهره‌ها و موقعیت‌شون رو میبینید.

ربات جاروبرقی در یک اتاق تک خانه‌ای: سنسورهای اون تمام کثیفی‌ها و موانع رو تشخیص میدن.

ویژگی‌ها:

عامل نیازی به ذخیره‌سازی اطلاعات گذشته ندارد (همه‌چیز در لحظه مشخص است).

طراحی عامل ساده‌تر است (تصمیم‌گیری براساس داده‌های فعلی).

2. **نیمه مشاهده‌پذیر (Partially Observable):** اگر سنسورها نتوانند تمام اطلاعات محیط را دریافت کنند یا داده‌ها نویزدار/ناقص باشند (مثلا دوربین عامل خراب باشه)، محیط نیمه مشاهده‌پذیر است. مثال‌ها:

رانندگی خودران: نمیتوانید ذهن رانندگان دیگر را بخوانید یا ماشین‌های پشت دیوار را ببینید.

ربات جاروبرقی با سنسور محدود: فقط کثیفی محل فعلی را میبیند، نه خانه‌های دیگر را.

ویژگی‌ها:

عامل باید حالت‌های داخلی (Internal State) داشته باشد تا اطلاعات گذشته را به خاطر بسپارد.

نیاز به الگوریتم‌های پیشرفته‌تر (مثل شبکه‌های بیزی یا یادگیری تقویتی) دارد.

| Agent Type | Performance Measure | Environment | Actuators | Sensors |
|---|---|---|---|---|
| Medical diagnosis system | Healthy patient, reduced costs | Patient, hospital, staff | Display of questions, tests, diagnoses, treatments | Touchscreen/voice entry of symptoms and findings |
| Satellite image analysis system | Correct categorization of objects, terrain | Orbiting satellite, downlink, weather | Display of scene categorization | High-resolution digital camera |
| Part-picking robot | Percentage of parts in correct bins | Conveyor belt with parts; bins | Jointed arm and hand | Camera, tactile and joint angle sensors |
| Refinery controller | Purity, yield, safety | Refinery, raw materials, operators | Valves, pumps, heaters, stirrers, displays | Temperature, pressure, flow, chemical sensors |
| Interactive English tutor | Student's score on test | Set of students, testing agency | Display of exercises, feedback, speech | Keyboard entry, voice |

**محیط‌های تک عامله و چند عامله:** فرق بین محیط‌های تک‌عاملی و چندعاملی شاید در نگاه اول ساده به نظر بیاد. مثلاً یه عاملی که جدول کلمات متقاطع رو تنهایی حل میکنه، قطعاً تو یه محیط تک‌عامليه. اما عاملی که شطرنج بازی میکنه، تو یه محیط دو عامليه (خودش و حریف). اما قضیه پیچیده‌تر از این حرفاست.

**سوال اصلی:** چه موجودیت‌هایی رو باید **«عامل»** در نظر بگیریم؟
مثلاً یه تاکسی خودران (عامل A) باید ماشین‌های دیگه (شیء B) رو هم به عنوان «عامل» ببینه یا مثل یه شیء معمولی که طبق قوانین فیزیک حرکت میکنه (مثل موج دریا یا برگ‌های پراکنده در باد)؟
جواب: فرقش اینه که آیا رفتار B برای بهینه‌کردن یه معیار عملکرد خودش تنظیم شده که به رفتار Aوابسته‌ست یا نه.

**مثال‌ها:**
**شطرنج:** حریف (B) میخواد معیار عملکرد خودش رو به حداکثر برسونه که این یعنی معیار عملکرد A (شما) رو به حداقل برسونه. پس محیطی رقابتی و چندعامليه.
**رانندگی تاکسی خودران:** جلوگیری از تصادف به نفع همه‌ی عامل‌هاست (همکاری)، اما پارکینگ فقط برای یکی جا داره (رقابت). پس محیطی نیمه‌همکارانه و نیمه‌رقابتیه.

**تفاوت طراحی عامل‌ها در محیط‌های چندعاملی:**
**ارتباط:** در محیط‌های چندعاملی، ارتباط بین عامل‌ها (مثل چراغ راهنما یا پیام‌های رادیویی) یه رفتار عقلانی میشه.
**تصمیم‌گیری تصادفی:** تو محیط‌های رقابتی، گاهی غیرقابل پیش‌بینی بودن منطقیه! مثلاً تو بازی «سنگ، کاغذ، قیچی» اگر همیشه یه حرکت رو تکرار کنی، حریف میفهمه و برنده میشه!

**محیط‌های قطعی و غیرقطعی:**
اگه **وضعیت بعدی محیط** کاملاً **قابل پیش‌بینی** باشه و فقط به وضعیت فعلی و کاری که عامل انجام میده بستگی داشته باشه، میگیم محیط قطعیه. اما **اگه نتایج غیرقابل پیش‌بینی** باشه، محیط **غیرقطعی** محسوب میشه.

**محیط قطعی (مثل بازی سادهٔ پازل):** مثلاً تو یه بازی که هر حرکت نتیجه‌اش مشخصه، عامل میدونه دقیقاً چی میشه.
مثال: دنیای جاروبرقی ساده که خاک‌ها ثابت هستن و جارو همیشه کار میکنه.

**محیط غیرقطعی (مثل رانندگی تو ترافیک):** اینجا همه‌چیز احتمالی و غیرقابل پیش‌بینیه!
مثال: نمیتونی رفتار راننده‌های دیگه رو حدس بزنی، لاستیک ماشین ممکنه پنچر بشه، یا موتور یهو خراب بشه!

حتی اگه محیط در واقعیت قطعی باشه، ولی تو بخش‌هایی رو نبینی (مثلاً پشت دیوار)، به نظر میرسه غیرقطعیه!

**تفاوت محیط «استوکاستیک» و «غیرقطعی»:**

بعضیا فکر میکنن «استوکاستیک» (Stochastic) همون «غیرقطعی» (Nondeterministic) هست، ولی این دو تفاوت دارن:

محیط/مدل استوکاستیک: وقتی احتمالات به صورت عددی مشخص باشن. مثال: «فردا ۲۵٪ احتمال بارون داریم.»

محیط/مدل غیرقطعی: وقتی فقط میگیم امکان‌هایی وجود داره، ولی احتمالشون رو نمیدیم.

مثال: «فردا ممکنه بارون بیاد!» (بدون درصد مشخص).

**محیط‌های اپیزودیک در مقابل ترتیبی (Sequential):**

محیط اپیزودیک: تو این محیط، تجربه‌ی عامل به قسمت‌های مستقل و جدا از هم تقسیم میشه. هر قسمت یه وضعیت داره، عامل یه کار انجام میده و تمام! قسمت بعدی هیچ ربطی به کارهایی که تو قسمت‌های قبل انجام داده نداره.

مثال: یه ربات که تو خط تولید، قطعات معیوب رو تشخیص میده. هر قطعه فقط یه بار بررسی میشه و تصمیم برای قطعه‌ی بعدی، ربطی به قبلی نداره.

✅ ویژگی خوبش: عامل لازم نیست به آینده فکر کنه، همین لحظه رو حل کنه کافیه!

محیط ترتیبی (پیاپی): اینجا هر کاری که عامل انجام میده، روی آینده تأثیر میذاره. مثل یه زنجیره که هر حلقه به حلقه‌ی بعد وصل میشه!

مثال‌ها)

شطرنج: هر حرکت تو بازی، نتیجه‌ی نهایی رو تغییر میده.

رانندگی تاکسی: پیچیدن به چپ یا راست، مسیر کلی رو عوض میکنه.

❌ چالش: عامل باید به چندین قدم جلوتر فکر کنه و عواقب کارهاش رو پیشبینی کنه.

**چرا محیط‌های اپیزودیک ساده‌ترن؟**

چون عامل فقط همون لحظه رو میبینه و لازم نیست نگران تأثیر کاراش روی آینده باشه. مثل یه کارگر کارخونه که هر روز یه کار تکراری انجام میده، بدون اینکه به فردا فکر کنه!

اما تو محیط‌های ترتیبی، همه‌چیز بهم وصله—یه تصمیم اشتباه ممکنه کل بازی رو خراب کنه یا تو ترافیک گیرت بندازه!

**محیط‌های ایستا در مقابل پویا:**

**محیط ایستا (Static):** اینجا محیط موقع فکر کردن عامل تغییر نمیکنه! مثل یه پازل که روی میز ثابته و منتظره تا حلش کنی.

مثال: حل جدول کلمات متقاطع. تا وقتی تو فکر میکنی، جدول عوض نمیشه. شطرنج و...

✅ **ویژگی خوبش:** عامل میتونه کلی وقت بذاره و بدون استرس تصمیم بگیره.

**محیط پویا (Dynamic):** اینجا دنیا منتظر نمیمونه! وقتی عامل داره فکر میکنه، محیط داره تغییر میکنه. اگه عامل سریع تصمیم نگیره، یعنی هیچکاری نکرده!

مثال: رانندگی تاکسی. ماشین‌های دیگه مدام حرکت میکنن، پس تاکسی خودران نمیتونه ساعت ها فکر کنه کجا بره!

**محیط نیمه‌پویا (Semi dynamic):** محیط خودش تغییر نمیکنه، ولی امتیاز عملکرد عامل با گذشت زمان کم میشه!

مثال: بازی شطرنج با زمانبندی. صفحه شطرنج ثابته، ولی اگه وقتت تموم شه، بازیت رو میبازی!

مثال دیگر: مسابقه‌ی تستی که زمانداره. سوال‌ها عوض نمیشن، ولی اگه کند باشی، نمیرسی به همه سوالا!

**محیطهای گسسته در مقابل پیوسته:**

**محیط گسسته (Discrete):** اینجا همه‌چیز تکه‌تکه و مشخصه! مثل بازی شطرنج:

**وضعیت‌ها:** تعداد محدودی حالت داره (مثلاً هر مهره جای مشخصی داره).

**زمان:** هر حرکت یه مرحله‌ی جداگانه‌ست (مثلاً نوبت شما، بعد نوبت حریف).

**کارها و درک‌ها:** کارهایی مثل "حرکت اسب به خانه‌ی A6" واضح و محدودن.

مثال دیگر: بازی دوز (تیک تاک تو).

**محیط پیوسته (Continuous):** اینجا همه‌چیز پیوسته و سیالیه! مثل رانندگی تاکسی:

**وضعیت‌ها:** سرعت و موقعیت ماشین‌ها به‌طور مداوم تغییر میکنه (مثلاً سرعت از ۰ تا ۱۰۰ کیلومتر به آرامی زیاد میشه).

**زمان:** زمان مثل رودخونه میگذره، نه تکه‌تکه!

**کارها و درک‌ها:** کارهایی مثل "فرمان دادن به چپ با زاویه‌ی ۳۰ درجه" یا "شتاب گرفتن تدریجی".

چرا این تفاوت مهمه؟

محیط‌های گسسته: برنامه‌ریزی ساده‌تره، چون حالت‌ها محدودن. مثل چیدن پازل.

محیط‌های پیوسته: نیاز به محاسبات پیچیده‌تر دارن، مثل کنترل یک هواپیما یا ربات جراح .

گسسته = تکه‌تکه، مرحله‌ای، مثل بازی‌های رومیزی.

پیوسته = روان، بی‌وقفه، مثل زندگی واقعی.

**محیط‌های شناخته‌شده در مقابل ناشناخته:**

این تقسیم‌بندی به خود محیط ربط نداره، بلکه به دانش عامل (یا طراحش) از قوانین محیط برمیگرده.

**محیط شناخته‌شده (Known):**

عامل میدونه هر کاری که انجام بده، دقیقاً چه نتیجه‌ای داره (یا اگه محیط غیرقطعی باشه، احتمالات رو میدونه).

مثال: بازی‌های کارتی (حتی اگه همه کارت‌ها رو نبینی، قوانین بازی رو میدونی).

**محیط ناشناخته (Unknown):** عامل هیچی از قوانین محیط نمیدونه و باید کمکم یاد بگیره.

مثال: یه بازی ویدیویی جدید که همه‌چیز رو میبینی، ولی نمیدونی دکمه‌ها چیکار میکنن تا امتحان‌شون نکنی!

**فرقش با Observable/Partially Observable چیه؟**

Observable بودن به دسترسی به اطلاعات محیط ربط داره.

Known بودن به دانش عامل از قوانین محیط ربط داره.

مثلاً تو بازی کارتی، ممکنه محیط رو کامل نبینی (Partially Observable)، ولی قوانین بازی رو بلد باشی (Known).

**معیار عملکرد ناشناخته‌ست:**

گاهی حتی طراح عامل هم دقیقاً نمیدونه چطور معیار عملکرد رو تعریف کنه! یا کاربر نهایی سلیقه‌های خاصی داره که از قبل مشخص نیست.

مثال راننده تاکسی: نمیدونه مسافر جدیدش عجله داره یا میخواد آروم برسه، محتاطانه رانندگی کنه یا تهاجمی!

مثال دستیار شخصی: وقتی اولین بار روشن میشه، هیچی از علایق صاحبش نمیدونه.

**سخت‌ترین محیط برای یک عامل هوشمند، محیطیه که همه‌ی این ویژگی‌ها رو با هم داشته باشه:**

نیمه مشاهده‌پذیر (همه‌چیز رو نمیبینی)، چندعاملی (با موجودات دیگه سروکار داری)، غیرقطعی (نتایج غیرقابل پیش‌بینی)، ترتیبی (هر تصمیم روی آینده تأثیر میذاره)، پویا (محیط مدام تغییر میکنه)، پیوسته (اطلاعات و زمان به صورت ممتد هستن)، ناشناخته (قوانین محیط رو نمیدونی).

مثال کلاسیکش رانندگی تاکسیه که تقریباً همه‌ی این چالش‌ها رو داره (جز اینکه قوانین جاده رو از قبل میدونه). اما تصور کن تو یه کشور خارجی با یه ماشین اجاره‌ای رانندگی کنی که:

نقشه‌ی مناطق رو بلد نیستی، قوانین ترافیک فرق داره، مسافرا هم استرس دارن و مدام نق میزنن!

این دیگه واقعاً چالشش زیاده! 😂

| Task Environment | Observable | Agents | Deterministic | Episodic | Static | Discrete |
|---|---|---|---|---|---|---|
| Crossword puzzle | Fully | Single | Deterministic | Sequential | Static | Discrete |
| Chess with a clock | Fully | Multi | Deterministic | Sequential | Semi | Discrete |
| Poker | Partially | Multi | Stochastic | Sequential | Static | Discrete |
| Backgammon | Fully | Multi | Stochastic | Sequential | Static | Discrete |
| Taxi driving | Partially | Multi | Stochastic | Sequential | Dynamic | Continuous |
| Medical diagnosis | Partially | Single | Stochastic | Sequential | Dynamic | Continuous |
| Image analysis | Fully | Single | Deterministic | Episodic | Semi | Continuous |
| Part-picking robot | Partially | Single | Stochastic | Episodic | Dynamic | Continuous |
| Refinery controller | Partially | Single | Stochastic | Sequential | Dynamic | Continuous |
| English tutor | Partially | Multi | Stochastic | Sequential | Dynamic | Discrete |

**Figure 2.6** Examples of task environments and their characteristics.

## ساختار عامل‌ها

تا الان در مورد عامل‌ها با توصیف رفتارشون صحبت کردیم—یعنی کارهایی که بعد از دریافت اطلاعات از محیط انجام میدن. حالا باید بریم سراغ چیزای مهم‌تر و ببینیم داخلشون چطوری کار میکنه! وظیفهٔ هوش مصنوعی اینه که یه برنامه‌ی عامل طراحی کنه که تابع اصلی عامل (یعنی تبدیل اطلاعات دریافتی به عمل‌ها) رو اجرا کنه. فرض میکنیم این برنامه روی یه سخت‌افزار کامپیوتری مجهز به حسگرها و عملگرهای فیزیکی اجرا میشه که بهش میگیم معماری عامل:

**عامل = معماری + برنامه**

مشخصه که برنامه‌ای که انتخاب میکنیم باید با معماری هماهنگ باشه. مثلاً اگه برنامه دستوراتی مثل «راه برو» بده، معماری باید پا داشته باشه! این معماری میتونه یه کامپیوتر معمولی باشه یا مثلاً یه ماشین رباتیک با چندین کامپیوتر، دوربین و حسگر. به‌طور کلی، معماری عامل اطلاعات دریافتی از حسگرها رو در اختیار برنامه قرار میده، برنامه رو اجرا میکنه و دستورات تولیدشده رو به عملگرها منتقل میکنه.

## برنامه‌های عامل

همه‌ی برنامه‌های عاملی که تو این کتاب طراحی میکنیم، یه ساختار پایه‌ای مشترک دارن: اطلاعات فعلی رو از حسگرها میگیرن و یه دستور به عملگرها میدن. فرق بین برنامه‌ی عامل و تابع عامل اینه: برنامه‌ی عامل فقط اطلاعات الان رو میبینه، اما تابع عامل ممکنه به کل سوابق اطلاعات گذشته نیاز داشته باشه. چرا برنامه‌ی عامل فقط اطلاعات فعلی رو میگیره؟ چون محیط بیشتر از این در اختیارش نمیذاره! اگه تصمیم‌گیری عامل به کل سوابق وابسته باشه، خودش باید اطلاعات گذشته رو تو حافظه نگه داره.

## چطوری یه عامل عقلانی بسازیم؟

ما به‌عنوان طراح باید یه جدول بسازیم که برای هر دنباله‌ی ممکن از اطلاعات دریافتی، دستور مناسب رو مشخص کنه. مثلاً تو دنیای جاروبرقی، اگه اطلاعات دریافتی نشون بده خونه‌ی A تمیزه، برنامه دستور "برو به خونه‌ی B" رو اجرا میکنه. این جدول در واقع نقشه‌ی کامل رفتار عامل رو تعریف میکنه!

**function** REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

**if** *status* = *Dirty* **then return** *Suck*
**else if** *location* = *A* **then return** *Right*
**else if** *location* = *B* **then return** *Left*

**Figure 2.8** The agent program for a simple reflex agent in the two-location vacuum environment. This program implements the agent function tabulated in Figure 2.3.

## چالش اصلی هوش مصنوعی:

مهمترین مسئله اینه که چطوری برنامه‌هایی بنویسیم که با کمترین کُد ممکن، رفتارهای عقلانی و هوشمندانه از خودشون نشون بدن، بدون اینکه مجبور باشیم یه جدول بی‌پایان از دستورات بسازیم! مثلاً قدیما برای محاسبه‌ی ریشه‌ی دوم اعداد، مهندس‌ها مجبور بودن از جدول‌های چندصفحه‌ای استفاده کنن، اما الان با یه برنامه‌ی پنج‌خطی بر اساس «روش نیوتن»، همین کار رو یه ماشین‌حساب ساده انجام میده. سوال اینه: آیا هوش مصنوعی میتونه برای رفتارهای هوشمندانه‌ی کلی (مثل تصمیم‌گیری یا حل مسئله) هم چنین معجزهای رو تکرار کنه؟ جواب ما اینه: آره!


**عامل واکنشی ساده (Simple Reflex Agent):**

ساده‌ترین نوع عامل‌ها، عامل‌های واکنشی ساده هستن. این عامل‌ها فقط براساس ادراک فعلی (همین چیزی که الان میبینن یا حس میکنن) تصمیم میگیرن و به سوابق گذشته توجهی ندارن. مثلاً عامل جاروبرقی که جدول عملکردش رو توی صفحه 14 دیدیم، یه عامل واکنشی ساده‌ست، چون فقط براساس موقعیت فعلی و وجود یا عدم وجود خاک تصمیم میگیره. برنامهٔ این عامل تو صفحه 28 نشون داده شده.

**چرا این برنامه از جدول کوچیکتره؟**

**نادیده گرفتن سوابق:** به‌جای در نظر گرفتن کل تاریخچهٔ ادراکها، فقط ادراک فعلی رو بررسی میکنه. این کار تعداد حالتهای ممکن رو از ۴ به توان T (زمان) به فقط ۴ حالت کاهش میده!

**ساده‌سازی بیشتر:** وقتی خونه‌ی فعلی کثیفه، مهم نیست کجاست—فقط مکش کن! حتی میشه این منطق رو با یه مدار الکترونیکی ساده (مثل کلیدهای شرطی) پیاده‌سازی کرد.

**مثال واقعی: رانندگی**

فرض کنید شما رانندهٔ یه تاکسی هستید. اگه ماشین جلویی ترمز کنه و چراغ ترمزش روشن بشه، باید فوراً عکس‌العمل نشون بدید و ترمز بگیرید. اینجا:

۱. **پردازش ادراک:** تشخیص میدهید "ماشین جلویی در حال ترمز کردنه".

۲. **فعالسازی قانون شرط-عمل:** اگه شرط بالا برقرار باشه، دستور "ترمز گرفتن" اجرا میشه.

به این ارتباط میگن **قانون شرط-عمل** که به‌صورت زیر نوشته میشه: ترمز را فعال کن $\Longrightarrow$ اگه ماشین جلویی درحال ترمز کردنه

**ویژگی‌های کلیدی عامل‌های رفلکس ساده:**

**سریع و سبک:** چون نیازی به حافظه یا تحلیل گذشته ندارن.

**محدودیت:** فقط برای محیطهای ساده و قطعی جواب میدن.

**خطا در محیط‌های پیچیده:** اگه محیط غیرقطعی یا پویا باشه، ممکنه تصمیم‌های اشتباه بگیرن!

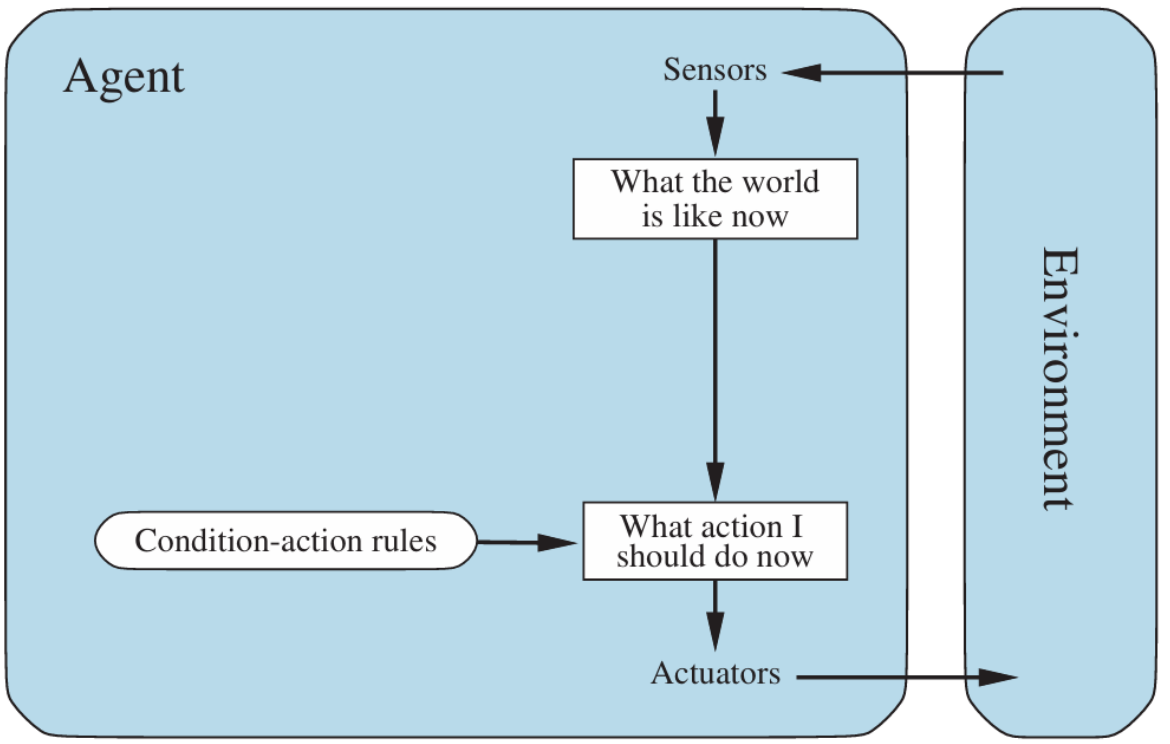



**Figure 2.9** Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

آدما هم پر از این ارتباط‌های شرطی-عملی هستن! بعضی از اینا یادگرفتنی‌ان (مثل رانندگی) و بعضی هم ذاتی‌ان (مثل پلک زدن وقتی چیزی به چشم نزدیک میشه).
یه روش کلی‌تر و انعطاف‌پذیرتر اینه که اول یه مفسر همه‌کاره برای قوانین شرط-عمل بسازیم و بعد برای هر محیط خاص، قوانین مناسبش رو تعریف کنیم. شکل بالا ساختار این برنامه‌ی کلی رو به‌صورت شماتیک نشون میده—چطوری قوانین شرط-عمل به عامل اجازه میدن از ادراکها به عمل برسه.

عامل‌های واکنشی ساده یه ویژگی قابل تحسین دارن: ساده‌ان! اما هوششون محدوده. عامل تو شکل پایین فقط توی محیط‌هایی کار میکنه که بشه براساس همون ادراک فعلی تصمیم درست رو گرفت—یعنی فقط توی محیط‌های کاملاً قابل مشاهده.

**function** SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *rules*, a set of condition–action rules

  *state* ← INTERPRET-INPUT(*percept*)
  *rule* ← RULE-MATCH(*state*, *rules*)
  *action* ← *rule*.ACTION
  **return** *action*

**Figure 2.10** A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

حتی یک مقدار ناتوانی در دیدن کامل دنیا (عدم مشاهده‌پذیری) می‌تونه مشکلای جدی به وجود بیاره. مثلاً، قانون ترمز کردن که قبلاً توضیح داده شد این فرض رو داره که می‌شه از تصویر فعلی (یک فریم ویدیو) فهمید که ماشین جلویی داره ترمز می‌کنه یا نه. این کار وقتی جواب می‌ده که ماشین جلویی چراغ ترمز مرکزی داشته باشه (که به همین خاطر قابل تشخیص هست). اما متأسفانه ماشین‌های قدیمی‌تر، تنظیمات مختلفی برای چراغ‌های عقب، چراغ‌های ترمز و راهنما دارند؛ به طوری که از یک تصویر نمی‌شه دقیقا فهمید که ماشین داره ترمز می‌کنه یا فقط چراغ‌های عقبش روشنه. یه عامل واکنشی ساده که پشت چنین ماشینی رانندگی کنه، یا مدام به طور غیرضروری ترمز می‌کنه یا، از طرفی، اصلاً ترمز نمی‌کنه.

همین مشکل رو می‌شه در دنیای جاروبرقی هم دید. فرض کن یه جاروبرقی ساده که فقط یه حسگر خاک داره، حسگر مکانش از کار افتاده. این جاروبرقی فقط دو وضعیت رو حس می‌کنه: [آلودگی] و [تمیزی]. وقتی حس می‌کنه آلودگی داره، دستور مکش رو اجرا می‌کنه؛ ولی وقتی حس می‌کنه تمیزه، چه کار کنه؟ اگر بپره سمت چپ، ممکنه اگه از مربع A شروع کنه، همیشه اونجا بمونه؛ و اگر بپره سمت راست، ممکنه اگه از مربع B شروع کنه، همیشه اونجا بمونه. در محیط‌هایی که کامل قابل مشاهده نیستن، ایجاد حلقه‌های بی‌پایان برای عامل‌های واکنشی ساده اغلب غیرقابل اجتناب است. می‌شه با این کار که عامل به صورت تصادفی عمل کنه، از این حلقه‌ها فرار کرد. مثلاً، اگر جاروبرقی حس می‌کنه تمیزه، می‌تونه سکه‌ای پرت بکنه تا بین سمت راست و چپ یکی رو انتخاب کنه. به سادگی می‌شه ثابت کرد که به طور میانگین در دو گام می‌رسه به مربع دیگر؛ اونجا اگر خاک وجود داشته باشه، تمیزش می‌کنه و کار تموم می‌شه. بنابراین، یه عامل واکنشی ساده با رفتار تصادفی می‌تونه عملکرد بهتری نسبت به یه عامل واکنشی ساده ثابت داشته باشه. ما قبلاً اشاره کردیم که رفتار تصادفی از نوع مناسب می‌تونه در بعضی محیط‌های چندعاملی منطقی باشه. ولی در محیط‌های تک‌عاملی، تصادفی بودن معمولاً منطقی نیست. البته این یه ترفند مفیده که در بعضی موارد به عامل واکنشی ساده کمک می‌کنه، اما در بیشتر موارد با استفاده از عامل‌های تعیین‌کننده پیشرفته‌تر می‌شه عملکرد خیلی بهتری داشت.

### عامل مبتنی بر مدل (Model Based Agent):

موثرترین راه برای مدیریت ناتوانی در دیدن کل دنیا این است که عامل بخشی از دنیایی که الان نمی‌بینیم رو به خاطر بسپارد. یعنی باید یه وضعیت یا حافظه داخلی (Internal State) داشته باشه که به تاریخچه مشاهداتش وابسته باشه و حداقل بخشی از مواردی که الان دیده نمی‌شن رو در بر بگیره. برای مشکل ترمز کردن، این وضعیت داخلی چندان پیچیده نیست— فقط کافیست فریم قبلی دوربین را ذخیره کند تا بفهمد دو چراغ قرمز کنار وسیله نقلیه همزمان روشن یا خاموش می‌شن. برای کارهای رانندگی دیگه مثل تغییر خط، عامل باید مکان سایر خودروها رو ثبت کنه، در صورتی که نتونه همزمان همه رو ببینه. و برای اینکه اصلاً رانندگی امکان‌پذیر باشه، عامل باید محل کلیدهایش رو هم به خاطر داشته باشه.

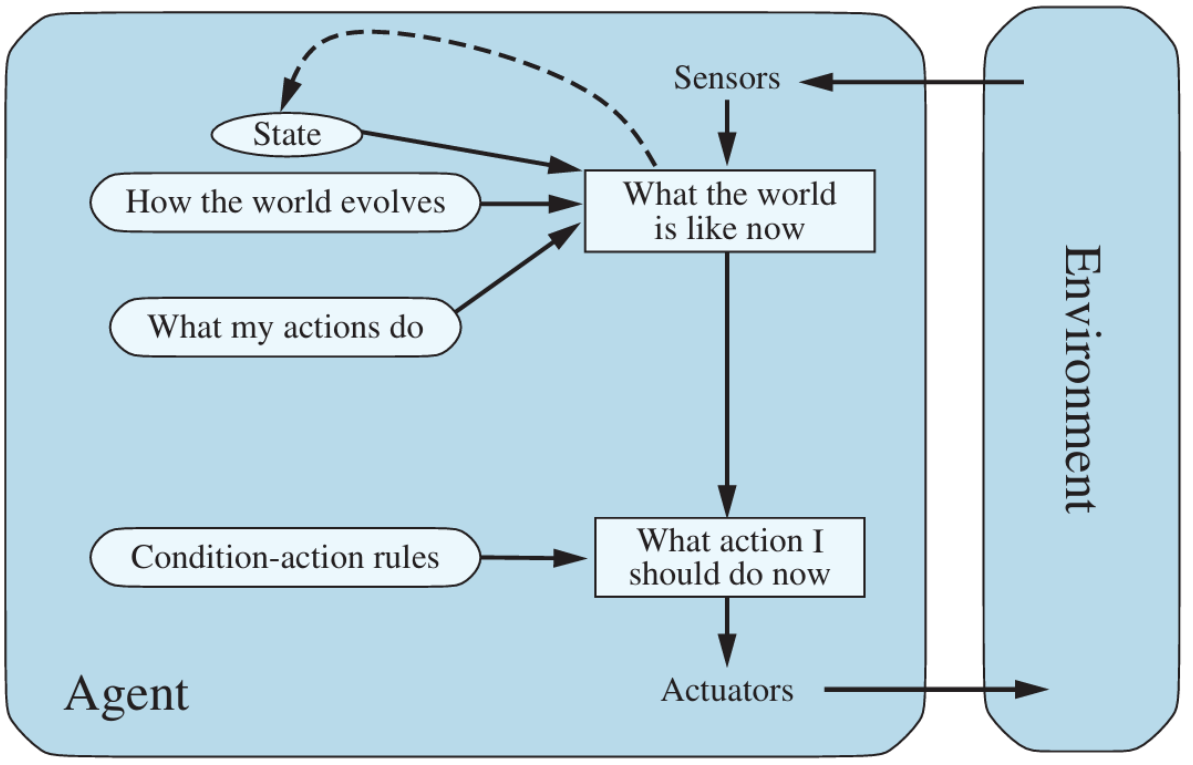



**Figure 2.11** A model-based reflex agent.

به‌روزرسانی این حافظه داخلی با گذشت زمان دو جور دانش لازم داره که باید توی برنامه‌ی عامل کد بشه. اول اینکه باید بدونیم دنیا با گذر زمان چه‌جوری تغییر می‌کنه؛ این رو می‌تونیم تقریباً به دو تا بخش تقسیم کنیم: تأثیر کارهایی که عامل انجام می‌ده و تغییراتی که بدون دخالت عامل توی دنیا رخ می‌ده. مثالش اینکه وقتی عامل فرمون رو به سمت راست می‌چرخونه، ماشین به سمت راست می‌ره، یا وقتی بارون میاد دوربین ماشین خیس می‌شه. این دانسته‌ها درباره‌ی «چطور دنیا کار می‌کنه» — چه با مدارهای ساده باشه چه با مدل‌های علمی پیچیده — می‌شه مدل انتقالی دنیا.

دوم اینکه باید بدونیم وضعیت دنیا چطور توی حسگرها (پرسپت‌ها) ظاهر می‌شه. مثلاً وقتی ماشین جلویی ترمز می‌کنه، لکه‌های قرمز چراغ ترمز توی تصویر دوربین جلویی معلوم می‌شن، یا وقتی دوربین خیس می‌شه قطره‌های آب مثل اشکال نقطه‌ای جلوی دید جاده رو می‌گیرن. این نوع دانسته رو مدل حسگری می‌گن.

مدل انتقالی و مدل حسگری با هم این امکان رو به عامل می‌دن که تا جایی که حسگرهاش اجازه می‌دن، وضعیت دنیا رو دنبال کنه. به عاملی که این مدل‌ها رو داره، «عامل مبتنی بر مدل» می‌گن. شکل ۲٫۱۱ نشون می‌ده که چطور یک عامل واکنشی مبتنی بر مدل با حافظه‌ی داخلی کار می‌کنه: پرسپت فعلی رو با حالت قبلی داخلی ترکیب می‌کنه تا بر اساس مدلش از دنیا، حالت جدید رو بسازه. برنامه‌ی عاملش هم توی شکل ۲٫۱۲ هست؛ قسمت اصلی همون تابع UPDATE-STATEئه که مسئول درست کردن حالت به‌روز شده است. جزئیات اینکه مدل‌ها و حالت‌ها چه جوری نشون داده بشن، بستگی داره به محیط و فناوری طراحی عامل.

فرقی نمی‌کنه چه شکلی بخوایم وضعیت دنیا رو نشون بدیم، تقریباً هیچ‌وقت نمی‌شه دقیقاً فهمید در یک محیط نیمه قابل مشاهده الان دنیا چه‌جوریه. به‌جاش همون باکسی که روش نوشته «الان دنیا چه شکلیه» (شکل ۲٫۱۱) بهترین حدس عامل رو نشون می‌ده (گاهی هم چند تا حدس مختلف اگر عامل چند حالت رو در نظر بگیره). مثلاً یه تاکسی خودکار ممکنه نتونه اطراف اون کامیون بزرگی که جلوش ایستاده رو ببینه و فقط می‌تونه حدس بزنه چرا راه‌بند شده. پس ممکنه همیشه در مورد وضعیت فعلی یه مقدار نااطمینانی وجود داشته باشه، ولی باز هم عامل باید تصمیم بگیره.

**function** MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *state*, the agent's current conception of the world state
    *transition_model*, a description of how the next state depends on the current state and action
    *sensor_model*, a description of how the current world state is reflected in the agent's percepts
    *rules*, a set of condition–action rules
    *action*, the most recent action, initially none

  *state* ← UPDATE-STATE(*state*, *action*, *percept*, *transition_model*, *sensor_model*)
  *rule* ← RULE-MATCH(*state*, *rules*)
  *action* ← *rule*.ACTION
  **return** *action*

**Figure 2.12** A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

## عامل مبتنی بر هدف

دونستن وضعیت فعلی محیط همیشه برای تصمیم‌گیری کافی نیست. مثلاً سر یک تقاطع جاده‌ای، تاکسی می‌تونه بپیچه چپ، بپیچه راست یا مستقیم بره. تصمیم درست وابسته‌ست به جایی که تاکسی می‌خواد بره؛ یعنی علاوه بر شرح وضعیت فعلی، عامل به یه جور اطلاعات هدف هم نیاز داره که توضیح بده چه موقعیت‌هایی مطلوب‌اند — مثلاً رسیدن به یه مقصد مشخص. برنامه‌ی عامل می‌تونه این اطلاعات هدف رو با همون مدل دنیای قبلی (مدل انتقال و مدل حسگری) ترکیب کنه تا اقدام‌هایی رو انتخاب کنه که به هدف ختم می‌شن. شکل ۲٫۱۳ ساختار عامل مبتنی بر هدف رو نشون می‌ده.

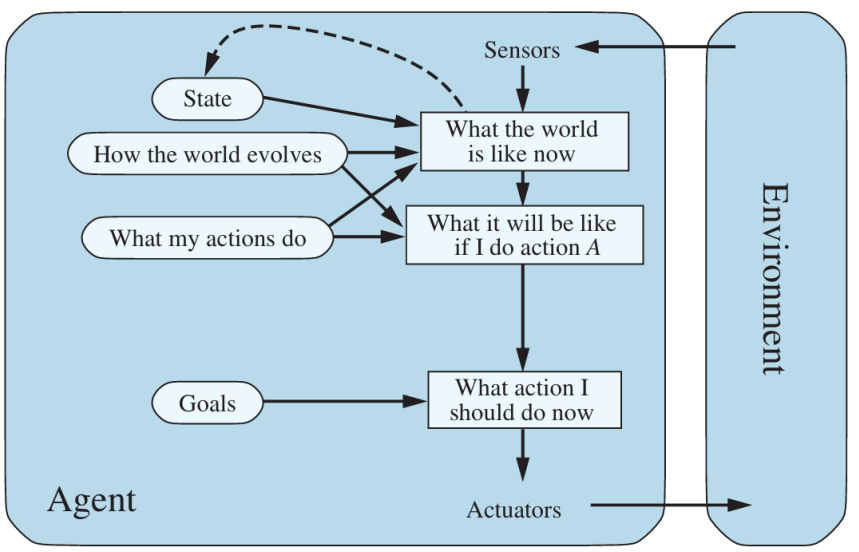



**Figure 2.13** A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

گاهی انتخاب عمل براساس هدف ساده‌ست – مثلاً وقتی با یه عمل به‌طور مستقیم می‌تونی به هدف برسی. ولی گاهی پیچیده‌ست – وقتی عامل باید فکر کنه یه سری پیچ و خم رو بکوبه تا راه رسیدن به هدف رو پیدا کنه. جست‌وجو (فصل‌های ۳، ۴ و ۶) و برنامه‌ریزی (فصل ۱۱) زیرشاخه‌های هوش مصنوعی هستن که به پیدا کردن توالی عمل‌ها برای رسیدن به هدف‌ها می‌پردازن.

توجه کن که این نوع تصمیم‌گیری با اون قاعده‌های شرط–عملی که قبلاً گفتیم، فرق بنیادینه، چون اینجا آینده رو در نظر می‌گیریم – «اگر این کار رو بکنم چی میشه؟» و «آیا اون منو به هدفم می‌رسونه؟» در طراحی عامل‌های واکنشی (رفلکس)، این اطلاعات عملاً وجود نداره؛ چون قاعده‌ها مستقیم از پرسپت میرن به عمل. عامل رفلکس وقتی چراغ ترمز می‌بینه ترمز می‌کنه، بی‌خیال این که چرا. اما عامل مبتنی بر هدف وقتی چراغ ترمز می‌بینه، به این خاطر ترمز می‌کنه که پیش‌بینی می‌کنه تنها کاریه که باعث می‌شه به هدفش – یعنی نخوردن به ماشین جلویی – برسه.

با این که به نظر می‌رسه عامل مبتنی بر هدف کم‌کاراتر باشه، اما انعطاف‌پذیری بیشتری داره، چون دانشی که تصمیم‌هاش رو پشتیبانی می‌کنه صریحاً نمایش داده شده و می‌تونیم تغییرش بدیم. مثلاً به راحتی می‌تونیم هدف رو عوض کنیم تا به یه مقصد دیگه بریم. ولی قاعده‌های عامل رفلکس برای گردش و مستقیم رفتن فقط برای همون یک مقصد تنظیم شدن و اگه بخوایم جای جدیدی بریم باید همه‌شون رو از اول بنویسیم.

### عامل مبتنی بر سودمندی

داشتن هدف به‌تنهایی برای تولید رفتار باکیفیت در بیشتر محیط‌ها کافی نیست. مثلاً خیلی از توالی‌های عمل ممکنه تاکسی رو به مقصد برسونن (پس هدف حاصل شده)، اما بعضی مسیرها سریع‌تر، ایمن‌تر، مطمئن‌تر یا ارزان‌تر از بقیه هستن. هدف‌ها فقط یه مرزبندی خام بین وضعیت‌های «خوشحال‌کننده» و «ناراحت‌کننده» ایجاد می‌کنن؛ در واقع می‌گن وضعیت خوبه یا بده، ولی نمی‌گن چقدر خوب یا چقدر بد.

برای مقایسه دقیق‌تر وضعیت‌های مختلف، عامل نیاز داره که بدونه هر وضعیت چقدر می‌تونه «خوشحالش» کنه. حالا چون «خوشحالی» واژه‌ی علمی و دقیقی نیست، اقتصاددان‌ها و دانشمندان کامپیوتر از واژه‌ی مطلوبیت یا Utility استفاده می‌کنن.

قبلاً دیدیم که معیار ارزیابی عملکرد (Performance Measure) به هر دنباله‌ای از وضعیت‌های محیط یه امتیاز اختصاص میده؛ پس به‌راحتی می‌تونه تشخیص بده کدوم مسیر رسیدن به مقصد تاکسی، بهتره. تابع مطلوبیت عامل یا همون Utility Function در واقع درونی‌سازی همین معیار بیرونی عملکرده. اگر تابع مطلوبیت داخلی و معیار ارزیابی بیرونی در هماهنگی باشن، عاملی که عمل‌هایی رو انتخاب کنه که مطلوبیتشون رو بیشینه کنه، طبق تعریف، عقلانی حساب میشه.

دوباره تأکید می‌کنیم که این تنها راه عقلانی بودن نیست — همون‌طور که قبلاً توی عامل دنیای جاروبرقی دیدیم، ممکنه عامل اصلاً ندونه تابع مطلوبیتش چیه، ولی همچنان رفتار عقلانی نشون بده. با این حال، مشابه عامل‌های مبتنی بر هدف، عامل‌های مبتنی بر مطلوبیت مزایای زیادی در انعطاف‌پذیری و یادگیری دارن.

علاوه بر این، در دو حالت خاص، هدف‌ها ناکافی هستن، ولی عامل مبتنی بر سودمندی همچنان می‌تونه تصمیم عقلانی بگیره:

۱. وقتی هدف‌ها در تضاد باشن و فقط بعضی از اون‌ها قابل دسترسی باشن (مثلاً سرعت و ایمنی)، تابع مطلوبیت مشخص می‌کنه چه معامله‌ای بین اون‌ها باید انجام بشه.
۲. وقتی چندین هدف وجود داره و هیچ‌کدوم با قطعیت قابل دستیابی نیست، تابع مطلوبیت راهی فراهم می‌کنه که احتمال موفقیت با اهمیت هدف‌ها سنجیده بشه.

در دنیای واقعی، جزئی‌نگری و عدم قطعیت، تقریباً همه‌جا حضور دارن؛ در نتیجه تصمیم‌گیری در شرایط عدم قطعیت هم اجتناب‌ناپذیره.

به‌صورت فنی، یک عامل عقلانی مبتنی بر مطلوبیت، عمل‌هایی رو انتخاب می‌کنه که «امید ریاضی مطلوبیت» رو بیشینه کنن — یعنی بر اساس احتمال‌ها و مطلوبیت‌های هر نتیجه، در مجموع کدوم عمل به طور متوسط بهترین نتیجه رو میده.

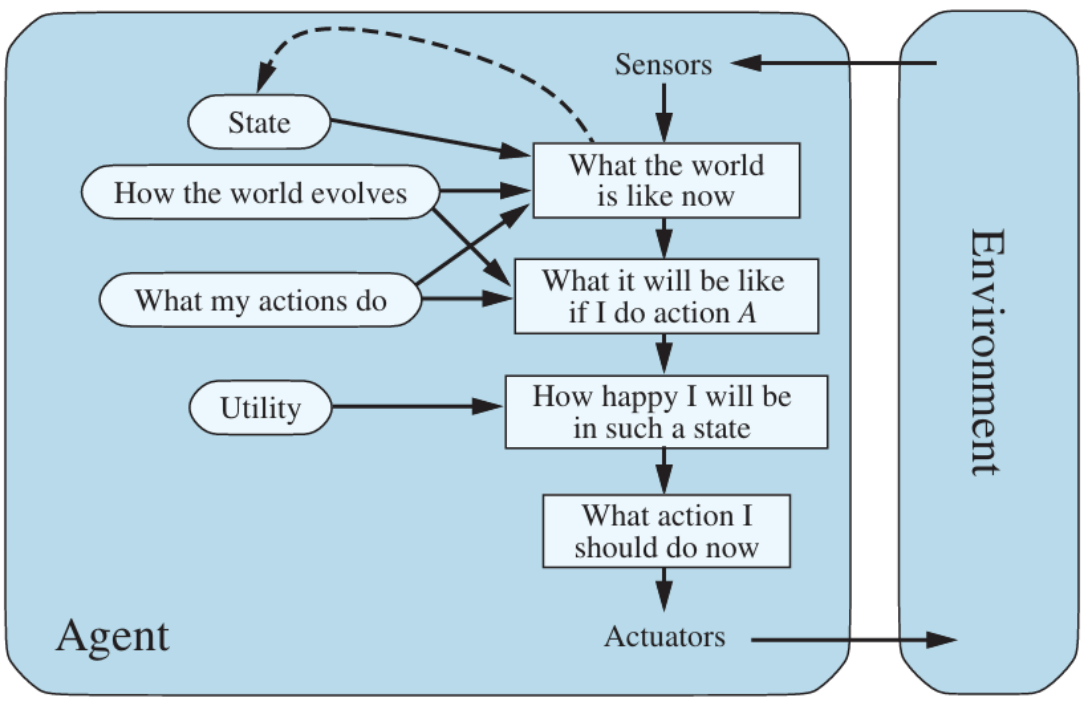



**Figure 2.14** A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

عامل‌هایی که یه تابع مطلوبیت صریح دارن، می‌تونن با الگوریتم‌های عمومی تصمیم‌گیری کنن؛ این الگوریتم‌ها وابسته به خود تابع نیستن، پس از این نظر طراحی‌شون خیلی انعطاف‌پذیره.
شاید این سؤال برات پیش بیاد که: «واقعاً همین‌قدر ساده‌ست؟ عامل‌هایی بسازیم که مطلوبیت انتظاری رو بیشینه کنن و تمام؟» پاسخ اینه که درسته که چنین عاملی رفتار هوشمندانه‌ای خواهد داشت، ولی نه، به این سادگی نیست.
یه عامل مبتنی بر مطلوبیت باید محیطش رو مدل کنه و وضعیت‌هاش رو پیگیری کنه؛ کاری که در عمل نیازمند تحقیقات سنگین در حوزه‌های ادراک، بازنمایی، استدلال و یادگیری بوده، و نتیجه‌ی این تحقیقات توی خیلی از فصل‌های این کتاب ارائه شده.
انتخاب عمل‌هایی که مطلوبیت انتظاری رو بیشینه کنن هم خودش یه مسئله‌ی سخته و به الگوریتم‌های هوشمندانه‌ای نیاز داره که موضوع چندین فصل دیگه‌ست.
حتی با وجود این الگوریتم‌ها هم، عقلانیت کامل در عمل معمولاً دست‌نیافتنیه، چون پیچیدگی محاسباتی همیشه مانع میشه همونطور که توی فصل ۱ اشاره شد.
نکته‌ی آخر این که: همه‌ی عامل‌های مبتنی بر مطلوبیت، لزوماً مبتنی بر مدل نیستن.

## عامل‌های یادگیرنده (Learning Agent)

تا اینجا در مورد برنامه‌های عامل (Agent) و روش‌های مختلف تصمیم‌گیریشون صحبت کردیم، ولی هنوز نگفتیم که این برنامه‌ها چطور ساخته میشن.
یه مقاله‌ی معروف از تورینگ تو سال ۱۹۵۰ هست که اونجا به این موضوع اشاره کرده. اون موقع تورینگ به این فکر می‌کرد که شاید بشه این ماشین‌های هوشمند رو با دست برنامه‌نویسی کرد، ولی وقتی حساب کرد چقدر زمان و انرژی می‌بره، به این نتیجه رسید که: «یه روش سریع‌تر و راحت‌تر باید وجود داشته باشه.»
راه‌حلی که پیشنهاد داد این بود که: بهتره ماشین‌هایی بسازیم که خودشون یاد بگیرن، و بعد بهشون آموزش بدیم.

الان هم تو خیلی از حوزه‌های هوش مصنوعی همین ایده استفاده میشه؛ یعنی به جای اینکه از اول همه چیز رو براشون بنویسیم، عامل‌هایی طراحی می‌کنیم که خودشون از تجربه یاد بگیرن.

هر مدلی از عامل — حالا چه مدل‌محور باشه، چه هدف‌محور یا سودمحور و... — می‌تونه به شکل «یادگیرنده» ساخته بشه یا نه؛ بستگی به نیاز داره.
یه مزیت بزرگ عامل‌های یادگیرنده اینه که می‌تونن تو محیط‌هایی که اولش هیچ شناختی ازش ندارن، شروع به کار کنن و کم‌کم با تجربه قوی‌تر بشن؛ قوی‌تر از چیزی که توی کد اولیه براشون در نظر گرفته شده بود.

یه عامل یادگیرنده از چهار بخش اصلی تشکیل میشه (تو شکل ۲٫۱۵ نشون داده شده). مهم‌ترین تفاوت اینه که:
«بخش یادگیرنده» مسئول بهتر کردن و پیشرفت عامل هست.
«بخش اجراکننده» (Performance Element) مسئول تصمیم‌گیری و انجام عمل‌هاست.

در واقع، اون چیزی که قبلاً بهش می‌گفتیم «عامل»، همین بخش اجراکننده بود. یعنی اطلاعات رو از محیط می‌گیره و تصمیم می‌گیره که چه کاری انجام بده.
حالا بخش یادگیرنده از طرف «ناظر» (Critic) بازخورد می‌گیره که داره چطور کار می‌کنه و بر اساس اون، بخش اجراکننده رو اصلاح و تقویت می‌کنه.

طراحی بخش یادگیرنده، خیلی بستگی به طراحی بخش اجراکننده داره. وقتی می‌خوایم یه عامل بسازیم که یاد بگیره، اول باید به این فکر کنیم که وقتی یاد گرفت، «چطوری قراره اون کار رو انجام بده؟»
بعد از این که طراحی بخش اجراکننده مشخص شد، تازه میشه یادگیری رو براش پیاده‌سازی کرد.

ناظر، به بخش یادگیرنده میگه که عملکرد عامل در مقایسه با یه استاندارد مشخص چقدر خوب بوده.
ناظر ضروریه، چون خود اطلاعاتی که از محیط میاد لزوماً نمی‌گه عامل موفق شده یا شکست خورده. مثلاً یه برنامه‌ی شطرنج ممکنه بفهمه «مات کرده» ولی باید بدون کمک ناظر متوجه بشه که این یه نتیجه‌ی خوبه.

یه نکته‌ی مهم اینه که اون «استاندارد عملکرد» باید ثابت باشه؛ یعنی عامل نباید بتونه اون رو تغییر بده که کار خودش رو خوب جلوه بده.

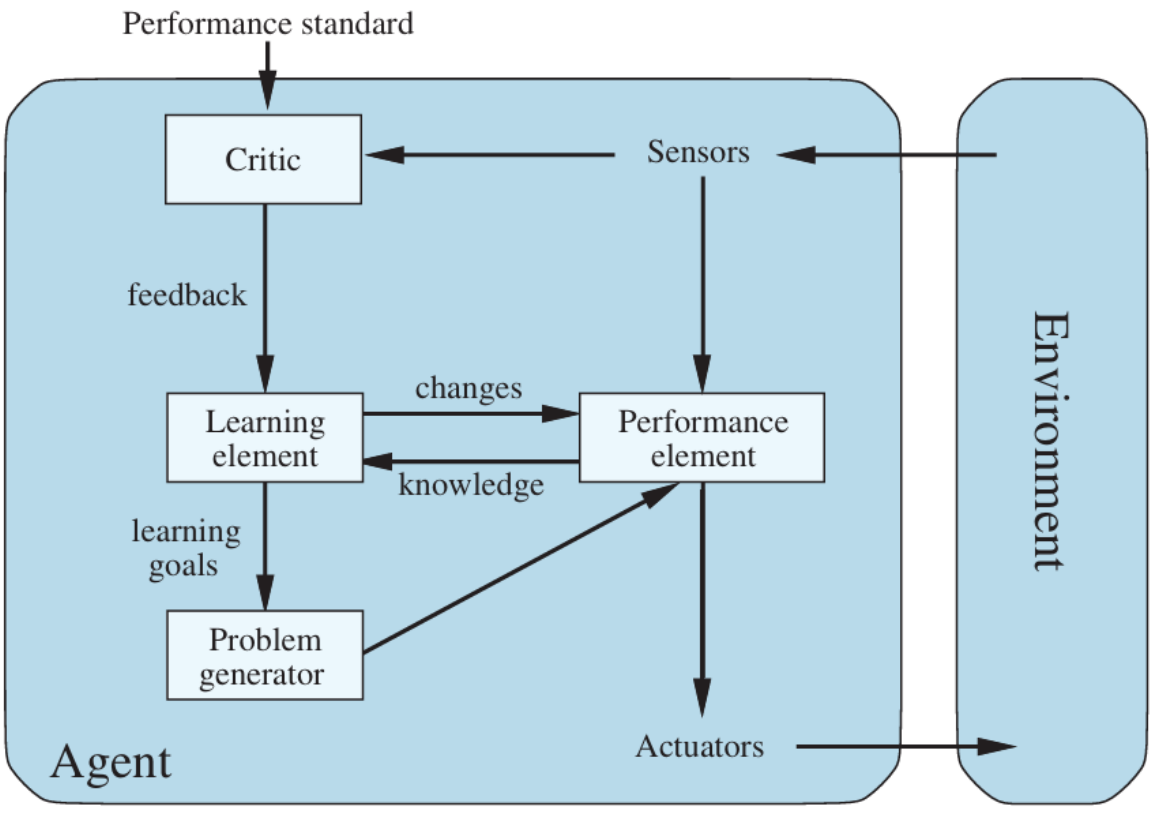



**Figure 2.15** A general learning agent. The "performance element" box represents what we have previously considered to be the whole agent program. Now, the "learning element" box gets to modify that program to improve its performance.

آخرین بخش از عامل یادگیرنده، «تولیدکننده‌ی مسئله» (Problem Generator) هست.
کار این بخش اینه که به عامل پیشنهاد بده چه کارهایی رو انجام بده که تجربه‌های جدید و مفید کسب کنه.
اگه همه چی دست بخش اجراکننده بود، همیشه همون کارهایی که بلده و مطمئنه خوبن رو تکرار می‌کرد، ولی گاهی باید کمی ریسک کرد و تجربه‌های جدید به دست آورد، حتی اگه کوتاه‌مدت کمی نتیجه‌ی بدی بده، در عوض ممکنه در بلندمدت به کشف راه‌های خیلی بهتری منجر بشه. همین اتفاقیه که تو آزمایش‌های علمی هم میفته. مثلاً گالیله که سنگ‌ها رو از بالای برج پیزا می‌انداخت، هدفش شکستن سنگ یا آسیب به عابران نبود؛ هدفش این بود که مغز خودش رو با یه نظریه‌ی بهتر درباره‌ی حرکت اشیاء، به‌روز کنه.

بخش یادگیرنده می‌تونه هر کدوم از بخش‌های دانش عامل رو تغییر بده. ساده‌ترین حالتش اینه که عامل مستقیم از دنباله‌ی اطلاعات محیط یاد بگیره. مثلاً با مشاهده‌ی حالت‌های پشت سر هم در محیط، می‌فهمه که «عمل من باعث چی شد؟» و «دنیا چطور به عمل من واکنش نشون داد؟».

یه مثال ساده: اگه یه تاکسی خودکار روی جاده‌ی خیس یه فشار خاص به ترمز وارد کنه، خیلی زود یاد می‌گیره که این فشار چقدر باعث کاهش سرعت میشه یا ممکنه سر بخوره.
تولیدکننده‌ی‌مسئله هم می‌تونه به عامل کمک کنه که بفهمه کجای مدلش نیاز به بهبود داره، مثلاً با امتحان کردن ترمز در شرایط مختلف آب و هوایی.

بهبود مدل عامل، حتی مستقل از نمره یا امتیاز بیرونی، معمولاً کار خوبیه؛ البته بعضی وقتا بهتره یه مدل ساده و کمی اشتباه داشته باشیم تا یه مدل خیلی دقیق ولی بسیار پیچیده و سخت برای محاسبه. اما وقتی می‌خوایم عامل یه رفتار خاص یا تابع سودمندی رو یاد بگیره، اطلاعات از «استاندارد عملکرد بیرونی» خیلی مهمه.
مثلاً فرض کن یه تاکسی خودکار، مسافراش رو تو مسیر اونقدر تکون بده که کسی بهش انعام نده؛ عامل باید از ناظر یاد بگیره که از دست دادن انعام، نشونه‌ی بدیه و این یعنی «رانندگی خشن» براش سودی نداره.
در واقع این استاندارد کمک می‌کنه که عامل بتونه بخشی از اطلاعات ورودی رو به شکل جایزه یا تنبیه تشخیص بده و رفتار خودش رو اصلاح کنه.
تو موجودات زنده هم استانداردهای درونی مثل درد و گرسنگی، نقش همین ناظر بیرونی رو دارن.

یه مثال دیگه: فرض کن اون تاکسی نمی‌دونه آدما از صدای بلند خوششون نمیاد، پس تصمیم می‌گیره دائماً بوق بزنه که مطمئن بشه عابرا متوجه حضورش میشن.
اما واکنش آدم‌ها – مثل گرفتن گوش، فحش دادن، یا حتی قطع کردن سیم بوق – کم‌کم به عامل این سیگنال رو میده که «ایده‌ی بوق زدن مداوم ایده‌ی خوبی نبوده!» و اینطوری تابع سود خودش رو اصلاح می‌کنه.
در مجموع، عامل‌ها بخش‌های مختلفی دارن که به شکل‌های مختلف تو برنامه پیاده میشن و در نگاه اول شاید روش‌های یادگیری متنوع به نظر برسن؛ ولی در واقع، همه‌ی این روش‌ها یه هسته‌ی مشترک دارن:
یادگیری یعنی این‌که عامل، بخش‌های مختلف خودش رو با استفاده از بازخوردهایی که از محیط و استاندارد دریافت می‌کنه، به‌روزرسانی کنه و در نتیجه عملکرد کلی خودش رو بهتر و بهتر کنه.