Uczenie się

* Opisz cztery główne typy uczenia się na podstawie doświadczenia (nadzorowane, nienadzorowane, częściowo nadzorowane i uczenie się ze wzmocnieniem) oraz związane z nimi techniki.

* Opisz następujące techniki uczenia się: indukcja, maszyny wektorów nośnych i propagacja wsteczna.

* Jeśli otrzymałeś sztuczną sieć neuronową ze sprzężeniem do przodu z wejściami, ósemkami i funkcją aktywacji, oblicz wyjście jednostki.

* Jeśli otrzymasz macierz nagród i losową eksplorację, napisz wzór na Q-learning, a następnie stwórz politykę.

* Zdefiniuj robotykę ewolucyjną i wyjaśnij, dlaczego powiązane metody nie są uważane za uczenie się.

* Zdefiniuj procesy w algorytmach genetycznych: selektywne rozmnażanie, losowe mutacje i rekombinację genetyczną.

* Jeśli otrzymamy zestaw par genomów i punkt przecięcia, wyprodukuj następne pokolenie w populacji.

* Zdefiniuj krytykę, problem z nowym terminem, nadmierne dopasowanie, problem z przypisaniem punktów kredytowych i naukę online.

Przegląd

Poprzednie części przekazywały wyzwania związane z organizowaniem i wdrażaniem prymitywów SENSE, PLAN i ACT dla lokomocji, nawigacji i manipulacji, rozpoznawania obiektów i deliberacji. Złożoność i trudność reakcji i rozważań wydają się sprzeczne z tym, jak naturalne są te zdolności dla danej osoby, co prowadzi do krzyku: czy roboty nie mogą po prostu nauczyć się robić rzeczy? Rzeczywiście roboty mogą uczyć się robić rzeczy, a także rozpoznawać przedmioty i sytuacje, ale niestety uczenie się jest kolejną ilustracją rubryki, że jeśli jest to łatwe dla człowieka, to trudne dla komputera. Chęć obdarowania robotów uczeniem się prowadzi do dwóch pytań: Czym jest uczenie się? oraz Jakie są rodzaje uczenia się? Chociaż można podejrzewać, że istnieją różne rodzaje uczenia się, sieci neuronowe, zwłaszcza uczenie głębokie, zajmują ważne miejsce w rozrywce i mediach informacyjnych. Dlatego naturalne jest pytanie: dlaczego roboty nie używają cały czas sieci neuronowych? Są jak mózg, prawda? Z bardziej praktycznego punktu widzenia projektowego, pytanie: dokąd prowadzi nauka w naszej architekturze operacyjnej? trzeba odpowiedzieć. W tej części postaram się odpowiedzieć na wszystkie te pytania. Część zaczyna się od zdefiniowania, czym jest uczenie się i opisania ogólnych rodzajów uczenia się. Typy uczenia się generalnie dzielą się na dwie kategorie: agent uczy się przez przykład lub doświadczenie lub agent uczy się przez rozumowanie lub analogię. Jedną z niespodzianek jest to, że algorytmy genetyczne i związana z nimi technika zwana symulowanym wyżarzaniem, która nie zostanie tutaj omówiona, nie są technicznie uczącymi się algorytmami, chociaż są związane z uczeniem się. Następnie przedstawiono bardziej szczegółowo trzy klasy algorytmów uczenia się przez przykład lub doświadczenie, poczynając od uczenia nienadzorowanego i popularnych sztucznych sieci neuronowych. Druga klasa algorytmów dotyczy uczenia nadzorowanego, które wykorzystuje inwestycje w eksplorację danych ukierunkowaną na naukę rozpoznawania wzorców. Dwie główne techniki uczenia nadzorowanego to maszyny indukcyjne i wektory nośne. Trzecią klasą algorytmów jest Q-learning, który jest prawdopodobnie najpopularniejszym rodzajem uczenia się w robotyce, ponieważ skupia się na nauce optymalnych działań, czyli tego, co dalej. Po zbadaniu typów uczenia się, spróbujemy połączyć to wszystko razem, dając przegląd, gdzie uczenie się pasuje do architektury robota. Następnie rozdział opisuje algorytmy

genetyczne i omawia ich związek z uczeniem się. Część kończy się spostrzeżeniami na temat uczenia się i sztucznej inteligencji

Uczenie się

Uczenie się definiuje się jako proces, w którym agent poprawia swoją wydajność w przyszłych zadaniach po dokonaniu obserwacji świata. Uczenie się pozwala agentowi dostosowywać się do zmian w świecie, na przykład zmian terenu lub zadań, lub do zmian samych w sobie, takich jak konieczność wydatkowania większej ilości energii, aby zrekompensować zużycie silników, w porównaniu do zmian długoterminowych, które być uchwycone przez procesy ewolucyjne, na przykład zmiany klimatyczne. Istnieje wiele technik uczenia maszynowego i w każdym przypadku, według Russella i Norviga, projektant musi wiedzieć:

* Komponent, który ma zostać ulepszony. Uczenie maszynowe może służyć do ulepszania funkcji w ramach dowolnego z pięciu podsystemów: planowania, kartografa, nawigacji, schematu motorycznego i percepcji.

* Wcześniejsza wiedza dostępna dla agenta. Czy robot budzi się po prostu jako tabula rasa, bez wcześniejszej wiedzy, czy też ma już sposób na wykonanie tej funkcji, ale musi ją poprawić?

* Reprezentacja danych i komponentu, którego należy się nauczyć. Czy dane są obrazami RGB? Lista wartości? Jak wiadomo z badania algorytmów, dobra reprezentacja może znacznie uprościć i przyspieszyć wykonanie algorytmicznej części rozwiązania.

* Dostępne informacje zwrotne, z których można się uczyć. Aby ulepszyć, robot potrzebuje pewnego rodzaju funkcji błędu sprzężenia zwrotnego. Robotowi można powiedzieć, czy działa dobrze, czy też może mieć odczyty z czujników, które pozwalają mu wewnętrznie mierzyć poprawę.

Oto cztery przykłady wykorzystania uczenia maszynowego w robotyce i tego, co projektant musiał wiedzieć w zakresie komponentów, wcześniejszej wiedzy, reprezentacji i informacji zwrotnych. Pierwszym przykładem jest jedno z oryginalnych zastosowań uczenia maszynowego w robotyce, które powstało pod koniec lat 80., kiedy robotnicy zaczęli uczyć się praw sterowania w celu kontrolowania trajektorii manipulatora lub innych powtarzalnych ruchów robota. Wiele przemysłowych zastosowań manipulacyjnych wymaga od robota wykonywania powtarzalnych czynności, na przykład trzymania tacki z częściami idealnie płasko podczas ich przenoszenia. W tych zastosowaniach znana jest pożądana trajektoria. Wyzwanie polega na określeniu prawa sterowania dla koordynacji momentu obrotowego i czasu poszczególnych przegubów robota o wielu stopniach swobody, zaprojektowanego do poruszania się po trajektorii i dostosowywania się do hałasu wynikającego ze zużycia siłowników lub czynników środowiskowych w miarę upływu czasu. Złożoność ciągłych interakcji między przegubami i momentami obrotowymi jest nieliniowa i bardzo trudna do matematycznego przedstawienia i rozwiązania. Na szczęście robota można obserwować, dzięki czemu można zmierzyć różnicę między trajektorią pożądaną, często nazywaną trajektorią referencyjną, a trajektorią rzeczywistą. Robot może wtedy uczyć się, poprzez powtarzające się lub iteracyjne próby, jak kontrolować siebie. W tym przykładzie elementem do poprawy jest prawo sterowania ruchem, wcześniejsza wiedza to trajektoria referencyjna, a sprzężenie zwrotne to błąd pomiaru między trajektorią rzeczywistą a referencyjną. Określone iteracyjne algorytmy uczenia będą miały różne reprezentacje danych, takie jak macierz korzyści uczenia się do rejestrowania uczenia się w iteracjach. Drugim przykładem jest uczenie się miejsc. Stałym problemem w robotyce jest przekształcanie zaszumionych danych sensorycznych na dane symboliczne, zwłaszcza w nawigacji zrobotyzowanej, gdzie istnieje duża różnica między afordancją bramy („Mogę zmienić kierunek”) a rozpoznawaniem miejsca („Jestem przed drzwiami do siedziby głównej”). Wiele technik uczenia miejsc dzieli świat na plamy lub wektory cech, takich jak

ramki widokowe. Robot wykorzystuje statystyki bayesowskie do powiązania ramki widoku, która nie będzie dokładnie odpowiadać temu, co widzi robot, ze współrzędnymi lokalizacji, co również będzie obarczone błędem. W tym przykładzie komponentem do ulepszenia jest mapa świata, wcześniejsza wiedza składa się z typów funkcji, które powinny być użyte, reprezentacją jest ramka widokowa lub wektor, a informacja zwrotna pochodzi z zestawu prób, w których robot podchodzi do tego miejsca pod różnymi kątami. Trzecim przykładem jest uczenie się terenu. Dla robotów często niepraktyczne jest uczenie się, jak poruszać się w trudnym terenie za pomocą symulacji komputerowej lub kontrolowanych prób. Trudny teren, na którym mogą występować zmiany wysokości, błoto lub skały oraz różne rodzaje roślinności, jest niezwykle trudny do wymodelowania. Jednym z rozwiązań jest ciągła aktualizacja modelu terenu za pomocą nauki online. Robot wykorzystuje swoje czujniki do przewidywania terenu przed nim, a następnie wykorzystuje informacje zwrotne, takie jak moment obrotowy (np. wskazujący na błoto), wibracje (np. wyboisty), IMU (np. wspinam się) itd., aby potwierdzić, czy projekcja była poprawna. W tym przykładzie komponentem, który należy poprawić, jest modelowanie terenu, wcześniejsza wiedza dotyczy tego, w jaki sposób czujniki określają cechy terenu, reprezentacją danych jest przesuwana mapa rzutowanego terenu, a informacje zwrotne dotyczą faktycznego wykorzystania. Czwartym przykładem jest uczenie się alokacji zachowań. Modułowość zachowań może wspierać uczenie się wyboru zachowań metodą prób i błędów w nowej sytuacji, na przykład jak znaleźć obiekt i przywrócić go. Robot może nauczyć się, kiedy używać każdego ze swoich zachowań, na przykład wyszukiwania, przechodzenia do celu, chwytania i unikania lub wykonywania podstawowych operacji we właściwej kolejności lub z odpowiednimi korzyściami. wracając do listy elementów projektu, komponentem do poprawy dla tego przykładu jest wybór akcji, dotychczasowa wiedza to zbiór dostępnych zachowań, reprezentacją jest lista wypróbowanych kombinacji i sekwencji zachowań oraz informacje zwrotne jest to, czy robot osiąga cel.

Rodzaje uczenia się przez przykład

Istnieje wiele form uczenia się, w tym uczenie się na przykładzie, uczenie się na doświadczeniu, uczenie się na podstawie demonstracji (zwane również uczeniem się naśladownictwa), uczenie się przez rozumowanie, uczenie się na podstawie przypadków lub na podstawie wyjaśnień oraz uczenie się przez analogię. Uczenie się na przykładzie, w którym agentowi przedstawia się przykłady koncepcji lub działania, jest najczęstsze i jest z powodzeniem stosowane w przypadku robotów. Tradycyjnym sposobem kategoryzacji uczenia się według przykładowych algorytmów jest styl informacji zwrotnej. Istnieją cztery kategorie informacji zwrotnych: nadzorowane, nienadzorowane, częściowo nadzorowane i uczenie się ze wzmocnieniem. Z tych czterech pierwsze dwa są powszechnie używane w robotyce do nauki postrzegania świata. Częściowo nadzorowane uczenie się jest również często stosowane do percepcji, ale rzadziej niż nadzorowane lub nienadzorowane uczenie się robotyki i nie będzie dalej omawiane. Uczenie się ze wzmacnianiem jest prawdopodobnie najczęstszą formą uczenia się robotów, ponieważ można je zastosować do uczenia się wieloetapowych sekwencji działań. Są to tak zwane zasady.

NADZOROWANA NAUKA

Uczenie się nadzorowane opiera się na wyraźnych i bezpośrednich informacjach zwrotnych. Robot jest szkolony w rozpoznawaniu koncepcji - obiektu, miejsca, stanu lub skojarzenia. Jest przedstawiony z przykładami, być może obrazem, które są oznaczone poprawnym wyjściem dla każdego wejścia. Robot bierze ten zestaw par (wejście, wyjście) i uczy się, jakie atrybuty na wejściu prowadzą do klasyfikacji wyjścia. Po przedstawieniu nowych danych wejściowych robot rozpoznaje, czy koncepcja jest obecna, czy nie. Na przykład szkolenie humanoidalnego robota do wykonywania zadania poprzez obserwowanie, jakich ruchów używa dana osoba. Ruchy osoby są oznakowanym wkładem do koncepcji, która jest zadaniem lub umiejętnością.

NAUKA NIENADZOROWANA

Nienadzorowane uczenie się wykorzystuje pośrednią informację zwrotną. Robot otrzymuje zestaw atrybutów dla zestawu danych wejściowych i uczy się rozpoznawać, w jaki sposób atrybuty się grupują, nie wiedząc, czego ma się uczyć. Robot bada zbiór danych wejściowych i odkrywa obiekty, miejsca, stany lub skojarzenia ukryte w danych wejściowych, badając matematyczne znaczenie występowania ich atrybutów. Nienadzorowane uczenie się jest często związane z eksploracją danych i klastrowaniem. Przykładem jest Rogue, robot, który nauczył się unikać obierania tras przez lobby w porach dnia, kiedy prawdopodobnie było tam tłoczno, i zamiast tego wybrał szybszą, choć dłuższą ścieżkę.

NAUKA PÓŁNADZOROWANA

Częściowo nadzorowane uczenie wykorzystuje mały zestaw oznaczonych par (wejście, wyjście) zgodnie z uczeniem nadzorowanym, aby pomóc kierować nienadzorowanym uczeniem się z nieznanych danych wejściowych. Na przykład autonomiczny samochód może otrzymać kilka zdjęć osoby na rowerze i kilka śladów, a następnie nauczyć się śledzić rowerzystę i przewidywać, gdzie rowerzysta może wskoczyć i wyskoczyć z ruchu.

NAUKA PRZEZ WZMACNIANIE

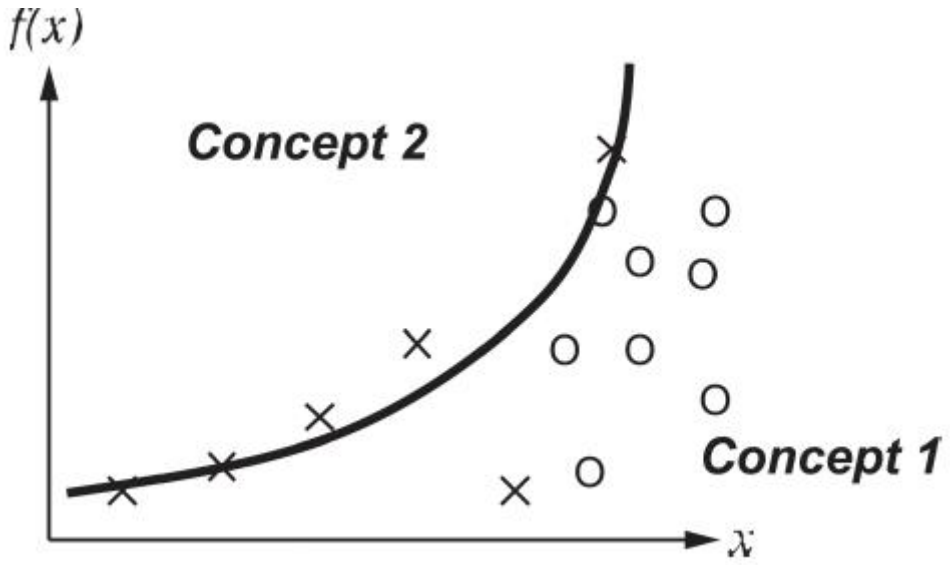
Uczenie się ze wzmacnianiem wykorzystuje informacje zwrotne z prób i błędów do uczenia się sekwencji działań. Robot otrzymuje cel, próbuje możliwych działań, aby osiągnąć cel, a następnie szacuje wartość każdego działania w sekwencji dochodzenia do celu. Uczenie ze wzmocnieniem jest podobne do uczenia nienadzorowanego, ponieważ atrybuty, w tym przypadku optymalny zestaw działań, są ukryte. W przeciwieństwie do uczenia nienadzorowanego, cel, czyli wynik oznaczony etykietą, jest znany. W przeciwieństwie do uczenia nadzorowanego, robot generuje własne dane wejściowe poprzez eksplorację. Jednym z przykładów jest Brachiator, robot, który nauczył się kołysać po szeregu poziomych szczebli jak gibon.128 Brachiator wykorzystał naukę wzmacniania, aby nauczyć się koordynować działania czuciowo-motoryczne, tak aby nogi generowały pęd kołysania, a chwytaki sięgały i chwytały następny uchwyt.

Wspólne nadzorowane algorytmy uczenia się

W uczeniu nadzorowanym agent obserwuje pewne pary przykładowe (wejście, wyjście) podczas fazy uczenia i uczy się funkcji h, która mapuje dane wejściowe na dane wyjściowe. Wyjście jest oznaczone jako dodatnie lub ujemne. Główne techniki to: indukcja, maszyny wektorów nośnych (SVM) i drzewa decyzyjne. Maszyny wektorów indukcyjnych i nośnych są bardzo podobne, a maszyny wektorów nośnych dominują teraz w nadzorowanym uczeniu się w robotyce.

WPROWADZENIE

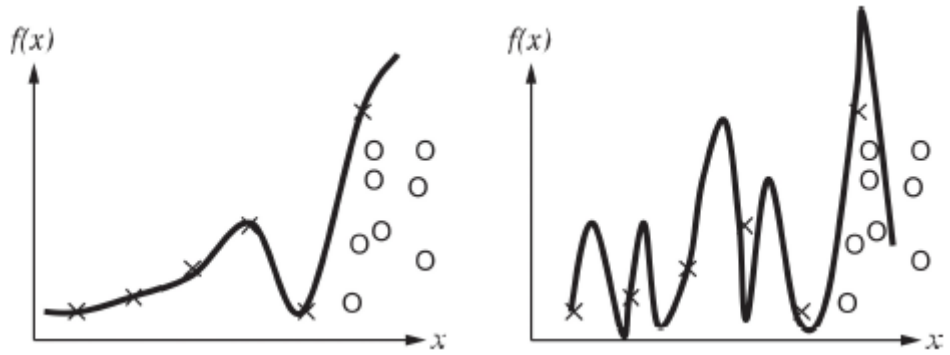
Indukcja wyprowadza pojęcie z oznaczonych par (wejście, wyjście). Podczas szkolenia niektóre z par będą przykładami pojęcia („Jest”) lub pozytywnymi przypadkami. Ale niektóre pary nie będą miały pojęcia lub będą negatywnym przykładem („Nie jest”). Algorytm uczący rysuje linię, która najlepiej dzieli dane wejściowe na „Jest” i „Nie jest”, jak pokazano na rysunku



Linia nazywa się hipotezą, h, chociaż technicznie jest to funkcja hipotezy. W praktyce dane wejściowe mogą mieć wiele atrybutów, dzięki czemu oś x jest naprawdę wielowymiarowa. Pojęcie obejmuje więcej niż „jest" lub „nie jest" i odzwierciedla kategorie większej koncepcji, na przykład owoce, które mogą zawierać „jabłka", „pomarańcze" i „winogrona". Zestaw cech x może być kolorem, kształtem i rozmiarem klasyfikowanych danych wejściowych.

PRZEPRACOWANIE

Głównym problemem we wprowadzaniu i uczeniu nadzorowanym w ogóle jest nadmierne dopasowanie. Możliwe, że algorytm generuje h, które doskonale klasyfikuje dane wejściowe podczas treningu, ale popełnia poważne błędy, gdy otrzymuje rzeczywiste dane wejściowe. Rysunek

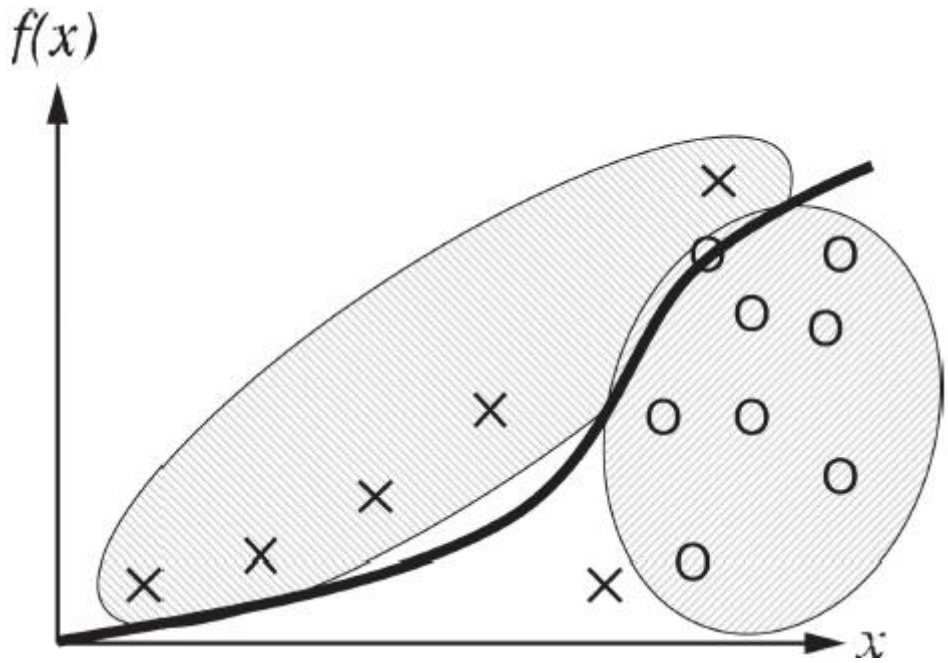


pokazuje przykład dwóch różnych funkcji h, które są zgodne z danymi uczącymi, ale prawdopodobnie nie będą dobrze klasyfikować nowych danych po ich uruchomieniu.

WSPARCIE MASZYN WEKTOROWYCH

Maszyny wektorów nośnych to indukcyjne mechanizmy uczenia się, które starają się uniknąć nadmiernego dopasowania, wykorzystując statystyczne właściwości danych. Klasyczne techniki indukcji zwykle starają się zminimalizować błąd między poszczególnymi wyjściami i h. Jest to koncepcyjnie podobne do dopasowania krzywej metodą najmniejszych kwadratów. Indukcja może

wytworzyć krzywą, która idealnie lub prawie idealnie dzieli wyjście na właściwą koncepcję. Jednak h może być przesadnie dopasowany. Maszyny wektorów nośnych starają się unikać nadmiernego dopasowania. Nie biorą pod uwagę poszczególnych wyników, ale patrzą na grupy wyników, które tworzą rozkład statystyczny, jak pokazano na rysunku



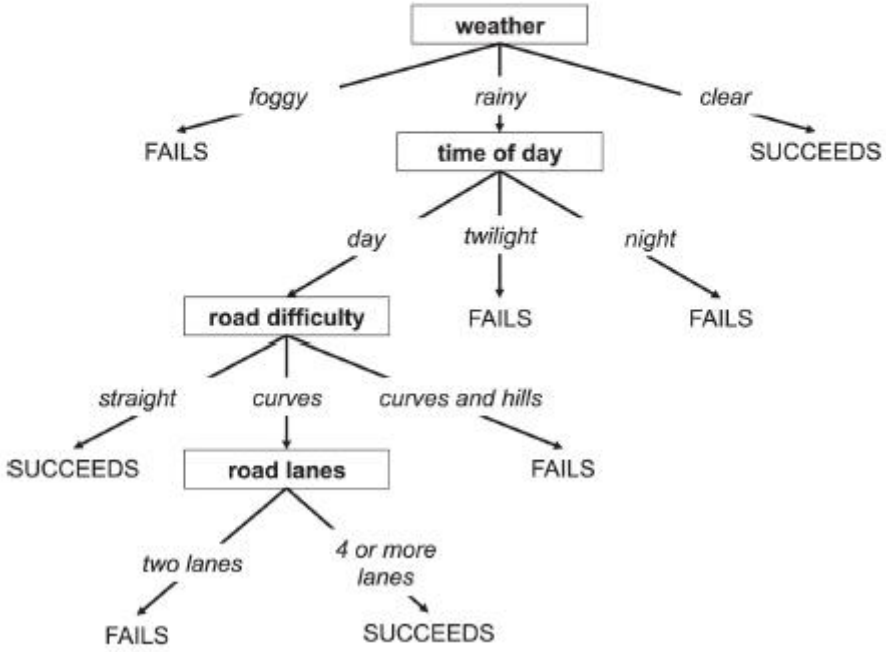
Krzywa jest dopasowywana do podziału tych grup, zmniejszając w ten sposób prawdopodobieństwo, że krzywa przepełni dane. Istnieje możliwość, że kilka wartości odstających zostanie błędnie sklasyfikowanych i spadnie po niewłaściwej stronie krzywej, ale taka możliwość istniała również w przypadku metod indukcyjnych.

DRZEWA DECYZYJNE

Drzewa decyzyjne różnią się od maszyn wektorów nośnych tym, że drzewa decyzyjne uczą się, który zestaw pomiarów jest cenny i w jakiej kolejności uzyskiwać pomiary. Algorytmy drzew decyzyjnych wykorzystują teorię informacji, aby nauczyć się najlepszego wyboru lub działania. Na przykład rozważ przewidywanie, kiedy autonomiczny system drogowy ulegnie awarii podczas jazdy, aby system mógł ostrzec kierowcę o możliwości przejęcia kontroli. Załóżmy, że robot miał dostęp do tabeli pomiarów dotyczących atrybutów terenowych i pogodowych dla różnych prób oraz tego, czy autonomiczny system śledzenia drogi zawiódł, a człowiek musiał przejąć kontrolę. Zobacz rysunek a. Na podstawie tych danych robot mógłby nauczyć się przewidywać, czy zawiedzie dla określonej kombinacji atrybutów. Ale w tym samym czasie może również dowiedzieć się, w jakiej kolejności przeszukiwać dane. Na przykład robot może się nauczyć, że jeśli jest zamglony, autonomiczne sterowanie zawsze zawodzi. W ten sposób robot może zaoszczędzić czas, pobierając najpierw raport pogodowy, aby sprawdzić, czy jest mglisty. Jeśli nie jest zamglony, robot może przyjrzeć się kolejnej najbardziej pouczającej właściwości. Wynikiem tego procesu jest drzewo decyzyjne, takie jak na rysunku . ID3 to klasyczny algorytm drzewa decyzyjnego.

time of day	weather	road lanes	road difficulty	result
day	clear	4 or more lanes	straight	SUCCEEDS
rainy	twilight	4 or more lanes	curves	FAILS
day	clear	two lanes	curves and hills	SUCCEEDS
foggy	twilight	two lanes	curves and hills	FAILS
rainy	day	4 or more lanes	curves	SUCCEEDS
foggy	night	4 or more lanes	curves	FAILS
...	...	...	...	...

a.



Typowe algorytmy uczenia nienadzorowanego

Nienadzorowane uczenie się uczy się wzorców ukrytych w danych wejściowych, nawet jeśli nie jest dostarczana żadna wyraźna informacja zwrotna. Jest to popularny rodzaj uczenia się i jest używany przez aplikacje internetowe do eksploracji danych o klientach w celu odkrywania trendów lub preferencji. W robotyce robot może uczyć się różnych rodzajów terenu, przechodząc przez teren i kojarząc z nim określoną wibrację lub określając, które czujniki najlepiej sprawdzają się w różnych środowiskach. Dwa główne algorytmy uczenia nienadzorowanego to algorytmy klastrowania lub rozpoznawania wzorców, z których prawdopodobnie najlepiej poznanym jest bayesowskie uczenie statystyczne, oraz sztuczne sieci neuronowe (ANN), które są podstawą technik uczenia głębokiego. SSN może być również nadzorowana. W tej sekcji omówiono sprzężeniową sieć ANN z progresją wsteczną na wysokim poziomie w celu zapoznania czytelnika z ogólną ideą i terminologią. Czytelnik będzie potrzebował bardziej szczegółowych badań, aby zastosować te koncepcje.

Grupowanie

Grupowanie próbuje nauczyć się zestawu pojęć lub działań, zauważając podobieństwa w zestawie wejściowym. Na przykład algorytm uczący się może zauważyć, że pomarańcze na drzewie są okrągłe, a liście mają inny kształt. Algorytm wyodrębniłby właściwości statystyczne, które definiują Pozycję 1 (pomarańcze) i Pozycję 2 (liście). Następnie, jeśli zostanie przedstawiony z nowym obrazem, program może obliczyć, jak blisko jest nowy obraz do pozycji 1 i 2 i oznaczyć nowy obraz najbliższym dopasowaniem. Wyzwanie w klastrowaniu polega na tym, że atrybuty, które mają zostać zgrupowane,

muszą być określone. Ponadto algorytm, taki jak grupowanie k-średnich, wymaga określenia liczby klastrów w celu wykonania; oznacza to, że projektant musi wiedzieć, co jest „ukryte" lub eksperymentalnie określić najlepszą liczbę przegród metodą prób i błędów.

SZTUCZNE SIECI NEURONOWE (ANN)

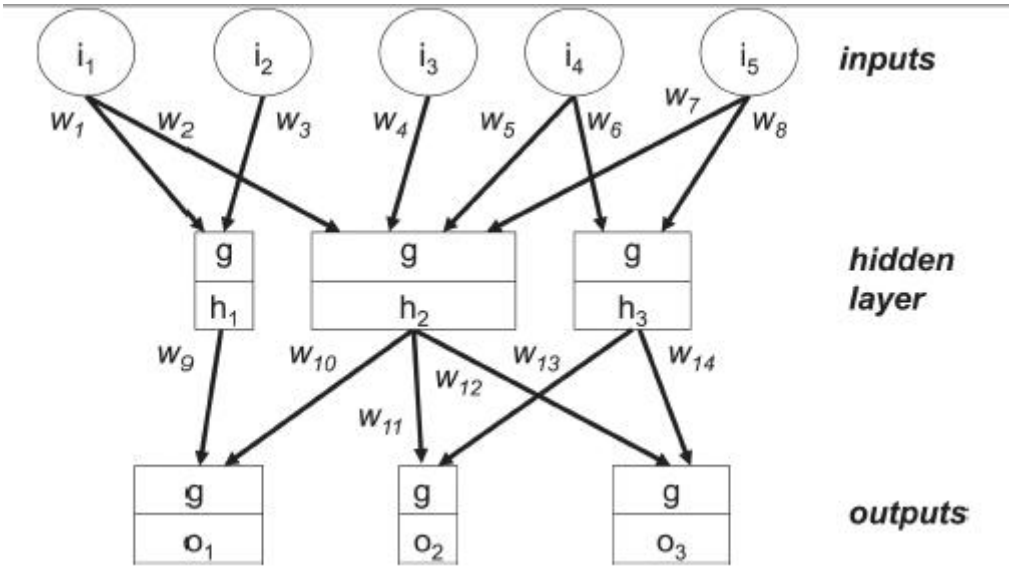
Sztuczne sieci neuronowe (ANN) były przedmiotem badań w ramach sztucznej inteligencji od momentu powstania tej dziedziny. Pomysł z neuronauki, zwłaszcza z prac Davida Rumelharta,180 polega na tym, że siła wejść powoduje odpalanie neuronów, tworząc w ten sposób bodziec dla innych neuronów, które następnie odpalają z różną siłą i tak dalej poprzez sieć połączeń, ostatecznie zwiększając siła obiektu wyjściowego lub koncepcji. Sieć połączeń może przechwytywać relacje nieliniowe, które są trudne do uzyskania na podstawie klastrów lub maszyn wektorów pomocniczych. SSN są zwykle używane w sytuacjach, w których istnieje duży zestaw atrybutów, ale nie ma modelu tego, w jaki sposób atrybuty przyczyniają się do rozpoznawania obiektu lub koncepcji, oraz gdy istnieje duży zestaw przykładów szkoleniowych umożliwiających zbieżność procesu uczenia się w sieci. W tej sekcji najpierw opisano sztuczne sieci neuronowe w kategoriach topologii sieci, a następnie wykorzystamy tę topologię do wyjaśnienia, jak działa najpopularniejsza metoda uczenia się, czyli propagacja wsteczna. Każdy węzeł lub jednostka w sieci ma zestaw ważonych danych wejściowych, które są przekształcane w dane wyjściowe. Odczyty lub wartości dla wystąpienia szkoleniowego są propagowane przez sieć w celu wygenerowania wartości dla danych wyjściowych sieci. Jednostką o najwyższej wartości jest wyjście sieciowe. Sieć uczy się na podstawie instancji szkoleniowej, porównując obliczony wynik sieci z idealnym wynikiem. Różnica polega na błędzie, który jest następnie wykorzystywany do dostosowania wag. Projektant określa początkową topologię sieci, co oznacza, że sieć nie nauczy się spontanicznie zupełnie nowego terminu. W efekcie nowy termin zostałby utworzony, gdy żadna z opcji wyjściowych określonych w topologii nie jest odpowiednia, ponieważ istnieje nowy obiekt lub koncepcja, które nie zostały pierwotnie zaprojektowane w systemie. Niestety, SNN są generalnie ograniczone przez zaprojektowaną topologię, a zatem z definicji nie mogą nauczyć się niczego, co zostało przedstawione jako wyjście potencjalne.

PROBLEM Z NOWYM TERMINEM

Wyzwanie polegające na nauce nowego terminu i odkryciu trudnej do znalezienia korelacji nazywa się problemem nowego terminu. Prowadzone są znaczące badania nad generowaniem nowych terminów, zarówno za pomocą SSN, jak i innych rodzajów uczenia się, ale obecnie nie ma praktycznego mechanizmu uczenia się dla robotyki.>

Topologia sieci

Rysunek przedstawia sieć z pięcioma wejściami sieci i lub interesującymi atrybutami, które ostatecznie doprowadzą do uzyskania siły lub pewności w trzech wyjściach sieci o.



Wejścia sieciowe są zwykle wartościami wykrywanymi, takimi jak procent zielonych pikseli w obiekcie, odnoga w górę lub dodatnia zmiana w enkoderze. Każdy węzeł w sieci reprezentuje neuron zwany jednostką. Sieć składa się z czterech komponentów, jednostek, wagi bodźca, jednostek ukrytych i funkcji aktywacji, które pozwalają jej uczyć się złożonych relacji. Jednostki są połączone krawędziami odzwierciedlającymi ważoną siłę bodźca do tej jednostki. Na przykład, bodziec dla procentu zielonych pikseli w obiekcie bez zieleni powinien wynosić 0, ale wartość bodźca dla całkowicie zielonego obiektu powinna być bardzo zbliżona do 1,0. Bodziec może zasilać wiele jednostek. Na przykład na rysunku bodźce z i_1, i_4 i i_5 trafiają do dwóch różnych jednostek. Jednak wartość lub ważność bodźców może być różna dla każdej jednostki odbierającej lub stymulowanej. Dlatego każdy bodziec dla jednostki ma powiązaną wagę, w_n dla tej jednostki. Wagi są nauczone. Na rysunku dane wejściowe do jednostki h_2 to inputh2 = $i_1 * w_2 + i_3 * w_4 + i_5 * w_7$.

FUNKCJA AKTYWACJI

PERCEPTRON

FUNKCJA LOGISTYCZNA

Urządzenie pobiera ważone dane wejściowe i stosuje funkcję aktywacji g_u, aby połączyć ważone bodźce i wytworzyć dane wyjściowe. Wyjście jest albo mnożone przez w łączące go z następną jednostką, albo jest mocą końcowego wyjścia sieciowego oj. Funkcja aktywacji może być dowolna. Jeśli funkcja aktywacji jest tylko progiem binarnym, nazywa się ją perceptronem. Perceptrony są proste i bardzo ograniczone. Innym rodzajem funkcji aktywacji jest funkcja logistyczna:

$$g_u = \frac{1}{1+e^{-u}}$$

Jeśli i_1 to procent zielonych pikseli w blobie, powiedzmy 0,9, przy w_1 = 0,8, a i_2 to stopień, w jakim kształt plamki przybliża okrąg, powiedzmy 0,5, przy w_3 = 0,7, to dane wejściowe do h_1 są

$$input_{h_1} = i_1 * w_1 + i_2 * w_3 = 0.9 * 0.8 + 0.5 * 0.7 = 1.07$$

Wyjście ukrytej jednostki h_1 byłoby:

$$g_{h_1} = \frac{1}{1+e^{-h_1}} = \frac{1}{1+e^{-1.07}} = \frac{1}{1+0.343} = 0.745$$

Wyjście jednostki może być wykorzystane jako bodziec dla innej jednostki neuronalnej lub jako siła ogólnego bodźca dla koncepcji wyjścia. Na rysunku wyjście h1 staje się wejściem tylko jednego wyjścia sieciowego, o1. Wszystkie koncepcje wielu wyników będą miały pewien stopień aktywacji, a ta z najwyższą aktywacją zostanie uznana za poprawną (chociaż SSN może popełnić błędy w klasyfikacji).

UKRYTE WARSTWY

Sztuczne sieci neuronowe są w stanie dokonywać nieliniowych klasyfikacji, ponieważ dzięki obecności ukrytych warstw mogą z dużą elastycznością kojarzyć wejścia z wyjściami. SSN nie ograniczają się do generowania wag dla koncepcji wyniku wyłącznie na podstawie bezpośrednich danych wejściowych. Na przykład sygnalizacja świetlna mogła wykryć sygnały wejściowe w postaci żółtego prostokąta, zielonego okrągłego kształtu wewnątrz żółtego prostokąta, czerwonego okrągłego kształtu wewnątrz prostokąta. Jeśli obiekt ma mocno żółty prostokąt i zielony okrągły kształt wewnątrz bodźca w postaci żółtego prostokąta lub mocno żółty prostokąt i czerwony okrągły kształt wewnątrz prostokąta, prawdopodobnie jest to sygnalizacja świetlna. Jeśli jednak zarówno zielony okrągły kształt wewnątrz żółtego prostokąta, jak i czerwony okrągły kształt wewnątrz prostokąta są silnie wykrywane w tym samym czasie, coś jest nie tak i sieć może nie być w stanie zidentyfikować obiektu. Elastyczność relacji uczenia się jest zapewniana w SNN przez pośrednie jednostki neuronalne zwane ukrytymi warstwami. Rysunek przedstawia SSN z jedną ukrytą warstwą.

GŁĘBOKA NAUKA

Słowo „ukryte” w określeniu ukryte warstwy okazuje się trafne. W sieciach z dużą liczbą jednostek i ukrytych warstw, zwanych także głębokim uczeniem, nieliniowe relacje między jednostkami stają się tak złożone, że analiza tego, czego sieć się nauczyła, jest trudna lub niemożliwa. Oznacza to, że projektanci nie byliby w stanie wyrazić sieci neuronowej w przykładzie sygnalizacji świetlnej w poprzednim akapicie za pomocą reguł „jeśli-to” lub drzewa decyzyjnego. Ten projektant musi tylko zaufać, że ukryta część sieci neuronowej wykonuje swoją pracę. Ukryte warstwy stanowią również wyzwanie projektowe, ponieważ optymalna liczba ukrytych warstw dla problemu jest nieznana; projektant może być zmuszony do uruchomienia danych uczących w kilku wersjach sieci z różną liczbą warstw ukrytych. Należy zauważyć, że rysunek 16.5 przedstawia sieć po nauczeniu się połączeń między jednostkami. Sieć często zaczyna się od skierowania wszystkich możliwych wejść do jednostki lub w pełni połączonego pojedynczego wykresu kierunkowego. Uczy się, że niektóre połączenia nie mają znaczenia dla konkretnej jednostki, dzięki czemu można wyeliminować te krawędzie.

WYPRZEDZENIE

NAWRACAJĄCY

Jednostki w sieci można łączyć na dwa podstawowe sposoby: sprzężenie zwrotne i rekurencyjne. W sieciach ze sprzężeniem do przodu wszystkie strzałki idą w jednym kierunku, a bodziec wznosi się z bodźców sensorycznych, aby aktywować koncepcję wyjścia. Rysunek przedstawia sieć feedforward. Sieć rekurencyjna umożliwia dwukierunkowe strzałki i strzałki, które schodzą do niższych ukrytych warstw, tworząc pętle. Najczęściej spotykane są sieci typu feedforward.M]

PROPAGACJA WSTECZ

Wcześniej topologia ilustrowała, jak klasyfikować obiekt lub koncepcję na podstawie danych wejściowych lub poruszania się naprzód w sieci, ale nie w jaki sposób sieć się uczy. Mechanizm uczenia

się w sieciach neuronowych polega na dostosowaniu wagi w_i w celu zmniejszenia błędu między wyjściem o_i obliczonym przez sieć a docelowym wyjściem, które powinien był obliczyć dla wejścia i_i. Najpopularniejszą metodą dostosowywania wag jest propagacja wsteczna. Propagacja wsteczna rozkłada błąd między wyjściami wstecz wzdłuż obciążników na krawędziach łączących. Na przykład na rysunku powyżej całkowity błąd E można oszacować jako:

$$\begin{aligned} E_{total} &= \Sigma \frac{1}{2}((target_{o_i} - out_{o_i})^2) \\ &= \frac{1}{2}(target_{o_1} - out_{o_1})^2 + \frac{1}{2}(target_{o_2} - out_{o_2})^2 + \frac{1}{2}(target_{o_3} - out_{o_3})^2 \end{aligned}$$

Propagacja wsteczna wykorzystuje ten błąd do aktualizacji wagi. Chodzi o to, że cząstkową pochodną zmiany błędu Etotal w odniesieniu do każdej indywidualnej wagi w_i można obliczyć jako:

$$\frac{\delta E_{total}}{\delta w_i}$$

Na przykład, aby dostosować w₉ w celu zmniejszenia błędu, należałoby obliczyć $\sigma E_{total}/\delta w_9$. Ponieważ δw_9 nie jest znana, można zastosować zasadę łańcucha:

$$\frac{\delta E_{total}}{\delta w_9} = \frac{\delta E_{total}}{\delta o_1} * \frac{\delta o_1}{\delta input_{o_1}} * \frac{input_{o_1}}{\delta w_9}$$

Reguła łańcucha mówi, że błąd wagi w₉ jest funkcją tego, jak bardzo zmienia się całkowity błąd ze względu na zmianę wyjścia o1, jak bardzo zmienia się wyjście o1 ze względu na zmiany wejścia na o_1. $input_1$ i jak bardzo inputo1 zmienia się z powodu zmiany w w₉. Matematycznie uogólnia to regułę delta:

$$\frac{\delta E_{total}}{\delta w_9} = -(target_{o_1} - output_{o_1}) * output_{o_1}(1 - output_{o_1}) * output_{h_1}$$

Reguła delta jest używana do obliczenia udziału w9 w całkowitym błędzie, jak bardzo zmienił się błąd w E_{total} z powodu błędu w w9. Aby skorygować w9, ten błąd jest odejmowany od w9. Jednak często stosuje się do reguły delta dyskont lub współczynnik uczenia, α, aby stłumić wszelkie duże zmiany obliczonego błędu spowodowane szumem w zbiorze uczącym. Aktualizacja dla w9, która jest często oznaczona indeksem górnym +, aby wskazać, że się zmieniła, to:

$$w_9{}^+ = w_9 - \alpha * \frac{\delta E_{total}}{\delta w_9}$$

Proces jest powtarzany, działając wstecz (stąd termin „propagacja wsteczna") od wyjść sieci, aż do skorygowania wszystkich wag. Sieć jest przedstawiona na innym przykładzie i pracuje do przodu, aby obliczyć dane wyjściowe sieci. Te wyjścia mogą nadal mieć znaczny błąd, który następnie prowadzi do fazy wstecznej propagacji. Po wielu próbach, być może setkach lub tysiącach, sieć skupia się na optymalnych wartościach wag. Jednak to optimum dotyczy określonej liczby warstw ukrytych i kategorii wyjściowych; inna liczba ukrytych warstw może dać lepszy wynik.

Nauka przez wzmacnianie

POLITYKA

Uczenie nadzorowane i nienadzorowane jest zwykle związane z rozpoznawaniem obiektów lub wzorców uczenia się, podczas gdy uczenie ze wzmocnieniem jest często stosowane, gdy agent chce poznać zasady, oznaczane jako Π, jakie działania podjąć w określonych okolicznościach. Polityka może być dość wyrafinowana. Rozważ naukę rzucania piłką, gdy następna akcja jest częścią większej sekwencji podnoszenia ramienia, podnoszenia stopy, ustawiania tułowia, zginania nadgarstka, rzucania nieco mocniejszymi, aby przezwyciężyć obecny wpływ wiatru i tak dalej. Co więcej, każdy element w sekwencji ma wiele zmiennych, które można regulować, w tym zmianę prędkości stawu, jeśli jego położenie jest nieco odchylone. Podstawowa idea polega na tym, że agent uczy się tych regulacji podczas eksploracji problematycznej przestrzeni, tak jak miotacz uczy się poprzez praktykę, jak koordynować swoje ciało, aby rzucić piłkę, która wzmacnia to, co działa, a co nie.

PROBLEM CESJI KREDYTOWEJ

W uczeniu się przez wzmacnianie celem jest nauczenie się Π, każdego zestawu działań, aby osiągnąć cel. Agent może stwierdzić, kiedy osiągnął bramkę, a nawet czy zbliżył się do celu, tak jak miotacz widzi wyniki rzutu. Ale agent na ogół nie zna użyteczności żadnych etapów pośrednich ani stanu - Czy windup był dobry? Czy stopa została podniesiona zbyt wysoko lub zbyt wcześnie? W idealnym przypadku robot chciałby ocenić użyteczność stanów pośrednich w celu zbudowania polityki która mówi: „Zawsze podnoś stopę tak wysoko, a następnie, jeśli wiatr wieje, dostosuj przystawkę…”. Jeśli osiągnięcie celu jest nagrodą, program uczenia się rozdziela nagrodę „do tyłu” na każdy stan w sekwencji w celu zwiększenia oceny użyteczności rzeczy, które wydawały się działać. Trudność w określeniu liczbowej miary wkładu każdego państwa w osiągnięcie celu nazywa się problemem alokacji kredytów. Problem z przypisaniem kredytu to pytanie: Jaki stan zadziałał najlepiej? Przypisanie kredytu jest powierzchownie podobne do wstecznej propagacji, ponieważ oba działają wstecz. Ale propagacja wsteczna zmniejsza błąd w generowaniu danych wyjściowych, podczas gdy przypisanie kredytu zwiększa nagrodę do stanów, które bezpośrednio prowadzą do danych wyjściowych.

PROCES DECYZJI MARKOWA

Algorytmy uczenia się ze wzmocnieniem zwykle zakładają, że to, czego się uczy, jest procesem decyzyjnym Markowa. Oznacza to, że robot może wybrać dowolną akcję, która jest możliwa z obecnego stanu, a wybór zależy tylko od aktualnego stanu. Wybór działania jest warunkowo niezależny od poprzednich wyborów. Miotacz może poeksperymentować z podnoszeniem stopy w dowolnym momencie wypróbowywania sekwencji miotania, o ile jest to możliwe. Głównymi technikami uczenia się przez wzmacnianie są funkcje użytkowe i Q-learning.

Funkcje użytkowe

Funkcje użytkowe zostały wykorzystane w jednym z oryginalnych badań uczenia się. W 1959 roku Samuels, nie zdając sobie sprawy, że warcaby są bardziej złożone obliczeniowo niż szachy, postanowił stworzyć program, który nauczyłby się grać w warcaby. Jego program warcabowy w końcu odniósł sukces i nadal jest podstawą uczenia maszynowego. Kluczowym pomysłem było przedstawienie aktualnej pozycji pionów na planszy, nauczenie programu użyteczności tej pozycji na planszy, a następnie, podczas gry, wyszukanie następnej najlepszej pozycji na planszy. Wartość pozycji na szachownicy była reprezentowana jako funkcja użyteczności, z wagą przypisaną do każdego aspektu pozycji na szachownicy: liczba pionów czarnych na szachownicy, BB, liczba pionów czerwonych na szachownicy, RB, liczba pionów czarne króle, BK, zagrożona liczba czerwonych pionów, RT i tak dalej. Wagi w funkcji użyteczności są podobne do wag w prostej SSN perceptronu, gdzie wartość V ruchu skutkującego pozycją na planszy b wynosi:

$V_b = BB * w_1 + RB * w_2 + BK * w_3 + RT * w_4$ …

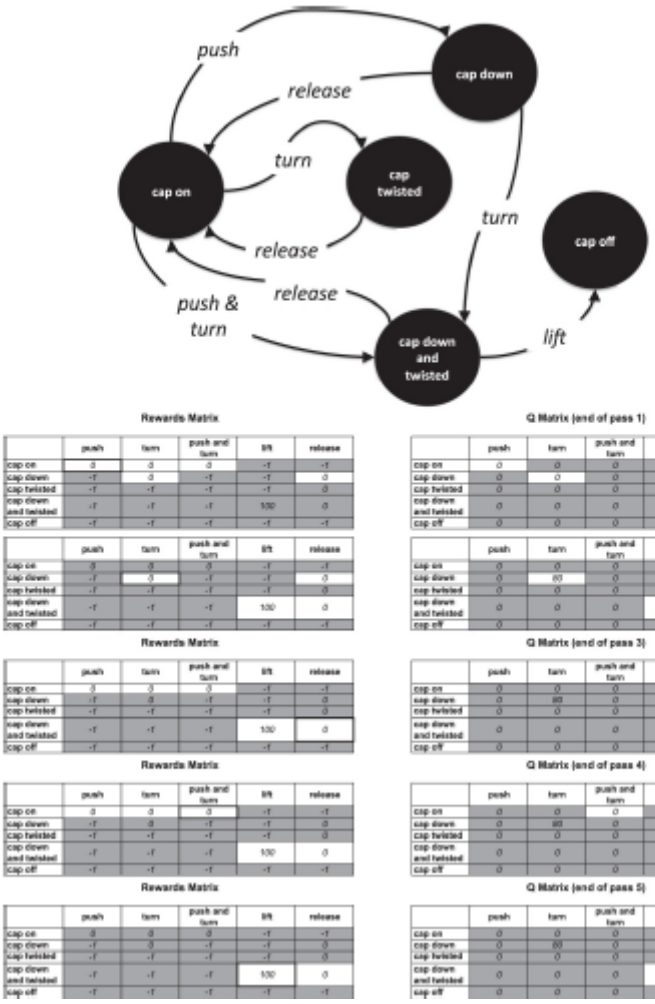
Poważnym wyzwaniem w funkcjach użyteczności jest określenie atrybutów, a nie wag. SSN mają podobny problem, ponieważ dane wejściowe do sieci muszą mieć charakter informacyjny.

Q-LEARNING

Funkcje użytkowe są intuicyjnie atrakcyjne, ale, jak stwierdził Samuels, może być trudno wygenerować pojedynczą funkcję, która wyraża wartość szerokiego zakresu możliwych stanów. Alternatywnym podejściem jest po prostu ocena użyteczności państwa w osiąganiu celu. Jeśli podniesienie stopy na początku windupu jest zwykle związane z dobrym nachyleniem, to stan ten powinien mieć wyższą pozycję na początku. Po raz kolejny problem polega na tym, jak rozwiązać problem przypisania kredytu, unikając reprezentowania, generowania i ustawiania dziesiątek wag. Jednym z rozwiązań jest Q-learning, najpowszechniejsza klasa algorytmów uczenia ze wzmocnieniem, w której wzmocnienie dla osiągnięcia celu jest rozłożone w sekwencji. Użyteczność wyboru działania w stanie jest następnie aktualizowana w wielu sekwencjach prób i ostatecznie zbiega się w celu uzyskania optymalnej polityki. Q symbolizuje funkcję Q, która oblicza oczekiwaną użyteczność Q podjęcia określonego działania a w stanie s. Najbardziej podstawowa formuła Q-learningu to:

$$Q(s,a) = R(s) + \gamma argmax_{a'}(Q(s,a'))$$

gdzie Q(s, a) jest szacowaną użytecznością bycia w stanie s i podjęcia działania a. Przewidywana użyteczność to nagroda R(s) za bycie w obecnym stanie plus γargmaxa' (Q(s, a')), która jest szacowanym wkładem z podjęcia działania a i raz w nowym stanie s', a następnie podjęcia najlepsze możliwe . Szacowane narzędzie składa się z trzech części. Pierwsza to nagroda a priori R(s) za osiągnięcie stanu s. W większości przypadków nagroda będzie znana tylko ze stanu celu, dlatego istnieje problem z przypisaniem kredytu. Drugi składnik, argmaxa' (Q(s, a')) to jednoetapowe spojrzenie w przyszłość, aby uzyskać szacowaną użyteczność następnego najlepszego kroku. Trzecią funkcją jest funkcja dyskontowa, γ, która zawiera pogląd, że im dalej stan pośredni znajduje się od stanu pożądanego, tym prawdopodobnie ma mniejszą użyteczność, a zatem powinien mieć zmniejszający się udział nagrody lub kredytu. Rysunek przedstawia przykład Q-learningu.



Rozważ robota, który otrzymuje butelkę i ma za zadanie zdjąć nakrętkę zabezpieczającą przed dostępem dzieci. Osoba może grzebać w próbach różnych działań, dopóki ograniczenie nie zostanie zniesione, a następnie spróbuje jeszcze kilka razy, aby upewnić się, że polityka działa. W Q-learningu robot robi mniej więcej to samo, choć nazywa się to eksploracją stanu. Początkowo robot zna tylko możliwe przejścia między stanami i akcjami oraz nagrodę za wybranie określonej akcji. Zbiór przejść to proces decyzyjny Markowa. Przypomnijmy, że proces decyzyjny Markowa nie ma pamięci, ponieważ nie obchodzi go, w jaki sposób robot dotarł do obecnego stanu; troszczy się tylko o to, co robić teraz, gdy już tam jest. Właściwość bez pamięci jest przydatna, ponieważ algorytm nie musi nadążać za wszystkimi możliwymi sposobami osiągnięcia stanu. Proces decyzyjny Markowa można zwizualizować jako diagram automatu skończonego, który wyraża możliwe przejścia między stanami i działaniami. W prawdziwym życiu mogą istnieć parametry pchania, które reprezentują robota pchającego mocniej lub bardziej miękko, dając bardziej złożony zestaw przejść. Wracając do problemu z zabezpieczeniem przed dziećmi, widoczna inspekcja diagramu na rysunku daje wiele zasad, aby znieść ograniczenie. Dwa najbardziej bezpośrednie to jednocześnie pchanie i obracanie, a następnie podnoszenie nasadki lub pchanie, obracanie i podnoszenie. pchnij i obróć, a następnie podnieś jest intuicyjnie atrakcyjna jako optymalna polityka, ponieważ wymaga tylko dwóch kroków, podczas gdy pchnij, obróć, a następnie podnieś wymaga trzech kroków. W ten sposób proces przypisywania kredytów może faworyzować krótsze sekwencje. Optymalne strategie nie będą łatwo widoczne ani odkryte przez algorytm wyszukiwania A* w bardziej realistycznych sytuacjach. Wyszukiwanie może tworzyć ścieżki, ale w tym

przypadku nie ma żadnych kosztów związanych z łączami, co uniemożliwia algorytmowi wyszukiwania zidentyfikowanie optymalnych ścieżek. Gdyby istniał proces przypisywania kredytów, można by ocenić użyteczność każdego kroku w sekwencji. Innym sposobem podejścia do problemu znalezienia polis jest rozważenie, w jaki sposób dana osoba po raz pierwszy radzi sobie ze zdjęciem zabezpieczenia przed dziećmi. Osoba może grzebać, to znaczy eksplorować przestrzeń (stanu, działania), próbować różnych rzeczy i ostatecznie zdjąć czapkę, aby osiągnąć stan docelowy i ostatecznie odnieść sukces dzięki polityce pchania, puszczania, pchania i obracania. Ta polityka wyraźnie nie jest optymalna, ale zadziała. Najlepiej byłoby, gdyby osoba zdała sobie sprawę, że posiadanie w polisie „pchnij i zwolnij" nie dodaje żadnej użyteczności (tj. zastosuj przypisanie kredytu), a następnie spróbuj ponownie, po prostu pchnij i obróć, a następnie podnieś i zobacz, czy to rzeczywiście zadziałało (tj. dalsza eksploracja). Gdyby tak było, pchnij i obróć, a podnoszenie zostanie zapamiętane jako prostsza polityka do zastosowania w przyszłości. Ewentualnie może się okazać, że dwie inne zasady, pchaj i obracaj jednocześnie lub pchaj i obracaj, pozwoliły na zdjęcie nasadki. Albo osoba ta może tylko pamiętać „zrób wszystko, dopóki czapka nie zostanie zepchnięta i przekręcona, a potem możesz ją podnieść". Niezależnie od stosowanej polityki, najcenniejszym stanem do osiągnięcia jest dojście do dolnej części czapki i zakręcony stan ponieważ cel można osiągnąć bezpośrednio z tego stanu. Q-learning próbuje powielić ten rodzaj ludzkiego uczenia się, badając sekwencje (stanu, działania) i aktualizując szacowaną użyteczność przy użyciu komputerowej strategii programowania dynamicznego. Q-learning powiela błądzenie, losowo eksplorując przestrzeń (stanu, działania). Powiela proces poznawczego przypisywania kredytów, w którym niektóre stany są określane jako obce lub bardziej wartościowe niż inne, za pomocą programowania dynamicznego. Przypomnijmy, że programowanie dynamiczne to technika programowania, która ogranicza obliczenia, przechowując częściowe rozwiązania lub rozwiązania podproblemów, które napotyka, gdy program pracuje nad rozwiązaniem większego problemu. W tym przypadku podproblemy są szacowaną użytecznością każdego (stanu, działania) dla osiągnięcia celu. Jeśli algorytm Q-learning może badać przestrzeń w wielu próbach, przejdzie przez wiele polityk, wyląduje we wszystkich stanach i, jeśli istnieje metoda przypisania kredytu, oszacuje użyteczność każdego z kroków. Ostatecznie algorytm zbiegnie się na szacowanej użyteczności każdego stanu, co z kolei prowadzi do optymalnej polityki. Jeśli szacowana użyteczność jest znana dla każdego stanu, agent wie, jakie jest najlepsze działanie, które należy podjąć w następnej kolejności, jeśli rozpocznie się z wyłączonym limitem. Następnie, kiedy robi następny krok i wchodzi w nowy stan, wie, jakie jest najlepsze działanie z tego nowego stanu i tak dalej. Zauważ, że agent nie musi pamiętać, jaką sekwencję wykonać, tylko co zrobić, jeśli jest w określonym stanie. Q-learning rozwiązuje problem akredytacji z funkcją dyskonta$_{\gamma}$. Ogólna idea jest taka, że Q(s, a) jest szacowaną użytecznością najlepszego możliwego wyboru następnego stanu i działania lub jednoetapowego spojrzenia w przyszłość. Zatem najlepszym możliwym wyborem działania jest maksimum Q(s', a'). Ale intuicyjnie bycie w (s, a) nie jest tak dobre, jak bycie w najlepszym możliwym (s', a'), więc Q(s, a) powinno być mniejsze niż argmax(Q(s', a')). Dlatego dyskontujemy argmax(Q(s', a')) o dowolną wartość, powiedzmy 80%, gdzie γ = 0,8. Jeśli eksploracja wyląduje na stanie cap down i skręcona i wykona windę, która przeniesie ją do celu, a cel ma nagrodę w wysokości 100, równanie 16.1 mówi, że szacunkowa wartość Q*(cap down and twisted, lift) wynosi 80 Następnie następnym razem, gdy eksploracja znajdzie się w stanie i akcji, której najlepszym wyborem jest góra w dół i skręcenie, uniesienie, wartość tego (stan, akcja) zostanie zaktualizowana do 64.

Przykład Q-learningu

Powyższy rysunek formalizuje możliwe przejścia między stanami w celu usunięcia zakrętki zabezpieczającej przed dziećmi z butelki. Ogólna metoda zdejmowania nasadki polega na jednoczesnym wciśnięciu i przekręceniu, a następnie podniesieniu nasadki. Alternatywnie nasadkę można wcisnąć, następnie przekręcić, a na koniec podnieść. Te przejścia stanów są znane a priori.

MATRYCA NAGRÓD

Nagrody a priori i bieżące oszacowane Q dla każdego stanu i działania są utrzymywane w równoległych macierzach pokazanych na rysunku. Zwróć uwagę, że w tym przykładzie jedyną nagrodą jest podniesienie limitu, jak pokazuje pojedynczy wpis 100 w macierzy nagród. To nie jest rzadkością; Q-learning jest często stosowany w sytuacjach, w których znany jest tylko stan końcowy. Jednak nagrody mogą być rozdzielane w całej macierzy nagród. Niezależnie od tego macierz nagród pozostanie niezmieniona podczas wykonywania algorytmu. Wszystkie wpisy w macierzy Q są inicjowane na 0 i są aktualizowane w miarę eksploracji agenta. Algorytm jest iteracyjny:

```
for each episode
     select a random initial state s
     do
       randomly select an action a from
          the set of possible a for that s
       estimate the utility of the new state s'
       update the Q-matrix
     until goal state is reached
```

Q-learning zakłada, że proces uczenia się nie będzie w stanie zbadać wszystkich możliwych kombinacji ze względu na dużą liczbę możliwości, dlatego projektant określa liczbę odcinków lub prób. Im więcej odcinków, tym bardziej proces uczenia się zbiega w kierunku optymalnej polityki.

NAUKA ONLINE

Q-learning może być używany do uczenia się online, czasami nazywanego uczeniem ciągłym, ponieważ robot wykonuje swoją normalną rutynę, korzystając z tego, czego nauczył się do tej pory. W uczeniu się online robot korzystałby z bieżących oszacowań użyteczności w macierzy Q, aby wybrać następną akcję, a jeśli wszystkie akcje prowadziłyby do stanów z 0, wybierałby losowo iw ten sposób eksplorował przestrzeń. Jest to przeciwieństwo uczenia nadzorowanego, w którym cała nauka odbywa się w trybie offline, na przykład uczenie się rozpoznawania różnych obiektów. Algorytm rozpoczyna odcinek. W przypadku nauki zdejmowania kapsli jedynym prawnym stanem początkowym jest zakręcenie. Aby rozpocząć naukę, cap on to s. Następnie algorytm wchodzi w pętlę do. W pierwszym przejściu przez pętlę do limit w rzędzie w macierzy nagród wskazuje, że są trzy możliwe akcje: pchanie, obrót, pchanie i obrót. Te trzy możliwości są zaznaczone na powyższym rysunku. Losowy wybór wybiera push, więc bieżąca krotka (s, a) to (cap on, push). Wykonanie tury akcji spowoduje nowy stan s cap down. Teraz można zastosować równanie do oszacowania użyteczności wyboru skrętu ze stanu nasadki; odpowiedni element w macierzy Q jest podświetlony na rysunku. Wracając do Matrycy Nagród, możliwe działania ze stanu ograniczenia to {turn, release}, które również są podświetlone.

$$\begin{aligned} Q(cap\,on, push) &= R(cap\,on, push) + \\ &\quad 0.8 argmax(Q(cap\,down, turn), Q(cap\,down, release)) \\ &= 0 + 0.8 argmax(0, 0) \\ &= 0 \end{aligned}$$

Robot wie w tej chwili, że zaczynając od nałożenia limitu, a następnie stosując pchnięcie, pchnięcie ma użyteczność równą 0. Jednak pchnięcie może mieć wyższą użyteczność, która zostanie odkryta później. Matryca Q pozostaje niezmieniona. Robot znajduje się w stanie czapki opuszczonej, co nie jest stanem docelowym, w którym czapka jest wyłączona, więc odcinek nie jest kompletny. Dlatego algorytm wykonuje drugie przejście przez pętlę do. Obecny stan to s=cap down, a następnym krokiem jest

losowe wybranie legalnej czynności z {turn, release}, co jest zaznaczone na rysunku. Załóżmy, że wybór to obrót, co oznacza, że robot zbadał zasadę rozpoczynania od czapki, stosowania pchania, a następnie obracania. To prowadzi do s'=cap down i skręcania. Użyteczność kroku (opuść, obróć) w tej sekwencji można oszacować za pomocą równania:

$$\begin{aligned} Q(cap\,down, turn) &= R(cap\,down, turn) + \\ &\quad 0.8 argmax(Q(cap\,down\,and\,twisted, lift), \\ &\quad Q(cap\,down\,and\,twisted, release)) \\ &= 0 + 0.8 argmax(100, 0) \\ &= 80 \end{aligned}$$

Macierz Q została zaktualizowana, aby pokazać, że użyteczność zastosowania odwrócenia od limitu w dół jest warta 80. Obecny stan to góra w dół i skręcona, co nie jest celem, więc algorytm znajduje się na szczycie pętli do dla trzeciego przejścia . Ponownie wybierany jest losowy stan, z (podniesienie, zwolnienie) jako opcje podświetlone na rysunku . Powiedzmy, że wybór losowy to zwolnienie, chociaż możemy spojrzeć na stany i wiedzieć, że wzrost byłby lepszy. W końcu robot to zrozumie. Powoduje to powrót robota do stanu czapki. Szacunkowa użyteczność (nasadka w dół i skręcona, zwolnienie) to:

$$\begin{aligned} Q(cap\,down\,and\,twisted, release) &= R(cap\,down\,and\,twisted, release) \\ &\quad + 0.8(Q(cap\,on, push), Q(cap\,on, turn), \\ &\quad Q(cap\,on, push\,and\,turn)) \\ &= 0 + 0.8 argmax(0, 0, 0) \\ &= 0 \end{aligned}$$

Matryca Q pozostaje niezmieniona. Algorytm rozpoczyna czwarte przejście przez pętlę do. Robot jest teraz z powrotem w stanie cap on z możliwością wyboru {push, turn, push and turn}, zaznaczonych na rysunku. Tym razem robot losowo wybiera pchanie i obracanie, co powoduje powrót robota do nasadki opuszczonej i skręconego. Szacunkowa użyteczność tego wyboru to:

$$\begin{aligned} Q(cap\,on, push\,and\,turn) &= R(cap\,on, push\,and\,turn) + \\ &\quad 0.8 argmax(Q(cap\,down\,and\,twisted, lift), \\ &\quad Q(cap\,down\,and\,twisted, release)) \\ &= 0 + 0.8(0, 0) \\ &- 0 \end{aligned}$$

Matryca Q pozostaje niezmieniona, mimo że widzimy, że robot jest naprawdę blisko celu. Rozpoczyna się piąte przejście przez pętlę do. Obecny stan s jest górny i skręcony z opcjami {podnieś, zwolnij} podświetlonymi na rysunku. Tym razem losowo wybiera windę. W tym przykładzie diagram stanu pokazuje, że nie ma żadnych działań do podjęcia po osiągnięciu celu, chociaż w niektórych aplikacjach możliwe jest osiągnięcie celu, a następnie odejście, zamiast pozostawania w tym stanie (pomyśl o małych dzieciach, które majstrują przy nowe zadanie). Szacowana użyteczność (cap down and twisted, lift) degeneruje się do nagrody R (cap down and twisted, lift):

$$\begin{aligned} Q(cap\,down\,and\,twisted, lift) &= R(cap\,down\,and\,twisted, lift) + \\ &\quad 0.8(no\,states) \\ &= 100 + 0.8(no\,states) \\ &= 100 \end{aligned}$$

Macierz Q jest aktualizowana, a ponieważ (ograniczenie w dół i przekręcenie, podniesienie) osiągnie stan docelowy, odcinek jest zakończony. Tworząc losowy spacer po przestrzeni (stanu, działania) {pchnij, obróć, zwolnij, popchnij i obróć, podnieś}, robot uczy się, że jeśli kiedykolwiek będzie w stanie nasadki w dół i skręconym, aby wybrać podnoszenie i jeśli jest w stanie nasadki opuszczonej powinien wybrać obrót. Kolejne odcinki będą tworzyć różne losowe spacery i ostatecznie pełna matryca Q zbiegnie się.

Dyskusja Q-learning

W powyższym przykładzie robot miał dobrze określone zmiany stanów, ale w rzeczywistości wszystkie stany i zmiany stanów mogą nie być znane, dopóki nie wystąpią. Wracając do przykładu miotacza rzucającego piłkę, system nie musi z góry wyraźnie wyrażać wszystkich stanów, ponieważ można je obliczyć na podstawie kinematyki w razie potrzeby, zasadniczo — chcesz mieć nogę w górze i rękę z powrotem? Cóż, jedyne fizyczne ruchy nogi to opuszczenie jej lub podniesienie jej wyżej. W takim przypadku algorytm Q-learning może dynamicznie budować tabelę przejść, gdy je napotka. Równanie jest najbardziej podstawowym algorytmem Q-learning, ale istnieje wiele bardziej wyrafinowanych wariantów. Niektóre warianty uwzględniają, że wstępne oszacowanie użyteczności może być optymistyczne i wymaga weryfikacji podczas kolejnych wizyt w tym stanie. Inne warianty wykorzystują stopę uczenia się α w połączeniu ze stopą dyskontową, aby nauka była bardziej stopniowa, tak aby robot nie zatrzymał się na pierwszej polityce, którą z powodzeniem uruchomi.

Robotyka ewolucyjna i algorytmy genetyczne

EWOLUCYJNA ROBOTYKA

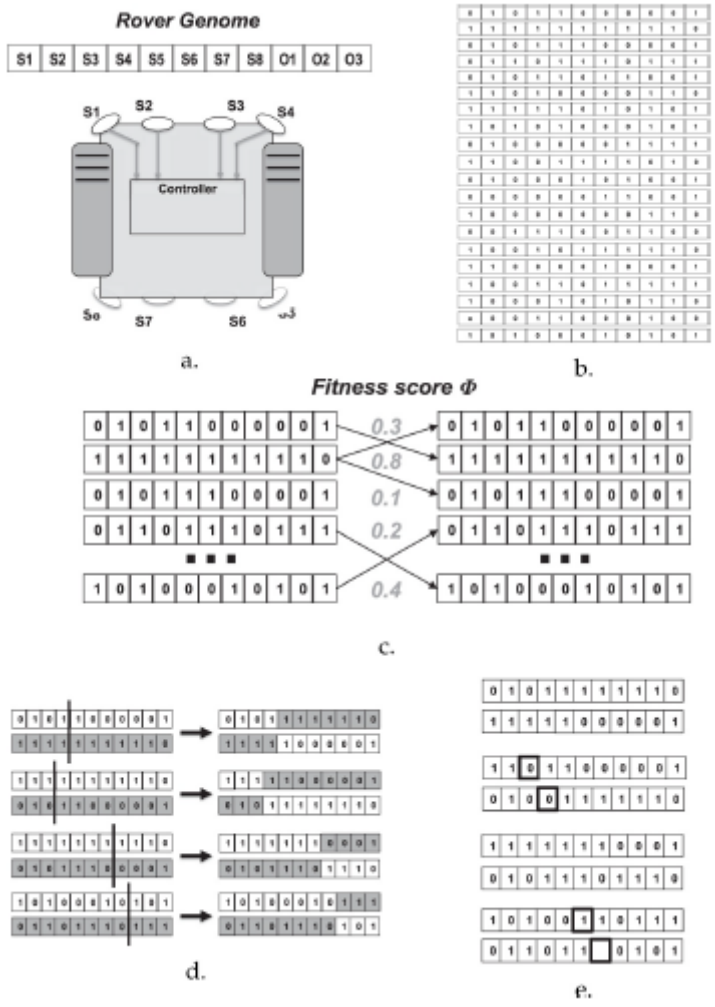
Robotyka ewolucyjna wykorzystuje procesy biomimetyczne do projektowania kształtu robota (morfologii), aby robot nauczył się nowych umiejętności motorycznych (takich jak chód), zadania lub przystosował się do bałaganu w środowisku. Chodzi o to, aby powielić ewolucję. Intuicyjnie ewolucja wydaje się być synonimem uczenia się, ponieważ agent poprawia swoje dopasowanie do świata. Istotną różnicą jest to, że uczenie się AI oznacza, w jaki sposób pojedynczy agent uczy się na podstawie własnych doświadczeń, a nie jak populacja agentów uczy się przez kilka wcieleń. Robotyka ewolucyjna zazwyczaj wykorzystuje algorytmy genetyczne, sztuczne sieci neuronowe lub ich kombinację w celu rozwinięcia możliwości robota. Sztuczne sieci neuronowe zostały już omówione; algorytmy genetyczne zostaną opisane bardziej szczegółowo poniżej. W praktyce robotyka ewolucyjna jest zainteresowana jedynie powielaniem adaptacji genetycznej, a nie zrozumieniem ewolucji kultury i społeczeństwa agentów, co jest czasami określane mianem epigenetyki.

ALGORYTMY GENETYCZNE

Algorytmy genetyczne (GA) podążają za ewolucyjnym procesem selektywnego rozmnażania, losowej mutacji i rekombinacji genetycznej, aby spełnić funkcję przystosowania. Symbol funkcji przystosowania może być zgodny z nomenklaturą rozpoznawania wzorców h, ale może być również zapisany jako φ.

GENOM

Rysunek ilustruje ogólny proces wykorzystania GA do ewolucji łazika planetarnego, który mógłby poruszać się przez skalistą pustynię, aby dotrzeć do pojazdu powrotnego próbki.



Jak pokazano na rysunku a, łazik ma dwa binarne czujniki celu, które widzą ponad skałami, więc jeśli pojazd powracający jest w zasięgu, łazik będzie mógł go zobaczyć. Łazik posiada osiem binarnych czujników przeszkód. Załóżmy, że łazik ma sterowanie poślizgowe i używa dekodera silnika, który destyluje dane wejściowe na trzy wyjścia binarne silnika. Genom łazika to wektor o rozmiarze 11 (8 wejść, 3 wyjścia), a genom odzwierciedla politykę kontrolowania robota. Celem jest nauczenie się mapowania między wejściami i wyjściami. Zbiór wejść to $2^8 = 256$ i jest $2^3 = 8$ możliwych wyjść. Oznacza to, że istnieje $8^{256} = 10^{231}$ możliwych asocjacji przechwyconych przez wektor do przeszukania. Duży rozmiar przestrzeni wyszukiwania motywuje do korzystania z techniki, która losowo próbkuje przestrzeń wyszukiwania, zamiast stosowania metody wyszukiwania opartej na drzewie, gałęzi i ograniczeniu, takiej jak algorytm wyszukiwania A* . Proces zaczyna się od generacji łazików działających na dość podobnym polu skalnym i tym samym celu. Pokolenie składałoby się z populacji pojedynczych łazików, z których każdy miałby inny genom. Każdy genom ma inną konfigurację wektora. Różne konfiguracje każdego łazika odzwierciedlają różnorodność populacji. W tym przykładzie istnieje 20 różnych konfiguracji łazików, tak więc populacja liczy 20 łazików, każdy z innym genomem, jak pokazano na rysunku b. Każdy pojedynczy łazik byłby umieszczany w losowym miejscu na polu skalnym i mógł poruszać się zgodnie ze swoim genomem, dopóki nie dotrze do celu, uderzy w przeszkodę lub skończy się czas. Każdy genom otrzymałby ocenę za swoje działanie za pomocą funkcji sprawności φ.

Projektowanie odpowiedniej funkcji dopasowania jest podobne do projektowania funkcji heurystycznej dla algorytmu wyszukiwania (patrz rozdział 14), ponieważ dobra funkcja prowadzi do lepszego rozwiązania, ale nie ma równoważnego kryterium dopuszczalności. Ten przykład wykorzystuje funkcję fitness z Keymeul

$$\phi = 0.5(1 - \frac{Distance_{remaining}}{Environmental_{Distance}} + 0.5(1 - \frac{Number_{steps}}{Steps_{maximum}})$$

Ta funkcja dopasowania zwraca wartość skalarną między [0,1], gdzie 1 oznacza doskonały wynik dla dopasowania. φ daje 50% kredytu na to, jak blisko łazik zbliża się do celu w tym okresie $\frac{Distance_{remaining}}{Environmental_{Distance}}$. $Distance_{remaining}$ to odległość między miejscem zatrzymania się łazika a celem; wynosi 0, jeśli łazik dotrze do celu. Odległość środowiskowa jest arbitralną miarą całkowitego obszaru, który łazik musi przebyć. Nawet jeśli łazik nie dotrze do celu, otrzymuje wyższy wynik, im bliżej celu się zbliża, zanim zawiedzie. Pozostałe 50% wyniku jest oparte na liczbie wykonanych kroków, kroków liczbowych lub cykli w pętli kontrolera robota. Ten wynik jest również arbitralną miarą maksymalnej liczby kroków uznanych za rozsądne dla łazika. $Number_{steps}$/ $Steps_{maximum}$ zbliża się do 1 jako całości 1- $Number_{steps}$/ $Steps_{maximum}$ zbliża się do 0, więc łazik, który robi mniej kroków, otrzyma wyższy wynik. Każdy pojedynczy łazik jest wielokrotnie testowany na polu skalnym. Można to zrobić, umieszczając pojedynczy łazik w losowym miejscu, pozwalając mu się poruszać, dopóki nie osiągnie celu lub skończy się czas, punktując go, a następnie przesuwając go w inne miejsce i kontynuując. załóżmy, że losowo wybrano 64 pozycje startowe. Pojedynczy łazik miałby zatem 64 próby, podczas gdy cała generacja 20 łazików miałaby 64 × 20 = 128 prób. Genom każdego łazika miałby ocenę sprawności dla każdej próby i średnią ocenę sprawności z 64 prób, jak pokazano na rycinie c.

FUNKCJA CROSSOVER

Po obliczeniu średnich wyników sprawności dla każdego z 20 łazików następnym krokiem jest zastosowanie selekcyjnego operatora genetycznego do obecnej populacji w celu stworzenia następnej generacji łazików poprzez rekombinację. Operator selekcji wybiera pary łazików, a następnie zamienia części ich wektorów za pomocą funkcji crossover. Jak pokazują strzałki na rysunku c, pary łazików są wybierane półlosowo, przy czym konfiguracje łazików o wyższej punktacji są sparowane częściej. Używając analogii z hodowlą zwierząt, łazik o wysokich wynikach trafia do stadniny. Algorytmy selekcji eliminują bardzo słabe wyniki, ale generalnie nie próbują eliminować wszystkich genomów o niskiej punktacji, ponieważ ich konfiguracje mogą mieć coś cennego, jeśli zostaną później połączone z inną konfiguracją lub jeśli łazik napotka inne środowisko; założenie to nazywa się „samolubną hipotezą DNA”. Po wybraniu par funkcja crossover określa, które geny należy zamienić na podstawie lokalizacji punktu crossover dla pary. Punkt przecięcia dla pary można wybrać losowo. Pokazują to pionowe linie na rycinie d. Można również zastosować operator genetyczny mutacji. Mutacja pomaga procesowi uwięzienia w lokalnym maksimum, w którym populacja może nie zmieniać się wystarczająco lub zmieniać się tylko w większych fragmentach utworzonych przez punkt przecięcia. Rysunek e pokazuje, że cztery łaziki miały mutacje. Po zastosowaniu selekcji, rekombinacji i mutacji powstaje nowa generacja genomów łazików. Proces może pozwolić na wzrost populacji lub wymusić kontrolę populacji i utrzymać każde pokolenie na 20 łazikach. Następna generacja genomów łazików przechodzi ten sam proces z tymi samymi 64 lokalizacjami początkowymi, stosuje operatory selekcji, rekombinacji i mutacji i tak dalej. W Kermuellen przydatny kontroler robota, który mógłby poruszać się po zaśmieconym obszarze, ewoluował w ciągu 20 pokoleń, chociaż ogólnie rzecz biorąc, heurystyka polega na tym, że do zaobserwowania zbieżności potrzeba 100 pokoleń. Powyższy przykład podkreśla dwa największe problemy związane z ewolucją: zajmuje dużo czasu i jest niepraktyczne w użyciu z fizycznymi robotami.

Projektant musiałby zaplanować uruchomienie łazików 1000 × 128 = 12 800 razy. Łazik może zostać uszkodzony przez uderzenie w skałę, co jest problemem, jeśli projektant ma tylko jednego robota do testowania i programuje 20 różnych konfiguracji. Nieodpowiednia konfiguracja może zrujnować łazik i zatrzymać ewolucję. Nawet jeśli łazik jest odporny na uszkodzenia, może wybić skały z pozycji lub w inny sposób zmienić środowisko testowe, więc projektant musi nie tylko poświęcić czas na ustawienie następnej konfiguracji łazika, ale także zresetować pole. Jeśli skonfigurowanie robota i zresetowanie pola zajęłoby projektantowi pięć minut, to 12 800 prób zajęłoby 1066 godzin lub więcej niż 44 dni pracy bez przerwy, aby ukończyć ewolucję. Te dwa problemy wyjaśniają, dlaczego robotykę ewolucyjną prawie zawsze przeprowadza się w symulacji komputerowej. Należy jednak pamiętać z rozdziału 10, że odczyty czujników w złożonych środowiskach mogą być bardzo trudne do poprawnej symulacji. Zestaw łazików wyewoluowanych w symulacji może nie działać tak dobrze, jak oczekiwano w świecie fizycznym. Łazik mógł mieć ukryte luki w zabezpieczeniach, ponieważ symulacja nie uchwyciła jakiegoś dziwactwa lub rzadkiej interakcji wykrywania w świecie fizycznym. Na szczęście algorytmy genetyczne mają drugie, być może bardziej praktyczne zastosowanie w robotyce: można je również wykorzystać do stworzenia zestawu testowego stanów dla innego algorytmu. Na przykład, zamiast używać algorytmu genetycznego do generowania genomów w celu uzyskania zbieżności na kontrolerze spełniającym funkcję przystosowania h, algorytm generuje przypadki testowe, w których funkcja fitness rejestruje, jak różne lub trudne są dane wejściowe. Na przykład może istnieć 1000 kombinacji skał i wzgórz. Inną strategią jest użycie algorytmów genetycznych do wytworzenia stanów, które stają się danymi wejściowymi do innego algorytmu. Na przykład, biorąc pod uwagę tę kombinację odczytów czujnika, dostosuj sieć neuronową tak, aby generowała właściwą reakcję.

Nauka i architektura

Poprzednie sekcje opisały rodzaje uczenia się, ale nie opisały, gdzie algorytmy mogłyby znajdować się w architekturze robota. To, gdzie nauka przebiega w kanonicznej architekturze hybrydowej, jest często nieistotne, ponieważ większość uczenia się odbywa się w trybie offline, to znaczy w symulacji, a nie online, gdzie robot uczy się, przechodząc przez swoją rutynę. Nauka online może znajdować się w dowolnym miejscu w architekturze kanonicznej, ponieważ robot może uczyć się reakcji, funkcji deliberatywnych lub funkcji interaktywnych, lub też w jakiejś kombinacji.

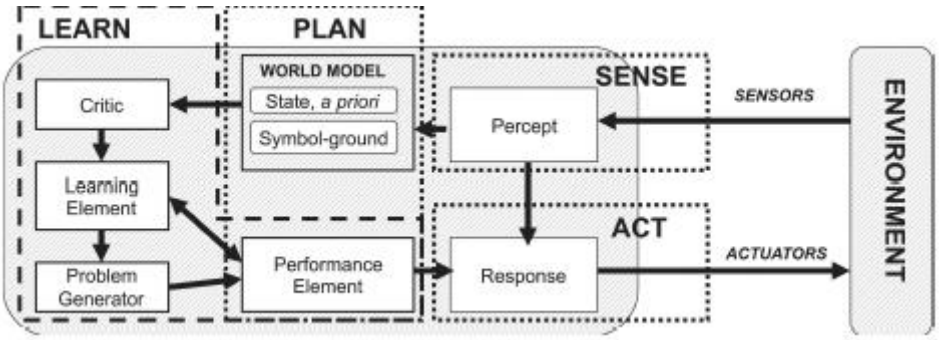
KRYTYK

ELEMENT NAUKI

ELEMENT WYDAJNY

GENEROWANIE PROBLEMÓW

Samo uczenie się online może mieć ogólną architekturę systemu, zgodnie z propozycją Toma Mitchella. Rysunek 8 przedstawia architekturę uczenia się w kategoriach czterech prymitywów robota: SENSE, PLAN, ACT i LEARN.



Przepływ architektoniczny zaczyna się od wyczuwania przez robota świata i tworzenia percepcji. Ta percepcja może być używana samodzielnie lub do aktualizacji modelu świata, w tym przetwarzania podstaw symboli. Postrzegany lub zaktualizowany model świata, w tym to, co było znane a priori, zostaje wprowadzony do krytyka. Krytyk to oddany agent, który identyfikuje, co jest nie tak z obecnym stanem świata lub można go poprawić w stosunku do celów lub oczekiwań. Krytyka można traktować jako sygnał błędu, na przykład, że obiekt został nieprawidłowo sklasyfikowany lub że rzut miotaczem wylądował nisko i w lewo, ale krytyk może być bardziej wyrafinowany. Wyniki krytyka trafiają do elementu uczenia się, który zapewnia informację zwrotną. Element uczący lub algorytm, który faktycznie przeprowadza uczenie się, pobiera dane wejściowe i dostosowuje element wydajności, co może stwarzać nowe problemy. Element wydajności to element, który faktycznie wykorzystuje naukę. W Q-learningu elementem wydajności byłaby polityka. Element uczący się może potrzebować robota, aby przerwał to, co robi, aby stworzyć własną miniaturową symulację problemów lub eksploracji stanu. Na przykład robot może zatrzymać się, aby ćwiczyć nową umiejętność. W robotyce opartej na chmurze robot może uzyskać dostęp do Internetu, aby spróbować znaleźć inne przykłady obiektu i dowiedzieć się, jak został on oznaczony przez inne systemy. Ta eksploracja, zarówno w trybie online, jak i offline, byłaby możliwa dzięki komponentowi generowania problemów. Instancja LEARN może być reaktywna lub deliberatywna: nie ma scentralizowanego modułu „ucz się wszystkiego". Różne rodzaje uczenia się są przydatne dla różnych funkcji robota, a każda instancja uczenia byłaby zaimplementowana zgodnie z tą samą ogólną architekturą. Architektura uczenia się nie wymaga, aby krytyk uważnie monitorował i rozważał rzeczywisty stan świata i porównywał go ze stanem idei świata. Reaktywny sygnał błędu, taki jak widziany przez Czyngisa uczącego się chodzenia po wejściu do przodu i bycia równym, jest krytykiem. Podobnie przed wznowieniem pracy robot nie musi generować nowych problemów, aby wypróbować to, czego właśnie się nauczył.

Luki i możliwości

Badanie przeprowadzone w 2012 r. przez Defense Science Board na temat autonomii systemów bezzałogowych dostarcza ważnych informacji na temat uczenia maszynowego i pozostałych luk w systemach autonomicznych. Najważniejsze to:

* Uczenie maszynowe zostało zastosowane przede wszystkim w bezzałogowych pojazdach naziemnych, a w mniejszym stopniu w pojazdach latających i morskich.

* Zastosowania były albo nieustrukturyzowanymi środowiskami statycznymi, takimi jak autonomiczna jazda przełajowa na pustyni, albo ustrukturyzowanymi, dynamicznie zmieniającymi się środowiskami, takimi jak autonomiczna jazda w mieście z innymi samochodami i pieszymi. Jednak niewiele aplikacji zostało opracowanych dla nieustrukturyzowanych i dynamicznych środowisk.

* Najczęściej stosowanym rodzajem uczenia się na przykładzie w systemach bezzałogowych jest uczenie nadzorowane. Zniechęcającym do korzystania z uczenia nadzorowanego jest to, że konstruowanie zestawów treningowych wymaga znacznego ludzkiego wysiłku.

* Nauka ze wzmocnieniem wymaga zbyt wielu prób, aby była praktyczna, ale proces można przyspieszyć poprzez uczenie się przez naśladownictwo, w którym człowiek wybiera kolejność stanów lub gdy człowiek udziela rad; w efekcie człowiek jest krytykiem.

* Uczenie maszynowe jest zwykle używane do rozpoznawania obiektów lub zasad uczenia się, ale nie jest używane tak często do wykrywania anomalii i świadomości sytuacji lub uczenia się nowego terminu. Uznanie, że element środowiska jest inny lub że robot może wpaść w pułapkę, nie zostało rozwiązane i pozostaje trudnym problemem, z którym musi sobie poradzić uczenie maszynowe.

Podsumowanie

Uczenie się na przykładzie to tylko jeden rodzaj uczenia się, ale to ten, który z największym powodzeniem został zastosowany w robotyce. Jednym z głównych zastosowań uczących się obiektów lub terenu jest zwykle uczenie nadzorowane lub nienadzorowane, gdzie duża liczba przykładów szkoleniowych jest prezentowana robotowi w trybie offline. Tworzenie zestawu szkoleniowego może być trudne, ponieważ liczba przykładów i kolejność prezentacji mogą wpływać na naukę. Co gorsza, algorytm może przepełnić h tam, gdzie działa idealnie dla zestawu uczącego, ale generuje błędy przy nowych danych. Ważne jest, aby walidować naukę za pomocą nowych danych, a nie ponownie używać tych samych przykładów, które zostały użyte w szkoleniu. Innym zastosowaniem uczenia się na przykładach jest poznawanie działań lub zasad, które określają, jakie działania należy podjąć i kiedy. Zasady są zwykle tworzone za pomocą Q-learningu. Nauka „nowego terminu" jest niezwykle trudna, ponieważ większość algorytmów uczących przyjmuje jakieś a priori założenie dotyczące tego, czego się uczy; na przykład, że należy się nauczyć pięciu rodzajów owoców; które podważają prawdziwe uczenie się nowych terminów; na przykład, że na obrazach jest tak naprawdę sześć rodzajów owoców. Uczenie się można zastosować do dowolnego aspektu działania robota i nie ogranicza się do warstw rozważanych lub reaktywnych. Nauka online jest rzadka; normą jest uczenie się offline, czyli uczenie się w symulacji. Symulacja może być zbiorem przykładów treningowych podawanych sztucznej sieci neuronowej lub robot może być dedykowany do eksploracji przestrzeni stanów za pomocą Q-learningu. Wdrażając nauczanie online, warto pomyśleć o uczeniu się jako składającym się z czterech elementów architektonicznych: krytyka, elementu uczenia się, generatora problemów i elementu wydajności. Wracając do pytań postawionych na wstępie, pierwsze pytanie brzmiało: Czym jest nauka? Jest to poprawa wydajności w przyszłych zadaniach po dokonaniu obserwacji świata. Rozdział zawierał odpowiedź Jakie są rodzaje uczenia się? zauważając, że istnieje wiele rodzajów uczenia się, ale uczenie się na przykładzie jest najczęstsze w przypadku robotów. Uczenie się przez przykład można podzielić na cztery główne typy: nienadzorowane, nadzorowane, wzmacniane i częściowo nadzorowane, chociaż częściowo nadzorowane uczenie się jest nadal nową subdziedziną i rzadko jest używane. W rozdziale omówiono również Dlaczego roboty nie korzystają cały czas z sieci neuronowych? Są jak mózg, prawda? Sztuczne sieci neuronowe są rzeczywiście podobne do mózgu, ale obecnie stanowią uproszczenie struktur neuronowych. Algorytmy często wymagają setek do tysięcy przykładów do celów szkoleniowych, a także pewnego sposobu określenia liczby ukrytych warstw, więc wdrożenie algorytmów może być trudne. SSN służą głównie do rozpoznawania lub klasyfikowania obiektów lub pojęć. W efekcie służą do uczenia się percepcji i ugruntowania symboli, a nie do uczenia się zasad czy sekwencji działań. Ostatnie pytanie brzmiało: dokąd prowadzi uczenie się w naszej architekturze operacyjnej? Ponieważ nie ma jednego elementu, którego należy się nauczyć, w architekturze nie ma jednego miejsca na „uczenie się". Zamiast tego, różne moduły edukacyjne mogą mieć wystąpienia w warstwach reaktywnej, deliberatywnej i interaktywnej.