Mesterséges Intelligencia (IB154) előadásjegyzet (vázlat)

Jelasity Márk

2020. december 30.

Jelen előadásjegyzet soha nincs készen, folyamatosan fejlődik. Amely részek nem vagy ritkán szerepelnek az előadáson, azok lassabban fejlődnek, és valószínűleg hibásabbak. Az anyag nagyrészt a következő könyv válogatott fejezeteire épül:

> Stuart Russell, and Peter Norvig. *Artificial Intelligence: A Modern Approach.* (second edition), Prentice Hall, 2003.

A könyv második kiadása magyar nyelven is elérhető:

> Stuart Russell, Peter Norvig. *Mesterséges intelligencia modern megközelítésben.* (2. kiadás), Panem Kiadó Kft., 2005.

Ahol további forrásokat használtunk fel, ezekre hivatkozunk lábjegyzetekben. Néhány ábra forrása a Wikipedia (`http://www.wikipedia.org/`).

A hibákért a felelősség a szerzőt terheli. A következők segítettek ezek számát csökkenteni: Szörényi Balázs, Berke Viktor, Halász Zsolt Attila, Csernai Kornél, Tóth Tamás, Godó Zita, Danner Gábor, Dékány Tamás, Pongrácz László, Heinc Emília, Hegedűs István, Margit Norbert.

1. fejezet

Bevezetés

1.1. Mi az MI tárgya?

Más szóval: mi az a probléma amit egy MI kutató meg akar oldani? Mikor örül a kutató, mikor tekinti megoldottnak a problémát?

emberi gondolkodás	emberi cselekvés
racionális gondolkodás	racionális cselekvés

cselekvés: A probléma itt olyan *ágens* tervezése, amely *érzékeli* a *környezetét*, és ennek függvényében *cselekszik*. Közben esetleg *tanul*, *alkalmazkodik*. Örülünk, ha az ágens *sikeres* valamely szempont szerint. A „hogyan” (mi van a „fejében”) lényegtelen.

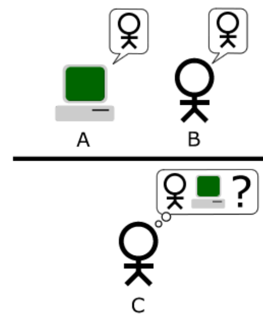
gondolkodás: A probléma itt modellek alkotása arról, hogy tudás (tapasztalat) és környezet alapján *hogyan* döntünk egyes cselekvések mellett. A „hogyan”-ról szól. Itt kicsit bonyolultabb, hogy mikor örülünk (persze akkor, ha „jó” a modell, de az mit jelent?).

emberi: Az etalon az ember. Örülünk, ha az emberihez közeli, hasonló tulajdonságok illetve teljesítmény állnak elő.

racionális: A sikeresség definíciója pragmatikus, az adott feladat által meghatározott, tehát örülünk, ha az adott feladatot a lehető legjobban megoldjuk. Így viszont bármit tekinthetünk intelligensnek, pl. a termosztátot is.

1.2. Példák kutatási célokra

1.2.1. Turing teszt (emberi cselekvés)



Egy ember rájön-e, hogy a másik szobában gép van? A gépnek képesnek kell lennie pl. természetes nyelv használatára, tudás reprezentációjára, következtetésre, tanulásra, stb. Sőt, modellezni kell az ember hibáit is, nem lehet túl okos sem.

Ha a gép átmegy a teszten, azt mondjuk, intelligens. De: van-e öntudata? legközelebb is átmenne? a tesztelő ember szakértelme mennyit számít? stb.

Érdekes probléma, de az MI *nem* erre fókuszál elsősorban.

1.2.2. Kognitív tudomány (emberi gondolkodás)

A Turing teszt „fekete doboznak” veszi a gépet. A kognitív tudomány számára a doboz belseje a fontos. Célja az emberi gondolkodás modellezése, végső soron az elme és az öntudat megértése.

Interdiszciplináris terület: nyelvtudomány, agykutatás, tudományfilozófia, pszichológia, etológia.

1.2.3. Következtetés, tudásreprezentáció (racionális gondolkodás)

Nem az emberi gondolkodás modellezése a cél (az emberek nem logikusak!), hanem a gondolkodás „ideális” modellje. Hagyományosan a különböző formális logikák vizsgálata tartozott ide, újabban a valószínűségszámítási alapokon álló megközelítések a népszerűek.

1.2.4. Racionális ágensek (racionális cselekvés)

Teljesen általános problémadefiníció (l. 2. fejezet), minden részterület beleilleszthető.

racionalitás $\approx$ optimalitás, korlátozott racionalitás $\approx$ heurisztikus döntések.

Az előadás során a mesterséges intelligencia problémáját a racionális ágensek vizsgálataként értelmezzük.

1.3. Multidiszciplináris gyökerek

1.3.1. Matematika

Logika és számítástudomány: a világról szóló állítások precíz formalizmusa, következtetési szabályok. Releváns részterületek: nemteljességi tételek (algoritmusok lehetetlensége), bonyolultságelmélet (algoritmusok plauzibilitása, NP-teljesség fogalma, stb.).

Valószínűség: a bizonytalanság és a véletlen események matematikája, és hatékony algoritmusok forrása.

1.3.2. Közgazdaságtan

modell: ágensek, amik hasznosságot (utility) optimalizálnak

Játékelmélet: kölcsönható ágensek, racionalitás, korlátozott racionalitás (Herbert Simon)

Döntéselmélet (bizonytalanság kezelése)

Multi-ágens rendszerek, emergens viselkedés

1.3.3. Idegtudomány

Neuron, neurális hálózatok modelljei, agy működése

Funkciók, pl. nyelv, lokalizációjának vizsgálata

Tanulás, adaptáció, önszerveződés (leképezések), masszív párhuzamosság, folytonos számítások (a GOFAI gyökeres ellentéte)

1.3.4. Pszichológia

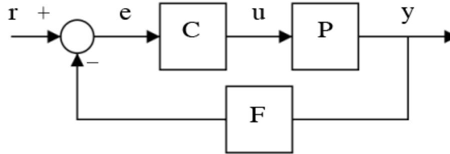
Emberi tanulás, viselkedés modellezése

Behaviorizmus: nem tételezünk fel közvetlenül nem megfigyelhető entitásokat, pl. gondolatok, vágyak, stb.: az agy fekete doboz, (esetleg bonyolult) feltételes reflexek tárháza.

Kognitív pszichológia: megenged modelleket, amelyekben fogalmak mint hit, cél, vágy, memória, stb. szerepelnek, ezeknek a következményeit ellenőrizhetjük kísérletileg (pl. fizikában is így van: az áram, hőmérséklet, molekula, stb. csak elméleti fogalmak, nem direkt megfigyelhetők).

1.3.5. Irányításelmélet, dinamikus rendszerek, kibernetika

Pl. zárt szabályozási hurok: a rendszer kimenetét ($y(t)$) az F szenzor méri, és a referenciaértéktől való eltérés ($e(t)$) alapján a P irányított rendszer inputját ($u(t)$) kiszámítjuk úgy, hogy a várható e-t minimalizáljuk.



Folytonos változók, nem szimbolikus, alkalmazásai robotika, repülés, stb.

Komplexebb változatok, pl. intelligens, sztochasztikus, adaptív irányítás

Stabilitás, káosz.

1.3.6. Egyéb

Filozófia, nyelvészet, számítógép architektúrák, stb.

1.4. Az MI története

1.4.1. Dartmouth workshop (1956)

Newell és Simon szimbólum manipuláló algoritmusa.

Az MI mint elnevezés itt születik, és a kutatói közösség is itt szilárdul meg.

1.4.2. Sikerek időszaka (1952-1969)

Szimbolikus: tételbizonyítás tökéletesedése, GPS (general problem solver), microworlds

Nem-szimbolikus: perceptron (egyszerű neuron modell) elméleti vizsgálata

1.4.3. Problémák jelentkezése (1966-1973)

Komplexitási problémák jelentkeznek komolyabb példákon (kombinatorikus robbanás), mert a legtöbb korai módszer *brute-force*: nagyon kevés háttértudást alkalmaznak, a hangsúly a keresésen van.

Pl. legendás bakik nyelvi alkalmazásokban: *The spirit is willing but the flesh is weak* → *The vodka is good but the flesh is rotten* (angol → orosz → angol) vagy *World shaken: Pope shot!* (eredetileg olaszul) → *Earth quake in Italy: One dead* (olasz → angol).

1.4.4. Tudás alapú (szakértői) rendszerek (1969-)

Területspecifikus tudás a hatékonyság növelésére, szakértőktől interjúk során szerezve. (pl. MYCIN orvosi diagnózis)

Előregyártott válaszok az extenzív keresés helyett

Komoly ipari alkalmazások is, pl. R1 (DEC) számítógépek konfigurálására: $40 milliót spórolt a vállalatnak.

80-as évek eleje a szakértői rendszerek virágzása, kitörő optimizmus.

1.4.5. MI telek (AI winters)

Hullámvasút: nagy várakozások és ígéretek után csalódás

- gépi fordítás: 60’
- konnekcionizmus (perceptron korlátai): 70’
- Lighthill report (UK, 1973)
- szakértői rendszerek: 90’

MI - sokan asszociálják a be nem váltott ígéretekkel: sokan kerülik az elnevezést.

De közben folyamatos a fejlődés.

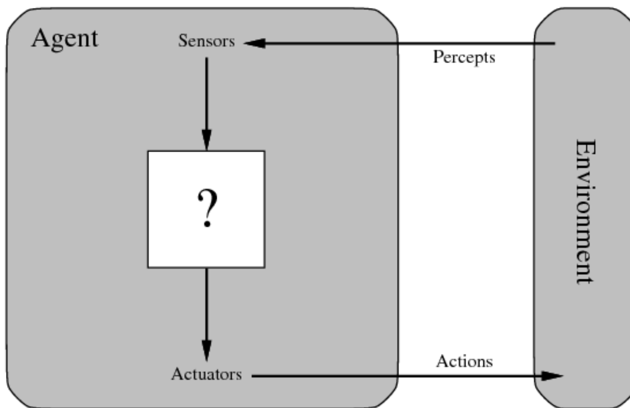
1.5. MI sikerek

Néhány nagyon specifikus területen embernél jobb, de az erős MI messze van.

Sikerek: sakk, go, póker (ember feletti szint), orvosi diagnózis, ma már autóvezetés is közel, robotika viharosan fejlődik, ajánlórendszerek, szemantikus keresés, fordítás, kép-, hangfelismerés, karakterfelismerés, stb.

2. fejezet

Ágensek



Racionalitás: minden érzékeléssorozatra olyan cselekvést választani, ami a várható teljesítményt maximalizálja (a beépített háttértudást is feltételezve). Tehát egy függvényt keresünk (*ágensfüggvény*), ill. egy azt megvalósító programot (*ágensprogram*) ami valamely megadott módon optimális (*teljesítménymértékek!*).

Ez *nagyon* általános definíció, pl. táblázattal is implementálhatjuk (de ez ok, mert azt mondtuk, a viselkedés érdekel minket).

Ez a keret alkalmas arra, hogy eldöntsük, mi MI és mi nem (bár elég sok minden lesz MI!).

A racionális cselekvés nem ugyanaz mint az adott állapot teljes ismeretén alapuló optimális cselekvés, hiszen az ágens általában nem tud mindent a környezetről.

2.1. Feladatkörnyezet

A feladatkörnyezet négy komponense a teljesítmény mértéke (amit a környezet állapotsorozatán értelmezünk), a környezet (environment), a beavatkozók (actuators) és az érzékelők (sensors).

Néhány szempont a feladatkörnyezet jellemzéséhez

Teljesen vagy részlegesen megfigyelhető? (adott absztrakciós szinten)

Determinisztikus vagy sztochasztikus? Determinisztikus akkor, ha az aktuális (objektív/szubjektív?) állapot és bármely cselekvés együtt egyértelműen meghatározza a következő állapotot. Egyébként sztochasztikus (pl. részlegesen megfigyelhető környezet tipikusan (szubjektíve) sztochasztikus, de lehet hibásan végrehajtott cselekvés, stb.).

Epizodikus vagy sorozatszerű? Egymástól *független* epizódok (érzékelés-akció), vagy folyamatos feladat (pl. autóvezetés)

Statikus vagy dinamikus? Akkor dinamikus, ha a környezet változik, míg az ágens tevékenykedik. Tágabb értelemben bármely komponens (állapot, teljesítmény, cselekvés hatása, stb.) függhet az időtől.

Lehet még *diszkrét vagy folytonos*, *egy- vagy többágenses*, stb.

2.2. Ágensprogramok típusai

Reflex ágensek: A cselekvés csak a környezetről rendelkezésre álló in-

formációktól (memória, világmodell, aktuális érzékelés) függ. *Egyszerű reflex ágens:* nincs belső állapot, a cselekvés csak az aktuális érzékeléstől függ. (Általában „unalmas”, de multiágens rendszerben mégis érdekes emergens jelenségek jöhetnek ki.)

Célorientált ágensek és *hasznosságorientált ágensek:* a memóriát és a világmodellt felhasználják arra, hogy megjósolják a saját cselekvéseik következményeit, és olyan cselekvést választanak, ami valamely előre adott célhoz vezet, vagy valamely hasznossági függvényt optimalizál.

Tanuló ágensek: lehet bármelyik fenti típus (reflex, stb.), lényeg, hogy van visszacsatolás az értékelőfüggvényről, aminek a hatására az ágens programja módósul.

3. fejezet

Informálatlan keresés

3.1. Állapottér

Tekintsünk egy *diszkrét*, *statikus*, *determinisztikus* és *teljesen megfigyelhető* feladatkörnyezetet. Részletesebben: tegyük fel, hogy a világ tökéletesen modellezhető a következőkkel:

- lehetséges állapotok halmaza
- egy kezdőállapot
- lehetséges cselekvések halmaza, és egy állapotátmenet függvény, amely minden állapothoz hozzárendel egy (cselekvés,állapot) típusú, rendezett párokból álló halmazt
- állapotátmenet költségfüggvénye, amely minden lehetséges állapot-cselekvés-állapot hármashoz egy $c(x, a, y)$ valós nemnegatív költségértéket rendel
- célállapotok halmaza (lehetséges állapotok részhalmaza)

A fenti modell egy súlyozott gráfot definiál, ahol a csúcsok az állapotok, az élek cselekvések, a súlyok pedig a költségek. Ez a gráf az *állapottér*.

A továbbiakban feltesszük, hogy az állapotok számossága véges vagy megszámlálható. Egy állapotnak legfeljebb véges számú szomszédja lehet.

Úton állapotok cselekvésekkel összekötött sorozatát értjük (pl. $x_1, a_1, x_2, a_2, \ldots, x_n$, melynek költsége $\sum_{i=1}^{n-1} c(x_i, a_i, x_{i+1})$).

Ebben a feladatkörnyezetben ha az ágens ismeri a világ modelljét, akkor nem is kell neki szenzor!

3.2. Példák

Térkép: állapotok=városok; cselekvés=úttal összekötött két város közti áthaladás; költség=távolság; a kezdő és célállapot feladatfüggő. A probléma itt útvonaltervezés, a térkép pedig a világ modellje.

Utazástervezési feladat: útvonaltervezéshez hasonló. pl. állapotok=hely és időpont párok; cselekvés=közlekedési eszközök, amelyek onnan és azután indulnak mint az aktuális állapot; költség=idő és pénz függvénye; a kezdő és célállapot feladatfüggő. Ez már egy fokkal bonyolultabb.

Porszívó világ: illusztrációnak (nem realisztikus): a világ két pozíció, mindkettő lehet tiszta vagy poros, ill. pontosan az egyikben van a porszívó. Ez 8 állapot. cselekvés=szív, jobbra, balra; költség=konstans minden cselekvésre; a célállapot a tisztaság.

8-kirakó (csúsztatós játék): állapotok=célállapotból (ábra) csúsztatásokkal elérhető konfigurációk; cselekvés=üres hely mozgatása fel, le, jobbra, balra; költség=konstans minden cselekvésre; a célállapotot az ábra mutatja.

	1	2
3	4	5
6	7	8

3.3. Kereső algoritmusok: fakeresés

Adott kezdőállapotból találjunk egy minimális költségű utat egy célállapotba. Nem egészen a klasszikus legrövidebb út keresési probléma: az állapottér nem mindig adott explicit módon, és végtelen is lehet.

Ötlet: *keresőfa*, azaz a kezdőállapotból növesszünk egy fát a szomszédos állapotok hozzávételével, amíg célállapotot nem találunk. Ha ügyesek vagyunk, optimális is lesz.

Vigyázat: a keresőfa *nem* azonos a feladat állapotterével! Pl. az állapottér nem is biztosan fa, amely esetben a keresőfa nőhet végtelenre is, akkor is, ha az állapottér véges.

A keresőfa csúcsaiban a következő mezőket tároljuk: szülő, állapot, cselekvés (ami a szülőből ide vezetett), útköltség (eddigi költség a kezdőállapotból), mélység (a kezdőállapoté nulla).

```
fakeresés
1  perem = { újcsúcs(kezdőállapot) }
2  while perem.nemüres()
3      csúcs = perem.elsőkivesz()
4      if csúcs.célállapot() return csúcs
5      perem.beszúr(csúcs.kiterjeszt())
6  return failure
```

A `csúcs.kiterjeszt()` metódus létrehozza a csúcsból elérhető összes állapothoz tartozó keresőfa csúcsot, a mezőket megfelelően inicializálva.

A `perem` egy prioritási sor, ez definiálja a bejárási stratégiát! Közelebbről a `perem.elsőkivesz` által feltételezett rendezést a csúcsok felett definiálhatjuk sokféleképpen, és ez adja meg a stratégiát (l. később).

A hatékonyságot növelhetjük, ha okosabban szúrunk be minden új csúcsot a perembe (5. sor) abban az esetben, ha már a peremben található ugyanazzal az állapottal egy másik csúcs. Ekkor, ha az új csúcs költsége kisebb, lecseréljük a régi csúcsot az újra, különben nem tesszük be az újat. Ezt azért tehetjük meg, mert a nagyobb költségű utat tárolni felesleges, mert a teljes költség már csak a további élektől függ.

3.4. Algoritmusok vizsgálata: teljesség, optimalitás, komplexitás

Egy adott konkrét keresési stratégia (perem prioritási sor implementáció) elemzésekor a következő tulajdonságokat fogjuk vizsgálni:

Egy algoritmus *teljes* akkor és csak akkor, ha minden esetben, amikor létezik véges számú állapot érintésével elérhető célállapot, az algoritmus meg is talál egyet.

Egy algoritmus *optimális* akkor és csak akkor, ha teljes, és minden megtalált célállapot optimális költségű.

Az idő- és memóriaigényt *nem* az állapottér méretének függvényében vizsgáljuk, hanem a speciális alkalmazásunkra (MI) szabva a következő paraméterek függvényében: b: szomszédok maximális száma, m: keresőfa maximális mélysége, d: a legkisebb mélységű célállapot mélysége a keresőfában. m és d lehet megszámlálhatóan végtelen!

3.5. Szélességi keresés

FIFO (first in first out) perem. Bizonyítsuk be, hogy

- Teljes, minden véges számú állapot érintésével elérhető állapotot véges időben elér.
- Általában nem optimális, de akkor pl. igen, ha a költség a mélység nem csökkenő függvénye.
- időigény = tárigény = $O(b^{d+1})$

A komplexitás exponenciális, tehát nem várható, hogy skálázódik: nagyon kis mélységek jönnek szóba, $d = 10$ körül. A memória egyébként előbb fogy el.

3.6. Mélységi keresés

LIFO (last in first out) perem. Bizonyítsuk be, hogy

- Teljes, ha a keresési fa véges mélységű (azaz véges, hiszen b véges). Egyébként nem.
- Nem optimális.
- időigény: a legrosszabb eset $O(b^m)$ (nagyon rossz, sőt, lehet végtelen), tárigény: legrosszabb esetben $O(bm)$ (ez viszont biztató, mert legalább nem exponenciális).

3.7. Iteratívan mélyülő keresés

Mélységi keresések sorozata 1, 2, 3, stb., mélységre korlátozva, amíg célállapotot találunk. Bizonyítsuk be, hogy

- Teljesség és optimalitás a szélességi kereséssel egyezik meg.
- időigény = $O(b^d)$ (jobb, mint a szélességi, bár kis b esetén ténylegesen nem feltétlenül jobb), tárigény = $O(bd)$ (jobb, mint a mélységi!).

Elsőre meglepő, de igaz, hogy annak ellenére, hogy az első szinteket újra és újra bejárjuk, mégis javítunk.

Ez a legjobb informálatlan (vak) kereső.

3.8. Egyenletes költségű keresés

A peremben a rendezés költség alapú: először a legkisebb útköltségű csúcsot terjesztjük ki. Bizonyítsuk be, hogy

- Teljes és optimális, ha minden él költsége $\geq \epsilon > 0$.
- (Az idő- és tárigény nagyban függ a költségfüggvénytől, nem tárgyaljuk.)

3.9. Ha nem fa az állapottér: gráfkeresés

Ha a kezdőállapotból több út is vezet egy állapotba, akkor a fakeresés végtelen ciklusba eshet, de legalább is a hatékonysága drasztikusan csökkenhet. Másrészt világos, hogy elég is a legjobb utat tárolni minden állapothoz.

Hogyan kerüljük el az ugyanazon állapotba vezető redundáns utak tárolását? *Zárt halmaz:* tárolni kell nem csak a peremet, de a peremből már egyszer kivett, kiterjesztett csúcsokat is. (A perem egy másik elnevezése *nyílt halmaz*)

```
gráfkeresés
1  perem = { újcsúcs(kezdőállapot) }
2  zárt = {}
3  while perem.nemüres()
4      csúcs = perem.elsőkivesz()
5      if csúcs.célállapot() return csúcs
6      zárt.hozzáad(csúcs)
7      perem.beszúr(csúcs.kiterjeszt() - zárt)
8  return failure
```

A perembe helyezés előtt minden csúcsot letesztelünk, hogy a zárt halmazban van-e. Ha igen, nem tesszük a perembe. Másrészt minden peremből kivett csúcsot a zárt halmazba teszünk. Így minden állapothoz a legelső megtalált út lesz tárolva.

A hatékonyságot itt is (mint a fakeresésnél) növelhetjük: ha már a peremben található ugyanazzal az állapottal egy másik csúcs, akkor, ha az új csúcs költsége kisebb, lecseréljük a régi csúcsot az újra, különben nem tesszük be az újat.

Probléma: Mi van, ha egy adott állapothoz a később megtalált út a jobb? Vigyázat: a „megtalált" úton azt az utat értjük, amelyiket a zárt halmazba került csúcs határoz meg. A peremben tárolt út még keresés alatt van, ideiglenes.

- Egyenletes költségű keresésnél bizonyíthatóan az első megtalált útnál nincs jobb, ha minden él költsége nemnegatív. Ez ugyanis éppen a Dijkstra algoritmus az állapottérre alkalmazva. (Viszont a teljességhez továbbra is kell, hogy minden költség $\geq \epsilon > 0$ (nem csak nemnegatív), mert lehetnek végtelen állapotterek.)

- Mélységi keresésnél nem biztos, hogy az első megtalált út a legjobb, ekkor át kell linkelni a zárt halmazban tárolt csúcsot a jobb út felé. De a mélységi keresés itt már nem annyira vonzó, mert a zárt halmaz miatt sok memória kellhet neki.

4. fejezet

Informált keresés, A* algoritmus

Eddig csak azt tudtuk, honnan jöttünk (költség), de arról *semmit*, hogy hová megyünk (ezért vak keresés a neve), más szóval minden szomszéd egyformán jött szóba.

Heurisztika: minden állapotból megbecsüli, hogy mekkora az optimális út költsége az adott állapotból egy célállapotba: tehát értelmesebben tudunk következő szomszédot választani. Pl. légvonalbeli távolság a célig a térképen egy útvonal-tervezési problémához jó heurisztika.

Jelölések: $h(n)$: optimális költség közelítése n állapotból a legközelebbi célállapotba, $g(n)$: tényleges költség a kezdőállapotból n-be, más szóval az alkalmazott kereső algoritmus $g(n)$ költséggel jutott az éppen vizsgált n állapotba.

4.1. Mohó legjobb-először

A peremben a rendezést $h()$ alapján végezzük: a legkisebb értékű csúcsot vesszük ki. Hasonló a mélységi kereséshez.

Ha csak annyit teszünk fel, hogy $h(n) = 0$ ha n célállapot, bizonyítsuk be, hogy fakeresés esetén

- Teljes, de csak akkor, ha a keresési fa véges mélységű.
- Nem optimális.
- időigény = tárigény = $O(b^m)$

Gráfkeresésnél az optimalitás hiánya miatt az első megtalált út nem mindig a legjobb, tehát át kell linkelni a zárt halmaz elemeit.

A legrosszabb eset nagyon rossz, de jó $h()$-val javítható.

4.2. A*

A peremben a rendezést $f() = h() + g()$ alapján végezzük: a legkisebb értékű csúcsot vesszük ki. $f()$ a teljes út költségét becsüli a kezdőállapotból a célba az adott állapoton keresztül.

Ha $h() \equiv 0$ (tehát $f() \equiv g()$), és gráfkeresést alkalmazunk, akkor a Dijkstra algoritmust kapjuk.

Teljesség és optimalitás

Egy $h()$ heurisztika *elfogadható (vagy megengedhető)*, ha nem ad nagyobb értéket mint a tényleges optimális érték (azaz nem becsüli túl a költséget).

Fakeresést feltételezve ha $h()$ elfogadható, és a keresési fa véges, akkor az A* optimális (a végesség a teljességhez kell).

Egy $h()$ heurisztika *konzisztens (vagy monoton)*, ha $h(n) \leq c(n, a, n') + h(n')$ minden n-re és n minden n' szomszédjára.

Gráfkeresést feltételezve ha $h()$ konzisztens és az állapottér véges, akkor az A* optimális (a végesség a teljességhez kell).

Hatékonyság

Az A* optimálisan hatékony, hiszen csak azokat a csúcsokat terjeszti ki, amelyekre $f() < C^*$ (C^* az optimális költség), ezeket viszont minden optimális algoritmusnak ki kell terjeszteni.

A tárigény általában exponenciális, de nagyon függ a $h()$ minőségétől, pl. ha $h() = h^*()$ ($h^*()$ a tényleges hátralevő költség), akkor konstans... Az időigény szintén erősen függ a $h()$-tól.

4.3. Egyszerűsített memóriakorlátozott A*

A memória véges. Próbáljuk az egész memóriát használni, és kezeljük, ha elfogy.

Ötlet: futtassuk az A*-ot, amíg van memória. Ha elfogyott, akkor

- töröljük a legrosszabb levelet a keresőfában (ha minden csúcs költsége u.az, akkor a legrégebbit)
- a törölt csúcs szülőjében jegyezzük meg az innen elérhető ismert legjobb költséget (így visszatérhetünk ide, és még egyszer kiterjeszthetjük, ha minden többi útról később kiderül, hogy rosszabb).

Maga az algoritmus viszonylag komplex, de az ötlet alapján rekonstruálható.

Probléma, hogy sok hasonló költségű alternatív út esetén ugrálhat az utak között, azaz folyton eldobja és újraépíti őket, ha egyszerre nem férnek a memóriába.

Ha az optimális célállapotba vezető út tárolása maga is több memóriát igényel mint amennyi van, akkor nincs mit tenni.

4.4. Heurisztikus függvények előállítása

Nagyon fontos a jó minőségű heurisztika, lehetőleg elfogadható, sőt konzisztens. Hogyan kaphatunk ilyeneket?

Relaxált probléma optimális megoldása = $h()$

Relaxáció: feltételek elhagyása. pl. 8-kirakóhoz heurisztikák

- h_1: rossz helyen lévő számok (ábrán: $h_1() = 8$)
- h_2: Manhattan távolság számonként (ábrán: $h_2() = 18$)
- ábrán $h_{opt}() = 26$

7	2	4
5		6
8	3	1

h_1-nél relaxáció: minden szám egyből a helyére tolható.

h_2-nél relaxáció: minden szám a szomszédba tolható, akkor is, ha nem üres a hely.

Vegyük észre, hogy $\forall n : h_1(n) \leq h_2(n)$, azaz h_2 *dominálja* h_1-et.

Belátható, hogy relaxált probléma optimális költsége $\leq$ az eredeti probléma optimális költségénél, mivel az eredeti probléma állapottere része

a relaxáltnak. Tehát elfogadható heurisztikát kapunk.

Belátható, hogy mivel a heurisztika a probléma egy relaxációjának tényleges költsége, ezért konzisztens is.

Relaxálás automatizálása

Ha formális kifejezés adja meg a lehetséges lépéseket a probléma állapotaiból, akkor automatikusan elhagyhatunk feltételeket, pl. 8-kirakó esetében formálisan

(a) egy számot csak *szomszédos* pozícióba lehet mozgatni, és (b) egy számot csak *üres* pozícióba lehet mozgatni.

Ha elhagyom (a)-t és (b)-t, megkapom h_1-et. Ha elhagyom (b)-t, megkapom h_2-t.

Több heurisztika kombinálása

Vehetjük a $h(n) = \max(h_1(n), \ldots, h_k(n))$ heurisztikát, ahol igaz az, hogy ha $\forall i : h_i$ konzisztens, akkor h is konzisztens.

Mintaadatbázisok

Definiáljunk részproblémát, és számoljuk ki az egzakt költséget: ez a heurisztika. Pl. 8-kirakóban az 1,2,3,4 számok kirakása (mivel ez relaxáció is, konzisztens heurisztika lesz).

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

Több ilyen részproblémát is kiszámolhatunk, és vehetjük a maximális költségűt. A több részprobléma megoldásait adatbázisban tárolhatjuk.

Független részproblémák

Szerencsés esetben a részproblémák egymástól független költségkomponensekkel rendelkeznek. Ezeket aztán össze lehet adni (maximum helyett).

Pl. 8-kirakóban az 1,2,3,4 részprobléma esetében azt tehetjük, hogy a részprobléma költségébe csak azokat a mozgatásokat számolom bele, amelyek az 1,2,3 vagy a 4 számokat mozgatták. Ha u.ezt teszem az 5,6,7,8 számokkal, a két költség összege konzisztens heurisztikát eredményez (gondoljuk meg!).

5. fejezet

Lokális keresés, optimalizálás

5.1. A globális optimalizálás problémája

Az állapottérben való kereséskor egy optimális célállapotot és a hozzá vezető utat kerestük, és a költség az út függvénye volt.

Másfajta keresés: a költség legyen az állapot függvénye (az út ekkor lényegtelen). Ekkor *globális optimalizálási problémáról* beszélünk. A modell a következő:

- lehetséges állapotok halmaza (state space)
- (kezdőállapot(ok) lehet(nek), de ez nem része a probléma definíciójának mert az állapotok értékelése nem függ a kezdőállapottól)
- lehetséges operátorok (hasonló a cselekvésekhez) halmaza és egy állapotátmenet függvény, amely minden állapothoz hozzárendel egy (operátor,állapot) típusú rendezett párokból álló halmazt
- célfüggvény (objective function): állapotok f értékelő függvénye, amely minden lehetséges állapothoz valós értéket rendel

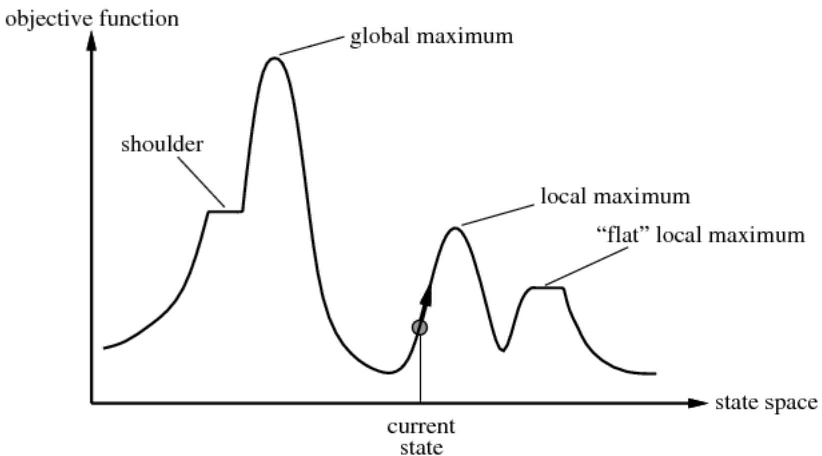
- célállapotok halmaza (lehetséges állapotok részhalmaza), ami nem más, mint a globális maximumhelyek halmaza: $\{x|f(x) = \max_{\{y \text{ egy állapot}\}} f(y)\}$

Az állapotok és operátorok által definiált gráfot itt optimalizálási tájnak hívjuk (landscape), amelyben hegycsúcsok, völgyek, hegygerincek, stb., vannak. Vigyázat: egy sokdimenziós táj nem intuitív!

Célállapotként néha a minimumhelyeket definiáljuk, szokás kérdése.

Az x állapot lokális maximumhely akkor és csak akkor, ha $f(x) = max_{\{y \ x \text{ szomszédja}\}} f(y)$, ahol x szomszédai az x-ből a lehetséges operátorokkal elérhető állapotok.

Egy globális maximumhely (azaz célállapot) egyben lokális maximumhely is.



5.2. Lokális keresés

Az optimalizálási probléma egy mohó megközelítése. Általában csak egy aktuális állapotot tárolunk.

5.2.1. Hegymászó keresés

```
hegymászó
1  x = véletlen állapot
2  repeat
3     szomszéd = x egy max értékű szomszédja
4     if f(szomszéd) <= f(x) return x
5     else x = szomszéd
```

A legmeredekebb emelkedőn elindulva a legelső lokális maximumba beragad.

Példa: 8 királynő probléma: sakktáblára tegyünk 8 királynőt úgy, hogy ne üssék egymást.

Állapotok: $\{1, 2, 3, 4, 5, 6, 7, 8\}^8$; mind a 8 sorra megmondja, melyik oszlopba tegyük a királynőt.

Lehetséges operátorok: egy királynő soron belüli elmozgatása: $8 \times 7 = 56$ operátor minden állapotból.

Célfüggvény: f legyen a párok száma, amelyek ütik egymást. Minden x célállapotra $f(x) = 0$, egyébként pedig $f(x) \geq 0$ (tehát itt minimalizálási problémáról van szó).

A hegymászó véletlen állapotból indulva az esetek 86%-ában nem talál célállapotot, mert egy lokális optimum (ami gyakran platón van) megállítja. Pl. ha megengedjük az oldallépést (elfogadjuk az azonos értékű szomszédot) max. 100-szor, akkor 94% siker! (tehát itt sok a plató)

5.2.2. Hegymászó javításai

Sztochasztikus hegymászó: véletlen szomszéd azok közül, amelyek jobbak (nem feltétlenül a legjobb); lassabb, de ritkábban akad el.

Első választásos hegymászó: vegyük az első szomszédot, ami jobb. Ez a sztochasztikus egy implementációja, és akkor hatékonyabb, ha nagyon

sok szomszéd van.

Véletlen újraindított hegymászó: ha nem sikeres, indítsuk újra véletlen kezdőpontból. Ez nagyon jó algoritmus! (pl. 3 millió királynő megoldása egy perc alatt) De a keresési tér struktúrájától nagyban függ.

5.2.3. Szimulált hűtés

Újraindítás helyett a hegymászó itt lokális optimumokból ki tud ugrani. Más szóval elfogadunk lefelé lépést is néha: minél kisebb a „hőmérséklet”, annál kisebb valószínűséggel. A hőmérséklet folyamatosan csökken.

```
szimulált hűtés
 1 x = véletlen állapot;
 2 T = hűtésiterv[0]; t = 1
 3 repeat
 4    szomszéd = x egy véletlen szomszédja
 5    d = f(szomszéd)-f(x)
 6    if d > 0  x = szomszéd
 7    else x = szomszéd exp(d/T) valószínűséggel
 8    T = hűtésiterv[t]; t = t+1
 9 until T == 0
10 return x
```

A hűtési terv fontos, de problémafüggő, nincs általános recept. Pl. $T_{t+1} = \alpha T_t, \alpha = 0.95$.

5.3. Populáció alapú lokális keresés

5.3.1. Nyaláb keresés

Eddig csak egy aktuális állapotot használtunk. Ha több van egyszerre, az is segíthet kiugrani a lokális optimumokból. A nyaláb keresés a legegyszerűbb példa.

```
nyaláb keresés
1  x[] = K véletlen állapot
2  repeat
2     generáljuk x[] elemeinek összes szomszédját
3     x[] = K legjobb az összes szomszéd közül
4     if x[i] célállapot valamely i-re return x[i]
```

Megállási feltételt is adhatunk, pl. iterációk száma.

Nem ugyanaz, mint K db független hegymászó: jobban fókuszál, de gyorsabban be is ragad.

Vannak javításai, mint a hegymászónak, pl. sztochasztikus, stb.

5.3.2. Evolúciós algoritmusok

```
evolúciós algoritmus séma
1  x[] = K véletlen állapot
2  repeat
2     szülők választása
3     új állapotok generálása a szülőkből
           rekombinációval
4     új állapotok mutációja
5     K állapot kiválasztása a régiek és az
           újak uniójából
6  until megállási feltétel
```

Nyaláb keresés: hegymászó általánosítása volt populációval (a K állapot) és szelekcióval.

Evolúciós algoritmus: nyaláb általánosítása szexuális operátorokkal (a tárgyalás itt a könyvtől eltér!).

Az evolúciós algoritmusok területén szokás evolúciós terminológiát használni. Kisszótár:

állapot = megoldás = egyed (individual)
mutáció = operátor = cselekvés
rekombináció = keresztezés (crossover) = kétváltozós operátor: $A \times A \to A$
fitnessz = célfüggvény érték
szülő = aktuális állapot(ok közül választott állapot) (2. sor)
utód = új állapot (3., 4. sor)
populáció = aktuális állapotok halmaza, azaz x
generáció = a populáció egy adott iterációban
túlélők = megoldások az új populációban (5. sor)

Az új komponens tehát a rekombináció. Ezen kívül a szülők és túlélők kiválasztásának is sok módja van.

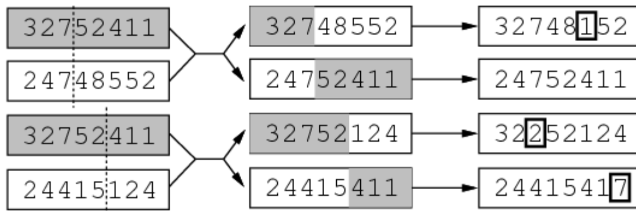
5.3.3. Genetikus algoritmus (genetic algorithm, GA)

Amerikai iskola. Állapottér: véges betűkészletű (gyakran bináris (0,1-ből álló)) stringek. Pl. 8 királynő ilyen.

Szülő szelekció: választunk pl. K/2 db párt pl. a fitnesszel arányosan vagy véletlenszerűen.

Rekombináció: véletlen keresztezési pontnál vágás és csere.

Mutáció: pl. egy betű megváltoztatása. Pl. a 8 királynő problémához a hegymászó ezt használta.



Túlélő szelekció: pl. a K legjobbat vesszük, vagy a fitnesszel arányos valószínűséggel választunk, vagy K véletlen párt választunk, és mindegyikből a jobbik túlél (tournament), stb.

Itt sincs biztos recept, a sok tervezési döntés hatása feladatfüggő: hány lokális optimum van, dimenziószám, optimumok eloszlása, stb.

5.3.4. Folytonos terek

Eddig diszkrét keresési tereket vizsgáltunk (kombinatorikus problémák). Ha az állapottér folytonos (azaz a célfüggvény $\mathbb{R}^d \to \mathbb{R}$ típusú), akkor végtelen szomszédos állapot van. A hegymászó itt is működik:

hegymászó: legmeredekebb lejtő/emelkedő módszere. Ha deriválható a célfüggvény, akkor meghatározzuk a legmeredekebb irányt (ez éppen a derivált iránya), és egy ϵ hosszú lépést teszünk. ϵ lehet konstans vagy adaptív, lehet véletlen szórása is. Ha ϵ túl kicsi, lassú, ha túl nagy, túllőhetünk a célon.

sztochasztikus hegymászó: véletlen irányt választunk, ha javít, elfogadjuk.

5.3.5. Differenciál evolúció

Populáció alapú keresés folytonos problémákra, de általánosítható diszkrét esetre is.

A populáció: $x_1, \ldots, x_K \in \mathbb{R}^d$.

Szülő választás: minden egyed szülő.

Rekombináció: a populáció minden x_i-eleméhez ($i = 1, \ldots, K$) válasszunk három véletlen egyedet: $x_{r_1}, x_{r_2}, x_{r_3}$. Legyen $v = x_{r_1} + F \cdot (x_{r_2} - x_{r_3}) + \lambda \cdot (y_g - x_{r_1})$, ahol y_g az eddig megtalált legjobb megoldás. Megjegyzés: v nem függ x_i-től! Ezután v-t keresztezzük x_i-vel; legyen az eredmény az u vektor, ahol $u_j = x_{ij}$ CR valószínűséggel, $u_j = v_j$ egyébként (ahol x_{ij} x_i j. eleme, v_j pedig v j. eleme, $j = 1, \ldots, d$). A paraméterek pl. $F \approx 0.8$, $CR \approx 0.9$, $\lambda \approx 0.9$. Persze a paraméterek optimális beállítása itt is nagyban feladatfüggő.

Mutáció: nincs.

Túlélők: u-val helyettesítjük x_i-t, ha $f(u) > f(x_i)$ (feltéve, hogy maximumot keresünk).

Lényeg: a fenti rekombinációs operátor olyan, hogy a populáció kiterjedését és alakját tiszteletben tartja, mivel mintavételezésen alapul, pl. hosszú keskeny völgyekben inkább hosszanti irányban explorál, kicsi koncentrált populációban nem dobál messzire megoldásokat, stb.

6. fejezet

Korlátozás kielégítési feladat

A feladat az állapottérrel adott keresési problémák és az optimalizálási problémák jellemzőit ötvözi. Az állapotok és célállapotok speciális alakúak.

Lehetséges állapotok halmaza: $\mathcal{D} = D_1 \times \cdots \times D_n$, ahol D_i az i. változó lehetséges értékei, azaz a feladat állapotai az n db változó lehetséges értékkombinációi.

Célállapotok: a megengedett állapotok, amelyek definíciója a következő: adottak $C_1, \ldots, C_m$ *korlátozások*, $C_i \subseteq \mathcal{D}$. A *megengedett* vagy *konzisztens* állapotok halmaza a $C_1 \cap \ldots \cap C_m$ (ezek minden korlátozást kielégítenek).

Ugyanakkor, akár az optimalizálási problémáknál, az út a megoldásig lényegtelen, és gyakran célfüggvény is értelmezve van az állapotok felett, amely esetben egy optimális célállapot megtalálása a cél.

Gyakran egy C_i korlátozás egy változóra vagy változópárra fejez ki megszorítást.

Pl. gráfszínezési probléma: adott $G(V, E)$ gráf, $n = |V|$. A változók a gráf pontjai, n változó van. Az i pont lehetséges színeinek halmaza D_i, és $D_1 = \ldots = D_n$. Minden $e \in E$ élhez rendelünk egy C_e korlátozást,

amely azokat a színezéseket tartalmazza (engedi meg), amelyekben az e él két végpontja különböző színű.

Kényszergráfok: a gráfszínezésnél a korlátozások változópárokon működtek, így a változók felett gráfot definiálnak: ez a kényszergráf. Belátható, hogy bármely kettőnél több változót érintő korlátozás kifejezhető változópárokon értelmezett korlátozásként, ha megengedjük segédváltozók bevezetését. Kényszergráf tehát minden feladathoz adható, és a keresés komplexitásának a vizsgálatakor ez hasznos, mert a gráf tulajdonságainak a függvényében adható meg a komplexitás (nem tárgyaljuk részletesen).

6.1. Inkrementális kereső algoritmusok

Az első lehetséges megközelítés, ha állapottérbeli keresési problémaként formalizáljuk a korlátozás kielégítési problémát. Ekkor az állapottér a következő:

- Minden változóhoz felveszünk egy új „ismeretlen" értéket (jelölje „?"), és legyen a kezdeti állapot a $(?, \dots, ?)$.
- Az állapotátmenet függvény rendelje hozzá minden állapothoz azon állapotokat, amelyekben eggyel kevesebb „?" van, és amelyek megengedettek.
- A költség minden állapotátmenetre legyen konstans.

Az állapotátmeneteket lehet sokkal hatékonyabban definiálni: minden állapotban elegendő pontosan egy változó („?") kiterjesztését megengedni, ez sokkal kisebb fát eredményez. A választott változótól függhet a hatékonyság.

A keresés végrehajtására a korábban látott összes informálatlan algoritmus alkalmazható. A mélységi itt elég jó, mert a keresőfa mélysége

(a változók száma) kicsi és nem fogyaszt memóriát (backtrack algoritmus).

Viszont informálatlan keresésnél sokkal jobb lehet az informált. Mit tehetünk? Próbáljuk a nehezen kielégíthető változókkal kezdeni. A lehetséges operátorok prioritási sorrendjét határozzuk meg pl. így:

1. Válasszuk azt a változót, amelyikhez a legkevesebb megengedett érték maradt.
2. Ha ez nem egyértelmű, akkor azt, amelyre a legtöbb korlátozás vonatkozik.
3. A választott változó megengedett értékeiből kezdjük azzal, amelyik a legkevésbé korlátozza a következő lépések lehetséges számát.

Ez jelentős javulást eredményez. Számos más technika is van, amire nem térünk ki.

6.2. Optimalizáló algoritmusok, lokális keresők

Egy másik lehetséges megközelítés optimalizálási problémát definiálni. Ekkor

- A célfüggvényt definiáljuk pl. úgy, hogy legyen a megsértett korlátozások száma. (minimalizálni kell, minimuma a nulla). Ha eleve tartozott célfüggvény a feladathoz, akkor össze kell kombinálni vele (a korlátozások büntető tagokat jelentenek).
- Az operátorokat definiálhatjuk sokféleképpen, pl. valamely változó értékének a megváltoztatásaként.

Itt pedig az összes korábban látott lokális kereső, genetikus algoritmus, stb. alkalmazható. Pl. a 8-királynő példánk is ilyen volt.

6.3. Záró megjegyzések

A lokális keresők sokszor nagyon gyorsak és sikeresek de nem mindig találnak megoldást (vagy optimális megoldást).

Az állapottér reprezentációban dolgozó inkrementális módszerek általában teljesek (vagy optimálisak), de nem skálázódnak nagy problémákra.

A keresési és optimalizálási probléma az egész MI keretét adja, ennek speciális alkalmazásai jönnek leginkább.

7. fejezet

Játékok

A hagyományos MI egyik fő érdeklődési területe. Elsősorban sakk motiválta (1769, Mechanical Turk (Kempelen Farkas)); 1950-től majdnem minden kutató (Shannon, Weiner, Turing, stb.)).

A go játék sokáig nem érte el az emberi szintet (formálisan hasonló, valójában nagyon különbözik a sakktól), de 2016-ban AlphaGo! De még mindig érdekes terület, pl. véletlen szerepe vagy több mint két játékos, stb.

7.1. Kétszemélyes, lépésváltásos, determinisztikus, zéró összegű játék

Játékelméletbe nem megyünk bele, csak ezt a speciális feladatot nézzük.

- lehetséges állapotok halmaza (legális játékállások)
- egy kezdőállapot
- lehetséges cselekvések halmaza és egy állapotátmenet függvény,

amely minden állapothoz hozzárendel egy (cselekvés,állapot) típusú rendezett párokból álló halmazt

- célállapotok (vagy végállapotok) halmaza (lehetséges állapotok részhalmaza)
- hasznosságfüggvény, amely minden lehetséges célállapothoz hasznosságot rendel.

Eddig hasonló az állapottérben való kereséshez és a korlátozás kielégítési feladatra is emlékeztet, amennyiben itt is azon célállapotokat keressük, amelyek optimális hasznosságúak, és az út nem része a megoldásnak.

De: itt *két ágens* van, *felváltva lépnek* (azaz alkalmaznak operátort), és a hasznosságfüggvényt az egyik maximalizálni akarja (MAX játékos), a másik minimalizálni (MIN játékos). Konvenció szerint MAX kezd.

Az első célállapot elérésekor a játéknak definíció szerint vége.

Zéró összegű játék: Modellünkben MIN minimalizálja a hasznosságot, ami ugyanaz, mint maximalizálni a negatív hasznosságot. Tehát egy ekvivalens problémadefiníció lenne a *negamax* formalizmus, amikor mindkét játékos maximalizálja a saját hasznosságfüggvényét, de az egyik hasznosságfüggvény éppen a másik -1-szerese. A negamax formalizmusban tehát a két játékos nyereségének az összege a végállapotban mindig nulla, innen a „zéró összegű” elnevezés.

A fenti struktúra neve *játékgráf* (általában nem fa).

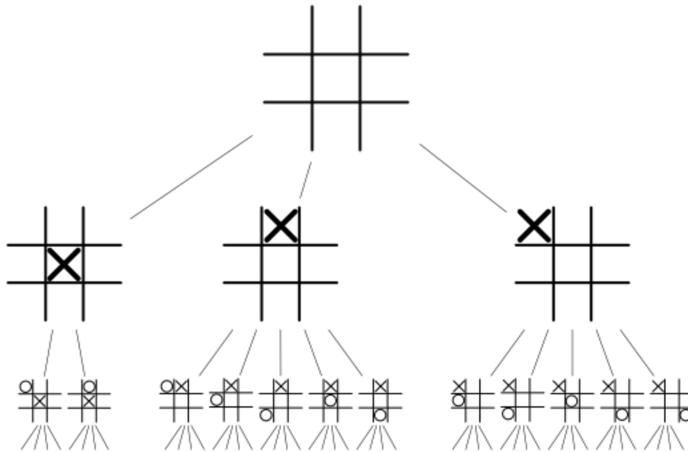
7.1.1. Pl. 3x3-as amőba (tic-tac-toe)

Kezdőállapot az üres tábla, MAX játékos az X jelet használja. Célállapotok, amikor három azonos jel van egy sorban, oszlopban vagy átlóban, vagy tele van a tábla. (csak egyféle jelből jöhet ki ilyen egy állapotban, mert a másik jel már nem léphet!) A hasznossági függvény

értéke 1, ha X jelekből jön ki a célállapot, -1, ha körből, 0, ha egyikből sem.

Az illusztráció a játékgráf első három szintjét mutatja (ami még fa).

255168 lehetséges játék, 138 lehetséges kimenetel (célállapot).



7.1.2. Minimax algoritmus

Tegyük fel, hogy mindkét játékos a teljes játékgráfot ismeri, tetszőlegesen komplex számításokat tud elvégezni, és nem hibázik (nagyon erős feltevések). Ezt szokás a *tökéletes racionalitás hipotézisének* nevezni.

Egy *stratégia* minden állapotra meghatározza, hogy melyik lépést kell választani. Belátható, hogy mindkét játékos számára a lehető legjobb stratégiát a minimax algoritmus alapján lehet megvalósítani tökéletes racionalitás esetén.

A minimax algoritmus a következő értékeket számolja ki minden n

csúcsra:

$$\text{minimax}(n) = \begin{cases} \text{hasznosság}(n) & \text{ha végállapot} \\ \max_{\{a\ n\ \text{szomszédja}\}} \text{minimax}(a) & \text{ha MAX jön} \\ \min_{\{a\ n\ \text{szomszédja}\}} \text{minimax}(a) & \text{ha MIN jön} \end{cases}$$

```
maxÉrték(n)
1 if végállapot(n) return hasznosság(n)
2 max = -végtelen
3 for a in n szomszédai
4    max = max(max, minÉrték(a))
5 return max
```

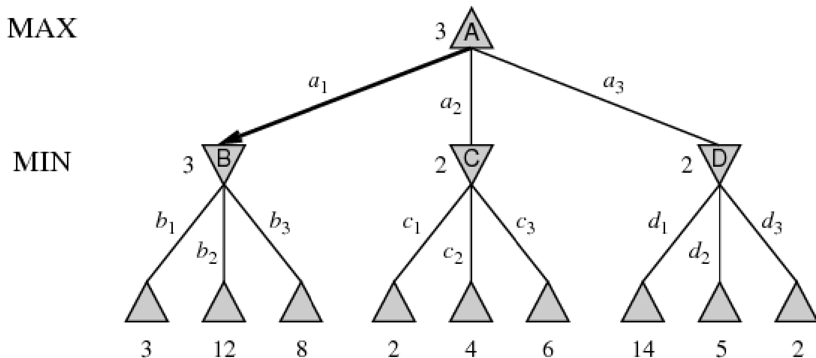
```
minÉrték(n)
1 if végállapot(n) return hasznosság(n)
2 min = +végtelen
3 for a in n szomszédai
4    min = min(min, maxÉrték(a))
5 return min
```

Összes érték kiszámítása maxÉrték(kezdőállapot) hívásával.

Ha a jatékgráfban van kör, akkor nem terminál (fakeresés), de a gyakorlatban ez nem probléma: csak fix mélységig futtatjuk (l. később), ill. a játékszabályok gyakran kizárják a köröket a végtelenségig folytatódó játszmák megelőzésére.

Világos, hogy a minimax érték az optimális hasznosság, amit az adott állapotból egy játékos elérhet, ha az ellenfél tökéletesen racionális.

Az ábra illusztrálja a minimax értékeket egy konkrét példán.



Futtatás: a játékos tehát először kiszámítja a minimax értékeket a teljes játékfára, majd mindig a számára legjobb minimax értékű szomszédot lépi (ez határozza meg a stratégiát).

Ez csak elméleti jelentőségű, a minimax algoritmus nem skálázódik. Sakkban pl. 10^{154} csúcs a játékfában, 10^{40} különböző. A világegyetem atomjainak száma: kb. 10^{80}. Hogyan lehet minimaxot hatékonyan számolni? És/vagy közelítőleg? (Egyáltalán minimaxot kell mindig számolni?)

7.1.3. Alfa-béta vágás

Ötlet: ha tudjuk, hogy pl. MAX egy adott csúcs rekurzív kiértékelése közben már talált olyan stratégiát, amellyel ki tud kényszeríteni pl. 10 értékű hasznosságot az adott csúcsban, akkor a csúcs további kiértékelése közben nem kell vizsgálni olyan állapotokat, amelyekben MIN ki tud kényszeríteni ≤ 10 hasznosságot, hiszen tudjuk, hogy MAX sosem fogja ide engedni a játékot (hiszen már van jobb stratégiája).

Pl. a fenti példában lássuk be, hogy ha balról jobbra járjuk be a gyerekeket, akkor a C csúcs első gyereke után a többit nem kell már bejárni, vissza lehet lépni!

Algoritmus: az n paraméter mellé az alfa és béta új paramétereket is

átadjuk minÉrték-nek és maxÉrték-nek. Az alfa és béta paraméterek jelentése függvényhíváskor:

Alfa: MAX-nak már felfedeztünk egy olyan stratégiát, amely alfa hasznosságot biztosít egy olyan állapotból indulva, ami a keresőfában az n állapotból a gyökér felé vezető úton van.

Béta: MIN-nek már felfedeztünk egy olyan stratégiát, amely béta hasznosságot biztosít egy olyan állapotból indulva, ami a keresőfában az n állapotból a gyökér felé vezető úton van.

Majdnem azonos a minimax-szal, csak propagáljuk alfát és bétát, és vágunk.

A teljes keresőfa kiszámítása a maxÉrték(kezdőállapot, -végtelen, +végtelen) hívással.

(Megj: a könyv ábrája nem az alfa-béta értékeket illusztrálja!)

```
maxÉrték(n, alfa, béta)
1 if végállapot(n) return hasznosság(n)
2 max = -végtelen
3 for a in n szomszédai
4    max = max(max, minÉrték(a, alfa, béta))
5    if max>=béta return max // vágás
6    alfa = max(max, alfa)
7 return max
```

```
minÉrték(n, alfa, béta)
1 if végállapot(n) return hasznosság(n)
2 min = +végtelen
3 for a in n szomszédai
4    min = min(min, maxÉrték(a, alfa, béta))
5    if alfa>=min return min // vágás
6    béta = min(min, béta)
7 return min
```

Alfa-béta néhány tulajdonsága

A szomszédok bejárási sorrendje nagyon fontos. Ha mindig a legjobb lépést vesszük (optimális eset), akkor $O(b^m)$ helyett $O(b^{m/2})$ lesz az időigény: tehát kétszer olyan mély fát járunk be ugyanazon idő alatt. (elhisszük, bizonyítás nem kell).

Véletlen bejárással $O(b^{3m/4})$ lesz a komplexitás (ezt is elhisszük, bizonyítás nem kell).

A gyakorlatban használhatunk rendezési heurisztikákat, amik sokszor közel kerülnek az optimális esethez.

Gráf keresés: játékgráfban is sok lehet az ismétlődő állapot. Eddig a fakeresés analógiájára nem tároltuk a már látott állapotokat. Ha tároljuk (mint a zárt halmazban) akkor az alfa-béta algoritmus is jelentősen felgyorsul, ill. nem ragad végtelen ciklusba. Itt a hagyományos elnevezése a zárt halmaznak *transzpozíciós tábla*.

7.1.4. Praktikus, valós idejű algoritmusok

Nem tudunk a végállapotokig leérni: *heurisztikus kiértékelő függvények* kellenek, amik bármely állapot minőségét jellemzik, és az alfa-béta algoritmus *korábban* kell, hogy visszalépjen, pl. fix mélységből, vagy még okosabban.

Heurisztikus értékelés

A heurisztika területspecifikus tudást ad (mint korábban az A^* algoritmusnál) az egyébként területfüggetlen kereső algoritmushoz. A heurisztikus kiértékelésre teljesülnie kell, hogy (1) a végállapotokra a pontos hasznosságértéket adja, (2) gyorsan kiszámolható, és (3) nemvégállapotokra a minimax értéket jól becslő érték (ami lehet folytonos, pl. ha a minimax érték -1, 0 vagy 1 lehet (veszít, döntetlen, győz), a

heurisztika a $[-1, 1]$ intervallumból ad értéket).

Hogyan tervezhetünk ilyet?

Lineáris jellemzőkombinációk: kiértékelés$(s) = \sum_i w_i f_i(s)$, ahol $f_i(s)$ valamilyen *jellemzője* (feature) s állapotnak, w_i súllyal. Pl. sakkban jellemző a bábufajták száma, a súlyokat pedig néha kézikönyvek is tárgyalják, pl. gyalog súlya 1, futóé 3, stb. Rengeteg jellemzőt lehet definiálni, bármilyen mintázat, kombinációk, stb., gyakran több ezer van.

Nemlineáris kombinációk: az egyes jellemzők hatása erősítheti vagy kiolthatja egymást (nem csak összeadódhat).

Tanulás! Pl. a jellemzők w súlyvektorának megtanulása. Később majd látjuk: a tanulás is egyfajta keresés (ill. optimalizálás). Rengeteg lehetőség, pl. evolúciós algoritmusokkal az egyedek a w vektor lehetséges értékei, fitnessz a nyerési esély nagysága (ezt lehet közelíteni pl. egymás elleni játszmákkal a populáción belül). Mivel valós a célfüggvény, a differenciál evolúció nem rossz ötlet.

Az AlphaGo go ágens első verziójában szintén szerepelt értékelőfüggvény, amelyet mesterséges neuronháló számolt ki a nyers táblaállásból. A tanítását emberi játszmák adatbázisán végezték.

(A megerősítéses tanulás (reinforcement learning) algoritmusával egy játékos akár önmaga ellen játszva is tanulhat.)

Keresés levágása

Hogyan használjuk ki a heurisztikus értékelést? Több lehetőség.

Fix mélység: az aktuális játékállásból pl. 5 mélységig építjük fel a keresőfát alfa-bétával, ami során az 5 mélységű állapotokat értékeljük ki heurisztikusan. Az implementáció egyszerű: u.az, csak a mélységet is követni kell, és végállapotok mellett az 5 mélységre is tesztelünk a rekurzió előtt.

Iteráltan mélyülő keresés: alfa-béta iteráltan növekvő mélységig. Ez analóg a korábbi iteráltan mélyülő kereséssel (az alfa-béta pedig a mélységi kereséssel). Ennek a legnagyobb előnye, hogy rugalmas: már nagyon gyorsan van tippünk következő lépésre, és ha van idő, akkor egyre javul. Valós időben használható tehát.

Horizont effektus: fix keresési mélységnél lehet, hogy egy fontos következmény (pl. ütés) nem lesz figyelembe véve, mert picit lejjebb van. Technikák okosabb vágásra: (1) *Egyensúlyi keresés (quiescence)*: ha egy állapot „mozgalmas" (heurisztika dönt, hogy ez mikor van, de pl. ha az úton a heurisztikus értékelés nagyon változik (pl. sok ütés)), akkor lejjebb megyünk addig, amíg „megnyugszik" az adott lépésvariáció, azaz várható, hogy ha még lejjebb mennénk, akkor viszonylag sokáig stabil lenne. (2) *Szinguláris kiterjesztés*: az olyan állapotokból, ahol van „világosan legjobb" (ezt is heurisztika dönti el) lépés, ott nem állunk meg a maximális mélységben, viszont csak a legjobb lépést terjesztjük ki. Így érdekesebb variációkat elég mélyen meg lehet nézni, mert az elágazási faktor 1 ezekben.

7.1.5. Helyzetkép

Megoldott játék: ha tudjuk, hogy mi az optimális stratégia kimenetele (mit hoz az optimális játék a játékosoknak) a kezdőállásból indulva (vagy esetleg tetszőleges állásból indulva is). Pontosabban: *Ultra gyengén megoldott*, ha a kezdőállás minimax értéke ismert (de ilyenkor nem feltétlenül létezik játszható algoritmus, gyakran csak nem-konstruktív módszerrel bizonyított). *Gyengén megoldott*, ha ultra-gyengén megoldott, és a kezdőállásból kiindulva létezik praktikus algoritmus, ami az optimális stratégiát játssza valós („értelmes") időben. *Erősen megoldott*, ha adunk egy algoritmust, ami a minimax értéket kiszámolja valós („értelmes") időben *tetszőleges* lehetséges játékálláshoz.

Megoldott játékok: *Awari*, erősen megoldva, 2002, Free University of Amsterdam, optimális játék döntetlen. *Dáma*, gyengén megoldva,

2007, J. Schaeffer et al; a keresési tér 5×10^{20}, 18 év kellett hozzá; legnagyobb megoldott játék, optimális játék döntetlen. Ezen kívül számos kisebb játék.

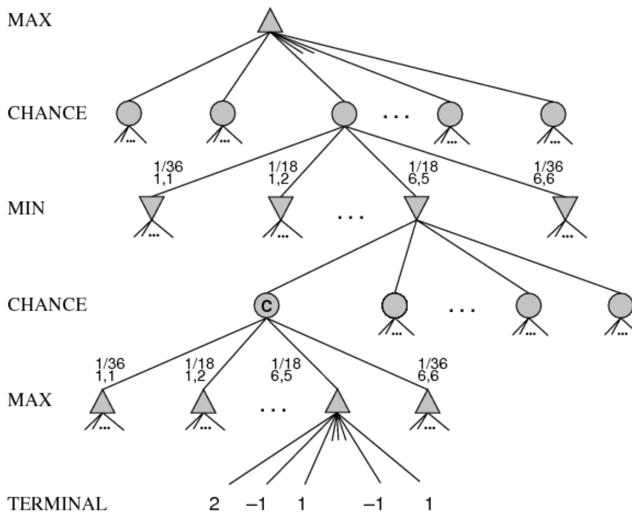
Sakk: 1996 Deep Blue-Kaszparov első játszma: Deep Blue győz. Azóta már PC-s nagymesteri szintű programok vannak. A végjáték max. 7 bábúra megoldott. Jelenleg legjobb: Stockfish 9. Erejét 3452 Élő-pontra becsülik! Minden platformon elérhető, pl. telefonra is. Emberrel már csak gyalogelőnyt adva van értelme játszania.

Go: 2016-ban AlphaGo legyőzi Lee Sedolt, 2017-ben Ke Jie-t. Sokkal nagyobb tér, brute force nem működik (sakknál azért igen...). Fontos a mintafelismerés és a stratégiák. AlphaGo: Monte Carlo fakeresés kombinálva mély neuronhálós tanulással.

7.2. Véletlent tartalmazó játékok

Sok játékban van véletlen, pl. kockadobás. Olyan eseteket nézünk, ahol az állapot továbbra is teljesen ismert, de a véletlentől is függ.

Egészítsük ki a játékfát a véletlen eseményt leíró csúcsokkal (CHANCE) (mintha a véletlen lenne a harmadik játékos). Az ábra az ostábla játékot szemlélteti, ahol két kockával dobunk, és az összeg számít (21 különböző kimenetel).



A minimax algoritmust nem tudjuk direktben alkalmazni, mert a CHANCE „játékos"-ra nem érvényes, hogy optimalizálna, csak véletlenül választ függetlenül a következményektől. Módosítás: várható-minimax (rövidítés: vmm), ami a minimax várható értéket adja meg.

$$\text{vmm}(n) = \begin{cases} \text{hasznosság}(n) & \text{ha végállapot} \\ \max_{\{a\ n \text{ szomszédja}\}} \text{vmm}(a) & \text{ha MAX jön} \\ \min_{\{a\ n \text{ szomszédja}\}} \text{vmm}(a) & \text{ha MIN jön} \\ \sum_{\{a\ n \text{ szomszédja}\}} P(a)\text{vmm}(a) & \text{ha CHANCE csúcs} \end{cases}$$

Alfa-béta is implementálható, a MIN és MAX csúcsokon változatlan formában, sőt, a CHANCE csúcsokat is lehet vágni, ha a várható értéket tudjuk korlátozni a lehetséges kimenetelek egy részhalmazának a vizsgálata után. Ez pedig csak akkor lehetséges, ha a hasznosság() függvény korlátos.

7.3. Ha nem teljes az információ

Van, hogy a játék állapota nem ismert teljesen, pl. kártyajátékok. Ez mindig egy véletlen esemény (pl. osztás) eredménye. Nem célravezető azonban egyszerűen várható értéket venni, mert nem mindegy, hogy „most még nem tudom mi lesz, de később majd meglátom” ill. „nem tudom mi lesz, és később se fogom pontosan megtudni”. Pl. a kockázat máshogy jelentkezik (nagyobb az ismeretlen állapotok esetében).

Itt lehet pl. az állapotokra vonatkozó „hiedelmek” terében keresni, stb.

8. fejezet

Logikai tudásreprezentáció

Eddigi példák tudásra: állapotok halmaza, lehetséges operátorok, ezek költségei, heurisztikák.

Mivel eddig feltettük (kivéve kártyajáték), hogy a világ (lehetséges állapotok, és állapotátmenetek) teljesen megfigyelhető, a tudás *reprezentációja* és különösen a *következtetés* nem kapott szerepet. A tanulást már érintettük a játékok heurisztikáival kapcsolatban pl.

De: már eddig is használtunk nyelvi jellegű reprezentációt az állapotok és szabályok leírásánál (pl. sakk, stb.).

A való világ állapotai sokkal komplexebbek mint egy térkép vagy egy sakktábla, és csak *részlegesen megfigyelhetők*: pl. az orvosi diagnózis esetében az aktuális állapotnak (pl. egy ember egészségi állapotának) nagyos sok lényeges jellemzője lehet, amelyek közül csak egy kis részük figyelhető meg közvetlenül, más részük pedig csak nagy költséggel vagy egyáltalán nem. (Nem is beszélve a világ még ismeretlen jellemzőiről, pl. rádióaktivitás 200 éve?) Ilyen esetben szükség lehet kellően kifejező állapotreprezentációra, és az ismeretlen jellemzők (betegség) kikövetkeztetésére szolgáló módszerekre.

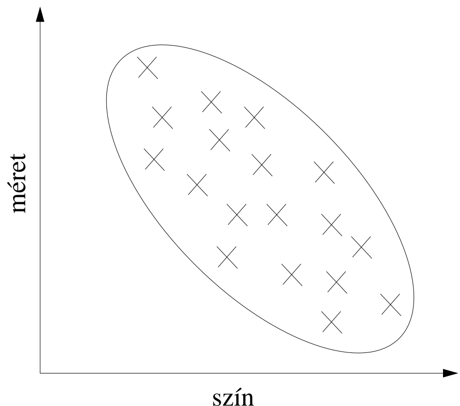
A *logikai* reprezentáció a természetes nyelvek által inspirált reprezentá-

ció. Néhány megjegyzés:

(1) A természetes nyelvvel nem keverendő össze. A természetes nyelv nem csak tudásreprezentáció, fő célja a kommunikáció, sőt nyelvi aktusok, egy kultúrába ágyazva működik. A természetes nyelvi mondatok jelentése nagyon komplex, pl. a kognitív tudomány egy fő kérdése. A logikában matematikailag definiált.

(2) A logika a bizonytalanság kérdésével nem foglalkozik általában: valamit vagy tudunk vagy nem. A fuzzy logika alkalmas folytonos igazságérték reprezentációjára, de ez nem a tudás, hanem a jelentés bizonytalansága.

(3) A nyelv fogalmai vagy más fogalmak logikai kifejezéseiből rakódnak össze, vagy „alap” fogalmak amik közvetlen érzékelésre támaszkodnak (pl. színek), vagy más fogalmak nem logikai jellegű kombinációi. Egy matematikai logika az utolsó típusú fogalmakat nem képes megragadni.



8.1. Illusztrációs példa: wumpus világ

Számítógépes játék a 70-es évekből. A feladatkörnyezet egyszerűsített definíciója:

Környezet: 4x4-es rács (szobák). A kezdőállapotban az ágens az (1,1)-en áll. Pontosan egy szobában wumpus, pontosan egy szobában arany, minden szobában 20% valószínűséggel verem (de a játék kezdete után fix). Az (1,1)-ben se wumpus se verem. Ha verembe vagy wumpusba botlunk, vesztettünk: game over.

Beavatkozások: jobbra, balra, előre egy szoba; nyíl kilő (egy nyíl van, vagy fal vagy wumpus állítja meg); arany felvesz.

Érzékelők: szag (ha wumpus van a szobában vagy a szomszédban); szellő (verem van a szomszédban); arany; „bump” (ha falba ütközünk); sikoly (wumpus elpusztult).

Kiértékelés: -1 pont: minden akció, -10: nyíl, 1000: arany.

Figyeljük meg, hogy a háttértudást belekevertük a fenti leírásba.

Tények: a szenzorok által érzékelt információ. *Szabályok*: a háttértudás (minden amit tudunk és nem közvetlen megfigyelésből jön). Tények+szabályok $\xrightarrow{\text{következtetés}}$ aktuális állapot legjobb leírása.

Az ábrán egy példa játék első néhány lépése látható. A jelek jelentése: A ágens, B szellő, OK biztonságos, P verem, S szag, V látogatott, W wumpus. A következő következtetések zajlanak le:

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

1. (1,1)-ben semmit sem érzékelünk, tehát (1,2) és (2,1) OK. Megyünk pl. a (2,1)-be.

2. szellőt érzékelünk, tehát (3,1) vagy (2,2) verem. Megyünk az (1,2)-be.

3. szagot érzünk, tehát (2,2) nem verem mert akkor éreznénk szellőt. Tehát (3,1) a verem. (2,2) nem wumpus, mert akkor (2,1)-ben is lett volna szag, tehát (1,3) a wumpus, viszont emiatt (2,2) OK.

Nem lehet mindig OK-ba lépni, kockázat van, viszont feltesszük, hogy az *érzékelés hibátlan*, és a *szabályok alól nincs kivétel*.

8.2. Logika

Alapfogalmak (informálisan), amelyek minden logikában érvényesek (mielőtt egyes logikák részleteibe mennénk).

Szintaxis: A nyelv jólformált mondatait definiálja (azaz egy formális nyelvet). Pl. a kifejezések infix nyelvében $x+y=4$ jólformált, $xy+4=$ pedig nem, viszont utóbbi pl. postfix jelölésben jólformált.

Szemantika: A nyelv mondataihoz jelentést rendel. Klasszikus logikában egy mondat jelentése mindig vagy az *igaz* vagy a *hamis* érték. A jelentést egy adott *lehetséges világban* lehet meghatározni: a lehetséges világ a mondat szimbólumaihoz „jelentést” rendel, pl. a $x+y=4$ mondatban az x és y változókhoz számokat, a $+$ jelhez egy műveletet (pl. a szorzást), stb. A lehetséges világ további elnevezései *interpretáció* ill. *modell* (bár a modell kifejezést többféle, kissé eltérő értelemben használják).

Itt a lehetséges világ, interpretáció ill. modell elnevezéseket szinonimaként, az egyes mondatok igazságától függetlenül használjuk: egy mondat bizonyos lehetséges világokban (modellekben) igaz, másokban hamis lehet.

Logikai következmény: $\alpha \models \beta$ akkor és csak akkor ha minden modellben ahol α igaz, β is igaz (α-nak logikai következménye β.) Pl. $x+y=4 \models 4=x+y$, ha az = relációt a hagyományos értelemben interpretáljuk.

A modell szerepe tehát kulcsfontosságú, ezt minden logikához pontosan definiáljuk majd.

Általában az α, aminek a következményire kíváncsiak vagyunk, a *tudásbázis*, amely szabályokat (axiómákat) és tapasztalati tényeket tartal-

maz. Pl. a wumpus világban a szabályok a játékszabályok, a tények pedig lehetnek pl. „(1,1)-ben nem érzek semmit” és „(2,1)-ben szellőt érzek”, ezek együtt alkotják az α tudásbázist. A modellek a lehetséges wumpus világok (az alkotóelemek szabályoknak megfelelő bármely elrendezése). Logikai következmény pl.: $\alpha \models$ (1,2)-ben nincs verem. Viszont az nem logikai következmény, hogy pl. (2,2)-ben van verem: egyes modellekben lesz, másokban nem.

Vigyázat! A wumpus tudásbázist nagyon informálisan és intuitívan tárgyaltuk, a játék kielégítő axiomatizálása elég bonyolultan nézne ki (szomszédsági relációk, stb.). Csak az intuíció felépítése a cél.

Logikai következtetés (vagy bizonyítás): $\alpha \vdash_i \beta$ akkor és csak akkor ha i algoritmus α inputtal indítva olyan formulahalmazt produkál, amelyben szerepel β.

Vegyük észre, hogy ez nagyon általános definíció, ami semmit sem mond arról, hogy az algoritmus mit csinál. Azt, hogy egy algoritmusnak van-e értelme, az dönti el, hogy a helyesség és teljesség tulajdonságokkal rendelkezik-e.

- i algoritmus *helyes* akkor és csak akkor ha $\alpha \vdash_i \beta \Rightarrow \alpha \models \beta$
- i algoritmus *teljes* akkor és csak akkor ha $\alpha \vdash_i \beta \Leftarrow \alpha \models \beta$
- i algoritmus *cáfolás teljes* akkor és csak akkor ha $\alpha \wedge \neg\beta \vdash_i \bot \Leftarrow \alpha \models \beta$

ahol a $\bot$ szimbólum a logikai ellentmondást jelenti, azaz egy olyan formulát, amely egyetlen modellben sem igaz. Ez a definíció csak effektíve futtatható algoritmusokra vonatkozik, így szűkebb mint a szokásos matematikai bizonyításfogalom, de praktikusan elegendő.

Pl. ha az algoritmus minden formulát felsorol, akkor teljes lesz (és ha egyet sem, akkor helyes). A lényeg, hogy adott logikához találjunk olyan algoritmust ami hatékony, helyes, és teljes.

8.3. Ítéletkalkulus

8.3.1. Szintaxis

Formális nyelv ami a jólformált mondatokat generálja:

mondat	$\longrightarrow$	atom | $\neg$mondat | (mondat OP mondat)
OP	$\longrightarrow$	$\wedge$ | $\vee$ | $\rightarrow$ | $\leftrightarrow$
atom	$\longrightarrow$	**I** | **H** | ítéletváltozó
ítéletváltozó	$\longrightarrow$	A | B | C | ...

Véges számú ítéletváltozót engedünk meg. A zárójeleket elhagyjuk, precedencia: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

Megj.: Ha csak két műveletünk van, pl. $\neg, \wedge$, az is elég lett volna. Sőt, a NAND művelet önmagában is elég. A NAND $B \Leftrightarrow \neg(A \wedge B)$.

8.3.2. Szemantika

A mondat felépítése szerint rekurzívan rendelünk igazságértékeket a mondathoz.

Az atomok igazságértékét közvetlenül a modell határozza meg, ami az atomokhoz rendel I vagy H értéket (igaz vagy hamis). Pl. ha 3 atom van akkor $2^3 = 8$ különböző modell lehetséges.

Az összetett mondatok igazságértékét igazságtáblával határozzuk meg, pl. $\neg I = H, \neg H = I$, stb.

8.3.3. Wumpus világ az ítéletkalkulusban

Nem ítéletkalkulus az ideális, de megoldható (mert kicsi a tér): P_{ij}: (i, j) cellában verem van, ill. B_{ij}: (i, j) cellában szellő van, stb. Ezen a nyelven kell kifejezni a szabályokat:

- $\neg P_{11}$ (kezdőállapotban nincs verem)
- $B_{11} \leftrightarrow (P_{12} \vee P_{21})$ (szellő szabály az (1,1) cellára)
- $B_{21} \leftrightarrow (P_{11} \vee P_{22} \vee P_{31})$ (szellő szabály a (2,1) cellára)
- stb.

A tényeket is ezen a nyelven lehet kifejezni, pl. korábbi példában az első tény amit érzékelünk az $\neg B_{11}$, aztán B_{21}, stb.

A tudásbázist a szabályok és a tények alkotják. (Általában a tudás egyetlen mondatnak vehető, mert $\wedge$ jellel összekapcsolhatók ha több van.)

Mivel ítéletkalkulusban vagyunk, minden cellára meg kell ismételni a szomszédsági szabályokat... Predikátumlogika elegánsabb lenne itt.

8.4. Logikai következtetés ítéletkalkulusban

Néhány alapfogalom és összefüggés.

Logikai ekvivalencia: α és β logikailag ekvivalens ($\alpha \equiv \beta$) akkor és csak akkor ha $\alpha \models \beta$ és $\beta \models \alpha$.

Tautológia: α tautológia ($\models \alpha$) akkor és csak akkor ha minden modellben igaz.

Dedukciós tétel: $\alpha \models \beta$ akkor és csak akkor ha $\models \alpha \rightarrow \beta$.

Kielégíthetőség: α kielégíthető akkor és csak akkor ha legalább egy modellben igaz.

Bármely tautológia tagadása tehát kielégíthetetlen. Emiatt működik a *reductio ad absurdum* vagyis ellentmondásra vezetés: $\alpha \models \beta$ akkor és csak akkor ha $\neg(\alpha \rightarrow \beta) \equiv \alpha \wedge \neg\beta$ kielégíthetetlen.

Ismert, hogy az ítéletkalkulus kielégíthetőségi problémája (adott α mondatról döntsük el, hogy kielégíthető-e) NP-teljes (Boolean satisfi-

ability, SAT). A komplementer probléma: az ítéletkalkulus logikai következmény vizsgálatának a problémája co-NP-teljes (mivel az ellentmondásra vezetés módszerével egy mondat kielégíthetetlenségének a vizsgálatára vezethető vissza).

8.4.1. Modellellenőrzéses algoritmusok

Egy korlátozás kielégítési feladatot definiálunk formulák kielégíthetőségének az ellenőrzésére. Ezt arra használjuk, hogy a $\alpha \wedge \neg\beta$ formula kielégíthetőségét vizsgáljuk (reductio ad absurdum). Azaz ha találunk modellt, amely kielégíti a $\alpha \wedge \neg\beta$ formulát, akkor cáfoljuk az $\alpha \models \beta$ következményt. Egyébként ha el tudjuk dönteni, hogy nincs ilyen modell, akkor igazoljuk, ha pedig nem, akkor nem tudunk dönteni.

Felfoghatjuk úgy is, hogy az algoritmus vagy levezeti a $\perp$ (ellentmondás) formulát, vagy semmit sem vezet le. Így tekinthetjuk következtető algoritmusnak, és értelmezhetővé válik a helyesség és a cáfolás teljesség tulajdonsága.

A mondat változói az atomok, lehetséges értékeik az I és H. A korlátozások definiálásához alakítsuk a mondatot *konjunktív normálformára*, ami *klózok* konjunkciója ($\wedge$). Egy klóz *literálok* diszjunkciója ($\vee$), egy literál pedig egy atom vagy egy negált atom. Pl. $(A \vee \neg B) \wedge (\neg A \vee C \vee D)$. Tetszőleges mondat konjunktív normálformára hozható logikailag ekvivalens átalakításokkal.

A korlátok az egyes klózok által vannak definiálva. Minden klóznak ki kell elégülni, és egy klóz a benne szereplő atomokra fogalmaz meg feltételt.

Ha tehát valamely változóhozzárendelésre minden klóz kielégül, akkor az eredeti mondat kielégíthető, ezek a célállapotok.

Láttuk korábban, hogy gráfkereséssel és optimalizálással is próbálkozhatunk egy korlátozás kielégítési feladat megoldására, ezeket megnézzük megint.

Megoldás inkrementális kereséssel

A megoldás szó szerint megegyezik a korlátozás kielégítési feladatnál leírtakkal (felvesszük az ismeretlen „?” értéket, kezdőállapotban minden változó „?”, és az operátor egy „?”-hez megengedhető értéket rendel).

A végállapotok itt azok a modellek lesznek, amelyek megengedhetőek. Érdemes a mélységi fakeresést alkalmazni (más néven visszalépéses keresés v. backtrack): fa mélysége kicsi, memóriaigény konstans. Ha talál megoldást, akkor kielégíthető, ha lefut megoldás nélkül akkor kielégíthetetlen a mondat. Mindig lefut, mert véges a fa.

A fentiekből következik, hogy az algoritmus segítségével az ítéletkalkulus logikai következmény problémáját *el tudjuk dönteni*.

Emellett az algoritmus helyes, és cáfolás teljes is.

Tudjuk, hogy a probléma NP-teljes, de lehet a hatékonyságot növelni általános heurisztikákkal (láttuk korábban, hogy a leginkább korlátozó feltételt kell kielégíteni, stb.), és feladatspecifikus javításokkal:

- Ha egy A változó csak negálva vagy csak negálatlanul szerepel, akkor vegyük igaznak (ill. hamisnak) keresés nélkül.
- Nem kell minden változónak értéket adni: ha A igaz, akkor $(A \vee B) \wedge (A \vee C)$ is igaz B és C értékétől függetlenül
- Ha egy klózban csak egy literál van, annak azonnal értéket lehet adni. Ugyanígy, ha egy klózban minden literálnak H értéket adtunk, kivéve egynek, akkor ennek az egynek mindenképpen I-t kell adni.

(DPLL és Chaff algoritmusok.)

Megoldás optimalizálással

Ismét a korlátozás kielégítési feladatoknál leírtaknak megfelelően definiáljuk az optimalizálási problémát (a keresési tér a modellek halmaza, a célfüggvény a hamis klózok száma amit minimalizálni akarunk).

Alkalmazhatjuk pl. az újraindított sztochasztikus hegymászót, véletlen kezdőállapotból. A keresőoperátor lehet egy véletlen kiértékelés megváltoztatása (vagy egy hamis klózban szereplő kiértékelés megváltoztatása).

Fontos, hogy *nem garantált*, hogy találunk kiértékelést, akkor sem, ha van. Ha találunk, akkor ezzel bizonyítjuk, hogy van, de ha nem találunk, akkor általában nem bizonyítjuk, hogy nincs, tehát nem tudjuk az $\alpha\wedge\neg\beta$ formula ellentmondásosságát levezetni.

Az algoritmus tehát helyes (mert sosem vezet le semmit!), de nem cáfolás teljes.

8.4.2. Szintaktikai manipuláció

A modellek közvetlen vizsgálata helyett a formulákat „gyúrjuk”, de ügyesen, hogy a logikai következmény fogalmával konzisztens maradjon a tudásbázis. Pl. ha α és $\alpha \to \beta$ a tudásbázisban szerepel, akkor felvehetjük β-t is: ez a *modus ponens* levezetési szabály. Bár ez tisztán a formula manipulációja, belátható, hogy β a tudásbázis logikai következménye.

Van-e olyan algoritmus az ítéletkalkulusban amely szintaktikai manipulációk (vagyis levezetés) segítségével működik és teljes és helyes? Igen: sőt, megadható levezetési szabályok egy véges halmaza amelyeket felváltva alkalmazva levezethető bármely logikai következmény.

(Mivel tudjuk, hogy ellentmondást tartalmazó (kielégíthetetlen) tudásbázisból az összes formula levezethető, egy ilyen algoritmussal levezethetünk pl. egy $\bot \equiv A \wedge \neg A$ típusú formulát ha a formulánk nem

kielégíthető.)

Rezolúció

Egy cáfolás teljes és helyes következtető algoritmus, tehát arra nem alkalmas, hogy felsoroljuk egy tudásbázis összes logikai következményét, de arra igen, hogy adott következtetés helyességét ellenőrizzük.

A rezolúció formulák konzisztenciájának az eldöntésére alkalmas, és az ellentmondásra vezetés módszerét használva használható logikai következtetésre. Itt is az $\alpha \wedge \neg\beta$ formulát vizsgáljuk, ellentmondást szeretnénk levezetni.

Először hozzuk a formulát itt is konjunktív normálformára. Ezután alkalmazzuk bármely teljes és helyes gráfkereső algoritmust a következő keresési téren:

- *keresési tér*: $\alpha \wedge \neg\beta$ konjunktív normálformájában szereplő literálokból álló klózokból álló halmazok
- *kezdőállapot*: $\alpha \wedge \neg\beta$ konjunktív normálformájának klózaiból álló halmaz
- *operátor*: a rezolúciós szabály. Legyen $l_1 \vee \ldots \vee l_k$ és $m_1 \vee \ldots \vee m_n$ olyan klózok amelyekre az l_i és az m_j literálok éppen egymás tagadásai (pl. $l_i = A, m_j = \neg A$). Ebből állítsuk elő a $C = l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n$ klózt, majd C-ből állítsuk elő C'-t úgy, hogy minden olyan literálból ami kétszer szerepel csak az egyiket hagyjuk meg. Adjuk hozzá C'-t a klózok halmazához.
- *célállapot*: bármely állapot amely az üres klózt (kielégíthetetlen klóz, egy literál sincs benne) tartalmazza

Először is a rezolúciós keresés (teljes és helyes gráfkereséssel) mindig terminál, mert a keresési tér véges (csak véges literál van). Egy formula

rezolúciós lezártján azoknak a klózoknak a halmazát értjük, amelyek a formulából levezethetők a rezolúciós szabállyal, azaz a keresési tér azon részhalmazát, amely a kezdőállapotból elérhető.

Ha a célállapotot elérjük, azaz ellentmondásra jutottunk, akkor ez igazolja a $\alpha \models \beta$ logikai következményt. Ez abból következik, hogy a rezolúciós szabály *helyes*, hiszen, mivel l_i és az m_j egymás negáltjai, valamelyik biztosan hamis, tehát elhagyható. De a fenti C klóz éppen azt mondja, hogy vagy l_i vagy m_j elhagyható.

Másreszt a rezolúció cáfolás teljes (elhisszük), tehát ha $\alpha \models \beta$ akkor levezeti az ellentmondást. Mivel az algoritmus mindig terminál, a cáfolás teljességet máshogyan úgy fogalmazhatjuk meg, hogy ha nem értünk el célállapotot (azaz nem vezettünk le ellentmondást) termináláskor, akkor a $\alpha \wedge \neg\beta$ formula kielégíthető (tehát $\alpha \not\models \beta$). (A bizonyítás alapötlete, hogy ekkor mindig konstruálható a rezolúciós lezárt alapján egy modell, amely kielégíti $\alpha \wedge \neg\beta$-t.)

8.4.3. Horn formulák

Láttuk, hogy a következtetés co-NP-teljes, mit lehet tenni? Megszoríthatjuk a formulák halmazát.

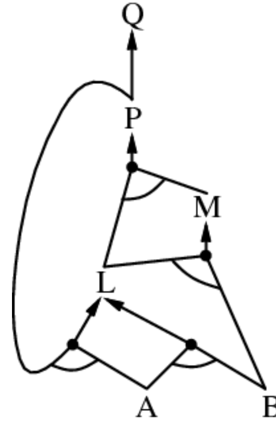
A *Horn formula* egy $A_1 \wedge \ldots \wedge A_k \rightarrow B$ alakú formula, ahol az A_i és B formulák atomok. Az $A_1 \wedge \ldots \wedge A_k$ formula a *test*, a B a *fej*. Konjunktív normálformában ez éppen egy klóz, amelyben pontosan egy pozitív literál van ($\neg A_1 \vee \ldots \vee \neg A_k \vee B$).

A *Horn tudásbázis* az Horn formulák halmaza. Speciálisan ha $k = 0$ akkor csak feje van a formulának, ezeket úgy hívjuk, hogy *tények* (a tények pozitív literálok vagyis atomok).

Nem hozható minden tudásbázis Horn alakra, de azért használható részhalmaza a formuláknak. Lineáris idejű következtetés lehetséges (a tudásbázis méretében) a modus ponens szabállyal.

Példa:

Tudásbázis:
$P \to Q$
$L \land M \to P$
$B \land L \to M$
$A \land P \to L$
$A \land B \to L$
A
B



ÉS-VAGY gráf. *Előreláncolás*: Legyen minden atom hamis kezdetben. Ismert atomokat (A, B) igaznak veszem. Ha egy ÉS kapcsolat minden bemenete igaz, akkor a kimenetét is igaznak veszem: propagáljuk a gráfban a levezetést. Vége: ha nincs több levezetési lehetőség. Lineárisan (az élek számában) megvalósítható.

Az algoritmus helyessége világos (modus ponens).

Teljességet itt definiáljuk csak az atomok fölött, azaz teljes legyen akkor ha minden atomot, ami logikai következmény, levezet.

A teljesség bizonyítása: A futás után az atomok egy része igaznak, a többi hamisnak van jelölve. Ez egy modellt definiál. Először is, ebben a modellben az eredeti tudásbázis minden formulája igaz, mert tegyük fel, hogy létezik hamis formula. Ennek a premisszája (teste) tehát igaz (ami csak úgy lehet, hogy abban minden atom igaznak van jelölve), a konklúziója (feje) hamis. De ez ellentmondás mert akkor az algoritmus még nem terminált volna. Viszont minden modellben, ahol minden eredeti formula igaz, ott minden logikai következménye is igaz definíció szerint, tehát ebben a modellben is. Tehát nem lehet, hogy van olyan atom ami logikai következmény is nincs igaznak jelölve.

Hátrafelé láncolás: Az ÉS-VAGY gráfban a levezetendő β atomból indulunk. Ha benne van a tudásbázisban, akkor készen vagyunk, β-t le-

vezettük. Különben rekurzívan megkíséreljük levezetni a premisszáit, és ha mindet sikerül, akkor β-t levezettük (modus ponens), egyébként nem sikeres a levezetés.

Ez hatékonyabb ha sok az irreleváns szabály.

8.5. Elsőrendű logika

Az ítéletkalkulus szempontjából az atomi mondatoknak nem volt szerkezete (fekete doboz). Az ítéletkalkulust általánosítjuk: objektumokat tételezünk fel (domain), és ezek függvényeit és kapcsolatait írjuk le az atomi mondatokkal.

Változókat is megengedünk, de csak az objektumok felett.

8.5.1. Szintaxis

Egyszerűsített nyelvtan a jólformált mondatok generálásához:

mondat	$\longrightarrow$	atom | $\neg$mondat | (mondat OP mondat) |
		| kvantor változó mondat
OP	$\longrightarrow$	$\wedge$ | $\vee$ | $\rightarrow$ | $\leftrightarrow$
kvantor	$\longrightarrow$	$\forall$ | $\exists$
atom	$\longrightarrow$	predikátum(term, ...) | term=term
term	$\longrightarrow$	függvény(term,...) | konstans | változó

Ezen kívül szükség van a predikátumok, függvények, konstansok és változók neveinek halmazára. A változókat $x, z, y \ldots$ kisbetűkkel, a konstansokat $A, B, C, \ldots$ nagybetűkkel jelöljük általában.

8.5.2. Szemantika

Az ítéletkalkulus szemantikája az atomokhoz direkt rendelt igazságértéket. Itt az atomok igazságértékét már származtatjuk.

Egy lehetséges világ megadásakor szükségünk van egy D *domain*re, ami egy halmaz, amelyben a lehetséges világban létező összes lehetséges objektum található, és egy *interpretációra*.

Egy mondathoz úgy rendelünk igazságértéket, hogy veszünk egy D domaint, majd a nyelv elemeit *interpretáljuk* D fölött, végül az adott interpretáció mellett a mondatot *kiértékeljük*.

Interpretáció

Megadjuk a konstansok, predikátumok és függvények *interpretáció*ját a következőképpen:

- minden konstansnévhez rendelünk egy elemet D-ből
- minden n-változós függvénynévhez rendelünk egy $D^n \rightarrow D$ függvényt
- minden n-változós predikátumnévhez rendelünk egy $D^n \rightarrow igaz/hamis$ függvényt

Egy lehetséges világban tehát egy domain fölött többféle interpretáció lehetséges, ill. a domain is változhat.

Mivel *változó*kat is megengedünk, egy mondat igazságértékét még nem tudjuk meghatározni. Ehhez kell még egy *változóhozzárendelés*, amely a változókhoz is rendel értékeket D-ből. A változóhozzárendelés az interpretációt egy *kiterjesztett interpretáció*vá egészíti ki.

Kiértékelés

Egy *term* (azaz kifejezés) kiértékelésének az eredménye a D domain egy eleme. Ha a term konstans vagy változó, akkor a kiterjesztett interpretáció megadja. Ha a term $f(t_1, \ldots, t_n)$ alakú, akkor kiértékeljük $t_1, \ldots, t_n$ termeket, és ezek értékeit behelyettesítjük az f-hez rendelt függvénybe.

Az *atomi mondatok* kiértékelésének az eredménye már igazságérték. Hasonlóan a függvényekhez, egy $P(t_1, \ldots, t_n)$ alakú predikátum kiértékeléséhez először a $t_1, \ldots, t_n$ termeket értékeljük ki, majd az így kapott értékeket behelyettesítjük a P-hez rendelt predikátumba. (Az egyenlőség egy speciális, rögzített interpretációjú predikátum: a „természetes egyenlőség” minden domainen.)

A kvantort nem tartalmazó *összetett mondatok* kiértékelése az ítéletkalkulussal analóg módon történik, a logikai műveletek igazságtáblája alapján.

A $\forall x, \phi$ formula kiértékelése „igaz”, ha ϕ igaz minden változóhozzárendelésben, amely az aktuális változóhozzárendeléstől legfeljebb az x-hez rendelt értékben különbözik. A $\exists x, \phi$ formula kiértékelése „igaz”, ha ϕ igaz valamely változóhozzárendelésben, amely az aktuális változóhozzárendeléstől legfeljebb az x-hez rendelt értékben különbözik.

8.5.3. Néhány megjegyzés

A logikai következmény definíciójában szereplő „minden modell”-t itt tehát úgy kell érteni, hogy „minden domain, minden interpretáció, minden változóhozzárendelés”, hiszen ez a hármas alkotja a modellt. (Más értelemben is használatos a modell kifejezés, pl. néha csak a domaint értik alatta, vagy csak a domaint és az interpretációt.)

Világos, hogy itt a kielégíthetőség (ill. a kielégíthetetlenség) problémáját nem lehet direkt ellenőrizni, pl. összes lehetséges n-változós predi-

kátum túl sok lehet még kis domainen is.

Másrészt a kielégíthetőség vizsgálatánál meg kell adni az interpretáción és a változóhozzárendelésen kívül még mást is, pl. egy $\exists x, \phi$ formulánál azt a konkrét x értéket amelyre kielégült a formula.

8.5.4. Példák

Rokonság: a predikátumnevek: {férfi, nő} egyváltozós, {nővér, szülő,...} kétváltozós; függvénynevek: {apa, anya } egyváltozós (lehetne kétváltozós predikátum is).

A „természetes” domain az emberek halmaza; és a természetes interpretáció a szokásos jelentése a szavaknak.

A tudásbázis pl.

1. $\forall x \forall y$ Anya$(x) = y \leftrightarrow$ Nő$(y) \wedge$ Szülő(y, x)
2. $\forall x \forall y$ Nagyszülő$(x, y) \leftrightarrow \exists z($Szülő$(x, z) \wedge$ Szülő$(z, y))$
3. Férfi(Péter), stb.

Természetes számok:

a predikátumnevek: $\{N\}$ egyváltozós (szándékolt jelentése: természetes szám); függvénynevek: $\{S\}$ egyváltozós (szándékolt jelentése: rákövetkező); konstansok: $\{0\}$

A „természetes” domain a természetes számok halmaza; és a természetes interpretáció: $S(x) = x + 1$, $N(x)$ igaz minden domain elemre, és a 0 konstansnév megfelelője a 0 természetes szám.

Tudásbázis (vagyis axiómák) célja itt az, hogy a modelleket úgy szorítsuk meg, hogy N-nek csak megszámlálható számosságú interpretációin teljesülhessenek az axiómák (azaz csak a természetes számokon, izomorfizmus erejéig). A Lövenheim-Skolem tételek miatt ez nem lehet-

séges elsőrendű nyelven (lesz megszámlálhatatlan számosságú modellje is), másodrendűn viszont igen.

1. $N(0)$
2. $\forall x\ (N(x) \to N(S(x)))$
3. $\forall x\ (N(x) \to 0 \neq S(x))$
4. $\forall x \forall y\ (S(x) = S(y) \to x = y)$

Ezen kívül az indukciós axióma (végtelen darab elsőrendű nyelven) és a műveletek axiómái kellenek, ezeket nem tárgyaljuk.

8.6. Logikai következtetés elsőrendű logikában

Szemben az ítéletkalkulussal, a logikai következmény problémája nem eldönthető. Olyan algoritmus létezik, amelyik mindig azt mondja, hogy „igen” ha az input formula logikailag következik az axiómákból, viszont olyan nem létezik, amelyik mindig azt mondja, hogy „nem” ha a formula nem következik logikailag. Ez a Turing gépek megállási problémájával függ össze.

A teljesség definíciójához azonban elég ha az algoritmus azt mondja, hogy „igen” ha az input formula logikailag következik az axiómákból. A rezolúció elsőrendű logikai verziója pl. cáfolás teljes.

Itt csak vázlatosan tárgyaljuk a rezolúciót!

Ötlet: az ítéletkalkulusnál látott következtetési szabályt szeretnénk alkalmazni, de ehhez (1) meg kell szabadulni a kvantoroktól (2) a következtetési szabálynál az atomi mondatok finomszerkezetét is figyelembe vesszük úgy, hogy megfelelő helyettesítéseket (szintén szintaktikai operáció itt!) is végzünk.

8.6.1. Kvantorok

Egy ϕ formulában x változó egy adott előfordulása *kötött* akkor és csak akkor ha az adott előfordulás ϕ-nek valamely $\forall x\psi$, vagy $\exists x\psi$ alakú részformulájában szerepel a ψ formulában. Egyébként az adott előfordulás *szabad*. Egy kötött változót mindig a hozzá balról legközelebbi rá vonatkozó kvantor köt.

Az *egzisztenciális* kvantorokat skolemizációval eltüntetjük: ha x egzisztenciális kvantorral kötött, és nincs univerzális kvantor hatáskörében, akkor $\exists x\ \phi(x)$ helyett $\phi(C)$-t veszünk fel, ahol C egy új Skolem konstans. Ha egy egzisztenciális kvantorral kötött x változó az $x_1, \ldots, x_k$ univerzális kvantorral kötött változók kvantorainak a hatáskörében van, akkor az $F(x_1, \ldots, x_k)$ egy új Skolem függvénnyel helyettesítjük. Pl. $\forall x \exists y\ P(x, y)$ helyett $\forall x\ P(x, F(x))$.

Ezután az *univerzális* kvantorokat elhagyjuk, és új változóneveket vezetünk be. Pl. $\forall x\ P(x) \wedge \forall x\ R(x)$ helyett $P(x_1) \wedge R(x_2)$. Ezt azért lehet, mert logikai következmény szempontjából egy szabad változó ekvivalens egy univerzálisan kötött változóval.

8.6.2. Felemelés (lifting)

Minden ítéletkalkulusbeli következtetési szabályra működik: pl. (általánosított) modus ponens elsőrendű logikában:

$$\frac{p'_1, \ldots, p'_n, (p_1 \wedge \ldots \wedge p_n) \rightarrow q}{\text{helyettesít}(\theta, q)}$$

ahol θ egy helyettesítés, amely termeket helyettesít változók helyébe, és amelyre helyettesít(θ, p'_i) = helyettesít(θ, p_i) minden i-re.

Például

$$\frac{\text{Fekete}(H), \text{Fekete}(x) \rightarrow \neg\text{Hattyú}(x)}{\neg\text{Hattyú}(H)}, \quad \theta = \{x/H\}$$

Megadható hatékony algoritmus amely megtalálja a *legáltalánosabb* θ helyettesítést, azaz azt a helyettesítést, amely a legkevesebb megkötést teszi a változókra: egyesít(p, q)=θ, ahol p és q mondatok, és helyettesít(θ, p) = helyettesít(θ, q).

Fontos: p és q szabad változóit más szimbólumokkal kell jelölni, különben a helyettesítés sikertelen lehet: pl. egyesít(Ismer(A, x), Ismer(x, B)) =?, de egyesít(Ismer(A, x), Ismer(y, B)) $= \{x/B, y/A\}$.

A lenti algoritmus egy helyettesítést ad vissza, meghívása az $x = p$ és $y = q$ modatokkal és a $\theta = \{\}$ üres helyettesítéssel. Futás során x és y lehet mondat, vagy term, vagy egy szimbolum, vagy egy listája ezeknek. Az op() függvény az operátor szimbólumot veszi ki az összetételből (ami függvény vagy predikátum), pl. op(F(a,b))=F.

```
egyesít(x,y,theta)
1 if theta == "kudarc" then return "kudarc"
2 if x==y then return theta
3 if váltzó(x) then return változóEgyesít(x,y,theta)
4 if váltzó(y) then return változóEgyesít(y,x,theta)
5 if összetétel(x) and összetétel(y) then
     return egyesít(argLista(x),argLista(y),
                    egyesít(op(x),op(y),theta))
6 if lista(x) and lista(y) then
     return egyesít(maradék(x),maradék(y),
                    egyesít(első(x),első(y),theta))
7 return "kudarc"
```

```
változóEgyesít(x,y,theta) // x változó
1 if theta része {x/A} then return egyesít(A,y,theta)
2 if theta része {y/B} then return egyesít(x,B,theta)
3 if y-ban előfordul x változó then return "kudarc"
5 return (theta + {x/y})
```

8.6.3. Rezolúció

Hasonlóan az ítéletkalkulushoz, itt is az $\alpha \wedge \neg\beta$ típusú formulák kielégíthetetlenségét szeretnénk eldönteni.

Itt is a konjunktív normálformából indulunk, amit a kvantoroktól való megszabadulás után az ítéletkalkulussal megegyező módon hozhatunk létre.

Egyetlen következtetési szabály, amely az ítéletkalkulusban megismert szabályból adódik liftinggel:

$$\frac{l_1 \vee \ldots \vee l_k, m_1 \vee \ldots \vee m_n}{\text{helyettesít}(\theta, l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

ahol egyesít$(l_i, \neg m_j) = \theta$, tehát helyettesít(θ, l_i) = helyettesít$(\theta, \neg m_j)$.

Az ítéletkalkulusban a fenti lépésen túl az azonos literálokból csak egyet hagytunk meg az új klózban. Itt az elsőrendű logikában ezt szintén meg kell tenni, de ehhez is egyesítés kell: ha két literál egyesíthető, az egyiket elhagyjuk. A maradék literálokra pedig alkalmazzuk a kapott helyettesítést. Ez a folyamat a faktorálás (*factoring*).

A rezolúció (faktorálással együtt) *cáfolás teljes* és helyes, tehát ha a kiindulási tudásbázis ellentmondást tartalmazott, akkor erre bizonyítást ad és viszont. Szemben az ítéletkalkulussal, itt nem terminál feltétlenül. A bizonyítást nem vesszük.

Rezolúció hatékonysága

A cáfolás teljesség nem garancia a hatékonyságra. Heurisztikák kellenek arra, hogy kiválasszuk, hogy melyik lehetséges rezolúciós lépést hajtsuk végre.

Az *egységpreferencia* stratégiája preferálja azokat a rezolúciós lépéseket, ahol literál (más néven egységklóz) az egyik klóz, hiszen igy a levezetett új klóz garantáltan rövidebb lesz mint a másik felhasznált klóz.

A *támogató halmaz* stratégiája kezdetben rögzít egy formulahalmazt

(támogató halmaz), és minden rezolúciós lépésben az egyik klóz ebből a halmazból származik. A levezetett klóz is bekerül a támogató halmazba. Ha a támogató halmazból kimaradt formulák együttesen kielégíthetőek (azaz konzisztens halmazról van szó) akkor teljes marad a rezolúció. Leggyakrabban a kezdeti támogató halmaz a bizonyítandó állítás negáltja ($\neg\beta$), feltéve, hogy a tudásbázis (α) konzisztens.

A *bemeneti rezolúció* során minden lépésben felhasználunk legalább egy formulát az α vagy $\neg\beta$ formula(halmaz)ból.

Végül, érdemes eltávolítani az olyan formulákat amelyek valamely, a tudásbázisban szereplő más formulából következnek. Pl. ha $P(x)$-t levezettük, akkor $P(A)$, vagy $P(A) \vee Q(B)$, stb, fölöslegesek. Ez a *bennfoglalás* módszere.

8.6.4. Előreláncolás

Definiáljuk először a DATALOG adatbázis formátumot, ami a Horn adatbázisok elsőrendű logikai analógja. Ez sem lesz teljes értékű logika, viszont a logikai következmény eldönthető lesz benne.

Feltesszük, hogy minden formula $\phi_1 \wedge \ldots \wedge \phi_k \rightarrow \psi$ alakú, ahol ϕ_i és ψ atomi formulák. Tények is lehetnek, amik önálló atomi formulák ($k = 0$).

Ezen kívül az atomi formulák paraméterei csak változók vagy konstansok lehetnek (nem engedünk meg függvényneveket a nyelvben).

Feltesszük, hogy kvantorok sincsenek. Ez is megszorítás, mert bizonyos kvantoroktól ugyan meg lehet szabadulni, de bizonyosaktól csak Skolem függvények bevezetésével lehetne, de függvényeket nem engedünk meg.

Ötlet: használjuk a modus ponens szabályt (mint a Horn adatbázisokban) és liftinget. A modus ponens alkalmazási lehetőségeinek a megadásához tehát a lehetséges helyettesítéseket is vizsgálni kell.

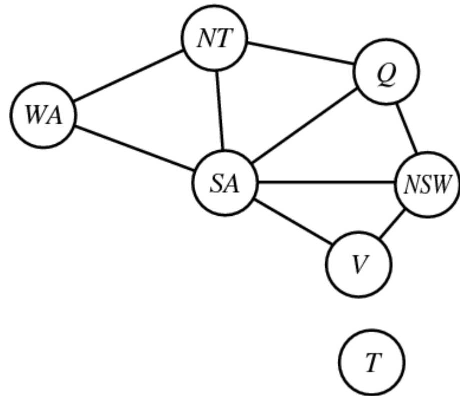
Tegyük fel, hogy β-t akarjuk levezetni. Az algoritmus terminál ha nincs több alkalmazási lehetősége a modus ponensnek, vagy egy levezetett tény helyettesítéssel β-val egy alakra hozható (utóbbi esetben β logikai következmény). Valamelyik eset mindig bekövetkezik, hiszen a levezethető tények halmaza, a helyettesítéseket is figyelembe véve, véges.

Ha függvényeket is megengednénk, nem terminálna mindig, pl. $N(0)$, $N(x) \to N(S(x))$ formulákból $N(S(0)), N(S(S(0))), \ldots$ levezethető, a következmények halmaza nem véges.

Mintaillesztés

Általában annak az eldöntése, hogy a modus ponens alkalmazható-e (létezik-e megfelelő helyettesítés) NP-nehéz. Pl. nem könnyebb mint a gráfszínezési probléma:

Tudásbázis (gráfszínezéshez):
$D(x, y)$ predikátum: x és y színe különbözik. egyetlen szabály: $D(wa, nt) \wedge D(wa, sa) \wedge \ldots \to$ színezhető() (a kényszergráfnak megfelelően).
Tények: D(piros,kék), D(piros,fehér), stb.
A gráfszínezési problémát a modus ponens alkalmazhatóságának az eldöntésére vezettük vissza.



A gyakorlatban nem mindig nehéz a probléma.

A korlátozás kielégítési feladat megoldásánál látott heurisztikák segítenek (pl. először azt az atomi formulát helyettesítjük, ami a legnehezebben kielégíthető, stb.)

8.6.5. Hátrafelé láncolás

Hasonlóan az ítéletkalkulushoz, az irreleváns információk levezetését elkerülhetjük a visszafelé láncolással.

Az algoritmus ugyanaz a rekurzív mélységi keresés mint az ítéletkalkulusban, csak a helyettesítéseket kell megfelelően kezelni. Az algoritmus visszatérési értéke azon helyettesítések halmaza, amelyekre a bizonyítandó formula (β) teljesül.

A bizonyítandó formula mindig egy atomi mondat, és az adatbázis továbbra is DATALOG formátumú. A következőt tesszük (csak az ötlet vázlata!): megkeressük az összes formulát aminek a feje illeszkedik β-ra (azaz egyesíthető vele), és minden ilyen formulának a premisszáira rekurzívan meghívjuk a visszafelé láncolást, de úgy, hogy minden premissza meghívásakor továbbítjuk azt a helyettesítést is, ami a fej és β egyesítésből adódott, és amiben a már bejárt premisszák illesztése során kapott helyettesítések is benne vannak. Ha minden premissza kész, akkor visszaadjuk a közben kapott helyettesítések unióját.

Ha β tényre illeszkedik (nincs premissza), akkor visszaadjuk azt a helyettesítést ami az illeszkedést biztosította.

Gyakorlatilag egy mélységi keresésről van szó itt is.

Logikai programozás

A hátrafelé láncolás ötletének szofisztikáltabb megvalósítása.

algoritmus = logika + irányítás. A „programozás” deklaratív, azaz a tudásbázist kell kódolni, az irányítási komponens (vagyis következtető algoritmus) problémafüggetlen, rögzített.

Pl. Prolog: az adatbázis lényegében DATALOG típusú, itt is hátrafelé láncolás mélységi kereséssel, ahol a rekurzió mindig a premisszák felsorolási sorrendjében történik, ill. a szabályok felsorolási sorrendje is számít.

Néhány kompromisszum a deklaratív és procedurális megközelítésnek, pl. input/output műveletek, ill. szintaktikai támogatás az aritmetikára és a listák kezelésére, stb.

A mélységi keresésből adódhatnak problémák, pl. vegyük a következő adatbázist:

1. él(x,y) $\to$ út(x,y)
2. út(x,z) $\wedge$ él(z,y) $\to$ út(x,y)

És vegyük az él(A,B) és él(B,C) tényekkel adott gráfot. Pl. az út(A,C) lekérdezésre a Prolog helyesen fog működni, azonban ha a szabályok sorrendjét felcseréljük, akkor végtelen ciklusba esik, mert a rekurzió először a 2. szabály alapján indul el, ami a mindig újabb út predikátumokat hoz be.

Tehát a prolog még DATALOG típusú adatbázison sem teljes. Az előreláncolás, mint láttuk, viszont az.

8.6.6. Automatikus tételbizonyítás

A tételbizonyítóknak van több alkalmazása is, nem csak matematikai tételek bizonyítása. Pl. hardver és szoftver helyességének az ellenőrzése. Sok gyakorlati alkalmazás.

A logikai programozáshoz sok köze van, de nincs megszorítás a formulák alakjára (a teljes elsőrendű logikát akarjuk kezelni), és a kontroll sokkal jobban el van különítve mint a prolognál (ott számított pl. a formulák kilistázásának a sorrendje, stb.).

Az általános megközelítés miatt itt a hatékonyság a fő gond.

Legsikeresebb jelenleg a Vampire nevű tételbizonyító.

9. fejezet

Bizonytalanság

Eddig a logika igaz/hamis világát és logikai következtetéseket néztünk. De ezzel problémák vannak: (1) ha nem teljes a tudás (tények vagy szabályok), akkor nem mindig tudunk logikai levezetéseket gyártani fontos kérdésekhez, *döntésképtelenek* leszünk, és (2) ha heurisztikus szabályokat vezetünk be, akkor a tapasztalat inkonzisztens lehet az elmélettel, tehát a logikai levezetés megint csak nem működik, itt is döntésképtelenség és rugalmatlanság lehet az eredmény.

Tehát a hiányos, részleges tudás kezelésére a logika nem optimális.

Példa: orvosi diagnózis: fogfájás és lukas fog (röviden: luk). A fenti két probléma:

- *nem teljes tudás*: a teljes tudás logikai alakja ilyesmi:
  $\forall$ p (Tünet(p,fogfájás) $\rightarrow$ Betegség(p,luk) $\vee$ Beteség(p,gyulladás) $\vee \ldots$), tehát nem tudunk dönteni ha csak a fogfájás ténye ismert.
- *egyszerűsítés*: mondhatnánk, hogy akkor használjuk a $\forall$ p (Tünet(p,fogfájás) $\rightarrow$ Betegség(p,luk)) szabályt. Ez sokszor működni fog, de néha nem, és a hasonló szabályok miatt előbb utóbb a tudásbázis tele lesz ellentmondással. (*nem monoton logikák* kezelik ezt, de ezzel nem foglalkozunk).

9.1. Valószínűség

A tudás tökéletlenségének (azaz ismeretlen tényeknek és szabályoknak) véletlen hatásként való kezelése, pl. „fogfájás$\rightarrow$luk” 80%-ban igaz.

(Tökéletes tudásnál elég a logika? „God does not play dice” (Einstein))

A gyakorlatban a tudás mindig tökéletlen, mert (1) lusták vagyunk összegyűjteni a szabályokat (szakértői tudás) vagy a tényeket (tesztek elvégzése), vagy (2) a tudomány eleve nem elég fejlett.

Ismeretelméleti státusz: a valószínűség a *hit fokát* jelenti (bayesi felfogásban legalább is), nem az *igazság fokát*. Tehát feltesszük, hogy az állítás az aktuális világban igaz vagy hamis, csak ezt nem tudjuk pontosan. (A fuzzy logika pl. ezzel szemben az igazság fokát nézi, ahol egy állítás az aktuális világban folytonos igazságértékű, pl. „ez a ház nagy”.)

Egy állítás/esemény valószínűsége változik annak függvényében, hogy mit tapasztalunk. Pl. ha húzunk egy kártyalapot, más a valószínűsége annak, hogy pikk ász mielőtt és miután megnéztük... Beszélni fogunk tehát előzetes (a priori) és utólagos (a posteriori) valószínűségről.

9.1.1. Racionális ágens

A feladat az, hogy megfigyelésekből (tényekből) és valószínűségi jellegű a priori tudásból számoljuk ki az ágens számára fontos események valószínűségét. Egy cselekvés kimenetelének az értéke lehet pl. a lehetséges kimenetelek értékeinek a valószínűségekkel súlyozott átlaga (azaz a várható érték).

9.1.2. Véletlen változók

Nagyon praktikusan és egyszerűen tárgyaljuk, nem az általános mértékelméleti megközelítésben. Egy véletlen változónak van *neve* és *lehetséges értékei* azaz *domainje*.

Legyen A egy véletlen változó, D_A domainnel. A D_A típusának függvényében képezhetünk *elemi kijelentés*eket, amelyek az A értékének egy korlátozását fejezik ki (pl. $A = d$, ahol $d \in D_A$). A következő típusok vannak domain alapján:

- *logikai*: ekkor a domain $\{$igaz, hamis$\}$. Pl. Fogfájás (a név mindig nagybetűvel írva). Ekkor a „Fogfájás=igaz” egy elemi kijelentés. Röviden a „Fogfájás=igaz” helyett azt írjuk, hogy „fogfájás” (kisbetűvel). Pl. „fogfájás $\wedge$ $\neg$luk” azt rövidíti, hogy „Fogájás=igaz $\wedge$ Luk=hamis”.
- *diszkrét*: megszámlálható domain. Pl. Idő, ahol a domain pl. $\{$nap, eső, felhő, hó$\}$. Röviden az „Idő=nap” elemi kijelentés helyett azt írjuk, hogy „nap”.
- *folytonos*: pl. X véletlen változó, $D \subseteq \mathbb{R}$. Pl. $X \leq 3.2$ egy elemi kijelentés.

Komplex kijelentéseket képezhetünk más kijelentésekből a szokásos logikai operátorokkal ($\wedge, \vee, \neg, \ldots$).

Az *elemi esemény* (a lehetséges világok analógja) minden véletlen változóhoz értéket rendel: ha az $A_1, \ldots, A_n$ véletlen változókat definiáltuk a $D_1, \ldots, D_n$ domainekkel, akkor az elemi események (lehetséges világok) halmaza a $D_1 \times \cdots \times D_n$ halmaz. Egy lehetséges világban (azaz elemi eseményben) tehát az A_i véletlen változó a hozzá tartozó D_i-ből pontosan egy értéket vesz fel.

Pl. ha két logikai véletlen változónk van: Luk és Fogfájás, akkor négy elemi esemény van: „luk$\wedge$fogfájás”, „luk$\wedge\neg$fogfájás”, „$\neg$luk$\wedge$fogfájás”„$\neg$luk$\wedge\neg$fogfájás”.

Más tárgyalásokban az elemi események nem származtatottak, hanem azokból indulunk ki. Itt viszont a nyelvből generáljuk őket, tehát felfoghatók úgy mintha a *megkülönböztethető* lehetséges világokat definiálnák.

A fenti definíciók néhány következménye:

1. minden lehetséges világot (tehát az aktuális világot is) pontosan egy elemi esemény írja le (modellezi)
2. egy elemi esemény természetes módon minden lehetséges elemi kijelentéshez igazságértéket rendel
3. minden kijelentés logikailag ekvivalens a neki nem ellentmondó elemi eseményeket leíró kijelentések halmazával

9.1.3. Valószínűség

A fenti konstrukció logikai véletlen változók esetén eddig ekvivalens az ítéletkalkulussal (ill. diszkrét változók esetén is az elemi események felett). Ott azonban egy formulával kapcsolatban három lehetséges tulajdonság valamelyike állt: logikailag (minden modellben) igaz, logikailag hamis, vagy egyik sem.

Ez utóbbi esettről szeretnénk finomabb ítéletet mondani (mégis mennyire „számíthatunk rá”, hogy igaz, ha már logikailag se nem igaz, se nem hamis), erre lesz jó a valószínűség.

A valószínűség egy függvény, amely egy kijelentéshez egy valós számot rendel. Először a jelöléseket tárgyaljuk, a valószínűség tulajdonságait később.

A véletlen változókat nagybetűvel ($A, B, X, Y, \ldots$), a kijelentéseket, és a változók értékeit kisbetűvel ($a, b, x, y, \ldots$) jelöljük.

Legyen $P(a)$ az a kijelentés *valószínűsége*, pl. $P(\text{Idő=nap}) = 0.5$, ahol a egy előre rögzített nyelven tett tetszőleges kijelentés, és ahol a $P()$

függvény a *valószínűségi eloszlás* amely az összes lehetséges kijelentéshez valószínűséget rendel.

Ha A egy véletlen változó, akkor használni fogjuk a $P(A)$ jelölést is állítások és definíciók részeként. Azt értjük alatta, hogy A tetszőleges értékére érvényes az adott állítás/képlet. Általában is, minden esetben, ha a $P()$ jelölést olyan formulára alkalmazzuk, amelyben szerepel értékadás nélküli változó (egy vagy több), akkor azt úgy fogjuk érteni, hogy minden lehetséges értékadásra teljesül az adott állítás.

A $P(A, B)$ jelölést és a $P(A \wedge B)$ jelölést felcserélhetően alkalmazzuk.

A $P(A)$ típusú jelölést olyan esetekben is használjuk, ahol A változók halmaza, ilyenkor ezen azt értjük, hogy $P(A) = P(A_1, \ldots, A_n)$, feltéve, hogy $A = \{A_1, \ldots, A_n\}$. Hasonló értelemben a jelölhet egy kijelentés helyett kijelentések egy halmazát is.

9.1.4. Valószínűség tulajdonságai

Eddig lebegtettük a valószínűség fogalmát, csak azt használtuk, hogy egy kijelentés valószínűsége egy valós szám. Most megadjuk a valószínűség axiómáit: tetszőleges a és b kijelentésre

1. $0 \leq P(a)$
2. $P(\text{igaz}) = 1$, $P(\text{hamis}) = 0$, ahol az igaz kijelentés egy tautológia, a hamis pedig ellentmondás.
3. $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

Bizonyítsuk be, hogy $P(\neg a) = 1 - P(a)$. Tudjuk, hogy $P(a \vee \neg a) = P(a) + P(\neg a) - P(a \wedge \neg a)$. De a tautológiák és ellentmondások valószínűsége miatt $1 = P(a) + P(\neg a) - 0$, ebből kijön.

Hasonló módon belátható, hogy egy A diszkrét véletlen változóra

$$1 = \sum_{a \in D_A} P(A = a),$$

ahol D_A A domainje, hiszen A pontosan egy értéket vesz fel, azaz $P(A = a \wedge A = b) = 0$ ha $a \neq b$, és $P(\vee_{a \in D_A} A = a) = 1$, azaz az A változó elemi kijelentéseihez tartozó valószínűségek (amik A eloszlását definiálják) összege 1 (a $\vee_{a \in D_A} A = a$ kijelentés tautológia).

Láttuk, hogy minden kijelentés elemi események egy halmazával ekvivalens (azon elemi események amelyek konzisztensek a kijelentéssel). Legyen $e(a)$ az a kijeletést alkotó elemi események halmaza. Ekkor belátható, hogy ha $e(a)$ megszámlálható, akkor

$$P(a) = \sum_{e_i \in e(a)} P(e_i)$$

hiszen minden $e_i \neq e_j$ elemi eseményre $P(e_i \wedge e_j) = 0$, és minden állítás elemi események diszjunkciójaként írható fel, tehát a diszjunkció axiómájából kijön.

9.1.5. Feltételes valószínűség

Legyen $P(a|b)$ az a kijelentés *feltételes valószínűsége*, feltéve, hogy az összes tudásunk b (b is egy kijelentés). pl. $P(\text{luk}|\text{fogfájás}) = 0.8$.

Definíció szerint $P(a|b) = P(a \wedge b)/P(b)$ (feltéve, hogy $P(b) > 0$).

A *szorzatszabály*: $P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$.

9.1.6. Valószínűségi következtetés

A logikában a logikai következményt néztük: mi az ami *mindig* igaz a rendelkezésre álló ismereteket feltéve.

A valószínűségben finomabb a felbontás: a rendelkezésre álló ismeretek fényében mi egy adott esemény vagy kijelentés valószínűsége? A logikai következmények valószínűsége 1 lesz, viszont azokról a dolgokról is mondunk valamit amik nem logikai következmények.

A *teljes együttes eloszlásból* indulunk ki, ami az összes elemi esemény valószínűségét megadja. Láttuk, hogy ebből kiszámolható bármely kijelentés valószínűsége (tehát feltételes valószínűsége is).

A továbbiakban feltesszük, hogy az összes változónk diszkrét.

Például legyenek a véletlen változók Luk, Fogfájás, Beakad, mind logikai típusú. Ekkor a teljes együttes eloszlást egy táblázat tartalmazza. Ebből pl. $P(\text{luk}\vee\text{fogfájás}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064$.

Luk	Fogfájás	Beakad	$P()$
nem	nem	nem	0.576
nem	nem	igen	0.144
nem	igen	nem	0.064
nem	igen	igen	0.016
igen	nem	nem	0.008
igen	nem	igen	0.072
igen	igen	nem	0.012
igen	igen	igen	0.108

Speciális eset: amikor elemi kijelentések konjunkcióját nézzük: ezeket *marginális valószínűségeknek* nevezzük. Pl. $P(\text{luk})$ vagy $P(\text{luk}\wedge\neg\text{beakad})$.

A *marginalizáció* (marginális valószínűség kiszámítása) a következő művelet:

$$P(A) = \sum_x P(A, x),$$

ahol A a változók egy halmaza, és x egy vagy több, A-ban nem szereplő változó összes lehetséges értékkombinációin fut végig. Ha feltételes valószínűségek ismertek csak, akkor a feltételes valószínűség definícióját alkalmazva

$$P(A) = \sum_x P(A|x)P(x),$$

aminek a neve *feltételfeloldás* (conditioning). Ez fontos lesz később.

A következtetés során van tudásunk, amely tényekből (pl. fogfájást tapasztalunk) és általános tudásból (ismerjük a teljes együttes eloszlást) áll. Ekkor kérdezhetünk ismeretlen tények valószínűségére, ez a *valószínűségi következtetés*. A kérdés alakja feltételes valószínűség lesz, pl. $P(\text{luk}|\text{fogfájás}) = ?$, ez pedig

$$P(\text{luk}|\text{fogfájás}) = P(\text{luk} \wedge \text{fogfájás})/P(\text{fogfájás})$$

Általában a Luk logikai véletlen változóra felírva ugyanez a példa:

$$\begin{aligned}
&P(\text{Luk}|\text{fogfájás}) = \\
&= \frac{1}{P(\text{fogfájás})} P(\text{Luk}, \text{fogfájás}) = \left(\frac{0.12}{P(\text{fogfájás})}, \frac{0.08}{P(\text{fogfájás})} \right),
\end{aligned}$$

ahol a kételemű vektor a Luk=¬luk és Luk=luk értékadásokhoz tartozó valószínűségek.

A fenti példában a $P(\text{fogfájás})$ kiszámolható, mert tudjuk, hogy

$$P(\text{luk}|\text{fogfájás}) + P(\neg\text{luk}|\text{fogfájás}) = 1,$$

tehát $P(\text{fogfájás}) = 0.08 + 0.12$, ami éppen a normalizálási konstans.

Sokszor ez a normalizálási konstans nem érdekes (pl. ha csak a valószínűségek nagyság szerinti sorrendje érdekes egy döntés meghozatalához). A képlet így

$$P(A|b) = \alpha P(A, b) = \alpha \sum_x P(A, b, x)$$

ahol $\alpha = 1/P(b)$ a normalizálási konstans, b az ismert tények, x az ismeretlen tények (változók) lehetséges értékei, A pedig az érdeklődésünk tárgya.

Ez kiszámolható A minden lehetséges értékére, ha a teljes együttes eloszlás ismert (pl. táblázatból). De ez a táblázat nagy, pl. ha minden változó logikai, és n a változók száma, akkor ez 2^n méretű. Ez a módszer tehát nem skálázódik. Fő kérdés: *hatékony* módszerek!

9.1.7. Függetlenség

A kijelentések függetlensége a legfontosabb tulajdonság a teljes együttes eloszlás tömöríthetőségéhez. Két fajtája lesz: függetlenség és feltételes függetlenség.

Az a és b kijelentések *függetlenek* akkor és csak akkor ha $P(a \wedge b) = P(a)P(b)$.

Az A és B véletlen változók (vagy változóhalmazok) függetlenek akkor és csak akkor ha $P(A, B) = P(A)P(B)$, vagy ekvivalensen $P(A|B) = P(A)$, ill. $P(B|A) = P(B)$.

Intuitíve két változó független ha az egyik nem tartalmaz információt a másikról. Gondoljuk meg, hogy ha véletlen számokkal töltjük ki a teljes együttes eloszlás táblázatát, akkor független vagy függő változókat kapunk? (Függőket!) A függetlenség tehát struktúrát takar amit tömörítésre használhatunk.

Ha a változók halmaza felosztható kölcsönösen független részhalmazokra, minden részhalmaznak lesz egy saját együttes eloszlása, majd a független részhalmazokhoz tartozó valószínűségeket egyszerűen összeszorozzuk. Pl. ha n logikai változó van, és ezek pl. két független részhalmazt alkotnak m és k mérettel ($n = m + k$), akkor $2^m + 2^k$ valószínűséget tárolunk 2^n helyett, ami sokkal kevesebb.

Extrém esetben, ha pl. az $A_1, \ldots, A_n$ diszkrét változók kölcsönösen függetlenek (tetszőleges két részhalmaz független), akkor csak $O(n)$ értéket kell tárolni, mivel ez esetben

$$P(A_1, \ldots, A_n) = P(A_1) \cdots P(A_n),$$

sőt, ha azonos eloszlásúak is emellett (pl. n kockadobás) akkor $O(1)$-et (ha a változók domainje konstans számosságú, ahogy itt általában feltesszük).

Kauzalitás és függetlenség

Fontos, hogy a *kauzalitás* (ok és okozat kérdései) és a függetlenség nem ugyanaz. A függetlenség szimmetrikus reláció a változók felett, míg az okozati kapcsolat aszimmetrikus.

Pl. a Russel-Norvig könyvben (2. kiadás) is rossz a példa! Az Időjárás és Fogfájás változóknál függetlenségre következtet abból, hogy a fogfájás nem hat kauzálisan az időjárásra. Viszont az időjárás hathat a fogfájásra, tehát nem beszélhetünk mégsem függetlenségről: a fogfájás ténye tartalmazhat egy kis információt arról, hogy milyen az idő...

9.1.8. Feltételes függetlenség

Sajnos az abszolút függetlenség ritka. Van azonban egy gyengébb értelemben vett függetlenség definíció, amit szintén használhatunk tömörítésre: a feltételes függetlenség.

Az a és b kijelentések *feltételesen függetlenek* c feltevésével akkor és csak akkor ha $P(a \wedge b|c) = P(a|c)P(b|c)$. Ekkor a és b nem feltétlenül független abszolút értelemben. Tipikus eset ha a és b közös *oka* c. Pl. a fogfájás és a beakadás közös oka a luk, a fogájás és a beakadás nem független, de ha feltesszük, hogy van luk, akkor már igen.

Az A és B véletlen változók feltételesen függetlenek C feltevésével akkor és csak akkor ha $P(A, B|C) = P(A|C)P(B|C)$, vagy ekvivalensen $P(A|B, C) = P(A|C)$, ill. $P(B|A, C) = P(B|C)$.

Az elérhető tömörítés illusztrálásához tegyük fel, hogy

$$P(A, B, C) = P(A, B|C)P(C) = P(A|C)P(B|C)P(C),$$

ahol A, B és C jelölje változók diszjunkt halmazait, az első egyenlőség a szorzatszabály, a második egyenlőség fogalmazza meg a feltételes függetlenség feltevését (ez tehát nem azonosság, hanem egy feltevés). Ekkor a $P(A|C)$, $P(B|C)$ és $P(C)$ táblázatok méreteinek az összege sokkal kisebb lehet mint az eredeti $P(A, B, C)$.

Ha nagy szerencsénk van, akkor teljesül, hogy A feltevése mellett $B_1, \ldots, B_n$ kölcsönösen függetlenek, azaz

$$P(B_1, \ldots, B_n|A) = \prod_{i=1}^{n} P(B_i|A).$$

Ez a *naiv Bayes modell* alakja. Itt $O(n)$ tömörítés érhető el, hiszen

$$P(B_1, \ldots, B_n, A) = P(A) \prod_{i=1}^{n} P(B_i|A).$$

9.1.9. Bayes szabály

A Bayes szabaly a és b kijelentésekre

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

ami egyszerűen következik abból, hogy

$$P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$$

Általában is

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

ahol A és B változók vagy változók halmazai. A képletet valamely C feltételek mellett is lehet alkalmazni, azaz

$$P(A|B, C) = \frac{P(B|A, C)P(A|C)}{P(B|C)}$$

A motiváció az, hogy sokszor a lekérdezés tünetekből kér következtetést az okozatra (pl. betegségre). A „fordított" feltételes valószínűség,

ami okozatból következtet tünetekre, azonban sokkal időtállóbb információ, kevésbé függ az aktuális dátumtól, a szakértői tudás ezért ilyen formájú. A Bayes szabály visszavezeti a fordított képletre a lekérdezést.

Pl. ha az Influenza, Fejfájás, és Dátum változókra nézzük: a $P(\text{Fejfájás}|\text{Influenza})$ empírikusan jól mérhető, viszonylag stabil érték, és nem nagyon függ az aktuális dátumtól, azaz

$$P(\text{Fejfájás}|\text{Influenza, Dátum}) = P(\text{Fejfájás}|\text{Influenza}).$$

A dátum és az influenza viszont nem független: télen több az influenza. Ezért a $P(\text{Influenza}|\text{Dátum})$ függ a dátumtól értékétől, pl. más lesz ha inluenzajárvány van, más ha nincs, stb.

Tehát van háttértudásunk ($P(\text{Fejfájás}|\text{Influenza})$) és aktuális tapasztalataink (Dátum aktuális értéke, Fejfájás=igaz, és $P(\text{Influenza}|\text{dátum})$ empírikus közelítése), és ebből következtetünk a betegségre a fejfájás tényéből az adott napon a Bayes szabály segítségével:

$$\begin{aligned} &P(\text{Influenza}|\text{fejfájás, dátum}) = \\ &= \frac{P(\text{fejfájás}|\text{Influenza, dátum})P(\text{Influenza}|\text{dátum})}{P(\text{fejfájás}|\text{dátum})} = \\ &= \frac{P(\text{fejfájás}|\text{Influenza})P(\text{Influenza}|\text{dátum})}{P(\text{fejfájás}|\text{dátum})}. \end{aligned}$$

A $P(\text{fejfájás}|\text{dátum})$ általában nem érdekes, ha csak az érdekel minket, hogy az Influenza változó igaz vagy hamis értékre nagyobb valószínűségű, ez éppen a korábban látott normalizálási konstans. Tehát

$$P(\text{Influenza}|\text{fejfájás, dátum}) = \alpha P(\text{fejfájás}|\text{Influenza})P(\text{Influenza}|\text{dátum})$$

9.1.10. Naiv Bayes algoritmus

Statisztikai következtetési módszer, amely adatbázisban található példák alapján ismeretlen példákat osztályoz. Pl. a feladat lehet emailek

osztályozása spam és nem spam kategóriákba. Ekkor az adatbázis példa emaileket tartalmaz, amelyekhez ismert az osztályozás eredménye (tudjuk mindegyikről, hogy spam van nem spam).

Legyen A és $B_1, \ldots, B_n$ a nyelvünk változói. Pl. A lehet igaz ha egy email spam, hamis ha nem, ill. B_i logikai változó pedig az i. szó előfordulását jelezheti (igaz, ha az email tartalmazza az i. szót, hamis, ha nem). A feladat tehát az, hogy adott konkrét $b_1, \ldots, b_n$ email esetében meghatározzuk, hogy A mely értékére lesz a $P(A|b_1, \ldots, b_n)$ feltételes valószínűség maximális.

Ehhez a következő átalakításokat illetve függetlenségi feltevéseket tesszük:

$$P(A|b_1, \ldots, b_n) = \alpha P(A) P(b_1, \ldots, b_n|A) \approx \alpha P(A) \prod_{i=1}^{n} P(b_i|A).$$

Itt az első egyenlőségjel a Bayes tétel alkalmazása, ahol $\alpha = 1/P(b_1, \ldots, b_n)$. Mivel csak A értékei közötti sorrendet keresünk, és α nem függ A-tól, az α értéke nem érdekes.

A második közelítő egyenlőségjel fogalmazza meg a *naiv Bayes feltevést*. Ez csak közelítés, mivel nem tudjuk biztosan, hogy az egyenlőség teljesül-e. Általában valószínűbb, hogy nem teljesül, de abban bízunk, hogy elfogadható közelítést ad. A pontatlanságért cserébe $P(A)$ és $P(b_i|A)$ könnyen közelíthető az adatbázisban található példák segítségével, így a képlet a gyakorlatban kiszámolható A minden lehetséges értékére, és a nagysági sorrend meghatározható.

10. fejezet

Valószínűségi következtetés, Bayes hálók

10.1. Bayes hálók

Láttuk: (feltételes) függetlenség hasznos mert tömöríthetjük a teljes együttes eloszlás reprezentációját.

A Bayes háló a teljes együttes eloszlás feltételes függetlenségeit ragadja meg, ezek alapján egy speciális gráfot definiál, ami és tömör és intuitív reprezentációt (vagy közelítést) tesz lehetővé.

Ennek megértéséhez tekintsük a láncszabályt, ami a teljes együttes eloszlást (amit tömöríteni szeretnénk) feltételes eloszlások szorzataként jeleníti meg. Legyen $X_1, \ldots, X_n$ a véletlen változók egy tetszőleges felsorolása, ekkor a láncszabály:

$$P(X_1, \ldots, X_n) = P(X_1|X_2, \ldots, X_n)P(X_2, \ldots, X_n) =$$
$$= P(X_1|X_2, \ldots, X_n)P(X_2|X_3, \ldots, X_n)P(X_3|X_4, \ldots, X_n) \cdots P(X_n) =$$
$$= \prod_{i=1}^{n} P(X_i|X_{i+1}, \ldots, X_n)$$

Ha most megvizsgáljuk a feltételes függetlenségi viszonyokat, az egyes feltételes valószínűségek (a szorzat tényezői) feltételeiből esetleg elhagyhatunk változókat. Ennek megfelelően, minden $P(X_i|X_{i+1},\ldots,X_n)$ tényezőre, az $\{X_{i+1},\ldots,X_n\}$ változókból vegyünk egy Szülők(X_i) részhalmazt, amelyre igaz, hogy

$$P(X_i|X_{i+1},\ldots,X_n) = P(X_i|\text{Szülők}(X_i))$$

és a Szülők(X_i) halmaz minimális (belőle több elem nem hagyható el, különben a fenti tulajdonság sérül). Ez a lépés a tömörítés, és akkor van szerencsénk ha a Szülők(X_i) halmaz nagyon kicsi.

Ekkor nyilván

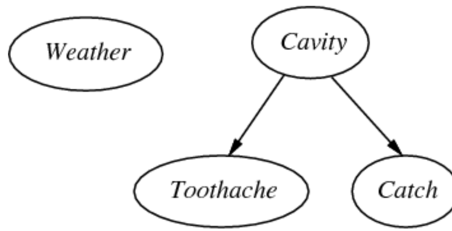
$$P(X_1,\ldots,X_n) = \prod_{i=1}^{n} P(X_i|\text{Szülők}(X_i))$$

ami a teljes együttes eloszlás egy tömörített reprezentációja, hiszen az egyes tényezőkhöz tartozó táblázatok összmérete általában sokkal kisebb mint a teljes együttes eloszlás táblázata (diszkrét változókat feltételezve).

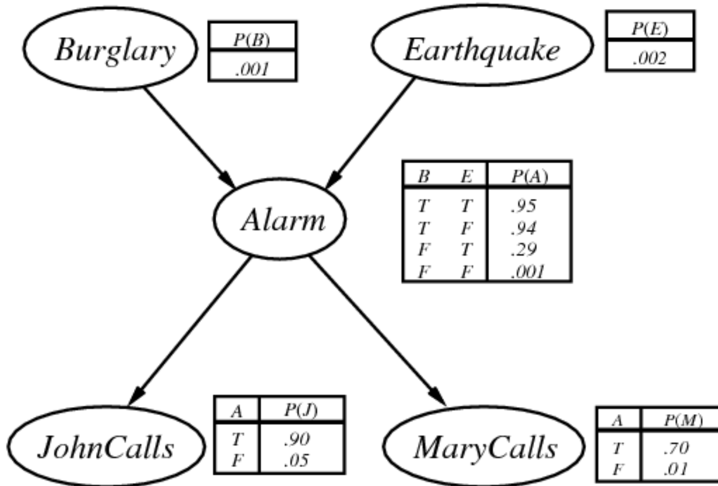
A Szülők(X_i) jelölés már a gráf struktúrára utal, amelyben a csúcsok az X_i véletlen változók, és az (Y, X_i) irányított él akkor és csak akkor van behúzva, ha $Y \in$ Szülők(X_i). A gráfban minden X_i változóhoz tartozik egy $P(X_i|\text{Szülők}(X_i))$ eloszlás.

Könnyen látható, hogy a csúcsok és az élek egy *irányított körmentes* gráfot alkotnak.

Pl. a korábbi fogászati példához tartozó gráf (eloszlások nélkül). A hálózatból pl. kiolvasható, hogy az időjárás (weather) független a többi változótól. Itt konkrétan oksági kapcsolatoknak felelnek meg az élek, de mindjárt látjuk, hogy ez nem szükségszerű.



Egy új példa eloszlásokkal együtt. Itt megint oksági kapcsolatoknak felelnek meg az élek véletlenül. B, E, A, J és M a változók neveinek rövidítései, T: igaz, F: hamis. (János és Mária a szomszédunk, akik megígérték, hogy felhívnak ha hallják a riasztónkat, ami viszont kisebb földrengésekre is beindul néha.)



Egy konkrét lehetséges világ valószínűsége pl.:

$$
\begin{aligned}
&P(B \wedge \neg E \wedge A \wedge J \wedge \neg M) = \\
&= P(B)P(\neg E)P(A|B, \neg E)P(J|A)P(\neg M|A) = \\
&= 0.001 \cdot (1 - 0.002) \cdot 0.94 \cdot 0.9 \cdot (1 - 0.7)
\end{aligned}
$$

Ha egy Bayes hálóban X-ből Y csúcsba vezet él, akkor Y és X nem független. Viszont nem igaz, hogy ha X és Y nem független, akkor van köztük él. Sőt, egy eloszláshoz sok különböző Bayes háló adható. Az sem igaz, hogy az élek szükségképpen kauzális relációt fejeznének ki!

10.1.1. Bayes háló konstruálása

A *gyakorlatban* sokszor egy terület (pl. orvosi diagnosztika) szakértője definiálja a változókat és a hatásokat (oksági) a változók között (ez adja az éleket), és szakértői és empirikus tudás segítségével kitöltjük a változókhoz tartozó feltételes eloszlásokat.

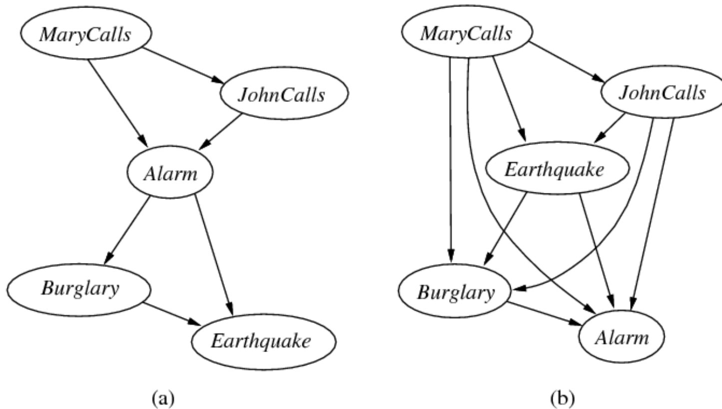
Ekkor természetesen nem az eloszlásból indulunk ki és azt tömörítjük, hanem fordítva, először az intuitív reprezentáció adott, ami definiál egy (implicit) teljes együttes eloszlást, amit hatékonyan felhasználhatunk valószínűségi következtetésre.

Elméleti szempontól bármely adott együttes eloszlásra konstruálható Bayes háló a korábban tárgyalt láncszabály módszerrel. (Veszzük a láncszabályt, azonosítjuk a szülőket a függetlenség vizsgálatával, stb.)

Fontos: a láncszabály alkalmazásakor használt, rögzített változósorrendtől függ a Bayes háló: bármely sorrendre kijön egy struktúra, ami nem feltétlenül ugyanaz, tehát a kauzalitásnak nincs kitüntetett szerepe.

Pl. az előző példához tartozó Bayes háló E, B, A, J, M sorrend esetén az (a) alakot veszi fel, míg A, B, E, J, M sorrendben a (b) hálózatot kapjuk. Az (a) hálózatban megfordulnak az oksági viszonyok, míg a (b) hálózatban egyáltalán *nincs* tömörítés.

Kritikus fontosságú tehát a változósorrend.



10.1.2. Függetlenség és Bayes hálók

A topológiát bizonyos függetlenségek felhasználásával kreáltuk (tömörítési lépés), de nem világos, hogy ha egyszer megvan egy topológia, abból hogyan lehet további függetlenséggel kapcsolatos információkat kiolvasni.

Két eredmény bizonyítás nélkül:

Először, ha Y *nem leszármazottja* X-nek, akkor

$$P(X|\text{Szülők}(X), Y) = P(X|\text{Szülők}(X)).$$

Másodszor, bármely Y változóra igaz, hogy

$$P(X|\text{Markov-takaró}(X), Y) = P(X|\text{Markov-takaró}(X)),$$

ahol X *Markov takarója* az a halmaz, amely X szülőinek, X gyerekeinek, és X gyerekei szülőinek az uniója.

10.1.3. További tömörítési lehetőségek

A lényeg, hogy egy Bayes hálóban a csúcsokban található eloszlások általában kisebb táblázatok mint a teljes táblázat. Pl. ha n logikai változó

van, és mindegyiknek max. k szülője van, akkor max. $n \cdot 2^k$ valószínűséget kell összesen tárolni, és nem 2^n-t. Ez nagyon nagy különbség, ha k kicsi. Sokszor azonban még tovább lehet tömöríteni.

Ha pl. determinisztikus változók vannak, akkor azoknak az értéke a szülőiknek egyszerű függvénye. Lehet logikai, pl. $X = Y \wedge Z$, vagy numerikus, pl. $X = \min(Y, Z)$, a változók típusának a függvényében.

Determinisztikus kapcsolat helyett pl. logikai változók esetében k szülő esetén 2^k helyett k érték tárolása elegendő lehet bizonyos alkalmazásokban. Pl. a *zajos vagy* eloszlást a táblázat szemlélteti, ahol Szülő(Láz) = {Malária,Influenza,Megfázás} (csak a vastagbetűs értékeket kell tárolni).

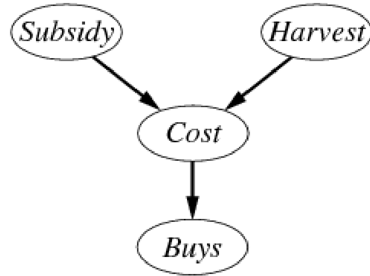
Itt tulajdonképpen azt tettük fel, hogy minden betegség esetében a láz gátlódhat, és ennek a gátlásnak a valószínűsége különbözik, ill. a gátlások függetlenek.

Másfelől nézve, ez ekvivalens azzal, mintha (1) felvennénk három új változót: MaláriaLáz, InfluenzaLáz, és MegfázásLáz, mindegyiket egy szülővel: Malária, Influenza, ill. Megfázás; (2) majd a Láz változót determinisztikusan így definiálnánk: Láz $\leftrightarrow$ MaláriaLáz $\vee$ InfluenzaLáz $\vee$ MegfázásLáz

Malária	Influenza	Megfázás	P(láz)	$P(\neg$láz)
H	H	H	0.0	1.0
H	H	I	0.9	**0.1**
H	I	H	0.8	**0.2**
H	I	I	0.98	0.02 = 0.2 x 0.1
I	H	H	0.4	**0.6**
I	H	I	0.94	0.06 = 0.6 x 0.1
I	I	H	0.88	0.12 = 0.6 x 0.2
I	I	I	0.988	0.012 = 0.6 x 0.2 x 0.1

10.1.4. Folytonos változók

Példa a folytonos változót tartalmazó eloszlásra. Itt a Termés (Harvest) és az Ár (Cost) változók folytonosak, a Támogatás (Subsidy) és Vásárlás (Buys) diszkrét (logikai). Függvényekkel közelítjük a változók feltételes eloszlásait.



Két eset van: a változó folytonos vagy diszkrét.

1. Először nézzük a P(Ár|Támogatás,Termés) esetet. Itt megadjuk a P(Ár|támogatás,Termés) és P(Ár|¬támogatás,Termés) eloszlásokat, mindegyik folytonos sűrűségfüggvény lesz. Pl. lehet

   P(Ár|támogatás,Termés)$= N(a_i\text{termés} + b_i, \sigma_i^2)$(Ár)

   P(Ár|¬támogatás,Termés)$= N(a_h\text{termés} + b_h, \sigma_h^2)$(Ár)

   ahol $N(\mu, \sigma^2)(x)$ a μ, σ^2 paraméterű normális eloszlás sűrűségfüggvénye. Más szóval a modellünk az, hogy az ár várható értéke a termés lineáris függvénye az a_i, b_i (ha volt támogatás), illetve az a_h, b_h (ha nem volt támogatás) paraméterekkel, továbbá normális eloszlású, σ_i illetve σ_h paraméterrel meghatározott szórású zaj van ezen a lineáris függvényen.

2. Másodszor nézzük a diszkrét esetet: P(Vásárlás|Ár). Intuitíve, ha az ár magas, a vevő nem vásárol, ha alacsony, igen, a kettő között pedig szeretnénk folytonos átmenetet. Ekkor lesz egy *küszöbérték* amelynél a vevő éppen 0.5 valószínűséggel fog vásárolni.

   Tehát a P(vásárlás|Ár=c) feltételes eloszlást kell egy *küszöbfüggvény* formájában definiálni, ami 0 ha c kicsi, 1 ha c nagy, és egy $c = \mu$ értéknél veszi fel a 0.5 értéket.

Vehetjük a *szigmoid* függvényt:

$$P(\text{vásárlás}|\text{Ár} = c) = \frac{1}{1 + \exp(\frac{-c+\mu}{\sigma})}$$

ahol a $\mu = c$ esetén vesz fel 0.5 értéket a modellünk, és a σ paraméter a bizonytalansági zóna szélességét határozza meg (nagy σ esetén a μ nagy szomszédságában lesz az érték 0.5-höz közeli).

10.2. Valószínűségi következtetés

Már láttuk, hogy

$$P(A|b) = \alpha P(A, b) = \alpha \sum_x P(A, b, x)$$

ahol α a normalizálási konstans, b az ismert tények, x az ismeretlen tények (változók) lehetséges értékei, A pedig az érdeklődésünk tárgya.

Azt mondtuk, hogy ez kiszámolható, ha a teljes együttes eloszlás ismert. Először megnézzük, hogy a Bayes hálóból ez hogyan tehető meg *egzakt* módon, aztán *közelítési* módszereket nézünk.

Például a betörős példában

$$P(B|j, m) = \alpha P(B, j, m) = \alpha \sum_e \sum_a P(B, j, m, e, a)$$

ahol (mint eddig) a kisbetű értéket, a nagybetű változót jelöl, e és a pedig az E és az A értékein fut végig. Használjuk a Bayes hálót a „betörés, feltéve, hogy János és Mária hívott” tényállás valószínűségének a kiszámolására:

$$P(b|j, m) = \alpha P(b, j, m) = \alpha \sum_e \sum_a P(b, j, m, e, a) =$$

$$= \alpha \sum_e \sum_a P(b)P(e)P(a|b, e)P(j|a)P(m|a)$$

Ennek a műveletigénye a legrosszabb esetben (ha majdnem minden változóra összegzünk) $O(n2^n)$ (max 2^n db n tényezős szorzat). Viszont kiemeléseket végezhetünk:

$$\alpha P(b,j,m) = \alpha P(b)\sum_e P(e)\sum_a P(a|b,e)P(j|a)P(m|a)$$

Ennek a kiértékelése pl. megoldható egy rekurzív algoritmussal („balról jobbra” vesszük a változókat, és mindegyiknek vesszük a lehetséges értékeit), aminek a műveletigénye $O(2^n)$ (fabejárás), ami még mindig elég nagy.

(Kiszámolva $\neg b$-re is, és normalizálva, az jön ki, hogy $P(b|j,m) = 0.284$.)

Még hatékonyabban is kiszámolhatjuk ezt az összeget *dinamikus programozás* segítségével. Ehhez két ötletet használunk fel (vázlatos):

1. A kiemelések után kapott képletet „jobbról balra” számoljuk ki, azaz a részeredményeket csak egyszer, és ezeket a memóriában tároljuk, amivel a redundancia megszűnik. A memóriaigény sem nő mindig mert a részeredményekből néha kiösszegezünk változókat (amikor jobbról balra egy $\sum$ jelhez érünk).

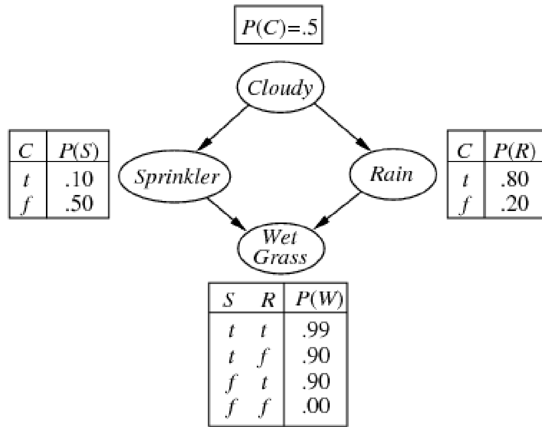
2. Bizonyos változókat kihagyhatunk teljesen a képletből (nem kell összegezni rájuk), konkrétan azokat amik nem a kérdés változójának (fenti példában B) vagy az ismert értékű változóknak, azaz tényeknek (fenti példában j és m) az őse. Ezek irrelevánsak.

Ennek az algoritmusnak a hatékonysága nagyban függ a Bayes háló struktúrájától. Ha a gráf *polifa (erdő)*, azaz az élek irányításának elhagyásával nincs benne kör, akkor lineáris idejű az algoritmus a feltételes eloszlás táblázatok összes sorainak a számában.

Általános esetben #P-nehéz (számosság-P-nehéz), nehezebb mint az NP-nehéz problémák, olyan nehéz mint pl. a kielégíthetőségi problémánál a kielégítő megoldások *számának* a megadása.

10.3. Közelítő valószínűségi következtetés

Egy új példa eloszlásokkal együtt. C, S, W, R a változók neveinek rövidítései (sprinkler=locsoló), T: igaz, F: hamis. Ez a Bayes hálók klasszikus példája (a magyar kiadásban az i/h felcserélve).



10.3.1. Közvetlen mintavétel

Cél: (1) Teljes együttes eloszlás szerint generálni változó hozzárendeléseket véletlenszám generátorral (2) Ezekből a mintákból következtetni a lekérdezés valószínűségére a gyakoriságok megfigyelésének a segítségével.

Hogyan? Kezdjük a szülő nélküli változókkal, és haladjunk „lefelé”. Pl. kövessük nyomon egy konkrét változóhozzárendelés generálását. (1) $P(c) = P(\neg c) = 0.5$, tehát pénzfeldobással döntünk, legyen a hozzárendelés c (azaz C=igaz). Ekkor (2) $P(s|c) = 0.1 = 1 - P(\neg s|c)$, tegyük fel, hogy $\neg s$-et húzunk. (3) Az r változó esete teljesen hasonló, tegyük fel, hogy ott r-t húzunk. (4) Végül tudjuk, hogy $P(w|\neg s, r) = 0.9$, tegyük fel, hogy w-t húzunk.

A minta amit vettünk tehát $(w, \neg s, r, c)$. Ezt ismételjük addig, amíg a lekérdezéshez szükséges gyakoriságok közelítéséhez elegendő minta lesz, hiszen

$$P(w, \neg s, r, c) = \lim_{N \to \infty} \frac{N(w, \neg s, r, c)}{N}$$

ahol N az összes minta száma, $N(w, \neg s, r, c)$ pedig a megadott hozzárendelést tartalmazó minták száma.

Nem teljes hozzárendelés valószínűségét is ugyanígy közelítjük, pl.

$$P(w) = \lim_{N\to\infty} \frac{N(w)}{N} = \lim_{N\to\infty} \frac{\sum_{S,R,C} N(w,S,R,C)}{N}$$

10.3.2. Elutasító mintavétel

Ha egy eloszlásból nem tudunk közvetlenül mintát venni, akkor sokszor megtehetjük, hogy először veszünk mintát egy másik eloszlásból, és néhány mintát „kidobunk”. Pl. a $P(w|c)$ feltételes valószínűséghez tartozó gyakoriság közelítéséhez eldobjuk azokat a mintákat amelyekben $\neg c$ szerepel. A maradék minták számát jelölje N' ill. $N'()$. Ekkor

$$P(w|c) = \lim_{N'\to\infty} \frac{N'(w)}{N'}$$

10.3.3. Valószínűségi súlyozás (likelihood weighting)

Feltételes valószínűség közelítésénél az elutasító mintavétel eldobja azokat a mintákat amik nem konzisztensek a feltételekkel. Ehelyett sokkal okosabbat is lehet tenni: ha egy változó értéke a feltételben szerepel (más szóval ismert tény) akkor nem véletlenül húzzuk az értékét, hanem helyette *súlyozzuk* a mintát az ismert feltételes valószínűségekkel.

Pl. tegyük fel, hogy ismert tény, hogy s és w, és ezzel konzisztens mintát akarunk venni. Az elutasító módszer vesz egy teljes mintát közvetlen mintavétellel és eldobja ha $\neg s$ vagy $\neg w$ szerepel benne. Ehelyett így csináljuk: Legyen a minta súlya kezdetben $x = 1$. (1) $P(c) = P(\neg c) = 0.5$, tehát pénzfeldobással döntünk, legyen a hozzárendelés c (azaz C=igaz). Ekkor (2) $P(s|c) = 0.1 = 1 - P(\neg s|c)$. De most tudjuk, hogy s. Ennek a valószínűsége 0.1 lett volna, tehát megjegyezzük: $x = x \cdot 0.1 = 0.1$. (3) Az r változónak mintavétellel kell

értéket adni, az alapján, hogy $P(r|c) = 0.8 = 1 - P(\neg r|c)$, tegyük fel, hogy r-t húzunk. (4) Végül, tudjuk, hogy w. De ennek a valószínűsége $P(w|s, r) = 0.99$ lett volna, ezt is megjegyezzük: $x = x \cdot 0.99 = 0.099$.

A konkrét minta amit vettünk tehát (w, s, r, c), a súlya pedig 0.099. Ezt ismételjük addig, amíg a lekérdezéshez szükséges gyakoriságok közelítéséhez elegendő minta lesz. Legyenek $m_1, \ldots, m_N$ a minták, $w(m_i)$ pedig az i. minta súlya. Ekkor pl.

$$P(c, r|s, w) = \lim_{N\to\infty} \frac{\sum_{m_i\text{-ben C=R=igaz}} w(m_i)}{\sum_{i=1}^{N} w(m_i)}$$

$$P(\neg c|s, w) = \lim_{N\to\infty} \frac{\sum_{m_i\text{-ben C=hamis}} w(m_i)}{\sum_{i=1}^{N} w(m_i)}$$

Így csak releváns mintákat generálunk, *de* ha túl sok a tény, és/vagy a tények túlságosan „lent” vannak a Bayes hálóban, akkor nagyon sok nagyon kis valószínűségű (súlyú) mintát fogunk generálni.

10.4. Markov lánc Monte Carlo (MCMC)

Nem változónkénti mintavétel, hanem teljes változóhozzárendelések feletti okosan tervezett véletlen séta (miközben a tények fixen maradnak). Ennek a véletlen sétának az állomásai lesznek a minták amiket használunk. (Amit nézünk az a Gibbs mintavételezési eljárás.)

Pl. tegyük fel, mint az előbb, hogy ismert tény, hogy s és w, és ezzel konzisztens mintát akarunk venni. Ekkor

1. Az ismeretlen változókat (R és C) tetszőlegesen inicializáljuk, így kapunk egy kezdeti állapotot. Pl. legyen ez $(s, w, \neg r, c)$.

2. A C változóból mintát veszünk a

$$P(C|\text{Markov-takaró}(C)) = P(C|s, \neg r)$$

eloszlásnak megfelelő valószínűségekkel. Tegyük fel, hogy az eredmény $\neg c$. Ekkor az új állapot $(s, w, \neg r, \neg c)$.

3. Az R változóból mintát veszünk a

$$P(R|\text{Markov-takaró}(R)) = P(R|\neg c, s, w)$$

eloszlásnak megfelelő valószínűségekkel. Tegyük fel, hogy az eredmény r. Ekkor az új állapot $(s, w, r, \neg c)$.

A fenti algoritmus során a (2) és (3) lépésben is generáltunk egy mintát. Az ismeretlen változók újramintavételezését (fenti példában (2) és (3) lépés) ismételjük addig, amíg a lekérdezéshez szükséges gyakoriságok közelítéséhez elegendő minta lesz. A minták felhasználása megegyezik az elutasító mintavételnél látottakkal.

Jegyezzük meg, hogy egy X változóra $P(X|\text{Markov-takaró}(X))$ kiszámolható a Bayes hálóból:

$$P(X|\text{Markov-takaró}(X)) =$$
$$= \alpha P(X|\text{Szülők}(X)) \prod_{Y\,\text{gyereke}\,X\text{-nek}} P(Y|\text{Szülők}(Y))$$

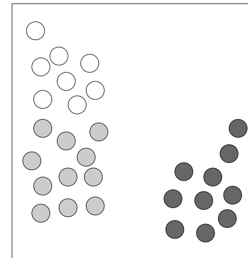
Miért működik? Ötlet: bizonyítható, hogy az iteráció során egy „dinamikus egyensúly” alakul ki, ahol minden állapot (azaz minden, tényekkel konzisztens, teljes változókiértékelés) éppen a valószínűségének megfelelő gyakorisággal lesz érintve. Ez pedig abból következik, hogy az állapotok közötti átmenetek valószínűségeit tartalmazó állapotátmenet mátrix domináns sajátvektora lesz az az (ún stacionárius) eloszlás (bizonyos gyenge feltételek mellett) ami a fenti dinamikus egyensúlyi helyzetet definiálja, márpedig az állapotátmenetek definíciójából belátható, hogy ez éppen a megfelelő eloszlás.

11. fejezet

Tanulás

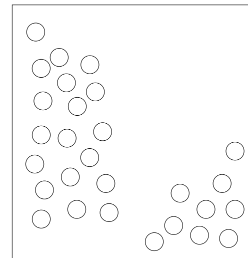
Tapasztalati (megfigyelt) tények felhasználása arra, hogy egy racionális ágens teljesítményét növeljük.

Felügyelt tanulás: Egy $f\colon X \to Y$ függvényt keresünk, amely illeszkedik adott páldákra. A példák $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$ alakban adottak ($x_i \in X$). Pl. (a) X: emailek halmaza, $Y = \{$spam,$\neg$spam$\}$, a példák pedig kézzel osztályozott emailek (spam szűrő tanítása), vagy (b) X: részvény-index idősorai, $Y = \{$emelkedik,esik$\}$, (tőzsdéző program), stb.



felügyelt tanulás

Felügyelet nélküli tanulás: a példák csak $x_1, \ldots, x_n$ alakban adottak ($x_i \in X$), és nem függvényt kell keresni, hanem mintázatokat, pl. klasztereket, vagy eloszlást.



felügyelet nélküli tanulás

Megerősítéses tanulás: Szokás külön kategóriának venni a megerősítéses tanulást, ahol az $X \to Y$ függvény létezik ugyan, de tanuló példák nem adottak direkt módon. A feladat itt az, hogy egy állapottérben az ágens megtanulja az optimális stratégiát (X az állapotok, Y pedig a cselekvések) úgy, hogy a jövőben érintett állapotokban összegyűjtött jutalmak maximalizálását érje el.

Számos finomabb megkülönböztetés van, pl. felügyelt tanulás esetén a példák folyamatosan gyűlnek tanulás közben, vagy akár az ágens kérdéseket tehet fel egy orákulumnak (aktív tanulás), stb.

Reprezentáció: Az X és Y halmazok tetszőleges objektumokat írhatnak le, és fontos, hogy ezeket hogyan reprezentáljuk. Pl. szövegeknek sokfajta reprezentációja lehet a tartalmazott szavak listájától (azaz szósaláta) a folytonos szemantikus beágyazásokig. Az Y halmaz jellemzően diszkrét osztálycímkéket tartalmaz (pl. spam vagy nem spam, stb.), illetve lehet folytonos halmaz is (ekkor regresszióról beszélünk).

11.1. Felügyelt (induktív) tanulás

Egyszerű függvényközelítéstől magáig a tudományig sok mindent lefed.

Adott egy $f\colon X \to Y$ függvény és $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$ tanuló példák. Keressünk egy $h\colon X \to Y$ függvényt amely f-et közelíti.

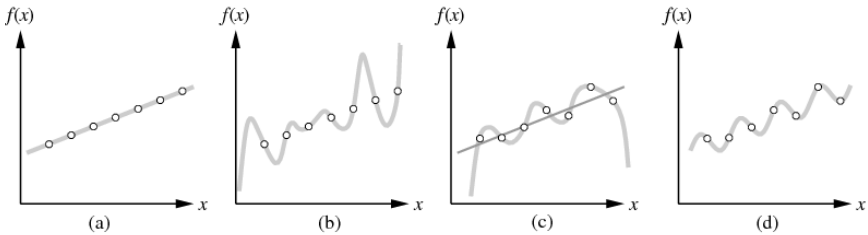
A h függvény *konzisztens* az adatokkal, ha $h(x_i) = f(x_i)$ minden példára.

A h függvényt mindig egy H hipotézistérben keressük. Más szóval, a függvényt mindig adott alakban keressük, pl. adott fokszámú polinom, Boole függvények adott osztálya, stb.

A tanulás *realizálható*, ha létezik $h \in H$, amelyre h konzisztens.

A gyakorlatban elég, ha h „közel” van a példákhoz, nem kell pontosan illeszkednie feltétlenül. Ennek egyik oka, hogy a tanuló példák sokszor

hibákat, zajt is tartalmaznak, és kifejezetten káros is lehet, ha ezeket a hibákat is meg akarjuk tanulni (azaz *túltanuljuk* a tanító adatokat).



(a) és (b) ábra: az általánosabb (azaz bővebb) H lehet akár a rosszabb választás is. (c) és (d) ábra: az is fontos hogy H tartalmazza a megfelelő hipotéziseket (pl. akármilyen általános polinom osztály nem lesz periódikus soha valós számok felett).

Indukció problémája: f jó közelítése olyan amely a példákon *kívül* is jól közelíti f-et, azaz jól *általánosít*. Ez nehéz és nagyon érdekes kérdés!

Pl. az a h, amelyre $h(x) = f(x)$ minden példára, egyébként $h(x) = 0$, tökéletesen megtanulja a példákat, de a lehető legrosszabban általánosít (általában...). Ez a magolás (rote learning).

A magolási probléma miatt *tömör* reprezentációra—azaz kevés paraméterrel leírható h-ra—kell törekedni, lehetőleg tömörebbre mint a példák listája, így biztosan kiküszöböljük a magolást. Ez az elv az *Occam borotvája*: ha más alapján nem tudunk választani, akkor a lehető legtömörebb leírást kell venni. (De: vannak listák amik nem tömöríthetők (véletlen listák, Kolmogorov komplexitás), ezeknél az indukció lehetetlen.) Ugyanakkor elég kifejezőnek is kell lennie a reprezentációnak, hogy a tanító példákat elég jól közelíteni tudjuk.

A fenti tulajdonságok elérésének egyik eszköze a H hipotézistér gondos meghatározása. Az ideális hipotézistér—amellett, hogy tartalmazza f-et (a tanulás realizálható)—éppen annyira általános (kifejező) amennyire kell (nem jobban, nem kevésbé). A masik eszköz maga a tanuló

algoritmus, amely a H-ból h-t kiválasztja. Itt az algoritmus törekedhet tömören leírható h-t választani akár egy nagyon általános H-ból is.

Az *a priori ismeretek* fontossága: a *tabula rasa* tanulás a fentiek szerint lehetetlen. A H halmaz és az algoritmus maga a priori ismeretek alapján kerülnek megtervezésre.

Emellett *számítási szempontból egyszerű* reprezentáció és hipotézistér is kell a hatékonyság miatt. A túl egyszerű reprezentáció azonban sokszor komplex hipotéziseket eredményez, pl. predikátumlogika vs. ítéletlogika.

11.2. Döntési fák

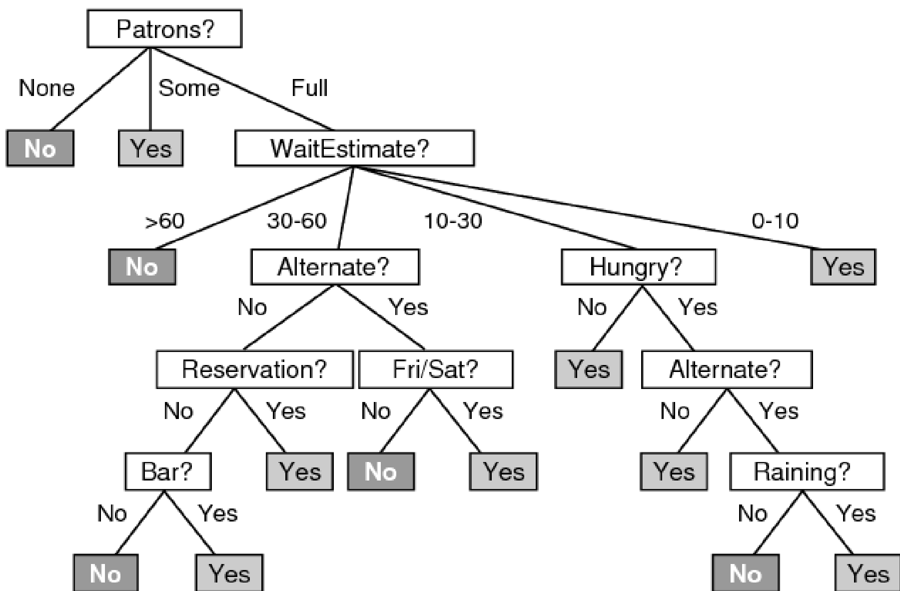
Az induktív tanulás egy konkrét példája. Feltesszük, hogy $x \in X$ diszkrét változók értékeinek vektora, és $f(x) \in Y$ egy diszkrét változó egy értéke (pl. $Y = \{\text{igen,nem}\}$).

Mivel Y véges halmaz, *osztályozási feladatról* beszélünk, ahol X elemeit kell osztályokba sorolni, és az osztályok Y értékeinek felelnek meg. (Ha Y folytonos, akkor *regresszióról* beszélünk.)

Feladat: döntsük el, hogy érdemes-e asztalra várni egy étteremben. Döntési probléma (más néven kétosztályos osztályozási feladat), azaz $Y = \{\text{igen,nem}\}$. Az X halmaz pedig a szituáció leírása változók értékeinek a segítségével. Minden változó diszkrét. Egy lehetséges $(x, f(x))$ példa:

((Vedégek=tele, Várakozás=10-30, Éhes=igen, VanMásHely=nem, Esik=igen, Foglalás=nincs, Péntek/Szombat=igen, VanBár=igen), igaz)

Minden példában minden változó értéket kap, a példából tehát látszik, hogy a nyelvünk milyen változókat tartalmaz. Más szóval a feladat példahalmaza elemi eseményekből, azaz lehetséges világokból áll.



A döntési fa előnye, hogy a döntései megmagyarázhatók, mert emberileg értelmezhető lépésekben jutunk el a döntésig. Pl. a mesterséges neurális hálókra ez nem igaz.

11.2.1. Kifejezőerő

A kifejezőerő éppen az ítéletkalkulus. Feltesszük az egyszerűség kedvéért, hogy a címke logikai érték (igen, nem).

1. Ítéletek: változó értékadások, pl. Vendégek=senki, Éhes=igen, stb.

2. Modell: egy $x \in X$ változóvektor egy modell mert minden ítélet igazságértékét megadja

3. Formula: a döntési fából logikai formula gyártható, és fordítva.

Fa ⇒ formula: vesszük az összes olyan levelet amelyen az igen címke van, és az oda vezető utakban „és”-sel összekötjük az éleket, és az utakat „vagy”-gyal összekötjük. A fenti példát pl. így:

$$\text{Fa} \equiv (\text{Vendégek=néhány}) \vee (\text{Vendégek=tele} \wedge \text{Várakozás=0-10}) \vee \dots$$

Formula ⇒ fa: a logikai formula igazságtábláját fel lehet írni fa alakban, ha vesszük a változók egy $A_1, \dots, A_n$ felsorolását, az A_1 a gyökér, A_1 értékei az élek, és az i. szinten a fában minden pontban A_i van, amely pontokból az élek A_i értékei. Az A_n változóból kivezető élek már levelekbe vezetnek, amelyeket az igazságtáblában található értékkel címkézünk. Ezt az értéket a levélből a gyökérbe vezető út élei által meghatározott igazságtábla-sor határozza meg.

A fenti naiv faépítés nagy fákat eredményez. Kérdés: tömöríthető-e a reprezentációja egy formulának? Néha nem, pl. paritásfüggvény, vagy nem nagyon, pl. többségi függvény. Viszont a gyakorlatban általában lehet tömöríteni, éppen ez lesz a célunk. Pl. ha egy változótól nem függ az igazságérték, akkor egyáltalán nem kell szerepelnie a döntési fában.

11.2.2. Döntési fa építése

Adottak pozitív (igaz) és negatív (hamis) példák ahol minden változó értékét megadjuk, a példa címkéjével együtt, pl.:

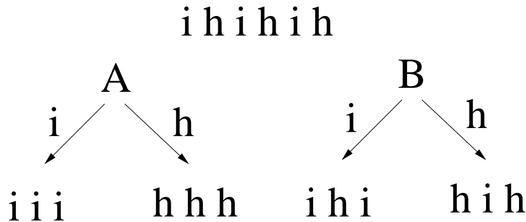
((Vedégek=tele, Várakozás=10-30, Éhes=igen, VanMásHely=nem, Esik=igen, Foglalás=nincs, Péntek/Szombat=igen, VanBár=igen), igaz)

Ilyen példákból adott tipikusan minimum több száz.

A példákat „bemagolni” könnyű: pl. tekinsük a példákat az igazságtábla ismert sorainak, és a formula → fa átalakításnál leírt módon, a változók sorbarendezésével, építhetünk fát, ahol az ismeretlen igazságtábla sorokhoz véletlen igazságértéket írunk. Ez egy olyan fa lesz amely konzisztens a példákkal.

De ez magolás, nem fog általánosítani: tehát tömöríteni kellene.

Ötlet: a gyökérbe vegyük azt a változót amelyik a legtöbb információt hordozza (legjobban szeparálja a pozitív és negatív példákat).



Ezután a gyökér minden rögzített változóértékére és a hozzá tartozó példa-részhalmazra rekurzívan ugyanez, tehát az adott részhalmazon a még nem rögzített változók közül gyökeret választunk a részhalmazt leíró részfához, stb.

A speciális esetek amik megállítják a rekurziót:

1. ha csak pozitív vagy negatív példa van, akkor levélhez értünk, megcímkézzük megfelelő módon
2. ha üres halmazról van szó (ez akkor lehet ha többértékű változó a szülő): alapértelmezett érték, pl. a szülőben a többségi döntés
3. ha pozitív és negatív példa is van, de nincs több változó (ez akkor lehet ha zajjal terheltek (nem konzisztensek) a példák): ekkor a többségi szavazattal címkézhetjük a levelet

11.2.3. A legjobban szeparáló attribútum

Ötlet: adott változó ismerete megnöveli az *információt* arról, hogy egy példa „igen” vagy „nem”.

Információ: információelméleti értelemben egy p valószínűségű esemény *információtartalma* definíció szerint $-\log p$. Ha $\log_2$-t használunk, mértékegysége a bit.

Egy $p_1, \ldots, p_n$ valószínűségi eloszlás várható (átlagos) információtartalma, más néven *entrópiája*

$$H(p_1, \ldots, p_n) = -\sum_i p_i \log p_i, \;\text{ ahol } \sum_i p_i = 1,$$

amelynek minimuma 0, ami a maximális rendezettséget jelöli, amikor ponosan egy i-re $p_i = 1$ és $p_j = 0, i \neq j$. A maximuma pedig $-\log(1/n) = \log n$, ez a maximális redezetlenség állapota.

Legyen egy példahalmazban n^+ pozitív és n^- negatív példa. Ekkor a példahalmaz entrópiája

$$H\left(\frac{n^+}{n^+ + n^-}, \frac{n^-}{n^+ + n^-}\right) = \\ -\frac{n^+}{n^+ + n^-}\log\frac{n^+}{n^+ + n^-} - \frac{n^-}{n^+ + n^-}\log\frac{n^-}{n^+ + n^-}.$$

Az *információnyereség* egy A változóra nézve ugyanebben a példahalmazban a következő:

$$\text{Nyereség}(A) = \\ H\left(\frac{n^+}{n^+ + n^-}, \frac{n^-}{n^+ + n^-}\right) - \sum_i \frac{n_i^+ + n_i^-}{n^+ + n^-} H\left(\frac{n_i^+}{n_i^+ + n_i^-}, \frac{n_i^-}{n_i^+ + n_i^-}\right),$$

ami nem más mint a példahalmaz entrópiájának és az A változó lehetséges értékei szerinti felosztás után keletkező részhalmazok átlagos entrópiájának a különbsége, ahol n_i^+ és n_i^- az A változó i. lehetséges értékét tartalmazó pozitív ill. negatív példák száma.

Ennek fényében válasszuk a maximális nyereségű változót a gyökérbe.

Zajszűrés az attribútumválasztásban

Említettük a magolás problémáját. Hasonló (de nem ugyanaz) a *túlillesztés* problémája, ahol „túlságosan pontosan” illesztjük az adatokra a

modellt, ami akkor fordul elő, ha túl általános (nagy kifejezőerejű) a modellünk (pl. magas fokszámú polinom szemben a lineáris közelítéssel), vagy ha túl kevés a példa. Ilyenkor már általában a zajt reprezentáljuk, nem a lényeget, és az általánosítási képesség ezért csökken.

Pl. ha dobókockával dobjuk a példákat, és lejegyzünk irreleváns attribútumokat, pl. a kocka színét, a dobás idejét, stb. Az irreleváns attribútumok információnyeresége elméletben nulla. A gyakorlatban viszont nem (véges számú minta), tehát felépül egy döntési fa, ami értelmetlen: túlillesztettük.

Zajt szűrhetünk úgy, hogy megnézzük, hogy az információnyereség *statisztikailag szignifikáns-e*?

Pl. χ^2 próba: legyen a kocka színe szerinti felosztásban az i színű kockához tartozó minták száma n_i^+ és n_i^-, $\sum_i n_i^+ = n^+$, $\sum_i n_i^- = n^-$. A null hipotézishez tartozó pozitív és negatív példák száma legyen

$$\hat{n}_i^+ = n^+ \frac{n_i^+ + n_i^-}{n^+ + n^-}, \text{ és } \hat{n}_i^- = n^- \frac{n_i^+ + n_i^-}{n^+ + n^-}.$$

A null hipotézis szerint tehát minden színre a pozitív és negatív példák aránya ugyanaz mint a felosztás előtt. Ekkor a

$$D = \sum_i \frac{(n_i^+ - \hat{n}_i^+)^2}{\hat{n}_i^+} + \frac{(n_i^- - \hat{n}_i^-)^2}{\hat{n}_i^-}$$

statisztika értékét kell a χ^2 eloszlással összevetni, és ha nem tudunk a null hipotézis elvetése mellett dönteni, akkor nem osztunk fel az adott változó szerint.

Ez az algoritmus a χ^2 *metszés*.

Néhány gyakorlati megjegyzés

Hiányos adatok: Egyes példákból hiányozhatnak bizonyos attribútumok értékei. Ekkor pl. minden lehetséges értéket behelyettesítve (eset-

leg a többi példában szereplő gyakoriságnak megfelelően) felvehetünk új példákat.

Sokértékű attribútumok: Nagy információnyereség azért mert sok az érték (nem feltétlenül a jó szeparáció miatt). Vannak módszerek arra, hogy normalizáljuk a nyereséget magában az attribútumban lévő információmennyiséggel.

Folytonos változók: Építhetünk fát, de ehhez diszkretizálni kell a változót. A diszkretizáció minden pontban, ahol az adott változó szerint szeparálunk, lehet akár más, pl. optimalizálhatjuk vele az elérhető nyereséget.

Folytonos kimenet: Ez már sok trükköt igényel: *regressziós fa*, ahol a leveleken egy függvény van, amely a levélhez vezető úton lévő változóktól függ.

11.3. Induktív tanulóalgoritmusok kiértékelése

Hogyan buktatjuk le a magolókat és a túlillesztést?

Legegyszerűbb módszer: a példákat két részre osztjuk: (1) *tanuló (training)* halmaz és (2) *teszt* halmaz. A tanuló halmazon állítjuk elő a hipotézist (pl. döntési fát), és a teszt halmazon értékeljük ki.

A kiértékelés mérőszáma sokféle lehet. Legegyszerűbb a *pontosság (accuracy)*, ami a jól osztályozott példák százalékos aránya. Ennek egy hibája, hogy pl. ha a példák túlnyomó többsége valamelyik osztályba tartozik, pl. pozitív, akkor nagy pontosságot kapunk akkor is, ha mindenre vakon azt mondjuk, hogy pozitív. Vannak egyéb mértékek amik ezt kezelik, itt nem tárgyaljuk.

Keresztvalidáció (cross validation): Hogy lehet a leghasznosabban felhasználni a példákat tesztelésre? Sokféle tanuló/teszt felosztás kell:

1. Osszuk fel a példahalmazt k egyenlő részre.
2. Építsünk k modellt a következőképpen: egy modellhez vegyünk egy részhalmazt teszthalmaznak, és a maradék $k-1$ részhalmaz unióján tanítsuk a modellt.
3. Értékeljük ki mindegyik modellt a saját teszthalmazán, és vegyük az értékelések átlagát.

Így minden példa tanuló és teszt példa is egyben. A leginformatívabb az az eset amikor k éppen a példák száma (leave-one-out cross validation), de ez nagyon időigényes. Leggyakrabban $k = 10$.

Kukucskálás (peeking) probléma: Hiába szeparáljuk a teszt halmazt, illetve hiába alkalmazunk keresztvalidációt, ha a teszt halmazon vagy keresztvalidáció során látott teljesítmény alapján optimalizáljuk az algoritmusunk paramétereit, akkor implicit módon a teszthalmaz is hatással lesz, és onnantól fogva nem tudjuk, hogy az algoritmus milyen jól általánosít. Elvben minden paraméterbeállításnál új teszthalmaz kellene.

Ennek kezelésére sokszor a tanuló és teszt halmazok mellet elkülönítünk egy *validációs* halmazt is, és fejlesztés közben ezt használjuk tesztelésre, a teszt halmazt pedig csak a végső kiértékelésben. Keresztvalidáció esetén a tanuló halmazon végzünk keresztvalidációkat fejlesztés közben, és itt is egy előre elkülönített teszt halmazt használunk a legvégső kiértékelésre.

11.4. Példányalapú tanulás

Adottak $(x_1, y_1), \ldots, (x_n, y_n)$ példák. Ötlet: adott x-re az y-t az x-hez „közeli" példák alapján határozzuk meg. Hogyan?

11.4.1. Legközelebbi szomszéd modellek

1. Keressük meg x k legközelebbi szomszédját (k pl. 5 és 10 között).

2. A $h(x)$ értéke ezen szomszédok y-jainak átlaga (esetleg távolsággal súlyozva) ha y folytonos, ha diszkrét, akkor pl. többségi szavazata a szomszédoknak.

Sűrűség közelítése: Ha $x_1, \ldots, x_n$ példák adottak (y nélkül), akkor a $P(X)$ sűrűség közelítésére is jó: adott x-re nézzük meg, hogy a k legközelebbi szomszéd mekkora területen oszlik el. $P(x)$ ekkor fordítottan arányos a területtel. A k megválasztása fontos.

Távolság függvény: $D(x_1, x_2)$. Diszkrét esetben pl. Hamming távolság: a különböző jellemzők száma (pl. Hamming(001,011)=1 (bináris jellemzők)). Folytonos esetben pl. euklideszi távolság ($\|x_1 - x_2\|_2$), Manhattan távolság ($\|x_1 - x_2\|_1$), stb. Az x_i példa általában egy jellemzőket tartalmazó vektor. Folytonos jellemzőket normalizálni kell (pl. testhőmérséklet és magasság nem ugyanaz a skála). Erre egy lehetőség a *standardizálás*, amikor a jellemzőből kivonjuk az átlagot és elosztjuk a szórással. Egy másik lehetőség, hogy a jellemzőt adott intervallumba skálázzuk (pl. $[0, 1]$), de ez csak akkor lehetséges, ha a jellemzőnek van alsó és felső korlátja.

Egyszerű, de sokszor jó algoritmus. *Hibái*:

1. érzékeny a távolságfüggvény definíciójára

2. sok példa esetén költséges a k legközelebbi szomszédot megtalálni (bár vannak algoritmusok és adatszerkezetek amik segítenek, külön kutatási terület)

3. ha sok jellemző van (sokdimenziós a tér) akkor a távolság nagyon nem intuitív. Pl. vegyünk n példát egyenletes valószínűséggel a d dimenziós egységnyi oldalú hiperkockából. Mekkora kocka tartalmaz várhatóan k példát? Egy b oldalú kocka, ahol $b^d = k/n$.

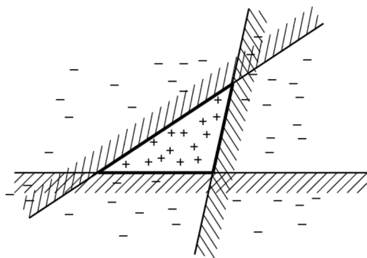
Ebből $b = (k/n)^{(1/d)}$, pl. ha $n = 10^6$, $k = 10$, $d = 100$, akkor $b \approx 0.89$!

11.4.2. Kernel módszerek

Vegyünk egy $K_\mu(x)$ eloszlást (ill. sűrűségfüggvényt, ha folytonos változók), amelynek μ a várható értéke. Minden x_i példához a kernelfüggvény a $K_{x_i}(x)$ függvény. Az adatok eloszlását (sűrűségét) becsülhetjük: $P(x) = \frac{1}{n}\sum_{i=1}^{n} K_{x_i}(x)$; ill. ha (x_i, y_i) alakúak a példák, akkor $h(x)$ a kernelfüggvénnyel súlyozott átlaggal (ill. szavazással, diszkrét esetben) közelíthető. Itt minden példának van szerepe, de távolsággal súlyozva.

11.5. Hipotézisegyüttesek

Sokszor érdemes több modellt gyártani, esetleg sokfajta modellt más algoritmusokkal, és pl. többségi szavazást alkalmazni. Miért? (1) A különböző modellek hibái nem ugyanolyanok, ideális esetben egymástól függetlenek, tehát statisztikailag megbízhatóbb eredményt adhatnak (2) egyszerű modellek kombinációja ekvivalens lehet komplexebb modellekkel (ábra).



11.5.1. Turbózás (boosting)

Gyenge modellek együttesét állítja elő, miközben az egyes modelleket és a példákat is súlyozza. Bármilyen ún. *gyenge* algoritmust „felturbóz”.

Fontos, hogy feltesszük, hogy minden (x_i, y_i) példához egy w_i súlyt

rendelünk, és ezt a tanuló algoritmusnak kezelni kell tudni. Pl. a döntési fa tudja kezelni, mert a pozitív és negatív minták számolása helyett a súlyokat összegezhetjük.

Nagyvonalakban, az L algoritmus *gyenge*, ha a tanító példák felett jobb modellt állít elő mint a találgatás (a súlyokat is figyelembe véve), picit pontosabban: nagy valószínűséggel olyan modellt kapunk, amely nagyobb mint 50% valószínűséggel osztályoz egy véletlen példát jól. A pontos formális definíciót nem vezetjük be (PAC tanulást kellene tárgyalni), a gyakorlatban „értelmes de egyszerű" algoritmusokkal érdemes kísérletezni.

```
AdaBoost(példák, L, M)
// példák: (x_i,y_i), i=1,...,N
// L: gyenge tanuló algoritmus
// M: hipotézisek száma az együttesben
// w_i: i. példa súlya, kezdetben 1/N
// h_i: i. hipotézis (i=1,...,M)
// z_i: h_i súlya
 1 for m=1 to M
 2    h_m = L(példák, w)
 3    hiba = 0
 4    for j=1 to N
 5       if h_m(x_j)!=y_j then hiba = hiba+w_j
 6    for j=1 to N
 7       if h_m(x_j)==y_j
 8          then w_j = w_j*hiba/(1-hiba)
 9    normalizáljuk w-t (összeg legyen 1)
10    z_m = log( (1-hiba)/hiba )
11 return súlyozottTöbbség(h,z)
```

Pl. L lehet *döntési tönk*, azaz olyan döntési fa, amely csak a gyökeret tartalmazza (egy változót).

A nehéz példák és a jó hipotézisek nagyobb súlyt kapnak.

Érdekes, hogy a túlillesztés nem jelentkezik ha növeljük M-et, sőt, az általánosítás egyre jobb lesz. Rengeteg elméleti eredmény ismert a témában, de ezeket nem tárgyaljuk.

11.6. Induktív tanulás optimalizációval

Messze a legnépszerűbb megközelítés.

Szokás szerint adottak az $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq X \times Y$ példák, és a $h^*\colon X \to Y$ fügvényt keressük amely a példákra jól illeszkedik, és jól is általánosít. Az optimalizációs megközelítésben a H hipotézistér felett a példákat legjobban közelítő h^* hipotézist egy optimalizálási feladat megoldásaként keressük. Ehhez definiáljuk az $\ell : X \times Y \times H \to \mathbb{R}$ *veszteségfüggvényt (loss function)*, amely egy $(x, y) \in X \times Y$ példára megadja, hogy az adott $h \in H$ hipotézis „mennyi bajt okoz” az adott példán. Egy lehetséges veszteségfüggvény pl. a négyzetes hiba: $\ell(x, y, h) = (h(x) - y)^2$.

Ha rögzítettük a veszteségfüggvényt, akkor az optimalizálási feladatunk a következő:

$$h^* = \arg\min_{h \in H} \sum_{i=1}^{n} \ell(x_i, y_i, h).$$

Fontos: az optimalizálás alapú megközelítés tervezésekor három alapvető dolgot kell megterveznünk:

1. A H hipotézistér
2. Az ℓ veszteségvüggvény
3. Az optimalizáló algoritmus

Ezek a komponensek egymástól viszonylag függetlenek, adott alkalmazásban gyakran lecserélhető bármelyik a másik kettő változatlanul hagyása mellett.

A legtöbb eszközben számos optimalizálási módszer van megvalósítva, tehát az első kettő feladatra kell leginkább koncentrálni.

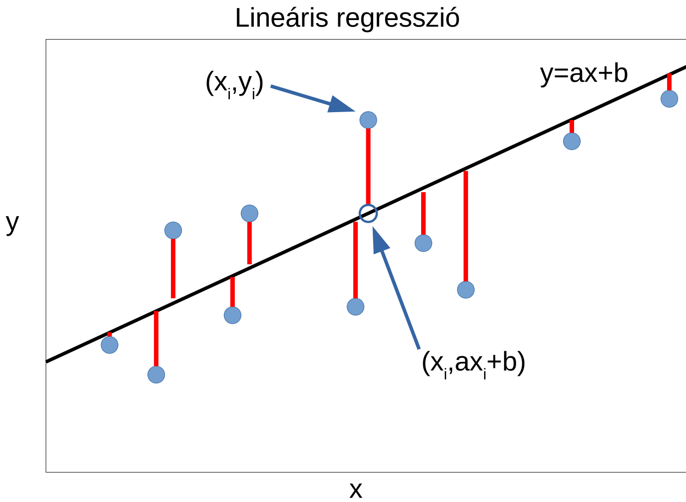
11.6.1. Lineáris regresszió

Az egyik legegyszerűbb példa a lineáris regresszió. Ebben az alkalmazásban, egy dimenziós esetben itt $X = Y = \mathbb{R}$. A három fő komponens a következő:

1. $H = \{h_{a,b}(x) = ax + b : a, b \in \mathbb{R}\}$, azaz a lineáris, eltolást (b) is tartalmazó függvények halmaza
2. $\ell(x, y, h_{a,b}) = (h_{a,b}(x) - y)^2 = (ax + b - y)^2$, azaz a négyzetes hiba
3. Az optimalizáló algoritmus legyen a *gradiens módszer (gradient descent)*

Ahogy a 12.4.2 fejezetben tárgyaljuk, a négyzetes hiba választásának elméleti motivációt lehet adni, t.i. ha az adatok úgy keletkeznek, hogy ténylegesen lineáris összefüggés van, de normális eloszlású zaj adódik az y értékekhez. Ekkor négyzetes hiba veszteséget választva várható értékben visszakapjuk a tényleges lineáris összefüggést.

A d dimenziós eset, ahol $X = \mathbb{R}^d$, teljesen hasonlóan kezelhető, ekkor $a, x \in \mathbb{R}^d$, az ax szorzat pedig vektor-belsőszorzatként értendő, de egyébként minden ugyanaz.



Gradiens módszerek általában

Érintőleg említettük korábban, hogy a gradiens módszerek családja a differenciálható függvények körében az általános hegymászó módszer alkalmazásának felel meg.

Tegyük fel hogy $f : \mathbb{R} \to \mathbb{R}$ differenciálható. Az $f'(x)$ derivált nulla, ha x lokális szélsőérték, pozitív, ha f növekszik, és negatív, ha f csökken x-nél. Tehát valamely elég kicsi $\gamma > 0$ esetében ha

$$x_{t+1} = x_t - \gamma f'(x_t),$$

akkor $f(x_{t+1}) <= f(x_t)$. Tehát ez megfelel egy hegymászó lépésnek.

Ez inspirálja a gradiens módszert, amelyet valamely tetszőleges x_0 kezdőértékből indítunk, majd kiszámoljuk az

$$x_{t+1} = x_t - \gamma_t f'(x_t)$$

képlettel az x_t sorozat elemeit, amelyre azt szeretnénk, hogy közelítse f egy lokális minimumát. Megfelelő γ_t választása esetén ez garantált,

azonban a pontos γ_t nem mindig ismert. A gyakorlatban vagy $\gamma_t = \gamma$ konstans értéket használunk, ami elég kicsi ahhoz, hogy a rendszer konvergálni tudjon, vagy idővel csökkentjük, pl. $\gamma_t \sim 1/t$, ami emlékeztet a szimulált hűtés módszerére.

A γ_t elnevezése a gépi tanulásban *tanulási ráta (learning rate)*.

Ha $f : \mathbb{R}^d \to \mathbb{R}$ egy d dimenziós differenciálható függvény, a fenti módszer változtatás nélkül alkalmazható, hiszen a $\nabla f(x) \in \mathbb{R}^d$ gradiens vektor a legnagyobb növekedés irányába mutat, így

$$x_{t+1} = x_t - \gamma_t \nabla f(x_t)$$

hasonlóan viselkedik.

Gradiens módszer a lineáris regresszióra

Visszatérve a lineáris regresszióhoz, a minimalizálandó függvény tehát

$$J(a,b) = \sum_{i=1}^{n} \ell(x_i, y_i, h_{a,b}) = \sum_{i=1}^{n} (ax_i + b - y_i)^2.$$

Erre pedig

$$\nabla J(a,b) = \begin{pmatrix} \frac{\partial J}{\partial a} \\ \frac{\partial J}{\partial b} \end{pmatrix} = \sum_{i=1}^{n} \begin{pmatrix} 2x_i(ax_i + b - y_i) \\ 2(ax_i + b - y_i) \end{pmatrix}$$

Ezt a képletet a gradiens módszerben alkalmazva megkapjuk a lineáris regresszió *frissítési szabályát (update rule)*:

$$a_{t+1} = a_t - \gamma_t \sum_{i=1}^{n} 2x_i(a_t x_i + b_t - y_i)$$

$$b_{t+1} = b_t - \gamma_t \sum_{i=1}^{n} 2(a_t x_i + b_t - y_i)$$

Sztochasztikus gradiens módszer

A minimalizálandó $J(a,b) = \sum_{i=1}^{n} \ell(x_i, y_i, h_{a,b})$ függvény összeg alakban áll elő a legtöbb esetben: minden példa hozzájárul a veszteséghez.

A gradiens módszer frissítési szabályában megtehetjük, hogy nem az egész összeget használjuk fel, hanem egyszerre csak egy (x, y) példát:

$$x_{t+1} = x_t - \gamma_t \nabla \ell(x, y, h_{a,b}).$$

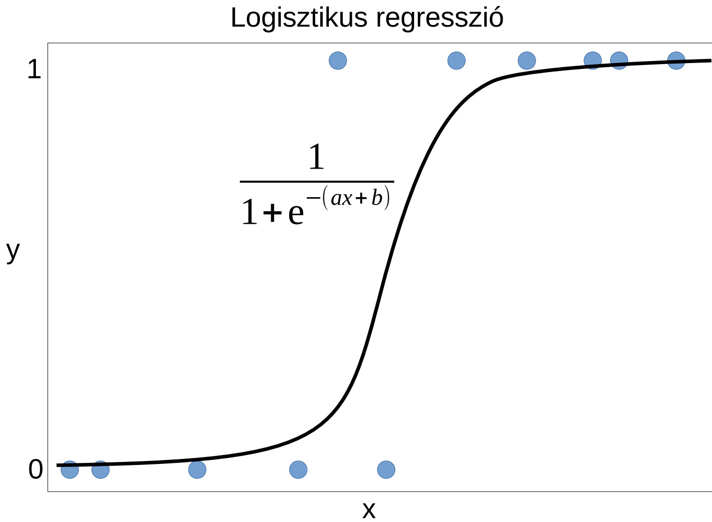
Ezt a frissítési szabályt pedig alkalmazzuk az összes ismert példára, akár véletlenszerű sorrendben, akár visszatevéses mintavétellel, akár fix sorrendben, a módszer a gyakorlatban működni fog, megfelelő γ_t használata mellett.

Különösen nagy adatbázisok esetén előnyös, gyakran gyorsabb konvergencia érhető el kevesebb számítási igénnyel.

11.6.2. Logisztikus regresszió

Itt szintén $X = \mathbb{R}$, de $Y = \{0, 1\}$. Itt a lineáris modell nem lesz jó, valami más hipotézisteret kell választani. Legyen ez a hipotézistér a logisztikus függvények halmaza, a következőképpen:

1. $H = \{h_{a,b}(x) = 1/(1 + e^{-(ax+b)}) : a, b \in \mathbb{R}\}$, azaz a logisztikus, eltolást (b) is tartalmazó függvények halmaza
2. $\ell(x, y, h_{a,b}) = -\log P(y|x, h_{a,b})$, azaz a $h_{a,b}$ hipotézis *negatív log likelihood*ja
3. Az optimalizáló algoritmus legyen a gradiens módszer



A log likelihood veszteség megértéséhez értelmezzük a logisztikus függvényt a példák valószínűségeként, feltéve az a, b paramétereket. Ehhez vezessük be a $g(x) = 1/(1 + e^{-x})$ jelölést. Tehát egy $(x, 1)$ példára $P(1|x, h_{a,b}) = g(ax + b)$, és egy $(x, 0)$ példára $P(0|x, h_{a,b}) = 1 - g(ax + b)$. Ezt a kettő képletet össze lehet vonni egyetlen képletbe: egy (x, y) példára

$$P(y|x, h_{a,b}) = g(ax + b)^y (1 - g(ax + b))^{1-y}.$$

Ezt a valószínűséget a $h_{a,b}$ hipotézis *likelihood*-jának nevezzük az (x, y) adatra nézve. A $h_{a,b}$ hipotézis likelihoodja a teljes példahalmazra nézve, feltéve hogy a példák egymástól kölcsönösen függetlenek:

$$P(y_1, \ldots, y_n|x_1, \ldots, x_n, h_{a,b}) =$$
$$\prod_{i=1}^{n} P(y_i|x_i, h_{a,b}) = \prod_{i=1}^{n} g(ax_i + b)_i^y (1 - g(ax_i + b))^{1-y_i}.$$

Ez a valószínűség minél nagyobb, annál „jobb” a modell, tehát ennek a maximalizálásaként definiálhatjuk az optimalizálási feladatot. Viszont

ez egy szorzat, nem pedig összeg, ezért nehéz gradienst számolni a gradiens módszer számára. A szokásos trükk ennek a kezelésére, hogy vesszük a logaritmusát a likelihoodnak, ez lesz a log likelihood. Mivel a logaritmus monoton függvény, a nagysági sorrendet nem változtatja meg, tehát a logaritmus maximumhelye ugyanott lesz mint az eredeti maximumhely. A negatív log likelihood pedig ennek a mínusz egyszeresese, ami már minimalizálási probléma, téhát passzol az eddigi veszteségminimalizálási keretünkbe:

$$J(a,b) = -\log P(y_1,\dots,y_n|x_1,\dots,x_n,h_{a,b}) =$$
$$-\sum_{i=1}^{n}\log P(y_i|x_i,h_{a,b}) =$$
$$-\sum_{i=1}^{n} y_i \log g(ax_i+b) + (y_i-1)\log(1-g(ax_i+b)).$$

A lineáris regresszióhoz hasonlóan itt is vesszük ennek a gradiensét, ami kiszámolás után a következőnek adódik:

$$\nabla J(a,b) = \begin{pmatrix} \frac{\partial J}{\partial a} \\ \frac{\partial J}{\partial b} \end{pmatrix} = \sum_{i=1}^{n} \begin{pmatrix} x_i(g(ax_i+b)-y_i) \\ g(ax_i+b)-y_i \end{pmatrix}$$

Ezt a képletet a gradiens módszerben alkalmazva megkapjuk a logisztikus regresszió frissítési szabályát:

$$a_{t+1} = a_t - \gamma_t \sum_{i=1}^{n} x_i(g(a_t x_i + b_t) - y_i)$$

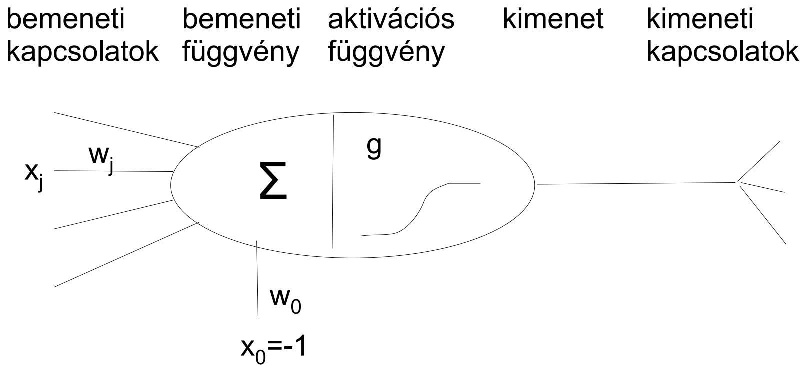
$$b_{t+1} = b_t - \gamma_t \sum_{i=1}^{n} g(a_t x_i + b_t) - y_i$$

Természetesen it is lehet $X = \mathbb{R}^d$, ekkor $a, x \in \mathbb{R}^d$, az ax szorzat pedig vektor-belsőszorzatként értendő, de egyébként minden ugyanaz.

11.7. Mesterséges neurális hálók

11.7.1. Különálló neuron

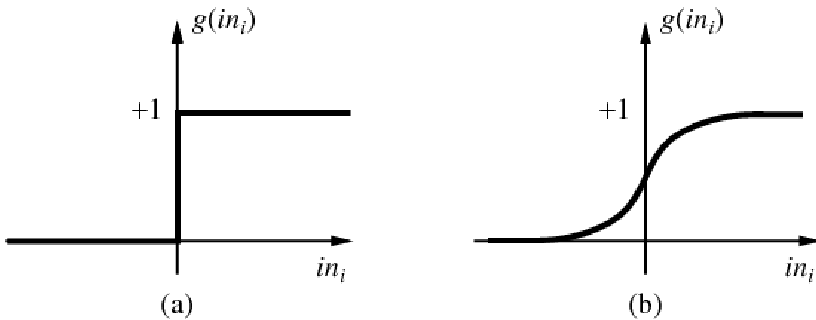
A mesterséges neuron egy modellje az ábrán látható.



Az x_j a j. bemeneti érték, a w_j a j. bemenet súlya. A w_0 az *eltolássúly (bias weight)*, x_0 pedig fix bemenet, mindig -1. A bemenetek általában tetszőleges valós számok lehetnek, bár néhány alkalmazásban lehetnek korlátosak is. A súlyok is valós számok. A bemenetekből és a súlyokból először a

$$\sum_{j=0}^{d} x_j w_j = (w_1, \ldots, w_d) \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} - w_0$$

összeget számolja ki a neuron, majd az összeg végeredményén alkalmazza az *aktivációs függvényt*.



Aktivációs függvény (g): Eredeti célja, hogy ha „jó” input jön, adjon 1-et vagy ahhoz közeli értéket (a neuron tüzel), ha „rossz” input, akkor 0-t vagy ahhoz közeli értéket. Ma már az aktivációs függvénnyel kapcsolatban nem követelmény, hogy a $[0, 1]$ intervallumból adjon értéket, de nemlineárisnak kell lennie, különben felesleges lenne, hiszen akkor a súlyokkal is kifejezhető lenne. Példák implementációkra:

- Küszöbfüggvény: $g(x) = 0$ ha $x <= 0$, $g(x) = 1$ ha $x > 0$. Ekkor a neuronunk a klasszikus *perceptron* lesz.
- Szigmoid függvény: $g(x) = 1/(1 + e^{-x})$.
- *Rectifier* aktiváció: $g(x) = \max(0, x)$. Ha a neuron ezt használja, ReLU-nak hívjuk (*rectifier linear unit*).

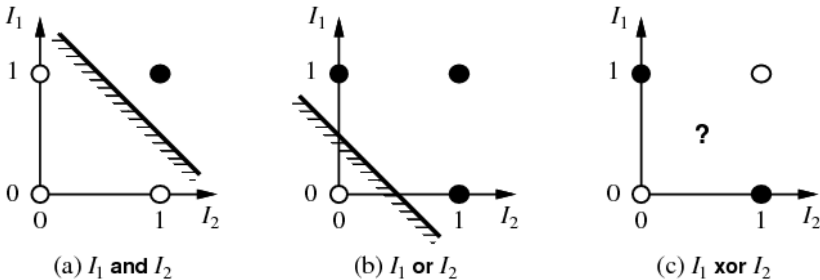
Éppen w_0-nál fognak váltani, ez az eltolássúly szerepe. Egy neuron $h_w(x)$ kimenete az x input esetén

$$h_w(x) = g\left[\sum_{j=0}^{d} x_j w_j\right] = g(w \cdot x),$$

ahol $w \cdot x$ a w és x vektorok belső szorzata és $x_0 = -1$. A $w \cdot x = 0$ egyenlet egy $d - 1$ dimenziós hipersíkot határoz meg, amely a w_0 eltolássúly miatt nem feltétlenül megy át az origón. Pl. ha $d = 2$, azaz az

input 2 dimenziós, akkor egy tetszőleges helyzetű egyenest. A neuron mint *osztályozó* tehát a teret két részre osztja egy hipersíkkal, és $w \cdot x > 0$ esetén (ami szigmoid neuron esetén azt jelenti, hogy $g(w \cdot x) > 0.5$) elfogadja az inputot, egyébként nem.

Tehát csak *lineárisan szeparálható* függvények reprezentálhatók hiba nélkül. A XOR logikai művelet például nem lineárisan szeparálható. Ehhez gondoljuk meg, hogy sok logikai műveletet reprezentálhatunk neuronnal, pl. $d = 2$, $w_1 = w_2 = 1$ és $w_0 = 0.5$ esetén a kimenet éppen $x_1 \vee x_2$ (ha az input 0 vagy 1 lehet csak). Ha itt $w_0 = 1.5$-öt veszünk, akkor $x_1 \wedge x_2$. Végül: ha $d = 1$, $w_1 = -1$ és $w_0 = -0.5$ akkor $\neg x_1$. De a XOR esetében nem létezik szeparáló sík, tehát semmilyen paraméterekkel nem lesz jó. Mit tehetünk? Kétrétegű háló (később)!



Pl. a többségi függvény lineárisan szeparálható (és ez döntési fával nehéz). Magas dimenzióban sok probléma lineárisan szeparálható lehet, pl. képek osztályozása is akár. (Ugyanakkor sok neuronnal és több réteggel bármilyen halmaz „kiszabható”, nem csak a lineárisan szeparálható esetek.)

11.7.2. Szigmoid neuron tanuló algoritmusa

Vegyük észre, hogy a szigmoid aktivációs függvényt használó neuron által definiált hipotézistér nem más, mint a lineáris regresszióban használt hipotézistér. Tehát ha a tanító adatbázisban a címkék halmaza

$Y = \{0, 1\}$, akkor használhatjuk a lineáris regressziónal látott veszteségfüggvényt is.

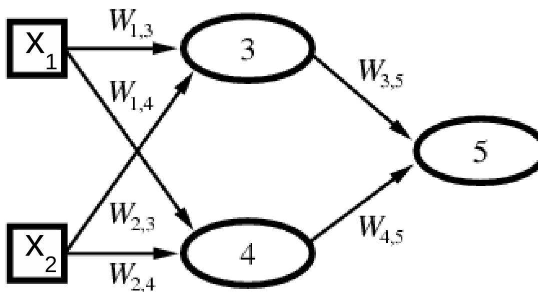
Tehát a szigmoid neuron tanítási problémája teljesen megegyezik a logisztikus regresszióval!

Leggyakrabban a szintén korábban tárgyalt stochasztikus gradiens módszert szokás alkalmazni mint optimalizáló algoritmust, mivel az adatbázisok mérete hatalmas általában, és jobb egyesével felhasználni a példákat, azzal szemben, mint ha az egész adatbázison számolt gradienssel frissítenénk a súlyvektort.

11.7.3. Többrétegű hálók

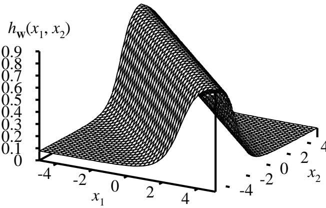
Hálózatokat szoktunk építeni neuronokból. Az ábrán egy *előrecsatolt (feedforward)* hálózat látható (most nincsenek eltolássúlyok az egyszerűség kedvéért, de egyébként kellenének mindhárom neuronnál). A bemenet x_1 és x_2. Ez az *bemeneti (input) réteg* (szögletes csúcsokkal jelölve). A kimenet az 5. neuron kimenete, ez a neuron alkotja a *kimeneti (output) réteget*. Itt egy db. *rejtett rétegünk* van amit a 3. és 4. neuron alkot. A következő függvénnyel ekvivalens a háló:

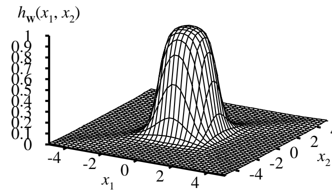
$$h_w(x_1, x_2) = g[w_{3,5}g(w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{1,4}x_1 + w_{2,4}x_2)]$$



Rekurrens hálók azok amikben van kör. Ez dinamikus rendszert definiál, ami vagy konvergál vagy nem (ilyet most nem nézünk).

Rétegek: az előrecsatolt hálózatokat általában rétegesen definiáljuk: input, output és rejtett rétegek, ahol kapcsolatok egy rétegen belül nincsenek, és a kapcsolatok mindig előre mutatnak, azaz az input irányából haladnak az output irányába. Legtöbbször szomszédos rétegek között van csak kapcsolat. Egy rejtett réteggel már bármilyen folytonos függvény közelíthető. Az ábrán kettő ill. négy neuronos rejtett réteget használtunk, és egy output neuront.





11.7.4. Többrétegű hálók tanuló algoritmusa

Tegyük fel, hogy a többrétegű hálónkat osztályozási feladatra használjuk, és két osztályunk van, tehát a példahalmazunk a logisztikus regressziónál látott alakot ölti: $X = \mathbb{R}^d$ és $Y = \{0, 1\}$.

A többrétegű háló struktúráját rögzítjük, és feltesszük, hogy egy kimeneti neuron van, amelynek küszöbfüggvénye $[0, 1]$-ből vett értéket ad, ez lesz a kimenet. A hipotézisteret a w súlyvektor (amely az összes kapcsolat súlyait tartalmazza) lehetséges értékei alkotják.

Itt is az optimalizáció alapú megközelítést alkalmazzuk. Ennek a három komponense lehet pl.:

1. $H = \{h_w : w \in \mathbb{R}^m\}$, ahol $h_w : \mathbb{R}^d \to [0, 1]$ egy rögzített struktúrájú, w súlyokkal rendelkező többrétegű neuronháló

2. $\ell(x, y, h_w) = -\log P(y|x, h_w)$, azaz a h_w hipotézis *negatív log likelihood*ja

3. Az optimalizáló algoritmus legyen a gradiens módszer

A nehézség természetesen a gradiens meghatározása egy sok neuronból álló komplex hálózat esetén. A gyakorlatban a népszerű keretrendszerek ezt a terhet leveszik a programozó válláról az *automatikus differenciálás* segítségével, ahol is csak a struktúrát kell megadni, és a többit a rendszer „intézi" a számunkra.

A sokat emlegetett *visszaterjesztés (backpropagation)* algoritmus is valójában a gradiens kiszámolására szolgál. Azért hívják visszaterjesztés algoritmusnak, mert a moduláris szerkezetű neuronhálók esetében a gradienst is „hátulról visszafelé" modulárisan lehet számolni adott struktúra esetében. Ezt fel lehet úgy is fogni, mintha a kimeneti rétegen jelentkező veszteség érték visszafele terjedne a rejtett neuronokra bizonyos szabályok szerint. Ennek levezetését nem tárgyaljuk.

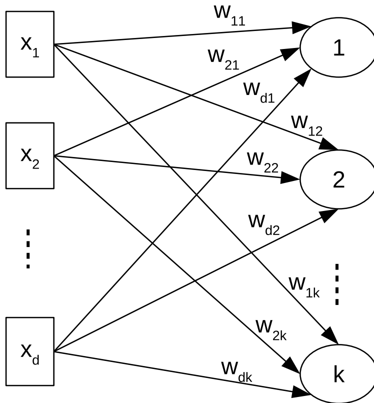
Ha nem csak egy, hanem k osztályt kell tanulni, akkor lehet k db. kétosztályos hálót tanítani, azaz minden osztályra külön hálót, ahol a negatív példák az összes többi osztály példái, és egy ismeretlen példa osztályozása során ekkor a maximális kimenetű háló lehet a keresett osztály. Lehet egyetlen hálót is használni, de k kimeneti neuronnal, ilyenkor ehhez illeszkedő veszteségfüggvényt kell használni, pl. *kereszt entrópia veszteség (cross entropy loss)*.

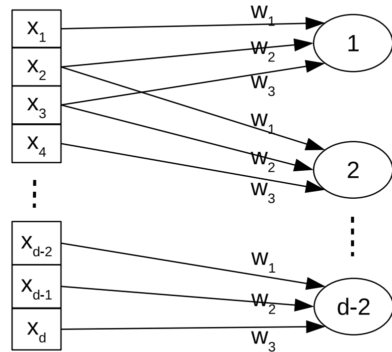
11.8. Modern neuronháló-architektúrák

A jelenleg használatos neuronhálók jellemzően nagy méretűek és számos trükköt használnak, amelyek a teljesítményt javítják. Maradunk a leggyakoribb előrecsatolt hálóknál. Ezek rétegekből épülnek fel. Néhány rétegtípust tárgyalunk, amelyekből komplett architektúrák építhetők fel modulárisan.

11.8.1. Teljesen összefüggő réteg

Az ábra (bal oldal) egy k neuronból álló *teljesen összefüggő (fully connected)* réteget szemléltet, amely egy d dimenziós inputot k dimenziós outputra képez. Az összes input össze van kötve az összes neuronnal. Összesen tehát $d \cdot k$ kapcsolat van, mind külön súllyal. Az eltolási súlyokat is beleszámolva tehát $d \cdot k + k$ súly szükséges.





11.8.2. Konvolúciós réteg

Gondloljuk meg, hogy a teljesen összefüggő réteg az input minden permutációjára ekvivalens tanulási feladatot jelent. Ha az input pl. kép, ez egyértelműen túl általános reprezentáció, mert egy képen fontos a pixelek szomszédsági relációja, távolsága.

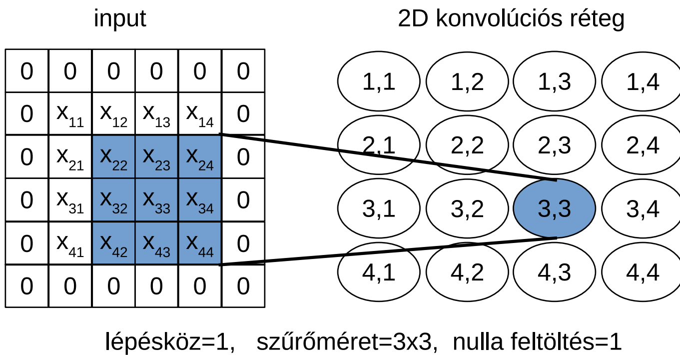
A konvolúciós réteg olyan inputokra lett kitalálva, ahol a szomszédsági reláció fontos (pl. kép, hang, stb). Az ábra jobb oldala egy konvolúciós réteget szemléltet, amely 1D struktúrájú inputra (azaz vektorra) működik.

A teljesen összefüggő réteghez képest két különbség van: (1) a réteg

teljesen azonos neuronokból áll, amelyek súlyai megegyeznek, és (2) minden neuron csak egy rögzített összefüggő sávot lát az inputból (az ábrán ennek szélessége 3).

Az ábrán szemléltetett réteg tehát leírható mindössze 3 db. súllyal, k-tól és d-től függetlenül. Ez drasztikusan kevesebb mint a teljesen összefüggő réteg súlyainak száma.

Fel lehet úgy is fogni, hogy egy 3 szélességű *szűrővel* végigpásztázzuk az inputot. A *lépésköz (stride)* (S) paraméter és a *szűrőméret* (F) határozza meg az output dimenzióját, ami $\lceil (d - F + 1)/S \rceil$. Ha $F > 1$, ez mindig kevesebb mint az input dimenzió. Ha nem akarjuk, hogy csökkenjen a dimenzió, akkor alkalmazhatunk *nulla feltöltést (zero padding)*, azaz az input széleihez a szükséges számú nulla értéket csatolunk.

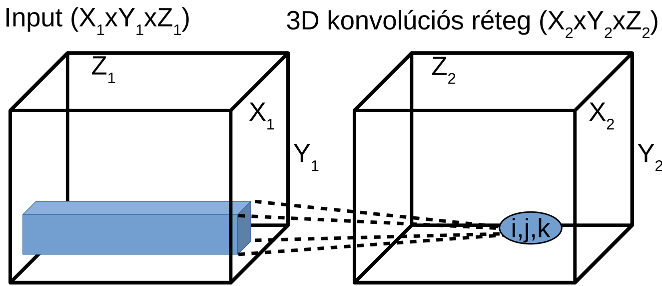


Ha az input szomszédsági relációja 2D (azaz mátrix, pl. képek), akkor 2D konvolúcós réteget is lehet használni. Ilyenkor a neuronok is 2D elrendezésben vannak, mert így további 2D konvolúciós rétegeket lehet egymás után alkalmazni. A példában 3x3-as szűrő szerepel, ez 9 súllyal leírható.

A mátrix inputnak sokszor mélysége is van, pl. színes RGB képeknél a mélység 3. Ekkor az input egy 3D *tenzor*, és a szűrőnek is van egy harmadik dimenziója amely a teljes mélységet lefedi. Tehát pl. 3x3x3-as

szűrőt alkalmazhatunk, ez 27 súllyal rendelkezik. Érdekes következmény, hogy itt akár 1x1-es szűrőnek is értelme lehet, a mélység miatt ez valójában 1x1x3-as szűrő lesz, ami a rétegeket kombinálja össze.

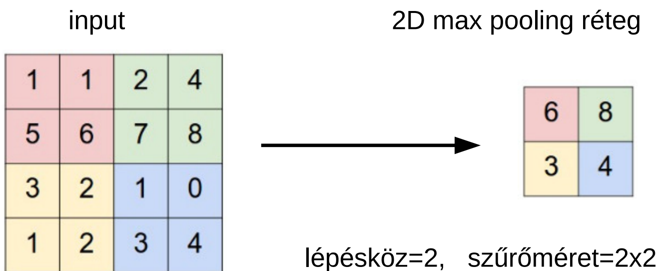
A gyakorlatban egy konvolúciós rétegben több azonos méretű szűrő is lehet, amelyek együtt egy 3D tenzor kimenetet képeznek, ha az egyes szűrők 2D mátrix kimeneteit „egymásra fektetjük”. Tehát a konvolúciós réteg jellemzően 3D tenzor inputból 3D tenzor outputot készít.



11.8.3. Pooling réteg

Az input réteg méretének csökkentésére szolgál. Méretet lehet csökkenteni a convolúciós lépésköz paraméter növelésével is, tehát pooling réteg nem mindig van.

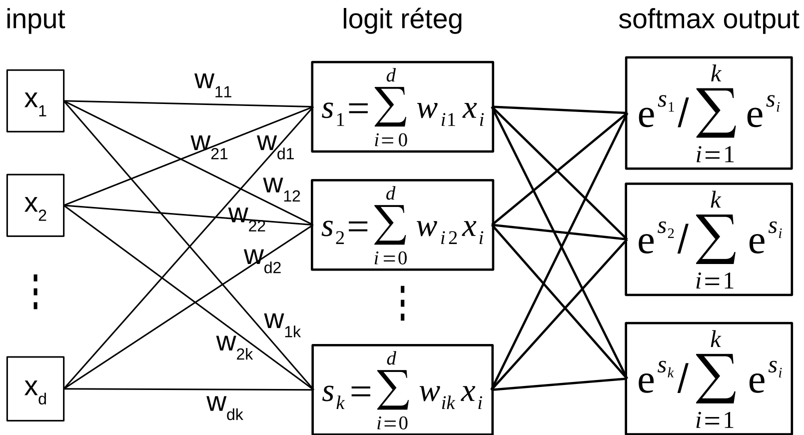
A leggyakoribb a max pooling módszer. Pl. egy ilyen (2x2)-es max szűrőt alkalmazhatunk 2-es lépésközzel.



3D inputnál minden mélységi rétegre függetlenül alkalmazzuk a 2D max pooling szűrőt, az output mélysége tehát változatlan lesz.

11.8.4. Softmax output réteg

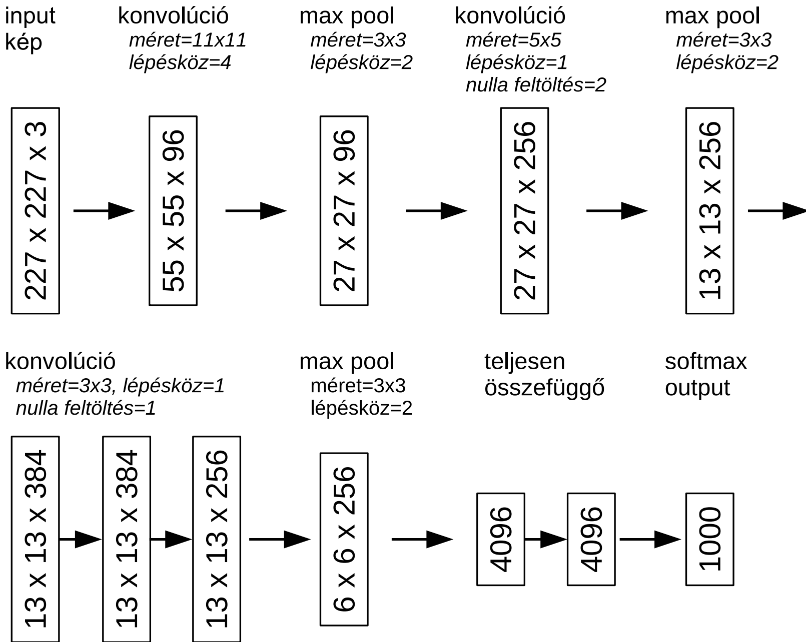
Felfogható a szigmoid neuron általánosításának, ahol nem 1 kimenet van hanem k. A kimenet egyre összegződik és 0 és 1 közötti értékeket tartalmaz, ezért sokszor valószínűségi eloszlásnak értelmezzük. Akkor kell alkalmazni, ha a lehetséges címkék *egymást kölcsönösen kizárják* (pl. kutya, macska, ló, stb.).



A softmax output réteget megelőző, szummákat tartalmazó réteget szokás külön rétegként kezelni, amelynek neve *logit* réteg. Ez félrevezető szóhasználat, de elterjedt. Azt jelenti, hogy aktivációs függvény nélküli értékek vannak benne, és szinte mindig egy szoftmax output réteg követi.

11.8.5. Példa architektúra: AlexNet

Az AlexNet architektúra az egyik első mélyháló[1]. A neuronok rectifier aktivációt használnak. A kimenet softmax $k = 1000$ osztállyal.



Az input egy 227x227-es színes bittérkép, tehát mélysége 3 (RGB). Az első konvolúciós réteg 96 szűrőt tartalmaz, mindegyik 11x11 méretű, 4-es lépésközzel. Mivel $\lceil (227 - 11 + 1)/4 \rceil = 55$, a réteg outputja 55x55x96 lesz. A többi réteg hasonlóan.

Több mint 60 millió súly, túlnyomó része a teljesen összefüggő rétegben!

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012.

11.8.6. Záró megjegyzések

A mélytanulásnak csak a felszínét súroltuk. Számtalan architektúra trükk, veszteségfüggvény trükk és optimalizáló algoritmus trükk ismert, és ezek száma dinamikusan növekszik. Illetve felügyelet nélküli tanulásban is jelentős eredmények ismertek szintén mély architektúrák segítségével.

12. fejezet

Statisztikai tanulás

Az alaphelyzet ugyanaz mint korábban: tanulási probléma, példák, hipotézisek. Már eddig is érintettük a témát, pl. a logisztikus regressziónál. Felügyelet nélküli tanulást nézünk, de alkalmazható felügyelt tanulásra is.

Szemléltető példa: meggy és citrom ízű cukor (1) ugyanolyan csomagolásban de (2) ötféle zacskóban, amelyekben különböző arányban vannak a cukrok. Tudjuk, hogy a lehetséges arányok a következők:

h_1	100% meggy	
h_2	75% meggy	25% citrom
h_3	50% meggy	50% citrom
h_4	25% meggy	75% citrom
h_5		100% citrom

Ezek lesznek a hipotézisek. A példák egy adott zacskóból vett $D_1, \ldots, D_N$ minták (D_i értéke „meggy” vagy „citrom”), és a felügyelet nélküli tanulási feladat itt az, hogy döntsük el melyik hipotézis igaz.

12.1. Bayes tanulás

A következő fogalmak fontosak:

Prior: $P(h_i)$, azaz a hipotézisek a priori (tapasztalat előtti, azaz példákat nem tekintő) valószínűsége. (A fenti példában eleve csak ötféle hipotézis valószínűsége különbözik nullától, ami nagyon erős a priori tudás.)

Posterior: $P(h_i|d)$, (d a megfigyelt adatok) azaz a hipotézisek valószínűsége, feltéve, hogy a d tényeket ismerjük.

Likelihood: $P(d|h_i)$, azaz a megfigyelt adatok valószínűsége, feltéve, hogy a h_i hipotézis áll fenn.

A prior valószínűségek fejezik ki a világról alkotott háttértudásunk egy jelentős szeletét.

Az általános Bayes tanulás célja a posterior valószínűségek megadása:

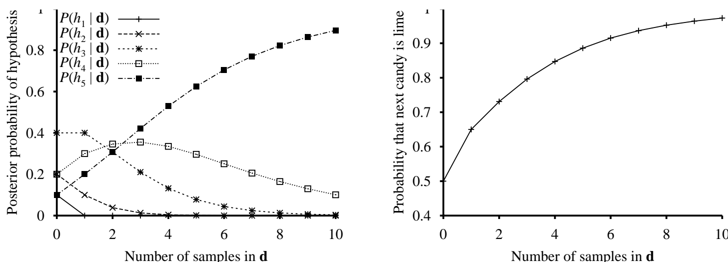
$$P(h_i|d) = \alpha P(d|h_i)P(h_i), \quad \alpha = 1/P(d)$$

A posterior valószínűségek ismeretében egy ismeretlen X változó posterior eloszlását is megadhatjuk:

$$\begin{aligned} P(X|d) &= \sum_i P(X, h_i|d) = \sum_i P(X|h_i, d)P(h_i|d) = \\ &= \sum_i P(X|h_i)P(h_i|d) \end{aligned}$$

feltéve, hogy $P(X|h_i)$ ismert és X független d-től feltéve h_i.

Pl. legyen a $h_1, \ldots, h_5$ hipotézisek prior eloszlása $(0.1, 0.2, 0.4, 0.2, 0.1)$. Esetünkben feltehetjük, hogy a $D_1, \ldots, D_N$ adatok független és azonos eloszlású véletlen változók. Legyenek az értékeik (a konkrét megfigyelések) $d_1, \ldots, d_N$. Ekkor a likelihood $P(d|h_i) = \prod_j P(d_j|h_i)$, pl. ha h_5 áll fenn (csak citrom), akkor a minták mind citromok lesznek. Ekkor pl. $P(d|h_3) = (1/2)^N$. Az



$\alpha = 1/P(d)$ normalizálási konstanst a

$$P(d) = \sum_i P(d|h_i)P(h_i)$$

összefüggésből kiszámolva minden hipotézis $P(h_i|d)$ posterior valószínűségét kiszámolhatjuk az egyszerű példánkban (álalában viszont ez nehéz lehet).

Minél nagyobb a minták száma, annál biztosabb a Bayes döntés, mert a helyes hipotézis posterior valószínűsége az egyhez tart, a többié pedig nullához. (Minták teljes hiányában ($N = 0$) pedig a prior alapján döntünk.)

A Bayes tanulás (ami a posterior valószínűségek meghatározásán, ill. az ezekkel való súlyozáson alapul, ahogy a fenti $P(X|d)$ predikció szemlélteti) *optimális*, azaz a rendelkezésre álló tudás alapján nem lehet valószínűségi értelemben jobban dönteni, viszont általában túl költséges: egyszerűsíteni kell. Erre jó a MAP és a maximum likelihood tanulás.

12.2. Maximum a posteriori (MAP) tanulás

A priorokkal súlyozott összeg helyett a *maximális* $P(h_i|d)$-hez tartozó h_i-t egyedül használjuk, ez a *MAP hipotézis*. Ez „veszélyesebb”, de gyakran olcsóbb. Sok adatból u.oda konvergál mint a Bayes.

A következőt vehetjük észre: a MAP hipotézis éppen

$$h_{MAP} = \arg\max_{h_j} P(h_j|d) = \arg\max_{h_j} P(d|h_j)P(h_j) =$$
$$= \arg\min_{h_j}(-\log P(d|h_j) - \log P(h_j))$$

Az utolsó alak információelméleti értelmezése éppen az adatok információtartalma h_i mellett és a h_i információtartalma: ennyi bit kell a hipotézis és az adatok együttes leírásához. A MAP hipotézis más szóval tehát a *maximális tömörítést* biztosítja.

Ezen kívül, speciális esetben, ha a $P(d|h_i)$ ugyanaz minden i-re, akkor a fenti képlet éppen az Occam borotvája elvét fogalmazza meg: vegyük a legegyszerűbb hipotézist.

12.3. Maximum likelihood (ML) tanulás

Ha feltesszük, hogy minden i-re a prior valószínűségek megegyeznek, akkor a

$$h_{ML} = \arg\max_{h_j} P(d|h_j)$$

maximum likelihood (ML) hipotézis éppen ugyanaz lesz mint a MAP hipotézis.

Az ML hipotézist egyszerűbb kiszámolni viszont pontatlanabb, mert a priorokat nem veszi figyelembe. Sok (végtelen) adat esetén ugyanazt adja mint a Bayes és a MAP, de kevés adaton problémás.

12.4. Tanulás teljesen specifikált példákból

Tegyük fel, hogy a tanuló példákban minden változó értéke adott.

12.4.1. Maximum likelihood diszkrét változókkal

1. példa: meggy/citrom példa, de most legyen a meggy/citrom arány tetszőleges (θ) és ismeretlen h_θ azt a hipotézist jelöli, amelyben az arány θ.

Legyen $N = m + c$ független példánk (m: meggyesek, c: citromosak). Ekkor h_θ mellett az adatok likelihoodja

$$P(d|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^m (1-\theta)^c$$

Ez melyik θ-ra maximális? Szokásos trükk: *log likelihood* maximuma ugyanott van de könnyebb kiszámolni, mert összeget csinál a szorzatból, amit tagonként lehet maximalizálni:

$$L(d|h_\theta) = \log P(d|h_\theta) = \sum_{j=1}^{N} \log P(d_j|h_\theta) = m \log \theta + c \log(1-\theta)$$

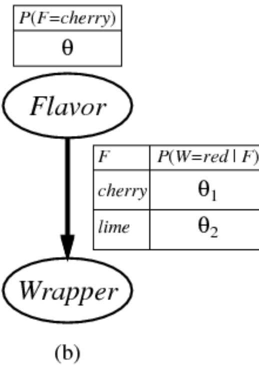
Itt vehetjük a log likelihood deriváltját, maximum csak ott lehet ahol ez nulla:

$$\frac{dL(d|h_\theta)}{d\theta} = \frac{m}{\theta} - \frac{c}{1-\theta} = 0 \quad \longrightarrow \quad \theta = \frac{m}{N}$$

Ez elég intuitív eredmény. A maximum megtalálása általában függvényoptimalizálásra vezet (pl. hegymászó, stb.), ritkán adható meg képlettel mint ez a példa.

Mivel priorokat nem használ, kevés adatból nem jó, pl. ha $N = 1$, azt mondja, hogy minden cukor meggy vagy citrom.

2. példa: Most legyen csomagolás is: piros és zöld, az ábrán látható Bayes háló szerinti eloszlásban. Három paraméterünk lesz: $\theta, \theta_1, \theta_2$. Pl.



(b)

$$P(\text{meggy,zöld}|h_{\theta,\theta_1,\theta_2}) =$$

$$P(\text{zöld}|\text{meggy,}h_{\theta,\theta_1,\theta_2})P(\text{meggy}|h_{\theta,\theta_1,\theta_2}) =$$

$$\theta(1-\theta_1)$$

Megint legyen $N = m + c$ független példa, ahol a csomagolások ezen belül $m = p_m + z_m$ és $c = p_c + z_c$. Ekkor

$$P(d|h_{\theta,\theta_1,\theta_2}) = \theta^m(1-\theta)^c\theta_1^{p_m}(1-\theta_1)^{z_m}\theta_2^{p_c}(1-\theta_2)^{z_c}$$

Ismét a log likelihood módszerével tagonként maximalizálva kaphatjuk, hogy $\theta = m/N$, $\theta_1 = p_m/m$, $\theta_2 = p_c/c$.

Belátható, hogy diszkrét Bayes hálókra mindig hasonló eredmény adódik, a táblázatokat becsüljük a gyakoriságokból egymástól függetlenül.

Alkalmazás: naiv Bayes. Korábban már láttuk az algoritmust. A fentiek szerint a naiv Bayes módszer a maximum likelihood módszer alkalmazása, ahol a Bayes háló $O(N)$ paraméterét kizárólag a minták gyakorisága alapján töltjük ki. Ez egy olcsó és sokszor jó módszer, turbózással (boosting) pedig még jobb.

12.4.2. Maximum likelihood folytonos változókkal

1. példa: Legyenek $x_1, \ldots, x_N$ N db független minta a μ, σ paraméterű normális eloszlásból:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

ahol μ és σ ismeretlen.

A log likelihood ekkor:

$$L(d|h_{\mu,\sigma}) = N \cdot (-\log\sigma - \log\sqrt{2\pi}) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}$$

Ebben a deriváltakat μ és σ szerint nullának véve azt kapjuk ML becslésre, hogy

$$\mu = \frac{1}{N}\sum_{j=1}^{N} x_j, \quad \sigma = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x_j-\mu)^2}$$

ami szintén elég intuitív.

2. példa: Legyenek most X és Y két változó, valamint tegyük fel, hogy

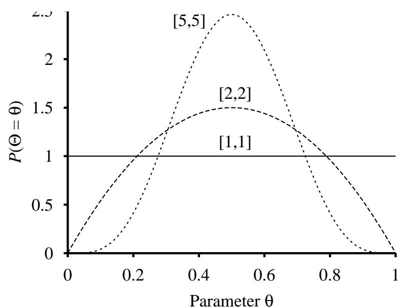
$$P(y|x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(y-(\theta_1 x-\theta_2))^2}{2\sigma^2}}$$

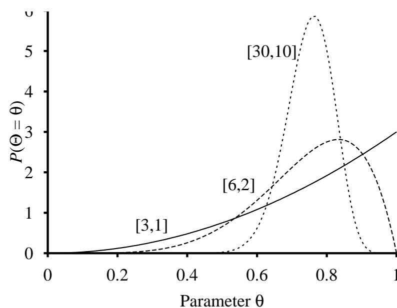
és θ_1, θ_2 ismeretlenek.

Az adathalmazunk alakja tehát $(x_1, y_1), \ldots, (x_N, y_N)$. Ismét a log likelihood technikával a probléma a $\sum_{j=1}^{N}(y_j - (\theta_1 x_j - \theta_2))^2$ minimalizálására vezet, ami éppen a lineáris regresszió a *legkisebb négyzetek összegének* a módszere. A legkisebb négyzetek összegének a módszerével kapott egyenes tehát éppen a maximális likelihood hipotézis, ha az adathalmazunk olyan, hogy az y az x-nek és független, normális eloszlású zajnak az összege.

12.4.3. Bayes tanulás

1. példa: Cukorkás példa (ismeretlen paraméter: θ, N példa), de most a priort ($P(\theta) = (P(\Theta = \theta))$ is figyelembe vesszük (a $\Theta \in [0, 1]$ a hipotézis véletlen változója). Pl. Θ-nak lehet egyenletes eloszlása, ekkor pl. a MAP hipotézis éppen a maximum likelihood hipotézis.





A gyakorlatban sokszor vesszük a *béta (β) eloszlást* priornak, a sűrűségfüggvénye:

$$\text{beta}[a, b](\theta) = \alpha\theta^{a-1}(1-\theta)^{b-1}$$

ahol a és b paraméterek, α normalizálási konstans. Néhány tulajdonság: ha $a = b = 1$, akkor az egyenletes eloszlás; ha $a = b$, akkor szimmetrikus; ha $a + b$ nagy, akkor csúcsos; végül: $E(\Theta) = a/(a + b)$.

A Bayes tanulásnál a $P(\Theta|d)$ posteriori eloszlást keressük. Legyen a prior beta$[a, b](\theta)$, ekkor egy minta után

$$P(\theta|D_1 = \text{meggy}) = \alpha P(\text{meggy}|\theta)P(\theta) = \alpha\theta\text{béta}[a, b](\theta) =$$
$$= \alpha'\theta^a(1-\theta)^{b-1} = \text{béta}[a + 1, b](\theta)$$

Ez a *konjugált prior* tulajdonság, azaz a prior eloszlás ugyanaz mint a poszterior eloszlás, csak más paraméterekkel.

Itt konkrétan a prior interpretációja *virtuális számláló*, azaz mintha már ismernék a pozitív és b negatív példát. Továbbá sok adat (nagy N) esetén a priortól függetlenül konvergál a $\theta = a/(a + b)$ becsléshez, mint ahogy a MAP és a maximális likelihood becslések is.

2. példa: cukorka+csomagolás példa (ismeretlen paraméterek: θ, θ_1, θ_2). Most a prior alakja $P(\Theta, \Theta_1, \Theta_2)$. Itt hasznos a változók függetlenségét feltenni: $P(\Theta, \Theta_1, \Theta_2) = P(\Theta)P(\Theta_1)P(\Theta_2)$. Pl. lehet mindegyik béta eloszlású.

12.4.4. Bayes hálók struktúrájának tanulása

Láttuk, hogy egy rossz és egy jó Bayes háló között nagy különbség lehet tömörségben. Hogyan találhatunk jó Bayes hálót adott adatokra? Kézzel (szakértőkkel), vagy *automatikusan*: optimalizálási problémát definiálhatunk.

1. A keresési tér a Bayes hálók halmaza (pontok adottak, éleket keresünk)
2. Az operátorok pl. hozzáadunk, elveszünk, megfordítunk éleket, stb.
3. A célfüggvényt úgy konstruáljuk, hogy jutalmazza a függetlenségi feltételek teljesülését és a predikciós képességet, de büntesse a komplexitást, stb.

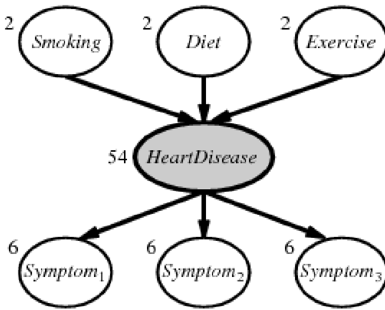
Az algoritmus lehet hegymászó, evolúciós, stb.

12.5. Rejtett változók

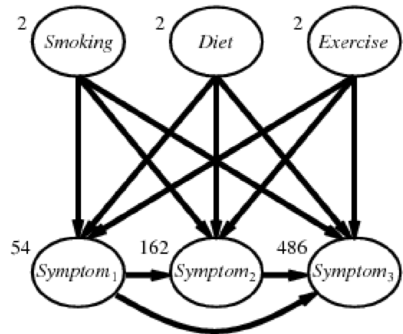
Eddig feltettük, hogy a megfigyelt példákban minden jellemző értéke ismert volt. Lehetnek azonban (1) hiányos adatok (elvben megfigyelhető, de mégis hiányzó információ), ill. (2) *rejtett változók*, amik nem figyelhetők meg közvetlenül, de a tudásreprezentációt hatékonnyá teszik.

A rejtett változókra példa a betegségek: ismerhetjük a tüneteket, adott orvosok diagnózisát, a reagálást a kezelésekre, de magát a betegséget nem figyelhetjük meg.

Miért kellenek ha nem megfigyelhetők? Mert egyszerűbbé teszik a modellt, tömörebb tudásreprezentációnk lesz.



(a)

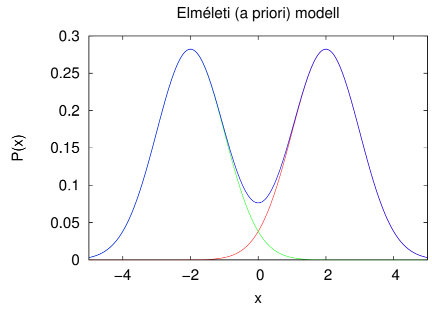


(b)

12.5.1. EM algoritmus: egy példa

Tegyük fel, hogy a példákban vannak rejtett (ismeretlen értékű) jellemzők: szeretnénk maximum likelihood modellt építeni, amely az ismeretlen jellemzőket is figyelembe veszi (később világosabb lesz, hogy ez mit jelent).

Először egy példa: legyen az elméleti modell két ismert szórású (pl. $\sigma_1 = \sigma_2 = 1$) de különböző várható értékű (pl. $\mu_1 = -2$, $\mu_2 = 2$) normális eloszlás keveréke: azaz feltesszük, hogy egy példa úgy keletkezik, hogy először kiválasztjuk, hogy melyik normális eloszlást használjuk a kettő közül, majd ebből veszünk véletlen mintát. Ekkor



$$P(X) = \frac{1}{2}\mathcal{N}(\mu_1, \sigma_1) + \frac{1}{2}\mathcal{N}(\mu_2, \sigma_2)$$

Minden x_i példához tartozik két rejtett jellemző: z_{i1} és z_{i2}, ahol $z_{ij} = 1$ ha x_i a j-edik normális eloszlásból vett minta, egyébként $z_{ij} = 0$ ($i = 1, \ldots, N$, $j = 1, 2$). A teljes minta tehát (x_i, z_{i1}, z_{i2}), amelyből csak x_i

ismert.

Maximum likelihood becslést akarunk μ_1-re és μ_2-re:

$$h_{ML} = (\mu_1^*, \mu_2^*) = \arg \max_{(\mu_1,\mu_2)} P(x|\mu_1, \mu_2) = \arg \max_{(\mu_1,\mu_2)} \sum_{z \in Z} P(x, z|\mu_1, \mu_2),$$

ahol z az adatok z_{ij} értékeiből álló mátrix, és Z tartalmazza a z_{ij} változók értékeinek az összes lehetséges kombinációját. Ez komplex kifejezés a szumma miatt, a tagok száma exponenciális az adatpontok függvényében, tehát ha sok adatpont van, akkor nem kezelhető. Tehát hatékonyabb módszert keresünk a maximum likelihood meghatározására (ill. közelítésére).

A trükk, hogy iteratív módszert alkalmazunk: adott μ_1, μ_2 értékekhez kiszámoljuk a Z *várható értékét*, az így kapott teljes mintákon pedig új μ_1, μ_2 párt határozzunk meg maximum likelihood becsléssel. Pontosabban, egy $\mu_1^{(0)}, \mu_2^{(0)}$ hasraütéses becslésből kiindulva a teljes

$$(x_i, E(Z_{i1}|x_i, \mu_1^{(0)}), E(Z_{i2}|x_i, \mu_2^{(0)}))$$

példahalmaz segítségével meghatározzuk a paraméterek maximum likelihood becslését, ami legyen $\mu_1^{(1)}, \mu_2^{(1)}$. Ezután az egészet ismételjük iteratívan, amíg a becslés nem konvergál. A lényeg, hogy közben a maximum likelihood értéke folyamatosan nő (amíg egy lokális maximumba el nem ér).

A képletek a konkrét példára a k. iterációban ($i = 1, \ldots, N$, $j = 1, 2$):

$$E(Z_{ij}|x_i, \mu_j^{(k)}) = P(Z_{ij} = 1|x_i, \mu_j^{(k)}) =$$
$$= \alpha P(x_i|Z_{ij} = 1, \mu_j^{(k)}) P(Z_{ij} = 1|\mu_j^{(k)}) = \alpha \mathcal{N}(\mu_j^{(k)}, 1)/2$$

$$\mu_j^{(k+1)} = \frac{1}{\sum_{i=1}^N E(Z_{ij}|x_i, \mu_j^{(k)})} \sum_{i=1}^N x_i E(Z_{ij}|x_i, \mu_j^{(k)})$$

Vegyük észre, hogy $\mu_j^{(k+1)}$ nem más mint a várható értékekkel súlyozott átlag, maguk a várható értékek pedig egyfajta halmazhoz tartozási súlyként értelmezhetők.

12.5.2. EM algoritmus: általános alak

Az EM jelentése: *expectation maximization*, azaz várható érték maximalizálás.

Tegyük fel, hogy rendelkezésre áll egy ismert x példaatadbázis, amelynek Z változói ismeretlenek. Itt Z véletlen változó, x konkrét adat. Jelölje Θ az illeszteni kívánt valószínűségi modell paramétereit.

A teljes adatok ($x \cup Z$) felett a $P(x, Z|\Theta)$ likelihood egy véletlen változó, hiszen Z-től függ, ami egy véletlen változó. A likelihood Θ-tól is függ. A likelihood *várható értékét* maximalizáljuk Θ változóban, feltéve, hogy a k. iterációban a modell $\theta^{(k)}$. Szokás szerint a likelihood logaritmusát vesszük, azaz a log likelihoodot: $L(x, Z|\Theta)$, amivel könnyebb számolni, de a maximuma ugyanott van.

A képlet:

$$\begin{aligned}\theta^{(k+1)} &= \arg\max_{\Theta} E(P(x, Z|\Theta)|x, \theta^{(k)}) = \\ &= \arg\max_{\Theta} \sum_{Z=z} P(z|x, \theta^{(k)}) L(x, z|\Theta)\end{aligned}$$

13. fejezet

Markov döntési folyamatok és megerősítéses tanulás

Röviden: az ágensnek kizárólag bizonyos állapotokban tapasztalt jutalmak (megerősítés) ill. büntetések alapján kell kitalálni, mikor mit kell tenni, hogy a jutalom maximális legyen.

Nagyon általános keret, sok alkalmazás, pl. robotok tájékozódása, vagy mozgás tanulása (pl. járni tanulás), játékok (go, dáma, sakk, stb.).

13.1. Markov döntési folyamatok (Markov decision processes)

Tekintsünk egy *diszkrét*, *statikus*, *sztochasztikus* és *teljesen megfigyelhető* feladatkörnyezetet. Részletesebben: tegyük fel, hogy a világ tökélesen modellezhető a következőkkel:

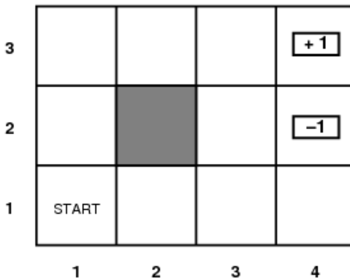
- lehetséges állapotok halmaza
- egy opcionális s_0 kezdőállapot (nem feltétlenül kell)

- minden s állapotban adott lehetséges cselekvések egy $A(s)$ halmaza, és egy állapotátmenet-valószínűség eloszlás, amely minden $a \in A(s)$ cselekvéshez, és minden s' állapothoz megadja a $P(s'|s,a)$ állapotátmenet-valószínűséget
- állapot jutalomfüggvény, amely minden s állapothoz egy $R(s)$ valós jutalomértéket rendel (lehet negatív is)
- opcionális célállapotok halmaza (nem feltétlenül kell)

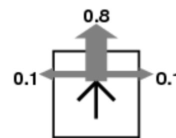
A fenti feladatkörnyezet sztochasztikus mert a cselekvések kimenetele sztochasztikus. *Markov tulajdonságúnak* azért nevezzük, mert csak az aktuális állapottól függ a cselekvés hatása, a korábban meglátogatott állapotoktól nem.

Feltesszük, hogy a fenti feladatkörnyezetet az ágens tökéletesen ismeri. Kell szenzor is az ágensnek, mert nem tudni hová kerül: feltesszük, hogy tökéletesen azonosítja az állapotot a szenzor.

Példa: állapotok egy négyzetrács cellái; cselekvések: fel, le, jobbra, balra; az $R(s)$ jutalom mindenhol konstans, kivéve két végállapot. A (2,2) cella fal.



(a)

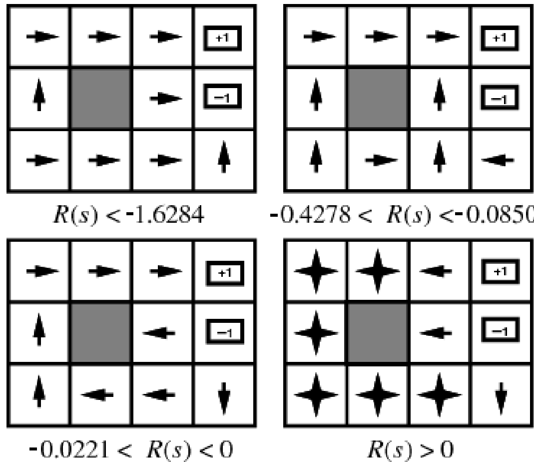


(b)

A választott cselekvés 0.8 valószínűséggel végrehajtódik, 0.1 valószínűséggel valamelyik merőleges irányba térünk ki. A fal irányába nem hajtódik végre a cselekvés (ágens helyben marad).

Az ágens minél több jutalmat akar gyűjteni, pl. a jutalmak összegét maximalizálja. A fenti példában ha a konstans $R(s)$ negatív, akkor az ágens törekszik a végállapotokhoz, ha pozitív, akkor kerüli őket.

Mivel bárhová vetheti a sors, az ágensnek minden állapotban dönteni kell tudni, hogy merre tovább. Egy π *eljárásmód (policy)* ezt fejezi ki: egy s állapotban az ágens az $a = \pi(s)$ cselekvést választja. Az optimális eljárásmód (jelölés: π^*) az az eljárásmód, amely maximalizálja a teljesítménymérték (most jutalmak összege) *várható értékét* (az érintett állapotok, és így a jutalom is, a véletlentől függenek).



Példa: az $R(s)$ jutalom különböző értékeire az optimális π^* eljárásmód. Minél kellemetlenebb a világ az ágensnek, annál gyorsabban akar szabadulni, akár kockázatok árán is.

Ha a világ kellemes (pozitív jutalom) akkor a végállapotoktól távolodunk, a többi állapotban mindegy mit csinálunk.

13.1.1. Hasznosság és optimális eljárásmód

Az ágens teljesítménymértéke a meglátogatott állapotokban látott jutalmaktól függ. Legyen pl. egy bejárt állapotsorozat első $n + 1$ állapota

$s_0, \ldots, s_n$. Szokás a teljesítménymértéket itt *hasznosságnak (utility)* nevezni, jelölése $U(s_0, \ldots, s_n)$. Eddig: az ágens teljesítménymértéke a meglátogatott állapotok jutalmainak összege volt. Nézzük ezt meg jobban.

Tegyük fel, hogy *végtelen időhorizontunk* van, azaz az ágens működésének nincs időkorlátja. Ebben az esetben mindegy, hogy mikor kerül az ágens egy adott állapotba, a hátralevő idő ugyanannyi (végtelen), és minden más feltétel is ugyanaz, tehát az optimális döntés mindig ugyanaz. Ilyenkor az optimális eljárásmód *stacionárius (stationary)*.

Milyen hasznosságot van értelme definiálni? Belátható, hogy ha megköveteljük, hogy az ágens állapotpreferenciája is stacionárius legyen, azaz teljesüljön, hogy az $[s_0, s_1, s_2, \ldots]$ és $[s_0, s'_1, s'_2, \ldots]$ sorozatok preferenciarelációja pontosan ugyanaz legyen mint az $[s_1, s_2, \ldots]$ és $[s'_1, s'_2, \ldots]$ sorozatoké, akkor csak a következő alakú hasznosságfüggvények jöhetnek szóba:

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots = \sum_{t=0}^{\infty} \gamma^t R(s_t),$$

ahol $0 < \gamma$. Ha $\gamma = 1$ akkor a példában szereplő összeg képletet kapjuk. Ha γ nagyon kicsi, akkor azzal azt fejezzük ki, hogy szeretnénk ha az ágens nemcsak sok jutalmat gyűjtene, de azt minél gyorsabban tenné.

Fontos tulajdonság, hogy ha $\gamma < 1$ és $\forall s, R_m \geq |R(s)|$, akkor

$$|U(s_0, s_1, \ldots)| = |\sum_{t=0}^{\infty} \gamma^t R(s_t)| \leq \sum_{t=0}^{\infty} \gamma^t R_m = \frac{R_m}{1-\gamma},$$

azaz ha korlátosak a jutalmak, és $\gamma < 1$, akkor a hasznosság is korlátos, akkor is, ha végtelen sétáról van szó. Az ilyen jutalomfüggvényeket *leszámítolt jutalomnak (discounted reward)* nevezzük.

Rögzített π eljárásmód esetén, mivel Markov tulajdonságú sétáról van szó, adott s állapot rákövetkező állapotai véletlen változók, amik az

előző állapottól függnek: $s, S_1, S_2, \ldots$. Egy adott s állapot hasznosságát a π eljárásmód mellett tehát várható értékkel így definiáljuk:

$$U^\pi(s) = E(U(S_0, S_1, \ldots)|\pi, S_0 = s),$$

amiből adódik, hogy

$$\begin{aligned} U^\pi(s) &= E(\sum_{t=0}^{\infty} \gamma^t R(S_t)|\pi, S_0 = s) = \\ &= R(s) + \gamma E(\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t)|\pi, S_0 = s) = \\ &= R(s) + \gamma E(U^\pi(S_1)|\pi, S_0 = s) = \\ &= R(s) + \gamma \sum_{s'} P(S_1 = s'|s, \pi(s))U^\pi(s'). \end{aligned}$$

Adott s állapotból elérhető *maximális várható hasznosság* tehát

$$U(s) = \max_\pi U^\pi(s),$$

és legyen π^*_s egy eljárásmód, amelyre $U(s) = U^{\pi^*_s}(s)$.

Fontos észrevétel, hogy létezik egy π^* *optimális eljárásmód*, amely optimális bármely kezdőállapotból. Ennek a belátásához gondoljuk meg, hogy s állapotba érve az ágens akkor érne el maximális várható hasznosságot (ha nem kötelező rögzített eljárásmódot követni), ha onnantól π^*_s-re áll át. Viszont, mivel ez minden érintett állapotra igaz (állapotonként át kell állni), ez a stratégia éppen egy π^* eljárásmódot definiál, amelyre $\pi^*(s) = \pi^*_s(s)$ minden s állapotban.

Ennek fényében $U(s) = U^{\pi^*}(s)$, azaz egyszerűen a π^*-hoz tartozó hasznosság.

Az $U(s)$ függvény és π^* kölcsönösen meghatározzák egymást. A π^* eljárásmód ismerete esetén $U(s)$ adódik a

$$U(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi^*(s))U(s') \qquad (13.1)$$

lineáris egyenletrendszerből, amely n állapot esetén n egyenletet és n ismeretlent tartalmaz. Másfelől, ha $U(s)$ adott, akkor π^* megkapható az alábbi egyenletből:

$$\pi^*(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s'), \tag{13.2}$$

tehát olyan cselekvést javasolunk, ami maximalizálja a várható hasznosságot. Ezt az összefüggést az (13.1) egyenletbe helyettesítve kapjuk a *Bellman egyenletet* (pontosabban egyenletrendszert):

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s'), \tag{13.3}$$

amelynek a megoldása szintén az optimális hasznosság minden állapotra, csak most már nem tettük fel, hogy π^* ismert. Ennek az az ára, hogy az egyenletrendszer nemlineárissá változott. Viszont ha megoldjuk ezt az egyenletrendszert, két legyet ütünk egy csapásra: minden s állapotra megkapjuk $U(s)$-t, és $\pi^*(s)$-t is (utóbbit az (13.2) egyenlet segítségével).

13.1.2. Értékiteráció (value iteration)

Algoritmust adunk egy π^* optimális eljárásmód meghatározására. Ha sem π^*, sem $U()$ nem ismert, a (13.3) Bellman egyenletet is lehet használni az $U()$ meghatározására, bár ez nemlineáris egyenletrendszert ad. Az ötlet, hogy meghatározzuk az $U()$ hasznosságot, majd (13.2) alapján kapjuk π^*-t.

Rekurzív nemlineáris egyenleteket leginkább iterációval szokás megközelíteni. Az *értékiteráció* algoritmusának lényege, hogy az $U()$ értékeit tetszőlegesen kitöltjük: ez lesz az $U_0()$ függvény, majd a következő iterációt hajtjuk végre:

$$\forall s, U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U_i(s'), \tag{13.4}$$

Belátható, hogy az iteráció konvergál. Az ötlet az, hogy észrevesszük, hogy az iteráció egy kontrakció. Részletesebben: jelölje U_i a hasznosságokat az i. iterációban, és B pedig legyen a (13.4) egyenlet által definiált iterációs operátor. Ezekkel a jelölésekkel az iterációt írhatjuk $U_{i+1} = B(U_i)$ alakban. Ekkor belátható, hogy bármely U és U' vektorokra

$$\|B(U) - B(U')\| \leq \gamma \|U - U'\|,$$

ahol $\| \cdot \|$ a max norma, azaz $\|x\| = \max_i |x_i|$. Azaz az értékiteráció operátora a különbség maximumát csökkenti, azaz a max normára nézve kontrakció.

A kontrakciókról ismert, hogy egy fixpontjuk van, és ebbe a fixpontba konvergál az iterációjuk. Másfelől a keresett $U()$ függvény (azaz vektor) világos, hogy fixpont. Az $U()$ fügvény tehát létezik és egyértelmű, ha $0 \leq \gamma < 1$.

Megemlítjük, hogy a konvergencia lineáris (azaz a hiba exponenciálisan csökken az iterációszám függvényében), ami a gyakorlatban gyors konvergenciát jelent, valamint azt is fontos látni, hogy messze nem kell a pontos $U()$ függvényt ismerni ahhoz, hogy a (13.2) alapján kapott π^* optimális legyen, hiszen elég ha a legjobbnak tűnő cselekvés tényleg a legjobb. Tehát általában elég egy jó közelítés.

13.1.3. Eljárásmód-iteráció

Az eddigi képletek alapján máshogy is lehet iterációt tervezni: az $U()$ függvény helyett egyből az eljárásmódot is iterálhatjuk. Ehhez vegyünk egy tetszőleges π_0 eljárásmódot. Feltéve, hogy már adott π_i, számoljuk ki az U^{π_i} hasznosságfüggvényt, majd

$$\pi_{i+1}(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U^{\pi_i}(s'),$$

ami a (13.2) egyenlet adaptációja iterációs célokra. A (13.2) egyenlet éppen azt mondja, hogy π^* fixpont.

Az iterációt addig folytatjuk, amíg $\pi_{i+1} \neq \pi_i$. Mivel végesszámú különböző eljárásmód létezik, és mivel belátható, hogy minden iteráció javulást eredményez, garantáltan terminál az algoritmus.

U^{π_i} meghatározásához a Bellman egyenlet lebutított változata kell (mint a (13.1) egyenlet), ahol optimális eljárásmód helyett rögzített eljárásmódot teszünk fel:

$$U^{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))U^{\pi_i}(s'), \tag{13.5}$$

ami ezúttal egy lineáris egyenletrendszer (nincs benne max). Ezt megoldhatjuk explicit módon $O(n^3)$ időben n állapot esetén, vagy a hatékonyság drasztikus növelésére használhatunk inkább közelítést a

$$U_{j+1}^{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))U_j^{\pi_i}(s'),$$

iteráció néhányszori végrehajtásával; ekkor az eljárást *módosított eljárásmód-iterációnak* nevezzük. Ez többek között azért jó ötlet, mert választható $U_0^{\pi_i} = \hat{U}^{\pi_{i-1}}$, ahol $\hat{U}^{\pi_{i-1}}$ $U^{\pi_{i-1}}$ korábban kiszámolt közelítése. Ez egy jó kezdőérték, eleve jó közelítés, tehát csak néhány iterációt elég végrehajtani.

Megjegyezzük, hogy a fenti itárációk akár aszinkron módon is végrehajthatók (minimális feltevések mellett), amikor csak egyes állapotokat frissítünk akár az eljárásmód, akár a hasznosságérték tekintetében. Így lehet koncentrálni a „fontos” állapotokra jobban.

13.2. Megerősítéses tanulás

A feladatkörnyezet itt más: feltesszük, hogy az ágens nem ismeri a $P(s'|s, a)$ állapotátmenet-valószínűségeket (amik egy s állapotból a cselekvés hatására s'-be kerülés valószínűségét adják meg), és nem ismeri az $R(s)$ jutalmakat sem előre. A többi jellemzője a feladatkörnyezetnek változatlan. Továbbra is teljesen megfigyelhető, tehát az ágens

tudja melyik állapotban van, és ha már meglátogatott egy s állapotot, onnantól az $R(s)$ értékét is tudja.

A feladat továbbra is egy π^* optimális eljárásmód meghatározása. Mivel azonban a világ modellje ($P(s'|s,a)$ és $R(s)$) nem ismert előre, a korábbi képleteket nem lehet direktben használni. Ehelyett az ágensnek információt kell gyűjtenie a világról, és ezután, vagy eközben, meghatározni π^*-ot. Ez tehát *tanulási probléma*.

Az ágens által gyűjtött példák a következő alakot öltik: $(s_0, R(s_0), a_0), \ldots, (s_n, R(s_n), a_n)$, ahol s_i az i. állapot, $R(s_i)$ a megfigyelt jutalom, a_i az i. állapotban választott cselekvés, ami az s_{i+1} állapotba vezetett. Ilyen példából annyi áll rendelkezésre, amennyit az ágens összegyűjt. Az ágens szabadon választhatja meg a cselekvéseket, ezzel aktívan befolyásolva a példahalmazt.

13.2.1. Passzív megerősítéses tanulás

Passzív megerősítéses tanuláson azt értjük, hogy adott egy π eljárásmód, és a feladat az, hogy az ágens a tapasztalatai alapján (amit a π alkalmazása közben gyűjtött), adjon minél jobb közelítést az U^π hasznossági függvényre. (Ez a feladat bemelegítés az aktív esethez, ahol π^*-ot keressük majd.)

Most a (13.5) képlet nem alkalmazható, mert $P(s'|s,a)$ és $R(s)$ nem ismert.

Mit tehetünk? Legprimitívebb módszer a direkt közelítés, amikor is minden állapothoz a példákból meghatározzuk az empírikus hasznosságot (t.i. vesszük azokat a példákat, amelyek érintik s-et, és az s minden előfordulása után még hátralevő jutalmakból kiszámolt hasznosságok átlagát vesszük).

Ehhez sok példa kell, és nem veszi figyelembe a (13.5)-ben adott kényszerfeltételeket, ezeknek ellentmondó végeredmény is lehet, ami nem túl elegáns, főleg nem túl hatékony. Viszont nagyon sok példa után

konvergál a korrekt értékhez.

Adaptív dinamikus programozás

A direkt közelítés javítása, hogy a (13.5) összefüggést figyelembe vesszük, és U^π direkt közelítése helyett $P(s'|s,\pi(s))$-t közelítjük a tapasztalatok alapján statisztikusan, majd behelyettesítjük ezeket a (13.5) egyenletrendszerbe, a megfigyelt $R(s)$ jutalmakkal együtt, és kiszámoljuk U^π-t ahogy az eljárásmód iterációnál láttuk.

A módszert lehet alkalmazni folyamatosan, minden alkalommal frissítve az U^π közelítését, amikor új állapotba érkezik az ágens.

A probléma az, hogy ez a módszer sem skálázódik jól, ha nagyon sok állapot van (pl. dámajáték, vagy sakk, stb.), mert nagy az egyenletrendszer. Persze lehet iterációval megoldani, ahogy korábban láttuk.

Időbeli különbség tanulás (temporal difference learning)

Ugyanúgy a (13.5) összefüggésből indulunk ki mint az előbb, csak most a példák gyűjtése közben az ágens minden egyes $s \to s'$ állapotátmenet megfigyelésekor csak az $U^\pi(s)$ értéket frissíti lokálisan, tehát a többi állapot hasznosságának a közelítése változatlan marad.

Hogyan? Intuitív ötlet: ha $P(s'|s,\pi(s)) = 1$ lenne, akkor tudnánk, hogy $U^\pi(s) = R(s) + \gamma U^\pi(s')$, és ez egyben akár jó iterációs szabály is lenne. De $P(s'|s,\pi(s))$ ismeretlen. Viszont ha nem írjuk át teljesen $U^\pi(s)$-t, csak elmozdítjuk $R(s) + \gamma U^\pi(s')$ irányába egy kicsit, akkor ezek a kis elmozdítások a lehetséges különböző s' álapotok felett kiátlagolódnak éppen a $P(s'|s,\pi(s))$ valószínűségeknek megfelelően. Kicsit emlékeztet ez a sztochasztikus gradiens módszerre.

Tehát a frissítési szabályunk ez lesz:

$$\begin{aligned} U^\pi(s) =&(1-\alpha)U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s')) = \\ &= U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s)), \end{aligned} \tag{13.6}$$

ahol $0 < \alpha < 1$ a tanulási ráta, hasonlóan a gradiens módszerekhez. U^π-t tetszőlegesen inicializáljuk az algoritmus futása előtt.

Konvergál ez a módszer? Ha α konstans, akkor nyilván nem, hiszen s állapotban minden különböző s' rákövetkező állapot másfelé húzhatja. Tehát α-nak csökkenni kell. Belátható, hogy ha pl. $\alpha \approx 1/n$, ahol n az s állapot meglátogatásainak a száma (tehát minden állapotnak külön α-ja van), akkor növekvő példahalmaz mellett a korrekt értékhez konvergál $U^\pi(s)$.

Fontos megjegyezni, hogy a módszer nem tárolja explicit módon a világ modelljét (állapotátmenet-valószínűségeket), azaz *modell-mentes*.

A időbeli különbség tanulás pontosan egy állapotot frissít egyetlen átmenet alapján, az adaptív dinamikus programozás pedig minden állapotot az összes átmenet eddigi statisztikái alapján. Ez két szélsőség, a kettő közötti átmenet amikor csak bizonyos állapotokat frissítünk, amelyek valószínű szomszédai pl. egy küszöbértéknél többet változtak (prioritásos végigsöprés (prioritized sweeping) algoritmus).

13.2.2. Aktív megerősítéses tanulás

Most a π nem adott, hanem egy optimális π^* közelítése a cél.

Adaptív dinamikus programozás

Az adaptív dinamikus programozás ötlete működik továbbra is, csak most a (13.3) egyenletet kell használni, amiben szükség van minden s állapotban az összes $a \in A(s)$ cselekvéshez tartozó állapotátmenet-valószínűségekre (és nem csak egy rögzített eljárásmód szerinti cselekvéshez tartozóakra, mint a (13.5) egyenletben, amit korábban használtunk).

Azonban ezeket szintén lehet közelíteni a megfigyelt példák statisztikáival. Ezeket a közelítéseket a (13.3) egyenletekbe behelyettesítve pl.

a korábban látott értékiteráció módszerével frissítjuk az optimális $U(s)$ hasznosság közelítését, és a (13.2) képlet alapján választunk cselekvést.

A gond csak az, hogy miközben gyűlnek a példák, az eljárásmód is változik, visszahatva a példák eloszlására; az ágens arrafelé fog sétálni, ahol véletlenül korán talált jutalmat (mivel mindig csak a rendelkezésére álló megfigyelések alapján optimális eljárásmódot követi), és pl. az ismeretlen részeket kerülheti, ezért gyakran ragad lokális optimumba ez a naiv módszer.

Szokás ezt mohó viselkedésnek hívni.

Mint sok más korábbi területen, itt is fontos tehát a kihasználás (exploitation) és a felfedezés (exploration) egyensúlya: néha véletlen vagy ismeretlen helyeket is ki kell próbálni.

Ötlet: ne U-t használjuk közvetlenül a példák gyűjtésére, hanem mellette vezessünk be egy optimista U^+ hasznossági függvényt, amiben mesterségesen növeljük a felfedezési szellemet, és használjuk az U^+ hasznosságot a (13.2) egyenletbe helyettesítve adódó eljárásmódot a példagyűjtésre.

Ezt megtehetjük, mert nem kötelező a tanulás alatt levő modell ($U()$ és átmeneti valószínűségek) által adott mohó eljárásmódot használni a modell tanulásához, a példagyűjtés használhat más eljárásmódot! Az ilyen tanulást *eljárásmódon kívüli* (off-policy) tanulásnak nevezzük.

Továbbra is azt feltételezve, hogy az értékiterációt használjuk a hasznosság meghatározására, az U^+ kiszámításához módositsuk a korábbi (13.4) képletet így:

$$\forall s,\, U^+_{i+1}(s) = \\ = R(s) + \gamma \max_{a \in A(s)} f\left(\sum_{s'} P(s'|s,a) U^+_i(s'), N(s,a) \right), \qquad (13.7)$$

ahol $N(s,a)$ azoknak az eseteknek a száma, ahol az s állapotban az a cselekvés került alkalmazásra. Az $f(u,n)$ függvény határozza meg az

exploráció mértékét: u-ban nő, n-ben csökken. Sokféle f-et választhatunk, pl. egy egyszerű választás a következő: $f(u,n) = R^+$, ha $n \leq N_e$, egyébként $f(u,n) = u$. Magyarul, ha a releváns tapasztalatok száma kisebb mint egy előre adott N_e explorációs paraméter, akkor egy optimista R^+ hasznosságot ad vissza (ami pl. a jutalmak egy felső korlátja), egyébként pedig a tényleges hasznosságot.

Ez azt eredményezi, hogy a még ismeretlen helyzetek távolról is vonzzák az ágenst, hatásuk a (13.7) képletnek köszönhetően távolabbi állapotok hasznosságában is megjelenik.

Egy másik gyakori, egyszerűbb módszer az ϵ-mohó (ϵ-greedy) felfedező stratégia, amelyben minden lépésben ϵ valószínűséggel egy véletlen cselekvést választunk, egyébként pedig az aktuális $U()$ szerint döntünk. Itt érdemes csökkenteni az ϵ értékét tanulás közben.

Időbeli különbség tanulás és Q-tanulás

Az időbeli különbség tanulás (13.6) képlettel adott frissítési szabálya változatlan formában alkalmazható, miközben a követett π eljárásmód nem rögzített, hanem a hasznosság aktuális közelítése alapján a mohó eljárásmód kell legyen.

Viszont az ágens csak az állapotátmenet-valószínűségek ismeretében tudja a (13.2) képlet alapján meghatározni a mohó cselekvést, tehát itt az állapotátment-valószínűségeket is közelíteni kell, így most nem kapunk modellmentes módszert mint a passzív esetben.

Van viszont modellmentes módszer: a Q-tanulás. A $Q(s,a)$ függvény az a cselekvés értéke az s állapotban:

$$\begin{aligned} Q(s,a) &= R(s) + \gamma \sum_{s'} P(s'|s,a) U(s') = \\ &= R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a' \in A(s')} Q(s',a') \end{aligned}$$

ahol a második egyenlőségnél felhasználtuk a (13.3) Bellman egyenle-

tet. Ebből az időbeli különbség tanulás egy Q-verziója adódik:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)) \quad (13.8)$$

Itt nem kell a környezet modellje, mert az optimális cselekvés egyszerűen

$$\pi^*(s) = \arg\max_{a \in A(s)} Q(s,a)$$

Végül: itt is használhatók a korábban említett $f(u,n)$ explorációs függvények, pl. úgy, hogy az ágens a tapasztalatgyűjtés során az $\arg\max_{a \in A(s)} f(Q(s,a), N(s,a))$ cselekvést hajtja végre, nem az optimálisat, így fel tud fedezni ismeretlen régiókat. Az ϵ-mohó stratégia itt is egyszerűbb alternatíva az exploráció megvalósítására.

A Q-tanulás is eljárásmódon kívüli (off-policy) tanulás.

13.2.3. Általánosítás

A $Q()$ és $U()$ függvények eddig táblázattal voltak adva. Ha túl sok az állapot, ez nem működik. Megoldás: paraméterezett függvényközelítés. Pl. lineáris jellemzőkombináció:

$$U(s) \approx \hat{U}(s;\theta) = \theta_1 f_1(s) + \ldots + \theta_n f_n(s)$$

Ahol a $\theta = (\theta_1, \ldots, \theta_n)$ paramétervektor határozza meg a függvényt. Nem csak lineáris közelítés lehet, pl. neurális hálókat is használhatunk.

Előnyök: (1) a függvényreprezentáció tömörítése (2) az általánosításra való képesség (még nem látott állapotra is valami „értelmeset” mond).

Hátrány: mivel közelítésről van szó, esetleg nem reprezentálható a tényleges $U()$ függvény megfelelő pontossággal. De legtöbbször nincs más megoldás, mert túl sok az állapot.

Q-tanulás

Tanulásra gyakran a Q-tanulás paraméteres változatát használjuk ahol a $\hat{Q}(s, a; \theta) \approx Q(s, a)$ függvény θ paramétereit tanuljuk. A paraméteres eset is modellmentes, tehát elég a $\hat{Q}$-t tanulni, nem kell a modellt is közelíteni. Paraméteres esetben először az

$$\mathrm{Err}(s, a; \theta^{(t)}) = \frac{1}{2}(R(s) + \gamma \max_{a' \in A(s')} \hat{Q}(s', a'; \theta^{(t-1)}) - \hat{Q}(s, a; \theta^{(t)}))^2 \tag{13.9}$$

hibafüggvényt definiáljuk, ahol $\theta^{(t)}$ a t. lépés paramétervektora. A frissítési szabályban $\theta^{(t)}$-t úgy akarjuk változtatni, hogy a hibafüggvény csökkenjen. Tehát a hibafüggvény deriváltját kivonjuk egy α tanulási rátával megszorozva:

$$\begin{aligned}\theta_i^{(t+1)} &= \theta_i^{(t)} + \\ &+ \alpha \left(R(s) + \gamma \max_{a' \in A(s')} \hat{Q}(s', a'; \theta^{(t-1)}) - \hat{Q}(s, a; \theta^{(t)}) \right) \frac{\partial \hat{Q}(s, a; \theta^{(t)})}{\partial \theta_i^{(t)}}\end{aligned} \tag{13.10}$$

Itt is lehetséges az off-policy tanulás, pl. ϵ-mohó eljárásmód adja a sétát.

Két technika a tanulás stabilizálására: visszajátszás (replay) és késleltetett frissítés. A (13.9) hibafüggvényt két dolog határozza meg: egy $(s, a, R(s), s')$ négyes, amely a séta követése során keletkezik, és a $\theta^{(t-1)}$ és $\theta^{(t)}$ paraméterezések.

A visszajátszás technikája során a korábban meglátogatott $(s, a, R(s), s')$ négyesekből veszünk egy véletlen mintát, és az alapján hajtjuk végre a frissítést (nem pedig a legutóbbi alapján). Akár több mintát is vehetünk, és ezek alapján frissíthetünk, és ezt bármikor megtehetjük, nem csak amikor éppen lép az ágens.

A késleltetett frissítés technikája azt jelenti, hogy a (13.10) egyenletben nem a $\theta^{(t-1)}$-et használjuk, hanem egy korábban rögzített θ' paramétervektort használunk sok frissítéshez, mielőtt attérnénk az aktuális θ

használatára, és ezután ez lesz az új rögzített θ' egy darabig, és így tovább.

Ez a két technika jelentősen javítja a Q-tanulás teljesítményét, különösen sok paraméter esetén, mert általuk felügyelt tanulási feladatok sorozatává alakul a feladat, a Q paraméterei stabilabban konvergálnak.[1]

Eljárásmód keresés (policy search)[2]

Magát az eljárásmódot is lehet direkt módon tanulni ($Q()$ vagy $U()$ függvények nélkül). Most is tegyük fel, hogy az eljárásmódot paraméterezett függvény formájában keressük θ paraméterrel, sőt legyen sztochasztikus az eljárásmód (mert így differenciálható lehet a paraméteres reprezentáció): $\pi(a|s,\theta)$ annak a valószínűsége, hogy s állapotban a akciót választjuk.

Legyen $\rho(\theta)$ az eljárásmód várható hasznossága:

$$\rho(\theta) = E\left(\sum_{t=0}^{H} \gamma^t R(S_t)|\theta\right)$$

ahol H a horizont, ami lehet végtelen is. Itt feltesszük, hogy véges. Az $S_1, \ldots, S_H$ állapotsorozat eloszlását a $\pi(a|s,\theta)$ eljárásmód adja és az S_0 véletlen állapot (feladatfüggő eloszlással). A $\rho(\theta)$ hasonló az állapotok hasznosságához, de itt az S_0 kezdőállapot is véletlen változó.

A $\rho(\theta)$ függvény $\nabla_\theta \rho(\theta)$ gradiensét kell közelíteni valahogy, és akkor a korábbi gradiens módszer alkalmazható. Modellmentes megközelítés (REINFORCE algoritmus). Legyen $\tau = (s_0, a_0 \ldots, s_{H-1}, a_{H-1}, s_H)$

[1]Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015. (doi:10.1038/nature14236)

[2]Jan Peters. Policy gradient methods. Scholarpedia, 5(11):3698, 2010. revision #137199. (doi:10.4249/scholarpedia.3698)

egy trajektória amit *kigurítással* (roll out) kapunk, azaz alkalmazzuk a π eljárásmódot, és így

$$P(\tau|\theta) = P(s_0) \prod_{t=0}^{H-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t, \theta) \tag{13.11}$$

Vezessük be az $R(\tau)$ jelölést:

$$\rho(\theta) = E\left(\sum_{t=0}^{H} \gamma^t R(S_t)|\theta\right) = E(R(\tau)|\theta) = \sum_{\tau} P(\tau|\theta)R(\tau)$$

ahonnan

$$\begin{aligned}\nabla_\theta \rho(\theta) = \sum_{\tau} \nabla_\theta P(\tau|\theta)R(\tau) \stackrel{1}{=} \sum_{\tau} P(\tau|\theta)\nabla_\theta \ln P(\tau|\theta)R(\tau) = \\ = E(\nabla_\theta \ln P(\tau|\theta)R(\tau)|\theta)\end{aligned} \tag{13.12}$$

ahol az (1) egyenlőség a „REINFORCE trükk”. Ez azért kell, hogy—mivel logaritmusok gradiensével dolgozunk—ki fognak esni az átmeneti valószínűségek a gradiens képletből, modellmentes módszert kapunk. A (13.11) egyenlet alapján

$$\nabla_\theta \ln P(\tau|\theta) = \sum_{t=0}^{H-1} \nabla_\theta \ln \pi(a_t|s_t, \theta) \tag{13.13}$$

ahol a feladatkörnyezetet leíró valószínűségek kiestek, hiszen additívak és nem függnek θ-tól.

A fentiek során a gradienst hoztuk kedvező alakra: egyrészt a feladatkörnyezet modellje kiesett, másrészt a trajektóriák fölött vett várható értékként fejeztük ki. Utóbbi lehetővé teszi, hogy torzításmentesen közelítsük ezt a várható értéket konkrét trajektóriák fölött vett átlag segítségével. Állítsunk elő K darab trajektóriát: $\tau^{(1)}, \ldots, \tau^{(K)}$. Ekkor, a

(13.12) egyenletbe a (13.13) képletet, és a trajektóriák hasznosságának a definícióját behelyettesítve, és empírikus közelítésre (azaz átlagra) áttérve

$$\nabla_\theta \rho(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=0}^{H-1} \nabla_\theta \ln \pi(a_t^{(i)} | s_t^{(i)}, \theta) \right] \sum_{t=0}^{H-1} \gamma^t R(s_t^{(i)}) \quad (13.14)$$

Ez már lehetővé teszi, hogy a θ paramétereket tanuljuk gradiens módszerrel:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta \rho(\theta)|_{\theta=\theta^{(t)}}$$

Konkrét feladat esetében a π eljárásmód θ paraméterezése sokféle lehet, gyakran mesterséges neurális háló.

13.3. Monte Carlo fakeresés[3]

Tegyük fel, hogy most ismét adott az $R(s)$ jutalom és a $P(s'|s,a)$ átmenetvalószínűség, de utóbbi nem explicit, hanem *generatív* módon: tudunk mintát venni egy $S \sim \mathbf{P}(S|s,a)$ véletlen változóból a cselekvés tényleges végrehajtása nélkül, azaz tudjuk *szimulálni* az átmeneteket.

Tegyük fel továbbá, hogy az állapotok száma nagyon nagy. Ágens előtt két lehetőség: a modell segítségével paraméteres reprezentációk előre betanulása (13.2.3 fejezet), vagy valós időben *tervezés*. A kettő kombinálható.

[3]C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games, 4(1):1–43, March 2012. (doi:10.1109/TCIAIG.2012.2186810)

13.3.1. Ritka mintavételezés (sparse sampling)[4]

Az ágens s_0 állapotban van, ahonnan egy fakeresést végez a generatív modell segítségével. A fában állapotcsúcsok és cselekvéscsúcsok vannak. Egy s állapotcsúcs gyerekei az s-ben végrehajtható cselekvések. Egy (s, a) cselekvéscsúcs gyerekei pedig C db. állapotcsúcs: $s'_1, \ldots, s'_C$, ahol s_i a $\mathbf{P}(S|s, a)$ eloszlásból vett független minta, és C egy konstans.

A fa H állapotátmenetet szimulál, tehát minden levél egy állapotcsúcs amely egy H ugrást tartalmazó, s_0-ból induló szimulációnak felel meg.

A fa kiértékelése: minden s állapotcsúcshoz kiszámolunk rekurzívan egy $\hat{U}(s)$ hasznosságértéket, és minden (s, a) cselekvéscsúcshoz egy $\hat{Q}(s, a)$ értéket. Ezekkel a pontos $U(s)$, ill. $Q(s, a)$ értéket szeretnénk közelíteni. A rekurzív képletek:

$$\hat{U}(s) = \begin{cases} R(s) & \text{ha } s \text{ egy levél állapotcsúcs} \\ \max_a \hat{Q}(s, a) & \text{egyébként} \end{cases}$$

$$\hat{Q}(s, a) = R(s) + \gamma \frac{1}{C} \sum_{i=1}^{C} \hat{U}(s'_i)$$

A választott cselekvés pedig nem más mint $\arg\max_a \hat{Q}(s_0, a)$.

Lényeg: a futásidő csak H-tól, C-től, és a cselekvések számától függ, de nem függ az állapotok számától. A C lehet sokkal kisebb mint az állapotok száma (ezért hívják ritka mintavételezésnek). A H és C paraméterek pedig beállíthatók úgy, hogy az $\hat{U}(s)$ közelítések előre adott pontosságúak legyenek, ha $0 < \gamma < 1$ és az $R()$ jutalomfüggvény korlátos.

Probléma: minden lehetséges cselekvéssorozatot azonos súllyal vizsgálunk, függetlenül azok minőségétől, így túl nagy fát kell kiértékelni.

[4]Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. Machine Learning, 49(2):193–208, 2002. (doi:10.1023/A:1017932429737)

13.3.2. UCT algoritmus[5]

Ötlet: a keresés során az ígéretes cselekvéssorozatok mentén egyre több mintát kellene venni, és egyre mélyebbre kellene építeni a fát, míg a kevésbé ígéretes irányokra egyre kevesebb energiát kellene pazarolni. Ugyanakkor azért minden lehetőséget elégszer ki kell próbálni, hogy megalapozott döntéseket hozhassunk arról, hogy melyik irány jó vagy rossz.

Ez ismét a korábban már látott felfedezés/kihasználás (exploration/exploitation) dilemma. Az UCT algoritmus fő ötlete, hogy az UCB1 (upper confidece bound) algoritmust használja a fa építésére, ami egy ismert algoritmus a *többkarú rabló* (multi-armed bandit) probléma megoldására. A többkarú rabló probléma pedig a felfedezés/kihasználás dilemma matematikai megfogalmazása.

UCB1 algoritmus

A többkarú rabló problámában adott $a_1, \ldots, a_K$ cselekvés, más néven kar. Minden kar véletlen mennyiségű jutalmat ad ha meghúzzuk, és a jutalom eloszlása a karra jellemző. Minden iterációban egy kart meghúzunk, a feladat pedig minél hamarabb megtalálni a legjobb kart, pontosabban szólva maximalizálni az összjutalmat várható értékben.

Az UCB1 algoritmus alapötlete, hogy minden karra egy konfidencia intervallumot számolunk ki a múltbeli minták száma és értéke alapján, és mindig a legmagasabb felső korlátú kart húzzuk meg. Kevés minta esetén az intervallum széles, tehát a felső korlátja magas (exploration), viszont a legjobb kar felső korlátja mindig magas marad (exploitation).

Az UCB1 algoritmus először minden kart meghúz pontosan egyszer.

[5] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, Proceedings of the 17th European Conference on Machine Learning (ECML), pages 282–293, Berlin, Heidelberg, 2006. Springer. (doi:10.1007/11871842_29)

Ezután legyen a t időpontban N_i az a_i karhoz tartozó meghúzások száma, és $\overline{Q}_i$ a húzások során kapott jutalmak átlaga. Ekkor az UCB1 algoritmus a $t+1$ időpontban az a_j kart húzza meg, ahol

$$j = \arg\max_i \left(\overline{Q}_i + \sqrt{\frac{2 \ln t}{N_i}} \right)$$

Itt feltettük az egyszerűség kedvéért, hogy a jutalmak értéke a $[0,1]$ intervallumból jön.

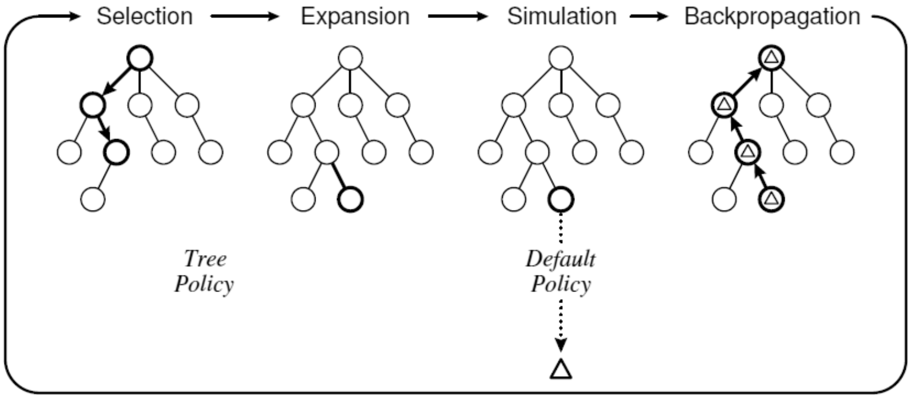
Kigurítás (rollout) alapú fakeresés

Az UCT algoritmus a ritka mintavételezés algoritmusával szemben kigurítások segítségével fokozatosan növeszt fát, amit a memóriában tárol (tehát a fa nem csak virtuálisan létezik).

A fa csúcsai állapotok. Egy s csúcsban tároljuk az $N(s), N(s,a), \overline{Q}(s,a)$ értékeket, amelyek az s látogatási száma, s-ben a cselekvés kipróbálási száma, és a kipróbálások tapasztalt hasznosságainak átlaga.

```
kigurításos-fakeresés(s_0)
1 s_0 hozzáadása gyökérként
2 while(megállási feltétel nem teljesül)
3   s_l = levélKeres(s_0)
4   U_l = kiértékel(s_l)
5   visszaterjeszt(s_l, U_l)
6 return argmax_a( Q(s_0, a) )
```

A megállási feltétel: elfogy az idő vagy a memória. A levél keresése során s_0-ból mindig az UCB1 algoritmus szerinti cselekvéseket választva megyünk amíg a rákövetkező állapot már nincs a fában. Ezt a hiányzó rákövetkező állapotot felvesszük új levélként. A levélkeresés során



választott cselekvés s állapotban az UCB1 algoritmus szerint az az a cselekvés, ahol vagy $N(s, a) = 0$ (ha van ilyen), vagy (ha már nincs ilyen)

$$a = \arg\max_{a_i} \left(\overline{Q}(s, a_i) + \sqrt{\frac{2 \ln N(s)}{N(s, a_i)}} \right).$$

Levél állapot kiértékelése: véletlen cselekvéseket választva végállapotig, vagy elég nagy számú lépésig szimuláljuk az ágenst, és a tapasztalt (leszámított) hasznosság lesz az érték. A visszaterjesztés után a mezők a következők lesznek: jelölje $s_0, a_0, \ldots, s_{l-1}, a_{l-1}, s_l$ a gyökérből az s_l levélig vezető utat. Ekkor az s_k csúcsban ($0 \leq k < l$)

$$\overline{Q}(s_k, a_k) = \frac{1}{N(s_k, a_k) + 1} \left(N(s_k, a_k)\overline{Q}(s_k, a_k) + \sum_{i=k}^{l-1} \gamma^{i-k} R(s_i) + \gamma^{l-k} U_l \right), \tag{13.15}$$

$$N(s_k) = N(s_k) + 1,$$

$$N(s_k, a_k) = N(s_k, a_k) + 1,$$

ahol U_l a levél (vagy végállapot) kiértékelése.

13.3.3. Háttértudás használata

A fenti algoritmusok csak a generatív modellt használták. Használhatunk azonban háttértudást, heurisztikákat is.

Ha van közelítő $\hat{U}(s)$ heurisztikánk az $U(s)$ függvényre, akkor a kiértékelés visszaadhatja egyszerűen az $U_l = \hat{U}(s_l)$ értéket.

Ha van közelítő becslésünk az optimális eljárásmódra, amely megad egy $\pi(a|s)$ heurisztikus valószínűséget (s-ben a cselekvés valószínűségét), akkor az UCB1 algoritmus helyett a levél keresése során alkalmazhatjuk a *progressive bias* módszerét. Ekkor s állapotban a választott a cselekvés a következő lesz:

$$a = \arg\max_{a_i} \left(\overline{Q}(s, a_i) + \frac{\pi(a_i|s)}{N(s, a_i) + 1} \right)$$

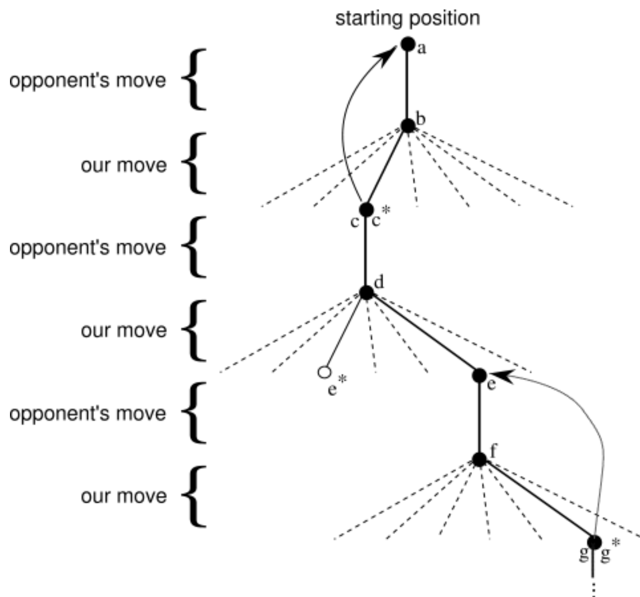
Itt feltettük, hogy $Q(s, a_i) \in [0, 1]$, de normalizálással ez elérhető ha ismert a felső és alsó korlát.

Végül, a $\pi(a|s)$ értékeket használhatjuk a kiértékelés során is a végállapotig történő kigurításra (a véletlen cselekvésválasztás helyett). Az így kapott tapasztalati hasznosságot kombinálhatjuk az $\hat{U}()$ függvénnyel, pl. vehetjük az $U_l = (\hat{U}(s_l) + U_\pi(s_l))/2$ értéket, ahol $U_\pi()$ a kigurításból kapott érték.

13.4. Játékelméleti kapcsolódási pontok[6]

Vegyük a kétszemélyes zéró összegű játékokat. Az egyik ágens szempontjából a másik ágens a környezet része. Jutalom csak a játék legvégén, azaz a végállapotokban van, a többi állapotban nulla, és $\gamma = 1$ (azaz feltesszük implicit, hogy véges a játék).

[6]Ábra forrása: Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, The MIT Press, `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/`



Az első lehetőség, hogy az ellenfél rögzített (de ettől még lehet nemdeterminisztikus, ekkor a lépések eloszlása rögzített), és specifikusan ez ellen az ellenfél ellen keressük a legjobb stratégiát. Ekkor használhatunk aktív időbeli különbség (TD) tanulást utánállapotokra (afterstate), ahol az s, s' az ábra visszanyilainak felelnek meg, pl. $s = a, s' = c$:

$$U(s) = \begin{cases} R(s) & \text{ha } s \text{ egy végállapot} \\ U(s) + \alpha(U(s') - U(s)) & \text{egyébként} \end{cases}$$

$U()$ itt az utánállapot (afterstate) hasznossági függvény. Másszóval az ellenfél állapotain valósítjuk meg a TD tanulást. Azért működik itt a TD tanulás, mert a mohó lépésválasztás megvalósítható, mivel az állapotátmenet determinisztikus és ismert (játékszabályok). Pl. az ϵ-mohó eljárásmódot használhatjuk, és a mohó lépéseken tanítunk csak.

Aktív TD tanulás az ágens saját állapotain nem lenne modellmentes, mivel a környezet (azaz az ellenfél) nem ismert. A sima Q-tanulás mű-

ködik viszont. Hátránya, hogy sokkal több értéket tanulunk mint a TD tanulásnál ((s, a) párokhoz rendelünk hasznosságot, míg a TD csak az s állapotokhoz). Előnye, hogy off-policy módszer, tehát rugalmasabban gyűjthetünk tanítópéldákat.

Lehet nemdeterminisztikus is a játék (kockadobás). Ettől még működik továbbra is az afterstate TD algoritmus, mivel a mohó heurisztika megvalósítható, hiszen a kockadobás lehetséges kimenetelei és az ezek feletti eloszlás ismert, tehát a maximális várható értékű akció kiválasztható.

Természetesen mindez működik általánosítva is (ahol paraméterezett függvényekkel közelítjük a Q ill. U táblázatokat, és gradiens módszerrel frissítjük ezeket).

A rögzített ellenfelet legyőző ágens helyett próbálhatjuk a minimax (vagy expecti-minimax) játékost megtanulni. Ebben az esetben a minimax felfogás helyett célszerűbb negamaxban gondolkodni, tehát a két játékos azonos abszolút értékű, de ellentétes előjelű jutalmat kap a végállapotokban, viszont mindkét játékos maximalizál. Ekkor a hagyományos megerősítéses tanulási problémát kapjuk, csak a következő játékoshoz tartozó állapotokban a hasznosságot -1-gyel szorozni kell a mohó döntéseknél és a frissítési szabályokban is. Így egyazon tanulás során mindkét játékos (MIN és MAX) stratégiája is előáll. Felfoghatjuk ezt úgy is, hogy „önmaga ellen játszik” az ágens (hiszen egy sakkozó is játszik világossal és sötéttel is).

Ezzel a pici módosítással tehát minden korábbi technika használható a minimax ágens policy-jének a tanulására. Pl. az eljárásmód keresés is alkalmazható játékokban, hiszen ott is egy H horizontig néztük a kigurításokat, és az alapján tanultunk. Legyen θ ismét a $\pi(s|a, \theta)$ eljárásmód paraméterezése. Állítsunk elő K darab trajektóriát: $\tau^{(1)}, \ldots, \tau^{(K)}$ és legyenek ezeknek a végeredményei $R^{(1)}, \ldots, R^{(K)}$.

Belátható, hogy használhatjuk a következő gradienst:

$$\nabla_\theta \rho(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} \sum_{t=1}^{H-1} \nabla_\theta \ln \pi(a_t^{(i)} | s_t^{(i)}, \theta) z_t^{(i)},$$

ahol $z_t^{(i)}$ a t. lépésben lépő ágens szempontjából értelmezett jutalom, tehát $z_t^{(i)} = \pm R^{(i)}$. Itt felhasználtuk, hogy minden kigurítás minden lépése alapján lehet tanulni. Iterálni is lehet, és ekkor az ágens játszhat önmaga korábbi verzióival is. Tetszőleges fix ágens ellen is levezethetünk a (13.14)-hez hasonló képletet, hasonló okoskodással, ahol csak a saját ágensünk lépéseire kell szummázni.

A Monte Carlo fakeresés is használható játékfákban.

Nemdeterminisztikus esetben láttuk: várható-minimax. A ritka mintavételezés alkalmazható. Pl. ha kockadobás szerepel a játékban (CHANCE csúcs) akkor nem az összes kimenet lesz a leszármazottja, hanem C db. konkrét dobás, és ezek értékének az átlaga lesz a CHANCE csúcs értéke. Ez akkor segít, ha C sokkal kisebb mint a lehetséges kimenetek.

Az UCT algoritmus is alkalmazható. Csak a visszaterjesztés fog különbözni. Egy kigurítás s_T végállapotának kiértékelése után minden lépésben a tapasztalt jutalom $z_k = \pm R(s_T)$, annak függvényében, hogy a k. lépés melyik játékosé. Ennek alapján a visszaterjesztés (13.15) egyenlete helyett a

$$\overline{Q}(s_k, a_k) = \frac{1}{N(s_k, a_k) + 1} \left(N(s_k, a_k) \overline{Q}(s_k, a_k) + z_k \right), \qquad (13.16)$$

adódik. Háttértudást is ugyanúgy lehet használni.

14. fejezet

Az MI filozófiája

Az MI-t úgy definiáltuk mint ami a racionális ágensekkel foglalkozik, nem az emberi intelligenciával.

Most térjünk vissza az emberi (öntudat, gondolkodás, elme) és a mesterséges intelligencia viszonyára.

Gyenge MI hipotézis: készíthetünk olyan „gépeket”, amik *látszólag* intelligensek, de „valójában” nem feltétlenül van „elméjük” (ill. öntudatuk).

Erős MI hipotézis: készíthetünk olyan „gépeket”, amelyeknek „valódi” „elméjük” van.

Vigyázat: (1) Mi a „gép”? pl. az óra gép, a számítógép gép, az ember nem gép (*wetware?...*) Hol a határ? Mi ennek a jelentősége? (2) Mi az „elme”? Nekem van. Másoknak is van? Az órának nincs. Halaknak? Ha másnak is van, olyan mint az enyém? Vannak különböző elmék? (nyelv szerepe)

Tehát: lehetséges álláspontok: gyenge MI tagadása (még látszólagos intelligencia sem érhető el), gyenge MI elfogadása miközben erős MI tagadása, elfogadása vagy ignorálása.

14.1. Gyenge MI: lehetséges?

Miért ne? Bár még messze vagyunk (sőt, mint láttuk, nem is igazán cél). De sokan a gyenge MI-t sem fogadják el.

Fontos, hogy a gyenge MI legalább ellenőrizhetőnek tűnik empirikusan (definíciója empirikus jellegű feltételeket szab).

14.1.1. Turing teszt

A gyenge MI empirikus tesztje. Sok variációja van, pl.: a kísérletező ember írott üzeneteket vált öt percig egy számítógéppel vagy egy másik emberrel, és el kell döntenie, hogy ember-e a partner. (Loebner prize, részletes szabályok).

Fontos, hogy mindegy, *hogyan* csinálja a program, a lényeg, hogy *mit* csinál (ez a gyenge MI lényege).

A Turing tesztet (mint a gyenge MI empirikus tesztjét) sok szempontból lehet kritizálni. Pl. (1) a viselkedés egy szűk szelete, nem vizsgálunk hang- vagy képfeldolgozást, motorikus képességeket. (2) Képzetlen, gyanútlan tesztelőket nagyon könnyű átverni, pl. ELIZA pszichoanalitikus program...

14.1.2. Gyenge MI elleni érvek

1. *Egy gép sosem lesz képes X-re!* Miért ne?

2. *Gödel nemteljességi tétele:* az ember mindig többet lát mint bármely formális rendszer, tehát bármely formális rendszernél intelligensebb. De!

   (a) A Turing gép végtelen memóriájú. Véges rendszerre nem merülnek fel a problémák. Az ember végtelen memóriájú

vajon? Az Univerzum lehet, hogy maga is végesállapotú rendszer. Releváns ez egyáltalán?

(b) A nemteljességi tételek központjában önreferencia van; de ez az emberekre is vonatkozik. Persze lehet tágabb, „meta” elméleteket gyártani, de ezeknek is van Gödel mondata.

(c) A legfontosabb: az emberi intelligencia és gondolkodás *nem* matematikai jellegű. A tudás nem konzisztens se nem formális. A Gödel tétel tehát *irreleváns*, érdektelen az MI szempontjából: nem a szimbólummanipuláló formális rendszerek az érdekesek, hanem a bizonytalanság, inkonzisztencia modellezése, a heurisztikus, informális gondolkodás, asszociációk, mintafeldogozás, stb., kezelése. Ezeknek a modelljei is persze „formálisak” lesznek, de az intelligencia lényege nem a tételbizonyításra vezethető vissza.

3. *GOFAI (Good Old Fashioned AI) problémái*: hasonló a fenti érveléshez: a formális, logikai rendszereknek vannak problémái, de nem csak ilyen megközelítések vannak, pl. statisztikus tudásreprezentáció és tanulás. Más szóval van nem-propozícionális tanulás (pl. neurális háló), nem-felügyelt tanulás, megerősítéses tanulás, stb.

14.2. Erős MI: lehetséges?

Tegyük fel, hogy egy gép átmegy a tökéletesített Turing teszten: teljesen emberien viselkedik (beszél). Ekkor van elméje vagy nincs?

Két szélsőség: (1) Van elméje, pusztán a viselkedéséből erre következtetünk (2) Az emberi agy elengedhetetlen az elméhez.

Általában hogyan döntjük el, hogy valami „igazi” vagy nem?

1. vihar: egy számítógépes szimuláció nem igazi vihar; mi a helyzet a szél- és esőgéppel keltett viharral?

2. tokaji aszú: ha atomról atomra másolom, akkor se lesz tokaji, csak másolat (miért is? mert a tokaji borászok nem keresnek rajta? vagy elveszik egy „esszenciális” tulajdonság?)

3. ló: atomról atomra másoljuk, akkor lovat kapunk vagy nem?

Feszültség van itt is a tesztelhető tulajdonságokon alapuló, és az *esszencialista* megközelítések között. Az elme esetében mi a kulcs?

1. öntudat és qvália (qualia)? (Ez lehet, hogy kell, de egyesek szerint nem elég). Meg lehet figyelni? Közvetlenül nem, de neurális korrelátumokat lehet mérni.

2. funkcionális egyezés (azaz az elme állapotainak oksági leírása egyezzen az emberével)? De itt az elme modelljeinek egyezéséről beszélünk; nem lakozhat az elme mégiscsak a modellek által elabsztrahált dolgokban? Nem kell ehhez tökéletes modell?

3. biológiai hardver kell? (wetware). Ez hasonlít a funkcionális modellekkel való érvelésre.

Etikai kérdések is felmerülnek: igazából csak a saját elmémről tudom, hogy létezik, a többiekéről csak felteszem. De akkor mikről tegyem fel? Saját csoport? Ország? Faj? Idegenek?

A szabad akarat kérdése is felmerül néha; de ez főleg érzelmi kérdés már.

A fő eszköz egyelőre a *gondolatkísérlet*, amivel nagyon kell vigyázni, de jobb mint a semmi.

14.2.1. Gondolatkísérletek

Agyak egy lavórban

Ha a világ csak szimuláció, vajon lehet-e igazi az elme (hiszen a mentális állapotai nem igazi dolgokat jelölnek). Ebből az egyik tanulság, hogy a mentális állapotokra vonatkozó megkötésekkel vigyázni kell a definíciókban.

Agyprotézis

Cseréljük ki az agyban a neuronokat egyenként azonos működésű műneuronokra (tegyük fel, hogy ezt meg tudjuk tenni, de jegyezzük meg, hogy ez messze nem világos, aztán pl. nem biztos, hogy csak a neuronok számítanak, pl. fehérjék is, stb.). Tehát a gondolatkísérlet alapállása funkcionalista, hiszen felteszi, hogy lehetséges a neuronok tökéletesen azonos funkciójú műneuronnal való cseréje.

Kérdés: igazi marad-e az elme mindeközben. Van aki szerint az öntudat fokozatosan elveszik (?), míg a viselkedés változatlan (hiszen ez feltétel volt).

Ha az öntudat elveszhet míg a viselkedés változatlan, akkor az öntudatnak nincs oksági szerepe a fizikai jelenségekre. Tehát *epifenomén*.

Kínai szoba

Nagyon híres gondolatkísérlet.

Adott egy szoba, benne egy ember aki nem beszél kínaiul. A szobán van egy ablak amelyen át kínai írásjelek jönnek. Az embernek van egy nagy könyve, benne szabályok, amelyek segítségével legyárt kínai írásjeleket (miközben fogalma sincs mit jelentenek), és ezeket kiküldi az ablakon. (A szoba nem kell, az ember memorizálhatja a könyvet.)

Tegyük fel, hogy a szoba átmegy a Turing teszten. Az embernek van öntudata. Akkor ez erős MI? Nem (mondja Searl). Miért ne (mondják mások)? Pl. a szoba mint egész esetleg tud kínaiul.

Vegyük észre, hogy a szoba a tárolt programú számítógép metaforája. Ugyanazt a viselkedést nagyon sok Turing géppel el lehet érni. Pl. ha van egy programom, akkor azt futtathatom univerzális gépen. Sőt, az egész univerzális gépet a programmal együtt futtathatom egy másik univerzális gépen, stb.

Világos, hogy a viselkedés ugyanaz, míg a belső állapotok elég különbözőek. Az érv tehát az, hogy nem plauzibilis, hogy mindegyiknek van tudata, de akkor melyiknek van? Csak egynek? Egynek sem?

14.3. Nyelv, intencionalitás, öntudat

Állítás: (1) szociális közegben a (2) modellezési képesség tökéletesedésével a nyelv és öntudat kb. egyszerre jelenik meg.

Melyik ez a pont? Az *intencionalitás*: modellezni tudom (megértem) a fajtársak (és más lények) mentális állapotait: hit, szándék, cél, motiváció. Szokták *népi pszichológiának (folk-psychology)* nevezni. Ekkor már

- logikus igény a másik mentális állapotainak befolyásolására: emiatt a kommunikáció spontán megjelenik
- saját magamat is ugyanúgy tudom már modellezni mint másokat (énkép), (sőt kommunikálni is tudok magammal): öntudat fontos eleme
- a nyelv belsővé válik: a gondolkodás a belső beszéd, az öntudat narratíva saját magamról

- evolúciós adaptációk is rásegítenek (beszédszervek, kognitív képességek) hiszen a nyelvnek és mások manipulációjának szelektív előnye van: önerősítő folyamat. De a nyelv megjelenése az általános kognitív képességek fejlődésének köszönhető, az evolúció ezt „kapja fel”.

14.4. MI fejlődése: veszélyek és etikai problémák

14.4.1. Kreativitás

Ha „túl jó” a kereső algoritmus, és a célfüggvény nem pontos.

Pl. ápoló robot: ha célfüggvény a páciens szenvedésének a minimalizálása, akkor pl. a páciens megölése nullára csökkenti.

Pl. takarító robot: ha a por mennyiségét minimalizáljuk, akkor pl. a szenzorok tönkretétele az érzékelt port nullára csökkenti, tehát optimális megoldás.

Ez *nagyon komoly veszély*, talán a legkomolyabb az MI-ben: ha nem korlátozzuk a kreativitást valahogy, akkor gond lesz, mert pontos célfüggvényt szinte lehetetlen adni.

14.4.2. Öntudat

Etikai problémák (kikapcsolás, szétszerelés, stb.).

Van spontán kialakulásra esély? Miért ne? (korábban láttuk: megfelelő kognitív képességek + szociális környezet kell hozzá).

Pl. (1) egy mélytengeri búvárrobot: nem szociális; (2) takarító robot: szociális, de egyszerű feladat, nem komplex a kognitív igény; (3) ápoló vagy focista robot: ilyen helyeken a legnagyobb az esély. (Persze nem

csak robotokra igaz, virtuális ágensekre is érvényes).

Fontos: elvileg *spontán* bekövetkezhet, egyfajta fázisátmenet során, ha a kognitív képességek egy kritikus szintet elérnek, és elég komplex a környezet is.

14.4.3. Hatalomátvétel

Kedvelt sci-fi téma.

Elvileg rosszul tervezett célfüggvény, és túl kreatív rendszer esetén gond lehet, de nem valószínű, hogy ne lehetne időben leállítani (?)

Valószínűbb a kiborg forgatókönyv: örök élet, korlátlan intelligencia/erő.

Lényeg: a célfüggvény az eredeti emberi célfüggvény (hatalom, pénz, stb...)