## 15.4. Programok mint valószínűségi modellek

Számos valószínűségi programozási nyelv arra az ötletre épült, hogy a valószínűségi modellek minden olyan programozási nyelvben definiálhatók végrehajtható kóddal, amely magában foglalja a véletlenszerűség forrását. Az ilyen modelleknél a lehetséges világok a végrehajtási nyomvonalak, és az ilyen nyomvonalak valószínűsége az a nyomvonal végrehajtásához szükséges véletlenszerű választások valószínűsége. Az így létrehozott PPL-ek a mögöttes nyelv minden kifejező erejét öröklik, beleértve az összetett adatstruktúrákat, a rekurziót és bizonyos esetekben a magasabb rendű függvényeket is. Sok PPL valójában számításelméletileg univerzális: bármilyen valószínűségi eloszlást reprezentálhat, amit egy megállni képes valószínűségi Turing-gép mintavételezésével lehet nyerni.

### 15.4.1. Példa: Szövegolvasás

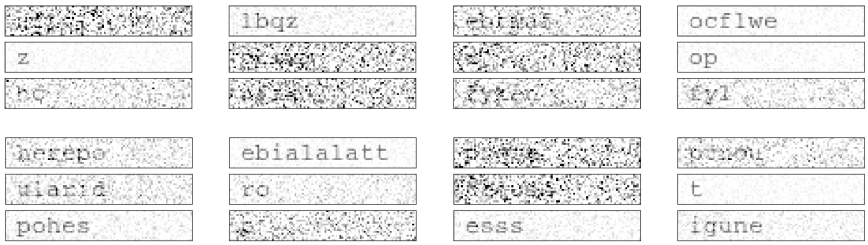
A valószínűségi modellezés és következtetés eme megközelítését egy degradált szöveget olvasó program írásával illusztráljuk. Az ilyen típusú modelleket lehet építeni a vízkár miatt elmosódott vagy homályos, vagy a nyomtatásra használt papír elöregedése miatt foltos szövegek olvasására. Ilyen modellek bizonyos típusú CAPTCHA-k feltörésére is építhetők. A 15.11. ábra egy két komponenst tartalmazó generatív programot mutat: (i) a betűsorozat létrehozásának módját, és (ii) e betűk zajos, elmosódott megjelenítésének generálását a rendelkezésre álló grafikus könyvtárral. A Generate-Image kilencszeri meghívásával generált példaképeket a 15.12. ábra mutatja (fent).

```
function GENERATE-IMAGE() returns egy kép betűkkel
    betűk ← GENERATE-LETTERS(10)
    return RENDER-NOISY-IMAGE(betűk, 32, 128)

function GENERATE-LETTERS(λ) returns betűsorozat
    n ~ Poisson(λ)
    betűk ← [ ]
    for i = 1 to n do
        betűk[i] ~ UniformChoice({a, b, c, ···})
    return betűk

function RENDER-NOISY-IMAGE(betűk, szélesség, magasság) returns zajos kép a betűkről
    tiszta_kép ← RENDER(betűk, szélesség, magasság, szöveg_alsó = 10, szöveg_bal = 10)
    zajos_kép ← [ ]
    zaj_variancia ~ UniformReal(0.1, 1)
    for sor = 1 to szélesség do
        for oszlop = 1 to magasság do
            zajos_kép[sor, oszlop] ~ N(tiszta_kép[sor, oszlop], zaj_variancia)
    return zajos_kép
```

**15.11. ábra** Generatív program optikai karakterfelismeréshez egy nyílt világ valószínűségi modelljéhez. A generatív program degradált képeket állít elő, amelyek betűsorozatokat tartalmaznak. Az egyes sorozatok generálását követően azokat 2D-s képpé alakították át és minden pixelhez additív zajt adtak hozzá.




**15.12. ábra** A felső panelen tizenkét degradált kép látható, amit a 15.11. ábra generatív programjának végrehajtásával kaptuk. A betűk száma, azok identitása, az additív zaj mennyisége és a pixelenkénti specifikus zaj mind a modell valószínűségi tárgytartományának része. Az alsó panelen tizenkét degradált kép látható, amit a 15.15. ábra generatív programjának végrehajtásával kaptuk. A betűk Markov-modellje jellemzően olyan betűsorozatokat eredményez, amelyeket könnyebb kiejteni.

### 15.4.2. Szintaktika és szemantika

A **generatív program** egy olyan végrehajtható program, amelyben minden véletlen választás egy véletlen változót határoz meg a kapcsolódó valószínűségi modellben. Képzeljük el, hogy lépésről lépésre visszakövetjük egy olyan program végrehajtását, amely véletlenszerű döntéseket hoz. Legyen $X_i$ a program $i$-edik véletlen választásához tartozó véletlen változó. Szokás szerint az $x_i$ az $X_i$ lehetséges értékeit jelöli. Nevezzük a generatív program **végrehajtási nyomá**t, vagyis a véletlenszerű választásokhoz tartozó lehetséges értékek sorozatát $\omega = \{x_i\}$-nek. A program egyszeri futtatása létrehoz egy ilyen nyomot, ebből származik a „generatív program” kifejezés.

Generatív program

Végrehajtási nyoma

Az összes lehetséges végrehajtási nyom $\Omega$ tere a generatív program által definiált valószínűségi modell mintaterének tekinthető. A nyomok feletti valószínűségi eloszlás az egyes véletlen választási valószínűségek szorzataként definiálható: $P(\omega) = \prod_i P(x_i|x_1, \ldots x_{i-1})$. Ez analóg egy OUPM-beli lehetséges világok feletti eloszlással.

Koncepcionálisan egyszerű bármilyen OUPM-et a megfelelő generatív programmá konvertálni. Ez a generatív program véletlen választásokat generál minden numerikus állításhoz és az ilyen numerikus állítások által implikált minden alap véletlen változó értékéhez. A generatív program többletmunkája az olyan adatstruktúrák létrehozása, amelyek az OUPM objektumait, függvényeit és a lehetséges világainak relációit képesek reprezentálni. Ezeket az adatstruktúrákat az OUPM következtetési motor automatikusan létrehozza, mert az OUPM felteszi, hogy minden lehetséges világ egy elsőrendű modellstruktúra, míg egy tipikus PPL ilyen feltételezést nem tesz.

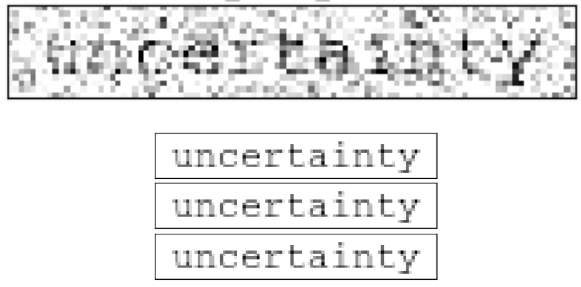
A 15.12. ábrán látható képek segítségével intuitív módon megérthetjük a $P(\Omega)$ valószínűségi eloszlást. Különböző mértékű zajt látunk, és a kevésbé zajos képeken változó hosszúságú betűsorozatokat. Legyen $\omega_1$ az ábra jobb felső sarkában lévő, a `ocflwe` betűket tartalmazó képnek a nyoma. Ha ezt a $\omega_1$ nyomot Bayes-hálózattá göngyölítenénk fel, akkor a hálónak 4104 csomópontja lenne: 1 csomópont az $n$ változóhoz, 6 csomópont a *betűk*$[i]$ változókhoz, 1 csomópont a *zaj_variancia* változóhoz és 4096 csomópont a *zajos_kép* képpontjaihoz. Látjuk tehát, hogy ez a generatív program egy nyílt világ valószínűségi modellt határoz meg. Az általa elkövetett véletlen választások száma nincs eleve korlátozva, hanem az $n$ véletlen változó értékétől függ.

### 15.4.3. Következtetési eredmények

Alkalmazzuk ezt a modellt az additív zajjal degradált betűképek értelmezésére. A 15.13. ábra egy degradált képet mutat, valamint három független MCMC-futtatás eredményét. Minden futtatásnál megmutatjuk a betűk interpretációját a Markov-lánc leállítása utáni nyomban. Mindhárom esetben az eredmény az `uncertainty` betűsorozat, ami arra utal, hogy a posterior eloszlás nagymértékben koncentrálódik a helyes értelmezésre.

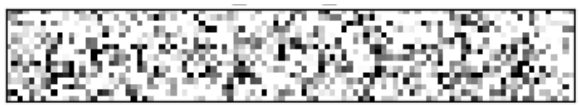
Most pedig rontsuk tovább a szöveget, összemosva annyira, hogy az emberek számára nehezen olvasható legyen. Ezen a nagyobb kihívást jelentő bemeneten a következtetés eredményét a 15.14. ábra mutatja. Ezúttal, bár úgy tűnik, hogy az MCMC-következtetés konvergált a (tudomásunk szerint) helyes betűszámhoz, az első betűt tévesen `q`-nak azonosították és a következő tíz betűből ötöt illetően bizonytalanság van.

Ezen a ponton számos lehetséges módja van a eredmények értelmezésének. Lehetséges, hogy az MCMC-következtetés megfelelően kevert és az eredmények jól tükrözik a valódi posteriort, adott modell és kép mellett. Ebben az esetben az egyes betűk bizonytalansága és a hiba az első betűben elkerülhetetlen. A jobb eredmények érdekében esetleg javítanunk kell a szövegmodellt vagy csökkentenünk kell a zajszintet. Az is lehet, hogy az MCMC-következtetés nem megfelelően keveredett. Ha 300 láncot 25 ezer vagy 25 millió iterációban futtatnánk, lehet, hogy egészen más eredményeloszlást találnánk, ami talán arra utalna, hogy az első betű valószínűleg inkább `u`, mint `q`.




**15.13. ábra** Zajos bemeneti kép (fent) és a 15.11. ábra modelljének három futtatásával, egyenként 25 MCMC-iterációval előállított következtetési eredmények (lent). Vegyük észre, hogy a következtetési folyamat a betűk sorrendjét helyesen dönti el.

Több következtetés lefuttatása pénzben és várakozási időben költséges lehet . Sőt, nincs megbízható teszt Monte Carlo következtetési módszerek konvergenciájára. Megpróbálhatnánk javítani a következtetési algoritmuson, talán egy jobb javaslateloszlás tervezésével MCMC-hez, vagy alulról felfelé képi jellegek használatával, jobb kezdő hipotéziseket javasolva. Ezek a fejlesztések további átgondolást, implementációt és hibakeresést igényelnek. A harmadik alternatíva a modell javítása. Például beépíthetnénk az angol szavak tudását, amilyen például a betűpárok valószínűségét. Mi most ezt a lehetőséget vesszük fontolóra.



| | |
|---|---|
| gnucitainty | uncertainty |
| gncurtbinty | uncsitainty |
| gncuftaintv | unchftsinty |

**15.14. ábra** Fent: Rendkívül zajos bemeneti kép. Balra lent: A 15.11. ábra függetlenbetű-modelljének 25 MCMC-iterációjával előállított három következtetési eredmény. Jobbra lent: A 15.15. ábra betű-bigram-modelljével előállított három következtetési eredmény. Mindkét modell kétértelműséget mutat az eredményekben, de az utóbbi modell eredményei a valószínű betűsorozatok prior ismeretét tükrözik.

```
function Generate-Markov-Letters(λ) returns betűsorozat
    n ~ Poisson(λ)
    betűk ← [ ]
    betű_valószínűségek ← Markov-Initial()
    for i = 1 to n do
        betűk[i] ~ Categorical(betű_valószínűségek)
        betű_valószínűségek ← Markov-Transition(betűk[i])
    return betűk
```

**15.15. ábra** Generatív program egy javított optikai karakterfelismerési modellhez, amelyben betűket egy betű-bigram-modell szerint generálunk, amelynek páronkénti betűfrekvenciáit az angol szavak listájából becsüljük.

### 15.4.4. A generatív program javítása Markov-modell bevonásával

A valószínűségi programozási nyelvek moduláris felépítésűek, ami megkönnyíti a mögöttes modell javító fejlesztését. A betűket nem független módon, hanem inkább szekvenciálisan generáló javított modell generatív programját a 15.15. ábra mutatja. Ez a generatív program egy Markov-modellt használ, amely minden betűt az adott előző betű függvényében sorsol, az átmenetvalószínűségeket pedig az angol szavak referencialistájából becsüljük meg. A 15.12. ábra ezzel a generatív programmal generált tizenkét mintaképet jelenít meg. Figyeljük meg, hogy a betűsorozatok lényegesen angolszerűbbek, mint a 15.11. ábra programjából generáltak. A 15.14. ábra jobb oldali paneljén az ebből a Markov-modellből származó következtetési eredményeket mutatja a nagy zajjal terhelt képre alkalmazva. Az értelmezések jobban megfelelnek a generáló nyomnak, bár még mindig van némi bizonytalanság.

### 15.4.5. Következtetés generatív programokban

Az OUPM-ekhez hasonlóan a generatív programokban a pontos következtetés általában megfizethetetlenül költséges, vagy lehetetlen. Ellenben könnyen belátható, hogy hogyan lehet elutasító mintavételezést végezni. Futtassuk a programot, csak azokat a

nyomokat tartsuk meg, amelyek egyetértenek a bizonyítékokkal, és számoljuk meg a különböző lekérdezési válaszokat, amiket ezekben a nyomokban megtalálunk. A valószínűségi súlyozás is egyszerű. Minden egyes generált nyomnál figyelemmel kísérjük a nyom súlyát az út során megfigyelt összes érték valószínűségének szorzatával.

A valószínűségi súlyozás csak akkor működik jól, ha az adatok megfelelően valószínűek a modell szerint. Nehezebb esetekben általában az MCMC a legjobb módszer. Az MCMC alkalmazása a valószínűségi programokra magában foglalja a végrehajtási nyomok mintavételezését és módosítását. Sok, az OUPM-eknél felmerülő szempont itt is érvényes. Emellett az algoritmusnak óvatosnak kell lennie a végrehajtási nyomok módosításánál, mint például a ha–akkor állítások eredménye, ami érvényteleníthet a nyom többi részét.

A következtetés további javulása több kutatásból származik. Egyes javítások alapvető változásokat idézhetnek elő azon problémák osztályában, amelyek egy adott PPL-lel még elvileg is megoldhatók. A korábban az RVM-eknél leírt következtetésemelésnek lehet ilyen hatása. Sok esetben az általános MCMC túl lassú, és speciális célú javaslateloszlások szükségesek ahhoz, hogy a következtetési folyamat gyorsan keveredhessen.

A PPL-ekkel kapcsolatos közelmúltbeli munkák egyik fontos szempontja az volt, hogy minél könnyebb legyen a felhasználók számára definiálni és használni az ilyen javaslateloszlásokat úgy, hogy a PPL-következtetés hatékonysága megegyezzen a konkrét modellekhez kidolgozott célorientált következtetés hatékonyságával.

Számos ígéretes megközelítés célja a valószínűségi következtetés többletterhelésének csökkentése. A 13.4.3. alfejezetben a Bayes-hálókhoz leírt fordítási ötlet az OUPM-ekben és PPL-ekben szintén alkalmazható következtetésre, és jellemzően két-három nagyságrendnyi gyorsulást eredményez. Az olyan algoritmusokhoz, mint az üzenettovábbítás és az MCMC, *speciális célú hardverre* vonatkozó javaslatok is születtek. A Monte Carlo hardver például kihasználja az alacsony pontosságú valószínűségi reprezentációkat és a masszív finomszemcsés párhuzamosságot, hogy 100–10 000-szeres sebesség- és energiahatékonysági növekedést tegyen lehetővé.

Adaptív javaslateloszlás

A tanuláson alapuló módszerek a sebességet is lényegesen javíthatják. Az **adaptív javaslateloszlások** fokozatosan megtanulhatják, hogyan kell létrehozni olyan MCMC-javaslatokat, amelyek észszerű valószínűséggel elfogadásra kerülhetnek és észszerűen hatékonyak a modell valószínűségi tér feltárásában a gyors keverés biztosítása érdekében. Lehetséges a mélytanulási modellek tanítása (lásd a 21. fejezetben) a fontossági mintavételhez szükséges javaslateloszlások reprezentálására, az alapul szolgáló modellből előállított szintetikus adatok felhasználásával.

Általában elvárható, hogy a programozási nyelvekre épülő formalizmusok a kiszámíthatóság korlátjába fognak ütközni, és ez a helyzet a PPL-ek esetében is. Ha azonban feltételezzük, hogy az alapul szolgáló program minden bemenetre és minden véletlen választásra terminálódik, akkor a valószínűségi következtetés elvégzésének további követelménye még mindig eldönthetetlenné teszi a problémát? Kiderül, hogy a válasz igen, de csak a végtelen pontosságú folytonos változókat tartalmazó számítási modellre. Ebben az esetben lehetségessé válik olyan kiszámítható valószínűségi modell megfogalmazása, amelyben a leállási probléma a következtetésben kódolt. Másrészt véges pontosságú számokkal és a valós alkalmazásokban jellemzően használt sima valószínűségi eloszlásokkal a következtetés továbbra is eldönthető marad.

## 19.7. Nemparametrikus modellek

A lineáris regresszió a tanítóadatokat használja arra, hogy egy rögzített paraméterhalmaz értékeit meghatározza. Ez definálja a $h_{\mathbf{w}}(\mathbf{x})$ hipotézisünket, és ekkor a mintapéldáinkat eldobhatjuk, mivel az információkat mind összegeztük a $\mathbf{w}$-ben. Azt a tanuló modellt, amely az adatokat egy rögzített (a tanítóminták számától független) méretű paraméterhalmazban összegzi, **parametrikus modell**nek nevezzük. Parametrikus modell

Ha az adathalmaz kicsi, akkor észszerű erős megszorításokat tennünk a megengedhető hipotézisekre, hogy elkerülük a túlilleszkedést. Viszont ha milliónyi vagy milliárdnyi tanítóadatunk van, amikből tanulhatunk, akkor jobbnak tűnik, ha az adatok inkább maguk hatnak, és nem erőltetjük, hogy hatásukat egy pici vektoron keresztül fejtsék ki. Ha az adatok azt mutatják, hogy az igazi megoldás nagyon girbe-gurba függvény, akkor nem szabad egyenesre vagy kevéssé görbült függvényekre korlátozzuk magunkat.

**Nemparametrikus modell**nek nevezzük azokat a modelleket, amelyek nem jellemezhetőek paraméterek korlátos halmazával. Például a 19.1. ábrán látható szakaszonként lineáris függvény az összes adatpontot megtartja a modell részeként. Az ilyen tanuló eljárásokat **példaalapú tanulás**nak úvagy **memóriaalapú tanulás**nak is nevezik. A legegyszerűbb példaalapú tanuló eljárás a **táblázat**: Vegyük az összes tanítómintát, és vegyük fel őket egy táblázatba. Aztán amikor $h(\mathbf{x})$-ra vagyunk kíváncsiak, akkor nézzük meg, hogy $\mathbf{x}$ szerepel-e a táblázatban, ha igen, akkor adjuk vissza a neki megfelelő $y$-t. Nemparametrikus modell Példaalapú tanulás Táblázat

Ennek a módszernek az a problémája, hogy nem általánosít jól: ha $\mathbf{x}$ nem szerepel a táblázatban, akkor nincs információnk az elfogadható értékről.

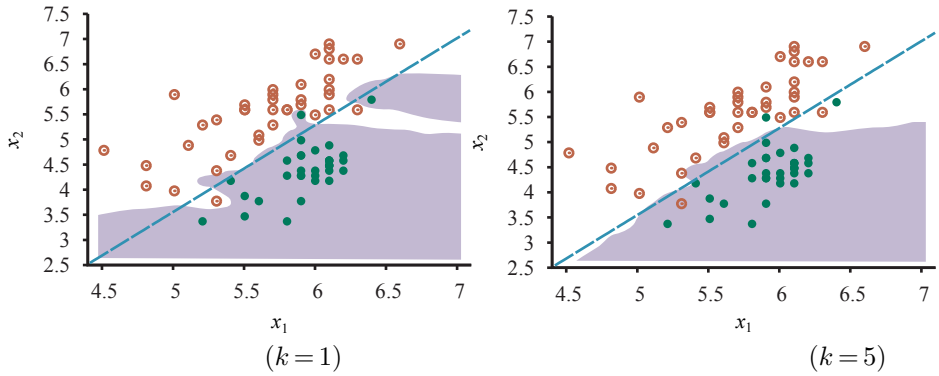
### 19.7.1. Legközelebbi szomszéd modellek

Egy kis módosítással javíthatunk a táblázatos módszeren: ha van egy $\mathbf{x}_q$-ra vonatkozó kérdésünk, akkor a helyett, hogy olyan példát keresnénk, amelyik azonos $\mathbf{x}_q$-val, $k$ olyan példát keresünk, amelyek a *legközelebb* vannak $\mathbf{x}_q$-hoz. Ezt $k$ **legközelebbi szomszéd** módszernek nevezzük. Az $\mathbf{x}_q$ pont $k$ legközelebbi szomszédjának halmazát $NN(k, \mathbf{x}_q)$-val fogjuk jelölni. Legközelebbi szomszéd

Osztályozást úgy végzünk, hogy megkeressük az $NN(k, \mathbf{x}_q)$ szomszédokat, és a leggyakoribb választ fogadjuk el válaszként–például ha $k = 3$ és a kimeneti értékek $\langle Igen, Nem, Igen \rangle$, akkor az osztályozás eredménye *Igen* lesz. Ahhoz, hogy bináris osztályozásnál elkerüljük a szavazategyenlőséget, rendszerint páratlan $k$-t használunk.

Regressziót úgy végzünk, hogy a $k$ szomszéd mediánját vagy átlagát vesszük, vagy megoldhatjuk a szomszédokon definiált lineáris regressziós problémát. A 19.1. ábra szakaszonként lineáris függvénye egy (triviális) lineáris regressziót old meg az $\mathbf{x}_q$ jobb- és baloldalán lévő egy-egy adatpontra. (Amennyiben az $x_i$ adatpontok ekvidisztánsak, akkor ezek lesznek a legközelebbi szomszédok.)

A 19.19. ábra a $k$-legközelebbi szomszéd osztályozás döntési határát mutatja $k = 1$ és 5 esetre a 19.15. ábra földrengés adathalmazán. A nemparametrikus modellek éppúgy ki vannak téve az alul-, illetve túlilleszkedésnek, mint a parametrikus modellek. Az ábrán bemutatott esetben az 1-legközelebbi szomszéd túlilleszkedik; túlreagálja a fekete kilógó adatot a jobb felső részen, és a fehér kilógó adatot az (5,4;3,7) pontban. Az




**19.19. ábra** (a) Egy $k$-legközelebbi-szomszéd modell, amely a 19.15. ábra adatain $k=1$ esetre mutatja a robbantás osztály kiterjedését. A túlilleszkedés nyilvánvaló. (b) $k=5$ alkalmazásával erre az adathalmazra megszűnik a túlilleszkedés.

5-legközelebbi szomszéd döntési határ jó, nagyobb $k$ alulilleszkedést okozna. Szokásos módon keresztvalidáció alkalmazható a legjobb $k$ kiválasztására.

Maga a „legközelebbi” szó is egy távolságmetrikát sugall. Hogyan mérjük az $\mathbf{x}_q$ lekérdezési pont és egy $\mathbf{x}_j$ mintapont távolságát? Tipikusan a távolságot a **Minkowski-távolság** vagy más néven $L^p$ norma segítségével mérjük, amelynek definíciója:

Minkowski-távolság

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p} .$$

$p=2$ esetén ez az euklideszi távolságot, $p=1$ esetén a Manhattan-távolságot adja. Logikai attribútumértékeknél azon attribútumok számát, amelyekben a két pont eltér, **Hamming-távolság**nak nevezzük. Gyakran akkor használunk euklideszi távolságot, ha a dimenziók hasonló tulajdonságokat mérnek, mint például részek szélessége, magassága és mélysége. Viszont Manhattan-távolságot használunk, ha nem hasonlóak, mint például egy páciens kora, súlya és neme. Vegyük észre, hogy ha csak a nyers számokat használjuk az egyes dimenziók mentén, akkor távolságot bármely dimenzió mértékegységének változása is befolyásolja. Például ha megváltoztatjuk a *magasság* dimenzió mértékegységét méterről mérföldre, miközben a *szélesség* és *mélység* dimenzióit változatlanul hagyjuk, akkor egészen más legközelebbi szomszédokat fogunk kapni. Szokásos megközelítés a **normalizálás** alkalmazása mindegyik dimenzióra. Minden egyes dimenzióra kiszámíthatjuk az értékek $\mu_i$ átlagát és $\sigma_i$ szórását, majd az $x_{j,i}$ értéket átskálázhatjuk az $(x_{j,i} - \mu_i)/\sigma_i$ normalizált értékre. A **Mahalanobis-távolság** néven ismert bonyolultabb távolságmetrika figyelembe veszi a dimenziók közti keresztkovarianciát is.

Hamming-távolság

Normalizálás

Mahalanobis-távolság

Ha a tér alacsony dimenziós és rengeteg adatunk van, a legközelebbi szomszéd módszerek nagyon jók: valószínű, hogy elég sok közeli pontunk van ahhoz, hogy jó választ kapjunk. Viszont ahogy a dimenziók száma növekszik, problémába ütközünk: a legközelebbi szomszédok a sokdimenziós terekben rendszerint nem igazán közeliek! Vizsgáljuk meg a $k$ legközelebbi szomszédokat egy $N$ pontból álló adathalmazban, amikor a pontok egyenletesen oszlanak el egy $n$-dimenziós egységoldalú hiperkocká-

ban. Egy pont $k$-szomszédságát azzal a legkisebb hiperkockával definiáljuk, amelyik tartalmazza a $k$ legközelebbi szomszédját. Legyen $\ell$ a szomszédság-hiperkocka átlagos élhossza. Ekkor a ($k$ pontot tartalmazó) szomszédság térfogata $\ell^n$, míg az összes $N$ pontot tartalmazó kocka térfogata 1. Tehát átlagosan $\ell^n = k/N$. Vegyük az $n$-dik gyökét mindkét oldalnak, az $\ell = (k/N)^{1/n}$ összefüggéshez jutunk.

Nézzük meg konkrétan a $k = 10$ és $N = 1\,000\,000$ esetet. Kétdimenziós esetben ($n = 2$; egységoldalú négyzet) az átlagos szomszédságra $\ell = 0{,}003$, ami egy kis töredéke az egységoldalú négyzetnek, de még 3 dimenziós esetben is $\ell$ csak 2%-a az egységoldalú kocka oldalhosszának. Viszont amikor a 17 dimenziós esethez érünk, $\ell$ a fele lesz az egységoldalú hiperkocka oldalhosszának, és 200 dimenziónál már 94%. Ezt a problémát a **dimenzionalitás átká**nak nevezzük.

Dimenzionalitás átka

Másik módja a probléma vizsgálatának: vegyük azokat a pontokat, amelyek az egységoldalú hiperkocka egy vékony, 1%-nyi külső héjába esnek. Ezek kilógó adatok, általában nehéz lesz jó értéket találni számukra, mert nem interpolálnunk, hanem inkább extrapolálnunk kell. Egydimenziós esetben ezek a kilógó adatok csak az egységhosszúságú szakasz pontjainak 2%-át teszik ki (azok a pontok, amelyekre $x < 0{,}01$ vagy $x > 0{,}99$), de 200 dimenzió esetén a pontoknak több mint 98%-a esik ebbe a vékony héjba – szinte az összes pont kilógó lesz. Ha előrelapozunk a 19.20. ábra (b) részéhez, akkor láthatunk egy kilógó adatokra történő rossz legközelebbi szomszéd illeszkedést.

Az $NN(k, \mathbf{x}_q)$ függvény koncepcionálisan triviális: adott $N$ példa halmaza és egy $\mathbf{x}_q$ lekérdezési pont, iteráljunk végig a példákon, és mérjük meg egyenként $\mathbf{x}_q$ távolságát tőlük, aztán tartsuk meg a $k$ legjobbat. Ha megelégszünk olyan megvalósítással, amelyik $O(N)$ futási időt igényel, akkor itt a történet vége. Viszont a példa alapú módszereket nagy adathalmazokra tervezték, így gyorsabb megvalósítást szeretnénk. A következő két alfejezet bemutatja, hogyan lehet fákat és hash-táblákat használni a számítás gyorsításához.

### 19.7.2. Legközelebbi szomszédok keresése $k$-d fákkal

Tetszőleges dimenziójú adatok esetén egy kiegyensúlyozott bináris fát **k-d fá**nak nevezünk, ha egy $k$-dimenziós fa. A $k$-d fa konstrukciója hasonló a kiegyensúlyozott bináris fáéhoz. Egy mintahalmazzal kezdünk, és a gyökércsomópontnál kettéosztjuk azt az $i$-dik dimenzió mentén, $x_i \leq m$ tesztet végezve el, ahol $m$ a minták mediánja az $i$-dik dimenzió mentén. Így a minták fele jut a bal oldali ágra, fele a jobb oldalira. Ezután rekurzívan konstruálunk egy-egy fát a bal és a jobb oldali mintahalmazra, amíg kevesebb, mint két minta marad. Azt, hogy melyik dimenzió alapján osszuk szét az adatokat, egyszerűen $i \bmod n$ alapján határozhatjuk meg a fa $i$-edik szintjén. (Vegyük észre; ahogy haladunk lefele a fában, a minták szétosztásánál többször is szükségünk lehet egy adott dimenzió tesztelésére.) Egy másik stratégia, hogy azt a dimenziót válasszuk, amely mentén legnagyobb az adathalmaz kiterjedése.

K-d fa

A $k$-d fán való pontos keresés ugyanolyan, mint a bináris fa esetén (azzal a kis bonyodalommal, hogy figyelni kell arra; melyik csomópontban melyik dimenziót teszteljük). Viszont a legközelebbi szomszéd keresés nehezebb. Ahogy haladunk lefelé az ágakon, és felezzük a mintahalmazt, bizonyos esetekben eldobhatjuk a minták felét. De nem minden esetben. Időnként a vizsgált pont nagyon közel esik az elválasztó ha-

tárhoz. Maga a vizsgált pont lehet például a határ bal oldalán, míg a $k$ közül egy vagy több legközelebbi szomszédja a jobb oldalon.

Tesztelnünk kell ezt az eshetőséget, kiszámítva a vizsgált pont távolságát a kettéosztás határától, és mindkét oldalon keresni, ha nem találunk $k$ mintát a bal oldalon, amelyek közelebb vannak, mint ez a távolság. E miatt a probléma miatt a $k$-d fák használata akkor javasolt, ha sokkal több példa van, mint dimenzió, lehetőleg legalább $2^n$ példánk legyen. Tehát a $k$-d fák alkalmazása jó választás kb. 10 dimenzióig, ha több ezer, vagy 20 dimenzióig, ha több millió példánk van.

### 19.7.3. Lokációérzékeny hash

A hash-táblák potenciálisan a bináris fáknál is gyorsabb keresést tesznek lehetővé. Viszont hogyan találhatunk legközelebbi szomszédokat, amikor a hash-kódok *pontos* egyezésen alapulnak? A hash-kódok véletlenszerűen osztják szét a kosarakba az értékeket, de mi azt szeretnénk, ha a szomszédos pontokat összecsoportosítaná ugyanabba a kosárba: egy **lokációérzékeny hash**elést (LÉH) akarunk.

Lokációérzékeny hash

Hash-ek felhasználásával nem tudjuk pontosan megadni az $NN(k, \mathbf{x}_q)$-t, de randomizált algoritmusok okos felhasználásával találhatunk egy *közelítő* megoldást. Először definiáljuk a **közelítőleg legközelebbi szomszédok** problémát: adott a mintapontoknak egy halmaza és egy $\mathbf{x}_q$ lekérdezési pont; találjunk meg nagy valószínűséggel egy olyan mintapontot, amelyik közel van $\mathbf{x}_q$-hoz. Pontosabban azt kívánjuk meg, hogy ha van egy olyan $\mathbf{x}_j$ pont, amelyik $\mathbf{x}_q$ $r$ sugarú környezetében van, akkor az algoritmus nagy valószínűséggel találjon egy $\mathbf{x}_{j'}$ pontot, amelyik az $\mathbf{x}_q$-tól $c\,r$ távolságon belül van. Ha nem található pont az $r$ sugarú környezetben, akkor az algoritmus visszatérhet kudarc jelzésével. A $c$ érték, illetve a „nagy valószínűség” az algoritmus hiperparaméterei.

Közelítőleg legközelebbi szomszédok

A közelítőleg legközelebbi szomszédok megtalálásához szükségünk van egy olyan $g(\mathbf{x})$ hash függvényre, amelyik rendelkezik azzal a tulajdonsággal, hogy bármely két $\mathbf{x}_j$ és $\mathbf{x}_{j'}$ pont esetén kicsi annak valószínűsége, hogy a hash-kódjuk megegyezik, ha a távolságuk nagyobb $c\,r$-nél, de nagy a valószínűség, ha a távolságuk kisebb, mint $r$. Az egyszerűség kedvéért minden egyes pontot bitsorozatként kezelünk. (Minden nem logikai attribútum kódolható logikai attribútumok halmazával.)

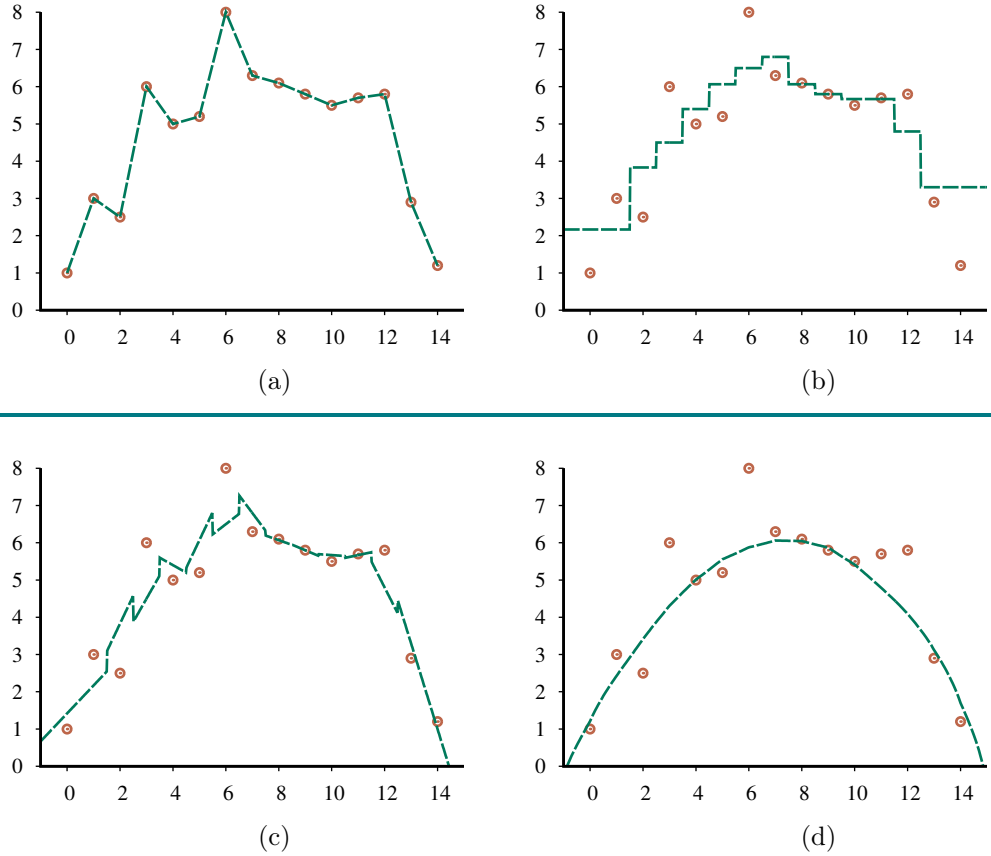
Arra az intuícióra támaszkodunk, hogy ha két pont közel van egymáshoz az $n$-dimenziós térben, akkor szükségszerűen közel lesznek, amikor levetítjük őket az egydimenziós térre (amely egy vonal). Valójában a vonalat diszkrét tartományokra – hash kosarakra – bonthatjuk úgy, hogy nagy valószínűséggel az egymáshoz közeli pontok ugyanabba a kosárba lesznek levetítve. Azoknak a pontoknak, amelyek távol vannak egymástól, a levetítése viszont valószínűleg különböző kosarakba fog kerülni, de mindig lesz néhány olyan vetítés, ami véletlenül egymástól távoli pontokat ugyanabba a kosárba vetít. Így a kosár, amelybe $\mathbf{x}_q$ kerül, sok olyan pontot fog tartalmazni (bár nem mindet), amelyek közel vannak $\mathbf{x}_q$-hoz, és tartalmazhat néhány olyan pontot, amelyek távol vannak tőle.

A LÉH trükkje, hogy *többszörös* véletlen vetítést hozunk létre, és kombináljuk őket. Egy véletlen vetítés nem más mint a reprezentáló bitsorozat egy véletlen részhalmaza. Kiválasztunk $\ell$ különböző véletlen vetítést, és létrehozunk $\ell$ hash táblát: $g_1(\mathbf{x}), \ldots, g_\ell(\mathbf{x})$. Aztán amikor egy $\mathbf{x}_q$ lekérdezési pontot vizsgálunk, akkor vesszük az $g_i(\mathbf{x}_q)$ kosárban található pontokat az összes hash táblára, és ennek az $\ell$ halmaznak

az uniója adja a $C$ jelöltpontokat. Ezek után kiszámítjuk a $C$ összes pontjára az $\mathbf{x}_q$-tól való távolságot, és vesszük a $k$ legközelebbit. Nagy valószínűséggel az összes olyan pont, amelyik közel van $\mathbf{x}_q$-hoz megjelenik legalább az egyik kosárban. Ugyan megjelenhet néhány távoli pont is, de ezeket figyelmen kívül tudjuk hagyni. nagy méretű valós problémákban, mint például közeli szomszédok keresése egy 13 millió – a webről származó 512-dimenziós – képből álló halmazban (Torralba *et al.*, 2008), a lokációérzékeny hash-eljárás a 13 millióból csak néhány ezer kép vizsgálatát igényli ahhoz, hogy a legközelebbi szomszédokat megtalálja – ez ezerszeres javulás a kimerítő kereséshez vagy a $k$-d fa megközelítéshez képest.

### 19.7.4. Nemparametrikus regresszió

Ebben a szakaszban inkább a *regresszió*val fogunk foglalkozni, nem az osztályozásssal. A 19.20. ábra néhány különböző modellre mutat példát. Az (a) ábrán talán a legegyszerűbb módszer látható, amelyet informálisan „kösd-össze-a-pontok" névvel illethetünk, vagy fölényeskedően „szakaszonként lineáris nemparametrikus regresszió"-nak.




**19.20. ábra** Nemparametrikus regressziós modellek: (a) a pontok összekötése, (b) a 3 legközelebbi szomszéd átlagolása, (c) lineáris regresszió a 3 legközelebbi szomszédra, (d) lokálisan súlyozott regresszió egy 10 szélességű kvadratikus kernellel.

Ez a modell egy olyan $h(x)$ függvényt hoz létre, amelynek egy $x_q$ lekérdezési pontot adva, az $x_q$ bal és jobb oldalán közvetlenül található tanítópontokat veszi, és interpolál köztük. Ha kicsi a zaj, akkor ez a triviális módszer valójában nem túl rossz – ezért lett a táblázatkezelők megjelenítő szoftverének egy standard eljárása. Viszont ha a zaj nagy, akkor a kapott függvény tüskés, és nem általánosít jól.

Legközelebbi szomszéd regresszió

A $k$-**legközelebbi szomszéd regresszió** a kösd-össze-a-pontok módszer javított változata. A helyett, hogy csupán az $x_q$ lekérdezési pont bal és jobb oldalára eső két példát használnánk, a $k$ legközelebbi szomszédot használjuk. (Itt most $k = 3$-at alkalmaztunk.) Nagyobb $k$ értékek hajlamosak lesimítani a csúcsok nagyságát, de az eredő függvénynek vannak szakadásai. A 19.20. ábra a $k$-legközelebbi-szomszéd regresszió két változatát mutatja. A (b) ábrán a $k$ legközelebbi szomszéd átlagát látjuk: $h(x)$ a $k$ pont $\sum y_j/k$ átlaga. Vegyük észre, hogy az $x = 0$ és $x = 14$ közelében lévő kilógó pontokat rosszul becsüli, mert csak az egyik oldalról kap adatokat (a szakasz belseje felől), és ezért figyelmen kívül hagyja a trendet. A (c) ábrán $k$-legközelebbi-szomszéd regressziót látjuk, amely a $k$ példára legjobban illeszkedő egyenest találja meg. Ez jobban felismeri a trendeket a kilógó adatok környezetében, de még szintén szakadásos. Mind a (b), mind (c) esetben felmerül a legjobb $k$ érték megválasztásának kérdése. Ahogy rendszerint, a válasz itt is a keresztvalidáció.

Lokálisan súlyozott regresszió

A 19.20. ábra (d) részén látható **lokálisan súlyozott regresszió** biztosítja a legközelebbi szomszédok nyújtotta előnyöket, a szakadások megjelenése nélkül. Ahhoz, hogy elkerüljük $h(x)$-nél a szakadásokat, el kell kerülnünk a szakadásokat a $h(x)$ becsléséhez felhasznált mintahalmazban. A lokálisan súlyozott regresszió alapgondolata, hogy minden $x_q$ lekérdezési pontnál a hozzá közeli mintapontoknak nagy súlyt adunk, a távolabbiaknak kisebbet, a legtávolabbiaknak pedig nulla lesz a súlya. A súlyok csökkenése a távolság növekedése mentén tipikusan fokozatos, nem ugrásszerű.

Kernel

Egy úgynevezett **kernelfüggvény** segítségével határozzuk meg az egyes példák súlyozását. Bemenete a lekérdezési pont és a példa közti távolság. A $\mathcal{K}$ kernelfüggvény a távolság csökkenő függvénye, maximuma a 0-nál van. Így $\mathcal{K}(Distance(\mathbf{x}_j, \mathbf{x}_q))$ nagyobb súlyt ad az $\mathbf{x}_q$ lekérdezési ponthoz – amelynek függvényértékét prediktálni akarjuk – közeli $\mathbf{x}_j$ példáknak. Az $\mathbf{x}$ teljes bemeneti terére a kernel integrálja véges kell hogy legyen – ha az integrálértéket 1-nek választjuk, akkor bizonyos számítások könnyebbek lesznek.

Egy $\mathcal{K}(d) = \max(0, 1 - (2|d|/w)^2)$ kvadratikus kernellel hoztuk létre a 19.20. ábra (d) részét, ahol a kernel szélessége $w = 10$ volt. Más alakú függvényeket is szoktak használni, mint például a Gauss-kernel. A szélesség tipikusan nagyobb jelentőségű, mint a konkrét jelalak: ez egy hiperparamétere a modellnek, értékét legjobb keresztvalidációval választani. Ha túl szélesre választjuk a kernelt, akkor alulilleszkedést tapasztalunk, ha túl keskenyre, akkor túlilleszkedést. A 19.20. ábra (d) részénél 10-es kernelszélesség egy sima görbét eredményez, ami úgy tűnik itt éppen megfelelő.

Ezek után már kézenfekvő a kernelfüggvénnyel megvalósított lokálisan súlyozott regresszió. Egy adott $\mathbf{x}_q$ lekérdezési pontra megoldjuk a következő súlyozott regresszió problémát:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(Distance(\mathbf{x}_q, \mathbf{x}_j)) \, (y_j - \mathbf{w} \cdot \mathbf{x}_j)^2 \,,$$

ahol *Distance* bármelyik – legközelebbi szomszéd módszereknél alkalmazott – távolságmetrika lehet. Utána az eredményt $h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$ adja.

Vegyük észre, hogy *mindegyik* lekérdezési pontra egy új regressziós problémát kell megoldanunk – ezt jelenti, hogy lokálisak vagyunk. (A szokásos lineáris regressziónál egyszer oldottuk meg a regressziós problémát globálisan, és utána ugyanazt a $h_{\mathbf{w}}$-t használtuk minden lekérdezési pontra.) Enyhíti ezt az extra munkát, hogy az egyes regressziós problémák könnyebben megoldhatók lesznek, mert csak a nem nulla súllyal rendelkező példákat kell figyelembe vennünk – a példákat, amelyek a lekérdezési ponttól a kernelfüggvény szélességébe esnek. Ha a kernel szélessége kicsi, akkor lehet, hogy csak kevés ilyen pont van.

A legtöbb nemparametrikus modellnek előnye, hogy könnyű a hagyj-ki-egyet keresztvalidációt elvégezni a nélkül, hogy mindent újra ki kéne számítanunk. Például $k$-legközelebbi-szomszéd modell esetén, ha van egy $(\mathbf{x}, y)$ tesztpéldánk, akkor vesszük a $k$ legközelebbi szomszédját egyszer, majd kiszámítjuk belőlük az $L(y, h(\mathbf{x}))$ mintára eső veszteséget. Ezt megjegyezzük, és ettől kezdve ezt használjuk, mint a hagyj-ki-egyet eredményét minden olyan példára, amelyik nem valamelyik szomszéd. Utána vesszük a $k+1$ legközelebbi szomszédot, és megjegyezzük sorban a $k$ különböző eredményt, amelyet az egyes szomszédok kihagyásával kapunk. $N$ példa esetén az egész eljárás $O(k)$, nem $O(kN)$.

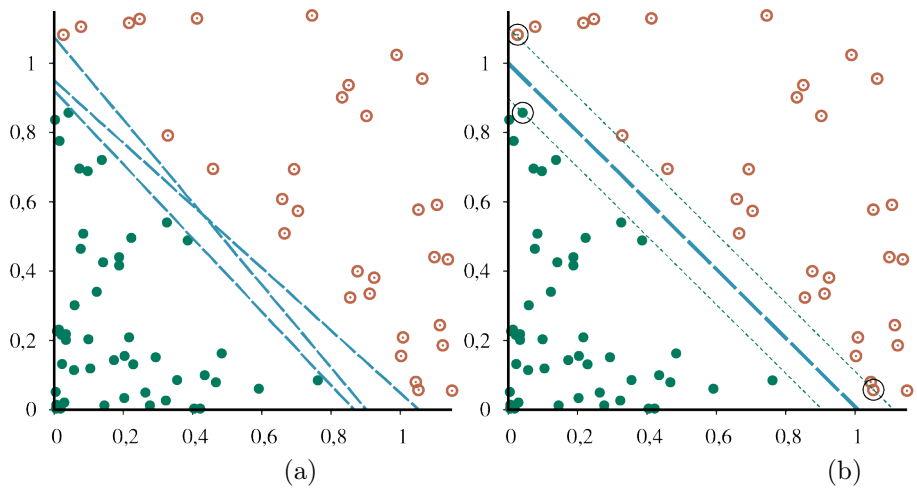
### 19.7.5. Szupport vektor gépek

Szupport vektor gép (SVM)

A korai 2000-es években a **szupport vektor gép (SVM)** modellosztály volt a legnépszerűbb megközelítés a „nem testreszabott” („off-the-shelf”) felügyelt tanulásra, amikor is nem rendelkeztünk a területre vonatkozó *a priori* tudással. Ezt a pozíciót mára átvették a mélytanuló hálók és a véletlen erdők, de az SVM megőrzött három vonzó tulajdonságot:

1. Az SVM-ek egy **maximális margójú elválasztás**t hoznak létre – egy döntési határt, amelyik a lehető legnagyobb távolságban van a példapontoktól. Ez segíti őket a jó általánosító képesség elérésében.
2. Az SVM-ek *lineáris* elválasztó hipersíkot hoznak létre, de képesek az adatokat magasabb dimenziós térbe ágyazni, az úgynevezett **kerneltrükk** alkalmazásával. Gyakran olyan adatok, amelyek az eredeti bemeneti térben nem voltak lineárisan szeparálhatók, lineárisan szeparálhatókká válnak a magasabb dimenziós térben.
3. Az SVM-ek nemparametrikusak, a szeparáló hipersíkot a példapontok egy halmaza határozza meg, nem valamilyen paraméterérték-gyűjtemény. Viszont míg a legközelebbi-szomszéd modelleknek az összes példát meg kell őrizniük, az SVM modell csak azokat a példákat tartja meg, amelyek legközelebb vannak az elválasztó hipersíktól – rendszerint csak egy kis konsstansszorosát a dimenziószámnak. Így az SVM-ek a nemparametrikus és parametrikus modellek előnyeit egyesítik: rendelkeznek a komplex függvények reprezentációjához szükséges flexibilitással, de ellenállnak a túlilleszkedésnek.

A 19.21. ábra (a) részén egy kétosztályos osztályozási példát látunk három jelölttel a döntési határra, mindegyikük lineáris szeparátor. Mindegyik konzisztens az összes példával, így a 0/1 veszteség szempontjából egyformán jók. A logisztikus regresszió




**19.21. ábra** Szupport vektor gép osztályozás: (a) Két osztályba sorolható pontok (üres narancsszínű és kitöltött zöld körök) és három jelölt a lineáris szeparálásra. (b) A maximális margójú szeparátor (vastag vonal), a **margó** (a vékony szaggatott vonalak által határolt terület) közepén fut. A **szupport vektorok** (erős fekete vonallal bekarikázva) a szeparátorhoz legközelebbi példák; itt három ilyen van.

valamilyen szeparáló egyenest fog találni, a vonal pontos helyzete függ az *összes* példaponttól. Az SVM-ben felhasznált alapvető meglátás, hogy bizonyos példák fontosabbak másoknál, és ha ezekre figyelünk, akkor jobb általánosító képességet érhetünk el.

Nézzük az (a) ábrán a legalsót a három szeparáló egyenes közül. Nagyon közel megy öt zölddel jelölt példához. Bár az összes példát helyesen osztályozza, de zavaró kell hogy legyen, hogy olyan sok példa van közel a határvonalhoz; ugyanis előfordulhat, hogy más – a zölddel jelölt osztályba tartozó – példák a vonal rossz oldalára esnek.

Az SVM foglalkozik ezzel a helyzettel: a helyett, hogy a tanítóadatokon várható *tapasztalati veszteséget* minimalizálná, az SVM megkísérli a várható *általánosítási* veszteséget minimalizálni. Ugyan nem tudjuk, hogy a még-nem-látott pontok hova fognak esni, de a valószínűségi feltevésünk alapján (hogy ugyanabból az eloszlásból származnak majd, mint a korábban látott példák) a számítógépes tanulás elmélete (a 19.5. alfejezet) ad néhány érvet arra, hogy az általánosítási hibát minimalizáljuk, ha azt a szeparáló egyenest választjuk, amelyik az eddig látott példáktól a lehető legmesszebb helyezkedik el. Ezt a szeparátort, amelyet a 19.21. ábra (b) részén látunk, **maximális margójú szeparátor**nak nevezzük. A **margó** a vékony szaggatott vonalak közti terület szélessége – kétszerese a szeparátor és a legközelebbi mintapont távolságának.

Maximális margójú szeparátor
Margó

Akkor hogyan találjuk meg ezt a szeparátort? Mielőtt az egyenleteket néznénk, beszéljünk a jelölésekről: hagyományosan az SVM-ek azt a konvenciót követik, hogy az osztályok címkéi +1 és -1, nem pedig +1 és 0, ahogy az eddigiekben jelöltük. Hasonlóképpen, míg mi korábban az eltolást belefoglaltuk a $\mathbf{w}$ súlyvektorba (bevezetve egy megfelelő fiktív $x_{j,0} = 1$ értéket), az SVM-eknél nem csinálják ezt; az eltolást megtartják egy külön $b$ paraméterként.

### 22.7.1. Játékokban való alkalmazás

Az 1. fejezetben bemutattuk Arthur Samuel a dámajáték megtanulására irányuló korai megerősítéses tanulási munkáját, amelyet 1952-ben kezdett. Eltelt néhány évtized, mire az ilyen kihívásoknak újra nekiláttak, ekkor Gerry Tesauro az ostáblajátékon dolgozott. Tesauro első kísérlete (1990) a Neurogammon program volt. Megközelítése az imitációs tanulás egy érdekes változata volt. A bemenet egy 400 játékból álló halmaz volt, amelyeket Tesauro önmaga ellen játszott. A Neurogammon nem egy eljárásmódot tanult, inkább minden $(s, a, s')$ lépést tanítóminták egy olyan halmazává alakított, amelyek mind úgy címkézték $s'$-t, mint egy $s$-ből más lépéssel elérhető $s''$-nél jobb pozíciót. A hálónak két elkülönült fele volt, egy $s'$ és egy $s''$ részére, és arra volt korlátozva, hogy a két fél kimeneteit összehasonlítva a jobbat válassza közülük. Ezzel a módszerrel mindkét felet arra kényszerítette, hogy egy $\hat{U}_\theta$ értékelőfüggvényt tanuljon meg. A Neurogammon megnyerte az 1989. évi Computer Olimpiát – az első tanulóprogram volt, amelyik valaha is nyert számítógépes játékversenyen –, de soha nem tudta meghaladni Tesauro (közepes) játéktudását.

Tesauro következő rendszere – TD-Gammon (1992) – Sutton frissen publikált MT-módszerét alkalmazta. Lényegében visszatért a Samuel által felfedezett megközelítéshez, de sokkal mélyebben megértve azt, hogy hogyan kell jól alkalmazni. Az $\hat{U}_\theta$ értékelőfüggvényt egy egyetlen rejtett rétegű, teljesen összekötött neuronháló reprezentálta, amelynek a rejtett rétegében 80 neuron volt. (Felhasznált néhány, a Neurogammonból átvett bemeneti tulajdonságot.) 300 000 tanítójáték után olyan szintet ért el, mint a világ három legjobb humán játékosa. A legjobb tízbe tartozó Kit Woolsey azt mondta: „Nem kérdéses számomra, hogy a pozíciók megítélésében messze jobb nálam.”

Mély Q-hálózat (MQH)

A következő kihívás az volt, hogy hogyan lehet tanulni nem játéktáblán adott diszkrét reprezentációkból, hanem a való élethez közelebb álló nyers bemeneti észlelésekből. 2012-ben a DeepMindnál egy csapat elkezdett egy **mély Q-hálózat**ot **(MQH)** fejleszteni, ez volt az első modern mély MT-rendszer. Az MQH egy mély neuronhálót alkalmaz a Q-függvény reprezentálására, ettól eltekintve tipikus megerősítéses tanulási rendszer. Az MQH-át külön-külön mind a 49 különböző Atari videójátékra megtanították. Megtanult szimulált autókat vezetni, ellenséges űrhajókat lelőni és ütővel labdát pattogtatni. Minden egyes esetben az ágens a nyers képekből tanult meg egy Q-függvényt, a jutalom a játékban elért pontszám volt. Összességében a rendszer nagyjából az ember szakértői szintjén teljesített, bár néhány játék gondot okozott neki. Különösen az egyik játék, a *Montezuma's Revenge* bizonyult messze túl nehéznek a számára, mivel ehhez kiterjedt tervezési stratégiákra volt szükség, és a jutalmak túl ritkán érkeztek. A további munka során olyan mély MT-rendszereket hoztak létre, amelyek sokkal szélesebb körben mutattak felfedezési viselkedést, és képesek voltak legyőzni a *Montezuma's Revenge*-et, valamint más nehéz játékokat.

A DeepMind által fejlesztett AlphaGo is megerősítéses tanulást használt arra, hogy legyőzze a gó játék legjobb humán játékosait is (lásd az 5. fejezetben). Míg az Atari játékoknál elég egy előrelátással nem rendelkező Q-függvény (mert ezek a játékok alapvetően reaktív jellegűek), a gó alapos előretekintést igényel. A Q-függvény, amelyet egy konvolúciós neurális háló valósít meg, önmagában elég pontos ahhoz, hogy a legtöbb amatőr játékost legyőzze, anélkül hogy bármiféle keresést alkalmazna.

### 22.7.2. Robotszabályozási alkalmazások

Rúdegyensúlyozás
Inverz inga

A 22.9. ábra (a) részén látható a híres **rúdegyensúlyozás**i vagy más néven **inverz inga** probléma. A feladat a kocsira gyakorolt jobbra vagy balra irányuló erővel a rudat nagyjából felfele ($\theta \approx 90°$) tartani, miközben a kocsi $x$ pozíciója a megengedett határok között marad. Sok ezer megerősítéses tanulással foglalkozó, valamint szabályozáselméleti publikációt írtak erről a látszólag egyszerű problémáról. Az egyik nehézség, hogy míg az állapotváltozók ($x$, $\theta$, $\dot{x}$ és $\dot{\theta}$) folytonosak, ugyanakkor a cselekvéseket diszkrétnek definiáljuk: lökés balra vagy jobbra; ez az úgynevezett **bang-bang szabályozás**.

Bang-bang szabályozás

Michie and Chambers (1968) munkája volt a legelső, amely ennek a feladatnak a tanulásával foglalkozott, ők egy valódi kocsit és rudat használtak, nem szimulációt. 30 kísérlet után a BOXES nevű algoritmusuk képes volt egy órán keresztül egyensúlyozni a rudat. Az algoritmus először diszkrét dobozokra bontotta a négydimenziós állapotteret – innen kapta a nevét. Utána addig futtatta a kísérleteket, amíg a rúd leesett. Negatív megerősítést asszociáltak a legutolsó dobozhoz, és utána ezt terjesztették vissza az egész sorozatra. Gyorsabb tanulást és jobb általánosító képességet lehet elérni, ha egy algoritmussal a jutalmak változását megfigyelve *adaptívan* osztjuk részekre az állapotteret, vagy folytonos állapotváltozók nemlineáris függvényapproximátorát használjuk, ilyen lehet például egy neurális hálózat. Manapság a *háromszoros* inverz inga (három rúd, a végeiknél csuklóval összekapcsolva) egy közönséges feladat – ez a mutatvány messze meghaladja a legtöbb ember képességeit, de megerősítéses tanulással megtanulható.


(a) (b)

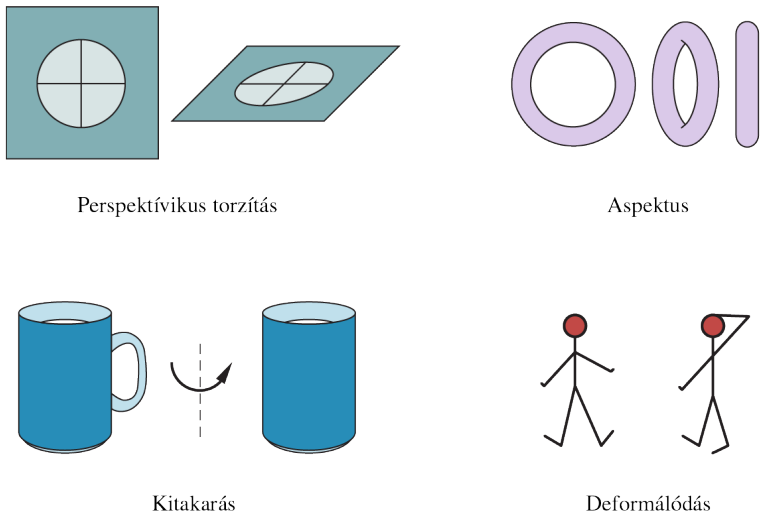
**22.9. ábra** (a) Egy mozgó kocsi tetején egy hosszú rúd egyensúlyozására szolgáló berendezés. A szabályzó jobbra vagy balra lökheti a kocsit megfigyelve a kocsi $x$ pozícióját, $\dot{x}$ sebességét, továbbá a rúd $\theta$ szögét és $\dot{\theta}$ szögsebességét. (b) Hat, időközönként felvett és egymásramásolt kép, amely egy önvezető helikoptert mutat a rendkívül nehéz „tölcsér” manőver (oldalirányú repülés úgy, hogy a gép orra mindig egy központi cél felé nézzen) végrehajtása közben. A helikoptert egy, a PEGASUS eljáráskereső algoritmussal fejlesztett eljárásmód vezérelte (Ng *et al.*, 2003). Egy szimulátormodellt fejlesztettek ki a valódi helikopternek a különböző vezérlési beavatkozásokra adott válaszainak megfigyelésére alapozva. Ezek után egész éjszaka futtatták a modellt. Sokféle szabályozót fejlesztettek ki különböző manőverekre. Az összes esetben az eredmény messze meghaladta a távirányítást használó szakértő humán pilóták teljesítményét. (A képet Andrew Ng engedélyével közöljük.)

ismereteket is arról, hogy milyen tárgyak várhatók egy jelenetben. Mostanában népszerű stratégia egy kép túlszegmentációjának előállítása, ami garantáltan nem hagyja el a valódi határok megjelölését, de valószínűleg számos hamis határszakaszt is megjelöl. Az így kapott, szuperképpontoknak (superpixel) nevezett régiók jelentősen csökkentik a különböző algoritmusok számítási komplexitását, mivel a szuperképpontok száma százas nagyságrendű lehet, összehasonlítva a több millió nyers képponttal. A tárgyak magas szintű ismereteinek kiaknázását a következő szakasz tárgyalja, a képeken lévő objektumok tényleges észlelése pedig a 25.5. alfejezet tárgya.

## 25.4. Képek osztályozása

A képek osztályozása két fő esetre bontható. Az egyikben a képek *tárgyakból* állnak össze, melyek osztályok egy adott taxonómiájából következnek, és melyek a képen más jelentőséggel nem bírnak – például egy ruházati vagy bútorkatalógus, ahol a háttérnek nincs jelentősége, és az osztályozó kimenete: „kasmírpulóver” vagy „irodai szék”.

A másik esetben minden kép több objektumot tartalmazó *jelenetet* mutat. Tehát a szavannán láthatunk zsiráfot és oroszlánt, a nappaliban pedig kanapét és lámpát, de a nappaliban nem számítunk zsiráfra vagy tengeralattjáróra. Ma már vannak olyan módszerek, melyek képesek felismerni, hogy egy kép által mutatott jelenet a „szavannán” vagy a „nappaliban” került rögzítésre.




**25.11. ábra** A megjelenés változásának fontos forrásai, amelyek miatt ugyanazon objektum különböző képei eltérőek lehetnek. Először is, az elemek perspektívából eredően torzulhatnak, mint például a bal felső sarokban lévő kör alakú terület. Ezt a területet olyan betekintési szögben látjuk, ami miatt az ábrán ellipszis alakú a képe. Másodszor, a különféle irányokból látszó tárgyaknak drámai módon megváltozhat az alakjuk, ezt a jelenséget nézőpontnak (aspektusnak) nevezzük. A jobb felső sarokban a fánk három különféle aspektusa látható. A kitakarás (okklúzió) miatt – a bal alsó sarokban – a bögre forgatásakor a bögre füle eltűnik. Ebben az esetben, mivel a test és a fogantyú ugyanahhoz a bögréhez tartoznak, önkitakarásról beszélhetünk. Végül, a jobb alsó sarokban látszik, ahogy néhány objektum jelentősen deformálódik.

Megjelenés

A modern rendszerek a képeken látható objektumokat **megjelenés**ük (azaz a színük és a textúrájuk) alapján osztályozzák, nem pedig a geometria alapján. Ennél a megközelítésnél két nehézség adódik. Először is, ugyanazon osztály különböző példányai eltérő megjelenésűek lehetnek – néhány macska fekete, mások pedig narancssárgák. Másodszor, ugyanaz a macska különböző hatásoktól függően eltérően nézhet ki különböző időpontokban (ahogy ez a 25.11. ábrán látható):

- **Világítás**, amely megváltoztatja a kép fényerejét és színhőmérsékletét.
- **Perspektívából eredő torzulás**, amely a betekintési szög függvényében torzítja a látott alakzat képét.
- **Nézőpont**, ami miatt az objektumok eltérőnek tűnnek, ha különböző irányokból nézik őket. Például oldalról nézve egy fánk úgy néz ki, mint egy sík ovális, de felülről inkább egy gyűrűre hasonlít.
- **Kitakarás**, ahol az objektum egyes részei el vannak rejtve. Az objektumok kitakarhatják egymást, vagy az objektumok egyes részei eltakarhatják a többit is, mely jelenséget **önkitakarás**nak nevezzük.
- **Deformáció**, amikor az objektum megváltoztatja az alakját. Például a teniszező mozgatja a karját és a lábát.

Modern módszerek ezeket a problémákat nagyon sok mintán tanított reprezentációk és osztályozók alkalmazásával kezelik, melyek általában konvolúciós neurális hálózatokra épülnek. Az osztályozó kellően gazdag tanítókészlet mellett sokszor fogja látni ezeket a hatásokat a tanulás során, így alkalmazkodni tud hozzájuk.

### 25.4.1. Képosztályozás konvolúciós neurális hálózatokkal

A **konvolúciós neurális hálózatok** (CNN-ek) látványos sikereket érnek el a képosztályozási feladatokban. Elegendő tanítóadattal és elegendő találékonysággal a CNN-ek nagyon sikeres képosztályozási és objektumdetektálási megoldások gerincét képezik, sokkal jobbakat, mint amit eddig egyéb módszerek alkalmazásával kaphatunk.

Az ImageNet adatkészlet történelmi szerepet játszott a képosztályozó rendszerek fejlesztésében, mivel több mint 14 millió képet biztosított konstrukciójukhoz, azokat több mint 30 000 részletesen megválasztott kategóriába sorolva. Az ImageNet továbbá ösztönözte a fejlődést egy évenkénti verseny meghirdetésével is. A versenyre benyújtott megoldásokat mind a legvalószínűbb osztályba történő (single best guess) besorolás pontossága, mind az öt legvalószínűbb osztályba történő besorolás (top 5) pontossága alapján értékelik, ez utóbbiban a rendszerek öt kimeneti címkét rendelhetnek hozzá egy bemenethez – *például malamut, husky, akita, szamojéd, eszkimó kutya.* Az ImageNetnek 189 *kutya*-alkategóriája van, így még a kutyaszerető embereknek is nehéz helyesen felcímkézni a képeket.

A 2010-es első ImageNet versenyen a rendszerek nem voltak képesek 70%-osnál jobb pontosságra a top 5 találat kategóriában. A konvolúciós neurális hálózatok bevezetése 2012-ben és azok későbbi finomítása 98%-os pontossághoz vezetett a top 5-ben (meghaladva az emberi teljesítményt), és 2019-re 87%-os pontosságot a top 1 találatmetrikák szerint. E siker elsődleges oka úgy tűnik, hogy a CNN-osztályozók által használt belső jellemzők és reprezentációk a mintákhoz adaptáltak, azaz az adatok

tanulásából származnak, nem pedig kutató által kézzel készítettek. Ez biztosítja, hogy a jellemzők és reprezentációk valóban hasznosak is az osztályozásban.

A képosztályozásában azért történt gyors előrelépés, mert rendelkezésre állnak nagy, kihívást jelentő adatkészletek, mint például az ImageNet; az ezen adatkészleteken alapuló, tisztességes és nyílt versenyek, valamint a sikeres modellek publikációi is. A versenyek nyertesei közzéteszik a kódot és gyakran a modellek előretanított paramétereit is, megkönnyítve mások számára a sikeres architektúrákkal való kísérletezést és azok további javítását.

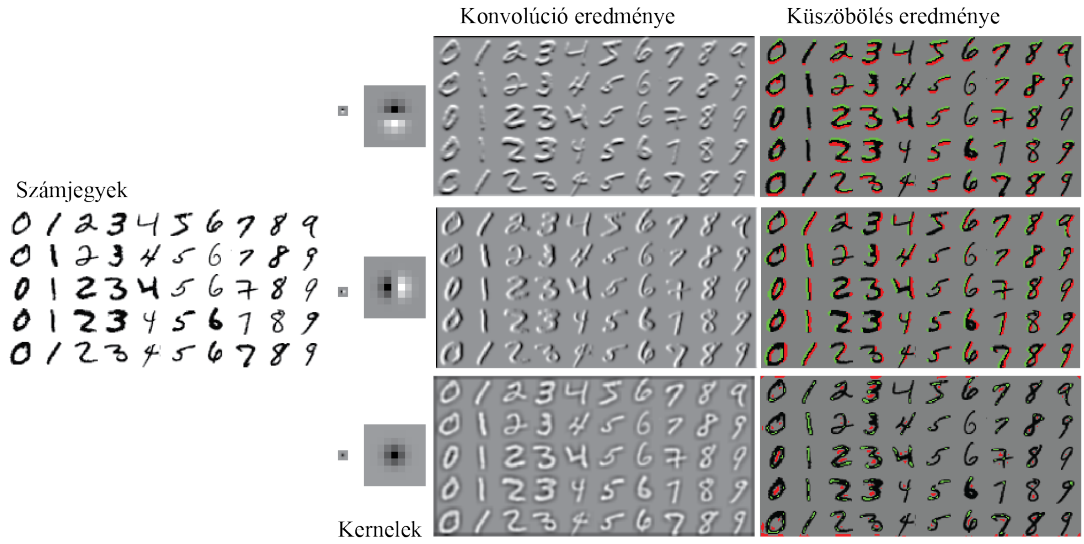
### 25.4.2. A konvolúciós neurális hálózatok jó képosztályozók

A képosztályozás legjobban az adatkészletek vizsgálatával érthető meg, de az ImageNet túl nagy ahhoz, hogy részletesebben megvizsgáljuk. Az MNIST adatkészlet 70 000 kézzel írott számjegy (0–9) képének gyűjteménye, amelyet gyakran bemelegítési adatkészletként használnak. Ezt az adatkészletet megnézve (néhány példa a 25.12. ábrán található), kiderül néhány fontos, meglehetősen általános tulajdonság. Készíthetünk egy számjegyről képet, amin aztán számos apró változtatást végezhetünk anélkül, hogy a számjegy típusa megváltozna: elmozgathatjuk, elforgathatjuk, világosabbá vagy sötétebbé, kisebbé vagy nagyobbá tehetjük. Ez azt jelenti, hogy az egyes képpontok értékei nem különösebben informatívak – tudjuk, hogy a 8-asnak valamilyen sötét képponttal kell rendelkeznie a közepén, és a 0-nak ilyen ponttal nem kell rendelkeznie, de ezeknek a sötét képpontoknak kissé eltérő helyük lesz a 8-asok különböző példányainál.

A képek másik fontos tulajdonsága, hogy kisebb részmintázatok meglehetősen informatívak lehetnek: A 0, 6, 8 és 9 számjegyek hurkokkal rendelkeznek; a 4 és a 8 számjegyek egymást keresztező görbékkel; az 1, 2, 3, 5 és 7 számjegyek vonalvégződéssel rendelkeznek, de esetükben nincs sem hurok, sem keresztezés; a 6 és 9 számjegy hurkokat és vonalvégződéseket tartalmaz. Ezen felül a helyi minták térbeli kapcsolatai is informatívak. Egy 1-es két vonalvégződéssel rendelkezik egymás felett; egy 6-os egy vonallal végződik egy hurok fölött. Ezek a megfigyelések egy olyan stratégiát sugallnak, amely a modern számítógépes látás központi eleme: hierarchikusan felépülő, eltérő absztrakciójú mintázatfelismerők eredményeit bemenetként kezelő, egyszerű mintázatkeresők alapján komplex alakzatfelismerési feladatok is megvalósíthatók.

Ez az, amit a konvolúciós neurális hálózatok igazán jól csinálnak. Egy rétegre – egy konvolúcióra, amelyet egy ReLU aktivációs függvény követ – mint egy helyi mintázatazonosítóra (lásd a 25.12. ábrát) érdemes gondolnunk. A konvolúció azt méri, hogy a kép egyes, csúszóablakban kivágott részletei mennyire hasonlítanak a kernelhez; a ReLU a kernellel negatívan korreláló ablakokra a kimenetet nullára állítja, míg a többi esetén nem módosít. Több kernellel történő szűrés több mintát talál. Az összetett minták pedig egymást követő ilyen szűrésekből álló rétegek sorozataként realizálhatók.

Vizsgáljuk meg az első konvolúciós réteg kimenetét. Mindegyik kimeneti pixel az adott pixel körüli ablak képpontjaitól függ. A ReLU kimenete (a rétegben alkalmazott szűrés után), amint láttuk, egy egyszerű mintakeresőt képez. Amennyiben erre illesztünk egy második réteget, akkor a második réteg minden egyes pixele az első réteg adott pixele körüli ablak értékeitől függ. Ez azt jelenti, hogy a második rétegbeli pixeleket az első réteg bemenetén lévő kép egy nagyobb ablaka befolyásolja, mint az első réteg esetén. Ezekre úgy kell tekintenünk, mint amelyek „minták mintáit” emelik ki. A




**25.12. ábra** A bal oldalon láthatunk néhány képet a MNIST adatkészletből. A két középső oszlopból a bal oldalinál három kernel jelenik meg. Ezeket egyrészt tényleges méretükben látjuk (apró blokkok), és nagyítva is, hogy azok tartalmát megismerhessük: a középszürke a nulla, a világos pozitív, a sötét pedig negatív értéknek felel meg. A két középső oszlopból a jobb oldalinál megjelenik annak az eredménye, ha a kernelekkel a képet megszűrjük. A jobb oldali oszlop képpontokat mutat, ahol a válasz nagyobb, mint egy pozitív küszöb (zölddel jelölve), vagy kisebb, mint egy negatív küszöb (pirossal). Esetünkben ezek a szűrések (felülről lefelé): vízszintes sávokat; függőleges sávokat; és (nehezebb észrevenni) vonalvégeket érzékelnek. Ezek az szűrések érzékenyek a sáv két széle közötti intenzitásváltozás előjelére is, így (például) egy vízszintes sáv, amely a felső részén világos és alul sötét, pozitív (zöld) választ ad, míg egy másik, amely a felső részén sötét és az alsó részén világos, negatív (piros) választ ad. Ezek a szűrések hatékonyak, de nem tökéletesek.

hálózat több szinten keres mintákat, és ezt úgy teszi, hogy a mintakeresésre használt szűrések kernelét az adatokból *tanulja*, és nem a programozó által előzetesen megadott kerneleket használ.

Noha a CNN „alapértelmezett változatának” tanítása eredményes, néhány további gyakorlati technika megismerése hasznos lehet. Az egyik legfontosabb az **adatkészlet bővítése**, amelyben az tanítási példákat lemásoljuk és valamelyest módosítjuk. Például véletlenszerűen eltolhatjuk, elforgathatjuk, vagy megnyújthatjuk a képet, vagy véletlenszerűen eltolhatjuk a képpontok árnyalatát kis mértékben. Az adatkészletnek a nézőpontbeli vagy a megvilágításbeli szimulált változtatása növeli az adatkészlet méretét, bár az új példák természetesen erősen korrelálnak az eredetikkel. Szintén lehetséges a tesztelés közbeni képkészlet-kibővítés, ahelyett hogy a tanítási időben tennénk ezt. Ebben a megközelítésben a képet többször is lemásoljuk és módosítjuk (például véletlenszerű kivágással), majd az osztályozót minden módosított képen lefuttatjuk. Az egyes így előálló bemenetekre adott kimeneteket ezután aggregáljuk, és továbbítjuk a végső osztályozónak, amely – ezeket mintegy szavazatként felhasználva – dönt az osztálybatartozásról.

Adatkészlet-bővítés

Jelenetek képeinek osztályozásakor minden képpont hasznos lehet. Objektumok képeinek osztályozásakor azonban egyes képpontok nem feltétlenül képezik részét az objektumnak, és ezek elterelhetik a figyelemet. Például ha egy macska egy kutyaágyon fekszik, azt szeretnénk, hogy az osztályozó a macska képpontjaira összpontosítson, ne pedig az ágyra. A modern képosztályozók ezt jól kezelik, és pontosan osztályozzák a képet „macská”-nak, még akkor is, ha csak néhány képpont van a macskáról. Ennek két oka van. Először is, a CNN-alapú osztályozók ügyesen figyelmen kívül hagyják a kimeneti osztályozás szerint nem megkülönböztető képi mintákat. Másrészt, a tárgyon kívüli minták lehetnek megkülönböztetőek (például egy macskajáték, egy nyakörv, kis csengővel, vagy egy tál macskaeledel valójában segíthet megmondani, hogy egy macskát látunk). Ezt a hatást **kontextus**nak nevezzük. A kontextus segíthet vagy hátráltathat, ez erősen függ az adott adatkészlettől és az alkalmazástól.

Kontextus

## 25.5. Objektumdetekció

A képosztályozók becslést adnak arra, hogy *mit* ábrázol a kép – a teljes képet egy osztályba sorolják. Az objektumkeresők több objektumot azonosítanak egy képen, jelezve, hogy az egyes objektumok *melyik osztályba* tartoznak, és megadják azt is, hogy az egyes vélelmezett objektumok a képen *hol* vannak (az objektum köré egy **befoglaló keret**et[1] rajzolva). Az osztályok halmaza előre rögzített.

Befoglaló keret

Objektumkeresőt úgy építhetünk, hogy egy kis **csúszó ablak**on át (egy téglalapon keresztül) nézünk a nagyobb képre. Minden egyes ablak által kivágott képet egy CNN-osztályozóval osztályozzuk. Ezután összegyűjtjük a magas pontszámokat kapó osztályokat, és figyelmen kívül hagyjuk a többi ablakot. A megmaradó konfliktusok feloldását elvégezve egy végső objektumhalmaz marad a kezünkben, a felismert objektumok helyeivel. Azonban van még néhány részlet, amelyet ki kell dolgozni:

Csúszó ablak

- **Döntsünk az ablak alakjáról:** Messze a legegyszerűbb választás a tengelyekkel párhuzamos oldalú téglalapok használata. (Ettől eltérő formájú maszkot, amely kivágja az objektumot a képről, szinte soha sem használnak, mert nehéz megjeleníteni és számolni vele.) Viszont még meg kell választanunk a téglalapok szélességét és magasságát.
- **Készítsünk osztályozót az ablakokhoz:** Azt már tudjuk, hogyan kell ezt csinálni egy CNN-nel.
- **Döntsük el, mely ablakokat nézzük:** Az összes lehetséges ablakból ki akarjuk választani azokat, amelyekben valószínűleg érdekes alakzat található.
- **Válasszuk ki, mely ablakokat jelezzük:** Az ablakok átfedésben lesznek, és nem akarjuk, hogy ugyanazt az objektumot többször jelezzük kissé eltérő ablakokban. Néhány tárgyat nem érdemes megemlíteni; gondoljunk a székek és az emberek számára egy nagy tömött előadóterem képén. Mindegyiket különálló objektumként kell jelezni? Szinte sosem, talán csak azokat a tárgyakat kell külön kiemelni, amelyek a képen nagynak látszanak (például az első sorban lévő székek és emberek). A választás az objektumkereső tervezett felhasználásától függ.

[1] A „keret” kifejezést a kép tengelyeivel párhuzamos oldalú téglalap alakú régiójaként használjuk, és az „ablak” kifejezést elsősorban a „keret” szinonimájaként értjük, de az ablak a bemeneti kép egy felismerendő objektumát lokalizálja, melyekhez a kimeneti képen egy befoglaló keret tartozik.

- **Jelöljük az objektumok pontos helyét ezeknek az ablakoknak a segítségével:** Ha egyszer tudjuk, hogy az objektum valahol az ablakban van, megengedhetjük magunknak, hogy több számítást végezzünk, hogy pontosítsuk az ablakon belüli helyetét.

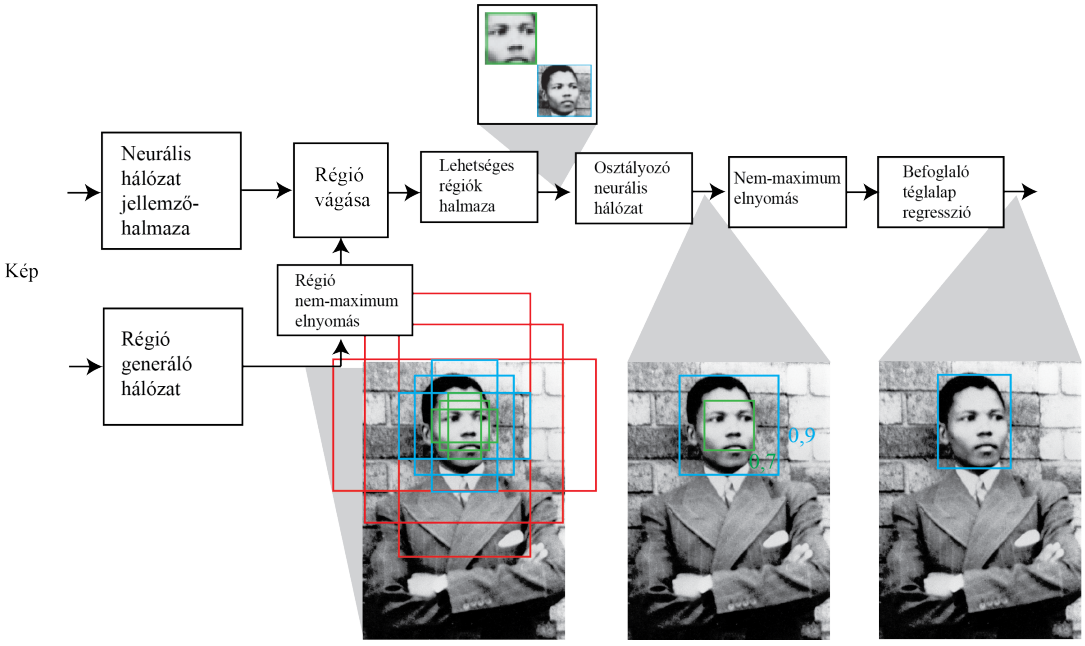
Nézzük meg alaposabban az ablak megválasztásának döntési problémáját. Az összes lehetséges ablak keresése nem hatékony – egy $n \times n$ képpontot tartalmazó képben $O(n^4)$ lehetséges téglalap alakú ablak van. Viszont tudjuk, hogy az objektumokat tartalmazó ablakok színei és textúrái általában eltérnek a többitől. Másrészt azokban az ablakokban, amelyek egy tárgyat félbevágnak, olyan régiók vagy élek vannak, amelyek az ablak oldalát metszik. Ezért értelme van egy olyan mechanizmusnak, amely pontozza az „objektumszerűséget" – minősíti, hogy a vizsgált téglalapban van-e egy objektum, függetlenül attól, hogy mi ez az objektum. Megtalálhatjuk azokat a téglalapokat, amelyek úgy néznek ki, mintha egy objektum lenne bennük, majd a lehetséges objektumokat csak azokon a téglalapokon kategorizáljuk, amelyek átjutnak az objektumszerűségi teszten.

Készítsünk egy háromdimenziós tömböt, ahol a tömb elemeit az ablak középpontja (két koordinátával) és az alakja indexeli. Minden olyan téglalapot, amelynek elég magas az objektumszerűségét minősítő pontszáma, **vizsgálandó régió**nak (region of interest, ROI) nevezzük, és egy osztályozónak kell ellenőriznie. A CNN-osztályozók azonban a rögzített méretű (felbontású) képeket várnak, míg az objektumszerűség-teszten átmenő téglalapok méretüket és alakjaikat tekintve különbözőek. Nem méretezhetjük a téglalapokat azonos méretűvé (csak geometriai torzítások árán, ami degradálná az osztályozás pontosságát), de azonos számú jellemzővel leírhatjuk ezeket a régiókat. Amennyiben az osztályozó bemenetére adott jellemző térkép pontjainak mintavételezésével nyerjük ki ezeket jellemzőket, a folyamatot **ROI-pooling**nak nevezzük. Ezt a rögzített méretű jellemzőtérképet ezután továbbítják az osztályozónak.

Ami a jelzendő ablakok eldöntésének problémáját illeti. Tegyük fel, hogy a $32 \times 32$ méretű ablakokat 1 képpontnyi léptetéssel nézzük: minden ablakot csak egy képpontnyi eltolás különböztet meg az előzőtől. Számos ablak lesz hasonló, és ezeknek hasonló pontszámokkal kell rendelkezniük. Ha mindegyiküknél a pontszám meghaladja a küszöböt, akkor sem akarjuk mindet jelezni, mert nagyon valószínű, hogy mind ugyanazon objektum képének kissé eltérő kivágásai. Másrészt ha a lépés túl nagy, akkor előfordulhat, hogy az objektumot egyetlen ablakban sem fedi, ezért elveszítjük. Ehelyett egy mohó algoritmust használhatunk, amelyet **maximumőrző ritkítás**nak hívunk. Először készítsünk egy rendezett listát az összes ablakról, amelynek pontszáma meghaladja a küszöböt. Ezután, amíg a listában vannak ablakok, válasszuk ki a legmagasabb pontszámmal rendelkező ablakot, és tekintsük úgy, hogy az tartalmaz egy továbbiakban vizsgálandó objektumot, majd töröljük a listáról az összes vele nagymértékben átfedő ablakot.

Maximumőrző ritkítás

Végezetül az objektumok pontos lokalizációjának problémája maradt. Tegyük fel, hogy van egy olyan ablakunk, amelynek magas a pontszáma, és amely a maximumőrző ritkítás algoritmusán is átment. Ez az ablak nem valószínű, hogy pontosan a megfelelő helyen lesz (emlékezzünk, általában viszonylag kis számú ablakot nézünk meg, kis számú lehetséges mérettel). Az osztályozó által kiszámított jellemzőreprezentációt




**25.13. ábra** A Faster RCNN két hálózatot használ. Egy fiatalkori Nelson Mandela képet adunk az objektumkeresőnek. Az egyik hálózat kiszámítja az „objektumszerűség" pontszámait a vizsgált képrégiókra (úgynevezett „horgonyrégiókra"), amelyek középpontjai egy rács pontjai. Minden rácsponton kilenc horgonyrégió található (három különböző területtel és oldalaránnyal). A példaképen egy zölddel jelölt belső és egy kékkel jelölt külső régió felelt meg az objektumszerűséget vizsgáló tesztnek. A második hálózat számolja ki a kép osztályozásra alkalmas reprezentációját. A legmagasabb objektumszerűségi pontszámmal rendelkező téglalapokat kivágják a jellemzőtérképről, méretüket ROI-poolinggal egységesítik, és továbbítják egy osztályozónak. A kék doboz magasabb pontszámot kap, mint a zöld doboz, és átfedésben van, tehát a zöld dobozt a maximumőrző ritkítás algoritmusával szűrjük ki. Végül a befoglalótéglalap-regressziót alkalmazzák a kék dobozon úgy, hogy az a lehető legpontosabban illeszkedjen az archoz. Ez azt jelenti, hogy a vizsgált régió középpontjainak, méreteinek és oldalarányainak viszonylag durva mintavételezése sem rontja feltétlenül az eszköz pontosságát. Fotó: Sipa/Shutterstock.

felhasználhatjuk a befoglaló téglalapunk korrigálására is. Ez a lépés az úgynevezett **befoglalótéglalap-regresszió**.

Befoglalótéglalap-regresszió

Az objektumkeresők kimenetének minősítése körültekintést igényel. Először szükség van egy tesztkészletre: egy olyan képgyűjteményre, ahol a képen lévő minden egyes objektum valódi kategóriájának címkéjével és ideális befoglaló téglalapjával van megjelölve. A téglalapokat és a címkéket általában emberi szakértők állítják elő. Ezután az egyes képeket az objektumkeresőnek adjuk, és összehasonlítjuk annak a kimenetét a referencia, elvárt kimenettel. El kell fogadnunk a néhány képpont távolságban lévő téglalapokat, mert a referencia-téglalapok sem tökéletesek. Az értékelésre használt metrikának egyensúlyoznia kell a felidézés (a valódi objektumokat milyen gyakorisággal találja meg, angolul: recall) és a precizitás (mennyi a fals, objektumként érzékelt jelölt) szerinti teljesítmény között.

**25.16. ábra** Ma már lehetséges emberek rekonstruálása egyetlen képből. Az egyes sorok egy-egy háromdimenziós test alakjának rekonstruálását mutatják, egyetlen kép felhasználásával. Ezek a rekonstrukciók azért lehetségesek, mert léteznek módszerek, amelyekkel a képhez viszonyítva megbecsülhetők az ízületek elhelyezkedése, azok állása, a test alakja és a test helyzete. Minden sor a következőket mutatja: bal szélen maga a kép; mellette a kép és a rekonstruált test egymásra helyezve; ez után a rekonstruált test egy másik nézete; végül a jobb szélen a rekonstruált test egy újabb nézete. A test eltérő nézetei sokkal nehezebbé teszik a rekonstrukció során elkövetett hibák eltussolását. Az ábra Angjoo Kanazawanától származik, az azt előállító megoldást Kanazawa *et al.* (2018a) ismerteti.

Képi információ alapján megállapítani, hogy mit tesznek a rajtuk szereplő emberek, már jóval nehezebb feladat. Kifejezetten könnyű kezelni a meglehetősen strukturált viselkedést mutató mozgóképeket, mint például a balettet, a tornát vagy a tai chit, ahol a cselekvést nagyon specifikus szavakkal leírhatjuk, melyek általában egyszerű háttér előtti nagyon pontosan körülhatárolt tevékenységekre utalnak. Jó eredmények érhetők el sok címkézett tanítóminta esetén megfelelő konvolúciós neurális hálózat alkalmazásával. Nehéz azonban bebizonyítani, hogy a módszerek valóban jól működnek, mivel jelentősen függhetnek a képen látható cselekvéskontextustól. Például egy olyan osztályozó, amely nagyon jól felismeri az „úszás” szekvenciákat, lehet, hogy csak egy úszómedence-felismerő, amely nem működne a folyóban úszókra.

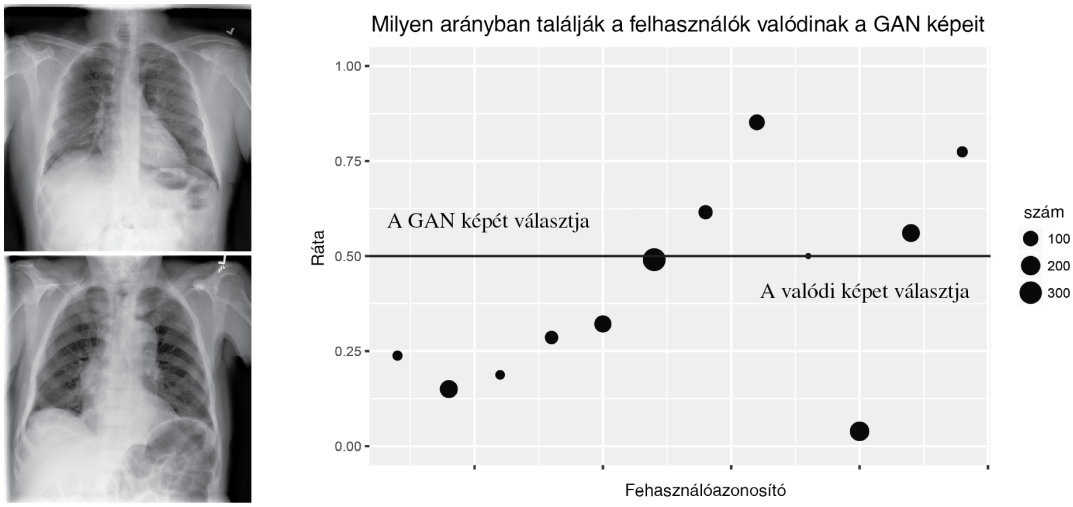
Más, általánosabb problémák továbbra is nyitottak – például hogy hogyan kapcsolható össze a test és az ahhoz közeli tárgyak megfigyelése a mozgó emberek céljaival és szándékaival. Az egyik nehézségi forrás az, hogy a hasonló viselkedések eltérően nézhetnek ki, míg különböző viselkedések hasonlónak látszhatnak, amint azt a 25.17. ábra mutatja.


**25.17. ábra** Ugyanaz a cselekvés nagyon eltérően nézhet ki, míg különböző tevékenységek hasonlónak látszódhatnak. Ezek a példák olyan tevékenységeket mutatnak be, amelyek a szereplők természetes viselkedése esetén kerültek rögzítésre; a címkéket kézzel határozták meg (nem pedig egy algoritmus). **Fent**: Példák a „hűtőszekrény nyitása" címkéjű jelenetekre, némelyik közeli felvételen, mások távolról. **Lent**: Példák a „kivenni valamit a hűtőből" címkére. Figyeljük meg, hogy mindkét sorban az alany keze közel van a hűtőszekrény ajtajához – a két kategória egymástól történő megkülönböztetéséhez elég finom megítélés szükséges arról, hogy hol van a kéz, és hol van az ajtó. Az ábra David Fouhey-nak köszönhető, és egy Fouhey *et al.* (2018) által leírt adathalmazból származik.

További nehézséget okoz az időbeliség. Sokszor a cselekvés időtartama is kritikus annak megítélése szempontjából, amint azt a 25.18. ábra szemlélteti. Az ábrán bemutatott másik fontos hatás az, hogy a viselkedés összeáll – azaz több, egyenként felismert cselekmény összeilleszthető úgy, hogy egy magasabb szintű viselkedés alakuljon ki, példánk esetében valamilyen harapnivaló elkészítése során.

Előfordulhat, hogy egyidejűleg független cselekvések is zajlanak, például énekelnek uzsonna készítése közben. Kihívás az is, hogy nincs egységes szótárunk a viselkedések egyes részleteinek pontos leírására. Az emberek általában azt gondolják, hogy sok viselkedésformának tudják a nevét, de ha megkérjük őket, akkor sem tudnak hosszú listákat készíteni ezekből a szavakból. Ez tovább nehezíti következetesen címkézett viselkedés-adatkészletek készítését.

A tanított osztályozók pontossága csak akkor garantálható, ha a tanító- és tesztminták ugyanazon eloszlásból származnak. Nem tudjuk ellenőrizni, hogy ez a megkötés teljesül-e a konkrét feladathoz kapcsolódó képekre is, de empirikusan megfigyelhetjük, hogy a képosztályozók és az objektumfelismerők jól működnek-e. A cselekvésekhez tartozó minták esetében azonban a tanító- és a tesztadathalmazon elérhető eredmények eltérő minőségűek lehetnek, mivel az emberek sok dolgot sok különböző kontextusban tehetnek. Tegyük fel például, hogy van egy gyalogosfelismerőnk, ami jól teljesít nagy adatkészlet esetén. Lesz olyan ritka jelenség (például emberek egykerekűznek), amelyek nem jelennek meg a tanítókészletben, így nem várható el, hogy a felismerő helyesen működjön ilyen esetekben. Nagy kihívás, hogy bebizonyítsuk, hogy egy felismerő pontos eredményt ad, bármit is tesznek a gyalogosok. Ez a jelenlegi apparátussal nehezen megvalósítható.




**25.27. ábra** GAN által készített tüdőröntgenképek. A bal oldali képpár közül az egyik egy valódi, a másik egy GAN által generált röntgenkép. A jobb oldalon annak a tesztnek az eredményei látszanak, amelyben radiológusokat kértek fel, hogy röntgenképpárokról állapítsák meg, melyik a valódi, és melyik a generált felvétel. Átlagosan az esetek 61%-ában választottak helyesen, valamivel jobb arányban, mint a véletlenszerű választás. Pontosságukban azonban különböztek egymástól – a jobb oldali táblázat mutatja a 12 különféle radiológus hibaarányát; egyikük hibaaránya közel 0%, míg egy másiké 80%-os volt. Az egyes foltok mérete jelzi a radiológus által megtekintett képek számát. Az ábra Alex Schwingnektől származik, és a Deshpande *et al.* (2019) által leírt rendszer alkotta.

milyen gyakran fordulnak elő (például a tüdőgyulladásos esetek). Az új adatkészletnek különböznie kell abban az értelemben, hogy nem tartalmazhat személyek azonosítására felhasználható információkat. Az új adatkészletnek szabályozhatónak kell lennie, hogy a különböző, jellemző esetek gyakorisága a konkrét alkalmazói kör igényeit kielégítse. Például a tüdőgyulladások gyakoribbak az időskorúak esetén, mint a fiatal felnőttekben. Ezen célok mindegyikét technikailag nehéz elérni, de olyan képkészleteket már sikerült létrehozni, amelyek időnként megtréfálják még a gyakorlott radiológusokat is (lásd a 25.27. ábrát).

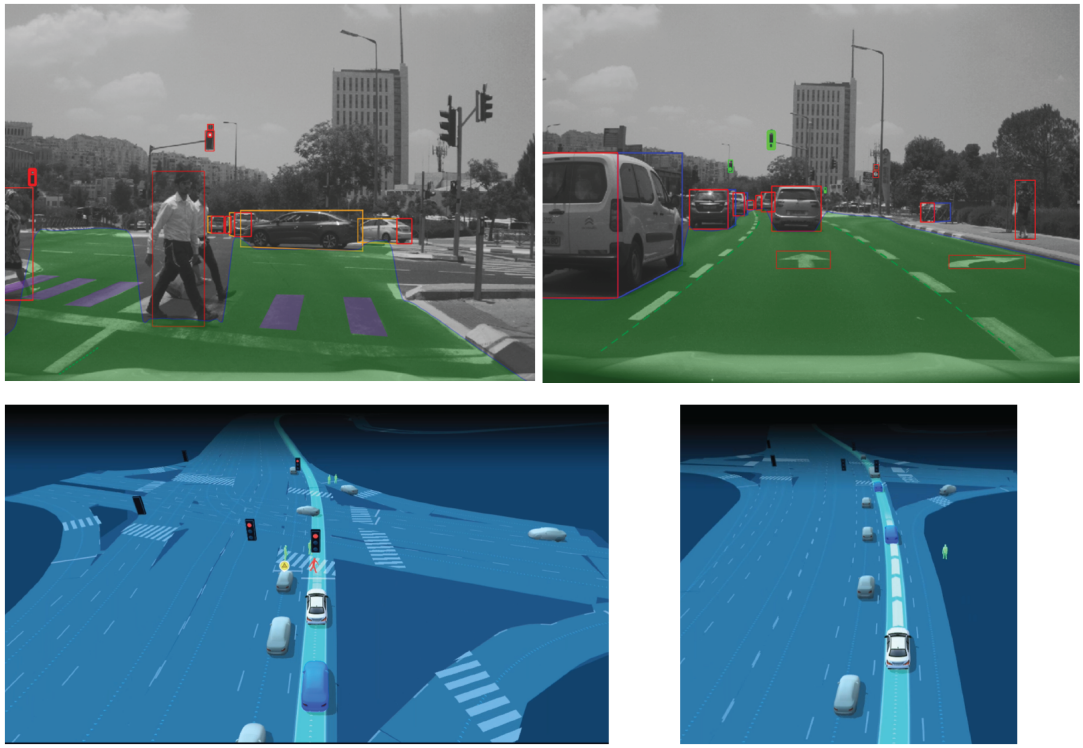
### 25.7.6. Mozgásirányítás látással

A látás egyik fő felhasználási területe információk szolgáltatása mind a tárgyak mozgatásához – azok felvételéhez, megragadásához, forgatásához stb. –, mind mozgás navigációjához, mely során az esetleges ütközéseket próbáljuk elkerülni. A látás e célokra történő felhasználásának képessége a legprimitívebb állati látásrendszerekkel is lehetséges. Sok esetben a látórendszer minimális abban az értelemben, hogy a rendelkezésre álló fénymezőből csak azokat az információkat vonja ki, amelyek az állatok viselkedéséhez szükségesek. Valószínűleg a modern látásrendszerek olyan korai, egyszerű szervezetekből fejlődtek ki, amelyek fényérzékeny foltot használtak annak érdekében, hogy a fény felé közeledjenek (vagy attól távolodjanak). A 25.6. alfejezetben láttuk, hogy a

legyek nagyon egyszerű optikai áramlásérzékelő rendszert használnak a falra történő leszálláshoz.

Tegyük fel, hogy a falakon való leszállás helyett egy önvezető autót akarunk építeni. Ez egy olyan projekt, amely sokkal nagyobb igényeket támaszt az észlelési rendszerrel szemben. Az önvezető autóban történő észlelésnek a következő feladatokat kell támogatnia:

- **Oldalirányú kontroll**: Gondoskodjon arról, hogy a jármű maradjon biztonságosan a sávjában, vagy szükség szerint zökkenőmentesen változtassa meg a sávját.
- **Haladási irányú kontroll**: Gondoskodjon arról, hogy az előtte lévő járműhöz képest biztonságos távolságban legyen.



**25.28. ábra** A Mobileye kamera alapú autonóm járművek számára fejlesztett érzékelőrendszer. **Felső sor**: Két kép az előlre néző kamerából, pár másodperc különbséggel. A szabad felületeket zöld szín jelöli – ahova a jármú fizikailag mozoghatna a közeljövőben. Az egyes objektumokat 3D befoglaló téglatestekkel jelölik (piros a hátulja, zöld az eleje, kék a jobb, sárga a bal oldala). Az objektumok között járművek, gyalogosok, (az oldalirányú irányításhoz szükséges) sávhatárjelölők, valamint egyéb útburkolati jelek, táblák és jelzőlámpák fordulnak elő. Az állatokat, oszlopokat, terelőbójákat, járdákat, korlátokat és egyéb objektumokat (például egy teherautóról leesett kanapé) nem jelöli. Minden objektumhoz ezután egy 3D-pozíciót és sebességet rendel. **Alsó sor**: A környezet teljes fizikai modellje. (A képek a Mobileye rendszer kimeneteit mutatják.)

- **Akadály elkerülése**: Kísérje figyelemmel a szomszédos sávokban lévő járműveket, és készüljön fel kitérő manőverekre. A gyalogosokat észlelje, és biztosítsa a biztonságos átkelésüket.
- **Tartsa be a forgalmi jelzéseket**: Ezekbe beletartoznak a közlekedési lámpák, a stoptáblák, a sebességkorlátozó táblák és a rendőrök kézjelzései is.

A vezető (ember vagy számítógép) feladata az, hogy megfelelő kormányzási, gyorsítási és fékezési műveleteket hajtson végre e részfeladatok legjobb megvalósítása érdekében.

Hogy jó döntéseket hozzon meg, a járművezetőnek létre kell hoznia a világ és a benne lévő tárgyak modelljét. A 25.28. ábra néhány, a modell felépítéséhez szükséges vizuális következtetést mutat be. Az oldalirányú kontrollhoz a sofőrnek meg kell őriznie az autó helyzetét és haladási irányát a sávok alakjának megfelelően. A haladási irányú kontrollhoz a vezetőnek biztonságos távolságot kell tartania az előtte lévő járműtől (amit esetleg nem könnyű azonosítani kanyarodó utakon). Az úttesten lévő akadályok elkerülése és az útjelzések követése további következtetéseket igényel.

Az utak emberek számára készültek, akik a látott információk alapján vezetnek, tehát elvben lehetséges csak a látott információk alapján vezetniük. A gyakorlatban azonban a kereskedelemben kapható önvezető autók számos egyéb érzékelőt használnak, beleértve a kamerákat, LIDAR szenzorokat, radarokat és mikrofonokat. A LIDAR vagy a radar lehetővé teszi a mélység közvetlen becslését, amely pontosabb lehet, mint a 25.6. alfejezetben szereplő, csak a látáson alapuló módszerek. Több érzékelővel általában javul a teljesítmény, és ez különösen fontos rossz látási viszonyok között; például a radar képes áthatolni a ködön, amely akadályozza a kamerákat és a LIDAR szenzorokat. A mikrofonok észlelni tudják a közeledő járműveket (különösen a megkülönböztető jelzésekkel felszerelteket), még mielőtt azok láthatóvá válnának.

Szintén sok kutatást végeztek a beltéri és kültéri környezetben navigáló mobilrobotokkal. Rengeteg alkalmazási lehetősége létezik, például a csomag- vagy pizzaszállítás utolsó lépései. A hagyományos megközelítések ezt a feladatot két szakaszra bontják, amint azt a 25.29. ábra mutatja:

- **Térkép készítése**: A szimultán helymeghatározás és térképezés (SLAMSimultaneous Localization and Mapping) (lásd az 1086. oldalon) a világ háromdimenziós modelljét készíti el, melynek része a robot helye is a világban (pontosabban, a robot kameráinak a helye). Ezt a modellt (amelyet általában akadályok pontfelhőjeként reprezentálnak) különböző kamerahelyzetekből készült képek sorozatából lehet felépíteni.
- **Útvonaltervezés**: Amint a robot hozzáfér ehhez a 3D-s térképhez, és lokalizálja magát azon, a célja az lesz, hogy találjon egy ütközésmentes pályát az aktuális pozíciójától a célig (lásd a 26.6. alfejezetben).

Ennek az általános megközelítésnek számos változatát vizsgálták. Például a kognitív feltérképezés és tervezés esetén a térképkészítést és az útvonaltervezést két modul végzi egy neurális hálózaton belül, melynek minden részét tanítják (end-to-end) egy veszteségfüggvény minimalizálása segítségével. Egy ilyen rendszernek nem kell teljes térképet készítenie – amely gyakran felesleges és szükségtelen –, ha mindössze csak annyi információra van szükségünk, hogy A pontból B pontba eljussunk anélkül, hogy akadályokba ütköznénk.

## 26.3. Milyen problémákat old meg a robotika?

Most, hogy már megismertük, milyen lehet egy robot hardvere, készen állunk arra, hogy egy robot ágens szoftverét megvizsgáljuk. Először számítási keretrendszert választunk az ágensünk számára. Korábban már tárgyaltuk a keresést determinisztikus környezetekben, a Markov döntési folyamatokat (MDF) sztochasztikus, de teljesen megfigyelhető környezetben, illetve a részlegesen megfigyelhető Markov döntési folyamatokat (RMMDF, angolul: partially observable Markov decision process, POMDP) részlegesen megfigyelhető környezetben, valamint a játékelméleti megközelítést azokban az esetekben, ahol az ágens nem elszigetelten cselekszik. Adott számítási keretrendszerhez meg kell határoznunk az egyes alegységeket: a jutalom- vagy hasznosságfüggvényeket, az állapotokat, a cselekvéseket, a megfigyelési tereket stb.

Korábban már megjegyeztük, hogy a robotikai feladatok nemdeterminisztikusak, részlegesen megfigyelhetők és multiágensesek. A 18. fejezetben vázolt játékelméleti megfontolások mellett az vehetjük észre, hogy az ágensek néha kooperálnak, néha pedig versenyeznek egymással. Egy szűk folyosón, ahol egyszerre csak egy ágens fér el, a robot és az ember együttműködnek, mert egyik sem akar összeütközni a másikkal. Más esetekben versenyeznek, hogy melyikük éri el gyorsabban a célját. Ha a robot túl tisztelettudó, és mindig átadja a helyet másoknak, akkor zsúfolt környezetben beragadhat, és sosem éri el a célját.

Így az elszigeteltségben cselekvő, és környezetüket ismerő robotok által megoldandó feladatot MDF-ként, míg hiányos információ esetén RMMDF-ként, és ha emberek környezetében kell cselekedniük, úgy játékként fogalmazzuk meg.

Ebben a megközelítésben mi legyen a robot jutalomfüggvénye? Általában a robot az ember szolgálatában cselekszik, és nem önmagáért – mint például amikor egy kórházban ételt kézbesít egy betegnek, a beteg „jutalmazására”. Annak ellenére, hogy a robot tervezői megkísérelhetnek egy megfelelő absztrakt jutalomfüggvényt specifikálni, a valódi jutalmat a robot felhasználója kapja, akinek a robot segíteni próbál. A robotnak ki kell találnia a felhasználójának a vágyait, vagy egy mérnökre kell támaszkodnia, aki ezt közelítőleg specifikálni tudja.

Ami pedig a robot cselekvési, megfigyelési és állapotterét illeti, a robot szenzorainak nyers adatai a megfigyelések leginkább általános formája (például a kamerákból érkező képek, a lidar visszavert sugarai), a cselekvései a motorokba irányított áram erőssége, az állapota pedig mindaz, amit tudnia kell a döntéshozatalhoz. Ez alapján óriási szakadék tátong az alacsony szintű megfigyelések és vezérlőáramok, és a robot által elkészítendő magas szintű tervek között. Ennek áthidalására a robotológusok ezt a feladatot kisebb részegységekre bontva egyszerűsítik le.

Tudjuk például, hogy amikor helyesen oldunk meg RMMDF-eket, akkor az észlelés és a cselekvés hatással vannak egymásra: az észlelés információt ad észszerű cselekvésekről, valamint a cselekvés is informálja az észlelést, mert az ágensek információgyűjtő cselekvéseket hajthatnak végre, annak reményében, hogy az információ a jövőben hasznos lesz. Ennek ellenére a robotok gyakran elválasztják az észlelést a cselekvéstől, pusztán felhasználva az észlelést, és közben úgy tesznek, mintha a jövőben több információra nem tennének szert. Mivel egy magas szintű cél, mint amilyen a „juss el

az ebédlőbe", távol áll egy „főtengely forgatása 1°-kal" jellegű motor parancstól, ezért mindenképpen szükséges a hierarchikus tervezés.

Feladattervezés

Gyakran egy háromszintű hierarchiát használunk a robotikában. A **feladattervezés**i szint dönt a magas szintű cselekvések (amelyeket néha cselekvési primitíveknek, vagy alcéloknak nevezünk) tervéről vagy eljárásmódjáról, mint amilyen az ajtóhoz mozgás, ajtó kinyitása, felvonóhoz mozgás, hívógomb megnyomása stb. A **mozgástervezés** feladata egy olyan útvonal keresése, amelyen a robot eljuthat az egyes alcélok pontjaiba. Végezetül, a robot aktuátorait használva a **vezérlés** segítségével követheti a tervezett útvonalat. Ebben a fejezetben elsődlegesen a mozgástervezéssel foglalkozunk, mert a feladattervezést általában diszkrét állapotok és cselekvések felett definiáljuk.

Vezérlés

Preferenciatanulás

Emberi predikció

Ezektől elkülönülve a **preferenciatanulás** felelős a végfelhasználó céljainak becsléséért, továbbá az **emberi predikció** arra szolgál, hogy megjósolja a robot környezetében lévő más emberek cselekvéseit. Mindezek együttesen határozzák meg a robot viselkedését.

Amikor egy feladatot kisebb darabokra bontunk fel, akkor csökkentjük a komplexitását, viszont óhatatlanul elszalasztunk lehetőségeket arra, hogy azok egymást segítsék. A cselekvés segíthet javítani az észlelésen, sőt segíthet abban, hogy eldöntsük, milyen észlelések hasznosak. Hasonlóan, a mozgás szintű cselekvések nem biztos, hogy kellően figyelembe veszik a mozgás útvonalának követését. Egyes tervszintű döntések pedig olyan terveket is eredményezhetnek, amelyek mozgási szinten végrehajthatatlanok. Mindazonáltal az egyes szinteken elért fejlődés szükségessé teheti az egyes szintek újraintegrálását, hogy együttesen végezzük a mozgástervezést és a vezérlést, valamint a feladat- és a mozgástervezést, és hogy a visszacsatolási hurkot bezárva újra összekössük az észlelést, cselekvést és a predikciót.

Napjainkban a robotika egyszerre jelenti az egyes területeken mutatott folyamatos fejlődést és ezen újdonságok mélyebb integrálását is.

## 26.4. Érzékelés a robotikában

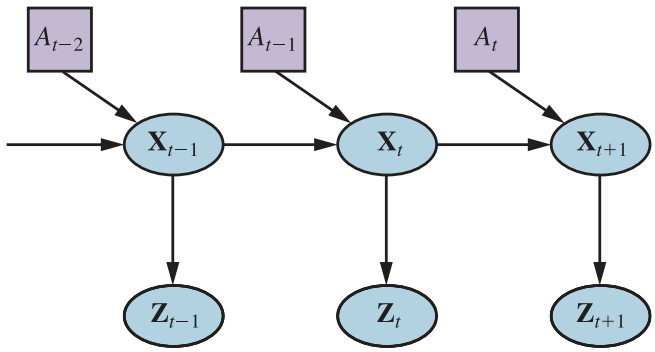
Az érzékelés az a folyamat, amely során a robot a szenzoraiból érkező jeleket a környezetének belső reprezentációjára képezi le. Ehhez gyakran az előző fejezetben tárgyalt gépi látási technikákat alkalmazzuk, de a robotok észlelését ki kell egészíteni további szenzorokkal, például pásztázó távolságmérőkkel és érintésérzékelőkkel.

Az érzékelés nehéz, mivel általában a szenzorok adatai zajjal terheltek, a környezet csak részlegesen megfigyelhető, nem jósolható viselkedésű és gyakran dinamikus is. Másképpen fogalmazva a robotok az **állapotbecslés** (vagy **szűrés**) összes problémájával szembesülnek, ahogyan ezt a 14.2. alfejezetben láttuk.

Ökölszabály, hogy egy jó belső reprezentációnak három tulajdonsága van:

1. Elég információt hordoz ahhoz, hogy a robot a megfelelő döntést meghozhassa.
2. Strukturáltsága lehetővé teszi, hogy hatékonyan frissíthető legyen.
3. Természetes, olyan értelemben, hogy a belső állapotváltozók megfeleltethetők egy-egy valós fizikai világbeli állapotváltozónak.

A 14. fejezetben megmutattuk, hogy a Kalman-szűrők, a rejtett Markov-modellek és a dinamikus Bayes-hálók alkalmasak egy részlegesen megfigyelhető környezet álla-




**26.4. ábra** A robotérzékelés felfogható úgy, mint cselekvések és mérések sorozatának időbeli egymásra hatása, ahogy azt ez a dinamikus Bayes-háló is illusztrálja.

potátmeneti és szenzormodelljeinek a reprezentálására. Ismertettünk egzakt és közelítő algoritmusokat a belső **hiedelmi állapot** – a környezet állapotváltozói felett értelmezett a posteriori valószínűségi eloszlás – frissítésére. Erre számos dinamikus Bayes-hálós modellt mutattunk be a 14. fejezetben. Robotikai problémák esetében a modellhez általában megfigyelt változóként a robot saját korábbi cselekvéseit is hozzávesszük.

A 26.4. ábra ennek a fejezetnek a jelöléseit mutatja: $\mathbf{X}_t$ a környezet állapota (beleértve a robotot is) a $t$ időpillanatban, $\mathbf{Z}_t$ a $t$ időpontbeli megfigyelés (az érzékelésből származó adatok), az $A_t$ pedig az érzékelést követően végzett cselekvés.

A szűrési (filtering) feladat, vagyis a hiedelmi állapot frissítése alapvetően azonos a 14. fejezetben tárgyaltakkal. A feladat az új hiedelmi állapot, $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t})$ kiszámítása a $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ aktuális hiedelmi állapotból és az új $\mathbf{z}_{t+1}$ megfigyelésből. Az alapvető különbség a 14.2. alfejezethez képest az, hogy (1) explicit feltételekkel élünk mind a döntést, mind a megfigyelést illetően, továbbá (2) *diszkrét* helyett *folytonos* változókkal kell dolgoznunk. Ezért módosítanunk kell az 550. oldalon található (14.5) rekurzív szűrőalgoritmust, összegzés helyett integrálást alkalmazva:

$$\begin{aligned}&\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t}) \\ &\quad = \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)\, P(\mathbf{x}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})\, d\mathbf{x}_t\,. \end{aligned} \tag{26.1}$$

Az egyenlet azt fejezi ki, hogy az $\mathbf{X}$ állapotváltozó feletti a posteriori eloszlást a $t+1$-edik időpillanatban rekurzívan számítjuk az egy időlépéssel korábbi megfelelő becslésből. A számításokhoz felhasználjuk a korábbi cselekvést, $a_t$-t, és az aktuális szenzoros megfigyelést, $\mathbf{z}_{t+1}$-et. Ha például egy futballozni képes robotot fejlesztünk, akkor $\mathbf{X}_{t+1}$ lehet a labda relatív helyzete a robothoz képest. A $\mathbf{P}(\mathbf{X}_t \,|\, \mathbf{z}_{1:t}, a_{1:t-1})$ a posteriori eloszlás mindazon állapotok felett értelmezett valószínűségi eloszlás, amelyek megőrzik mindazt, amit a korábbi érzékelő mérésekből és irányításokból tudunk. A (26.1) egyenlet megmondja, hogyan becsüljük rekurzívan ezt a pozíciót, folyamatosan felhasználva az újabb szenzoradatokat (például kameraképeket) és a robot mozgásparancsait. A $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)$ valószínűségi eloszlást **állapotátmeneti modell**nek vagy más néven **mozgásmodell**nek hívjuk, míg $\mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1})$ valószínűségi eloszlást **szenzormodell**nek.

Mozgásmodell

### 26.4.1. Helymeghatározás és feltérképezés

Helymeghatározás

A **helymeghatározás** problémája a dolgok (beleértve magát a robotot is) pontos helyzetének meghatározása. Ahhoz, hogy egyszerűvé tegyük a dolgot, feltételezzük, hogy a robotunk lassan, sík terepen mozog, és pontos térképe van a környezetéről. (Egy ilyen térkép látható a 26.7. ábrán.) Egy mobilrobot helyzetét két derékszögű koordinátával, $x$ és $y$-nal, valamint az irányát jellemző szöggel írjuk le, ahogy ez a 26.5. ábrán (a) részén is látható. Ha ezt a három értéket egy vektorba fogjuk össze, akkor bármely állapot megadható $\mathbf{X}_t = (x_t, y_t, \theta_t)^\top$ formában.

Kinematikai közelítésben minden cselekvés felbontható két „pillanatnyi" sebességre: egy $v_t$ (transzlációs) sebességre és egy $\omega_t$ (rotációs) szögsebességre. Kis $\Delta t$ időkre a robot mozgásának egy durva determinisztikus modellje az alábbi alakban adható meg:

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta t \end{pmatrix} .$$

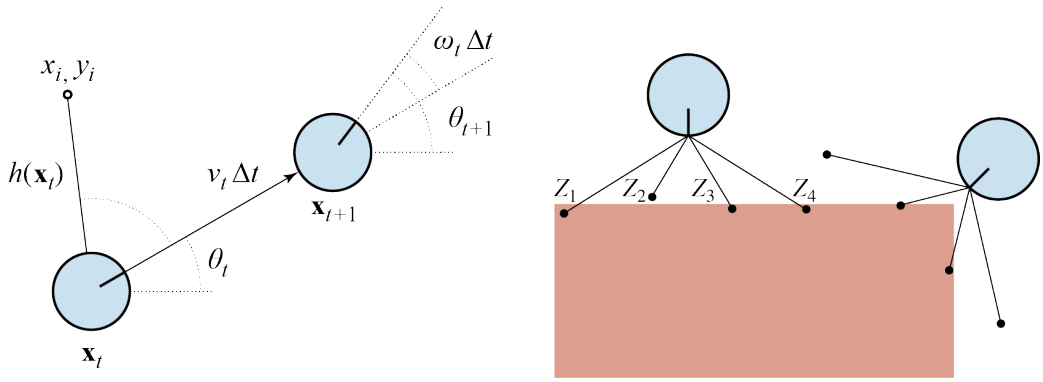
Az $\hat{\mathbf{X}}$ jelölés a determinisztikus állapotbecslésre vonatkozik. Természetesen a fizikai robotok valamilyen módon mindig megjósolhatatlanok maradnak. Ezt gyakran modellezik egy $f(\mathbf{X}_t, v_t, \omega_t)$ középértékű, $\mathbf{\Sigma}_x$ kovarianciájú Gauss-eloszlással. (Lásd még az A. függelék matematikai definícióit.)

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, v_t, \omega_t) \;=\; \mathcal{N}(\hat{\mathbf{X}}_{t+1}, \mathbf{\Sigma}_x) .$$

Ez a valószínűségi eloszlás modellezi az $a_t$ mozgás hatását a robot helyzetére.

A következő lépésben szükségünk van egy szenzormodellre. Kétféle szenzormodellről beszélhetünk. Az első azon alapul, hogy a szenzor **referenciapontok**nak nevezett

Referenciapont




(a) (b)

**26.5. ábra** (a) Egy mobilrobot egyszerűsített kinematikai modellje. A robotot a kör jelképezi, és a sugár mutatja a haladási irányt. A robot $\mathbf{x}_t$ állapota a $(x_t, y_t)$ pozícióból és a $\theta_t$ orientációból áll. A robot új, $\mathbf{x}_{t+1}$ állapotát a pozíció $v_t\Delta_t$-vel, az orientáció $\omega_t\Delta_t$-vel való frissítésével kapjuk meg. Az ábrán továbbá fel van tüntetve egy $t$ időpontban megfigyelt referenciapont (tereptárgy) az $(x_i, y_i)$ pontban. (b) A pásztázó távolságmérés modellje. Egy adott távolságú méréshez $(z_1, z_2, z_3, z_4)$ tartozó két lehetséges robothelyzet látható. Sokkal valószínűbb, hogy a bal oldali helyzetből származnak a távolságmérések.

*állandó* és *felismerhető* jellemvonásokat érzékel és azonosít a környezetében. Minden referenciapontnak meghatározza a távolságát és a szögét. Tegyük fel, hogy a robot az $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$ állapotban van, és érzékel egy ismert $(x_i, y_i)^\top$ helyen lévő referenciapontot. Ha nem vesszük figyelembe az érzékelés zajosságát, akkor egy egyszerű geometriai számítással megkapható a távolság és a szög (lásd a 26.5. ábra (a) részét):

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix} .$$

A zaj azonban most is torzítja a méréseinket. Egyszerűsítésképpen normális eloszlású, $\mathbf{\Sigma}_z$ kovarianciájú zajt tételezünk fel, amely a következő szenzormodellt eredményezi:

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \mathcal{N}(\hat{\mathbf{z}}_t, \mathbf{\Sigma}_z) .$$

Némileg különböző szenzormodellt használunk távolságmérőkből álló **érzékelőtömbök**re, amelyek mindegyike a robothoz képest rögzített helyzetben van. Az ilyen érzékelők egy $\mathbf{z}_t = (z_1, \ldots, z_M)^\top$ vektort adnak vissza távolságadatokkal, amelyek a robothoz képest rögzített irányokban lévő tárgyakról hordoznak információt. Adott $\mathbf{x}_t$ pozícióban legyen $\hat{z}_j$ a $j$-edik sugár irányában lévő legközelebbi akadály pontos távolsága. Csakúgy, mint korábban, most is Gauss-zajt tételezünk fel. Általánosan vehetjük úgy, hogy a különböző sugáriránybeli hibák független és azonos eloszlásúak, így:

Érzékelőtömb

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \alpha \prod_{j=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2} .$$

A 26.5. ábra (b) része két különböző pozíció esetén mutat be egy példát a négyirányú távolságmérő szkennerre. A kettő közül az egyik sokkal nagyobb valószínűséggel adhatta a megfigyelés négy értékét. A pásztázó távolságméréses modellt a referenciapontos modellel összehasonlítva látható, hogy az előbbinek az az előnye az utóbbival szemben, hogy nincs szükség egy referenciapont *azonosítására*, mielőtt egy tartománypásztázás értelmezhető lenne. Sőt az is előfordulhat – ahogy a 26.5. ábra (b) részének esetében is –, hogy a robot egy jellegtelen fallal találkozik. Ugyanakkor, ha *van* egy tisztán látható, azonosítható referenciapont, akkor abból azonnal meghatározható a pontos helyzet.

A 14.4. alfejezetben már találkoztunk a Kalman-szűrővel, amely a hiedelmi állapotot egyetlen többváltozós Gauss-eloszlással reprezentálja, illetve a részecskeszűrővel, amely a hiedelmi állapotot az állapotokhoz rendelt részecskeegyüttessel reprezentálja. A legtöbb modern lokalizáló algoritmus e két módszer egyikét használja a robot $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ hiedelmi állapotának leképzésére.

A részecskeszűrős lokalizációs algoritmust **Monte Carlo lokalizáció**nak (Monte Carlo localization, MCL) hívják. Az MCL alapvetően megegyezik a 14.17. ábrán (lásd az 578. oldalon) bemutatott részecskeszűrős algoritmussal, mindössze annyit kell módosítanunk, hogy szükségünk van a megfelelő mozgás- és szenzormodellre. A 26.6. ábra egy pásztázó távolságméréses modellel megvalósított változatot mutat be. A 26.7. ábra szemlélteti az algoritmus működését, hogy hogyan azonosítja a robot saját tartózkodási helyét az irodaépületben. Az első képen még a részecskék egyenletesen oszlanak el az előzetes információ alapján, ami a teljes bizonytalanságot jelzi a robot pozícióját illetően. A második képen már az első mérések alapján klasztereket alkotnak a részecskék, oda tömörülve, ahol valószínűleg tartózkodik a robot. A harmadik esetben pedig

Monte Carlo lokalizáció

```
function Monte-Carlo-Helyzetmeghatározás(a, z, N, P(X'|X, v, ω), P(z|z*),
        térkép)
  returns S, a következő időpont mintahalmaza
  inputs: a, a robot v és ω sebességei
          z, egy pásztázó távolságmérés M mérési ponttal
          P(X'|X, v, ω), mozgási modell
          P(z|z*), pásztázó távolságmérő zajmodell
          térkép, a környezet 2D térképe
  persistent: S, egy N mintából álló vektor
  local variables: W, egy N súlyokból álló vektor
                   S', egy N mintából álló ideiglenes vektor

  if S üres then
      for i = 1 to N do        // inicializálás
          S[i] ← mintavételezés a P(X_0)-ból
  for i = 1 to N do            // frissítési ciklus
      S'[i] ← mintavételezés a P(X'|X = S[i], v, ω)-ból
      W[i] ← 1
      for j = 1 to M do
          z* ← Egzakt-tartomány(j, X = S'[i], térkép)
          W[i] ← W[i] · P(z_j| z*)
      S ← Súlyozott-Minta-Helyettesítéssel(N, S', W)
  return S
```

**26.6. ábra** A Monte Carlo lokalizáló algoritmus, független zajjal terhelt, pásztázó távolságméréses modellt használva.

már elég információ áll rendelkezésre a mérésekből, hogy az összes részecske ugyanoda jusson.

A Kalman-szűrő a másik széles körben használt lokalizáló algoritmus. A Kalman-szűrő a $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ posteriort egy Gauss-eloszlással reprezentálja. Az eloszlás középértékét $\mu_t$-vel, kovarianciáját pedig $\mathbf{\Sigma}_t$-vel jelöljük. A fő probléma a Gauss-eloszlású hiedelmekkel az, hogy csak egy $f$ lineáris mozgási modell és egy $h$ lineáris mérési modell mellett zártak. Nemlineáris $f$ vagy $h$ esetén a szűrő frissítésének eredménye általában már nem lesz Gauss-eloszlású. Ezért a Kalman-szűrőt használó lokalizációs algoritmusok a mozgás- és szenzormodelleket **linearizálják**. A linearizálás egy nemlineáris függvény adott lokális lineáris közelítése. A 26.8. ábra a linearizálás elvét mutatja be egy (egydimenziós) robotmozgási modellen. A bal oldalon fut a mozgás nemlineáris $f(\mathbf{x}_t, a_t)$ modellje (az $a_t$ vezérlés nem szerepel az ábrán, mert nem játszik szerepet a linearizálásban). A jobb oldalon ezt egy $\tilde{f}(\mathbf{x}_t, a_t)$ lineáris függvénnyel közelítjük. A lineáris függvény a $\mu_t$ pontban érinti az $f$-et, ami a $t$ időre vonatkozó becslésünk középértéke. Ezt a linearizálást (első fokú) **Taylor-sorfejtés**nek (Taylor expansion) hívják. Az olyan Kalman-szűrőt, amely $f$-et és $h$-t Taylor-sorfejtéssel linearizálja, **kiterjesztett Kalman-szűrő**nek (KKSz) (extended Kalman filter, EKF) hívjuk. A 26.9. ábra egy KKSz helymeghatározó algoritmust használó robot becslés-sorozatát mutatja.

Linearizálás

Taylor-sorfejtés