

# Gépi tanulás

egyetemi jegyzet

Bolgár Bence

Budapesti Műszaki és Gazdaságtudományi Egyetem  
Méréstechnika és Információs Rendszerek Tanszék  
2021. augusztus



# Tartalomjegyzék

Előszó 5

1 Bayesi valószínűségelmélet 1

1.1 A bayesi gondolkodásmód 1

1.2 Naïv bayesi osztályozók 8

2 Lineáris regresszió 11

2.1 Valószínűségi modell 11

2.2 Maximum likelihood megoldás 13

2.3 Bázisfüggvények 14

2.4 MAP becslés és regularizáció 18

2.5 A bias-variancia dilemma 19

2.6 Bayesi lineáris regresszió 22

3 Lineáris klasszifikáció 25

3.1 Valószínűségi modell 25

3.2 Maximum likelihood megoldás 29

3.3 Bázisfüggvények több dimenzióban 31

3.4 Az implementáció részletei 32

3.5 Klasszifikációs modellek kiértékelése 34

## 4 CONTENTS

4	Neurális hálózatok	37
4.1	Logisztikus regresszió és a perceptron	37
4.2	Többrétegű neurális hálózatok	39
4.3	Tanítás hibavisszaterjesztéssel	40
4.4	Aktivációs függvények és veszteségfüggvények	42
4.5	Regularizáció neurális hálózatokban	45
4.6	Optimalizáció neurális hálózatokban	48
4.7	Konvolúciós neurális hálózatok	52
4.8	Automatikus differenciálás	55
5	Variációs közelítés	61
5.1	Evidence lower bound (ELBO)	61
5.2	Bayesi logisztikus regresszió	64
5.3	Variációs autoenkóder	67
5.4	Versengő modellek	69
6	Expectation-Maximization	73
6.1	Rejtett változók és maximum likelihood	73
6.2	Klaszterezés Gauss-keverék eloszlásokkal	74
7	Markov Chain Monte Carlo	81
7.1	Következtetés mintavétellel	81
7.2	Gibbs-mintavétel	85
8	Szupportvektor-gépek	95
8.1	Maximális margójú osztályozók	95
8.2	Optimalizáció a Lagrange-módszerrel	99
8.3	Szupportvektorok és predikció	100
8.4	Nemlineáris megoldás és a kernel trükk	102

9	<i>Főkomponens-analízis</i>	105
9.1	<i>Dimenzió-redukció</i>	105
9.2	<i>Kernel PCA</i>	109
10	<i>Megerősítéssel tanulás</i>	111
10.1	<i>Többkarú rablók</i>	111
10.2	<i>Monte Carlo Tree Search</i>	117
10.3	<i>AlphaZero</i>	120
11	<i>Federált tanulás</i>	121
11.1	<i>Valószínűségi modell</i>	121
11.2	<i>MAP megoldás</i>	122
11.3	<i>Variációs közelítés</i>	123
A	<i>Newton–Raphson módszer</i>	129
B	<i>Feltételes optimalizáció Lagrange-multiplikátorokkal</i>	133
C	<i>Közelítő módszerek integrálok kiszámításához</i>	135
	<i>Irodalomjegyzék</i>	137



# Előszó

Ez a jegyzet a BME Gépi tanulás c. tárgyhöz készül. Elsősorban a 2020/21. tanév őszi félévének előadásaira épül, amelyeket a járvány miatt távoktatásban voltunk kénytelenek tartani. Az anyag feldolgozásához a BSc-s analízis, lineáris algebra és valószínűségszámítás tárgyak ismerete szükséges. Az elmélet tárgyalása során bayesi szemléletmódot követünk, amelybe a klasszikus és modern eredmények, algoritmusok egyaránt illeszkednek; áttekintünk egészen friss kutatásokat is. Mindazonáltal fontos megjegyezni, hogy ez nem egy *deep learning* kurzus, és bár megismerkedünk a neurális hálózatok alapjaival és módszereivel, a mélytanuló architektúrákat – néhány kivételtől eltekintve – nem részletezzük. A gyakorlatokhoz Python-t fogunk használni (a notebook-ok megtalálhatók a tárgy tanszéki honlapján), viszont a jegyzetben a példák és algoritmusok Julia nyelven szerepelnek. Azért döntöttünk így, mert amellet, hogy ez a nyelv kitűnően alkalmas a gépi tanulási számítások megvalósítására, a matematikai formalizmushoz is közel áll. Minden kód változtatás nélkül futtatható – olyannyira, hogy a jegyzetben szereplő ábrák is közvetlenül ezen kódrészletek hívásával jönnek létre a pdf generálása során. Kérdéseket, visszajelzéseket örömmel fogadok a tanszéki e-mail címenem.

BOLGÁR BENCE

Budapest

2021. augusztus 31.





## 1. fejezet

# Bayesi valószínűségelmélet

Ebben a fejezetben megismerkedünk a Bayes-tétellel, amely nagyon sok gépi tanulási módszer alapját képezi – így az itt megismert ötletek, számítások a későbbi fejezetekben is gyakran vissza fognak köszönni. Különböző eloszlásokkal, közelítésekkel és egyszerűsítésekkel, de végső soron mindvégig az lesz a cél, hogy a megfigyelt adatokból ismeretlen mennyiségekre következtessünk, prediktív modelleket állítsunk fel.

### 1.1 A bayesi gondolkodásmód

Kezdésnek vegyünk egy egyszerű példát. Legyen egy érménk, amellyel fejet (jelölje mondjuk 1) vagy írást (0) lehet dobni. A célunk az, hogy sok-sok megfigyelt dobás alapján megpróbáljuk kikövetkeztetni, hogy az érme mekkora valószínűséggel dob egyiket vagy másikat. Jelölje  $y_i$  az  $i$ -edik megfigyelésünket,  $\theta$  pedig az említett ismeretlen valószínűséget, amelyre most az érme *paramétereként* fogunk gondolni. Mindezt a valószínűségszámítás nyelvén a következőképpen írhatjuk<sup>1</sup>:

$$p(y_i = 1 \mid \theta) = \theta,$$

azaz a fej dobásának valószínűsége a  $\theta$  paraméter ismeretében megegyezik  $\theta$ -val (nyilván, hiszen éppen így határoztuk meg a paraméter jelentését). Rögtön felírhatjuk az írásra vonatkozó valószínűséget is:

$$p(y_i = 0 \mid \theta) = 1 - \theta.$$

<sup>1</sup> Ha ez a fajta felírás esetleg nem volna ismerős, javasoljuk a Valószínűségszámítás c. tárgyban tanultak átfutását.

A két mennyiséget egyben is kezelhetjük, így megkapjuk  $y_i$  eloszlását<sup>2</sup>:

$$p(y_i | \theta) = \theta^{y_i} (1 - \theta)^{1-y_i}, \quad (1.1)$$

amit *Bernoulli-eloszlásnak* nevezünk. Mivel  $\theta$  becsléséhez az összes megfigyelést fel szeretnénk használni (jelöljük egyben  $\mathbf{y}$ -nal), felírjuk ezek együttes eloszlását:

$$\begin{aligned} p(\mathbf{y} | \theta) &= \prod_i p(y_i | \theta) \\ &= \prod_i \theta^{y_i} (1 - \theta)^{1-y_i} \\ &= \theta^{\sum_i y_i} (1 - \theta)^{\sum_i 1-y_i} \\ &:= \theta^h (1 - \theta)^t, \end{aligned} \quad (1.2)$$

ahol feltettük, hogy a megfigyelések függetlenek és azonos eloszlásúak<sup>3</sup> – azaz helyesen jártunk el, amikor az egyes megfigyelésekre vonatkozó eloszlásokat összeszoroztuk. A  $h$ -val jelölt mennyiség egyszerűen a dobott fejek száma,  $t$  pedig az írásoké.

### 1.1.1 Maximum likelihood (ML) becslés

Hogyan állapíthatjuk meg ezek alapján  $\theta$  értékét? Roppant egyszerűen – olyan  $\theta$ -t keresünk, ami a legjobban megmagyarázza a megfigyeléseket, más szóval olyat, ami a maximalizálja a  $p(\mathbf{y} | \theta)$  valószínűséget<sup>4</sup>. Ezt a függvényt, sőt, általában a

$$p(\text{megfigyelések} | \text{paraméterek})$$

alakú függvényeket *likelihood*-nak nevezzük, ha a paraméter függvényeként gondolunk rájuk. A likelihood-ra és az ML becslésre az 1.1. ábrán láthatunk példát.

A likelihood maximalizálásához először negatív logaritmust veszünk, mivel ez gyakran kényelmesebbé teszi a feladat megoldását<sup>5</sup>; a logaritmus szigorúan monoton, úgyhogy nem csaltunk, a szélsőérték helyek éppen ugyanott lesznek, mint az eredeti függvénynél. A negatív előjel miatt maximalizálás helyett minimalizálási feladatot oldunk meg:

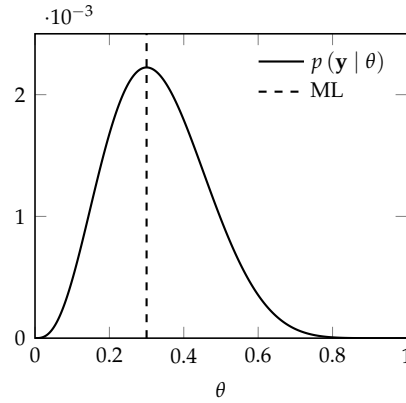
$$L(\theta) := -\ln p(\mathbf{y} | \theta) = -h \ln \theta - t \ln(1 - \theta). \quad (1.3)$$

<sup>2</sup> Vegyük észre, hogy ha  $y_i$  helyére 1-et, vagy 0-t helyettesítünk, éppen az előbbi formulákat kapjuk vissza.

<sup>3</sup> Angol szakkifejezéssel *iid*, „independent and identically distributed”.

<sup>4</sup> Így gondolkodunk: „Ezzel a  $\theta$ -val valószínűleg ilyesmi megfigyeléseket látnánk, míg egy másikkal ez kevésbé valószínű”.

<sup>5</sup> Ez nem csak a számolások egyszerűsödését jelenti, hanem numerikusan is sokkal kedvezőbb – számítógépen sok apró valószínűség összeszorozásakor könnyen beleütközhetünk a számábrázolás korlátaiba, míg logaritmusokkal számolva csupán összegezni kell, ahol ilyesmi nemigen fordul elő.



1.1. ábra. Likelihood függvény és maximum likelihood becslés  $h = 3$  és  $t = 7$  mellett. A  $\theta$  paraméter legvalószínűbb értéke  $\frac{3}{10}$ -nek adódik.

A negatív log-likelihoodra gondolhatunk egyfajta veszteségfüggvényként (*loss*) is, hiszen minél kisebb az értéke, annál „jobb” a becslésünk  $\theta$ -ra. Az (1.3) egyenlet jobb oldalán látható veszteségfüggvényt *keresztentrópiának* nevezzük. Minimalizálásához a jól ismert módszert használjuk: deriválunk, majd a deriváltat zérusra állítva megvizsgáljuk a lehetséges szélsőérték helyeket. Lépésről lépésre

$$\frac{\partial L}{\partial \theta} = -\frac{h}{\theta} + \frac{t}{1-\theta} = 0,$$

amelyet átrendezve a következő – nem túl meglepő – megállapításra jutunk:

$$\theta = \frac{h}{h+t}.$$

A  $\theta$ -ra vonatkozó maximum likelihood becslés tehát a fejek aránya az összes dobáson belül.

### 1.1.2 A Bayes-tétel

A maximum likelihood becslés jól működik, ha sok megfigyeléssel dolgozunk, kevés adatnál viszont már nem annyira. Meggondolatlanság volna például egyetlen írás dobásából arra következtetni, hogy az érme 1 valószínűséggel írást dob. A valóságban általában van egyfajta előzetes hiedelmünk arról, hogy egy érme milyen valószínűségek mentén „működik”, milyen  $\theta$  értékeket tartunk hihetőnek.

Bár formálisan nem lesz nagy a változás, valójában itt a valószínűségeknek egy alapvetően más értelmezésére fogunk átváltani (persze a valószínűség továbbra is egy egzakt matematikai definícióval bír, de nem mindegy, hogy hogyan gondolunk rá). Szokatlan lehet például „hiedelmekről” hallani, szemben az előző szakasz kombinatorikus,  $\frac{\text{hasznos eset}}{\text{összes eset}}$ -jellegű  $\theta$ -ja után.

A hétköznapiokban a valószínűség tényleg inkább egyfajta szubjektív hiedelmet tükröz. Gondoljunk például arra a mondatra, hogy „holnap 80% eséllyel esni fog az eső”. Ez a kombinatorikus-frekventista értelmezésben valami olyasmit jelentene, hogy Dr. Strange módjára „ötször előre mentünk az időben, és ebből négyszer esett”. A hiedelmek formális kezelésére ad módot a bayesi valószínűségelmélet, amellyel az induktív következtetés, tanulás is megragadható.

Az előzetes hiedelmeinket a *prior* írja le, amely egy eloszlás  $\theta$  felett, még mielőtt az adatokat láttuk volna:

$$p(\theta).$$

Arra, hogy az előzetes hiedelmeinket (prior) és a megfigyeléseinket (likelihood) szintetizáljuk, a Bayes-tétel ad lehetőséget:

$$p(\theta | \mathbf{y}) = \frac{p(\mathbf{y} | \theta) p(\theta)}{p(\mathbf{y})}.$$

Észrevehetjük, hogy a nevezőben nem szerepel  $\theta$ , tehát az előbbi egyenletet arányosság erejéig felírva<sup>6</sup>

$$\underbrace{p(\theta | \mathbf{y})}_{\text{poszterior}} \propto \underbrace{p(\mathbf{y} | \theta)}_{\text{likelihood}} \underbrace{p(\theta)}_{\text{prior}}. \quad (1.4)$$

Az egyenlet bal oldalán látható *poszterior* képviseli a „frissített” hiedelmeinket  $\theta$ -ra vonatkozóan, amelybe tehát már a megfigyelt adatok is beépülnek.

Egyes kutatások szerint ez játszódik ez az emberi agyban is<sup>7</sup>: a világra vonatkozó előzetes hiedelmeinket jelentő *a priori* eloszlást az idegsejtjeink hálózatával alkalmasan reprezentáljuk, majd az érzékszerveinken át beérkező adatokkal frissítjük; azt pedig a Bayes-tétel mondja meg, hogy ennek hogyan kell történnie.

<sup>6</sup> A „ $\propto$ ” szimbólum jelentése: konstans szorzótényezőtől eltekintve egyenlő.

<sup>7</sup> L. Aitchison és M. Lengyel, “THE HAMILTONIAN BRAIN: EFFICIENT PROBABILISTIC INFERENCE WITH EXCITATORY-INHIBITORY NEURAL CIRCUIT DYNAMICS”, *PLoS Comput Biol*, 12. évf., 12. sz., e1005186., 2016.

A következő kérdés a prior megválasztása. Ezt a nagyon érdekes és mély filozófiai problémát<sup>8</sup> első körben kikerüljük, és olyan priort választunk, amellyel könnyen tudunk számolni. Az (1.4) egyenletben láttuk, hogy szoroznunk kell, tehát válasszunk olyan mennyiséget, ahol a priorban az (1.2) likelihoodhoz hasonlóan  $\theta$  és  $1 - \theta$  hatványai szerepelnek, azaz legyen a priorunk béta-eloszlású  $\alpha$  és  $\beta$  hiperparaméterekkel<sup>9</sup>:

$$p(\theta | \alpha, \beta) = \text{Beta}(\theta | \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \cdot \theta^{\alpha-1}(1 - \theta)^{\beta-1},$$

ahol az első, bonyolultnak tűnő tag csupán a normalizációt szolgálja (a görbe alatti terület így lesz 1, amit egy eloszlástól elvárunk).

Most már mindent ismerünk a poszterior kiszámolásához. Az (1.4) egyenlet alapján

$$\begin{aligned} p(\theta | \mathbf{y}, \alpha, \beta) &\propto p(\mathbf{y} | \theta) \cdot p(\theta | \alpha, \beta) \\ &= \theta^h (1 - \theta)^t \cdot \theta^{\alpha-1} (1 - \theta)^{\beta-1} \cdot \text{const.} \\ &= \theta^{h+\alpha-1} (1 - \theta)^{t+\beta-1} \cdot \text{const.} \\ &\propto \text{Beta}(\theta | \alpha + h, \beta + t), \end{aligned} \tag{1.5}$$

azaz arra jutottunk, hogy a poszteriorunk is béta-eloszlású<sup>10</sup>! Az efféle priorokat – nevezetesen, ahol a poszterior is ugyanezen eloszlást követi, csak más paraméterekkel – *konjugált prioroknak* nevezzük, azaz például mondhatjuk, hogy a béta-eloszlás konjugált prior a Bernoulli-eloszlásra nézve. A frissítés ezekben az esetekben általában egyszerű, nálunk például

$$\text{Beta}(\theta | \alpha, \beta) \rightsquigarrow \text{Beta}(\theta | \alpha + h, \beta + t),$$

ahol érdekes felfedezést tehetünk: az előzetes hiedelmeinket az  $\alpha$  és  $\beta$  „virtuális esetszámok” kódolják, amelyekhez hozzáadódnak a valóban megfigyelt esetszámok. Ezt az eljárást mutatja be az 1.1 algoritmus.

```
function posterior_Beta_Bernoulli(y; α=1, β=1)
    h = sum(y)
    t = length(y) - h
    return Beta(α+h, β+t)
end
```

<sup>8</sup> S. Rathmanner és M. Hutter, “A Philosophical Treatise of Universal Induction”, *Entropy*, 13. évf., 6. sz., 1076–1136. old., 2011.

<sup>9</sup> A hiperparaméterek olyan paraméterek, amelyeket mindvégig változtatlanul hagyunk; gondolhatunk rá úgy, mint a gépi tanulási módszerünk bemenetére, amelyekkel a tanulást szabályozzuk.

<sup>10</sup> Itt valójában kihasználtuk azt, hogy az ilyen alakú kifejezésekhez tartozó normalizációs konstans ismert. De nem is kell foglalkoznunk vele: az egyenlet elején egy eloszlás szerepel, a végén pedig azt kaptuk, hogy arányosság erejéig egyenlő egy béta-eloszlással; de mivel ezek mind eloszlások, ez csak egyenlőséggel teljesülhet.

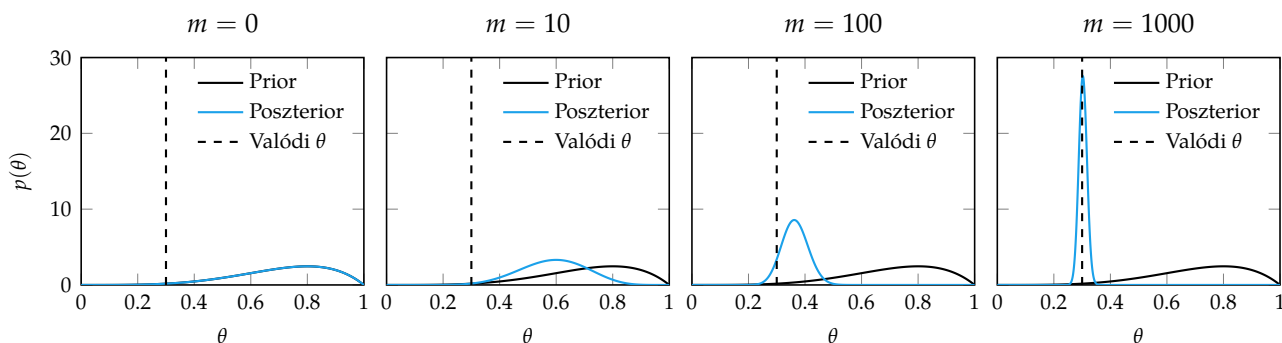
1.1. algoritmus. A poszterior eloszlás kiszámolása az  $\mathbf{y}$  megfigyelések és az  $\alpha$ ,  $\beta$  hiperparaméterek mellett.

Vegyük észre, hogy most sokkal „okosabb” objektumot kaptunk:  $\theta$  poszterior eloszlását, ami  $\theta$  bizonytalanságáról is hordoz információt. Ha ezzel nem szeretnénk foglalkozni, és az ML becsléshez hasonlóan csak egy pontbecslés érdekel minket, könnyen gyárthatunk ilyen, ha a likelihood helyett az (1.5) poszteriort maximalizáljuk (*maximum a posteriori*, MAP); ez már nem csak a megfigyeléseket, hanem az előzetes hiedelmeinket is tartalmazni fogja. A korábbi eljárást megismételve<sup>11</sup>

$$\theta_{\text{MAP}} = \frac{\alpha + h - 1}{\alpha + h + \beta + t - 2}.$$

<sup>11</sup> Próbáljuk meg önállóan levezetni: vegyük az (1.5) kifejezés negatív logaritmusát, a deriváltat állítsuk zérusra és oldjuk meg az egyenletet.

Az 1.2 ábrán a megfigyelések számának hatását láthatjuk a prior és poszterior viszonyára. Kevés adatnál a poszteriort a prior dominálja, a modell sokkal inkább az előzetes hiedelmekre hagyatkozik, mint a megfigyelésekre (éppen ezért fontos a jó prior választása!). Sok adatnál a prior kevés szerephez jut, a poszteriort az adatok dominálják. Látjuk, hogy a bayesi következtetés egyfajta egyensúlyt képvisel a prior és az adat között – de ne feledjük, hogy emögött egzakt matematikai megfontolások álltak; úgy is mondhatjuk, hogy bizonytalan környezetben így „kell” helyesen következtetni.



Mindenesetre túlzás volna azt állítani, hogy a bayesi valószínűségelmélet – vagy legalábbis, ahogyan mi használtuk – az egész emberi induktív következtetést egy csapásra formalizáltuk volna. A gépi tanulásban a prior megválasztása rendszerint csak egy technikai eszköz, amellyel a paramétereket a kívánt régió-

1.2. ábra. Prior és poszterior eloszlások  $m$  megfigyelés esetén  $\alpha = 5$ ,  $\beta = 2$  priorral. A szemléltetés érdekében szándékosan „rossz” priort választottunk.

ba kényszeríthetjük, nem pedig a háttértudás pontos megragadásának eszköze (emlékezzünk vissza, hogy a könnyű kezelhetőség volt az elsődleges szempont).

### 1.1.3 Teljesen bayesi következtetés

A korábbiakban még mindig nem használtuk ki a bayesi megközelítés teljes erejét. Általában nem is igazán  $\theta$  érdekel minket, hanem egy  $y_{új}$  értéket szeretnénk jósolni, azaz a kérdésünk

Az előzetes hiedelmeket és az adatokat figyelembe véve mekkora a valószínűsége annak, hogy a következő alkalommal pl. fejet dobunk?

Az erre vonatkozó *prediktív eloszlást* úgy kapjuk, hogy – a teljes valószínűség tételét felhasználva –  $\theta$  minden lehetséges értéke szerint összegzünk, megsúlyozva azok poszterior valószínűségével:

$$\begin{aligned} p(y_{új} = 1 \mid \mathbf{y}, \alpha, \beta) &= \int p(y_{új} = 1 \mid \theta) p(\theta \mid \mathbf{y}, \alpha, \beta) d\theta \\ &= \int \theta \cdot \text{Beta}(\theta \mid \alpha + h, \beta + t) d\theta \\ &= \mathbb{E}_{\text{Beta}(\theta \mid \alpha + h, \beta + t)}[\theta] \\ &= \frac{\alpha + h}{\alpha + h + \beta + t} \end{aligned}$$

Ezt az eljárást nevezzük *bayesi modellátlagolásnak* is; a prediktív eloszlás kiszámításához minden lehetséges modellt figyelembe vettünk, „jókat” és „rosszakat” egyaránt, ám a „rossz” (valószínűtlen)  $\theta$ -k alacsony súllyal szerepelnek. Az utolsó egyenlethez egyszerűen kikerestük a béta-eloszlás várható értékét korábbi jegyzeteinkből<sup>12</sup>.

<sup>12</sup> Vagy Wikipediáról.

### 1.1.4 MAP becslés nem konjugált priorokkal

A maximum a posteriori becslésnél erős feltevéssel éltünk a prior alakját illetően, nevezetesen a könnyű számolhatóságra törekedtünk. Általánosabb esetben a MAP becslés bonyolultabb, mert könnyen olyan egyenletre jutunk, amelyet nem tudunk zárt formában megoldani. Nézzünk erre egy példát. Legyen a priorunk ún. Kumaraswamy-eloszlású:

$$p(\theta \mid \alpha, \beta) = \alpha \cdot \beta \cdot \theta^{\alpha-1} \cdot (1 - \theta)^{\beta-1},$$

a likelihood-unk változatlanul

$$p(\mathbf{y} | \theta) = \theta^h (1 - \theta)^t,$$

a poszteriorunk pedig a kettő szorzata:

$$\begin{aligned} p(\theta | \mathbf{y}, \alpha, \beta) &\propto p(\mathbf{y} | \theta) \cdot p(\theta | \alpha, \beta) \\ &= \alpha \cdot \beta \cdot \theta^{\alpha-1} \cdot (1 - \theta)^{\beta-1} \theta^h (1 - \theta)^t \end{aligned}$$

A maximalizáláshoz ismét vesszük a negatív logaritmust, amelyre veszteségfüggvényként gondolunk:

$$L(\theta) = -\ln \alpha - \ln \beta - (\alpha - 1 + h) \ln \theta - t \ln(1 - \theta) - (\beta - 1) \ln(1 - \theta^\alpha)$$

Majd vesszük a deriváltat:

$$\frac{\partial L}{\partial \theta} = -\frac{\alpha - 1 + h}{\theta} + \frac{t}{1 - \theta} + (\beta - 1) \frac{\alpha \theta^{\alpha-1}}{1 - \theta^\alpha}.$$

Sajnos ennek az kifejezésnek a zérushelyét nem tudjuk zárt formában felírni, így a Newton–Raphson módszerhez folyamodunk (A függelék), és numerikusan keressük a megoldást. Ehhez szükség lesz a kifejezés deriváltjára:

$$\frac{\partial^2 L}{\partial \theta^2} = \frac{\alpha - 1 + h}{\theta^2} + \frac{t}{(1 - \theta)^2} + (\beta - 1) \alpha \cdot \left[ \frac{(\alpha - 1) \theta^{\alpha-2}}{1 - \theta^\alpha} + \frac{\alpha \theta^{2(\alpha-1)}}{(1 - \theta^\alpha)^2} \right],$$

majd az A.1. algoritmussal kiszámoljuk a zérushelyet (1.3. ábra).

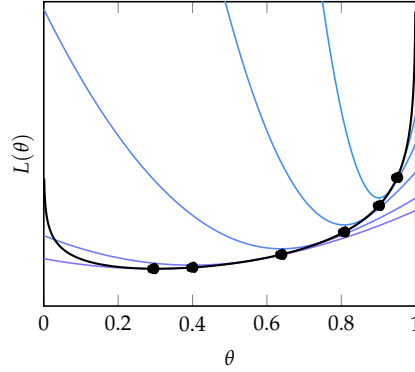
## 1.2 Naïv bayesi osztályozók

Készen állunk, hogy az első, gyakorlatban is használt gépi tanulási modellünket összeállítsuk. A naïv bayesi osztályozókat többosztályos feladatokban használjuk, azaz míg a pénzérménél két lehetséges kimenetel – ha úgy tetszik, két osztály – volt, most  $y_i \in \{1, 2, \dots, K\}$  értékű lehet<sup>13</sup>. Ezeket az  $y_i$  változókat mostantól *osztálycímkéknek* (*label*) fogjuk nevezni. A korábbiaktól eltérően a modellünknek bemenete is lesz: ezek a minták, amelyeket osztályokba szeretnénk sorolni, és  $\mathbf{x}_i \in \mathbb{R}^D$  valós vektorok formáját öltik<sup>14</sup>. Az *ellenőrzött* vagy *felügyelt* gépi tanulási módszerek a következőképpen működnek:

<sup>13</sup> Klasszikus példa a kézzel írott számjegyek felismerése.

<sup>14</sup> Ez a felállítás, nevezetesen, hogy a mintáink sokdimenziós valós vektorok, nagyon gyakori a gépi tanulásban. A legtöbb adatot (képek, szöveges adatok, orvosi leletek stb.) ilyen formában kapjuk meg, vagy ilyen alakra hozzuk. Ebben a jegyzetben is szinte kizárólag is vektros adatokkal fogunk dolgozni.





1.3. ábra. A veszteségfüggvény minimalizálása Newton–Raphson módszerrel. A  $\theta_0 = 0.95$  pontból indulva néhány lépésben elérjük a minimumot; az egyes lépésekben a veszteségfüggvény (fekete) másodrendű becslései kék paraboláként ábrázolódnak.

1. Tanítási fázis (*training*): az algoritmusnak ismert  $(\mathbf{x}_i, y_i)$  párokat mutatunk, az pedig megpróbálja valamiképpen felismerni az összefüggéseket, mintázatokat keresni az adatokban,
2. Tesztelés fázis (*testing*): az így nyert tudást felhasználjuk egy új, ismeretlen címkéjű  $\mathbf{x}$  minta besorolására (megfelelő  $y$  jóslására).

Nézzük, hogy történhet ez a gyakorlatban. Annak a valószínűségét, hogy az  $\mathbf{x}_i$  mintát a  $k$ . osztályba soroljuk, a Bayes-tétel felhasználásával a következőképpen írhatjuk:

$$p(y_i = k | \mathbf{x}_i) = \frac{p(\mathbf{x}_i | y_i = k) p(y_i = k)}{p(\mathbf{x}_i)} \propto p(\mathbf{x}_i | y_i = k) p(y_i = k). \quad (1.6)$$

Minden  $\mathbf{x}_i$  mintára a legvalószínűbb osztályt keressük<sup>15</sup>. Mivel a nevező független  $k$ -tól (mindegyik osztályra ugyanaz), ezt akár el is hagyhatjuk, a döntésünket nem fogja befolyásolni. A számlálóban szereplő mennyiségek közül  $p(y_i = k)$ -t könnyen becsülhetjük a tanító adathalmazban a  $k$ . osztályba eső minták arányával. Nehezebb a helyzet a  $p(\mathbf{x}_i | y_i = k)$  valószínűséggel, ahol a következő feltételezésekkel élünk:

- Az egyes koordináták függetlenek egymástól<sup>16</sup>, azaz

$$p(\mathbf{x}_i | y_i = k) \approx p(x_i^1 | y_i = k) \cdot p(x_i^2 | y_i = k) \cdot p(x_i^3 | y_i = k) \cdots,$$

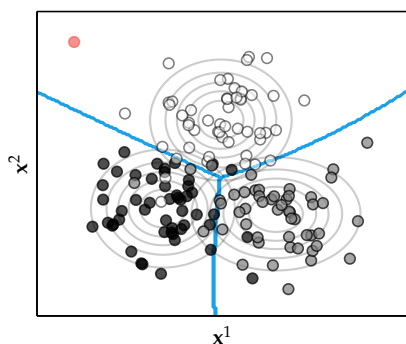
<sup>15</sup> Azaz itt is egy MAP becslésre fogjuk kihozni a dolgot.

<sup>16</sup> Innen származik a módszer család neve is, hiszen ez egy igen naiv feltevés.

- A koordináták minden osztály esetében normális eloszlást követnek  $\mu_k^l$  várható értékkel és  $\sigma_k^{2l}$  szórással:

$$p(x_i^l | y_i = k) = \mathcal{N}(x_i^l | \mu_k^l, \sigma_k^{2l}).$$

A várható értékekre és szórássokra gondolhatunk úgy, mint a  $k$ . osztályba tartozó mintahalmaz „középpontjára” és „kiterjedésére” az egyes koordinátatengelyek mentén (1.4 ábra). Ezek közvetlenül a tanító adatokból kiszámolhatók, amivel a tanulási fázist le is zártuk; a tesztelés során elég az (1.6) egyenlet alapján kiszámolni a poszteriori és az új mintát a legmagasabb értéket kapott osztályba sorolni (lásd az 1.2 algoritmust).



1.4. ábra. Naiv bayesi osztályozó három osztályra. Az egyes osztályokhoz tartozó mintákat pontok, a tanult normális eloszlások szintvonalait fekete ellipszisek ábrázolják. A kék színnel jelölt döntési felület az egyes osztályokhoz tartozó régiókat határolja; ha a tesztelési fázisban egy új mintát szeretnénk osztályba sorolni, a besorolást az dönti el, hogy ezen határ melyik oldalára esik. A piros mintát így a „fehér” osztályba sorolnánk.

A naiv bayesi osztályozók könnyen általánosíthatók másféle eloszlásokra is<sup>17</sup>; meglepően jól működnek kategorikus adatoknál, illetve kicsi-közepes adatmennyiség esetén. Bár a függetlenségi feltevés miatt jól skálázhatók, nagy dimenziójú, komplex, nagy mennyiségű adatnál (pl. manapság a mélytanulás alkalmazási területein) prediktív teljesítményük általában elmarad az újabb algoritmusokétól.

<sup>17</sup> Multinomiális eloszlással főleg szövegosztályozási feladatokban használatosak; ekkor a minták nagy dimenziós vektorok, amelyek meghatározott kulcsszavak előfordulásának számát kódolják. A módszer naivitása abban nyilvánul meg, hogy az egyes szavakat függetlennek tekinti, ami összefüggő szövegeknel nyilván nem igaz. Ennek ellenére a megközelítés szép sikereket ért el a spamszűrés területén.

```

struct NaiveBayes
    K    # osztályok száma
    D    # dimenziók száma
     $\mu$    #  $p(x|y)$  várható értékek
     $\sigma$  #  $p(x|y)$  szórások
    py   #  $p(y)$  osztályvalószínűségek

    NaiveBayes(K, D) =
        new(K, D, zeros(K,D), ones(K,D), zeros(K))
end

function train!(m::NaiveBayes,X,y)
    for k in 1:m.K
        class_k = y.==k
        m. $\mu$ [k,:] .= mean(X[class_k,:],dims=1)[:]
        m. $\sigma$ [k,:] .= std(X[class_k,:],dims=1)[:]
        m.py[k] = mean(class_k)
    end
end

function predict(m::NaiveBayes,X)
    N = size(X,1)
    lnpxy = zeros(N,m.K)
    for i in 1:N, k in 1:m.K
        pxy = MultivariateNormal(m. $\mu$ [k,:],m. $\sigma$ [k,:])
        lnpxy[i,k] = logpdf(pxy,X[i,:])
    end
    return argmax.(eachrow(lnpxy .+ log.(m.py')))
end

```

1.2. algoritmus. Naiv bayesi algoritmus többosztályos osztályozásra. Figyeljük meg, hogy a predikció során valószínűségek helyett azok logaritmusával számolunk, így elkerülhetjük a numerikus problémákat (pl. számábrázolás). Az adatvektorokat az  $X$  mátrix tartalmazza (sorokként),  $y$  pedig a címkék vektora.



## 2. fejezet

# Lineáris regresszió

Ebben a fejezetben a gépi tanulás, adatelemzés egyik alapeszközével, a lineáris regresszióval foglalkozunk. Az itt előkerülő ötletekre sok más algoritmusnál, például a neurális hálózatoknál is vissza fogunk utalni. A következő szakaszokban az a célunk, hogy az 1. fejezetben megismert eszköztárat alkalmazva valószínűségelméleti keretbe foglaljuk a módszer családot, valamint gyakorlatban is használható algoritmusokat adjunk.

### 2.1 Valószínűségi modell

Az 1.2 szakaszhoz hasonlóan olyan adatokkal fogunk dolgozni, amelyek párokban érkeznek: bemenetek és hozzájuk tartozó kimenetek. A korábbiakkal ellentétben azonban a feladat nem osztályozás, hanem *regresszió*, azaz a kimenetek folytonos értékeket vehetnek fel<sup>1</sup>:

$$\begin{aligned}\mathbf{x}_i &\in \mathbb{R}^D, \\ y_i &\in \mathbb{R}.\end{aligned}$$

Lineáris regressziónál a kettő között lineáris függvénykapcsolatot teszünk fel:

$$y_i \approx \mathbf{w}^\top \mathbf{x}_i + b, \tag{2.1}$$

<sup>1</sup> Ilyen feladat például, ha használt autók árának megbecsülésére szeretnénk gépi tanulási modellt felállítani. Az  $\mathbf{x}_i$  bemenet az  $i$ . használt autó leírását tartalmazza: súly, gyártási év, kilométeróra állása stb., a hozzá tartozó  $y_i$  kimenet pedig az autó ára. A célunk a kettő közötti összefüggés megállapítása, amelyet további, korábban nem látott autók árának megbecsülésére használhatunk fel.

ahol  $\mathbf{w}$ -t *súlyvektornak*,  $b$ -t *eltolásnak* nevezzük; ezeket szeretnénk az adatokból valamiképpen megállapítani<sup>2</sup>. Ha megvagyunk, a (2.1) egyenlet felhasználásával, egyszerű behelyettesítéssel már egy tetszőleges új  $\mathbf{x}$  mintához tudunk  $y$ -t mondani.

A későbbi bonyoldalmak elkerülése érdekében alakítsuk át az  $\mathbf{x}_i$  bemeneteket. Vegyünk egy  $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^{D+1}$  függvényt, amely mindössze annyit tesz, hogy egy  $\mathbf{x}_i$ -hez hozzáfűz egy fix 1-es értéket; mostantól ezzel a módosított  $\phi(\mathbf{x})$  bemenettel fogunk számolni. Ahhoz, hogy a skaláris szorzat értelmes legyen, hasonlóképpen  $\mathbf{w}$ -t is bővítjük. Az extra elem fogja  $b$  szerepét játszani:

$$\begin{bmatrix} \mathbf{w}^0 = b \\ \vdots \\ \mathbf{w} \\ \vdots \end{bmatrix}^\top \underbrace{\begin{bmatrix} 1 \\ \vdots \\ \mathbf{x}_i \\ \vdots \end{bmatrix}}_{\phi(\mathbf{x}_i)} \quad (2.2)$$

Mostantól tehát  $b$ -vel nem kell külön foglalkoznunk, és a feladat a következő formát ölti:

$$y_i \approx \mathbf{w}^\top \phi(\mathbf{x}_i). \quad (2.3)$$

Ahhoz, hogy a korábbi eszközeinket használni tudjuk, át kell térnünk a valószínűségek nyelvére. Tegyük fel, hogy a (2.3) egyenlet valamekkora  $\varepsilon$  zajtól eltekintve teljesül<sup>3</sup>, a zaj pedig 0 várható értékű,  $\beta^{-1}$  varianciájú normális eloszlást követ:

$$y_i = \mathbf{w}^\top \phi(\mathbf{x}_i) + \varepsilon, \\ \varepsilon \sim \mathcal{N}(\varepsilon \mid 0, \beta^{-1}).$$

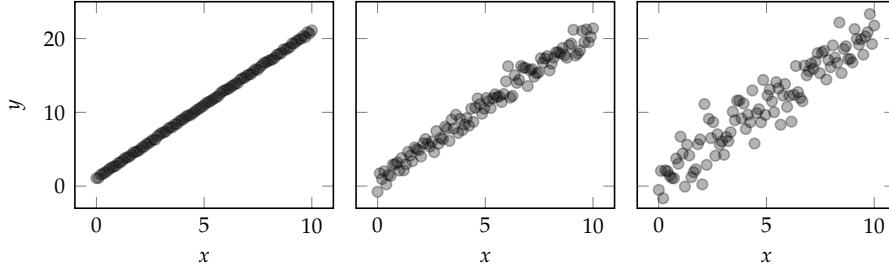
Additív zajról lévén szó,  $y_i$  eloszlása is hasonlóan alakul, csupán a várható értéket kell eltolni. Az összefüggés tehát valószínűségi formában így írható<sup>4</sup>:

$$p(y_i \mid \mathbf{x}_i, \mathbf{w}, \beta) = \mathcal{N}(y_i \mid \mathbf{w}^\top \phi(\mathbf{x}_i), \beta^{-1}) \\ = \sqrt{\frac{\beta}{2\pi}} e^{-\frac{\beta}{2} (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2}.$$

<sup>2</sup> Vegyük észre, hogy ez nem más, mint az  $y = ax + b$  egyenes egyenletének többdimenziós analógja; másképpen, itt egy hipersík normálvektorát és eltolását keressük.

<sup>3</sup> Erre gondolhatunk úgy, hogy  $y_i$  értékét nem tudjuk pontosan megismerni, például mérési hiba miatt.

<sup>4</sup> Emlékezzünk vissza a normális eloszlás képletére.



2.1. ábra. Lineáris függvénykapcsolat egy dimenzióban, különböző mértékű additív zajjal terhelve. A célunk, hogy megtaláljuk az adatokra legjobban illeszkedő egyenest.

Végül, mivel nem egyes mintákat, hanem az összeset egyszerre szeretnénk tekinteni (jelölje  $\mathbf{X}$  és  $\mathbf{y}$ ), az előző fejezetben használt függetlenségi feltevésünket megtartva felírhatjuk az együttes eloszlást (likelihoodot):

$$p(\mathbf{y} | \mathbf{X}, \mathbf{w}, \beta) = \prod_i \mathcal{N}(y_i | \mathbf{w}^\top \phi(\mathbf{x}_i), \beta). \quad (2.4)$$

## 2.2 Maximum likelihood megoldás

Készen állunk arra, hogy maximum likelihood megoldást keressünk, azaz a (2.4) függvényt  $\mathbf{w}$  szerint maximalizáljuk. Az előző fejezethez hasonlóan negatív logaritmust véve

$$-\ln p(\mathbf{y} | \mathbf{X}, \mathbf{w}, \beta) = \underbrace{\frac{\beta}{2} \sum_i (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2}_{L(\mathbf{w})} + \text{const.}, \quad (2.5)$$

ahol a  $\mathbf{w}$ -t nem tartalmazó tagokat egy konstansba gyűjtöttük, és így megkaptuk az  $L(\mathbf{w})$  veszteségfüggvényt, amelynek minimumát keressük. Jobban megnézve ez a veszteségfüggvény nem más, mint a korábbi tanulmányainkból ismert *négyzetes hiba*<sup>5</sup>. A kifejezések egyszerűségének érdekében legyen mostantól

$$\phi_i := \phi(\mathbf{x}_i).$$

Az  $L(\mathbf{w})$  veszteség minimalizálásához az ismert módszerhez folyamodunk: gradiensét veszünk, ezt zérusra állítva megvizsgáljuk a lehetséges szélsőérték helyeket:

$$\nabla_{\mathbf{w}} L = -\beta \sum_i (y_i - \mathbf{w}^\top \phi_i) \phi_i = 0.$$

<sup>5</sup> Szemléletesen, az  $y_i$  valóság és a  $\mathbf{w}^\top \phi(\mathbf{x}_i)$  becslések közötti eltérést szeretnénk minimalizálni.

Az egyenlet megoldásához szükséges némi mátrix-barkácsolás<sup>6</sup>. Vegyünk egy  $\Phi$  mátrixot, ami soraiban a  $\phi_i$  transzformált mintákat tartalmazza; a mátrix-szorzás definíciójára visszaemlékezve észrevehetjük, hogy

$$\underbrace{\begin{bmatrix} \vdots & \vdots & \vdots \\ \phi_1 & \phi_2 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix}}_{:=\Phi^\top} \cdot \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix}}_{\mathbf{y}} = \sum_i y_i \phi_i,$$

amire szükségünk van a gradiens kiszámításához. Hasonlóan adódik, hogy

$$\sum_i \mathbf{w}^\top \phi_i \cdot \phi_i = \Phi^\top \Phi \mathbf{w},$$

amivel a gradiens praktikusabb alakra hozható:

$$\nabla_{\mathbf{w}} L = -\beta \left( \Phi^\top \mathbf{y} - \Phi^\top \Phi \mathbf{w} \right) = 0.$$

Végül az egyenletet átrendezve megkapjuk  $\mathbf{w}$ -t:

$$\mathbf{w} = \left( \Phi^\top \Phi \right)^{-1} \Phi^\top \mathbf{y}, \quad (2.6)$$

amit a predikcióhoz a (2.3) egyenlet mátrixos alakjába helyettesítünk:

$$\mathbf{y}_{\text{új}} = \Phi_{\text{új}} \mathbf{w}.$$

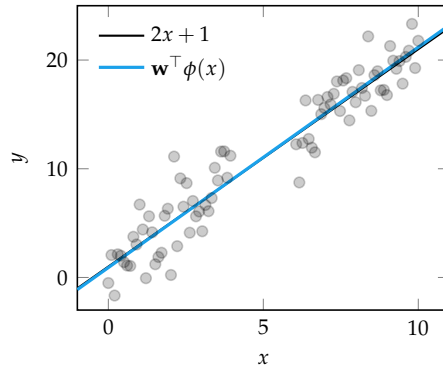
<sup>6</sup> A legtöbb tudományos kódban, gépi tanulási könyvtárban az efféle műveletek mátrix-szorzásokként implementálják. Erre a párhuzamosítás, a számítások hatékonysága miatt van szükség: a grafikus vagy tenzorprocesszorok lényegében ilyen műveletekre vannak optimalizálva. Őva intünk mindenkit attól, hogy for ciklusokat használjon ilyen célra, ahogy egyébként az egyenlet sugallaná!

```
struct BasicLinearRegression
    w # modellsúlyok

    BasicLinearRegression(D) = new(zeros(D))
end
function train!(m::BasicLinearRegression, Φ, y)
    m.w .= (Φ'Φ) \ (Φ'y)
end
function predict(m::BasicLinearRegression, Φ_test)
    y_test = Φ_test * m.w
    return y_test
end
```

2.1. algoritmus. Lineáris regresszió. Vegyük észre a „backslash” operátor használatát a mátrix-invertálás helyett!





2.2. ábra. Lineáris regresszió egy dimenzióban. A valódi függvénykapcsolatot fekete egyenes, a mintákat fekete pontok, az illesztett egyenest kék vonal ábrázolja.

Egy kis kitérő. Gépi tanulásban gyakran találkozunk mátrixok invertálásával, ami  $\mathcal{O}(n^3)$  költségű művelet. Az inverzre azonban szinte sosem önmagában van szükségünk, hanem másik mátrixszal vagy vektorral megszorozva. Ezekben az esetekben célszerűbb lineáris egyenletrendszerként gondolni a problémára:

$$\mathbf{A}^{-1}\mathbf{b} = \mathbf{x} \rightsquigarrow \mathbf{Ax} = \mathbf{b},$$

amelyre nagyon hatékony algoritmusaink vannak (minden valamirevaló lineáris algebra csomag tartalmaz ilyen műveleteket). A számításigényes invertálás és az inverz memóriában való tárolása helyett így gyorsabb kódot kapunk.

### 2.3 Bázisfüggvények

Mi a helyzet akkor, ha a függvénykapcsolatunk nem lineáris? Továbbra is egydimenziós mintáknál maradván szeretnénk például polinomokat illeszteni. Egy legfeljebb  $D$ -edfokú polinom általánosságban így írható fel:

$$\sum_{d=0}^D \mathbf{w}^{(d)} x_i^d = \mathbf{w}^{(0)} + \mathbf{w}^{(1)} x_i + \mathbf{w}^{(2)} x_i^2 + \dots = \mathbf{w}^\top \phi(x_i),$$

ahol az együtthatókat  $\mathbf{w}$  elemei képviselik<sup>7</sup>, a  $\phi$  függvény pedig a következőkép-

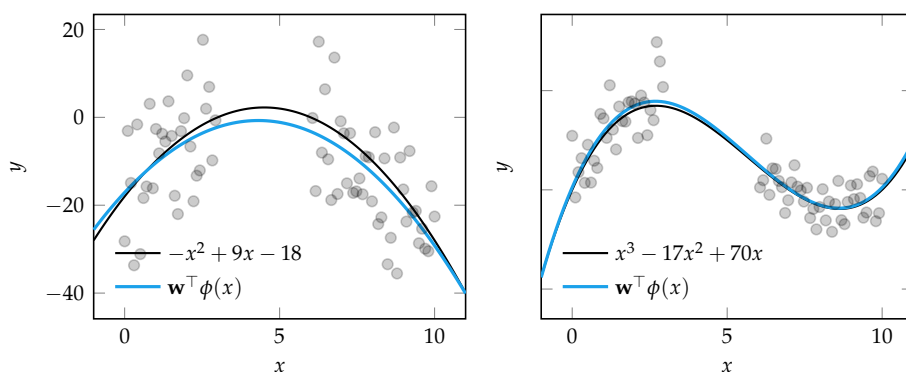
<sup>7</sup> Kivételesen zárójellel jelölve, hogy a hatványozástól meg tudjuk különböztetni.

pen alakítja át a mintáinkat:

$$\phi(x_i) = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \\ \vdots \end{bmatrix}.$$

Ebből kifolyólag a  $\phi$  leképezést, pontosabban ennek koordináta-függvényeit *bázisfüggvényeknek* is szokás nevezni<sup>8</sup>. Ha ebben a formában keressük a maximum likelihood megoldást (azaz továbbra is a négyzetes hibát minimalizáló  $\mathbf{w}$  súlyvektort), a korábbi algoritmuson egyáltalán nem kell változtatni, automatikusan a legjobb nemlineáris illesztést kapjuk.

<sup>8</sup> A név adja magát, mivel így  $y_i$ -t ezen bázisban írjuk fel, ahol a koordinátákat  $\mathbf{w}$  szolgáltatja.



2.3. ábra. Másod- és harmadfokú polinomiális regresszió a  $\phi$  függvény alkalmas választásával. Az algoritmus változatlan, mindössze a minták transzformációját cseréltük le.

### 2.3.1 Radiális bázisfüggvények

Polinomiális regresszió helyett általánosabb nemlineáris illesztést kaphatunk ún. *radiális bázisfüggvények* alkalmazásával. Valójában itt is csupán a  $\phi$  transzformáció ügyes megválasztásáról van szó, de most  $x_i$  hatványai helyett  $b_l$  „bázispontokat” tűzünk ki, és minden mintát ezen pontokhoz viszonyított távolságának segítségével reprezentálunk. A leggyakrabban használt radiális bázisfüggvény a Gauss RBF<sup>9</sup>, amely egy dimenzióban

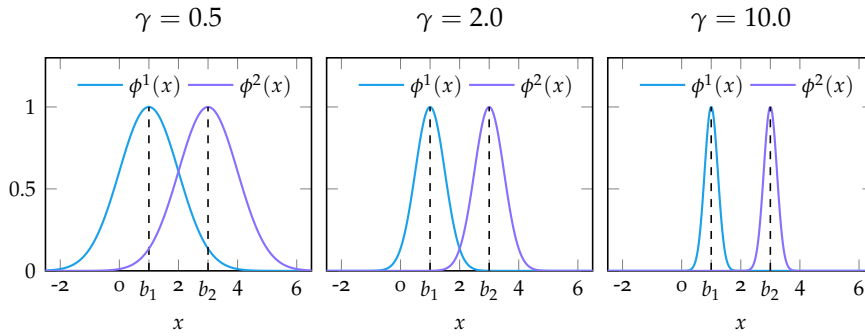
$$\phi^l(x_i) = e^{-\gamma(x_i - b_l)^2},$$

<sup>9</sup> A név onnan ered, hogy a formula nagyon hasonlít a normális eloszlás képletéhez.

ahol  $b_l$  az  $l$ -edik bázispont. Az  $x_i$  mintapont reprezentációja tehát

$$\phi(x_i) = \begin{bmatrix} e^{-\gamma(x_i-b_1)^2} \\ e^{-\gamma(x_i-b_2)^2} \\ \vdots \\ e^{-\gamma(x_i-b_L)^2} \end{bmatrix}.$$

A bázisfüggvény  $\gamma$  paramétere azt befolyásolja, hogy a bázispontok milyen távolra „látnak el”: magas érték esetén  $x_i$  reprezentációja csak akkor lesz zérustól eltérő, ha a minta valamelyik bázispont közvetlen közelébe esik, míg alacsony érték esetén a bázisfüggvények átfednek (2.4. ábra).



2.4. ábra. Gauss RBF különböző  $\gamma$  paraméterekkel.

Extrém alacsony értéknél a reprezentációkra

$$\phi(x_i) \approx \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

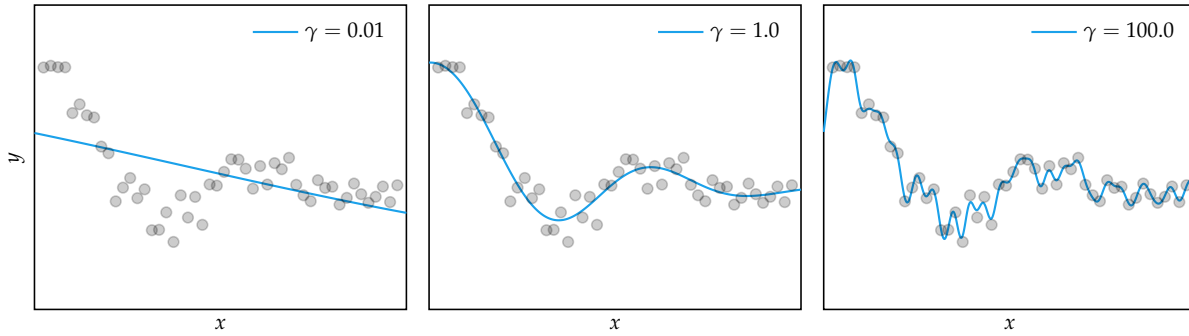
azaz a  $\mathbf{w}$  együtthatók beállításával tulajdonképpen csak egy konstans tanulunk<sup>10</sup>.

Extrém magas értékeknél, ha például  $x_i \approx b_2$ , akkor

$$\phi(x_i) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

<sup>10</sup> A szabadsági fokaink száma 1-re redukálódik: bárhogyan is próbálkozunk, az azzal lesz egyenértékű, mintha a  $\mathbf{1}^\top \mathbf{w}$  konstans állítanánk be. A négyzetes hibát minimalizáló konstans nem más, mint az  $y$  kimenetek átlaga.

azaz a  $\mathbf{w}$  együtthatók felhasználásával minden egyes bázispont „megjegyezhet” egy-egy  $y$  értéket. A 2.5 ábra bemutatja, hogy e két szélsőség között milyen hatással van az illesztésre a  $\gamma$  paraméter.



2.5. ábra. Nemlineáris regresszió Gauss RBF bázisfüggvénnyel. Összesen 100 bázispontot osztottunk el egyenletesen az értelmezési tartományon.

## 2.4 MAP becslés és regularizáció

Az eddigiekben maximum likelihood becslést végeztünk, amely nem vett figyelembe semmiféle előzetes megkötést a  $\mathbf{w}$  modellparaméterekről. Térjünk vissza a bayesi szemléletmódhoz, és tegyük fel a következő priort<sup>11</sup>  $\mathbf{w}$ -re:

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{\frac{D}{2}} e^{-\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}}.$$

Ez a prior szemléletesen annyit jelent, hogy a  $\mathbf{w}$  súlyvektor elemeit zérus közelében szeretnénk látni. A Bayes-tétel szerint

$$p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \beta, \alpha) \propto p(\mathbf{y} | \mathbf{X}, \mathbf{w}, \beta) p(\mathbf{w} | \alpha) \\ \propto e^{-\frac{\beta}{2} \sum_i (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2} \cdot e^{-\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}},$$

ahonnan a már megszokott negatív logaritmust véve megismételjük a korábbi számításunkat:

$$-\ln p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \beta, \alpha) = \underbrace{\frac{\beta}{2} \sum_i (y_i - \mathbf{w}^\top \phi_i)^2}_{L(\mathbf{w})} + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} + \text{const.}$$

<sup>11</sup> Ez ismét csak „technikai” jellegű prior, amellyel nem komplex háttértudásunkat szeretnénk megfogalmazni, hanem – egy később tisztázandó, de jól meghatározott értelemben – arra törekszünk, hogy a modell „egyszerű” legyen (lásd a 2.5 szakaszt).

A (2.5) egyenletben kiszámolt veszteségfüggvényünk mellé most egy új tag került be. Az efféle tagokat, amelyek tehát arról gondoskodnak, hogy a  $\mathbf{w}$  súlyvektor elemei – szebben mondva  $\mathbf{w}$  normája – kicsik maradjanak, *regularizációs tagnak*, a súlyvektor normájának csökkentését pedig *regularizációnak* nevezzük. A gradien-sünk most

$$-\beta \left( \Phi^T \mathbf{y} - \Phi^T \Phi \mathbf{w} \right) + \alpha \mathbf{w} = 0,$$

amelyet átrendezve

$$\mathbf{w} = \left( \Phi^T \Phi + \lambda \mathbf{I} \right)^{-1} \Phi^T \mathbf{y},$$

ahol  $\lambda = \alpha / \beta$ . A (2.6) egyenlettel összevetve látjuk, hogy csupán egy apró módosítással kell élnünk, hogy a ML helyett MAP megoldást kapjunk.

A regularizáció hatását legegyszerűbben egy példán keresztül érthetjük meg. Először is módosítsuk az algoritmusunkat:

```
struct LinearRegression
    w # modellsúlyok
    λ # regularizáció

    LinearRegression(D; λ=0) = new(zeros(D), λ)
end
function train!(m::LinearRegression, Φ, y)
    m.w .= (Φ'Φ + m.λ*I) \ (Φ'y)
end
function predict(m::LinearRegression, Φ_test)
    return Φ_test*m.w
end
```

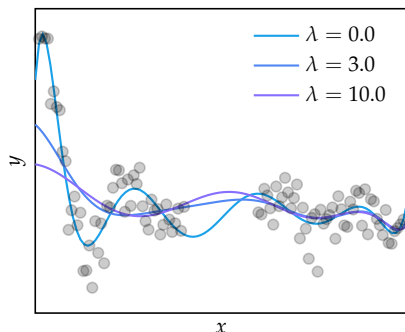
2.2. algoritmus. Lineáris regresszió regularizációval.

A 2.6 ábrán látjuk az illesztések eredményét különböző regularizációs együtthatók mellett. A regularizációval az illesztett függvény oszcillációját tudjuk mér-sékelni, amely – mint látni fogjuk – a modell általánosító képességére lesz jótékony hatással<sup>12</sup>. A következő szakaszban ezt vizsgáljuk meg részletesebben.

## 2.5 A bias-variancia dilemma

Egy gépi tanulási modelltől – sőt, valójában bármilyen tudományos modelltől – két dolgot várunk: egyrészt illeszkedjen a megfigyeléseinkhez, másrészt pedig szolgáltasson pontos jóslatokat a további kísérleteket illetően. Kevés hasznát vesszük

<sup>12</sup> További kellemes mellékhatás, hogy az egységmátrix  $\lambda$ -szorozásának hozzáadásával a problémát numerikusan is kezelhetőbbé tesszük (a sajátértékek eltolásával a  $\Phi^T \Phi$  mátrix kondíciós számát csökkentjük).



2.6. ábra. Lineáris regresszió különböző erősségű regularizációval. A  $\lambda$  paraméter szemléletesen azt befolyásolja, hogy az illesztett függvény mennyire oszcilláljon.

egy olyan modellnek, amely „mindent megmagyaráz, de semmit sem jósol meg”<sup>13</sup>. A bázisfüggvényes megközelítéssel megtehetjük például, hogy minden tanítómintát felveszünk bázispontnak,  $\gamma$ -t pedig kellően nagynak választjuk. Ekkor a modellünk tökéletesen memorizál, reprodukál minden tanító be- és kimenetet, ám a tesztmintákra – hacsak nem esnek nagyon-nagyon közel egy bázispont-hoz – nem ad értelmes jóslatot, nem képes általánosítani.

Az általánosítóképesség vizsgálatához tekintsük át ismét a modellünket. Egy  $f$  függvénykapcsolatot szeretnénk véges sok adatból megbecsülni, amelyeket zérus várható értékű, normális eloszlású zaj terhel:

$$y = f(\mathbf{x}) + \varepsilon,$$

$$\varepsilon \sim \mathcal{N}(\varepsilon \mid 0, \beta^{-1}).$$

Láttuk, hogy lineáris regressziónál az  $\hat{f}$  becslés a következő formát ölti:

$$\hat{f}(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}),$$

ahol a  $\phi$  bázisfüggvények gondoskodnak a nemlineáris esetekről. Sajnos a mintahalmaz végeessége nehezé teszi mind a becslést, mind a modell minősítését. A valódi  $p(\mathbf{X}, \mathbf{y})$  eloszlást nem ismerjük, csupán egy ebből származó mintahalmaz áll rendelkezésünkre; ennek ellenére általánosságban, a konkrét mintahalmaztól függetlenül kellene minősíteni a modellünket.

Ennek érdekében az elméleti vizsgálódások során az összes lehetséges mintahalmaz feletti várható értékkel fogunk dolgozni. A modell várható négyzetes hibája a következőképpen dekomponálható (a várható érték tehát most az összes

<sup>13</sup> Ahogyan erre Laplace is rámutatott Napóleonnal folytatott – állítólagos – párbeszédében.

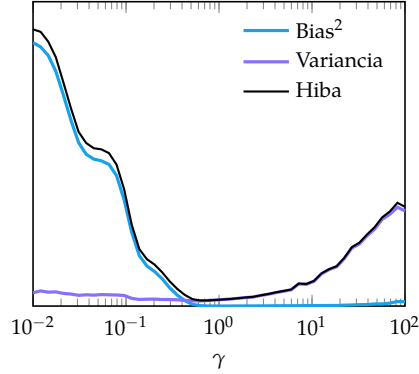
lehetséges mintahalmaz felett értendő):

$$\begin{aligned}
 \mathbb{E} \left[ (y - \hat{f}(\mathbf{x}))^2 \right] &= \mathbb{E} \left[ \left( f(\mathbf{x}) + \varepsilon - \hat{f}(\mathbf{x}) \right)^2 \right] \\
 &= \mathbb{E} \left[ \left( f(\mathbf{x}) - \mathbb{E} [\hat{f}(\mathbf{x})] + \mathbb{E} [\hat{f}(\mathbf{x})] - \hat{f}(\mathbf{x}) + \varepsilon \right)^2 \right] \\
 &= \left( f(\mathbf{x}) - \mathbb{E} [\hat{f}(\mathbf{x})] \right)^2 + \mathbb{E} \left[ \left( \mathbb{E} [\hat{f}(\mathbf{x})] - \hat{f}(\mathbf{x}) \right)^2 \right] + \mathbb{E} [\varepsilon^2] \\
 &\quad + 2 \left( f(\mathbf{x}) - \mathbb{E} [\hat{f}(\mathbf{x})] \right) \mathbb{E} \left[ \left( \mathbb{E} [\hat{f}(\mathbf{x})] - \hat{f}(\mathbf{x}) \right) \right] \\
 &\quad + 2 \left( f(\mathbf{x}) - \mathbb{E} [\hat{f}(\mathbf{x})] \right) \mathbb{E} [\varepsilon] \\
 &\quad + 2 \mathbb{E} \left[ \varepsilon \left( \mathbb{E} [\hat{f}(\mathbf{x})] - \hat{f}(\mathbf{x}) \right) \right] \\
 &= \underbrace{\left( f(\mathbf{x}) - \mathbb{E} [\hat{f}(\mathbf{x})] \right)^2}_{\text{bias}^2} + \underbrace{\mathbb{V} [\hat{f}(\mathbf{x})]}_{\text{variancia}} + \underbrace{\mathbb{V} [\varepsilon]}_{\text{zaj}},
 \end{aligned}$$

ahol a várható érték tulajdonságai mellett az  $\varepsilon$  zaj függetlenségét használtuk ki. A tagok értelmezése a következő:

- A bias (torzítás) megmutatja, hogy mekkora az eltérés a valódi függvény és a becslés várható értéke között, azaz mennyire „alkalmas” az alkalmazott modell az összefüggés megragadására. Nagy bias esetén *alulilleszkedésről* (*underfitting*) beszélünk, a módszerünk túl egyszerű ahhoz, hogy az összefüggést megtanulja (pl. a 2.5. ábra első diagramja).
- A variancia megmutatja, hogy a modell becslései mennyire térnek el egymástól különböző, zajos mintahalmazokon tanítva. Nagy variancia arra enged következtetni, hogy a modell a zajra is ráilleszkedik, azaz *túlilleszkedik* (*túltanul, overfitting*), és ebből fakadóan az általánosító képessége gyenge lesz. Erre úgy is gondolhatunk, hogy a modellünk túl komplex (pl. a 2.5. ábra utolsó diagramja).
- A megfigyelési zajjal semmit nem tudunk kezdeni; ez a modellünktől független, mindenképpen jelen van.

A modell komplexitásának befolyásolására általában több eszköz közül is választhatunk. Regressziónál a túlilleszkedés elkerülésére próbálkozhatunk például a regularizációs együttható növelésével (amely a varianciacsökkentés irányába hat) vagy a bázispontok számának csökkentésével.



2.7. ábra. Négyzetes hiba, bias és variancia alakulása egy görbeillesztési feladatban az RBF különböző  $\gamma$  paraméterei mellett. A  $\gamma$  paraméter optimális értéke 0.5 körül található; ez alatt a modell alulilleszkedik (magas bias, alacsony variancia), felette pedig túlilleszkedik (alacsony bias, magas variancia).

## 2.6 Bayesi lineáris regresszió

Végül tekintsük a lineáris regresszió teljesen bayesi változatát. Az előző fejezethez hasonlóan bayesi modellátlagolást kell végezni, azaz a

$$p(y_{\text{új}} | x_{\text{új}}, \mathbf{X}, \mathbf{y}, \beta, \alpha) = \int p(y_{\text{új}} | x_{\text{új}}, \mathbf{w}, \beta) p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \beta, \alpha) d\mathbf{w}$$

integrált kell megoldani. Ettől most eltekintünk – kiszámolható zárt formában, de sok barkácsolást igényel – és csak a végeredményt közöljük. A  $\mathbf{w}$  modellparaméterek poszterior eloszlása normálisnak adódik:

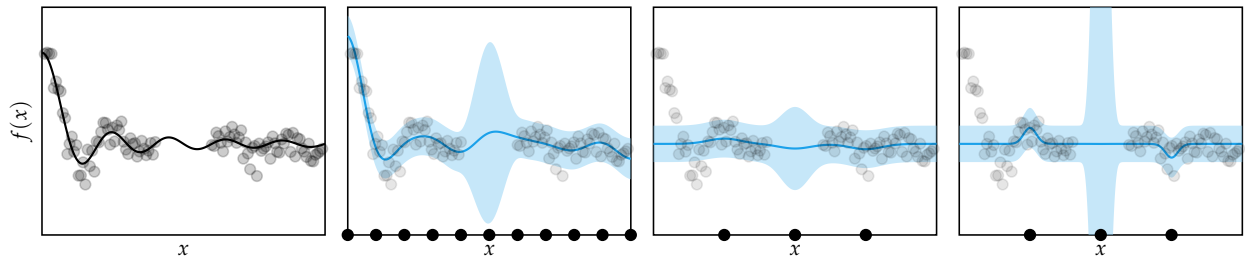
$$\begin{aligned} p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \beta, \alpha) &= \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}, \boldsymbol{\Sigma}), \\ \boldsymbol{\mu} &= \boldsymbol{\Sigma}^{-1} (\beta \boldsymbol{\Phi}^T \mathbf{y}), \\ \boldsymbol{\Sigma} &= \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \alpha \mathbf{I}. \end{aligned}$$

Ezek után a normális eloszlás tulajdonságait kihasználva megmutatható, hogy a keresett eloszlás is normálisnak adódik, mégpedig

$$\begin{aligned} p(y_{\text{új}} | x_{\text{új}}, \mathbf{X}, \mathbf{y}, \beta, \alpha) &= \mathcal{N}(y_{\text{új}} | \mu, \sigma^2), \\ \mu &= \phi(x_{\text{új}})^T \boldsymbol{\mu}, \\ \sigma^2 &= \frac{1}{\beta} + \phi(x_{\text{új}})^T \boldsymbol{\Sigma}^{-1} \phi(x_{\text{új}}) \end{aligned}$$

paraméterekkel. A módszer egy lehetséges megvalósítását mutatja a 2.3 algoritmus. Az eredmények a 2.8 ábrán láthatók.





2.8. ábra. Bayesi lineáris regresszió várható értékkel és szórással. A bázispontok helyét fekete körök jelölik. A bal oldalon a valódi függvény látható, a további ábrákon illesztések más-más  $\gamma$  paraméterrel.

```

struct BayesianLinearRegression
     $\mu$  # várható érték
     $\Sigma$  # kovarianciamátrix
     $\alpha$  # súly prior (hiperparaméter)
     $\beta$  # pontosság (hiperparaméter)

    BayesianLinearRegression(D; $\alpha$ =0.1, $\beta$ =100.0) =
        new(zeros(D), zeros(D,D),  $\alpha$ ,  $\beta$ )
end
function train!(m::BayesianLinearRegression,  $\Phi$ , y)
     $\alpha$ ,  $\beta$  = m. $\alpha$ , m. $\beta$ 

    m. $\Sigma$  .=  $\beta * \Phi' \Phi + \alpha * \mathbf{I}$ 
    m. $\mu$  .= m. $\Sigma \setminus (\beta * \Phi' y)$ 
end
function predict(m::BayesianLinearRegression,  $\Phi_{\text{test}}$ )
     $\mu$ ,  $\Sigma$ ,  $\beta$  = m. $\mu$ , m. $\Sigma$ , m. $\beta$ 

    p $\mu$  =  $\Phi_{\text{test}} * \mu$ 
    p $\sigma^2$  = 1/ $\beta$  .+ sum( $\Phi_{\text{test}}' .*(\Sigma \setminus \Phi_{\text{test}}')$ , dims=1)
    return p $\mu$ , p $\sigma^2$ 
end

```

2.3. algoritmus. Bayesi lineáris regresszió.



## 3. fejezet

# Lineáris klasszifikáció

Az 1.2. szakaszban megismerkedtünk egy egyszerű osztályozóval, a 2. fejezet pedig a lineáris modellek egy családját tárgyalta. Ebben a fejezetben a két ötlet ötvöztetésével új modellt állítunk össze, amit a gyakorlatban is sokszor használunk, és a neurális hálózatok alapegységeként is gondolhatunk rá.

### 3.1 Valószínűségi modell

A felállítás egészen hasonló a regressziós esethez: a bemenetek  $D$ -dimenziós vektorok, a kimenetek viszont két osztályt reprezentálnak<sup>1</sup>.

$$\begin{aligned} \mathbf{x}_i &\in \mathbb{R}^D, \\ y_i &\in \{0, 1\}. \end{aligned}$$

<sup>1</sup> Egyelőre tehát bináris osztályozásnál maradunk; a többosztályos eseteket nagyobb általánosságban a következő fejezetben tárgyaljuk.

Gondolhatunk például egy orvosi diagnosztikai problémára: az  $\mathbf{x}_i$  minta az  $i$ . páciensről írja le (például kora, neme, laborleletei stb.),  $y_i$  pedig a „egészséges” vagy „beteg” állapotot. A célunk, hogy a modellt ismert esetekkel tanítva később egy  $\mathbf{x}_{új}$  páciensről is meg tudjuk állapítani, hogy egészséges vagy beteg. Mi lehet tehát annak a valószínűsége, hogy a páciensünk beteg? A Bayes-tételt felhasználva<sup>2</sup>

$$p(y_i = 1 | \mathbf{x}_i) = \frac{p(\mathbf{x}_i | y_i = 1) p(y_i = 1)}{p(\mathbf{x}_i | y_i = 1) p(y_i = 1) + p(\mathbf{x}_i | y_i = 0) p(y_i = 0)} \quad (3.1)$$

$$= \frac{1}{1 + \frac{p(\mathbf{x}_i | y_i = 0) p(y_i = 0)}{p(\mathbf{x}_i | y_i = 1) p(y_i = 1)}}. \quad (3.2)$$

<sup>2</sup> A Bayes-tételnek ez az alakja mindössze annyiban különbözik a korábbtól, hogy nevezőben használtuk a teljes valószínűség tételét.

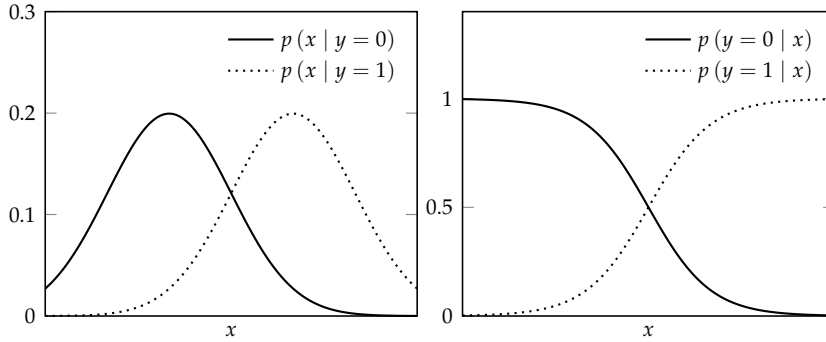
A  $p(\mathbf{x}_i | y_i = 1)$ -vel jelölt mennyiség a „beteg” osztályba tartozó minták feltételes eloszlását jelöli (például milyen laborleletekre számítunk egy beteg ember esetében), míg a  $p(y_i = 1)$  a betegség *a priori* valószínűsége (például milyen gyakran fordul elő az adott populációban).

A feltételes eloszlások megbecsülése nehéz feladat. Ebben a fejezetben nem is igen vállalkozunk rá, hanem teszünk néhány egyszerűsítő feltevést<sup>3</sup>:

$$p(\mathbf{x}_i | y_i = 1) = \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = \frac{1}{Z} \cdot e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_1)},$$

$$p(\mathbf{x}_i | y_i = 0) = \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_0, \boldsymbol{\Sigma}) = \frac{1}{Z} \cdot e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_0)},$$

azaz feltesszük, hogy a két osztályhoz tartozó minták normális eloszlást követnek, ráadásul a kovarianciamátrixuk is megegyezik. A  $Z$  konstans a normalizációt jelöli, ami a két eloszlásnál ugyanaz. E feltevésünk mellett a (3.1) egyenlet bal és jobb oldalán látható eloszlásokról a 3.1. ábra tájékoztat.



<sup>3</sup> Ez tényleg durva egyszerűsítés. Semmi okunk nincs azt hinni, hogy a feltételes eloszlások valóban ilyen egyszerűek volnának; csupán a könnyű számolhatóságra törekszünk. Később, az 5. fejezetben megvizsgáljuk, hogy hogyan lehet ezeket is az adatokból megbecsülni.

3.1. ábra. Az egyes osztályokhoz tartozó minták feltételes eloszlása és a poszterior valószínűségek egy dimenzióban. A két grafikon között a Bayes-tétel biztosítja az átjárást. Az  $x$  minta alacsony értéke esetén valószínűbb, hogy a páciensünk egészséges ( $y = 0$ ); magas értéknél fordított a helyzet.

Végezzük el a (3.2) egyenlet nevezőjében látható osztást. Az  $\mathbf{x}_i$ -t tartalmazó tagokra

$$\frac{p(\mathbf{x}_i | y_i = 1)}{p(\mathbf{x}_i | y_i = 0)} = e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_0)},$$

$$= e^{(\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0))^\top \mathbf{x}_i - \frac{1}{2}(\boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_0)},$$

azaz a négyzetes tagok kiestek, a hatványkitevőben csak  $\mathbf{x}_i$ -ben lineáris és konstans tagok maradtak. Egy huszárvágással gyűjtsük most össze a lineáris tagok

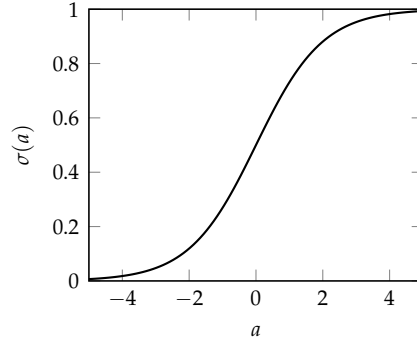
együtthatóit  $\mathbf{w}$ -be, az összes  $\mathbf{x}_i$ -t nem tartalmazó tagot pedig  $b$ -be. A keresett valószínűség a következőképpen egyszerűsödik<sup>4</sup>:

$$p(y_i = 1 | \mathbf{x}_i) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x}_i + b)}} = \sigma(\mathbf{w}^\top \mathbf{x}_i + b),$$

ahol a

$$\sigma(a) := \frac{1}{1 + e^{-a}}$$

ún. *szigmoid* függvényt vezettük be.



<sup>4</sup> Úgy is mondhatjuk, hogy eszünk ágában sincs a lineáris és konstans tagokat a fenti bonyult formában számolgatni, sem pedig a kovarianciamátrixok felírásával vacakolni; megelégszünk azzal, hogy *valamilyen*  $\mathbf{w}$ -re és  $b$ -re igaz az állítás. Célravezetőbb  $\mathbf{w}$ -re és  $b$ -re ismeretlen paraméterként gondolni, és például maximum likelihood módon tanulni őket.

3.2. ábra. Sigmoid függvény. Vegyük észre, hogy a függvény a  $(0, 1)$  intervallumba képez; szemléletes jelentése az  $y = 1$  osztályba tartozás valószínűsége.

Ez már lényegében egy lineáris modell. Ha  $b$ -t a lineáris regresszióhoz hasonlóan, a (2.2) egyenletben látott módon hozzáfűzzük a  $\mathbf{w}$  súlyvektorhoz, csupán abban térünk el a regressziós esettől, hogy most a skaláris szorzat eredményét egy további  $\sigma$  függvény a  $(0, 1)$  intervallumba képezi. Összefoglalva tehát a keresett valószínűségek lineáris/sigmoid összefüggéssel becsülhetők:

$$p(y_i = 1 | \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^\top \phi(\mathbf{x}_i)) := \sigma_i,$$

$$p(y_i = 0 | \mathbf{x}_i, \mathbf{w}) = 1 - \sigma_i.$$

A modell befejezéséhez kövessük az (1.1) egyenletnél alkalmazott stratégiát. Az iménti két esetet egyesítve az  $y_i$  kimenet eloszlása Bernoulli-eloszlást követ:

$$p(y_i | \mathbf{x}_i, \mathbf{w}) = \sigma_i^{y_i} (1 - \sigma_i)^{1-y_i} = \text{Bern}(y_i | \sigma(\mathbf{w}^\top \phi(\mathbf{x}_i))),$$

a minták között függetlenséget feltéve pedig felírhatjuk a likelihoodot:

$$p(\mathbf{y} | \mathbf{X}, \mathbf{w}) = \prod_i \sigma_i^{y_i} (1 - \sigma_i)^{1-y_i} = \prod_i \text{Bern}(y_i | \sigma(\mathbf{w}^\top \phi(\mathbf{x}_i))). \quad (3.3)$$

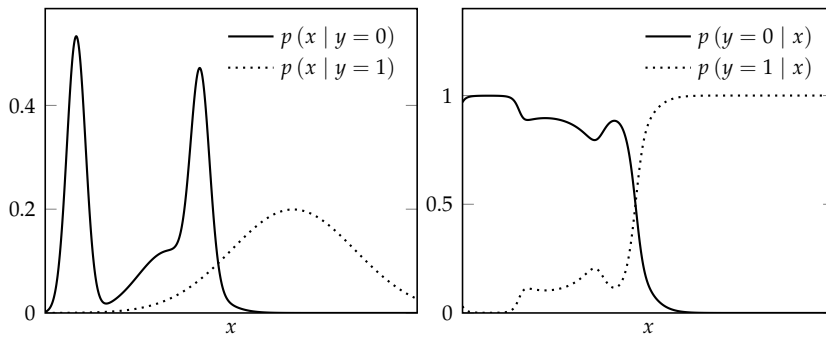
A célunk ismét a likelihood maximalizálása a  $\mathbf{w}$  súlyvektor szerint; ennek birtokában könnyedén adhatunk becslést egy új mintára is, csupán a

$$y_{\text{új}} = \begin{cases} 0, & \text{ha } \sigma(\mathbf{w}^\top \phi(\mathbf{x}_{\text{új}})) \leq 0.5 \\ 1, & \text{ha } \sigma(\mathbf{w}^\top \phi(\mathbf{x}_{\text{új}})) > 0.5 \end{cases}$$

formulát kell alkalmaznunk.

Ezt a modellt – kissé félrevezető módon – *logisztikus regresszió*nak is szokták hívni, és klasszikus példája az ún. *diszkriminatív modellek*nek. Azzal, hogy a  $\mathbf{w}$  súlyvektort közvetlenül az adatból tanuljuk, elveszítettük a lehetőséget, hogy az adatok feltételes eloszlásának várható értékéről, kovarianciájáról bármit megtudjunk, a modell csupán az  $p(\mathbf{y} | \mathbf{X}, \mathbf{w})$  valószínűségeket próbálja becsülni.

A generatív modellek ezzel szemben a teljes  $p(\mathbf{X}, \mathbf{y})$  együttes eloszlást próbálják megtanulni, amivel gyakorlatilag mindent megkapunk: outlier detekciót, osztályozást, mintagenerálást stb., ez azonban sok mintát és számítási erőforrást követel meg. Ha csupán osztályozni szeretnénk, ez pazarló lehet (lásd a 3.3. ábrát).



3.3. ábra. Ha az osztályokba tartozó minták bonyolultabb eloszlást követnek, a posztterior valószínűségekre még mindig „elég jó” lehet a szigmoid becslés; nem szükséges mintákat és erőforrásokat pazarolni  $x$  feltételes eloszlásainak megtanulására, ha az osztályozást nem befolyásolják. A bal oldali grafikonon lévő fekete „huplik” például nemigen vannak hatással az osztályozásra, jobb oldalt még mindig „majdnem” szigmoidokat látunk.

### 3.2 Maximum likelihood megoldás

A maximum likelihood megoldás kiszámolásához a már jól ismert receptet követjük. A veszteségfüggvény a likelihood negatív logaritmusaként adódik:

$$L(\mathbf{w}) = -\ln p(\mathbf{y} | \mathbf{X}, \mathbf{w}) = -\sum_i y_i \ln \sigma_i + (1 - y_i) \ln(1 - \sigma_i),$$

amelyet *keresztentropiának* nevezünk.

Ebben a fejezetben az egyszerűség kedvéért a maximum likelihood megoldással foglalkozunk. A teljesen bayesi verziót későbbre halasztjuk, mert bonyoltabb technikákat igényel; a maximum a posteriori becsléshez a lineáris regresszióhoz hasonlóan a

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I})$$

priort tesszük fel, majd a Bayes-tétel alapján

$$p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \alpha) \propto p(\mathbf{y} | \mathbf{X}, \mathbf{w}) p(\mathbf{w} | \alpha),$$

így a veszteségfüggvénybe ismét csak egy regularizációs tag kerül be:

$$L(\mathbf{w}) = -\ln p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \alpha) = -\sum_i y_i \ln \sigma_i + (1 - y_i) \ln(1 - \sigma_i) + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}.$$

A keresztentropia minimalizálásához a szükségünk lesz a szigmoid függvény deriváltjára:

$$\sigma'(a) = \frac{-1 \cdot -e^{-a}}{(1 + e^{-a})^2} = \frac{1}{1 + e^{-a}} \cdot \frac{e^{-a}}{1 + e^{-a}} = \sigma(a) \cdot (1 - \sigma(a)),$$

így a gradiens a következő alakot ölti<sup>5</sup>:

$$\begin{aligned} \nabla_{\mathbf{w}} L &= -\sum_i y_i \frac{1}{\sigma_i} \sigma_i (1 - \sigma_i) \phi_i - (1 - y_i) \frac{1}{1 - \sigma_i} \sigma_i (1 - \sigma_i) \phi_i \\ &= -\sum_i (y_i (1 - \sigma_i) - (1 - y_i) \sigma_i) \phi_i \\ &= -\sum_i (y_i - \sigma_i) \phi_i. \end{aligned}$$

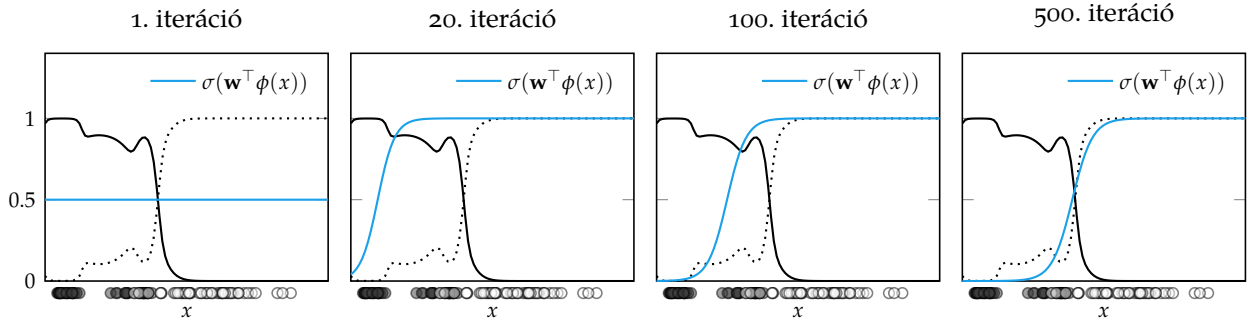
<sup>5</sup> Figyeljük meg, hogy ha a  $\sigma_i$  becslés „egyetért” az  $y_i$  kimenettel (pl. egy 1 címkével rendelkező mintára 0.99-es valószínűséget mond), akkor a gradiens is kicsinek adódik.

Sajnos most nem tehetjük meg azt, amit a regressziónál: hiába állítjuk zérusra a gradienst, nem jutunk olyan egyenlethez, amelyet könnyen meg tudnánk oldani. Ehelyett a veszteségfüggvény minimalizálásához az 1.1.4. szakaszhoz hasonlóan numerikusan próbálkozunk, azaz  $\mathbf{w}$ -t iteratív módon mindaddig módosítjuk, amíg minimumba nem jutunk. Erre a legegyszerűbb séma, ha mindig a gradienssel ellentétes irányba lépünk, azaz

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L, \quad (3.4)$$

ahol az  $\eta$  a *tanulási tényező* (*learning rate*), amely a lépések nagyságát szabályozza<sup>6</sup>.

<sup>6</sup> Ennek belövésére a következő fejezetben látunk stratégiákat.



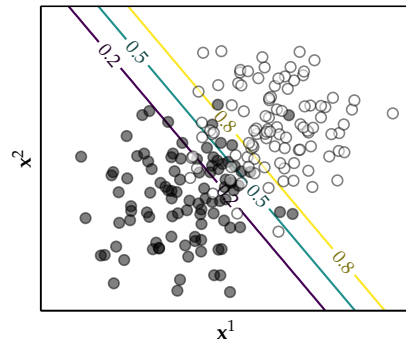
Az egydimenziós esetet a 3.4. ábra, a kétdimenziós esetet a 3.5. ábra, a teljes eljárást a 3.1. algoritmus szemlélteti; a hatékony implementációhoz ismét a műveletek mátrixos alakját használjuk:

$$\begin{aligned} \sum_i (y_i - \sigma_i) \phi_i &\rightsquigarrow \Phi (\mathbf{y} - \boldsymbol{\sigma}), \\ \begin{bmatrix} \mathbf{w}^\top \phi(\mathbf{x}_1) \\ \mathbf{w}^\top \phi(\mathbf{x}_2) \\ \vdots \\ \mathbf{w}^\top \phi(\mathbf{x}_N) \end{bmatrix} &\rightsquigarrow \Phi \mathbf{w}, \end{aligned} \quad (3.5)$$

ahol a  $\Phi$  mátrix soraiban a transzformált  $\phi(\mathbf{x}_i)$  adatokat tartalmazza.

3.4. ábra. Logisztikus regresszió egy dimenzióban. Az  $x$  tengelyen a tanítóminták láthatók. A fekete vonalak a két osztály valódi poszterior valószínűségét ábrázolják; a kék vonal a modellből származó szigmoid becslés.





3.5. ábra. Logisztikus regresszió két dimenzióban. Az adatokat osztályuktól függően fekete illetve fehér pontok ábrázolják. A szintvonalak a tanult modell kimenetét, azaz a fehér osztály becscült valószínűségét ábrázolják.

```

struct LogisticRegression
    w # modellsúlyok

    LogisticRegression(D) = new(zeros(D))
end
function train!(m::LogisticRegression,  $\Phi$ , y;  $\eta$ =0.01, iters=100)
    for i in 1:iters
        s =  $\sigma$ .( $\Phi$  * m.w)
        d = y .- s

        m.w .+=  $\eta$  *  $\Phi^T$  d
    end
end
function predict(m::LogisticRegression,  $\Phi_{\text{test}}$ )
    return  $\sigma$ .( $\Phi_{\text{test}}$  * m.w)
end

```

3.1. algoritmus. Logisztikus regresszió gradiens-módszerrel.

### 3.3 Bázisfüggvények több dimenzióban

A 2.3. szakaszban láttuk, hogyan lehet egy lineáris regressziós modellt nemlineárisra tenni a  $\phi$  leképezés módosításával. Az ötlet klasszifikációs modelleknél ugyanúgy működik; ezt nézzük meg általános, többdimenziós esetben. Használjuk a Gauss RBF többváltozós verzióját és az egyszerűség kedvéért válasszuk bázispontoknak magukat a mintapontokat<sup>7</sup>. Ekkor egy minta reprezentációja

$$\phi(\mathbf{x}_i) = \begin{bmatrix} k(\mathbf{x}_i, \mathbf{x}_1) \\ k(\mathbf{x}_i, \mathbf{x}_2) \\ \vdots \\ k(\mathbf{x}_i, \mathbf{x}_N) \end{bmatrix},$$

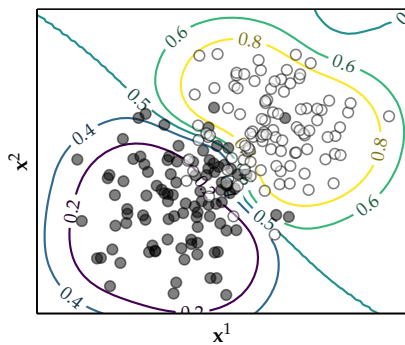
a mintákat a sorokban tartalmazó  $\Phi$  mátrix pedig

$$\Phi = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix},$$

ahol  $k$ -t a következőképpen definiáltuk<sup>8</sup>:

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}.$$

A tanult nemlineáris osztályozót a 3.6. ábra mutatja.



<sup>7</sup> Ha így járunk el, a  $\gamma$  paraméter megválasztásánál ügyelnünk kell, hogy ne illeszkedjen alul vagy túl a modell.

<sup>8</sup>  $k(\cdot, \cdot)$ -ra úgy is gondolhatunk, mint egy függvényre, ami a minták „hasonlóságát” méri. A hasonlóságokon alapuló tanulás ötletét részletesebben is körbejárjuk a 8. fejezetben.

3.6. ábra. Nemlineáris osztályozás logisztikus regresszióval és Gauss RBF bázisfüggvénnyel. A szintvonalak a „fehér” osztályba tartozás becslét valószínűségét ábrázolják.

### 3.4 Az implementáció részletei

Végül megnézzünk néhány gyakorlati szempontot is. Gépi tanulási számításoknál gyakran előfordul, hogy ha csak a tankönyvben olvasott formulákat implementáljuk, a számábrázolás korlátai vagy más numerikus anomáliák miatt az algoritmus rosszul, vagy egyáltalán nem működik<sup>9</sup>. Optimalizációs algoritmusból rengeteget ismerünk; részletesebben a következő fejezetben fogunk foglalkozni velük, most csupán – illusztrációképpen – a Newton–Raphson módszert nézzük meg.

<sup>9</sup> Ez sokszor úgy jelentkezik, hogy az eddig helyesen működő algoritmusunk egyszer csak elkezd NaN-okat dobálni.

#### 3.4.1 Numerikus stabilitás

Különösen hatványozás esetén kell ügyelnünk a számábrázolás kérdésére. A szigmoid függvényben például találkozunk  $e$ -addal is; a túlsordulás elkerülése érdekében célszerű ennek argumentumát úgy alakítani, hogy a kitevőbe negatív szám kerüljön. A 3.1. táblázat azt mutatja, hogy a szigmoidot, illetve ennek származékait milyen formában érdemes felírni az  $a$  argumentumtól függően.

	$\sigma(a)$	$1 - \sigma(a)$	$\ln \sigma(a)$	$\ln(1 - \sigma(a))$
$a > 0$	$\frac{1}{1+e^{-a}}$	$\frac{e^{-a}}{1+e^{-a}}$	$-\ln(1+e^{-a})$	$-a - \ln(1+e^{-a})$
$a \leq 0$	$\frac{1}{1+e^a}$	$\frac{e^a}{1+e^a}$	$-\ln(1+e^a)$	$a - \ln(1+e^a)$

3.1. táblázat. Numerikus stabilitás érdekében végzett átalakítások.

A táblázat felhasználásával a keresztentrópiát is „biztonságosabb” formára hozhatjuk<sup>10</sup>:

$$-y \ln(\sigma(a)) - (1-y) \ln(1 - \sigma(a)) = \begin{cases} a(1-y) + \ln(1+e^{-a}), & \text{ha } a \geq 0 \\ -ay + \ln(1+e^a) & \text{különben.} \end{cases}$$

<sup>10</sup> A keresztentrópia gépi tanulási könyvtárakban, pl. TensorFlow-ban vagy PyTorch-ban is így van megvalósítva.

```
function σ(a)
    u = exp(-abs(a))
    return a>0 ? 1/(1+u) : u/(1+u)
end
function BCE(a,y)
    u = log1p(exp(-abs(a)))
    return a>0 ? a*(1-y)+u : -a*y+u
end
```

3.2. algoritmus. Numerikusan stabil szigmoid függvény és keresztentrópia.

### 3.4.2 Optimalizáció a Newton–Raphson módszerrel

Másodrendű módszerként a Newton–Raphson algoritmustól<sup>11</sup> gyorsabb konvergenciát remélünk. Egy-egy lépés kiszámolásához a gradiens mellett a Hesse-mátrixra van szükségünk<sup>12</sup>:

$$\begin{aligned}\nabla_{\mathbf{w}} L &= - \sum_i \underbrace{(y_i - \sigma_i)}_{:=d_i} \phi_i = -\Phi^\top \mathbf{d}, \\ \nabla_{\mathbf{w}}^2 L &= \sum_i \underbrace{\sigma_i(1 - \sigma_i)}_{:=\sigma'_i} \phi_i \phi_i^\top = \Phi^\top \mathbf{A} \Phi,\end{aligned}$$

ahol  $\mathbf{A}$  diagonális mátrix, ami a  $\sigma'$  deriváltakat tartalmazza a főátlón:

$$\mathbf{A} = \begin{bmatrix} \sigma'_1 & & 0 \\ & \ddots & \\ 0 & & \sigma'_N \end{bmatrix}$$

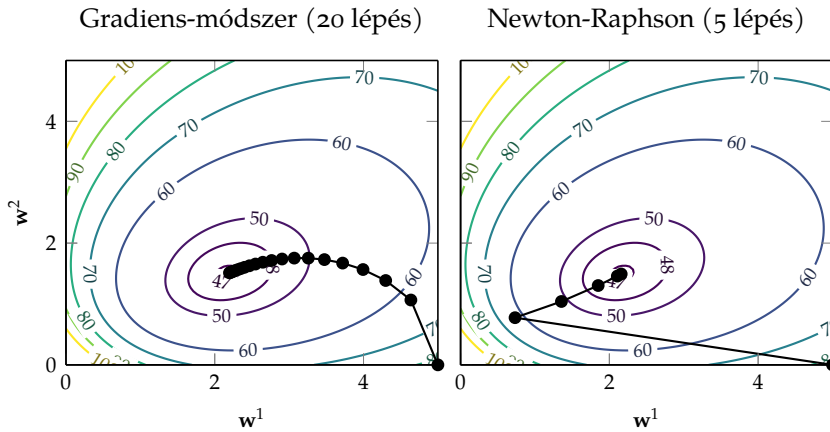
Az iteráció során megtett lépések

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \cdot (\Phi^\top \mathbf{A} \Phi)^{-1} \Phi^\top \mathbf{d},$$

ahol az  $\alpha$  lépéshosszt többféleképpen is megválaszthatjuk (a 3.3. algoritmus egy egyszerű iránymenti keresési sémát használ).

<sup>11</sup> Lásd az A függelékét.

<sup>12</sup> A Hesse-mátrix elemszáma négyzetesen nő a minták számával, ennél fogva a módszer csak kis-közepes adatmennyiségnél alkalmazható.



3.7. ábra. A gradiens-módszer és a Newton–Raphson eljárás konvergenciája egy kétdimenziós problémában. A veszteséget, mint  $\mathbf{w}$  függvényét szintvonalakkal, az algoritmusok lépéseit fekete vonallal jeöltük. Az utóbbi módszer lépései költségesebbek, de az  $\begin{bmatrix} 5 \\ 0 \end{bmatrix}$  pontból indítva jóval kevesebb lépés alatt konvergál az optimális  $\mathbf{w}$ -hez.

```

function train_newton!(m::LogisticRegression,  $\Phi$ , y; iters=5)
    for i in 1:iters
        l =  $\Phi$  * m.w                # Logit
        s =  $\sigma(l)$                 # Sigmoid aktiváció
        d = y .- s

        g = - $\Phi'$ d                  # Gradiens
        H =  $\Phi' * (s .* (1 .- s) .* \Phi)$  # Hesse-mátrix
        p = -H \ g                  # Új irány

        # Iránymenti keresés visszalépéssel
         $\alpha$  = 1.
        loss = sum(BCE.(l,y))
        while sum(BCE.( $\Phi$ *(m.w .+  $\alpha$ *p),y)) > loss +  $\alpha$ *dot(g,p)*1e-4
             $\alpha$  *= 0.5
        end
        m.w .+=  $\alpha$ *p
    end
end

```

3.3. algoritmus. Newton–Raphson módszer logisztikus regresszióra iránymenti kereséssel. Az  $\alpha$  lépéshossz beállításához először kiszámítjuk a lépés irányát, majd ezen irány mentén – megközelítőleg – minimumot keresünk.

### 3.5 Klasszifikációs modellek kiértékelése

A gépi tanulási munkafolyamatok fontos eleme a modellek prediktív teljesítményének kiértékelése, amelyet a – szándékosan erre a célra fenntartott – teszt adathalmazon végzünk. Gyakran előfordul azonban, hogy nem áll rendelkezésre dedikált tesztalmaz, így ennek előállításáról is magunknak kell gondoskodnunk. Kézenfekvő stratégia a *keresztkiértékelés*, ahol a tesztmintákat a teljes adathalmazról választjuk le, például:

1. Az adathalmazt véletlenszerűen 5 megközelítőleg egyenlő részre vágjuk,
2. Az előbbiekből közül 4 rész képezi a tanítóhalmazt, a fennmaradó pedig a teszt-halmazt, amelyen a prediktív teljesítményt mérjük<sup>13</sup>,
3. A kiértékelést elvégezzük többször oly módon, hogy mindig másik rész játssza a tesztalmaz szerepét (tehát összesen  $5\times$ ),
4. Az 1-3. lépéseket megismétljük  $10\times$ , és összegyűjtjük (vagy átlagoljuk) a kapott teljesítménymetriákat.

<sup>13</sup> Fontos, hogy az így előállított tesztalmazhoz semmilyen formában nem nyúlhatunk hozzá, az előfeldolgozás során sem! Gyakori hiba például, hogy az adatok normalizálása a szétválasztás előtt, a teljes adathalmazt figyelembe véve történik. Ez „csalás”, hiszen így a mintátlagba a tesztminták is beleszól, azaz rejtett módon a teszt adathalmaz tulajdonságait is felhasználjuk a tanítás során.

A következő kérdés a teljesítménymetrikáké. Klasszifikációnál szokás felírni a *konfúziós mátrixot*, amely két osztály esetén a következőképpen fest:

	Jósolt +	Jósolt –
Valódi +	TP	FN
Valódi –	FP	TN

3.2. táblázat. Konfúziós mátrix két osztályra. A TP, FP, TN, FN jelölések rendre a valós pozitív, fals pozitív, valós negatív, fals negatív találatok számát jelentik.

A prediktív teljesítmény kiértékeléséhez használatos metrikákat az előbbi esetszámokból származtatjuk:

$$\begin{aligned}\text{recall vagy true positive rate (TPR)} &= \frac{TP}{TP + FN} \\ \text{false positive rate (FPR)} &= \frac{FP}{FP + TN} \\ \text{precision} &= \frac{TP}{TP + FP}\end{aligned}$$

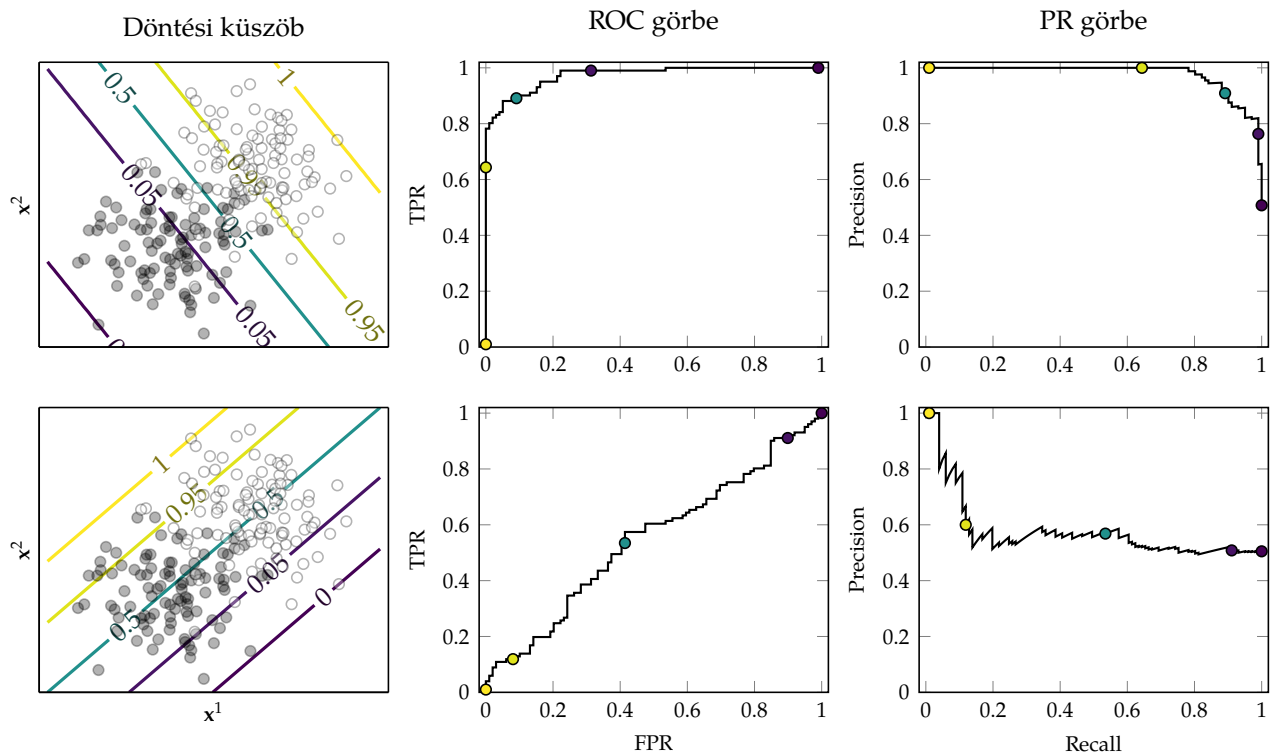
Az így kapott értékek függenek a *döntési küszöbtől*, azaz attól, hogy a szigmoid kiemenet mely értékétől kezdve jósoljuk az adott tesztmintát pozitívnak. Ha például a döntési küszöböt zérusnak választjuk, akkor minden tesztmintát automatikusan pozitívnak jósolunk (azaz FN és TN értéke zérus lesz). Ebben az esetben a TPR és az FPR értékek is triviálisan 1-nek adódnak. Hasonlóképpen, ha a döntési küszöböt 1-re állítjuk, akkor mindent negatívnak jósolunk, azaz TPR és FPR is zérus lesz<sup>14</sup>. A szélsőségek között „tologatva” a döntési küszöböt más-más TPR és FPR értékeket kapunk. Az FPR-TPR közötti összefüggést írja le az ún. *receiver operating characteristic (ROC)* görbe; hasonlóképpen származtathatjuk a *precision-recall (PR)* görbét is (3.8. ábra.).

<sup>14</sup> A biztonság kedvéért ellenőrizzük le ezeket az állításokat a fenti formulákba való behelyettesítéssel.

A görbék alakjára vonatkozóan – némi gondolkodás után – a következő megfigyeléseket tehetjük:

- Az ROC görbe mindenképpen a (0, 0) pontból indul és az (1, 1) pontba tart,
- Egy tökéletes osztályozó ROC görbéje érinti a (0, 1) pontot, egy „jó” osztályozóé pedig megközelíti azt<sup>15</sup>,
- Egy teljesen véletlenszerűen működő osztályozó ROC görbéje az  $y = x$  egyenes mentén, „átlósan” halad,
- A PR görbe mindenképpen a (0, 1) pontból indul és az (1,  $r$ ) pontba tart, ahol  $r$  elárulja, hogy a tesztalmazban mekkora a pozitívok aránya.

<sup>15</sup> Szokás kiszámolni az ROC görbe alatti területet (AUC), amely egy tökéletes osztályozónál 1-nek adódik, véletlenszerű osztályozónál pedig 0.5-nek; kisebb értékek-nél az osztályozó „fordítva” működik.



3.8. ábra. ROC és PR görbék „jó” és „rossz” lineáris klasszifikációs modellekre. A különböző színekkel jelölt döntési küszöbökhez tartozó pontokat megfelelő színű körök jelzik az ROC és PR görbéken. Látjuk, hogy egy jól működő osztályozó ROC görbéje megközelíti a  $(0, 1)$  pontot, míg egy gyakorlatilag véletlenszerűen működő osztályozó ROC görbéje az  $x = y$  egyenes közelében marad.





## 4. fejezet

# Neurális hálózatok

Ebben a fejezetben általánosítjuk az előző fejezetben megismert fogalmakat, és megérkezünk az egyik legelterjedtebb gépi tanulási módszer családjához, a neurális hálózatokhoz.

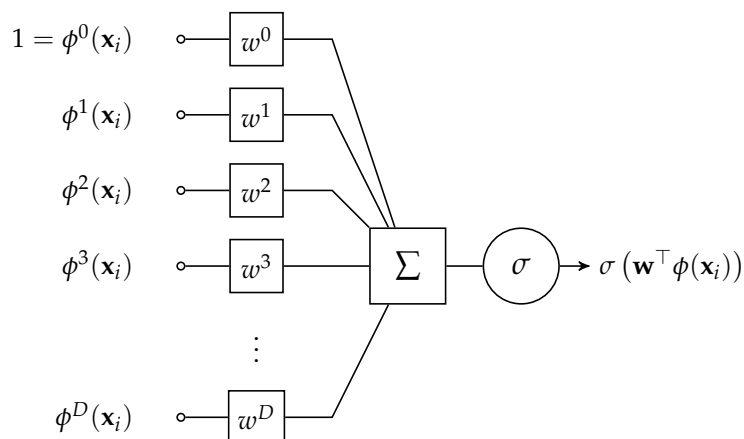
### 4.1 Logisztikus regresszió és a perceptron

Idézzük fel az előző fejezet osztályozó modelljét, amely egy  $\mathbf{x}_i$  bemenetet és a hozzá tartozó  $y_i$  kimenetet feltéve a következőképpen festett:

$$p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^\top \phi(\mathbf{x}_i)).$$

Vizsgáljuk meg, milyen lépéseket végzünk el az egyenlet jobb oldalán látható leképezés során!

1. Az  $\mathbf{x}_i$  bemenő mintát valamilyen  $\phi$  függvénnyel transzformáljuk,
2. A kapott vektort megszorozzuk a  $\mathbf{w}$  súlyvektorral, azaz  $\phi(\mathbf{x}_i)$  és  $\mathbf{w}$  elemeit páronként összeszorozzuk, majd a szorzatokat összeadjuk; erre úgy is gondolhatunk, hogy a transzformált bemenet minden egyes eleméhez egy-egy – a későbbiekben beállítható, finomhangolható – súlyt rendelünk,
3. A kapott skalárra alkalmazzuk a  $\sigma$  függvényt, amely a megsúlyozott és összegzett bemenet alapján a végső kimenetet dönti el.



4.1. ábra. Perceptron, mint az agyi információfeldolgozás valószínűségi modellje.

A 4.1. ábra az előbbi lépéseket szemlélteti diagramos formában. A modellt eredetileg az agyi információfeldolgozás formalizálására javasolták, és a hangzatos *perceptron* nevet kapta<sup>1</sup>.

Ha jobban megnézzük, a diagram valóban „virtuális idegsejtre” emlékeztet; a súlyozott bemenő ágak megfelelnek az idegsejt ingerület felvételéért felelős rövid nyúlványainak (*dendritek*), az összegző a *sejttestnek*, ahol az is eldől, hogy az idegsejt milyen választ ad (*szigmoid aktiváció*), a kimenő ág pedig a válasz továbbadásáért felelős hosszú nyúlvány (*axon*) analógja. Az idegsejtműködést vezérlő differenciálegyenletek leírása Hodgkin és Huxley nevéhez fűződik, és mintegy öt évvel megelőzte a perceptront; a párhuzamok mélyebben is megjelennek, például a válasz (*akciós potenciál*) kialakulása során az ioncsatornák nyílása valóban szigmoid karakterisztikát követ. Fontos azonban megjegyezni, hogy ezeknek a neurális modelleknek kevés köze van ahhoz, ahogy az agyi „osztályozás”, például objektumok felismerése történik; célszerű a perceptront továbbra is statisztikai eszközként tekinteni.

<sup>1</sup> F. Rosenblatt, “THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN.”, *Psychological Review*, 65. évf., 6. sz., 386–408. old., 1958.

## 4.2 Többrétegű neurális hálózatok

A ma használatos neurális hálózatokhoz a perceptron általánosításain keresztül jutunk el. Először is, a továbbiakban nem fogunk az osztályozáshoz ragaszkodni, és a valószínűségi értelmezést is jórészt elengedjük, így arra sincs különösebb okunk, hogy a szigmoid függvényhez ragaszkodjunk<sup>2</sup>. Jelöljük ehelyett az *aktivációs függvényt* általánosan  $g$ -vel, azaz egy neuron a következő számítást fogja megvalósítani:

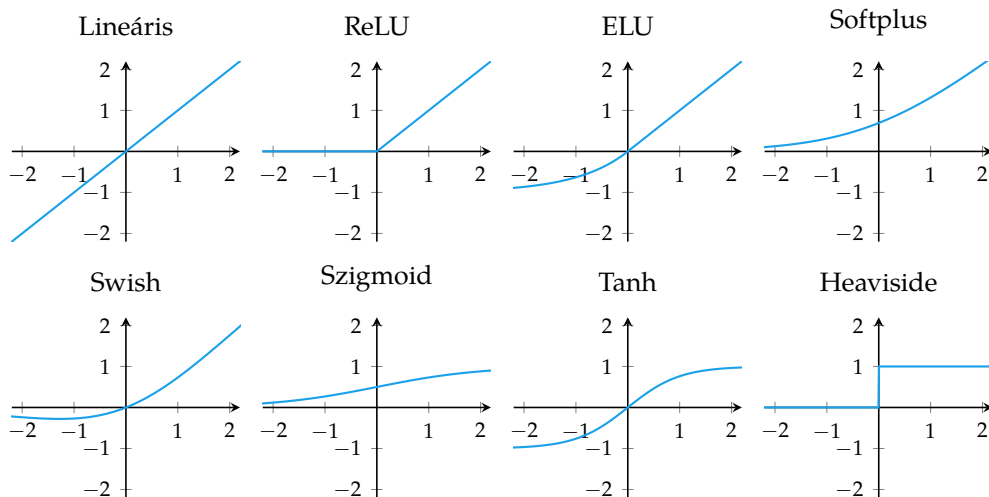
$$g(\mathbf{w}^\top \phi(\mathbf{x}_i)).$$

Néhány lehetőség  $g$ -re:

```
linear(a)    = a
ReLU(a)      = max(0, a)
ELU(a)       = ifelse(a < 0, exp(a) - 1, a)
softplus(a)  = log1p(exp(a))
swish(a)     = a / (1 + exp(-a))
Heaviside(a) = ifelse(a < 0, 0, 1)
```

4.1. algoritmus. Néhány aktivációs függvény (nem feltétlenül numerikusan stabil implementáció!).

Nézzük meg őket grafikonon is:



4.2. ábra. Gyakran használt aktivációs függvények.

A neuronra (perceptronra) mostantól úgy fogunk gondolni, mint elemi egységre, és elkezdünk bonyolultabb architektúrákat összeépíteni. Azzal kezdjük, hogy több neuront egy *rétegbe* (*layer*) szervezünk, ahol minden egyes neuron ugyanazokat a bemeneteket kapja meg. Írjuk fel az összes bemenő mintát mátrixos alakban soronként, ahogy megszoktuk:

$$\Phi = \begin{bmatrix} \cdots & \phi(\mathbf{x}_1)^\top & \cdots \\ \cdots & \phi(\mathbf{x}_2)^\top & \cdots \\ & \vdots & \\ \cdots & \phi(\mathbf{x}_N)^\top & \cdots \end{bmatrix},$$

majd a neuronok súlyvektorait is rendezzük mátrixba, de oszloponként:

$$\mathbf{W} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_n \\ \vdots & \vdots & & \vdots \end{bmatrix}.$$

Több neuron kimenetét a (3.5) egyenlethez hasonlóan kapjuk, de ezúttal a réteghez tartozó súlymátrixszal kell szoroznunk:

$$g(\Phi \mathbf{w}) \rightsquigarrow g(\Phi \mathbf{W}),$$

ahol  $g$  alkalmazása elemenként történik. Az eredmény egy olyan mátrix lesz, ahol a sorok száma megegyezik a bemenő minták számával, az oszlopok száma pedig a neuronok számával; az  $(i, j)$ -edik elem az  $j$ . neuron válasza az  $i$ . bemenetre<sup>3</sup>:

$$\begin{bmatrix} g(\phi(\mathbf{x}_1)^\top \mathbf{w}_1) & g(\phi(\mathbf{x}_1)^\top \mathbf{w}_2) & \cdots & g(\phi(\mathbf{x}_1)^\top \mathbf{w}_L) \\ g(\phi(\mathbf{x}_2)^\top \mathbf{w}_1) & g(\phi(\mathbf{x}_2)^\top \mathbf{w}_2) & \cdots & g(\phi(\mathbf{x}_2)^\top \mathbf{w}_L) \\ \vdots & \vdots & \ddots & \vdots \\ g(\phi(\mathbf{x}_N)^\top \mathbf{w}_1) & g(\phi(\mathbf{x}_N)^\top \mathbf{w}_2) & \cdots & g(\phi(\mathbf{x}_N)^\top \mathbf{w}_L) \end{bmatrix}.$$

Végül kapcsoljunk egymás után több réteget:

$$g^{(3)}(g^{(2)}(g^{(1)}(\Phi \mathbf{W}^{(1)}) \mathbf{W}^{(2)}) \mathbf{W}^{(3)}).$$

Itt ügyelnünk kell a dimenziók egyeztetésére a rétegek között: egy réteg kimenetében az oszlopok számának meg kell egyeznie a következő réteg súlymátrixában a sorok számával<sup>4</sup>. Az első (tehát a fenti képletben a legbelső) réteget szokás bemeneti, a legutolsót kimeneti, a közteseket pedig rejtett rétegeknek nevezni.

<sup>3</sup> Erre úgy is gondolhatunk, hogy a mátrix soraiban továbbra is transzformált mintákat tartalmaz, de a transzformáció most abban áll, hogy az adatot átküldtük még egy rétegnyi neuronon is. A így transzformált minta dimenzionalitása nyilvánvalóan megegyezik a réteg neuronszámával.

<sup>4</sup> Másképpen, a következő réteg neuronjainak fogadnia kell az előző rétegből származó „továbbtranszformált” mintákat.

### 4.3 Tanítás hibavisszaterjesztéssel

A neurális hálózatunk tanításánál a már megszokott módon járunk el:

1. A hálózatnak ismert bemenet-kimenet párokat mutatunk (tanító halmaz),
2. A hálózat kimeneteit összevetjük az elvárt kimenetekkel, ahol az eltérést a veszteségfüggvény méri,
3. A hálózat súlyait gradiens-alapon addig-addig hangoljuk, amíg el nem jutunk a veszteségfüggvény egy minimumához,
4. A tanítás után reménykedünk benne, hogy a hálózat új, ismeretlen mintákra is helyes válaszokat fog adni.

Az eljárás személtetéséhez vegyünk egy 100 mintából álló adathalmazt, ahol a minták 10 dimenziósak, valamint három neurális réteget, amelyek rendre 5, 3, 2 neuronból állnak. Jelölje a rétegek sorszámát ( $l$ ), az egyes rétegek kimenetét  $\mathbf{o}^{(l)}$ ,  $\mathbf{o}^{(0)} := \Phi$  pedig jelölje a bemenetet. Összefoglalva:

$l$	$\dim(\mathbf{W}^{(l)})$	$\mathbf{o}^{(l)}$	$\dim(\mathbf{o}^{(l)})$
(0)	—	$\Phi$	$100 \times 10$
(1)	$10 \times 5$	$g^{(1)}(\mathbf{o}^{(0)} \mathbf{W}^{(1)})$	$100 \times 5$
(2)	$5 \times 3$	$g^{(2)}(\mathbf{o}^{(1)} \mathbf{W}^{(2)})$	$100 \times 3$
(3)	$3 \times 2$	$g^{(3)}(\mathbf{o}^{(2)} \mathbf{W}^{(3)})$	$100 \times 2$

4.1. táblázat. Egy egyszerű neurális architektúra. Vegyük észre, hogy az utolsó réteg két neuront tartalmaz, tehát a megszokott példáinkkal ellentétben a neurális hálózatunknak most több kimenete van.

Vegyük most az  $L$  veszteségfüggvényt, azaz<sup>5</sup>

$$L(\mathbf{o}^{(3)}) = L(g^{(3)}(\mathbf{o}^{(2)} \mathbf{W}^{(3)}))$$

a hálózat prediktív teljesítményét minősíti. A veszteségfüggvény gradiense az utolsó réteg súlyai szerint a láncszabály felhasználásával számolható. Vegyük észre, hogy a  $\mathbf{W}^{(3)}$  súlymátrix az  $\mathbf{o}^{(3)}$  kimenetben van elrejtve, azaz „át kell differenciálnunk” az  $L$  veszteségfüggvényen, a  $g^{(3)}$  aktivációs függvényen és az  $\mathbf{o}^{(2)} \mathbf{W}^{(3)}$  szorzaton<sup>6</sup>:

$$\underbrace{\nabla_{\mathbf{W}^{(3)}} L}_{3 \times 2} = \underbrace{\mathbf{o}^{(2) \top}}_{3 \times 100} \underbrace{(\nabla_{\mathbf{o}^{(3)}} L \circ g^{(3)'}(\mathbf{o}^{(2)} \mathbf{W}^{(3)}))}_{100 \times 2},$$

<sup>5</sup> Az egyszerűség kedvéért az elvárt kimeneteket kihagytuk a notációból.

<sup>6</sup> Ellenőrzésképpen megjelenítettük az egyes részeredmények méretét is. Látjuk, hogy a gradiens mérete megegyezik a súlymátrixéval, úgyhogy valószínűleg nem rontottuk el a számolást.

ahol  $\circ$  elemenkénti szorzást jelent. A többi súlymátrix szerinti gradienshez csupán a láncszabállyal kell egyre beljebb és beljebb lépkednünk. A  $\mathbf{W}^{(2)}$  súlymátrixhoz például  $\mathbf{o}^{(2)}$ -n keresztül jutunk el, a korábbi deriváltak pedig a láncszabály értelmében szorzótényezőként fognak megjelenni. Sokat egyszerűsít, ha egyes részeredményeket elnevezünk:

$$\delta^{(3)} := \nabla_{\mathbf{o}^{(3)}} L \circ g^{(3)'}(\mathbf{o}^{(2)} \mathbf{W}^{(3)}).$$

A  $\mathbf{W}^{(2)}$  szerinti gradiens így

$$\nabla_{\mathbf{W}^{(2)}} L = \underbrace{\mathbf{o}^{(1)\top}}_{5 \times 3} \left( \underbrace{\delta^{(3)}}_{5 \times 100} \underbrace{\mathbf{W}^{(3)\top}}_{100 \times 2} \circ \underbrace{g^{(2)'}(\mathbf{o}^{(1)} \mathbf{W}^{(2)})}_{100 \times 3} \right).$$

Ismét elnevezve

$$\delta^{(2)} := \delta^{(3)} \mathbf{W}^{(3)\top} \circ g^{(2)'}(\mathbf{o}^{(1)} \mathbf{W}^{(2)})$$

az első réteg súlymátrixa szerinti gradiens<sup>7</sup>

$$\nabla_{\mathbf{W}^{(1)}} L = \underbrace{\mathbf{o}^{(0)\top}}_{10 \times 5} \left( \underbrace{\delta^{(2)}}_{10 \times 100} \underbrace{\mathbf{W}^{(2)\top}}_{100 \times 3} \circ \underbrace{g^{(1)'}(\mathbf{o}^{(0)} \mathbf{W}^{(1)})}_{100 \times 5} \right).$$

Készen vagyunk az összes réteggel. A súlymátrixok frissítését a legegyszerűbb esetben a (3.4) egyenletben megismert módon végezhetjük, azaz

$$\mathbf{W}^{(n)} \leftarrow \mathbf{W}^{(n)} - \eta \nabla_{\mathbf{W}^{(n)}} L,$$

ahol  $\eta$  továbbra is a tanulási tényezőt jelöli.

<sup>7</sup> Látjuk, hogy a gradienseket a hálózat kimenete felől, a bemenet felé haladva tudjuk kiszámolni a hiba visszafelé „terjesztésével” (*backpropagation*), ahonnan az algoritmus neve is származik. A 4.8. szakaszban látni fogjuk, hogy a gondolat ennél sokkal általánosabb, valójában visszafelé-módú automatikus differenciálásról van szó.

```
struct MLP
  net
end
MLP(
  Chain(
    Dense(10,5,relu),
    Dense(5,3,relu),
    Dense(3,2)
  )
)
```

4.2. algoritmus. A példában szereplő többrétegű neurális hálózat (multilayer perceptron) megvalósítása.

#### 4.4 Aktivációs függvények és veszteségfüggvények

Tegyük fel, hogy regressziót szeretnénk végezni neurális hálózatokkal. Az előző szakaszban tárgyalt tanítási eljáráshoz szükségünk van konkrét aktivációs függvényekre és veszteségfüggvényre. Emlékezzünk vissza a 2. fejezet regressziós modelljeire, ahol valamilyen  $\mathbf{x}_i \in \mathbb{R}^D$  bemenetekre és  $y_i \in \mathbb{R}$  kimenetekre olyan  $f$  függvényt kerestünk, ami bizonyos értelemben jól megragadja a kettő közötti kapcsolatot, azaz

$$y_i \approx f(\mathbf{x}_i).$$

Az  $f$  függvény lehetett például lineáris ( $\mathbf{w}^\top \mathbf{x}_i + b$ ), polinomiális ( $\mathbf{w}^\top \phi_{poly}(\mathbf{x}_i)$ ), vagy akár valami sokkal bonyolultabb is. Esetünkben – a későbbi tárgyalásmód megkönnyítése érdekében – gondoljunk  $f$ -re úgy, mint egy tetszőlegesen bonyolult neurális hálózatra, ám az utolsó réteg aktivációs függvénye nélkül.

A regressziónál feltettük, hogy az  $y_i \approx f(\mathbf{x}_i)$  kapcsolat valamilyen normális eloszlású zajtól eltekintve egyenlőséggel teljesül; majd felírtuk a likelihood-ot, amelyből a veszteségfüggvényt is megkaptuk, nevezetesen

$$p(y_i | \mathbf{x}_i) = \mathcal{N}(y_i | f(\mathbf{x}_i), \beta^{-1}) \xrightarrow{-\ln} \frac{\beta}{2} (y_i - f(\mathbf{x}_i))^2.$$

A neurális hálózatok nyelvére lefordítva ez annyit tesz, hogy regressziós esetben a veszteségfüggvényt négyzetes hibának választjuk, az utolsó réteg kimenetével pedig nem is kell semmi továbbit kezdenünk<sup>8</sup>. A rejtett rétegek aktivációs függvényeiről még nem szóltunk; itt kevesebb megkötés van, manapság leginkább a *ReLU* függvényt „szokás” választani.

Nézzük most az osztályozást, ahol az  $y_i \in \{0, 1\}$  címkékkal némi valószínűségelméleti barkácsolást követően<sup>9</sup> a  $\sigma$  függvényre jutottunk:

$$p(y_i = 1 | \mathbf{x}_i) = \frac{e^{f(\mathbf{x}_i)}}{1 + e^{f(\mathbf{x}_i)}} = \sigma(f(\mathbf{x}_i)),$$

azaz az utolsó rétegben még egy szigmoid aktivációs függvényt kell használnunk; az így felépített hálózat az  $y_i = 1$  osztályba tartozás valószínűségét jósolja. Az  $y_i = 1$  és  $y_i = 0$  eseteket egybeépítve a likelihood Bernoulli-nak, a veszteségfüggvény pedig *bináris keresztentrópiának* adódott:

$$p(y_i | \mathbf{x}_i) = \text{Bern}(y_i | \sigma(f(\mathbf{x}_i))) = \sigma(f(\mathbf{x}_i))^{y_i} \cdot (1 - \sigma(f(\mathbf{x}_i)))^{1-y_i} \\ \xrightarrow{-\ln} -y_i \ln \sigma(f(\mathbf{x}_i)) - (1 - y_i) \ln(1 - \sigma(f(\mathbf{x}_i))).$$

<sup>8</sup> Ez egyenértékű azzal, hogy az utolsó réteg aktivációs függvénye lineáris. A 4.2. ábráról nem is tudnánk mást választani, hiszen  $f$ -nek az egész  $\mathbb{R}$ -be kell képeznie; más esetben előfordulhatna olyan  $y_i$ , amit a hálózat soha nem lesz képes „eltalálni”.

<sup>9</sup> Lásd a 3.1. szakaszt.

Hogyan válasszuk meg az aktivációs függvényt, ha többsztályos osztályozást szeretnénk végezni? Legyen  $y_i \in \{1, 2, \dots, K\}$ . A korábbiakhoz hasonlóan itt is egy valószínűségi modelltől érdemes elindulnunk:

$$p(y_i | \mathbf{x}_i) = ?$$

Ehhez a kétosztályos gondolatmenetet általánosítjuk  $K$  osztályra, és a hálózat kimenetét  $K$  méretűre választjuk. Jelölje  $f^k(\mathbf{x}_i)$  a kimenet  $k$ . komponensét. Ekkor a  $k$ . osztályra a 3.1. szakaszban megismert egyszerűsítéssel

$$p(y_i = k | \mathbf{x}_i) = \frac{p(\mathbf{x}_i | y_i = k) p(y_i = k)}{p(\mathbf{x}_i | y_i = 1) p(y_i = 1) + p(\mathbf{x}_i | y_i = 2) p(y_i = 2) + \dots + p(\mathbf{x}_i | y_i = K) p(y_i = K)}$$

$$= \frac{e^{f^k(\mathbf{x}_i)}}{e^{f^1(\mathbf{x}_i)} + e^{f^2(\mathbf{x}_i)} + \dots + e^{f^K(\mathbf{x}_i)}} := \text{softmax}^k(f(\mathbf{x}_i)),$$

ahol az ún. *softmax* függvényt definiáltuk:

$$\text{softmax}(f(\mathbf{x}_i)) = \begin{bmatrix} \frac{e^{f^1(\mathbf{x}_i)}}{\sum_{j=1}^K e^{f^j(\mathbf{x}_i)}} \\ \frac{e^{f^2(\mathbf{x}_i)}}{\sum_{j=1}^K e^{f^j(\mathbf{x}_i)}} \\ \vdots \\ \frac{e^{f^K(\mathbf{x}_i)}}{\sum_{j=1}^K e^{f^j(\mathbf{x}_i)}} \end{bmatrix}.$$

A softmax kimenetének  $k$ . komponense tehát a  $k$ . osztályba tartozás valószínűségét jelenti. Természetesen a numerikus stabilitást itt is érdemes szem előtt tartani.

```
function softmax(a)
    m = maximum(a)
    ea = exp.(a .- m)
    return ea ./ sum(ea)
end
function logsoftmax(a)
    m = maximum(a)
    ea = exp.(a .- m)
    return a .- log(sum(ea)) .- m
end
```

4.3. algoritmus. Softmax aktivációs függvény és logaritmusának numerikusan stabil implementációja. A túlcsordulás elkerülése érdekében levonjuk a bemenet maximumát, amely a softmax függvény értékén nem változtat (bizonyítsuk be!). Hasonló trükköt alkalmazunk a logaritmus kiszámításakor is.



A kétosztályos esethez hasonlóan a  $p(y_i = k | \mathbf{x}_i)$  valószínűségeket egybeépíthetjük, ha Bernoulli helyett a kategorikus eloszlást<sup>10</sup> használjuk:

$$p(y_i | \mathbf{x}_i) = \text{Cat}(y_i | \text{softmax}(f(\mathbf{x}_i))) = \prod_k \text{softmax}^k(f(\mathbf{x}_i))^{y_i^k}.$$

<sup>10</sup> Amit ezen okból kifolyólag tréfásan Multinoulli-nak is szoktak nevezni.

Az  $y$  változóra – talán kissé zavart keltő módon – itt úgy gondolunk, mint egy vektorértékű indikátor-változóra, amely csupa zérust és egyetlen 1-est tartalmaz a „megfelelő” helyen<sup>11</sup>, azaz például

$$y_i = 2 \quad \rightsquigarrow \quad y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

<sup>11</sup> Ezt nevezzük *one-hot* kódolásnak.

A veszteségfüggvényt a megszokott módon, a likelihood negatív logaritmusaként kapjuk:

$$-\ln p(y_i | \mathbf{x}_i) = -\sum_k y_i^k \cdot \ln(\text{softmax}^k(f(\mathbf{x}_i))) = -\sum_k y_i^k \cdot \left( f^k(\mathbf{x}_i) - \ln \sum_j e^{f^j(\mathbf{x}_i)} \right),$$

és *kategorikus keresztentrópiának* nevezzük.

Érdekes megfigyelni, hogy valójában mit is büntet ez a veszteségfüggvény. A jobb oldalon szereplő  $\log \text{sumexp}$  mennyiségről például tudjuk<sup>12</sup>, hogy

$$\ln \sum_j e^{f^j(\mathbf{x}_i)} \approx \max_j f^j(\mathbf{x}_i),$$

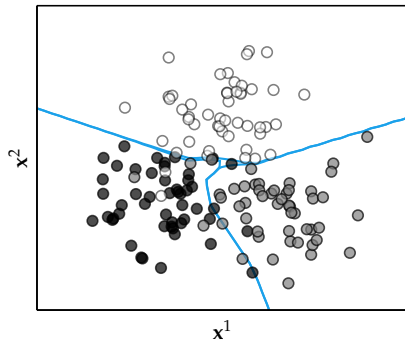
<sup>12</sup> Gondoljunk arra, hogy a szummát az exponenciális függvény miatt a legnagyobb komponens dominálja.

és a

$$\sum_k y_i^k \cdot (\dots)$$

tag pedig „kiválasztja” a valódi osztályt; minden más osztály a szummában zérus együtthatóval szerepel<sup>13</sup>. A kettőt összevetve arra jutunk, hogy ha a  $k$ . az  $\mathbf{x}_i$  minta valódi osztálya, és  $f^k(\mathbf{x}_i)$  értéke eltér a  $\max_j f^j(\mathbf{x}_i)$  kifejezés értékétől, zérusnál nagyobb veszteséget kapunk. Más szóval, a veszteségfüggvény azt bünteti, ha a neurális hálózat kimenetének legnagyobb komponense nem a  $k$ -adik.

<sup>13</sup> Továbbra is *one-hot* kódolás mellett.



4.3. ábra. Háromosztályos osztályozás többrétegű neurális hálózattal és softmax aktivációs függvényvel.

#### 4.5 Regularizáció neurális hálózatokban

A neurális hálózatok különösen hajlamosak a túlilleszkedésre. A problémát többféleképpen is orvosolhatjuk; a teljesség igénye nélkül néhány technika<sup>14</sup>:

- A neurális architektúra megszorítása (például rétegek számának, méretének csökkentése),
- Korai leállítás (a tanítás megállítása, amikor a prediktív teljesítmény egy dedikált *validációs halmazon* mérve már nem növekszik),
- Zaj hozzákeverése az általánosítóképesség növelése érdekében,
- A neuronok véletlenszerűen kiválasztott részhalmazainak ki-be kapcsolása (*dropout*),
- A neurális hálózat súlyainak korlátozása (*weight decay*).

A súlyok megszorítása a lineáris regresszió regularizált változatához hasonlóan történik. A szemléltetéshez használjunk egyetlen neuront, amelynek súlyvektorára priort teszünk:

$$p(y | \mathbf{x}, \mathbf{w}) = \mathcal{P}(y | f(\mathbf{w}^\top \phi(\mathbf{x}))),$$

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I}),$$

a poszterior így

$$p(\mathbf{w} | \mathbf{x}, y, \alpha) \propto p(y | \mathbf{x}, \mathbf{w}) \cdot p(\mathbf{w} | \alpha).$$

<sup>14</sup> Ennek a jegyzetnek nem célja egy általános neurális hálózat/deep learning kurzus anyagának felölelése, úgyhogy a technikákat nem részletezzük.

A veszteségfüggvény a poszterior negatív logaritmusaként adódik<sup>15</sup>:

$$\begin{aligned} -\ln p(\mathbf{w} \mid \mathbf{x}, y, \alpha) &= -\ln p(y \mid \mathbf{x}, \mathbf{w}) - \ln p(\mathbf{w} \mid \alpha) \\ &= \underbrace{-\ln p(y \mid \mathbf{x}, \mathbf{w})}_{\text{korábbi loss}} + \underbrace{\frac{\alpha}{2} \|\mathbf{w}\|_2^2}_{\text{regularizáció}}, \end{aligned}$$

<sup>15</sup> Tehát most is MAP megoldást keresünk.

azaz a korábbi veszteségfüggvényünk egy extra regularizációs taggal bővül<sup>16</sup>.

Attól függően, hogy milyen priort használunk (a fenti példában normális eloszlásút), a regularizációs tag is más-más formát ölthet. Néhány gyakrabban használt példány:

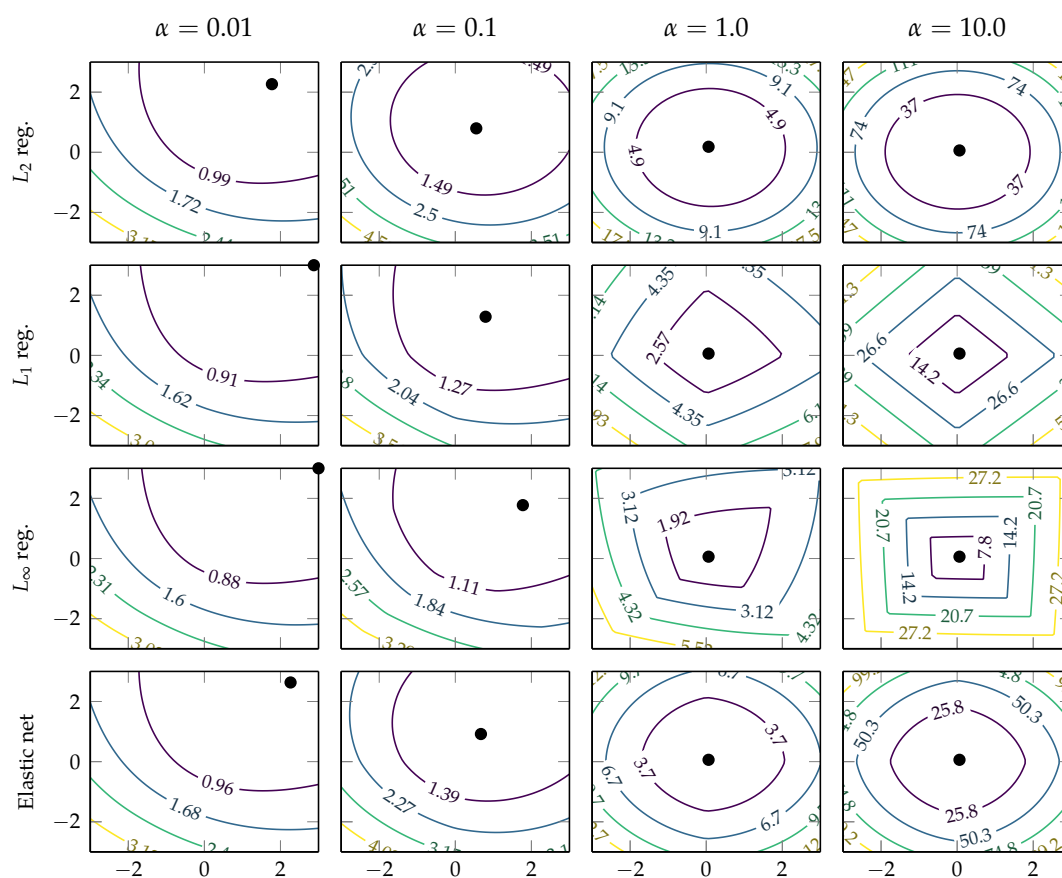
- $\|\cdot\|_2$  ( $L_2$ -norma, normális eloszlású prior). A leggyakrabban használt regularizációs tag, amely a súlyokat egyenletesen csökkenti.
- $\|\cdot\|_1$  ( $L_1$ -norma, Laplace-eloszlású prior). Akkor érdemes használni, ha ún. *ritka* megoldást keresünk; ez a fajta regularizáció úgy csökkenti a súlyokat, hogy minél több legyen zérus.
- $\|\cdot\|_\infty$  ( $L_\infty$ -norma vagy max-norma). Egzotikusabb, kevésbé agresszív regularizáció, amely csak a súlyok legnagyobbikát csökkenti („clipping”).
- Elastic net. Az  $L_1$  és  $L_2$  normák lineáris kombinációja, amely a 2010-es évek közepéig volt igazán népszerű.

<sup>16</sup> Figyeljük meg, hogy a megoldásnak két némiképp ellentétes célnak kell megfelelnie: egyrészt a veszteséget kell minimalizálni, másrészt a súlyokat kell bizonyos értelemben kicsin tartani. A kettő közötti egyensúlyt az  $\alpha$  regularizációs együttható szabályozza.

A regularizációs tagok hatását a 4.4. ábra szemlélteti.

#### 4.6 Optimalizáció neurális hálózatokban

A korábbi fejezetekben a modelljeink viszonylag egyszerű veszteségfüggvénnyel bírtak; vagy volt analitikus megoldásunk, vagy legalábbis paramétereiben lineáris modellekkel foglalkoztunk. Ha a modell paramétereiben lineáris, a veszteség pedig konvex függvénye a kimenetnek, akkor a veszteségfüggvény, mint a paraméterek függvénye is konvex lesz, amit könnyű minimalizálni. Általában viszont – a többi rétegnek köszönhetően – a neurális modellek már a súlyokban sem lineárisak, a veszteségfüggvényük nagyon bonyolult lehet, ami igencsak megnehezíti az optimalizációt.



4.4. ábra. Kétdimenziós súlyvektorral rendelkező regularizált modellek veszteségfüggvénye különböző típusú regularizációs tagok és regularizációs együtthatók mellett. A minimumot fekete pont jelöli. Érdeemes megfigyelní, hogy mindegyik tag igyekszik az origó felé „terelni” súlyvektort, ám mindegyik másképpen.

#### 4.6.1 Nehézségek

A tipikus anomáliák sorba szedéséhez célszerű a veszteségfüggvényt egy pont körül Taylor-sorba fejteni<sup>17</sup>. Jelölje  $\theta$  a neurális hálózat összes súlyát, ekkor

$$L(\theta) \approx L(\theta_0) + \nabla_{\theta} L \Big|_{\theta_0} (\theta - \theta_0) + \frac{1}{2} (\theta - \theta_0)^{\top} \mathbf{H} \Big|_{\theta_0} (\theta - \theta_0).$$

A felülemülő problémákhoz vizsgáljuk meg a  $\mathbf{H}$  Hesse-mátrix sajátértékeit.

*Lokális minimumok.* Ha  $\nabla_{\theta} L = 0$  és  $\mathbf{H}$  összes sajátértéke pozitív, lokális minimumban vagyunk. Ez általában nem okoz problémát; a lokális minimumok a gyakorlatban ritkák, és még mindig „elég jók” ahhoz, hogy kielégítő prediktív teljesítményt kapjunk.

*Nyeregponatok.* Ha  $\nabla_{\theta} L = 0$  és a  $\mathbf{H}$  mátrixnak vannak pozitív és negatív sajátértékei is, ún. nyeregpontra vagyunk. Ezek gyakoribbak a lokális minimumoknál, és bizonyos algoritmusoknak gondot okozhatnak<sup>18</sup>. Ritka azonban az az eset, hogy pontosan egy nyeregpontra érkezve ragadna be az algoritmus (leginkább a súlyok helytelen inicializálásánál fordul elő).

*Platók.* Ha  $\nabla_{\theta} L \approx 0$  és  $\mathbf{H}$  sajátértékei is zérus közelében vannak, a veszteségfüggvény lokálisan lapos. Az efféle kiterjedt platókról az optimalizációs algoritmus csak nagyon lassan, vagy egyáltalán nem képes kijutni. Ilyen jelenség léphet fel például, ha mindenütt szigmoid aktivációs függvényt használunk, ám történetesen az összes aktiváció a szigmoid függvény lapos részére kerül<sup>19</sup>. Éppen ezért újabban a rejtett rétegekben ReLU aktivációt szokás használni, amelynek képe zérus felett lineáris, tehát nem lapul el és erős gradiensünk lesz. Az optimalizáció azonban még így is beragadhat, ha túl sok neuron kerül kikapcsolt állapotba (ReLU zérus alatti része).

*Keskeny völgyek, szakadékok.* A leggyakrabban felmerülő probléma az, ha  $\mathbf{H}$  sajátértékei között sok nagyságrend eltérés van<sup>20</sup>. A mátrix kondíciószáma a

$$\kappa(\mathbf{H}) = \frac{|\lambda_{\max}(\mathbf{H})|}{|\lambda_{\min}(\mathbf{H})|}$$

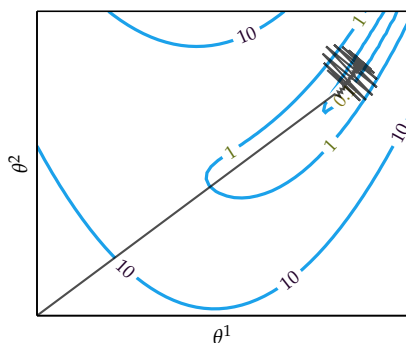
<sup>17</sup> Feltesszük, hogy a veszteségfüggvény kétszer differenciálható; a sorba fejtés lényegében azt árulja el nekünk, hogy „lokálisan hogyan néz ki” a veszteségfüggvény.

<sup>18</sup> A klasszikus másodrendű módszerek például előszeretettel ragadnak be nyeregpontokba.

<sup>19</sup> A tankönyvek rendszerint ezt jelölik meg az „eltűnő gradiens” (*vanishing gradient*) probléma klasszikus okaként, különösen, ha ráadásul négyzetes hibát használunk veszteségfüggvénynek.

<sup>20</sup> Intuitíve arról van szó, hogy bizonyos irányokban a függvény nagyon gyorsan változik, míg más irányokban szinte egyáltalán nem.

hányados; a magas kondíciószámmal rendelkező mátrixot – és a kapcsolódó optimalizálási problémát – rosszul kondicionáltnak nevezzük. A gradiens-alapú módszerek ilyenkor csak rendkívül lassan, a völgy szélei között ide-oda pattogva, oszcillálva képesek haladni.



4.5. ábra. Gradiens-alapú optimalizáció rosszul kondicionált Hesse-mátrix esetén. Az algoritmus oszcilláló mozgásra kényszerül.

A jelenség magyarázata az, hogy a gradiens mindig merőleges a szintvonalakra; ha a völgy túl keskennyé válik, az algoritmus minden lépésben „túlszalad”, a következő lépésben pedig ellenkező irányú korrekcióra kényszerül. A probléma elkerüléséhez választhatunk alacsonyabb tanulási tényezőt, ám ez szintén a tanulást lassítja. Másik megoldás az ún. *batch normalizáció* használata, ami a neurális hálózat összes rétegének kimenetét zérus átlagra és egységnyi varianciára normalizálja, a tanulás során pedig ezeket az extra műveleteket a 4.3. szakaszban látottakhoz hasonlóan visszaterjeszti; a völgyek ekkor jobban navigálhatóvá válnak.

#### 4.6.2 Algoritmusok

Az előző szakaszban tárgyalt problémák elkerüléséhez valamivel bonyolultabb algoritmusokra van szükség. Kézenfekvőnek tűnhet másodrendű módszerek, például a Newton–Raphson módszer többdimenziós analógjainak alkalmazása. A neurális hálózatok azonban kifejezetten akkor hasznosak, ha nagy mennyiségű tanító adat áll rendelkezésre, ez pedig jórészt kizárja a másodrendű módszereket – a Hesse-mátrix tárolása például a minták számában négyzetes költségű<sup>21</sup>.

Valójában éles adaton a gradiens-módszer sem használható eddig tárgyalt formájában, hacsak nincs nagyon sok memóriával rendelkező szuperszámítógépünk.

<sup>21</sup> Természetesen léteznek approximatív megoldások (pl. az L-BFGS algoritmus), amelyek időnként neurális hálózatok kapcsán is előkerülnek.

A gyakorlatban a tanító adatokat inkább nem egyszerre, hanem kisebb adagokban (ún. *minibatch-ekben*) adjuk az algoritmusnak, így a tárkomplexitás a töredékére csökken<sup>22</sup>. Mivel a minibatch-ek kiválasztása általában véletlenszerűen történik, a gradiens-módszer ezen variánsát *Stochastic Gradient Descent* (SGD) névvel illetik. A minták függetlenségét kihasználva

$$\nabla_{\theta} L(f(\mathbf{X}, \theta), \mathbf{y})) = \nabla_{\theta} \sum_i L(f(\mathbf{x}_i, \theta), y_i) = \sum_i \nabla_{\theta} L(f(\mathbf{x}_i, \theta), y_i),$$

azaz a gradiens elméletileg előállítható a minibatch-eken vett gradiensek összegeként. A gyakorlatban minden egyes minibatch-et súlyfrissítés követ; az algoritmus sztochasztikus természetének köszönhetően immár nem mindig a tényleges – teljes adaton számolt – gradiens irányába mozdulunk el, csupán „átlagosan” haladunk egy minimum felé.

```
struct GradientDescent
    η # tanulási tényező

    GradientDescent(θ; η=0.001) = new(η)
end
function step!(opt::GradientDescent, θ, ∇f)
    return @. θ - opt.η * ∇f
end
```

<sup>22</sup> Ez gyakorlatilag elkerülhetetlen, ha például GPU-n szeretnénk tanulni; általában 10-1000 méretű minibatch-ekkel dolgozunk. A minibatch-méret eldöntése általában kísérletezés kérdése, érdemes a hardver optimális kihasználására törekedni

4.4. algoritmus. Gradiens-módszer egyszerű megvalósítása.

Mind a lokális minimumokból, nyeregpontokból, platókból való kiszabadulásban, mind az oszcillációk visszafogásában segít a *momentum-módszer*, amely „lendületet” ad az algoritmusnak:

```
struct Momentum
    η # tanulási tényező
    α # momentum csillapítás
    v # momentum vektor

    Momentum(θ; η=0.001, α=0.9) = new(η, α, zeros(length(θ)))
end
function step!(opt::Momentum, θ, ∇f)
    @. opt.v = opt.α*opt.v + opt.η*∇f
    return @. θ - opt.v
end
```

4.5. algoritmus. Momentum-módszer. Egy lépés az aktuális gradiens és az előző lépés lineáris kombinációjaként adódik, amely túllendítheti az algoritmust a lokális minimumokon, valamint az oszcillációkat is mérsékli.

A tanulási tényező megválasztása nehéz kérdés<sup>23</sup>: túl kis érték esetén a hálózat nagyon lassan vagy egyáltalán nem tanul, túl nagy érték esetén pedig össze-vissza ugrál a hibafelületen és nem konvergál (különösen a kis minibatch-méretből adódó „zajos” lépések esetén). A megfelelő érték attól is függ, hogy a tanulás melyik fázisában járunk: az elején még távol vagyunk az optimumtól és megengedhetünk nagyobb ugrásokat, a vége felé pedig remélhetőleg valamilyen optimum közelébe kerültünk, tehát apró lépésekkel kell finomhangolnunk.

Ezt az ötletet használja a tanulási tényező adaptív beállítására az *AdaGrad* algoritmus:

```
struct AdaGrad
    η # tanulási tényező
    G # akkumulált gradiens

    AdaGrad(θ; η=0.1) = new(η, zeros(length(θ)))
end
function step!(opt::AdaGrad, θ, ∇f)
    @. opt.G += ∇f^2
    return @. θ - opt.η/sqrt(opt.G) * ∇f
end
```

<sup>23</sup> A tanulási tényező beállításának bevált módszere a GSD (Grad Student Descent): bízzuk a munkát egy doktoranduszra, aki addig csavargatja, amíg egyszer csak működik.

4.6. algoritmus. AdaGrad algoritmus a tanulási tényező adaptív beállításával. Minden iterációnál leosztunk az akkumulálódó gradienssel, a lépések így egyre kisebbek lesznek.

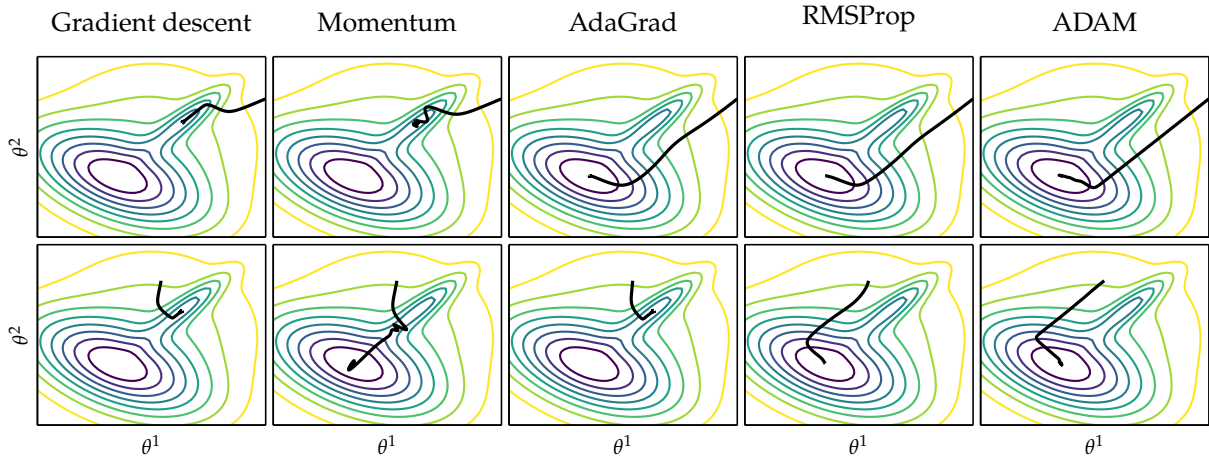
Az AdaGrad algoritmus módosított változata az *RMSProp*, amely valamivel bonyolultabb sémát alkalmaz a tanulási tényező adaptív változtatására:

```
struct RMSProp
    η # tanulási tényező
    α # csillapítás
    G # akkumulált gradiens

    RMSProp(θ; η=0.01, α=0.9) = new(η, α, zeros(length(θ)))
end
function step!(opt::RMSProp, θ, ∇f)
    @. opt.G = opt.α*opt.G + (1-opt.α)*∇f^2
    return @. θ - opt.η/sqrt(opt.G) * ∇f
end
```

4.7. algoritmus. Az RMS-Prop algoritmus exponenciális lecsengetést alkalmaz a gradiens-akkumulációnál.





4.6. ábra. Optimalizációs algoritmusok konvergenciája a veszteségfüggvény szintvonaláival, különböző helyekről indítva.

A korábbi ötleteket ötvözi az *ADAM* algoritmus, amely adaptív tanulási tényezővel, első- és másodrendű momentumbecslésekkel dolgozik:

```
mutable struct ADAM
    η # tanulási tényező
    α # csillapítás
    β # csillapítás
    v # első momentum
    G # második momentum
    i # iteráció

    ADAM(θ; η=0.01, α=0.9, β=0.9) =
        new(η, α, β, zeros(length(θ)), zeros(length(θ)), 0)
end
function step!(opt::ADAM, θ, ∇f)
    @. opt.v = opt.β*opt.v + (1-opt.β)*∇f
    @. opt.G = opt.α*opt.G + (1-opt.α)*∇f^2
    opt.i += 1
    v = opt.v ./ (1-opt.α^opt.i)
    G = opt.G ./ (1-opt.β^opt.i)
    return @. θ - opt.η/sqrt(G) * v
end
```

4.8. algoritmus. Az ADAM algoritmus.

Végül a 4.9. ábra összefoglalja a tanítás menetét.

```

function train!(mlp::MLP, x, y; epochs=10, batchsize=512)
    ps = params(mlp.net)
    opt = ADAM()
    data = DataLoader((x,y),batchsize=batchsize,shuffle=true)

    for epoch in 1:epochs, batch in data
        x_,y_ = batch
        grad = gradient(ps) do
            logitcrossentropy(mlp.net(x_),y_)
        end
        update!(opt, ps, grad)
    end
end

```

4.9. algoritmus. Többosztályos osztályozó neurális hálózat tanítása a Flux könyvtár felhasználásával. Az adatokat 512 méretű minibatchekben adagoljuk, a tanításhoz az ADAM algoritmust használjuk. Vegyük észre, hogy a gradienst nem nekünk kell megadni; a 4.8. szakaszban látni fogjuk, hogyan lehet automatikusan kiszámolni. A tanítás így néhány sornyi kóddal megoldható.

## 4.7 Konvolúciós neurális hálózatok

A valós adatelemzési, gépi tanulási problémák egyik legjellemzőbb vonása az, hogy az adat több szabadsági fokkal rendelkezik, mint maga az adatot generáló „valóság”. Egy orvosi adatelemzési feladatban például egy-egy páciens sokdimenziós leírása nem vehet fel akármilyen értékeket; gondoljunk csak a testsúly, koleszterinszint és vérnyomás összefüggéseire. Úgy is mondhatjuk, hogy az adatokat tartalmazó vektortérben az adatok egy alacsonyabb dimenziójú sokaságon helyezkednek el<sup>24</sup>, a látott értékek például egy közös okra, a súlyos elhízásra vezethetők vissza.

Hasonló jelenséget látunk képfeldolgozási feladatokban, ahol a szomszédos vagy egymás közelében lévő képpontok színe függ össze, például mert egy kutyát vagy arcot ábrázolnak. Minderről az eddig megismert, *teljesen összekötött*<sup>25</sup> neurális hálózatok mit sem tudnak, hiszen minden neuron részesül a teljes bemenetből. A lokális megőrzéséhez tehát olyan neurális hálózatot kell építenünk, ahol egy-egy neuron a képnek csak egy kisebb foltjából kap bemenetet<sup>26</sup>. Ezt a foltot a neuron *receptív mezejének* is nevezik.

A képi adatok további jellemzője a *térbeli invariancia*, azaz az objektumok többé-kevésbé ugyanúgy festenek, bárhol is tűnnek fel a képen. Ha egy neuron megtanult valamilyen alakzatot felismerni a maga receptív mezejében (például a bal felső sarokban), a jobb alsó sarokért felelős neuron nagyjából ugyanazzal a súlyvektorral ismerheti fel az alakzatot. Más szóval nemcsak, hogy a súlyok nagy része zérus, hanem súlyokon több neuron is osztozhat. Egy-egy réteg súlymátrixa tehát

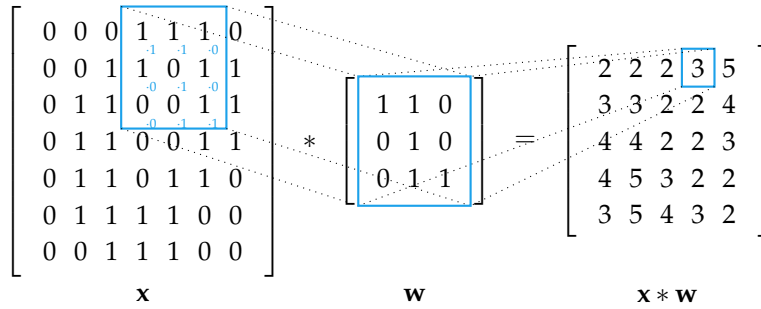
<sup>24</sup> Az efféle látens struktúra felderítésére több módszert is látunk a 10. fejezetben.

<sup>25</sup> Szakkifejezéssel *dense* vagy *fully connected*, ami alatt azt értjük, hogy a következő réteg egy neuronja a megelőző réteg összes neuronjából kap bemenetet

<sup>26</sup> Ennek kellemes következménye, hogy a paraméterszám is radikálisan csökken; a réteghez tartozó súlymátrix ritka, elemeinek többsége zérus.

speciális szerkezettel rendelkezik: egyrészt ritka, másrészt sok ismétlődő elemet tartalmaz<sup>27</sup>.

A konvolúciós neurális hálózat (CNN) ezeket az ötleteket a következőképpen valósítja meg. Az ismétlődő súlyok  $K \times K$  méretű mátrix formáját öltik, amelyet *kernel*nek vagy *filter*nek nevezünk. Jelöljük  $\mathbf{w}$ -vel ( $K$  tipikusan 3-10 méretű). Ezt a kernelt „toljuk végig” az  $\mathbf{x}$  bemeneten, páronként összeszorozva a kép és a kernel megfelelő pozícióján található értékeket, majd összegzünk, ahogy a neurális hálózatoknál már megszoktuk. A műveletet konvolúciónak nevezzük, és a következő ábra szemlélteti:



<sup>27</sup> A gyakorlatban ezért a réteg kimenetének számolását nem is a szokásos mátrix-szorzással valósítjuk meg, hiszen az túl pazarló volna.

4.7. ábra. Konvolúció  $3 \times 3$  kernel-mérettel. Figyeljük meg, hogy alapesetben a kimenet mérete csökken a bemenethez képest.

Formulával<sup>28</sup>

$$(\mathbf{x} * \mathbf{w})^{ij} = \sum_{m=-K}^K \sum_{n=-K}^K \mathbf{x}^{i+m, j+n} \cdot \mathbf{w}^{mn}$$

Korábbi tanulmányaink során láttunk már hasonlót. Vízszintes irányú élek kiemeléséhez például használható a

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel-kernel. A konvolúciós neurális hálózatok hasonló kerneleket alkalmaznak, egy képi régióra jellemzően több különbözőt is (*csatornák*). A trükk az, hogy a kernelek most egy neurális hálózat súlyrendszerének felelnek meg, tehát gradiens-alapon tanulhatók. A tanítás során a súlyok úgy állnak be, hogy a hálózat megtanulja felismerni az aktuális feladat megoldásához – például képek osztályozásához – fontos képi jegyeket. Az egymás után csatolt konvolúciós

<sup>28</sup> A kép széleinek kezelése többféleképpen történhet. Ha ugyanakora képet szeretnénk kapni, mint a bemenet, *padding*-et kell végeznünk, azaz a bemenet széleit virtuálisan szegélyezzük zérussal (*zero padding*), vagy az értékek ismétlésével (*same padding*). Ha kifejezetten csökkenteni akarjuk a képméretet (például kis paraméterszámra törekszünk), a kernelt használhatjuk nagyobb lépésközzel (*stride*).

rétegek gyakran megtanulnak egyre bonyolultabb alakzatokat felismerni; míg a korai rétegek elemi formákat, a későbbiek egyre komplexebb és absztraktabb alakzatokat detektálnak.

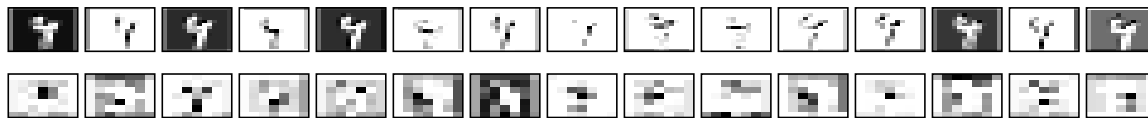
A terminológia itt főleg élettani eredetű. A „receptív mező” kifejezést eredetileg a retina és a látókéreg idegsejtjeinél használták. A konvolúciós hálók emlékeztetnek arra, ahogy a látórendszer működik; a retina egyfajta előfeldolgozást végez, helyi kontrasztokat érzékel. A látókéreg első rétege egyszerű alakzatokat – különböző orientációjú éleket – ismer fel, további rétegei pedig a korábbiakból építkezve egyre bonyolultabb alakzatokat detektál.

Végül vizsgáljuk meg, hogyan végezhetünk osztályozást vagy regressziót konvolúciós hálózattal. Az ötlet csupán annyi, hogy a konvolúciós rétegek a képek adaptív előfeldolgozását végzik, fontos jegyeket kiemelve; a hálózat utolsó rétegei pedig teljesen összekötöttek és működésükben nem különböznek a korábban megismert neurális hálózatoktól. Erre az architektúrára mutat példát a 4.10. algoritmus, amelyet kézzel írott számjegyek felismerésére használunk.

```
MLP(
  Chain(
    Conv((4,4), 1=>15, relu, stride=2, pad=1),
    Conv((4,4), 15=>15, relu, stride=2, pad=1),
    flatten,
    Dense(735, 10)
  )
)
```

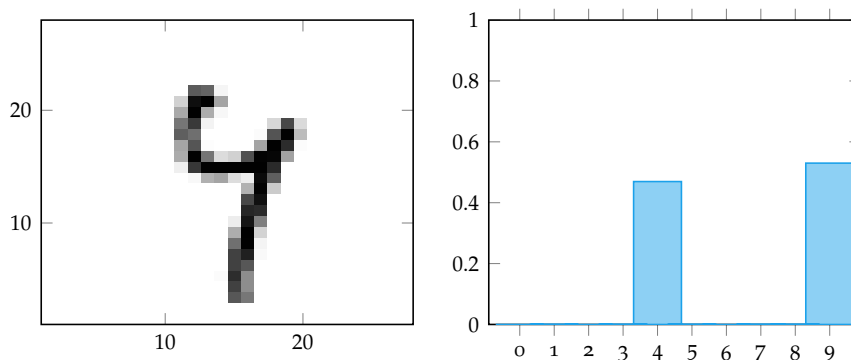
4.10. algoritmus. Képfelismerésre használható neurális hálózat  $4 \times 4$  konvolúciós kernelekkel és 15 csatornával két rétegben. A bemenetet  $28 \times 28$  méretű képek alkotják; a kimenet a lehetséges 0-9 számjegyeknek megfelelően 10 méretű.

A hálózatot a 4.9. algoritmussal tanítjuk. Érdekes megfigyelni a tanítás során kialakuló átmeneti reprezentációkat, azaz az egyes konvolúciós rétegek kimenetét, amelyek tehát a fontosnak ítélt jegyeket tartalmazzák:



4.8. ábra. Konvolúciós rétegek kimenete a 4.10. neurális hálózatban.

Az átmeneti reprezentációk értelmezése általában nehézkes. Tapasztalatok szerint a neurális hálózatok nem olyan reprezentációkkal dolgoznak, mint az emberek, így nemigen lehet megmondani, hogy mi alapján jutott a hálózat az adott következtetésre. A példában szereplő hálózat mindazonáltal kb. 99% pontosságot ér el a számjegyek felismerésében. A 4.9. ábra egy kevésbé egyértelmű esetet mutat:



4.9. ábra. A bal oldalon a bemenetet látjuk, a jobb oldalon pedig a softmax kimenetet, azaz a számjegy osztályának poszterior eloszlását.

#### 4.8 Automatikus differenciálás

Végül ejtsünk pár szót a gradiens-alapú tanítás gyakorlati megvalósításáról. A gradiens előállítását esetenként bonyolult számolást kíván meg (lásd 4.3. szakasz), ám maga az elv egyszerű: csupán a deriválási szabályokat kell teljesen mechanikusan alkalmazni. Az efféle „favágást” jobb a számítógépre bízni, ahogy a 4.9. algoritmusnál is láttuk. Az ötlet ráadásul nem korlátozódik a neurális hálózatok tanítására; bármilyen (differenciálható) numerikus program gradiense előállítható a szabályok ismételt alkalmazásával<sup>29</sup>. A 4.6.2. szakasz optimalizációs módszereit hozzávéve rögtön megkapjuk a következtető algoritmusokat anélkül, hogy akár egy sornyi kódot leírnánk (persze magát a modellt azért még nekünk kell összeállítani). A PyTorch, TensorFlow, Flux stb. könyvtárak ezt az eszköztárat adják a kezünkbe, kiegészítve a GPU, TPU-n történő futtatással.

Nézzünk erre egy példát. Vegyünk egy

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}$$

<sup>29</sup> Az automatikus differenciálás és differenciálható programozás ( $\partial P$ ) újabban nagy népszerűsége tett szert nem csak a gépi tanulásban, hanem fizikai rendszereknél, például a meteorológiában vagy a robotikában is.

függvényt, amely három lépésben számolható:

$$F(\mathbf{x}) = h(g(f(\mathbf{x}))),$$

ahol

$$\begin{aligned} f: \mathbb{R}^3 &\rightarrow \mathbb{R}^3, \\ g: \mathbb{R}^3 &\rightarrow \mathbb{R}^6, \\ h: \mathbb{R}^6 &\rightarrow \mathbb{R}, \end{aligned}$$

amelyek akár egy neurális hálózat rétegei is lehetnek<sup>30</sup>, vagy bármilyen más differenciálható leképezések. Általánosságban az  $F$  függvény  $l$ -edik komponensének<sup>31</sup> deriváltja az  $\mathbf{x}^i$ . komponens szerint a láncszabály felhasználásával írható:

$$\frac{\partial F^l}{\partial \mathbf{x}^i} = \sum_k \sum_j \frac{\partial h^l}{\partial g^k} \cdot \frac{\partial g^k}{\partial f^j} \cdot \frac{\partial f^j}{\partial \mathbf{x}^i}.$$

A jobb oldalon szereplő számításban felfedezhetjük a deriváltakat tartalmazó mátrixok (Jacobi-mátrixok) szorzatát. Emlékeztetőül az  $F$  függvény Jacobi-mátrixa

$$\nabla F = \left[ \frac{\partial F^l}{\partial \mathbf{x}^i} \right] = \begin{bmatrix} \frac{\partial F^1}{\partial \mathbf{x}^1} & \frac{\partial F^1}{\partial \mathbf{x}^2} & \cdots & \frac{\partial F^1}{\partial \mathbf{x}^I} \\ \frac{\partial F^2}{\partial \mathbf{x}^1} & \frac{\partial F^2}{\partial \mathbf{x}^2} & \cdots & \frac{\partial F^2}{\partial \mathbf{x}^I} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F^L}{\partial \mathbf{x}^1} & \frac{\partial F^L}{\partial \mathbf{x}^2} & \cdots & \frac{\partial F^L}{\partial \mathbf{x}^I} \end{bmatrix},$$

amely tehát a  $h$ ,  $g$  és  $f$  függvények Jacobi-mátrixának szorzataként áll elő:

$$\nabla F = \nabla h \cdot \nabla g \cdot \nabla f.$$

Ha tehát a fontosabb elemi függvények, „primitívek” – összeadás, kivonás, szorzás, exponenciális stb. – deriváltjait elő tudjuk állítani, ezekből a bonyultabb programok gradiense is kiszámolható, csupán az egyes lépésekben adódó Jacobi-mátrixokat kell összeszorozni. Az előbbi formulát képszerűen is ábrázolhatjuk:

$$\underbrace{\begin{bmatrix} * & * & * \end{bmatrix}}_{\nabla F} = \underbrace{\begin{bmatrix} * & * & * & * & * & * \end{bmatrix}}_{\nabla h} \cdot \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\nabla g} \cdot \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\nabla f}.$$

<sup>30</sup> Ebben az esetben az  $\mathbf{x}$  változó a súlyokat jelenti.

<sup>31</sup> Természetesen most csak egy komponensünk van, tehát  $l = 1$ .

Hogyan érdemes nekiállni  $\nabla F$  kiszámításának? Elindulhatunk például jobbról, a  $\nabla g$  és  $\nabla f$  mátrixokat összeszorozva marad még

$$\underbrace{\begin{bmatrix} * & * & * \end{bmatrix}}_{\nabla F} = \underbrace{\begin{bmatrix} * & * & * & * & * & * \end{bmatrix}}_{\nabla h} \cdot \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\nabla g \cdot \nabla f},$$

a végső gradienst így egy  $1 \times 6$  és egy  $6 \times 3$  mátrix szorzata adja. A teljes számolás során összesen 72 darab szorzást és 51 darab összeadást kell elvégeznünk. Másrészt viszont ha  $\nabla h$  és  $\nabla g$  szorzatával kezdünk, akkor

$$\underbrace{\begin{bmatrix} * & * & * \end{bmatrix}}_{\nabla F} = \underbrace{\begin{bmatrix} * & * & * \end{bmatrix}}_{\nabla h \cdot \nabla g} \cdot \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\nabla f},$$

azaz ugyanaz az eredmény egy  $1 \times 3$  és egy  $3 \times 3$  mátrix szorzataként áll elő. A teljes műveletigény 27 szorzás és 21 összeadás. Általánosságban sok bemenettel és kevés kimenettel rendelkező programok esetében – mint például a neurális hálózatok – célszerű az utóbbit választani.

A konkrét megvalósítások előtt még azt kell észrevennünk, hogy sok esetben pazarló lehet a teljes Jacobi-mátrix kiszámítása. Ha például

$$g(\mathbf{x}) = \begin{bmatrix} 2\mathbf{x}^1 + 3 \\ 2\mathbf{x}^2 - 1 \\ 2\mathbf{x}^3 \\ 3 \\ 4 \\ 5 \end{bmatrix},$$

akkor a Jacobi-mátrix sok zérust tartalmaz:

$$\nabla g = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Vegyük azt az esetet, amikor jobbról balra haladunk. Figyelembe véve, hogy ez a  $\nabla g$  mátrix nem önmagában kell, csupán jobbról meg szeretnénk szorozni valamivel, nem érdemes konkrétan mátrixként reprezentálni; jobb, ha közvetlenül a neki megfelelő leképezést valósítjuk meg. Például:

```
# Mátrixszal:
∇g_times(v) = [2 0 0
                0 2 0
                0 0 2
                0 0 0
                0 0 0
                0 0 0] * v

# JVP-vel:
∇g_times(v) = vcat(2v, zeros(3, size(v, 2)))
```

4.11. algoritmus. A  $\nabla g$  Jacobi-mátrix szorzása jobbról egy  $v$  vektorral vagy mátrixszal. A mátrixszorzás elvégzése helyett annak eredményét egyszerűbben is kiszámolhatjuk; esetünkben ez egy lineáris operátor, amely egy vektort kétszeresére nyújt, majd hozzácsap három 0-t (mátrix-értékű  $v$  esetén ugyanezt alkalmazzuk  $v$  összes oszlopára).

A szakirodalom a műveletet *jacobian vector product* (JVP) névvel illeti. A jobbról balra haladó kiértékelés nem más, mint JVP-k kompozíciója; a primitív JVP-k lényegében a deriválási szabályokkal egyeznek meg, amelyek könnyen implementálhatók. Ugyanígy járunk el, ha balról jobbra haladva értékelünk ki. Ebben az esetben a Jacobi-mátrixok mindig balról szorzódnak; az ennek megfelelő művelet a *vector jacobian product* (VJP).

Legyen most

$$f(\mathbf{x}) = \begin{bmatrix} \mathbf{x}^1 + \mathbf{x}^2 \\ \mathbf{x}^1 \cdot \mathbf{x}^3 \\ 3\mathbf{x}^2 \end{bmatrix}, \quad \nabla f = \begin{bmatrix} 1 & 1 & 0 \\ \mathbf{x}^3 & 0 & \mathbf{x}^1 \\ 0 & 3 & 0 \end{bmatrix},$$



és tegyük fel, hogy birtokunkban van a  $\nabla f$ -fel való szorzás JVP formájában. Ez lehetővé teszi, hogy  $f$  összes komponensének  $\mathbf{x}^1$  szerinti deriváltját kiszámítsuk.

$$\frac{\partial f}{\partial \mathbf{x}^1} = \begin{bmatrix} 1 & 1 & 0 \\ \mathbf{x}^3 & 0 & \mathbf{x}^1 \\ 0 & 3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = JVP_f \left( \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ \mathbf{x}^3 \\ 0 \end{bmatrix}.$$

Hasonlóképpen járhatunk el az  $\mathbf{x}^2$  szerinti deriváltakra<sup>32</sup>:

$$\frac{\partial f}{\partial \mathbf{x}^2} = \begin{bmatrix} 1 & 1 & 0 \\ \mathbf{x}^3 & 0 & \mathbf{x}^1 \\ 0 & 3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = JVP_f \left( \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}.$$

A JVP-k tehát arra jók, hogy az összes kimenet deriváltját megkapjuk egy-egy bemenetre nézve (magyarán a Jacobi-mátrix egy-egy oszlopát).

VJP-kkel dolgozva fordított a helyzet: ekkor egy-egy kimenet deriváltját kaphatjuk meg az összes bemenetre nézve, azaz a Jacobi-mátrix egy-egy sorát<sup>33</sup>:

$$\begin{aligned} \nabla_{\mathbf{x}} f^1 &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ \mathbf{x}^3 & 0 & \mathbf{x}^1 \\ 0 & 3 & 0 \end{bmatrix} = VJP_f \left( \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \right) = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \\ \nabla_{\mathbf{x}} f^2 &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ \mathbf{x}^3 & 0 & \mathbf{x}^1 \\ 0 & 3 & 0 \end{bmatrix} = VJP_f \left( \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{x}^3 & 0 & \mathbf{x}^1 \end{bmatrix}. \end{aligned}$$

A jobbról balra haladó stratégiát *előre-módú* (*forward-mode*) automatikus differenciálásnak nevezik, mivel a deriváltakat a számítással párhuzamosan számoljuk: ahogy az  $f, g, h$  műveletek egymást követik, ugyanúgy rögtön a  $\nabla f \cdot \mathbf{v}$ ,  $\nabla g \cdot \nabla f \cdot \mathbf{v}$ ,  $\nabla h \cdot \nabla g \cdot \nabla f \cdot \mathbf{v}$  szorzatokat is kiszámoljuk. Ennek előnye, hogy a gradiensek számolásához szükséges információt nem kell a memóriában tárolni. Formálisan egy  $f$  művelet a következő formát ölti:

$$\mathbf{x} \mapsto \left( f(\mathbf{x}), \Delta \mapsto \nabla f \Big|_{\mathbf{x}} \cdot \Delta \right),$$

ahol az első elem a függvény kimenete, a másik pedig a JVP-t megvalósító eljárás, amit azon nyomban ki is értékelünk.

<sup>32</sup> Az  $\mathbf{x}^3$  szerinti esetet ellenőrizni házi feladat.

<sup>33</sup> Látjuk, hogy egyetlen kimenet és sok bemenet esetén ez a legpraktikusabb, hiszen egy csapásra az összes deriváltat megkapjuk. Neurális hálózatoknál az egyetlen kimenet lehet a veszteség, a VJP-k kompozíciójával pedig egy menetben a súlyokra vonatkozó összes derivált (gradiens) a birtokunkba kerül. Ez nem más, mint a hiba-visszaterjesztés algoritmus álruhában.

A balról jobbra haladó stratégiát *viSSzafele-módú* (*reverse mode*) automatikus differenciálásnak nevezik, mivel a deriváltakat fordított sorrendben számoljuk. Formálisan az  $f$  művelet a következőképpen néz ki:

$$\mathbf{x} \mapsto \left( f(\mathbf{x}), \tilde{\Delta} \mapsto \tilde{\Delta}^\top \cdot \nabla f \Big|_{\mathbf{x}} \right),$$

ahol a második elem a VJP-t megvalósító eljárás. Ennek kiértékeléséhez először végig kell futtatni a teljes számítást (*forward pass*, tehát  $f$ ,  $g$ , majd  $h$ ), „megjegyezni” az deriváltak számolásához szükséges információt (VJP-ket), majd a végén fordított sorrendben komponálni a VJP-ket ( $\tilde{\mathbf{v}}^\top \cdot \nabla h$ ,  $\tilde{\mathbf{v}}^\top \cdot \nabla h \cdot \nabla g$ , majd  $\tilde{\mathbf{v}}^\top \cdot \nabla h \cdot \nabla g \cdot \nabla f$ ).

Végül pár szó az implementációról. Látjuk, hogy a műveleteket ki kell egészíteni a deriváltak számításához szükséges információval; az automatikus differenciálást támogató programcsomagok erre meglehetősen eltérő filozófiát követnek. Az *operator overloading* technikán alapuló könyvtárak (pl. PyTorch, JAX, TensorFlow) speciális wrapper adattípusokat használnak, amelyekre nézve a primitív operátorok túlterheltek, és rögtön a JVP/VJP-ket is kiszámolják. A *source code transformation* eszközök (pl. Zygote) a forráskód transzformációjával (saját compiler-rel) állítják elő a gradienseket számító kódot. Újabb megközelítés az Enzyme könyvtár, amely az LLVM alacsony szintű reprezentációját használja, így sokkal inkább nyelvfüggetlen, és képes a már optimalizált kódhoz is deriváltakat előállítani.

## 5. fejezet

# Variációs közelítés

Ebben a fejezetben áttérünk a generatív modellezésre. A korábbiaknál jóval nehezebb feladatra vállalkozunk: míg eddig egy  $p(y | \mathbf{x})$  mennyiségre voltunk kíváncsiak (azaz címkéket jósoltunk az adatpontokhoz), most az együttes  $p(\mathbf{x}, y)$  eloszlást keressük<sup>1</sup>. Az együttes eloszlás birtokában akár új adatokat is tudunk generálni.

<sup>1</sup> Ebben a fejezetben általánosabban leszünk, az esetleges címkéket is beleértjük az adatba, és csupán  $p(\mathbf{x})$ -et írunk.

### 5.1 Evidence lower bound (ELBO)

Egy valószínűségi modellben szereplő változókat két csoportra oszthatjuk: megfigyelt és rejtett változókra. A megfigyelt változók alatt rendszerint olyasmit értünk, ami a következtető algoritmusunk bemeneteként szolgálhat; jelölje mind-ezen adatok összességét  $\mathbf{x}$ . Bayesi következtetésnél a célunk a  $\mathbf{z}$ -vel jelölt rejtett változók eloszlását megbecsülni; ez tehát azon változók összessége, amelyekre következtetni szeretnénk<sup>2</sup>. Ezen eloszlások birtokában készen is vagyunk a generatív modellel; új adatok generálásához csupán mintavételezni kell, ami általában könnyen kivitelezhető<sup>3</sup> (lásd a 7. fejezetet).

A feladat tehát a rejtett változók eloszlásának megbecsülése az adatok ismeretében; formálisan a  $p(\mathbf{z} | \mathbf{x})$  mennyiségre vagyunk kíváncsiak. Szokásunkhoz híven próbálkozzunk meg a Bayes-tétellel:

$$p(\mathbf{z} | \mathbf{x}) = \frac{p(\mathbf{x} | \mathbf{z}) p(\mathbf{z})}{p(\mathbf{x})} = \frac{p(\mathbf{x} | \mathbf{z}) p(\mathbf{z})}{\int p(\mathbf{x} | \mathbf{z}) p(\mathbf{z}) d\mathbf{z}},$$

<sup>2</sup> Lineáris regressziónál például az ismeretlen súlyvektor játssza a rejtett változók szerepét, de lehetnek sokkal bonyolultabb, hierarchikus modelljeink is sokféle rejtett változóval.

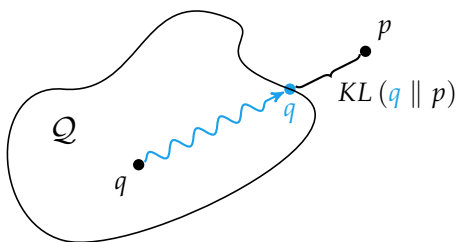
<sup>3</sup> Ebben az értelemben tehát minden teljesen bayesi modell generatív.

azonban itt kellemetlen meglepetéssel szembesülünk: a nevezőt általában nem tudjuk kiszámolni<sup>4</sup>! A korábban használt másik trükkünk, miszerint nagyvonalúan eltekintünk a nevezőtől – hiszen  $\mathbf{z}$ -ben konstans – és csak a számlálót maximalizáljuk  $\mathbf{z}$  szerint, megint csak nem működik, mivel most nem egy optimális értéket szeretnénk kapni  $\mathbf{z}$ -re, hanem egy eloszlást.

A megoldáshoz vezessünk egy  $q(\mathbf{z})$  közelítő (ún. *variációs*) eloszlást; ezt fogjuk addig-addig javítgatni, amíg „közel nem kerülünk” a valódi  $p(\mathbf{z} | \mathbf{x})$  eloszláshoz. A kettő közötti eltérést a *Kullback–Leibler divergenciával*<sup>5</sup> mérjük:

$$KL(q(\mathbf{z}) \parallel p(\mathbf{z} | \mathbf{x})) := - \int q(\mathbf{z}) \ln \frac{p(\mathbf{z} | \mathbf{x})}{q(\mathbf{z})} d\mathbf{z}.$$

A  $q(\mathbf{z})$  eloszlást mi mondhatjuk meg. Ha egy rögzített  $\mathcal{Q}$  függvénycsaládból választjuk, a feladatunk ilyesféleképpen alakul:



Ez egyben azt is jelenti, hogy a variációs módszerekkel általában nem a valódi eloszlást találjuk meg<sup>6</sup>. A függvénycsalád megválasztása éppen ezért kulcsfontosságú. Általában olyasmire törekszünk, ami kellően flexibilis, tehát közel kerülhetünk a valódi eloszláshoz, másrésztől viszont könnyen kezelhető<sup>7</sup>. A következő felbontás képezi a variációs módszerek alapját:

$$\begin{aligned} \underbrace{\int q(\mathbf{z}) \ln \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} d\mathbf{z}}_{:= \mathcal{L}[q(\mathbf{z})]} - \underbrace{\int q(\mathbf{z}) \ln \frac{p(\mathbf{z} | \mathbf{x})}{q(\mathbf{z})} d\mathbf{z}}_{KL(q(\mathbf{z}) \parallel p(\mathbf{z} | \mathbf{x}))} &= \int q(\mathbf{z}) \ln \frac{p(\mathbf{x}, \mathbf{z})q(\mathbf{z})}{p(\mathbf{z} | \mathbf{x})q(\mathbf{z})} d\mathbf{z} \\ &= \int q(\mathbf{z}) \ln p(\mathbf{x}) d\mathbf{z} \\ &= \ln p(\mathbf{x}) \int q(\mathbf{z}) d\mathbf{z} \\ &= \ln p(\mathbf{x}), \end{aligned}$$

<sup>4</sup> Az integrálra általában konjugált modelleknél van zárt formulánk; egy példa erre a bayesi lineáris regresszió, ahol az integrálban szereplő mindkét eloszlás normális.

<sup>5</sup> Ez bizonyos értelemben a kanonikus választás, de léteznek másféle divergenciák is. Vegyük észre, hogy a feladat ezt a mennyiséget minimalizálni a  $q$  függvény (!) szerint. Az efféle problémák fizikában nagyon gyakoriak, *variációs számítás*-sal oldhatók meg, innen a módszer család neve.

5.1. ábra. Következtetés a variációs elvvel. A  $q \in \mathcal{Q}$  eloszlást úgy módosítjuk, hogy a lehető „legközelebb” kerüljön a valódi  $p$  eloszláshoz.

<sup>6</sup> Ellentétben a 7. fejezet módszerével, amelyek egzaktnak.

<sup>7</sup> Ennek érdekében  $q(\mathbf{z})$ -t gyakran neurális hálózatokkal paraméterezzük fel. A későbbi szakaszokban látni fogunk egy egyszerű és egy valamivel bonyolultabb példát.

ahol az  $\mathcal{L}[q(\mathbf{z})]$  mennyiség neve *evidence lower bound (ELBO)*<sup>8</sup>, a KL-divergenciát pedig már ismerjük. A lényegét kiírva kaptunk egy

$$\ln p(\mathbf{x}) = \mathcal{L}[q(\mathbf{z})] + KL(q(\mathbf{z}) \parallel p(\mathbf{z} | \mathbf{x})) \quad (5.1)$$

alakú felbontást. Az egyenlet jobb oldalán szereplő két mennyiség közül a KL-divergenciát szeretnénk  $q(\mathbf{z})$  szerint minimalizálni. A variációs módszerek trükkösen járnak el: a közvetlen minimalizálás helyett inkább  $\mathcal{L}$ -t maximalizálják  $q(\mathbf{z})$  szerint. Mivel egyenlet bal oldala  $q(\mathbf{z})$ -ben konstans, a KL-divergenciának kötelessége lesz csökkenni!

Nézzük, hogyan valósulhat meg  $\mathcal{L}$  maximalizálása a gyakorlatban<sup>9</sup>. Vegyünk negatív előjelet (tehát mostantól minimalizálunk), valamint használjuk fel, hogy  $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} | \mathbf{z}) p(\mathbf{z})$  és szedjük kétfelé az ELBO-t:

$$\begin{aligned} -\mathcal{L}[q(\mathbf{z})] &= -\int q(\mathbf{z}) \ln \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} d\mathbf{z} \\ &= -\int q(\mathbf{z}) \ln p(\mathbf{x} | \mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \ln \frac{p(\mathbf{z})}{q(\mathbf{z})} d\mathbf{z} \\ &= \underbrace{\mathbb{E}_{q(\mathbf{z})}[-\ln p(\mathbf{x} | \mathbf{z})]}_{\text{várható veszteség}} + \underbrace{KL(q(\mathbf{z}) \parallel p(\mathbf{z}))}_{\text{„regularizáció”}}. \end{aligned} \quad (5.2)$$

Az első tagban felfedezhetünk egy negatív log-likelihood-jellegű mennyiséget<sup>10</sup>, a második tag pedig arról gondoskodik, hogy a  $q(\mathbf{z})$  eloszlás ne távolodjon el túlságosan a  $p(\mathbf{z})$  priortól, azaz a korábbi fejezetekben látott regularizációs tagokhoz hasonlóan viselkedik<sup>11</sup>.

Nézzük meg, hogy a várható veszteséggel mit tudunk kezdeni. A kiszámolásához több verzió lehetséges:

1. A várható érték zárt formában számolható. Ez ritka; éppen azért használjuk a variációs közelítést, mert nem bírunk ezekkel az integrálokkal.
2. Közelítjük az integrált. A Laplace-approximáció<sup>12</sup> például osztályozási problémáknál jól működik; jónéhány más esetben kevésbé.
3. Kvadraturát használunk. Ha a variációs eloszlás normális, a Gauss–Hermite kvadratura<sup>13</sup> nagyon gyorsan és pontosan tudja közelíteni a várható értéket.

<sup>8</sup> Mivel  $KL(\cdot \parallel \cdot) \geq 0$ , rögtön látjuk, hogy  $\ln p(\mathbf{x}) \geq \mathcal{L}[q(\mathbf{z})]$ , azaz jogos az „alsó korlát” kifejezés. Az „evidencia” kifejezés pedig arra utal, hogy az ELBO-t mindig egy megfigyelt adathalmaz alapján számoljuk.

<sup>9</sup> A most mutatott módszer csupán a sok közül, vannak más stratégiák is.

<sup>10</sup> Aha, felbukkant egy veszteségfüggvény! Kár, hogy el van dugva egy várható érték belsejébe.

<sup>11</sup> Ez a megközelítés akkor működik, amikor a KL-divergenciát zárt formában tudjuk számolni, például ha mind a priort, mind a variációs eloszlást normális eloszlásúnak tesszük fel. Ha a KL-divergencia nem számolható zárt alakban, akkor az 5.4. szakasz nyújt segítséget.

<sup>12</sup> Lásd a C. függelékét.

<sup>13</sup> Lásd a C. függelékét.

4. Ha semmi más nem működik, de a variációs eloszlásból legalább mintavételezni tudunk, a várható értéket közelíthetjük  $S$  darab  $\mathbf{z}_s \sim q(\mathbf{z})$  minta átlagával is, azaz

$$\mathbb{E}_{q(\mathbf{z})} [-\ln p(\mathbf{x} | \mathbf{z})] \approx \frac{1}{S} \sum_{s=1}^S -\ln p(\mathbf{x} | \mathbf{z}_s).$$

A gyakorlatban sokszor egyetlen minta is elég, de előfordul, hogy gradiens-alapú optimalizációnál túlságosan sztochasztikus gradienseket kapunk, ami jelentősen lassítja a tanulást.

A következő szakaszokban megvizsgáljuk  $q(\mathbf{z})$  megválasztását különféle modellek esetében, valamint bevetjük a 4. fejezet módszereit a hatékony tanuláshoz.

## 5.2 Bayesi logisztikus regresszió

A 3. fejezet logisztikus regressziós modelljeinél csak ML vagy MAP megoldásokat tudtunk keresni. Variációs módszerrel azonban teljesen bayesi megoldást is kaphatunk, ha az előző szakaszban  $\mathbf{z}$  szerepét a rejtett  $\mathbf{w}$  súlyokra,  $\mathbf{x}$  szerepét pedig az  $(\mathbf{X}, \mathbf{y})$  megfigyelt be- és kimenetekre osztjuk. Magukat az adatpontokat most békén hagyjuk: egyelőre nem cél a teljes adat eloszlásának modellezése, csupán a logisztikus regressziót szeretnénk „bayesiesíteni”<sup>14</sup>.

A logisztikus regresszió valószínűségi modellje a (3.3) egyenlet alapján

$$p(\mathbf{y} | \mathbf{X}, \mathbf{w}) = \prod_i \text{Bern}(y_i | \sigma(\mathbf{w}^\top \phi(\mathbf{x}_i))),$$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \mathbf{I}).$$

A  $p(\mathbf{w} | \mathbf{X}, \mathbf{y})$  poszteriorra vagyunk kíváncsiak. Vezessük be a közelítő variációs eloszlást a következőképpen:

$$q(\mathbf{w}) := \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)),$$

azaz az eloszlást normálisnak választottuk az egyszerű számolás érdekében. A  $\boldsymbol{\mu}$  és  $\boldsymbol{\sigma}^2$  paraméterek a *variációs paraméterek*, amelyeket gradiens-alapon szeretnénk optimalizálni. Az optimalizációnál a negatív ELBO játssza a célfüggvény szerepét, ami a következőképpen alakul<sup>15</sup>:

<sup>14</sup> A teljes generatív modellezésre a következő szakaszban látunk majd példát.

<sup>15</sup> Vesd össze az (5.2) egyenlettel.

$$\begin{aligned}
-\mathcal{L}[q(\mathbf{w})] &= \mathbb{E}_{q(\mathbf{w})} [-\ln p(\mathbf{X}, \mathbf{y} | \mathbf{w})] + KL(q(\mathbf{w}) \parallel p(\mathbf{w})) \\
&= \mathbb{E}_{q(\mathbf{w})} [-\ln p(\mathbf{y} | \mathbf{X}, \mathbf{w})] + KL(q(\mathbf{w}) \parallel p(\mathbf{w})) + \text{const.} \\
&= \mathbb{E}_{q(\mathbf{w})} \left[ \sum_i BCE(\mathbf{y}_i, \mathbf{w}^\top \phi(\mathbf{x}_i)) \right] + KL(q(\mathbf{w}) \parallel p(\mathbf{w})) + \text{const.}
\end{aligned}$$

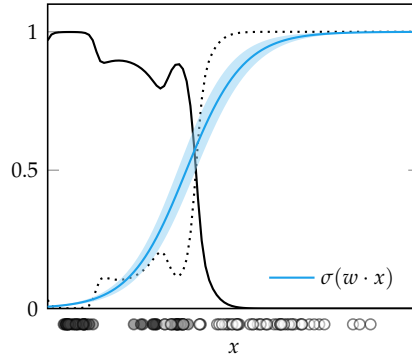
ahol kihasználtuk, hogy  $\mathbf{w}$  független  $\mathbf{X}$ -től, tehát

$$p(\mathbf{X}, \mathbf{y} | \mathbf{w}) = p(\mathbf{y} | \mathbf{X}, \mathbf{w}) p(\mathbf{X} | \mathbf{w}) = p(\mathbf{y} | \mathbf{X}, \mathbf{w}) p(\mathbf{X}) = p(\mathbf{y} | \mathbf{X}, \mathbf{w}) \cdot \text{const.}$$

A várható veszteségben a korábbi logisztikus regresszióval teljesen analóg módon bináris keresztentropia kerül elő, a két normális eloszlás közötti KL-divergenciát pedig zárt alakban számolhatjuk<sup>16</sup>:

$$KL(q(\mathbf{w}) \parallel p(\mathbf{w})) = \frac{1}{2} \sum_n \left( (\mu^n)^2 + (\sigma^2)^n - \ln(\sigma^2)^n - 1 \right).$$

Most már minden adott gradiens-módszer alkalmazásához; ki tudjuk számolni a várható veszteséget (például a mintavételezés módszerrel) és a KL-divergenciát is. Az eredményeket egy dimenzióban az 5.2. ábra, két dimenzióban az 5.3 szemlélteti.



5.2. ábra. Bayesi logisztikus regresszió egy dimenzióban. Az  $x$  tengelyen a minták láthatók, a kék függvény a  $q(w)$  variációs eloszlás alapján számolt szigmoid, ahol most már a bizonytalanságot is látjuk.

Az 5.1. algoritmus egy egyszerű implementációt mutat be.

```

struct BayesianLogisticRegression
    μ      # q(w) várható érték
    logσ²  # q(w) log variancia

    BayesianLogisticRegression(dim) =
        new(zeros(dim), zeros(dim))
end
function train!(m::BayesianLogisticRegression, x, y; η=0.01, iters=100)
    ps = Params([m.μ, m.logσ²])
    for i in 1:iters
        grad = gradient(ps) do
            σ² = exp.(m.logσ²)
            kl = 0.5*sum(@. m.μ^2 + σ² - m.logσ² - 1)
            w = m.μ .+ randn(size(σ²)) .* σ²
            bce = sum(BCE.(x*w, y))
            elbo = -bce + kl
        end

        m.μ .+= η*grad[m.μ]
        m.logσ² .+= η*grad[m.logσ²]
    end
end

```

5.1. algoritmus. Variációs bayesi logisztikus regresszió. A variációs eloszlás várható értékét és varianciájának logaritmusát tanuljuk (így a variancia nem csúszhat zérus alá az optimalizáció során). A tanuláshoz a legegyszerűbb gradiens-módszert használjuk.

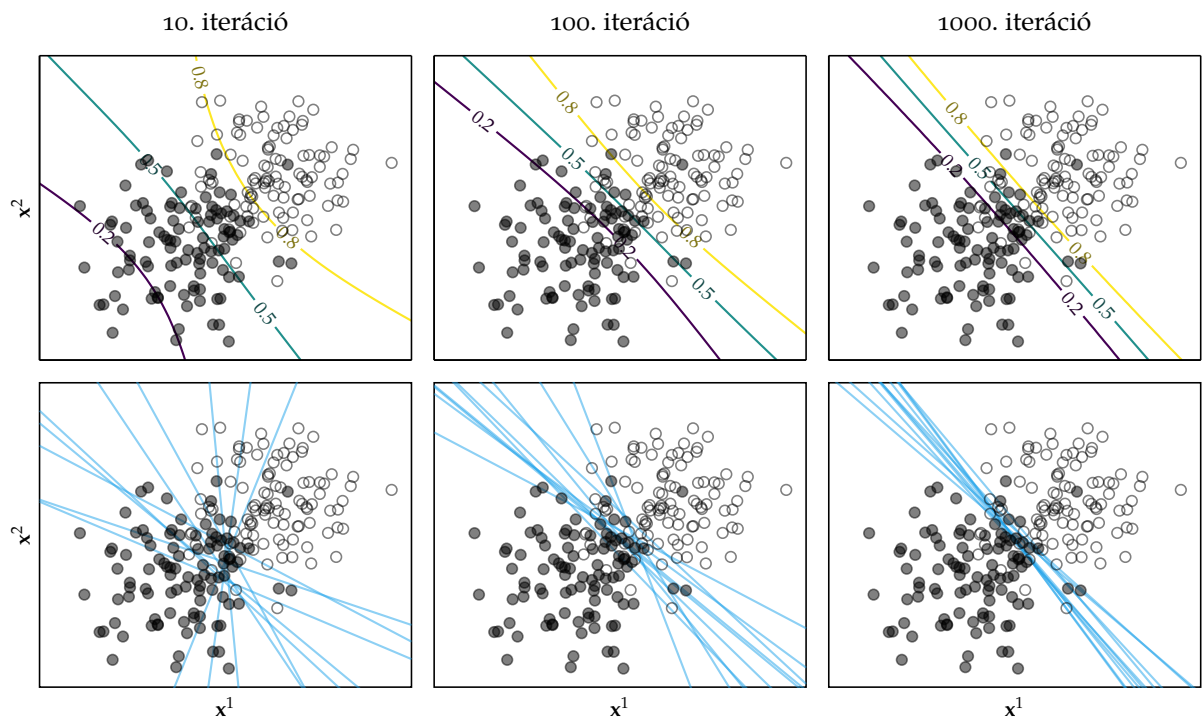
### 5.3 Variációs autoenkóder

Variációs autoenkóderrel a teljes adat eloszlását –  $p(\mathbf{x})$ -et – modellezhetjük. Ehhez speciális valószínűségi modellt teszünk fel, azaz  $p(\mathbf{x} | \mathbf{z})$  alakját megköjtjük. Szemléletesen mondva megszabjuk, hogy egy bizonyos, alacsony dimenziójú  $\mathbf{z}$  rejtett változó ismeretében hogyan lehet mintákat generálni. Ez durva megkötésnek tűnhet, hiszen így jelentős megkötéseket teszünk  $p(\mathbf{x})$ -re is; éppen ezért a generatív folyamatot „adaptívan”, neurális hálózatokkal valósítjuk meg. Sőt, ha már itt tartunk, hasonlóan járunk el a  $q(\mathbf{z})$  variációs eloszlással is, tehát mindkét eloszlást neurális hálózatokkal paraméterezzük, amelyeknek  $\psi$  és  $\theta$  súlyrendszerei tanulhatók. Formálisan<sup>17</sup>

$$\begin{aligned}
 p(\mathbf{x} | \mathbf{z}) &\rightsquigarrow p_{\psi}(\mathbf{x} | \mathbf{z}) := \mathcal{N}(\mathbf{x} | g_{\psi}(\mathbf{z}), \mathbf{I}), \\
 q(\mathbf{z}) &\rightsquigarrow q_{\theta}(\mathbf{z} | \mathbf{x}) := \mathcal{N}(\mathbf{z} | \mu_{\theta}(\mathbf{x}), \sigma_{\theta}^2(\mathbf{x})).
 \end{aligned}$$

<sup>17</sup> Természetesen nem kötelező a normális eloszlásokhoz ragaszkodni, de ez a leggyakoribb választás.



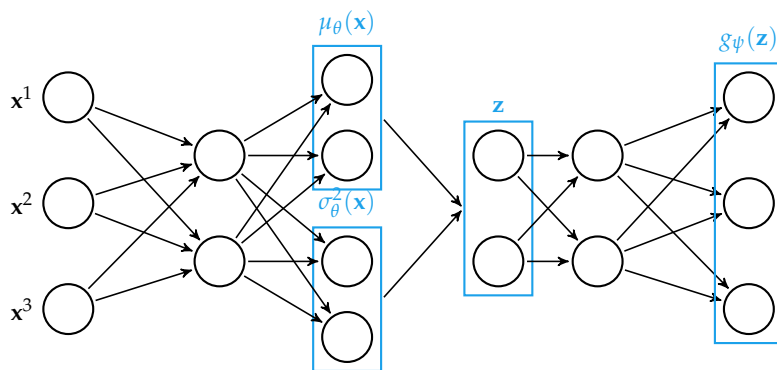


5.3. ábra. Bayesi lineáris regresszió két dimenzióban. A felső sor a posztterior várható értéket, az alsó sor pedig a posztteriorból származó mintákat mutatja a tanítás során.

A variációs eloszlás paramétereit tanuló hálózatok – nevezetesen  $\mu_\theta$  és  $\sigma_\theta^2$  – bemenetét  $\mathbf{x}$  szolgáltatja<sup>18</sup>. Ettől az architektúra tényleg egyfajta autoenkóder struktúrát vesz fel, hiszen  $\mathbf{x}$  egyfajta bemenetként és célváltozóként is megjelenik: bemenetként  $q_\theta(\mathbf{z} | \mathbf{x})$ -ben, célváltozóként  $p_\psi(\mathbf{x} | \mathbf{z})$ -ben. Ezekkel a választásokkal

$$-\mathcal{L}[q_\theta(\mathbf{z} | \mathbf{x})] = \underbrace{\mathbb{E}_{q_\theta(\mathbf{z} | \mathbf{x})} \left[ \frac{1}{2} \sum_n (\mathbf{x}^n - g_\psi(\mathbf{z})^n)^2 \right]}_{\text{„rekonstrukciós loss”}} + \underbrace{KL(q_\theta(\mathbf{z} | \mathbf{x}) \| p(\mathbf{z}))}_{\text{„regularizáció”}},$$

ahol  $p(\mathbf{z})$ -t rendszerint standard normális eloszlásnak választjuk. Diagramon:



5.4. ábra. Variációs autoenkóder sematikus rajza. Tényleg egyfajta autoenkódert látunk: a bemeneti  $\mathbf{x}$  mintát próbáljuk a hálózat másik végén rekonstruálni.

A célfüggvény első tagja az  $\mathbf{x}$  minta és a  $g_\psi(\mathbf{z})$  rekonstrukció közötti várható veszteséget méri, ahol a veszteségfüggvény  $-\ln p_\psi(\mathbf{x} | \mathbf{z})$ . A második tag arról gondoskodik, hogy a  $q(\mathbf{z} | \mathbf{x})$  variációs eloszlás ne távolodjon el a  $p(\mathbf{z})$  priortól. A variációs autoenkóderre tehát úgy is gondolhatunk, hogy egy enkóderből és egy dekóderből áll; előbbi a rejtett  $q(\mathbf{z} | \mathbf{x})$  eloszlás paramétereit tanulja, míg utóbbi a  $q(\mathbf{z})$ -ből vett minta alapján rekonstruálja az eredeti bemenetet. Úgy is mondhatjuk, hogy  $\mathbf{x}$ -et átküldjük egy „információs útszűkületen”; mivel  $\mathbf{z}$  rendszerint sokkal alacsonyabb dimenzionalitású, mint  $\mathbf{x}$ , a hálózat kénytelen robusztus módon a lehető legtöbb információt eltárolni  $\mathbf{x}$ -ből az optimális rekonstrukcióhoz.

<sup>18</sup> Ezt minden további nélkül megtehetjük; lényegében csak egy speciális alakot választottunk  $q(\mathbf{z})$ -nek. Más generatív architektúrák (pl. GAN) nem élnek ezzel a lehetőséggel; valójában el is tekinthetnénk tőle és tanulhatnánk  $q(\mathbf{z})$  paramétereit közvetlenül, nem pedig egy  $\mathbf{x}$  bemenetre kötött neurális hálózattal.

Nézzük a variációs autoenkóder „alkatrészeit”:

```
struct VAE
  enc      # enkóder
  q_μ      # q(z)
  q_logσ²  # q(z)
  dec      # dekóder

  VAE(enc, q_μ, q_logσ², dec) =
    new(enc, q_μ, q_logσ², dec)
end
```

5.2. algoritmus. Variációs autoenkóder megvalósítása. Az előző szakaszhoz hasonlóan a variancia logaritmusát tanuljuk.

Az enkódert és a dekódert neurális hálózatokként valósítjuk meg.

```
VAE(
  Chain(                                     # enkóder
    flatten,
    Dense(28*28, 500, relu),
    Dense(500, 500)),
  Dense(500, 2),                             # q(z)
  Dense(500, 2),                             # q(z)
  Chain(                                     # dekóder
    Dense(2, 500, relu),
    Dense(500, 500, relu),
    Dense(500, 28*28, tanh),
    x->reshape(x, 28, 28, 1, :))
)
```

5.3. algoritmus. Variációs autoenkóder létrehozása. A rejtett változó dimenzionalitását 2-re állítottuk.

A tanulás menetét az 5.4. algoritmus mutatja.



5.5. ábra. Variációs autoenkóder tanítása után rekonstruált minták az 5.3. architektúrával.

```

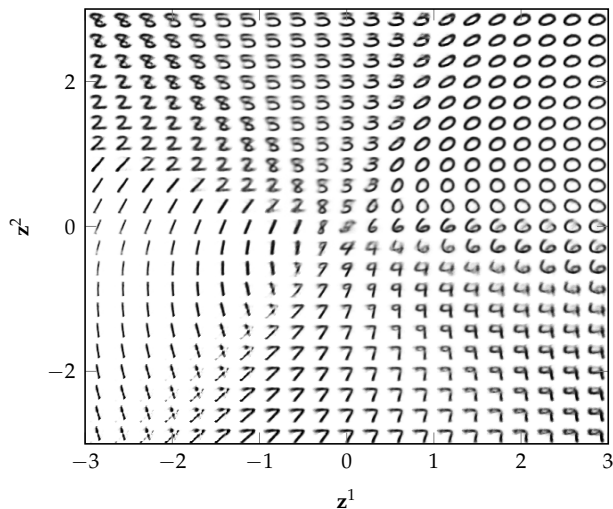
function train!(vae::VAE, x; epochs=10, batchsize=512)
    ps = params(vae.enc, vae.q_μ, vae.q_logσ², vae.dec)
    opt = ADAM()
    data = DataLoader(x, batchsize=batchsize, shuffle=true)

    for epoch in 1:epochs, batch in data
        grad = gradient(ps) do
            a = vae.enc(batch)
            μ, logσ² = vae.q_μ(a), vae.q_logσ²(a)
            σ² = exp.(logσ²)
            kl = 0.5*sum(@. μ² + σ² - 1 - logσ²)
            z = μ .+ randn(size(σ²)) .* σ²
            sq = -sum((batch .- vae.dec(z)).^2)
            elbo = -sq + kl
        end
        update!(opt, ps, grad)
    end
end

```

5.4. algoritmus. Variációs autoenkóder tanítása a Flux könyvtár felhasználásával.

Mintageneráláshoz mindössze a dekódert kell használnunk; ez éppen arra lett „kiképezve”, hogy egy ismert eloszlásból<sup>19</sup> származó mintákat  $x$ -ekké alakítson. Érdekes megfigyelni, mit látunk, ha a dekódert különböző  $z$ -ekkel hajtjuk meg:



<sup>19</sup> Ez úgy értendő, hogy a variációs eloszlásból, de mivel az remélhetőleg közel van a  $p(z)$  priorhoz, utóbbit használhatjuk a generáláshoz.

5.6. ábra. A  $z$  rejtett változó hatása a generált mintákra. Látjuk, hogy a rejtett változók kétdimenziós térben más-más számjegyek máshonnan szeretnek generálódni.

### 5.4 Versengő modellek

Mind a bayesi lineáris regressziónál, mind a variációs autoenkódernél feltettük, hogy az ELBO-ban a KL-divergenciát zárt formában tudjuk számolni. Ez csak akkor igaz, ha „egyszerű” variációs eloszlást és priort választunk. Tekintsünk most el ettől. A negatív ELBO továbbra is<sup>20</sup>:

<sup>20</sup> Lásd az (5.2) egyenletet.

$$\begin{aligned} -\mathcal{L}[q_\theta(\mathbf{z} | \mathbf{x})] &= \mathbb{E}_{q_\theta(\mathbf{z} | \mathbf{x})} [-\ln p_\psi(\mathbf{x} | \mathbf{z})] + KL(q_\theta(\mathbf{z} | \mathbf{x}) \parallel p(\mathbf{z})) \\ &= \mathbb{E}_{q_\theta(\mathbf{z} | \mathbf{x})} [-\ln p_\psi(\mathbf{x} | \mathbf{z})] - \mathbb{E}_{q_\theta(\mathbf{z} | \mathbf{x})} \left[ \ln \frac{p(\mathbf{z})}{q_\theta(\mathbf{z} | \mathbf{x})} \right]. \end{aligned}$$

Figyeljük meg, hogy a második várható értékben szereplő hányados értéke egy adott  $\mathbf{z}$  esetén

$$\ln \frac{p(\mathbf{z})}{q_\theta(\mathbf{z} | \mathbf{x})} = \begin{cases} \ll 0, & \text{ha } \mathbf{z} \sim q_\theta(\mathbf{z} | \mathbf{x}), \\ \gg 0, & \text{ha } \mathbf{z} \sim p(\mathbf{z}). \end{cases}$$

Alkalmazzuk erre a szigmoid függvényünket:

$$\sigma \left( \ln \frac{p(\mathbf{z})}{q_\theta(\mathbf{z} | \mathbf{x})} \right) = \begin{cases} \sim 0, & \text{ha } \mathbf{z} \sim q_\theta(\mathbf{z} | \mathbf{x}), \\ \sim 1, & \text{ha } \mathbf{z} \sim p(\mathbf{z}). \end{cases}$$

Ez egészen olyan, mintha valamiféle osztályozási feladattal állnánk szemben: egy-egy  $\mathbf{z}$ -ről szeretnénk eldönteni, hogy a  $p(\mathbf{z})$  priorból, vagy a  $q_\theta(\mathbf{z} | \mathbf{x})$  variációs eloszlásból származik. Erre az osztályozási feladatra vezessünk be egy  $D(\mathbf{x}, \mathbf{z})$  neurális hálózatot, amelyet *diszkriminátornak* fogunk nevezni. Az erre vonatkozó veszteségfüggvény legyen

$$\mathbb{E}_{p(\mathbf{z})} [\ln \sigma(D(\mathbf{x}, \mathbf{z}))] + \mathbb{E}_{q_\theta(\mathbf{z} | \mathbf{x})} [\ln (1 - \sigma(D(\mathbf{x}, \mathbf{z})))] ,$$

ami valójában a bináris keresztentrópiának a „folytonos változata”; másképp írva

$$\int p(\mathbf{z}) \cdot \ln \sigma(D(\mathbf{x}, \mathbf{z})) + q_\theta(\mathbf{z} | \mathbf{x}) \cdot \ln (1 - \sigma(D(\mathbf{x}, \mathbf{z}))) d\mathbf{z}.$$

Ha ez nem győzött meg minket, másképp is igazolhatjuk, hogy a diszkriminátor veszteségfüggvényét célszerű így választani. Korábban láttuk, hogy a<sup>21</sup>

$$p \ln \sigma + q \ln (1 - \sigma),$$

<sup>21</sup> Ez ugyanaz, mint az előbbi egyenlet, csak egyszerűsítettünk a notáción.

típusú mennyiségek szélsőértékhelye egyszerű deriválással és zérusra rendezéssel adódik:

$$\frac{p}{\sigma} - \frac{q}{1-\sigma} = 0 \quad \Rightarrow \quad \sigma = \frac{p}{p+q}.$$

Az optimális  $\sigma$ -t tisztességesen kiírva

$$\begin{aligned} \sigma &= \frac{1}{1 + e^{-D(\mathbf{x}, \mathbf{z})}} = \frac{p(\mathbf{z})}{p(\mathbf{z}) + q_{\theta}(\mathbf{z} | \mathbf{x})} \\ e^{-D(\mathbf{x}, \mathbf{z})} &= \frac{q_{\theta}(\mathbf{z} | \mathbf{x})}{p(\mathbf{z})} \\ D(\mathbf{x}, \mathbf{z}) &= \ln \frac{p(\mathbf{z})}{q_{\theta}(\mathbf{z} | \mathbf{x})}, \end{aligned}$$

azaz a diszkriminátor kimenete a fenti veszteségfüggvénnyel tényleg a keresett mennyiséget közelíti.

A tanulásnál tehát kettős feladatunk van: egyrészt tanítjuk az eredeti modelünket az ELBO maximalizálásával, másrészt a diszkriminátort is a fenti veszteség minimalizálásával. Úgy is mondhatnánk, hogy a modell generatív része megpróbálja minél jobban „utánozni” a  $p(\mathbf{z})$  priort a  $q_{\theta}(\mathbf{z} | \mathbf{x})$  variációs poszteriorral, míg a diszkriminátor arra törekszik, hogy a két eloszlásból származó mintákat megkülönböztesse egymástól. Az így összeállított a modellt *versengő autoenkódernek*<sup>22</sup> nevezzük.

<sup>22</sup> Angolul *adversarial autoencoder*, AAE.

## 6. fejezet

# Expectation-Maximization

Ebben a fejezetben továbbra is olyan modellekkel foglalkozunk, amelyek rejtett változókat tartalmaznak, ám egyszerűbb célt tűzünk ki: csupán ML vagy MAP becslést keresünk<sup>1</sup>. A gradiens-módszer helyett új algoritmust vezetünk be, ami

- Akkor is működik, amikor a log-likelihood gradiense nehezen számolható,
- Mindig érvényes paraméterbecslést ad<sup>2</sup>,
- Sok problémában numerikusan stabilabb és gyorsabban konvergál.

### 6.1 Rejtett változók és maximum likelihood

Emlékezzünk vissza az 5. fejezetben látott szereposztásra. Vegyünk egy valószínűségi modellt, amiben  $\mathbf{x}$  megfigyelt és  $\mathbf{z}$  rejtett változók szerepelnek, majd az (5.1) egyenlet alapján adjunk alsó korlátot a  $p(\mathbf{x} | \theta)$  likelihood logaritmusára<sup>3</sup>:

$$\ln p(\mathbf{x} | \theta) \geq \int q(\mathbf{z}) \ln \frac{p(\mathbf{x}, \mathbf{z} | \theta)}{q(\mathbf{z})} d\mathbf{z} =: \mathcal{L}[q(\mathbf{z})],$$

ahol bevezettük a  $q(\mathbf{z})$  variációs eloszlást a rejtett változók felett. Az előző fejezettel ellentétben  $q(\mathbf{z})$ -t most nem neurális hálózattal vagy más kifinomult módszerrel paraméterezzük, hanem úgy választjuk meg, hogy

$$q(\mathbf{z}) := p(\mathbf{z} | \mathbf{x}, \hat{\theta}),$$

<sup>1</sup> Voltaképpen már a 2-4. fejezetekben is ezt tettük.

<sup>2</sup> Azaz nem kell amiatt aggódnunk, hogy az optimalizáció során esetleg értelmetlen paramétereket kapunk, például egy normális eloszlás szórása negatívnak adódik, ami naiv gradiens-módszerrel könnyen előfordulhat.

<sup>3</sup> Az előző fejezettel ellentétben most nevesítenünk kell a likelihood paramétereit is, amit általánosságban  $\theta$ -val jelöltünk; erre szeretnénk pontbecslést adni.

ahol  $\hat{\theta}$  valamilyen rögzített paraméterezés<sup>4</sup>. Behelyettesítve

$$\begin{aligned}\mathcal{L}[q(\mathbf{z})] &= \int p(\mathbf{z} | \mathbf{x}, \hat{\theta}) \ln p(\mathbf{x}, \mathbf{z} | \theta) d\mathbf{z} - \int p(\mathbf{z} | \mathbf{x}, \hat{\theta}) \ln p(\mathbf{z} | \mathbf{x}, \hat{\theta}) d\mathbf{z} \\ &= \mathbb{E}_{p(\mathbf{z} | \mathbf{x}, \hat{\theta})} [\ln p(\mathbf{x}, \mathbf{z} | \theta)] + \text{const.}\end{aligned}\quad (6.1)$$

adódik, ahonnan csak a  $\theta$ -ban nem konstans tagok érdekelnek minket.

A formula – és az a tény, hogy  $\mathcal{L}$ -t továbbra is maximalizálni szeretnénk – kétlépéses iteratív algoritmust sugall (EM algoritmus):

- **E-lépés.** Rögzítjük  $\hat{\theta}$ -t és kiszámoljuk a várható értéket<sup>5</sup>,
- **M-lépés.** A függvényt maximalizáljuk  $\theta$  szerint; az optimum lesz az új  $\hat{\theta}$ .

Rögtön látjuk, hogy a módszerrel az  $\mathcal{L}$  alsó korlát minden iterációban emelkedik, így előbb-utóbb olyan  $\theta$  paraméterezést találunk, ami a  $p(\mathbf{x} | \theta)$  likelihoodot is maximalizálja.

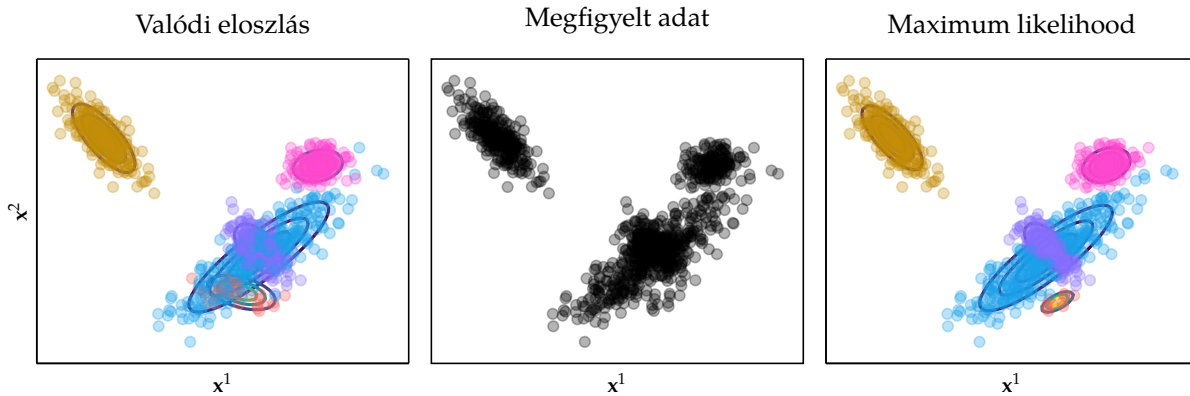
<sup>4</sup> Vegyük észre, hogy ez gyökeresen eltér az előző fejezettől, ahol  $q$ -t szándékosan nagyon általánosnak választottuk, hogy a rejtett változók felett szinte bármilyen eloszlást meg tudjunk tanulni. Most viszont az eloszlás alakja rögzített, és csak a  $\hat{\theta}$  paraméterek frissítésére lesz lehetőségünk, ahogy az ML/MAP megoldásoknál már megszoktuk.

<sup>5</sup> Ami tehát  $\theta$  függvénye lesz; gondolhatunk rá úgy, mint egyfajta negatív várható veszteségre a rejtett változókra vonatkozó, aktuális  $p(\mathbf{z} | \mathbf{x}, \hat{\theta})$  becslésünk mellett.

## 6.2 Klaszterezés Gauss-keverék eloszlásokkal

Nézzük meg az EM algoritmust egy konkrét példán. Klaszterezésnek azt a feladatot nevezzük, amikor az  $\mathbf{x}_i$  adatpontjainkat csoportokba (*klaszterekbe*) szeretnénk sorolni, ám az osztályozással ellentétben a csoportok előre nem definiáltak<sup>6</sup>. A klaszterek száma legyen  $K$ ; feltesszük, hogy minden egyes klaszter egy-egy normális eloszlással modellezhető, amelyek paramétereit keressük.

<sup>6</sup> Azaz nincsenek megfelelő  $y_i$  címkek; ezt a felállást *nemellenőrzött* tanulásnak is szokás nevezni.



6.1. ábra. Klaszterezés két dimenzióban Gauss-keverék eloszlással.



### 6.2.1 Valószínűségi modell

Jelölje  $\mathbf{z}_i$  az  $i$ . mintához tartozó valódi klasztert – ő lesz a rejtett változónk, amelynek eloszlását kategorikusnak választjuk a  $K$  darab lehetséges klaszternek megfelelően<sup>7</sup>:

$$p(\mathbf{z}_i | \boldsymbol{\pi}) = \prod_{k=1}^K \pi_k^{z_{ik}},$$

<sup>7</sup> A  $\pi_k$  személetes jelentése „annak az *a priori* valószínűsége, hogy egy minta a  $k$ . klaszterbe esik”.

ahol a többsztályos osztályozáshoz hasonlóan  $\mathbf{z}_i$ -re *one-hot* módon kódolt vektorként gondolunk:

$$\mathbf{z}_i = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^\top,$$

a paramétereikről pedig tudjuk, hogy

$$\sum_k \pi_k = 1.$$

A minták eloszlását így a következőképpen írhatjuk:

$$p(\mathbf{x}_i | \mathbf{z}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{ik}},$$

ahol a  $\boldsymbol{\mu}_k$  és  $\boldsymbol{\Sigma}_k$  a  $k$ . klaszter várható értéke illetve kovarianciamátrixa<sup>8</sup>; kihasználtuk továbbá, hogy  $\mathbf{z}_i$  vektor pontosan egy eleme 1, a többi zérus; ez az elem mindig egyetlen klasztert választ ki.

<sup>8</sup> Ezek szemléletesen a klaszter „központját” és „alaját” képviselik.

A becsülendő paraméterek a fent felsorolt paraméterekből állnak össze, nevezetesen,

$$\boldsymbol{\theta} := \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$$

A likelihood pedig – az egyszerűség kedvéért most csak egy mintára felírva – a teljes valószínűség tételének felhasználásával adódik:

$$\begin{aligned} p(\mathbf{x}_i | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \sum_{\mathbf{z}_i} p(\mathbf{z}_i | \boldsymbol{\pi}) p(\mathbf{x}_i | \mathbf{z}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= \sum_k \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \end{aligned} \quad (6.2)$$

azaz tényleg normális eloszlásoknak egyfajta keverékéről, súlyozott átlagáról beszélhetünk.

## 6.2.2 Expectation lépés

Első lépésként behelyettesítjük az ismert mennyiségeket a (6.1) egyenletbe<sup>9</sup>:

<sup>9</sup>Most már az összes mintára együttesen felírva.

$$\begin{aligned}
 \mathcal{L} &= \mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} [\ln p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})] \\
 &= \mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} \left[ \ln \left( \prod_i p(\mathbf{x}_i \mid \mathbf{z}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\mathbf{z}_i \mid \boldsymbol{\pi}) \right) \right] \\
 &= \mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} \left[ \sum_i \ln \left( \prod_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{ik}} \cdot \pi_k^{z_{ik}} \right) \right] \\
 &= \sum_i \sum_k \mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} [z_{ik}] (\ln \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \ln \pi_k),
 \end{aligned}$$

ahol kihasználtuk a várható érték linearitását. Az E-lépés tehát lényegében az  $\mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} [z_{ik}]$  mennyiség kiszámítását takarja, amelyre

$$\begin{aligned}
 \mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} [z_{ik}] &= \mathbb{E}_{p(\mathbf{z}_i \mid \mathbf{x}_i, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} [z_{ik}] \\
 &= \sum_{\mathbf{z}_i} z_{ik} p(\mathbf{z}_i \mid \mathbf{x}_i, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) \\
 &= \frac{\sum_{\mathbf{z}_i} z_{ik} p(\mathbf{x}_i \mid \mathbf{z}_i, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) p(\mathbf{z}_i \mid \hat{\boldsymbol{\pi}})}{p(\mathbf{x}_i \mid \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} \\
 &= \frac{\hat{\pi}_k \mathcal{N}(\mathbf{x}_i \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)}{\sum_k \hat{\pi}_k \mathcal{N}(\mathbf{x}_i \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)},
 \end{aligned}$$

ahol a harmadik egyenlőségnél a Bayes-tételt, a negyediknél pedig a (6.2) egyenletet használtuk fel. A kapott

$$\gamma_{ik} := \frac{\hat{\pi}_k \mathcal{N}(\mathbf{x}_i \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)}{\sum_k \hat{\pi}_k \mathcal{N}(\mathbf{x}_i \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)}$$

mennyiséget *responsibility* névvel illetjük, és a formulát megvizsgálva szemléletes jelentést is társíthatunk hozzá: „annak mértéke, hogy a  $k$ . klaszter mennyire felelős az  $i$ . mintáért”. Ha ugyanis a számláló magas értéket vesz fel, az azt jelenti, hogy az  $\mathbf{x}_i$  minta nagy valószínűségű a  $k$  klasztert képviselő normális eloszlás szerint, például azért, mert közel van a középpontjához. A nevező a normalizálásért felel: összességében az egész számítás meglehetősen hasonlít egyfajta súlyozott softmax függvényre.

### 6.2.3 Maximization lépés

Az E-lépés után az alsó korlát a következőképpen fest:

$$\mathcal{L} = \sum_i \sum_k \gamma_{ik} (\ln \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \ln \pi_k).$$

A maximalizálást először  $\boldsymbol{\mu}$  szerint végezzük el a szokásos módon<sup>10</sup>:

$$\nabla_{\boldsymbol{\mu}_k} \mathcal{L} = - \sum_i \gamma_{ik} \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0,$$

ahonnan  $-\boldsymbol{\Sigma}^{-1}$ -zel történő egyszerűsítés és átrendezés után adódik, hogy

$$\boldsymbol{\mu}_k \cdot \underbrace{\sum_i \gamma_{ik}}_{:=N_k} = \sum_i \gamma_{ik} \mathbf{x}_i.$$

Végül a  $\boldsymbol{\mu}_k$  paraméterek frissítésére a

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_i \gamma_{ik} \mathbf{x}_i$$

meglehetősen egyszerű formulát kapjuk. A klaszterközéppontok tehát az összes minta súlyozott átlagaként képződnek, ahol a súlyokat a megfelelő responsibility-értékek adják<sup>11</sup>.

Hasonlóan levezethető, hogy a kovarianciamátrixok frissítése a

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_i \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top$$

egyenlet alapján történik. Végül nézzük meg a  $\pi_k$  paraméterek frissítését. Elhagyva a  $\pi_k$ -t nem tartalmazó tagokat, a megoldandó feladat

$$\begin{aligned} \max_{\pi_k} \quad & \sum_i \sum_k \gamma_{ik} \ln \pi_k \\ \text{s.t.} \quad & \sum_k \pi_k = 1, \end{aligned}$$

amelyet a Lagrange-multiplikátoros módszerrel oldhatunk meg. A módosított célfüggvény

$$\sum_i \sum_k \gamma_{ik} \ln \pi_k - \lambda \left( \sum_k \pi_k - 1 \right),$$

<sup>10</sup> Emlékezzünk vissza, hogy a normális eloszlás sűrűségfüggvénye

$$\frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)}.$$

<sup>11</sup> Korábbi tanulmányainkból ismerts lehet a  $K$ -means algoritmus, ami lényegében speciális esete a mi algoritmusunknak. Míg ott minden mintát egyértelműen hozzárendelünk egy-egy klaszterhez, nálunk a klaszterek osztosznak a „felelősségen”.

amelyet  $\pi_k$  szerint deriválva és zérusra állítva

$$\frac{\sum_i \gamma_{ik}}{\pi_k} - \lambda = \frac{N_k}{\pi_k} - \lambda = 0 \Rightarrow \pi_k = \frac{N_k}{\lambda}.$$

Szummázzuk ki az utóbbi egyenletet  $k$  szerint; mivel  $\sum_k \pi_k = 1$  és legyen

$$\sum_k N_k := N,$$

arra jutunk, hogy

$$1 = \frac{N}{\lambda} \Rightarrow \lambda = N.$$

A  $\pi_k$  valószínűségek frissített értéke tehát a

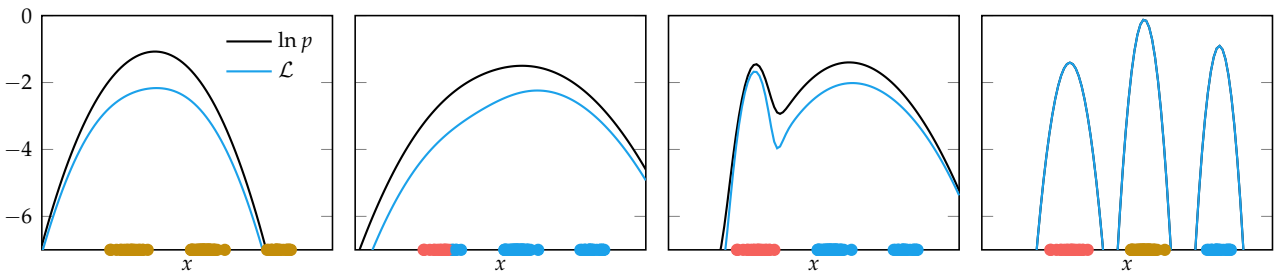
$$\pi_k = \frac{N_k}{N}$$

egyenlet alapján, a megfelelő klaszter responsibility értékeinek összegéből képezhető.

A tanulás után az  $i$ . mintához jószolt klasztert a responsibility vektorból nyerhetjük ki:

$$\hat{z}_i = \arg \max_k \gamma_{ik}.$$

A teljes eljárást a 6.1. algoritmus mutatja be, a 6.2. ábra pedig egy dimenzióban szemlélteti az eredményeket.



Az ábrán jól láthatjuk az aktuális paraméterek alapján számolt, az egyes mintákra vonatkoztatott log-likelihood és alsó korlát alakulását (figyelem: az eddig tárgyalt teljes log-likelihoodhoz és alsó korláthoz még a minták szerint összegezni kell, és a szigorú monoton növekedés is csak az összes mintán számolt alsó korlátra érvényes!).

6.2. ábra. Klaszterezés egy dimenzióban. A minták az  $x$  tengelyen láthatók, a színek a jószolt klaszter-hozzárendelést jelölik. Látható, hogy a futás végére (10 iteráció alatt) megtaláltuk a helyes klaszterezést.

```

struct EMClustering
    K # klaszerek száma
     $\pi$  # prior beletartozás
     $\mu$  # középpontok
     $\Sigma$  # kovarianciamátrixok
     $\gamma$  # responsibility
    z # hozzárendelések

    EMClustering(D,M,K) = new(
        K,
        ones(K)/K,
        [randn(D)          for k in 1:K],
        [Matrix(I(D)*1.)  for k in 1:K],
        [zeros(M)         for k in 1:K],
        zeros{Int,M}
    )
end

function train!(m::EMClustering, X; iters=1000)
    for i in 1:iters
        p = [ m. $\pi$ [k]*pdf(MultivariateNormal(m. $\mu$ [k],m. $\Sigma$ [k]),X)
              for k in 1:m.K ]
        for k in 1:m.K
            m. $\gamma$ [k] .= p[k]./sum(p)
        end
        N = sum.(m. $\gamma$ )

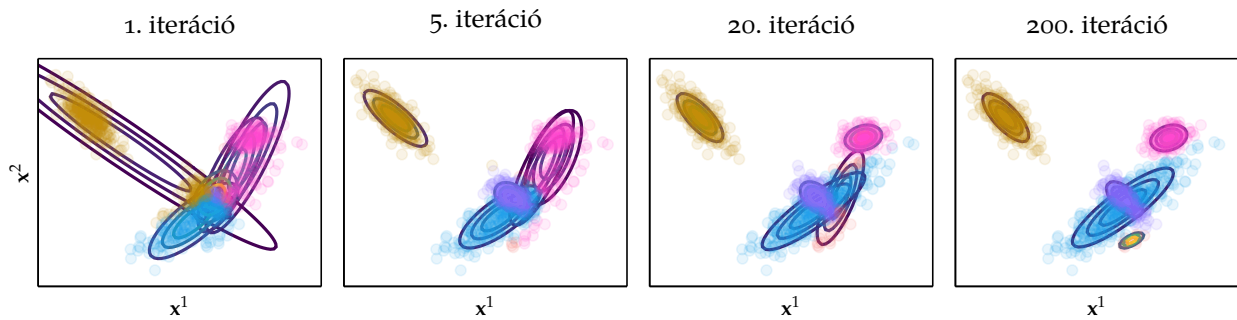
        for k in 1:m.K
            d = X .- m. $\mu$ [k]
            g = m. $\gamma$ [k] ./ N[k]
            m. $\Sigma$ [k] .= Symmetric(g'*d*d')
            m. $\mu$ [k] .= X*g
        end

        m. $\pi$  .= N/sum(N)
    end
    m.z .= argmax.(vcat.(m. $\gamma$ ...))
end

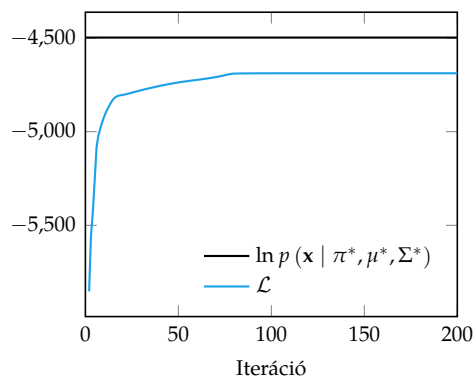
```

6.1. algoritmus. Klaszterezés Gauss-keverék eloszlásokkal és az EM algoritmussal. Az eljárás nem igényel gradiens-alapú optimalizációt. A paraméterek frissítésénél ügyelnünk kell a sorrendre: először célszerű a kovarianciamátrixokat frissíteni, és csak utána a várható értékeket. Ellenkező esetben a kovarianciamátrixokat már  $\mu$  frissített értékei alapján számolnánk, ami felboríthatja az algoritmus működését.

Végül nézzük meg a klaszterezést két dimenzióban:



Az alsó korlát alakulását – immár a teljes mintán – a 6.4. ábra mutatja.



6.3. ábra. Klaszterezés két dimenzióban. Az illesztett keverékeloszlást szintvonalakkal ábrázoltuk.

6.4. ábra. Az alsó korlát változása az iterációk számával. Látjuk, hogy  $\mathcal{L}$  minden körben emelkedik, közelít a valódi  $\ln p(\mathbf{x} \mid \pi^*, \mu^*, \Sigma^*)$  likelihoodhoz.

## 7. fejezet

# Markov Chain Monte Carlo

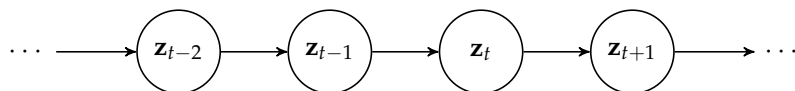
Ebben a fejezetben megismerünk egy módszer családot, amellyel szinte bármilyen modellben végezhetünk valószínűségi következtetést. Az MCMC eszköztár felhasználásával egy ismeretlen eloszlásból tudunk mintavételezni, vagyis az eloszlást a gyűjtött minták összességével közelíthetjük. A variációs technikákkal szemben a Monte Carlo módszerek egzakt következtetést nyújtanak<sup>1</sup>, cserébe viszont rendszerint lassabbak a variációs módszereknél<sup>2</sup>.

### 7.1 Következtetés mintavétellel

Az eljáráshoz először meg kell ismerkednünk a *Markov-láncokkal*. Elsőrendű Markov-lánc alatt egy olyan sztochasztikus folyamatot értünk, ahol egy  $\mathbf{z}_t$  jövőbeli állapot a  $\mathbf{z}_{t-1}$  jelenbeli állapot ismeretében feltételesen független a  $\mathbf{z}_{t-2}, \mathbf{z}_{t-3}, \dots$  múltbeli állapotoktól, azaz formálisan

$$p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \dots) = p(\mathbf{z}_t \mid \mathbf{z}_{t-1}).$$

A  $\mathbf{z}$  állapotok tehát valószínűségi természetűek; a láncot a valószínűségi gráfos modellek nyelvén így ábrázolhatjuk:



<sup>1</sup> Pontosabban aszimptotikusan egzakt következtetést – végtelen sok mintával garantáltan megtalálnánk a valódi eloszlást.

<sup>2</sup> Tréfásan úgy szokás mondani, hogy a „variációs módszer az, amit addig implementálunk, amíg várjuk, hogy a Markov-láncunk konvergáljon”.

7.1. ábra. Elsőrendű Markov-lánc diagramja.

A  $t$ . időpontban lévő eloszlást könnyen kifejezhetjük, ha a teljes valószínűség tételét használjuk, nevezetesen

$$p(\mathbf{z}_t) = \sum_{\mathbf{z}_{t-1}} \underbrace{p(\mathbf{z}_t | \mathbf{z}_{t-1})}_{:=T(\mathbf{z}_{t-1}, \mathbf{z}_t)} p(\mathbf{z}_{t-1}).$$

A  $T(\mathbf{z}_{t-1}, \mathbf{z}_t)$  objektum az állapotátmeneti valószínűségekről tájékoztat. Ha  $\mathbf{z}_t$  és  $\mathbf{z}_{t-1}$  véges sok értéket vehet fel, akkor  $T(\mathbf{z}_{t-1}, \mathbf{z}_t)$ -t szokás mátrixként ábrázolni és állapotátmeneti mátrixnak hívni; például

$$T_{ij}^{(t)} := p(\mathbf{z}_t = j | \mathbf{z}_{t-1} = i)$$

annak a valószínűsége, hogy az  $i$ . állapotból a  $j$ . állapotba ugrunk a  $t$ . időpontban. Ha az állapotátmeneti valószínűségek nem függnak  $t$ -től, a Markov-láncot *homogénnek* hívjuk. Ezt mostantól fel is fogjuk tenni, az állapotokat illetően viszont általánosabbak leszünk, és folytonos értékű  $\mathbf{z}$ -ket is megengedünk.

A Markov-láncok egy fontos tulajdonsága, hogy létezhetnek ún. *invariáns eloszlások*, amelyeket az állapotátmenet „változtatlanul hagy”<sup>3</sup>, azaz

$$p^*(\mathbf{z}) = \sum_{\mathbf{z}'} p^*(\mathbf{z}') T(\mathbf{z}', \mathbf{z}).$$

A  $p^*$  eloszlás invarianciájára elégséges feltétel a *részletes egyensúlyi feltétel*<sup>4</sup>:

$$p^*(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') = p^*(\mathbf{z}') T(\mathbf{z}', \mathbf{z}).$$

Ezt könnyen bizonyíthatjuk, ugyanis ha az előbbi fennáll, akkor

$$\begin{aligned} \sum_{\mathbf{z}'} p^*(\mathbf{z}') T(\mathbf{z}', \mathbf{z}) &= \sum_{\mathbf{z}'} p^*(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') \\ &= p^*(\mathbf{z}) \sum_{\mathbf{z}'} T(\mathbf{z}, \mathbf{z}') \\ &= p^*(\mathbf{z}) \underbrace{\sum_{\mathbf{z}'} p(\mathbf{z}' | \mathbf{z})}_{=1} \\ &= p^*(\mathbf{z}) \end{aligned}$$

Az MCMC módszerek lényege az, hogy olyan Markov-láncot konstruálunk, amelynek az invariáns eloszlása a keresett, ismeretlen eloszlás; a láncot „járatva” ebből az eloszlásból veszünk sok-sok mintát<sup>5</sup>.

<sup>3</sup> Ilyenből több is lehet. Ha például az állapotátmenet az identitás-leképezés, akkor minden eloszlás triviálisan invariáns eloszlás lesz.

<sup>4</sup> Angolul *detailed balance*.

<sup>5</sup> Eképpen gondolkodunk: „bár az eloszlást nem tudom zárt alakban felírni, mintavételezni talán tudok belőle, és elég mintával mindent megtudhatok róla”.



### 7.1.1 Metropolis–Hastings algoritmus

A célunk tehát az, hogy olyan Markov-láncot konstruáljunk, amely a kedvenc  $p$  eloszlásunkból mintavételez<sup>6</sup>. Ahhoz, hogy  $p$  invariáns eloszlás legyen, elegendő a részletes egyensúlyi feltételt kikényszeríteni. Ehhez definiáljuk az állapotátmeneti valószínűséget a következőképpen:

$$T(\mathbf{z}, \mathbf{z}') := \underbrace{q(\mathbf{z}' | \mathbf{z})}_{\text{„proposal”}} \underbrace{A(\mathbf{z}, \mathbf{z}')}_{\text{„acceptance”}}.$$

A  $q$  eloszlást *proposal* névvel illetjük, a feladata pedig az, hogy a  $\mathbf{z}$  aktuális állapot alapján új  $\mathbf{z}'$  állapotot „javasoljon”<sup>7</sup>. Az  $A$  valószínűség azt mondja meg, hogy mekkora valószínűséggel fogadjuk el a  $q$ -ból vett mintát, azaz a javasolt követő állapotot.  $A$ -t így definiáljuk:

$$A(\mathbf{z}, \mathbf{z}') := \min \left( 1, \frac{p(\mathbf{z}')q(\mathbf{z} | \mathbf{z}')}{p(\mathbf{z})q(\mathbf{z}' | \mathbf{z})} \right).$$

Ez a választás bizonyos értelemben nagyon kényelmessé teszi a dolgunkat. A  $p$  eloszlás „nehéz” része ugyanis mindig a normalizációs konstans<sup>8</sup>, amelyet kiírva

$$p(\mathbf{z}) = \frac{1}{Z_p} \tilde{p}(\mathbf{z}).$$

Vegyük észre, hogy  $A$ -ban a  $Z_p$  konstans a számlálóban és a nevezőben is szerepel, így kiesik a törtből és elég a nem normalizált valószínűségeket kiértékelni:

$$A(\mathbf{z}, \mathbf{z}') = \min \left( 1, \frac{\tilde{p}(\mathbf{z}')q(\mathbf{z} | \mathbf{z}')}{\tilde{p}(\mathbf{z})q(\mathbf{z}' | \mathbf{z})} \right),$$

amit könnyen megtehetünk, hiszen csak a  $\tilde{p}$ -ba kell az aktuális állapotot behelyettesítenünk<sup>9</sup>. Végül bizonyítsuk be, hogy ezekkel a választásokkal teljesül a részletes egyensúlyi feltétel:

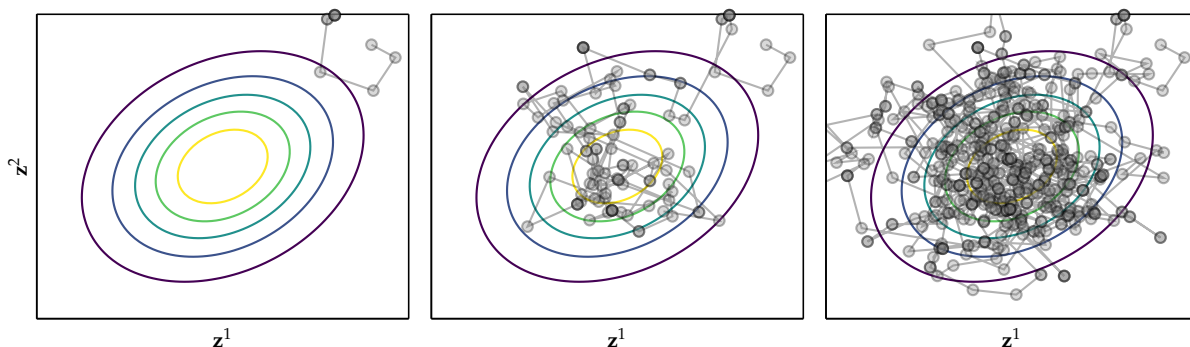
$$\begin{aligned} p(\mathbf{z})T(\mathbf{z}, \mathbf{z}') &= p(\mathbf{z})q(\mathbf{z}' | \mathbf{z}) A(\mathbf{z}, \mathbf{z}') \\ &= p(\mathbf{z})q(\mathbf{z}' | \mathbf{z}) \min \left( 1, \frac{p(\mathbf{z}')q(\mathbf{z} | \mathbf{z}')}{p(\mathbf{z})q(\mathbf{z}' | \mathbf{z})} \right) \\ &= \min (p(\mathbf{z})q(\mathbf{z}' | \mathbf{z}), p(\mathbf{z}')q(\mathbf{z} | \mathbf{z}')) \\ &= p(\mathbf{z}')q(\mathbf{z} | \mathbf{z}') \min \left( \frac{p(\mathbf{z})q(\mathbf{z}' | \mathbf{z})}{p(\mathbf{z}')q(\mathbf{z} | \mathbf{z}')}, 1 \right) \\ &= p(\mathbf{z}')q(\mathbf{z} | \mathbf{z}') A(\mathbf{z}', \mathbf{z}) \\ &= p(\mathbf{z}')T(\mathbf{z}', \mathbf{z}). \end{aligned}$$

<sup>6</sup> A  $p^*$  jelölés helyett mostantól egyszerűen  $p$ -t írunk.

<sup>7</sup> Ezért könnyen mintavételezhetőnek választjuk, leggyakrabban egy aktuális állapotra centrált normális eloszlásnak.

<sup>8</sup> Jusszon eszünkbe, hogy ebben mindig valamiféle integrál szerepel, hiszen ez gondoskodik arról, hogy a görbe alatti terület egységnyi legyen.

<sup>9</sup> Aktuális állapot alatt most egy teljes változó-konfigurációt értünk, ami tartalmazza megfigyelt és rejtett változókat is; a következtetés során lényegében véletlenszerű tippeket adunk a rejtett változók értékére, miközben az elfogadási valószínűségek úgy vannak beállítva, hogy a tippek összessége a valódi, ismeretlen eloszlást fogja közelíteni.



7.2. ábra. Normális eloszlás közelítése Metropolis–Hastings algoritmussal. Az algoritmus véletlen bolyongást végez, fontos azonban, hogy az így kapott állapotok a valódi eloszlást közelítik.

Az eljárást a 7.2. ábra és a 7.1. algoritmus szemlélteti. A Metropolis–Hastings algoritmus véletlen bolyongást végez, amelynek során a valószínű konfigurációkat preferálja; emellett kis valószínűséggel ugyan, de előfordulnak valószínűtlen állapotok is. A bizonyításunk alapján a gyűjtött minták a keresett eloszlást közelítik.

Az MCMC algoritmusoknak még számos válfaja ismert, amelyek közül csak az egyszerűbbeket tárgyaljuk. A leggyakrabban használt variánsok a proposal-ok előállításánál Langevin-diffúziót vagy Hamilton-mechanikát (pl. NUTS) használnak, így lényegesen gyorsabban konvergálnak és hatékonyabban járják be a teret („mixing”), mint a Metropolis–Hastings algoritmus, és jóval nagyobb dimenzionalitás mellett is használhatók. A konvergencia és mixing vizsgálata, a paraméterek finomhangolása, és általában a Markov-láncok diagnosztikája külön művészet, amit itt nem tudunk tárgyalni.

### 7.1.2 Metropolis algoritmus

A Metropolis algoritmus szimmetrikus proposal eloszlást használ:

$$q(\mathbf{z} \mid \mathbf{z}') = q(\mathbf{z}' \mid \mathbf{z}),$$

így az elfogadási valószínűségek a következő egyszerű formát öltik<sup>10</sup>:

$$A(\mathbf{z}, \mathbf{z}') = \min \left( 1, \frac{\tilde{p}(\mathbf{z}')}{\tilde{p}(\mathbf{z})} \right).$$

<sup>10</sup> Vegyük észre, hogy ez a módszer nem nyilatkozik az új állapotok előállításáról; ezt bárhogyan megtehetjük.

```

function MetropolisHastings(p, z; σ=1.0, num_samples=1000)
    samples = zeros(num_samples, length(z))

    for i in 1:num_samples
        qz = MultivariateNormal(z, σ)
        z' = rand(qz)
        qz' = MultivariateNormal(z', σ)

        qz'z, qzz' = pdf(qz', z), pdf(qz, z')
        pz, pz' = pdf(p, z), pdf(p, z')
        A = min(1, pz' * qzz' / (pz * qz'z))

        if rand() < A
            z .= z'
        end
        samples[i, :] .= z
    end
    return samples
end

```

7.1. algoritmus. Metropolis–Hastings algoritmus.

## 7.2 Gibbs-mintavétel

A Gibbs-mintavétel a Metropolis–Hastings algoritmusnak egy különösen hatékony változata, amelyet konjugált eloszlásoknál használhatunk. A  $\mathbf{z}$  állapotot – amely a modellünk megfigyelt és rejtett változóit tartalmazza – most elemenként fogjuk tekinteni, azaz jelölje  $\mathbf{z}^k$  a modellünk  $k$ . változóját,  $\mathbf{z}^{\setminus k}$  pedig az összes többi változót. A Gibbs-mintavétel egyszerre csak egyetlen változót frissít és az alábbi feltételes eloszlásokat használja proposal-ként<sup>11</sup>:

$$\begin{aligned}
 q(\mathbf{z} \mid \mathbf{z}') &:= p(\mathbf{z}^k \mid \mathbf{z}'^{\setminus k}), \\
 q(\mathbf{z}' \mid \mathbf{z}) &:= p(\mathbf{z}'^k \mid \mathbf{z}^{\setminus k}).
 \end{aligned}$$

<sup>11</sup> Innen a megkötés: ki kell számolnunk a  $k$ . változó feltételes eloszlását a többi változó ismeretében. Ezt konjugált modelleknél tehetjük meg zárt alakban.

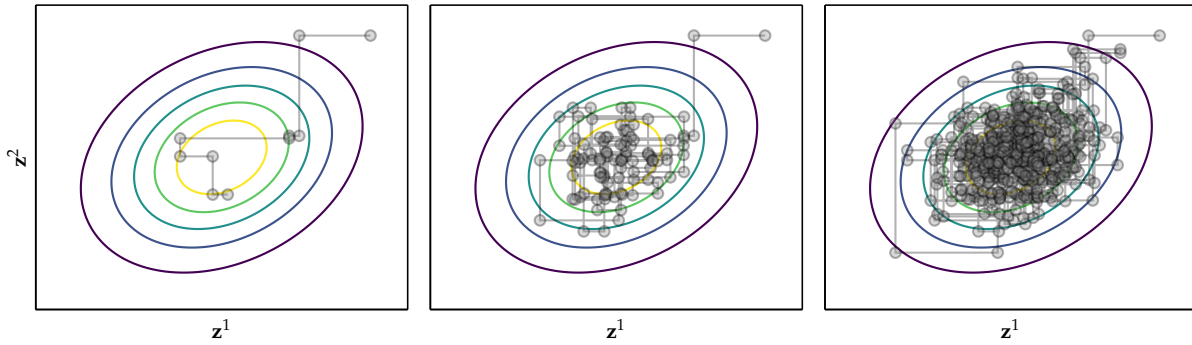
A teljes valószínűség tétele szerint

$$p(\mathbf{z}') = p(\mathbf{z}'^k \mid \mathbf{z}'^{\setminus k}) p(\mathbf{z}'^{\setminus k}),$$

amelyet behelyettesítünk az elfogadási valószínűségekre vonatkozó formulába:

$$A(\mathbf{z}, \mathbf{z}') = \min \left( 1, \frac{p(\mathbf{z}'^k | \mathbf{z}'^{\setminus k}) p(\mathbf{z}'^{\setminus k}) p(\mathbf{z}^k | \mathbf{z}'^{\setminus k})}{p(\mathbf{z}^k | \mathbf{z}^{\setminus k}) p(\mathbf{z}^{\setminus k}) p(\mathbf{z}'^k | \mathbf{z}^{\setminus k})} \right) = 1.$$

Itt kihasználtuk, hogy  $\mathbf{z}'^{\setminus k} = \mathbf{z}^{\setminus k}$ . Ezzel egyfajta iteratív sémát kaptunk, amely úgy néz ki, hogy minden iterációban csupán egy-egy változó értékét frissítjük, míg a többit változatlanul hagyjuk; az elfogadási valószínűség az előbbiek alapján 1-nek adódik, másképpen, az így javasolt követő állapotok mindig elfogadásra kerülnek. Erre úgy is gondolhatunk, hogy a Gibbs-mintavétel mindig a tengelyekkel párhuzamosan lép:



7.3. ábra. Normális eloszlás közelítése Gibbs-mintavétellel.

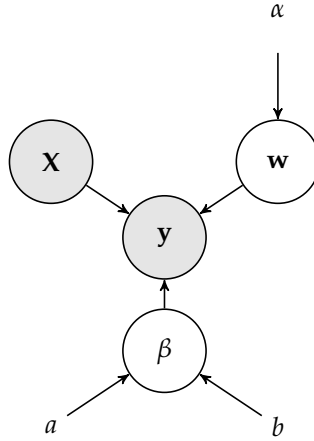
### 7.2.1 Bayesi lineáris regresszió Gibbs-mintavétellel

Írjuk le ismét a bayesi lineáris regresszióra vonatkozó valószínűségi modellt azzal a különbséggel, hogy most a  $\beta$  pontosságot is tanuljuk, azaz priort teszünk rá<sup>12</sup>:

$$\begin{aligned} p(y_i | \mathbf{w}, \mathbf{x}_i, \beta) &= \mathcal{N}(y_i | \mathbf{w}^\top \phi(\mathbf{x}_i), \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} e^{-\frac{\beta}{2} (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2}, \\ p(\mathbf{w} | \alpha) &= \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I}) = \sqrt{\frac{\alpha}{2\pi}} e^{-\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}}, \\ p(\beta | a, b) &= \mathcal{Ga}(\beta | a, b) = \frac{b^a}{\Gamma(a)} \beta^{a-1} e^{-b\beta}. \end{aligned}$$

A bayesi következtetés során  $\mathbf{X}$  és  $\mathbf{y}$  játsszák a megfigyelt változók szerepét,  $\beta$  és  $\mathbf{w}$  pedig a rejtett változók, amelyekre nézve a poszterior eloszlást keressük.

<sup>12</sup> Emlékezzünk vissza, hogy a 2. fejezetben  $\beta$  értékét csupán beállítottuk valamire.



7.4. ábra. Lineáris regresszió valószínűségi modellje, ahol a pontos-ságot is tanuljuk. A szürke csomópontok megfigyelt változókat (adatokat) jelölnék, a nem bekarikázott csomópontok pedig a hiperparaméterek.

A Gibbs-mintavételhez szükségünk van a rejtett változókra felírt feltételes eloszlásokra, azaz

- $p(\beta | \mathbf{y}, \mathbf{w}, \mathbf{X}, a, b)$ -re, ahol a  $\beta$  változót leszámítva minden mást rögzítettünk,
- $p(\mathbf{w} | \mathbf{y}, \mathbf{X}, \beta, \alpha)$ -ra, ahol a  $\mathbf{w}$  változót leszámítva minden mást rögzítettünk.

Tekintsük az  $\mathbf{y}$ ,  $\mathbf{w}$  és  $\beta$  változók együttes eloszlását:

$$\begin{aligned}
 p(\mathbf{y}, \mathbf{w}, \beta | \mathbf{X}, \alpha, a, b) &= \prod_{i=1}^N p(y_i | \mathbf{w}, \mathbf{x}_i, \beta) \cdot p(\mathbf{w} | \alpha) \cdot p(\beta | a, b) \\
 &= \left( \frac{\beta}{2\pi} \right)^{\frac{N}{2}} e^{-\frac{\beta}{2} \sum_{i=1}^N (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2} e^{-\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}} \cdot \beta^{a-1} e^{-b\beta} \cdot \text{const.}
 \end{aligned}$$

A feltételes eloszlásokat úgy kaphatjuk meg, hogy ebből a megfelelő változót tartalmazó tagokat kigyűjtjük. Ezzel arányosság erejéig meg is van az eloszlás (hiszen a többi tag nem tartalmazza az adott változót), és ha jól választottuk meg a modellünket, a szükséges eloszlást és paramétereket le is olvashatjuk<sup>13</sup>. A  $\beta$

<sup>13</sup> Ahogy ezt a konjugált eloszlásoknál már megszokhattuk.

változó esetén például a következőképpen gondolkodunk:

$$\begin{aligned} p(\beta \mid \mathbf{y}, \mathbf{w}, \mathbf{X}, a, b) &= \beta^{\frac{N}{2}+a-1} \cdot e^{-\left(b+\frac{1}{2}\sum_{i=1}^N (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2\right)\beta} \cdot \text{const.} \\ &= \mathcal{G}a\left(\beta \mid a + \frac{N}{2}, b + \frac{1}{2}\sum_{i=1}^N (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2\right), \end{aligned}$$

azaz  $\beta$ -ra gamma-eloszlást kaptunk a fenti paraméterekkel. A  $\mathbf{w}$  változóra vonatkozó feltételes eloszlás

$$p(\mathbf{w} \mid \mathbf{y}, \mathbf{X}, \beta, \alpha) = e^{-\frac{\beta}{2}\sum_{i=1}^N (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2 - \frac{\alpha}{2}\mathbf{w}^\top \mathbf{w}} \cdot \text{const},$$

amit még a megfelelő alakra kell hoznunk. Ehhez felbontjuk a hatványkitevőben szereplő zárójelet:

$$\begin{aligned} &\sum_i (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2 \\ &= -2\sum_i y_i \cdot \phi(\mathbf{x}_i)^\top \mathbf{w} + \sum_i \mathbf{w}^\top \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_i)^\top \mathbf{w} + \text{const.} \\ &= -2\left(\sum_i y_i \cdot \phi(\mathbf{x}_i)\right)^\top \mathbf{w} + \mathbf{w}^\top \left(\sum_i \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top\right) \mathbf{w} + \text{const.} \\ &= -2\left(\Phi^\top \mathbf{y}\right)^\top \mathbf{w} + \mathbf{w}^\top \Phi^\top \Phi \mathbf{w} + \text{const.} \end{aligned}$$

A kitevőben összevonva a lineáris ill. kvadratikus tagokat, majd a teljes négyzetté kiegészítés módszerét használva

$$\begin{aligned} p(\mathbf{w} \mid \mathbf{y}, \mathbf{X}, \beta, \alpha) &= e^{-\frac{1}{2}\mathbf{w}^\top (\beta\Phi^\top \Phi + \alpha\mathbf{I})\mathbf{w} + (\beta\Phi^\top \mathbf{y})^\top \mathbf{w}} \cdot \text{const.} \\ &= \text{const.} \cdot e^{-\frac{1}{2}(\mathbf{w}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{w}-\boldsymbol{\mu})} \\ &= \mathcal{N}(\mathbf{w} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \end{aligned}$$

adódik, ahol

$$\begin{aligned} \boldsymbol{\mu} &= (\beta\Phi^\top \Phi + \alpha\mathbf{I})^{-1} (\beta\Phi^\top \mathbf{y}), \\ \boldsymbol{\Sigma} &= \beta\Phi^\top \Phi + \alpha\mathbf{I}. \end{aligned}$$

A teljes négyzetté kiegészítés a középiskolaival teljesen analóg módon történik. Most egy

$$-\frac{1}{2} \mathbf{w}^\top \mathbf{A} \mathbf{w} + \mathbf{b}^\top \mathbf{w} + \text{const.}$$

alakú egyenletünk van, amire egyváltozósan így járnánk el:

$$-\frac{1}{2} a w^2 + b w + \text{const.} = -\frac{1}{2} a \left( w - \frac{b}{a} \right)^2 + \text{const.}'$$

Többváltozósan ez

$$-\frac{1}{2} \left( \mathbf{w} - \mathbf{A}^{-1} \mathbf{b} \right)^\top \mathbf{A} \left( \mathbf{w} - \mathbf{A}^{-1} \mathbf{b} \right) + \text{const.}$$

Az eljárást a 7.2. algoritmus mutatja, az eredmények a 7.5. és a 7.6. ábrán láthatók.

```
function LR_Gibbs(X, y; α=0.1, a₀=1, b₀=1, iters=1000, burnin=100)
    N,D = size(X)
    w = zeros(D)
    ws = zeros(iters,D)
    βs = zeros(iters)

    for it in 1:(iters+burnin)
        d = y .- X*w
        β = rand(Gamma(a₀ + N/2, 1/(b₀ + sum(d.^2)/2)))
        w .= rand(MvNormalCanon(β*X'y, β*X'X + α*I))

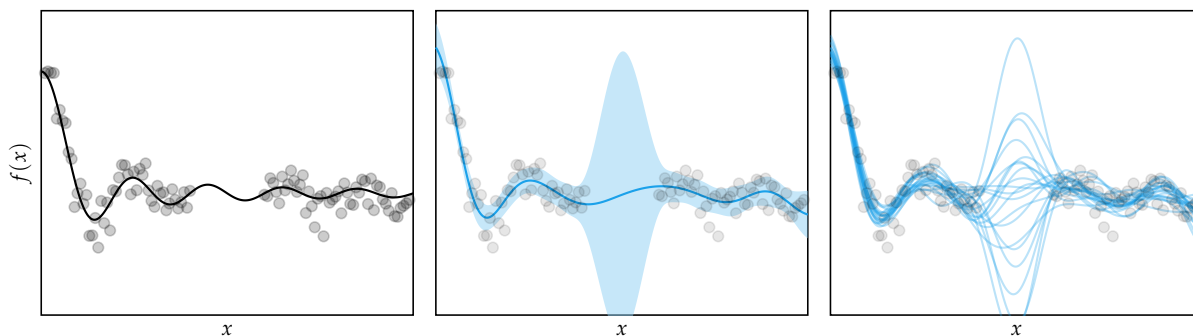
        if it>burnin
            βs[it-burnin] = β
            ws[it-burnin,:] = w
        end
    end
    return βs, ws
end
```

7.2. algoritmus. Bayesi lineáris regresszió Gibbs-mintavétellel. Az eloszlások paraméterezése eltér a korábban írtaktól: a legtöbb könyvtár a gamma-eloszlásnál a  $b$  paraméter reciprokát várja, a többváltozós normális eloszlást pedig ún. kanonikus paraméterekkel adtuk meg (ami lényegében megegyezik a fentivel, de egy invertálást meg lehet úszni ezen a módon).

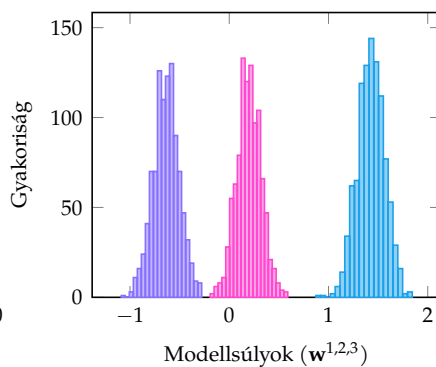
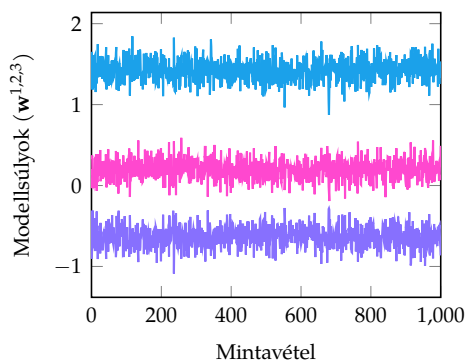
### 7.2.2 Changepoint-modellek

A changepoint-modellek egy idősorban beálló hirtelen változást modelleznek<sup>14</sup>.

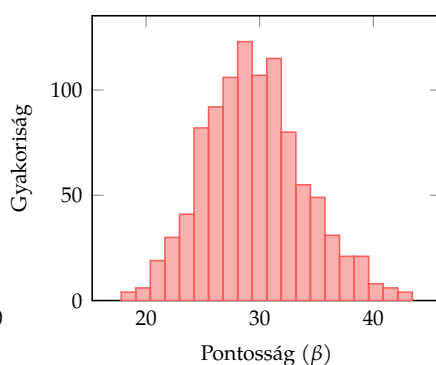
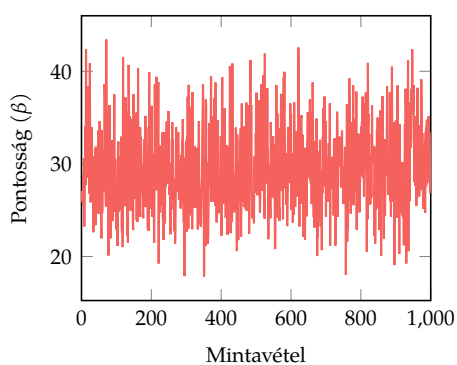
<sup>14</sup> Kézenfekvő példa a heti halálozások és a járványhullám esete; a járvány berobbanása előtt más folyamat generálja a halálozási statisztikákat, mint utána.



7.5. ábra. Gibbs-mintavétel bayesi lineáris regresszióra. Bal oldalon a valódi függvényt, a középen a prediktív eloszlást, a jobb oldalon az utóbbiból vett mintákat látjuk.



7.6. ábra. A tanult poszterior a  $\mathbf{w}$  és  $\beta$  változókra három dimenziós lineáris regressziós feladatban. A bal oldalon a gyűjtött mintákat látjuk (ún. *trace plot*), a jobb oldalon pedig a poszterior eloszlást közelítő histogramokat. Figyeljük meg, hogy a  $\mathbf{w}$  súlyok normális eloszlást sejtetnek, míg a  $\beta$  pozitív irányba elnyúlik, ahogy az a gamma-eloszlástól várjuk.





Adott időintervallumban bekövetkező események számának modellezésére rendszerint Poisson-eloszlást használunk; tegyük fel, hogy a  $k_i$ -vel jelölt esetszámok is ilyen eloszlást követnek. Feltesszük azt is, hogy az esetszámokat valójában két folyamat generálja:

- Az egyik a változás beállása előtt (tehát az esetszámok eleinte Poisson-eloszlást követően ismeretlen  $\lambda_1$  paraméterrel),
- A másik a változás beállása után (az esetszámok ugyanúgy Poisson-eloszlásúak, de már egy másik,  $\lambda_2$  paraméterrel).

Jelölje  $N$  az összes időpont számát,  $n$  pedig azt a pillanatot, ahol a folyamat átvált egyikből a másikba. A feladatunk, hogy  $\lambda_1$ ,  $\lambda_2$  és  $n$  eloszlását megbecsüljük a megfigyelt  $k_i$ -k mellett. Továbbra is bayesi következtetésnél maradva priorokat teszünk a paraméterekre<sup>15</sup> és összeállítjuk a valószínűségi modellt:

<sup>15</sup> Ügyelve a kiszámíthatóságra, azaz a konjugált priorok használatára.

$$\begin{aligned} p(n | \pi) &= \text{Cat}(n | \pi) = \prod_j \pi_j^{\text{one\_hot}(n)_j}, \\ p(\lambda_1 | \alpha_1, \beta_1) &= \mathcal{Ga}(\lambda_1 | \alpha_1, \beta_1) = \frac{\beta_1^{\alpha_1}}{\Gamma(\alpha_1)} \lambda_1^{\alpha_1-1} e^{-\beta_1 \lambda_1}, \\ p(\lambda_2 | \alpha_2, \beta_2) &= \mathcal{Ga}(\lambda_2 | \alpha_2, \beta_2) = \frac{\beta_2^{\alpha_2}}{\Gamma(\alpha_2)} \lambda_2^{\alpha_2-1} e^{-\beta_2 \lambda_2}, \\ p(\mathbf{k} | \lambda_1, \lambda_2, n) &= \prod_i p(k_i | \lambda_1, \lambda_2, n). \end{aligned}$$

Az  $n$  változó tehát kategorikus eloszlást követ  $\pi$  paraméterrel, aminek  $j$ . eleme annak az *a priori* valószínűségét írja le, hogy a  $j$ . időpontban áll be a változás. A  $\lambda_1$  és  $\lambda_2$  paraméterek *a priori* eloszlását gammának választjuk, mivel ez konjugált a Poisson-eloszlásra nézve. Az  $i$ . megfigyelt esetszámra pedig

$$p(k_i | \lambda_1, \lambda_2, n) = \begin{cases} \text{Poisson}(k_i | \lambda_1) = \frac{\lambda_1^{k_i}}{k_i!} e^{-\lambda_1}, & \text{ha } i \leq n, \\ \text{Poisson}(k_i | \lambda_2) = \frac{\lambda_2^{k_i}}{k_i!} e^{-\lambda_2}, & \text{ha } i > n. \end{cases}$$

A feltételes eloszlások felírásához az előző szakasszal teljesen analóg módon először az együttes eloszlást írjuk fel:

$$\begin{aligned} p(n, \lambda_1, \lambda_2, \mathbf{k} | \alpha_1, \beta_1, \alpha_2, \beta_2, \pi) \\ &= p(n | \pi) p(\lambda_1 | \alpha_1, \beta_1) p(\lambda_2 | \alpha_2, \beta_2) p(\mathbf{k} | \lambda_1, \lambda_2, n) \\ &= \prod_j \pi_j^{\text{one\_hot}(n)_j} \cdot \frac{\beta_1^{\alpha_1}}{\Gamma(\alpha_1)} \lambda_1^{\alpha_1-1} e^{-\beta_1 \lambda_1} \cdot \frac{\beta_2^{\alpha_2}}{\Gamma(\alpha_2)} \lambda_2^{\alpha_2-1} e^{-\beta_2 \lambda_2} \cdot \prod_{i=1}^n \frac{\lambda_1^{k_i}}{k_i!} e^{-\lambda_1} \cdot \prod_{i=n+1}^N \frac{\lambda_2^{k_i}}{k_i!} e^{-\lambda_2}, \end{aligned}$$

ahonnan a  $\lambda_1$  és  $\lambda_2$  változókat kigyűjtve

$$p(\lambda_1 | n, \mathbf{k}, \alpha_1, \beta_1) \propto \lambda_1^{\alpha_1-1+\sum_{i=1}^n k_i} e^{-(\beta_1+n)\lambda_1},$$

$$p(\lambda_2 | n, \mathbf{k}, \alpha_2, \beta_2) \propto \lambda_2^{\alpha_2-1+\sum_{i=n+1}^N k_i} e^{-(\beta_2+N-n)\lambda_2}.$$

A Gibbs-mintavételhez szükséges feltételes eloszlások tehát ismét gammának adódnak:

$$p(\lambda_1 | n, \mathbf{k}, \alpha_1, \beta_1) = \mathcal{Ga}\left(\lambda_1 \left| \alpha_1 + \sum_{i=1}^n k_i, \beta_1 + n \right.\right),$$

$$p(\lambda_2 | n, \mathbf{k}, \alpha_2, \beta_2) = \mathcal{Ga}\left(\lambda_2 \left| \alpha_2 + \sum_{i=n+1}^N k_i, \beta_2 + N - n \right.\right).$$

Az  $n$ -re vonatkozó feltételes eloszlás bonyolultabb, de a valószínűségeket szerencsére ki tudjuk számolni  $n$  minden lehetséges értékére. A megfelelő tagok kigyűjtésével számoljuk ki például  $n = 5$ -re:

$$p(n = 5 | \lambda_1, \lambda_2, \mathbf{k}, \pi) = \pi_5 \cdot \prod_{i=1}^5 \frac{\lambda_1^{k_i}}{k_i!} e^{-\lambda_1} \cdot \prod_{i=6}^N \frac{\lambda_2^{k_i}}{k_i!} e^{-\lambda_2},$$

és ezt  $n$  minden értékére megtehetjük. Az eljárást a 7.3. algoritmus szemlélteti, az eredmények a 7.7. ábrán láthatók.

```

function changepoint_Gibbs(k; iters=1000)
     $\lambda_1, \lambda_2, n$  = zeros(iters), zeros(iters), zeros(Int, iters)
    E, V, N = mean(k), std(k), length(k)
     $\alpha, \beta$  = E^2/V, E/V

     $\lambda_1[1]$  =  $\alpha/\beta$ 
     $\lambda_2[1]$  =  $\alpha/\beta$ 
    n[1] = N+2
    sk = cumsum(k)

    for m in 2:iters
        sn1 = sk[n[m-1]]
        sn2 = sk[end] - sn1
         $\lambda_1[m]$  = rand(Gamma( $\alpha + sn1$ , 1/( $\beta + n[m-1]$ )))
         $\lambda_2[m]$  = rand(Gamma( $\alpha + sn2$ , 1/( $\beta + N - n[m-1]$ )))

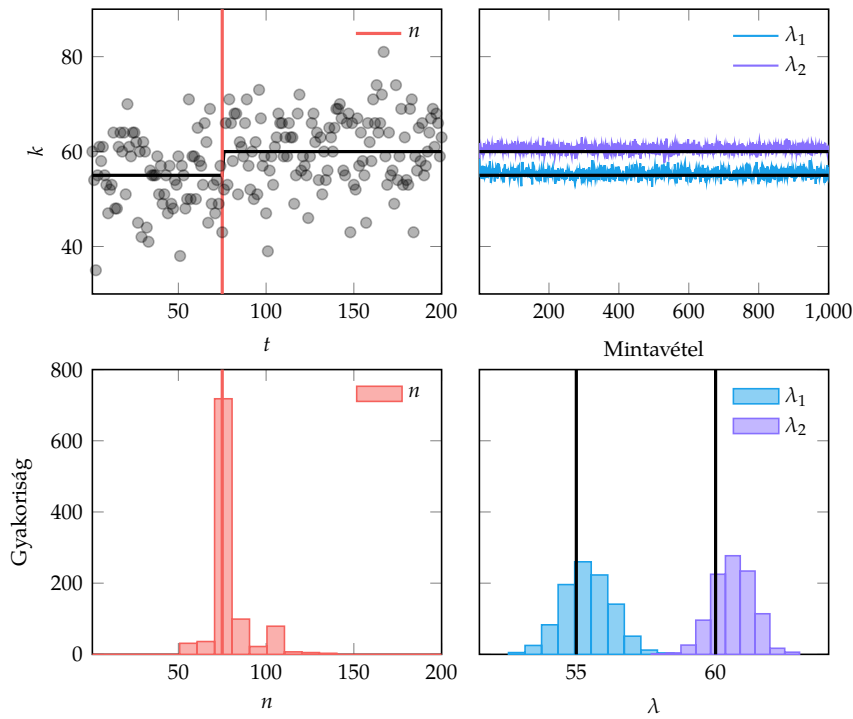
        frw = cumsum(logpdf.(Poisson.( $\lambda_1[m]$ ), k))
        rev = reverse(cumsum(logpdf.(Poisson.( $\lambda_2[m]$ ), reverse(k))))
        lp = frw .+ rev

        n[m] = wsample(1:N, exp.(lp .- maximum(lp)))
    end

    return  $\lambda_1, \lambda_2, n$ 
end

```

7.3. algoritmus. Következtetés a fenti changepoint-modellben Gibbs-mintavétellel. A  $k_i$  eset-számok összegét az  $n$ . időpontig hatékonyan számolhatjuk, ha a  $k$  tömbre először kumulatív szum-mát számolunk (lásd tetszőleges Algoritmusok és adatszerkezetek tárgy); hasonló trükkal élhetünk a  $n$  frissítésénél is, ha a Poisson-eloszlás sűrűségfüggvényének logaritmusát használjuk a direkt számolás helyett. Ez egyben a numerikus stabilitást is szolgálja.



7.7. ábra. Következtetés a fenti changepoint-modellben Gibbs-mintavétellel. Az első ábra az eset-számokat, a változás pillanatát és a két folyamat  $\lambda$  paraméterét szemlélteti. A második ábra a gyűjtött  $\lambda_1$  és  $\lambda_2$  minták trace plotja; az alsó két ábra pedig a közelítő eloszlásokat szemlélteti.

## 8. fejezet

# Szupportvektor-gépek

Ebben a fejezetben jórészt elhagyjuk a parametrikus modelleket és a valószínűségi értelmezést: ahelyett, hogy ismert alakú eloszlások ismeretlen paramétereire számolnánk ki poszterior eloszlásokat, közvetlenül a leképezést tanuljuk, miközben nem teszünk feltevéseket annak alakjára<sup>1</sup>. Bár így a bizonytalanság modellezésétől el kell tekintenünk, a szupportvektor-gépek (SVM) több előnnyel is bírnak: elméleti szempontból statisztikai garanciát kapunk a prediktív teljesítményre, gyakorlati szempontból pedig könnyen megoldható konvex optimalizálási problémákra vezetnek.

<sup>1</sup> Az ilyen modelleket kissé félrevezető módon *nemparametrikus* modelleknek szokták nevezni, bár sokszor mégiscsak véges sok paraméter optimalizálására lyukadunk ki.

### 8.1 Maximális margójú osztályozók

A maximális margójú osztályozók bevezetésének mély statisztikai tanuláselméleti motivációja van, amibe ebben a jegyzetben nem megyünk bele, és a statisztikai garanciákat sem érintjük<sup>2</sup>. Ehelyett nézzünk meg egy heurisztikus megközelítést.

<sup>2</sup> V.N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.

#### 8.1.1 Veszteségfüggvény

Emlékezzünk vissza a lineáris osztályozási modellünkre (3. fejezet), ahol a poszterior negatív logaritmusát a

$$L(\mathbf{w}) = -\ln p(\mathbf{w} | \mathbf{X}, \mathbf{y}, \alpha) = -\sum_i y_i \ln \sigma_i + (1 - y_i) \ln(1 - \sigma_i) + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}$$

veszteségfüggvényt adta.

Vizsgáljuk meg a fenti kifejezést abban az esetben, amikor  $y_i = 1$ . Ekkor a keresztentrópiában csak a  $\ln \sigma_i$  tag fog „túlélni”, amelyre

$$\ln \sigma_i = \ln \left( \frac{1}{1 + e^{-\mathbf{w}^\top \phi(\mathbf{x}_i)}} \right) = -\ln \left( 1 + e^{-\mathbf{w}^\top \phi(\mathbf{x}_i)} \right).$$

Hasonlóképpen az  $y_i = -1$  esetben a megmaradó tagban

$$\ln (1 - \sigma_i) = -\ln \left( 1 + e^{\mathbf{w}^\top \phi(\mathbf{x}_i)} \right)$$

értéket vesz fel. Vegyük észre, hogy a hatványkitevők csupán egy előjelben különböznek; ezeket a kifejezéseket könnyen egyesíthetjük, ha egy ravasz húzással újradefiniáljuk a címkéket, mégpedig  $y_i \in \{-1, +1\}$  formában<sup>3</sup>. Az új jelöléssel

$$L(\mathbf{w}) = \sum_i \ln \left( 1 + e^{-y_i \mathbf{w}^\top \phi(\mathbf{x}_i)} \right) + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}.$$

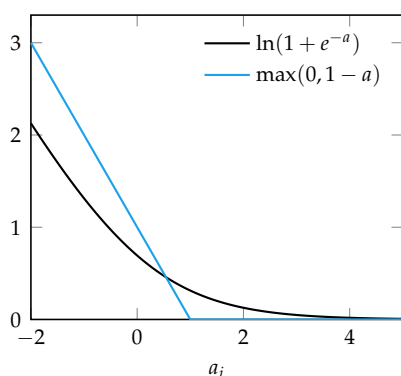
A hatványkitevőben szereplő kifejezést nevezzük el:

$$a_i := y_i \mathbf{w}^\top \phi(\mathbf{x}_i).$$

A keresztentrópia így  $\ln(1 + e^{-a_i})$  alakban írható; ezzel az átírással éppen úgy tudnánk osztályozást végezni, mint a korábbi fejezetekben. Most azonban egy még ravaszabb húzással a keresztentrópiát lecseréljük valami nagyon hasonlóra<sup>4</sup>:

$$\ln(1 + e^{-a_i}) \rightsquigarrow \max(0, 1 - a_i).$$

ahol az új függvényt *hinge loss* néven ismeri a szakirodalom.



<sup>3</sup> Azaz ha eddig  $y_i = 0$  jelentette a „negatív” osztályt, mostantól ezt  $y_i = -1$ -gyel fogjuk jelölni, míg  $y_i = 1$  változatlanul a pozitív osztályba tartozást jelenti.

<sup>4</sup> Vegyük észre az előbbiben a softplus, utóbbiban a ReLU függvényt!

8.1. ábra. Keresztentropia és hinge loss viszonya. A két függvény nagyon hasonlóan viselkedik: zérus alatt mindketten lineárisak, míg felette a 0-ba tartanak.

Az optimális  $\mathbf{w}$  vektor megtalálásához immár a

$$\min_{\mathbf{w}} \sum_i \max(0, 1 - y_i \mathbf{w}^\top \phi(\mathbf{x}_i)) + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}$$

optimalizációs feladatot kell megoldanunk. Mivel a  $\max(0, 1 - y_i \mathbf{w}^\top \phi(\mathbf{x}_i))$  kifejezés akkor optimális (zérus), ha  $1 - y_i \mathbf{w}^\top \phi(\mathbf{x}_i) \leq 0$ , így is felírhatjuk a feladatot<sup>5</sup>:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \mathbf{w}^\top \mathbf{w} \\ \text{s.t.} \quad & y_i \mathbf{w}^\top \phi(\mathbf{x}_i) \geq 1. \end{aligned}$$

Hagyományosan a  $\mathbf{w}$  súlyvektor mellett a  $b$  eltolást is szoktuk szerepeltetni<sup>6</sup>:

$$\min_{\mathbf{w}, b} \quad \frac{1}{2} \mathbf{w}^\top \mathbf{w} \quad (8.1)$$

$$\text{s.t.} \quad y_i (\mathbf{w}^\top \phi(\mathbf{x}_i) + b) \geq 1. \quad (8.2)$$

Vizsgáljuk meg, hogy mit is jelent ez a kényszerfeltétel. Emlékezzünk vissza, hogy ha egy  $\mathbf{w}^\top \phi + b$  alakú egyenletbe  $\phi$ -t helyettesítve zérust kapunk, az éppen azt jelenti, hogy a  $\phi$  pont rajta van a  $\mathbf{w}$  normálvektorú,  $b$ -vel eltolt hipersíkon. Ha az eredmény pozitív, a  $\phi$  pont a hipersík „fölött”, negatív érték esetén „alatta” helyezkedik el<sup>7</sup>. Ennek megfelelően a kényszer azt követeli meg, hogy az  $y_i = 1$  címkéjű minták  $\phi(\mathbf{x}_i)$  reprezentációi a hipersík „fölött”, az  $y_i = -1$  címkéjű mintákéi pedig „alatta” helyezkedjenek el; az elválasztásnak ráadásul egy bizonyos „biztonsági tartalék”, *margó* mellett kell teljesülnie.

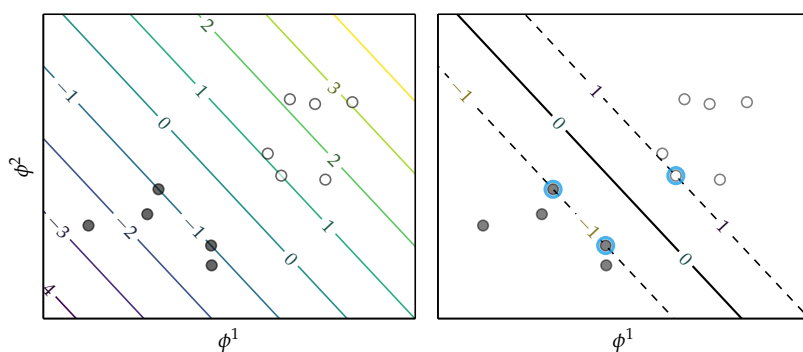
<sup>5</sup> Az  $\alpha$  együtthatóról elfeledkezünk, mivel ez csupán a célfüggvényt skálázza; az  $\frac{1}{2}$  együtthatót a későbbi számítások egyszerűsítése érdekében megtartjuk.

<sup>6</sup> Emlékezzünk vissza, hogy a 2. fejezetben eredetileg azért vezettük be a  $\phi$  függvényt, hogy  $b$ -vel ne kelljen külön foglalkoznunk. Később azonban rájöttünk, hogy  $\phi$  másra is jó, például egy lineáris osztályozót nemlineárisra tehetünk vele. Az SVM-típusú modelleknél a  $b$ -t szokás külön szerepeltetni a felírásban; ettől még  $\phi$  ugyanúgy jó lesz a nemlinearizálásra!

<sup>7</sup> Gondoljuk meg, hogy  $y_i = -1$  esetén a (8.2) kényszer a

$$\mathbf{w}^\top \phi(\mathbf{x}_i) + b \leq -1$$

alakot ölti.



8.2. ábra. SVM-nél olyan elválasztó hipersíkot keresünk, amely nem csak, hogy elválasztja a pozitív és a negatív osztályt, hanem ezt egy margó mellett teszi. A margón elhelyezkedő mintákat szupportvektoroknak nevezzük; bizonyos értelemben ezek „támasztják” a hipersíkot.

Pontosabban fogalmazva margó alatt a legközelebbi minták hipersíktól való távolságát értjük. Tudjuk, hogy egy pont és a hipersík előjeles távolságát a

$$\left( \frac{\mathbf{w}}{\|\mathbf{w}\|} \right)^\top \phi(\mathbf{x}_i) + \frac{b}{\|\mathbf{w}\|}$$

formula adja meg. A (8.2) kényszerfeltételt  $\|\mathbf{w}\|$ -vel leosztva a bal oldalon éppen egy minta hipersíktól való távolságát kapjuk:

$$\underbrace{\left( \frac{y_i \mathbf{w}^\top \phi(\mathbf{x}_i) + b}{\|\mathbf{w}\|} \right)}_{\text{távolság}} \geq \frac{1}{\|\mathbf{w}\|} \rightsquigarrow \max.$$

Másrésről viszont a (8.1) célfüggvényben  $\|\mathbf{w}\|^2$ -et minimalizáljuk, azaz ekvivalensen  $\frac{1}{\|\mathbf{w}\|}$ -t maximalizáljuk. A feladatra tehát úgy is gondolhatunk, hogy a két osztályt a legnagyobb margóval elválasztó hipersíkot keressük<sup>8</sup>.

<sup>8</sup> Az ilyen osztályozókat *maximum margin classifier*-nek is nevezik.

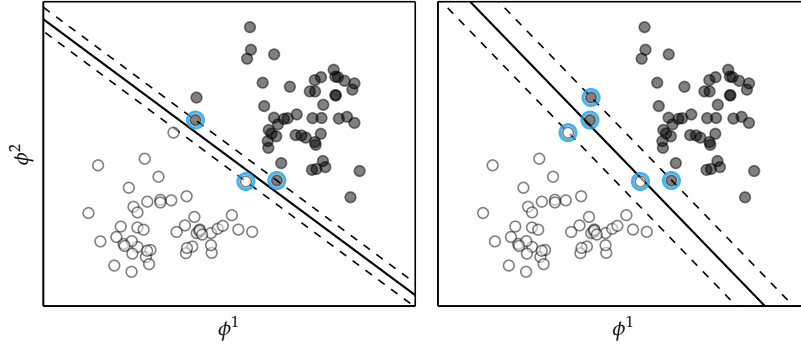
### 8.1.2 Soft-margin felírás

Könnyen előfordulhat, hogy a mintahalmazunk lineárisan nem szeparálható, vagy valamilyen más okból szeretnénk gyengíteni a kényszerfeltételt. Vezessük be a  $\xi_i$  ún. *slack* változókat:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \mathbf{w}^\top \mathbf{w} + C \sum_i \xi_i \\ \text{s.t.} \quad & y_i \left( \mathbf{w}^\top \phi(\mathbf{x}_i) + b \right) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned}$$

Az új felírás szerint nem akarjuk a margón kívülre esést minden egyes mintára kikényszeríteni; bizonyos mintákra megengedjük, hogy a margón belülre kerüljenek, sőt, akár a hipersík „rossz” oldalára is átlógnak. A margó áthágásának mértékét a mintához tartozó  $\xi_i$  érték jellemzi. Mindazonáltal a célfüggvényben azt is megfogalmazzuk, hogy a  $\xi_i$  változók alacsonyak maradjanak. Engedékenységünket a  $C$  hiperparaméter jellemzi, amelynek szerepéről a 8.3. ábra tájékoztat. Fontos megjegyezni, hogy túl magas  $C$  érték választásával – amikor egyáltalán nem vagyunk engedékenyek – könnyen túlilleszthetjük a modellt, míg túl alacsony  $C$  az ellenkező irányba, az alulilleszkedés felé vihet.





8.3. ábra. Illesztett hipersíkok különböző  $C$  értékek mellett. A bal oldali ábrán  $C$ -nek magas értéket választottunk; a slack változók minimalizálása nagy súlyt kap a célfüggvényben, így minden minta a „megfelelő” oldalra kerül (vagyis, így az esetleges zajra is könnyen rátanulhat a modell). A jobb oldalon  $C$  értéke kicsi, így egyes minták a margón belülre kerülnek, viszont a zaj kevésbé befolyásolja az illesztést.

## 8.2 Optimalizáció a Lagrange-módszerrel

Feltételes optimalizálási feladatról lévén szó, ismét a Lagrange-módszerhez folyamodunk<sup>9</sup>. A kényszerfeltételeket a megfelelő Lagrange-multiplikátorokkal szorozva és a célfüggvényhez adva

$$L = \frac{1}{2} \mathbf{w}^\top \mathbf{w} - \sum_i \alpha_i y_i (\mathbf{w}^\top \phi(\mathbf{x}_i) + b) + \sum_i (C - \alpha_i - \beta_i) \xi_i + \sum_i \alpha_i. \quad (8.3)$$

Vegyük először a  $\mathbf{w}$  szerinti gradienst és állítsuk zérusra:

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_i \alpha_i y_i \phi(\mathbf{x}_i) = 0,$$

ahonnan

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i). \quad (8.4)$$

Hasonlóképpen

$$\frac{\partial L}{\partial b} = - \sum_i \alpha_i y_i = 0. \quad (8.5)$$

Végül

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0, \quad (8.6)$$

<sup>9</sup> A módszer részleteit illetően lásd a B függelék.

amit – mivel az  $\alpha_i, \beta_i$  Lagrange-multiplikátorok definíció szerint nemnegatívak – úgy is írhatunk, hogy

$$0 \leq \alpha_i \leq C.$$

A kapott kifejezéseket a (8.3) egyenletbe visszahelyettesítve<sup>10</sup> és ismét áttérve minimalizálásra megérkezünk a duál problémához:

<sup>10</sup> Próbáljuk meg ezt önállóan ellenőrizni!

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) - \sum_i \alpha_i \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C, \end{aligned}$$

amelyet vektoros formában is felírhatunk:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^\top \mathbf{Q} \alpha - \mathbf{1}^\top \alpha \\ \text{s.t.} \quad & \alpha^\top \mathbf{y} = 0, \\ & 0 \leq \alpha_i \leq C, \end{aligned}$$

ahol a  $\mathbf{Q}$  mátrixot a következőképpen definiáltuk<sup>11</sup>:

<sup>11</sup> A  $\mathbf{K}$  mátrixot illetően lásd a fejezet utolsó szakaszát.

$$\mathbf{Q}_{ij} := y_i y_j \underbrace{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)}_{:= \mathbf{K}_{ij}}.$$

A duál probléma tehát egy lineáris kényszerekkel bíró kvadratikusan problémára (LCQP) alakítható. Amennyiben  $\mathbf{Q}$  pozitív definit – márpedig ezt meg fogjuk követelni tőle – a célfüggvény konvex és a feladat könnyen megoldható<sup>12</sup>. Nehézséget jelenthet viszont, hogy a  $\mathbf{Q}$  mátrix négyzetesen nő a minták számával – az SVM tehát ezen formájában inkább kis-közepes mintaszámú adatoknál használható.

<sup>12</sup> Számos standard algoritmus létezik LCQP feladatok megoldására. Az egyik leggyakrabban használt algoritmus SVM esetén a *sequential minimal optimization* (SMO) nevet viseli, amelynek részleteit itt nem tárgyaljuk.

### 8.3 Szupportvektorok és predikció

Vizsgáljuk meg, hogy az optimumban milyen feltételeknek kell teljesülniük. A KKT-feltételek közül a *complementary slackness* szerint

$$\alpha_i \left( y_i \left( \mathbf{w}^\top \phi(\mathbf{x}_i) + b \right) - 1 + \xi_i \right) = 0, \quad (8.7)$$

$$\beta_i \xi_i = 0. \quad (8.8)$$

A duál megengedettségi feltétel alapján egy-egy  $\alpha_i$  változónál az alábbi három eset egyike kell, hogy fennálljon:

- $0 < \alpha_i < C$ . Ekkor így gondolkodhatunk<sup>13</sup>:

<sup>13</sup> Lásd a (8.6) egyenletet.

$$C - \alpha_i - \beta_i = 0 \Rightarrow \beta_i > 0.$$

Következésképpen a (8.8) egyenletet felhasználva

$$\xi_i = 0,$$

azaz mivel a  $\xi_i$  slack változó értéke zérus, a (8.7) egyenlet szerint a mintánk éppen a margón helyezkedik el:

$$y_i (\mathbf{w}^\top \phi(\mathbf{x}_i) + b) = 1.$$

Más szóval, az olyan minták, amelyekhez a megfelelő  $\alpha_i$  zérus és  $C$  közé esik az optimalizáció után, szupportvektorok lesznek.

- $\alpha_i = C$ . Ekkor az előbbi érvelés alapján  $\beta_i = 0$  adódik és

$$\xi_i > 0,$$

azaz ezek a minták a margón belül, vagy akár a hipersík rossz oldalán helyezkednek el. Őket is a szupportvektorok közé soroljuk.

- $\alpha_i = 0$ . Kizárásos alapon ezek a „jól viselkedő” minták – a hipersík megfelelő oldalán helyezkednek el.

Végül nézzük meg, hogy az optimális  $\alpha$  vektort hogyan használhatjuk új minták osztályozására<sup>14</sup>. A (8.4). egyenletben megállapítottuk, hogy

<sup>14</sup> Technikailag kellene fog a  $b$  érték is; ennek kiszámítását a 8.1. algoritmus mutatja.

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i),$$

ahonnan egy új  $\mathbf{x}$  mintára

$$\mathbf{w}^\top \phi(\mathbf{x}) + b = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}) + b.$$

A szumma kiszámításához csupán a szupportvektorokra van szükség<sup>15</sup>, hiszen minden más mintához tartozó  $\alpha_i$  zérus; ezt úgy is mondhatjuk, hogy az SVM-nél ritka megoldást találunk. Az eredmény előjele megadja, hogy az  $\mathbf{x}$  minta a hipersík melyik oldalára esik, azaz melyik osztályba kell sorolnunk.

<sup>15</sup> Ez igen kényelmes a tárkomplexitás és hatékonyság szempontjából – az optimalizáció után elég csupán a nem zérus  $\alpha$ -koz tartozó értékeket eltávolítani, ami jellemzően csak kis része az összes adatnak.

```

mutable struct SVMRBF
    α      # duál paraméterek
    x_sv   # szupportvektorok
    y_sv   # címkék
    γ      # RBF paraméter
end
function train!(m::SVMRBF, x, y; C=1)
    Q = y .* rbf(x,x,m.γ) .* y'
    n = length(y)
    α = nothing

    svm = Model(Ipopt.Optimizer)
    @variable(svm, 0 <= α[1:n] <= C)
    @constraint(svm, α'y == 0)
    @objective(svm, Min, dot(α,Q,α)/2-sum(α))

    optimize!(svm)
    sv = value.(α) .> 1e-6
    m.x_sv = x[sv,:]
    m.y_sv = y[sv]
    m.α = value.(α)[sv]
end
function predict(m::SVMRBF, x_test)
    K_sv_y = rbf(m.x_sv, m.x_sv, m.γ) .* m.y_sv'
    K_tsv_y = rbf(x_test, m.x_sv, m.γ) .* m.y_sv'
    b = sum(K_sv_y * m.α .- m.y_sv) / length(m.α)
    y_pred = K_tsv_y * m.α .- b
    return y_pred
end

```

8.1. algoritmus. Szupportvektor-gép tanítása egy optimalizáló programcsomag felhasználásával. A cél-függvény és a kényszerek specifikálása után elvégezzük az optimalizációt, majd csak a nem zérus értékű  $\alpha$ -kat és szupportvektorokat tartjuk meg. A hipersík eltolása, azaz a  $b$  érték a „pozitív” és „negatív” szupportvektorok origótól vett, hipersíkra merőleges távolságainak átlagából adódik. A műveleteket – a hatékony számítás érdekében – vektorizált módon valósítottuk meg.

## 8.4 Nemlineáris megoldás és a kernel trükk

A szupportvektor-gépek egyik leghasznosabb tulajdonságát eddig elhanyagoltuk. Vegyük észre, hogy a gép a mintákat a tanulás és a predikció idején is csupán  $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$  formájában „látja”<sup>16</sup>; ezeket az értékeket összegyűjthetjük egy  $\mathbf{K}$  mátrixba, amelyet *kernel mátrixnak* vagy csak egyszerűen *kernelnek* neveznek:

$$\mathbf{K} = \begin{bmatrix} \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_1) & \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_P) \\ \phi(\mathbf{x}_2)^\top \phi(\mathbf{x}_1) & \phi(\mathbf{x}_2)^\top \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_2)^\top \phi(\mathbf{x}_P) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_P)^\top \phi(\mathbf{x}_1) & \phi(\mathbf{x}_P)^\top \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_P)^\top \phi(\mathbf{x}_P) \end{bmatrix}.$$

A kernelre egyfajta hasonlósági mátrixként is gondolhatunk:  $(i, j)$ -edik eleme arról árulkodik, hogy az  $i$ . minta mennyire „hasonlít” a  $j$ . mintához. Ezt a hasonlóságot anélkül is megszabhatjuk, hogy a  $\phi$  bázisfüggvényeket definiálnánk. Vegyünk egy  $k$  hasonlóságfüggvényt, és használjuk inkább ezt a számításoknál<sup>17</sup>:

$$\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) \rightsquigarrow k(\mathbf{x}_i, \mathbf{x}_j)$$

Hagyományos választás a Gauss-féle RBF függvény:

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2},$$

amelynek vektorizált megvalósítását az alábbi algoritmus szemlélteti.

```
function rbf(X1,X2,gamma)
    X1^2, X2^2 = sum(X1.^2,dims=2), sum(X2.^2,dims=2)
    X1X2 = X1*X2'
    return exp.(-gamma .* (X1^2 .- 2X1X2 .+ X2^2))
end
```

A helyzet valójában ennél is trükkösebb. Bármilyen hasonlóságfüggvényt is választunk, megmutatható, hogy tartozik hozzá egy megfelelő  $\phi$  leképezés. Legyen például  $\mathbf{x}_i \in \mathbb{R}^2$ :

$$\mathbf{x}_i = \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix},$$

<sup>16</sup> Nem láttunk olyat, hogy  $\phi(\mathbf{x})$  önmagában álna!

<sup>17</sup> A hasonlóságfüggvénynek azért pár dolgot tudnia kell. A  $K$  mátrixnak például szimmetrikusnak és pozitív definitnek kell lennie. Ez a két tulajdonság automatikusan teljesülne, ha a  $\phi(\mathbf{x}_i)$  reprezentációkat használnánk, „saját” hasonlóságfüggvényénél viszont nekünk kell gondoskodni róla. Ezen túl nincsenek megkötések: a kernel függvény akár szakterület-specifikus tudást is hordozhat.

8.2. algoritmus. RBF kernel számolása.

a hasonlóságfüggvény pedig legyen a skaláris szorzat négyzete:

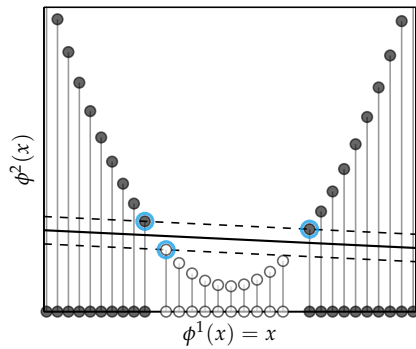
$$\begin{aligned}
 k(\mathbf{x}_i, \mathbf{x}_j) &:= (\mathbf{x}_i^\top \mathbf{x}_j)^2 \\
 &= (x_i^1 x_j^1 + x_i^2 x_j^2)^2 \\
 &= x_i^1 x_j^1 x_i^1 x_j^1 + x_i^1 x_j^1 x_i^2 x_j^2 + x_i^2 x_j^2 x_i^1 x_j^1 + x_i^2 x_j^2 x_i^2 x_j^2 \\
 &= \begin{bmatrix} x_i^1 x_i^1 \\ x_i^1 x_i^2 \\ x_i^2 x_i^1 \\ x_i^2 x_i^2 \end{bmatrix}^\top \begin{bmatrix} x_j^1 x_j^1 \\ x_j^1 x_j^2 \\ x_j^2 x_j^1 \\ x_j^2 x_j^2 \end{bmatrix}
 \end{aligned}$$

Arra jutottunk, hogy az eképpen megválasztott kernel maga után vonja a

$$\phi(\mathbf{x}_i) = \begin{bmatrix} x_i^1 x_i^1 \\ x_i^1 x_i^2 \\ x_i^2 x_i^1 \\ x_i^2 x_i^2 \end{bmatrix}$$

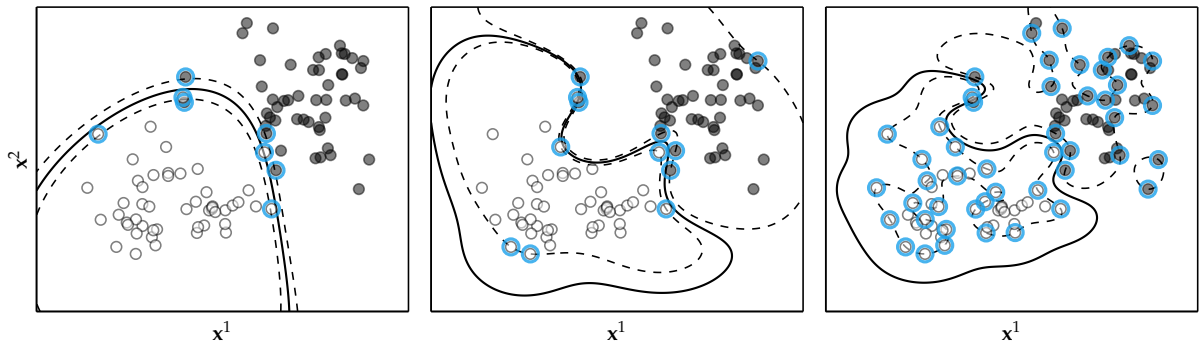
reprezentáció létezését is<sup>18</sup>. A korábbiól jól ismert gondolatra jutunk: a mintáinkat egy magasabb dimenziójú térbe transzformáljuk, és ott végezzük el a lineáris elválasztást. A magasabb dimenziójú reprezentációkat azonban itt nem kell explicit előállítanunk – különben a kód rengeteg időt töltene el nagy dimenziójú vektorok szorozgatásával – ehelyett használhatjuk a kernel függvényt, ami rendszerint könnyen számolható. Ezt az megközelítést *kernel trükk* névvel szokás illetni.

<sup>18</sup> Ez a konkrét reprezentáció olyan, mintha jellemző-párokat tartalmazna. Pixel-párok vizsgálata például nagyon jól jöhet képfeldolgozási, éldetekciós feladatokban.



8.4. ábra. Kernel trükk egy dimenzióban. A mintákat egy magasabb dimenziójú térbe transzformáljuk, ahol a lineáris elválasztás elvégezhető; az eredeti térben ez nemlineáris elválasztásnak felel meg.

A 8.5. ábra egy eképpen konstruált nemlineáris osztályozót mutat. A  $k$  megválsztásával – ekvivalensen  $\phi$  megválsztásával – az algoritmus nemlineáris viselkedését befolyásoljuk.



8.5. ábra. Nemlineáris SVM Gauss RBF függvénnel különböző  $\gamma$  értékek mellett. A  $\gamma$  érték növelésével a modell előbb-utóbb túlilleszkedik; megszorodnak a szupportvektorok, végül mindegyik minta szupportvektorrá válik.





## 9. fejezet

# Főkomponens-analízis

Ebben a fejezetben az egyik legalapvetőbb, leggyakrabban bevetett adatelemzési technikát vizsgáljuk meg. A főkomponens-analízis<sup>1</sup> a klaszterezéshez hasonlóan nemellenőrzött tanulási módszerek közé tartozik, és az adatok előfeldolgozására, rejtett összefüggéseinek feltárására használható.

<sup>1</sup> Angolul principal component analysis, PCA.

### 9.1 Dimenzió-redukció

Gyakran előfordul, hogy az adatunk egyes jellemzői redundánsak<sup>2</sup>, azaz valójában összefüggenek és ugyanarról a rejtett változóról hordoznak információt. Úgy is mondhatjuk, hogy az adatokat generáló „valóság” szabadsági fokainak száma kisebb, mint az adatok dimenzionalitása. Ilyen esetekben célszerű a rejtett dimenzionalitás felfedéséhez a meglévő jellemzőket összegyűjteni, és belőlük olyanokat konstruálni, amelyek egyrészt kevesebben vannak, másrészt egymáshoz képest is érdemi többletinformációt hordoznak<sup>3</sup>. A főkomponens-analízis során ezt a lehető legkézenfekvőbb módon, a jellemzők lineáris kombinációival oldjuk meg.

Tegyük fel, hogy  $d$ -dimenziós mintáinkat szeretnénk jóval kevesebb,  $m$  dimenzió felhasználásával ábrázolni:

$$\mathbf{x} \in \mathbb{R}^d \rightsquigarrow \hat{\mathbf{x}} \in \mathbb{R}^m, \quad m \ll d$$

Egy  $\mathbf{x}$  minta felírása  $\mathbb{R}^d$  egy adott bázisában

$$\mathbf{x} = \sum_{i=1}^d x^i \mathbf{b}_i,$$

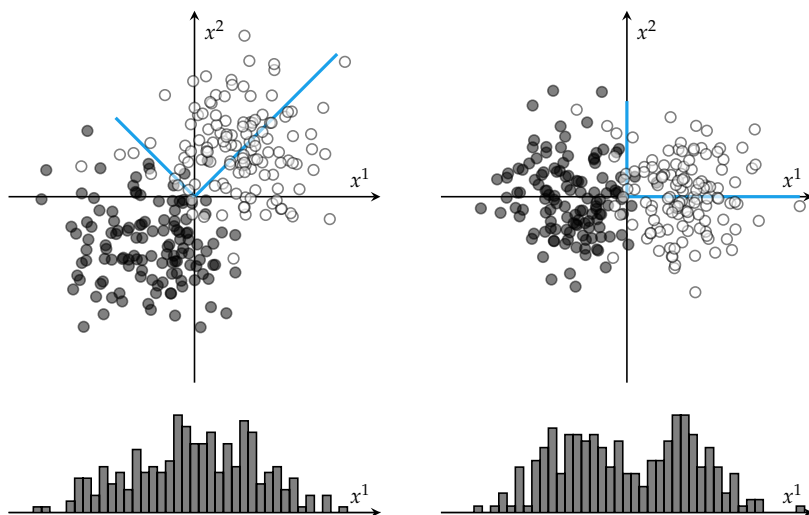
<sup>2</sup> Egy szélsőséges példát nézve tegyük fel, hogy egy táblázatban az egyik oszlop csupán annyiban különbözik egy másiktól, hogy annak kétszerese; nyilván nem várjuk, hogy bármelyikük többletinformációt hordozna a másikhoz képest.

<sup>3</sup> Úgy is mondhatjuk, hogy az információt tömöríteni szeretnénk; minél kevesebb dimenzió felhasználásával minél többet megőrizni.

ahol  $x^i$  az  $\mathbf{x}$  minta  $i$ . koordinátája (jellemzője),  $\mathbf{b}_i$  pedig az  $i$ . bázisvektor<sup>4</sup>. Az  $\hat{\mathbf{x}}$  közelítés felírása hasonlóképpen történik, azzal a különbséggel, hogy itt csak  $m$  darab bázisvektort használunk fel:

$$\hat{\mathbf{x}} = \sum_{i=1}^m x^i \mathbf{b}_i.$$

A célunk tehát, hogy találjunk  $m$  darab olyan bázisvektort, ami az adatokat valamilyen értelemben jól jellemzi. A 9.1 ábra egy ilyen bázis konstrukcióját mutatja be egy konkrét példán.



9.1. ábra. Új jellemzők konstruálása a régiek lineáris kombinációjaként. A kék vonalak az új bázisvektorokat jelölik; a jobb oldali ábra bázistranszformáció eredményeként jött létre. A fekete-fehér osztályok az algoritmus számára nem ismertek, csupán a szemléltetést szolgálják. Az új bázisban jól látszik, hogy az adatok két normális eloszlásból jönnek, és ehhez elég mindössze az  $x^1$  koordinátákat megvizsgálni; az  $x^2$  jellemzőket akár el is hagyhatnánk. Így egyetlen transzformációval láthatóvá tettünk egy olyan tulajdonságot, ami az eredeti bázisban a koordináták alapján még egyáltalán nem látszott.

Fontos, hogy az új bázisvektorok keresése során ortonormált bázisban gondolkodunk, azaz a skaláris szorzatokra

$$\mathbf{b}_i^\top \mathbf{b}_j = \begin{cases} 1 & \text{ha } i = j, \\ 0 & \text{különben,} \end{cases}$$

vagy másképpen

$$\mathbf{b}_i^\top \mathbf{b}_j = \delta_{ij}.$$

### 9.1.1 A főkomponensek meghatározása

Az új bázisvektorok meghatározásánál azt a célt tűzzük ki, hogy egy  $\mathbf{x}$  minta és ennek  $\hat{\mathbf{x}}$  közelítése során minél jobban megőrizzük a geometriai viszonyokat, azaz a kettő közötti különbség pitagoraszsi értelemben legyen minimális<sup>5</sup>. A hiba várható értéke

$$\begin{aligned}\mathbb{E} [\|\mathbf{x} - \hat{\mathbf{x}}\|^2] &= \mathbb{E} \left[ \left\| \sum_{i=1}^d x^i \mathbf{b}_i - \sum_{i=1}^m x^i \mathbf{b}_i \right\|^2 \right] = \mathbb{E} \left[ \left\| \sum_{i=m+1}^d x^i \mathbf{b}_i \right\|^2 \right] \\ &= \mathbb{E} \left[ \sum_{i=m+1}^d \sum_{j=m+1}^d x^i x^j \mathbf{b}_i^\top \mathbf{b}_j \right] = \mathbb{E} \left[ \sum_{i=m+1}^d (x^i)^2 \right] \\ &= \mathbb{E} \left[ \sum_{i=m+1}^d \mathbf{b}_i^\top \mathbf{x} \mathbf{x}^\top \mathbf{b}_i \right] = \sum_{i=m+1}^d \mathbf{b}_i^\top \mathbb{E} [\mathbf{x} \mathbf{x}^\top] \mathbf{b}_i \\ &= \sum_{i=m+1}^d \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i,\end{aligned}$$

ahol  $\mathbf{C}$  az adatok kovarianciamátrixa<sup>6</sup>. A hiba minimalizálása során a

$$\begin{aligned}\min_{\mathbf{b}} \quad & \sum_{i=m+1}^d \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i \\ \text{s.t.} \quad & \mathbf{b}_i^\top \mathbf{b}_i = 1\end{aligned}$$

problémát kell megoldanunk. Ekvivalens problémára jutunk, ha észrevesszük, hogy a kovarianciamátrix nyoma

$$\text{tr}(\mathbf{C}) = \sum_{i=1}^d \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i = \sum_{i=1}^m \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i + \sum_{i=m+1}^d \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i,$$

és mivel a nyom nem függ a bázistól, a jobb oldalon szereplő két kifejezés összege konstans. A feladatot emiatt az második tag minimalizálása helyett az első tag maximalizálásként is megfogalmazhatjuk<sup>7</sup>:

$$\begin{aligned}\max_{\mathbf{b}} \quad & \sum_{i=1}^m \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i \\ \text{s.t.} \quad & \mathbf{b}_i^\top \mathbf{b}_i = 1.\end{aligned}$$

<sup>5</sup> Ezt minimális hibájú rekonstrukciónak is hívják. A levezetésben felhasználjuk azt a trükköt is, hogy

$$\mathbf{x}^\top \mathbf{b}_j = \sum_{i=1}^d x^i \mathbf{b}_i^\top \mathbf{b}_j = x^j.$$

<sup>6</sup> Abban az esetben, amikor az adatok várható értéke zérus, azaz előzetesen normáltuk az adatokat az átlag levonásával.

<sup>7</sup> Ennek intuitív értelmezése az, hogy a minimális hibájú rekonstrukciót úgy is elérhetjük, ha olyan új tengelyeket keresünk, amelyek mentén az adatok varianciája a lehető legnagyobb (így őrizzük meg a legtöbb információt).

A feladat megoldásához ismét a Lagrange-módszerhez folyamodunk. A célfüggvényhez hozzávesszük a kényszerfeltételeket  $\lambda_i$  Lagrange-multiplikátorokkal:

$$L = \sum_{i=1}^m \left[ \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i - \lambda_i (\mathbf{b}_i^\top \mathbf{b}_i - 1) \right],$$

majd gradienst veszünk a  $\mathbf{b}_i$  bázisvektor szerint és zérusra állítjuk:

$$\nabla_{\mathbf{b}_i} L = 2\mathbf{C} \mathbf{b}_i - 2\lambda_i \mathbf{b}_i = 0.$$

Az egyenletet rendezve egy jól ismert eredményre jutunk: a

$$\mathbf{C} \mathbf{b}_i = \lambda_i \mathbf{b}_i \quad (9.1)$$

sajátérték-sajátvektor feladatot kell megoldanunk. A célfüggvénybe helyettesítve

$$\sum_{i=1}^m \mathbf{b}_i^\top \mathbf{C} \mathbf{b}_i = \sum_{i=1}^m \lambda_i \mathbf{b}_i^\top \mathbf{b}_i = \sum_{i=1}^m \lambda_i,$$

ennek maximalizálásához pedig úgy kell eljárunk, hogy az  $m$  darab keresett bázisvektor közé a legnagyobb sajátértékhez tartozó sajátvektorokat válogatjuk be.

### 9.1.2 Megoldás SVD-vel

Bár ilyen sajátérték-sajátvektor feladatokat papíron is megoldottunk, használjuk fel a modern numerikus könyvtárak előnyeit. A kovarianciamátrixra

$$\mathbb{E} [\mathbf{x} \mathbf{x}^\top] = \frac{1}{N} \sum_i \mathbf{x}_i \mathbf{x}_i^\top = \frac{1}{N} \mathbf{X}^\top \mathbf{X},$$

amely esetleg nagyra hízhat, és numerikus problémákba is ütközhetünk. Sokkal kényelmesebb szinguálrís értékek szerinti felbontással (SVD) próbálkoznunk<sup>8</sup>:

$$\mathbf{X} = \mathbf{U} \mathbf{D} \mathbf{V}^\top,$$

ahonnan

$$\mathbf{X}^\top \mathbf{X} = \mathbf{V} \mathbf{D} \mathbf{U}^\top \mathbf{U} \mathbf{D} \mathbf{V}^\top = \mathbf{V} \mathbf{D}^2 \mathbf{V}^\top,$$

azaz a sajátvektorok mátrixát így is megkaphatjuk.

<sup>8</sup> Még jobb, ha az SVD-nek a „gazdaságos” verzióját választjuk, ahol eleve csak a legnagyobb sajátértékekhez tartozó sajátvektorokat számoljuk ki. Erre ad lehetőséget a Lánczos-algoritmus.

## 9.2 Kernel PCA

Végül használjuk fel a szokásos trükkünket az algoritmus nemlinearizálására:

$$\begin{aligned} \mathbf{x}_i &\rightsquigarrow \phi(\mathbf{x}_i), \\ \mathbf{X}^\top \mathbf{X} &\rightsquigarrow \underbrace{\Phi^\top \Phi}_{\mathbf{K}} \end{aligned}$$

ahol  $\mathbf{K}$  a 8. fejezetben megismert kernel mátrix. A bázisvektorokat most speciális alakban, a minták reprezentációinak lineáris kombinációiként fogjuk keresni<sup>9</sup>:

$$\mathbf{b}_i := \sum_j \alpha_j \phi(\mathbf{x}_j) = \Phi^\top \boldsymbol{\alpha},$$

így a (9.1) egyenletet a következőképpen írhatjuk:

$$\frac{1}{N} \Phi^\top \Phi \Phi^\top \boldsymbol{\alpha} = \lambda \Phi^\top \boldsymbol{\alpha}.$$

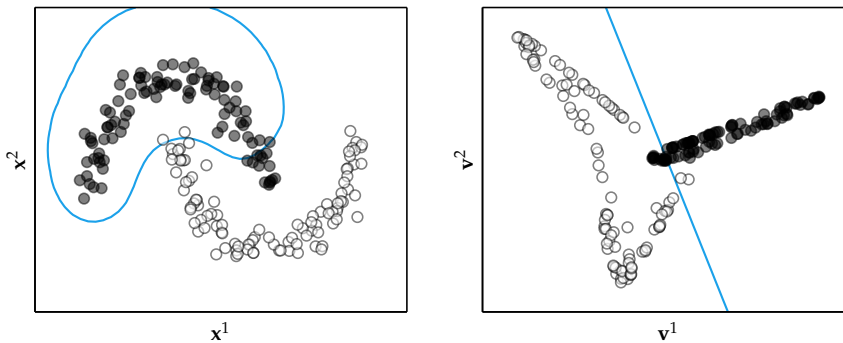
Vezessünk be mindkét oldalon egy  $\Phi$ -vel való szorzást. Innen

$$\frac{1}{N} \Phi^\top \Phi \Phi^\top \Phi \boldsymbol{\alpha} = \lambda \Phi^\top \Phi \boldsymbol{\alpha},$$

a kernel mátrix inverzével szorozva pedig

$$\mathbf{K} \boldsymbol{\alpha} = N \lambda \boldsymbol{\alpha}.$$

A nemlinearizáláshoz tehát annyit kell csupán változtatnunk, hogy kovarianciamátrix helyett a kernel mátrixot használjuk.



9.2. ábra. Kernel PCA és a hozzá tartozó bázistranszformáció. A bal oldalon az eredeti minták ábrázolódnak, jobb oldalon a transzformált minták, ahol a bázisvektorok a kernel mátrix két legnagyobb sajátértékéhez tartozó sajátvektorok. A transzformált térben egy lineáris elválasztás – itt egyszerű logisztikus regresszióval – az eredeti térben nemlineáris elválasztásnak felel meg.

<sup>9</sup> Egyáltalán nem világos, hogy miért lehet ezt megtenni, miért is lehetünk biztosak abban, hogy az optimális bázisvektorok a minták reprezentációi által kifeszített altérben helyezkednek el. A jó hír az, hogy erről tétel van; a még jobb, hogy ezt nem bizonyítjuk.



## 10. fejezet

# Megerősítéses tanulás

Ebben a fejezetben olyan gépi tanulási algoritmusokat vizsgálunk, amelyek nem csak tanulnak és következtetnek, hanem felfedeznek és aktívan beavatkoznak, hatnak a környezetükre. Áttekintjük a valószínűségszámítás egy érdekes újrafelfedezett alkalmazását, a többkarú rablókat, amire több modern megerősítéses tanulási algoritmus épül.

### 10.1 Többkarú rablók

Az egyik legegyszerűbb formalizmus a megerősítéses tanulás megragadására a többkarú rabló nevet viseli<sup>1</sup>. Gondolhatunk rá úgy, mint egy klassztikus játék-  
gépre – félkarú rablóra – csak éppen  $K$  darab karral. Szerencsés esetben egy-egy kart meghúzva jutalomban részesülhetünk; a karok egymástól függetlenül, ismeretlen eloszlás alapján osztogatják a jutalmat. Természetesen az a célunk, hogy a karok húzogatóásával minél több pénzt nyerjünk, ám az nem világos, hogy milyen stratégiát érdemes ehhez követni. Logikusnak tűnik, hogy

- Kiismerjük a karokat, azaz megbecsüljük a rejtett eloszlást a kapott jutalmak mögött,
- A megszerzett tudást felhasználva maximalizáljuk az összes jutalmat.

Érezzük, hogy a két cél ellentmond egymásnak: a gép kiismeréséhez rossz karokat is meg kell húznunk, ezzel azonban az összes nyereményünket csökkentjük. A karok valószínűségi természete miatt az is előfordulhat, hogy pechünk van, és

<sup>1</sup> Másképpen  $K$ -karú rabló, angolul *multi-armed bandit*.

egy egyébként jó kart meghúzva éppen nem nyerünk, így érdemes lehet a pillanatnyilag rossznak gondolt karokat is többször tesztelni. Ezt a dilemmát „felfedezés vagy kihasználás” (*exploration vs. exploitation*) néven ismeri a szakirodalom.

Vegyük a következő valószínűségi modellt. Az egyszerűség kedvéért tegyük fel, hogy a gép csak  $y \in \{0, 1\}$  jutalmat tud adni, mindegyik kar más-más valószínűséggel. A valószínűségeket jelölje

$$\theta_1, \theta_2, \dots, \theta_K,$$

azaz Bernoulli-eloszlással felírva<sup>2</sup>

$$p(y = 1 \mid \theta_k) = \theta_k,$$

$$p(y = 0 \mid \theta_k) = 1 - \theta_k.$$

<sup>2</sup> Választhatnánk más eloszlásokat is, például folytonos jutalmak esetén normális eloszlást, nemnegatív egész értékű jutalmak esetén Poisson-eloszlást, és így tovább.

### 10.1.1 Stratégiák a $K$ -karú probléma megoldására

**Mohó stratégia.** Az egyik legegyszerűbb stratégia, ha egy kicsi  $\varepsilon$  valószínűséggel a felfedezés mellett döntünk, azaz véletlenszerűen választunk kart; egyébként pedig mindig az aktuálisan legjobbnak gondolt – eddig átlagosan a legtöbb jutalmat adó – kart húzzuk meg.

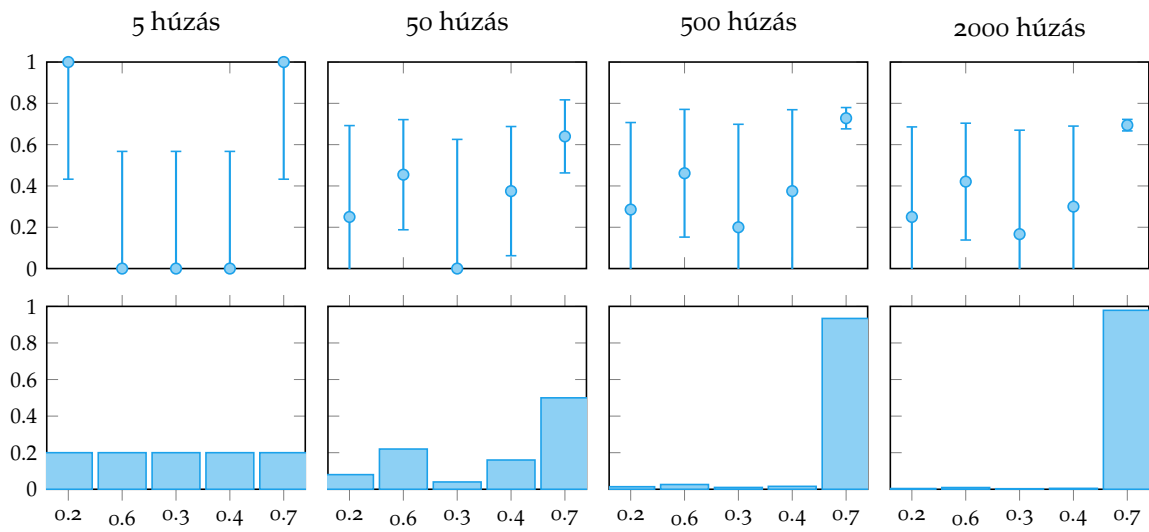
```
struct ε-greedy
    y::Vector{Int} # eddigi összjutalom mindegyik karnál
    n::Vector{Int} # húzások száma mindegyik karnál
    ε::Float64     # felfedezés valószínűsége
    K::Int         # karok száma
    ε-greedy(K, ε) = new(zeros(K), zeros(K), ε, K)
end
function choose(a::ε-greedy)
    return rand() < a.ε ? rand(1:a.K) : argmax(a.y ./ a.n)
end
function update(a::ε-greedy, arm, reward)
    a.y[arm] += reward
    a.n[arm] += 1
end
```

10.1. algoritmus.  $\varepsilon$ -mohó algoritmus. Kis valószínűséggel véletlenszerűen választunk kart, különben pedig az eddigi legjobbat húzzuk. A húzás után, a jutalom ismeretében frissítjük a meghúzott karhoz tartozó statisztikákat.



*Upper confidence bound.* Ennél a stratégiánál nem csak a karok jóságát modellezzük, hanem a bizonytalanságot is; a karválasztásnál a jutalom valószínűsége mellett megbecslünk egy konfidenciaintervallumot is<sup>3</sup>. A döntést mindig a konfidenciaintervallum felső határa alapján hozzuk meg. Ezt úgy is mondhatjuk, hogy a stratégia optimista, hiszen minden karról az elhíhető legjobbat feltételezzük. A 10.1. ábrán például 50 húzás után az 5. kart húznánk meg.

<sup>3</sup> Vagyis a legszélesebb intervallumot, aminél még hajlandók vagyunk elhinni, hogy a valódi, ismeretlen  $\theta_k$  valószínűség belesik.



Egyáltalán nem világos, hogy miért is jó ez a stratégia. Intuitíve arról van szó, hogy a  $k$ . kar meghúzása után, a megfigyelések felhalmozódásával a konfidenciaintervallum gyorsan keskenyedek, ráközelít a valódi  $\theta_k$ -ra. Így egy egyértelműen rossz kar hamar „lelepleződik”. A választásra vonatkozó formula

$$\frac{y_k}{n_k} + \sqrt{\frac{\alpha \ln N}{n_k}},$$

ahol  $y_k$  az  $k$ . karhoz tartozó korábbi jutalmak összege,  $n_k$  a  $k$ . kar húzásainak száma,  $N$  az összes karhúzás száma, az  $\alpha$  paraméter pedig a felfedezés és a kihasználás közötti egyensúlyt szabályozza. A stratégiáról egyébként megmutatható, hogy bizonyos értelemben optimális – erre és a formulára vonatkozó bizonyítás megtalálható az eredeti közleményben<sup>4</sup>.

10.1. ábra. Konfidenciaintervallumok alakulása UCB stratégia esetén  $K = 5$  karra. A felső sor az átlagos jutalmakat és a konfidenciaintervallumokat mutatja, az alsó sor pedig az adott kar húzásának relatív frekvenciáját. A valódi, ismeretlen valószínűségek az vízszintes tengelyen szerepelnek. Vegyük észre, hogy a végére a „legjobb”, 5. karhoz tartozó  $\theta_5$ -re egészen pontos becslést kapunk, míg a többire nem. Az algoritmus nem pazarolja a húzásait a rosszabb karok pontos felderítésére.

<sup>4</sup> P. Auer, N. Cesa-Bianchi és P. Fischer, “Finite-Time Analysis of the Multiarmed Bandit Problem”, *Machine Learning*, 47. évf., 235–256. old., 2002.

```

struct UCB
  y::Vector{Int} # eddigi összpontszám mindegyik karnál
  n::Vector{Int} # húzások száma mindegyik karnál
  α::Float64    # bátorsági paraméter
  K::Int        # karok száma
  UCB(K,α) = new(zeros(K),zeros(K),α,K)
end
function choose(a::UCB)
  N = sum(a.n)
  return N < a.K ? N+1 : argmax(@. a.y/a.n + sqrt(a.α*log(N)/a.n))
end
function update(a::UCB, arm, reward)
  a.y[arm] += reward
  a.n[arm] += 1
end

```

10.2. algoritmus. UCB algoritmus. Kezdetben minden kart meghúzzunk egyszer, majd a konfidencia-intervallum felső határa alapján döntünk. A frissítés megegyezik az előző algoritmusnál látottal.

*Thompson-mintavétel.* Végül vessünk be egy kis bayesi gondolkodást, ami egészen természetes módon kombinálja a véletlenszerű felfedezést és a tudás kihasználását. Tegyük minden karra egy-egy priort, ami jelen esetben Béta-eloszlású<sup>5</sup>. A  $k$ . karra

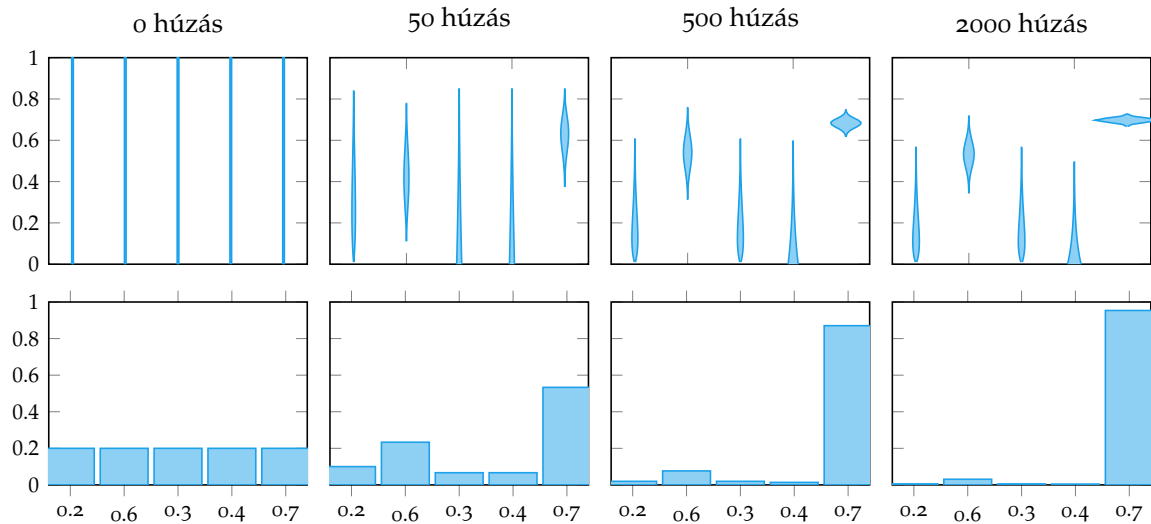
<sup>5</sup> Bernoulli-eloszláson ez konjugált; más eloszlásoknál más priort kell használnunk.

$$p(\theta_k | \alpha_k, \beta_k) = \text{Beta}(\theta_k | \alpha_k, \beta_k).$$

Ez a  $\theta_k$  értékére vonatkozó előzetes hiedelmeinket testesíti meg. Húzás után a megfigyelt jutalom segítségével frissítjük az eloszlást. A poszterior a már jól ismert Bayes-tételből következik:

$$p(\theta_k | y, \alpha_k, \beta_k) \propto p(y | \theta_k) p(\theta_k | \alpha_k, \beta_k) = \text{Beta}(\theta_k | \alpha_k + y, \beta_k + 1 - y),$$

azaz ha a  $k$ . karral nyertünk, akkor az  $\alpha_k$  paramétert, egyébként a  $\beta_k$  paramétert növeljük eggyel. A karválasztás úgy történik, hogy mindegyik eloszlásból veszünk egy-egy mintát, majd a legnagyobb adódóhoz tartozó kart húzzuk meg. A poszteriorok alakulását a 10.2. ábra mutatja, az eljárás pszeudokódját pedig a 10.3. algoritmus szemlélteti.



10.2. ábra. Thompson-mintavétel. A felső sorban a Béta-poszteriorok ábrázolódnak. A legjobb karokhoz tartozó  $\theta$ -kat pontosan jósoljuk, míg a rosszabbaknál jelentősen laposabb a poszterior.

```

struct Thompson
     $\alpha$ ::Vector{Float64}
     $\beta$ ::Vector{Float64}
    K::Int
    Thompson(K) = new(ones(K),ones(K),K)
end
function choose(a::Thompson)
    return argmax(rand.(Beta.(a. $\alpha$ ,a. $\beta$ )))
end
function update(a::Thompson,arm,reward)
    a. $\alpha$ [arm] += reward
    a. $\beta$ [arm] += 1-reward
end

```

10.3. algoritmus. Thompson-mintavétel. Az  $\alpha$  és  $\beta$  paraméterek a karokhoz tartozó Béta-eloszlásokat írják le. Úgy választunk kart, hogy mindegyik eloszlásból veszünk egy-egy mintát, majd a legnagyobb adódóhoz tartozó kart húzzuk meg. A paraméterek frissítését a Bayes-tétel diktálja.

## 10.1.2 Az ágensek minősítése

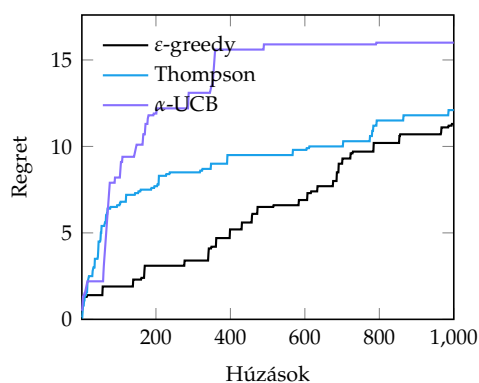
Az ágensek minősítéséhez vezessük be a *regret*<sup>6</sup> fogalmát. A

$$\Delta_k := \max_j \theta_j - \theta_k$$

érték azt írja le, hogy egyetlen húzásnál mennyi jutalomtól esünk el azzal, hogy nem a lehető legjobb húztuk meg; a regret ezt az összes karhúzásra vonatkoztatva adja meg<sup>7</sup>:

$$R := \sum_k \Delta_k \mathbb{E}[n_k].$$

A stratégiák nagyon különbözőképpen viselkednek a regret szempontjából. A 10.3. ábra egy jellegzetes szituációt mutat: az  $\epsilon$ -mohó stratégia regretje lineárisan nő a húzások számával, míg a másik két algoritmusé ellapuló, logaritmikus.



<sup>6</sup> Magyarul leginkább megbánásnak fordíthatnánk.

<sup>7</sup> Szemléletes jelentése: mennyire „bánjuk”, hogy nem végig a legjobb kart húzogattuk.

10.3. ábra. Regret alakulása a húzások során különböző stratégiák esetében.

Eddig is kerülgettük egy stratégia optimalitásának fogalmát. Megmutatható, hogy a legjobb elérhető regret logaritmikus, amit például az UCB algoritmus meg is tud valósítani. Valójában az UCB kidolgozása során olyan algoritmust kerestek, ami kellően egyszerű és tudja a logaritmikus regretet is. Ebből a szempontból a naiv mohó algoritmus tehát kifejezetten rosszul teljesít, bár ezt is lehet javítani adaptív  $\epsilon$  paraméter használatával.

## 10.2 Monte Carlo Tree Search

Az eddig tárgyalt módszerek kevésbé emlékeztettek a klasszikus megerősítéses tanulási algoritmusokra, ahol rendszerint ágensekről, állapotokról és cselekvésekről beszélünk<sup>8</sup>. A többkarú rablók azonban ilyen kontextusban is nagyon jól használhatók – egy adott állapotban a várhatóan legnagyobb jutalomhoz vezető cselekvés kiválasztása kitűnően leírható egy többkarú rabló-feladatként.

Vegyünk például egy játékot, amelynek lehetséges állapotait egy irányított gráf csúcsaiként ábrázoljuk, az élek pedig a lehetséges lépéseket jelentik. A gyökér a kezdőállapot, a gráfon pedig a játék szabályait követve, az élek mentén bolyonghatunk mindaddig, amíg el nem jutunk egy nyertes vagy vesztes állapotba.

A Monte Carlo Tree Search (MCTS) algoritmus<sup>9</sup> az UCB módszert használja az optimális cselekvés kiválasztására, a statisztikák frissítésére pedig egy sztochasztikus eljárást alkalmaz, amellyel az állapottér felrobbanása megkerülhető. Az UCB algoritmushoz nyilván kell tartani, hogy hányszor nyertünk egy adott állapotból, illetve hányszor jártunk már ott (a korábbi  $y_k$  és  $n_k$  értékek analógjai), amit az egyszerűség kedvéért mostantól pl.  $[3, 5]$  formában fogunk írni. Az algoritmus négy lépést iterál:

1. *Selection*. Az UCB algoritmussal a lehetséges követő állapotok közül kiválasztjuk a legnagyobb jutalommal kecsegtetőt, mégpedig rekurzív jelleggel, tehát mindaddig, amíg vannak a gráfban követő állapotok.
2. *Expansion*. A kiválasztott levél-állapotot kifejtjük, azaz a gyerekeit  $[0, 0]$  értékkel bevesszük a gráfba.
3. *Rollout*. A kiválasztott állapotból véletlenszerű lépésekkel végigjátsszuk a játékot; a folyamat végén vagy kapunk jutalmat, vagy nem.
4. *Backpropagation*. A játék végén kapott jutalmat visszaterjesztjük a kiinduló állapotig, frissítve az  $[y, n]$  értékeket.

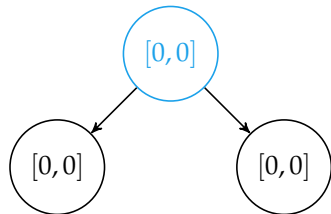
Nézzük mindezt egy példán. Kezdetben csak a gyökérállapotot tudjuk választani (selection):



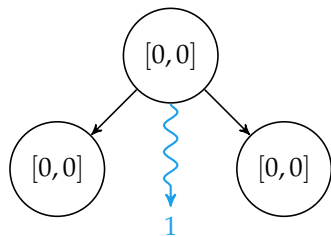
<sup>8</sup> Javasoljuk a Mesterséges intelligencia c. tárgyon tanultak felidézését, esetleg a Russell–Norvig könyv fellelőzését.

<sup>9</sup> L. Kocsis és C. Szepesvári, “Bandit Based Monte-Carlo Planning”, *Machine Learning: ECML 2006*, 2006.

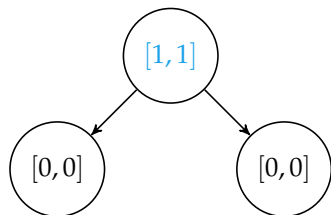
Kifejtjük az állapotot (expansion):



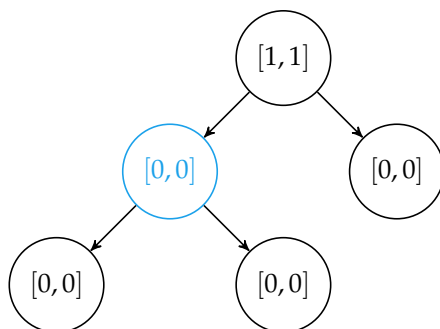
Végigjátsszuk a játékot (rollout); tegyük fel, hogy nyertünk:



A nyereménnyel frissítjük az állapotot (backpropagation):

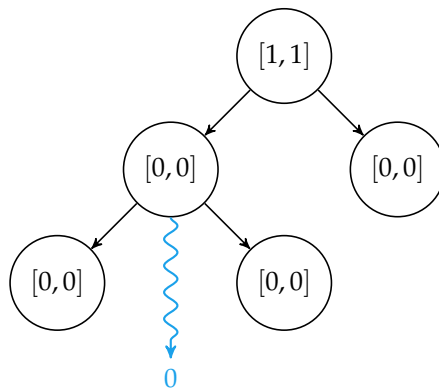


A következő iterációban kiválasztjuk a gyökeret, majd a lehetséges követő állapotok közül UCB-vel döntünk. Ebben a lépésben már levélbe jutottunk<sup>10</sup>, úgyhogy lépünk a kifejtésre:

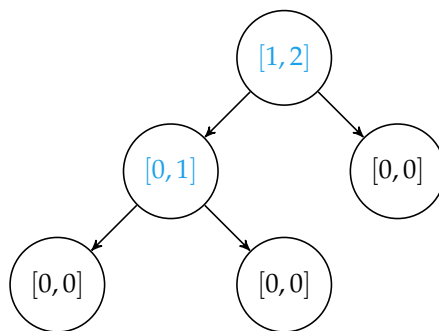


<sup>10</sup> Ha nem ez lenne a helyzet, addig ismételnénk a kiválasztást, amíg el nem jutunk egy levélbe, amit kifejtethetünk, vagy esetleg végállapotba.

Tegyük fel, hogy a rollout során nem nyertünk:



Visszafelé haladva frissítjük a statisztikákat.



Könnyen elképzelhető, hogy ahogy gyűlnek az értékek, az UCB algoritmus egyre jobb és jobb teljesítményt fog elérni a „jó” lépések kiválasztásában. Az algoritmus másik nagy előnye, hogy a gráf dinamikusan épül, azaz az egyértelműen „rossz” csúcsok nemigen kerülnek kifejtésre. Szintén hasznos tulajdonság, hogy a gráfot lehet „interaktív” módon is építgetni: kétszemélyes játéknál ellenfelünk lépése után nem a gyökérből futtatjuk újra az algoritmust, hanem abból az állapotból, amelyikben éppen vagyunk. Megtehetjük például, hogy ellenfelünk minden egyes lépése után futtatunk néhány száz iterációnyi MCTS-t, majd az összegyűjtött statisztikák alapján lépünk egyet.

### 10.3 AlphaZero

Végül nézzük meg az MCTS algoritmus egy modern, neurális hálózatokkal megtűzdelt változatát, ami szép sikereket ért el az emberi Go-játékosok legyőzésében<sup>11</sup>. A lépések kiválasztásához a következő egyenletet használjuk:

$$Q(s, a) + c \cdot P(s, a) \cdot \sqrt{\frac{\ln N(s)}{N(s, a) + 1}}. \quad (10.1)$$

Itt több tag is magyarázatra szorul. A  $Q(s, a)$  érték a korábbi  $\frac{y_k}{n_k}$  analógja, amely az  $s$  állapotban végrehajtott  $a$  cselekvés várható hasznosságáról tájékoztat. A  $c$  hiperparaméter a felfedezés mértékét („bátorságot”) szabályozza. A  $P(s, a)$  érték korábban nem volt. Ezt úgy választjuk, hogy tanulható legyen, lényegében ez teszi adaptívvá az algoritmust. Végül az  $N(s)$  és  $N(s, a)$  kifejezések a korábbi  $N$  és  $n_k$  értékekkel egyeznek meg – azaz hányszor jártunk az adott csúcsban, illetve hányszor választottuk ott az adott cselekvést – kicsit más jelölésmóddal.

Az AlphaZero két neurális hálózatot használ. A klasszikus MCTS algoritmussal szemben itt nem történik rollout, hanem a visszaterjesztendő jutalmat egy  $v(s)$  neurális hálózat (*value network*) tanulja. Hasonlóképpen  $P(s, a)$  értékeket egy  $\mathbf{p}(s)$  softmax kimenettel rendelkező neurális hálózat szolgáltatja, amely minden  $a$  lépéshez egy 0 és 1 közötti számot rendel (*policy network*).

A tanuláshoz az MCTS algoritmus végeredménye szolgáltatja a mintákat. A (10.1) választással „járatott” MCTS algoritmus ugyanis egyrészt előbb-utóbb eljut egy végállapotba, ahol a valódi jutalommal szembesül, másrészt a csúcsokban összegyűlő statisztikák rendszerint sokkal jobban tükrözik az optimális választásokat, mint a  $\mathbf{p}(s)$  értékek. A tanítóhalmaz tehát

$$\begin{aligned} &(s_1, \boldsymbol{\pi}_1, v^*), \\ &(s_2, \boldsymbol{\pi}_2, v^*), \\ &\vdots \\ &(s_n, \boldsymbol{\pi}_n, v^*) \end{aligned}$$

alakú, ahol  $s_1, s_2, \dots, s_n$  a bejárt állapotok,  $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \dots, \boldsymbol{\pi}_n$  az ezekben az állapotokban kialakuló választási preferenciák (minden  $a$ -hoz egy-egy valószínűség),  $v^*$  pedig az  $s_n$  végállapotban kapott valódi jutalom. A  $\mathbf{p}(s)$  hálózat tanításánál keresztentropia, a  $v(s)$  hálózaton pedig négyzetes hiba a veszteségfüggvény.

<sup>11</sup> D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez és tsai., “A GENERAL REINFORCEMENT LEARNING ALGORITHM THAT MASTERS CHESS, SHOGI, AND GO THROUGH SELF-PLAY”, *Science*, 362. évf., 6419. sz., 1140–1144. old., 2018.



## 11. fejezet

# Federált tanulás

Ebben a fejezetben megvizsgáljuk, hogyan lehet tanulni, ha tanítóadatok több kliensre vannak szétosztva. Kézenfekvő volna ezeket az adatokat egyesíteni, és a korábban megismert technikák bármelyikét bevetni, ám ezt nem tehetjük meg: a kliensek rendszerint ódzkodnak attól, hogy érzékeny információt „dobjanak be a közönsbe”<sup>1</sup>. *Federált tanulás* alatt olyan algoritmusokat értünk, amelyek továbbra is egy közös modell tanítását célozzák, ám az az adatok mindvégig a klienseknél, „biztonságban” maradnak. A federált tanulás a gépi tanulásnak egy viszonylag friss területe, ahol még nincs széles körben elfogadott, egységes matematikai keret a modellek leírására<sup>2</sup> – mi továbbra is a bayesi utat próbáljuk követni.

### 11.1 Valószínűségi modell

A korábbi fejezetekhez hasonlóan jelölje  $(\mathbf{X}, \mathbf{y})$  a tanítóadatokat,  $\theta$  pedig a modell-paramétereket<sup>3</sup>. Tegyük fel azt is, hogy az adatok  $K$  kliensre vannak szétosztva; a kliensek mindegyike valamiképpen hozzájárul a posztteriorhoz, és ezek a hozzájárulások függetlenek. A Bayes-tételt felhasználva a posztterior

$$p(\theta | \mathbf{X}, \mathbf{y}) = p(\theta) \cdot \prod_{k=1}^K \frac{1}{Z_k} p(\mathbf{y}_k | \mathbf{X}_k, \theta),$$

ahol ismét visszaköszön a  $p(\theta)$  prior, ám a likelihood-ot faktorizálva jelenítettük meg:  $(\mathbf{X}_k, \mathbf{y}_k)$  a  $k$ . klienshez tartozó „adatcsomag”<sup>4</sup>, a normalizációt pedig a  $Z_k$  együtthatókba rejtettük el.

<sup>1</sup> Klasszikus példa erre az okostelefonok autocorrect szolgáltatása, amelyet a felhasználók által bevitt szavakon tanítunk; abba valószínűleg kevesen egyeznének bele, hogy az üzenetei kikerüljenek a telefonjáról.

<sup>2</sup> E jegyzet írásának időpontjában legalábbis.

<sup>3</sup> Az általánosság érdekében ennél jobban nem is részletezzük a modellt. Ez a keret épp olyan jó lesz osztályozásra, regresszióra, vagy bármi másra.

<sup>4</sup> A  $k$  index tehát most több tanítómintát takar.

Ismét kétféleképpen járhatunk el:

1. Maximalizáljuk a poszteriort  $\theta$  szerint, azaz MAP megoldást keresünk, elvésszítve a bizonytalanságra vonatkozó információt,
2. Megőrizzük a modell valószínűségi természetét, és a korábban tanult technikák valamelyikével közelítjük a poszteriort.

## 11.2 MAP megoldás

Nézzük először a MAP megoldást. Negatív logaritmust véve

$$-\ln p(\theta | \mathbf{X}, \mathbf{y}) = -\ln p(\theta) + \sum_k \ln Z_k - \sum_k \ln p(\mathbf{y}_k | \mathbf{X}_k, \theta),$$

ahol csak a  $\theta$ -t tartalmazó tagokat tartjuk meg; az egyes kliensek ezután maguk is elvégezhetik az optimalizációt a saját adataik felhasználásával<sup>5</sup>.

Az egyetlen gond az, hogy a formulában  $\theta$  paraméterek mellett nem látunk  $k$  indexet; ez a  $\theta$  globális, míg az adatok lokálisak<sup>6</sup>. Ezt orvosolandó, bevezetjük a lokális  $\theta_k$  paramétereket (gyakorlatilag a kliensek „saját verziója”  $\theta$ -t illetően), a végső, globális paraméterek pedig a lokálisak valamilyen összesítésével, például egyszerű átlagolásával adódhatnak:

$$p(\mathbf{y}_k | \mathbf{X}_k, \theta) \leftarrow p(\mathbf{y}_k | \mathbf{X}_k, \theta_k),$$

$$\theta \leftarrow \frac{1}{K} \sum_{k=1}^K \theta_k.$$

Ezt a módszert *federated averaging* (*FedAvg*) néven említi a szakirodalom. A megvalósítás során szokás vegyíteni a kétféle lépést: néhány epoch-nyi lokális tanítást a paraméter-aggregálás követ<sup>7</sup>, majd ismét lokális tanítás következik, és így tovább.

A FedAvg módszer a gyakorlatban sokszor nem működik, ha a klienseknél lévő adatok nem függetlenek és azonos eloszlásúak (nem IID)<sup>8</sup>. Ilyen esetekben a lokális modellek túlságosan távol „sodródhatnak” egymástól, és az átlagolással a modellparaméterek jórészt értelmetlen értékeket vesznek fel (*client drift*).

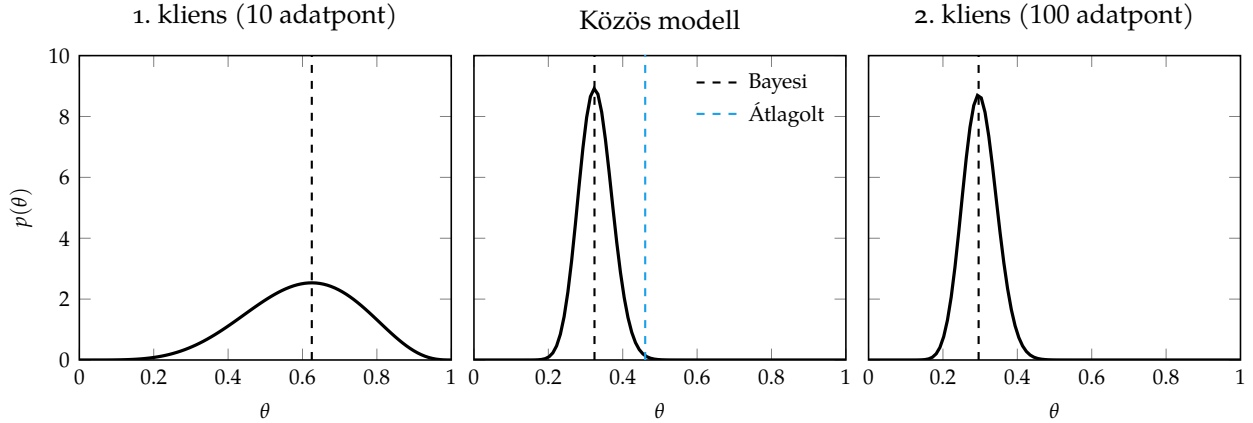
A jelenséget úgy is értelmezhetjük, hogy az átlagolós séma nem veszi figyelembe az egyes kliensek adatainak statisztikai tulajdonságait (pl. mintaszám); lényegében minden kliensre egyformán hagyatkozik. Ezzel szemben egy bayesi sémától azt várjuk, hogy a bizonytalanság explicit modellezésével ezeket az eltéréseket automatikusan kezeli. Erre mutat példát a 11.1. ábra.

<sup>5</sup> Úgy is lehetne fogalmazni, hogy most az algoritmust visszük az adatokhoz, és nem fordítva.

<sup>6</sup> Használhatunk egyfajta „adagatós” sémát is, ahol egyszerre egyetlen kliens dolgozik a  $\theta$  paramétereken; ha befejezte a tanítást, továbbadja a modellt a következőnek. Ez a módszer csak kevés kliensnél célravezető, ráadásul gyakran lép fel a *catastrophic forgetting* jelensége is, ahol az új információ hatására a modell „elfelejti” a korábban tanultakat.

<sup>7</sup> Neurális modelleknél szokás a paraméterek helyett a gradiensek átlagolása a tanítás során. A direkt paramétermegosztás ugyanis biztonsági kockázatot is jelenthet, hiszen ezekből – hacsak nem alkalmazunk kifinomult kriptográfiai eljárásokat – gyakran visszafejthetők a tanító adatok.

<sup>8</sup> Az okostelefonos példánál maradván, más-más emberek más-más stílusban, szóhasználatlaltal kommunikálnak.



### 11.3 Variációs közelítés

A bizonytalanság modellezéséhez célszerű a variációs módszercsaládhoz folyamodni. Vezessük be a  $q(\theta)$  közelítő poszteriórt, amelyet úgy választunk, hogy a valódi poszteriorhoz hasonlóan faktorizálódjon; más szóval, mindegyik kliens kap egy saját „darabkát” belőle<sup>9</sup>:

$$q(\theta) := q_0(\theta) \cdot \prod_{k=1}^K q_k(\theta),$$

ahol klienseket továbbra is  $k$  indexeli, valamint

$$q_0(\theta) := p(\theta).$$

A  $q_k(\theta)$  függvényre úgy is gondolhatunk, mint a  $k$ . kliens hozzájárulására a modellhez. Ha  $k$ . kliens – bármilyen okból – frissítette saját hozzájárulását, a közös  $q(\theta)$  poszterior frissítéséhez a következőképpen kell eljárni<sup>10</sup>:

$$q(\theta) \leftarrow \frac{q^{\text{régi}}(\theta)}{q_k^{\text{régi}}(\theta)} \cdot q_k(\theta)$$

Hogyan frissítheti a  $k$ . kliens a saját darabkáját? Kézenfekvő volna a valódi poszterior megfelelő darabkájához közelíteni:

$$q_k(\theta) \quad \text{vs.} \quad \frac{1}{Z_k} p(\mathbf{y}_k | \mathbf{X}_k, \theta).$$

11.1. ábra. Poszterior és paraméterátlag eltérése két kliensnél. A paraméterátlagolás félrevezető eredményt szolgáltat; a helyes bayesi séma szerint a közös modell sokkal inkább figyelembe veszi a kisebb bizonytalansággal (pl. nagyobb mintaszámmal) bíró klienst.

<sup>9</sup> Ezeknek a  $q_k(\theta)$  „darabkáknak” nem is kell normalizált sűrűségfüggvénynek lenniük; elég, ha a teljes  $q(\theta)$  az.

<sup>10</sup> Intuitíve, „kiosztjuk” a globális modellből a korábbi hozzájárulást, és a helyére „beszorozzuk” az újat.

Ennél ügyesebben is eljárhatunk, ha mindkét „oldalt” megszorozzuk a  $\frac{q^{\text{régi}}(\theta)}{q_k^{\text{régi}}(\theta)}$  mennyiséggel, így a bal oldalon a teljes variációs poszterior jelenik meg<sup>11</sup>:

$$\underbrace{q_k(\theta) \cdot \frac{q^{\text{régi}}(\theta)}{q_k^{\text{régi}}(\theta)}}_{q(\theta)} \quad \text{vs.} \quad \frac{1}{Z_k} p(\mathbf{y}_k | \mathbf{X}_k, \theta) \cdot \frac{q^{\text{régi}}(\theta)}{q_k^{\text{régi}}(\theta)}$$

A közelítést a már jól ismert KL-divergencia minimalizálásával végezzük:

$$KL \left( q(\theta) \parallel \frac{1}{Z_k} p(\mathbf{y}_k | \mathbf{X}_k, \theta) \cdot \frac{q^{\text{régi}}(\theta)}{q_k^{\text{régi}}(\theta)} \right) = \int q(\theta) \ln \frac{p(\mathbf{y}_k | \mathbf{X}_k, \theta) q^{\text{régi}}(\theta)}{Z_k q_k^{\text{régi}}(\theta) q(\theta)} d\theta$$

A  $Z_k$  normalizáló konstansról meg tudunk szabadulni, ha a logaritmus azonosságait felhasználjuk, valamint észrevevessük, hogy  $q(\theta)$  integrálja 1-nek adódik:

$$\int q(\theta) \ln \frac{1}{Z_k} d\theta = -\ln Z_k \int q(\theta) d\theta = -\ln Z_k = \text{const.},$$

azaz ez a tag nem játszik szerepet a minimalizációban. Vezessük be a

$$\tilde{p}(\theta) := \frac{q^{\text{régi}}(\theta)}{q_k^{\text{régi}}(\theta)}$$

„effektív priort”<sup>12</sup>. A KL-divergencia megmaradó tagjaiba behelyettesítve azt a következő veszteségfüggvényt kapjuk<sup>13</sup>:

$$\begin{aligned} L(\theta) &:= - \int q(\theta) \ln p(\mathbf{y}_k | \mathbf{X}_k, \theta) d\theta - \int q(\theta) \ln \frac{\tilde{p}(\theta)}{q(\theta)} d\theta \\ &= \underbrace{\mathbb{E}_{q(\theta)} [-\ln p(\mathbf{y}_k | \mathbf{X}_k, \theta)]}_{\text{várható veszteség}} + \underbrace{KL(q(\theta) \parallel \tilde{p}(\theta))}_{\text{analitikus/GAN}}, \end{aligned}$$

amelyet a variációs kövteztetésnél tárgyalt módon optimalizálhatunk (5.3. szakasz). A teljes eljárást a 11.2. ábra szemlélteti.

### 11.3.1 Az implementáció részletei

Az eljárás során az osztások numerikus szempontból kellemetlen meglepetéseket okozhatnak; épp ezért célszerű olyan paraméterezést választani, ahol ez a probléma nem is igen merülhet fel. Az exponenciális eloszláscsalád általánosságban a következőképpen írható<sup>14</sup>:

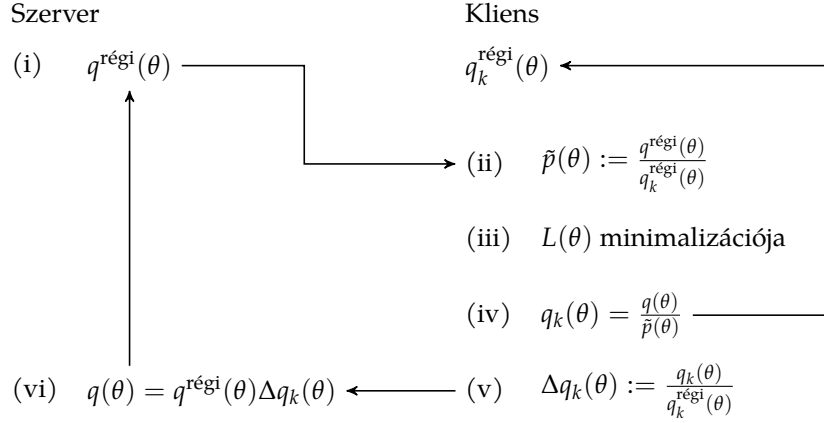
$$p(\theta | \boldsymbol{\eta}) = h(\theta) e^{T(\theta)^\top \boldsymbol{\eta} - A(\boldsymbol{\eta})},$$

<sup>11</sup> Azaz a  $k$ . darabka frissítése során a teljes  $q(\theta)$  variációs poszteriorot is figyelembe tudjuk venni; végső soron nem az a cél, hogy az egyes darabkák jók legyenek, hanem a teljes közelítő poszterior!

<sup>12</sup> A következő formula alapján jogosnak tűnik ez az elnevezés: tényleg egy VAE-jellegű veszteségfüggvényre jutottunk, ahol a prior szerepét  $\tilde{p}(\theta)$  játssza.

<sup>13</sup> A veszteségfüggvény intuitív értelmezése: az első tag az adatra való illeszkedésről gondoskodik, a második pedig arról, hogy ne sodródjunk túl távol a konszenzusos modellől.

<sup>14</sup> A családba sok, korábbról már ismerős eloszlás beletartozik: Gauss, Poisson, Beta, Gamma stb.



11.2. ábra. Bayesi federált tanulási séma. A szerver elküldi a klienseknek a globális modellt; azok frissítik saját hozzájárulásukat a lokális adataik alapján, majd statisztikai korrekciót küldenek vissza a szervernek. A szerver összesíti a korrekciókat és előáll az új globális modell.

ahol  $T$  valamilyen *elégséges statisztika*,  $\boldsymbol{\eta}$  pedig egy paramétervektor, amelyet *természetes paramétereknek* is szokás nevezni. Az  $A(\boldsymbol{\eta})$  tag a normalizációs konstans szerepét játssza<sup>15</sup>:

$$A(\boldsymbol{\eta}) := \ln Z(\boldsymbol{\eta}),$$

ahol

$$\ln Z(\boldsymbol{\eta}) = \int h(\theta) e^{T(\theta)^\top \boldsymbol{\eta}} d\theta.$$

A normális eloszlás például a következőképpen fest ezzel a felírással:

$$q(\theta \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\theta-\mu)^2}{2\sigma^2}} = e^{\frac{-\theta^2}{2\sigma^2} + \frac{2\mu\theta}{2\sigma^2} - A},$$

ahonnan leolvashatjuk, hogy<sup>16</sup>

$$T(\theta) = \begin{bmatrix} \theta \\ \theta^2 \end{bmatrix}, \quad \boldsymbol{\eta} = \begin{bmatrix} \frac{\mu}{\sigma^2} \\ -\frac{1}{\sigma^2} \end{bmatrix}, \quad h(\theta) = 1.$$

A 11.2. ábrán látható sémában az osztások most kivonásokká egyszerűsödnek, hiszen

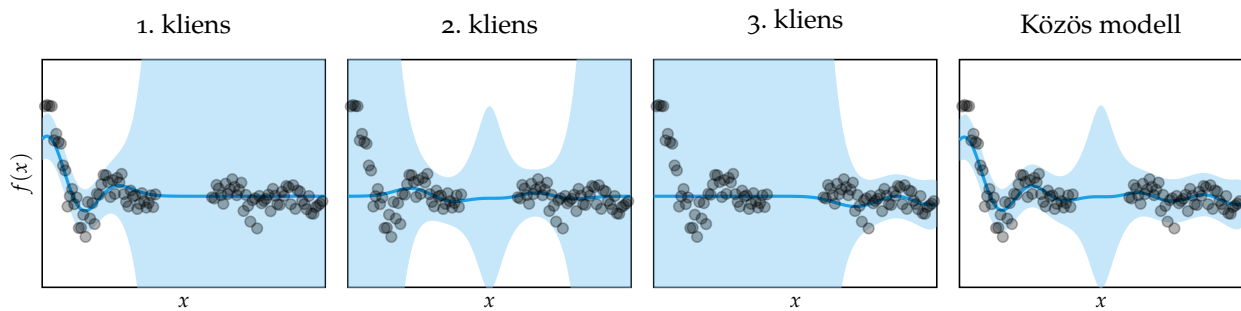
$$\frac{e^{T(\theta)^\top \boldsymbol{\eta}}}{e^{T(\theta)^\top \boldsymbol{\eta}'}} = e^{T(\theta)^\top (\boldsymbol{\eta} - \boldsymbol{\eta}')}. \quad \square$$

<sup>15</sup> Mindössze annyi történt, hogy a normalizációt a hatványkitevőbe „költöztettük”.

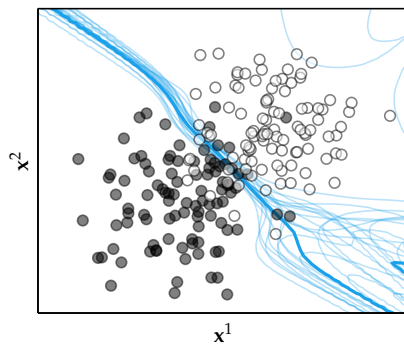
<sup>16</sup> Természetesen  $A$ -t is ki tudnánk számolni, de nem lesz rá szükségünk; korábbi jó szokásunkhoz híven a normalizáció mindvégig implicit marad.

Összességében tehát elég lesz  $q$ -t ügyesen, az  $\eta$  természetes paraméterekkel felírni<sup>17</sup>; az optimalizáció szempontjából mindegy, hogy milyen paraméterezést választunk, ám a többi lépést jelentősen megkönnyíti. A 11.3. ábra az algoritmus kimenetét mutatja be regressziós esetben:

<sup>17</sup> Ekkor természetesen az exponenciális eloszláscsaládra korlátozzuk a módszerünket, de ezzel hajlandók vagyunk kiegyezni.



A klasszifikációt pedig a 11.4 ábrán láthatjuk:



11.3. ábra. Bayesi nemlineáris regresszió három klienssel, akik az adatoknak csak az első, második és harmadik harmadát „látják”.

11.4. ábra. Klasszifikáció három klienssel; a kék vonalak a globális variációs poszteriorból származó minták.

### 11.3.2 Kód

Végül essen néhány szó a konkrét megvalósításról. A következő választásokkal élünk:

- Az egyszerűség kedvéért egyetlen (bayesi) súlyvektort használunk normális eloszlással<sup>18</sup>,

<sup>18</sup> Azaz bayesi (nem)lineáris modellt választunk, vagy másképpen, egyrétegű neurális hálózatot.

- A veszteségfüggvényt nem kötjük meg, azaz regressziós, klasszifikációs vagy más jellegű feladatot is megoldhatunk,
- Szintén nem kötjük meg a kliensek számát, ezek párhuzamosan is működhetnek.

```

struct NormalLayer
    η1
    η2
end
NormalLayer(dim) = NormalLayer(zeros(dim), zeros(dim))
Base.:+(p::NormalLayer, q::NormalLayer) =
    NormalLayer(p.η1 .+ q.η1, p.η2 .+ q.η2)
Base.:-(p::NormalLayer, q::NormalLayer) =
    NormalLayer(p.η1 .- q.η1, max.(1e-6, p.η2 .- q.η2))

```

11.1. algoritmus. Normális eloszlású súlyvektor természetes paraméterekkel. A séma által megkövetelt szorzás-osztás helyett elég a hatványkitevőben összeadást-kivonást végezni. Külön gondoskodunk arról, hogy a paraméter értéke ne fusson ki az értelmezési tartományból.

```

function kl(p::NormalLayer, q::NormalLayer)
    sum(@. (p.η2/q.η2 +
            (p.η1/p.η2 - q.η1/q.η2)^2*q.η2 -
            log(p.η2) + log(q.η2) - 1)/2)
end

```

11.2. algoritmus. KL-divergencia számítása két természetesen paraméterezett normális eloszlás között.

```

mutable struct Client
    X      # Adatok
    y      # Címkék
    qk     # Paraméter
    nll    # Negatív log-likelihood
    κ      # KL-tag súly (regularizáció)

    Client(X, y, nll; κ=0.1) = new(
        X, y,
        NormalLayer(size(X, 2)),
        nll, κ)
end

```

11.3. algoritmus. Kliens implementációja tetszőleges veszteségfüggvénnyel (negatív log-likelihood).

```
mutable struct Server
    q
    clients

    Server(p,clients) = new(p,clients)
end
```

11.4. algoritmus. Szerver, amely a globális modellt tartalmazza és koordinálja a tanulást.

```
function update!(c::Client, q; α=0.01)
    q = deepcopy(q)
    qk = deepcopy(c.qk)
    p = q-qk

    grad = gradient(Params([q.η1,q.η2])) do
        smp = randn(size(q.η2))./sqrt.(q.η2) .+ q.η1./q.η2
        out = c.X*smp
        c.nll(out,c.y) + kl(p,q)*c.κ
    end

    @. q.η1 -= α*grad[q.η1]
    @. q.η2 -= α*grad[q.η2]

    c.qk = q - p
    Δq = c.qk - qk

    return Δq
end
```

11.5. algoritmus. Kliens frissítésének lépései. A kliens a globális modell megérkezése után gradiens-alapon optimalizál, majd statisztikai korrekciót küld vissza.

```
function train!(m::Server; num_epochs=10)
    for i in 1:num_epochs
        Δq_ = []
        for c in m.clients
            Δq = update!(c,m.q)
            push!(Δq_,Δq)
        end

        for i in 1:length(Δq_)
            m.q += Δq_[i]
        end
    end
end
```

11.6. algoritmus. A szerver frissítési sémája; kiküldi a klienseknek a globális modellt, majd begyűjti a korrekciókat és frissít a globális modellt.



## A. függelék

# Newton–Raphson módszer

A Newton–Raphson módszert differenciálható függvények gyökeinek megtalálására, vagy másképpen szólva egyenletmegoldásra használhatjuk. Vegyünk egy  $f : \mathbb{R} \rightarrow \mathbb{R}$  függvényt, majd fejtsük Taylor-sorba első rendig egy  $x_0$  pont körül<sup>1</sup>:

<sup>1</sup> Ez a függvény „legjobb” lineáris közelítése  $x_0$  körül.

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

Ahelyett, hogy  $f$  gyökeit keresnénk, egyelőre beérjük a lineáris közelítés zérushelyével, azaz a

$$0 = f(x_0) + f'(x_0)(x - x_0)$$

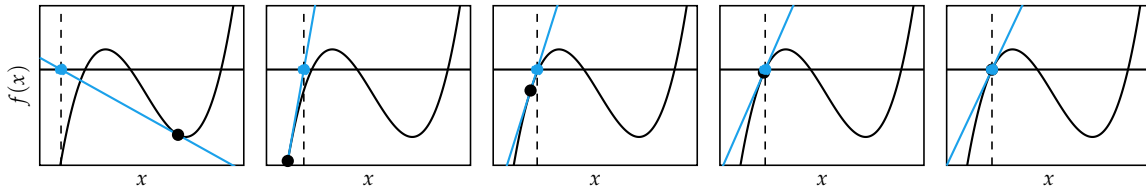
egyenletet kell megoldanunk. Átrendezés után a zérushelyre

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

adódik. A Newton–Raphson módszer lényege, hogy a gyökkeresést iteratívan végezzük, azaz  $x = x_0$ -ból indulva a

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

lépéseket ismételtetjük. Az eljárást az A.1. ábra szemlélteti.



A.1. ábra. Gyökkeresés Newton–Raphson módszerrel.

Az eljárás tehát lépésekben:

1. Elindulunk egy  $x$  pontból,
2. Az  $x$  pontban érintőt húzunk a függvényhez,
3.  $x$  új értéke az érintő zérushelye,
4. Ismétljük a 2-3. lépéseket, amíg a  $f(x)$  értéke konvergál 0-hoz<sup>2</sup>.

Ha a függvény kétszer differenciálható, az eljárást felhasználhatjuk stacionárius pontok (pl. minimumhely) keresésére is; ehhez csupán  $f'(x)$ -re kell alkalmaznunk. A séma ebben az esetben

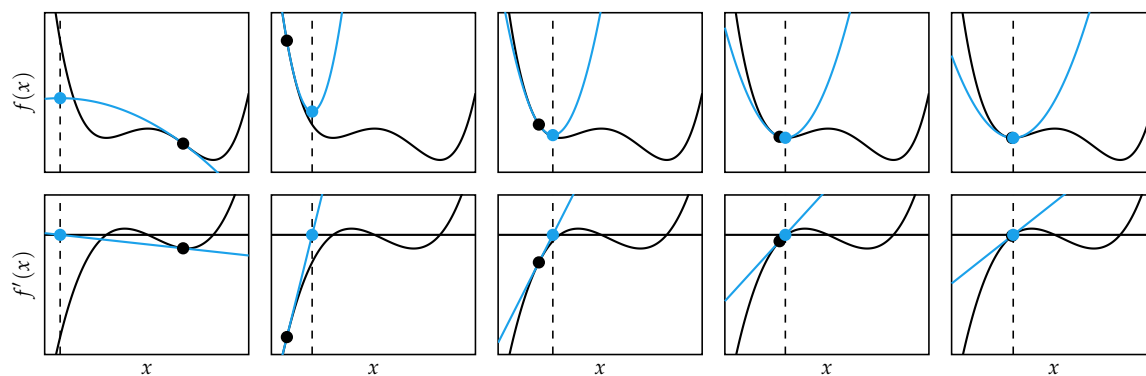
$$x \leftarrow x - \frac{f'(x)}{f''(x)}$$

formát ölti. Az A.1 algoritmus egy egyszerű implementációt mutat be.

```
function NewtonRaphson(x, f', f''; ε=1e-12)
    d = f'(x)
    while d^2 > ε
        x -= d / f''(x)
        d = f'(x)
    end
    return x
end
```

<sup>2</sup> Nem megyünk bele, hogy milyen feltételek mellett számíthatunk gyors konvergenciára. Legyen elég annyi, hogy az eljárás rendszerint jól működik a gyakorlatban, ha a valódi zérushelytől „nem túl távol” indítjuk a keresést.

A.1. algoritmus. Newton–Raphson algoritmus egyszerű implementációja stacionárius pontok keresésére.



Az eljárást az A.2. ábra mutatja; a lépések intuitíve a következőképpen értelmezhetők:

1. Elindulunk egy  $x$  pontból,
2. Megkeressük a függvényt az  $x$  pontban legjobban közelítő parabolát,
3.  $x$  új értéke a parabola szélsőértékhelye,
4. Ismételjük a 2-3. lépéseket, amíg a derivált meredeksége konvergál 0-hoz.

A.2. ábra. Newton-Raphson algoritmus stacionárius pontok keresésére. A felső sor a függvényt, az alsó a függvény első deriváltját ábrázolja a megfelelő közelítésekkel.