

DRAFT

NeuralMath

Egy neurális hálózat
matematikai anatómiája

Magyar

A PORTUGÁL EREDETI MAGYAR FORDÍTÁSA



Americo Cunha Jr.

neuralmath.org/hu/

$$\hat{y} = \sigma(Wx + b)$$

Impresszum

Egy neurális hálózat matematikai anatómiája

Americo Cunha Jr.

Laboratório Nacional de Computação Científica (LNCC), Petrópolis, Brasil
Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, Brasil

ORCID: [0000-0002-8342-0363](https://orcid.org/0000-0002-8342-0363) · americo@lncc.br

Hivatkozás

Cunha Jr, A. (2026). A anatomia matemática de uma rede neural (Version 1.0). Zenodo.

<https://doi.org/10.5281/zenodo.23212381>



neuralmath.org/hu/

Eredeti cikk

Cunha Jr, A. A anatomia matemática de uma rede neural. Professor de Matemática Online (PMO). Manuscrito submetido, 2026.

Licenc

A szöveg a Creative Commons Nevezd meg! 4.0 Nemzetközi (CC BY 4.0) licenc alatt terjeszthető. [CC-BY-4.0](https://creativecommons.org/licenses/by/4.0/)

DOI: [10.5281/zenodo.23212381](https://doi.org/10.5281/zenodo.23212381) · **Verzió:** 1.0 (2026-10)

Kiegészítő anyagok

[Ez a kiadás a weben](#)

[Futtatható notebook \(Google Colab\)](#)

[Forráskód \(GitHub\)](#)

Hibát talált? Javításokat és javaslatokat itt várunk:

<https://github.com/amicocunhajr/NeuralMath/issues>

Előszó

Ez a szöveg egy egyszerű ötletből született: tegyük láthatóvá egy kis neurális hálózat belsejében zajló matematikai műveleteket. Sok bevezető bemutatásban a neurális hálózatok erős, de átláthatatlan eszközként jelennek meg. Az olvasó megtanul egy könyvtárat meghívni, betanítást futtatni és veszteséggörbét vizsgálni anélkül, hogy feltétlenül látná az adatokból döntést előállító műveletsort. Itt az ellenkező utat követjük. Egy elemi geometriai problémából indulunk, a lineáris osztályozót mesterséges neuronként értelmezzük, bevezetjük az aktivációs függvényeket, neuronokat kapcsolunk össze egy kis hálózattá, majd lépésről lépésre eljutunk a gradiensekhez, a gradienscsökkenéshez és a visszatérjesztéshez.

A szöveg viszonylag szerény kalkulus- és statisztikai előismeretekkel rendelkező olvasóknak készült: hallgatóknak, tanároknak és önálló tanulóknak. A cél nem az, hogy a matematikát eltüntető mértékben egyszerűsítsünk, hanem hogy az új gondolatokat ismerős elemekre építsük: síkbeli pontra, egyenesre, függvénykompozícióra, deriváltra és láncszabályra. A Python- és NumPy-kód ennek a konstrukciónak természetes következménye; az egyenletek és az algoritmus ugyanannak az objektumnak két leírása. Ezért a legjobb módszer úgy olvasni, hogy a notebook közben nyitva van, és minden lépést lefuttatunk, majd módosítunk.

Ez a szöveg a 26 nyelven elérhető NeuralMath gyűjtemény része. Minden változat ugyanazokat az egyenleteket, ábrákat, numerikus példákat és szakaszszámozást őrizi, így különböző nyelvű olvasók együtt tanulhatnak. A matematika és a számítástechnika tanulásának fontos akadályja nemcsak fogalmi, hanem nyelvi is, ezért minden olvasó megérdemel egy belépési pontot a saját nyelvén.

Céлом, hogy az olvasó kevésbé titokzatos képpel fejezze be az utat a neurális hálózatokról. Nem kell úgy tenni, mintha a modern neurális rendszerek egyszerűek lennének. Elég megmutatni, hogy a mai alkalmazások mérete és összetettsége alatt továbbra is olyan matematikai gondolatok működnek, amelyek megvizsgálhatók, levezethetők, megvalósíthatók és megérthetők.

Americo Cunha Jr. · 2026

Erről a kiadásról

Ez a NeuralMath gyűjtemény magyar kiadása, a portugál eredeti fordítása. FILL IN: how the translations were produced. A portugál szöveg marad a referencia-változat; eltérés esetén az az irányadó.

A fordítást még nem ellenőrizte anyanyelvi lektor.
Javításokat itt várunk:
<https://github.com/amicocunhajr/NeuralMath/issues>.



Google Colab · HU

Olvassa nyitott notebook mellett: a szöveg minden szakasza egy futtatható és módosítható kódrészletnek felel meg. Olvassa be a QR-kódot, hogy megnyissa e kiadás notebookját a Google Colabban. A másik 25 nyelvű változat az utolsó oldalon található.

Egy neurális hálózat matematikai anatómiája

Americo Cunha Jr 

Összefoglalás

Egyszerű osztályozási feladaton keresztül adunk matematikai bevezetést a mesterséges neurális hálózatokhoz. Egy lineáris osztályozó természetes módon vezet a mesterséges neuron fogalmához; ezt követően tárgyaljuk a súlyokat, az eltolásokat (biasokat), az aktivációs függvényeket, valamint a neuronok kis hálózattá kapcsolását. A tanítást veszteségfüggvények, deriváltak, gradiensmódszer és visszatérjesztés segítségével vezetjük be. Végül a modellt közvetlenül Pythonban valósítjuk meg, így az egyenletek és az algoritmus közötti kapcsolat teljesen egyértelművé válik. Célunk annak bemutatása, hogy az analitikus geometria, a lineáris algebra és a differenciálszámítás jól ismert fogalmai hozzáférhető alapot adnak a neurális hálózatok alapvető mechanizmusainak megértéséhez.

Kulcsszavak: mesterséges neurális hálózatok; gépi tanulás; matematikai oktatás; visszatérjesztés; Python.

1. Mikor képes tanulni egy számítógép?

A mindennapi életben szinte észrevétlenül hozunk osztályozási döntéseket. Ha ránézünk egy gyümölcsre, gyorsan felismerjük, hogy alma vagy narancs; amikor üzenet érkezik a telefonunkra, megkülönböztetjük a valódi kommunikációt a kénytelen reklámtól; orvosi elemzésben több mérés együtt támogathat egy döntést; pénzügyi intézményekben pedig számos információt értékelnek a hitelnyújtás előtt. Mindezekben a példákban egyszerre több jellemzőt veszünk figyelembe egy válasz előállításához.

Sok hagyományos számítógépes programban az ilyen válaszhoz vezető szabályokat a programozó kifejezetten megadja. Ez a megközelítés jól működik, ha a szabályok ismertek és világosan megfogalmazhatók. Kevésbé hatékony azonban akkor, amikor az adatok nagyon változatosak, vagy amikor nehéz előre felsorolni egy jó döntéshez szükséges összes feltételt. Ez természetes kérdéshez vezet:

Készíthetünk-e olyan programot, amely példákból tanulja meg a döntési szabályt ahelyett, hogy azt előre megkapná?

Ez a *gépi tanulás* egyik központi gondolata, amely a mesterséges intelligencia azon területe, amely adatokból illesztett modellekkel foglalkozik [2, 3]. Felügyelt tanulási feladatban az algoritmus példákat kap a hozzájuk tartozó helyes válaszokkal együtt, és olyan modellt

keresünk, amely a tanítás során nem használt esetekre is hasznos előrejelzéseket ad. Az erre a célra kidolgozott sok modell közül különösen jelentőssé váltak a *mesterséges neurális hálózatok*, amelyeket többek között képfelismerésben, nyelvfeldolgozásban, jelelemzésben és idősor-előrejelzésben alkalmaznak [6].

A neurális hálózatoknak a matematika, a számítástechnika és az iskolai oktatás közötti hídként való használatát Batista és Martins már tárgyalták a *Professor de Matemática Online* folyóiratban [1]; munkájuk tanárképzéssel, Pythonnal és számjegyfelismeréssel kapcsolatos tevékenységet mutat be. Jelen cikk kiegészítő megközelítést követ. Fő célunk egy kis neurális hálózat belső matematikájának feltárása: az analitikus geometriától indulunk, majd lépésről lépésre eljutunk a gradiens kiszámításáig és a visszaterjesztés kézi megvalósításáig.

A modern alkalmazások kifinomultsága ellenére a neurális hálózat matematikai magja viszonylag ismert fogalmakkal is bevezethető. Alapszíntén a neurális hálózat olyan függvények kompozíciója, amelyek paramétereit adatok alapján állítjuk be. Egy egyszerű geometriai problémával kezdünk, az elválasztó egyenest mesterséges neuronként értelmezzük újra, több neuront összekapcsolunk, megvizsgáljuk az aktivációs függvények szerepét, és megmutatjuk, miként használhatók a deriváltak a hálózat paramétereinek automatikus módosítására. Ezután egy teljes, kisméretű példát valósítunk meg Pythonban, közvetlen megfelelést tartva fenn az egyenletek és a kód között.

2. A probléma matematikai reprezentációja

A neurális hálózat működésének megértéséhez szándékosan egyszerű problémával kezdünk: almák és narancsok megkülönböztetésével. Egy gyümölcsnek sok olyan tulajdonsága van, amely hasznos lehet ehhez a feladathoz, például a színe, tömege, átmérője, felülete vagy illata. Egy kezdeti matematikai modellnek azonban nem kell ezt a teljes összetettséget tartalmaznia. Mindössze két könnyen mérhető mennyiséget használunk: a tömeget grammal és az átmérőt centiméterben.

Minden gyümölcsöt tehát egy rendezett párral reprezentálunk:

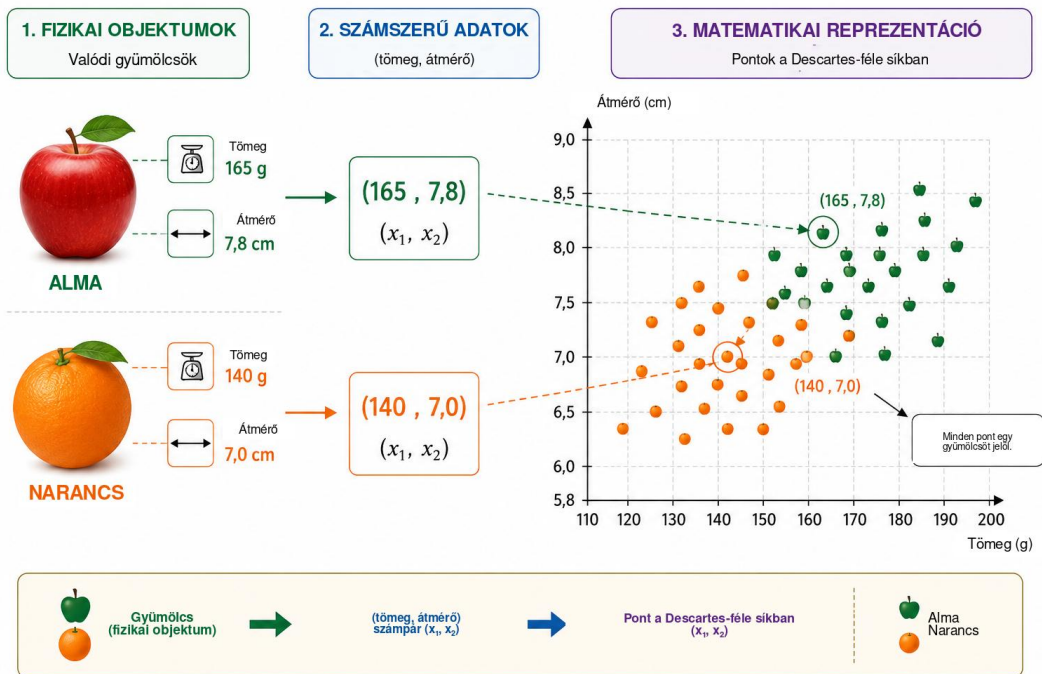
$$(x_1, x_2),$$

ahol

$$x_1 = \text{tömeg} \quad \text{és} \quad x_2 = \text{átmérő}.$$

Így a (155, 7.2) pár egy 155 g tömegű és 7.2 cm átmérőjű gyümölcsöt jelent. A fizikai objektumot két számszerű mérésre cserélve minden gyümölcsőhöz egy pontot rendelünk a Descartes-féle síkban: az első koordináta a tömeg, a második az átmérő. Az 1. ábra ezt a reprezentációt szemlélteti almák és narancsok hipotetikus halmazán.

Ez az átalakítás megváltoztatja a probléma látszólagos természetét. Ahelyett, hogy közvetlenül azt kérdeznénk, miként ismerhető fel egy alma vagy narancs, geometriai kérdést tehetünk fel: hogyan választhatók szét a két osztályhoz tartozó pontok a síkban? Ez a nézőpontváltás alapvető a gépi tanulásban. A képeket, szövegeket, hangokat és kísérleti jeleket számszerű reprezentációvá kell alakítani, mielőtt egy algoritmus feldolgozhatná őket. Példánkban minden objektumnak csak két koordinátája van; valós alkalmazásokban egy reprezentáció több száz, több ezer vagy akár több millió komponensből is állhat. Az elv



1. ábra: A fizikai megfigyeléstől a matematikai modellezésig. Minden gyümölcsöt először két mérhető mennyiség — tömeg és átmérő — ír le, majd az (x_1, x_2) rendezett párral reprezentáljuk. A tömeg-átmérő síkban minden megfigyelés ponttá válik, így az almák és narancsok megkülönböztetése két osztály geometriai szétválasztásának problémájaként értelmezhető.

ugyanaz: minden példához számok egy gyűjteményét rendeljük, a modell pedig olyan szabályt keres, amely ezt a reprezentációt a kívánt válaszhoz rendeli.

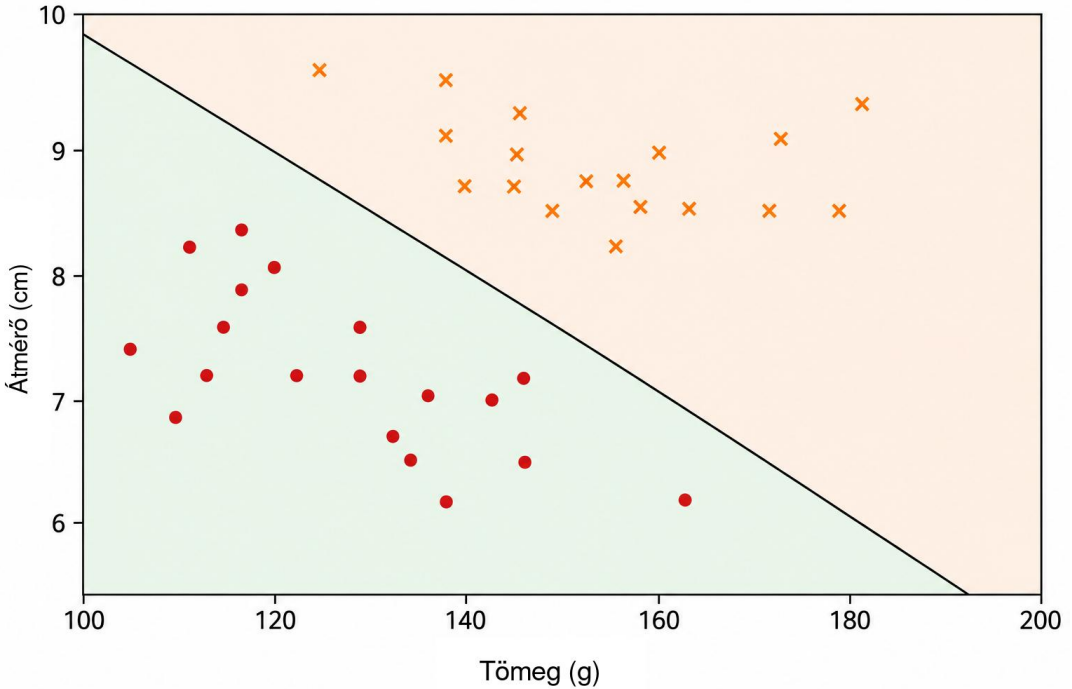
A legegyszerűbb rendelkezésre álló geometriai szabállyal kezdünk, és megkérdezzük, hogy egyetlen egyenes képes-e megfelelően elválasztani az almákhoz tartozó pontokat a narancsokhoz tartozó pontoktól.

3. Az egyenes mint osztályozó

Miután minden gyümölcsöt pontként ábrázoltunk a Descartes-féle síkban, az osztályozás geometriai problémává válik: el tudja-e választani egy egyenes az almákat a narancsoktól? Ez a kérdés megragadja a gépi tanulás egy fontos módszercsaládjának, a *lineáris osztályozóknak* az alapötletét.

Tekintsünk egy, az adatfelhőt metsző egyenest, ahogy az 2. ábra mutatja. Az egyenes két félsíkra osztja a síkot. Ha az almákhoz tartozó pontok többsége az egyik oldalon, a narancsokhoz tartozó pontok többsége pedig a másikon helyezkedik el, akkor egy új gyümölcs egyeneshez viszonyított helyzete osztályozási szabályként használható. Tömegének és átmérőjének megmérése után a gyümölcsöt pontként ábrázoljuk, majd meghatározzuk, hogy

az egyenes melyik oldalán fekszik.



2. ábra: Lineáris osztályozó példája a tömeg–átmérő síkban. Az egyenes döntési határként működik: az egyik félsík pontjai az egyik, a másik félsík pontjai a másik osztályba kerülnek. A döntés tehát kizárólag attól függ, hogy a pont az egyenes melyik oldalán helyezkedik el.

A (x_1, x_2) Descartes-féle síkban egy egyenes felírható

$$w_1x_1 + w_2x_2 + b = 0, \quad (w_1, w_2) \neq (0, 0).$$

A (w_1, w_2) vektor merőleges az egyenesre, ezért meghatározza annak irányát, míg a b paraméter a helyzetét befolyásolja. Ez a reprezentáció kis redundanciát tartalmaz: w_1 , w_2 és b ugyanazzal a nemnulla konstanssal történő megszorítása pontosan ugyanazt az egyenest adja. A határt leíró paraméterek ezért nem egyértelműek. Számszerű értelmezésük a változók skálájától is függ. Mivel a tömeget grammal, az átmérőt centiméterben mérjük, w_1 és w_2 számszerű nagysága nem hasonlítható közvetlenül össze a különböző mértékegységek figyelembevétele nélkül.

Az egyenes egyenlete egyszerű módot ad annak eldöntésére, hogy egy pont a határ melyik oldalán található. Defináljuk a pontszámot:

$$s(x_1, x_2) = w_1x_1 + w_2x_2 + b.$$

Ha $s(x_1, x_2) = 0$, a pont az egyenesen fekszik. Ha $s(x_1, x_2) > 0$, az egyik félsíkban, ha $s(x_1, x_2) < 0$, a másikban helyezkedik el. Ha úgy döntünk, hogy a pozitív pontszámok az

almáknak, a negatívak pedig a narancsoknak felelnek meg, akkor a szabály:

$$\begin{cases} s(x_1, x_2) \geq 0 & \Rightarrow \text{alma,} \\ s(x_1, x_2) < 0 & \Rightarrow \text{narancs.} \end{cases}$$

Például tekintsük a következő határt:

$$w_1 = \frac{1}{10}, \quad w_2 = 2, \quad b = -30.$$

Egy 160 g tömegű és 7.8 cm átmérőjű gyümölcs esetén

$$\begin{aligned} s &= \frac{1}{10} \times 160 + 2 \times 7.8 - 30 \\ &= 16 + 15.6 - 30 \\ &= 1.6. \end{aligned}$$

Mivel a pontszám pozitív, a pont abban a félsíkban van, amelyet az almákhoz társítottunk. Az osztályozó tehát almának címkézné ezt a gyümölcsöt.

Nem minden w_1 , w_2 és b választás eredményez jó osztályozást. Egyes egyenesek rosszul választják szét a két csoportot, míg mások kevesebb hibát okoznak. A tanulási feladat éppen az, hogy korábban címkézett példák segítségével olyan paraméterértékeket határozzunk meg, amelyek csökkentik a modell hibájának valamilyen mértékét. Ahelyett, hogy közvetlenül beprogramoznánk az egyenes helyzetét, azt szeretnénk, hogy az algoritmus az adatok alapján állítsa be az együtthatókat.

Egyetlen egyenes jól működhet, ha az osztályok közel lineárisan szeparálhatók. Összetettebb adathalmazok nemlineáris határt igényelhetnek. Mielőtt ezzel a korláttal foglalkoznánk, az imént létrehozott osztályozót elemi számítási egységként értelmezzük újra: mesterséges neuronként.

4. A mesterséges neuron

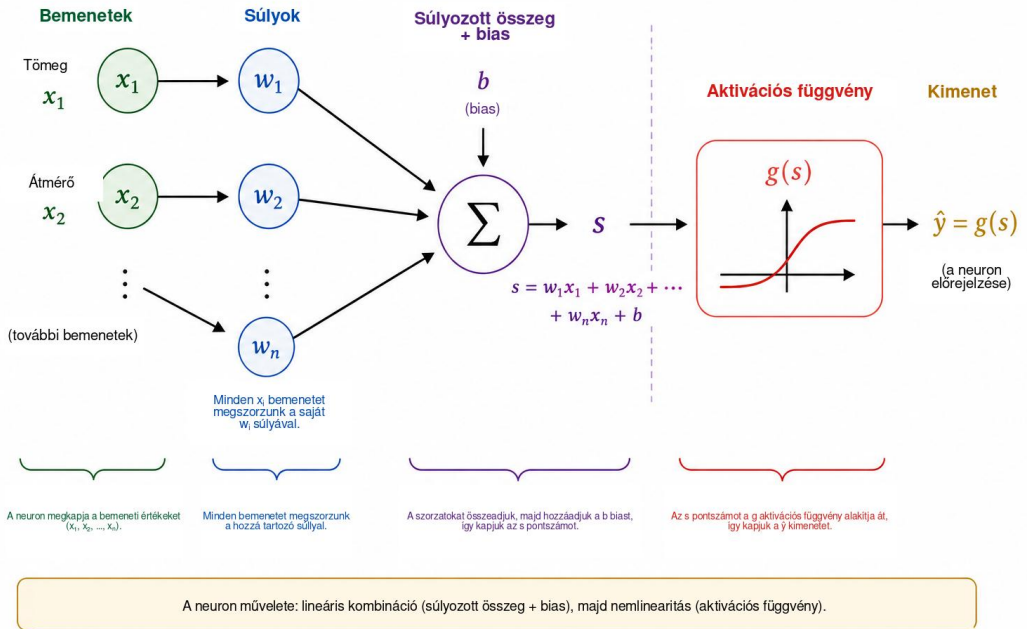
Az előző részben az

$$s = w_1 x_1 + w_2 x_2 + b$$

kifejezést annak meghatározására használtuk, hogy egy gyümölcs az egyenes melyik oldalán található. Ugyanez a kifejezés tekinthető a *mesterséges neuron* által végzett számítás első lépésének. A neuron számszerű bemeneteket kap, súlyok és eltolás segítségével kombinálja őket, majd egy matematikai függvényt alkalmaz a kimenet előállításához.

Példánkban a bemenetek a tömeg x_1 és az átmérő x_2 . Minden bemenetet megszorozunk egy *súly*nak nevezett paraméterrel; a szorzatokat összeadjuk egy b konstans taggal, amelyet *bias*nak (eltolásnak) nevezünk. Az 3. ábra szematikusan mutatja ezt a számítást.

A súlyok határozzák meg, hogyan befolyásolják a bemenetek változásai a pontszámot. Ha x_2 -t és minden más mennyiséget rögzítünk, akkor Δx_1 változás $w_1 \Delta x_1$ értékkel módosítja s -t. Hasonlóan, Δx_2 változás $w_2 \Delta x_2$ értékkel módosítja a pontszámot. Az előjel is számít: pozitív súly esetén a bemenet növekedése növeli a pontszámot, negatív súly pedig az ellenkező hatást váltja ki. Ezt az értelmezést óvatosan kell használni, ha a változók különböző



3. ábra: Egy mesterséges neuron működésének sematikus ábrája. Minden x_i bemenetet megszorozunk egy w_i súllyal; a szorzatokat a b bias-szal együtt összeadjuk, így kapjuk az s pontszámot. Ezután a g aktivációs függvény a pontszámot $\hat{y} = g(s)$ kimenetűvé alakítja, amely köztes mennyiségként vagy végső előrejelzésként használható.

mértékegységekkel és skálákkal rendelkeznek. Emiatt gyakorlati megvalósításokban a bemeneti adatokat tanítás előtt gyakran normalizálják vagy standardizálják.

A b bias az affin transzformáció konstans tagja. Geometriailag lehetővé teszi a döntési határ eltolását. Ha $b = 0$, akkor

$$w_1x_1 + w_2x_2 = 0$$

az egyenest arra kényszeríti, hogy átmenjen az origón. Bias jelenlétében a határ más helyre tolható. Ezzel ekvivalensen, ha $x_1 = x_2 = 0$, akkor a pontszám egyszerűen $s = b$.

A pontszámoknak nem kell a neuron végső kimenetének lennie. Általában egy g *aktivációs függvényt* alkalmazunk rá:

$$\hat{y} = g(s) = g(w_1x_1 + w_2x_2 + b).$$

A számítás tehát két lépésből áll:

$$s = w_1x_1 + w_2x_2 + b,$$

$$\hat{y} = g(s).$$

Az előző rész paraméterértékeivel $s = 1.6$. Ha a lépcsőfüggvényt választjuk,

$$g(s) = \begin{cases} 1, & s \geq 0, \\ 0, & s < 0, \end{cases}$$

akkor $\hat{y} = g(1.6) = 1$. Konvenciónk szerint a gyümölcsöt almának osztályozzuk. Ennél az aktivációnál a döntés akkor változik, amikor s áthalad a nullán, ezért az osztályhatárt pontosan

$$w_1x_1 + w_2x_2 + b = 0$$

adja.

Egyetlen neuron tehát egyszerű matematikai objektum állítható paraméterekkel. Valódi jelentősége akkor jelenik meg, amikor sok ilyen egységet összekapcsolunk. A hálózat felépítése előtt azonban meg kell értenünk, miért nélkülözhetetlen az aktivációs függvény.

5. Miért van szükség aktivációs függvényre?

A neuron először affin transzformációval kiszámít egy pontszámot, és csak ezután alkalmaz aktivációs függvényt. Ez a második lépés nélkülözhetetlen ahhoz, hogy a neurális hálózat bonyolult kapcsolatokat tudjon reprezentálni. Ennek megértéséhez képzeljünk el egy olyan hálózatot, amelyben egyáltalán nem használunk aktivációs függvényeket. Ekkor minden egység alakja

$$y = w_1x_1 + w_2x_2 + b.$$

Egydimenziós szemléltetésként tegyük fel, hogy az első egység

$$h = ax + b$$

értéket állít elő, a második pedig

$$y = ch + d$$

értéket számít. Behelyettesítéssel

$$y = c(ax + b) + d = acx + (cb + d),$$

ami továbbra is affin függvénye x -nek. Ugyanez igaz több dimenzióban is: affin transzformációk kompozíciója továbbra is affin. Ezért sok réteg hozzáadása nemlineáris művelet nélkül nem teszi a hálózatot alapvetően kifejezőbbé; az egész kompozíció helyettesíthető egyetlen affin transzformációval.

Az aktivációs függvény megszünteti ezt a korlátot. Az s pontszám kiszámítása után a neuron egy általában nemlineáris leképezést alkalmaz:

$$h = g(s).$$

Ha több ilyen egységet összekapcsolunk, az eredő kompozíció már nem redukálható egyetlen affin transzformációra. Ez teszi lehetővé bonyolultabb döntési határok felépítését.

A legegyszerűbb aktiváció a lépcsőfüggvény,

$$g(s) = \begin{cases} 1, & s \geq 0, \\ 0, & s < 0. \end{cases}$$

Bináris döntést állít elő, ezért kiváló az osztályozás megértéséhez. Deriváltja azonban $s \neq 0$ esetén nulla, $s = 0$ -nál pedig nem létezik. Mivel a gradiensalapú tanítás deriváltakat használ annak meghatározására, hogyan változnak a paraméterek, a lépcsőfüggvény nem alkalmas erre a tanítási mechanizmusra.

Egy alternatíva a szigmoid függvény,

$$\sigma(s) = \frac{1}{1 + e^{-s}}.$$

Minden valós számot szigorúan 0 és 1 közötti értékre képez. Nagyon negatív s esetén a kimenet 0-hoz, nagyon pozitív s esetén 1-hez közelít; a kettő között az átmenet sima. A szigmoid mindenütt differenciálható, és

$$\sigma'(s) = \sigma(s)(1 - \sigma(s)).$$

Mivel kimenetei 0 és 1 között vannak, gyakran használják bináris osztályozók kimeneti rétegében. Megfelelő veszteségfüggvénnyel együtt, valószínűségi megfogalmazásban a kimenet az 1-gyel jelölt osztály valószínűségének becsléseként értelmezhető. Rejtett rétegekben azonban a szigmoid nehézséget okozhat: nagy abszolútértékű bemeneteknél a deriváltja nagyon kicsivé válik, ami hozzájárul a mély hálózatok *eltűnő gradiens* problémájához.

Gyakran használt rejtettréteg-aktiváció a ReLU (*Rectified Linear Unit*),

$$\text{ReLU}(s) = \max(0, s) = \begin{cases} 0, & s < 0, \\ s, & s \geq 0. \end{cases}$$

Bár két lineáris szakaszból áll, a ReLU globálisan nem lineáris. Deriváltja 0 $s < 0$ esetén és 1 $s > 0$ esetén. $s = 0$ -nál a szokásos derivált nem létezik, ezért a szoftveres megvalósítások valamilyen konvenciót használnak a gradiensre. Pozitív bemeneteknél a derivált nem csökken s növekedésével, így elkerülhető a szigmoid egyik telítődési mechanizmusa.

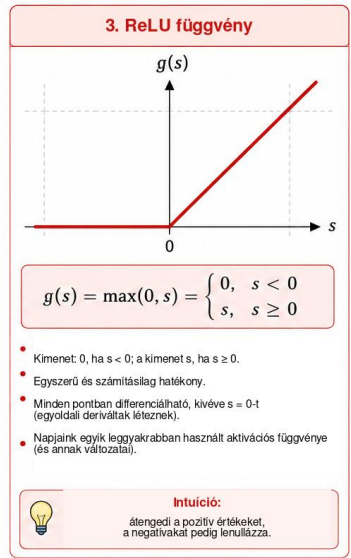
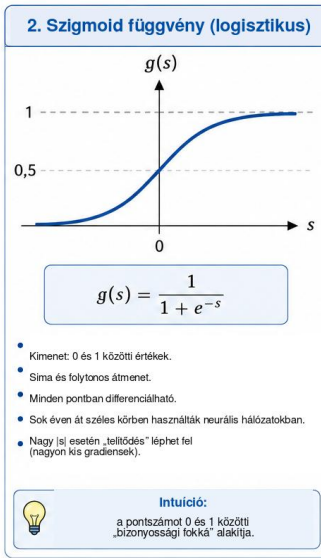
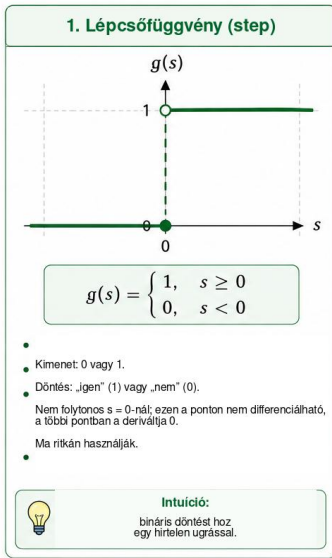
A 4. ábra vizuálisan hasonlítja össze ezt a három aktivációt: a lépcsőfüggvény hirtelen ugrását, a szigmoid sima átmenetét és a ReLU meredekségváltozását.

Fő tulajdonságaikat az alábbi táblázat foglalja össze.

Függvény	Kimenet	Derivált	Tipikus használat
Lépcső	$\{0, 1\}$	Nulla $s \neq 0$ esetén; $s = 0$ -nál nem létezik	Történeti modellek és didaktikai példák
Sigmoid	$(0, 1)$	Minden s esetén létezik	Bináris osztályozás kimeneti rétege
ReLU	$[0, \infty)$	0 $s < 0$ esetén és 1 $s > 0$ esetén	Rejtett rétegek

E függvények részletei eltérnek, de közös fogalmi szerepük a nemlinearitás bevezetése. Ez a nemlinearitás teszi lehetővé, hogy a következő rész kis hálózata olyan kapcsolatokat is reprezentáljon, amelyeket egyetlen egyenes nem tud megragadni.

A továbblépni kívánó olvasóknak Haykin [5] klasszikus tárgyalást ad a neurális hálózatokról, míg Goodfellow, Bengio és Courville [3] tágabb összefüggésben ismertetik az aktivációs függvényeket és a mély hálózatok tanítását. Nielsen szabadon hozzáférhető könyve [7] különösen jól használható a neuronokkal, aktivációkkal és tanulással kapcsolatos intuíció kialakításához.



Összefoglalás: Az aktivációs függvény nemlinearitást visz a neuronba. Nélküle (vagy ha mind lineáris) az egész hálózat egyetlen lineáris osztályozóként (egyeneként) viselkedik.

4. ábra: A neurális hálózatok bevezető tárgyalásaiban gyakran szereplő három aktivációs függvény összehasonlítása. A lépcsőfüggvény hirtelen döntést hoz; a szigmoid sima átmenetet biztosít 0 és 1 között; a ReLU pedig lenullázza a negatív bemeneteket, miközben megtartja a pozitívakat. A nemlineáris aktivációk megakadályozzák, hogy több réteg egymásra építése egyetlen affin transzformációra egyszerűsödjön.

6. Egy kis neurális hálózat felépítése

Most olyan hálózatot tekintünk, amelynek két bemenete, egy három neuront tartalmazó rejtett rétege és egy kimeneti neuronja van. A bemenetek továbbra is

$$x_1 = \text{tömeg}, \quad x_2 = \text{átmérő},$$

és mindkettőt minden rejtett neuron megkapja, ahogy az 5. ábra mutatja.

Jelölje a rejtett réteg kimeneteit h_1 , h_2 és h_3 . Mindegyik neuron ugyanazt a két bemenetet kapja, de saját súlyokkal és bias-szal rendelkezik:

$$h_1 = g(w_{11}x_1 + w_{12}x_2 + b_1),$$

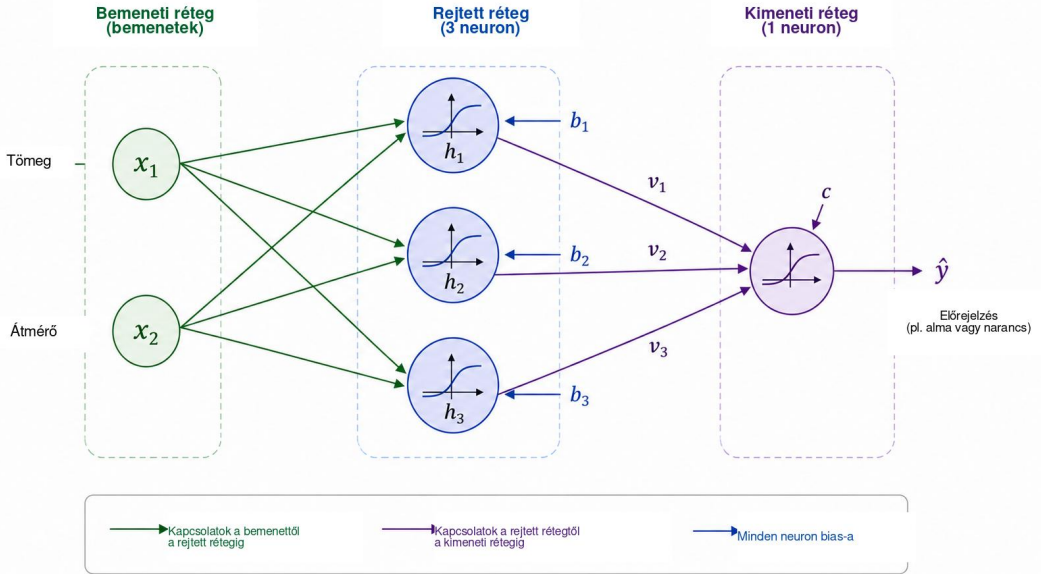
$$h_2 = g(w_{21}x_1 + w_{22}x_2 + b_2),$$

$$h_3 = g(w_{31}x_1 + w_{32}x_2 + b_3).$$

Bár mindhárom neuron ugyanazt a tömeget és átmérőt kapja, paramétereik különbözősége miatt eltérő válaszokat adhatnak. A tanítás során ezeket a paramétereket úgy módosítjuk, hogy a köztes aktivációk hasznosak legyenek az osztályozás szempontjából.

A három h_1 , h_2 és h_3 mennyiség ezután a kimeneti neuronhoz kerül, amely

$$s_{\text{out}} = v_1h_1 + v_2h_2 + v_3h_3 + c$$



5. ábra: A neurális hálózati számítás bevezetéséhez használt didaktikai 2–3–1 architektúra. A két bemenet — tömeg és átmérő — három rejtett rétegbeli neuront táplál. Ezek h_1, h_2, h_3 aktivációit ezután a kimeneti neuron kombinálja, hogy előállítsa a $\hat{y}$ előrejelzést. Minden kapcsolatnak állítható súlya, minden neuronnak pedig bias-a van.

értéket számít, majd

$$\hat{y} = g_{\text{out}}(s_{\text{out}})$$

kimenetet állít elő. Bináris problémánkhoz szigmoid kimenetet használunk:

$$\hat{y} = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3 + c), \quad 0 < \hat{y} < 1.$$

Ha az almára $y = 1$, a narancsra $y = 0$ jelölést használunk, akkor a 0.5 küszöb az alábbi osztálydöntést adja:

$$\hat{y} \geq 0.5 \Rightarrow \text{alma}, \quad \hat{y} < 0.5 \Rightarrow \text{narancs}.$$

Az 1-hez közeli értékek tehát erősen az almához, a 0-hoz közeli értékek a narancshoz társulnak. Ahogy korábban megjegyeztük, $\hat{y}$ valószínűségként való számszerű értelmezéséhez további feltételezések szükségesek a modelltől és annak tanításáról. Az itt használt osztályozási szabályhoz elegendő $\hat{y}$ -t a hálózat folytonos kimeneteként tekinteni.

Az architektúra központi eleme a köztes reprezentáció:

$$(x_1, x_2) \longrightarrow (h_1, h_2, h_3) \longrightarrow \hat{y}.$$

A hálózat két jellemzőt kap, és három új mennyiséget épít fel a végső válasz előállítására előtt. Ezeket az aktivációkat nem kézzel választjuk: a súlyoktól és biasoktól függenek, ezért a paraméterek tanításával együtt változnak.

Kis mérete ellenére a hálózat már tizenhárom állítható paraméterrel rendelkezik. A három rejtett neuronban

$$3(2 + 1) = 9$$

paraméter található, a kimeneti neuron pedig

$$3 + 1 = 4$$

paramétert ad hozzá. Tehát

$$9 + 4 = 13.$$

Ugyanez a számlálás mátrixos jelölésben is átlátható. Definiáljuk

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad W^{(1)} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Ekkor a rejtett réteg tömören így írható:

$$\mathbf{h} = g(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}),$$

ahol g komponensenként hat. A kimenetre

$$\hat{y} = \sigma(W^{(2)}\mathbf{h} + b^{(2)}),$$

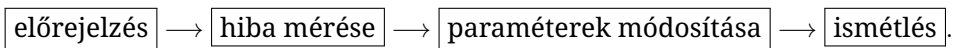
ahol

$$W^{(2)} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}.$$

Ez a jelölés előrevetíti a Python-program szerkezetét. Amikor egy gyümölcsöt a bemenetre adunk, és egymás után kiszámítjuk az aktivációkat egészen $\hat{y}$ -ig, *előreterjesztést* (*forward pass*) végzünk. A központi kérdés továbbra is az: hogyan határozható meg automatikusan a tizenhárom paraméter úgy, hogy az előrejelzések illeszkedjenek az adatokhoz?

7. Hogyan tanul egy neurális hálózat?

A súlyok és biasok előre nem ismertek. Olyan példák segítségével kell őket beállítani, amelyeknél a helyes válasz rendelkezésre áll. Ezt a folyamatot *tanításnak* nevezzük, és az alábbi ciklussal foglalhatjuk össze:



Tekintsünk egy 140 g tömegű és 7.0 cm átmérőjű gyümölcsöt, amelynek helyes osztálya narancs, tehát $y = 0$. Tegyük fel, hogy a tanítás egy pontján a hálózat $\hat{y} = 0.80$ kimenetet ad. Az előrejelzés és a cél közötti eltérés egyik egyszerű mértéke a négyzetes hiba:

$$E = \frac{1}{2} (y - \hat{y})^2.$$

Ebben a példában

$$E = \frac{1}{2} (0 - 0.80)^2 = 0.32.$$

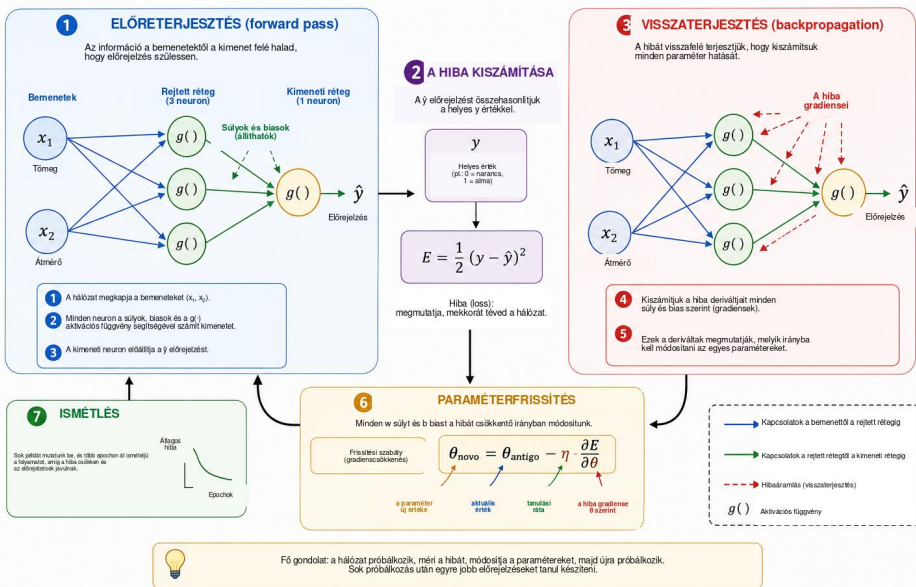
Ez a kifejezés egyetlen példához tartozó *veszteségfüggvény*. Mivel négyzet, $E \geq 0$, és a céltól távolabb eső előrejelzések nagyobb büntetést kapnak. Az $1/2$ tényező nem változtatja meg a minimum helyét, de egyszerűsíti a deriváltakat. Bináris osztályozásban gyakran más veszteségeket, különösen keresztentrópiát használnak. Itt a négyzetes hibát választjuk, mert algebraja átlátható és elegendő az optimalizálás és a visszaterjesztés bevezetéséhez [2, 3].

Ha a tanítóhalmaz N példát tartalmaz, egy globális mérték az átlagos veszteség:

$$J = \frac{1}{N} \sum_{i=1}^N E_i.$$

Minden előrejelzés függ a hálózat súlyaitól és biasaitól, így J is függ valamennyi paramétertől. A tanítás azt jelenti, hogy olyan paraméterértékeket keresünk, amelyek mellett J kicsi.

A módosítás mechanizmusának megértéséhez először képzeljük el, hogy egyetlen w súly kivételével minden paramétert rögzítünk. Ekkor a veszteséget $J = J(w)$ függvényként tekinthetjük. Az 6. ábra összefoglalja a tanítás alaplépéseit, amelyeket most megvizsgálunk.



6. ábra: A neurális hálózat tanításának alaplökusa. Az előreterjesztés a bemeneteket előrejelzéssé alakítja; a veszteségfüggvény méri az előrejelzés és a cél közötti eltérést; a visszaterjesztés a láncszabály segítségével kiszámítja a gradienseket; a gradiensmódszer pedig ezeket használja a súlyok és biasok frissítésére. E ciklus sok epochon át történő ismétlését nevezzük tanulásnak.

A differenciálszámításban kevésbé jártas olvasók számára a derivált itt a *lokális érzékenység* mértékeként értelmezhető: megmutatja, hogyan hajlamos megváltozni a veszteség, ha egy paramétert kicsit módosítunk. Előjele jelzi, melyik irány növeli vagy csökkenti lokálisan a függvényt, abszolútértéke pedig e helyi változás erősségét méri.

A

$$\frac{\partial J}{\partial w}$$

derivált a lokális meredekséget adja. Ha pozitív, akkor w kis növelése várhatóan növeli J -t, ezért a súlyt az ellenkező irányba kell mozdítani. Ha negatív, w növelése lokálisan csökkenti J -t. Mindkét esetet leírja

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial J}{\partial w},$$

ami a *gradienscsökkenés* alapötlete. A pozitív η szám a *tanulási ráta*; a lépés nagyságát szabályozza. Túl kicsi érték lassú tanítást, túl nagy érték pedig oszcillációt vagy instabilitást okozhat.

Sokparaméteres hálózatban jelölje θ bármely súlyt vagy biast. Ekkor a frissítés

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \frac{\partial J}{\partial \theta}.$$

Az összes parciális derivált együtt alkotja a veszteség *gradiensét*. Geometriailag a gradiens a leggyorsabb lokális növekedés irányába mutat, ezért a negatív gradiens lokális csökkenési irányt ad.

Még ki kell számítanunk ezeket a deriváltakat akkor is, amikor a hálózat több függvény kompozíciója. Az előreterjesztés során az információ útja

bemenetek $\rightarrow$ rejtett réteg $\rightarrow$ kimenet.

Annak meghatározásához, hogy egy korai réteg paramétere hogyan befolyásolja a veszteséget, matematikailag fordított irányban követjük a függőségi láncot. Ezt az eljárást *visszaterjesztésnek* (*backpropagation*) nevezzük. Lényegében a láncszabály szisztematikus és hatékony alkalmazásáról van szó. Rumelhart, Hinton és Williams [8] klasszikus hivatkozást adnak az eljárás rejtett rétegeket tartalmazó hálózatokban való elterjedéséhez; Nielsen [7] részletes és különösen didaktikus levezetést közöl.

Egy egyszerű függőségi láncban, ahol E függ $\hat{y}$ -tól, $\hat{y}$ függ egy h aktivációtól, az pedig egy w súlytól, a láncszabály szerint

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w}.$$

Minden tényező a w súlynak a végső veszteségre gyakorolt hatásának egy-egy szakaszát írja le. Nagyobb hálózatok sok ilyen láncot tartalmaznak, és a visszaterjesztés úgy szervezi a számítást, hogy a köztes deriváltak újra felhasználhatók legyenek, ahelyett hogy minden paraméterhez külön újraszámolnánk őket.

A gyakorlatban a tanítás sok példát használ. A tanítóhalmazon való egy teljes áthaladást *epochnak* nevezzük. A paraméterek frissíthetők minden egyes példa után, ahogy a cikk későbbi didaktikai programjában, vagy kis példacsoportok, úgynevezett *mini-batchek* után, ami nagyobb hálózatoknál elterjedt. Mindkét esetben az előrejelzések, veszteségek, gradiensok és frissítések ciklusa sok epochon át ismétlődik.

E fejlődés konkrét szemléltetésére követhetjük ugyanannak a rögzített példának az előrejelzését a tanítás különböző szakaszaiban. Tekintsük ismét a 140 g tömegű és 7.0 cm átmérőjű gyümölcsöt, amelynek helyes osztálya narancs. A 1. táblázat egy hipotetikus fejlődést

1. táblázat: A hálózat ugyanarra a gyümölcsre adott előrejelzésének hipotetikus alakulása a tanítás során. Az ismételt paraméterfrissítések egy kezdetben hibás kimenetet a célcím-kével összhangban lévő osztályozás felé mozdíthatnak.

Epoch	Előrejelzés $\hat{y}$	Veszteség E	Osztályozás
1	0.86	0.37	Alma
5	0.71	0.25	Alma
10	0.55	0.15	Alma
20	0.31	0.05	Narancs
40	0.09	0.004	Narancs

mutat. Ezek az értékek nem egy konkrét futtatáshoz tartoznak; csupán a mechanizmust szemléltetik.

Kezdetben a kimenet közel van 1-hez, ezért a gyümölcsöt tévesen almának osztályozza a hálózat. A paraméterek módosításával az előrejelzés csökken, és végül átlépi a 0.5 küszöböt, így a helyes osztályt adja. A folyamat során a súlyok és biasok számszerű értékei változnak; a hálózat architektúrája változatlan marad.

A tanítóadatokon mért veszteség csökkentése nem végső cél. Egy hasznos hálózatnak olyan példákra is jó előrejelzéseket kell adnia, amelyeket nem használtunk a paraméterek illesztéséhez. Ezt a képességet *általánosításnak* nevezzük. Ha a modell túl szorosan alkalmazkodik a tanítóhalmaz sajátosságaihoz és új adatokon rosszabbul teljesít, *túltanulásról* beszélünk. Gondosabb értékelésnél az adatokat gyakran tanító-, validációs és teszhalmazra osztják: az első a paraméterek illesztésére szolgál, a második a modellfejlesztés közbeni döntéseket támogatja, a harmadikat pedig a végső értékelésre tartják fenn. Az általánosításról és a modellértékelésről bővebben Faceli és munkatársai [2], valamint Goodfellow, Bengio és Courville [3] írnak.

Ezzel eljutottunk a neurális hálózatban történő tanulás matematikai jelentéséhez. Ebben az összefüggésben a tanulás azt jelenti, hogy egy összetett függvény paramétereit úgy módosítjuk, hogy az adatokon mért hibát csökkentsük, és ideális esetben új példákra is jól általánosítsunk. A hálózat előrejelzést készít, veszteséget számít, deriváltakkal meghatározza, hogyan hatnak paramétereire a veszteségekre, majd kis korrekciók sorozatát hajtja végre.

8. A matematikától a Pythonig

Most az egyenletektől egy futtatható program felé lépünk. Nem speciális neurális hálózati könyvtárat használunk, hanem a cikkben felépített matematikai műveleteket közvetlenül, NumPy segítségével valósítjuk meg [4]. Így sorról sorra láthatóvá válik, hogyan jelennek meg a kódban a lineáris kombinációk, az aktivációs függvények, a veszteség számítása, a visszaterjesztés és a gradienescsökkenés.

A meggyőzőbb végső példához szintetikus adathalmazt használunk a tömeg-átmérő tartomány szélesebb részén. Továbbra is csak két fizikai jellemzővel dolgozunk — tömeg grammban és átmérő centiméterben —, de most 64 tanítópéldát osztunk el a sík egy tág tartományában, és további nyolc példát tartunk fenn szemléltető tesztként. Az adatok szintetikusak maradnak, és almákra, illetve narancsokra ésszerű tartományokban helyezked-

nek el. Úgy konstruáljuk őket, hogy nemlineáris döntési határt eredményezzenek anélkül, hogy minden pont néhány túlságosan mesterséges klaszterbe tömörülne.

A geometriai ötlet egyszerű. Referenciagörbét definiálunk a tömeg-átmérő síkban, az al-mákat e görbe feletti két sávba, a narancsokat pedig alatta lévő két sávba helyezzük. A tömeghez és az átmérőhöz kis determinisztikus perturbációkat adunk, hogy a megfigyelések szélesebb tartományt töltsenek ki. Az így kapott példa gazdagabb a korábbi szemléltetések-nél: több pontot tartalmaz, az adatok a tartomány nagy részén eloszlának, és a megfelelő döntési határ egyértelműen görbültebb egy egyenesnél.

Ehhez a végső számítási kísérlethez a rejtett réteget tizenkét ReLU-neuronra bővítjük. Ez nem változtatja meg a korábbi részekben vizsgált fogalmi szerkezetet; csupán növeli a hálózat képességét egy összetettebb határ reprezentálására. A háromneuronos didaktikai hálózat 13 paramétere helyett most 49 állítható paraméterünk van: az első rétegben 24 súly és 12 bias, majd a kimeneti rétegben 12 súly és egy bias. Az egyenletek alakja pontosan ugyanaz marad; csak a vektorok és mátrixok lesznek nagyobbak.

A teljes program az alábbi.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

def relu(s):
    return np.maximum(0, s)

def relu_derivative(s):
    return (s > 0).astype(float)

def sigmoid(s):
    return 1 / (1 + np.exp(-s))

def base_boundary(m):
    return 6.68 + 0.74*np.exp(-((m-155)/20)**2) \
        + 1.50/(1 + np.exp(-(m-202)/5))

def build_data():
    masses_1 = np.linspace(118, 212, 16)
    masses_2 = masses_1 + np.array([
        1.2, -1.1, 0.8, -1.3, 1.0, -0.8, 1.1, -1.0,
        1.3, -0.9, 0.9, -1.2, 1.0, -1.0, 0.8, -0.7
    ])
    k = np.arange(16)

    a1 = np.column_stack([
        masses_1,
        base_boundary(masses_1) + 0.22 + 0.05*np.sin(0.8*k)
    ])
    a2 = np.column_stack([
        masses_2,
        base_boundary(masses_2) + 0.58 + 0.06*np.cos(0.45*k)
    ])
```

```

o1 = np.column_stack([
    masses_1,
    base_boundary(masses_1) - (0.22 + 0.04*np.cos(0.7*k))
])
o2 = np.column_stack([
    masses_2,
    base_boundary(masses_2) - (0.55 + 0.05*np.sin(0.5*k))
])

X_train = np.vstack([a1, a2, o1, o2])
y_train = np.array([1.0]*32 + [0.0]*32)

X_test = np.array([
    [121.0, 7.20], [135.0, 7.45],
    [152.0, 7.75], [171.0, 7.32],
    [121.0, 6.28], [152.0, 6.86],
    [189.0, 6.48], [208.0, 7.55]
])
y_test = np.array([1,1,1,1,0,0,0,0], dtype=float)

return X_train, y_train, X_test, y_test

def predict(X, mean, std, W1, b1, W2, b2):
    Xn = (X - mean) / std
    outputs = []
    for x in Xn:
        h = relu(W1 @ x + b1)
        y_hat = sigmoid(W2 @ h + b2)
        outputs.append(y_hat)
    return np.array(outputs)

X_train, y_train, X_test, y_test = build_data()

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
Xn = (X_train - mean) / std

rng = np.random.default_rng(42)
W1 = 0.1 * rng.standard_normal((12, 2))
b1 = np.zeros(12)
W2 = 0.1 * rng.standard_normal(12)
b2 = 0.0
eta = 0.05
n_epochs = 6000
history = []

for epoch in range(n_epochs):
    for i in rng.permutation(len(Xn)):
        x = Xn[i]
        target = y_train[i]

        z1 = W1 @ x + b1
        h = relu(z1)
        z2 = W2 @ h + b2
        y_hat = sigmoid(z2)

        delta2 = (y_hat - target) * y_hat * (1 - y_hat)
        dW2 = delta2 * h

```

```

    db2 = delta2
    delta1 = (delta2 * W2) * relu_derivative(z1)
    dW1 = np.outer(delta1, x)
    db1 = delta1

    W1 -= eta * dW1
    b1 -= eta * db1
    W2 -= eta * dW2
    b2 -= eta * db2

errors = []
for x, target in zip(Xn, y_train):
    h = relu(W1 @ x + b1)
    y_hat = sigmoid(W2 @ h + b2)
    errors.append(0.5 * (target - y_hat)**2)
history.append(np.mean(errors))

def make_figure():
    masses = np.linspace(116, 214, 500)
    diameters = np.linspace(6.0, 8.85, 500)
    M, D = np.meshgrid(masses, diameters)
    grid = np.column_stack([M.ravel(), D.ravel()])
    P = predict(grid, mean, std, W1, b1, W2, b2).reshape(M.shape)

    fig, ax = plt.subplots(figsize=(9.2, 6.3))
    ax.contourf(
        M, D, P,
        levels=[0.0, 0.25, 0.50, 0.75, 1.0],
        colors=["#f3e0cb", "#ecd5c4", "#dbe7cf", "#cde3bf"],
        alpha=0.95
    )
    ax.contour(M, D, P, levels=[0.5],
               colors=["#5a1a8a"], linewidths=2.2)

    mask = y_train == 1.0
    ax.scatter(X_train[mask,0], X_train[mask,1], marker="o",
               s=48, facecolors="white", edgecolors="black",
               linewidths=1.1, label="Alma - tanítás")

    mask = y_train == 0.0
    ax.scatter(X_train[mask,0], X_train[mask,1], marker="s",
               s=48, facecolors="white", edgecolors="black",
               linewidths=1.1, label="Narancs - tanítás")

    mask = y_test == 1.0
    ax.scatter(X_test[mask,0], X_test[mask,1], marker="^",
               s=92, color="#1f77b4", label="Alma - teszt")

    mask = y_test == 0.0
    ax.scatter(X_test[mask,0], X_test[mask,1], marker="x",
               s=110, color="#ff7f0e", linewidths=2.0,
               label="Narancs - teszt")

    ax.set_xlabel("Tömeg (g)")
    ax.set_ylabel("Ötmetér (cm)")
    ax.set_title("A neurális hálózat kimenete tanítás után")

    handles = [
        Line2D([0],[0], marker='o', markersize=9,
               markerfacecolor='white', markeredgcolor='black',
               linestyle='None', label='Alma - tanítás'),
        Line2D([0],[0], marker='s', markersize=9,

```

```

        markerfacecolor='white', markeredgecolor='black',
        linestyle='None', label='Narancs - tanítás'),
    Line2D([0],[0], marker='^', markersize=9, color='#1f77b4',
        linestyle='None', label='Alma - teszt'),
    Line2D([0],[0], marker='x', markersize=10, color='#ff7f0e',
        linestyle='None', label='Narancs - teszt'),
    Line2D([0],[0], color='#5a1a8a', lw=2.2,
        label='A neurális hálózat döntési határa')
]
ax.legend(handles=handles, loc='center left',
        bbox_to_anchor=(1.02, 0.5), frameon=True)

fig.tight_layout()
plt.show()

```

Bár a kód hosszabb, mint a cikk elején vizsgált első egyenletek, szerkezete továbbra is átlátható. A `build_data()` függvény automatikusan létrehozza a 64 tanítópéldát és a nyolc szemléltető teszt-példát. Ezután mindkét változót standardizáljuk:

$$x_j^* = \frac{x_j - \mu_j}{\sigma_j},$$

ahol μ_j és σ_j kizárólag a tanítóhalmazból számítható. Ez megakadályozza, hogy a teszt-halmaz befolyásolja a paraméterillesztést, és a tömeget meg az átmérőt összehasonlítható számszerű skálára hozza.

Az előreterjesztés lényegében nem változik. Az egyetlen különbség, hogy az első réteg most tizenkét rejtett aktivációból álló vektort állít elő. Mátrixos jelölésben

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{h} = \text{ReLU}(\mathbf{z}^{(1)}),$$

$$\mathbf{z}^{(2)} = W^{(2)}\mathbf{h} + \mathbf{b}^{(2)}, \quad \hat{y} = \sigma(\mathbf{z}^{(2)}).$$

Ez a négy sor továbbra is az algoritmus matematikai magját tartalmazza. A tanítás is megőrzi a korábban bemutatott logikát: előrejelzés számítása, hiba mérése, az érzékenységi információ visszaterjesztése deriváltakkal, majd a paraméterek frissítése kis gradienslépésekkel.

A kódban a kimeneti réteg visszaterjesztése itt jelenik meg:

```
delta2 = (y_hat - target) * y_hat * (1 - y_hat)
```

a rejtett réteg felé történő terjesztés pedig itt:

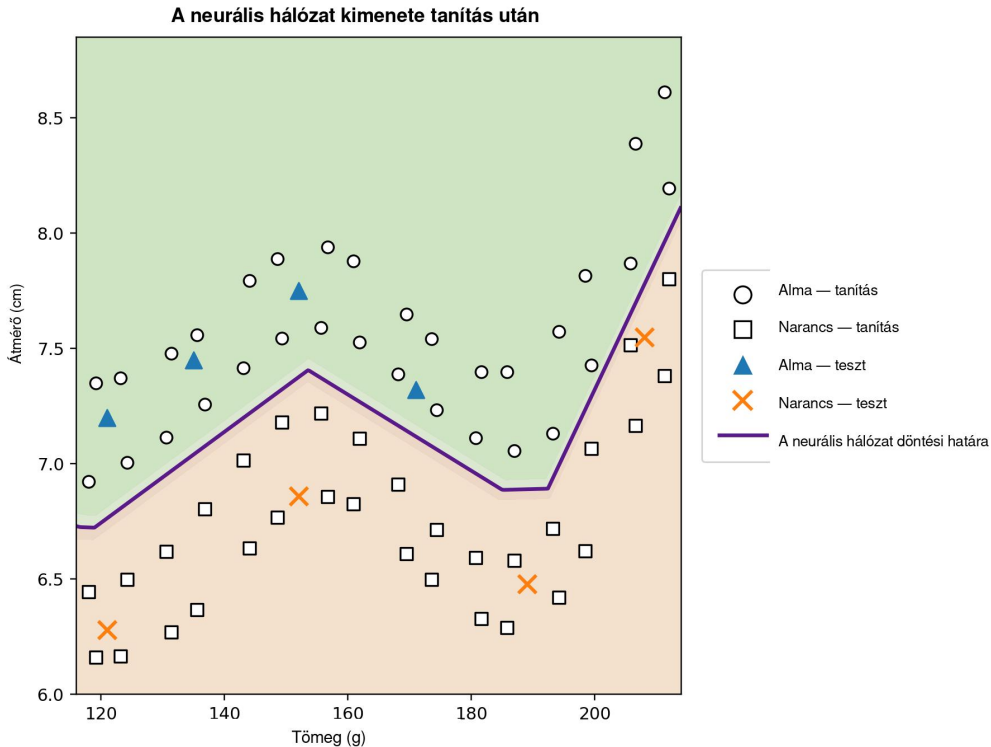
```
delta1 = (delta2 * W2) * relu_derivative(z1)
```

majd minden súly és bias frissítése következik. Mivel a példákat minden epochban megkeverjük, és a paramétereket minden megfigyelés után frissítjük, ez a megvalósítás a *sztochasztikus gradienscsökkenés* elemi formája.

Az itt használt véletlen maggal és hiperparaméterekkel a végső átlagos veszteség nagyságrendje 10^{-5} – 10^{-4} , és mind a 64 tanítópélda, mind a nyolc szemléltető teszt-példa a várt osztályozást kapja. Ezt nem szabad szigorú statisztikai teljesítményértékelésként értelmezni:

az adatok szintetikusak és didaktikai célból készültek. Feladatuk annak bemutatása, hogy kevésbé triviális környezetben miként tanulhat meg egy kis neurális hálózat egyértelműen nemlineáris döntési határt.

A 7. ábra mutatja a program végső kimenetét a tömeg-átmérő síkban. Minden tanítópont látható. A kiemelt görbe a $\hat{y} = 0.5$ szint, és a hálózat által megtanult döntési határt jelenti; az árnyékolt tartományok a két előre jelzett osztályt mutatják. A körök tanító almákat, a négyzetek tanító narancsokat, a háromszögek tesztalmákat, a keresztek tesztarancsokat jelölnek. A jelmagyarázat a fő rajzterületen kívül helyezkedik el, így egyetlen adatpontot sem takar el.



7. ábra: A Python kimenete a neurális hálózat tanítása után. A szintetikus adathalmaz 64 tanítópéldát tartalmaz a tömeg-átmérő sík széles tartományában, valamint nyolc további példát szemléltető teszthez. A kiemelt görbe a $\hat{y} = 0.5$ szint, és a megtanult döntési határt jelenti; az árnyékolt tartományok a hálózat által előre jelzett osztályokat mutatják. A körök tanító almákat, a négyzetek tanító narancsokat, a háromszögek tesztalmákat, a keresztek tesztarancsokat jelölnek.

Egy két egyszerű pontcsoportot elválasztó egyenessel kezdtünk, és egy gazdagabb számítási példáig jutottunk, amelyben görbült határ jelenik meg a hálózati paraméterek automatikus módosításának eredményeként. A matematikához semmi rejtélyeset nem adtunk hozzá: a megvalósítás csupán nagyobb léptékben és nagy sebességgel szervezi és ismétli az affin kombinációkat, aktivációs függvényeket, deriváltakat, mátrixszorzásokat és gradiensalapú

frissítéseket [3, 7].

A cikkhez kapcsolódó minden számítási anyag — beleértve a kódot és egy futtatható Google Colab jegyzetfüzetet — elérhető a <https://neuralmath.org> címen, egységes hozzáférési pontot biztosítva azoknak az olvasóknak, akik szeretnék felfedezni, reprodukálni vagy módosítani a példákat.

Nyilatkozat a mesterséges intelligencia használatáról. A szerző az OpenAI által fejlesztett ChatGPT-t támogató eszközként használta a szöveg megfogalmazásához, nyelvi ellenőrzéséhez, a kifejtés megszervezéséhez és didaktikai ábrák létrehozásához. A szerző teljes felelősséget vállal a munka végső tartalmáért, beleértve a matematikai anyag kidolgozását, a kódot és a felhasznált hivatkozásokat.

Hivatkozások

- [1] M. Batista and A. O. Martins. “Redes Neurais no Ensino Básico”. *Professor de Matemática Online*, v. 10, n. 4, p. 454–481, 2022. doi: 10.21711/2319023x2022/pmo1032
- [2] K. Faceli, A. C. Lorena, J. Gama, T. A. de Almeida, and A. C. P. L. F. de Carvalho. *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina*. 2. kiadás. Rio de Janeiro: LTC, 2021.
- [3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. Elérhető: <http://www.deeplearningbook.org>
- [4] C. R. Harris, K. J. Millman, S. J. van der Walt et al. “Array programming with NumPy”. *Nature*, v. 585, p. 357–362, 2020. doi: 10.1038/s41586-020-2649-2
- [5] S. Haykin. *Redes Neurais: Princípios e Prática*. Fordította: Paulo Martins Engel. 2. kiadás. Porto Alegre: Bookman, 2000.
- [6] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. *Nature*, v. 521, p. 436–444, 2015. doi: 10.1038/nature14539
- [7] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. Elérhető: <https://neuralnetworksanddeeplearning.com>
- [8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. *Nature*, v. 323, p. 533–536, 1986. doi: 10.1038/323533a0

Americo Cunha Jr

NeuralMath

Matematika · neurális hálózatok · Python

Olvassa más nyelveken

	العربية · ar	ja · 日本語
bn · বাংলা		ko · 한국어
cs · Čeština		nl · Nederlands
de · Deutsch		pl · Polski
el · Ελληνικά		pt · Português (eredeti)
en · English		ro · Română
es · Español		ru · Русский
	فارسی · fa	sv · Svenska
fr · Français		tr · Türkçe
hi · हिन्दी		uk · Українська
hu · Magyar		
id · Bahasa Indonesia		vi · Tiếng Việt
it · Italiano		zh · 简体中文
		اردو · ur



<https://neuralmath.org>

DOI: 10.5281/zenodo.23212381 · CC-BY-4.0 · Verzió 1.0