Függvényapproximáció neurális hálók segítségével

BSc szakdolgozat
2020. május

Bede Ábel Márk
matematika szakos hallgató

Témavezető:

Izsák Ferenc, docens
Alkalmazott Analízis és Számításmatematikai Tanszék

Eötvös Loránd Tudományegyetem, Természettudományi Kar

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Izsák Ferencnek, aki felkeltette érdeklődésemet a téma iránt és akihez mindig fordulhattam a felmerülő kérdéseimmel. Köszönettel tartozom a türelméért, illetve a rengeteg időért és energiáért, amit rám és a munkámra fordított. Szakértő iránymutatása és hasznos tanácsai tették lehetővé, hogy a dolgozat a jelenlegi formájában megvalósulhasson.
Köszönöm a Mindenhatónak, akinek kegyelméből létrejött a dolgozatom. Erőt adott a nehezebb időszakokban, és mindig támaszkodhattam rá.

Jelölések

$x; [x(k)]$ Bemenetek vektora [a tanító halmaz k-adik eleme esetén]

$d; [d(k)]$ Elvárt kimenetek vektora [a tanító halmaz k-adik eleme esetén]

w Súlyok vektora

b Torzítás

v Összegző csomópont értéke

φ Aktivációs függvény

$y; [y(k)]$ A perceptron által produkált kimenet [a tanító halmaz k-adik bemenetére]

Φ Neurális háló, mint paraméterhalmaz

L Rétegek száma a neurális hálóban

N_i Az i-edik réteg kimeneti dimenziója, az $i+1$-edik réteg bemeneti dimenziója

$M_j(\Phi)$ A (nemnulla) súlyok száma a j-edik rétegben

$\|A\|_{l_0}$ Az A mátrix nemnulla elemeinek száma

$R(\Phi)$ A Φ neurális háló realizációja

$N(\Phi)$ A Φ neurális háló neuronjainak száma

$f(x, w)$ A neurális hálózat realizációja az x bemenetre és w paraméterhalmazra

$L(f(x, w), d)$ Veszteségfüggvény: a realizáció és az elvárt d kimenet egy függvénye

$\mathcal{E}(x, w)$ Az x ponthoz tartozó hiba a w paraméterhalmazon

$R(w)$ Kritériumfüggvény a w paraméterhalmazon

$R_e(w)$ Várható kockázat a w paraméterhalmazon

C_l A neurális háló l-edik rétegében lévő neuronok halmaza

$\mathcal{T}$ Az $[a, b]$ intervallum n elemű felosztása

$S_{\boldsymbol{p}}(I, \mathcal{T})$ A $\mathcal{T}$ elemein $\mathbf{p}$ rendben polinomiális abszolút folytonos függvények tere

$S_p^n(I)$ n elemű szabad alappontú felosztáson p fokszámú spline függvények tere

Tartalomjegyzék

1. fejezet

Bevezetés

Az elmúlt években széles körben elterjedt a neurális hálózatok használata. Ennek az a magyarázata, hogy rendkívül sokszínű a segítségükkel megoldható műszaki és tudományos problémák halmaza. Ezen problémák között nem mindig egyértelmű az összefüggés, azonban javarészt visszavezethetőek a következő két alapvető feladatra: osztályozásra és függvényközelítésre. A növekvő népszerűségük abban is rejlik, hogy sok esetben hatékonyabban működnek, mint a hagyományos algoritmikus eljárásokkal dolgozó eszközök.

A neurális hálók legfőbb alkalmazási területe a gépi tanulás, vagyis a hálóknak tanuló rendszerként történő gyakorlati alkalmazása. Csak a mesterséges tanulási területet említve, használhatók többek között parciális differenciálegyenletek numerikus megoldására, de akár beszéd és karakterfelismerésre, illetve jel- és képfeldolgozásra is. Az ilyen felismerési feladatokon kívül is sok olyan probléma merül fel a mindennapi életben, melyre nem adható ésszerű műveletigényű algoritmikus megoldás. A gyakorlati megfigyelésekre vonatkozó szabályszerűségek feltárását például tipikusan célszerű neurális hálókkal megvalósítani. Ezek többnyire függvényközelítési (*approximációs*), illetve függvényillesztési (*interpolációs*) feladatok.

Függvényinterpoláció során adott ponthalmazra szeretnénk előre meghatározott feltételeknek megfelelő függvényt illeszteni. Ekkor olyan függvényt akarunk megadni, amely adott pontokban adott értékeket vesz fel. Praktikusan a feltételek a feladat numerikus megoldhatósága érdekében vannak kiszabva, azonban gyakran ezeknek megfelelő függvény egyáltalán nem is illeszkedhet a megadott ponthalmazra. Gyakorlatban ez például a megfigyeléseink természetes zaja miatt többször előfordul. Ez esetben a feltételeknek megfelelő függvények halmazának akarjuk meghatározni az adott pontokat legjobban közelítő elemét. Függvényközelítésről beszélünk, amikor egy bonyolultabb, nehezen kiértékelhető függvényt szeretnénk jobban kezelhető, ismert függvények segítségével, ezek kompozíciójával leírni. Jellemzően ilyenkor nem várjuk el, hogy a közelítésünk értéke minden pontban megegyezzen az eredetiével, hanem valamilyen hibakorlátot engedünk meg például az értékek abszolút eltérésében. Előfordul, hogy megfelelő mennyiségű paraméter felhasználásával tetszőlegesen jó közelítést elérhetünk. Azonban ez nem cél, mert a kiértékelés, illetve a megfelelő paraméterek megtalálása azok számának növekedésével egyre több időt vehet igénybe. Túl alacsony hibakorlát megkövetelésével éppen a közelítés előnyeit veszítjük el: a gyorsabb, egyszerűbb számításokat. A közelítésnek számítástechnikai alkalmazásokban van nagy jelentősége, ahol a teljesítmény az elsődleges szempont, és a tényleges megoldásra nincs szükség, vagy kikövetkeztethető a közelítésből.

Ebben a dolgozatban a neurális hálók és az imént leírt függvényillesztési és közelítési feladatok kapcsolatát vizsgáljuk meg. A 2. fejezetben részletesebb leírást adunk a neurális hálók építőelemeiről, szerkezetéről és működésének alapjairól. A 3. fejezetben bemutatjuk és összehasonlítjuk a neurális hálók betanításának néhány lehetséges módját. Kitérünk több itt szem előtt tartandó szempontra és a hatékony neurális hálók építésére gyakorlatban sokszor alkalmazott technikákra. Ezt követően azt vizsgáljuk meg, milyen lehetőségeket nyújtanak a neurális hálózatok az említett approximációs és interpolációs problémák megoldására, kitérve az ilyen jellegű megközelítés hatékonyságának elemzésére is. A 4. fejezetben valós függvények neurális hálókkal történő szakaszonként lineáris közelítésének lehetőségét mutatjuk be részletesen, saját programozási eredményekkel is kiegészítve az elméletet. Az elmélet alapján megfogalmazunk egy, a közelítés konvergencia-sebességére vonatkozó tételt is. Az 5. fejezetben tárgyaljuk a többdimenziós függvények közelítésére vonatkozó lehetőségeket. Először kitérünk a vektorértékű függvényeket közelítő hálózat egy lehetséges konstrukciójára, majd megvizsgáljuk a többváltozós függvények esetét is. Itt szintén kimondunk a közelítés konvergencia-sebességére vonatkozó tételt, melyet be is bizonyítunk, és gyakorlatban is ellenőrzünk. A dolgozat végén megtalálhatóak az elméleti munka gyakorlati kiegészítése céljából MATLAB-ban megírt saját programok, melyek magyarázata kommentek formájában a programok kódjaiban kapott helyet.

2. fejezet

Neurális hálók általában

A neurális hálózatok meghatározása több módon is történhet. A neurális hálózat lényegében egy olyan számítási mechanizmus, amelynek alapjául mintaként az idegrendszer szolgált mind szerkezetét, mind működését tekintve ([8]). Az elnevezését is ennek köszönheti. Közismert, hogy a biológiai rendszerek milyen hatékonyan oldanak meg bizonyos problémákat. Gondolhatunk például arra, milyen sikeresen működik az emberi agy képek és jelek felismerése és értelmezése során. Ez motivációként szolgál egy hasonló működésű matematikai struktúra létrehozására. Azonban a neurális hálózatoknak nem céljuk az idegrendszer pontos matematikai modellezése, csak az ezekről szerzett általános megfigyeléseket vették át. Örököltek olyan alapvető tulajdonságokat, mint például a tanulásra való képesség, illetve szerkezetet tekintve a sok kisebb, egyszerű alapegységből való felépítés és az ezek között fennálló összeköttetési rendszer.

Sokszor előfordul a gyakorlati problémák megoldása során, hogy adottak egy jelenséggel kapcsolatos méréseink, megfigyeléseink, de nem rendelkezünk pontos matematikai modellel, amely az adott jelenséget jól leírná, vagy pedig a modellünk teljesen alkalmatlan számításokra. A neurális hálónak sem célja a vizsgált jelenséget pontosan modellezni, hanem a jelenséget fekete dobozként kezelve próbál bizonyos törvényszerűségeket, összefüggéseket felfedezni a megfigyelések, vagyis szolgáltatott adatok halmazán. Tehát a hálózat a jelenséget nem tárja fel, kizárólag a bemeneti adatainkra (*input*) és az ezekre adott, a jelenségből származó kimeneti értékekre (*output*) támaszkodik.

Ennek előnye, hogy a problémák ilyen jellegű megközelítéséhez csak adatokra van szükség a vizsgált jelenségről, nem kell azt pontosan és átfogóan ismerni vagy annak egy számítógépes absztrakcióját leprogramozni. Hátránya, hogy így a jelenség mögöttes felépítéséről és működéséről nem szerezhetünk pontos képet, a hálózatunk nem foglalja magában annak alapvető ismertetőjegyeit, nem ”hasonlít” a jelenségre. Akkor tekintjük a hálózatunkat jónak, ha tetszőleges inputra ugyanolyan vagy nagyon hasonló outputot eredményez, mint a jelenség.

2.1. Neurális hálók működése

A neurális hálók működése alapvetően három fázisra bontható fel. Ezek a megfelelő modell megalkotása, illetve a hálózat betanítása és használata.

Az alkalmazandó modellt annak megfelelően választjuk ki, hogy milyen problémát akarunk megoldani a hálózat segítségével. Fontos megjegyezni, hogy a neurális hálózatoknak több típusa van. Mi a továbbiakban kizárólag a *többrétegű perceptron* modellel foglalkozunk. Ezen belül további pontosításra szorul hálózatunk elvárt működése, szerkezete. Ahhoz, hogy a megoldandó feladatra megfelelően illeszkedjen a hálózatunk, konkrét specifikációra is szükség van.

A hálózatot annak betanítása során illesztjük hozzá a rendelkezésre álló adatainkhoz. Ennek szintén több lehetséges formája van, mi az *ellenőrzött tanulás* (supervised learning) módszerrel foglalkozunk, amiről a 3. fejezetben adunk részletesebb leírást. Ekkor a hálózat betanítását *tanító pontok* segítségével végezzük. Ezek konkrét $x(1), x(2), \ldots, x(n)$ input adatok a hozzájuk tartozó ismert jelenség által kibocsátott $d(1), d(2), \ldots, d(n)$ kimenetekkel. Ezek halmaza a *tanító halmaz*. A hálózat betanítása alatt azt értjük, hogy a tanító halmaz elemei alapján megválasztjuk a hálózat paramétereit valamilyen számítási mechanizmus segítségével. Ezt akkor tekintjük sikeresnek, ha a hálózat az összes ismert input-output hozzárendelést jól közelíti. Itt számtalan numerikus matematikai módszer alkalmazható, amelyek segítségével a hálózat kimenetei és az elvárt kimenetek egy meghatározott függvényét, a *veszteségfüggvényt* minimalizálni tudjuk. Ez a függvény jellemzi azt, milyen értelemben szeretnénk közeli értékeket kapni a tanító halmaz elemeihez.

A betanítás végeztével azzal a feltételezéssel élünk, hogy a hálózat által szolgáltatott eredmény tetszőleges inputra jó approximáció a jelenség ismeretlen output értékére. Tehát a hálózat által adott értéket fogjuk igaznak tekinteni minden esetben.

2.2. Felépítés

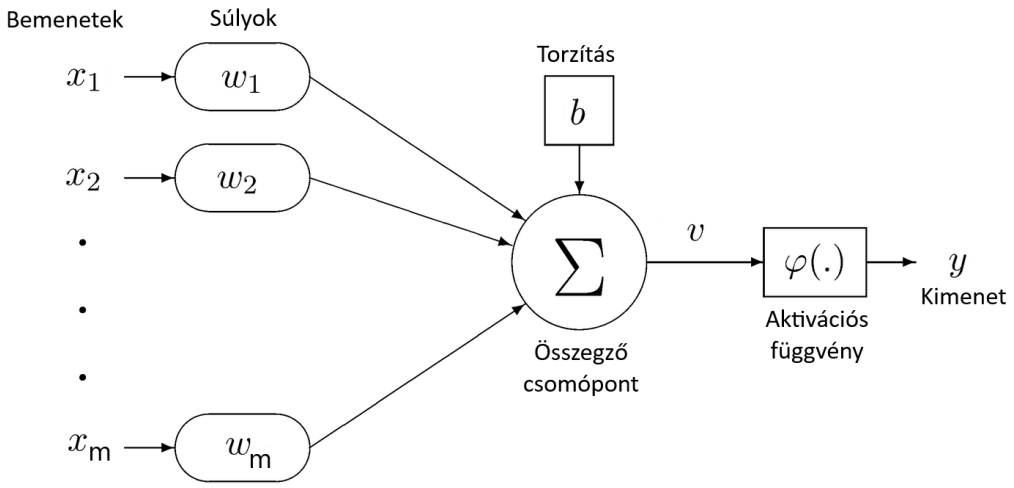
A neurális hálók alapvető információfeldolgozó egysége a perceptron, más néven neuron, amely a többrétegű perceptron-modell fundamentális építőeleme. A legegyszerűbb szerkezetű neurális hálót egyetlen perceptron alkotja. Ennek leírásánál a [8] munkát követjük.

2.2.1. A perceptron

A következőekben elemezzük a perceptron felépítését, melyet a 2.2.1 ábra is szemléltet. Minden vektort oszlopvektornak tekintünk.

- Bemenet (input): Ez az m számból álló bemenő jel. Úgy tekintünk rá, mint $[x_1, \ldots, x_m]$ m dimenziós vektor.
- Súlyok (synaptic weights): A $[w_1, \ldots, w_m]$ vektort nevezzük súlyvektornak, elemeit súlyoknak. Ezek jellemzően közelítései valamilyen kívánt ismeretlen értékeknek. Pont azok meghatározása lesz a feladat a hálózat betanításánál.

- Torzítás (bias): Egy b-vel jelölt skalármennyiség, mely a súlyokhoz hasonlóan szintén meghatározandó.
- Összegző csomópont (summing junction): A bemenetnek a következő súlyozott összege: $v = b + \sum_{i=1}^{m} w_i x_i$.
- Aktivációs függvény: Egy φ-vel jelölt függvény, amely az előző összeget a megoldandó problémának megfelelően tovább transzformálja. Ezt később részletesebben tárgyaljuk.
- Kimenet (output): A bemenethez a perceptron által rendelt y érték, amelyet a következőképp kaphatunk meg: $y = \varphi(v)$.



2.2.1. ábra. A perceptron felépítése.

2.2.2. A többrétegű perceptron

A továbbiakban jellemezzük a többrétegű perceptront (multi layer perceptron, MLP). Mivel ez az egyetlen vizsgált modellünk, az egyszerűség kedvéért a továbbiakban neurális hálóként hivatkozunk rá. A gyakorlatban is ezt alkalmazzák legtöbbször. Megkülönböztetjük a neurális hálót, mint paraméterek halmazát és a háló realizációját. A háló realizációja egy, a hálónak megfeleltethető függvény, amelyet a paraméterek által meghatározott affin lineáris transzformáció és a φ aktivációs függvény többszöri alkalmazásával kaphatunk meg. A definícióinkban a [15] cikket fogjuk követni.

2.2.1 Definíció. *Legyen $L \in \mathbb{N}$ és legyen $N_0, N_1, ..., N_L \in \mathbb{N}$. Neurális hálónak nevezzük a következő Φ-vel jelölt mátrix-vektor sorozatot: $\Phi = ((A_1, b_1), (A_2, b_2), \ldots, (A_L, b_L))$, ahol $A_l \in \mathbb{R}^{N_l \times N_{l-1}}$ és $b_l \in \mathbb{R}^{N_l}$, $(l = 1, \ldots, L)$.*

2.2.2 Definíció. *Legyen az előző jelölés mellett adott Φ neurális háló.*

- *A Φ* rétegeinek *nevezzük az egyes (A_l, b_l) mátrix-vektor párokat, $(l = 1, \ldots, L)$.*
- *Ennek megfelelően legyen a* rétegek száma *vagy a háló* mélysége $L(\Phi) := L$.
- *Az utolsó réteget* kimeneti, *a többit pedig* rejtett *rétegnek nevezzük. Az első rétegre* bemeneti *rétegként is hivatkozunk.*
- *N_i-t nevezzük az i-edik* réteg kimeneti dimenziójának, *illetve az $i+1$-edik* réteg bemeneti dimenziójának, *$(i = 1, \ldots, L-1)$*
- Φ bemeneti dimenziója *legyen N_0, a* kimeneti dimenziója *pedig N_L.*
- *Legyen a* súlyok száma a *j-edik* rétegben*:*

  $$M_j(\Phi) := \|A_j\|_{l^0} + \|b_j\|_{l^0},$$

  ahol $\|M\|_{l^0}$ az M mátrix nemnulla elemeinek számát jelöli.
- *A háló súlyainak száma vagy másképp a* mérete *legyen $M(\Phi) := \sum_{j=1}^{L} M_j(\Phi)$.*

2.2.3 Definíció. *Legyen $\varphi\colon \mathbb{R} \to \mathbb{R}$ adott aktivációs függvény. Ekkor a Φ neurális hálóhoz tartozó realizációnak nevezzük azt az $R(\Phi)$ függvényt, amelyre*

$$R(\Phi) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}, \quad R(\Phi)(x) := x_L,$$

ahol x_L (rekurzív módon) a következőképpen számolható:

$x_0 := x$

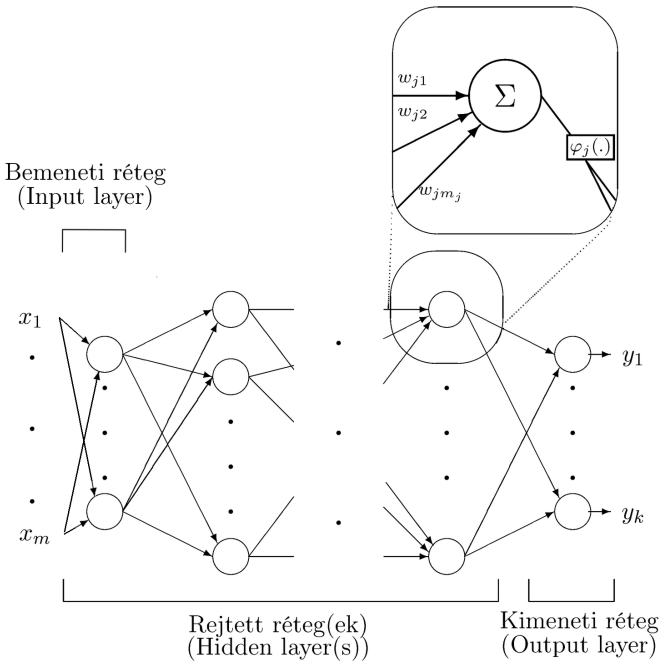
$x_l := \varphi(A_l x_{l-1} + b_l)$ *minden* $l = 1, \ldots, L-1$ *esetén*

$x_L := A_L x_{L-1}$.

Itt a φ függvényt úgy értelmezzük, hogy komponensenként hat a vektorértékű bemenetre. A realizáció függ a φ függvénytől, az egyszerűség kedvéért azonban ezt nem jelöljük. A jelenlegi definícióban az aktivációs függvényt nem alkalmazzuk a kimeneti réteg összegző csomópontjain. Ez azonban természetesen megtehető, ha szükség van a kimeneti adatok transzformációjára.

2.2.4 Megjegyzés. *A realizáció definíciójából megfigyelhető, hogy az A_l mátrixok egyes sorai a b_l vektorok megfelelő elemeivel és az aktivációs függvénnyel egy perceptront (vagy másképp neuront) alkotnak. Ezért a Φ realizációjának ilyen módon történő definiálásával nem csupán több rétegbe helyezünk el egy-egy perceptront, hanem egy rétegen belül többet egymás mellé kapcsolva használunk. Tehát az előző jelölésekkel N_j (a j-edik kimeneti dimenzió) éppen a j-edik réteg neuronjainak a száma is egyben. Ennek megfelelően Φ neuronjainak száma $N(\Phi) := \sum_{j=1}^{L} N_j$. Jelölje ezentúl C_l az l-edik réteg neuronjainak halmazát.*

A neurális hálózat előbbi konstrukcióját **előrecsatolt hálózatnak** (*feedforward network*) nevezzük. Ennek szerkezetét mutatja be a 2.2.2 ábra. A hálózatban csak és kizárólag az egymás melletti rétegek között van kommunikáció olyan értelemben, hogy egy réteg outputja a tőle jobbra elhelyezkedő réteg inputjaként szolgál. Az ilyen kapcsolatokat jelöljük az ábrán nyilakkal. Ezen az összeköttetési rendszeren keresztül áramlik a jel balról jobbra. Ez mutatja az előrecsatolt hálózat szerkezetének fő ismertetőjegyeit: egy rétegen belül, illetve távolabb eső rétegek között nincsenek kapcsolatok, viszont egy neuron összeköttetésben van minden szomszédos rétegbe eső neuronnal. Ez persze nem jelenti azt, hogy bármely ilyen neuronpár között kell is információnak áramlani. Ez megszüntethető, ha a receptor (vagyis az információt bemenetként feldolgozó) neuron rétegéhez tartozó mátrix adott sorvektorának megfelelő paraméterét kinullázuk.



2.2.2. ábra. A többrétegű perceptron felépítése [8] alapján.

2.3. Az aktivációs függvény

Ezt a függvényt használják a neurális hálózat realizációja során arra, hogy az összegző csomópontok értékeit a problémának megfelelő intervallumba transzformálják, ezért transzfer függvénynek is nevezik. Enélkül a betanítási folyamat könnyen megbénulhat a különböző divergens értékű neuron paramétersorozatok miatt. A függvény az aktivációs jelzőt szintén biológiai analógia alapján kapta. Absztrakt módon ez reprezentálja az egyes ideg neuronok akciós potenciálját. Valójában azonban ez a potenciál kétféle állapotú lehet csak: a neuron kisül (”tüzel”), ha a belé érkező jelek összerőssége meghalad egy bizonyos értéket. Ellenkező esetben az ideg nyugalmi állapotban marad, és nem generál kimenetet. Fontos azt is megjegyezni, hogy aktivációs függvények nélkül a realizáció csupán affin lineáris transzformációk komponálása, ezért maga is mindig affin lineáris lenne, amely nyilván nem alkalmas általánosan problémamegoldásra.

Az alkalmazások során az aktivációs függvények leggyakrabban jobbról folytonos eloszlásfüggvények: monoton növő függvények, melyek határértéke $-\infty$-ben 0, $+\infty$-ben pedig 1; vagy ilyen függvények transzformáltja úgy, hogy az értékkészletük a $[-1, 1]$ intervallumba essen. Az ilyen transzformáltakat a szimmetrikus jelzővel illetjük a 0-ra szimmetrikus értékkészletük miatt. Azonban az aktivációs függvényeknek ezeken belül és ezek mellett is több típusa van, ezeket a függvények általános jellemzői alapján csoportosíthatjuk. A következőekben mutatunk példákat az egyes függvénytípusokra [16] nyomán. Mostantól m legyen a függvényre vonatkozó meredekségi paraméter.

- Lineáris: A kevésbé bonyolult problémáknál alkalmazható, amikor a rejtett neuronok kimeneti értékeinek intervallumát nem szükséges korlátozni. Például:

$$\varphi(x) = mx.$$

- Szakaszonként lineáris: Ezek segítségével már a feladatnak megfelelően korlátozhatjuk a rejtett rétegek értékkészletét, és a realizáció sem feltétlen affin lineáris. Nézzük erre a gyakorlatban legtöbbet alkalmazott példát:

  2.3.5 Definíció. *ReLU függvénynek (Rectified Linear Unit) nevezzük a következő függvényt:*

$$\varphi : \mathbb{R} \to \mathbb{R}, \quad \varphi(x) = \max\{0, x\}.$$

  Ezt használva aktivációs függvénynek a neurális hálót ezentúl ReLU hálózatnak nevezzük.

- Szigmoid: Olyan eloszlásfüggvényeket értünk alatta, melyek grafikonja S alakú. Legtöbbször ezeket érdemes használni hatékonyságuk miatt, amikor a kimeneti adathalmaz értékei a $[0, 1]$ intervallumban mozognak (vagy ide transzformálhatóak). Példa ilyen függvényre a *logisztikus függvény*:

$$\varphi(x) = \frac{1}{1 + e^{-2mx}}.$$

- Szimmetrikus szigmoid: Egy szigmoid típusú függvény transzformáltja szimmetrikus aktivációs függvénnyé. Szintén gyakori hatékonysága miatt. Például:

$$\varphi(x) = \tanh(mx) = \frac{2}{1 + e^{-2mx}} - 1.$$

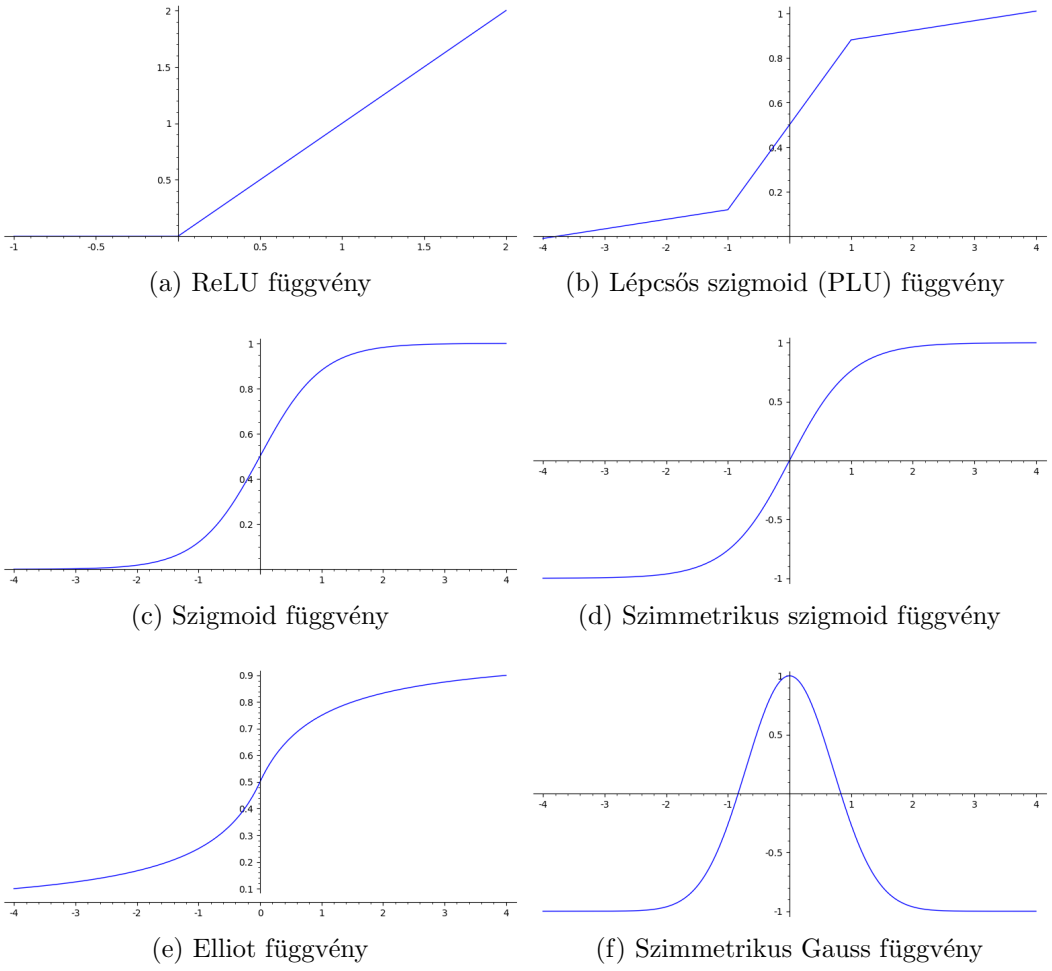
- Lépcsős (szimmetrikus) szigmoid: Szakaszonként lineáris közelítései az előzőeknek. A betanítás általában kevésbé pontos, de gyorsabb.

- (Szimmetrikus) Gauss: Finomabb irányítást tesznek lehetővé a neuron értékkészletének meghatározásában. Példák rendre:

$$\varphi(x) = e^{-(mx)^2}, \ (\varphi(x) = 2e^{-(mx)^2} - 1).$$

- (Szimmetrikus) Elliot: A szigmoid függvénytípusok más módon történő közelítései, melyeket a gyorsabb betanítás érdekében alkalmaznak (ld. [7]). Példák rendre:

$$\varphi(x) = \frac{\frac{1}{2}mx}{1+|mx|} + \frac{1}{2}, \quad \left(\varphi(x) = \frac{mx}{1+|mx|}\right).$$



(a) ReLU függvény

(b) Lépcsős szigmoid (PLU) függvény

(c) Szigmoid függvény

(d) Szimmetrikus szigmoid függvény

(e) Elliot függvény

(f) Szimmetrikus Gauss függvény

2.3.3. ábra. Aktivációs függvények grafikonjai.

A megoldandó problémának megfelelő aktivációs függvény megválasztása alapvető fontosságú hatékony neurális háló készítéséhez. Azonban fontos megjegyezni, hogy a gyakorlati eredmények a különböző függvénytípusok teljesítményének tesztelésében azt mutatják, hogy ha egy bizonyos függvény segítségével egy hálózat sikeresen betanítható, akkor több más (a feladat szempontjából megfelelő) típus alkalmazása sem okoz drasztikus különbséget a folyamat pontosságában és lefutási idejében. Pontos mérési adatok megtalálhatóak a [16] munkában.

Nem szükséges megkövetelni, hogy a neurális háló minden perceptronja ugyanazt az aktivációs függvényt használja. Definiálhatunk például rétegenként, sőt akár perceptrononként különbözőt is, ám erre általában nincsen szükség.

2.4. A veszteségfüggvény és a kritériumfüggvény

A betanítási folyamat során úgy módosítjuk a neurális hálózat paramétereit, hogy az jól approximáljon, vagyis konkrét bemenetekre kimeneti értékei "közel" legyenek az adathalmazunk alapján elvárt értékekhez. Ez így azonban nem jóldefiniált, szükségünk van még egy, a közelítés minőségét jellemző mértékre is. Ez a háló realizáció adott x inputra adott válaszának és az erre elvárt d kimenetnek egy nemnegatív értékű függvénye. Ezt nevezzük veszteségfüggvénynek (*loss function*), illetve néha hibafüggvénynek vagy költségfüggvénynek. Ennek megválasztása nemcsak az optimum helyét befolyásolja, hanem az eljárásunk konvergenciájának sebességét és feltételeit is. A leírásunkban az [1] munka jelöléseit követjük. Fontos hangsúlyozni, hogy a realizáció kiértékelésekor kapott eredmény nemcsak a bemenettől függ, hanem a hálózatban aktuálisan használt paraméterektől is, amelyek a tanítás során folyamatosan változnak. Ezért a betanítás során a realizáció nemcsak a bemenetnek, hanem az aktuálisan használt paraméterhalmaznak a függvénye is, amelyet w-vel jelölünk. Ennek megfelelően jelölje tehát $f(x, w) := R(\Phi)(x)$ a realizációt adott w paraméterhalmazon. Ennek segítségével a veszteségfüggvényt pedig jelölje:

$$L(f(x, w), d), \quad L : \mathbb{R}^{N_L} \times \mathbb{R}^{N_L} \to \mathbb{R}_0^+.$$

A veszteségfüggvényt gyakran az elvárt és produkált kimenetek különbségének, a *hibának* függvényeként definiálják az előbbi általánosabb megközelítés helyett. Jelölje a w paraméterhalmazon vett x ponthoz tartozó (lokális) hibát:

$$\mathcal{E}(x, w) := d - f(x, w).$$

Ekkor a veszteségfüggvényre hivatkozhatunk az egyszerűbb $L(\mathcal{E}(x, w))$ jelöléssel is.
A szakirodalom által vizsgált veszteségfüggvények jellemzően konvex függvények. Erre nyilván azért van szükség, hogy biztosítva legyen, hogy elérhető egy globális minimum is. Tekintsünk néhány példát a gyakran előforduló veszteségfüggvényekre.

- Regressziószámítási feladatoknál sokszor használják a hiba négyzetes függvényét:

$$L(\mathcal{E}(x, w)) = \frac{1}{2}\|\mathcal{E}(x, w)\|^2.$$

- Ennél bizonyos esetekben jobb konvergencia sebesség érhető el a hálózat betanításakor az ún. *ε-intenzív hiba függvénnyel* (ld. [18]):

$$L(\mathcal{E}(x, w)) = \max\{\|\mathcal{E}(x, w)\| - \varepsilon, 0\}.$$

- Osztályozás alapú feladatok megoldásához bináris esetben például két osztályba akarjuk sorolni az elemeket, ezeket az osztályokat 0, illetve 1 értékekkel jelöljük. Ekkor praktikusan $f(x, w)$ annak a valószínűsége, hogy az x-re elvárt válasz 1. Látszik, hogy olyan aktivációs függvényre van szükség, amellyel az adatokat az utolsó lépésben a $[0, 1]$ intervallumba lehet transzformálni (tehát azt az utolsó lépés során is alkalmazni kell az adatokon). Ez esetben népszerű veszteségfüggvény a *keresztentrópia függvény*, melynek képlete két osztály esetében:

$$L(f(x, w), d) = -(d \ \ln(f(x, w)) + (1 - d) \ \ln(1 - f(x, w))).$$

- Másik megközelítésben a két osztályt ± 1-el jelölve jól alkalmazható az ún. *Hinge* veszteségfüggvény:

$$L(f(x,w),d) = \max\{1 - d \cdot f(x,w), 0\}$$

A betanított rendszerünk megfelelő minősítéséhez információra van szükségünk annak általánosító képességéről, vagyis az ismeretlen inputok közelítésének mértékéről is. Az elméleti cél a betanítás során tehát az ún. *várható kockázatot* minimalizálni:

$$R_e(w) := \mathbb{E}_{x,d}[L(f(x,w),d)] = \int\limits_{\mathbb{R}^{N_0} \times \mathbb{R}^{N_L}} L(f(x,w),d)\ \mathrm{d}\mathbb{P}(x,d)$$

ahol $\mathbb{P}(x,d)$ a jelenséget leíró adathalmazt jellemző eloszlás. Problémát okoz, hogy a veszteségfüggvényt csak a T tanító halmaz pontjaiban tudjuk kiértékelni, hiszen d csak ott ismert. Az adatokat általánosan jellemző eloszlás pedig ismeretlen. Ezért alkalmazzák a *tapasztalati kockázatot*, vagy másképp *kritériumfüggvényt* (*criterion function*), amely a veszteségfüggvény ismert értékeinek átlaga, vagyis a veszteség tapasztalati várható értéke:

$$R(w) := \frac{1}{|T|} \sum_{(x,d) \in T} L(f(x,w),d).$$

3. fejezet

Neurális hálók betanítása

Az ellenőrzött tanulás módszere során a célunk tehát az, hogy miután megválasztottuk a megfelelő struktúrát, a tanító halmaz elemeit feldolgozva kiszámítsuk a neurális hálózat előbbiekben bevezetett R kritériumfüggvényét minimalizáló w paraméterhalmazt. Regressziószámítási feladatoknál jellemzően ezt a betanítási módszert szükséges alkalmazni. Megjegyzendő, hogy létezik az ún. *nem ellenőrzött tanulási* módszer is neurális hálók betanítására. Ekkor a hálózatba nem érkeznek bemeneti adatokhoz tartozó elvárt kimenetek. A hálózat kizárólag a bemeneteket feldolgozva próbál meg összefüggéseket felfedezni az adathalmazon. Függvényközelítési és illesztési feladatoknál ez a módszer nem alkalmazható, ezérti itt kizárólag az ellenőrzött tanulási algoritmusok leírására szorítkozunk.

3.1. A perceptron tanítása

A többrétegű perceptron tanítási elvének könnyebb megértéséhez célszerű először megvizsgálni lehetséges algoritmusokat az egyszerű perceptron tanítására. Sok esetben az itt szerzett megfigyelések felhasználhatóak az összetettebb hálózatok leírásánál is. Vizsgáljunk meg néhány lehetséges példát a perceptron betanítására [8] mintájára. Tekinstsük a következő problémát: adottak $(x(i), d(i)) \in \mathbb{R}^m \times \mathbb{R}$ pontok, $(i = 1, \ldots, n)$, ezekre illesszünk $g : \mathbb{R}^m \to \mathbb{R}$ lineáris függvényt a hiba négyzetes függvényét alkalmazva veszteségfüggvényként. Ez a feladat az ún. lineáris regresszió. Mivel lineáris függvény illesztése a cél, ezért ez alkalommal a $\varphi = \mathrm{id}$ aktivációs függvényt használjuk. Egyszerűsítés céljából írjuk felül a perceptron modellhez használt $x(i)$ és w jelöléseket a következőképpen: $x(i) := [1, x_1(i), \ldots, x_m(i)], w := [b, w_1, \ldots, w_m]$ oszlopvektorok. Ekkor teljesül a perceptron kimenetére, hogy

$$y(i) = b + \sum_{j=1}^{m} w_j x_j(i) = \sum_{j=0}^{m} w_j x_j(i) = w^T x(i) \quad \forall i \in \{1, \ldots, n\}.$$

3.1.1. A ridge regresszió

Legyenek az X mátrix sorvektorai rendre az $x(i)$ vektorok. Írjuk fel ennek segítségével a kritériumfüggvényt!

Felhasználva, hogy a veszteségfüggvény a négyzetes hibafüggvény, adódik az összefüggés:

$$R(w) = \frac{1}{2n}\sum_{i=1}^{n}(\mathcal{E}(x(i), w))^2 = \frac{1}{2n}\sum_{i=1}^{n}(w^T x(i) - d(i))^2 = \frac{1}{2n}\|Xw - d\|^2.$$

Mivel ez nemnegatív, w-re nyilván optimális, ha $Xw = d$. Ezt módosítva kapjuk a következő normálegyenletet: $X^T Xw = X^T d$, ahonnan invertálhatóságot feltételezve kapjuk optimális w-re a képletet:

$$w = (X^T X)^{-1} X^T d$$

Azonban a fenti mátrix akkor és csak akkor invertálható, ha a pontok egy hipersíkon helyezkednek el, de általában nem ez a helyzet. Ekkor az optimális megoldást

$$w = (X^T X + \lambda I)^{-1} X^T d$$

alakban szokták keresni ennél a modellnél valamilyen λ pozitív számra. Ezt az eljárást nevezik *ridge regressziónak*.

3.1.2. A gradiens módszer

Gyakorlatban sokszor egy másik megközelítés bizonyul hatékonynak. Ismert tény, hogy egy többváltozós függvény a gradiensvektorával ellentétes irányban csökken a leggyorsabban. Tehát a gradiens vektor mentén elmozdulva negatív irányban közelebb kerülhetünk a lokális minimumhoz, ez az ún. *gradiens módszer*. Ennek megfelelően az aktuálisan használt w súlyvektort a kritériumfüggvény lokális gradiense szerint lecseréljük w_{next}-re, melyet a tanító halmaz feldolgozásával határozhatunk meg. Ezt ismételgetve lépkedünk diszkrét módon a függvény szélsőértékének irányába. Így $R(w_{\text{next}})$ érték praktikusan jobb közelítése a szélsőértéknek. Most vizsgáljuk meg a gradiens kiszámításának egy lehetséges módszerét. A leírásunkban a [8] munka logikáját követjük, az ott leírtak megfelelő átdolgozásával az általános gradiens módszer esetére. Nevezzük k-adik lokális hibának:

$$\mathcal{E}(k) := \mathcal{E}(x(k), w) = d(k) - w^T x(k).$$

Ezzel a jelöléssel a gradiens módszer lépése általánosan:

$$w_{\text{next}} := w - \lambda \nabla R(w), \quad \nabla R(w) = \frac{1}{n}\sum_{k=1}^{n}\frac{\partial L(\mathcal{E}(k))}{\partial w}, \tag{3.1.1}$$

ahol $\lambda > 0$ tanulási paraméter. Ehhez tehát a veszteségfüggvény gradiensére van szükségünk a tanítóhalmaz pontjaiban. A veszteségfüggvényre

$$L(k) := L(\mathcal{E}(k)) = \frac{1}{2}(\mathcal{E}(k))^2, \quad \frac{\partial L(k)}{\partial w} = \frac{\partial L(k)}{\partial \mathcal{E}(k)} \cdot \frac{\partial \mathcal{E}(k)}{\partial w}.$$

Megvizsgálva a hiba és a veszteségfüggvény alakját kapjuk a parciális deriváltakra, hogy:

$$\frac{\partial L(k)}{\partial \mathcal{E}(k)} = \mathcal{E}(k), \quad \frac{\partial \mathcal{E}(k)}{\partial w} = -x(k).$$

Tehát végeredményül

$$w_{\text{next}} := w + \frac{\lambda}{n}\sum_{k=1}^{n}\mathcal{E}(k)x(k).$$

3.1.6 Megjegyzések. *A bevezetett λ értéke annak a vektornak a hosszát határozza meg, amely mentén elmozdulunk a szélsőérték irányába. Ez alapvetően befolyásolja az eljárás konvergenciasebességét. Nagyobb értékre a konvergencia gyorsabb lehet, azonban túl nagy értékre például az optimum átlépése miatt könnyen oszciállóvá válhat a lépéssorozat. Gyakorlatban λ értékét is célszerű adaptálni az algoritmus során. Az eljáráshoz szükségünk van egy kezdeti súlyvektorra, amelynek meghatározására szintén sokféle módszer adható. Ettől is nagymértékben függ a konvergencia, sőt ha a kritériumfüggvénynek van lokális minimuma, amely nem globális, akkor rossz kezdeti értékekkel az eljárás csak lokális minimumba fog konvergálni.*

3.2. A többrétegű perceptron tanítása

Az előbb bemutatott, a súlyokat iteratívan módosító megközelítést szokták alkalmazni a többrétegű perceptron betanításánál is. A tanítás lépései tehát a következők:

1. Kezdeti súlyok meghatározása: Megadjuk, vagy kiszámítjuk a kezdetben használt, a feladat szempontjából alkalmas (A_l, b_l) mátrix-vektor párokat.

2. Realizáció kiértékelése: A tanító halmaz bemeneti jeleit végigáramoltatjuk a hálózatunkon, vagyis kiszámítjuk a neurális hálónak a pontokban felvett értékét a realizáció képletének megfelelően.

3. Kritériumfüggvény kiértékelése: Összehasonlítjuk az így kapott értékeket az elvártakkal a veszteségfüggvényt használva minden pontban.

4. Leállási feltételek vizsgálata: Megvizsgáljuk, célszerűbb-e megpróbálni tovább optimalizálni a hálózat paramétereit, mint leállítani az eljárást és megtartani a jelenlegieket.

5. Elmozdulás a paramétertéren a lokális minimum irányába: A megfelelő gradiens vektor kiszámításának segítségével eltoljuk a paramétereket.

6. Visszatérünk a 2. pontra.

Egy ilyen ciklust nevezünk az angol szakirodalmat követve *epoch*nak. Vizsgáljuk meg, hogyan kell a többrétegű perceptron betanítása során a hálózat súlyait megváltoztatni a hiba csökkentéséhez a gradiens módszer szerint. A perceptron esetéhez hasonló átdolgozást mutatunk be. Látni fogjuk, hogy a módszer közvetlen alkalmazása nehézségekbe ütközik, mely ezért egy további ötletes kiegészítést igényel.

3.2.1. A hiba visszaterjesztés

Alkalmazzuk a perceptron felépítésének leírásakor megadott jelöléseket. Legyen $L \geq l \geq 2 \in \mathbb{N}$. A hálózat j indexű C_l-beli perceptronjának k-adik tanító pont bemenetére adott kimenetére teljesül, hogy

$$y_j(k) = \varphi_j(v_j(k)), \quad v_j(k) = \sum_{i \in C_{l-1}} w_{ji} y_i(k),$$

ahol w_{ji} az i-edik sorszámú neuronból a j-edikbe mutató élen lévű súly, valamint $w_{j0} := b_j$ és $y_0(k) = 1$, melyek szintén tagjai az előző összegnek (hasonlóan a korábbi egyszerűsítő jelölésekhez). Ekkor, ha C_L a kimeneti réteg neuronjainak halmaza, akkor

$$L(k) = \frac{1}{2}\|\mathcal{E}(k)\|^2 = \frac{1}{2}\sum_{j\in C_L}(\mathcal{E}_j(k))^2 = \frac{1}{2}\sum_{j\in C_L}(d_j(k) - y_j(k))^2.$$

Az előző esethez hasonlóan adódik a lépés iránya az egyes súlyokra vonatkoztatva a gradiens módszert alkalmazva a (3.1.1) képletből:

$$\Delta w_{ji} := w_{ji,next} - w_{ji} = -\frac{\lambda}{n}\sum_{k=1}^{n}\frac{\partial L(k)}{\partial w_{ji}}.$$

Ehhez a veszteségfüggvény gradiensére, vagyis a paramétertér összes koordinátája szerinti parciális derviáltjaira szükség van, amelynek közvetlen kiszámítása numerikusan nehezen oldható meg. Erre ad megoldást nekünk a **hiba visszaterjesztés** (*error back-propagation*) algoritmus, amely a neurális hálók hatékonyságának egyik legfőbb alapját képezi. Az algoritmus alapötlete, hogy a lokális gradienseket rekurzívan, rétegenként visszafele haladva határozzuk meg.

Vegyük észre, hogy az összegben szereplő parciális derivált vektor kifejthető az összetett függvényre vonatkozó differenciálási szabály segítségével a következőképpen:

$$\frac{\partial L(k)}{\partial w_{ji}} = \frac{\partial L(k)}{\partial \mathcal{E}_j(k)}\cdot\frac{\partial \mathcal{E}_j(k)}{\partial y_j(k)}\cdot\frac{\partial y_j(k)}{\partial v_j(k)}\cdot\frac{\partial v_j(k)}{\partial w_{ji}},$$

ahol

$$\frac{\partial L(k)}{\partial \mathcal{E}_j(k)} = \mathcal{E}_j(k), \quad \frac{\partial \mathcal{E}_j(k)}{\partial y_j(k)} = -1, \quad \frac{\partial y_j(k)}{\partial v_j(k)} = \varphi_j'(v_j(k)), \quad \frac{\partial v_j(k)}{\partial w_{ji}} = y_i(n).$$

Ezen összefüggések felhasználásával következnek:

$$\delta_j(k) := -\frac{\partial L(k)}{\partial v_j(k)} = \mathcal{E}_j(k)\varphi_j'(v_j(k)), \quad \Delta w_{ji} = \frac{\lambda}{n}\sum_{k=1}^{n}\delta_j(k)y_i(k).$$

Ez a képlet csak a kimeneti rétegre alkalmazható közvetlenül, mert ott ismertek a d elvárt kimeneti értékek, tehát ott számítható ki $\mathcal{E}_j(k)$.
Vizsgáljuk meg, mit mondhatunk a rejtett rétegek esetében. Legyenek i, j ebben a sorrendben a C_{l-1}, C_l egymást követő rétegek neuronjai (vagyis i biztosan rejtett rétegben van). Ekkor

$$v_j(k) = \sum_{h\in C_{l-1}} w_{jh}\varphi_h(v_h(k)), \quad \frac{\partial v_j(k)}{\partial v_i(k)} = w_{ji}\varphi_i'(v_i(k)).$$

Ezt felhasználva

$$\delta_i(k) = -\frac{\partial L(k)}{\partial v_i(k)} = -\sum_{j\in C_l}\frac{\partial L(k)}{\partial v_j(k)}\cdot\frac{\partial v_j(k)}{\partial v_i(k)} = \sum_{j\in C_l}\delta_j(k)w_{ji}\varphi_i'(v_i(k))$$

Itt már minden érték ismert, ha a későbbi rétegre előzetesen kiszámoltuk a megfelelő értékeket. Tehát ezzel az eljárással rétegenként visszafele haladva rekurzívan kiszámolhatjuk az új súlyokat.

3.2.2. Az aktivációs függvény deriváltja

Látható, hogy a lokális gradiens vektor miatt szükségünk van az aktivációs függvény deriváltjára. Ezért majdnem mindenütt deriválható aktivációs függvényt kell választani, hogy biztosítva legyen a parciális deriváltakra vonatkozó láncszabály alkalmazhatósága. Azonban látni fogjuk [8]-ból, hogy célszerű azt is megkövetelni, hogy az aktivációs függvény előálljon egy közönséges differenciálegyenlet megoldásaként.

Például a hiperbolikus függvényre

$$\varphi(x) = \frac{1}{1+e^{-2mx}}, \quad \varphi'(x) = 2m\frac{e^{-2mx}}{(1+e^{-2mx})^2}.$$

Vegyük észre, hogy

$$\varphi'(x) = 2m\frac{(1+e^{-2mx})-1}{(1+e^{-2mx})^2} = 2m\left(\frac{1}{1+e^{-2mx}} - \frac{1}{(1+e^{-2mx})^2}\right) = 2m(\varphi(x)-\varphi(x)^2)$$

Vagyis a gradiens módszernél használt derivált már korábban kiszámított értékekből közvetlenül nyerhető, hiszen

$$\varphi_i'(v_i(k)) = 2m\varphi(v_i(k))(1-\varphi(v_i(k))) = 2my_i(k)(1-y_i(k))$$

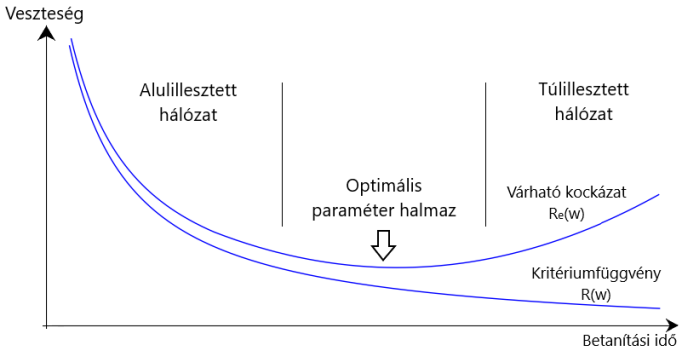
Hasonló összefüggés felírható a szimmetrikus szigmoid függvényekre is. Emiatt a legtöbb feladatra sokáig szigmoid aktivációs függvényeket használtak. Azonban a mély neurális hálók térnyerésével előtérbe került az a probléma, hogy egyes lokális gradiensek könnyen eltűnnek a tanítás során (ld. [14]). E problémának megszüntetésére alkalmazzák a *ReLU* függvényt, azonban így a realizáció csak szakaszonként lineáris függvényként valósítható meg, ráadásul korlátozott számú szakasszal. Ez viszont akkor problémás, ha a közelítendő függvényre nem illeszthető hatékonyan kevesebb szakaszból álló közelítés. Ilyen esetekben a szükséges paraméterek száma és a rétegek száma is megnőhet, mivel több töréspontra van szükség, mely az aktivációs függvény többszöri alkalmazásával érhető el. Ennek orvoslására alkalmasak például olyan szakaszonként lineáris függvények, melyek több (min. 2-3) törésponttal rendelkeznek és a deriváltjuk nagyobb az origótól távolabb eső szakaszokon is, vagyis nem korlátozzuk intervallumba az értékeiket (mint például 2.3.3b).

3.3. A hálózat illesztettsége

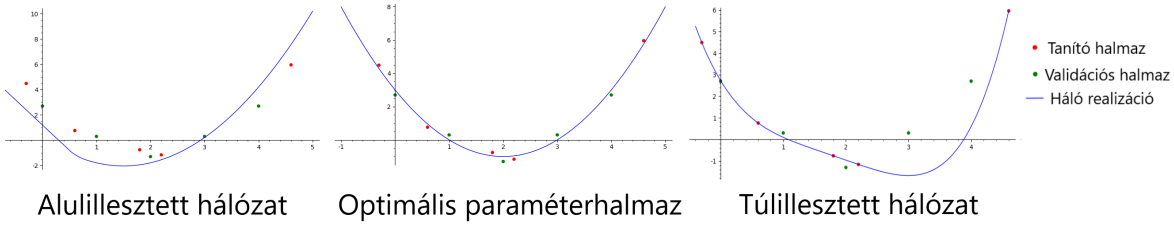
Megjegyzendő, hogy nem célszerű minden rendelkezésre álló információt a tapasztalati közelítés javítására felhasználni. Vagyis nem érdemes minden, a jelenségről elérhető adatpárt bevenni a tanító halmazba. Az adathalmazt gyakorlatban szétválasztják a tanító halmaz, a *validációs halmaz* és a *teszthalmaz* elemeire. Ez utóbbi szerepét később mutatjuk be. A validációs halmaz fontosságát az adja, hogy a tanítás célok tekintetében valójában 2 irányú:

- **Alulillesztés kiküszöbölése:** Kézenfekvő cél, hogy a realizáció által produkált kimenetek jó közelítései legyenek a tanító halmazénak, vagyis közel kerüljünk a kritériumfüggvény minimumához. Ellenkező esetben a hálózatunk alulillesztett, és nem működik hatékonyan.

- **Túlillesztés elkerülése:** Fontos szempont, hogy ne a várható kockázat kárára kerüljünk a tapasztalati kockázat szélsőértékéhez közel, vagyis ne álljon elő az ún. *túlillesztés* jelensége (lásd 3.3.1, 3.3.2 ábrák). Ekkor a hálózatunk jól közelíti a tanító halmaz elemeit, de más esetekben nagyon pontatlan, vagyis rosszul általánosít. Ekkor a hálózat a tanító halmaz elemei közötti olyan bonyolult összefüggéseket tanul meg, amelyek az adathalmazt általánosan nem jellemzik. Ennek ellenőrzésére alkalmazzuk az eljárás közben a validációs halmazt, amelyen vett veszteség egy, a tanító halmaztól független közelítése az ismeretlen várható kockázatnak. Ennek segítségével tehát becsülhető $|R_e(w) - R(w)|$ értéke, és még a túlillesztés bekövetkezése előtt megállíthatjuk a tanítást. Emiatt az algoritmus általában nem éri el a kritériumfüggvény minimumát.



3.3.1. ábra. Az illesztettség alakulása a tanítási idő függvényében .



3.3.2. ábra. A realizáció grafikonjának alakulása.

A tanítás megállási feltételeire az említett illeszkedésre vonatakozókon kívül nem lehet általános szabályokat megfogalmazni. Praktikusan a következő feltételek teljesülése esetén érdemes leállítani az algoritmust:

- Ha a gradiens vektor elég kicsi, mert ekkor a hálózatban használt súlyok már nem változnak jelentős mértékben.

- Ha a kritériumfüggvény értéke alig csökken egy epoch során.

- Ha nagy lesz egy epoch lefutási ideje vagy az epochok száma.

3.4. A sztochasztikus gradiens módszer

Az előzőekben megmutattuk, hogyan számítható ki az optimális paraméterhalmaz a gradiens módszer segítségével. Azonban a neurális hálók alkalmazásának elterjedése maga után vonta az igényt komplex és óriási méretű adathalmazok hálókkal történő feldolgozására is. Ezek a problémák sokszor annyira nagy volumenűek és bonyolultak, hogy akár több tízmilliós nagyságrendű is lehet a szükséges paraméterek száma a megfelelő hatékonyság eléréséhez. Ráadásul, például egy modern beszédfelismerésre tervezett többrétegű perceptron betanításához akár több ezer órányi beszédanyagra is szükség lehet, amely több száz millió elemű mintát jelent. Itt a szokványos, imént bemutatott gradiens módszer már nem megfelelő. Ilyen esetekben ún. *sztochasztikus algoritmusokat* célszerű alkalmazni, melyek közül bemutatunk egy példát [3] gondolatmenetét követve.
Vezessünk be egyszerűsítő jelölést a veszteségfüggvény általános alakjára, majd ebből írjuk fel a kritériumfüggvényt a következő módon:

$$L(i,w) := L(f(x(i),w),d(i)), \quad R(w) = \frac{1}{n}\sum_{i=1}^{n} L(i,w),$$

ahol $n := |T|$. Ilyen jelölésekkel a már bemutatott gradiens módszer a következőképpen módosítja a súlyokat:

$$w_{\text{next}} := w - \lambda \nabla R(w) = w - \frac{\lambda}{n}\sum_{i=1}^{n} \nabla L(i,w).$$

Ezzel a lépéssel rengeteg probléma adódik, ezeket később összegezzük. Ennél fogva cseréljük le a következő módszerre:

$$w_{\text{next}} := w - \lambda \nabla L(i,w),$$

ahol $i \in \{1,\dots,n\}$ diszkrét eloszlás szerint vett véletlen index. Az eljárást vizsgálva megfigyelhető, hogy az így kapott $\{w(k)\}_{k=1,\dots}$ egy sztochasztikus folyamat, melyet az $\{i_k\}_{k=1,\dots}$ véletlen sorozat határoz meg, amely miatt az algoritmust *sztochasztikus gradiens módszernek* nevezik. Ez persze nem garantálja önmagában, hogy egy lépés során a súlyvektor a kritériumfüggvényt tekintve csökkenő irányban változik. Azonban bizonyítható, hogy ha az előbbi várható értékben csökkenő, akkor az eljárás a kritériumfüggvény minimuma felé konvergál. Gyakorlatban ráadásul ez a konvergencia igen gyors, ami ezt a módszert rendkívül hatékonnyá teszi összetett problémák kezelése során. De nemcsak komplex hálók esetén használható a módszer, például az egyszerű perceptron esetében is jól alkalmazható, ez a híres *Adaline* (*Adaptive linear unit*).

3.4.1. A módszer előnyei

Látható, hogy a kritériumfüggvény gradiense a tanító ponthalmaz méretének megfelelő tagszámú összeg. Tehát ennyiszer kell a veszteségfüggvény parciális deriváltjait kiszámolni a hálózat minden egyes súlya szerint. Azaz a lépésszám a halmaz méretével arányosan nő, így numerikusan nagyon drága. Másrészről ha n nagyságrendben felülmúlja a számítások során használt vektorok normáit, akkor az itt összeadott vektorok ennek megfelelően rendkívül kicsik, így a numerikus hibák is meghaladják a tolerálható értékeket. Megfigyelhető, hogy az új megközelítéssel nincs arra szükség, hogy minden tanító halmazbeli pontot felhasználjunk hálózat betanítási algoritmusának futása során. Ez az előzőeken felül további előnyöket is magában hordoz.

Például több száz millió elemű minta esetén a mintaelemek eloszlása szempontjából gyakran már pár ezer megfigyelés is elegendően reprezentatív, mely segítségével az eloszlás már jól becsülhető. Vagyis azok segítségével is pontosan illeszthető a hálózat a tanító halmaz elemeire. Továbbá az is sokszor előfordul, hogy a tanítóhalmaz egyes elemei egymásnak ellentmondó információt kódolnak el, például a megfigyelések zaja vagy azok több, egymással nem konzisztens forrásból való összetétele miatt. Ekkor ezek a szokásos gradiens módszernél felesleges, eredménytelen számításokhoz vezetnek. Továbbá ha egyes minta elemek többször fordulnak elő, (ami szintén előfordulhat a több egymástól független információ forrás igénybevétele miatt) akkor ezek nagyobb súllyal szerepelnek az egyes iterációk lépésvektor kombinációjában. Ezekre a sztochasztikus megközelítés szintén megoldást nyújt. Arra is lehetőségünk van, hogy mindig azt a tanító pontot használjuk, amelyik lokális hibája a legnagyobb, így potenciálisan tovább gyorsíthatjuk a tanulási folyamatot.

3.5. Hálózatépítési technikák

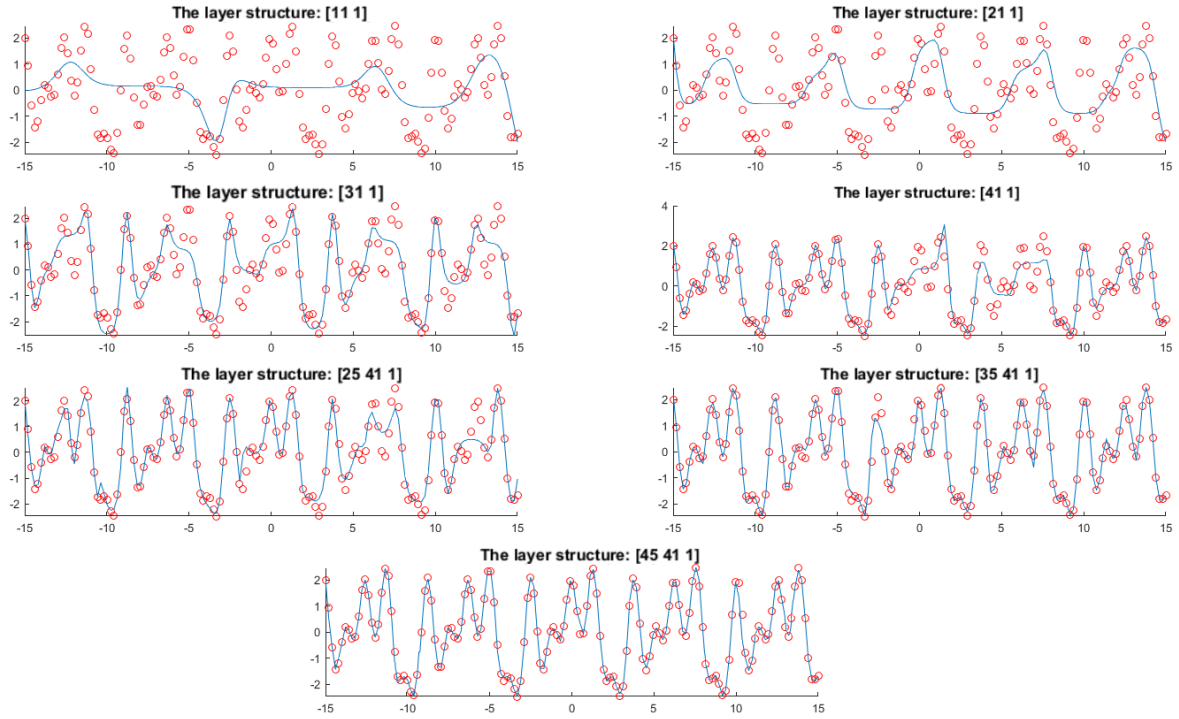
A betanítási folyamat sikerességét alapvetően befolyásolja a hálózat mérete, szerkezete. Már önmagában nehéz feladat meghatározni, milyen struktúrát válasszunk optimális hatékonyságú hálózat eléréséhez. Jellemzően pontos iránymutatást ebben sem lehet adni, csak általános elveket lehet megfogalmazni. Ilyen például, hogy az ún. *bottleneck* (palacknyak) réteget tartalmazó hálózatok általánosító képessége bizonyos esetekben jobb. A bottleneck olyan rejtett réteg, amely lényegesen kevesebb neuront tartalmaz, mint az azt megelőző réteg. Részletekért lásd a [13] cikket. Azonban az alkalmazandó struktúra mindig az adott problémához igazítandó, és tapasztalati úton határozható meg. Célszerű tehát több lehetséges alapszerkezettel is betanítani a hálózatot, és elvetni azokat, amelyek nem produkálnak elegendően jó eredményt. Ezért érdemes nem felhasználni a betanításhoz a rendelkezésre álló megfigyeléseink egy részét, hanem a teszthalmaz számára megőrizni őket. A teszthalmazt a betanított hálózatok teljesítményének felmérésére alkalmazzuk. Ennek segítségével adunk becslést azok megbízhatóságára, pontosságára, és kiválaszthatjuk a számunkra optimálisat. A megvizsgálandó hálószerkezetek megalkotásának általánosan kétféle megközelítése lehetséges:

- **Hálózatmetszés:** (*network pruning*) Kiindulunk egy nagyméretű, sok paraméterrel rendelkező, mély szerkezetből. Ha a betanítás túl hosszú vagy a háló túlillesztett, akkor csökkentjük a rétegek vagy a paraméterek számát a számítási tapasztalatok alapján. Ezzel a tanítási idő lerövidíthető és megszüntethető a túlillesztettség, de alulillesztéshez vezet, ha túl kevés paramétert hagyunk meg.

Ennek módszernek nagyméretű hálózatok esetén különösen hatékony típusa a *dropout* eljárás. A célja olyan neuronok törlése a hálózatból, amelyek hatása aránylag a legkisebb. A lényege, hogy a tanítás során a neurális háló rejtett neuronjait egy előre meghatározott p valószínűséggel töröljük, majd ezt összehasonlítjuk az eredeti hálózat kimenetével. A klasszikus algoritmus leírását [21]-ben találhatjuk meg. Ennek az eredmények egy javítási javaslata is megtalálható a [4] cikkben. Azonban nemcsak egész neuronok törlésével csökkenthetjük le a hálózatunk méretét, hanem az egyes súly paraméterek törlésével is. Például alkalmazhatjuk a szokásos kritériumfüggvény helyett annak egy kombinációját a súlyok egy ún. *büntetőfüggvényével*. Ezt jól megválasztva a valójában szükségtelen kapcsolatok kicsi súllyal szerepelnek majd a hálózatban (részletekért lásd a [20] munkát). A megfelelő büntetőfüggvény kiválasztása általában oly módon történik, hogy egyaránt nagyobb értékkel ”bünteti” a feleslegesen sok kapcsolatot tartalmazó hálózatot, illetve a túlzottan nagy súlyokat a hálózatban. Népszerű például a súlyok négyzetösszege. Ez a súlyok törlésére vonatkozó tetszőleges kritériummal kiegészíthető.

- **Hálózatnövelés:** (*network growing*) Először kevesebb paraméterű, sekély (azaz kevés réteget tartalmazó) szerkezetet próbálunk ki, melynek a betanítása gyors. Abban az esetben, ha a tanítás során mégsem tudunk elegendően jó eredményt elérni, és a hálózat a betanítás után alulillesztett marad, akkor megnöveljük a rétegek számát vagy azokban a neuronok, illetve a szabad súlyok számát. Így javíthatunk az alulillesztettségen, azonban ezek túlillesztéshez vezetnek, ha nem tartjuk őket korlátok között. Ekkor a probléma gyökere, hogy túl sok paraméterrel akarjuk leírni a jelenség lényeges összefüggéseit, a megoldandó problémához képest túl nagy a felhasznált neuronok száma. A 3.5.3 ábra egy ilyen algoritmus lefutása során mutatja be a realizáció alakulását, nyomon követve a rétegekben aktuálisan használt neuronok számát is. Az ehhez írt program megtalálható a 48. oldalon.

Ez a módszer sok esetben jobb a hálózat metszésnél, mivel a szükségtelenül nagy hálózattal való kezdeti probléma megközelítés feleslegesen megnövelheti a számítási időt. Fontos szempont azonban, hogy a hálózatot úgy növeljük, hogy a korábbi számításainkat is fel tudjuk használni, és a már betanított hálózatokból származó lehetséges előnyöket ne veszítsük el. A [2] cikkben arra találunk megoldást, hogyan növeljük a hálózatot neuronok hozzáadásával dinamikusan, azaz páruhazmosan annak betanításával.



3.5.3. ábra. Hálózatnövelés a készített segédprogrammal.

Ha ezekkel a módszerekkel nem tudunk megfelelő illeszkedést elérni, akkor módosíthatjuk a tanító és a validációs halmazok méretének arányát is. Megnövelve a tanító halmaz méretét, így több adatot szolgáltatva a neurális hálónak a tanításhoz javítható az alulillesztettség. Mivel azonban így a validációs halmaz mérete csökken, nehezebb elkerülni a túlilleszkedést. Hasonló elven, a validációs halmaz arányát megnövelve pedig a túlilleszkedésen lehet javítani az alulillesztés veszélye mellett. Mindezek mellett természetesen a teszthalmaz arányát is változtathatjuk a megfigyeléseinkben, de azt túlzottan csökkentve már nehéz jó becslést adni a hálózat teljesítményéről. Látható, hogy ezek hatékonyság szempontjából optimális arányának meghatározása nem egyértelmű feladat. Ezzel foglalkozik részletesen a [9] és [10] cikk.

Amennyiben a megfigyeléseink arányának finomhangolásával sem elegendően pontos a hálózat, akkor megváltoztathatjuk a strukturát: más aktivációs vagy veszteségfüggvényt használhatunk, illetve módosíthatjuk a rétegekben a szabad paraméterek elhelyezkedését vagy a betanítási algoritmust. Célszerű lehet az is, hogy több viszonylag hatékony, de egymástól elegendő mértékben különböző, gyorsan betanítható modell eredményeinek átlagát vesszük a végleges realizáció eredményének. A gyakorlati tapasztalatok azt mutatják, hogy az adathalmazunk elvárt kimeneti értékeit érdemes előfeldolgozni, vagyis centrálni, illetve normalizálni a felhasználás előtt. Ezzel jelentősen növekedhet a betanított hálózat megbízhatósága (ld. [12]). A legegyszerűbb példa erre az adatok megfelelő transzformációja a $[0, 1]$ vagy $[-1, 1]$ intervallumok valamelyikébe.

4. fejezet

Szakaszonként lineáris közelítés neurális hálókkal

A következőkben megmutatjuk, hogy tetszőleges szakaszonként lineáris függvény megadható alkalmas sekély neurális hálóval. Pontosítva, tetszőleges szakaszonként lineáris függvény előállítható, mint egy 2 rétegű neurális háló realizációja. Ennek az eredménynek a fontosságát az adja, hogy ezáltal végezhetünk lineáris spline interpolációt illetve közelítést neurális hálók segítségével is.

4.0.7 Megjegyzés. *Szakaszonként lineáris függény szimulációja esetén nem választhatunk deriválható aktivációs függvényt, mint amilyenek például az általánosságban elterjedt szigmoid típusú függvények. Ennek az az oka, hogy deriválható aktivációs függvényt alkalmazva a realizáció deriválható transzformációk sorozata, hiszen az aktivációs függvényt affin lineáris transzformációkkal komponáljuk. Ezért ilyen függvénnyel a realizáció is deriválható függvény lesz. Ezért legyen a továbbiakban az általunk használt aktivációs függvény a korábban már definiált ReLU függvény.*

4.1. Közelítés rögzített alappontokon

4.1.8 Definíció. *Legyen adott $I = (a, b)$, $(a, b \in \mathbb{R})$ nyílt intervallum és legyen ezen adott az $a = x_0 < x_1 < \cdots < x_n = b$ osztópontok által meghatározott felosztás. Ezt ezentúl röviden az (a, b) intervallum n elemű felosztásnak fogjuk nevezni, és $\mathcal{T}$-val jelöljük. Azt mondjuk, hogy $I_i = (x_{i-1}, x_i)$, $i = 1, \ldots, n$ a $\mathcal{T}$ felosztáshoz tartozó részintervallumok.*

4.1.9 Definíció. *Legyen adott $I = (a, b)$, $(a, b \in \mathbb{R})$ nyílt intervallum, és legyen ehhez adott $\mathcal{T}$ tetszőleges n elemű felosztás. Legyen $\boldsymbol{p} = (p_i)_{i=1,\ldots,n} \subset \mathbb{N}$ egy $\mathcal{T}$ részintervallumain értelmezett polinom-fokszám eloszlás. Definiáljuk $S_{\boldsymbol{p}}(I, \mathcal{T})$ folytonos, szakaszonként polinomiális függvények terét a következő módon:*

$$S_{\boldsymbol{p}}(I, \mathcal{T}) := \{v \in H^1(I) : v|I_i \in \boldsymbol{P}_{p_i}(I_i) \ , i \in \{1, \ldots n\}\}$$

ahol $H^1(I) := W^{1,2}(I) = \{f : I \to \mathbb{R}$ abszolút folytonos, $f' \in L^2(I)\}$ Szoboljev tér.

Jelölje $\boldsymbol{p} = (p, \dots, p)$*, ekkor legyen a* p *fokszámú szabad alappontú* n *szakaszon spline függvények tere:*

$$S_p^n := \bigcup \{S_{\boldsymbol{p}}(I, \mathcal{T}) : \mathcal{T} \text{ egy } n \text{ elemű felosztás } I\text{-n}\}\,.$$

A következőekben [15] segítségével megfogalmazunk egy tételt, melyet kiegészítünk egy tételhez kapcsolódó saját állítással, majd a kettőre együtt adunk egy saját konstruktív bizonyítást. Ezután a tételben meggondoltakat hasznájuk fel valós függvények közelítésére.

4.1.10 Tétel. *Legyen* $I = (a, b)$ *nyílt intervallum és ezen adott egy* $\mathcal{T}$ n *elemű felosztás. Legyen* $v \in S_{\boldsymbol{1}}(I, \mathcal{T})$ *szakaszonként lineáris abszolút folytonos függvény. Ekkor létezik egy* Φ^v *ReLU hálózat, amelyre* $R(\Phi^v) = v$*,* $L(\Phi^v) = 2$*,* $M(\Phi^v) \leq 3n + 1$*,* $M_1(\Phi^v) \leq 2n$*,* $M_2(\Phi^v) \leq n + 1$*.*

4.1.11 Állítás. *Belátható az is, hogy az előző becslések élesek. Vagyis, ha a* v *függvény nem speciális, tehát egymást követő szakaszokon különböző nem konstans lineáris függvényként viselkedik, akkor az előbbi paraméterszám optimális (minimális). Tehát nem adható meg olyan konstrukció, amely rögzített* n *darab alapponton tetszőleges* v *szakaszonként lineáris függvényt előállít egy* Φ *sekély (2 rétegű) ReLU hálózat segítségével úgy, hogy* $M(\Phi) \leq 3n$*.*

Bizonyítás. Tegyük fel, hogy Φ a feltételeknek eleget tevő neurális háló. Ekkor

$$\begin{aligned} &R(\Phi^v)(x) = A_2[\varphi(A_1 x + b_1)] + b_2 = \\ &= \left(\frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}}\right)(x - x_{i-1}) + v(x_{i-1}), \quad \text{ha } x_{i-1} \leq x \leq x_i \ (i = 1, \dots, n). \end{aligned}$$

Itt $A_1 \in \mathbb{R}^{m\times 1}$ és $b_1 \in \mathbb{R}^m$, valamint $A_2 \in \mathbb{R}^{1\times m}$ és $b_2 \in \mathbb{R}$ valamilyen m egészre, mert $R(\Phi^v) : \mathbb{R} \to \mathbb{R}$. Vegyük észre, hogy a realizációt megadó fenti függvény pontosan azon x pontokban nem differenciálható, ahol $\varphi(A_1 x + b_1)$ nem differenciálható. Ezek olyan pontok, ahol az $A_1 x + b_1$ kifejezésnek zérushelye van legalább egy koordinátájában, valamint az $x_1, \dots, x_{n-1}$ interpolációs alappontok e halmaznak biztosan részei. Meggondolható, hogy ReLU függvénnyel az egyes koordináták segítségével félegyeneseken tudjuk leírni a realizáció viselkedését. A v függvény n darab különböző szakaszon viselkedik feltehetően a szomszédosaktól eltérő lineáris függvényként. Tehát legalább ennyi koordinátára szükségünk van a rejtett rétegben, mivel az egyes szakaszoknak a félegyeneseknek különböző metszetébe kell esniük. Így tehát $m \geq n$, vagyis optimális esetben $m = n$. Vegyük észre, hogy az A_1, A_2 mátrixok minden koordinátája nemnulla, mivel v feltehetően egy közbülső szakaszon sem konstans. Meggondolható, hogy A_1 választható úgy, hogy minden eleme pozitív legyen. Ekkor az x_{j-1}, x_j alappontok között $\varphi = id$ pontosan j darab koordinátában, és azonosan 0 függvény a maradék $n - j$ koordinátában, ahol $j = 1, \dots, n$. Itt tetszőleges sorrendet választhatunk, mert a koordináták szerepe szimmetrikus. Vagyis ha egy rögzített sorrendet permutálunk, akkor az A_2 mátrix koordinátáinak megfelelő permutációjával az eredményünk nem változik. Vegyük tehát itt a koordinátákat index szerint növekvő sorrendben.

Vizsgáljuk meg sorban az $x_{i-1} \leq x \leq x_i$ eseteket.

- $i = 1$ esetben:

  $$A_2[\varphi(A_1x+b_1)]+b_2 = A_2[(A_1)_1x+(b_1)_1, 0, \ldots, 0]+b_2 = (x-x_0)\frac{v(x_1)-v(x_0)}{x_1-x_0}+v(x_0).$$

  Látható, hogy $(A_1)_1$ választható 1-nek, mert a képlet az ismeretlen paramétereket tekintve invariáns a numnulla konstansszoros változtatásokra a kikötések számánál nagyobb szabadsági fok miatt. Ekkor $(b_1)_1 = -x_0$, illetve $b_2 = v(x_0)$, és $(A_2)_1 = \frac{v(x_1)-v(x_0)}{x_1-x_0}$. Általános esetben ezek egyike sem 0.

- $i = 2$ esetén:

  $$\begin{aligned} &A_2[\varphi(A_1x + b_1)] + b_2 = (A_2)_1[(A_1)_1x + (b_1)_1] + (A_2)_2[(A_1)_2x + (b_1)_2] + b_2 = \\ &= \frac{v(x_1) - v(x_0)}{x_1 - x_0}(x - x_0) + (A_2)_2[(A_1)_2x + (b_1)_2] + v(x_0) = \\ &= (x - x_1)\frac{v(x_2) - v(x_1)}{x_2 - x_1} + v(x_1) = v(x) \end{aligned}$$

  Előzőhöz hasonlóan élhetünk az $(A_1)_2 = 1$ választással. Az egyenlet két oldalán x együtthatója egyenlő, ezért

  $$(A_2)_2 = \frac{v(x_2) - v(x_1)}{x_2 - x_1} - \frac{v(x_1) - v(x_0)}{x_1 - x_0}.$$

  A két oldalon a konstansok értéke is megegyezik, ezért

  $$\begin{aligned} (A_2)_2(b_1)_2 &= (-x_1)\frac{v(x_2) - v(x_1)}{x_2 - x_1} + v(x_1) - \left[\frac{v(x_1) - v(x_0)}{x_1 - x_0}(-x_0) + v(x_0)\right] = \\ &= (-x_1)\left[\frac{v(x_2) - v(x_1)}{x_2 - x_1} - \frac{v(x_1) - v(x_0)}{x_1} - \frac{(v(x_1) - v(x_0))x_0}{(x_1 - x_0)x_1}\right] = \\ &= (-x_1)\left[\frac{v(x_2) - v(x_1)}{x_2 - x_1} - \frac{(v(x_1) - v(x_0))(x_1 - x_0 + x_0)}{(x_1 - x_0)x_1}\right] = (A_2)_2(-x_1). \end{aligned}$$

  Vagyis $(b_1)_2 = -x_1$. Alkalmazzunk innentől teljes indukciót. Tegyük fel, hogy az optimális konstrukció mátrixainak és vektorainak koordinátái $(k-1)$-ig a következő módon állnak elő:

  - $(A_1)_j := 1$ $(\forall j = 1, \ldots, k-1)$,
  - $(b_1)_j := -x_{j-1}$ $(\forall j = 1, \ldots, k-1)$,
  - $(A_2^v)_j := \begin{cases} \frac{v(x_j)-v(x_{j-1})}{x_j-x_{j-1}} - \frac{v(x_{j-1})-v(x_{j-2})}{x_{j-1}-x_{j-2}}, & \text{ha } j = 2, \ldots, k-1 \\ \frac{v(x_j)-v(x_{j-1})}{x_j-x_{j-1}}, & \text{ha } j = 1, \end{cases}$
  - $b_2^v := v(x_0) \in \mathbb{R}$.

- Legyen $i = k$, vagyis $x_{k-1} \le x \le x_k$. Ekkor

  $$\begin{aligned} &A_2[\varphi(A_1x + b_1)] + b_2 = b_2 + \sum_{i=1}^{k}(A_2)_i[(A_1)_ix + (b_1)_i] = \\ &= \frac{v(x_k) - v(x_{k-1})}{x_k - x_{k-1}}(x - x_{k-1}) + v(x_{k-1}) = v(x). \end{aligned}$$

Az előző egyenletből következik, hogy

$$\sum_{i=1}^{k}(A_2)_i[(A_1)_i x + (b_1)_i] = v(x) - b_2 =$$
$$= \left(\frac{v(x_k) - v(x_{k-1})}{x_k - x_{k-1}}\right)(x - x_{k-1}) + v(x_{k-1}) - v(x_0) =$$
$$= \left(\frac{v(x_k) - v(x_{k-1})}{x_k - x_{k-1}}\right)(x - x_{k-1}) + \sum_{i=1}^{k-1}(v(x_i) - v(x_{i-1})) =$$
$$= \left(\frac{v(x_k) - v(x_{k-1})}{x_k - x_{k-1}}\right)(x - x_{k-1}) + \sum_{i=1}^{k-1}\left(\frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}}\right)[(x - x_{i-1}) - (x - x_i)]$$
$$= (x - x_0)\frac{v(x_1) - v(x_0)}{x_1 - x_0} + \sum_{i=2}^{k}(x - x_{i-1})\left(\frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}} - \frac{v(x_{i-1}) - v(x_{i-2})}{x_{i-1} - x_{i-2}}\right).$$

Innen pedig azt kapjuk, hogy

$$(A_2)_k[(A_1)_k x + (b_1)_k] = (x - x_{k-1})\left(\frac{v(x_k) - v(x_{k-1})}{x_k - x_{k-1}} - \frac{v(x_{k-1}) - v(x_{k-2})}{x_{k-1} - x_{k-2}}\right).$$

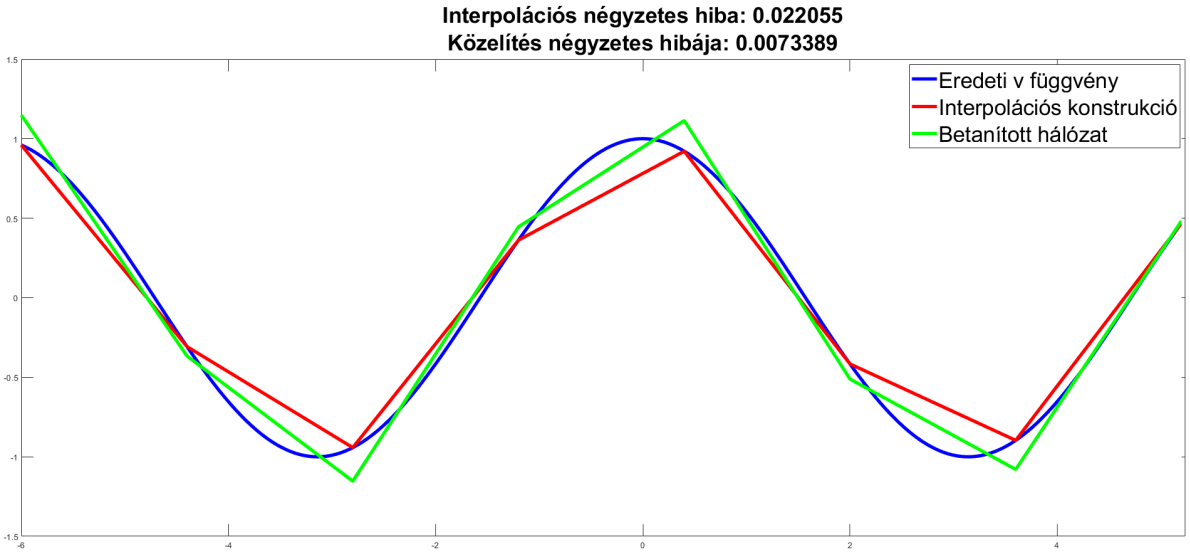
Ismét élhetünk az $(A_1)_k = 1$ választással, amiből egyértelműen következik, hogy $(b_1)_k = -x_{k-1}$ és

$$(A_2)_k = \left(\frac{v(x_k) - v(x_{k-1})}{x_k - x_{k-1}} - \frac{v(x_{k-1}) - v(x_{k-2})}{x_{k-1} - x_{k-2}}\right).$$

□

4.1.12 Megjegyzés. *Beláttuk tehát, hogy rögzített alappontokon tudunk lineáris spline interpolációt végezni ReLU hálózatok segítségével is, a paraméterek rögzítésével.*

Meggondolható, hogy ebből a konstrukcióból kiindulva könnyedén megadható az a lineáris spline függvény is, amelynek töréspontjai a rögzített alappontokon helyezkednek el és ezek közül a legjobban közelíti a vizsgált v függvényt, lásd a 4.1.1 ábrán. Ekkor már azonban szükség van az előbbi konstrukcióval létrehozott hálózat betanítására is, mely során kizárólag a kimeneti rétegben engedjük a súlyok adaptációját. Látható, hogy az előbbi konstrukció kimeneti rétegének súlyai a $v(x_i)$ függvényértékektől csak lineárisan függenek, és a konstrukcióban csak ezeket változtatjuk. Emiatt az optimális paraméter halmaz ez esetben egyszerű lineáris optimalizálási feladattá redukálható. Természetesen a szokásos betanítási eljárások is működnek, csak a kimeneti réteg súlyait módosítva. Ez a korábban már említett λ tanulási paraméter segítségével elérhető, amennyiben annak értékét rétegenként különbözőnek vesszük. Ezt valósítjuk meg a 45. oldalon található saját program segítségével. Érdemes lehet a hatékonyabb közelítés érdekében ekvidisztáns felosztást választani.



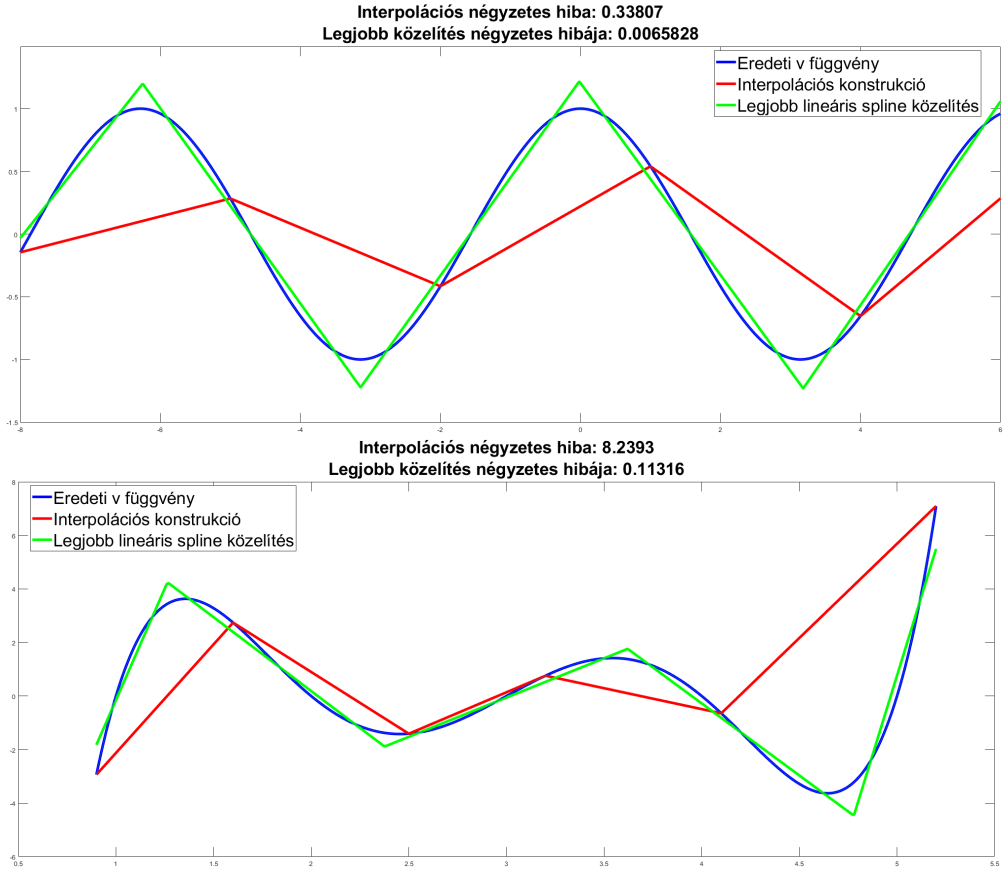
4.1.1. ábra. Közelítés rögzített alappontokon.

4.2. Szabad alappontú spline közelítés

Valójában a rögzített alappontokon történő közelítés gyakorlatban sokkal ritkábban szükséges. Általában nem szükségszerű meghatároznunk, hogy a spline függvény mely szakaszokon viselkedjen lineárisan, sokkal fontosabb, hogy a lehető legjobban közelítsen egy adott függvényt, lásd például a 4.2.2 ábrát. Ezt úgy érjük el, hogy a közelítés alappont halmazát nem rögzítjük előre, hanem az is meghatározandó. Az alappontok halmazának meghatározása külön egy nehéz numerikus matematikai feladat. Vizsgáljuk meg, hogyan oldhatjuk meg a feladatot neurális hálók segítségével.

Vegyük észre, hogy a ReLU hálózat 4.1.10 tétel szerint történő konstrukciójából kiindulva a szabad alappontokon történő lineáris spline közelítés könnyen megoldható. Ehhez csak azt kell feltennünk, hogy a közelítendő függvényt elegendően sok pontjában polinom időben ki tudjuk értékelni. Ehhez a feladathoz annyiban kell módosítanunk az előző megoldásunkat, hogy a hálózat betanítása során a rejtett réteg torzításvektorában is megengedjük az értékek adaptációját. A feladat azonban lineáris optimalizálás segítségével már nem oldható meg, ugyanis a konstrukcióban a súlyok az x_i alappontoktól nemlineárisan függenek. Ebben az esetben a betanítás során az adott függvényt univerzálisan legjobban közelítő szakaszonként lineáris függvényhez jutunk. Ennek alappontjait is könnyen visszakaphatjuk, amennyiben a bemeneti réteg mátrixát csupa 1-esnek választjuk, és a tanulási paramétert 0-ra vesszük itt. Az alappontokat ekkor a b_2^v vektor elemeiből olvashatjuk ki a betanítási folyamat után. Az ezt megvalósító program a 47. oldalon található.

Megjegyzendő, hogy az optimális lineáris spline becslést tetszőleges kezdeti súlyokkal jellemzően nem tudjuk elérni. Ennek az oka a már korábban említett probléma, hogy a betanítási eljárás megreked a paraméter térbeli lokális minimumok egyikén.
Ezért van szükség arra, hogy az optimum egy elegendően jó közelítéséből induljunk ki, mint amilyen a kezdeti interpolációs konstrukció is. Ilyen kezdeti súlyokkal a konvergencia rendkívül gyors még nagyméretű problémák esetén is.



4.2.2. ábra. Közelítés szabad alappontokon.

Látható, hogy ezzel a módszerrel akár végtelen sokszor differenciálható függvényeket is hatékonyan lehet közelíteni. Az eljárás előnye, hogy a probléma méretétől és bonyolultságától függetlenül elég két réteget használni, valamint a súlyok alkalmas inicializálása sem jelent problémát. Lévén, hogy a rejtett mátrix súlyait nem módosítjuk, n neuron esetén csak mindössze $2n+1$ paramétert kell optimalizálni, amely nagyon kevés, így az eljárás nagy adathalamaz és n mellett is hatékony marad. Ráadásul a legfontosabb, hogy ilyen inicializálással jellemzően nem fordulhat elő a hálózat túlillesztése sem a szerkezet egyszerűségéből fakadóan. A betanításnál az $x_0, \ldots, x_{n-1}$ alappontok helyét és az ott felvett értékeket állítjuk be, valamint a felső intervallumon való extrapoláció meredekségét. Itt több paraméter praktikusan jobb közelítést jelent, amint azt később látni is fogjuk. Vizsgáljuk meg, milyen elméleti és gyakorlati becslés adható az ilyen jellegű közelítés konvergenciarendjére vonatkozóan. Az elméleti becslés kimondásához szükségünk van a következő [5] szerinti definíciókra:

4.2.13 Definíció. *Az $f \in L^p(I)$, $(I = (a,b))$ függvény t-beli simasági arányszámának (modulus of smoothness) nevezzük a következő mennyiséget:*

$$\omega_p^2(f,t) = \sup_{|h|\leq t} \|\Delta_h^2 f\|_{L^p(I)},$$

$$\textit{ahol } \Delta_h^2 f(x) := \begin{cases} f(x-2h) - 2f(x-h) + f(x), & \textit{ha} \quad x, x-h, x-2h \in I, \\ 0 \ \textit{egyébként.} \end{cases}$$

4.2.14 Definíció. *Legyen $I = (a, b)$ intervallum, $n \in \mathbb{N}$, $0 < p, q \leq \infty$, legyen $0 < \alpha \leq 1$, valamint legyen $s = n + \alpha$. Besov-térnek nevezzük a következő függvényteret:*

$$B^s_{q,p}(I) := \left\{ f \in W^{n,p}(I) : \quad \int_0^1 \left| \frac{\omega_p^2(f^{(n)}, t)}{t^\alpha} \right|^q \frac{dt}{t} < \infty \right\},$$

ahol

$$W^{n,p}(I) := \left\{ f \in C^{(n-1)}(I) : \quad f^{(n-1)} \text{ abszolút folytonos}, f^{(n)} \in L^p(I) \right\}$$

magasabb rendű Szoboljev tér az $\|f\|_{W^{n,p}(I)} := \left(\sum_{j=0}^{n} \|f^{(j)}\|^p_{L_p(I)} \right)^{\frac{1}{p}}$ *normával ellátva. A Besov-tér normája ezzel:*

$$\|f\|_{B^s_{q,p}(I)} := \left(\|f\|^q_{W^{n,p}(I)} + \int_0^1 \left| \frac{\omega_p^2(f^{(n)}, t)}{t^\alpha} \right|^q \frac{dt}{t} \right)^{\frac{1}{q}}.$$

Kiderül [15]-ből, hogy ezt a bonyolult függvényteret kell használnunk a szabad alappontú lineáris spline közelítés hibájának becslésére:

4.2.15 Tétel. *Legyen $0 < q < q' \leq \infty$, legyen $s < \max\{2, 1 + \frac{1}{q}\}$ és legyen $0 < s' < \min\{1 + \frac{1}{q'}, s - \frac{1}{q} + \frac{1}{q'}\}$, valamint $0 < p, p' \leq \infty$. Ekkor $\exists\ C > 0$ konstans úgy, hogy $\forall n \in \mathbb{N}$-re és $\forall f \in B^s_{q,p}(I)$-re $\exists\ h^n \in S^n_1$, hogy*

$$\|f - h^n\|_{B^{s'}_{q',p'}(I)} \leq C n^{-(s-s')} \|f\|_{B^s_{q,p}(I)}.$$

4.2.16 Megjegyzés. *A tételbeli C konstans függ a q, q', p, p', s, s' számok értékeitől, nem függ azonban az f függvénytől, valamint n-től, azaz a felosztás elemszámától.*

Az imént láttuk, hogy tetszőleges n elemű felosztáson vett lineáris spline függvény előállítható egy $n + 1$ neuronból álló 2 rétegű ReLU hálózat realizációjaként. Ezért az imént kimondott tételt átfogalmazhatjuk a neurális halókkal elérhető legjobb közelítésre vonatkozó hibabecsléssé a következő módon:

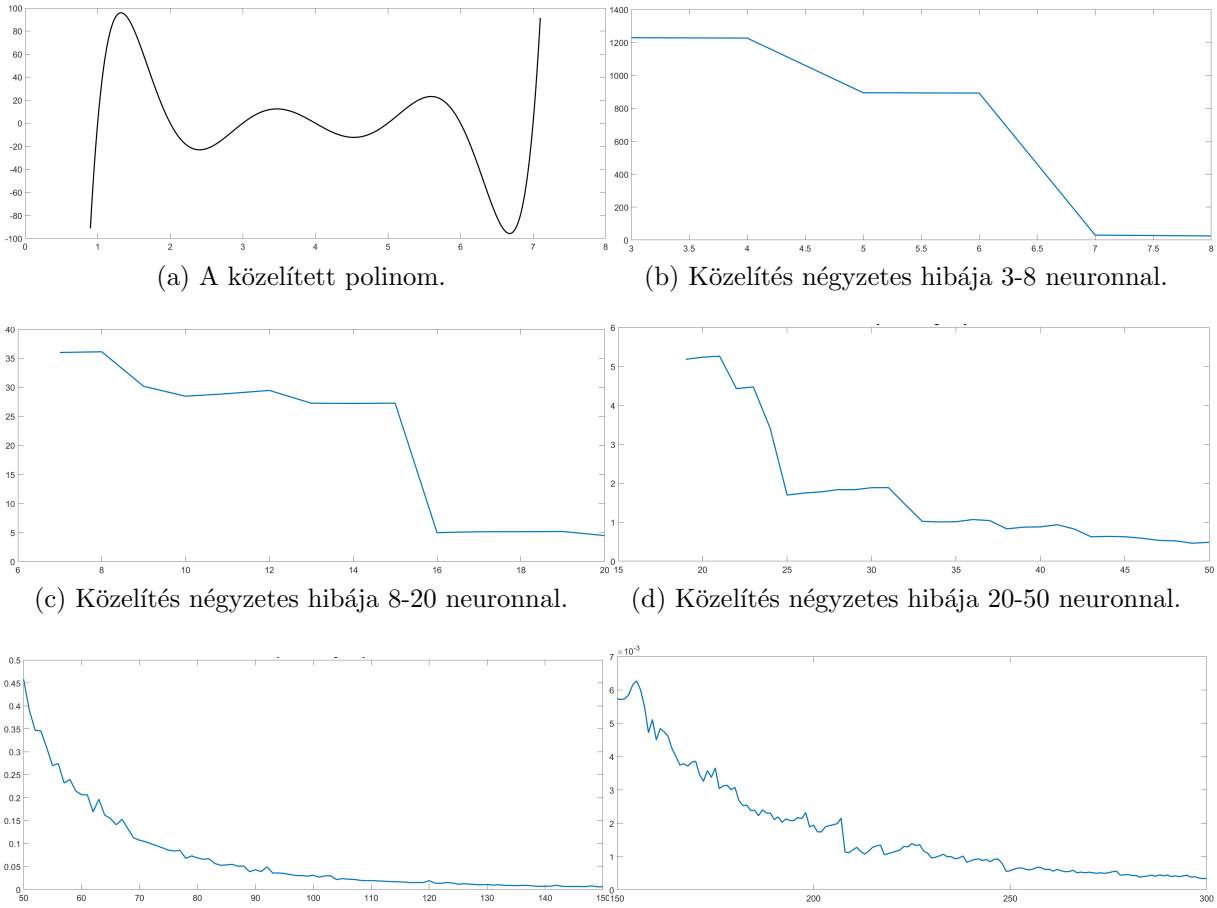
4.2.17 Következmény. *Legyen q, q', p, p', s, s' úgy, mint előbb. Ekkor $\exists\ C > 0$ konstans úgy, hogy $\forall n \in \mathbb{N}$-re és $\forall f \in B^s_{q,p}(I)$-re $\exists\ \Phi^f$, hogy $N(\Phi^f) = n + 1$, $L(\Phi^f) = 2$ és*

$$\|f - \Phi^f\|_{B^{s'}_{q',p'}(I)} \leq C n^{-(s-s')} \|f\|_{B^s_{q,p}(I)}.$$

Ebből is következik a neurális hálóknak az a más függvénytereken (sőt például korlátos aktivációs függvények esetén több dimenzióban is régóta [11]) bizonyított tulajdonsága, hogy akár már két rétegű neurális hálóval is tetszőlegesen jól közelíthető bármilyen folytonos függvény. Továbbá elérhető ugyanolyan polinomiális sebességű konvergencia a hálózat méretének függvényében, mint a spline felosztás elemszámában.

4.3. A tapasztalati konvergenciarend

Vizsgáljuk meg, hogy a tapasztalati eredmények alátámasztják-e az előbbi eljárás polinomiális konvergenciáját a rejtett réteg neuronjainak függvényében. Közelítsük a $p(x) = x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 + 13068x - 5040$ polinomfüggvényt a $[0.8, 7.2]$ intervallumon a szabad alappontú lineáris spline módszerrel neurális hálóval. Növeljük az n (vagyis a rejtett réteg neuronszámának) értékét, és vizsgáljuk meg, hogyan alakul a hálózat négyzetes hibája.



(a) A közelített polinom.

(b) Közelítés négyzetes hibája 3-8 neuronnal.

(c) Közelítés négyzetes hibája 8-20 neuronnal.

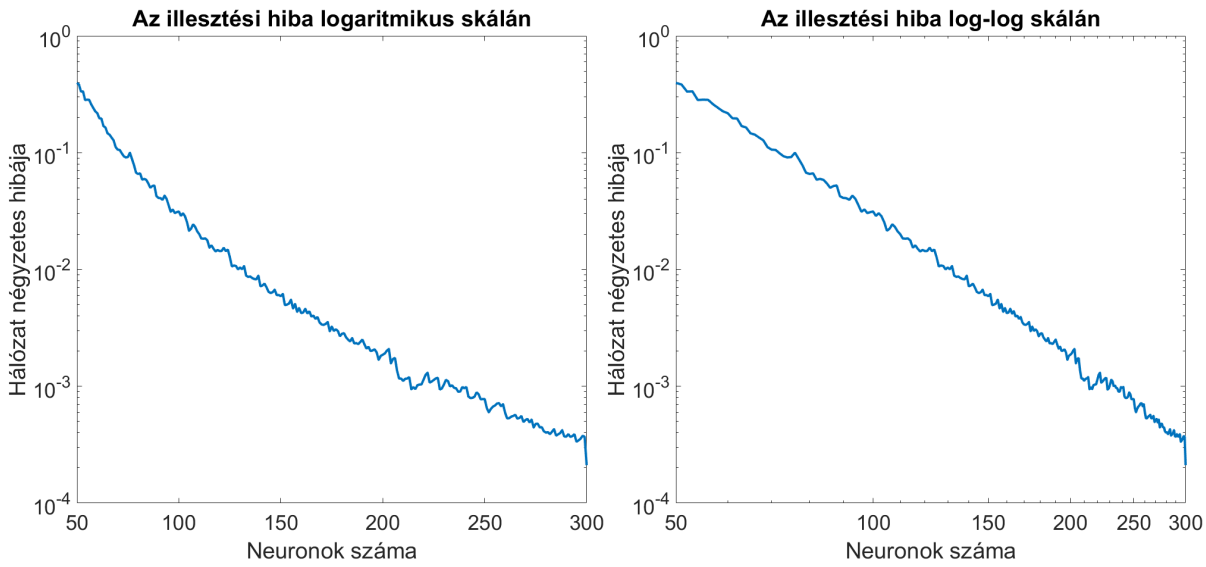
(d) Közelítés négyzetes hibája 20-50 neuronnal.

(e) Közelítés négyzetes hibája 50-150 neuronnal.

(f) Közelítés négyzetes hibája 150-300 neuronnal.

4.3.3. ábra. Polinom közelítés hibája lineáris spline függvény esetén.

Láthatóan a függvény elegendő neuronnal minden határon felül közelíthető. A konvergencia nem monoton, ez többek között a MATLAB betanítási algoritmusában szereplő véletlen paraméterek hatása. Ezek alapján azonban nem nyilvánvaló, hogy a gyakorlatban elért konvergencia-sebesség a 4.2.17 következmény elméleti eredményéhez hasonlóan polinomiális vagy pedig exponenciális-e. Vizsgáljuk meg tehát az előbbi eredményeket a tengelyek átskálázásával is.

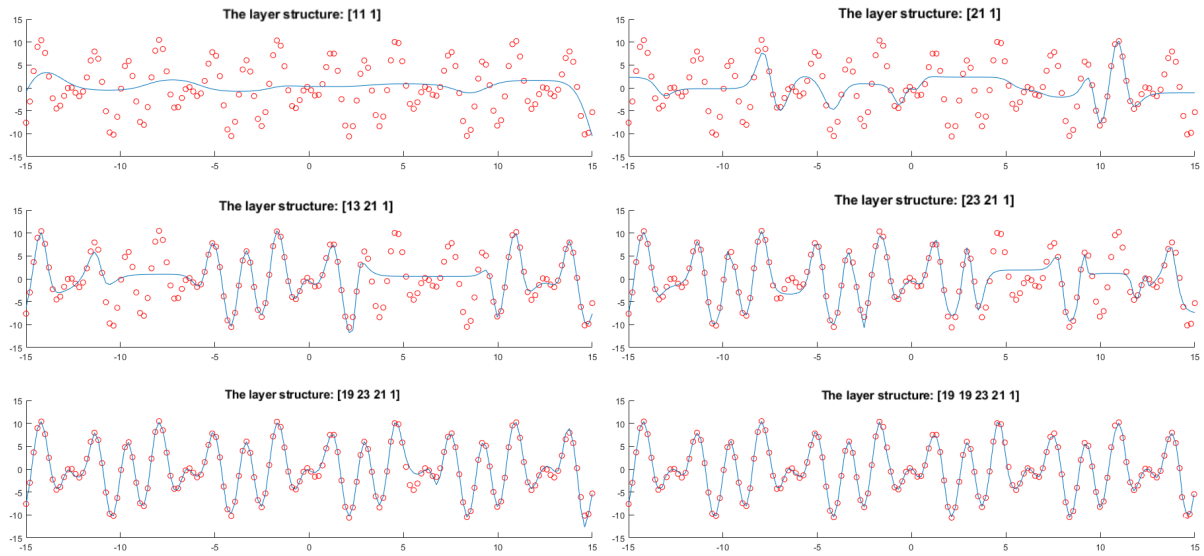


4.3.4. ábra. A szabad alappontú spline-közelítés konvergenciája különböző skálákon.

Látható, hogy a logaritmikus skálán nem lineárisan helyezkednek el az eredmények, viszont a log-log skálán igen. Ez alapján már világos, hogy a gyakorlati konvergenciasebesség megfelel az elméleti polinomiális rendnek.

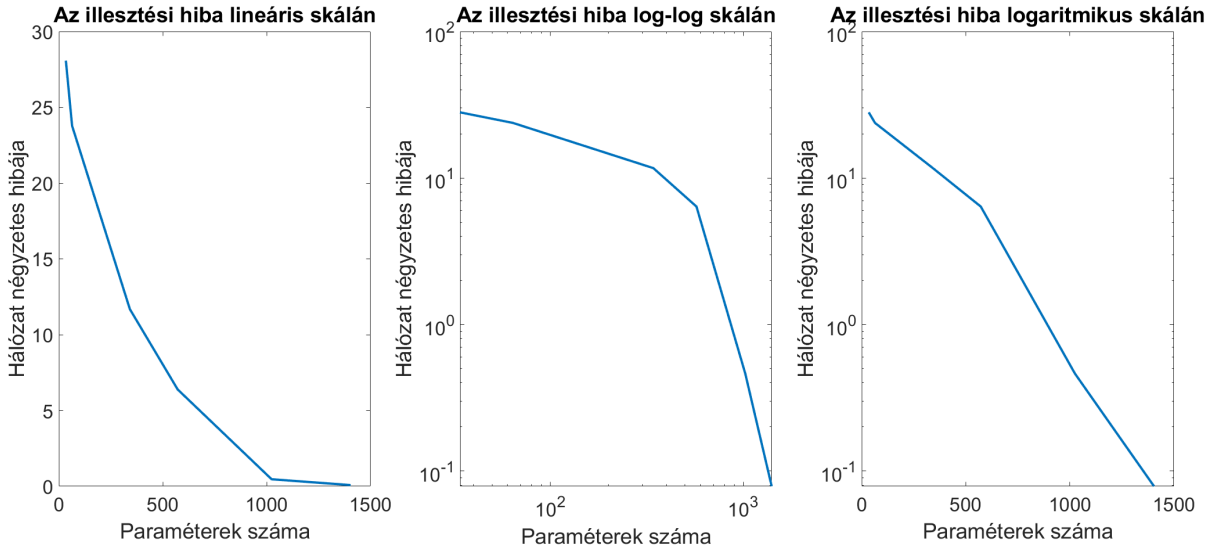
Érdekes és fontos kérdés, hogy exponenciális sebességgel is közelíthető-e egy adott f függvény valamilyen normában. Az erre vonatkozó elméletet nem tárgyaljuk, de rámutatunk, hogy az általunk írt eljárás, amelynek lényege egy hálózatnövelési technika (ld. 3.5 szakasz és 48. oldal) alkalmas lehet ilyen közelítés előállítására. A programban egyaránt növeljük a hálózat rétegeinek számát és az egyes rétegekben lévő neuronok számát. A közelítés hibáját nem célravezető a neuronok számában vizsgálni, mivel az egyes neuronok szerepe nem szimmetrikus, sőt nem is egyforma mennyiségű paraméterrel járulnak hozzá a hálózathoz. Célszerűbb tehát a paraméterszám függvényében vizsgálni a közelítés hibáját.

A programban a közelített függvényhez létrehozott hálózatot tapasztalatilag meghatározott nagyságú lépésekben növeljük úgy, hogy nagyságrendben hasonló hatékonyságbeli növekedést érjünk el a mélység és a szélesség (vagyis egy rétegen belüli maximális neuronszám) változtatásával. Ez azonban azt eredményezi, hogy viszonylag kevés diszkrét lépésben elérünk az optimális hálózati struktúrához, így kevés adat alapján tudunk csak a növekedés nagyságrendjére következtetni. A konvergencia rend vizsgálatához nem praktikus olyan egyszerű függvényt közelíteni, mint a fent szereplő hetedfokú polinom. Ugyanis a tapasztalatok azt mutatják, hogy megfelelő struktúrával már 30 paraméter is elég e célra, amelyet az algoritmus 3 lépésben megtalál. Tehát itt átskálázás segítségével sem kapunk megbízható információt a konvergenciarendről. Alkalmazzunk tehát egy bonyolultabb függvényt, melyet trigonometrikus függvények lineáris kombinációjából állítunk elő.



4.3.5. ábra. Hálózatépítési algoritmussal kapott realizáció grafikonok

Megjegyzendő, hogy az algoritmus eredményére hatványozottan teljesül a MATLAB betanítási algoritmusában levő véletlen paraméterek hatása. Előfordul, hogy ilyen eredményhez többszöri próbálkozással jutunk csak el. Érdemes lehet a minimális iterációszámot, illetve az egyes lépések neuronszám növekményeit feladatnak megfelelően akár tapasztalati úton finomhangolni. A dolgozatba általános esetben is viszonylag megfelelően működő paraméterekkel került be a kód. Meggondolható, hogy az utolsó iteráció neuronszámánál nem érdemes többet használni a közelítéshez, hiszen azzal a hálózatot túlillesztenénk, és rosszabb eredményt kapnánk. Most ellenőrizzük a négyzetes hiba alakulását a paraméterszám függvényében különböző skálákon.



4.3.6. ábra. Hálózatépítési algoritmus konvergenciája különböző skálákon.

Az eredményből láthatóan következtethetünk exponenciális konvergenciára. Térjünk most át magasabb dimenziós problémák tárgyalására.

5. fejezet

Többdimenziós függvények közelítése

5.1. Vektorértékű függvények esete

Az előző fejezetben megmutattuk, hogy valós függvények egy bizonyos terében tetszőlegesen jó közelítést tudunk találni ReLU hálózatok segítségével. Ezt az eredményt szeretnénk most általánosítani magasabb dimenziós esetre is. Vizsgáljuk meg, mi mondható, ha a közelítendő függvény képtere $\mathbb{R}^d$, ahol $d \geq 2$. Kézenfekvő megoldás, hogy ekkor a koordinátafüggvényeket külön-külön közelítjük egy-egy hálózattal, és ezen hálózatokból, mint építőelemekből alkotjuk meg a sokdimenziós képterű függvényt közelítő hálózatot. Megmutatjuk [17] alapján, hogyan lehet ReLU hálózatokat egymás mellé kapcsolva, az eredeti kimeneteket megtartva növelni a kimeneti dimenziót.

5.1.18 Állítás. *Legyen $L, N_0 \in \mathbb{N}$, és legyen $\Phi_1 = ((A_1^1, b_1^1), \ldots, (A_L^1, b_L^1))$ és $\Phi_2 = ((A_1^2, b_1^2), \ldots, (A_L^2, b_L^2))$ két ReLU hálózat L réteggel és N_0 bemeneti dimenzióval. Ekkor létezik $\widetilde{\Phi} = ((\widetilde{A}_1, \widetilde{b}_1), \ldots, (\widetilde{A}_L, \widetilde{b}_L))$ ReLU háló L réteggel és N_0 bemeneti dimenzióval, melyre $M(\widetilde{\Phi}) = M(\Phi^1) + M(\Phi^2)$ és $R(\widetilde{\Phi})(x) = \left(R(\Phi^1)(x), R(\Phi^2)(x)\right), \forall x \in \mathbb{R}^{N_0}$.*

Bizonyítás. Válasszuk meg $\widetilde{\Phi}$ rétegeit a következőképp:

$$\widetilde{A}_1 := \begin{pmatrix} A_1^1 \\ A_1^2 \end{pmatrix}, \quad \widetilde{b}_1 := \begin{pmatrix} b_1^1 \\ b_1^2 \end{pmatrix}, \quad \widetilde{A}_l := \begin{pmatrix} A_l^1 & 0 \\ 0 & A_l^2 \end{pmatrix}, \quad \widetilde{b}_l := \begin{pmatrix} b_l^1 \\ b_l^2 \end{pmatrix}, \forall\ 2 \leq l \leq L.$$

□

Az itt szereplő $\widetilde{\Phi}$-t a Φ^1 és Φ^2 neurális hálók *parallelizációjának* nevezzük, és $P(\Phi^1, \Phi^2)$-vel jelöljük. A fenti formulákban nem használtuk ki, hogy az aktivációs függvény ReLU, ezért az állítás bármilyen azonos aktivációs függvényekkel rendelkező hálózatpár esetén igaz. Lehetőség van különböző inputokkal rendelkező hálózatok összekapcsolására is, az első réteget is a többihez hasonlóan megválasztva. Ezt nevezik *teljes parallelizációnak*. Ez önmagában még nem garantálja nekünk a koordinátafüggvényekre illesztett hálózatok együttes használhatóságát, ugyanis a módszer csak egyforma mélységű hálózatokra működik. Ezért további meggondolásokra is szükség van: hogyan tehetünk különböző mélységű hálózatokat egyforma mélységűvé a realizációt nem változtatva meg.

5.1.19 Állítás. *$\forall L \in \mathbb{N}$ és $\forall k \in \mathbb{N}$ esetén $\exists\, \Phi_{k,L}^{Id}$ ReLU hálózat úgy, hogy $\Phi_{k,L}^{Id}$ mélysége L, bemeneti dimenziója k, és $M(\Phi_{k,L}^{Id}) \leq 2kL$, valamint $R(\Phi_{k,L}^{Id})(x) = x, \forall x \in \mathbb{R}^k$.*

Bizonyítás. Jelölje I_k a k dimenziós egységmátrixot. Ekkor

- $L = 1$ esetben legyen $\Phi_{k,1}^{\mathrm{Id}} = (I_k, 0)$.
- $L = 2$ esetben legyen $\Phi_{k,2}^{\mathrm{Id}} = ((A_1, b_1), (A_2, b_2))$, ahol

$$A_1 := \begin{pmatrix} I_k \\ -I_k \end{pmatrix}, \quad b_1 := 0, \quad A_2 := \begin{pmatrix} I_k & -I_k \end{pmatrix}, \quad b_2 := 0.$$

- $L \geq 3$ esetben legyen

$$\Phi_{k,L}^{\mathrm{Id}} := \left(\left(\begin{pmatrix} I_k \\ -I_k \end{pmatrix}, 0 \right), \underbrace{(I_{2_k}, 0), \ldots, (I_{2_k}, 0)}_{L-2}, \left(\begin{pmatrix} I_k & -I_k \end{pmatrix}, 0 \right) \right).$$

□

Ezzel beláttuk, hogy tetszőleges mélységű és dimenziójú ReLU hálózat adható identikus realizációval. Jellemzően ezt más aktivációs függvények esetén nem lehet elérni. Ezt fogjuk kombinálni a következő állítással a célunk érdekében:

5.1.20 Állítás. *Legyenek Φ^1, Φ^2 ReLU hálózatok rendre L_1, L_2 réteggel, valamint legyen $N_{L_2}^2 = N_0^1$, vagyis Φ^1 bemeneti dimenziója egyezzen meg Φ^2 kimeneti dimenziójával. Ekkor $\exists\, \widehat{\Phi}$ ReLU hálózat $L_1 + L_2$ réteggel, melyre $M(\widehat{\Phi}) \leq 2M(\Phi^1) + 2M(\Phi^2)$, továbbá $R(\widehat{\Phi}) = R(\Phi^1) \circ R(\Phi^2)$.*

Bizonyítás. Legyenek $\widehat{\Phi}$ paraméterei a következőek:

$$\widehat{\Phi} = \left((A_1^2, b_1^2), \ldots, (A_{L_2-1}^2, b_{L_2-1}^2), \left(\begin{pmatrix} A_{L_2}^2 \\ -A_{L_2}^2 \end{pmatrix}, \begin{pmatrix} b_{L_2}^2 \\ -b_{L_2}^2 \end{pmatrix} \right), \left(\begin{pmatrix} A_1^1 & -A_1^1 \end{pmatrix}, b_1^1 \right), (A_2^1, b_2^1), \ldots, (A_{L_1}^1, b_{L_1}^1) \right).$$

□

Az előbbi $\widehat{\Phi}$ hálót a Φ^1, Φ^2 hálók *konkatenációjának* nevezzük, és $\Phi^1 \odot \Phi^2$-vel jelöljük. Más aktivációs függvények esetében is van lehetőség hálózatok konkatenációjára, azonban az eredmény $L_1 + L_2 - 1$ réteg mélységű lesz, és más módszerrel juthatunk el hozzá. Mindezek felhasználásával bizonyítható már az alábbi állítás is:

5.1.21 Állítás. *Legyen $L_1, L_2, N_0 \in \mathbb{N}$, és legyen $\Phi_1 = ((A_1^1, b_1^1), \ldots, (A_{L_1}^1, b_{L_1}^1))$ és $\Phi_2 = ((A_1^2, b_1^2), \ldots, (A_{L_2}^2, b_{L_2}^2))$ két ReLU hálózat rendre L_1, L_2 réteggel és N_0 bemeneti dimenzióval. Ekkor létezik $\widetilde{\Phi} = ((\widetilde{A}_1, \widetilde{b}_1), \ldots, (\widetilde{A}_L, \widetilde{b}_L))$ ReLU háló $L = \max\{L_1, L_2\}$ réteggel és N_0 bemeneti dimenzióval, melyre $R(\widetilde{\Phi})(x) = \left(R(\Phi^1)(x), R(\Phi^2)(x)\right), \forall x \in \mathbb{R}^{N_0}$.*

Bizonyítás. Legyen például $L_1 < L_2$. Ekkor legyen

$$\widehat{\Phi}^1 = \Phi^1 \odot \Phi_{N_0, L_2 - L_1}^{\mathrm{Id}}, \qquad \widetilde{\Phi} = P(\widehat{\Phi}^1, \Phi^2).$$

Ekkor

$$M(\widetilde{\Phi}) \leq 4N_0(L_2 - L_1) + 2M(\Phi^1) + M(\Phi^2).$$

□

Az állításból fakadóan általánosíthatjuk ReLU hálózatok parallelizációját különböző mélységű hálózatokra is, a $P(\Phi^1, \Phi^2)$ jelölést megtartva. Ezzel beláttuk, hogy ReLU hálózat esetében többdimenziós problémákhoz is elég az egyes koordinátafüggvényeket külön hálózatokkal közelíteni, majd ezek parallelizációját használni a vizsgált függvényközelítésére. Az említett módszereket használják gyakorlatban bonyolult hálózatok egyszerűbb építőelemekből való konstruálására.

5.2. Többváltozós függvények esete

Most belátjuk, hogy L rétegű ReLU hálóval L-ben exponenciálisan közelíthető tetszőleges k változós analitikus függvény. Ráadásul elegendő $k+4$ *szélességű* hálózatot, vagyis rétegenként legfeljebb $(k+4)$ neuront használni. A [6] munka eredményeit használjuk fel, a bizonyításokat a forrásnál részletesebben kidolgozva, néhány helyen javítva. A vektorok innentől a szokásos vastagított betűs jelöléssel meg lesznek különböztetve. A $\varphi = \mathrm{ReLU}$ aktivációs függvény jelölést fenntartva bevezetjük a következő jelöléseket:

- Kettőspont jelölés alsó indexben: $x_{m:n} := \{x_i \in \mathbb{R} : i = m, \dots, n\}$, valamint $x_{m:n,p:q} := \{x_{i,j} \in \mathbb{R} : i = m, \dots, n; j = p, \dots, q\}$.
- Lineáris kombináció: $y \in \mathcal{L}(x_1, \dots, x_n) \Leftrightarrow \exists\, \beta_i \in \mathbb{R}$, hogy $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$. Itt $x_1, \dots, x_n$-re változóként tekintve, y ezeknek lesz egy elsőfokú polinomja.
- ReLU lineáris kombináció: $y \in \tilde{\mathcal{L}}(x_1, \dots, x_n) \Leftrightarrow y = \varphi(z); z \in \mathcal{L}(x_1, \dots, x_n)$. A ReLU hálózat egy rejtett rétegbeli neuronjának kimenete a bemeneteinek ReLU kombinációja. Itt a változókban elsőfokú polinomon alkalmazzuk a ReLU aktivációs függvényt. Nevezzük most az ilyet ReLU polinomnak. A célunk tehát egy k változós függvény közelítése ReLU polinomok kompozícióinak segítségével.

5.2.22 Definíció. *Legyen adott $f : (x_1, \dots, x_k) \to \mathbb{R}$ függvény. Ha léteznek $y_{1:L,1:M}$ változók úgy, hogy*

$$y_{1,m} \in \tilde{\mathcal{L}}(x_{1:k}), \quad y_{l+1,m} \in \tilde{\mathcal{L}}(x_{1:k}, y_{l,1:M}), \quad f \in \mathcal{L}(x_{1:k}, y_{1:L,1:M}), \tag{5.2.1}$$

ahol $m = 1, \dots, M; l = 1, \dots, L-1$, akkor azt mondjuk, hogy f eleme az $\mathcal{F}_{L,M}(\mathbb{R}^k)$ neurális háló osztálynak*, és $y_{1:L,1:M}$ a* rejtett változói*.*

Meggondolható, hogy a definíció alapján $\mathcal{F}_{L,M}$ elemei olyan függvények, amelyek módosított neurális hálónak is tekinthetőek. Ezek a szokásos előrecsatolt szerkezeten felül további kapcsolatokat tartalmaznak, mégpedig az input és az output van összekapcsolva minden egyes réteggel.

5.2.23 Állítás. *Egy $f \in \mathcal{F}_{L,M}(\mathbb{R}^k)$ függvény felírható egy $L+1$ rétegű, $M+k+1$ szélességű ReLU hálózat realizációjaként.*

Bizonyítás. Legyenek $y_{1:L,1:M}$ az f rejtett változói, melyek teljesítik (5.2.1)-et, és legyen

$$f = \alpha_0 + \sum_{i=1}^{k} \alpha_i x_i + \sum_{l=1}^{L} \sum_{m=1}^{M} \beta_{l,m} y_{l,m}.$$

Célunk olyan $h_{1:L,1:M+k+1}$ helyettesítő változók bevezetése, mellyel

$$h_{1,m} \in \tilde{\mathcal{L}}(x_{1:k}); \quad h_{l+1,m} \in \tilde{\mathcal{L}}(h_{l,1:M+k+1}); \quad f \in \mathcal{L}(h_{L,1:M+k+1}), \tag{5.2.2}$$

$\forall\, m = 1, \ldots, M+k+1$; és $l = 1. \ldots, L-1$ esetén. Ehhez megfelelő választás:

$$h_{l,1:M} = y_{l,1:M}, \quad h_{l,M+1:M+k} = x_{1:k},$$

ha $l = 1, \ldots, L$, továbbá

$$h_{1,M+k+1} = \alpha_0 + \sum_{i=1}^{k} \alpha_i x_i, \quad h_{l+1,M+k+1} = h_{l,M+k+1} + \sum_{m=1}^{M} \beta_{l,m} h_{l,m},$$

ha $l = 1, \ldots, L-1$. Meggondolható, hogy ezekre teljesül (5.2.2). □

5.2.24 Állítás. *A neurális háló osztályokra teljesülnek a következő állítások:*

- $\mathcal{F}_{L_1,M}(\mathbb{R}^k) + \mathcal{F}_{L_2,M}(\mathbb{R}^k) \subseteq \mathcal{F}_{L_1+L_2,M}(\mathbb{R}^k)$, *vagyis*

$$f_1 \in \mathcal{F}_{L_1,M}(\mathbb{R}^k), f_2 \in \mathcal{F}_{L_2,M}(\mathbb{R}^k) \Rightarrow f_1 + f_2 \in \mathcal{F}_{L_1+L_2,M}(\mathbb{R}^k) \tag{5.2.3}$$

- $\mathcal{F}_{L_2,M}(\mathbb{R}^{k+1}) \circ \mathcal{F}_{L_1,M+1}(\mathbb{R}^k) \subseteq \mathcal{F}_{L_1+L_2,M+1}(\mathbb{R}^k)$, *vagyis*

$$\begin{aligned} &f_1(x_1, \ldots, x_k) \in \mathcal{F}_{L_1,M+1}(\mathbb{R}^k), f_2(y, x_1, \ldots, x_k) \in \mathcal{F}_{L_2,M}(\mathbb{R}^{k+1}) \Rightarrow \\ &f_2(f_1(x_1, \ldots, x_k), x_1, \ldots, x_k) \in \mathcal{F}_{L_1+L_2,M+1}(\mathbb{R}^k) \end{aligned} \tag{5.2.4}$$

5.2.25 Definíció. *Legyen $g(\boldsymbol{x}), \boldsymbol{x} \in [-1,1]^k$ folytonos függvény, továbbá $\mathcal{F}$ a $[-1,1]^k$ halmazon folytonos függvények egy osztálya, amelyről minden esetben feltesszük, hogy egy függvénnyel együtt annak minden konstansszorosa is az osztályban van. Ekkor a kettő L_∞-távolsága legyen:*

$$dist(g, \mathcal{F}) = \inf_{f \in \mathcal{F}} \max_{\boldsymbol{x} \in [-1,1]^k} |g(\boldsymbol{x}) - f(\boldsymbol{x})|. \tag{5.2.5}$$

5.2.26 Állítás. *A bevezetett távolságfogalomra az alábbiak teljesülnek:*

- *Legyenek g_1, g_2 folytonos függvények, és $\mathcal{F}_1, \mathcal{F}_2$ folytonos függvények osztályai. Ekkor*

$$dist(\alpha_1 g_1 + \alpha_2 g_2, \mathcal{F}_1 + \mathcal{F}_2) \leq |\alpha_1| dist(g_1, \mathcal{F}_1) + |\alpha_2| dist(g_2, \mathcal{F}_2), \tag{5.2.6}$$

  ahol $\alpha_1, \alpha_2 \in \mathbb{R}$.

- *Legyen $g_1 : [-1,1]^k \to [-1,1]$ folytonos és $g_2 : [-1,1]^{k+1} \to [-1,1]$ folytonos, valamint az első változója szerint Lipschitz-folytonos is. Legyenek ismét $\mathcal{F}_1$ és $\mathcal{F}_2$ az $([-1,1]^k)$, illetve $([-1,1]^{k+1})$ halmazokon értelmezett folytonos függvények egy-egy osztálya. Ekkor*

$$dist(g_2(g_1(\boldsymbol{x}), \boldsymbol{x}), \mathcal{F}_2 \circ \mathcal{F}_1) \leq L_{g_2} dist(g_1, \mathcal{F}_1) + dist(g_2, \mathcal{F}_2), \tag{5.2.7}$$

  ahol L_{g_2} a g_2 függvény első változója szerinti Lipschitz-konstans.

Az előző két állítás bizonyítása megtalálható [6]-ban. Most rátérünk az alfejezet elején megfogalmazottak tételbe foglalására, és annak bizonyítására.

5.2.27 Tétel. *Legyen f egy analitikus függvény a $[-1,1]^k$, $(k \geq 2)$ halmazon, azaz hatványsora itt abszolút konvergens. Ekkor $\forall\ \delta > 0\ \exists\ \hat{f}$, amely megadható egy L mélységű és $k+4$ szélességű ReLU hálózat realizációjaként és minden $\boldsymbol{x} \in [-1+\delta, 1-\delta]^k$ esetén teljesül, hogy*

$$|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})| < 2 \sum_{\boldsymbol{n} \in \mathbb{N}^k} |a_{\boldsymbol{n}}| \cdot \exp\left(-k\delta\left(e^{-1} L^{\frac{1}{2k}} - 1\right)\right). \tag{5.2.8}$$

A tétel bizonyításához szükségünk van további lemmákra.

5.2.28 Lemma. *A $g : [-1,1] \to \mathbb{R}$, $g(x) = x^2$ függvény a rétegek L száma szerint exponenciális sebességgel közelíthető mély neurális hálók segítségével, azaz:*

$$dist(g, \mathcal{F}_{L,2}) \leq 2^{-2L}. \tag{5.2.9}$$

Bizonyítás. Elég találni tehát olyan $f \in \mathcal{F}_{L,2}$ függvényt, melyre

$$|x^2 - f(x)| \leq 2^{-2L}, \quad x \in [-1,1].$$

Vegyük a következő $g : [0,1] \to [0,1]$ ún. *fűrészfog* függvényt:

$$g(y) = 2y - 4\varphi\left(y - \frac{1}{2}\right) = \begin{cases} 2y, & \text{ha } 0 \leq y \leq \frac{1}{2}, \\ 2 - 2y, & \text{ha } \frac{1}{2} \leq y \leq 1. \end{cases} \tag{5.2.10}$$

Könnyen ellenőrizhető (ld. [22]), hogy a $g^l(x) := \underbrace{g \circ \cdots \circ g}_{l}(x)$ kompozíció az (5.2.1a) ábrán látható alakú.
Az is belátható [24] alapján, hogy az

$$f_m(x) = x - \sum_{s=1}^{m} \frac{g^s(x)}{2^{2s}},\ x \in [0,1]$$

alakban definiált függvény éppen az x^2 függvényt a $[0,1]$ intervallumon (2^m+1) ekvidisztáns alapponton interpoláló függvény (lásd 5.2.1b ábra). Erre teljesül, hogy

$$|f_m(x) - x^2| \leq 2^{-2(m+1)}.$$

Ezért elég megmutatni, hogy $f(x) := f_{L-1}(|x|) \in \mathcal{L}(y_{1:L,1:2})$, ahol $y_{1:L,1:2}$ az (5.2.1) formula szerinti. Legyenek f rejtett változói

$$\begin{aligned} y_{1,1} &= \varphi(x), & y_{1,2} &= \varphi(-x), \\ y_{2,1} &= \varphi(y_{1,1} + y_{1,2}), & y_{2,2} &= \varphi\left(y_{1,1} + y_{1,2} - \frac{1}{2}\right), \\ y_{l+1,1} &= \varphi(2y_{l,1} - 4y_{l,2}), & y_{l+1,2} &= \varphi\left(2y_{l,1} - 4y_{l,2} - \frac{1}{2}\right), \end{aligned}$$

ahol $l = 2, \dots, L-1$. Látható, hogy $y_{1,1} + y_{1,2} = |x| = y_{2,1}$, valamint $y_{2,2} = \varphi(|x| - \frac{1}{2})$, ezért $g(|x|) = 2y_{2,1} - 4y_{2,2}$. Indukcióval megmutatjuk, hogy

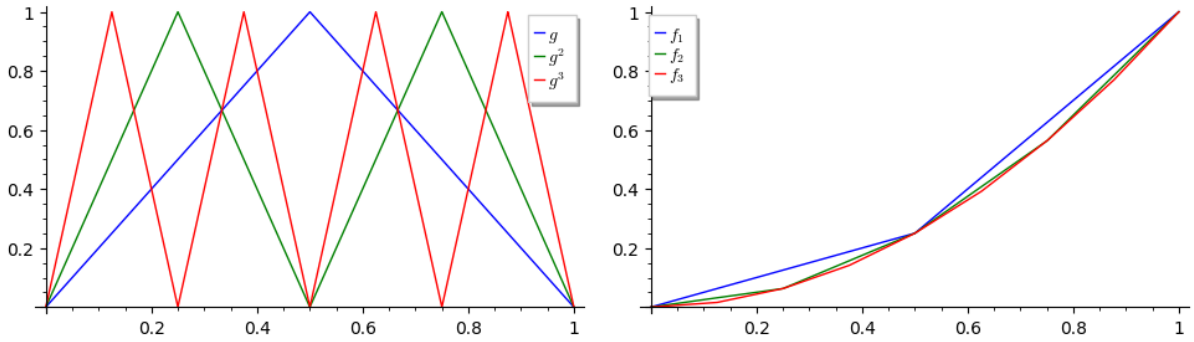
$$g^{l-1}(|x|) = 2y_{l,1} - 4y_{l,2};\ l = 2, \dots, L.$$

Ezt $l = 1$-re láttuk, most tegyük fel, hogy $m-1$-ig teljesül. Ekkor rendre az indukciós feltevést, a g-re adott (5.2.10) formulát, valamint $y_{l+1,1}$ és $y_{l+2,1}$ rekurzív definícióját felhasználva kapjuk, hogy

$$g^m(|x|) = g(g^{m-1}(|x|)) = g(2y_{m,1} - 4y_{m,2}) =$$

$$= 2(2y_{m,1} - 4y_{m,2}) - 4\varphi\left(2y_{m,1} - 4y_{m,2} - \frac{1}{2}\right) = 2y_{m+1,1} - 4y_{m+1,2}.$$

□



(a) Fűrészfog függvény és kompozíciói önmagával.

(b) Az f_m függvények.

5.2.1. ábra. Az x^2 függvény közelítése SageMath segítségével

5.2.29 Lemma. *A $g(x,y) = xy$; $x, y \in [-1,1]$ L-ben szintén exponenciális sebességgel közelíthető:*

$$dist(g, \mathcal{F}_{3L,2}) \leq 3 \cdot 2^{-2L}. \tag{5.2.11}$$

Bizonyítás. Vegyük észre, hogy

$$g(x,y) = xy = 2\left(\frac{x+y}{2}\right)^2 - \frac{1}{2}x^2 - \frac{1}{2}y^2 = 2h\left(\frac{x+y}{2}\right) - \frac{1}{2}h(x) - \frac{1}{2}h(y),$$

ahol (5.2.9) miatt $\exists\ f_h \in \mathcal{F}_{L,2}$, hogy $\|h - f_h\|_\infty \leq 2^{-2L}$. Ekkor

$$f_g(x,y) := 2f_h\left(\frac{x+y}{2}\right) - \frac{1}{2}f_h(x) - \frac{1}{2}f_h(y)$$

jelöléssel (5.2.3) és a háromszög egyenlőtlenség miatt $f_g \in \mathcal{F}_{3L,2}$ és $\|g - f_g\|_\infty \leq 3 \cdot 2^{-2L}$ teljesül. □

5.2.30 Lemma. *Egy k változóban p-edfokú $M_p(\boldsymbol{x})$ monom is közelíthető L-ben exponenciálisan, azaz*

$$dist(M_p(\boldsymbol{x}), \mathcal{F}_{3(p-1)L,\,3}) \leq 3(p-1)\cdot 2^{-2L}. \tag{5.2.12}$$

Bizonyítás. Legyen $M_p(\boldsymbol{x}) = x_{i_1}x_{i_2}\dots x_{i_p}; i_1,\dots,i_p \in \{1,\dots,k\}$. Alkalmazzunk teljes indukciót. A $p=2$ eset következik (5.2.11)-ből, így most tegyük fel, hogy $q \in \mathbb{N}$-ig teljesül az állítás, és vizsgáljuk meg a $q+1$ esetet. Legyen $g(x,y) = xy$ és legyen $M_{q+1}(\boldsymbol{x}') = M_q(\boldsymbol{x})x_{i_{q+1}} = g(M_q(\boldsymbol{x}), x_{i_{q+1}})$. Ekkor rendre az (5.2.4), (5.2.7) és (5.2.11) becslések, végül az indukciós feltétel felhasználásával kapjuk, hogy

$$dist(M_{q+1}(\boldsymbol{x}'), \mathcal{F}_{3qL,\,3}) \leq dist(g(M_q(\boldsymbol{x}), x_{i_{q+1}}), \mathcal{F}_{3L,2}\circ\mathcal{F}_{3(q-1)L,\,3}) \leq$$
$$L_g dist(M_q, \mathcal{F}_{3(q-1)L,3}) + dist(g, \mathcal{F}_{3L,2}) \leq 3q\cdot 2^{-2L},$$

ahol azt is figyelembe vettük, hogy $L_g = 1$ a $[-1,1]$ intervallumon. □

5.2.31 Lemma. *Egy k változós p-edfokú $P_p(\boldsymbol{x}) = \sum_{|\boldsymbol{n}|\leq p} a_{\boldsymbol{n}}\boldsymbol{x}^{\boldsymbol{n}}$, $\boldsymbol{x}\in[-1,1]^k$; $\boldsymbol{n} = (n_1,\dots,n_k)\in\mathbb{N}^k$ polinom is közelíthető L-ben exponenciálisan:*

$$dist\left(P_p(\boldsymbol{x}), \mathcal{F}_{3\binom{p+k}{k}(p-1)L,\,3}\right) < 3(p-1)\cdot 2^{-2L}\sum_{|\boldsymbol{n}|\leq p}|a_{\boldsymbol{n}}| \tag{5.2.13}$$

Bizonyítás. Meggondolható, hogy a legfeljebb p fokszámú k változós monomok száma egyenlő a pontosan p fokszámú $k+1$ változós monomok számával, hiszen valamely változó helyére 1-et helyettesítve bijekciót hozunk létre a két halmaz között. Ezek száma pedig $(k+1)$ elem p-ed osztályú ismétléses kombinációinak száma, vagyis $\binom{(k+1)+p-1}{p} = \binom{p+k}{k}$. Ennélfogva bármely p fokszámú k változós polinom legfeljebb $\binom{p+k}{k}$ darab különböző monom összegeként áll elő. Ezt felhasználva az állítás már bizonyítható az egyes becslésekben rendre az (5.2.3), (5.2.6) és (5.2.12) egyenlőtlenségeket felhasználva:

$$\text{dist}\left(P_p(\boldsymbol{x}), \mathcal{F}_{3\binom{p+k}{k}(p-1)L,\,3}\right) \leq \text{dist}\left(\sum_{\boldsymbol{n}\leq p} a_{\boldsymbol{n}}\boldsymbol{x}^{\boldsymbol{n}},\ \sum_{|\mathbf{n}|\leq p}\mathcal{F}_{3(p-1)L,\,3}\right) \leq$$
$$\sum_{|\mathbf{n}|\leq p}|a_{\boldsymbol{n}}|\text{dist}\left(\boldsymbol{x}^{\boldsymbol{n}}, \mathcal{F}_{3(p-1)L,\,3}\right) \leq 3(p-1)\cdot 2^{-2L}\sum_{|\mathbf{n}|\leq p}|a_{\boldsymbol{n}}|.$$

□

Ezek segítségével már bizonyítható az analitikus függvények 5.2.27 közelítési tétele:
Bizonyítás. Legyen a közelítendő f analitikus függvény a következő alakú:

$$f(\boldsymbol{x}) = P_p(\boldsymbol{x}) + R(\boldsymbol{x}) := \sum_{|\boldsymbol{n}|\leq p} a_{\boldsymbol{n}}\boldsymbol{x}^{\boldsymbol{n}} + \sum_{|\boldsymbol{n}|>p} a_{\boldsymbol{n}}\boldsymbol{x}^{\boldsymbol{n}},$$

ahol feltehető, hogy $\sum_{\mathbf{n}\in\mathbb{N}^k}|a_{\mathbf{n}}| = 1$. Célunk tehát találni egy $\hat{f}$ függvényt, melyre

$$\|f-\hat{f}\|_\infty < 2\varepsilon,$$

ahol $\varepsilon := \exp\left(-k\delta\left(e^{-1}L^{\frac{1}{2k}}-1\right)\right)$, vagyis $L = \left[e\left(\frac{1}{k\delta}\log\frac{1}{\varepsilon}+1\right)\right]^{2k}$.

Ekkor $\boldsymbol{x} \in [-1+\delta, 1-\delta]^k$ esetén az $|\boldsymbol{n}| > p$ becslést, majd az $a_{\boldsymbol{n}}$ együtthatók összegére vonatkozó feltevést használva

$$|R(\boldsymbol{x})| = |\sum_{|\boldsymbol{n}|>p} a_{\boldsymbol{n}} \boldsymbol{x}^{\boldsymbol{n}}| < \sum_{|\boldsymbol{n}|>p} |a_{\boldsymbol{n}}|(1-\delta)^p \leq (1-\delta)^p,$$

ahonnan a $p = \frac{1}{\delta}\log\frac{1}{\varepsilon}$ választással

$$|R(\boldsymbol{x})| < (1-\delta)^{\frac{1}{\delta}\log\frac{1}{\varepsilon}} < e^{-log\frac{1}{\varepsilon}} < \varepsilon$$

adódik. Tudjuk az (5.2.13) becslésből, hogy $dist\,(P_p, \mathcal{F}_{L,3}) < 3(p-1)\cdot 2^{-2L'}$, ahol

$$L' = \frac{1}{3}\cdot\frac{L}{p-1}\binom{p+k}{p}^{-1} > \frac{L}{p}\left[\frac{(p+k)^k}{(k/e)^k}\right]^{-1} = L\left[e\left(\frac{1}{k\delta}\log\frac{1}{\varepsilon}+1\right)\right]^{-k}\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)^{-1} =$$
$$= \left[e\left(\frac{1}{k\delta}\log\frac{1}{\varepsilon}+1\right)\right]^{k}\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)^{-1}, \tag{5.2.14}$$

ha $k \geq 2$. Az első becslésnél kihasználtuk, hogy

$$\frac{p!k!}{(p+k)!} = \frac{k!}{(p+1)\ldots(p+k)} > \frac{k!}{(p+k)^k} > 3\frac{\left(\frac{k}{e}\right)^k}{(p+k)^k},$$

ahol az utolsó egyenlőtlenség a Stirling formula és a $k \geq 2 \Rightarrow \sqrt{2k\pi} > 3$ összefüggés következménye. A második egyenlőséget $p = \frac{1}{\delta}\log\frac{1}{\varepsilon}$ behelyettesítéssel kaptuk, ahol

$$\frac{p+k}{k} = \frac{\frac{1}{\delta}\log\frac{1}{\varepsilon}+k}{k} = \frac{1}{k\delta}\log\frac{1}{\varepsilon}+1.$$

Ezután behelyettesítettük a bizonyítás elején ε függvényében L-re adott képletet. A fenti (5.2.14) becslést folytatva a binomiális tétel alapján csak egyetlen tagot veszünk, az első tényezőt elhagyjuk, majd kihasználjuk, hogy $k \geq 2$, amiből kapjuk, hogy

$$\left[e\left(\frac{1}{k\delta}\log\frac{1}{\varepsilon}+1\right)\right]^{k}\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)^{-1} > e^k\cdot\left(\frac{1}{k\delta}\log\frac{1}{\varepsilon}\right)^{k}\cdot\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)^{-1} >$$
$$> e^k\cdot\left(\frac{1}{k\delta}\log\frac{1}{\varepsilon}\right)^{2}\cdot\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)^{-1} = \frac{e^k}{k^2}\cdot\frac{1}{\delta}\log\frac{1}{\varepsilon} > \frac{1}{\delta}\log\frac{1}{\varepsilon} \gg \log\frac{1}{\varepsilon}+\log\frac{1}{\delta},$$

ahol az utolsó egyenlőtlenség akkor teljesül, ha $\varepsilon \ll 1$ fennáll. Ez könnyen látható, ha mindkét oldal exponenciális függvényét vesszük:

$$\exp\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right) = \left(\frac{1}{\varepsilon}\right)^{\frac{1}{\delta}} \gg \frac{1}{\varepsilon}\cdot\frac{1}{\delta} = \exp\left(\log\frac{1}{\varepsilon}+\log\frac{1}{\delta}\right).$$

Kihasználva az L-re kapott becslést adódik, hogy

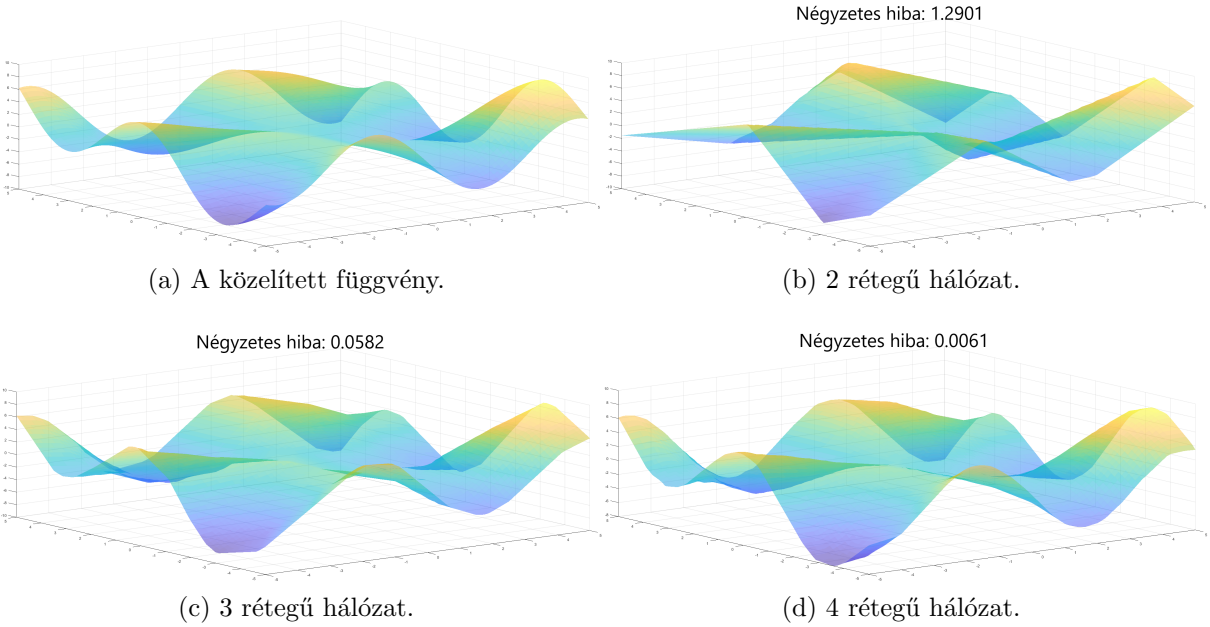
$$dist(P_p, \mathcal{F}_{L,3}) < 3(p-1)\cdot 2^{-2L'} \ll p\cdot 2^{-2\left(\log\frac{1}{\varepsilon}+\log\frac{1}{\delta}\right)} = \frac{1}{\delta}\log\frac{1}{\varepsilon}\cdot 2^{\log\delta^2\varepsilon^2} <$$
$$< \frac{1}{\delta}\log\frac{1}{\varepsilon}\cdot\delta^2\varepsilon^2 < \varepsilon^2\log\frac{1}{\varepsilon} < \varepsilon.$$

Ekkor tehát $\exists\ \hat{f} \in \mathcal{F}_{L,3}$ úgy, hogy $\|P_p - \hat{f}\|_\infty < \varepsilon$, és

$$\|f - \hat{f}\|_\infty \leq \|f - P_p\|_\infty + \|P_p - \hat{f}\|_\infty < 2\varepsilon.$$

□

Vizsgáljuk meg, hogy a tapasztalati eredmények visszatükrözik-e az elméleti exponenciális konvergencia-sebességet. Ehhez közelítsük az $f(x, y) = y \sin x - x \cos y$ függvényt a $[-5, 5]^2$ halmazon ReLU hálózattal. A hálózat szélessége legyen 10, teszteljük a $2, 3$ és 4 mélységű hálózatokat.



(a) A közelített függvény.

(b) 2 rétegű hálózat.

(c) 3 rétegű hálózat.

(d) 4 rétegű hálózat.

5.2.2. ábra. Térgörbe közelítés hibája mély ReLU hálózattal.

Több réteggel nincs értelme próbálkozni, mivel a függvény a 4 rétegű hálózat paraméter számával is jól leírható. Ennél több réteg használata túlillesztéshez vezet. A felhasznált szabad paraméterek száma visszakövetkeztethető az egyes háló struktúrák definiálásának módjából. Így meggondolható, hogy $2, 3$ és 4 réteg esetén rendre $41, 151$ és 261 paramétert alkalmaztunk. Fontos, hogy a kimondott közelítési tételek elsősorban nem abban segítenek bennünket, hogy eldöntsük, az aktuálisan használt közelítésünk hatékonyságát érdemes-e megpróbálni javítani több paraméter használatával. Ugyanis, mint korábban tárgyaltuk, a problémához képest túl sok paraméter alkalmazása több hátránnyal jár, mint előnnyel. Az eredmények fontossága legfőképpen abból származik, hogy információt nyerünk a jelenlegi paraméterszámmal elérhető legjobb közelítés hibájának nagyságrendjéről. Így pontosabb képet kapunk arról, a közelítésünk közel van-e a kritériumfüggvény globális minimumához, vagy érdemes a hálózatot más módszerekkel újra tanítani.

Következményként megkaptuk tehát, hogy a többrétegű perceptron *univerzális approximátor*, azaz bármilyen (esetünkben analitikus) függvényt tetszőlegesen jól tudunk vele közelíteni, ráadásul rétegek számában exponenciális sebességgel. Mindezt úgy kapjuk, hogy az általunk alkalmazott aktivációs függvény nem is deriválható. Ez bepillantást enged számunkra a neurális hálózatok megdöbbentő kifejezőerejébe. Összevetve a 4. és az 5. fejezet végkicsengését felmerül a kérdés, a hálózatok mélységének vagy szélességének növelése kritikusabb-e a teljesítmény érdekében. Ebben a szakirodalom megosztott (lásd [19], [23]), azonban láthatóan mindkétféleképpen célt érhet, aki a problémákat a neurális hálók nyújtotta eszköztárral kívánja megoldani.

Készített segédprogramok

Legjobb rögzített alappontú lineáris spline közelítés

```
function fixedNodeNet = linSplineInterp(v, nodes, needToTrain)
% linSplineInterp: Megkonstruálunk egy neurális hálót 2
% réteggel, amelynek realizációja szakaszonként lineárisan
% interpolálja a bemeneti v függvényt a rögzített alappont
% halmazon (nodes), és visszaadunk egy neurális hálót
% ilyen realizációval. Ha szükséges, betanítás segítségével
% létrehozzuk a legjobb rögzített alappontú közelítést
% és összehasonlítjuk a közelítések hatékonyságát.
%   Bemenetek:  - v:     Bármilyen valós függvény
%               - nodes: az interpolációs alappontok halmaza
%               - needToTrain: logikai változó, a hálózat
%                   betanítás szükségességét határozza meg
%   Kimenet:
%       - fixedNodeNet: v függvényen lineáris spline
%         módon interpoláló neurális háló

% Meghatározzuk az osztópontok által meghatározott szakaszok
% számát
n = length(nodes)-1;
% A 4.1.10 tétel bizonyításában adott konstrukció megvalósítása
A1 = ones(n,1);
b1 = -nodes(1:end-1)';
A2 = zeros(1,n);
A2(1)=(v(nodes(2))-v(nodes(1)))/(nodes(2)-nodes(1));
for i=3:n+1
    A2(i-1) = (v(nodes(i))-v(nodes(i-1)))/(nodes(i)-nodes(i-1))
        - (v(nodes(i-1))-v(nodes(i-2)))/(nodes(i-1)-nodes(i-2));
end
b2 = v(nodes(1));

% Létrehozunk egy függvény közelítésre alkamas neurális hálót,
% melynek egy "n" neuronból álló rejtett rétege van
fixedNodeNet = fitnet(n);
% Az adatok feldolgozását eltöröljük
```

```
34 fixedNodeNet.input.processFcns = {'removeconstantrows'};
35 fixedNodeNet.output.processFcns = {'removeconstantrows'};
36 % Beállítjuk a ReLU aktivációs függvényt a rejtett rétegben
37 fixedNodeNet.Layers{1}.transferFcn = 'poslin';
38 % A kiértékelt adatokat használva beállítjuk a hálózat
39 % mátrix-vektor dimenzióit, vagyis a hálózat konfigurációját
40 fixedNodeNet = configure(fixedNodeNet,nodes,v(nodes));
41 % Beállítjuk a korábban kiszámolt súlyokat
42 fixedNodeNet.IW{1,1} = A1;
43 fixedNodeNet.LW{2,1} = A2;
44 fixedNodeNet.b{1} = b1;
45 fixedNodeNet.b{2} = b2;
46
47 % Megvizsgáljuk, szükség van-e betanításra
48 if(needToTrain)
49 % Lemásoljuk a hálózatot
50     net = fixedNodeNet;
51 % Rögzítjük a rejtett réteg súlyait, hogy
52 % a töréspontok helye ne változzon meg
53     net.inputWeights{1,1}.learn = 0;
54     net.biases{1}.learn = 0;
55     % Az alappontok intervallumán adunk finom felosztást
56     x = (nodes(1):0.01:nodes(n+1));
57 % Kiértékeljük a v függvényt a felosztásunkon (kirajzoláshoz)
58     vx = v(x);
59 % Betanítjuk a hálózatot a függvény kiértékeléseit használva
60     net = train(net,x,vx);
61 % Kiértékeljük a hálózatokat a felosztáson
62     fixedSplineX = fixedNodeNet(x);
63     netX = net(x);
64 % Kiszámítjuk a finom felosztáson a hálózatok négyzetes hibáját
65     perf = perform(fixedNodeNet,vx,fixedSplineX);
66     perf2 = perform(net,vx,netX);
67 % Kirajzoljuk a grafikonokat
68     plot(x, vx, 'b', x, fixedSplineX, 'r', x, netX, 'g', '
       lineWidth', 4)
69     hold on
70     t = title({['Interpolációs négyzetes hiba: ',num2str(perf)
       ], ['Közelítés négyzetes hibája: ', num2str(perf2)]});
71     t.FontSize = 26;
72     l = legend({'Eredeti v függvény','Interpolációs konstrukció
       ','Betanított hálózat'},'Location','southeast');
73     l.FontSize = 26;
74 end
75 end
```

Univerzálisan legjobb lineáris spline közelítés

```
function freeNodeNet = bestLinearSplineApproximation(v, nodes)
% bestLinearSplineApproximation: Egy adott függvényre
% meghatározzuk a közel legjobb lineáris spline közelítést,
% és visszaadunk egy neurális hálót ilyen realizációval.
%   Bemenetek: - v:     Bármilyen valós függvény
%              - nodes: kezdeti interpolációs alappont halmaz
%   Kimenet:
%       - freeNodeNet: A legjobb közelítés hálója

n = length(nodes)-1;
% Kiindulunk a v függvényen rögzített pontokon (nodes)
% interpoláló neurális hálóból
fixedNodeNet = linSplineInterp(v, nodes, false);
% Lemásoljuk a hálózatot
freeNodeNet = fixedNodeNet;
% A rejtett réteg súlyait nem frissítjük, így
% a torzítás vektorából visszanyerhetjük az alappontokat
freeNodeNet.inputWeights{1,1}.learn = 0;
% Az alappontok intervallumán adunk finom felosztást
x = (nodes(1):0.01:nodes(end));
% Kiértékeljük a v függvényt a felosztásunkon (kirajzoláshoz)
vx = v(x);
% Betanítjuk a hálózatot a függvény kiértékeléseit használva
freeNodeNet = train(freeNodeNet,x,vx);
% Kiértékeljük a hálózatokat a felosztáson
fixedSplineX = fixedNodeNet(x);
freeSplineX = freeNodeNet(x);
% Kiszámítjuk a finom felosztáson a hálózatok négyzetes hibáját
perf = perform(fixedNodeNet,vx,fixedSplineX);
perf2 = perform(freeNodeNet,vx,freeSplineX);
% Kirajzoljuk a grafikonokat
plot(x, vx, 'b', x, fixedSplineX, 'r', x, freeSplineX, 'g', '
    lineWidth', 4)
hold on
plot(x,vx)
hold off
t = title({['Interpolációs négyzetes hiba: ', num2str(perf)], [
    'Legjobb közelítés négyzetes hibája: ', num2str(perf2)]});
t.FontSize = 26;
l = legend({'Eredeti v függvény','Interpolációs konstrukció','
    Legjobb lineáris spline közelítés'},'Location','northeast');
l.FontSize = 26;
end
```

Hálózatépítés hálónövelés módszerrel

```
function netw = FindNetworkForFunction(f, a, b)
% FindNetworkForFunction: Visszatérünk egy neurális hálóval,
% mely az f függvényt az [a,b] intervallumon közelíti.
% Bemenetek:     - f: valós függvény
%                - a,b: valós számok, melyre a < b, ezek
%                     az intervallum végpontjai
% Kimenet:       - netw: Neurális háló, amely az f függvényt
%                      az [a,b] intervallumon jól közelíti

% Kiértékeljuk a függvényt az intervallum egy felosztásán.
x = linspace(a, b, (b-a)*5);
y = f(x);
% Igény szerint változtatható paraméterek:
% Súlyokon alkalmazott lépésköz, az iterációk minimális száma
depthWeightStep = ceil(length(x)/25);
widthWeightStep = ceil(length(x)/15);
minNumOfIterations = 4;
% A köztes lépések neurális hálóinak x vektor pontjain való
% kiértékeléseit tároljuk majd el a values mátrix oszlopaiban
values = zeros(length(x), 30);
% Az aktuálisan legjobb neurális háló kimeneti rétegének súlya
% és rétegeinek száma, illetve a súlyokat tartalmazó vektor
numOfWeights = depthWeightStep;
numOfLayers = 1;
weights = widthWeightStep;
weightmatrix = zeros(30,30);
layerMatrix = [];
% A hiba inicializálása valamilyen nagy kezdeti értékre
error = 100000;
for i=1:30
    % készítünk két új neurális hálót: az egyikhez
    % hozzáadunk egy új réteget, a másikban
    % pedig megnöveljük az utolsó rejtett réteg méretét
    weights1=[numOfWeights, weights];
    net1 = fitnet(weights1);
    weights(1)=weights(1)+widthWeightStep;
    net2 = fitnet(weights);
    net1 = train(net1,x,y);
    net2 = train(net2,x,y);
    % Kiszámítjuk mindkét esetben a hiba értéket
    net1X = net1(x);
    net2X = net2(x);
    error1 = perform(net1,y,net1X);
    error2 = perform(net2,y,net2X);
```

```
45      % Ha egyik esetben sem érunk el javulást, akkor megállunk
46      if min(error1,error2)>error && i>minNumOfIterations
47          ind=i-1;
48          break;
49      else
50          % Frissítjük az aktuális hibavektort
51          error = min(error1,error2);
52      end
53      % A jobb eredményt nyújtó háló adatait tartjuk meg
54      if error1 < error2
55          weights = weights1;
56          numOfLayers = numOfLayers+1;
57          values(:,i)=net1X;
58          netw=net1;
59      else
60          weights
61          numOfWeights = numOfWeights + depthWeightStep;
62          values(:,i)=net2X;
63          netw=net2;
64      end
65      weightmatrix(i,1:numOfLayers) = weights;
66      layerMatrix = [layerMatrix, numOfLayers];
67  end
68  % Kirajzoljuk a folyamat közben használt hálók grafikonjait
69  for i=1:ind
70      subplot(ceil(ind/2),2,i);
71      hold on
72      plot(x,f(x),'rO')
73      plot(x,values(:,i));
74      title(['The layer structure: ', mat2str([weightmatrix(i,1:
            layerMatrix(i)),1])])
75      hold off
76  end
77  end
```

Irodalomjegyzék

[1] e. a. Altrichter Márta, Horváth Gábor. *Neurális hálózatok*. Panem, 2006. Budapest.

[2] M. Azimi-Sadjadi, S. Sheedvash, and F. Trujillo. Recursive dynamic node creation in multilayer neural networks. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 4:242–56, 02 1993.

[3] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Rev.*, 60(2):223–311, 2018.

[4] S. Cai, J. Gao, M. Zhang, W. Wang, G. Chen, and B. C. Ooi. Effective and efficient dropout for deep convolutional neural networks. *ArXiv*, abs/1904.03392, 2019.

[5] R. DeVore and R. Sharpley. Besov spaces on domains in $\mathbb{R}^d$. *Transactions of the American Mathematical Society*, 335, 06 1995.

[6] W. E and Q. Wang. Exponential convergence of the deep neural network approximation for analytic functions. *Science China Mathematics*, 61, 09 2018.

[7] D. L. Elliott. A better activation function for artificial neural networks, 1993.

[8] I. Fazekas. *Neurális hálózatok*. Debreceni Egyetem, Informatikai Kar, 2013. egyetemi jegyzet.

[9] I. Guyon. A scaling law for the validation-set training-set size ratio. In *AT and T Bell Laboratories*, 1997.

[10] I. Guyon, J. Makhoul, R. Schwartz, and V. Vapnik. What size test set gives good error rate estimates?. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20:52–64, 01 1998.

[11] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[12] T. Jayalakshmi and S. A. Statistical normalization and back propagation for classification. *International Journal Computer Theory Engineering (IJCTE)*, 3:89–93, 01 2011.

[13] J. Kruschke. Improving generalization in backpropagation networks with distributed bottlenecks. *Proceedings International Joint Conference Neural Networks (Baltimore*, vol IV:443 – 447 vol.1, 02 1989.

[14] A. Nicolae. Plu: The piecewise linear unit activation function, 2018.

[15] J. Opschoor, P. Petersen, and C. Schwab. Deep relu networks and high-order finite element methods. *Analysis and Applications*, 12 2019.

[16] P. S. P. Sibi, S. Allwin Jones. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology*, 47(3):1264–1268, 2013.

[17] P. Petersen and F. Voigtländer. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 09 2017.

[18] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[19] I. Safran and O. Shamir. Depth-width tradeoffs in approximating natural functions with neural networks, 2016.

[20] R. Setiono. A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[22] M. Telgarsky. Representation benefits of deep feedforward networks. *ArXiv*, abs/1509.08101, 2015.

[23] J. Wang and S. Li. Comparing the influence of depth and width of deep neural network based on fixed number of parameters for audio event detection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2681–2685, 04 2018.

[24] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103 – 114, 2017.