Eötvös Loránd Tudományegyetem

Természettudományi Kar

Temmel Bálint

Neurális közönséges differenciálegyenletek

Szakdolgozat

Matematika BSc, Elemző Matematikus Szakirány

Témavezető

Fekete Imre

Egyetemi adjunktus

Alkalmazott Analízis és Számításmatematikai Tanszék

Budapest

2019

Köszönetnyilvánítás

Ahhoz, hogy az ebben a dolgozatban a szavakat egyáltalán leírhattam, természetesen sok ember támogatására volt szükség.

Szeretném közülük név szerint kiemelni elsősorban Fekete Imrét, aki nem csak gyakorlatvezetőként, de témavezetőként is kiválónak bizonyult, így mind az órái, mind a konzultációi inspirálóan hatottak és sok szakmai tapasztalatot nyújtottak. Külön köszönöm neki, hogy elapadhatatlan segítőkészségével még az ünnepek alatt is kész volt az idejéből áldozni a dolgozatomra.

Továbbá köszönöm szüleimnek ,családtagjaimnak a mérhetetlenül sok támogatást, szaktársaimnak a lelki és szakmai iránymutatásokat, illetve hálás köszönet mindenkori tanáraimnak akik fáradságos munkával egyengették útjaim.

Tartalomjegyzék

1. Bevezetés **4**

2. A neurális hálókról **5**

2.1. A perceptron 7

2.2. A többrétegű perceptron 12

2.2.1. A többrétegű perceptron tanítása 14

2.2.2. Hibavisszaterjesztéses módszerrel kapcsolatos megjegyzések 17

2.3. A reziduális hálók 18

3. A kezdetiérték-feladatokról **21**

3.1. Folytonos elméleti összefoglaló 21

3.2. Numerikus elméleti összefoglaló 23

3.2.1. Runge–Kutta módszerek 24

3.2.2. Többlépéses módszerek 26

4. Neurális közönséges differenciálegyenletek **30**

4.1. Felügyelt tanítás és az adjungált egyenlet 31

4.2. Implementáció és példák 34

5. Összefoglalás **36**

1. fejezet

Bevezetés

Manapság igen gyakran találkozhatunk az adattudomány (data science) vagy például a mesterséges intelligencia (artificial intelligence röviden AI) kifejezésekkel. Utóbbi kapcsán a kevésbé szakavatottaknak olykor rémisztő gondolataik ébredhetnek. A már szakavatottabbak ugyanakkor a mai trendeknek megfelelően érdekes és izgalmas elgondolásokkal is megismerkedhetnek, mint például a gépi tanulás (machine learning) vagy a neurális hálózatok (neural networks) világa.

A dolgozat a neurális hálózat témaköréhez kapcsolódóan egy újfajta elgondolást fog bemutatni. Az Olvasó megismerheti, hogyan lehet ötvözni a több mint 100 éves múltra visszatekintő közönséges differenciálegyenletek numerikus módszereinek világát a manapság egyre népszerűbb neurális hálózatok (továbbiakban neurális hálók) témakörével.

Ennek alapját a „*Neural Ordinary Differential Equaitons*" c. cikk képezi [1]. A szakmai berkekben nagy visszhangot kiváltó publikáció *Neural Information Processing Systems* (NIPS) konferencián 4854 cikk közül *Best Papers Award* díjban részesült. A témával kapcsolatos kutatások még javában folynak. Jól jelzi ezt, hogy a cikk legutolsó verzióját arXivon 2019. december 14-én frissítették a Torontói Egyetem kutatói. Egyelőre sok megválaszolatlan kérdés van, de már sokan úgy vélik, hogy ez az irány rendkívül biztató.

A következő fejezetekben a cikkhez szükséges neurális hálók, majd a közönséges differenciálegyenletek témakörét tekintjük át olyan szellemben, hogy az utolsó fejezetben megérthessük, hogy ez a díjnyertes cikk miben tartalmaz új és korszerű gondolatot.

2. fejezet

A neurális hálókról

Számos alkalmazási területnél a természet ihlette neurális hálók alkalmazása a legcélravezetőbb. Ezek az ún. „mesterséges" neuron hálózatok mind felépítésükben (egymással kapcsolatban álló elemi műveletvégzésre alkalmas egységek), mind a tapasztalatokra épülő tanulási módszereikben nagyon sok hasonlóságot mutatnak a biológiai neuron hálózatokkal.

A szakdolgozat céljaiból eredendően nem fogunk sokat foglalkozni a neurális hálók alkalmazásaival (habár lesz róla szó), inkább a felépítésükre és a működési mechanizmusukra fektetjük a hangsúlyt. A fejezet főként az [5] jegyzetet követi, mely Haykin klasszikus neurális hálók könyvén is alapszik [10]. Célunk ebben a fejezetben az, hogy megfelelő alapot nyújtsunk az alapgondolatok megértéséhez, később pedig a közönséges differenciálegyenletek és a neurális hálók ötvözéséhez.

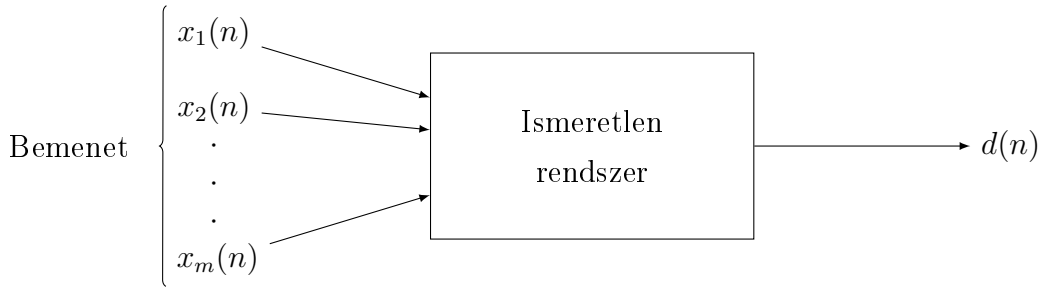
Amikor neurális hálókat akarunk alkalmazni egy természettudományos vagy műszaki probléma megoldására, akkor nem célunk a problémát a hagyományos értelemben modellezni, egyes törvényszerűségeket feltárni, hanem pusztán a bemeneti és a kimeneti adatokra támaszkodva, a közöttük lezajló eseményekre egyfajta fekete dobozként tekintve kívánjuk a problémát kezelni.

Jelölje a bemeneti adatokat az

$$x(n) = (x_1(n), x_2(n), \ldots, x_m(n))^T$$

m-dimenziós vektorok, ahol $n = 1, 2, \ldots$ az aktuális időpillanatok. A rendszer $x(n)$ be-

meneti értékeihez tartozó kimeneti értékeket $d(n)$-nel jelöljük. A folyamatot a 2.1. ábra szemlélteti.



2.1. ábra. A modellező rendszer.

A neurális hálók alkalmazását három fő lépésben írhatjuk le.

- ◇ A modell megalkotása.
- ◇ A modell (be)tanítása.
- ◇ A modell használata.

Alapvető fontosságú a második lépés, vagyis a neurális hálók tanítása. A hálók tanításának három nagy módja létezik.

- ◇ Tanítóval történő vagy felügyelt tanítás (supervised learning)

  Az ellenőrzött tanítás esetében rendelkezésre áll egy ún. tanító adathalmaz, amely tartalmazza a bemeneteket és az ezekhez elvárt válaszokat. Maga a tanítás pedig úgy történik, hogy a neurális háló bemenetekre adott kimeneteleit összehasonlítjuk az elvárt válaszokkal. Ez alapján próbáljuk a hálózat paramétereit módosítani úgy, hogy a kimenetek és az elvárt válaszok között a lehető legkisebb legyen az eltérés.

- ◇ Tanító nélküli tanítás (unsupervised learning)

  Ha nem ellenőrzött tanulásról beszélünk, akkor nem állnak rendelkezésre a bemenetekhez elvárt válaszok. A hálónak magának kell kialakítania valamilyen viselkedést, amivel a bemeneteli adatokban hasonlóságokat, korrelációkat, illetve csoportokat és kategóriákat fedhet fel.

- ◇ Analitikus tanítás

Analitikus esetben a háló tanítása szintén tanító adathalmazon történik, viszont itt a hálót nem lépésről lépésre, hanem közvetlen számításokkal és matematikai összefüggések kiértékelésével, analitikus módszerekkel alakítjuk ki.

Mivel a dolgozat szempontjából csak az ellenőrzött a tanítás lényeges, ezért majd ezt használjuk a későbbiekben.

A neurális hálók fejlődési történetére és a legkülönbözőbb típusaikra nem térünk ki. Az érdeklődő Olvasó ezeket Haykin könyvében [10] részletekbe menően megtalálja. Ugyanakkor a fogalmak és a folyamatok jobb megértése miatt tárgyaljuk a neurális hálók első típusaként ismert perceptront (egy neuronból), illetve az ezek segítségével alkotott többrétegű perceptront (multi layer perceptron röviden MLP).

2.1. A perceptron

Általánosságban elmondhatjuk, hogy a neurális hálók olyan rétegekbe szervezett egymással kapcsolatban lévő neuronok összessége, melyek a bemenet és a kimenet között valamilyen nemlineáris leképezést valósítanak meg. A későbbi vizsgálódás megalapozásához elsőként a neuron felépítését mutatjuk meg.

Maga a neuron (idegsejt) egy információ feldolgozó egység, amely a neurális háló alapegysége. Az egy neuronból álló perceptron pedig a legegyszerűbb neurális háló. A perceptron részei:

Bemenet (input)
Az $x(n) = (x_1(n), x_2(n), \ldots, x_m(n))^T$ ismert m-dimenziós vektor a bemenet. Minden $n = 1, 2, \ldots, N$ időpontban érkezik egy m számból álló bemenő jel. Ezek az adatok ismertek.

Súlyok (synaptic weight)
A cél a kezdetben ismeretlen $w_1, w_2, \ldots, w_m$ súlyok meghatározása. A valódi súlyok n-edik időpillanatbeli súlyai a $w_1(n), w_2(n), \ldots, w_m(n)$ súlyok. Az egyes lépések során ezeket finomítjuk. Az $x = (x_1, x_2, \ldots, x_m)^T$ vektor a valódi, míg az $x(n) = (x_1(n), x_2(n), \ldots, x_m(n))^T$ pedig a közelítő súlyvektor.

Torzítás (bias)
A b torzítás valódi értéke sem ismert, így ehhez is lépésenkénti közelítéssel juthatunk el. A torzítás n-edik közelítését $b(n)$ jelöli.

Összegző csomópont (summing junction)
Ez a bemeneti adatok

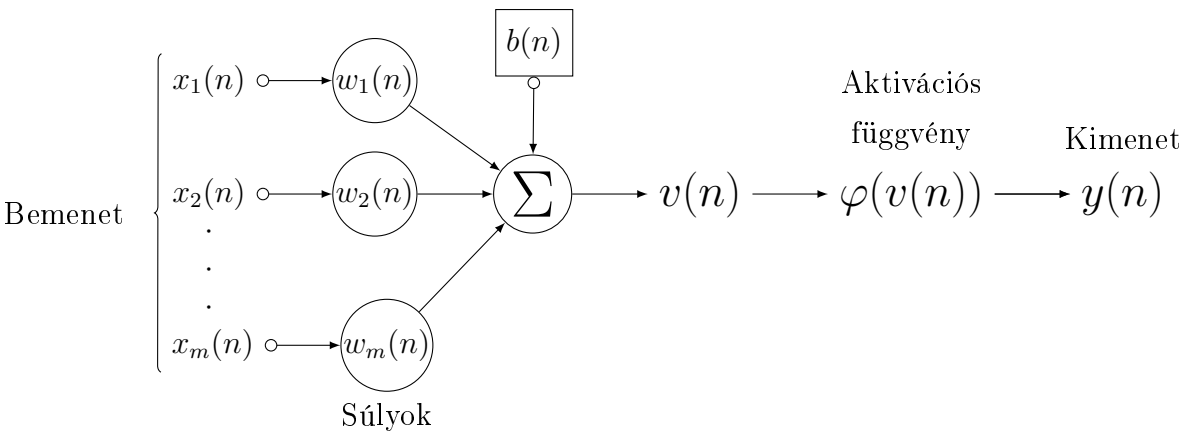
$$v(n) = b(n) + \sum_{i=1}^{m} w_i(n)x_i(n)$$

súlyozott összegét képezi.

Aktivációs függvény (activation function vagy transfer function)
Egy olyan φ függvény, ami a fenti $v(n)$ értéket transzformálja úgy, hogy a kapott érték a feladat szempontjából alkalmas intervallumba essen. A függvényt nekünk kell megadnunk.

Kimenet (output)
A neuron által az $x(n)$ bemeneti értékhez rendelt $y(n) = \varphi(v(n))$ érték.

A 2.2. ábra szemlélteti a perceptron sematikus felépítését.



2.2. ábra. A perceptron sematikus felépítése.

Tegyük fel, hogy a választott neurális háló sémánk egy perceptron és rendelkezésre állnak olyan adatok, ahol ismerjük az $x(n)$ bemeneti vektorokat, illetve a hozzájuk tartozó $d(n)$ kimeneti értékeket. Ezeket **tanító pontoknak** nevezzük.

Most tekintsük a perceptron által az $x(n)$ bemenethez szolgáltatott $y(n)$ kimenet és a tényleges $d(n)$ kimenet közötti

$$\varepsilon(n) = (d(n) - y(n))^2$$

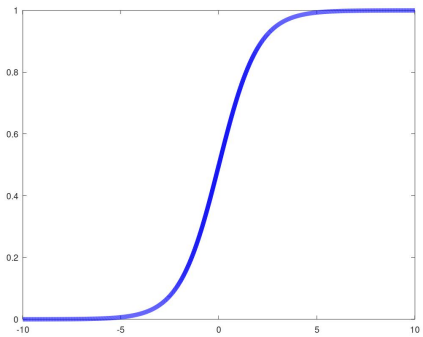
négyzetes különbséget. A fenti érték a hálózat négyzetes hibája. Ennek a súlyok módosításával történő minimalizálását mondjuk a **háló tanításának vagy tanulásának** (training). Ezt általában a tesztelés folyamata követi, ahol a már betanított háló általánosítóképességét vizsgálják.

Aktivációs függvények

Ahogy láthattuk az aktivációs függvények feladatra jellemző választása fontos a kimenetek tekintetében. Ebben az alfejezetben felsoroljuk a leggyakrabban használt aktivációs függvényeket, melyek a 2.3.–2.8. ábrákon láthatók. Ezek tipikusan S-szerű függvények és jellemző tulajdonságuk, hogy monoton növekvőek és jobbról folytonosak. További tulajdonság lehet, hogy a határértékük mínusz végtelenben 0, míg plusz végtelenben 1.

i, Logisztikus függvény

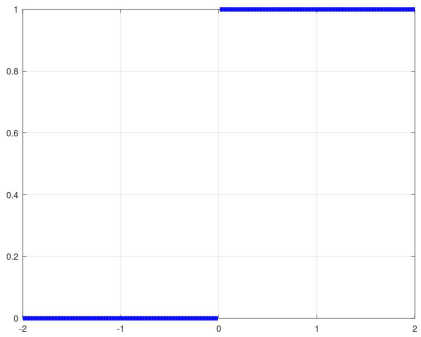
$$\varphi(x) = \frac{1}{1 + exp(-ax)}, x \in \mathbb{R}$$



2.3. ábra. A logisztikus aktivációs függvény $a = 1$ választással.

ii, Küszöb függvény (threshold function, Heaviside function, hard limit)

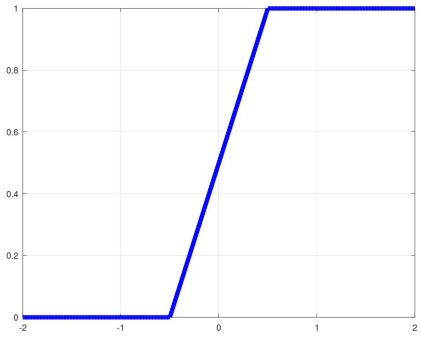
$$\varphi(x) = \begin{cases} 1 & \text{ha } x \geq 0 \\ 0 & \text{ha } x < 0 \end{cases}$$



2.4. ábra. A küszöb aktivációs függvény.

iii, Szakaszonként lineáris függvény (telítéses lineáris)

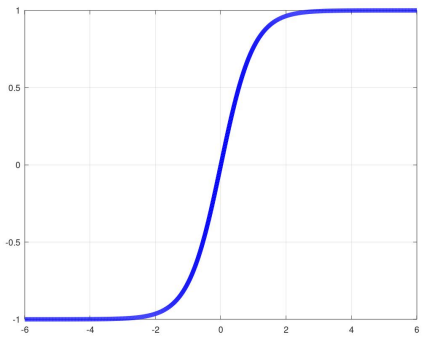
$$\varphi(x) = \begin{cases} 0 & \text{ha } x \leq -0.5 \\ x + 0.5 & \text{ha } -0.5 \leq x < 0.5 \\ 1 & \text{ha } x \geq 0.5 \end{cases}$$



2.5. ábra. A szakaszonként lineáris aktivációs függvény.

iv, Tangens hiperbolikus függvény

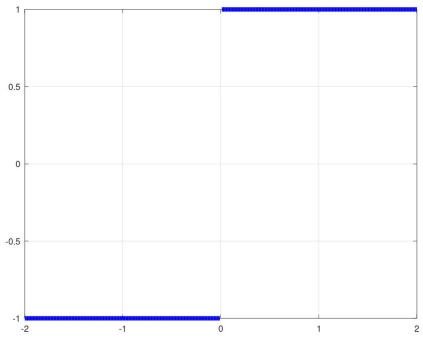
$$\varphi(x) = \tanh(x)$$



2.6. ábra. A tangens hiperbolikus aktivációs függvény.

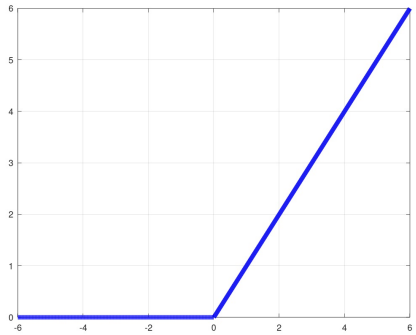
v, Szignum függvény (speciális küszöb függvény)

$$\varphi(x) = \begin{cases} -1 & \text{ha } x < 0 \\ 0 & \text{ha } x = 0 \\ 1 & \text{ha } x > 0 \end{cases}$$



2.7. ábra. A szignum aktivációs függvény.

Érdemes megjegyezni, hogy a fentiektől eltérő struktúrájú függvényeket is alkalmaznak, például gyakori a $\varphi(x) = x$ identitás vagy a lentebb is látható $\varphi(x) = \max(0, x)$ ReLu választás.



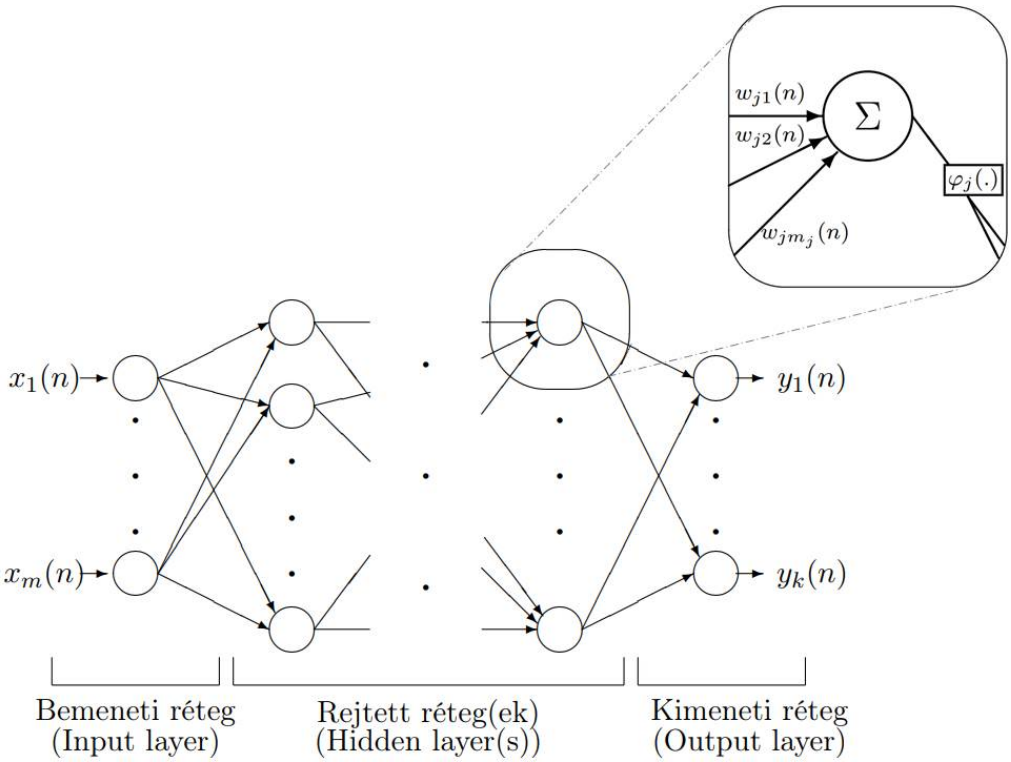
2.8. ábra. A ReLu aktivációs függvény.

2.2. A többrétegű perceptron

Ha valaki neurális hálókra gondol, akkor nagy valószínűséggel egy többrétegű perceptront képzel el, ugyanis ez talán a legismertebb hálózati séma. Ennél a felépítésnél nemcsak egyszerűen összekapcsoljuk a neuronokat, hanem rétegekbe is szervezzük őket. Ezeket a rétegeket három csoportba oszthatjuk.

- Bemeneti réteg (input layer)
- Rejtett réteg (hidden layer)
- Kimeneti réteg (output layer)

Rejtett rétegekből tetszőleges számú, bemeneti és kimeneti rétegekből viszont csak egy-egy réteg lehet a hálózatban. Ezen felül a bemeneti rétegről érdemes tudni, hogy számítást, illetve jelfeldolgozást nem végez, feladata csupán a bemeneti adatok továbbítása. A többrétegű perceptron sematikus felépítése a 2.9. ábrán látható.

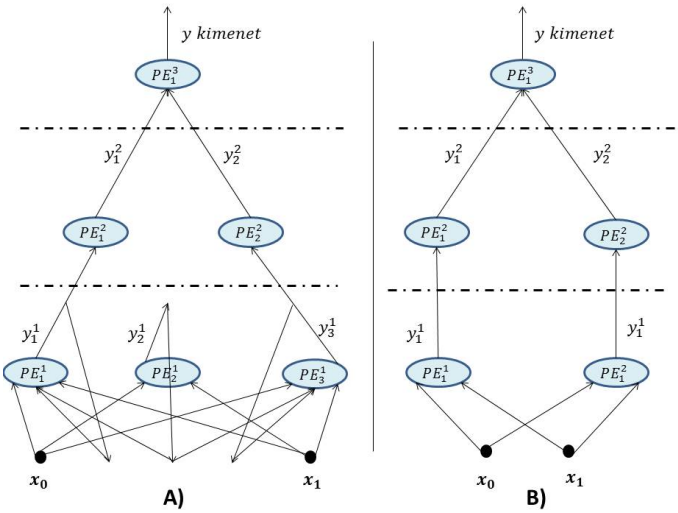


2.9. ábra. A többrétegű perceptron sematikus felépítése [5].

A fenti modell esetében nincs rétegen belüli és nincs a távolabbi rétegek között sem kapcsolat, viszont minden neuron kapcsolatban áll a közvetlen szomszédos réteg összes neuronjával. További fontos tulajdonsága a többrétegű perceptronoknak, hogy minden neuronjának saját aktivációs függvénye és saját súlyai vannak.

A neurális hálókat topológiailag leginkább irányított gráfként tudjuk reprezentálni, melyben a csomópontoknak a neuronok, a kapcsolatoknak pedig a gráf élei felelnek meg. Ebből kiindulva két főcsoportot különböztetünk meg aszerint, hogy az általuk reprezentált irányított gráf egyirányú, vagy esetleg tartalmaz hurkot vagy visszacsatolást. Előbbi esetben **előrecsatolt**, utóbbi esetben **visszacsatolt** hálókról beszélhetünk. A két esetet a 2.10. ábra mutatja.

A többrétegű perceptron egy előrecsatolt háló, azaz a bemeneti jel a rétegről rétegre halad előre, mígnem a kimeneti réteg megadja a kimenő jelet.



2.10. ábra. A visszacsatolt és előrecsatolt hálók.

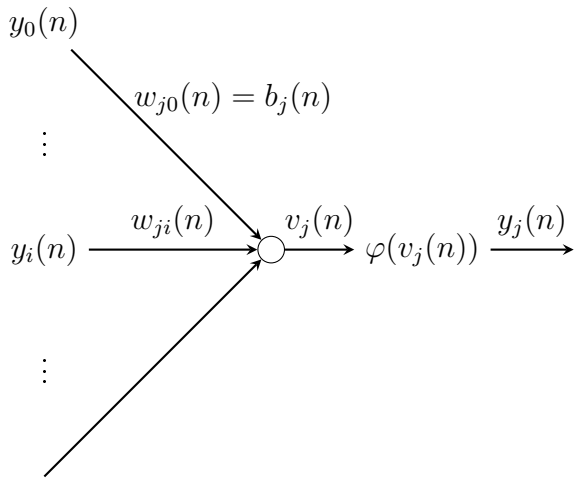
2.2.1. A többrétegű perceptron tanítása

A többrétegű perceptron tanítása négy főbb lépésből áll.

- ◇ Megadjuk a kezdeti súlyokat.
- ◇ A bemeneti jelet végigáramoltatjuk a hálón a súlyok megváltoztatása nélkül.
- ◇ A kapott kimeneti jelet összevetjük az elvárt (tényleges) kimeneti jellel.
- ◇ A hibát visszaáramoltatjuk. A súlyokat a hiba csökkentése érdekében módosítjuk.

A tanítás során tipikusan a tanító pontokat többször átáramoltatjuk. Az összes tanító pont egyszeri átáramoltatását egy **epochnak** nevezzük.

A tanítás kritikus lépése a súlyok módosítása annak érdekében, hogy a hiba minimális legyen. Ez az ún. hibavisszaterjesztéses (error back-propagation) módszerrel történik. A felhasznált jelölések értelmezéséhez tekintsük a 2.11. ábrán egy neuront a többrétegű perceptronban.



2.11. ábra. Egy neuron a többrétegű perceptronban.

Vezessük be az alábbi jelöléseket és formális definíciók írása helyett ezeket listaszerűen felsoroljuk.

- Az i, j, k hármas az i-edik, j-edik és k-adik neuront jelöli balról jobbra haladással.
- Az n jelöli az n-edik időpillanatot, azaz ez a tanítás n-edik lépése.
- Az i-edik neuron kimenete $y_i(n)$ és egyben a j-edik neuron bemenete, feltéve ha j rétege közvetlenül i rétege után áll.
- Legyen $y_0(n) = 1$.
- Az i-edik neuronból a j-edik neuronba mutató élen lévő súly $w_{ji}(n)$.
- A j-edik neuron torzítása $b_j(n) = w_{j0}(n)$.
- A j-edik neuron által produkált összegzett értékét

$$v_j(n) = \sum_{i \in \{j \text{ előtti réteg}\}} w_{ji}(n) y_i(n)$$

  jelöli.
- A j-edik neuronhoz tartozó aktivációs függvény φ_j.
- A j-edik neuron kimenete $y_j(n) = \varphi_j(v_j(n))$.
- Az $y_j(n)$-hez viszonyított valódi kimeneti érték d_j.

Jelölje továbbá C a kimeneti réteg neuronjainak halmazát. Ekkor az

$$\bar{\varepsilon} = \frac{1}{N}\sum_{n=1}^{N}\varepsilon(n) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\sum_{j\in C}e_j^2(n)\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}\sum_{j\in C}(d_j(n)-y_j(n))^2\right)$$

az **átlagos hiba**, ahol N a tanító pontok számát jelöli.

A súlyok korrekcióját a

$$w_{ji}(n+1) - w_{ji}(n) = \Delta w_{ji}(n) = -\eta\frac{\partial\varepsilon(n)}{\partial w_{ji}(n)} \tag{2.1}$$

gradiens módszer segítségével kapjuk, ahol $\eta > 0$ a tanulási paraméter. Érdemes megjegyezni, ha η kicsi, akkor a súlyok is kis mértékben változnak, viszont lassú lesz a konvergencia. Nagy η esetében a konvergencia gyorsabb, viszont az algoritmus instabillá válhat.

Az ε függvényértékeinek meghatározása után, a háló paramétereit (súlyok és torzítás) a végső kimenettől kezdve visszafelé haladva módosítjuk. Általában különböző súlyok és nemlineáris függvények határozzák meg a háló felépítését, így egy adott pontban a (2.1) jobboldali tag (ún. **veszteségfüggvény**) értékét valamely bemeneteinek módosítása jobban befolyásolja mint a többi bemenetelének a módosítása. Ezeket az arányokat a veszteségfüggvény iránymenti deriváltjai határozzák meg. A hosszas levezetést elhagyjuk. Az érdeklődő Olvasó a fejezet alapjául szolgáló [5] jegyzetben részletesen megtalálhatja. A súlyok aktualizálási mechanizmusa

$$\Delta w_{ji}(n) = \eta\delta_j(n)y_i(n), \tag{2.2}$$

ahol $y_i(n)$ a már korábban említett j-edik rétegét közvetlenül megelőző rétegbeli i-edik neuron kimenete, míg a lokális gradiens

$$\delta_j(n) = \begin{cases} e_j(n)\varphi_j'(v_j(n)) & \text{rejtett rétegek esetén,} \\ \varphi_j'(v_j(n)) \displaystyle\sum_{k\in\{j \text{ utáni réteg}\}} \delta_k(n)w_{kj}(n) & \text{kimeneti rétegek esetén.} \end{cases}$$

A (2.2) formula lényege abban áll, hogy a j-edik neuron lokális gradiense megkapható a j-edik utáni réteg lokális gradiensei segítségével. A kimeneti réteg lokális gradiensei közvetlenül számolhatóak, így rétegről-rétegre visszafelé haladva az összes lokális gradiens meghatározható.

2.2.2. Hibavisszaterjesztéses módszerrel kapcsolatos megjegyzések

Ahogy az előző alfejezetben láthattuk az algoritmus lényegében a következő lépéseket hajtja végre:

1) Lefuttatja a bemeneti adatokat a hálózaton.

2) Kiszámítja a veszteségfüggvényt.

3) Meghatározza a lokális gradienseket.

4) Végrehajtja a gradiens lépést.

Természetesen felmerülnek gyakorlati kérdések is. Többek között ilyen, hogy milyen leállási feltételeket határozzunk meg. Négy tipikusnak mondható leállási feltételt említünk meg.

- ◇ Ha a gradiens vektor kellően kicsi, akkor megállunk, ugyanis ekkor már a súlyok nem változnak jelentős mértékben.
- ◇ Ha egy súlymódosítás előtt és után kapott átlagos négyzetes hiba az előírt tolerancia értékénél kisebb.
- ◇ Ha az epochok száma túl nagy.
- ◇ Ha a futási idő túl sok.

Ahogy láthattuk a hibavisszaterjesztéses módszer gradiens alapú, így érzékeny a kezdeti értékekre, azaz a kezdeti súlyok megválasztása fontos feladat. Gyakori jelenség lehet, hogy az algoritmus csupán egy lokális minimumot talál meg.

Fontos leszögezni, hogy a numerikus matematikában számos eljárás létezik az ε függvény w szerinti minimumhelyének meghatározására. A többrétegű perceptron esetében a gradienseket a hibavisszaterjesztéses algoritmussal számítják ki, majd pedig ezeket a gradienseket a választott eljárásban felhasználják. Így a többrétegű perceptron tanítására több módszer is rendelkezésre áll, melyek közül ki lehet választani azt, amelyik a feladathoz a leginkább megfelelő. A teljesség igénye nélkül megemlítünk néhány gyakorlatban leginkább használt numerikus módszert.

A konjugált gradiens módszer. Ennél a módszernél a hibavisszaterjesztéses algoritmus által kiszámítjuk a gradienst, viszont a lépés nem a negatív gradiens irányába történik, hanem egy bizonyos ortogonalizációs eljárással meghatározott módosított irányba. A módszer főbb variánsai: Fletcher–Reeves-formula, Polak–Ribiére-formula és a Powel–Beale-formula ([10] 188–194. oldal).

Kvázi Newton-módszerek. Ebben az esetben a gradienst szintén a hibavisszaterjesztéses algoritmussal határozzák meg, majd ezeket a gradienseket felhasználva keresik meg az ε függvény minimumát egy Newton-módszerhez hasonló iterációs eljárással. Leggyakrabban a Broyden–Fletcher–Goldfarb–Shanno-formulát alkalmazzák ([10] 194–197. oldal).

2.3. A reziduális hálók

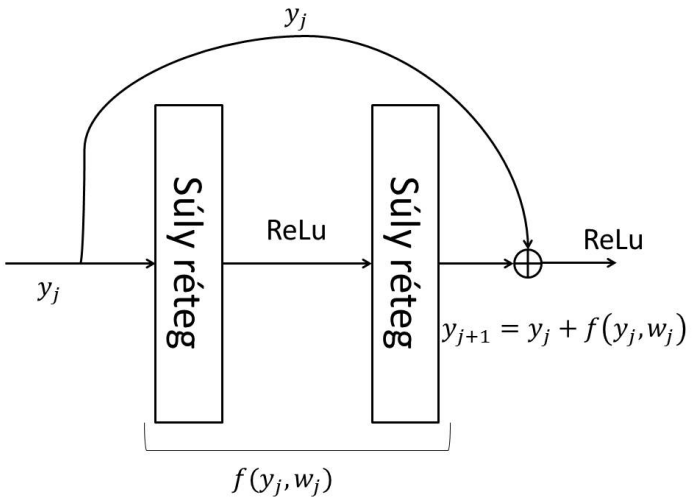
A háló felépítésével kapcsolatban tudjuk, hogy a használhatóságot és az általánosító képességet is nagyban befolyásolják az olyan paraméterek, mint például a rétegek száma és ezek mérete. Ennek alapján két út adódik egy háló megalkotására.

- A háló növelése: egy kicsi hálóból kiindulva addig növeljük azt, amíg a háló már jól betaníthatóvá válik.
- A háló metszése: az eőző eljárás fordítottját végezzük, vagyis egy nagy és jól betanított hálót addig redukálunk (csökkentjük a neuronok és a súlyok számát), amíg még jó eredményt ad.

Miután a konvolúciós hálók áttörést értek el a képfeldolgozás terén [12] több kérdés is felmerült a hálók felépítésével kapcsolatban. Amikor a hálók mélyítésével, vagyis a rétegek számának növelésével próbáltak javítani az elért pontosságon, akkor szembesültek azzal a problémával, hogy egy bizonyos pont után a háló rosszabbul kezd el teljesíteni [6]. Túltanulásról nem lehetett szó, mivel ez a jelenség akkor is fennállt amikor a hozzáadott rétegek csupán egy identitás leképezést valósítottak meg. Ebből kifolyólag a háló felépítésének módosítására volt szükség, hogy a problémát orvosolni tudják.

Egészen 2015 decemberéig kellett várni, amikor is egy újfajta hálózati felépítéssel rukkoltak elő, mely **reziduális háló** néven vált ismertté [11]. Szokásos esetben a bemeneti

jel rétegről rétegre halad előre a hálóban úgy, hogy egy adott réteg a kapott bemeneti jelen végrehajt egy leképezést, majd az így kapott jel lesz a következő réteg bemeneti jele. Ezzel szemben a reziduális hálókban egy adott réteg bemeneti jeléhez hozzávesszük egy őt megelőző réteg bemeneti jelét is, vagyis úgymond egy réteg(ek)en átívelő **rövidre zárást** (shortcut) hozunk létre. Egy ilyen rövidre zárás által közrefogott részre **reziduális blokként** fogunk hivatkozni. Ebben az egy blokkban legyen y_j a bemenet, y_{j+1} a kimenet, míg f a közrefogott rétegek között megvalósuló leképezés. Egy reziduális blokk sémája a 2.12. ábrán látható.



2.12. ábra. Egy reziduális blokk ReLu aktivációs függvénnyel.

Tegyünk meg itt egy érdekes észrevételt. Vegyünk ugyanis egy reziduális hálót, amely a fenti reziduális blokkokból épül fel. Adott y_0 bemenettel, N réteg mellet fennáll az

$$y_{j+1} = y_j + f(y_j, w_j), \qquad j = 0, \ldots, N-1$$

összefüggés, w_j a j-edik réteghez tartozó súlyvektor. Szorozzuk meg a fenti kifejezésben f-et mesterségesen $h = 1$-gyel. Ekkor az

$$y_{j+1} = y_j + hf(y_j, w_j), \qquad j = 0, \ldots, N-1 \tag{2.3}$$

összefüggés adódik.

A fenti (2.3) egyenlet egy numerikus módszerek világában jártas Olvasó számára könnyen szemet szúr. Tudniillik ez lesz a következő fejezetben részletesen is tárgyalt ún.

kezdetiértek-feladatok megoldására szolgáló explicit Euler-módszerre hasonlít. Ez az észrevétel volt, ami kitárja a közönséges differenciálegyenletek numerikus módszerei és a neurális hálók közötti kaput.

2.3.1. Megjegyzés. A reziduális hálóknak is megannyi fajtája létezik a rétegek számától, méretétől és a rövidre zárások által kihagyott lépések számától függően. Fontos tulajdonságuk, hogy nagyon egyszerűen képesek az identitás függvényt megtanulni, ezáltal ha a háló akár több száz réteg mély is mégsem fogja negatívan befolyásolni a teljesítményét. A rövidre zárások lehetnek alapvetően akár paraméter nélküliek és paraméterezettek is. Az ezzel kapcsolatos részletes konklúziókért lásd [11].

Mielőtt belemerülünk a neurális közönséges differenciálegyenletek világába a következő fejezetben tömören tárgyaljuk a kezdetiérték-feladatok elméleti és numerikus aspektusait is.

3. fejezet

A kezdetiérték-feladatokról

Ahogy az előző fejezet legvégén láthattuk, hogy egy a numerikus analízisben jártas Olvasó a reziduális hálók egy blokkjára tekinthet úgy, mint egy közönséges differenciálegyenlet kezdetirték-feladatát numerikusan megoldó explicit Euler módszer egy lépésére.

Ugyanakkor ebben a területben kevésbé jártasak számára ebben a fejezetben röviden összefoglaljuk a differenciálegyenletekkel kapcsolatos és számunkra szükséges elméletet. Emellett vázoljuk a legnépszerűbb numerikus eljárások megértéséhez szükséges alapokat, illetve tételesen felsoroljuk a később használandó módszereket is.

3.1. Folytonos elméleti összefoglaló

Az alfejezetben a [3] és [16] könyvek megfelelő részei alapján közöljük a legfontosabb definíciókat, tételeket.

Differenciálegyenletek segítségével az egyenletben szereplő ismeretlen függvény és annak deriváltja(i) közötti összefüggést írhatjuk le. A közönséges szó arra arra utal, hogy az ismeretlen függvény egyváltozós. Formálisan a közönséges differenciálegyenletekre vonatkozó definíció az alábbi.

3.1.1. Definíció. *Legyen $G \subset \mathbb{R} \times \mathbb{R}^d$ egy összefüggő nyílt halmaz (tartomány) és $f : G \to \mathbb{R}^d$ egy folytonos leképezés. Az*

$$u'(t) = f(t, u(t))$$

kifejezést ***közönséges differenciálegyenletnek*** *mondjuk.*

3.1.2. Definíció. *Legyen $G \subset \mathbb{R}\times\mathbb{R}^d$ egy tartomány, $(t_0, u_0) \in G$ egy adott pont $\big(t_0 \in \mathbb{R}, u_0 \in \mathbb{R}^d\big)$, valamint $f : G \to \mathbb{R}^d$ egy folytonos leképezés. A*

$$u'(t) = f(t, u(t)) \tag{3.1}$$

$$u(t_0) = u_0 \tag{3.2}$$

feladatot ***kezdetiérték-feladatnak****, vagy más szóval* **Cauchy-feladatnak** *nevezzük.*

3.1.3. Definíció. *Az olyan $u : I \to \mathbb{R}^d$ (I nyílt intervallum) folytonosan differenciálható függvényt, melyre*

- $\{(t, u(t)) : t \in I\} \subset G$,
- $u'(t) = f(t, u(t))$ *minden* $t \in I$ *esetén,*
- $t_0 \in I$ *és* $u(t_0) = u_0$

a ***kezdetiérték-feladat megoldásának*** *nevezzük.*

Az alkalmazások szempontjából is fontos a kezdetiérték-feladat megoldásának egyértelműsége.

3.1.4. Definíció. *Legyen $G \subset \mathbb{R} \times \mathbb{R}^d$ egy tartomány. Az $f : G \to \mathbb{R}^d$ leképezést második változójában Lipschitz-tulajdonságúnak nevezünk, ha létezik $L > 0$ konstans, hogy minden $(t, p_1), (t, p_2) \in G$ esetén*

$$|f(t, p_1) - f(t, p_2)| \leq L|p_1 - p_2|.$$

Az egyértelműségre vonatkozó tétel az alábbi.

3.1.5. Tétel. *Legyen $f : G \to \mathbb{R}^d$ folytonos függvény, ahol*

$$G = \{(t, u) \in \mathbb{R} \times \mathbb{R}^d : |t - t_0| \leq a \text{ és } |u_0| \leq b\}$$

henger, $(t_0, u_0) \in \mathbb{R} \times \mathbb{R}^d$ és $0 < a < \infty$, $0 < b < \infty$. Legyen $M = \max\limits_{(t,u)\in G} |f(t, u)|$ és tegyük fel, hogy az f függvény második változójában Lipschitz-tulajdonságú. Ekkor a (3.1)-(3.2) kezdetiérték-feladatnak egyértelműen létezik megoldása a $[t_0 - \delta, t_0 + \delta]$ intervallumon, ahol $\delta = \min\{a, \frac{b}{M}\}$.

A differenciálegyenleteknek többnyire csak speciális esetekben létezik analitikus megoldása. Így a következő alfejezetben a (3.1)-(3.2) kezdetiérték-feladat megoldására szolgáló numerikus módszerek elméletének alapjait, illetve a később szükséges numerikus eljárások egyfajta összegzéséről lesz szó a konkrét módszerek kiemelésével.

3.2. Numerikus elméleti összefoglaló

Az egyszerűség kedvéért –de az általános elvesztése nélkül– a továbbiakban a (3.1)-(3.2) kezdetiérték-feladat skaláris $d = 1$ esetét tekintjük. Az alapfogalmak definiálásakor és a szerkezet felépítésében a [4] jegyzetre támaszkodunk.

Definiáljuk az

$$\omega_h = \{t_n = nh; n = 0, 1, \ldots, N; h = T/N\}$$

ún. **ekvidisztáns rácsháló**t. Célunk az, hogy az ezen értelmezett $y(t_n),\ t_n \in \omega_h$ rácsfüggvény jól közelítse a folytonos feladat $u(t)$ megoldását ezekben a rácspontokban, azaz

$$y(t_n) \approx u(t_n)\ t_n \in \omega_h.$$

A kevesebb írásért bevezetjük az $y_n := y(t_n)$ jelölést. Ekkor a (3.1) feladatban szereplő $u'(t)$ függvényt a tanult elsőrendű jobboldali véges differenciahányados alapján közelítjük a $t = t_n$ pontban, azaz

$$u'(t_n) \approx \frac{y_{n+1} - y_n}{h}. \tag{3.3}$$

Ekkor közvetlenül a feladat jobb oldalára az

$$f(t_n, u(t_n)) \approx f(t_n, y_n) \tag{3.4}$$

összefüggés adódik. Így a fenti (3.3)-(3.4) alapján

$$\frac{y_{n+1} - y_n}{h} = f(t_n, y_n), \qquad \text{ahol } n = 0, 1, \ldots$$

Az egyenletet rendezve az

$$y_{n+1} = y_n + hf(t_n, y_n), \qquad \text{ahol } n = 0, 1, \ldots \tag{3.5}$$

ún. explicit Euler módszerhez jutunk, ahol $n = 0$ kezdetiérték esetén az y_0 értékének a kezdetiérték-feladat u_0 érték szokás választani. Mivel y_{n+1} közvetlen kiszámítható y_n-ból, ezért a fenti (3.5) módszer egylépéses és explicit. Ezt az egyenletet már korábban más alakban a (2.3) egyenletnél láthattuk.

A továbbiakban a származtatott módszer alapján értelmezzük a numerikus analízis alapfogalmait: a konzisztenciát, a stabilitást és a konvergenciát.

3.2.1. Definíció. *Ha a numerikus módszer*

$$\Psi_n^{(1)} = -\frac{u(t_{n+1}) - u(t_n)}{h} + f(t_n, u(t_n))$$

lokális approximációs hibájára teljesül, hogy

$$\lim_{h\to 0} \Psi_n^{(1)} = 0,$$

akkor a módszert ***konzisztensnek*** *nevezzük. A módszert p-edrendben konzisztensnek hívjuk, ha $p > 0$-ra igaz, hogy*

$$\Psi_n^{(1)} = \mathcal{O}(h^p).$$

3.2.2. Definíció. *Azt mondjuk, hogy a numerikus módszer* ***konvergens*** *a $t^* \in \omega_h$ pontban, ha*

$$\lim_{h\to 0} |y_n - u(t^*)| = 0. \tag{3.6}$$

Abban az esetben, ha (3.6) teljesül a $[0, T]$ intervallum minden t^* pontjában, akkor a numerikus módszert konvergensnek nevezzük az egész intervallumon. A numerikus módszer p-edrendben konvergens, ha

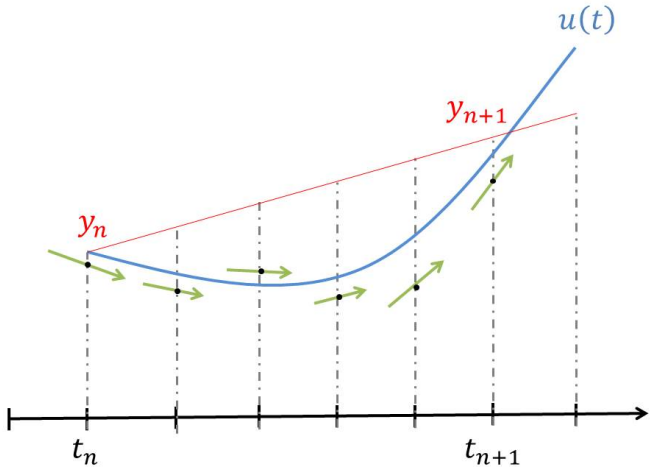
$$|y_n - u(t^*)| = \mathcal{O}(h^p) \tag{3.7}$$

teljesül, ahol $p \geq 0$.

A fenti definícióban szereplő kifejezésben szokásos az $e_n := y_n - u(t_n)$ **globális hiba** jelölés bevezetése. Ezen definíció mellett igaz, a numerikus analízis alaptétele, miszerint ha egy egylépéses módszer p-ed rendben konzisztens és stabil (azaz folytonosan függ a bemenő adatoktól), akkor a módszer p-edrendben konvergens. Közvetlen számolással adódik, hogy az explicit Euler módszer elsőrendben konzisztens és a numerikus analízis alaptételének értelmében elsőrendben konvergens módszer. A formális bizonyításért és a részletekért lásd [4].

3.2.1. Runge–Kutta módszerek

Az 19. században nagy erőfeszítéseket tettek annak érdekében, hogy magasabb rendben konzisztens módszereket találjanak. E folyamat során az derült ki, hogy konisztenciarendbeli javulást tudunk elérni, ha több közbülső (t_n, t_{n+1}) pontban, ún. lépcsőkben kiszámított értékkel lépünk az eredeti pontból. Ezt szemlélteti a lenti 3.1. ábra is. A 20. század elején alapokat megalkotó matematikusok után hagyománytiszteletből a róluk elnevezett



3.1. ábra. Runge-Kutta módszer alapgondolata.

ún. Runge–Kutta módszerek mind a mai napig az egyik legalapvetőbb numerikus eljárásul szolgálnak közönséges differenciálegyenletek megoldására.

Az általános s-lépcsős Runge–Kutta módszer alakja:

$$
\begin{aligned}
k_1 &= f(t_n + c_1 h, y_n + h(a_{11}k_1 + a_{12}k_2 + \ldots + a_{1s}k_s)) \\
k_2 &= f(t_n + c_2 h, y_n + h(a_{21}k_1 + a_{22}k_2 + \ldots + a_{2s}k_s)) \\
&\vdots \\
k_s &= f(t_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k_2 + \ldots + a_{ss}k_s)),
\end{aligned}
$$

ahol a_{ij}, c_i valós paraméterek. Az y_{n+1} értéket pedig ezen lépcsők lineáris kombinációjaként kaphatjuk meg. Mégpedig

$$y_{n+1} = y_n + h(b_1k_1 + b_2k_2 + \ldots b_sk_s), \qquad b_i \in \mathbb{R}.$$

A módszert adott s esetén az a_{ij}, b_i és c_i paraméterei határozzák meg. Bevezetve az $\mathbf{A} = (a_{ij})_{i,j=1}^s \in \mathbb{R}^{s\times s}$, $\mathbf{c} = (c_i)_{i=1}^s$ és $\mathbf{b}^T = (b_i)_{i=1}^s \in \mathbb{R}^s$ jelöléseket az s-lépcsős Runge–Kutte módszerek tömör formában megadhatóak az alábbi

$$
\begin{array}{c|c}
\mathbf{c} & \mathbf{A} \\
\hline
 & \mathbf{b}^T
\end{array}
$$

alakban. Az ilyen típusú táblázatot bevezetője nyomán **Butcher-táblónak** nevezzük.

Az erre a numerikus családra jellemző konzisztencia feltételeket, konvergencia tételeket és további elméleti eredményeket nem közöljük, mert a dolgozat szempontjából nem releváns. Annyit megjegyeznénk, hogy alkalmas feltételek mellett a numerikus analízis alaptétele fennáll. Bővebben az érdeklődő olvasó a [8] és [9] szakkönyvekben olvashat.

A neurális közönséges differenciálegyenletek megoldása során az alábbi egylépéses numerikus módszereket használjuk.

A korábban már tárgyalt **explicit Euler** módszer. A módszer Butcher-tablója az alábbi:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

- Egy középponti szabály típusú módszer természetes explicitté tétele során nyerjük a másodrendű ún. **explicit középponti módszert**, melynek Butcher-tablója

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array}$$

- A harmadik egy közkedvelt negyedrendű módszer, az ún. 3/8**-os RK4** (RK Runge–Kutta). Alkalmazások során azért preferálják, ugyanis ehhez a módszerhez a szakirodalomban klasszikusnak számító RK4 módszerhez kisebb hibakonstanssal rendelkezik. A módszer Butcher-tablója

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 \\ 2/3 & -1/3 & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & 1/8 & 3/8 & 3/8 & 1/8 \end{array}$$

3.2.2. Többlépéses módszerek

Az eddig leírt numerikus módszerek a megoldásfüggvény közelítését egy adott rácsháló pontban, az azt megelőző rácsháló-pontbeli közelítésből határoztuk meg. Igaz volt ez a

Runge–Kutta típusú módszerekre is, ahol annyi volt a különbség, hogy egy lépés között a (lépcsők számától függően) s számú pontban köztes értékeket is generálunk. Ezzel szemben a többlépéses módszerek az új közelítést egynél több megelőző rácspontbeli közelítésből határozzák meg.

3.2.3. Definíció. *Az adott $\alpha_0, \alpha_1, \ldots, \alpha_k$ és $\beta_0, \beta_1, \ldots, \beta_k$ együtthatók mellett az*

$$\alpha_0 y_0 + \alpha_1 y_{n-1} + \cdots + \alpha_k y_{n-k} = h[\beta_0 f_n + \beta_1 f_{n-1} + \cdots + \beta_k f_{n-k}], \tag{3.8}$$

iterációt, ahol $n = k, k+1, \ldots$ ***lineáris k-lépéses módszernek*** *nevezzük.*

A többlépéses módszert az α_j és $\beta_j (j = 0, 1, \ldots, k)$ $2k+2$ szabad paramétere definiálja. Ugyanakkor konvenció szerint (az egyértelműséghez) feltesszük, hogy $\alpha_0 = 1$. A módszer explicit, ha $\beta_0 = 0$. Egyébként a módszer implicit. Az $\frac{1}{h}\sum_{j=0}^{k} \alpha_j y_{n-j}$ az $u(t)$ megoldásfüggvény deriváltját, míg a $\sum_{j=0}^{k} \beta_j f_{n-j}$ az $f(t, u(t))$ függvényt approximálja a $t = t_n$ pontban.

A lineáris többlépéses módszerek családjában a Runge–Kutta módszer családdal szemben a konzisztencia feltételek elegánsan számolhatóak.

3.2.4. Tétel. *Az (3.8) lineáris többlépéses módszer p-edrendű, ha a módszert definiáló paraméterekre teljesülnek az*

$$\alpha_0 = 1, \qquad \sum_{j=0}^{k} \alpha_j = 0$$

$$\frac{1}{s}\sum_{j=0}^{k} j^s \alpha_j + \sum_{j=0}^{k} j^{s-1} \beta_j, \qquad s = 1, 2, \ldots, p$$

feltételek.

A lineáris többlépéses módszerek között fontos szerepet játszanak a bal oldali deriváltat a legegyszeűbb $\frac{y_n - y_{n-1}}{h}$ véges differenciás approximációval közelítő módszerek. Ekkor az α_j paramétereire az

$$\alpha_0 = 1, \alpha_1 = -1, \alpha_2 = \alpha_3 = \cdots = \alpha_k = 0$$

választás adódik.

3.2.5. Definíció. *Az*

$$y_n - y_{n-1} = h \sum_{j=0}^{k} \beta_j f_{n-j} \tag{3.9}$$

alakú módszereket k-lépéses ***Adams-módszereknek*** *nevezzük.*

Míg az explicit Adams-módszerek implementálása magától értetődő, addig implicit esetben több lehetőség is bevett. Amikor egy implicit többlépéses módszer realizálására kerül sor, akkor általában a létrejövő nemlineáris egyenletrendszert valamilyen iterációs módszer segítségével oldjuk meg. Ez tipikusan az egyszerű iteráció, vagy merev rendszerek esetében a Newton-módszer.

Létezik ugyanakkor egy út, amivel általában nem-merev feladatok esetén el lehet kerülni a p-edrendű lineáris többlépéses módszer nemlineáris feladatát. Az elgondolás szerint előre meghatározott számú iterációs lépést szándékozunk tenni a linearizálás céljából. Ezek a lépések a következők:

- Egy explicit, szintén p-edrendű módszerrel megjósoljuk (prediktálás – predict) a korábbi időpontbeli közelítésekből a t_n pontbeli közelítést. Ehhez tipikusan alkalmas explicit Adams-módszert szokásos alkalmazni. Legyen ez a módszer a következő:

$$(P) \qquad y_n^{(0)} + \hat{\alpha}_1 y_{n-1} + \cdots + \hat{\alpha}_k y_{n-k} = h(\hat{\beta}_1 f_{n-1} + \cdots + \hat{\beta}_k f_{n-k}).$$

- Ezután ezzel az értékkel kiértékeljük (evaluate) az $f(t, u(t))$ függvényt a $t = t_n$ pontban. Azaz

$$(E) \qquad f_n^{(0)} = f(t_n, y_n^{(0)}).$$

- Ezt az értéket behelyettesítve az implicit lineáris többlépéses módszerbe korrigáljuk (correct) a megoldást:

$$(C) \qquad y_n^{(1)} + \alpha_1 y_{n-1} + \cdots + \alpha_k y_{n-k} = h(\beta_0 f_n + \cdots + \beta_k f_{n-k}).$$

A második (E) és a harmadik (C) lépések ismételhetőek mindig az utolsó kiszámolt értékkel. A t_n pontbeli közelítés az $y_n = y_n^{(m)}$ érték lesz az m-edik végrehajtás után. Ezt az eljárást $P(EC)^m$ **módszernek** nevezzük.

Sokan hozzávesznek a fentiekhez még egy kiértékelő lépést is, vagyis még pluszban az

$$(E) \qquad f_n^{(m)} = f(t_n, y_n^{(m)})$$

értéket is meghatározzuk. Ezt azért számoljuk ki, mert amikor áttérünk a következő időpontra azaz a t_{n+1} pontban határozzuk meg a közelítést a $P(EC)^m$ módszerrel, akkor a jósló (C) formulában a jobb oldalon $\hat{\beta}_1 f_{n-1}$ helyett $\hat{\beta}_1 f_n$ szerepel majd, és ehhez az $f_n = f_n^{(m)}$ értéket használjuk. Ezzel a kiegészítéssel a módszert $P(EC)^m E$ **típusú módszernek** nevezzük.

3.2.6. Definíció. *A $P(EC)^m$ és $P(EC)^m E$ típusú módszereket* ***prediktor-korrektor*** *(avagy jósló-javító) módszernek nevezzük.*

4. fejezet

Neurális közönséges differenciálegyenletek

Ahogy azt a 2. fejezet legvégén láthattuk a reziduális hálók egy blokkjában (2.3) explicit iteratív eljárást vehettünk észre. Az ottani motiváció mellett, valamint a 3. fejezet ismereteivel felvértezve folytonos határátmenetet képezve tulajdonképpen egy w paraméterrel rendelkező

$$\begin{aligned} \frac{\mathrm{d}u(t)}{\mathrm{d}t} &= f(t, u(t), w) \\ u(0) &= u_0 \end{aligned}$$

kezdetiérték-feladathoz jutunk. A 3. fejezetben ismertetett alapvető módszereinek segítségével (és egyes esetekben alkalmas ún. változó lépésközű verzióinak felhasználásával) meg tudjuk oldani a fenti feladatot. Ezt a felírási módot a dolgozat alapjául szolgáló [1] cikk mellett már többen is észrevették. Például megtalálható ez Lu és társszerzői [14], valamint Haber és Ruthotto [7] is.

Az [1] cikk többek között arra keresi a választ, hogy egyáltalán miért érdemes ezzel a témakörrel foglalkozni és milyen előnyökkel járhat. Röviden összefoglalva lentebb ezeket az előnyöket soroljuk fel.

Hatékonyabb memóriafelhasználás

A szerzők ötletének köszönhetően (lásd következő alfejezet) létezik olyan gradiens meghatározási mód, amelyek során kezdetiérték-feladatokat oldunk meg és nem

szükséges a közbülső lépések értékeinek tárolása. Ez lehetővé teszi a modell tanítását úgy, hogy közben a felhasznált memória állandó értékű a mélység függvényében.

Széleskörű választék kezdetiérték-feladatok megoldására A 3. fejezetben láthattuk, hogy a több mint 120 év során nagy választék áll rendelkezésünkre ilyen típusú feladatok numerikus megoldására. Mivel a dolgozat kereteit meghaladja, de már az említett változó lépésközű (adaptív) módszerek segítségével robusztus, nagy pontosságú módszereket használhatunk, mely figyeli a lokális hiba mértékét és ennek megfelelően választ lépésközt. Ez pedig egyrészt azonnali költségcsökkentést is jelent [8], [9].

Paraméterezés hatékonyságának javulása
Folytonosnak tekinthető esetben a paraméterezésről is elmondható, hogy egy folytonos függvényként valósul meg. Így a felügyelt tanítás során felhasznált paraméterek száma csökkenthető.

Folytonos idősor alapú modellek
A folytonos felépítésű modellek szerkezetükből adódóan képesek feldolgozni a tetszőleges időpontokban érkező adatokat, ami által hatékony eszközei lehetnek a folytonos idősor alapú modellek elemzésének. Fontos alkalmazási terület lehet a pénzügyi idősorok területe.

4.1. Felügyelt tanítás és az adjungált egyenlet

A neurális hálókról szóló részben már tárgyaltuk, hogy hogyan is zajlik egy háló tanítása. Egyrészt definiálunk alkalmas módon egy veszteségfüggvényt (például többrétegű perceptron esetében (2.1) egyenlet jobboldala), majd ezt optimalizáljuk (abban az esetben a hibavisszaterjesztéses algoritmus segítségével). Az előző esetben, azaz diszkrét neurális hálók esetében ez az algoritmus (még hogyha igen sok művelet elvégzésére is volt szükség) matematikailag viszonylag egyszerűen tudta kezelni a tanítást.

Ugyanakkor ebben a folytonos esetben a hibavisszaterjesztéses algoritmus alkalmazása a kezdetiérték-feladatot módszer által szolgáltatott megoldáson sajnos a megszokott módon nem lehetséges. Ennek feloldását tárgyaljuk. A feladat mivoltából adódóan tekintsük a következő skalár értékű

$$L(u(t_1)) = L(u(t_0) + \int_{t_0}^{t_1} f(t, u(t), w)\mathrm{d}t) = L(\mathrm{ODESolve}(u(t_0), f, t_0, t_1, w)) \tag{4.1}$$

veszteségfüggvényt, ahol a függvény bemenetét a kezdetiérték-feladatot megoldó (röviden ez az angol elnevezésből adódó) ODESolve általunk választott numerikus eljárás szolgáltatja. A fenti (4.1) egyenlet optimalizálásához L w szerinti gradiensének kiszámolására van szükségünk. A részletes számítási folyamatot mellőzzük, csupán a legfontosabb lépésekre szorítkozunk. Az érdeklődő Olvasó az [1] függelékeiben megtalálhatja ezeket. Az optimalizálásnak két lépése van.

1) Először meghatározzuk a veszteségfüggvény gradiensét $u(t)$ minden pillanatában, azaz

$$a(t) = \frac{\mathrm{d}L}{\mathrm{d}u(t)}.$$

Ez az érték az ún. **adjungált**. A dinamikáját egy másik közönséges differenciálegyenlet adja meg, melyet egyfajta láncszabály analógiaként képzelhetünk el. Ez a

$$\frac{\mathrm{d}a(t)}{\mathrm{d}t} = -a(t)\frac{\mathrm{d}f(t, u(t), w)}{\mathrm{u(t)}} \tag{4.2}$$

közönséges differenciálegyenlet, melyet egy másik ODESolve segítségével numerikusan megoldhatunk. Vegyük észre, hogy ezt az egyenletet időben visszafelé a $\mathrm{d}L/\mathrm{d}u(t_1)$ kezdeti értékkel kell megoldanunk. A cikk egyik másik szép elméleti eredménye, hogy a teljes $u(t)$ teljes trajektóriáját ismerjük. Ugyanis $u(t)$ időben visszafelé számítható a fenti adjungált módszerrel (az eredeti gondolat a [15] könyvben megtalálható).

2) A w szerinti gradiens kiszámítása egy további integrál kiértékelését igényli, mely egyaránt függ $u(t)$ és $a(t)$ értékétől. Ez a

$$\frac{\mathrm{d}L}{\mathrm{d}w} = -\int_{t_1}^{t_0} a(t)^T \frac{\mathrm{d}f(t, u(t), w)}{\mathrm{d}w}\mathrm{d}t \tag{4.3}$$

kiértékelést jelenti.

A (4.2)-(4.3) egyenletekben szereplő

$$a(t)^T \frac{\mathrm{d}f(t, u(t), w)}{\mathrm{d}u(t)} \text{ és } a(t)^T \frac{\mathrm{d}f(t, u(t), w)}{\mathrm{d}w}$$

vektor-Jacobi mátrix típusú szorzások ún. automatikus differenciálással kiértékelhetőek, ráadásul f kiértékelésével összemérhető költségen. Mindhárom integrál egy ODESolve hívással kiértékelhető és ezek egyetlen vektorba fűhetőek.

Az eljáráshoz tartozó pszeudokód az alábbi:

Input: w, t_0, t_1 , $u(t_1)$, $\mathrm{d}L/\mathrm{d}u(t_1)$

$\frac{\mathrm{d}L}{\mathrm{d}(t_1)} = \frac{\mathrm{d}L}{\mathrm{d}z(t_1)}^T f(t_1, u(t_1), w)$ ▷ Gradiens t_1 szerint

$s_0 = [u(t_1), \frac{\mathrm{d}L}{\mathrm{d}u(t_1)}, \mathbf{0}_{|w|}, -\frac{\mathrm{d}L}{\mathrm{d}t_1}]$ ▷ Kezdeti értékek definiálása

def dinamika$(t, [u(t), a(t), \cdot, \cdot], w)$ ▷ Dinamika definiálása

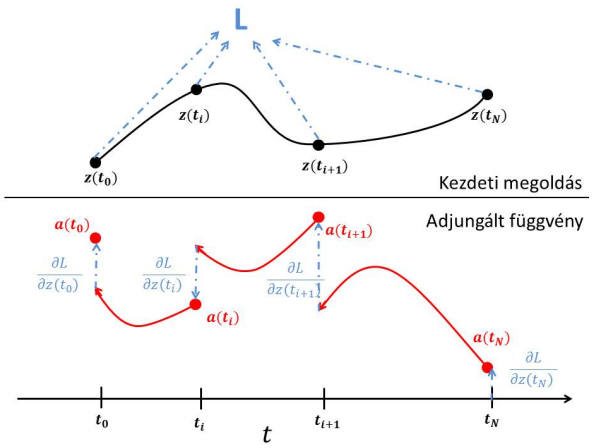
return $[t, f(u(t), w), -a(t)^T \frac{\mathrm{d}f}{\mathrm{d}u}, -a(t)^T \frac{\mathrm{d}f}{\mathrm{d}w}, -a(t)^T \frac{\mathrm{d}f}{\mathrm{d}t}]$ ▷ Vektor-Jacobi számolás

$[u(t_0), \frac{\mathrm{d}L}{\mathrm{d}u(t_0)}, \frac{\mathrm{d}L}{\mathrm{d}w}, \frac{\mathrm{d}L}{\mathrm{d}t_0}] = \mathrm{ODESolve}(s_0, \mathrm{dinamika}, t_1, t_0, w)$ ▷ Vissza idejű megoldás

return $\frac{\mathrm{d}L}{\mathrm{d}u(t_0)}, \frac{\mathrm{d}L}{\mathrm{d}w}, \frac{\mathrm{d}L}{\mathrm{d}t_0}, \frac{\mathrm{d}L}{\mathrm{d}t_1}$ ▷ Gradiensek visszaadása

---

Mindeddig feltettük, hogy a veszteségfüggvény csak egyetlen időpillanattól függ. Ha közbülső $t_1, t_2, \ldots, t_{N-1}$ időpillanatok is közrejátszanak, akkor a fenti lépéseket az ezek által meghatározott $[t_{N-1}, t_N], [t_{N-2}, t_N - 1], \ldots, [t_0, t_1]$ részintervallumokra iteratív módon alkalmazzuk és a nyert gradienseket összegezzük. A teljes eljárás sémáját szemlélteti több részintervallum esetén a 4.1. ábra.



4.1. ábra. A teljes eljárás sematikus ábrája.

4.2. Implementáció és példák

Az [1] cikk szerzői az előző alfejezetben ismertetett eljárást egy kompakt Python könyvtárként implementálták [2]. Az ODESolve kezdetiérték-feladat megoldói között megtalálhatjuk a 3. fejezetben megismert explicit Euler, explicit középponti, 3/8-os RK4, explicit Adams és implicit Adams módszereket. Ezen módszerek mellett pár változó lépésközű és rendű módszer is megtalálható.

A már megismert eljárást a képfeldolgozásban legnépszerűbb MNIST (Modified National Institute of Standards and Technology database) adatbázison is tesztelték. Egy korábbi eredményt felhasználva az egyrétegű MLP hálót [13], a reziduális hálók elméletén alapuló ResNetet, a saját eljárásukat az ODE-Netet, illetve annak direkt visszacsatolásos Runge–Kutta verzióját (amely nem az adjungált módszeren alapszik) RK-Netet hasonlították össze. Ezek eredményeit a lenti táblázat foglalja össze.

	Teszt hiba	Paraméterek száma	Memória	Idő
1 MLP	1.60%	0.24 millió	-	-
ResNet	0.41%	0.60 millió	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 millió	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 millió	$\mathcal{O}(1)$	$\mathcal{O}(\tilde{L})$

A fenti táblázatban L a ResNetben lévő rétegek számát jelöli, míg $\tilde{L}$ az ODESolve függvényhívásainak számát egyetlen lépés során. Látható, hogy az ODE-Net a legjobban teljesítő eljárás. Habár az RK-Net hasonlóan teljesít, de az ODE-Net memóriaigénye sokkal jobb.

Egy további érdekes analízis volt részükről a betanított ODE-Net statisztikája. Eredményeiket összefoglalva elmondhatjuk, hogy

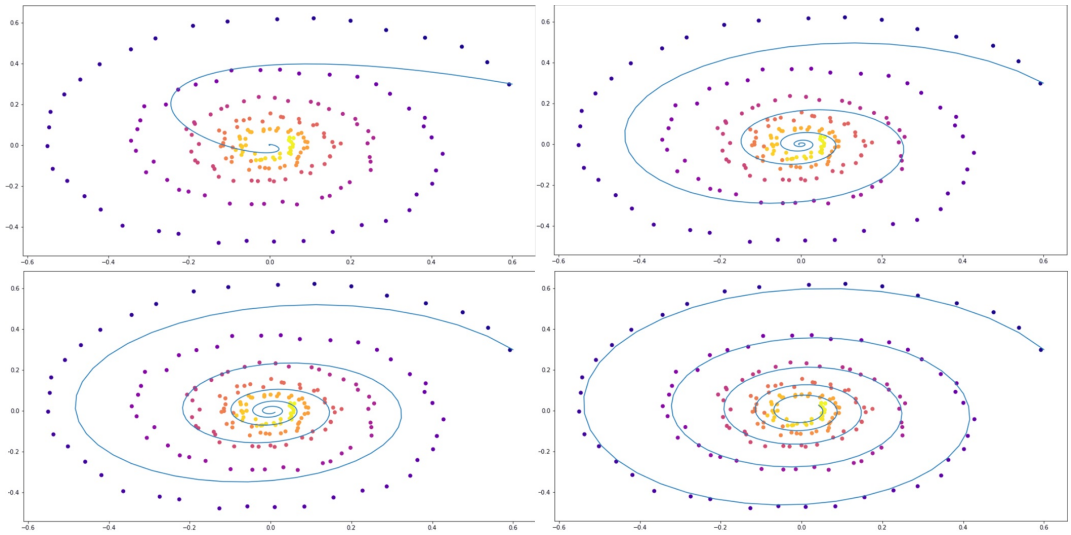
- a hiba kontrollálható,
- a forward hívások száma időben arányos a függvényhívások számával.

Eddigi vizsgálódásunk alapján is könnyen látható, hogy ebben a kontextusban a háló mélysége nehezen definiálható. A tapsztalat azt mutatja, hogy az epochok számának növelésével a függvényhívások száma is nő.

Végezetül a [17] GitHub könyvtár Jupyter Notebookját használva egy egyszerű példán keresztül demonstráljuk a részletesen tárgyalt ODE-Net típusú neurális közönséges differenciálegyenlet módszerét. A kezdetiérték-feladatot az explicit Euler módszerrel oldjuk meg. A háló az alábbi

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = \begin{bmatrix} -0.1 & -1.0 \\ 1.0 & -0.1 \end{bmatrix} u(t)$$

differenciálegyenlet rendszer trajektóriáját tanulja meg egy kezdeti trajektória alapján. A tanulás folyamatát a 4.2. ábra szemlélteti.



4.2. ábra. A tanulási folyamat balról jobbra fentről lefelé szemlélteve.

5. fejezet

Összefoglalás

Az adattudomány és ezen belül a gépi tanulás, mélytanulás területén még megannyi fejlődési lehetőség van, amihez mi csak egy kis ízelítőt tudtunk nyújtani a dolgozatunkkal. A tudomány ezen területén a fejlős iránti vágy továbbra is szakadatlanul erősödik, ami előre sejteti számunkra, hogy a gépi tanulás és az ezzel kapcsolatos alkalmazások egyre nagyobb jelentőségnek fognak örvendeni a jövőben.

A neurális közönséges differenciálegyenletek elmélete reziduális (és más alkalmas) hálók esetében közönséges differenciálegyenleteket megoldva és az adjungált módszer segítségével egy új felügyelt tanítási eljárást tesz lehetővé. Az elmélet új kérdéseket vet fel és nagy visszhangjának köszönhetően ezeken már számtalan kutató dolgozik is.

A neurális közönséges differenciálegyenletek pedig úgy vélem ékes példái annak, hogy születhet még új a nap alatt, amihez sokszor nem kell más, csak hogy a már meglévő, jól kiforrott tudásunkra más szemszögől nézzünk.

Irodalomjegyzék

[1] T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud: Neural Ordinary Differential Equations, Advances in Neural Information Processing Systems 31, p. 6571–6583 (2018)

[2] T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud: Neural Ordinary Differential Equations, GitHub repository, https://github.com/rtqichen/torchdiffeq (2018)

[3] Faragó István, Horváth Róbert: Numerikus módszerek, TypoTeX Kiadó (2016)

[4] Faragó István, Numerikus modellezés és közönséges differenciálegyenletek numerikus megoldási módszerei, Jegyzet (2013)

[5] Fazekas István: Neurális hálózatok, Jegyzet (2013)

[6] X. Glorot and Y. Bengio: Understanding the difficulty of training deep feedforward neural networks, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256 (2010)

[7] E. Haber and L. Ruthotto: Stable architectures for deep neural networks. Inverse Problems, 34 (1):014004 (2017)

[8] E. Harier, S. P. Nørsett and G. Wanner: Solving Ordinary Differential Equations I, Nonstiff Problems, Springer (1987)

[9] E. Hairer and G. Wanner: Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems, Springer (1991)

[10] S. Haykin: Neural networks: A Comprehensive Foundation, Prentice Hall, 2nd Edition (1999)

[11] K. He, X. Zhang, S. Ren and Jian Sun: Deep Residual Learning for Image Recognition. CVPR, p. 770–778 (2016)

[12] A. Krizhevsky, I. Sutskever and G. E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems 25, p. 1097–1105 (2012)

[13] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-based learning applied to document recognition, Proceedings of the IEEE, 86(11):2278–2324 (1998)

[14] Y. Lu, A. Zhong, Q. Li and B. Dong: Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations, Proceedings of the 35th International Conference on Machine Learning, PMLR 80:3276–3285 (2018)

[15] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze and E. F. Mishechenko: The Mathematical Theory of Optimal Processes, John Wiley & Sons (1962)

[16] Simon L. Péter, Közönséges differenciálegyenletek, Jegyzet (2007)

[17] https://msurtsukov.github.io/Neural-ODE/ (2019)