

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Ivanics Kristóf

Járvány modellezés neurális differenciálegyenletekkel

Konzulens

Dr. Bolgár Bence

BUDAPEST, 2021

Tartalomjegyzék

[Összefoglaló 4](#_Toc89881173)

[Abstract 5](#_Toc89881174)

[1 Bevezetés 6](#_Toc89881175)

[2 Differenciálegyenletek 7](#_Toc89881176)

[2.1 Analitikus megoldás 7](#_Toc89881177)

[2.2 Numerikus megoldás 9](#_Toc89881178)

[2.2.1 Egylépéses módszerek 9](#_Toc89881179)

[2.2.2 Lineáris többlépéses módszerek 11](#_Toc89881180)

[2.2.3 Implicit és explicit módszerek 12](#_Toc89881181)

[2.2.4 Adaptív módszerek 12](#_Toc89881182)

[3 Neurális hálózatok 14](#_Toc89881183)

[3.1 Előrecsatolt neurális hálózatok 14](#_Toc89881184)

[3.2 Rekurrens hálózatok 18](#_Toc89881185)

[3.3 Neurális háló tanítási technikák 20](#_Toc89881186)

[Irodalomjegyzék 21](#_Toc89881187)

Hallgatói nyilatkozat

Alulírott **Ivanics Kristóf**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 08.

...…………………………………………….

Ivanics Kristóf

Összefoglaló

A járvány idején a lokális vírus kitörések[1] azonosításának és előrejelzésének képessége kulcsfontosságú ahhoz, hogy az egészségügyi szervek megfelelő lépéseket tegyenek. Közvetlenül vagy közvetve nagy mennyiségű adat keletkezik a vírus terjedésével kapcsolatban, különféle adatforrásokból. A források közé tartozik a közösségi médiában történt felmérések, az internetes szokásokról történt elemzés, a mobiltelefon GPS adatai és a mobil fitnesz eszközök. Ezek az adatsorok földrajzi régiókra bontva adottak. Ebben a dolgozatban az adat alapú modellek prediktív képességét vizsgálom a COVID19-hez kapcsolódó adatok sorok felhasználásával. Reális azt várni, hogy a felmérések válaszainak időbeli változása egy régióban korrelál az adott régió új fertőzések számával. Ezen túlmenően ezek a jelek potenciálisan korai előre jelzői lehetnek a helyi víruskitöréseknek. Ennek a kapcsolatnak a megismerése érdekében egy neurális differenciálegyenletet (neurális ODE) [2] alkalmaztam a jel változásának sebességének előrejelzéséhez.

A mély neurális hálózat modellek új családját mutatták be az elmúlt pár évben. Ahelyett, hogy megadnánk a rejtett rétegek diszkrét szekvenciáját, egy neurális hálózat segítségével paraméterezzük a rejtett állapot deriváltját. A hálózat kimenetét egy differenciálegyenlet megoldóval számítjuk ki. Ezeknek a folyamatos mélységű modelleknek állandó memória költségekkel tudnak dolgozni, kiértékelési stratégiájukat (pl.: lépésköz) az egyes bemenethez igazítják, és nagy előnyük, hogy numerikus pontosság és sebesség egyensúly könnyen paraméterezhető a lépésközzel. Ez az új hálózat család a neurális differenciálegyenletek, amik segítségével modelleztem a különböző adatsorok közötti összefüggést.

Abstract

During a pandemic, the ability to identify and forecast local virus outbreaks[1] is key in order for health officials to take appropriate action. A large amount of data is being generated, directly or indirectly, related to the spread of the virus, on various spatial and temporal scales. Sources include social media, internet habits,mobile phone GPS, and mobile fitness devices. These data series(signals) are given for geographic regions. In this work I investigate the predictive power of data-driven models using data generated related to COVID19. It is reasonable to expect that the local variation in time of the survey responses for a region is correlated with new virus infections in that region. Moreover, these signals have the potential of being early indicators of local virus outbreaks. In order to learn this relation, a neural ordinary differential equation (neural ODE) [2] was used to parameterize the signal's rate of change.

A new family of deep neural network models has been presented in previous years. Instead of specifying a discrete sequence of hidden layers, we parameterize the derivative of the hidden state using a neural network. The output of the network is computed using a blackbox differential equation solver. These continuous-depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed. This new family of networks is the neural differential equations, which I used to model the relationship between the different signals.

# Bevezetés

A feladat egy olyan hibrid neurális differenciálegyenlet (későbbiekben: NODE) létrehozása volt, amely alkalmas a járványterjedés dinamikájának modellezésére. A hálózatot való életből vett adatokon tanítottam. Ezek az adatok publikusan elérhetőek API-n keresztül[3]. Éppen ezért a feladat megoldásának első két lépése a neurális differenciálegyenletekről történő irodalomkutatás és az adatsorok beszerzése volt az API -n keresztül. Ezt követte a NODE implementációja a választott gépi tanulás keretrendszerben. Erre én a Julia programozási környezetet használtam. Az utolsó lépés az illesztett modell kiértékelése, illetve prediktív képességének a vizsgálata volt. Itt a tanítás folyamán keletkezett veszteségeket elemeztem, valamint a modell prediktív és általánosító képességét ábrázoltam. A jelen dokumentum felépítése követni fogja a feladatok elvégzésének sorrendjét.

A feladat indokoltsága abszolút megkérdőjelezhetetlen, hiszen mind a feladat célja: lokális koronavírus kitörések előrejelzése, mind az alkalmazott technológia: neurális differenciálegyenletek kutatás tárgyát képzik napjainkban. Sajnos az adatok csak az Amerikai Egyesült Államok területére érhetők el ezért a magyar körülményeket nem tudtam elemezni. Ez a feladat szempontjából nem releváns, de érdekes lett volna az általunk ismert járványgörbéket viszont látni a feladat keretein belül.

Korábban végeztem hasonló témában kutatást, abban a munkában a differenciálegyenletek paramétereinek becslésével foglalkoztam. A paraméterek becslését gradiens alapú optimalizációval végeztem. Jelen dokumentumban paraméter kereséssel nem fogok foglalkozni, hiszen azt korábban megtettem, csak klasszikus neurális differenciálegyenletekkel és azok felhasználásáról.

# Differenciálegyenletek

Az olyan egyenleteket, melyben az ismeretlen függvény deriváltja, illetve deriváltjai szerepelnek, differenciálegyenleteknek[4] nevezzük. Tehát egy differenciálegyenletben szerepelhetnek:

* konstansok;
* egy vagy több független változó;
* az ismeretlen függvény, illetve függvények közönséges, illetve parciális deriváltja, illetve deriváltjai

Ha a differenciálegyenletben egyetlen független változó van, akkor a derivált közönséges derivált. Ebben az esetben közönséges differenciálegyenletekről beszélünk. Ezen kívül léteznek a parciális differenciálegyenletek, ha a differenciálegyenletben kettő vagy több független változó van. Valamint algebro-differenciálegyenletek, ahol a differenciálegyenletek mellett a megoldásnak az algebrai mellékfeltételnek is eleget kell tennie. És sok egyéb differenciálegyenlet típus létezik, de ezeket bemutatás szinten sem érintem.

Ebben a dolgozatban közönséges differenciálegyenletekkel fogok foglalkozni. Ezek az alábbi alakokat vehetik fel, implicit alak:

(2.1)

Explicit alak:

(2.2)

## Analitikus megoldás

Ha a differenciálegyenlet felírható bizonyos alakokban akkor a differenciálegyenletnek megadható az analitikus megoldása. Mutatok néhány alakot, amiknek megadható az analitikus megoldása. Korlátozzuk magunkat most az elsőrendű differenciálegyenletekre, ezek olyan egyenletek, ahol az ismeretlen függvénynek csak az első deriváltja jelenik meg.

Ha ebből az 2.1 vagy 2.2 egyenletből az y’ kifejezhető, akkor elsőrendű explicit differenciálegyenletről beszélünk ezeket a következő alakban írhatjuk fel:

(2.3)

Az adott x0, y0 esetén jutunk a Cauchy problémához, más néven kezdetiérték probléma (IVP: initial value problem).

(2.4)

Azt mondjuk, hogy a ϕ megoldásfüggvénye a (2.4) Cauchy problémának, ha van olyan (a, b) intervallum (a < b), hogy

(2.5)

ϕ-nek a grafikonja az (2.4) Cauchy probléma megoldásgörbéje.

A 2.3-as egyenlet mind a két oldal ki-integrálása után az alábbi alakot kapjuk, ha az függvény integrálható x szerint:

(2.6)

ahol megfelelő C érték választásával teljesíthető a Cauchy probléma. Egy másik alak, ahol viszonylag egyszerű a megoldásgörbe kiszámítása a szeparábilis differenciálegyenletek, ezek a következő alakba írhatók:

(2.7)

ahol f és g is folytonos az adott intervallumon. A szeparábilis egyenleteknek megoldásgörbéit az alábbi módon kaphatjuk meg:

(2.8)

Persze itt is fel kell tennünk azt, hogy ezek a függvények integrálhatóak. A dolgozatnak nem célja minden analitikusan megoldható differenciálegyenletet bemutatni ezért több analitikus megoldást nem mutatok be csak felsorlom az így megoldható differenciálegyenletet típusokat a teljesség igénye nélkül: közvetlenül integrálható egyenletek (2.6), szeparábilis egyenletek (2.8), homogén egyenletek, lineáris egyenletek, Bernoulli-féle egyenletek, egzakt egyenletek.

## Numerikus megoldás

Az előző fejezetben láthattuk, hogy mely differenciálegyenletet típusoknak adható meg a matematikai megoldása bizonyos körülmények között. Minden ilyen megoldásnak feltétele volt, hogy integrálható legyen az integrálban található formula, ám ez nagyon sok problémánál nem áll fenn. Ezekben az esetekben alkalmazhatóak a numerikus differenciálegyenlet megoldók. Ezek valamekkora numerikus hiba mellett közelítik a megoldást. Ezeknek az iteratív numerikus megoldóknak én most két családját fogom bemutatni: az egylépéses [5] numerikus megoldókat és a lineáris többlépéses [6] megoldókat.

Az iteratív numerikus megoldóknak az a működési elvük, hogy a megoldásfüggvényt úgy közelítik, hogy az adott pontban vett deriváltja mentén lépnek egy lépésköznyit majd újra kiértékelik a deriváltat meghatározó függvényt abban a pontban, ahova az előző lépésben érkeztek. Tekintsük a 2.3-as egyenletet példának a derivált kiszámítása egy egyszerű behelyettesítés ezt minden feltétel nélkül megtehetjük. Fontos megjegyezni, hogy ehhez a megoldáshoz szükség van kezdeti értékekre, hiszen valamivel el kell kezdeni a helyettesítést.

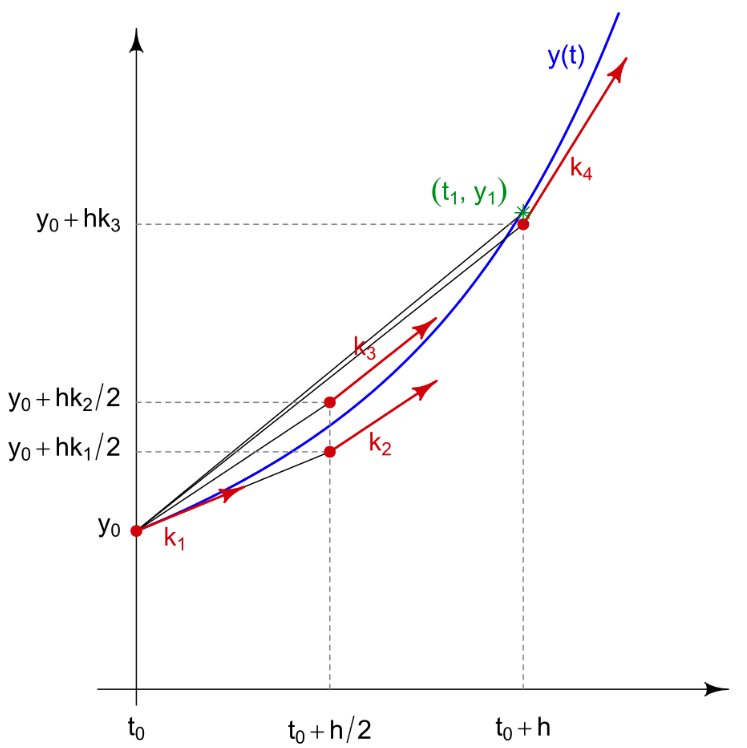
### Egylépéses módszerek

Az egylépéses iteratív numerikus megoldók csak az utolsó állapot alapján határozzák meg a következőt, ezért is hívják őket egylépéses módszereknek. Ezek közül a legegyszerűbb az Euler módszer, ahol a 2.4-es egyenletben megadott kezdetiérték problémát oldjuk meg az alábbi módon:

(2.9)

-t -val is szokás jelölni ez a lépésköz. Tehát azt jelenti, hogy egy iterációban mekkorát lépünk a megoldásgörbén. A 2.9-es egyenleten az történik, amit korábban már korábban részleteztem, a megoldásgörbe pontjait egy iteratív módszerrel határozzuk meg pontról pontra. A megoldásgörbét minden pontban közelítjük a deriváltján vett kellően kicsi lépéssel.

Az Euler módszernél vannak szofisztikáltabb egylépéses módszerek, ezek közül egyet be is mutatok a Runge-Kutta(RK) módszert. Ezt a módszert Carl Runge és Wilhelm Kutta fejlesztette ki körülbelül 1900 körül. Az Euler módszerhez képest az a fejlődés, hogy a lépésközben is kiszámítja a deriváltat bizonyos szabályok szerint így pontosabb képet kaphatunk arról, hogy következő pont helyzetéről.



2.1 Negyedrendű Runge-Kutta módszer [7]

Az Euler módszerhez hasonlóan itt is a 2.4-es egyenletben megadott kezdetiérték problémára keressük a numerikus megoldást. A 2.1-es képen a különböző értékeket az alábbi módon számíthatjuk ki:

(2.10)

(2.11)

(2.12)

(2.13)

Ezeknek a meredekségeknek a súlyozott átlagából kapjuk meg a végső becslésünket a -ra.

(2.14)

Az Euler módszerhez képest ez megnövekedett számítási kapacitást igényel, de cserébe a lokális és a globális is hiba a töredékére csökken. Ez számszerűsíthető is a negyedrendű Runge-Kutta módszer közelítés hibája negyedrendű ez azt jelenti, hogy választott lépésköz zsugorításakor annak negyedik hatványával zsugorodik a hibára adott felső becslés. Ennek a matematikai hátterét[8] nem tárom fel.

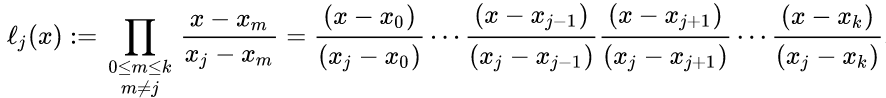
### Lineáris többlépéses módszerek

A lineáris többlépéses módszerek is a 2.4-es egyenletben vázolt kezdetiérték problémát oldják meg. A lineáris többlépéses módszerek azon gondolaton alapulnak, hogy a korábban kiszámított függvény értékeket érdemes lehet felhasználni. Ezt úgy teszik, hogy a korábbi pontokat interpolálják, ebből egy polinom keletkezik. Ezt a polinomot integrálják a következő lépésig. Adottak a korábban kiszámított pontok a görbén:

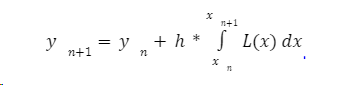


Ezekből a Lagrange interpolációs polinom így áll elő:

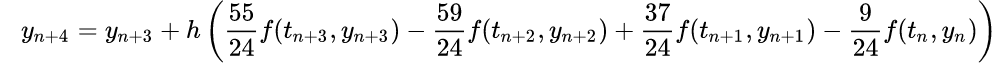
 (2.15, 2.16)



A lineáris többlépéses módszerek ezt a polinomot integrálják a következő lépésig:

 (2.17)

A polinom integrálása és az és az helyettesítésé után a korábbi pontoknak a lineáris kombinációját kapjuk meg. Ezt a módszert Adams-Bashforth módszernek nevezzük. A negyedrendű Adams-Bashforth módszer képlete (2.18):



A 2.15-2.17 -es egyenletek eredményét láthatjuk a 2.18-as egyenletben. Elvégeztük a Lagrange interpolációt, majd elvégeztük az integrálást és tényleg a korábbi pontok lineáris kombinációját kaptuk.

### Implicit és explicit módszerek

A numerikus megoldókat az alapján is csoportosíthatjuk, hogy azok explicit vagy implicit módszerek-e. Az eddig bemutatott módszerek mind explicit módszerek voltak, tehát az explicit ki volt fejezve az egyenletekben. Implicit esetben mind a két oldalon megjelenik az tag. E miatt nem adható meg egy pontban behelyettesítéssel a megoldásgörbe deriváltja. Tekintsük a következő példát:

A képen szöveg látható

Automatikusan generált leírás (2.18)

Ez, a korábbiakhoz hasonlóan egy explicit probléma az egyenlet átrendezésével és kezdeti értékek megadásával könnyen tudunk a korábban bemutatott iteratív numerikus megoldókkal megoldásgörbét keresni. Ennek a feladatnak az implicit változata a következőképpen néz ki:

A képen szöveg látható

Automatikusan generált leírás (2.19)

Ebben az esetben a korábbinál nehezebb dolgunk van, hiszen nem fejezhető ki explicit az tag. Jelen példában itt egy másodfokú egyenletet kell megoldani. Ezeket a magasabb rendű gyök kereséseket a gyakorlatban iteratív Newton-Raphson módszerrel szokták megtenni, vagy rögzített pont iterációval. Ez a számítási igény jóval megnöveli, de bizonyos problémáknál csak ilyen implicit megoldókkal tudunk stabil eredményt elérni, ezek a merev problémák. Implicit és explicit módszerekről remek intuitív magyarázat található a [9]-es hivatkozásban.

### Adaptív módszerek

Tulajdonképpen a neurális differenciálegyenletek legnagyobb előnye a residual network -ökkel szemben, hogy a problémához tud alkalmazkodni úgy, hogy adaptívan tudja a lépésközt változtatni. Míg a residual network-ök diszkrét előre definiált számú lépésből állnak a NODE-k "folytonos" mélységűek az adaptív lépésköz miatt. Erre mutatok példát mind egy lépéses mind többlépéses esetben.

Egylépéses esetben RK párok párhuzamosan becsülik a következő lépést, az RK párból az egyik mindig magasabb rendű a másiknál. Azt feltételezzük, hogy a magasabb rendű RK igen közel van a valódi megoldáshoz. Ha a kétféleképpen felparaméterezett RK módszer egymástól egy előre meghatározott toleranciaszintnél jobban eltér egymástól akkor a lépést újraindítjuk egy kisebb lépésközzel.

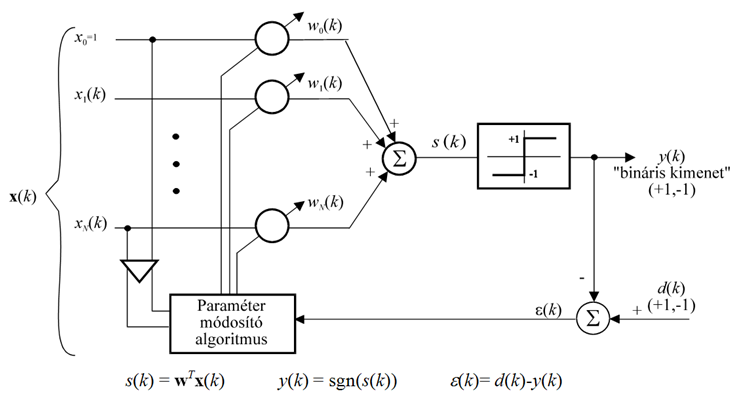
A lineáris többlépéses esetben is hasonlóan két numerikus megoldó dolgozik párhuzamosan. Az egyik egy korábban bemutatott Adams-Bashforth modell a másik pedig egy implicit Adams-Multon modell. Ez ugyan azon az elven működik, mint az Adams-Bashforth modell. Abból fakadóan, hogy ez egy implicit verziója a többlépéses módszereknek más lineáris kombinációját kell venni a megoldásgörbe pontjainak. Annyi trükk van a módszer implementációban, hogy az explicit módszerben megkapott y(k+1) pontot helyettesíthetjük be az implicit módszerbe így az a plusz gyökkeresés kihagyható. Az adaptív RK módszerhez hasonlóan itt is a két módszer közötti különbséget viszonyítjuk a toleranciaszinthez. Ezt a módszert Adams-Bashforth-Multon módszernek vagy predictor-corrector módszernek szokták hívni.

# Neurális hálózatok

A neurális differenciálegyenletek másik összetevője a differenciálegyenleteken kívül a nevéből adódóan a neurális hálózatok. Ebben a fejezetben bemutatom a neurális hálózatokat. Ez csupán egy áttekintés lesz a neurális hálózatokról, hiszen a feladat megoldásához nem feltétlenül szükséges ismerni az összes hálózat típust és az ezekhez tartozó tanulási módszereket. A feladat megoldásához főként arra alapozunk, hogy a neurális hálózatok alkalmazhatók univerzális függvény approximátorként. Ez egy valódi függvény közelítését jelenti. Szükség lesz még arra a tudásra, hogy hogy lehet egy adatsorra modellt illeszteni. A megfelelő eredmény érdekében itt több technikát is be kellett vetnem, ezeket később részletezem. A differenciálegyenlet megoldóknak a (2.9)-es egyenletben látható függvényre van szüksége. Nekünk adatsoraink állnak rendelkezésre ehhez hasonló függvény pedig nem. Ezért az adatok felhasználásával ezt a függvényt közelítjük egy előrecsatolt neurális hálózattal. Az imént tárgyaltak alapján az előrecsatolt hálózatokat fogom bemutatni és az adatsor modellezéséhez szükséges technikákat. Nem fogok a neurális hálózatok mélységeibe belemenni hiszen nem ez a dolgozat célja, csak annyira érintem a területet, amennyit mindenképp érdemes tudni, ha a neurális differenciálegyenleteket szeretnénk fejleszteni. Illetve ezeknek a neurális differenciálegyenletek előnyei és hátrányait szeretnénk megvizsgálni.

## Előrecsatolt neurális hálózatok

Definíció szerint az előrecsatolt neurális hálózat, olyan háló, amelynek gráfbeli reprezentációja nem tartalmaz hurkot. Ez azt jelenti, hogy a hálózatban nem terjed visszafele információ csak a bemenettől a kimenet felé. Ezek az előrecsatolt hálók elemi neuronokból állnak. Ezek a neuronok is tulajdonképpen neurális hálózatok, amik lineárist leképzést tudnak megvalósítani. Ezek a neuronok az idegsejt által inspirált, ennek néhány tulajdonságát megragadó modellek. Az első ilyen neuront Frank Rosenblatt javasolta 1956-ban ő ezt a biológiai vetülete miatt perceptronnak nevezte el. Ezt a perceptront bemutatom, mert ez tekinthető a mesterséges intelligencia tudományág egyik alapkövének.



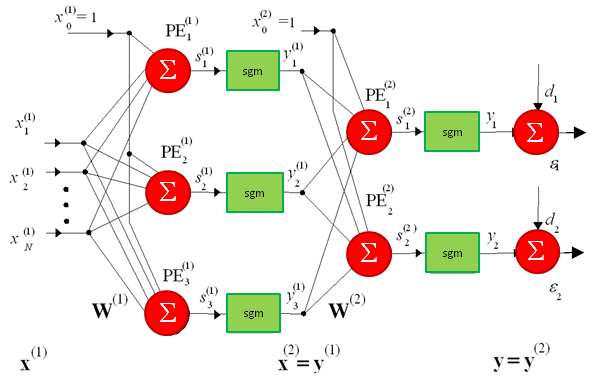
3.1. ábra - Az egyszerű perceptron felépítése a hibaképzéssel és a paramétermódosítással [12]

Az aktivációs függvény (ami jelen esetben egy előjel függvény) bemenete a hálózat bemenetének és a hálózat súlyainak a lineáris kombinációja. Az aktivációs függvény jelen esetben binárissá alakítja a lineáris kombinációt, tehát ez egy bináris osztályozást végez. Ez geometriailag azt jelenti, hogy a teret a súlyok által definiált sík (vagy hipersík) két részre osztja. A bemenet vektor ki van egészítve egy egyessel, amihez tartozik súly. Ez az eltolást valósítja meg. Ezért érdemes általában az adatokat normalizálni, hogy ezt a síkot ne kelljen olyan nagyon eltolni az origóból ezzel effektívebb tanítást valósíthatunk meg. A veszteség itt az egész hálózat kimenetének és a mintához tartozó címkének a különbsége. Fontos még ejteni szót arról, hogy a paraméter módosító algoritmus, hogy működik. A súlyok módosítását a következő egyenlet mutatja be:

(3.1)

A 3.1 -es egyenlet alapján súlyok akkor módosultnak csak, hogyha a kimenet és a bemenethez tartozó címke nem egyezik meg, tehát csak akkor módosulnak a súlyok, ha a bináris besorolás hibás lesz. Ebből következik, hogy ez a módszer nem törekszik arra, hogy a hipersík minél nagyobb margóval válassza el a két osztályt, mint például a szupport vektor gépek. Ha a hiba viszont nem nulla, akkor a bemeneti vektorral ellentétes irányba forgatjuk el a súlyvektort, ezzel módosítás után a súlyvektor és a bemenet szorzata közelebb lesz a kívánt értékhez. A Rosenblatt percepronhoz hasonló neuront hozott létre három évvel később Bernard Widrow. Az általa ajánlott neuronban a veszteséget nem a neuron kimenet alapján számították ki, hanem a súly-bemenet skaláris szorzat és a kívánt címke különbségét vették. Ebből adódóan itt nem csak akkor változnak a súly paraméter értékei, amikor a kimenet hibás, hanem minden egyes iterációba a súlyt a kívánt kimenet és a valós kimenet között vett négyzetes eltérés értéke alapján változtatja a paraméter módosító algoritmus. Ez a struktúra adaline-néven lett híres.

Az elemi perceptronokból építhető komplexebb struktúrával bonyolultabb nemlineáris leképzéseket is megvalósíthatunk ezeket Multi Layer Perceptronnak hívjuk. A nevéből adódóan ezek több rétegbe szervezett neuronokat jelent. Fontos, hogy a kimenet a súlyok folytonos differenciálható függvénye legyen. Valamint az, hogy az alkalmazott aktivációs függvények nemlineárisak legyenek különben a hálózat csupán arra lenne képes, hogy a bemenet és egy vektor lineáris kombinációját kiszámolja.



3.2. ábra - A többrétegű perceptron felépítése [12]

A 3.2-es ábrán látható egy többrétegű perceptron felépítése. Az bemutatott példában egy kétrétegű perceptront fogunk megvizsgálni. Ez ugyan olyan perceptronokból épül fel, mint amiket korábban tárgyaltam. Az ábrán a PE feliratú körök az sgm feliratú téglalapokkal kiegészítve jelölnek egy perceptront. Minden perceptronnál a felső indexbe található, hogy melyik rétegben tartozik az alsó indexben pedig az látható, hogy abban a rétegben ez hányadik perceptron. A második rétegben található perceptronok bemenetei az első rétegbe található perceptronoknak a kimenetei. A súlyok optimális értékének a megkeresése összetartozó bemenet-kimenet párok segítségével történik. Ez a felügyelt tanulásnak a definíciója. Legtöbbször az adaline-nál bemutatott négyzetes hibát szokás használni az MPL-nél is a veszteség meghatározására. A struktúra miatt biztosak lehetünk abban, hogy a hibafelület a tanítandó paraméterek folytonos és paraméterek szerint differenciálható függvénye. Ez teszi lehetővé a gradiens alapú módszereket. Ez heurisztikusan azt jelenti, hogy minden paraméterre kiszámítjuk, hogy az adott paraméter értéke mennyire befolyásolta a veszteséget. Majd ez alapján frissítjük a paramétereket. Ezen az alapon működik, a még ma is legelterjedtebb tanító eljárás a back-propagation [13]. Ezt nem fogom külön részletezni, mert az akár egy egész diplomamunka témája is lehetne.

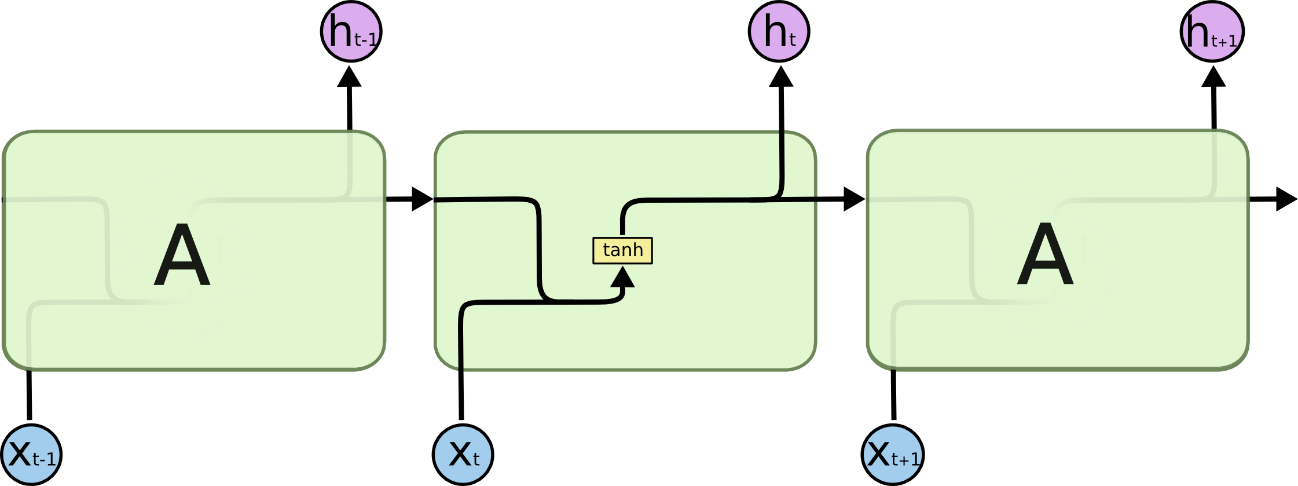


3.3 Egy neurális hálózat gráf ábrázolása [14]

A 3.3-as ábrán látható egy neurális hálózat gráfként történő ábrázolása. Ezt szokták fully connected (teljesen csatolt) hálózatnak is nevezni azért, mert az egymást követő rétegben minden neuron össze van kötve minden neuronnal. Én ilyen fully connected előrecsatolt hálókat használtam arra, hogy a (2.9) -es egyenletben látható függvényt approximáljam. Ezeknek a hálóknak a tanítása szintén az említett és hivatkozott back-propagation eljárással bizonyult a leghatékonyabbnak. Itt a bemeneti és kimeneti neuronok száma viszonylag egyértelműen megkapható a feladatból. A bemeneti réteg annyi neuronból áll, amennyi numerikus elemre felbontható a tanítóak mintapont készlet bemeneti értékei. Azét emeltem ki a numerikus elem kifejezést, mert a bemenetbe lehet akár kategorikus változó is, amit először le kell számokra képezni, hogy a hálózat számára értelmezhető legyen. A kimeneti réteg pedig annyi neuronból áll, hogy a hálózat kimenete és a mintaponthoz tartozó kimeneti érték könnyen összehasonlítható legyen. Ennél sokkal érdekesebb kérdés az, hogy a rejtett rétegek, amik a kimeneti és bemeneti rétegek között találhatóak, hogy épüljenek fel. Megfelelő számú és méretű rejtett réteget kell a hálózatunkba tennünk, hogy a lehető legjobb közelítést érjük el. Ha a hálózat túl nagy sok adatra és sok iterációra van szüksége, hogy az összefüggéseket a paramétereinek beállításával rögzítse. Ezzel ellentétbe, ha a hálózat túl egyszerű akkor lehet, hogy nem képes az adatokból az összefüggéseket megtanulni. Persze, ha jó hálózat struktúrával próbálkozunk akkor se garantált a siker, illetve nem optimális hálózat struktúrákból is lehet jó eredményt kihozni. A háló struktúráján kívül a siker múlik a tanulási hiperparaméterek megválasztásán és a tanulási technikán is, ezeket későbbi fejezetekbe részletezem.

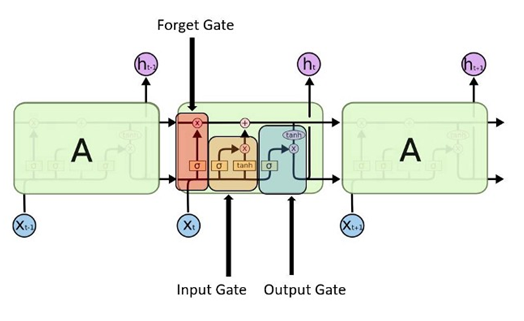
## Rekurrens hálózatok

A rekurrens neurális hálózatok (RNN) a mesterséges neurális hálózatok egy osztálya, ahol a gráf alapú reprezentációba az előrecsatolt hálóval ellentétben lehetnek hurkok. Ez teszi azt lehetővé, hogy ezek a hálózatoknak időbeli dinamikus viselkedése legyen. Az RNN-ek használnak belső állapotot, ami memóriaként segít az idősorok feldolgozását. A memória segítségével tud az idősorokban hosszabb távú összefüggéseket találni. Felépítésükből ebbe a rekurrens osztályba tartozó hálózatok főleg az idősoros adatok feldolgozásában előnyösek, mint például a beszédfelismerés. Ezt a hálózat típust azért mutatom be, mert ezek a neurális differenciálegyenletek klasszikus neurális hálózat megfelelői.



3.4 A standard RNN felépítése [15] [16]

A 3.4-es ábrán látszik egy rekurrens neurális hálózat kiterített formában. A kiterített megjelenítés az értelmezést segíti, a gyakorlatban ez az egyszerű hálózati elem az ábrán látható hálózatnak a harmada. A standard RNN tehát az ábrán látható egy „zöld modulból” áll és az a kimenete, amelyik a másik modulba vezet gyakorlatilag egy visszacsatolás önmagába. Ez valósítja meg a korábban tárgyalt belső állapotot egy fejletlen szinten, ami az időben lévő összefüggések felfedezését teszi lehetővé. Ez a kezdetleges modell is használható a gyakorlatban, de ennél be fogok mutatni egy jobban kezelhető rekurrens felépítést. A standard RNN-nel a probléma főleg az, hogy a bemenetbe az igazán hosszútávú összefüggéseket nem tud felismerni, mert nincs egy erre dedikált csatorna. A másik probléma az, hogy nem feltétlen stabil a tanulási folyamat. A struktúra szenved az eltűnő gradiens és a felrobbanó gradiens problémától. Az előbbi azt jelenti, hogy ahogy terjesztjük vissza a súlymódosítást a kimenettől a bemeneti réteg felé a gradiens nullához közeli lesz, ami miatt a súlyok változatlanok maradnak és a hálózat nem tanul semmit. A felrobbanó gradiens probléma ennek az ellentéte. Ahogy a hálózatba terjesztjük vissza a gradienst az egyre csak nő és a súlyokat annyira elmozdítja, hogy a hálózat használhatatlanná válik. Az említett két problémának a kiküszöbölésére érdemes jó aktivációs függvényt választani. Aktivációs függvényeknél ennél az aspektusnál arra érdemes figyelni, hogy a deriváltja megfelelő intervallumon elég nagy, de ne legyen túl nagy se. Az aktivációs függvény helyes kiválasztásán kívül az háló struktúrájának a változtatása is segíthet itt jönnek képbe az LSTM típusú hálózatok.



3.4 LSTM kapuk [15] [16]

Az LSTM (Long Short Term Memory) a nevéből adódóan az RNN problémájára miszerint nem képes hosszútávú összefüggések megtalálására rögtön választ ad. Az alapvető gondolat ugyan az a kétféle cella között (RNN és LSTM) csak az LSTM egy lépéssel az RNN előtt jár, be is mutatom, miért. Ahogy az RNN az LSTM is minden lépésben bemenetnek kapja az adott lépéshez tartozó bemenetet () valamint az előző lépének a kimenetét és egy belső állapotot reprezentáló vektort. Ezt a vektort használhatja a cella memóriának, amit minden lépésben frissít. A frissítést kapuk segítségével teszi meg. Az első kapu a forget (felejtő) kapu, ami a bemenet és az előző lépés kimenete alapján bizonyos elemeket törölhet a belső állapotból. A második kapu a bemeneti kapu, ami szintén az előző lépés kimenetét és a jelenlegi lépés bemenetét használja. Ez a kapu frissíti a cella belső állapotát. Az utolsó kimeneti kapu a nevéből következően a kimenet előállításáért felelős, amit a cella kimenetére továbbít és a következő lépésbe visszacsatolja.

Az LSTM teljes megértésének érdekében érdemes azt is megvizsgálni, hogy működnek a kapuk. Ezt legjobban az ábrán jelölt aktivációs függvények árulják el. Ahol egy aktivációs függvény nevével vagy jelével ellátott téglalapot látunk az ábrán ott egy olyan előrecsatolt réteg található, aminek a kimenetén a jelölt aktivációs függvény van. Definícióból adódóan, ahol sigmoid aktiváció van ott az előre csatolt háló kimenet vektor elemei nulla és egy közé esnek, ahol a tanh aktiváció van az ábrán ott a kimenet mínusz egy és egy közé esik. Az input kapunál sigmoid aktivációt találunk, aminek a kimenete össze van szorozva elemenként a cella belső állapotával. A kapu funkciója eldönteni, hogy a bemenet alapján mit kell megtartani a belő állapotból és mit nem. Nyilván, ahol a felejtő kapu kimenete egy az megmarad a rejtett állapotba, ahol pedig a kapu kimenet nulla azt eldobjuk. A következő kapu a belső állapot frissítő kapu. Ennek a szerepe az, hogy a cella bemenete alapján frissítse a memóriát. Itt egy tanh aktivációs réteggel a bemenetből képzünk egy vektort, amivel frissítenénk a memóriát. Ezen a vektoron még végrehajtunk egy felejtő kapuval megegyező operációt. Kiválasztjuk a frissítő vektor melyik elemei kerüljenek be ténylegesen a memóriába. A frissítő kapuhoz hasonló elven működik a kimeneti kapu is. Itt a kimenet a belső állapot alapján van számítva, tehát itt található a tanh aktivációval rendelkező előre csatolt réteg. A feature selectiont végrehajtó sigmoid aktiváció pedig a bemenet alapján

## Neurális háló tanítási technikák

Irodalomjegyzék

1. Matıas Nunez, Nadia L. Barreiro, Rafael A. Barrio and Christopher Rackauckas: Forecasting virus outbreaks with social media data via neural ordinary differential equations (2021)
2. Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud: *Neural Ordinary Differential Equations*, (2019. dec.)
3. Delphi Research Group: *Delphi’s Epidata API*, (2020) <https://cmu-delphi.github.io/delphi-epidata/>
4. Fritz Józsefné dr. Kónya Ilona: *Differenciálegyenletek,* (2010, feb)
5. Michael Zeltkevic: *Runge-Kutta Methods* (1998. ápr.)
6. Jonathan Goodman: *ODE, Linear Multistep methods* (2018. tavasz)
7. *Runge–Kutta methods* <https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods>
8. J.C. Butcher and P.B. Johnston: *Estimating local truncation errors for Runge-Kutta methods* (1992. feb.)
9. Douglas Wilhelm Harder: *Topic 14.6: Stiff Differential Equations*, (2005) <https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/14IVPs/stiff/complete.html>
10. Delphi research group: *COVIDcast API client*, (2021) <https://cmu-delphi.github.io/covidcast/covidcast-py/html/signals.html#signals>
11. Rackauckas, Christopher and Ma, Yingbo and Martensen, Julius and Warner, Collin and Zubov, Kirill and Supekar, Rohit and Skinner, Dominic and Ramadhan, Ali: *Universal differential equations for scientific machine learning,* <https://diffeqflux.sciml.ai/dev/>, (2020)
12. Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József: *Neurális hálózatok*, 3. fejezet, 4. fejezet <http://mialmanach.mit.bme.hu/neuralis/ch03s01> (2006)
13. Massimo Buscema: *Back Propagation Neural Networks* <https://www.researchgate.net/publication/13731614_Back_Propagation_Neural_Networks> (1998)
14. Facundo Bre, Juan M. Gimenez: *Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks* <https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051> (2017)
15. Christopher Olah: *Understanding LSTM Networks* <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (2015)
16. Aditi Mittal: Understanding RNN and LSTM <https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e> (2019)