Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Lindmayer Kinga

2025

Szegedi Tudományegyetem

Informatikai Intézet

Mesterséges neuronhálók kiértékelésének vizsgálata a számábrázolás hatásának kimutatására

Készítette:

Lindmayer Kinga

programtervező informatikus hallgató

Témavezető:

Dr. Csendes Tibor

egyetemi tanár

Szeged

2025

Feladatkiírás

A hallgató feladata, hogy egy helyesen betanított neurális hálózat kiértékelésében vizsgálja a számítógépes környezetben keletkező kerekítési hibák előfordulását. Mutassa be, hogy ezek az eltérések milyen módon keletkeznek és mekkora különbséget okozhatnak. Ennek teszteléséhez használja az MNIST adatbázist és a megvalósítása Python nyelven történjen.

Tartalmi összefoglaló

- **A téma megnevezése:**

  A dolgozatom témája a neuronhálók kiértékelésének vizsgálata számítógépes környezetben, ami első sorban az egyes neuronokban történő műveletek eredményének vizsgálatára fókuszál és arra, hogy ezek milyen hatással vannak a neuronháló kimenetére.

- **A megadott feladat megfogalmazása:**

  A feladatom az volt, hogy bemutassam számítógépes környezetben mit jelent a kerekítési hiba és ez egy neuronhálózat vonatkozásában, hogyan állhat elő. Ehhez vizsgálnom kellett azt, hogy a neuronok kiértékelése során milyen módon lehet hatással az összeadások sorrendje annak kimenetére és azt, hogy ez egy neuronháló kiértékelésére és tanulására ez milyen hatással van. Illetve ezeken felül feladatom volt, hogy ezt az eltérést helyesen betanított neuronháló esetében kimutassam.

- **A megoldási mód:**

  A megoldáshoz, először egy kézi számolási feladatban tanulmányoztam, hogy milyen módon jelentkezhetnek a kerekítési hibák és ezt részletesen szemléltettem azt egy szimulált neuronháló segítségével.

  Megvizsgáltam, hogy két különböző futásban, de azonos kezdeti modellnek a tanított súlyaiban mekkora eltérések jelentkezik. Az elért eredmények alapján részletesebben megvizsgáltam, hogy egy neuron kimenetének vonatkozásában, hogyan lehet előidézni az eltéréseket, és az után azt is, hogy egy valós neurálishálózatban a tanulási folyamatok során ez, hogyan viselkedik.

- **Alkalmazott eszközök, módszerek:**

  A kézi számolási feladathoz Python nyelven írt kóddal, segítettem a munkámat, hogy a kerekítési hibákat minél pontosabban tudjam bemutatni. Annak prezentálására, hogy valós tanulási folyamatok végeredményeként milyen eltérések keletkeznek azt a Google-nak egy nyílt forráskódú könyvtárát használva mutattam be. A Tensorflow könyvtár a tanítást megbízhatóan hajtja végre és teszt adatokon nyújtott teljesítménye több mint 90%-os eredményt hoz. Itt az MNIST adatbázist használtam, ami lehetőséget adott a neuronhálók teljesítményének vizsgálatára is.

A neuronok és neuronhálók részletesebb megfigyeléséhez szintén Python-t használtam, ahol saját implementációkat alkalmaztam, az itt elért kimutatásokat is diagrammokon keresztül demonstráltam.

- **Elért eredmények:**

  Bemutattam a dolgozatom során, hogy a kerekítési hibák valós eltéréseket okoznak a tanított neurális hálózatokban. A teszt adatokon is jól teljesítő tanított modelleken, azonos bemeneti paraméterekkel és egyező tanuló adatokkal történt a tréning, mégis a végeredményként kapott kimenetek eltértek egymástól.

  Ezt igazolva jobban a mélyére ástam annak, hogy egy neuron kiértékelése során milyen valós eltérések keletkezhetnek a számítógépes környezetben. Ehhez írtam, egy saját kódot, ami segítségével teljesen véletlen szerű permutációkat sikerült megalkotnom és így olyan értékekkel, amik egy modell paraméterei lehetnek sikerült eltéréseket generálnom. Ezt a gondolatot tovább vittem és a tanulás folyamatban valóra előrehaladás során is vizsgáltam, hogy ezek a különbségek milyen irányba változnak. Sikerült grafikonokkal is szemléltetnem, hogy a keletkező eltérések hatással vannak egymásra és iterációkon keresztül ezek értéke növekszik.

- **Kulcsszavak:**

  Python, gépi tanulás, ellenséges példák, numerikus instabilitás

Tartalom

1. Bevezetés

A mesterséges intelligencia és gépi tanulás a mai világ számos területén jelen van és a hétköznapi életünk szerves részévé vált. Alkalmazzák az orvosi diagnosztikában, a fordító szoftverekben, pénzügyi előrejelzésekben és az önvezető autók világában is. Ezért is ez egy kifejezeten gyorsan fejlődő iparág, ahol elengedhetetlen az, hogy megbízható és pontos kiértékelések történjenek. Fontos ez azért is mert nem csak szerteágazó a felhasználása, hanem mert egyre nagyobb döntéseket bízunk algoritmusokra.

A gépi tanulás egyik iránya a neurális hálózatok, amelyek sok rétű problémák megoldására is alkalmas. Népszerűségét az adja, hogy rugalmasak és jól alkalmazkodnak különböző típusú problémák megoldására. Az általam felsorolt területek között is vannak képi és nyelvi problémák, amikre ez is kiválóan alkalmazható. De amellett, hogy több területen is helyt áll előnyei közé tartozik, hogy taníthatósága viszonylag egyszerű, jól skálázható és nagyobb adatmennyiségeken is jól teljesít.

Minden jó tulajdonsága mellett a neurális hálózatoknak is vannak hiányosságai, amik támadási felületet adnak ezért is ezek feltárása a mesterséges intelligencia kutatási területének kiemelt fókuszában állnak.

1.1. A dolgozat célja

A dolgozat célja, hogy kimutassa a kerekítési hibák okozta eltéréseket és azok hatásait egy helyesen betanított neurális hálózatra. A számítógépes környezetben végzet műveletek során előfordulhatnak olyan apró eltérések a lebegő pontos számábrázolás miatt, amelyek hatással lehetnek a tanulási folyamat eredményére. A feladatom, hogy megvizsgáljam ezek az eltérések milyen módon befolyásolják a hálózat viselkedését még akkor is, ha a tanítás hibátlanul történt meg. A jelenség bemutatásához egy egyszerű neurális hálózatot és az MNIST adatbázist fogom használni.

A probléma részletes feltárása miatt több oldalról is megközelítem a kérdést és részletesen bemutatom, hogyan jelennek meg ezek az eltérések a tanulás során és végül a már betanított neuronháló esetében. Lehet, hogy a bemutatott különbségek elsőre elhanyagolhatónak tűnnek, de a hálózat összetettségé miatt ezek az eltérések hatással lehetnek a modell végső teljesítményére.

1.2. A dolgozat felépítése

A dolgozatom elején röviden leírom azokat az általános információkat, amik segítik a dolgozat értelmezését. Itt általánosságban beszélek a gépi tanulásról neurális hálózatkról és az ellenséges példáról.

Ezután egy egyszerűbb számítási feladat során prezentálom, hogyan keletkezhetnek a kerekítési hibák egy neurális hálózatban, ehhez egy kézzel megalkotott neuronhálónak egy tanulási fázisát fogom bemutatni olyan értékekkel, amik indukálják a kerekítési hibákat.

Miután általánosságban bemutattam a jelenséget egy valós példán keresztül is bemutatom, hogy egy helyesen tanító modell szerint is előállnak a kerekítési hibák, így ez a tézis valós alapokon nyugszik. Ennek bemutatására egy nyílt forráskódú tanító algoritmust használtam, amik a tesztelési fázisban is megfelelően teljesítettek. Az ezek alapján levont következtetések mentén megvizsgálom, hogy pontosan milyen műveletek hatására állhatnak elő ezek az eltérések és milyen módon generálhatóak. Ehhez műveleti szinten veszem szemügyre mi történik egy neuron kiértékelése során és egy valós neuron segítségével igyekszem az összeadás asszociativitását vizsgálni és aztán ebből kifejlődve a neurális hálózatnak a tanulása során is végig kísérem a történéseket. Végezetül pedig összefoglalom, hogy milyen eredményeket tudtam elérni és hogy milyen új kérdéseket vetnek fel azok.

2. Elméleti áttekintés

2.1. A neurális hálózatok

A mesterséges intelligencia több területből áll össze, ennek egyik része a gépi tanulás melynek célja, hogy valós időben tanuljon és javítson a saját kiértékelő képességén. Ehhez arra van szükségünk, hogy megfelelő tanító adatokkal és teszt halmazzal is rendelkezzünk, hogy a tréning sikerességéről biztosan meggyőződhessünk. Azért is fontos, hogy ezt a két csoportot elkülönítsük mert a célunk nem az adatok pontos megtanítása lesz, hanem hogy minél jobban tudjon általánosítani a modellünk.

A gépi tanulás alkalmas arra, hogy csoportosítási feladatokat oldjon meg. A dolgozatom is egy ilyen problémán keresztül vizsgálja majd a neurális hálókat. Ehhez az MNIST adatbázist fogom tanulási alapul venni, amiben kézzel írt számjegyek vannak. A megfelelően tanult modellek képesek arra, hogy egy tetszőleges bemenetet megfelelőcsoportba sorolja be. Ennek

az adatbázisnak az előnye, hogy a tanuló adat előre felcímkézett, a teszteléshez van egy külön elhatárolt halmaz és emellett ingyenesen hozzáférhető.

2.1.1. Neuron

A neuron a neuronhálózatnak az alapegysége, ami önálló kiértékeléseket végez. Alapjául az emberi neuronok működése szolgált, ami több inger hatására egy jelet továbbít. Ennek megfelelően bemenete a jellemzők és az azokhoz tartozó súly értékek, illetve minden neuronhoz tartozik egy bias érték is ezek alkotják az ingert. Egy neuron kiértékelése veszi ezeknek a szorzatát összegzi, hozzáadja a bias értéket és ennek eredménye a neuron kimenete a jel. Ezekután egy aktivációs függvény határozza meg, hogy mi lesz a végleges kimenete. Ennek a függvénynek a szerepe az, hogy a linearitását megszakítsa a modellnek és képessé tegye arra, hogy összetettebb összefüggéseket is létrehozzon. Ebből adódóan ez egy nem lineáris és differenciálható függvény kell legyen. Jelenleg legnépszerűbb és legelterjedtebb a ReLu és azért, mert igazolhatóan gyorsabban konvergál a megoldás felé, mint más nem lineáris függvények teszik. A bias érték alapvetően egy eltolási érték, ami az aktivációs függvény kimenetét befolyásolja.

Egy neuron két osztály megkülönböztetésére alkalmas, gyakorlatban ez azt jelenti, hogy képes eldönteni, hogy az adott input az osztályhoz tartozik vagy sem. A tanulás ebben az esetben is iteratív módon történik, a kapott érték, ha megfelel a várt értéknek nem kell módosítást végrehajtani viszont, ha hibás eredményt adott akkor abban az esetben tanítjuk a súlyokat. Ezt gradiens módszerrel is meglehet tenni. Lényeg, hogy egy olyan költség függvényt válasszunk, ami megfelelően felméri a hálózat hibáját és emellett deriválható is így vizsgálható milyen irányba kell módosítani a súlyokat. A következő tanítási ciklusban így már egy optimalizáláson átesett súly értékekkel számolhatunk.

A neuron hátránya, hogy önmagában csak lineáris szét választásra alkalmas viszont, ha az lehetséges akkor azt véges lépésben eléri.

2.1.2. Neurális háló

A neuron alacsony reprezentációs képességnek feloldására született meg a neurális hálózat, amely magában hordozza a neuron erősségét azonban komplexebb problémák megoldására is alkalmas. Ebben az esetben a neuronok hálózatba rendeződnek és kapcsolódnak egymáshoz. A bemeneti réteg az, ami a jellemzőket feldolgozza és ezután a következő rétegek bemenete az előző réteg kimenete lesz. Ennek köszönhető az, hogy nagyobb bonyolultságú problémák

megoldására is alkalmas, mert a külön álló neuronok csoportosító képessége összeadódik. Egy modell minél mélyebb architektúrával rendelkezik a teljesítő képessége azzal arányosan növekszik a komplexebb problémák megoldásának tekintetében.

A tanulás során minden súly érték optimalizáción megy keresztül, hogy a hálózat teljesítménye javuljon. Egy ilyen hálózat már képes lineárisan nem szétválasztható csoportok szeparációjára. Így az általunk vizsgált MNIST adatbázisnak a tanítására is. A probléma megoldására alapvetően már egy rejtett réteggel rendelkező neuronháló is képes, azonban dolgozatom során ezt két rétegűvel fogom megtenni. Ennek oka, hogy olyan modelleket esetében vizsgáljam a felvetést, ami összetettebb problémák megoldása során is használatos.

A neurális hálózatok hátrányai közé tartozik, hogy nagy mennyiségű tanuló adatra van szüksége ahhoz, hogy megfelelő legyen a tanítás és elkerülhető legyen, hogy az adatok túltanulása történjen meg.

2.1.3. Ellenséges példák

Az ellenséges példák jelensége a neurális hálózatok hiányosságára hívják fel a figyelmet. Ebben az esetben ugyanis a bemeneti érték valamilyen manipulációján keresztül a már betanított hálózat hibás osztályozást fog tenni ráadásul ezt úgy, hogy nagy pontossági valószínűséget társít mellé. Erre egy példa amikor a rendszám felismerő szoftver hibásan ismeri fel a karaktereket annak ellenére, hogy emberi szemmel egyértelműen azonosíthatóak. Ebben az esetben a támadók valamilyen zajt helyeznek el, ami miatt téves osztály besorolás fog történni.

A gépi tanulásnak ez egy sokat kutatott területe, mert ez a hiányosság rámutat arra, hogy a tanulás során a statisztikai mintázatok alapján tanul és azoktól nem tud elvonatkoztatni.

Ez egy komoly biztonsági kockázat mert bizonyos szoftverek tekintetében nem elfogadhatóak ekkora mértékű hibák. A kutatások elsősorban olyan modellek megalkotására fókuszál, amik ellenálló képessége nagyobb. Ennek egy iránya a robosztus tanítás, amelynek célja, hogy a modellek ne csak a tanító adathalmazra legyenek érzékenyek, hanem a szándékos torzításokra is megfelelően reagáljanak.

2.2. Számítógépes számábrázolás

A számítógépes számábrázolás korlátja, hogy véges nagyságban tudjuk a számokat tárolni. Azért, hogy minél nagyobb intervallum lefedése lehetséges legyen jött létre a lebegő pontos szám ábrázolás, aminek lényege, hogy a használt számrendszernek a hatványával szorzunk és így kevesebb biten tudjuk tárolni a számokat. Ennek egy szabványa az IEEE 754, ami

megszabja, hogy 32 bites ábrázolás esetén 1 bit teszi ki az előjelet, 8 bit a kitevőt és 23 bit a mantisszát a szám lényegi részét. Dupla lebegő pontos számábrázolás esetében pedig ennek kétszeres tárterülete áll rendelkezésre, amit a kitevő és a mantissza között oszt szét az úgy, hogy az utóbbi 52 biten lesz ábrázolva. A lebegő pontos számok esetében, minden művelet elvégzése előtt közös kitevőre kell hozni a számokat és a művelet elvégzése után normalizálni kell az eredményt, ami a mantissza egész részének egy jegyűre alakítását jelenti. Ez az eljárás magában hordozza, hogy kerekítési hiba történjen, mert ezzel értékes számjegyeket veszíthetünk.
A lebegő pontos számábrázolás kerekítési hibájának bemutatására egy ismertebb példa:

$$10^{23} + 2025 - 10^{23}$$

ahol eredményként a matematika szabályai szerint 2025-öt várnánk azonban amennyiben ebben a sorrendben történik a kiértékelés lebegőpontos számok használatával 0-át fogunk kapni. Ennek oka, hogy eltérő nagyságrendű számokat adunk össze és vonunk ki és ebben az esetben ez kerekítési hibát okoz.

3. Probléma szemléltetése

3.1. Számítási példa

Ebben a fejezetben egy kisebb neurális modell egy tanítási iterációjának mentém fogom prezentálni, hogy milyen eltéréseket okozhatnak a kerekítési hibák. Alapvetően maga a neurális hálózat mérete és kezdő paramétereinek megválasztásánál az volt a szempont, hogy minél látványosabban és minél érthetőbb példát tudjak adni. Ez megnyilvánul abban, hogy a kezdeti súlyok nagyságrendileg sokszínűek míg a neurális háló mérete kicsi.

3.1.1. A számítás során használt jelölések és képletek

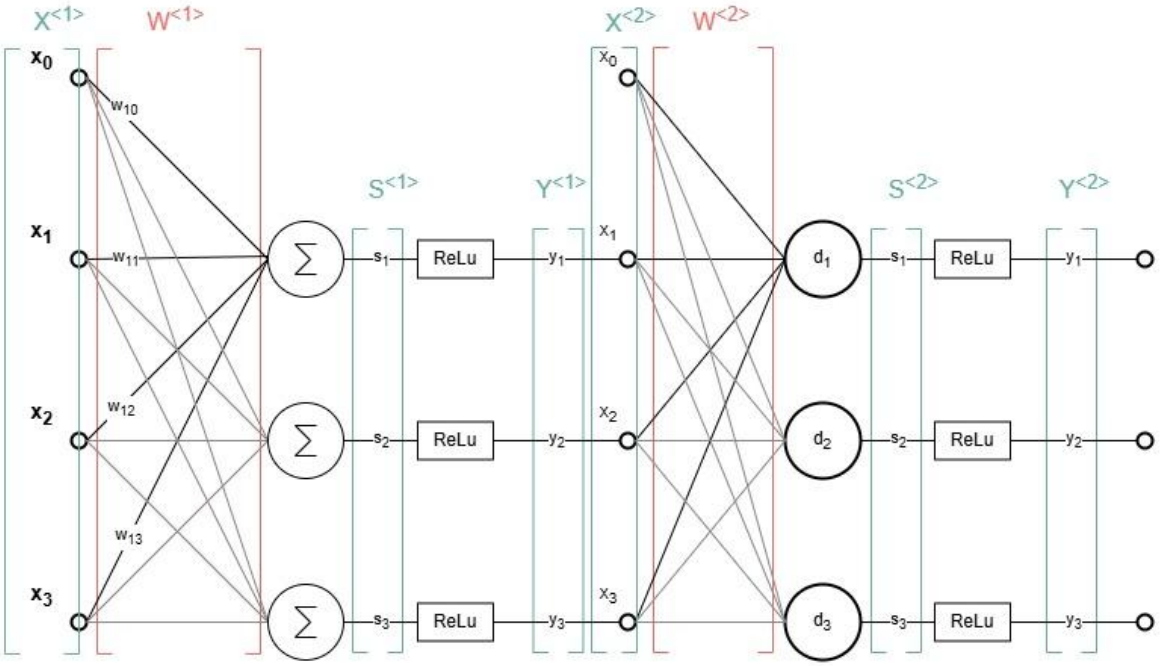
Az 3.1. ábrán látható jelöléseket fogom használni a későbbi számításaim során, a képen egy két rétegből álló neuronhálózat látható, amely három input értékkel rendelkezik és a be-, illetve kimeneti rétegen is három-három kiértékelő neuront tartalmaz. Egy neuron kiértékelése az alábbi képlet alapján történik:

$$s_i = \sum_{n=1}^{n<n} w_{in} * x_n + b \qquad (3.1)$$

ahol s_i egy neuronnak a jel értéke w_{in} egy x_i inputhoz tartozó súly érték és b a neuronhoz tartozó bias érték. Amennyiben erre a summa értékre alkalmazom az aktivációs függvényt, ami jelen esetben a ReLu:

$$\varphi(s_i) = \begin{cases} 0 \text{ ha } s_i \leq 0 \\ s_i \text{ ha } s_i > 0 \end{cases} \tag{3.2.}$$

akkor megkapom a neuron kimenetét y_i-t, illetve, ha ez a bemeneti rétegnek egy neuronja akkor a kimeneti réteg egy x_i inputját eredményezi.



3.1. ábra: Neuronháló a számítási feladathoz

Az egy réteghez tartozó azonos típusú értékeket lehet vektorként, valamint a súlyok esetében mátrixként is jelölni. Az ábrán ezeket én az azonos nagybetűkkel jelöltem felső indexben elhelyezve a rétegnek a sorszámát. A számolás során a bias értéket szokás úgy kezelni, hogy a neuron súly vektornak nulladik eleméhez tesszük és ezzel párhuzamosan az X vektorba is egy konstans egyet helyezzünk el. Ennek gyakorlati haszna, hogy a tanítás során nem kell külön a biasra is végrehajtani a számítást. Az ábrán használt jelöléseknél az X vektor már tartalmazza is az x_0-t. A kimeneti réteg neuronjait d_i-vel jelöltem.

A számításaim során azt fogom vizsgálni, hogy az összeadások sorrendje milyen mértékben befolyásolja a végeredményt. Ahhoz, hogy jegyzeteim átláthatóak maradjanak bevezettem egy saját jelölést, aminek a segítségével a különböző permutációkat tudtam jelölni.

$$a_i = x_1 * w_{i1}, \qquad b_i = x_2 * w_{i2}, \qquad c_i = x_3 * w_{i3}, \; y_i = x_0 * w_{i0}$$

A tanulási fázisban a megválasztott költség függvény az átlagos négyzetes eltérés és annak egy bemeneti y_i értékére vett parciális deriváltja:

$$E = \frac{1}{2}\sum_{i=1}^{N}(d_i - y_i)^2 \,;\, \frac{\partial E}{\partial y_i} = d_i - y_i \tag{3.3.}$$

A gradiens módszerből levezetve egy w_{in} súlynak az optimalizálására használt képlet a következő:

$$w_{in} = w_{in}^{régi} + \mu * x_n * \frac{\partial E}{\partial y_i} \tag{3.4.}$$

Az első rétegre a hiba visszavezetésével tudjuk meghatározni hiba mértékét, ami alapján a tanítás elvégezhető ehhez az alábbi képletet alkalmaztam:

$$\delta^{<1>} = \sum_{n=1}^{N^{<2>}} (\delta^{<2>} f'\left(s_n^{<2>} w_{in}^{<2>}\right) \tag{3.5.}$$

3.1.2. A számítás lépésről lépésre

A neuronhálózat bemenetei és az elvárt kimenete a fent meghatározott jelölések szerint a következőek:

x_0	x_1	x_2	x_3	d_1	d_2	d_3
1	10,2	1	0,75	1	0	0

$$W^{<1>} = \begin{bmatrix} 3 * 10^5 & 1 * 10^8 & 5{,}3 * 10^{-5} \\ -3{,}01 * 10^6 & 1 * 10^{-10} & 1{,}19841 * 10^5 \\ 8{,}6 * 10^{-5} & -1 * 10^8 & 5 * 10^5 \\ 7{,}7 * 10^{-4} & 1{,}86 * 10^{-7} & -5{,}3 * 10^{-4} \end{bmatrix}$$

$$W^{<2>} = \begin{bmatrix} 3 * 10^5 & 1 * 10^8 & -5{,}3 * 10^{-5} \\ -3{,}01 * 10^5 & 1 * 10^{-10} & -1{,}19841 * 10^5 \\ 8{,}6 * 10^{-5} & -1 * 10^8 & -\ 5 * 10^5 \\ 7{,}7 * 10^{-4} & 1{,}86 * 10^{-7} & 5{,}3 * 10^{-4} \end{bmatrix}$$

A számítás első lépéseként az első rétegen kell végre hajtani a neuronok kiértékelését. Ehhez vesszük először a $W^{<1>}$ mátrix első oszlopát, ami az első neuronhoz tartozó súlyokat tartalmazza és összeszorozzuk az $X^{<1>}$ vektorral. Az így kapott számok összegét fogom vizsgálni abban a tekintetben, hogy az összeadás sorrendje befolyásolja-e annak a kimenetét számítógépes környezetben. Ehhez veszem a négy szám összes permutációját, ami összesen huszonnégy variációt jelent. Azokon végig iterálva és elvégezve az összeadásokat keresem a minimum és maximum értékeket. Ez az első neuron tekintetében így alakulnak:

$$\text{min: } s_1 = a_1 + \left((y_1 + b_1) + c_1\right) = -2770199{,}9993365$$

$$\text{max: } s_1 = \left((a_1 + y_1) + b_1\right) + c_1 = -2770199{,}9993364997$$

Ezek után sorra a következő neuronokat hasonló módon vizsgálva az alábbi szélső értékeket kapjuk:

$$\text{min: } s_2 = ((a_2 + y_2) + c_2) + b_2 = 1,\mathbf{341104507446289} * 10^{-7}$$

$$\text{max: } s_2 = ((y_2 + b_2) + a_2) + c_2 = 1,\mathbf{40520000263387043} * 10^{-7}$$

$$\text{min: } s_3 = \big((a_3 + b_3) + c_3\big) + y_3 = 1722378{,}1246554\mathbf{998}$$

$$\text{max: } s_3 = \big((y_3 + a_3) + b_3\big) + c_3 = 1722378{,}124655\mathbf{5}$$

Ezeket az eltérésekre tekinthetünk úgy, hogy két $S^{<1>}$ vektort eredményeznek úgy rendezve, hogy az egyikben a minimum, a másikba pedig a maximum jel értékeket vesszük. Az eltérés a kettő között:

$$S_{max}^{<1>} - S_{min}^{<1>} = [\,4{,}65661287 * 10^{-10} \;\; 6{,}40955189 * 10^{-9} \;\; 2.32830644 * 10^{-10}]$$

Ezért is a második réteg neuronjainak kiértékelését mindkét jel vektor alapján elfogom végezni és így azon a rétegen már azok eltérését fogom vizsgálni. Ehhez előtte persze az aktivációs függvényt is végrehajtom és az így kapott vektort kiegészítem a biashoz tartozó egyes értékkel. A két jel vektorral számolva a második réteg inputjai a következőek:

$$X_{min}^{<2>} = [1\ 0\ 1, \mathbf{341104507446289} * 10^{-7}\ 1722378{,}1246554\mathbf{998}]$$

$$X_{max}^{<2>} = [1\ 0\ 1, \mathbf{40520000263387043} * 10^{-7}\ 1722378{,}124655\mathbf{5}]$$

Ezen a rétegen is a korábbival megegyező algoritmust használtam, viszont ezekben az esetekben nem volt eltérés ezért az egyetlen kimenet a következő - a két különböző bemenet függvényében:

$$S_{min}^{<2>} = [3{,}01326231 * 10^5\ 9{,}9999986\mathbf{9} * 10^7\ 9{,}1279\mathbf{3304} * 10^2\ ]$$

$$S_{max}^{<2>} = [3{,}01326231 * 10^5\ 9{,}9999986\mathbf{3} * 10^7\ 9{,}1279\mathbf{0100} * 10^2\ ]$$

Ha összevetjük a második réteg neuronjainak minimum és maximum kimenetét látható eltérést kapunk:

$$S_{max}^{<2>} - S_{min}^{<2>} = [0\ -0{,}64095497\ -0.00320478\ ]$$

Ezzel meg is kaptuk a neuronhálózat kimenetét és ezeket megvizsgálva látható, hogy az elvárt kimenettől mennyire térnek el és az alapján, ha szükséges elvégezzük a tanítási lépéseket.

Mivel mindkét kimenet esetében eltérés tapasztalható az elvárt értékhez mérten ezért a tanítási iteráció első lépéseként kiszámítottam a hibákat a kimeneti rétegeken.

$$\delta_{min}^{<2>} = [-3{,}0132522 * 10^5\ -9{,}9999984 * 10^7 - 9{,}1279\mathbf{327} * 10^2\ ]$$

$$\delta_{max}^{<2>} = [-3{,}0132522 * 10^5\ -9{,}9999984 * 10^7 - 9{,}1279\mathbf{010} * 10^2\ ]$$

Ezek alapján a kimeneti réteghez tartozó súlyoknak a módosított értékeket tartalmazó mátrixa az alábbi módon néz ki.

$$W_{min}^{<2>} = \begin{bmatrix} 2{,}6986747 * 10^{5} & 9{,}0 * 10^{7} & -9{,}1279\mathbf{381} * 10 \\ 8{,}3472810 * 10^{10} & 2{,}7701996 * 10^{13} & 2.5274\mathbf{216} * 10^{8} \\ -\mathbf{3{,}9550862} * 10^{-3} & -1{,}0 * 10^{8} & -5{,}0 * 10^{5} \\ -5{,}1899597 * 10^{10} & -1{,}7223778 * 10^{13} & -1.5721\mathbf{752} * 10^{8} \end{bmatrix}$$

$$W_{max}^{<2>} = \begin{bmatrix} 2{,}6986747 * 10^{5} & 9{,}0 * 10^{7} & -9{,}1279\mathbf{068} * 10 \\ 8{,}3472810 * 10^{10} & 2{,}7701996 * 10^{13} & 2{,}5274\mathbf{126}e * 10^{8} \\ -\mathbf{4{,}1482220} * 10^{-3} & -1{,}0 * 10^{8} & -5{,}0 * 10^{5} \\ -5{,}1899597 * 10^{10} & -1{,}7223778 * 10^{13} & -1{,}5721\mathbf{698} * 10^{8} \end{bmatrix}$$

Ezek után az első rétegre való hiba visszafejtés a minimum és maximum hiba vektorok alapján:

$$\delta_{min}^{<1>} = [1{,}093\mathbf{90056} * 10^{8} \;\; 9{,}9999981\mathbf{3} * 10^{15} \;\; -1{,}90837\mathbf{65} * 10\,]$$

$$\delta_{max}^{<1>} = [\,1{,}093\mathbf{8968} * 10^{8} \;\; 9{,}9999981 * 10^{15} \;\; -1{,}9083775 * 10\,]$$

És az ezalapján a tanított súlyok a következő képen alakultak:

$$W_{min}^{<1>} = \begin{bmatrix} 1{,}123\mathbf{90060} * 10^{7} & 9{,}99999920 * 10^{14} & -1{,}90832\mathbf{460} \\ 1{,}11276\mathbf{85}6 * 10^{8} & 1{,}01999\mathbf{76} * 10^{16} & 1{,}19821\mathbf{53}1 * 10^{5} \\ 1{,}093\mathbf{90060} * 10^{7} & 9{,}99999\mathbf{786} * 10^{14} & 4{,}9999809\mathbf{4} * 10^{5} \\ 8{,}2042\mathbf{5450} * 10^{6} & 7.4999987\mathbf{3} * 10^{14} & -1{,}43181\mathbf{324} \end{bmatrix}$$

$$W_{max}^{<1>} = \begin{bmatrix} 1{,}123\mathbf{8968} * 10^{7} & 9{,}9999992 * 10^{14} & -1{,}9083\mathbf{245} \\ 1{,}1127\mathbf{647} * 10^{8} & 1{,}0199998 * 10^{16} & 1{,}1982153 * 10^{5} \\ 1{,}093\mathbf{8968} * 10^{7} & 9{,}9999979 * 10^{14} & 4{,}9999809 * 10^{5} \\ 8{,}2042\mathbf{255} * 10^{6} & 7.4999987 * 10^{14} & -1{,}4318131 \end{bmatrix}$$

3.1.3. A kapott eredmény értelmezése

A számolás során végeredményként egészen nagy eltérést kaptam a tanított súlyok kiszámításánál. Látható, hogy míg az első réteg neuron kiértékelésénél kisebb mértékű eltérések voltak tapasztalhatóak a második rétegben már nagyságrendekkel nagyobb különbség keletkezett.

Az összeadásoknál jól látható, hogy az eltérő nagyságrendű számok különböző sorrendű összeadása eltéréseket okozott. Ennek oka, hogy a számítás során közös kitevőre hozzuk a törteket elvégezzük az összeadást ezután újra normalizáljuk az alakját. A számábrázolás korlátossága miatt ez eredményezi azt, hogy eltérést tapasztalhatunk a számok összeadási sorrendje alapján.

Ez az első rétegnek a második neuronjához tartozó minimum összeg az alábbi sorrendet jelenti:

$$((1{,}02 * 10^{-9} + 1 * 10^{8}) + \ 1.395 * \ 10^{-7}) + (-1 * 10^{8})$$

Ez, ha számítógépes környezetben dupla lebegő pontos számokkal hajtjuk végre az összeadásokat külön-külön az alábbi rész eredményeket kapjuk:

$$1{,}02 * 10^{-9} \ + 1 * 10^{8} \ = \ 0{,}00000000000000000102 * 10^{8} \ + 1 * 10^{8}$$
$$= \ 1{,}00000000000000000102 * 10^{8} \ = \ 1.000000000000000 * \ 10^{8}$$

Látható, hogy először közös kitevőre hozzuk a két számot, ami mindig a nagyobb hatványhoz történik, ez ebben esetben nyolc. Ezután elvégezzük az összeadás műveletét és végrehajtjuk a normalizálást. Ebben az utolsó lépésben látható az, hogy elvesznek a tizedes jegyek a számábrázolás korlátossága miatt, így a végeredmény az megegyezik a nagyságrendekkel nagyobb bemeneti értékkel. Ez figyelhető meg a következő összegben is, annyi eltéréssel, hogy mivel nagyobb számot adtunk hozzá ehhez a számhoz ezért csak részben vesznek el a tizedesjegyek.

$$1.0 * \ 10^{8} + \ 1.395 * \ 10^{-7} = \ 1.0 * \ 10^{8} + \ 0{,}000000000000001395 * \ 10^{8}$$
$$= \ 1{,}000000000000001395 * \ 10^{8} = \ 1{,}0000000000000013 * \ 10^{8}$$
$$1{,}0000000000000013 * \ 10^{8} \ - 1 * \ 10^{8}$$
$$= 0{,}0000000000000013 * \ 10^{8} \sim 1.341104507446289 \ * \ 10^{-7}$$

Az utolsó összeg esetében látható, hogy normál alakra hozza a végeredményt, ami egy nagyon kicsi szám és a korábbi pontatlan ábrázolás miatt itt is eltérések jönnek elő. Ha ezt az összeget nem lebegő pontosan adjuk össze, hanem papíron és pontosan akkor a végeredmény: 0,00000014052.

A következő rétegnek a kimeneténél már az eltérés nagyobb mértékben jelentkezett mert a nagy abszolút értékű számokkal szoroztunk, ami hasonlóan eltéréseket okozhat.

Végeredményként elmondható, hogy ebben a példában látványos eltéréseket okozott a számok különböző sorrendben történő összeadása, azonban abból eredően, hogy ez a példa kiélezetten olyan adatokkal lett feltöltve, amik ezt látványosan bemutatták további vizsgálatok szükségesek, hogy valós adatokon ez milyen mértékben okoz eltérést.

3.2. Python TensorFlow könyvtár

A Google által fejlesztett nyílt forráskódú gépi tanulási könyvtárt elsősorban mély tanulásra alkalmazható. Támogatja a többrétegű neuronhálók építését és tanítását. A példámban azt

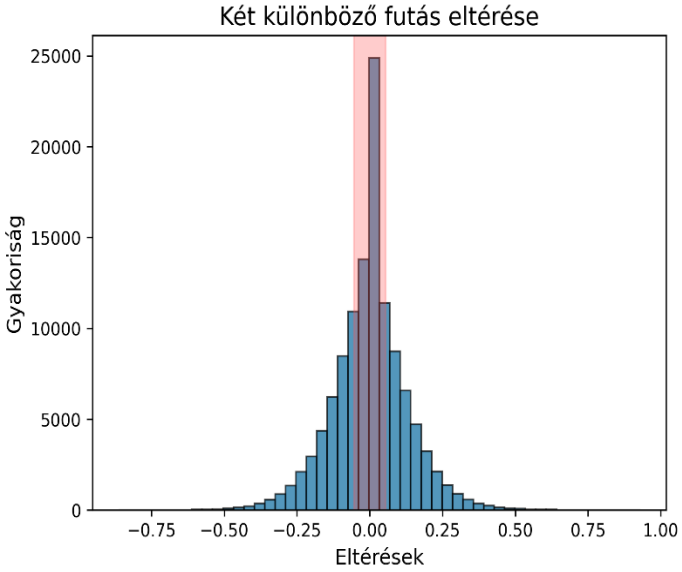
vizsgáltam, hogy az MNIST adatbázis tanulása során két különböző futás során a tanult neuronhálóban mekkora eltérések keletkeznek.

Ehhez arra volt szükség, hogy biztosítsam azt, hogy pontosan ugyanazon paramétereket adjam meg a tanításhoz. Ezt úgy tettem meg, hogy létrehoztam egy több rétegű modellt, ami két rejtett réteggel rendelkezik és ennek a kezdeti súlyait lementettem. Mivel a tanuló adathalmaz pontosan megegyezik, illetve az algoritmus során sorrendben történik azok feldolgozási ezért adott volt, hogy ha bármilyen eltérés tapasztalható akkor az már a számítások során keletkező hiba eredményeként jöhet létre.

Ezért tehát azt vizsgáltam, hogy ha egymás után kétszer lefuttattam ugyanazokkal a kezdeti súlyokkal akkor a már tanított modellben mekkora eltérések tapasztalhatóak. Ezekből az eltérésekből készítettem egy hisztogramot, ami egy gyakorisági függvény és megmutatja, hogy az adatok, hogyan oszlanak el a megadott érték tartományban.

A 3.2. ábrán jól látható, hogy a gyakoriság eloszlását szemléltető függvényen az értékek abszolút értéke nullához közeli értéket vesz fel, egész konkrétan az adatok elemzésekor is az volt látható – többszöri futások vizsgálatánál is -, hogy az értékek fele az a $\pm 0{,}055$ intervallumon helyezkedik el, viszont a szélső értékek abszolút értéke nagyságrendileg az egyet közelíti.

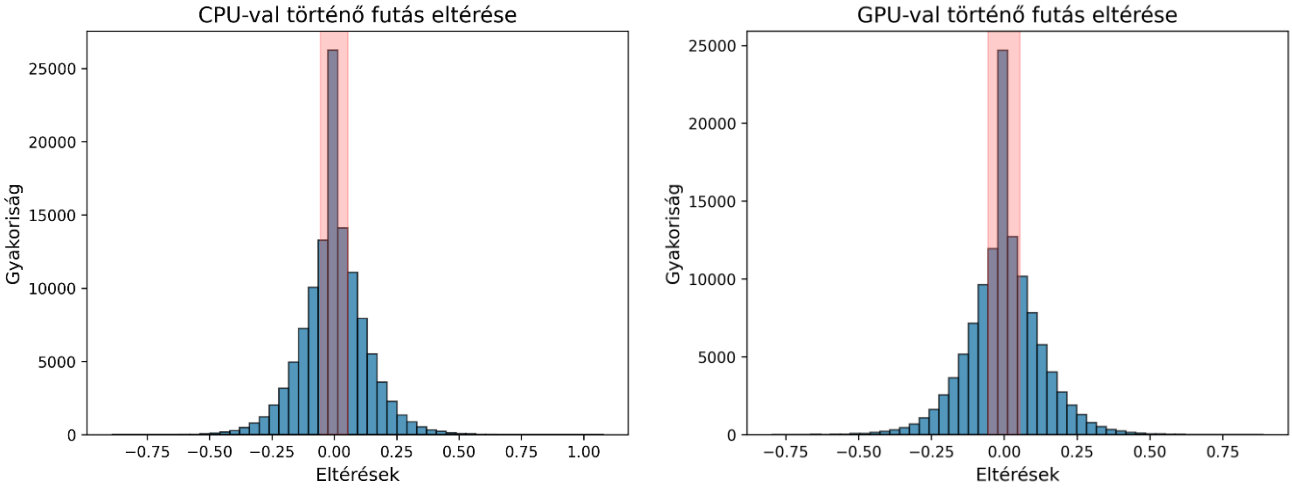
Ezek az eltérések azért keletkezhetnek, mert mikor egy neuronhálózat tanítása zajlik, az többnyire több szálon történik és az hogy a számítási folyamatok szétosztása történik az véletlenszerűen történik. Ez okozza azt, hogy a számok nem pontosan ugyanolyan sorrendben kerülnek összeadásra, ami a kerekítési hibák miatt eltérő eredményeket adhat.



3.2. ábra Két különböző futás eredményéből kapott eltérés

Így tehát vizsgáltam azt is, hogy ha módosítom azt, hogy melyik processzorok vesznek részt a számításokban akkor esetleg eltérő nagyságrendű eltérések, kilengések lesznek tapasztalhatóak. A koncepció e-mögött az, hogy ezzel valamilyen módon kívülről hatás lehet gyakorolni arra, hogy hogyan történjen a számítási feladatok szétosztása a processzusok között mivel bizonyos kapacitásokat kizárok a kiértékelésből.

Ehhez ugyanazokkal a kezdeti paraméterekkel hajtottam végre a számításokat, azonban egyik esetben a GPU-t (Graphics Processing Unit) vontam ki a számítási folyamatokból a másik esetben pedig a CPU-t (Central Processing Unit) a viszonyítás pedig az előző diagramnál is használt egyik kiértékelés volt. A két processzor eltér egymásban attól, hogy míg a CPU az általános számítások elvégzésére használatos és lassabb addig a GPU nagyobb igényű számításoknál akár párhuzamos szálon történő kalkulációkra is képes és emellett nagyobb sebességre is. Azonban mindezek ellenére az ezekből készült grafikonokon - ami a 3.3 ábrán látható - is hasonló eltérések mutatkoztak, mint ami a beavatkozás nélkülin volt. Habár észrevehetőek eltérések azonban a számokat megvizsgálva, hasonlóan alakultak a minimum és maximum értékek, illetve a pirossal kiemelt érték tartományban is azonos valószínűséggel szerepelnek a különbségek, ami az összes érték 40-45%-át teszi ki. Így tehát az kimondható, hogy nem javított sem nem rontott az, ha módosítottunk a számítási környezeten. De összességében vizsgálva a kapott eltéréseket jelentősnek mondható az eredmény, amit kaptunk.



3.3. ábra A számítási környezet módosításaival kapott eltérés

4. Neuron kiértékelésének vizsgálata

Ebben a fejezetben azt fogom vizsgálni, hogy a tanulási fázisok során milyen mértékben változik a hiba nagysága és erre milyen környezeti tényezők vannak hatással. Ahhoz, hogy ezt szemléltetni, tudjam szükséges először annak a problémának a megoldása, hogy előidézzem a kerekítési hibákat. Ezt a korábbi példámban úgy oldottam meg hogy vettem a számok összes lehetséges permutációját és kiválasztottam a két szélső értékét az összegeknek. Ebben a fejezetben már egy valós neurális hálónak egy neuronját fogom vizsgálni, ahol a rejtett rétegeken a neuronok száma 128 ez azt jelentené 128! műveletet kellene elvégezni, ami hatalmas erőforrásokat igényelne.

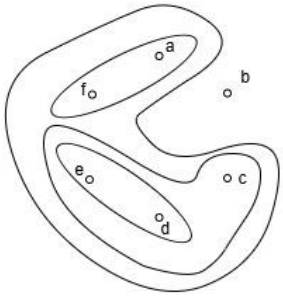
A problémát először úgy közelítettem meg, hogy a bemeneti számokat mit rendezett lista vettem és minden elemet hozzáadtam a szomszédos elemhez, aztán az ebből létrejött eredmény listán hajtottam végre ugyanezt a műveletet. A második sorrend kialakításakor pedig először két részre bontottam a bemeneti halmazt úgy hajtottam végre ezt a kiértékelést és azok eredményét a végén adtam össze. Ez nem hozott eredményt mert bár többszöri tesztelés során keletkezett olyan esett, ahol ez előhozta a kerekítési hibát, de az esetek túl nyomó többségében nem. Viszont nekem arra volt szükségem, hogy minél gyorsabban generálhassak két eltérő eredményt adó műveleti sorrendet.

Ezért egy olyan algoritmus a megoldás, ami a kapott számok halmazát teljesen véletlenszerűen adja össze. Ez azért is fontos, mert itt is az eltérő nagyságrendű számok összeadása okozhat hibát. Ezért olyan különböző csoportosításokra van szükség, ahol ez a helyzet előáll emellett az is szükséges, hogy két egymás utáni sorrend generálás során minél eltérőbb szerkezetű műveleti sorrendet kapjunk ehhez minden szabályszerűséget kerülni kell. Az általam leírt megközelítésben is ez volt a probléma.

4.1. Hipergráf

A gráfok definíció szerint pontokból és élekből álló halmaz, ahol a pontokat élek kötik össze és minden él pontosan két pontot érint. A Hipergráf ezt annyiban módosítja, hogy nem kettő, hanem tetszőleges számú pontokat köt össze, ezeket az éleket pedig hiperéleknek nevezik. Ez egy halmaz elmélet struktúra, ami csoportosítási problémák megoldására kiválóan alkalmazható kiegészítve azzal, hogy egy csúcs pontosan egy élhez tartozhat, illetve, hogy egy hiper él egy külön álló csúcs csoportot jelképez, ami, mint összesség szintén tartozhat egy hiper élhez. Ezzel egy diszjunkt gráf jön létre, amely minden csúcsot lefed.

A jelenlegi problémára is tekinthetünk egy fajta csoportosítási problémaként. Mert azt kell meghatároznunk, hogy a zárójelek elhelyezésével milyen csoportokon milyen sorrendben hajtjuk végre az összeadás műveletét. Ebben az esetben a számok a csúcsok és minden hiper él egy zárójelezést határoz meg. Mivel minden csúcs csak egy élt érinthet ezért kizárható, hogy egy érték kétszer kerüljön hozzá adásra és abban az esetben, ha egy hiperél tartalmaz egy másikat az matematikailag is kifejezhető a megfelelő zárójelek elhelyezésével.



4.4. ábra: Példa egy hat elemű hipergráfra

A 4.4. ábrán látható példa mutat egy ilyen csoportosítás, ami összeadáskánt a következőképpen értelmezhető:

$$\sum = ((a + f) + \Big(\big(c + (d + e)\big)\Big) + b$$

4.1.1. Megvalósítás

A feladat tehát az, hogy egy tetszőleges nagyságú tömb elemeit véletlenszerűen csoportokra bontsuk. Ehhez először vettem a rendezett tömböt és egy generált random számot összeszoroztam a tömb méretével és ez a szorzat adja, hogy az első rész halmazba mennyi elemet kell véletlenszerűen kiválogatni. A véletlen számot nulla-egy intervallumon belül generálom ennek köszönhetően ez a szorzat garantáltan kisebb lesz a tömb méreténél. Ez a szorzat tört számokat eredményez ezért szükséges, hogy kerekítve legyen az érték. Ez azt jelenti, hogy minden x szám az x±0,5 intervallumról lesz kerekítve kivétel a nulla és a lista méretével megegyező szám, mivel ezek az intervallum szélén helyezkednek el. Ez azt eredményezi, hogy nem megegyező valószínűséggel kerülnek kiválasztásra ebben a felállásban. Ugyanakkor az, hogy nulla elemet válasszunk ki annak gyakorlatban nincs haszna ezért abban az esetben is, ha nullát kapok akkor azt úgy kezelem mintha a lista hosszát kaptam volna, így minden szám ugyanakkora valószínűséggel kerül kigenerálásra. Az így kapott szám határozza meg, hogy mennyi elemet veszünk ki a listából, amit aztán egy új listába gyűjtünk.

Addig hajtjuk végre ezt a lépést ameddig az eredeti tömbben nem marad több elem. Ezután mivel ahhoz, hogy egész konkrétan meghatározzuk az összeadások sorrendjét

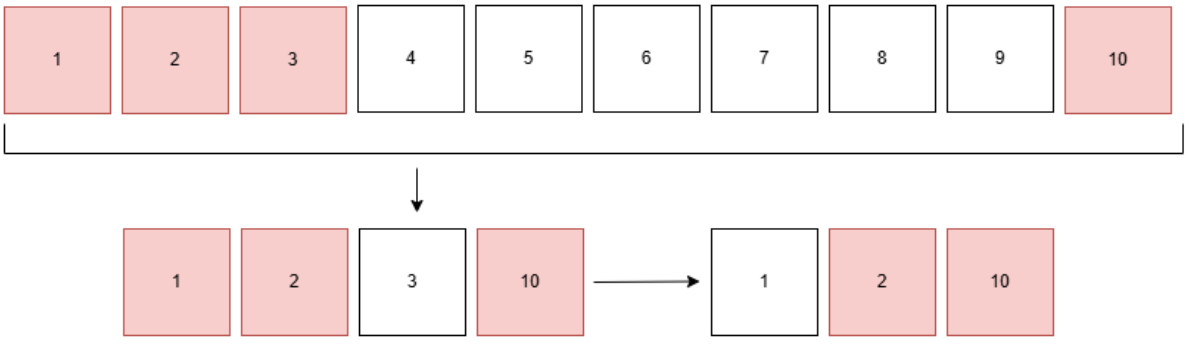
szükséges, hogy minden létrejött rész halmazra elvégezzük ezt a műveletet ameddig nem bontottuk le olyan mélységre, hogy már csak egy vagy két elemű részhalmazok léteznek. Ennek a megvalósítását a 4.5.-ös ábrán látható kód részlet tartalmazza.

```python
def get_random_elem_and_remove(lista):  1 usage
    random_index = random.randrange(len(lista))
    item = lista.pop(random_index)
    return item

def get_subset(lista):  1 usage
    element_num = round(len(lista) * random.random())
    element_num = element_num if element_num != 0 else len(lista)
    subset = []
    for _ in range(element_num):
        subset.append(get_random_elem_and_remove(lista))
    return subset

def split_for_subset(lista):  5 usages
    subsets = []
    i = 0
    while len(lista)!=0:
        i = i+1
        subset = get_subset(lista)
        if len(subset)>2:
            subset = split_for_subset(subset)
        subsets.append(subset)
    return subsets
```

4.5. ábra Hipergráf implementációja

A split_for_subset(lista) hívással generálhatunk ilyen módon egy hiper gráfot. Ez egy rekurzív függvény tehát önmagára hivatkozik ennek eredményeképpen minden dimenzióban rész halmazokra bontja bementet.

Egy konkrét példában ez azt jelenti hogyha veszünk egy tíz elemű tömböt akkor első lépésként abból véletlenszerűen kiválaszt véletlen számú elemet.



4.6. ábra Hipergráf szerinti bontás

Ezután ezeken a kiválasztott elemeken hajtja végre ezt a műveletet egészen addig amig a kiválasztott elem lista mérete nem egy vagy kettő. Ezután felépíti rész halmazt a generált

csoportosításból. A 4.6-os ábrán szerepel egy példa, ahol egy tíz elemű listából kiválasztunk véletlenül véletlen számú elemet. Ez után ezen a négy elemű rész halmazon is végrehajtjuk ezt újra és kiválasztunk hármat random. Itt is véletlen számú elemet választunk ki, ami kettő ebben az esetben. Itt a rekurzió megszakad és felépítjük a tömböket, ami a zárójeles jelöléssel a következőképpen néz ki:

$$\Big(\big((10+2)+1\big)+3\Big)$$

Azért határoztam meg azt, hogy addig bontson részhalmazokra egy tömböt ameddig nem egy vagy nem kettő elemű mert ezzel az egész konkrét összeadási sorrendet meghatározza, illetve kilépési pontként is szolgál, hogy ezzel elkerülhető legyen, hogy végtelen ciklusba kerüljön a függvény.

Miután az eredeti halmazból először kiválasztott listán végig ér, tehát a fenti ábrán vett példában a (1 ,2,3,10) számokat kisebb részegységekre bontotta például a korábban leírt módon akkor a fennmaradó listából választ egy következő véletlenszerű rész halmazt, amin szintúgy végrehajtja a tovább bontást és ezt folytatja egészen addig míg az eredeti tömbből el nem fogytak az elemek.

A példának egy lehetséges felbontása a kód kimenete szerint az alábbi módon néz ki.

[[[[[10, 2], [1]], [3]]], [[6], [8, 5]], [9], [7], [4]]

Ez az eredmény amennyibe összegként írjuk le így néz ki:

$$\Big(\Big(\big(\big((10+2)+1\big)+3\big)+\big(6+(8+5)\big)+9\Big)+7\Big)+4$$

4.1.2. Alkalmazás egy neuron tekintetében

Az implementált függvények segítségével már elő idézhető, hogy egymás után teljesen véletlen szerű halmazok jöjjenek létre. Valós példaként, hogy ezt bemutatni tudjam az MNIST egy tanuló adatát hívtam segítségül.

Ezt az ábrát bemeneti paraméterré alakítottam, ennek eredménye egy 784 jellemzőből álló vektor és ehhez generáltam egy súly vektort ezzel lényegében egy neuront imitálva. Vettem ezeknek az elemenkénti szorzatát és az eredményként kapott vektoron végeztem el a csoportosítást és az aszerint történő összeadást. Optimalizálásként, a szorzatnak a listájából eltávolítottam a 0 értékű elemeket mivel azok a végösszegen nem módosítanak azonban a részhalmazokra bontás futási idejét, illetve az abból való kiolvasás idejét lényegesen megnöveli. Ellenőrzésképpen háromszor hajtottam végre csoportosítást és vizsgáltam meg azok összegének eltérését. Ebben az esetben is jelentkezett hiba, aminek mértéke 10^{-6} volt

nagyságrendileg. Ez önmagában kis érték ahhoz képest, amit a korábbi hisztogramokon láthattunk viszont a tanulás során ezek a hibák halmozódhatnak.

4.2. Kerekítési hibák a neurális hálózatban

Eddig sikerült kimutatnom egy neuron esetében már azt, hogy milyen kerekítési hiba lép fel, ha egy bemeneti vektor és a hozzá tartozó súlyok szorzatát összegzem, de kérdés, hogy ez a tanulási fázisok során milyen mértékben növekszik és miképpen hat a súlytényezőkre. Ahhoz, hogy ezt vizsgálni tudjam a korábban használt kiértékelést fogom hasznosítani, amit a korábbi számítási példámnál mutattam. Ennek segítségével vizsgálni fogom, hogy ahogyan halad előre a feldolgozás és aközben az eltérések milyen módon változnak.

4.2.1. A tanuló algoritmus

Az implementációmat a korábbi fejezetben leírt képletek szerinti alkottam meg, ez nem egy optimalizált tanuló algoritmus, viszont arra megfelelő, hogy bemutassam azt, hogy kisebb mennyiségű tanulási fázis során mekkora eltérések tapasztalhatóak.

```
def relu(sign_vector):
    return np.where(sign_vector > 0, sign_vector, 0).astype(np.float32)

def relu_derivative(input_vector):  1 usage
    return np.where(input_vector > 0, 1, 0).astype(np.float32)

def deviation(actual_vector, expected_vektor):
    return expected_vektor-actual_vector.astype(np.float32)

def deviation_link_back(sign_vector, deviation_vector, weights):  5 usages
    relu_deriv = relu_derivative(sign_vector)
    hidden_delta = deviation_vector * relu_deriv
    result = weights @ hidden_delta
    return result.astype(np.float32)

def modify_weights(weights_before, deviation, vector_x):  10 usages
    delta = 0.01 * np.outer(vector_x, deviation).astype(np.float32)
    w = weights_before + delta
    return w
```

4.7. ábra Neuronhálót kiértékelő függvények

Első lépésként meghatároztam a tanuláshoz szükséges függvényeket, a neuron összegző algoritmusa a korábban ismertetett hiper gráfos megközelítéssel történik, az aktivációs függvény ebben az esetben is a ReLu. A 3.1.1-es fejezetben ismertetett képletek alapján a 4.7. ábrán látható a kód részlet.

A korábbi példámtól eltérően a tanulási rátának nem konstans értéket adtam meg mert ahogy, írtam ez az algoritmus nem jól optimalizált és elég hamar előjött az a probléma, hogy a neurális háló instabillá vált. Ennek egy jelentkező tünete az volt, hogy ha több iterációt szerettem volna végrehajtani akkor túlcsordulás a szorzásban figyelmeztetést kaptam és aztán hibát dobott a program azzal az üzenettel, hogy az automatikusan észlelt tartomány nem véges. Ennek egy lehetséges oka, hogy a megadott tanulási ráta túl nagy ezért a tanulási függvény nem konvergál a megoldás felé, mert túl nagyot lép. Ennek kiküszöbölésére, lehet optimalizálni ezt az értéket azzal arányosan ahogy haladunk előre a tanulás során. Ezt úgy tettem meg, hogy a tanulási rátát így állapítom meg: $rate = 0.01/(1 + 0.1 * i)$, ahol i az iterációk számát jelöli.

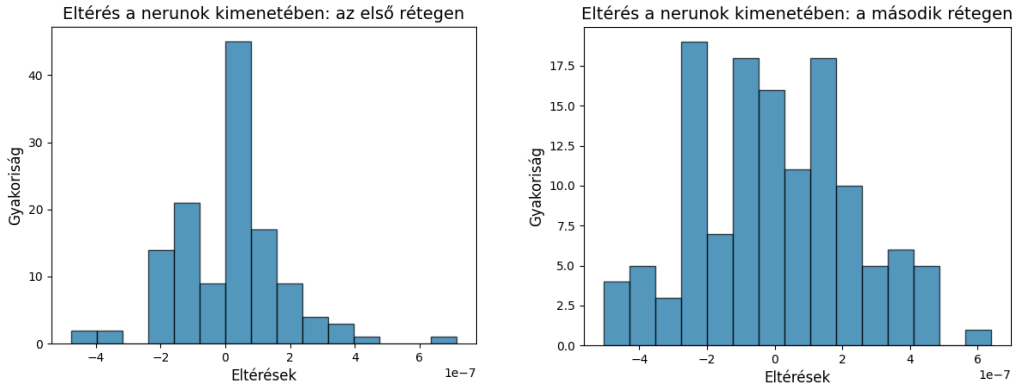
Ahhoz, hogy megfelelően vizsgálni tudjam a tanulási folyamatokat ezért szükségem volt arra, hogy a tanulási fázisokat, mint entitás tudjam kezelni mert akkor könnyebben tudom az eredményeket összevetni. Ezért létrehoztam a LearnPhase modellt, ami lényegében egy tanulási ciklus összes állapotát tárolja el. A tanulási lépéseket már az inicializálás során meg is teszi és saját változóiba elmenti, ennek megvalósítása a mellékletben található.

A súlyok meghatározásánál látható, hogy egy két rejtett réteggel rendelkező neuronhálóról beszélünk, amiknek mérete az MNIST tanuláshoz van előkészítve. Ezért is a bemeneti rétegen a százhuszonnyolc neuronhoz egyesével tartozik a hétszázsznyolcvannégy inger. Ami a kép, az eredeti 28x28 pixeles méretének a kilapítása.

A tesztelés során létrehozok egy neuronhálót és azoknak az adatait lementem, ezekután párhuzamosan két szálon tanítom az általam megalkotott modell mentén. Így a folyamat során lementem az egyes iterációknak az állapotát és ezzel lehetővé válik, hogy a keletkező eltéréseket megvizsgáljam.

4.2.2. A neuronok összegének vizsgálata

A tanítás során két iterációt indítottam el párhuzamosan, ugyanazokkal a kezdeti paraméterekkel és a fázisok során megegyező tanító adattal.



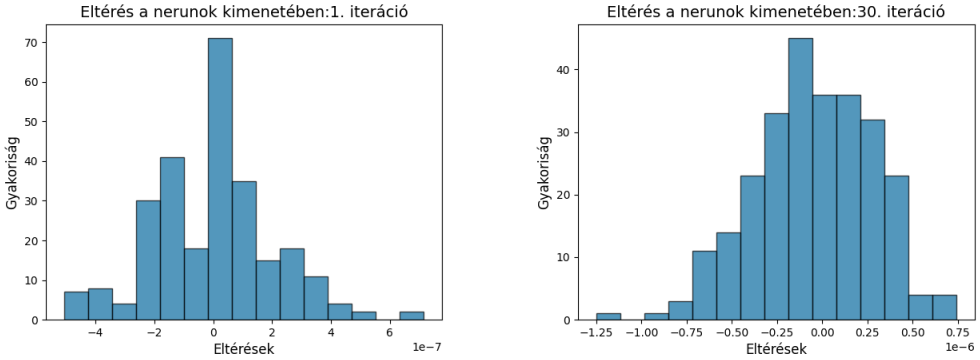
4.8. ábra Neuronok kimenetének eltérése rétegenként

A korábbi számítási példa elemzésekor feltűnt, hogy a tanulási fázison belül a második réteg neuronjai esetében nagyobb eltérések keletkeztek. Ennek szemléltetésére látható két diagram a 4.8. ábrán. A baloldali hisztogram az első réteg neuronjainak az eltérése látható, a jobb oldalin meg a második réteg eltérésének hisztogramja. A harmadik rétegről külön nem készítettem diagramot mert az összesen tíz kimeneti neuronnal rendelkezik így azok értékét különállóan nem tudtam megjeleníteni. Az látható ezeken az ábrákon, hogy hasonló mértékű eltérések voltak tapasztalhatóak azonban az első réteg esetében ezek jobban a nulla értékhez tömörülnek ezt mutatja az is, hogy a jobb oldali diagram skálázása nagyobb.

Ennek oka, hogy az első neuron kimenete befolyással van a második rétegnek a bemenetére, azt is fontos megjegyezni, hogy ezek az eltérések még az aktivációs függvények alkalmazása előtti állapotra vonatkozik, az értelemszerűen elnyeli ezeket az eltéréseket, ha a neuron kimenete negatív értékű. Az első rétegen a százhuszonnyolc eltérésből huszonöt volt nulla míg a következő rétegen már csak tizenöt.

Ezzel a korábban tett állítás valós, hogy a rétegeken előre haladva ezek az eltérések növekednek így tehát egymással hatással vannak. Viszont felmerül a kérdés, hogy a tanulási fázisokon átívelően ezek az eltérések bármilyen módon változnak és ha igen mekkora mértékben. Ennek vizsgálatához szintén párhuzamosan két szálon tanítottam az adatokat és hisztogramot készítettem az első és a harmincadik iterációban keletkező különbségekről.

A bementi adatok az összes neuron kiértékeléséből származik, ez összesen fázisonként 266 értéket jelent, ami a két neuronháló kiértékelésének a különbségéből állt elő. Ebben az esetben is az aktivációs függvény előtti értékeket vizsgáltam. A 4.9. ábra bal oldalán látható az első réteghez tartozó eltérések, a jobb oldalon pedig a harmincadik tanulási fázishoz tartozóak.

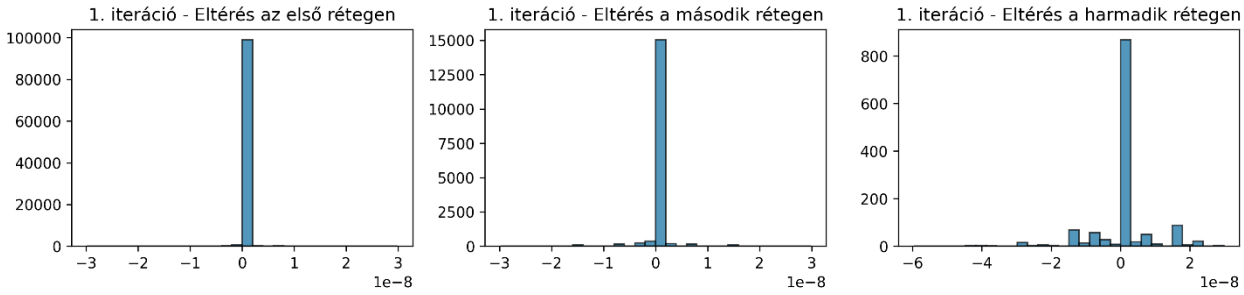


4.9. ábra Neuronok kimenetének eltérése

Itt már látható, hogy a kettőt összevetve a tanulás későbbi fázisában az eltérések egy nagyságrenddel nagyobbak, illetve, hogy ezen egyenletesebben oszlanak el az értékek míg a tanulás elején sűrűbb eloszlást mutat 0 körül. Ezzel kimutathatóvá vált, hogy a kisméretű eltérések egymással hatással vannak és így ezek mértéke erősödik.

4.2.3. A tanulási fázisok vizsgálata

A neuronok kimenetének vizsgálata után azt is szemügyre vettem, hogy a tanított súlyok mennyiben térnek el egymástól a különböző iterációk során és hogy a tapasztalt eltérések milyen hatást gyakorolnak azokra.

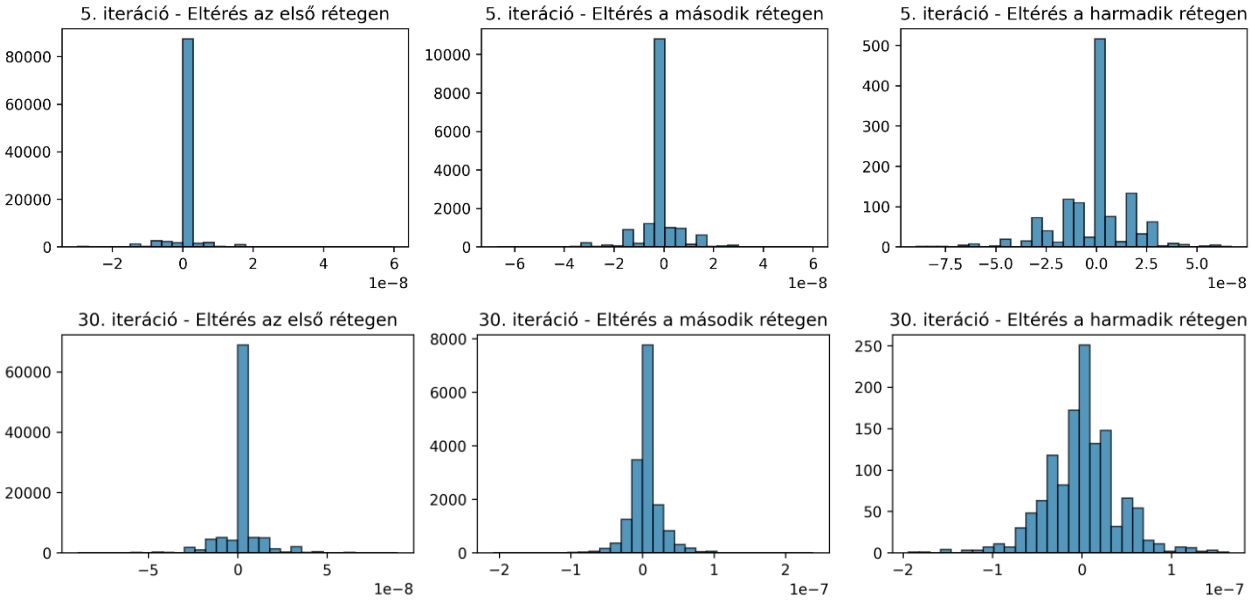


4.10. ábra Különböző futásokból a súlyok eltérése rétegenként – első iteráció

A 4.10. ábrán az látható, hogy az első iteráció végén tehát mikor még csak egy kép alapján történt a tanítás akkor a két futás között mekkora eltérés jelentkezett. Leolvasható a diagramról, hogy a kimeneti rétegen lévő eltérések szórása látványosabb, de még az is elenyésző míg a bemeneti rétegen lévőé kisebb.

A 4.11. ábrán az ötödik és harmincadik tanulási fázis utáni súlyok eltérése látható. Látszik, hogy ahogy a tanulásban haladunk előre egyre több különbség tapasztalható. Az ötödik iterációban is a tanult súly már nagyobb eloszlást mutat az elsőhöz képest a harmincadik után pedig még inkább és már nagyságrendileg is nagyobbak ezek az értékek.

Mindegyik diagrammon látható, hogy a kimeneti rétegnek az eltérése nagyobb, mint a két rejtett rétegben lévőé. Ennek egyik oka, az előző fejezetben bemutatott tapasztalat, hogy a rétegekben előre haladva növekszik a különbségek értéke ezért is a tanulás egyenletéből adódóan ezek a súlyok értékében is hatást gyakorolnak.



4.11. ábra Különböző futásokból a súlyok eltérése rétegenként – ötödik és harmincadik iteráció

4.2.4. Eredmények értelmezése

A grafikonok alapján egyértelműen kimutatható az, hogy két azonos architektúrával rendelkező neuronmodell azonos tanítási adatokkal végzet tréningezése során, már a kezdeti iterációk során is eltérések keletkeztek a számítógépes szám ábrázolás okozta kerekítési hibák következményében. Az is jól látható a grafikonokon keresztül, hogy tanulási iterációkon átívelően milyen módon változnak a tanult súlyok különbségei., ez alapján elmondható, hogy a tanulási folyamat során nem eltűnnek ezek a jelentkező eltérések, hanem erősödhetnek is a hiba terjedés útján.

Ez a jelenség összességében arra utal, hogy az azonos tanítási körülmények ellenére sem tekinthetjük a modell tanulását teljes mértékben determinisztikusnak, mert. bizonyos mértékben eltérő hálózatok lesznek a tanítási folyamat végeredményei. A különbségek ellenére, a tanított hálózatok funkcionálisan hasonló eredményt adnak és valószínűleg a teszt adathalmazon hasonlóan jól teljesítenek. Viszont robusztusság és megbízhatóság szempontjából akár komoly kockázati tényezőt is jelenthetnek.

Különösen fontos ez az ellenséges példák vonatkozásában, mert ezek az apróbb eltérések viszont elegendőek lehetnek ahhoz, hogy a két modell különbözőképpen reagáljon, azonos támadások esetében. Így ez rámutat arra, hogy ezen támadások sikeressége nem csak az adatok manipulálásának eredményeképpen, hanem a belső numerikus instabilitás következménye is lehet.

5. Összegzés

Egy kézzel írt számítási feladat során végig vezettem az olvasót, hogy hogyan is működnek a neurális hálózatok és hogy hogyan keletkezhetnek a kerekítési hibák a számítógépes környezetben.

A dolgozatom során bemutattam, hogy egy neuron kiértékelése során pusztán a számok különböző sorrendben történő összeadása különböző eredményeket okoz számítógépes környezetben. Ennek demonstrálásához egy hiper gráf implementációt használtam. Ez a megközelítés sikeresen megoldást adott arra, hogy nagyobb számhalmaz esetében is könnyen és gyorsan tudjak különböző műveleti csoportosításokat létrehozni úgy, hogy az összeadások után valóban különböző eredményeket adjanak. Viszont ez a megközelítés nem adott arra lehetőséget, hogy azt is vizsgáljam, hogy a párhuzamosított számítások során milyen belső felosztások és operációs sorrendek vannak. Ezek megközelítéséhez más típusú gráf modellek alkalmazhatóak, mint például az Erdős–Rényi vagy a Barabási–Albert gráfok.

Bemutatom azt is, hogy neuronok során keletkező hibák milyen módon vannak hatással a tanulási folyamatra és azt is, hogy ezek az eltérések a fázisokon előre haladva egyre nagyobb eltéréseket indukálnak. A vizsgálat fókuszában az állt, hogy részletesen leírjam és bemutassam azt, ami a jelenség hátterében áll. Azt tehát dolgozatomban nem vizsgáltam, hogy ez a hálózat robusztusságára és teljesítő képességére milyen hatással van, ehhez általánosságban kellett volna vizsgálnom és nem csak egy-egy futás eredményére koncentrálnom. Azonban összeségében sikerült kimutatnom azt, hogy helyesen betanított neurális hálózatok esetében is a számítógépes környezet okozta kerekítési hibák látható eltéréseket tudnak okozni.

Irodalomjegyzék

[1] https://www.inf.u-szeged.hu/~tothl/ann/Neuronhalok-egyben.pdf (2025. 05. 21).

[2] Brassai, Sándor Tihamér NEURÁLIS HÁLÓZATOK ÉS FUZZY LOGIKA (Scientia Kiadó Kolozsvár·2019)

[3] Daniel Zombori, Balázs Bánhelyi, Tibor Csendes, István Megyeri, Márk Jelasity FOOLING A COMPLETE NEURAL NETWORK VERIFIER In: Proceedings of the International Conference on Learning Representations (ICLR 2021), 2021.

[4] Hanwei Zhang, DEEP LEARNING IN ADVERSARIAL CONTEXT

[5] Julien Girard-Satabin, VERIFICATION AND VALIDATION OF MACHINE LEARNING TECHNIQUES

Mesterséges neuronhálók kiértékelésének vizsgálata

Nyilatkozat

Alulírott Lindmayer Kinga programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Számítógépes Optimalizálás. Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2025. 05. 21.

Lindmayer Kinga

Köszönetnyilvánítás