Megtanult adatok elfeledtetése mély neurális hálókban

BSc Szakdolgozat

Írta:

Bicskei Beatrix

Matematika BSc

Alkalmazott Matematikus Specializáció

Témavezető:

Benkő Mária Beatrix

Matematikai Intézet

Számítástudomány Tanszék

Eötvös Loránd Tudományegyetem

Természettudományi Kar

Budapest, 2023

NYILATKOZAT

Név: Bicskei Beatrix

ELTE Természettudományi Kar, szak: Matematika BSc

NEPTUN azonosító: GSLBZE

Szakdolgozat címe:
Megtanult adatok elfeledtetése mély neurális hálókban

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2023.06.06.

Bicskei Beatrix

a hallgató aláírása

Köszönetnyilvánítás

Szeretném megköszönni a témavezetőmnek, Benkő Beatrixnak a lehetőséget, hogy nála írhattam a szakdolgozatomat e rendkívül izgalmas témából, illetve a rengeteg tartalmas konzultációt, melyek során sokat tanulhattam a neurális hálókról és kapcsolódó témakörökről.

Tartalomjegyzék

1. Bevezetés

A technológia teljesen beépült a mindennapjainkba: az interneten olvassuk a híreket, kommunikálunk ismerőseinkkel, vásárolunk, tájékozódunk stb. Mindez óriási mennyiségű adatot generál, amit az általunk igénybe vett szolgáltatások üzemeltetői fel is szeretnek használni arra, hogy javítsák a termékeiket, és akár további fogyasztásra ösztönözzenek. Ezt különböző gépi tanulási modellekkel tudják megtenni, amelyek képesek hatalmas mennyiségű adat gyors és hatékony feldolgozására. E modellek már minden mobileszközben, böngészőben stb. futtathatók és hasznos kiegészítésként használhatók, hála a nagy (és folyamatosan növekvő) számítási kapacitásoknak, amelyek manapság a rendelkezésünkre állnak.

A gépi tanulás lényege, hogy egy adott adathalmazban mintázatokat keresünk. Általánosságban elmondható, hogy minél több adaton tanítunk modelleket, annál jobb, de előfordulhatnak szituációk, amikor el szeretnénk távolítani adatokat, és ez nem olyan egyszerű, mint egy közönséges adatbázisból törölni rekordokat. Egy gépi tanulási modell ugyanis gyakran „emlékszik" az adatokra, amiken tanították.

Egy tipikus eset az adatvédelem és -biztonság: manapság megnőtt az igény arra, hogy a felhasználók szabadon törölhessék az adataikat, és ezt jogszabályok is garantálják, ilyen például az Európai Unió 2016-ban megjelent GDPR (General Data Protection Regulation) nevű rendelete. Fontos továbbá, hogy a hibás adatok törölhetőek legyenek az egészségügyben alkalmazott modellek esetén, hiszen hibás adatokból hibás predikciók is következhetnek, és ez emberéletekbe kerülhet. Ezen kívül a közösségi oldalakon, illetve egyéb online szolgáltatások esetén előfordulhat, hogy a felhasználó kellemetlen ajánlatokat kap néhány rendkívüli keresési előzménye után, és hiába törli az előzményeit, nem változik semmi, mert a modell a már kitörölt adatokat is felhasználta a tanulás során.

A fenti példákon keresztül tehát világossá válhat, hogy szükség van módszerekre, amelyekkel szándékosan elfeledtethetünk adatokat a gépi tanulási modelljeinkből. Szakdolgozatom célkitűzése, hogy általános bevezést nyújtson a gépi tanulás területén belül a szándékos elfeledtetés témakörébe, részletesen bemutasson konkrét algoritmusokat, amelyek mély neurális hálókon is alkalmazhatóak, és kísérleteken keresztül demonstrálja ezek hatékonyságát. A második fejezetben a szakirodalomban használt alapvető definíciókat, fogalmakat és módszereket vezetem be, a harmadik fejezetben pedig részletesen bemutatok egy egzakt és egy közelítő algoritmust. A negyedik fejezetben saját kísérleteken keresztül mutatom be a harmadik fejezetben leírt Fisher-algoritmus teljesítményét a CIFAR-10 adathalmazon. Az ötödik fejezetben végül található egy összefoglaló a szakdolgozatról, illetve kitekintés további kísérletekre és lehetséges vizsgálódási irányokra.

2. Machine unlearning - Adatok elfeledtetése

Az adatok szándékos elfeledtetésének legegyszerűbb módja, ha a tanító adathalmazból eltávolítjuk a törölni kívánt adatokat, és újratanítjuk a teljes modellt. Ez azonban nem hatékony, mert sok időt és erőforrást igényel. Az elfeledtetési algoritmusok ezt hivatottak javítani különböző módszerekkel, azaz ezeket alkalmazva nem feltétlenül kell újratanítanunk a teljes modellt.

Itt érdemes megemlíteni kapcsolódó fogalmakat is, mint a katasztrofikus felejtés, illetve a differenciált adatvédelem. Előbbi alapvetően a hosszú távú tanuláshoz kapcsolódik, melynek során szekvenciálisan tanítjuk a modellt több feladaton vagy adathalmazon, és ennek során a korábbi tanítási iterációkban megtanult minták felismerésében drasztikus teljesítménycsökkenés következhet be - ezt nevezzük katasztrofikus felejtésnek. Ez nem alkalmas gépi adatok elfeledtetésére, mivel ekkor nem irányított módon történik a felejtés. Ugyanakkor hosszú távú tanulási modellekben már megpróbálkoztak irányított elfeledtetéssel (ezt a 2.2 fejezetben részletesebben ki is fejtem).

A differenciált adatvédelem algoritmusokra vonatkozik. Egy algoritmus differenciált adatvédelemmel rendelkezik, ha bármely adathalmazbeli elem lecserélése esetén az algoritmus kimenete „közel" marad az eredeti adathalmazon kapott kimenethez (a közelségre természetesen létezik pontos definíció, de ez nem tárgya e szakdolgozatnak), azaz a kérdés az, hogy mennyire őrzi meg az algoritmus az egyes tanító adatpontokra vonatkozó egyedi jellemvonásokat. Ezzel szemben a „machine unlearning" terület azzal foglalkozik, hogy különböző módszereket találjunk egy adatpont avagy adathalmaz eltávolítására. Tehát nem egy már adott algoritmust vizsgálunk ekkor, mint a differenciált adatvédelem esetén, hanem mi magunk keresünk egy hatékonyat az elfeledtetésre. Ekkor persze adódhat, hogy differenciált adatvédelemmel rendelkező algoritmusokat használjunk elfeledtetésre, de Sekhari et al. (2021) alapján ez nem feltétlenül éri meg, mert bebizonyították, hogy a differenciált adatvédelemmel rendelkező algoritmusokkal a sima elfeledtetési algoritmusokhoz képest kisebb mennyiségű adatpontot feledtethetünk el úgy, hogy a teszthalmazon jó maradjon a teljesítmény.

Ahhoz, hogy össze tudjuk hasonlítani az algoritmusokat, szükség van pontos fogalmakra, amelyeket az alábbiakban a Mercuri et al. (2022) és Nguyen et al. (2022) cikkek alapján foglalok össze.

2.1. Alapfogalmak

Legyen $\mathcal{X}$ a bemeneti adatok tere, $\mathcal{Y}$ a kimeneti tér, és egy adatpont a $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ egy eleme. Legyen $\mathcal{Z}^*$ az adathalmazok tere. Egy $D \in \mathcal{Z}^*$ adathalmaz adatpontok

multihalmaza.

A tanítás célja egy $h : \mathcal{X} \to \mathcal{Y}$ hipotézisfüggvény megtanulása egy D adathalmaz esetén. $\mathcal{H}$ az ezen függvények halmaza. E h függvény egy adott $x \in \mathcal{X}$ inputhoz hozzárendel egy $y = h(x) \in \mathcal{Y}$ kimenetet, így a tanító algoritmusra tekinthetünk úgy, mint egy $\mathcal{A} : \mathcal{Z}^* \to \mathcal{H}$ leképezésre, melynek célja, hogy minimalizálja a nemnegatív valós értékű $L(h, D)$ költségfüggvényt. A vizsgált tanító algoritmusok randomizáltak, azaz egy bizonyos fokú véletlenszerűség figyelhető meg bennük, például a tanítás során a súlyokat véletlenszerűen inicializáljuk, vagy éppen az tanító adatokat véletlenszerűen választjuk ki a sztochasztikus gradiens módszer során. Ezeket a fogalmakat a iii. függelékben definiáltam.

Az elfeledtetés során a D tanító adathalmazban $n \geq 1$ minta van, és a dimenzió is legalább 1. $\mathcal{A}$ randomsága miatt a $\mathcal{A}(D)(x)$ kimenet egy $x \in \mathcal{X}$ pontban tekinthető valószínűségi változónak. Ha $\mathcal{A}$ parametrikus, akkor létezik egy $\theta \in \Theta$ fix dimenziójú paramétertér, mely meghatározza a $h_\theta = \mathcal{A}(D)$ hipotézisfüggvényt, és ebben az esetben a θ és a h_θ jelölések közül bármelyik használható a betanított modell jelölésére. Az elfeledtető algoritmus célja, hogy eltávolítsa a hatását egy m méretű $D_u \subseteq D$ részhalmaznak a betanított $\mathcal{A}(D)$ modellben.

2.1. Példa. *E szakdolgozatban a mesterséges neurális hálókon való elfeledtetést fogom részletesebben vizsgálni, és ebben az esetben tipikusan képosztályozási problémákat tekintünk. Ennek következtében itt $\mathcal{X}$ a színes képek halmaza, és $\mathcal{Y}$ az osztálycímkék, $\mathcal{A}(D)$ a sztochasztikus gradiens módszerrel betanított neurális háló, $L(h, D)$ pedig a kereszt-entrópia.*

2.2. Definíció (**Frissítőmechanizmus**)**.** *A frissítőmechanizmus egy*

$$\mathcal{U} : \mathcal{H} \times \mathcal{Z}^* \times \mathcal{Z}^* \to \mathcal{H}$$

leképezés, amely bemenetnek egy $h \in \mathcal{H}$ függvényt és két adathalmazt, $D, D_u \in \mathcal{Z}^*$-t használ, és a kimenete egy új $\mathcal{U}(h, D, D_u) \in \mathcal{H}$ modell, ahol D az eredeti adathalmaz, és D_u ennek egy részhalmaza.

Az alábbiakban definiálom a eltávolító mechanizmusokat, amelyek az elfeledtető algoritmusok definíciójához szükségesek.

2.3. Definíció (**Egzakt eltávolító mechanizmus**)**.** *Egy $\mathcal{U}$ frissítőmechanizmus egy $\mathcal{A}$ véletlen tanító algoritmusra pontosan akkor egzakt eltávolító mechanizmus, ha ugyanazt a modellt eredményezi, mint amit a teljes újratanulással kaptunk volna. Pontosabban, $\mathcal{U}$ pontosan akkor egzakt, ha minden $D \in \mathcal{Z}^*$-ra, $D_u \subseteq D$-re és $x \in \mathcal{X}$-re a $\mathcal{U}(\mathcal{A}(D), D, D_u)(x)$ és $\mathcal{A}(D \backslash D_u)(x)$ ugyanazokat az eloszlásokat adják.*

A fenti definícióból a legegyszerűbb eset könnyen látható, ez a teljes újratanítás:

2.4. Definíció (**Naiv eltávolító mechanizmus**). *Egy $\mathcal{A}$ véletlen tanító algoritmus esetén az $\mathcal{U}^*$ naiv frissítőmechanizmus a*

$$(\mathcal{A}(D), D, D_u) \mapsto \mathcal{A}(D \backslash D_u)$$

leképezés, ahol $D_u \subseteq D$.

Az eltávolító mechanizmusok természetesen nem csak pontosak lehetnek. Léteznek ugyanis úgynevezett közelítő eltávolító mechanizmusok, melyek ugyan nem ugyanazt a valószínűségi eloszlást eredményezik, mint amit a teljes újratanítás, megadhatunk hozzájuk követelményeket, amelyekkel meggyőződhetünk róla, hogy a D_u információ el lett távolítva a h modellből.

2.5. Definíció (**Elfeledtetési tanúsítvány**). *Adott $\mathcal{A}$ véletlen tanító algoritmus és $\mathcal{U}$ frissítőmechanizmus esetén $\mathcal{U}$ elfeledési tanúsítványa ($\mathcal{C}_\mathcal{U}$) ellenőrizhető követelmények egy halmaza, amelyek kiszámolhatók és kívülről ellenőrizhetők. A követelményeknek ki kell mutatniuk, hogy bármely D adathalmaz és $D_u \subseteq D$ részhalmaz esetén az $\mathcal{U}(\mathcal{A}(D), D, D_u)$ modell kellően kevés információt tartalmaz D_u-ról. A „kellően kevés" pontos meghatározása a használt követelményektől függ.*

A fenti definícióban a követelmények lehetnek például annak bizonyítása, hogy az eltávolító mechanizmus pontos, vagy a közelítő eltávolító mechanizmusok esetén használható az alábbi követelmény, amely ellenőrzése megfelel annak belátásának, hogy a frissítőmechanizmus és a naiv eltávolító mechanizmus kimenetei statisztikailag megkülönböztethetetlenek:

2.6. Definíció (ϵ**-tanúsítvány**). *Legyen $\epsilon > 0$. Azt mondjuk, hogy egy frissítőmechanizmus $\mathcal{U}$ ϵ-tanúsítvánnyal rendelkezik az $\mathcal{A}$ véletlen tanító folyamatra nézve, ha minden $D \in \mathcal{Z}^*$-re, $D_u \subseteq D$ eltávolítandó halmazra, $x \in \mathcal{X}$ bemenetre és $\mathcal{T} \subseteq \mathcal{Y}$-ra*

$$e^{-\epsilon} \leq \frac{P(\mathcal{U}(\mathcal{A}(D), D, D_u)(x) \in \mathcal{T})}{P(\mathcal{U}(\mathcal{A}(D \backslash D_u))(x) \in \mathcal{T})} \leq e^{\epsilon}.$$

A fenti definíciók segítségével immár definiálhatjuk, hogy mit is értünk elfeledtető algoritmusok alatt.

2.7. Definíció (**Naiv elfeledtetés**). *$(\mathcal{U}^*, \mathcal{C}_{\mathcal{U}^*})$ a naiv elfeledtető algoritmus, ahol $\mathcal{U}^*$ a naiv elfeledtető mechanizmus. Az elfeledtetési tanúsítvány ebben az esetben az a triviális tény, hogy $\mathcal{U}^*$ egzakt.*

2.8. Definíció (Egzakt és közelítő elfeledtetési algoritmusok). *Egzakt elfeledtetési algoritmusnak nevezzük az* $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ *párt, ahol* $\mathcal{U}$ *egy egzakt elfeledtetési mechanizmus. Ebben az esetben* $\mathcal{C}_\mathcal{U}$ *tartalmaz egy bizonyítékot, hogy* $\mathcal{U}$ *egy egzakt elfeledtetési mechanizmus. Közelítő elfeledtetési algoritmusnak nevezünk egy elfeledtető algoritmust, amely nem egzakt.*

2.2. Az adatok eltávolításának típusai

Számtalan különböző szituáció előfordulhat, melyben szeretnénk adatokat törölni, és ezek során természetesen többféle eltávolítási kérelemtípus merülhet fel. Az alábbiakban a gépi tanulási modellekből való eltávolítási szándék esetén felmerülőket foglalom össze.

Elem eltávolítása. Ez a leggyakoribb eset. Ilyenkor bizonyos adatpontokat szeretnénk eltávolítani a tanító adathalmazból.

Jellemzők eltávolítása. Megtörténhet, hogy nem csak egy adat eltávolítására van szükség, de e adathalmaznak van egy közös tulajdonsága vagy közös osztálycímkéje. Ilyen például, amikor egy spam szűrő néhány kártékony e-mail-címet rosszul klasszifikál, vagy éppen amikor egy álláshirdetés esetén a jelentkezőket előszűrő program bőrszín és nem alapján is ítélkezni kezd. Ezen esetekben pl. nem elég egyszerűen elfeledtetni a gyanús e-maileket vagy az emberek profiljait. Ehelyett inkább a modellel az adott tulajdonságot avagy címkét érdemes elfeledtetni. Ez egyrészt hatékonyabb, mert folyton újratanítani elfeledtetés esetén kifejezetten számításigényes, illetve túl sok adat elfeledtetése ronthat a modell minőségén.

Egy lehetséges megoldás a Guo et al. (2022) cikkben található, amely a faktorizált reprezentációt használja. Ennek lényege, hogy a jellemzők közti korrelációk és a jellemzők kimenetre gyakorolt hatása alapján már el tudunk távolítani egy-egy jellemzőt úgy, hogy a többi megmaradjon, és a modell pontossága ne sérüljön nagyot. Ha például vannak képeink emberekről, és a számon tartott jellemzőik a nagy orr, bajusz és a mosolygás, akkor a mosolygást mint jellemzőt megpróbálhatjuk ezzel a módszerrel elfeledtetni.

Osztály eltávolítása. Sok esetben az elfeledtetendő adat egy vagy akár több osztályhoz is tartozhat. Arcfelismerő rendszerekben például egy ember arca egy osztálynak felel meg - azaz több millió osztály is elképzelhető - , ugyanakkor anélkül kell eltávolítanunk az ember arcát, hogy használnánk róla bármi képet. Ilyenkor csak az osztályra magára támaszkodhatunk.

Ezt a Tarun et al. (2021) cikkben úgy oldják meg, hogy az eltávolítandó osztályra a klasszifikációs hibát hozzáadott zajjal maximalizálják, majd a modellt így frissítik. Ekkor a modell teljesítménye a maradék osztályokon is romolhat, így ezek után egy javító lépésre is szükség van, hogy ez helyreálljon. Csupán néhány lépés alatt sikerült így elfeledtetni az osztályokat úgy, hogy a pontosság továbbra is jó maradt.

Feladat eltávolítása. A hosszú távú tanulás (Parisi et al., 2019) alapötlete, hogy több különböző feladat megtanulása akár egymást is erősítheti, hiszen köztük lehet korreláció. Azaz a gépi tanulási modellek nem feltétlenül csak egy, hanem több feladatra is taníthatóak szekvenciálisan.

Ebben az esetben szükség lehet feladatok elfeledtetésére, például ha egy ápolásban segítő robotot tekintünk, hasznos lenne, ha a specifikus segítési mechanizmusokat el tudná feledni, miután az adott páciensnek nincs rá többé szüksége. Ez egy különösen bonyolult feladat, mert a hosszú távú tanulás függhet a megtanult feladatok sorrendjétől is. Tehát ha egy feladatot eltávolítanánk, előfordulhat, hogy katasztrofikus felejtés következik be, és minden feladatra jelentősen romlik a modell teljesítménye (Liu et al., 2022), főképp mivel a hosszú távú tanulási módszerek is pont emiatt teljesítenek általában jelentősen rosszabbul.

Liu et al. (2022) megoldása erre az, hogy a modellnek előre megmondjuk, hogy melyik feladatot tanulja és jegyezze meg tartósan, melyiket tanulja meg úgy, hogy később vagy elfeledtetjük vagy tartósan megjegyeztetjük, illetve melyik feladatot akarjuk biztosan teljesen elfeledtetni.

Adatfolyam eltávolítása. Bizonyos szituációkban adatok egy sorozatát kellene elfeledtetni. Például ha a tanító adathalmazt ellenséges szándékkal manipulálják egy támadás során, és a szennyezett adatok fokozatos eltávolításával szeretnénk visszanyerni a modell teljesítményét.

Az adatfolyamok eltávolítása lehet adaptív vagy nem adaptív. Az adaptív esetben az, hogy mi az eltávolítandó adat, attól függ, hogy mi az aktuális elfeledtetett modell. Nem adaptív esetben a köztes állapotoktól nem függ, hogy mi az adat, amit el kell távolítani. A fenti példában a hibás tanító adatokat adaptívan távolítanánk el, minden egyes elfeledtetés után megvizsgálva, hogy melyik adatot lenne a modell szempontjából legelőnyösebb eltávolítani következőnek.

2.3. Elfeledtető algoritmusok kiértékelése

Ahhoz, hogy egy $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ elfeledtető algoritmust kiértékelhessünk, szükség van különböző szempontokra. Az alábbiakban négyet mutatok be: a hatékonyságot, a hatásosságot, a konzisztenciát és a tanúsíthatóságot.

2.9. Definíció (Hatékonyság). *$(\mathcal{U}, \mathcal{C}_\mathcal{U})$ hatékonysága az $\mathcal{U}$ eltávolító mechanizmus relatív időbeli sebességjavítása az $\mathcal{U}^*$ naiv eltávolító mechanizmushoz képest.*

Egy $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ elfeledtető algoritmus hatékonyságát a gyakorlatban úgy mérhetjük, hogy a hányadosát vesszük annak az időnek, mely ahhoz szükséges, hogy a $h^\mathcal{U} = \mathcal{U}(\mathcal{A}(D), D, D_u)$ elfeledtetett modellt megkapjuk, és annak az időnek, mely a $h^* = \mathcal{U}^*(\mathcal{A}(D), D, D_u)$ modell megkapásához szükséges. Azaz:

$$\texttt{Hatékonyság}(h^\mathcal{U}) := \frac{h^*\text{-hoz szükséges idő}}{h^\mathcal{U}\text{-hoz szükséges idő}}. \tag{2.1}$$

2.10. Definíció (Hatásosság). *$(\mathcal{U}, \mathcal{C}_\mathcal{U})$ hatásossága a $\mathcal{U}(\mathcal{A}(D), D, D_u)$ elfeledtetett modell teljesítménye a naiv újratanított teljesítményéhez képest a validációs- vagy teszthalmazon.*

A hatásosságot úgy mérhetjük, hogy összehasonlítjuk az elfeledtetett modell teljesítményét a teszthalmazon a naivan elfeledtetettel.

Legyenek y^*_{pred} a naivan elfeledtetett modell predikciói, és $y^\mathcal{U}_{\text{teszt}}$ az újratanított modellé az y_{teszt} teszthalmazon. Legyen $\mathcal{M}$ egy valós értékű metrika a teljesítményre, és $\mathcal{M}^*_{\text{teszt}} = \mathcal{M}(y^*_{\text{pred}}, y_{\text{teszt}})$, illetve $\mathcal{M}^\mathcal{U}_{\text{teszt}} = \mathcal{M}(y^\mathcal{U}_{\text{pred}}, y_{\text{teszt}})$ a naivan elfeledtetett modell és az újratanított modell teljesítménye a teszthalmazon. Egy $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ modell hatékonysága így például a fenti két mennyiség különbségének abszolútértéke:

$$\texttt{Hatásosság}(h^\mathcal{U}) := |\mathcal{M}^\mathcal{U}_{\text{teszt}} - \mathcal{M}^*_{\text{teszt}}|. \tag{2.2}$$

Minél kisebb a hatásosság értéke, annál közelebb van az elfeledtetett modell a naivan elfeledtetetthez, tehát a hatásosság esetén kisebb értékeket szeretnénk kapni.

2.11. Definíció (Konzisztencia). *$(\mathcal{U}, \mathcal{C}_\mathcal{U})$ konzisztenciája a hasonlóság mértéke a $\mathcal{U}(\mathcal{A}(D), D, D_u)$ elfeledtetett modell és a naiv újratanított modell között.*

A definíciót másképp megfogalmazva a konzisztencia azt méri, hogy az elfeledtető algoritmus az ideális elfeledtetett modellt eredményezi-e. Az egzakt algoritmusok esetében ez természetesen garantáltan magas a szintje. A szakirodalomban számos mértéket lehet találni erre a kiértékelési szempontra. Mivel a szakdolgozat a neurális hálókon alkalmazott elfeledtetési algoritmusokra fog leginkább koncentrálni, elsőként a konzisztenciára egy parametrikus modelleken alkalmazott mértéket mutatok be példaként.

Egy $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ elfeledtető algoritmus konzisztenciája parametrikus modelleken az algoritmussal elfeledtetett és a naivan elfeledtetett modellek paramétereinek távolsága az euklideszi normában:

$$\texttt{Konzisztencia}_\theta(h_\theta^\mathcal{U}) := ||\theta^\mathcal{U} - \theta^*||_2, \tag{2.3}$$

ahol $\theta^\mathcal{U}$ és θ^* a $h^\mathcal{U}$ és h^* paraméterei. Minél kisebb az értéke, annál jobb, hiszen a paraméterek annál közelebb vannak egymáshoz. Ezt a metrikát használják például a DeltaGrad kiértékelésében (Wu et al., 2020).

Klasszifikációs modellekhez (He et al., 2021) a következő mérték adható:

$$\texttt{Konzisztencia}_y(h^\mathcal{U}) := \frac{100}{n_{\text{teszt}}} \sum_{i=1}^{n_{\text{teszt}}} 1_{y^*_{\text{pred},i} = y^\mathcal{U}_{\text{pred},i}}, \tag{2.4}$$

ahol $y^*_{\text{pred},i}$ a naivan újratanított modellek predikciói, $y^\mathcal{U}_{\text{pred},i}$ pedig az elfeledtetett modell predikciói. Ez a mennyiség azon predikciók arányát adja meg, amelyek megegyeznek a naivan újratanított és az elfeledtetett modell között. Minél magasabb az értéke, annál jobb, természetesen.

Ennek létezik megfelelője regressziókra is (Golatkar et al., 2020). Ehhez először is definiálni kell a KL-divergenciát:

$$\text{KL}(P||Q) := \mathbb{E}_{X\sim P}\left[\log\left(\frac{p(X)}{q(X)}\right)\right]. \tag{2.5}$$

Ez a mennyiség azt méri, hogy P és Q valószínűségi eloszlások mennyire vannak közel egymáshoz. Minél kisebb az értéke, annál hasonlóbbak. Pontosan akkor 0 az értéke, ha $p(X) = q(X)$.

A KL-divergenciát az elfeledtetett modell és a naivan újratanított modell által adott kimenetek eloszlására alkalmazva:

$$\texttt{Konzisztencia}_{\text{KL}}(h^\mathcal{U}) := \text{KL}(\mathbb{P}(h^\mathcal{U}(x))||\mathbb{P}(h^*(x))). \tag{2.6}$$

2.12. Definíció (Tanúsíthatóság). *$(\mathcal{U}, \mathcal{C}_\mathcal{U})$ tanúsíthatósága az $\mathcal{U}$ eltávolító mechanizmus képessége arra, hogy a D_u adathalmazt eltávolítsa a $\mathcal{U}(\mathcal{A}(D), D, D_u)$ elfeledtetett modellből. A $\mathcal{C}_\mathcal{U}$ az ehhez a képességhez tartozó mértékeket és garanciákat tartalmazza.*

A tanúsíthatóság biztosítékként szolgál arra, hogy az elfeledtetett modellben valóban nincs már benne az elfeledtetni kívánt információ, és hogy az elfeledtetési kérelem teljesen végre lett hajtva a szabályozásoknak megfelelően. Ennek egy fontos része az inferenciatámadásokkal szembeni ellenállóképesség kvantifikálása és maximalizálása - az inferenciatámadás során a támadó kizárólag a betanított modell

kimeneteit használva meg tudja határozni a tanító adathalmazt, amely ha érzékeny adatokból állt, magas biztonsági kockázatot jelenthet.

A naivan újratanított modell pontossága a D_u eltávolított adathalmazon megadja azon információnak a mennyiségét, amellyel az elfeledtetett modell várhatóan rendelkezni fog az elfeledtetett adatokról úgy, hogy még sosem látta ezen adatokat. Ha a modellben maradt adatok közt van az eltávolítottakhoz hasonló, akkor ez a mennyiség elég nagy lehet. Ez a pontosság mindenesetre egy remek viszonyítási alap, amihez az elfeledtetett modellünk pontosságát a D_u halmazon mérhetjük.

Legyen $\mathcal{M}_u^* := \mathcal{M}(y_u^*, y_u)$ a naiv modell teljesítménye és $\mathcal{M}_u^{\mathcal{U}} := \mathcal{M}(y_u^{\mathcal{U}}, y_u)$ az elfeledtetett modell teljesítménye D_u-n. Ekkor a tanúsíthatóság:

$$\texttt{Tanúsíthatóság}(h^{\mathcal{U}}) := \frac{|\mathcal{M}_u^{\mathcal{U}} - \mathcal{M}_u^*|}{|\mathcal{M}_u^{\mathcal{U}}| + |\mathcal{M}_u^*|} \cdot 100. \tag{2.7}$$

A fenti képlet a szimmetrikus abszolút hibának felel meg. Az angol elnevezése alapján (*symmetric absolute percentage error*) SAPE($\mathcal{M}_u^{\mathcal{U}}, \mathcal{M}_u^*$)-nek is jelölhető.

Maga a formula azért hasznos, mert a $\mathcal{M}_u^*$ kisebb értékeit jobban bünteti.

A `Tanúsíthatóság` nagyobb értékei arra utalnak, hogy az elfeledtetett modell még tartalmaz információt a kitörölt adatokról, amit nem szabadna. Ugyanakkor mivel a `Tanúsíthatóság` csak a teljesítménybeli hasonlóságot méri a kitörölt adatokon, az alacsony értékei nem feltétlenül jelentenek valóban magas tanúsíthatóságot. Például ha a naivan újratanított modellnek magas a teljesítménye D_u-n és a `Tanúsíthatóság` alacsony, akkor az eredeti, a naivan újratanított és az elfeledtetett modell esetében is hasonló a sikerességi arány a jósolt értékeken. Ebben az esetben a `Tanúsíthatóság` nem mutatja meg, hogy az elfeledtetett modell valóban elfelejtette-e az adatokat vagy sem, mert magas lesz a teljesítménye D_u-n akkor is, ha az benne volt a tanító adathalmazban, meg akkor is, ha nem.

Metrikaként használható például a Shannon-féle kölcsönös információ (Golatkar et al., 2020). Ez két valószínűségi változó, X és Y között méri az információ mennyiségét. A képlete:

$$I(X;Y) := \mathbb{E}_x\big[\mathrm{KL}(P_{Y|X}||P_Y)\big], \tag{2.8}$$

ahol $P_{Y|X}$ az Y X-re vett feltételes valószínűségének eloszlása és P_Y az Y valószínűségi eloszlása. Ha D_u-t valószínűségi változónak tekintjük (például mert véletlenszerűen választjuk ki a kitörlendő adatpontokat), akkor az információt, mely D_u-ban és az elfeledtetett modellben is benne van, $I(D_u;\mathcal{U}(\mathcal{A}(D), D, D_u))$ jelöli.

2.4. Az elfeledtetés sikerességének ellenőrzése

Az elfeledtető módszerek ellenőrzésének célja, hogy belássuk, hogy az elfeledtetett modellek és a naivan újratanítottak között alig van eltérés - már ha van egyáltalán. A 2.3 fejezetben bemutatott kiértékelési metrikák és az ellenőrzési módszerek között az a különbség, hogy míg előbbieket használhatjuk optimalizálásra, az ellenőrzési módszereket kizárólag kiértékelésre lehet. Az alábbiakban a Nguyen et al. (2022) második fejezete alapján mutatok be lehetséges eljárásokat.

Tulajdonság-beillesztési teszt. Hozzáadunk minden adatponthoz egy tulajdonságjegyet, amely az elfeledtetendő adatokkal tökéletesen korrelál, míg a maradékkal 0 a korrelációja. Miután betanítottuk a modellt, ezek súlya nullától jelentősen el fog térni, ugyanakkor az elfeledtetés során azt várjuk, hogy a súlyok nullává változzanak (hiszen csak az elfeledtetendő adatokkal korrelál az extra tulajdonság).

Olyankor érdemes ezzel élni, ha az eltávolítandó adatnak van egy egyedi tulajdonsága, amely megkülönbözteti őt a maradék adattól, illetve megjegyzendő, hogy jelenleg csak lineáris és logisztikus modellek esetén alkalmazható az eljárás (Izzo et al., 2021).

Az elfeledés mérése. Hogy a modell mennyire felejtette el a kijelölt adatokat, különböző adatvédelmi támadásokkal, például inferenciatámadással is ellenőrizhetjük (Jagielski et al., 2022). Azt mondjuk, hogy a modell α-felejt egy tanítóhalmazbeli adatmintát, ha azon a mintán α-nál nem ér el nagyobb sikerességi arányt. A differenciált adatvédelemnél rugalmasabb ez a definíció.

Információszivárgás. A modell frissítése sok gépi tanulási modell esetében azzal járhat, hogy némi információ kiszivárog. Chen et al. (2021) kidolgozott egy inferenciatámadást annak detektálására, hogy egy eltávolított adat benne van-e a tanító adathalmazban. Ezen felül két metrikát is bevezetett a szivárgás mérésére.

- *Romlási szám:* azon adatminták, melyek jelenlétét a tanító adathalmazban magasabb bizonyossággal tudjuk megállapítani az adott támadás esetén, mint hagyományos támadással, és az összes adatminta számának aránya.
- *Romlási ráta:* a helyes osztályra vonatkozó kimeneti valószínűség átlagos javulási aránya az adott támadás és a hagyományos támadások között.

Adattartalmazásra következtető támadás (membership inference attack). E támadás arra szolgál, hogy megállapítsuk, a modell szivárogtat-e adatot, így

ennek segítségével az adat-elfeledtetési módszerünk hatékonyságát jól tudjuk mérni (Chen et al., 2021).

Backdoor támadás. A célja a gépi tanulási modell megtévesztése úgy, hogy szennyezett adatokat adunk hozzá az adathalmazhoz. Ha a szennyezett adatokat nem sikerült elfeledtetni, akkor a szennyezett adatokon továbbra is rossz predikciókat kapunk ezen adatok esetében.

Lassító támadás. Az elfeledtetett modell és az eredeti modell outputja közötti különbség ellenőrzésén túl érdemes lehet magának az elfeledtetésnek a gyorsaságára is figyelmet szentelni. Előfordulhat ugyanis, hogy egy támadás szándékosan lassítani akarja a folyamatot. Formálisan megfogalmazva: legyen $\mathcal{A}(D)$ a kezdeti modellünk, amely az $\mathcal{A}$ algoritmus segítségével lett tanítva a D adathalmazon. A támadó célja, hogy egy $D_s \subset D$ adathalmazt beszennyezzen úgy, hogy maximalizálja a D_s eltávolításának számításigényét egy $\mathcal{U}$ elfeledtetési mechanizmus használata esetén.

Osztályok közti zavar tesztelése. A lényege, hogy a kimenetek vizsgálatával eldöntsük, hogy az elfeledtetett modellből kinyerhető-e még információ az elfeledtetett adatokról. Pontosabban, kiválasztunk egy $S \subset D$ részhalmazt az adatokból, amely két kiválasztott osztály elemeit tartalmazza. Ezután véletlenszerűen kicserélünk címkéket az adatok között - ezzel kapunk egy S' halmazt. S' és $D \backslash S$ egy új D' tanító adathalmazt formálnak, amelyen egy új modellt tanítunk. Ezen az elfeledtetni kívánt adatokat S' tartalmazza. Goel et al. (2022) ebből egy tévesztési mátrix segítségével egy elfeledtetési pontszámot számolt ki, amely minél alacsonyabb, annál jobb elfeledtetett modellt jelent.

3. Elfeledtetési módszerek

Az elfeledtető algoritmusok két csoportba sorolhatók: lehetnek egzaktak és közelítőek. Az egzakt elfeledtetés sokkal több szamítást igényel, hiszen az pontosan ugyanazt az eredményt kell produkálja, mint amikor az adathalmazból kitöröljük a kívánt adatot és teljesen újratanítjuk a modellt. Ugyanakkor ez sokkal biztonságosabb a közelítő elfeledtetésnél. Ha nem távolítjuk el teljesen az adatot, akkor belátható például, hogy a neurális hálók esetén a tanító adathalmazt rekonstruálni lehet (Haim et al., 2022), illetve a tanítást manipulálni is lehet, ha a támadó hozzáfér a tanító adathalmazhoz (Shumailov et al., 2021).

A manapság elérhető egzakt algoritmusok közül az egyik legjobb az ARCANE (Yan et al., 2022). Ez egy klasszifikációs algoritmussal és az adatok előfeldolgozásával jelentős hatékonyságbeli növekedést ér el, sőt az adatok egy nagyobb hányadának eltávolításakor (azaz több mint 10% esetén) is jó eredményeket ér el. Egy korábbi népszerű algoritmus a SISA (Bourtoule et al., 2019), mely a tanító adathalmaz egy átrendezésével (ún. sharding and slicing módszerrel) csökkenti le az újratanítás idejét.

Közelítő elfeledtetési algoritmus például a DeltaGrad (Wu et al., 2020), amely az első tanításból (azaz amikor még minden adatot használtunk) elment információkat, és ezek segítségével hatékonyabban számítja ki az új paramétereket az eltávolítás után. Ebből természetesen az is következik, hogy ez csak parametrikus modellek esetén alkalmazható. Egy további - szintén parametrikus modellekre alkalmazható - algoritmus az Influence (Guo et al., 2019), amely meghatározza az eltávolítandó adat befolyását az eredeti modell paramétereire, és egy Newton-lépés segítségével eltávolítja ezt a befolyást. Egy további, Newton-módszert használó algoritmus a Fisher-algoritmus (Golatkar et al., 2020). Ez utóbbit fogom részletesebben megvizsgálni, és kísérletekben kipróbálni.

Az alábbiakban két algoritmust fogok részletesebben bemutatni, a SISA-t a Bourtoule et al. (2019) és a Mercuri et al. (2022) cikkek segítségével, illetve a Fisher-algoritmust a Golatkar et al. (2020) és Mercuri et al. (2022) cikkek segítségével.

3.1. SISA

A SISA az egyik legismertebb egzakt elfeledtetési algoritmus. Nevét az angol *Sharded*, *Isolated*, *Sliced* és *Aggragated* szavak kezdőbetűinek köszönheti, amelyek az algoritmus által alkalmazott műveletekhez kapcsolódnak (magyarul *feldarabolt*, *izolált*, *szeletelt*, *aggregált*).

A SISA algoritmus egyes lépései nagy vonalakban:

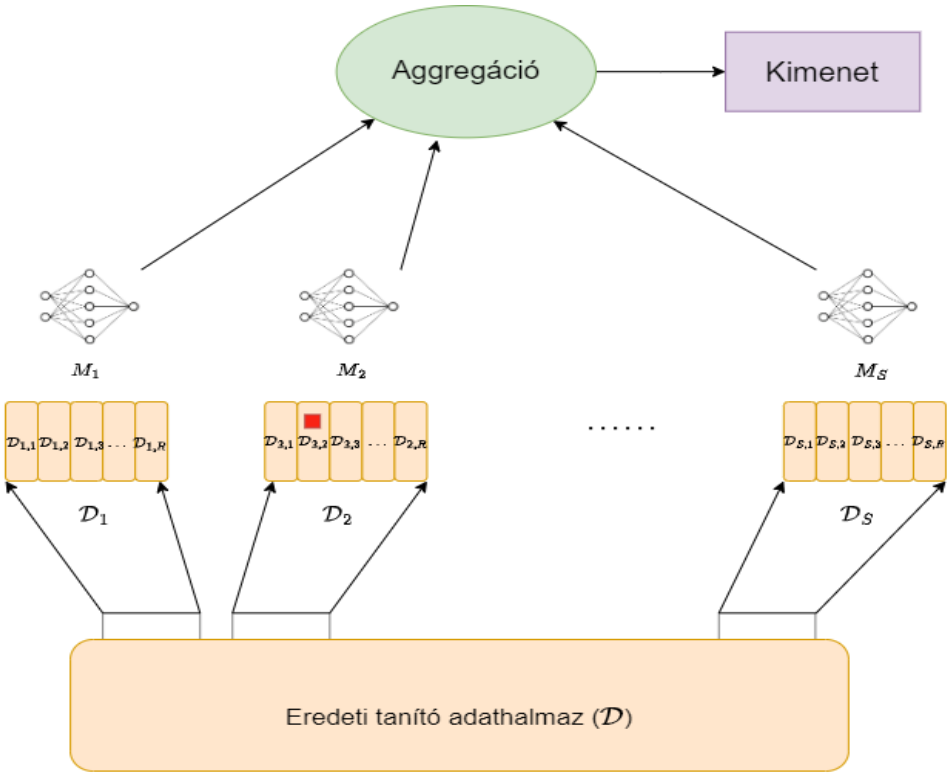
1. **Sharding (feldarabolás).** Az eredeti tanító adathalmazt megközelítőleg egyenlő nagyságú darabokra osztjuk úgy, hogy minden tanító adatpont pontosan egy darabban szerepel.

2. **Isolation (izoláció).** Minden darabot a többi darabtól izoláltan tanítunk, ezzel lekorlátozva egy adatpont hatását egyetlen adatdarabra.

3. **Slicing (szeletelés).** Minden adatdarab további szeletekre van bontva, amelyek az algoritmusnak fokozatosan vannak bemutatva a tanítás során.

4. **Aggregation (aggregáció).** Minden egyes modelldarabot összesítünk, hogy ezzel megkapjuk a végső modellpredikciót egy adatpontra. Ezt számos különböző módszerrel megtehetjük.

Amennyiben egy adatpontot el szeretnénk távolítani, csak az azt tartalmazó adatdarabot szükséges újratanítani, sőt az újratanítást elég az adatpontot tartalmazó szelettől kezdeni. Ennek köszönhetően a naiv elfeledtetéshez képest ez az algoritmus gyorsabb - a pontos sebességjavulás az adatdarabok és -szeletek számától függ.



1. Ábra. A SISA algoritmus működése. $\mathcal{M}_s$ az s-edik modelldarab, $\mathcal{D}_s$ az s-edik modelldarab, $\mathcal{D}_{s,r}$ az r-edik szelete az s-edik modelldarabnak, illetve a piros négyzet az eltávolítandó adat. Az ábrát az eredeti cikkbeli ábra mintájára készítettem.

3.1. Példa (**SISA-tanítás**). *Ahogy az 1. ábrán is látható, a $\mathcal{D}$ adathalmazt S darab darabra bondjuk, melyeknek további R darab szeletét vesszük. Minden egyes adatdarabon tanítunk egy modellt úgy, hogy az adatokat szeletenként adagoljuk tanításkor, és minden alkalommal, mielőtt új adatot adnánk hozzá, elmentjük az aktuális paramétereket.*

Amikor elfeledtetünk adatot, akkor csak azt az $\mathcal{M}_s$ modellt kell újratanítanunk, amely $\mathcal{D}_s$ adatdarabja tartalmazta az elfeledtetendő adatot, és azt ráadásul elég azoktól az elmentett paraméterektől kezdeni, amelyeket az elfeledtetendő adatot tartalmazó szelet felhasználása előtt mentettünk el.

---

1. Algoritmus Kezdeti tanítás a SISA algoritmussal

Bemenet: tanító adathalmaz (D), adatdarabok száma (S), szeletek száma (R), minden szeletre az epochok száma (e)

Kimenet: $h = (h_1, \ldots, h_S)$ modellek együttese, $\tilde{h} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^{S}$ köztes modellállapotok

Eljárás SISA TANÍTÁS (D, S, R, e)
  felosztjuk véletlenszerűen az adatokat $D_1, \ldots, D_S$ darabokra, és elmentjük minden adatpontra azon adatdarab indexét, amelyhez tartozik
  minden D_i-t tovább osztunk R szeletre (ezek $D_{i,1}, \ldots, D_{i,R}$), és elmentjük minden adatpontra azon szelet indexét, amelyhez tartozik
  véletlenszerűen inicializáljuk $(\tilde{h}_{1,0}, \ldots, \tilde{h}_{S,0})$-t
  Ciklus $i = 1; i \leq S; i++$
    Ciklus $j = 1; j \leq R; j++$
      $h_{i,j} \leftarrow$ TANÍTÁS$\Big(D_{i,1} \cup \ldots \cup D_{i,j} \mid \tilde{h}_{i,j-1}\Big)$ e_j epochon keresztül
      elmentjük a $h_{i,j}$ modell $\tilde{h}_{i,j}$ állapotát
    Ciklus vége
    $h_i \leftarrow h_{i,R}$
  Ciklus vége
  Visszatérés $h = (h_1, \ldots, h_S)$, $\tilde{h}^{\mathcal{U}} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^{S}$
Eljárás vége

---

2. Algoritmus SISA eltávolítási mechanizmus

Bemenet: $h = (h_1, \ldots, h_S)$ modellek együttese, $\tilde{h} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^{S}$ elmentett köztes állapotok, D adathalmaz az elmentett darab- és szeletindexekkel, D_u eltávolítandó adat, az adatdarabok száma (S), szeletek száma (R), minden szeletre az epochok száma (e)

Kimenet: $h^{\mathcal{U}} = (h_1^{\mathcal{U}}, \ldots, h_S^{\mathcal{U}})$ elfeledtetett modell, $\tilde{h}^{\mathcal{U}} = (\{\tilde{h}_{i,0}^{\mathcal{U}}, \ldots, \tilde{h}_{i,R}^{\mathcal{U}}\})_{i=1}^{S}$ köztes modellállapotok

Eljárás SISA Elfeledtetés $(h, \tilde{h}, D, D_u, S, R, e)$
  Ciklus $i = 1; i \leq S; i++$
    $h_{i,R}^{\mathcal{U}} \leftarrow h_i$ ▷ modell inicializálása
    $\{\tilde{h}_{i,0}^{\mathcal{U}}, \ldots, \tilde{h}_{i,R}^{\mathcal{U}}\} \leftarrow (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^{S}$ ▷ állapotok inicializálása
    Ha $\exists z \in D_u$ i darabindexszel **akkor**
      $r_i \leftarrow$ megkeressük a legkisebbb indexet minden $z \in D_u$-ra, aminek i a darabindexe
      Ciklus $j = r_i; j \leq R; j++$
        $D'_{i,j} \leftarrow D_{i,j} \backslash (D_u \cap D_{i,j})$
        $h_{i,j}^{\mathcal{U}} \leftarrow \tilde{h} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^{S}$
        $h_{i,j}^{\mathcal{U}}$-ra elmentjük a $\tilde{h}_{i,j}^{\mathcal{U}}$ modellállapotot
      Ciklus vége
    Elágazás vége
    $h_i^{\mathcal{U}} \leftarrow h_{i,R}^{\mathcal{U}}$
  Ciklus vége
  Visszatérés $h^{\mathcal{U}} = (h_1^{\mathcal{U}}, \ldots, h_S^{\mathcal{U}})_i$, $\tilde{h}^{\mathcal{U}} = (\{\tilde{h}_{i,0}^{\mathcal{U}}, \ldots, \tilde{h}_{i,R}^{\mathcal{U}}\})_{i=1}^{S}$
Eljárás vége

3.1.1. Futási idő

A SISA algoritmus négy fő műveletéből kettő során gyorsíthatunk az elfeledtetésen a naiv újratanításhoz képest: a feldaraboláskor, illetve a szeleteléskor.

Az alábbiakban tegyük fel, hogy nem tudjuk, hogy egy adott adatpont mekkora valószínűséggel lesz elfeledtetve. Ekkor a $\mathcal{D}$ adathalmazt egyenletesen particionálhatjuk S darabra úgy, hogy $\cap_{k\in[S]}\mathcal{D}_k = \varnothing$ és $\cup_{k\in[S]}\mathcal{D}_k = \mathcal{D}$. Minden $\mathcal{D}_k$-n tanítunk egy M_k modellt $\mathcal{D}_k$ minden adatpontját felhasználva. Az u felhasználó d_u adatponjta az S db adatdarab mindegyikében egyforma valószínűséggel lehet. Hogy a lehető legjobb javulást érjük el az újratanítás sebességében, feltesszük, hogy minden d_u pontosan egy adatdarabban lehet.

Ha a felhasználó azt szeretné, hogy d_u el legyen feledtetve, először is meg kell határoznunk, d_u melyik adatdarabban van (legyen ez $\mathcal{D}_u$), majd a megfelelő modellt

$\mathcal{D}_u \setminus d_u$-n teljesen újra kell tanítanunk. Ezzel kapunk egy új (M'_u) modellt. Ezzel szemben a naivan elfeledtetett modellt $\mathcal{D} \setminus d_u$-n tanítanánk újra. Mivel $|\mathcal{D}| \gg |\mathcal{D}_u|$, az újratanításhoz szükséges idő a naiv elfeledtetés esetében sokkal nagyobb. Mindez egy adatpont elfeledtetése esetén legfeljebb S-szeres várható sebességnövekedést jelent.

Az adatdarabok felszeletelésével és a paraméterek elmentésével tovább faraghatunk az elfeledtetés idején. Minden $\mathcal{D}_k$ adatdarabot R diszjunkt szeletre osztunk úgy, hogy $\cap_{i\in[R]}\mathcal{D}_{k,i} = \varnothing$ és $\cup_{i\in[R]}\mathcal{D}_{k,i} = \mathcal{D}_k$. e epochon át tanítjuk a SISA TANÍTÁS-sal, és így megkapjuk az M_k modellt. Ha az u felhasználó d_u adatpontja a $\mathcal{D}_k$ adatdarabban van, akkor annak R darab szeletének bármelyikében azonos valószínűséggel lehet.

Amennyiben a d_u-t el szeretnénk feledtetni, akkor elsőnek meg kell találnunk a szeletet, amiben elhelyezkedik ($\mathcal{D}_{k,u}$), majd az u. lépéstől kezdve újra kell tanítani $\mathcal{D}_{k,u} \setminus d_u$-n. Ezzel egy új modellt, $M'_{k,u}$-t kapjuk.

Ugyan minden epoch rövidebb ideig tart, ha csak a szeletek egy részhalmazát tanítjuk éppen, növekvő mennyiségű szelet tanítása tovább tart, hiszen minden alkalommal, amikor egy új szeletet adunk hozzá, újrakezdjük a tanítást. Azt szeretnénk elérni, hogy a szeleteléssel és nélküle is ugyanannyi ideig tartson a tanítás.

Legyen $D = \frac{N}{S}$ az egyes darabokban lévő pontok száma, ahol N az adathalmaz mérete. Legyen e' az epochok száma szeletelés nélkül. Ekkor meg szeretnénk határozni, hogy mekkora legyen az $e = \sum_{i=1}^{R} e_i$ epoch-ok száma R darab szelet betanítássa esetén, ahol e_i az $\frac{iD}{R}$ nagyságú minta betanításához szükséges epochok száma. Tegyük fel továbbá, hogy minden szelet ugyanolyan hosszú, azaz $\forall i$-re $e_i = \frac{e}{R}$. Az alábbi egyenlőséget írhatjuk ekkor fel:

$$e'D = \sum_{i=1}^{R} e_i \frac{iD}{R}.$$

Innen egyszerűsítéssel és átrendezéssel:

$$Re' = \sum_{i=1}^{R} e_i i.$$

Felhasználva, hogy $e_i = \frac{e}{R}$:

$$R^2 e' = e \sum_{i=1}^{R} i$$

$$e' = e\frac{R(R+1)}{2}\frac{1}{R^2} = \frac{e}{R}\frac{(R+1)}{2}$$

$$e = \frac{2R}{(R+1)}e'.$$

Ebből adódik, hogy egyetlen adatpont elfeledtetése esetén ez legfeljebb $\frac{R+1}{2}\times$ gyorsítást eredményez.

A darabolással együtt így a SISA algoritmus legjobb esetben $\frac{(R+1)S}{2}\times$ gyorsabb a naiv újratanításhoz képest.

3.1.2. Előnyök és hátrányok

A SISA algoritmus bármilyen fokozatosan (pl. gradiens-módszerrel) tanított modellre alkalmazható, sőt, a szeletelés művelet nélkül minden gépi tanulási modellre, például a döntési fákra is. egzaktságának köszönhetően a konzisztenciája és a tanúsíthatósága garantálva van. Ugyanakkor a Koch & Soll (2023) cikk szerint az algoritmus érzékeny a kiegyensúlyozatlan osztályméretekre. Ha kisebb elemszámú osztályok is jelen vannak az adathalmazban, akkor minél kiegyensúlyozatlanabb velük az adathalmaz, annál rosszabb lesz a pontossága a modellnek.

Az S paraméter, azaz az adatdarabok száma erősen összefügg a hatékonysággal. Amennyiben növeljük az értékét, a SISA hatékonysága növekszik, ugyanakkor a teljes gépi tanulási modellünk prediktív teljesítménye romlik. Egyszerűbb gépi tanulási feladatok esetén $S > 20$ esetén jelentős zuhanás figyelhető meg a pontosságban. Ha pedig R értékét, azaz a szeletek számát növeljük, csökkenti az újratanítás idejét, de a tárhelyigényünk nagyobb lesz, hiszen több köztes állapotot kell így eltárolnunk.

Továbbá, (Koch & Soll, 2023) alapján a SISA algoritmus tovább optimalizálható, ha tudjuk, hogy egy adott adatpont mekkora valószínűséggel lesz eltávolítva. Az *Eurobarometer* (European Commission and Directorate-General for Justice and Consumers, 2019) felmérése alapján például jelentős összefüggés figyelhető meg az emberek kora, anyagi helyzete, képzettsége, internetfelhasználási szokásai, stb. és aközött, hogy élnek-e a jogukkal, hogy töröltessék az adataikat. A nagy közösségi oldalak rendelkeznek ezen adatok jelentős részével, tehát e szempont vizsgálata is hasznos lehet a SISA implementálása során. Koch & Soll (2023) azt az összefüggést tárta fel, hogy ha először feldaraboljuk az adathalmazt - ahogy eredetileg is csinálnánk az algoritmus szerint -, majd a darabokban a szeleteket sorba rendezzük a kitörlési valószínűségeik alapján (a magasabb valószínűségű szeleteket későbbre helyezzük), akkor a kis elemszámú osztályok esetében kimutatható teljesítménybeli változások jelennek meg. Ha kisebb az ilyen osztályokhoz tartozó elemek eltávolítási valószínűsége, a teljesítmény sokkal jobb, míg ha magas az eltávolítási valószínűség, akkor rosszabb.

3.2. Fisher-elfeledtetés

A Fisher algoritmus egy közelítő elfeledtető algoritmus, amely parametrikus modellekre alkalmazható. A módszer a Newton-módszer, illetve zajok hozzáadásának ötvözése segítségével távolít el adatot egy előre betanított modellből. Pontos

felső korlátokat is tudunk adni az elfeledtetett adatokról megmaradt információ mennyiségére.

3.2.1. Alapfogalmak a Fisher-algoritmushoz

Jelölje $P(w|D)$ az $\mathcal{A}$ tanító algoritmus lehetséges kimeneteinek eloszlását, ahol $w = \mathcal{A}(D)$.

3.2. Definíció (**Tisztítófüggvény**). *Az $S(w)$ tisztítófüggvény egy sztochasztikus függvény egy gépi tanulási modell súlyain. $P(S(w)|D)$ a lehetséges súlyok eloszlását jelöli, miután a D adathalmazon alkalmaztuk először az $\mathcal{A}$ tanító algoritmust, majd az $S(w)$ tisztítófüggvényt.*

Legyen h_w egy modell, amit a $D = D_u \sqcup D_r$ adathalmazon tanítottunk. Az elfeledtetés („tisztítófolyamat") abból áll, hogy alkalmazunk egy $S(w, D_u)$ függvényt a súlyokon, hogy biztosítsuk, hogy egy „támadó" (algoritmus), amely rendelkezik a h_w modellel, nem tud kiszámítani egy $f(w)$ „olvasófüggvényt", amellyel rekonstruálhat információkat D_u-ról.

Megjegyzés. *D_u egyes tulajdonságait mindig meg tudjuk határozni anélkül, hogy valaha láttuk volna az eredeti adathalmazt. Például ha D arcképekből áll, azt biztosan tudjuk, hogy a képeken két szem valószínűleg megtalálható lesz. Az elfeledtetésben ehelyett az számít, hogy $f(w)$ mennyi további információt tud kihámozni egy D_u adathalmazból a w súlyokat kihasználva, amelyeket a komplementeréből, D_r-ből nem tudnánk meghatározni.*

3.3. Definíció (**Optimális tisztítófüggvény**). *Egy adott f olvasófüggvényre egy optimális tisztítófüggvény $S(w, D_u)$ (avagy $S(w)$), ha létezik egy másik, $S_0(w)$ függvény, amely nem függ D_u-tól, és amire*

$$\mathrm{KL}(P(f(S(w))|D) \parallel P(f(S_0(w))|D_r)) = 0. \tag{3.1}$$

Megjegyzés. *Hogy lássuk, S_0 miért nem függhet D_u-tól, vegyük az $S(w) = w$ identitást, és $S_0(w) = w'$-t, ahol w'-t a D-n való újratanításból kapjuk. Ekkor $w' \sim p(w|D)$. Ekkor a KL-divergencia 0 lesz, de nem távolít el semmi információt, mert $S(w)$ az identitás.*

Az $S_0(w)$ függvényre tekinthetünk elfeledtetési tanúsítványként, hiszen megmutatja, hogy $S(w)$ megkülönböztethetetlen egy olyan modelltől, ami sosem látta D_u-t. A 3.1 feltétel triviálisan is teljesíthető, pl. $S(w) = S_0(w) = c$ választással. Ugyanakkor nekünk az is célunk, hogy D_r-ről a lehető legtöbb információt tartsuk meg.

3.4. Állítás. *Legyen D_u egy valószínűségi változó, pl. egy véletlen mintavétel az elfeledtetendő adathalmazból. Legyen Y a tulajdonság, ami D_u-tól függ. Ekkor*

$$I(Y; f(S(w))) \leq \mathbb{E}_{D_u}[\mathrm{KL}(P(f(S(w))|D) \parallel P(f(S_0(w))|D_r))], \tag{3.2}$$

ahol $I(Y; f(S(w)))$ a Shannon-féle kölcsönös információt jelöli Y és $f(S(w)))$ között.

Ha a 3.1 egyenlet nullát ad, akkor a fenti állítás azt jelenti, hogy egy $f(w)$ olvasófüggvény kimenetéből nem tudjuk meghatározni, hogy a $w' = S(w)$ modell tanítva volt-e D_u-n vagy sem. Másszóval az adattartalmazásra következtető támadások nem lesznek sikeresek. Ezt szeretnénk általánosítani, hiszen a gyakorlatban nem tudhatjuk, milyen $f(w)$ olvasófüggvényt használ a támadónk. Az alábbi lemma segítségével garantálhatjuk a biztonságot bármilyen f-re:

3.5. Lemma. *Bármilyen $f(w)$ függvényre igaz az alábbi:*

$$\mathrm{KL}(P(f(S(w))|D) \parallel P(f(S_0(w))|D_r)) \leq \mathrm{KL}(P(S(w)|D) \parallel P(S_0(w)|D_r)).$$

A fenti lemma meg az állítás bizonyítása túlmutat e szakdolgozat tertalmán, de az eredeti cikkben (Golatkar et al., 2020) megtekinthető.

Ennek köszönhetően elég az alábbi kifejezést minimalizálni:

$$KL(P(S(w)|D) \parallel P(S_0(w)|D_r)). \tag{3.3}$$

3.6. Példa (**Elfeledtetés zaj hozzáadásával**). *Tegyük fel, hogy a modell w súlyai korlátosak. Legyen $S(w) = S_0(w) = w + \sigma n$, ahol $n \sim \mathcal{N}(0, 1)$ a tisztító folyamat, amely normál eloszlású zajt ad a súlyokhoz. Ahogy a σ szórás növeljük, teljes elfeledtetés következik be:*

$$\mathrm{KL}(P(S(w)|D) \parallel P(S_0(w)|D_r)) \xrightarrow{\sigma \to \infty} 0.$$

Ugyan a nagy szórású zaj hozzáadása valóban segíti az elfeledtetést, ez egyúttal használhatatlanná is teszi a modellünket. Az elfeledtetés során ugyanis úgy szeretnénk eltávolítani a lehető legtöbb információt egy adathalmazról, hogy a modell pontosságát közben megtartjuk.

A fenti példában látható, hogy szükségünk van egy új mennyiségre, amely minimalizálása során a modellünk nem válik pontatlanná.

3.7. Definíció (**Lagrange-függvény**).

$$\mathcal{L} = \mathbb{E}_{S(w)}[L_{D_r}(w)] + \lambda \mathrm{KL}(P(S(w)|D) \parallel P(S_0(w)|D_r)), \tag{3.4}$$

ahol $L_{D_r}(w)$ a modell hibafüggvénye az újratanított D_r adathalmazon.

Az Lagrange-függvény első tagját minimalizálni nem olyan nehéz, ugyanakkor mély neurális hálók esetén a $P(w|D)$ meghatározása már igen.

3.2.2. A stabilitás és a lokális elfeledtetési korlát

Egy adott $\mathcal{A}(D)$ sztochasztikus tanító algoritmus esetén választhatunk egy ϵ véletlen kezdeti értéket. Ha $\mathcal{A}(D, \epsilon)$-t tekintjük, ez már az adathalmaz és a kezdeti érték egy determinisztikus függvénye.

Ekkor feltesszük az alábbiakat:

1. Az elfeledtetendő D_u egy kis része a teljes D adathalmaznak (különben jobban megérné teljesen újratanítani a modellt)

2. Az $\mathcal{A}(D, \epsilon)$ stabil, azaz ha az D és D' csak egy kicsit térnek el, a $\mathcal{A}(D, \epsilon)$ és a $\mathcal{A}(D', \epsilon)$ kimenetei közel vannak egymáshoz. Ekkor azt várjuk, hogy a 3.3 egyenletben $P(S(w)|D)$ és $P(S_0(w)|D_r)$ szintén közel vannak, ami megkönnyíti az elfeledtetést.

3.8. Állítás (**Lokális elfeledtetési korlát**). *Legyen $\mathcal{A}(D, \epsilon)$ egy tanító algoritmus ϵ kezdeti értékkel (ekkor $P(S(w)|D) = \mathbb{E}_\epsilon[P(S(w)|D, \epsilon)]$. Ekkor az alábbi korlátot kapjuk:*

$$\mathrm{KL}(P(S(w)|D) \parallel P(S_0(w)|D_r)) \leq \mathbb{E}_\epsilon\big[\mathrm{KL}(P(S(w)|D, \epsilon) \parallel P(S_0(w)|D_r, \epsilon))\big].$$

Bizonyítás. A jelölés egyszerűsítése érdekében írjuk át az egyenletet az alábbi alakra:

$$\mathrm{KL}(Q(w) \parallel R(w)) \leq \mathbb{E}_\epsilon\Big[KL(Q(w|\epsilon) \parallel R(w)|\epsilon)\Big],$$

ahol $Q(w) = P(S(w)|D)$ és $R(w) = P(S(w)|D_r)$. A bal oldalt ekkor az alábbi módon írhatjuk át:

$$\mathrm{KL}(Q(w) \parallel R(w)) = \int Q(w) \log \frac{Q(w)}{R(w)}\, dw \;= \int \mathbb{E}_\epsilon[Q(w|\epsilon)] \log \frac{\mathbb{E}_\epsilon[Q(w|\epsilon)]}{\mathbb{E}_\epsilon[P(w|\epsilon)]}\, dw \;\overset{*}{\leq}$$
$$\int \mathbb{E}_\epsilon\Big[Q(w|\epsilon) \log \frac{Q(w|\epsilon)}{P(w|\epsilon)}\Big] = \mathbb{E}_\epsilon\Big[\int Q(w|\epsilon) \log \frac{Q(w|\epsilon)}{P(w|\epsilon)}\, dw\ \Big] =$$
$$= \mathbb{E}_\epsilon[\mathrm{KL}(Q(w) \parallel P(w))],$$

ahol $(*)$-nál a logaritmusok összegére vonatkozó egyenlőtlenséget használjuk ki. □

A lokális elfeledtetési korlátban csak az elfeledtetés átlagát tekintjük egy bizonyos kezdeti értékkel, nem pedig a lehetséges kimenetek globális eloszlását a véletlen kezdeti értékekre. Az alábbi következményben láthatunk erre egy konkrét példát.

3.9. Következmény (**gaussi elfeledtetés**). *Tekintsük azt az esetet, ahol $S(w) = h(w) + n$ és $S_0 = w + n'$, ahol $n, n' \sim \mathcal{N}(0, \Sigma)$ gauss eloszlású zaj és $h(w)$ egy*

determinisztikus függvény. Mivel ϵ kezdeti értékre a $w = \mathcal{A}(D, \epsilon)$ súlyok az adatok egy determinisztikus függvénye, $P(S(w)|D, \epsilon) = \mathcal{N}(h(\mathcal{A}(D, \epsilon)), \Sigma)$ és hasonlóan $P(S_0(w)|D_r, \epsilon) = \mathcal{N}(h(\mathcal{A}(D_r, \epsilon))$. Ekkor a 3.8 állítást használva:

$$\mathrm{KL}(P(S(w)|D) \parallel P(S_0(w)|D_r)) \leq \frac{1}{2}\mathbb{E}_\epsilon\Big[(h(w) - w')^T\Sigma^{-1}(h(w) - w')\Big], \qquad (3.5)$$

ahol $w = \mathcal{A}(D, \epsilon)$ és $w' = \mathcal{A}(D_r, \epsilon)$.

A fentiek alapján tehát a $\mathrm{KL}(P(S(w)|D) \parallel P(S_0(w)|D_r))$ kifejezéshez találtunk egy sokkal egyszerűbb felső korlátot, továbbá mindez három egyszerű, de általános elfeledtetési eljárásra utal:

1. Alkalmazunk egy $h(w)$ függvényt, hogy w-t és w'-t közelebb hozzuk egymáshoz (pl. $h(w) - w'$-t minimalizáljuk a 3.5 egyenletben).

2. Hozzáadunk zajt, amely Σ kovarianciája nagy $h(w) - w'$ irányába.

3. A két eljárás együtt.

3.2.3. Az optimális négyzetes tisztítóalgoritmus

Tegyük fel, hogy kvadratikus hibafüggvényünk van, például a négyzetes hibafüggvény [5.2. egyenlet], illetve nullához közelítő tanulási rátával tanítunk, ami az alábbi folytonos gradiens módszeres optimalizálást eredményezi:

$$\mathcal{A}_t(D, \epsilon) = w_0 - (I - e^{-\eta A t})A^{-1}\nabla_w L_D(w)|_{w=w_0},$$

ahol $A = \nabla^2 L_D(w)$. Ezeket a feltételeket később lazítjuk.

3.10. Állítás (Optimális tisztítóalgoritmus). *Legyen a hibafüggvény $L_D(w) = L_{D_u}(w) + L_{D_r}(w)$ és tegyük fel, hogy $L_{D_u}(w)$ és $L_{D_r}(w)$ is négyzetesek. Tegyük fel, hogy a $\mathcal{A}_t(D, \epsilon)$ optimalizáló algoritmus a t időpillanatban véletlenszerűen inicializált súlyokkal számolt hibafüggvény gradiens-folyamával. Vegyük az alábbi tisztítófüggvényt:*

$$h(w) = w + e^{-Bt}e^{At}d + e^{-Bt}(d - d_r) - d_r,$$

ahol $A = \nabla^2 L_D(w)$, $B = \nabla^2 L_{D_r}(w)$, $d = A^{-1}\nabla_w L_D$ és $d_r = B^{-1}\nabla_w L_{D_r}$. Ekkor $h(w)$-ra igaz, hogy $h(\mathcal{A}_t(D, \epsilon)) = \mathcal{A}_t(D_r, \epsilon)$ minden ϵ kezdeti értékre és t időre, továbbá $S(w) = h(w)$ eltávolítja a modellből a D_u információt:

$$\mathrm{KL}(P(S(w)|D, \epsilon) \parallel P(w|D_r, \epsilon)) = 0.$$

Megjegyzés. *Ha $t \to \infty$, azaz ha az optimalizáló algoritmus bekonvergált, egy sima Newton-módszert kapunk:*

$$S_\infty(w) = w - B^{-1}\nabla L_{D_r}(w). \qquad (3.6)$$

A 3.10 állítás megköveteli, hogy a hibafüggvényünk négyzetes legyen, ami hétköznapi esetben nem feltétlenül teljesül. Továbbá a gyakorlatban az optimalizálás diszkrét lépésekben történik a gradiens-módszerrel, és nem pedig gradiens-folyammal. A továbbiakban így ezeken a feltevéseken lazítunk a 3.9 következményben bemutatott tisztítófolyamat egy további szabadsági fokának segítségével, ami nem más, mint a zaj.

3.11. Állítás (**Robusztus tisztítófolyamat**). *Tegyük fel, hogy közel van $h(w)$ egy w'-höz egy normális eloszlású zajtól eltekintve, azaz $h(w) - w' \sim \mathcal{N}(0, \Sigma_h)$. Továbbá tegyük fel, hogy $L_{D_r}(w)$ (lokálisan) négyzetes $h(w)$ körül. Ekkor az $S(w) = h(w) + n$ alakú optimális tisztítófolyamatot (ahol $n \sim \mathcal{N}(0, \Sigma)$), ami minimalizálja a 3.7 egyenletet $\Sigma B \Sigma = \lambda \Sigma_h$ esetén kapjuk, ahol $B = \nabla^2 L_{D_r}(w)$. Ha a hiba izotropikus, azaz $\Sigma_h = \sigma_h^2 I$ az identitás többszöröse, $\Sigma = \sqrt{\lambda \sigma_h^2} B^{-\frac{1}{2}}$.*

A 3.10 és 3.11 állításokból az alábbi tisztítófolyamatot kapjuk:

$$S_t(w) = w + e^{-Bt} e^{At} d + e^{-Bt}(d - d_r) - d_r + (\lambda \sigma_h^2)^{\frac{1}{4}} B^{-\frac{1}{4}} n, \tag{3.7}$$

ahol $n \sim \mathcal{N}(0, I)$ és B, d és d_r ugyanazok, mint a 3.10 állításban. Ha $t \to \infty$ (azaz az optimalizálás majdnem bekonvergált), ez egy zajos Newton-módszerré válik, ami könnyebben alkalmazható a gyakorlatban:

$$S_t(w) = w - B^{-1} \nabla L_{D_r}(w) + (\lambda \sigma_h^2)^{\frac{1}{4}} B^{-\frac{1}{4}} \epsilon. \tag{3.8}$$

A λ itt egy hiperparaméter, amely az elfeledtetendő adatokból fennmaradó információ és a megtartandó adatokon elérendő pontosság közötti egyensúlyt szabályozza. A σ_h a gradiens-módszer folytonos gradiens-folyammal való közelítéséből eredő hibát fejezi ki.

3.2.4. Az adatok egy halmazának elfeledtetése

Miután a modellünk be lett tanítva, kijelölhetjük az elfeledtetendő adatokat a D_u halmaz megadásával. Ugyanakkor a gyakorlatban nem feltétlenül rendelkezünk D_r-rel, azaz a maradék adattal, amin a modellt tanítottuk. Ugyanakkor feltehetjük, hogy $L_D(w)$ minimumánál vagyunk, azaz $\nabla L_D(w) = 0$. Ennek köszönhetően tudjuk, hogy $\nabla L_{D_r}(w) = -\nabla L_{D_u}(w)$ és $\nabla^2 L_{D_u}(w) = \nabla^2 L_D(w) - \nabla^2 L_{D_r}(w)$. E azonosságokat használva ahelyett, hogy újraszámolnánk a gradienseket és a Hesse-mátrixot a teljes adathalmazon elég mindezt kiszámolni az elfeledtetendő adathalmazon, feltéve, hogy a $\nabla^2 L_D(w)$ Hesse-mátrixot elmentettük az eredeti D adathalmazon való tanítás végén. Ez természetesen nem kötelező lépés, csak akkor ajánlott, ha D_r-rel nem rendelkezünk a későbbiekben.

3.2.5. A Hesse-mátrix közelítése és a Fisher-elfeledtetés

A gyakorlatban egy mély neurális háló esetében túl költséges a Hesse-mátrixot kiszámolni. Általában azt sem tudjuk, hogy pozitív definit-e. Ezt orvosolandó, használható a Levenberg-Marquardt pozitív szemidefinit approximáció:

$$\nabla^2 L_D(w) \approx \mathbb{E}_{x\sim D, y\sim p(y|x)}\left[\nabla_w \log p_w(y|x) \nabla_w \log p_w(y|x)^T\right]. \tag{3.9}$$

A Hesse-mátrix ezen közelítése egybeesik a Fisher információs mátrixszal. Ezen approximáció ráadásul egzakt bizonyos problémák (pl. lineáris regresszió) esetén.

Mivel a Fisher információs mátrix túl nagy ahhoz, hogy eltároljuk a memóriában, kiszámíthatjuk az átlóját. A Fisher-algoritmust bemutató cikk szerzője ugyan a kísérleteiben nem találta az átlót B kellően jó közelítésének egy teljes Newton-lépéshez ($h(w) = w - B^{-1}\nabla L_{D_r}(w)$ a 3.8 alapján), arra azért jó, hogy a használatával zajt adjunk hozzá. Ezzel egyszerűsíthetjük a folyamatot, és így $h(w) = w$, a Fisher információs mátrix pedig a zaj. Az egyszerűsített tisztítófolyamat:

$$S(w) = w + (\lambda\sigma_h^2)^{\frac{1}{4}} F^{-\frac{1}{4}}, \tag{3.10}$$

ahol F a Fisher információs mátrix a w pontban a D_r adathalmazra, λ pedig egy hiperparaméter, amely az elfeledtetés, illetve a modell hibájának egyensúlyát állítja be.

Mivel $h(w) = w$, egy Newton-lépés helyett az eljárás azon alapszik, hogy w és w' közel vannak egymáshoz, ami pedig a sztochasztikus gradiens-módszer stabilitásától függ. Összességében ez az eljárás úgy értelmezhető, hogy zajt adunk hozzá a modellhez úgy, hogy azok a súlyok, melyek D_u-ról igen, de D_r-ról nem tartalmaztak információt, eltűnjenek.

3. Algoritmus Fisher-algoritmus

Bemenet: betanított modell θ paraméterei, D tanító adathalmaz, D_u elfeledtetendő adathalmaz, σ zajparaméter, λ paraméter a 3.10. egyenletből

Kimenet: az elfeledtetett modell $\theta^{\mathcal{U}}$ paraméterei

1: **Eljárás** FISHER-ELFELEDTETÉS (θ, D, D_u, σ)
2: $D' \leftarrow D \setminus D_u$
3: $\delta \leftarrow \nabla L(\theta^{\mathcal{U}}, D')$
4: $F \leftarrow L$ és D' a θ paraméterek Fisher információs mátrixa a D'-re nézve
5: $\theta^{\mathcal{U}} \leftarrow \theta + (\lambda\sigma_h^2)^{\frac{1}{4}} F^{-\frac{1}{4}}$
6: **Visszatérés** $\theta^{\mathcal{U}}$
7: **Eljárás vége**

4. Kísérletek

Ebben a fejezetben a Fisher-algoritmussal végzett kísérleteimet mutatom be. Ehhez egy mély mesterséges neurális hálót használtam, melyet a CIFAR-10 adathalmazon tanítottam. Az eredeti cikk szerzőinek implementációját (Golatkar, 2020) használtam fel kiindulási pontként a kísérleteim alapjául, de kiegészítettem és jelentősen leegyszerűsítettem a kódot. Az összes kísérletem reprodukálható a publikusan elérhető `https://github.com/btrxb/FisherForgetting` GitHub oldalon található kódbázisomban.

4.1. Adathalmaz, technikai részletek

A kísérletekhez a CIFAR-10 (Krizhevsky, 2009) adathalmazt használtam, amely 60000 32×32 méretű képből, és 10 képosztályból áll. Ebből 50000 kép alkotja a tanító adathalmazt, és 10000 a teszthalmazt. Minden osztályhoz ugyanannyi kép tartozik. Az osztályok címkéi sorrendben a következők: repülőgép, autó, madár, macska, őz, kutya, béka, ló, hajó, teherautó. Adataugmentálást nem használtam a modellek tanítása során, csak a képeket normalizáltam mindig a tanítóhalmazon kiszámolt színcsatornánkénti átlaggal és szórással.

2. Ábra. Az nulladik adathalmaz elemei, azaz képek repülőgépekről. Amikor a különböző méretű adathalmazok elfeledtetését hasonlítottam össze, ezen adathalmazt rögzítettem elfeledtetendőnek.

A kísérleteket a Google Colab környezetben, illetve az ELTE szervergépein, GPU-val végeztem, a PyTorch csomag (Paszke et al., 2019) használatával. A modell egy ResNet-18 modell (He et al., 2016). Az eredeti cikk (Golatkar et al., 2020) szerzői a filterek számát a felére csökkentették minden rétegben, illetve az utolsó reziduális blokkot elhagyták - én viszont a kísérleteimben egy más ResNet-18 neurális hálóval dolgoztam, amely részletes felépítése az 1. táblázatban található, továbbá a filterszámot a 0,4-szeresére csökkentettem, mert különben a hibafüggvényfüggvény nem konvergált. Egy ResNet-blokkban [2. táblázat] az egyes konvolúciós rétegek előtt egy-egy batch-normalizáló réteg található (Ioffe & Szegedy, 2015), a konvolúció után

mindig ReLu aktiváció szerepel, az utolsó lineáris rétegnél pedig softmax aktiváció. A B függelékben bővebben is kifejtem, hogy mi is a ResNet, azaz egy reziduális háló.

Réteg neve	**Kimenet mérete**	**ResNet18 blokk**	**Filterek száma**	**Stride**
Konvolúció	[-1, 64, 32, 32]		64	1
Batch-normalizáló réteg	[-1, 64, 32, 32]			
1. konvolúciós blokk	[-1, 25, 32, 32]	$\begin{bmatrix} 3\times 3 \\ 3\times 3 \end{bmatrix} \times 2$	25 25	1
2. konvolúciós blokk	[-1, 51, 16, 16]	$\begin{bmatrix} 3\times 3 \\ 3\times 3 \end{bmatrix} \times 2$	51 51	2
3. konvolúciós blokk	[-1, 102, 8, 8]	$\begin{bmatrix} 3\times 3 \\ 3\times 3 \end{bmatrix} \times 2$	102 102	2
4. konvolúciós blokk	[-1, 204, 4, 4]	$\begin{bmatrix} 3\times 3 \\ 3\times 3 \end{bmatrix} \times 2$	204 204	2
Pooling réteg	[-1, 204, 1, 1]		204	
Lineáris	[-1, 10]			

1. táblázat. A használt ResNet-18 modell felépítése.

ResNet-18 blokk
Batch-normalizáló réteg
3×3 konvolúciós réteg
Batch-normalizáló réteg
3×3 konvolúciós réteg

2. táblázat. Egy ResNet-18 konvolúciós blokk felépítése.

4.2. A kísérlet felépítése

4.2.1. A tanítás és a Fisher-algoritmus

1. A CIFAR-10-es adathalmazon betanítottam a ResNet-18 [1. táblázat] modellt a 3. táblázatban található hiperparaméterekkel. Adataugmentációt nem végeztem.

2. A modellt finomhangoltam, az eredeti 0,01-es tanulási ráta helyett 0,0001-et használva. A finomhangolás során kijelöltem az elfeledtetendő osztályt, illetve az elfeledtetni kívánt képek számát, majd véletlenszerűen kiválasztva annak

Hiperparaméterek	
Batchméret	128
Epoch-ok száma	31
Hibafüggvény	L_{CE}
Regularizációs együttható (L^2)	0,001
Tanulási ráta	0,01
Tanulási ráta finomhangoláskor	0,0001

3. táblázat. Hiperparaméterek. Az L_{CE} kereszt-entrópiát a függelékben [5.3] definiáltam.

megfelelő számú képet, lecseréltem azokat a megtartani kívánt adathalmaz szintén véletlenszerű elemeire. Tehát ennek során a modell már csak a megtartandó adatokon, azaz D_r-en tanult tovább.

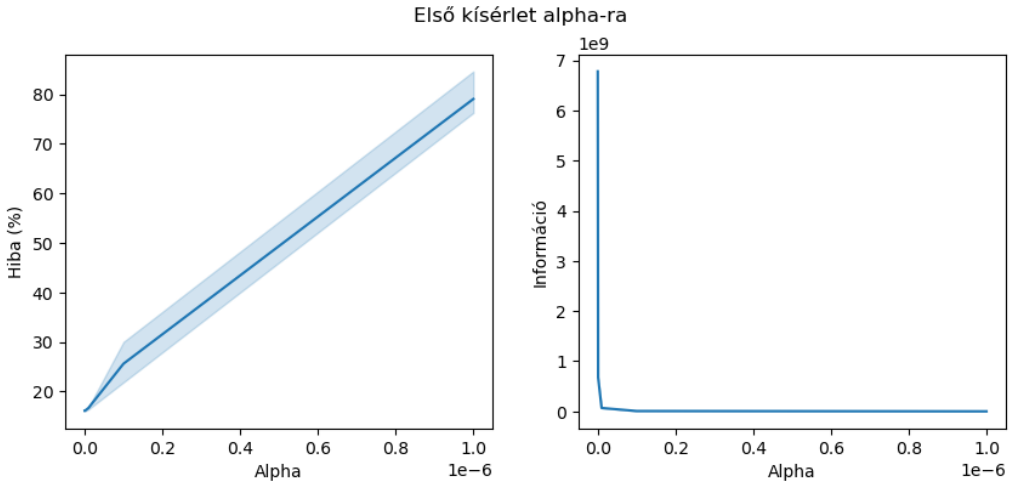
3. A 3.9 egyenletben meghatározott módszerrel előállítottam a Fisher információs mátrix diagonális approximációját a hibafüggvény paraméterek szerinti másodrendű parciális deriváltjai alapján, majd a 3.10 tisztítófüggvényt alkalmaztam a finomhangolt modell paramétereire.

4. Tanítottam egy modellt kizárólag D_r-en.

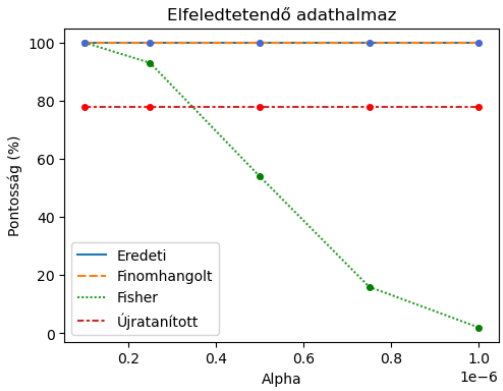
A fenti lépések elvégzése után négy modellel rendelkeztem: az eredeti teljes adathalmazon tanítottal, a finomhangolttal, a Fisher-algoritmussal elfeledtetettel és a nulláról, kizárólag a megtartandó adatokon újratanítottal. A továbbiakban leírt kísérletekben ezeket használtam fel a Fisher-algoritmus kiértékelésére.

4.2.2. Az optimális paraméter megkeresése

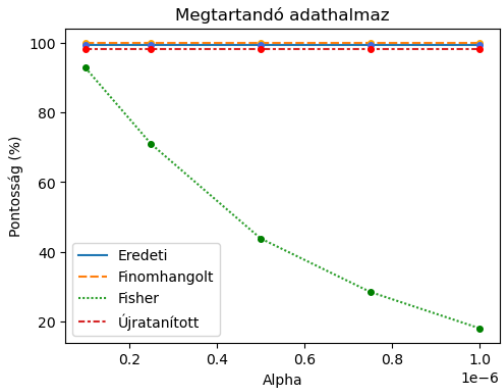
A kísérletek során kiderült, hogy a $S(w) = w + (\lambda\sigma_h^2)^{\frac{1}{4}}F^{-\frac{1}{4}}$ egyenletben, azaz a tisztítófüggvényben a lambda paraméter jelentős befolyással van a Fisher-algoritmus eredményességére. Az implementációban az $\alpha = \lambda^2$ értéket változtattam, és három véletlen inicializálásból kapott eredményeket tekintettem. $\alpha = 10^{-10}$ érték esetén az elfeledtetett modell pontossága ugyan magas volt a teszthalmazon, D_u-n, azaz az elfeledtetendő adathalmazon is magas volt, vagyis a modell nem felejtette el a kijelölt adatpontokat. Ezzel szemben 10^{-6} érték esetén ugyan elfeledte az adatpontokat, a teszthalmazon és D_r-en is jelentősen megromlott a pontossága, amit az elfeledtetés során el szeretnénk kerülni. Éppen ezért különböző α értékekre kipróbáltam 100 adatpont elfeledtetését.



3. Ábra. A Fisher-algoritmussal elfeledtetett modell hibáinak aránya az α függvényében.



4. Ábra

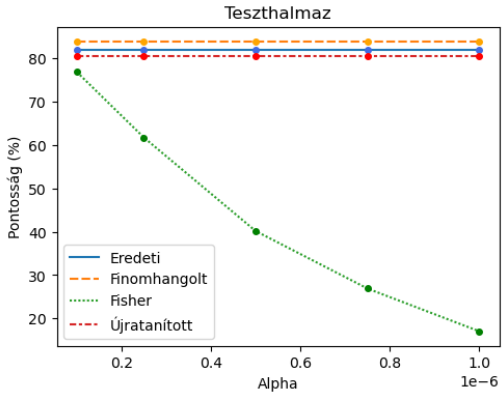


5. Ábra

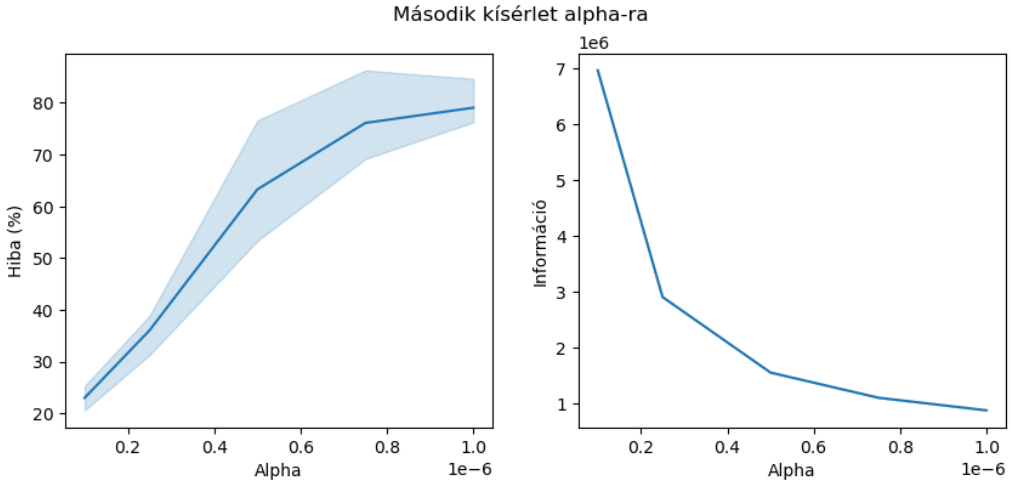
Elsőnek 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}, 10^{-10} értékeket vizsgáltam. 10^{-7} és 10^{-10} között nem változott számottevően az algoritmus eredményessége (a 3. ábrán látható a hibák aránya, illetve a súlyokban található információ mennyisége), ugyanakkor 10^{-6} és 10^{-7} között már igen, ezért tovább vizsgáltam a modelleket ezen két érték közötti α-kra. A 4., 5. és az 6. ábrákon látható a modellek teljesítménye. Ezen esetben 10^{-7}, $2,5 \cdot 10^{-7}$, $5 \cdot 10^{-7}$, $7,5 \cdot 10^{-6}$ és 10^{-6} értékekre számoltam ki a pontosságokat.

A 4., 5. és 6. ábrák alapján két α értéket tűnt érdemesnek megvizsgálni: $\alpha = 10^{-7}$ a legjobb pontosságot eredményezte a teszthalmazon, de D_u-n 100%-os volt a pontossága, ami kételyekre adhat okot, hiszen az gyanúsan magas érték egy olyan halmazon, amiről nem lenne szabad tudnia a modellnek az elfeledtetés után. A másik α érték, ami szóba jöhetett, az a $2,5 \cdot 10^{-7}$ volt, mert az az első vizsgált érték, amelyen csökkent a pontosság D_u-n.

Végül a 7. ábra döntötte el a kérdést, ugyanis azon látható, hogy $\alpha = 10^{-7}$ esetén még túl sok az információ a modellben. A továbbiakban ezért az $\alpha = 2,5 \cdot 10^{-7}$



6. Ábra



7. Ábra. A Fisher-algoritmussal elfeledtetett modell hibáinak aránya az α függvényében.

értékre végeztem a kísérleteket.

4.2.3. További kísérletek

A kísérletek során mind a 10 CIFAR osztályra kipróbáltam a Fisher-algoritmust, és ekkor az elfeledtetendő adathalmaz nagyságát 100-on rögzítettem. Megvizsgáltam továbbá azt is, hogy mi történik, ha az osztályt rögzítem (a nulladik osztályt, azaz a repülőgépeket [2. ábra]), de változtatom az elfeledtetendő adathalmaz nagyságát. Négy különböző méretre végeztem el a kísérleteket: 100, 500, 1000 és 5000 képet jelöltem ki elfeledtetésre (5000 egy teljes osztálynak felel meg). Mindezt az alábbi fejezetben következő szempontok alapján értékeltem ki.

4.3. Kiértékelés

Pontosság. A 8. ábrán látható, hogy a megtartandó adathalmazon a Fisher-algoritmussal elfeledtetett modell pontosságát nem befolyásolta sem a különböző osztályok, sem pedig a különböző D_u méretek vizsgálata. Az elfeledtetett modell szinte mindig rosszabbul teljesített a többinél, viszonylag stabil pontossági értékekkel.

Az elfeledtetendő adathalmazon a különböző osztályok esetében hullámzó volt a Fisher-algoritmussal elfeledtetett modell teljesítménye, ezen értékek alapján bizonyos osztályokat sokkal jobban elfelejtett a modell. A második, ötödik és hatodik osztályok elemeit, azaz a madarakat, az őzeket és a kutyákat sokkal kisebb arányban ismerte fel, mint a többi osztályéit. Különböző D_u méretek esetén pedig 5000-nél, azaz a teljes osztály elfeledtetése esetén meredeken lecsökkent a pontosság, ami arra utal, hogy az algoritmus teljes osztályokat jobban el tud feledtetni.

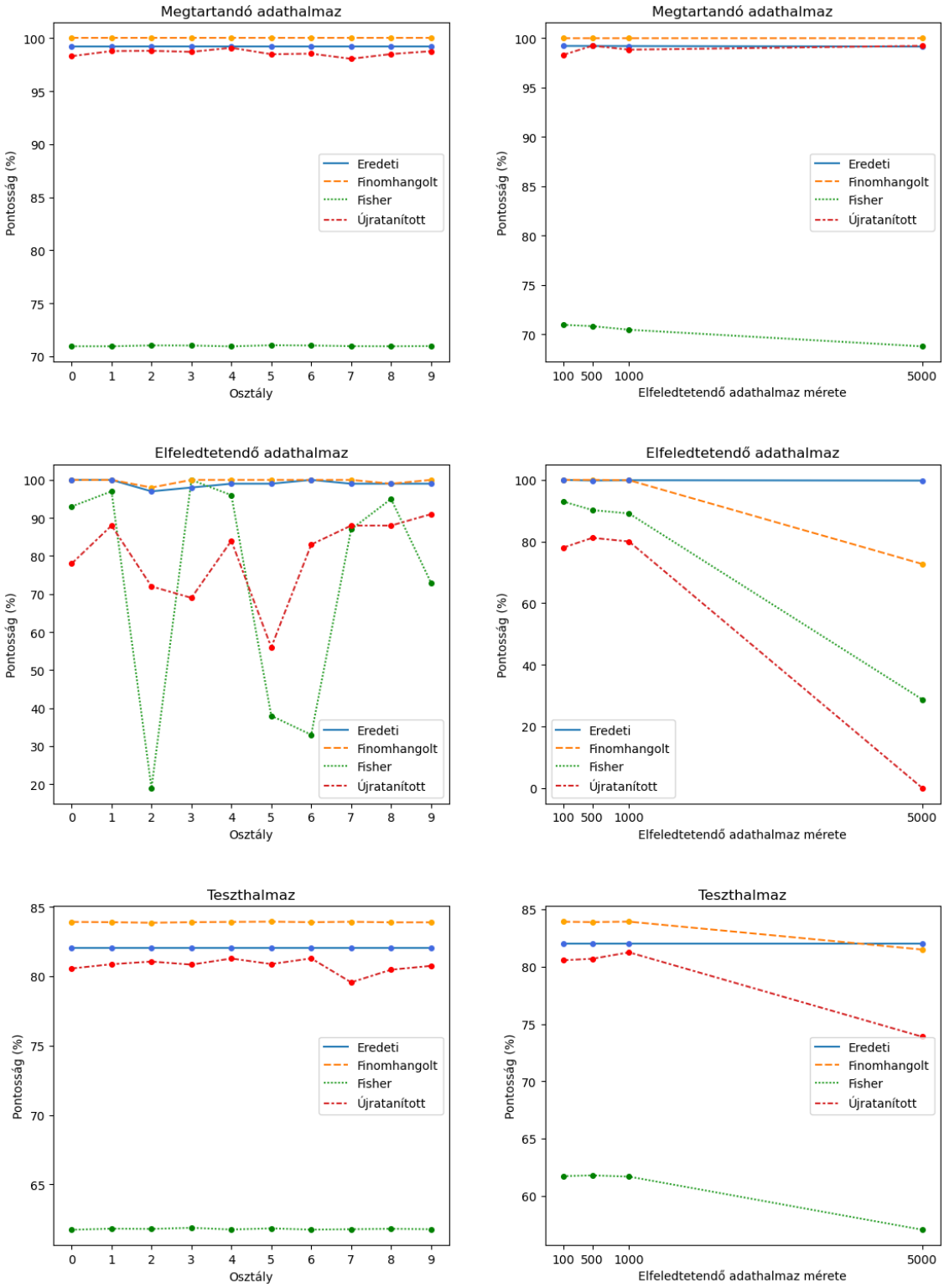
A teszthalmazon a különböző osztályok esetén az elfeledtetett modell teljesítménye szinte egyáltalán nem változott, végig vagy 20%-kal rosszabb volt a többi modellnél. A különböző méretű D_u-k esetén viszont már az is megfigyelhető, hogy a Fisher-algoritmussal elfeledtetett modellel egyetemben a finomhangolt modell teljesítménye is lecsökkent a teljes osztály elfeledtetése esetén, sőt, a várakozásainknak megfelelően a naivan újratanított modell pontossága is nagyot csökkent.

Hatékonyság. A hatékonyság méréséhez feljegyeztem a naiv újratanítás és a Fisher-elfeledtetés futásidejét minden kísérlet során, majd az osztályonkénti kísérletek esetében vettem az egyes futásidők átlagát - mivel az értékek rendkívül közel voltak egymáshoz -, és alkalmaztam rá a 2.1. képletet. Az osztályonkénti kísérletek során $\frac{781{,}115s}{10724{,}989s} = 0,0728$ értéket kaptam a hatékonyságra. A különböző méretű elfeledtetendő adathalmazok esetében külön-külön is kiszámoltam a hatékonyságot, ezen értékek a 4. táblázatban láthatóak.

$\mathbf{D_u}$ **mérete**	**Hatékonyság**
100	0,082
500	0,089
1000	0.090
5000	0.071

4. táblázat. A Fisher-algoritmus hatékonysága különböző méretű D_u-kra.

Összességében megállapítható, hogy az algoritmus egyáltalán nem hatékony.

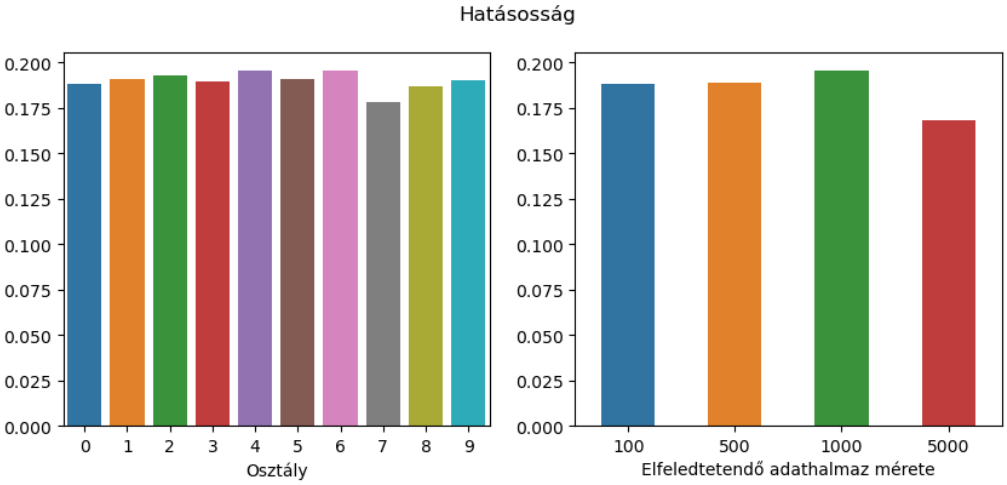


8. Ábra. A pontosság a különböző osztályokon és különböző méretű D_u-kra.

Ennek oka a Fisher információs mátrix kiszámolásának rendkívüli lassúsága: ez körülbelül három órát vett igénybe a kísérleteim során, míg a naiv újratanítás mindössze 15 percig tartott.

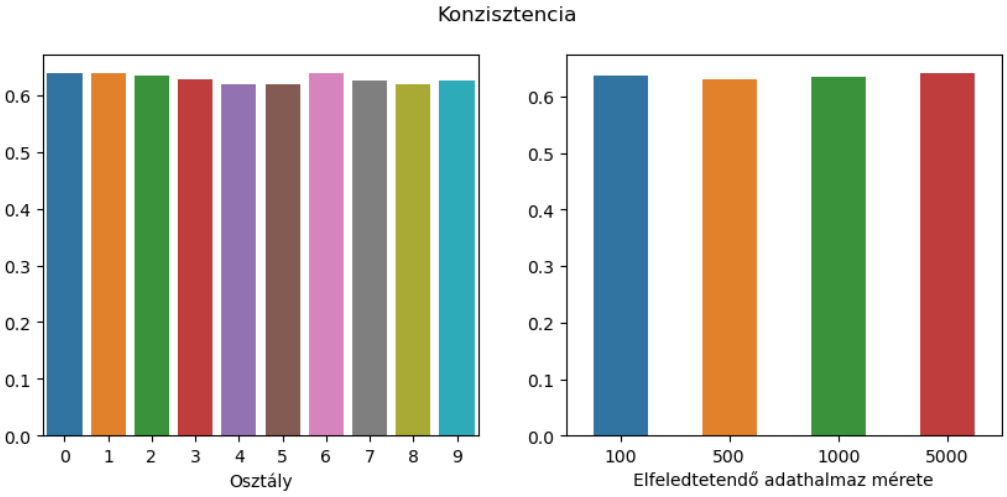
Hatásosság. A hatásosság a 2.2. képlet alapján számolható ki, azaz az elfeledtetett

modell és a naivan újratanított modell teszthalmazon vett teljesítményei különbségének abszolútértékét vettem. A 9. ábrán látható, hogy az összes kísérlet hasonló eredményt adott, azaz az értékek mind 0,19 körül mozognak.



9. Ábra. A hatásosság a különböző osztályokon és különböző méretű D_u-kra.

Konzisztencia. A konzisztencia az elfeledtetett modell és a naivan újratanított modell paramétereinek távolsága a második normában a 2.4. képlet alapján. A 10. ábrán látható, hogy az összes kísérlet kb. hasonló értékeket eredményezett, mégpedig 0,63 körülieket.

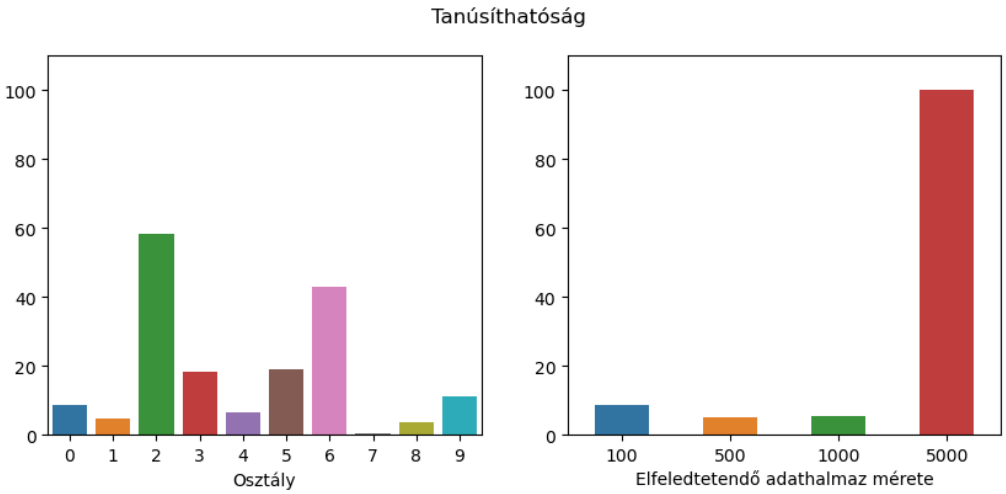


10. Ábra. A konzisztencia a különböző osztályokon és különböző méretű D_u-kra.

Tanúsíthatóság. Ezen szempont esetében nagy különbségeket figyeltem meg a kísérletek során. A 11. ábrán látható, hogy különböző osztályok elfeledtetése esetén a legmagasabb értékekkel a második osztály - amely a madarak képeit tartalmazza - rendelkezett. A tanúsíthatósági értéke kiugróan magas, 100-ból majdnem 60. Utána pedig további magas értékkel, kb. 45-tel a hatodik

osztály, azaz a békák adathalmaza következett. A macskák és a kutyák, azaz a harmadik és a hatodik osztály 20-20 körüli értékekkel majdnem holtversenyben a harmadikok a tanúsíthatósági eredmények sorrendjében. E nagyobb értékek abból fakadnak, hogy a naivan újratanított modell és az elfeledtetett modell pontosságai között nagy a különbség az elfeledtetendő adathalmazon.

A különböző méretű adathalmazokon mért tanúsíthatóság 100-tól 1000-ig alacsony, majd 5000 adatpont, azaz a teljes osztály elfeledtetése esetén felugrik 100-ra. A magas érték ebben az esetben azzal magyarázható, hogy a 2.7. képletben definiált tanúsíthatóság értéke jobban bünteti, ha az az elfeledtetendő adathalmazon a naivan újratanítottnak alacsonyabb a pontossága az elfeldtetett modellénél, hiszen ez arra utal, hogy még maradt információ a modellben a naivan újratanítotthoz képest - és egy teljes osztály elfeledtetése esetén a naivan újratanított modell teljesítménye a nulladik osztályon, azaz a repülőgépeken 0 volt.



11. Ábra. A tanúsíthatóság a különböző osztályokon és különböző méretű D_u-kra.

4.4. Tanulságok

A Fisher-algoritmus sajnos nem váltotta be a gyorsaságát illetően a hozzá fűzött reményeket. A cikk szerzői úgy fogalmaztak, hogy egy „egyszerű” Newton-lépésből áll lényegében a tisztítófolyamat, azonban ehhez ki kell számolni a súlyok Fisher információs mátrixát, amely jelentősen lelassítja a folyamatot nagy modellek esetén, mert $\mathcal{O}(n^2)$ egy Hesse-mátrix kiszámításának bonyolultsága, és ez rosszul skálázódik. Ezzel szemben ugyan a SISA algoritmust nehezebb implementálni, maga a működési elve egyszerű, ráadásul ez egy egzakt algoritmus, és biztosan tudjuk róla, hogy az eltávolítandó adat már nem lesz benne, hiszen nélküle tanul újra. A Fisher-algoritmus mellett

szól, hogy szép matematikai alapokon nyugszik, ugyanakkor futásidő szempontjából, illetve a kevés adat elfeledtetése szempontjából sem meggyőzőek az eredményei - a „machine unlearning" területen folyó kutatások célja pedig pont az lenne, hogy a naiv újratanításhoz képest csökkentsék a futásidőt úgy, hogy a pontosság is jó maradjon mindeközben.

A Fisher-algoritmus szerzői az általam elvégzett kísérletek közül a 100-as és az 5000-es elfeledtetendő adathalmaz-méreten végzetteket közölték a cikkükben. Az ő eredményeiket nem sikerült teljesen reprodukálnom, a pontosság jellemzően elmaradt mind a megtartandó-, elfeledtetendő- és teszthalmazon. Például míg egy osztály elfeledtetése esetén a szerzők kb. 96%-os pontosságot mértek a megtartandó adathalmazon, az általam végzett kísérletekben ez csak 70% körül volt. Továbbá ugyanezen osztályméreten a teszthalmazon ők 74%-os pontosságot, én pedig mindössze 55%-ot tapasztaltam. Ennek az lehet az oka, hogy a szerzők által használt modell kissé eltér az általam alkalmazott ResNet modelltől: a szerzők a filterek számát a felére csökkentették, illetve az utolsó reziduális blokkot elhagyták.

Mindezen felül az eredeti cikkhez képest számos más új kísérletet is elvégeztem, amelyek eredményei hozzájárulhatnak az algoritmus jobb megértéséhez: megvizsgáltam különböző osztályok esetén, illetve 500-as és 1000-es D_u méretekre is, hogy hogyan változnak a pontosságok. Ezen eseteket kiértékeltem emellett a Mercuri et al. (2022) cikkben található szempontok szerint is, amelyeknek hála még több információnk van az algoritmus alkalmazhatóságáról.

5. Összefoglaló

Az adatok elfeledtetése neurális hálókban egy rendkívül fiatal tudományterület, folyamatosan jelennek meg új cikkek egyre hatékonyabb módszerekkel, és ahogy a mesterséges intelligencia szabályozása egyre kiforrottabbá - és várhatóan szigorúbbá - válik, úgy lesz egyre nagyobb az igény is a további kutatásokra a témában.

Az egyik rendkívül szembetűnő és megoldásra váró probléma jelenleg az egységes kiértékelési szempontok hiánya, ez cikkenként elég változó, és meg is nehezíti így a különböző algoritmusok összehasonlítását. Ez várhatóan javulni fog, különösen ahogy a különböző hatóságok, állami szervek pontosabb követelményeket fogalmaznak meg az adatbiztonság és az adatok eltávolíttatásához fűződő jogok kapcsán. Ami pedig a vizsgált algoritmusokat illeti, SISA algoritmussal kapcsolatban az egyik legnagyobb gond, hogy az interneten elérhető publikus implementációi nemigen működnek, és így rendkívül nehéz vele kísérleteket végezni - miközben széles körben alkalmazzák a praktikussága miatt. A Fisher-algorimtussal kapcsolatban pedig kiderült, hogy a Fisher információs mátrix kiszámítása jelentős javításra szorul. A kísérletekben, akárcsak az eredeti cikk szerzői is, csak diagonális approximációt használtam, mivel az egész mátrixot nem igazán lehet a jelenlegi eszközökkel ilyen hatalmas adatmennyiség esetén reális időben kiszámolni. A jövőben, ha egyszer ez jobban implementálva lenne, biztosan érdekes lenne kipróbálni a levezetett tisztítóalgoritmust, mert a diagonális a gyakorlatban jelenleg csak további zaj hozzáadására használható: a tisztítófolyamatot magát muszáj volt rendkívül leegyszerűsíteni a jobb eredmények érdekében.

A Fisher-algoritmust még számos további kísérletben lehetne vizsgálni. Számítási kapacitás hiányában nem tudtam sajnos minden osztályra kipróbálni, hogy mi történne, ha a teljes osztályt elfeledtetnénk - ezek eredményeit összehasonlítva feltehetően megjelennének olyan összefüggések, hogy ha hasonló képosztályokat feledtetünk el teljesen (pl. kutyákat és macskákat), akkor az elfeledtetett modell teljesítménye kevésbé romlik, ha ezen osztályok egyike van csak elfeledtetve. Innen következik továbbá a kérdés, hogy mi történik, ha nem csak egy, hanem kettő vagy több osztályt feledtetünk el. Mindezen felül adataugmentálással is ki lehetne próbálni a Fisher-algoritmust, akárcsak teljesen új, még az eredeti cikkben sem vizsgált adathalmazokon és modelleken.

Irodalomjegyzék

Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. *CoRR*, abs/1912.03817, 2019. URL `http://arxiv.org/abs/1912.03817`.

Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. When machine unlearning jeopardizes privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 896–911, 2021.

European Commission and Directorate-General for Justice and Consumers. *The General Data Protection Regulation : report.* Publications Office, 2019. doi: doi/10.2838/579882.

Shashwat Goel, Ameya Prabhu, and Ponnurangam Kumaraguru. Evaluating inexact unlearning requires revisiting forgetting. *arXiv preprint arXiv:2201.06640*, 2022.

Aditya Golatkar. Selective forgetting, 2020. `https://github.com/AdityaGolatkar/SelectiveForgetting`.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. Certified data removal from machine learning models. *CoRR*, abs/1911.03030, 2019. URL `http://arxiv.org/abs/1911.03030`.

Tao Guo, Song Guo, Jiewei Zhang, Wenchao Xu, and Junxiao Wang. Vertical machine unlearning: Selectively removing sensitive information from latent feature space. *arXiv preprint arXiv:2202.13295*, 2022.

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. *arXiv preprint arXiv:2206.07758*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR '16, pages 770–778. IEEE, June 2016. doi: 10.1109/CVPR.2016.90. URL http://ieeexplore.ieee.org/document/7780459.

Yingzhe He, Guozhu Meng, Kai Chen, Jinwen He, and Xingbo Hu. Deepobliviate: a powerful charm for erasing data residual memory in deep neural networks. *arXiv preprint arXiv:2105.06209*, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate data deletion from machine learning models. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2008–2016. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/izzo21a.html.

Matthew Jagielski, Om Thakkar, Florian Tramer, Daphne Ippolito, Katherine Lee, Nicholas Carlini, Eric Wallace, Shuang Song, Abhradeep Thakurta, Nicolas Papernot, et al. Measuring forgetting of memorized training examples. *arXiv preprint arXiv:2207.00099*, 2022.

Jeff Ullman Jure Leskovec, Anand Rajaraman. *Mining of Massive Datasets*. Cambridge University Press, 3 edition, 2019. http://www.mmds.org/.

Korbinian Koch and Marcus Soll. No matter how you slice it: Machine unlearning with SISA comes at the expense of minority classes. In *First IEEE Conference on Secure and Trustworthy Machine Learning*, 2023. URL https://openreview.net/forum?id=RBX1H-SGdT.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Bo Liu, Qiang Liu, and Peter Stone. Continual learning and private unlearning. In *Conference on Lifelong Learning Agents*, pages 243–254. PMLR, 2022.

A. Aldo Faisal Marc Peter Deisenroth and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. https://mml-book.github.io/.

Salvatore Mercuri, Raad Khraishi, Ramin Okhrati, Devesh Batra, Conor Hamill, Taha Ghasempour, and Andrew Nowlan. An introduction to machine unlearning. *arXiv preprint arXiv:2209.00939*, 2022.

Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.01.012. URL https://www.sciencedirect.com/science/article/pii/S0893608019300231.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember what you want to forget: Algorithms for machine unlearning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=pvCLqcsLJ1N.

Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A. Erdogdu, and Ross J. Anderson. Manipulating SGD with data ordering attacks. *CoRR*, abs/2104.09667, 2021. URL https://arxiv.org/abs/2104.09667.

Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning. *arXiv preprint arXiv:2111.08947*, 2021.

Yinjun Wu, Edgar Dobriban, and Susan Davidson. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, pages 10355–10366. PMLR, 2020.

Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. Arcane: An efficient architecture for exact machine unlearning. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 4006–4013. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/556. URL `https://doi.org/10.24963/ijcai.2022/556`. Main Track.

Függelék

A. A neurális hálók alapjai

i. Felépítés

Az alábbiakban a (Goodfellow et al., 2016), (Marc Peter Deisenroth & Ong, 2020) és (Jure Leskovec, 2019) könyvek segítségével összefoglalom a szakdolgozathoz szükséges alapfogalmakat.

A neurális hálók úgynevezett mesterséges „neuronok"-ból állnak, amelyek a következő módon épülnek fel: egy neuron bemenete egy $x \in \mathbb{R}^j$ vektor, és a kimenete $y = \sum_{i=1}^{j} w_i x_i + b$, ahol $w \in \mathbb{R}^j$ súlyokat jelöl, és $b \in \mathbb{R}$ egy eltolást. A kimenetre alkalmazható egy aktivációs függvény, $\sigma : \mathbb{R} \to \mathbb{R}$, amely jellemzően egy folytonos, és majdnem mindenütt differenciálható függvény. Minderre azért van szükség, mert a neurális hálókban ún. „gradient descent"-et, azaz gradiens módszert alkalmazunk.

Aktivációs függvények például lehetnek:

- szigmoid: $f(x) = \frac{1}{1+e^{-x}}$,
- TanH: $f(x) = \frac{2}{1+e^{-2x}} - 1$,
- ReLU: $f(x) = \begin{cases} 0 \text{ ha } x < 0 \\ x \text{ ha } x \geq 0 \end{cases}$,
- softmax: $\mu(x) = [\mu(x_1), \mu(x_2), \ldots, \mu(x_n)]$, ahol $x \in \mathbb{R}^n$ és $\mu(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$.

A fent leírt neuronok rétegekbe rendeződnek, és ezek összességét nevezzük neurális hálóknak. Ennek az első rétegét bemeneti rétegnek nevezzük, az utólsót kimeneti rétegnek, és köztük ún. rejtett rétegek helyezkednek el. Minden rétegnek tetszőleges számú neuronja lehet, ami természetesen hatással van a neurális háló működésére.

Ha egy réteget tekintünk, annak a kimenete egy $z \in \mathbb{R}^k$ vektor és k a neuronok száma, a réteg súlyvektorait egy $W = [w_1, \ldots, w_k]$ mátrixba rendezhetjük, és a neuronokhoz tartozó eltolásokat pedig egy $b \in \mathbb{R}^k$ eltolásvektorba. Ezekkel, és a fentebb bevezetett jelölésekkel egy réteg felírható $z = \sigma(Wx + b)$ alakban. Több (pl. N darab) réteg összekapcsolásával függvénykompozíciót kapunk, amely az alábbi módon néz ki:

$$\hat{y} = \sigma(W_N(\ldots\sigma(W_2(\sigma(W_1 x + b_1) + b_2)\ldots) + b_N). \tag{5.1}$$

Itt $y \in \mathbb{R}^m$, ahol m az utolsó réteg neuronszáma.

A függvénykompozíciót megadhatjuk egy egyszerűbb alakban is:

$$\hat{y} = f_\theta(x) = f_N \circ f_{N-1} \circ \ldots \circ f_2 \circ f_1(x),$$
$$\text{ahol } \theta = (W_1, b_1, \ldots, W_N, b_N) \text{ a paraméterek.}$$

Két vagy több réteget összekapcsolva ugyanúgy lineáris lesz az eredmény, ezért is van szükség aktivációs függvényekre, amelyek megszűntetik a linearitást.

ii. A rétegek típusai

Sűrűn kapcsolt réteg. Ha egy adott réteg bemenetként megkapja az előző réteg összes kimenetét, ezt sűrűn kapcsolt rétegnek nevezzük. Megtehetjük egyébként azt is, hogy egy rétegen belül véletlenszerűen választunk ki minden neuronhoz pontosan m darab bemenetet az előző rétegből valamilyen random m-re.

Konvolúciós réteg. Ezek esetében a neuronokra gondolhatunk úgy, hogy rácsszerűen helyezkednek el, és például 2 dimenzió esetén a rács (i, j) koordinátájának megfelelő helyén a neuron bemenete az előző réteg (i, j) koordinátájának egy kis környezetéből származó értékei. Konvolúciós neurális hálókat a képfelismerésben szokás használni, mivel kihasználják a képek lokális struktúráját, illetve jelentős paraméterszám-csökkenést eredményeznek.

Batch-normalizáló réteg. A batch-normalizáló rétegek úgy működnek, hogy először a bemenetre batchenként kiszámolják a batch átlagát és szórását, ezekkel normalizálják őket, majd elemenként beszorozzák a kapott normalizált vektorokat egy γ számmal és hozzáadnak egy β értéket. A γ és β értékek nem hiperparaméterek - ezeket a modell megtanulja, tehát így be tudja magának állítani a lehető legoptimálisabb értékeket.

Pooling réteg. Ezek lényege, hogy lecsökkentik a rétegek dimenzióját úgy, hogy a filter által lefedett értékekből kiszámolnak egyetlen számot, amivel ezt az összes számot helyettesítik. Ez lehet például az összes érték átlaga vagy maximuma.

iii. Tanítás

Hibafüggvény

Amikor neurális hálókat használunk, tipikusan adott egy $\{(x_i, y_i)\}_{i=1}^N \in \mathbb{R}^n \times \mathbb{R}^m$ párokat tartalmazó adathalmaz, ahol x_i a bemenet, y_i az elvárt kimenet, és ezekre szeretnénk függvényt illeszteni, amelyekkel a legjobban tudjuk $\hat{y}_i$-vel y_i-t közelíteni bármely i-re. A bemeneti adathalmazt felosztjuk egy tanító, egy validációs és egy

teszthalmazra. A validációs halmaz arra szolgál, hogy finomhangoljuk a modell paramétereit, míg a teszthalmaz már a végleges modell kiértékelésére való. A modell pontossága a helyesen eltalált címkék aránya.

Ahhoz, hogy a modell teljesítményét mérhessük a megfelelő halmazokon, definiálunk egy hibafüggvényt, amellyel azt szeretnénk mérni, hogy mennyire tér el a kapott kimenet a tanító halmazból származó kívánt kimenettől. Ez a következő módon néz ki: $L : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$.

Regressziós probléma A legkisebb négyzetes eltérésre vagyunk kíváncsiak. Ha a tanító adathalmazból származó kimenet y_i, és a modellből kapott kimenet $\hat{y}_i$, akkor a hibafüggvény a teljes adathalmazra:

$$L(y_i, \hat{y}_i) = (\hat{y}_i - y_i)^2. \tag{5.2}$$

Klasszifikációs probléma A bemenetet n db osztályba szeretnénk sorolni. Ekkor (x_i, p) párokat keresünk, ahol $p = [p_1, p_2, \ldots, p_n]$. A p_k annak a valószínűségét adja meg, hogy az x_i bemenet mekkora valószínűséggel tartozik az k-adik osztályba. Ha q a modellből kapott kimenet, akkor egy lehetséges hibafüggvény például a KL-divergencia:

$$\mathrm{KL}(p||q) = \sum_{k=1}^{n} p_k \log \frac{p_k}{q_k}.$$

Egy másik lehetőség a keresztentrópia, amelyet a Fisher-algoritmusos kísérleteimben is használtam:

$$L_{CE} = -\sum_{k=1}^{n} y_{o,k} \log(p_{o,k}), \tag{5.3}$$

ahol $y_{o,k}$ egy indikátor változó, amely akkor 1, ha az o elem k-edik osztályba tartozik, és $p_{o,k}$ a jósolt valószínűség, hogy az o elem az k-edik osztályba tartozik.

A gradiens módszer

A gradiens módszer egy optimalizáló algoritmus, amelyben egy függvény lokális minimumát keressük a gradiense segítségével. A neurális hálók esetében ez azt jelenti, hogy a hibafüggvényt, amit definiálunk hozzájuk, minimalizálni szeretnénk. Ehhez a gradiens-ereszkedés módszert használjuk tipikusan, és a negatív gradiensek irányába módosítjuk a modell paramétereit iteratívan. A gradiensek kiszámolásához a hiba-visszaterjesztés módszert használhatjuk, melynek lényege, hogy a hibafüggvény

gradiensét kiszámoljuk a kimenettől visszafelé rétegenként a bemenetig a láncszabály segítségével.

Ha θ egy paramétervektor, akkor a kiszámolt gradiensével ellentétes irányba, azaz a legnagyobb csökkenés irányába mozdulunk el vele: $\theta \leftarrow \theta - \eta g(\theta)$, ahol $g(\theta)$ a θ gradiense. Az η a tanulási ráta, amely egy paraméter. Ennek segítségével meghatározzuk a lokális minimum felé tett lépéseink nagyságát. Ha η túl kicsi, akkor nagyon lassan fog konvergálni, ha túl nagy, akkor oszcillálás is előfordulhat. Az algoritmus akkor áll le, ha a hibafüggvény nem csökken jelentősen az egymást követő lépések során (azaz ha a lokális minimumban vagyunk), vagy ha elértünk egy fix számú iterációt.

A sztochasztikus gradiens módszer

A gradiensek kiszámolása nagy neurális háló esetén, sok adaton igen számításigényes lehet. Emiatt szeretnénk, ha közelítőleg is elég lenne meghatározni a gradienst. Ehhez a bemenetet véletlenszerűen ún. batch-ekre osztjuk, amelyeknek meghatározott mérete van. A gradienst a bemenet ezen részhalmazán számítjuk csak ki. Ez azért elég, mert ahhoz, hogy a gradiens-módszer konvergáljon, elég, hogy legyen egy torzítatlan közelítésünk a tényleges gradiensről. A kicsi „mini-batchek"-kel nagyon gyorsan lehet számolni, és általában nem az a célunk, hogy a pontos minimumot megtaláljuk - tipikusan nem is garantált, hogy a globális minimum megtalálható.

Regularizáció

A tanítás során felléphet az a probléma, hogy a modellünk a tanító halmazon túl jó eredményeket ér el, miközben a teszthalmazon sokat hibázik, azaz ha a modellünket alkalmazni szeretnénk, nem kapunk kellően pontos eredményeket, mert tetszőleges bemeneten nem teljesít jól. Ezt nevezzük túltanulásnak. Hogy ezt megelőzzük, regularizációt alkalmazunk, ami azt jelenti, hogy módosítjuk a tanító algoritmust úgy, hogy a teszthalmazon csökkentsük a hibák mennyiségét anélkül, hogy a tanító halmazon továbbra is jól teljesítsen a modellünk.

Legyen L egy hibafüggvény, és ehhez adjunk hozzá egy $\Omega(\theta)$ paraméternorma-hibatagot, ami bünteti ezen norma növekedését. Ekkor egy új hibafüggvényt kapunk:

$$\hat{L}(y, \hat{y}) = L(y, \hat{y}) + \alpha\Omega(\theta), \tag{5.4}$$

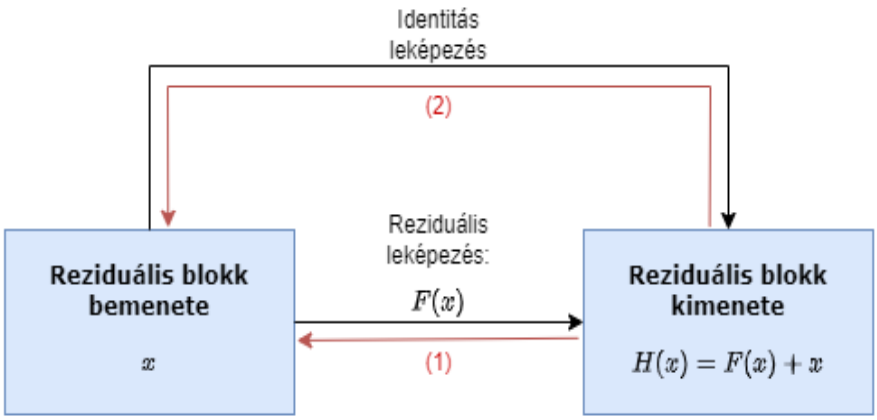
ahol $\alpha \in [0, \infty)$ hiperparaméter, amely szabályozza, hogy az Ω tag mennyit módosít a hibafüggvényen.

Az Ω tag lehet például az L^2, azaz az euklideszi norma ($\Omega(\theta) = \frac{1}{2}||\theta||_2^2$), vagy az L^1 norma ($\Omega(\theta) = ||\theta||_1$).

B. Reziduális hálók

A klasszikus előrecsatolt modellek (azaz az egyszerűen egymás után rakott rétegekből álló modellek) pontossága romlik, ha a háló mérete jelentősen megnő. Ez azért történik, mert a hiba-visszaterjesztés során a gradiens egyre csökken, és mire eléri a bemeneti réteget, olyan kicsi lesz a nagyon mély hálók esetén, hogy nem történik érdemi változás a kezdeti rétegek súlyain - a tanulás így nem hatékony. A reziduális hálók ezt a problémát orvosolják.

A reziduális hálók ugyanúgy rétegekből állnak, mint az egyszerűbb társaik, ugyanakkor ezek ún. reziduális blokkokba rendeződnek. A blokkok bemenete és kimenete egyrészt össze vann kötve a klasszikus rétegeken keresztül, ezt a szakaszt nevezzük reziduális leképezésnek, és $F(x)$-szel jelöljük, másrészt meg van egy identitás leképezés is köztük, amely a bemeneti x vektort egyszerűen továbbadja a kimenetnek úgy, ahogy van. A blokk kimenete $H(x) = F(x) + x$.



12. Ábra

A tanítás során az F(x) reziduális leképezést megtanulja a modell, majd a hiba-visszaterjesztés az alábbi módon zajlik:

1. A gradienseket visszaterjesztjük az (1) úton. Tekintsük a következő függvényt:

$$y = F(x, W_i) + x,$$

   ahol x a reziduális blokk bemenete, y a kimenete, és W_i a blokkban megtalálható rétegek súlymátrixai. Ekkor $H(x) = \sigma(F(y))$, ahol σ a ReLU aktivációs függvény.

2. A gradienseket visszaterjesztjük a (2) úton, és így az semmilyen súlyréteggel nem találkozik. A gradiensek így nem tűnnek el, azaz eljutnak az első rétegekhez is.