Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Hálózati Rendszerek és Szolgáltatások Tanszék

Biztonságos Aggregálás Federált Gépi Tanulásban Súlyozott Kliensekkel

Készítette

Jakab Máté

Konzulensek

Dr. Ács Gergely;

Horváth Máté;

Dr. Pejó Balázs

2020

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2020. 12. 10.

.....

Jakab Máté

1. Összefoglaló

A gépi tanulás egyik népszerű területe a federált tanulás, ahol több résztvevő dolgozik össze a saját erőforrásaikat felhasználva azért, hogy egy közös modellt építsenek egy központi szerver segítségével. A federált tanulás rendkívül sok adatvédelmi problémát rejt magában, számos támadó modell létezhet adott problémától függően, amely ellen védekezni kell. Jelen dokumentumban az egyik támadó modell ellen dolgoztunk egy olyan védekezési mechanizmust, amely lehetőséget biztosít a rosszindulatú résztvevők kiszűrésére úgy, hogy bárki megismerné az egyes résztvevők modellfrissítéseit vagy a federált tanulás során számított közös modellt.

A tanítás során a résztvevők a saját adatbázisukat használják fel, így a körönként beküldött modellfrissítéseikből a rosszindulatú támadók visszafejthetnének az egyes résztvevők adatbázisára vonatkozó értékes mintázatokat. Jelen dokumentumban ismertettünk egy olyan titkosítási koncepciót, amely erre a biztonsági problémára nyújt megoldást. A koncepció az ElGamal titkosítás additívan homomorf változatán alapszik, a titkosítási műveletek biztonságosságát az is tovább növeli, hogy egy biztonságosnak számító elliptikus görbén (prime192v2) számolunk.

Amennyiben egy résztvevő adatbázisa szándékosan manipulálva van, akkor ezzel ronthatja a federált tanulás által kiszámolt közös modell pontosságát, ezért fontos védekezni ez ellen is. Jelen dokumentumban egy kliensekre vonatkozó teljesítménymutató (modell pontosság) alapján súlyoztuk a klienseket, amellyel nagy mértékben csökkentettük a támadás hatását.

A federált tanulás szimulációjához az MNIST kézzel írott számokból álló adatbázisát használtuk fel, ezt daraboltuk fel azonos méretű részekre, ezeket a részeket kapták meg az egyes résztvevők. Megvizsgáltuk a titkosítási koncepciót futásidő szempontjából. Szimuláltuk a rosszindulatú támadókat is az adatbázis kimeneti értékeinek tanulás előtti manipulálásával, ezeknek a kiszűrése sikeres volt a súlyozó algoritmus segítségével. A súlyozási algoritmus szerint számolt súlyokat összehasonlítottuk a résztvevők Shapley-értékeivel.

2. Abstract

One popular area of machine learning is federated learning, where multiple participants work together using their own resources to build a common model using a central server. Federated learning involves an enormous number of privacy issues, and there may be several offensive models depending on the specific issue you need to defend against. In this paper, we have developed a defense mechanism against one of the offensive models that provides an opportunity to screen out malicious participants so that anyone can learn about the model updates for each participant or the common model calculated during federated learning.

During training, participants use their own database, so that malicious attackers could decipher valuable patterns for each participant's database from their model updates submitted per turn. In this document, we have described an encryption concept that addresses this security issue. The concept is based on an additively homomorphic version of El-Gamal encryption, and the security of encryption operations is further enhanced by computing an elliptical curve (prime192v2) that is considered secure.

If a participant’s database is intentionally manipulated, it can degrade the accuracy of the common model calculated by federated learning, so it is important to guard against this as well. In this document, we weighted clients based on a client performance indicator (model accuracy), which greatly reduced the impact of the attack.

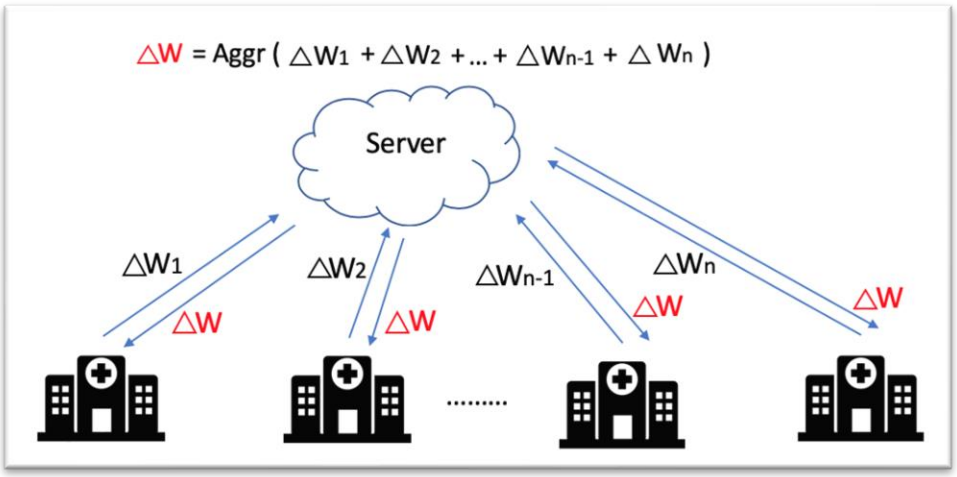
To simulate federated learning, we used the MNIST database of handwritten numbers, which was divided into equal-sized parts, which were received by each participant. We examined the encryption concept in terms of runtime. We also simulated malicious attackers by manipulating the database output values before learning, which were successfully filtered out using the weighting algorithm. Weights calculated according to the weighting algorithm were compared with the Shapley values of the participants.

3. Bevezetés

Adott cégek rendelkeznek adatbázisokkal, és ezeket az adatbázisokat felhasználva be tudnak tanítani egy modellt arra, hogy egy adott kérdésben dönteni tudjon. Például gyógyszergyártó cég rendelkezhetnek olyan adatbázisokkal, amely adott egészségügyi paraméterű emberek gyógyszerezéséről és ennek hatékonyságáról tárol információt. Ezeket az adatbázisokat felhasználva be tudnak tanítani egy neurális hálót, amelynek kimeneteként olyan modellt kapnak, amely pontosabb ajánlást tud adni arról, hogy adott paraméterek alapján mekkora mennyiségű hatóanyagot ajánlott egy adott ember szervezetébe juttatni a gyógyulás érdekében.

Viszont nem minden esetben rendelkeznek a cégek elegendő méretű vagy hasznosságú adatbázissal, és ekkor felmerül az igény arra, hogy más cégek adatbázisait implicit felhasználják. Erre nyújt megoldást a federált tanulás, ahol a résztvevők körönként járulhatnak hozzá ahhoz, hogy az általuk közösen fejlesztett modell egyre pontosabb legyen.

A federált tanulás lehetőséget biztosít arra, hogy ugyanazt a feladatot neurális háló használatával megoldani kívánó résztvevők (kliensek) egymás segítségét kihasználják anélkül, hogy a bizalmas adataikat egymással vagy bárki mással megosztanák. Ezzel a módszerrel egy adott feladatra specifikált modellt közösen tudnak fejleszteni erőforráshatékonyan egy központi szerver segítségével. Ezt a felépítést mutatja be az 1. ábra is.



1. ábra – Federált tanulás felépítése; alul a kliensek, felül a szerver van megjelenítve [1]

Könnyen belátható, hogy ez előrelépést jelentene, ugyanis több adat nagyobb rálátást adna egy adott kérdésben, pontosabb kimeneteket kaphatnának. Viszont a cégek által –

akár évek alatt összegyűjtött – adat rendkívül nagy értékű, nem lenne célszerű, ha ezt az üzleti előnyt elengednék, és a szellemi tulajdonukat kiadnák. Emellett az adatok publikálása sértene adatvédelmi szempontokat is, mint például az általános adatvédelmi rendeletet, azaz a GDPR-t [2]. Továbbá az adatok nagy mérete és az erőforrások limitjei miatt az a kivitelezhető, ha a cégek kizárólag a saját adataikkal számolnak.

Jelen dokumentumban a gépi tanulás adatvédelmi sérülékenységeit vizsgáltuk meg egy népszerű szegmensén, a federált tanulás témakörén belül. Célunk az, hogy egy olyan biztonságos federált tanulást valósítsunk meg, amely képes arra, hogy súlyozással csökkentse a hatását a federált tanulásban résztvevő rosszindulatú szereplőknek. A biztonságosság alapja a központi szerver által történő biztonságos aggregálás. Ehhez egy olyan titkosítási koncepciót mutatunk be, melynek segítségével a federált tanulásban résztvevők információi, modelljei nem kompromittálódnak.

Belátható, hogy az elsődleges feladat, amit el kell kerülnünk, az az, hogy bármilyen adatot szivárogtassanak az egyes résztvevők. Federált tanulás esetén a közös modell építésekor az adatbázisukat nem, a modelljüket meg kell osztaniuk a résztvevőknek ahhoz, hogy a modellek aggregálásával kiszámíthassuk a közös modellt. Mivel maga a modell is szivárogtat ki adatot, ezért ezeket a modelleket titkosítás nélkül adatvédelmi szempontból nem biztonságos megosztani. Ahhoz, hogy lássuk, hogy ez mekkora kockázatot jelent, az alábbi cikkekben bővebben olvashatunk az erre alapozó támadásokról: [3], [4]. A modellszivárogtatás mértékét az alábbi könyvtár segítségével is mérhetjük: [5]

A modell általi adatszivárogtatás megszüntetése miatt kerestünk titkosítási koncepciót, melyhez felhasználtunk egy módosított ElGamal titkosítási algoritmust, amelynek titkosítási és dekódolási műveletei egy elliptikus görbén kerülnek kiszámításra. A megoldásunkban törekedtünk arra, hogy a federált tanuláshoz szükséges további kommunikáció is teljes mértékben biztonságos legyen, erről részletesebben a 8.2 alfejezetben írtunk.

A federált tanulás során a kliensek alapvetően ugyanolyan arányban vesznek részt a központi modell frissítésében. Viszont a kliensek között szerepelhet olyan, aki nem tanítja a modellt, vagy akár el is rontja. A modell elrontásának oka lehet az, hogy egy kliens rossz minőségű adattal rendelkezik, de egy rosszindulatú támadás is ezt eredményezheti. Az emiatt történő torzítás érdekében a szervernek valamilyen teljesítménymutató alapján súlyoznia kell a klienseket, és ezek alapján kell kiszámolnia, hogy mi legyen a központi modell. A későbbiekben ismertetett súlyozási algoritmus használatával a rosszindulatú

kliensek olyan súlyt kapnak, hogy azok nem (vagy csak kis mértékben) befolyásolják a modellt.

Amennyiben a súlyozás alkalmas arra, hogy kiszűrje a rosszindulatú támadókat a teljesítmény alapján, akkor arra is képes, hogy a többi klienst is rangsorolni tudja. Így a biztonság növelése mellett a hatékonyságot is növelni tudjuk. A súlyozásnál törekedtünk arra, hogy egy olyan megoldást használjunk, amely minden klienst helyesen, hatékonyságnak megfelelően súlyoz. A súlyozás helyességét egy általánosan elfogadott (igazságosnak tekinthető) metrikával, a Shapley-értékkel hasonlítottuk össze.

Az első bekezdésben ismertetett probléma két fő aspektusát bontottuk ki. Az egyik aspektus a biztonság volt. A biztonságos kommunikáció és az adatvédelem biztosítása federált tanulás esetén rendkívül fontos. Emiatt rögzítésre került, hogy ki kinek mit továbbíthat, ki milyen információt ismerhet, mi a titkosítási koncepció, ez miért biztonságos. Szó esett a javasolt kommunikációs csatornáról és a federált tanulás központi szerverével kapcsolatos elvárásainkról. A másik aspektus a súlyozás kérdése volt. A kliensek súlyozásánál bemutatásra került, hogy ennek miért van jelentősége, hogyan képes arra, hogy a rosszindulatú támadásokat kiszűrje. Összehasonlítottuk a klienseknek osztott súlyokat a Shapley-értékeikkel. A bemutatott titkosítási és súlyozási algoritmusokat a gyakorlatban is megvalósítottuk, egy egyszerű federált tanulásos példán keresztül megvizsgáltuk és méréseket végeztünk.

4. Kapcsolódó munka

A federált tanulás esetén többnyire olyan résztvevők akarnak együtt dolgozni, ahol az információmegosztás mértékének a lehető legkevesebbnek kell lennie. Elérendő cél, hogy a federált tanulás során a szellemi tulajdon kompromittálódása a lehető legkisebb mértékű legyen. Tekintve, hogy a federált tanulásnál közös modell a résztvevők modelljeit veszi alapul, ezért ez a cél elérése rendkívül nehéz feladat. Minden probléma más és más, ezért számos megoldási koncepcióval találkozhatunk, különböző szintű adatvédelemmel. A probléma megoldásának alapja általában egy titkosítási koncepció, amely segítségével elkerülhető, hogy a támadók lehallgassák a kommunikációt és megismerjék az egyes kliensek modellfrissítéseit vagy a közös modellt.

Elkülöníthetünk megoldásokat aszerint, hogy az aggregálást végző központi szerver a klienseket azonos mértékben veszi figyelembe [6], vagy valamilyen algoritmus alapján súlyozza őket [7]. A megoldásunkban a modellek hatékonyságát vizsgáltuk, de léteznek olyan megoldások is, amelyek a kliensek adatbázisának a méretét is figyelembe veszik a hatékonyság mellett [8].

A megoldásunkban súlyozás utáni átlagolással próbáljuk robusztussá tenni az aggregálást, de léteznek olyan algoritmusok is, ahol átlagolás helyett a végén egy darab véletlenszerűen kiválasztott kliens modellfrissítése szerint módosul a központi modell. Ez a megoldás felvet adatvédelmi problémákat, ugyanis abban az esetben, ha az i+1. kör kimeneteként kapott közös modell paramétereiből kivonnánk az i. kör kimeneteként kapott modell paramétereit, akkor megkapnánk az i. kör végén kiválasztott kliens modellfrissítését.

Ha a résztvevőket a federált tanulás elején nem ellenőrizzük, és nem győződünk meg arról, hogy mindenkinek az a célja, hogy a lehető legjobb legyen a közös modell, akkor szükséges detekciót alkalmaznunk annak érdekében, hogy elkerüljük a modell elrontását okozó rosszindulatú támadásokat. Az általunk bemutatott megoldásban egy teljesítménymutató alapján súlyozzuk a klienseket, és a súlyozás szerinti arányban vesznek részt a közös modellben. Egy másik hozzáállás szerint a modell rontását elkerülhetnénk azzal is, ha a rosszindulatúnak detektált klienseket „száműznénk” a federált tanulásból [9], de ekkor felmerülne annak a kockázata, hogy egy jóindulatú szereplőt tiltunk ki.

A súlyozó algoritmusunk az egyes résztvevők modelljének pontosságát használja fel a súlyozáshoz, a pontosság maximális értéke 1, tehát minél kisebb az érték, annál valószínűbb, hogy rosszindulatú támadóról van szó. Ehelyett használható a veszteségfüggvény (*loss function*) is akár, ekkor a nagyobb értéket kapó modellek lesznek a potenciális támadók.

A rosszindulatú támadók számos módszerrel próbálkozhatnak annak érdekében, hogy elrontsák a federált tanulást, a megoldásunkban egy adatmérgezéses támadást szimuláltunk, de vizsgálandó probléma lehet például a modellfrissítés mérgezése is [10]. Egy részletes összefoglalót találhatunk a federált tanulásáról lehetséges támadásokról a [11] publikációban. A federált tanulást nem csak rosszindulatú támadással lehet elronthatni, létezhetnek nem rosszindulatú problémák, például kliens által történő számítási hiba vagy szerver és kliens közötti kommunikációs hiba stb.

5. Háttértudás

A fejezet során bemutatjuk a szükséges előismereteket, rögzítjük azokat az információkat, amelyek a munkában nem lettek kiemelve, azaz egyértelműnek vesszük azokat. Az ismertetőhöz felhasználtuk az alábbi cikkeket: [12], [13], [14].

5.1. Neurális hálózat

Neurális hálózatnak nevezzük azt az információfeldolgozó modellt, amely:

- azonos vagy hasonló típusú lokális feldolgozást végző műveleti elem, neuron összekapcsolt rendszeréből áll,
- rendelkezik tanulási algoritmussal, amely általában a mintázatok segítségével történő tanulást jelenti,
- rendelkezik a megtanult információ felhasználását lehetővé tevő információ előhívási algoritmussal.

A fentiek értelmében a neurális hálózatok működésénél tipikusan két fázist különböztethetünk meg. Az első fázis, melyet tanulási fázisnak nevezünk, a hálózat kialakítására szolgál, melynek során a hálózatban valamilyen módon eltároljuk a rendelkezésre álló mintákban rejtve meglévő információt. Eredményként egy olyan információfeldolgozó modellt kapunk, melynek használatára általában a második fázisban, az előhívási fázisban kerül sor.

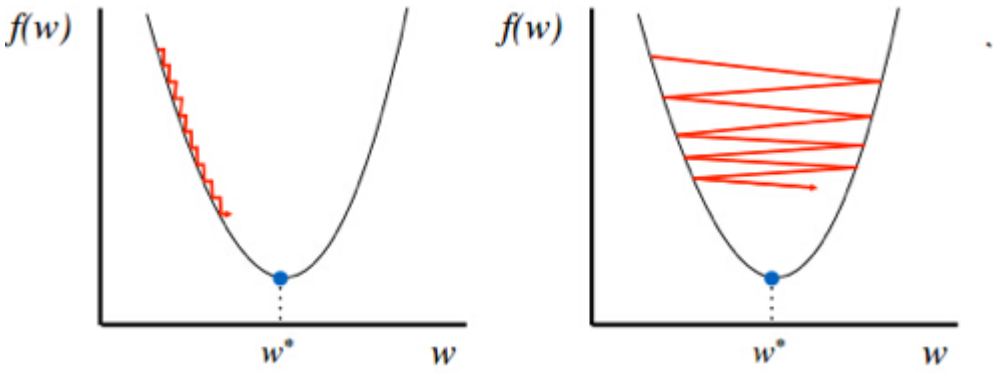
Neurális hálók tanításánál a bemeneti (és a kimeneti) minták alapján a neurális háló paramétereit módosítjuk a kívánt leképezés megvalósítása érdekében, anélkül, hogy előzetesen a problémához illeszkedő speciális modellt vagy algoritmust alkotnánk. A neurális háló építőelemei rendszerint egy véges eszköztárból választhatóak, viszont a segítségükkel alkotott modell számos problémára tud helyes, vagy elegendő mértékben pontos megoldást nyújtani. A munkánk során a neurális háló paramétereinek beállítása a használt könyvtár (Keras [15]) segítségével rejtve maradt.

A neurális háló hatékonyságát növelhetik az ún. *hiperparaméterek*, amelyek nem a használt adattól függenek, beállításukat általában a háló készítője végzi. Amiatt, hogy a vizsgált paraméterekről alkotott képet ne változtassák meg nagy mértékben a további hiperparaméterek, ezért az implementációban ezeket az értékeket rögzítettük, minden résztvevőnél azonos beállításokkal inicializáltuk a neurális hálókat.

A hiperparaméterek közé tartozik az *epochok száma,* a neurális háló tanulási fázisának ciklusainak száma. Az általunk bemutatott példákban a federált tanulás kliensei körönként egy-egy iterációt (epochot) végeztek el. Ez a tesztelés gyorsításának érdekében lett ilyen alacsony értékre beállítva a korlátozott számítási kapacitás miatt.

A neurális háló tanításához a leggyakrabban használt optimalizáló algoritmus a *gradiens süllyedés*. A gradiens a modell frissített paramétereivel egyezik meg, jelen dokumentumban a modellfrissítés fogalmat használtuk. A gradiens süllyedés egy numerikus számítás, amely lehetővé teszi számunkra, hogy megtudjuk, hogyan állítsuk be a neurális háló paramétereit úgy, hogy a kimenettől való eltérés a lehető legkisebb legyen. A gradiens süllyedés egyik fajtáját, a *mini-batch* sztochasztikus gradiens süllyedést használtuk, mely segítségével ahelyett, hogy a hálózatot az összes mintával tanítanánk, minden körben csak néhány véletlenszerű elemet használunk. Mini-batchek segítségével a tanítás kevesebb memóriát igényel, és a tanítás is gyorsabban végbemegy, mintha az összes mintát egyszerre használnánk a tanításra. Az implementációnkban ez a szám rögzítve lett, batch méretnek minden résztvevőnél 32-t állítottunk be. A (korábban említett) kimenettől való eltérést a veszteségfüggvény (*loss function*) alapján számítható ki. A legismertebb veszteségfüggvények az alábbi cikk segítségével ismerhetők meg: [16].

A sztochasztikus gradiens süllyedés egyik legfontosabb paramétere a tanulási ráta (*learning rate*). A tanulási rátával lehet meghatározni, hogy egy modellfrissítés során mekkora mértékben változhat a modellfrissítés. A legtöbb munkában ez az érték futásidő alatt változik, értéke optimalizálódik, viszont az implementációnkban ezt az értéket 0.01-re rögzítettük. A 2. ábra bemutatja, hogy a tanulási ráta helyes megválasztása mennyire fontos a szuboptimális modell megtalálása szempontjából.



2. ábra – A tanulási ráta túl alacsony (balra) és túl magas értéke esetén [17]

5.2. Shapley-érték

A játékelmélet azzal a kérdéssel foglalkozik, hogy mi a racionális viselkedés olyan helyzetekben, ahol minden résztvevő döntésének eredményét befolyásolja a többiek lehetséges választása. A játékelméletek között el kell különítenünk a kooperatív és a nem kooperatív játékokat. Kooperatív játék esetén a játékosok között kialakul egy együttműködés. Az együttműködésben résztvevő tagok igazságos értékelésére ad egy lehetőséget a Shapley-érték. Habár a Shapley-érték 1951-ben lett publikálva, a mai napig használják többek között a gépi tanulással kapcsolatos problémák esetén is (pl.: [18], [19]). A Shapley-értékről bővebben a [20] hivatkozás alatt olvashatunk.

A kooperatív játék jelen esetben a federált tanulás, a játékosok a kliensek. A kliensek Shapley-értékét kiszámolva megkaphatunk egy olyan értéket, ami alapján eldönthető, hogy egy adott kliens mekkora mértékben járul hozzá a haszonhoz a közös modell pontossága szempontjából. Azaz a Shapley-érték segítségével választ kaphatunk arra a kérdésre, hogy mennyire fontosak az átfogó együttműködés szempontjából az egyes kliensek.

5.2.1. Kiszámítása

A kliensekhez tartozó Shapley-érték kiszámításához meg kell vizsgálni a kliensek összes koalíciójához tartozó modellek pontosságát. Egy koalíció pontosságának kiszámításához vegyük a koalícióban szereplő modellek frissítéseit, ezeket aggregáljuk, majd az így kapott modell pontosságát meghatározzuk a 8.1.1 pontban említett módszerrel. Az összes koalíció száma ebben az esetben a kliensek számától függ, n kliens esetén 2^n darab ilyen koalícióval kell számolnunk. Ebbe természetesen beletartozik az üreshalmaz is, az ehhez tartozó modell legyen a szervertől kapott legelső modell, amelyen nem végeztünk tanítást!

Amennyiben a fenti értékek rendelkezésre állnak, akkor kiszámolható a kliensek pontosság szerinti Shapley-értéke. Az alábbi képlet alapján az *i* kliens Shapley-értékét kaphatjuk meg:

$$SHAP_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\ (|N| - |S| - 1)!}{|N|!} (pontosság(S \cup \{i\}) - pontosság(S))$$

A fenti képletben található *N* az összes klienst tartalmazó halmaz, az *S* változó pedig az *i* kliens nélküli koalíciókat fogja tartalmazni. A pontosság függvény megadja, hogy a paraméterben megadott koalíció mekkora pontosságú.

Például amennyiben 5 kliens esetén a 0. kliens Shapley-értékét szeretnénk kiszámolni, akkor S értéke 2^{5-1}=16-féle lehet, így a szumma is 16 tagból fog állni. Vizsgáljuk meg az egyik tagot úgy, hogy S értéke legyen az {1,2,3} koalíció! Ekkor a (pontosság({0,1,2,3}) - pontosság({1,2,3}) be van szorozva a következő súllyal: (3! · (5-3-1)!)/(5!)=1/20.

Az eredeti koncepció szerint minden koalícióra egymástól függetlenül tanítást kell végezni, a tanított modellek pontossága alapján pedig kiszámolható a kliensek Shapley-értékét. A megoldásunk során adat alapú Shapley-értéket számítunk, ami minden körben a modellfrissítések alapján kiszámolja a koalíciók aggregált modelljének pontosságait, és ebből számítja a kliensekhez tartozó Shapley-értékeket. Ez utóbbi időhatékonyabb megoldás, ugyanis nem kell 2^n tanítást elvégeznünk, a korlátozott számítási kapacitások miatt ezt a verziót használtuk.

5.2.2. Előnye és hátránya

A Shapley-érték olyan értékeket garantál, amelyre tekinthetünk igazságos elosztásként, amennyiben igazságos alatt a következő alapvető tulajdonságokat értjük:

- Hatékonyság
    - Hatékonynak tekinthető, ugyanis a kliensek Shapley-értékeinek összege megegyezik az összes kliens koalíciójának pontosságával, tehát az összes „előny” el van osztva a kliensek között. A Shapley-érték negatív értéket is kaphat, tehát az „előny” negatív is lehet.
    - $\sum_{i \epsilon N} SHAP_i = pontosság(N)$

- Szimmetria
    - A Shapley-érték szimmetrikus, mert két kliens (a és b) Shapley-értéke egyenlő, ha a $pontosság(S \cup \{a\}) = pontosság(S \cup \{b\})$ teljesül az összes olyan $S \epsilon N$ koalícióra, amelyben nem szerepel a két kliens közül egyik sem.
    - $\text{pontosság}(\text{S} \cup \{\text{a}\}) = \text{pontosság}(\text{S} \cup \{\text{b}\})\ \forall\ \text{S} \epsilon \text{N}\{\text{a}, \text{b}\} \rightarrow \text{SHAP}_\text{a} = \text{SHAP}_\text{b}$
- Linearitás
    - A Shapley-érték lineáris, ugyanis két kliens Shapley-értékének összege megegyezik a két kliens kombinációjának (összegének) Shapley-értékével.
    - $SHAP_{a+b} = SHAP_a + SHAP_b$
- Haszontalan kliens
    - A haszontalan kliens Shapley-értéke akkor és csak akkor nulla, ha a $pontosság(S \cup \{a\}) = pontosság(S)$ teljesül az összes olyan $S \epsilon N$ koalícióra, amelyben nem szerepel az a kliens.
    - $pontosság(S \cup \{a\}) = pontosság(S)\ \forall\ S \epsilon N\{a\} \leftrightarrow SHAP_a = 0$

Az előnyök mellett van egy hátulütője, mégpedig az, hogy a koalíciók száma a kliensek számától függően exponenciálisan nő, így exponenciálisan sok számításra is van szükség a Shapley-értékek kiszámításához. Emiatt a súlyozás Shapley-értékkel való értékelés a számítási kapacitás szerint korlátos. A 9.1.3 alfejezetben egy gyakorlati példa elemzésével meghatároztuk, hogy a Shapley-érték szerint igazságosnak tekinthető-e az ismertetett súlyozási algoritmus.

Az adatmérgezéses támadások elleni (súlyozás segítségével történő) védekezés koncepcióját ismertettük, a védekezési mechanizmus további részleteit a következő alfejezetben ismertetjük.

6. Támadó modell

A federált tanulás helyes és biztonságos működését megnehezíthetik a federált tanulásban résztvevő szereplők (a központi szerver és a kliensek), esetleg a kíváncsi vagy rosszindulatú külső entitások (*outsiders*). Ahhoz, hogy a jelen dokumentumban ismertetett megoldás védelmet biztosítson a fent említett potenciális támadók ellen, rögzítenünk kell néhány feltételt. A biztonságos kommunikáció kialakításához vizsgálni kell, hogy melyik résztvevő mit dekódolhat, mi számít titkos információnak, ki mit küldhet el egy másik résztvevőnek. El kell érnünk, hogy mindenki annyit tudjon, amennyit szükséges tudnia. A problémát meg kell vizsgálnunk a központi szerver, a kliensek és a külső entitások szempontjából is.

6.1. Központi szerver

A szerver őszinte, de kíváncsi (*honest-but-curious*), azaz helyesen végzi el a megoldási protokoll szerinti feladatait, de próbál további információkat is megismerni. Tehát a federált tanulás szerinti algoritmustól nem tér el, de a beérkezett adatok alapján törekszik arra, hogy további (akár privát) információkat is megtudjon.

6.2. Kliensek

Jelen dokumentumban védekezési mechanizmust nyújtunk egy olyan (rosszindulatú kliensek által történő) adatmérgezéses támadásra, amely a közös modell degradálását, pontosságának csökkenését érné el. *Rosszindulatú* támadónak tekintjük az olyan klienseket, amelyek nem nyújtanak előnyt a közös modell számára, sőt akár a közös modell elrontása a céljuk.

Ezek a kliensek lehetnek ún. *freeriderek* vagy másnéven *ghostok* (magyarra fordítva talán a *potyautas* lenne a legmegfelelőbb kifejezés), akik nem vesznek részt a közös „munkában”, nem tanítják a körök során a kapott modellt, csak saját célra felhasználják azt.

Emellett lehetnek olyan kliensek, amelyek szándékosan olyan bemeneti értékekkel tanítják a modellt, amelyekhez randomizált kimeneti értékek tartoznak (pl. a kézzel írt számjegyekhez random számjegy tartozik). Ennél is rosszabb hatással van a közös modellre az, hogyha nem randomizált, hanem minden esetben rossz kimeneti érték tartozik a bemeneti értékekhez. Ezt a támadást nevezzük adatmérgezésnek. Jelen esetben az adatmér-

gezés a rosszindulatú kliensek számával egyenesen arányos. A nagymértékű adatmérgezés során fennáll annak a veszélye, hogy a közös modell teljes mértékben elromoljon, a pontossága minimálisra csökkenjen. A 9.1.1 alfejezetben megvizsgáltuk a közös modell pontosságát az adatmérgezés mértékének, azaz a rosszindulatú kliensek számának függvényében.

A rosszindulatú klienseket el kell különítenünk a „hasznos", jól tanító kliensektől, ezeket a klienseket a továbbiakban *jóindulatú* klienseknek fogjuk nevezni.

6.3. Külső entitások

Fontos támadási pont lehet a szerver és a kliensek közötti kommunikáció lehallgatása a külső támadók által, amelynek köszönhetően privát adatokra tehetnének szert jogosulatlanul. A megoldásunkban egy olyan titkosított csatornát és titkosítási protokollt használunk, amelynek köszönhetően a résztvevők akkor sem kompromittálnának privát információt, ha a résztvevők közötti kommunikációt lehallgatnák.

6.4. Teljesítendő feltételek

Tekintsük meg a teljesítendő biztonsági feltételeket és feladatokat a kliensek (K), a szerver (S) és a külső entitások (O) oldaláról!

- K_1: A kliensek nem ismerhetik meg a szerver tesztadatbázisának kimeneti értékeit.
- K_2: A kliensek nem ismerhetik meg egymás modellfrissítéseit, de az aggregált közös modellt igen.
- S_1: A szerver nem ismerheti meg sem a kliensektől kapott modellfrissítéseket, sem pedig az aggregált modellt.
- O_1: A külső entitások nem ismerhetnek meg semmilyen adatot (sem a kliensek modellfrissítését, sem a közös modellt).

A fenti feltételek teljesítésére a 8.2 alfejezetben ismertetjük a megoldásokat.

7. Federált tanulás

A federált tanulás fontosságát már röviden bemutattuk, az alábbi fejezetben pedig egy eddigieknél részletesebb képet adunk a résztvevőkről, a végbemenő folyamatokról, az elvárt működésről. A fogalmak megértésének könnyítése érdekében bemutatjuk a gyakorlatban használt adatbázist és a neurális hálót is.

7.1. Résztvevők

A federált tanuláshoz szükség van egy központi szerverre és legalább két kliensre. A kliensek maximális számára nincsen korlátozás. A kliensek adatbázisainak egyforma formátumra hozhatónak kell lennie, azaz azonos bemenetekkel és kimenetekkel kell rendelkezzenek. Ez a neurális háló helyes működése miatt szükséges feltétel, erről a 7.2 alfejezetben lesz szó. További információ a federált tanulással kapcsolatban a [11] publikációban található.

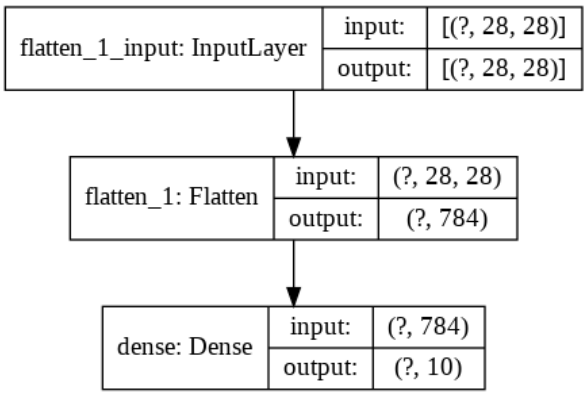
3. ábra – Az MNIST adatbázis néhány bemeneti értéke kézzel írt számjegyekről [21]

A gyakorlati megvalósításban az MNIST egyik adatbázisát használtuk, amely kézzel írt számjegyeket és azokhoz tartozó értékeket tartalmaz [22]. A bemeneti értékek egyenként 28 pixel szélességű és hosszúságú szürke árnyalatú képek. Egy képet egy 28x28-as vektor jelképez, ahol 1 esetén fekete a pixel, 0 esetén fehér. Az adatbázis tanító adatbázisában 60000, a tesztadatbázisban 10000 kép-érték páros található. A tanító adatbázist daraboltuk fel annyi egyenlő darabra, ahány klienssel szimuláltuk a futást (általában 10-100 kliens). 100 kliens esetén 60000/100 = 600 különböző kép-érték párost kapott egy kliens. A rosszindulatú támadást is szimuláltuk, amit úgy értünk el, hogy a támadók adatbázisában szereplő képekhez tartozó értékeket módosítottuk egy 0-9-es skálán lévő véletlen egész

számra. Így a rosszindulatú támadók a tanítás során helytelen kép-érték párosok alapján tanulnak, emiatt rontják a közös modell pontosságát is (adatmérgezés). A rosszindulatú támadók aránya módosítható volt, ezt az eredmények kiértékelésekor vizsgáltuk.

7.2. Kliensek modelljei

A megoldásunkban a federált tanulásban résztvevő kliensek modelljei azonos neurális hálók. Ez egy szükséges feltétel ahhoz, hogy a modellek aggregálását a központi szerver el tudja végezni, és olyan eredményt adjon, amelyet érdemes tovább tanítani. Törekedtünk arra, hogy egy egyszerű (rejtett réteg nélküli) modellt használjunk, ugyanis a későbbiekben bemutatott algoritmusok szempontjából ez lényegtelen, viszont a futási időt nagy mértékben le tudtuk csökkenteni.



4. ábra – A gyakorlati megoldásban használt neurális háló

A használt modell olyan rétegekből áll, amelyek esetén minden rétegnek pontosan egy bemeneti és egy kimeneti adathalmaza van. Mivel az írott számjegy felismerés feladata a gépi tanulás témakörében egy egyszerű feladat, így az alábbi egyszerű hálót alkottunk meg, amely az ábrán (4. ábra) látható. A 28x28-as képeket először a Flatten réteg segítségével 784 hosszú 1 dimenziós listákra konvertáljuk, majd a kimeneti Dense rétegre kötjük. A Dense (másnéven fully-connected) rétegnél alapértelmezetten jelen van egy torzítás (bias), amely a réteg kimeneteihez rendel eltolást, tehát egy 10 hosszú vektorként jelenik meg. A biason kívül a modell rendelkezik egy 784·10 méretű súlymátrixszal, így a kliensek modellje a felsorolt 784·10+10 = 7850 paraméterrel reprezentálható. A kliensek tanítás során ezeket az paramétereket módosítják úgy, hogy minél pontosabb jóslatot biztosítsanak. Jelen dokumentumban ezekre az értékekre a modell paramétereiként hivatkoztunk. Az implementációban a modellek veszteségfüggvényeként a *sparse categorical*

crossentropy függvényt választottuk [23], ez megfelelő kettő vagy annál több típusú kimeneti értékű adatbázis esetén (jelen esetben 10-féle kimenetünk van). A modell pontosságát a *sparse categorical accuracy* metrika szerint mértük [24].

7.3. Folyamatleírás

A modellfrissítéseket valamilyen módon aggregálni is kell, ez a központi szerver feladata lesz. Amennyiben a szerver megkapta a frissített modelleket a kliensektől, akkor aggregálja, majd az aggregált modellt visszaküldi a klienseknek, ezután indul a federált tanulás egy újabb köre, a kliensek újra taníthatják a (már közös) modellt. Fontos kiemelni, hogy a szerver limitálhatja az egy körben résztvevő kliensek számát, a kiválasztottak aránya általában 10-25% között van. Az egy körben kiválasztott kliensek limitációja a változatosságot tudja növelni, ami általában a modell pontosságát is növelni szokta.

Hangsúlyt fektettünk arra, hogy a klienseket hatékonyságuk szempontjából súlyozni tudjuk, a szerver által kiszámolt súlyokról a 8.1 alfejezetben olvashatunk. A súlyozás következtében a modellek frissítései különböző mértékben számítanak majd bele az aggregált modellbe.

7.3.1. Modellfrissítések aggregálása

A központi szervernek a modellfrissítéseket először súlyozni majd aggregálni kell. Aggregálás alatt azt értjük, hogy minden kliens modelljét megszorozza az adott klienshez tartozó súllyal majd az eredményeket összeadja. Így az aggregált modell felépítése egyezni fog a kliensek modelljeivel, és a kliensek képesek lesznek arra, hogy tovább tanítsák ezt a modellt.

Legyen t a kliensek modelljeinek paraméterei közül az első, w pedig a szerver által a kliensekhez számolt súly. Ekkor a modellek (i = 0,1…k) t_i értékeinek súlyozott aggregáltja t_{agg}, amely kiszámítása az alábbi módon lehetséges:

$$t_{agg} = \sum_{i=0}^{k} w_i * t_i$$

A modell paramétereinek számával megegyező számú ilyen műveletet kell elvégeznünk a súlyozott-aggregált modell kiszámításához, ez a 7.2 alfejezetben ismertetett modell esetén 7850 operációt jelent. Egy operáció a fenti képlet szerint megy végbe (k darab szorzás,

majd k darab összeadás). Ahhoz, hogy ezeket ki is lehessen számítani, ismertessük a képletben szereplő ismeretlen értéket, a szerver által kiszámolt súlyt!

8. Megoldás

Célunk az, hogy egy olyan biztonságos aggregálást valósítsunk meg, amely képes arra, hogy a 6. fejezetben ismertetett támadó modell ellen védelmet biztosítson. Ennek érdekében szükséges meghatároznunk egy olyan súlyozást, amely kiszűri (alacsony súllyal látja el) a federált tanulásban résztvevő rosszindulatú szereplőket. A biztonságos kommunikáció szükséges feltétele többek között az, hogy létezzen egy biztonságos aggregálás. Az aggregáláshoz olyan titkosítási koncepciót mutatunk be, melynek segítségével a federált tanulásban résztvevők információi, modelljei nem kompromittálódnak.

8.1. Súlyozás

A szervernek az egyik feladata, hogy valamilyen algoritmus segítségével súlyozza a klienseket, ugyanis ezáltal elkerülhető, hogy ugyanakkora súllyal vegyenek részt a közös modell frissítésében azon kliensek, akiknek a hozzájárulása eltérő. Ez azért rendkívül fontos, mert ki kell szűrnünk az esetleges adatmérgezéses támadásokat is, amelyek a közös modell pontosságának csökkentésével járnának. El kell kerülnünk azt, hogy ugyanakkora „beleszólása” legyen egy olyan kliensnek, aki nagyságrendekkel kisebb tanuló adatbázissal rendelkezik, rossz teljesítményű modelleket küld (aminek akár a hibás adatokkal való tanítás is lehet az oka). Következésképpen a biztonság és a teljesítmény növelésével is járhat egy megfelelő súlyozási algoritmus használata.

8.1.1. Súlyozás teljesítmény alapján

Fontos kiemelni, hogy az általunk bemutatott súlyozási algoritmusban figyelmen kívül hagyjuk a kliensek adatbázisának méretét. Az implementációban egyenlő méretű adatbázissal rendelkezik az összes kliens, kizárólag a kliensek teljesítményét használjuk fel a súlyozáshoz. Az egyenlő adatbázisméret nem feltétele a súlyozó algoritmus működőképességének, viszont nem vizsgáljuk a különböző adatbázisméretek esetén a hatékonyságot. Az értékek kiszámításához a szervernek szükséges tudnia azt, hogy hogyan teljesítettek a kliensek egy-egy körben. Ezt természetesen nem „kérdezheti meg” a kliensektől, ha azt feltételezzük róluk, hogy rosszindulatú résztvevők is lehetnek. Emiatt erről a szervernek kell döntenie.

A súlyozásnál használt a teljesítménymutatónak a *pontosságot* választottuk. Ehhez feltételeznünk kell, hogy a szerver rendelkezik egy tesztadatbázissal, ugyanis jelen esetben a

pontosságot ez alapján fogja számítani. A tanítási folyamat során a szerver körönként elküldi a tesztadatbázis bemeneti értékeinek egy azonos szeletét (akár az egészet), a kliensek pedig megjósolják az ezekhez tartozó kimeneti értékeket. A tesztadatbázis bemeneti értékeihez tartozó kimeneti értékek természetesen rejtve maradnak a kliensek számára. A pontosság megadja a valószínűségét annak, hogy egy kliens egy adott bemenetre helyes kimenetet ad, értéke 0 és 1 közé fog esni minden esetben.

$$0 \leq a\ kliens\ pontossága\ = \frac{helyesen\ értékkel\ jósolt\ tesztadatok\ száma}{összes\ tesztadat} \leq 1$$

A megoldásunk során többféle algoritmust is teszteltünk, melyekben minden esetben szerepelt a fent említett pontosság mint változó. A kliensek súlyainak összege minden esetben 1, ugyanis a súlyozás célja, hogy meghatározzuk, hogy melyik kliens milyen arányban „szólhat bele" a közös modell felépítésébe. Ehhez a súlyozás során mindig normalizálni kell az algoritmusok kimeneteként kapott értékeket úgy, hogy összegre vonatkozó feltétel teljesüljön. A normalizálás az algoritmus kimeneteként kapott értékeknek az értékek összegével való leosztásával ekvivalens.

8.1.2. Kliensek korábbi teljesítménye

Megfontolandó döntés volt, hogy a kliensek múltja befolyásolja-e a súlyozó algoritmust, tehát ha például az előző körben rossz pontosságú modellt küldött vissza, akkor feltételezzük-e, hogy a következő körben is rosszul fog teljesíteni. A kérdésünk a következő: valószínűsíthető-e, hogy a körök között módosul egy kliens viselkedése (pl. jóindulatú résztvevőből támadó lesz)?

Amennyiben igen, akkor valószínűleg a kliens által küldött modell pontossága is különbözne, ezért megfontolandó döntés, hogy mekkora mértékben számítson a korábbi súly, vagy akár számítson-e egyáltalán. Ellenkező esetben számíthatunk arra, hogy a kliens hatékonysága hasonló mértékű lesz (tehát a korábbi súly nagy valószínűséggel tükrözni fogja a valóságot), ezért érdemes lehet a korábbi súlyokat elmenteni, és felhasználni az új súly kiszámításakor.

Az általunk vizsgált esetben feltételezhetjük, hogy a kliensek viselkedése elhanyagolható mértékben fog változni körönként, tehát a futás során nem lesz például a támadóból jól teljesítő kliens, és nem változik a tanulás kezdete után az adatbázisa semelyik kliensnek, ezért a szerver súlyozási algoritmusának érdemes figyelembe vennie a múltban kapott

súlyokat is, azaz például az előző körben kapott súlynak érdemes szerepelnie a súly kiszámításának képletében. Természetesen az első körben ez az érték mindenkinél egyenlő (1/k, ahol k az egy körben kiválasztott kliensek száma).

8.1.3. Multiplicative Weight Update alapú súlyozás

A súlyozáshoz több algoritmust is megvizsgáltunk a korábbiakban, ezeket hatékonysági és biztonsági szempontból is értékeltünk [25]. Ezek közül a széles körben használt Multiplicative Weight Update (MWU) alapú algoritmussal foglalkoztunk mélyrehatóan, erről bővebben a [26] publikációban olvashatunk. Az MWU algoritmust általában optimalizációs problémák megoldására használják (pl.: [27], [28]).

Az MWU algoritmusok segítségével iteratívan dönthet a szerver, és a kliensek súlyait frissítheti annak a visszajelzésnek megfelelően, hogy egy kliens jól vagy rosszul teljesített. A 8.1.2 alfejezetben ismertettük, hogy az általunk vizsgált koncepciónál érdemes a múltbeli teljesítményt figyelni, ezért a használt súlyozási algoritmusunk ennek megfelelően MWU alapú. A MWU egy általános algoritmus, amelynek az alapgondolatát mutatjuk be a következőkben.

Adott egy halmaz (jelen esetben kliensek), melyek között adott egy eloszlás. Minden lépésben az adott *i* klienshez rendelt értéket megszorozzuk (*1+C(i)*)-vel, ahol *C(i)* egy valamilyen módon számolt jutalom. Ezután szükséges egy normalizálás, hogy az új értékek is a megfelelő eloszlást teljesítsék (jelen esetben az összegüknek 1-nek kell lennie). A használt algoritmusban az *i* kliens *C(i)* értéke annak a súlynak a negáltjával fog megegyezni, amelyet a legutóbbi kiválasztás során kapott a szervertől.

A klienshez rendelt érték kiszámítására használható algoritmusok tárháza széleskörű, jelen dokumentumban az Adaptive Boosting (AdaBoost) metaalgoritmust használtuk fel. Az AdaBoost a MWU-t alkalmazza, a teljesítmény javítása érdekében használható. A témáról bővebben a [29] publikációban olvashatunk.

Az AdaBoost képletét felhasználó algoritmus segítségével dönthetünk arról, hogy melyik kliens mekkora súllyal rendelkezzen. Az algoritmus a következő:

1. Adott körben a kiválasztott kliensek „beleszólásának” mértékét (b) az alábbi képlet szerint kiszámoljuk, ahol p a pontosság:

$$b = 0.5 * ln \frac{p}{1.01 - p}$$

   Az eredeti képletben a nevezőben 1.01 helyett 1 szerepel, de 100%-os pontosság esetén a nevezőben 0 lenne, ezért módosítottuk 1.01-re, ez rendkívül kis mértékben befolyásolja az algoritmus hatékonyságát.

2. A kapott értékek az Euler-féle szám (e) kitevőjében jelennek meg, ezt pedig megszorozzuk az előző súllyal (w_{i-1}).

$$w_i = w_{i-1} * e^b$$

3. A súlyokat normalizáltuk úgy, hogy összegük 1 legyen, azaz mindegyiket különkülön leosztottuk a súlyok összegével.

8.2. Biztonsági feltételek teljesítése

A federált tanulás helyes működése érdekében teljesülnie kell a korábbiakban (6.4) ismertetett feltételeknek. A biztonsági feltételek teljesítésének megoldásait részletezzük az alábbiakban

8.2.1. Biztonságos kommunikáció

A kliensekre vonatkozó egyik feltétel (6.4 – K_1) szerint nem ismerhetik meg a kliensek a szerver tesztadatbázisának kimeneti értékeit. Amennyiben kikötjük, hogy a szerver nem küldheti el a klienseknek a tesztadatbázis kimeneti értékeit, akkor ez a feltétel teljesül is. Ez azért fontos, mert egyébként a körönként kiválasztott kliensek a jósolt értékeket manipulálni tudnák, így nem tükröznék a valóságot a kiszámolt *pontosságok*, tehát a kliensek súlyozása pontatlan lenne. Megismertük, hogy a szerver a rábízott feladatokat helyesen végzi el, ezért ezt a feltételt teljesítettnek tekintjük.

Ahhoz, hogy a kliensek a használt neurális háló felépítését és a kulcsokat el tudják juttatni egymásnak, egy titkosított és autentikált kommunikációs csatorna meglétét feltételezzük a kliensek között, amely megvalósítható pl. a Transport Layer Security (TLS) segítségével, egy megvalósítási példát itt találhatunk: [30]. Amennyiben ez a kommunikációs csatorna nem létezne, akkor sérülne a titkosítás biztonsága, mert a szerver vagy külső entitások hozzáférhetnének a titkos kulcshoz, így a kliensek adataihoz is (pl.: modellfrissítések, közös modell).

A többi feltétel teljesíthető többféle titkosítási koncepció segítségével is. Ezek közül a saját megoldásunkat fogjuk ismertetni. Az általunk használt koncepcióról elmondható, hogy az alapja egy aszimmetrikus kulcsú titkosítás, melynek privát és publikus kulcsa az összes kliensnél azonos. Erről a titkosításról későbbiekben bővebben is lesz szó a 8.2.2 alfejezetben.

Az azonos publikus és privát kulcsok miatt az egyes kliensek titkosított modellfrissítései nem juthatnak el a többi klienshez, ugyanis dekódolni tudnák azokat. Természetesen semelyik kliensnek nem célja, hogy ezeket a szerveren kívül a többi klienssel megossza, ezért csak a szerver oldaláról kell vizsgálni az esetet. A szerver kizárólag a titkosítottan aggregált modellt küldheti el a kliensek számára, a kliensektől kapott titkosított modellfrissítéseket nem. Ismét hivatkozhatunk arra, hogy a szerver „őszinte”, ezért a rábízott

feladatokat teljesíti, tehát ha leszögezzük, hogy nem küldheti el a kliensek modellfrissítését semelyik résztvevőnek sem, akkor a kliensek nem ismerhetik meg egymás modellfrissítéseit (6.4 – K_2).

Egyik kliensnek sem érdeke titkosítás nélkül megosztani a szerverrel a modellfrissítését vagy a közös modellt. Így a szerver csak abban az esetben tudja dekódolni a kapott titkosított modellfrissítéseket, ha valamilyen módon az egyik kliens megosztaná vele a titkos kulcsot, és ezt a szerver kihasználná. Viszont a szerver „őszinte", így ha rögzítjük, hogy ezt nem teheti meg, akkor nem fogja megszegni a szabályokat. Így nem ismerheti meg sem a modellfrissítéseket, sem pedig a közös modellt, ezáltal az S1 feltétel teljesülni fog.

A szerver és a külső entitások nem ismerheti meg sem az egyes kliensek modellfrissítéseit, sem pedig az aggregált modellt (6.4 – S_1, O_1). Ennek ellenére képesnek kell lennie arra, hogy súlyozza őket a 8.1 alfejezetben ismertetett módon, majd a súlyozott értéküket aggregálja. Hogyan teljesülhet egyszerre az, hogy a szerver nem tudja dekódolni, műveletet (súlyozás és aggregálás) mégis tud végezni rajta? A megoldás kulcsszava az additív homomorfizmus, a titkosítási részleteit a következő alfejezetben ismertetjük.

8.2.2. Biztonságos aggregálás

A biztonságos aggregáláshoz a klienseknek olyan megbízható eljárással kell titkosítaniuk a modellfrissítéseiket, hogy azokon elvégezhető legyen a súlyozás és az aggregálás művelete.

Vizsgáljuk meg a fent említett műveleteket! A súlyozás alatt egy *szám* súllyal ($s>0$) való összeszorzását értjük, amely természetesen megegyezik azzal, ha a *számot s-szer* ismételten adjuk össze. Az aggregálás pedig ezen súlyozott értékek összeadását jelenti. Ebből látható, hogy a kliensek általi titkosításnak szükséges és elegendő feltétele, hogy additívan homomorf legyen. A homomorf titkosításokról bővebben a következő publikációban olvashatunk: [31].

Az általunk választott megoldás az ElGamal titkosításon alapul, amelynek legismertebb változata multiplikatívan homomorf, erről bővebben a [32] cikkben olvashatunk. Azért az ElGamal titkosítást választottuk, mert biztonságosnak tekinthető a döntési Diffie-Hellman feltételezés számítási nehézségének köszönhetően. A multiplikatívan homomorf titkosítási koncepció nem megfelelő számunkra, de az algoritmus módosításával additívan homomorffá tehetjük [33].

8.2.2.1. Módosított ElGamal titkosítás

A titkosítási algoritmus ismertetéséhez először bemutatjuk a használt kulcsokat, amelyeket a G ciklikus csoport segítségével generálunk. Z_p a pozitív egészek halmazát jelenti. A modulokat nem jelöltük az alábbi képleteknél.

- $g \rightarrow G\ tetszőleges\ csoporteleme$
- $x \rightarrow Z_p\ tetszőleges\ eleme$
- $h \rightarrow g^x$

Ebből a publikus kulcs a *(g, h)* páros, a privát kulcs pedig az *x* érték.

8.2.2.1.1. Titkosítás

Az értékek titkosításánál az eredeti ElGamal titkosításhoz hasonlóan szükséges egy randomizált érték, amelynek köszönhetően minden titkosított értékhez más és más rejtjelezett üzenet (kriptoszöveg) tartozik, ennek jelölése *y*. A titkosítandó üzenet jelölése *M,* értéke pedig kizárólag Z_p-beli egész szám lehet, ennek kiküszöbölését a 8.2.2.3.1 alfejezetben ismertetjük.

- $y \rightarrow G\ tetszőleges\ csoporteleme\ (minden\ érték\ titkosításánál\ egyedi)$
- $M \rightarrow Z_p\ tetszőleges\ eleme$

A fenti két érték és a publikus kulcs segítségével kiszámítható a *(c_1, c_2)* páros, amely az *M* üzenethez tartozó kriptoszöveg lesz.

$$c_1 = g^y$$

$$c_2 = g^M \cdot h^y$$

8.2.2.1.2. Dekódolás

A kapott (c_1, c_2) kriptoszöveg dekódolását az alábbi művelettel készítjük elő:

$$m = \frac{c_2}{{c_1}^x} = \frac{g^M \cdot h^y}{g^{yx}} = \frac{g^M \cdot g^{xy}}{g^{yx}} = \frac{g^M \cdot g^{xy}}{g^{yx}} = g^M$$

M-et a fent látható módon csak a g kitevőjében kaptuk meg, ezt a klienseknek vissza kell majd fejteniük. Erre a legegyszerűbb mód az, hogy a folyamat elején elkészítenek egy táblát, amiben *M* lehetséges értékeihez társítják a g^M értékeket, majd ebből vissza tudják

keresni a kiszámolt g^M-hez tartozó *M*-et. Természetesen ez *M* lehetséges értékeinek számától függően egy rendkívül költséges folyamat, ugyanis végig kell mennünk az elemeken mindaddig, amíg meg nem találjuk a keresett értéket.

8.2.2.1.3. Additív homomorfizmus

Ellenőrizzük egy egyszerű példán keresztül, hogy az algoritmus additívan homomorf-e, azaz elvégezhető-e a kriptoszövegeken a súlyozás és az aggregálás!

Adott két kliens (*1*, *2*), ezeknek egy-egy titkosítandó értéke (M_1 és M_2), és a szerver által ezekhez a kliensekhez rendelt súlyok (W_1 és W_2 egész számok).

Vizsgáljuk meg először a súlyozást az 1. kliensen!

$$c_{1_1} = g^{y_1}$$

$$c_{2_1} = g^{M_1} \cdot h^{y_1}$$

Mivel a kitevőben szerepel a súlyozandó érték, ezért a hatványazonosságok szerint a kriptoszövegeket egymással össze kell szorozni ahhoz, hogy az összegük jelenjen meg a kitevőben. Tehát az 1. kliens súlyozásánál a kriptoszövegek W_1-szer kell ismételten összeszorozni ahhoz, hogy a kitevőben M_1W_1 jelenjen meg.

$$c_{1_1}{}^{W_1} = g^{y_1{}^{W_1}} = g^{y_1 W1}$$

$$c_{2_1}{}^{W_1} = (g^{M_1} \cdot h^{y_1})^{W_1} = g^{M_1 W_1} \cdot h^{y_1 W_1} = g^{M_1 W_1} . g^{x y_1 W_1}$$

A dekódolási művelettel meg is kapjuk a keresett $g^{M_1 W_1}$ értéket.

A 2. kliens súlyozása is hasonlóképpen fog történni, a kriptoszövegek a következők lesznek:

$$c_{1_2}{}^{W_2} = g^{y_2{}^{W_2}} = g^{y_2 W_2}$$

$$c_{2_2}{}^{W_2} = (g^{M_2} \cdot h^{y_2})^{W_2} = g^{M_2 W_2} \cdot h^{y_2 W_2} = g^{M_2 W_2} . g^{x y_2 W_2}$$

A kriptoszövegeken végzett súlyozó művelet után ahhoz, hogy az összegüket megkapjuk egyszerűen az előző gondolatmenethez hasonlóan össze kell szorozni a kriptoszövegeket.

$$\boldsymbol{c_1} = c_{1_1}{}^{W_1} \cdot c_{1_2}{}^{W_2} = g^{y_1 W_1} . g^{y_2 W_2} = g^{y_1 W_1 + y_2 W_2}$$

$$\boldsymbol{c_2} = c_{2_1}{}^{W_1} \cdot c_{2_2}{}^{W_2} = (g^{M_1 W_1} . g^{x y_1 W_1}) \cdot (g^{M_2 W_2} . g^{x y_2 W_2}) =$$

$$= g^{M_1 W_1 + M_2 W_2} \cdot g^{x(y_1 W_1 + y_2 W_2)}$$

A privát és publikus kulcsról ismét megemlítenénk, hogy ez az összes kliens esetén azonos, ezért nem kaptak indexet sem. A $\frac{c2}{{c_1}^x}$ művelet elvégzése után a $g^{M_1W_1+M_2W_2}$ értéket kapjuk, amelynek kitevőjében az általunk kívánt érték jelenik meg. Látható, hogy a kitevőben lehetséges értékek halmaza ezekkel a műveletekkel tovább nőtt, így az érték megtalálása tovább nehezedik.

A megoldás során nagy szerepet kapott az, hogy a biztonsági szint a lehető legnagyobb mértékű lehessen úgy, hogy a kulcsok bitszámát alacsonyan tudjuk tartani. Ennek elérése érdekében egy széles körben elterjedt technológiát használtunk fel, az elliptikus görbéken alapuló titkosítást.

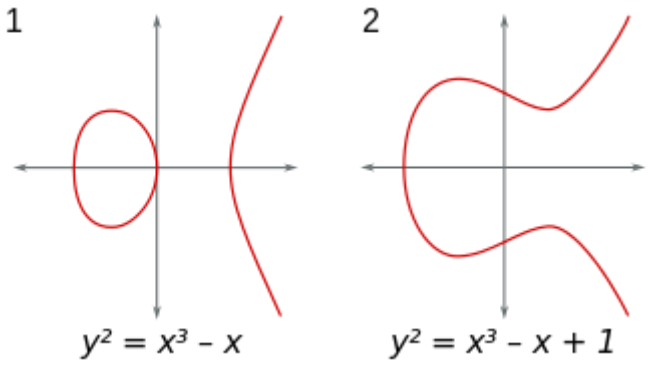
8.2.2.2. Elliptikus görbe

Az elliptikus görbe alapú rejtjelezés legnagyobb előnye a megnövelt sebesség, ugyanis az elliptikus görbe alapú algoritmusok lényegesen kisebb kulcsméretet használnak, mint a nem elliptikus görbén alapuló megfelelőik. A kulcsméret növelésével drámaian növekszik a biztonsági erőssége ezeknek az algoritmusoknak. A 1. táblázat látható, hogy a szimmetrikus kulcsú és a nem elliptikus görbe alapú aszimmetrikus kulcsú titkosítások (pl. RSA) biztonsági szint szempontjából mekkora kulcshosszok esetén összemérhetők.

Szimmetrikus	RSA	Elliptikus görbe alapú
80 bit	1024 bit	160 bit
112 bit	2048 bit	224 bit
128 bit	3072 bit	256 bit
192 bit	7680 bit	384 bit
256 bit	15360 bit	512 bit

1. táblázat – A különböző típusú titkosítások bitben mért kulcshosszai soronként közel azonos biztonsági szint alapján [34]

Egy elliptikus görbe egy olyan struktúra, melynek (*x, y*) pontjaira teljesül az $y^2 = x^3 + ax + b$ egyenlet. Biztonságosságát az adja, hogy a görbén található pontokon végzett műveletek egyszerűen végezhetőek el, viszont ezek inverz művelete már a diszkrét logaritmus problémához vezet a paraméterek megfelelő választása mellett [34].



5. ábra – Két egyszerű elliptikus görbe grafikonja [35]

A megoldásban használt prime192v2 elnevezésű elliptikus görbe kellő biztonságot biztosít. Az elliptikus görbe paramétereit megtekinthetjük az alábbi weboldalon: [36].

Fontos kiemelni, hogy az elliptikus görbéken található pontokon végzett műveleteket összeadásként, a többszörös összeadást pedig szorzásként szokták említeni, viszont a megoldásunkban a szorzás és hatványozás fogalmát fogjuk használni. A továbbiakban amennyiben két elliptikus görbe szorzásáról beszélünk, akkor a [34] szerinti összeadást, hatványozás esetén pedig a szorzást értjük.

8.2.2.3. Módosított ElGamal titkosítás elliptikus görbén

Az 8.2.2.1 alfejezetben bemutatott megoldás átalakítható úgy, hogy elliptikus görbén alapuljon. Ami az eredeti titkosításban a *G* ciklikus csoport, az ennél a megoldásnál maga az elliptikus görbe lesz. A publikus kulcs első tagja (*g*) egy tetszőleges elliptikus görbe pont, a privát kulcs (*x*) pedig egy nagy bitszámú egész szám, ami egy tetszőleges elliptikus görbe pont x-tengelyhez tartozó értéke (azaz egy Z_p-beli elem).

A privát és publikus kulcsok mellett meg kell határozni, hogy mik lehetnek az M titkosítandó üzenetek, és a titkosításnál az *y* értékek. Az *y* *g*-hez hasonlóan az elliptikus görbe egy tetszőleges pontja lesz, a korábbiakhoz hasonlóan minden érték esetén egyedi. Az *M* az *x*-hez hasonlóan Z_p-beli elem lehet.

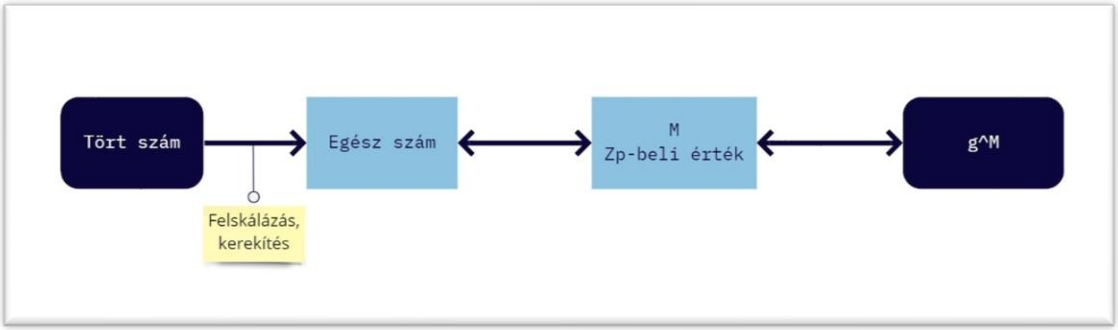
8.2.2.3.1. Titkosítandó üzenetek megfeleltetése Zp-beli elemmel

Viszont a titkosítandó üzeneteink a modell paraméterei, amelyek általában tört értékek, ezért szükséges egy megfeleltetést a *Zp*-beli elemek és a modell paraméterei között. A használt megfeleltetés a következő:

1. A tört értékeket egy adott számmal felskálázzuk (*skálázási érték*), majd egészre kerekítjük.
    a. Ez a skálázási érték rendkívül fontos az algoritmus sebessége szempontjából, erről később részletesen is beszélünk.
2. Generálunk egy tetszőleges Z_p-beli elemet (zp).
3. Az egészre kerekített értékek szerint meghatározzuk a törtekhez tartozó Zp-beli elemeket:
    a. A pozitív értékekhez ($zp + zp + \ldots + zp$) összeget rendelünk, ahol a tagok száma a pozitív egész értékkel egyenlő.
    b. A negatív értékekhez $0 - (zp + zp + \ldots + zp)$ összeget rendelünk, ahol a zárójelben lévő tagok száma a negatív értékkel egyenlő.
    c. A 0-hoz 0-t rendelünk.
    d. Megjegyzés: a kriptográfia alapjait megvalósító könyvtár nem képes Zp-beli elemek szorzására, kizárólag összeadására, ezért nem helyettesítettük a fenti műveleteket egy egyszerű szorzással.

Fontos ismét kiemelni, hogy dekódolás során nem a Z_p-beli értékeket (M) fogjuk megkapni, hanem a g^M értékeket, így érdemes az M és g^M között is egy megfeleltetést eltárolni.

Mivel a Keras lehetőséget biztosít arra, hogy limitáljuk a paraméterek értékének intervallumát, ezért beállítottuk, hogy ez az érték -1 és 1 közötti legyen. Ezeket a törteket kell megfeleltetnünk Z_p-beli elemekkel.



6. ábra – A kliens paramétere és a dekódolt érték közötti leképezés folyamata

A kliens paraméterei törtek, ezekből először egész számokat kell csinálnunk. Ezt a törtek felskálázásával, majd kerekítésével tehetjük meg. Ennek a módszernek egy választható értéke a *skálázási érték*. Minél kisebb a skálázási érték, annál kevesebb egész számot

rendelhetünk a törtekhez. Ezek az egész számok egyértelműen meghatározzák a Z_p-beli elemeket is (M), amelyekből kiszámítható a g^M is. Ez a folyamat látható az ábrán (6. ábra).

A fentiek alapján a skálázási értéktől függ, hogy mennyire lesz gyors a dekódolás, ugyanis minél kisebb a skálázási érték, annál kevesebb Z_p-beli érték között keresünk, és annál gyorsabban megtaláljuk a hozzárendelt egész számot. Az egész szám megtalálása után a skálázási értékkel való visszaosztással pedig meg is kaphatjuk a törtet. (Ezért nem kétirányú a nyíl a tört és az egész szám között, mert nem feltétlenül egyértelmű a megfeleltetés).

Fontos megemlíteni, hogy a skálázás utáni kerekítés adatvesztést is jelent, tehát a modell paraméterei a dekódolást követően csak korlátozott számú értéket vehetnek fel a skálázási értéktől függően. Az adatvesztés következtében a modell pontossága is csökken. Amennyiben a skálázási érték s, akkor a dekódolható értékek száma s (negatív értékek) + 1 (a nulla) + s (pozitív értékek) lesz.

Például hogyha a skálázási érték 10, tehát tízzel szorozzuk meg a kliens paramétereit, majd egészre kerekítjük ezeket az értékeket, akkor minden törthöz tartozó érték a {-10, -9, …, -1, 0, 1, …, 9, 10} halmaz egyik tagja lesz (mivel az összes érték -1 és 1 közötti), és így habár a dekódoláskor könnyen meg fogjuk találni a 21 egész értéknek megfelelő g^{Zp} értékeket, de a modell így nem fog megfelelő pontosságot adni, mert a modell paraméterei csak a {-1, -0.9, …, -0.1, 0, 0.1, …, 0.9, 1} halmazból kerülhetnek ki.

Tehát azt a legkisebb skálázási értéket kell megtalálnunk, amely csak kis mértékben csökkenti a modell pontosságát a titkosítás és dekódolási műveletek következményeként. Ez minden modellnél más és más, a jelenlegi példánkban a több értéket is leteszteltünk, ami alapján meghatároztunk egy értéket.

Amennyiben megtaláltuk a megfelelő skálázási értéket, döntenünk kell hasonló kérdésben a súlyozás szempontjából is. Ugyanis a titkosítási algoritmusnál kikötöttük 8.2.2.1.3, hogy a súlyok (W) értéke is csak egész szám lehet (csak egész számszor lehet összeszorozni a kriptoszövegeket önmagukkal), ezért ezeket is fel kell skáláznunk a megfelelő eredmény elérése érdekében. Ismét megemlítenénk, hogy a kiszámolt súlyértékek 0 és 1 közötti törtek. A felskálázás hasonló lesz, megszorozzuk az értékkel, majd kerekítünk. A skálázási érték azt fogja megadni, hogy amennyiben a súlyok csak egész számok lehetnek, akkor összesen hány súlyt oszthatunk szét. Mivel körönként kapják meg a kliensek a szervertől a súlyt, ezért minden körben annyi kliens között kell szétosztanunk a súlyt, ahányan ki lettek választva.

Például 5 kiválasztott kliens esetén amennyiben skálázási érték 20, akkor egy megfelelő megfeleltetés a {0,6,8,1,5}, azaz az első kliens beleszólása 0%, a másodiké 60% lesz stb. Felmerülő probléma, hogy a súlyozás során kapott értékek felskálázása és kerekítése után az összegük nem lesz egyenlő a skálázási értékkel a kerekítési hiba következtében. Előző példánál maradva lehetséges, hogy a {0,5,8,1,4} értékeket kapjuk, viszont ezek összege 18 lesz, a maradék 2-t is el kell osztani. Dönthetünk úgy, hogy a maradékkal megnöveljük a legnagyobb súlyt, de el is oszthatjuk úgy, hogy a legnagyobb súlyú klienstől indulva mindenki kap egyet addig, amíg a maradékból hátra van. Amennyiben több súlyt osztottunk szét a skálázási értéknél, akkor pedig elvehetünk a kliensektől a legkisebb súlyú klienstől indulva egy-egy súly. Ezt addig folytatjuk, amíg a súlyok összege egyenlő nem lesz a súlyok skálázási értékével.

Fontos megjegyezni, hogy az egy körben kiválasztott kliensek számát is érdemes figyelembe venni a súlyozás skálázási értékének meghatározásakor. Sok kliens esetén alacsony skálázási értéknél kevés súlyt oszthatunk szét a kliensek között, ami miatt egy kliensnek túl nagy beleszólása lehet, akár az összes súlyt is megkaphatja. Emiatt a központi modell csak az adott kliens modellfrissítését tartalmazná, ami biztonsági szempontból nem megfelelő, ugyanis kompromittálódna a többi kliens számára az adott kliens.

A modell paramétereinek és a súlyoknak a felskálázási értékétől függ, hogy a titkosított értékek dekódolásakor mekkora halmazban kell keresnünk az értékeket. Amennyiben a modell paraméterei -1 és 1 között vannak, a súlyozás skálázási értéke s_s, a kliensek paramétereinek skálázási értéke s_k, akkor a lehetséges értékek halmaza az alábbi határok közötti egész számokat (A) tartalmazza:

$$-1 \cdot s_s \cdot s_k \leq A \leq 1 \cdot s_s \cdot s_k$$

Például, ha a modell paramétereinek skálázási értéke 100, a súlyozásé pedig 20, akkor a 2000 és 2000 közötti egész számokhoz rendelt g^M értékek között kell keresnünk dekódoláskor (100·20+1+100·20 = 4001 érték). Amennyiben megtaláltuk, akkor csak le kell osztanunk a skálázási értékek szorzatával (2000), és megkapjuk az aggregált modell egyik paraméterének értékét (ha a g^M-hez tartozó egész szám a 145, akkor a közös modell adott paramétere 145/2000 = 0.0725 lesz).

8.2.2.3.2. Összegzés

A fenti módosításokkal minden értéknek meghatároztuk az elliptikus görbe megfelelőjét, így az elliptikus görbén végzett módosított ElGamal titkosítás működőképes. Ez a megoldás használható a kliensek modellfrissítéseinek titkosítására, a szerver ennek a koncepciónak a segítségével el tudja végezni a súlyozást és az aggregálást is, és ezt az aggregált modellt a kliensek dekódolni is képesek.

Fontos kiemelni, hogy a titkosítás aszimmetrikus, viszont ezt nem használtuk ki teljes mértékben. A publikus kulcsokat megoszthatjuk bárkivel, tehát titkosított értékeket bárkitől kaphatnánk, ez akár növelheti a koncepció gyakorlati felhasználhatóságának a spektrumát.

8.3. Megoldás pszeudokódja

A megoldásunk komplex, a súlyozás befolyásolja a titkosítás helyes működését, egymástól nehezen elkülöníthetőek megvalósítás szinten, ezért a federált tanulás egészét mutatjuk be biztonságos aggregálással és súlyozással. A már korábban bemutatott algoritmusokat nem mutatjuk be részletesen. A kliensek és szerver inicializálása után a run_server függvénnyel indíthatjuk a federált tanulást, amely körönként elküldi a súlyozott, aggregált titkosított modellt a kliensek számára.

1. clients init: p_k, s_k, mw_0, $len_m \leftarrow len(mw_0)$
   p_k: publikus kulcs
   s_k: privát kulcs
   mw_0: első közös modell paraméterei
   len_m: a modell paramétereinek száma

2. server init: db, r, k, rk, $cw_i \leftarrow 1/k$, ahol i = 0, 1…k
   db: adatbázis (bemeneti és kimeneti értékek)
      db_x: bemeneti értékek
      db_y: kimeneti értékek
   r: körök száma
   k: kliensek száma
   rk: egy körben kiválasztott kliensek száma
   cw: kliensek súlyai (kezdetben mindenkinek 1/k)

3. run_server()
   i. for ciklus i = 0, 1…r
      i. group ← random(k, rk)
         random: rk darab k-ból tetszőlegesen kiválasztott kliens
      ii. enc_g, $p_g \leftarrow$ clientupdate(db_x), ahol g = group összes tagja
         clientupdate: a kliens frissíti a modellt, majd visszaküldi a jósolt értékeket és a titkosított modellfrissítést
         enc: titkosított kliensek listája
         p: a visszaküldött jóslatok listája
      iii. cw ← weightupdate(p, cw)
         weightupdate: a szerver által meghatározott súlyok frissítése
      iv. enc ← weighting(enc_g, cw_g), ahol g = group összes tagja
         weighting: a kiválasztott kliensek titkosított par. súlyozása
      v. $enc_{agg} \leftarrow$ aggmodels(enc)
         aggmodels: a kiválasztott kliensek titkosítottan súlyozott par. aggregálása
      vi. sendmodel(enc_{agg})
         sendmodel: a titkosított modell elküldése az összes kliens számára

4. clientupdate(db_x) – g kliens
    i. $mw_g \leftarrow$ train(mw_g)
        i. $mw_g \leftarrow$ train(mw_0), ha mw_g = null
        ii. $mw_g \leftarrow$ train(decrypt(enc_{agg})), ha mw_g != null

        train: neurális hálón végzett tanítás

        decrypt: a szerver által küldött súlyozottan aggregált modell dekódolása
    ii. $p_g \leftarrow$ predict(mw_g, db_x)

        predict: db_x bemenetekre mw által jósolt kimenetek
    iii. $enc_g \leftarrow$ encrypt(mw_g)
        i. $mw_{gi} \leftarrow$ float_to_zp(mw_g) i = 0,1… len_m
        ii. enc $\leftarrow$ encryptZp(mw_g) i = 0,1… len_m

            float_to_zp: modell par. megfeleltetése Zp-beli értékkel

            zp_enc: modell par. megfeleltetett Z titkosítása
    iv. return enc_g, p_g

5. mw $\leftarrow$ decrypt(enc_{agg})
    i. $mw_i \leftarrow$ decryptZp(enc_{aggi}) i = 0,1… len_m
    ii. $mw_i \leftarrow$ zp_to_float(mw_i) i = 0,1… len_m
    iii. return mw

        decryptZp: a kriptoszövegből egy Zp-beli elemet számít

        zp_to_float: egy Zp-beli elemből kiszámítja a tört számot

9. Eredmények

A dokumentumban ismertetett koncepciókat, azaz a szerver általi súlyozást és titkosítási algoritmust Python (v3.5.2) nyelven megvalósítottuk, működőképességüket ellenőriztük. A neurális hálóval való műveletekhez a Keras (v2.4.3) könyvtárt használtuk [15]. A programkód megtalálható a következő linken: [25]. Ebben a fejezetben megvizsgáljuk, hogy a módszerünk a 7. fejezetben bemutatott környezetben mennyire hatékonyan viselkedik. A súlyozással kapott aggregált modell pontosságát figyelve a súlyozást értékeljük különböző mértékű adatmérgezéses támadás esetén. Emellett a súlyokat összehasonlítjuk a kliensekhez tartozó Shapley-értékekkel is. A titkosításnál szó lesz arról, hogy milyen sebességgel történik meg a titkosítás és a dekódolás, továbbá a futási idő csökkentésének lehetőségeit ismertetjük.

9.1. Súlyozás

A szerver a kliensek súlyát a pontosságuk alapján számolta ki a 8.1 alfejezetben ismertetett algoritmus segítségével.

9.1.1. A modell pontossága a jóindulatú kliensek arányának függvényében

Az egy körben kiszámolt súlyok értéke nagy mértékben függ attól, hogy a szerver milyen pontosságú klienseket választ egy adott körben. Ha azt feltételezzük, hogy léteznek rosszindulatú támadók is, akkor létezhet olyan balszerencsés kör a federált tanulás során, amikor csak rosszindulatú támadók lettek kiválasztva. Természetesen ebben az esetben a súlyozás értelmetlen lenne, mert így biztos, hogy bármilyen súlyozással csak rontani tudjuk a közös modellt. Tehát a súlyozási koncepciónk limitációjának tekinthető az, hogy a súlyok relatívak az adott körben résztvevőkre.

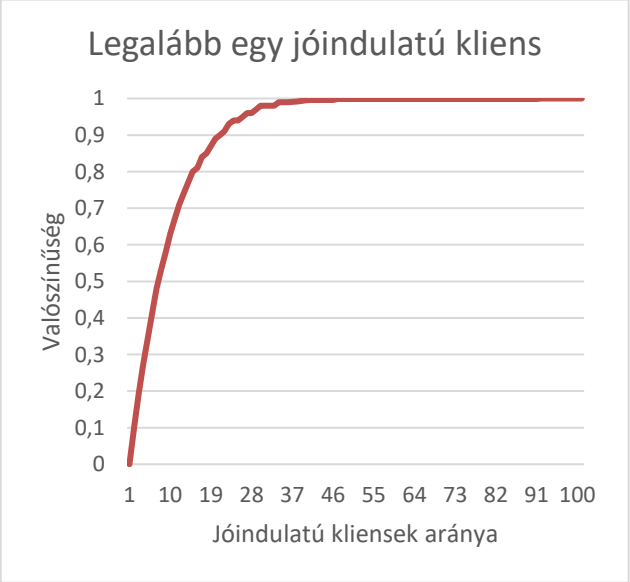
Amennyiben a szerver a súlyozással el tudja különíteni rosszindulatú klienseket a jól teljesítő kliensektől (tehát a rosszindulatú kliensnek csak rossz adata van), úgy a központi modell pontossága javulni fog, még akkor is, ha egy kivételével az összes kiválasztott kliens rosszindulatú. Ezért érdemes megvizsgálni, hogy különböző esetekben mekkora valószínűséggel választ ki a szerver úgy klienseket, hogy közöttük legalább egy jóindulatú kliens van.

Egy adott körben legalább egy jóindulatú kliens kiválasztása attól a két paramétertől függ, hogy mekkora az egy körben kiválasztásra kerülő kliensek aránya, és mekkora a jóindulatú kliensek aránya. Például vegyünk 100 kliensből körönként 10-et. A kliensek közül 5 jóindulatú, 95 rosszindulatú. Ekkor P annak a valószínűsége, hogy legalább egy jóindulatú kliens lesz kiválasztva egy körben, ahol

$$P = 1 - \frac{\binom{rosszindulatú}{kiválasztott}\binom{jóindulatú}{0}}{\binom{összes}{kiválasztott}} = 1 - \frac{\binom{95}{10}\binom{5}{0}}{\binom{100}{10}} \approx 0.416.$$

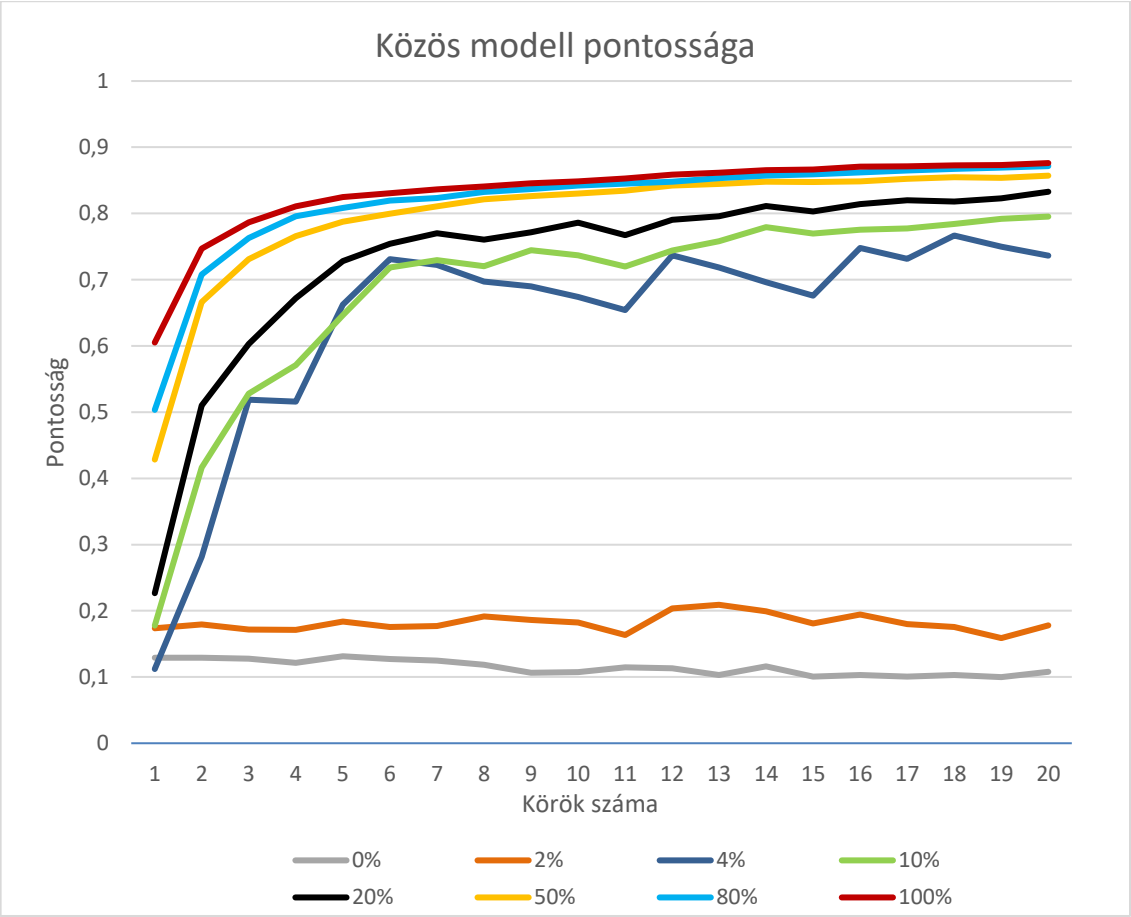
Ebben az esetben a jó kliensek aránya 5%, az egy körben kiválasztott kliensek aránya pedig 10%. Látható, hogy ilyen arányok esetén is viszonylag magas a valószínűsége annak, hogy legalább egy kliens nem lesz rosszindulatú a kiválasztottak között.

Rögzítsük az egy körönként kiválasztott kliensek arányát 10%-ra! Ekkor a P valószínűség az ábrán (7. ábra) látható módon változik a jóindulatú kliensek arányának függvényében. 20%-os arány esetén P már a 90%-ot, 36% esetén pedig már a 99%-ot is meghaladja. Ez azt jelenti, hogy a federált tanulás során már alacsony arányú jóindulatú kliens esetén is ritka lesz az, hogy csak rosszindulatú klienseket választ ki a szerver, így a megoldás limitációja nem jelent nagy biztonsági kockázatot.



7. ábra – Legalább egy jóindulatú kliens kiválasztásának valószínűsége a jóindulatú kliensek arányának függvényében

Tekintsük meg, hogy hogyan működik a gyakorlatban a federált tanulás az ismertetett súlyozással adott százalékú rosszindulatú támadó esetén. Ismétlésként megemlítenénk, hogy rosszindulatú támadókat úgy szimuláltunk, hogy az adatbázisát elrontottuk (minden írott számjegy képéhez véletlenszerű számjegyet rendeltünk). Fontos megjegyezni, hogy az egy körben kiválasztottak aránya mindig 10% volt.



8. ábra – A közös modell pontossága jóindulatú kliensek arányának megváltoztatásával körönkénti lebontásban

Az ábrán (8. ábra) láthatjuk, hogy amennyiben a jóindulatú kliensek aránya 0%, akkor természetesen mindig csak rosszindulatú klienst választ ki a szerver, így a közös modell 20 kör után sem lesz képes többre, mint ~10% pontosságra, ugyanis az összes adatbázis úgy van elrontva, hogy minden írott számjegy képéhez véletlenszerű számjegyet rendeltünk, tehát az ezeken a „mérgezett” adatokon való tanítás után sem fogunk jobb eredményt elérni, a közös modellünk egy „egyjegyű véletlenszám generátor” lesz.

A jóindulatú kliensek 2%-os arányánál ~20%, 4%-os arányánál pedig már egy jelentősen jobb eredményt, ~75%-os pontosságot értünk el. Megjegyzésként megemlítenénk, hogy

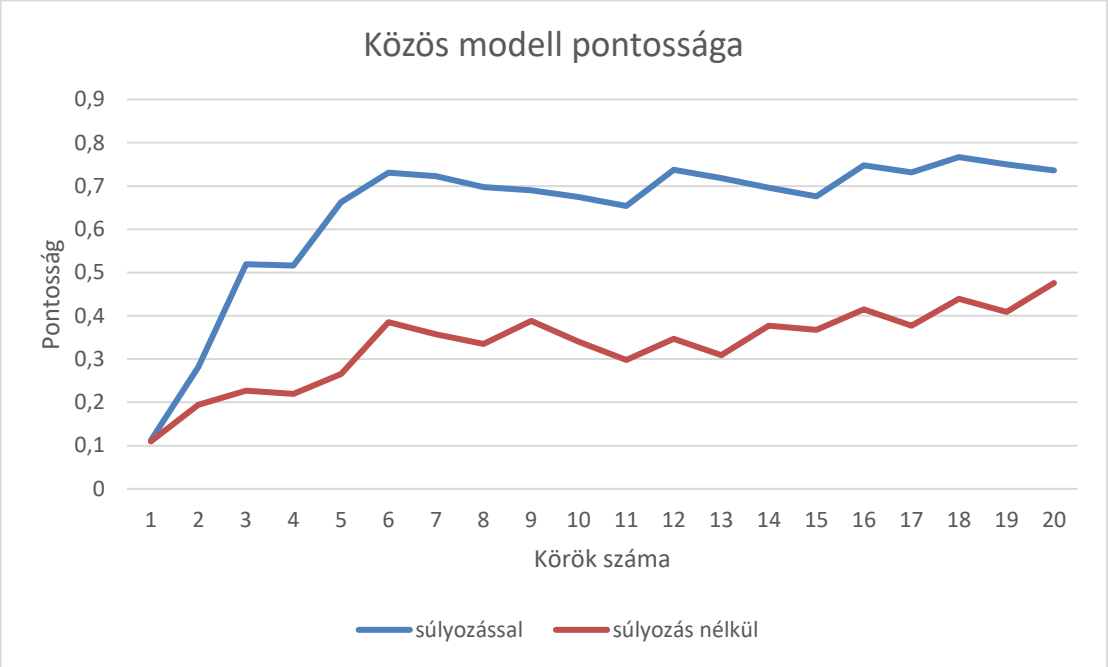
a negatív irányba történő módosulásokból azt sejthetjük, hogy az adott körben csak rosszindulatú klienseket választott ki a szerver. A korábbi számításaink szerint (7. ábra) ez ~100%-34%=66% eséllyel következik be 4%-os arány esetén, amit nagyjából tükröz is a táblázatban látható sötétkék adatsor, ugyanis 20 körből átlagosan 13-szor lett rosszabb a közös modell, mint az előző körben, ami 65%-ot jelent. A vizsgálat során rögzítettük, hogy melyik körben ki vehet részt. Természetesen az előző modellnél akkor is lehet rosszabb a modell, ha csak jóindulatú klienseket választ ki a szerver, de amennyiben csak rosszat, akkor szinte biztosak lehetünk benne, hogy csökkenni fog a közös modell pontossága.

Látható, hogy a jóindulatú kliensek arányának növelésével egyre kiegyenlítettebben növekszik a közös modell pontossága is. Amennyiben ez az arány 50%, akkor már közel olyan eredményeket (~85%) érünk el a súlyozással, mint abban az esetben, ha nincs rosszindulatú kliens (~87%). Ez a szám alapvetően mindkét esetben alacsonynak tekinthető, ezt a hiperparaméterek optimalizálásával és egy bonyolultabb modell segítségével javítani lehetett volna, viszont a megoldásunkban nem a legnagyobb pontosságot akartuk elérni, hanem azt, hogy jóindulatú kliensek arányának csökkenésével a pontosság a lehető legkisebb mértékben romoljon.

Ha nincs rosszindulatú kliens, tehát a jóindulatú kliensek aránya 100%, akkor ~87%-os pontosságot érünk el akkor is, ha a súlyozó algoritmus helyett mindenki egyenlő súlyt kap (w=1/n, ahol n a kliensek száma). A mérések során igazolódott az az állítás, hogy ebben az esetben csak nagyon kis mértékben fog számítani, hogy mekkora súlyt osztunk az egyes klienseknek, hiszen mindenki azonos modellen rendkívül hasonló adatbázisok segítségével ugyanannyi iterációt tanít. A különbség akkor lesz szembetűnő, ha vannak rosszindulatú támadók is a kliensek között.

9.1.2. Aggregálás súlyozással és súlyozás nélkül

Már részleteztük, hogy amennyiben 4% a jóindulatú kliensek aránya, és használja a szerver a súlyozó algoritmust, akkor ~75%-os pontosságot ér el a közös modell 20 kör alatt. Milyen értékeket kapunk abban az esetben, ha minden kliens egyenlő súlyt kap? Mekkora a különbség a súlyozó algoritmussal és a súlyozás nélküli aggregálással történő federált tanulás között? Amennyiben a jóindulatú kliensek aránya 4%, akkor az ábrán (9. ábra) látható eredményeket kaptuk. Látható, hogy a súlyozó algoritmussal már a 3. körben olyan pontosságot értünk el, mint a súlyozás nélkülivel 20 kör alatt.



9. ábra - A közös modell körönkénti pontossága, amennyiben a jóindulatú kliensek aránya 4%.

A súlyozó algoritmus használatakor rosszindulatú kliensek kevesebb beleszólással rendelkeztek a közös modell számításakor, kisebb súlyt kaptak a szervertől, így a modell pontossága gyorsabban javult, mint a súlyozás nélküli esetben. Általánosságban elmondható, hogy amennyiben a jóindulatú kliensek aránya nem 0%, akkor a súlyozó algoritmus hatékonyabban működik, mint a súlyozás nélküli módszer.

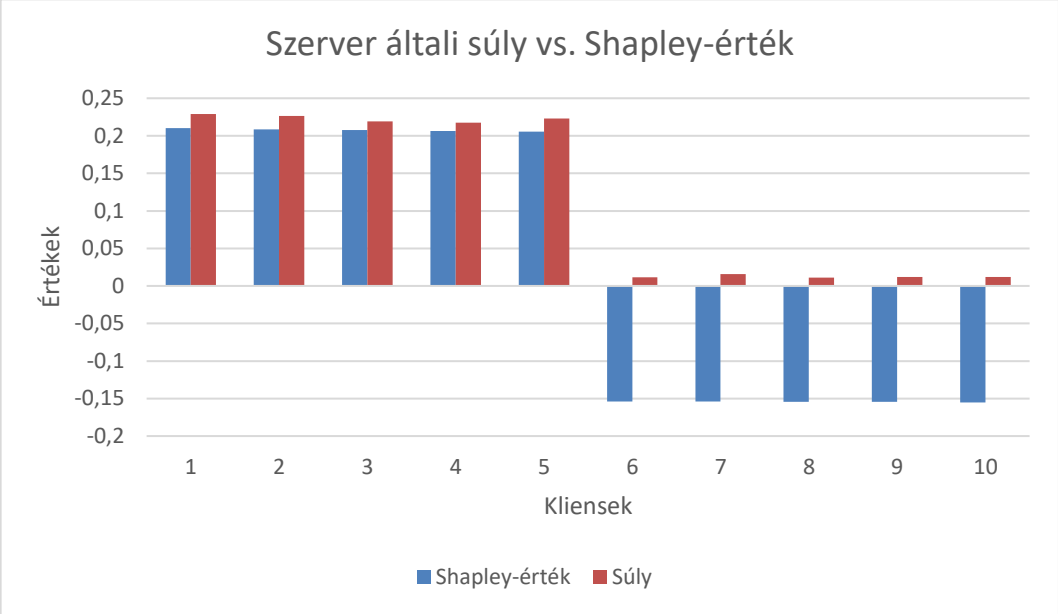
9.1.3. Súlyozás értékelése Shapley-érték segítségével

A 5.2 alfejezetben ismertettük, hogy a Shapley-érték egy igazságosnak tekinthető metrika. Ahhoz, hogy a súlyozásról is el tudjuk mondani ezt, össze kell hasonlítanunk a Shapley-értékekkel. A súlyozással elérendő cél, hogy helyesen döntsük arról, hogy egy adott

kliens jóindulatú vagy nem. Természetesen a laboratóriumi környezetben ismerjük azt egy adott kliensről, hogy jóindulatú vagy sem, viszont a gyakorlatban már nem vagyunk ilyen egyszerű helyzetben. Ahhoz, hogy pontosabb képet kapjunk a szerver által kiszámolt súlyok és a Shapley-érték kapcsolatáról, vizsgáljunk meg egy egyszerű példát!

A 5.2.2 alfejezetben olvashattuk, hogy az összes koalíció kiszámítása a kliensek számától exponenciálisan függ, $O(2^n)$ nehézségű, emiatt kiszámítása nagy n esetén költséges. A véges számítási kapacitásunk miatt a vizsgálat során a kliensek darabszámát limitáltuk.

Számoljuk ki a Shapley-értékeket és a szerver által osztott súlyokat úgy, hogy 10 kliens szerepel a tanulásban, a jóindulatú kliensek száma pedig 5. Az ábrán (10. ábra) látható az egyes kliensekhez tartozó Shapley-érték és a szerver által kiszámolt súly.



10. ábra – Kliensek Shapley-értékei és a szerver által osztott súlyai

Azt nem jelenthetjük ki, hogy a súlyozás alapján azonos sorrendbe kerülnek a kliensek, mint Shapley-érték alapján, viszont a jóindulatú és rosszindulatú kliensek magas pontossággal elkülöníthetőek, ugyanis a rosszindulatú kliensek Shapley-értékei negatívak és ezzel párhuzamosan a szerver is 0 körüli súlyt osztott számukra.

9.1.3.1. Biztonsági probléma

A fenti példában az *összes* résztvevő alapján számoltuk ki az egyes kliensekre vonatkozó Shapley-érték szerinti hozzájárulásának mértékét. Ez fontos ahhoz, hogy a Shapley-érték 5.2.2 alfejezetben ismertetett előnyei adottak legyenek, és a kiszámolt értékeket igazságos

elosztásnak tekinthessük. Emiatt módosítanunk kellett a federált tanulást úgy, hogy a körönként kiválasztott kliensek száma megegyezzen az összes kliens számával. Így a teljes populációra ki tudtuk számolni a Shapley-értéket.

Természetesen így az összes koalíció pontosságát ki kell számolni, ami felvet biztonságtechnikai kérdéseket. A fő probléma az, hogy az összes koalícióhoz tartozó aggregált modellt dekódolni kellene ahhoz, hogy a képletben szereplő pontosságot (és ezáltal a Shapley-értéket) megismerhessük. Az érték kiszámításához azokat a koalíciókat is vizsgálnunk kellene, amelyekben csak egy kliens modellfrissítése szerepel, ezáltal pedig különkülön elérhető lenne az összes kliens modellfrissítése. Ezt nem tudjuk elkerülni, ezért a titkosítási protokollt szükséges leválasztani ahhoz, hogy a szerver súlyait a Shapley-értékekkel össze tudjuk hasonlítani, esetlegesen párhuzamot tudjunk vonni vele. A Shapley-érték kiszámítását csak laboratóriumi körülmények közötti összehasonlítás miatt csináltuk, nem része a titkos súlyozó protokollnak.

9.1.3.2. Megfeleltetés

A Shapley-értékekkel kapcsolatban fontos megemlíteni, hogy negatív értékeket is kaphatunk, viszont a szerver által kiszámolt súlyokról ez nem mondható el, azok kizárólag 0 és 1 közötti értékek lehetnek, ez bemutatott példában látszik is (10. ábra).

Amennyiben párhuzamot szeretnénk vonni a két adathalmaz között, szükséges egy megfeleltetés. Ez a megfeleltetés egy olyan függvény, amely a Shapley-értékekhez súlyokat rendel. Ennek megtalálása nem egyértelmű feladat, de amennyiben megtaláltuk, akkor lesz egy Shapley-érték – súly megfeleltetésünk. Amennyiben a függvény inverzét kiszámoljuk, úgy pedig az ellenkező irányban (Shapley-érték – súly) is lesz egy megfeleltetés. Ez azért lehet hasznos, mert így akár a Shapley-értéket is megbecsülhetjük a szerver súlyainak kiszámításával, amelynek kiszámítási ideje a Shapley-érték számításával ellentétben a kliensek számától nem exponenciálisan, hanem lineárisan függ. Így a Shapley-érték becslésekor a 9.1.3.1 alfejezetben ismertetett biztonsági probléma (titkosítás hiánya) nem lenne jelen.

9.2. Titkosítás

Ahhoz, hogy az elliptikus görbén végezhessük a műveleteket, egy külső könyvtárat használtunk, a Charmot [37], egy évek óta fejlesztett, rengeteg előnnyel járó kriptográfiai keretrendszert, amelyet számos munkában használnak (pl.: [38]), ugyanis a titkosítási koncepciók prototípus-készítését jelentősen megkönnyíti, és támogatja a fejlett kriptográfiai megvalósítást. A titkosítási koncepciót már bemutattuk (8.2.2), ebben az alfejezetben ismertetjük a titkosítási és dekódolási időt.

9.2.1. Futási idő

Az általunk vizsgált kézzel írt számjegyek felismerésére készített egyszerű neurális esetén ezt a dekódolási műveletet 7850-szer kell végrehajtani, ennek a futási idejét egy virtuális gépen (i7-6500U CPU @ 2.50 GHz, 2GB memória, Ubuntu 64 bites operációs rendszer) vizsgáltuk.

A futási időt jelentősen növelő műveletek a titkosítás és a dekódolás. Ez különböző skálázási értékek esetén a 2. táblázat látható módon alakul. A táblázatban megjelenítettük a közös modellek pontosságait 5 kör után. A federált tanulásban résztvevők száma 100, körönként 10-et választ ki a szerver. A jóindulatú kliensek aránya 50%. A súlyozás skálázási értékét rögzítettük 100-ra, ugyanis a körönként kiválasztott 10 kliens esetén kellő pontosságú súlyokat kapunk így is. Továbbá egy változó paraméter vizsgálatával pontosabb képet kaphatunk az eredményekről.

s_k	s_s	tanítás	titkosítás	súlyozás, aggregálás	dekódolás	pontosság
10	**100**	2s	14s	5s	5s	**10.1%**
20	**100**	2s	13s	7s	5s	**31.3%**
35	**100**	2s	12s	4s	5s	**34.4%**
50	**100**	2s	15s	5s	6s	76.9%
100	**100**	2s	14s	6s	6s	80.3%
200	**100**	2s	12s	5s	24s	79.9%
1000	**100**	3s	13s	5s	**63s**	80.8%

2. táblázat – Egyes folyamatok futásideje a kliens skálázási értékének függvényében

Vizsgáljuk meg a különböző skálázási értékek esetén mért futásidőket! (2. táblázat)

- Tanítás
    - Természetesen a tanítás független a titkosítási algoritmustól, így a tanítási idő sem változott.
- Titkosítás
    - A titkosításnál a különböző esetekben csak a kezdeti kliens paraméter – g^M leképezések kiszámítási ideje különbözött, de ezt federált tanulásonként csak egyszer kell elvégezni, és ez néhány másodperc alatt meg is történt, az ezután történő titkosításokhoz ugyanannyi számítást kellett elvégezni.
- Szerver által történő súlyozás és aggregálás
    - A súlyozás és aggregálás az egy körben kiválasztott kliensek számától és a súlyozás skálázási értékétől függ, de mi csak azt az esetet vizsgáltuk, amikor 10 kliens van kiválasztva körönként, így a súlyok nagy mértékű növelése nem volt indokolt (100 súlyértéket 10 kliens között elosztva kellő pontosságot kapunk).
- Dekódolás
    - A táblázatban látható, hogy a dekódolási idő volt az egyetlen, amelyik műveletnél volt lényegi különbség a futásidőben a különböző nagyságú skálázási értékek között. A megnövekedett dekódolási időt a megnövekedett lehetséges értékek száma okozta.

Mivel a célunk az, hogy minél kevesebb mértékben csökkenjen a pontosság a skálázás következményeként, ezért a skálázási értékeket minél nagyobbnak kell választani, viszont ez a futásidőt nagy mértékben növelheti (ugyanis a dekódolási idő exponenciális növekedésű). Fontos kiemelni, hogy a fent mért adatok egy olyan modell használatával mértünk, amelynél a tanulható paraméterek száma 7850. Bonyolultabb modellek esetén ez a szám akár milliós nagyságrendű lehet, így érdemes a skálázási paraméterek közül a legkisebb olyat választani, amivel még nem csökken a közös modell pontossága, és közel valós képet ad. A kliens paramétereinél a 100-as skálázási érték volt az, amellyel a futási idő a legalacsonyabban maradt, de a pontosság nem csökkent nagy mértékben.

10. Konklúzió, jövőbeli munka

Jelen dokumentumban bemutattuk a federált tanulás fontosságát, ismertettük résztvevőit és a fontosabb tulajdonságait. Bemutattuk a közös modell degradálását és önző viselkedést célzó támadó modellt, meghatároztuk a résztvevőkre vonatkozó feltételeket. Kidolgoztunk egy védelmi mechanizmust, amely lehetőséget ad a rosszindulatú kliensek kiszűrésére anélkül, hogy bárki (beleértve az aggregálást végző szervert) megtanulná egy másik résztvevő modellfrissítését. Ehhez ismertettünk egy súlyozó algoritmust, amely segítségével növeltük a védelmet az esetleges adatmérgezéses támadások ellen. Bemutattunk egy titkosítási algoritmust is, amely a biztonságos kommunikációt segítette. A védelmi mechanizmus hatékonyságát kiértékeltük úgy, hogy az MNIST adatbázis segítségével generáltunk jó- és rosszindulatú klienseket, amelyek egy központi szerver segítségével történő federált tanulásban vettek részt.

A súlyozó algoritmus a szerver számára meghatározza, hogy egy kliens hány százalékban számítson a közös modell számításakor. Az algoritmus képes arra, hogy a kliensek által egy kör alatt tanított modellek pontossága alapján kiszűrje a rosszindulatú támadásokat. A kiszűrés olyan módon megy végbe, hogy a rosszindulatú támadók számára osztott súlyt alacsonyan tartja, amennyiben az egy körben kiválasztott kliensek között jóindulatú kliens is szerepel. A megoldás limitációja, hogy amennyiben csak és kizárólag rosszindulatú kliens lett kiválasztva a federált tanítás egyik körében, akkor a rosszindulatú kliensek között osztja szét a súlyokat, így biztosan csökkenni fog a közös modell pontossága. Jövőbeli munka lehet ezen körök detektálása. A detektálás után dönthetünk úgy, hogy kihagyjuk az adott kört – ezzel elkerülve a modell pontosságának csökkenését.

A titkosítási algoritmusnak köszönhetően a központi szerver úgy tudta súlyozni és aggregálni a kliensek által küldött modellfrissítéseket, hogy azok titkosítva maradtak. Ehhez szükséges követelmény volt, hogy a titkosítás additívan homomorf legyen. A titkosítás az ElGamal titkosításon alapszik, a műveleteket elliptikus görbén végeztük. A megoldás limitációja, hogy minden kliens azonos titkos kulccsal rendelkezik, ezért amennyiben egy kliens által küldött titkosított modellfrissítés egy másik klienshez jutna, akkor az dekódolni tudná. Jövőbeli munka lehet egy olyan titkosítási koncepció kidolgozása, amely esetén a klienseknek különböző titkosítási kulcsuk van, viszont a szerver továbbra is képes a titkosított értékeken történő súlyozásra és aggregálásra. Megemlítendő, hogy a jelenlegi titkosítási koncepció aszimmetrikus kulcsú, de nem használja ki azt a lehetőséget,

hogy a publikus kulcs megosztásával bárki titkosíthatja adatokat, a publikus kulcsot kizárólag a privát kulccsal rendelkező kliensek használják. Jövőbeli munka lehet, hogy az ebből adódó lehetőségeket felmérjük (pl.: a szerver is tudna a publikus kulcs megosztásával információt küldeni a kliensek számára).

A súlyozás és titkosítás segítségével a védelmi mechanizmust működőképesnek és használhatónak nyilvánítottuk, a titkosítást kiértékeltük futási idő szempontjából is egy egyszerű neurális hálót használva. Megfigyeltük, hogy a futási időt javarészt a dekódolási idő növelte. Ismertettük, hogy a dekódolási időt hogyan lehet minimálisra csökkenteni adott adatbázis és neurális háló esetén úgy, hogy a modell pontossága ne csökkenjen. Példákat hoztunk arra, hogy a súlyozási algoritmus javítja a modell pontosságát, és értékeltük a súlyozást egy igazságosnak tekinthető elosztás, a Shapley-érték alapján. Korrelációkat fedeztünk fel a Shapley-érték és a szerver által kiosztott súlyok között, ami alapján a súlyozási algoritmust egy megfelelő algoritmusnak tekintettük.

Irodalomjegyzék

[1] V. Mugunthan, „A layman's introduction to Privacy-Preserving Federated Learning", *Medium*, febr. 27, 2020. https://medium.com/@vaikkunthmugunthan/a-laymans-introduction-to-privacy-preserving-federated-learning-8ca0e6c73ad4 (elérés dec. 08, 2020).

[2] „Általános adatvédelmi rendelet". https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:32016R0679 (elérés dec. 09, 2020).

[3] L. Melis, C. Song, E. De Cristofaro, és V. Shmatikov, „Exploiting Unintended Feature Leakage in Collaborative Learning", *ArXiv180504049 Cs*, nov. 2018, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/1805.04049.

[4] M. Nasr, R. Shokri, és A. Houmansadr, „Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning", *2019 IEEE Symp. Secur. Priv. SP*, o. 739–753, máj. 2019, doi: 10.1109/SP.2019.00065.

[5] Rishav Chourasia, Aadyaa Maddi, Mihir Harshavardhan Khandekar, Sasi Kumar Murakonda, and Reza Shokri, *Machine Learning Privacy Meter*. Data Privacy and Trustworthy Machine Learning Research Lab, 2020.

[6] C. Beguier és E. W. Tramel, „SAFER: Sparse Secure Aggregation for Federated Learning", *ArXiv200714861 Cs Stat*, szept. 2020, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/2007.14861.

[7] Y. Liu, J. Peng, J. Kang, A. M. Iliyasu, D. Niyato, és A. A. A. El-Latif, „A Secure Federated Learning Framework for 5G Networks", *IEEE Wirel. Commun.*, köt. 27, sz. 4, o. 24–31, aug. 2020, doi: 10.1109/MWC.01.1900525.

[8] J. Guo, Z. Liu, K.-Y. Lam, J. Zhao, Y. Chen, és C. Xing, „Secure Weighted Aggregation in Federated Learning", *ArXiv201008730 Cs*, okt. 2020, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/2010.08730.

[9] S. Li, Y. Cheng, W. Wang, Y. Liu, és T. Chen, „Learning to Detect Malicious Clients for Robust Federated Learning", *ArXiv200200211 Cs Stat*, febr. 2020, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/2002.00211.

[10] J. Zhang, J. Chen, D. Wu, B. Chen, és S. Yu, „Poisoning Attack in Federated Learning using Generative Adversarial Nets", 0 2019, o. 374–380, doi: 10.1109/TrustCom/BigDataSE.2019.00057.

[11] P. Kairouz *és mtsai.*, „Advances and Open Problems in Federated Learning", *ArXiv191204977 Cs Stat*, dec. 2019, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/1912.04977.

[12] „1.1. A neurális hálózat definíciója, működése | Mesterséges Intelligencia Elektronikus Almanach". http://project.mit.bme.hu/mi_almanach/books/neuralis/ch01s01 (elérés dec. 09, 2020).

[13] J. Durán, „Everything You Need to Know about Gradient Descent Applied to Neural Networks", *Medium*, szept. 20, 2019. https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14 (elérés dec. 09, 2020).

[14] J. Brownlee, „What is the Difference Between a Parameter and a Hyperparameter?”, *Machine Learning Mastery*, júl. 25, 2017. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/ (elérés dec. 09, 2020).

[15] „Keras: the Python deep learning API”. https://keras.io/ (elérés dec. 08, 2020).

[16] R. Parmar, „Common Loss functions in machine learning”, *Medium*, szept. 02, 2018. https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23 (elérés dec. 09, 2020).

[17] S. V. and S. Das, *Deep Learning*. .

[18] J. Zou és O. Petrosian, „Explainable AI: Using Shapley Value to Explain Complex Anomaly Detection ML-based Systems”, szept. 2020.

[19] „Interpretation of machine learning models using shapley values: application to compound potency and multi-target activity predictions | SpringerLink”. https://link.springer.com/article/10.1007/s10822-020-00314-0 (elérés dec. 08, 2020).

[20] „5.9 Shapley Values | Interpretable Machine Learning”. https://christophm.github.io/interpretable-ml-book/shapley.html (elérés dec. 08, 2020).

[21] „How to Train a Model with MNIST dataset | by Abdullah Furkan Özbek | Medium”. https://medium.com/@afozbek_/how-to-train-a-model-with-mnist-dataset-d79f8123ba84 (elérés dec. 08, 2020).

[22] „MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges”. http://yann.lecun.com/exdb/mnist/ (elérés dec. 08, 2020).

[23] „tf.keras.losses.SparseCategoricalCrossentropy | TensorFlow Core v2.3.1”. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy (elérés dec. 09, 2020).

[24] „tf.keras.metrics.SparseCategoricalAccuracy | TensorFlow Core v2.3.1”. https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy (elérés dec. 09, 2020).

[25] Jakab Máté, *jakabmate98/szakdolgozat*. 2020. https://github.com/jakabmate98/szakdolgozat. (elérés dec. 10, 2020).

[26] S. Arora, E. Hazan, és S. Kale, „The Multiplicative Weights Update Method: a Meta-Algorithm and Applications”, *Theory Comput.*, köt. 8, sz. 1, o. 121–164, máj. 2012, doi: 10.4086/toc.2012.v008a006.

[27] I. Panageas, G. Piliouras, és X. Wang, „Multiplicative Weights Update as a Distributed Constrained Optimization Algorithm: Convergence to Second-order Stationary Points Almost Always”, *ArXiv181005355 Math*, jan. 2020, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/1810.05355.

[28] Z. Huang és X. Zhu, „Near Optimal Jointly Private Packing Algorithms via Dual Multiplicative Weight Update”, *ArXiv190500812 Cs*, máj. 2019, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/1905.00812.

[29] A. J. Wyner, M. Olson, J. Bleich, és D. Mease, „Explaining the Success of AdaBoost and Random Forests as Interpolating Classifiers”, *ArXiv150407676 Cs Stat*, ápr. 2017, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/1504.07676.

[30] R. Holz, J. Amann, O. Mehani, M. Wachs, és M. A. Kaafar, „TLS in the wild: an Internet-wide analysis of TLS-based protocols for electronic communication”, *Proc. 2016 Netw. Distrib. Syst. Secur. Symp.*, 2016, doi: 10.14722/ndss.2016.23055.

[31] „Homomorphic Encryption — Theory and Application | IntechOpen”. https://www.intechopen.com/books/theory-and-practice-of-cryptography-and-network-security-protocols-and-technologies/homomorphic-encryption-theory-and-application (elérés dec. 08, 2020).

[32] „ElGamal Encryption System”. https://mathstats.uncg.edu/sites/pauli/112/HTML/secelgamal.html (elérés dec. 08, 2020).

[33] O. Ugus, D. Westhoff, R. Laue, A. Shoufan, és S. A. Huss, „Optimized Implementation of Elliptic Curve Based Additive Homomorphic Encryption for Wireless Sensor Networks”, *ArXiv09033900 Cs*, márc. 2009, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/0903.3900.

[34] „Elliptic Curve Cryptography - OpenSSLWiki”. https://wiki.openssl.org/index.php/Elliptic_Curve_Cryptography (elérés dec. 08, 2020).

[35] „Elliptic curve”, *Wikipedia*. dec. 05, 2020, Elérés: dec. 08, 2020. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=Elliptic_curve&oldid=992386894.

[36] „prime192v2”. https://neuromancer.sk/std/x962/prime192v2 (elérés dec. 08, 2020).

[37] J. A. Akinyele, M. D. Green, és A. D. Rubin, „Charm: A framework for Rapidly Prototyping Cryptosystems”, 617, 2011. Elérés: dec. 08, 2020. [Online]. Elérhető: http://eprint.iacr.org/2011/617.

[38] M. Baza, M. Pazos-Revilla, M. Nabil, A. Sherif, M. Mahmoud, és W. Alasmary, „Privacy-Preserving and Collusion-Resistant Charging Coordination Schemes for Smart Grid”, *ArXiv190504666 Cs*, febr. 2020, Elérés: dec. 08, 2020. [Online]. Elérhető: http://arxiv.org/abs/1905.04666.