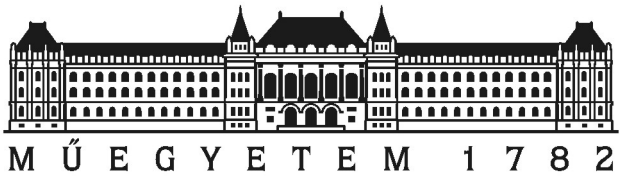



Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

# Federált bayesi tanulási módszerek

Sándor Dániel
Konzulens: Dr. Antal Péter

2020 tavasz

**Kivonat**

Ez a dokumentum a 2020 tavaszi féléves önálló laboratóriumom összefoglalóját tartalmazza. Témám a federált bayesi tanulási módszerek voltak. Ennek keretében a feladatom egy olyan megoldás megvalósítása volt, amelyben gyógyszercégek egy közös modellt taníthatnak. A rendelkezésre álló adatok különböző molekulák jellemzői voltak, ezekből kellett a molekulák számos cél tulajdonságát megjósolni. Mindeközben fontos volt, hogy az adatok nem publikusak, tehát mindenki csak a saját adatához férhet hozzá. A konkrét feladat a kimenten lévő tulajdonságok kapcsoltságának vizsgálata volt: a cél paraméterek között tapasztalati út alapján nagy összefüggések megfigyelhetők, ezért érdemes lehet olyan modellel próbálkozni, amely figyelembe veszi ezeket és ennek ismeretében képes prediktálni. Összesítve tehát olyan federált tanulás megvalósítása, ahol a kliens részen is egy futtatható modell található, amely figyelembe veszi a kimenetek kapcsoltságát.

## Tartalomjegyzék

**1. Bevezetés** **2**

**2. Federált tanulás** **2**
2.1. Federált modell átlagolás . . . . . 3

**3. Bayesi tanulás** **4**
3.1. Restricted Boltzmann Machine . . . . . 4
3.2. Hibrid módszerek . . . . . 7

**4. Architektúrák** **8**

**5. Tanítások és kirtékelés** **9**

**6. Konklúzió és jövőbeli tervek** **10**

# 1. Bevezetés

A gyógyszergyártás talán az egyik legvédettebb iparág, olyan szempontból, hogy minden gyógyszercégnek érdeke a saját információinak titkolása, nehogy egy versenytárs a kezébe kerülve felhasználhassa saját technológiái megvalósítására, ezzel hátrányba hozva az eredeti céget. Ennek ellenére minden gyógyszercég jólfelfogott érdeke lehet kooperáció bizonyos területeken, hogy közös erővel hamarabb érhessék el céljaikat. A gyógyszerkutatás erre a problémájára adhat választ a federált tanulás, amikor is az adatok titkossága mellett közös modell tanítanak, amely elérhető lesz minden résztvevő számára és segíthet a további kutatásokban.

Az egyik fontos probléma molekulák tulajdonságainak előrejelzésekor, hogy a kimenetek nem függetlenek egymástól, tehát meghatározó lehet egy kimenet egy másik szempontjából. Ez a kimeneti kapcsoltság problémája. A problémára egy jó megoldás lehet a bayesi gépit tanulási modellek családja. Ennek a két problémának vegyítésével foglalkoztam az önálló laboratóriumom során.

A dokumentum felépítése a következő: Először ismertetem a federált tanulás alapjait, egy konkrét algoritmust kiemelve, ezután rátérek a bayesi tanulásra, különös tekintettel az RBM-ekre, és az ezeket felhasználó hibrid architektúrákra. Ez után bemutatom a tervezett architektúrát és az ebből megvalósított részeket.

# 2. Federált tanulás

A federált tanulás az a folyamat, amikor több kliens közösen tanít egy modellt, (akár egy központi szerver segítségével) miközben a tanító adat decentralizált marad. [1] Törekszik az adatgyűjtés minimalizálására, és képes csökkenteni a tradicionális módszerek okozta alapvető adatbiztonsági és titoktartási kockázatokat. Leggyakoribb formája a cross-device federated learning, itt a kliensek száma nagy, elérhetőségük kicsi, erre példa okostelefonokon található személyes adatokból tanulás, pl.: prediktív gépelés funkciók relevánsabb ajánlatainak megjelenítése.

Másik széles körben elterjedt forma a cross-silo federált tanulás, itt általában kevesebb kliens van jelen (általában 2-100), de ezek megbízhatóbbak, általában nagyobb az egy kliensre eső adatmennyiség. Az önálló laboratóriumom során a cross-silo változattal foglalkoztam hangsúlyosabban, és nagyságrendileg 10 klienssel értem el az eredményeimet. A módszer használatának általában két fő oka lehet: Egy vállalat földrajzilag szeparált alegységei között nehézkes az adat megosztása, ezért kisebb kommunikációs többletet jelent modellek kommunikálása, mint az adatoké, vagy másik esetben több cég közös célból szeretne egy modellt tanítani, de az ipari titoktartás miatt nem megvalósítható az adatok aggregációja. A feladat során a második eset modellezése volt a cél. A cross-silo tanulás során fontos az adatok szeparációjának definiálása: vertikális (azonos paraméterek szerint jellemzett adatpontok vannak a klienseknél), horizontális (egy adatpont különböző jellemzői más-más klienseknél találhatók). A projektem során vertikális federált tanulást valósítottam meg.

A feladat megvalósítása pysftbent történt, ez egy generikus, adatvédelem centrikus deep learning keretrendszer. [4] Ennek segítségével kellő absztrakciós szinten lehet federált tanulást megvalósító modelleket tanítani. Ennek alapeleme a worker-nek nevezett egység, amely a számításokat végzi, ez jelen esetben a klienst fogja szimbolizálni. Főbb tulajdonsága, hogy adattal rendelkezik, és képes a modell tanításához szükséges számításokat elvégezni. A különlegessége, hogy a federáltsághoz kapcsolódó utasításokkor új objektum, úgynevezett syft tensor keletkezik, ezek láncot alkotva hivatkoznak a hagyományos torch tensorra a végén.

## 2.1. Federált modell átlagolás

Az egyik leggyakoribb federált tanulási algoritmus a federált modell átlagolása megbízható harmadik fél által. Ennek lényege, hogy a lokális SGD alapú tanítást a klienseken kombinálja egy kommunikációs résszel, ahol a szerver végez átlagolást a tanított modelleken. [2] Ennek előnye, hogy nem kiegyensúlyozott, nem IID adaton is képes tanulni, mindeközben alacsonyan tartva a szükséges kommunikációs körök számát.

Az SGD alkalmazható naivan is a problémára, ahol egy kommunikációs körben egy választott kliensen egy minibatch adat kiértékelése zajlik. Ez a megközelítés számításilag kivitelezhető, de sok kör tanításra van szükség a konvergencia eléréséhez. Ehhez képest a federált modellátlagolás során három faktor fogja meghatározni a kommunikációhoz szükséges körök számát: C a kliensek aránya amelyeket kiválasztunk az adott körben számítás elvégzésére, E a kliensen elvégzett epoch-ok száma egy körben, B a minibatch mérete, amelyet a kliensek használnak a számításhoz, B = $\infty$ esetén fullbatch-es tanításról beszélünk.

Tipikus implementációja, hogy minden kliens kiszámolja a gradienst a saját adatán, majd a központi szervernek, amely aggregálja a beérkező üzeneteket és a gradiensek átlagával módosítja a súlyokat a modellben. De könnyen belátható az is, hogy, ha minden kliens módosítja a súlyait a kiszámolt gradiensnek megfelelően, a módosított súlyok átlaga éppen ugyanazt a súlyt eredményezi, mintha a gradiensek átlagával léptünk volna a megfelelő irányba, praktikus okok miatt én ezt a módszert alkalmaztam. A két módszer közti különbséget a következő két képlet szemlélteti:

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k \tag{1}$$

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k \text{ és } w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{2}$$

Ahol t a kör száma, w a súlyvektor, k a kliens száma, $\eta$ a bátorsági tényező, g a gradiens, K az adott körben kiválasztott kliensek száma. Az algoritmus pszeudokódja alább látható.

Elméletben az algoritmus valós, nem-konvex problémákra nem garantál konvergenciát és tetszőlegesen rossz modell előállítható vele. Azonban a tapasz-

**Algorithm 1** Federált modell átlagolás

**Szerveren végzett:**
$w_0$ inicializálása
**for** $t = 1, 2, \ldots$ **do**
    $S_t$ = kliensek véletlen halmaza
    **for** $k \in S_t kliensekre$ **do**
        $w_{t+1}^k \leftarrow w_t - \eta g_k$
    **end for**
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
**end for**
**Kliensen végzett:**
**for** $epoch \in E$ **do**
    batch-ek $\leftarrow$ adat B részre ostása
    **for** $b \in batchek$ **do**
        $w \leftarrow w - \eta \nabla l(w; b)$
    **end for**
**end for**
Súlyok küldése a szervernek.

talatok azt mutatják, hogy gyakorlatban, bizonyos feltételek betartásával a modell jó eredményt képes elérni. Ilyen feltétel például, hogy a kliensek azonos kezdeti feltételekből induljanak ki, ez kellően intuitív, hiszen a gradiens irányába lépéskor feltétel, hogy azonos pontból induljanak, ekkor tud csak az átlag értelmezhető értéket felvenni.

Az algoritmus működésének fontos feltétele a megbízható harmadik fél megléte, ennek jelentése, hogy a szerverre elküldött súlymódosításokat a szerver látja, ebből pedig tud következtetni az adatra, ennek feloldására a gyakorlatban megbízható hardveres megoldásokat szoktak alkalmazni, vagy zajt kevernek a modellek kimenetére, ez azonban lassítja a konvergenciát. Megemlítendő azonban, hogy a kockázatok jelentősen csökkennek a hagyományos, nem elosztott, adatközpontban gyűjtött adatokhoz képest, még abban az esetben is, ha azok anonimizáltak.

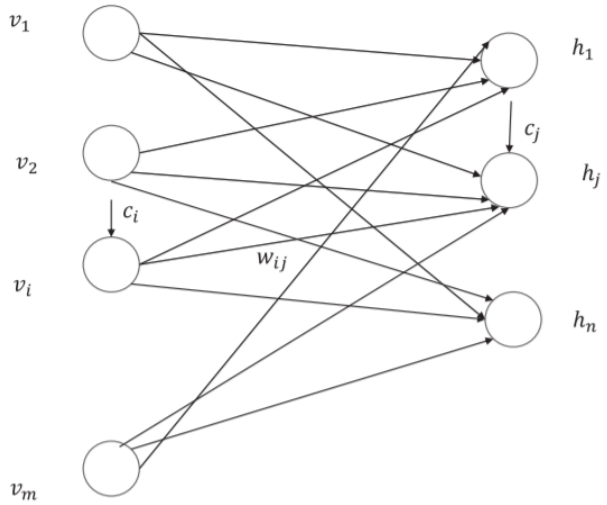
# 3. Bayesi tanulás

A bayes-i modellek a gépi tanulásnak azon osztálya, ahol a következtetést egy oly módon végzik, hogy paramétereiket egy posteriori eloszlásból kapják, melynek megalkotásához felhasználunk egy prior eloszlást, a releváns ismeretlen paramétreket és a modellből kapott likelihood függvényt.

## 3.1. Restricted Boltzmann Machine

Ezen modellekre egy jó példa a nem felügyelt tanulási algoritmusok családjába tartozó Restricted Boltzmann Machine (RBM). [3] A módszer a Boltzmann

egyenletet valószínűségi eloszláshoz használja. Az architektúra, mint a lejjebb lévő képen látszik két rétegből áll: egy látható (v, mint visible) és egy rejtett (h, mint hidden). A rétegeken belül nincsenek összeköttetések (ezért restricted), azonban mindkét réteg minden egysége össze van kötve a másik réteg minden egységével, teljes páros gráfot alkotva. Az egy rétegben elhelyezkedő egységek feltételesen függetlenek egymástól, ha adott a másik réteg. Leggyakrabban komplex architektúrák részeként használják, ld. később. Ha a rejtett réteg mérete kisebb, mint a láthatóé, akkor dimenzióredukciós tulajdonsággal bír (akár egy autoencoder vagy PCA).




Az RBM energia alapú modell, adott pillanatban (adott h és v vektorokkal) az együttes eloszlás energiája a következő képpen írható fel:

$$P(v,h) = \frac{e^{-E(v,h)}}{Z} \tag{3}$$

$$Z = \sum_v \sum_h e^{-E(v,h)} \tag{4}$$

Ahol E(v,h) az aktuális (v,h) együttes energiája, Z, pedig egy normalizáló konstans. Ez a valószínűség a Boltzmann eloszláson alapszik, mind a Boltzmann-állandót, mind a hőmérsékletet 1-nek feltételezve. Az energia megadható a következő módon:

$$E(v,h) = -b^T v - c^T h - v^T W h \tag{5}$$

Behelyettesítve a valószínűség egyenletébe:

$$P(v,h) = \frac{e^{b^T v + c^T h + v^T W h}}{Z} \tag{6}$$

Ahol b és c a megfelelő rétegbeli bias vektorok és W a rétegek közti súlyvektor. A modell lényege, hogy a fentebb leírt együttes valószínűséget több eseményre

kiszámolja. Azonban, mivel Z számítása nehéz, ezért P(v,h) számítása is nehéz lesz. Kis eseménytérre könnyű a szummákat kiszámolni, de ez a lehetőség ritkán áll fent. De a feltételes valószínűség könnyebben számolható:

$$P(h|v) = \frac{P(v,h)}{\sum_h P(v,h)} = \frac{e^{c^T h + v^T W h}}{\sum_h e^{c^T h + v^T W h}} = \frac{\prod_{j=1}^{n} e^{c_j h_j + v^T W[:,j] h_j}}{\sum_h \prod_{j=1}^{n} e^{c_j h_j + v^T W[:,j] h_j}} = \quad (7)$$

$$= \prod_{j=1}^{n} \frac{e^{c_j h_j + v^T W[:,j] h_j}}{\sum_{h_j=0}^{1} e^{c_j h_j + v^T W[:,j] h_j}} \quad (8)$$

Amely pontosan a vektor komponensenkénti feltételes valószínűség. Ha $h_j$-be behelyettesítjük az 1-et vagy a 0-t akkor adódik:

$$P(h_j = 1|v) = \sigma(c_j + v^T W[:j]) \quad (9)$$

És ugyanez v-re is levezethető:

$$P(v_j = 1|h) = \sigma(b_i + W[i:]h) \quad (10)$$

Ahol $\sigma$ a logisztikus szigmoid függvény.

Tanításkor hivatkozzunk a paraméterek együttesére a következő módon: $\theta = [b; c; W]$. Az ebből adódó likelihood függvény $L(\theta) = \prod_{t=1}^{m} P(v^{(t)}/\theta)$, ahol $v^{(t)}$-k a bemeneti vektorok. Ebből a log likelihood:

$$C = logL(\theta) = \sum_{t=1}^{m} logP(v^{(t)}/\theta) \quad (11)$$

Ebbe behelyettesítve az együttes valószínűséget:

$$C = \sum_{t=1}^{m} log \sum_h e^{-E(v^{(t)},h)} - mlogZ = \sum_{t=1}^{m} log \sum_h e^{-E(v^{(t)},h)} - mlog \sum_v \sum_h e^{-E(v,h)} \quad (12)$$

A hibafüggvény gradiense a következő képpen alakul:

$$\nabla_\theta(C) = \sum_{t=1}^{m} E_{P(h|v^{(t)},\theta)}[\nabla_\theta(-E(v^{(t)},h))] - mE_{P(h,v|\theta)}[\nabla_\theta(-E(v,h))] \quad (13)$$

Láthatóan a gradiens két tagból áll: az első a megfigyelt adat likelihoodját növeli, a másik csökkenti a modell adatpontjainak likelihoodját. Egyszerűsítéssel és a paraméterek behelyettesítésével (az energiára számolt várható értékekbe) a következő összefüggések adódnak:

$$\nabla_b(C) = \sum_{t=1}^{m} v^{(t)} - mE_{P(h,v|\theta)}[v] \quad (14)$$

$$\nabla_c(C) = \sum_{t=1}^{m} \widehat{h}^{(t)} - mE_{P(h,v|\theta)}[h] \quad (15)$$

$$\nabla_W(C) = \sum_{t=1}^{m} v^{(t)}\widehat{h}^{(t)} - mE_{P(h,v|\theta)}[vh^T] \tag{16}$$

Ezen összefüggések alapján a hálózat gradient descent-tel tanítható, az egyetlen nehézség a várható értékek számítása. Erre megoldás lehet egy MCMC mintavételezés: a Gibbs-mintavételezés. Ennek lényege, hogy a következő minta kiszámításához az összes eddigi minta eloszlását mintavételezi véletlenszerűen, ezért nem ragad be lokális régiókba. Ha m mintát akarunk generálni, akkor az előző lépést m-szer végezzük el. Ennek egy variánsa a Blokk Gibbs-mintavételezés ennek lényege, hogy csoportosítja a változókat, és együtt mintavételezhetőek lesznek, ez lesz alkalmas például a h mintavételezésére adott v mellett. Ebből a várható érték a következő módon számolható:

$$E[f(h,v)] = \frac{1}{M}\sum_{t=1}^{M} f(h^{(t)}, v^{(t)}) \tag{17}$$

Ahol M a generált minták száma P(v,h) együttes valószínűségből. A mintavételezéskor könnyebbség, hogy az azonos rétegbeli egységek egymástól függetlenek. A fenti képlet alapján tehát mindhárom várhatóérték kiszámolható, az egyetlen nehézség, hogy M mintát kéne generálni hozzá, amely számításigényes feladat. Ennek megoldására vezetik be a kontrasztív divergenciát. A kontrasztív divergencia a várható értékeket becsüli mindössze pár lépésnyi Gibbs-mintavételezés alapján. Fontos, hogy minden adatpontra elvégzi a műveletet. A gradiens képlete a következő képpen módosul tehát:

$$\nabla_\theta(C) \approx \sum_{t=1}^{m} E_{P(h|v^{(t)},\theta)}[\nabla_\theta(-E(v^{(t)}, h))] - \sum_{t=1}^{m}[\nabla_\theta(-E(\overline{v}^{(t)}, \overline{h}^{(t)}))] \tag{18}$$

RBM-et sikeresen használtak korábban Drug-Target Interaction prediktálására. [7] Ezen eredmények alapján, tehát van létjogosultsága az RBM-nek a témában.

## 3.2. Hibrid módszerek

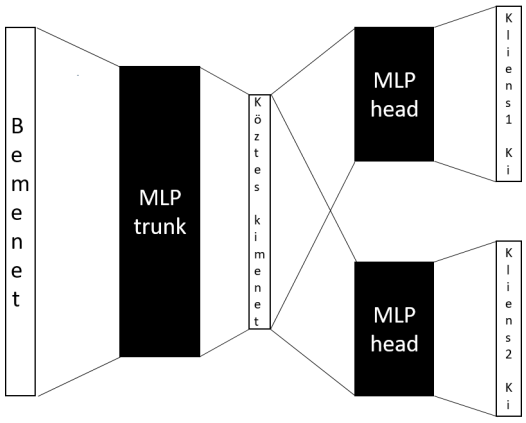
Mint azt korábban említettem az RBM-et ritkán használják önmagában, általában más megoldásokkal párosul. Egy jó példa lehet a következő tanulmány: [5], melyben egy RBM-mel kapcsolt neurális hálózattal végeztek érzelem felismerést.

A felvázolt modell egy Gaussian Mixture Modell (GMM) alapján modellezi a beszélők érzelem függő tulajdonságait. A GMM lényege, hogy több normál eloszlás lineáris kombinációjának tekinti az adatok háttéreloszlását. A beszédminták tulajdonságaiból megtanulható ez az eloszlás. Ezzel a módszerrel minden egyes érzelemhez előállítottak egy GMM modellt a beszéd kivonatolt jellemzői alapján. Az architektúra komplexitását az adja, hogy a végső osztályozást egy neurális hálózat végzi, amely bemenetnek megkapja a GMM-ekből képzett címkét, illetve a konkrét a GMM bemenetén is megadott kivonatolt jellemzőket. A GMM címke bináris értékek összessége, arra vonatkozóan, hogy a tanító adatok alapján tartozhat-e az adott érzelembe a kiértékelni kívánt minta.

Egyéb esetben használhatják akár osztályozásra [6], vagy egyszerű reprezentációtanulásra is. Én a feladatomban a kimeneti kapcsoltságok architektúrába való kódolásához használtam fel RBM-et a modellemben.

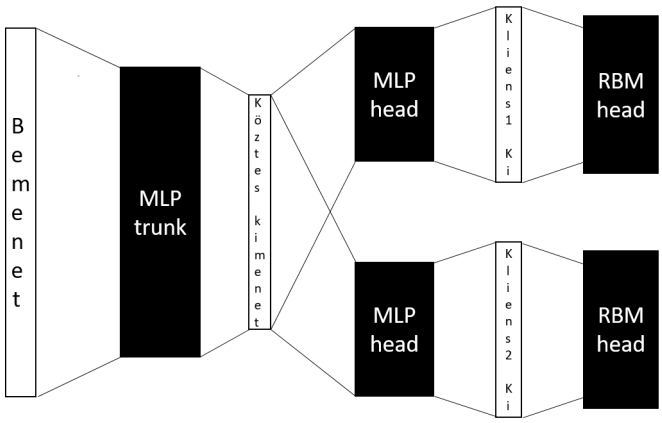
# 4. Architektúrák

A projekt alapvető architektúrája a következő képpen épül fel: a kliensek közösen tanítanak egy (trunk) hálózatot, majd mindenki az ez által előállított adatokat a saját maga (head) hálózatával értékeli ki. Tehát a tanítás közben egyszerre vagy elosztva, de két hálózatnak kell tanulnia. A trunk rész elsődleges célja a dimenziócsökkentés, hogy a nagyméretű molekula leírásokból, előállítson egy kompakt reprezentáció, amelyből a head hálózat képes lesz tanulni a célparamétereit. Mivel a reprezentációtanulás komplex feladat, ezért a modellnek ez a része amely federáltan, nagy mennyiségű adatból fog tanulni. Ezzel ellentétben a reprezentációból a célértékek előállítása könnyebb, erre várhatóan elég lesz a kliensek saját, lokális adata. A kiindulási architektúra tehát a következő volt: a trunk és a headek is egy rétegű hálózatok voltak, amelyek a bemenetről egy reprezentációt tanultak (trunk), majd ebből a kimenetet (head). A trunk minden esetben szigmoid aktivációval rendelkezett. A képen látható a kezdeti architektúra két kliensre, a feketével jelölt részek a hálózat részei, a fehérek az adatokat jelképezik.




A feladatban a célértékek hasonlók, nem tekinthetők egymástól függetlennek, ezt jelenti a kimeneti kapcsoltság. Ebből adódóan érdemes lehet olyan modellt használni, amely figyelembe veszi ezt a tulajdonságot. Az RBM, mivel irányítatlan éleket tartalmaz, alkalmas lehet erre a feladatra. Az új architektúrában tehát minden kliens head hálózatát egy lokális RBM-re cserélem. Ezen is elvégzem a tanítást, majd a hibát így terjesztem vissza a trunk hálózaton.

Látható, hogy a felhasznált head MLP-re itt is szükség van, ennek oka a dimenziók beállítása, mivel mint korábbiakból látható az RBM kimeneti dimenziója megegyezik a bemeneti dimenziójával. Ezen kívül az architektúra




felhasználható az RBM utólagos offline tanítására, ebben az esetben autoencoder jellegű szerepe lenne és a prediktált zajos értékekből kellene visszaállítani a megfelelő kimenetet.

Ezen túl, mivel az RBM alapú hibrid megoldások kezdetben nem adtak megfelelő eredményeket, megpróbálkoztam egy bottleneck-et tartalmazó head megalkotásával, amely hasonló elven működik, mint az RBM, talán kevésbé képes a kimeneti kapcsoltság architektúrális modellezésére.

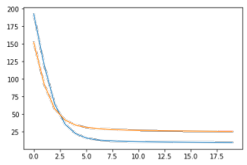
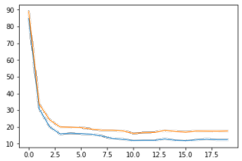
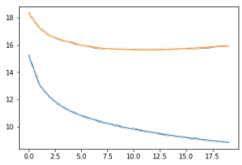
# 5. Tanítások és kirtékelés

A feladat során elsőként az alap modellemet tanítottam a 50 epochig, ebből a tapasztalat az volt, hogy az első 20 epochban volt képes tényleges javulást elérni, ezért ezután minden architektúrát 20 epochig tanítottam. Minden tanítás federált modellátlagolással történt az egy rétegű enkóder jellegű trunkon. Ezen túl mindig tíz kliensre osztottam szét egyenlően az adatot. A megvalósítás pysyftben és pytorchban készült.

Az RBM modell különleges tanítási szempontból, mivel ez nem tanítható backpropagation-nel, ezért itt speciális módszereket kell alkalmazni. Az első megoldási lehetőségem a következő volt: Minden iterációban az RBM-et az MLP kimenetén tanítottam, ezzel a módszerrel nem volt képes tanulni a modell, mivel az MLP kimenete túl gyorsan változott, amire az RBM nem volt képes rátanulni, emiatt nem volt konzisztens a kimenet sem, ami miatt az MLP sem volt képes tanulásra. A második módszer során az RBM-et az MLP-től függetlenül (de azzal felváltva) tanítottam, ez képes volt a konvergenciára, kellően jó eredményeket ért el.

A bottleneck hálózat kimenetén három réteg volt a bemenet és a kimenet is a kimeneti dimenzióval volt egy méretű, közte pedig egy kevesebb neuronból álló réteg állt. Ezt a hálózatot teljes egészében lehet backpropagation-nel tanítani, így könnyen konvergált. A három architeltúra közül ez volt képes leggyorsabban tanulásra, már a 2. epoch környékén értelmes kimenetet adott, azonban ez

a hálózat volt képes leghamarabb a túltanulásra. A gyors konvergencia oka vélhetően azonban nem az architektúra volt, hanem a rétegszám megnövelése.

A tanítások eredményeit a fenti ábra mutatja. Minden esetben a kimeneten mért négyzetes átlagos hibákat ábrázolja a kép, narancssárgával látható a teszt adaton, kékkel a tanító adaton elért hiba.

Az első ábrán az alap MLP-MLP architektúra eredménye látható, a második az RBM hibrid, a harmadik, pedig a bottleneck alapú hálózat tanulása látható. Leolvasható az ábráról, hogy az alap architektúra korai szakaszán a legmagasabb a hiba, ez nagyjából 10 epoch alatt beáll egy stabilan alacsony eredményre, a hibája, hogy a tesztadaton elért hiba nem csökken le kellő mértékben.

Az RBM architektúra szintén magasabb kezdeti hibáról indul (de jóval alacsonyabb, mint az MLP), de hamarabb konvergál, ez nagyjából 4 epoch alatt történik, azt érdemes megfigyelni, hogy itt áll a legközelebb a tanító és a tesztadaton mért hiba, tehát ez a hálózat van legkevésbé kitéve túltanulásnak, vélhetően ezt tovább tanítva is javuló eredményeket kapnánk.

A bottleneck hálózat alacsony kezdeti hibáról indul, ez vélhetően a rétegszámnak köszönhető. Gyorsan nagyon pontos eredményeket képes elérni, de ezután hamar elindul a túltanulás a hálózaton.

A modellek legjobb eredményei (az RBM és bottleneck hálózatokon korai leállást alkalmazva):

| | Tanító | Teszt |
|---|---|---|
| MLP-MLP | 9,76 | 25,5 |
| RBM hibrid | 11,83 | 16,10 |
| Bottleneck hibrid | 9,67 | 15,63 |

A táblázatból leolvasható, hogy az RBM alapú megoldás éri el a második legjobb teszt adaton mért hibát, és itt a legkisebb a tanító és teszt hibák közötti különbség. Ez alapján tehát elmondható, hogy van létjogosultsága az RBM alapú architektúráknak a kimeneti kapcsoltság vizsgálata során. Az egyetlen hátránya, hogy tovább tart a tanítása, mivel az RBM saját implementációban készült el.

# 6. Konklúzió és jövőbeli tervek

A fentiekben bemutattam a félév során elkészített munkámat. Részleteztem a federált tanulás módszertanát, bemutattam a bayesi tanulás egy lehetséges megvalósítását. Ezután ismertettem az eredményeimet különböző hálózatokon elért

teljesítmények formájában. A legfontosabb eredményem a kimeneti összefüggések prediktálására felhasznált RBM hibrid modell volt, mely az elvárásoknak megfelelően teljesített és érdemes lehet valós környezetekben is alkalmazni a fent jellemzett probléma megoldására.

A témának azonban nagyon nagy részét nem fedtem le, és sok lehetőség adódik a jövőre nézve. Egy tipikusan témába vágó feladat lenne a federált tanulás megvalósítása titkosított adatokon, ebben az esetben nem lenne szükség megbízható harmadik félre a modellátlagoláskor. Másik lehetőség, hogy az architektúra nagy szabadsági fokkal rendelkezik, ebből adódóan nem lenne fontos, hogy minden kliens ugyanazokat a kimeneteket várja el, ebből adódóan eltérő architektúrájú head-ek használata is racionális döntés lehet, kérdés azonban, hogy ez befolyásolná-e a tanulást. Ezen túl a jelenlegi architektúrán is számos továbbfejlesztés elképzelhető, egy ilyen lehetne, hogy az RBM-eket előbb tanítjuk, mint az MLP trunk-ot, ezzel gyorsítva a federált tanulás részt, illetve jobb eredményt is eredményezhet, ha a statikus RBM alapján kell a hálózatnak rátanulnia a kimenetre.

Összefoglalva tehát az RBM alapú hibrid hálózatok felhasználása a kimeneti kapcsoltság problémáját tekintve indokolt és az előzetes feltételezések beteljesültek. Ezen túl federált környezetbe is könnyen integrálhatók, kombinálható a gradiens alapú modell aggregáció módszerével.

# Hivatkozások


[1] Peter Kairouz és tsai. “Advances and open problems in federated learning”. *arXiv preprint arXiv:1912.04977* (2019).

[2] H Brendan McMahan és tsai. “Federated learning of deep networks using model averaging”. (2016).

[3] Santanu Pattanayak, Pattanayak és Suresh John. *Pro Deep Learning with TensorFlow*. Springer, 2017.

[4] Theo Ryffel és tsai. “A generic framework for privacy preserving deep learning”. *arXiv preprint arXiv:1811.04017* (2018).

[5] Ismail Shahin, Ali Bou Nassif és Shibani Hamsa. “Emotion Recognition using hybrid Gaussian mixture model and deep neural network”. *IEEE Access* 7 (2019), 26777–26787. old.

[6] Yi Sun, Xiaogang Wang és Xiaoou Tang. “Hybrid deep learning for face verification”. *Proceedings of the IEEE international conference on computer vision*. 2013, 1489–1496. old.

[7] Yuhao Wang és Jianyang Zeng. “Predicting drug-target interactions using restricted Boltzmann machines”. *Bioinformatics* 29.13 (2013), i126–i134. old.