Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Kiss Richárd, Pogány Domonkos, Sándor Dániel

IMPLICIT VISSZACSATOLÁSON ALAPULÓ KOLLABORATÍV SZŰRŐS AJÁNLÓRENDSZER MÉLY NEURÁLIS HÁLÓVAL

TDK dolgozat

KONZULENS
Dr. Szűcs Gábor

BUDAPEST, 2019

Tartalomjegyzék

Absztrakt

A mai világban a jelentős adatnövekedésnek és a szolgáltatók közötti versenynek köszönhetően megnőtt az igény az egyre jobb teljesítményű ajánlórendszerek kifejlesztésére. Ezeknek a rendszereknek a célja, hogy minél relevánsabb ajánlatokat adjon a felhasználóknak, ezzel növelve a vásárlások számát/oldalon töltött időt/felhasználói élményt.

Az ajánlórendszerek szakirodalmában alapvetően kétféle csoportosítás létezik. Az egyiknél az adatok származása alapján történik csoportosítás, itt megkülönböztetünk implicit és explicit értékeléseket / visszajelzéseket. Explicit visszajelzésnek nevezzük, amikor egy felhasználó a termék fogyasztása (használata, megnézése stb.) után értékelést (pl. pontszámot) rendel egy termékhez. Implicit visszacsatolásnak pedig, amikor a felhasználó és a termék közti interakcióból automatikusan származnak az adatok (pl. megvette-e a terméket vagy sem). Másik csoportosításnál pedig a termékekről rendelkezésre álló információk alapján választhatjuk szét a két ajánlórendszer típust: tartalom alapú szűrés, ahol a termékekhez plusz információk (ember által meghatározott tulajdonságok) is vannak, és ez alapján történik az ajánlás, illetve a kollaboratív szűrés, mely matematikai konstrukciók segítségével látens tulajdonságokat rendel a termékekhez/felhasználókhoz. A TDK dolgozatunkban az utóbbival, illetve az implicit visszajelzés típusú ajánlásokkal foglalkoztunk részletesen.

Kezdetben az úgynevezett baseline módszert használták az iparban, azaz minden felhasználónak egységesen a legjobb termékeket ajánlották. Ezután terjedtek el a hasonlóság alapú módszerek, amelyek a felhasználóknak egyedi ajánlásokat tettek, a hozzájuk hasonló felhasználók, illetve az általuk használt termékek alapján. Az első komolyabb előrehaladást a témában a mátrixfaktorizációs modellek jelentették, amelyek a felhasználókat és a termékeket is ugyanabban a látens térben reprezentálják, majd ezek cosinus távolsága alapján rangsorolják a releváns termékeket. A legfrissebb kutatások arra irányulnak, hogy neurális hálóval végezzék a látens vektorok hozzárendelését. Dolgozatunkban kitértünk a régebbi ajánlórendszerek és a legfrissebb módszerek leírására, elemzésére. Különböző pontosság metrikák alapján összehasonlítottuk és kombináltuk őket egy saját ajánlórendszer létrehozása érdekében, majd kiértékeltük a kapott eredményeket. A dolgozatunkat egy összefoglaló résszel zártuk, ahol röviden reflektáltunk a bevezetőben említettekre, azaz, hogy az ajánlórendszerek kifejlesztésére egyre nagyobb igény mutatkozik.

Abstract

Collaborative Filtering type Recommendation System by Deep Neural Network based on Implicit Feedback

In our modern world, due to the significant growth of data and the competition of service providers, the need for recommender systems with higher performance has grown. The goal of these systems is to offer the most relevant recommendations to users, thus increasing the number of purchases/ times spent on site/ user experience.

In the literature of recommender systems, two main classifications exist. In one of them, we classify based on the origin of data, and we differentiate implicit and explicit ratings / feedbacks. We call it an explicit feedback, when the user assigns a rating (e.g. a score) to the item after the interaction (consumption, use, view, etc.). At another (at implicit feedback), the data is derived from an interaction between the user and an item (e.g. purchase an item or not). In the other type of classification, we separate two types of recommender systems based on information available about items: the first is content-based filtering, when we have additional information about items (properties defined by humans), the second is collaborative filtering, which assigns latent properties to items/users based on mathematical construction. In our TDK paper, we deal with the latter one and with implicit feedback type recommendations in detail.

In the beginning, the so-called baseline method was used in the industry, and the best items were recommended for every user, uniformly. Subsequently, the similarity-based methods started to spread, which give unique recommendations to users, based on similar users and the items they used. The first major breakthrough in the field was matrix factorization model, which represents both the users and the items in the same latent space and based on their cosine distances the relevant items were ranked. The most recent studies intend to use neural networks to assign latent vectors.

In our paper we detail the description and analysis of older recommender systems and the newest techniques as well. We compare them based on different metrics, and we present our own developed recommender system combined from them and evaluated results. We finish our paper with a summary, where we briefly reflect on the introduction, namely the need for developing recommender systems has grown.

1 Bevezetés az ajánlórendszerek témakörébe

1.1 Ajánlórendszerek típusai

A mai világban a jelentős adatnövekedésnek és a szolgáltatók közötti versenynek köszönhetően megnőtt az igény az egyre jobb teljesítményű ajánlórendszerek kifejlesztésére. Ezeknek a rendszereknek a célja kettős. Egyrészt minél relevánsabb ajánlatokat kell, hogy adjon a rendszer a felhasználóknak azért, hogy segítsen nekik eligazodni a nagyon széles választékú termékek, szolgáltatások között. Másrészt ezeknek a rendszereknek célja, hogy segítsen a szolgáltatóknak eljuttatni a termékeit, szolgáltatásaikat a felhasználókhoz azzal, hogy növelik a vásárlások számát/oldalon töltött időt/felhasználói élményt. Általánosságban elmondható, hogy működésük során az információ szűrésével adnak egy előrejelzést a felhasználók értékeléseiről egy adott termékre. Ezek fontossága az utóbbi időben jelentősen felértékelődött, mivel olyan mennyiségű adat halmozódott fel a nagyobb szolgáltatóknál, hogy annak feldolgozására komplex rendszerekre van szükség [1].

Felhasználói visszajelzés szempontjából két megközelítési mód (explicit és implicit) lehetséges. Explicit visszajelzésnek nevezzük amikor a felhasználó tudatosan egy értékelést rendel a termékhez, ez megjelenhet pl. egy skálán történő értékeléssel, like gomb megnyomásával, vagy akár lista készítésével a neki tetsző termékekből. Mivel ez több felhasználói interakciót kíván meg, ezért ritkábban fordul elő, hogy ilyen explicit visszajelzésű adathalmaz rendelkezésre állna, viszont általában pontosabb ajánlások készíthetők ezt alapul véve. Implicit visszajelzésnek minősül a felhasználó minden más interakciója a termékkel, pl. megnézte, kosárba tette, megvette, rákattintott. Ezen interakciókat az adott szolgáltató könnyen követheti, nagy mennyiségben gyűjtheti és tárolhatja, ezért ezekből az adatokból több áll rendelkezésünkre, de nehezebb pontos ajánlást adni belőle.

Rendelkezésre álló adatok alapján is két módszer létezik: tartalom alapú és a kollaboratív szűrés. Tartalom alapú szűrésről beszélünk, ha a termékhez van valamilyen fajta többlet információnk, például címkék a jellemzőiről. Ekkor ezen többletinformációkat vesszük és ezek alapján konstruáljuk az ajánlásokat. Kollaboratív szűrés esetén az alap ötlet, hogy egy felhasználó értékelt egy terméket (akár impliciten is, egy hozzá köthető tevékenységgel), és ezen értékelések alapján rendel a rendszer egy hasonlóságot a termékekhez vagy a felhasználókhoz. Működése során egy felhasználó- és egy termékhalmazt, valamint az azokat értékelésre (részben) leképező függvényt használ [2] oly módon, hogy matematikai konstrukciók segítségével látens tulajdonságokat rendel a termékekhez/felhasználókhoz. A TDK dolgozatunkban az utóbbival, illetve az implicit visszajelzés típusú ajánlásokkal foglalkoztunk részletesen.

Kezdetben az úgynevezett baseline módszert használták az iparban, azaz minden felhasználónak egységesen a legjobb termékeket ajánlották. Ezután terjedtek el a példány alapú hasonlóság módszerek (más néven legközelebbi szomszéd módszerek), amelyek a felhasználóknak egyedi ajánlásokat tettek, a hozzájuk hasonló felhasználók, illetve az általuk használt termékek alapján. Ezen belül megkülönböztetünk felhasználó-, illetve termék alapú módszereket, ahogy ezt a következő alfejezetben részletesen bemutatjuk.

1.2 Hasonlóságon alapuló módszerek

A felhasználó alapú hasonlósági módszerek lényege, hogy intuitívan azt feltételezi, hogy azon felhasználók, akik eddig hasonló termékekkel léptek interakcióba, a jövőben is hasonló termékekkel akarnak foglalkozni. Tehát a feladat azon felhasználók (userek) megkeresése, akik a leginkább hasonlítanak ahhoz a felhasználóhoz, akinek ajánlást akarunk tenni. A módszer két lépésben működik: Első körben veszi az aktív felhasználót és kialakítja a klikkjét. Ennek során egy hasonlósági függvényt definiál, ami két felhasználó értékeléseit kapja bemenetként és egy hasonlósági számot ad vissza, majd ez alapján az aktív felhasználóhoz leghasonlóbb felhasználókat beveszi a klikkbe (az intuíciónak megfelelően csak a hasonló felhasználók alapján fog ajánlani). Második lépésben a klikkben szereplő felhasználók által értékelt termékekhez rendel egy számot, amely azt mutatja, hogy várhatóan mennyire értékelné az aktív felhasználó a terméket. A szám hozzárendelését egyrészt az befolyásolja, hogy az aktív felhasználóhoz mennyire hasonló user értékelte, másrészt pedig hogy a klikkbeli felhasználó hogyan (milyen pontszámra) értékelte az adott terméket. A fenti két lépést elvégezve csak egy termék halmazt kapunk a hozzájuk tartozó várható értékeléssel, ezt azonban aggregálni kell a top-N ajánlás kialakításához, amit legegyszerűbben úgy tehetjük meg, hogy az N legnagyobb várható értékelésűt ajánljuk csökkenő sorrendben, de elképzelhetők más összefűzési technikák is például a diverzitás növelése érdekében.

A másik, azaz a termék alapú szomszéd módszereket gyakrabban használják a sok felhasználójú rendszerekben. Az intuíció itt is hasonló, csak másik oldalról megközelítve: egy hasonló termékekkel interakcióba lépő felhasználó a jövőben is hasonló termékekkel szeretne interakcióba lépni. Itt is két lépésben történik a működés: Elsőként a termék klikkjét határozzuk meg, ahol a módszer ugyanaz, mint a felhasználóknál (hasonlósági függvény alapján: két termék akkor hasonló, ha azonos felhasználók értékelték minél magasabbra). Majd a módszer egy klikket formál a leghasonlóbb termékekből, és a klikkben szereplő termékek alapján történik az ajánlás.

Ezen módszerek implicit és explicit értékelésekre is jól működnek. Gyakorlatban a futási időt erősen meghatározza a hasonlósági függvény választása, itt koszinusz hasonlóságot vagy Pearson korrelációt szoktak gyakran alkalmazni.

Az első komolyabb előrehaladást az ajánlórendszerek területén a mátrixfaktorizációs modellek jelentették, amelyek a felhasználókat és a termékeket is ugyanabban a látens térben reprezentálják, majd ezek koszinusz távolsága alapján rangsorolják a releváns termékeket, ezekkel bővebben a 2.1-es alfejezetben foglalkozunk. A legfrissebb kutatások arra irányulnak, hogy neurális hálóval végezzék a látens vektorok hozzárendelését. Dolgozatunkban kitérünk a régebbi ajánlórendszerek és a legfrissebb módszerek leírására, elemzésére. Különböző pontosság metrikák alapján összehasonlítjuk a szakirodalomban található ajánlórendszereket egy saját fejlesztésű ajánlórendszerrel, majd kiértékeljük a kapott eredményeket. A dolgozatunkat egy összefoglaló résszel zárjuk, ahol röviden reflektálunk a bevezetőben említettekre, azaz, hogy az ajánlórendszerek kifejlesztésére egyre nagyobb igény mutatkozik.

2 Az ajánlórendszerek modell alapú módszerei

Kollaboratív szűrés esetén a célunk több ezer felhasználó és több ezer termék között lezajlott interakciókat vezetni és tárolni, ez alapján egy modellt építeni és a jövőbeli interakciókat predikálni. Például jó volna egy webshop esetében, ha a vásárlónak nem kellene végigböngésznie az összes terméket, hanem az ajánlásaink között megtalálná amit akart. Sőt így esetleg olyan vásárlási tranzakciók is megtörténnek, amikre a rendszer nélkül nem került volna sor, mivel a vásárló nem talált volna rá az adott termékre. A következő alfejezetekben olyan modelleket fogunk ismertetni, amelyeket ennek a feladatnak a megoldására találtak ki.

2.1 Látens feature alapú modellek implicit esetben

Az első feladat az adatok gyűjtése és tárolása, amit többféle szempont szerint is megoldhatunk [9]. Mivel felhasználók és termékek közötti interakciókat kell tárolnunk, így általában egy mátrix formájában gyűjtjük össze az adatokat. Az egyik ilyen mátrix az interakciós mátrix, a dolgozatunkban ezt R-el jelöljük. R sorai a felhasználóknak, oszlopai a termékeknek felelnek meg, az i. sorban lévő j. elem értéke megegyezik az i. felhasználó és a j. termék között lezajlott interakciók számával. Egy másik szempont szerint nem szükséges ennyire részletesen tárolni az adatokat, elegendő csak bináris formában: az interakciók tárolásának ezt a fajta mátrixát preferenciamátrixnak hívjuk, és P-vel jelöljük. Ez tehát hasonlít az előző mátrixhoz, P[i,j] = 1 ha volt interakció az adott felhasználó és termék között, 0 ha nem. Például egy filmes adatbázis esetén R mátrixban tárolnánk, hogy egy felhasználó egy filmet hányszor látott, P-ben, hogy látta e már az adott filmet.

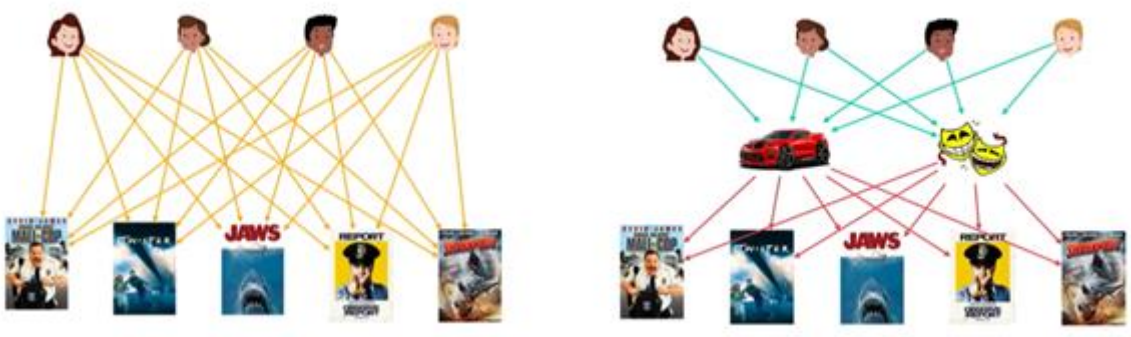
TDK dolgozatunkban kizárólag implicit visszacsatolással, foglalkoztunk, ahol egyféle adattárolási lehetőség volt a bináris információkra: a P mátrix. A cél ezen mátrix alapján egy modellt építeni, ami képes visszaadni egy hasonló mátrixot a predikált jövőbeli értékekkel. A visszaadott mátrix alapján ajánlásokat tehetünk a felhasználóknak. Például, ha egy felhasználó meglátogatja a webshop oldalunkat, akkor egyszerre nagyjából 5-10 terméket jeleníthetünk meg előtte, ezt még át tudja tekinteni. Fontos, hogy ez a pár termék összhangban legyen a felhasználó preferenciáival, vagyis olyan termékeket ajánljunk neki, amiket valószínűleg meg is fog nézni vagy venni. Egy jó modelltől elvárjuk, hogy akkor jelezzen egy termék és felhasználó között interakciót, vagyis P adott sorában és oszlopában akkor szerepeljen egyes, ha ezen termék valóban releváns a felhasználó számára.

Ilyenkor egy felhasználónak ajánlhatjuk azokat a tárgyakat, melyekre a predikált preferenciamátrixban egyes szerepel, de a valóságban még nem zajlott le közöttük interakció. Egy ilyen predikált preferenciamátrix előállításának egyik módja a mátrixfaktorizáció.

Mátrixfaktorizáció során a P mátrixot próbáljuk felírni két kisebb mátrix szorzataként [10]. A két kisebb mátrix közül az egyik a felhasználókhoz, a másik a termékekhez tartozik, a felhasználók mátrixát U-val, a termékekét I-vel jelöljük. Célunk, hogy az R vagy P-beli értékeket ezekből ki tudjuk számítani egy szorzással: $P = U^T I$. Lényegében a felhasználókhoz és a termékekhez is egy-egy k dimenziós látens térbeli vektort rendelünk.

Ezen vektorok skalárszorzata adja meg, hogy az adott felhasználó az adott terméket mennyire kedveli, vagyis interakció mennyire valószínű közöttük.

Például N darab nézőnk és M darab filmünk van, eltároljuk, hogy ki melyik filmet nézte már meg (sárga nyíl a 2.1.1. ábrán, ez felel meg a P mátrixnak). Ezután minden nézőhöz és filmhez eltárolunk egy-egy k dimenziós vektort, a nézők vektorait az U mátrixban (ábrán zöld nyilak), a filmekét az I mátrixban (piros) tároljuk. Az egyszerűség kedvéért legyen k=2, azaz – ahogy az ábrán látható – egy kétdimenziós látens térbe képezzük le az eredeti mátrixokat. Például ez a két dimenzió az "akció" és a "vígjáték", ha egy néző az akciófilmeket kedveli, de a vígjátékokat nem, akkor az akcióhoz tartozó értéke nagy, a vígjátékhoz kicsi. Hasonlóan a filmek esetében, ha egy film inkább vígjáték jellegű, akkor az akció értéke kicsi, a vígjáték értéke nagy. Szorzatuk pedig visszaadja az eredeti mátrixot. Természetesen egy valós példában nem mi választjuk meg a látens tér dimenzióit, ezeket a modell tanulja.

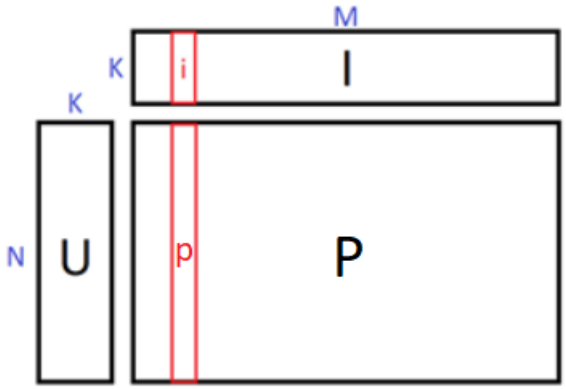


2.1.1. ábra: Mátrixfaktorizáció komplexitása

Ezzel egyrészt jelentős méretcsökkenést érhetünk el, ha a modellünkben P helyett U és I mátrixokat tároljuk csak. Egy valós esetben akár 10.000 felhasználónk és 10.000 filmünk is lehet, és ezeket például egy 100 dimenziós térbe képeznénk le. Ilyenkor P mérete 10_8 lenne, míg I és U együttes mérete csak 10.000*100*2 = $2*10_6$, vagyis a szükséges tárterület az eredeti méret 2 százaléka. A mátrixfaktorizáció ereje azonban nem a méretcsökkenésben rejlik, hanem az általánosítóképességben. Mivel a rendelkezésre álló adatok nagy részének elhagyásával próbáljuk újra előállítani őket, így a modell képes a legfontosabb összefüggéseket megtanulva reprezentálni az adatot úgy, hogy az elég általános legyen ahhoz, hogy egy új, eddig nem látott felhasználóra is illeszkedjen, és így ajánlásokat tudjon tenni.

2.2 iALS

Feladatunk a látens vektorok értékeit kiszámolni úgy, hogy a kapott szorzat minél jobban megközelítse P-t, hiszen egy ajánláshoz elég azt megmondani, hogy az adott felhasználó interakcióba lép-e egy adott termékkel, nem kell megmondanunk, hogy hányszor [9]. Az előző alfejezetben bemutatott I és U értékeit az implicit Alternating Least Squares (iALS) módszer [18][19] úgy számítja ki, hogy az egyiket rögzíti, és a másikat próbálja csak számolni P segítségével, majd fordítva. A mellékelt ábrán (2.2.1. ábra) látható, hogy amikor I egy oszlopát (i) akarjuk kiszámolni U és p segítségével, akkor egy lineáris regressziót kell megoldanunk.



2.2.1. ábra: iALS módszer látens tere

A megoldandó regresszióban Ax=b egyenlet megoldását keressük, ahol A egy N*K méretű mátrix (azaz a felhasználók U mátrixa), b egy N méretű vektor (p vektor), x pedig egy M méretű vektor (az I termékmátrix egy oszlopa), és x az ismeretlen. Ha N=K akkor 1 megoldása lehet az egyenletnek, ha N<K akkor több is, ha pedig N>K akkor az is előfordulhat, hogy nincs megoldása.

Kézenfekvő megoldás lenne az $x=A^{-1}b$, de az A mátrixnak nem mindig létezik inverze. Pszeudo inverze viszont mindig előállítható: $A^{+}=(A^{T}A)^{-1}A^{T}$. Sőt, ha egy mátrixnak létezik inverze, akkor az megegyezik a pszeudo inverzével. A célunk egy olyan x vektort találni, melyre Ax-b közel van nullához. Veszteségfüggvényünk az általános négyzetes hiba, vagyis annál kisebb a modell hibája, minél közelebb van Ax-b-hez:

$$L = ||b - Ax||^2 = \frac{1}{N}\sum_{i=1}^{N}(b_i - (A^T)_i{}^T X)^2$$

Ez egy konvex hibafüggvény, így minimumát ott veszi fel, ahol az x szerinti deriváltja nulla:

$2A^T(b - Ax) = 0$, ebből következik, hogy $x = (A^T A)^{-1}A^T b = A^{+}b$

Az iALS-nál használt hibafüggvényünk ezért hasonlít az általános lineáris regresszió veszteségfüggvényére [4].

$$L = ||UI - P||_2^2 + \lambda(||U||_2^2 + ||I||_2^2)$$

Az eltérés a $\lambda(||U||_2^2 + ||I||_2^2)$ tag, amit regularizációs céllal vezetünk be. (Ez alacsonyan tartja az U és I mátrixok L2 normáját, ami segít elkerülni a túltanulást.)

A tanítás során I-t rögzítjük és minden *u* vektornál (U mátrix szeletei felhasználónként) majd úgy változtatjuk *u*-t, hogy a részfeladat veszteségfüggvényét minimalizáljuk, majd fordítva (*i* vektorral megyünk végig I-n). Ez azért előnyös, mert ilyenkor van analitikus megoldás *u* és *i* módosításához.

$$L_u = ||p_u - uI||_2^2 + \lambda||u||_2^2 \text{ és } L_i = ||p_i - Ui||_2^2 + \lambda||i||_2^2$$

L_u rögzített I-re és u felhasználóra, L_i rögzített U-ra és i felhasználóra a hibafüggvény. Ezeket egyenlővé téve nullával, majd rendezve expliciten megkaphatjuk, hogy milyen *u* és *i* mellett lesz a hiba minimális (L_u és L_i konvexek, tehát ahol a gradiens nulla, ott lesz a minimum), ha csak *u*-t vagy *i*-t változtathatjuk.

$$\frac{\partial L_u}{\partial u} = 2(p_u - uI)I^T + 2\lambda u = 0 \Rightarrow u = p_u I^T (II^T + \lambda E)^{-1}$$

$$\frac{\partial L_i}{\partial i} = 2U^T(p_i - Ui) + 2\lambda i = 0 \Rightarrow i = (U^T U + \lambda E)^{-1} U^T p_i$$

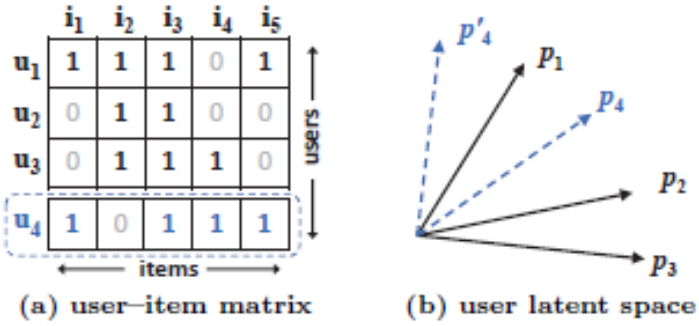
ahol *E* az egységmátrix (mérete: KxK). Látható, hogy az új *u* értékek számításánál az egyetlen tag, ami *u*-tól függ, az a p_u. Adódik a lehetőség, hogy $I^T(II^T + \lambda E)^{-1}$-t előre kiszámítsuk és eltároljuk, így nem kell újra és újra kiszámolni minden új *u*-hoz (ugyanígy $(U^T U + \lambda E)^{-1} U^T$ sem függ *i*-től). Viszont ez még mindig nem jól skálázható: A mátrix inverziót becsülhetjük $O(k^3)$-al, az $U^T U + \lambda E$ és $II^T + \lambda E$ rendre $O(nk^2)$ és $O(mk^2)$, valamint az összes *u*-ra és *i*-re az új vektor kiszámítása $O(mkn)$, *n* darab felhasználó és *m* darab item mellett. Egy tanító iteráció futtatása tehát $O(k^3 + (m+n)k^2 + mnk)$.

Kiértékelés, ajánlás

A betanított U és I mátrixokat használhatjuk arra, hogy pontos ajánlásokat tegyünk a felhasználóknak. Ezek az ajánlások a látens térbeli reprezentációk hasonlóságán alapulnak. Például ajánlhatunk a termékek hasonlósága alapján, azaz megnézzük mely tárgyak hasonlítanak leginkább a felhasználó által megvett tárgyakhoz. A hasonlóságot a két termék látens vektorának pontonkénti szorzata jelenti, esetünkben $I_i^T I_j$. Ez nem más, mint a látens térbeli vektorok által közbezárt szög koszinuszának és a vektorok hosszainak szorzata. A koszinusz hasonlóságtól ez abban tér el, hogy a vektorok nagyságát is figyelembe veszi. Ugyanakkor megmarad az a tulajdonsága, hogy a közbezárt szög minél kisebb, annál nagyobb lesz ez a szorzat, és így a hasonlóság is. Ajánlhatunk a felhasználók hasonlósága alapján is, i. felhasználónak a hozzá leghasonlóbb felhasználók által megtekintett, de általa még nem megtekintett termékeket ajánlhatjuk. U és I szorzata alapján is tehetünk ajánlásokat, egy felhasználónak javasolhatjuk azokat a termékeket, amiket a modell szerint kedvelne, de még nem lépett vele interakcióba, azaz U^TI-ben egyes de P-ben nulla.

Modell hátrányai:

Az iALS egy lineáris modell, a vektorokkal csak lineáris függvényeket végez. A hasonlóságokat a látens térbeli vektorok közötti szögek koszinusza adja. Ez nagy hátránya a modellnek, mivel így a dimenzió csökkentése miatt nem mindig tudjuk az eredeti hasonlóságot megőrizni. Tekintsük azt a P mátrixot, ami a 2.2.2. ábrán látható; itt az u_1-re jobban hasonlít u_2 mint u_3, hiszen több dimenzió mentén veszik fel ugyanazt az értéket. A másik ábrán ezeknek a vektoroknak a reprezentációja található egy kisebb, kétdimenziós térben, ezek p-vel vannak jelölve. Látszik, hogy közelebb van p_2 p_1-hez mint p_3. Ha jön egy 4. felhasználó, aki legjobban u_1-re hasonlít, de jobban hasonlít u_3-ra mint u_2-re akkor nem tudjuk elhelyezni a kétdimenziós térben úgy, hogy ezek a hasonlóság arányok megmaradjanak.



2.2.2. ábra: iALS limitációi

A másik hátrány a hibafüggvényből adódik, mivel a problémát egy regressziós problémaként próbáltuk felírni, így négyzetes hibafüggvényt kaptunk [11]. De a négyzetes hibafüggvény feltételezi, hogy a megfigyelések normális eloszlásból származnak; azonban az implicit feedback-es adatok szinte kizárólag csak bináris típusúak, és mivel nem folytonos értékkészlettel rendelkeznek, így nem alakul ki haranggörbe típusú sűrűségfüggvény az eloszlásnál. Az implicit feedback-es problémára tehát nem illeszkedik a négyzetes hibafüggvény. A hibafüggvény javításához a modellünk kimenetét is meg kell változtatnunk, ehhez bevezetünk egy predikált y-t (ez egy 0 és 1 közötti szám), amely jelentse azt, hogy mennyire valószínű, hogy interakció lenne az i. termék és az u. felhasználó között. Az implicit feedback esetén a bináris osztálycímke, az y_{ui} 1 értéket vesz fel, hogyha az *i*-edik termék releváns az *u* felhasználó számára, egyébként 0 értékű. Így a modellünk likelihood függvényét megkapjuk, ha összeszorzunk minden predikált kimeneti valószínűséget (külön szedve a releváns és nem relevánsakat), ha valóban 1 volt az érték akkor a szorzatban a predikált y_{ui}, különben 1-predikált y_{ui} szerepel, ahogy ezt a következő képletben látható:

$$p_{likelihood} = \prod_{(u,i)\in Y} \hat{y}_{u,i} \cdot \prod_{(u,j)\in Y^-} (1 - \hat{y}_{u,j})$$

Ez egy 0 és 1 közötti érték, azt mutatja meg, hogy mennyire pontos a modellünk, ezt kell maximalizálni. Könnyebb számolni viszont a likelihood negatív logaritmusát, ami már bármilyen nem negatív értéket felvehet, minél nagyobb értéket vesz fel annál pontatlanabb a modellünk, azaz ez már úgy viselkedik, mint egy hibafüggvény.

$$L = -\sum_{(u,i)\in Y} \log \hat{y}_{u,i} - \sum_{(u,j)\in Y^-} \log(1 - \hat{y}_{u,j})$$

Ha a jobboldal első tagjában minden predikált y-ját 1-el megszoroznánk, akkor nem változna az eredmény, az implicit feedback esetén a valódi osztálycímke pedig pont 1 értéket vesz fel. Így alkalmazhatjuk azt a trükköt, hogy 1 helyett y_{ui}-vel szorozzuk meg. A 2. tag esetén is használhatunk hasonló trükköt, de mivel ott a valódi osztálycímke értéke 0, így (1- y_{ui})-vel szorzunk meg mindent. Így egyesíthetjük ezeket egy szumma jel alatt mindkét részhalmazra, az L hibafüggvényünk tehát:

$$L = -\sum_{(u,i)\in Y\cup Y^-} \mathrm{y}_{u,i} \cdot \log \hat{y}_{u,i} + (1 - \mathrm{y}_{u,i}) \log(1 - \hat{y}_{u,j})$$

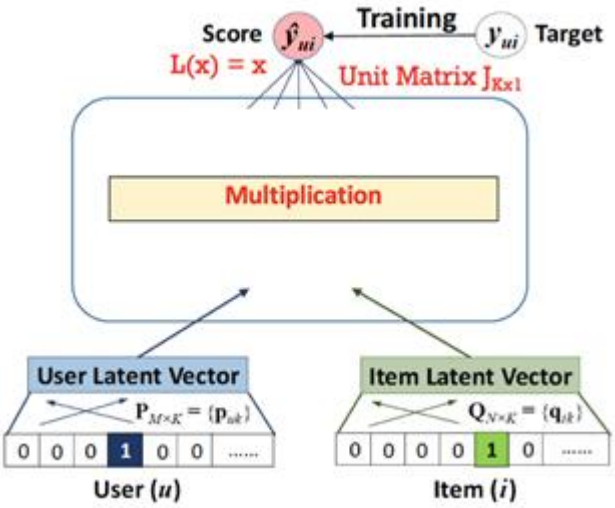
Ez pedig nem más, mint a predikált és valódi értékek bináris kereszt entrópiája, ez lesz az új hibafüggvényünk. Ez pontosabb, viszont már nem konvex, így a tanulás nehezebb lesz, nem csak egy deriválás. Lényegében a kollaboratív filtering problémát visszavezettük egy bináris klasszifikációra, amire léteznek jobb megoldások is mint az iALS.

3 Implicit visszacsatolású kollaboratív szűrés mély neurális hálóval

3.1 Generalizált mátrixfaktorizáció (GMF)

Újragondolva az iALS módszert és annak hibáit, készenállunk egy jobb mátrixfaktorizációs modell létrehozására, ami már az előbb leírt kereszt entrópiát veszi veszteségfüggénynek. A GMF [11] egy általánosított mátrixfaktrorizációs modellnek tekinthető, ami még örökli a linearitásból adódó problémákat, de már egy klasszifikációs problémaként kezeli a kollaboratív szűrést, így pontosabb eredményeket érhetünk el vele.

A klasszifikációt egy három szintű neurális hálóval oldja meg (ld. 3.1.1. ábra), ahol a hálónak 2 bemeneti vektora, és egy kimeneti skalár értéke van. A bemeneti vektorok rendre egy felhasználó és egy termék one-hot kódolással (vagyis m darab felhasználó esetén az i. felhasználót egy m dimenziós vektorral írunk le, aminek i. eleme 1, a többi pedig 0) ábrázolva. Hasonlóan az n darab terméket egy-egy n dimenziós vektorral írjuk le, a felhasználókat és termékeket ábrázoló vektorok különböző hosszúak lehetnek. A kimenet pedig egy adott felhasználó egy adott termékkel való interakciójának valószínűsége a modell szerint. Az első szinten a felhasználó és a termék (azonos k dimenziójú) látens térbeli reprezentációját állítjuk elő, ahogyan azt a hagyományos módszerrel is tennénk. Az egyes látens „feature”-höz tartozó értékeket most a bemeneti és első szintet összekötő súlyokban tároljuk, mivel a bemenet one-hot kódolással lett előállítva, így egy súly pontosan egy felhasználó (vagy termék) pontosan egy featuréhez tartozó értékét tárolja. A második szinten az első szint két látens vektorának pontonkénti szorzata található, ezért tekinthető ez is egy mátrixfaktorizációs módszernek, hiszen ugyanúgy egy látens térbe képez le, ahol a térbeli vektorok szorzata jelenti a hasonlóságot. A kimenet egy aktivációs függvény (szigmoid) kimenete, így 0 és 1 közé szoríthatjuk, azaz egy valószínűséget adhatunk vissza. A szigmoid bemenete a két k hosszú látens vektor súlyozott szorzata. Vagyis a feature vektorok egyes dimenzióinak értékeit összeszorozzuk, majd az adott dimenzióhoz tartozó súllyal megszorozzuk, ezt minden dimenzióra megcsináljuk és végül az eredményeket összegezzük. Így a kimeneti szint súlyainak segítségével az egyes feature-ök fontosságát fejezhetjük ki, ezt a hagyományos mátrixfaktorizációs módszerekkel nem tehetjük meg.



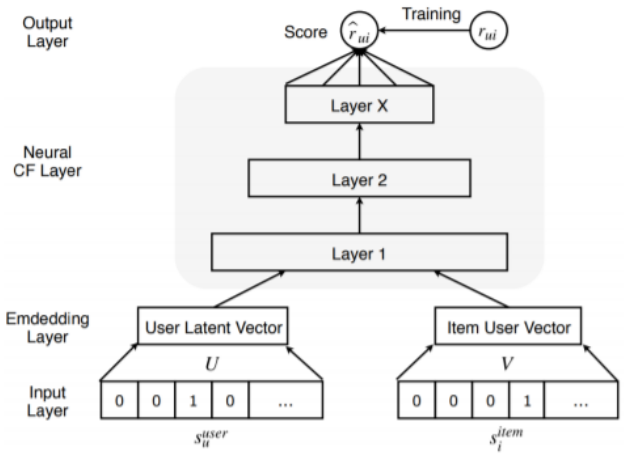
3.1.1. ábra: GMF architektúrája [14]

A modellnek három tanulható súlymátrixa van (ld. 3.1.1. ábra), a felhasználó és a termék látens térbeli reprezentációját tároló két mátrix, és az egyes látens térbeli koordináták fontosságát súlyozó kimeneti mátrix. Az összes modellbeli művelet deriválható, így tanítható backpropagation-nel, ahol hibának a kimeneti valószínűség és a tényleges érték közötti bináris kereszt entrópiát vesszük.

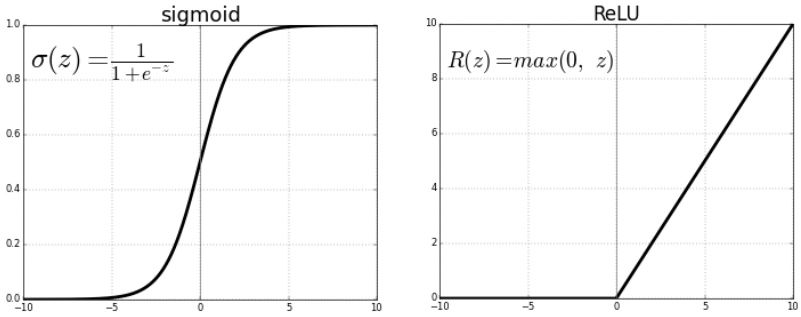
3.2 Többrétegű mély neurális háló (MLP)

Az általánosított mátrix faktorizációs módszer már valamivel komplexebb modellt képes megtanulni, mint a hagyományos módszerek. De még mindig csak a feature vektorok egy lineáris függvényét tudja előállítani. Ahhoz, hogy a modellünk magasabb rendű összefüggéseket is képes legyen felfedezni, több szintű architektúrára és nemlineáris aktivációs függvényekre lesz szükségünk.

Az Multi-Layer Perceptron (MLP) [11] modellnek is ugyanaz a két bemeneti vektora, és az MLP is egy 0 és 1 közötti számot ad kimenet gyanánt. Az első szinten az MLP modellben is a felhasználók és termékek reprezentációja található, amit ugyanúgy állít elő, mint a GMF modell, de nem feltétlenül egyezik meg méretben vele. A második szinten a pontonkénti szorzás helyett a két feature vektort konkatenáljuk, ez lesz a következő szint bemenete. Ezután a torony architekúra szerint (ld. 3.2.1. ábra) egymás fölé helyezünk rétegeket úgy, hogy azok mérete fele akkora legyen, mint az alattuk levő. A fentebbi, kisebb rétegek absztraktabb tulajdonságokat képesek megtanulni. A rétegek között ReLU [15] aktivációs függvényt használunk a gyorsabb konvergencia érdekében. Vagyis, ha egy réteg kimenete pozitív, azt megtartja, ha negatív azt lenullázza, ezzel érjük el a tanuláshoz szükséges nemlinearitást. A bemenettől függetlenül állandó a gradiense (a pozitív szakaszban), ellenben a szigmoiddal, aminek a lineáris szakaszában (a 3.2.2. ábrán, ha z kisebb mint -5 vagy nagyobb mint 5) nullához közeli a gradiense, ami jelentősen lassítja a tanulást. Az utolsó, azaz a kimeneti réteg aktivációs függvénye pedig egy szigmoid, mert ez 0 és 1 közé tudja szorítja a kimeneti értéket, amit tekinthetünk a bemenetként kapott felhasználó és termék közötti interakció megvalósulásának valószínűségének. Ez a modell is hasonlóan tanítható a backpropagation módszerével.



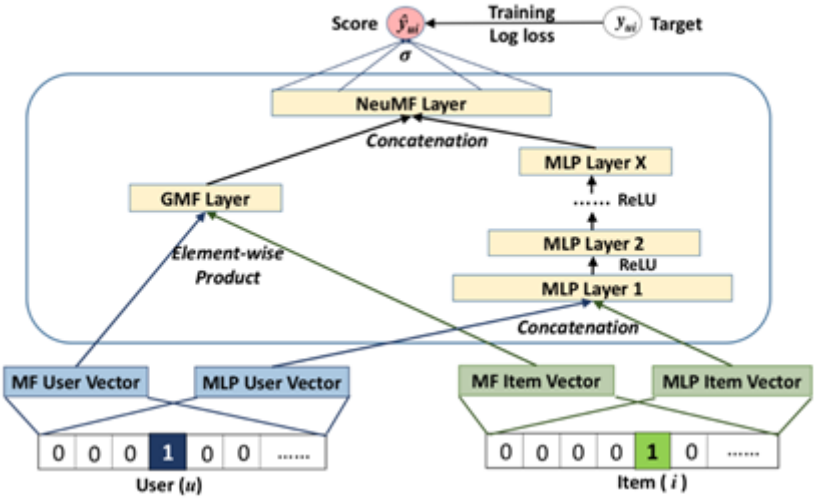
3.2.1. ábra: MLP architektúrája



3.2.2. ábra: Szigmoid és ReLU függvények

3.3 GMF és MLP fúziója: NCF

A lineáris és a mélyebb összefüggéseket kereső modellek ötvözésével még az előzőeknél is jobb eredményt érhetünk el. A Neural Collaborative Filtering (NCF) [11] modell tartalmazza a GMF és az MLP modellt (ld. 3.3.1. ábra). A két modell inputja azonos, ez lesz az összefésült modell inputja is. Lehetnének a látens vektorok is közösek, de a GMF viszonylag nagy feature vektor használatakor ad pontosabb eredményeket, míg az MLP esetén a számítási kapacitás miatt célszerűbb kisebb vektorokat használni. A két modell kimenetét konkatenáljuk, és fölé helyezzük a kimeneti szintet (NeuMF réteg), aminek aktivációs függvénye szigmoid, kimenete az adott felhasználó és termék közötti interakció valószínűsége. A legfelső mátrix segítségével tanulhatja meg a modell a GMF és MLP rész kimenetét súlyozni.

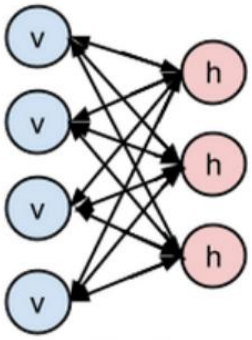


3.3.1. ábra: Fúziós modell architektúrája [14]

A célfüggvény nem konvexitása miatt a modell könnyen megragadhat egy lokális minimumnál, ezen javít valamelyest, ha a súlyokat egy előtanulási fázison vetjük át a végleges tanulás előtt. Vagyis a két modellt egymástól függetlenül tanítjuk, amíg egy megoldáshoz nem kezdenek el konvergálni. Ehhez az Adam [13] optimalizálást használjuk, a gyorsabb konvergencia érdekében. Ez a tanulási rátát a modell minden paraméterére külön adaptívan változtatja az egyes gradiensek irányának, nagyságának és momentumának megfelelően. A momentum segítségével csökkenthető a lokális minimumban ragadás esélye, az adaptivitás pedig gyorsítja a tanítást, ha szükséges, a végén pedig lelassítja azt, így elkerülve az optimum körüli oszcillációt. Ezt követően az előre betanított súlyú háló egészét tanítjuk. Adam helyett itt már csak Stohastic Gradient Descent-et (SGD-t) [16] alkalmazhatunk, mert nem tudunk momentumot számolni, mivel az egyes súlyok frissítéséhez szükséges információk, vagyis az előző lépés gradiensei nem állnak többé rendelkezésünkre az előtanítás utáni egyesítés miatt.

3.4 Restricted Boltzmann Machine

A Restricted Boltzmann Machine (RBM) [5][6][8] energia alapú modell a BM (Boltzmann Machine) egy speciális esete, melyben a látható (visible, $v \in \Re^m$) és rejtett (hidden, $h \in \Re^n$) rétegek között akkor és csak akkor van összeköttetés, ha azok nem azonos halmazból valók (hidden vagy visible), azaz a neuronok teljes páros gráfot alkotnak (ld. 3.4.1. ábra). A megkötés következtében a tanításnál lehetőségünk adódik majd egy hatékonyabb algoritmust használatára.



3.4.1. ábra: RBM architektúrája

Egy RBM-et egy $W \in \Re^{m \times n}$ mátrix, egy $a \in \Re^{m}$ oszlopvektor és egy $b \in \Re^{n}$ oszlopvektor alkotják, ahol W a súlymátrix, a és b pedig ofszet vektorok. Energia alapú modelleknél energiát (egy skalárt) rendelünk a modell változóinak minden konfigurációjához. A célunk az, hogy a megfigyelt adatokra minimális legyen a modell energiája. Ezzel azt érjük el, hogy a megfigyelt adathalmazhoz "hasonló" adatokra a modell konfiguráció valószínűsége nagy lesz, míg a tanító adatokhoz "nem hasonló" adatokra kicsi (a valószínűség fordítottan arányos az energiával, mint az később látszik). Az RBM energiáját egy (v, h) párra a következő függvény adja meg:

$$E(v, h) = -a^T v - b^T h - v^T W h$$

Az RBM-ek esetén a közös valószínűségi eloszlás egy-egy v és h-ra a következő:

$$P(v, h) = \frac{1}{Z} e^{-E(v,h)},$$

ahol Z egy normalizáló konstans (az eloszlásnak kiintegrálva 1-et kell adnia), melynek értéke $e^{-E(v,h)}$, az összes (v, h) konfigurációra: $Z = \sum_{(v,h)} e^{-E(v,h)}$

Hasonlóan a valószínűsége egy bizonyos v vektornak az összes lehetséges h-val vett valószínűségek összege:

$$P(v) = \frac{1}{Z} \sum_{h} e^{-E(v,h)}$$

Nagy v-k és h-kra $P(v)$ és Z kiszámítása rendkívül nehéz lehet, viszont, mivel az RBM-ek teljes páros gráfok, a h-k kölcsönösen függetlenek adott v-kre és fordítva, ezért $P(v|h)$ és $P(h|v)$ könnyen számítható. A feltételes valószínűségek a következőképp alakulnak:

(a) $P(h|v) = \prod_{j=1}^{n} P(h_j|v)$, hasonlóan (b)$P(v|h) = \prod_{i=1}^{m} P(v_i|h)$

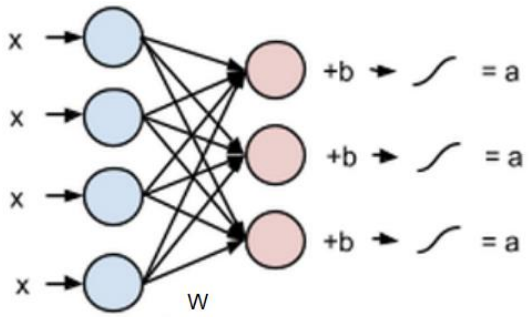
Továbbá:

(a)$P(h_j = 1|v) = \sigma(b_j + \sum_{i=1}^{m} \omega_{i,j} v_i)$, hasonlóan (b) $P(v_i = 1|h) = \sigma(a_i + \sum_{j=1}^{n} \omega_{i,j} h_j)$

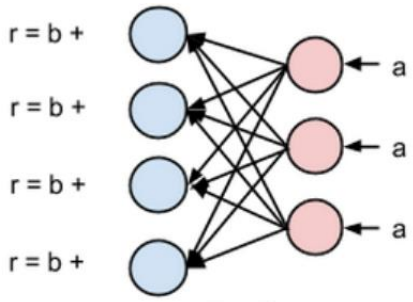
ahol a σ a szigmoid függvény. Mivel v és h vektorban az elemek függetlenek egymástól, így (a) és (b) egyszerre számolható i, j összes értékére, egy mátrixműveletbe szervezhető, amit párhuzamosítás segítségével gyorsabban elvégezhetünk, mintha ciklusba számolnánk elemenként:

(a) $h' = \sigma(W^T v + b)$, hasonlóan (b) $v' = \sigma(W h + a)$

ahol σ-t a vektor minden elemére külön végezzük el. Az alábbi két ábrán (3.4.2. és 3.4.3. ábrák) ezt a két műveletet látjuk:



3.4.2. ábra: RBM valószínűség 1. [8]



3.4.3. ábra: RBM valószínűség 2. [8]

RBM tanítása

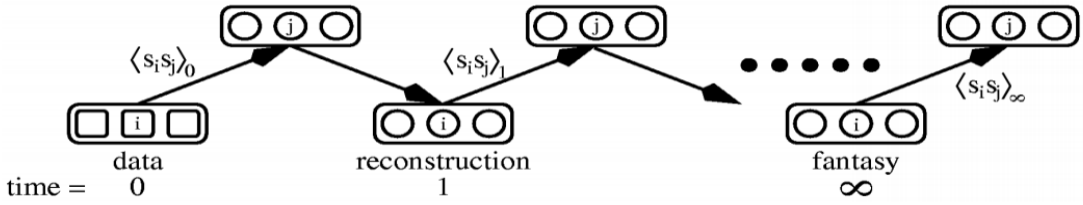
Az RBM tanításakor azt szeretnénk, ha a súlyokba kódolva lennének a látens tulajdonságok, ezáltal relevánsabb ajánlatokat tudunk adni; ezt egy gradiens alapú algoritmussal érjük el. Ilyenkor az input adat és rekonstruált adatra eloszlásokként tekintünk. A két eloszlás távolságát a *Kullback Leiber Divergencia* alapján határozzuk meg, ami a két eloszlásgörbe nem átfedő területeit adja meg:

$$D_{KL}(P||Q) = \sum_{x \epsilon X} P(x) log(P(x)/Q(x))$$

A tanítás során ezt a távolságot minimalizáljuk az összes input adatra, hogy a rekonstrukció után minél közelebbi eloszlást kapjunk az eredetihez (minimalizáljuk az energiát). A modell egyensúlyi eloszlását (P_∞-t) szeretnénk egyenlővé tenni a tanító adathalmaz eloszlásával (P_0-al), kis átalakítás után:

$$D_{KL}(P_0||P_\infty) = \sum P_0 log(P_0) - \sum P_0 log(P_\infty)$$

Ezzel az a probléma, hogy a $\sum P_0 log(P_\infty)$ tag kiszámításához egy végtelen hosszú *Markov Lánc*-ot (ld. 3.4.4. ábra) kell kiszámítanunk.



3.4.4. ábra: RBM Markov lánc [7]

Kontrasztív Divergencia

Geoffrey E. Hinton ötlete [7] az, hogy nem ezt, hanem a $D_{KL}(P_0||P_\infty)$ és $D_{KL}(P_1||P_\infty)$ közötti különbséget (ez a loss függvény) minimalizáljuk, ahol P_1 a modell eloszlása az első *Gibbs* iteráció után. Gibbs iterációnak vagy lépésnek nevezzük amikor veszünk egy inputot, v-t, abból az eloszlásból mintavételezünk egy h-t és a h-ból kiszámoljuk v'-t. Ekkor $\sum P_0 log(P_\infty)$ helyett elég $\sum P_0 log(P_1)$-et kiszámolnunk, tehát egyetlen *Gibbs* iteráció elég.

Mivel a P_1 eloszlás közelebb van P_∞-hez, mint a P_0 eloszlás, a különbsége a két *KL-divergenciának* nagyobb, mint nulla. Tehát ennek a különbségnek a minimalizálásával sok iteráción keresztül ugyanazt a hatást érjük el, mintha a P_0 és P_∞ közötti *KL-divergenciát* minimalizálnánk.

A modell frissítése

A loss függvényből gradienst számolva a modellparaméterek szerint a következőket kapjuk:

$$\Delta W = \varepsilon(vh^T - v'h'^T)\ \Delta a = \varepsilon(v - v')\ \Delta b = \varepsilon(h - h')$$

ahol megfelelő ε tanulási rátával szorzunk, hogy megfelelő konvergenciasebességet kapjunk és elkerülhessük az oszcillálást.
Tehát a tanítási algoritmus:

1) Vegyünk egy inputot, v-t, ebből az eloszlásból mintavételezünk egy h-t.
2) h-ból kiszámoljuk v'-t, majd ebből h'-t.
3) Kiszámoljuk a gradienseket, majd frissítjük (W, a, b)-t.
4) Ha a megállási feltétel teljesül, GOTO 1) egy másik inputtal, különben leállunk.

Leállási feltétel lehet:

- Egy meghatározott epoch szám, mely esetén a modell a tanító adathalmazon ennyiszer megy végig.
- Egy meghatározott hiba, ami ha kisebbet kapunk, akkor megállunk.
- *Early stopping*, ha a validáló adatokon mért hiba egy bizonyos számnál (*patience*) több epochon keresztül nő, leállítjuk a tanítást és visszatöltjük *patience* epochal előbbi állapotra. Ez azért szükséges, mert, ha a tanító adathalmazon csökken, a teszt halmazon nő a hiba, az a túltanulás jele. A visszaállítás nagyon fontos, különben a túltanult modellt fogjuk használni.

Ajánlás RBM-mel

Mi az RBM-et ajánlórendszerhez használtuk, így egy felhasználónak való ajánláskor a következő lépéseket tettük:

1) A felhasználó vektorából kiszámoltuk a rejtett rétegbeli aktivációkat.
2) A rejtett rétegből visszafele kiszámoltuk a látható réteg aktivációját (lényegében egy Gibbs lépést hajtottunk végre).
3) A kapott v' vektornak az elemeit csökkenő sorrendbe rendeztük az ajánlás elkészítéséhez.
4) Azokat a termékeket, amikkel már interakcióba lépett a felhasználó: eldobtuk az ajánlatok közül, így készítettük a végső ajánlást.

4 Implementáció és kiértékelés

4.1 Jósági indikátorok

Egy modell elkészítése során több jósági metrikát is érdemes számításba venni, mivel lehet, hogy egyes szempontok alapján jól fog teljesíteni, míg másikon nem. A mérőszámok jelentésével tehát azért érdemes megismerkedni, hogy a végén átfogó képünk alakuljon ki az adott modell teljesítményéről [3].

4.1.1 Keresztvalidáció

A modellek jóságának mérésére az alapvető módszer a keresztvalidáció. A keresztvalidáció lényege, hogy egy modell pontosságát nem mérhetjük vissza azon adatokon, amelyeken tanítottunk, mivel ez túlzottan optimista képet eredményezne. Ehelyett elkülönítjük az adataink egy részét validációra, és ezeken „a még látatlan" adatokon ellenőrizzük a modellünk pontosságát. Ennek az adatfelosztásnak több módja is lehetséges [3].

A **hold-out** egy adatfelosztásra hagyatkozik, tehát az adatok egyik részén tanítjuk és a maradék részén végezzük a kiértékelést. Ennek a módszernek a veszélye, hogy nem minden adat vesz részt a sem a tanításban, sem a kiértékelésben.

A **v-fold** keresztvalidáció során az adatokat v részre osztjuk és sorban minden v-1 részen tanítjuk, majd a maradékon egyből kiértékeljük. Ez túlhaladja a hold-out módszert abban a tekintetben, hogy itt már minden adat részt vesz a tanításban és a kiértékelésben is, azonban még ezt is a részleges adatosztás kategóriájába soroljuk, mivel nem minden adatkombináció vesz részt a tanításban és a kiértékelésben. Aránylag nagy pontossága és kedvező komplexitása miatt mi ezt a módszert használtuk az ajánlórendszerek kiértékelésére.

A **leave-p-out** típusú, keresztvalidáció lényege, hogy minden adatkombináció részt vesz a tanításban, tehát ha n adatunk van, akkor n-p adaton tanítunk, majd a maradékon értékeljük ki, de minden lehetséges n-p adatkombinációra sor kerül egy-egy tanítás során. Ezért ezt kimerítő adatosztásnak nevezzük, a legpontosabb eredményt ez fogja adni várhatóan, de számítási komplexitása miatt ritkán alkalmazott.

A **leave-one-out** egy speciális (leave-p-out esetén p=1, vagy v-fold esetén v=n) esete a keresztvalidációnak, amikor minden egyes adat ponton visszamérjük a többivel tanított modellt.

4.1.2 Pontossági mértékek

A **Root Mean Square Error** (RMSE), egy olyan pontosság metrika, amelynek lényege, hogy mérje a jósolt és a tényleges értékek közti különbséget. Működése során a két érték eltérését négyzetre emeljük, majd kiátlagoljuk, végül gyököt vonunk belőle.

Hasonló a **Mean Absolute Error** (MAE), amely a hiba abszolútértékét veszi, majd azon átlagol. Az RMSE elsősorban a négyzetre emelés miatt a nagyobb hibákat hangsúlyozza ki, így elsősorban olyan környezetekben használjuk, ahol fontos, hogy a nagy és kis hibák súlyai elkülönüljenek. Egyszerűen implementálható, tanítás közben hibafüggvénynek is

hatékonyan alkalmazható. Hibája az ajánlórendszerek témakörében, hogy az alacsonyra és a magasra rangsorolt termékek esetén a hibát ugyanakkora súllyal veszi figyelembe, ami nem kívánatos, mivel a felhasználót csak a magasra értékelt termékek hibái fogja érdekelni.

A **Precision** (pontosság) és a **Recall** (fedés) metrikák, amelyek a jósolt értékek eltérése helyett, az ajánlás relevánsságára koncentrálnak. A Recall a teszthalmazon megadja, hogy az ajánlott és egyben releváns elemek hány százalékát adják az összes releváns elemnek.

$$Recall = 100 \cdot \frac{|T_i^x \cap \tau P_i^x|}{|T_i^x|}$$

A Precision ezt megfordítja, és azt vizsgálja, hogy az ajánlott és egyben releváns elemek hány százalékát adják az ajánlatnak, azaz a teljes ajánlási listának.

$$Precision = 100 \cdot \frac{|T_i^x \cap \tau P_i^x|}{|\tau P_i^x|}$$

A **Discounted Cummulative Gain** a sorrendezés jóságára ad egy mérőszámot, mely az egyik legelterjedtebb indikátor az ajánlórendszerek és a keresőmotorok területén. Az alapja a Cummulative Gain, amelyet egy top-N ajánlásra számolhatunk az ajánlások relevánsságának összegéből. Ennek használatához szükség van megfelelő relevánssági függvény definiálására (ahol rel_i az i-edik pozícióban levő elemről adja meg, hogy releváns-e vagy sem: ha releváns, akkor értéke 1, különben 0).

$$CG_p = \sum_{i=1}^{p} rel_i$$

Ezt a DCG annyiban viszi tovább, hogy a listán elfoglalt hellyel súlyozza logaritmikusan.

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2(i+1)}$$

4.2 Teszteléshez használt adathalmaz

A modelleket a *retailrocket* [17] adathalmazon próbáltuk ki. Ezeket az adatokat 4 és fél hónapon keresztül gyűjtötték egy valós online kereskedelemmel foglalkozó szolgáltatásból. A felhasználók a következő tranzakciókat hajthatják végre: megnyitás (a felhasználó rákattint a termék adatlapjára), kosárhoz adás és vásárlás. Mi csak a megnyitást vettük figyelembe, a továbbiakban ezt hívjuk interakciónak. Az adatok egy listában érhetők el, amiben az szerepel, hogy melyik felhasználó milyen termékkel milyen tranzakciót hajtott végre és mikor. A mi esetünkben a tranzakció végrehajtásának ideje nem releváns. Ez a formátum a modellek tanítására nem megfelelő, ezért egy mátrixot konstruáltunk (ez a P mátrix) a rekordokból. Ebben a mátrixban egy sor egy felhasználót reprezentál, míg egy oszlop egy terméket. Egy felhasználó és termék párosra a mátrixban 1-et írunk, ha volt interakció a kettő között, különben 0-át. Az adatok között nagyon sok olyan felhasználó, illetve termék szerepel, ami túlságosan kevés interakcióban található,

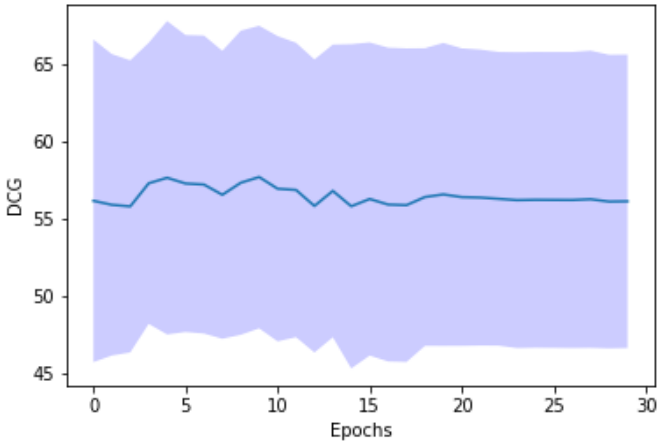
ezért szűrtük az adatokat a következő módon: Először kiválasztottuk azokat a felhasználókat, akiknek legalább 100 interakciójuk volt, majd a kapott adathalmazból kiszűrtük azokat a termékeket, amiknek az előfordulása 50-nél kevesebb. Erre azért van szükség, hogy sűrűbb legyen az adathalmaz.

4.3 Implementáció: paraméterválasztás

Az előbb bemutatott elméleti modellekben nem esett szó még a tanulás egyik legfontosabb összetevőjéről, a hiperparaméterek megválasztásáról. Ezek a modell fix, nem tanulható paraméterei, ilyenek például a regularizációs együttható, a batch méret, egy neurális hálóban a rétegek száma, elrendezése. Ahhoz, hogy ezeket helyesen be tudjuk állítani, implementálni kell a modelleket különböző hiperparaméterekkel, és össze kell vetni a tanulás sikerességét. Egyes hiperparaméterek megválasztásához léteznek jól bevált módszerek, heurisztikák, például a batch méret általában 32 szokott lenni. Vannak paraméterek, amiknek megválasztása az adott feladattól függ, ilyen például a látens vektorok mérete, természetesen ezeknek is van ésszerű alsó és felső korlátja, de közöttük nem tudni melyik értéknél lesz a háló a leghatékonyabb. Például RBM esetén a rejtett réteg mérete ne legyen 20-nál nagyságrendekben kisebb, mert abból nem lesz képes a háló újra generálni egy több száz dimenziós vektort.

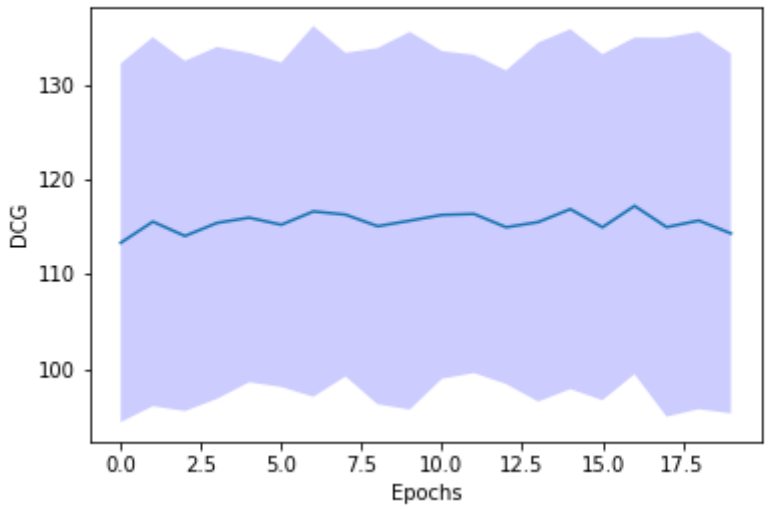
iALS:

A modell paraméterei a látens tér dimenziója, a hibafüggvényben szereplő regularizációs lambda paraméter (ez minél nagyobb, annál erősebb a regularizáció), illetve az epochok (iterációk) száma. A mérések során a felhasználók és termékek száma százas nagyságrendben mozgott, így a látens vektorokat 20 dimenziósra állítottuk. 30 epochig tanítottuk a rendszert, 5 méretű k-fold keresztvalidációval mértük vissza a modell teljesítményét, azaz ötször tanítottuk úgy, hogy a tesztadatok az összes adat ötödét tették ki, az öt lefutás alatt változott, hogy melyik a teszt és tanulóadat. Az így kapott eredményeket ábrázoltuk a diagrammokon, az átlag egy görbével (vastag vonallal), a szórás a körülötte lévő sáv átmérőjével egyezik meg. Lambdának először 0.1 értéket adtuk, de ez még nem volt elég a túltanulás eliminálására, kipróbáltuk 10 és 100 értékkel is, kiderült, hogy az erős regularizáció (lambda=100-nál) ugyan megoldja a túltanulás problémáját, de az iALS esetén jelentősen csökkenti a pontosságot. Így a legjobb iALS kialakítása érdekében a lambda paramétert 10-nek választottuk és az összes mérést ezzel végeztük. Az iALS módszerrel elért DCG értékeket a 4.3.1. ábrán láthatjuk.



4.3.1. ábra: iALS módszer DCG értékei

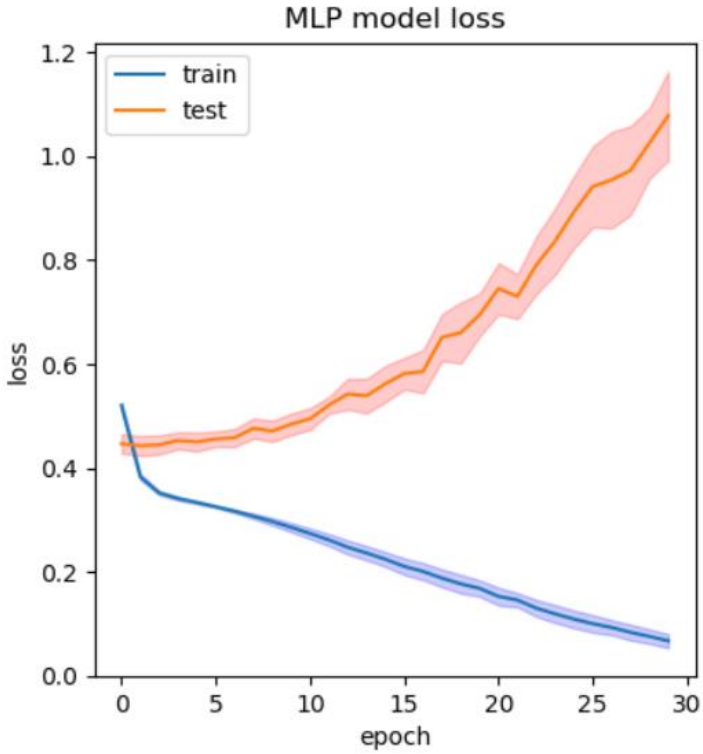
GMF:
A modell egyetlen paramétere a látens vektorok hossza, most is 20 hosszú vektorokat használtunk, hogy összehasonlíthassuk az iALS teljesítményével. A tanulás hiperparaméterei a batch és epoch mérete, előtanulás során 256 méretű batchekkel dolgoztunk, azaz a súlyok frissítéséhez szükséges gradienst 256 adat hibáinak átlagából számoltuk. Összesen 20 epochig tanítottuk a hálót, vagyis a háló minden tanulóadatot összesen 20-szor látott. A 4.3.2. ábrán látható, hogy a GMF modell jobb DCG eredményeket ér el ugyanazon az adathalmazon.



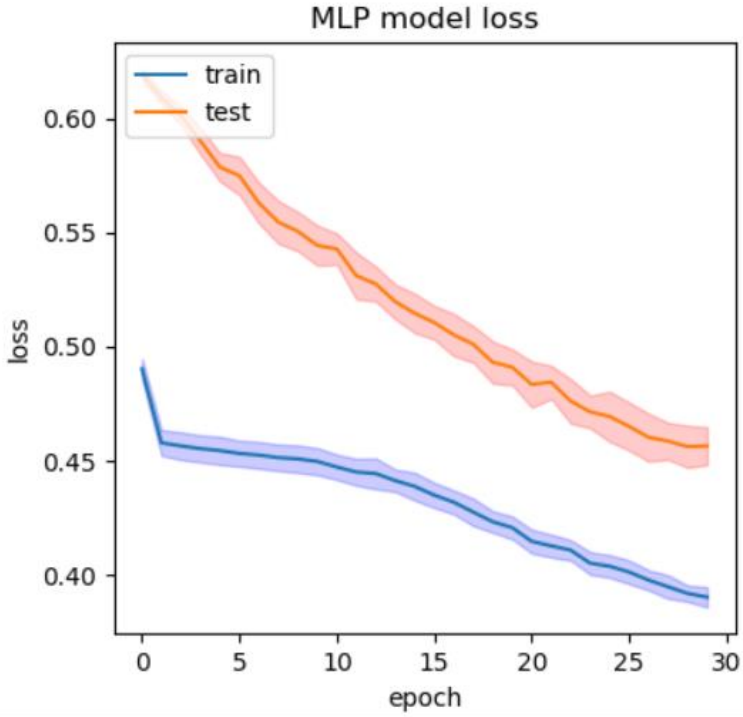
4.3.2. ábra: GMF módszer DCG értékei

MLP:
A modell architektúrális hiperparaméterei a látens vektorok hossza és a szintek száma, ez egyértelműen meghatározza a háló szerkezetét. Itt is 20 hosszú látens vektorokat használtunk, ezeket konkatenálva kaptuk meg az MLP legalsó rétegét, erre kerül rá egy 64, majd egy 32, ezután egy 16 és végül egy 8 neuronból álló réteg, összesen 5 rétegű az architektúránk. Ezt a modellt is 256 batch mérettel tanítottuk.

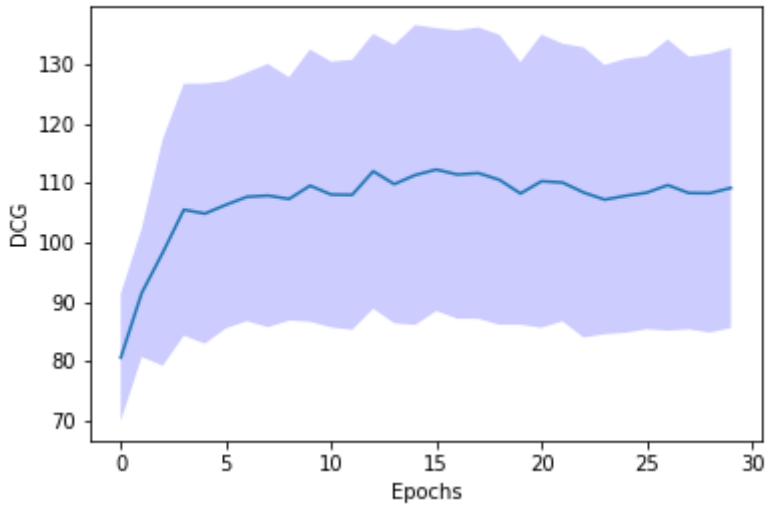
A cikkben [11] leírtak alapján tanítva az MLP is jelentősen túltanult, ez látható az 4.3.3. ábrán, ahol a tanítás során mért loss függvényt ábrázoltuk a tanuló és egy elkülönített teszt állományon. A probléma megoldására több módszert is próbáltunk, az első a (már az iALS-nál is használt) regularizáció, a második a dropout réteg bevezetése volt (azaz minden epochban néhány véletlenszerű neuront “kivesz” a hálóból, és a hozzájuk tartozó súlyok nem számítanak a következő réteg aktivációjába). A harmadik és egyben legjobb eredményeket az úgynevezett Gauss zaj bevezetésével értük el (4.3.4. ábra), ezt úgy tettük, hogy a háló konkatenáló rétegéhez hozzáadtunk egy 0 várható értékű és 2 szórású Gauss eloszlású zajt, ez elég volt ahhoz, hogy a háló általánosítson, és ne a tanulóadatra tanuljon rá. Végül az MLP módszerrel elért DCG értékek a 4.3.5. ábrán láthatók.



4.3.3. ábra: MLP tanítás a Gauss-zaj nélkül



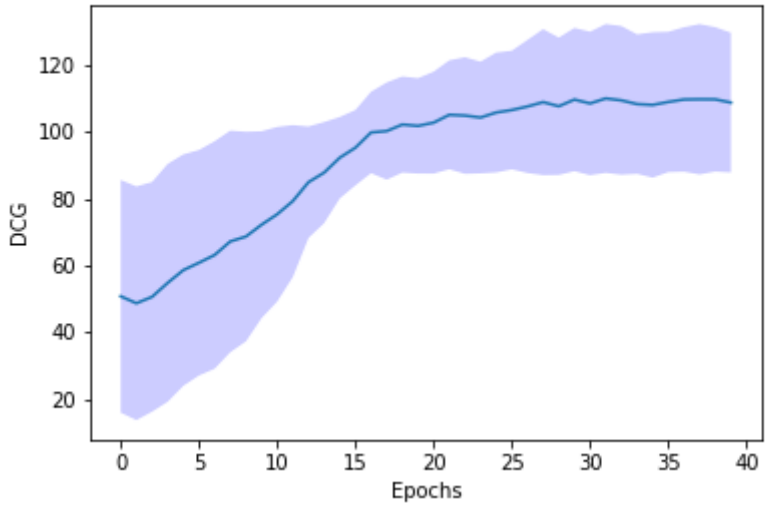
4.3.4. ábra: MLP tanítás a Gauss-zajjal



4.3.5. ábra: MLP DCG értékei

NCF

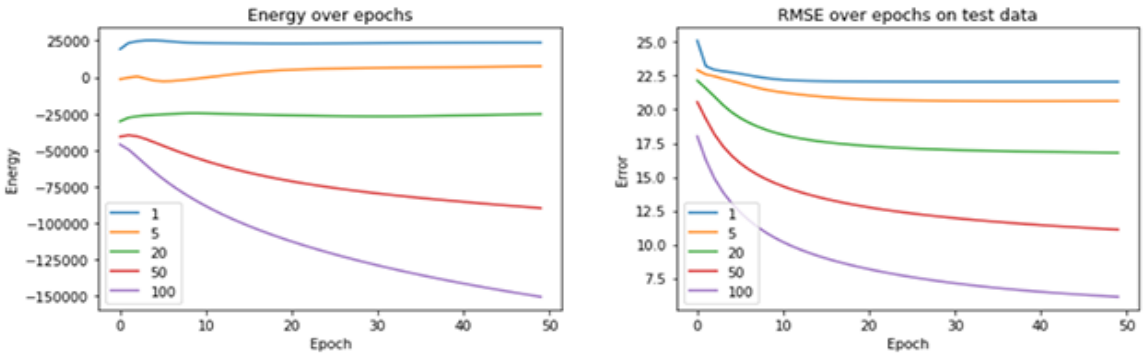
Az egyesített hálót is 256 batch mérettel tanítottuk, a GMF modellt 10, az MLP modellt 25 epochig tanítottuk, mivel az előzőekben láthattuk, hogy ezután már nem tudtak ezen felül tanulni (vagy elkezdtek túltanulni), az egész rendszert 40 epochig tanítottuk, SGD módszerrel [16]. Az NCF módszerrel elért DCG értékeket a 4.3.6. ábrán láthatjuk.



4.3.6. ábra: NCF DCG

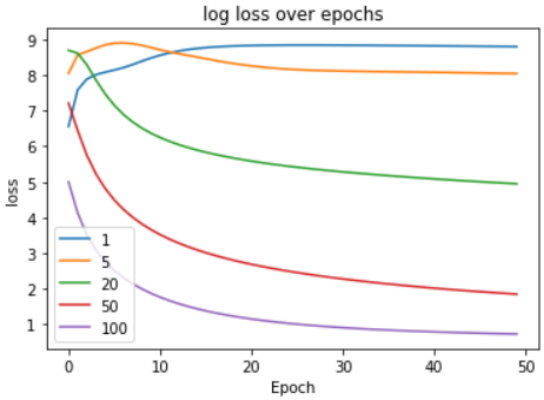
RBM

Létrehoztunk 5 RBM-et rendre 1, 5, 20, 50 és 100 méretű látens vektorokkal, majd 50 epochon keresztül tanítottuk őket (batch size-ot 1-nek választottuk).



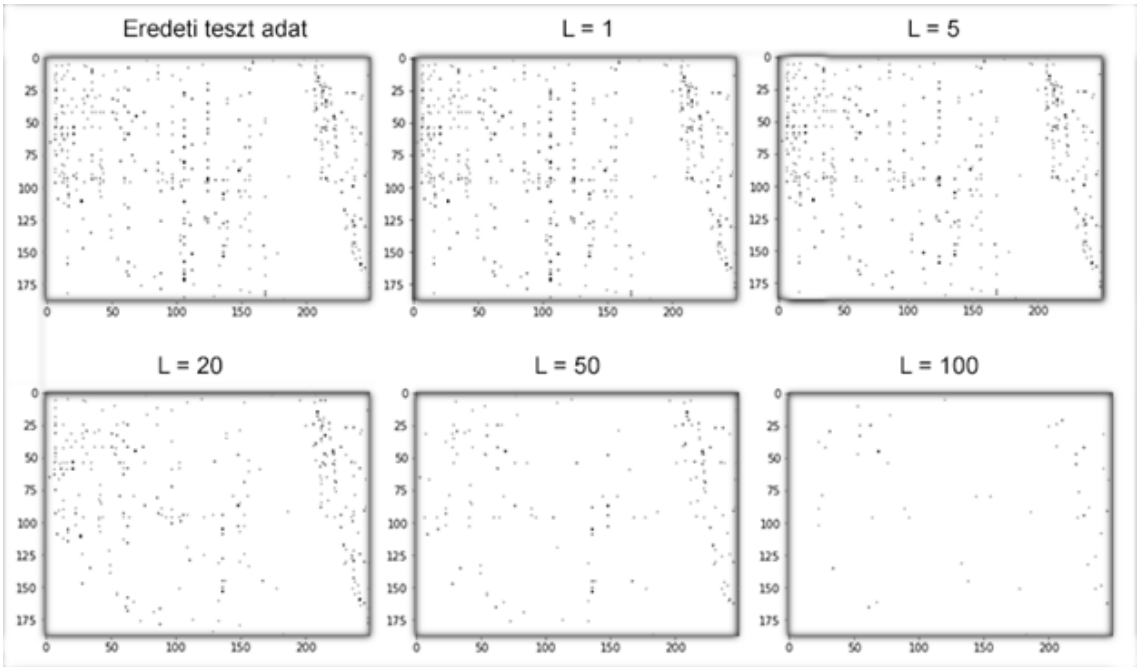
4.3.7. ábra: RBM tanítása – energia és RMSE

A 4.3.7. ábrán az első tanítási lépéstől kezdve láthatók a mérési eredmények. A baloldali ábrán a modellek energiáját számoltuk az adathalmazra minden epochban. Látszik, hogy az 50-nél kevesebb látens dimenziójú RBM-eknek nem igazán csökkent az energiájuk a nagyobbakéhoz képest (sőt növekedtek is). A jobboldali diagrammon a modell által predikált és a tényleges teszt adat közötti *RMSE* látható. Ezen látható, hogy minél több dimenziójú a látens tér, annál jobban rá tud tanulni az adathalmazban rejlő mintázatokra. Túltanulás nem tapasztalható egyik esetben sem ez alapján.



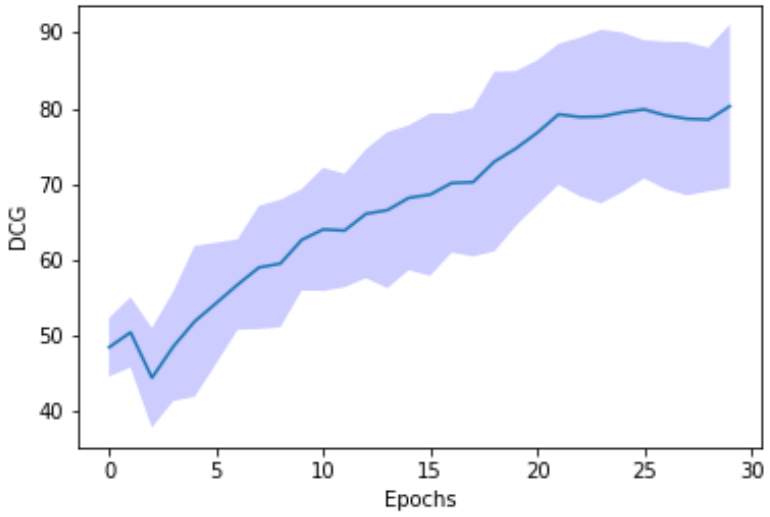
4.3.8. ábra: RBM tanítása – loss függvény

A 4.3.8. ábrán a *bináris keresztentrópia*, mint loss függvény látható az első tanítástól kezdve. Ez a diagramm is azt mutatja, hogy a nagyobb modellek sokkal jobban tanulnak (az 1 és 5 méretű RBM-eknek emelkedik is a keresztentrópiájuk).



4.3.9. ábra: RBM eredményei

A 4.3.9. ábrán az eredeti tesztmátrix, és az L méretű RBM-ek által predikált kerekített mátrixok különbségei láthatók. Itt a második képtől kezdve azt láthatjuk, hogy milyen arányban találja el a modell, hogy melyik elemekkel volt interakció (fekete pixelek azok az elemek, melyeket nem talált el). Minél nagyobb L, annál pontosabb a modell, ez olvasható le ezekről a képekről is. (A teljesen ideális az lenne, ha teljesen fehér képet kapnánk.) Végül az RBM módszerrel elért DCG értékeket a 4.3.10. ábra mutatja.



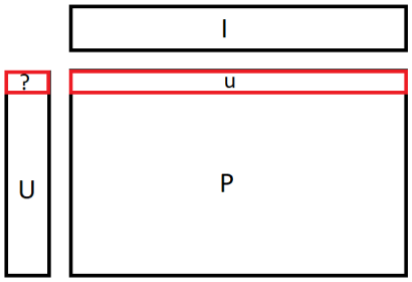
4.3.10. ábra: RBM módszer DCG értékei

4.4 Módszerek a cold start probléma szempontjából

A kollaboratív szűrés alapú módszerek közös hiányossága, hogy új felhasználó (vagy új termékre is hasonló módon ez megfogalmazható) bekerülésekor nem tudnak az új felhasználónak ajánlani termékeket, mivel nincsenek még tranzakcióik; ezt cold start problémának hívják. A módszerek nagy része nincs is felkészülve új felhasználó esetére még abban az esetben sem, hogyha a tranzakciós információk (legyen az vásárlás, megtekintés, meghallgatás stb.) egyből rendelkezésre állnának az új felhasználó beérkezésével együtt; ezt ugyanis csak úgy oldják meg, hogy az új felhasználóval bővítik a tanulóállományt (tranzakciós mátrix sorainak bővítésével) és ezen újra futtatják a tanuló algoritmust. Tehát olyan módszer tudja legalább részben kezelni ezt a problémát, ahol nincs szükség újabb tanulásra az új felhasználó esetén, azaz azt nem tanulási időben, hanem a predikció során kezeli és adja a tranzakciós információk alapján az ajánlásokat. A mi célunk is ez volt, és ha az eredeti problémát nem is oldottuk meg, de azt a jellegű cold start problémát igen, ahol az új felhasználót még a meglevő információi alapján sem tudták egyből (azaz tanulás nélkül) kezelni. A továbbiakban sorra vesszük a módszereket ebből a cold start probléma szempontjából.

iALS

Az iALS esetén ugyan nem a predikciós lépésben, de megoldható a cold start probléma (ld. 4.4.1. ábra). Legyen u, az új felhasználó vektora (egy sor lenne P-ben), ilyenkor tudunk hozzá egy lépésben számolni egy látens vektort. I, vagyis a termékek tanított látens mátrixa adott, keressük azt a látens vektort, amivel I-t megszorozva u-t kapjuk, ezt a lineáris regresszionál bemutatott módszerrel megkaphatjuk, majd az így szerzett látens vektorral megszorozzuk I-t és megkapjuk a becsült u vektort, ezt csökkenő sorba rendezve megkapjuk az ajánlási listánkat, ebben az előbb szereplő termékek relevánsabbak a felhasználó számára a modell szerint.



4.4.1. ábra: iALS cold-start esetén

NCF

Az NCF esetén még bonyolultabb a probléma megoldása. Modellünk egy-egy one-hot kódolású vektort vár inputként, ha ajánlani szeretnénk egy új felhasználónak, akkor hozzá kell rendelni egy a rendszer bemeneti méretének megfelelő vektort, ezt úgy tehetjük meg, hogy egy hasonlóság alapú módszerrel megkeressük a k darab, az interakciók alapján leghasonlóbb felhasználót, és a hozzájuk rendelt one hot kódokat összegezzük és elosztjuk k-val. Így, ha ezt adjuk a felhasználói bemenetre, akkor a hozzá tartozó látens vektor a hozzá hasonló felhasználók vektorának átlaga lesz. Minden termékre kiszámíthatjuk a modell által predikált valószínűségeket az adott felhasználóhoz. Így

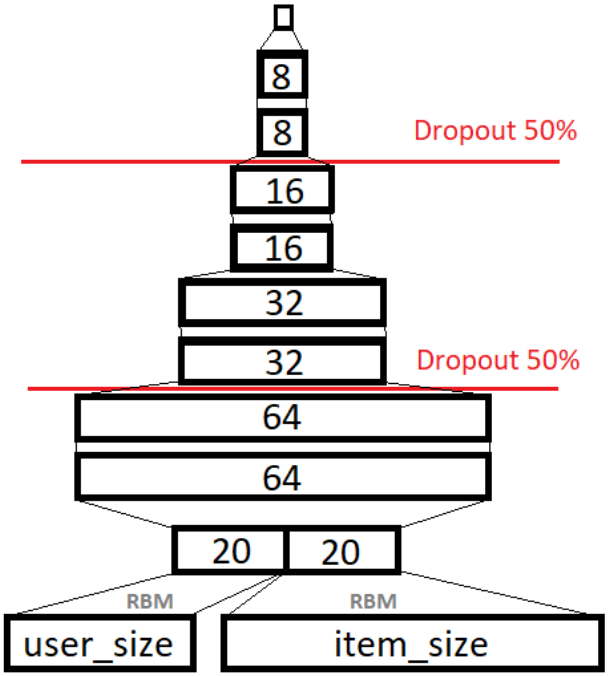
kapuk egy ajánlás listát, ez tartalmazza minden termékhez az interakció valószínűségét, ezt csökkenő sorrendbe rendezve kiválaszthatjuk belőle a legnagyobb top-n értékhez tartozó termék indexét, ezeket tudjuk a felhasználónak ajánlani.

RBM

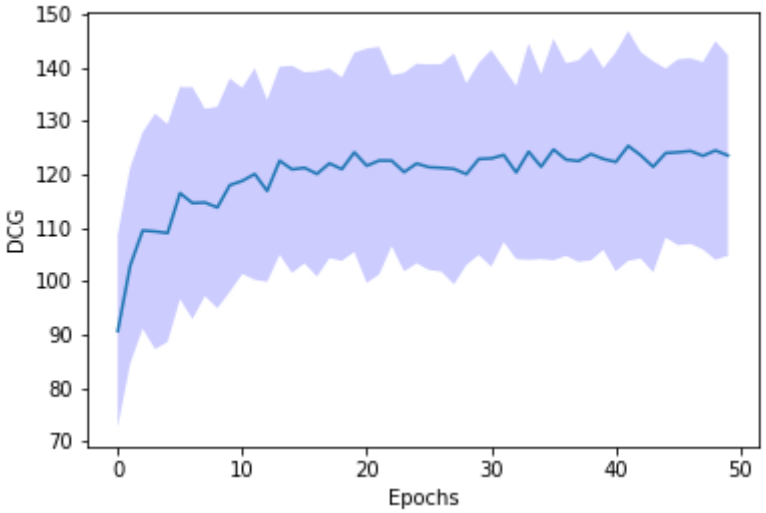
Az RBM bemenetén egy felhasználóhoz tartozó interakciós vektort vár. Így, ha egy új felhasználónak kell ajánlást készíteni, akkor megtehetjük, hogy minden átalakítás nélkül beadjuk a vektort a rendszer inputjára. Ez működni fog, hiszen mérete megegyezik, és a háló a megtanult súlyok alapján el tudja készíteni a rejtett rétegbeli reprezentációját. Az RBM úgy képezi le a felhasználók látens vektorát, hogy az egyes input neuronokhoz tartozó súlyokat összegzi, mert, ha volt interakció az adott felhasználó és termék között, akkor az adott input neuron értéke 1 különben 0. Így, ha két felhasználó vektora hasonló, akkor a rejtett rétegbeli reprezentációjuk is hasonló lesz. Vagyis az újonnan érkező felhasználó látens vektora hasonló lesz a rá hasonló felhasználók látens vektorára, ezért tud a háló általánosítani. Az ajánlásunkat egy Gibbs lépés után visszakapott predikált felhasználói vektor alapján tehetjük.

5 Double RBM Deep Neural Network (DDNN)

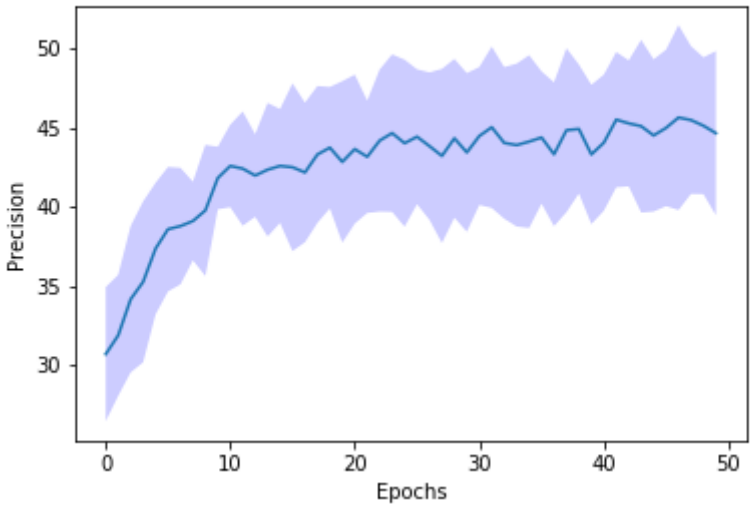
Szerettük volna kihasználni az NCF modell erősségét és az RBM modell egyszerű kezelhetőségét, ezért kidolgoztunk egy új módszert (ezt Double RBM Deep Neural Network-nek röviden DDNN-nek neveztük el) az előzőleg bemutatott módszereket, mint ötleteket felhasználva. Modellünk két RBM-ből és egy mély neurális hálóból épül fel (ld. 5.1. ábra). Először egy-egy rejtett réteggel (itt is 20-nak választottuk a rejtett réteg nagyságát) rendelkező RBM-et tanítunk a felhasználók és a termékek vektorával, majd ezen modellek rejtett rétegeit konkatenáltuk, az így kapott vektororra pedig egy többrétegű neurális hálót helyeztünk. Mivel a neurális háló bemenetét RBM-ek állítják elő a felhasználó és a termék vektorából, így ez a modell is tudja kezelni a cold-start problémát. Az RBM által létrehozott rejtett régek reprezentálják az adott felhasználót vagy terméket. Az RBM leképezése rejtett rétegbe felfogható egyfajta tömörítésnek is, lecsökkenti a vektorok méretét úgy, hogy csak az alapvető tulajdonságait tartja meg, amik segítségével vissza tudja állítani őket majdnem eredeti állapotukba. Ezek a rejtett vektorok tehát tekinthetők az NCF modell látens vektorainak, melyek az adatokat reprezentálják egy látens térben. Mivel ez már a fontosabb összefüggések alapján kiemelte a főbb feature-öket az eredeti vektorokból, így egy neurális háló képes lehet ez alapján megtanulni és predikálni az interakció valószínűségét egy felhasználó és egy termék között, ha adott azok RBM által generált rejtett vektora. A GMF modell viszont nem fog tudni tanulni ezen inputok mellett, hiszen a modell csupán összeszorozza a két vektort, ebből próbál következtetéseket levonni, az RBM viszont nem úgy állítja elő a látens vektorokat, hogy azok szorzatából ezt meg lehessen tenni. Ezért modellünkben csak az NCF modell MLP részét vettük át. Az MLP rész 8 rétegű, legalsó szinten két 64 méretű réteg található, majd ezt követi két 32, két 16 és a végén két 8 méretű réteg, közöttük ReLU aktivációs függvényt használtunk, és a túltanulás ellen bevezettünk minden második réteg közé 1-1 dropout réteget is. A kimeneti rétegen sigmoid aktiváció található és 50 epochig tanítottuk a hálót, 256-os batch mérettel.



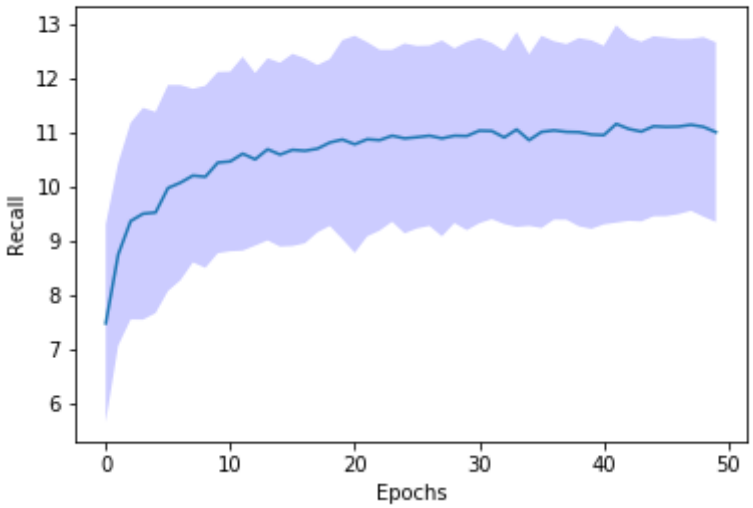
5.1. ábra DDNN architektúrája



5.2. ábra: DDNN DCG értékei



5.3. ábra: DDNN Precision értékei



5.4. ábra: DDNN Recall értékei

A 5.2, 5.3., 5.4. ábrákon mutatjuk a kapott DCG, Precision és Recall értékeket. Ahogy ezeken látszik: sikerült jobb eredményeket elérni a DCG indikátornál, mint a többi konkurens módszer, és a módszerünk előnye, hogy a cold-start problémát is kezeli.

6 Eredmények

A tanítás után összegezzük a legfontosabb értékeket. Erre a már korábbi grafikonokon is szereplő Recall, Precision és DCG értékeket vettük alapul, mivel ezek adnak tényleges képet a modellek teljesítményéről, viszont nem ezek alapján tanítottuk a modelleket. Az egységes méréseink alapján a lentebb látható összehasonlító táblázatokat (6.1 és 6.2. táblázat) állítottuk össze, amiből kiolvasható a három általunk mért kiértékelési metrika és az eredmények; az implementált modellekben félkövérrel kiemelve találhatóak a legjobb értékek az adott metrikában. Az értékek között van, ahol nagy szórás figyelhető meg, de összességében minden modellnek van legalább egy jó tulajdonsága, amely miatt érdemes lehet vele foglalkozni. Ezért egyikről sem jelenthető ki, hogy alsóbbrendű lenne, csupán a sokféle megközelítés eredménye a változatosság az értékekben.

6.1. táblázat. Ajánlórendszer módszerek összehasonlítása top-50 esetén

	Recall	**Precision**	**DCG**
iALS	**25,00**	**91,20**	57,73
RBM	15,82	29,63	81,40
GMF	10,32	41,95	117,21
MLP	12,19	44,04	112,25
NCF	12,21	43,05	110,09
DDNN	11,15	45,64	**125,36**

6.2. táblázat. Ajánlórendszer módszerek összehasonlítása top-20 esetén

	Recall	**Precision**	**DCG**
iALS	**45,68**	**76,49**	63,01
RBM	11,67	53,72	71,67
GMF	12,42	20,3	77,31
MLP	14,47	20,88	72,58
NCF	14,19	19,88	71,45
DDNN	13,27	24,31	**84,80**

Külön érdekesség, hogy ugyan a legtöbb területen az iALS éri el a legjobb eredményt, addig a DCG-ben az általunk alkotott DDNN algoritmus érte el a legjobb eredményt top-20 és top-50 ajánlásnál is. Ezt azért is tartjuk fontosnak kiemelni, mivel a DCG az egyik

legjelentőségteljesebb mérőszám; ugyanis a felsoroltak közül, ez az egy, amely figyelembe veszi az ajánlások sorrendjét is, amely a való életben is fontos. Hiszen jelen esetben 50 (illetve 20) ajánlást teszünk egy felhasználónak, amelyre a valóságban kevés esély van, hogy mindegyiket végig böngészné, ezért lesz fontosabb az, hogy az 50 ajánlásunkat megfelelő sorrendben tegyük meg (recall és a precision esetében mindegy, hogy például a 8 valóban releváns termék a top 50-en belül hol helyezkedik el; a DCG azonban különbséget tesz és jóval magasabbra értékeli azt a változatot, ahol az első 8 találat a releváns azzal szemben, ahol a 43-50-ig levő találatok relevánsak).

7 Összefoglalás

Zárásként reflektálunk az ajánlórendszerek helyére világunkban, Amint azt láthattuk az ajánlórendszerek komoly fejlődésen mentek keresztül az elmúlt években. A kezdetleges baseline módszertől, a hasonlóságon alapulókon keresztül eljutottak a látens tereket alkalmazó modellekig, amelyeknek jelentős része már neurális hálót használ. A számítási kapacitások növekedésével ezen módszerek nagyfokú felhasználására számíthatunk az ipar több területén. Azonban ezen módszereknek is megvannak a maguk hátrányai, lásd cold-start probléma.

A dolgozatunkban mi megkíséreltünk egy választ adni a technológia javítására. Több modell komplex elemzése után az eredmény a DDNN algoritmus lett, amely azontúl, hogy egy versenyképes modell, a már említett cold-startot is képes kezelni. Az algoritmus kombinálja a Boltzmann Machine és az NCF modelleket, mindkettő előnyös tulajdonságait megtartva, kiemelve azokat: az NCF (amely már maga is egy összetett modell) pontosságával, és az RBM képességével a cold-start kezelésére egyaránt rendelkezik. Ezzel rámutat arra, hogy az ajánlórendszerek területén gyakran a hibrid modellek hozzák a legjobb eredményeket (mint azt a Netflix prize is mutatta) és kombinációjuk további kutatást igényel.

Ezen rendszerekre egyre nagyobb szükség lesz a jövőben, ugyanis egyre több fajta adat gyűlik össze, amelyek egyedi megközelítéseket igényelnek, és a több szempont figyelembevétele csak összetett modellekkel lesz lehetséges. Tehát meggyőződésünk, hogy a modellek kombinálása adott szituációkban való helytálláshoz célravezető és egyre fontosabb irányvonal a jövő ajánlórendszereinek területén.

Köszönetnyilvánítás

A TDK dolgozatban ismertetett eredmények a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar Balatonfüredi Hallgatói Kutatócsoport szakmai közössége keretében jöttek létre a régió gazdasági fejlődésének elősegítése érdekében. Az eredmények létrehozása során figyelembe vettük a balatonfüredi központú Rendszertudományi Innovációs Klaszter által megfogalmazott célkitűzéseket, valamint a párhuzamosan megvalósuló EFOP 4.2.1-16-2017-00021 pályázat támogatásával elnyert „BME Balatonfüredi Tudáscentrum" térségfejlesztési terveit.

A kutatás az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg (EFOP-3.6.2-16-2017-00013, Innovatív Informatikai és Infokommunikációs Megoldásokat Megalapozó Tematikus Kutatási Együttműködések).

Irodalomjegyzék

[1] Zibriczky Dávid (2015) Perszonalizált tartalomajánló szolgáltatás IPTV és OTT rendszerek számára, Hiradástechnika, LXX évf., 49-55.

[2] Ziegler, C. N., McNee, S. M., Konstan, J. A., & Lausen, G. (2005). Improving recommendation lists through topic diversification. In Proceedings of the 14th international conference on World Wide Web (pp. 22-32). ACM.

[3] Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. Statistics surveys, 4, 40-79.

[4] He, Xiangnan, et al. "Fast matrix factorization for online recommendation with implicit feedback." Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. ACM, 2016.

[5] Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann machines for collaborative filtering." Proceedings of the 24th international conference on Machine learning. ACM, 2007.

[6] Hinton, Geoffrey E. "A practical guide to training restricted Boltzmann machines." Neural networks: Tricks of the trade. Springer, Berlin, Heidelberg, 2012. 599-619.

[7] Hinton, Geoffrey E. "Training products of experts by minimizing contrastive divergence." Neural computation 14.8 (2002): 1771-1800.

[8] A Beginner's Guide to Restricted Boltzmann Machines (RBMs) - https://skymind.ai/wiki/restricted-boltzmann-machine

[9] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In 2008 8th IEEE International Conference on Data Mining (pp. 263-272).

[10] Xue, H. J., Dai, X., Zhang, J., Huang, S., & Chen, J. (2017). Deep Matrix Factorization Models for Recommender Systems. In IJCAI (pp. 3203-3209).

[11] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). Neural collaborative filtering. In Proceedings of the 26th international conference on world wide web (pp. 173-182). International World Wide Web Conferences Steering Committee.

[12] Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR), 52(1), 5.

[13] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In ICLR, pages 1-15, 2014.

[14] Paper Review: Neural Collaborative Filtering Explanation & Implementation. https://towardsdatascience.com/paper-review-neural-collaborative-filtering-explanation-implementation-ea3e031b7f96

[15] Li, Y., & Yuan, Y. (2017). Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems* (pp. 597-607).

[16] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade* (pp. 421-436). Springer, Berlin, Heidelberg.

[17] Kaggle: RetailRocket adathalmaz, https://www.kaggle.com/retailrocket/ecommerce-dataset

[18] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining* (pp. 263-272). Ieee.

[19] Hidasi, B., & Tikk, D. (2012). Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 67-82). Springer, Berlin, Heidelberg.