Eötvös Loránd Tudományegyetem

Természettudományi Kar

Virág Fausztin Asztrik

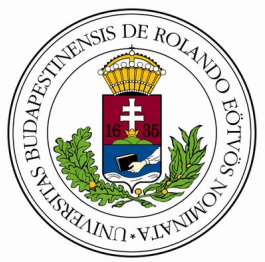
Neuronhálók alkalmazása implicit ajánlórendszerekben

MSc Alkalmazott Matematikus Szakdolgozat

Témavezető:

Dr. Benczúr András

Operációkutatás Tanszék



Budapest, 2019

Köszönetnyilvánítás

Szeretném megköszönni a családomnak a támogatást. Már önmagában a témárt hálával tartozom Benczúr András témavezetőmnek, valamint, hogy hozzáférést biztosított a SZTAKI számítógépes erőforrásaihoz. Továbbá nagyon köszönöm Kelen Domokosnak a rengeteg technikai és szakmai segítséget, amit munkám során tőle kaptam.

Tartalomjegyzék

1. fejezet

Bevezetés

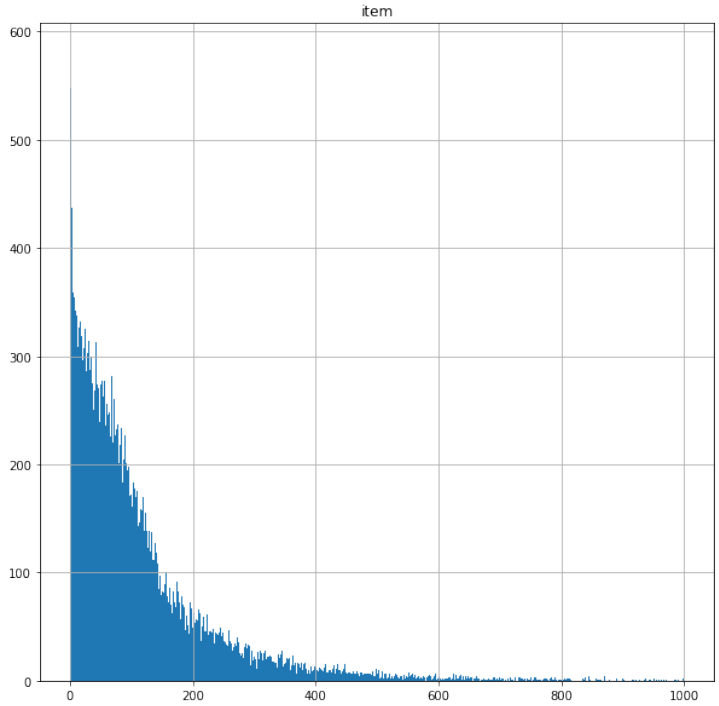
Az internetes vásárlások és tartalom megosztás világában már nem szorolunk rá arra, hogy a földrajzi helyzetünk, a közelünkben lévő helyi médiumok, vagy épp boltok kínálatára korlátozzuk a figyelmünket. Azonban a bőség zavara egy teljesen új problémakört hozott létre: amikor már az egyénnek nincs lehetősége végigböngésznie mindent, ami aktuálisan eleget tesz a céljainak és követelményeinek például egy vásárlás során, akkor szükség van valami külső behatásra, ami segíteni tudja. Milyen jó lenne, ha keresni sem kellene, hanem csak az ölünkbe pottyana. A másik oldalról, az eladó felől nézve milyen kényelmes lenne, ha a vevő nem válogatna, hanem megvenne mindent, amit elsőre mutatunk neki. Ezt az idealist világot próbálják kiszolgálni az ajánlórendszerek. Persze vegyes érzések juthatnak eszünkbe erről a műfajról, de ha kicsit a probléma mögé tekintünk, akkor egy nagyon érdekes kihíváshoz lehet szerencsénk.

Valószínűleg mindnyájan elmondhatjuk magunkról, hogy a napi internet használatunk során idősorok egész seregét hagyjuk magunk után. És joggal vetődik fel az ötlet, hogy ha alapul vesszük azt, hogy az elmúlt években milyen eredményeket tudtunk elérni videók, élőbeszéd, szöveg feldolgozás esetében, amik mind mind szekvenciák, akkor miért ne próbálhatnánk meg az ott bevált technológiát használni a böngészési előzményeinkre is. Így hamar eljutunk a dolgozatunk témájához, ami ebből a két irányból indul el: bemutatjuk az ajánlórendszerek problémáját, néhány elemi megoldással együtt. Ezután végigjárjuk a neuronhálók egyik legelterjettebb alkalmazását szekvenciális adatokra, megvizsgáljuk az ezzel járó nehézségeket, majd a legvégén megnézzünk egy módszert arra, hogy az utóbbi hogyan kínálhat megoldást az előbbire.

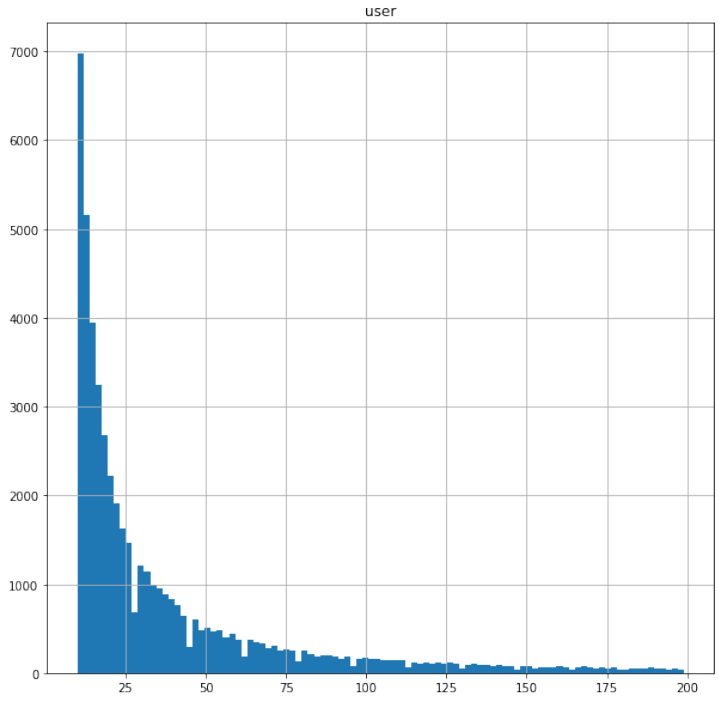
2. fejezet

Adatok

A dolgozatban ismertetett módszereket a *lastfm* adathalmazon [1], [2] fogjuk kipróbálni. Az adathalmazból csak az implicit információra fogunk koncentrálni, vagyis rendelkezésünkre állnak azon adatok, hogy adott felhasználó mikor hallgatott meg egy zenét. Ismétlődés nincs az adatok között, vagyis egy felhasználónak az *előzményei (history)* egy ismétlődés nélküli lista az általa meghallgatott zenék azonosítójáról. (Az időbélyegekből mi most csak a sorrendiséget tartjuk meg, két hallgatás között eltelt időt nem vesszük figyelembe.) A legrövidebb előzmény 1 hosszú, míg a leghosszabb 5211. Ha megszámoljuk, hogy az egyes zeneszámokat hányan hallgatták meg, akkor azt láthatjuk, hogy a legkevesebbet hallgatott számot is 10 ember meghallgatta, míg a legtöbbett hallgatott számnak 9139 hallgatója akadt. A 2.1,2.2 ábrákon az ezekből készült hisztogrammokat ábrázoltuk. A láthatóság kedvéért mindkettőnek a két végét elhagytuk.



2.1. ábra. A felhasználók előzményeinek a hosszáról készült hisztogram.



2.2. ábra. A zeneszámok hallgatottságából készült hisztogram.

3. fejezet

Ajánlórendszerek

Az ajánlórendszerek feladata alapvetően az, hogy a *felhasználóknak (user)* olyan *termékeket (item)* javasoljon, ami rendszerint érdekelheti őket. A termék szinte bármi lehet, könyv, árucikk egy webshopban, vagy videó, zene, esetleg film, vagy épp sorozat. Az ajánláshoz sokféle adatot is felhasználhatnak, mint például tartalmi - hasonló témájú könyv valószínűleg érdekelheti még a felhasználót - a felhasználók és a termékek közötti interakciókból származó információt - hány csillagra értékelte a felhasználó, miket értékelt még hasonló számokkal stb. Az ajánlórendszerek minősége mind a felhasználóknak, mind a szolgáltatóknak kedvez. Nyilván, ha az ajánlás jó és vásárol még a vásárló, az megéri a boltnak, de ugyanakkor az interneten a tartalom mennyisége irtózatosan nagy tud lenni. Ha a felhasználó kézhez kapja a számára legrelevánsabb termékeket, sok időt megspórolhat neki, ha csak azakot kell végigböngésznie. Vagy épp amikor befejezünk egy jó sorozatot, vagy könyvet, akkor ahhoz hasonlót már szinte azonnal kapunk, mint ajánlás a mai internetes közegben. Ha definiálnunk kéne valahogy a problémát, akkor legjobb lesz, ha az Ajánlórendszerek kézikönyve [29] által kínált meghatározást követjük.

3.0.1. Definíció. *Az ajánlási probléma az, amikor meg szeretnénk becsülni egy felhasználó reagálását egy új termékre a rendszerben tárolt korábbi adatok alapján. Az ajánló redszerek olyan régi és/vagy új termékeket javasolnak a felhasználónk, ahol a becsült reakció kellően kedvező.*

3.0.2. Megjegyzés. A reagálás alapján három féle csoportra oszthatók az ajánlási problémák: skaláris, bináris, unáris.

- *Skaláris*: Vagy másképp *értékelős (ratings)* reagálások. A legyakoribb példa, $1 - 5$ csillaggal való értékelés. Ekkor ismert néhány felhasználó-termék párra egy szám,

hogy mennyire tetszik az adott felhasználónak az adott termék. Ilyen a Netflixen a filmek értékelése.

- *Bináris*: A felhasználó csak kétféle képpen értékelheti az adott terméket, ez rendszerint a "tetszik/nem tetszik". Ismerős lehet YouTube videók felfele/lefele mutató szavazós rendszeréből.

- *Unáris*: A felhasználónak nincs módja külön értékelésre, mindössze annyi információ áll rendelkezésünkre, hogy az adott felhasználó interakcióba lépett-e az adott termékkel vagy sem. Ez általában azt jelenti, hogy megvásárolta-e a terméket, vagy megtekintette-e. Itt példaként megemlíthetjük a Spotify zenehallgató szolgáltatást.

Szokás úgyis nevezni a skaláris és bináris reagálásokat, mint *explicit visszajelzések*. Míg csak a megtekintések által nyújtott információra *implicit visszajelzés* néven hivatkozhatunk. Nem feltétlenül kizárt néhány kisérő adat ismerete, mint például egy-egy felhasználó milyen sorrendben tekintette meg a termékeket. A fenti kategóriák persze főleg steril körülmények között különülnek el. A fenti szolgáltatók is igyekszenek minnél több további adatra is hagyatkozni a saját ajánlórendszerük esetén.

Legyen a felhasználók halmaza $\mathcal{U}$, a termékeké pedig $\mathcal{I}$. Továbbá legyen $\mathcal{R}$ a rögzített értékelések, $\mathcal{S}$ pedig egy értékelés lehetséges értékei (pl.: $\mathcal{S} = \{1, 2, 3, 4, 5\}$ vagy $\{\text{tetszett}, \text{nem tetszett}\}$). Adott $u \in \mathcal{U}$ és $i \in \mathcal{I}$ esetén jelölje r_{ui} az u felhasználó értékelését az i terméken. Az u felhasználó által értékelt termékek halmaza $\mathcal{I}_u$, fordítva pedig, azon felhasználók halmaza, akik értékelték az i terméket legyen $\mathcal{U}_i$. Az egyszerűbb jelölésért $\mathcal{U}_{ij} := \mathcal{U}_i \cap \mathcal{U}_j$ és $\mathcal{I}_{uv} := \mathcal{I}_u \cap \mathcal{I}_v$.

Ekkor két fontos problémáról beszélhetünk, a *legjobb termék (best item)* és a *top-N* ajánlásról. Az előbbiről akkor szoktunk beszélni, ha valamilyen bináris vagy skaláris ajánlórenszerről van szó. Legyen $f \colon \mathcal{U} \times \mathcal{I} \to \mathcal{S}$ függvény a becslésünk arra, hogy az egyes felhasználóknak melyik termék mennyire tetszik, azaz $f(u, i)$ jelentse u felhasználó (remélt) értékelését az i terméken. Adott u_a esetén a legjobb termék:

$$i^* := \max_{i \in \mathcal{I} \setminus \mathcal{I}_{u_a}} f(u_a, i). \tag{3.1}$$

Az ajánlás minőségét egy előre definiált teszt halmazon szokás ellenőrizni. Osszuk fel az $\mathcal{R}$-et $\mathcal{R}_{\text{tan}}$ tanító- és $\mathcal{R}_{teszt}$ teszthalmazra. Bizonyos módszerek tesztelésénél ezenfelül érdemes a következő módon módosítani az $\mathcal{I}$ halmazt:

$$\mathcal{I} \leftarrow \mathcal{I} \setminus \{i | \nexists u \exists u' : r_{ui} \in \mathcal{R}_{\text{tan}}, r_{u'i} \in \mathcal{R}_{\text{teszt}}\}.$$

Nemes egyszerűséggel azért van erre szükség, mert a csak a teszt halmazban felbukkanó termékekről nincs semmilyen előismeretünk, amit így azt sok esetben nem tudjuk kezelni. A két talán legyakoribb hibafüggvény erre a problémára az *átlagos abszolút hiba (Mean Absolute Eroor- MAE)*:

$$\mathrm{MAE}(f) := \frac{1}{|\mathcal{R}_{\mathrm{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\mathrm{test}}} |f(u,i) - r_{ui}| \tag{3.2}$$

és az *átlagos négyzetgyökös hiba (Root Mean Squared Error - RMSE)*:

$$\mathrm{RMSE}(f) := \sqrt{\frac{1}{|\mathcal{R}_{\mathrm{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\mathrm{test}}} (f(u,i) - r_{ui})^2}. \tag{3.3}$$

Implicit visszajelzések esetén nem nagyon beszélhetünk olyanról, hogy a felhasználó melyik terméket hogyan értekeli/értékelné. Ezért ebben az esetben inkább a top-N ajánlást használjuk. Ezért itt az aktuális u_a felhasználónak egy $L(u_a)$ N elemű listát ajánlunk, ami az $\mathcal{I}$ egy N elemű rendezett részhalmaza. Legyen $T(u_a)$ azon i termékek halmaza, amelyre $r_{u_a i} \in \mathcal{R}_{\mathrm{teszt}}$. Ekkor beszélhetünk az ajánlás *precizitásáról (precision)* és *felidézéséről (recall)*:

$$\mathrm{Precision}(L) := \frac{1}{\mathcal{U}} \sum_{u \in \mathcal{U}} \frac{|L(u) \cap T(u)|}{|L(u)|} \tag{3.4}$$

$$\mathrm{Recall}(L) := \frac{1}{\mathcal{U}} \sum_{u \in \mathcal{U}} \frac{|L(u) \cap T(u)|}{|T(u)|} \tag{3.5}$$

vagyis míg előbbi azt mondja meg, hogy átlagosan hány javaslatunk volt jó, addig az utóbbi azt mondja meg, hogy a valódi megtekintésekből átlagosan mennyit sikerült eltalálnunk. Ha szeretnénk azt is mérni, hogy mennyire jó az $L(u)$ sorrendje, akkor érdemes használni az *átlagos reciprok rang (Mean Reciprocal Rank - MRR)* függvényt:

$$\mathrm{MRR}(L) := \frac{1}{\sum_{u \in \mathcal{U}} |T(u)|} \sum_{u \in \mathcal{U}} \sum_{i \in T(u)} \frac{1}{\mathrm{rang}(i, L(u))}, \tag{3.6}$$

ahol $\mathrm{rang}(i, L(u))$ jelöli azt, hogy az i hagyadik az $L(u)$ listán, ha nincs rajta, akkor legyen ∞ és így a reciprokát definiáljuk 0-nak. A legszemléletesebb interpretációja ezeknek a tesztelési metódusoknak az, ha úgy képzeljük el a rendelkezésre álló adatainkat, hogy van egy $|\mathcal{U}| \times |\mathcal{I}|$ mátrixunk, és az u sor i-edik eleme tartalmazza azt, hogy hogyan értékelte az u felhasználó az i terméket. A teszteléshez, pedig "letakarunk" néhány értéket a mátrixból, majd a megmaradó elemeken tanítjuk a modellünket, a tesztelést pedig a letakart elemeken végezzük.

A továbbiakban az implicit visszajelzéső esetre fogunk koncentrálni, lévén a mi adatunk is csak ennyi információt hordoz csak. Mindenképp érdemes megjegyezni, hogy annak ellenére, hogy nyilvánvalóan sokkal kevesebb az információ tartalma egy ilyen adatnak, cserébe sokkal, de sokkal több mennyiség gyűjthető belőlük. Gondoljunk csak bele, hogy vajon egy ember az interneten általa megtekintett videók közül hányról ad vissza valamilyen értékelést. Ráadásul abban sem lehetünk biztosak, hogy a felhasználókat mindig tudjuk azonosítani. Lehet hogy a weboldal használatához nem szükséges regisztráció. Ilyen esetekben beszélünk *session alapú* ajánlásokról. Egy session a mi esetünkben jelentse azt, amíg képesek vagyunk egy-egy felhasználót nyomon követni. Az ilyen tipusú adatok rendszerint rengeteg felhasználót tartalmaznak, ugyanis a különböző session-ök különböző felhasználóként tárolhatók az eddigi terminológiánkban. Azonban ebben az esetben nem igen feltételezhetünk semmilyen ismert háttérinformációt a felhasználókról.

3.1. Collaborative filtering

A *szociális filterezésnek (social/collaborative filtering-CF)* nevezzük azokat a módszereket, amik $\mathcal{R}$-ből származó információk alapján végzik az ajánlást. Az ötlet mögött azok a feltételezések állnak, hogy ha a megfigyeléseink alapján két felhasználó hasonlóan értékeli ugyanazokat a termékeket, akkor más termékeket is hasonlóan értékelhetnek, vagy épp ha egy terméket sokan ugyanúgy értékelnek, akkor talán más felhasználók is hasonlóan vélekednek róla. Ugyanígy az implicit visszajelzések terén. Az ajánlórendszerek ezen fajtáját két további csoportra bonthatjuk, a *hasonlósági* vagy *szomszédsági (neighborhood)* és *modell alapú (model-based)* eljárásokra.

Az utóbbiak azok a megoldások, amik valamilyen tanuló algoritmust használnak az $\mathcal{R}_{\text{tan}}$ halmazon és utána azt alkalmazzák az $\mathcal{R}_{\text{teszt}}$ halmazra. A későbbi fejezetekben lesz még erre példa.

A szomszédsági eljárásnak nevezzük azokat a megoldási módszereket, amikor az ajánlást valemilyen közvetlen számolás révén kapjuk az $\mathcal{R}_{\text{tan}}$ halmazból. A legegyszerűbb ilyen módszer a *népszerűségi modell (popularity model)* ami minden u felhasználónak az N leggyakoribb terméket ajánlja vagyis

$$\text{pop}(i) := |\{u : r_{ui} \in \mathcal{R}_{\text{tan}}\}|$$

alapján rendezzük a termékeket nemnövekvő sorrendbe és legyen $L_p(u)$ az előbbi rendezés első N eleme, minden $u \in \mathcal{U}$-ra. Vegyük észre, hogy itt a termékek rendezése min-

den felhasználónál ugyanaz. A *hasonlósági ajánlórendszerek* [30] célja az, hogy minden felhasználónak, az általa már megtekintett termékekhez hasonlót javasoljon. Legyen H egy hasonlósági mátrix, vagyis jelölje H_{ij} az i és j termék hasonlóságát, valamint legyen $w(u, j, \mathcal{I}_u, \mathcal{R}_{\tan}, H)$ egy súlyfüggvény, amivel egy felhasználó előzményeihez, azaz korábban megtekintett termékeihez súlyt rendelhetünk a termékek hasonlósága segítségével.

Algorithm 1 Hasonlósági ajánlórendszer

procedure HASONLOSAGI_AR($\mathcal{U}, \mathcal{I}, \mathcal{R}_{\tan}, H, w$)
  $I \leftarrow$ egy lista $\mathcal{I}$ elemeivel
  $\mathcal{L} \leftarrow \emptyset$
  for $u \in \mathcal{U}$ **do**
    for $j \in \mathcal{I}$ **do**
      $y_{uj} \leftarrow w(u, j, \mathcal{I}_u, \mathcal{R}_{\tan}, H)$
    $L_u \leftarrow I$
    Rendezzük L_u elemeit nemnövekvő sorrendbe az y_{uj} értékek alapján
    $L_u \leftarrow L_u[1 : N]$
    $\mathcal{L} \leftarrow \mathcal{L} \cup L_u$
  return $\mathcal{L}$

3.1.1. Megjegyzés. A népszerűségi modell ennek egy speciális esete, amikor is $w(u, j, \mathcal{I}_u, \mathcal{R}_{\tan}, H) = \mathrm{pop}(j)$.

Néhány további példa a hasonlóság definiálásához.

3.1.2. Definíció. *Jaccard hasonlóság:*

$$H_{ij}^{(Jaccard)} := \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$

3.1.3. Definíció. *Koszinusz hasonlóság:*

$$H_{ij}^{(cos)} := \frac{\sum\limits_{u \in U_{ij}} r_{ui} r_{uj}}{\sqrt{\sum\limits_{u \in U_i} r_{ui}^2} \sqrt{\sum\limits_{u \in U_j} r_{uj}^2}}$$

3.1.4. Definíció. *Pearson hasonlóság:*

$$H_{ij}^{(Pearson)} := \frac{\sum\limits_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum\limits_{u \in U_i} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum\limits_{u \in U_j} (r_{uj} - \bar{r}_j)^2}},$$

ahol $\bar{r}_i = \frac{|\mathcal{U}_i|}{|\mathcal{U}|}$.

3.1.5. Megjegyzés. Nem más mint, a termékek megtekintéséből számolt Pearson korreláció.

Vegyük észre, hogy korábban említett felhasználó-termék mátrix interpretációt tekintve, ezeke nem mások, mint a termékeknek megfelelő oszlopvektorok között vett távolság metrikák. A Jaccard és a koszinusz hasonlóságnál legyen

$$w(u, j, \mathcal{I}_u, \mathcal{R}_{\tan}, H) := \frac{\sum\limits_{i \in \mathcal{I}_u} H_{ij} r_{ui}}{\sum\limits_{i \in \mathcal{I}_u} H_{ij}},$$

a Pearson hasonlóságnál pedig használjuk a

$$w(u, j, \mathcal{I}_u, \mathcal{R}_{\tan}, H) := \bar{r}_j + \frac{\sum\limits_{i \in \mathcal{I}_u} H_{ij}(r_{ui} - \bar{r}_i)}{\sum\limits_{i \in \mathcal{I}_u} H_{ij}}$$

súly függvényt. Az így kapott hasonlósági eljárásokat fogjuk összevetni a később modell alapú ajánlórendszerünkkel.

4. fejezet

Beágyazás

A rendelkezésre álló adatunk, mint láttuk nem más, mint pusztán egy felsorolása a felhasználók és termékek közötti interakcióknak. Ahhoz, hogy mesterséges neuronhálókkal fel tudjuk dolgozni ezeket, kell találnunk egy alkalmas vektorreprezentációt. Hogy pontosan miknek keressük a vektorreprezentációját? Mi ebben a dolgozatban csak a termékekre fogunk koncentrálni, de egy természetes és akkár nagyon hasznos kiegészítés is lehet az, ha tudunk találni a felhasználóknak is egy megfelelő interpretálását. Azonban a fejezetben tárgyalt módszer, mint látni fogjuk legfeljebb szitnaktikailag alkalmazható rájuk, de valódi segítséget nem tud nyújtani. Tehát a *beágyazás (embedding)* allatt azt értjük, hogy egy halmaz elemeihez (jelen esetben a zeneszámok id értékeihez) találjunk egy "megfelelő" leképezést egy adott vektortérbe.

4.1. Osztályozási problémák

Az *osztályozási probléma* [3] egy rendkívül általános és gyakori probléma kör a gépi tanulásban.

4.1.1. Definíció. *Legyen a lehetséges bemenetek halmaza X, a lehetséges címkék, vagy osztályok véges halmaza pedig Y. Tekintsünk egy $\phi\colon X \to Y$ függvényt, amit rendszerint csak egy $H \subset X$ halmazon ismerünk. Osztályozási problémának azt nevezzük, amikor egy olyan $g\colon X \to Y$ függvényt keresünk, ami egy adott $T \subset X - H$ teszthalmazon a minnél több helyen egyezik ϕ-vel, másképp a $G := \{t | t \in T, g(t) = \phi(t)\}$ a lehető legnagyobb.*

Ha $|Y| = 2$, akkor bináris-, $|Y| > 2$ esetén pedig több osztályos osztályzási problémáról beszélünk.

4.1.2. Megjegyzés. Feltehető, hogy $Y = 1, 2, \ldots, m$.

A 4.1.1 problémára napjainkban az egyik leghíresebb példa a képek osztályozása. Például MRI felvételek alapján diagnosztizálni mélyhálós módszerekkel rákot [4]. Osztályozási probléma annak az eldöntése is, hogy milyen objektum van egy képen, de még sorolhatnánk a további példányait a problémakörnek ezrével.

Neuronhálók esetén ez a probléma a következő képpen néz ki. $X \subset \mathbb{R}^n$, a $H' := \{(x, y)|x \in H, y \in \mathbb{R}^{|Y|}\}$ tanító halmazon tanítjuk az F_Θ hálót (Θ optimalizálandó paraméterekkel), ahol az y vektorok úgynevezett *one-hot vektorok*, vagyis ha az x az $i - edik$ osztályba tartozik ($i \in Y$), akkor $y = e_i$ az i-edik egységvektor. Tanítás esetén egy nagyon fontos kérdés, hogy milyen hibafüggvény szerint is optimalizálunk. Erre precíz választ a következő alfejezt nyújt nekünk, de előtte még vizsgáljuk meg a motivációnkat. Minden bemenetre a lehető legmagabiztosabban szeretnénk tudni predikálni egy osztályt. Ehhez szükségünk lesz egy jól megválasztott aktivációs függvényre az utolsó rétegen. Osztályozási problémák esetén a leggyakoribb választás a softmax, aminek az okait a következőkben fogjuk elemezni.

4.1.3. Definíció. $\mathrm{softmax}\colon \mathbb{R}^m \to [0,1]^m$ *függvény i-edik eleme:*

$$\mathrm{softmax}(z)_i := \frac{e^{z_i}}{\sum_{j=1}^{m}(e^{z_j})}$$

Az első megfigyelés nem más, mint hogy a softmax eredménye egy diszkrét valószínűségi eloszlás, ugyanis könnyen látszik, hogy $\mathrm{softmax}(z)_i \in [0,1]$ minden i-re, valamint összegük 1. A népszerűség útján már 1902-ben elindult, akkor még egy általánosabb, úgy nevezett *Gibbs eloszlás* [8] formájában bukkant fel. Eredetileg a statisztikus mechanikában és termodinamikában írja le zárt diszkrét rendszerek állapoteloszlását [8],[9]. Az elnevezés eredete a történelem homályába vészni látszik, de feltehetően a "max" a maximumhely megtartása, a "soft" pedig a deriválhatóságot hivatott jelölni [7].

Ha tehát $z = F_\Theta(x)$, akkor $\mathrm{softmax}(z)_i = P(g(x) = i|\Theta)$, ahol $g(x)$ a hálók által tanult ϕ-t közelítő függvény. Tanítás során természetesen tudjuk, hogy $\phi(x) = y = e_k$ adott k-ra. Tekintsük ekkor a k-adik osztály *likelihood függvényét.*

$$L_k(\Theta) = P(g(x) = i|\Theta)$$

Becsülhetjük a problémát a *maximum likelihood módszerrel*, ahol maximumhely keresés miatt rendszerint a logaritmusát szokás venni, amit ezért *loglikelihood függvénynek* is neveznek. A mai optimalizáló algoritmusok rendszerint minimalizálnak, ezért általában a

negáltját veszik, amire így végső soron használhatunk egy újabb elnevezést is, mégpedig a *negatív-loglikelihood módszert*. A hiba visszaterjesztő algoritmus [?],[6] elindításához szükségünk lesz az utolsó réteg-beli deriváltakra. Nézzük meg, mi lesz a derivált, ha $softmax$ aktivációs függvényt és negatív loglikelihood hibát számolunk [10]. Jelölje $p_k := P(g(x) = k|\Theta) = softmax(z)_k$ rögzített x-re, $L := -\ln(p_k)$.

$$\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial z_i} = \frac{\partial\left(-\ln(p_k)\right)}{\partial p_i}\frac{\partial p_i}{\partial z_i} \tag{4.1}$$

Ami 0, ha $i \neq k$, különben pedig:

$$\begin{aligned}
\frac{\partial p_k}{\partial z_k} &= \frac{\partial \operatorname{softmax}(z_k)}{\partial z_k} \\
&= \frac{\partial \frac{e^{z_k}}{\sum_{j=1}^{m} e^{z_j}}}{\partial z_k} \\
&= \frac{e^{z_k}\sum_{j=1}^{m} e^{z_j} - e^{2z_k}}{\left(\sum_{j=1}^{m} e^{z_j}\right)^2} \\
&= \frac{e^{z_k}}{\sum_{j=1}^{m} e^{z_j}} - \frac{e^{2z_k}}{\left(\sum_{j=1}^{m} e^{z_j}\right)^2} \\
&= \operatorname{softmax}(z_k)\left(1 - \operatorname{softmax}(z_k)\right) \\
&= p_k(1 - p_k)
\end{aligned} \tag{4.2}$$

Ezt visszahelyettesítve (4.1) egyenlőségbe, azt kapjuk, hogy a hiba az utolsó rétegben a bemenethez tartozó osztálynál

$$\frac{\partial L}{\partial z_i} = \frac{\partial\left(-\ln(p_k)\right)}{\partial p_i}\frac{\partial p_i}{\partial z_i} = -\frac{1}{p_k}p_k(1 - p_k) = p_k - 1$$

4.2. A kereszt entrópia hibafüggvény

A 4.1.1 definícióban feltételeztük, hogy minden elem pontosan egy osztályba tarozik. Gondolhatunk egy kicsit absztaktabban is a problémára, ahol inkább a tanítóhalmazbeli bemenet-kimenet párokra koncentrálunk és nem egy konkrét halmaz partícionálására (osztályokra bontására).

4.2.1. Definíció. *Legyen a lehetséges bemenetek halmaza X, a lehetséges osztályok véges halmaza pedig Y. $A := \{(x, P)|x \in X, P \in \Delta_Y\}$ esetén adott $H \subset A$ tanítóhalmaz és $T \subset A - H$ teszthalmaz mellett absztrakt osztályozási problémának nevezzük amikor egy olyan $g\colon X \to \Delta_Y$ függvényt keresünk, amire $c(T, g)$ a lehető legnagyobb.*

4.2.2. Megjegyzés. Vegyük észre, hogy mind a tanító-, mind a teszthalmazban is egy X-beli elemhez több eloszlás is tartozhat. Általában mégis feltételezhetünk valami mögöttes logikát, aminek a kiismerését várjuk a tanítás során.

4.2.3. Példa. Tegyük fel, hogy van egy (hosszú) $\mathcal{S}$ szövegünk. Legyen X az $\mathcal{S}$ karaktersorozatnak a részsorozatai, és Y a karakterek. A tanítóhalmaz álljon a szövegben előforduló karakter sorozatokból és az őket követő karakterekből, oly módon, hogy ha s karaktersorozat előfordul úgy, hogy egyszer y_1 karakter követi, másszor pedig y_2, akkor legyen a tanítóhalmazban benne az (s, χ_{y_1}) és az (s, χ_{y_2}) is, ahol χ_y jelöli az y karakter indikátor-eloszlását. A teszthalmaz az egyszerűség kedvéért legyen $\emptyset$. Hangzásilag remek szövegek generálhatók ilyen módon tanított *RNN* (később) modellekkel. Kárpáthy példakódja remekez kezdés a karakter alapú szöveggenerálásákhoz [11].

A softmax aktivációs függvénnyel szintén valószínűségi eloszlást kapunk, így ezt továbbra is tudjuk használni. Viszont annak ellenére, hogy a negatív likelihood módszer a fenti példán még alkalmazható lenne, maga a 4.2.1 problémán már nem, lévén, hogy tetszőleges eloszlások is szerepelhetnek a minták között, nem csak indikátor-eloszlások. Ezért most egy általánosabb hibafüggvényt fogunk bemutatni.

Boltzmann 1872-ben az ideális gázok sűrűségének időbeli változásának viszgálatakkor alkotta meg a H-elméletét [13]. Shannon 1948-ban azt az információelméleti kérdést boncolgatta, hogy hogyan lehet üzenetküldés céljából minnél hatékonyabban kódolni az információt. Néhány (intuitív) alaptulajdonság definiálása után azt találta, hogy ha adott egy $P = (p_1, p_2, \ldots, p_n)$ diszkrét eloszlás a lehetséges üzeneteken, akkor az alaptulajdonságok akkor és csak akkor teljesülnek, ha általa először bizonytalanságnak (uncertainly)

$$H(P) = -K \sum_{i=1}^{n} p_i \log(p_i) \tag{4.3}$$

formában írható le, ahol log a 2-es alapú logaritmust jelöli. [12]. Ekkor a $K = 1$ esetet *entrópiának (entropy)* hívjuk. (Valójában a fizikában ugyanez a K a megfelelő fizikai állandóval helyettesítve szintén entrópia névre hallgat.) A továbbiakban használni fogjuk a $H(p_1, p_2, \ldots, p_n) := H((p_1, p_2, \ldots, p_n))$ egyszerűsített jelölést is.

Az entrópiára szokás úgy gondolni, mint a "megjósolhatóság" vagy épp egy komplexitás mérőszámára. Például, ha van N féle üzenetünk ugyanakkor valószínűséggel, akkor az üzenetek kódolására szükségünk van $\log(N)$ bitre, ami pont az egyenletes eloszlás entrópiája, vagy a másik véglet, amikor egyetlen egy fajta üzenetünk van, akkor nyilván elég egyetlen bit is, de ezek fordítva is igazak:

4.2.4. Állítás. *1.* $H(p_1, p_2, \ldots, p_n) = 0$ *akkor és csakk akkor, ha* $\exists 1 \leq i \leq n$, *hogy* $p_i = 1$.

2. $H(P) \leq \log(n)$ $\forall P$ *n elemú diszkrét valószínűségi eloszlásra.*

Visszatérve az eredeti célunkhoz, adva van nekünk egy P eloszlás n elemen és szeretnénk, hogy az általunk predikált Q eloszlás a tanítás során egy jobban "hasonlítson" P-hez. Könnyen látszik, hogy önmagában $H(P)$ és $H(Q)$ viszonya nem biztos, hogy sokat elmond magáról, mert például $H(p_1, p_2, p_3) = H(p_2, p_1, p_3)$, annak ellenére hogy a két eloszlás valójában nem is egyenlő. Ennél egy sokkal alkalmasabb összehasonlítási módot nyújt a *keresztentrópia (cross-entropy).*

4.2.5. Definíció. *Legyen* P *és* Q *egy eloszlás* n *elem felett. Ekkor* $H(P, Q) := -\sum_{i=1}^{n} p_i \log(q_i)$ *értéket* P *és* Q *keresztentrópiájának nevezzük.*

4.2.6. Megjegyzés. $H(P, P) = H(P)$

4.2.7. Megjegyzés. $H(P, Q)$ nem szimmetrikus.

4.2.8. Megjegyzés. Ha $P = (p_1, p_2, \ldots, p_n)$ olyan, hogy $p_i = 1$ valamely i-re, akkor $H(P, Q)$ pont a negatív loglikelihood hiba.

A most következő eredmény garantálja nekünk, hogy ha a lehető legtovább csökkentjük a keresztentrópiát, akkor valóban megkapjuk a P eloszlást. A bizonyítás a [8] oldalon leírtakat követi.

4.2.9. Tétel. (Gibbs egyenlőtlenség) *Legyen* P *és* Q n *elemű diszkrét eloszlás. Ekkor* $H(P, Q) \geq H(P)$ *és* $H(P, Q) = H(P)$ *akkor és csak akkor, ha* $P = Q$.

Bizonyítás. Mivel $\log x = \frac{\ln x}{\ln 2}$ ezért ha az egyenlőtlenség igaz e alapú logaritmussal, akkor iagz marad 2-es alapúval is. A bizonyításhoz használjuk az analízisből ismert következő egyenlőtlenséget.

4.2.10. Állítás. $\ln x \leq x - 1 \ \forall x > 0$ *és egyenlő, akkor és csak akkor, ha* $x = 1$

Legyen $I := \{i | 1 \leq i \leq n \text{ és } p_i > 0\}$. Ekkor a következők igazak:

$$-\sum_{i \in I} p_i \ln\left(\frac{q_i}{p_i}\right) \geq -\sum_{i \in I} p_i \left(\frac{q_i}{p_i} - 1\right) = -\sum_{i \in I} q_i + \sum_{i \in I} p_i \geq -1 + 1 = 0 \tag{4.4}$$

Ahol az első egyenlőtlenség a 4.2.10 miatt, a második pedig mert valószínűségi eloszlások. Átalakítás után, majd visszaírva a $p_i = 0$ tagokat megkapjuk az eredeti egyenlőtlenséget.

$$0 \leq -\sum_{i \in I} p_i \ln\left(\frac{q_i}{p_i}\right) = -\sum_{i \in I} p_i \ln(q_i) + \sum_{i \in I} p_i \ln(p_i) = -\sum_{i=1}^{n} p_i \ln(q_i) + \sum_{i=1}^{n} p_i \ln(p_i)$$

A (4.4) egyenlőséggel teljesül akkor és csak akkor, ha $\forall i \in I$-re $\frac{q_i}{p_i} = 1$ (ekkor a második egyenlőtlenség is egyenlőséggel teljesül), tehát valóban a tétel egyenlőtlensége akkor és csak akkor teljesül egyenlőséggel, ha $p_i = q_i$ minden $1 \leq i \leq n$. □

Ahogy a bizonyításból is kiderül, e alapú logaritmus esetén is igaz marad az állítás, aminek örülünk, mert neuronhálók esetén a deriváltak számolása miatt inkább ez preferált és nem az informatikában megszokott 2-es alapú.

4.3. A word2vec módszer

A *word2vec* módszer [14] a nyelv feldolgozásban egy egyszerű, de annál nagyszerűbb eljárás. A célja az, hogy a hasonló jelentésű szavakat valamilyen metrika szerint egymáshoz közeli vektorokra képezzen le. Teszi mindezt teljesen felügyelés mentesen. A modell tanítása azon az elképzelésen alapszik, hogy a hasonló szavak hasonló kontextusban fordul(hat)nak elő. A következőkben ennek a módszernek a különböző komponenseit ismertetjük.

Adva van egy S szöveg, amire rendszerint *korpusz* néven hivatkoznak. Ez rendszerint hosszabb egybefüggő szövegek összefűzése. Általában különböző előfeldolgozást elvégeznek rajta, mint például gyakori, de jelentést nem hordozó szavak, mint névelők kitörlése, azonban ezek most kevésbbé fontosak nekünk. Ezután *tokenizáljuk* a szöveget, vagyis minden egyes szóhoz hozzárendelünk egy számot. Így minden szó megfelel a $\{1, 2, n_V\}$ egy elemének. Ahol az n_V szótár mérete nyilván a szövegben előforduló különböző szavak száma. Majd egy a méretű ablakot végig csúsztatnak a szövegen. Ez a a méret definiálja, hogy mekkora hosszúságú részeket veszünk kontextusnak a tanulás során.

4.3.1. Definíció. *Tekintsük a szövegben a* k*-adik szót és jelöljük* w_k*-val. Legyen az ablak méret* a *és legyen* t *és* t' *olyan, hogy* $t + t' + 1 = a$*. Ekkor a* k*-adik szó kontextusának nevezzük a következő halmazt:* $\{w_{k-t}, w_{k-t+1}, \ldots, w_{k-1}, w_{k+1}, w_{k+2}, \ldots w_{k+t'}\}$

Ezen fogalmak mentén két különböző 4.2.1-t kielégítő problémát tudunk definiálni.

4.3.2. Definíció. *Vegyük a korpuszban az összes szó összes előfordulását és a hozzájuk tartozó kontextusokat adott t és t' mellett. Vagyis tekintsük (C, w) kontextus és szó párokat, ahol C és w ugyanabból az ablakból származik. Ekkor $H := \{(\hat{C}, \chi_w)\}$, ahol $\hat{C}$ a C kontextus-beli szavak halmaza és χ_w jelöli a w szó indikátor eloszlását a korpusz-beli különböző szavak felett. T legyen üreshalmaz. Az így kapott absztrakt osztályozási problémát Continuous Bag-of-Word, továbbiakban CBOW modellnek nevezzük.*

4.3.3. Megjegyzés. Mivel egy szó többször is előfordulhat a korpuszban, így több különböző kontextushoz is tartozhat ugyanaz a szó, vagy épp fordítva ugyanahhoz a kontextushoz is tartozhat két különböző szó.

4.3.4. Megjegyzés. A kontextusból csak a benne lévő szavak halmazát tekintjük, így a kontextusból elveszítjük a szavak szórendiségét.

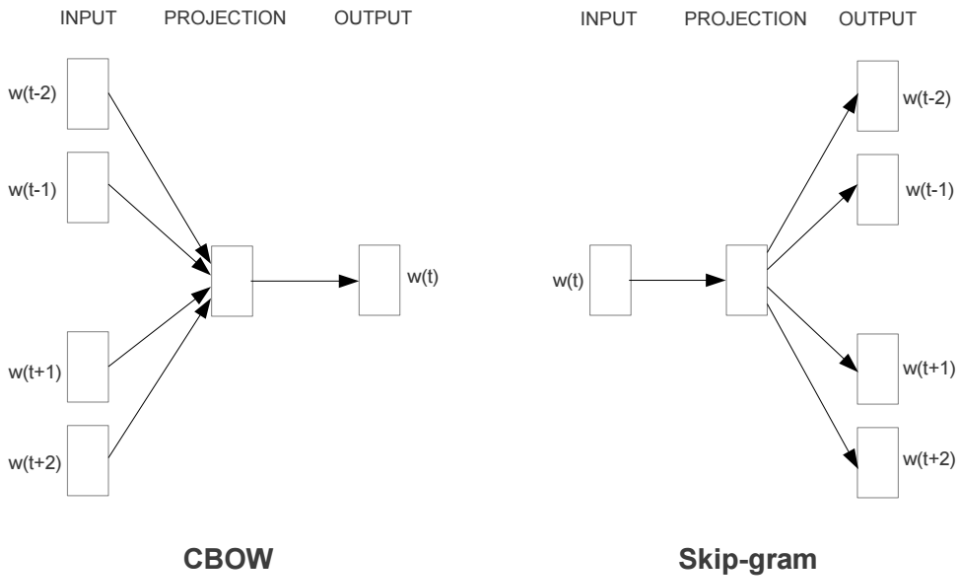
4.3.5. Definíció. *Vegyük a korpuszban az összes szó összes előfordulását és a hozzájuk tartozó kontextusokat adott t és t' mellett. Vagyis tekintsük (C, w) kontextus és szó párokat, ahol C és w ugyanabból az ablakból származik. Ekkor $H := \{(\chi_w, P_C)\}$, ahol P_C jelöli azt az eloszlást, ami a C-n kívüli szavakhoz 0 valószínűséget rendel, míg a C-n belüli szavakon egyenletes eloszlást ír le. T legyen üreshalmaz. Az így kapott absztrakt osztályozási problémát Continuous Skip-gram modellnek nevezzük.*

Nagyon hasonlít ahhoz a nyelvi tesztekhez, amikor hiányos mondatokat kell kiegészíteni. Itt is erről van szó, hogy tekintünk egy részt (egy ablakot) a szövegből, ahol az egyik szót kihagyjuk. A CBOW esetén megpróbáljuk kitalálni, hogy melyik szó maradt ki a tanulás során, a Skip-gram esetén pedig a kihagyott szó alapján próbáljuk kitalálni a szövegkörnyezetet. Azonban maga a word2vec módszer nem egyszerűen ezen problémák megoldása, hanem fontos része, hogy hogyan tesszük mindezt. Ugyanazt a modellt fogjuk definiálni a CBOW és Skip-gram megoldására is. Legyen $x \in \mathbb{R}^{n_V}$ rögzített, ekkor tekintsük a következő két rétegű neuronhálót.

$$\begin{aligned} h &= xW_{xh} \\ y &= \text{softmax}(hW_{hy}) \end{aligned} \tag{4.5}$$

Ahol $W_{xh} \in \mathbb{R}^{n_V \times k}$ és $W_{hy} \in \mathbb{R}^{k \times n_V}$ Ha a bemenet a w szó, akkor x legyen a w-nek megfelelő one-hot vektor, ha pedig $\hat{C}$ szavak halmaza, akkor

$$x_i := \begin{cases} \frac{1}{|\hat{C}|} & \text{ha } i \in \hat{C} \\ 0 & \text{különben} \end{cases} \tag{4.6}$$



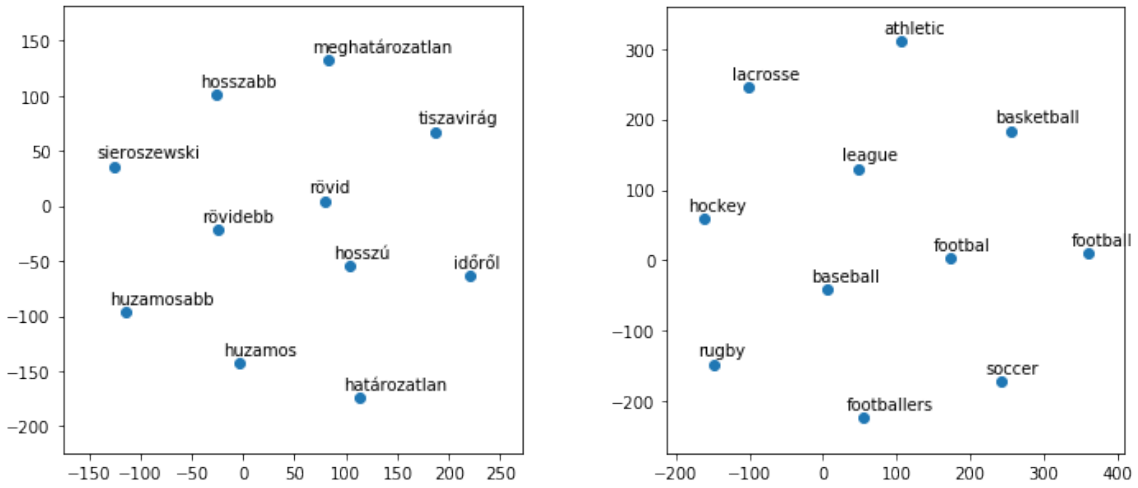
4.1. ábra. Példa a CBOW és a Skip-gram modellre: 5 nagyságú ablakból vesszük ki a középső szót [14]

Hasonló formában kódolhatók a kimenetként várt eloszlások is. Továbbá vegyük észre, hogy (4.5)-ben definiált h rejtett rétegnek nincs aktivációs függvénye. Tegyük fel, hogy a bement a w szó, ami az i-edik a szótárban. Ekkor

$$h = xW_{xh} = \sum_{i=1}^{n_V} x_i W_{xh}^{(i)} = W_{xh}^{(i)},$$

vagyis egy adott szóra, a modellben a belső réteg nem más, mint az első súlymátrix megfelelő sora. Ez lesz az i-edik szó *beágyazása (embedding)*, továbbiakban, csak a szó vektoraként hivatkozunk majd rá. A $W_x h$ pedig a *beágyazó (embedding)*, vagy *kódoló mátrix (encoding matrix)*. Hasonlóan, ha a bemenet egy kontextus, akkor a rejtett h réteg nem más, mint a kontextusban lévő szavak vektorának a mértani közepe. Így ha n_V nagy, amit el is várunk igazából, akkoris h számolásánál megússzuk a nagy méretű mátrixszorzást. Most ismertetünk egy módszert, amivel hasonlóan sok számolást tudunk megspórolni, csak épp a második W_{hy} rétegben. Tanítás során szeretnénk a kereszt-entrópiát használni, mint hibafüggvény, azonban ehhez szükségünk lenne az egész y vektorra, amivel együtt járna a hW_{hy} mátrix szorzás és a softmax kiszámítása, amik szintén nagyon költségesek nagy n_V mellett.

A megoldás meglehetősen intuitív, de a gyakorlatban annál kifizetődőbbnek mtutat-



(a) A "rövid" szó környezete

(b) A "football" szó környezete

4.2. ábra. Példa a beágyazásokra word2vec segítségével [16], bal oldalt a korpusza teljes magyar nyelvű wikipedia volt, míg jobb oldalt az angol nyelvű.

kozik. Ha megnézzük a hibafüggvény kiszámolását

$$L(\hat{y}, y) = -\sum_{i=1}^{n_V} \hat{y}_i \ln(y_i) = -\sum_{i \in \hat{C}} \hat{y}_i \ln(y_i), \tag{4.7}$$

láthatjuk, hogy az n_V dimenziós kimenett nagyon kis részénél kell csak ismernünk. Persze a softmax kiszámolásához, vagy épp a hiba visszaterjesztésénél szükségünk lenne mindegyik kimenetre, azonban a gyakorlati alkalmazás azt mondja, hogy ha jó stratégiával választunk ki kevés kimenetet, a tanítás továbbra is sikeres marad. A módszer neve *negatív mintavételezés* [19]. Ahogy (4.7) mutatja, $\hat{C}$ tartalmazza a "pozitív" mintákat, vagyis, hogy hol kell növelni a kimeneti értéket. A tanítás során a soron következő iterációban egy Q eloszlás szerint válasszunk n db olyan kimenetet, ahol $\hat{y}_i = 0$. Mikolov ajánlása szerint a következő Q eloszlással érdemes elvégezni mindezt:

$$Q(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=1}^{n_V} f(w_j)^{\frac{3}{4}}}, \tag{4.8}$$

ahol $f(w)$ a w szó gyakorisága a korpuszban. Az így kapott $\hat{C}_{\text{neg}}$ és $\hat{C}$ indexeket úgy tekintjük, mintha csak azokból állna a kimenet és aszerint számoljuk a softmax-ot és minden továbbit is a hiba visszaterjesztés során.

4.4. Az item2vec módszer

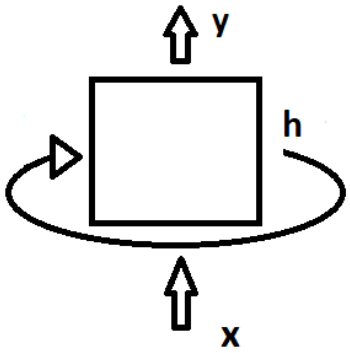
Ahogy a szavak esetén, az írott szövegben van valamiféle mögöttes struktúra, hasonló jelentésű szavak hasonló kontextusban fordulnak elő, úgy feltételezhetjük, hogy valamiféle mintázat lehet abban is, hogy milyen terméket milyen más termékekkel nézünk meg közel egy időben. Az *item2vec* [17] nem más, mint az előbb megismert word2vec módszer alkalmazása a felhasználók és termékek interakiójára. A termékeknek szintén vehetjük a one-hot vektor reprezentációját, valamint tekintsünk úgy a felhasználók előzményeire, mintha azok mondatok lennének. Arra érdemes figyelni, hogy amikor az ablakot végig csúszatjuk az előzményeken, akkor egy ablakba mindig csak egy felhasználótól származó megtekintések legyenek, mert nem hordoz nekünk információt az, hogy mi volt az utolsó termék egy felhasználó előzményébe és mi volt az első egy másikében.

5. fejezet

Rekurrens neuronhálók

Az előre csatolt hálóknak rendszerint meg van határozva, hogy hány dimenziós a bemenetük és a kimenetük. Azonban rengeteg olyan problémakör van, ahol vagy egyik vagy másik hossza nem tudható előre. Ahogy már szó eset a 4.2.3 példában is, ha szöveget szeretnénk generálni, akkor ott is rendszerint megadunk egy kezdeti állapotot, utána pedig egy előre nem meghatározott hosszúságú szöveget szeretnénk kapni. De jó példa lehet a szövegfordítás is, ahol még az sem biztos, hogy a lefordítot szöveg ugyanannyi szóból fog állni, mint a kezdeti. Ezekre kínálnak megoldást a *rekurrens neuronhálók (recurrent neural network - RNN)*, amelyek már tartlamznak visszafele mutató kapcsolatokat is. Tipikus alkalmazásuk közé tartoznak még a hallott szöveg, videó feldolgozások és minden további olyan eset, ahol a bemenet vagy a kimenet változó hosszúságú *szekvencia*.

Általánosságban elmondható, hogy a rekurrens hálók bemenete is kimenete vektorok egy sorozata, amikre rendszerint úgy tekintünk, hogy ezek valamiféle időbeli sorozatok, amiket háló időpillanatról időpillanatra dolgoz fel. Hála a visszamutató kapcsolatoknak elvben képes a háló a lényeges információk megtartani végig a szekvencia olvasás közben, vagyis a szekvencia végén olyan információt is fel tud használni, amit a szekvencia elejéről szűrt ki. A fejezetben körüljárjuk ennek a nehézségeit, ugyanis a tanítás során komoly problémák merülhetnek fel abban, hogy ezen időn átívelő következetéseket és azok hibáit hogyan is tudjuk visszaterjeszteni a hálón eredményesen. Megismerünk majd két népszerű architektúrát *(LSTM, GRU)*, ami a tanítás során felmerülő komplikációkat képes kezelni.



5.1. ábra. A vanilla RNN kompakt ábrázolása, a rejtett réteg önmagával is össze van kötve

5.1. Vanilla RNN

A *vanilla RNN* [21] talán úgy fordítható a legjobban, hogy egyszerű RNN. Nem más, mint egy olyan neuronháló, aminek egyetlen rejtett rétege van, ami önmagával is össze van kötve. Ennk egy kompakt és szemléletes ábrája látható a 5.1 ábrán is. A pontosabb vizsgálódás kedvéért nézzünk előbb egy pontosabb definíciót is.

5.1.1. Definíció. *A* $\left[x^{(t)}\right]_{t=1}^{T}$, $x^{(t)} \in \mathbb{R}^n$ *véges vektor sorozatot* T *hosszú* n *dimenziós szekvenciának fogjuk nevezni.*

5.1.2. Definíció. *Legyenek* $W_{be} \in \mathbb{R}^{n\times k}$, $W_{rek} \in \mathbb{R}^{k\times k}$ *és* $W_{ki} \in \mathbb{R}^{k\times m}$ *élsúly mátrixok,* f *és* g *aktivációs függvények. Ekkor* $\left[x^{(t)}\right]_{t=1}^{T}$, $x^{(t)} \in \mathbb{R}^n$ *bemeneti szekvencia mellet a vanilla rekurrens neuonháló rejtett rétegének, vagy belső állapotának nevezzük* $\left[h^{(t)}\right]_{t=1}^{T}$, $h^{(t)} \in \mathbb{R}^k$ *szekvenciát és kimenetének* $\left[y^{(t)}\right]_{t=1}^{T}$, $y^{(t)} \in \mathbb{R}^m$ *szekvenciát, ha*

$$\begin{aligned} h^{(t)} &= f\left(W_{be}x^{(t)} + W_{rek}h^{(t-1)} + b\right) \\ y^{(t)} &= g\left(W_{ki}h^{(t)}\right). \end{aligned} \tag{5.1}$$

5.1.3. Megjegyzés. A vanilla RNN esetén f rendszerint tanh, g peding a σ, azaz a sigmoid függvény.

5.1.4. Megjegyzés. A kimenet és a belső állapot hosszát a bemeneti szekvencia hosszának definiáltuk. A valóságban ez sokszor jelenthei azt, hogy hogy az $x^{(t)}$ szekvencia utolsó néhány eleme csak valami előre definiált érték, pl a 0 vektor, az $y^{(t)}$ hossza pedig addig tart, amíg nem ad valami speciális megállási értéket. Ilyen és hasonló módszerekkel ez az

architektúra alkalmas a már bevezetőben tárgyalt különböző konstrukciókra, hogy akár 1 vektorból, vagyis 1 hosszú vektorból hogyan lehet hosszabb szekvenciát generálni, ahol az egyes vektorok megfeltethetők karaktereknek.

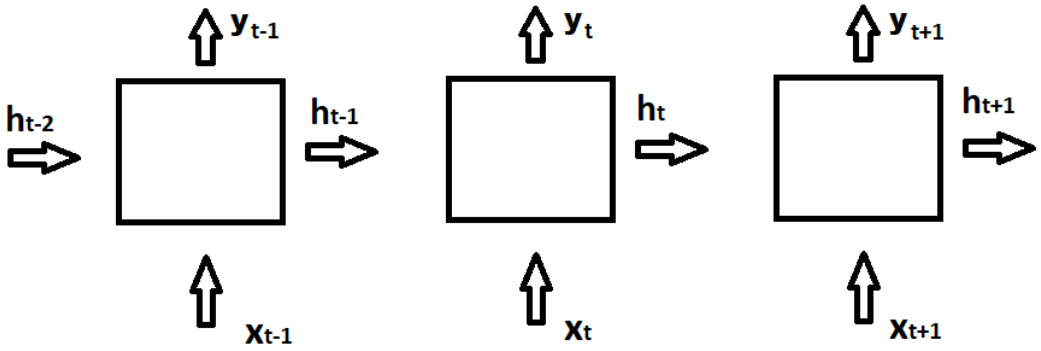
A konstruckió előnye, hogy a h belső állapot valójában egy fajta memóriaként funkcionál, vagyis egy későbbi $y^{(t')}$ kimenetre kihatással lehet egy korábbi $x^{(t)}$ bemenet is. Vagyis képes lehet hosszabb időbeli függőségek felismerésére és kihasználására. Sajnos ezek a remek tulajdonságok csak nagyon nehezen, vagy épp csak alig jellemzők a vanilla RNN-re. Már az a kezdeti intuició gyanakvásra adhat okot, ha arra gondolunk a W_{be} által kiszűrt információt mindig hozzáadjuk a W_{rek} által újra struktúrált memóriához. Érezhetően a szekvencia elejéről kiszűrt információ egyre "halványul" a memóriában. Viszont ezen túl is a tanítás során egzaktan megfigyelhető problémák merülnek fel. Ezek felderítéséhez vizsgáljuk meg a tanítás folyamatát.

A tanítás során itt a hiba visszaterjesztő algoritmust fogjuk használni. Ahogy 5.2 ábrán láthatjuk, valójában egy előre csatolt hálóhoz hasonló architectúra ez is, annyi különbséggel, hogy bemenetet nem csak a legelején kap, hanem sokkal többször, valamint az élsúly mátrixok nem különbözőek rétegenként, hanem nagyon sok helyen szerepel ugyanaz a három W_{be}, W_{rek}, W_{ki} mátrix. Ezért *megosztott (shared)* mátrixoknak nevezzük őket. Rögzített $\left(\left[x^{(t)}\right]_{t=1}^{T}, \left[\hat{y}^{(t)}\right]_{t=1}^{T}\right)$ (továbbiakban csak $\left(x^{(t)}, \hat{y}^{(t)}\right)$) bemenet-kimenet szekvencia esetén legyen az L hibafüggvény a következő:

$$\begin{aligned} L^{(t)} &:= l(y^{(t)}, \hat{y}^{(t)}) \\ L &:= \sum_{t=1}^{T} L^{(t)} \end{aligned} \tag{5.2}$$

valamilyen l veszteség függvény szerint. A hiba visszaterjesztő algoritmust az ilyen időben "kigörgetett" rekurrens hálókon *időn keresztüli hibavisszaterjesztő (backpropogation through time - BPTT)* algoritmusnak hívjuk. A gradiensek kiszámításakkor a legérdekesebb eset a W_{rek} szerinti derivált.

$$\frac{\partial L}{\partial W_{\text{rek}}} = \sum_{t=1}^{T} \frac{\partial L^{(t)}}{\partial W_{\text{rek}}} \tag{5.3}$$



5.2. ábra. A vanilla RNN időbeli kigörgetése, látszik, hogy a rejtett réteg függ az előző rejtett rétegtől és az aktuális bemenettől

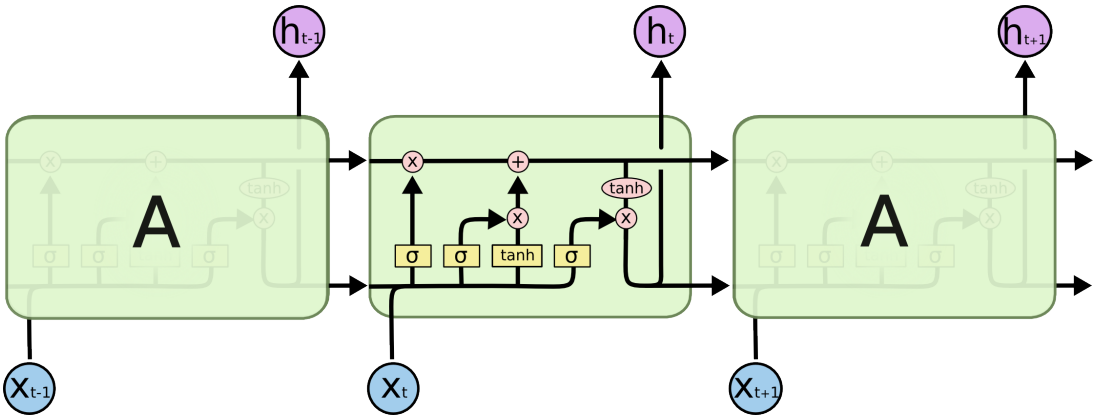
Alkalmazzuk a láncszabályt az összeg egyik tagjára.

$$\begin{aligned}\frac{\partial L^{(t)}}{\partial W_{\text{rek}}} &= \frac{\partial L^{(t)}}{\partial y^{(t)}}\frac{\partial y^{(t)}}{\partial h^{(t)}}\frac{\partial y^{(t)}}{\partial W_{\text{rek}}} \\ &= \frac{\partial L^{(t)}}{\partial y^{(t)}}\frac{\partial y^{(y)}}{\partial h^{(t)}}\left[\frac{\partial^* h^{(t)}}{\partial W_{\text{rek}}}+\right. \\ &+ \frac{\partial h^{(t)}}{\partial h^{(t-1)}}\left(\frac{\partial^* h^{(t-1)}}{\partial W_{\text{rek}}}+\right. \\ &\dots \\ &\left.\left.+ \frac{\partial h^{(3)}}{\partial h^{(2)}}\left(\frac{\partial^* h^{(2)}}{\partial W_{\text{rek}}}+\frac{\partial h^{(1)}}{\partial h^{(1)}}\frac{\partial^* h^{(1)}}{\partial W_{\text{rek}}}\right)\right)\right] \\ &= \sum_{i=1}^{t}\frac{\partial L^{(t)}}{\partial y^{(t)}}\frac{\partial y^{(t)}}{\partial h^{(t)}}\frac{\partial h^{(t)}}{\partial h^{(i)}}\frac{\partial^* h^{(i)}}{\partial W_{\text{rek}}}\end{aligned} \tag{5.4}$$

Ahol a második egyenlőség a láncszabály és a szorzat derivált szabályából következik, az utolsó egyenlőség pedig egy egyszerűbb jelölés az olvashatóság kedvéért, amiben $\frac{\partial^* h^{(i)}}{\partial W_{\text{rek}}}$ jelentse azt, amikor $h^{(i-1)}$-et konstansnak tekintjük a W_{rek} szerinti deriváltban, míg

$$\frac{\partial h^{(t)}}{\partial h^{(i)}} = \prod_{j=i+1}^{t}\frac{\partial h^{(j)}}{\partial h^{(j-1)}} = \prod_{j=i+1}^{t} W_{\text{rek}}^{\top}\,\text{diag}\left(f'\left(h^{(j-1)}\right)\right). \tag{5.5}$$

5.1.5. Tétel. *Legyen λ_1 a W_{rek} legnagyobb abszolútértékű sajátértéke. Ha f aktivációs függvény korlátos, $\|\text{diag}(f'(h_k))\| < \gamma$ (valamilyen $\gamma \in \mathbb{R}$) és $\lambda_1 < \frac{1}{\gamma}$, akkor $\frac{\partial h^{(t)}}{\partial h^{(i)}} \to 0$ $t \to \infty$ esetén. Továbbá, ha $\left\|\frac{\partial h^{(t)}}{\partial h^{(i)}}\right\| \to \infty$, akkor $\lambda_1 > \frac{1}{\gamma}$.*



5.3. ábra. Egy LSTM blokk a t-edik időpillanatban [24]

5.1.6. Megjegyzés. Amikor a derivált 0-hoz tart, úgy nevezzük, hogy gradiens eltűnés, míg a másik véglet a gradiens felrobbanása. A tétel valójában csak egy elégséges feltételt fogalmaz meg az előbbire és egy szükségest azt utóbbira.

Bizonyítás. Először belátjuk az elégséges állítást. A feltételekből könnyen adódik, hogy

$$\left\| \frac{h^{(j)}}{\partial h^{(j-1)}} \right\| < \left\| W_{\text{rek}}^{\top} \right\| \left\| \text{diag}(f'(h^{(j-1)})) \right\| < \frac{1}{\gamma}\gamma = 1.$$

Vagyis $\exists \alpha < 1$, hogy

$$\left\| \frac{h^{(j)}}{\partial h^{(j-1)}} \right\| < \alpha$$

Ekkor (5.5) miatt

$$\left\| \frac{h^{(t)}}{\partial h^{(i)}} \right\| < \prod_{j=i+1}^{t} \left\| \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right\| < \alpha^{t-i} \to 0,$$

ha $t \to \infty$. Az állítás második fele, következik ebből, ugyanis ha $\lambda_1 < 1$, akkor eltűnne a gradiens, nem pedig felrobanna. □

A tétel által vázolt jelenség meglehetősen ellehetetleníti a hosszútávú üsszefüggések tanulását. Több módszer is létezik a probléma kezelésére, például a gradiens normalizálása, vagy nagyságának korlátozása. A következőkben olyan megoldást veszünk szemügyre, ahol maga a háló architecturája igyekszik kiküszöbölni a gradiens eltűnését vagy épp felrobbanását.

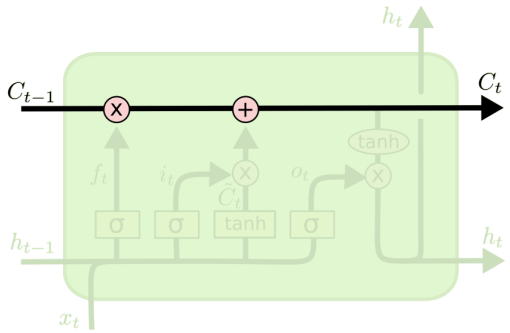
5.2. Long short-term memory (LSTM)

A *long short-term memory (LSTM)* arhcitektúra először 1997-ben jelent meg [22]. A visszacsatolást több valamivel bonyolultabb módszer kíséri. Azóta több változata is van, de a legismertebb továbbfejlesztése, amire azóta a legtöbben gondolnak 2000-ben került publikálásra [23]. Mi is ezt az utóbbi definíciót fogjuk alapul venni. A $*$-gal fogjuk jelölni két vektor elemenkénti szorzását.

5.2.1. Definíció. *Legyenek W_f, W_i, W_o és W_c élsúly mátrixok b_f, b_i, b_o és b_c a megfelelő eltolás vektorok. Ekkor $\left[x^{(t)}\right]_{t=1}^{T}$, $x^{(t)} \in \mathbb{R}^n$ bemeneti szekvencia mellett a t-edik időpillanatban jelölje $f^{(t)}$ a felejtő kaput, $i^{(t)}$ a bemeneti kaput, $o^{(t)}$ a kimeneti kaput, $\tilde{c}^{(t)}$ az új lehetséges memóriát vagy cella kimenetet, $c^{(t)}$ a memóriát vagy cellát és $h^{(t)}$ a cella kiemenetét.*

$$\begin{aligned} f^{(t)} &= \sigma\left(W_f[h^{(t-1)}, x^{(t)}] + b_f\right) \\ i^{(t)} &= \sigma\left(W_i[h^{(t-1)}, x^{(t)}] + b_i\right) \\ o^{(t)} &= \sigma\left(W_o[h^{(t-1)}, x^{(t)}] + b_o\right) \\ \tilde{c}^{(t)} &= \tanh\left(W_c[h^{(t-1)}, x^{(t)}] + b_c\right) \\ c^{(t)} &= f^{(t)} * c^{(t-1)} + i^{(t)} * \tilde{c}^{(t)} \\ h^{(t)} &= o^{(t)} * \tanh\left(c^{(t)}\right) \end{aligned} \tag{5.6}$$

A fent definiált mechanizmus egy blokkokba zárja a visszacsatalolást és az azt segítő további műveleteket. Az időbeli kigörgetésnél a $h^{(t)}$ a blokk kimenete, ezt és a $c^{(t)}$ cella kimenetet használja fel az időben rákövetkező blokk. A működés egyik legismertebb ábrázolása a 5.3 ábra, és további remek magyarázó ábrák találhatók Christopher Olah blog bejegyzésében [24]. Kezdetben külön tanító eljárásokat használtak az LSTM esetén, de mint később kiderült a BPTT hatékonyabb tud lenni ennél [25]. Ami a vanilla RNN esetén



5.4. ábra. A cella tartalma szabályozottan kerül frissítésre, ezért több hasznos információt tud megtartani a régmúltról és kevesebb haszontalan információ kerül bele a jelenből. Ugyanezek a műveletek a gradiens számolásnál kiegyensúlyozzák a hibavisszaterjesztést, így kevésbé hajlamos a gradiens eltűnni vagy épp felrobbani.

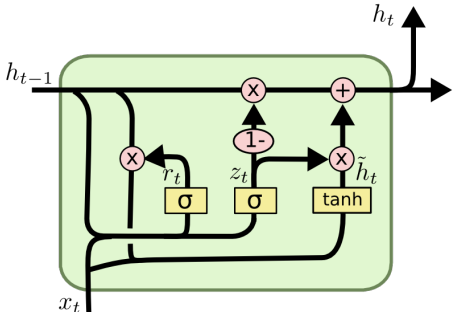
a $h^{(t)}$ volt, itt a $c^{(t)}$ az ami érdekes a hiba visszaterjesztés során.

$$
\begin{aligned}
\frac{\partial c^{(t)}}{\partial c^{(t-1)}} &= \frac{\partial c^{(t)}}{\partial f^{(t)}} \frac{\partial f^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial c^{(t-1)}} \\
&+ \frac{\partial c^{(t)}}{\partial i^{(t)}} \frac{\partial i^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial c^{(t-1)}} \\
&+ \frac{\partial c^{(t)}}{\partial \tilde{c}^{(t)}} \frac{\partial \tilde{c}^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial c^{(t-1)}} \\
&+ \frac{\partial^* c^{(t)}}{\partial c^{(t-1)}} \\
&= c^{(t-1)} * \sigma'\left(W_f \left[h^{(t-1)}, x^{(t)}\right]\right) W_f * o^{(t-1)} \tanh'\left(c^{(t-1)}\right) \\
&+ \tilde{c}^{(t)} * \sigma'\left(W_i \left[h^{(t-1)}, x^{(t)}\right]\right) W_i * o^{(t-1)} \tanh'\left(c^{(t-1)}\right) \\
&+ i^{(t)} * \tanh'\left(W_c \left[h^{(t-1)}, x^{(t)}\right]\right) W_c * o^{(t-1)} \tanh'\left(c^{(t-1)}\right) \\
&+ f^{(t)}
\end{aligned}
$$

Könnyen találhatók olyan bemenet-kimenet párok, ahol $\frac{\partial c^{(t)}}{\partial c^{(t-1)}}$ elemeinek abszolút értéke nagyobb vagy épp kisebb 1-től [26]. Épp ezért a gradiens nem tud olyan egyértelmű körülmények között felrobbani vagy épp eltűnni.

5.3. Gated Recurrent Unit (GRU)

A *Gated Recurrent Unit (GRU)* [27] az LSTM egy variánsa, azonban valamivel egyszerűbb struktúrájú, mint nagyobb múltú elődje. Az LSTM-nél bemutatott számolásoktól most



5.5. ábra. GRU blokk: LSTM-hez hasonló, de egyszerűbb kialakítású [24]

eltekintünk, hasonló mechanizmus megmutatható itt is. Sok esetben a GRU és LSTM egymás egyenrangú alternatívái, de sok esetben a GRU mégis eredményesebbnek bizonyul [28], talán az egyszerűbb architektúra miatt. De erős túlzás lenne azt állítani, hogy minden esetben jobb a GRU-t használni az LSTM kárára.

5.3.1. Definíció. *Legyenek W_z, W_r és W élsúly mátrixok b_z, b_r és b a megfelelő eltolás vektorok. Ekkor $\left[x^{(t)}\right]_{t=1}^{T}$, $x^{(t)} \in \mathbb{R}^n$ bemeneti szekvencia mellett a t-edik időpillanatban jelölje $z^{(t)}$ a frissítő kaput, $r^{(t)}$ a reset kaput, $\tilde{h}^{(t)}$ a memóriát, $h^{(t)}$ a végső memóriát.*

$$\begin{aligned} z^{(t)} &= \sigma\left(W_z\left[h^{(t-1)}, x^{(t)}\right] + b_z\right) \\ r^{(t)} &= \sigma\left(W_r\left[h^{(t-1)}, x^{(t)}\right] + b_r\right) \\ \tilde{h}^{(t)} &= \tanh\left(W\left[r^{(t)} * h^{(t-1)}, x^{(t)}\right] + b\right) \\ h^{(t)} &= (1 - z^{(t)}) * h^{(t-1)} + z^{(t)} * \tilde{h}^{(t)} \end{aligned} \tag{5.7}$$

A működésének szemléletes interpretációja a következő. A frissítő kapu szabályozza azt, hogy mennyi iformációt használjunk a múltból, míg a reset kapu befolyásolja azt, hogy mennyit felejtsünk el belőle. A reset kaput az aktuális memória meghatározásánál használjuk, míg a frissítő kaput már a blokk kimenetének a kiszámításához.

A fejezet végén még jegyezzük meg, hogy az RNN-ek tanításánál is alkalmazhatjuk a mini-batch alapú tanítást.

5.3.2. Megjegyzés. A szekvenciák is rendezhetők batchekbe. Azonban ilyenkor egy batch értelemszerűen nem egy mátrix lesz, hanem egy tenzor, ahol az i-edik minta a batchben egy szekvencia, aminek j-edik eleme a szekvencia j-edik elemét leíró vektor. Mivel a szekvenciák hossza nem szükségszerűen egyforma, ezért a batchbe rendezésnél a leghosszabb szekvencia határozza a tenzor megfelelő dimenzióját, ilyenkor a rövidebb szekvenciák végére megfelelő mennyiségű 0-t írnak.

6. fejezet

GRU4Rec

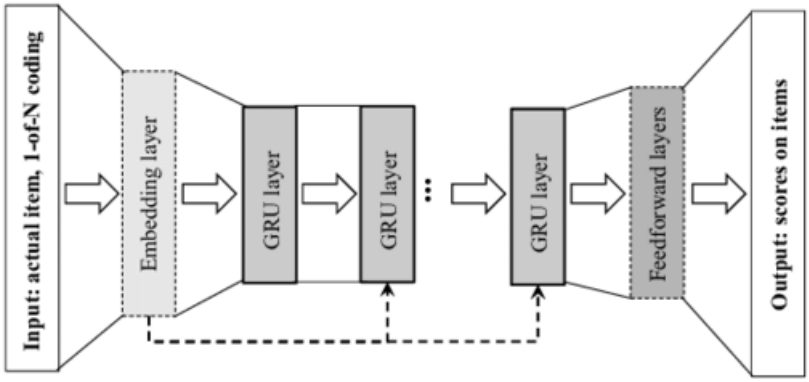
A *GRU4Rec* [18] egy modell alapú CF ajánló rendszer session alapú ajánlásokra. Az alapvető ötlet, hogy a session-öket, mint szekvenciákat valamilyen RNN-nel dolgozzák fel és a háló kimenete alapján javasolnak a felhasználónak termékeket.

6.1. Háló architektúra

A háló bemenete (4.6) típusú, nyilván termékekre értve. Ezután a szerzők ajánlanak egy beágyazó réteget. Azért célszerű ilyenkor előre betanított beágyazást alkalmazni, mert ha nem lenne, végeredményül a tanítás után amúgy is kapnánk egy olyan réteget, ami ezt a célt szolgálná, de cserébe a tanítási időt túlzottan megnövelné. A mélyhálók esetén egy gyakran alkalmazott technika, hogy hibavisszaterjesztésnél biztosítanak, egy "rövidebb" utat is, hogy a későbbi rétegekből kellő mennyiségű információ visszajusson a korábbi rétegekbe a gradiens számolásnál, hasonlót láthattunk az LSTM esetén is a 5.4 ábrán. Itt is a beágyazó réteg össze van kötve további mélyebben fekvő GRU blokkokkal. A szerzők kipróbáltak más rekurrens típust is, úgy mint LSTM vagy épp vanilla RNN, de a GRU blokkokat találták a leghatékonyabbnak.

6.2. Batch alapú tanítás

A különböző session-ök nagyon különböző hosszakkal bírhatnak. Ezért a 5.3.2 megjegyzésben említett 0-kal való kiegészítés egyáltalán nem lenn hatékony. Nagyon sok olyan batch-et kapnánk, ami rövid időn belül csak egy-két szekvenciára végezne valódi tanítást. Ebben az esetben ezért kidolgoztak egy *párhuzamos mini-batch (parallel mini-batch)* nevű



6.1. ábra. A cikk tapasztalati eredményei között szerepel, hogy egy beágyazó réteg növeli a performanciát. Továbbá érdemes lehet a beágyazást összekötni mélyebben fekvő GRU blokkokkal is, így a hiba visszaterjesztés alatt "rövidebb utat" tesz meg a gradiens számolás. ami így ellensúlyozza a háló mélységéből származó lehetséges gradiens eltűnést. [18]

	t_1	t_2	t_3	t_4	t_5	t_6
S_1	$i_{1,1}$	$i_{1,2}$	$i_{1,3}$	0	0	0
S_2	$i_{2,1}$	$i_{2,2}$	$i_{2,3}$	$i_{2,4}$	$i_{2,5}$	$i_{2,6}$
S_3	$i_{3,1}$	$i_{3,2}$	0	0	0	0

6.1. táblázat. Közönséges szekvenciális mini-batch-ek

módszert. Ehhez először vegyük a session-ök, vagy szekvenciák egy véletlen permutációját. És a k elemű batchben, a t-edik időben ha véget ért az egyik szekvencia, akkor nem 0-át írunk utána, hanem elkezdük a következő, még fel nem használt szekvenciát. Arra kell figyelni a tanítás során, hogy az ilyen váltásokhoz tartozó időpillanatban a rejtett rétegek újra legyenek inicializálva. Azonban a gyorsaság kedvéért a háló tanításánál mellőzni kényszerülünk a már korábban megimsert BPTT algoritmust. Egyszerűen túl változatos a session-ök hossza, így a hosszak egy batchen belül sem lennének kellően kiegyensúlyozva. Ahogy a szerzők Github repójánál az issue-kban olvasható, ez egy olyan háló, amit tanításnál csak Markov folyamatnak tekintenek, de használatnál RNN-nek [31].

t_1	t_2	t_3	t_4	t_5	t_6	
$i_{1,1}$	$i_{1,2}$	$i_{1,3}$	$i_{4,1}$	$i_{4,2}$	$i_{4,3}$	...
$i_{2,1}$	$i_{2,2}$	$i_{2,3}$	$i_{2,4}$	$i_{2,5}$	$i_{2,6}$	...
$i_{3,1}$	$i_{3,2}$	$i_{5,1}$	$i_{5,2}$	$i_{5,3}$	$i_{5,4}$	...

6.2. táblázat. Párhuzamos mini-batchek: a szekvenciális batch-eket a 0-val való kitöltés helyett a következő batch-ből származó session-ökkel kezdi feltölteni.

6.3. Hibafüggvény

A pontonkénti összehasonlító függvények, mint a kereszt-entrópia túl instabilnak bizonyul, ha csak nem direktben negatív likelihoodot számolunk. Ezért a cikk a páronkénti összehasonlító hibafüggvényeket preferálja és konstruál is külön erre a feladatra alkalmas függvényt. Általuk *TOP1* hiba az s session egy időpillanatában:

$$L_s = \frac{1}{N_s} \sum_{j=1}^{N_s} \left[\sigma\left(\hat{r}_{sj} - \hat{r}_{si}\right) + \sigma\left(\hat{r}_{sj}^2\right) \right], \tag{6.1}$$

ahol $\hat{r}_{sk}$ a háló kimenete az s session már beolvasott része után a k-adik termékre, i pedig a valóságban a sessiont követő termék. A termékek hatalmas darabszáma miatt itt is negatív mintavételezést alkalmaztak, ahol a negatív minták kiválasztásának a valószínűsége szintén arányos volt a tanítóhalmazban való előfordulásuk gyakoriságával, mint ahogy láttuk ezt a word2vec esetén is.

6.4. Eredmények a lastfm adathalmazon

Az adatok tisztításában a [18] cikket követtük. A legalább 5-ször meghallgatott zeneszámokat tartottuk meg, valamit töröltük azokat a felhasználókat, akik csak egyetlen egy zenét hallgattak meg. Ezután a session-ök körülbelül 20%-át félretettük teszt halmaznak. Ugyanezt a felosztást használtuk mindegyik modell kiértékelésénél.

Az alap modellekhez a turicreate implementációkat használtuk fel [30]. Sajnos számolási kapacitás híján nem volt lehetőségünk az eredetihez hasonló nagyságrendű paraméterek használatára. A futtatáshoz a szerzők eredeti kódját használtuk fel [31]. Eredetileg 2048 volt a hatékony negatív minták száma, mi most csak 15-t használtunk és nem 10, hanem csak 5 epochig tanítottunk. A nagy méretű teszthalmazokon az evaluáció is elég költséges

	Precission@5	Recall@5	Precission@10	Recall@10
Népszerűségi	0.185722	0.009933	0.165137	0.017441
Jaccard	0.185722	0.009933	0.156649	0.016074
Koszinusz	0.185722	0.009933	0.159505	0.016535
Pearson	0.185722	0.009933	0.165137	0.017441
GRU4Rec - kereszt-entrópia	–	–	–	–
GRU4Rec - top1	–	–	–	–

6.3. táblázat. Látszik, hogy rövid ajánló lista esetében nincs számottevő különbség az alapmodelleknél.

	Precission@20	Recall@20	Precission@100	Recall@100	MRR@20	MRR@100
Népszerűségi	0.146098	0.029774	0.024910	0.109444	0.179886	0.032211
Jaccard	0.138145	0.027648	0.057753	0.285190	0.179886	0.107665
Koszinusz	0.140545	0.028359	0.058518	0.290448	0.179886	0.105992
Pearson	0.146098	0.029774	0.024913	0.109448	0.179886	0.032204
GRU4Rec - kereszt-entrópia	–	0.000325	–	–	0.000051	–
GRU4Rec - top1	–	0.005564	–	–	0.001107	–

6.4. táblázat. Az 5 epoch-os tanítás a kevés negatív mintavételezéssel sajnos még nagyon alulmúlja az alap modelleket. Látszik, hogy egy hosszabb ajánlási listánál már a koszinusz és Jaccard modell számottevően jobban teljesít.

volt, így megtartottuk a cikk szerinti metrikákat. A 6.3 és 6.4 táblázatokban a végső futások eredménye látható. A metrika neve mögötti szám jelöli, hogy az evaluációnál milyen hosszú ajánló listát használtunk.

Irodalomjegyzék

[1] https://www.last.fm/ 2019.05.30.

[2] O. Celma *Music Recommendation and Discovery in the Long Tail* 3. fejezet, Springer, 2010

[3] MOHAMED ALY *Survey on Multiclass Classification Methods*, 2005 (https://www.cs.utah.edu/ piyush/teaching/aly05multiclass.pdf)

[4] GEERT LITJENS, THIJS KOOI, BABAK EHTESHAMI BEJNORDI, ARNAUD ARINDRA ADIYOSO SETIO, FRANCESCO CIOMPI,MOHSEN GHAFOORIAN, JEROEN A.W.M. VAN DER LAAK, BRAM VAN GINNEKEN, CLARA I. SANCHEZ *Survey on Deep Learning in Medical Image Analysis*, 2017 (https://arxiv.org/pdf/1702.05747.pdf)

[5] STUART RUSSELL, PETER NORVIG *MESTERSÉGES INTELLIGENCIA*, 2005

[6] SEBASTIAN RUDER *An overview of gradient descent optimization algorithms*, 2016

[7] https://medium.com/@u39kun/is-the-term-softmax-driving-you-nuts-ee232ab4f6bd (2019. 05. 13.)

[8] JOSIAH WILLARD GIBBS *Elementary principles in statistical mechanics* 32-35. old., Scribner's sons, 1902

[9] NAGY KÁROLY *Termodinamika és statisztikus mechanika* 227. old, Tankönykiadó Vállalat, 1991

[10] https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/ 2019. 05. 12.

[11] https://gist.github.com/karpathy/d4dee566867f8291f086 (2019. 05. 12.)

[12] C. E. SHANNON *A Mathematical Theory of Communication* 10-12. old. 1948 (http://math.harvard.edu/ ctm/home/text/others/shannon/entropy/entropy.pdf) 2019.05.13.

[13] CÉDRIC VILLANI *H-therom and beyond: Boltzmann's entropy in today's mathematics* (https://www.math-berlin.de/images/stories/villani-boltzmann-talk.pdf) 2019.05.13.

[14] TOMAS MIKOLOV, KAI CHEN, GREG CORRADO, JEFFREY DEAN, *Efficient Estimation of Word Representations in Vector Space*, 2013 (https://arxiv.org/abs/1301.3781)

[15] TOMAS MIKOLOV, WEN-TAU YIH, GEOFFREY ZWEIG *Linguistic Regularities in Continuous Space Word Representations* (https://www.aclweb.org/anthology/N13-1090)

[16] CSABAI ISTVÁN, RIBLI DEZSŐ, PATAKI BÁLINT ÁRMIN, BAGOLY ATTILA *Deep learing and machine learning in science notes* https://patbaa.github.io/physdl/ (2019.05.21.) 10. előadás

[17] OREN BARKAN, NOAM KOENIGSTEIN *ITEM2VEC: Neural Item Embedding for Collaborative Filtering* (https://arxiv.org/ftp/arxiv/papers/1603/1603.04259.pdf)

[18] BALÁZS HIDASI, ALEXANDROS KARATZOGLOU, LINAS BALTRUNAS, DOMONKOS TIKK *Session-Based Recommendations with Recurrent Neural Networks*, 2016 (https://arxiv.org/pdf/1511.06939.pdf)

[19] XIN RONG *word2vec Parameter Learning Explained*, 2014 (https://arxiv.org/pdf/1411.2738.pdf)

[20] YOAV GOLDBERG, OMER LEVY *word2vec Explained: Deriving Mikolov et al.'s Negative Sampling Word-Embedding Method*, 2014

[21] RAZVAN PASCANU, TOMAS MIKOLOV, YOSHUA BENGIO *On the difficulty of training Recurrent Neural Networks* 2013 (https://arxiv.org/pdf/1211.5063.pdf - 2019.05.24.)

[22] S. HOCHREITER, J. SCHMIDHUBER *Long short-term memory.* Neural Computation, 9(8):1735–1780,1997

[23] FELIX A. GERS, JÜRGEN SCHMIDHUBER, FRED A. CUMMINS *Learning to Forget: Continual Prediction with LSTM* 2000

[24] CHRISTOPHER OLAH *Understanding LSTM Networks* http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (2018.05.26.)

[25] ALEX GRAVES, JÜRGEN SCHMIDHUBER *Framewise Phoneme Classification withBidirectional LSTM and Other Neural NetworkArchitectures* 2005

[26] https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html 2019.05.29.

[27] KYUNGHYUN CHO, BART VAN MERRIENBOER, CAGLAR GULCEHRE, DZMITRY BAHDANAU, FETHI BOUGARES, HOLGER SCHWENK, YOSHUA BENGIO *Learning Phrase Representations using RNN Encoder–Decoderfor Statistical Machine Translation* 2014

[28] JUNYOUNG CHUNG, CAGLAR GULCEHRE, KYUNGHYUN CHO, YOSHUA BENGIO *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*

[29] FRANCESCO RICCI, LIOR ROKACH, BRACHA SHAPIRA *Recommender Systems Handbook*, Springer, 2011

[30] https://apple.github.io/turicreate/docs/api/generated/turicreate.recommender.item$_s imilarity_r ec$ //*github.com/hidasib/GRU*4*Rec*2019.05.30.