Korpusztisztítás és sorvégi kötőjelek kezelése karakteralapú neurális nyelvmodellel

Pethő Gergely, Sass Bálint, Simon László, Lipp Veronika

Nyelvtudományi Kutatóközpont
pagstudium@gmail.com
{sass.balint,simon.laszlo,lipp.veronika}@nytud.hu

Kivonat Cikkünk célkitűzése kettős: egyrészt bemutatunk egy olyan egyszerű és általános módszert, amellyel karakteralapú nyelvmodellek hasznosíthatóak egyebek mellett korpuszok tisztításában, másrészt ismertetünk egy olyan konkrét, tiszta magyar sajtónyelvi korpuszon tanított nyelvmodellt, amelyre építve jó eredményeket értünk el e módszer alkalmazásával. Továbbá nyilvánosan elérhetővé tesszük az akár karakter-, akár szószintű rekurrens neurális nyelvmodellek konfigurálását és (újra)tanítását szolgáló, Pythonban írt alkalmazást, amellyel a nyelvmodellünket tanítottuk, és amelynek segítségével akár ez a magyar sajtónyelvi modell hozzáigazítható más jellegű tanítókorpuszokhoz, akár új modell tanítható be. A bemutatott kétirányú LSTM-nyelvmodell erőforrásigénye aránylag szerény, és a javasolt módszert követve közvetlenül, vagyis az adott részfeladatra történő bármilyen további betanítás nélkül jól használható a korpusztisztítás során felmerülő feladatok széles körére, például idegen nyelvű, túl sok zajt tartalmazó szövegrészek azonosítására, szórványos OCR-hibák és hiányzó ékezetek javítására. A nyelvmodellt a sorvégi elválasztások egyértelműsítése feladatra értékeltük ki: a módszer teljesítménye ezen a feladaton meghaladta a nagyon magas baseline-t.
Kulcsszavak: karakteralapú nyelvmodell, n-gram-modell, LSTM, kétirányú nyelvmodell, autoregresszió, OCR, hibajavítás, korpuszok előfeldolgozása

1. Bevezetés

Az olyan kisebb méretű, kézzel összeállított korpuszok kivételével, mint amilyen például a *Magyar történeti szövegtár,* az *Ómagyar korpusz* vagy akár a *BNC* (Csengery és mtsai, 2016; Simon és Sass, 2012; Aston és Burnard, 1998), a „tisztítás", „zajszűrés" problematikája szinte minden esetben felmerül. Egy méretét tekintve szavak százmillióit tartalmazó szövegadatbázis rendszerint a webaratásnak nevezett technikával állítható elő, esetleg kisebb-nagyobb részkorpuszok kombinálásával vagy optikai karakterfelismertetéssel kapott szövegek felhasználásával. Ezekkel a módszerekkel gyorsan építhető nagyméretű korpusz, azonban annak anyagát – éppen a források sajátosságai miatt – szinte minden esetben több szempontból is javítani kell, illetve kiszűrni belőle a nem javítható részeket. Egy adott szövegadatbázis nyelvészeti, lexikográfiai célú felhasználása esetében az elvárások akár különösen szigorúak is lehetnek.

A korpusztisztításnak a tanulmányunkban részletesen ismertetett speciális részfeladatára a *visszaválasztás* (dehyphenation) terminust használjuk. Ezen a sorvégi elválasztójelek eltávolítását értjük oly módon, hogy helyreállítjuk az elválasztott szavak elválasztás előtti helyes formáját. A feladat nem triviális, négy lehetőség között kell dönteni: (1) Jelentős számban az egyszerű esettel találkozunk, mikor egyszerűen törölhetjük a kötőjel+sortörés kombinációt (‘*kere-/tes*’ → ‘*keretes*’). (2) Létezik ennek a digráfkezeléssel nehezített formája (‘*hosz-/szú*’ → ‘*hosszú*’). (3) További eset, mikor megmarad a kötőjel (‘*egyszer-/kétszer*’ → ‘*egyszer-kétszer*’). (4) Végül előfordul, hogy a sortörést szóközre kell cserélni: ‘*bal-/és jobboldali*’ → ‘*bal- és jobboldali*’. Ez a probléma mindenhol felmerül, ahol a kiindulópont egy elválasztásokat tartalmazó nyomtatott mű.

A kötőjel (diviz, „-”) írásjelnek csak az egyik funkciója a sorvégi elválasztás jelölése, a további lehetséges pozíciók feltérképezéséhez a magyar helyesírás részletszabályainak ismerete szükséges. A kötőjel a szövegekben előfordulhat a szóhatár jelöléseként toldalékok kapcsolásakor *(Tolouse-ban, 2022-ben)* vagy összetételekben *(balett-táncos, négy-öt)*, kettős családnevekben és földrajzi nevekben *(Batthyány-Strattmann, Decsi-Nagy-Holt-Duna)*, de szótagoláskor is *(á-bé-cé)*. Általában tapad az előtte álló szó utolsó és az utána következő szó első betűjéhez, de nem mindig, és ritkán megismétlődhet a sor elején (ha egy kötőjellel írt szó éppen a kötőjelnél kerül elválasztásra). A kötőjelnek a *„bükk-, tölgy- és kőrisfák”* vagy a *„gépgyártó, -szerelő és -javító műhely”* mellérendelő szerkezetekben való megjelenése tipográfiailag nagyon hasonló az elválasztójelként való használatához, ami szintén nehézséget okozhat az utóbbi beazonosításában.

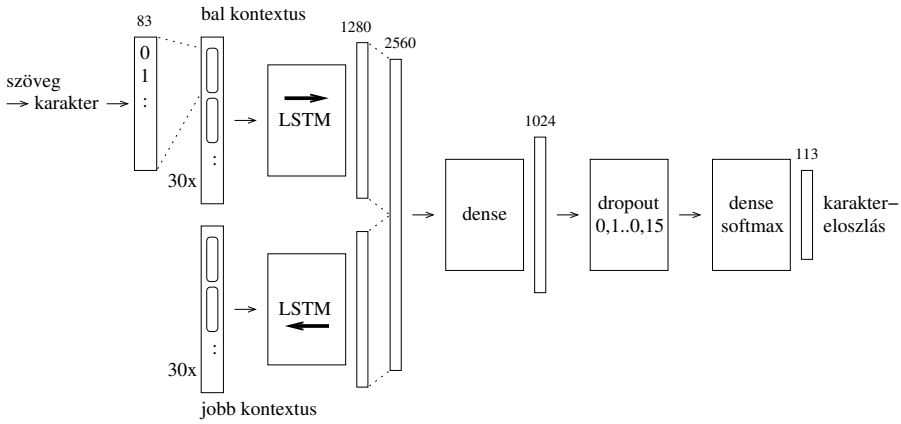
Annak részletezése, hogy a szöveges anyagokban található különféle hibákat és zajt mi okozza, és ezek hogyan javíthatók, szétfeszítené ennek a tanulmánynak a kereteit. Számos ilyen természetű probléma a szövegek OCR-ezéséből ered. Az OCR-hibák kézi javítása költséges és időigényes, ezért régóta kutatják az erre szolgáló automatikus módszereket. Ismert, hogy az írásjel képének önmagában való azonosítása esetében az output mindenképpen több hibát tartalmaz, mint akkor, ha az OCR-motor rendelkezik beépített nyelvmodellel (Mori és mtsai, 1999; Cheriet és mtsai, 2007). Laki és mtsai (2022) kutatásaik során OCR-hibákat tartalmazó szövegeket detektáló és javító modelleket implementáltak, valamint saját silver standard párhuzamos korpuszt építettek. Arra a megállapításra jutottak, hogy amikor kizárólag az OCR-hibák javítása volt a cél, akkor a modelljeiknek a Context-based Character Correction (CCC) modellel való kombinációja mutatkozott a leghatékonyabbnak.

2. A nyelvmodell kialakítása és jellemzői

Jelen tanulmányban a korpusztisztítási és -javítási feladatok megoldására nyelvmodellek által előállított valószínűségi predikciók közvetlen felhasználását javasoljuk a 3. részben leírt módon. Az alkalmazott nyelvmodellt Python nyelven, a TensorFlow gépi tanulási és a Keras mélytanulási keretrendszerben implementáltuk. A modell a jól ismert LSTM (Hochreiter és Schmidhuber, 1997) architektú-

rán alapul, hagyományos LSTM-rétegeket építettünk be egy céljainkra alkalmas saját fejlesztésű kétirányú autoregresszív neurális modellbe.

Amint ismert, az LSTM a rekurrens neurális hálók (RNN; Goodfellow és mtsai, 2016, 10. fejezet, Goldberg, 2017, 14. fejezet, Kamath és mtsai, 2019, 7. fejezet) egyik típusa. Az RNN-ek az előzményszekvenciának egy rögzített dimenzionalitású vektorreprezentációját állítják elő oly módon, hogy ezt a reprezentációt a k darab kontextuselem olvasása során lépésről lépésre, elemenként frissítik. Ez a reprezentáció az egyszerű Markov- vagy más néven n-gram-modellekkel (vö. Jurafsky és Martin, 2021, 3. fejezet) ellentétben nem diszkrét, hanem folytonos, és ennek köszönhetően a rekurrens neurális hálók sokkal sikeresebben képesek általánosítani a tanítóadatok alapján.



1. ábra: A modell architektúrája. A számok a dimenziót jelzik.

Az általunk betanított modell architektúrája az 1. ábrán látható. Bemenete nyers, UTF–8 kódolású szöveg, amelynek karaktereit első lépésben egy karakterkódoló egy-egy ritka numerikus vektorrá alakítja. A neurális háló bemeneti rétegét e vektorok egy $2k$ szélességű szekvenciája alkotja, ebből az első/második k vektor a végső kimenetként keresett középső karakter bal/jobb kontextusában szereplő karaktereket reprezentálja. Az első k vektort egy balról jobbra haladó, a második k vektort pedig egy jobbról balra haladó LSTM-réteg dolgozza fel. Mindkét LSTM-réteg kimenete egy-egy vektor, amely a bal, illetve jobb kontextus egészét reprezentálja. E két vektor konkatenációja a bemenete a következő perceptronrétegnek, ezt dropout és egy újabb perceptronréteg követi, amelynek softmax aktivációval kapott kimenete a lehetséges középső karakterek valószínűségi eloszlását közelíti. A nyelvmodell mérete mai mércével mérve aránylag kicsi, mintegy 11 millió paramétert tartalmaz. Ennek köszönhetően a modell gépigénye nem jelentős, memóriaigénye gyakorlatilag elhanyagolható, egy korszerű asztali gépen akár CPU-n is viszonylag jól futtatható, egy mondatnyi bemenetet érzékelhető késleltetés nélkül fel tud dolgozni. Geforce RTX 3060-as GPU-n 1 millió karaktert kb. 3,5 perc alatt prediktál, így hosszabb korpuszok feldolgozására akár egyetlen asztali gépen is alkalmas.

Tanítás során a szövegen egy $2k + 1$ karakter szélességű ablakot léptetünk egyesével úgy, hogy a középső karakternek megfelelő one-hot vektort kezeljük a neurális háló kimeneteként keresett célértékként, és a modell az általa adott előrejelzésnek e célértékhez viszonyított hibája alapján tanul kategorikus keresztentrópia veszteségfüggvénnyel. A rétegek jellemzői rendre a következők:

Karakterkódoló. A megszokottól eltérően nem véletlenszerűen inicializált és a modell betanítása során tanult karakterbeágyazást használtunk a bemenetben szereplő karakterek reprezentációjához, sem one-hot kódolást, hanem egy olyan ritka vektort, amely figyelembe veszi a magyar helyesírás és morfológia bizonyos relevánsnak feltételezhető sajátságait. A nyers karaktereket a következő bináris jegyek alapján jellemezzük: ±*betű*, ±*nagybetű*, ±*hosszú* (magánhangzó), ±*mellékjeles* (idegen betű), ±*szóköz*, ±*kötőjel*, ±*számjegy*. Ezt követően minden karaktert normalizálunk: kisbetűvé alakítjuk, a hosszú magyar magánhangzókat a rövid megfelelőjükké alakítjuk, az idegen mellékjeles betűkről eltávolítjuk a mellékjelet (pl. *ä*>*a*). Ezt a normalizálást persze a kimenetben már nem végezzük el, tehát a kimenetben az ékezetes és mellékjeles betűket helyesen prediktáljuk.

Ennek az inputreprezentációnak a motivációja, hogy a környező karakterek prediktálása szempontjából sokszor lényegtelen például, hogy egy magánhangzó rövid vagy hosszú változata jelenik meg éppen. Ennek oka lehet a tőtípusokban előforduló allofónia (*órakor* vs *órák*), az egyes rövid–hosszú párok (*i–í, u–ú, ü–ű*) gyakori tévesztése, vagy az idegen szavak helytelen írása (*Dalí* helyett *Dali*). Ugyanígy sokszor az sem lényeges, hogy egy betűnek a kis vagy a nagy változata (*Budapest* vs *budapesti*, mondatkezdő nagybetűk), vagy hogy pontosan melyik számjegy jelenik meg az adott ponton (*január 8.*). A betűk normalizálásának és jegyekre bontásának az a célja, hogy ezzel segítsük a hálózatot a releváns mintázatok megtanulásában, ami különösen a ritka és gyakran elrontott mellékjeles betűk szempontjából lesz várhatóan különösen hasznos. Amennyiben egy normalizált karakter gyakorisága a tanítókorpuszban nem ért el egy előre meghatározott küszöbértéket, a nyers karaktert a speciális OOV karakterként normalizáltuk. A 83 dimenziós karaktervektorok a 72-féle gyakori karakter, az OOV karakter, a három vezérlőkarakter (szöveg eleje, vége, maszkolás) one-hot kódolásából és a fenti bináris jegyeknek megfelelő plusz 7 dimenzióból épülnek fel.

Próbaképpen a kézi kódolás helyett szokványos, véletlenszerűen inicializált és a nyelvmodellel együtt tanult sűrű karaktervektorokkal is próbálkoztunk. Azt tapasztaltuk, hogy a megkülönböztetendő karakterek számánál (112) jelentősen alacsonyabb (16 vagy 32) dimenzionalitású ilyen beágyazásvektorokra építve markánsan kevésbé sikeresen prediktált a két egyébként azonos szerkezetű modell, és a karakterek számával közel azonos méretű, 64 dimenziós beágyazásvektorral sem teljesített a bináris vektorhoz képest jobban. Így az utóbbiak használata mellett döntöttünk, de az általunk elérhetővé tett alkalmazás igény szerint konfigurálható úgy is, hogy beágyazásvektorokkal tanuljon egy új modellt.

Bemeneti réteg. Két részből áll: a jobb és a bal kontextus karakterkódjaiból. A modellünkben bal és jobb kontextusként egyaránt 30–30 karaktert használtunk. Kísérleteinkben azt tapasztaltuk, hogy a 15 karakteres kontextus következetesen enyhén gyengébb predikciókat eredményezett, ugyanakkor a kontextus

50 karakterre növelésével a perplexitás egyáltalán nem javult tovább. Úgy tűnik tehát, hogy az ennyire tág kontextus vizsgálatával a modell nagyon ritkán jut a középső karakter predikcióját segítő információhoz, míg a tág kontextusnak a rekurrens rétegben kiszámolt reprezentációba beépülő zaja megnehezíti a hasznos mintázatok megtalálását a középső karakterhez közelebbi környezetben.

Kétirányú LSTM-réteg. Szintén két részből áll: az egyik LSTM-réteg a bal kontextuson balról jobbra halad végig, míg a másik a jobb kontextuson jobbról balra. Az input feldolgozása megáll a prediktálandó karakter előtti, illetve utáni karakternél, vagyis a prediktálandó középső karaktert értelemszerűen nem építi be a reprezentációvektorba. A két LSTM-réteg egymástól független, egymással nem kommunikálnak, és egymás inputját sem látják, kimenetük egy-egy 1280 dimenziós vektor. Kísérleteztünk kisebb LSTM-rétegekkel is, de Karpathy és mtsai (2015, 1. ábra) eredményeivel összhangban azt láttuk, hogy a prediktálóképesség nő a rekurrens réteg dimenzionalitásának növelésével.

Megjegyzendő, hogy az általunk választott kétirányú LSTM-architektúra eltér a szokásosan választott implementációktól. Kétirányú RNN-eket többnyire kétféle módon szoktak gépi tanulási modellekbe beépíteni. Szövegosztályozás vagy szekvenciagenerálás esetén szokásos alkalmazási mód, hogy a két rekurrens réteg ugyanazt a hosszabb szekvenciát dolgozza fel (balról jobbra, illetve jobbról balra), így a két RNN együttesen jól képes reprezentálni a szekvencia egészét. Elemenkénti címkézés esetén ellenben mind az előre, mind a hátra haladó RNN-réteg karakterenként generál egy-egy vektorreprezentációt oly módon, hogy az egyik az adott elem bal, míg a másik a jobb kontextusáról szóló információkat összegzi, beleértve az adott elemet is. Mivel esetünkben ezektől eltérő, autoregressziós feladatot kellett megoldanunk, ahol lényeges, hogy a prediktálandó karaktert magát nem olvashatjuk be, célszerűnek láttuk egy olyan hálózat felépítését, amely az említett módon karakterről karakterre halad, az adott karakternek csak egy szűkebb környezetét vizsgálja, és a két rekurrens réteg a középső karakter beolvasása nélkül középen „találkozva” prediktálja azt.

Sűrűn kapcsolt réteg. Bemenete a bal és a jobb LSTM-réteg végső állapotának konkatenációja, azaz egy $1280 + 1280 = 2560$ dimenziós vektor. Maga a sűrűn kapcsolt réteg 1024 neuront tartalmaz. Kísérleteztünk ettől eltérő mérettel is: ha növeltük a neuronok számát, nem tapasztaltuk a predikciós képesség további javulását, míg ha csökkentettük, akkor romlott.

Dropout. Ismert, hogy a dropout révén történő regularizáció növeli a mély neurális modellek predikciós képességét azáltal, hogy a modellt a hasznos általános mintázatok megtanulására kényszeríti a tanítóadatok „bemagolása” helyett. Az elsőként kipróbált 0,5-ös dropouttal a modell predikciós képessége a validációs adatokon mérve messze elmaradt a dropout nélküli azonos méretű modellétől. Ugyanakkor a dropout teljes elhagyása esetén a tanítás során túlillesztés lépett fel, az elszálló gradiensek hatására a részben betanult hálózat ugrásszerűen és helyrehozhatatlanul elromlott. Végül legsikeresebbnek a kezdetben 0,1-es (majd a második epochtól 0,15-re növelt) dropout bizonyult, ez ugyanis egyfelől elegendő volt a gradiensek elromlásának megelőzéséhez, másfelől pedig a perplexitás lehető legkisebb mértékű romlásával járt.

Kimeneti réteg. Softmax aktivációs sűrűn kapcsolt réteg. A 113 dimenzió mindegyike egy-egy one-hot kódolt prediktálandó középső karakternek felel meg: egy érték a ritka karaktereket magában foglaló OOV kategória, a többi egy-egy konkrét betűt vagy írásjelet kódol. A (természetesen normalizálatlan) outputkarakterek számát a bemeneti normalizált karakterek számához hasonlóan kaptuk: a 112-féle gyakori karakter az OOV karakterrel együtt adja ki a dimenziószámot.

A fentieket összegezve: miután a megszokott rácskereséses eljárással megvizsgáltuk a szóba jöhető modellkonfigurációk és hiperparaméterek széles körét, azon modell teljes betanítása és alkalmazása mellett döntöttünk a továbbiakban, amely egy kiválasztott, mintegy 1 millió szó terjedelmű, a tanítókorpusszal azonos műfajú és témájú (azaz magyar sajtónyelvi) validáló korpuszon a legjobb predikciós teljesítményt nyújtotta a legjobb feldolgozási idő mellett. Az így meghatározott modell az 1. ábrán látható. Az általunk kézzel meghatározott bináris jegyes karakterkódolást alkalmazza, a prediktálandó karakter 30–30 karakternyi kontextusát veszi tekintetbe balra és jobbra. Az LSTM rétegek 1280 cellát tartalmaznak, ezek konkatenációja szolgál bemenetként egy 1024 neuronból álló sűrűn kapcsolt rétegnek, amely eleinte 0,1, majd 0,15 arányú dropouttal tanul. Néhány azonos felépítésű LSTM-modellünk perplexitásértéke az 1. táblázatban látható.

1. táblázat. Néhány modell perplexitása különböző hiperparaméterek mellett.

LSTM	sűrű	dropout	epoch	perplexitás	pontosság
512	512	0,5	2	1,182	94,96
512	512	0,1	3	1,112	96,54
1280	1024	0,1→0,15	3	1,081	97,31

3. Módszer: alternatívák közötti döntés egy nyelvmodell által prediktált perplexitások alapján

Módszerünk lényege, hogy közvetlenül használja fel minden nyelvmodell tulajdonképpeni célját, raison d'être-ét: az arra való képességüket, hogy megadják egy szekvencia ismeretében a szekvencia lehetséges következő elemeinek a valószínűségi eloszlását. A nyelvmodellnek ezt a funkcióját a talán megszokottabb használati móddal ellentétben nem szekvenciák generálásában, hanem felismerésében hasznosítjuk. A nyelvmodell kimenetének „közvetlen" felhasználásán azt értjük, hogy nem építünk rá transzfertanulás jellegű eljárással összetettebb gépi tanulási modellt, valamint nem közlünk a nyelvmodellel semmilyen azzal kapcsolatos információt, hogy mire akarjuk használni a kimenetét, hanem úgymond tökéletesen „zero–shot" elven generáltatunk vele predikciókat.

Ez utóbbi az elképzelhető legegyszerűbb módon történik: a karakteralapú nyelvmodellünkkel karakterről karakterre haladva feldolgoztatunk egy karaktersorozatot úgy, hogy az lépésenként a bal és jobb 30–30 karakter alapján előrejelezze a pillanatnyi középső karakterpozíció valószínűségi eloszlását, majd ezt összevetjük a ténylegesen a karaktersorozatban az adott helyen szereplő karakterrel. E predikciókból kiszámoljuk az adott karaktersorozat egészének átlagos

karakterenkénti valószínűségét, illetve pontosabban (ami ezzel egyenértékű) a karaktersorozatnak a nyelvmodell által becsült perplexitását. A modell által adott válasz mindenekelőtt arra használható fel, hogy kiválasszuk a hibátlant vagy legalábbis a legkevésbé hibásat egyazon szövegrészlet több különböző változata közül. Ezeket az alternatívákat generálhatjuk mi magunk, amire a 5. részben mutatunk konkrét példát – hasonló jellegű alkalmazási módok az ékezetek pótlása, nem tapadó és rosszul formázott idézőjelek, gondolat- és kötőjelek visszatapasztása, valamint véletlenszerűen beszúródott szóközök által széttördelt szavak töredezettségmentesítése. Elképzelhető viszont az is, hogy a feldolgozandó szövegváltozatok eleve adottak, például ugyanazon nyomtatott dokumentum alapján több különböző OCR-motor által generált kimenetek. Lényeges szempont, hogy mivel a nyelvmodellünk karakterszinten értékeli ki a megadott változatokat, nem két vagy több teljes mondat vagy akár szöveg között kell választanunk, hanem ezek megfelelő illesztését feltételezve képesek vagyunk akár karakterenként válogatva a különböző inputokból összerakni a helyes vagy legalábbis a rendelkezésre álló leghelyesebb outputot. Mivel zajt jól tűrő neurális nyelvmodellt használunk, az többnyire olyan helyzetekben is képes használható választ adni, amikor nemcsak a javítandó elem hibás, hanem annak kontextusa is zajos.

Alább, az 5. fejezetben részletesen ismertetjük módszerünk alkalmazását és kiértékeljük predikcióinak minőségét egy esettanulmány kapcsán. Emellett itt röviden bemutatunk egy-két példa alapján több más olyan alkalmazási lehetőséget, amelyek tapasztalataink szerint könnyen implementálhatóak és jó eredményeket adnak. A modellek építéséhez és kiértékeléséhez szükséges kód, maguk a modellek és egyéb kapcsolódó információk elérhetők a https://github.com/ril-lexknowrep/character_language_models repozitóriumban.

3.1. Erősen hibás szövegrészek szűrése

Egy nyelvmodell értelemszerűen akkor fogad el egy neki átadott szekvenciát, azaz abban az esetben tulajdonít neki alacsony perplexitást, ha az az adott modell által reprezentált nyelven íródott. Konkrétabban a karakteralapú nyelvmodellünk lényegében bármely magyarul íródott helyes mondatnak 1 és 2 közötti, inkább 1-hez közeli perplexitást tulajdonít. Ettől drasztikusan eltér az erősen hibás vagy tisztán OCR-szemétből álló szekvenciák prediktált perplexitása, például az alábbié 15,3:

```
Nílusi hajóul közvetlen chalerjarultal.
5* m luxuslupóval: 129 900 Ftíőtől
```

Egy extrémebb, 2900 körüli perplexitású példa:

```
A a L ét • a  ffc Mkm bML A hWiWWHK1**(re a nMtjjww áHő
```

Könnyen kiszűrhetjük tehát az erőteljesen zajos mondatokat vagy szakaszokat azáltal, hogy meghatározunk empirikusan egy perplexitási küszöbértéket, amely nagy biztonsággal arra utal, hogy az adott szekvencia erősen hibás, és minden ilyen mondatot vagy szakaszt törlünk a korpuszból. Gondolhatnánk, hogy ez

a feladat a Pythonban rendelkezésre álló nyelvfelismerő könyvtárakkal könnyen megoldható, azonban ez nincs így. Kísérleteinkben például a `langid` sok esetben „magyarabbnak” értékelte a jelentősen zajos változatot, mint a hibátlant.

3.2. Idegen nyelvű szövegrészek szűrése

A karakteralapú nyelvmodellek egyik klasszikus alkalmazási területe a nyelvfelismerés (vö. 4. szakasz): ha egy mondatnak egy magyar nyelvmodell jelentősen alacsonyabb perplexitást tulajdonít, mint egy angol nyelvmodell, akkor biztosak lehetünk benne, hogy a mondat nem angolul van; és ha azt is biztosan tudjuk, hogy csak a két nyelv jöhet szóba, akkor biztosan magyarul van. Ugyan ez az előbbiekből nem következik, és nem is magától értetődő, de történetesen nem szükséges több nyelvmodell ahhoz, hogy idegen nyelvű részeket kiszűrjünk egy magyar nyelvű korpuszból, hanem elég az egyetlen magyar nyelvmodell is. Míg, mint említettük, a magyar mondatok perplexitása tipikusan az 1–2, nagyon szokatlan esetekben a 3–4 tartományban mozog, addig az idegen nyelvű szövegeké rendszerint 10 fölötti, így az utóbbiak ily módon egyértelműen elkülöníthetőek az előbbiektől. Az idegen nyelvű részek így gyakorlatilag „zajként” sorolhatók be és törölhetőek a rontott szövegekkel együtt, ugyanúgy egy perplexitási küszöbértéket mint szűrési feltételt használva.

3.3. OCR-javítás

A perplexitás alapján történő választás módszerének különösen ígéretes felhasználási területe az OCR-hibák javítása. Ha rendelkezésünkre áll egyazon digitalizált dokumentum több különböző OCR-motorral szöveggé alakított változata, ezek a nyelvmodellre támaszkodva összevethetőek, és a legalacsonyabb perplexitású változat tekinthető helyesnek. Egy tipikus példa:

```
V1: a község egyik hivatalnokát háza elõtt agyonszarták.
V2: a község egyik hivatalnokát háza elõtt agyonszurták.
V3: a község egyik hivatalnokát háza elõtt agyonszúrták.
```

A V1 változatot az ABBYY Finereader 12, a V2-t a FR 15 generálta (az utóbbi a digitalizált dokumentumban szereplő régies írásmódú alakot tartalmazza), míg a V3 általunk kézzel javított változat. Nyelvmodellünk az utolsó szavakra a következő összesített perplexitást prediktálja a fenti (egyébként azonos) kontextusokban: *agyonszarták* 10,16, *agyonszurták* 6,45, valamint *agyonszúrták* 1,72.

A nyelvmodell igen magabiztosan választja ki a helyes változatot a három lehetőség közül. Emellett figyelemre méltó, hogy a tartalmilag helyes, csupán az ékezetet nélkülöző *agyonszurták* sokkal jobb eredményt mutat, mint a helyesen írt, de tartalmilag értelmetlen *agyonszarták*. Vegyük észre, hogy a nyelvmodell a kontextusban jelen lévő zaj ellenére (a Finereader outputjában tildés *õ* szerepel *ő* helyett) képes meghozni a helyes döntést. További viszonylag kiterjedt kísérleteket is folytattunk e módszerrel végzett OCR-javítással, és azt tapasztaltuk, hogy az eljárás rendkívül sikeresen javít olyan típusú hibákat, amelyek egy szótáralapú helyesírás-ellenőrzőnek nehézséget okoznának, így különösen elrontott központozási jeleket (pl. pont helyett vessző).

4. A módszer szakirodalmi előzményei

A szakirodalmat áttekintve úgy látjuk, hogy az általunk követett egyszerű eljáráshoz hasonló megközelítést ritkán alkalmaznak; vagy ha mégis, akkor egyszerűsége miatt nem számolnak be róla. Az egyetlen általunk ismert terület, ahol jól dokumentálva van a karakteralapú nyelvmodellek (ezen belül mindenekelőtt az n-gram-modellek) által kiadott valószínűségi, illetve perplexitási predikciók közvetlen felhasználása, a nyelvfelismerés.

Amint korábban utaltunk rá, ennek az az alapötlete, hogy ugyanazon szöveg valószínűségét/perplexitását több különböző nyelv modelljével prediktáltatjuk, és amelyik modell a legvalószínűbbnek tartja az adott szöveget, azt azonosítjuk annak nyelveként. Ez a módszer az általunk itt javasolt módszernek pont a fordítottja: míg itt egyazon szöveg különböző változatainak a valószínűségére adunk becslést egyetlen modellt használva, és így választunk a szövegváltozatok közül, addig a nyelvfelismerés kapcsán ugyanazon szöveg valószínűségére több különböző nyelvmodell alapján adnak becslést, és így választanak a nyelvmodellek közül. Az n-gram-modellek ilyen használatáról szóló klasszikus szakirodalom Dunning (1994). A téma egy későbbi releváns feldolgozása Řehůřek és Kolkus (2009), amely jó áttekintést ad az egyéb kapcsolódó irodalomról is. Az itt javasolt megoldáshoz legközelebb Pethő és Mózes (2014) áll, amely viszont hagyományos Markov-modellt használ nyelvfelismerésre.

Karakteralapú megközelítésre az ékezetek és egyéb mellékjelek pótlása témakörében is találunk példát. Mihalcea és Nastase (2002) a prediktálandó karakter bal és jobb 5 karakter széles kontextusában elhelyezkedő karakterekre mint jegyekre épít döntési fát, illetve esetalapú tanulót. Ennek a megoldásnak nyilvánvaló hátulütője a nyelvmodell használatával szemben, hogy a betanított modell kizárólag azt a feladatot képes megoldani, amelyet megtanult.

5. Esettanulmány: visszaválasztás

5.1. Célkitűzés

A választott feladatunk a következő: sorvégi elválasztást tartalmazó anyagban előállítani az eredeti elválasztatlan szöveget. Amint a bevezetőben említettük, ez azért nem triviális, mert a kötőjel+sortörés kombinációt nem szabad egyszerűen törölni, hanem négy különböző esetet kell kezelni. Ezekre a következő kódokkal hivatkozunk: `{1}` egyszerű törlés; `{2}` törlés digráfkezeléssel; `{3}` a kötőjel megtartása; `{4}` a kötőjel megtartása és a sortörés szóközre cserélése. A 3. részben leírt módszert úgy alkalmazzuk, hogy legeneráljuk a fenti négy lehetséges alternatívát, és azt fogadjuk el jónak, amelyikre a legkisebb perplexitást kapjuk.

5.2. Anyagok

Mind az n-gram-, mind az LSTM-modellt, mind pedig a szabályalapú visszaválasztó eszközünk esetalapú komponensét azonos szövegkorpuszon tanítottuk

be, amely az MNSZ2 (Oravecz és mtsai, 2014) sajtónyelvi részén alapul. Ezt a részkorpuszt reguláris kifejezésekkel javítottuk és szűrtük, és ezáltal elhárítottunk benne különböző karakterkódolási hibákat és egyéb anomáliákat (pl. nem tapadó írásjelek). A tanítókorpusz kiválasztása során döntő szempont volt, hogy ne tartalmazzon optikai karakterfelismeréssel nyert, valamint eredetileg elválasztott formában tárolt szövegeket, tehát minőségi szempontból a lehető legtisztább legyen; sajtókorpusz legyen, miután magunk is sajtónyelvi korpusz tisztítására kívánjuk használni; valamint elég nagy méretű legyen. Tanítókorpuszunk terjedelme mintegy 250 millió szövegszó, illetve közel 2 milliárd karakter. E korpusz a 90-es évek végétől a 2010-es évek elejéig terjedő szövegeket tartalmaz, és 11 online és nyomtatott sajtótermék (egyebek mellett az Origo, a 168 óra, a HVG, a Népszabadság, a Népszava és a ComputerWorld) anyagaiból áll össze.

5.3. Kiértékelés

A neurális rekurrens modellel történő visszaválasztás kiértékelése céljából annak sikerességét egy nagyon egyszerű baseline mellett egy kombinált szabályalapú–esetalapú elven működő, szintén általunk írt visszaválasztó alkalmazással vetettük össze, valamint az általunk javasolt, perplexitások összehasonlításán alapuló módszert úgy is vizsgáltuk, hogy a kétirányú LSTM-modell helyett kétirányú n-gram-modellel számíttattuk a perplexitásokat.

A szabályalapú–esetalapú alkalmazásunk dióhéjban összefoglalva azt vizsgálva hoz döntést az inputszövegében szereplő kötőjelek feloldásának miként jéről, hogy a sor végén álló kötőjel két oldalán álló egy-egy szó a magyar `hunspell` szerint egybeírva egy szót alkot-e (ez az alkalmazás szabályalapú része), valamint hogy a feldolgozott tanítókorpuszban a két szó előfordult-e egybeírva, illetve kötőjellel írva, és amennyiben igen, e két írásmód egymáshoz képest mennyire volt gyakori (ez az esetalapú komponens). Ezeket és további szempontokat összekapcsolva az alkalmazás heurisztikusan egyértelműsíti a két szó közötti kötőjeleket.

2. táblázat. Az n-gram-modellek perplexitásértékei.

n	perplexitás	pontosság	memória
5	1,707	83,28	7 GB
6	1,495	87,52	25 GB
7	1,424	89,37	70 GB
8	1,404	90,17	180 GB

A neurális rekurrens modell alternatívájaként egy alapvetően hagyományos, bár kétirányú n-gram-modellt is kipróbáltunk. Különböző rendű, magunk által implementált n-gram-modellekkel próbálkoztunk, ezek rendje 5-től 8-ig terjedt. A kétirányú n-gram-modell úgy állt össze, hogy két egyszerű egyirányú, szükség esetén eggyel alacsonyabb rendű modellre visszalépő n-gram-modellel prediktáltuk a középső karakter valószínűségi eloszlását úgy, hogy az egyik modell balról jobbra, a másik jobbról haladt a középső karakter felé. A két n-gram-modell által prediktált valószínűségi eloszlásvektort elemenként összeszoroztuk, majd

a szorzatokat normalizáltuk 1-re, és ennek eredményét tekintettük a kétirányú n-gram-modell által előrejelzett valószínűségi eloszlásnak.

A különböző méretű n-gram-modelljeinkre a 2. táblázatban lévő perplexitásértékeket mértük a validáló korpuszon. A kiértékeléshez a 7-gram-modellt használtuk a hasonló pontosságú, de 250 millió szós korpuszon tanítva kezelhetetlenül nagy memóriaigényű 8-gram-modell helyett. Összehasonlításképpen (ld. 1. táblázat) a kétirányú LSTM-modelleknek azonos korpuszon mért perplexitása ehhez képest jelentősen jobb volt, és memóriaigénye is elhanyagolható.

A kiértékelés a következőképpen történt. Tetszőleges sima szövegből kiindulva a szöveget szabályos automatikus elválasztással 40 karakter szélességű sorokra tördeltük (a `pyphen` csomag segítségével), szimulálva egy elválasztásokat tartalmazó nyomtatott mű felépítését. Minden sor végén a megfelelő kóddal megjelöltük, hogy az adott ponton az eredetiben melyik elválasztási eset szerepelt, azaz hogy mi lesz a helyes visszaválasztás. A kiértékelés során csak a sorok végére eső kötőjelekkel foglalkoztunk, a sorok belsejében lévőkkel nem. Az így annotált gold fájlokból az osztálykódokat leválasztva kaptuk a inputfájlokat, ezeken futtattuk a visszaválasztó algoritmusokat, melyeknek a feladata a kód prediktálása volt. A kiértékelési mérőszámok a két kódsor összevetésével egyszerűen adódtak.

Négy konfigurációt vetettünk össze.

1. **triviális** baseline: ez egyszerűen egyszerűen törli az az összes kötőjel+sortörés kombinációt, `{1}`-es kódot oszt ki minden sorvégi kötőjelnek, azaz minden esetben úgy tekinti, mintha egyszerű elválasztást kellene visszaalakítani
2. **szabály-** és esetalapú módszer (ld. fentebb)
3. perplexitásalapú döntés kétirányú **7-gram** nyelvmodellel (ld. fentebb)
4. perplexitásalapú döntés kétuirányú **LSTM** nyelvmodellel (ld. 2. és 3. rész)

A kiértékelést egy 100 000 sort tartalmazó korpuszon futtattuk le, mely 30 592 sorvégi elválasztást tartalmaz. Ennél a méretnél a legritkább kategóriára is van nagyjából 100 előfordulás. Ennek a korpusznak a felén kiértékelve gyakorlatilag ugyanazokat a számokat kapjuk, így nem érdemes tovább növelni a kiértékelő korpusz méretét.

5.4. Eredmények

A 3. táblázatban látható eredményeink azt mutatják, hogy a szabályalapú és a perplexitásalapú eljárások hasonlóan jó eredményt adnak: a probléma természetéből adódó nagyon magas triviális baseline fölött teljesítenek, a hibák számát 40-60%-kal csökkentik. A legnehezebb, `{4}`-es kategóriában a szabályalapú módszer kifejezetten rosszul teljesít, a két nehéz kategóriában az LSTM-modell adja a legjobb eredményt. Érdekes, hogy a szóban forgó alkalmazás tekintetében az n-gram modell kicsivel jobbnak tűnik az LSTM-nél. Látjuk azt is, hogy a kategóriák sorra egyre nehezebb feladatot jelentenek a módszereknek.

A validálókorpusz hibáit elemezve azt látjuk, hogy azon túl, hogy nagyon ritkán elrontja a digráf kezelését (‘*hoszszú*’), csak a tényleg nehéz esetekben téved az LSTM-et használó eljárás. Helyesen kezeli például a ‘*lopottautó- vagy fegyverkereskedelemből*’ és a ‘*harmincasok-negyvenesek*’ kifejezést. Előfordul, hogy

lenyeli a kötőjelet (‘*védelmipénzszedés*’ a helyes ‘*védelmipénz-szedés*’ helyett vagy ‘*diszkódrogfogyasztótáborral*’ a helyes ‘*diszkódrogfogyasztó-táborral*’ helyett), vagy meghagyja a szóközt (‘*ex- Ybl-bankos*’ az ‘*ex-Ybl-bankos*’ helyett). A szintén helytelen eredményt jelentő ‘*Gelá-ban*’ azért is érdekes, mert az effajta ritka és idegen földrajzi nevek toldalékolt alakjait az anyanyelvi nyelvhasználó is kötőjellel írja néha, hiába kell szabály szerint egybeírni.

3. táblázat. A négy konfiguráció kiértékelése.

		pontosság	fedés	F_1	accuracy
triviális	{1}	0,981	**1,000**	0,991	
	súlyozott átlag	0,963	0,981	0,972	
					0,981
szabály	{1}	**0,999**	0,994	0,997	
	{2}	1,000	**0,989**	**0,994**	
	{3}	0,718	0,654	0,685	
	{4}	0,125	0,163	0,142	
	súlyozott átlag	**0,994**	0,988	0,991	
					0,988
7-gram	{1}	0,997	0,998	**0,998**	
	{2}	1,000	**0,989**	**0,994**	
	{3}	**0,774**	0,692	0,731	
	{4}	0,316	**0,378**	0,344	
	súlyozott átlag	0,993	**0,993**	**0,993**	
					0,993
LSTM	{1}	0,996	0,997	0,997	
	{2}	1,000	0,749	0,856	
	{3}	0,697	**0,825**	**0,755**	
	{4}	**0,579**	0,337	**0,426**	
	súlyozott átlag	0,992	0,992	0,992	
					0,992

Köszönetnyilvánítás. A modellek tanítását az ELKH Cloud infrastruktúrán futtattuk. Köszönetet mondunk az ELKH Cloud (lásd: Héder és mtsai, 2022; https://science-cloud.hu/) használatáért, ami hozzájárult a publikált eredmények eléréséhez.

6. Konklúzió

Cikkünkben egy speciális karakteralapú kétirányú LSTM architektúrát mutatunk be, amely autoregresszió révén egy kérdéses karakter(szekvencia) perplexitását adja meg, annak bal és jobb oldali kontextusa alapján. Egy ilyen modell segítségével egy egyszerű módszert alkalmazva arról tudunk dönteni, hogy egy szöveg bizonyos változatai közül melyik a hibátlan (legkevésbé hibás) verzió. Ezt a módszert alkalmasnak tartjuk számos korpusztisztító lépés megvalósítására, ezt példákon mutattuk be. Részletesen a visszaválasztás problémájára értékeltük ki a módszert, azaz elválasztásokat tartalmazó szövegben állítottuk helyre az eredeti szöveget a kötőjelek, szóközök és szükség esetén a digráfok helyes kezelésével. Baseline-nal összevetve azt kaptuk, hogy ez a viszonylag kevés erőforrást igénylő neurális módszer jó teljesítményt nyújt a nehéz esetek kezelésében is.

Hivatkozások

Aston, G., Burnard, L.: The BNC Handbook: Exploring the British National Corpus with SARA. Edinburgh University Press, Edinburgh (1998)

Cheriet, M.E.A., Kharma, N.N., Liu, C.L., Suen, C.Y.: Character Recognition Systems: A Guide for Students and Practitioners. Wiley-Interscience, 605 Third Avenue New York, NY United States (2007)

Csengery, K., Lipp, V., Simon, L.: A magyar történeti szövegtár bővítése. korhatáremelés 2010-re. In: Ittzés, N.e. (szerk.) A magyar nyelv nagyszótára. VI. kötet. Di–ek., pp. 926–927. MTA Nyelvtudományi Intézet, Budapest (2016)

Dunning, T.: Statistical identification of language. Technical Report MCCS. New Mexico State University (1994)

Goldberg, Y.: Neural Network Methods for Natural Language Processing. No. 37 in Synthesis Lectures on Human Language Technologies, Morgan & Claypool (2017)

Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org

Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)

Héder, M., Rigó, E., Medgyesi, D., Lovas, R., Tenczer, S., Török, F., Farkas, A., Emődi, M., Kadlecsik, J., Mező, G., Pintér, A., Kacsuk, P.: The past, present and future of the ELKH Cloud. Információs Társadalom 22(2), 128–137 (2022)

Jurafsky, D., Martin, J.H.: Speech and Language Processing, 3. kiadás. Kézirat. Stanford University, Stanford (2021), https://web.stanford.edu/~jurafsky/slp3/

Kamath, U., Liu, J., Whitaker, J.: Deep Learning for NLP and title = variable-length and variable-language texts variable-length and variable-language texts, Speech Recognition. Springer, Cham (2019)

Karpathy, A., Johnson, J., Li, F.: Visualizing and understanding recurrent networks. CoRR abs/1506.02078 (2015), http://arxiv.org/abs/1506.02078

Laki, L.J., Kőrös, Á., Ligeti-Nagy, N., Nyéki, B., Vadász, N., Yang, Z.G., Váradi, T.: OCR-hibák javítása neurális technológiák segítségével. In: XVIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2022). pp. 417–430. Szegedi Tudományegyetem, Szeged (2022)

Mihalcea, R., Nastase, V.: Letter level learning for language independent diacritics restoration. In: COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002) (2002), https://aclanthology.org/W02-2021

Mori, S., Nishida, H., Yamada, H.: Optical character recognition. J. Wiley, NY United States (1999)

Oravecz, Cs., Váradi, T., Sass, B.: The Hungarian Gigaword Corpus. In: Calzolari, N., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., Piperidis, S. (szerk.) Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC 2014). pp. 1719–1723. European Language Resources Association (ELRA), Reykjavik, Iceland (2014)

Pethő, G., Mózes, E.: An n-gram-based language identification algorithm for variable-length and variable-language texts. Argumentum 10, 56–82 (2014)
Simon, E., Sass, B.: Nyelvtechnológia és kulturális örökség, avagy korpuszépítés ómagyar kódexekből. Általános Nyelvészeti Tanulmányok 24, 243–264 (2012)
Řehůřek, R., Kolkus, M.: Language identification on the web: Extending the dictionary method. In: Proceedings of the 10th International Conference on Intelligent Text Processing and Computational Linguistics. pp. 357–368 (2009)