Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Deep Learning alapú szótagolás

Önnáló laboratórium

Készítette
Németh Gergely Dániel

Konzulens
Ács Judit

2017. május 12.

Abstract

Kivonat

Tartalomjegyzék

1. Bevezetés

2. Szótagoló programok

Az elterjedt szótagoló algoritmusok kétféle csoportba bonthatóak: szabály- vagy szótáralapúak. A szabad szoftverek világában *de facto* a TEXszótagolási algoritmusát használják.

A TEX eredeti szótagoló algoritmusát Prof. Knuth tervezte 1977 nyarán [3]. Ez három fő szótagolási szabályt alkalmazott: 1) utótag leválasztás, 2) előtag leválasztás és 3) magánhangzó - mássalhangzó - mássalhangzó - magánhangzó (vccv) elválasztás, azaz ha ilyen betűnégyes található a szóban, legtöbb esetben a mássalhangzók mentén elválasztható. Ez a három szabály gyakran alkalmazható, azonban már az első algoritmusban kiegészült kisebb szabályokkal („break vowel-q" vagy „break after ck") és kivételek listájával(300 szó).

A TEX82 verzióhoz megjelent Liang szótagoló algoritmusa [5], aminek legfontosabb újítása a minta(*patterns*) alapú szótagolás bevezetése volt. Ennek lényege, hogy a szótagolási szabályok mintákra definiálódtak és az algoritmus a szótagolás során ezeket a mintákat keresi a szóban.

A TEX jelenlegi verziójában a Hunspell szótagoló algoritmust alkalmazzák [6]. Ez az eljárás Liang algoritmusán alapszik, amit nyelvfüggő speciális szótagolási kiterjesztésekkel (*non-standard hyphenation extension*) egészít ki.

2.1. Liang algoritmusa

Liang szótagoló algoritmusának alapját a szótagolási minták alkotják [5]. Az algoritmust az angol `hyphenation` elválasztását az alábbi módon végzi:

Az algoritmus először megnézi, hogy a szó benne van-e a kivételek listájában (ez lényegében teljes szavakat tartalmaz mintaként). A *hyphenation* szó nincs a kivételek listájában.

Ezután a szó elejére és végére illeszt egy pont jelző karaktert. Ennek jelentősége azoknál a mintáknál lesz, amelyek akkor érvényesülnek, ha a szó elején, vagy a végén szerepelnek.

```
.hyphenation.
```

Ezt követően a minták között keres illeszkedést. A *hyphenation* szóra ezek az alábbiak: `hy3ph, he2n, hena4, hen5at, 1na, n2at, 1tio, 2io` [5, 37. oldal] A megfelelő mintákat ráillesztve a szóra, a bennük szereplő számokat a szó karakterei közé szúrva az 1. ábrán szereplőket kapjuk.[1]

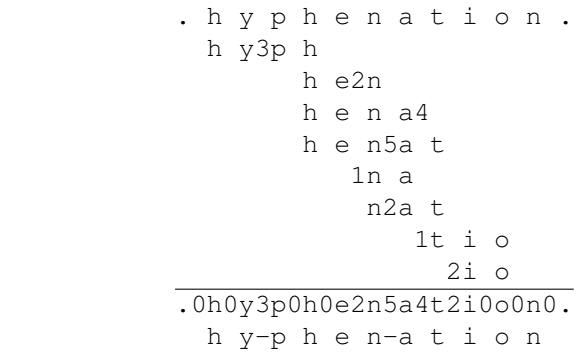
Az algoritmus következő lépésében minden két szomszédos karakter közé illesztünk egy számot. Alapértelmezetten 0-t és ha ennél nagyobb számot találtunk a mintaillesztésnél, a legnagyobb kapott értéket adjuk meg.

A szótagolás szabálya innen már csak egy lépés: a páratlan számok mentén elválasztunk, a párosoknál nem. Ezzel megkaptuk a `hy-phen-ation` elválasztást.

2.1.1. Minták választása

A fenti algoritmus hatékonyságát nyilvánvalóan a minták mennyisége és hatékonysága határozza meg. Csak azok a szavak választódnak el, amelyekre illeszkedik minta és csak azok lesznek jó elválasztások, melyekre jó minta illeszkedik.

[1] Az ábrázolásmód Németh cikkéből származik [6].



1. ábra. A *hyphenation* szótagolása

2.2. Hunspell

2.2.1. Szótagolási hibák a magyar Hunspellben

Hunspell szótagolása:	Helyesen:
au-tó-val	a-u-tó-val
szem-üveg-gel	szem-ü-veg-gel
has-izom	has-i-zom
messze	mesz-sze

3. Deep learning alapú módszerek

3.1. Deep learning bevezetés

A gépi tanulás területén egyre nagyobb teret nyernek a deep learning alapú módszerek. A hagyományos osztályozási feladatokon túl, szekvenciális, ún. taggelési problémák megoldására is alkalmasak a neurális hálók, különös tekintettel a rekurrens neurális hálókra.

A mély tanulás bevetését a nyelvtechnológia területén az a tény élteti, hogy az elektronikus kommunikáció korában jelentős mennyiségű nyelvi adatbázisok állnak rendelkezésünkre, így a deep learning legfontosabb eleme, a megfelelő méretű tanulóadat elérhető. Az ismertetett elválasztási algoritmusok kulcseleme, hogy megfelelő elválasztási mintákat ismerjünk fel az adathalmazon (*korpusz*). A képfelismerés terén sokkal komplexebb minták megtalálásában is jelentős eredményeket értek el a deep learning segítségével, így e módszer bevetése a szótagolás problémakörében megalapozott.

3.1.1. Keretrendszerek

Deep learning rendszerek tervezésénél az utóbbi időben ipari és kutatói területen egyaránt elterjedőben vannak a magas szintű keretrendszerek. Ezek legnagyobb előnye, hogy az aktív fejlesztői közösség a gyorsan fejlődő tudományág eredményeit alkalmazható technológiákként tárja a felhasználók elé. A jelentősebb keretrendszerek:

TensorFlow[2], Torch7[3], Keras[4]. Az itt ismertetett eljárások megvalósításához a Keras nyújtotta lehetőségekkel valósultak meg [1].

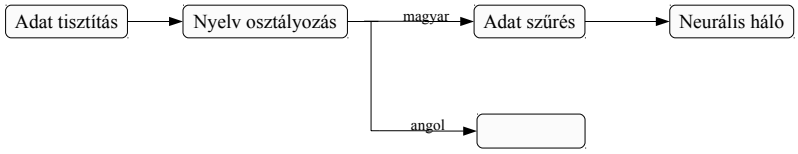
3.2. Tanító adatok

Mély tanulás esetén az eredmény minőségét alapvetően meghatározza a tanítóhalmaz minősége. Jelen esetben a korpuszok és azok megfelelő szótagoltsága.

A hálótervezés során a Magyar Webkorpusz leggyakoribb 100000 szaván kísérleteztünk [2, 4].

4. A feldolgozás folyamata

A szótagoló jelenleg magyar nyelvre van tervezve, azonban az eljárás lényege univerzális.



2. ábra. A feldolgozás folyamata

4.1. Adattisztítás

Az adatfeldolgozás során az alábbi tisztítási eljárásokat végeztem:

kisbetűsítés A karaktereket kisbetűssé tettem.

speciális karakter szűrés Az alábbi speciális karaktereket töröltem a szövegből(a *python* `string.punctuation` környezetfüggően választja meg ezeket, így máshol eltérhetnek): ! " # $ % & ’ ( ) * + , - . / : ; < = > ? @ [ ] ^ _ ‘ { | } ~

szám szűrés A szövegben található számokat töröltem.

Majd a szavakat egyedivé tettem (egy számlálóban eltárolva az előfordulási gyakoriságot).

4.2. Adat szűrés

A háló sajnos nem minden előforduló esetet tud kezelni, így a későbbi implementálásig ezeket szűröm.

[2]TensorFlow: https://www.tensorflow.org/
[3]Torch7: http://torch.ch/
[4]Keras: https://keras.io/

Speciális karakterek: az adott nyelvre nem jellemző karakterek bezavarhatnak a tanításba, így ezeket célszerű kivenni. A magyar elválasztó esetén csak a magyar ábécé betűit hagytam meg.

Kettős betűk: az osztályozó jellegéből adódóan a plusz karakter bevonásával képződő elválasztást (például a *ssz*-ből *sz-sz*) a hálón kívül dolgozom fel.

4.3. Nyelv osztályozó

A tanítások kiértékelése során arra a következtetésre jutottam, hogy a hunspell és az én elválasztóm közötti eltérések jelentős része akkor adódik, ha idegen nyelvű szót próbálunk meg elválasztani. Így az elválasztó javításának érdekében egy nyelv osztályozót is bevezettem.

A jelenlegi nyelvválasztó egy egyszerű bigram gyakoriság alapú osztályozó.

5. Előrecsatolt neurális háló

A neurális háló lényege ebben ez esetben az, hogy a szó minden karakterére eldöntsük a környezetében lévő karakterek segítségével, hogy a karakternél van-e elválasztás.

A szavakat minden karakter mentén felbontjuk az adott ablakméretre, majd ezekre az ablakokra tanítva oldunk meg egy osztályozó problémát, ami ad egy címkét a karakternek és a címke alapján végezzük az elválasztást.

5.1. Karakter címkézés

Kétféle címkézést használunk. Az első eset előnye, hogy több címke van így több információt adunk át a modellnek, a másodiké pedig, hogy a jósolt címkézés minden esetben elválasztássá alakítható. Ez a BMES esetében nem teljesül: ha két karakterre egymás után E-t jósol, az nem alakítható rögtön valós elválasztássá.

A jelenlegi munkában a BM címkézést elemeztük.

BMES

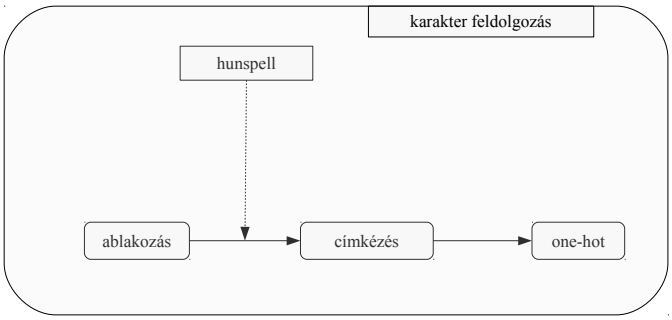
- B: Begin, szótag elején álló karakter
- M: Middle, közepén lévő karakter, se előtte, se utána nincs elválasztás
- E: End, a karakter után elválasztás van
- S: Single, egy karakterből álló szótag

A `le-o-párd` szó címkézése: `BESBMME`

BM

- B: Begin, szótag elején álló karakter
- M: Minden egyéb eset

A `le-o-párd` szó címkézése: `BMBBMMM`



3. ábra. Tanítóadat előállítása

5.2. Adat-előkészítés

Ha az *nyúlanyó* szó *a* karakterét szeretnénk egy 5-ös szimmetrikus ablakú környezetben tanítóadatnak előkészíteni, **BM** címkézéssel, akkor az alábbi előkészületeket kell tennünk:

5.2.1. Tanítás a karakter környezetéből

A jelölésrendszerünk alapján az 5-ös szimmetrikus ablak azt jelenti, hogy a karakter előtt és a karakter után is két-két további karaktert veszünk, és ez alapján az öt karakter alapján próbáljuk meg eldönteni, milyen címke illeszkedik a középső karakterre. Ezt jelöljük *2-1-2*-vel. Az *nyúlanyó* szó és az *a* betű esetén ez *úl a ny* ablakot jelent.

Ha az ablak túlnyúl a szó szélén, az elterjedt jelölés alapján a szó elején ^ karaktereket, illetve a szó végén további $ karaktereket illesztünk az ablakba.

5.2.2. One-hot kódolás

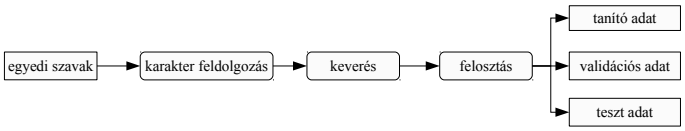
Ahhoz, hogy a karakterekből a keras által feldolgozható tanítóadat legyen, one-hot kódolást végzünk rajta.

A karakterek esetében ez azt jelenti, hogy a megengedett karaktereket, jelen esetben a magyar ábécé karaktereit és a kezdő-, befejezőkaraktereket egymás után felsoroljuk, majd az adott karaktert úgy reprezentáljuk, hogy egy ezzel azonos hosszúságú tömbben a karakternek megfelelő sorszámú helyre egyest, minden további mezőbe nullást írunk.

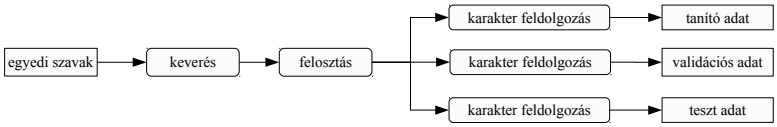
A **BM** illetve **BMES** címkézést ugyanígy reprezentáljuk egy 2 avagy 4 hosszú tömbbel.

5.2.3. Lapítás

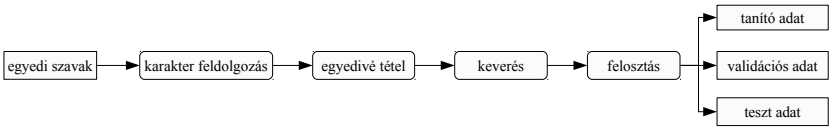
Ha a fentieket elvégezzük egy 5 hosszú ablakra, akkor a magyar ábécét figyelembe véve 5 darab $35 + 2$ hosszú tömböt kapunk. A háló bemenetét úgy kapjuk meg, hogy ezt ellapítjuk egy $5 \cdot 37$ hosszú tömbbé (amiben 5 helyen 1-es, minden további helyen 0-ás szerepel). Ez lesz a tanítóadatunk.



4. ábra. A kezdeti adatgenerálás



5. ábra. Egyedi szavak felosztása



6. ábra. Egyedi adatok

5.3. Tanító-, validációs- és tesztadatok

6. Eredmények

Az első tanítások eredményei az 1. táblázatban láthatóak.

a 2. táblázat a hiperparaméterek optimalizálási eredményeit foglalja össze. Itt a 2-1-2 környezetű karakter-címkézés 2-10 rétegű, rétegenként 10-100 neuronú tanítások közül láthatjuk azokat, amelyeknek validációs hibája megközelíti a 4 százalékot.

A 3. táblázat a karakter címkézéséhez körülötte megvizsgált karakterek számát változtattuk. Jól látható, hogy egy-egy karakter bevétele még jóval kevesebb információt árul el a címkézésről, mint a többi. Hasonlóképp észrevehető, hogy a túl nagy ablak is ront a helyzeten. Ez azért jelenik meg, mivel a nagyobb ablakok bevezetésével egyre ritkábbak lesznek az adott típusú minták előfordulása az adatokban.

6.1. A táblázatokban használt rövidítések

- Length: *a-1-b* a vizsgált karakter előtt a utána b karaktert veszünk be környezetébe, ami alapján a címkézést végezzük.
- Tag: a címkézés BM vagy BMES
- NLayer: a háló rejtett rétegeinek száma
- NHidden: a rejtett rétegek mérete
- Val_loss: a validációs adatok binary(BM) illetve categorical(BMES) crossentropy hibafüggvénnyel számolt hibája
- Test_w: maximumkiválasztás után hibás eredményt adó teszt adatok száma
- Epochs: a tanításhoz szükséges epoch-ok száma. A tanítás végét early stoping módszerrel határoztuk meg.

No	Tag	Length	Loss	Batch size	Val_loss	Epochs
1	BMES	2-1-0	cat_cross	512	0.616	483
2	BM	2-1-0	bin_cross	1024	0.262	475
3	BM	2-1-0	cat_cross	1024	0.261	543
4	BM	2-1-1	bin_cross	1024	0.063	247
5	BM	2-1-2	bin_cross	1024	0.050	213
6	BM	3-1-3	bin_cross	1024	0.047	223
7	BMES	2-1-2	cat_cross	1024	0.107	389
8	BMES	3-1-3	cat_cross	1024	0.090	769

1. táblázat. Előrecsatolt háló tanítása különböző paraméterekkel

No	Tag	Length	NLayer	NHidden	Epochs	Val_loss	Test_w
2.1	BM	2-1-2	2	10	308	0.05025	0.01568
2.2	BM	2-1-2	2	20	250	0.04454	0.01352
2.3	BM	2-1-2	2	30	189	0.04274	0.01264
2.4	BM	2-1-2	2	40	170	0.04151	0.01185
2.5	BM	2-1-2	2	50	196	0.04207	0.01129
2.6	BM	2-1-2	2	60	167	0.04068	0.01119
2.7	BM	2-1-2	2	70	166	0.04155	0.01122
2.8	BM	2-1-2	2	80	175	0.03994	0.01073
2.9	BM	2-1-2	2	90	179	0.03962	0.01050
2.10	BM	2-1-2	2	100	145	0.03911	0.01100
3.7	BM	2-1-2	3	70	176	0.04221	0.01155
3.8	BM	2-1-2	3	80	161	0.03881	0.01095
3.9	BM	2-1-2	3	90	171	0.03872	0.01037
3.10	BM	2-1-2	3	100	178	0.03886	0.01062
4.7	BM	2-1-2	4	70	208	0.04024	0.01116
4.8	BM	2-1-2	4	80	171	0.03995	0.01077
4.9	BM	2-1-2	4	90	177	0.03894	0.01077
4.10	BM	2-1-2	4	100	177	0.03885	0.01090
5.4	BM	2-1-2	5	40	199	0.04423	0.01225
5.5	BM	2-1-2	5	50	205	0.03992	0.01136
5.6	BM	2-1-2	5	60	220	0.03955	0.01090
5.7	BM	2-1-2	5	70	179	0.04013	0.01063
5.8	**BM**	**2-1-2**	**5**	**80**	**207**	**0.03937**	**0.01032**
5.9	BM	2-1-2	5	90	218	0.03995	0.01076
5.10	BM	2-1-2	5	100	186	0.03927	0.01092
6.7	BM	2-1-2	6	70	202	0.04186	0.01149
6.8	BM	2-1-2	6	80	218	0.04048	0.01076
6.9	BM	2-1-2	6	90	157	0.04715	0.01221
6.10	BM	2-1-2	6	100	217	0.03926	0.01072
7.7	BM	2-1-2	7	70	254	0.04432	0.01218
7.8	BM	2-1-2	7	80	221	0.04335	0.01169
7.9	BM	2-1-2	7	90	224	0.03943	0.01046
7.10	BM	2-1-2	7	100	210	0.04165	0.01198

2. táblázat. Hiperparaméter optimalizálás

No	Tag	Length	NLayer	NHidden	Epochs	Val_loss	Test_w
3.6	BM	1-1-1	5	60	303	0.09159	0.03617
3.7	BM	1-1-1	5	70	303	0.09156	0.03634
3.8	BM	1-1-1	5	80	229	0.09155	0.03582
3.9	BM	1-1-1	5	90	252	0.09137	0.03544
3.10	BM	1-1-1	5	100	196	0.09281	0.03595
3.11	BM	1-1-1	5	110	249	0.09129	0.03601
5.6	BM	2-1-2	5	60	222	0.04062	0.01113
5.7	BM	2-1-2	5	70	202	0.03969	0.01069
5.8	BM	2-1-2	5	80	222	0.04209	0.01099
5.9	BM	2-1-2	5	90	214	0.03992	0.01070
5.10	BM	2-1-2	5	100	198	0.03950	0.01049
5.11	BM	2-1-2	5	110	190	0.03995	0.01052
7.6	BM	3-1-3	5	60	133	0.03853	0.00988
7.7	BM	3-1-3	5	70	131	0.04250	0.01096
7.8	BM	3-1-3	5	80	147	0.04433	0.01245
7.9	BM	3-1-3	5	90	149	0.03865	0.00982
7.10	BM	3-1-3	5	100	137	0.03813	0.00972
7.11	BM	**3-1-3**	**5**	**110**	**136**	**0.03926**	**0.00952**
9.6	BM	4-1-4	5	60	121	0.03903	0.01005
9.7	BM	4-1-4	5	70	119	0.04316	0.01143
9.8	BM	4-1-4	5	80	125	0.04126	0.01104
9.9	BM	4-1-4	5	90	118	0.03934	0.01052
9.10	BM	4-1-4	5	100	116	0.03831	0.01072
9.11	BM	4-1-4	5	110	126	0.03967	0.01082
11.6	BM	5-1-5	5	60	112	0.04542	0.01183
11.7	BM	5-1-5	5	70	111	0.04082	0.01087
11.8	BM	5-1-5	5	80	119	0.04566	0.01220
11.9	BM	5-1-5	5	90	99	0.04360	0.01005
11.10	BM	5-1-5	5	100	103	0.04013	0.01035
11.11	BM	5-1-5	5	110	98	0.04117	0.01064

3. táblázat. Karakter-környezet ablak méretének optimalizálása

Hivatkozások

[1] François Chollet. Keras, 2015.

[2] Péter Halácsy, András Kornai, Nemeth Laszlo, Rung Andras, István Szakadát, and Tron Viktor. Creating open language resources for hungarian. 2004.

[3] Donald Ervin Knuth. *TEX and METAFONT: New directions in typesetting*. American Mathematical Society, 1979.

[4] András Kornai, Péter Halácsy, Viktor Nagy, Csaba Oravecz, Viktor Trón, and Dániel Varga. Web-based frequency dictionaries for medium density languages. In *Proceedings of the 2nd International Workshop on Web as Corpus*, pages 1–8. Association for Computational Linguistics, 2006.

[5] Franklin Mark Liang. *Word hyphenation by computer*. Department of Computer Science, Stanford University, 1983.

[6] László Németh. Automatic non-standard hyphenation in openoffice. org. *COMMUNICATIONS OF THE TEX USERS GROUP TUGBOAT EDITOR BARBARA BEETON PROCEEDINGS EDITOR KARL BERRY*, page 32, 2006.