Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek

Tanszék

Szemantikus reprezentáció magyar nyelv esetén

Témavezető:

Grad-Gyenge László

Szerző:

Kántor Attila

Programtervező Informatikus MSc

Budapest, 2020

Tartalomjegyzék

1. fejezet

Bevezetés

A természetesnyelv-feldolgozás (NLP) a mesterséges intelligencia azon részterülete, amely az emberi eredetű beszélt és írott nyelvből történő információkinyeréssel és ezen tudás felhasználásával foglalkozik. A szemantikus reprezentáció az NLP intenzíven kutatott témaköre, amely algoritmusai képesek a természetes nyelven írott szövegek és szövegrészletek numerikus ábrázolására. A módszerek alapja, hogy a szavakat, vagy szavak listáját leképezzék valamely vektortérbe azok szemantikai tartalma alapján.

Az így kapott vektoroknak számos felhasználási módja létezik, például információ visszakeresés, dokumentum összegzés, chatbot-ok implementálása, gépi fordítás, stb. Napjainkban a legjobb eredményeket az ezeken a területeken kutató és fejlesztő óriáscégek által publikált módszerek érik el, de a hasonló technikák akadémiai körökben is nagy figyelmet kapnak.

A modern reprezentációs módszerek meghatározó tényezői az alapjukként szolgáló neurális háló és a tanításukra használt feladatok, adathalmazok. Bár léteznek többnyelvű reprezentációs modellek is, a meglévő technikák nagy részéről kijelenthető, hogy az nyelvfüggő. A nyelvfüggőség azt jelenti, hogy egy adott modell csak olyan nyelvű problémák esetén alkalmazható, amilyen nyelven tanították azt.

A friss eredmények azt mutatják, hogy címkézett adatokon történő felügyelt tanítás után modellünk magasabb teljesítményre lehet képes. Ez a tény problémát jelenthet a kevésbé beszélt nyelvek esetén, ahol csak elvétve, vagy egyáltalán nem léteznek ilyen tanítóadatok. A kevésbé populáris nyelveken való tanítás során jellemzően csak a nem felügyelt tanulás eszköztárából választhatunk.

A magyar egy nem túl széles körben beszélt nyelv, így a nyelvi modellek tanításához használható források is limitáltak. Diplomamunkám célja a meglévő módszerek vizsgálata, majd ezen tudás alapján olyan tanítóhalmazok létrehozása és technikák implementálása, amelyek alkalmasak lehetnek a kis és közepes nyelvek – így a magyar nyelv – szemantikai és szintaktikai tulajdonságainak ábrázolására.

2. fejezet

Előzmények

Ahogy a nyelvet is szétválaszthatjuk elemeire – például lexéma (szó), szintagma (szószerkezet) , mondat – , úgy a nyelvi elemeket reprezentáló módszereket is csoportosíthatjuk. A nyelvi elemek és a közöttük található nyelvtani kapcsolatok matematikai ábrázolására való törekvés már az előző évszázad közepén megjelent. Az idő során a különböző nyelvi elemek reprezentációs módszerei párhuzamos módon fejlődtek, ám a figyelem napjainkban leginkább a magasabb szintekre összpontosul. A mondatokat és a magasabb nyelvi szinteket ábrázoló algoritmusok jobbnál jobb pontosságot mutatnak a különböző NLP feladatok megoldását illetően.

2.1. Reprezentáció a szavak szintjén

A szószintű reprezentációs módszerek azt a célt szolgálják, hogy a természetes nyelven írott szöveg szavait numerikusan feldolgozhatóvá tegyék. Ha egy algoritmus képes abszolválni ezt a célt, a számítógép többé már nem karakterláncokat, hanem jelentést talál a bemenet mögött.

Bár az a gondolat, hogy szavakat matematikailag ábrázoljunk már a '80-as években felütötte a fejét, ezek a módszerek többnyire ritka reprezentációkat eredményeztek. A ritka reprezentációk csak kevés esetben hoznak hatékony megoldást. Számításigényük nagy lehet és néhány feladat esetén a kellő pontosság elérésére is alkalmatlanok.

2.1.1. Szótár keresés

A legegyszerűbb technika a szótár keresés, mely során L nyelv minden eleméhez injektív módon egy természetes számot rendelünk. L elemeit szótövezhetjük (*lemmatization*) is, így kisebb szótárat kapunk.

Ez egy kezdetleges és relatíve kis memóriaigényű algoritmus, azonban feladatainkat könnyedén félrevezetheti. A természetes nyelven írott szöveg szavai között csak ritkán találhatunk rendezést. A szótár keresést alkalmazva a szöveget feldolgozó rendszer fontosabbnak ítélheti azon szavakat, melyek nagyobb azonosítóval rendelkeznek, így hasonló esetekben a módszer használhatatlanná válik.

2.1.2. Valószínűség-alapú ábrázolás

Valószínűség-alapú ábrázolásnak nevezünk minden olyan módszert, amely a matematikai valószínűségszámítás eszközeit használja, többnyire eloszlást és gyakoriságot. Ezen reprezentációkat gyenge szemantikai erejük ellenére a mai napig alkalmazzák. Egyszerűek, de memóriaigényük nagy és a tanításuk is körülményes.

Gyakoriság és feltételes valószínűség

A csoportot képviselő alapvető algoritmus a gyakoriság alapú leképezés, amely azt az információt veszi figyelembe, hogy a dokumentumban hányszor szerepel egy adott szó. Használhatunk relatív gyakoriságot is, ha a gyakoriságot elosztjuk a dokumentumok összes szavának számával. Az így kinyert adat akár egyszerűbb szociális média analízisre is alkalmas lehet.

Szekvenciális adatok feldolgozására megfelelő választás lehet a feltételes valószínűség alapú leképezés, mely segítségével képesek lehetünk a szekvencia következő elemének prediktálására az előzőek függvényében.

Tf-Idf

A tf-idf egy statisztikai módszer, amely arra hivatott, hogy egy szó előfordulásának fontosságát ragadja meg egy dokumentumban, a dokumentumhalmazban. Az algoritmus a Bag Of Words (BOW) modellen alapszik, mely lényege, hogy L szótár esetén egy adott $d \in D$ dokumentumot egy $v \in \mathbb{N}^{|L|}$ vektor reprezentál. Ahányszor

előfordul $w \in L$ szó d dokumentumban, $v_{index(w)}$ értéke eggyel növekszik, egyébként marad 0.

A tf-idf két részből áll: *term frequency* és *inverse document frequency*. A végeredmény a két metrika szorzata. Mindkét metrikára több variáció is van, a legnépszerűbb a következő:

1. Definíció.

$tf\,(t,d) = \log\left(1 + freq\,(t,d)\right)$, *ahol freq(t,d) t szó gyakorisága d dokumentumban.*

$idf\,(t,D) = \log\left(\dfrac{N}{count\,(d \in D : t \in d)}\right)$, *ahol D dokumentumhalmaz elemszáma N.*

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

Bár a módszer számításigénye kicsi, továbbá jó választás lehet olyan esetben, ahol dokumentumok hasonlóságát szeretnénk mérni, csak lexikális információ reprezentálására képes.

Megjegyzés. *Természetesen a később bemutatott módszerekben is fellelhetők matematikai valószínűségszámítási eszközök.*

2.1.3. Szóvektorok

A valószínűségi modellek jól generalizálnak ritka bemenet esetén, azonban ha sűrűbb a bemenet, azok az algoritmusok bizonyulnak hasznosabbnak, amelyek a szavak jelentéstartalmát is képesek ábrázolni.

Azon feladatok során, amikor a szemantikának nagyobb szerepe van – ilyen lehet az írott szöveg érzelmi tartalmának vizsgálata – , nem használhatjuk a fenti technikákat. Olyan reprezentációs módszert kell találnunk, amely képes komplexebb problémákat is megoldani. Ilyen probléma például, ha egy szó több jelentéssel is bír (pl.: mész), a szinonímák és a kontextusfüggő szóhasználat (pl.: víz - H_2O).

A szóvektorok részben megoldást nyújthatnak ezen komplikációkra. Szóvektorokat úgy kapunk, ha a lexémákat leképezzük valamely vektortérbe. Ha két szó szemantikai tartalma hasonló, szóvektoruk távolsága kicsi.

One-hot kódolás

2. Definíció. *Legyen L egy $n \in \mathbb{N}$ elemű nyelv. Ekkor $w \in L$ szó one-hot kódolásán $v \in \{0,1\}^n$ vektort értjük, ahol*

$$v_i = \begin{cases} 1, & \text{ha } L_i = w \\ 0, & \text{egyébként.} \end{cases}$$

A *one-hot* kódolás egy egyszerű és nem hatékony reprezentációs módszer, azonban mégis a szóvektorokhoz sorolhatjuk. Minden szóvektort a vektortér egy-egy dimenziója reprezentál, így a vektorok merőlegesek egymásra. Az algoritmus legfőbb gyengesége, hogy képtelen relációs információkat és szemantikát kódolni, így nem tudja kezelni a szinonímákat, teljesen különböző szavaknak tekinti azokat.

Megjegyzés. *A one-hot kódolás ritka reprezentációt eredményez.*

Szóvektorok - folytatás

Ha egy gyors megoldásra lenne szükségünk, vagy egyszerűen szeretnénk neurális hálónk bemenetére juttatni a szöveg szavait a *one-hot* kódolás jó választás lehet. Azonban ha jelentéstartalmat szeretnénk modellezni, ennél komplexebb reprezentációs módszerre lesz szükségünk, ilyen lehet például a szóbeágyazás.

A szóbeágyazás azon a feltevésen alapszik, hogy a hasonló kontextusban előforduló szavak hasonló jelentéstartalommal bírnak. A Word2Vec és a GloVe algoritmusok képesek ezen lexémák közötti relációs információ feldolgozására.

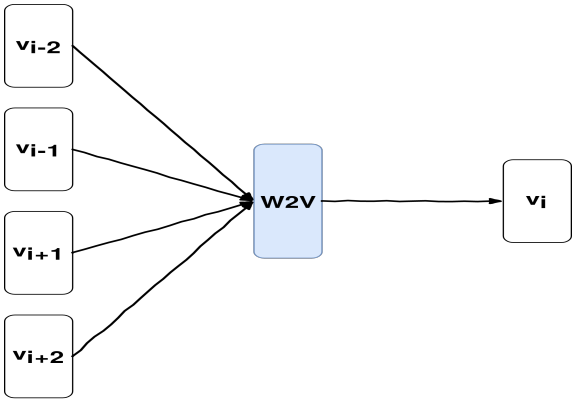
Word2vec

A Word2Vec [1] módszer egy sekély neurális hálón alapuló szóbeágyazási algoritmus, melyet 2013-ban mutattak be. A háló tanítását a szerzők alapvetően két felügyelet nélküli feladattal végezték: Continuous Bag of Words (CBOW) vagy Skip-Gram.

A tanítás során a mondatokat tokenekre bontották és one-hot kódolták. Majd a szöveg minden egyes tokenjén végigiterálva a következőket hajtották végre:

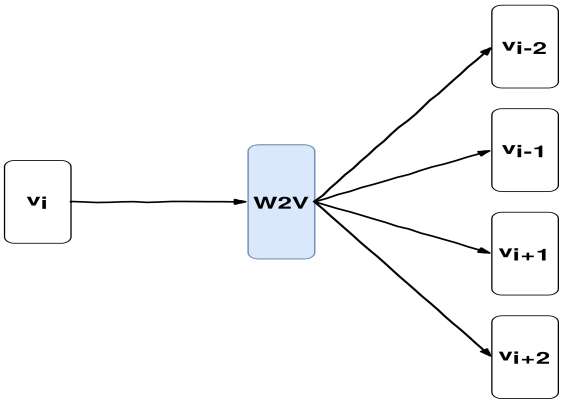
A CBOW modell szerint a háló bemenete v_i $(i \in |D|)$ vektorra a v_i vektor k méretű kontextusa $(v_{i-k}, ..., v_{i-1}, v_{i+1}, ..., v_{i+k} : k \in \mathbb{N})$, azaz a környezetében lévő

vektorok. A háló feladata prediktálni v_i vektort a kontextus függvényében. A folyamat közben a háló rejtett rétegében létrejön a Word2Vec reprezentáció.



2.1. ábra. CBOW modell

Skip-Gram modell esetén pont az ellenkezője történik. A háló bemenete v_i ($i \in |D|$) vektor lesz. A tanítás célja, hogy a háló prediktálja az i. szó k méretű kontextusának one-hot kódolt vektorait $(v_{i-k}, ..., v_{i-1}, v_{i+1}, ..., v_{i+k} : k \in \mathbb{N})$, közben a háló a rejtett rétegében megtanulja a Word2Vec reprezentációt.



2.2. ábra. Skip-Gram modell

Egy jól tanított Word2Vec modell a hasonló jelentéstartalmú szavak vektorait közelre képezi egymáshoz a vektortérben. A további precizitás érdekében finomhangolhatjuk a tanítási paramétereket. Ilyen beavatkozás lehet ha növeljük a halmaz méretét, amellyel Word2Vec modellünket tanítjuk, vagy emeljük a kontextus ablak méretét és a reprezentációs dimenziót.

Megjegyzés. *A Skip-Gram modell a ritka szavak, míg a CBOW modell a gyakori szavak esetén készít pontosabb reprezentációt.*

GloVe

A Word2Vec bemutatását követő évben újabb nagy lépés történt a szóbeágyazás világában, a *Stanford University* NLP kutatócsoportja publikálta a GloVe módszert. A GloVe (*Global Vectors*) [2] reprezentációs módszer a Word2Vec-hez képest egy korpusz lokális statisztkáján kívül a globális statisztikáit is figyelembe veszi.

3. Definíció. *Adott egy korpusz, melynek elemszáma V. Az* $X \in \mathbb{N}^{V \times V}$ *mátrixot közös előfordulási mátrixnak nevezzük, ahol* X_{ij} *az a szám, ahányszor i. szó kontextusában j. szó megjelenik.*

A GloVe modell tanítása egy korpusz közös előfordulási mátrixának nemnulla elemein történik. A GloVe modell egy log-bilineáris modell, amely feladata, hogy kiszámítsa a következő szó valószínűségét azon kontextusa alapján.

A módszer mögötti intuíció az, hogy a közös előfordulási valószínűségek hányadosa értékes információval szolgálhat a leképezés során. Így a feladat célja, hogy a tanult szóvektorok skaláris szorzata megegyezzen a szavak közös előfordulási valószínűségének logaritmusával. Mivel $\log\left(\frac{A}{B}\right) = \log(A) - \log(B)$, így ez a cél összekapcsolja az előfordulási valószínűségek arányszámát a vektorok távolságával.

Ugyan a globális statisztikáknak köszönhetően a GloVe több feladatban is túlteljesíti a Word2Vec-et, a tanításához szükséges közös előfordulási mátrix memóriaigénye magas. Paraméterhangolás esetén újból fel kell építeni a mátrixot, amely költséges művelet.

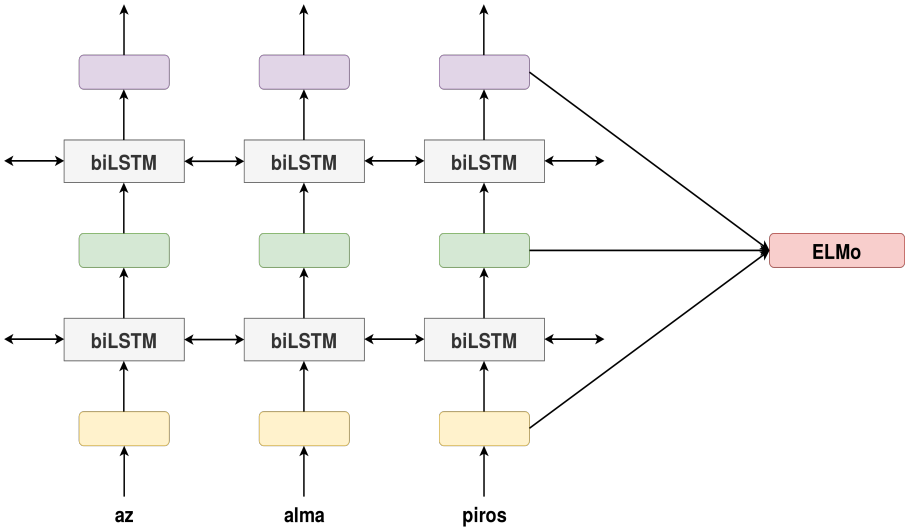
ELMo

A Word2Vec és a GloVe már képes szemantikus információ leképezésére, azonban esetükben az ellentétes szópárok közel kerülnek egymáshoz. Azon feladatoknál, melyeknél az ellentétes szavak kiemelt szerepűek – például a hangulatelemzés – limitációk jelentkezhetnek, továbbá ezen algoritmusok rosszul kezelik az ismeretlen szavakat is.

A Word2Vec és a GloVe csak szavankénti kontextusfüggetlen reprezentáció tanulására képes. Ez azt jelenti, hogy a szótárban található szavakhoz bijektív módon hozzárendelnek egy vektort, így nem számít az adott szó szövegkörnyezete, amelyre aktuálisan alkalmazzák. Az Embeddings from Language Models (ELMo) [3] fi-

gyelembe veszi a lexémák kontextusát, mondaton belüli elhelyezkedését is, továbbá használat közben állítja elő a vektorokat.

A modell tanításához használt neurális háló több réteg kétirányú LSTM (BiLSTM) konkatenációja. A különböző rétegek más és más típusú információt képesek kinyerni.



2.3. ábra. ELMo modell

Az ELMo a különböző rétegek kimenetének feladatspecifikus kombinációján alapszik. Egy adott NLP feladatra minden BiLSTM réteg egyedi súlyt kapott. A végső háló 2 darab BiLSTM rétegből állt, minden LSTM réteg mérete 4096 volt. Az így kapott sekély kétirányú módszer jelentősen javított a szóvektorok pontosságán.

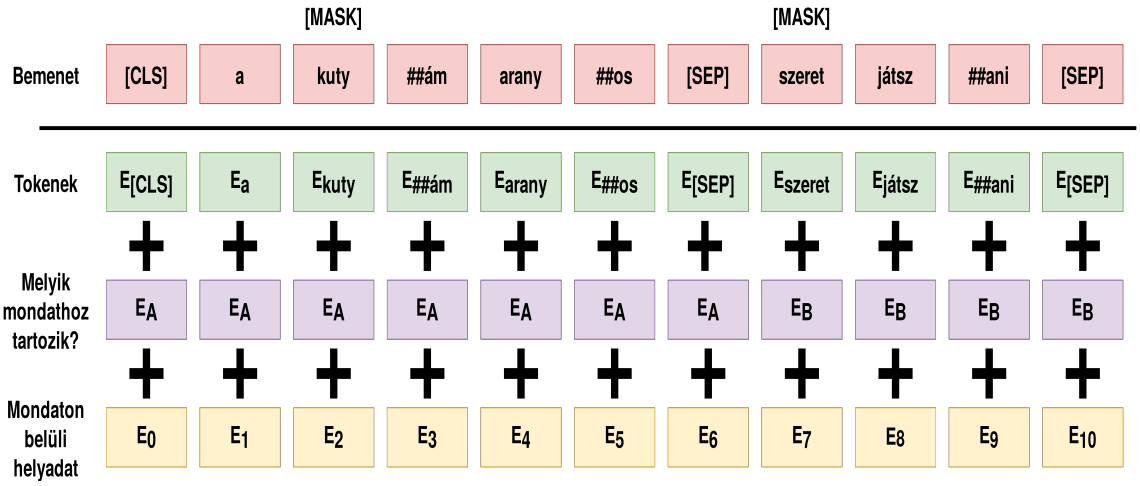
Bár az ELMo egy karakter (konkatenáció) alapú reprezentációs algoritmus, szavakat ábrázol. Ezen tulajdonsága alapján képes kezelni az addig nem látott szavak problémáját is.

BERT

Egy 2018-ban publikált cikk [4] rámutatott arra, hogy a karakteralapú algoritmusok – tehát amelyek karaktereket fogadnak bemenetül tanulás során – nem teljesítenek olyan jól, mint a szóalapú társaik. A Bidirectional Encoder Representations from Transformers (BERT) [5] egy a Google által kifejlesztett transformer architektúrájú nyelvi modell. Az ELMo-hoz hasonlóan ez is kétirányú, azaz egy szó mindkét

oldali kontextusát figyelembe veszi a tanulás alatt. A BERT azonban bemenetként nem szavakat és nem is karaktereket kap, hanem szótöredékeket.

A szerzők a tanítást a *transfer learning* szerint két fázisra bontották: előtanítás és finomhangolás. Az előtanítás két feladatból állt: *következő mondat* és *maszkolás*. A bemenetben megadták a szótöredék tokeneket, a tokenekhez tartozó mondaton belüli helyadatokat és azt, hogy az adott token A vagy B mondat közül melyikhez tartozik.



2.4. ábra. A BERT bemenete

A *következő mondat* esetében a mélyháló feladata volt kitalálni, hogy A[SEP]B input mondatokra B rákövetkezője-e A-nak. A *maszkolás* során véletlenszerűen letakarták a tokeneket a mondatokban és a mélyháló megpróbálta kitalálni, hogy eredetileg melyik szó volt a [MASK] token helyén. A [CLS] token a klasszifikációs feladat alatt a mondatot ábrázolta, a [SEP] a mondatok közötti szeparátor volt és a [MASK] a letakart szótöredékeket helyettesítette. A továbbiakban a modell finomhangolása az adott NLP feladat szerint történt.

Míg az ELMo különböző balról-jobbra és jobbról-balra olvasó rétegek konkatenációjaként állítja elő a vektorokat, addig a BERT a valódi mély architektúrájával csak egyszer dolgozza fel a tokeneket. A *transformer* architektúra nem igényel vektoriális bemenetet, saját reprezentációt épít a tokenek számára is. A BERT a szóvektorok előállításán kívül a teljes szekvenciát is képes ábrázolni, ezt a bemeneti [CLS] tokenhez tartozó reprezentációs vektor segítségével teszi.

A BERT szótöredék alapú megoldása egyesíti a karakteralapú modellek előnyét a szóalapú modellek előnyével. Képes kezelni az ismeretlen szavakat és performanciája mégis magas marad. A *következő mondat* feladat a szövegben található mondatok közötti relációk, a *maszkolás* pedig a mondatokon belüli szemantikai és szintaktikai információ ábrázolását segíti. Több NLP feladat megoldásában is jelenleg a BERT a *State-of-the-art*.

2.2. Reprezentáció a mondatok és magasabb nyelvi elemek szintjén

Ahogy a lexémák szemantikai tartalmát sem határozza meg az őket alkotó karakterek lánca, úgy a mondatok sem értelmezhetőek pusztán a magukban foglalt szavak halmaza alapján. A mondatok és magasabb nyelvi elemek interpretálása során fontos tényezők lehetnek a bennük lévő szintaktikai viszonyok és a kontextus is.

Néhány NLP feladatnál, mint például a dokumentumok szemantikus keresésénél, vagy szöveg összegzésnél szükség lehet magasabb szintű reprezentációkra. Ezek a módszerek szavak helyett mondatokat, bekezdéseket, vagy akár egész dokumentumokat tesznek numerikusan értelmezhetővé.

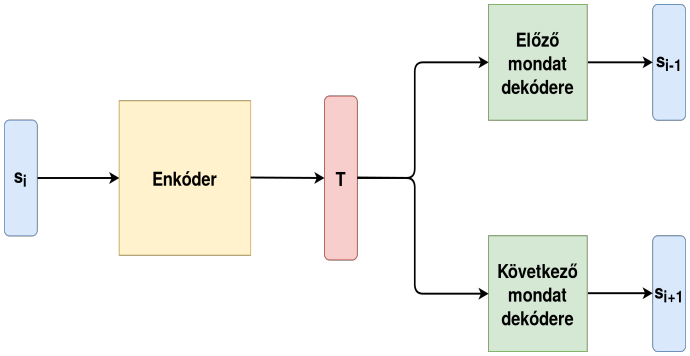
2.2.1. Mondatvektorok

A szóvektorokhoz hasonlóan úgy kaphatunk mondatvektorokat, ha mondatokat helyezünk el egy vektortérbe. A tanítás során azonban a sorrendiség, az egyes lexémák változó súlya és a szintaktikai viszonyok megnehezíthetik dolgunkat. Szükség van egy technikára, mely segítségével leképezhetjük és szemantikai tartalmuknál fogva összegezhetjük a megfelelő rendezett szóvektorok sorozatát, így hozzájutva az adott mondat reprezentációjához. A módszerünk akkor hatékony, ha az azonos jelentéstartalmú mondatvektorok klaszterekbe tömörülnek a vektortérben.

Skip-thought vektorok

A Skip-though [6] egy 2015-ben bemutatott mondatreprezentációs módszer – a Skip-Gram algoritmus kiterjesztése – , amely a környező mondatokat is figyelembe veszi a tanulás során.

A szerzők rekurrens enkóder-dekóder architektúrát használtak a tanításhoz. A neurális háló bemenete mondathármasok szavainak Word2Vec vektoraiból állt. A háló feladata s_i mondat esetén s_{i-1} és s_{i+1} mondatok generálása volt.



2.5. ábra. A Skip-thought enkóder-dekóder architektúrája

Az enkóder és dekóder blokkokhoz rekurrens hálót használtak, melyek lehetnek LSTM és GRU rétegek is. Az enkóder célja az volt, hogy a legjobb teljesítményével segítse a dekóder blokkokat, míg a dekóder blokkok célja minimalizálni az előző és a következő mondat rekonstrukciós hibáját. Olyan szavak esetén, melyeket a háló még nem ismert, tanítottak egy $f : V_{w2v} \rightarrow V_{rnn}$ lineáris leképezést, ahol V_{w2v} és V_{rnn} rendre a Word2Vec és a rekurrens modell szótára. A tanult reprezentáció vektora a rejtett, úgy nevezett *thought* vektor (T).

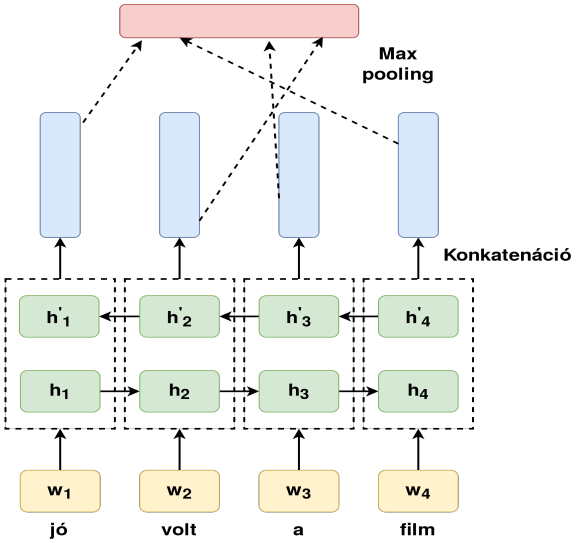
Bár a *Skip-thought* módszer képes a mondatokon belüli és kívüli sorrendiségi információ leképezésére is, csak olyan esetben teljesít megfelelően, ahol az egyes mondatok – melyekre alkalmazzák – megfelelő kontextusban szerepelnek, nem izoláltak.

InferSent

2017-ben a Facebook kutatói jelentős áttörést értek el a mondatszintű reprezentációs módszerek terén, a technika neve InferSent [7]. Hasonló algoritmusokkal ellentétben a szerzők felügyelt tanítást végeztek, melyhez az SNLI adathalmazt [8]

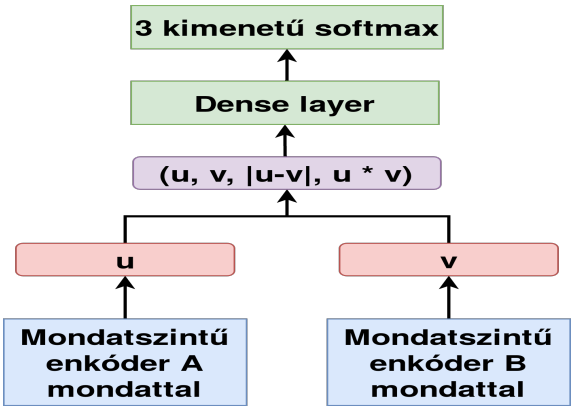
vették igénybe. A cikk megmutatta, hogy egy kisebb adathalmazon történő felügyelt tanítás felülmúlhatja a nagyobb adathalmazon nem felügyelt módon tanított modellek teljesítményét.

Az SNLI adathalmaz 570 ezer darab – emberek által írt és címkézett – mondatpárból áll. A címkék a következők: következmény, ellentmondás és semleges. Négyféle neurális architektúrát összemérve a legpontosabb eredményt a BiLSTM *max pooling* prezentálta.



2.6. ábra. A BiLSTM max pooling architektúra

Az SNLI feldolgozásához szükséges NLI feladat speciális szerkezetet igényelt. Mivel kontextusfüggetlen reprezentációt akartak előállítani, amely izolált formában is működik, a mondatpárok GloVe vektorait szeparáltan enkódolták.



2.7. ábra. Az NLI feladat

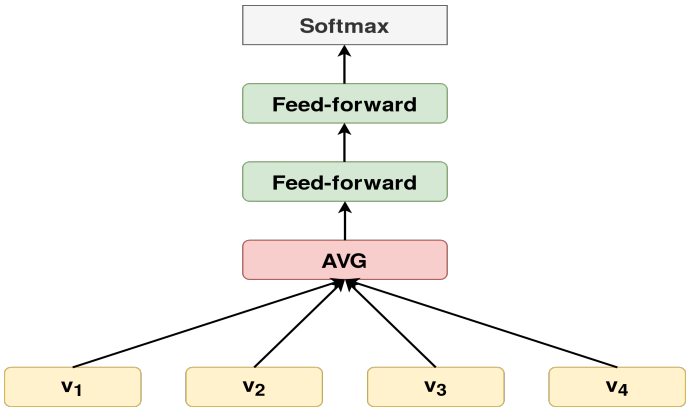
Az így készült u és v vektorokból egy speciális reprezentáció készült: u, v, $|u - v|$ és $u * v$ (vektoriális szorzat) konkatenációjával, melyet végül egy 3 osztályú klasszifikáló hálóba vezettek.

A szerzők a reprezentációs vektorméret növelésével pontosabb eredményt kaptak, de a vektorok memóriaigénye emelkedett. Az InferSent megoldja a kontextusfüggőség problémáját, így a módszer szövegrészletekre is alkalmazható.

Megjegyzés. *Az InferSent napjaink egyik legjobb teljesítményű mondatszintű szemantikus reprezentációs algoritmusa.*

USE

Az InferSent bemutatását követő évben a Google Research csapata a modern reprezentációs módszereket vizsgálta a *transfer learning* aspektusából. A Universal Sentence Encoder (USE) [9] egy mondatszintű algoritmus, mely célja, hogy a használója könnyedén igényeire tudja formálni, annak érdekében, hogy pontosabb leképezést kapjon. A szerzők két architektúrát használtak: a BERT-ben említett *transformer*-t és a DAN-t (*Deep Averaging Network*).



2.8. ábra. DAN architektúra

A *transformer* modell egyik algráfja a mondatokban lévő szavak kontextusfüggő reprezentációját állítja elő. A folyamat során figyelembe veszi az egyes lexémák sorrendi és egyéni információit is, majd összegzi őket, így megkapja a végső mondatszintű reprezentációt. A *DAN* modell az input tokenek vektorait először átlagolja, majd *feed forward* rétegek segítségével előállítja a mondatvektorokat. A USE szavak-

ból, mondatokból, vagy akár rövidebb bekezdésekből is képes 512 méretű vektorokat generálni.

A neurális hálók tanítását két részre bontották, bemenetként angol nyelvű szöveget kaptak. Az első rész a *Skip-thought*-hoz hasonló módon, dialógusokból vett mondat-válasz párokkal, illetve felügyelt módon a *Stanford Natural Language Inference* (SNLI) korpuszon történt.

A cikkben kiemelt szerepet kapott a tanítás második fázisa. Számos módon finomhangolták a modelleket és mérték a teljesítményüket. A feladatok közé tartozott, hogy filmes értékelések szövege alapján ki kellett találnia a neurális hálóknak az értékelések pontszámát 1 és 5 között. Továbbá vásárlói vélemények hangulati töltetét kellett prediktálniuk.

Az algoritmusokat kipróbálták a szavak szintjén, a mondatok szintjén és a kettő módszer konkatenációjaként is. A legjobb teljesítményt a mondatszintű reprezentációk mutatták. A transformer architektúra pontosabb eredményt hozott, mint a DAN alapú modell, de a transformer modell $\mathcal{O}(n^2)$, míg a DAN modell $\mathcal{O}(n)$ időkomplexitású volt a bemeneti hossz függvényében. Továbbá memóriahasználatban is kedvezőbb választás volt a DAN.

A GloVe-hoz hasonlóan a USE is képes asszociációkra, de jóval gyengébb ezen képessége az olyan kényes témák esetében, mint a szexizmus és a rasszizmus. Ez a tény alkalmassá teheti a USE-t az ipari használatra is.

A szerzők rávilágítottak arra, hogy kevés adat esetén jó választás lehet a *transfer learning* módszere és a magasabb szintű reprezentációk pontosabb eredményt érhetnek el a legtöbb feladat esetében.

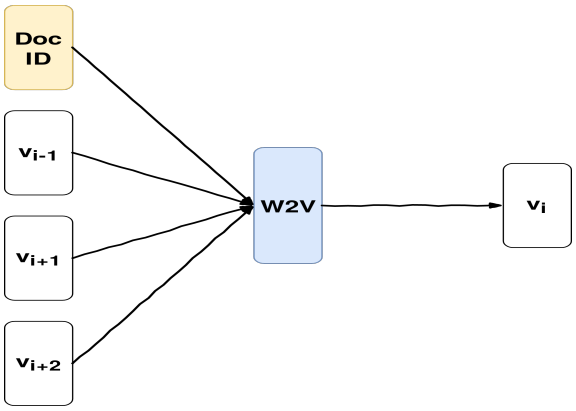
2.2.2. Dokumentumszintű reprezentáció

Ahogy a technológia fejlődik, úgy növekszik a világon az egységnyi idő alatt előállított információ mennyisége is. Gyakori eset, hogy ez írott formában, dokumentumokban jelenik meg. Dokumentumnak tekinthetünk minden, a mondatnál hosszabb emberi nyelven írott szöveget.

Bár a magasabb nyelvi egységek értelmezése és feldolgozása sok területen előkerülő feladat, mégsem triviális. Nagy kihívást jelent a szemantikai szimilaritás mérése,

az olyan gyakorlati problémákat nem is említve, mint a duplikációk kiszűrése a fórumokról, vagy a szociális média analízis.

2014-ben a Word2Vec szerzői előálltak egy dokumentumszintű reprezentációs algoritmussal. A Doc2Vec [10] módszer a Word2Vec modell kiterjesztése a dokumentumok szintjére. Mivel a dokumentumokat nem lehet a szavakhoz hasonló logikai struktúrába rendezni, ezért a megszokott *CBOW* modell bemeneti vektorai mellé egy speciális, a magasabb nyelvi elem azonosítóját jelölő vektort konkatenáltak. Az algoritmus neve PV-DM. A modell tanítása végén a speciális vektor reprezentációja képviselte a dokumentumot.



2.9. ábra. Doc2Vec PV-DM architektúra

A Word2Vec-hez hasonlóan a Doc2Vec-nek is létezik *Skip-Gram* alternatívája, ez a PV-DBOW. A PV-DBOW modell gyorsabb és memóriahasználat szempontjából is gazdaságosabb a PV-DM-hez képest.

Mivel relatíve kevés algoritmus képes dokumentumszintű modellezésre és azok teljesítménye is limitált, a Doc2Vec egy jó választás lehet. A modell egyszerre mutat jó teljesítményt és a használata is egyszerű.

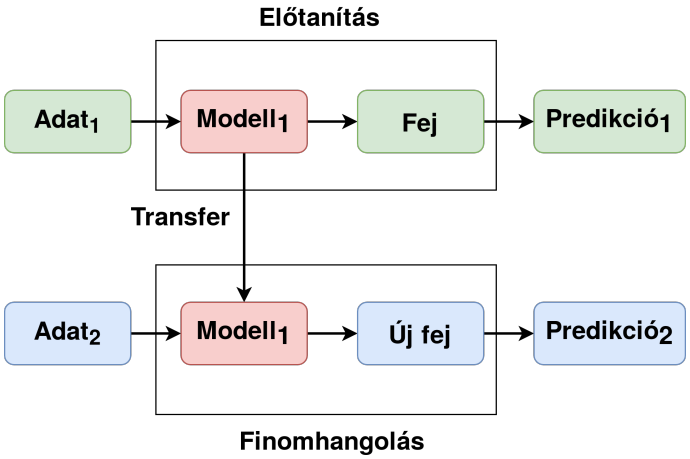
2.3. Transfer learning

A modern szemantikus reprezentációs algoritmusok tanítása összetett folyamat. A feladatok során egyszerre kell több szempontra figyelni, melyek befolyásolhatják a modellünk pontosságát. Példának okáért mondatszintű reprezentációnknak képesnek kell lennie értelmezni a lexémák egymáshoz fűződő viszonyait és a mondatok

közötti kohéziót is. A *transfer learning* egy kiváló eszköz arra, hogy modellünket több aspektus szerint tanítsuk.

A *transfer learning* napjainkban közkedvelt tanítási módszer, melynek ötletét az NLP ágazata a számítógépes látás eszközkészletéből merítette. A folyamatot két fázisra lehet bontani: előtanítás és a finomhangolás. Az előtanítás általában nagy mennyiségű adaton történik. A finomhangolás az előtanítás után kapott modell – adott NLP feladathoz szükséges – speciális feladatokon való tanítását jelenti, amely szignifikánsan kevesebb adatot igényel.

4. Definíció. *Jelölje D_s a forrástartományt, D_t a céltartományt, T_s a forrástartományhoz tartozó feladatot, továbbá X_t és Y_t rendre a T_t célfeladathoz tartozó inputváltozók és címkék halmazát. A* ***transfer learning*** *célja megtanulni $P(Y_t|X_t)$ feltételes eloszlást D_t-ben D_s által gyűjtött információ alapján úgy, hogy $D_s \neq D_t$ vagy $T_s \neq T_t$.*



2.10. ábra. Transfer learning

Az előtanítási szakasz olyan feladattal kezdődik, amely kellőképpen generalizál és a neurális hálónk sok, hasznos és általános információhoz tud jutni. A folyamathoz használt adathalmaz általában nagy mennyiségű annotálatlan adatot tartalmaz, de lehetnek kivételek, például a USE esetében.

A finomhangolási fázis alatt használt feladatok az előtanítás után kapott modell súlyait alkalmazzák, de a bemeneti adatok és a feladatok végrehajtásához szükséges fej lehet eltérő is. Ezen szakasz állhat feladatok sorozatából is, ekkor a súlyokat

inkrementálisan használják azok. A sikeres végrehajtást követően modellünk képes lesz komplexebb összefüggések felismerésére és jobb teljesítmény elérésére.

A jelenlegi trendek szerint a reprezentációs módszerek tanítási módja nagyobb hangsúlyt kap, mint maga a neurális háló szerkezete. A *transfer learning* használata a numerikus ábrázolás során még kiaknázatlan terület, mely rendkívül sok eredményt hozhat a jövőben.

3. fejezet

Adatkonverzió és adathalmazok

Ahogy az előző fejezetben is láthattuk, a mai modern szemantikus reprezentációs modellek többsége neurális hálók segítségével képezi le a nyelvi elemeket valamely vektortérbe. A neurális modellek a feladatok során felfedezik az adathalmaz rejtett mintáit és megtanulják az elemeinek eloszlását. Kevés adat esetén nem várhatjuk el a hálónktól a megfelelő pontosságot, mivel a tanítóminta nem reprezentatív az adott problémára. Ezen felül a túltanulás is erősen eltérítheti a tanulási folyamatot.

Bár az olyan nyelveken, amelyeken a kutatásokat folytatják és amelyeket széles körben beszélnek előfordulhat emberi beavatkozás által annotált adat is – ilyen például az SNLI – , a reprezentációs módszerek tanítását jellemzően auto-annotált adatokon végzik. Auto-annotált adatnak tekintünk minden olyan adatot, amelyek címkézését nem ember hajtotta végre. Az auto-annotált tanítóhalmazok hátulütője, hogy pontosságuk sokszor nem éri el az emberi szintet és jelentős zajt is tartalmazhatnak. A reprezentációs algoritmusok a kisebb méretű, de emberi intelligencia által annotált halmazokon precízebb eredményt érnek el. [7]

A magyar nyelv a kisebb körben használt, közepes nyelvek közé tartozik, így bátran vonhatjuk le azt a következtetést, hogy a web és egyéb források által hozzáférhető szöveges tartalmak mennyisége is erősen limitált. Ennek okán munkám során fontos tényezőnek tartottam, hogy olyan jellegű adatokkal dolgozzak, melyek könnyen megszerezhetőek. Megfelelő választásnak bizonyultak a többnyelvű, publikus adathalmazok és az olyan profilú online elérhető dokumentumok, melyeket valamely *webscraper*-el össze lehet gyűjteni. Az így kialakult módszerek alkalmasak lehetnek arra, hogy akár más, kevésbé széleskörűen beszélt nyelvek esetén is

alkalmazzák őket.

3.1. A nyers szöveg általános előkészítési lépései

A nyers szöveg előkészítése elengedhetetlen folyamat az NLP feladatok során, mely nélkül értelmetlen eredményeket kapnánk. A jól elkülöníthető lépések után olyan kimenethez jutunk hozzá, amely lényegesen jobb feltételeket biztosít algoritmusunknak ahhoz, hogy képes legyen a dokumentumokat numerikusan értelmezni.

Az előkészítési szakasz a legtöbb esetben az úgynevezett tokenekre való bontással kezdődik. A **tokenizáció** a dokumentumok granularitásának növelésére szolgál. A bekezdéseket mondatokra, majd szavakra oszthatjuk, így hozzáférhetünk az olyan relációs információkhoz is, melyeket az alacsonyabb rétegek tárolnak.

Az adathalmaz **tisztítása**, vagy zaj csökkentése az olyan karakterek és karakterláncok eltávolítását jelenti, amelyek nem elemei a célnyelvnek. Adataink tartalmazhatnak akár speciális karaktereket, írásjeleket, HTML tag-eket, számokat és túl rövid – például 1 karakter hosszú – tokeneket is, melyek megzavarhatják modellünk működését. A tisztítás során törölhetjük az adott nyelvben sűrűn előforduló szavakat (*stopword*) is – például névelők – , így csak azok a tokenek maradnak a halmazban, amelyek valódi információtartalommal bírnak.

A szöveg **normálása** olyan módosításokat jelent, amelyek során az adathalmazunk tokenjeit azonos alakra hozzuk. A tokeneket kis-, vagy nagybetűssé konvertálhatjuk, illetve a numerikus tartalommal rendelkező szavakat számokká alakíthatjuk. Természetesen ebben az esetben is célszerű törölni a numerikus tokeneket, ha a tisztítás során is így jártunk el.

Megkülönböztethetünk két **szótövezési** formát, a *stemming*-et és a *lemmatization*-t. Mindkét módszernek az a célja, hogy eltávolítsa a ragokat a szótövekről. Míg a *stemming* egy nyers heurisztikákon alapuló módszer, addig a *lemmatization* pontosan próbálja meg szótári alakba konvertálni a szavakat szótár és morfológiai analízis segítségével. A normálás és szótövezés után egy csökkentett elemszámú szótárat kapunk, így az eredeti állapothoz közelítő pontossággal, de szignifikánsan kevesebb számítás- és memóriaigénnyel el tudja végezni az algoritmusunk a feladatát.

Az előkészítés végső lépése lehet az **n-gram**-ok bevezetése az adatsorunkba. Az n-gram kifejezés egy n hosszú tokenszekvenciára utal, tehát a "New York" szóösszetétel 2-gram (bigram) lesz. Az n-gramok építése az n-gram modell feladata. A dokumentumhalmazunkon tanított n-gram modell az adott token prediktálását végzi el az előző $n-1$ token függvényében. Vegyük példának az előbbi bigram-ot ($n = 2$):

$$P(\text{New York bigram}) = \frac{P(\text{A szám, ahányszor New és York egyszerre szerepelt})}{P(\text{A szám, ahányszor New szerepelt})} \tag{3.1}$$

N-gram modellünk minden n hosszú tokenszekvencia esetén elvégzi a számítást, majd a legmagasabb előfordulási valószínűségű szópárokat "_" jellel konkatenálja, tehát "New York" esetén New_York-ot kapunk. Egy jól működő bigram modell elegendő lehet a feladatra, általában nincs szükség magasabb szintű összevonásra. A bigramok megkönnyíthetik nyelvi modellünk munkáját azzal, hogy a vélhetően összetett fogalmak különálló tokenjeit konkatenálják, így a tanítás során az algoritmusunk egy tokenként kezelheti a népszerű kifejezéseket.

Korábbi tapasztalataim azt mutatják, hogy a fenti technikák együttes alkalmazása lényegesen javíthatja az NLP feladatok – megfelelő pontossággal való – megoldásának esélyeit, ennélfogva a munkám során használt adathalmazok mindegyike maradéktalanul átesett az egyes előkészítési lépéseken.

A szótövezés során két *lemmatizer* algoritmust hasonlítottam össze, ezek a Hunspell [11] és a Hungarian SpaCy [12]. Az adathalmazok előkészítése alatt úgy tűnt, hogy a Hungarian SpaCy kevésbé mohó módszerrel vágja le a ragokat, ezért úgy döntöttem, hogy a továbbiakban azt használom, ugyanakkor nem vetem el annak a lehetőségét sem, hogy az erősebb szótövezés pontosabb végeredményt hozhat.

Az implementációt Python nyelven végeztem, továbbá a SpaCy [13] és az NLTK [14] nevű könyvtárakat használtam segítségül.

3.2. Magyar Wikipédia oldalak

A Wikipédia [15] a világ egyik legnagyobb többnyelvű, szabadon szerkesztett online enciklopédiája. Több, mint 50 000 000 dokumentumot tartalmaz, melyek egy-egy témakört, vagy fogalmat írnak le.

A szemantikus reprezentációs algoritmusok tanítása Wikipédia cikkeken nem új keletű ötlet. Számos nyelvi modell alapszik ezen az adathalmazon, többek között a BERT is. Az online enciklopédia jól dokumentált alkalmazásprogramozási interfésszel rendelkezik, így tudtam én is hozzájutni a magyar nyelvű oldalak szövegéhez.

A letöltött nyers adathalmaz mérete összesen 2.4 GB, melynek a wiki_hu nevet adtam. A wiki_hu 459 286 darab magyar nyelven írt Wikipédia cikket, 16 301 289 sort és 150 333 446 tokent tartalmaz. A tanításhoz szükséges előkészítés után az adathalmaz mérete 2 GB-ra, a sorok száma 12 592 489-re, a tokenek száma pedig 86 605 435-re csökkent. A hozzáfűzött reményekkel ellentétben magyar nyelvű cikkek relatíve elenyésző mennyiségben szerepelnek a Wikipédia adatbázisban. Következésképp a halmaz nem bizonyult megfelelőnek a probléma megoldására, így a továbbiakban csak a különböző technikák tesztelésére tudtam használni.

3.3. OSCAR adathalmaz

Az OSCAR (*Open Super-large Crawled ALMAnaCH coRpus*) [16] egy nyelvi klasszifikáló algoritmussal készült adathalmaz, melyet a szerzők a Common Crawl [17] szétválogatásából és szűréséből kaptak, majd a sorait összekeverték. A Common Crawl egy 2011 óta gyűjtött publikus webarchívum. Az OSCAR magyar nyelvű szegmensének teljes mérete összesen 40 GB.

Az előkészítési szakasz előtt szétválasztottam az adathalmazt két egyenlő részre, így két darab 20 GB-os szeletet kaptam. A továbbiakban az eredeti adatsor első felével folytattam az előkészületeket, melynek az oscar_hu nevet adtam. Az oscar_hu nyers változata 127 654 271 sort és 5 168 152 283 darab tokent tartalmaz. Az előkészítési procedúra után 15 GB-ra csökkent a méret, 63 692 408 sor és 1 626 357 463 darab token maradt.

Ugyan az oscar_hu nem tartotta meg a sorok közti relációkat, azonban az így kapott adatsokaság – a mennyiségénél és annál a ténynél fogva, hogy a Word2Vec csak lokális információkkal dolgozik – alkalmasnak bizonyult a szóbeágyazás tanítására.

3.4. A Hungarian Webcorpus

A Hungarian Webcorpus [18] a ma létező legnagyobb kifejezetten magyar nyelvű korpusz, melyet a Budapesti Műszaki Egyetem Média Oktató és Kutató Központja gyűjtött 2003-ban a SzóSzablya projekt keretein belül.

A korpusz 18 millió .hu domain-al rendelkező weboldal szövegéből áll, melyekből eltávolították a duplikált tartalmakat és az értelmetlen sorokat. Az így kapott adathalmazra helyesírás ellenőrző szoftvert is futtattak. A publikált dokumentumok szavainak csupán 4%-a volt felismerhetetlen a helyesírás ellenőrző szerint, így a végeredményben szereplő dokumentumok kevesebb nyelvtani hibát tartalmaznak, mint egy átlagos nyomtatott dokumentum. A végső korpusz 589 millió szóból áll, melyet 1 221 000 magyar nyelvű weboldalról töltöttek le.

Az elérhető fájlok ISO Latin-2 formátumban voltak, így az előkészítési folyamat előtt átkonvertáltam őket UTF-8 formátumba. Ez az átalakítás további megoldandó karakterproblémákhoz vezetett. A Hungarian Webcorpus eredeti mérete 18 GB volt, mely a tisztítási lépés után 7.8 GB-ra csökkent. A jelentős méretváltozás az XML tag-ek törlésére vezethető vissza. Az előkészítés utáni változatban a méret tovább zsugorodott, így a végeredmény egy 6.6 GB-os adathalmaz lett.

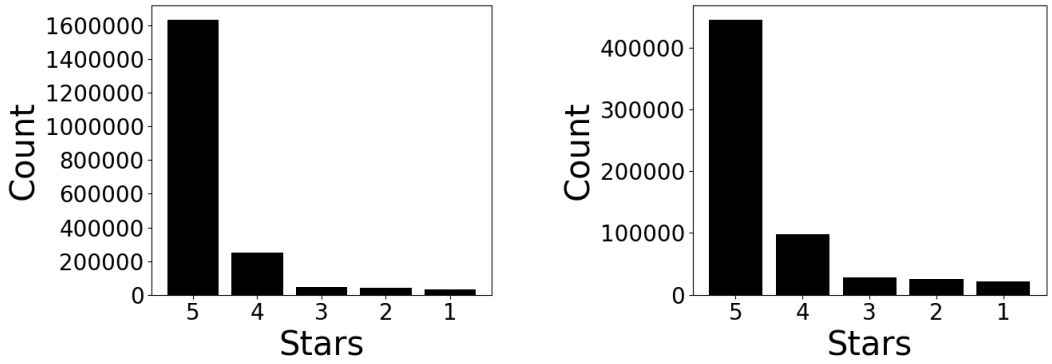
Az 1 221 405 fájl az előkészületek után összesen 127 711 725 sort és 589 209 017 darab tokent tartalmazott.

3.5. Termék vélemények

Az Árukereső [19] a legnépszerűbb magyar nyelvű online áruösszehasonlító oldal, melyen több mint 16 millió termék és 3 500 partner részletes adata szerepel. A felhasználóknak lehetőségük van anoním módon szövegesen véleményezni az adott kereskedőt, vagy árucikket, továbbá 1-től 5 csillagig osztályozni annak minőségét.

A weboldalon található véleményekhez egy saját kezűleg fejlesztett *webscraper* segítségével jutottam hozzá. A letöltött halmaz mérete összesen 943 MB, amely 141 064 termék és 2209 áruház értékelését tartalmazza.

A 2 006 369 darab vélemény eloszlása a csillagok száma szerint a következő:



(a) Összes

(b) Legalább 10 token hosszú

3.1. ábra. Vélemények eloszlása

A leghosszabb vélemény 842 tokenből áll. A túl rövid értékelések kevés információval szolgálnak, így az eredeti adathalmazból a 10 tokennál rövidebb bejegyzéseket eltávolítottam.

Az így kapott adatsor átesett a feljebb említett előkészítési lépéseken. Ezen felül a 3 csillagos "semleges" véleményeket eltávolítottam, az 1-2 csillaggal rendelkező értékelések "negatív", a 4-5 csillaggal rendelkezők pedig "pozitív" címkét kaptak.

A halmazban a "pozitív" címkéjű vélemények szerepeltek túlnyomó többségben. A kiegyensúlyozatlansági probléma kivédésére több elemet töröltem a "pozitív" címkéjű elemek közül, így a két csoport egyenlő elemszámmal szerepel. Az előállított auto-annotált halmaz 94982 adatponttal rendelkezik, mindegyik osztály 47491 méretű.

A *velemenyek*-nek elnevezett adathalmaz alkalmas lehet a szöveg érzelmi tartalma szerinti bináris klasszifikáció kivitelezésére.

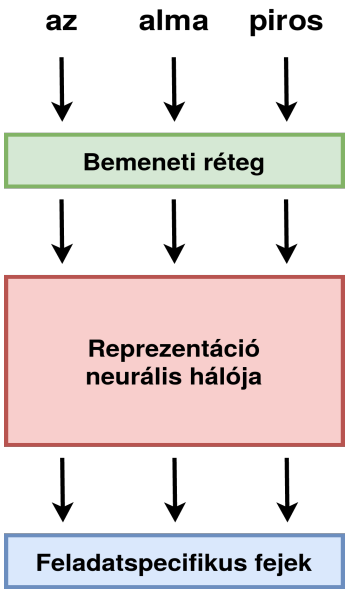
4. fejezet

Egy magyar nyelvű reprezentációs algoritmus

A szemantikus reprenzentációs módszerek kutatása intenzíven felgyorsult az elmúlt évtizedben. Bár a szélesebb körben beszélt nyelvek esetében – például angol, kínai – számos technika és adathalmaz is elérhető, a kis és közepes nyelveknek egyelőre nélkülözniük kell ezeket. A probléma feltehetőleg részben a kutatási terület újszerű jellegéből, részben pedig a nagy tömegek igényeinek hiányából fakad.

Ismereteim szerint magyar nyelven a tárgyalt kategóriák közül kizárólag szóbeágyazási modellek léteznek – mint a FastTex [20], Word2Vec és az ELMo – , továbbá a lehetséges tanítási feladatok is korlátozottak, így többnyire csak felügyelet nélküli tanítás kivitelezhető. A diplomamunkám során megoldandó feladat egy magyar nyelvű mondat/paragrafus szintű nyelvi modell elkészítése, amely alapjául az előzményekben megismert módszerek szolgálnak. Továbbá olyan autoannotált adathalmazok létrehozása, majd vizsgálata, melyeket a tanítási folyamathoz használok fel. Az így kapott előre tanított nyelvi modell reményeim szerint alkalmas lesz a későbbi NLP feladatokhoz szükséges finomhangolásra, továbbá a tanításhoz használt algoritmusok más munkák segítségére is lehetnek.

A feladat megoldására szolgáló módszer alapvetően három részből áll: a bemeneti rétegből, a reprezentáció létrehozásához használt neurális hálóból és a modell tanításához definiált feladatokból, az ehhez alkalmazott fejekből. Mivel a szóalapú megoldások általában pontosabb eredményt mutatnak, mint a karakter, vagy szótöredék alapú modellek, így ebben az esetben is szavak kerültek feldolgozásra.



4.1. ábra. A módszer magasszintű architektúrája

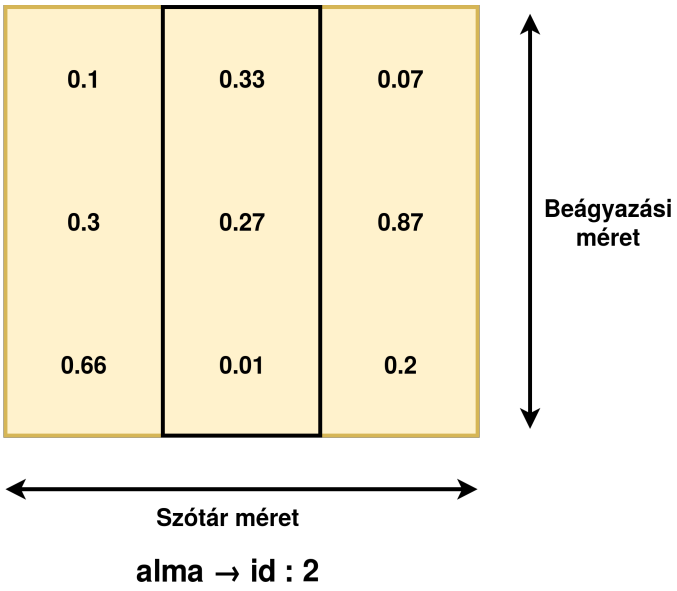
Az implementációt Python nyelven végeztem a Tensorflow [21] nevű könyvtár segítségével.

4.1. Az architektúra bemeneti rétege

Ahogyan több módszer esetén is láthattuk, előfordulhat, hogy a neurális hálók bemenetére már eleve vektorizált formában érkeznek a tokenek. A feladat megoldásához használt architektúrában az input koordinálását egy bemeneti réteg végzi. Ezen réteg konfigurálható attól függően, hogy az inputra a tokenek enkódolt formában érkeznek, vagy az algoritmus a számára megadható szóbeágyazási modellt használja. Ha a tokenek nem vektor formájában kerülnek a bemenetre, akkor a bemeneti réteg a tokenekhez rendelt egyedi azonosító számok szekvenciáját fogadja.

A tanítás során minden esetben a mélyháló számára megadott Word2Vec - CBOW szóbeágyazási modelleket használtam, melyek 300 dimenziós szóvektorokat tartalmaztak. A szóbeágyazások tanítóalgoritmusa 5 token széles ablakot alkalmazott. Az előtanítás során átadott reprezentációs mátrixot a GPU a memóriában tárolja és műveleteket is végez vele, ezért a nagyobb, oscar_hu halmazon tanított modell rendkívül memóriaigényesnek, továbbá a wiki_hu tanítóhalmaz – a Word2Vec tanításához – túl kisméretűnek bizonyult. Így a végső választás az oscar_sm modell-

re esett, ami az oscar modell kisebb méretű szótárral rendelkező változata. A wiki 658 129, az oscar modell 2 335 673, az oscar_sm pedig 645 136 darab vektorból áll.



4.2. ábra. Az embedding lookup

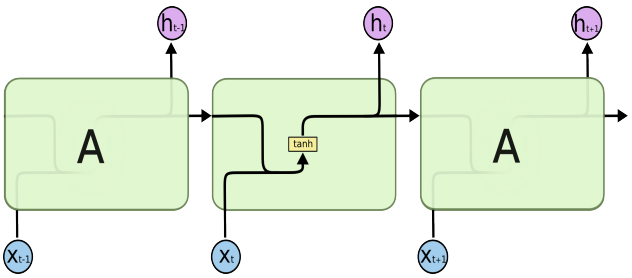
A bemeneti réteg fix súlyokkal rendelkezik, tehát a tanulási folyamat során nem változtatja azokat. A réteg a számára átadott beágyazási mátrix elemei közül kikeresi az azonosítóknak megfelelő elemeket, majd továbbítja őket a kimenetre. Ezt a folyamatot *embedding lookup*-nak nevezzük.

4.2. A reprezentáció neurális hálója

A rekurrens neurális hálók (RNN) használata a szemantikus reprezentációs modellek esetén gyakori technika. Míg a mesterséges neurális hálók (ANN) csak önálló bemenet fogadására képesek, addig a rekurrens neurális hálók alkalmasak szekvenciális input feldolgozására is. Ilyen szekvencia például az idősori, vagy a szöveges adat is. A szekvenciális bemenetet az a tulajdonság különbözteti meg önálló bemenettől, hogy az input elemei függenek egymástól, hatással lehetnek a szomszédaikra. Több önálló input esetén ez a reláció nem érvényes.

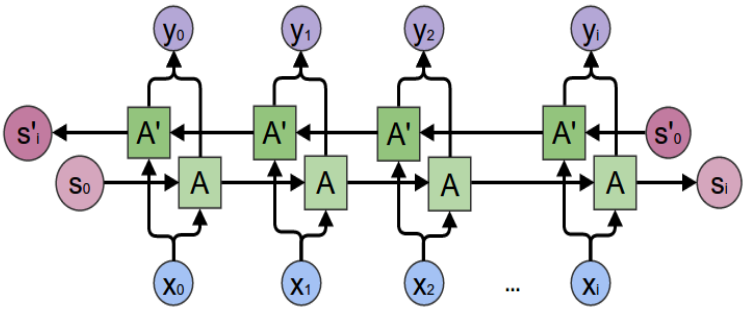
A rekurrens neurális hálók képesek megtanulni a szekvencia elemei közötti kapcsolatokat. Az RNN a tanulási folyamat során "emlékszik" az előző elemektől gyűjtött információkra, majd azok segítségével generálja a kimenetet/kimeneteket. A

számítás során használt vektorokat nemcsak az input befolyásolja, hanem a rekurrens háló rejtett állapotvektorai is. A rejtett állapot megtanulja a folytonos bemenet elemei közti függőségeket, majd minden tanítási lépés során frissül. Ennélfogva minden egyes bemeneti elem más és más műveleten esik át.



4.3. ábra. Az RNN cella [22]

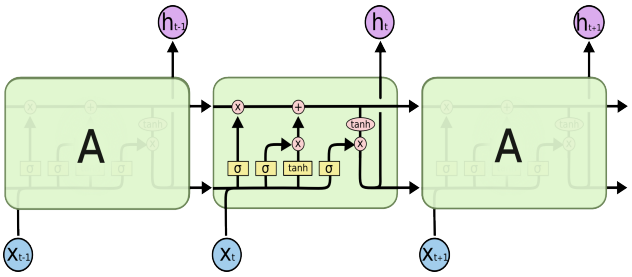
Bizonyos esetekben, ahol a múltból származó információ elegendő lehet a háló számára – például következő token generálása az előzőek függvényében – , az egyszerű RNN jó opció lehet. Azonban olyan feladatok során, melyeknél fontos a bemeneti adatok kontextusa – például a nyelvi modellek – , más megoldásra van szükség. A BiRNN architektúra lényege, hogy az inputot két, egymással ellentétes irányú rekurrens háló olvassa. Az így kapott kimeneti vektorok páronkénti konkatenációja lesz a BiRNN output-ja.



4.4. ábra. A BiRNN architektúra [23]

Az RNN-ek legegyszerűbb formájának (*Vanilla RNN*) azonban van egy nagy gyengesége, ami a hosszútávú információkat illeti. Gradiensnek hívjuk azokat az értékeket, melyeket a háló a súlyai frissítésére használ. Vanilla RNN esetén a visszaterjesztési művelet (*backpropagation*) alatt annyira lecsökkenhetnek az eleve túl kicsi gradiensek, hogy a hozzá tartozó rétegek megállnak a tanulásban. Ezt a problémát *vanishing gradients* problémának hívjuk.

Az LSTM (*Long short-term memory*) architektúra megoldást nyújt a *vanishing gradients* problémára. Az LSTM a megszokott hosszútávú memória mellé bevezeti a rövidtávú memóriát is. Olyan belső műveletei vannak, melyek képesek szabályozni az adott cellán belüli információáramlást. Ezen műveleteket kapuknak nevezzük. A kapuk eldönthetik, hogy mely információ lesz fontos a továbbiakban és melyiket lehet törölni. Így a módszer csak releváns információt enged a hosszútávú memóriába.

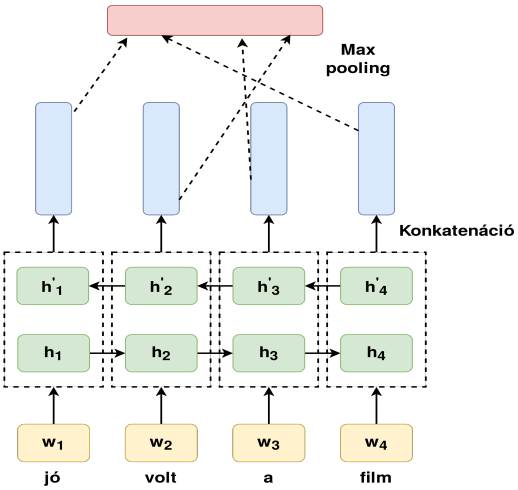


4.5. ábra. Az LSTM cella [22]

Az általam a feladat megoldására választott architektúra az InferSent-ben kiváló eredményeket prezentáló BiLSTM max pooling.

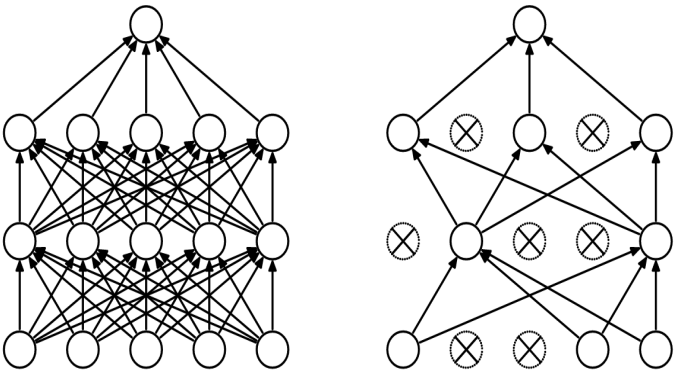
4.2.1. A BiLSTM réteg szerkezete

A BiLSTM egy kétirányú rekurrens neurális háló (BiRNN), amely LSTM cellákat használ. Az NLP feladatok természetes nyelven írott szöveggel operálnak, így a rekurrens neurális háló az egyik alternatíva a változó hosszúságú szekvenciális adatok feldolgozására. A kétirányú modell figyelembe veszi a feldolgozandó token kontextusát a tanulás során és eltárolja a sorrendi információkat is. Az LSTM cellák alkalmazása széles körben elterjedt technika, amely amellett, hogy képes kezelni az RNN gyengeségeit, a jelenlegi egyik legpontosabb megoldásnak bizonyul.



4.6. ábra. A BiLSTM max pooling architektúra

A sűrű rétegek tanítása közben az egyes neuronok között kialakulhatnak keresztfüggőségek, így túltanulhat a modellünk az adott adathalmazra. A *dropout* egy olyan regularizációs technika, amely kikényszeríti, hogy az egyes neuronok önállóan tanuljanak, így véd a túltanulás ellen és a neurális háló is jobban fog generalizálni. A tanítási fázis során az összes iteráció, összes *batch*-e esetén minden neuron és hozzá tartozó aktiváció $1 - p$ valószínűséggel véletlenszerűen kidobásra kerül. A teszt fázis alatt az összes neuron cselekvőképes, de az aktivációkat a helyes működés miatt p állandóval szorozni kell. Bár a tanítási idő minden *epoch* során kevesebb lesz, a *dropout* körülbelül megduplázza a konvergációhoz szükséges iterációk számát. A reprezentáció tanulására szolgáló neurális háló mindkét LSTM rétegére konfigurálható *dropout*-ot alkalmaztam.



4.7. ábra. A dropout vizualizációja [24]

A BiLSTM réteg által generált szekvenciális kimenet egy *pooling* rétegbe vezet.

4.2.2. Pooling technika

A *pooling* ötletét szintén a számítógépes képfeldolgozás ágazatától kölcsönözte az NLP. Míg a konvolúciós rétegek esetén a *feature map*-ek kisebb szegmensein elvégzendő a *pooling* művelet, addig az NLP-ben vektorokra értendő. A *max pooling* réteg a kapott bemenet megadott tengelyei mentén választja ki a legnagyobb értékeket. Analóg módon a *mean pooling* az átlagos értékeket veszi alapul.

A BiLSTM-ben található rejtett rétegek kimeneteinek páronkénti konkatenációján végzett pooling művelet képes kiválasztani a hasznos információkat az egyes tokeneket/részszekvenciákat reprezentáló vektorokból. Az így kapott sorvektor lesz a szöveges bemenet végső reprezentációja.

A nyelvi modell neurális hálójának implementációja egy konfigurálható *pooling* réteget tartalmaz, így a megoldás *max* és *mean pooling*-gal, továbbá pooling nélkül is képes dolgozni.

4.2.3. Paraméterek és konfigurálhatóság

A módszer neurális hálójának implementációja során törekedtem a minél széleskörűbb konfigurálhatóságra, így elősegítve a könnyebb testreszabhatóságot és az újrafelhasználhatóságot.

A modellre vonatkozó paraméterek a következők:

- *use_embedding_layer* : Alkalmazzon a háló bemeneti réteget, vagy vektorizált az input.
- *word_embedding_dim* : A kapott szóbeágyazási vektorok mérete.
- *num_hidden* : A végső reprezentáció mérete, a rejtett LSTM rétegek méretének kétszerese.
- *dropout_keep_prob* : Dropout valószínűség.
- *pooling* : *Pooling* fajtája. (Max, Mean)

A neurális háló paraméterezhetősége lehetőséget biztosít arra, hogy az architektúra más típusú bemenettel, más célra is felhasználható legyen. Ezen konfigurációkon felül több, a tanítási folyamathoz kapcsolódó érték is állítható.

4.3. A nyelvi modell tanítása

Bár az alkalmazott architektúra teljesítménye jelentősen hozzájárul a szemantikus reprezentációs modellek pontosságához, az elmúlt néhány év során mégis inkább a tanítóhalmazok és a tanítási módszerek felé irányult a figyelem.

A neurális hálóknak a korábbi hagyományos, feladatspecifikus tanítás alatt egyszerre kellett megérteni a dokumentumhalmaz nyelvét és megtanulni az adott feladathoz szükséges ismereteket. A *transfer learning* technika segítségével ez a két folyamat különválasztható. Az előtanulás során feldolgozott nagy mennyiségű adat hatására a háló megragadja a dokumentumok nyelvi sajátosságait, így a finomhangolás alatt a modell koncentrálhat csak az adott feladatra. Az eredmény legtöbbször egy jobban működő reprezentáció.

Az előtanítási feladatok olyan jellegű kihívások elé állítják a reprezentáció neurális hálóját, amelyek során egy erős, általános tudást képes megszerezni. A probléma megoldására implementált feladat a BERT előtanításához hasonló *multi-task learning*.

A *multi-task learning* egyszerre több feladaton történő tanulást jelent, ami jobb generalizációra készteti a hálót. Az egyik feladat a **maszkolás**, amely az egyes szavak közötti szemantikai és szintaktikai relációk ábrázolását támogatja. A másik feladat a **következő mondat**, mely a mondatok közötti kohézió reprezentációját segíti.

Mivel az említett feladatok számára tanítóadatot bármely célnyelvű korpuszból könnyedén lehet generálni, ezért elméletileg az előtanítás a végtelenségig skálázható. Az input előállítása során a *Hungarian Webcorpus* nevű adathalmazzal dolgoztam, mivel az jelentős mennyiségű szöveges adatból áll, és a mondatok az *oscar_hu* adathalmazzal ellentétben sorrendtartóak.

A generálás alatt az algoritmus rögzített tokenszámú szekvenciákra osztotta a korpuszt. Olyan esetben, mikor nem volt elég token a szekvencia kitöltésére – például rövid dokumentumok, dokumentumvégek – , "[PAD]" speciális helykitöltő *padding* tokeneket konkatenált a dokumentum szavaihoz.

A maszkolás a BERT-ben alkalmazott technika alapján történt: az algoritmus az előállított tokenszekvenciák elemeinek 15%-át véletlenszerűen kiválasztotta.

- A kiválasztott tokenek 80%-át a "[MASK]" speciális tokennel helyettesítette.

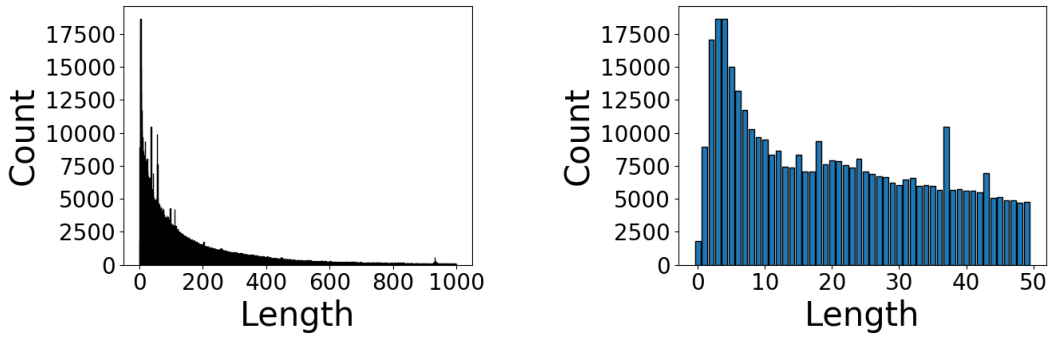
- 10%-át a szótárból választott véletlenszerű szóra cserélte.
- A maradék 10% esetében maradt az eredeti token.

A BERT szerzője szerint, ha 100% maszkolt tokent prediktálna a rendszer, akkor nem feltétlen lenne képes megfelelő minőségű reprezentációt generálni a nem maszkolt szavaknak. Ha a kiválasztott tokenek 90%-át maszkolná és 10%-ot random választott szóval helyettesítene, az arra késztetné a modellt, hogy úgy gondolja, az adott szó soha nem helyes. Ha pedig a kiválasztott tokenek 90%-át maszkolná és 10% maradna az eredeti szó, a modell lemásolná a nem kontextusfüggő beágyazásokat. [25]

A *következő mondat* feladathoz szükséges "mondatpárokat" az így előállított tokenszekvenciákon végigiterálva generálta az algoritmus. S szekvenciát 0.5 valószínűséggel S után következő szekvenciával, 0.5 valószínűséggel a korpuszból választott véletlenszerű szekvenciával konkatenálta. A két szekvencia közé egy speciális "[SEP]" mondathatárt jelző tokent illesztett. A végső adathalmaz az így kapott A[SEP]B alakú maszkolt, fix méretű szekvenciákból álló halmaz, mely mérete 13.3 GB volt.

Megjegyzés. *A "mondat" szó ebben az esetben tokenek tetszőleges méretű sorozatát jelenti.*

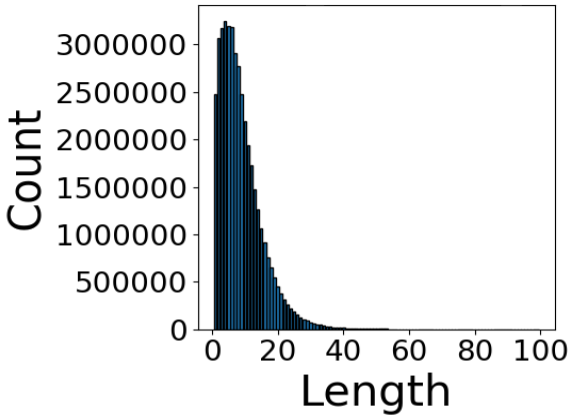
A bemenet előállításához szükség volt arra az információra, hogy maximum hány token hosszúságú szekvenciákra darabolja az algoritmus a dokumentumokat és mi a minimális soronkénti tokenszám, amelyet még elfogadjon.



(a) Az 1000 token alatti dokumentumok száma

(b) Az 50 token alatti dokumentumok száma

4.8. ábra. Dokumentumok hosszának eloszlása



4.9. ábra. A Hungarian Webcorpus-ban található 100 token hosszúság alatti mondatok hosszának eloszlása

A dokumentumokra és a dokumentumok soraira vonatkozó mérések alapján minimális tokenszámnak 10-et, maximális szekvenciahossznak 100-at határoztam meg. A cél az volt, hogy a lehető legtöbb információt sikerüljön kinyerni a dokumentumokból és csak relatíve kevés esetben kelljen *padding*-et alkalmazni, továbbá ne legyen a konfigurációnak teljesíthetetlen memóriaigénye. A végső bemeneti hossz így 201 token lett.

Bemeneti réteg használata esetén a neurális háló inputjára a mondatpárok tokenjeinek Word2Vec azonosítója kerül. Az *embedding lookup* művelet ezen azonosítók segítségével választja ki a réteg számára átadott beágyazási mátrixból az adott tokenekhez tartozó szóvektorokat. Felmerülhet a kérdés, hogy mi történik a speciális tokenekkel ebben az esetben. A speciális tokeneket a Word2Vec modell nem ismeri, így azok nem kerültek be a beágyazási mátrixba sem. A probléma megoldására bevezettem a meglévő Word2Vec dimenziók mellé minden egyes speciális token számára egy saját dimenziót és hozzá tartozó azonosítókat. Ezen tokeneket reprezentáló vektorok kiterjedése a saját dimenziójukban 1, az összes többi dimenzióban 0. Így a speciális vektorok merőlegesek a többi vektorra. Ennek okán a speciális tokenek elméletileg nem befolyásolják a szemantikai információkat a tanulás során. A bemeneti rétegnek átadott beágyazási mátrix mérete – az [UNK], Word2Vec modell által nem ismert szavakat jelölő tokennel együtt – $(\text{word\_embedding\_dim} + 4) \times$ szótár méret volt.

A beágyazási mátrixban tárolt vektorok hosszának felhasználhatóságáról meg-

oszlanak a vélemények. Egyesek szerint a hosszok eltéríthetik a modellt és bizonyos feladatok esetén hatékonyabban működhetnek az azonos hosszú vektorok, így a tanítás során mindkét lehetőséget – azaz normálva és normálás nélkül is – kipróbáltam.

Bár a *maszkolás* és a *következő mondat* feladatok a *multitask learning* során megosztják egymással a bemeneti réteget és a reprezentáció tanulásához szükséges hálót, a feladatok megoldásához használt fejek és a feladatspecifikus bemenetek különböznek.

4.3.1. Maszkolás feladat

A maszkolási feladat alatt a fej célja kitalálni, milyen tokenek voltak a [MASK] tokenek helyén. A feladat abszolválásához a fejnek szüksége volt a mondatpárok generálása során létrejött egyéb információra is. Ilyen információ a maszkolt tokenek szekvencián belüli pozíciója, Word2Vec azonosítója és súlya. Mivel a [MASK] tokenek megfelelő eloszlása érdekében az algoritmus a [PAD] tokeneket is letakarta, ezért fontos tényező volt a súlyok bevezetése. Egy maszkolt token súlya 0, ha a token eredetileg [PAD] token volt, egyébként 1.

A fej a BiLSTM háló *pooling* nélküli, szekvenciális kimenetével dolgozik. A kimenetből kiválasztja a maszkolt szavak reprezentációit. Az így kapott *num_hidden* hosszú vektorokat egy sűrűn kapcsolt, GELU aktivációval [26] ellátott rétegen vezeti át, melynek kimeneti mérete megegyezik a kibővített Word2Vec modell beágyazási dimenziójával. Majd a sűrű rétegtől kapott kimenetet réteg normálja [27] és megszorozza a beágyazási mátrixszal, annak érdekében, hogy az egyes vektorok a szótár dimenziójába kerüljenek, továbbá *log softmax* aktivációt alkalmaz, hozzájutva a maszkolt tokenekre vonatkozó valószínűségek logaritmusához a teljes szótárra nézve. A feladat tanítási *loss* függvénye az egyes eredeti maszkolt tokenekre vonatkozó logaritmusok összegének ellentettjének átlaga.

4.3.2. Következő mondat feladat

A *következő mondat* feladat során a fejnek ki kell találnia, hogy A[SEP]B bemenet esetén B szekvencia A rákövetkezője-e. A tanítás során a fej számára biztosítani kell a szekvenciákhoz tartozó címkéket. Ezen címkéket a bemeneti adathalmazt elő-

állító algoritmus generálja. Előfordulhat olyan eset, mikor az algoritmus nem tud az adott dokumentumból A mondat mellé B mondatot választani, mert a dokumentum túl rövid, vagy éppen dokumentumhatárra ért. Ekkor értelemszerűen minden esetben hamis lesz a címkéje az adott bemeneti szekvenciának. Ez a működés kiegyensúlyozatlansági problémához vezethet, melyet az algoritmusban alkalmazott számláló hatékonyan ki tud védeni. Ha többségbe kerül a negatív esetek száma, az algoritmus 1 valószínűséggel pozitív eseteket generál.

Megjegyzés. *A jobb megértés érdekében a dolgozathoz mellékeltem egy példa bemenetet. [A]*

A fej a BiLSTM háló *max pooling* utáni vektoriális kimenetével, tehát a mondatokat reprezentáló vektorokkal dolgozik. A mondatvektorokra klasszikus bináris klasszifikációt alkalmaz, azaz egy sűrűn kapcsolt rétegen vezeti át őket, melynek kimenete 2 széles és az aktivációs függvénye a *log softmax*. A feladat tanítási *loss* függvénye az így kapott logaritmusok összegének ellentettjének átlaga lesz.

A nyelvi modell tanítása - folytatás

A *multitask learning* során a neurális háló egyszerre két feladatot próbál megoldani, így jobban generalizál. A háló a GD (*Gradient Descent*) nevű algoritmus segítségével minimalizálja a globális *loss* függvényt, amely a két feladat *loss* függvényének összege. A GD számára implementáltam egy konfigurálható *decay* algoritmust. A *learning rate decay* a *loss* függvény értékének *epoch*-onkénti túl lassú csökkenése esetén öttel osztja a *learning rate*-et, így elérve a jobb konvergációs képességet. Az algoritmus minden *epoch* során menti a *transfer* számára szükséges súlyokat és a tanítás aktuális állapotát is.

A jól paraméterezhető tanítás érdekében az implementáció során törekedtem a minél széleskörűbb konfigurálhatóságra. A következő tanításra vonatkozó értékek állíthatók:

- *batch_size* : A tanításhoz használt *minibatch*-ek mérete.
- *num_inputs* : Az input vektorok mérete. Word2Vec azonosítók esetén 1.
- *num_time_steps* : A bemenet szélessége.

- *min_sentence_length* : A legkisebb tokenszám amit az algoritmusnak figyelembe kell vennie.
- *max_sentence_length* : Az egyes mondatok *padding* utáni hossza.
- *masked_lm_prob* : A tokenek maszkolási valószínűsége.
- *max_predictions_per_seq* : A legnagyobb maszkolható tokenszám egy mondatpár esetén.
- *learning_rate_start* : A *learning rate* kezdőértéke.
- *lr_decay* : *Learning rate decay* ki/be.
- *lr_decay_threshold* : Az előző és az aktuális *epoch loss* érték mekkora különbsége estén ossza le a *learning rate*-et öttel. Az alapérték 0, tehát ha magasabb, vagy ugyan akkora az aktuális *epoch loss* érték, mint az előző.
- *epochs* : Az adathalmazon történő iterációk száma.
- *mask_padding* : Maszkolja-e a *padding* tokeneket.

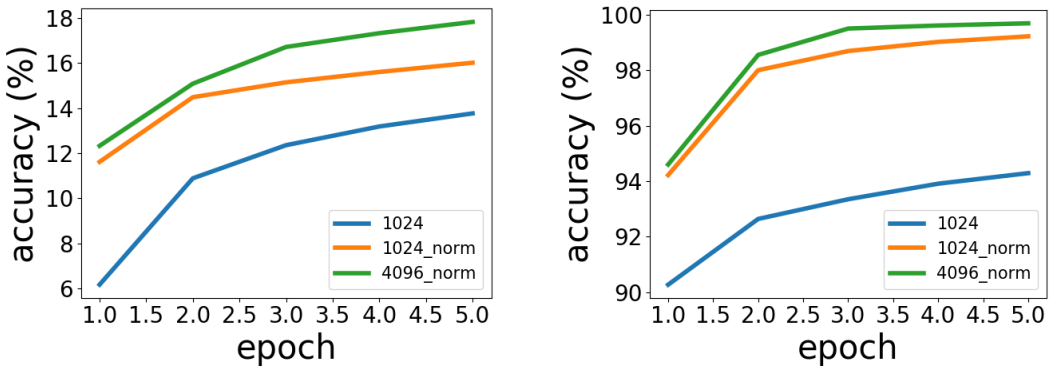
A futtatáshoz használt szervergép konfigurációja a következő: 2 x Intel®Xeon®Processor E5-2640 v4 25M Cache 2.40GHz (10 mag) CPU, 2 x NVIDIA GeForce GTX 1080 Ti GPU, 4 x 32 GB DDR4 RAM.

A neurális modellek kezdeti, teljes adathalmazon történő tanítása *epoch*-onként 4 napot vett volna igénybe, így a generált halmaz méretét a negyedére csökkentettem. Minden modell tanítása esetén 32 méretű *batch*-et és 0.1 kezdeti *learning rate*-et használtam, a *dropout* valószínűsége 0.5, az *epoch*-ok száma 5 volt.

Méret	**Normált bemenet**	**Tanítási pontosság**	
		Maszkolás	Következő mondat
1024	Nem	13,76%	94,29%
1024	Igen	16,01%	99,21%
4096	Igen	17,83%	99,69%

4.1. táblázat. A különböző méretű és bemenetű modellek tanítási pontossága az egyes feladatokra az utolsó epoch után

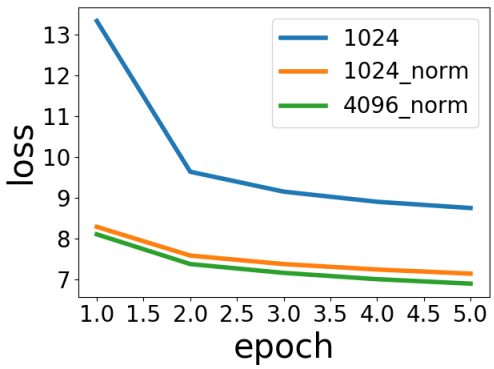
Ahogy a fenti táblázat mutatja, a normált bemenettel való tanítás jelentősen emelte a modellek tanítási pontosságát, továbbá a rejtett méret növelése is további javuláshoz vezetett. A legjobb eredményt a 4096 széles rejtett réteggel rendelkező modell érte el, amely bemeneti rétege normált Word2Vec vektorokkal dolgozott. A modell tanítási pontossága az 5. *epoch* után a maszkolás feladatra 17,83%, a következő mondat feladatra 99,69% volt.



(a) Maszkolás feladat (b) Következő mondat feladat

4.10. ábra. A különböző modellek tanítási pontosságának alakulása

A 4.10-es ábrán látható, hogy a normált bemenetű modellek eleve magasabb tanítási pontossággal kezdtek, továbbá a modell méretének növelésével a konvergálás is hatékonyabbá vált.



4.11. ábra. A tanítási loss függvények alakulása a különböző modellek esetén

A 4.11-es ábra alapján elmondható, hogy normált bemenet esetén kisebb loss összeggel indult a tanítás és a nagyméretű modellnek sikerült elérni a 7 alatti értéket is.

4.4. Mondatvektorok generálása a nyelvi modell segítségével

A szemantikus reprezentációs vektorok felhasználási területe igen széleskörű. Többek között alkalmazhatóak írott szöveg összegzésére, dokumentumok keresésére, chatbot-ok implementálására, dokumentumok szemantikus hasonlóságának mérésére és szemantikai tartalom szerinti ajánlásra is.

Egy eleve betanított modell mentett súlyainak használata nem igényel komoly erőforrásokat és tárhelyet. A vektorok generálása egy egyszerű folyamat, mely során a reprezentáció fej nélküli neurális hálója – azaz a BiLSTM max pooling – ezen súlyok szerint inicializálásra kerül. Ezt követően a bemenetet a tanításához hasonló módon az input-ra juttatva a neurális modell előállítja a szövegrészletet reprezentáló vektort. A művelet során a háló súlyai nem változnak, azaz képtelen a további tanulásra.

A vektor generálás elvégzésére elegendő lehet egy CPU is, így a kellő pontosságot elérve a módszer akár ipari rendszerekbe is építhető, vagy különböző webes alkalmazások esetén felhasználói élmény fokozására is felhasználható.

5. fejezet

A módszer kiértékelése

A modern szemantikus reprezentációs algoritmusok célja, olyan univerzális és általános modellek előállítása, amelyek bármely rendszerben képesek igazodni az adott kihívásokhoz és megfelelő pontossággal teljesíteni az eléjük tűzött feladatokat.

A nyelvi modellek teljesítményének mérése nem triviális feladat. Bár a publikációk során a szerzők többnyire saját módszerrel mérnek, vannak már meglévő komplex kiértékelési keretrendszerek – például SentEval [28], GLUE [29] – és az igény is egyre nagyobb ezekre. A teljesítmény mérésére szolgáló rendszerek segítségével egységes képet kaphatunk a módszerünk pontosságáról, illetve a csalás lehetősége is korlátozott. A kiértékelési folyamat közben a reprezentációs modelleknek olyan feladatok sorozatát kell megoldaniuk, mint a vélemény-polaritás, bináris érzelmi analízis, következtetés vizsgálat, szemantikus hasonlóság.

Mivel a munkám során tanított modellek mindegyike magyar nyelvű, így nem használhattam ezen megoldásokat. Szükségem volt egy saját kiértékelési feladat implementálására.

5.1. Vélemények bináris érzelmi analízise

Az általam létrehozott modellek pontosságának mérésére használt feladatnak több szempont szerint is meg kellett felelni. Jó kiértékelési feladat az, amelyik kellőképpen "nehéz", így a modellek közti teljesítménybeli különbségek jól interpretálhatóak. Továbbá a feladathoz tartozó adathalmaznak elegendő elemet kell tartalmaznia ahhoz, hogy a feladat elkerülje a túltanulás problémáját és hiteles kimenetet kap-

junk végeredményül. A *velemenyek*-nek nevezett adathalmaz bináris osztályozása a vélemények érzelmi tartalma alapján megfelelő kihívásnak minősült ahhoz, hogy az általam tanított modellek pontosságát meg tudjam határozni.

A mérés során az annotált tanítóhalmaz elemeit – azaz pozitív vagy negatív osztályba tartozó értékeléseket – a meglévő modellek segítségével vektortérbe képeztem. Majd az így kapott vektor-címke párokkal tanítottam különféle osztályozó algoritmusokat. Ezen algoritmusok teszthalmazon történő kiértékelésének kimenete adta meg az adott szemantikus modell pontosságát.

Szükségem volt egy olyan viszonyítási alapra, amely a már létező magyar nyelvű módszerek teljesítményét szimbolizálja. Erre a célra az értékelések szavait az előre betanított *oscar* és *oscar_sm* Word2Vec modellekkel leképezve, majd átlagolva megkaptam a véleményeket reprezentáló 300 hosszú vektorokat. Mivel az értékelések változó méretűek, továbbá legalább 10 tokenből állnak, a BiLSTM modellek esetén csak az első 201 token került a bemenetre a vektorok generálása során. A túl rövid paragrafusokat a tanításhoz hasonlóan *padding*-eltem.

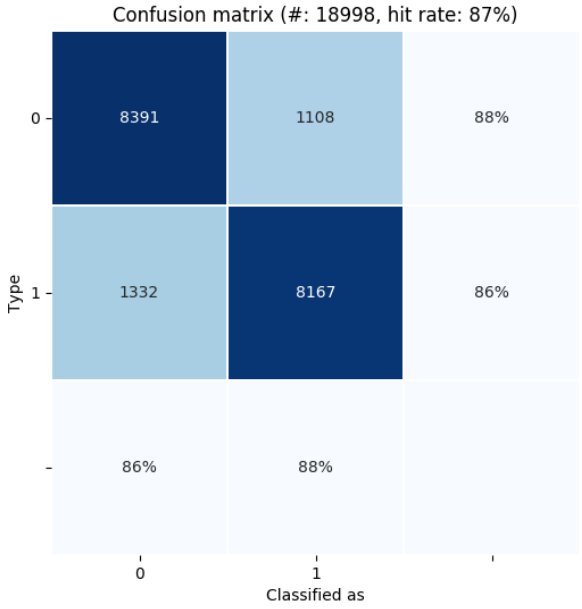
Egy bináris osztályozási feladat teljesítménye többféle metrika szerint is mérhető. Ezek közül a legnépszerűbb a klasszifikációs **pontosság** (*accuracy*), amely a helyesen prediktált elemek számának és az összes elem számának hányadosa. A **tévesztési mátrix** (*confusion matrix*) leírja és vizualizálja modellünk teljesítményét az igaz pozitívként, igaz negatívként, hamis pozitívként és hamis negatívként prediktált elemek számának segítségével. A **vevő működési karakterisztika** (*receiver operating characteristic* - ROC) görbe az igaz pozitív és a hamis pozitív arányok közötti kapcsolatot szemlélteti. Egy véletlenszerű modell görbéje a főátlón helyezkedik el, ahol a függőleges tengelyen az igaz pozitív, a vízszintes tengelyen a hamis pozitív arányok szerepelnek. Minél nagyobb a görbe alatti terület, annál jobb a modellünk performanciája.

Modell / Osztályozó	Linear SVM	XGBoost	Random Forest
w2v_sm	85,45%	82,18%	83,64%
w2v_sm_norm	85,57%	82,71%	**84,18%**
w2v_lg	85,28%	82,87%	83,77%
w2v_lg_norm	85,59%	83,15%	83,85%
lstm_1024	81,53%	80,82%	80,84%
lstm_1024_norm	85,37%	83,48%	83,41%
lstm_4096_norm	**87.16%**	**83,71%**	83,90%

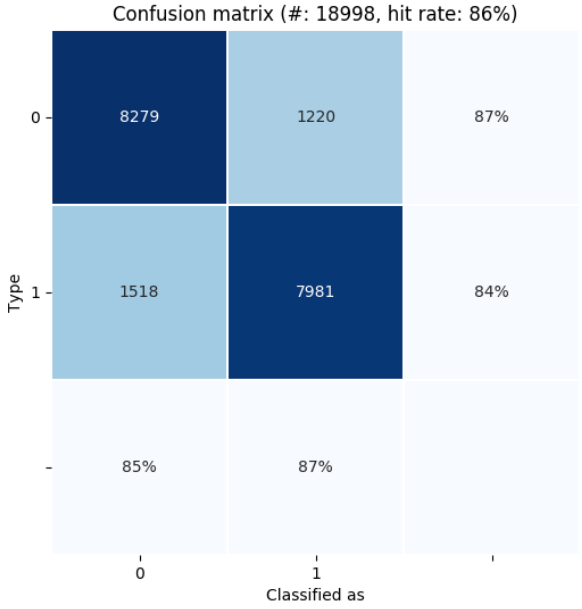
5.1. táblázat. A modellek klasszifikációs pontossága a velemenyek adathalmazon végzett bináris osztályozási feladat esetén. A w2v_sm az oscar_sm, a w2v_lg az oscar halmazon tanított modelleket, a norm posztfix a normált bemenetet jelöli. A számok a modellek nevében a reprezentációs vektor méretére utalnak.

Az 5.1-es táblázat a *velemenyek* adathalmazon mért klasszifikációs pontosságokat foglalja össze. A kiértékeléshez használt osztályozó algoritmusok a Linear SVM, XGBoost és Random Forest. A *w2v_sm* Word2vec modellt az *oscar_sm* halmazon, a *w2v_lg*-t az *oscar* halmazon tanítottam. Az *lstm* prefixű modellek a bemutatott módszerrel előállított nyelvi modellek.

A mért eredmények azt mutatják, hogy a munkám során ismertetett módszer esetén a bemeneti rétegnek átadott *w2v_sm* és a jelentősen nagyobb szótárral rendelkező *w2v_lg* teljesítménye közötti különbség elenyésző. Továbbá a normált bemenettel tanított modellek használata közben az algoritmusok nagyobb százalékban osztályoznak megfelelően. A feladatot legnagyobb pontossági értékkel abszolváló modell az *lstm_4096_norm*, amely alkalmazása során a helyesen klasszifikált elemek aránya 87,16% volt.



(a) lstm_4096_norm

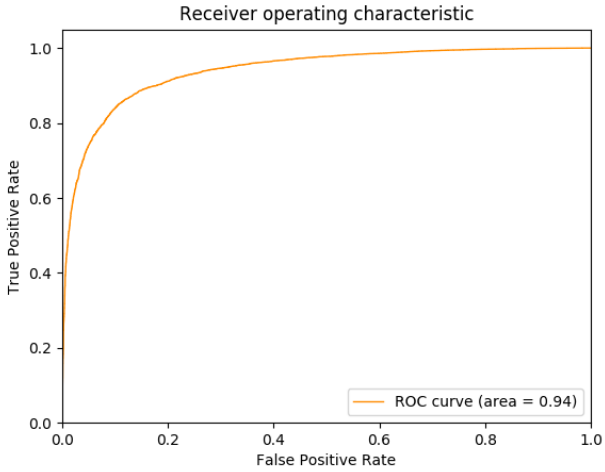


(b) w2v_lg_norm

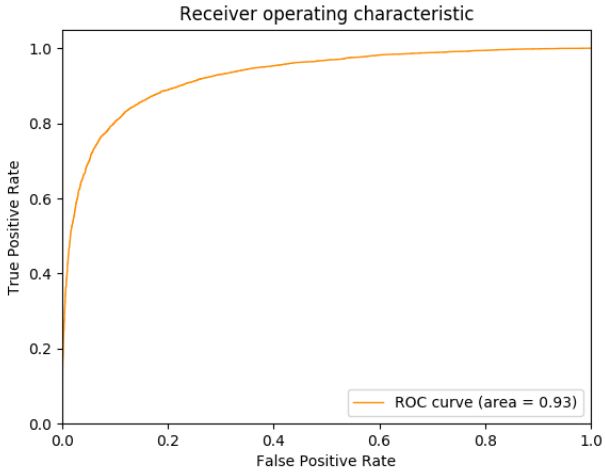
5.1. ábra. A bináris klasszifikáció során legjobban teljesítő LSTM és Word2Vec alapú modellek tévesztési mátrixa

Az 5.1-es ábra a legpontosabb általam bemutatott és Word2Vec alapú modell tévesztési mátrixát ábrázolja a Linear SVM oszályozó algoritmusra a bináris klasszi-

fikáció során. Az *lstm_4096_norm* modell segítségével az osztályozó 298 esetben helyes döntést hozott, míg a Word2Vec vektorok alkalmazása során helytelenül választott. Az igaz pozitív, igaz negatív, hamis pozitív, hamis negatív csoportok a két esetben relatíve kiegyensúlyozottak.



(a) lstm_4096_norm



(b) w2v_lg_norm

5.2. ábra. A bináris klasszifikáció során legjobban teljesítő LSTM és Word2Vec alapú modellek ROC görbéje

Az 5.2-es ábra a fent található tévesztési mátrixokhoz tartozó ROC görbéket illusztrálja. Az *lstm_4096_norm* modellre vonatkozó függvény görbe alatti területe nagyobb, mint a Word2Vec modell esetén, amely szerint az LSTM alapú reprezen-

tációs módszer segítségével a Linear SVM algoritmus pontosabb osztályozásra képes a bináris klasszifikációs feladat során.

Ugyan a kiértékeléshez használt módszer csak egy feladatra koncentrál, így egy bizonyos aspektus szerint méri a modellek teljesítményét, mégis viszonyítási alapként szolgál. A dolgozatban bemutatott reprezentációs algoritmus az általam vizsgált összehasonlítási szempontok alapján körülbelül másfél százalékkal pontosabb osztályozásra képes az említett Word2Vec modellekhez képest.

6. fejezet

Összegzés

A természetesnyelv-feldolgozás és annak ágazata, a szemantikus reprezentáció mind akadémiai, mind ipari értelemben gyorsuló ütemben fejlődő területek, amelyek a mai napig rengeteg felderítésre váró lehetőséget tartogatnak. A numerikus ábrázoló algoritmusok teljesítményének növekedése olyan gyakorlati alkalmazások pontosságának javulását indukálják, melyek segítségével hatékonyan kiszűrhető a gyűlöletbeszéd a szociális médiából, vagy akár eredményesen felvehető a harc az álhírek terjedésével szemben.

Diplomamunkám végeredménye egy kétirányú mondat- és paragrafusszintű előre tanított szemantikus reprezentációs modell, amely képes leképezni a magyar nyelven írt mondatokat a szemantikus térbe. Magyar nyelvű tanítóadat a nyelv beszéltségéből fakadóan nem, vagy csak elvétve elérhető. A prezentált neurális háló tanításához használt adathalmaz előállítása nem igényel emberi címkézést, olcsó és a végtelenségig skálázható. Továbbá a tanításra szolgáló feladatok jellegéből fakadóan, a reprezentációs algoritmus jó eséllyel átültethető más kis és közepes nyelvre is.

A munkám során több, különféle magyar nyelvű adathalmazt vizsgáltam, melyek akár Word2Vec modellek, akár a bemutatott nyelvi modell létrehozására is alkalmasak lehetnek. A dolgozatban ismertetett módszer tanítása alatt számos paraméterrel és beállítással próbálkoztam, majd a legjobban teljesítő jelöltek ábrázolási pontosságát kimértem egy kiértékelési adathalmaz segítségével, továbbá összehasonlító elemzéseket végeztem rajtuk. A vizsgált esetek közül a dolgozatban bemutatott módszer másfél százalékkal pontosabbnak bizonyult a Word2Vec-en alapuló megoldáshoz képest.

6.1. Továbbfejlesztési lehetőségek

A szemantikus reprezentációs módszerekben rejlő lehetőségek a mai napig ismeretlenek és feltérképezetlenek. A tanulás vagy vektorgenerálás során optimalizált paraméterbeállítások és kombinációk további teljesítménybeli javulást hozhatnak. A 4.10-es ábra alapján valószínűsíthető, hogy a 4096 méretű modell további tanítást követően magasabb pontossággal oldaná meg a maszkolási feladatot. A normált bemenetű modellek tanítási loss görbéi (4.11) pedig azt mutatják, hogy érdemes lehet kisebb *learning rate*-ekkel próbálkozni.

A jövőben javasolt lehet a *Gradient Descent* helyett más optimalizáló algoritmust választani, továbbá a mélyhálóban alkalmazott *max pooling*-ot *mean pooling*-ra cserélni. A megfigyelések alapján a GloVe szóbeágyazási módszer jobb reprezentációs képességekkel bír, mint a Word2Vec, így célravezető lehet a beágyazási rétegben GloVe modellt használni. Bizonyos esetekben a mélyebb architektúrák hatékonyabban tudják kinyerni a szekvenciális szöveges adatokból származó információkat. Ebből kifolyólag több egymásra illesztett BiLSTM, vagy akár BiGRU réteg együttes tanítása pontosabb végeredményhez vezethet.

Az általam létrehozott modellek mindegyike csupán előtanított, finomhangolást nem végeztem rajtuk. A *transfer learning* segítségével finomhangolt modellek több esetben jelentősen jobban teljesítenek, továbbá a folyamat adatigénye is kisebb az előtanításénál. Az elkövetkező időben a meglévő neurális modelleket egy olyan feladattal tervezem tovább tanítani, amely során a háló célja egy értelmező kéziszótárból kinyert cikkek alapján kitalálni a cikkekhez tartozó szót.

Jelen működés szerint a maszkolt szavak közül csak a *padding* tokenek kerülnek 0 súllyal a bemenetre. Ha sok ismeretlen szó található a tanítási halmazban, akkor a modellünk elfogulttá válhat az azokat jelölő token felé, így nehezebbé téve a tanulási folyamatot. Ennélfogva előfordulhat, hogy az ilyen tokenek nullával történő súlyozása jobb konvergációra készteti a modellt.

Összegzés - folytatás

A magyar nyelvű természetesnyelv-feldolgozás dolgozatom által érintett szegmense egyelőre meglehetősen kezdetleges állapotú, ennek köszönhetően számtalan lehetőséget rejt. A jövőben olyan alkalmazási területek is elérhetik a kívánt pontosságot, melyek ma még igencsak limitáltak, így a mobiltelefonunkon található személyi asszisztensünkkel akár magyar nyelven is kommunikálhatunk.

Bízom benne, hogy az általam tanításra használt feladatok és technikák segítségül lehetnek a további kis és közepes nyelveken létrehozandó szemantikus reprezentációs modellek fejlesztésénél.

6.2. Köszönetnyilvánítás

Végül, de nem utolsó sorban szeretnék köszönetet mondani témavezetőmnek, Grad-Gyenge Lászlónak, aki tanácsaival és szakértelmével hozzájárult diplomamunkám létrejöttéhez, továbbá Kincsőnek a rengeteg támogatásért.

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg (EFOP-3.6.3-VEKOP-16-2017-00002).

Megjegyzés. *A diplomamunkához mellékeltem a bemutatott módszer neurális hálójának implementációját. (model.py)*

A. függelék

Példa bemenet

tokens - bemeneti tokenek: *szerény termékeny magyar filmipar meglepő film tűni többnyire szociális probléma dráma foglalkozik műv színvonal átlag felüli tűni magyar filmgyártás [MASK] ambíció mozi hoz lét magyar filmkészítő megbecsül külföld [MASK] kései fejlődés magyarország film viszony késő tör stúdió jön lét mindkettő [MASK] indúlt fejlődés kezdetben irodalmi mű dolgozik [MASK] lévő osztrák magyar monarchia égisz es kommunista vezetés [MASK] [MASK] meglepő mód hónapos uralom film készül [MASK] kommunista uralom összeomlás horty kapillaritás [MASK] [MASK] kiemelkedő tehetség hagy [MASK] időszak kertész mihály michael curtiz fejős pál paul fejos balázs béla peter [MASK] lugosi [MASK] lukács pál paul [MASK] [MASK] év terroruralom honi filmgyártás [SEP] mély álom sűllyedt as [MASK] év német befolyás [MASK] felűlmúlta propaganda mennyiség napi parancs háború utáni neorealista mozgalom magyar kifejezésmód csodaszép alkotás devizaátutalás maga valahol európa amely árva [MASK] szenvedés keresztűl mutat háború pusztítás felnőtt okoz elítélnék borzalom önfejű tesz kommunista államosít filmipar ideológiai [MASK] lát szociális probla foglalkozó film [MASK] hazai filmgyártás eredmény virágzó filmipar [MASK] propaganda jellemez hullám as belezsúfolni rendezők [MASK] képes kialakít nyelvezet [MASK] képi szimbólum nemzeti téma ültet alkotás francia hullám eltérő magyar utasít próbál eltér hagyomány mag [MASK] rendezők kézdi kovács sándor pál sajátos stílus alakít mozi [MASK] állam elkötelezett filmfesztivál elért eredmény [MASK]*

input_ids - bemeneti Word2Vec azonosítók: *5382 12900 2 37008 2156 95 33811 2388 503 175 3356 561 7814 2660 2184 7771 33811 2 33898 645137 15313*

1521 157 557 2 40114 14740 1413 645137 16398 769 125 95 1321 3408 1890 2110 97 557 3391 645137 234617 769 7873 1930 799 220 645137 246 2188 2 8201 24238 498151 3877 946 645137 645137 2156 76 2825 5559 95 287 645137 3877 5559 10192 271054 628178 645137 645137 1481 2518 251 645137 420 5560 1075 4142 232007 59044 1002 6207 645139 1142 1111 6077 645137 35671 645137 5265 1002 6207 645137 645137 1 569799 19546 33898 645138 710 1182 645139 357 645137 1 259 5069 645137 534521 6988 743 453 3182 1216 1555 387909 2411 2 48872 9266 1074 586461 645139 2413 297 645139 9402 645137 3310 54345 154 1216 9095 625 530 645139 14327 40193 11 3877 43443 37008 11581 645137 45 503 645139 1923 95 645137 423 33898 137 9693 37008 645137 6988 2639 3143 357 588735 54795 645137 231 4395 14592 645137 4026 3558 122 230 5189 1074 701 3143 1203 2 6365 375 4907 1190 486 645137 54795 63938 1004 477 1002 2668 933 1373 1521 645137 730 4695 8935 3109 137 645137

masked_lm_ids - maszkolt szavak azonosítói: *12941 20703 726 150472 37008 43443 13246 271054 31 13246 18991 86 222962 1111 46797 357 498151 556 387909 74 104 3914 11009 1074 1 1935 526 1324 171 645*

masked_lm_weights - maszkolt szavak súlyai: *1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1*

masked_lm_positions - maszkolt szavak pozíciói: *19 28 40 47 56 57 64 68 69 70 71 75 88 90 94 95 105 109 117 123 129 145 151 157 162 164 168 184 194 200*

sentence_labels - következő mondat címke: *1*

Irodalomjegyzék

[1] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[3] Matthew E. Peters et al. "Deep contextualized word representations". In: *Proc. of NAACL*. 2018.

[4] Rami Al-Rfou et al. "Character-Level Language Modeling with Deeper Self-Attention". In: *CoRR* abs/1808.04444 (2018). arXiv: 1808.04444. URL: http://arxiv.org/abs/1808.04444.

[5] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv e-prints*, arXiv:1810.04805 (2018), arXiv:1810.04805. arXiv: 1810.04805 [cs.CL].

[6] Ryan Kiros et al. "Skip-Thought Vectors". In: *CoRR* abs/1506.06726 (2015). arXiv: 1506.06726. URL: http://arxiv.org/abs/1506.06726.

[7] Alexis Conneau et al. "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *CoRR* abs/1705.02364 (2017). arXiv: 1705.02364. URL: http://arxiv.org/abs/1705.02364.

[8] Samuel R. Bowman et al. *A large annotated corpus for learning natural language inference*. 2015. arXiv: 1508.05326 [cs.CL].

[9] Daniel Cer et al. “Universal Sentence Encoder for English”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 169–174. DOI: 10.18653/v1/D18-2029. URL: https://www.aclweb.org/anthology/D18-2029.

[10] Quoc V. Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*. 2014. arXiv: 1405.4053 [cs.CL].

[11] Hunspell. *hunspell/hunspell*. 2020. URL: https://github.com/hunspell/hunspell.

[12] Oroszgy. *oroszgy/spacy-hungarian-models*. 2020. URL: https://github.com/oroszgy/spacy-hungarian-models.

[13] *spaCy · Industrial-strength Natural Language Processing in Python*. 2020. URL: https://spacy.io/.

[14] *Natural Language Toolkit*. 2020. URL: https://www.nltk.org/.

[15] *Main Page*. 2020. URL: https://www.wikipedia.org/.

[16] Pedro Javier Ortiz Suárez, Benoît Sagot, and Laurent Romary. “Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures”. In: *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*. Cardiff, United Kingdom, July 2019. URL: https://hal.inria.fr/hal-02148693.

[17] 2020. URL: https://commoncrawl.org/.

[18] Péter Halácsy et al. “Creating Open Language Resources for Hungarian”. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC’04)*. Lisbon, Portugal: European Language Resources Association (ELRA), May 2004. URL: http://www.lrec-conf.org/proceedings/lrec2004/pdf/525.pdf.

[19] *Árukereső.hu*. 2020. URL: https://www.arukereso.hu/.

[20] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *arXiv preprint arXiv:1607.04606* (2016).

[21] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[22] *Understanding LSTM Networks*. 2020. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[23] *Neural Networks, Types, and Functional Programming*. 2020. URL: http://colah.github.io/posts/2015-09-NN-Types-FP/.

[24] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[25] Rani Horev. *BERT – State of the Art Language Model for NLP*. 2019. URL: https://www.lyrn.ai/2018/11/07/explained-bert-state-of-the-art-language-model-for-nlp/#appendix-A.

[26] Dan Hendrycks and Kevin Gimpel. "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units". In: *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415.

[27] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[28] Alexis Conneau and Douwe Kiela. "SentEval: An Evaluation Toolkit for Universal Sentence Representations". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://www.aclweb.org/anthology/L18-1269.

[29] Alex Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *CoRR* abs/1804.07461 (2018). arXiv: 1804.07461. URL: http://arxiv.org/abs/1804.07461.

Ábrák jegyzéke