Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek

Tanszék

Szemantikus reprezentáció magyar nyelv esetén

Témavezető:
Grad-Gyenge László
egyetemi tanársegéd

Szerző:
Kántor Attila
programtervező informatikus MSc

Budapest, 2020

Az eredeti szakdolgozati / diplomamunka témabejelentő helye.

Tartalomjegyzék

1. fejezet

Bevezetés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit [**dahl1972structured**]. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod.[1]

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus [**cormen2009algorithms**, **krasner1988mvc**]. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. **dijkstra1979goto** praesent eu consequat purus [**dijkstra1979goto**].

[1] Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

2. fejezet

Előzmények

Ahogy a nyelvet is szétválaszthatjuk elemeire – például lexéma (szó) , szintagma (szószerkezet) , mondat – , úgy a nyelvi elemeket reprezentáló módszereket is csoportosíthatjuk. Ugyan a nyelvi elemek és a közöttük található nyelvtani kapcsolatok matematikai ábrázolására való törekvés már az előző évszázad közepén megjelent, valódi eredményt csak az elmúlt egy-két évtized tud felmutatni. Az idő során a különböző nyelvi elemek reprezentációs módszerei párhuzamos módon fejlődtek, de 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 értelmet is 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 a neurális hálónkat könnyedén félrevezetheti. A természetes nyelven írott szöveg szavai között csak ritkán találhatunk rendezést. A szótár keresést alkalmazva neurális modellünk fontosabbnak ítélheti azon szavakat, melyek nagyobb azonosítóval rendelkeznek, így ebben az esetben 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 dokumentumok halmazában 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 következő szó 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. A modell a Bag Of Words (BOW) modellen alapszik, mely lényege, hogy L szótár esetén egy adott $d \in D$ dokumentumot egy $v \in \{0,1\}^{|L|}$ 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 adatok 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 esetén, 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 Euklideszi 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 modellünk 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 feldolgozni ezen relációs információt a lexémák között.

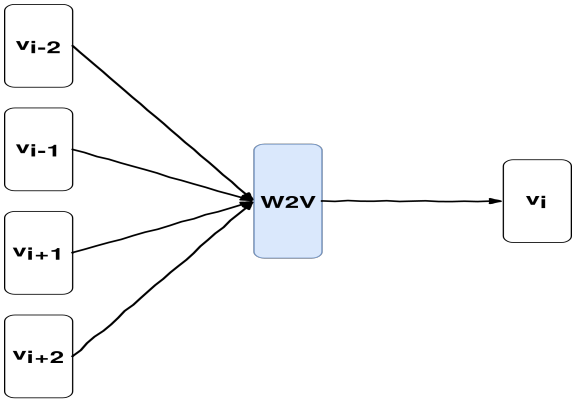
Word2vec

A Word2Vec 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 token-ekre bontották és one-hot kódolták. Majd a szöveg minden egyes token-jén végigiterálva a következőket hajtották végre:

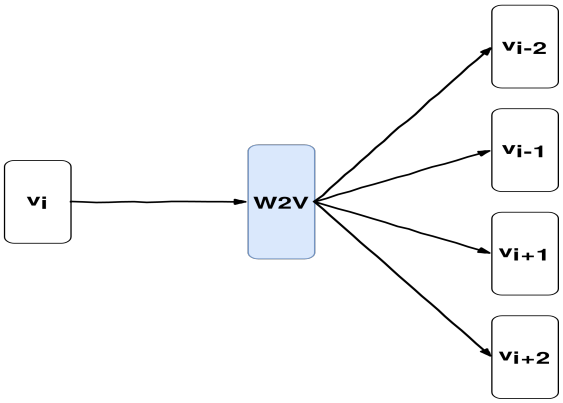
A CBOW modell szerint a háló bemenete v_i $(i \in |D|)$ vektorra a v_i vektor k méretű kontextusa $(v_{i-k}, ..., v_{i-1}, v_{i+1}, ..., v_{i+k} : k \in \mathbb{N})$, azaz a környezetében lévő

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 szóvektorokat közel re képezi egymáshoz a vektortérben. A teljesítmény növelése 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*) 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 építenie a mátrixot, mely költséges művelet.

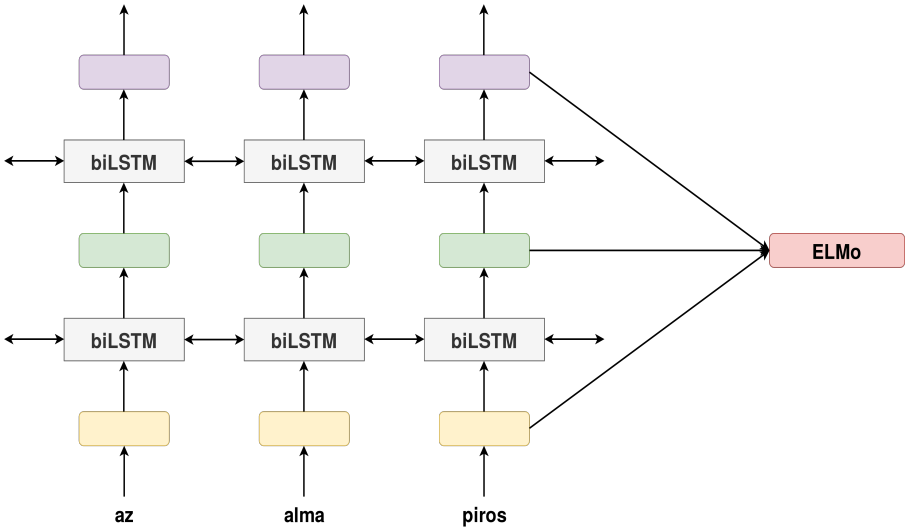
ELMo

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.

Míg a Word2Vec és a GloVe csak szavankénti kontextusfüggetlen reprezentáció tanulására képes – azaz nem számít az adott szó környezete, melyre alkalmazzák –, addig az Embeddings from Language Models (ELMo) figyelembe veszi a lexémák

kontextusát, mondaton belüli elhelyezkedését is. Az ELMo 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 eltárolni.



2.3. ábra. ELMo modell

Az ELMo a különböző rétegek kimenetének feladatspecifikus kombinációján alapszik. Egy adott NLP feladatra minden BiLSTM réteg egyedi súlyt kap. A végső háló 2 darab BiLSTM rétegből áll, minden LSTM réteg 4096 széles.

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.

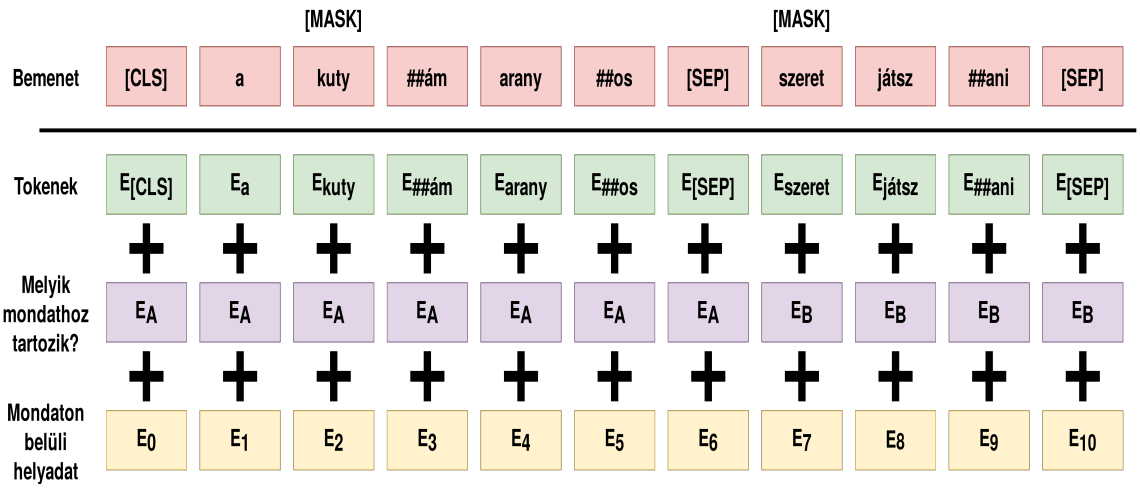
BERT

Egy 2018-ban publikált cikk rámutatott arra, hogy a karakteralapú algoritmusok nem teljesítenek olyan jól, mint a szóalapú társaik. A Bidirectional Encoder Representations from Transformers (BERT) 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 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 token-eket, a token-ekhez 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 kitalálni, hogy A[SEP]B input mondatokra B rákövetkezője-e A-nak. A *maszkolás* során véletlenszerűen letakarták a token-eket 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 ábrázolja, a [SEP] a mondatok közötti szeparátor és a [MASK] a letakart szótöredékeket helyettesíti.

A továbbiakban a modell finomhangolása az adott NLP feladat szerint történik.

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 token-eket. A *transformer* architektúra nem igényel vektoriális bemenetet, saját reprezentációt épít a token-ek számára is.

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.

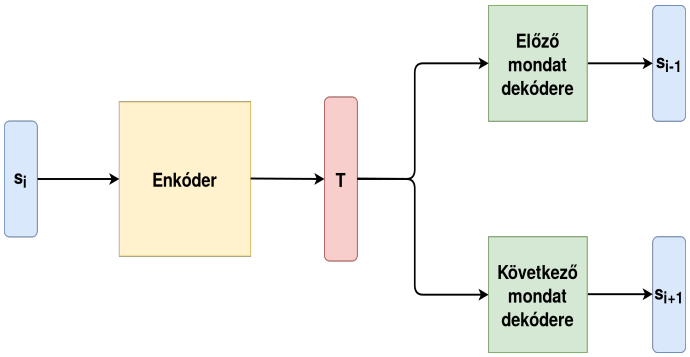
2.2.1. Mondatvektorok

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 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 a legjobb teljesítményével segíteni 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 reprezentáció vektora a rejtett, úgy nevezett *thought* vektor (T).

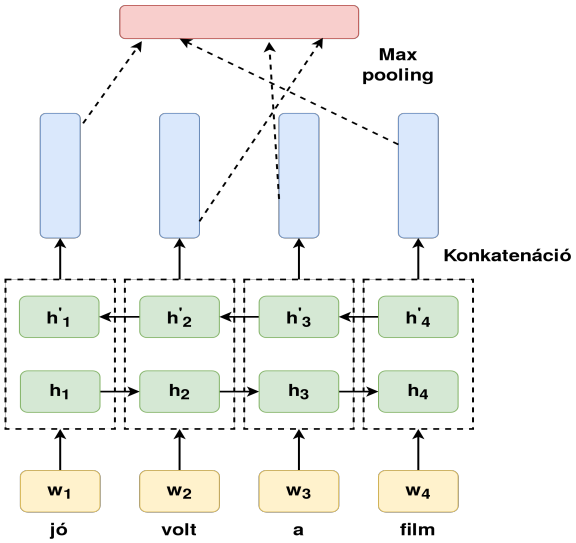
Bár a *Skip-thought* módszer képes a mondaton belüli és kívüli sorrendiségi információ leképezésére is, csak olyan esetben teljesít megfelelően, ahol az egyes mondatok – melyekre alkalmazzák – megfelelő kontextusban szerepelnek, nem izoláltak.

InferSent

2017-ben a Facebook kutatói jelentős áttörést értek el a mondatszintű reprezentációs módszerek terén, a technika neve InferSent. Hasonló algoritmusokkal ellentétben a szerzők felügyelt tanítást végeztek, melyhez az SNLI adathalmazt vették igénybe. A cikk megmutatta, hogy egy kisebb adathalmazon történő felügyelt tanítás felülmúlhatja a nagyobb adathalmazon nem felügyelt módon tanított modellek teljesítményét.

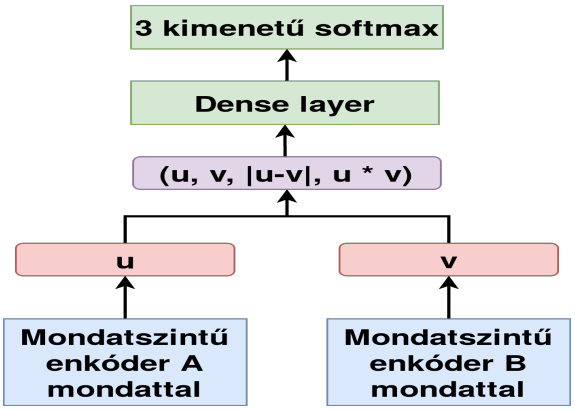
Az SNLI adathalmaz 570 ezer darab – ember által írt és címkézett – mondatpárból áll. A címkék a következők: következmény, ellentmondás és semleges.

Négyféle neurális architektúrát összemérve a legpontosabb eredményt a BiLSTM + max pooling mutatta.



2.6. ábra. A BiLSTM + max pooling architektúra

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



2.7. ábra. Az NLI feladat

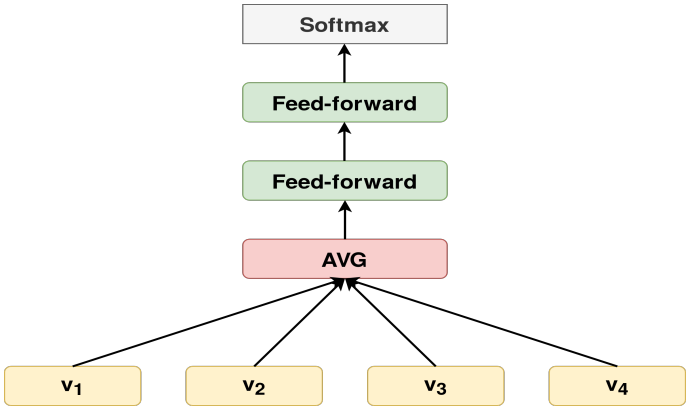
Az így készült u és v vektorokból egy speciális reprezentáció készült: u, v, $|u - v|$ és $u * v$ (vektoriális szorzat) konkatenációjával, melyet végül egy 3 osztályú klasszifikáló hálóba vezettek.

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

Megjegyzés. *Az InferSent napjaink egyik legjobb teljesítményű 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) 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 szavakból, mondatokból, vagy akár rövidebb bekezdésekből is képes 512 méretű vektorokat generálni.

A neurális hálók tanítását két részre bontották, bemenetként angol nyelvű karakterláncokat 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 cikk során kiemelt szerepet kapott a tanítás második fázisa. Számos módon finomhangolták a modelleket és mérték a teljesítményüket. A feladatok közé tarto-

zott, hogy filmes értékelések szövege alapján ki kellett találnia a neurális hálóknak az értékelések pontszámát 1 és 5 között. Továbbá vásárlói értékelések 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 a bemeneti hossz függvényében. Továbbá memóriahasználatban is kedvezőbb választás 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.

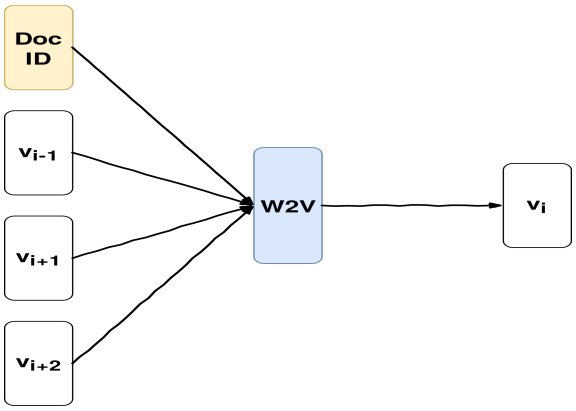
A szerzők rávilágítottak arra, hogy kevés adat esetén jó választás lehet a *transfer learning* módszere, és a magasabb szintű reprezentációk pontosabb eredményt érhetnek el a legtöbb feladat esetében.

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 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 képviseli 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 könnyű.

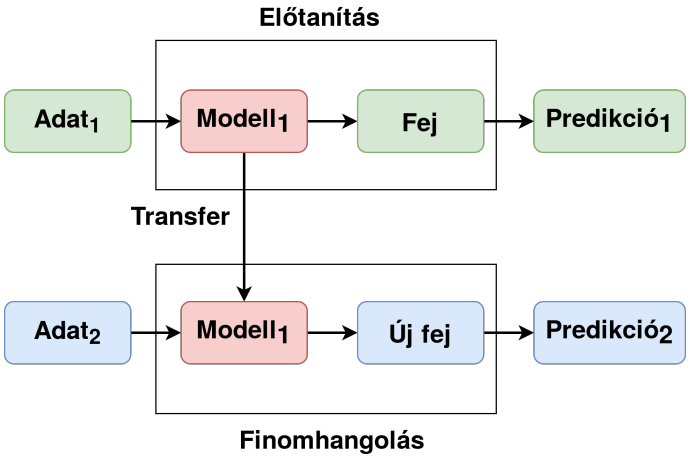
2.3. Transfer learning

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 tanítsuk több aspektus szerint.

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 in-*

putváltozók 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, mely kellőképpen generalizál és a neurális hálónk sok, hasznos és általános információhoz tud jutni. A folyamathoz használt adathalmaz általában nagy mennyiségű annotálatlan adatot tartalmaz, de vannak kivételek, például az InferSent 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 pontosabb eredmény 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

Az adathamazok és az előkészítés

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

Bár az olyan nyelveken, amelyeken a kutatásokat folytatják és amelyeket széles körben beszélnek előfordulhat ember 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 kevesebb, de humán annotált halmazokon precízebb eredményt érnek el.

A magyar nyelv a kisebb körben használt nyelvek közé tartozik, így bátran vonhatjuk le azt a következtetést, hogy a web és egyéb források által hozzáférhető tartalmak mennyisége is erősen limitált. 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.

3.0.1. Általános előkészítési lépések

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 token-ekre 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 – token-eket is, melyek megzavarhatják modellünk működését. A tisztítás során törölhetjük az adott nyelvben sűrűn előforduló szavakat (*stopword*) is – például névelők – , így csak azok a token-ek maradnak a halmazban, amelyek valódi információtartalommal bírnak.

A szöveg **normálása** olyan módosításokat jelent, mely során az adathalmazunk token-eit azonos alakra hozzuk. A token-eket kis-, vagy nagybetűssé konvertálhatjuk, illetve a numerikus tartalommal rendelkező szavakat számokká alakíthatjuk. Természetesen ebben az esetben is célszerű törölni a numerikus token-eket, 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 adathalmazunkba. Az n-gram kifejezés egy n hosszú token szekvenciára utal, tehát a "New York"

kifejezés 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:

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

N-gram modellünk minden n hosszú token szekvencia 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 token-eit konkatenálják, így a tanítás során az algoritmusunk egy token-ként kezelheti a népszerű kifejezéseket.

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

A szótövezés során két *lemmatizer* teljesítményét hasonlítottam össze, ezek a BSI beépített *lemmatizer*-e és a HungarianSpacy. Az adathalmazok előkészítése alatt úgy tűnt, hogy a HungarianSpacy 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 és az NLTK nevű könyvtárakat használtam segítségül.

3.0.2. Magyar wikipédia

A Wikipédia a világ egyik legnagyobb többnyelvű, szabadon szerkesztett online enciklopédiája. Több, mint 6 000 000 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 token-t tartalmaz.

A tanításhoz szükséges előkészítés után az adathalmaz mérete 2 GB-ra, a sorok száma 12 592 489-re a token-ek száma pedig 86 605 435-re csökkent.

A hozzáfűzött reményekkel ellentétben a magyar nyelvű cikkek relatíve elenyésző mennyiségben szerepelnek a Wikipédia adatbázisban. Következésképp a tanítóhalmaz nem bizonyult elegendőnek, így a továbbiakban csak a különböző technikák tesztelésére tudtam használni.

3.0.3. OSCAR

Az OSCAR (*Open Super-large Crawled ALMAnaCH coRpus*) egy nyelvi klasszifikáló algoritmussal készült adathalmaz, melyet a szerzők a Common Crawl 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 adatmennyiség első felével folytattam tovább az előkészületeket, melynek az oscar_hu nevet adtam.

Az oscar_hu nyers változata 127 654 271 darab sort és 5 168 152 283 darab token-t tartalmaz. Az előkészítési procedúra után 15 GB-ra csökkent a méret, 63 692 408 darab 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.

3.0.4. Hungarian Webcorpus

A Hungarian Webcorpus a legnagyobb mai magyar nyelvű korpusz, melyet a Budapesti Műszaki Egyetem Média Oktató és Kutató Központja gyűjtött 2003-ban

a WordSword 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 dokumentumokra 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. Az így kapott korpusz 589 millió szót tartalmaz, melyet 1221 millió magyar nyelvű weboldalról töltöttek le.

A letölthető 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 mérete 18 GB volt, mely a tisztítási lépés után 7.8 GB-ra csökkent. A jelentős méretváltozás az XML tag-ek törlésére vezethető vissza. A token-izált változatban a méret tovább zsugorodott, így a végeredmény egy 6.6 GB-os adathalmaz lett.

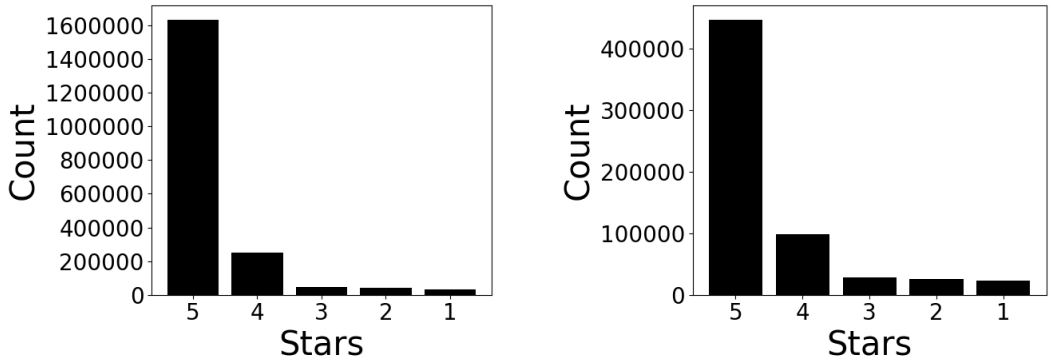
A 1 221 405 darab fájl a tisztítás után összesen 127 711 725 darab sort és 589 209 017 darab token-t tartalmaz.

3.0.5. Árukereső vélemények

Az Árukereső a legnagyobb magyar 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 adatokhoz egy saját kezűleg fejlesztett webscraper segítségével jutottam hozzá. A halmaz mérete összesen 943 MB, amely 141 064 termék és 2209 áruház véleményeit tartalmazza.

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



(a) Összes

(b) Legalább 10 hosszú

3.1. ábra. Vélemények eloszlása

A túl rövid vélemények kevés információval szolgálnak, így eltávolítottam az eredeti adathalmazból a 10 token-nál rövidebb véleményeket. A leghosszabb vélemény 842 token-ből áll.

4. fejezet

A módszer leírása

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 és a nagy tömegek igényeinek hiányából fakad.

Ismereteim szerint magyar nyelven a tárgyalt kategóriák közül kizárólag szóbeágyazási modellek léteznek – mint a FastTex, Word2Vec és az ELMo – , továbbá a lehetséges tanítási feladatok is korlátozottak, így többnyire csak felügyelet nélküli tanítás elvégzése lehetséges. A diplomamunkám során megoldandó feladat egy mondat/paragrafus szintű nyelvi modell elkészítése, amely alapjául az előzményekben megismert módszerek szolgálnak. Továbbá olyan humán és autoannotált adathalmazok létrehozása és 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 létrehozott adathalmazok és 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 kerülnek feldolgozásra.

(KÉP: architektúra)

Az implementációt Python nyelven végeztem a Tensorflow nevű könyvtár segít-

ségével.

4.1. Bemeneti réteg

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

A tanítás során minden esetben a mélyháló számára megadott szóbeágyazási modellt használtam. A választott algoritmus a Word2Vec - CBOW volt. Eleinte, az implementáció alatt a wiki-hu adathalmazon tanított Word2Vec-et alkalmaztam, azonban az adatsor kis mérete miatt más megoldásokat kellett keresnem. Ezek után több, az oscar_hu halmazon tanított modellt is kipróbáltam. A végső és legjobb alternatíva egy kisebb, X méretű szótárral rendelkező, az oscar_hu adatsoron tanított szóbeágyazás lett. A kiválasztási szempontok közé tartozott a pontosság és a memóriaigény, melyet a későbbiekben kifejtek.

(lookup kép)

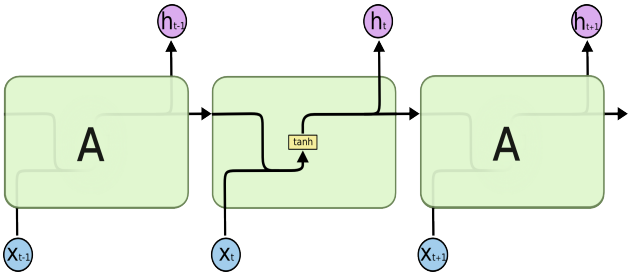
A bemeneti réteg fix súlyokkal rendelkezik, tehát a tanulási folyamat során nem változtatja azokat. A mélyháló a számára átadott beágyazási mátrix elemei közül kikeresi az azonosítóknak megfelelő elemeket, majd továbbítja őket a kimenetre. Ezt a folyamatot *embedding lookup*-nak nevezzük.

(ide még kéne valami)

4.2. A reprezentáció neurális hálója

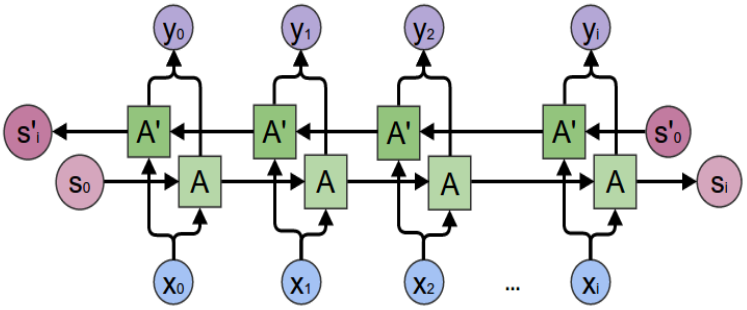
A rekurrens neurális hálók (RNN) használata a szemantikus reprezentációs modellek esetén gyakori technika. Míg a mesterséges neurális hálók csak önálló bemenet fogadására képesek, addig a rekurrens neurális hálók alkalmasak szekvenciális input

feldolgozására is. Ilyen szekvencia például az idősori adat, vagy a szöveges adat is. A szekvenciális bemenetet az különbözteti meg önálló bemenettől, hogy a szekvenciális input elemei függenek egymástól, hatással lehetnek a szomszéd elemekre, több önálló input esetén ez a reláció nem érvényes. A rekurrens neurális hálók képesek megtanulni az adatsor elemei közötti kapcsolatokat. Az RNN a tanulási folyamat során "emlékszik" az előző bemenetekből gyűjtött információkra, majd azok segítségével generálja a kimenetet/kimeneteket. A számítás során használt vektorokat nem csak az input súlyai befolyásolják, 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.



4.1. ábra. Az RNN cella

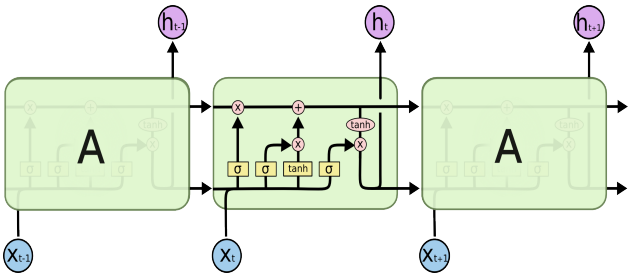
Bizonyos esetekben, ahol a múltból származó információ elegendő lehet a háló számára – például következő token generálása az előzőek függvényében – , az RNN jó opció lehet. Azonban olyan feladatok során, melyeknél fontos a bemeneti adatok kontextusa – például a nyelvi modellek – , más megoldásra van 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.



4.2. ábra. A BiRNN architektúra

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 a túl kicsi gradiensek, hogy a hozzá tartozó rétegek megállnak a tanulásban. Ezt a problémát a *vanishing gradients* problémának hívjuk.

Az LSTM (*Long short-term memory*) architektúra megoldást nyújt a *vanishing gradient* 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. A módszer csak releváns információt enged a hosszútávú memóriába.



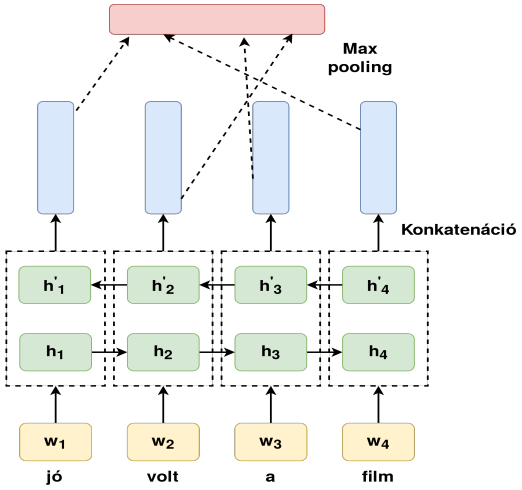
4.3. ábra. Az LSTM cella

(Írjak még az LSTMről?, kapuk, aktivációk részletesen)

Az általam a feladat megoldására választott architektúra az InferSent-ben kiváló eredményeket prezentáló BiLSTM + Max Pooling.

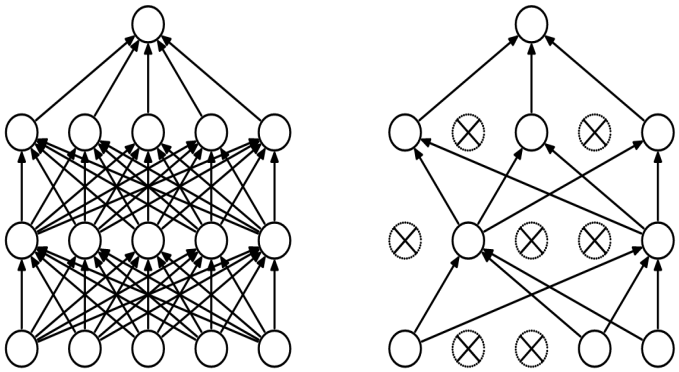
4.2.1. A BiLSTM

A BiLSTM egy kétirányú rekurrens architektúra (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 adat 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, az egyik jelenlegi legpontosabb megoldásnak bizonyul.



4.4. 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 duplázza a konvergációhoz szükséges iterációk számát. A reprezentáció tanulására szolgáló neurális háló mindkét LSTM rétegére konfigurálható *dropout*-ot alkalmaztam.



4.5. ábra. A dropout vizualizációja

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

4.2.2. Pooling réteg

A *pooling* ötletét szintén a számítógépes képfeldolgozás ágazatától kölcsönözte az NLP. Míg 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 átlagot veszi alapul.

A BiLSTM-ben található rejtett rétegek kimeneteinek páronkénti konkatenációján végzett pooling művelet képes kiválasztani a hasznos információt az egyes token-eket/rész szekvenciákat reprezentáló vektorokból. Az így kapott sorvektor lesz a bemeneti szekvencia 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 reprezentáció neurális hálójának implementációja során törekedtem a minél széleskörűbb konfigurálhatóságra, így elősegítve a könnyebb testreszabhatóságot és az újrafelhasználhatóságot.

A felhasználó által állítható, modellre vonatkozó paraméterek a következők:

- *use_ embedding_ layer* : Alkalmazzon a háló bemeneti réteget, vagy vektorizált az input.

- *word_embedding_dim* : A bemeneti 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. Tanítás

Bár a szemantikus reprezentációs modellek pontosságához jelentősen hozzájárul az alkalmazott architektúra teljesítménye, 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 az adott 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 megoldásához implementált feladat a BERT előtanításához hasonló *multi-task learning*.

A *multi-task learning* több feladaton egyszerre történő tanulást jelent, ami jobb generalizációra készteti a hálót. Az egyik feladat a **maszkolás**, amely az egyes szavak közötti szemantikai és szintaktikai jelentés 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 *hungarianWebcorpus* nevű adathalmazzal dolgoztam, mivel az jelentős mennyiségű szöveges adatból áll, és a sorok az OSCAR adathalmazzal ellentétben sorrendtartóak.

A generálás során 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* token-eket konkatenált a dokumentum szavaihoz.

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

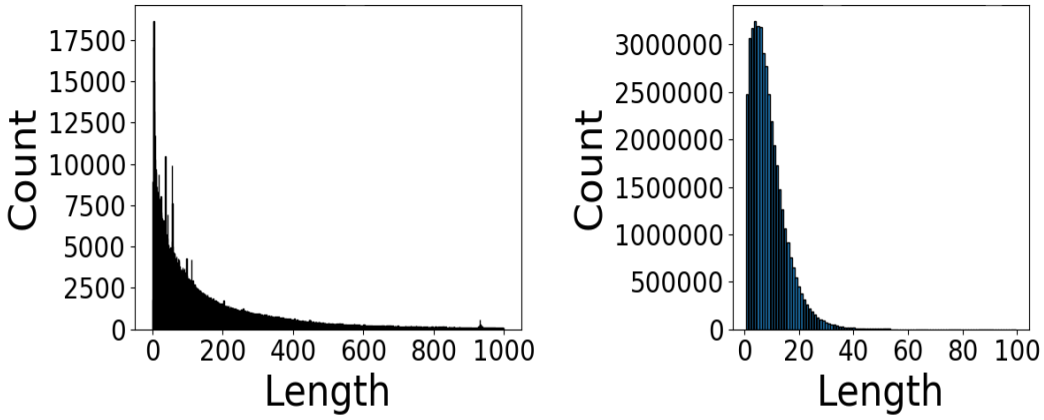
- A kiválasztott token-ek 80%-át a "[MASK]" speciális token-nel helyettesítette.
- 10%-át a szótárból választott véletlenszerű szóval helyettesítette.
- A maradék 10% esetében maradt az eredeti token.

A BERT szerzői szerint, ha a maszkolt token-ek 100%-át prediktálná a rendszer, nem feltétlen lenne képes megfelelő minőségű reprezentációt generálni a nem maszkolt szavaknak. Ha a kiválasztott token-ek 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 token-ek 90%-át maszkolná és 10% maradna az eredeti szó, a modell lemásolná a nem kontextusfüggő beágyazásokat.

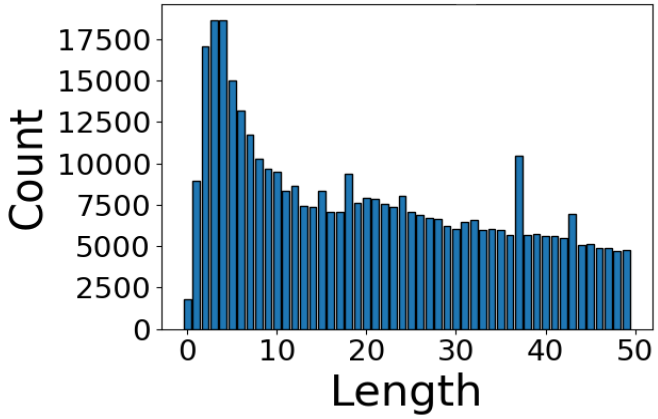
A *következő mondat* feladathoz szükséges "mondatpárokat" az így előállított tokenszekvenciákon végigiterálva generálta az algoritmus. S szekvenciát 0.5 valószínűséggel S után következő szekvenciával, 0.5 valószínűséggel a korpuszból választott véletlenszerű szekvenciával konkatenálta. A két szekvencia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

Megjegyzés. *A "mondat" szó ebben az esetben token-ek 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 token-szám, amelyet még elfogadjon.



4.6. ábra. Az 1000 token alatti dokumentumok száma (balra); A 100 token alatti dokumentumok száma (jobbra)



4.7. ábra. Az 50 token hosszúság alatti mondatok

A dokumentumokra és a dokumentumok soraira vonatkozó mérések alapján minimális token-számnak 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*-ot alkalmazni, továbbá ne legyen a konfigurációnak teljesíthetetlen memóriaigénye. A végső bemeneti hossz így 201 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 token-ekhez tartozó szóvektorokat. Felmerülhet a kérdés, hogy mi történik a spe-

ciális token-ekkel ebben az esetben. A speciális token-eket 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 a 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 és azonosan 1 távolságra kerültek tőlük. Ennek okán a speciális token-ek nem befolyásolják a tanulás során a szemantikai információkat. A bemeneti rétegnek átadott beágyazási mátrix mérete – az [UNK], Word2Vec modell által nem ismert szavakat jelölő token-nel együtt – (word_embedding_dim + 4) × szótár méret lesz.

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

4.3.1. Maszkolás feladat

A maszkolási feladat során a neurális háló célja kitalálni, milyen token-ek voltak a [MASK] token-ek helyén. A feladat abszolválásához a fejnek szüksége van a mondatpárok generálása során létrejött egyéb információra is. Ilyen információ a maszkolt token-ek szekvencián belüli pozíciója, Word2Vec azonosítója és súlya. Mivel a [MASK] token-ek megfelelő eloszlása érdekében az algoritmus a [PAD] token-eket is letakarja, ezért fontos a súlyok bevezetése. Egy maszkolt token súlya 0 lesz, 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 ellátott rétegen vezeti át, melynek kimeneti mérete megegyezik a kibővített Word2Vec modell beágyazási dimenziójával. Majd a sűrű rétegtől kapott kimenetet megszorozza a beágyazási mátrixszal, annak érdekében, hogy az egyes vektorok a szótár dimenziójába kerüljenek, továbbá *log softmax* aktivációt alkalmaz, hozzájutva a maszkolt token-ekre vonatkozó valószínűségek logaritmusához a teljes szótárra nézve. A feladat tanítási *loss* függvénye az egyes eredeti token-ekre vonatkozó logaritmusok összegének ellentettjének átlaga.

4.3.2. Következő mondat feladat

A *következő mondat* feladat alatt a neurális hálónak 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 előállító 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 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.

Tanítás - folytatás

A *multitask learning* során a neurális háló egyszerre két feladatot próbál megoldani, így jobban generalizál. A háló az SGD (*Stochastic Gradient Descent*) nevű algoritmus segítségével minimalizálja a globális *loss* függvényt, amely a két feladat *loss* függvényének összege. Az SGD 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.

configs

Tanítási részletek? halmaz vágása, befejezés

4.4. Vektorok generálása

mentett súlyok betöltése, súlyok fagyasztása, fej nélküli használat

két kép kicserélése

5. fejezet

A módszer kiértékelése

6. fejezet

Összegzés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Ábrák jegyzéke