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

Szövegösszegzés Dokumentum-Vektor Embedding Megközelítéssel

TDK Dolgozat

Készítette
Nagy Péter Géza

Konzulens
Dr. Szegletes Luca

2019

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Nagy Péter Géza*, hallgató kijelentem, hogy ezt a dolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik.

Budapest, 2019. október 28.

Nagy Péter Géza
hallgató

Kivonat

Dolgozatomban a természetes-nyelv feldolgozás (angolul: Natural Language Processing - NLP) egyik kurrens problémájával foglalkozok, a hosszú szövegek automatizált rövidítésével, más szóval szövegösszegzéssel (angolul: Text Summarization). A folyamatot az a szándék vezérli, hogy a kimenet ne csak egyszerűen rövidebb legyen, de koherens és olvasható is egyúttal.

Egy erre képes algoritmusnak számtalan alkalmazási módja elképzelhető. Példának okáért napi szinten is nagyon sok könyv jelenik meg világszerte, és természetesen nincs idő mindet elolvasni. Néhányan úgy próbálják megoldani a problémát, hogy a kiszemelt könyv szavait egyenként jelenítik meg tipikusan egy okostelefon kijelzőjének közepén, mindegyiket a másodperc töredékéig. Ezzel a módszerrel körülbelül 120 szót lehet befogadni percenként (angolul: Words Per Minute), viszont lényegesen kevesebb információ marad meg az eredeti olvasási élményhez képest.

Ezzel szemben a módszer, amit bemutatok a dolgozatomban inkább azt célozza meg, hogy minél kevesebb legyen az információveszteség és lehetőleg az olvasási élmény se csorbuljon. Ezt úgy valósítom meg, hogy a program elhagyja az eredeti szövegből az elbeszélés szempontjából nem olyan lényeges részeket és csak a hasznos marad meg. Cél, hogy az olvasónak ne legyen hiányérzete a rövidített szöveg olvasása során.

Az elkészítendő szövegösszegző program egy neurális hálózatot fog használni, amely képes teljes dokumentumok többdimenziós vektorizálására. A hálózat tanításához a CNN/Daily mail adathalmazt használom fel, amely hosszú szöveg- rövidített szöveg párokat tartalmaz. A tanítás úgynevezett felügyeletlen tanítási módszerrel történik (angolul: unsupervised learning), ami azért is jelent kihívást, mert a hálózat kiértékelésénél nincs konkrét elvárt érték, nehezen lehet megállapítani, hogy jó irányba halad-e a tanítás. Szerencsére a választott adathalmazom rövidített szövegeit össze tudom hasonlítani a hálózat kimenetével és megállapíthatók különböző szöveghasonlóságot mérő metrikák (mint például a ROUGE pontszám).

A módszerem négy, jól definiálható lépésből áll. Először vektorizálom a rövidítendő szöveget, másodszor vektorizálom a szöveg mondatait is külön-külön, harmadszor pedig megmérem a távolságot a teljes szöveg vektora és a mondatokhoz tartozó vektorok között. Utolsó lépésben pedig megtartom az eredeti szöveg vektorához legközelebb álló valahány mondatot.

Dolgozatom első felében bemutatom a már létező megoldásokat a szövegösszegzés problémájára és hogy az én megoldásom hogyan illik a képbe. Ezen felül részletesen körüljárom a megvalósított program működését, az elkészítési fázisokat, a felhasznált technológiákat, valamint a neurális hálózat működését.

Abstract

In my study, I deal with one of the current problems of Natural Language Processing (NLP), the automated abbreviation of long texts, in other words Text Summarization. The process is driven by the intention to not only make the output shorter, but also coherent and readable.

There are many ways to apply an algorithm capable of this. For example, many books are published daily throughout the world, and of course there is no time to read them all. Some try to solve the problem by displaying the words of the selected book one by one, typically in the middle of a smartphone's display, each for a fraction of a second. This method can accommodate about 120 words per minute (Words Per Minute), but with significantly less information than the original reading experience.

In contrast, the method I present in my paper is aimed at minimizing the loss of information and keeping the reading experience. I accomplish this by leaving out the non-narrative parts of the original text and leaving only the useful ones. The goal is to ensure that the reader does not feel deficient when reading the abbreviated text.

The text summarization program I introduce will use a neural network capable of multidimensional vectorizing entire documents. To teach the network, I use the CNN/Daily mail dataset, which contains long text-shortened text pairs. Teaching is done by using a so-called unsupervised learning method, which is also challenging because there is no specific expected output value for evaluating the network, and it is difficult to determine if teaching is going in the right direction or not. Fortunately, I can compare the shortened texts in my selected dataset with the network output and calculate different text similarity metrics (such as the ROUGE score).

My method consists of four well-defined steps. Firstly, I vectorize the text to be abbreviated, secondly I vectorize the sentences of the text separately, and thirdly, I measure the distance between the original document vector and the vectors associated with the sentences. In the final step, I keep the top N sentences closest to the vector of the original text.

In the first part of my paper, I present the existing solutions for the text summarization problem and how my approach fits into the picture. In addition, I will describe in detail the operation of the implemented program, the preparation stages, the technologies used and the operation of the neural network.

1. fejezet

Bevezetés

Ebben a fejezetben vázolom a területet, amin dolgoztam, a feladat fontosságát, valamint a módszereket vázlatosan, amikkel meg lehet oldani a problémát.

1.1. Mit értünk természetes-nyelv feldolgozás (angolul: Natural Language Processing) alatt?

A **természetes-nyelv feldolgozás (NLP)** számítási algoritmusok készítésével foglalkozik, amelyek **automatizáltan elemzik és reprezentálják az emberi nyelvet**. Az NLP-alapú rendszerek sokféle alkalmazást tettek lehetővé, például a Google hatékony keresőmotorját, az önvezető autókba beépített, valamint az otthonokban, okostelefonokban megtalálható digitális asszisztenseket. Az NLP arra is alkalmazható, hogy megtanítsa a gépeket **a természetes nyelvhez kapcsolódó összetett feladatok** - például gépi tanulás és párbeszédgenerálás - **végrehajtásának képességére**.

Az NLP problémáinak tanulmányozására használt módszerek nagy részben hosszú ideig sekély gépi tanulási modelleket és időigényes, **kézzel rögzített jellemzőket (angolul: features) alkalmaztak**. Mivel egyrészt a nyelv dinamikusan változik, mindig karban kell tartani ezeket a modelleket, másrészt lehetséges, hogy a modell megközelítése elve nem helytálló.

Ugyanakkor, az utóbbi időkben megjelent **mélytanulási (angolul: Deep Learning) technikák** – mint például a mostanában sikeresen alkalmazott szóbeágyazó réteg (angolul: Embedding Layer) – segítségével a neurális hálózat alapú modellek **kiváló eredményeket értek el a különböző nyelvfeldolgozással kapcsolatos feladatokban**, összehasonlítva a hagyományos gépi tanulási modellekkel, mint például az SVM (Support Vector Machine) vagy a logisztikus regresszió.

1.2. Mit értünk szövegösszegzés (angolul: Text Summarization) alatt?

A szövegösszegzés, illetve szövegösszefoglalás a **nagy terjedelmű szövegek tömör és pontos összefoglalásának elkészítésének technikája**, miközben a hasznos információkat közvetítő szakaszokra összpontosít **az általános jelentés elvesztése nélkül**. Az automatizált szövegösszegzésnek az a kitűzött célja, hogy az olyan szövegeket, amiknek a **manuális összegzése időben költséges lenne, algoritmizáltan, számítógép segítségével végezzük el**.

Az utóbbi néhány évben bebizonyosodott, hogy a gépi tanulásra (angolul: Machine Learning) alapuló algoritmusok alkalmasak arra, hogy **azonosítsák a dokumentumok megértéséhez szükséges fontos részleteket és tényeket közvetítő szakaszokat**, ami belátható, hogy nélkülözhetetlen a szövegösszegzési feladat elvégzéséhez.

1.3. A feladat fontossága

A jelenlegi, digitális térben zajló tartalommennyiség robbanásban (ami tipikusan nem strukturált adat) szükség van automatizált szövegösszegzést végző eszközökre, hogy a tartalomfogyasztók **gyors betekintést nyerhessenek az igazi tartalomról általánosságban**, hiszen nincs idő minden könyvet, cikket, egyéb dokumentumot elolvasni.

Mindazonáltal a tapasztalatok alapján belátható, hogy ezeknek a tartalmaknak a **jelentős része redundáns**, esetleg jelentéktelen, amely így nem fedi le a tényleges jelentését a dokumentumnak. Például, ha egy online megjelent hírben egy specifikus információt keresünk, lehetséges, hogy az egész cikket el kell olvassuk ahhoz, hogy megtaláljuk, amit keresünk, **miközben rengeteg időt elpazarlunk**. Éppen ezért, az olyan automata összegzők – amelyek képesek a fontos részeket kiemelni és a lényegteleneket elhagyni – nélkülözhetetlenek és egyre fontosabbá válnak korunkban.

Az ilyen módon implementált szövegösszefoglalók így **növelhetik egy adott dokumentum olvashatóságát**, miközben lehetővé teszik, hogy az érintett területről **több információt gyűjtsünk be**, mintha nem használnánk ilyen eszközt.

1.4. A szövegösszegzők fő változatai

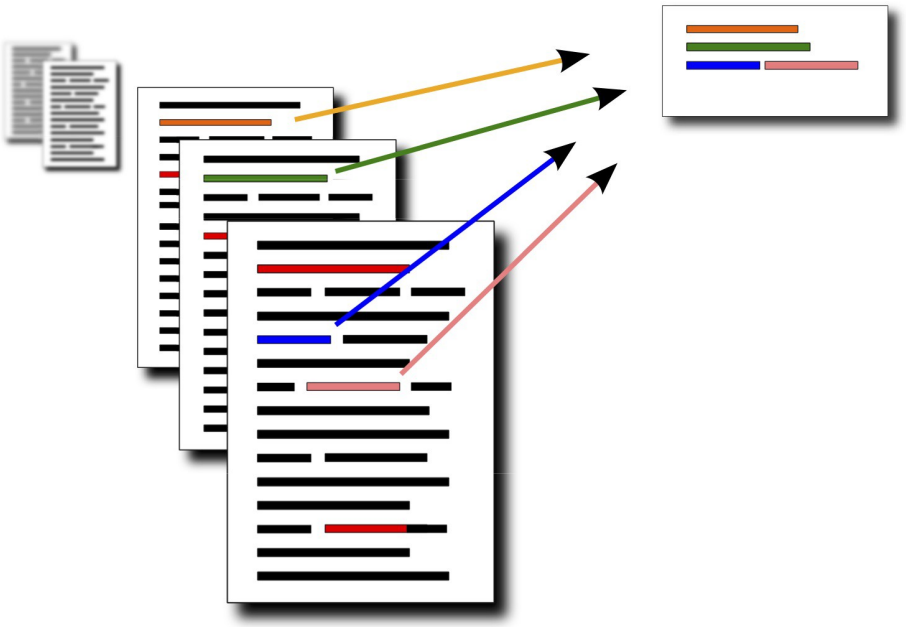
Tulajdonképpen **kétféle módszert különböztethetünk meg**: kivonat (angolul: extraction) és absztrakció alapút.

1.4.1. Kivonat-alapú összegzés

Kivonat-alapú összegzésben **kiválasztjuk és kombináljuk a szavak azon alhalmazát az eredeti szövegből, ami a legjobban reprezentálja az eredeti jelentést**. Egy jó hasonlat rá például az, ahogyan **a szövegkiemelőt is használjuk: egyszerűen kiemeljük a legfontosabb részeket**.

Gépi tanulási módszerek esetén a kivonatolás általában úgy működik, hogy **a szöveg bizonyos szekciót (tipikusan mondatait) súlyozzuk, sorrendezzük**, majd ezeket használjuk fel, hogy megkapjuk a kívánt összegzést.

Sokféle típusú algoritmus létezik, hogy ezeket súlyokat megállapítsuk és rangsoroljuk őket a relevanciájuk és egymáshoz való hasonlóságuk alapján, majd egymás után illesszük őket, hogy előállítsuk az összefoglalót. Ehhez kapcsolódóan látható egy illusztráció a 1.1. ábrán[1].



1.1. ábra. *Kivonat alapú összegzés*

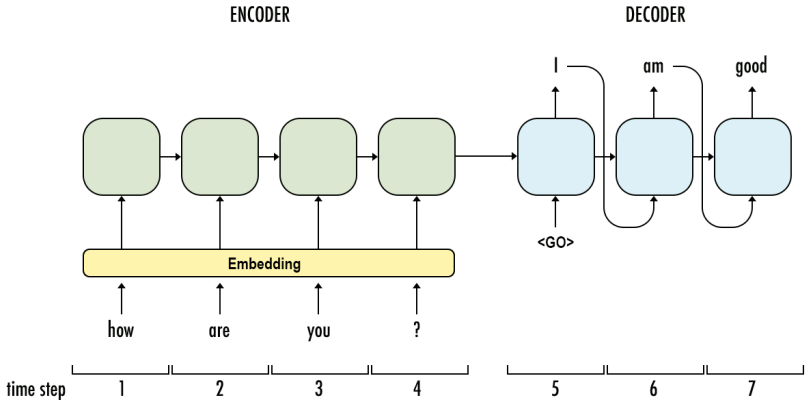
Munkám során ezt a módszert választottam a szövegösszefoglaló algoritmus implementálására. A dolgozatomban bemutatott megközelítés nem kézzel megszerkesztett szabályokon alapul. Ehelyett, egy **mesterséges intelligencia bázisú módszert választottam, mert az utóbbi évtizedben bebizonyosodott, hogy természetes-nyelv feldolgozásban kifejezetten jól tud működni**.

1.4.2. Absztrakció-alapú összegzés

Absztrakció-alapú összegzés esetén korszerű mélytanulási techikákat alkalmazunk az eredeti dokumentum **átfogalmazására**, illetve rövidítésére, pont ahogy az emberek is teszik. Mivel ezek a gépi tanulási módszerek **új mondatokat hozhatnak létre** (amik nem találhatók meg az eredeti dokumentumban), képesek lehetnek arra, hogy egy tömörebb megfogalmazással átadják **ugyanazt az információ mennyiséget, más minőségben**. Erre látható egy illusztráció a 1.2. ábrán[2].

Habár az absztrakció-alapú összegzés jobban teljesít, mint a kivonat-alapú, **rendkívül bonyolult egy jól működő algoritmust elkészíteni**. Ráadásul lehetséges,

[1] https://miro.medium.com/max/1460/1*5_t4EJl1Iy9B1w5EtX1Zog.jpeg
[2] https://miro.medium.com/max/1329/1*O7a0ShsYNCjxs-lNeux8MA.png



1.2. ábra. *Absztrakció alapú összegzés sequence-to-sequence neurális hálózattal*

hogy **különböző nyelvtanú természetes nyelvekre nem is működik ugyanaz az algoritmus**. Például a német vagy a magyar nyelvek agglutinációval állítják elő a szavakat, ami azt jelenti, hogy morfémákat (a nyelv legkisebb jelentéssel bíró részeit) ragasztják egymás után, miközben mindegyik módosít valamennyit a szó jelentésén. Az angolban is használt flexió esetén a szóalakok nehezen szegmentálhatók, a toldalékok a szótővel sokszor egybeolvadnak.

A két technikát természetesen lehet **ötvözni** is. Például egy kivonat-alapú algoritmus kiválasztaná az N legfontosabb mondatot, az absztrakció-alapú pedig átfogalmazná úgy, hogy vagy beletenne az eredeti helyett rövidebb átvezető mondatokat, vagy mégjobban lerövidíthetné a már meglévő összegzést.

1.5. Fordítások

Korábbi tanulmányaim során többször találkoztam olyan magyar fordításokkal, amelyek eltorzítják az eredeti szó, illetve fogalom jelentését, ami így kifejezetten megnehezítette a szöveg értelmezését. Annak érdekében, hogy tanulmányom minél olvashatóbb legyen, **igyeztem megtartani az eredeti angol kifejezéseket**.

2. fejezet

Szakirodalmi áttekintés

Ebben a fejezetben vázolok néhány state-of-the-art szövegösszefoglalót, valamint írok a munkám során felhasznált, fontosabb technológiákról. Tanulmányom során több cikkből is gyűjtöttem tudást[1].

2.1. Meglévő megoldások

Ismeretes, hogy a szövegösszegzési probléma egy népszerű terület a természetesnyelv-feldolgozás témakörön belül. Ennek megfelelően **számos meglévő megoldás létezik rá**, különböző nyelvekre.

A PapersWithCode[2] weboldal tartalmaz egy táblázatot **a legfrissebb eredményekről a szövegösszegzés területén**. Látszódik, hogy **az adathalmaz választásom szerencsés volt**, mert ezeket az eredményeket ugyanezzel az adathalmazzal érték el, így könnyebb volt összehasonlítanom a munkám ezekkel a modellekkel.

Már a neurális hálózatoknak adott nevekből kiderült számomra, hogy a BERT (Devlin et al., 2018) hálózatot használják fel a legjobb eredményeket elérő modellek. Felvetődött bennem is, hogy én is erre az útra térjek, viszont miután átolvastam néhány cikket, **úgy döntöttem, hogy a fókuszom középpontjába inkább egy kivonat-alapú megoldást helyezek**. Attól tartottam, hogy ugyanabba a hibába esek, mint a BSc szakdolgozatomban, amikor magyar nyelvre készítettem morfológiai egyértelműsítőt, miszerint a magyar nyelv összetettségével nem fog megbirkózni – a **tanításhoz több erőforrásra lenne szükségem**.

Ugyanakkor a linkelt táblázatban jelenleg egy olyan megoldás vezet (Liu and Lapata, 2019), amely BERT alapú, ráadásul egy általános keretrendszert szolgáltat mind absztrakciós, mind kivonatolós futtatásra.

Raffel et al. (2019) másik megoldása **transfer learning-et alkalmaz**, de nem csak szövegösszegzésre, hanem például szöveg klasszifikálásra, valamint a kérdés-válasz problémákra is megfogalmaz egy megoldást. A transfer learning a gépi tanulás egy

[1] https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/

[2] https://paperswithcode.com/sota/document-summarization-on-cnn-daily-mail

olyan területe, ahol egy neurális hálózattal megtaníttatnak egy problémát, majd **az immár tanított neuronokkal egy másik problémát próbálnak megoldani** (például ha egy neurális hálózat megtanult biciklit vezetni, akkor kiegészíthető úgy, hogy autót is tudjon).

2.2. A Word Embedding réteg

Habár mégis természetes nyelvű szöveggel foglalkozom, a neurális hálózat számokkal, ezért fontos, hogy **valamilyen szám-alapú reprezentációja legyen minden szöveg elemnek**, amit a hálózat megkap, így tud vele számolni. **Ebben segít az embedding (beágyazó) neurális réteg**.

Például a BSc-s szakdolgozatomban kétféle embeddiggel is dolgoztam. Az egyik esetben az egyes szavakhoz rendeltem egy-egy egyedi értéket (előfordulási sorrendben egy autoinkremens sorszámot), a másik esetben pedig az egyes morfémákhoz (a nyelv legkisebb jelentéssel bíró egysége, szórészlet).

2.2.1. Word2Vec - szavakból vektor

Ugyanakkor itt nem áll meg a történet, ugyanis kiderítették a Word2Vec készítése során (Mikolov et al., 2013), hogy sokkal pontosabb, jobb minőségű eredményeket el lehet érni, ha **egy szót, illetve annak a jelentését nem egy darab számmal próbáljuk kifejezni** (ezt hívják Bag of Words (BOW) technikának), **hanem többel**, amik már a tanulási folyamatban is részt vesznek.

Ha **a szavak nagyobb sűrűségű, vektoros reprezentációját veszem elő** - melyekkel természetesen matematikai műveletek sokaságát lehet elvégezni -, akkor például a "király" és a "férfi" szavak koszinusz (néha euklideszi) távolsága kicsi lesz, míg a "király" és "ütvefúró" szavak távolsága nagy. Ezen összefüggések megállapítására az Embedding neurális réteg két technikát alkalmaz: a **Skip-gram modellt (SG)** és a **Continous-bag-of-words modellt (CBOW)**, ezekről később beszélek bővebben.

A tanítóadathalmaz áljon a 2.1 táblázatban található mondatokból.

Mondat	Tokenizálva
Ez a diploma ötös.	Ez/0 a/1 diploma/2 ötös/3 ./4
Ez a TDK kiváló.	Ez/0 a/1 TDK/5 kiváló/6 ./4

2.1. táblázat. *Példa adathalmaz*

A mondat oszlop triviális, a tokenizált változat viszont abban érdekes, hogy **a pontot is külön vettem** (így nem lesz két külön szó az "ötös" és az "ötös."), valamint (bár ez nem feltétlen a tokenizálás részt) hozzárendeltem az egyes szavakhoz egy számértéket előfordulási sorrend szerint.

Eszerint numerikus vektorokként ábrázolva a BOW technika szerint az előbbi adathalmazt, a (2.1) mátrixot kapom, mint tanító adathalmazt.

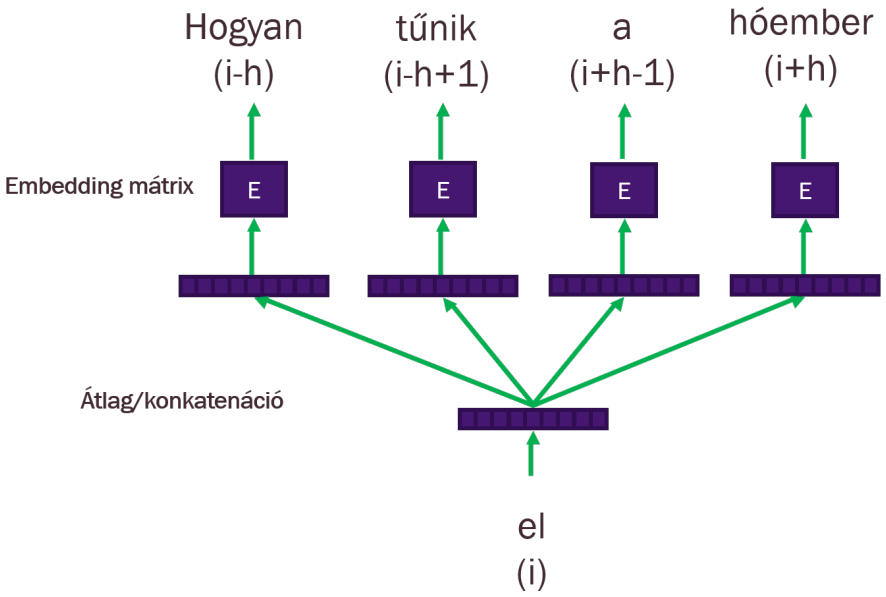
$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 5 & 6 & 4 \end{bmatrix} \tag{2.1}$$

Skip-gram (SG) Modell

Ez a modell tulajdonképpen egy **N méretű szóablakot tol végig a szövegen** (aminek a szavaiból szeretnénk embedding vektorokat), majd **feltanít egy 1 rejtett rétegből álló neurális hálózatot**.

Ez a hálózat egy **szintetikus feladatot old meg**, hogy ha megadunk neki **egy bemeneti szót**, akkor adja meg, hogy **a környező N-1 szóval** (elvárt kimenet) **mekkora valószínűséggel fordul elő**. Ezt a megközelítést a 2.2 képlet szemlélteti. A képletben a w (word) a szóablak egy adott szavát jelzi, alsó indexben az ablakon belüli sorszám található. **Az embedding mátrix tulajdonképpen a rejtett réteg lesz**, így ha a réteg például 64 neuront tartalmaz, akkor 64 dimenziós word embeddingeket kapunk.

$$p(w_{i-h}, ..., w_{i+h}|w_i) \tag{2.2}$$

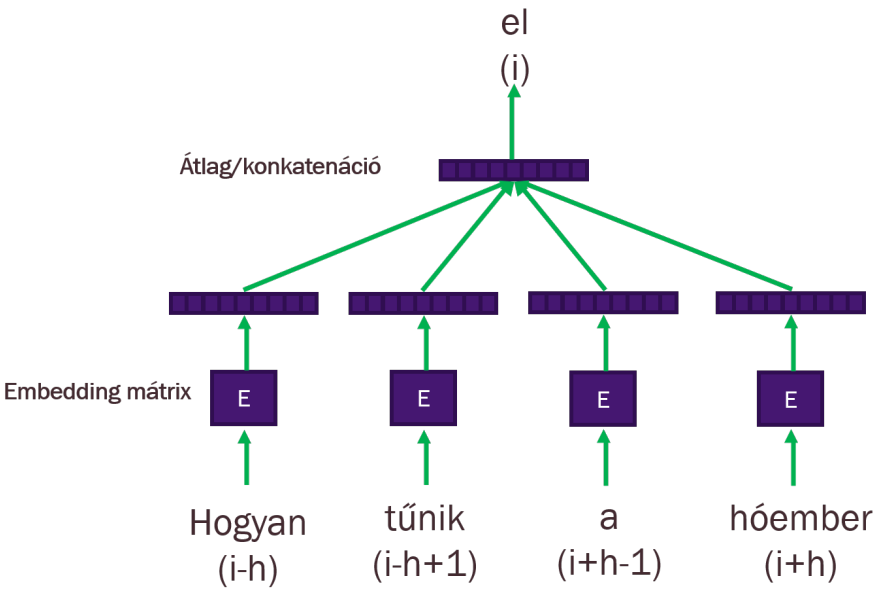


2.1. ábra. *Skip-gram módszer*

Continuous-bag-of-words (CBOW) Modell

A CBOW is **egy rejtett réteget használ**, mint az SG - itt is embedding mátrix lesz belőle -, viszont **a feladatban különbözik**, amit meg kell oldania a neurális hálózatnak. Az SG modellel ellentétben itt **nem egy darab bemeneti szó lesz**, hanem **egy szó ablak szavainak átlaga/összefűzése és az elvárt kimenet az ablak közepén lévő szó**. Ezt a megközelítést pedig a 2.3 képlet szemlélteti.

$$p(w_i|w_{i-h}, ..., w_{i+h}) \tag{2.3}$$



2.2. ábra. *Continuous-bag-of-words módszer*

Haszálat

Egy embedding rétegnek **három hiperparamétere van**, amit be kell állítani egy neurális hálózat esetében: **a szótár mérete** (most 7, mert ennyi szó van összesen, a pont írásjellel együtt), az **embedding vektorok mérete** (például 8), a **bemeneti vektorok hossza** (most 5).

Ebben a specifikus esetben a 2.2 embedding táblázat fog létrejönni.

Index	Embedding vektor
0	[0.2, 0.9]
1	[0.8, 0.3]
2	[0.5, 1.4]
3	[0.7, 1.1]
4	[1.7, 1.6]
5	[2.2, 0.2]
6	[0.4, 0.9]

2.2. táblázat. *Embedding mátrix*

Az embedding vektorokat **vertikálisan összekonkatenálva** kapom az embedding mátrixot, amin dolgozik az embedding réteg a tanítás folyamán. Ami itt érdekesség, hogy például a "diploma" és "TDK", valamint az "ötös" és "kiváló" szavak jelentése közel azonos. Ennek következtében a tanítási folyamat során **az ezekhez a szópárokhoz tartozó vektorok közel fognak kerülni egymáshoz**. Ha az vektortér legfeljebb három dimenziós, akkor triviális ennek az elképzelése, ugyanis **a hasonló szavakhoz tartozó vektorok hasonló irányba fognak mutatni**.

Megjegyzem, hogy magyar nyelvre nehezebb ilyen vektorokat előállítani az agglutinatív jellege miatt, ugyanis sokszor már **egy betű eltérés is nagy jelentésbeli különbséget okozhat**, illetve számolni kell a szövegkörnyezettel is.

Az embedding kiértékelése

Nos, ez egy nagyon bonyolult kérdés, ami megnehezíti az én munkámat is, de az emberi ítélőképesség még éppen hogy használható. Rendelkezésre áll például néhány **adathalmaz, amiket nyelvészek készítettek el, amiben szópárok találhatók, valamint egy hozzájuk társított számérték, ami azt jelzi, hogy mennyire hasonlóak egymáshoz (például egy szubjektív 10-es skálán értékelték kérdőívek segítségével)**. Ilyen listát a választott embedding modell is elő tud állítani, mely ha ugyanabban a formátumban van, mint a nyelvészek táblázata, akkor felhasználhatók összehasonkításra, a modell kiértékelésére. Ilyen módszer lehet például a **Spearman korreláció**[3].

2.2.2. Doc2Vec - dokumentumból vektor

A legegyszerűbb és leggyorsabb megoldás nyilván az lenne, ha *a vektorizálandó dokumentum szavaiból vektorokat készítenék, majd kiátlagolnám őket*, de ez egy *nagyon durva közelítés volna.* Viszont egy **paragrafus vektor (PV) bevezetésével** (Le and Mikolov, 2014) sokkal **jobb eredményeket el lehet érni**.

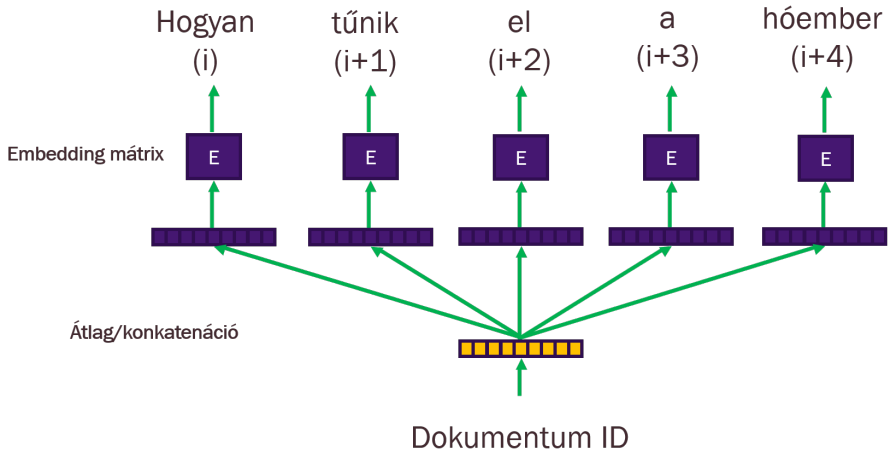
Az alapgondolat nagyon egyszerű: úgy teszünk, **mintha létezne egy speciális szó minden dokumentumhoz**, amik máshol nem fordulnak elő. Ez a vektor pont úgy fog viselkedni, mintha egy átlagos szó vektor lenne, amit a rejtett réteggel állítunk elő, **egyedül a tanító adathalmaz előállításában van különbség, a hálózat struktúrájában nincs**. Két módszert ismeretek rá a következőkben, mindkettőt használtam munkám során.

Paragraph Vector - Distributed Bag of Words (PV-DBOW)

Magyarul ez a Paragrafus Vektor - Elosztott szózsákok módszere, ami az **SG modellhez analóg**. Itt annyi a különbség, hogy ebben az esetben **a bemeneti szó csak a dokumentum vektor lesz**, és **a dokumentumban lévő szavakat próbálja megjósolni a hálózat**. A működést az 2.4 képlet írja le, ahol a dokumentum egyedi azonosítója a d szám.

$$p(w_{i-h}, ..., w_{i+h}|d) \tag{2.4}$$

[3]https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

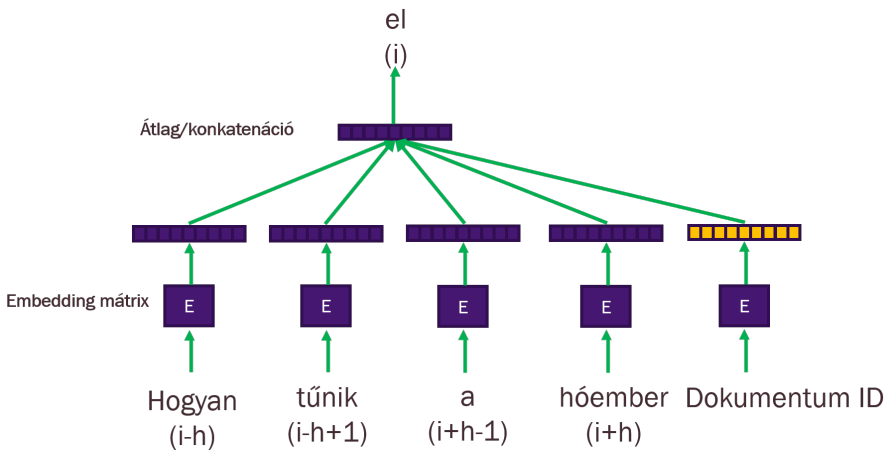


2.3. ábra. *Paragraph Vector - Distributed Bag of Words*

Paragraph Vector - Distributed Memory (PV-DM)

Ez pedig magyarul a Paragrafus Vektor - Elosztott memória változat. Ez a módszer **analóg a Word2Vec-es CBOW modellhez**. Úgy kapjuk meg a dokumentum vektorokat, hogy **a neurális hálózatot arra tanítjuk, hogy jósolja meg a szóablak középső szavát a környező szavak átlaga és a dokumentum vektor alapján**, melyeket bemenetként kap. Ebből látható, hogy **pontosabb eredményt produkál a PV-DBOW-nél, mivel megőrzi a szavak sorrendjét**. Az elvárt kimenetet az 2.5 képlet írja le, ahol a dokumentum egyedi azonosítója a d szám.

$$p(w_i|w_{i-h}, ..., w_{i+h}, d) \tag{2.5}$$



2.4. ábra. *Paragraph Vector - Distributed Memory*

2.3. Kiértékelés a ROUGE segítségével

A ROUGE[4] (Lin, 2004) a **Recall-Oriented Understudy for Gisting Evaluation** kifejezést rövidíti. Ez alapvetően különböző típusú **metrikák gyűjteménye az automatizáltan generált szövegek, valamint a gépi fordítások minőségének kiértékeléséhez**.

Úgy működik, hogy **összehasonlítja az automatikusan összeállított összefoglalót** vagy a fordítást **olyan referencia szövegekkel, amelyeket ember fogalmazott meg** – az én esetemben a CNN újságírói. Hagyatkozván a ROUGE terminológiára, az automatikusan összeállított összefoglalókat **system summary**-nek fogom hívni, az ember által megfogalmazottakat pedig **model summary**-nek.

2.3.1. Nehézségek a használatban

Meg kell említsem, hogy ez tulajdonképpen egy több, mint 10 éve írt **Perl script**. Ettől eltekintve, egy remek algoritmus gyűjtemény, viszont **Pythonból nehézkes a használata**. Létezik rá részleges Python implementáció, viszont **munkám során a wrappert (csomagolót) használtam kiértékelésre**[5], ami meghívja az eredeti scriptet.

Érdekesség, hogy **mind a wrappert, mind az implementációt pyrouge-nak hívják** és ez az elején lassította a munkát (például a *pip install pyrouge* tulajdonképpen melyik csomagot is telepíti fel – a wrappert, ami nekem pont jó). Végül kiderült, hogy tulajdonképpen mindkettőre szükségem van, mert **a másik csomag tartalmazza az eredeti Perl scriptet** valamiért, és azt kellett beállítsam a wrappernek egy segéd binárison keresztül, hogy oda hívjon át[6]. Mindemellett Linuxon és Windows-on is más lépéseket kell végrehajtani.

2.3.2. Működése egy példán keresztül

A következőkben **a ROUGE működését egy példán keresztül szemléltetném**. Tekintsük a következő **model summary**-t (ember fogalmazta meg):

```
the cat was under the bed
```

Most pedig nézzük meg ezt a **system summary**-t (gép fogalmazta meg):

```
the cat was found under the bed
```

Ha csak az egyes szavakat vesszük figyelembe, akkor a system summary és a model summary között **egymást átfedő szavak száma 6**. Ez túl sokat egyelőre nem

[4] https://en.wikipedia.org/wiki/ROUGE_(metric)

[5] https://github.com/bheinzerling/pyrouge

[6] https://stackoverflow.com/questions/47045436/how-to-install-the-python-package-pyrouge-on-microsoft-windows/47045437

mond, viszont szerencsére rendelkezésünkre áll két ennél beszédesebb metrika, a **precision** és a **recall**. Megjegyzem, hogy különböző szövegkörnyezetekben máshogy definiálták őket, de most azt fogom taglalni, **ahogy a ROUGE interpretálta őket**.

A recall számítása

ROUGE esetében a recall arra utal, hogy a **model summary-ből a system summary mekkora részt fed le**. Ha az az átfedő szavakat vesszük figyelembe, akkor a (2.6) képlet alapján számíthatjuk ki.

$$Recall = \frac{átfedö\ szavak\ száma}{model\ summary\ szavainak\ száma} \tag{2.6}$$

A fentebbi példában a recall (2.7) alapján alakulna. Ez azt jelenti, hogy a model summary összes szavát megragadta a system summary.

$$Recall = \frac{6}{6} = 1.0 \tag{2.7}$$

Ez egy szép eredmény szövegösszefoglaló rendszer esetében, de **ez a metrika még mindig nem mond el mindent a system summary minőségéről, ugyanis az hosszabb is lehet, mint a model summary**, miközben annak minden szavát tartalmazza, így 100%-os eredményt kapnánk. Ez nyilvánvalóan azt jelenti, hogy a ellentétes hatást értünk el a system summary generálásakor, mert bőbeszédűbb, mint a model summary.

A precision számítása

Ez az a pont, ahol a precision képbe jön. Ennek szempontjából lényegében **azt mérjük, hogy a system summary mekkora része releváns**, szükséges. A precisiont a (2.8) képlet segítségével számíthatjuk ki.

$$Percision = \frac{átfedö\ szavak\ száma}{system\ summary\ szavainak\ száma} \tag{2.8}$$

A fentebbi példában a precision (2.9) alapján alakulna. Eszerint a **system summary-ben szereplő 7 szóból valójában csak 6 szükséges**.

$$Precision = \frac{6}{7} = 0.86 \tag{2.9}$$

Ellenben, ha a következő system summary-t tekintjük, a számok máshogy alakulnak.

```
the tiny little cat was found under the big funny bed
```

Ebben az esetben a precision már (2.10) alapján alakul.

$$Precision = \frac{6}{11} = 0.55 \tag{2.10}$$

Ez már nem néz ki olyan jól. Ennek oka az, hogy **nagyon sok felesleges szó van az összefoglalóban**. Ez alapján körvonalazódik, hogy a percision egy nagyon fontos metrika az összefoglaló algoritmusok értékelésében.

A recall és a precision eredője: az F-score

A fentiek alapján látszódik, hogy **érdemes kiszámolni mind a precision-t, mind a recallt**. Viszont mi a helyzet, akkor ha van több neurális modellünk, és az **egyiknél nagy a precision, de kicsi a recall, a másiknál meg éppen fordított a helyzet?**

Erre a célra tökéletesen megfelel az **F-score, amely a kettő harmonikus átlaga**, melynek 1 a legjobb értéke (tökéletes precision és recall), és 0 a legrosszabb. Ennek a képlete a (2.11) részben található. Elmondható, hogy tulajdonképpen abban különbözik az egyszerű kiátlagolástól, hogy itt **a kisebbet nagyobb súllyal veszi figyelembe**.

$$Fscore = 2 * \frac{precision * recall}{precision + recall} \tag{2.11}$$

Neurális hálózatok tanításánál általában loss-t és accuracy-t mérnek. Az általam használt esetben **az accuracy-nek megfeleltethető az F-score**.

2.3.3. A ROUGE metrikák alfajtái

A ROUGE-N, ROUGE-S és ROUGE-L metrikák tulajdonképpen **nem a szavakat nézik, hanem az n-grammok átfedéseit**, amelyek n betűnként feldarabolt részek a szövegben. Ezekből van több nevesebb változat, például az unigramm (betűnként feldarabolt változat), bigram (kétbetűnként), trigam (három betűnként), stb. Ez főleg akkor jön jól, ha agglutinatív nyelvre (például magyarra) készített összefoglalót szeretnénk kiértékelni, mert ha a szavakat tekintenénk darabolási alapnak, akkor már 1 betű eltérése is teljes eltérést mutatna.

- ROUGE-N - az unigram (ROUGE-1), bigram (ROUGE-2), trigram (ROUGE-3) és magasabb rendű n-gramm átfedéseket méri
- ROUGE-L - Least Common Subsequence (LCS - legritkábban előforduló alszekvencia) segítségével állapítja meg a Longest Matching Sequence (LMS - leghosszabb átfedő szekvencia) számadatot. Az LCS nagy előnye, hogy nem követeli meg az egymást követő átfedéseket, hanem inkább az olyan szekvencián belüli átfedéseket nézi, amelyek tükrözik a mondat-szintű szórendet.

- ROUGE-S - Bármely szópár előfordulást néz egy mondatban, sorrendet figyelembevéve. Ezt másképp skip-gram előfordulásnak is hívják. Például a skip-bigram az olyan szópár előfordulásokat nézi, amelyek között maximum 2 másik szó van.
- ROUGE-W: Súlyozott LCS alapú statisztika, ami az egymást követi LCS-ket részesíti előnyben.
- ROUGE-SU: Skip-bigram és unigram alapú együttes előfordulást elemző statisztika.

Például míg a ROUGE-1 az unigrammok átfedéseit nézi a system és a model summary között, addig a ROUGE-2 a bigrammok átfedéseit.

A fentebbi példa alapján **megmutatom, hogy hogyan kell a ROUGE-2 metrikát kiszámolni**.

Legyen a következő a system summary:

```
the cat was found under the bed
```

Ez pedig a model summary:

```
the cat was under the bed
```

Bigrammok a system summary-ben:

```
the cat, cat was, was found, found under, under the, the bed
```

Bigrammok a model summary-ben:

```
the cat, cat was, was under, under the, the bed
```

A ROUGE-2 szerint a recall (2.12) alapján alakul.

$$ROUGE - 2_{Recall} = \frac{4}{5} = 0.8 \tag{2.12}$$

Tehát a system summary 4 bigrammot talált el az model summary-ben található 5-ből, ami nagyon jónak számít. A ROUGE-2 szerinti precision (2.13) alapján alakul.

$$ROUGE - 2_{Precision} = \frac{4}{6} = 0.67 \tag{2.13}$$

Az itt kapott precision azt mondja nekünk, hogy az összes system summary bigrammból 67% -kal van átfedésben a model summary. Megyjegyzem, hogy **ahogyan az összefoglalók** – mind a system, mind a model – **egyre hosszabbá válnak, úgy**

egyre kevesebb átfedő bigramm lesz. Ez különösen igaz az absztrakciós összefoglalók esetében, ahol nem közvetlenül használják újra a mondatokat az összefoglaláshoz.

Azért **érdemesebb ROUGE-1-et, ROUGE-2-t (vagy más nagyobb felbontású) metrikát alkalmazni, mert figyelembe veszik az előállított szöveg olvashatóságát is a szavak sorrendje miatt**. Tehát ha a system summary jobban követi a model summary szavainak sorrendjét, mondhatjuk azt, hogy nagyobb a kohézió. Ennek később nagy szerepe lesz a modelljeim kiértékelésében.

2.3.4. A ROUGE Python wrapper (pyrouge) bemenete

Most, hogy ismertettem a ROUGE működését, rátérek a pyrouge[7] alkalmazására. A könyvtár **két mappát vár, ami a bemeneti fájlokat tartalmazza**.

Az **egyik könyvtár a system summary-ket** tartalmazza, **a másik pedig a model summary-ket**, úgy hogy a fájloknak speciális, **reguláris kifejezésre illeszkedő nevük kell, hogy legyen**, ugyanis ezalapján párosítja össze a két mappában lévő fájlokat.

A system summary mappájában a fájloknak a *cnn.(+).txt* regexre kell illeszkedniük, míg a model summary fájljainak a a *cnn.[A-Z].#ID#.txt* mintára. A működés során a pyrouge végig iterál a system summary fájljain, kiolvassa belőlük a számazonosítót (+), majd megkeresi az összes fájlt a model summary mappájából, amelyben megtalálható az *#ID#* helyen az adott szám. Elképzelhető, hogy több model summary is rendelkezésünkre áll, ezeket nagybetűkkel *[A-Z]* kell azonosítani a fájlnévben. Az én esetemben csak egy model summary áll rendelkezésre egy cikkhez, így mindenhol nagy *A* betű szerepel.

Egy példa fájlstruktúra látható a 2.5. ábrán.

A pyrouge-ban több védelem is van a fájlok validására. Például felesleges fájlok nem lehetnek egyik mappában sem. Ha egy fájlnak nincs párja, a pyrouge kivételt dob.

Példaként tekintsük meg a cnn.A.1.txt (model summary) fájl tartalmát:

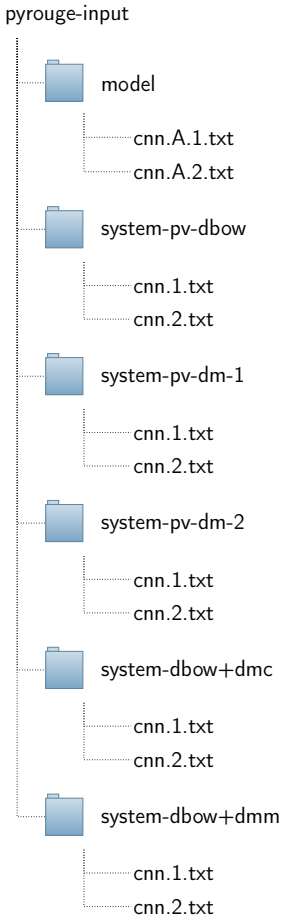
```
1 scores villages are damaged destroyed after two earthquakes
2 iranian media rescue operations have ended after the saturday
   ↪ quakes
3 historic sites are also damaged
4 iran sits major fault lines and prone devastating earthquakes
```

A cnn.1.txt (system summary) pedig ugyanígy néz ki:

```
1 the head emergency services gholamreza masoumi told the
   ↪ semiofficial fars news agency that people were injured
2 the first earthquake with magnitude hit saturday local time
   ↪ according the geological survey which said the epicenter was
   ↪ kilometers miles southwest ahar
3 iran sits major fault lines the collision the arabia and eurasia
   ↪ plates and has been prone devastating earthquakes
```

Azért tartalmaz csak három mondatot (soronként egyet), mert **átlagosan három mondatot fogalmaztak meg az újságírók a cikkeikhez**.

[7] https://github.com/bheinzerling/pyrouge



2.5. ábra. *Példa a ROUGE elvárt fájl bemeneteire*

2.3.5. A ROUGE Python wrapper (pyrouge) kimenete

Lentebb található egy példa a pyrouge kimenetére. Amint látható, a script **több metrikát is előállít és valóban mindegyikhez három számérték tartozik**. Nem nehéz kiszámolni, ez összesen 24 lebegőpontos számadat egy automatizáltan generált összefoglaló értékelésére. Ezen kívül **konfidencia intervallumokat is szolgáltat az algoritmust**.

```
1  ---------------------------------------------
2  1 ROUGE-1 Average_R: 0.21718 (95%-conf.int. 0.17040 - 0.26410)
3  1 ROUGE-1 Average_P: 0.19421 (95%-conf.int. 0.16179 - 0.22779)
4  1 ROUGE-1 Average_F: 0.18914 (95%-conf.int. 0.15600 - 0.22208)
5  ---------------------------------------------
6  1 ROUGE-2 Average_R: 0.05698 (95%-conf.int. 0.03839 - 0.07751)
7  1 ROUGE-2 Average_P: 0.04500 (95%-conf.int. 0.03029 - 0.06174)
8  1 ROUGE-2 Average_F: 0.04728 (95%-conf.int. 0.03225 - 0.06444)
9  ---------------------------------------------
10 1 ROUGE-3 Average_R: 0.02222 (95%-conf.int. 0.01034 - 0.03508)
11 1 ROUGE-3 Average_P: 0.01613 (95%-conf.int. 0.00660 - 0.02780)
12 1 ROUGE-3 Average_F: 0.01816 (95%-conf.int. 0.00799 - 0.02950)
13 ---------------------------------------------
14 1 ROUGE-4 Average_R: 0.01079 (95%-conf.int. 0.00258 - 0.01951)
15 1 ROUGE-4 Average_P: 0.00824 (95%-conf.int. 0.00179 - 0.01579)
16 1 ROUGE-4 Average_F: 0.00910 (95%-conf.int. 0.00227 - 0.01699)
17 ---------------------------------------------
18 1 ROUGE-L Average_R: 0.19692 (95%-conf.int. 0.15640 - 0.23772)
19 1 ROUGE-L Average_P: 0.17810 (95%-conf.int. 0.14944 - 0.20790)
20 1 ROUGE-L Average_F: 0.17212 (95%-conf.int. 0.14292 - 0.20064)
```

```
21 ----------------------------------------------
22 1 ROUGE-W-1.2 Average_R: 0.09381 (95%-conf.int. 0.07523 - 0.11326)
23 1 ROUGE-W-1.2 Average_P: 0.13728 (95%-conf.int. 0.11458 - 0.16083)
24 1 ROUGE-W-1.2 Average_F: 0.10209 (95%-conf.int. 0.08569 - 0.11892)
25 ----------------------------------------------
26 1 ROUGE-S* Average_R: 0.05448 (95%-conf.int. 0.03582 - 0.07722)
27 1 ROUGE-S* Average_P: 0.03633 (95%-conf.int. 0.02338 - 0.05177)
28 1 ROUGE-S* Average_F: 0.03435 (95%-conf.int. 0.02335 - 0.04693)
29 ----------------------------------------------
30 1 ROUGE-SU* Average_R: 0.06399 (95%-conf.int. 0.04355 - 0.08757)
31 1 ROUGE-SU* Average_P: 0.04664 (95%-conf.int. 0.03238 - 0.06316)
32 1 ROUGE-SU* Average_F: 0.04173 (95%-conf.int. 0.03005 - 0.05476)
```

Ezekhez dictionary-ként is hozzáférek, ezekkel a kulcsokkal (nem soroltam fel mindet, a többi ROUGE-hoz tartozó kulcsoknak hasonló neve van):

'rouge_1_recall', 'rouge_1_recall_cb', 'rouge_1_recall_ce',
'rouge_1_precision', 'rouge_1_precision_cb', 'rouge_1_precision_ce',
'rouge_1_f_score', 'rouge_1_f_score_cb', 'rouge_1_f_score_ce',

Ebből a halmazból **csak a vastagon szedett részre volt szükségem**, hol mindre, hol csak az F-score-ra. Ezeket reguláris kifejezésekkel nyertem ki, amiről említést is teszek a megfelelő részeknél.

3. fejezet

Az adathalmaz előfeldolgozása

Ebben a fejezetben azt taglalom, hogy a készítendő neurális hálózat bemenetét hogyan állítottam elő lépésről lépésre. Tárgyalom többek között az adathalmaz eredetét, próbálkozásaimat, választásom miértjét, a korpusz felépítését, valamint az előállított adatstruktúrát.

3.1. Két változat, két próbálkozás

A Szakirodalmi áttekintés fejezetben bemutatott szövegösszefoglaló módszerek többek között abban közösek, hogy **ugyanazokat az adatforrásokat használták feltételezhetően annak érdekében, hogy az eredmények összehasonlíthatóak legyenek egymással**.

A CNN[1], illetve a Daily Mail[2] hírportálokon található cikkekről több adathalmazt is készítettek, de mint később kiderült, amiatt, mert másképp tárolják el bennük az adatokat, nem tudtam alkalmazni őket.

Két adathalmazt mutatok be, amelyek ezeken a portálokon található cikkeket tartalmazzák, csak különböző formátumban. Az egyiket a Get To The Point: Summarization with Pointer-Generator Networks című tanulmányban (See et al., 2017) közölték, melynek mind a feldolgozott változata, mind az előállító forráskódja megtalálható GitHubon[3].

Először ezzel próbálkoztam, mivel úgy tapasztaltam, hogy **ez az elterjedtebb változat a szövegösszefoglaló rendszerek körében**. Pontosabban azokéban, amelyek absztrakciót alapú megközelítést alkalmaznak. Erre sajnos csak később jöttem rá, mert mint kiderült, **az adathalmaz formátuma arra a feladatra, amire én szerettem volna használni, nem volt alkalmas**, illetve csak nehezen lehetett volna alkalmassá tenni.

Két része volt az adathalmaznak. A TensorFlow Example objektumaiban tárolt adatok – melyek kiolvasása szintén komplikált volt, de ezt most nem részletezném

[1] https://edition.cnn.com/
[2] https://www.dailymail.co.uk/home/index.html
[3] https://github.com/abisee/cnn-dailymail

```
(?<=(<s>))(\w|\d|\n|[().,\-:;@#$%^&*\[\]\"'+-/\/®°º!?{}|‘~]| )+?(?=(<\/s>))
```

3.1. ábra. *A mondatok kinyerésére felhasznált reguláris kifejezés*

–, felváltva voltak cikkek, illetve absztraktok, amelyek a cikkírók által írt rövid összefoglalók. Valamiért **a cikkeket és az absztraktokat különböző módon tárolták el**.

A cikkek már szavanként voltak tokenizálva, ami még nem is volt nagyon probléma, viszont az absztraktok mondatok szerint. Egy mondatot $< s >$ és $< /s >$ XML tagre emlékeztető karakterszekvenciák közé raktak, melyek megkövetelték, hogy egyedi reguláris kifejezést készítsek a mondatok kinyeréséhez. Ezt a 3.1. jeleníti meg, viszont mivel végső soron **nem ez volt a választott adathalmaz formátum, ezért a jelentését nem részletezném**.

3.2. A választott adathalmaz eredete

A **DeepMind Q&A adathalmazt**[4] a Hermann et al. (2015) tanulmányban mutatták be. Eredetileg azzal a céllal készült, hogy az általuk készített sqeuence-to-sequence **neurális hálózatot megtanítsák a bemenetként kapott szöveg elolvasására és értelmezésére**. A tanulmányban a hálózatot arra használták fel, hogy **a felhasználó által feltett kérdésekre tudjon válaszolni**.

Az **adatgyűjtő programot**[5] amit készítettek a CNN és a Daily Mail **angol nyelvű hírportálokról készített pillanatfelvételen futtatták le**, a Wayback Machine[6] oldalon. Sajnos az oldal instabilitása miatt nehézkes mégegyszer lefuttatni a scriptet, ezért a szerzők letölthető, tömörített formátumban közzétették az adatokat. További nehézség – ami engem szerencsére nem érintett munkám során –, hogy néhány országban letiltották a Wayback Machine-t.

Míg a **CNN 90 ezer, addig a Daily Mail oldalról ezer dokumentumot gyűjtöttek össze**. Minden dokumentumhoz átlagosan 4 kérdés tartozik, melyek mindegyike egy mondat, amiből hiányzik egy szó vagy kifejezés, amit meg lehet találni az adott dokumentumból.

A hírportálokon található cikkekből **két különálló korpuszt készítettek**, hogy mindenki eldönthesse, melyiket szeretné használni. A formátum azonos, tehát akár össze is fűzhetők.

Mindkét adathalmazhoz három-három letölthető állomány tartozik. Külön letölthetők a kérdések, a dokumentumok (a szerzők sztoriként hivatkoznak rá), valamint a nyers HTML állományok. A következőkben bővebben **a dokumentumokat tartalmazó *.story fájlok szerkezetét mutatom be**. Megjegyzem, hogy ez a kiterjesztés szabványos, más programok is használják, létezik hozzá szintaxis kiemelés is PyCharmban.

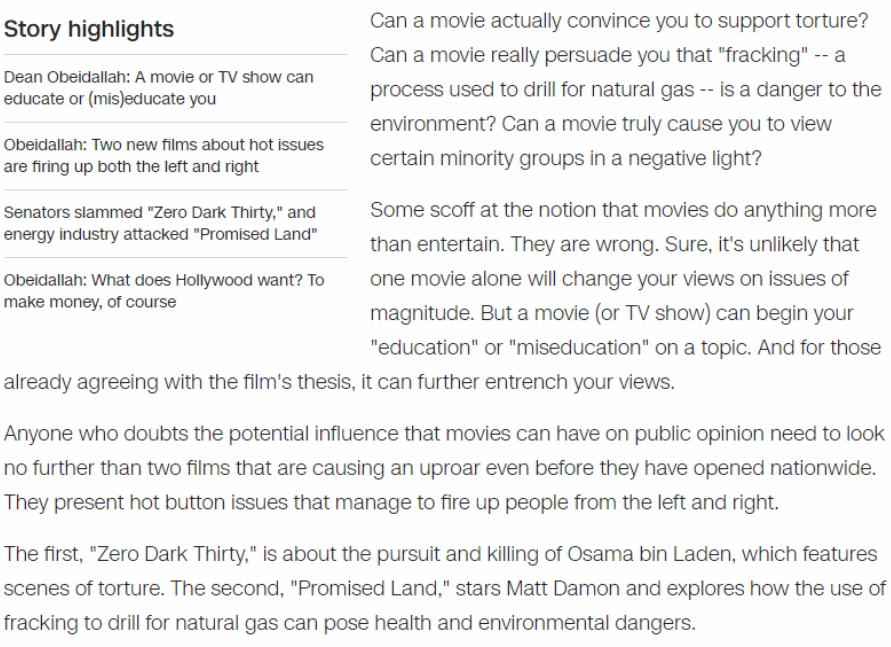
[4]https://cs.nyu.edu/ kcho/DMQA/
[5]https://github.com/deepmind/rc-data
[6]https://archive.org/web/

Tekintsük a következő példát:

```
(CNN) -- Can a movie actually convince you to support torture? Can a movie really persuade you that "fracking" -- a process used to drill for natural gas -- is a danger to the environment? Can a movie truly cause you to view certain minority groups in a negative light?

Some scoff at the notion that movies do anything more than entertain. They are wrong. Sure, it's unlikely that one movie alone will change your views on issues of magnitude. But a movie (or TV show) can begin your "education" or "miseducation" on a topic. And for those already agreeing with the film's thesis, it can further entrench your views.

[...]

The opinions expressed in this commentary are solely those of Dean Obeidallah.

@highlight

Dean Obeidallah: A movie or TV show can educate or (mis) educate you

@highlight

Obeidallah: Two new films about hot issues are firing up both the left and right

@highlight

Senators slammed "Zero Dark Thirty," and energy industry attacked "Promised Land"

@highlight

Obeidallah: What does Hollywood want? To make money, of course
```

A fájl tartalma a CNN oldalán a 3.2 ábrán látható módon jelenik meg.

Az állomány szerkezetileg **két fő részre bontható**. Az első részben **a cikk bekezdései találhatók**, üres sorokkal elválasztva, majd @*highlight* szöveget tartalmazó sorok alatti üres sor alatt a cikk írója által írt **összefoglaló mondat vagy mondatok találhatók**. Ezek a kiemelések néha idézetek a cikkből, néha csupán gondolat összefoglalói egy bekezdésnek.



Story highlights

Dean Obeidallah: A movie or TV show can educate or (mis)educate you

Obeidallah: Two new films about hot issues are firing up both the left and right

Senators slammed "Zero Dark Thirty," and energy industry attacked "Promised Land"

Obeidallah: What does Hollywood want? To make money, of course

Can a movie actually convince you to support torture? Can a movie really persuade you that "fracking" -- a process used to drill for natural gas -- is a danger to the environment? Can a movie truly cause you to view certain minority groups in a negative light?

Some scoff at the notion that movies do anything more than entertain. They are wrong. Sure, it's unlikely that one movie alone will change your views on issues of magnitude. But a movie (or TV show) can begin your "education" or "miseducation" on a topic. And for those already agreeing with the film's thesis, it can further entrench your views.

Anyone who doubts the potential influence that movies can have on public opinion need to look no further than two films that are causing an uproar even before they have opened nationwide. They present hot button issues that manage to fire up people from the left and right.

The first, "Zero Dark Thirty," is about the pursuit and killing of Osama bin Laden, which features scenes of torture. The second, "Promised Land," stars Matt Damon and explores how the use of fracking to drill for natural gas can pose health and environmental dangers.

3.2. ábra. *A CNN oldalán megjelenő .story állomány tartalma (részlet)*

3.3. Betöltés

Egy mappában $*.story$ állományok vannak, melyek nevei a tartalmuk hash-ei. Végigiterálok a mappa tartalmán, majd UTF-8 kódolással **betöltöm a fájlokat egyenként**. Akár egy fájlba is összefűzhetném az egészet, viszont mivel nem kell gyakran preprocesszálni, ezért ettől most eltekintettem.

Egy adott story fájlban megkeresem az első *@highlight* szöveget, majd elmentem a pozícióját. Ezután **elmentem egy változóba az eddig a pozícióig tartó szöveget, ez lesz a cikk**, a többit pedig **felbontom a** *@highlight* **string mentén és string tömbként tárolom**. Ezen kívül **szükséges az egyes szövegekre meghívni egy tisztító metódust**, ami kiszedi a felesleges whitespace-eket (felesleges sortörés, szóköz).

Előfordult olyan, hogy egy $*.story$ állomány nem volt teljes, mert **nem tartalmazott cikket egyáltalán**, csak highlightokat. Szerintem ez azért lehetséges, mert **csak képek voltak az adott cikkben**, azokat meg nyilván nem lehetett szövegként megjeleníteni értelmesen. Ezeket a dokumentumokat **az adatbetöltés során kiszűrtem**.

3.3.1. Adat tisztítás

Annak érdekében, hogy a neurális hálózat **minél több hasznos információt kapjon**, elengedhetetlen a korpusz megtisztítása. A folyamat keretén belül az olyan **redundáns információkat távolítom el**, amik az adott mondat jelentéséhez nem vagy csak nagyon kis mértékben járulnak hozzá, viszont a háttérszámítások illet-

ve **a tanítás gyorsabban fog zajlani**. A feldolgozási ötletet a Machine Learning Mastery[7] oldalról merítettem.

Bár magától értetődőnek tűnhet, fontosnak tartom megjegyezni, hogy **az adattisztítás minden esetben az adott problémára specializálódik**. Lehetséges, hogy egy másik probléma esetén a most eltávolított szövegrészletek fontosak.

Ahelyett, hogy egy meglévő tokenizációs eljárást használtam volna, egy **egyedi megoldást alkalmaztam**, amely bővebben és jobban testreszabható. Ha már eleve kész megoldásokat használtam volna fel, akkor ezeket pipeline szerűen egymásba kellett volna ágyazzam és **ugyanazt a feladatot sokkal lassabban végezte volna el a számítógép**.

Az előző bekezdés gondolatára jó példa az, hogy helyesnek tartottam **eltávolítani a szövegből az írásjeleket**, mivel **nem adnak hozzá a szöveg információtartalmához** olyan sokat. Emellett a **kis- és nagybetűket is eltávolítottam**. Ezt a két lépést márpedig egy iterációban is el lehet végezni.

Továbbá, annak érdekében, hogy a neurális hálózat által felépített, korpusz alapján készült szótár ne legyen olyan nagy és könnyebben elférjen a memóriában, **eltávolítottam a számokat**, illetve az úgynevezett **stop word**-öket is (mint például az *is*, *or*, *as*...). Ezutóbbit úgy oldottam meg, egyszerűen **kihagytam a szövegből a két karakternél vagy annál rövidebb szavakat**.

Megjegyzendő, hogy egyes osztálykönyvtárak erre egy külön, **előre felépített listát** alkalmaznak. Személy szerint ezt nem tartom jó iránynak, mert a természetes nyelvek nyelvtana dinamikus – ezt talán nem szükséges magyaráznom miért – és bármikor létrejöhetnek új szavak – akár hangulatjelek, mint például a *:)*, *XD* –, amik miatt újra kellene építeni, át kellene gondolni a listát. Ezen kívül **a keresés (lookup) is időbe kerülne**.

Így tehát, ha a pipeline-os módszernél maradok, akkor ez két iteráció lenne két külön osztálykönyvtárral, amik a stacktrace-t is mélyítették volna, tehát mindjárt két dolog miatt is lassabb lett volna az algoritmus. Ez első olvasásra másodrendű fontossággal bírhat, viszont egy ekkora, az előbb bemutatott adathalmaz méretnél rendkívüli fontossággal bír, értékes perceket, órákat lehet vele nyerni.

Továbbá észrevehető, hogy a fájlok tipikusan **ugyanazzal a "(CNN) –" szöveggel kezdődnek**, amik nyilván a forrás portált jelölik meg. Néha található előttük egy városnév is, amik az újság kirendeltségét jelölük, de ez számomra érdektelen, így eltávolítok az első –-ig minden szöveget. Ezt úgy oldottam meg, hogy megkerestem a *(CNN) –* első előfordulását a cikk első sorában (ha volt egyáltalán), ezt követően eltávolítottam mindent addig a pontig.

[7] https://machinelearningmastery.com/prepare-news-articles-text-summarization/

3.3.2. Memóriagazdálkodás

Tapasztalataimból kiindulva **igyekeztem mindenhol listát használni generátorok helyett**. Korábban jobban szem előtt tartottam a memória gazdálkodást, miszerint minél kevesebb adatot szerettem volna a memóriában tartani (mivel ebből nem áll rendelkezésre elegendő a laptopomban).

Azonban ennek az volt a hátránya, hogy egyszer csak **elfogyott a memória tréning közben** és kezdhettem elölről az egészet. Magyarán most azt az utat választottam, hogy ha lehet, akkor **minél hamarabb derüljön ki, hogy az adathalmaz nem fog elférni a memóriában**, hogy ezzel is időt spóroljak.

3.3.3. Az előállított adatstruktúra

Először is, egy **olyan adatstruktúrára törekedtem, amelyet fel tudok használni mind a tanítás, mind pedig a kiértékelési fázisokban is**. Eszerint a mondatok megkülönböztetésére továbbra is szükségem volt, hiszen az általam bemutatott eljárás egyik eszenciája az lesz, hogy **a mondatokat külön-külön is vektorizálom**.

A betöltést, valamint az adattisztítást figyelembe véve egy adott cikkből **egy dictionary adatszerkezetet készítettem el**, ami tulajdonképpen kulcs-érték párokat tárol el. Ezeket pedig a memóriagazdálkádási résznek megfelelően egy listában tárolom.

Először is valahogyan azonosítanom kell a cikkeket. Ehhez az adott $*.story$ állomány **feldolgozási sorrendjét** használtam fel és *id* néven tárolom. A mappában található fájlokat **ábécé sorrendben olvasom fel**, így garantált, hogy a felolvasási sorrend mindig ugyanaz marad. Természetesen új fájlokat már nem adok hozzá, így elcsúszni sem csúszhatnak el a beolvasási azonosítók.

Ezen kívül eltárolom *document* kulcs alatt az adat tisztítás szekcióban taglalt módon az **újságcikket**. A harmadik érték, amit eltárolok, az a *model_summary* kulcs alatt található szöveg, amely a szintén tisztított, **újságcikkíró által megfogalmazott model summary-ket tartalmazza** soremelő karakterrel.

Mindezeket figyelembe véve a következő, JSON-ként ábrázolt adatstruktúrát állítottam elő:

```
{
    "id": "0",
    "document": [
        "its official president barack obama wants
         ↪lawmakers weigh whether use military force
         ↪syria",
        "obama sent letter the heads the house and senate
         ↪ saturday night hours after announcing that
         ↪believes military action against syrian
         ↪targets the right step take over the alleged
         ↪use chemical weapons",
```

```
        // ...
    ],
    "model_summary": "syrian official obama climbed the
     ↪top the tree doesnt know how get down\nobama sends
     ↪ letter the heads the house and senate\nobama seek
     ↪ congressional approval military action against
     ↪syria\naim determine whether were used not whom
     ↪says spokesman",
    "story_stats": {
        "mean": 15.481012658227849,
        "median": 14,
        "max": 44,
        "min": 2
    },
    "highlights_stats": {
        "mean": 9.75,
        "median": 9.0,
        "max": 13,
        "min": 8
    }
}
```

Ebben jól látható, hogy valóban csupa kisbetűs minden szövegelem, valamint az írásjelek is hiányoznak, illetve a szavak csak szóközzel vannak elválasztva. A helytakarékosság érdekében csak **az első két tisztított mondatát hagytam benne a példában**.

Ezen kívül **eltárolok még néhány adatot statisztikai szempontból**. Ilyen például a *story_stats*, ami a dokumentum **mondathosszait** tárolja el szavak szerint, illetve a *highlights_stats*, ami az összefoglaló mondatokról tárol ugyanilyen jellegű információt. A tényleges működés során egyiket sem használom fel, csak **döntéstámogatásban játszottak szerepet** a hiperparaméterek megválasztásánál.

3.3.4. Adathalmaz felosztása

Attól függően, hogy egy beolvasott, preprocesszált dokumentumot hogyan használok fel, több lehetőségem is van. Bár megtehetném azt, hogy a neurális hálózat kiértékelését csak ugyanazzal az adathalmazzal végzem el, mint amivel feltanítottam, nem lenne célravezető, mert **nem tudnék teljes körű következtetést levonni**. Egy viszont idáig biztos: **tanító (tréning) adathalmazra szükség van**.

Előfordulhat például az, hogy úgymond **túltanítom a hálózatot véletlenül**. Ez azt jelenti, hogy a hálózat a bemeneti mintapontokat nagyon jól megtanulja, viszont sosem látott adatra az **általánosító képesség hiánya miatt nem az elvártnak megfelelően reagál**.

Ahhoz, hogy megelőzzem a túltanítást, **a tanítás folyamatát tehát kontrollálni kell valahogy**. Méréseket kell végezni, ami alapján el lehet dönteni, hogy

folytatódhat-e a tanítás vagy sem. Ez egy másik, úgynevezett **validációs adathalmazt** indukál, mely teljesen elkülönül a tréning adathalmaztól.

Az is látható, hogy közvetve bár, de **szerepet játszik a végső hálózat kialakításában**, a hiperparaméterek megválasztásában. Ennek megfelelően egy harmadik, **kiértékelési (teszt) adathalmazra** is szükség van.

Az a konvenció, hogy a tréning adathalmaz általában nagyobb, a másik kettő kisebb. Az én esetemben **a tréning az összes adat 80%-a, a validáció és a teszt adathalmazok pedig 10-10%-ot tesznek ki** a megmaradt adathalmazból.

3.3.5. Erőforrás problémák

Mint később kiderült, a Doc2Vec osztálykönyvtár által, a bemenő adathalmaz alapján **felépített szótár olyan nagyra sikerült, hogy kénytelen voltam átskálázni az eredeti adathalmazt**. Ez azt jelenti, hogy amikor felolvastam $*.story$ fájlokat tartalmazó mappa tartalmát, csak az első valahány százalékát tartottam meg, **a többit már be sem olvastam és később sem használtam fel**.

Bár feltételezhetem, hogy **a szótár mérete logaritmikusan változik** a beolvasott fájlok mennyiségével (hiszen ha beolvastam 100 cikket, kis valószínűséggel lesz új szó a 101.-ben), mégis **sokat javult a tanítási idő, mert egy epoch-ban kevesebbszer kell iterálnom**.

A kevesebb adat miatt viszont **romlott a hálózat generalizáló képessége**. Abban a helyzetben voltam, hogy vagy lesz egy relatíve nagy – de nem a legnagyobb – adathalmazzal feltanított hálózatom, vagy semmilyen.

3.3.6. Folyamat követés

Mivel nagy adathalmazzal dolgozok, sajnos **sokáig tart az adatok preprocesszálása**, valamint a hálózat tanítása is. Fontosnak tartottam valamilyen **folyamat vizualizációt** bevezetni, hogy tudjam, hogy nem-e akadt el valamin a program, illetve lássam, hogy mennyi idő van még vissza a futtatás eredményéig.

Erre a **TQDM Python csomagot választottam**, amely a terminálablak karakteres megjelenítését használja fel az adott folyamat vizualizációjára. További részletekért érdemes felkeresni a GitHub[8] oldalát.

[8] https://github.com/tqdm/tqdm

4. fejezet

Az architektúra

Ebben a fejezetben taglalom a létrehozott Doc2Vec modelleket és a kiértékelésükre alkalmazott módszereket.

4.1. A Gensim Doc2Vec Python könyvtár

Először is fontosnak tartom megemlíteni, hogy magát a Doc2Vec algoritmust – amit a szakirodalmi áttekintésben vázoltam – nem implementáltam le, hanem a **Gensim** (Řehůřek and Sojka, 2010), széles körben felhasznált osztálykönyvtárat használtam fel helyette azért, hogy **elkerüljem az esetleges implementációs hibákat, valamint hogy időt spóroljak**.

Sajnos csak később derült ki, hogy habár TensorFlow-t (Abadi et al., 2015) használ belül az algoritmus, valamiért a készítő **azt nem oldotta meg, hogy GPU-val is lehessen tanítani a neurális hálózatot**, ami feltételezhetően rendkívül lelassította a tanítási folyamatot.

Szerencsére rendelkezésemre állt az Automatizálási és Alkalmazott Informatikai Tanszék számítógépe, aminek ki tudtam használni egyszerre több processzormagját, mert **a szálak számát már meg lehetett adni a Gensim Doc2Vec-nek** – ezúton is köszönet illeti a Tanszéket.

4.2. Felhasznált Doc2Vec modell paraméterek

Egy Doc2Vec modellt az ugyanilyen nevű osztály példányosításával lehet létrehozni. Konstruktorában **számos inicializáló hiperparaméter** megadását teszi lehetővé, melyek közül csak azokat használtam, amelyeket fontosnak ítéltem meg.

A felhasznált paraméterek a következők:

- dm - Definiálja a tanítási algoritmust. Ha értéke 1, PV-DM-et használ, minden más esetben PV-DBOW-t.

- dm_concat - Ha értéke 1, konkatenálja a kontextus vektorokat ahelyett, hogy összegezné vagy átlagolná őket. Megjegyzendő, hogy a konkatenáció egy nagyságrendekkel nagyobb modellhez vezet, mivel a bemenet többé nem egy szóvektor méretű lesz.
- vector_size - A word embedding vektorok mérete.
- negative - Ha nagyobb, mint 0, negatív mintavételezést fog használni. Értéke definiálja, hogy hány szót használjon fel zajként (általában 5 és 20 közé esik).
- hs - Ha értéke 1, hierarchikus softmax függvényt fog használni a model tanítására. Ha értéke 0, és a *negative* értéke nemnulla, negatív mintavételezést fog használni[1]
- min_count - Minden szót kihagy, ami ennél kevesebbszer fordul elő.
- sample - Lebegőpontos küszöbérték, amely megmondja, hogy mekkora valószínűséggel dobjon el a rendszer gyakori szavakat, ezzel is kiegyenlítve kicsit a szavak eloszlását.
- window - A legnagyobb távolság az aktuális és megjósolt szó között egy szekvencián belül.
- workers - Ennyi szálat fog indítani a tanításhoz. Minél több szálat használ, annál gyorsabb lesz a tanítás.
- alpha - Kezdeti tanítási gyorsaság (learning rate).
- min_alpha - Az alpha lineárisan fog csökkenni a tanítás során erre az értékre.
- epochs - Ennyi tanítási iterációt fog menni a korpuszon. Neve megtévesztő, valójában nem epoch, hanem egy epochon belül fog ennyi iterációt végezni.

4.2.1. A létrehozott modellek

Három alapmodellt hoztam létre, amelyek a 4.1. táblázatban tekinthetők meg. Workernek minden esetben az *összes CPU mag-2* értéket használtam. Ahol *Default* érték található, azt a modellre bíztam megállapításra. Mindegyik modell esetében *vector_size*-nak 100-at állítottam be, *negative*-nek 5-öt, *hs*-nek 0-át, *min_count*nak 2-őt, *sample*-nek 0-át, *epochs*-nak 20-at, *alpha*nak 0.05-öt.

	dm	dm_concat	window
PV-DBOW	0	0	Default
PV-DM-1	1	0	Default
PV-DM-2	1	1	5

4.1. táblázat. *A létrehozott alapmodellek*

[1] https://towardsdatascience.com/hierarchical-softmax-and-negative-sampling-short-notes-worth-telling-2672010dbe08

4.2.2. Modellek kombinációja

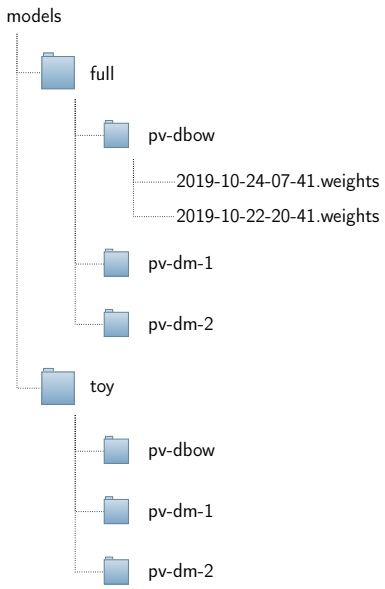
Le és Mikolov (2014) megjegyzi, hogy PV-DBOW modellből származó **paragrafus vektorok kombinációja PV-DM vektorokkal javítja a teljesítményt**. A ConcatenatedDoc2Vec osztály példányosításakor lehetővé teszi, hogy korábban létrehozott Doc2Vec modell példányokat adjak át neki, majd a konkatenálást megoldja magának belülről.

Ennek megfelelően létrehoztam egy *DBOW+DMM* és egy *DBOW+DMC* modellt is, így összesen 5 modell tanítását fogom a későbbiekben bemutatni: **PV-DBOW, PV-DM-1, PV-DM-2, DBOW+DMM, DBOW+DMC**.

4.2.3. A modellek visszatöltésének módja

Természetesen **a modelleket nem elég lementeni, vissza is kell tölteni őket** a háttértárról kiértékelés céljából. Létrehoztam egy *models* mappát, amiben két másik mappa található. Az egyik a *toy*, ami a kis, a másik a *full*, ami pedig a nagy adathalmazzal történő tanítás eredményét tárolja.

A két mappában további almappák találhatók minden Doc2Vec modellhez. A Doc2Vec modelleket **a mentés pillanatában aktuális időponttal mentettem le**, ami így dokumentációs célt is szolgál a fájlnévben. Erre található egy példa a 4.1. ábrán.



4.1. ábra. *Példa az eltárolt modellek rendszerezésére*

A fájlokat **egy mappán belül csökkenő ábécé sorrendbe rendezem és a lista elején lévő fájlt olvasom vissza**, ugyanis ez a legutóbbi modell. Megjegyzendő, hogy azért nincs modell mappa a példával a DBOW+DMM és DBOW+DMC modellekhez, mert ezek a felhasznált modellek súlyaival dolgoznak.

4.3. A szövegösszefoglalók generálásának módszere

A kivonat-alapú szöveg összefoglaló algoritmus viszonylag egyszerű, a *pyrouge_summarize.py* fájlban definiáltam. **Benete maga a Doc2Vec modell** (a nevével együtt, hogy a system summary mappáját meg tudjam határozni), **valamint a dokumentumok**, amiket össze kívánok foglaltatni.

Először is **el kell készítsem a system summary mappáját**. Ha még nem létezik a mappa, akkor létrehozom ezzel a névvel: *pyrouge-input/system-MODELL-NÉV*. Ha már létezett, akkor kitörlöm a tartalmát, nehogy az egyes kiértékelések egymásra hassanak véletlenül.

4.3.1. A dokumentum vektorizálása

Ezt követően végigiterálok az összes dokumentumon, majd **elő kell állítsam a teljes dokumentumhoz tartozó vektort**. Erre **két lehetőségem van**.

Ha a tréning adathalmaz dokumentumaiból szeretnék összefoglalókat készíteni, akkor megtehetem azt, hogy **kiszedem a modellből az adott dokumentumhoz tartozó vektort** a tréningkor felhasznált string tag segítségével. Erre tipikusan akkor van szükségem, ha diagonsztikai célból lefuttatnám a kiértékelő algoritmust a tréning adathalmazra és össze akarom hasonlítani például a teszt adathalmazon készített eredményekkel, hogy lássam, mennyire általánosít jól az algoritmus.

A másik lehetőségem az, hogy **a Doc2Vec modellel kiszámíttatom a vektort**. A publikus API-ján található függvénynek átadom a dokumentum szavait egy listában és az visszaadja a dokumentumhoz tartozó paragrafus vektort. Ez egyébként egy nemdeterminisztikus folyamat a háttérben, ami random faktorokkal is dolgozik, így néha egy kicsit más vektort kapok vissza ugyanarra a dokumentumra – viszont ez nem okoz problémát a kis változás miatt.

4.3.2. A mondatok vektorizálása

Mivel mondatonként tároltam el a dokumentumot, megtehettem azt, hogy az egyes mondatok szavait az előző függvénynek átadjam és visszakapjam a hozzájuk tartozó paragrafus vektorokat.

4.3.3. A mondatok ragsorolása

Az előzőeket összefoglalva tehát végeredményben **ahány mondat van a dokumentumban, annyi vektorom lesz, plusz egy, ami a teljes dokumentumhoz tartozik**.

A lényeg tulajdonképpen az, hogy **a mondatokhoz tartozó vektorokat összeszorzom skalárisan egyenként a teljes dokumentumhoz tartozó vektorral**.

Ehhez előbb minden vektort normalizálnom[2] kell. Egy geometriai példával élve, ha minden vektor 3 dimenziós, a akkor skaláris szorzással nem az adott mondat és a teljes szöveg közti szöget, hanem annak koszinuszát kapom meg. Ennek megfelelően az érték -1 és 1 közé esik, ahol 1 jelenti a teljes egyezést.

Miután minden mondathoz megállapítottam, hogy mennyire hasonlítanak a teljes szövegre, csökkenő sorrendbe rendezem őket. Ezután **megtartom a leghasonlóbb néhány darabot**. Ezt a számot 3-nak állapítottam meg, ugyanis a model summary-k átlagosan ugyanennyi mondatot tartalmaznak.

A kiválasztott mondatokat a ROUGE korábban ismertetett, elvárt formátumának megfelelően a system summary mappájába írom úgy, hogy minden sorban egy mondat legyen. Megjegyzem, hogy mind a model, mind a system summary-kből a tisztított formátumot írom ki fájlokba.

4.4. A tanítási folyamat

4.4.1. Az adatok betöltése

Első lépésként természetesen az adatokat töltöm be, enélkül tanítás sem volna. A 3. fejezetnek megfelelően három részre daraboltam a betöltött adathalmazt, viszont **tanításnál csak a tréning és a validációs adathalmazokra volt szükségem**.

4.4.2. Az adatok előkészítése a Gensim Doc2Vec-nek

A Doc2Vec számára a Gensim által szolgáltatott *TaggedDocument* példányokat kell átadni mind a szótár építéshez, mind a tanításhoz. **Egy TaggedDocument** tulajdonképpen **egy dokumentumot reprezentál, melyhez tag-eket rendelhetünk** (legalább egyet szükséges).

Egy dokumentumot **szavakra bontva** kell átadni (unicode string tokenek listája) és a **tag-eket ugyanilyen formátumban** várja. Utóbbi érdekesebb, ezt az általános felhasználásra találta ki a készítő, ugyanis gyakori eset, hogy például az újságírók a cikkeikhez néhány szavas hashtag-eket emelnek ki. **Az én esetemben azonban a cikk azonosítója** (*id* a preprocesszálás fejezetben) **a tag string formátumban**.

Bár az előző fejezetben mondatokat tároltam el listában tokenizálva, most "ömlesztve" van rájuk szükségem (egy hosszú lista az összes szóval). A szerencsés adattisztítás miatt elegendő volt szóközönként feldarabolnom a szöveget, majd egy trükkös *list comprehension* nyelvi elemmel előállítottam a kért adatformátumot.

[2]https://www.programcreek.com/python/example/100965/gensim.matutils.unitvec

4.4.3. Folyamat követése

A korábban vázolt TQDM könyvtáron kívül fontosnak tartottam **grafikusan is ábrázolni a tanulási folyamatot**. Erre a *matplotlib*[3] Python könyvtárat használtam fel.

Hibakeresési, valamint folyamat vizualizációs szempontból igyekeztem **hasznos információkat kiírni a konzolra**, mint például azt, hogy a Doc2Vec modell elkezdte-e felépíteni a szótárat a tanító adathalmazból, végzett-e már vele, elkezdte-e a tanítást, az elmentett neurális hálózat súlyok helyét.

Két, egymásba ágyazott ciklus hajtja végre a modellek tanítását. A **külső a modelleken** iterál végig, **a belső pedig a tanítási epoch-okon**. Egy epoch futása során a Doc2Vec modell végrehajt egy tanítási lépést az összes dokumentumon a tréning adathalmazból[4].

Bár megadhattam volna a Doc2Vec modellnek egy *min_alpha* értéket, ameddig az *alpha* értéket (ami a learning rate) lineárisan csökkentette volna tanulás során, viszont így nem tudtam volna validálást végezni tanítás során az adathalmazon hitelesen. Éppen emiatt trükközönöm kellett. Korábban említettem, hogy fix alfával indulok minden modell esetében, azonban csökkenteni is szerettem volna azt a tanítás során, hogy jobb eredményt érjek el (magyarázat[5]). Ennek érdekében az epoch-ot végrehajtó ciklus végén **egy kis konstanssal mindig csökkentettem az alfát és a minimumot erre állítottam be**. Ez tulajdonképpen egy kézzel implementált *decaying learning rate* egyszerűbb, nem adaptív változata.

Validáció

A validációt néhány epoch-onként hajtom végre. Úgy oldottam meg, hogy az epoch sorszámát maradékosan elosztom a korábban említett *config.py*-ban található *validation_modulo* konstanssal – ha ez nulla, futtattam a validációt. Tipikusan 5 epochonként végeztem el a műveletet.

Első lépésben egy **pontszámot állapítok meg, amely jellemzi a Doc2Vec modell aktuális állapotát**. Fentebb esett szó arról, hogy **nem áll rendelkezésemre loss, illetve accuracy** érték.

Ahhoz, hogy egy saját accuracy-t meg tudjak állapítani, **le kellett generálnom az aktuális modell alapján a system summary-ket** (a validációs adathalmazban található model summary-ket csak egyszer kellett – ez nem függ egy modelltől sem, csak az adathalmaztól).

Miután ez megvolt, **a pyrouge-nak átadtam a két mappát és legeneráltattam a metrikákat** (24 lebegőpontos szám – lásd 2.3.5. rész). Viszont ezekből **csak az F-score-okat tartottam meg**, így leszűkült 8 számra. **Ezeknek vettem a súlyozott átlagát** aszerint, úgy, hogy azokat a metrikákat vettem nagyobb súllyal

[3] https://matplotlib.org/

[4] https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[5] https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

számításba, amelyek **előnyben részesítik az kivonat-alapú szövegösszefoglaló algoritmussal készült system summary-ket**. **A validációs pontot kiszámoló képlet** (4.1) szerint alakult. Az R rövidítése ROUGE, az F pedig F-score-t jelöl.

$$10*R1F+3*R2F+2*R3F+1*R4F+10*RLF+3*RW12F+2*RSF+3*RSUF \tag{4.1}$$

Még mielőtt elkezdtem volna a tanítást, már készítettem is egy validációs mérést, kíváncsiságból, hogy a tanítás ront-e vagy javít rajta. Meglepő módon azt tapasztaltam, hogy **a tanítás az esetek többségében rontott, aztán elkezdett javulni**. Ezzel kapcsolatban nem találtam dokumentációt, de feltételezem, hogy a hálózat súlyainak inicializációjakor előtanításból származó vektorokat is felhasználtak.

Minden további validáció eredményeként kapott **pontszámot összehasonlítottam a korábbi validáció pontszámával és megnéztem, hogy százalékosan mennyit változott**. Ha a százalék egy **küszöbérték alatt volt, a tanítást leállítottam**, mert azt jelentette, hogy a hálózat már nem tanul újat (vagy lokális minimumra került és módosítani kellene a tanulási rátát).

A másodiktól kezdve minden **validációkor generáltam ábrát, amely a súlyozottan átlagolt F-score-okat ábrázolja**. Ezen kívül, hogy a tanulási trendet jobban meg lehessen állapítani szemrevételezéssel, egy konfigurálható **mozgó átlagot** is rátettem az ábrára.

Fontos megjegyeznem, hogy a kivonat-alapú szövegösszefoglalók kiértékelésénél főleg a recall játszik szerepet, így többször is megfordult a fejemben, hogy csak súlyozott recallokat ábrázolok F-score-ok helyett. Azonban egy összetettebb értéket jobbnak találtam ábrázolni, mert így nyilván több információhoz jutok.

4.5. Kiértékelés

Már fejlesztés közben feltűnt, mikor még kis adathalmazzal próbálkoztam, hogy akármelyik Doc2Vec modellt nézem, **az általánosító képességük nem olyan kifinomult**. Éppen ezért a kiértékelés végző programot úgy készítettem el, hogy **ne csak a teszt adathalmazra tudjon grafikonokat készíteni, hanem akármelyikre**. Később kiderült, hogy az eredmények az elvárásaimnak megfelelően alakultak.

4.5.1. Bemenet és kimenet

A kiértékelést végző fájl neve *main_evaluate.py*. **Bemenete az az adathalmaz, amin a kiértékelést végezem, valamint a modellek, a kimenete pedig 8 kép** (egy-egy minden ROUGE kiértékeléshez), amelyeket a *figures* mappába mentek le *png* formátumban.

Minden képet prefixelem az adathalmaz fajtájával (train, validation vagy test), így nem keverednek össze (persze külön mappákba is generálhattam volna őket, így viszont könnyebb váltogatni közöttük). Így minden fájl neve erre a mintára illeszkedik:

(train/validation/test)_rouge_[1-8].png. A fájl nevének végén lévő szám sorrendben a következő ROUGE kiértékeléseket jelöli: ROUGE-1, ROUGE-2, ROUGE-3, ROUGE-4, ROUGE-L, ROUGE-W-1.2, ROUGE-S*, ROUGE-SU*.

4.5.2. Megvalósítás

Első lépésben természetesen betöltöm a kiválasztott adathalmazt, ugyanis **ki kell írjam a model summary-ket** a korábban vázolt módon egy mappába fájlonként. Külön odafigyeltem, hogy parancssori argumentumként meg lehessen adni, hogy melyik adathalmazt töltse be a program.

Ezután végigiterálok a modelleken, majd a korábban vázolt módon kiírom a system summary-ket az adott modellhez tartozó mappába. Miután ez megvan, a pyrougenak átadom a két mappát és **a kapott eredményekből meghagyom csak a recallt, precisiont és F-score-t minden modellhez**. Ez így 24 lebegőpontos szám. Az olvashatóbb ábrák miatt **mindegyiket felszoroztam 100-zal**. Ezen kívül **az Y tengely maximális értékét mindegyiknél 45-re állítottam**, hogy az oszlopok magassága mindegyik ábrán ugyanúgy szemléltesse az arányokat.

Minden ROUGE kiértékeléshez készítettem egy ábrát, tehát Doc2Vec modellekhez tartozó értékeket ez alapján kellett csoportosítsam. Például tartozik egy ábra a ROUGE-1 szerinti kiértékeléshez, amin mind az 5 Doc2Vec modellhez tartozó recall, precision és F-score értékek láthatók oszlop diagrammon ábrázolva.

5. fejezet

Az eredmények kiértékelése

Ebben a fejezetben az elkészített Doc2Vec modelleket kiértékelem és összehasonlítom egymással is és néhány state-of-the-art megoldással.

5.1. A tanulási folyamat

Az előző fejezetben ismertettem, hogy **három olyan Doc2Vec modellt készítettem, amelyet tanítani kellett, valamint kettő másikat, ami a három tanított konkatenációja**. Először **szemrevételezem a tanított modellek teljesítményét** a tanulás során. Mint korábban említettem, a következő grafikonokon látható adatok **a validációs adathalmazon végzett időközi kiértékelések eredményei**.

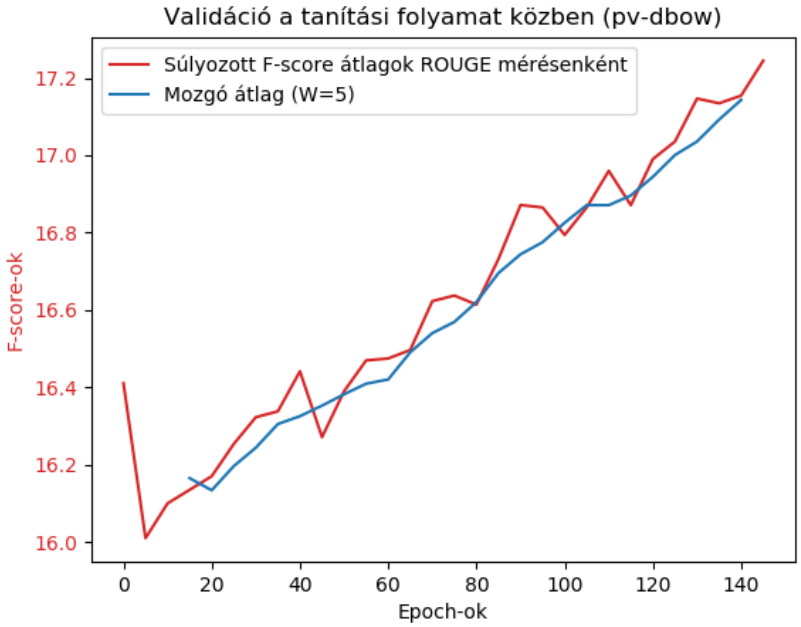
Bár az ábrákból nem feltétlenül látszódik, **mindegyik modellt 150 epoch-on keresztül tanítottam, körülbelül 10 órán keresztül külön-külön**. Emellett ismételten kiemelném, hogy **nem a teljes adathalmazzal dolgoztam**, mert a szótárméret miatt a tanítással nem végeztem volna belátható időn belül. Ebből eredően **a teszt adathalmazon történő kiértékelés sem sikerülhetett olyan jól, hiszen a hálózat ismeretlen szavakkal találkozott**.

5.1.1. PV-DBOW

Mivel a 5.1. grafikonon jól látható, hogy határozottan, **lineárisan javul a szövegösszegzési minőség a tanítás során**, itt nem mond sokat a mozgó átlag. Azonban jól kivehető a grafikonból, hogy **tanítás nélkül is viszonylag jó eredményt ért el a hálózat**, majd az első tanítás után bezuhant és körülbelül 40 epoch-nak el kellett telnie ahhoz, hogy visszamenjen az eredeti szintre.

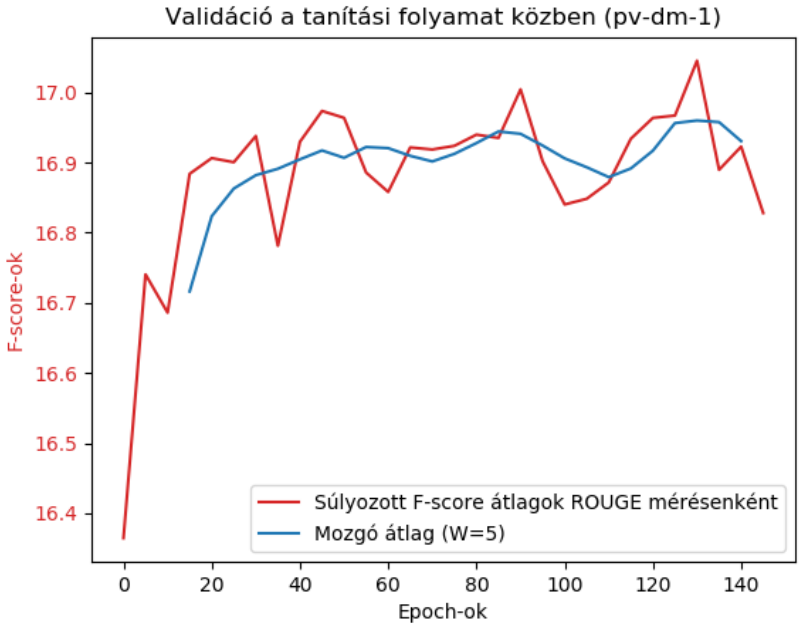
5.1.2. PV-DM-1

Ennél (5.2. ábra) a modellél nem fordult elő az, ami az elsőnél, mégpedig hogy romlott volna a szövegösszegzés minősége, sőt. Viszont szerintem **ez a modell kevésbé ígéretes, mint az előző**, mivel a mozgó átlagból látszódik, hogy még csak



5.1. ábra. *A PV-DBOW modell tanulása*

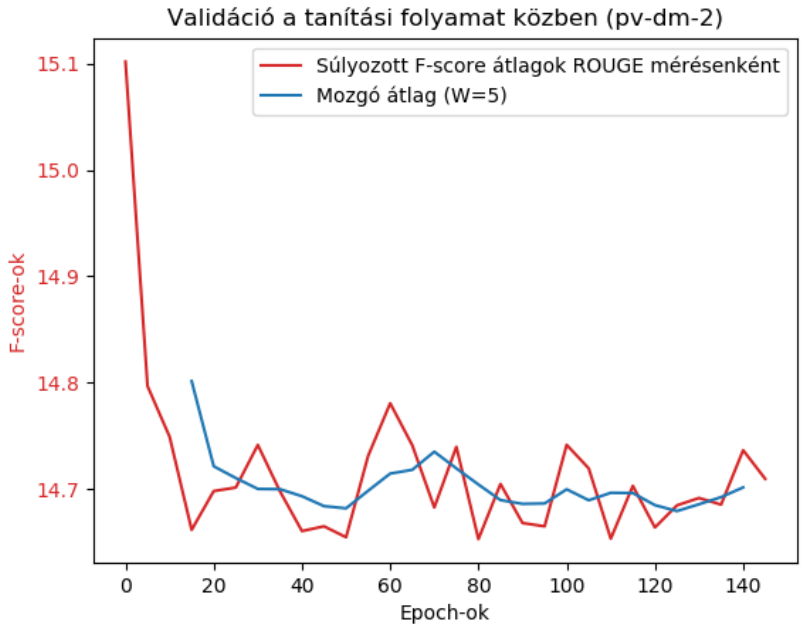
lineárisan sem javul az epoch-ok elteltével a hálózat teljesítménye. Valójában 30 epoch után akár abba is lehetett volna hagyni a tanítást és körülbelül ugyanolyan eredményt ért volna el, mint az első 110 epoch után.



5.2. ábra. *A PV-DM-1 modell tanulása*

5.1.3. PV-DM-2

Az 5.3. ábrán látható, hogy a helyzet itt még érdekesebb, ugyanis jól kivehető, hogy **tanítás következtében szinte csak romlott a hálózat teljesítménye**. Az elején egy nagy esés következett be, majd utána a mozgóátlag alapján látható, hogy szinte állandó maradt a teljesítménye.



5.3. ábra. *A PV-DM-2 modell tanulása*

5.2. A kiértékelési grafikonok

Habár eredetileg úgy terveztem, hogy csak a teszt adathalmaz segítségével készített kiértékeléseket mutatom be tanulmányomban, úgy döntöttem, hogy **a tanítási adathalmazzal készítettet is belefoglalom összehasonlítási alapként**. A következőkben három tényt emelnék ki a grafikonok alapján.

Az első az, hogy **a recall érték szinte minden grafikonon magasabb a precision és F-score értékeknél**. Ez a metrika számítási módja miatt van, az indokot a 2.3.3 alfejezetben részleteztem.

A második, amit megemlítenék pedig az, hogy **a hálózatok generalizáló képessége elmarad az elvártnál**. Utóbbit visszavezetném a tanításra, hiszen **jól látszódott, például az első modell esetében, hogy több tanítással jóval nagyobb teljesítményre lenne képes**.

Végül pedig levonom a következtetést, hogy **a Doc2Vec modellek közül nem igazán tudnék győztest kihirdetni, hiszen egyik sem teljesíti túl számottevően a többi modellt** – kivéve talán a PV-DM-2. Megjegyezném, hogy a konkatenált modellek sem emelkedtek ki a többi közül, így **a jövőben nem próbálkoznék velük**.

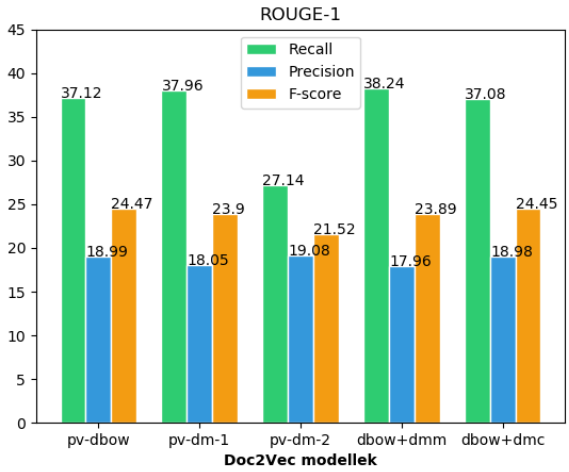
5.2.1. Összehasonlítás state-of-the-art modellekkel

A szakirodalmi áttekintés fejezetben említést tettem a PapersWithCode[1] oldalon található táblázatról. Mint kiderült, a ROUGE-1, ROUGE-2, ROUGE-L oszlopok **az F-score-okat tartalmazzák**.

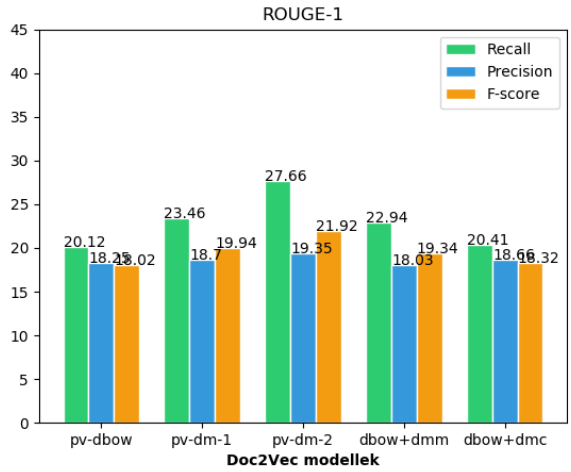
Ha összehasonlítom az eredményeket, akkor az tapasztalható, hogy **bár a modelljeim F-score-jai alacsonyabbak – a 9. helyet foglalnák el a táblázatban –, a recall alapján ígéretesnek tűnnek a tanító adathalmazon**. Viszont sajnos nem tudom elhelyezni a táblázatban a modelleket, mert a recall értékeket nem tartalmazza a táblázat – feltételezhetően azért, mert **a legtöbb publikációban nincs róla szó**.

Ez jó példa arra, hogy ha még sosem látott adathalmazzal dolgoznak, akkor nem tűnnek ki, viszont **ha a tréning adathalmazt szeretnénk összefoglaltatni velük, akkor alkalmazhatók**.

[1] https://paperswithcode.com/sota/document-summarization-on-cnn-daily-mail

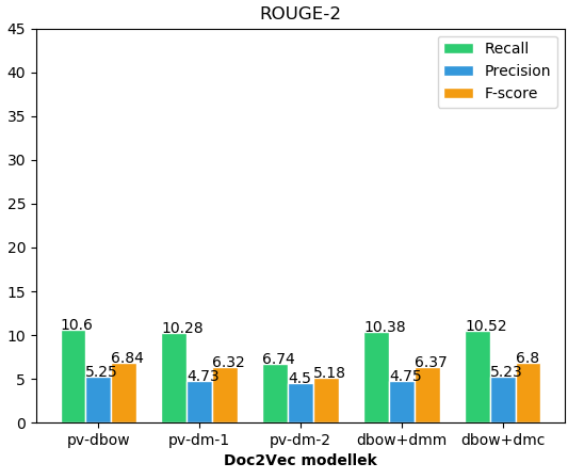


(a) A tanítási adathalmazon

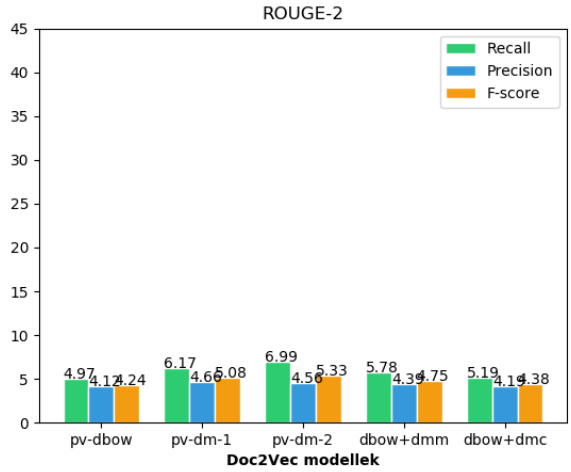


(b) A teszt adathalmazon

5.4. ábra. *ROUGE-1 szerinti kiértékelés*

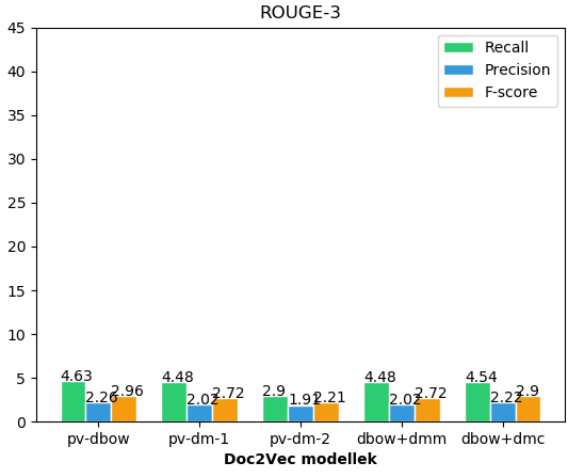


(a) A tanítási adathalmazon

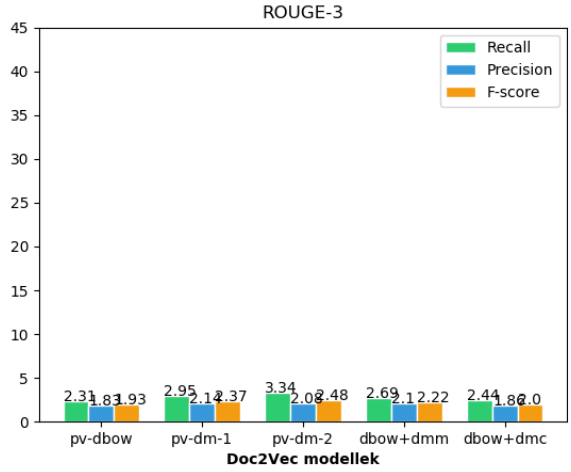


(b) A teszt adathalmazon

5.5. ábra. *ROUGE-2 szerinti kiértékelés*

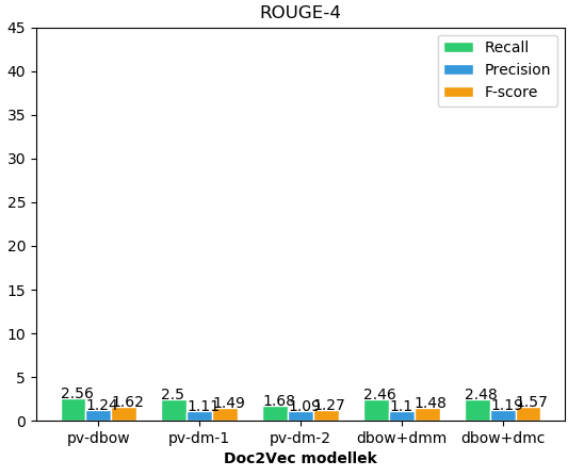


(a) A tanítási adathalmazon

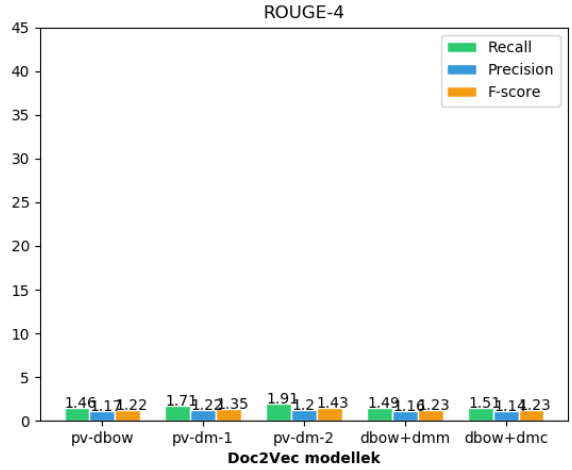


(b) A teszt adathalmazon

5.6. ábra. *ROUGE-3 szerinti kiértékelés*

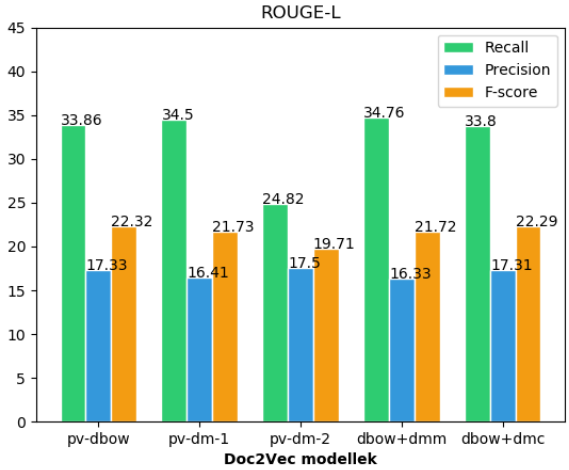


(a) A tanítási adathalmazon

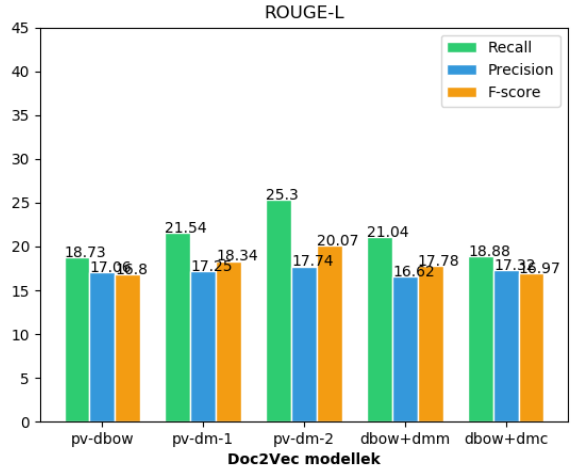


(b) A teszt adathalmazon

5.7. ábra. *ROUGE-4 szerinti kiértékelés*

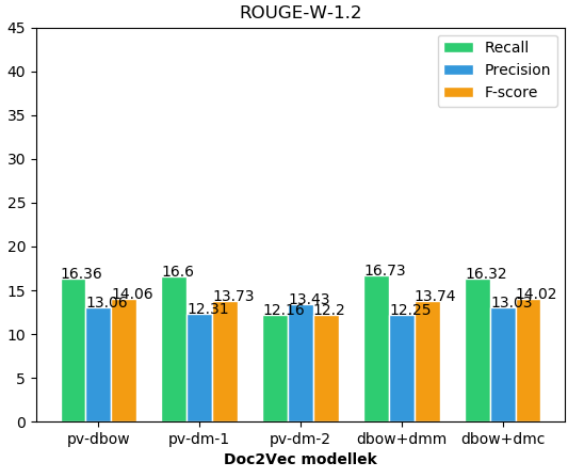


(a) A tanítási adathalmazon

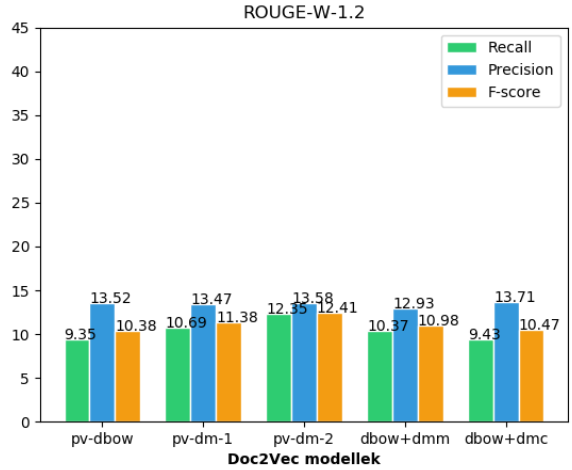


(b) A teszt adathalmazon

5.8. ábra. *ROUGE-L szerinti kiértékelés*

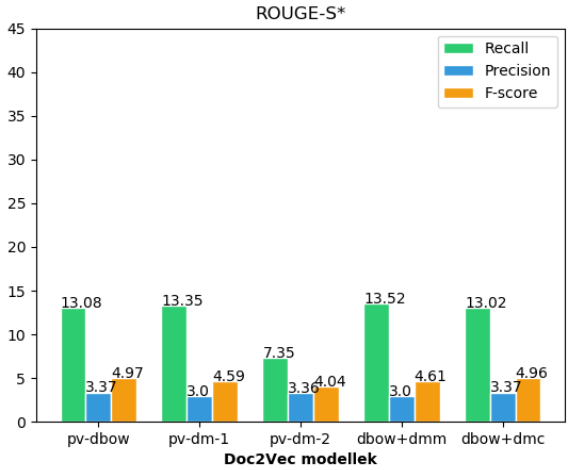


(a) A tanítási adathalmazon

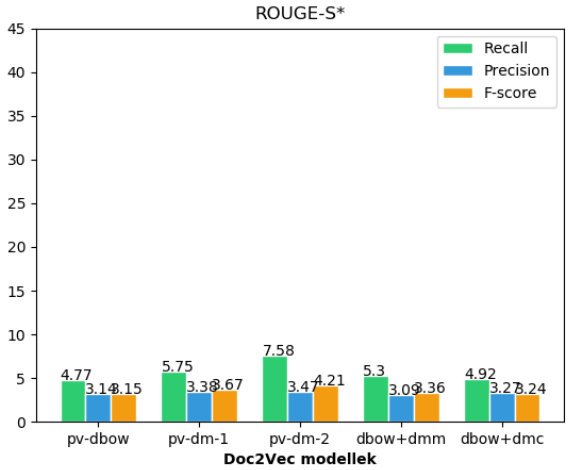


(b) A teszt adathalmazon

5.9. ábra. *ROUGE-W-1.2 szerinti kiértékelés*

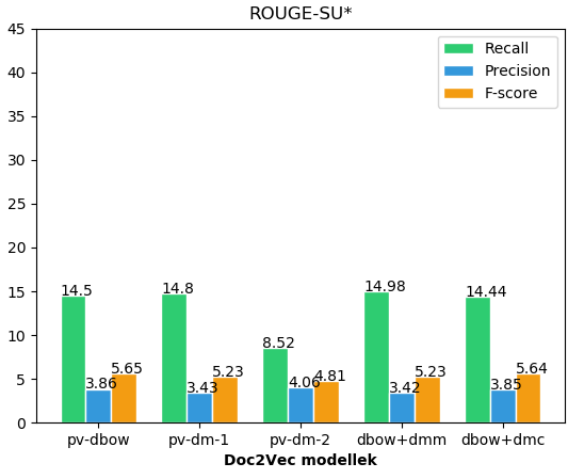


(a) A tanítási adathalmazon

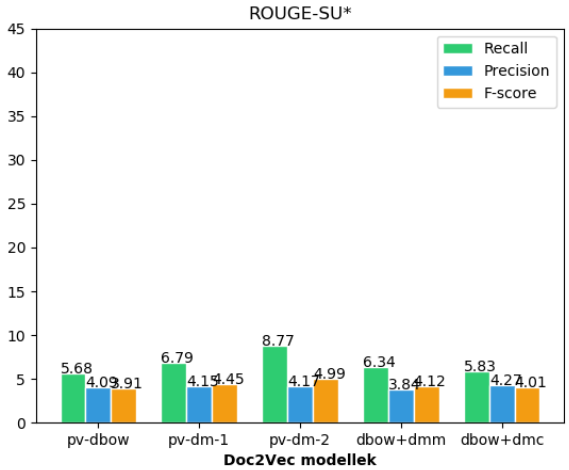


(b) A teszt adathalmazon

5.10. ábra. *ROUGE-S* szerinti kiértékelés*



(a) A tanítási adathalmazon



(b) A teszt adathalmazon

5.11. ábra. *ROUGE-SU* szerinti kiértékelés*

5.3. Példa összefoglalók

Vegyük alapul a következő újságcikket, melyből a bekezdéseket elválasztó üres sorokat helytakarékosság miatt kivettem:

(CNN) – Usain Bolt rounded off the world championships Sunday by claiming his third gold in Moscow as he anchored Jamaica to victory in the men's 4x100m relay. The fastest man in the world charged clear of United States rival Justin Gatlin as the Jamaican quartet of Nesta Carter, Kemar Bailey-Cole, Nickel Ashmeade and Bolt won in 37.36 seconds. The U.S finished second in 37.56 seconds with Canada taking the bronze after Britain were disqualified for a faulty handover. The 26-year-old Bolt has now collected eight gold medals at world championships, equaling the record held by American trio Carl Lewis, Michael Johnson and Allyson Felix, not to mention the small matter of six Olympic titles. The relay triumph followed individual successes in the 100 and 200 meters in the Russian capital. "I'm proud of myself and I'll continue to work to dominate for as long as possible," Bolt said, having previously expressed his intention to carry on until the 2016 Rio Olympics. Victory was never seriously in doubt once he got the baton safely in hand from Ashmeade, while Gatlin and the United States third leg runner Rakieem Salaam had problems. Gatlin strayed out of his lane as he struggled to get full control of their baton and was never able to get on terms with Bolt. Earlier, Jamaica's women underlined their dominance in the sprint events by winning the 4x100m relay gold, anchored by Shelly-Ann Fraser-Pryce, who like Bolt was completing a triple. Their quartet recorded a championship record of 41.29 seconds, well clear of France, who crossed the line in second place in 42.73 seconds. Defending champions, the United States, were initially back in the bronze medal position after losing time on the second handover between Alexandria Anderson and English Gardner, but promoted to silver when France were subsequently disqualified for an illegal handover. The British quartet, who were initially fourth, were promoted to the bronze which eluded their men's team. Fraser-Pryce, like Bolt aged 26, became the first woman to achieve three golds in the 100-200 and the relay. In other final action on the last day of the championships, France's Teddy Tamgho became the third man to leap over 18m in the triple jump, exceeding the mark by four centimeters to take gold. Germany's Christina Obergfoll finally took gold at global level in the women's javelin after five previous silvers, while Kenya's Asbel Kiprop easily won a tactical men's 1500m final. Kiprop's compatriot Eunice Jepkoech Sum was a surprise winner of the women's 800m. Bolt's final dash for golden glory brought the eight-day championship to a rousing finale, but while the hosts topped the medal table from the United States there was criticism of the poor attendances in the Luzhniki Stadium. There was further concern when their pole vault gold medalist Yelena Isinbayeva made controversial remarks in support of Russia's new laws, which make "the propagandizing of non-traditional sexual relations among minors" a criminal offense. She later attempted to clarify her comments, but there were renewed calls by gay rights groups for a boycott of the 2014 Winter Games in Sochi, the next major sports event in Russia.

A 19 mondatból álló cikk átlagosan 22 szavas mondatokból áll, a medián szintén 22. A leghosszabb mondat 35, a legrövidebb pedig 10 szavas.

Az újságíró által megfogalmazott összefoglaló mondatok összefűzése alkotta a model summary-t:

> Usain Bolt wins third gold of world championship Anchors Jamaica to 4x100m relay victory Eighth gold at the championships for Bolt Jamaica double up in women's 4x100m relay

A következő összefoglalókat **a tréning adathalmazban részt vett cikkből generáltam**, mert úgy tapasztaltam, hogy így racionálisabb eredményt kapok. A fejezet korábbi részeiben látható volt a diagrammokon, hogy **a hálózatok generalizálási képessége nem feltétlenül tenné lehetővé az elfogadható összefoglalók generálását**.

5.3.1. System summary a PV-DBOW modellel

> In other final action on the last day of the championships, France's Teddy Tamgho became the third man to leap over 18m in the triple jump, exceeding the mark by four centimeters to take gold. Their quartet recorded a championship record of 41.29 seconds, well clear of France, who crossed the line in second place in 42.73 seconds. (CNN) – Usain Bolt rounded off the world championships Sunday by claiming his third gold in Moscow as he anchored Jamaica to victory in the men's 4x100m relay.

Mondat sorszám	Koszinusz távolság	Szavak száma
14	0.2636	27
10	0.2456	16
1	0.239	19

Az átlagos mondathossz 21, a medián 19. A legrövidebb mondat 16, a leghosszabb 27 szóból áll.

5.3.2. System summary a PV-DM-1 modellel

> The U.S finished second in 37.56 seconds with Canada taking the bronze after Britain were disqualified for a faulty handover. She later attempted to clarify her comments, but there were renewed calls by gay rights groups for a boycott of the 2014 Winter Games in Sochi, the next major sports event in Russia. (CNN) – Usain Bolt rounded off the world championships Sunday by claiming his third gold in Moscow as he anchored Jamaica to victory in the men's 4x100m relay.

Az átlagos mondathossz 20, a medián 19. A legrövidebb mondat 16, a leghosszabb 26 szóból áll.

Mondat sorszám	Koszinusz távolság	Szavak száma
3	0.401	16
19	0.3812	26
1	0.3422	19

5.3.3. System summary a PV-DM-2 modellel

> Germany's Christina Obergfoll finally took gold at global level in the women's javelin after five previous silvers, while Kenya's Asbel Kiprop easily won a tactical men's 1500m final. There was further concern when their pole vault gold medalist Yelena Isinbayeva made controversial remarks in support of Russia's new laws, which make "the propagandizing of non-traditio nal sexual relations among minors" a criminal offense. Kiprop's compatriot Eunice Jepkoech Sum was a surprise winner of the women's 800m.

Mondat sorszám	Koszinusz távolság	Szavak száma
15	0.1833	24
18	0.1729	30
16	0.1033	10

Az átlagos mondathossz 21, a medián 24. A legrövidebb mondat 10, a leghosszabb 30 szóból áll.

5.3.4. System summary a DBOW+DMM modellel

> Earlier, Jamaica's women underlined their dominance in the sprint events by winning the 4x100m relay gold, anchored by Shelly-Ann Fraser-Pryce, who like Bolt was completing a triple. Germany's Christina Obergfoll finally took gold at global level in the women's javelin after five previous silvers, while Kenya's Asbel Kiprop easily won a tactical men's 1500m final. The 26-year-old Bolt has now collected eight gold medals at world championships, equaling the record held by American trio Carl Lewis, Michael Johnson and Allyson Felix, not to mention t he small matter of six Olympic titles.

Mondat sorszám	Koszinusz távolság	Szavak száma
9	0.6614	22
15	0.6311	24
4	0.6291	31

Az átlagos mondathossz 26, a medián 24. A legrövidebb mondat 22, a leghosszabb 31 szóból áll.

5.3.5. System summary a DBOW+DMC modellel

> Earlier, Jamaica's women underlined their dominance in the sprint events by winning the 4x100m relay gold, anchored by Shelly-Ann Fraser-Pryce, who like Bolt was completing a triple. (CNN) – Usain Bolt rounded off the world championships Sunday by claiming his third gold in Moscow as he anchored Jamaica to victory in the men's 4x100m relay. Germany's Christina Obergfoll finally took gold at global level in the women's javelin after five previous silvers, while Kenya's Asbel Kiprop easily won a tactical men's 1500m final.

Mondat sorszám	Koszinusz távolság	Szavak száma
9	0.7432	22
1	0.6955	19
15	0.6884	24

Az átlagos mondathossz 22, a medián 22. A legrövidebb mondat 19, a leghosszabb 24 szóból áll.

6. fejezet

Konklúzió és Továbbfejlesztési Lehetőségek

6.1. Konklúzió

Tanulmányom keretén belül sikeresen **elkészítettem öt Doc2Vec modellt**, amik segítségével hosszabb szövegeket lehet lerövidíteni a kohézió figyelembevételével. Ezekről többféle **méréseket készítettem**, melyeket grafikusan ábrázoltam.

Érdemes megjegyezni, hogy az adatfelhasználás rétegződése miatt a Doc2Vec modelleket **akár ki is lehetne cserélni** valami egészen másra – ami ugyanebben a formában adja ki a számadatokat –, ugyanis maga a rangsoroló algoritmus (koszinusz távolságok megállapítása, majd ez alapján rangsorolás) eléggé rugalmas ahhoz, hogy **felhasználható legyen máshol is**.

Néhány tízes nagyságrendű cikk halmazt **percek leforgása alatt szinte tökéletesen sikerült megtanulnia**, viszont olyan cikkekre nem működött jól, amiket tanítás során nem látott – legyen szó a validációs vagy a teszt adathalmazról. Ez arról árulkodik, hogy **a generalizáló képességük nem olyan jó**.

Ezt a tényt viszont **semmiképp sem nevezném hátránynak**, hiszen gondoljunk bele, hogy máris mekkora, ésszel fel nem fogható cikk, könyv, filmfelirat, stb. adathalmaz áll rendelkezésre a világban. Ha például az a célunk, hogy *meglévő* könyveket szeretnénk kivonatolással lerövidíteni, vagy *meglévő* cikkekben szeretnénk a fontos részeket kiemelni – amire jó analógia a szövegkiemelő használata –, tökéletesen alkalmazható a megoldás.

Természetesen elképzelhető, hogy **több tanítással jobb eredményt érhettek volna el a hálózatok**, viszont az előbbi példa miatt nem látnám szükségszerűnek.

6.1.1. Érdekesség a generált összefoglalókról

Annak érdekében, hogy az olvasó egy cikket végig olvasson, feltételezhetjük, hogy **az elején kell valamilyen fontos információnak lennie**. Érdekes egybeesés, az előző fejezetben található összefoglalók többségében **beválogatták az első mondatot a három legérdekesebb mondatba** – ez látszódik a *(CNN)* előtagból is.

Emellett az is említésre méltó, hogy **tipikusan hosszabb mondatokat válogatott ki az algoritmus**, valószínűleg azért, mert ezekben szerepel a legtöbb információ. Valóban, statisztikailag racionális, hogy a hosszabb mondatok több információt tartalmaznak.

6.2. Továbbfejlesztési lehetőségek

Érdekes volna egy olyan kísérletet is végezni, amiben a neurális hálózat egy **kevésbé megtisztított adathalmazon** is tanult, hogy össze lehessen hasonlítani az információveszteség hatását a neurális hálózat teljesítményében.

Ezen kívül nagyon érdekes lenne, ha **nem csak mondatokat rangsorolnék a dokumentumokon belül, hanem szavakat tartalmazó n-grammokat többféle n-re**. Habár itt a kohézióval problémák lennének, szép alkalmazása lenne egy **Chrome plugin**, amely kiemelné a fontosabb részegységeket egy eredeti formátumában megjelenített – például – TDK dolgozatban. Ezzel tulajdonképpen ajánlásokat is lehetne adni a felhasználónak, hogy **mely részekre érdemes jobban odafigyelnie**.

Sőt, akár **színekkel is lehetne bővíteni a kiemelést**. Például a legfontosabb részeket pirossal, a kevésbé fontosakat kékkel emelné ki, és a kettő között **színátmenetet** lehetne húzni. Ezek alapján felállítható volna egy hőtérkép (heatmap) a szöveg egyes részeinek fontossága alapján.

Ezek megvalósításához természetesen **több tanításra lenne szüksége a hálózatnak**, komolyabb gépeken, több szálon. Továbbfejlesztési lehetőségnek éppen ezért azt is felvetném, hogy a Doc2Vec interfészét felhasználva **egy olyan implementációt helyeznék a jelenlegi helyébe, amely képes GPU-val gyorsítva tanulni**, ez nagyon sokat dobna az eredményeken korábbi tapasztalataimból kiindulva.

Érdekes lenne még **adaptív tanulási rátával** is próbálkozni a lineárisan csökkenő helyett, ugyanis más gyakorlati alkalmazásban tapasztaltam, hogy ígéretes eredményeket lehet vele produkálni.

Köszönetnyilvánítás

Ezúton köszönöm konzulensemnek, Dr. Szegletes Lucának féléves, kitartó munkáját. Segítőkészsége és töretlen bátorítása nélkül egy határozottan sokkal rögösebb utat kellett volna végigjárnak. Gondolok itt például a Doc2Vec megemlítésére, a rendszeres konzultációs alkalmakra, valamint az ezen kívüli segítségekre Slacken – csak hogy néhányat említsek.

Ezen kívül köszönettel tartozom Recski Gábornak, aki iránymutatást adott tanulmányom témájával kapcsolatban, illetve Ács Juditnak, aki bevezetett a természetesnyelv feldolgozás világába.

Végül köszönettel tartozom mindenkinek, akitől valaha bármit is tanulhattam.

Acknowledgement

I would like to sincerely thank my university supervisor, Luca Szegletes, PhD her constant support throughout the whole semester. Without her helpfulness and encouragement, I would have had to deal with many more blockers. I am thinking, for example, of mentioning Doc2Vec, regular consultation sessions, and outside help on Slack - just to name a few.

In addition, I would like to thank Gábor Recski, who provided guidance on the topic of my study, and Judit Ács, who introduced me to the world of natural language processing.

Finally, I owe thanks to everyone, who I could ever learn anything from.

Irodalomjegyzék

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL `http://arxiv.org/abs/1506.03340`.

Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. 2014.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W04-1013`.

Yang Liu and Mirella Lapata. Text summarization with pretrained encoders, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL `http://arxiv.org/abs/1301.3781`.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.