FELADATKIÍRÁS

Az autoencoder a deep learning egyik népszerű architektúrája nem felügyelt tanulásra, amit egyfajta tömörítési eljárásként is lehet értelmezni. A hallgató feladata magyar nyelvű szavak rekonstrukciója autoencoder, illetve variational autoencoder segítségével.

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

Velkey Géza

AUTOENCODERES KÍSÉRLETEK

Magyar nyelvű szavak tömörítése és rekonstrukciója

KONZULENS

Ács Judit

BUDAPEST, 2017

Tartalomjegyzék

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Autoencoders are neural networks which aim to reconstruct their input with the least amount of distortion. Using autoencoders eliminates the need for – often expensive – labeled training samples, by turning an unsupervised learning setting into a supervised one. In the simplest case, an autoencoder is a feed forward neural network with one or more hidden layers which are typically much smaller than the network's input layer, creating a compressed representation of the input. We present a series of autoencoder experiments using Hungarian words as their input. Our architectures include deep autoencoders with more than one hidden layer and variational autoencoders. We also experiment with different preprocessing steps such as replacing di- and trigraphs with a single character. Our error analysis of the reconstruction errors gives insight into frequent morphological paradigms occuring in Hungarian.

1 Bevezetés

A gépi tanuló algoritmusok alkalmazásai körében sok példa van arra, hogy kulcsfontosságú a bemeneti adatok megfelelő reprezentálása. A mély neurális hálózatok ebben bizonyítottan jól teljesítenek [1], ezek közül pedig nagy népszerűségnek örvendenek az autoencoderek, melyek felépítése igen egyszerű, a bemenetüket próbálják visszaadni a kimenetükön.

A számítógépekkel történő nyelvfeldolgozás során gyakran kerülünk olyan helyzetbe, hogy a bemenet dimenziója nagyon nagy, mint például egy nyelv összes szavát tartalmazó szótár esetén, amit általában úgy oldanak meg, hogy egy embedding réteg csökkenti a bemenet dimenzióját és word vectorokat készít a szavakból. Ebben az önálló laboratórium feladatban több fajta szó reprezentációval is kísérleteztünk magyar nyelven.

2 Kapcsolódó Munkák

Az autoencoderek első alkalmazása [2] a Principal Component Analysis (PCA) nemlineáris általánosítása volt. Rájöttek, hogy az autoencoderek teljesítménye nagyban függ az inicializásuktól is, és Restricted Boltzmann Machine-t (RBM) alkalmaztak a neurális hálózatok előtanítására.

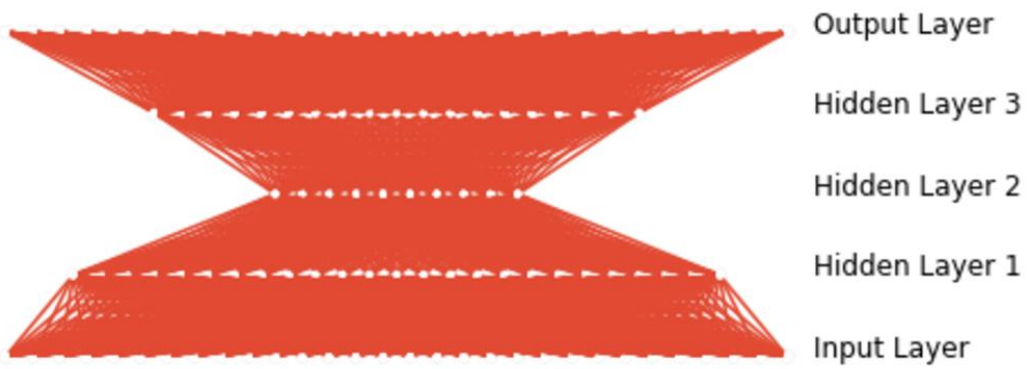
Az autoencodereket sok különböző területen alkalmazták mind NLP, mind képfeldolgozás és számítógépes biológiai adatfeldolgozó alkalmazásokban is. [3] sentiment analízisre használta a hálót, [4] egy jóval több rétegből álló autoencoderrel angol- kínai fordítást valósított meg. [5] rekurrens autoencodert készített LSTM cellákból, mellyel bekezdéseket állított vissza word vectorokból. [6] sikeresen használta az architektúrát információ visszaállítására.

A legújabb eredmények között található [7], mely egy Split-Brain architektúrát használ, amivel mi is több kísérletet végeztünk magyar nyelven.

Mivel az eddig NLP-vel kapcsolatos eredmények főként bekezdések, mondatok visszaállításra és word vectorok alkalmazására voltak kialakítva, a magyarhoz hasonló nyelvek szavakon belüli struktúráját nem használták fel, mely fontos addicionális információt tartalmaz.

3 Architektúra

Az autoencoderek működésük alapján két részre bonthatóak, ezek az enkóder és a dekóder. Az enkóder végzi a tömörítést, és a legkisebb réteg tartalmazza a bemeneti információt jóval kevesebb dimenzióra leképezve. A dekóder az enkódolt információból képezi vissza a bemenetet.



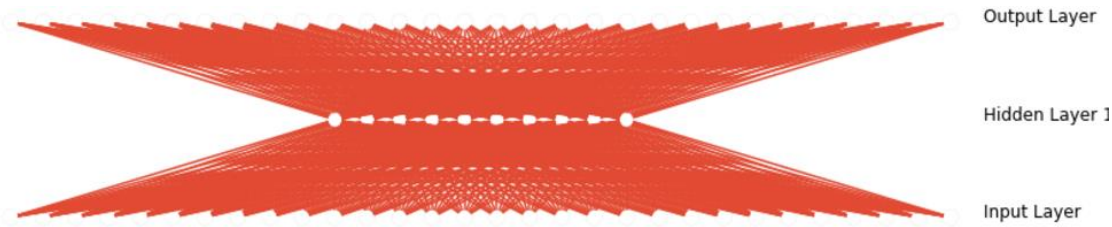
A képen látható egy tipikus felépítés, a hidden layer 2 az enkóder

A mi kísérleteink során fully connected és variational autoencoderekkel foglalkoztunk, és teszteltük a legfrissebb architectúrát, melyet split-brain autoencodernek neveznek.

Az osztályok interfészei megegyeznek, sok közös metódus van, emiatt könnyen megoldható örökléssel a különböző kísérletek létrehozása. A deep learning támogató library a Google által fejlesztett és karbantartott Tensorflow volt. Erre jellemző, hogy először létrehozunk egy computational graph-ot, melyen később futtatunk számításokat, mint a train, predict, loss számítás, stb. A különböző kísérleti osztályok között a lényegi különbség azon függvények között van, melyek a graph felépítését, és a súlyok inicializálását végzik. A közös ősük (Autoencoder) egy abstract osztály, melynek create_graph függvénye pure virtual, azaz minden leszármazottjának implementálnia kell. Az autoenkóderek alapvetően a bemenetüket próbálják reprodukálni, ezért a train függvény nem is kér label-eket a bemenetéhez. A későbbi kísérletek érdekében viszont mégis hozzá lehet adni kimenetet is adott bemenetekhez, így használható több célra is az autoencoder és tudjuk tesztelni a zajérzékenységét, valamint, hogy más feladatkörökben, ahol nem feltétlen egyezik az input és output mennyire teljesít jól az adott modell.

3.1 Fully Connected Autoencoder

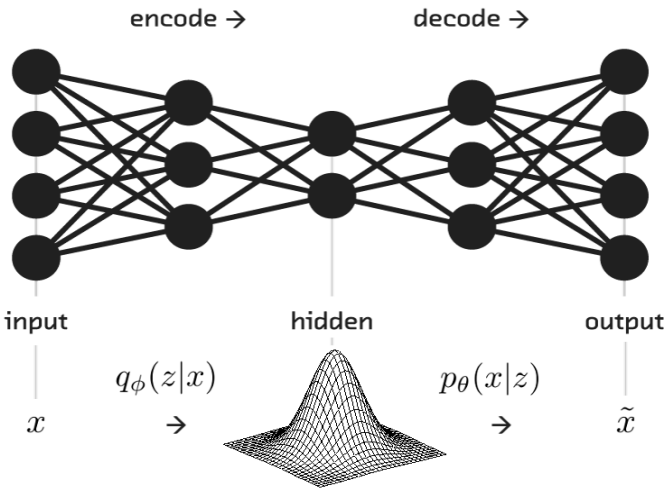
Ezen kísérleti osztály (Autoencoder_FFNN) felépítése a legegyszerűbb, közvetlenül az Autoencoder osztályból származik. A felépítése megegyezik egy fully connected neurális hálóval, a különbség annyi, hogy a rétegek az enkóder rétegig egyre kevesebb neuronból állnak, majd az után pedig egyre nagyobbakból, és végül a kimenetének ugyanannyi a dimenziója mint a bemenetnek. A hálóban kísérletfüggően minimum 1, maximum 7 rejtett réteg van. Az utolsó kivételével minden réteg ugyanolyan felépítésű, tartalmaz egy mátrixszal való szorzást és egy nonlinearitás függvényt. Az utolsó rétegnél a nemlinearitás csak információvesztéssel járna ezért nem alkalmazzuk. A tanítása során elegendő a bemenetet adni a modellnek, mivel ilyenkor automatikusan hozzárendeli kimenetként azt is, és tanulja a legjobb tömörítést az encoding rétegre a hiba visszapropagálásával. A hibafüggvény az Euclidesi norma alapján számítódik a háló kimenete és a megfelelő kimenet között.



Itt az egyetlen rejtett réteg az enkóder

3.2 Variational Autoencoder

A variational autoencoder (Autoencoder_Variational osztály) is fully connected rétegekből áll, azonban a veszteségfüggvényében lényegesen eltér az egyszerű autoencodertől. Ebben az esetben a veszteségfüggvényünk két függvénynek az összege. Az egyik mutatja, hogy mennyire helyesen reprezentáljuk az outputot, míg a másik az mutatja, hogy a rejtett réteg mennyire közelíti a normális eloszlást. Így ez a hálózat is az tanulja meg, hogy hogyan reprezentálja a kimenetét a legjobban, de eközben a látens réteg közelíti a Gauss-eloszlást.

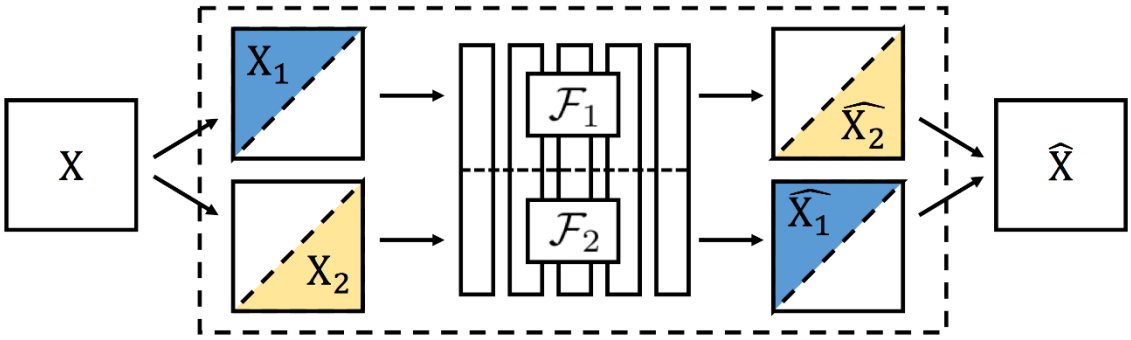


Az ábrán látható a Gauss-kényszerítés a középső rétegen

Itt megjelenik az a probléma, hogy ha a hálózat kimenete nagyon jól közelíti a bemenetét, a rejtett réteg el fog térni a kívánt eloszlástól, és a veszteségben tanítás közben oszcilláció állhat be, vagy be is akadhat korábban a tanítás.

Ez a modell várhatóan kevésbé lesz érzékeny a bemeneti zajokra, és a köztes réteget változtatva új magyar nyelvbe illő szavak kreálhatóak, szimplán azzal az egyszerű lépéssel, hogy mintákat veszünk a normál eloszlásból, és ezt adjuk a dekóder bemenetére.

3.3 Split-Brain Autoencoder



A Split-Brain Autoencoder felépítése [7]

Ennél az architektúránál az autoencoder két, lényegében nem autoencoder szerepet betöltő párhuzamos neurális hálóból áll. Mindkét háló feladata az, hogy a kép másik felét jósolják meg, így a kimenetből is összeállhat a teljes információ. Ezzel a módszerrel sokkal nehezebb feladatuk van a hálóknak, és itt rá vannak kényszerítve, hogy a bemenetük logikus felépítésének közelítését, a mintákat tanulják meg. Ezt úgy

alakítottuk ki, hogy az abstract ősből leszármaztattunk egy hálót, melynek a bemenete és kimenete már nem egyezik meg, és az inputokat pedig vagy karakterenként, vagy pozíciónként szétválasztottuk, és így kellett a hálónak visszaalakítania a bemenetét.

3.4 Segmentation

Az előbbi architektúrák eredményei alapján új ötletek születtek az ilyen autoencoder jellegű architektúrák alkalmazásaira. Az egyik NLP-beli felhasználás a szegmentálás, avagy szavak tagolása. Erre a feladatra jól rá tud tanulni a neurális háló, és a kimenetén megjelenő vektorból pedig következtethetünk a szó helyes szegmentálására is. A felépítése a következő: a bemenetén az adott input szó vektorizált megfelelőjét kapja, a kimenetére pedig az input szó helyesen szegmentált változatát tesszük, majd minimalizáljuk a háló hibáját. Tanítás után tehát a bemenetre bármely ismeretlen magyar szót adva megpróbálja megállapítani a szegmenshatárokat, és azokat jelölve ('+' jellel) adja a kimenetét.

A szegmentáló háló pontosságát növelhetjük, ha csak egy helyen szegmentálandó szavakkal kísérletezünk. A szegmentálandó szót a háló bemenetére adva a haló kimenetét nem vizsgáljuk, hanem csak összehasonlítjuk (Euclidesi távolságát vesszük) a lehetséges szegmentált változatoktól, pl *almafajta* bemenetre összehasonlítjuk a kimenetet ezekkel:

```
a+lmafajta
al+mafajta
alm+afajta
alma+fajta
almaf+ajta
almafa+jta
almafaj+ta
almafajt+a
```

A legkisebb távolságú szegmentálást választjuk, mivel értelemszerűen az hasonlít legjobban a valódi, helyes felbontásra.

4 Adatok és Előfeldolgozás

A bemeneteket az MNSZ2 [9]- ből generáltuk. A következő szűrések alkalmazásával:

- Minden szót kiszűrtünk, mely nem tartalmazott magyar karaktert
- Csak uniq szavakon dolgoztunk, egy szó egyszer szerepelt
- A kísérletek szempontjából érdekes, hogy a tanult szavak gyakoribbak-e vagy nem, mivel az egyes adatok szórása különbözik, ezért gyakori és véletlenszerűen kiválasztott szólistákkal is kísérleteztünk
- Mivel az adat egy internetes crawler kimenete, a szavak közül eltávolítottuk a UMBC WebBase [10] alapján a leggyakoribb 10000 angol szót
- A szavakat kisbetűssé tettük, hogy kiszűrjük a mondat eleji nagybetűk zavaró hatását
- Készítettünk olyan bemenetet is, melyben a digraph-okat kicseréltük egyetlen karakterre (a digraph kezdőbetűje capitalként, pl cs->C), mivel ezek jelentésileg az esetek döntő többségében így reprezentálhatóak
- Az előbbi bemenet esetében keletkezett kisszámú fals pozitív eredmény is, ezek hatása viszont nem számottevő az autoencoder működésében
- A szavak hosszát korlátozzuk, mivel egy fix méretű neurális háló bemenetét adják és ennek korlátos a dimenziója, így a legnagyobb szóhosszt 20-nak választottuk, így lefedjük a bemenet 96%-át, és az ennél hosszabb szavak meghatározó többsége összetett szó, melyek összetevői szerepelnek a listában

A szűrést követően a frekvencialista 6209349 elemet tartalmazott, ami túl sok lett volna a kísérletek futásideje miatt, ezért a tanításhoz a következő adathalmazokat képeztük:

- 200000 leggyakoribb szó
- 200000 véletlenszerűen kiválasztott szó

- 200000 leggyakoribb szó, melyben a digraphok cserélve lettek
- 200000 véletlenszerűen kiválasztott szó, digraph-ok cserélve lettek

További Split-Brain Autoencoder és Segmentálásos kísérletekhez az alábbi listák készültek:

- 200000 leggyakoribb szó szétválasztva helyindexek szerint, egyik oszlopban páros helyek, másik oszlopban páratlan helyek szerepelnek, '_'-vel paddelve a helyek amiket az adott oszlop nem ismer
- 200000 leggyakoribb szó szétválasztva magánhangzó-mássalhangzó szerint, egyik oszlopban a szavak magánhangzói, másik oszlopban a magánhangzók szerepelnek, '_'-vel paddelve
- 400000 tetszőlegesen kiválasztott szó, melyekhez társul a szegmentált változatuk is, ezeket a segmentation kísérlethez használtuk

A fenti corpusok beolvasására több osztály is létrejött, de van egy közös ősük, melyben az interface is implementálva van, így ugyanazokkal a függvényekkel kezelhetőek.

4.1 Feature kinyerés

A corpus kezelő osztályok ősében van implementálva a szavak bemenetté, és a kimenet szavakká alakításáért felelős függvények.

A bemeneti szavak minden karaktere egy V dimenziós vektorba van leképezve, ahol V a corpus abc-jének a hosszával egyenlő, tehát egy V dimenziós one-hot encoding modellt használunk karakterszinten. Ezeket a vektorokat sorban egymás után fűzve kapjuk meg a szót reprezentáló feature vektort. A szavak hosszát korlátoztuk 20-ra, tehát a rövidebb szavak elejét space karakterrel paddeljük, hogy mindegyik vektor ugyanakkora dimenziójú legyen. A teljes bemeneti dimenzió így 20V. Az alábbi ábrán látható, hogy pozíciónként hogyan változik a karakterek entrópiája az összes 10 hosszú magyar szót figyelembe véve. Mivel a szavak végén láthatóan kisebb az entrópia, itt könnyebb logikus szabályokat találni, így könnyebben tömöríthető is lehet.

A program akármilyen bemenetre adaptívan abc-t készít, így nem jelent számára problémát az sem, ha teljesen más nyelvű, vagy akár karakterkészletű nyelven kéne tanulnia, mivel ehhez automatikusan alkalmazkodni képes.

5 Kísérleti Beállítások

A neurális hálók bemeneti és kimenete tehát 20V hosszú vektorok, a tanuló algoritmus ebből a loss-t Euclideszi távolság alapján számolja, és propagálja vissza a súlyokra. A kiementi vektor nincs korlátozva 0 és 1 értékekre, bármilyen valós számot felvehetnek. Így a kimeneti vektorból a szót úgy képezzük, hogy az adott karakterhez tartozó vektort vizsgáljuk, és abból kiválasztjuk a legnagyobb értéket, és az ehhez tartozó vektort hozzáfűzzük a kimeneti szóhoz. Így a kimeneti szó a bemenetihez hasonlóan 20 hosszú, szóközökkel tűzdelt lesz.

5.1 Alapvető kísérleti beállítások, környezeti változók

optimalizáló függvény	tf.train.AdamOptimizer()
nemlinearitás	tf.nn.sigmoid() (sokkal jobban teljesített mint a relu, vagy tanh)
rétegek száma	3…9
enkóder neuronok száma	10-1000
tanítási türelem (early stopping)	30 lépésenként mintavételezi a loss-t a validation data-n, ezt vizsgálja 20 egymást követő pontban, ha növekedik az értéke, early stopping módba vált és kiértékeli a tanítást
train-validation-test felbontás	80%-10%-10%, a teszt adatot egyáltalán nem látja a tanulás során, csak azon értékeljük ki a teljesítményét
kísérletek futásszáma	2, hogy eliminájuk az inicializálásból eredő szórást

5.2 Evolúciós algoritmus

A hiperparaméterek optimalizálásához létrehoztunk egy evolúciós algoritmust, melynek populációja neurális hálókból áll, az egyének paraméterei pedig az adott háló rétegszáma, és a rétegekben lévő neuronok száma egy listában tárolva.

Az algoritmus működése röviden a következő:

1. Létrehozunk egy populációt (40 fő), mindegyik egyén paramétereit random inicializálva, így sok különböző rétegszámú és neuronszámú háló keletkezik
2. Mindegyik neurális hálót létrehozzuk és tanítjuk, majd eltároljuk a total loss-t és a character accuracy-t
    a. Kiválasztjuk a legjobban teljesítő 30%-ot és megtartjuk
    b. A populáció 10%-át mutációnak vetjük alá, ekkor változhat az egyes rétegekben lévő neuronok száma, és a rétegek száma is
    c. Végigfutunk a maradékon és abból 10% valószínűséggel választunk véletlenszerűen túlélőket,
    d. Amíg el nem érjük az eredeti populáció lélekszámát, véletlenszerűen házasítunk tetszőleges egyéneket, ekkor a gyerekük az apától a háló bal felét, az anyától a háló jobb felét örökli, így nagyon aszimmetrikus mutánsok is létrejöhetnek, melyeknek az életképessége kétes

A 2-es lépés egy generáció, a konvergencia nagyon gyors, általában 3 generáció elegendő az optimálishoz közeli beállítást megtalálni. Ezzel nagyon lecsökkentettük a szükséges kísérletek számát, mivel nem kell bejárni a nagyon költséges sok dimenziós, sok rétegű területeket, mivel azok teljesítménye elmarad a kisebb rétegszámú hálókétól.

5.3 Kimerítő bejárás

Ezen módszer lényege az volt hogy egységes képet kapjunk a hiperparaméterek teréről, és azt könnyen kiértékelhetővé tegyük. Itt értelemszerűen csak az egy rétegű hálóval próbálkoztunk, mivel több rétegnél nagyon gyorsan megugrik a kísérletek száma. Az enkóder réteg neuronszámát változtattuk 20-tól egészen 500-ig 20-as lépésekben.

6 Eredmények kiértékelése

A kiértékelés alapjául a test adaton végzett karakterszintű pontosság, szószintű pontosság és átlagos Levenshtein távolság szolgál, melyet a rekonstruált szó és a bemeneti szó között nézünk.

6.1 Autoencoder, Variational Autoencoder

Miután lefutottak a kísérletek arra az eredményre jutottunk, hogy az egy rejtett rétegű sima autoenkóder teljesít legjobban, mégpedig akkor ha a rejtett réteg a legnagyobb. Több réteg esetén a többszörös nemlinearitás információveszteséggel is járhat, így a rekonstrukciót nehezíti. Ez egyáltalán nem meglepő, minél több neuronunk van, annál egyszerűbb a feladat, például a bemenettel egyező neuronszámú hidden layer esetén elegendő lenne megtanulnia az identitás függvényt a hálónak. 500-as rejtett réteggel már szinte tökéletes volt a visszaállítási pontosság.

6.2 Split-Brain Autoencoder

6.3 Segmentation

7 Konklúzió

Tömörítésre aligha használható az architektúra, inkább a sémafelismerés, és

Irodalomjegyzék

[1] Y.Bengio,A.Courville,andP.Vincent,“Representationlearning: Areview and new perspectives,” IEEE Trans. PAMI, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” science, vol. 313, no. 5786, pp. 504–507, 2006.

[3] W. Rong, Y. Nie, Y. Ouyang, B. Peng, and Z. Xiong, “Auto-encoder based bagging architecture for sentiment analysis,” Journal of Visual Languages & Computing, vol. 25, no. 6, pp. 840–849, 2014.

[4] S. Lu, Z. Chen, B. Xu, et al., “Learning new semi-supervised deep autoencoder features for statistical machine translation.,” in ACL (1), pp. 122– 132, 2014.

[5] J. Li, M.-T. Luong, and D. Jurafsky, “A hierarchical neural autoencoder for paragraphs and documents,” arXiv preprint arXiv:1506.01057, 2015.

[6] P. Mirowski, M. Ranzato, and Y. LeCun, “Dynamic auto-encoders for semantic indexing,” in Proceedings of the NIPS 2010 Workshop on Deep Learning, pp. 1–9, 2010.

[7] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning requires rethinking generalization.” 2017.

[8] P. Baldi, “Autoencoders, unsupervised learning, and deep architectures.,” ICML unsupervised and transfer learning, vol. 27, no. 37-50, p. 1, 2012.

[9] Cs. Oravecz, T. Váradi, and B. Sass, “The Hungarian Gigaword Corpus,” in Proceedings of LREC 2014, 2014.

[10] L. Han, A. L. Kashyap, T. Finin, J. Mayfield, and J. Weese, “Umbc_ebiquity-core: Semantic textual similarity systems,” in Second Joint Conference on Lexical and Computational Semantics (*SEM), (Atlanta, Georgia, USA), pp. 44–52, Association for Computational Linguistics, 2013.

Függelék