

TDK-dolgozat

Szabó Gergő

2022

Szegedi Tudományegyetem

Természettudományi és Informatikai Kar

Informatikai Intézet

Számítógépes Algoritmusok és Mesterséges Intelligencia tanszék

**Transzformer alapú mély neuronhálós magyar nyelvi
elemző modell**

Készítette:

Szabó Gergő

III. évf.

programtervező informatikus BSc

Témavezetők:

Dr. Farkas Richárd

egyetemi docens

Szántó Zsolt

tudományos segédmunkatárs

Szeged

2022

Tartalomjegyzék

Bevezetés	5
1. Természetes nyelvfeldolgozás	7
1.1. A természetes nyelvfeldolgozás kezdeti motivációja	7
1.2. spaCy	8
1.3. Tokenizáló	9
1.4. Mondatra bontó	10
1.5. Szótövezés	10
1.6. Szófaji címkéző	10
1.7. Morfológia	11
1.8. Függőségi elemző	11
1.9. Névelem-felismerő	12
2. Adatbázisok	14
2.1. Szeged Korpusz	14
2.2. Universal Dependencies Hungarian Szeged	14
2.3. NYTK-NerKor	15
2.4. SzegedNER	15
3. Gépi tanulás	17
3.1. Mély tanulás	18
4. Nyelvi modellek	21
4.1. Szóbeágyazások	21
4.2. Transzformer	22
5. Kapcsolódó munkák	25
6. Rendszer felépítése	26
6.1. Többfeladatos tanulás	26
6.2. Transzfer tanulás	27
6.3. Gráf alapú függőségi elemző	28
6.4. Beam search alkalmazása névelem-felismerésre	29
7. Eredmények	30
7.1. Eredmények, összehasonlítások	30
7.1.1. Tokenizálás, mondatra bontás	31
7.1.2. Morfó-szintaktikai elemzés	31

7.1.3.	Gráf alapú függőségi elemző eredményei	32
7.1.4.	Névelem-felismerés	33
7.2.	Memóriahasználat	34
7.3.	Jövőbeli tervek	36
8.	Összegzés	37
	Köszönetnyilvánítás	38
	Irodalomjegyzék	39

Bevezetés

Napjainkban a mesterséges intelligencia behatárolja az életünket. Legyen szó akár egy könnyebb vagy éppen egy nehezebb feladatról. Ma már képesek vagyunk olyan rendszereket építeni, amelyek rendkívüli módon segíti az emberi életet. Csak vegyük például a háztartásbeli okos gépeket avagy az önvezető autót. Az internet korát éljük, aminek az egyik legjövödelmezőbb része az adat. Ahhoz hogy ezeket képesek legyünk feldolgozni, szükség lehet olyan alkalmazásokra, amik képesek információt kinyerni a szövegekből. Erre a megoldás a természetes nyelvfeldolgozás (*Natural Language Processing, NLP*).

A természetes nyelvfeldolgozásban, mint a gépi tanulás más területein, az utóbbi években a nagy erőforrás-igényű, cserébe nagy pontosságú mély neurális hálózatokra épülő megoldások, azon belül is a transzformer alapú architektúrák nyertek teret. Szövegek elemzésére magyar nyelven is számos gépi tanuló megoldás létezik már, de kutatásomban olyan módszereket javaslok, amelyeket még nem alkalmaztak. Az általam fejlesztett eszköz az első magyar nyelvre átfogó nyelvi előelemzést adó transzformer alapú keretrendszer. Tartalmaz tokenizálót, mondatra bontót, lemmatizálót, szófaji címkézőt, morfológiai elemzőt, függőségi elemzőt és névelem-felismerést.

Kutatásom további célja a gépi tanulás és a mély neuronhálók működésének vizsgálata, hogy kiderüljön, mekkora pontosság javulás lehetséges nagyobb hálózatokkal és ezeknek mekkora erőforrásigénye van. Megéri-e a több erőforrást igénylő modell vagy megelégedhetünk a hagyományos rendszerek teljesítményével? Mennyit tud javítani a pontosságon a transzformer alapú rendszer a korábbi megoldásokhoz képest? A kutatásomban egy transzformer alapú mély neuronhálót fejlesztettem a spaCy keretrendszer segítségével, amelyet összehasonlítok hagyományos gépi tanuló és korábbi mély rekurrens neuronhálós megoldásokkal nagy hangsúlyt fektetve az erőforrásigények és a pontosság megértésére.

Vizsgálataim során azt találtam, hogy a transzformer alapú rendszer az általam vizsgált legjobb rendszerhez képest szinte mindegyik területen pontosabbnak bizonyult a kutatásban használt kiértékelő adatbázison (Universal Dependencies). A függőségi elemzés során az UAS 12 százalékpontot javult és a LAS pedig 13 százalékpontot. Így a függőségi elemző esetében sikerült state-of-the-art eredményeket elérni. A többi elemző esetén is megfigyelhető legalább 1-2 százalékpontnyi javulás. A pontosság növekedése mellett azonban a várakozásoknak megfelelően a CPU-n a memóriaigény másfélszer több lett és az időigény pedig megközelítőleg tízszeresére nőtt. Ezek alapján jövőbeli kutatási cél lehet az erőforrásigények optimalizálása, valamint további tesztek segítségével a pontosság növelése.

1. fejezet

Természetes nyelvfeldolgozás

1.1. A természetes nyelvfeldolgozás kezdeti motivációja

A természetes nyelvfeldolgozás, angol nevén az NLP (*"Natural Language Processing"*) gyökerei 1950-ig vezethetők vissza a mesterséges intelligencia és a nyelvészet keresztezéseként ([Nadkarni és mtsai, 2011](#)). Ezek együttműködése terített meg a mai modern nyelvfeldolgozásnak. Napjainkban például nyelvfeldolgozást használunk többek között a gépi fordításra, tehát a szöveg fordításra. Mára annyira kinőtte magát ez a terület, hogy szinte bárki nyugodtan használatba veheti, előfordulhat minimális hiba, de összességében helyes fordítást kapunk. Egyik ilyen alkalmazás például a Google fordító. Emellett, ha szövegekkel dolgozunk, akkor névelem-felismerő segítségével megtudhatjuk, hogy milyen tulajdonneveket említ a például egy hírportál, anélkül, hogy elolvasnánk az egész blogbejegyzést. Kereshetünk olyan szövegeket is, amelyek hasonlítanak egymásra vagy akár vizsgálhatjuk a szavak különböző jelentéseit más-más szövegkörnyezetben. Ezek mind olyan esetek, amelyek szükségessé teszik a nyelvfeldolgozást.

Az ilyen fajta rendszerekhez elengedhetetlen az előnyelvfeldolgozási lépés. Egy chatbot végalkalmazáshoz alapvetően szükség van olyan adatra, ami már fel van dolgozva. Ezt az előfeldolgozási lépést végzi egy olyan rendszer, ami képes elemezni a szöveget, itt jön képbe a természetes nyelvfeldolgozás. Az előfeldolgozási lépés után könnyedén "föle" helyezhető egy alkalmazás, ami ki tudja használni ezeket az információkat.

Ebben a fejezetben részletesen ki fogok térni minden olyan részére a nyelvfeldolgozásnak, ami érintett ebben a kutatásban.

1.2. spaCy

A természetes nyelvfeldolgozásra ma már rengeteg architektúra készült, amik segítik a szövegek elemzését. Olyan rendszert akartam választani, ami mindenekelőtt képes megoldani a legfontosabb szöveg-előfeldolgozási feladatokat, beleértve a tokenizálást, a mondatra bontást, a szófaji címkézést, a szótövesítést, a függőségi elemzést, névelemfelismerést és a szóbeágyazás ábrázolását is. Továbbá, ami "iparilag erős" szövegfeldolgozó rendszernek számít. Ezek mellett fontos volt az, hogy nemzetközileg elismert annotálási szabályoknak megfeleljen. Ezen túlmenően, egy iparág-központú rendszernek fejlesztőbarátnak, személyre szabhatónak és könnyen integrálhatónak kell lennie, mivel az NLP-modulok a gyakorlati alkalmazásokban egy nagyobb rendszer szerves részét képezik. Ezek a követelmények azt jelentik, hogy szilárd dokumentációnak is rendelkezésre kell állnia.

Ezek feltételek mérlegelése után választottam a spaCy-t ([Honribal, 2015](#)), ami megfelel ezeknek a tényezőknek. A spaCy egy nyílt forráskódú architektúra Pythonban, ami nyelvfeldolgozásra lett fejlesztve. Ugyan jelenleg is számos nyelven elérhető, ezek száma folyamatosan növekszik. Az elmúlt hónapok során például magyar nyelvre, HuSpaCy néven is publikálásra került ([Orosz és mtsai, 2022](#)) egy új, pontosabb változat.

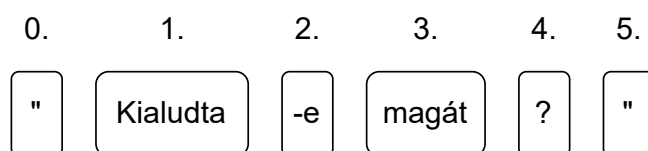
Kutatásaim célja, hogy ipari szintű magyar előelemzőláncot hozzak létre, ezért a kísérletek és a modellek tesztelése ebben a keretrendszerben zajlottak. Fontos kiemelni, hogy a cél egy egységes modell létrehozása, ami tartalmaz minden elemzőt. Ahhoz, hogy teljeskörűen megértsük a folyamatokat, vizsgáljuk meg először az őket alkotó komponensek sorrendjét és működését.

1.3. Tokenizáló

Első lépésként egy tokenizáló végig iterál az egész szövegen. Ennek során a tokenizáló egyszerűen csak veszi az egész dokumentum halmazát és szavakra bontja, szegmentálja. Ez gyakorlatilag egy előfeldolgozás a többi komponens számára.

A tokenizálás első lépése, és egyben legegyszerűbb szabálya az, hogy szóköz mentén vágjuk a szavakat tokenekre. Ez azonban még nem elegendő számunkra, ezért a folyamat újra megismétlődik, ám ezúttal már más szabályok figyelembe vételével is. A második iterálás során, amikor kötőjelekkel vannak elválasztva a szavak vagy vessző van a mondatban, esetleg idézőjelek, mondatvégi pontok vagy egyéb írásjelek, akkor ezeket mind-mind külön kell vennie a tokenizálónak. Az iterálást balról jobbra hajtja végre a rendszer, és ahol a szabály érvényesül, ott végrehajtja a vágást, majd addig folytatja, míg a végére nem ér a szövegnek.

Példaként nézzük meg a következő mondatot: *"Kialudta-e magát?"*. Elvárt esetben ezt 6 részre kellene tagolni, ahogy az 1.1. ábrán is látható, tehát az írásjeleket is külön kellene szegmentálnia. Ellenben az egyszerű szóköz alapú szabály csak 2 részre tokenizálná ezt a mondatot. Itt lép közbe a spaCy tokenizálója, ami megoldja ezeket a problémákat.



1.1. ábra. Példa a helyes tokenizálásra.

Figyelnie kell arra is, hogy mi számít rövidítésnek, hiszen ezeket nem szabad szétbontania. Ha tehát a mondat végét jelölő pontról van szó, akkor ott mindenképpen külön kell szegmentálnia az azt megelőző szótól. Ezzel szemben a rövidítéseknel, például az *"U.S."*-nél nem áll fenn ez a szabály, ezt egy tokennek kellene vennie.

Egy manapság új probléma, az "emojik" használata kissé megnehezítette az eddigi tokenizálást. A hangulatjelek használatakor valójában írásjelek kombinációi képeznek egy emotikont. Emiatt ezeket semmiképpen sem szabad tokenizálni, mert elvesz az írásjelek együttes jelentése. A spaCy-be azonban már beintegrálták ezt a funkciót.

1.4. Mondatra bontó

A mondatra bontó ("*Sentencizer*") nem sokban tér a tokenizáló elvétől, azonban itt nem szavakra bontunk, hanem mondatokra. Létezhet olyan módszer, hogy a mondatszegmentálást a függőségi elemző végezné, azonban emellett elérhető egy kisebb hatékonyságú, egyszerűbb szabály alapú rendszer is, ahol nincs szükség gépi tanított modellre a használathoz. Viszont a kutatásomban mégis egy gépi tanulás alapú modellt használtam, mely döntést későbbiekben indoklok.

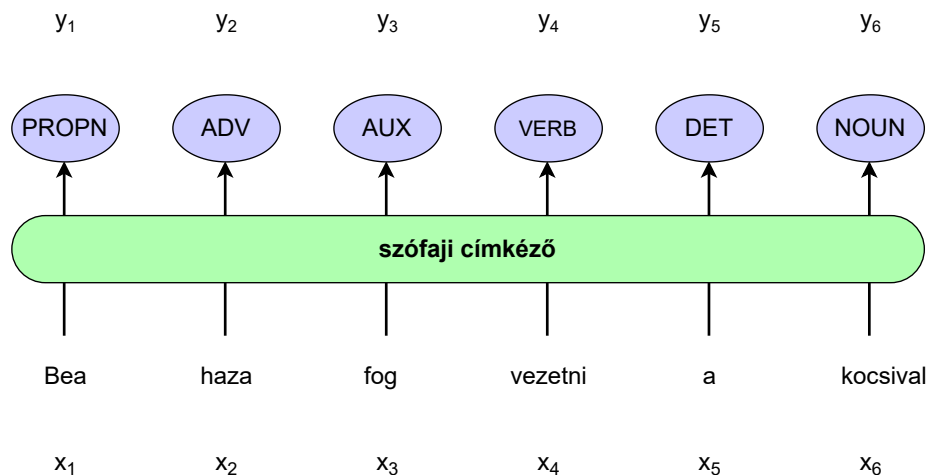
1.5. Szótövezés

A szótő ("*lemma*") tulajdonképpen a szónak azon legkisebb rész szava, ami gyakorlatilag a szótári alakja. A szótő meghatározására az egyik legkönnyebb módszer az a *stemming*, amikor csak szimplán "lecsapjuk" a toldalékot a szó végéről, például "*tavak*" → "*tav*". A példából is látszik, hogy ez a módszer nem túl intuitív. Erre megoldás lehet a lemmatizálás, ami arra szolgál, hogy a ragozatlan szótári, kanonikus formára hozza a szavakat. A lemmatizálás során is már egy gépi tanulás modellt használok, ami az *Edit-Tree-Lemmatizer* nevet kapta. Alapvetően a mondatra bontó és a szótövező esetén is egy egyszerű szabály alapú rendszerrel is el lehetne érni a kívánt eredményt, viszont én ebben az esetben mind a kettőnél a gépi tanulás mellett döntöttem, amelyre a későbbiekben kitérek.

1.6. Szófaji címkéző

A szófaji címkéző gyakorlatilag minden egyes szóra, pontosabban minden tokenre jósolni fog egy szófajt, ahogyan az 1.2. ábrán látható. Itt is érdemes lehet megemlíteni a gépi tanulást. A szófaji címkéző esetén a szabály alapú rendszereket korán leváltották a gépi tanulás alapú rendszerek, amik így lehetővé teszik, hogy eddig nem látott tokenre is tudjon olyan címkét jósolni, ami a megfelelőbb. Ez a 3. fejezetben kerül kifejtésre.

A szófaji címkézés az a folyamat, amely során az input összes szavához hozzárendelünk egy szófajt. Tehát az input a tokenizált szavak sorozata $x_1, x_2, \dots, x_n$ és a kimenet pedig $y_1, y_2, \dots, y_n$ címkék sorozata. Minden y_i pontosan megfelel minden x_i bemenetének



1.2. ábra. Szófaji címkéző működése.

(Jurafsky és Martin, 2020), ahogy az 1.2. ábrán bemutattam. Gyakorlatban ezt például egy chatbot-hoz szokás használni, mert egy adott kérdéshez ismerni kell a szófajokat egy mondatban.

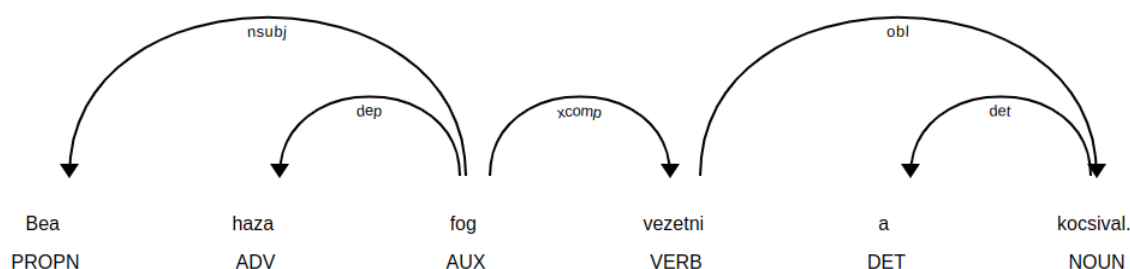
1.7. Morfológia

A morfológia a nyelvészet azon ága, amely a szavak belső szerkezetét vizsgálja. A magyar nyelv alapvetően egy agglutináló nyelv (Booij, 2012), ami azt jelenti, hogy a szavaknak a szótövéhez végződéses tartozhatnak, illetve akár előtag is társulhat egy adott szóhoz. A magyarban ezek a toldalékok fejezik ki a mondat szavainak egymáshoz való viszonyát, ellentétben például az angolnal, ahol ezeket főként különálló, funkciószavak írják le. Az angolban így a magyarhoz képest nagyon kevés toldalalék létezik és azokat le lehet kezelni pár egyszerű szabállyal. Éppen ezért lesz nehezebb a magyar nyelvre egy morfológia elemzést végrehajtani, mert egy egyszerű előtag vagy toldalalék képes megváltoztatni a szó szerkezetét, jelentését.

1.8. Függőségi elemző

A függőségi elemzés ("*Dependency Parsing*") egy mondat nyelvtani szerkezetének elemzése a mondat szavaira nézve. A függőségi elemzésben az élek két szó kapcsolatát írják le a mondaton belül. Az élek típusát pedig a hozzárendelt függőségi címkék hatá-

rozzák meg. Például, ahogy az 1.3. ábrán látható, a *Bea* $\rightarrow$ *fog* között létezik egy *nsubj*¹ címkéjű függőségi kapcsolat.



1.3. ábra. Egy függőségi elemzés vizualizálva.

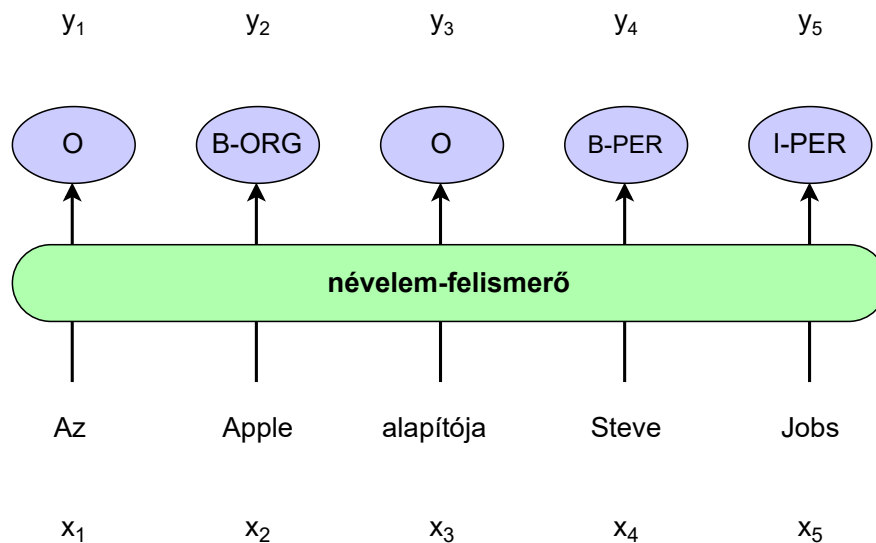
Láthatjuk tehát, hogy több függőségi címke létezik, de fontos kiemelni, hogy mindig pontosan csak két szó között állhat fenn ilyen kapcsolat. Az egyik minden esetben a szülő és a másik pedig a gyerek lesz.

1.9. Névelem-felismerő

A névelem-felismerő ("*Named Entity Recognition*") feladata egyrészt, hogy kinyerje a névelemeket a szövegből. Másrészt pedig, hogy a kinyert adatot besorolja egy címke halmazba. Tulajdonképpen tulajdonneveket keresünk a szövegekben, ezzel kiemelve a névelemeket a mondatokban. Az entitások 4 főcsoportba tartozhatnak: személynév, intézménynév és földrajzi név, illetve van egy nagy összefoglaló halmaz is, az egyéb. Sokféle annotálási elv létezik, ami foglalkozik több entitás típussal is (Weischedel és mtsai, 2013; Ding és mtsai, 2021; Novák és Novák, 2022), de a kutatásomban csak ezt a 4 fő entitást fogom használni.

A feladat folyamata kicsit hasonló a szófaji címkézőhöz, amit az 1.6 fejezetben bemutatam. Itt is címkéket rendelünk tokenekhez. Ahogyan a legtöbb esetben, a címkék itt is függhetnek egymástól. Így tehát, ahogy a példán is látszik, ha egy hagyományos név szerepel a szövegben, akkor képes megtalálni az elejét "B-PER" és a végét "I-PER" a névnek. Ha hosszabb tulajdonnév lenne, akkor az első kivételével mindegyiket "I-PER"-nak fogja jelölni. A nem entitások egy nagyon egyszerű "O" címkét kapnak. Alapvetően több jelölési szabvány létezik, de többnyire könnyen konvertálhatóak egymásra. A spaCy épp

¹<https://universaldependencies.org/u/feat/index.html>



1.4. ábra. A névelem-felismerő működése.

ezért alakítja át magának ezeket a jelöléseket egyaránt a tanuláshoz és a predikáláshoz is. Ahogyan az 1.4. ábrán látható is, a példa mondat IOB szabványt követ, viszont a rendszer belsejében más formátumú jelölést használ a spaCy. Gyakorlatilag csak egyszerűen átkonvertálja magának BILOU formátumúra (ha egyébként nem BILOU formátumú volt az adat), mert ezzel egyéb más rendszernek javíthatja a pontosságát. A BILOU elnevezés a felhasznált betűket jelenti.

2. fejezet

Adatbázisok

Négyféle kézzel annotált adatbázis lett felhasználva a kutatásom során. Ennek az oka az volt, hogy nem létezik jelenleg magyar nyelvre egy egységes korpusz, amiben minden információ létezik a tanulásokhoz. Ennek köszönhetően a korpuszok között van egy kis inkonzisztencia. Viszont előnye, hogy így nagyobb lesz az általánosító képessége a jövőben a modellnek. Ezeket mérlegelve lettek megválasztva a tanulási fázisok. Ezek majd az 6.2. fejezetben kerülnek kifejtésre.

2.1. Szeged Korpusz

A Szeged Korpusz egy kézzel annotált adatbázis, amely jelenleg közel 1,4 millió tokenből, 145 ezer különböző szóból és további 225 ezer írásjelből áll ([Csendes és mtsai, 2004](#)). A korpuszban megtalálható szépirodalmi szöveg, 14-16 éves tanulók fogalmazásai, újságcikkek, számítástechnikai szövegek, jogi szövegek, illetve gazdasági és pénzügyi rövid hírek. Továbbá a korpusz tartalmaz tokenizálást, mondatra bontást, szótövesítést, szófaji címkézést és morfológiai elemzést.

2.2. Universal Dependencies Hungarian Szeged

Az Universal Dependencies Hungarian Szeged ([Nivre és mtsai, 2017](#)) korpusz gyakorlatilag a Szeged Korpusz része, de jóval kisebb méretű, mint az eredeti Szeged Korpusz. Pontosabban a Szeged Korpuszból át lett konvertálva egy kisebb rész az Universal Depen-

dencies Hungarian Szeged korpuszba, amit aztán kézzel ellenőrizve lett és javítva, ahol szükséges volt (Vincze és mtsai, 2017). Hátránya ennek a korpusznak az, hogy az adatok száma nagyon megcsappan, számszerűsítve a Szeged Korpusznak mindössze csak 3,3%-át lehet így felhasználni, tanító halmaznak pedig csak közel 1%-át. Ennek a hibának a kiküszöbölése majd az 6.2. fejezetben kerül kifejtésre. Ezért mivel az összehasonlítás miatt és mert mindenki által az UD formátum nagyobb preferenciát nyert, ezért a kutatásban is az UD fájlformátum lett használva. Jelenleg összesen 37 univerzális függőségi címke van használatban (De Marneffe és mtsai, 2014).

2.3. NYTK-NerKor

A NYTK-NerKor¹ (Simon és Vadász, 2021) egy magyar nyelvre íródott gold standard² 1 millió tokent tartalmazó korpusz névelem-felismerésre. A NerKor kicsit eltér az eddigi kettő adatbázistól. Ebben a korpuszban a névelemek kerültek előtérbe, és bár elérhetőek a korpusz egy részében a morfológiai elemzések is, ahogyan az eddigi kettőben, de azok a kutatásom során nem kerültek felhasználásra. A NerKor adatbázis gyakorlatilag 6 nagyobb témából lett megírva: regény, jog, hír, weboldal, wikipédia és egyéb szövegekből. A tanító-validációs-tesztelő (80%-10%-10%) halmazokban egyenletes arányban lettek elosztva a témák.

2.4. SzegedNER

A SzegedNER (Szarvas és mtsai, 2006a) szintén névelem-felismerésre egy gold standard² korpusz magyar nyelvre. Különbsége a NerKor-tól, hogy kisebb, pontosabban 200 ezer tokenből áll és itt csak 2 téma van: bűnügyi és üzleti szövegek. Ellenben viszont sokkal tisztább és kevesebb zaj van a szövegekben, mint a NerKor-ban. Mondateloszlásban is sokkal kiemelkedőbb, de ez nem véletlen, mert a NerKor-ban például a regény szekcióban előfordulnak kiegyensúlyozatlan mondatok a párbeszéd miatt. Viszont ez nem éppen hátrány, mert a mai szövegekben sajnos sok a redundáns/hibás szöveg és ezekre,

¹Nyelvtudományi Kutatóközpont

²Szakértőkkel anonizált, hitelesített adatbázis

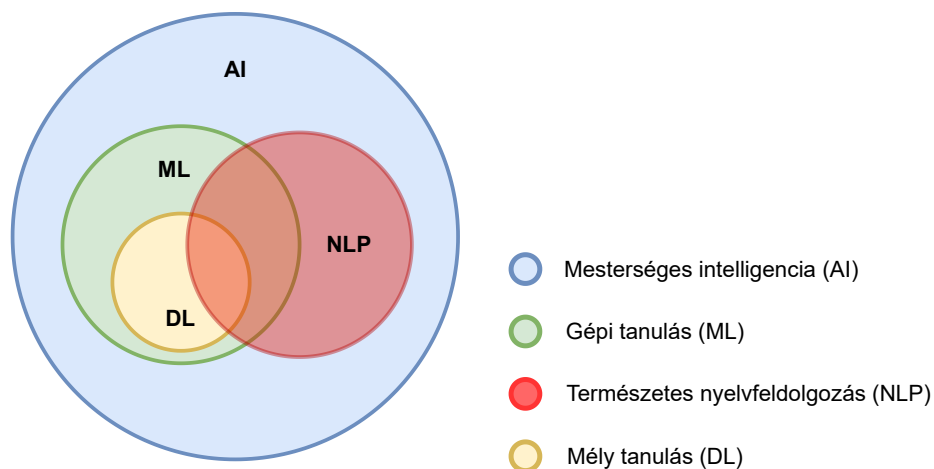
akarva-akaratlanul, de fel kell készülnünk. Erre megoldás az előző fejezetben bemutatott NerKor, illetve pontosan ezért lett a két korpusz közösen használva, hogy fel legyen készítve a modell ezekre a várható hibákra.

3. fejezet

Gépi tanulás

"Ha adott egy jól specifikált T feladat és P teljesítménymetrika, akkor gépi tanulásról beszélünk, ha a rendszer egyre több E tapasztalat/megfigyelés begyűjtése esetén egyre jobban tudja megoldani a T feladatot a P metrikában mérve." (Mitchell, 1997)

A gépi tanulás, angol nevén "machine learning" hazánkban is kezd egyre jobban elterjedni. Intelligens eszközök, olyan kommunikációt segítő "robotok", mint a telefonszolgálatok automatikus ügyintézője, de még egy számunkra egyszerű filmajánlórendszer mögött is ilyen rendszerek vannak. Ma az interneten bárki találkozhat személyre szabott hirdetésekkel, reklámokkal, már nagyon sok háztartásban találhatók robotporszívók is, amik ugyanúgy gépi tanulás alapúak, mint a folyamatosan fejlődő önvezető autó technológiák.



3.1. ábra. A gépi tanulás elhelyezkedése a mesterséges intelligencián belül.

A gépi tanulás korszakát éljük, amikor sok mai feladat átruházható lenne egy algoritmusra. Habár maga az ötlet korán sem mai, pontosabban az 1970-es években ([Kubat és Kubat, 2017](#)) lett korszakalkotó ez a terület. Ami nagyon korlátozta az akkori tényezőket, az az adat hiánya volt, illetve a számítási kapacitás. Hiába álltak rendelkezésre ezek a rendszerek akkor, nem tudták kamatoztatni, mert nem volt elég adatuk és erőforrásuk. Ma már ez kicsit sem akadály, az internet és az évről-évre bővülő hardverek miatt elérhetővé vált rengeteg információ és adat, ami csak arra vár, hogy elemezzük.

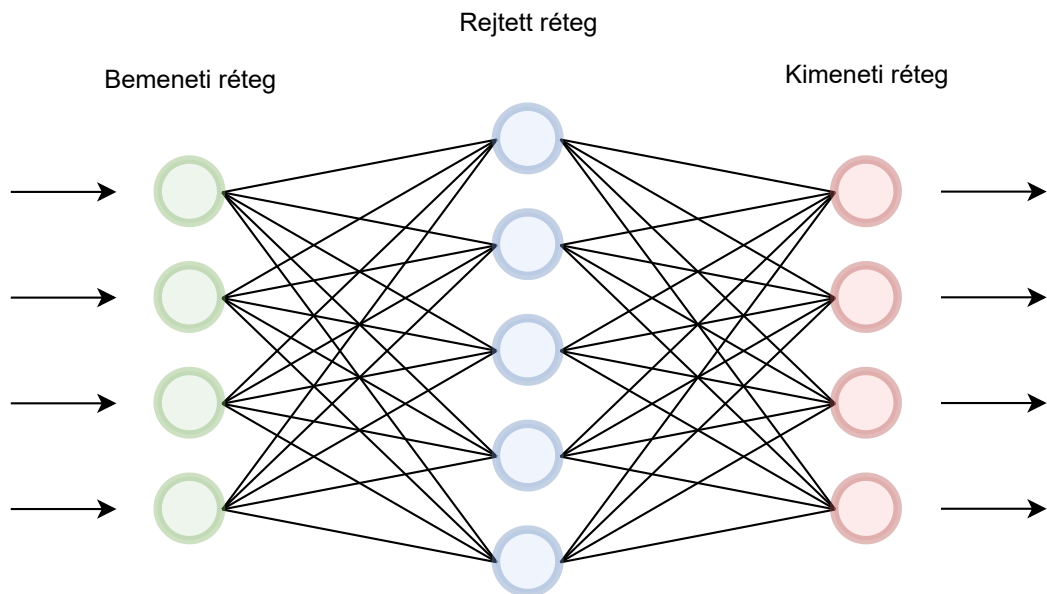
A gépi tanulás három csoportra bontható: felügyelt tanulás ([Learned-Miller, 2014](#)), felügyelet nélküli tanulás ([Ghahramani, 2003](#)) és megerősítéssel tanulás ([Sutton és Barto, 2018](#)). A következő bekezdésben a dolgozatomban használt felügyelt tanulást, azon belül is az osztályozást ([Ikonomakis és mtsai, 2005](#)) fogom kifejteni.

A felügyelt tanulás alapja az, hogy vannak úgynevezett címkék a példákban (ez az annotálás). Ezeket a címkéket akarja predikálni a modell a tanulás során, illetve ezekbe az osztályokba kell majd besorolni a modellnek eddig nem látott példákat is. Valószínűség alapján hoz meg egy döntést, ha szimplán csak két osztályunk van, akkor binárisan dönt a predikálásról. Viszont ha több osztályunk van, akkor mindegyik osztályra hoz egy valószínűségi változót és abba fogja besorolni az adott példát, amelynek a legtöbb az értéke.

3.1. Mély tanulás

A kutatásom szempontjából legfontosabb és manapság legkorszerűbb terület a mély tanulás ([O'shea és Hoydis, 2017](#)). A mély tanulás áll a legközelebb az emberi agy működéséhez, ugyanis ebben az esetben mesterséges neuronok segítségével építjük meg a modellünket. Az elmúlt években rendkívüli módon vált népszerűvé ez a terület, sok egyéb tudományterületen belül is. Beszédfelismerésben, játékokban, képfeldolgozásban, szövegfeldolgozásban és még számtalan területen terjedt el ([Maier és mtsai, 2019](#)). Alapjaiban a módszer hasonló az eddig taglaltakhoz, annyiban tér el, hogy a részletesebb reprezentációnak köszönhetően a tanulás ideje drasztikusan megnőhet. Ugyanakkor egy neurális hálózatban részt vesznek neuronok is, amelyek segítik a rendszert, hogy még pontosabb legyen.

A neuronok felelősek azért, hogy a megtanult függőségek, súlyok a lehető legjobban

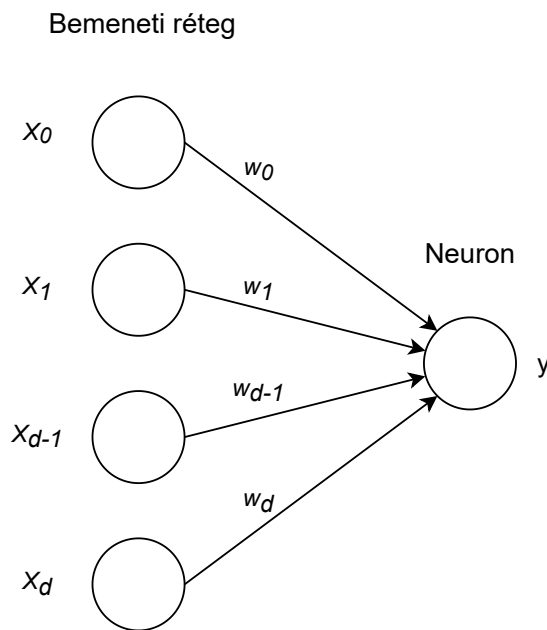


3.2. ábra. Egy neurális háló vizualizációja.

reprezentálják a valóságot. Kutatásom során minden tanítás mély tanuláson alapul. A mély tanulás onnan kapta a nevét, hogy a modell több rétegből, szintből áll, ahogyan az 3.2. ábrán látható. Gyakorlatilag megfeleltethető egy gráfnak a neurális hálózat, aminek sok szintje van. A mély tanulási módszerek abban a tekintetben jók, hogy több reprezentációs szinttel vagyis réteggel rendelkeznek. Ezek mindegyik szintje egy magasabb, absztraktabb reprezentációvá alakítják át az adatot. Több ilyen réteg összekapcsolása lehetővé teszi a mélyebb, összetettebb függvények megtanulását. A különböző rétegek célja tehát, hogy magára igazítsa az adatokat. Fontos kiemelni, hogy ezeket az összefüggéseket saját maga találja, alakítja, nincs emberi beleszólás ebben a folyamatban.

Egy neuron felépítése a következőképpen történik: a d az összes minta elemszáma, az i az aktuálisan vizsgált valahányadik elem, a w a súly, és az a pedig a jellemző értéke. Az y kimenete lesz az adott neuronnak, amit visz tovább akár több ilyen gócponton keresztül egészen a kimeneti rétegig.

$$y = f\left(\sum_{i=1}^d w_i a_i\right)$$



3.3. ábra. Egy neuron működése.

A mély tanulás leggyakoribb formája, lehet az akár mély neurális háló, akár nem mély, a felügyelt tanulás. Ehhez szükség van az adathalmaz szétvágására tanító, validációs, tesztelési halmazokra, bemeneti jellemzőkre és címkézett célváltozókra, majd valamilyen számításra, ami megadja a későbbiekben a predikciót. Ahogy az a 3.3. ábrán látható. Egy neuron működésekor megkapja a példát, amit legelőször nulla tapasztalattal megnéz és mond rá valamit. Ezután egy célfüggvény segítségével megméri, hogy mekkora volt a hiba mértéke, tehát megnézi az elvárt és a predikált értéket, és hibát számol köztük.

Ezután ezt a hibát visszaterjeszti a hálón. Ez a folyamat a hibavisszaterjesztés, ami abban segít, hogy a belső paramétereit úgy módosítsa, ahogy a feladathoz a legjobban illeszkedjen. Ezeket hívják súlyoknak, amik az 3.3. ábrán a " w " betűket jelentik. Egy tipikus mély tanulási rendszerben akár több, mint százmillió ilyen állítható súly lehet. Ezeket a súlyvektorokat addig állítja magában, ameddig a hiba mértéke konvergálni fog a 0-hoz. Ez persze nagyon nehéz feladat, de arra törekszik, hogy minél kevesebb legyen a hiba mértéke.

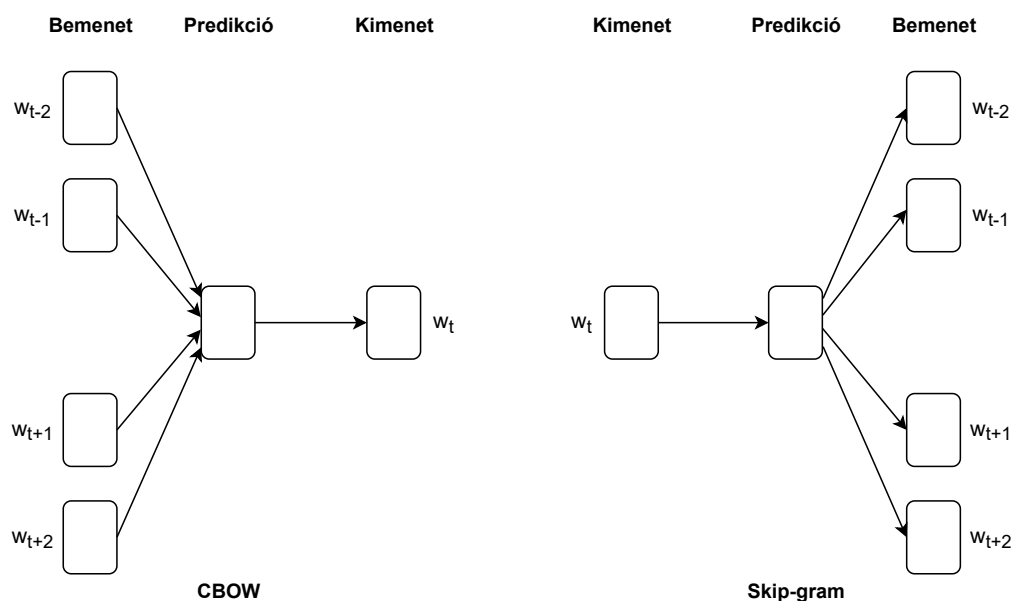
4. fejezet

Nyelvi modellek

4.1. Szóbeágyazások

A szóbeágyazás a vizsgált dokumentumszókincs egyik legnépszerűbb ábrázolása. Más szóval az összes különböző szót eltárolja valamilyen módon, ez lesz a szótár. Gyakorlatilag képes arra, hogy egy szót a szöveggörnyezete által egy vektortérbe helyezzen, figyelve a szemantikai és szintaktikai hasonlóságra más szavakkal. A szóbeágyazások a szavakat egy jellemzően pár száz dimenziós térbe képezik le úgy, hogy a jelentésben egymáshoz közel álló szavak közel kerüljenek a vektortében is egymáshoz.

Egy szóbeágyazás tanítása megelőzi ebben a környezetben számunkra tényleges célra szánt modellt. Egy mára már egyszerű Word2Vec-et nem annyira költséges tanítani. Ebben az esetben egy szimpla szóbeágyazás tanítása sem drága, mert nincs szükség annotált adatra. Egyszerűen elég egy nagy írott szöveg és konkrétan ez már egy annotált szöveg, így a szimpla mondat már egy tanuló példa lesz. Tanítása hasonló elveken alapul, mint amiről eddig szó volt: tanító példák, kiértékelés valamilyen metrika alapján, tesztelés. Kétféle tanulási módszer létezik, mindkettő neurális hálót használ. A CBOW modell gyakorlatilag kontextus alapján vagyis környező szavak elosztott reprezentációit kombinálja, hogy megjósolja az adott középső szót. A másik a Skip-gram modell, ami hasonló elveken alapul annyi különbséggel, hogy megfordítja a folyamatot. Vagyis a predikálandó szó elosztott reprezentációját használja a kontextus előrejelzéséhez. Ezen modellek működése a 4.1. ábrán látható.



4.1. ábra. A bal oldali modell a Continuous Bag of Words Model (CBOW), a jobb oldali pedig a Skip-gram.

4.2. Transzformer

A Bert (*Bidirectional Encoder Representations from Transformers*) gyakorlatilag egy "szóbeágyazás", ami segíti a modellek reprezentációs képességét (Devlin és mtsai, 2018). A BERT és minden hasonló kaliberű rendszer vektorokat generál az adott tokenekhez. Annyiban tér el az olyan hagyományos statikus rendszerektől, mint például a Word2Vec (Goldberg és Levy, 2014), hogy a BERT képes mondatreprezántációt létrehozni, tehát figyel a kontextusra is akár predikálás közben is. Ezzel szemben a Word2Vec minden szóhoz egy statikus szövektort rendel a tanítás során, ami a későbbiekben nem fog megváltozni. Amit a szóbeágyazáskor tanult egy tokenre, azt a vektort fogja mindig mondani az adott szóra. Itt jön képbe a BERT, ami figyelembe veszi a kontextust és képes arra, hogy akár mondatreprezántációkat hozzon létre. Ezzel segíti azt a tényezőt, hogy egy szónak, mondatnak lehet két értelme is. Ezt a két értelmet a kontextus és a szó összetételével képes megkülönböztetni.

Más szóval a BERT képes arra, hogy ne csak önmagában egy szót reprezentáljon vektorként, ahogyan a Word2Vec-nél van, hanem figyeljen az egész mondatra, kontextusra. Ugyanakkor képes arra is, hogy minden szóra emlékezzen és ezáltal minden további információra is, amit az adott szavak hordoztak. Ezzel szemben egy rekurrens rendszernél

(BiLSTM) elveszhetnek lényegi részletek, ahogyan tovább adódnak az információk a hálón belül. Ezzel elérhetővé vált, hogy a többértelműség ne legyen akkora probléma, mint a szóbeágyazásos rendszerek esetében.

A tanulás egyrészt úgy történik, hogy kitakar egy véletlen szót a mondatban tanulás közben (*Masked-Language Modeling, MLM*). Ezután a nem kimaszkolt többi szó segítségével megjósolja, hogy mi állhatott ott. Ez úgy történik, hogy a kimaszkolt szó kapcsolatban van az összes többi szóval és így a 12 rétegen keresztül folyamatos számítások segítségével és egy softmax aktivációs függvénnyel a végén kapunk egy szót, amit oda gondol. Tehát az eddig tanultak alapján mit mondana az adott két szó közé. Ellenőrzése egyszerű, hisz birtokunkban van az eredeti szöveg és csak össze kell hasonlítani, hogy eltalálta-e vagy nem. Bár nehéz dolga van és sokszor nem találja el a kívánt szót, de mégis képes arra, hogy segítse a fölé rakott végső modelleket abban, hogy eldöntse, hogy egy adott szó pontosan mit reprezentál az életben.

A másik módszer pedig a következő mondat predikciója (*Next Sentence Prediction, NSP*). Belső szerkezete nem sokban tér el az előzőtől. Viszont a célfeladat ebben az esetben, ahogy a neve is mondja, megmondani, hogy milyen mondat következhet az adott mondat után. Ebben az esetben a mondatba bekerül két token: a "[CLS]" és a "[SEP]". Az előbbi az első mondat elejét jelenti és az utóbbi pedig azt, ahol el van szeparálva a mondat, tehát ez a mondat elválasztó token, illetve a legvégére is kerül egy "[SEP]" token. Minden tokenhez ebben az esetben hozzáadódik egy mondatbeágyazás is, amely jelzi, hogy "A" vagy "B" mondatról van szó. Ugyanakkor minden tokenhez hozzáadódik az is, hogy melyik pozícióban állt a mondatban, megőrizve a sorrendiséget. Ezek után az eljárás hasonló, mint az MLM-nél, ezen értékek mentén próbálja megtanulni, hogy "A" mondat után milyen "B" mondat következhet.

Amiről az imént szó volt, ezek a kiértékelések a tanulás közben folytatódnak, ez a hibavisszaterjesztési folyamat. A BERT modelleket tényleges feladatokon szokták kiértékelni. Például MNLI¹ (mondatpárok jóslása), SST-2² (vélemény detekció) vagy SQuAD³ (kérdés megválaszolás) feladatokon.

¹Multi-Genre Natural Language Inference

²Sentiment Analysis

³Stanford Question Answering Dataset

Jelen kutatás azért jöhetett létre, mert már létezett magyar előtanított nyelvi modell, a huBERT (Nemeskey, 2021). Ezen nyelvi modellek tanítása rettentő módon költségesek, viszont amint ez sikerül, utána finomhangolhatóak szinte bármilyen NLP-s feladatra. A huBERT egy 12 rétegből (blokkból), egyenként 768 dimenzióból álló mély neurális háló, ahol úgynevezett "önfigyelés" van (*Self-Attention*), ami arra szolgál minden rétegen keresztül, hogy minél pontosabb, absztraktabb reprezentációja legyen a bemenetnek és így képes legyen feldolgozni a kontextuális információt. Amint ez sikerült, fölé helyezhető egy további réteg, akár egy szófaji címkéző, névelem-felismerő, vagy bármi, amiről eddig szó volt. Ezeket utána pedig külön feladatokként is lehet finomhangolni, ezért annyira praktikus egy BERT nyelvi modell.

5. fejezet

Kapcsolódó munkák

Számos - a kutatásomban használt rendszerhez - hasonló nyelvi elemző létezik, például az angol BERT nyelvi modell ([Devlin és mtsai, 2018](#)), aminek a belső szerkezete szinte teljesen megegyezik az általam használt nyelvi modellhez ([Nemeskey, 2021](#)). Magyar nyelvre nem létezik még teljesen transzformer alapú előfeldolgozó rendszer, amiben megtalálható az összes elemző. Létezik azonban egy hatékony transzformer alapú elemző, az emBERT ([Nemeskey, 2020](#)), ami kifejezetten csak névelem-felismerésre fejlesztett előfeldolgozó rendszer. Ami nem mellesleg ugyanúgy a huBERT előtanított nyelvi modelljét használja ([Nemeskey, 2021](#)) ugyanúgy, ahogy az általam fejlesztett rendszer is.

Léteznek más magyar nyelvű előelemzőláncok is. Kutatásom célja, hogy megvizsgáljam, hogy a legújabb transzformer alapú előelemzők hogyan teljesítenek a korábbi megoldásokkal szemben. A rendszerek a következők: HuSpaCy CNN (*Convolutional Neural Networks*) alapú large modellje ([Orosz és mtsai, 2022](#)), magyarlánc ([Zsibrita és mtsai, 2013](#)), Stanza ([Qi és mtsai, 2020](#)), UDPipe ([Straka, 2018](#)) és az emtsv ([Simon és mtsai, 2020](#); [Indig és mtsai, 2019a,b](#); [Váradí és mtsai, 2018, 2017](#)), amibe bele lett integrálva az emBERT ([Nemeskey, 2020](#)) transzformer alapú NER rendszere. A feladatok tekintetében a HuSpaCy és a Stanza képes elemezni az összes eddig taglalt feladatot, illetve az emtsv is most már, mert kiadástól nézve 2 év után belekerült a névelem-felismerés (viszont a függőségi elemzője nem UD formátumú). Az UDPipe-ből és a magyarláncból ebben az esetben a névelem-felismerő maradt ki az eszközök tárházából. Létezik továbbá még egy önmagában névelem-felismerő rendszer ([Szarvas és mtsai, 2006b](#)) is.

6. fejezet

Rendszer felépítése

A rendszer felépítése áll tehát egy szavakra (tokenekre) bontóból, mondatra bontóból, szótövesítőből, szófaji címkézőből, morfológiai elemzőből, függőségi elemzőből és végül a névelem-felismerőből. Mindegyik elemző alapja az előtanított nyelvi modell, vagyis a huBERT (Nemeskey, 2021). Felette pedig mindegyik célfeladathoz külön elemző tartozik. Pontosabban a mondatra bontó, szófaji címkéző és a morfológiai elemző esetén a spaCy alapértelmezett rendszerét választottam. A szótövesítőt illetően egy úgynevezett *Edit-Tree-Lemmatizer* lett alkalmazva, ami a spaCy-ben egy új rendszernek mondható. A függőségi elemző helyére egy *biaffine* nevezetű gráf alapú függőségi elemző került, ami ugyancsak egy új fejlesztés a spaCy-ben. A névelem-felismerő esetében pedig szintúgy egy alapértelmezett rendszert választottam, de annyi kiegészítéssel, hogy *beam-search* lett alkalmazva a névelem-felismerésre. Mivel nem elérhető magyar nyelvre egy olyan korpusz, ami tartalmazná az összes elemzőt, ezért egyéb módszerek segítségével tanítottam őket. Ugyanakkor az elemzők esetében is, ahogyan szó volt róla, kipróbálásra kerültek egyéb megoldások, amit kifejték a 6.3. és 6.4. fejezetekben.

6.1. Többfeladatos tanulás

A többfeladatos tanulás, angol nevén a *multi-task learning* lényege, hogy egyszerre több feladatot tanul a háló és így olyan kapcsolatok jönnek létre, amelyek képesek segíteni egymásnak implicit módon. Ennek köszönhetően a különböző elemzők akár akarva-akaratlanul, hozzájárulnak egymás sikeréhez. Például a szótövesítő figyelembe

tudja venni a szófajokat, amely hasznos tud lenni, ha nehezen tud dönteni.

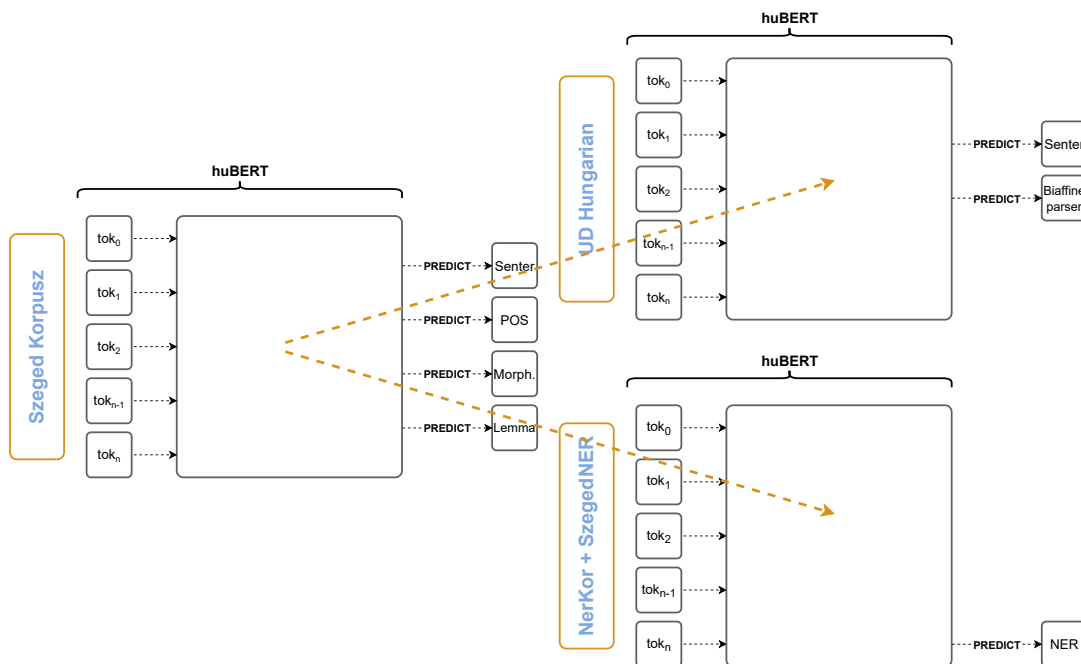
A Szeged Korpusz egyszerre tartalmaz szótöveket, szófajokat, morfológiai leírást és mondathatárokat, így az ezekre épülő elemzőket egy lépésben többfeladatos tanulási feladatként felírva tanítottam.

6.2. Transzfer tanulás

Olyan rendszerek esetében, mint amilyen a kutatásban is foglalkozok, a feladatok nagyon hasonlítanak egymáshoz. Ezért létezik egy olyan eljárás, ami képes arra, hogy a hasonló feladatok tudjanak segíteni egymásnak. Angol nevén a *transfer learning* lényege, hogy van egy probléma, ennek eltávolítjuk a megszerzett tudását és átruházzuk egy másik hasonló problémára, ami így képes arra, hogy figyelembe vegye az eddig tanultakat. Például személygépkocsikat kell felismerni a rendszernek és ezt a tudást átadjuk egy olyan rendszernek, ami teherautókat ismer fel.

Mint arról szó esett, három adatbázist használtam fel a fejlesztés során. Erre azért volt szükség, mert magyar nyelvre nincsen egy átfogó korpusz, ami tartalmazna minden számunkra szükséges elemzést. Így ennek kiküszöbölésére azt a döntést hoztam, hogy a SZC (Szeged Korpusz) és az UD (Universal Dependencies) korpuszokat is felhasználom a mondatra bontó, szófaji címkéző, morfológiai elemző és a szótövező rendszerekhez. Viszont ahogy a bekezdés elején szó volt, a SZC-ben nincs kézzel annotált Universal Dependencies szabványnak megfelelő függőségi elemzés, ezért külön vettem a két folyamatot. Amit lehetett, tehát a függőségi elemző (és persze a névelem-felismerő) kivételével mindent az SZC-n tanítottam. Ezután folyamodtam a továbbtanításhoz, felhasználva így az SZC adta információkat a függőségi elemzőnek, ha nem is explicit módon. A tesztek alapján úgy tűnt, hogy csak a mondatra bontó továbbtanítása hoz érdemi részt, így csak ez lett továbbtanítva az elemzők közül. Természetesen maga a modell szintén tovább lett tanítva az UD-n. Így ahogy az 6.1. ábrán is látszik, a bal oldali modellt továbbtanítom a függőségi elemzővel és a mondatra bontóval a már meglévő függőségekkel, amely így már a névelem-felismerőn kívül minden elemzőt tartalmaz valamilyen formában.

Ugyanezt a lépést megtettem a névelem-felismerő esetében is, ahogy az 6.1. ábra alsó modelljét vesszük szemügyre. A gondolat ugyanaz, egyszerűen csak fogjuk a belső



6.1. ábra. Transzfer tanulás ábrázolása.

szerkezetét (tehát a súlyokat) az első modellnek és továbbadjuk, mert a tesztek alapján sikerült 89,88%-ról 90,99%-ra javítani a pontosságot. Ez nem feltétlenül számít soknak, de a számok alapján segíthet a névelem-felismerőnek az, ha tudja a szónak a szófaját és az egyéb jellemzőit.

6.3. Gráf alapú függőségi elemző

Annak érdekében, hogy javítsam a függőségi elemzés hatékonyságát, lecseréltem a rendszer által alapértelmezésként használt átmenet alapú elemzőt egy gráf alapú, biaffine figyelmi (*attention*) mechanizmussal rendelkező megoldásra (Dozat és Manning, 2016). Az átmenet alapú rendszerek a függőségi fát osztályozási problémák sorozataként építik fel, amikben lehetőség van két vizsgált szó között él behúzására vagy balról jobbra haladva a vizsgált szavak léptetésére. Ezek a megoldások a mohó, lokális döntéseik miatt rosszul tudnak teljesíteni hosszú mondatok esetén. Ezzel szemben a gráf alapú elemzők esetén az algoritmus elején van egy olyan gráf, ahol mindegyik szó össze van kötve egymással. Ezután pedig próbálnak optimalizálni az élsúlyok segítségével és elhagyni olyan éleket, amelyeknek alacsony a valószínűsége. Céljuk, hogy egy maximális feszítőfát alkítsanak ki.

6.4. Beam search alkalmazása névelem-felismerésre

A névelem-felismerő esetén is sikerült egy hasonló rendszert találni, amivel jobb eredményt lehetett elérni, viszont ebben az esetben, már kisebb volt a javulás. Ez a *beam ner* nevet kapta, ami gyakorlatilag "beam search"-öt csinál. Ennek az alapja szintúgy egy átmenet alapú elemző annyi különbséggel, hogy itt folyamatosan nézi a korábbi lépésekben kapott legjobb eredményeket, míg ugyebár a függőségi elemző esetében mindig csak két szót vizsgál, ahogyan az a 1. fejezetben taglalva volt.

7. fejezet

Eredmények

7.1. Eredmények, összehasonlítások

Hogy mérni lehessen a rendszer hatékonyságát, összevetettem a legnépszerűbb rendszerekkel. A kiértékelések (a névelem-felismerő kivételével) a Hungarian Universal Dependencies Corpus tesztkészletén (De Marneffe és mtsai, 2021) készültek és a CoNLL 2018 Shared Task-on¹ alkalmazott szkript felhasználásával történtek. Összehasonlítás-képpen négy rendszert választottam. Az első az emtsv-t (Simon és mtsai, 2020; Indig és mtsai, 2019a,b; Váradi és mtsai, 2018, 2017), amely kifejezetten a magyar nyelv igényeire lett fejlesztve, második az UDPipe (Straka, 2018), amelyet alaprendszerként használnak a CoNLL versenyeken, és a harmdik a Stanza (Qi és mtsai, 2020), amely magas pontszámokat ért el az UD korpuszok terén. Ugyanakkor a nemrégiben publikált HuS-paCy large (Orosz és mtsai, 2022) modelljével is össze fogom hasonlítani, ami a webcorpus 2.0-n (Halácsy és mtsai, 2004) tanított statikus szóbeágyazásokat, illetve egy 4 rétegű CNN hálót használ.

A Stanza² és UDPipe³ szerzői már kimérték a rendszerüket az UD tesztalmazon, de ez nem mondható el az emstv-ről. Itt a rendszert az alapértelmezett konfigurációs beállítások megtartásával lefutattam az általam alkalmazott tesztalmazon. Ugyanakkor fontos kiemelni, hogy az emtsv eredményei olyan szempontból nem számítanak becsületesnek, hogy nem lett újratanítva az általam használt adatbázis vágásokkal, hanem csak egyszerű

¹https://universaldependencies.org/conll18/conll18_ud_eval.py

²<https://stanfordnlp.github.io/stanza/performance.html>

³<https://ufal.mff.cuni.cz/udpipe/1/models>

kiértékelés történt. Így a kiértékelés során előfordulhatott átfedés az adott adathalmazok között.

7.1.1. Tokenizálás, mondatra bontás

A 7.1. táblázatban szereplő számok F1-score-t mutatnak. A számok ismeretében arra lehet következtetni, hogy a szavakra bontásban mindegyik rendszer megfelelően működik. Viszont a HuSpaCy (Orosz és mtsai, 2022) szabály alapú tokenizálója nagyon minimálisan, de felülmúlja a többi rendszer pontosságát. A mondatra bontás viszont már összetettebb feladat, ebben az esetben már az látható, hogy minimálisan, de a transformer modell meghaladja a társai eredményét.

	Tokenizálás	Mondatra bontás
Stanza	99,87%	97,00%
UDPipe	99,80%	95,90%
emtsv	99,77%	98,67%
HuSpaCy large	99,89%	97,66%
HuSpaCy transformer	99,89%	98,89%

7.1. táblázat. Tokenizálás és mondatra bontás eredmények mérése az UD teszhalmazán.

7.1.2. Morfó-szintaktikai elemzés

A 7.2. táblázatban már kicsit érdekesebb az eredmények. Kiemelhető, hogy a transformer modell szépen teljesít minden területen, de néhol még így is akad vetélytársa. Ebben az esetben a morfológiai elemzésben a Stanza minimálisan, de jobb a transformer modellnél. A szófaji címkézés esetében viszont pontosabb, ugyanis a large modellhez képest is sikerült 3 százalékpontot javítani a transformer modellnek. Nem mellesleg a szótövezés feladatban is legalább 4 százalékpontot javított minden rendszerhez képest.

	Szótövezés pontosság	PoS pontosság	Morf. pontosság
Stanza	94,25%	96,03%	93,76%
UDPipe	88,50%	90,60%	88,50%
emtsv	94,94%	89,19%	89,12%
HuSpaCy large	94,82%	94,70%	89,03%
HuSpaCy transformer	98,56%	97,21%	93,60%

7.2. táblázat. Szótövesítés, szófaji címkéző (PoS) és morfológiai elemző kiértékelése az UD teszt-halmazán.

7.1.3. Gráf alapú függőségi elemző eredményei

A 7.3. táblázatban látható az átmenet alapú függőségi elemző, gráf alapú függőségi elemző, illetve a többi rendszer eredményei. Mivel az emtsv nem szolgáltat Universal Dependencies formátumú függőségi elemzést, így ő ebben az esetben nem mérhető ezzel a feladattal.

	UAS	LAS
Stanza	83,62%	78,86%
UDPipe	72,80%	67,20%
HuSpaCy large	79,03%	73,17%
HuSpaCy transformer átmenet alapú	89,95%	83,94%
HuSpaCy transformer gráf alapú	91,42%	87,54%

7.3. táblázat. Átmenet alapú és gráf alapú függőségi elemzés összehasonlítása a meglévő rendszerekkel.

Az eredményeket megnézve megkérdőjelezhetetlen volt, hogy a gráf alapú (biaffine) rendszer mellett döntsek, ugyanis képes volt közel 1,5 százalékpontot javítani az UAS esetében. Azonban a LAS-t illetőleg megközelítőleg több, mint 3,5 százalékpontot sikerült javítani. Ugyanakkor fontos kiemelni, hogy gráf alapú (biaffine) függőségi elemzővel kimagasló eredményt sikerült elérni, ami jelenleg state-of-the-art eredménynek számít.

7.1.4. Névelem-felismerés

A névelem-felismerő esetében sajnos picit más a helyzet a kiértékelések gyanánt. Mivel az UD nem tartalmaz névelemeket, ezért erre a már tárgyalt NerKor (Simon és Vadász, 2021) és a SzegedNER (Szarvas és mtsai, 2006a) korpuszt használtam. Az UDPipe-ban sajnos nincs névelem-felismerő, így azt nem lehetett bevonni a vizsgálatba. Hogy bővítssem a repertoárt, kettő korábbi névelem-felismerőt is bevettem az összehasonlításba, hogy még nagyobb képet kapjunk. Az egyik a Szarvas és mtsai (2006b), amely döntési fákat használ. A másik pedig Simon (2013), amely egy lineáris modellt használ és rejtett Markov modelleket kombináló statisztikai címkézővel rendelkezik. A harmadik új rendszer pedig az emBERT. A Stanza esetén viszont fontos kiemelni, hogy ebben az esetben újra lett tanítva az általam használt adatbázisokkal (Eszter és mtsai, 2022).

	SzegedNER	NerKor	Kombinált
Simon (2013)	95,06%	–	–
Szarvas és mtsai (2006b)	94,77%	–	–
emBERT	97,40%	92,09%	92,99%
Stanza	91,78%	80,53%	83,75%
HuSpaCy large	95,31%	80,75%	84,56%
HuSpaCy transformer hagyományos NER	97,01%	88,27%	90,46%
HuSpaCy transformer Beam NER	97,37%	89,13%	90,99%

7.4. táblázat. F1-score-on mért névelem-felismerés összehasonlítása a SzegedNER tesztalmazon, a NerKor tesztalmazon és a kombinált halmazon is.

A NerKor adatbázis esetében a szerzők által meghatározott tanító-validáló-teszt halmazokra bontást alkalmaztam. A SzegedNER esetében pedig az összehasonlítás érdekében a (Szarvas és mtsai, 2006b) cikk által meghatározott vágást alkalmaztam.

Így tehát mivel bevettem a vizsgálatba a régebbi rendszereket, ezért külön korpuszokra is le lettek mérve a rendszerek. A Beam search alkalmazása hozzávetőlegesen 0,5 százalékpontot javított a meglévő rendszeren, ezért ezt a megoldást alkalmaztam a továbbiakban. Az emBERT-tel való összehasonlítás során a SzegedNER korpusz esetében szinte fej-fej mellett van a két modell (emBERT és a Beam NER transformer modell), de sajnos a NerKor esetében kicsit alulmarad. Ebből az okból kifolyólag a kombinált korpuszon is jobb az emBERT, ami nem véletlen, mert a 12 réteg feletti CRF nagyban hozzájárul a jelenlegi sikeréhez. Lehetséges, hogy az általam fejlesztett rendszernek is segítene, ezért további kísérletek gyanánt jövőbeli terv, hogy készüljön a HuSpaCy transformer modellhez is egy huBERT-CRF modell is, mert jelenleg nem létezik ilyen implementáció a spaCy-ben.

7.2. Memóriahasználat

Az olyan erőforrásigények fontos szerepet játszanak ezenknél a rendszereknél, mint például a gyorsaság és a memóriahasználat. Ezért ebben a fejezetben összehasonlítottam⁴ az eddig taglalt rendszereket. A többi rendszere való tekintettel az igazságos összehasonlítás a névelem-felismerő nélkül fog történni. Azonban a large modellel lesz még egy kisebb összehasonlítás a névelem-felismerővel is és ezúttal már GPU-val, hogy nagyobb képet kapjunk az erőforrásigényekről. A mérések az UD tesztalmazán készültek.

A 7.5. táblázatban jól látható, hogy a transformer modell a többi rendszerhez képest több erőforrást igényel. Egyedül az emtsv, amivel szinte ugyanazok az eredmények jöttek ki. A Stanza-hoz képest a gyorsaság nem szembeötlő, viszont ebben az esetben a memóriahasználata rosszabb a transformer modellnek. Az UDPipe ellenben mind a két esetben nagyszerűen remekel a memóriaigények tekintetében, ugyanakkor ez a rendszer minden elemző esetén alul marad a többi rendszerhez képest.

⁴Minden kísérletet egy Intel Core i7-8750H CPU-val rendelkező számítógépen végeztem, 16 GB RAM Ubuntu Linux 20.04 LTS rendszerrel.

	Gyorsaság (token/mp)	Memóriahasználat (GB)
Stanza	222	0,9
UDPipe	1741	0,4
emtsv	122	3,9
HuSpaCy large	1364	3,0
HuSpaCy transformer	123	4,2

7.5. táblázat. A CPU-n összehasonlított rendszerek gyorsasága (token/másodpercben mérve) és a maximális memóriahasználatuk.

Megfigyelhető az is, hogy a transformer modell memóriahasználatban képes tartani a lépést az elődjéhez képest hozzávetőlegesen, tehát egy mai középkeletgóriás számítógép képes megbirkózni vele. Nézzük meg, mit produkálna a GPU-n⁵ és immáron a névelem-felismerővel együtt a két rendszer.

	Gyorsaság (token/mp)	Memóriahasználat (GB)
HuSpaCy large	4882	4,1
HuSpaCy transformer	699	4,3

7.6. táblázat. A large és a transformer modell összehasonlítása a GPU-n, névelem-felismerő rendszerrel.

Jól látható a 7.6. táblázatban, hogy a memóriahasználata nem tér el annyira egymáshoz képest a két modell, viszont időben nézve jócskán le van maradva a társától a transformer modell. Ezeket mérlegelve azt lehet leszűrni, hogy ha van egy célfeladat és annyira nem számít az idő, akkor ott mindenképpen a pontosabb transformer modellt ajánlanám. Viszont ahol közrejátszik az időtényező, tehát például egy chat-bot alkalmazásnál, ott mindenképpen egy gyorsabb modellt ajánlanék, még ha kicsit pontatlanabb is. Ugyanakkor a large modellhez képest, ha a GPU-t használunk, akkor 82 másodpercről 13 másodpercre csökken (UD teszthalmazán tesztelve) a futásidő, ami jelentősen kevesebb.

⁵GTX1060 Max-Q 6 GB

7.3. Jövőbeli tervek

A kutatás során talákoztam pár olyan hibával, amivel nem számoltam. Kiderült, hogy a morfológiai elemző egy adott esetet szinte mindig elront. Ezt a hibát, ha sikerülne a jövőben kijavítani, akkor elképzelhető, hogy túlszárnyalja a legjobb jelenlegi rendszert is. A névelem-felismerőt illetően igaz nem fordult elő hasonló hiba, viszont ahogyan szó volt az 7.2 táblázatban, lehetséges, hogy a huBERT fölé helyezett CRF réteg akár pár százalékpontot is képes lenne javítani a jelenlegi pontosságán. Ezzel sikerülhetne elérni az emBERT számait, azaz state-of-the-art eredményeket is. Ugyan ez nem elérhető a spaCy-ben, ezért kézzel kellene leimplementálni, de mégis olyan továbbfejlesztés lenne, amivel talán sikerülhetne elérni, hogy minden területen jobbnak bizonyuljon ez a rendszer magyar nyelvre.

8. fejezet

Összegzés

Elkészítettem a HuSpaCy-hez¹ egy transzformer alapú modellt², ami minden hagyományos nyelvi elemzőt, tokenizálót, mondatra bontót, szótövezőt, szófaji és morfológiai egyértelműsítőt, függőségi elemzőt, illetve ezeken felül névelem-felismerőt is tartalmaz, továbbá ezen eredmények teljes mértékben reprodukálhatóak. Első helyen végzett a hét feladatból ötön a magyarra már meglévő rendszerekkel összehasonlítva. Dolgozatomban bemutattam, hogy a transformer modellt és a rá épülő elemzők hogyan működnek. A rendszer hátránya a hosszú futási idő. Olyan alkalmazások esetén, ahol fontos a gyorsaság, alkalmasabb lehet egy másik megoldást választani, ám ahol ez a szempont kevésbé hangsúlyos, ott a transzformer alapú megoldás adja a legjobb eredményt. Ugyanakkor, ha rendelkezünk egy jobb GPU-val, akkor akár az időtényező is elenyészőnek számíthat.

A függőségi elemzés és a névelem-felismerés problémákra több megoldást is kipróbáltam és rendszeremben a magyar nyelvre legmegfelelőbbet alkalmaztam. Az eredmények magukért beszélnek, a függőségi elemző esetén hatalmas ugrást tett a transzformer alapú megoldás az elődjeihez képest, így sikerült egy state-of-the-art rendszert létrehoznom. Összefoglalva kijelenthető, hogy a várakozásomnak megfelelően az eddigi rendszereken felül teljesített a transformer modell.

¹<https://github.com/huspace/huspace/tree/develop>

²https://huggingface.co/huspace/hu_core_news_trf

Köszönetnyilvánítás

Kutatásaimat az Innovációs és Technológiai Minisztérium és a Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal támogatta a Mesterséges Intelligencia Nemzeti Laboratórium keretében.

Irodalomjegyzék

- Booij, G.: The grammar of words: An introduction to linguistic morphology. Oxford University Press (2012)
- Csendes, D., Csirik, J., Gyimóthy, T.: The szeged corpus: A pos tagged and syntactically annotated hungarian natural language corpus. In: International Conference on Text, Speech and Dialogue. pp. 41–47. Springer (2004)
- De Marneffe, M.C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., Manning, C.D.: Universal stanford dependencies: A cross-linguistic typology. In: LREC. vol. 14, pp. 4585–4592 (2014)
- De Marneffe, M.C., Manning, C.D., Nivre, J., Zeman, D.: Universal dependencies. Computational linguistics 47(2), 255–308 (2021)
- Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
- Ding, N., Xu, G., Chen, Y., Wang, X., Han, X., Xie, P., Zheng, H.T., Liu, Z.: Few-nerd: A few-shot named entity recognition dataset. arXiv preprint arXiv:2105.07464 (2021)
- Dozat, T., Manning, C.D.: Deep biaffine attention for neural dependency parsing. arXiv preprint arXiv:1611.01734 (2016)
- Eszter, S., Vadász, N., Lévai, D., Dávid, N., Orosz, G., Szántó, Z.: Az nytk-nerkor több szempontú kiértékelése (2022), https://www.academia.edu/70256871/Az_NYTK_NerKor_t%C3%B6bb_szempont%C3%BA_ki%C3%A9rt%C3%A9kel%C3%A9se

- Ghahramani, Z.: Unsupervised learning. In: Summer school on machine learning. pp. 72–112. Springer (2003)
- Goldberg, Y., Levy, O.: word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014)
- Halácsy, P., Kornai, A., Németh, L., Rung, A., Szakadát, I., Trón, V.: Creating open language resources for Hungarian. In: Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC’04). European Language Resources Association (ELRA), Lisbon, Portugal (May 2004), <http://www.lrec-conf.org/proceedings/lrec2004/pdf/525.pdf>
- Honnibal, M.: Introducing spaCy (Feb 2015), <https://explosion.ai/blog/introducing-spacy>
- Ikonomakis, M., Kotsiantis, S., Tampakas, V.: Text classification using machine learning techniques. WSEAS transactions on computers 4(8), 966–974 (2005)
- Indig, B., Sass, B., Simon, E., Mittelholcz, I., Vadász, N., Makrai, M.: One format to rule them all – the `emtsv` pipeline for Hungarian. In: Proceedings of the 13th Linguistic Annotation Workshop. pp. 155–165. Association for Computational Linguistics, Florence, Italy (aug 2019a), <https://www.aclweb.org/anthology/W19-4018>
- Indig, B., Sass, B., Simon, E., Mittelholcz, I., Kundráth, P., Vadász, N.: `emtsv` — egy formátum mind felett. In: Berend, G., Gosztolya, G., Vincze, V. (szerk.) XV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2019). pp. 235–247. Szegedi Tudományegyetem Informatikai Tanszékcsoport, Szeged (2019b)
- Jurafsky, D., Martin, J.H.: Sequence labeling for parts of speech and named entities. Speech and Language Processing (2020)
- Kubat, M., Kubat: An introduction to machine learning, vol. 2. Springer (2017)
- Learned-Miller, E.G.: Introduction to supervised learning. I: Department of Computer Science, University of Massachusetts p. 3 (2014)

- Maier, A., Syben, C., Lasser, T., Riess, C.: A gentle introduction to deep learning in medical image processing. *Zeitschrift für Medizinische Physik* 29(2), 86–101 (2019)
- Mitchell, T.M.: Does machine learning really work? *AI magazine* 18(3), 11–11 (1997)
- Nadkarni, P.M., Ohno-Machado, L., Chapman, W.W.: Natural language processing: an introduction. *Journal of the American Medical Informatics Association* 18(5), 544–551 (2011)
- Nemeskey, D.M.: Egy embert próbáló feladat (2020)
- Nemeskey, D.M.: Introducing huBERT. In: XVII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2021). p. TBA. Szeged (2021)
- Nivre, J., Agić, Ž., Ahrenberg, L., Antonsen, L., Aranzabe, M.J., Asahara, M., Ateyah, L., Attia, M., Atutxa, A., Augustinus, L., és mtsai: Universal dependencies 2.1 (2017)
- Novák, A., Novák, B.: Nerkor 1.41 e (2022)
- Orosz, G., Szántó, Z., Berkecz, P., Szabó, G., Farkas, R.: Huspacy: an industrial-strength hungarian natural language processing toolkit. *arXiv preprint arXiv:2201.01956* (2022)
- O’shea, T., Hoydis, J.: An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking* 3(4), 563–575 (2017)
- Qi, P., Zhang, Y., Zhang, Y., Bolton, J., Manning, C.D.: Stanza: A python natural language processing toolkit for many human languages. *arXiv preprint arXiv:2003.07082* (2020)
- Simon, E.: Approaches to Hungarian Named Entity Recognition. Ph.D.-értekezés, PhD School in Cognitive Sciences, Budapest University of Technology and Economics (2013)
- Simon, E., Indig, B., Kalivoda, Á., Mittelholcz Iván, S.B., Vadász, N.: Újabb fejlemények az e-magyar háza táján. In: Berend, G., Gosztolya, G., Vincze, V. (szerk.) XVI. Magyar Számítógépes Nyelvészeti Konferencia. pp. 29–42. Szegedi Tudományegyetem Informatikai Tanszékcsoport, Szeged (2020)

- Simon, E., Vadász, N.: Introducing nytk-nerkor, A gold standard hungarian named entity annotated corpus. In: Ekstein, K., Pártl, F., Konopík, M. (szerk.) Text, Speech, and Dialogue - 24th International Conference, TSD 2021, Olomouc, Czech Republic, September 6-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12848, pp. 222–234. Springer (2021)
- Straka, M.: UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In: Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. pp. 197–207. Association for Computational Linguistics, Brussels, Belgium (Oct 2018), <https://www.aclweb.org/anthology/K18-2020>
- Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
- Szarvas, G., Farkas, R., Felföldi, L., Kocsor, A., Csirik, J.: A highly accurate named entity corpus for hungarian. In: Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06) (2006a)
- Szarvas, G., Farkas, R., Kocsor, A.: A multilingual named entity recognition system using boosting and C4.5 decision tree learning algorithms. In: International Conference on Discovery Science. pp. 267–278. Springer (2006b)
- Váradí, T., Simon, E., Sass, B., Gerőcs, M., Mittelholtz, I., Novák, A., Indig, B., Prószéky, G., Vincze, V.: Az e-magyar digitális nyelvfeldolgozó rendszer (2017)
- Váradí, T., Simon, E., Sass, B., Mittelholcz, I., Novák, A., Indig, B.: E-magyar—a digital language processing system (2018)
- Vincze, V., Simkó, K., Szántó, Z., Farkas, R.: Universal Dependencies and morphology for Hungarian - and on the price of universality. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. pp. 356–365. Association for Computational Linguistics, Valencia, Spain (Apr 2017), <https://aclanthology.org/E17-1034>
- Weischedel, R., Palmer, M., Marcus, M., Hovy, E., Pradhan, S., Ramshaw, L., Xue, N., Taylor, A., Kaufman, J., Franchini, M., és mtsai: Ontonotes release 5.0 ldc2013t19. Linguistic Data Consortium, Philadelphia, PA 23 (2013)

Zsibrita, J., Vincze, V., Farkas, R.: magyarlanc: A Toolkit for Morphological and Dependency Parsing of Hungarian. In: Proceedings of Recent Advances in Natural Language Processing 2013, pp. 763–771. Association for Computational Linguistics, Hissar, Bulgaria (2013)