TANSZÉKVEZETŐ

DIPLOMATERVEZÉSI FELADAT

Kulcsár Dániel
Mérnökinformatikus hallgató részére

Nyelvmodellek kiértékelése magyar nyelvre

A nyelvmodellek olyan neurális hálózatok, amelyek mondatokhoz rendelnek valószínűségi eloszlásokat. A gyakorlatban szavakhoz és mondatokhoz rendelnek folytonos vektorokat, amik más neurális komponensek bemeneteként szolgálnak. A statikus nyelvmodellek fix vektorokat rendelnek egy-egy szóhoz, míg a dinamikus nyelvmodellek az egész mondat alapján hozzák létre a szavak reprezentációját, így ugyanaz a szó más-más vektorral van reprezentálva a kontextus függvényében. Közös jellemzőjük, hogy nagy adaton tanított, akár több száz millió paraméterrel rendelkező neurális hálózatok.

A nagyméretű előretanított nyelvmodellek (pretrained language models) az utóbbi években forradalmasították a természetes nyelfeldolgozást. Ezek a nyelvmodellek széles körben használtak magasabb szintű alkalmazásokban, mint pl. a gépi fordítás, a beszédfelismerés vagy a szövegkivonatolás. A legismertebb ilyen nyelvmodell, a 2018-ban több változatban kiadott BERT, amelyek közül az egyik változat 100 nyelvet támogat, köztük a magyart. A soknyelvű BERT-et a Google kutatói egyszerre tanították a 100 legnagyobb Wikipédián, amik között a magyar is szerepelt. Nemrég elkészült a csak magyar adaton tanított BERT változat, a HuBERT.

A hallgató feladata a többnyelvű BERT kiértékelése magyar nyelvre és összehasoníltása a magyar HuBERT-tel. A kiértékelést elsősorban morfológiai feladatokon végezzük, amikhez jelenleg kevés tesztelőadat áll rendelkezésre. A morfológiai feladatok mellett a lista bővíthető szekvenciális címkézési feladatokkal pl. Part-of-speech tagging vagy named entity recognition (névelemfelismerés).

A hallgató feladatának a következőkre kell kiterjednie:

- Magyar nyelvű tesztelőadatok bővítése.
- A BERT kiértékelése magyarra és potenciálisan más kevés erőforrással rendelkező nyelvre
- A BERT összehasonlítása más soknyelvű modellel pl. Az XLM-RoBERTával.
- A BERT összehasonlítása a magyar HuBERT-tel.
- A hibák automatikus és kézi elemzés

Tanszéki konzulens: Ács Judit, tanársegéd

Budapest, 2020. október 10.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék
H-1117 Budapest, Magyar tudósok krt 2. Q.B.207
Telefon: 463-2870. Fax: 463-2871
www.aut.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Nyelvmodellek kiértékelése magyar nyelvre

Szakdolgozat

Készítette

Kulcsár Dániel

Konzulens

Ács Judit

2020. december 11.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Kulcsár Dániel*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) 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. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. december 11.

Kulcsár Dániel
hallgató

Kivonat

Jelen dokumentum célja néhány mély tanuló nyelvi reprezentáló model megismerése és ezek összehasonlítása teljesítmény szempontjából. Ehhez magyar nyelvű adatokon értékelem ki és vizsgálom meg a modeleket. A nyelvi reprezentáló modelek a BERT [3] modelnek egy-egy verziói. Az összehasonlítást probing segítségével végzem el, ezzel mérem meg a BERT modelek teljesítőképességét. A probing széles körben elterjedt "black box" modelek működésének megismerésére. Jelen esetben a BERT modelnél arra használom, hogy megtudjuk mennyi információt sikerült kinyernie a kontextusból a modelnek, és ez az információ mennyire alkalmazható egy későbbi task/feladat megoldására. Egy osztályozás során mérem meg a pontosságát a modeleknek. A lehetséges osztályokat az adatfájlok tartalmazzák.

Az összehasonlítás során a model egyes rejtett rétegeinek teljesítőképességét és a hibák kiértékelését is elvégzem. A cél ezekből levonni a következtetéseket az adott BERT modelek képességeire és a nyelvi reprezentációs modelek jövőjére, a magyar nyelvvel szerzett tapasztalatok alapján.

Abstract

This document's goal is to learn more about a few language representation models and compare them to each other by their performance. For this, there will be hungarian datasets evaluated on each of the models for further examination. The said language representation models are all based on the BERT [3] model. The comparition will happen through probing, thus measuring the performance of each BERT model. The probing is a technique that is often used to get to know more about "black box" models. In this paper we will use it for the BERT models in a way, so we can measure the amount of information, the BERT model extracted from the context of the words. Also measuring how this information can be used in downstream tasks. The task we measure is classification, which is supported by the datasets' file structure.

Through the comparition we will review the BERT model's individual hidden layers' performance and fails as well. The goal is to conclude how well the BERT models performed, and what the future can bring in the language representation tasks in hungarian.

1. fejezet

Bevezetés

A dolgozat célja különböző mély tanuló modelek összehasonlítása, olyan szempontokból, hogy melyikből tudunk probing segítségével pontosabb eredményt elérni, ha magyar nyelvű adatokat használunk. Ezen modelek alapja a BERT [3] (Bidirectional Encoder Representations from Transformers) nevű, a HuggingFace által fejlesztett Transformerre [7] épülő, nyelvi reprezentációs model. Ezen model egyik legnagyobb újítása a korábbiakkal szemben az volt, hogy mind a szórész előtti, mind a szórész utáni kontextusokat is figyelembe vette az eredmény/kimenet előállítása során, illetve self-attention rétegeket is felhasznál. Régóta keresik a mély tanulással foglalkozó kutatók, a nyelvészekkel együtt, a megfelelő modelt, mely úgy képes reprezentálni a szöveget, hogy ennek kimenetéből egyértelműen visszakaphassunk olyan információkat a szövegről, amelyek megkönnyíthetik további feladatok/taskok elvégzését a szövegen. Ilyen feladat vagy task lehet az egyes szavak adott szófajba sorolása, vagy annak eldöntése, hogy melyik többesszámú és melyik egyes számú az adott főnevek közül, ha adva van hozzá a kontextus is. Leggyakrabban az adott feladattól függ, hogy milyen reprezentáció a legjövedelmezőbb megoldandó task megoldására.

A BERT, egy kvázi "black box" model, tehát nem látni a belső működését, pusztán a végeredmény alapján következtethetünk, hogy mennyi információt sikerült kinyerni. A kinyert információ mértékének mérésére lett kitalálva az úgynevezett "probing", aminek lényege, hogy a BERT model kimenete által kinyert információk letesztelésére felépítenek egy neurális hálót. Ennek célja, hogy a neurális háló tanul-e a megfelelő feladat megoldásának érdekében, megfelelő pontosságot ér-e el, majd a tanulás után a teljesen új teszt adatokra, a neurális hálónk az általunk elvárt precizitással működik a feladat elvégzése során. Ezen neurális hálót bármely BERT model kimenetére be tudjuk tanítani, hogy az alapján oldja meg a feladatot. Tehát az egyes megvizsgálandó BERT modeleket azonos

szöveg bemenettel megvizsgáljuk úgy, hogy a kimenetüket a neurális háló felhasználásával leteszteljük. Így kiderül, hogy melyik kimenet segítségével oldható meg nagyobb pontossággal az a feladat, amelynek megoldására tanítjuk a neurális hálót, melyik lesz pontosabb a teszt adatokon, ezzel egy jó összehasonlítási alapot kapunk a BERT modeleink között.

2. fejezet

Model megismerése

2.1. Motiváció

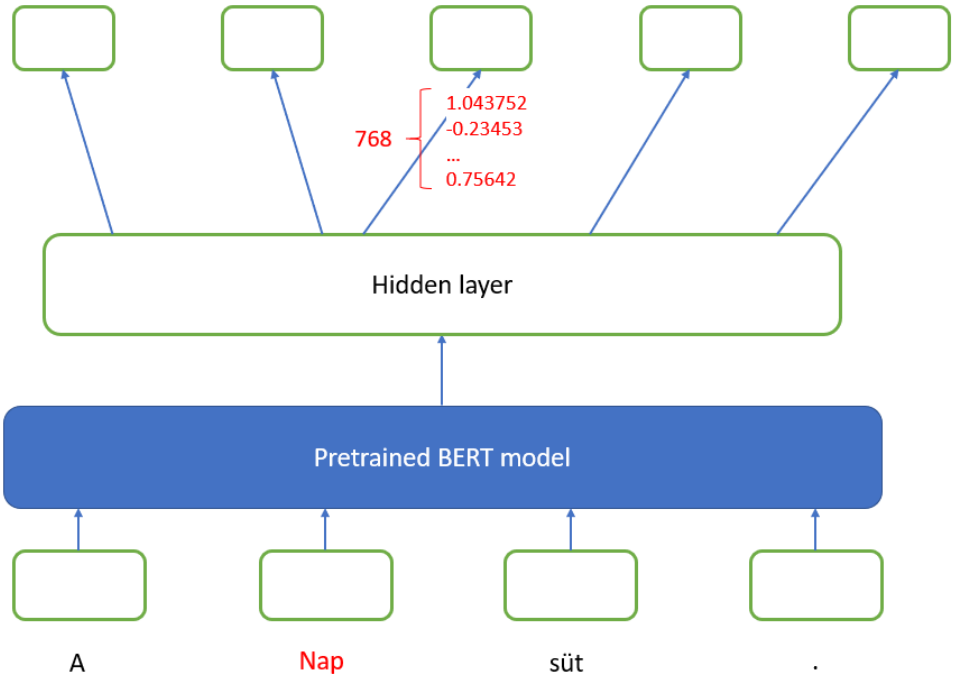
A különböző nyelvi szövegek feldolgozás során felmerülő különböző feladatok, elvégzendő elemzések, valamint végrehajtandó kiértékelések folyamán felmerülő probléma, amely akadályt gördít a vizsgálók elé az, hogy hogyan lehetne úgy reprezentálni egy szöveg által tartalmazott adatokat gépi úton, hogy az az emberek számára is értelmezhető legyen. Rendelkezésre állnak különböző szöveg reprezentáló mély tanuló modelek, amelyek a bemenetként megadott szöveget feldolgozva, abból kontextuális információkat kinyerve gépileg feldolgozható formában ezt a kimenetükön visszaadják. Ezt a kimenetet kell átalakítani általunk is értelmezhető formába. Ehhez meg kell vizsgálni, hogy ezen szöveg reprezentáló modelek hogyan is épülnek fel.

Egy a mai napig előszeretettel alkalmazott model család manapság a természetes nyelv feldolgozás területén magas potenciállal rendelkező BERT model. Ez a model számtalan esetben bizonyult hasznosnak az NLP (Natural Language Processing) területén. Ez a model egy black box model, amely a kihívások nagy részét okozza, hiszen nem látni bele, hogy a model pontosan milyen számításokat miért végez, tehát csak találgathatunk a visszaadott tensorok információtartalmán, annyit tudunk, hogy szórészekre bontja az egyes szavakat egy tokenizáló segítségével és ezen szórészekhez rendel valamilyen kontextuális információkat.

A probléma megoldására az egyik legelterjedtebb és legnépszerűbben alkalmazott metodika az úgynevezett probing vagy probing. Ezen technológia lényege, hogy a BERT model bemeneteként funkcionáló szövegből kiválasztunk egy adott szót, tegyük fel mondatonként, ennek helyzetét számontartjuk és a BERT által visszaadott tensorok közül tudjuk, hogy melyek vonatkoznak erre a szóra. Ezt felhasználva egy másik feladatot/tas-

kot szeretnénk megoldani egy neurális háló segítségével, amely a súlyait a kapott tensorok alapján tanulja meg, és a neurális háló teljesítménye fogja megmondani, hogy mennyire jól tudta kinyerni az információt a kontextusból a BERT model.

A projekt során probinggal különböző teszt adatok segítségével különböző pretrained BERT modelek lettek megvizsgálva, annak érdekében, hogy az eredményeket magyar nyelvű bemeneti szövegek esetén össze lehessen hasonlítani. A probing során használt task/feladat egyes mondatok célszavainak az osztályozása, az utolsó szórész tensora alapján, mely a 2.1. képen is ábrázolva van a mondatban szereplő Nap szóra.

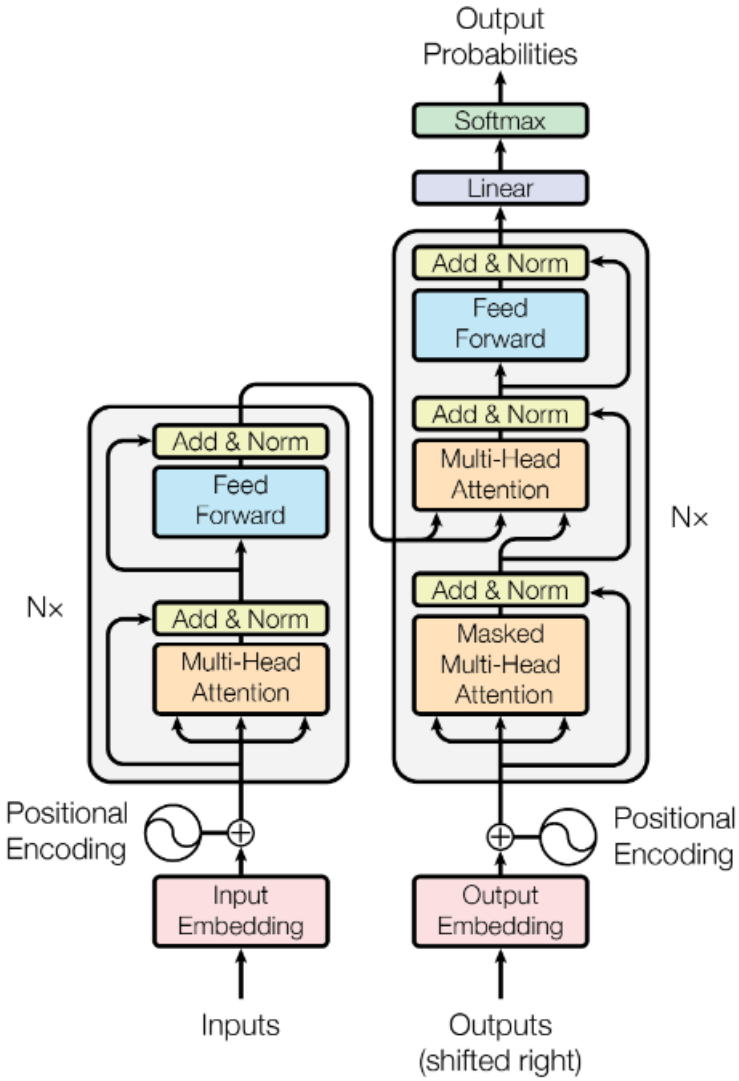


2.1. ábra. a Nap szóra való probing

2.2. Elméleti háttér

A Transformers [7] egy Google által kidolgozott technológia, amelyet sok más mély tanuló model alkalmazott sikeresen a közelmúltban, beleértve a BERT szöveg reprezentáló modelt. A Transformers legfontosabb újítása az volt, hogy az azelőtti megoldásokban használt RNN és LSTM hálókat teljes mértékben lecserélte az attention vagy magyarul „figyelem” mechanizmusra, és így is megfelelő teljesítményt ért el. A model egy kódoló-dekódoló modelként épül fel, mind a kódoló, mind a dekódoló felében a modelnek az attention mechanizmus és egy fully connected Feed Forward network segítségével történik meg a bemenet

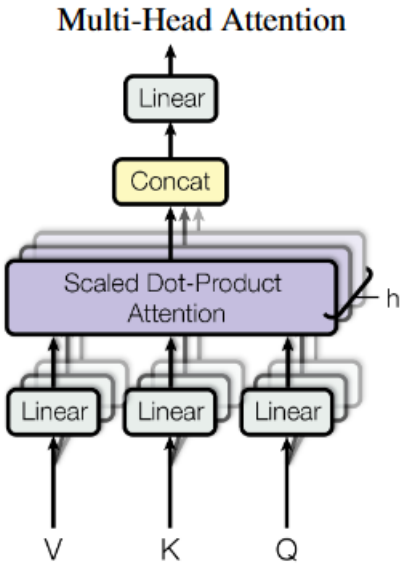
átalakítása, de alkalmaznak úgynevezett residual kapcsolatokat is, amelyek egy-egy részét a hálónak „kikerülik” ezzel a benne rejlő információkat nem károsítva.



2.2. ábra. A Transformers model architektúrája [7]

Az attention mechanizmus lényege az, hogy az egyes részeihez a bemenetnek vektorokat rendel, amelyeknek értékei attól függ, hogy az adott szövegrész mely más szövegben szereplő elemekkel van kapcsolatban, pl. egy mutatónévmás és a szó, amire utal, akkor az egyik a másik attention vektorában magasabb értékeket kap. Ezt multi head attention módszerrel teszi, aminek az a lényege, hogy nem egyszer végzi el a mechanizmust, hanem többször párhuzamosan és végül ezek eredményét konkatenálja, hogy végül a megfelelő attention-ök legyenek hozzárendelve minden egyes szöveg részlethez, és így továbbítódjanak a model következő komponense felé. Végül a Transformers kódolója és dekódolója esetén is a fully connected Feed Forward network fogja a végső jelentősebb átalakításokat

(még van egy softmax és egy linear layer utána dekódolás esetén) elvégezni a bemenet adott részein, mely egy egyszerű néhány rejtett réteggel rendelkező háló, más-más paraméterekkel és dimenziókkal rétegenként.



2.3. ábra. Multi-Head Attention felépítése [7]

A BERT [3] vagyis Bidirectional Encoder Representations from Transformers a nevéből is adódóan az előbb bemutatott Transformers model felhasználásával végzi el a szöveg reprezentáló feladatát. Ennek következtében a BERT is kódoló-dekódoló modelt használ, attention mechanizmussal, viszont mindkét irányban elvégzi ezt, a bal és jobboldali kontextusra is a bemeneti szekvencián. A bemenetet egy tokenizáló segítségével szórészekre (WordPiece embedding) bontja, oly módon, hogy a szótárában szereplő szórészek közül azokra bontja szét a bemeneti szekvenciában szereplő mondatokat, amelyek a legvalószínűbben fordulnak elő a BERT-nek betanított szekvenciákban. A tokenizáló a bemenete elé és mögé különleges tokeneket helyet el, ez a CLS és a SEP tokenek. A tokenizált bemenetből a BERT kinyeri a kontextuális információkat a rejtett rétegei segítségével, melyek száma 12. A dolgozatban vizsgált BERT modelek már előre betanítottak, tehát a rejtett rétegeinek a súlyai már bizonyos tanuló adatok alapján elő lettek készítve ismeretlen inputok számára.

2.3. Tokenizálók összehasonlítása

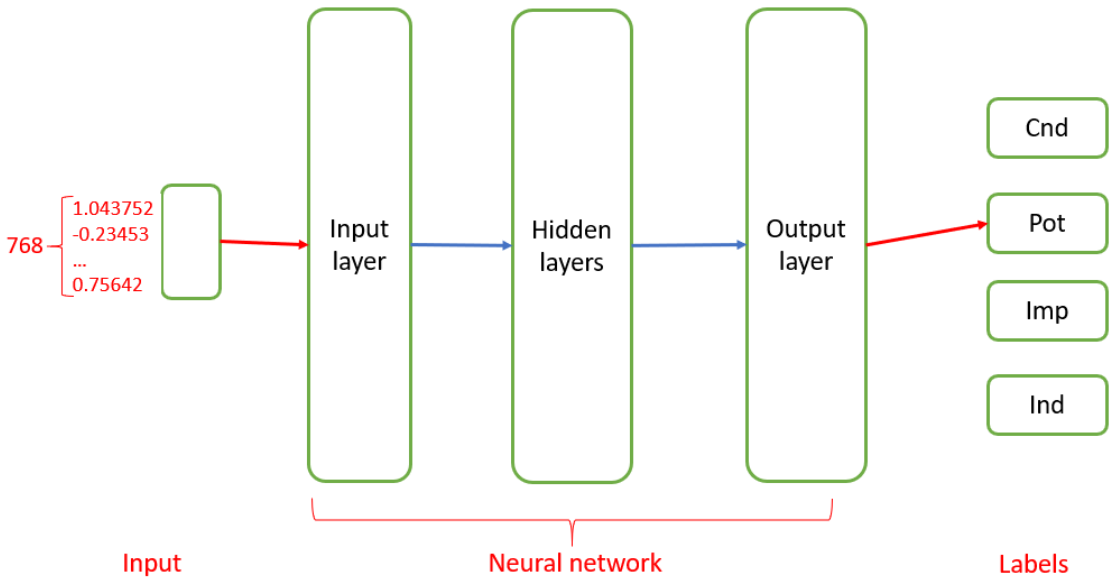
A BERT multilingual kis- és nagybetű érzékeny modeljét teszteltem le tokenizálás szempontjából, különböző bemenetekkel, kézzel. Ezzel egyidőben a kísérletet megismételtem a SZTAKI (Számítástechnikai és automatizálási kutatóintézet) által is fejlesztett magyar BERT-en [5] az eredményeiket pedig összehasonlítottam. Azt tapasztaltam, hogy a magyar model, mivel nagyobb "szótárral" dolgozott, több gyakran felhasznált szórészt vélt felhasználandónak a szöveg tokenizálásakor, ennek hatására pedig kevesebb darabra vágta a szavakat átlagosan. Ez azért lényeges, mert így a további kiértékelések során kisebb méretű tuple-el kell dolgoznia a BERT-nek és a probing hálónak is. Fontos, hogy jó, mindent lefedő adatokon tanult BERT modelt használjunk emiatt, hiszen, ha csak egy szakterületen belül lévő kifejezéseket ismer a szótárában, a más területekről érkező szöveget nagy teljesítményvesztéssel képes csak feldolgozni.

3. fejezet

Összehasonlító program

Ahhoz, hogy több modelt össze lehessen hasonlítani magyar nyelvre, egy programra, melynek segítségével a probingot végző neurális háló eredményeit olyan formában el lehet menteni, hogy azt utána ki lehessen értékelni. A program feladata lesz a bemeneti szöveget a megfelelő BERT modelen átengedni, hogy a BERT rejtett rétegei közül a megfelelő tensort kiválasztva felhasználja azt a neurális háló osztályozásra tanítása céljából. Ehhez a pytorch környezetet alkalmaztam előző félévbeli tapasztalataim miatt, így nem volt teljesen idegen a környezet, amiben dolgoztam, továbbá a Huggingface BERT modelje elsősorban tensorflow és pytorch fejlesztési környezeteket támogat alapértelmezetten, tehát egy támogatott környezetben tudtam dolgozni, amely nyelven jól dokumentált a BERT model, tehát ez volt a legcélszerűbb megoldás. A BERT modelek, amelyek vizsgálatát elvégeztem, mind a Transformers által kidolgozott AutoModel és AutoTokenizer által elérhető modelek, azért ezeket használtam, hiszen így egységesíteni lehetett a modelt, hogy csak egy-egy helyen kell átírni egy string-et, hogy más BERT modelt és tokenizálót vizsgáljon éppen, amit paraméterként is megkaphat a program. A BERT modelek rejtett rétegei által visszaadott kimenetet adom tovább a probing neurális hálónak. Ezen rejtett réteg úgy épül fel, hogy minden egyes szórész kap egy 768 dimenziós vektort, amin keresztül reprezentálta a model az adott szórészt és kontextusát. Karaktereket ugyanúgy veszi, mint az egyes szórészeket, tehát ezek is ugyanúgy rendelkeznek egy-egy vektorral, melyek reprezentálják. Az adott bemeneten szereplő összes szórész 768 méretű tensorai közül egyszerű indexeléssel választottam először ki a megfelelőt, ez azért volt akkor célszerű, mert az adatfájlok tartalmazzák a mondatban szereplő célszó indexét, ezt az indexet tokenizálás alapján megfelelő számmal növelve a kimeneti tensorok között is megtalálható a célszó megfelelő szórészének tensora, egy 768 méretű vektor. Ezt a vektort kellett a későbbiekben a probinghoz hasz-

nált neurális háló bemenetére beadni, hogy rátanuljon az összes teszteset segítségével az osztályozásra. A neurális hálók tanítása során alkalmazott módszer szerint train adatok segítségével tanítottam a hálót, a túltanulás elkerülését pedig a dev adatok segítségével ellenőriztem, ez azért fontos, hogy ismeretlen bemenetek esetén is pontos eredményeket adjon az osztályozó. Ezt a tanulás után a teszt adatokkal ellenőrzöm. A tanuló, teszt és dev adatok kezeléséhez, illetve a program során adatmanipulációra a python pandas környezetét használtam. A DataFrame-ekkel sokkal gyorsabban és egyszerűbben lehetett ekkora mennyiségű adatot is kezelni, tehát ez tűnt a legegyszerűbb megoldásnak, a kézi kezelésnek túl nagy méretű train adatok miatt. Az adatokat, melyeket felhasználtunk tsv fájlokban tároltuk, hatékonyan ki lehetett nyerni az adatokat a fájlokból, mert egy jó, és könnyen átlátható adatstruktúrát tartalmaztak a tsv fájlok. A neurális háló modelje a 3.1. ábrán látható:



3.1. ábra. Az osztályozó működése egy bemenetére érkező tensor esetén

3.1. Fájl struktúra

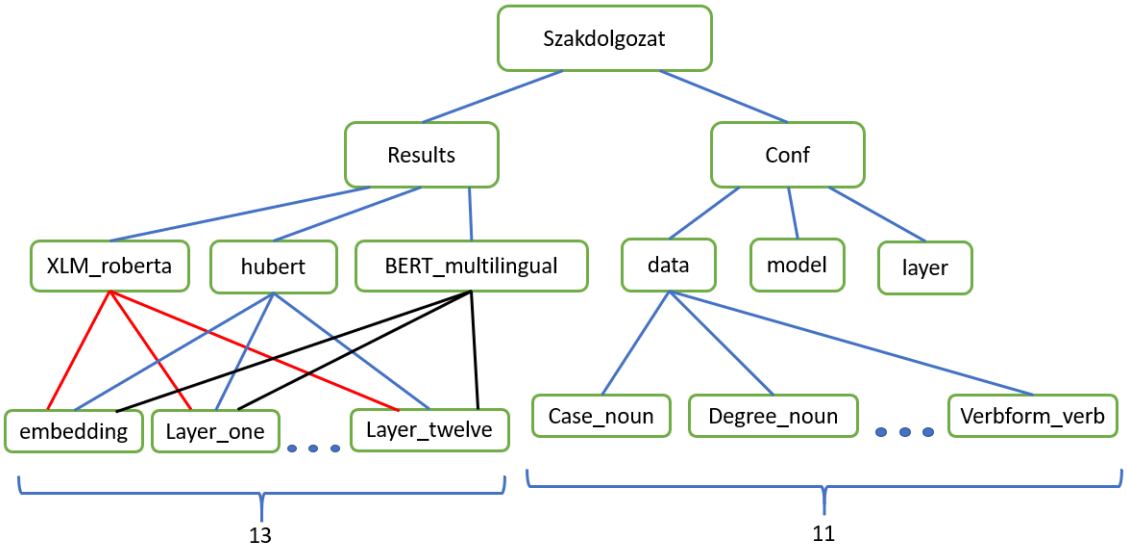
Az adatfájlokat tsv formátumban tárolva fel lehetett építeni egy adatstruktúrát, amelyet könnyű beolvasni és kezelni. Minden sora egy mondatot tartalmazott, melyet mint kontextust használtunk a célszavakkal benne. Minden egyes mondat egy-egy teszteset, melyeket a BERT modelen átengedve, a célszó utolsó szórészét kinyerve a neurális háló ezt osztá-

lyozva mutatja meg a BERT model teljesítményét. Ez a struktúrában, úgy jelent meg, hogy egy tsv fájl négy oszlopból áll. Egyikben maga a mondat, ami a kontextust adja, egy másik oszlopban található a szó, amit majd ki kell nyernünk, miután átfuttattuk a mondatot a BERT modelen. Ennek megtalálásához nyújt számunkra segítséget a harmadik oszlop, amely azt mutatja meg, hogy hányadik szó a mondatban az éppen keresett szó. Az utolsó oszlop a sorokban pedig, a szónak az éppen adatfájl típusától függő szófaja, egyes vagy többesszáma, vagy éppen fokozása, melynek a helyes megtalálása lesz a neurális háló feladata. Ezen utolsó oszlop különbözteti meg az egyes fájl csoportokat (amelyek részei a teszt, tanuló és dev adatfájlok), vagyis, hogy a szó milyen tulajdonságának megtalálására kell a neurális hálónak rájönnie a tanítása során. A pandas könyvtár segítségével DataFrame-ekben tároltam az adatokat, így elég volt a megadott oszlopnevekre hivatkozni és így egyszerűen el lehetett érni az éppen keresett adatokat.

A különböző adat csoportokat saját fájljaikban saját almappájukban tároltam, ezzel egyszerűbbé téve a beolvasásukat, amely hydra környezet által konfigurálva zajlott. Ehhez egy conf mappában egy data almappa alatt volt található minden adat csoport almappája. A program konfiguráláshoz elég mindössze egy +paraméterként megadni az adat csoport nevét, pl. degree_adj. Ezek az adatok az adatgeneráló script segítségével lettek létrehozva.

Az egyes modelek konfigurációját tartalmazó yaml típusú fájlokat, szintén a conf mappában levő, model almappában helyeztem el, amiatt, hogy ezek is könnyen konfigurálhatóak legyenek egy-egy futtatás során a hydra környezet segítségével. Egy másik későbbi konfiguráció volt a rétegek konfigurálása, hogy a BERT model mely rejtett rétegének kimenetét használja a neurális háló tanulásra a teszteseteknél. Ennek a konfigurációját is a hydra környezet segítségével oldottam meg az egyszerűség kedvéért, tehát az ezek konfigurációját tartalmazó fájlokat is egy erre létrehozott layer nevű almappában tároltam yaml fájlok formájában.

Az adott BERT model kimeneteivel mért eredmények a teszt adatokon kiértékelés céljából szintén elmentésre kerültek egy ennek megfelelő mappába. Egy results mappán belül minden vizsgált BERT modelnek létrehoztam egy almappát, amelyeken belül szintén konfigurációs megoldás miatt minden rétegnek van egy almappája, amelyben az adott model adott rétegen teljesített eredménye szerepel adat csoportonként különböző fájlokban. A projekt fontos fájljait tartalmazó mappák struktúrája a 3.2. ábrán látható:

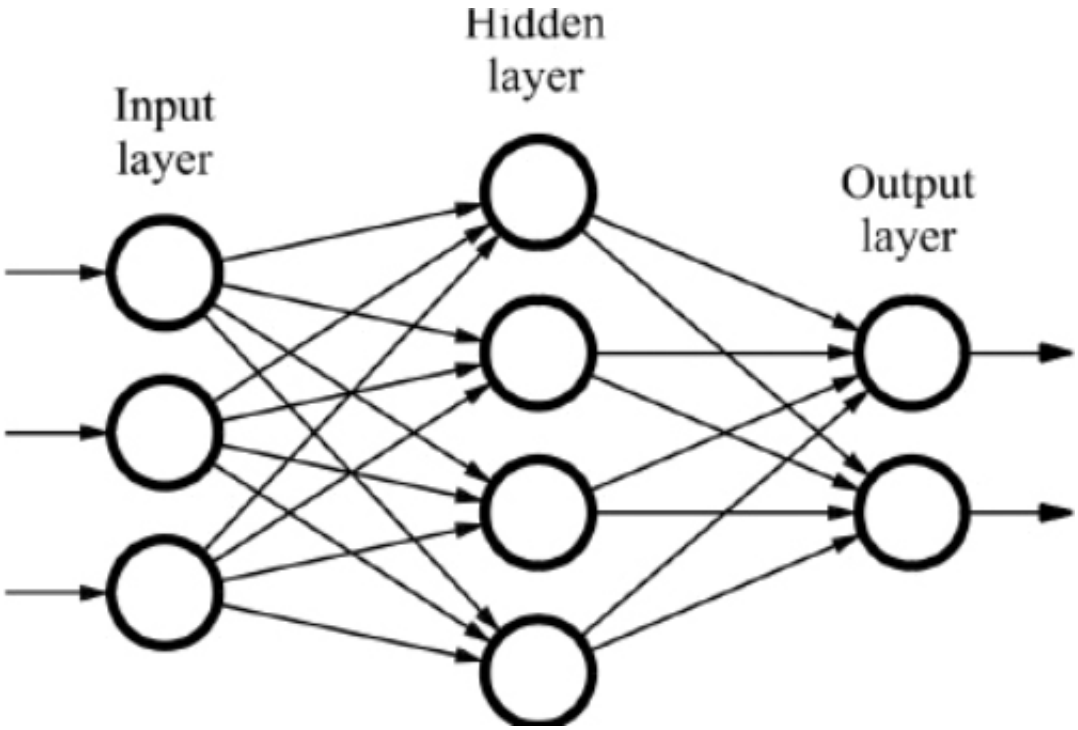


3.2. ábra. A fontos fájlok mappái

3.2. A program

A probingot és a kiértékelést végző program, amelyet a BERT modelek teljesítményének összehasonlítására használtam, egy egyszerű neurális hálóval végzi el a vizsgálatot. Az adatfájlok utolsó oszlopában szereplő osztályok visszaadása a cél az adott célszavakra. A neurális hálót tehát osztályozási feladatra kell betanítani a tanuló adatok segítségével, hogy a BERT által a célszó utolsó szórészének kódolt változatából meg tudja mondani, hogy melyik osztályba tartozik. Az osztályok értelmezése tartománya az adatfájlok utolsó oszlopa az adott adat csoportban. A programban a BERT model és a tokenizáló az AutoModel és az AutoTokenizer transformers könyvtárbeli elemekkel kerül példányosításra. Ez egy már betanított (pretrained) BERT modelt ad vissza. Tokenizálás során a célszó utolsó szórészének indexe a tokenizált mondatban mentésre kerül a program későbbi részének számára. Ezt a célszó szórész számának és első szórészének indexével valósítottam meg, mivel a célszó tokenizálás előtti indexe már rendelkezésre állt, a célszó szórészeinek száma pedig szintén egyszerűen kinyerhető tokenizálás után. A tokenizálás eredményét egy új DataFrame-ben tárolom el, amely a tokenizált szavakat és adataikat tartalmazza. Tehát a neurális háló a tokenizált mondatok célszavait kapja meg tokenizált, BERT modelen átengedett formában (vagyis a 768 méretű tensort a szó utolsó szórészéhez). Mivel az adatfájlok sorainak száma lehet meglehetősen magas szám is, ezért egy iterátor osztállyal futtatom végig az összes mondatot a probingot végző neurális háló tanításakor. Ennek az a lényege, hogy egyszerre a teljes adatmennyiségnek apróbb részét engedjük csak át a neurális hálón,

hogy tudja kezelni. Ennek megvalósításánál yield-et használtam, ezzel valósítottam meg a batch-elést. Maga a Classifier a torch.nn.Module-ból lett átemelve, lényegében egy szimpla neurális háló, bekalibrálható mennyiségű rejtett réteggel, bemeneti réteggel, és kimeneti réteggel. A bemeneti réteg mérete a BERT modelen átengedett mondatok száma, tanítás során a batch méret lesz. A rejtett réteg számnak megválasztása legtöbb esetben egy fontos optimalizálási szempont a neurális háló tanulási hatékonysága szempontjából. Figyelni kell, hogy ne adjunk meg se túl kis értéket, se túl nagy értéket a rejtett réteg számnak. Kis számú rejtett réteg esetén a neurális háló lehetséges, hogy nem fog elég jól tanulni, és nem éri el a kívánt pontosságokat tanulás során, az adott számú epoch alatt. Abban az esetben, ha túl sok rejtett réteget adunk meg a neurális hálónak, lehetséges, hogy túltanul a háló, vagyis a tanuló adatokat szinte memorizálja, tehát ezekre nagyon jól működik, de más, nem tanult adatok esetén a model jelentős mértékben kisebb pontosságot mutat. A batch mérete is egy fontos optimalizálási szempont, túl nagy esetén lassul a háló tanulási ideje, ha tud egyáltalán tanulni, túl alacsonynál pedig rengeteg iterációnk lesz, ami szintén idő amíg lefut. A kimeneti réteg mérete a címkék száma, tehát például egyesszám és többesszám eldöntésénél 2 a kimeneti réteg, de a melléknevek fokának eldöntésénél, már 3 lesz a kimeneti réteg mérete.



3.3. ábra. Egy tipikus neurális háló kinézete [1]

A model tanulása során az egyik fontos szerepet az optimalizáló és a hiba kiszámítására használt CrossEntropyLoss kapja. Ezeknek is sok változata van, én az Adam optimizert

használtam, mivel már korábban is bevált egy általam használt neurális hálóval, illetve a torch.nn könyvtárban található CrossEntropyLoss-t, mely a keretrendszerben szerepelve célszerű megoldás volt. Ezek segítségével az egyes iterációkban előforduló hibák mértékétől függően javíthatóak lesznek a haló súlyai. Ezzel kapcsolódik össze a CrossEntropyLoss, amelynek feladata kiszámítani, hogy a háló által mért valószínűségei az egyes osztályoknak milyen mértékben térnek el a válóságtól, ha a legvalószínűbbnek nem a megfelelő osztályt találta. Ezzel a funkcióval tehát már minden egyes iterációnál fogjuk tudni, hogy a neurális háló mennyire helyesen tippeli meg a címkéjét az adott szónak. A pontosság kiszámítását egy adott teszteset csoportra az alább látható 3.1. egyenlet szemlélteti, ahol az egyezés 1 és 0 értékek között mozog, akkor 1, ha megegyezik az osztálya a célszónak azzal, amit a program javasolt. A hossz az osztályozott adatokat tartalmazó tensorok mérete, tehát egy egyszerű átlagszámítással végezzük el a pontosság megmérését.

$$\text{accuracy} = \frac{\sum \text{egyenlőségek}}{\text{hossz}} \tag{3.1}$$

A hiba mértékét visszavezetve az egyes súlyokra backpropagation-el tudunk javítani az egyes súlyokon a hálóban, olyan mértékben, amilyen mértékben a veszteség ezt indokolja. Számon tartjuk a programban a tanuló ciklus során, hogy éppen mennyi a vesztesége a tanuló és dev adatokra, illetve mennyi a pontossága. A pontosságot 3.1. egyenlet alapján számolom ki itt is. A tanulás során az optimizer a hiba mértéke alapján javítja a neurális háló súlyait, egy-egy súly kis mértékben fog változni, hiszen sok súly adja ki a végeredményt. A számuk a rejtett réteg dimenziójától függ. Egyes lefutások úgynevezett epoch-ok végeztével kiírom az éppen aktuális lefutásra érvényes pontosságot és veszteséget, mind a tanuló adatok, mind a dev adatok esetén. Ennek segítségével később megvizsgálható, hogy mennyit javult az egyes epochok alatt, mikor érte el a maximumát és mikor tanult már túl a háló.

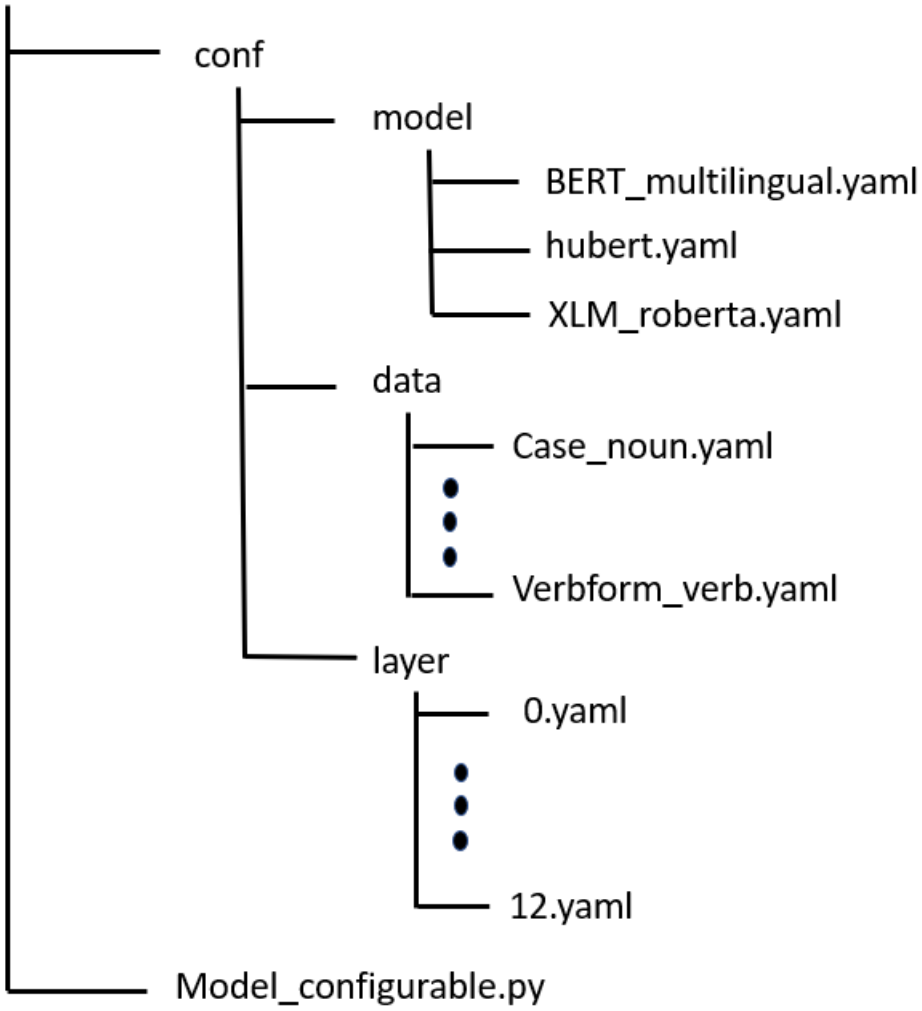
Ez a program lett konfigurálhatóvá téve, amiatt, hogy meg lehessen neki adni paraméterként az egyes BERT modeleket, amelyeknek teljesítményét megmérte a program, majd az eredményeket elmentette egy ennek megfelelő fájlban későbbi kiértékelés céljából. Ezáltal képet kapunk arról, hogy mely BERT model tudta a legjobban elmenteni a célszavak kontextusát annak utolsó szórészének a tensorába egyéb taskok megoldásának szempontjából. A konfigurálást a hydra környezet segítségével valósítottam meg, mivel ezzel egyszerűen lehet megvalósítani konfigurálható python fájlokat, és kevés előismerettel is használható. Tehát a program logikáját átvittem egy python fájlba, az eddig használt

szemléletes, de nem erre a célra kitalált jupyter notebookból teljes mértékben, és a hydra segítségével és .yaml kiterjesztésű fájlokkal be tudtam állítani, hogy a BERT model nevét paraméterként át tudja venni a python file, és az ennek megfelelő modelt tudja felhasználni a vizsgálat elvégzéséhez. Azonban nem csak a BERT model, hanem az adatfájlok is változhatnak, ezzel tágabb képet nyújtva az egyes BERT modelek teljesítményéről és pontosságáról. Szintén a hydra környezet segítségével valósítottam meg, hogy a python fájl futtatásánál paraméterként meg lehessen adni, melyik adat csoportot használja a program, mint train, dev és teszt adatokat. Ezek mind .tsv kiterjesztésű fájlok saját mappában adat csoportonként mely adat csoportok például melléknév fokszáma, főnevek egyesszáma vagy többesszáma stb. . Ennek a konfigurálható programnak a segítségével egyszerűbben lehetett összehasonlítani a BERT modeleket, és a teljesítményüket kiértékelni.

Később a konfiguráció kiterjesztésre került a BERT model rejtett rétegeire, a hydra által használt mappákat és fájlokat a 3.4. ábra foglalja össze:

Az előzőekben ismertetett program, a BERT model rejtett rétegei közül, pusztán az embedding réteget használja fel, ennek kimenetét továbbította a neurális háló felé tanulási célból. Ez azonban nem vezetett az elvárt eredményre, tekintve, hogy az embedding réteg lényegi információkat, amik alapján akár osztályba is lehetne sorolni a megfelelő szót, nem tud kinyerni az éppen számára vizsgált mondatokból és a célszavakból, hiszen nem ez a fő feladata. Ennek érdekében felmerült az igény arra, hogy megvizsgálhatóak legyenek a BERT model más-más rétegei által kinyert információk az eredeti szöveg kontextusából, tehát az a mechanizmus is beépítésre került, hogy az éppen konfigurációként megadott BERT model típusnak mely rétege legyen az, amelynek a kimenetét felhasználjuk a probing során. Ezen kimenetek már lényegi információt is tartalmaztak a célszó kontextusáról, és ez a későbbi eredményeken is meglátszott. Ezen konfiguráció hozzáadása után már egy egyszerűbben beállítható, mégis sok lehetőséget kínáló programmá vált az összehasonlító program. Ezután meg lehetett állapítani, hogy az egyes adat csoportokon melyik rétegek végeznek pontosabb információ kinyerést a kontextusból, amiből lehet következtetni akár arra is, hogy az adott réteg mely részére fókuszál jobban a kontextusnak.

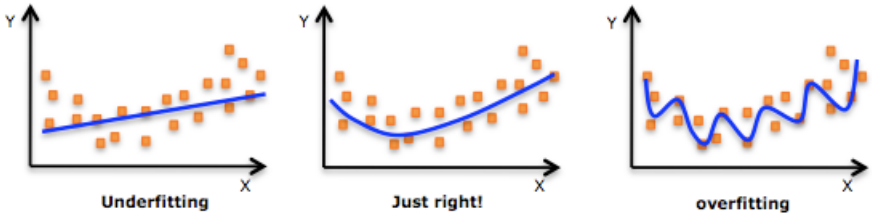
Igen lényeges része volt a program végső fázisai során, hogy ne engedjük a neurális hálónak meg, hogy túltanuljon a tanuló adatokra. Ez azt jelenti, hogy a neurális háló olyan módon változtatja meg a rejtett súlyait, hogy a lehető legpontosabban tudja visszaadni a tanuló adat minden egyes sorára, az ezen mondatnak a célszavára megfelelő osztályt. Ellenben mivel csak a tanuló adattal tanul a háló, az összes lehetséges adatot pedig lehetetlen neki megadni, tehát ha olyan adatot kapna, amilyent a tanuló adatfájl nem tartalmaz,



3.4. ábra. A program konfigurálásához szükséges fájlok és helyeik

akkor a háló ezen új adatok osztályozását sok hibával fogja tudni csak megtenni. Ezt elkerülendő alkalmazzuk a dev adatokat, melyekre nem tanul a háló, hogy epochonként megnézzük a pontosságát a hálónak ismeretlen adatokra. Ezek felhasználásával van ún. early stopping vagyis "korai megállás" megvalósítva a hálóban, melynek jelentése, hogy folyamatosan monitorozzuk a hibát, melyet a tanuló és dev adatokon vét a háló, és miután már többször csökken a dev adatokon vett pontosság, ez azért történik, mert a háló túltanul, ilyenkor meg kell szakítani a tanulást, tehát ha már több epochon keresztül sem javul a dev adatokon történt osztályozás pontossága leáll a tanulás, ezzel elkerülve a túltanulást. A túltanulás a 3.5. ábra szemlélteti:

Miután a konfiguráció előzetesen végrehajtott lépései megtörténtek, és a túltanulás is nagyrészt elkerülhető lett, az eddig nem használt teszt adatokat használtam a végső eredmény mérésére, amiatt, hogy mindenféleképpen elkerüljük azt, hogy ismert adatokat



3.5. ábra. Hogyan néz ki egy alultanulás és egy túltanulás [6]

adjunk be a neurális háló osztályozójának. Ennek érdekében a teszt adatokat, hasonlóan a tanuló és dev adatokhoz, szintén átengedtem az aktuális BERT modelen, és a mondatok célszavainak utolsó szórész tensorával leteszteltem az osztályozót, hogy mennyire pontos. Ennek a tesztnek az eredményét szintén további felhasználásra, az egyes epochok alatt kiírt dev és train pontosság mellett, egy fájlba mentette a program, illetve, hogy melyik adat csoportra vonatkozott, valamint, hogy melyik rétegét használtuk a BERT modelnek a teszt során.

3.3. Futtató script

Annak érdekében, hogy az összes modelről adatokat lehessen gyűjteni, egy külön programot írtam, amely lefuttatta az előbb bemutatott programot egy BERT modelen, minden réteg és minden adat csoport felhasználásával. Ezeknek a futtatásoknak, amely során az adott BERT modelt az adat csoport minden mondatának kódolása során felhasználja a program, az volt a lényege, hogy minden rétegnek minden adat csoporthoz lehessen eredményt rendelni, amelyeket kiértékelés céljából elmentsen a program. Ezeket az eredményeket a nekik megfelelő fájlba menti.

Kezdetben egy egysoros program volt mindössze annyit tett, hogy egy ciklus segítségével lefuttatta a programot tartalmazó python fájlt minden egyes adat csoport felhasználásával egyesével, a BERT model a konfigurációban pedig a BERT multilingual cased model volt. A konfigurációban szereplő réteg az első, már nem embedding réteg volt, hiszen ekkor már elvárhattunk egy jobb pontosságot a neurális hálótól, ez a réteg ugyanis már valódi kontextuális információt is ki tud nyerni. A ciklus egyes lépéseiben az egyes adat csoportokat változtatta a script, tehát végül minden egyes adat csoporton, minden osztályozási szempont alapján kiértékelésre került a BERT multilingual model az osztályozó neurális háló segítségével.

Ez a script is később konfigurálhatóvá lett alakítva, meg lehetett adni, hogy melyik BERT model típuson futtassa a három közül a python fájlban szereplő programot az első nem embedding rétegen minden adatcsoporton. A fájlba mentés is ezzel konfigurálható lett, hiszen különböző BERT modelek eredményeit célszerű volt különböző mappákban elhelyezni. Ezt a problémát a program oly módon küszöböli ki, hogy minden egyes BERT model opcióhoz az eredmények mappában létre lett hozva egy almappa, amelyben az ennek a BERT modelnek a különböző adat csoportokon elért eredményei lettek elmentve.

A végső lépés az egyes rétegek bevonása volt a program működésébe. Ennek érdekében kibővítettem a ciklusokat, manuálisan, hogy a paraméterként megadott BERT modelt minden egyes adat csoportra, minden egyes rétegre tesztelje le a python program segítségével. Mivel a python program a rétegekre is konfigurálhatóvá vált, így annyi változtatást kellett csak tenni, hogy az egyes BERT model mappákon belül minden rétegnek legyen egy almappája, ahol az egyes adat csoportokon elért eredményeit lehessen tárolni.

Annak az ideje, amíg egy BERT modelen átfutott egy bemeneti fájl, hogy végeredményként megkapjuk a BERT model által visszaadott szöveg reprezentációt, az összehasonlító program többi részéhez képest igen nagy. Az volt idő szempontjából logikus, hogy a scriptet több részre törve párhuzamosan lehessen futtatni a kiértékeléseket. A tmux és egy távoli GPU szerver segítségével ehhez egy eszköz is elég volt. A fájlrendszerből ezután már ki lehetett nyerni a kapott eredményeket a teszt adatokra és az egyes epochokra.

4. fejezet

Az összehasonlítás

Ebben a fejezetben a BERT modeleket és ezek teljesítményét hasonlítom össze, annak megállapítása céljából, hogy melyiknek sikerült a legpontosabb eredményeket elérnie, mely rétegekkel és adat csoportokkal. Az is szempont volt, hogy mely adat csoportok mely rétegek számára egyszerűen megoldhatóak és melyek kevésbé.

4.1. Korai tapasztalatok

A neurális hálók egyik fontos tulajdonsága, hogy memorizálni tudják akár a teljesen véletlenszerű adatokat is megfelelő súlyok segítségével, ez azonban nem jelenti, hogy azok a modelek később használhatóak lennének bármilyen feladat pontos és hatékony megoldására. Az embedding réteg eredményei által való tanítása az osztályozó neurális hálónak hasonlóképpen történt. A tanuló adatokon való pontosság ugyan javult, de a dev adatokon ugyanez az elvárt pontosságtól jóval alulmaradva elkezdett csökkenni, vagyis már a túltanulás fázisba lépett a háló. Az elvárt eredmények, melyek 90% vagy afeletti értékek lettek volna, az adat csoport jellegétől függően (kimeneti réteg mérete). Ez leginkább a degree_adj esetében volt szembetűnő, hiszen ennél az adat csoportnál 3 osztály közül kerülhet ki a célszónak az osztálya, a neurális háló pedig pusztán 68%-os pontosságot ért el, ezzel jelentősen alulmúlva az elvárt értéket, amelyet szerettünk volna elérni a tanítással.

Ennek oka az volt, hogy az embedding réteg által visszaadott eredmény, mint a neurális háló bemenete kevésbé releváns információt tárolt a kontextusról, tekintve, hogy fő célja ennek a rétegnek pusztán annyi, hogy egységes méretűre hozza az adatokat további feldolgozás céljából. Ennek eredményeként szinte semmilyen kontextuális információ nem fog megjelenni a BERT model kimenetén ebből a rétegből, ezt támasztotta alá a neurális háló eredménye is az egyes adat csoportokra.

Ezután más rétegek kipróbálásával sikerül elérni olyan eredményeket, melyek közelítettek az elvártakhoz, azonban megfigyeltem, hogy vannak olyan adat csoportok, amelyeknek osztályozási feladatát egyes rétegek "jobban megoldották", tehát az általuk kinyert információ az adott adat csoportnál jobb teljesítmény eléréséhez vezetett a neurális háló osztályozása során. Ebből azt a következtetést vontam le, hogy bizonyos rétegek más-más aspektusait mentették el a kontextusnak a célszó körül annak utolsó szórészébe. Ezzel bizonyos adat csoportok osztályozása során az eredményük kifejezetten előnyösen alkalmazható, míg más adat csoportok esetén akár az embedding réteg pontosságát érik csak el.

További megfigyelésem volt, hogy az alkalmazott torch.nograd függvény segítségével lefuttatott BERT modelek voltak a leglassabb komponensei az összehasonlítást végző programnak. Ezeket azonban elég volt egyszer elvégezni adatcsoportonként és futtatásonként. Ennek kiküszöböléseként párhuzamosan futtattam a programot különböző konfigurációkkal az eredmény gyorsabb kinyerésére.

4.2. SZTAKI Hubert, BERT multilingual

Korábbi tapasztalatok a BERT multilingual esetében, melyet a github oldalán is kifejtenek több esetben, hogy egy multilingual model általában nem lesz annyira pontos, mint egy model, amit konkrétan arra a nyelvre terveztek, hiszen amit csak egy nyelvre készítettek, annak a tanítása is csak azon az egy nyelven fog történni, több, gyakran előforduló szórész fog például a tokenizálás során bekerülni a "szótárba", amiből a tokenizáló utána kiválogatja az éppen megfelelőket, ezzel több fontos igekötőt és toldalékot nagyobb eséllyel fog egyben tartani a tokenizáló. Ezzel pedig több fontos információ kerül kimentésre a kontextusból. A 4.1. ábra szemlélteti, hogy egy multilingual és egy egynyelvű model teljesítménye mennyiben tér el.

System	Chinese
XNLI Baseline	67.0
BERT Multilingual Model	74.2
BERT Chinese-only Model	77.2

4.1. ábra. A tisztán kínai nyelvre tanított BERT a multilinguallal szember [4]

Ebből kiindulva az előzetes elvárás az volt, hogy a Hubert [5] valamivel jobban fog teljesíteni, mint a BERT multilingual. Tekintve, hogy egy magyar nyelven tanított (Common Crawl web archív és a magyar Wikipédia adataiból) szöveg reprezentációs model, szintén Transformers alapokra épül, multi head attentiont alkalmazva. Ennek segítségével távolabbi kapcsolatokat is képes észlelni, mint egy egyszerű RNN tudna, a [7] cikkben kifejtik a pontos működést mélyebben.

A magyar BERT segítségével végzett vizsgálatok alapján a fenti feltevés látszik igazolódni, az elvárt pontosságokat tudta hozni a neurális háló, ha ezen BERT model kimenetét kapta meg, mint bemenetek. Ez azt sugallta, hogy a magyar BERT jól ki tudta nyerni az információt a kontextusból az egyes adat csoportoknál.

A következő oldalon látható ennek számszerűsített változata, amikor a magyar BERT model segítségével az egyes adat csoportok minden változatával kipróbáltuk az osztályozást. Az egyes adatcsoportok tanuló adatait, dev adatait és teszt adatait is átengedtük a HuBert-en. Majd a BERT model kimenetéből az egyes mondatok adott célszavainak utolsó szórészének kódolt verzióját kivéve, melyet az éppen adott futásban vizsgált réteg rejtett súlyai állítottak elő. Ezen szórészek felhasználásával, mint bemenetek lett megtanítva a neurális háló, amely a tanuló adatokon tanult, és a dev adatokkal a túltanulást elkerülve ún. early stoppinggal állt le. Végül pedig a teszt adatok segítségével megnézve a teljesítményt, a 4.1. táblázat egyes soraiban és oszlopaiban láthatjuk vizsgált réteg és vizsgált adat csoport függvényében.

A legjobban teljesítő adat csoportok voltak a verbform verb, a második a number verb és a harmadik a number noun.

A legjobban teljesítő rétegek a 6 és a 11 voltak.

Ugyanezen teszteket elvégezve a BERT multilingual BERT model segítségével az eredményeket a 4.2. táblázat mutatja be.

A Hubert által produkált eredményeket összehasonlítva a fentebb látható BERT multilingual model használatával kapott eredményekkel láthatóak azok a különbségek, amikre számítottunk, hiszen több adat csoport esetén is, a rétegek nagyobbik része jobban teljesített a Hubert modellel, mint amilyen pontosságot sikerült elérni a BERT multilingual rétegei által. Ez leginkább a "mood verb" adat csoport során tűnik ki, amelynél a BERT multilingual által végzett szövegfeldolgozás nem volt elég jó még a 90% pontosság elérésére sem egyik réteg esetében sem.

Mindkét model által kinyert információk tesztelésének eredményeit figyelve, azt vettem észre, hogy mind a kettő BERT model típus a "verbform verb" adat csoport esetén

HuBert	case noun	degree adj	mood verb	number adj	number noun
embedding	0.9491	0.6866	0.845	0.95	0.865
Layer 1	0.9722	0.9303	0.985	0.985	0.975
Layer 2	0.9769	0.9701	0.95	0.995	0.97
Layer 3	0.9769	0.9602	0.955	0.995	0.98
Layer 4	0.9769	0.9652	0.945	0.995	0.97
Layer 5	0.9769	0.9602	0.945	0.995	0.98
Layer 6	0.9769	0.9602	0.945	0.99	0.99
Layer 7	0.9769	0.9502	0.945	0.99	0.98
Layer 8	0.9769	0.9453	0.94	0.99	0.98
Layer 9	0.9769	0.9502	0.95	0.985	0.98
Layer 10	0.9722	0.9353	0.945	0.995	0.98
Layer 11	0.9722	0.9453	0.945	0.975	0.985
Layer 12	0.9676	0.9502	0.945	0.975	0.98
HuBert	number verb	person psor noun	person verb	tense verb	verbform verb
embedding	0.955	0.8607	0.9602	0.905	0.95
Layer 1	0.965	0.904	0.9801	0.97	0.98
Layer 2	0.98	0.9353	0.9701	0.98	0.985
Layer 3	0.995	0.9303	0.9751	0.975	0.985
Layer 4	0.99	0.9104	0.9701	0.97	0.995
Layer 5	0.995	0.9154	0.9751	0.975	1
Layer 6	0.985	0.9104	0.9652	0.975	1
Layer 7	0.97	0.9303	0.9652	0.97	1
Layer 8	0.975	0.9055	0.9652	0.975	1
Layer 9	0.97	0.8905	0.9602	0.97	1
Layer 10	0.98	0.9154	0.9303	0.98	1
Layer 11	0.985	0.9403	0.9353	0.985	1
Layer 12	0.98	0.9453	0.9502	0.98	1

4.1. táblázat. HuBert által elért eredmények

BERT	case noun	degree adj	mood verb	number adj	number noun
embedding	0.9167	0.7164	0.635	0.945	0.84
layer 1	0.9398	0.7662	0.835	0.965	0.885
layer 2	0.9491	0.8209	0.85	0.975	0.895
layer 3	0.9583	0.9055	0.88	0.975	0.925
layer 4	0.9583	0.8806	0.865	0.97	0.925
layer 5	0.9583	0.9204	0.835	0.97	0.935
layer 6	0.9583	0.9403	0.845	0.97	0.95
layer 7	0.963	0.9353	0.825	0.97	0.955
layer 8	0.963	0.9303	0.75	0.97	0.945
layer 9	0.963	0.9254	0.76	0.98	0.945
layer 10	0.9537	0.9104	0.78	0.98	0.955
layer 11	0.9491	0.9204	0.775	0.98	0.945
layer 12	0.9444	0.8955	0.82	0.985	0.94
BERT	number verb	person psor noun	person verb	tense verb	verbform verb
embedding	0.925	0.7164	0.9055	0.825	0.9
layer 1	0.93	0.8955	0.9403	0.925	0.965
layer 2	0.93	0.9005	0.9353	0.93	0.965
layer 3	0.935	0.9104	0.9254	0.94	0.955
layer 4	0.95	0.9254	0.9403	0.92	0.96
layer 5	0.95	0.9005	0.9353	0.905	0.965
layer 6	0.945	0.9104	0.9303	0.91	0.97
layer 7	0.955	0.9204	0.9154	0.935	0.975
layer 8	0.945	0.9005	0.9055	0.9	0.98
layer 9	0.955	0.8905	0.9403	0.89	0.97
layer 10	0.945	0.9756	0.9303	0.93	0.97
layer 11	0.945	0.8856	0.9204	0.905	0.98
layer 12	0.95	0.8557	0.8905	0.89	0.97

4.2. táblázat. Bert által elért eredmények

tudta a neurális hálót a lehető legtöbb kontextuális információval ellátni annak érdekében, hogy az osztályozás helyesen zajlódjon le. Ekkor a legtöbb tesztesetre sikeresen tudta beosztani a célszót a neki megfelelő osztályba. Ezek a Finite és Infinite osztályok, amely magyarul nagyjából a határozott és általános ragozásra fordíthatóak le, ha mindenképpen szeretnénk lefordítani.

Az egyik lehetséges indok, hogy miért ezen adat csoport adatainak segítségével tanítva a neurális hálót, érte el a neurális háló a legjobb, legpontosabb eredményeket az osztályozás tekintetében az, hogy mindössze ezen kettő opció közül kellett választania. A random választás is már 50% pontosságot adna, gyorsabban látható javulás a tanulás során, ellentétben például egy 3 opció közüli választást tartalmazó adat csoport esetén.

Egy másik magyarázat lehet ezeknek az eredményeknek a fennállására a magyar nyelv sajátosságaiban és az összehasonlítás során végzett pontos lépésekben keresendő. Tekintve, hogy ahhoz, hogy egy igéről lehessen dönteni, hogy a 2 opció közül melyik osztályba sorolható leginkább az adott célszó, elegendő a célszó utolsó szórésze, abban az esetben, ha a BERT model úgy osztotta részekre a célszót és a szöveget, hogy az utolsó szórésze a célszónak, tartalmazza a toldalékot. Ez az utolsó rag pedig a legárulkodóbb annak eldöntésére, hogy Finite vagy Infinite az ige, ha a kontextusát is jól elmenti a BERT. Mivel a Hubertnél nagyobb a mérete a szótárának, ezért itt nagyobb eséllyel fog a teljes toldalék az ige végén bekerülni egy önálló szórészbe, ezzel magyarázható a nagy pontosság, míg a BERT multilingual esetén nem minden esetben kerül be egy önálló szórészbe, ezáltal kevesebb információ jut el a neurális hálóig a pontos kontextusról.

4.3. BERT multilingual, XLM-Roberta-val

Annak érdekében, hogy többet megtudjunk arról mennyire jó teljesítőképességekkel rendelkezik a EBRT model többnyelvű változata, nem elég egy előreláthatóan jobb modellel összevetni, amely mivel pusztán magyar nyelvre készült, jobb eredményeket képes produkálni, mint egy alapértelmezetten multilingual model. Ennek érdekében a következő logikus lépés az volt, hogy összehasonlítsam a BERT multilingual model teljesítőképességét és tulajdonságait egy másik, szintén több nyelv elfogadására, és feldolgozására tervezett model teljesítményével, amely jelen esetben az XLM-RoBerta [2] nevű többnyelvű BERT model volt.

Elöljáróban megemlíteném, hogy a két model egyik különbsége a tanulóadat, melyeken őket tanították, ugyanis ezen adattól is, a témáktól is függenek az egyes modelek szótárai,

XLM roberta	case noun	degree adj	mood Verb	number adj	number noun
embedding	0.9537	0.6418	0.8700	0.9500	0.8950
layer 1	0.9491	0.7363	0.8550	0.9450	0.8950
layer 2	0.9630	0.8060	0.9250	0.9500	0.9300
layer 3	0.9630	0.8856	0.9150	0.9700	0.9350
layer 4	0.9630	0.9602	0.9000	0.9650	0.9550
layer 5	0.9630	0.9353	0.9150	0.9650	0.9550
layer 6	0.9676	0.9652	0.9200	0.9650	0.9700
layer 7	0.9676	0.9552	0.9250	0.9650	0.9600
layer 8	0.9630	0.9701	0.9300	0.9700	0.9700
layer 9	0.9630	0.9652	0.9250	0.9750	0.9850
layer 10	0.9630	0.9502	0.9300	0.9750	0.9900
layer 11	0.9583	0.9502	0.9200	0.9750	0.9850
layer 12	0.9583	0.9602	0.9350	0.9750	0.9800
XLM roberta	number verb	person psor noun	person verb	tense verb	verbform verb
embedding	0.9500	0.8259	0.9552	0.9650	0.9500
layer 1	0.9450	0.8159	0.9453	0.9300	0.9550
layer 2	0.9500	0.9204	0.9502	0.9650	0.9700
layer 3	0.9600	0.9353	0.9602	0.9750	0.9750
layer 4	0.9600	0.9204	0.9652	0.9700	0.9700
layer 5	0.9700	0.9055	0.9701	0.9800	0.9800
layer 6	0.9650	0.8955	0.9751	0.9650	0.9850
layer 7	0.9750	0.8905	0.9751	0.9750	0.9900
layer 8	0.9650	0.9005	0.9701	0.9750	0.9950
layer 9	0.9700	0.8706	0.9652	0.9800	0.9950
layer 10	0.9700	0.8756	0.9652	0.9800	0.9950
layer 11	0.9600	0.8856	0.9801	0.9750	0.9900
layer 12	0.9800	0.8905	0.9751	0.9700	0.9950

4.3. táblázat. RoBerta által elért eredmények

illetve rejtett súlyai, amikkel dolgozik. A BERT multilingual a már említett Wikipédiákon lett betanítva, a 100 legnagyobb adatbázissal rendelkező, különböző nyelvű Wikipédiákon tanították ki a BERT egyes súlyait és szótárát, hogy mely szórészek fordulnak elő a legnagyobb valószínűséggel egy adott szövegben. Ezzel szemben az XLM RoBerta tanulóadatai a Common Crawl adatbázisból kerültek ki a tanult nyelvekre. Ez az adatbázis az interneten gyűjtött adatokból található, melyeket 2011 óta folyamatosan gyűjtenek és a teljes mérete petabyte-okban mérhető már. Maga a RoBerta 2.5 TB-nyi adaton tanult ebből, és ennek az adatnak a segítségével tanulta meg a szótárát, szintén a legvalószínűbb szórészekkel, illetve a rejtett súlyait.

A 4.3. táblázat azt ábrázolja, hogy az XLM-RoBerta [2] felhasználásával, mennyire pontosan tudta az ezáltal visszaadott kimenetet megtanulni a neurális háló, hogy tanítás

után ennek a hálónak az addig érintetlen teszt adatokon. Akárcsak az eddigi esetekben, más-más rétegek más-más adat csoportok esetén más-más pontosságot értek el.

Összehasonlítva a BERT multilingual eredményeivel az XLM-RoBerta által visszaadott tensorok segítségével elért eredményeket, itt is az egyik legkevésbé pontosan megtanult adat csoport a "mood verb", amely esetén a BERT multilingual bizonyos rétegei annyira alacsony teljesítményt értek el, hogy a neurális háló nem tudott egyes rétegek esetén még 80%-os pontosságot sem elérni. A RoBerta ennél már jobban teljesített ezen adat csoport esetén, mint korábban a BERT többnyelvű változata, de ez a BERT model sem tudta 95% fölé juttatni a neurális háló pontosságát.

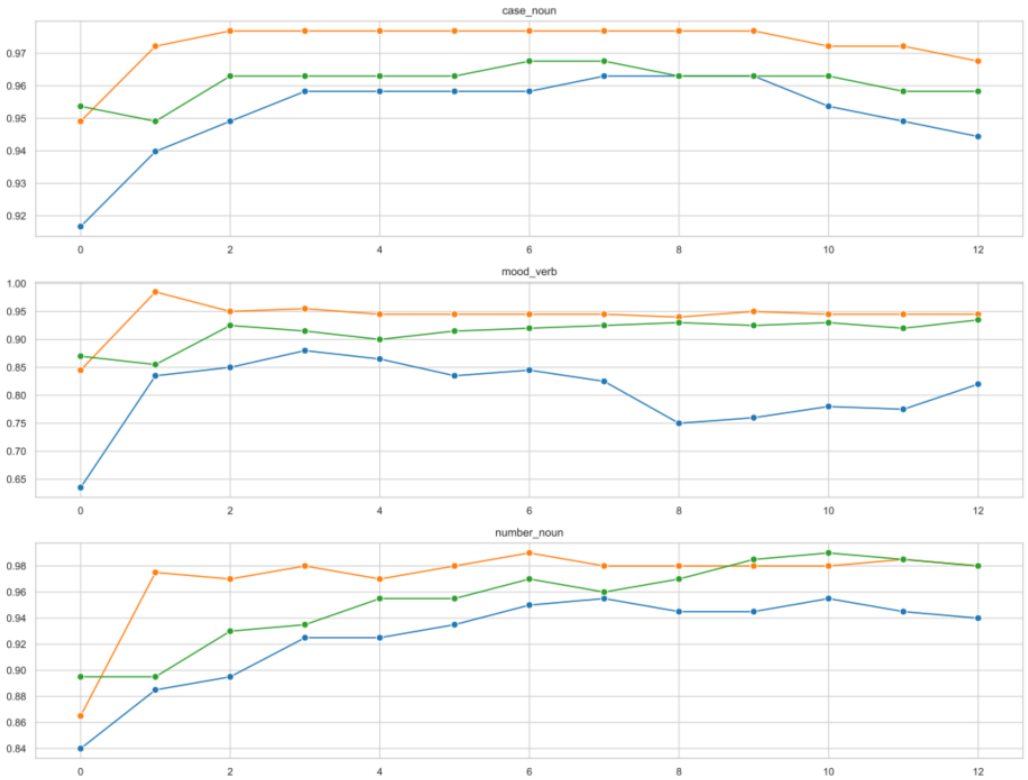
Emellett egy másik adat csoport melyet elég nehezen tudott kezelni mindkét model az a "person psor noun", amelynek lényege, hogy az adott főnevet birtokló személy milyen személy, mely 3 lehet (1,2,3), ezt lassabb megtanulni valamennyivel, hiszen a random osztályozó itt már csak 33%-al tudna teljesíteni pontosság terén. Ezt alátámasztandó, a melléknevek fokának megállapítása során ("degree adj") az első néhány réteg esetében még nem sikerül elég kontextuális információt kinyerni ennek eldöntésére, és az elvárt pontosságot hozniuk, csak a további rétegek tudják ezt a feladatot megoldani. Annak ellenére, hogy az utolsó szórészben lévő kontextuális információ ebben az esetben nagy segítség, tekintve, hogy a 'bb' végződés már csak két lehetőséget hagy a háromból osztályozás során, feltéve, hogy a 'bb' toldalék benne van a szótárában a BERT modeleknek.

A legjobb adatcsoport, melyet mindkét model jól tud lekezelni ismét csak a "verbform verb" adat csoport, magyarban ez nagyjából a határozott és általános ragozású igék esetét jelenti, ha mindenképp le akarnánk fordítani. Ezen modelek hamar jó teljesítménnyel tudtak átalakítani a neurális háló tanulása számára, aminek lehetséges okait a BERT multilingual és a HuBert eredményeinek összehasonlítása során már kifejtettem.

4.4. Adat csoportok

Az egyes adat csoportok tanulása, és kiértékelése más-más pontossággal történt, hiszen más-más méretű a kimeneti dimenzió, az osztályok száma. Az elért pontosság mértéke függ továbbá a használt BERT modeltől és a használt rétegtől is, amelynek tensorát felhasználjuk. Ezek közös összehasonlítására a legegyszerűbb ábrázolni a kapott eredményeket grafikonokon, amelyeket adat csoportonként készítünk. A jelmagyarázat a 4.2. ábrán és 3 az adat csoportok közül a 4.3. ábrán látható

4.2. ábra. Jelmagyarázat a grafikonokhoz

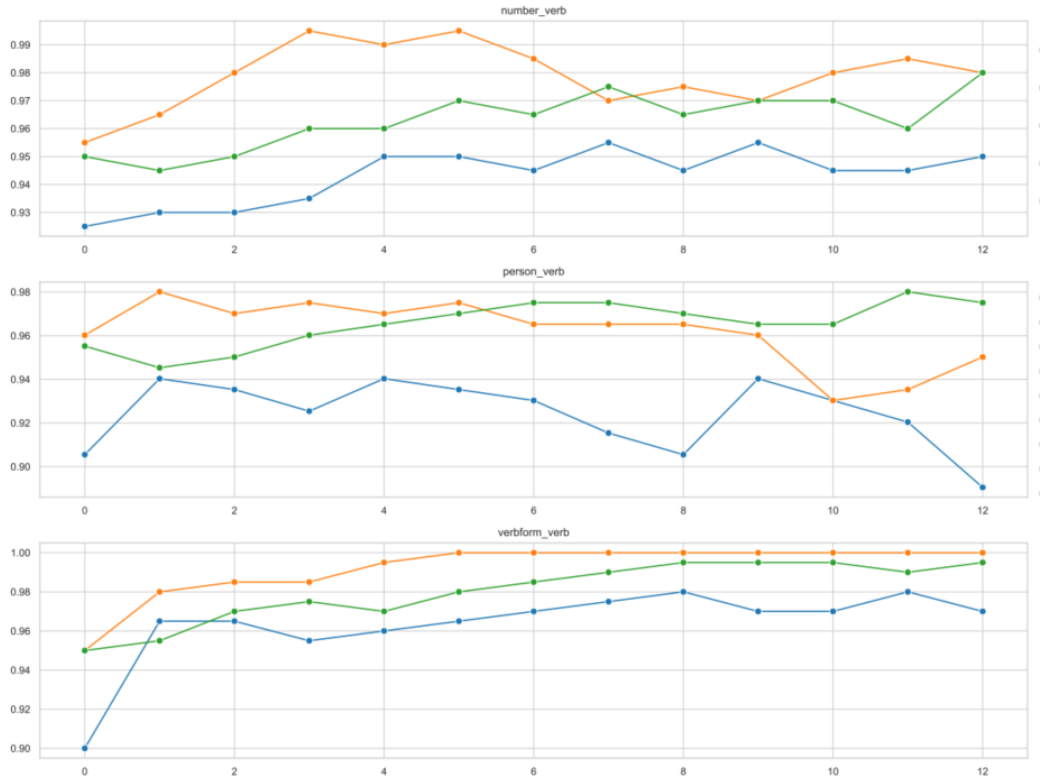


4.3. ábra. case_noun, mood_verb, number_noun eredményei egyes modelekkel és rétegekkel

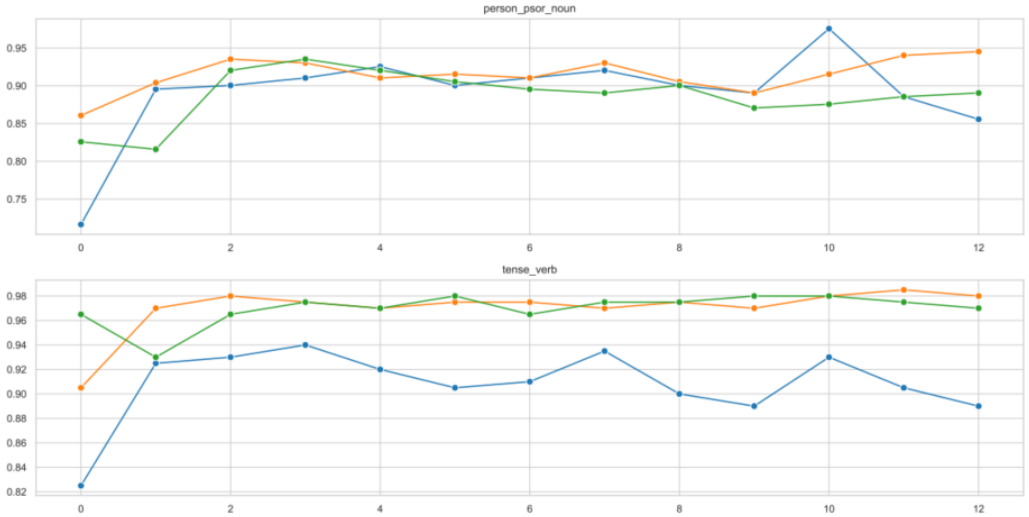
Az ábrák vízszintes tengelyén helyezkednek el az egyes rétegek, a függőleges tengelyes pedig a pontosságok. Egy-egy pont egy adatnak fele meg, ami a mérés eredménye volt a teszt adatokon, adott adat csoporton adott réteggel, adott BERT model segítségével. A 0. réteg az embedding réteg, amely nem tárolt lényegi kontextuális információkat, így ez lesz a legkevésbé pontos eredményekkel rendelkező réteg.

A 4.1., 4.2., és a 4.3. táblázatokban szereplő eredmények alapján levont következtetések itt is igazolódni látszanak. A HuBert a legpontosabb az egyes esetekben, a RoBerta a második, a BERT multilingual pedig a harmadik. Megfigyelhető az is, hogy az adat csoportoktól függ, hogy éppen melyik réteg teljesít jobban a többinél, illetve elérik-e az elvárt pontosságot, kb. 90%-ot, vagy sem.

A további adat csoportok eredményei a 4.4., 4.5. és a 4.6. ábrákon láthatóak.



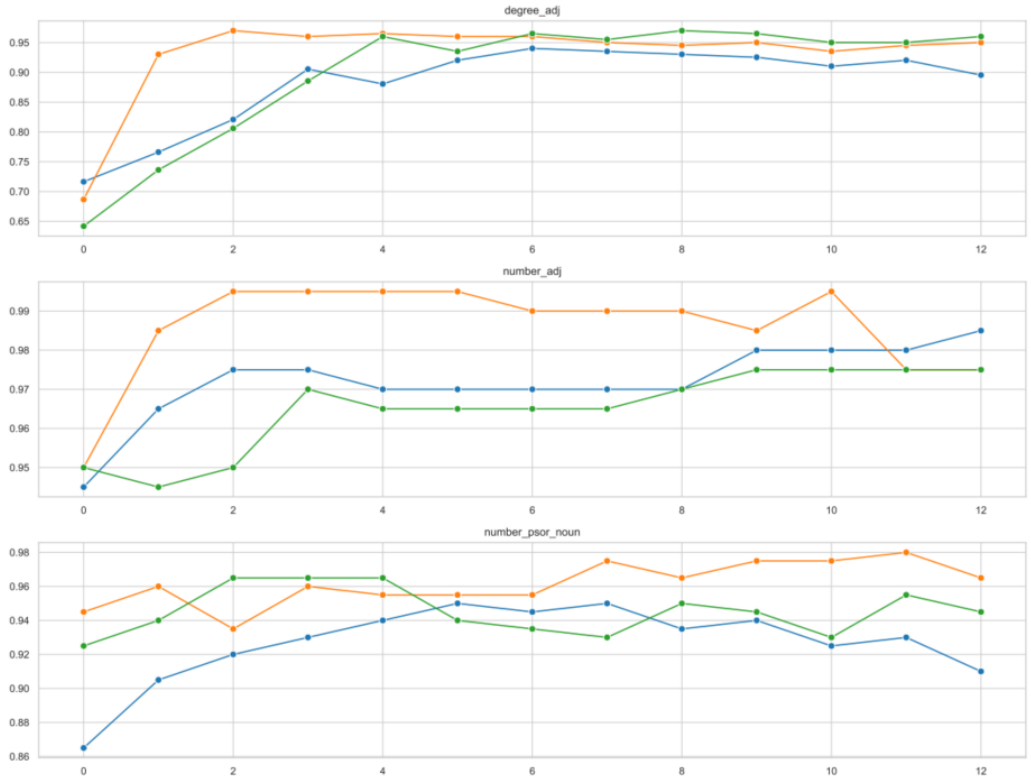
4.4. ábra. number_verb, person_verb, verbform_verb eredményei egyes modelekkel és rétegekkel



4.5. ábra. person_psor_noun, tense_verb, verbform_verb eredményei egyes modelekkel és rétegekkel

A további adat csoportok eredményeit vizsgálva megfigyeltem, hogy egyes rétegek akár az embedding rétegnél is rosszabbul tudnak teljesíteni. Ennek oka lehet, hogy az adott réteg az adott feladat szempontjából nem tárol elég releváns információt a célszó utolsó tensorába, míg más rétegek igen.

A Hubert, RoBerta, BERT multilingual sorrend továbbra is fennáll ilyen sorrendben.



4.6. ábra. degree_adj, number_adj, number_psor _noun eredményei egyes modelekkel és rétegekkel

A korábban megfigyelt "mood verb" mellett, a BERT multilingual több másik adat csoporton is gyenge teljesítményt produkál.

4.5. Hibák kiértékelése

A pontatlanságok lehetséges okainak kiderítése során elkerülhetetlen megvizsgálni a hibás eseteket és kiértékelni ezeket. Ennek érdekében kigyűjtöttem a tesztesetek közül a teszt adatokból azokat, amelyekre gyengén teljesítettek az egyes modelek, és ezen tesztesetek alapján létrehoztam kategóriákat, amelyek a hibák lehetséges okait reprezentálják. Az egyszerű áttekinthetőség és az online bárhonnan való elérés érdekében Google Spreadsheetet alkalmaztam a feladat megoldása során.

A tesztesetek mondatai, célszavai, valamint az elvárt és a kapott osztályok segítségével végeztem a kategorizálást, hogy mi okozhatta a neurális háló rossz osztályozását. A teszt adatok 200 mondatból álltak, tehát a kevésbé pontos esetekben volt 10-15 hibás eset. Tehát azon adat csoportok, modelek és rétegek kombinációjánál volt csak érdemes megtenni a kiértékelést, ahol alacsony pontosságot ért el a háló. Ilyen volt az XLM-RoBerta és a BERT multilingual esetén a "mood verb" adat csoport. Utóbbinál bármely réteg megfelelt,

hiszen sok félreosztályozás történt vele, az XLM-RoBerta esetén az első néhány rétegből választottam egyet a vizsgálatra, ahol 90% vagy akörüli pontosságot ért el a háló.

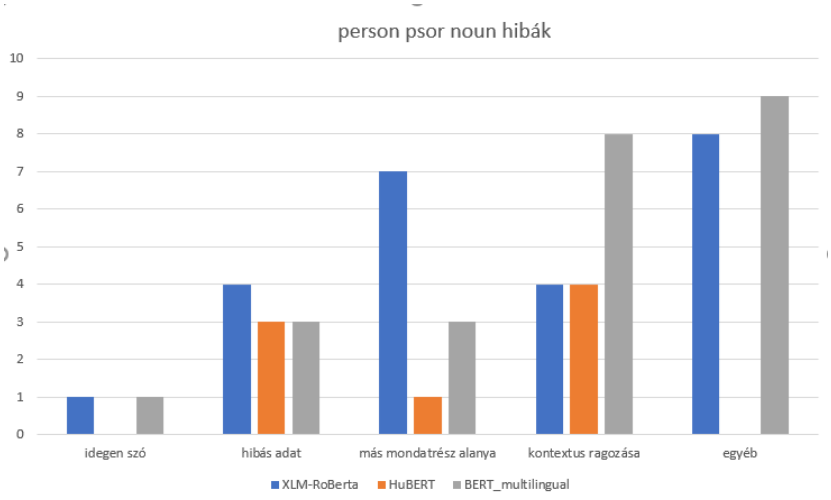
A HuBert esetén a nagy pontosság miatt kevés lehetőség adódott a hibák mélyebb kielemzésére, a "person psor noun" adat csoportot vizsgáltam meg az első rétege szerint, ahol kb. 90% volt a pontossága. Ezt az adat csoportot a többi model egy-egy rétegével is megvizsgáltam, hogy össze tudjam őket hasonlítani, hogy mik a tipikus hibái az egyes BERT modeleknek.

person psor noun: Ebben az adat csoportban a célszó birtokos személyét kellett eltalálni (1,2,3) a toldalék a legtöbb esetben árulkodó volt. Néhány teszteset folyamán mind a három BERT model hibásan vette ki az információt a mondatból, azonban előfordult olyan is, mikor az adat hibássága miatt nem sikerült a neurális hálónak eltalálnia az osztályt egyik model esetén sem. Észleltem viszont a BERT multilingualnél azt, hogy a hibás adat ellenére, a szó végződése hasonló volt egy toldalékhoz, ezért az osztályozó eltalálta a "helyes" osztályt, de fordítva is volt rá példa. A HuBert model [5] használatával ejtett hibák fele az adat hibássága miatt következett be, egy másik tipikus hibaforrása volt az, amikor a kontextus legtöbb többi szavának ragozása nem egyezik meg a célszóéval, így a célszó ragozását félreosztályozta. A BERT multilingual [4] használatával a hibás adatokon kívül leginkább akkor hibázott az ozsztályozás, amikor a mondat egy másik mondatrészében más alany szerepelt, mint a keresett célszó birtokosa, ekkor ugyanis, az adott alany személye lett a visszakapott osztály. A magas hibaszáma miatt, itt több hibaesetet nem lehetett besorolni egy konkrét kategóriába sem, ezeket egy "egyéb" kategóriaként kezeltem. Az XLM-RoBerta [2] használatával az előbb említett hiba kategóriákon kívül előfordult, hogy a kontextus ragozása, illetve félreérthetősége okozta valószínűleg azt, ami az osztályozás végül rossz kimenetéhez vezetett.

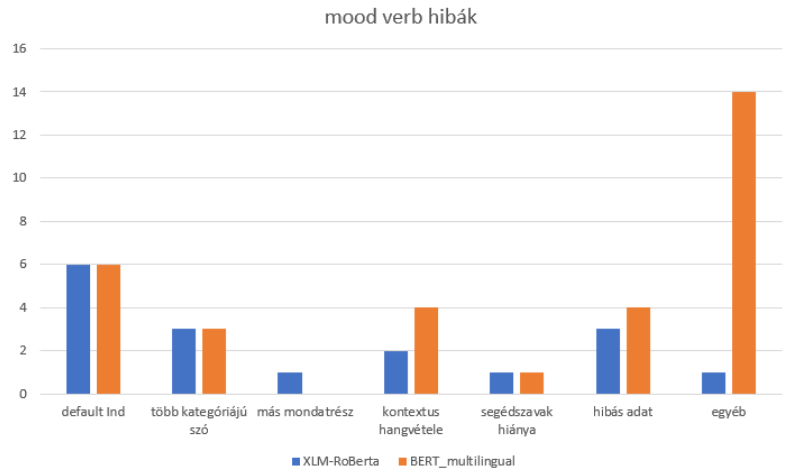
mood verb: Ez az adat csoport a célszóként szereplő igéket osztja négy osztályba. Ezek az Indicative, Imperative, Conditional és a Potential. Egyes források ezek közül egy-egy esetben többet is használnának, tehát egyes esetekben annak eldöntése, hogy melyik a négyből az osztály, a neurális háló számára is nehéz. A félreérthető vagy kétértelmű teszteseteket sem a vizsgált XLM-RoBerta , sem a BERT multilingual használatával sem sikerült jól osztályozni. Néhány hibás adat itt is előfordult, de elenyésző esetben. A BERT multilingual az alacsony pontossága miatt ezen adat csoport esetén is követett el egy kategóriába sem sorolható hibákat. Egy másik tipikus hibaforrás az volt, hogy egy kontextusból kijelentőnek titulálható mondatban szerepelt egy kijelentő vagy feltételes ige, ezzel kvázi

találgatásra késztetve a BERT modelt. Azt is észrevettem a BERT multilingual néhány tévedésénél, hogy ha nem volt biztos az osztályban, akkor az Indicative osztályt választotta. Az XLM-RoBerta használatával történt tévedések nagyobb részét teszi ki a hibás adat, és kevesebb az "egyéb", tehát kategória nélküli tévedés. A kétértelmű mondatok félreosztályozása itt is előkerült, olyan módon, hogy a célszó kontextusában a mondat más tagmondata nem azt az osztályt sugallja, ami a helyes. Néhány esetben a szó akár mindkét osztályba való besorolása tűnt az indoknak a hibára. A BERT multilingualhoz hasonlóan azonban itt is feltűnt, hogy sok Indicative tévedése van, tehát egy alapértelmezett döntése lehet a hálónak a kétes kontextusoknál.

A hibák kategóriákba sorolását a 4.7. ábra és a 4.8. ábra mutatja.



4.7. ábra. A három model hibáinak kategóriái a person psor noun adat csoportra.



4.8. ábra. BERT multilingual és az XLM-RoBerta hibáinak kategóriái a mood verb adat csoportra.

5. fejezet

Összefoglalás

A projekt során mélyebben megismertem több BERT model működését és lemértem teljesítményüket néhány adat csoport adataira, melyek egy scripttel lettek generálva. Az egyes adat csoportok sajátosságaitól függően más-más teljesítményt produkáltak a modelek és egyes rétegeik, ezzel bizonyítva, hogy mely feladat megoldására alkalmasak és melyekre kevésbé. Több szempontból is kiemelendő a SZTAKI által is fejlesztett HuBERT tisztán magyar nyelvre készített BERT model [5] . A teljesítménye jóval felülmúlta a vizsgált multilingual modeleket, amelyek az XLM-RoBerta [2] és a BERT multilingual [3]. Ennek egyik oka a nagyobb "szótárral" való rendelkezés, vagyis több szórészt ismer a tokenizáló, illetve, hogy az említett két model tanításai során használt adatok mindegyikén tanult, míg azok csak az egyiken, ezzel több kontextust megismerve, súlyait eszerint pontosítva.

Abban az esetben, ha valaki a legpontosabb BERT modelt szeretné használni egy adott nyelvű szövegre, akkor a legjobban az azon a nyelven tanított BERT modellel jár, hiszen a tapasztalatok, és a mostani eredmények is azt sugallják, hogy ezek a legtöbb esetben túlteljesítik a multilingual modeleket. Ha valaki többnyelvű környezetben használná a BERT modelt, akkor egyértelműen a multilingual modelek a célszerűbbek erre a célra, hiszen nem annyival rosszabb a teljesítményük, hogy megérje felbontani a szöveget apróbb részekre nyelv szerint és mindet a saját nyelvét használó BERT modelen átfuttatni.

Néhány adat csoport esetén az XLM-RoBerta legalább olyan jól teljesített, mint a HuBERT, ez arra enged következtetni, hogy a multilingual modelek teljesítőképessége is javulni fog még a jövőben, ezzel az elavultabb egynyelvű modelek nyelvein javulást elérve, ha azon a nyelven nem érné meg újra betanítani egy korszerűbb BERT modelt.

Lezárásképpen tehát megfogalmazható, hogy mind a multilingual, mind az egynyelvű BERT modelek előtt fényes jövő áll a természetes nyelvfeldolgozás területén.

Irodalomjegyzék

[1] Neural network. URL "https://databricks.com/glossary/neural-network".

[2] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Slav Petrov Jacob Devlin. Bert multilingual. https://github.com/google-research/bert/blob/master/multilingual.md, 2019.

[5] Dávid Márk Nemeskey. *Natural Language Processing Methods for Language Modeling.* PhD thesis, Eötvös Loránd University, 2020.

[6] Om Parkash. What are the key trade offs between overfitting and underfitting. URL "https://www.quora.com/What-are-the-key-trade-offs-between-overfitting-and-underfitting".

[7] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, pages arXiv–1910, 2019.