

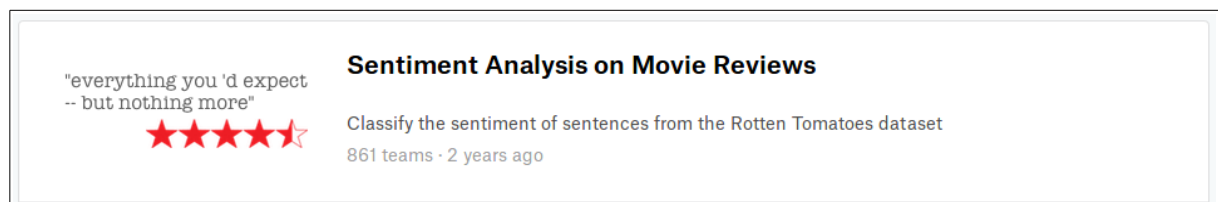
Filmkritikák érzelemelemzése

Neurális hálózatok házi feladat

Készítette: Sümegi Márk

Feladat leírása:

A feladatot a kaggle nevű honlapon találtam, ahol machine learning versenyeket hirdetnek ki különböző témákban, melyeken bárki részt vehet. A feladatot a rotten tomatoes nevű filmértékelő honlap ihlette, ahol a felhasználók rövid kritikákat írhatnak egy adott filmről, majd azt 1-től 5-ig terjedő skálán csillagokkal értékelik. A verseny célja minél nagyobb pontossággal meghatározni egy adott kritikából, hogy ezen kritika írója hány csillagra értékelte a filmet.



Ugyan első ránézésre a megoldás akadémiai területeken kívüli felhasználásra kevés esélyt lát az ember, ennek ellenére úgy vélem a természetes nyelvek feldolgozása, mondatok hangulatának elemzése, pozitív vagy negatív töltetének észlelése egy fontos kutatási terület, és hasznos lehet a jövőben.

Training Data:

A versenyfelületen közzétett tanítóadat halmaz a következőképp nézett ki: Kicsit több mint 150 000 felcímkézett részmondat (0-tól 4-ig, ahol a 0 a legrosszabb és 4 a legjobb) amik 8544 különböző kritika felbontásából születtek.

155939	8537	Reign of Fire is so incredibly inane that it is laughingly enjoyable .	3
155940	8537	is so incredibly inane that it is laughingly enjoyable .	2
155941	8537	is so incredibly inane that it is laughingly enjoyable	2
155942	8537	is so incredibly inane	1
155943	8537	so incredibly inane	0
155944	8537	incredibly inane	0
155945	8537	that it is laughingly enjoyable	3
155946	8537	it is laughingly enjoyable	3
155947	8537	is laughingly enjoyable	4
155948	8537	is laughingly	1

Külön tesztadat halmazt nem biztosítottak, így ~30 000 véletlenszerűen kiválasztott sort (20%) elkülönítettem későbbi tesztelésre, és a maradék 120 000 részmondattal dolgoztam tovább.

Mondatok vektorizálása:

A későbbi tanítás megkönnyítése miatt a részmondatokat azonos hosszúságú vektorokként szerettem volna reprezentálni, de olyan formában, hogy a mondat vektorizálása minél kevesebb információvesztéssel járjon. A konzisztens eredmények elérésének kritikus feltétele ezen kívül, hogy a hasonló mondatok vektorizálás után is hasonlóak maradjanak. A problémát a következőképpen oldottam meg:

Első lépésként készítettem egy szótárat a 120 000 részmondat fontosnak tűnő szavaiból. Ez azt jelenti, hogy egy listába gyűjtöttem azokat a szavakat amik egy adott számnál többször szerepelnek a tanítóhalmazban, mivel úgy gondoltam, hogy ezek a szavak fognak nagyobb valószínűséggel a teszteléskor is felbukkanni. A szavak különböző nyelvtani formáit nem vettem külön kategóriába, a szótári alak meghatározásához az NLTK (Natural Language Toolkit) lemmatize funkcióját használtam. A továbbiakban azt a megoldást mutatom be amit egy 5068 szavas szótár segítségével sikerült elérnem.

Minden részmondatot egy 1-esek és 0-ák 5068 elemű tömbjével írtam le, úgy, hogy ha a részmondatban szerepel a szótár i. eleme, akkor a mondathoz tartozó tömb i. eleme 1-es értéket vesz fel, egyébként 0-t.

Szemléltetésképpen tegyük fel, hogy a szótárunk csak 4 elemű a következő bejegyzésekkel:

{good, bad, ugly, movie}

Ebben az esetben a „good movie” kifejezés {1,0,0,1} vektorral, a „The good, the bad and the ugly” kifejezés pedig {1,1,1,0} vektorral lesz reprezentálva.

A fenti példából látszik, hogy ezzel a megoldással a szavak sorrendjéből származó információt elveszítjük, valamint a szótárban nem megtalálható szavakat egyszerűen nem vesszük figyelembe, ami könnyen pontatlanságokhoz vezethet a jövőben. Ellenben a hasonló mondatok hasonló vektorokká konvertálása adott, és a megegyező hosszúságot is elértük.

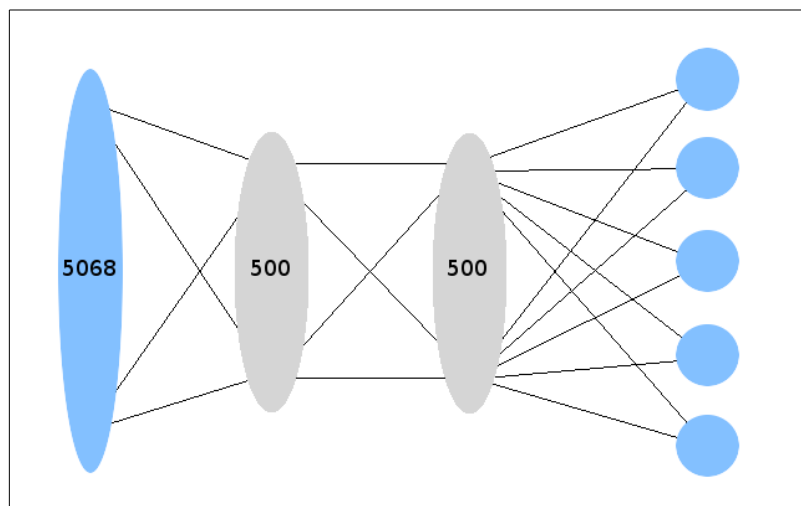
TensorFlow:

A TensorFlow egy nyílt forráskódú szoftverkönyvtár ami kifejezetten machine learning feladatok hatékony megoldására lett kifejlesztve a Google Brain Team által. Jelenleg az 1.0-s release érhető el, aminek a GPU verziója támogatja a tanítás videokártyával történő gyorsítását. (Csak Linux rendszereken, a CUDA és a cuDNN telepítése után, megfelelő számítási kapacitással rendelkező NVIDIA kártyákon)

A számításigényes TensorFlow függvények natív C/C++ ban vannak implementálva, ennek ellenére a könyvtár könnyen használható pythonból. Pythonban csak egy ún. tensorflow sessiont csinálunk, amit felparaméterezünk igényeink szerint. Amikor ezt a sessiont futtatjuk, akkor kerülnek kiszámolásra az adatok a háttérben, aminek a végén az eredményeket visszkapjuk. A könyvtárra épülnek további API-k, (pl. keras) amik egy egyszerűsített, magasabb szintű interface-t biztosítanak a felhasználóknak, de ilyeneket ebben a projektben nem használtam.

A neurális háló felépítése:

A megoldásom egy többrétegű, előrecsatolt neurális hálóval működik. Az input réteg 5068 db. node-ból áll, itt kerül bele a hálóba a vektorizált részmondat. A két rejtett rétegben 500-500 node található. Ennél többet terveztem, de a tesztek során kiderült, hogy a node-ok számának növelése az eredményen nem javít, csupán a feladat számításigényét növeli.



A háló output rétegében 5 db node található, ami az öt lehetséges értékelést reprezentálja. Felülről számítva ahányadik végül a legnagyobb kimenetet generálja (vagyis amelyik legerősebben tüzel) annyi pontot adunk a mondatnak.

A neurális háló tanítása:

A 120 000 tanítóadattal 10 epochot hajtottam végre, vagyis az összes adat végigfuttatása a hálón, majd az ezekhez tartozó backpropagation összesen 10-szer történt meg felváltva. A hibafüggvény ezután beállt egy viszonylag stabil értékre.

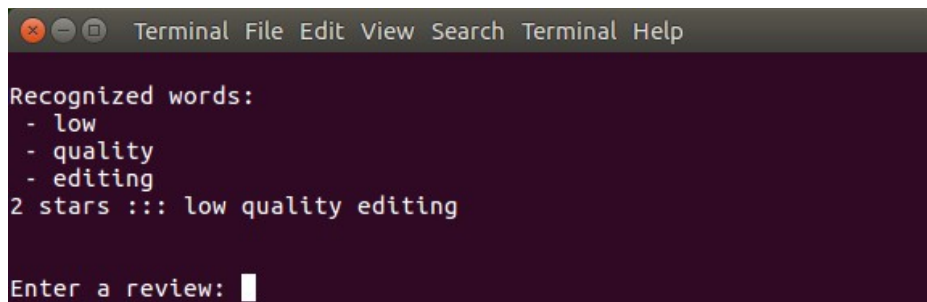
A loszt a TensorFlowba beépített softmax cross entropy függvénnyel számoltam, optimalizálónak pedig a beépített AdamOptimizer-t használtam.

A tanítási folyamat (az adatok előfeldolgozását és a szótár felépítését nem számolva) egy NVIDIA GTX 960M típusú videokártyán körülbelül 30 percet vett igénybe.

Hogy a kiszámolt eredmény ne vesszen el, lefutás után a modellt a TensorFlow checkpoint fájlformátumában elmentettem, ami a program újraindításakor automatikusan betöltődik, így rögtön lehet használni a modellt. Később megírtam, hogy ezt a mentési funkciót epochonként végezze el a program, így ha a tanítás közben történik valami, akkor se kelljen előről kezdeni.

Interface:

Az elkészült program kipróbálható és tesztelhető terminálból:



```
Terminal File Edit View Search Terminal Help

Recognized words:
- low
- quality
- editing
2 stars ::: low quality editing

Enter a review: █
```

Tesztelés:

Az elkülönített 30 000 példára a modell ebben az állapotában az esetek 45%-ában találja el, hogy egy filmkritika hány csillagra ítelt egy filmet. Ezt az eredményt árnyalja a tény, hogy az esetek több mint egynegyedében a program azért nem tud döntést hozni, mert a bemeneti mondat egyik szavát sem tartalmazza a szótár. Továbbá az is érdekes kérdés, hogy a hibás esetek mekkora százaléka kritikus hiba (abszolút pozitív mondatra negatívát hinni és fordítva), és mekkora százalékban téved csak plusz-mínusz egy csillaggal.

A modellt kézzel tesztelve véletlenszerűen kiválasztott valós hosszúságú kritikákkal ígéretes eredményeket figyelhetünk meg:

1 star ::: The plot is so senseless it could have been written up by bleary, sleep-deprived parents. It's all just a pretext for a pitiful one-joke movie.

4 stars ::: The weight of the piece , the unerring professionalism of the chilly production , and the fascination embedded in the lurid topic prove recommendation enough .

2 stars ::: Disney has ripped a jewel out of its casing and set it in something far more elaborate; the effect is garish rather than nostalgic, frustrating rather than memorable.

4 stars ::: An exceptionally smart, brooding picture with some terrific performances.

5 stars ::: A clever, brilliant film, superbly designed by Nathan Crowley and dramatically lit by Wally Pfister.

További érdekesség még, hogy annak ellenére, hogy a vektorizációnál a mondat elvesztette a struktúráját, mégis képes a modell különböző mondatszerkezeteket pl. tagadást felismerni:

3 stars ::: not

4 stars ::: very good

2 stars ::: not very good

Láthatjuk, hogy a „not” önmagában neutrális (tehát nem a „not” szócska negativitása húzza le a mondatot) a „very good” pedig a pozitív oldalon van, ennek ellenére a „not very good” mégis a negatív oldalra kerül. Ehhez hasonló példák még a következők:

1 star ::: bad

3 stars ::: not bad

2 stars ::: not so good

4 stars ::: solid

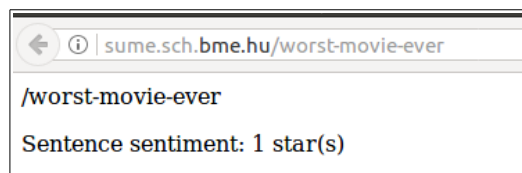
3 stars ::: not too solid

Fejlesztési lehetőségek:

További fejlesztési lehetőség, hogy a bemeneti vektorok ne csak 1 és 0 értéket vehessenek fel, hanem a felvett érték arányos legyen a szó tanítóadatban való előfordulásainak számával, ezzel indikálva a szó fontosságát. A szótár méretének növelése is nagy valószínűséggel javítaná a pontosságot, például, ha szavakhoz hozzá tudnánk rendelni szinonímákat.

Másik megközelítés lehetne még visszacsatolt háló használata, ami a nyelvfeldolgozásban bevett szokás. Ennek segítségével meg lehetne őrizni a mondatok szerkezetét, a szavak sorrendjét. TensorFlow-ban például implementálva van az LSTM cell.

A programot kiegészítettem egy egyszerű szerverszolgáltatással. Amennyiben a program fut, az alábbi képen látható módon bárki elérheti és használhatja a saját szórakozására. Ehhez kapcsolódó fejlesztési lehetőség lenne még felhasználók visszajelzései alapján folyamatosan tovább tanítani a modellt.



2017.05.17