Önálló laboratórium beszámoló BME-TMIT

Készítette: Aklan Péter Neptun-kód: E0TTPA
Szak: villamosmérnöki Szakirány: Infokommunikációs rendszerek
Ágazat: Infokommunikációs hálózatok és alkalmazások
E-mail cím: pethun9@gmail.com

Konzulens(ek): Orosz Péter, Skopkó Tamás
Email címe(k): orosz@tmit.bme.hu,

Tanév: 2021/2022. 2. félév
Téma címe: Felhőszolgáltatások minőségvizsgálata deep learning eszközökkel

Feladat:

A felhő-alapú szolgáltatások, platformok rohamos elterjedése a mindennapjainkra is jelentős hatás gyakorol. A felhasználók szolgáltatásokkal kapcsolatos növekvő elvárásai mellett a felhasználói kör is folyamatosan bővül. Emiatt kritikus feladat az internetszolgáltatók számára a szolgáltatások átviteli minőségének folyamatos monitorozása. Emellett a platformszolgáltatók is igyekeznek a szolgáltatás minőségét kiszolgálói oldalon is optimalizálni, ami a forgalmi tulajdonságok változását vonja maga után. Ennek a változásnak a lekövetését hagyományos algoritmusokon alapuló monitoring rendszerekkel csak nehézkesen lehet megvalósítani, hiszen minden módosítást validálni kell. Ezt a problémakört mélytanuló algoritmusokkal hatékonyan lehet kezelni, megfelelő módszerek kidolgozásával alkalmassá tehetjük a neurális hálózatokat 7/24-alapú hálózatfelügyeletre.

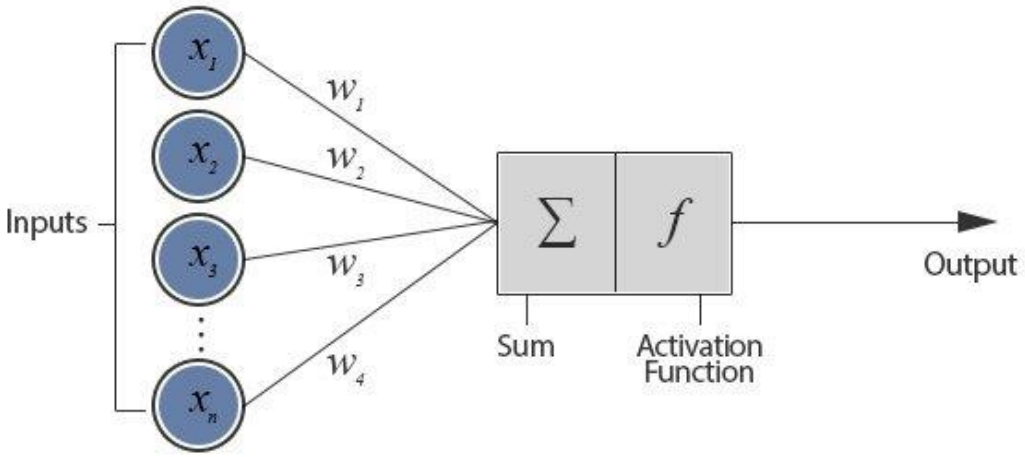
1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

Bevezető/elméleti összefoglaló

A deep learning a machine learning egy részhalmaza, amely lényegében egy három vagy több rétegből álló neurális hálózat. Ezek a neurális hálózatok megpróbálják szimulálni az emberi agy viselkedését – bár messze nem felel meg a képességeinek –, lehetővé téve számára, hogy nagy mennyiségű adatból „tanuljon”. Míg az egyrétegű neurális hálózat továbbra is hozzávetőleges előrejelzéseket tud készíteni, további rejtett rétegek segíthetnek az optimalizálásban és a pontosság finomításában.

A deep learning számos mesterséges intelligencia-alkalmazást és szolgáltatást vezérel, amelyek javítják az automatizálást, emberi beavatkozás nélkül végezve analitikai és fizikai feladatokat. A mély tanulási technológia a mindennapi termékek és szolgáltatások (például digitális asszisztensek, hangvezérlésű TV-távirányítók és a hitelkártya-csalás észlelése), valamint a feltörekvő technológiák (például az önvezető autók) mögött áll.

A legegyszerűbb neurális hálózat a „perceptron”, amely a legegyszerűbb formájában egyetlen neuronból áll. Hasonlóan a biológiai neuronokhoz, az egyetlen mesterséges neuron egy egyszerű fastruktúra, amely bemeneti csomópontokkal és egyetlen kimeneti csomóponttal rendelkezik, amely mindegyik bemeneti csomóponthoz kapcsolódik.



1. ábra a perceptron felépítése [1]

A perceptronok részei az 1. ábra alapján balról jobbra:

1. Bemeneti csomópontok. Minden bemeneti csomóponthoz numerikus érték tartozik, amely bármilyen valós szám lehet. Ezek lehetnek pozitívak vagy negatívak, egész vagy decimális számok.
2. Kapcsolatok. Hasonlóképpen a bemenetekhez, a bemeneti csomóponttól induló kapcsolatokhoz súly tartozik, és ez tetszőleges valós szám lehet.
3. Ezután a bemeneti csomópontok összes értékét és a kapcsolatok súlyát összeszorozzuk, amit súlyozott összegű bemenetként használunk fel a következő rétegben.
4. Ez az eredmény lesz az átviteli vagy aktiválási függvény bemenete. A legegyszerűbb, de triviális esetben ez az átviteli függvény egy azonosságfüggvény lenne, f(x)=x vagy (y=x). Ebben az esetben (x) a bemeneti csomópontok és a kapcsolatok súlyozott összege. Csakúgy, mint egy biológiai neuron, ami csak egy bizonyos küszöb túllépése esetén ad kimenetet, a mesterséges neuron is csak akkor ad kimenetet, ha a bemenetek összege túllép egy küszöbértéket.
5. Ennek eredményeként megvan a kimeneti csomópont, amely a bemeneti csomópontok súlyozott összegének függvényéhez (például a szigmoid függvényhez) van társítva.
6. Végül a perceptronnak lehet egy további paramétere (ami az 1. ábrán nincs feltüntetve), az úgynevezett bias, amelyet egy további bemeneti csomóponthoz társított súlynak tekinthet, amely állandóan 1-re van állítva. A bias érték kritikus, mert lehetővé teszi az aktiválási függvény eltolását a balra vagy jobbra, ami meghatározhatja a tanulás sikerét.

Ennek a modellnek az a logikai következménye, hogy a perceptronok csak numerikus adatokkal működnek. Ez azt jelenti, hogy minden névleges adatot numerikus formátumba kell konvertálnia.

A perceptronok hálózatai többrétegű perceptronok, és ezzel fogunk foglalkozni a beszámoló többi részében, ami Pythonban van megvalósítva a Keras segítségével. A többrétegű perceptronokat „előre csatolt neurális hálózatoknak” is nevezik. Ezek összetettebb hálózatok, mint a perceptron, mivel több neuronból állnak, amelyek rétegekbe rendeződnek. A rétegek számát általában két-háromra korlátozzák, de elméletileg nincs korlátozás.

A rétegek úgy működnek, mint a biológiai neuronok, amelyekről fentebb olvashattunk: az egyik réteg kimenetei a következő réteg bemeneteiként szolgálnak.

A rétegek között megkülönböztethetünk bemeneti réteget, rejtett rétegeket és kimeneti réteget. A többrétegű perceptronok gyakran teljesen összekapcsolódnak. Ez azt jelenti, hogy egy adott réteg minden perceptronja között van kapcsolat a következő réteg minden perceptronjával. Annak ellenére, hogy a csatlakoztatás nem követelmény, általában ez így van.

Vegyük figyelembe, hogy míg a perceptron csak az osztályok közötti lineáris elválasztást képviselheti (0 vagy 1), a többrétegű perceptron legyőzi ezt a korlátot, és összetettebb döntési határokat is képviselhet.

A rétegek közötti kapcsolatokról és egy neurális háló működéséről 3Blue1Brown [2] csatornáján lehet további videót nézni.

A munka állapota, készültségi foka a félév elején

Témalabor során a Discordról összegyűjtött forgalmat lehetett felhasználni kiindulásnak, illetve a konzulens által kapott és interneten összegyűjtött oktatási anyagot.

A következő dolgokat / fájlokat / leírásokat /programokat kaptam készen kézhez, tehát nem a félévi munkám során született:

- Discordról készített pcap fájlok wiresharkból lementve
- Python fejlesztői környezet: PyCharm 2021.3.2 (Community Edition)
- Datacamp Keras tutorial [1]

2. Az elvégzett munka és eredmények ismertetése

Az általam konkrétan elvégzett munka bemutatása

A félév elejei irodalomkutatás után, az oktatóanyag alapján összeraktam egy példahálózatot. Ez először a vörös- és fehér borok besorolását tanulta kémiai tulajdonságai alapján. Ennek megértése kulcsfontosságú volt a tovább haladáshoz, hogy megértsem a Keras alap szintű működését. Ennek a hálózatnak a bemenete egy táblázat, amiben meg van adva bor típusa. A kimenete pedig egy predikció, amely a tulajdonságok alapján megadja a bor típusát 0-1 közötti biztossági értékkel.

Ebből kiindulva nekem a Discordhoz is kellett egy bemenet és egy kimenet, ami anomália detektor lesz és azt tudja prediktálni az adatok alapján, hogy a Discord forgalma megfelelő e. Itt az 1 érték az, amikor a forgalommal nincsen probléma és a 0 pedig amikor valami nem felel meg az elvárásoknak.

```
# Importing data files
discord_voice_good = pd.read_csv("dc_voice_good.csv", sep=';', header=None)
discord_ctrl_good = pd.read_csv("dc_ctrl_good.csv", sep=';', header=None)
discord_ctrl_good = discord_ctrl_good.fillna(0) # fill NaN cells with 0

discord_voice_bad = pd.read_csv("dc_voice_bad.csv", sep=';', header=None)
discord_ctrl_bad = pd.read_csv("dc_ctrl_bad.csv", sep=';', header=None)
discord_ctrl_bad = discord_ctrl_bad.fillna(0) # fill NaN cells with 0
```

2. ábra bemeneti fájlok importálása

Az első bemenet, ami felmerült az a Discord hangkapcsolatának a streamje. Ez a kapcsolat egy tipikus VoIP kapcsolat, ebből adódóan UDP-t használ a késleltetés elkerülése végett és ha van adat ~50 csomag/másodperces sebességgel folyik. A lementett pcap fájlokat Wiresharkkal megnyitva exportálni lehet a forgalmat az I/O graph menüben. Az exportált csv fájloknak különböző osztást lehet használni, ez most jelen esetben másodperces, illetve csomagokat számolunk. Ebből egy csomag/másodperces táblázatunk van, ahol 1 sor az egy 133 másodperces felvétel a Discordról és ebből van 20 felvételünk, szóval 20 sor és 133 oszlop. Eleinte ez volt az egyetlen bemenet, amit a Pythonba importáltam a 2. ábrán látható módon Discord_voice_good néven, viszont ez kevésnek bizonyult ezért később kibővítettem.

A Discord működéséből adódóan amikor nem beszél senki, de kapcsolat fent van tartva, nem folyik a streamen adat. Ez azt jelenti, hogy az UDP kapcsolaton a csomag/másodperc (pps)

0. Ha csak ezt adjuk be a neurális hálónak, amivel anomáliákat keresünk, nem fogja jól megtanulni a különbséget a kettő között mert nincs (még akkor is, ha ez a Discord rendeltetésszerű működése). Ezt kiküszöbölve vezettem be a második bemenetet. Ez az UDP-hez tartozó TCP vezérlő kapcsolat, ami minden esetben ott van a hang kapcsolat mellett. Ennek az exportálása bit/másodpercben történt meg. Ebből ugyanúgy 20 sor és 133 oszlop van, mint a hang kapcsolatnál és hasonlóan importálom be a Pythonba Discord_ctrl_good néven.

```
def TimeSinceLastZero(input_table):
    data_help = discord_voice_good.to_numpy()  # numpy array only used for
getting size of array

    df = pd.DataFrame(data=input_table)
    TimeSinceLastX = np.empty([np.shape(data_help)[0],
np.shape(data_help)[1]])

    # For loop for filling in TimeSinceLastPacket_bad ndarray
    for row_no, row in df.iterrows(): # iterating in rows where row_no is
the row number, row is the data in a whole row
        col_iter = 0  # column iterator, 0 at the beginning of each row
        TSLX = 0
        for cell in row:  # iterating in columns where cell is one cell of
the row
            if input_table.equals(discord_ctrl_bad) or
input_table.equals(discord_ctrl_good): # counting bits or packets
                if cell == 0: # if there are no bits/s TSLB counts up
otherwise it's 0
                    TSLX += 1
                else:
                    TSLX = 0
            else:
                if cell < 40:  # if there are less than 40 packets/s TSLP
counts up otherwise it's 0
                    TSLX += 1
                else:
                    TSLX = 0
            TimeSinceLastX[row_no, col_iter] = TSLX  # filling the result
table used later
            col_iter += 1
    return TimeSinceLastX

TimeSinceLastPacket_good = TimeSinceLastZero(discord_voice_good)
TimeSinceLastBit_good = TimeSinceLastZero(discord_ctrl_good)
TimeSinceLastPacket_bad = TimeSinceLastZero(discord_voice_bad)
TimeSinceLastBit_bad = TimeSinceLastZero(discord_ctrl_bad)
```

3. ábra Time Since Last Bit és Packet előállítása

A két bemenet még kevésnek bizonyult a korábbi probléma kiküszöböléséhez, ugyanis a TCP kapcsolaton se folyamatosan jön adat, ezért van, hogy mindkét bemenet 0, vagy közel 0. Emiatt jelen esetben fontos az időbeliség. Nem mindegy, hogy most éppen nem beszél senki, vagy már nem jön egyáltalán adat semelyik csatornán 10 másodperce. Ezért számolnunk kell az időt amióta nem jött elég adat az adott streamen. Ezt Pythonba importálás után tudjuk a

legegyszerűbben megtenni. Erre egy függvény van, ami a 3. ábrán látszik. Ezen átfut az összes bemenet és attól függően, hogy vezérlő, vagy hang kapcsolat van, számolja az utolsó adat óta eltelt időt másodpercben. Ez a TSLX segéd változó a függvényben. Abban az esetben, ha jön adat, ez az érték 0 marad. Ez a függvény feldolgozza az egész táblázatokat és ugyanakkora méretű táblázatot készít belőlük (20 sor 133 oszlop).

A fenti kiinduló táblázatoknak valójában a duplája lesz az egész adatállomány, ahogy a 2. ábrán látható *_bad néven, mivel a neurális háló betanításához szükség van példa értékű rossz adatra. Ezek az eredeti TCP és UDP táblázatok lemásolásával és manuális elrontásával történtek. A további két táblázat, amit ezekből készít a Python ugyanazon a függvényen mennek keresztül, mint a „jó” példák.

Ezzel előállt a 4 darab bemenetünk, amit a neurális háló meg fog kapni. Mivel még ebben a formában nem lehet így beadni neki a 4 táblázatot ahogy van, ezért fel kell dolgozni a nyers adatokat. Ezt a folyamatot preprocessingnek hívjuk. Ez a neurális háló soronként tanítja az adatokat, ami azt jelenti a 4 táblázat összefüggő adatát 1 sorba kell rendezni. Ehhez végig kell iterálni a táblázat összes elemén. A gyakorlatban ez úgy néz ki, hogy a táblázatok egy-egy sorát kivesszük és egy segéd táblázatba rakjuk oszlopokba transzponálva. Ezt a segéd táblázatot aztán hozzá tudjuk fűzni, a végleges táblázathoz. A végén lesz két táblázat ebből a jó és rossz adatokkal, de ahhoz, hogy tanításkor felismerje a neurális háló is, hozzá rakunk egy extra oszlopot 1-esekel és 0-val kitöltve. Ezt a két táblázatot szintén egymás alá fűzzük.

```
X = discord_full.iloc[:, 0:4]

y = np.ravel(discord_full.type)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

scaler = StandardScaler().fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)
```

4. ábra Tanulás-tesztelés felosztás és standardizálás

A nagy táblázatunkat az összes adattal és 5 oszloppal fel kell osztani X és Y változókra, ami a 4. ábra első két sorában történik meg. Ez annyit tesz, hogy az Y érték lesz az, amit majd a neurális háló prediktál. Így értelemszerűen ez a típus oszlopunk lesz a 0-kal és 1-esekkel feltöltve. Az X lesz a 4 bemenettel teli táblázat. Ezt követően felosztjuk az adatállományt 2

részre, tanításra és tesztelésre. Itt véletlenszerűen választja ki azokat a sorokat, amik tesztelésre kerülnek. Ennek a lényege, hogy legyen egy tesztelési része az adatoknak, amiből nem tanítjuk a neurális hálót. Ezek lesznek a már betanított algoritmus bemenetei, amelyből a predikciókat fogja megmondani. Jelen esetben ez az összes adat 33%-a, ami a test_size=0.33-ból látszik a 4. ábrán. Ezeket táblázatokat sem mindegy, hogyan kapja meg a neurális háló. Normalizálni, kell, vagy standardizálni az adatokat. Itt az utóbbit visszük véghez. Abban az esetben, ha nem történne meg az egyik sem, a végeredmény biztos helytelen lenne, mert levágná az értékeket, amik nagyobb 1-nél.

A bemenetek feldolgozásával ezzel vége van és minden úgy néz ki, ahogy azt a neurális háló elvárja. Most kezdünk el dolgozni Keras nevű Python csomaggal. Ez a csomag mögött áll az egész neurális háló modellje. Ez a csomagnak mögött áll a TensorFlow, ami gépi tanulással és mesterséges intelligenciával foglalkozó backend.

```
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(20, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=20, batch_size=133, verbose=1)
```

5. ábra Neurális háló modell létrehozása

Először létre kell hoznunk egy modellt, ez az 5. ábrán az első sorban látható sequential osztályút használjuk. 3 rétegje lesz ennek a hálózatnak. Az első a bemeneti réteg, az utolsó a kimenet és a kettő között található a rejtett réteg. Jelenleg Dense rétegeket használunk, a zárójelen belül pedig a következőket lehet állítani balról jobbra:

- Rétegen belül a neuronok száma
- Aktivációs függvény
- Bemenet alakja, amely mivel soronként kapja meg a 4 oszlopos táblázatból, ezért itt egy 4 hosszú vektor (vagyis X_train oszlopainak száma)

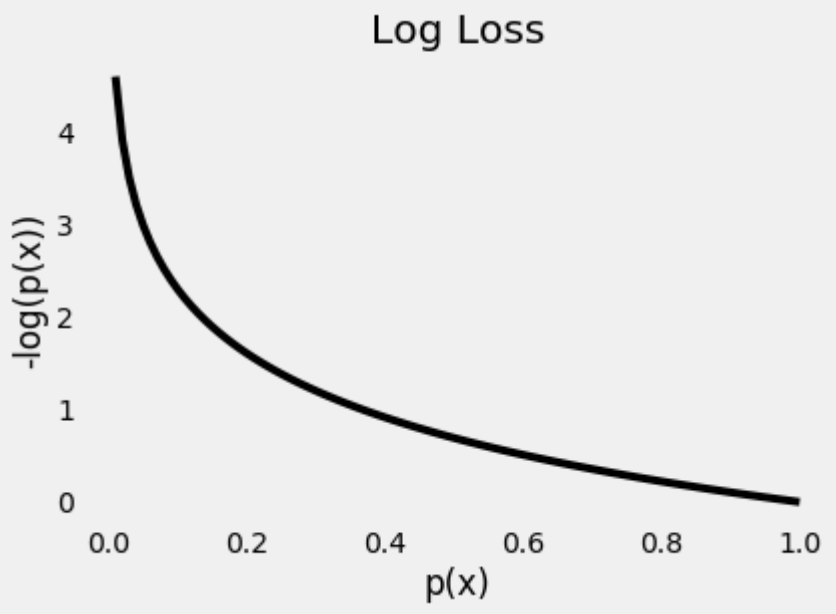
A neurális hálózat optimalizálásának jelentős része itt zajlik, a megfelelő rétegek, aktivációs függvények és neuronok mennyiségének megfelelő kiválasztásával. A kimeneti rétegnek kell egyedül sigmoidnak maradnia, mert az a függvény az, aminek a kimenete 0-1

között skáláz, ami nekünk kell. A fit függvényt használva adjuk be az X és Y táblázatokat, ami alapján tanul a háló. Itt az epochs változó adja meg hányszor fusson át az egész adatállományon, a batch_size pedig, hogy mekkora részletekben dolgozza fel az adatokat.

```
y_pred = model.predict(X_test)

for i in range(100):
    print(y_pred[i])

score = model.evaluate(X_test, y_test, verbose=1)

print(score)
```

6. ábra Modell tesztelése és pontossága

Az utolsó rész amire rátérünk az a modellnek a tesztelése predikcióval. A korábban kettéválasztott tanító és tesztelő adatállományból itt használjuk fel az utóbbit. A 6. ábra 1. sora végzi el a predikciót és a ciklusban kiírja az első 100-at. Ennek a lényege inkább csak pár eredmény megtekintése, illetve, ha fel akarnánk használni manuálisan a predikciókat összehasonlításra. Ezt a lépést nekünk az evaluate függvény teszi meg. Ennek a kimenete változó, az 5. ábrán látható compile függvény loss és metrics argumentumaitól függ ennek a kimenete. Ebből következőleg két kimenete lesz a függvénynek.



7. ábra Logaritmikus veszteség [3]

Az első a binary cross-entropy, vagy logaritmikus veszteség, ami a 7. ábrán látható. A predikciók logaritmikus értékének a negatív átlagát vesszük és ez lesz az eredmény. Erről részletesebben a Towards Data Science publikációjában lehet olvasni [3] Ez azt mondja meg,

hogy milyen jó a modellünk a predikcióban. Minél közelebb van az eredmény a 0-hoz annál pontosabb a predikció.

A másik a predikciók helyességének átlagát adja meg. Tehát hány százalékban felel meg a predikció a várt eredménynek.

```
[0.4357956051826477, 0.785876989364624]
```

8. ábra tesztelés végeredménye

Az én eredményeim a 8. ábrán látszanak. A logaritmikus veszteség 0.43-0.44 között mozog és a predikciók pontossága 78-79% körül van. Ebből látszik, hogy a háló nagyrészt működik, de van még mit rajta javítani.

A következő félévben, szakdolgozatra, elsősorban pontosítani kell a neurális háló működését a modell változtatásával, illetve tovább lehet építeni több irányba. Megvan a lehetőség arra, hogy valós időben dolgozza fel az adatot és ahogy Discordon mérem a forgalmat egyből adja ki a predikciókat, vagy akár azt is megmondhatja, hogy konkrétan mi a probléma a forgalommal. Ez azt jelentené, hogy nem csak anomália detektor lenne, hanem meg tudja mondani például a csomagvesztést, vagy ha kevés a sávszélesség. Ehhez már egy több szintű neurális hálóra lenne szükség.

Összefoglalás

- Elolvastam a Datacamp Keras útmutatóját [1] és a Towards Data Science binary cross-entropy publikációját [3]
- Megnéztem 3blue1brown deep learning oktatóvideóit [2]
- Készítettem 131 sor kódot
- Írtam heti beosztású munkatervet
- Írtam 10 oldal beszámolót

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

[1] Datacamp Keras tutorial

[2] Towards Data Science binary cross-entropy

[3] 3Blue1Brown deep learning oktatóvideó

Csatlakozó egyéb elkészült dokumentációk / fájlok / stb. jegyzéke:

A forrásfájlok és a teljes kód megtekinthető a Github repositorymban