Dr. Buzáné dr. Kis Piroska

Ismerkedés a TensorFlow rendszerrel

Absztrakt. A számítógépi hardver teljesítmények növekedése ösztönzően hat a különböző célú szoftverek kidolgozására, többek között a gépi tanulási módszerek fejlesztésére. A mesterséges neurális hálók kutatása több évtizedes múltra tekint vissza. A közelmúltban néhány rendkívül sikeres alkalmazásnak, mint például a képfelismerésnek, a beszéd-felismerésnek köszönhetően kiemelkedő népszerűségre tettek szert. A többrétegű neurális hálók programozására a Google kutatói – saját tapasztalataik alapján és más fejlesztők munkájának figyelembe vételével – 2015-ben kifejlesztették a TensorFlow rendszert. Ez a munka a „deep learning" célra megalkotott, és a gyakorlatban már bizonyított rendszer fő vonásait ismerteti.

Kulcsszavak: deep learning, mesterséges intelligencia, neurális hálók

Bevezetés

A TensorFlow egy szoftverkönyvtár, gépi tanulási algoritmusok leírására és végrehajtására. A TensorFlow rendszerben kifejlesztett számítások változatlanul vagy csekély változtatással végrehajthatók nagyon eltérő hardver eszközökön a mobil telefonoktól és tabletektől kezdve, grafikus kártyákon (GPU) át, sok számítógépből álló elosztott számítógép-rendszerekig. A TensorFlow roppant flexibilis, nagyon széles körű algoritmusok megvalósítására alkalmas, beleértve a deep neural network – sokrétegű neurális háló – alkalmazásait, például a beszédfelismerésben, a számítógépi látásban, megjelenítésben, a robotikában, az információ kinyerésben, a számítógépek elleni támadások felderítésében, és az agykutatásban [1]. Alkalmazható a számítástudomány más területein is.

A fejlesztés előzményei

A TensorFlow rendszer kifejlesztésének előzményei a GoogleBrain projektre nyúlnak vissza, amely 2011-ben indult azzal a céllal, hogy mind kutatási, mind Google alkalmazások számára sokrétegű neurális hálók használata váljon lehetővé nagyon széles körűen [2]. A projekt első, korai részeként a DistBelief rendszer készült el széles körű kutatásokra [3-11] .

A GoogleBrain kutatócsoporttal szorosan együttműködve több mint 50 kutatócsoport alkotott mély neurális háló modelleket a DistBelief rendszer használatával, többek között a Google keresőrendszer [12], Google Fotók [13], Google Térkép és Utcakép [14], Google Fordító [15], YouTube számára. A DistBelief rendszerrel szerzett tapasztalatok és neurális hálókat használó rendszerek jobb megismerése alapján került kifejlesztésre a TensorFlow rendszer, amely képes nagy tömegű adat alapján gépi tanulási modellek létrehozására. A fejlesztők ügyeltek arra, hogy a TensorFlow rendszer egyrészt flexibilis legyen a kutatás számára (az új modellek gyorsan megvalósíthatóak legyenek, és a velük való kísérletek könnyen elvégezhetőek legyenek), másrészt a valós alkalmazásokkal szemben támasztott követelményeknek megfelelően robosztus és hatékony legyen.

A TensorFlow rendszer sokrétegű neurális háló modellek készítésén kívül széles körben alkalmazható más célokra is, ideértve a más jellegű gépi tanulási algoritmusok és a különféle numerikus számítások implementálását.

Alapelvek és főbb tulajdonságok

A TensorFlow számítást egy irányított gráf írja le. Adatáramlás a gráf élei mentén történik. A TensorFlow gráfban mindegyik csúcs egy műveletet reprezentálhat és mindegyik csúcsnak lehet nulla vagy több inputja, ugyanígy nulla vagy több outputja. A gráf normál élei mentén áramló értékek tenzorok, tetszőleges dimenziójú vektorok. Egy-egy elem típusát a gráf konstruálásakor specifikálják. Lehetnek a gráfban speciális élek is, amelyek mentén nem történik adatáramlás, hanem kontrol célokat szolgálnak.

Egy TensorFlow műveletnek neve van és egy absztrakt számítást reprezentál. A műveletnek lehetnek attributumai, amelyeket a gráf konstruálásakor kell megadni. A TensorFlow kernel egy művelet egy olyan konkrét implementációja, amely egy adott eszköztípuson – pl. CPU, GPU – futtatható. A kliens programok a TensorFlow rendszerrel session létrehozásával kerülnek kapcsolatba. A session létrehozásához a Session interface rendelkezésre bocsájt egy Extend metódust azért, hogy a kurrens session további éleket és csúcsokat kezelhessen. A session létrehozásakor a kezdeti

gráf üres. A Session interface által szolgáltatott másik alapvető művelet a Run. Ez a művelet megkeresi a kiírandó output neveket és kiszámolja az értékeiket. TensorFlow implementáció az egyes csúcsok közötti függőségi viszonyok figyelembe vételével képes végrehajtani a műveleteket.

A neurális hálók súlyait kereső iteratív optimalizációs eljárások során tipikusan hasonló számításokat hajtjuk végre egymást követően sokszor. Minden egyes tanító példára végig kell számolni a háló összes kapcsolatának új súlyait és minden neuron új torzítási értékeit [16]. Ez azt jelenti, hogy igen sokszor lényegében ugyanazon számítási műveletsort hajtjuk végre, esetenként más-más adatokkal. Hasonló a helyzet az ajánlórendszerekben [17] vagy hatóanyagok és farmakológiai támadáspontok közötti kapcsolatokat kereső [18] mátrix faktorizációs eljárások esetén is. A hasonló számítások nagyszámú végrehajtása miatt a TensorFlow rendszer használóinak többsége egyszer hoz létre egy sessiont egy számítási gráffal és azután a teljes gráfot vagy annak egy részgráfját hajtja végre tetszőlegesen sokszor a Run hívásával.

A támogatott frontend nyelv a C++ és a Python.

Kiterjesztések, beépített szolgáltatások, optimalizálások, vizualizációk

A TensorFlow rendszer használatát nagyszámú optimalizáló algoritmus segíti, valamint a nevezetes, ismert gépi tanulási algoritmusok rendelkezésünkre állnak. A TensorFlow rendszernek van például beépített automatikus gradiens kiszámító szolgáltatása.

Gyakran előfordul, hogy a kliens a teljes gráfnak csak egy részgráfját kívánja végrehajtani. A Run metódus lehetővé teszi egy tetszőleges részgráf végrehajtását és tetszőleges adat bevitelét és az adatkinyerést a gráf bármely éle mentén.

A TensorFlow rendszer tartalmaz optimalizálást a számítási gráfban előforduló redundancia kiszűrésére, a memóriahasználatra, az adatbevitelre, a kernel implementáció kiválasztására.

A felhasználók számára a számítási gráfok szerkezetének áttekintését segíti a TensorBoard vizualizáló eszköz. Ez az eszköz a gráf megértésén túl a gépi tanulási

modell általános viselkedésének tanulmányozását is lehetővé teszi. A TensorBoard-dal különféle összegező statisztikák is készíthetők és megjeleníthetők.

Program TensorFlow rendszerrel

A TensorFlow rendszer használatát a honlapján közzétett információk és számos tutorial segíti. Mint már említettük, a TensorFlow rendszerben az adatok egysége a tensor, a TensorFlow programok virtuálisan egy irányított gráfnak tekinthetők, az élek mentén áramlanak az adatok, a csúcsok adatokat és műveleteket reprezentálnak. Minden csúcs inputja lehet több tenzor vagy akár nulla tenzor és mindegyik csúcs outputja egy tenzor.

A TensorFlow-val készített programok általában két részből állnak:

- számítási gráf felépítése,
- számítási gráf futtatása.

Példaként tekintsünk egy nagyon egyszerű gráfot. Vegyünk fel két csúcsot, amelyek az 5 és a 9 konstansoknak felelnek meg.

```
node1 = tf.constant(5.0, dtype=tf.float32)
node2= tf.constant(9.0)
```

Megjegyzés: A második csúcs felvételekor kihasználtuk a dtype=tf.float32 alapértelmezést.
Ha ezután kiadunk egy print utasítást

```
print(node1, node2)
```

akkor ennek az eredménye

```
Tensor(”Const:0”, shape=(),dtype=float32)
Tensor(”Const_1:0”, shape=(), dtype=float32)
```

várakozásunktól eltérően nem az 5 és a 9 számok.
A csúcsok kiértékeléséhez le kell futtatnunk a szekcióban a számítási gráfot.
Az alábbi kóddal szekció objektumot hozunk létre, majd futtatjuk a számítási gráfot a csúcsok kiértékelésére:

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

Az alábbi, várt eredményt kapjuk:

```
[5.0, 9.0]
```

Egyszerű, de akár összetett számításokat is végezhetünk, mivel a gráf csúcsaiban műveleteket is megadhatunk. Példaként adjuk össze az előző két konstansunkat.

```
node3 = tf.add(node1, node2)
print("node3: ", node3)
print("sess.run(node3): ", sess.run(node3))
```

A print utasítások eredménye:

```
node3 :  Tensor("Add: 0", shape=(), dtype=float32)
sess.run(node3) : 14.0
```

A gráf paraméterezhető úgy, hogy tudjon külső értékeket, úgy nevezett „placeholder"-eket fogadni. A placeholder esetén nem tudjuk előre, hogy milyen értéket kell tárolni, az értéket később kapja meg.

Például legyen az a és b placeholder, később kapnak értéket:

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b   # + az összeadás művelet rövid
                     # megadása
```

A példában két input paramétert adtunk össze. Ezt a számítási gráfot többször is tudjuk alkalmazni, az összeadandók típusa is változhat:

```
print(sess.run(adder_node, {a : 4,  b: 12}))
print(sess.run(adder_node, {a : [3, 1],  b: [1, 2]}))
```

Az eredmény:

```
16
[4.  3. ]
```

További műveletekkel, például egy szorzás művelettel bonyolultabb számítási gráfot készíthetünk.

```
add_and_hatszor = adder_node * 6.
print(sess.run(add_and_hatszor,  {a: 3, b: 5}))
```

Az eredmény:

```
48
```

A gépi tanulás során tipikusan olyan modellre van szükségünk, amely tetszőleges inputot tud fogadni. A modell taníthatóvá tételére a gráfot módosítanunk kell úgy, hogy ugyanabból az inputból új kimeneteket produkáljon. A változók teszik lehetővé, hogy egy gráfhoz tanítható paraméterek tartozhassanak. Nézzük az alábbi példát változó alkalmazására:

```
w = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linearis_modell = w*x +b
```

A TensorFlow programban az összes változó inicializálásához explicite hívni kell egy speciális műveletet:

```
init = tf.global_variables_initializer()

sess.run(init)
```

Fontos megjegyeznünk, hogy a sess.run hívása előtt nincsenek inicializálva a változók. Mivel x placeholder, a linearis_modell -t többször is kiértékelhetjük x különböző értékeinél:

```
print(sess.run(linearis_modell,{x : [ 4, 2, 3, 7 ]}))
```

A modell tanítása

A munkánk során a feladatunk megoldására létrehozunk egy modellt, azonban nem tudjuk, mennyire jó az a modell. A modellünk kiértékeléséhez meg kell adnunk a hiba (loss) függvényt. A hiba függvény azt méri, hogy a modell a megkövetelt adatoktól mennyire tér el. Gyakori a négyzetes eltérés függvény, mint hiba függvény használata.

A linearis_modell példánkban a modell kiértékeléséhez szükségünk van egy placeholderre, legyen ez y. Példánkban a lineáris regresszióra vonatkozó standard hibamodellt használjuk. Esetünkben a linearis_modell–y egy vektort (negyzetes_hiba_vektor) hoz létre, amelynek mindegyik eleme a szóban forgó példa hibája, ezután a hibák négyzetösszegét vesszük, majd a négyzetes hibákat összegezzük, s eredményül egy skalárt kapunk:

```
y = tf.placeholder(tf.float32)

negyzetes_hiba_vektor = tf.square(linearis_modell-y)

hiba = tf.reduce_sum(negyzetes_hiba_vektor)
```

Előfordulhat, hogy a modellparaméterek (w,b) értéke ismert és ilyen esetekre futtatjuk a megalkotott modellünket. Ekkor a hibafüggvény értékének illik közelítőleg nullának lennie.

A gépi tanulás lényege, hogy automatikusan megtaláljuk a modellparaméterek értékeit. A következő bekezdésben azzal foglalkozunk, hogyan tudjuk ezt megtenni.

A hibafüggvény minimalizálására a TensorFlow rendelkezésre bocsájt optimalizáló algoritmusokat – optimizers – amelyek mindegyik változó kismértékű változását idézik elő. Ezek közül a legegyszerűbb a gradiens lejtő (gradient descent). Ez az optimalizáló deriváltak számításán alapul. Elsősorban bonyolultabb esetekben (többek között sokrétegű neurális hálók, „egzotikus" regularizációs tényezők illetve aktivációs függvények esetén) a deriváltak kézi számítása hosszadalmas és hibalehetőségeket tartalmaz. A TensorFlow azonban automatikusan elvégzi a számításokat a hibahatárok figyelembevételével:

```
optimizer = tf.train.GradientDescentOptimizer(0.01)

train = optimizer.minimize(hiba)
```

Példa gépi tanulásra

Nézzünk egy egyszerű mintapéldát gépi tanulásra a TensorFlow segítségével. Kézírású számjegyek felismerésére készítünk egy modellt. Egy ilyen a feladatot tekintünk a programozás tanulásakor a képernyőre kiírt első üzenet, a „Hello Word" megfelelőjének a TensorFlow-val való ismerkedés során. Adott egy adatállomány – MNIST– amely kézírással írt számjegyek képeit tartalmazza. Mindegyik kép 28×28 pixel. Ezt a négyzet alakú képet átalakíthatjuk vektor formájúra, amely vektor 28×28=784 komponenst tartalmaz. Az átalakításnál csak arra kell ügyelnünk, hogy mindegyik kép esetén ugyanazt a módszert alkalmazzuk. (Az egyszerű alkalmazásoknál a 2D képek 1D képekké alakítása elfogadott, míg a számítógépi látással/megjelenítéssel kapcsolatos legjobb módszerek azonban kihasználják a képek 2D szerkezetét is.) Az adatállomány mindegyik eleméhez egy címke is tartozik, amely egy 0 és 9 közötti számjegy, s a képen látható számot mutatja. Esetünkben a címke egy 10 komponensű „one-hot" vektor. Az 1 számjegyet egyetlen pozíción, az *n*-ediken tartalmazza, annak megfelelően, hogy melyik számjegy látható a képen, a többi komponens 0. A gépi tanítási elvet követve, adataink egy részét – többnyire a többségét – tanításra, egy további résztét modellünk jóságának mérésére használjunk.

Osztályba sorolás elméleti alapja - a matematikai modell

Ha egy egyedről/tárgyról el kell döntenünk, hogy több különböző egyed/tárgy közül melyikkel milyen valószínűséggel egyezik meg, akkor erre a célra a „softmax" függvény használható, mivel a „softmax" megad egy listát az egyezési valószínűségekre vonatkozóan, ahol az értékek 0 és 1 közöttiek, és az összegük 1. Röviden: a „softmax" valószínűségeloszlást ad meg.

A softmax regresszió két lépésből áll:

- kiszámítunk egy evidenciát, egy számot mindegyik osztály esetén, amely egy input adatnak az adott osztályba tartozására utal
- megadjuk az input adat egyes osztályokba tartozási valószínűségeit.

Adott *x* input adat esetén egy *i* osztályba tartozás evidenciája

$$evidencia_i = \sum_j W_{i,j} x_j + b_i,$$

ahol W_i az *i*-edik osztályra vonatkozó súlyokat jeleneti, b_i a torzítási (bias) érték, *j* index az *x* input képünk összes pixelének összegzésére.

Ezt követően az evidenciákból valószínűségeloszlást származtatunk. Ez az általunk megjósolt y valószínűség az egyes osztályokba tartozás valószínűsége:

$$y = softmax(evidence)$$

Itt a „softmax" aktiválási vagy kapcsolási függvénykánt szolgál. A lineáris függvényünk outputját az általunk kívánt formára alakítja. Ez esetben tíz osztályra vonatkozó valószínűségeloszlássá.

Röviden úgy foglalhatjuk össze a „softmax" tevékenységét, hogy egy *x* inputhoz kiszámítja az egyes osztályokba tartozás súlyait, azután megadja az osztályokba tartozási valószínűségeket. Röviden felírva:

$$y = softmax(Wx + b)$$

A modell definiálása a TensorFlow rendszerben

Néhány szükséges előkészítő sor után mindössze egyetlen sor TensorFlow utasítás elegendő a modell definiálására.

Előkészületek

A használat előtt a TensorFlow-t importálni kell:

```
import tensorflow as tf
```

A művelet végzéséhez egy *x* változó definiálása:

```
x=tf.placehorder(tf.float32, [None, 784])
```

Így megadtunk, hogy a képek adatállományunk minden egyede egy 784 dimenziós vektor. A „None” itt azt jelenti, hogy a beolvasandó egyedek száma bármennyi lehet.

A modellünkhöz szükségünk van súlyokra és bias értékekre, s ezek kezdeti értékét - akár tetszőlegesen is megadhatjuk - most nullának választjuk:

```
W=tf.Variable(tf.zeros([784, 10]))

b=tf.Variable(tf.zeros([10]))
```

A modell felírása

A modell implementálása mindössze egyetlen sor:

```
y=tf.nn.softmax(tf.matmul(x,W)+b)
```

Az, hogy a modell ilyen egyszerűen felírható, nem azért van, mert a TensorFlowt úgy tervezték, hogy a “softmax” regresszió különösen könnyen végrehajtható legyen, hanem mert maga a TensorFlow nagyon flexibilis sokféle numerikus számítás leírására a gépi tanulástól a fizikai szimulációig. A modellt a definiálása után különféle eszközökön futtathatjuk, például CPU-n, GPU-n vagy okos telefonon.

A modell tanítása

A modellünk tanításához meg kell határoznunk, mit értünk jó modellen. A gépi tanulás esetén tipikusan azt mondjuk meg, mit jelent az, hogy rossz a modell. Megpróbáljuk minimalizálni a hibát, és minél kisebb a hibahatár, annál jobb a modellünk. Nagyon általános a keresztentrópia hibafüggvény használata, amelynek definíciója az alábbi:

$$H_{y'}(y) = -\sum_i y_i' \log(y_i),$$

ahol y a modellünk által megjósolt, az y' pedig az igazi valószínűségeloszlás.

Nagyvonalakban szólva azt mondhatjuk, hogy a keresztentrópia azt méri, hogy mennyire elégtelenek/helytelenek a jóslataink a valóságos helyzet leírására. A keresztentrópia implementálásához szükségünk van egy újabb placeholderre – legyen ez y_ – a helyes válasz bevitelére:

```
y_=tf.placehorder(tf.float32, [None, 10])
```

Ezután következhet a keresztentrópia implementálása:

```
cross_entropy=tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),
      reduction_indices=[1]))
```

Megjegyzés. A keresztentrópia számítására más lehetőség is rendelkezésre áll.

Amit a modellünkkel megcsináltatni akarunk, azt a TensorFlow-val könnyen megtaníthatjuk neki. A TensorFlow ismeri a teljes számítási gráfot, automatikusan használja a backpropagation – a hiba visszaterjedési – algoritmust.

A tanítás során a hiba csökkentésére használhatjuk a gradiens módszeren alapuló optimalizáló eljárást:

```
train_step=tf.train.GradientDecsentOptimizer(0.05).
     minimize(cross_entropy)
```

Most már a modellünket interaktív session-ba tehetjük:

```
sess=tf.InteractiveSession()
```

Először inicializálni kell a változókat:

```
tf.global_variables_initializer().run()
```

Ezután kezdődhet a tanítás. Példánkban 2000-szer futtatjuk a tanítási lépést, az input adatokat 100-as kötegekben hozzuk be.

```
for in range(2000):

batch_xs, batch_ys=mnist.train.next_batch(100)

sess.run(train_step,feed_dict={x:batch_xs,y_:batch_ys})
```

A modell kiértékelése

A munkánkat nagyon megkönnyíti a tf.argmax függvény, amely a vektorban/tenzorban előforduló legnagyobb elem indexét adja eredményül.

A tf.argmax(y,1) mindegyik input adat esetén azt a címkét eredményezi, amelyet a modellünk a legvalószínűbbnek talál, míg a tf.argmax(y_ ,1) a helyes címke.

A modell jóslásainak helyességét ellenőrizhetjük az alábbi összehasonlítással:

```
helyes_joslas=tf.equal(tf.argmax(y,1), tf.argmax(y_ ,1))
```

Az eredmény egy „True”, „False” értékeket tartalmazó lista, amelyet lebegőpontos számokká konvertálunk, majd kiszámítjuk az átlagértéket.

```
pontossag=tf.reduce_mean(tf.cast(helyes_joslas,

tf.float32))
```

Modellünk pontosságát a teszt adatainkra nézve kiszámíttatjuk és kiíratjuk:

```
print(sess.run(pontossag,feed_dict={x: mnist.test.images,
      y_: mnist.test.labels}))
```

A fent bemutatott programkód (teszt2.py):

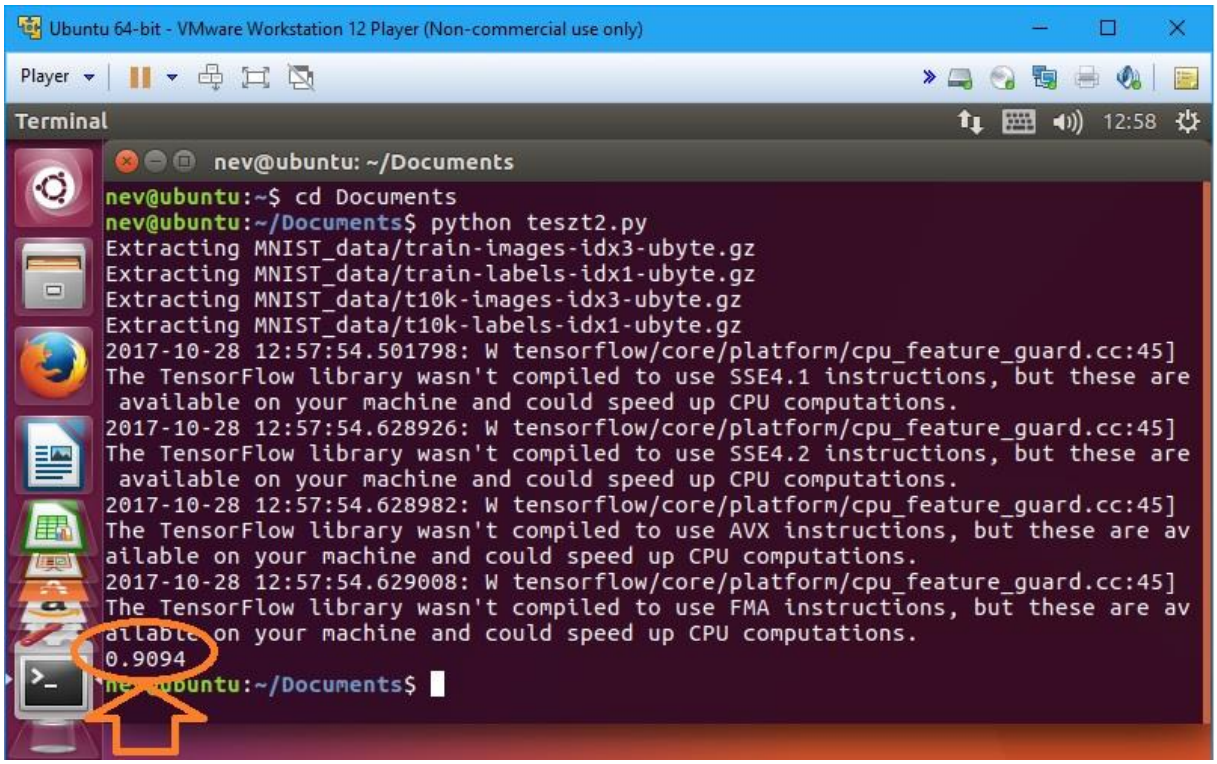
```
from tensorflow.examples.tutorials.mnist import input_data import
            tensorflow as tf


# Import data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)



# Define loss and optimizer

y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),
           reduction_indices=[1]))

train_step=tf.train.GradientDescentOptimizer(0.05).minimize(
           cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train

for _ in range(2000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# Test trained model

helyes_joslas = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
pontossag = tf.reduce_mean(tf.cast(helyes_joslas, tf.float32))
print(sess.run(pontossag, feed_dict={x: mnist.test.images, y_:
           mnist.test.labels}))
```

Az adatokat a `http://yann.lecun.com/exdb/mnist/` helyről töltöttük le a `MNIST_data` könyvtárba.

A program működtetése az 1. ábrán látható.

Amint a 1. ábra mutatja, a példaként megalkotott modell a konkrét esetben 90.94% pontosságot produkált, általában a hozzá hasonló nagyon egyszerű modellekkel a tapasztalatok szerint 90%-ot meghaladó, 91%-93% pontosság érhető el. Nagy javulást

lehet azonban elérni viszonylag csekély változtatásokkal, amelyek 97%-os, sőt 99,7%-os pontossághoz vezetnek.



1. ábra: a bemutatott eljárás működése

Kitekintés

A példában szereplő lineáris regresszió nem igényelt sok kódot, azonban ennél lényegesen bonyolultabb modelleket is alkothatunk, amelyek lényegesen több kódot tartalmaznak. A TensorFlow magas szintű absztrakciót tesz lehetővé struktúrákra, funkciókra, mintákra nézve.

A gépi tanulás támogatására rendelkezésre áll a tf.contrib.learn magas szintű TensorFlow könyvtár az alábbi tartalommal:

- tanítási ciklusok futtatása,
- kiértékelő ciklusok futtatása,
- adat halmazok / állományok kezelése,
- adatbevitel kezelése.

A tf.contrib.learn könyvtárban sok ismert modell is megtalálható. A tf.contrib.learn könyvtár nem kényszeríti ránk az előre definiált modelljeit. Ha olyan modellt kívánunk alkotni, amelyik nincs beépítve a TensorFlow könyvtárba, akkor is használhatjuk a tf.contrib.learn könyvtár adatállományok kezelésére, bevitelére, modellek tanításra vonatkozó magas szintű támogatását. A TensorFlow tutorialjaiból kezdők és haladók egyaránt sok ötletet meríthetnek.

Virtuális gép az ismerkedéshez

A TensorFlow használatához több szoftvercsomagra is szükség van. Az operációs rendszer (Ubuntu Linux) és a megfelelő komponensek installálása linuxos előismereteket, időt és munkát igényel.
Azért, hogy megismerkedhessünk a TensorFlow rendszer tényleges használatával Adam Geitgey összeállított egy megfelelően konfigurált és szoftverekkel "felszerelt" virtuális gépet.
Ez elérhető a:

`https://medium.com/@ageitgey/try-deep-learning-in-python-now-with-a-fully-pre-configured-vm-1d97d4c3e9b`

címen.

Ekkor egy

Ubuntu Linux Desktop 16.04 LTS rendszert kapunk, benne

- python3
- face_recognition
- keras 2.0
- TensorFlow 1.0
- dlib 19.4
- OpenCV 3

komponensekkel. Ennek jelenleg legfrissebb változata 2017.03.28-i, mérete: 5.4G. Ez a virtuális gép Windows,Linux rendszerekben a VMware Workstation Player-el, Mac rendszerekben a VMWare Fusion-al működtethető.

Mintapéldák elérhetőek a TensorFlow dokumentációján felül az alábbi helyen: https://github.com/aymericdamien/TensorFlow-Examples

Összefoglalás

A Google szotfver fejlesztői által megalkotott TensorFlow rendszer gyakorlati alkalmazásaival nap mint nap találkozunk. A főként a sokrétegű mesterséges neurális háló modellek programozására kidolgozott rendszer számos más területen is jól alkalmazható. A TensorFlow könyvtárban rendelkezésre álló segédanyagok lehetővé teszik színvonalas modellek gyors és egyszerű létrehozását.
A Google menedzsmentje a TensorFlow rendszer alkalmazását, terjesztését azzal is segíti, hogy a www.tensorflow.org honlapról a TensorFlow szabadon letölthető.

Irodalomjegyzék

[1] Regina Meszlényi, Krisztian Buza, and Zoltán Vidnyánszky. „Resting state fMRI functional connectivity-based classification using a convolutional neural network architecture.” Frontiers in Neuroinformatics, https://doi.org/10.3389/fninf.2017.00061 (2017).

[2] Jeffrey Dean, Gregory S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc’Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In NIPS, 2012. Google Research PDF.

[3] Quoc Le, Marc’Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In ICML’2012, 2012. Google Research PDF.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In International Conference on Learning Representations: Workshops Track, 2013. arxiv.org/abs/1301.3781.

[5] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. Technical report, arXiv:1412.7449, 2014. arxiv.org/abs/1412.7449.

[6] Andrea Frome, Greg S Corrado, Jonathon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. DeVISE: A deep visual-semantic embedding model. In Advances in Neural Information Processing Systems, pages 2121–2129, 2013. research. google.com/pubs/archive/41473.pdf.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In CVPR'2015, 2015. arxiv.org/abs/1409.4842.

[8] Andrej Karpathy, George Toderici, Sachin Shetty, Tommy Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 1725–1732. IEEE, 2014. research. google.com/pubs/archive/42455.pdf.

[9] Matthew D. Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Mark Mao, Ke Yang, Quoc Le, Patrick Nguyen, Andrew Senior, Vincent Vanhoucke, Jeff Dean, and Geoffrey E. Hinton. On rectified linear units for speech processing. In ICASSP, 2013. research. google.com/pubs/archive/40811.pdf.

[10] Geoffrey E. Hinton, Li Deng, Dong Yu, George E.Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen,Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Process. Mag., 29(6):82–97, 2012. www.cs.toronto.edu/˜gdahl/papers/deepSpeechReviewSPM2012.pdf.

[11] Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, Marc'Aurelio Ranzato, Matthieu Devin, and Jeffrey Dean. Multilingual acoustic models using distributed deep neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, pages 8619–8623. IEEE, 2013. research. google.com/pubs/archive/40807.pdf.

[12] Jack Clark. Google turning its lucrative web search over to AI machines, 2015. www.bloomberg.com/news/articles/2015-10-26/googleturning-its-lucrative-web-search-over-to-ai-machines.

[13] Chuck Rosenberg. Improving Photo Search: A step across the semantic gap, 2013. googleresearch.blogspot.com/2013/06/improvingphoto-search-step-across.html.

[14] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In International Conference on Learning Representations, 2014. arxiv.org/pdf/1312.6082.

[15] Otavio Good. How Google Translate squeezes deep learning onto a phone, 2015. googleresearch.blogspot.com/2015/07/how-googletranslate-squeezes-deep.html.

[16] B. Kis Piroska, Buza Antal, Bevezetés az adatbányászat egyes fejezeteibe, ISBN: 978-963-08-5773-4, 2013

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009).

[18] Ladislav Peska, Krisztian Buza, Julia Koller. „Drug-Target Interaction Prediction: a Bayesian Ranking Approach.” Computer Methods and Programs in Biomedicine 152 (2017). https://t.co/kuTGwL8acc