Gradus Vol 5, No 2 (2018) 360-367
ISSN 2064-8014

# A TensorFlow rendszer és a mély tanulás

# TensorFlow system and deep learning

*Dr. Buzáné dr. Kis Piroska*[1]*

[1]Matematika és Számítástudományi Tanszék, Informatikai Intézet, Dunaújvárosi Egyetem, Magyarország

***Kulcsszavak:***
gépi tanulás
többrétegű mesterséges neurális háló
mély tanulás
TensorFlow rendszer

***Keywords:***
machine learning
multi layer neural network
deep learning
TensorFlow system

***Cikktörténet:***
Beérkezett 2018. augusztus 01.
Átdolgozva 2018. szeptember 04.
Elfogadva 2018. október 01.

***Összefoglalás***

*A mesterséges neurális hálók kutatása terén 2006-ban jelentős áttörést hoztak Hinton és kutatócsoportja eredményei. Elindult a neurális háló modellek harmadik nagy hulláma, a több rejtett rétegű hálók korszaka. Az ilyen modellek alkalmazásának elterjesztésére indult el a GoogleBrain projekt 2011-ben. Ennek első szakaszában, a DistBelief rendszerrel számos sikeres, jól ismert alkalmazás számára készítettek mély neurális háló modelleket. A DistBelief eredményei és tapasztalatai alapján fejlesztették ki a második szakaszban a TensorFlow rendszert, amely egy nyílt forráskódú szoftverkönyvtár gépi tanulási modellek létrehozására. A TensorFlow rendszer a többrétegű neurális háló modellek megalkotásán kívül széles körben alkalmazható más célokra is, ideértve a különféle algoritmusok és numerikus számítások implementálását.*

*Ez a munka néhány egyszerű példán keresztül bemutatja a TensorFlow rendszer szemléletét.*

***Abstract***

*The results of Hinton et al. lead to a notable breakthrough in the research of the artificial neural networks in 2006. The third great wave of neural networks research started at that time which is the era of neural networks with multiple hidden layers. GoogleBrain project was started to spread application of such models in 2011. In the first period, in deep neural network models were developed for numerous well-known applications with the DistBelief system. In the second period based on the results and experiences with DistBelief, TensorFlow system has been developed. TensorFlow is an open-source software-library for machine learning. However, while TensorFlow has been developed for wide application of deep neural networks, it is usable for great number of other purposes too, for example, implementation of various algorithms and numerical calculations.*

*The current work gives an introduction into the philosophy of TensorFlow system through some simple examples.*

*Kapcsolattartó szerző. Tel.: +36 25 551 673
E-mail cím: buzanedr.kis.piroska@uniduna.hu

# 1. Bevezetés

A GoogleBrain projekt 2011-ben indult azzal a céllal, hogy népszerűsítse a sokrétegű neurális háló modellek alkalmazását kutatási és ipari célokra [1]. A projekt első szakaszában a DistBelief rendszer készült el széles körű kutatási célokra. A GoogleBrain kutatócsoporttal szorosan együttműködve több mint 50 kutatócsoport alkotott mély neurális háló modelleket a DistBelief rendszer használatával, többek között a Google keresőrendszer, Google Fotók, Google Térkép és Utcakép, Google Fordító, YouTube számára. A DistBelief rendszerrel szerzett tapasztalatok és a 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 rugalmas, másrészt robosztus és hatékony legyen. A rugalmasság a kutatás számára fontos az új modellek gyors megvalósításához, valamint a velük való kísérletek könnyű kivitelezéséhez. Ugyanakkor a rendszer robosztus és hatékony, ezért a TensorFlow-val készített modellek a valós alkalmazások számára is megfelelnek.

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

# 2. A TensorFlow gráf

A TensorFlow rendszerben a számításokat egy irányított gráf írja le. Ebben a gráfban a csúcsok rendszerint egy-egy műveletet reprezentálnak. Mindegyik csúcsnak lehet nulla, egy vagy több inputja, ugyanígy nulla, egy vagy több outputja. Adatáramlás a gráf élei mentén történik. A gráf normál élei mentén áramló értékek tenzorok, tetszőleges dimenziójú vektorok. Egy-egy elem típusát a gráf létrehozásakor kell megadnunk. 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 létrehozásakor kell megadni.

A kliens programok a TensorFlow rendszerrel session létrehozásával kerülnek kapcsolatba. A session létrehozásakor a kezdeti gráf üres. A session létrehozásához a `Session` interface rendelkezésre bocsájt egy `Extend` metódust abból a célból, hogy a számítási gráfot további élekkel és csúcsokkal bővíthessük. A `Session` interface által szolgáltatott másik alapvető művelet a `Run`. Ez a művelet megkeresi a kiírandó output neveket és kiszámolja az értékeiket. A TensorFlow implementáció az egyes csúcsok közötti függőségi viszonyok figyelembevételével képes végrehajtani a műveleteket.

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 `Run` metódus lehetővé teszi egy tetszőleges részgráf végrehajtását és tetszőleges adat bevitelét és az adatkinyerést a gráf bármely éle mentén.

# 3. Szolgáltatások, optimalizálások, vizualizációk

A TensorFlow rendszer lehetővé teszi egy számítási gráf vagy részgráf tetszőleges számú végrehajtását, elősegítve ezzel a hasonló, ismételt számítási eljárások kényelmes kivitelezését. A neurális hálók súlyait kereső iteratív optimalizációs eljárások során tipikusan hasonló számításokat hajtunk végre egymást követően sokszor. Minden egyes tanító példára végig kell számolni a háló összes kapcsolatának új súlyait és minden neuron új torzítási értékeit [2]. Ez azt jelenti, hogy igen sokszor lényegében ugyanazon számítási műveletsort hajtjuk végre, esetenként más-más adatokkal. Hasonló a helyzet az ajánlórendszerekben [3] vagy hatóanyagok és farmakológiai támadáspontok közötti kapcsolatokat kereső [4] mátrix faktorizációs eljárások esetén is.

A TensorFlow-ban nagyszámú optimalizáló algoritmus, továbbá nevezetes, ismert gépi tanulási algoritmusok is rendelkezésünkre állnak, mint például a beépített automatikus gradiens kiszámító szolgáltatás.

A TensorFlow rendszer tartalmaz optimalizálást a számítási gráfban előforduló redundancia kiszűrésére, a memóriahasználatra, az adatbevitelre, egy-egy eljárás konkrét implementációjának kiválasztására (például mátrixszorzás számítható CPU-n vagy GPU-n).

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.

## 4. Programfejlesztés TensorFlow rendszerrel

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

A TensorFlow-val készített programok általában két részből állnak:

– számítási gráf felépítése,

– számítási gráf futtatása.

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:

$$a = tf.placeholder(tf.float32)$$

$$b = tf.placeholder(tf.float32)$$

$$szorzo\_node = a * b$$

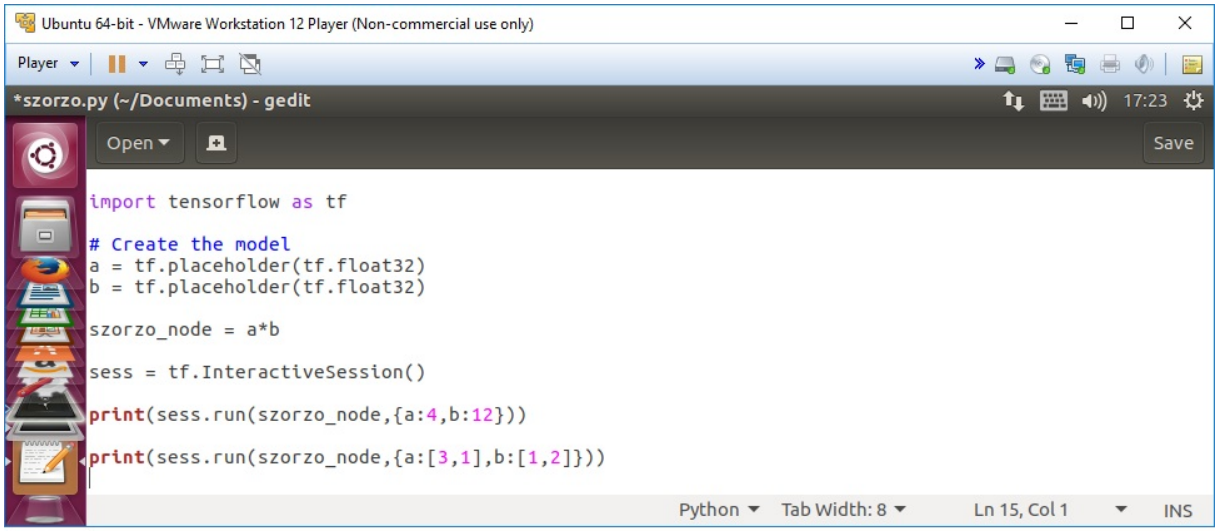
A példában két input paramétert szorzunk össze. Ezt a számítási gráfot többször is tudjuk alkalmazni, a szorzandók típusa is változhat:

$$print(sess.run(szorzo\_node, \{a : 4, b : 12\}))$$

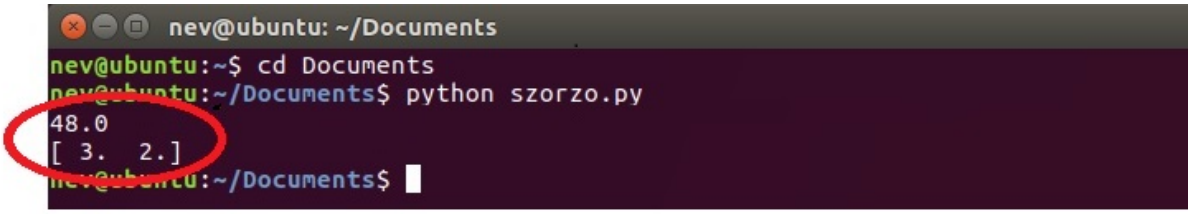
eredménye: 48.0

$$print(sess.run(szorzo\_node, \{a : [3, 1], b : [1, 2]\}))$$

eredménye: [3. 2.] A TensorFlow programunk az 1. ábrán, a futtatás eredménye a 2. ábrán látható.




*1. ábra. Szorzás a TensorFlow rendszerrel*




*2. ábra. A szorzás eredménye*

A gépi tanulás során tipikusan olyan modellre van szükségünk, amely – bizonyos korlátok között – tetszőleges inputot tud fogadni. A változók lehetővé teszik, hogy egy gráfhoz tanítható paraméterek tartozhassanak.

Egy gépi tanítás során tipikusan egy optimalizációs feladatot oldunk meg: például neurális hálók tanítása során olyan súlyokat keresünk, hogy az adott neurális hálózat a feladatát lehető legkisebb hibával oldja meg. Ehhez definiálnunk kell egy hibafüggvényt. Gyakori a négyzetes eltérés függvény, mint hibafüggvény használata.

A következő példában a modell kiértékeléséhez létre hozunk egy placeholdert, s használjuk a standard hibamodellt. Az $y$ placeholder egy olyan vektort reprezentál, amely a modell elvárt kimenetét tartalmazza az egyes tanítópéldányokra, miközben a $pelda\_modell$ vektor a modell tényleges kimenetet tartalmazza az egyes tanítópéldányokra.

Esetünkben a $pelda\_modell - y$ egy vektort ($negyzetes\_hiba\_vektor$ ) hoz létre, amelynek mindegyik eleme egy-egy tanítópéldány hibája, ezután a hibák négyzetösszegét vesszük, s eredményül egy skalárt kapunk:

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

$$negyzetes\_hiba\_vektor = tf.square(pelda\_modell - y)$$

$$hiba = tf.reduce\_sum(negyzetes\_hiba\_vektor)$$

A gépi tanulás lényege, hogy automatikusan megtaláljuk a modellparaméterek értékeit. Vegyük észre, hogy adott tanítóadatok mellett a hiba a modellparaméterek függvénye. A hibafüggvény optimumának megkeresése tehát a megfelelő modellparaméterek megtalálását eredményezi. A következő bekezdésben azzal foglalkozunk, hogyan tudjuk ezt megtenni.

A hibafüggvény minimalizálására a TensorFlow rendelkezésre bocsájt optimalizáló algoritmusokat, amelyek mindegyik változó kismértékű változását idézik elő. Ezek közül a legegyszerűbb a gradiens lejtő (gradient descent). A TensorFlow azonban automatikusan elvégzi a számításokat a hibahatárok figyelembevételével:

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

$$train = optimizer.minimize(hiba)$$

# 5. Példa gépi tanulásra

Lássunk egy egyszerű mintapéldát gépi tanulásra a TensorFlow segítségével. Kézírású számjegyek felismerésére készítünk egy modellt. Adott egy adathalmaz – MNIST – amely kézírással írt számjegyek képeit tartalmazza. Mindegyik kép $28 \times 28$ pixel. Ezt a négyzet alakú képet átalakíthatjuk vektor formájúra, amely vektor $28 \times 28 = 784$ komponenst tartalmaz. Az átalakításnál csak arra kell ügyelnünk, hogy mindegyik kép esetén ugyanazt a módszert alkalmazzuk. Minden egyedhez (képhez) egy címke is tartozik a képen látható számjegynek megfelelően. 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 tanításra, további részét pedig arra használjuk, hogy becsüljük, hogy mekkora hibával dolgozik a modellünk új adatokra.

## 5.1. Az osztályba sorolás

Ha egy egyedről el kell döntenünk, hogy több különböző osztály közül melyibe milyen valószínűséggel tartozik, használható a `softmax` függvény, mivel a `softmax` megad egy diszkrét valószínűségeloszlást az egyes osztályokba tartozási valószínűségekre vonatkozóan.

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.

Itt a `softmax` aktiválási vagy kapcsolási függvényként szolgál. Röviden úgy foglalhatjuk össze a `softmax` tevékenységét, hogy egy $x$ inputhoz kiszámítja az egyes osztályokba tartozás súlyait, azután megadja az osztályokba tartozási valószínűségeket.

## 5.2. A modell definiálása a TensorFlow rendszerben

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

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 megadtuk, hogy a képeket tartalmazó adathalmazunk minden egyede egy 784 dimenziós vektor. A `None` itt azt jelenti, hogy a beolvasandó egyedek száma bármennyi lehet.

A modellünkhöz szükségünk van súlyokra ($W$) és torzítási (bias, $b$) 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 implementálása mindössze egyetlen sor:

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

Az, hogy a modell ilyen egyszerűen felírható, annak köszönhető, hogy maga a TensorFlow nagyon rugalmas 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.

## 5.3. 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, minél kisebb a hiba, annál jobb a modellünk. Nagyon általános a keresztentrópia hibafüggvény használata:

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

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:

$$cross\_entropy{=}tf.reduce\_mean(-tf.reduce\_sum(y\_{*}tf.log(y), reduction\_indices{=}[1]))$$

Mindazt, 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 visszaterjesztés – algoritmust.

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

$$train\_step = tf.train.GradientDescentOptimizer(0.05).minimize(cross\_entropy)$$

Ezután a modellünket interaktív session-ba tehetjük:

$$sess = tf.InteractiveSession()$$

Először inicializálni kell a változókat:

$$tf.global\_variables\_initializer().run()$$

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

$$for\ in\ range(2000):$$

$$batch\_xs, batch\_ys = mnist.train.next\_batch(100)$$

$$sess.run(train\_step, feed\_dict = \{x : batch\_xs, y\_ : batch\_ys\})$$

## 5.4. 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.

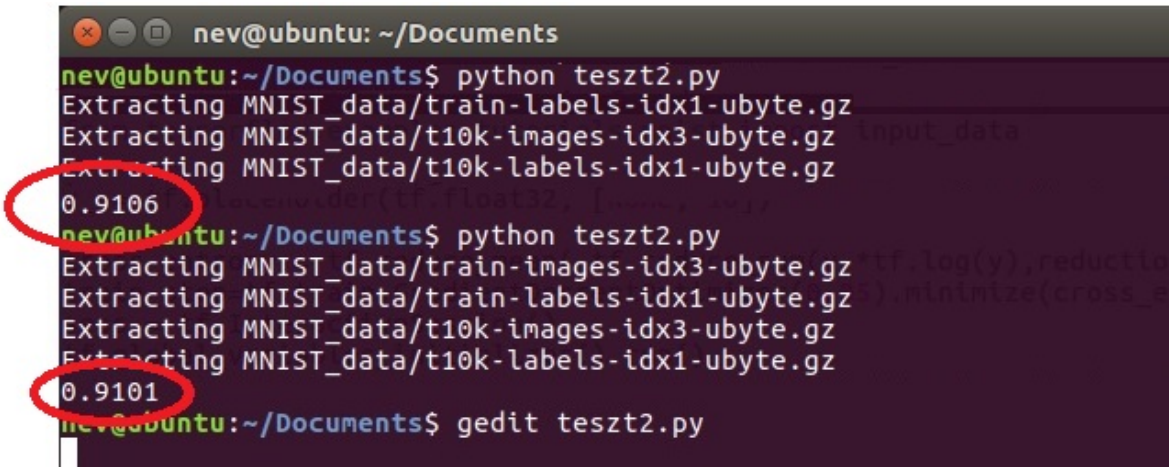
$$pontossag = tf.reduce\_mean(tf.cast(helyes\_joslas, tf.float32))$$

Modellünk pontosságát a teszt adatainkra nézve kiszámíttatjuk és kiíratjuk:

$$print(sess.run(pontossag, feed\_dict = \{x : mnist.test.images, y\_ : mnist.test.labels\}))$$

A program két futtatásának eredménye a 3. ábrán látható.

Amint a 3. ábra mutatja, a példaként megalkotott modell a konkrét esetben 91% pontosságot produkált. Általában a hozzá hasonló nagyon egyszerű modellekkel a tapasztalatok szerint 90%-ot meghaladó, 91%-93% pontosság érhető el.




*3. ábra. A bemutatott eljárás működése*

A TensorFlow rendszerben megtalálható eljárások segítségével azonban jelentős javulás érhető el. A pontosság 3.000 tanító lépés után elérhet legalább 96%-ot, sőt 20.000 lépés után közel 100%-ot, amint a 4. ábra mutatja.


*4. ábra. A modell pontossága a tanító lépések függvényében.*

## 6. A TensorFlow könyvtár

A példában szereplő modell nem igényelt sok kódot, azonban ennél lényegesen bonyolultabb, több kódot tartalmazó modelleket is alkothatunk. 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 ezen könyvtár anyagait adatállományok

kezelésére, bevitelére, modellek tanításra vonatkozó magas szintű támogatásra. A TensorFlow tutorialjaiból kezdők és haladók egyaránt sok ötletet meríthetnek.

A TensorFlow használatához több szoftvercsomagra is szükség van. Az operációs rendszer (Ubuntu Linux) és a megfelelő komponensek installálása linuxos előismereteket, időt és munkát igényel. A TensorFlow rendszer használatához Adam Geitgey összeállított egy megfelelően konfigurált és szoftverekkel ellátott virtuális gépet, amely az alábbi honlapon érhető el:
`https://medium.com/@ageitgey/try-deep-learning-in-python-now-with-a-fully-pre-configured-vm-1d97d4c3e9b`

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

# 7. Összefoglalás

Sokrétegű neurális hálókat szerteágazóan használnak, például a beszédfelismerésben, a számítógépi látásban, megjelenítésben, a robotikában, az információ kinyerésben, a számítógépek elleni támadások felderítésében, és az agykutatásban [5]. A TensorFlow rendszer roppant rugalmas, algoritmusok nagyon széles körének megvalósítására alkalmas. 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.

# Hivatkozások

[1] 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.

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

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

[4] Ladislav Peska, Krisztian Buza, Julia Koller. "Drug-Target Interaction Prediction: a Bayesian Ranking Approach." Computer Methods and Programs in Biomedicine. `https://www.journals.elsevier.com/computer-methods-and-programs-in-biomedicine` `https://t.co/kuTGwL8acc`

[5] Regina Meszlényi, Krisztian Buza, and Zoltán Vidnyánszky. "Resting state fMRI functional connectivity-based classification using a convolutional neural network architecture." arXiv preprint arXiv:1707.06682 (2017).