Gépi Tanulás Előadás 2

Felügyelt Tanulás: Mesterséges Mély Neurális Hálózatok

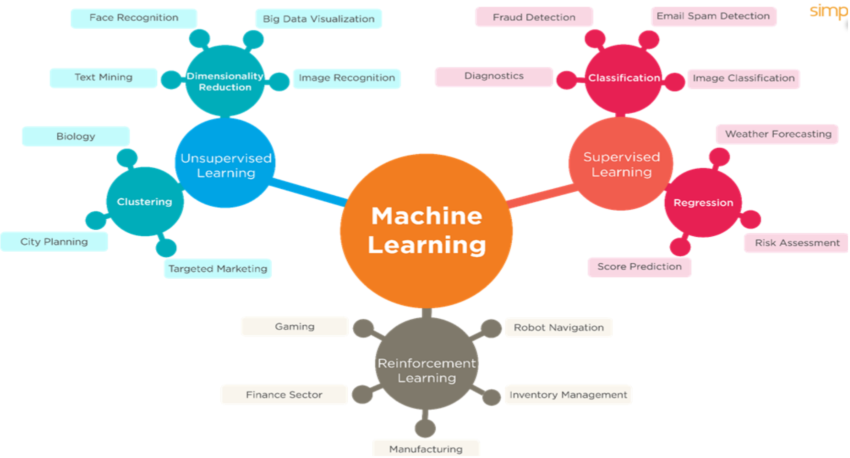
Milacski Zoltán Ádám[1]

[1]Eötvös Loránd Tudományegyetem
Programozáselmélet és Szoftvertechnológiai Tanszék
srph25@gmail.com

2019. május 8.

Gépi Tanulás

Felügyelt, Felügyeletlen és Megerősítéses



- Múlt órán: Felügyelt, KNN, RF;
- Ma: Felügyelt, Mesterséges Mély Neurális Hálózatok (ANN, DNN);
- Jövő órán: Felügyeletlen, Megerősítéses.

- Adottak az $(x_n, y_n), n = 1, \ldots, N$ tanítópárok, keressük az optimális θ^* paraméterbeállítást az $f(\theta, \cdot)$ paraméteres leképezéshez úgy, hogy $f(\theta^*, x_n) \approx y_n$ (közelítsük az $x_n \mapsto y_n$ leképezést):

$$\min_\theta L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^{N} \ell\Big(\underbrace{f(\theta, x_n)}_{\hat{y}_n}, y_n\Big),$$

ahol $\ell(\hat{y}_n, y_n)$ a hibafüggvény és θ a paramétervektor.
Becslés ($f(\theta^*, x_n)$ kiszámítása adott θ^*-ra) nagyon gyors!
Tanítás (minimalizáló θ^* megkeresése) nagyon lassú!
Általában: nemkonvex optimalizálási feladat, NP-nehéz a θ^* globális optimumot megtalálni.

- Jó hírek: működik, de csak ha...
  - ... N elég nagy! Az y_n-ek összegyűjtése drága humán munka... (UL?),
  - ... $\ell(\hat{y}_n, y_n)$, $f(\theta, x_n)$ és az optimalizálási módszer megfelelően vannak megválasztva! Nehéz humán munka, sok kísérlet... (UL?),
  - ... θ közel van θ^*-hoz! Egyelőre nem tudjuk bizonyítani, de megy, így feltehetően igaz... (θ^* egyébként is túl mohó és túltanulásra vezet...).

Paraméteres leképezés és Optimalizálási módszer szorosan összetartozik.
Összehasonlítás:

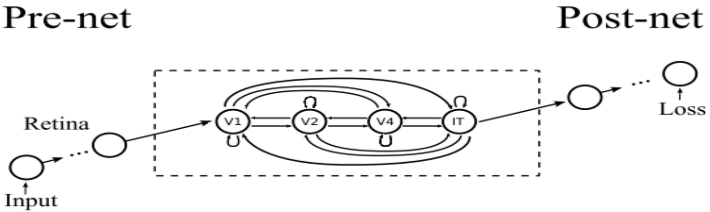
Módszer	θ	$f(\theta, x_n)$	$\min_\theta$	Megjegyzés
KNN	(x_n, y_n), $n = 1, \ldots, N$	$\sum_{n=1}^{N} w_n^{KNN} y_n$	nincs, θ^* ismert	nem tömörít, lassú, glob. opt. θ^*, K-ra érzékeny
RF	legj. vágó vált., t küszöbök	$\sum_{n=1}^{N} w_n^{RF} y_n$	mohó, faépítés	tömörít, gyors, nódusra opt., össz. szubopt. θ^*
DNN	w_j súlyok	$h(\sum_{j=1}^{J} w_j x_{n,j} + b)$ többsz. össz. fv.	gradiens-módszer	tömörít, gyors, össz. lok. opt., glob. opt θ^*?

Mesterséges Mély Neurális Hálózatok

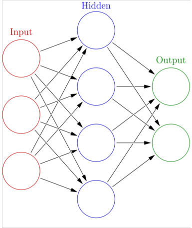
- Mesterséges Mély Neurális Hálózat (DNN): Összetett függvény alakú paraméteres leképezés:

$$f(\theta, x_n) = g_K\left(\theta_K, g_{K-1}\left(\cdots g_2\left(\theta_2, g_1(\theta_1, x_n)\right)\cdots\right)\right)$$

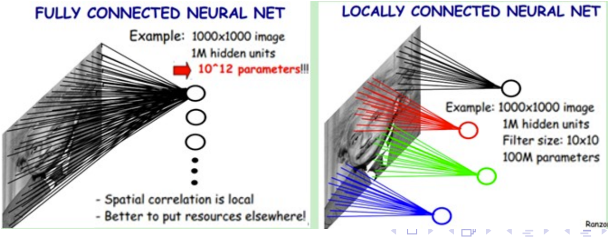
ahol $\theta = \{\theta_1, \ldots, \theta_K\}$ a parametérek (súlyok). Nemkonvex, mert θ_i és θ_j $(i \neq j)$ szorzata megjelenik $f(\theta, x_n)$-ben.
Réteg: $x_n^{(k)} = g_k(\theta_k, x_n^{(k-1)})$
Hálózat (előreterjesztés): $f(\theta, x_n)$.
A rétegek alacsonyabb szintű leíró változókat kombinálnak össze magasabb szintűekké az összetett függvény alak miatt.
A főemlősök látórendszerét utánozza:



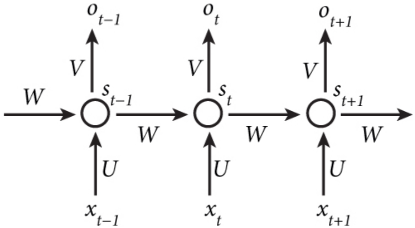
- Rétegek:
  - Sűrű (Teljes Konnektivitású, 2D vektorok): $x_n^{(k)} = h_k(W_k x_n^{(k-1)} + b_k)$



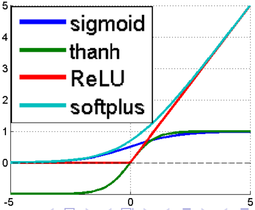
  - Konvolúciós (Lokális Konnektivitású, 4D képek): $x_n^{(k)} = h_k(W_k * x_n^{(k-1)} + b_k)$



- Rétegek:
  - Rekurrens (3D idősorok): $x_{n,t}^{(k)} = h_k(W_k^{hh} x_{n,t-1}^{(k)} + W_k^{xh} x_{n,t}^{(k-1)} + b_k)$



- Nemlinearitások: h_k elemenkénti nemlineáris függvény a rétegekben.
  - sigmoid: $h_k(z) = \frac{1}{1+e^{-z}}$,
  - tanh: $h_k(z) = \tanh(z)$,
  - relu: $h_k(z) = \max(0, z)$,
  - softmax: $h_k(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{l} e^{z_i}}$.



Mesterséges Mély Neurális Hálózatok

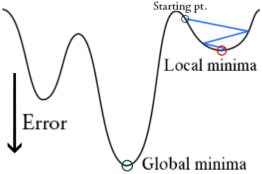
Optimalizálási Módszer: Gradiens-módszer

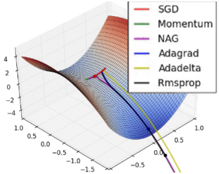
Keressük θ^*-ot! $f(\theta, x_n)$ deriválható θ szerint, így $L(\theta, x_n, y_n)$ is.

- Sztochasztikus Gradiens-módszer (SGD): Negatív gradiens a lokális optimum fele mutat, lépjünk picit ebbe az irányba! Legyen $\theta_0 \sim \mathcal{N}(0, 0.001^2)$, ezután:

$$\theta_{l+1} := \theta_l - \alpha \frac{\partial L(\theta, x_n, y_n)}{\partial \theta}\bigg|_{\theta=\theta_l}.$$

  Tanulási ráta (lépésköz): α, pici szám, pl. 0.001.
  Visszaterjesztés: $\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$, automatikus gépi deriválással (láncszabály...).
  Minibatch: (x_n, y_n) párok véletlen részhalmaza minden lépésben.





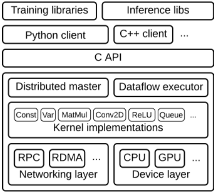
Elakadhat nyeregpontokban!

- Kvázi-Newton módszerek: ki tudnak mozdulni nyeregpontokból adaptív tanulási rátákkal, pl. RMSProp, Adadelta, Adam.

Használjunk GPU-t gyorsasághoz: a tenzorműveletek zavarbaejtően párhuzamosak. SGD megfelelő kevés memóriához.
Probléma: GPU programozás túl alacsony szintű (lassú és hibákkal teli fejlesztés)...
Megoldás: kódoljunk magas szinten (pl. Python-ban) és fordítsuk le alacsony szintű GPU kódra!
Ehhez speciális szoftvereszközök szükségesek:

- CUDA, OpenCL: Alacsony szintű, GPU kód
- Tensorflow, Theano, CNTK, PyTorch: Közepes szintű (Back-end), szimbolikus előreterjesztés, automatikus szimbolikus deriválás, fordítás GPU kódra és meghívás konkrét numerikus értékekkel
- Keras: Magas szintű (Front-end), nemlinearitások, rétegek, hibafüggvények, gradiens-módszerek
- OpenAI Gym: Megerősítéses Tanulás framework
- Hyperopt: Hiperparaméter keresés
- Sacred: Kísérletek logolása és reprodukciója
- Elasticluster: Elosztott számítások felhőben

- Fejezzük ki algoritmusunkat szimbolikus formában, számítási gráf felépítésével
- Építés fázis:
  - Tensor: típusos többdimenziós tömb (statikus típus, dimenzió, méret).
  - Operation (op): kap nulla vagy néhány tenzort, számol velük, majd visszaad nulla vagy néhány tenzort (szimbolikus gradiense implementálva van).
  - Variable: állapotok tárolása több hívás (végrehajtás) közöttm (tf.assign op).
  - Placeholder: bemenetek, kijelölik a ‘feed’ műveleteket.
- Végrehajtási fázis:
  - Session: műveleteket eszközre (CPU vagy GPU) helyezi, metódusokat ad a végrehajtásukhoz, tenzorokat ad vissza numpy ndarray-ként.
  - Fetch: műveletek kimeneteinek kinyerése, gráf végrehajtása.
  - Feed: művelet kimeneteinek lecserélése konkrét tenzor értékre.

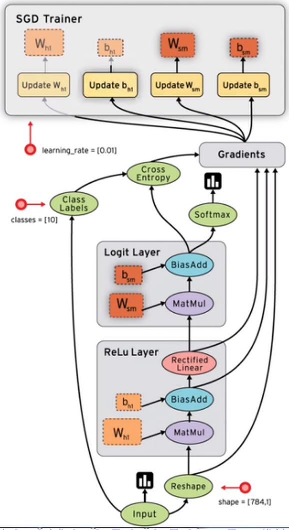



```python
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
  print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))

# output:
# [array([ 14.], dtype=float32)]
```

Hogyan csináljunk Mesterséges Mély Neurális Hálózatot Tensorflow-ban?

Name	Math	Tensorflow
Bemenet, kimenet	$x_n, y_n, \quad n = 1, \ldots, N$	tf.placeholder
Paraméter	θ	tf.Variable
Előreterjesztés	$f(\theta, x_n)$	tf.nn ops
Hibafüggvényion	$L(\theta, x_n, y_n) = \frac{1}{N}\sum_{n=1}^{N} l\left(f(\theta, x_n), y_n\right)$	tf.losses ops
Szimbolikus gradiens	$\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$	tf.gradients op
Inicializáció	θ_0	tf.random_normal_initialize
Gradiens-módszer	$\theta_{l+1} := \theta_l - \alpha \left.\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}\right\rvert_{\theta=\theta_l}$	tf.train.Optimizer
Tanít, tesztel, becsül	$x_n := X_n, y_n := Y_n$	tf.Session.run(), fetch, fee

Előny: Az $\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$ szimbolikus gradiens automatikusan számolható a tf.gradients op által (láncszabály többszöri ismétlésével a számítási gráfon, ahol minden op-nak ismert a gradiense; ezt rendkívül nehéz lenne papíron levezetni). Ez nagyban egyszerűsíti a kísérletezgetést (csak az Előreterjesztést kell cserélgetni, ami könnyű)!
Hátrány: Nehéz debug-olni.



Probléma: Tensorflow-ban rendre ugyanazokat a számítási részgráfokat (rétegek, nemlinearitások) kell megadni, újra implementálgatni őket felesleges. Ezek magasabb absztrakciós szintet képviselnek.
Megoldás: Keras magasabb szinten rendszerezi őket objektum-orientáltsággal (osztályok és öröklődés)!

- Könnyű és gyors prototípus gyártás (felhasználóbarát, moduláris, bővíthető).
- Sok beépített réteg osztály, amik kombinálhatóak is.
- Használhat TensorFlow-t, CNTK-t vagy Theano-t a háttérben (keras.backend).
- Csak Python (nincs szükség egyéb konfigurációs fájlokra, így bővíthető).
- Egyszerű keras.models.Model metódusok tanításra, tesztelésre, becslésre: fit(), evaluate(), predict().

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense, Activation
model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
classes = model.predict(x_test, batch_size=128)
```

Algorithm 1: The layered thresholding algorithm.

Assuming two layers for simplicity, Algorithm 1 can be summarized in the following equation

$$\mathbf{\Gamma}_2 = \mathcal{P}_{\beta_2}\left(\mathbf{D}_2^T \mathcal{P}_{\beta_1} \left(\mathbf{D}_1^T \mathbf{X} \right) \right).$$

Comparing the above with Equation (1), given by

$$f(\mathbf{X}, \{\mathbf{W}_i\}_{i=1}^2, \{\mathbf{b}_i\}_{i=1}^2) = \mathrm{ReLU}\left(\mathbf{W}_2^T \cdot \mathrm{ReLU}\left(\mathbf{W}_1^T \mathbf{X} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right),$$

one can notice a striking similarity between the two.

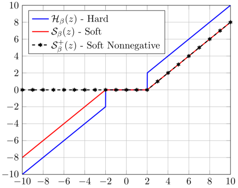


Figure 3: The thresholding operators for a constant $\beta = 2$.

- Tételek: ekvivalencia konvolúciós ReLU hálók és ritka reprezentáció között (utóbbira sok bizonyított tétel van, pl. globálisan optimális θ^*-ra). De lesz több tétel is hamarosan...
- Jobb rétegek: Kapszulák (Hinton) magasabb rendű invariáns változókkal.
- Jobb fejlesztői eszközök: imperatív programozás, debug-olás...

Ajánlott Irodalom

- Online Kurzusok
  - Vincent Vanhoucke kurzusa: Tensorflow
  - Andrew Ng kurzusa: Gép Tanulás Bevezető
  - NVIDIA kurzusa: Szoftvereszközök
  - Geoffrey Hinton kurzusa: Elmélet
  - Andrej Karpathy kurzusa: Konvolúciós Hálók
  - Stephen Boyd kurzusa: Konvex Optimalizálás
  - Georgia Tech kurzusa: Megerősítéses Tanulás
- Forráskódok
  - Keras útmutató
  - Keras példák
  - Keras GitHub (haladó)
- Tudományos Cikkek
  - Google Scholar (haladó): Yann LeCun, Joshua Bengio, Geoffrey Hinton, Ilya Sutskever, Christian Szegedy, Alex Krizhevsky, Andrew Ng, Quoc Le, Vincent Vanhoucke, Diederik Kingma

Alkalmazások

Youtube videók