Neurális hálózatok alkalmazása spam szűrésre

András Mamenyák[1] and Roland Bamli[1]

[1]Mérnök informatikus (BSc) szakos hallgató, Debreceni Egyetem

2014. május 15.

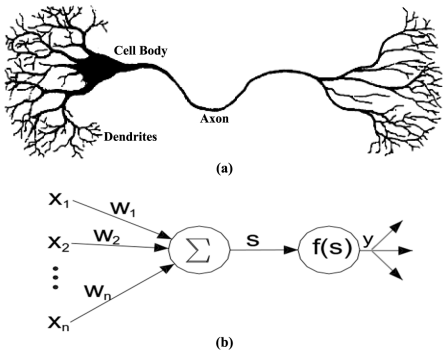
1. Neurális hálózatok

1.1. A neurális hálózatok kialakulása

A neurális hálózatok a mesterséges inteligencia egy típusa, amelyet az állatok központi idegrendszere, különösen az agy ihletett, amely képes a tanulásra, a mintafelismerésre is. Megalkotásához biológiai ismeretekre és az idegsejt működésének pontosabb megismerésére volt szükség. Ez csak a 20. században valósult meg. Az első neuron modelt 1947-ben alkotta meg McCullock és Pitts, az első mesterséges neuront pedig Rosenblatt 1958-ban. A neurális hálózatok egy ígéretes, új tudományterület, mely Webos 1974-es "back propagation" algoritmusa és annak 1986-os újra felfedezése után indult igazán fejlődésnek.

1.2. A mesterséges neuron felépítése, működése

Egy mesterséges neuron, mint a biológiai, több bemenettel és egy kimenettel rendelkezik (1. ábra). Egy általános neuron működése szerint meghatározza a bemenetek súlyozott összegét és ezen végrehajt valamilyen nem lineáris leképezést. Ez utóbbit nevezik aktivációs, transzfer vagy aktiváló függvénynek. A végeredmény pedig a neuron kimeneti jele. Egy másik változat a lineráris összegzést megvalósító neuron, amikor nem történik lineáris leképezés.



1. ábra. A biológiai neuron (a) és a mesterséges neuron (b) összehasonlítása

A 1. ábrán a neuron bemeneteit x_i jelöli, a kimeneti jel pedig y. Először a bemenetek súlyozott összegei kerülnek meghatározásra:

$$s = \sum_{i=0}^{n} W_i \cdot x_i = W^T \cdot x$$

Abban az esetben, ha a neuron lineáris összegzést valósít meg, ezzel már meg is kaptuk a kimeneti jelet:

$$y = s = W^T \cdot x$$

Nem lineáris esetben szükség van még a nem lineáris leképezésre. Ebben az esetben a neuron kimeneti jele a következő:

$$y = f(s) = f(W^T \cdot x)$$

ahol $f(s)$ az aktivizációs függvény. Erre a célra a négy leggyakrabban használt függvény a lépcső- vagy szignumfüggvény, a „telítéses lineáris" függvény, a tangens hiperbolikusz függvény és a szigmoid függvény.

Használnak egy másik elterjedt neuron típust is a RBF (Radial Bass Function) hálózatokban. Ennél a típusnál nincs lineáris összegzés, az összes bemenet az aktivizációs függvénybe kerül, mely több bemenet esetén több változós függvény lesz.

1.3. A neuron hálózatok felépítése

A neuronokból álló hálózatokat nevezzük neurális hálózatoknak. Ezekben minden neuron ugyanolyan, vagy hasonló műveleteket végez, a többi neurontól függetlenül, lokálisan. Tehát ezek a hálózatok olyan információfeldolgozó eszközök, amelyek párhuzamos, elosztott működésre, tanulásra képesek. Általában irányított gráffal reprezentáljuk őket. A neuronok a gráf csomópontjai, míg a gráf élei a kimenetek és bemenetek közötti kapcsolatot reprezentálják. Megvalósíthatók szoftveresen, hardveresen, vagy a kettő kombinációjaként is.

A neuronok három fajtáját különböztetjük meg:

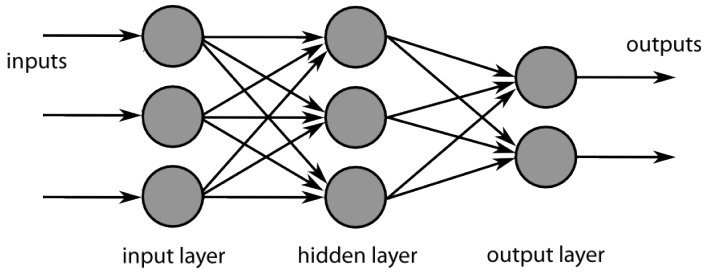
1. **bemeneti neuronok:** Egy bemenetű, egy kimenetű, buffer jellegű neuronok, jelfeldolgozó feladatuk nincs. Bemenetük a hálózat bemenete, kimenetük más neuronok meghajtására szolgál.

2. **rejtett neuronok:** Ezek a neuronok végzik a jelfeldolgozást. Kimenetük és bemenetük is más neuronokhoz csatlakozik.

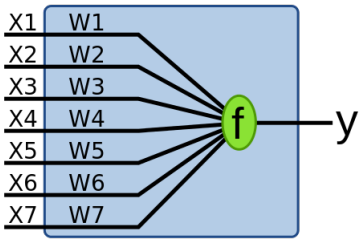
3. **kimeneti neuronok:** A környezet felé továbbítják kimenetüket.

A neuronokat álltalában típusa alapján rétegekbe szervezzük. Ennek megfelelően beszélhetünk bemeneti rétegről, retjett réteg(ek)ről és kimeneti rétegről.

A neuronhálózatokat az egyes neuronok közötti összeköttetési rendszer alapján két fő csoportba sorolhatjuk. Beszélhetünk előrecsatolt hálózatokról (3. ábra) és visszacsatolt hálózatokról.



2. ábra. Előrecsatolt neuron hálózatok felépítése.



3. ábra. Egy Perceptron hálózatbeli neuron működése.

1.4. Előrecsatolt (Feedforward) hálózatok

Az előrecsatolt hálózatokban a jel csak egy irányba terjedhet, a bemeneti rétegtől, a rejtett rétegeken át, a kimeneti réteg felé. Nincsen visszacsatolás (hurok), a neuronok kimenete nincs hatással arra rétegre, amelyben találhatóak. Széles körben használják minta felismerésre.

1.4.1. Single-layer Perceptron hálózatok

A Perceptron hálózatok a legegyszerűbb neurális hálózatok, melyekben a bemeneti réteg neuronjai közvetlenül a kimeneti réteghez csatlakoznak. A hálózat a Perceptron algoritmus szerint működik, melyet 1957-ben dolgozott ki Frank Rosenblatt. Jellemzői:

- **képes megtanulni egyszerű minták felismerését**
- **kétrétegű:** az input rétegnek csak elosztó szerepe van, Perceptronok valójában csak egy rétegben vannak
- **folyamatos értékű és bináris inputtal egyaránt működhet**
- **offline módon tanul, időciklusokban működik**
- **egy perceptron azt dönti el, hogy egy input minta két osztály közül melyikhez tartozik.**
- **felügyelt tanulást végez**

Az algoritmus a működése a során a bemeneti jelek (x_i) és a hozzájuk tartozó súlyok (w_i) alapján számolja ki a hálózat kimenetét. Az alábbi képlet szerint:

$$n = f(\sum_{i=0}^{m} x_i w_i),$$

ahol

$$f(x) = \begin{cases} 1, & \text{ha } x > t \\ 0, & \text{egyébként} \end{cases}$$

A hálózat inicializálásakor meg kell adnunk a súlyok kezdőértékét és a t (threshold) értékét. A súlyoknak vagy 0-t, vagy véletlenszerűen választott kis érteket érdemes adni. Ezen kívül értéket kell adnunk még r-nek (learning rate). Ez egy 0 és 1 közé eső szám lehet. Amennyiben túl nagyra választjuk, a peceptron ingadozni fog a megoldás körül.

Miután megvan a hálózat kimenete (n), a várt kimenetből (z), kiszámoljuk a hibát (e):

$$e = z - n$$

Ezután a korrigálást:

$$d = r * e$$

Végül frissítjük a súlyokat:

$$w_i = w_i + d * x_i$$

Amikor a hiba értéke 0, nem változtatunk a súlyokon. A tanulási folyamat akkor ér véget, ha a egyik train bemenetre sem kell változtatni a súlyokon, vagyis a hálózat mindegyikre a várt eredményt adja.

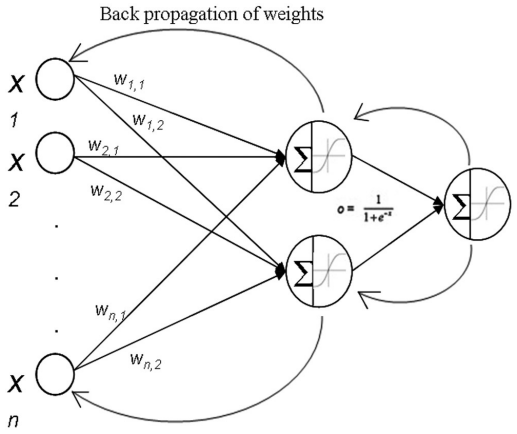
1.4.2. Többrétegű előrecsatolt hálózatok

Ezekben a hálózatokban több számítást végző réteg található meg. Minden egyes neuron kimenete a következő rétegre van kapcsolva. A többrétegű hálózatoknál több tanuló algoritmus közül is választhatunk, a legelterjedtebb a back-propagation algoritmus.

1.4.3. Back-propagation algoritmus

A back-propagation, teljes nevén „backward propagation of errors", magyarul hiba-visszaterjesztési eljárás, egy tanulási algoritmus, melyet gyakran használnak a neurális hálózatokban. Ez egy felügyelt tanulási módszer, melynek szüksége van egy nagy adatbázisra a bemenetekkel és a kívánt kimenetekkel. Alkalmazása az előrecsatolt hálózatoknál a leghasznosabb. Használatához meg kell követelnünk, hogy a neuron hálózat réteges felépítésű, a neuron átviteli függvénye pedig deriválható legyen.A tanítás alapgondolata, hogy az elvárt és a számított kimenetek eltérését, mint a háló súlyaitól függõ hibát értelmezzük és ezen, a súlyok terében értelmezett hibafüggvényen hajtunk végre egy minimális pont keresést. Az algoritmusban a tanulás lényegében a hátrafelé terjedés folyamata, mely során minimalizálni kell az elvárt és a tényleges output vektor közötti négyzetes eltérést, Euklideszi távolságot.

Működése alapján két fázisra lehet osztani, terjedésre (propagation) és a súlyok frissítésére. A terjedés során a jel mind előre, mind hátra a szinapszisok és a neuronok szintjén lokális információk alapján terjed. A súlyok frissítése a neuron kimenetére visszaérkezett jel alapján történik (4. ábra).



4. ábra. A back-propagation algoritmus működése.

1.5. Visszacsatolt (Recurrent) hálózatok

Akkor nevezünk egy neurális hálózatot visszacsatoltnak, ha a topológiáját reprezentáló irányított gráf tartalmaz hurkot. Ez esetben beszélhetünk globális és lokális visszacsatolásról.

- **elemi visszacsatolás (recurrent connections):** egy réteg egy neuronjának kimenete közvetlenül egyik saját bemenetére van visszacsatolva,
- **laterális visszacsatolás (lateral connections):** valamely réteg(ek) neuronjainak kimenetei ugyanazon réteg neuronjainak bemeneteire kapcsolódnak, de nem értjük ide a neuron önmagára való visszacsatolását.
- **rétegek közötti visszacsatolás (inter-layer connections):** több réteget tartalmazó hurkot hoznak létre a gráfon.

2. Spam szűrés

2.1. Bevezetés

Az elektronikus üzenetküldés nagyon fontos részévé vált a mindennapi életnek. Egy olyan részévé, mely segítségével ingyen és egyszerűen küldhetünk üzeneteket a világ bármely pontjára. Sajnos ezeknek az üzeneteknek egy része kéretlenül érkezik, ezeket nevezzük spam üzeneteknek. Bizonyára mindenkit zavarnak az ilyen üzenetek, de különösen a vállalatok számára okoznak növekvő problémát, illetve jelentős anyagi kárt. A spam üzenetek következményei:

- **idő veszteség**
- **tárhely veszteség**
- **fontos üzenet elvesztése**
- **hálózat túlterhelése**

- **rosszindulatú programok**

A spam szűrési techinkákat működésük alapján így csoportosíthatjuk:

- **lista alapú szűrők:** Blacklist, Whitelist,
- **tartalom alapú szűrők:** Word-Based Filters, Heuristic Filters, Bayesian Filters
- **egyébb szűrők:** DNS Lookup System, Challenge/Response System

2.2. Neurális hálózatok alkalmazása

A neurális hálózatokat nem olyan olyan széles körben elterjedtek, mint például a Bayes-szűrés. Ennek oka lehet, hogy a Bayes-szűrés elég egyszerű, mégis nagyon jó hatékonysággal működik. A neurális hálózatoknak nagyobb az erőforrás igénye, és több mintára van szüksége.

A hálózat működéséhez szükség van minta üzenetekre. Ezekről tudnunk kell, hogy spamek-e, a tanulás folyamán ugyanis ezek lesznek a várt kimenetek (1 ha spam, 0 ha nem). A bemeneti paramétereket az üzenetek szövegéből kell kiszámítanunk. Ezek alapján hasonlítja majd össze a neurális hálózat az üzeneteket és ez alapján próbálja spam és nem spam csoportokra bontani.

3. A program futtatása és a kimenet értelmzése

A program fordítása és futtatása az alábbi módon végezhető el:

```
andras@G53SW:~/Programs/Neural$ cmake .
andras@G53SW:~/Programs/Neural$ make
andras@G53SW:~/Programs/Neural$ ./neural

Start training.
16 normal and 24 spam messages
Number of training iterations: 100000
End training.

Start testing.
4 normal and 18 spam messages


Input: SPAM
Output: 1

Input: SPAM
Output: 0.999999

Input: NORMAL
Output: 0.0877907

Input: NORMAL
Output: 9.26975e-07
```

```
Input: SPAM
Output: 0.998536

Input: SPAM
Output: 0.997332

Input: SPAM
Output: 1

Input: SPAM
Output: 0.879897

Input: SPAM
Output: 1

Input: SPAM
Output: 1

Input: NORMAL
Output: 0.891214

Input: SPAM
Output: 0.999947

Input: SPAM
Output: 0.999998

Input: NORMAL
Output: 0.00230046

Input: SPAM
Output: 1

Input: SPAM
Output: 1

Input: SPAM
Output: 1

Input: SPAM
Output: 0.501791

Input: SPAM
Output: 0.736173

Input: SPAM
Output: 0.865628

Input: SPAM
```

```
Output: 1

Input: SPAM
Output: 0.99985

End testing.
```

A neurális hálót

```
const int Niter = 10000;
```

iteráción keresztül „tanítattjuk” 24 spam és 16 nem spam üzenettel, a bemenet az üzenetek paraméterei, a kimenet a hozzájuk tartozó érték. Ez alapján állítódnak be az egyes neuronokhoz tartozó súlyok, amik kezdetben random értékek voltak. Ezután az éles tesztben ráengedjük a 4+18 független üzenetet és amint a fenti kimenetben látható, csupán 1 esetben kaptunk egy spam-re bizonytalan értéket (0.501791). Ebből jól látható, hogy a neurális hálónk megfelelően működik, megtanulta milyen paraméterek jellemzik a spam és nem spam üzeneteket.

4. Konklúzió

A projekt során a ma népszerű kutatási területnek számító neurális hálózattal foglalkoztunk. Megismerkedtünk az alapvető felépítésével és működésével ezekenek a gépi tanulást végző rendszereknek, valamint sikerült a spam mintát helyesen felismerő programot készítetnünk.

A program letölthető a `https://github.com/mamenyaka/Neural` github tárolóból.