UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA APLIKOVANEJ INFORMATIKY

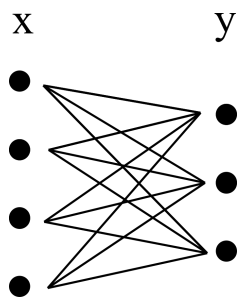
Úvod do Umelej Inteligencie
Cvičenie 9 - Perceptrón

November 16, 2022

9. PERCEPTRÓN

Budeme programovať neurónovú sieť - jednovrstvový spojitý perceptrón - ktorá sa naučí klasifikovať vstupné dáta do viacerých tried (kategórií).

Model:



Medzi každým vstupom x_j a výstupom y_i je spojenie s váhou w_{ij} - na začiatku sú tieto váhy náhodné, pričom počas učenia sa iteratívne vylepšujú. K n vstupným neurónom pridávame ešte jeden špeciálny x_{n+1}, tzv. *bias*, ktorý je vždy rovný 1.

Počítanie výstupu:

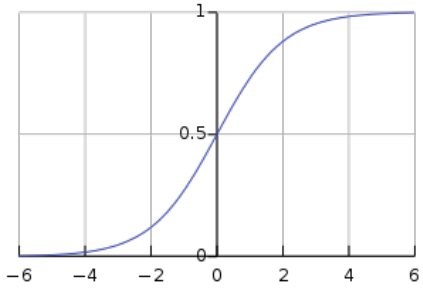
Výstup z perceptrónu sa počíta ako váhovaný súčet všetkých vstupov, ktorý sa preženie cez tzv. aktivačnú funkciu:

$$\boldsymbol{y_i} = f\Big(\sum_{j=1}^{n+1} w_{ij}.x_j\Big), \qquad x_{n+1} = 1$$

Pokiaľ si vezmeme vektor $\boldsymbol{x}$ ako (stĺpcový) vektor všetkých vstupov a biasu $\boldsymbol{x} = (x_1, ..., x_n, x_{n+1})^T$, ďalej vektor $\boldsymbol{y}$ ako vektor všetkých výstupov $\boldsymbol{y} = (y_1, ..., y_m)^T$ a maticu $\boldsymbol{W}$ o veľkosti $(m \times (n+1))$, ktorá obsahuje všetky váhy w_{ij}, tak potom vieme vypočítať všetky výstupy naraz v jednom kroku, pomocou jednoduchého vzorca:

$$\boldsymbol{y} = f(\boldsymbol{W}.\boldsymbol{x}),$$

V tomto prípade berieme, že funkcia f sa aplikuje na každý prvok výsledného vektora samostatne. Ako aktivačnú funkciu f budeme používať logistickú sigmoidu, ktorá má navyše jednoduchú deriváciu:



$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{x}}} \qquad \boldsymbol{f}'(\boldsymbol{x}) = \boldsymbol{y}.(\mathbf{1} - \boldsymbol{y})$$

Počítanie chyby:

Na trénovanie máme pripravenú trénovaciu množinu, ktorá obsahuje n-rozmerné vektory $\boldsymbol{x}$, a k nim prislúchajúce požadované výstupy - m-rozmerné vektory $\boldsymbol{d}$. Výpočet chyby je analogický minulému cvičeniu - opäť ide o regresnú chybu, akurát potrebujeme sumovať cez všetky súradnice našich bodov, t. j. pre jeden bod:

$$e = \sum_{i=1}^{m} (d_i - y_i)^2 = ||\boldsymbol{d} - \boldsymbol{y}||^2$$

Celková chyba E je jednoducho súčtom (alebo priemerom) chýb e na všetkých príkladoch z trénovacej množiny.

Úprava váh:

Na minimalizáciu chyby používame *gradient descent* metódu, pri ktorej upravujeme váhy w_{ij} v opačnom smere ako je derivácia chyby e, z čoho nám vypadne vzorec:

$$w_{ij} := w_{ij} + \alpha \cdot (d_i - y_i) \cdot f'(w_i x) \cdot x_j$$

kde α je konštanta, tzv. rýchlosť učenia. Keď doplníme deriváciu sigmoidy do tohoto vzorca (namiesto f'), dostaneme:

$$w_{ij} := w_{ij} + \alpha \cdot (d_i - y_i) \cdot y_i \cdot (1 - y_i) \cdot x_j$$

A opäť., aby sme nemuseli počítať všetky váhy w_{ij} osobitne, aj toto sa dá spraviť na jeden krok pomocou maticového zápisu:

$$\boldsymbol{W} := \boldsymbol{W} + \alpha \cdot (\boldsymbol{\delta} \times \boldsymbol{x}), \qquad \delta_i = (d_i - y_i) \cdot y_i \cdot (1 - y_i)$$

Všimnite s, že na násobenie vektorov $\boldsymbol{\delta} \times \boldsymbol{x}$ sa používa tzv. outer product, t. j. výsledkom je matica.

Trénovanie:

Samotné trénovanie prebieha v iteráciach - epochách: v jednej epoche sa prejde všetkými príkladmi z trénovacej množiny, a pre každý z nich sa vypočíta výstup siete, jej chyba, a upravia sa váhy. Počas každej epochy sa príklady vyberajú v náhodnom poradí, aby sme sa vyhli nežiadúcim efektom pri učení. Končíme po určitom počte epoch, alebo keď celková chyba epochy E klesla dostatočne nízko.

Algorithm 1 Neural network training pseudocode:

```
1: procedure TRAINING
2:     Weight matrix W initialization
3:     while stopping criterion is not met do
4:         E ← 0
5:         for each input x and its target d in shuffle(inputs) do
6:             y ← compute output for x
7:             e ← compute error of y
8:             E ← E + e
9:             adjust weight matrix W using x, y and d
```

Dáta:

Budeme rozpoznávať číslice, ktoré sú "nakreslené" na 4×7 pixeloch, napr.: (v dátach je čierna reprezentovaná jednotkou a biela nulou):

Takýchto 10 číslic je nakopírovaných N-krát (parameter v programe, default je 10x), a následne sú mierne zašumené - každý "pixel" sa s malou pravdepodobnosťou prehodí. Takto získame bohatší dataset, na ktorom budeme trénovať rozpoznávanie (zašumených) číslic.

Vstupný vektor $\boldsymbol{x}$ bude mať teda $4 \times 7 = 28$ zložiek (plus bias). Pri rozpoznávaní budeme číslice klasifikovať do jednej z desiatich tried 0 až 9, pričom budeme používať tzv. one-hot kódovanie výstupu: výstupný vektor $\boldsymbol{y}$ (alebo $\boldsymbol{d}$) má 10 zložiek - všade sú nuly, iba tam, do ktorej kategórie $\boldsymbol{x}$ patrí, je jednotka. Napríklad číslica nula bude mať výstup $\boldsymbol{d}$=(1,0,0,0,0,0,0,0,0,0), sedmička zas $\boldsymbol{d}$=(0,0,0,0,0,0,0,1,0,0), atď.
Keď budete sieť učiť, tak ako výstup siete nedostanete hodnoty presne nula a jedna, ale niečo medzi. "Víťazný" neurón je potom ten s najvyššou hodnotou.
Napríklad výstup $\boldsymbol{y}$= (0.4,0.2,0.1,0.1,0.3,0.2,0.9,0.1,0.2) teda budeme chápať tak, že sieť rozpoznala sedmičku. Máte pripravené dva vstupné súbory: v súbore *numbers.in* rozpoznávate číslice tak, ako je popísané vyššie. V súbore *odd_even.in* sú tiež číslice, no rozlišujete ich iba do dvoch tried - párne/nepárne (taktiež pomocou one-hot kódovania). Rozdelenie na párne a nepárne je ľahšia úloha - sieť by sa ju mala naučiť za niekoľko desiatok epoch. Rozpoznávanie čísel je ťažšie - učenie môže trvať aj niekoľko stoviek epoch.

Úloha (1b):

Do pripravenej kostry programu *perceptron.py* doprogramujte nasledovné pomocné funckie, plus hlavný trénovací cyklus:

- *initialize_weights()* - inicializuje maticu váh *self.W* (treba zadať iba veľkosť matice)
- *sigmoid(x)* - vypočíta hodnotu sigmoidy, pričom *x* je číslo (skalár). Pokaľ funkciu spravíte využitím *np.exp*, tak *x* môže byť aj vektor a funkcia vráti opäť vektor.
- *compute_output(x)* - vypočíta výstup siete pre vstupný vektor *x*. **Výstupom je výsledný vektor y.**
- *compute_error(d, y)* - pomocou požadovaného výstupu *d* a vášho výstupu *y* (oba sú vektory) vypočíta (regresnú) chybu *e*. Výstupom je teda jedno číslo.
- *train(num_epoch)* - využitím predchádzajúcich funkcií natrénuje neurónovú sieť. Sieť sa bude učiť *num_epoch* epoch.

Numpy:

Ako ste si všimli, v tejto úlohe sa veľa hráme s vektormi a maticami. Preto budeme používať python-ovskú knižnicu *numpy*, vďaka ktorej je počítanie s vektormi a maticami veľmi jednoduché a pohodlné. Vďaka *numpy* teda netreba riešiť počítanie $\boldsymbol{y}$, e, $\boldsymbol{\delta}$, či $\boldsymbol{W}$ po zložkách vo for-cykloch, všetko sa dá vypočítať na jeden riadok pomocou vektorov. Rýchlokurz *numpy* je aj v kóde, venujte pozornosť napr. rôznym možnostiam násobenia dvoch vektorov a matíc.
Poznámka pre špekulantov: použitie knižníc ako TensorFlow, Theano, Keras, Lasagne, atď. je zakázané.