

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

NEURÓNOVÁ SIETĚ S EXPONENCIÁLNOU
VRSTVOU
BAKALÁRSKA PRÁCA

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

NEURÓNOVÁ SIET' S EXPONENCIÁLNOU
VRSTVOU
BAKALÁRSKA PRÁCA

Študijný program: Aplikovaná Informatika
Študijný odbor: Aplikovaná Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: RNDr. Kristína Malinovská, PhD.

1 Východiská práce

1.1 Teória

1.1.1 Teória neurónových sietí a anatómia nervového systému

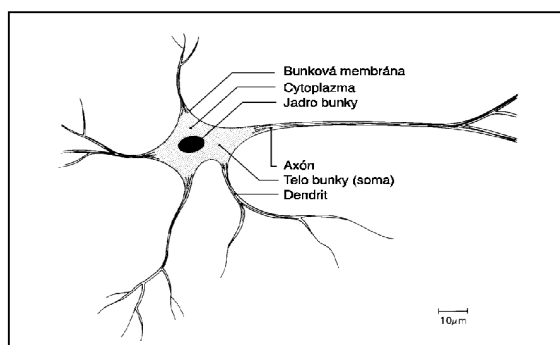
Teória neurónových sietí, sa snaží pochopiť nervový systém a vysvetliť jeho správanie sa vzhľadom k spracovaniu informácií v nervových bunkách, na základe neurofyziológických poznatkov. Hlavnou funkciou nervového systému, je riadiť organizmus a spracovávať informácie. Tento prenos je zabezpečený pomocou neurónov.

1.1.2 Neurón

Priemerný ľudský mozog sa skladá priemerne z 10^{12} neurónov, kde jeden neurón má v priemere 10^3 až 10^5 prepojení na iné neuróny. Je schopný nielen prijímania a reagovania na informácie zo všetkých vnemov, ale aj generalizácií (vyvolať reakciu na podnet, ktorá bola naučená iným podnetom).

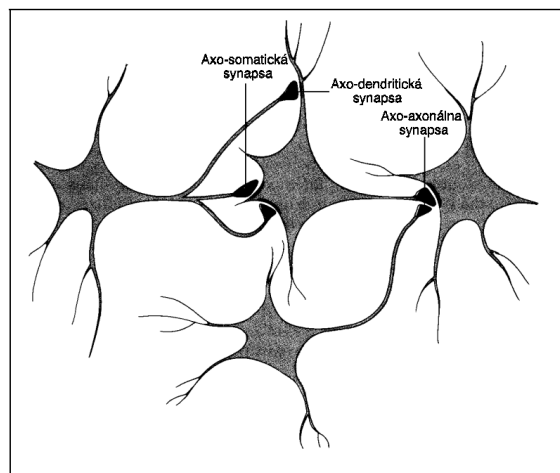
Samotný neurón sa skladá z tela nazývaného soma a niekoľkých výbežkov, ktoré je možné rozdeliť na 2 typy :

- dendrity, ktoré z informatického hľadiska predstavujú vstup
- axón, ktorý predstavuje výstup



Obr. 1: Nervová bunka (neurón).

Neuróny vytvárajú funkčné spojenia medzi axómom jednej bunky a membránou druhej. Tieto tesné spoje nazývame Synapsy. Väčšinou spájame axóm jedného neurónu s časťou druhého, podľa čoho ich rozdelujeme na axo-axonálne, axo-dendritické alebo axo-somatické. Jeden neurón obsahuje priemerne 7000[DA.05] takýchto spojení, pričom každé z nich má určitú silu ktorou ovplyvňuje axómový signál. Tieto sily (váhy) sa upravujú v procese učenia sa.



Obr. 2: Rôzne druhy synáps.

Pre zjednodušenie si môžeme proces neurónu predstaviť nasledovne. Aby nastala aktivácia (Akčný potenciál), musí napätie v neuróne presiahnuť prah excitácie, ktorý je zvyčajne okolo $-60mV$. Neurón po dendritoch prijíma kladné vzruchy (elektrické signály), ktoré pokračujú do jadra neurónu. Môžu pochádzať buď z viacerých synáps naraz, alebo z tej istej synapsy ak je aktivovaná rýchlo za sebou. V some sa tieto vzruchy sčítavajú aj so zvruchmi opačného napätia, ktoré zostali z predošlej aktivácie. Podľa veľkosti úspešného prekonania prahu, neurón spustí niekoľko akčných potenciálov ktoré putujú po axóne ďalej. Následne sa napätie v neuróne vráti na kľudové hodnoty, pri čom vznikajú už spomínané zvruchy záporného napätia.

1.1.3 Perceptrón

Umelý neurón sa pokúša podstatovo priblížiť tomu biologickému.

Skúsme si definovať takýto umelý neurón ktorý vymyslel Frank Rosenblatt v roku 1957 nazývaný perceptrón. Zoberme y je výstup z neurónu. Vstupné vzruchy z dendrit budeme reprezentovať pomocou vektora ktorý budeme označovať ako

$$x = (x_1, x_2, \dots, x_n)^T$$

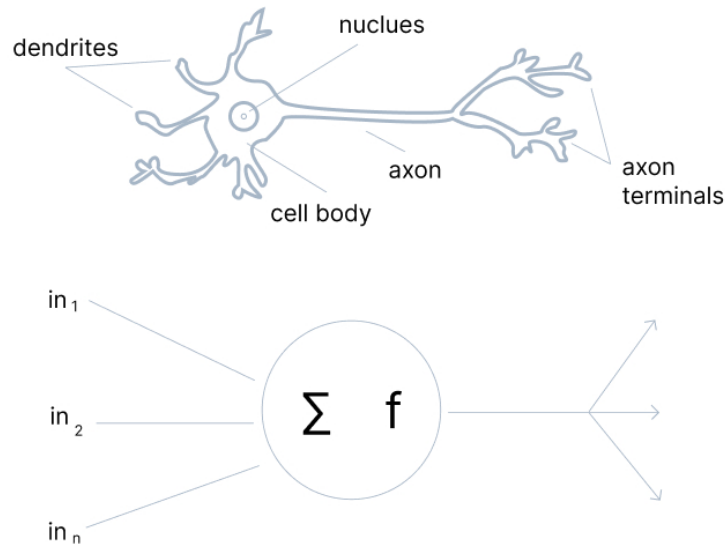
kde T označuje že vektor je transponovaný.

Silu na konkrétnej synapse, budeme nazývať váhou. Každá dentrita má svoju váhu, takže nám vznikne vektor váh, ktorý budeme označovať nasledovne

$$w = (w_1, w_2, \dots, w_m)^T$$

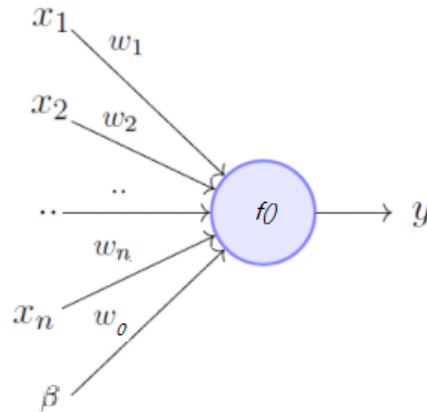
Za predpokladu že $f()$ je aktivačná funkcia, výstupnú aktivitu potom môžeme vypočítať nasledovne :

$$y = f(net) = f(w^T x)$$



Obr. 3: Porovnanie biologického a umelého neurónu

Aby sme mohli aktivačnú funkciu posúvať, pridáme takzvaný bias ku každému vstupu. Budeme ho označovať β . Bias bude pre všetky vstupy rovnaký a bude mať svoju váhu označovanú w_0 . Zvyčajne sa biasu priraduje inicializačná hodnota 1.

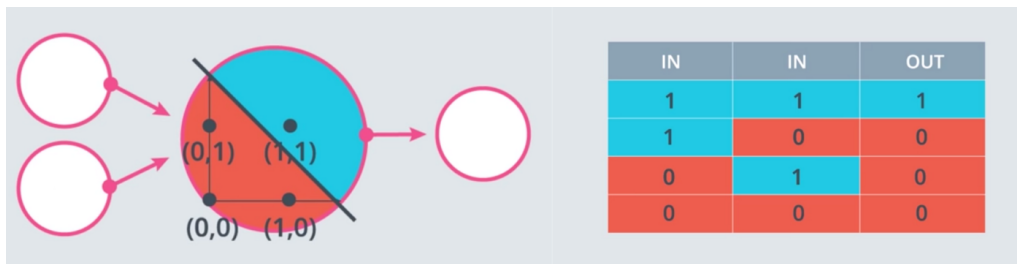


Obr. 4: Jednoduchá vizualizácia perceptrónu s biasom.

Aktuálny perceptrón vieme naučiť napríklad AND funkciu za pomoci klasifikácie. Ako aktivačnú funkciu použijeme Binary-Step. Zdefinujeme si prah označovaný θ , ktorý bude musieť výstup z funkcie prekonať aby bol aktivovaný. Práh nám rozdelí priestor do viacerých tried. Toto môžeme nazvať klasifikáciou. Našu funkciu budeme definovať nasledovne:

$$y = 1 \quad \text{if } \beta + \sum_{i=1}^n w_i x_i \geq \theta$$

$$y = 0 \quad \text{if } \beta + \sum_{i=1}^n w_i x_i < \theta$$

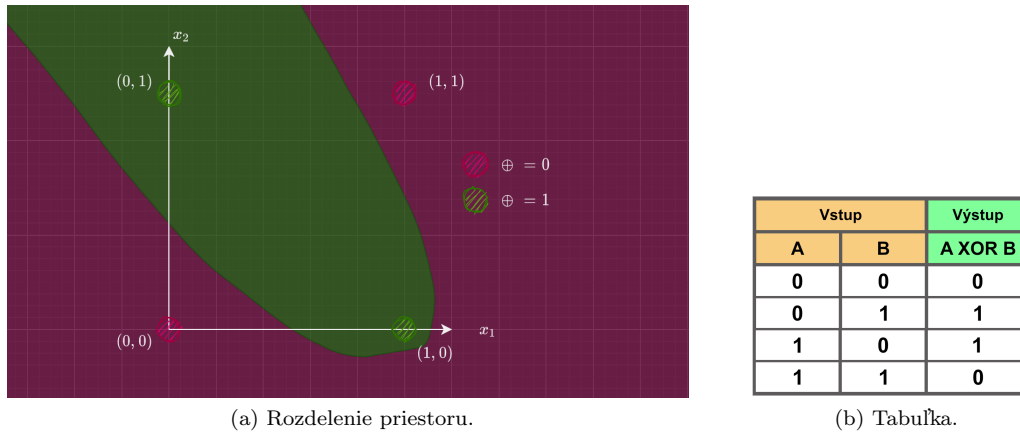


Obr. 5: Perceptrón pri konfigurácii kde $\beta = 1$, $w_1 = 1$, $w_2 = 1$, $w_0 = -1,5$, $\theta = 0$

1.1.4 XOR

Funkcia XOR je taktiež ako AND jedna zo základných logických funkcií. V dnešnej dobe, pri neúronových sieťach používa ako jeden zo základných problémov, ktoré musí byť daný model schopný vyriešiť.

Perceptrón je jeden z počiatočných modelov, takže je pochopiteľné že týmto "testom" neprešiel a dlhšiu dobu boli neurónové siete považované zlým univerzálnym výpočtovým prostriedkom. Problém je vtom že síce perceptrón je schopný binárnej separácií, ale na správna klasifikácia podľa funkcie XOR musí byť aj nelineárna. Aj tieto obmedzenia perceptrónov, popísali Minsky a Papert v roku 1969.



(a) Rozdelenie priestoru.

(b) Tabuľka.

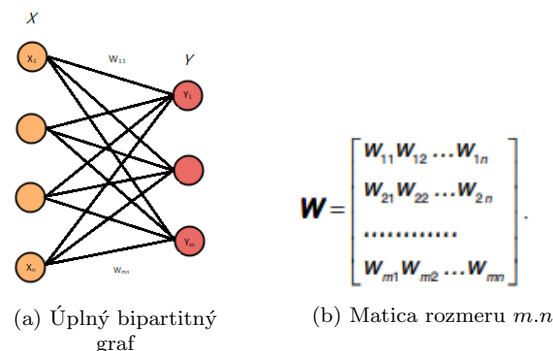
Obr. 6: Funkcia XOR

1.1.5 Back-propagation

V roku 1986 publikoval Rumelhart, Hinton a Williams nový pohľad na umelé neuróny. Nebrali sa do úvahy ako logické prepínače, ale skorej ako analógové elementy ktoré obsahovali spojitú vstupno-výstupnú funkciu. Zaviedli pravidlo spätného šírenia chýb ktoré efektívne upravuje váhy nazývaný (*error back – propagation* v skratke často referencovaný ako *backprop*). Algoritmus ešte nie je dostatočný na to, aby sa naučil riešiť ľubovlnú úlohu, ale model už zvládne mnoho úloh ktoré nie je možné vyriešiť za pomoci lineárnej separácie, ako aj spomínaný XOR. Táto metóda má však obmedzenia. Je aplikovateľná iba na perceptróny ktoré majú 3 a viac vrstiev. Taktiež tieto neuróny nemôžu byť prepojené "každý s každým", ale iba smerom dopredu na ďalšiu vrstvu. V prvá úroveň takejto siete predstavuje vstupy a posledná predstavuje výstup. Nazývame ich Multi-Layer Perceptron.

1.1.6 Definícia váh ak nasledujúca vrstva obsahuje viacero neurónov

Zatiaľ sme si definovali váhy iba pre perceptrón, ktorý na výslednej vrstve obsahuje iba 1 neurón. Ak by naša sieť pozostávala z n vstupných neurónov a m výstupných neurónov, dali by sa váhy vizualizovať pomocou bipartitného grafu, kde by množiny by boli jednotlivé vrstvy. V praxi ich budeme reprezentovať maticou o veľkosti $m.n$.



(a) Úplný bipartitný graf

(b) Matica rozmeru $m.n$

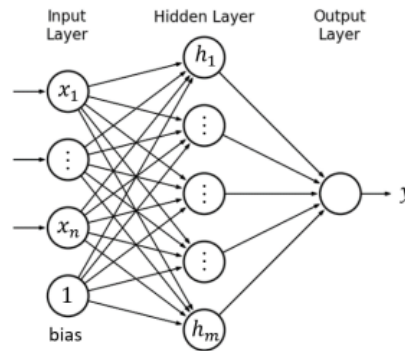
Obr. 7: Vizualizácia váh

1.1.7 Multi-Layer Perceptrón

Multi-Layer Perceptrón, tzv. MLP, sú schopné sa učiť na príkladoch a následne generalizovať nové vstupy. Môžu sa skladať z ľubovoľného počtu skrytých vrstiev, ktoré môžu mať rôzny počet neurónov. Hodnoty pre jednotlivé vrstvy počítame podobne ako pri jednoduchom perceptróne, avšak musíme vždy začať na vstupnej vrstve a pokračovať iteratívne aby sme sa dostali k výsledku. Definujme MLP, s n rozmerným vstupom, jednou skrytou vrstvou ktorá má m neurónov reprezentované vektorom $h = (h_1, h_2, \dots, h_m)$, výstupnou vrstvou ktorá má 1 neurón. Váhy W , sa skladajú z matice w^{hid} medzi vstupnou a skrytou vrstvou a vektorom w^{out} medzi skrytou a výstupnou vrstvou. Na vstupnej vrstve budeme pridávať aj bias. Uvažujeme že neuróny na skrytej vrstve majú určitú aktivačnú funkciu f_{hid} a neuróny na výstupnej vrstve majú určitú aktivačnú funkciu f_{out} . Potom platí :

$$\begin{aligned} net_i^{hid} &= \sum_{j=1}^n w_{ij}^{hid} \cdot x_j \\ h_i &= f_{hid}(net_i^{hid}) \\ net^{out} &= \sum_{i=1}^m w_i^{out} \cdot h_i \\ y &= f_{out}(net^{out}) \end{aligned}$$

(a) vzorce na výpočet hodnôt

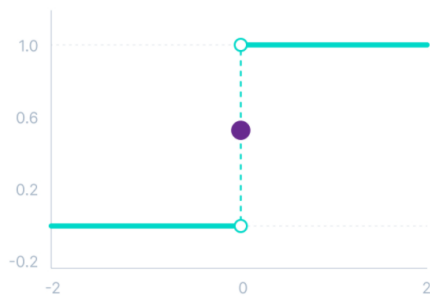


(b) vizualizácia

1.1.8 Aktivačné funkcie

Aktivačná funkcia má za úlohu rozhodnúť či bude neurón aktivovaný alebo nie.

- Binary Step - túto funkciu sme použili pri AND. V skratke nám vráti 1 alebo 0 podľa toho či bola prekonaná daná prahová hodnota. Spadá pod lineárne funkcie a nie je možné ju použiť na problémy, kde je nutná klasifikácia na viac ako 2 triedy.



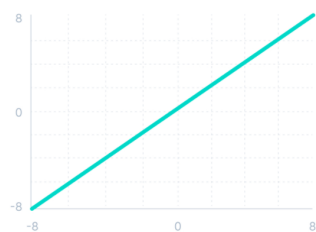
(a) graf

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

(b) funkcia

Obr. 8: Binary Step

- Lineárna funkcia - nijak neovplyvní výsledok neurónu. Má však svoje využitie pri MLP, ak narábame s dátami kde sa y pohybuje v intervale $(-\infty, +\infty)$. Vtedy ju použijeme na výstupnej vrstve. Nemôžeme ju však použiť na všetkých neurónoch, lebo by nám vrstvy skolabovali do jednej lineárnej.



(a) graf

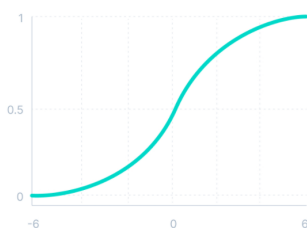
$$f(x) = x$$

(b) funkcia

Obr. 9: Lineárna

Nasledujúce funkcie už budú nelineárne. Účelom nelineárnej aktivačnej funkcie je zaviesť nelinearitu do výstupu neurónu. Naš model sa dokáže naučiť a vyhodnocovať aj zložitejšie úlohy aj preto, že je možné spiatočnej propagácii chýb, pretože derivácia týchto funkcií priamo súvisí s gradientom.

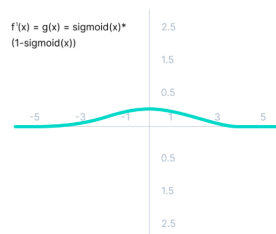
- Sigmoid - zobrazuje ľubovoľné reálne číslo na intervale $(0, 1)$. Používa sa pri modeloch, ktorých výstup je odhad pravdepodobnosti alebo pri binárnej klasifikácii. Má tvar písmena S, takže neobsahuje žiadne skoky, alebo drastické zmeny hodnôt. Jej nevýhodou je, že nie je symetrická vzhľadom na 0 a jej derivácia naberať strašne malé hodnoty mimo intervalu $(-3, 3)$, čo sťažuje tréning siete a robí ho nestabilným. Tento jav "Vanishing gradient problem"



(a) graf

$$f(x) = \frac{1}{1 + e^{-x}}$$

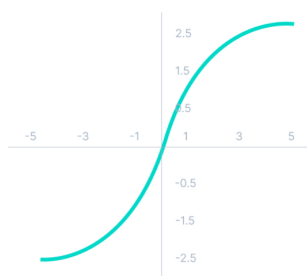
(b) funkcia



(c) derivácia

Obr. 10: Sigmoida

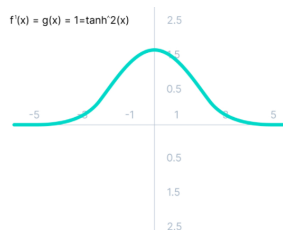
- Tanh - je veľmi podobná sigmoide. Rozdiel je v tom že jej obor hodnôt je v intervale $(-1, 1)$, pri čom je súmerná podľa 0. Vieme pomocou nej povedať či výsledok je záporný, neutrálny alebo kladný. Zvyčajne sa používa na skrytých neurónoch, lebo mapuje dáta okolo 0, čo uľahčuje učenie na nasledujúcej vrstve. Nevýhodou je že tak isto ako sigmoida, trpí na Vanishing gradient problem.



(a) graf

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

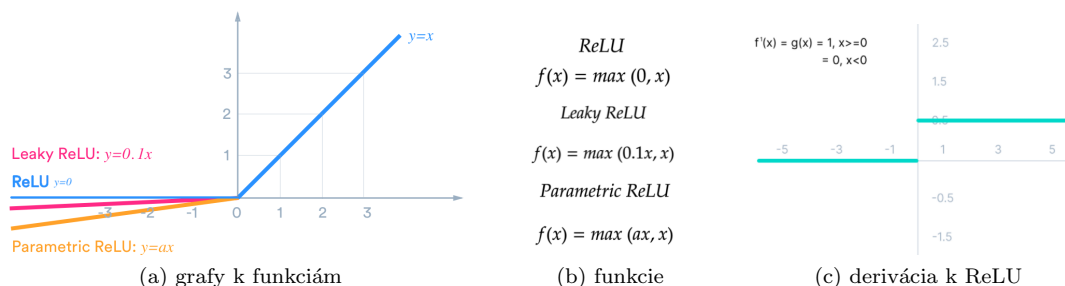
(b) funkcia



(c) derivácia

Obr. 11: Tahn

- ReLU - aj keď sa môže na javiť lineárna, opak je pravdou. Vieme vypočítať jej deriváciu, takže ju môžeme využívať pri backpropagácii. Je používaná vo väčšine dnešných modelov. Definujeme ju ako $\max(0, x)$, takže má jednoduchší výpočet ako Sigmoida či Tahn a netrpí Vanishing gradient problem-om ako spomínané funkcie. ReLU je jednoduchšia výpočtovo aj tým, že nie všetky neuróny sa aktivujú a gradient sa rýchlo približuje ku globálnemu minimu. Na druhú stranu nám vzniká problém mŕtvych neurónov, kvoli tomu že derivácia má pre zápornú stranu hodnotu 0, čo znamená že neurón sa už nikdy neupraví. Situácia mŕtvych neurónov zvyčajne nastáva, ak je learning rate príliš vysoký alebo má bias priveľkú negatívnu hodnotu. Toto sa dá riešiť cez úpravou funkcie na takzvanú Leaky ReLU. Vtedy sa počíta maximum z 0.1 násobku x a samotného x ju zapíšeme ako $\max(0.1x, x)$. Derivácie pre záporné hodnoty je rovná 0.1. Ak by tento problém neprestáva, môžeme násobok ľubovoľne zvýšiť, všeobecne ho budeme označovať ako parameter a . Táto verzia sa nazýva tzv. Parametric ReLU = $\max(ax, x)$. V tomto prípade je podobne, ako pri Leaky, derivácia rovná a .



Obr. 12: ReLU

1.1.9 Rozdelenie dát

Dáta pre neurónové siete máme ukladané vo formáte (x, d) , kde x je vstupný vektor a d je požadovaný výstup. Množinu týchto dát rozdelujeme na 3 podmnožiny nasledovne:

- Trénovacie dáta - A_{train} - tieto dáta, sa budú používať na učenie NS, v každej epoche sa vypočíta pre ne hodnota ktorú budeme označovať y . Následne sa pomocou backprop-u upraví váhy podľa rozdielu d a y .
- Validačné dáta - A_{valid} - táto množina má za úlohu aby sme sieť nepretrénovali a optimalizovali jej parametre, niekedy sa prelínajú s trénovacou množinou
- Testovacie dáta - A_{test} - dáta z testovacej množiny ostávajú počas celého procesu učenia siete a optimalizácie skryté, použijú na záverečné ohodnotenie, ako veľmi dobre je natrénovaná sieť schopná generalizácie

Dáta sa rozvoľujú náhodne v ľubovoľnom pomere. Zaužívaný je napríklad pomer 70:20:10 Trénovacie:Validačné:Testovacie

1.1.10 Adaptácia a úprava váh na MLP

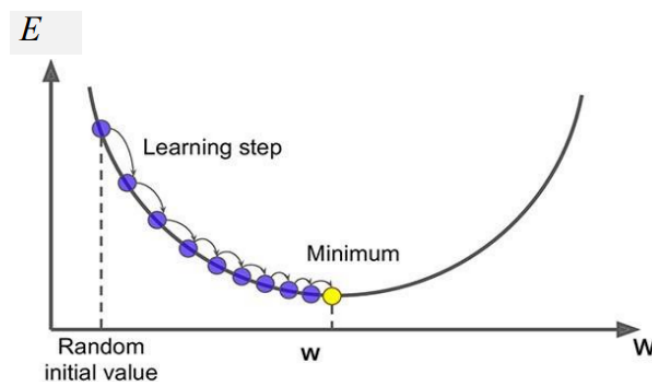
Adaptácia alebo aj učenie siete, je hľadanie takých koeficientov, aby medzi požadovanou hodnotou a hodnotou ktorú pre konkrétny vstup vypočítala naša sieť bola čo najmenšia odchylka pre akúkoľvek trénovaciu množinu údajov.

Uvažujme že $A_{train} = \{(x^1, d^1), (x^2, d^2), \dots, (x^p, d^p), \dots, (x^P, d^P)\}$ kde x^p je vstupný vektor a d^p je požadovaný výstup. Zdefinujme si error function pre MLP:

$$E = \frac{1}{2} \sum_{i=0}^P (d^p - g(w \cdot x^p))^2$$

kde g je aktivačná funkcia

Ďalej budeme pokračovať pomocou gradačného zostupu. Táto metóda upravuje váhy tak, aby chyba E dosiahla svoje minimum.



Obr. 13

Gradient chybovej funkcie E s ohľadom váh $w = (w_1, w_2, \dots, w_m)$ je vektor, ktorý sa skladá z parciálnych derivácií E podľa individuálnej váhy. Vieme zapísať nasledovne:

$$\text{grad}(E) = \Delta E = \left(\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right)$$

Podľa gradientu vieme upraviť konkrétnu váhu nasledovne :

$$w_i = w_i - \alpha \frac{\partial E}{\partial w_i}$$

kde α je konštanta ktorú nazývame rýchlosť učenia (rýchlosť učenia, bola spomenutá pri ReLU), ktorá nám upravuje veľkosť zmeny ktorú vykonáme. Niekedy je vhodné aby sa aplikovalo takzvané momentum, kde sa α dynamicky upravuje podľa E z predošlej epochy, aby sme mohli dosahovať presnejšie hodnoty. Vždy platí že $\alpha \neq 0$.

Danú formulu vieme upraviť na takzvané δ (delta) pravidlo.

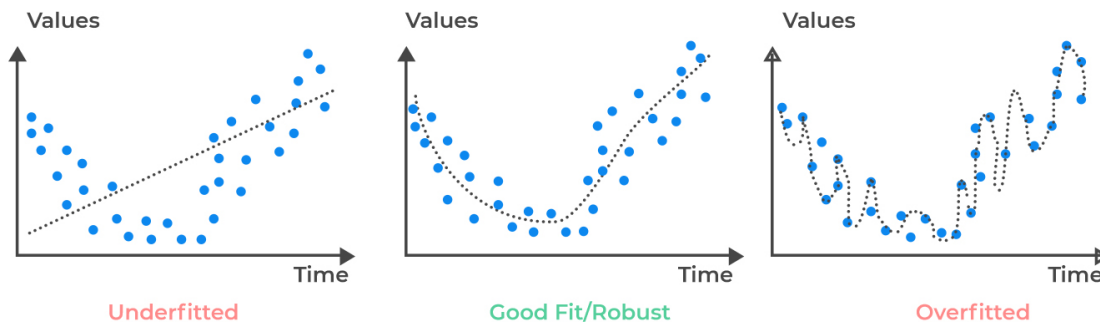
$$w_i = w_i + \alpha \delta g' x_i^p$$

$$\delta = (y^p - g_w(x^p))$$

$$g' = \text{derivácia aktivačnej funkcie}$$

1.1.11 Overfitting

Overfitting znamená že daná sieť až priveľmi dobre počíta výstupy pre trénovaciu množinu na úkor generalizácie nad testovacou. Tento jav môže nastať ak sieť trénujeme na priveľkom počte epóch, alebo máme až priveľa neurónov/vrstiev. Aby sme sa mu vyhli, pracujeme s validačnou množinou podľa ktorej upravujeme rýchlosť učenia a neuróny na vrstvách.



Obr. 14

1.2 Technológie

1.2.1 Python [pyt]

Python je open-source, vysokoúrovňový, multi-paradigmaticý programovací jazyk, ktorý v roku 1991 ho navrhol Guido van Rossum. Používa dynamické typovanie. Taktiež je interpretovaným jazykom, takže urýchľuje cyklus úpravy a následného debugovania. Má široké využitie, najmä vďaka veľkému množstvu externých knižníc

1.2.2 Pandas

Pandas je open-source softvérová knižnica pre jazyk Python. Používa sa na narábanie s dátami a ich analýzu. Jej výhodou je že dokáže spracovať dáta z CSV, Excel súboru, alebo aj SQL databázy.

1.2.3 Numpy

Numpy je softvérová knižnica pre jazyk Python. Je široko používaná pre prácu s vektormi, maticami alebo aj viacrozmernými polami. Obsahuje taktiež matematické funkcie, ktoré sú aplikovateľné na tieto dátové štruktúry. Sú rýchlejšie ako keby sme ich písali vo vanilla pythone (myslené v3.) a to vďaka optimalizáciám vzhľadom na dané štruktúry. Taktiež existuje viacero knižníc, ktoré Numpy ako keby ďalej rozširujú.

1.2.4 PyTorch

PyTorch je open-source framework pre jazyk Python, ktorý má široké využitie v oblasti machine learningu (strojové učenie). Môžeme ho prirovnať knižnici Numpy s tým rozdielom že PyTorch je urýchlené pomocou GPU. Doplním. Sľubujem.

Literatúra

[DA.05] Drachman DA. Do we have brain to spare? 2005.

[pyt] Python official website. <https://www.python.org/>. [Online; accessed 8-February-2022].