STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 1: Matematika a statistika

Neuronové sítě

Maximilián Alexander Mašek
Plzeňský kraj

Plzeň 2022

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 1: Matematika a statistika

Neuronové sítě

Neural networks

Autoři: Maximilián Alexander Mašek
Škola: Gymnázium Františka Křižíka, Sokolovská 1165/54, 323 00 Plzeň
Kraj: Plzeňský kraj
Konzultant: Mgr. František Kaska

Plzeň 2022

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Plzni dne 21.3.2022 ……
Maximilián Alexander Mašek

Poděkování

Tímto bych chtěl rád poděkovat panu Mgr. Františku Kaskovi, který mi při práci pomohl, vždy když jsem potřeboval. Na závěr bych chtěl poděkovat i panu Mgr. Danielu Namovi, který si vzal ten čas a vysvětlil nám všechny náležitosti psaní této práce.

Anotace

Ve své práci jsem se zabýval neuronovými sítěmi a jejich matematickým principům. Cílem mé práce bylo vysvětlit fungování neuronových sítí a zároveň vytvořit model, jehož funkčnost bych ukázal na konkrétním příkladu. K vytvoření takového modelu jsem použil programovací jazyk Python, který je pro tuto práci z mnoha důvodů výhodnou volbou. V programu byla použita knihovna zvaná „Numpy“, která výrazně ulehčila práci s maticemi a vektory.

Klíčová slova

Neuron; Neuronová síť; Perceptron; Přenosová funkce; Učící pravidlo; Učící parametr; Učení s učitelem;

Annotation

In my work, I dealt with neural networks and their mathematical functioning. The goal of my work was to explain the functioning of neural networks and create a functional model, which functionality I would show on an example. To create such a model, I used the Python programming language, which is a convenient option for this work, for many reasons. A library called "Numpy" was used in the program, which greatly facilitated the work with matrices and vectors.

Keywords

Neuron; Neural network; Perceptron; Activation function; Learning rule; Learning rate; Supervised learning;

Obsah

1 Úvod

V této práci jsem se zaměřil na umělé neuronové sítě. A o co se vlastně jedná? Umělé neuronové sítě jsou matematickým modelem biologických neuronových sítí využívaných naším mozkem. Teď ve zkratce víme, o co se jedná, ale k čemu vůbec slouží? Umělé neuronové sítě nám dávají možnost řešit i nelineární problémy, které by klasický program nevyřešil. To znamená, že kdybychom si v případě nelineárního problému vstupní vzorky vyjádřili v grafu, zjistili bychom, že by je nebylo možné rozdělit pomocí přímky. Kdybychom se podívali na náš svět, zjistíme že málokterý problém je lineární. To že by ho klasický program nedokázal vyřešit není úplně přesné, bylo by to ale velice náročné, protože bychom všechny souvislosti, které se neuronová síť naučí sama, museli ručně vypsat. A to je ta průlomová myšlenka. Není tedy lepší, abychom program naučili fungovat, aniž bychom museli vlastnosti ručně vypisovat?

A právě tato myšlenka, vytvořit umělé vědomí, provází lidstvo již velice dlouho. Výsledkem tohoto hledání jsou, již výše zmíněné, umělé neuronové sítě, které jsou podstatou defacto každé umělé inteligence dnešní doby. Na rozdíl od lidského mozku, který má kolem sto miliard neuronů, dokážeme zatím vytvořit sítě s miliony neuronů. Proto bych řekl, že k vytvoření skutečného funkčního vědomí nás čeká ještě velmi dlouhá cesta. To ale v žádném případě neznamená, že nám jsou umělé neuronové sítě k ničemu, ba naopak. Dnes zažívají obrovský rozmach a jsou hojně využívány. Například u autonomního řízení vozidel, jako třeba prototypy firmy Tesla, nebo u autopilota v letadlech. Používají se také u programů pro rozpoznávání obličej, jako třeba takzvané „FaceID“ značky Apple, nebo „Windows Hello“ značky Microsoft. Hlavní předností neuronových sítí je jejich univerzální využití, ať už v těžkém průmyslu, nebo třeba v medicíně a v mnohých dalších oborech. Zkrátka, jejich možnosti jsou obrovské a dají se implementovat téměř všude. A to je také důvod, proč je aktuálně umělá inteligence tak populární.

Mým hlavním cílem bylo osvětlit toto téma i těm, co nedisponují odbornými znalostmi v oboru. Musím také dodat, že ač se jedná o stále diskutovanější téma, je opravdu těžké najít relevantní zdroje v českém jazyce, které by ale zároveň byly pochopitelné také pro člověka bez odborných znalostí. Myslím si, že pochopit neuronové sítě je pro nás velice důležité a čím více lidí se tomuto tématu bude věnovat, tím bude, dle mého názoru, rychlejší pokrok.

Zároveň bych řekl, že je to velikým lákadlem, zvlášť pro mladou generaci, protože nedostatek odborníků a zároveň zvyšující se poptávka je dobrou příležitostí naskočit do tohoto oboru. Z mého pohledu bude umělá inteligence ve světě hrát čím dál tím větší roli. Musím však dodat, že bez základních znalostí matematiky se u čtení této práce neobejdete. Přečtení této práce doporučuji každému nadšenci do umělé inteligence, který by chtěl také zlehka porozumět jejímu fungování.

2 ÚVOD DO NEURONOVÝCH SÍTÍ

V této kapitole si nejprve vysvětlíme, co to umělé neuronové sítě jsou a jak jsou inspirovány fungováním lidského mozku. Poté se podíváme, kde a jak se tyto sítě v dnešní době využívají. A nakonec zlehka zavítáme také do historie a představíme si například první funkční model umělého neuronu.

2.1 Definice

Umělá neuronová síť je systém, který zpracovává informace, skládající se z množství jednotek (neuronů), které si na základě daného propojení posílají informace ve formě aktivace, či naopak „deaktivace“ jednotlivých neuronů. Do jisté míry jsou tato propojení odrazem struktury biologického mozku. Aby se neuronové sítě mohly učit, potřebují učící algoritmy, kterých existuje velké množství. Velkou výhodou je univerzálnost využití. Takovéto sítě umožňují řešit problémy v různých oblastech statistiky, technologie nebo ekonomie. Protože neuronovou síť aplikujeme na konkrétní problém, nedokáže se, na rozdíl od našeho mozku, zaměřit i na řešení dalších problémů (Ebert, 2019 str. 6).

2.2 Využití

Ač se to na první pohled nezdá, setkáváme se s nimi téměř denně a to například, když hledáme obrázky pomocí internetového prohlížeče Google.

Možné využití neuronových sítí stále roste, a to hlavně díky neustále zvyšujícímu se výkonu počítačů, díky kterému se potřebná doba pro trénování těchto sítí značně zkracuje. Dalším důvodem je také fakt, že se databáze vzorků zvětšují a větší množství vzorků má za následek zvýšení přesnosti sítě.

Je neuvěřitelné, ale zároveň také děsivé, co všechno se za pár let dokázaly naučit. Umělé neuronové sítě najednou umí i to, co dříve zvládl jen člověk jako například skládat písně, hrát šachy nebo i psát básně. Relativně nový program zvaný „GPT-3“ od neziskové společnosti OpenAI dokáže dokonce psát programy vlastní, nebo vést konverzaci.

Musím přiznat, že ani já jsem zprvu nedokázal rozeznat báseň vytvořenou programem a báseň vytvořenou člověkem. Až po důkladném pročtení bylo znát, že ta báseň napsaná programem postrádala hlubší význam.

Neuronové sítě využívá také velké množství aplikací i her, jako například známá funkce „Windows Hello“, která díky uložení snímku Vašeho obličeje dokáže později při přihlášení určit, zda se jedná o Vás, či nikoliv. Nespornou výhodou je přípustná odchylka. To znamená, že se nemusí jednat o 100% totožný snímek, ale stačí dostatek společných vlastností.

Umělé neuronové sítě však dokážou zachránit i život. V medicínské sféře se testují programy pro poznávání zhoubných nádorů podle obrázků. Na základě mnoha testů lze tvrdit, že mají lepší výsledky než člověk. Velkou výhodou je také fakt, že diagnózu provede program takřka okamžitě a bez potřeby experta v oboru. A právě včasná diagnóza může pacientům mnohdy zachránit život.

2.3 Historie neuronových sítí

Historie neuronových sítí sahá od roku 1943 až do současnosti. Přelomovými objevy byly Perceptron na konci 50. let a algoritmus zpětného šíření chyby (angl. Back-Propagation), v polovině 70. let. Mezi tím byla takzvaná zima, ve které byla zpochybněna užitečnost a proveditelnost neuronových sítí. Hlavní překážkou byl v té době nedostatek výkonného hardwaru. Lidský mozek je studován již několik tisíc let. První krok k umělým neuronovým sítím byl však učiněn až v roce 1943, kdy neurofyziolog Warren McCulloch a mladý matematik Walter Pitts napsali článek o tom, jak neurony vůbec fungují. Vytvořili pomocí elektrických obvodů jednoduchý neuron, který měl více vstupních signálů a pouze jeden výstup.

V roce 1949 opět otevřel koncept neuronů Donald Hebb, když ve své knize „The Organization of Behaviour“ poukázal na to, že „pokud biologický neuron aktivuje jiný neuron, tak spojení mezi nimi zesílí. To znamená, že váha mezi dvěma neurony se zvětší pokaždé, když mají stejný výstup. Pokud se dva neurony aktivují ve stejný čas, je posíleno jejich spojení“ (Farkaš, 2019 str. 3). S přibývajícím výkonem tehdejšího hardwaru bylo nakonec v roce 1950 možné nasimulovat hypotetickou neuronovou síť. Poprvé se o to pokusil vědec z IBM Nathanial Rochester. Bohužel však jeho pokus skončil neúspěchem.

V roce 1957 vynalezl Frank Rosenblatt Perceptron. Jednalo se v podstatě o McCullochův a Pittsův model, ale s přidaným algoritmem učení, který již dokázal rozdělit lineárně separovatelná data. Již v roce 1958 Frank Rosenblatt využil svůj algoritmus Perceptronu pro vývoj prvního neuropočítače „Mark I Perceptron“ v laboratoři „Cornell Aeronautical Laboratory“. Za pomocí senzoru o velikosti, 20krát 20 pixelů dokázal rozpoznat jednoduché číslice (Ebert, 2019 str. 9). V roce 1960 Bernhard Vitro a Marcia Hoff, ze Stanfordské Univerzity, vyvinuli modely, které nazvali "Adeline" a "Madaline". Adeline byl vyvinut tak, aby rozpoznal binární obrazce, takže pokud četl bity z telefonní linky, mohl předpovědět další bity. Madaline byla první neuronová síť aplikovaná na problém reálného světa za pomoci adaptivního filtru, který eliminuje ozvěny na telefonních linkách. Přestože je tento systém velice starý, je stále komerčně využíván (Volná, 2002). Tento model využíval učící pravidlo Delta, které si popíšeme později (Ebert, 2019 str. 9).

„V roce 1969 Marvin Minsky a Seymour Papert ve své knize Perceptrons ukázali několik vážných nedostatků Perceptronu. Nejdůležitější poznatek byl, že Perceptrony nejsou schopny řešit triviální nelineární problém typu XOR (Exkluzivní disjunkce). Tento poznatek vedl k velkému poklesu zájmu o umělé neuronové sítě většiny vědců“ (Farkaš, 2019 str. 4).

V 80. letech přichází Kunihiko s konvolučními neuronovými sítěmi obsahující více neuronových vrstev a svým modelem nazvaným „Neocognitron“ (Ebert, 2019 str. 9).

Na začátku 90. let nastává velký rozmach, a například i známá americká agentura DARPA začala aktivně podporovat výzkum neuronových sítí, zanedlouho následovaly i jiné organizace (Volná, 2002). V roce 1986 přichází algoritmus zpětného šíření chyby, též Back-Propagation pro vícevrstvé neuronové sítě (Volná, 2002). Problém Perceptronu, který nedokázal vyřešit nelineární problémy, se však podařilo vyřešit roku 1989, kdy tým New Yorské univerzity již známý algoritmus Back-Propagation aplikoval na konvoluční sítě. Model této neuronové sítě pojmenovaný „LeNet“ byl poté úspěšně otestován na datech tvořící ručně psané číslice. Tento model byl od roku 1990 ve Spojených státech využívána pro automatické rozpoznání poštovního směrovacího čísla. V roce 2011 byl algoritmus tohoto modelu upraven tak, aby mohl fungovat za pomoci grafické karty, což několikanásobně zrychlilo proces trénování a otevřely se možnosti s daleko větším množstvím trénovacích dat.

Umělé neuronové sítě fungující právě na tomto principu, byly také první, které dokázaly předčít člověka v rozpoznání obrazců (Ebert, 2019 str. 10).

Tím to však nekončí. Také dnes přicházejí vědci stále s různými modifikacemi těchto základních algoritmů, které dosahují čím dál lepších výsledků. Dalšímu výzkumu nahrává také fakt, že je dnes umělá inteligence využívána téměř všude a možné využití stále roste. Další nespornou výhodou je také prudce stoupající výkon dnešních počítačů, čímž se otvírají dveře komplexnějším neuronovým sítím s větším množstvím trénovacích dat.

3 PŘENOSOVÁ FUNKCE

V této kapitole se podíváme již na konkrétní funkci, která je podstatná pro fungování neuronové sítě tím, že se na základě ní mění hodnota výstupu jednotlivých neuronů sítě. Vysvětlíme si, proč je přenosová funkce potřeba a jak vlastně funguje. Také si podrobně představíme ty nejznámější a dnes nejpoužívanější. Ukážeme si jejich výhody a nevýhody a rozdíl mezi použitím funkce lineární či nelineární.

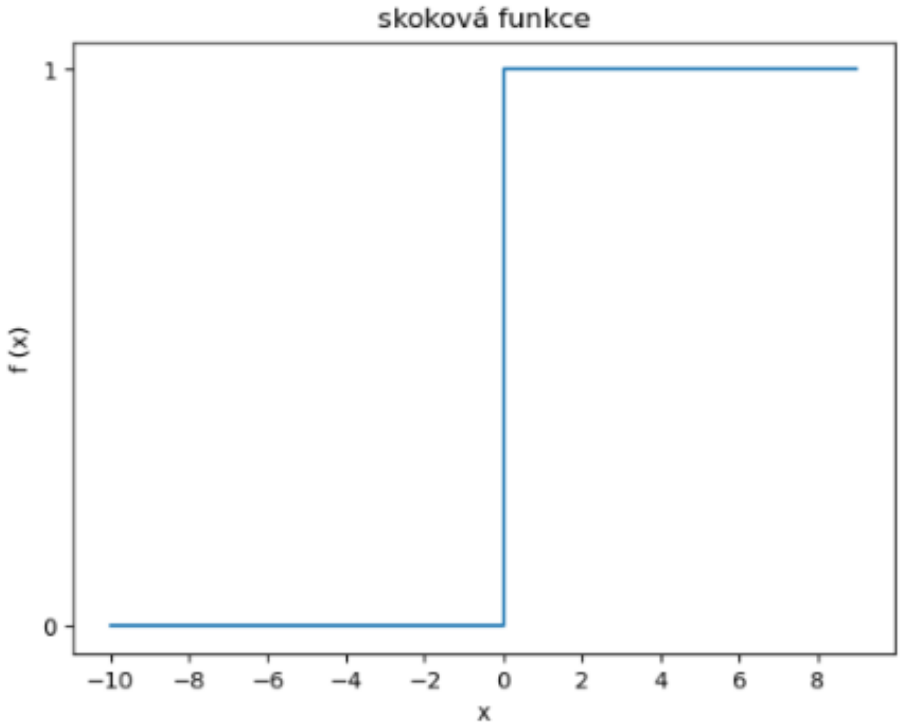
3.1 Definice

Přenosová, nebo také aktivační funkce neuronu, definuje jeho výstup na základě sady vstupů. Nejprve se tedy provede součet vstupů vynásobených váhami, neboli vážený součet. Funkce pak provede určitý typ operace, která závisí na námi vybrané přenosové funkci, aby transformovala součet na číslo mezi libovolnou dolní hranicí a libovolnou horní hranicí. Přenosová funkce je vlastně inspirována aktivitou v našem mozku, ve kterém jsou neurony aktivovány různými podněty. Vysvětleme si to na následujícím příkladu. Pokud ucítíte něco příjemného, tak určité neurony v mozku na základě podnětů vyšlou signál, čímž se aktivují, jiné naopak signál nevyšlou a zůstanou v inaktivním stavu. To může být také reprezentováno binárně, což znamená nulou pro neaktivaci a jedničkou pro aktivaci neuronu (Borzymowski, 2019 str. 22). Ale jsou tyto funkce vůbec potřeba a nešlo by to i bez nich?

Odpověď na tuto otázku není úplně jednoznačná. Mohu mít neuron bez přenosové funkce. Takový neuron by tedy využil lineární funkci $f(x) = x$. Problém však je ten, že by takový model mohl řešit jen čistě lineární problémy, pro které by ale taková neuronová síť nebyla vůbec potřeba a rychleji by se vyřešily klasickou formou. Zajímavější to však bude v případě řešení nelineárního problému jako třeba u klasifikace či regrese. Představme si následující problém. Potřebujeme od sebe rozlišit následující vzorky: obrázky lidí a obrázky zvířat. Kdybychom si vzorky zvířat i lidí definovali jako body grafu, zjistili bychom, že by se pomocí nějaké nelineární funkce dali rozdělit do dvou tříd, a to je přesně to, o co nám jde. Pokud bychom ale přenosovou funkci nepoužili, museli bychom vzorky rozdělit pomocí přímky, což by v našem případě nebylo možné.

3.2 Nejpoužívanější přenosové funkce

3.2.1 Skoková funkce

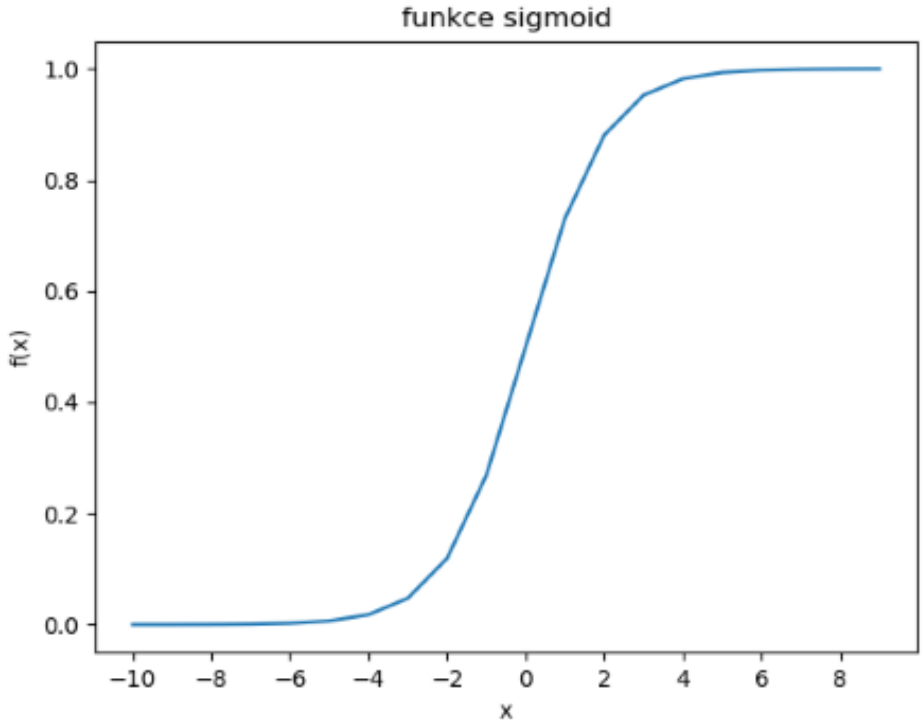


Obrázek 1: Graf skokové funkce

Toto je vůbec nejstarší přenosová funkce, která byla využívána již McCullochovím modelem neuronu. Jedná se o jednoduchou binární funkci. To znamená, že výstup je buď hodnota nula, nebo jedna. Tedy pokud je vstupní hodnota záporné číslo, bude výstup této funkce číslo nula, kdežto když bude vstupní hodnota kladná nebo nula, bude výstup funkce číslo jedna.

Velký problém této přenosové funkce však nastává při klasifikaci vstupů, kde se zjišťuje, jaké třídě vzorek náleží na základě libovolných společných rysů. Funkce by tedy mohla hodnotu jedna přiřadit pouze jedné třídě a zbytek tříd by musel mít hodnotu nula. Při takové klasifikaci je však vysoká pravděpodobnost, že se aktivuje více neuronů, což by znamenalo, že by neuronová síť musela vzorek přiřadit více třídám současně. A to je také důvod proč se tato přenosová funkce již moc nevyužívá (Borzymowski, 2019 str. 22).

3.2.2 Sigmoidní funkce

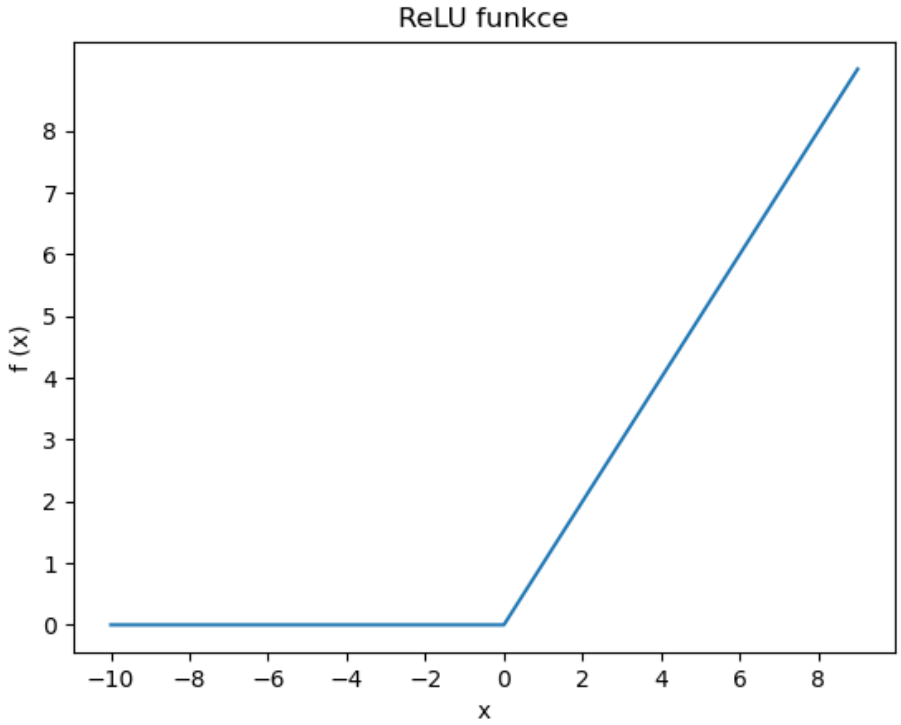


Obrázek 2: Graf sigmoidní funkce

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

Podívejme se teď na přenosovou funkci zvanou „Sigmoid“. Jedná se o standartní logistickou křivku. Funkce přijímá vstup a pokud má vstup velmi negativní hodnotu, bude výstup této funkce číslo velmi blízké nule. Naopak pokud je hodnota vstupu velmi kladná, bude výstup funkce číslo blízké jedné. Funkce sigmoid má tedy dolní hranici hodnotu nula a horní hranici hodnotu jedna. Jeden z problémů této přenosové funkce je však ten, že čím vyšší mají vstupy absolutní hodnoty, tím více se stávají nerozlišitelné, neboli rozdíl mezi nimi se stává velmi malým. Kdyby tento problém nastal, znamenalo by to, že by se síť i po dlouhé době trénování prakticky nic nenaučila (Ebert, 2019 str. 15).

3.2.3 ReLU funkce



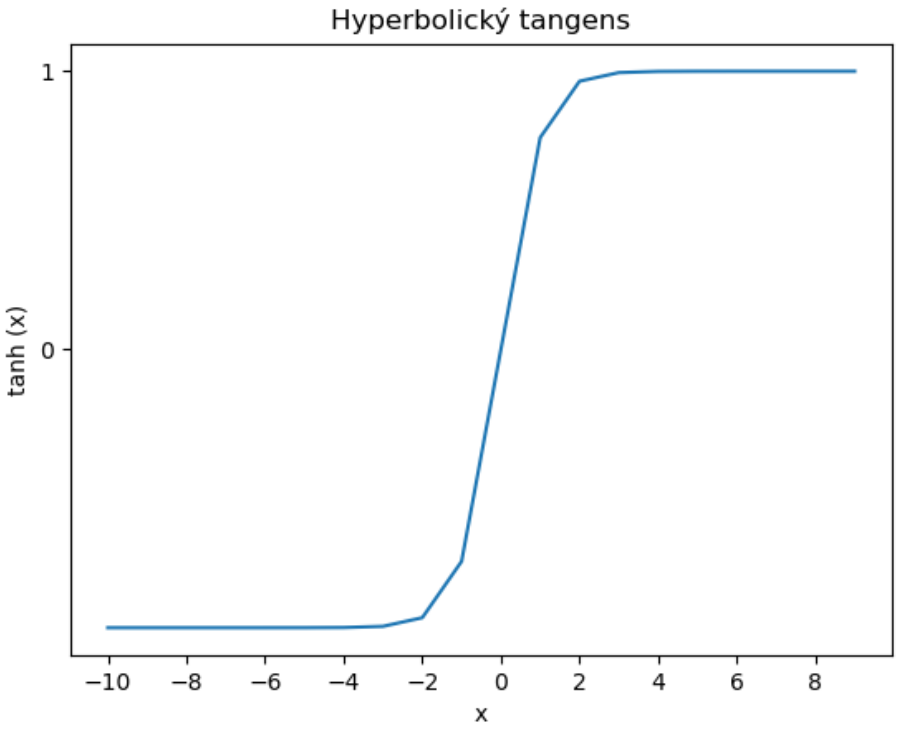
Obrázek 3: Graf funkce ReLU

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

Tato přenosová funkce (angl. Rectified Linear Unit) je jedna z nejpoužívanějších v klasifikačních problémech dnešní doby, a to hlavně díky jejímu nenáročnému výpočtu ale i přesto dobrým vlastnostem. Když se podíváme na graf výše, zjistíme, že se jedná o poměrně jednoduchou funkci. Vidíme, že pokud je hodnota vstupu menší než nula, je výstup přenosové funkce nula, avšak pokud je vstup větší nebo roven nule, svoji hodnotu si zachová a výstup funkce bude stejný jako její vstup neboli f (x) = x. Jedna z nevýhod je, že ve funkci není rozdíl mezi záporným číslem a nulou, což může vést k problémům. A proto existuje více modifikací této funkce jako například takzvaný „Leaky ReLU“ kde je f(x) v záporných hodnotách rovna p*x (p je parametr za který dosadíme číslo v závislosti na chtěném sklonu funkce). Protože si zde ale představujeme jen základní přenosové funkce, nebudeme se této modifikaci dále věnovat. Je však dobré vědět že existuje.

Narozdíl od sigmoidní funkce, která má jako výstup vždy nějakou hodnotu mezi nulou a jedničkou, narážíme u přenosové funkce „ReLU“ na problém, že se kvůli možnému nulovému výstupu můžou některé neurony stát neaktivními, a to i po celou dobu trénovacího procesu, čímž je značně narušena efektivita celé neuronové sítě. Toto můžeme ale vidět i jako výhodu, protože tím, že se neurony stanou neaktivními, se značně zjednoduší výpočet, a tudíž se zkrátí i potřebná doba pro trénování sítě (Ebert, 2019 str. 17).

3.2.4 Hyperbolický tangens



Obrázek 4: Graf funkce hyperbolický tangens

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

Funkce hyperbolický tangens je také hojně využívána a je často upřednostňována před sigmoidní přenosovou funkcí. A to proto, že oproti sigmoidní funkci nabývá hyperbolický tangens hodnot mezi 1 a −1, čímž se neztrácí hodnota záporného čísla a také je výsledná hodnota blíže nule, což je lepší pro učící algoritmus.

Nevýhodu má ale stejnou, čím jsou vstupní čísla v absolutní hodnotě větší, tím je rozdíl mezi nimi menší. Neboli, pokud jsou vstupní čísla příliš vysoká stane se rozdíl mezi nimi zanedbatelný, a to může mít za následek, že by je neuronová síť nedokázala rozlišit (Ebert, 2019 str. 16). Jediný případ, kdy je dávána přednost sigmoidní funkci je u výstupní vrstvy klasifikační neuronové sítě, u které potřebujeme zjistit, jak moc si je síť jistá, že se jedná o danou třídu. Výstup tedy bude číslo mezi nulou a jedničkou, a to poté můžeme i snáze vyjádřit procentuálně.

4 JEDNOVRSTVÁ NEURONOVÁ SÍŤ (PERCEPTRTON)

V této kapitole si ukážeme základní jednovrstvou neuronovou síť zvanou „Perceptron“ a podrobně se podíváme na její stavbu. Na základě nabytých vědomostí z předchozích kapitol si na příkladu vysvětlíme fungování jednoduchého učícího pravidla. Spolu s těmito vědomostmi si ukážeme fungování Perceptronu na dvou příkladech s již reálnými problémy. Také si řekneme, proč druhý příklad nelze jednovrstvou neuronovou sítí vyřešit a tím si zároveň odpovíme na otázku proč vůbec vznikly i sítě vícevrstvé.

4.1 Definice

Abychom mohli začít vytvářet vlastní neuronovou síť, musíme si nejdříve představit neuron a neuronovou síť, která obsahuje pouze jediný neuron. Ta nejjednodušší se nazývá Perceptron. Perceptron je nejjednodušším matematickým modelem neuronové sítě, inspirovaným skutečným fungováním neuronů v živém organismu.

Jednoduše řečeno, funguje takový Perceptron tak, že má x vstupů a jeden výstup. Vstupy se vynásobí zpočátku náhodně vygenerovanými váhami a suma těchto čísel plus práh (viz níže) je poté porovnána s požadovaným výsledkem. Tím dostaneme takzvanou chybu neuronové sítě, pomocí které se později budou optimalizovat váhy tak, aby byla ve výsledku chyba co nejmenší (Farkaš, 2019 str. 2). To tedy znamená, že se jedná o „Supervised learning“ neboli učení s učitelem, kdy jsou sítí předloženy vstupní ale i výstupní trénovací vzorky.

4.2 Stavba Perceptronu

4.2.1 Vstupy

Vstupy Perceptronu jsou podněty z vnějšího prostředí, charakterizovány libovolným reálným číslem. V našem případě se jedná o vektor hodnot X, který obsahuje různé informace o zkoumané věci. Vstupy mohou být například i číselně vyjádřené obrázky a právě na tomto principu funguje jakékoliv rozpoznání obličeje, zvířete nebo jiné věci. Vstup je vždy vyjádřen libovolnou číselnou řadou.

4.2.2 Váhy

Váhy jsou zpočátku náhodně vygenerovaná čísla, který mi se poté násobí vstupy. Hodnoty vah i prahů jsou právě to, kde se projeví učící proces. Jejich hodnota se mění na základě vypočítané chyby sítě. Na obrázku jsou váhy definované jako ohodnocení hran.

4.2.3 Výstupy

Výstupy jsou výsledky neuronové sítě, které se počítají sumou vstupů vynásobených váhami. Díky výstupu sítě můžeme vypočítat takzvanou chybu sítě, a to tak, že od požadovaného výsledku odečtu skutečný výstup sítě. Na základě této chyby je možné pomocí dopředného algoritmu postupně měnit hodnotu jednotlivých vah (Pircher, 2017 str. 15).

4.2.4 Prahy

Práh, nebo také často nazývaný „Bias“ je hodnota nacházející se uvnitř samotného neuronu, která se stará o posunutí celého optimalizačního grafu. Zpočátku se jeho hodnota nastavuje na hodnotu 1, na základě algoritmu dopředného šíření chyby se optimalizuje (Pircher, 2017 str. 15).

4.2.5 Dopředné šíření signálu

Abychom mohli Perceptron vůbec něco učit, musíme si nejdříve představit funkci, pomocí které může Perceptron určit svůj výstup.

$$y = \sum_{i} w_i x_i + b$$

Vidíme, že se jedná o poměrně jednoduchou rovnici. Abychom získali výstup Perceptronu značený y musíme vstupní hodnoty x vynásobit jim náležící váhou w. Pokud má Perceptron více než jeden vstup, tak se hodnoty vstupů vynásobených váhami sečtou. Provede se takzvaný vážený součet a poté se pouze přičte hodnota prahu daného neuronu b (Terhag, 2018 str. 18).

4.3 Delta pravidlo učení

4.3.1 Vysvětlení

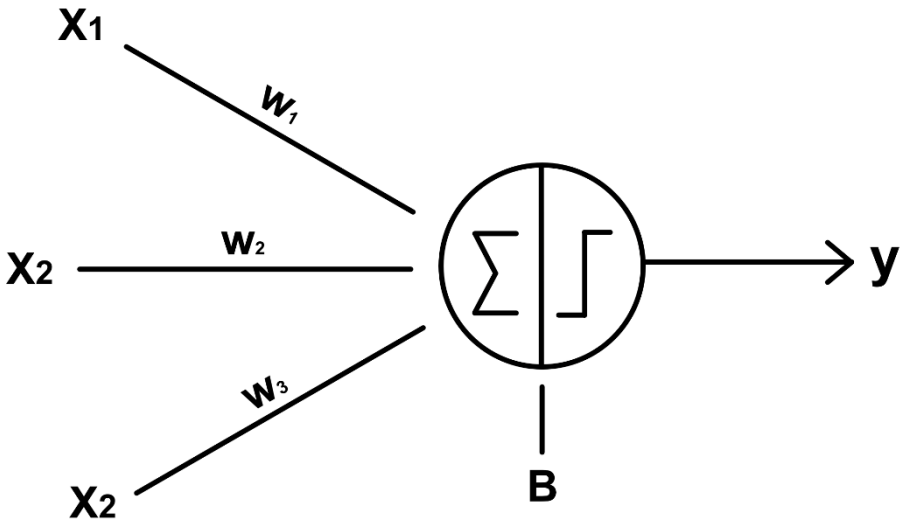
Toto pravidlo se převážně využívá pro učení neuronů s výhradně lineární přenosovou funkcí. Po úpravě se však dá využít i u neuronů s nelineární přenosovou funkcí.

Pravidlo lze zapsat následovně:

$$\Delta w_{ik} = \varepsilon * \delta_i * x_k$$

$$\delta_i = y_i - y_i(out)$$

A co tyto dvě rovnice znamenají?



Obrázek 5: Perceptron

Podívejme se na obrázek jednoduchého neuronu. Vidíme, že tento neuron má tři vstupní hodnoty, které se značí jako x_k (protože máme tři vstupy tak k = 1 až 3). Každý vstup má přiřazenou váhu w_{ik}, kde *i* značí daný neuron. Výstup neuronu je y_i. Zbývá nám ε, přičemž se jedná o takzvaný učící parametr, který určuje rychlost učení sítě.

Abychom mohli spočítat Δw_{ik} musíme určit chybu neuronu, kterou dostaneme odečtením skutečného výstupu ($y_i(out)$) od požadovaného výstupu (y_i) podle druhé rovnice, tím nám vyjde δ_i. Poté dosadíme do první rovnice. Nové váhy spočítáme tak, že ke stávajícím přičteme nově získané Δw_{ik} (Ploner, a další, 2009 stránky 10,11).

4.3.2 Příklad

Abychom fungování lépe pochopili, podívejme se na konkrétní příklad.

Máme definované vstupy sítě $(2 \quad 3 \quad 1)$ a chceme, aby výstup neuronu byl součet vstupů tedy 6. Váhy neuronu jsou zpravidla náhodně vygenerovány a jedná se většinou o čísla v intervalu (-1 ; 1). V našem případě zvolíme váhy jako vektor $(0.6 \quad 0.3 \quad -0.2)$.

Nejprve provedeme vážený součet pro získání výstupu neuronu. Abychom si ulehčili práci, můžeme využít skalární součin vektorů, který se spočítá jako $a_x b_x + a_y b_y + a_z b_z$. Vyjde nám hodnota 1.9. Abychom dostali chybu funkce odečteme skutečný výstup od požadovaného výstupu tedy $\delta_i = 6 - 1.9$. Chyba je tedy 4.1.

Za učící parametr dosadíme číslo 0,01 (proč použijeme právě tuto hodnotu, si vysvětlíme až v další kapitole). Teď už máme vše a stačí nám pouze dosadit do první rovnice.

$$\Delta w_{ik} = (0.01 * 4.1) * (2 \quad 3 \quad 1)$$

$$\Delta w_{ik} = (0.082 \quad 0.123 \quad 0.041)$$

Pro upravení vah přičteme nově získané Δw_{ik} a přičteme jej k aktuálním vahám.

$$(new)\, w_{ik} = (0.082 \quad 0.123 \quad 0.041) + (0.6 \quad 0.3 \quad -0.2)$$

$$(new)\, w_{ik} = (0.682 \quad 0.423 \quad -0.159)$$

Abychom zjistili, zda je výpočet opravdu správný, můžeme opět vypočítat chybu neuronu s novými váhami a porovnat jí s předchozí chybou (Ploner, a další, 2009). Nová chyba by měla být menší než původní. A opravdu, nová chyba 3,526 je skutečně menší než původní hodnota 4,1. A takto bychom mohli pokračovat, dokud by nebyla chyba minimální.

4.4 Perceptron v praxi

Abychom si fungování Perceptronu ukázali i v praxi a s problémy skutečného světa, vybral jsem dva, na kterých výše uvedený algoritmus vyzkoušíme. Oba problémy jsem zkoušel pomocí webu https://neuronal-network.netlify.app/, který jsem vytvořil a jehož zdrojový kód bude k dispozici.

4.4.1 Ukázka 1

První problém bude lineárního charakteru. Půjde o to, aby byly vstupní hodnoty sečteny a poté vynásobeny číslem dva. Protože chceme konkrétní výstup neuronu bez nechtěné úpravy, potřebujeme k tomu přenosovou funkci, která může nabýt jakýchkoliv hodnot bez omezení. Proto využijeme přenosovou funkci „Identity“ kde f(x) = x.

Jako vstupní hodnotu jsem zvolil vektor $(1 \quad 3 \quad 4)$ tudíž výstup trénovacího vzorku bude číslo 16. A jako neznámý vstup jsem zvolil vektor $(2 \quad 3 \quad 2)$. Ideálně by tedy výsledek měl být 14.

Učící parametr je standartně nastaven na 0.01 (proč tomu tak je si vysvětlíme později) a počet opakování jsem nastavil na 10 000. Výsledkem bylo číslo 11.27 a to znamená, že byla odchylka vůči skutečnému výsledku 19.5 %. Na první pohled se tato hodnota může jevit jako vysoká, ale musíme také brát v úvahu, že obvykle mívají neuronové sítě ohromné množství vstupů a počet opakování několik desítek milionů. Kvůli tomu jsou také často trénované na superpočítačích, které díky obrovskému výpočetnímu výkonu zvládnou takovýto počet opakování za extrémně krátkou dobu, na rozdíl od klasických počítačů či notebooků.

Abych vám ale dokázal, že Perceptron skutečně funguje a „nevyhazuje“ jen náhodná čísla, použil jsem tři trénovací vzorky a zvolil 100 000 opakování. Použitá trénovací data byly vstupní vektory $(1 \quad 3 \quad 4), (2 \quad 6 \quad 5), (0 \quad 7 \quad 3)$ a výstupní vektor $(16 \quad 26 \quad 20)$. Tady byl již výsledek 13.999… s odchylkou takřka 0 % o poznání přesnější. Vidíme tedy že po přidání pouhých dvou trénovacích vzorků a zvýšení počtu opakování jsme dosáhli mnohem lepších výsledků. Platí tedy, že s nárůstem počtu vzorků a počtem opakování roste i přesnost výsledku perceptronu.

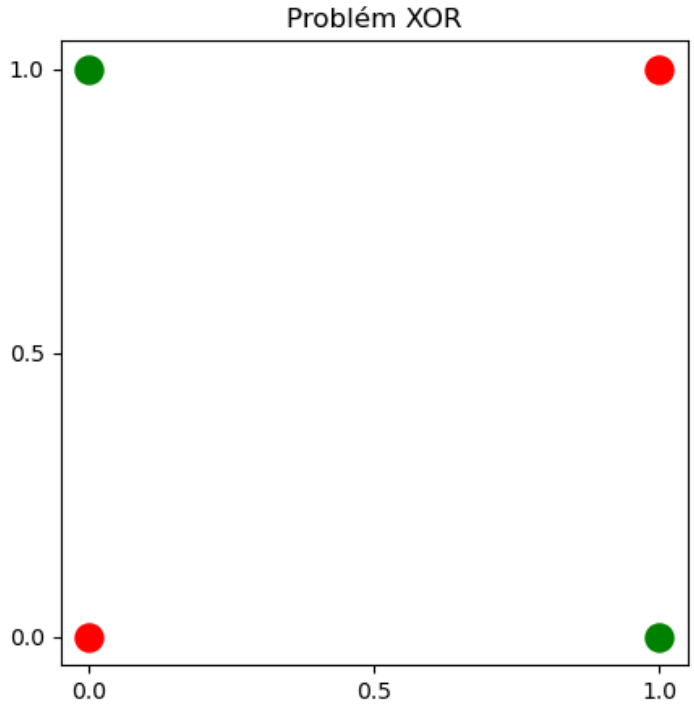
4.4.2 Ukázka 2

Jako druhý příklad jsem vybral známý problém „XOR“, který byl do jisté míry příčinou vzniku vícevrstvých sítí. XOR nebo česky Exkluzivní disjunkce, je pravdivá právě tehdy, pokud přesně jedna její část je pravdivá. Jinak je nepravdivá. Proč stojí právě tento problém za vznikem vícevrstvých sítí jsme si již uváděli.

Tentokrát byly vstupy trénovacích vzorků vektory $(0 \quad 0), (0 \quad 1), (1 \quad 0), (1 \quad 1)$ a výstupní vzorky vektor $(0 \quad 1 \quad 1 \quad 0)$.

Počet opakování jsem nastavil na 10 000 iterací. Po dosazení vektoru $(1 \quad 1)$ vyšel výsledek 0.66 oproti reálnému výsledku 0, což znamená, že odchylka byla 66 %, a to je opravdu hodně. Situace se nezměnila ani s navýšením počtu opakování na 1 000 000. Po dosazení toho samého vektoru vyšla znovu odchylka 66 % (Kvůli náhodnému zvolení vah nebyla hodnota identická, ale lišila se jen o pár tisícin, a to po zaokrouhlení vedlo ke stejnému výsledku).

Proč tomu ale tak je? Abychom si mohli na tuto otázku odpovědět, musíme se nejprve pozorněji podívat ne tento konkrétní problém. Podívejme se tedy na XOR (Exkluzivní disjunkce) v grafu.



Obrázek 6: Problém XOR (Exkluzivní disjunkce)

V grafu můžeme vidět dvě skupiny, zelené puntíky jsou výstupy, kdy XOR vyšel 1 a naopak červené puntíky jsou ty, co vyšly nula. Teď vám dám malý úkol. Zkuste se zamyslet a pokuste se tyto dvě skupiny, tedy zelené a červené puntíky, od sebe oddělit přímkou. Správná odpověď je, že přímkou je rozdělit nelze, a to znamená, že jsou lineárně neseparovatelné. Výstup této ukázky tedy je, že jednoduchý perceptron nedokáže vyřešit příklady nelineárního charakteru. Kdo by chtěl ještě další důkaz, ukážeme si tento problém ještě na soustavě rovnic. Z fungování umělého neuronu popsaného výše, si lze odvodit následující rovnice.

$$Ow_1 + Ow_2 = 0$$

$$1w_1 + Ow_2 = 1$$

$$Ow_1 + 1w_2 = 1$$

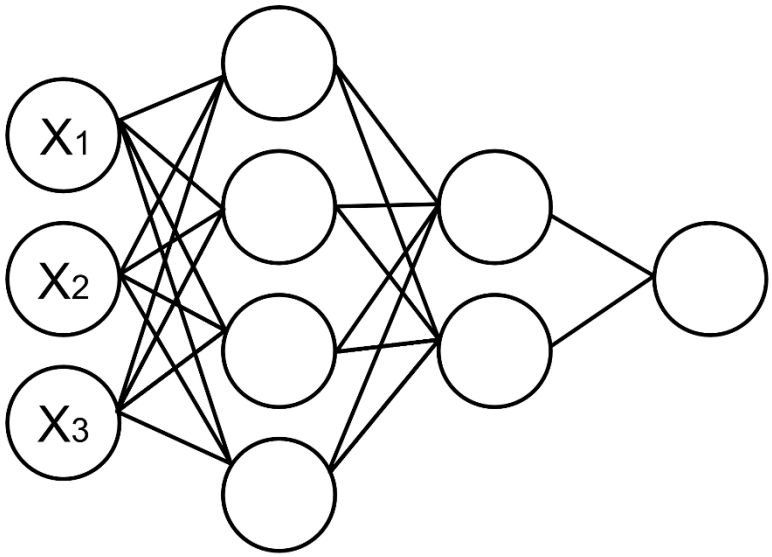
$$1w_1 + 1w_2 = 0$$

Úkolem je najít w_1 a w_2 takové, aby vyhovovaly všem čtyřem rovnicím. Vyjde nám, že příklad nemá řešení.

5 VÍCEVRSTVÁ NEURONOVÁ SÍT

V této kapitole si ukážeme vícevrstvé neuronové sítě a popíšeme si jejich stavbu. Řekneme si, co vůbec stálo za jejich vznikem. Poté se zaměříme na již lehce složitější učící algoritmus zvaný „Back propagation“ a na příkladu si ukážeme jeho fungování. Na závěr, tentokrát úspěšně, vyřešíme druhý příklad z předchozí kapitoly. Vyřešíme tedy problém zvaný „XOR“.

5.1 Definice



Obrázek 7: Vícevrstvá neuronová síť

S příchodem Perceptronu roku 1958 nastal velký pokrok ve výzkumu neuronových sítí. Avšak když kolem roku 1989 Minsky a Papert přišli na to, že Perceptron dokáže řešit jen lineárně spravovatelné problémy, muselo přijít řešení. Vícevrstvá neuronová síť se skládá z libovolného množství vrstev, které jsou tvořeny mnoha navzájem propojenými neurony, popsané již v předchozí kapitole. Jednotlivé vrstvy rozdělujeme na vstupní, skryté a výstupní.

V případě, že má síť více než jednu skrytou vrstvu, jako na obrázku, nazývá se hluboká neuronová síť (angl. Deep neuronal network).

Právě takováto síť dokáže úspěšně vyřešit například problém XOR (Exkluzivní disjunkce) s kterým přišli Minsky a Papert. Celou síť si můžeme představit jako takový filtr, kde každá vrstva hledá jiné vlastnosti (Pircher, 2017 stránky 17,18).

5.2 Dopředné šíření signálu vícevrstvé sítě

Signál dopředného šíření u vícevrstvé sítě je velice podobný tomu v jednovrstvé síti. Avšak vypočítaný výstup jednoho neuronu nemusí automaticky být výstupem celé neuronové sítě, ale může sloužit i jako vstup do dalšího neuronu. A to je také hlavní princip algoritmu dopředného šíření vícevrstvé sítě. Když se podíváme na obrázek výše, zjistíme, že výstup jedné vrstvy slouží jako vstup vrstvy druhé, a takto to pokračuje až k vrstvě výstupní.

5.3 Algoritmus zpětného šíření chyby (angl. Back propagation)

5.3.1 Definice

Back propagation je algoritmus minimalizující výslednou chybu neuronové sítě za pomoci gradientního sestupu, jedná se tedy opět o „Supervised learning“ neboli učení s učitelem. Gradientní sestup je algoritmus, pomocí kterého hledáme ideálně globální minimum neznámé funkce.

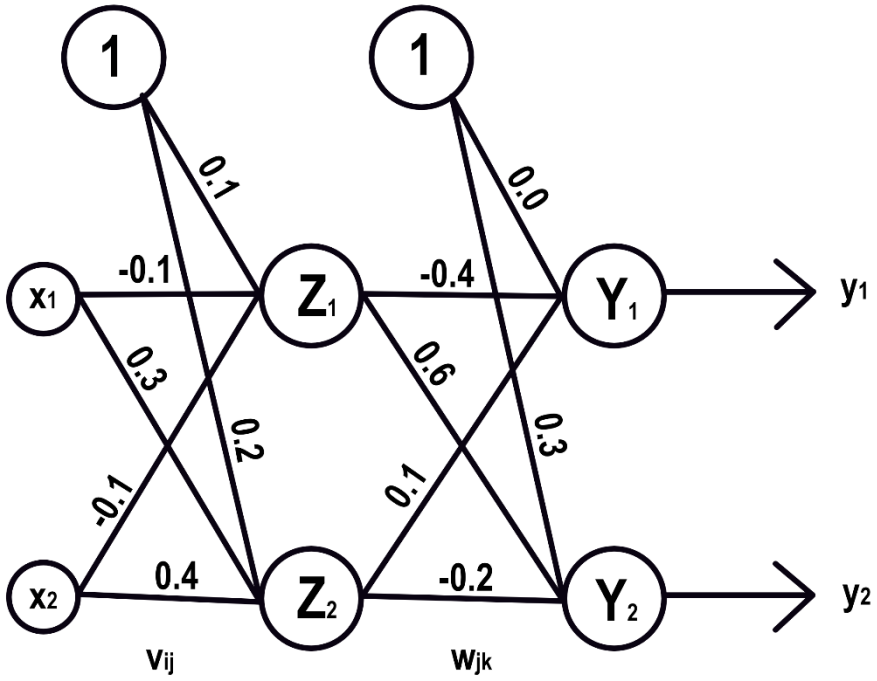
Ale co to znamená? Pojďme si to ukázat na následujícím příkladu. Představte si horolezce, který stojí na vrcholku hory, a to uprostřed sněhové bouře tak silné, že je sotva vidět na krok. Aby se horolezec mohl dostat zpět domů do údolí, musí po malých krůčcích nahmatávat okolí a postupovat směrem většího klesání. Velikost kroku horolezce je v případě gradientního sestupu reprezentován takzvaným učícím parametrem (angl. Learning rate), který má vždy hodnotu mezi nulou a jedničkou.

Pokud by byl definován nulou, znamenalo by to, že by se náš horolezec vůbec nepohyboval a nejspíš by na hoře umrzl. S neuronovou sítí by to bylo stejné, hodnoty vah by byly s každou iterací stále stejné, což znamená, že by se síť nic neučila. To si můžeme i velice snadno matematicky dokázat díky delta učícímu pravidlu, popsanému již v předchozí kapitole, z kterého do jisté míry bude vycházet i tento algoritmus.

Jak jsme již popisovali, je výsledek chyby sítě vynásobený vstupy poté učícím parametrem a výsledná hodnota je pak přičtena k dosavadním hodnotám vah. Avšak kdyby byl parametr nula, tak by se k hodnotám přičetla pouze nula, což znamená, že by se hodnoty vah nezměnily. Pokud je ale naopak zvolen parametr moc velký, může se stát, že bude náš horolezec údolí přeskakovat, ale nikdy se do něj nedostane. Také proto je správné zvolení hodnoty tohoto učícího parametru velice důležité.

Výše jsem uváděl, že se snažíme najít ideálně globální minimum funkce. Dostat se k tomuto minimu je však velice nepravděpodobné a ani se s tím nepočítá. Kdybychom si představili náhodnou funkci v prostoru, zjistili bychom, že minimum je více, avšak to největší je jen jedno, a to se nazývá minimum globální. Ostatní jsou minima lokální (Pircher, 2017 str. 27). Ale co se stane, když uvízneme právě v takovémto lokálním minimu? Stane se to, že se nedostaneme na nulovou chybu sítě, ale to nám vůbec nevadí. Kdybychom měli nulovou chybu, znamenalo by to, že funkce dokáže vložené vzorky identifikovat s přesností sta procent. Ale jak by to bylo s neznámými vzorky s podobnými rysy? Ty by rozpoznány nebyly, protože by síť měla nulovou toleranci a identifikovala by jen data identická s vloženými vzorky. To by však bylo pro náš typ problému, u kterého se počítá s určitou tolerancí neefektivní.

5.3.2 Příklad



Obrázek 8: Vícevrstvá neuronová síť (příklad)

Abychom mohli začít počítat konkrétní příklad, musíme si nejprve definovat tvar neuronové sítě. V našem případě se jedná o tvar (2 2 2), to znamená tři vrstvy po dvou neuronech, jedna vstupní, jedna skrytá a jedna výstupní.

Nejprve si určíme tréninkové vzorky. Vstupní vektor (0.5 0.3) musí mít právě dvě hodnoty, protože počet neuronů vstupní vrstvy je dva. To samé nastane také u výstupů, jelikož má výstupní vrstva také dva neurony budou i dva výstupy $t_{1,2}(0.1 \quad 0.3)$. Prahy mohou být buďto přímo uvnitř neuronu, nebo mohou fungovat jako další vstup do každé vrstvy. To znamená, že se v každé vrstvě navýší počet vstupů o jeden další. Přesně tak, jak je prezentováno v obrázku výše.

Jako přenosovou funkci skryté vrstvy použijeme hyperbolický tangens, popsaný výše. Přenosová funkce výstupní vrstvy bude funkce „Identity“ to znamená funkce f(x) = x. Váhy máme, využijeme totiž ty z obrázku.

Když už máme všechny počáteční hodnoty, pustíme se nejprve do dopředného šíření signálů podle rovnice popsané již dříve.

Dopředné šíření skryté vrstvy

$$z\_in_1 = (0.1 \cdot 1) + (-0.1 \cdot 0.5) + (-0.1 \cdot 0.3) = 0.02$$

$$z\_in_2 = (0.2 \cdot 1) + (0.3 \cdot 0.5) + (0.4 \cdot 0.3) = 0.47$$

$$z_1 = tanh(0.02) = 0.02$$

$$z_2 = tanh(0.47) = 0.43$$

Pomocí těchto rovnic máme vypočítané výstupy skryté vrstvy, které teď použijeme jako vstupní hodnoty vrstvy výstupní.

Dopředné šíření výstupní vrstvy

$$y\_in_1 = (0.0 \cdot 1) + (-0.4 \cdot 0.02) + (0.1 \cdot 0.43) = 0.035$$

$$y\_in_2 = (0.3 \cdot 1) + (0.6 \cdot 0.02) + (-0.2 \cdot 0.43) = 0.226$$

$$y_1 = 0.035$$

$$y_2 = 0.226$$

Poté, co dostaneme výstupy celé neuronové sítě, jsme schopni vypočítat její chybu tím, že výsledek odečteme od námi na začátku definovaného výstupu.

Chyba neuronové sítě

$$e_1 = 0.1 - 0.035 = 0.065$$

$$e_2 = 0.3 - 0.226 = 0.074$$

Výpočet hodnot delta neuronů výstupní vrstvy

Pomocí chyby můžeme vypočítat hodnotu δ výstupních neuronů.

$$\delta_k = 2 \cdot (t_k - y_k)\, \varphi'(y\_in_k)$$

A teď nám stačí dosadit do rovnice. Také víme, že derivací funkce f(x) = x je hodnota 1. Můžeme ji taktéž rovnou dosadit a vyjdou nám dvě hodnoty.

$$\delta_1 = 2 \cdot (t_1 - y_1)\, \varphi'(y\_in_1) \;=\; 2e_1 \;\cdot\; 1 \;\; = 0.13$$

$$\delta_2 = 2 \cdot (t_1 - y_2)\, \varphi'(y\_in_2) \;=\; 2e_2 \;\cdot\; 1 = \; 0.148$$

Výpočet hodnot delta neuronů skryté vrstvy

$$\delta_j = \sum_{k=1} \delta_k \, w_{jk} \, \varphi' \, (\, z\_in_j \,)$$

Ano, tato rovnice možná vypadá trošku strašidelně, ale jakmile si ji vysvětlíme a vypočítáme, zjistíme, že to tak hrozné není. w_{jk} jsou váhy mezi skrytou a výstupní vrstvou, δ_k jsou hodnoty delta výstupních neuronů, které jsme počítali v minulém kroku a $\varphi' \, (\, z\_in_j \,)$ je derivací hyperbolického tangentu a to je $1 - (z)^2$.

$$\delta_1 = (0.13 \cdot (-0.4) \;+\; 0.148 \cdot 0.6)\, \varphi'(z\_in_1) \;=\; 0.0368 \cdot \; (1 \;-\; (0.02)^2 \,) \;=\; 0.03679$$

$$\delta_2 = (0.13 \cdot 0.1 \;+\; 0.148 \cdot (-0.2))\, \varphi'(z\_in_2) \;=\; -0.0166 \cdot \; (1 \;-\; (0.43)^2 \,) \;=\; -0.01353$$

Aktualizace vah výstupní vrstvy

Když už máme téměř vše spočítané, stačí nám provést aktualizaci vah podle vzorců, které jsme si vysvětlovali již v předchozí kapitole.

$$\Delta w_{jk} = \varepsilon * \delta_k * z_j$$

Tabulka 1

NEURON 1	**NEURON 2**
$\Delta w_{01} = \varepsilon \cdot 0.13 \cdot 1 = 0.13 \cdot \varepsilon$	$\Delta w_{02} = \varepsilon \cdot 0.148 \cdot 1 = 0.148 \cdot \varepsilon$
$\Delta w_{11} = \varepsilon \cdot 0.13 \cdot 0.02 = 0.0026 \cdot \varepsilon$	$\Delta w_{12} = \varepsilon \cdot 0.148 \cdot 0.02 = 0.00296 \cdot \varepsilon$
$\Delta w_{21} = \varepsilon \cdot 0.13 \cdot 0.43 = 0.0559 \cdot \varepsilon$	$\Delta w_{22} = \varepsilon \cdot 0.148 \cdot 0.43 = 0.06364 \cdot \varepsilon$

Za učící parametr ε si dosadím 0.01 a podle rovnice $w_{jk} = w_{jk} + \Delta w_{jk}$ aktualizujeme váhy.

Tabulka 2

NEURON 1	**NEURON 2**
$w_{01} = 0.013$	$w_{02} = 0.3148$
$w_{11} = -0.39974$	$w_{12} = 0.600296$
$w_{21} = 0.10559$	$w_{22} = -0.193636$

Aktualizace vah skryté vrstvy

Ten samý postup použijeme i pro aktualizaci vah skryté vrstvy.

$$\Delta v_{ij} = \varepsilon * \delta_j * x_i$$

Tabulka 3

NEURON 1	**NEURON 2**
$\Delta v_{01} = \varepsilon \cdot 0.03679 \cdot 1 = 0.03679 \cdot \varepsilon$	$\Delta v_{02} = \varepsilon \cdot (-0.01353) \cdot 1 = -0.01335 \cdot \varepsilon$
$\Delta v_{11} = \varepsilon \cdot 0.03679 \cdot 0.5 = 0.0184 \cdot \varepsilon$	$\Delta v_{12} = \varepsilon \cdot (-0.01353) \cdot 0.5 = -0.006765 \cdot \varepsilon$
$\Delta v_{21} = \varepsilon \cdot 0.03679 \cdot 0.3 = 0.011 \cdot \varepsilon$	$\Delta v_{22} = \varepsilon \cdot (-0.01353) \cdot 0.3 = -0.00406 \cdot \varepsilon$

Za učící parametr ε si opět dosadím 0.01 a podle rovnice $v_{ij} = v_{ij} + \Delta v_{ij}$ aktualizujeme váhy.

Tabulka 4

NEURON 1	**NEURON 2**
$v_{01} = 0.103679$	$v_{02} = 0.18665$
$v_{11} = -0.09816$	$v_{12} = 0.2993235$
$v_{21} = -0.0989$	$v_{22} = 0.399594$

A takto by vypadala jedna iterace této neuronové sítě (Dolezel, 2020). Abychom však dokázali, že se výsledná chyba opravdu zmenšuje, provedl jsem znovu výpočet dopředného šíření, tentokrát však s aktualizovanými váhami. Chyby e_1 a e_2 mi skutečně vyšly menší.

A to jsem chtěl dokázat. Chyba se opravdu zmenšila a kdybychom tento postup opakovali vícekrát, chyba by se ještě zmenšila.

$$e_1 = 0.064$$

$$e_2 = 0.021$$

5.4 Vícevrstvá neuronová síť v praxi

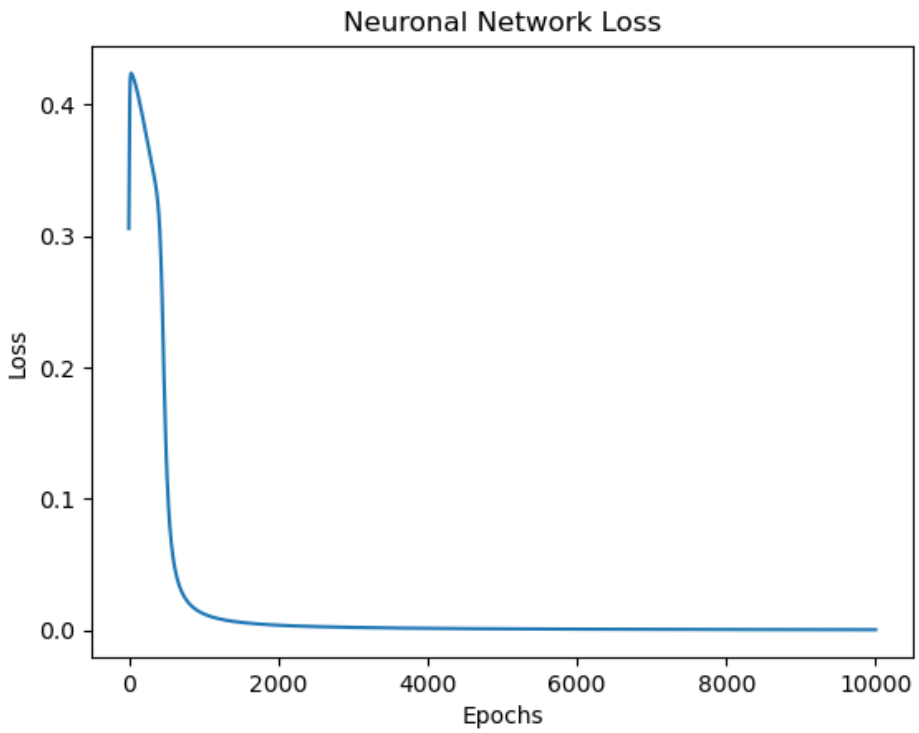
V předchozí kapitole o perceptronu jsme si ukázali jeho funkci a zároveň jeho možnosti využití. V druhém příkladu jsme zjistili, že je jeho využití značně omezeno a že si nedokáže poradit s nelineárním problémem XOR. Z tohoto důvodu si tento příklad zopakujeme a použijeme přitom vícevrstvou síť, jejíž fungování jsme si vysvětlili výše. Pro výpočet tohoto příkladu jsem použil programovací jazy Python, jehož výhodou byla knihovna zvaná „Numpy“, která mi pomohla s maticemi.

Abychom mohli pokračovat, musíme si opět nejdříve definovat tvar sítě. Pro zjednodušení jsem použil ten nejjednodušší možný, to znamená jedna vstupní vrstva, jedna skrytá vrstva a vrstva výstupní.

Jako vstupy trénovacích vzorků jsem použil vektory $(0 \quad 0), (0 \quad 1), (1 \quad 0), (1 \quad 1)$ a výstupní vzorky vektor $(0 \quad 1 \quad 1 \quad 0)$. V podstatě stejné trénovací vzorky, jako v předchozí kapitole. Jako přenosovou funkci jsem zvolil hyperbolický tangens. Proč, to si ukážeme později.

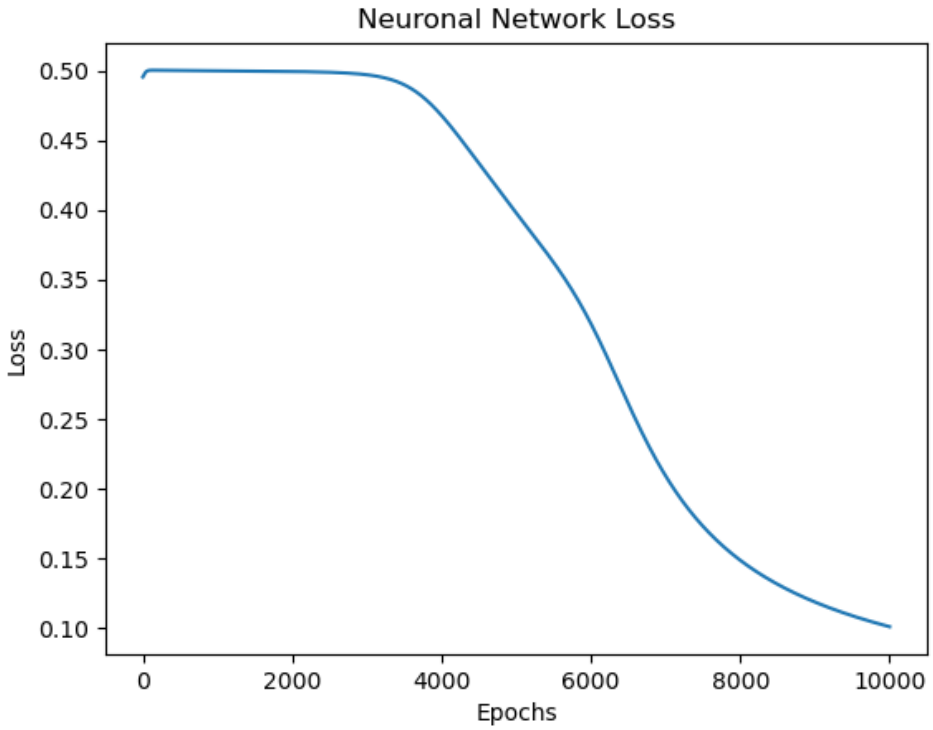
Počet opakování byl tentokrát 10 000. Po dosazení vektoru $(0 \quad 1)$ mi výsledek vyšel 0.999, to znamená odchylka 0.1 %, což je velice blízko ideální hodnotě.

Nyní se podíváme na graf, který nám ukáže, jak moc odchylka (angl. Loss) klesala v závislosti na počtu opakování (angl. Epochs).



Obrázek 9: Graf klesající chyby (Tanh)

Můžeme vidět, že od 2 000 opakování se odchylka takřka nezměnila a to znamená, že jsme mohli teoreticky celý proces učení zastavit u 2 000 opakování. Proč jsem to neudělal má ale důvod, a to ten, že bych Vám chtěl ukázat rozdíl ve zvolení správné přenosové funkce. Když se teď podíváme na stejný graf jen s tím rozdílem, že byla použita přenosová funkce sigmoid a ne hyperbolický tangens, zjistíme, že byl celý učící proces značně pomalejší a výsledek 0.905 byl také o poznání méně přesný.



Obrázek 10: Graf klesající chyby (Sigmoid)

Abychom si udělali představu, jak moc by bylo použití funkce sigmoid pomalejší, provedl jsem srovnání. Pro dosažení stejné míry přesnosti, v našem případě s odchylkou 0.01, potřebujeme u sítě s použitou funkcí sigmoid provést 10 074 opakování, kdežto s funkcí hyperbolický tangens pouze 527.

Vidíme tedy, že zvolení vhodné přenosové funkce se značně odráží na celkové efektivitě celé sítě.

6 ZÁVĚR

Cílem této práce bylo vytvořit funkční model neuronové sítě a prezentovat její funkčnost na příkladu. Tento cíl jsem splnil a sítě jsem vytvořil dvě. Jednu, na které jsem ukázal funkčnost jednovrstvé neuronové sítě, zároveň jsem odhalil i její nedostatky. A síť druhou, která právě nedostatky jednovrstvé sítě dokázala vyřešit. Jako příklad jsem použil problém XOR, který je známý tím, že ho nelze lineárně vyřešit. Abych se ale vůbec dostal k řešení tohoto problému, musel jsem nejprve vysvětlit fungování umělé neuronové sítě a všechny náležitosti k tomu patřící. Jelikož jsem nenašel dost užitečných zdrojů na téma umělých neuronových sítí v českém jazyce, musel jsem dohledat cizojazyčné zdroje. Šlo především o odborné vysokoškolské práce. A protože velká část výzkumů umělé inteligence probíhá v Německu, je i většina mých zdrojů v německém jazyce. Spolu s matematickým fungováním neuronové sítě jsem se zlehka podíval i do její historie, která, jak jsem zjistil, je mnohem delší, než jsem předpokládal. Další, pro mě náročný úkol, bylo neuronovou síť naprogramovat a na příkladu také otestovat.

Dle mého názoru jsem vytyčené cíle mé práce splnil, avšak musím přiznat, že celková náročnost tohoto tématu byla nad má očekávání. I přesto si ale myslím, že jsem do tématu do jisté míry pronikl a byl bych schopen s ním nadále pracovat. Největší překážkou bylo pro mne pochopit algoritmus zpětného šíření chyby, kde byla použita vyšší matematika, mně předtím neznámá. Nakonec jsem to však zvládl a mohl jsem dopsat i tuto kapitolu mé práce.

Řekl bych, že pro mě byla tato práce přínosem a jistě jím bude i pro každého, kdo si jí přečte. Jedná se o velice aktuální téma s velkým potencionálem, je tedy přínosné se mu podrobněji věnovat.

7 BIBLIOGRAFIE

Borzymowski, Henryk. 2019. *Automatische Textgenerierung für finanzielle Berichte.* Mnichov, Německo : Ludwig Maximilians Universität, 20. Leden 2019.

Dolezel, Petr. 2020. INUI2, NNUI2 - Přednáška (Dopředná vícevrstvá umělá neuronová síť). *YouTube.* [Online] 23. Březen 2020. https://www.youtube.com/watch?v=67yfzNWs4nA&t=1474s.

Ebert, David. 2019. *Bildklassifizierung mit Neuronalen Netzen.* Vitzenburg, Německo : Hochschule Merseburg, 1. Duben 2019.

Farkaš, Matin. 2019. *Realizace neuronové sítě s využitím grafických procesorů.* Plzeň : Západočeská univerzita, 2019.

Pircher, Thomas. 2017. místo neznámé : Hochschule Mittweida, 28. září 2017.

Ploner, Patrick Werner a Klaukien, Moritz. 2009. *Neuronale Netze.* Innsbruck, Německo : Universität Innsbruck, 14. Květen 2009.

Terhag, Felix. 2018. *Reinforcement Learning zur Erstellung.* Köln, Německo : Universität zu Köln, 3. Prosinec 2018.

Volná, Eva. 2002. *Neuronové sítě.* Ostrava, Česká republika : Ostravská univerzita, 2002.

8 SEZNAM OBRÁZKŮ A TABULEK

9 PŘÍLOHY

Součástí této práce bylo také vytvoření programu. Oba programy najdete na přiloženém usb flash disku, v případě elektronické verze této práce, ve složce programy.