SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16605-111403

ANALÝZA EFEKTIVITY PRUNINGOVÝCH ALGORITMOV

BAKALÁRSKA PRÁCA

2023 **Viet Nguyen Hoang**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16605-111403

ANALÝZA EFEKTIVITY PRUNINGOVÝCH ALGORITMOV

BAKALÁRSKA PRÁCA

Študijný program :	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Michal Hlavatý
Konzultant ak bol určený:	Ing. Michal Hlavatý

2023

Viet Nguyen Hoang

Slovenská technická univerzita v Bratislave
Ústav automobilovej mechatroniky

Fakulta elektrotechniky a informatiky
2022/2023

ZADANIE BAKALÁRSKEJ PRÁCE

Autor práce:	Viet Nguyen Hoang
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Evidenčné číslo:	FEI-16605-111403
ID študenta:	111403
Vedúci práce:	Ing. Michal Hlavatý
Vedúci pracoviska:	Ing. Ján Cigánek, PhD.
Názov práce:	**Analýza efektivity pruningových algoritmov**
Jazyk, v ktorom sa práca vypracuje:	slovenský jazyk
Špecifikácia zadania:	Cieľom práce je spracovať problematiku a vyhodnotiť jednotlivé algoritmy z ohľadom na ich efektivitu a prakticky dokázať jednotlivé výsledky analýzy na jednoduchých demonštračných príkladoch v jazyku Python. 1. Naštudujte a spracujte problematiku umelých neurónových sietí. 2. Naštudujte a spracujte problematiku pruningových algoritmov. 3. Implementujte jednotlivé algoritmy v praxi a vypracujte jednoduchý demonštračné príklady v jazyku Python. 4. Analyzujte a vyhodnoťte výsledky vypracovaných príkladov a efektivitu vybraných algoritmov
Termín odovzdania práce:	02. 06. 2023
Dátum schválenia zadania práce:	
Zadanie práce schválil:	

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program :	Aplikovaná informatika
Vyberte typ práce	Analýza efektivity pruningových algoritmov
Autor:	Viet Nguyen Hoang
Vedúci záverečnej práce:	Ing. Michal Hlavatý
Konzultant ak bol určený:	Ing. Michal Hlavatý
Miesto a rok predloženia práce:	Bratislava 2023

Umelá inteligencia a neurónové siete zaznamenali v poslednej dobe obrovský pokrok, či už z popularizačného hľadiska, ale aj z hľadiska výskumu a vývoja. Vďačíme tomu najmä výkonnému hardwareu ktorý sa stáva čoraz dostupnejší a aj výkonnejší. Na zefektívnenie trénovacieho procesu umelých neurónových sietí existujú aj iné spôsoby ako len vylepšiť výkon počítača. Jedným z týchto spôsobov je aj pruning, s ktorým je spojených niekoľko výhod, ale aj nevýhody.

Cieľom tejto bakalárskej práce bolo porovnať efektivitu rôznych pruningových algoritmov, za účelom zrýchlenia času trénovania neurónovej siete, zníženia jej veľkosti alebo počtu parametrov, pričom presnosť modelu by mala byť po pruningu v porovnaní s testovacími dátami zachovaná alebo klesla len minimálne. Práca sa v teoretickej časti zaoberá stručnou históriou umelých neurónových sietí, ich teóriou, typmi učenia a pruningovými metódami. V praktickej časti sa práca zaoberá implementáciou doprednej a konvolučnej neurónovej sieti, ich trénovaním a následným použitím rôznych pruningových algoritmov. V poslednej kapitole porovnávame dosiahnuté výsledky a efektivitu jednotlivých algoritmov. V tejto kapitole sa nachádzajú zdrojové kódy napísané v programovacom jazyku Python a grafy zobrazujúce výslednú efektivitu pruningových algoritmov. V záverečnej časti práce dosiahnuté výsledky analyzujeme a zhodnotíme.

Kľúčové slová: umelá neurónová sieť, pruning, efektivita pruningu, štruktúrovaný pruning, neštruktúrovaný pruning, globálny pruning, náhodný pruning

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Bachelor Thesis:	Efficiency analysis of pruning algorithms
Autor:	Viet Nguyen Hoang
Supervisor:	Ing. Michal Hlavatý
Consultant:	Ing. Michal Hlavatý
Place and year of submission:	Bratislava 2023

Artificial intelligence and neural networks have made tremendous progress in recent times, both in terms of popularization and research and development. This is mainly due to powerful hardware that is becoming more and more affordable and also more powerful. There are other ways to make the training process of artificial neural networks more efficient than just improving the performance of the computer. One of these ways is pruning, which has several advantages but also disadvantages associated with it.

The aim of this bachelor thesis was to compare the effectiveness of different pruning algorithms, in order to speed up the training time of the neural network, reduce its size or the number of parameters, while the accuracy of the model should be maintained or decreased only minimally after pruning compared to the test data. The theoretical part of the thesis deals with a brief history of artificial neural networks, their theory, types of learning and pruning methods. In the practical part, the thesis deals with the implementation of feedforward and convolutional neural networks, their training and the subsequent use of different pruning algorithms. In the last chapter, we compare the results obtained and the effectiveness of the different algorithms. This chapter contains source codes written in Python programming language and graphs showing the resulting

efficiency of the pruning algorithms. In the final part of the thesis, we analyze and evaluate the achieved results.

Vyhlásenie autora

Podpísaný Viet Nguyen Hoang čestne vyhlasujem, že som Bakalársku prácu Analýza efektivity pruningových algoritmov vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Uvedenú prácu som vypracoval pod vedením Ing. Michal Hlavatý.

V Bratislave dňa 09.06.2023

.....

podpis autora

Poďakovanie

Týmto by som sa rád poďakoval svojmu vedúcemu práce Ing. Michalovi Hlavatému za odborné vedenie, cenné pripomienky, rady, inšpirácie a za čas a ochotu, ktorú si našiel počas môjho vypracovania bakalárskej práce.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek a značiek

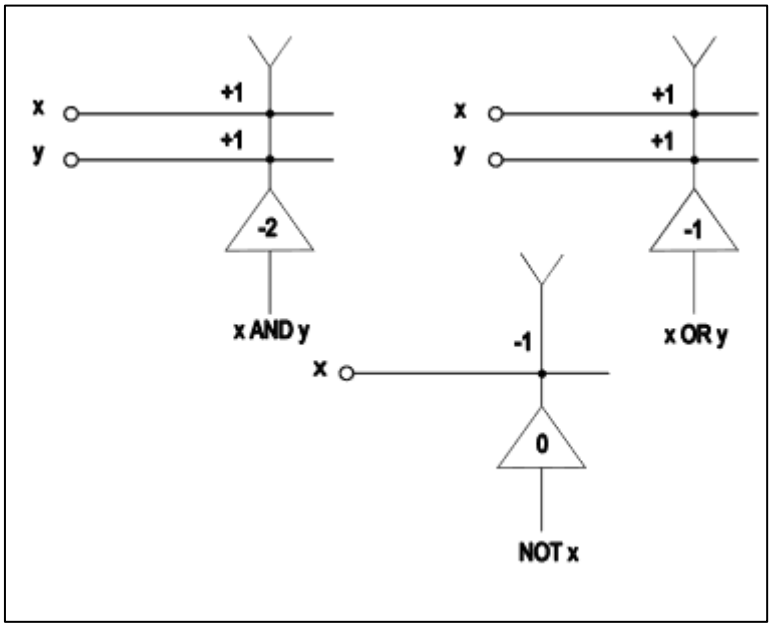
AI – umelá inteligencia

ANN – umelá neurónová sieť

Úvod

Umelá neurónová sieť, všeobecne známa ako neurónová sieť, je inšpirovaná štruktúrou a funkciou ľudského mozgu. Je navrhnutá tak, aby modelovala spôsob, akým ľudský mozog spracováva informácie a používa sa na vykonávanie úloh, ktoré sa zvyčajne spájajú s ľudskou inteligenciou. História neurónových sieti siaha do 40. a 50. rokov 20. storočia, keď výskumníci prvýkrát začali rozmýšľať nad použitím počítačových modelov na napodobenie štruktúry a funkcie ľudského mozgu [1].

Prvý koncept neurónových sietí predstavili neurofyziológ Warren McCulloch a matematik Walter Pitts. V roku 1943 napísali článok o tom, ako by mohli fungovať neuróny v ľudskom mozgu. Na ich opis navrhli jednoduchý model neurónu (neurónová sieť), ktorý sa dal využiť na vykonávanie základných logických operácií (Obr.1). Táto myšlienka položila základy pre budúci výskum v oblasti ANN [2].

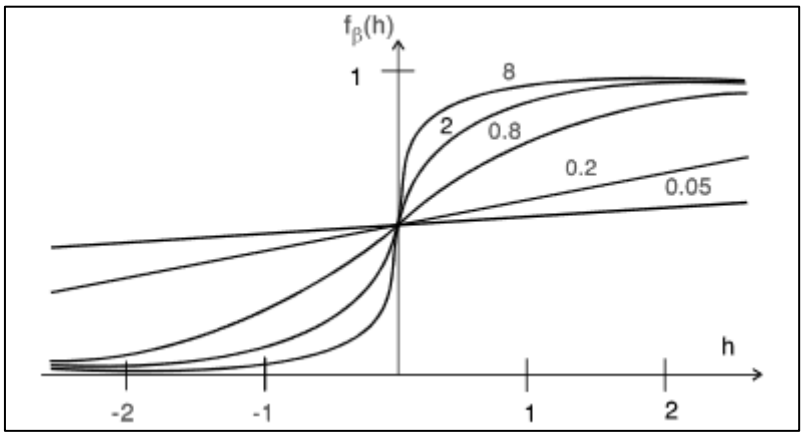


Obrázok 1: príklad McCullochových-Pittsových neurónov [1]

V nasledujúcich rokoch mnohí výskumníci významne prispeli do oblasti umelých sieti. Jedny zo známejších boli Frank Rosenblatt, Bernard Widrow a Ted Hoff, ktorí vyvinuli prvé prototypy neurónových sietí. Rosenblatt v roku 1958 svetu predstavil „perceptrón“, ktorý vznikol pomocou natrénovania McCullochove-Pittsove siete tak, aby vedeli rozpoznávať a klasifikovať objekty. Bola to jednovrstvová sieť, ktorá dokázala

vykonávať jednoduché binárne klasifikačné úlohy, ako napríklad rozpoznávame ručne písaných číslic. V roku 1960 Widrow a Hoff taktiež pomocou McCullochove-Pittsove siete vyvinuli neurónovú sieť a pomenovali ju ADELINE (ADAptive LInear NEuron), kde prvý krát ukázali minimalizáciu globálnej funkcie systému počas jej učenia. Aj keď sa tieto prototypy považovali za významný míľnik v oblasti ANN, čoskoro sa zistili jej obmedzenia. Boli schopné riešiť lineárne separovateľné problémy (AND, OR), avšak nedokázali vyriešiť lineárne neseparovateľné problémy (XOR), čo viedlo k stagnácii vo výskume na niekoľko desaťročí [1].

Až v roku 1986 sa vyriešil tento problém zavedením pravidla učenia metódou spätného šírenia sa chýb pre viacvrstvové perceptróny, ktorých autormi boli David Rumelhart, Geoffrey Hinton a Ronald J. Williams. Avšak z jednoduchého modelu neurónu McCullochovho-Pittsovho typu sa stal zložitejší analógový element, ktorý častokrát mal tvar sigmoidálu, obsahujúci spojitú vstupno-výstupnú funkciu [1].



Obrázok 2: príklad analógového elementu sigmoidálneho tvaru [1]

Bol to kľúčový moment vo vývoji ANN, pretože umožnil sieťam sa učiť z vlastných chýb a postupne zlepšovať svoj výkon.

V 80. a 90. rokoch 20. storočia sa o oblasť ANN opäť zvýšil záujem, predovšetkým vďaka pokroku v oblasti počítačového hardvéru a dostupnosti veľkého množstva dát. Bolo vyvinutých množstvo nových architektúr, napríklad rekurentné neurónové siete a konvolučné neurónové siete, ktoré umožnili neurónovým sieťam vykonávať zložitejšie úlohy, napríklad rozpoznávanie obrazu či reči a spracovanie prirodzeného jazyka [1].

V posledných rokoch zaznamenali neurónové siete rapídny rast a vývoj, ktorý bol spôsobený dostupnosťou veľkého množstva dát a výkonných výpočtových zdrojov. V súčasnosti sa neurónové siete stali špičkovou technológiou, ktorá v spôsobila revolúciu v oblasti umelej inteligencie. Ako dobre vieme AI aplikácie sú všade okolo nás a čoraz častejšie sa stávajú súčasťou nášho každodenného života. Vďaka ANN a AI máme možnosť s dostatočným množstvom trénovacích dát vygenerovať obrazy, fotky, hudby či texty, ktoré sú na štýl vstupných dát, ako napríklad maľby na štýl najpopulárnejších renesančných maliarov. Taktiež v poslednej dobe sa stalo populárne generovanie zvuku a hlasu ľudí, aj zosnulých, a následne ich použitie na dabovanie textu, čo v niektorých prípadoch môže mať nebezpečné následky. Ďalšom veľmi populárnou aplikáciou sa stal ChatGPT od spoločnosti OpenAI, ale taktiež ďalší chatboti. ChatGPT na základe vstupu generuje odpovede podobné ľudským, ktoré sú častokrát presné, vďaka čomu sa stál rýchlejšou alternatívou hľadania odpovedí na internete, hlavne medzi študentami, ale aj programátormi.

V prvej časti tejto práce sa budeme venovať teoretickému opisu umelej neurónovej sieti, typom základných architektúr a typom učenia. V ďalšej časti opíšem pruning, na čo slúži a rôzne typy pruningových metód. V poslednej časti tejto práce porovnáme rôzne pruningové algoritmy. Budeme porovnávať ich efektivitu, časovú náročnosť, presnosť v porovnaní pred použitím pruningu a v záverečnej časti vyhodnotíme dosiahnuté výsledky.

1 Umelá neurónová sieť

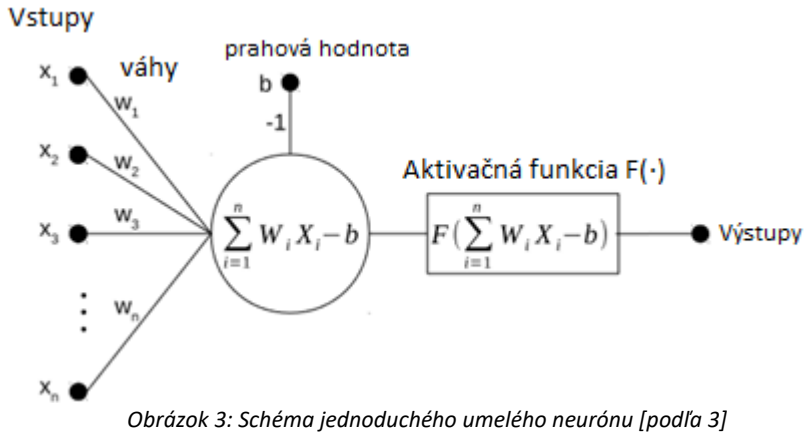
ANN je výpočtový model inšpirovaný štruktúrou a funkciou ľudského mozgu. Je navrhnutý tak, aby napodobňoval schopnosti ľudského mozgu učiť sa a spracúvať informácie, ktoré využíva na riešenie zložitých problémov. Cieľom neurónovej siete je učiť sa a zovšeobecňovať vzory zo vstupných údajov, vďaka čomu je obzvlášť užitočná na rozpoznávanie vzorov, klasifikáciu, predikciu alebo generovanie kreatívnych výstupov na základe prijatých vstupov, ako sú obrazy, fotky, texty, hudba, ...

Vo všeobecnosti pozostáva neurónová sieť z 3 typov vrstiev: vstupná, skrytá a výstupná vrstva, so vzájomne prepojenými uzlami, známymi aj ako umelé neuróny, ktoré prijímajú a spracovávajú vstupy a posielajú ich ďalej sieťou. Vrstvám neurónovej sieti sa bližšie budem venovať v kapitole 1.3.

Neurónové siete sú navrhované tak, aby rozpoznali vzory a robili rozhodnutia na základe vstupných dát. Sú dôležitým základom modernej umelej inteligencie a strojového učenia.

1.1 Umelý neurón

Každý umelý neurón prijíma vstupy, spracúva ich a výstupy sa potom odovzdajú ako vstup ďalším neurónom, až kým sa nezíska konečný výstup siete. Každý umelý neurón vykoná na vstupoch jednoduchý výpočet, ktorý sa dá opísať:



Obrázok 3: Schéma jednoduchého umelého neurónu [podľa 3]

Schéma pozostáva z n vstupov x_1 až x_n, ktoré sú prepojené pomocou váh s koeficientami w_1 až w_n. Váha prepojenia určuje, aký má vstup vplyv na výslednú aktivitu neurónu. Funkcia F je aktivačná funkcia, ktorej výsledok je výsledná aktivita neurónu.

Hodnota b je prah excitácie, ktorý môže byť aj konštanta, slúžiaci na posunutie aktivačnej funkcie smerom ku kladným alebo záporným hodnotám podľa potreby.

Umelý neurón získava vstupné signály z predchádzajúcich vrstiev alebo rovno zo vstupných údajov. Vstupu reprezentujú určité vzory alebo vlastnosti spracovaných údajov, ktoré následne dostanú svoju váhu, ktorá znázorňuje ich vplyv na výstupoch.

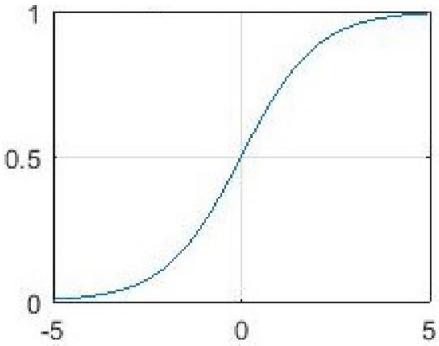
1.2 Aktivačné funkcie

Aktivačná funkcia určuje level aktivity neurónu na základe váženého priemeru súčtov a prahu excitácie, na základe čoho sa rozhodne, či sa neurón aktivuje alebo nie. Sigmoid, Tanh alebo ReLU sú aktivačné funkcie, ktoré umožňujú modelovanie aj zložitých vzťahov [4].

S/N	Funkcia	Výpočtová rovnica
1	Sigmoid	$f(x) = \left(\frac{1}{(1+exp^{-x})} \right)$
2	HardSigmoid	$f(x) = max\left(0, min\left(1, \frac{(x+1)}{2}\right)\right)$
3	SiLU	$a_k(s) = z_k \alpha(z_k)$
4	dSiLU	$a_k(s) = \alpha(z_k)(1 + z_k(1 - \alpha(z_k)))$
5	Tanh	$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right)$
6	Hardtanh	$f(x) = \begin{pmatrix} -1 & if\ x < -1 \\ x\ if\ -1 = & x \leqslant 1 \\ 1 & if\ x > 1 \end{pmatrix}$
7	Softmax	$f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$
8	Softplus	$f(x) = \log(1 + exp^x)$
9	Softsign	$f(x) = \left(\frac{x}{\lvert x \rvert + 1} \right)$
10	ReLU	$f(x) = max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$
11	LReLU	$f(x) = \alpha x + x = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases}$
12	PReLU	$f(x_i) = \begin{pmatrix} x_i\ , & if\ x_i > 0 \\ a_i x_i\ , & if\ x_i \leq 0 \end{pmatrix}$
13	RReLU	$f(x_i) = \begin{pmatrix} x_{ji}\ , & if\ x_{ji} \geq 0 \\ a_{ji} x_{ji}\ , & if\ x_{ji} < 0 \end{pmatrix}$
14	SReLU	$f(x) = \begin{pmatrix} t_i^r + a^r{}_i (x_i - t^r{}_i), & x_i \geq t^r{}_i \\ x_i\ , & t_i^r > x_i > t_i^l \\ t^l{}_i + a^l{}_i \left(x_i - t^l{}_i\right), & x_i \leq t^l{}_i \end{pmatrix}$
15	ELU	$f(x) = \begin{pmatrix} x, & if\ x > 0 \\ \alpha\ exp(x) - 1, & if\ x \leq 0 \end{pmatrix}$
16	PELU	$f(x) = \begin{pmatrix} cx, & if\ x > 0 \\ \alpha\ exp^{\frac{x}{b}} - 1, & if\ x \leq 0 \end{pmatrix}$

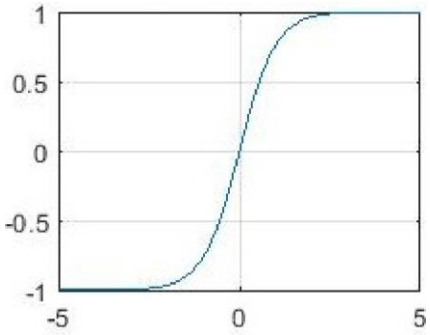
Obrázok 4: populárne druhy aktivačných funkcií [podľa 4]

- **Sigmoid** – výsledkom je hodnota medzi 0 a 1, je zvyčajne používaná pre modely, kde potrebujeme predpovedať pravdepodobnosť, vďaka tomu, že pravdepodobnosť je len v rozsahu od 0 do 1. Najčastejšie používaná v dopredných sieťach [4].



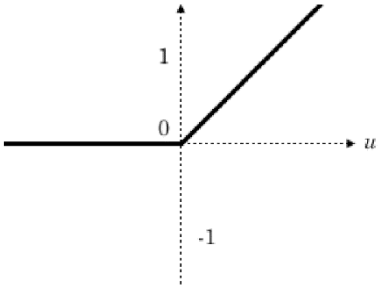
Obrázok 5: funkcia sigmoid [4]

- **Tanh** – výstupom funkcie je centrovaný okolo 0, ktorej hodnoty sú od -1 do 1, vďaka čomu môžeme hodnoty začleniť do skupín negatívne, neutrálne, pozitívne. Stala sa preferovanou funkciou v porovnaní so sigmoid vo viacvrstvových sieťach, vďaka jej lepšiemu tréningovému výkonu [4].



Obrázok 6: funkcia tanh [4]

- **ReLU** – najčastejšie používaná aktivačná funkcia pre aplikácie hlbokého učenia s doteraz dosiahnutými skvelými výsledkami. Je jednoduchá na použitie, efektívna a menej výpočtovo náročná. Výpočet ReLU je veľmi jednoduchý, keďže zahŕňa len porovnávanie jeho vstupov s hodnotou 0, vďaka čomu je potrebných menej matematických výpočtov v porovnaním so Sigmoid alebo Tanh aktivačnými funkciami. ReLU pomáha prechádzať exponenciálnemu rastu výpočtových požiadaviek potrebných na prevádzku neurónovej siete, keďže s rastúcou veľkosťou neurónovej siete, výpočtové požiadavky na pridávanie ďalších ReLU rastú lineárne [4].

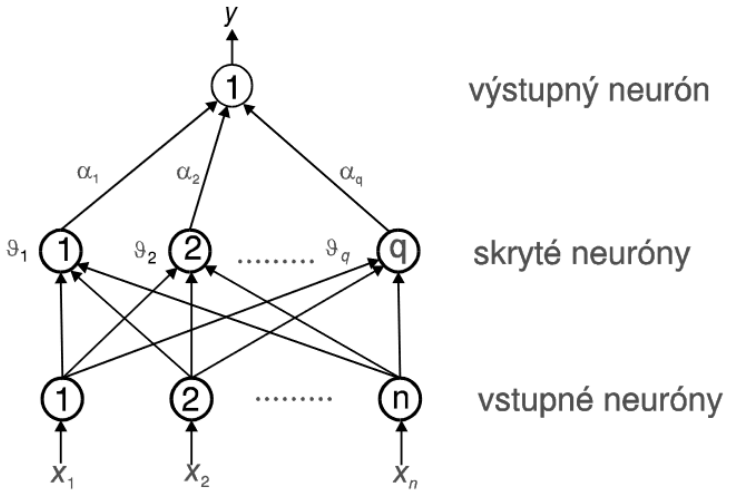


Obrázok 7: funkcia ReLU [8]

Aj keď Sigmoid, Tanh a ReLU patria medzi najviac používané aktivačné funkcie, stále majú svoje nedostatky, preto existujú rôzne variácie a pred použitím jednej z nich je odporúčané si preštudovať ich použitie [4].

1.3 Vrstvy neurónovej siete

ANN pozostávajú z vrstiev obsahujúcich umelé neuróny. Prvá vrstva sa nazýva **vstupná** vrstva a pozostáva z neurónov pre každé vstupné dáta do neurónovej siete. Táto vrstva prijíma vstupné dáta a posiela ich ďalej sieťou. Posledná vrstva je **výstupná** vrstva a pozostáva z neurónov pre každý výstup. Výstupná vrstva obsahuje výsledok alebo výstup riešeného problému. Vrstvy medzi nimi sa nazývajú **skryté** vrstvy, keďže počas trénovania nepoznáme ich pravé hodnoty, len hodnoty vstupov a výstupov. Umožňujú neurónovej sieti sa učiť zložité úlohy a dosiahnuť vynikajúce výsledky. Práve skryté vrstvy sú zodpovedné za vynikajúci výkon a zložitosť neurónových systémov.



Obrázok 8: znázornenie štandardnej neurónovej siete s dopredným šírením, ktorá obsahuje jednu vrstvu skrytých neurónov [1]

Hlavným účelom skrytých vrstiev je extrahovať informácie zo vstupných údajov, nájsť zložité vzory a základné štruktúry a následne ich poslať do ďalšej vrstvy, vďaka čomu sa môže neurónová sieť naučiť zovšeobecňovať z trénovacích údajov a robiť presné predikcie alebo klasifikácie na doposiaľ nepozorovaných údajoch. Vrstvy sú hierarchicky rozdelené, nižšie vrstvy neurónových sietí častokrát zachytávajú len jednoduché vzory, vyššie vrstvy sa učia ich kombinovať, vďaka čomu dokážu robiť predpovede a klasifikácie. Podľa zložitosti riešeného problému môžeme meniť počet a veľkosť skrytých vrstiev. Najjednoduchším úlohám častokrát postačuje len jedna skrytá vrstva, pridaním skrytých

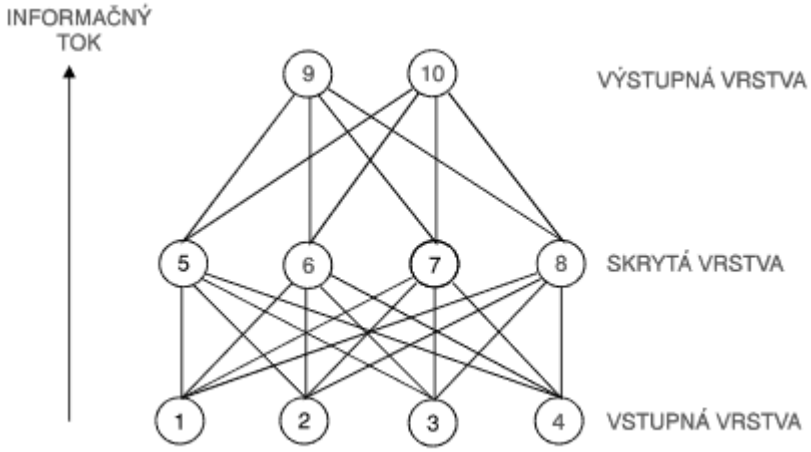
vrstiev umožníme neurónovej sieti modelovať aj veľmi zložité problémy s dostačujúco presnými výstupmi, pričom zvýšením počtu skrytých vrstiev môže dôjsť k zvýšeniu výpočtových požiadaviek.

1.4 Architektúra neurónovej siete

Pod pojmom architektúra neurónovej siete môžeme chápať celkové usporiadanie a organizáciu siete. Zahŕňa usporiadanie vrstiev, počet prepojení medzi neurónmi a tok informácií v sieti. Podľa zložitosti a rozsiahlosti riešeného problému je dôležitý vhodný návrh architektúry pre dosiahnutie čo najlepšieho výkonu a efektívneho riešenie daného problému. Medzi najzaužívanejšie architektúry patria architektúra s dopredným šírením (feed-forward), rekurentná a konvolučná architektúra [5].

1.4.1 Architektúra s dopredným šírením (Feed-forward network)

Taktiež známa ako viacvrstvový perceptrón (multi-layer perceptrons), je jedna z najznámejších a najviac používaných architektúr. Skladá sa zo vstupnej vrstvy, jednej alebo viacerých skrytých vrstiev a výstupnej vrstvy. Tok informácií ma pevne daný smer, od vstupnej vrstvy do skrytých vrstiev, až po výstupnú. Výhodou takejto siete je jednoduchosť architektúry, efektívne rozpoznávanie vzorov, vďaka čomu excelujú v úlohách, v ktorých je potrebné rozpoznávanie vzorov, klasifikácia či regresia [6].



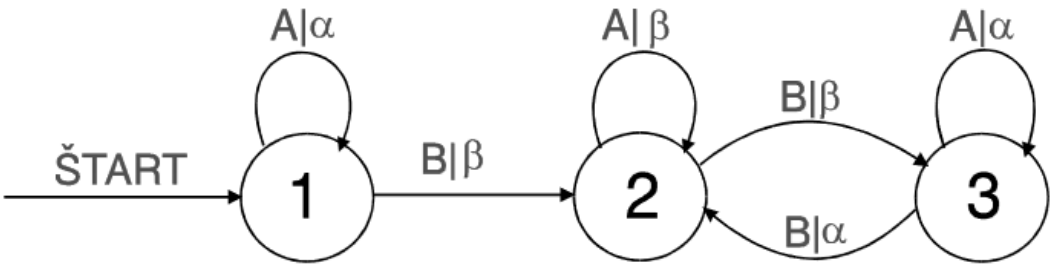
Obrázok 9: znázornenie trojvrstvovej neurónovej siete s dopredným šírením [1]

Keďže informačný tok je pevné daný, dopredné neurónové siete nemajú možnosť uchovania minulých vstupov, čo ich obmedzuje pri úlohách, kde je potrebné modelovanie

časových údajov alebo sekvenčných modelov. Napriek výhodám používania sietí s dopredným šírením, v určitých prípadoch existujú obmedzenia. Siete s dopredným šírením na správne fungovanie vyžadujú veľké množstvo údajov. S rastúcou zložitosťou úloh taktiež rastie aj počet skrytých vrstiev, čo má za následok zvýšenie výpočtových nárokov.

1.4.2 Rekurentné neurónové siete (Recurrent network)

Obsahujú dva spôsoby šírenia signálu. Prvý je rovnaký ako pri dopredných sieťach, druhý umožňuje, aby sa signál z výstupnej vrstvy neurónov vrátil naspäť do skrytej vrstvy alebo dokonca do vstupnej vrstvy, čo umožňuje zapamätanie si údajov z predchádzajúcich iterácií.. Taktiež obsahuje spojenia, ktoré umožňujú cyklický tok informácií v sieti, vďaka čomu na rozdiel od dopredných sietí majú schopnosť modelovať úlohy aj s časovou štruktúrou vďaka svojej schopnosti zachytiť sekvenčné závislosti a časové vzory. Kľúčom rekurentných sietí je rekurentný neurón, ktorý má spojenie vytvárajúce cyklické spojenie. Tieto spojenia umožňujú neurónom spracovávať a zachytávať časovo závislé alebo sekvenčné údaje, časové závislosti a vzory [5].



Obrázok 10: príklad rekuretnej neurónovej siete s rekurentným neurónom [1]

Rekurentné siete sú častokrát zložitejšie ako siete s dopredným šírením, tým pádom sú aj výpočtovo zložitejšie.

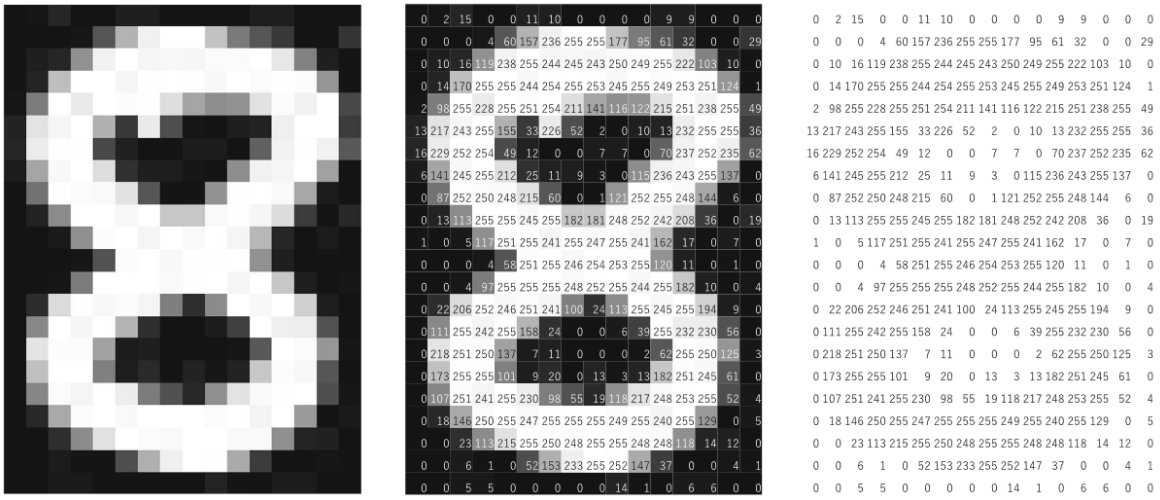
Dopredné a rekurentné neurónové siete sú univerzálnym aproximátorom, vďaka ich vynikajúcej adaptívnej schopnosti, vďaka ktorej môžu aprximovať ľubovoľné nelineárne dynamické systémy s požadovanou presnosťou [6].

1.4.3 Konvolučné neurónové siete (Convolutional network)

Konvolučné neurónové siete špeciálne navrhnuté na spracovanie a analýzu vizuálnych vstupov. Stali sa jedným z najpopulárnejších nástrojov na rozpoznávanie obrazov a tvárí, detekciu objektov a ďalších oblastí počítačového videnia. Konvolučná ANN sa zväčša skladá z 3 vrstiev: konvolučná, pooling a lineárnej vrstvy.

Konvolučná vrstva je základnou zložkou konvolučnej architektúry. Vykonáva extrakciu príznakov, ktorá zvyčajne pozostáva z kombinácie lineárnych a nelineárnych operácií, t.j. konvolučné operácie a aktivačné funkcie.

Na vstupe je vizuálny zdroj a po ňom nasleduje niekoľko po sebe idúcich konvolučných vrstiev. Vrstvy sa skladajú z množiny filtrov, ktoré sa používajú na extrakciu príznakov prostredníctvom operácie konvolúcie. Konvolúcie je sčítanie hodnôt zo vstupu vynásobených hodnotami filtra. Ich hlavnou úlohou je zachytávanie vzorov a prvkov, ako sú hrany, textúry a tvary.



Obrázok 11: premena vstupného zdroja na maticu čísel [9]

Pooling slúži na znižovanie vzorkovania, pomáha znižovať priestorové rozmery máp prvkov vytvorené konvočnými vrstvami. Vrstvy poolingu sa zvyčajne používajú na postupné znižovanie vzorkovania, čím zmenšujú veľkosť máp a zároveň zachovávajú dôležité informácie.

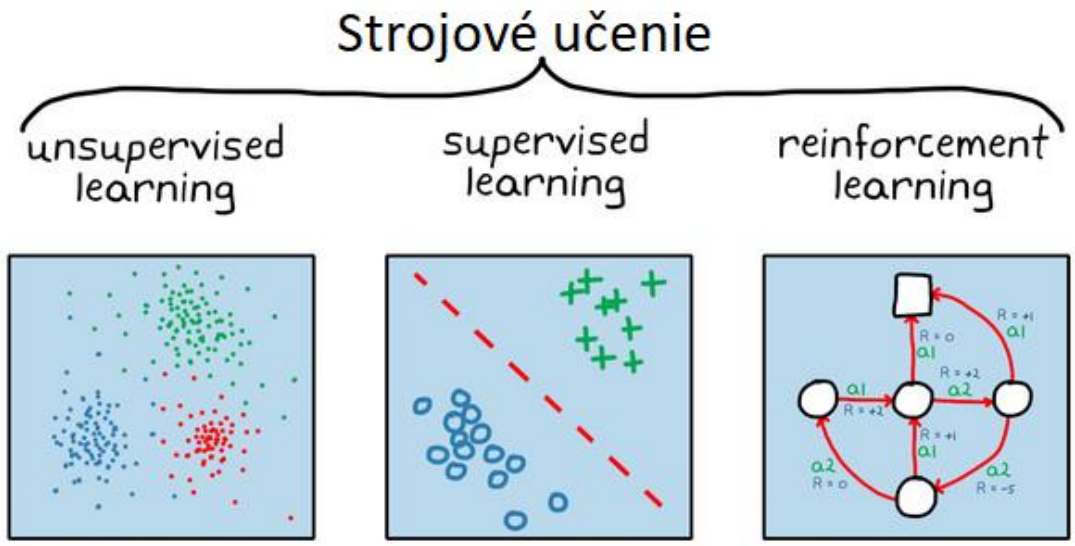
Výstupná vrstva je lineárna vrstva. Jej úlohou je namapovať naučené vzory predchádzajúcimi vrstvami a poslať ich na výstup s požadovanými predpoveďami. Na výstupnej vrstve sa používa Softmax aktivačná funkcia, slúži na pravdepodobnostné

rozdelenie tried. Trieda s najvyššou pravdepodobnosťou sa vyberie ako predpovedaná výsledná trieda. Podporuje sieť, aby priraďovala vysoké pravdepodobnosti správnym triedam a zároveň minimalizovala pravdepodobnosti nesprávnych tried.

Konvolučné ANN sú dominantné v rôznych oblastiach počítačového videnia, pretože sú špeciálne navrhnuté na spracovanie a extrakciu zmysluplných funkcií z vizuálnych údajov, ako sú obrázky a videá [9].

1.5 Typy učenia

Typ učenia neurónovej siete zohráva dôležitú rolu počas trénovania neurónovej siete na efektívne učenie a presné výsledky. Učenie slúži na optimalizáciu váh a prahu excitácie, minimalizovanie chýb a zovšeobecnenie údajov.



Obrázok 12: typy strojového učenia [15]

1.5.1 Učenie s učiteľom (Supervised learning)

Učenie s učiteľom je jedna z najpoužívanejších učiacich metód. Každý vstup je spojený s príslušným cieľovým výstupom. Počas trénovania sa upravujú dáta a skresľuje sieť tak, aby sa minimalizoval rozdiel medzi predpovedaným výstupom neurónovej siete a skutočným výstupom. Najpoužívanejšími algoritmami na implementáciu učenia s učiteľom je metóda adaptácie pomocou spätného šírenia chyby (backpropagation)

v kombinácii s gradientovým spádom (gradient descent), ktorý vypočíta grardient chyby vzhľadom na jej parametre a upraví príslušné váhy a prahy excitácie.

1.5.2 Učenie bez učiteľa (Unsupervised learning)

Metóda učenia bez učiteľa sa používa v prípade malého množstva porovnateľných údajov alebo keď nie sú vôbec k dispozícii. Algoritmus učenia nemá informáciu o požadovaných aktivitách výstupných neurónov v priebehu trénovania o ktoré by sa mohol opierať. Adaptácia váh odzrkadľuje štatistické vlastnosti trénovacej množiny, hľadaním vzorov, vzťahov alebo štruktúr vo vstupných údajoch. Medzi obľúbenou technikou učenia bez učiteľa je SOM (SamoOraganizujúca sa Mapa), umožňuje realizovať zobrazenie zachovávajúce topológiu a zobraziť tak charakteristické príznaky trénovacej množiny dát [1].

1.5.3 Reinforcement learning

Reinforcement learning (učenie s posilňovaním) je typ učenie, kde agent vie interagovať so svojím prostredím. Agent je typ programu, ktorý sa môže rozhodovať na základe prostredia alebo vstupov. Učenie funguje na báze odmeňovania požadovaného chovania alebo rozhodnutí a potrestania nepožadovaných. Agent interaguje s prostredím a učiť sa na základe pokusov a omylov. Vďaka tomu, že agent je odmeňovaný za požadované chovanie, snaží sa získať maximálnu odmenu, vďaka čomu minimalizuje alebo sa úplne vyhne nepožadovaným akciám.

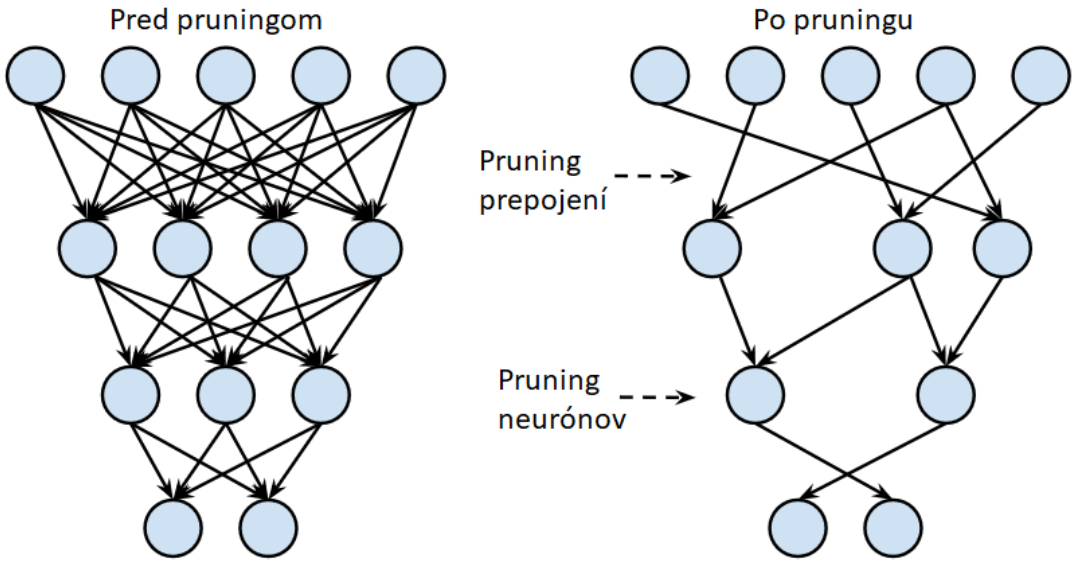
V súčasnosti máme prístup k veľkému množstvu dát, čo pre trénovanie neurónových sietí znamená veľké množstvo vstupných údajov, vďaka ktorým ich vieme čoraz presnejšie natrénovať, aby výsledné dáta spĺňali naše požiadavky. Najčastejšie je každý neurón pomocou váh prepojený s každým neurónom, ale čím viac ma neurónová sieť vstupných neurónov a neurónov v skrytej vrstve, tým viac samotný proces trénovania môže byť časovo a nákladovo náročný. To môže obmedziť ich využitie v rôznych aplikáciách a zariadeniach. Riešením tohto problému môže byť technika známa ako „pruning“.

2 Pruning

Počas používania ANN sa vyskytuje niekoľko problémov, ktoré vedú k zníženiu generalizačným schodnostiam siete. Ak voľba topológie siete nie je vhodná, môže dôjsť k niekoľkým problémom [5]:

- čas učenia je príliš dlhý
- vysoké výpočtové nároky
- nadmerné učenie (overfitting)
- veľa lokálnych miním
- veľký objem parametrov alebo dát

Pruning je technika používaná na odstránenie menej dôležitých prepojení či celých neurónov v neurónovej sieti, čo vedie k zjednodušeniu siete a zníženiu jej veľkosti. Hlavným cieľom pruningu je zachovanie presnosti trénovaných dát a zároveň zredukovať jej zložitosť. Pruning môže byť použitý počas trénovania alebo po ňom. Existujú rôzne typy pruningových algoritmov, každý so svojimi výhodami a nevýhodami na základe použitia [7].



Obrázok 13: príklad pruningu v umelej neurónovej sieti [podľa 7]

2.6 Váhový pruning

Váhový pruning slúži na odstránenie váh, ktoré sú blízko k 0, čo vedie k redukcii zložitosti a počtu parametrov neurónovej siete, vďaka čomu je konečná sieť jednoduchšia a umožňuje ľahšie nasadenie do aplikácii.

Je to jednoduchá a efektívna metóda redukcie veľkosti siete, ktorá môže zlepšiť zovšeobecnenie siete. Bohužiaľ nemusí zakaždým viesť k výraznej redukcii zložitosti siete. Môže viesť k veľkej strate presnosti, ak dôjde k agresívnemu pruningu. V prípade nastavenie vysokej intenzity pruning počas váhového pruningu, môže dôjsť k odstráneniu váh, ktoré bolo dôležitou súčasťou siete [13].

2.7 Jednotkový pruning

Jednotkový pruning odstraňuje neuróny či celé vrstvy zo siete. Vedie to k výraznému zníženiu počtu parametrov v neurónovej sieti a môže viesť k rýchlejšej a efektívnejšej sieti.

Metóda štruktúrovaného pruningu môže výrazne znížiť zložitosť siete a prispieť k lepšie interpretovateľnému modelu, ale táto metóda si vyžaduje zložitejší proces určenia odstránenia parametrov, čo môže byť časovo a nákladovo náročnejšie. Odstránenie celých vrstiev či neurónov taktiež môže vieš k redukcii presnosti, ak boli dôležitou súčasťou siete [13].

2.8 Veľkostný pruning

Veľkostný pruning odstraňuje prepojenia alebo neuróny na základy prahovej hodnoty. Po natrénovaní siete sa neurónom priradí absolútna hodnota na základe ich váhy. Ak ich váha je menšia ako prahová hodnota, tak budú odstránené.

Je veľmi jednoduchý a efektívny, ale ak je prahová hodnota nastavená príliš vysoko, môže dôjsť k agresívnemu pruningu, čo spôsobí odstránenie veľkého množstva neurónov, čo môže zapríčiní stratu presnosti, ak sa odstránili neuróny s dôležitými vzormi [13].

2.9 Štruktúrovaný pruning

Štruktúrovaný pruning odstráni celé štruktúry, myšlienkou je odstrániť skupiny váh alebo neurónov, ktoré sú pre výkon siete menej dôležité.

Môže viesť k jednoduchšej a výkonnejšej sieti, avšak môže vyžadovať viac výpočtových zdrojov, čo vedie k zvýšeniu nákladov. Taktiež môže dôjsť k zníženiu flexibility, pretože sa odstraňujú celé skupiny váh alebo neurónov [13].

2.10 Náhodný pruning

Náhodne sa zvolia neuróny na odstránenie. Veľmi jednoduchá implementácia, ale keďže ide o náhodné odstránenie parametrov, môže zlepšiť či zhoršiť efektivity, výkon alebo presnosť [13].

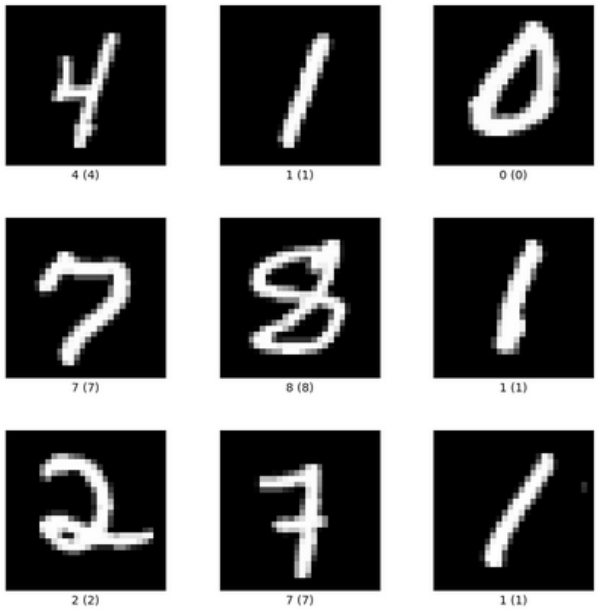
3 Analýza efektivity pruningových algoritmov

V tejto časti bakalárskej práce sa budeme venovať analýze efektivity pruningových algoritmov. Ako programovací jazyk využívame Python 3.11 s vývojovým prostredím IDLE. V našom kóde používame verejne dostupný framework Pytorch 2.0, obsahujúci knižnice na vytvorenie architektúry, trénovanie a testovanie dát, pruning, atď., ktorým sa budeme hlbšie venovať počas opisu implementácie nášho kódu.

Na vybraných dátach natrénujeme neurónovú sieť, aby sme mali základné údaje o konečnej presnosti, dĺžke trénovania a počtu parametrov. Potom na nenatrénovanú sieť aplikujeme rôzne pruningové algoritmy s rôznymi intenzitami pruningu, natrénujeme sieť a výsledky porovnáme s originálnou sieťou.

3.11 Voľba dátového balíka a architektúry

Ako vstupný súbor údajov sme si zvolili MNIST dátový balík, obsahujúci ručne písané číslice. Súbor údajov obsahuje trénovaciu sadu 60 000 príkladov a testovaciu sadu 10 000 príkladov. Každý obrázok má rozmery 28x28 pixelov, čo predstavuje celkovo 784 pixelov na obrázok. Obrázky sú uložené ako 8-bitové hodnoty sivej, kde intenzita každého pixelu je reprezentovaná jedným bitom v rozsahu od 0 (čierna) do 255 (biela).



Obrázok 14: číslice MNIST dátový balík [10]

Tento dátový balík sme si zvolili kvôli jeho jednoduchosti a dostupnosti vďaka čomu sa urýchli trénovací a testovací proces a zároveň dostatočnému počtu údajov, pomocou ktorých sme schopní ukázať efektivitu rôznych pruningových algoritmov.

```
from torchvision.datasets import MNIST
from torch.utils.data import DataLoader

# Define the transform for the data
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Load the dataset
trainset = MNIST(root='./data', train=True, download=True, transform=transform)
testset = MNIST(root='./data', train=False, download=True, transform=transform)

# Define the dataloaders
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64, shuffle=True)
```

Obrázok 15: načítanie trénovacej a testovacej sady z MNIST súboru údajov

Neurónové siete, na ktorých budeme testovať pruningové algoritmy sú: jednoduchá dopredná neurónová a konvolučná neurónová sieť.

3.12 Neuronová sieť s dopredným šírením

Dopredná neurónová sieť obsahuje vstupnú vrstvu o veľkosti 28x28, čo zodpovedá 784 vstupným pixelom obrázka. Potom obsahuje 2 skryté vrstvy so 100 vstupnými neurónmi, ReLU aktivačnú funkciu a nakoniec výstupnu vrstu s 10 neurónmi, keďže budeme čísla klasifikovať do skupín od 0 do 9.

Skladá sa zo vstupnej vrstvy, jednej alebo viacerých skrytých vrstiev a výstupnej vrstvy. Tok informácií ma pevne daný smer, od vstupnej vrstvy do skrytých vrstiev, až po výstupnú .

```
import torch.nn as nn

# Define the FNN model
class FNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(FNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_dim, hidden_dim)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        return out

modelFNN = FNN(28*28, 100, 10)
```

Obrázok 16: zadefinovanie architektúry s dopredným šírením [12]

3.12.1Stratová a optimizačná funkcia a trénovanie modelu

```
# Train the model
def trainModel(model):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01)
    model.train()
    trainStart = time.time()
    for epoch in range(5):
        for images, labels in trainloader:
            images = images.view(-1, 28*28).requires_grad_()
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
        print(f'Epoch {epoch+1}: training loss: {loss.item()}')

    trainEnd = time.time()
    trainTotal = trainEnd - trainStart
    minutes = round(trainTotal // 60)
    seconds = round(trainTotal % 60)
    print(f'Time training: {minutes}min {seconds}sec')
```

Obrázok 17: implementácia stratovej a optimalizačnej funkcie a trénovanie modelu[12]

- *CrossEntropyLoss()* – je používaná na meranie ako dobre klasifikačný model funguje v strojovom učení. Loss (strata) sa meria ako číslo medzi 1 a 0, pričom 0 znamená dokonalý model. Cieľom je vo všeobecnosti dostať model čo najbližšie k hodnote 0. Porovnáva rozdiel medzi zisteným rozdelením pravdepodobnosti klasifikačného modelu a predpovedaným rozdelením [12].
  - *optim.SGD(params, learning rate)* – optimalizačná funkcia, ktorá je súčasťou PyTorch frameworku. Slúži na zmenu atribútov, ako sú napr. váhy a rýchlosť učenia, s cieľom znížiť stratovú hodnotu. SGD (Stochastic Gradient Descent) je typ gradientového spádu, ktorý sa snaží o častejšiu aktualizáciu parametrov modelu. Takže namiesto aktualizácie za 1 epochu sa parametre aktualizujú každú iteráciu [12].
- *Epoch* – počet epoch hovorí o tom, koľkokrát algoritmus učenia použije celú sadu tréningových vzoriek. Je dôležité zvoliť vhodný počet epoch, malý počet spôsobí nedostačujúco presné výsledky, naopak veľký počet zapríčiní stagnáciu zlepšenia a zbytočne zvýši finančné nároky modelu [12].
- *model.train()* – slúži na informovanie modelu, že sa začne trénovanie [12]

Funkcia *trainModel(model)* ma vstupný parameter model, čo je nami zadefinovaná architektúra zo MNIST dátového balíka. Nastaví sa stratová a optimalizačná funkcia a informujeme model, že začíname trénovanie. Zavoláme funkciu *time()* z knižnice time na zaznamenanie dĺžky trénovania. Funkcia prejde cez 5 iterácií epochov, počas každej epochy funkcia iteruje cez trénovaciu sadu, ktorá obsahuje obrázky číslic a ich zaradenie. Obrázky sme museli normalizovať pomocou *images.view(-1, 28*28)* do 1D tenzora, pretože model očakáva pre každý obrázok sploštený vstup. Keďže iterujeme cez 60 000 trénovacích dát, každou iteráciou musíme gradient vymazať pomocou *optimizer.zero_grad()*. Model vykonáva dopredné šírenie pomocou *model(images)*, tým sa vypočítajú predpovedané výstupy pre každý obrázok. Potom je vypočítaná strata a uložená do premennej loss. Gradient parametrov vzhľadom na stratu sa vypočíta pomocou *loss.backward()* a optimalizátor aktualizuje parametre modelu na základne vypočítaných gradientov pomocou *optimizer.step()*. Na konci epochy nám funkcia vypíše stratu za aktuálnu epochu. Po konci cyklu sa znova zavolá funkcia *time()* a vypočíta sa celkový čas trénovania.

```
Epoch 1: training loss: 0.42746683955192566
Epoch 2: training loss: 0.6672544479370117
Epoch 3: training loss: 0.708014726638794
Epoch 4: training loss: 0.26769500970840454
Epoch 5: training loss: 0.14225395023822784
Time training: 0min 46sec
```

Obrázok 18: strata za jednotlivé epochy

3.12.2Výpočet presnosti modelu

Funkcia *evaluateModel(model)* zoberie vstupnú hodnotu model a porovná jej predpovedané výsledky s testovacími dátami. Na začiatku sa zavolá funkcia *model.eval()*, aby model vedel, že začne evaluácia. Funkcia *torch.no_grad()* sa zavolá na zlepšenie efektivity, keďže počas evaluácie nepotrebujeme počítať gradient. Zavolá sa *model(inputs)*, aby sa porovnali všetky predikcie modelu s testovacími dátami. Ak model správne predpovedal číslicu, pripočíta sa do parametra správnych obrázok a na konci sa vypočíta celková presnosť predpovede.

```
def evaluateModel(model):
    correct = 0
    total = 0
    model.eval()
    with torch.no_grad():
        for data in testloader:
            inputs, labels = data
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Accuracy of the network on the test images: {100 * correct / total} %')
```

```
Accuracy of the network on the test images: 92.5199966430664 %
```

Obrázok 19:implementácia funkcie na zistenie presnosti modelu a presnosť modelu

3.12.3Výpočet počtu parametrov

```
def get_total_parameters_count(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)


def get_pruned_parameters_count(pruned_model, total_params_count):
    params = 0
    for param in pruned_model.parameters():
        if param is not None:
            params += torch.nonzero(param).size(0)
    print('Pruned Model parameter count:', params)
    print(f'Compressed Percentage: {(100 - (params / total_params_count) * 100)}%')
```

Obrázok 20: funkcie na výpočet parametrov

- *get_total_parameters_count(model)* – funkcia zoberie vstupný model a vypočíta celkový počet parametrov

```
Original Model paramete count: 98700
```

Obrázok 21: počet parametrov originálneho modelu

- *get_pruned_parameters_count(model)* – funkcia zoberie vstupný model po pruningu a vypočíta celkový počet parametrov po pruningu. Model to pruningu musíme kontrolovať týmto spôsobom, pretože knižnica *torch.nn.utils.prune* po pruningu len zmení hodnotu váh na 0, ale nezmaže ich, ani po zavolaní funkcie *prune.remove().* Kebyže budeme porovnávať veľkosť modelov pred a po pruningu pomocou funkcie *size(),* zistíme, že majú rovnakú

veľkosť, keďže aj váhy s hodnotou 0 zaberajú bajty v pamäti. Preto veľkosť po pruningu zisťujeme tak, že spočítame všetky nenulové váhy.

3.12.4Pruningové algoritmy

V kóde som implementoval 4 druhy pruningu, každý ma ako vstup originálny model, intenzitu pruning vyjadrenú v percentách a celkový počet originálnych parametrov na porovnanie s výsledným počtom po pruningu:

```
def L1Structured(modelCNN, percentage, total_params_count):
    model = copy.deepcopy(modelCNN)
    print(f'\nPruning L1 Structured {percentage * 100}%')
    prune.ln_structured(model.fc1, 'weight', amount=percentage, n=1, dim=0)
    prune.ln_structured(model.fc2, 'weight', amount=percentage, n=1, dim=0)
    prune.ln_structured(model.fc3, 'weight', amount=percentage, n=1, dim=0)

    trainModel(model)

    prune.remove(model.fc1, 'weight')
    prune.remove(model.fc2, 'weight')
    prune.remove(model.fc3, 'weight')

    evaluateModel(model)

    get_pruned_parameters_count(model, total_params_count)
```

Obrázok 22: štruktúrovaný pruning podľa L1 normy

Na každú vrstvu sa aplikuje pruning podľa L1 normy. L1 norma je súčet absolútnej hodnoty vektora. Potom sa trénuje model po aplikácii pruningu a funkcia prune.remove() zabezpečí trvalú aplikáciu pruningu. Nakoniec sa vyhodnotí presnosť a porovná sa počet parametrov.

```
def L2Structured(modelCNN, percentage, total_params_count):
    model = copy.deepcopy(modelCNN)
    print(f'\nPruning L2 Structured {percentage * 100}%')
    prune.ln_structured(model.fc1, 'weight', amount=percentage, n=2, dim=0)
    prune.ln_structured(model.fc2, 'weight', amount=percentage, n=2, dim=0)
    prune.ln_structured(model.fc3, 'weight', amount=percentage, n=2, dim=0)

    trainModel(model)

    prune.remove(model.fc1, 'weight')
    prune.remove(model.fc2, 'weight')
    prune.remove(model.fc3, 'weight')

    evaluateModel(model)

    get_pruned_parameters_count(model, total_params_count)
```

Obrázok 23: štruktúrovaný pruning podľa L2 normy

L2 norma sa vypočíta ako odmocnina zo súčtu mocniny hodnôt vektora.

```
def globalL1Unstructured(modelCNN, percentage, total_params_count):
    model = copy.deepcopy(modelCNN)
    print(f'\nPruning L1 Unstructured {percentage * 100}%')
    parameters_to_prune = (
    (model.fc1, 'weight'),
    (model.fc2, 'weight'),
    (model.fc3, 'weight')
    )

    prune.global_unstructured(
        parameters_to_prune,
        pruning_method=prune.L1Unstructured,
        amount=percentage,
    )

    trainModel(model)

    prune.remove(model.fc1, 'weight')
    prune.remove(model.fc2, 'weight')
    prune.remove(model.fc3, 'weight')

    evaluateModel(model)

    get_pruned_parameters_count(model, total_params_count)
```

Obrázok 24: globálny pruning podľa normy L1

Globálny pruning je častokrát účinnejší ako lokálny, v tomto prípade štruktúrovaný, keďže PyTorch nemá implementovaný globálny štruktúrovaný pruning. Globálny pruning aplikuje pruning na všetky vrstvy, čím vieme dosiahnuť menší počet konečných parametrov.

```
def globalRandomUnstructured(modelCNN, percentage, total_params_count):
    model = copy.deepcopy(modelCNN)
    print(f'\nPruning Random Unstructured {percentage * 100}%')
    parameters_to_prune = (
    (model.fc1, 'weight'),
    (model.fc2, 'weight'),
    (model.fc3, 'weight')
    )

    prune.global_unstructured(
        parameters_to_prune,
        pruning_method=prune.RandomUnstructured,
        amount=percentage,
    )

    trainModel(model)

    prune.remove(model.fc1, 'weight')
    prune.remove(model.fc2, 'weight')
    prune.remove(model.fc3, 'weight')

    evaluateModel(model)

    get_pruned_parameters_count(model, total_params_count)
```

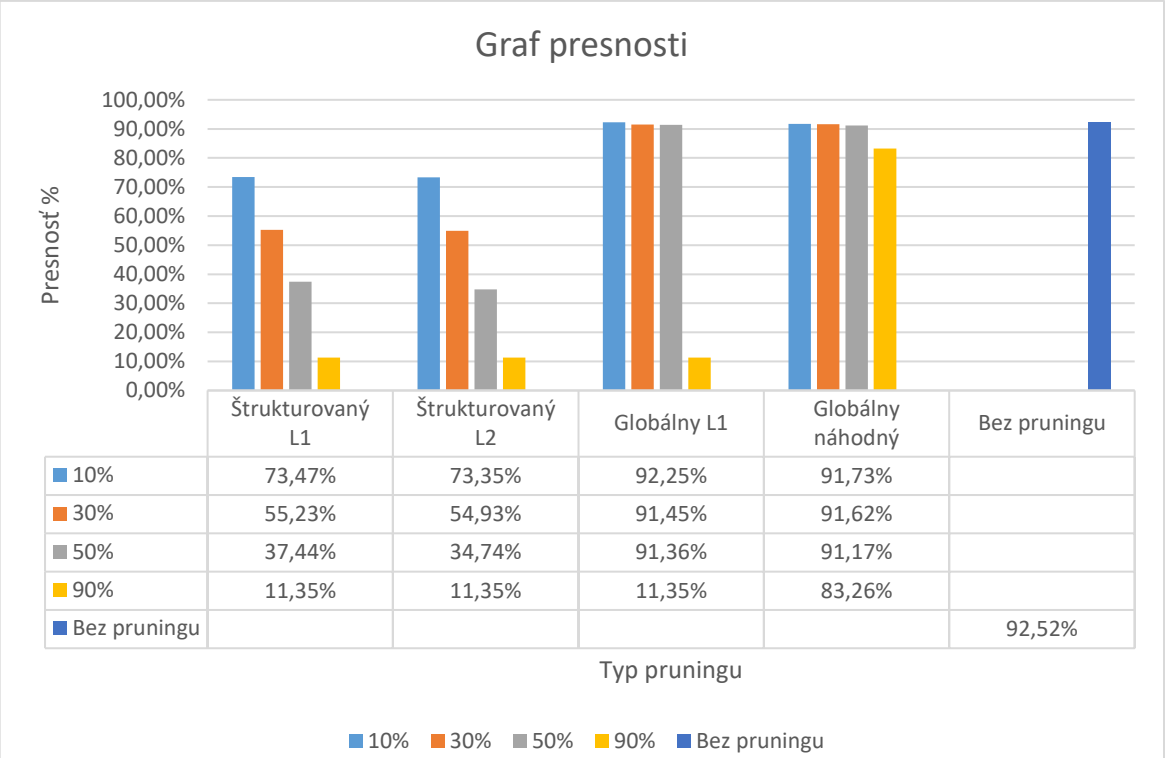
Obrázok 25: globálny náhodný pruning

Pruning založený na náhodnom výbere váh, ktoré budú vynulované. Má potenciál byť najlepším, ale zároveň aj najhorším pruningový algoritmus, keďže zo siete môže odstrániť len váhy, ktoré vôbec alebo veľmi málo prispievali k výslednej presnosti siete alebo naopak môže odstrániť dôležité váhy, ktoré najviac prispievali k výslednej presnosti siete.

3.12.5Porovnanie pruningových algoritmov

Z časového hľadiska všetky pruningové algoritmy majú približne rovnako dlhú trénovaciu dobu, keďže ako bolo spomenuté v kapitole 3.12.3, parametre sa len nulujú, ale neodstraňujú sa. Preto počas trénovania sa stále prechádza cez neuróny s nulovou váhou a časový rozdiel je zanedbateľný.

Ale to isté neplatí o počte parametrov po pruningu a konečnej presnosti voči testovacím dátam. Každý pruningový algoritmus sme aplikovali so intenzitou 10%, 30%, 50% a 90% a konečné výsledky sú nasledovné:



	Štrukturovaný L1	Štrukturovaný L2	Globálny L1	Globálny náhodný	Bez pruningu
10%	73,47%	73,35%	92,25%	91,73%	
30%	55,23%	54,93%	91,45%	91,62%	
50%	37,44%	34,74%	91,36%	91,17%	
90%	11,35%	11,35%	11,35%	83,26%	
Bez pruningu					92,52%

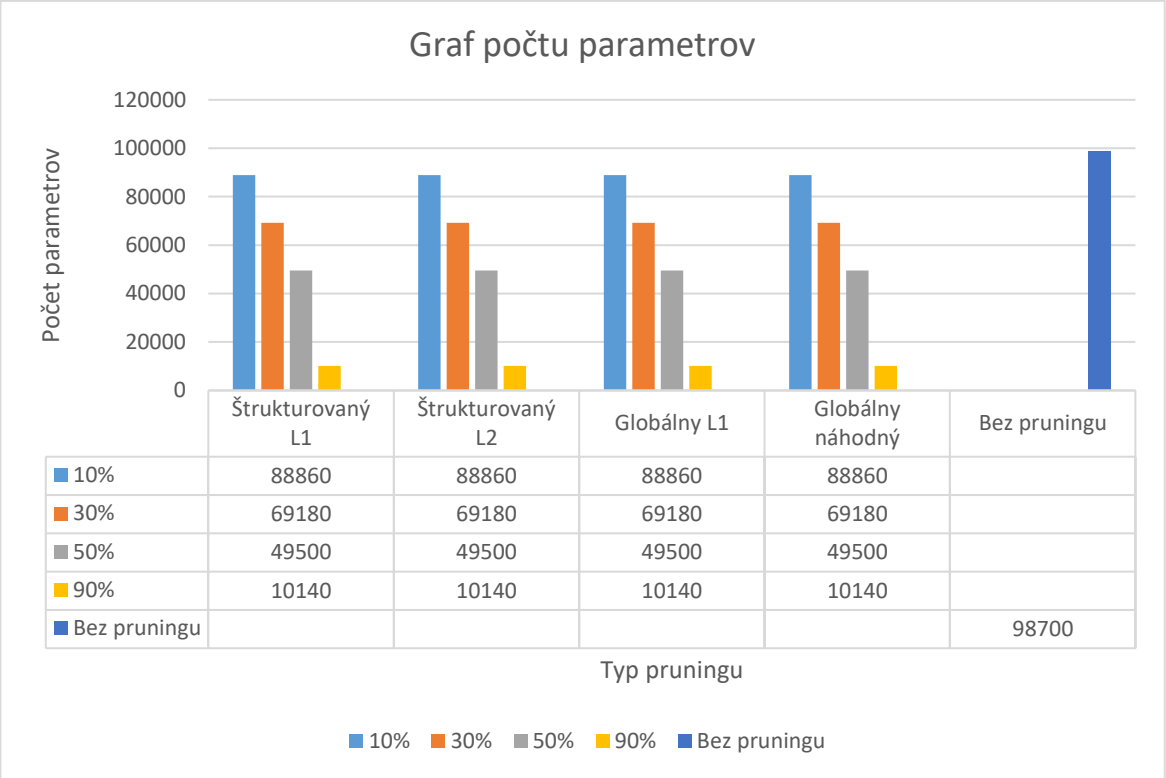
Graf 1: graf presnosti na doprednej sieti

Na grafe výslednej presnosti voči trénovacím dátam môžeme vidieť, že globálny náhodný pruning si zachoval vynikajúcu presnosti pri vynulovaní 90% váh voči ostatným pruningovým metódam. Problém s náhodným pruningom je ten, že nie je konzistentný.

```
Pruning Random Unstructured 90.0%
Epoch 1: training loss: 3.5949525833129883
Epoch 2: training loss: 1.8374311923980713
Epoch 3: training loss: 1.0700910091400146
Epoch 4: training loss: 0.775068461894989
Epoch 5: training loss: 0.8052637577056885
Time training: 0min 46sec
Accuracy of the network on the test images: 75.33000183105469 %
Pruned Model parameter count: 10140
Compressed Percentage: 89.72644376899696%
```

Obrázok 26: ďalšie spustenie náhodného pruningu

Na obrázku môžeme vidieť, že presnosť spadla až na 75%, pretože náhodný pruning vynuloval váhy, ktoré mali v sebe dôležité informácie o vzoroch. V jednoduchej neurónovej sieti, ako je táto s dopredným šírením, sa odstránenie dôležitej váhy nemusí až tak výrazne prejaviť, ale pri konvolučné budú väčšie rozdiely medzi jednotlivými behmi, ale to hlbšie opíšem v nasledujúcej kapitole.



Graf 2: graf počtu parametrov doprednej siete

Podľa Graf 2. vieme povedať, že intenzita pruningu pôsobila rovnako s každým algoritmom. Zakaždým bol konečný počet parametrov po pruningu znížený o rovnakú hodnotu v závislosti od intenzity pruningu.

3.13 Konvolučná neurónová sieť

Konvolučné neurónové siete sú špeciálne navrhnuté na spracovanie a analýzu vizuálnych vstupov, sieť je zložitejšia konvolučná neurónovú sieť, ktorá obsauje 2 konvolučné vrstvy, ReLU aktivačnú funkciu, 2 pooling vrstvy a 1 lineárnu výstupnú vrstvu

```
import torch.nn as nn

# Define the CNN model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=2)
        self.relu1 = nn.ReLU()
        self.avgpool1 = nn.AvgPool2d(kernel_size=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=2)
        self.relu2 = nn.ReLU()
        self.avgpool2 = nn.AvgPool2d(kernel_size=2)
        self.fc1 = nn.Linear(32 * 7 * 7, 10)

    def forward(self, x):
        out = self.conv1(x)
        out = self.relu1(out)
        out = self.avgpool1(out)
        out = self.conv2(out)
        out = self.relu2(out)
        out = self.avgpool2(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)

        return out

modelCNN = CNN()
```

Obrázok 27: zadefinovanie konvolučnej neurónovej sieti [12]

3.13.1Stratová a optimizačná funkcia a trénovanie modelu

```
def trainModel(model):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01)
    model.train()
    trainStart = time.time()
    for epoch in range(5):
        for images, labels in trainloader:
            log_ps = model(images)
            loss = criterion(log_ps, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        print(f'Epoch {epoch+1}: training loss: {loss.item()}')

    trainEnd = time.time()
    trainTotal = trainEnd - trainStart
    minutes = round(trainTotal // 60)
    seconds = round(trainTotal % 60)
    print(f'Time training: {minutes}min {seconds}sec')
```

Obrázok 30: implementácia stratovej a optimalizačnej funkcie a trénovanie modelu

Stratová a optimalizačná funkcia je rovnaká ako u siete s dopredným šírením, jediný rozdiel pri konvolučnej sieti je, že nemusíme normalizovať obrázky. Taktiež funkcia na výpočet presnosti voči testovacím dátam, celkového počtu parametrov a parametrov po pruningu je rovnaká.

3.13.2Pruningové algoritmy

Aj v tomto prípade budeme testovať predošlé 4 pruningové algoritmy s rôznymi intenzitami pruningu. Keďže konvolučná neurónová sieť obsahuje okrem poslednej lineárnej vrstvy aj konvolučné vrstvy, budeme musieť upraviť pruningové funkcie.

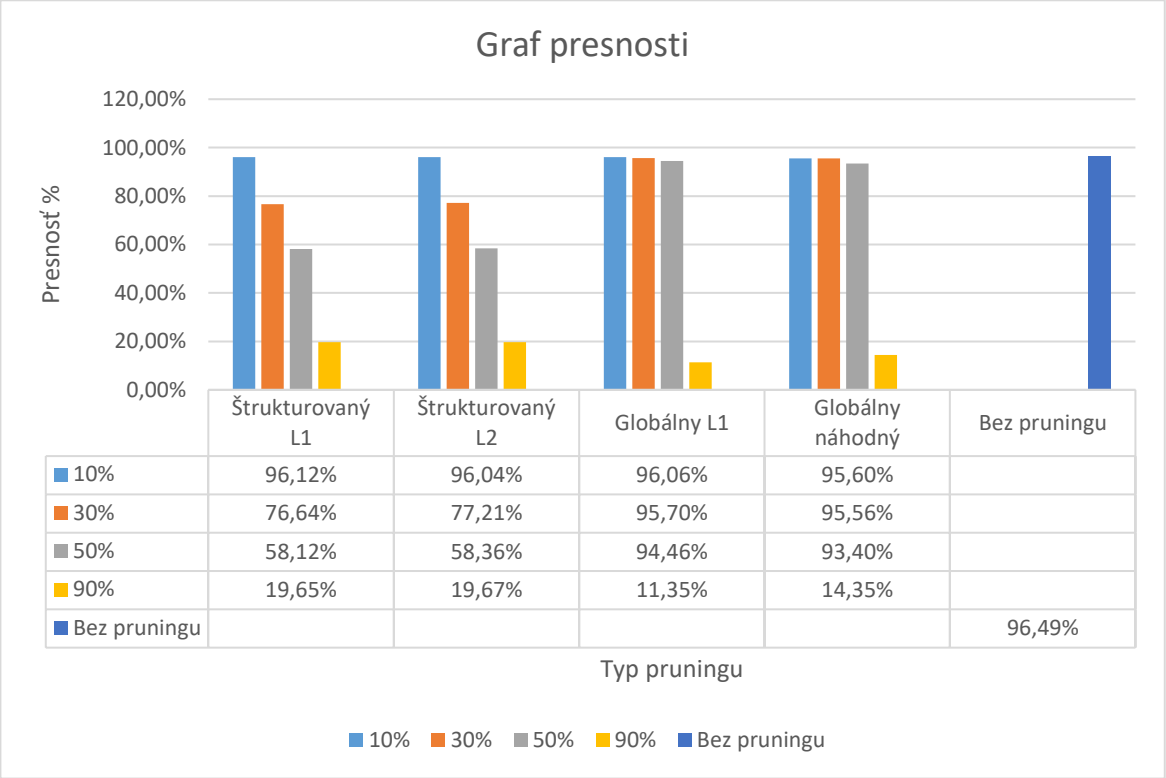
```
prune.ln_structured(model.conv1, 'weight', amount=percentage, n=1, dim=0)
prune.ln_structured(model.conv2, 'weight', amount=percentage, n=1, dim=0)
prune.ln_structured(model.fc1, 'weight', amount=percentage, n=1, dim=0)

trainModel(model)

prune.remove(model.conv1, 'weight')
prune.remove(model.conv2, 'weight')
prune.remove(model.fc1, 'weight')
```

Obrázok 31: implementáciu pruningu konvolučných vrstiev

Do každej pruningovej funkcie sme pridali pruning pre konvolučnú vrstvu a po trénovaví sme spravili pruning trvalým pre každú vrstvu.

3.13.3Porovnanie pruningových algoritmov

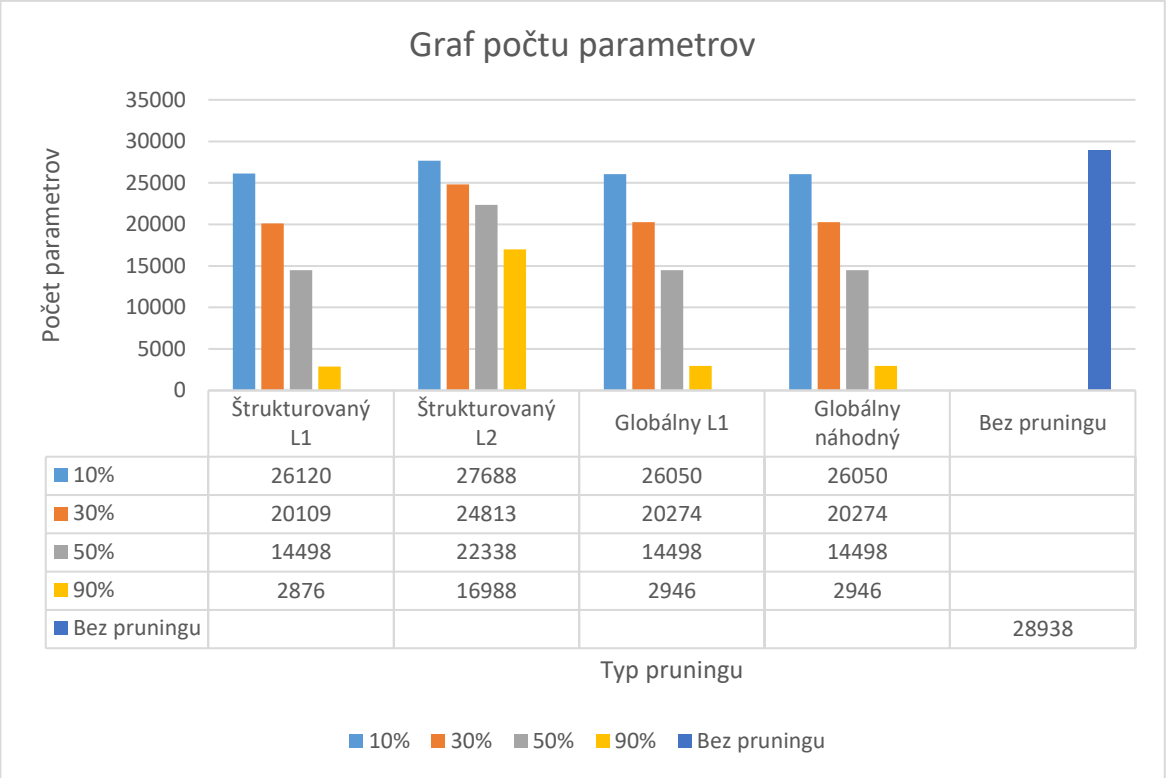
Aj v tomto prípade z časového hľadiska nie je nijaký veľký rozdiel medzi trénovaním s alebo bez použitia pruningu. Napriek tomu, že hodnota váh neurónov je 0, stále sa sčítajú hodnoty vynásobené filtrami, preto ani v tomto prípade sa nedá zredukovať čas trénovania neurónovej siete.



	Štrukturovaný L1	Štrukturovaný L2	Globálny L1	Globálny náhodný	Bez pruningu
10%	96,12%	96,04%	96,06%	95,60%	
30%	76,64%	77,21%	95,70%	95,56%	
50%	58,12%	58,36%	94,46%	93,40%	
90%	19,65%	19,67%	11,35%	14,35%	
Bez pruningu					96,49%

Graf 3: graf presnosti konvolučnej siete

Na základe grafu presnosti vieme povedať, že globálny L1 a globálny náhodný majú najlepšie výsledky v pomere intenzity pruningu a zachovanej presnosti. Tentokrát globálny náhodný pruning s intenzitou 90% vynuloval dôležité váhy, ktoré mali veľký dopad na zachovanie vzorov modelu.



Graf 4: graf počtu parametrov konvolučnej siete

Na grafe počte parametrov po pruningu pozorujeme, že štruktúrovaný L2 pruning nebol až taký účinný v redukovaní parametrov po pruningu ako ostatné pruningové metódy. Aj tentokrát pruningové algoritmy, okrem L2, znížili počet parametrov na základe vstupnej intenzity.

Záver

Cieľom práca bolo zistiť efektivity rôznych pruningových algoritmov a vyhodnotiť ich vplyv na umelé neurónové siete. Bolo treba vyhodnotiť časovú úsporu, rozdiel veľkosti alebo počtu parametrov a zachovanie presnosti voči testovacej sade dát. Na riešenie úlohy bolo potrebné naprogramovať vlastný program v Pythone pomocou frameworku PyTorch, kde bola zadefinovaná architektúra siete. Následne sme zvolili vhodný súbor dát, implementovali sme funkciu na trénovanie a vyhodnotili sme pruning neurónovej siete.

Zvolili sme si dva typy neurónových sietí, a to neurónovú sieť s dopredným šírením a konvolučnú neurónovú sieť. Cieľom výberu bolo ukázať rozdiel pruningu na jednoduchej doprednej architektúre a zložitejšej konvolučnej neurónovej sieti.

Následne sme si zvolili štyri pruningové algoritmy – štruktúrovaný s L1 normou, štruktúrovaný s L2 normou, neštruktúrovaný globálny s L1 normou a neštruktúrovaný globálny s náhodným pruningom. Následne sme jednotlivé pruningové algoritmy aplikovali na jednotlivé architektúry.

Pre neurónovú sieť s dopredným šírením bol najefektívnejší globálny náhodný pruning, ktorý znížil počet nenulových váh o 90% a zároveň si zachoval presnosť 83.26%. Aj keď dosahoval nejlepšie výsledky u všetkých štyroch intenzitách pruningu, náhodný pruning nie je spoľahlivá pruningová metóda, keďže po každom spustení presnosť nebola konzistentná. Výsledok testov ukázal, že neštruktúrovaný globálny pruning s L1 normou mal 2. najlepšie a zároveň najkonzistentnejšie výsledky. Preto za najvhodnejšie pre túto sieť označujem neštruktúrovaný globálny pruning s L1 normou.

Konvolučná neurónová sieť mala tiež podobné výsledky, ako sieť s dopredným šírením, ale u tejto sieti sa ukázali väčšie rozptyly presnosti u neštruktúrovanom globálnom náhodnom pruningu. Preto aj v tomto prípade vieme označiť neštruktúrovaný globálny pruning s L1 normou za najlepšiu pruningovú metódu pre tento typ siete.

Zoznam použitej literatúry

1. KVASNIČKA, V. -- BEŇUŠKOVÁ, Ľ. -- POSPÍCHAL, J. -- FARKAŠ, I. -- TIŇO, P. -- KRÁĽ, A. Úvod do teórie neurónových sietí. Bratislava : IRIS-Knižní klub, 1997. 285 strany. ISBN 80-88778-30-1.
2. KVASNIČKA, V. -- POSPÍCHAL, J. Warren McCulloch and Walter Pitts - foundations of logical calculus, neural networks and automata. In *Artificial intelligence and cognitive science IV.* 1st vyd. Bratislava: Nakladateľstvo STU, 2014, s. 93--123. ISBN 978-80-227-4208-5.
3. Ferreira, Tiago & Mattheakis, Marios & Protopapas, Pavlos. (2021). A New Artificial Neuron Proposal with Trainable Simultaneous Local and Global Activation Function. Dostupné na internete: <https://www.researchgate.net/publication/348563211_A_New_Artificial_Neuron_Proposal_with_Trainable_Simultaneous_Local_and_Global_Activation_Function>
4. Nwankpa, Chigozie & Ijomah, W. & Gachagan, Anthony & Marshall, Stephen. (2020). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. Dostupné na internete: <https://www.researchgate.net/publication/328826136_Activation_Functions_Comparison_of_trends_in_Practice_and_Research_for_Deep_Learning>
5. CIGÁNEK, J. -- OSUSKÝ, J. Structure optimization of artificial neural networks using pruning methods. In CIGÁNEK, J. -- KOZÁKOVÁ, A. *2018 Cybernetics & Informatics (K&I).* Bratislava: Slovak Chemical Library, 2018, ISBN 978-1-5386-4420-1.
6. CIGÁNEK, J. -- KOCÚR, M. Neuro-Fuzzy modeling of dynamic systems in energetics using pruning methods. In *ACCS/PEIT 2017.* Piscataway: IEEE, 2017, s. 239--244.
7. Song Han and Jeff Pool and John Tran and William J. Dally. (2015). Learning both Weights and Connections for Efficient Neural Networks. Dostupné na internete: <https://arxiv.org/abs/1506.02626>
8. Pauly, Leo & Peel, Harriet & Luo, Shan & Hogg, David & Fuentes, Raul. (2017). Deeper Networks for Pavement Crack Detection. 10.22260/ISARC2017/0066. Dostupné na internete:

<https://www.researchgate.net/publication/319235847_Deeper_Networks_for_Pavement_Crack_Detection>

9. Yamashita, R., Nishio, M., Do, R.K.G. *et al.* Convolutional neural networks: an overview and application in radiology. *Insights Imaging* **9**, 611–629 (2018). Dostupné na internete: <https://doi.org/10.1007/s13244-018-0639-9>
10. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11), 2278-2324, 1998. Dostupné na internete: <http://yann.lecun.com/exdb/mnist/>
11. Huan Wang and Qiming Zhang and Yuehai Wang and Yu Lu and Haoji Hu. (2019). Structured Pruning for Efficient ConvNets via Incremental Regularization. Dostupné na internete: <https://arxiv.org/abs/1804.09461>
12. PyTorch: An open-source deep learning framework. Dostupné na internete: <https://pytorch.org/>
13. Vadera, Sunil & Ameen, Salem. (2022). Methods for Pruning Deep Neural Networks. IEEE Access. 10. 1-1. 10.1109/ACCESS.2022.3182659. Dostupné na internete: https://www.researchgate.net/publication/361288529_Methods_for_Pruning_Deep_Neural_Networks
14. Hammoudeh, Ahmad. (2018). A Concise Introduction to Reinforcement Learning. 10.13140/RG.2.2.31027.53285. Dostupné na internete: https://www.researchgate.net/publication/323178749_A_Concise_Introduction_to_Reinforcement_Learning
15. MathWorks: Reinforcement Learning. Dostupné na internete: < https://www.mathworks.com/discovery/reinforcement-learning.html>
16. Amidi, S., Amidi, A. „Deep learning“ 2019, Stanford. Dostupné na internete: <https://stanford.edu/~shervine/teaching/cs-230/>