VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE HLUBOKÝCH NEURONOVÝCH SÍTÍ

DEEP NEURAL NETWORK OPTIMIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN BAŽÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Bažík Martin**
Obor: Informační technologie
Téma: **Optimalizace hlubokých neuronových sítí**
Deep Neural Network Optimization
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s problematikou neuronových sítí a hlubokých neuronových sítí.
2. Vyberte si vhodnou knihovnu pro práci s hlubokými neuronovými sítěmi a seznamte se s jejím použitím.
3. Na zvolených testovacích úlohách ověřte funkčnost knihovny. Porovnejte dosažené výsledky s výsledky v literatuře.
4. Navrhněte optimalizace studované neuronové sítě s ohledem na vylepšení některého z jejích parametrů, např. doby zpracování apod.
5. Zjistěte chybu a další parametry, které dosahuje optimalizovaná neuronová síť pro zvolené testovací úlohy.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sekanina Lukáš, prof. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Cieľom tejto bakalárskej práce bolo navrhnúť, implementovať a analyzovať rôzne optimalizácie vybraných hlbokých neurónových sietí. Ich účelom bolo zlepšenie sledovaných parametrov neurónovej siete. Implementované optimalizácie spočívajú v práci s reprezentáciou dát využívaných operáciami neurónovej siete a hľadaní najlepšej kombinácie jej hyperparametrov. Jednotlivé optimalizácie boli prevedené na konvolučných neurónových sieťach založených na architektúre LeNet-5 pri využití dátových sád MNIST, CIFAR-10 a SVHN. Implementácia a následná optimalizácia neurónových sietí boli prevedené využitím knižnice Tiny-dnn v programovacom jazyku C++.

Abstract

The goal of this thesis was to design, implement and analyze various optimizations of deep neural networks, in order to improve the observed parameters. The optimizations are based on modification of the data representation used by neural network operations and searching for the best combination of its hyper-parameters. The convolutional neural networks used for these optimizations were built on LeNet-5 architecture and trained on MNIST, CIFAR-10, and SVHN datasets. The neural networks and their optimizations were implemented within Tiny-dnn library using C++ programming language.

Kľúčové slová

hlboké neurónové siete, optimalizácia, aproximácia, strojové učenie, umelá inteligencia, konvolučné neurónové siete

Keywords

deep neural networks, optimization, approximation, machine learning, artificial intelligence, convolutional neural networks

Citácia

BAŽÍK, Martin. *Optimalizace hlubokých neuronových sítí.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Lukáš Sekanina, Ph.D.

Optimalizace hlubokých neuronových sítí

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením prof. Ing. Lukáša Sekaninu, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Bažík
14. mája 2018

Poďakovanie

Ďakujem prof. Ing. Lukášovi Sekaninovi, Ph.D. za pomoc a odborný dohľad pri riešení bakalárskej práce.

Obsah

Kapitola 1

Úvod

Hlboké neurónové siete patria v súčasnosti medzi najpopulárnejšie riešenia klasifikačných problémov, medzi ktoré patrí napríklad rozpoznávanie obrazu a zvuku. Ich princíp je založený na štruktúre neurónov rozvrhnutých do niekoľkých navzájom prepojených vrstiev, ktoré modelujú fungovanie nervového systému (mozgu). Nejde však o novú metódu, jej pôvod totiž siaha do päťdesiatych rokov 20. storočia, kedy bol model ADALINE [31] využitý na filtrovanie šumu na telefónnej linke pomocou binárnej klasifikácie. Časová a pamäťová zložitosť využívaných algoritmov však dlhodobo limitovala ich využitie na riešenie zložitejších problémov.

Neustále rastúci výkon osobných počítačov a výpočtových klastrov čiastočne odstraňuje tento problém. Implementácia hlbokých neurónových sietí na prenosných a vstavaných zariadeniach však napreduje len pomaly. Tieto zariadenia majú často limitované zdroje, medzi ktoré patria nedostatočný procesorový výkon, malá operačná pamäť, alebo nízky úložný priestor. Často je tiež kladený vysoký dôraz na minimalizovanie príkonu, menovite na zariadeniach internetu vecí (skrátene IoT). Spomínané problémy je možné aspoň čiastočne vyriešiť nasadením vhodných optimalizačných metód.

Cieľom tejto práce je navrhnúť optimalizácie hlbokej neurónovej siete, ktoré by znížili určité faktory jej výpočtovej zložitosti a zvýšili jej presnosť. Tieto metódy však môžu ovplyvniť ostatné parametre hlbokých neurónových sietí, ako napríklad chybu klasifikácie a dobu učenia. Z toho dôvodu je potrebné sledovať vplyv optimalizačných metód na tieto parametre.

Kapitoly 2 a 3 predstavia základné princípy, ktoré stoja za vznikom hlbokých neurónových sietí, ako aj ich základnú štruktúru. Prevedú čitateľa od fundamentálnych základov biologického neurónu až po implementáciu konvolučných neurónových sietí. Kapitola 4 uvedie existujúce knižnice využívané pre implementáciu hlbokých neurónových sietí. Následne sa v kapitole 5 overí správne fungovanie knižnice, ktorá bola zvolená pre túto prácu. V kapitole 6 budú prevedené jednotlivé optimalizácie a zhodnotené ich výsledky. Záver predstavuje kapitola 7.

Kapitola 2

Úvod do neurónových sietí

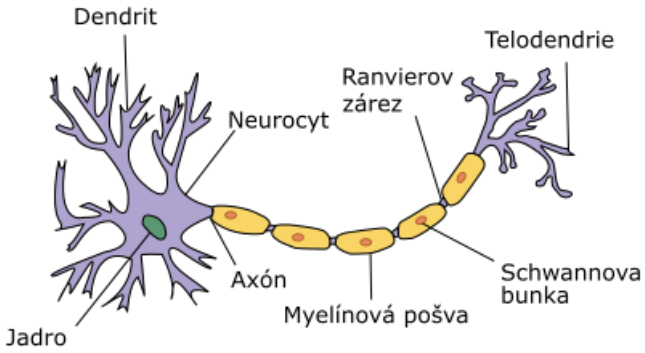
V oblasti informatiky a počítačových vied sa využívajú mnohé algoritmy inšpirované prírodou. Patria medzi ne aj neurónové siete, ktoré uvedie táto kapitola. Sekcia 2.1 predstaví biologickú motiváciu, ktorá stála za vznikom prvých neurónových sietí. Nasledujúca sekcia 2.2 sa zameriava na základnú štruktúru a metódy umelého neurónu, ako základnej stavebnej jednotky neurónových sietí. Tie sa pre potreby neurónových sietí spájajú do vrstiev. Sekcia 2.3 načrtne princíp viacvrstvových neurónových sietí.

2.1 Biologická motivácia

Pre správne porozumenie fungovania umelého neurónu je najprv potrebné pochopiť, aké procesy prebiehajú v reálnom systéme, ktorý sa týmto spôsobom modeluje.

> „Neurónová sieť je prepojené zostavenie jednoduchých výpočtových prvkov, jednotiek alebo uzlov, ktorých funkcionalita je voľne založená na biologickom neuróne. Výpočtová schopnosť siete je uložená v sile (váhe) medzijednotkových spojení, získaných procesom adaptácie (učenia sa) na množine trénovacích vzorov.“ [8, str. 13]

Ľudský mozog obsahuje približne 100 miliárd neurónov, z ktorých každý sa skladá z troch základných častí. Sú nimi `dendrity`, `telo` a `axón`, viz obrázok 2.1. Informácie sú medzi neurónmi prenášané cez `synapsy` v podobe elektrického `impulzu`.



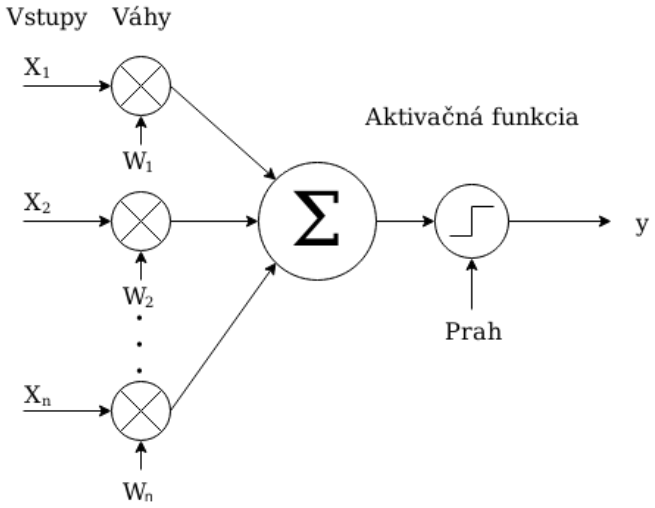
Obr. 2.1: Biologický neurón, prevzaté z [10].

Nervové bunky sú excitabilné, čo znamená, že po ich stimulácii dochádza ku chemickým zmenám generujúcim slabý elektrický impulz. Pri podnete, ako napríklad podráždenie

pokožky, sa na membráne, ktorá tvorí povrch neurónu, otvoria iónové kanáliky, čím sa zvýši jej elektrický potenciál. Potenciál je v počiatočnom stave na hodnote $-70\,\text{mV}$ a nazýva sa pokojový potenciál. Ak po stimulácii dosiahne prahovú hodnotu $-55\,\text{mV}$, otvoria sa aj ďalšie iónové kanáliky a hodnota potenciálu rastie, až kým nedosiahne hodnotu akčného potenciálu. Týmto sa spustí reťazová reakcia, ktorá pošle impulz ďalej cez axón. Membrána sa následne repolarizuje a hodnota potenciálu sa vráti do pôvodného stavu. Na konci axónov sa nachádzajú synapsy, ktoré posielajú tento impulz na dendrity ďalšieho neurónu. Existujú dva druhy synáps, elektrické a chemické. V chemických synapsách dochádza ku prevodu elektrického signálu na chemickú reakciu, čím je možné impulz modifikovať, a teda mu pridať aj určitú váhu. Informácie sa medzi neurónmi prenášajú pomocou neurotransmiterov, ktoré zapríčinia zmenu potenciálu na prijímajúcom neuróne. Týmto spôsobom môžu vyvolať nový impulz alebo aktívne potlačiť jeho vznik [22][8].

2.2 Umelý neurón

Umelý neurón je možné definovať ako matematickú funkciu, ktorá tvorí základnú stavebnú zložku neurónových sietí. Na vstup umelého neurónu, ktorý reprezentuje biologické dendrity, prichádzajú hodnoty $X_1, X_2, \ldots, X_n$, viz obrázok 2.2. Tieto hodnoty modelujú funkciu neurotransmiterov biologického neurónu. Prijaté hodnoty sú následne vynásobené prislúchajúcimi váhami spojení $W_1, W_2, \ldots, W_n$ modelujúc váhy v chemických synapsách biologického systému. Takto vynásobené hodnoty sa v ďalšom kroku sčítajú aritmetickým súčtom. Výsledok reprezentuje aktuálnu hodnotu elektrického potenciálu v biologickom neuróne a nazýva sa tiež vnútorný potenciál. V poslednom kroku je potrebné namodelovať generovanie akčného potenciálu. Na tento účel slúži aktivačná funkcia, ktorá prináša nelineárnosť do celého systému. Najjednoduchšou aktivačnou funkciou je Heavysideova funkcia, ktorej skok je posunutý na hodnotu prahu. Tento prah priamo reprezentuje prah akčného potenciálu potrebného pre aktiváciu neurónu. Výstup aktivačnej funkcie predstavuje výstup neurónu y, ktorý sa cez synapsy prenáša na ďalší neurón.



Obr. 2.2: Model umelého neurónu podľa Rosenblatta

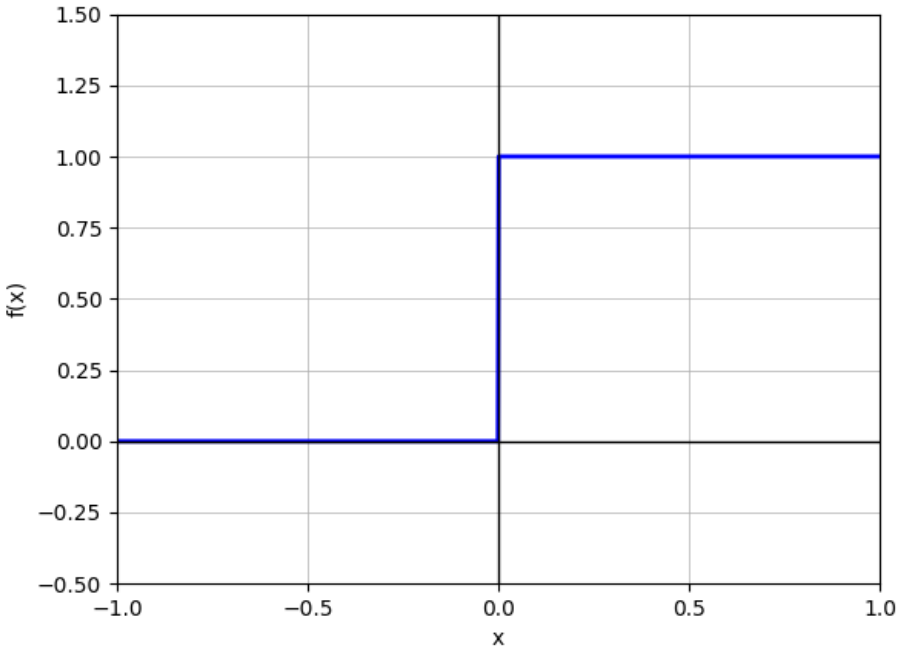
2.2.1 Perceptrón

Prvý model perceptrónu vytvoril Frank Rosenblatt v roku 1958. Tento model bol definovaný ako stroj, ktorý je schopný učiť sa klasifikovať vzory pomocou zmeny váh spojení vedúcich do prahového prvku [35, str. 18]. Ide o prvý model neurónovej siete tvorenej jediným umelým neurónom, ktorého aktivačnou funkciou je Heavisideova funkcia $f(x)$ s prahom θ,

$$f(x) = \begin{cases} 0 & x < \theta \\ 1 & \theta \leq x. \end{cases} \tag{2.1}$$

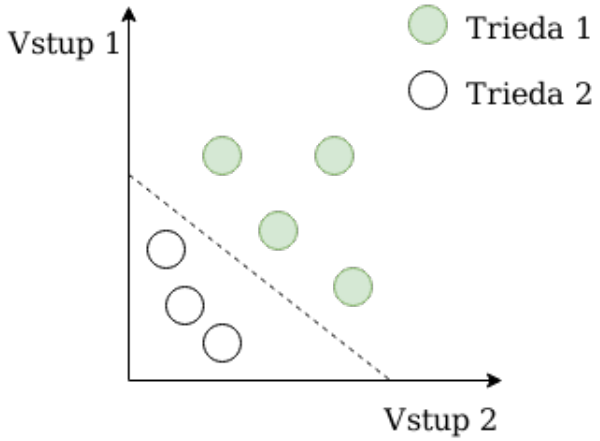
Z praktických dôvodov je však jednoduchšie pracovať s prahom v hodnote 0, viz vzorec 2.2 a obrázok 2.3. Prah θ je preto vhodné vyňať ako jedno zo vstupných spojení, ktorého hodnota je vždy 1 a váha $-\theta$. Toto spojenie sa tiež nazýva `bias`. Vstup aktivačnej funkcie x predstavuje vnútorný potenciál, teda vážený súčet vstupov perceptrónu $x = \sum_n W_n X_n$.

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \leq x \end{cases} \tag{2.2}$$



Obr. 2.3: Graf Heavisideovej funkcie na intervale $\langle -1, 1 \rangle$

Limitáciou perceptrónu je však to, že dokáže klasifikovať iba lineárne oddeliteľné vstupy. Tento fakt vyplýva z toho, že perceptrón dokáže vrátiť iba jednu z dvojice hodnôt 0 a 1, pričom každá hodnota môže byť naviazaná práve na jednu z tried. Tieto triedy rozdeľujú vstupy pomocou nadroviny, ktorej počet dimenzií je daný počtom vstupov perceptrónu. Graf 2.4 zobrazuje rozdelenie vzorov perceptrónu s dvomi vstupmi, teda 2D priestor rozdelený priamkou na dve triedy. Keďže preceptrón je neurónovou sieťou, prináša oproti samotnému neurónu novú funkciu, ktorou je aktualizovanie jednotlivých váh spojení pomocou procesu učenia sa.



Obr. 2.4: Lineárne oddeliteľné vzory

2.2.2 Učenie

`Učenie` patrí medzi základné metódy neurónových sietí, pretože umožňuje prispôsobiť algoritmus tak, aby vykonával požadovanú funkciu. Učenie v zmysle neurónových sietí je úprava váh jednotlivých spojení za účelom zvýšenia presnosti riešenia úlohy siete. Je ho možné rozdeliť na učenie s alebo bez učiteľa. `Učenie s učiteľom` pracuje s množinou vstupov a im zodpovedajúcich výstupov. Výstup siete pre daný vstup je porovnaný s očakávaným výstupom a na základe odchýlky je vypočítaná chyba, ktorá je následne využitá v algoritme učenia. Táto chyba je reprezentovaná funkciou, ktorá sa nazýva `chybová funkcia` (anglicky loss function). Na základe veľkosti chyby upraví učiaci algoritmus váhy siete v prospech jej zníženia. Najčastejšími úlohami, ktoré využívajú učenie s učiteľom, sú `klasifikácia` a `segmentácia`. Špeciálnym prípadom učenia s učiteľom je `učenie posilňovaním` (anglicky reinforcement learning), ktoré nepotrebuje dvojice vstupov a výstupov. Učenie prebieha za účelom dosiahnutia maxima globálnej odmeny.

Pri `učení bez učiteľa` nie sú dostupné korešpondujúce výstupy a učenie prebieha čisto na základe vnútornej štruktúry siete. Učenie bez učiteľa sa sústredí najmä na `zhlukovanie`, kde neurónová sieť klasifikuje vstup čo najpresnejšie do jedného zo zhlukov.

Zmena váhy spojenia v kroku t je definovaná vzťahom 2.3, kde $\vec{w}_i(t)$ predstavuje váhový vektor neurónu `i` a $\Delta\vec{w}_i(t)$ vektor inkrementu jeho váh. Hodnota inkrementu je definovaná špecifickým `pravidlom učenia` [33, str. 31].

$$\vec{w}_i(t+1) = \vec{w}_i(t) + \Delta\vec{w}_i(t) \tag{2.3}$$

Inkrement váh je ďalej možné vyjadriť ako vzorec 2.4, kde c reprezentuje `koeficient učenia` (anglicky learning rate), r je funkcia `pravidla učenia` a $\vec{x}(t)$ je vstupný vektor v kroku t. Vstupmi funkcie pravidla učenia sú vektor váh spojení $\vec{w}_i(t)$, aktuálny vstup $\vec{x}(t)$ a očakávaný výstup pre výpočet chyby $d_i(t)$.

$$\Delta\vec{w}_i(t) = cr[\vec{w}_i(t), \vec{x}(t), d_i(t)]\vec{x}(t) \tag{2.4}$$

Delta pravidlo

Jedným z pravidiel, ktoré je možné použiť pri učení, je `Delta pravidlo učenia`. Toto pravidlo vychádza z výpočtu štvorcovej chyby E medzi hľadaným výstupom d_i a skutočným výstupom o_i vo vzorci

$$E = \frac{1}{2}(d_i - o_i)^2. \tag{2.5}$$

Keďže výstup o_i je možné definovať ako hodnotu aktivačnej funkcie $f(\vec{w}_i(t)\vec{x}(t))$ neurónu i v kroku t, vzorec pre výpočet chyby je možné upraviť na tvar

$$E = \frac{1}{2}[d_i - f(\vec{w}_i(t)\vec{x}(t))]^2. \tag{2.6}$$

Vektor gradientu chyby ∇E je následne vypočítaný deriváciou chyby podľa váhového vektoru $\vec{w}_i(t)$ vo vzťahu

$$\nabla E = \frac{\partial E}{\partial \vec{w}_i(t)} = -(d_i - f(\vec{w}_i(t)\vec{x}(t)))\frac{\partial f(\vec{w}_i(t)\vec{x}(t))}{\partial \vec{w}_i(t)}\vec{x}(t). \tag{2.7}$$

Výstup aktivačnej funkcie $f(\vec{w}_i(t)\vec{x}(t))$ je možné spätne nahradiť hodnotou skutočného výstupu o_i. Vstup derivovanej aktivačnej funkcie f nahradíme za výstup neurónovej siete net_i vedúcej do tohto neurónu. Zovšeobecnený vzorec pre výpočet vektoru gradientu chyby má tvar

$$\nabla E = -(d_i - o_i)\frac{\partial f(net_i)}{\partial \vec{w}_i(t)}\vec{x}(t). \tag{2.8}$$

Po dosadení Delta pravidla, získaného vo vzorci 2.8, do vzorca pre výpočet inkrementu váh 2.4 dostaneme vzorec 2.9, ktorý predstavuje inkrement váh na základe Delta pravidla učenia.

$$\Delta\vec{w}_i(t) = -c\nabla E = c(d_i - o_i)\frac{\partial f(net_i)}{\partial \vec{w}_i(t)}\vec{x}(t) \tag{2.9}$$

Týmto spôsobom sa pri každom kroku aktualizujú hodnoty váh. Pre niektoré vstupné hodnoty to môže znamenať zvýšenie chyby, celková chyba sa však postupne blíži k nule. Tento postupný zostup gradientom chybovej funkcie sa tiež nazýva `zostup gradientu` (anglicky gradient descent). Pre nájdenie globálneho minima chybovej funkcie je nevyhnutné správne zvoliť krok, ktorým sa bude zostupovať po krivke gradientu. Pri príliš malých krokoch môže zostup gradientu skončiť v lokálnom minime, a tým zamedzí akékoľvek ďalšie znižovanie chyby. Naopak, ak je krok príliš veľký, nemusí hodnota gradientu konvergovať k nule. Správne zvolenie koeficientu učenia je preto nevyhnutné pre získanie požadovaných hodnôt váh spojení [35, str. 59–67].

2.3 Viacvrstvové neurónové siete

Existuje mnoho spôsobov ako klasifikovať neurónové siete, aby sme ich boli schopný pochopiť, študovať a analyzovať. Jedným takým rozdelením je delenie na základe počtu vrstiev. Základným modelom neurónovej siete je perceptrón, spomínaný v sekcii 2.2.1. Avšak pri väčšine reálnych problémov je potrebné klasifikovať do viac ako dvoch tried, a teda je nevyhnutné pridať ďalšie vrstvy. Neurónové siete s výrazne väčším počtom vrstiev sa nazývajú

hlbokými, neexistuje však ucelená definícia, koľko vrstiev predstavuje hlbokú neurónovú sieť. Vrstvy je možné rozdeliť na `vstupnú`, `skryté` a `výstupnú`.

Prvú vrstvu každej neurónovej siete tvorí vstupná vrstva. Úlohou tejto vrstvy je načítať vstupné dáta a poslať ich hodnoty pomocou vážených spojení na prvú skrytú vrstvu. Vstupná vrstva zvyčajne neobsahuje vstupné váhy ani aktivačnú funkciu.

Po vstupnej vrstve nasledujú skryté vrstvy, ktorých úlohou je implementovať riešenie úlohy neurónovej siete. Základné vrstvy, ktoré sú odvodené od perceptrónu, implementujeme pomocou aktivačných funkcií, ktoré berú na vstup vážený súčet výstupov aktivačných funkcií predchádzajúcich vrstiev. Je ich možné definovať rovnicou $y = f(\sum_{i \in I}(w_i f_j(...)))$, kde y reprezentuje výstup neurónu, f je aktivačná funkcia aktuálnej vrstvy, w_i je váha i-tého spojenia a f_j je aktivačná funkcia predchádzajúcej vrstvy. Okrem týchto klasických skrytých vrstiev existujú aj ďalšie druhy, ktoré budú spomenuté neskôr.

Poslednú vrstvu tvorí výstupná vrstva, ktorej úlohou je na základe hodnôt vypočítaných v skrytých vrstvách ponúknuť výstup. Táto vrstva, tak ako skryté vrstvy obsahuje váhy spojení a aktivačnú funkciu.

Ďalším možným delením je delenie podľa topológie. Dvomi základnými typmi sú `dopredné` (anglicky feedforward) a `rekurentné` (anglicky recurrent). Dopredné neurónové siete vychádzajú z modelu perceptrónu. Informácia v takomto type neurónovej siete putuje iba jedným smerom, a to sekvenčne usporiadanými vrstvami od vstupnej až po výstupnú vrstvu. Tento typ neurónových sietí sa využíva najmä na klasifikáciu obrazu.

Pomocou dopredných neurónových sietí nie je možné riešiť istú triedu problémov, kde je potrebné sledovať predchádzajúce výsledky, ako napríklad prenosový bit pri sčítaní [23, str. 50]. Kvôli tomu vznikli rekurentné neurónové siete, ktoré na rozdiel od dopredných sietí umožňujú spätnú propagáciu dát. Rekurentné neurónové siete sa teda od dopredných neurónových sietí líšia tým, že obsahujú aspoň jednu spätnú slučku [9, str. 23]. Niektoré čiastočné operácie sú tak znovu použiteľné priamo v štruktúre neurónovej siete. O ich implementáciu sa starajú pamäťové bunky, ktoré si na špecifikovaný čas uložia hodnotu signálu, ktorý nimi prechádza. Medzi najpoužívanejšie implementácie týchto pamäťových buniek patria v súčasnosti `dlhá krátkodobá pamäť` [11] (skrátene LSTM) a `uzavretá rekurentná jednotka` [5] (skrátene GRU). Medzi využitia rekurentných neurónových sietí patrí rozpoznávanie hlasu, ovládanie robotov, alebo rozpoznávanie rukopisu.

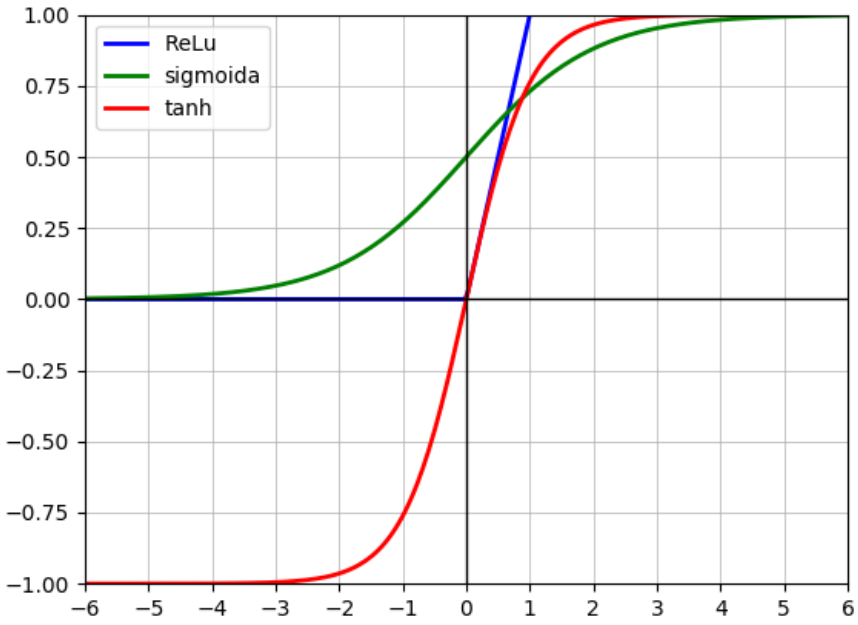
2.3.1 Aktivačné funkcie

Heavisideova funkcia by v mnohých prípadoch mohla byť ideálnou voľbou aktivačnej funkcie. Problémom však je, že deriváciou tejto funkcie je Diracova funkcia, ktorú nie je možné použiť v procese učenia. Z toho dôvodu ju je potrebné nahradiť inými aktivačnými funkciami. Medzi najpoužívanejšie patria `ReLu`, `tanh` a `sigmoida`, viz obrázok 2.5.

Funkcia Rectified Nonlinear Unit (ReLu) je zadaná predpisom $f(x) = max(0, x)$. Funkcia aj jej derivácia sú monotónne. Záporné vstupy sa však okamžite vynulujú, čo môže v špecifických prípadoch spôsobiť problém straty dát. Aj napriek tomu patrí medzi najpopulárnejšie aktivačné funkcie. Obor funkčných hodnôt je daný intervalom $\langle 0, \infty)$.

Ďalšou možnosťou pre aktivačnú funkciu je hyperbolický tangens (tanh). Táto funkcia je definovaná vzorcom 2.10, je monotónna, jej derivácia však nie je. Obor funkčných hodnôt je daný intervalom $(-1, 1)$, čím predstavuje ideálnu aktivačnú funkciu pre binárnu klasifikáciu.

$$f(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.10}$$



Obr. 2.5: Grafy aktivačných funkcií ReLu, tanh a sigmoidy

Predpis logistickej funkcie (sigmoidy) je definovaný vzorcom 2.11. Hlavnou výhodou tejto funkcie je, že obor jej funkčných hodnôt je na intervale $(0, 1)$, a teda umožňuje modelovanie pravdepodobnosti. Rovnako ako hyperbolický tangens je monotónna, pričom jej derivácia nie je.

$$f(x) = \frac{1}{1 - e^{-x}} \tag{2.11}$$

Generalizáciou logistickej funkcie je funkcia `softmax`, ktorá je definovaná vzorcom 2.12, kde $\vec{x}$ je vektor K vstupných hodnôt vrstvy. Táto aktivačná funkcia predstavuje aproximáciu funkcie maxima, ktorá by bola najideálnejšou pre klasifikáciu do vzájomne exkluzívnych tried. Funkcia maxima však nie je spojitá, a tak sa nedá derivovať a využiť pre učenie. Výstup tejto funkcie predstavuje pravdepodobnosť, že vstup patrí do danej triedy. Obor funkčných hodnôt tejto funkcie je totiž $(0, 1)$ a súčet výstupov je vždy 1.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \text{pre } i = 1, 2, 3, \ldots, K \tag{2.12}$$

2.3.2 Spätná propagácia

`Spätná propagácia` (anglicky Backpropagation) je základnou metódou, ktorá sa stará o učenie neurónových sietí. Ide o zovšeobecnenie Delta pravidla učenia pre viacvrstvové neurónové siete. Základným princípom je spätná propagácia chyby medzi jednotlivými vrstvami neurónovej siete. Učenie sa skladá z dvoch fáz, `propagácie` a `aktualizácie váh`.

V prvej fáze sa pre jednotlivé vstupy vypočítajú príslušné výstupy neurónovej siete a z nich sa následne pomocou chybovej funkcie vypočíta celková chyba siete. V nasledujúcich

príkladoch bude ako chybová funkcia využitá štvorcová chyba, definovaná vo vzorci 2.5. Výstupy jednotlivých aktivačných funkcií sa následne spätne propagujú po vrstvách od výstupnej vrstvy až po vstupnú. Týmto sa vypočítajú `delty`, ktoré predstavujú vplyv danej váhy na celkovú chybu a sú využité na výpočet inkrementu váhy. Výpočet delty je odvodený od vzorca na výpočet vektoru gradientu chyby 2.8. Pre naviazanie chyby na predchádzajúce vrstvy sa využíva reťazové pravidlo derivácie podľa vzorca 2.13, kde w_{ij} reprezentuje j-tú váhu vstupného spojenia neurónu i. Výraz out_i predstavuje výstup aktivačnej funkcie tohto neurónu a net_i jej vstup.

$$\nabla E = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_i}\frac{\partial out_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} \tag{2.13}$$

Prvú parciálnu deriváciu $\frac{\partial E_{total}}{\partial out_i}$ je možné pre výstupnú vrstvu priamo vyhodnotiť, pretože out_i korešponduje s výstupom siete o. Výsledkom je vzorec 2.14, kde o predstavuje skutočný výstup a d očakávaný.

$$\frac{\partial E}{\partial out_i} = \frac{\partial E}{\partial o} = \frac{\partial \frac{1}{2}(d-o)^2}{\partial o} = (o-d) \tag{2.14}$$

Pre ďalšie vrstvy je však potrebné počítať s tým, že E je funkcia vstupov všetkých neurónov množiny L, pričom táto množina predstavuje množinu neurónov prijímajúcich vstupy od neurónu i. Teda reprezentuje, aký vplyv má aktuálny neurón na celkovú chybu. Derivácia bude mať tvar 2.15. Neuróny množiny L sú o jednu vrstvu bližšie ku výstupnému neurónu, ktorého hodnotu poznáme. Rekurzívne použitie tohto vzorca umožňuje vypočítať túto deriváciu pre každú vrstvu.

$$\frac{\partial E}{\partial out_i} = \sum_{l \in L}\left(\frac{\partial E}{\partial net_l}\frac{\partial net_l}{\partial out_i}\right) = \sum_{l \in L}\left(\frac{\partial E}{\partial out_l}\frac{\partial out_l}{\partial net_l}w_{jl}\right) \tag{2.15}$$

Ďalšou parciálnou deriváciou je $\frac{\partial out_i}{\partial net_i}$. Tento výraz predstavuje deriváciu aktivačnej funkcie $\phi(net_i)$ podľa jej vstupu net_i,

$$\frac{\partial out_i}{\partial net_i} = \frac{\partial \phi(net_i)}{\partial net_i}. \tag{2.16}$$

Nakoniec derivujeme vstup aktivačnej funkcie net_i podľa váhy w_{ij}. Výraz net_i ako jediný obsahuje hodnotu váhy, pretože je definovaný ako vážený súčet vstupov. Vzorec 2.17 vyjadruje, že derivovaním vstupu neurónu podľa váhy spojenia získame hodnotu vstupného spojenia out_j, a teda výstup predchádzajúceho neurónu j.

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial \sum_{k=0}^{n}(w_{ik}out_k)}{\partial w_{ij}} = \frac{\partial w_{ij}}{\partial w_{ij}}out_j = out_j \tag{2.17}$$

Po spojení všetkých parciálnych derivácií získame upravený vzorec gradientu chybovej funkcie zo vzorca 2.13, v ktorom sa zbavíme výpočtu parciálnych derivácií. Finálny vzorec pre výpočet gradientu chybovej funkcie má tvar 2.18, kde o_j predstavuje výstup predchádzajúcevo neurónu a δ_i deltu súčasného neurónu.

$$\nabla E = o_j\delta_i \tag{2.18}$$

Už skôr spomínanú deltu δ_i je na základe vyjadrených parciálnych derivácií možné pre najmenšiu štvorcovú chybu a aktivačnú funkciu ϕ definovať ako rekurzívny vzorec 2.19.

$$\delta_i = \frac{\partial E}{\partial out_i}\frac{\partial out_i}{\partial net_i} = \begin{cases} (o-d)\,\frac{\partial \phi(net_i)}{\partial net_i} & \text{ak } j \text{ je výstupným neurónom} \\ \sum_{l \in L}(\delta_l w_{jl})\,\frac{\partial \phi(net_i)}{\partial net_i} & \text{inak} \end{cases} \tag{2.19}$$

V druhej fáze sa následne vypočítajú hodnoty inkrementu váhy na základe získaných hodnôt delt δ a nastavia nové váhy. Vzorec pre výpočet ikrementov váh má tvar 2.20, kde c predstavuje koeficient učenia a záporné znamienko zabezpečuje zostup po gradiente chybovej funkcie [23, str. 163–168].

$$\Delta w_{ij} = -co_j\delta_i \tag{2.20}$$

2.3.3 Chybové funkcie

Pre správne fungovanie algoritmu učenia neurónovej siete je nevyhnutná chybová funkcia na jej ohodnotenie. V sekcii 2.2.2 bola na tento účel využitá štvorcová chyba. V reálnych problémoch sa však namiesto tejto funkcie používa funkcia `strednej štvorcovej chyby` (skrátene MSE). Táto funkcia je daná predpisom vo vzorci 2.21, kde d_i predstavuje očakávaný výstup, o_i skutočný výstup a n počet výstupov. Ide o priemer štvorcových chýb, definovaných vo vzorci 2.5.

$$E = \frac{1}{n}\sum_{i=1}^{n}(d_i - o_i)^2 \tag{2.21}$$

Okrem nej je však možné pre neurónové siete použiť aj ďalšie chybové funkcie. Príkladom takej funkcie je `krížová entropia` (anglicky cross entropy). Predpis pri klasifikácii do dvoch tried je definovaný vzorcom 2.22, kde log predstavuje prirodzený logaritmus.

$$E = -(d\log(o) + (1-d)\log(1-o)) \tag{2.22}$$

Pre využitie na klasifikáciu do viac ako dvoch tried je potrebné funkciu upraviť na vzorec 2.23, kde n predstavuje počet tried.

$$E = -\sum_{i=1}^{n} d_i \log(o_i) \tag{2.23}$$

2.3.4 Optimalizačné algoritmy

Pre aktualizáciu váh však samotná spätná propagácia nestačí. Na jej využitie v reálnych prípadoch vznikli `optimalizačné algoritmy`. Základným optimalizačným algoritmom je `dávkový zostup gradientu` (anglicky batch gradient descent) [4, str. 5]. Princíp tohto algoritmu je založený na aktualizácii váh až po spracovaní všetkých položiek dátovej sady. To znamená, že pre výpočet inkrementu každej váhy je potrebné sčítať chyby všetkých položiek dátovej sady, čo je časovo aj priestorovo náročné. Z toho dôvodu vznikol algoritmus minidávkový zostup gradientu (anglicky minibatch gradient descent), ktorý si pri každej iterácii vyberie iba podmnožinu dátovej sady pevnej veľkosti.

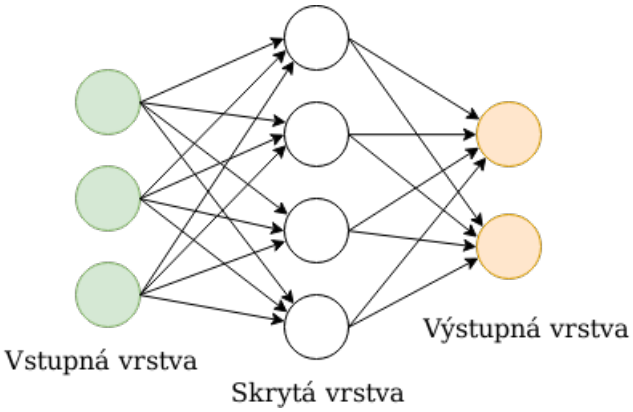
Na princípe zníženia množstva položiek využitých pri každej iterácii je postavený aj algoritmus `stochastický zostup gradientu` (skrátene SGD). Pri tomto algoritme sa najprv náhodne usporiadajú položky dátovej sady a následne sa táto usporiadaná postupnosť prechádza po jednej položke. Ku aktualizácii váhy dochádza po spracovaní každej položky.

Týmto spôsobom sa zmenší objem dát, ktorý je potrebné uložiť pre jednu iteráciu. Aktualizácia po každej položke však znamená, že niektoré položky spôsobia zvýšenie chyby. Z tohto dôvodu sa stochastický zostup často kombinuje s minidávkovým zostupom.

Okrem vyššie spomínaných algoritmov existujú aj ďalšie, ktoré sú postavené na stochastickom zostupe. Tieto algoritmy využívajú adaptívnu úpravu koeficientu učenia. Patria medzi ne AdaGrad [12] (algoritmus adaptívneho gradientu), RMSProp [28] (propagácia strednej štvorcovej chyby) a Adam [15] (odhad adaptívneho momentu).

2.3.5 Viacvrstvový perceptrón

Na základe perceptrónu vznikol viacvrstvový model doprednej neurónovej siete `viacvrstvový perceptrón` (skrátene MLP). Jeho úloha je rovnaká ako jednovrstvového perceptrónu, a to klasifikovať dáta do tried. Viacvrstvový perceptrón však dokáže, vďaka robustnejšej architektúre, klasifikovať aj lineárne neoddeliteľné vzory. Viacvrstvový perceptrón sa skladá minimálne z troch vrstiev, a to vstupnej, skrytej a výstupnej, ale môže obsahovať aj viac skrytých vrstiev. Tieto vrstvy sú plne-prepojené, čo znamená, že každý neurón skrytej a výstupnej vrstvy má prepojenie na všetky neuróny predchádzajúcej vrstvy. Aktivačné funkcie musia byť nelineárne a derivovateľné. Populárnou metódou na jeho trénovanie je spätná propagácia. Tento typ neurónových sietí je možné použiť napríklad na `spracovanie prirodzené jazyka` (skrátene NLP). Obrázok 2.6 zobrazuje topológiu viacvrstvového perceptrónu s jednou skrytou vrstvou, ktorý berie na vstup tri hodnoty a klasifikuje na ich základe do dvoch tried.



Obr. 2.6: Viacvrstvový perceptrón

2.3.6 Ďalšie druhy neurónových sietí

Okrem sietí, ktoré vychádzajú priamo z perceptrónu, existujú aj ďalšie, ktoré majú svoju štruktúru upravenú na špecifickú úlohu. Medzi tieto neurónové siete patria `konvolučné neurónové siete` (skrátene CNN), ktoré sú spomínané v nasledujúcej kapitole 3. Ďalšími typmi neurónových sietí sú `neurónové siete s časovým oneskorením` [29] (skrátene TDNN) a pravdepodobnostné neurónové siete [34] (skrátene PNN).

Kapitola 3

Hlboké neurónové siete

Táto kapitola sa sústredí na podskupinu viacvrstvových neurónových sietí, ktoré sa nazývajú hlboké neurónové siete. Sekcia 3.1 predstaví motiváciu, ktorá stála za vznikom hlbokých neurónových sietí. V hlbokých neurónových sieťach sa okrem plne-prepojených vrstiev z predchádzajúcej kapitoly využívajú aj iné typy, ktoré sa následne spájajú do jednej neurónovej siete. Sekcia 3.2 predstaví jednu triedu takýchto sietí, ktorými sú konvolučné neurónové siete.

3.1 Motivácia

Počítače boli od svojho počiatku navrhované tak, aby dokázali spracovať úlohy dané formálnou špecifikáciou. Tieto úlohy dokázali vyriešiť rýchlejšie ako ľudia, ak boli vôbec pre ľudí riešiteľné. Problém však nastáva, ak ide o úlohy, ktoré nie je možné formálne popísať alebo to je časovo náročné. Medzi takéto úlohy patrí napríklad rozpoznávanie obrazu a zvuku. Človek dokáže tento problém vyriešiť okamžite, avšak, keď má tento problém formálne popísať, nastáva problém. Vzniklo niekoľko projektov, ktoré sa snažili tieto problémy riešiť pomocou `bázy znalostí`. Ide o komplexnú dátovú štruktúru, ktorá je využívaná expertnými systémami pre implementáciu umelej inteligencie. Žiaden projekt však neprinie-sol výrazné úspechy najmä kvôli potrebe vytvárať zložité formálne pravidlá. Človek totiž na riešenie tohoto problému využíva biologickú neurónovú sieť, ktorá je uložená priamo v mozgu a je učená od narodenia. Na obraz biologického modelu sa teda začali vytvárať umelé neurónové siete. Kľúčový vplyv na fungovanie neurónovej siete má `reprezentácia` dát, ktorá určuje formát používaných dát. Pre spracovanie neurónovou sieťou je potrebné získať `príznaky` (anglicky features) tejto reprezentácie. Tie sa v minulosti museli získavať ručne, čo bolo časovo náročné. Z toho dôvodu vznikol prístup, ktorý sa volá `príznakové učenie`. Jednou z prvých foriem tohoto učenia bol `autoenkóder`, ktorý najprv zakódoval vstup na inú reprezentáciu a tú následne dekódoval na pôvodnú. Táto sieť vo svojich váhach obsahovala určité príznaky, ktoré sa dali neskôr použiť ako vstup neurónovej siete. Rôzne autoenkódery sa tak používali na získanie rozličných príznakov. Pre príznaky je potrebné, aby sa zamerali na vlastnosti, ktoré dokážu jasne rozlíšiť medzi objektami. Tieto vlastnosti sa tiež nazývajú variačné faktory (anglicky variation factors) a patrí medzi ne napríklad farba a pozícia objektu. Získať vysokoúrovňové príznaky môže byť náročné, pretože požadujú chápanie na vyššej úrovni abstrakcie. Z toho dôvodu boli vytvorené hlboké neurónové siete [7, str. 2-4].

`Hlboké neurónové siete` dokážu získať tieto príznaky samé. Pri implementovaní hlbokých neurónových sietí je možné zložité reprezentácie vyjadriť pomocou kombinácie jednoduchších. Prvá úroveň dokáže napríklad rozoznať hrany. Ďalšia spojí hrany a rozozná rohy, po spojení ktorých je možné rozlíšiť časti objektov. Čím viac vrstiev neurónová sieť obsahuje, tým komplexnejšie príznaky je možné detegovať.

3.2 Konvolučné neurónové siete

Konvolučné neurónové siete predstavujú triedu hlbokých neurónových sietí, ktoré poskytujú výborné výsledky pri rozpoznávaní obrazu a zvuku. Ide o mnohovrstvový perceptrón, ktorý bol navrhnutý na rozonávanie dvojrozmerných útvarov s vysokou mierou nezávislosti na transformáciách, rotáciách a zmene veľkosti [9, str. 201]. Jednou z prvých hlbokých neurónových sietí, ktoré dosahovali prelomové výsledky pri rozpoznávaní obrazu, je Lenet-5 [17], viz sekciu 5.1. Táto neurónová sieť dokázala rozpoznať číslice tejto dátovej sady MNIST, viz sekciu 5.2.1, s presnosťou 99,05%. Medzi hlavné vrstvy konvolučných neurónových sietí patria `plne-prepojené` (anglicky fully-connected), `konvolučné` (anglicky convolutional) a `spájacie` (anglicky pooling) vrstvy.

3.2.1 Konvolučné vrstvy

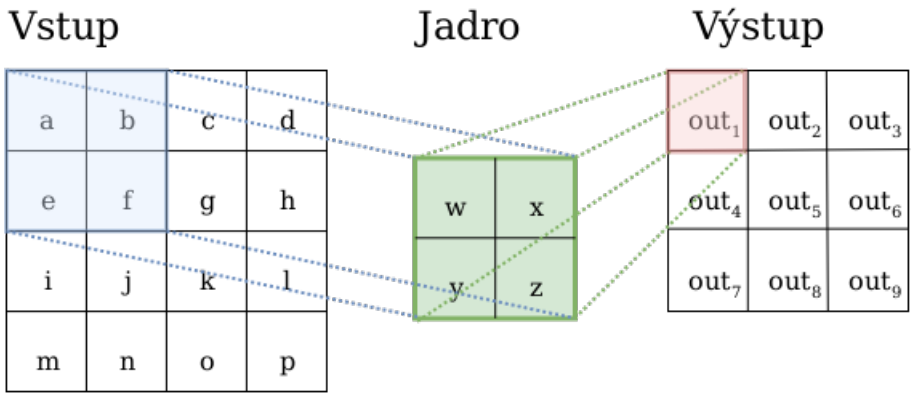
`Konvolúcia` je matematická operácia medzi dvomi funkciami s argumentami z oboru reálnych čísel [7, str. 327]. Zvyčajne sa zapisuje vo formáte $s(t) = (x * w)(t)$. V prípade konvolučných neurónových sietí x reprezentuje vstup a w `konvolučné jadro` (anglicky kernel) v kroku t. Výstupom je `aktivačná mapa` (anglicky feature map). V konvolučných neurónových sieťach sa však často využívajú dvojrozmerné vstupy, pre ktoré platí vzorec:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.1}$$

V tomto prípade konvolúcia prechádza dvomi osami i a j, pričom I predstavuje vstup a K predstavuje jadro, ktoré obsahuje jednotlivé váhy. V klasických neurónových sieťach sa však často využíva upravená verzia, ktorá sa nazýva vzájomná korelácia:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{3.2}$$

Hodnoty m a n predstavujú v upravenom vzorci koordináty na osiach jadra o veľkosti $M \times N$. Aplikácia konvolúcie jadra s rozmerom 2×2 dvojrozmerným vstupom s rozlíšením 4×4 je ilustrovaná na obrázku 3.1. Výstup bod $out1$ aktivačnej mapy má v tomto prípade hodnotu $wa + xb + ye + zf$. Týmto spôsobom prebehne konvolúcia jadra celým vstupom, čím získame výstupnú aktivačnú mapu o rozmeroch 3×3.



Obr. 3.1: Príklad konvolúcie vstupu o veľkosti 4×4 konvolučným jadrom s rozmermi 2×2

Vlastnosti

Klasické neurónové siete zvyčajne využívajú vektorový súčin medzi vstupným vektorom a vektorom váh spojení. Konvolučné neurónové siete sa však prezentujú **riedkou spojitosťou** (anglicky sparse connectivity). Toto je možné dosiahnuť použitím jadra, ktoré je menšie ako vstup. Príznaky je totiž mnohokrát možné detegovať na menšom počte okolitých bodov vstupu. Tieto body predstavujú **recepčné pole** výstupnej jednotky $S(i, j)$, ktorého rozlíšenie je definované veľkosťou jadra. Výhodou tohto prístupu je, že na rozdiel od maticového súčinu vyžaduje výrazne menej operácii, čím sa jeho výpočtová zložitosť znižuje.

Ďalšou vlastnosťou konvolučných neurónových sietí je **zdieľanie parametrov**. V klasických neurónových sieťach je každá váha matice použitá práve raz pri maticovom súčine. Konvolúcia však povoľuje znovu použitie váh jadra. Z toho vyplýva, že je možné využiť jednu maticu jadra na konvolúciu celým vstupom. Týmto spôsobom sa vytvoria filtre, ktorých váhy umožnia vyhľadávať špecifické príznaky. Výhodou zdieľania parametrov je zníženie pamäťovej náročnosti.

Treťou vlastnosťou, ktorú prináša konvolúcia a zdieľanie parametrov do neurónových sietí, je **ekvivariancia** ku posunu. V prípade, že sa body obsahujúce príznak posunú po niektorej osi vo vstupnej matici, hodnota nájdeného príznaku sa posunie aj vo výstupnej aktivačnej matici. Toto predstavuje výhodu využitia menšieho jadra, ktoré berie do úvahy iba niekoľko okolitých pixelov. V niektorých prípadoch je však zdieľanie parametrov nežiadané, napríklad ak potrebujeme rozoznať oči na tvári. Nie je nevyhnutné, aby jadro, ktoré sa stará o detekciu očí, hľadalo oči na brade alebo čele. Konvolúcia nie je ekvivariantná k ďalším transformáciám, ako napríklad rotácia a úprava mierky [7, str. 327].

Parametre

Okrem veľkosti jadra je možné nastaviť aj ďalšie parametre konvolučnej vrstvy. Jadro sa pri konvolúcii posúva po osiach vstupu po krokoch, ktoré sa nazývajú **strieda** (anglicky stride). Zachovanie rozlíšenia je možné dosiahnuť ďalším parametrom konvolučnej vrstvy, ktorým je **výplň** (anglicky padding). Ak by v príklade na obrázku 3.1 bola využitá výplň, znamenalo by to pridanie rámika vyplneného nulami okolo matice vstupu, čím by následný výstup mal rovnaké rozlíšenie ako pôvodný vstup. Pri konvolúcii trojrozmerného vstupu trojrozmerným jadrom, napríklad farebného obrázku, sa hodnota tretieho rozmeru nazýva **hĺbka** (anglicky depth). Ako aktivačná funkcia konvolučných vrstiev sa často používa ReLu.

3.2.2 Spájacie vrstvy

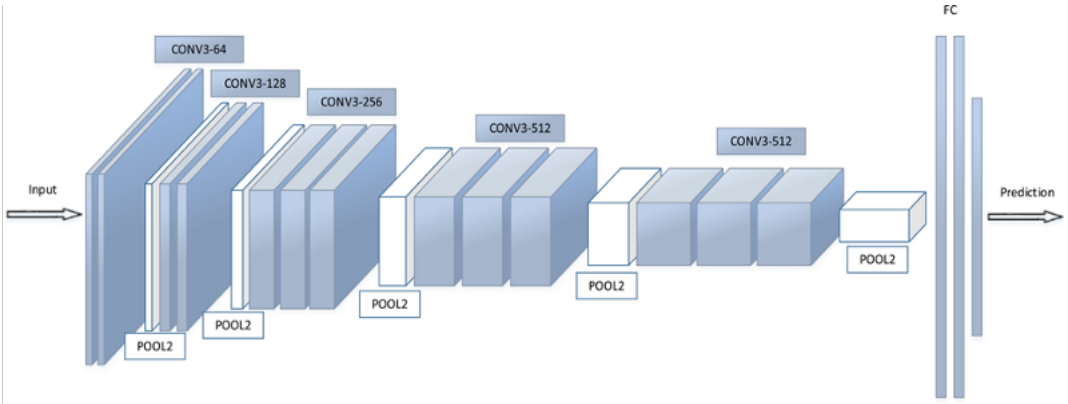
Spájacia funkcia nahrádza výstup siete na určitom mieste zhrňujúcou štatistikou okolitých výstupov [7, str. 335]. Cieľom tejto operácie je znížiť rozlíšenie aktivačnej mapy. Spájanie pomáha limitovať vplyv malých posunov na detekciu príznaku. Toto je podstatné, ak je existencia príznaku dôležitejšia ako jeho pozícia. Spájacia funkcia využíva obdĺžnikové okno s rozmermi $N \times M$, ktoré iteruje celým vstupom a spája hodnoty nachádzajúce sa v danom okne. Medzi najpoužívanejšie druhy spájacích vrstiev patrí `spájanie maximom` (anglicky max pooling) a `spájanie priemerom` (anglicky average pooling).

3.2.3 Plne-prepojené vrstvy

Plne-prepojené vrstvy sa využívajú najmä na výstupné vrstvy, kde je potrebné zmeniť počet neurónov na počet klasifikovaných tried. Pozerá sa na výstupy predchádzajúcej vrstvy, reprezentujúcu určitú úroveň príznakov a určí príznaky, ktoré sú najrelevantnejšie pre danú triedu.

3.2.4 Príklad konvolučnej neurónovej siete

Medzi populárne neurónové siete patria v súčasnosti architektúry postavené na modele neurónovej siete VGG-16 [27]. Jedna možná variácia sa nachádza na obrázku 3.2. Ide o architektúru, ktorá získala na prestížnej súťaži `ImageNet Challenge 2014` prvé a druhé miesto v lokalizačných a klasifikačných úlohách. Vstupom neurónovej siete je obrázok o veľkosti 224×224 v RGB formáte. Sieť sa skladá z niekoľkých konvolučných vrstiev `CONV3`. Tieto vrstvy využívajú recepčné pole o veľkosti 3×3 so striedou o veľkosti 1. Veľkosť aktivačnej mapy je rovnaká ako veľkosť vstupu vďaka výplni, ktorá zabraňuje zníženiu rozlíšenia. Všetky konvolučné vrstvy využívajú ReLu aktivačnú funkciu. O zníženie veľkosti aktivačnej mapy sa stará vrstva spájajúca maximá. Posledné tri vrstvy sú plne-prepojené, pričom posledná obsahuje soft-max aktivačnú funkciu.



Obr. 3.2: Architektúra neurónovej siete VGG-16, prevzaté z [14]

Kapitola 4

Knižnice

Táto kapitola sa sústredí na existujúce knižnice poskytujúce prostredie pre implementáciu hlbokých neurónových sietí v programovacom jazyku C++.

4.1 Caffe2

`Caffe2` je aplikačný rámec pre vývoj aplikácií využívajúcich strojové učenie hlbokých neurónových sietí. Vychádza z knižnice Caffe [13], ktorá bola vytvorená na Kalifornskej univerzite v Berkeley. Pôvodná knižnica Caffe bola užitočná na veľké projekty najmä kvôli výkonnej a dobre otestovanej C++ implementácii. Bola navrhnutá na implementáciu konvolučných neurónových sietí. Je použiteľná v programoch napísaných v programovacích jazykoch C++, Python a Matlab. Podporuje učenie ako na procesore, tak na grafickej karte. Neposkytovala však dostatočnú flexibilitu, čo stálo za vznikom Caffe2.

Caffe2 poskytuje na rozdiel od Caffe možnosť distribuovaného učenia, podporu rôznorodého hardvéru a mobilných zariadení. Dáta sú v Caffe2 organizované v bloboch (trieda `Blob`). Blob je pomenovaný úsek v pamäti, ktorý obsahuje najmä tenzory a v prípade jazyka Python aj polia knižnice Numpy. Všetky bloby sú uložené v pracovných prostrediach (trieda `workspace`), kde ich je možné vkladať alebo ich odtiaľ odobrať. Základnú abstrakciu knižnice Caffe2 tvorí sieť (trieda `net`), ktorá tvorí graf operátorov (trieda `Operators`). Operátory berú na vstup bloby a produkujú ďalšie bloby, teda predstavujú flexibilnú verziu vrstiev, aktivačných funkcií a chybových funkcií. Pred spustením neurónovej siete je potrebné vytvoriť jej model a ten inicializovať v pracovnom prostredí. Caffe2 umožňuje použitie predtrénovanej neurónovej siete zo súboru. Knižnicu Caffe2 je potrebné pre použitie na zariadenie inštalovať. Ide o open-source projekt šírený pod licenciou Apache License 2.0.

4.2 Tensorflow

Ďalšou knižnicou pre implementáciu hlbokých neurónových sietí v programovacom jazyku C++ je `Tensorflow`. Tensorflow je knižnica založená na práci s `tenzormi`, ktoré reprezentujú dátový tok neurónovej siete. Každý tenzor má svoj dátový typ a tvar. Neurónová sieť je v knižnici Tensorflow reprezentovaná grafom dátového toku (trieda `Graph`). Tento graf sa skladá z uzlov, ktoré reprezentujú operácie (trieda `Operation`) vykonávajúce funkciu neurónovej siete. Dáta sú medzi jednotlivými uzlami prenášané tenzormi (trieda `Tensor`) po hranách grafu. Tenzory reprezentujú dáta grafu, medzi ktoré patria jednotlivé váhy a dáta prechádzajúce grafom. Pri výpočtoch je graf predaný triede `Session`, ktorá vyhod-

notí výsledok grafu pre definovaný vstup. Medzi podporované programovacie jazyky patrí Python, C++, Java a Go. Knižnica podporuje učenie na grafickej karte. Tak ako v prípade Caffe2 ide o open-source projekt šírený pod licenciou Apache License 2.0.

4.3 Tiny-dnn

`Tiny-dnn` [21] je knižnica, ktorá sa prezentuje minimalistickou implementáciou hlbokých neurónových sietí v programovacom jazyku C++. Ide o knižnicu založenú iba na hlavičkových súboroch (anglicky header-only) bez závislosti na externých knižniciach. Pri použití je z toho dôvodu potrebné spolu s vlastným programom preložiť aj túto knižnicu, čím vznikne jeden prenositeľný binárny súbor bez závislostí na dynamických knižniciach. Knižnica je takto prispôsobená na použitie pre vstavané systémy a aplikácie IoT. Jej využitie na náročnejšie projekty však limituje nedokončená implementácia učenia na GPU, čo by výrazne urýchlilo výpočty. Knižnica taktiež obsahuje nedokončenú implementáciu kvantizačných výpočtov, vhodnú pre implementáciu na jednoduchšie zariadenia.

Základnú triedu tejto knižnice tvorí `network`, ktorá predstavuje model neurónovej siete. Táto trieda využíva šablónu, ktorá umožňuje vytvoriť dva typy sietí. Prvou možnosťou je sekvenčný model typu `sequantial`, ktorý umožňuje implementovať iba neurónové siete bez vetvenia vrstiev. Do pripraveného sekvenčného modelu sa jednotlivé vrstvy vkladajú pomocou operátoru `«`. Pre umožnenie vetvenia siete je potrebné využiť typ `graph`. Jednotlivé vrstvy sú postavené na triede `layer`. Knižnica implementuje niekoľko optimalizačných metód pre aktualizáciu jednotlivých váh, ako napríklad SGD, Adam, Adagrad a RMSProp. Po naučení neurónovej siete metódou `train` je možné sieť pre uloženie serializovať vo formáte json alebo v binárnej podobe. Knižnica tiež podporuje načítanie neurónových sietí knižnice Caffe2 zo sekcie 4.1. Implementuje tiež spracovanie obrazových súborov a dátových sád MNIST a CIFAR. Jej najväčšou výhodou je však jej kompaktnosť a jednoduchosť použitia, vďaka čomu s ňou dokáže pracovať aj začiatočník. Knižnica Tiny-dnn je open-source projekt šírený pod licenciou The BSD 3-Clause License.

Kapitola 5

Experimentálne overenie funkčnosti knižnice

Táto kapitola predstaví experimenty, ktorých úlohou bolo overiť funkčnosť knižnice Tiny-dnn zo sekcie 4.3 zvolenej pre účely tejto práce. Prvá sekcia 5.1 predstaví architektúru konvolučnej neurónovej siete LeNet-5, ktorá predstavuje základ neurónových sietí využitých v týchto experimentoch. Experimenty budú prevedené na dátových sadách, ktoré sú uvedené v sekcii 5.2. Sekcia 5.3 obsahuje jednotlivé experimenty.

5.1 LeNet-5

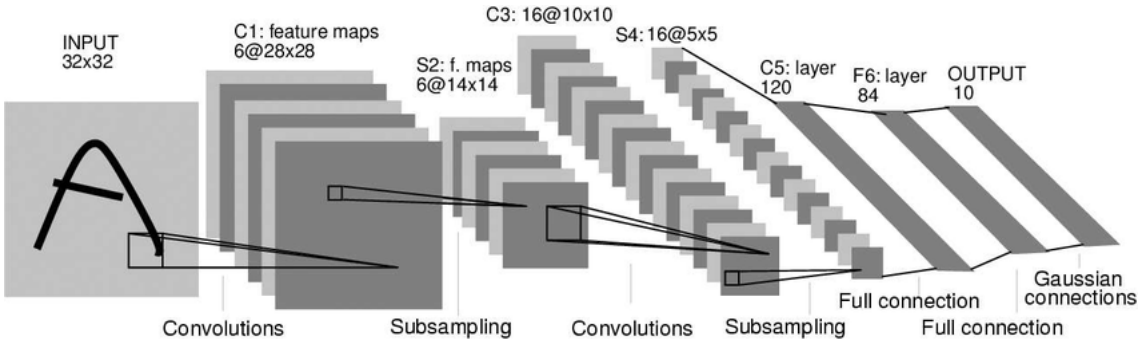
LeNet-5 [17] je architektúra konvolučnej neurónovej siete, ktorá sa skladá zo siedmich sekvenčne prepojených vrstiev. Táto architektúra využíva konvolučné vrstvy s filtrami veľkosti 5×5, striedou 1 a bez výplne. Spájacie vrstvy majú rozmer 5×5 a využívajú striedu 1. Vstup neurónovej siete tvoria obrázky znakov dátovej sady s rozmermi 32×32 pixelov. Pre načítanie vstupu a zároveň extrahovanie prvých príznakov sa používa konvolučná vrstva `C1`, ktorej výstupom je 6 aktivačných máp s rozmerom 28×28. Veľkosť aktivačných máp je následne zredukovaná spájacou vrstvou `S2` na rozmer 14×14. Zredukované mapy sú následne napojené na ďalšiu konvolučnú vrstvu `C3`, ktorá vytvorí 16 máp veľkosti 10×10 obsahujúce sekundárne príznaky. Tieto dve vrstvy však nie sú plne prepojené.

Pre zníženie úmery závislosti modelu na trénovanej sade sa využíva `tabuľka spojení` (anglicky connection table), ktorá spája iba špecifické aktivačné mapy z vrstvy S2 so vstupom vrstvy C3. Znaky `X` v tabuľke 5.1 reprezentujú platné spojenia medzi spojovanými

S2 \ C3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	O	O	O	X	X	X	O	O	X	X	X	X	O	X	X
1	X	X	O	O	O	X	X	X	O	O	X	X	X	X	O	X
2	X	X	X	O	O	O	X	X	X	O	O	X	O	X	X	X
3	O	X	X	X	O	O	X	X	X	X	O	O	X	O	X	X
4	O	O	X	X	X	O	O	X	X	X	X	O	X	X	O	X
5	O	O	O	X	X	X	O	O	X	X	X	X	O	X	X	X

Tabuľka 5.1: Tabuľka spojení medzi vrstvami S2 a C3 (X – prepojené, O – neprepojené)

vrstvami. Po vrstve C3 nasleduje ďalšia spájacia vrstva `S4`, ktorej výsledkom je 16 máp príznakov. Na túto vrstvu je napojená posledná konvolučná vrstva `C5`, ktorá zo 16 máp vrstvy S4 extrahuje 120 príznakov (mapy príznakov majú rozmer 1×1). Dátovú sadu MNIST, na klasifikáciu ktorej bola sieť navrhnutá, je však potrebné klasifikovať na 10 tried. Z toho dôvodu sa príznaky z vrstvy C5 postupne zredukujú na 10 výstupných neurónov využitím plne-prepojených vrstiev. Sieť je ukončená plne-prepojenou vrstvou s aktivačnou funkciou RBF (anglicky radial basis function). Originálny návrh sa nachádza v obrázku 5.1.



Obr. 5.1: Architektúra neurónovej siete LeNet-5, prevzaté z [17]

5.2 Dátové sady

Nasledujúce dátové sady predstavujú klasifikačné úlohy, ktorých riešenia sa využívajú na porovnanie jednotlivých klasifikačných algoritmov. Táto sekcia postupne predstaví dátové sady MNIST, CIFAR-10 a SVHN.

5.2.1 MNIST

`MNIST` [18] je dátová sada obsahujúca ručne písané číslice a ich označenie. Je to jednoduchá klasifikačná úloha, ktorá obsahuje iba desať na sebe nezávislých tried. Konvolučná neurónová sieť LeNet-5 dosahuje na tejto dátovej sade chybu v rozmedzí $0,8 - 0,95\%$ v závislosti na využití deformácií číslic. Deformácie totiž umožnia nájsť príznaky, ktoré inak môžu zostať nepovšimnuté. Najlepšie výsledky na tejto sade dosahuje zbor (anglicky committee) 35 konvolučných sietí s využitím elastických deformácií [6]. Klasifikačná chyba v tomto prípade dosiahla iba $0,23\%$, avšak kvôli svojej robustnosti je jej učenie časovo náročnejšie ako LeNet-5. Kvalita tejto neurónovej siete sa však blíži schopnostiam človeka, ktorý dokáže túto sadu klasifikovať s chybou približne $0,2\%$.

Dátová sada sa skladá zo štyroch súborov tvoriacich dvojice, z ktorých jeden reprezentuje dáta a druhý ich značky. Prvá dvojica je vytvorená pre použitie na učenie neurónovej siete a obsahuje 60000 príkladov. V druhej dvojici určenej na testovanie neurónovej siete sa nachádza 10000 príkladov, kde prvých 5000 je jednoduchšie rozpoznateľných ako posledných 5000. V súčasnosti sa MNIST považuje za jednoduchý problém a reálna evaluácia sa vykonáva na dátových sadách, ako napríklad CIFAR-10 a SVHN.

5.2.2 CIFAR-10

`CIFAR-10` [16] je dátová sada, ktorá rovnako ako MNIST obsahuje desať tried. Klasifikácia je však náročnejšia. Dátová sada sa skladá z0 60000 farebných obrázkov s rozlíšením 32, ktoré

sú rovnomerne rozdelené po 6000 obrázkov na triedu. Dátová sada je rozdelená na 50000 trénovacích obrázkov a 10000 testovacích obrázkov. Tie sú rozdelené do piatich trénovacích zložiek a jednej testovacej zložky. Testovacia zložka obsahuje 1000 obrázkov každej triedy a zvyšné obrázky sú náhodne rozdelené do testovacích zložiek po 10000 obrázkov na zložku. Triedy predstavujú dopravné prostriedky a živočíchy, menovite: lietadlo, auto, vták, mačka, jeleň, pes, žaba, kôň, loď a kamión. Všetky triedy sa vzájomne vylučujú, takže dátová sada neobsahuje veľké autá alebo malé nákladné autá, ktoré je ťažké rozlíšiť. Chyba klasifikácie sa v súčasnej dobe pohybuje medzi $11 - 26\%$.

5.2.3 SVHN

Dátová sada `SVHN` [20] sa, podobne ako dátová sada MNIST, využíva na klasifikáciu číslic. Táto dátová sada obsahuje fotografie čísel budov získaných z aplikácie `Google Street View`. Celkovou veľkosťou 600000 označených obrázkov z reálneho prostredia je výrazne náročnejšia na klasifikáciu ako spomínaný MNIST. Dátová sada je poskytovaná v dvoch možných formátoch, celé obrázky tabuliek s číslami alebo orezané jednotlivé číslice. V tejto práci bude využitý druhý spomínaný formát. Obrázky sú rozdelené na 73257 trénovácích a 26032 testovacích príkladov. Ostatné obrázky boli určené ako menej zložité a sú dostupné extra. Najúspešnejšia neurónová sieť na tejto dátovej sade je sieť publikovaná v článku [19] s chybou $1,69\%$. Bežná konvolučná neurónová sieť z článku [25] dosiahla chybu $4,9\%$ [1].

5.3 Experimenty

Táto sekcia obsahuje tri experimenty, ktoré overia funkčnosť knižnice Tiny-dnn na dátových sadách zo sekcie 5.2.

5.3.1 Experimentálne prostredie

Experimenty prebiehali na osobnom počítači `Lenovo Y50-70` využívajúc procesor `Intel Core i7-4720HQ` ($2,60$ GHz 6 MB), ktorý podporuje inštrukčnú sadu `AVX` využívanú knižnicou Tiny-dnn. Procesor bol podporovaný operačnou pamäťou `DDR3L SDRAM` s kapacitou 8 GB a kmitočtom 1600 MHz. Pre účely experimentov bola vytvorená testovacia trieda `Neural_net`, ktorá obsahuje jednotlivé implementácie neurónových sietí, ako aj upravený algoritmus učenia. Tento algoritmus kontroluje prvých 5 hodnôt validačnej chyby na testovacej sade. Validačná chyba je následne vypočítaná iba pre epochy, ktoré zodpovedajú zadefinovanému skoku. Pri základnom nastavení sa vždy preskakuje jedna epocha. Dôvodom využitia tohto skoku je zníženie množstva výpočtov validačnej chyby, ako výpočtovo náročnej operácie. Učenie taktiež využíva obmedzenie výpočtov, ktoré neprinášajú zlepšenie presnosti siete. Ak pri použití predvolených nastavení desať po sebe idúcich vyhodnotení neprinesie zlepšenie, učenie sa zastaví.

5.3.2 Konfigurácia parametrov učenia

Učenie prebiehalo po malých dávkach pomocou adaptívneho optimalizačného algoritmu Adagrad. Veľkosť dávky bola nastavená na hodnotu 16 a počiatočný koeficient učenia bol nastavený podľa príkladu v knižnici Tiny-dnn na hodnotu $0,01 \times \sqrt{davka}$, čo v tomto prípade znamená hodnotu $0,04$. Ako chybová funkcia bola použitá stredná štvorcová chyba (MSE). Učenie bolo obmedzené na 30 epoch (iterácií celou dátovou sadou), viz tabuľku 5.2. Učenie prebehlo desaťkrát využívajúc systémový čas ako semiačko (anglicky seed) pre

generátor náhodných čísel na inicializáciu váh spojení siete. Toto semiačko je v základnom nastavení knižnice konštantné, a teda klasifikácia je aj po opakovanom spustení zhodná.

Koeficient učenia	Dávka	Chybová funkcia	Zostup gradientu	Epochy
0.04	16	MSE	Adagrad	30

Tabuľka 5.2: Nastavenie parametrov učenia

5.3.3 Experiment 1

Experiment implementuje upravenú verziu neurónovej siete LeNet-5, pričom vychádza z návrhu obsiahnutého v príkladoch dostupných v knižnici Tiny-dnn. Pre tento experiment bolo vybrané učenie na dátovej sade MNIST.

Implementácia

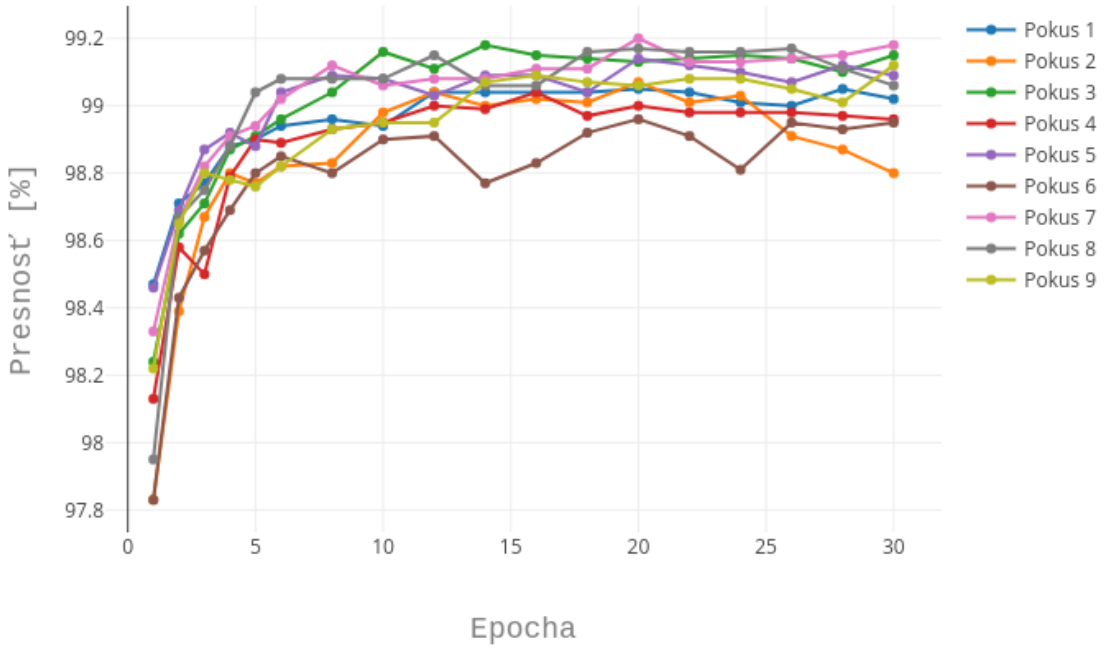
Implementácia neurónovej siete `net` sa nachádza vo výpise 5.1. Na rozdiel od pôvodného návrhu LeNet-5 sa po každej konvolučnej vrstve `conv` nachádza vrstva `relu`, ktorá reprezentuje aktivačnú funkciu ReLu. Medzi konvolučnými vrstvami sa nachádzajú dva druhy spájacích vrstiev. Sú nimi `max_pool`, spájajúca maximom, a `ave_pool`, ktorá pre každú oblasť vypočíta priemernú hodnotu. Tabuľka spojení 5.1 je definovaná v poli `S2_feature_map_tbl` a prevedená na tabuľku metódou `connection_table`. Poslednú vrstvu tvorí aktivačná funkcia softmax.

```
net << conv(32,32,5,1,6) << relu() // C1
    << max_pool(28,28,6,2) // S2
    << conv(14,14,5,6,16,
        connection_table(S2_feature_map_tbl,6,16)) << relu() // C3
    << ave_pool(10,10,16,2) // S4
    << conv(5,5,5,16,120) << relu() // C5
    << fc(120, 84) << relu() // F6
    << fc(84, 10) << softmax(); // F7
```

Výpis 5.1: LeNet-5 v Tiny-DNN

Výsledky

Na obrázku 5.2 je zobrazený vývoj presnosti klasifikácie sieťou na testovacej zložke počas učenia. Neurónová sieť dokázala v svojom maxime dosiahnu presnosť klasifikácie $99,2\%$, čo predstavuje chybu $0,8\%$ spadajúcu do referenčného intervalu zo sekcie 5.2.1. Jednotlivé pokusy dosahujú podobné výsledky. Počas jedného pokusu však nebola neurónová sieť, pre danú kombináciu inicializovaných váh, schopná učenia. Tento pokus z toho dôvodu nie je zahrnutý v grafe. Všetky ostatné pokusy dosiahli svoje maximum v intervale $98,96-99,2\%$. Na obrázku je tiež možné vidieť, že pokus 2 začal ku koncu učenia prudko klesať. Toto môže byť spôsobené pretrénovaním neurónovej siete, kedy sa neurónová sieť príliš viaže na trénovaciu sadu. Jedna epocha učenia trvala v priemere $37,046$ sekúnd.



Obr. 5.2: Graf priebehu učenia neurónovej siete LeNet-5 dátovou sadou MNIST

5.3.4 Experiment 2

Druhý experiment preveruje knižnicu Tiny-dnn na dátovej sade CIFAR-10 zo sekcie 5.2.2 na upravenej verzii neurónovej siete LeNet-5.

Implementácia

Implementácia neurónovej siete pochádza z príkladov neurónových sietí, ktoré sú obsiahnuté priamo v knižnici Tiny-dnn. Táto sieť tak ako v prvom experimente je postavená na neurónovej sieti LeNet-5, viz výpis 5.2. Na rozdiel od neurónovej siete v prvom experimente konvolučné vrstvy využívajú výplň na zachovanie rozlíšenia aktivačnej mapy pomocou parametru `tiny_dnn::padding::same`. Keďže vstup obsahuje tri farby, vstupná konvolučná vrstva obsahuje tri vstupné kanály. Zmena nastala aj v počte kanálov určujúcich počet aktivačných máp medzi jednotlivými vrstvami. Pre vyššiu náročnosť úlohy bolo potrebné zväčšiť šírku neurónovej siete. Aktivačná funkcia bola tiež presunutá až za spájaciu vrstvu.

```
net << conv(32, 32, 5, 3, 32, tiny_dnn::padding::same) // C1
    << max_pool(32, 32, 32, 2) << relu() // P2
    << conv(16, 16, 5, 32, 32, tiny_dnn::padding::same) // C3
    << max_pool(16, 16, 32, 2) << relu() // P4
    << conv(8, 8, 5, 32, 64, tiny_dnn::padding::same) // C5
    << max_pool(8, 8, 64, 2) << relu() // P6
    << fc(4 * 4 * 64, 64) << relu() // FC7
    << fc(64, 10) << softmax(); // FC10
```

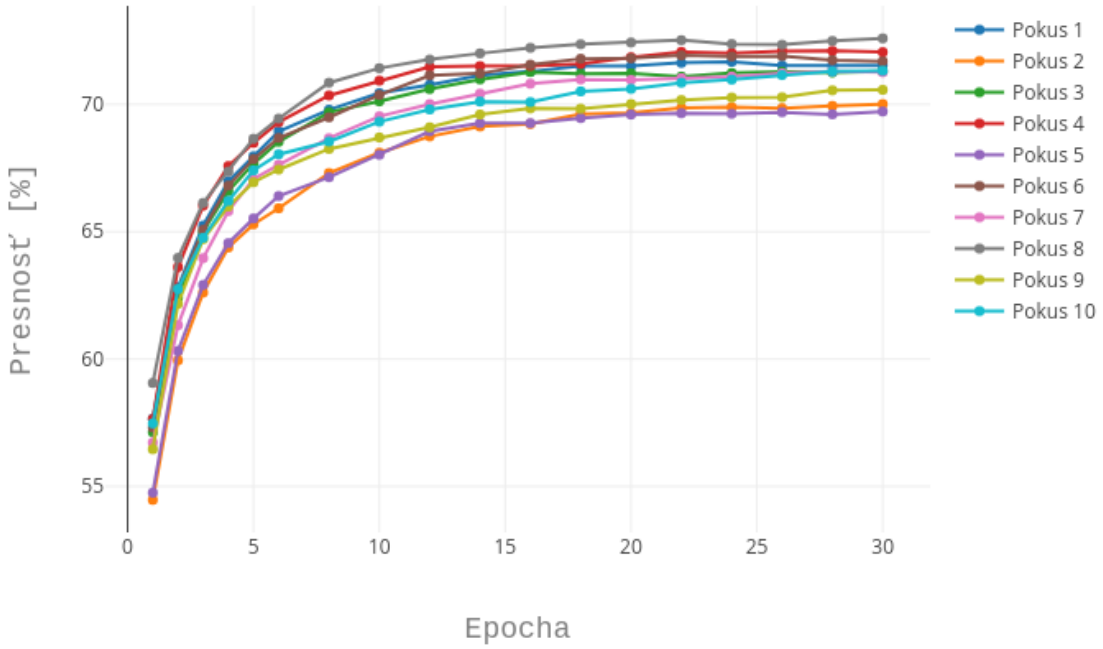
Výpis 5.2: LeNet-5 pre CIFAR-10 v Tiny-dnn

Učenie

Koeficient učenia musel byť v tomto prípade znížený na hodnotu $0,002 \times \sqrt{davka}$. Pôvodná hodnota bola totiž príliš vysoká a neurónová sieť nebola schopná sa učiť. Toto môže byť zapríčinené prekročením globálneho minima chybovej funkcie už po prvom kroku. Použitá hodnota bola zvolená na základe prevedených experimentov, kedy sa presnosť klasifikácie po 30 epochách stabilizovala.

Výsledky

Graf na obrázku 5.3 zobrazuje priebeh učenia neurónovej siete z výpisu 5.2 na dátovej sade CIFAR-10. Neurónová sieť dosiahla v svojom maxime presnosť $72,59\%$, čo prestavuje klasifikačnú chybu $27,41\%$. Táto chyba o $1,41\%$ prevyšuje hornú hranicu referenčného intervalu zo sekcie 5.2.2. Narozdiel od prvého experimentu je učenie stabilnejšie a nedochádza k výraznejšiemu zníženiu presnosti klasifikácie v priebehu učenia. Jedna epocha učenia trvala priemerne $108,52$ sekúnd.



Obr. 5.3: Graf priebehu učenia neurónovej siete dátovou sadou CIFAR-10

5.3.5 Experiment 3

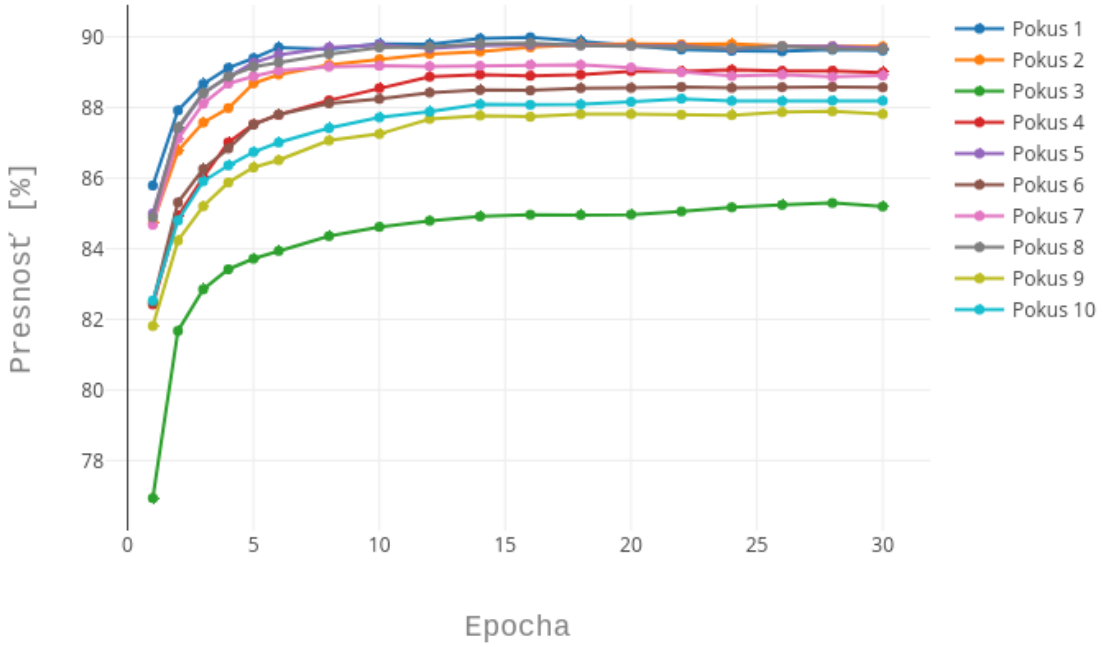
Posledný experiment využíva neurónovú sieť využitú v sekcii 5.3.4 na dátovej sade SVHN.

Učenie

Rovnako ako v prípade druhého experimentu sa nebola neurónová sieť schopná učiť pri koeficiente učenia využitom v prvom experimente. Z toho dôvodu bol znížený na hodnotu $0,005 \times \sqrt{davka}$, ktorá bola rovnako ako v sekcii 5.3.4 zistená experimentálne.

Výsledky

Neurónová sieť dosiahla vo svojom najvyššom bode presnosť klasifikácie 89, 98%. Chyba, ktorá predstavuje 10, 02%, dvojnásobne prevýšila referenčné hodnoty. Toto môže byť zapríčinené nedostatočnou hĺbkou alebo šírkou neurónovej siete pre tento klasifikačný problém. Podľa grafu na obrázku 5.4 dosahujú jednotlivé prípady učenia podobné hodnoty. Pre jeden pokus z desiatich však kombinácia inicializovaných váh spôsobila výrazne nižšiu presnosť klasifikácie.



Obr. 5.4: Graf priebehu učenia neurónovej siete dátovou sadou SVHN

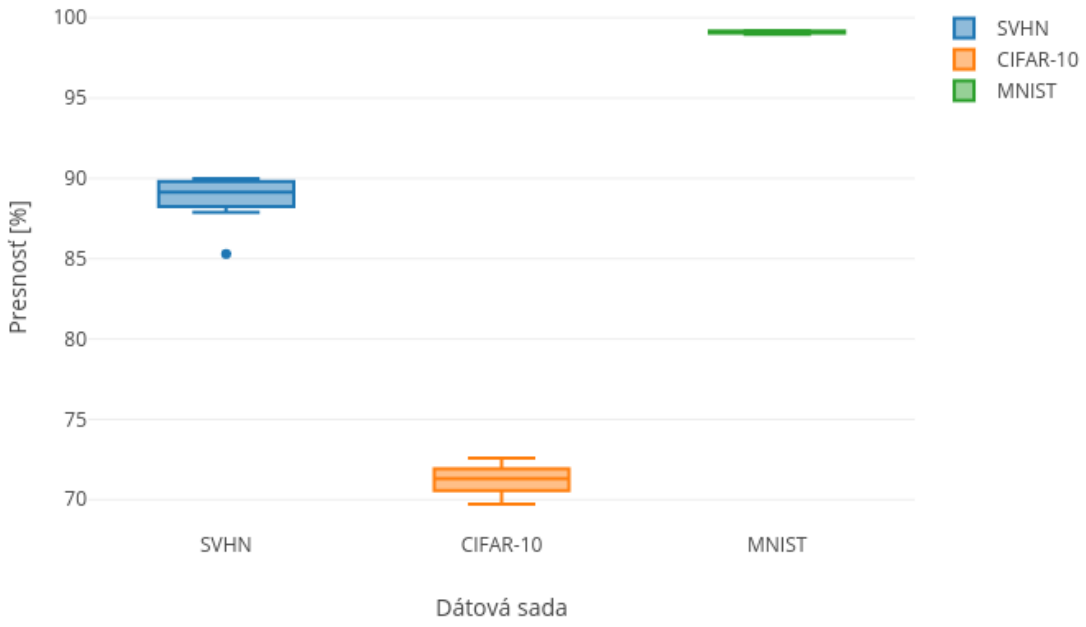
5.3.6 Porovnanie výsledkov

Na základe dát získaných z experimentov bol vytvorený graf obrázku 5.5, ktorý predstavuje štatistické zhodnotenie maximálnych hodnôt jednotlivých behov učenia neurónovej siete pre dátové sady SVHN a CIFAR-10. Dátová sada SVHN obsahuje jednu hodnotu, ktorá leží mimo rozsah boxplotu. To znamená, že pre jednu z desiatich náhodných inicializácií váh nebola dosiahnutá požadovaná kvalita neurónovej siete. Podľa grafu medzikvartilový rozsah predstavuje pre dátovú sadu SVHN 1, 5597% a pre sadu CIFAR-10 1.35%. V prípade dátovej sady MNIST bol medzikvartilový rozsah ešte menší a dosiahol hodnotu 0, 125%.

Dátová sada	Priemer	Smerodajná odchýlka	Medián	Medzikvartilový rozsah
SVHN	88.76997	1.3468	89.13645	1,5597
CIFAR-10	71.249	0.8656	71.31	1.35
MNIST	99.1033	0.0744	99.12	0,125

Tabuľka 5.3: Vyhodnotenie najlepších klasifikácií dátových sád

Na základe medzikvartilového rozsahu a smerodajnej odchýlky z tabuľky 5.3, ktorá sa v najhoršom prípade pohybuje okolo hodnoty 1%, je možné náhodnú inicializáciu váh pre tieto neurónové siete ďalej zanedbávať.



Obr. 5.5: Porovnanie nalepších hodnôt jednotlivých pokusov učenia neurónových sietí na dátových sadách SVHN, CIFAR-10 a MNIST

Kapitola 6

Optimalizácie neurónovej siete

Pre získanie požadovaných vlastností neurónovej siete je často potrebné previesť jej optimalizáciu. Táto kapitola sa zaoberá niekoľkými optimalizáciami neurónových sietí, ktorých cieľom je zlepšiť jeden zo sledovaných parametrov. Medzi tieto parametre patria najmä presnosť riešenia úlohy, výpočtová a pamäťová zložitosť. Presnosť neurónovej siete je možné optimalizovať už počas učenia správnym výberom hyper-parametrov, ktorým sa zaoberá sekcia 6.1. Za účelom zníženia výpočtovej náročnosti, napríklad pre využitie na vstavaných zariadeniach, je možné operácie s pohyblivou rádovou čiarkou nahradiť operáciami nad celými číslami. Touto tématikou sa zaoberá sekcia 6.2. Posledná sekcia 6.3 predstaví možnosť zníženia pamäťovej náročnosti obmedzením počtu desatinných miest potrebných na uloženie váh neurónovej siete. Jednotlivé experimenty prebiehali v prostredí špecifikovanom v sekcii 5.3.

6.1 Optimalizácia hyper-parametrov

Hlboké neurónové siete obsahujú desiatky hyper-parametrov, ktoré ovplyvňujú, ako sa bude natrénovaná neurónová sieť správať. Medzi tieto parametre patrí napríklad koeficient učenia, chybová funkcia alebo hĺbka, resp. šírka, neurónovej siete. Výber nastavovaných a pevných hyper-parametrov závisí na človeku, ktorý vykonáva experimenty. Nájdenie hodnoty jedného hyper-parametru nepredstavuje zložitý problém. Pri nastavovaní viacerých hyper-parametrov však vzniká viacrozmerný priestor a je potrebné vziať do úvahy, že jednotlivé hyper-parametre sa môžu navzájom ovplyňovať. Z toho dôvodu sa využívajú rôzne optimalizačné algoritmy, ktoré efektívne prechádzajú tento multidimenzionálny priestor s cieľom nájsť optimálnu kombináciu. V tejto sekcii budú využité tri z nich, a to `ručné ladenie`, `sekvenčné prehľadávanie` (anglicky grid search) a `náhodné prehľadávanie` (anglicky random search) [26]. Medzi zložitejšie optimalizačné algoritmy patria `Bayesovská optimalizácia` a `stromovo štruktúrovaný Parzenov odhad` [2].

6.1.1 Ručné ladenie

Najjednoduchším spôsobom optimalizácie hyper-parametrov neurónovej siete je experimentálne vyskúšať niekoľko ručne zvolených hodnôt hyper-parametru. Na základe ich výsledkov zvoliť ten, ktorý najviac vyhovuje špecifikácii. Tento postup je možné bez problémov využiť v prípade, že optimalizujeme iba jeden hyper-parameter, ktorého hodnotu vieme odhadnúť na základe predchádzajúcej skúsenosti alebo referenčnej literatúry.

Problém

Neurónová sieť, ktorú budeme optimalizovať, sa nachádza vo výpise 6.1. Ide o neurónovú sieť založenú na architektúre LeNet-5, ktorá bola prevzatá z príkladov ku knižnici Tiny-dnn. Ručným ladením tejto neurónovej siete vznikla neurónová sieť využitá v experimente v sekcii 5.3.3. Pri tejto optimalizácii neboli upravované parametre učenia, ktoré sú definované v tabuľke 6.1, ale samotná architektúra siete. Učenie prebiehalo na dátovej sade MNIST.

```
net << conv(32,32,5,1,6) << tanh() // C1
    << ave_pool(28,28,6,2) // S2
    << conv(14,14,5,6,16,
        connection_table(S2_feature_map_tbl,6,16)) << tanh() // C3
    << ave_pool(10,10,16,2) // S4
    << conv(5,5,5,16,120) << tanh() // C5
    << fc(120, 10) << tanh(); // F6
```

Výpis 6.1: Pôvodná sieť LeNet-5 v Tiny-DNN

Koeficient učenia	Dávka	Chybová funkcia	Zostup gradientu	Epochy
0.032	16	MSE	Adagrad	30

Tabuľka 6.1: Nastavenie parametrov učenia

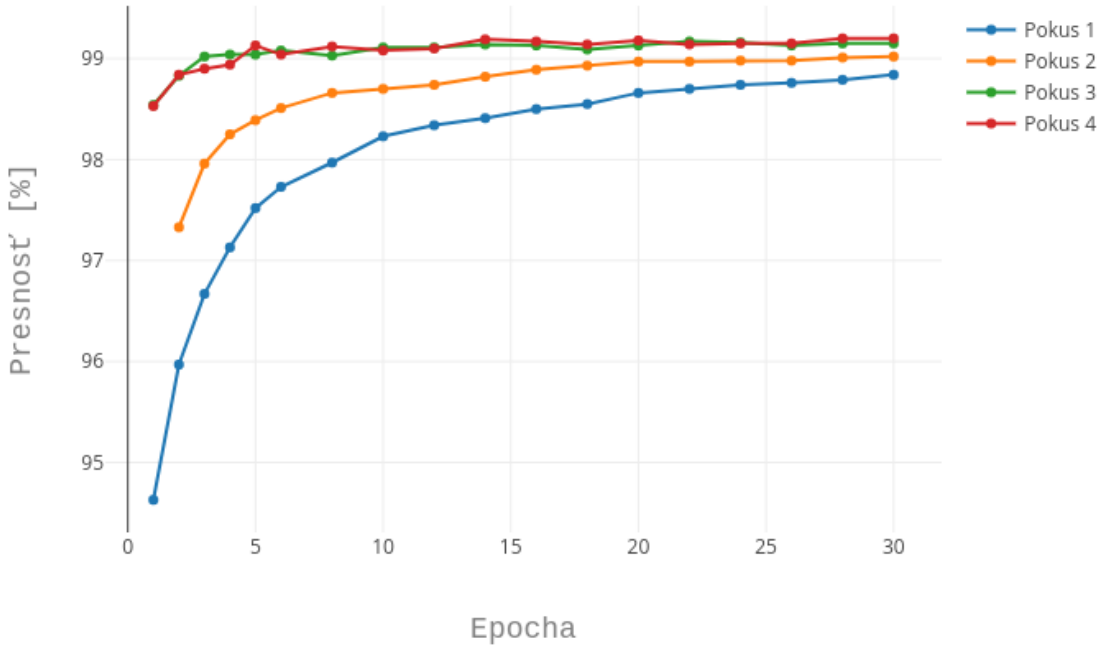
Optimalizácia

Učením pôvodnej siete bola dosiahnutá maximálna presnosť $98,84\%$, čo nezodpovedá požadovanej presnosti zo sekcie 5.2.1. V grafe na obrázku 6.1 je priebeh jej učenia označený ako `Pokus 1`. V prvom kroku optimalizácie (viz `Pokus 2`) bola prvá spájacia vrstva prevedená zo spájania priemerom (`ave_pool`) na spájanie maximom (`max_pool`). Z grafu bola pre vyššiu prehľadnosť odstránená prvá validačná hodnota presnosti 87.38% pre Pokus 2. Pri spájaní maxím sa zanedbávajú nízke aktivačné hodnoty v okolí, a tým sa zabraňuje prenosu šumu na ďalšiu vrstvu. Keďže sa pracuje s nízkou úrovňou príznakov, ako sú napríklad diagonály, je možné tieto hodnoty pre dátovú sadu MNIST zanedbať. Týmto spôsobom sa zvýšila presnosť klasifikácie neurónovej siete na hodnotu $99,02\%$. Pri opakovanom prevedení experimentu s inými počiatočnými váhami bola presnosť pôvodnej verzie siete o 0.01% lepšia ako optimalizovaná. Táto optimalizácia teda nemusí za každých okolností poskytnúť zlepšenie siete, avšak pre vyššiu maximálnu presnosť bola ponechaná. Pri pokuse nahradiť aj druhú spájaciu vrstvu za `max_pool` sa však presnosť klasifikácie mierne zhoršila na hodnotu $98,96\%$. Z toho dôvodu bola nahradená iba prvá spájacia vrstva.

V druhom kroku optimalizácie (viz `Pokus 3`) bola následne nahradená aktivačná funkcia `tanh`. Na skrytých vrstvách bola táto funkcia nahradená za `ReLu` funkcie a na výstupnej vrstve za funkciu `Softmax`. Funkcia ReLu, kvôli monotónnosti jej gradientu, patrí medzi najobľúbenejšie aktivačné funkcie pre konvolučné siete. Výber funkcie Softmax na výstupnú vrstvu pre dátovú sadu MNIST je odôvodnený v sekcii 2.3.1. Presnosť klasifikácie v tomto prípade dosiahla hodnotu $99,17\%$. Po opakovaní experimentu bola dosiahnutá presnosť $99,22\%$.

Posledným krokom (viz `Pokus 4`) optimalizácie bolo zvýšenie hĺbky pridaním plne-prepojenej vrstvy obsahujúcej 84 neurónov, tak aby sieť zodpovedala pôvodnému návrhu

LeNet-5. Táto optimalizácia však nepriniesla zlepšenie. Presnosť klasifikácie bola $99,2\%$, pričom po opakovaní učenia bola dosiahnutá presnosť iba $99,08\%$ a $99,15\%$.



Obr. 6.1: Priebeh učenia optimalizovaných neurónových sietí

Výsledok

Ručné ladenie prinieslo požadované výsledky, vyžaduje si však porozumenie problematiky použitej dátovej sady, ako aj informácie z odbornej literatúry a iných zdrojov. Využitie tejto optimalizačnej techniky je tiež limitované počtom optimalizovaných hyper-parametrov. V tomto experimente bola optimalizovaná štruktúra neurónovej siete, ktorej požadovaná presnosť bola známa. Priestor optimalizovaných parametrov bol taktiež malý, keďže v knižnici Tiny-dnn sa nachádzajú iba dve spájacie vrstvy a výber aktivačných funkcií bol inšpirovaný naštudovanou literatúrou. Viacdimenzionálny priestor hyper-parametrov, ktoré môžu dosahovať omnoho viac hodnôt, však nie je možné týmto spôsobom plne pokryť.

6.1.2 Sekvenčné prehľadávanie

Pre nastavenie hyper-parametrov, ktorých hodnotu nie je možné priamo odvodiť z literatúry alebo obdobných riešení, je potrebné použiť komplexnejší prístup. Jednou možnosťou optimalizácie týchto hyper-parametrov je sekvenčné prehľadávanie nimi určeného priestoru, ktorý je definovaný množinou užívateľom zvolených hodnôt. Tá môže obsahovať presné špecifikované hodnoty alebo interval hodnôt, z ktorého budú rovnomerne vybrané hodnoty. Pri využití tohto algoritmu sa prehľadajú všetky možné kombinácie zadefinovaných hyper-parametrov, z čoho vyplýva, že ide o výpočtovo náročnú úlohu.

Problém

Pri využití architektúry neurónovej siete LeNet-5 definovanej vo výpise 5.1 je potrebné nájsť najlepšiu kombináciu parametrov učenia pre dátovú sadu MNIST. Kvôli výpočtovej náročnosti prehľadávania všetkých kombinácií parametrov, bola vybraná iba dvojica parametrov, ktoré sa budú v tomto príklade optimalizovať. Sú nimi koeficient učenia a optimalizačný algoritmus zostupu gradientu.

Konfigurácia parametrov učenia

Tabuľka 6.2 obsahuje pevne stanovené parametre učenia, ktoré sú prevzaté zo sekcie 5.3. Hodnoty koeficientu učenia a optimalizačného algoritmu, ktoré boli prehľadávané, sa nachádzajú v tabuľke 6.3. Optimalizačné algoritmy boli zvolené tak, aby boli obsiahnuté opačné strany spektra, teda komplexný algoritmus s adaptívnym koeficientom učenia Adagrad, ale aj základný algoritmus zostupu bez modifikácie koeficientu učenia SGD. Hodnoty koeficientu učenia obsahujú najvyššiu hodnotu $0,04$, ktorá, ako bolo zistené v predchádzajúcich experimentoch, stále umožňuje učenie neurónovej siete a dve mierne nižšie hodnoty. Ďalšie dve hodnoty boli zvolené tak, aby pokryli aj nižšie hodnoty. Toto rozloženie sa snaží ilustrovať vplyv nízkych a vysokých váh na finálnu presnosť naučenej neurónovej siete pri obmedzení počtu epoch.

Dávka	Chybová funkcia	Epochy
16	MSE	30

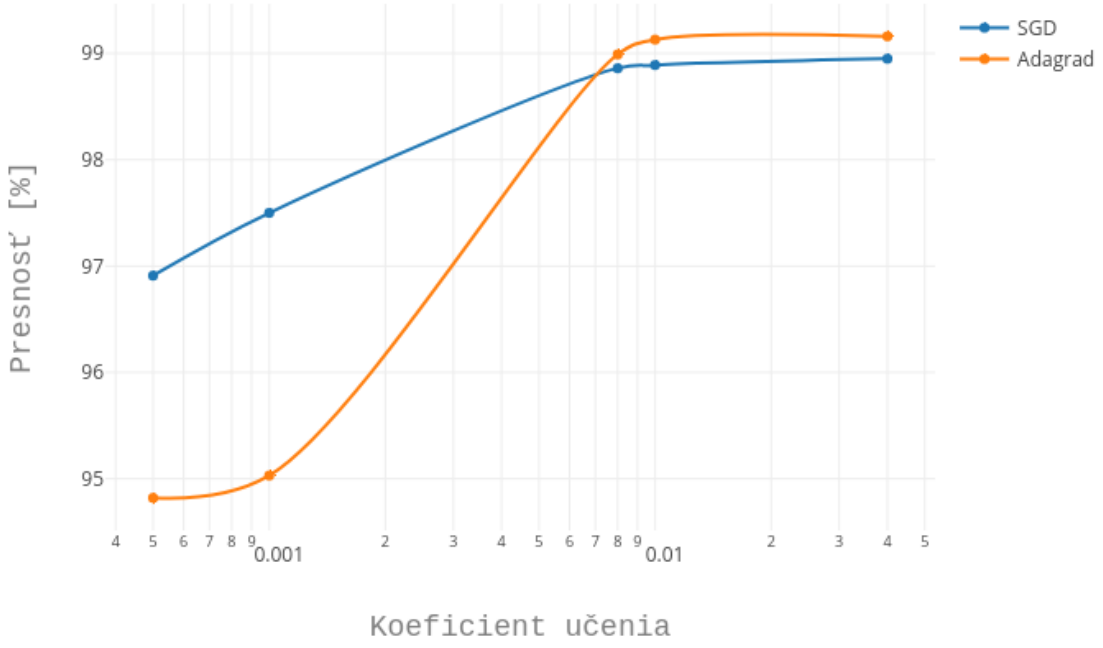
Tabuľka 6.2: Nastavenie parametrov učenia

Koeficient učenia	0.04	0.01	0.008	0.001	0.0005
Zostup gradientu	SGD	Adagrad			

Tabuľka 6.3: Optimalizované parametre učenia

Výsledok

Najvyššiu presnosť klasifikácie $99,16\%$ dosiahla kombinácia algoritmu zostupu Adagrad a koeficientu učenia $0,04$. V prípade využitia základného algoritmu zostupu gradientu SGD bola najvyššia presnosť na hodnote $98,95\%$, ktorá leží mimo rozsah smerodajnej odchýlky najlepšej kombinácie definovanej v tabuľke 5.3. Optimalizačný algoritmus Adagrad totiž dokáže vďaka adaptívnej aktualizácii gradientu nájsť minimum chybovej funkcie s väčšou presnosťou. Pri využití optimalizačného algoritmu Adagrad bola dosiahnutá vyššia presnosť aj pre hodnoty koeficientu učenia $0,01$ a $0,008$. Pre jeho nižšie hodnoty $0,001$ a $0,0005$ však učenie algoritmom SGD dosiahol vyššiu presnosť klasifikácie. Algoritmus Adagrad totiž nízku počiatočnú hodnotu koeficientu učenia ešte zníži, čo má za následok pomalý zostup gradientom a počet epoch nestačil na dosiahnutie gradientu. Optimalizačný algoritmus Adagrad s vysokým koeficientom učenia je teda v porovnaní s algoritmom SGD jasným víťazom. Hodnota koeficientu učenia však nesmie, ako bolo zistené v sekcii 5.3.4, prekonať istú hodnotu, pretože stratí schopnosť učiť sa.



Obr. 6.2: Sekvenčné prehľadávanie priestoru určeného optimalizačným algoritmom a koeficientom učenia

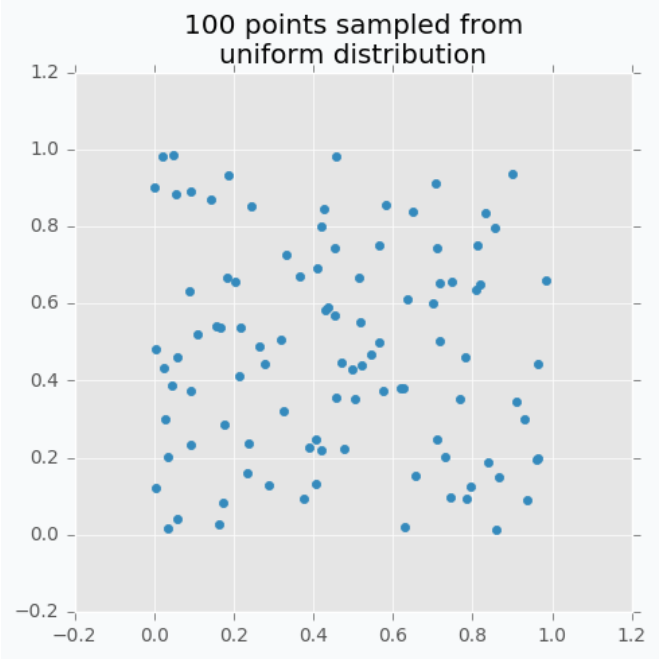
6.1.3 Náhodné prehľadávanie

Náhodné prehľadávanie priestoru hyper-parametrov umožňuje, na rozdiel od sekvenčného, znížiť počet prípadov učenia neurónovej siete. Učenie je výpočtovo náročná úloha, a teda pre získanie relevantných výsledkov v relatívne krátkom čase je potrebné tento počet limitovať. Pri využití tohto prehľadávania sa vyhodnocujú iba náhodne vybrané n-tice hyper-parametrov. Tie je však potrebné vygenerovať tak, aby rovnomerne pokrývali celý priestor hyper-parametrov.

Je ich možné získať využitím generátoru náhodných čísel definovaným pravdepodobnostným rozložením, napríklad rovnomerným. Prvých sto hodnôt vygenerovaných týmto rozložením sa nachádza v grafe na obrázku 6.3.

Haltonova postupnosť

Tieto hodnoty však nepokrývajú priestor dostatočne rovnomerne. Z toho dôvodu je podľa článku [26] vhodné nahradiť generátor náhodných čísel kvázináhodnou postupnosťou. Pre účely tejto práce bola využitá `Haltonova postupnosť`. Túto postupnosť je možné využiť ako náhradu za generátor náhodných čísel, pretože dokáže rovnomerne a efektívne pokryť jednotkový N-rozmerný priestor. Pre jeden rozmer je Haltonova postupnosť generovaná výberom prvočísla r a rozširovaním postupnosti celých čísel $1, 2, \ldots, g, \ldots G$ na základ r podľa vzorca 6.1, kde $0 \leq b_l(g) \leq r - 1$ a $r^L \leq g \leq r^{L+1}$. Prvá podmienka znamená, že hodnota číslic $b_l(g)$ pri rozšírení základu musí byť nižšia ako hodnota základu. Druhá podmienka určuje maximálnu hodnotu l pri rozšírení celého čísla g na základ r. Hodnota l reprezentuje mocninu, na ktorú bude základ zvýšený.



Obr. 6.3: Generátor náhodných čísel postavený na rovnomernom rozložení, prevzaté z [26]

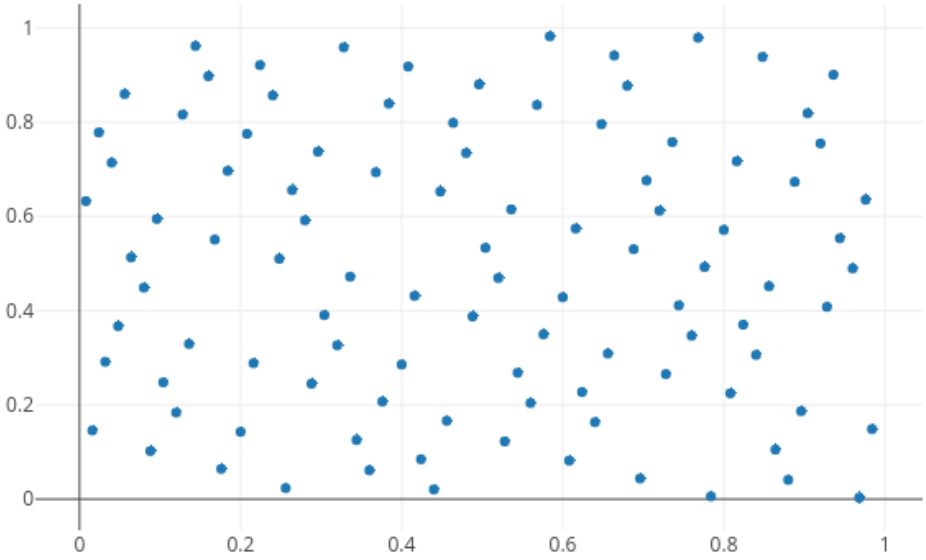
$$g = \sum_{l=0}^{L} b_l(g) r^l \tag{6.1}$$

Premenná g ($g = 1, 2, \ldots, G$) teda reprezentuje digitalizovanú hodnotu celočíselného reťazca $b_L(g)b_{L-1}(g)\ldots b_1(g)b_0(g)$. Haltonova postupnosť je získaná radikálnou inverziou g na základ r zobrazením cez radikálny bod:

$$\varphi_r(g) = 0 \cdot b_0(g)b_1(g)\ldots b_{L-1}(g)b_L(g) \quad \text{v základe } r = \sum_{l=0}^{L} b_1(g) r^{-l-1}. \tag{6.2}$$

Ako príklad si vyberieme hodnotu prvočíselného základu $r = 3$ a celého čísla postupnosti $g = 5$. Číslo 5 je možné pri prevode na základ 3 rozpísať ako $5 = 1 \times 3^1 + 2 \times 3^0$. Jeho digitalizáciou získame číslo v trojkovej sústave 12. Otočením okolo radikálneho bodu funkciou $\varphi_r(g)$ dostaneme hodnotu $0,21$, rozšírením ktorej na základ 3 získame hodnotu $2 \times 3^{-1} + 1 \times 3^{-2} = 7/9$. Hodnota $7/9$ teda reprezentuje piatu hodnotu Haltonovej postupnosti o základe 3. Vo všeobecnosti platí, že postupnosť o základe r obsahuje cykly dĺžky r monotónne stúpajúcich čísel [3].

Po vygenerovaní dvojrozmerného priestoru skladajúceho sa zo 100 bodov, kde jedna os predstavuje Haltonovu postupnosť o základe 2 a druhá os o základe 3, získame priestor na obrázku 6.4. Pri porovnaní s priestorom vygenerovaným generátorom náhodných čísel založenom na rovnomernom rozložení sú body rozmiestnené rovnomernejšie, zachovávajú si však istú dávku náhodnosti. Pre vyšší počet dimenzií je možné použiť vyššie hodnoty základu.



Obr. 6.4: Prvých sto bodov vygenerovaných pomocou Haltonových postupností o základe 2 a 3

Problém

Je potrebné nájsť vhodnú kombináciu parametrov učenia pre neurónovú sieť definovanú výpisom 5.1, ktorá je učená na dátovej sade MNIST. Tak, ako v prípade sekvenčného prehľadávania, bola zvolená dvojica optimalizovaných parametrov, ktorými sú v tomto prípade veľkosť dávky a koeficient učenia.

Konfigurácia parametrov učenia

Pevne zvolené parametre učenia sa nachádzajú v tabuľke 6.4. Hodnoty veľkosti dávky boli vygenerované pomocou Haltonovej postupnosti o základe 3, ktorej hodnoty boli transformované na interval $(6; 70)$. Generovanie Hodnoty koeficientu učenia bolo zabezpečené prevedením modifikovanej Haltonovej postupnosti o základe 2 na logaritmickú mierku využitím exponenciálnej funkcie o základe e. Týmto spôsobom boli hodnoty koeficientu učenia upravené na interval $(e^{-12}; e^{-4})$, resp. $(0,00000614421; 0,01831563888)$. Rozsah bol nastavený tak, aby pokrýval hodnotu $0,01$, ktorá bola využitá v predchádzajúcom experimente a zároveň aj výrazne nižšie hodnoty. Zostup gradientu je zabezpečený základným algoritmom SGD, ktorý nemodifikuje veľkosť koeficientu učenia a vďaka tomu umožňuje lepšie zachytiť jeho vplyv na celkovú presnosť klasifikácie v kombinácii s veľkosťou dávky. V sekcii 6.1.2 bolo totiž zistené, že tento algoritmus dosahuje relatívne vysokú presnosť aj pre nižšie hodnoty koeficientu učenia. Chybová funkcia bola nastavená na krížovú entropiu pre viac tried (`CROSS_ENTROPY_MULTICLASS`), ktorá by mala prezentovať lepšiu voľbu pre klasifikačnú úlohu. Učenie bolo obmedzené na 50 epoch.

Zostup gradientu	Chybová funkcia	Epochy
SGD	CROSS_ENTROPY_MULTICLASS	50

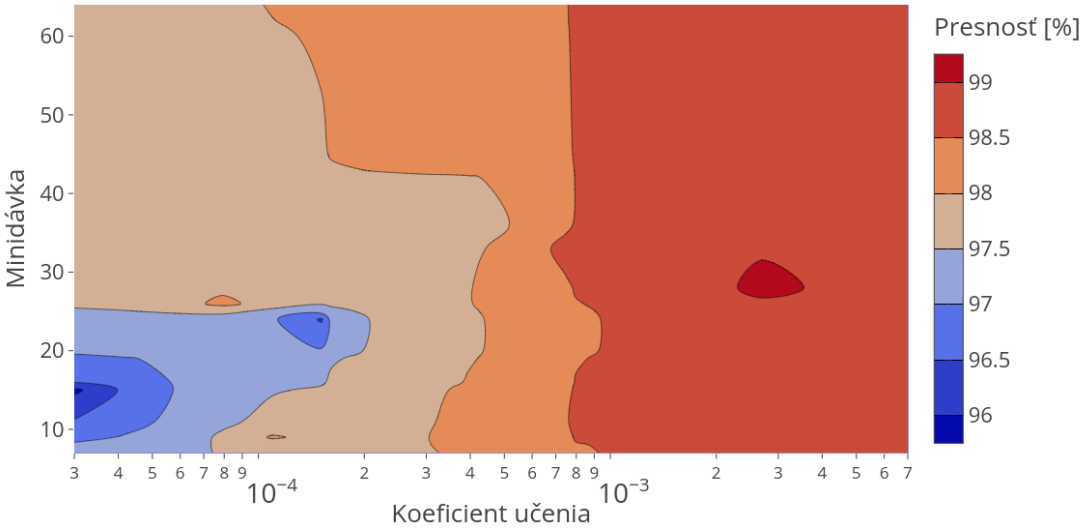
Tabuľka 6.4: Nastavenie parametrov učenia

Výsledok

Najpresnejšiu klasifikáciu s hodnotou 99, 13% dosiahla neurónová sieť, ktorá využila koeficient učenia 0, 002685 a veľkosť dávky 28. Na základe grafu na obrázku 6.5 je možné zhodnotiť, že vyššie hodnoty koeficientu učenia dosahujú vyššiu presnosť. Potvrdzuje to tvrdenie zo sekcie 6.1.2, že pre dosiahnutie vysokej presnosti je potrebné využiť vysokú hodnotu koeficientu učenia, avšak v tomto prípade nebola najvyššia presnosť dosiahnutá najvyšším koeficientom učenia. Ide o koeficient učenia, ktorý bol dostatočne vysoký pre dosiahnutie minima pri určenom počte epoch. Správne zvolená veľkosť dávky však tiež zohrala svoju úlohu.

Problémom, ktorý sa pri učení vyskytol, však je chybová funkcia krížovej entropie. Tá po prekročení svojho minima vykonáva delenie nulou, a preto sa váhy nastavia na hodnotu $-nan$ a zamedzí sa ďalšie učenie. Pri učení tak mala neurónová sieť iba jednu možnosť dosiahnuť minimum chybovej funkcie. Toto chovanie sa potvrdilo pri spustení učenia s optimalizačným algoritmom Adagrad. Z toho dôvodu túto po teoretickej stránke pre tento typ úlohy ideálnu chybovú funkciu neodporúčam využívať v rámci knižnice Tiny-dnn.

Počas učenia nastalo niekoľko prípadov, kedy sa neurónová sieť nezačala učiť. Tieto hodnoty nie sú obsiahnuté v grafe. Išlo o prípady využitia príliš nízkeho koeficientu učenia. Najvyššia hodnota koeficientu učenia, pri ktorej nebolo dosiahnuté učenie, bola 0, 000032 pri veľkosti dávky 55. Pri hodnote 0.000042 a dávke 60 však bola dosiahnutá presnosť 97, 6%. Učenie totiž v tomto prípade začalo až po 8 epochách. Pri nízkych hodnotách koeficientu učenia sa teda výsledky učenia prejavia až po niekoľkých epochách.



Obr. 6.5: Náhodné prehľadávanie

6.1.4 Záver

V tejto sekcii boli optimalizované hyper-parametre neurónovej siete s cieľom zvýšenia jej presnosti riešenia klasifikačnej úlohy. Ručná optimalizácia ilustruje postupnú úpravu modelu neurónovej siete na základe znalosti dátovej sady a naštudovanej literatúry. Najväčší vplyv na zvýšenie presnosti klasifikácie mal prevod aktivačných funkcií skrytých vrstiev na ReLU a výstupnej na softmax. Mierne zlepšenie priniesla aj výmena prvej spájacej vrstvy ma `max_pool`. Posledná optimalizácia, ktorá pridala podľa architektúry LeNet-5 jednu plne-prepojenú vrstvu však nepriniesla očakávané zlepšenie. Táto technika je však nevhodná pre komplexnejšie prehľadávanie priestoru hyper-parametrov.

Sekvenčné prehľadávanie priestoru hyper-parametrov sa zameriavalo najmä na vplyv výšky koeficientu učenia na presnosť klasifikácie pre daný algoritmus zostupu gradientu. Víťazom sa stal algoritmus s adaptívnym koeficientom učenia Adagrad pri využití vysokej hodnoty koeficientu učenia $0,04$. Pre nižšie hodnoty koeficientu učenia však základný algoritmus SGD dosiahol vyššiu presnosť.

Posledná optimalizácia demonštrovala využitie Haltonovej postupnosti na náhodné prehľadávanie priestoru hyper-parametrov. V tomto prípade sa optimalizovala veľkosť dávky a koeficient učenia. Vyššia presnosť bola dosiahnutá pri vyšších hodnotách koeficientu učenia, hoci aj pre nižšie bolo dosiahnuté ich minimum. Počas tohto experimentu bol objavený problém s chybovou funkciou `cross_entropy_multiclass`. Táto funkcia totiž po dosiahnutí minima delila nulou a znemožnila ďalšie učenie.

6.2 Optimalizácia prevodom na celé čísla

Tak ako väčšina knižníc pre implementáciu hlbokých neurónových sietí aj knižnica Tiny-dnn využíva operácie s pohyblivou rádovou čiarkou. Tieto operácie sú však výpočtovo náročnejšie v porovnaní s rovnakými operáciami prevedenými nad celými číslami. Procesor musí navyše pre výpočty s pohyblivou rádovou čiarkou obsahovať matematický koprocesor (skrátene FPU), čo môže v prípade vstavaných zariadení predstavovať zbytočné zvýšenie komplexnosti návrhu a potrebného príkonu. Tieto jednotky sú totiž rozmerom väčšie a obsahujú viac tranzistorov. Ako riešenie tohto problému je pre tieto zariadenia možné využiť už spomínané operácie nad celými číslami. Aby však toto bolo možné vykonať, je potrebné previesť čísla s pohyblivou rádovou čiarkou na celočíselnú reprezentáciu. Dvomi základnými princípmi, ktoré je možné využiť, sú prevod na `číslo s pevnou rádovou čiarkou` alebo `kvantizácia`.

6.2.1 Čísla s pevnou rádovou čiarkou

Číslo s pevnou rádovou čiarkou je na rozdiel od pohyblivej rádovej čiarky, ktorá je reprezentovaná exponentom a mantisou, tvorené N-bitovým slovom, ktorého hodnota závisí čisto na aktuálnej interpretácii. Ak je interpretované ako číslo s pevnou rádovou čiarkou bez znamienka, môže nadobúdať hodnoty podmnožiny P definovanou vzorcom 6.3, kde P môže nadobúdať 2^N hodnôt a p predstavuje N-bitové slovo (celé číslo). Jeho reprezentáciu označujeme $U(a, b)$, kde a predstavuje počet bitov pre celú časť desatinného čísla a b pre desatinnú časť. Pre číslo bez znamienka musí platiť $a = N - b$.

$$P = \{p/2^b | 0 \leq p \leq 2^N - 1, p \in Z\} \tag{6.3}$$

Operácie v hlbokých neurónových sieťach však obsahujú aj negatívne hodnoty. Naj populárnejšou reprezentáciou celých čísel so znamienkom je dvojkový doplnok. Upravený vzorec 6.4 obsahuje reprezentáciu pre číslo s pevnou rádovou čiarkou využívajúci dvojkový doplnok. Pre reprezentáciu takého čísla musí platiť vzťah $a = N - b - 1$ [32, str. 4].

$$P = \{p/2^b | -2^{N-1} \leq p \leq 2^{N-1} - 1, p \in Z\} \tag{6.4}$$

Aritmetické operácie sú vykonávané nad N-bitovým slovom p ako nad obyčajným celým číslom. Výsledok sčítania a odčítania môže požadovať jeden bit navyše, a teda môže dôjsť ku pretečeniu ako pri bežnom celočíselnom type. Skutočný problém však predstavuje násobenie a delenie. Pri násobení získame výsledok s dvojnásobnou dátovou šírkou, teda slovo, ktoré obsahuje $2 \times N$ bitov. Túto hodnotu je následne potrebné posunúť o b bitov doprava, aby sme získali výsledok na N bitoch. Delenie vyžaduje rozšírenie delenca ešte pred prevedením operácie na $2 \times N$ bitov. Táto hodnota je následne posunutá o b bitov doľava a až potom je vykonané delenie pôvodným deliteľom, aby bola zachovaná maximálna presnosť delenia na stanovenom počte bitov.

6.2.2 Kvantizácia

Kvantizácia je podobná prevodu na číslo s pevnou rádovou čiarkou, avšak nachádza sa tu niekoľko rozdielov. Pri využití kvantizácie sa čísla s pohyblivou rádovou čiarkou dynamicky konvertujú na celočíslené hodnoty na základe minimálnej a maximálnej použitej hodnoty. Kvantizácia v knižnici Tiny-dnn je založené na základe algoritmu využitého v knižnici Tensorflow. Čísla s pohyblivou rádovou čiarkou zo vstupu siete sa v ňom prevedú na 8-bitovú reprezentáciu, operácie sa vykonajú nad týmito hodnotami a následne sa na výstupe prevedú naspäť [30]. Avšak knižnica Tiny-dnn nepodporuje prenos kvantizovaných dát medzi jednotlivými vrstvami siete, a tak sa kvantizácia vykonáva na vstupe každej vrstvy a pri výstupe vrstvy sa následne prevedie späť. Na rozdiel od knižnice Tensorflow je možné kvantizácia využiť aj na spätnú propagáciu aj napriek tomu, že v článku [30] sa trénovanie pomocou kvantizovaných hodnôt neodporúča. Osem bitov totiž nemusí stačiť na reprezentáciu spätnej propagácie a zostup gradientu. Jednotlivé aritmetické operácie zvyšujú počet bitov potrebných pre uloženie hodnôt. Ako bolo spomenuté v sekcii 6.2.1, sčítanie a odčítanie požaduje jeden bit naviac a násobenie dvojnásobok bitov. Z toho dôvodu je po týchto operáciách potrebné transformovať dáta späť na pôvodných osem bitov.

6.2.3 Experiment 1

Experiment bol zameraný na učenie neurónovej siete pri využití kvantizačných metód a čísel s pevnou rádovou čiarkou. Kvantizácia je, ako už bolo spomenuté v sekcii 6.2.2, súčasťou knižnice Tiny-dnn. Medzi príkladmi využitia knižnice sa nachádza príklad na neurónovú sieť využívajúcu kvantizáciu na učenie, táto časť knižnice však nie je plne implementovaná a po stiahnutí knižnice z oficiálneho repozitáru nie je tento príklad preložiteľný. Pre tento experiment bola upravená existujúca štruktúra knižnice, v ktorej boli operácie spätnej a doprednej propagácie pre konvolučnú a plne-prepojenú vrstvu nahradené za v knižnici implementované kvantizačné metódy.

Optimalizácia prevodom na čísla s pevnou rádovou čiarkou taktiež využíva vrstvy obsiahnuté v knižnici. Vstupné hodnoty vrstiev sú pred výpočtami prevedené na dátový typ s pevnou rádovou čiarkou `Fixed<T,X>`, ktorý sme sme pre tento experiment implementovali na základe triedy `fixed_point` [24], kde `T` predstavuje dátový typ, v ktorom bude uložená

hodnota a X počet bitov na celú časť desatinného čísla. Operácie sú následne vykonané v tomto dátovom type a na výstupe sú spätne prevedené na čísla s pohyblivou rádovou čiarkou, rovnakým spôsobom ako fungujú aj interné kvantizované vrstvy. Knižnica obsahuje konfiguračný súbor `config.h`, ktorý navádza na použitie dátového typu s pevnou dátovou čiarkou ako interný typ pre všetky výpočty `float_t`. Pokus týmto spôsobom integrovať vlastný dátový typ však bol neúspešný, pretože knižnica na viacerých miestach volá nad týmto dátovým typom štandardné funkcie, ktoré požadujú interný typ jazyka C++. V tomto experimente boli využité dve konfigurácie tohto dátového typu, viz tabuľku 6.5.

Názov	Dátový typ	Dĺžka [bit]	Celá časť [bit]	Desatinná časť [bit]
fixed<int16_t,4>	int16_t	15	4	11
fixed<int32_t,8>	int32_t	31	8	23

Tabuľka 6.5: Konfigurácia dátového typu s pevnou rádovou čiarkou

Pri tomto experimente bola využitá sieť z výpisu 5.1. Aktivačné funkcie `ReLu` však boli v tejto architektúre nahradené za funkcie `tanh`. Pri využití kvantizácie a pevnej rádovej čiarky sa pôvodná neurónová sieť nebola schopná učiť. Parametre učenia boli nastavené podľa tabuľky 6.6. Veľkosť koeficientu učenia musela byť kvôli dátovému typu `Fixed<int16_t,4>` nastavená nižšie ako v doterajších experimentoch, aby bolo zabezpečené učenie neurónovej siete. Maximálny počet epoch bol 280, ale v prípade zastavenia rastu presnosti na testovacích dátach bolo učenie zastavené skôr. Zostup gradientu bol vykonaný Adagrad algoritmom. Chybová funkcia bola nastavená na krížovú entropiu pre dvojicu tried upravenú na viac výstupných tried `cross_entropy` (pozor, implementácia nie je zhodná s `cross_entropy_multiclass`). Pri výbere veľkosti celej a desatinej časti desatinného čísla je potrebné brať do úvahy, aby celá časť bola dostatočne veľká a zabránilo sa pretečeniu dátového typu. Desatinná časť však musí byť dostatočná pre zabezpečenie procesu učenia. Na základe prevedených experimentov bola zvolená varianta s veľkosťou celej časti pre typ int16_t 4 bity. Pre veľkosť 5 bola desatinná časť už príliš malá pre proces učenia.

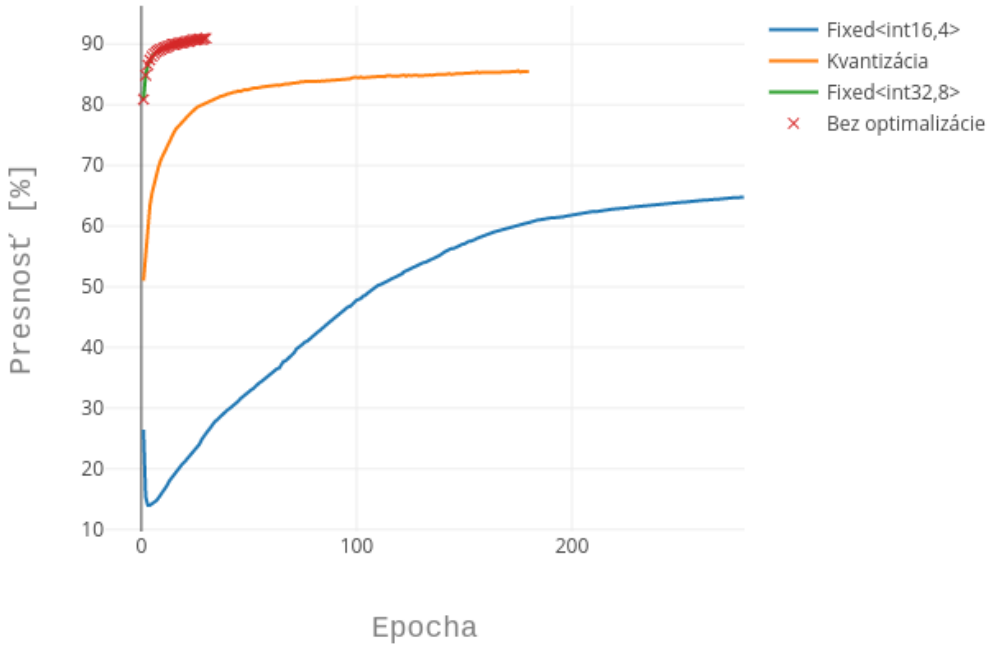
Koeficient učenia	Dávka	Chybová funkcia	Zostup gradientu
0.00004	16	CROSS_ENTROPY	Adagrad

Tabuľka 6.6: Nastavenie parametrov učenia

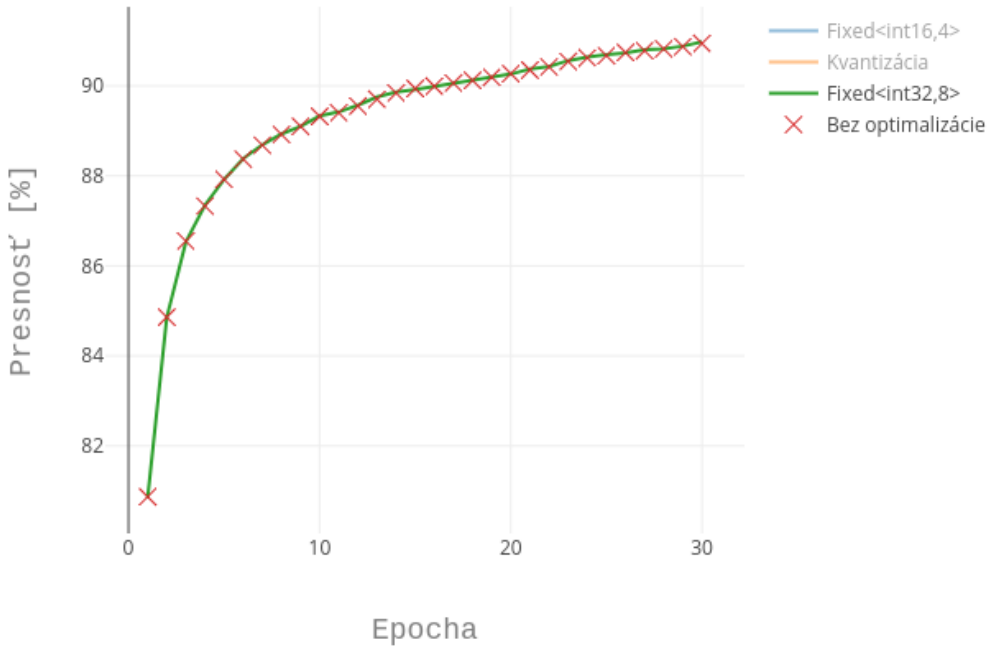
Výsledok

Graf na obrázku 6.6 zobrazuje priebeh učenia pre rôzne optimalizácie. Najvyššia presnosť $90,98\%$ bola po 30 epochách dosiahnutá pomocou optimalizácie využívajúcej dátový typ `Fixed<int32_t,8>`. Presnosť pri tejto optimalizácii kopíruje graf učenia bez využitia optimalizácie, pričom najväčší rozdiel medzi nimi predstavuje $0,05\%$, viz graf na obrázku 6.7. Ide však o 32-bitovú reprezentáciu, teda zaberá rovnaký priestor ako dátový typ `float` využívaný knižnicou Tiny-dnn na bežných architektúrach. Násobenie a delenie navyše vyžaduje, aby procesor podporoval 64-bitový celočíselný typ `int64_t`. Druhú najvyššiu presnosť zaznamenala optimalizácia kvantizáciou ktorá dosiahla presnosť $85,47\%$ po

180 epochách. Najnižšiu úspešnosť 64, 78% zaznamenalo učenie využitím 16-bitového dátového typu `Fixed<int16_t,4>`, toto učenie totiž ani po 280 epochách nenašlo minimum chybovej funkcie.

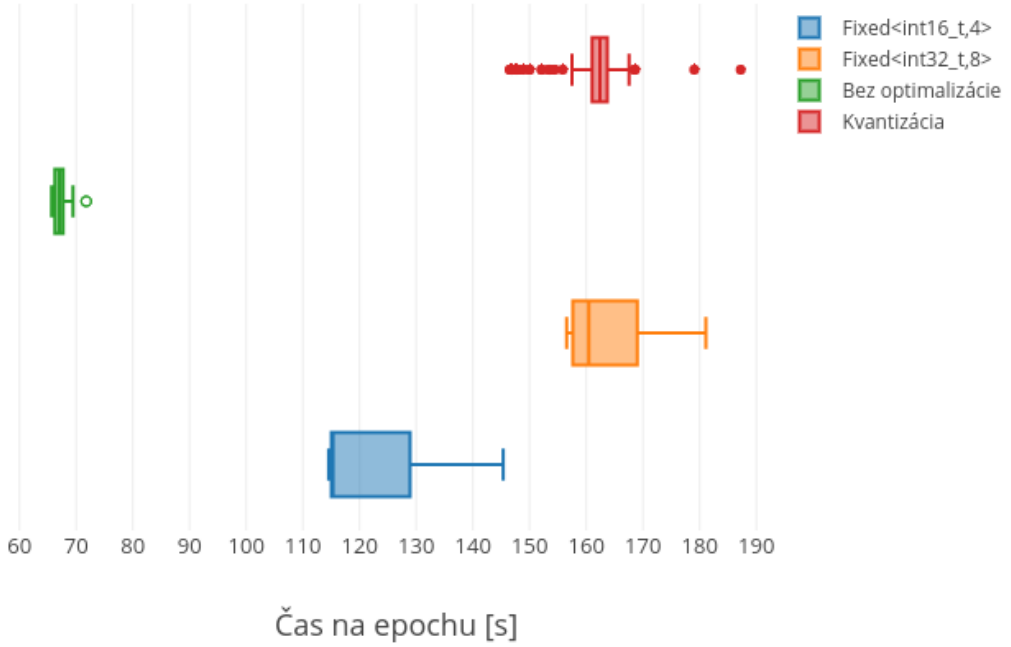


Obr. 6.6: Porovnanie optimalizácií



Obr. 6.7: Porovnanie optimalizácie dátovým typom Fixed<int32_t,8> a učenia bez optimalizácie

Jednotlivé optimalizácie mali vplyv aj na čas potrebný na spracovanie jednej epochy učenia, viz obrázok 6.8. Prevod a následný výpočet v dátovom type `Fixed` dosahoval vyššiu rýchlosť učenia ako integrované kvantizačné metódy. Medián doby učenia na epochu 32-bitovým typom dosiahol hodnotu 160, 45 sekúnd a 16-bitovou verziou 115, 30 sekúnd. Kvantizačné metódy potrebovali na prevedenie jednej epochy učenia 162, 45 sekúnd.



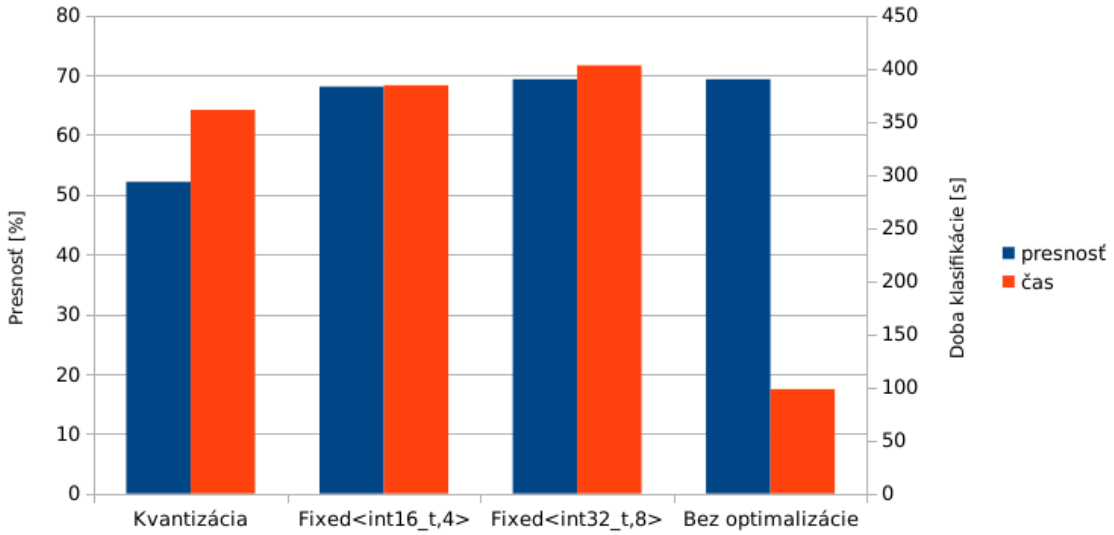
Obr. 6.8: Porovnanie doby učenia

6.2.4 Experiment 2

Optimalizáciu na celočíselný dátový typ však nie je nevyhnutné využívať priamo na učenie. Neurónovú sieť je totiž možné naučiť na dátovej sade využitím výkonného počítača. V reálnej aplikácii sa následne načíta už naučená neurónová sieť a vykonáva sa iba dopredná propagácia. Tento experiment sa preto zaoberá využitím optimalizácií na vopred naučených neurónových sieťach a dátových sadách definovaných v kapitole 5. Klasifikácia týmito neurónovými sieťami bola optimalizovaná pomocou metód využitých v predchádzajúcom experimente, a to kvantizáciou a prevodom na dátový typ `Fixed`.

Cifar-10

Optimalizácia bola vykonaná na neurónovej sieti natrénovanej v sekcii 5.3.4 na dátovej sade CIFAR-10. Klasifikácia podľa využitej siete dosiahla presnosť 69.31% za 98, 292 sekúnd. Rovnaká presnosť bola dosiahnutá aj pri využití dátového typu `Fixed<int32_t,8>`, doba klasifikácie, 402, 963 sekúnd, však bola 4-krát vyššia, viz obrázok 6.9. Dôvodom tejto vysokej doby je najmä konvertovanie dátového typu pred a po prevedení každej operácie. Druhá najvyššia presnosť 68, 08% za 384, 236 sekúnd bola dosiahnutá pomocou dátového typu `Fixed<int16_t,4>`. Najnižšiu presnosť 52, 15% za 361, 086 sekúnd dosiahlo využitie kvantizácie.

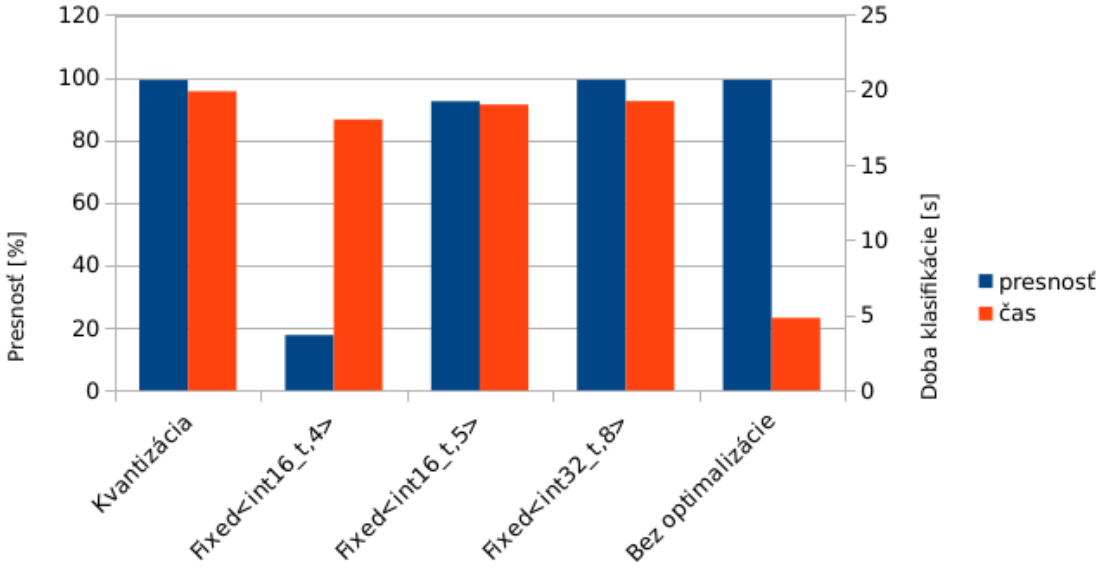


Obr. 6.9: Porovnanie optimalizácií pre CIFAR-10

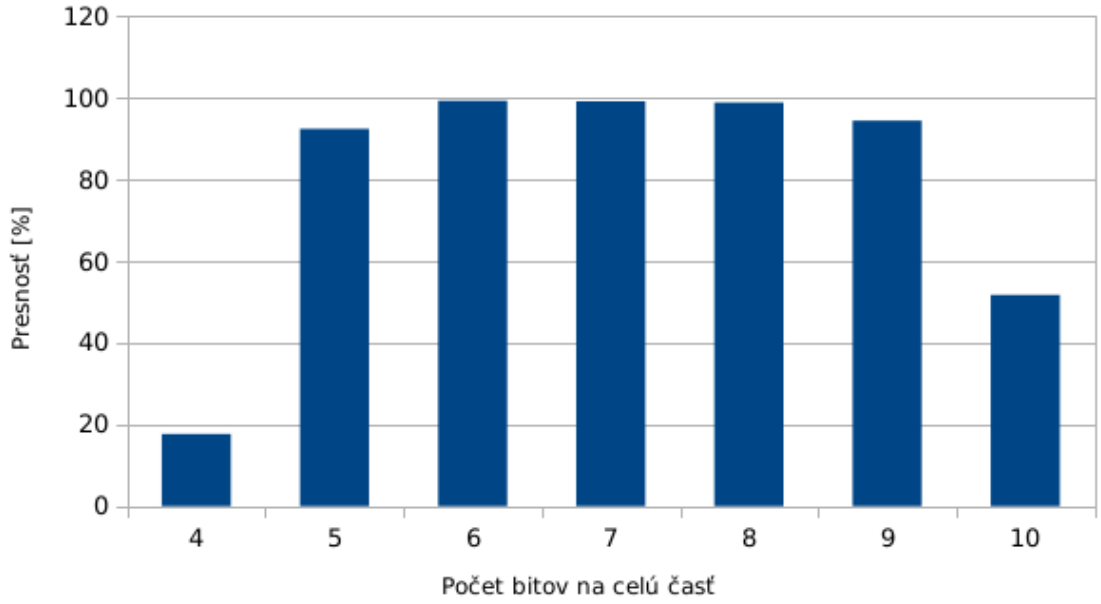
MNIST

Pri tomto experimente bola využitá neurónová sieť zo sekcie 5.3.3 učená na dátovej sade MNIST. Presnosť klasifikácie bez optimalizácií bola $99,22\%$ a trvala $4,824$ sekúnd. Najvyššia presnosť klasifikácie pri využití optimalizácie bola dosiahnutá pri prevode na dátový typ `Fixed<int32_t,8>`, viz obrázok 6.10. Rovnako ako pri dátovej sade CIFAR-10 bola týmto dátovým typom dosiahnutá rovnaká presnosť klasifikácie ako bez využitia optimalizácie. Doba klasifikácie bola v tomto prípade taktiež štvornásobná. Využitím dátového typu `Fixed<int16_t,5>` bola dosiahnutá presnosť $92,4\%$ za $19,031$ sekúnd. Tento dátový typ bol využitý namiesto typu`Fixed<int16_t,4>`, pretože pri jeho použití nebolo možné previesť klasifikáciu. To mohlo byť zapríčinené pretečením celej časti desatinného čísla. Klasifikácia totiž dosiahla presnosť iba $17,66\%$. Klasifikácia pri optimalizovaní kvantizáciou však v tomto prípade dosiahla presnosť $99,16\%$, čo je len o $0,06\%$ menej ako referenčná hodnota bez optimalizácie.

Pri využití dátového typu `Fixed<int16_t,4>` bola zistená možnosť pretečenia celej časti desatinného čísla. Z toho dôvodu bol vykonaný ďalší experiment, počas ktorého sa postupne menil počet bitov určený pre celú časť desatinného čísla v intervale $\langle 4, 10\rangle$. Graf na obrázku 6.11 predstavuje výstup tohto experimentu. Podľa neho bola navyššia presnosť klasifikácie $99,3\%$ dosiahnutá v prípade využitia šiestich bitov na celú časť, čo reprezentuje deväť bitov na desatinnú časť. Táto presnosť bola dokonca o $0,08\%$ vyššia ako v prípade bez využitia optimalizácie. Pri využití piatich a následne štyroch bitov pre celú časť bolo očakávané udržanie alebo mierne zlepšenie presnosti klasifikácie tejto siete. Presnosť však začala dramaticky klesať postupne na 92.4% a $17,66\%$. Je teda možné predpokladať, že hodnoty potrebné na vyhodnotenie tejto siete prevyšujú hodnotu 31 (najvyššiu hodnotu vyčísliteľnú na piatich bitoch). Po vykonaní tohto experimentu na sieti zo sekcie 5.3.4 na intervale $\langle 3, 6\rangle$ bola najvyššia presnosť pri využití veľkosti štyroch bitov.



Obr. 6.10: Porovnanie optimalizácií pre MNIST



Obr. 6.11: Porovnanie konfigurácií typu Fixed<int16_t,X> pre MNIST, kde $X \in \{4, .., 10\}$

6.2.5 Záver

Prevod operácií neurónovej siete s pohyblivou rádovou čiarkou na výpočty s celými číslami je možné previesť dvomi spôsobmi, kvantizáciou a dátovým typom s pevnou rádovou čiarkou. Pri optimalizácii výpočtov už počas učenia došlo pri obidvoch optimalizáciách ku predĺženiu času potrebného na výpočet jednej epochy učenia. Za toto predĺženie však môže najmä časté konvertovanie dátových typov. Metódy využívajúce dátový typ `Fixed` boli mierne rýchlejšie ako kvantizácia. Problém v tomto experimente nastal, keď bolo potrebné rozložiť bity typu

int16_t na celú a desatinnú časť. Kvôli spätnej propagácii je potrebné nastaviť veľký počet bitov na desatinnú časť, čo môže spôsobiť pretekanie celej časti dátového typu. Dátový typ na základe tohto experimentu nie je vhodný na učenie neurónovej siete,

Táto forma optimalizácie však často nie je potrebná a optimalizáciu stačí previesť iba pre doprednú propagáciu. Týmto spôsobom optimalizované siete dosiahli pre dátové sady CIFAR-10 a MNIST podobnú presnosť ako klasifikácia bez optimalizácie. Časová náročnosť je však kvôli pretypovaniu stále vysoká. Správne rozloženie bitov pre dátový typ `Fixed` je nevyhnutné pre získanie požadovaných vlastností klasifikácie. Pri nastavení typu na `Fixed<int16_t,6>` bola dosiahnutá presnosť $99,3\%$, ktorá je dokonca vyššia ako originálna presnosť klasifikácie.

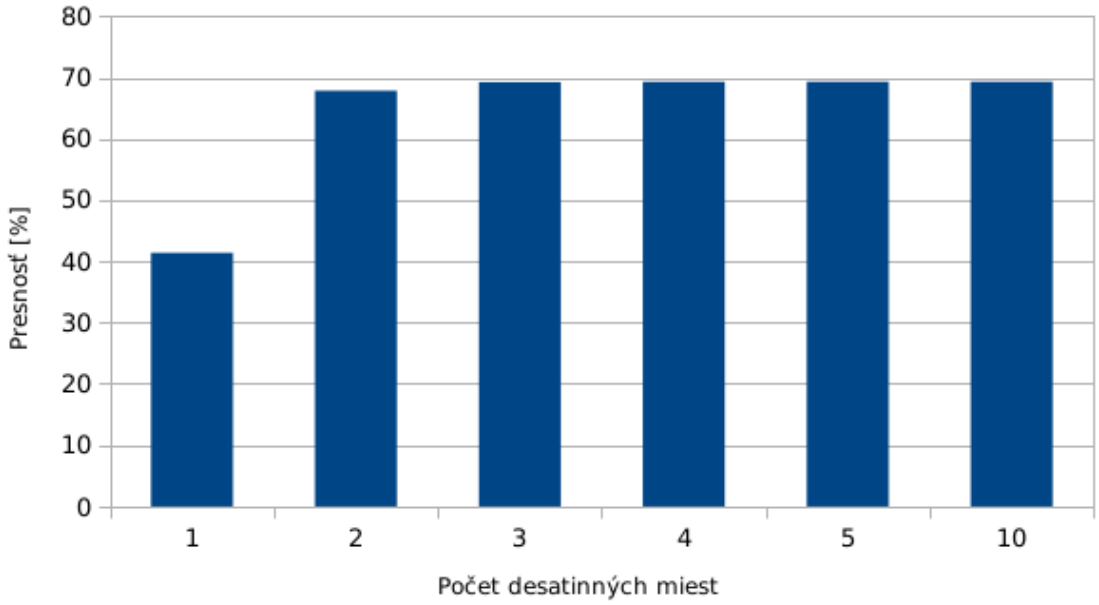
6.3 Optimalizácia zaokrúhlením váh

Okrem výpočtu a presnosti neurónovej siete je možné optimalizovať aj úložný priestor potrebný na pamäťovom médiu pre uloženie váh neurónovej siete. Po naučení neurónovej siete na dátovej sade získame model tejto siete, ktorý obsahuje jej architektúru a hodnoty váh, ktoré je potrebné exportovať pre ďalšie využitie. V knižnici Tiny-dnn je možné tento model exportovať dvomi spôsobmi, a to v binárnej podobe alebo vo formáte `JSON`. Pre túto optimalizáciu bol zvolený formát JSON. Súbor v tomto formáte je desaťkrát väčší, avšak umožňuje jednoduchým skriptom zaokrúhliť váhy uložené ako reťazce vo forme desatinného čísla na intervale $\langle -1, 1\rangle$ na istý počet desatinných miest. Touto optimalizáciou sa sleduje vplyv zníženia presnosti váh na schopnosť neurónovej siete správne klasifikovať.

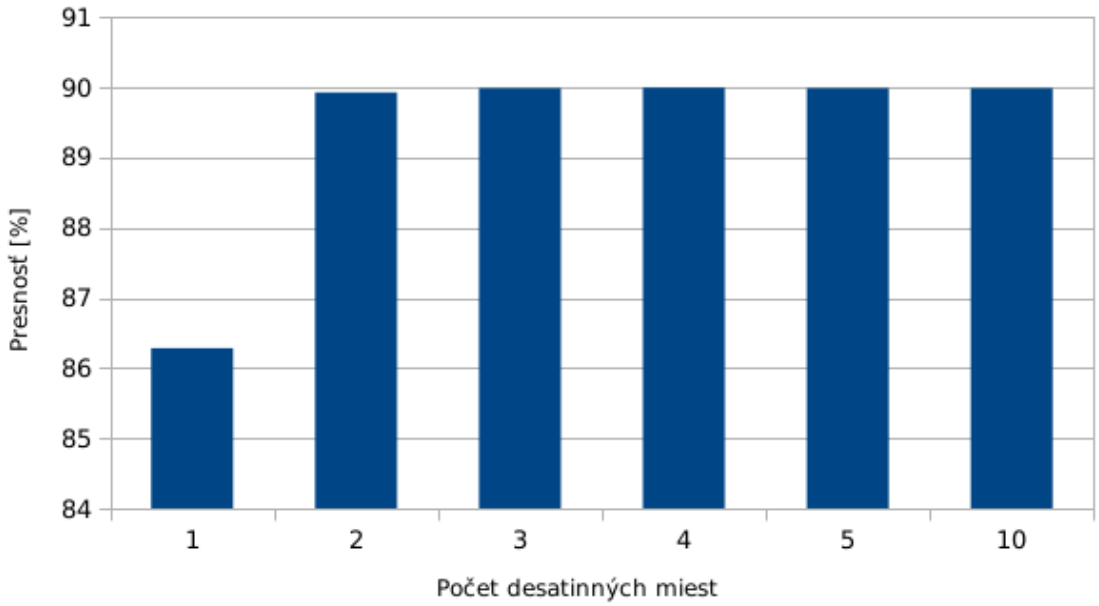
Optimalizácia bola prevedená na neurónovej sieti naučenej v sekcii 5.3.4 na dátovej sade CIFAR-10. Táto neurónová sieť bola vybraná, pretože ide o zložitejšiu klasifikačnú úlohu ako v prípade dátovej sady MNIST a obsahuje viac trénovateľných váh. Chyba spôsobená zaokrúhlením by sa mala výraznejšie propagovať na výsledku. Hodnoty váh boli počas tejto optimalizácie postupne zaokrúhlené na 10, 5, 4, 3, 2 a 1 desatinných miest a následne boli tieto neurónové siete použité na klasifikáciu dátovej sady CIFAR-10. Táto optimalizácia nie je priamo využiteľná pre optimalizáciu neurónovej siete, ale snaží sa ilustrovať vplyv zníženej presnosti reprezentácie uloženej váhy na jej klasifikačnú schopnosť.

6.3.1 Výsledok

Presnosť klasifikácie touto neurónovou sieťou bez úprav bola $69,31\%$ a tú presnosť si zachovala až do zaokrúhlenia na dve desatinné miesta, viz obrázok 6.12. Presnosť po tejto úprave dosiahla hodnotu $67,85\%$, čo stále predstavuje uspokojivý výsledok. Zaokrúhlenie na jedno desatinné miesto však znížilo presnosť klasifikácie na $41,4\%$. Dve desatinné miesta teda stačia na zachovanie presnosti klasifikačnej schopnosti. Pre potvrdenie tejto hypotézy bol následne prevedený experiment na tejto neurónovej sieti, ktorá bola naučená na dátovej sade SVHN, viz obrázok 6.13. Zaokrúhlenie na jedno desatinné miesto spôsobilo zníženie presnosti o $3,7069\%$. Pri zaokrúhlení na dve desatinné miesta sa presnosť klasifikácie zhoršila iba o $0,0614\%$.



Obr. 6.12: Optimalizácia zaokrúhlením hodnôt váh v intervale $\langle -1, 1 \rangle$ pre dátovú sadu CIFAR-10



Obr. 6.13: Optimalizácia zaokrúhlením hodnôt váh v intervale $\langle -1, 1 \rangle$ pre dátovú sadu SVHN

Kapitola 7

Záver

Táto práca sa zaoberala optimalizáciami hlbokých neurónových sietí za účelom zníženia ich zložitosti a zvýšenia presnosti klasifikácie. Optimalizácie boli vykonané na dvoch neurónových sieťach založených na architektúre LeNet-5 využitím knižnice Tiny-dnn. Tieto neurónové siete boli učené na dátových sadách MNIST, CIFAR-10 a SVHN. Účelom prvej optimalizácie bolo nájsť ideálnu kombináciu hyper-parametrov neurónovej siete, a tým optimalizovať presnosť klasifikácie sieťou už počas učenia. V tejto časti boli demonštrované tri rôzne prístupy, a to ručné, sekvenčné a náhodné prehľadávanie priestoru hyper-parametrov s využitím Haltonovej postupnosti. Ďalšou navrhnutou optimalizáciou bolo prevedenie operácií siete využívajúcich čísla s pohyblivou rádovou čiarkou na operácie nad celočíselnými hodnotami. Cieľom tejto optimalizácie bolo odstrániť potrebu využitia matematického koprocesoru, najmä pre využitie na vstavaných zariadeniach. Posledná optimalizácia zaokrúhlením uložených hodnôt váh spojení sledovala koľko desatinných miest je potrebných pre správne fungovanie neurónovej siete, čím je možné ušetriť priestor v prípade serializácie neurónovej siete. Do budúcnosti je naplánované vyskúšať ďalšie optimalizácie hyper-parametrov, ako TPE a Bayesovskú optimalizáciu, a vytvoriť z nich kompletnú knižnicu.

Literatúra

[1] Benenson, R.: *What is the class of this image?* Február 2016, [Online; navštívené 20.04.2018].
URL http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[2] Bergstra, J. S.; Bardenet, R.; Bengio, Y.; aj.: *Algorithms for Hyper-Parameter Optimization*. In *Advances in Neural Information Processing Systems 24*, editácia J. Shawe-Taylor; R. S. Zemel; P. L. Bartlett; F. Pereira; K. Q. Weinberger, Curran Associates, Inc., 2011, s. 2546–2554.

[3] Bhat, C. R.: *Simulation estimation of mixed discrete choice models using randomized and scrambled halton sequences. Transportation Research Part B*, ročník 37, 2002: s. 837–855.

[4] Bottou, L.: *Online Algorithms and Stochastic Approximations*. In *Online Learning and Neural Networks*, editácia D. Saad, Cambridge, UK: Cambridge University Press, 1998.

[5] Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; aj.: *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. CoRR*, ročník abs/1406.1078, 2014.
URL http://arxiv.org/abs/1406.1078

[6] Ciresan, D. C.; Meier, U.; Schmidhuber, J.: *Multi-column Deep Neural Networks for Image Classification. CoRR*, ročník abs/1202.2745, 2012, [Online; navštívené 04.01.2018].
URL http://arxiv.org/abs/1202.2745

[7] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, [Online; navštívené 24.12.2017].
URL http://www.deeplearningbook.org

[8] Gurney, K.: *An introduction to neural networks*. UCL Press, 1997, ISBN 1-85728-503-4.

[9] Haykin, Simon S.: *Neural Networks and Learning Machines*. Pearson Education, 2009, ISBN 978-0-13-147139-9.

[10] Helix84: *Schéma typického neurónu*. 2007, [Online; navštívené 05.04.2018].
URL https://commons.wikimedia.org/wiki/File:Neuron_slk.svg

[11] Hochreiter, S.; Schmidhuber, J.: *Long Short-Term Memory. Neural Comput.*, ročník 9, č. 8, November 1997: s. 1735–1780, ISSN 0899-7667, doi:10.1162/neco.1997.9.8.1735.

[12] J Duchi, E. H.; Singer, Y.: *Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research*, ročník 12, 2011.

[13] Jia, Y.; Shelhamer, E.; Donahue, J.; aj.: *Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093*, 2014.

[14] Khiyari, H. E.; Wechsler, H.: *Face Recognition across Time Lapse Using Convolutional Neural Networks. Journal of Information Security*, ročník Vol.07No.03, 2016: str. 11, doi:10.4236/jis.2016.73010.

[15] Kingma, D. P.; Ba, J.: *Adam: A Method for Stochastic Optimization. CoRR*, ročník abs/1412.6980, 2014.

[16] Krizhevsky, A.: *Learning multiple layers of features from tiny images.* Technická správa, Department of Computer Science, University of Toronto, 2009.

[17] Lecun, Y.; Bottou, L.; Bengio, Y.; aj.: *Gradient-based learning applied to document recognition.* In *Proceedings of the IEEE*, 1998, s. 2278–2324.

[18] LeCun, Y.; Cortes, C.: *MNIST handwritten digit database.* 2010.
URL http://yann.lecun.com/exdb/mnist/

[19] Lee, C.-Y.; Gallagher, P. W.; Tu, Z.: *Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree.* In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, *Proceedings of Machine Learning Research*, ročník 51, editácia A. Gretton; C. C. Robert, Cadiz, Spain: PMLR, Máj 2016, s. 464–472.

[20] Netzer, Y.; Wang, T.; Coates, A.; aj.: *Reading Digits in Natural Images with Unsupervised Feature Learning.* In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[21] Nomi, T.: *tiny-dnn Documentation.* 2017, [Online; navštívené 03.01.2018].
URL https://media.readthedocs.org/pdf/tiny-dnn/latest/tiny-dnn.pdf

[22] Parker, S.: *Ľudské telo.* Ikar, 2008, ISBN 978-80-551-1731-7.

[23] Rojas, R.: *Neural Networks.* Springer-Verlag, 1996, ISBN 978-3-642-61068-4.

[24] Schregle, P.: *Fixed Point Class.* November 2009, [Online; navštívené 26.4.2018].
URL https://www.codeproject.com/Articles/37636/Fixed-Point-Class

[25] Sermanet, P.; Chintala, S.; LeCun, Y.: *Convolutional neural networks applied to house numbers digit classification.* In *ICPR*, IEEE Computer Society, 2012, ISBN 978-1-4673-2216-4, s. 3288–3291.

[26] Shevchuk, Y.: *Hyperparameter optimization for Neural Networks.* December 2016, [Online; navštívené 20.04.2018].
URL http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html

[27] Simonyan, K.; Zisserman, A.: *Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR*, ročník abs/1409.1556, 2014, 1409.1556.
URL http://arxiv.org/abs/1409.1556

[28] Tieleman, T.; Hinton, G.: *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning*. 2012, [Online; navštívené 09.04.2018].

[29] Waibel, A.; Hanazawa, T.; Hinton, G.; aj.: *Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech, and Signal Processing*, ročník 37, č. 3, Marec 1989: s. 328–339, ISSN 0096-3518, doi:10.1109/29.21701.

[30] Warden, P.: *How to Quantize Neural Networks with TensorFlow*. Máj 2016, [Online; navštívené 26.4.2018].
URL https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/

[31] Widrow, B.; Hoff, M. E.: *Adaptive switching circuits*. Technická správa, Stanford University Stanford Electronics Labs, 1960.

[32] Yates, R.: *Fixed-Point Arithmetic: An Introduction*. Technická správa, Digital Singal Labs, 2007, [Online; navštívené 26.4.2018].
URL https://courses.cs.washington.edu/courses/cse467/08au/labs/l5/fp.pdf

[33] Yegnanarayana, B.: *Artificial Neural Networks*. Asoke K. Ghosh, 1999, ISBN ISBN-81-203-1253-8.

[34] Zeinali, Y.; Story, B.: *Competitive probabilistic neural network*. ročník 24, 01 2017: s. 1–14.

[35] Zurada, J. M.: *Introduction to Artificial Neural Systems*. West Publishing Company, 1992, ISBN 0-314-93391-3.