VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ZJEDNODUŠENÉ NÁSOBENÍ V KONVOLUČNÍCH NEURONOVÝCH SÍTÍCH

SIMPLIFIED MULTIPLICATION IN CONVOLUTIONAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PAVEL JUHAŇÁK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Juhaňák Pavel, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Zjednodušené násobení v konvolučních neuronových sítích**
Simplified Multiplication in Convolutional Neural Networks
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s problematikou neuronových sítí a konvolučních neuronových sítí.
2. Seznamte se s knihovnami pro práci s konvolučními neuronovými sítěmi.
3. Na zvolených testovacích úlohách ověřte funkčnost vybrané knihovny.
4. Pro zvolené části neuronové sítě navrhněte nahrazení standardní operace násobení jednodušším výpočtem.
5. Vyhodnoťte dopad upravené operace násobení na přesnost výstupu a implementační cenu neuronové sítě.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 26. října 2018

Abstrakt

Tato práce se zaměřuje na problematiku klasických i konvolučních neuronových sítí. Jsou zde probrány konvenční techniky hardwarového násobení a možnosti jeho optimalizace v kontextu umělých neuronových sítí. Je navržena metoda zjednodušeného násobení, a to násobení bez násobičky. Tato metoda je implementována a integrována do knihovny TypeCNN. Poté je proveden odhad ceny hardwarového řešení jak konvenčního, tak zjednodušeného násobení. Představeny jsou dostupné nástroje pro práci s konvolučními neuronovými sítěmi a datové sady pro jejich testování v úloze klasifikace obrazů. Jsou navrženy testovací architektury a metodika testování a experimentů. Následně jsou zhodnoceny výsledky testů jak z pohledu úspěšnosti klasifikace, tak ceny hardwarového řešení.

Abstract

This thesis provides an introduction to classical and convolutional neural networks. It describes how hardware multiplication is conventionally performed and optimized. A simplified multiplication method is proposed, namely multiplierless multiplication. This method is implemented and integrated into the TypeCNN library. The cost of the hardware solution of both conventional and simplified multipliers is estimated. The thesis also introduces software tools developed to work with convolutional neural networks and datasets used to test them in the image classification task. Test architectures and experimentation methodology are proposed. The results are evaluated, and both the classification accuracy and cost of the hardware solution are discussed.

Klíčová slova

umělá inteligence, soft computing, neuron, neuronové sítě, konvoluční neuronové sítě, násobení, optimalizace, zjednodušené násobení, násobení bez násobičky

Keywords

artificial intelligence, soft computing, neuron, neural networks, convolutional neural networks, multiplication, optimization, simplified multiplication, multiplierless multiplication

Citace

JUHAŇÁK, Pavel. *Zjednodušené násobení v konvolučních neuronových sítích.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Zjednodušené násobení v konvolučních neuronových sítích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Lukáše Sekaniny, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Juhaňák
25. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu práce prof. Ing. Lukáši Sekaninovi, Ph.D., za vedení, odbornou pomoc a cenné rady při vytváření této práce. Také bych chtěl poděkovat své rodinně za podporu a motivaci nejen při vytváření této práce, ale také v osobním životě.

Obsah

Kapitola 1

Úvod

Neuronové sítě jako takové spadají do výpočetního paradigmatu tzv. soft computingu. Všechny algoritmy či výpočetní modely spadající do této oblasti nalézají inspiraci v přírodě a jejich společným rysem je, že se do jisté míry dokáží vypořádat s nepřesnostmi a chybami v reálném světě. Stejně tak tomu je u neuronových sítí, které k těmto vlastnostem přidávají schopnost naučit se různé funkce na základě ukázkových vstupů a výstupů. Díky této vlastnosti se mohou naučit uspokojivě řešit problémy, pro něž neznáme algoritmické řešení, nebo je konvenční metoda řešení velmi výpočetně náročná.

Konvoluční neuronové sítě jsou podtřídou klasických neuronových sítí a slouží především k rozpoznávání a klasifikaci obrazu. Tyto sítě jsou dnes čím dál oblíbenější a nacházejí uplatnění v oblasti samočinných robotů či aut, v bezpečnostních kamerách, pro rozpoznávání poznávacích značek či různých předmětů a v mnoha dalších oblastech. Jejich většímu rozšíření, zejména v oblasti nízkopříkonových vestavěných systémů, však brání dlouhá doba nutná k jejich učení, především jejich velikost na čipu a spotřeba elektrické energie. Právě proto je potřeba hledat cesty, jak optimalizovat tyto parametry. Optimalizace je vykoupena ztrátou přesnosti, kterou se snažíme minimalizovat. Jednou z cest je i hledání možností, jak nahradit konvenční operaci násobení.

Tato práce se tedy zaměřuje na problematiku klasických i konvolučních neuronových sítí. Objasní jejich inspiraci, vznik a další jejich vlastnosti. Nahlédneme na konvenční postupy provádění operace násobení a možnosti jejich optimalizace. Dále se zaměříme na dostupné knihovny pro práci s neuronovými sítěmi. Provedeme návrh a implementaci vybrané techniky zjednodušeného násobení a do jedné z knihoven ji integrujeme. Na vybraných úlohách a architekturách realizujeme experimenty jak s konvenčním, tak se zjednodušeným násobením. Provedeme odhad ceny zjednodušeného násobení při implementaci v hardwaru. Cílem práce je nalézt vhodný kompromis mezi přesností neuronové sítě a cenou násobení.

Kapitola 2 se věnuje klasickým neuronovým sítím, jejich inspiraci v biologickém neuronu, historii a formálnímu modelu neuronu. Také jsou zde popsány architektury neuronových sítí a jejich aktivační funkce. V následující kapitole 3 jsou představeny konvoluční neuronové sítě, jejich vlastnosti a typy jejich vrstev. V kapitole 4 je nastíněna problematika násobení v neuronových sítích. Nejprve jsou zde probrány možnosti zobrazení čísel v počítači a konvenční techniky jejich násobení. Následně jsou zkoumány techniky optimalizace operace násobení. V kapitole 5 jsou představeny dostupné nástroje pro práci s neuronovými sítěmi. V následující části práce, popsané v kapitole 6, je proveden návrh operace zjednodušeného násobení (násobení bez násobičky) a popsána její implementace a integrace do knihovny TypeCNN. V další kapitole 7 jsou popsány datové sady pro testování neuronových sítí a architektury, které budou k testování použity. Následně je navržena metodika testování

a provádění experimentů. Dále jsou představeny a diskutovány jejich výsledky. V kapitole 8 je proveden odhad ceny hardwarové implementace jak konvenčního, tak zjednodušeného násobení a jejich porovnání. Kapitola 9 se věnuje konečnému zhodnocení experimentů a jejich ceny při hardwarové implementaci. Je diskutována ztráta přesnosti a její výměna za zrychlení výpočtu a ušetření hardwarových zdrojů. V kapitole 10 shrnuji tuto práci a nastiňuji moznosti dalšího rozšíření.

Kapitola 2

Neuronové sítě

Oblast neuronových sítí řadíme k výpočetnímu paradigmatu zvanému soft computing. V následující kapitole představím rozdíly mezi tímto a tradičním výpočetním paradigmatem. Dále bude probrána struktura a funkce jak biologického, tak umělého neuronu. Detailněji se budeme věnovat funkci umělého neuronu a jeho začleňování do neuronových sítí. Představeny budou také často používané aktivační funkce. V této kapitole čerpám převážně z následujících zdrojů [8] [13] [15] [23] [34] [36].

2.1 Cesta k soft computing

Tradiční výpočetní postupy

Tradičně se nejvíce využívá výpočetní paradigma založené na důsledné algoritmizaci metody řešení. Pro řešení problému tímto způsobem je třeba problému detailně porozumět (analyzovat ho) a navrhnout analytické řešení. Analytické řešení následně algoritmizujeme, čímž umožníme řešení tohoto problému na počítači.

Pro nalezení analytického řešení je potřeba skutečná odborná znalost řešeného problému. Například pro modelování fyzikálních procesů je třeba jim opravdu porozumět. Hledání analytického řešení je také velmi závislé na kvalitě vstupních dat. Pokud bychom například řešili problém lineární regrese a naše data byla zašuměná, problém vůbec nemusíme vyřešit, nebo můžeme nalézt nesprávné řešení. Při nalezení správného analytického řešení je nám odměnou, že následné výsledky jsou přesné a spolehlivé.

Počítač při výpočtu především s reálnými čísly může zanést do výsledku chybu, např. zaokrouhlením (chyba vzniklá způsobem uložení a počítání s čísly v počítači), která se především při rekurzivních a iteračních algoritmech může nepříjemně zvětšovat. Dalším problémem je, že strojový výpočet těchto řešení může být velmi výpočetně náročný. Příkladem je numerické řešení velkých soustav parciálních diferenciálních rovnic.

Ne vždy je však možné nalézt přesné analytické řešení, a proto se často uchylujeme k optimalizaci a spokojíme se s přibližným řešením, což do výsledků zanáší další nepřesnosti. Často je však problém tak složitý, že jeho analytické řešení neexistuje, nebo by jeho nalezení, popřípadě nalezení přibližného řešení, bylo tak časově náročné, že to pro danou úlohu nemá smysl. Možná neexistence přesného řešení je dalším problémem tohoto paradigmatu. Ovšem pokud problém není příliš složitý a máme k dispozici solidní vstupní data, je analytický přístup k řešení problému efektivní. V opačném případě je vhodné zvolit alternativní přístup k řešení problému a tím může být právě soft computing.

Soft computing

Soft computing jako odvětví informatiky má několik různých směrů, mezi něž patří především neuro, evoluční, rojové a fuzzy výpočty. Všechny tyto přístupy spojuje inspirace v přírodě. Neuropočítání je inspirováno nervovou soustavou živých organismů, stavbou a funkcí jednotlivých neuronů a jejich propojením pro řešení problému. Evoluční počítání je inspirováno biologickou evolucí, kdy evoluční proces může být chápán jako řešení optimalizačního problému. Rojové výpočty jsou inspirovány především chováním kolonií hmyzu, jako jsou roje včel a kolonie mravenců, které řeší problém nalezení potravy.

Společným rysem metod inspirovaných v přírodě je jejich určitá schopnost tolerovat chyby a nepřesnosti reálného světa. V reálném světe nejsou organismy nikdy postaveny před naprosto stejný problém, a přesto ho dokáží vyřešit. Tyto metody využívají nepřesnost vstupních dat k dosažení větší robustnosti řešení a jeho lepšího vztahu s realitou. Výpočet pomocí těchto metod tedy nekončí nalezením teoretického optima, ale nalezením dostatečně dobrého řešení, jehož další zpřesňování by na výsledek mělo tak malý vliv, že toto zpřesňování již nemá smysl.

Při řešení problému paradigmatem soft computingu tedy neprogramujeme přímo řešení daného problému, ale programujeme zvolenou, přírodou inspirovanou, metodu, jejímž pomocí problém řešíme. Pro řešení problému tedy není nutná jeho hluboká znalost, nicméně jistá znalost je třeba pro zvolení adekvátní metody užité pro řešení. S hlubší znalostí problému je také možné lépe nastavit různé parametry zvolené metody a tak například dostat řešení v kratším čase.

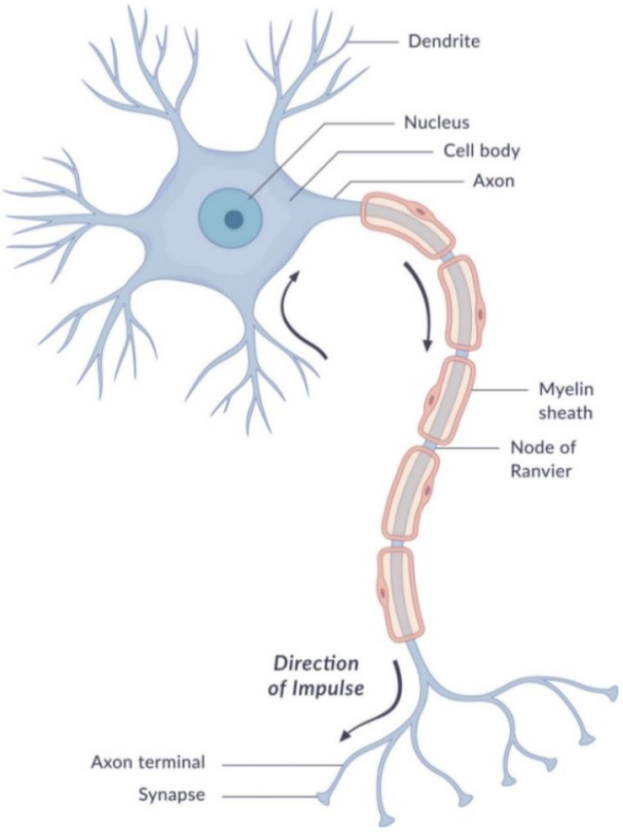
Pokud se zaměříme konkrétně na neuronové sítě, tak kromě výše zmíněných vlastností, které jsou společné pro všechny přírodou inspirované algoritmy, neuronové sítě jsou programovatelné za pomocí vstupních dat a k nim příslušných dat výstupních. To znamená, že na rozdíl od tradičních metod, nebo některých jiných soft computing metod, u nichž pro konkrétní problém musíme implementovat konkrétní řešení, jedna neuronová síť může řešit více různých problémů. Toho je dosaženo učením neuronové sítě na příslušných vstupních a výstupních datech. Tedy to, jaký problém bude daná neuronová síť řešit, závisí na tom, jaký problém ji řešit naučíme.

Pro porovnání přístupu tradičních metod a soft computingu uveďme problém řešený bankami, zda udělit klientovi půjčku. Při tradičním přístupu by bylo potřeba najít analytické řešení a matematické vztahy mezi např. platem klienta za určité období, jeho věkem, počtem dětí, dosaženým vzdělání, pracovním zaměřením, atd. a následně provést algoritmizaci a řešení. Těchto vztahů by bylo nesmírné množství a nalézt je by bylo velmi složité, nebo nemožné a následný výpočet by byl příliš výpočetně náročný. Naopak při přístupu soft computingu za použití např. neuronových sítí bychom síť naučili tento problém řešit na základě zkušeností s minulými klienty. Na vstup sítě bychom vložili zvolená data a učili ji, zda tento klient půjčku bezproblémově splácel, či nikoliv. Neuronová síť by se tak naučila asociace, které by bylo velice těžké, nebo nemožné nalézt analyticky.

2.2 Biologický neuron

Neuronové sítě jsou inspirované nervovou soustavou živých organizmů, která se skládá z jednotlivých, navzájem propojených neuronů. Neuron je tedy základním stavebním prvkem nervové soustavy. Neuron je specializovaná buňka určená k přijímání, zpracování, uchování a předávání informace. Neuron se skládá z těla neuronu, dendritů a axonu (schéma neuronu můžeme vidět na obrázku 2.1). Každý neuron v lidském mozku má 10^2 až 10^5 dendritů, což

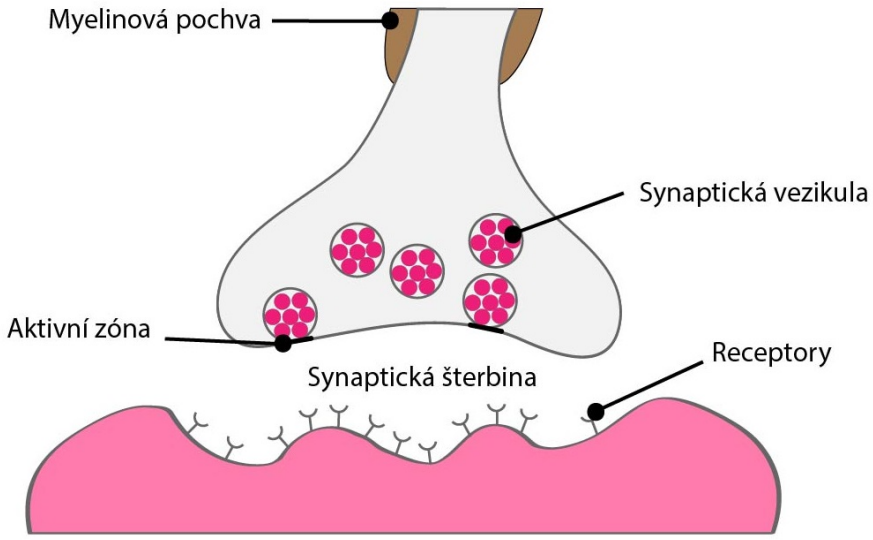
jsou krátké 2-3 mm dlouhé výběžky z těla neuronu sloužící jako jeho vstupy. Jeden axon, což je jeden až několik cm dlouhý výběžek z těla neuronu, který je na svém konci rozvětven, slouží jako výstup neuronu.



Obrázek 2.1: Schéma biologického neuronu [24]

Neuron je se svým okolím spojen pomocí axonu a dendritů prostřednictvím tzv. synapse. Ta je tvořena jedním z rozvětvení axonu jednoho neuronu a jedním z dendritů druhého neuronu. Lidský mozek tvoří přibližně 10^{11} neuronů, jež jsou mezi sebou provázány přibližně 10^{14} synapsemi. Synapsi můžeme chápat také jako rozhraní pro přenos informace mezi dvěma neurony. Informace je mezi těmito neurony přenášena chemickým procesem, který nazveme vzruch, při němž axon sekretuje neurotransmiter, který je registrován příslušnými receptory nacházejícími se na dendritech (schéma synapse můžeme vidět na obrázku 2.2). Každá synapse má specifickou vlastnost, kterou nazýváme synaptická vazba, nebo také váha. Váha synapse určuje, jak je neuronem registrována informace přicházející na konkrétní dendrit v porovnání s informací, kterou vyslal axon druhého neuronu. Pokud má synaptická vazba excitační charakter, pak je neuronem registrován silnější vzruch, než byl skutečně vyslán. Naopak, pokud má synaptická vazba inhibiční charakter, je vzruch částečně potlačen a následně je registrován jako slabší. V současné době se domníváme, že právě synaptická vazba je nositelem informace [34].

Funkce neuronu samotného je pak registrovat vzruchy na svých dendritech, a pokud suma těchto vzruchů přesáhne určitý práh, začne neuron periodicky vysílat vzruch svým axonem. Tento vzruch je pak dále registrován dalšími neurony. Neuron vzruch periodicky

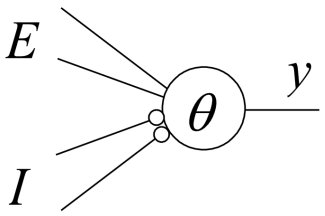


Obrázek 2.2: Schéma synapse biologických neuronů [20]

vysílá tak dlouho, dokud se suma vzruchů na jeho dendritech nedostane pod práh. Síla vysílaného vzruchu, práh a váha synapse se mění učením. V průběhu života je u živých organismů také měněno vzájemné propojení neuronů. Při učení vznikají synapse nové a procesem zapomínání některé synapse zanikají.

2.3 Historie

Mezi prvními, kdo přišel s myšlenkou inspirovat se neurony živých organismů a napodobit je pro provádění výpočtů, byli pánové Warren McCulloch a Walter Pitts. Tito v roce 1943 jako první předvedli model umělého neuronu. Jejich model umělého neuronu měl explicitně určeno, které vstupy jsou excitační a které inhibiční, dalším parametrem tohoto neuronu byl jeho aktivační práh. Výpočet probíhal tak, že byla provedena suma excitačních a suma inhibičních vstupů a na základě těchto hodnot a aktivačního prahu byla pomocí tabulky určena výstupní hodnota. Schéma tohoto neuronu a aktivační tabulku můžeme vidět na obrázku 2.3. Tento neuron byl schopen realizovat základní logické funkce AND, OR a NOT.



ΣE	ΣI	y
-	> 0	0
$< \theta$	$= 0$	0
$\geq \theta$	$= 0$	1

Obrázek 2.3: Schéma prvního umělého neuronu a tabulka jeho aktivační funkce [36]

Dalšího zásadnějšího pokroku v této oblasti dosáhl Frank Rosenblatt, který roku 1957 předvedl zobecněný model neuronu, který nazval perceptron. Perceptron počítá s reálnými čísly a již nemá separované excitační a inhibiční vstupy, ale každou vstupní hodnotu násobí hodnotou váhy pro tento vstup. Do jeho skokové aktivační funkce tedy vstupuje suma vektoru vstupů vynásobeného vektorem vah. Učení perceptronu znamená hledání vhodného váhového vektoru pro daný problém. Rosenblatt navrhl učící algoritmus, který v konečném čase nalezne odpovídající váhový vektor.

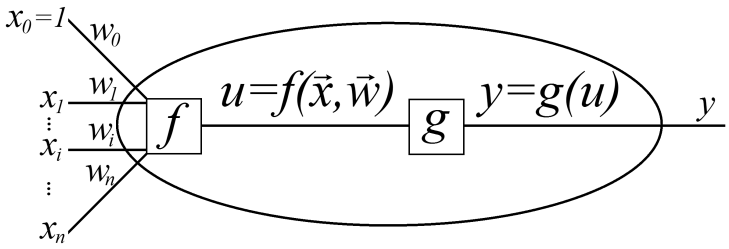
Následoval postupný rozvoj těchto jednovrstvých sítí. Zásadním problémem těchto jednovrstvých sítí však je jejich schopnost řešit problémy s lineárně separovatelnými vstupními vektory. Roku 1969 ve své knize Perceptrons na tento problém upozorňují pánové Marvin Minsky a Seymour Paper. Ukazují zde, že sítě tohoto typu nedokáží řešit ani tak zdánlivě jednoduchý problém, jako je logická funkce XOR. Tento problém sice lze řešit pomocí vícevrstvé sítě, konkrétně pomocí dvouvrstvé sítě s dvěma perceptrony ve vstupní a jedním ve výstupní vrstvě, ale v této době neexistoval algoritmus, který by zvládl takovouto síť učit. Především kvůli této knize a zdání neřešitelnosti tohoto problému vývoj oblasti neuronových sítí po dobu více než deseti let stagnoval.

Významnou událostí, která opět zvedla zájem o oblast neuronových sítí, byla publikace učícího algoritmu zpětného šíření chyby nazvaného Backpropagation. Tento algoritmus roku 1986 ve své práci publikovali pánové David Rumelhart, Geoffrey Hinton a Ronald Williams. Sami však tento algoritmus nevymysleli, ale navázali na práci Arthura Brysona a Yu-Chi Ho, kteří o tomto algoritmu hovořili již v roce 1969. Nicméně v důsledku úpadku zájmu o neuronové sítě se algoritmus neuchytil a uznání se mu dostalo až díky výše zmíněným pánům. Algoritmus je pomocí zpětného šíření chyby schopný efektivně trénovat vícevrstvé neuronové sítě. Díky tomuto algoritmu se tedy podařilo vyřešit XOR problém, který Minsky označil jako neřešitelný a kvůli kterému stagnovalo celé toto odvětví. Díky tomuto algoritmu se také zvedl zájem o neuronové sítě, jejich výzkum a využití v různých oblastech.

2.4 Umělý neuron

Neuronové sítě i samotný umělý neuron za dobu své poměrně dlouhé existence prošel vývojem, avšak idea inspirace biologickým neuronem zůstala stejná. Proto je současný formální model neuronu podobný tomu úplně prvnímu a od modelu perceptronu představeného již v roce 1957 se téměř neliší. Současný obecný model neuronu můžeme vidět na obrázku 2.4. V současném obecném modelu neuronu jsou všechny hodnoty, se kterými neuron pracuje, reálná čísla. Vidíme zde, x_1 až x_n vstupních hodnot, kterými modelujeme dendrity – vstupy biologického neuronu. Ke každému vstupu přísluší jedna váha, tedy máme zde w_1 až w_n vah, odpovídajících synaptickým vahám. Díky použití reálných čísel není třeba vstupy/váhy rozdělovat na inhibiční a excitační. Váhy se zápornou hodnotou, popřípadě váhy z intervalu $(0; 1)$, můžeme chápat jako inhibiční. Umělý neuron může mít stejně jako biologický neuron nastavený určitý vnitřní práh, avšak v současné době tento vnitřní práh nahrazujeme (modelujeme) další vstupní hodnotou $x_0 = 1$ a jí příslušnou váhou w_0. Tento práh umělého neuronu nazýváme bias. Tento přístup k modelování prahu má výhodu v tom, že hodnota prahu, tedy hodnota váhy w_0, se mění během učení spolu s ostatními vahami a není proto nutné předem analyzovat a volit hodnotu vnitřní váhy neuronu. Bias společně se vstupy tvoří vstupní vektor $\vec{x} = (x_0, \ldots, x_n)$ a váha biasu společně s ostatními vahami tvoří vektor vah $\vec{w} = (w_0, \ldots, w_n)$.

Vstupní vektor $\vec{x}$ a váhový vektor $\vec{w}$ jsou vstupy funkce f, kterou nazýváme bázová funkce. Bázová funkce má za úkol z těchto vstupních vektorů vytvořit jednu hodnotu u,



Obrázek 2.4: Schéma obecného modelu neuronu [36]

takzvaný vnitřní potenciál, podobně jako se tomu děje při sčítání vzruchů na dendritech v biologickém neuronu. V současné době můžeme vybírat ze dvou bázových funkcí a to ze standardně používané lineární bázové funkce (vztah 2.1) a zřídka používané radiální bázové funkce (vztah 2.2). Zde x_i značí i-tý vstup, w_i váhu i-tého vstupu, x_0 společně s w_0 jsou biasem a n je počet vstupů.

$$u = f(\vec{x}, \vec{w}) = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

$$u = \sqrt{\sum_{i=0}^{n} (w_i - x_i)} \tag{2.2}$$

Vnitřní potenciál neuronu u je vstupem funkce g, kterou nazýváme aktivační funkce. Aktivační funkce určuje výstupní hodnotu celého neuronu y. Touto funkcí modelujeme proces, kdy se biologický neuron na základě prahu a všech jeho vstupů rozhodne, zda a jak silný vzruch vyšle svým axonem. Aktivačních funkcí existuje výrazně větší množství než bázových, a proto se jim podrobněji budeme věnovat v kapitole 2.6. Obecně můžeme celý proces výpočtu neuronu zapsat vztahem:

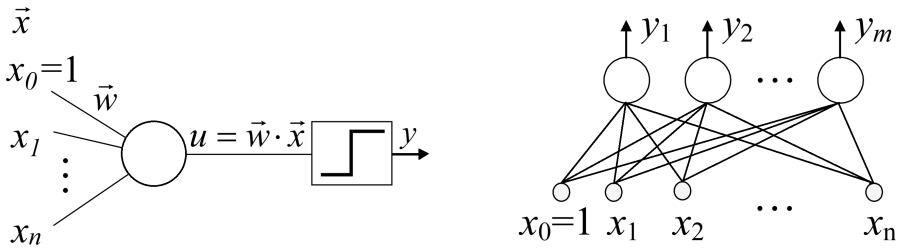
$$y = g(u) = g\Big(f(\vec{x}, \vec{w})\Big) \tag{2.3}$$

Perceptron

Perceptron je považován za základní autonomně funkční model neuronu. Tento model je ve skutečnosti téměř totožný s obecným modelem neuronu uvedeným výše. Rozdíl je pouze v tom, že má pevně danou bázovou i aktivační funkci. Perceptron používá lineární bázovou funkci a skokovou aktivační funkci danou následujícím předpisem:

$$y = \begin{cases} 1 & \text{pro } u > 0 \\ -1 & \text{pro } u < 0 \\ y^{old} & \text{pro } u = 0 \end{cases} \tag{2.4}$$

Název perceptron je také používán pro nejjednodušší jednovrstvou (o vrstvách dále v sekci 2.5) neuronovou síť složenou z 1 až n perceptronů. Schéma jednoho perceptronu a sítě perceptron můžeme vidět na obrázku 2.5.



Obrázek 2.5: Schéma jednoho perceptronu a sítě perceptron [36]

Takováto síť je typicky složena pouze z jednoho perceptronu, který má 1 až n vstupů a jeden výstup. Taková to síť dokáže klasifikovat do dvou tříd. Přidáváním neuronů do jediné vrstvy zvyšujeme počet rozpoznatelných tříd N a to podle následujícího vztahu:

$$N = \begin{cases} \sum_{i=0}^{n} \binom{m}{i} & \text{pro } m > n \\ 2^m & \text{pro } m \leq n \end{cases} \tag{2.5}$$

Kde n je počet vstupů a m počet výstupů (počet neuronů). I když má síť více neuronů, neurony v jedné vrstvě se navzájem neovlivňují a proto se učí samostatně. Algoritmus učení perceptronu tedy funguje následovně. Nejdříve je v nultém kroku inicializován váhový vektor $\vec{w_0}$ a to na náhodná malá čísla, např. $-0.5 < w_i < 0.5$. Následně jsou na vstupy neuronů přiloženy vstupní hodnoty. Pro jednotlivé neurony je spočítán vnitřní potenciál u (vztah 2.1), který dále vstupuje do aktivační funkce (vztah 2.4) a pro každý z neuronů je spočítána jeho aktuální výstupní hodnota y. Aktualizace váhového vektoru je pak prováděna řešením následující rovnice:

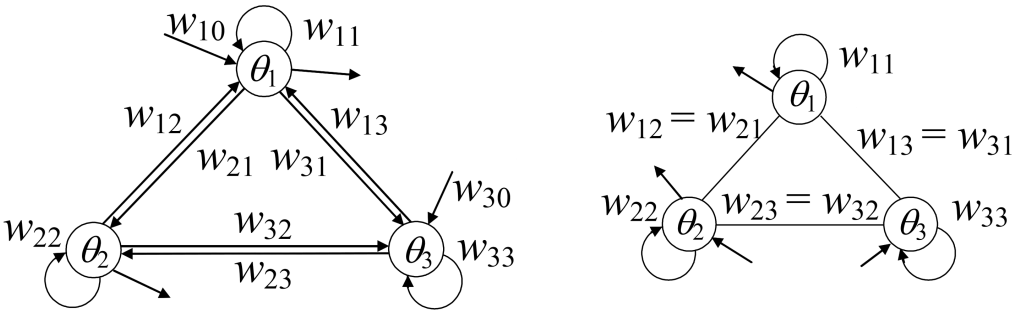
$$\vec{w_k} = \vec{w_{k-1}} + \mu * (d_k - y_k) * \vec{x_k}, \tag{2.6}$$

kde $k = 1, 2, ...$ je krok učení, $\vec{w_k}$ aktualizovaný váhový vektor, $\vec{w_{k-1}}$ současný váhový vektor, d_k požadovaná výstupní hodnota neuronu, y_k aktuální výstupní hodnota neuronu, $\vec{x_k}$ vstupní vektor a μ je takzvaný učící koeficient, který je volen jako $0 < \mu < 1$. Tato rovnice je řešena zvlášť pro každý z neuronů, čímž je provedena aktualizace jim příslušných váhových vektorů. Algoritmus učení je prováděn tak dlouho, dokud dochází ke změnám ve vektorech vah, popřípadě je překročen maximální počet kroků učení.

2.5 Architektura

Tato kapitola představuje architektury umělých neuronových sítí sestavených ze základních kamenů – neuronů. Obecně neuronovou síť modelujeme jako vážený orientovaný graf. Uzly tohoto grafu jsou neurony samotné a ohodnocením uzlů vyjadřujeme práh neuronu. Orientované hrany mezi uzly značíme propojení dvou neuronů, přičemž výstupní hodnota neuronu (uzlu) z něhož hrana vychází je brána na vstup neuronu (uzlu) do něhož hrana vstupuje. Ohodnocením hran vyjadřujeme hodnotu váhy příslušného vstupu neuronu. Hodnoty vah hran jsou obecně různé, to však nevylučuje, že některé z nich mohou nabývat stejných hodnot.

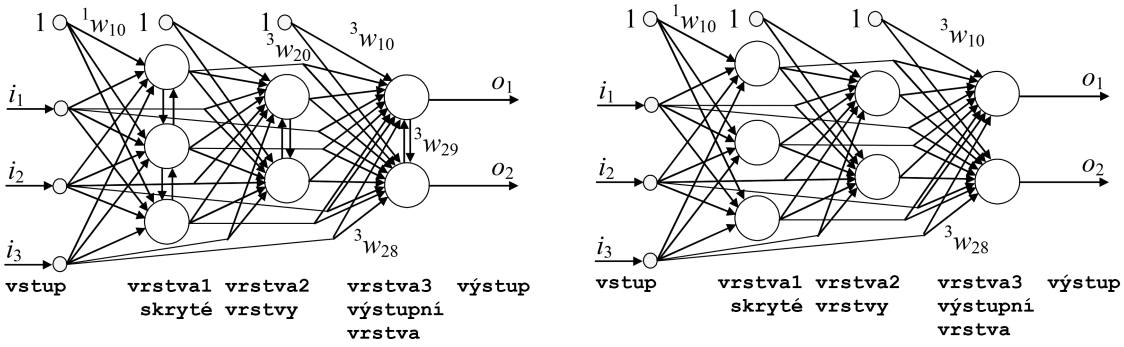
Nejobecnější architekturou neuronové sítě je takzvaná plně propojená síť. Tato síť odpovídá výše uvedenému obecnému popisu a nejsou na ni kladeny žádné další požadavky. V této síti tedy mohou být propojeny libovolné dva neurony a to včetně propojení uzlu sama na sebe s libovolným ohodnocením hran. Podobnou architekturou je plně propojená symetrická síť, jež přidává podmínku symetričnosti pro ohodnocení hran, tedy pokud mezi neurony A a B vedou hrany $A \to B$ a $B \to A$, musí se jejich váhy rovnat $w_{ab} = w_{ba}$. Neurony těchto sítí mohou být vstupní i výstupní zároveň, což je nevýhodné pro výpočet odezvy, či jejich učení. Ukázku sítí těchto architektur můžeme vidět na obrázku 2.6. Sítě, v nichž některý z neuronů může být ovlivněn svým výstupem, neboli graf této sítě obsahuje cyklus, nazýváme jako sítě rekurentní. Tyto sítě jsou nejvíce podobné svým biologickým předlohám, avšak výpočet jejich odezvy či jejich učení je časově náročné a pro sítě větších rozměrů prakticky nemožné. Další nevýhodou těchto sítí je, že je nutné předem určit prahy neuronů.



Obrázek 2.6: Schéma plně propojené a plně propojené symetrické sítě [36]

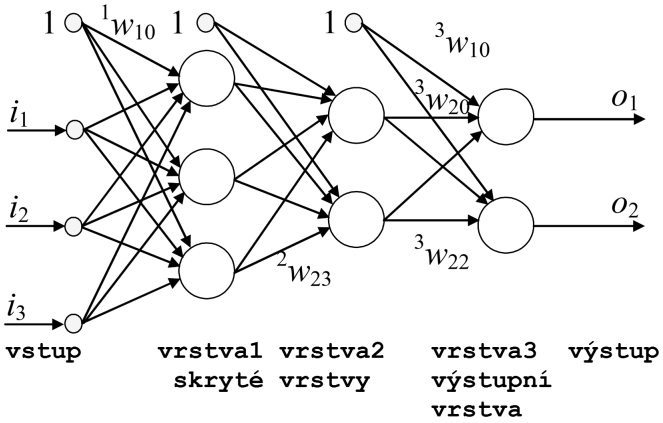
Problémy s komplexností předešlých architektur pro učení a výpočtu odezvy sítě z části řešíme explicitním určením vstupních a výstupních neuronů. Dále zavádíme takzvané vrstvy neuronů. Pro neurony v jedné vrstvě musí platit, že nijak neovlivňují neurony z vrstev předchozích. Síť splňující tyto požadavky nazýváme vrstvová síť. Tato síť však stále zůstává rekurentní, protože se neurony v jedné vrstvě mohou ovlivňovat navzájem a je tedy možné aby neuron ovlivnil sám sebe. Při zobrazování vrstvové sítě často modelujeme vstupní vektor jako první (vstupní) vrstvu sítě, složenou z formálních neuronů, jejichž vstupní i výstupní hodnoty jsou totožné. Dalšími formálními neurony, či formálními vstupy, modelujeme (zápornou) hodnotu prahu neuronu, takzvaný bias. A to tak, že do každé vrstvy neuronů kromě vrstvy výstupní, přidáme nultý formální neuron, který propojíme se všemi neurony vrstvy následující. Tento formální neuron má na vstupu i výstupu hodnotu 1 a bias skutečného neuronu je tak určen hodnotou váhy připadající ke vstupu z neuronu formálního. Pokud ze sítě vrstvové architektury odstraníme vazby mezi neurony stejné vrstvy, získáme síť acyklickou. Graf odpovídající takovéto síti již nemůže obsahovat žádný cyklus, a proto je výpočet odezvy této sítě, či její trénování, výrazně rychlejší než u sítí rekurentních. Sítě těchto architektur můžeme vidět na obrázku 2.7. Zde symbolem w_{ij}^{l} značíme váhu j-tého vstupu i-tého neuronu ve vrstvě l. Poslední vrstvu neuronů, která již není propojena s dalšími neurony, nazýváme vrstva výstupní. Vrstvy nacházející se mezi vstupní (formální) a výstupní vrstvou nazýváme skryté vrstvy.

Acyklickou síť, u níž povolíme vytváření vazeb pouze mezi neurony sousedních vrstev, nazýváme dopředná síť. Výsledkem je, že jednotlivé vrstvy komunikují pouze s předchozí a následující vrstvou, čímž je umožněno například zřetězené zpracování po vrstvách a to



Obrázek 2.7: Schéma vrstvové a acyklické sítě [36]

především v hardwarové implementaci. U těchto sítí (stejně jako u acyklických) je také možné provádět paralelní zpracování neuronů ve stejné vrstvě. Největším přínosem této architektury je však umožnění postupného učení po vrstvách a to především za pomoci algoritmu backpropagation. Praktickou výhodou je také omezení počtu vah v síti a předem známý maximální počet vah pro neuron, který je roven počtu neuronů v předchozí vrstvě.



Obrázek 2.8: Schéma plně propojené dopředné sítě [36]

Za dnes prakticky nejpoužívanější architekturou neuronové sítě můžeme označit plně propojenou dopřednou síť, jejíž schéma můžeme vidět na obrázku 2.8. Jak napovídá název, jedná se o dopřednou síť, v níž je každý neuron propojen se všemi neurony následující vrstvy. Tato síť obsahuje alespoň dvě vrstvy a to vrstvu výstupní a formální vstupní vrstvu, tedy síť může obsahovat 0 až N skrytých vrstev. Počet vstupních a výstupních neuronů je určen úlohou, neboli pro daný problém se jejich počet nemění. Počet skrytých vrstev a počty neuronů v jednotlivých skrytých vrstvách určuje programátor a pro jeden úkol je možné volit různé jejich kombinace. Přesnost odezvy sítě závisí na této volbě, bohužel však v současné době neexistuje algoritmus či návod, jak určit optimální počet skrytých vrstev či počtu neuronů v nich. Volba tedy závisí na zkušenostech programátora, popřípadě je možné použít některý z optimalizačních algoritmů, např. genetický algoritmus. Obecná poučka říká, že první skrytá vrstva by měla obsahovat více neuronů než vrstva vstupní

a počty neuronů v dalších skrytých vrstvách by se měly snižovat směrem k počtu neuronů ve výstupní vrstvě, nicméně tento přístup jistě není optimální a jeho výsledek není zaručen. Pokud je počet skrytých vrstev vysoký[1], nazýváme takovou síť hlubokou.

2.6 Aktivační funkce

Jak bylo zmíněno výše, na rozdíl od bázových funkcí, které známe dvě, prakticky je téměř výhradně používána lineární bázová funkce (vztah 2.1), aktivačních funkcí existuje poměrně více a má tedy smysl si některé z nich představit. Následující aktivační funkce jsou tedy používány společně s lineární bázovou funkcí. Aktivační funkce transformuje vnitřní potenciál neuronu na výstupní hodnotu a v závislosti na úloze, či na pozici vrstvy v rámci vrstvového modelu neuronové sítě, může být výhodné volit různé aktivační funkce. Obecně je výhodou, pokud je aktivační funkce diferencovatelná, neboť to umožňuje její použití společně s učícím algoritmem backpropagation, který na aktivační funkci klade právě tuto podmínku.

Skoková aktivační funkce

Nejstarší a dnes již nepoužívanou je skoková aktivační funkce (vztah 2.4), která byla použita v rámci modelu perceptron. Problémem této funkce je v jejím malému (binárnímu) oboru hodnot, ale především tato funkce není diferencovatelná.

Aktivační funkce „Ramp“

Tato po částech lineární, nebo také saturovaná lineární funkce je dána předpisem 2.7. Má poněkud zajímavější průběh než skoková aktivační funkce a řeší její nevýhodu v binárním oboru hodnot. Avšak i tato funkce má problém s diferencovatelností.

$$g(x) = \begin{cases} 1 & \text{pro } x > 1 \\ 0 & \text{pro } x < 0 \\ x & \text{jinak} \end{cases} \tag{2.7}$$

Usměrněná lineární funkce

Tato funkce známější pod zkratkou ReLU (Rectified linear unit) je dána vztahem:

$$g(x) = max(0, x) \tag{2.8}$$

ReLU je v současnosti velice oblíbená a často používaná především v rámci konvolučních vrstev konvolučních neuronových sítí. Výhodou této funkce je její jednoduchost a tedy vyšší rychlost výpočtu oproti např. hyperbolickému tangentu. Další výhodou je také rychlost konvergence neuronové sítě, která ji používá. Práce [18] ukazuje, že v porovnání s hyperbolickým tangentem ReLU konverguje až 6x rychleji. Nevýhodou ReLU je, že v průběhu učení může dojít k nastavení takové váhy, že daný neuron již nikdy nebude aktivován. Tento problém nastává především v případě zvolení vysokého koeficientu učení. Vyvarovat se mu tedy lze zvolením vhodně malého koeficientu učení, nebo zvolením aktivační funkce Leaky ReLU, která tento problém částečně řeší, nebo alespoň zmenšuje jeho dopad.

[1] Není určena přesná hranice, ale uvažujme více než větší jednotky.

Sigmoida

Sigmoida je první používanou nelineární aktivační funkcí, která je používaná dodnes. Nejčastější používaný tvar sigmoidy vrací výstup v intervalu $(0; 1)$, její střední hodnota leží v bodě $(0; 0.5)$ a je dána rovnicí 2.9. Výhodou sigmoidy je tedy její nelinearita, omezený obor hodnot a především existující derivace.

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

Hyperbolický tangens

Tato nelineární aktivační funkce je svými vlastnostmi velmi podobná sigmoidě, až na to, že v nejčastěji používaném tvaru, jehož předpis je dán rovnicí 2.10, leží její obor hodnot v intervalu $(-1; 1)$ a její střední hodnota v bodě $(0, 0)$. Právě proto ve většině klasických úloh konverguje rychleji [8] a je tedy oblíbenější než sigmoida, také je vhodnější pro použití ve vícevrstvých sítích. Tanh je často používán v klasických dopředných sítích a také v plně propojených vrstvách konvolučních neuronových sítí.

$$g(x) = tanh(x) \tag{2.10}$$

SoftMax

SoftMax je aktivační funkcí, která je často používána u výstupní vrstvy jak klasických, tak konvolučních neuronových sítí řešících úlohu klasifikace. Tato funkce určuje pravděpodobnost, s jakou vstupní vzorek patří do každé z rozpoznávaných tříd, tedy součet pravděpodobností všech tříd je 1. Předpis funkce SoftMax je dán rovnicí:

$$g(x_i) = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}}, \tag{2.11}$$

kde K značí počet rozpoznávaných tříd (počet neuronů výstupní vrstvy). Vidíme zde, že pro výpočet výstupní hodnoty jednoho neuronu (pravděpodobnosti této třídy) funkce počítá s vnitřními potenciály všech neuronů výstupní vrstvy. Jako výsledek klasifikace je tedy označena třída s nejvyšší pravděpodobností.

Kapitola 3

Konvoluční neuronové sítě

Konvoluční neuronové sítě řadíme mezi dopředné vrstvové neuronové sítě. Na rozdíl od klasických neuronových sítí skládajících se pouze z plně propojených vrstev, konvoluční sítě se skládající ze čtyř hlavních typů vrstev (viz kapitola 3.1). Konvoluční sítě jsou v současné době čím dál oblíbenější a nacházejí uplatnění v oblasti zpracování přirozeného jazyka, ale především v oblasti rozpoznávání a klasifikace obrázků. Dvěma hlavními motivacemi pro vznik těchto sítí jsou snížení paměťových a časových nároků výpočtu, a upřednostnění lokální před globální informací spojené se snahou o nezávislost klasifikace na deformacích či posunutí vstupních vzorů oproti vzorům trénovacím. V této kapitole čerpám převážně z následujících zdrojů [14] [30] [36].

Uvažujme například černobílý vstupní obrázek z datové sady MNIST (obr. 7.1) o velikosti 28×28 pixelů, tedy vstupní vrstva by obsahovala 784 neuronů. Pokud bychom použili plně propojenou skrytou vrstvu o například 400 neuronech, pak by se pro tuto vrstvu muselo uchovat a také naučit přes 300 000 vah. Pokud bychom však místo plně propojené použili vrstvu konvoluční, s jedním filtrem (bude objasněno dále) o velikosti 8×8 pixelů, pak bychom v této skryté vrstvě měli také 400 neuronů, ale potřebovali bychom uchovat a naučit pouze 65 vah a to proto, že váhy filtru jsou sdílené mezi neurony konvoluční vrstvy. Přestože se v praxi používají nízké desítky filtrů, počet vah je oproti plně propojené vrstvě stále řádově menší.

Vstupem klasické neuronové sítě je vektor, tedy 1D struktura, ovšem obrázky jsou 2D (stupně šedi), popřípadě 3D (barevné) struktury a pro použití v takové síti je nejprve převádíme do 1D a to typicky čtením po řádcích. Pokud takovouto síť naučíme rozpoznávat vycentrované obrázky a následně na vstup přiložíme stejný obrázek, avšak posunutý, síť ho nemusí rozpoznat, protože po převedení tohoto obrázku do 1D si jeho vektor nemusí být podobný s vektorem příslušné předlohy. Řešením je nedívat se na obrázek jako na jeden vektor, avšak zaměřit se na jednotlivé pixely a jejich okolí, tedy upřednostníme lokální informaci před globální. Na jednotlivé pixely 2D/3D obrázku se tedy budeme dívat pomocí takzvaných filtrů, jejichž úkolem je detekovat určité rysy (například svislou čáru) v okolí pixelu. V klasickém přístupu ke klasifikaci obrázků filtry navrhují vývojáři, v konvolučních neuronových sítích se tyto filtry vytváří automaticky a to učením sítě. Takovýto filtr, na rozdíl od plně propojené vrstvy, detekuje daný rys stejně dobře, ať se ve vstupním obrázku nachází kdekoliv.

3.1 Typy vrstev

Konvoluční vrstva

Jak napovídá již název, tak tato vrstva na vstupním obrázku provádí operaci konvoluce. Diskrétní operace konvoluce používaná v oblasti neuronových sítí je dána vzorcem:

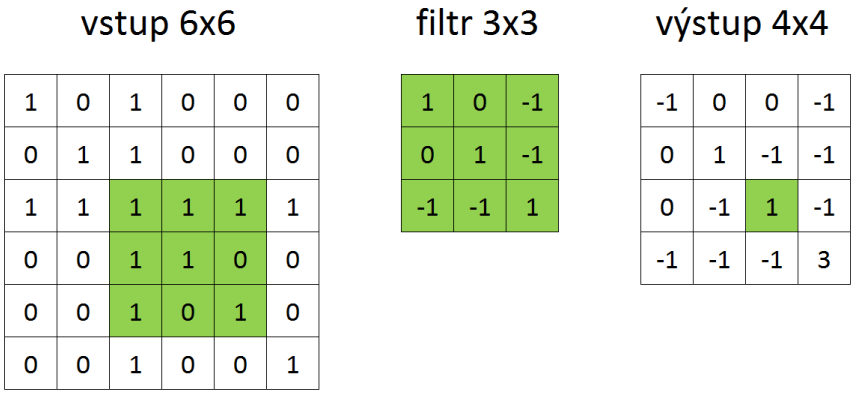
$$C(i,j) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} I(i+u, j+v)F(u,v), \tag{3.1}$$

kde funkce I vrací hodnotu vstupního pixelu a funkce F hodnotu filtru na příslušných souřadnicích. Operace konvoluce tedy provádí vážený součet hodnoty pixelu na souřadnicích $(i; j)$ a hodnot pixelů v jeho okolí s hodnotami filtru (též nazývaným maskou, nebo jádrem konvoluce), které nazýváme váhami. Výsledkem konvoluce je matice takzvaných příznaků, někdy zvaných rysů (features).

Parametry konvoluční vrstvy jsou rozměry filtru (šířka a výška), počet filtrů, počet okrajových nul (padding) a krok posunu filtru (stride). Při praktickém použití je často používán čtvercový filtr, tedy $a = b$ a rozměry filtru jsou pak dány vzorcem:

$$\check{s}i\check{r}ka = v\acute{y}\check{s}ka = 2a + 1 \tag{3.2}$$

Tímto docílíme, že zkoumaný pixel bude ležet ve středu filtru. Počtem filtrů určujeme dimenzionalitu výstupu konvoluční vrstvy, jelikož každý z filtrů vytvoří jednu výstupní matici. Rozšíření obrázku o okrajové nuly používáme, pokud požadujeme, aby rozměry výstupních matic byly totožné s rozměry vstupního obrázku. Naopak zvolením kroku většího než jedna docílíme zmenšení rozměru výstupů. V praxi se však většinou používá krok o velikosti jedna, protože k zmenšení rozměrů matic slouží vrstva poolingová.

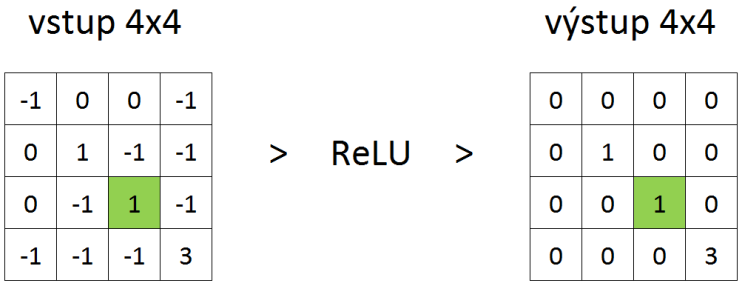


Obrázek 3.1: Ukázka operace konvoluce

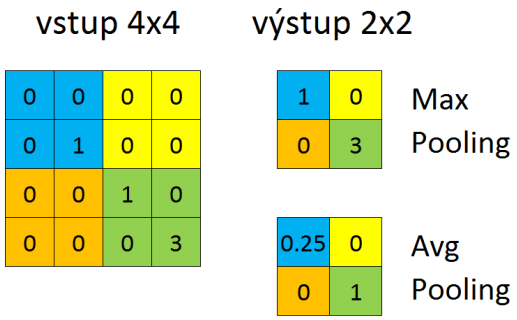
Vstupní obrázek je tedy nejprve rozšířen o okrajové nuly. Následně je k jeho hornímu levému rohu přiložen filtr, spočítán výsledek konvoluce a ten je uložen do výsledné matice. Poté je filtr posunut o zvolený počet pixelů a opět počítána konvoluce. To je prováděno do zpracování celého vstupního obrázku tímto filtrem a následně opakováno pro filtry další. Na obrázku 3.1 vidíme ukázku operace konvoluce pro vstupní matici (černobílý obrázek) o velikosti 6×6 bez okrajových nul, s filtrem o velikosti 3×3 prováděnou s krokem jedna.

Aktivační vrstva

Jak bylo řečeno výše, výstupem konvoluční vrstvy jsou matice příznaků. Jejich hodnoty bychom ovšem s trochou nadsázky mohli nazvat jakýmsi vnitřním potenciálem pomyslného neuronu, protože jsou získány podobně, jako hodnoty vnitřního potenciálu neuronu za pomocí lineární bázové funkce. Za konvoluční vrstvou tedy typicky následuje vrstva aktivační, která za pomocí aktivační funkce převede matice příznaků na výstupní matice, tedy převede hodnotu vnitřního potenciálu na výstupní hodnotu stejně jako u jediného neuronu. Na obrázku 3.2 vidíme transformaci matice příznaků na výstupní matici pomocí oblíbené aktivační funkce ReLU (kapitola 2.6). Parametrem aktivační vrstvy je pouze požadovaná aktivační funkce, popřípadě parametry této aktivační funkce.



Obrázek 3.2: Ukázka aktivační vrstvy s funkcí ReLU



Obrázek 3.3: Ukázka Max a AVG poolingu

Poolingová vrstva

Hlavním účelem poolingové, nebo též seskupující či podvzorkovací vrstvy je redukce velikosti matic (dimenzionalitu zachovává nezměněnou). Parametry této vrstvy jsou velikost masky (výška a šířka) a krok posunutí masky. Tuto masku posunujeme po vstupní matici podobně jako filtr u konvoluční vrstvy s tím rozdílem, že neprovádíme operaci konvoluce, ale nad hodnotami aktuálně zvolenými maskou provádíme funkci maximum (Max), nebo průměr (Avg). V praxi se nejčastěji používá Max pooling s maskou o velikosti 2×2 s krokem 2, která redukuje velikost matic na polovinu. Masky by se při posouvání po vstupu neměly překrývat, tedy velikost kroku by měla být větší, nebo rovna velikosti masky. Použití funkce maximum při poolingu přímo zvyšuje odolnost sítě proti posunutí vstupů [12]. Na obrázku 3.3 vidíme ukázku Max a Avg poolingu s maskou o velikosti 2×2 a krokem 2.

Plně propojená vrstva

Plně propojené vrstvy se v konvolučních sítích typicky nachází v zadních vrstvách a vrstva výstupní je prakticky vždy plně propojenou vrstvou. Plně propojená vrstva je ve skutečnosti klasická dopředná neuronová síť vložená dovnitř sítě konvoluční. Jejím účelem je přiřadit rysy extrahované dřívějšími vrstvami sítě k jednotlivým klasifikovaným třídám, tedy určit které rysy značí kterou třídu. Parametry této vrstvy jsou počet neuronů ve vrstvě a aktivační funkce, popřípadě její parametry. Výstupní plně propojená vrstva typicky používá aktivační funkci SoftMax (kapitola 2.6) a skryté plně propojené vrstvy často používají funkci hyperbolický tangens (kapitola 2.6).

Kapitola 4

Násobení v neuronových sítích

Jak je zřejmé z předchozího výkladu, násobení je v oblasti neuronových sítí klíčovou operací. Násobení je především používáno při počítání vnitřního potenciálu (bázové funkce) neuronu a je prováděno tak často, že i u relativně malé neuronové sítě hovoříme o desetitisících násobení (okolo 80 000 u sítě BPN (kapitola 7.2)) při počítání odezvy na jeden vstupní vzorek, u hlubokých neuronových sítí se tento počet může pohybovat v řádech statisíců i více. Přestože při počítání aktivačních funkcí můžeme používat i náročnější operace, například hyperbolický tangent (kapitola 2.6), během výpočtu jedné odezvy jsou počítány zhruba v řádu stovek pro malé až střední sítě a až tisíců pro hluboké sítě, v závislosti na počtu neuronů, tedy jeden výpočet pro jeden neuron. Těchto náročnějších operací je tedy prováděno výrazně méně než operací násobní, například u výše zmíněné BPN se téměř blížíme poměru 1 tanh na 1 000 násobení. Z toho vyplývá, že operace násobení je skutečně kritická, a pokud chceme zrychlit výpočet neuronové sítě, popřípadě snížit její cenu a velikost na čipu v případě hardwarové implementace, je zaměření se na operaci násobení správná volba.

V této kapitole si představíme možnosti zobrazení čísel v počítačích a konvenční způsoby provádění násobení s těmito reprezentacemi. Dále si představíme některé používané i nové techniky zvýšení rychlosti, snížení ceny či velikosti na čipu operace násobení, a to především z pohledu použitelnosti v oblasti neuronových sítí. V této kapitole čerpám převážně z následujících zdrojů [4] [5] [10] [31] [32].

4.1 Zobrazení čísel v počítačích

Přímý kód

Přímý kód je pro člověka nejpřirozenějším zobrazením čísel v binární soustavě. Převod čísla z binární soustavy do desítkové je dán následujícím vzorcem.

$$\sum_{n=0}^{N-2} b[n]2^n, \tag{4.1}$$

kde N značí počet bitů čísla a $b[n]$ hodnotu bitu na pozici n. V přímém kódu lze zapsat jak kladná, tak záporná čísla a to tak, že nejvyšší bit je vyhrazen jako znaménkový a určuje tedy, zda následující číslo je kladné $b[N-1] = 0$, nebo záporné $b[N-1] = 1$. To však způsobí, že existují dva zápisy hodnoty 0 (+0 a -0). Při použití tohoto zápisu je však problematické

hardwarově implementovat aritmetické operace, problém činí především operace sčítání dvou čísel s odlišným znaménkem.

Dvojkový doplněk

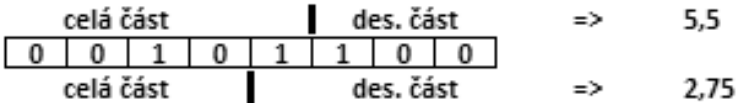
Dvojkový doplněk, který je nejčastěji používaným kódováním pro zobrazení celých čísel v počítači, řeší výše zmíněné problémy spojené s přímým kódem. Kladná čísla jsou v tomto kódu shodná s kladnými čísly v kódu přímém. Záporné číslo k číslu kladnému vytvoříme inverzí bitů kladného čísla a přičtením jedničky, naopak k získání kladného čísla ze záporného opět invertujeme bity záporného čísla a opět přičteme jedničku. I v tomto kódování chápeme nejvyšší bit jako znaménkový. Hlavní výhodou čísel v doplňkovém kódu je jednodušší hardwarová implementace aritmetických operací, a to především proto, že tyto operace jsou totožné nezávisle na znaménku operandů. To platí i pro operaci posunu (shift), avšak při posouvání doprava je třeba uvažovat tzv. aritmetický posun, který zachovává hodnotu znaménkového bitu.

Pevná řádová čárka

Pro kódování desetinných čísel s pevnou řádovou čárkou (fixed-point) kde, jak napovídá název, máme předem pevně určeno kolik bitů reprezentuje celou a kolik řádovou část, nejčastěji používáme dvojkový doplněk, u něhož si pouze zvolíme rozdělení na celou a řádovou část. V počítači je tedy číslo uloženo úplně stejně, rozdíl je pouze v tom, jak jednotlivé hodnoty chápeme a interpretujeme. Na obrázku 4.1 vidíme, jak může být jedno číslo různě interpretováno na základě zvolení hranice mezi celou a řádovou částí. Převod čísla z binární soustavy do desítkové je dán následujícím vzorcem.

$$\sum_{n=0}^{C-1} b_C[n]2^n + \sum_{n=0}^{D-1} b_D[n]2^{-(n+1)}, \tag{4.2}$$

kde $N = C + D$, C značí počet bitů celé části, D počet bitů řádové části, $b_C[n]$, resp. $b_D[n]$ hodnotu bitu v celé, resp. řádové části čísla indexované od řádové čárky doleva, resp. doprava. Celé číslo ve dvojkovém doplňkovém kódu tedy můžeme chápat jako speciální případ čísla s pevnou řádovou čárkou, které má 0 řádových míst. Toto je velmi výhodné především proto, že aritmetické operace mohou probíhat stejně jako u dvojkového doplňku, bez ohledu na řádovou čárku v případě, že operandy mají stejný počet řádových míst, což ale není přílišné omezení, spíše standardní situace. Pouze při násobení dvou desetinných čísel je po provedení násobení potřeba určit celou a řádovou část výsledku. Počet řádových míst (bitů) výsledku je roven součtu počtu řádových míst operandů, tedy pokud budeme násobit dvě 8 bitová čísla s 3 řádovými bity, výsledek bude mít 16 bitů, z čehož bude 6 bitů řádových.



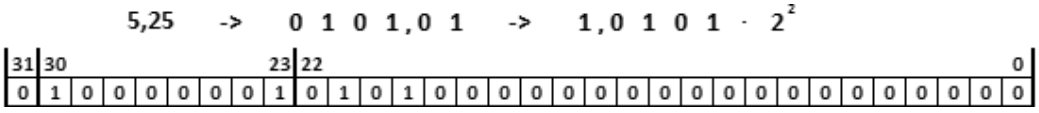
Obrázek 4.1: Ukázka různé interpretace hodnoty uložené v typu pevné řádové čárky

Pohyblivá řádová čárka

Potřeba čísel s pohyblivou řádovou čárkou (floating-point) byla vyvolána neschopností čísel s pevnou řádovou čárkou měnit dynamický rozsah a přesnost „za běhu“, a právě to čísla s pohyblivou řádovou čárkou dokáží. Jsou zobrazována v takzvané vědecké notaci, kde je hodnota čísla určena znaménkovým bitem s, mantisou m, základem b (v případě binárních čísel 2) a exponentem e. Na obrázku 4.2 můžeme vidět ukázku převodu dekadického čísla na číslo binární a následně do notace pohyblivé řádové čárky dle normy IEEE 754-1985 single precision [1], což je 32 bitová hodnota, kde MSB (nejvýznamnější bit) je znaménkový bit, následuje 8 bitů exponentu (e + 127) a zbylých 23 bitů normalizovaná mantisa bez úvodní jedničky. Následující vzorec:

$$(-1)^s * (1 + m) * 2^e \tag{4.3}$$

slouží pro převod čísla v notaci pohyblivé řádové čárky zpět do dekadického tvaru. Hlavní nevýhodou čísel s pohyblivou řádovou čárkou je složitost a doba trvání aritmetických operací s nimi. Je třeba pracovat zvlášť s exponenty a mantisami, čísla různě denormalizovat a opět normalizovat a podobně. Tyto operace jsou tak specifické, že vznikly samostatné koprocesory, které je provádějí (Floating Point Unit) avšak ty jsou svou spotřebou a místem zabíraným na čipu nevhodné pro použití ve větším počtu, a to především ve vestavěných systémech.



Obrázek 4.2: Ukázka převodu čísla do reprezentace s pohyblivou řádovou čárkou

4.2 Konvenční techniky násobení

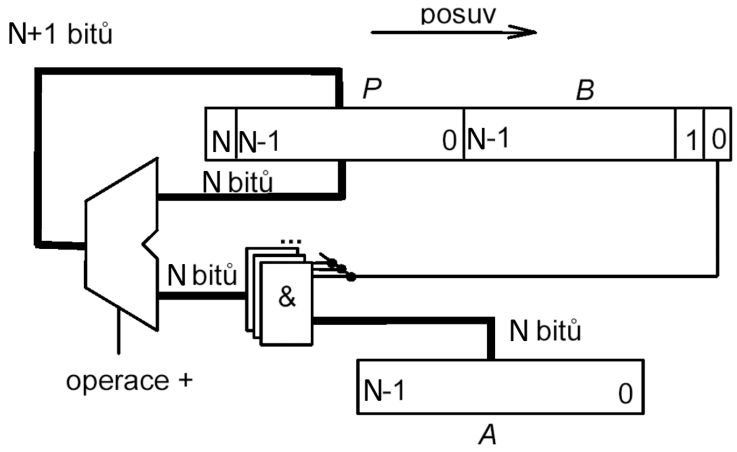
Násobení s pevnou řádovou čárkou

Sekvenční násobička

Jak již napovídá název, tak tato násobička provádí násobení postupně, a tedy vynásobení N bitových operandů jí bude trvat N cyklů, v nichž se musí vykonat následující operace. Nejprve je proveden logický součin operandu A a LSB (nejméně významný bit) registru B, následně je tento součin sečten se spodními N bity registru P, výsledek je uložen opět do registru P a je proveden posuv doprava registru PB, tento proces se opakuje N krát. Nízká rychlost násobení je tedy největší nevýhodou této násobičky. Díky své konstrukci však zabírá poměrně málo místa na čipu, a proto nalezne své uplatnění v některých vestavěných systémech, ovšem v oblasti neuronových sítí je kvůli své nízké rychlosti nepoužitelná. Schéma sekvenční násobičky můžeme vidět na obrázku 4.3.

Kombinační násobička

Tato násobička, jak opět napovídá název, je kombinačním logickým obvodem, a tedy výsledek násobení vrací v jednom cyklu. Schéma kombinační násobičky můžeme vidět na obrázku 4.4. V jednotlivých řadách hradel jsou vyrobeny dílčí součiny operandu A a jednotlivých



Obrázek 4.3: Schéma sekvenční násobičky [31]

bitů operandu B. Dílčí součiny jsou vždy posunuty o jeden bit doleva oproti předchozímu a následně sečteny sčítačkami s postupným přenosem. Tento algoritmus je velice podobný se sekvenční násobičkou s tím rozdílem, že zde je vše prováděno současně. Pro n bitové operandy je zpoždění této násobičky a počet potřebných hradel (velikost na čipu):

$$zpožděni = 14n - 18 \rightarrow \mathcal{O}(n) \quad plocha = 14n^2 - 19n \rightarrow \mathcal{O}(n^2) \quad [4] \tag{4.4}$$

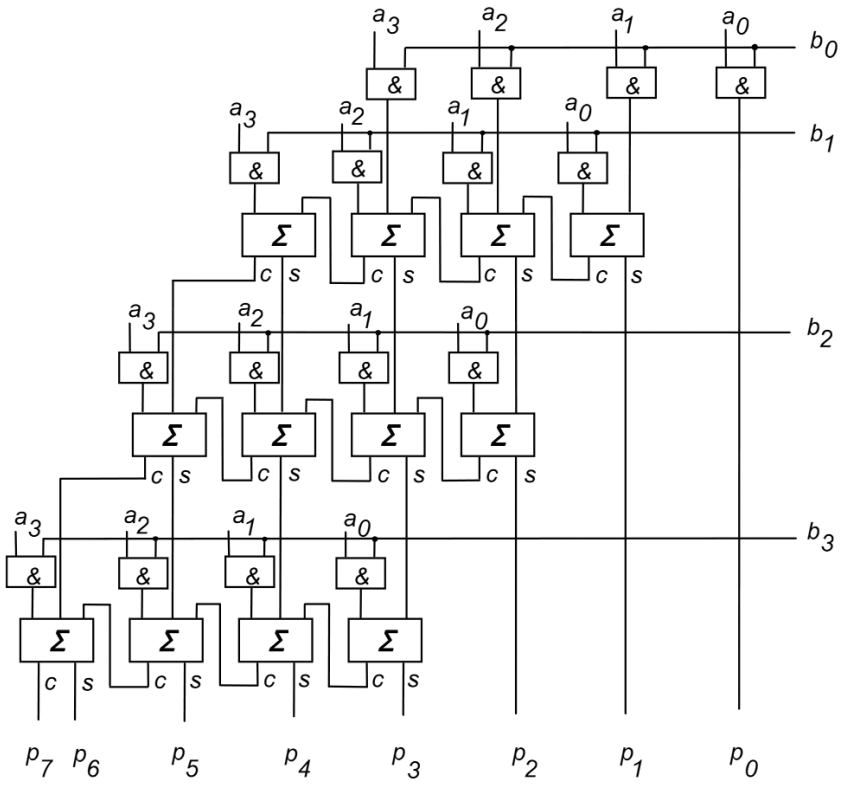
Násobička s uchováním přenosu

Tato násobička je svou strukturou velmi podobná kombinační násobičce s tím rozdílem, že zavádí podmínku nezávislosti sčítaček na sčítačkách ležících ve stejné úrovni. Právě závislost na předchozích sčítačkách na stejné úrovni zpomalovala násobení v kombinační násobičce a to proto, že nejlevější sčítačka v úrovni musí čekat na všechny ostatní, než se k ní dostane přenos. Zde se s přenosy generovanými na jedné úrovni již dále na této úrovni nepočítá, místo toho jsou předány jako vstup sčítaček o úroveň níž a jednu pozici vlevo. Takovou sčítačku nazýváme sčítačkou s uchováním přenosu. Až na poslední úrovni sítě je provedeno sčítání za pomocí sčítaček s postupným přenosem. Pro n bitové operandy je zpoždění této násobičky a počet potřebných hradel (velikost na čipu):

$$zpožděni = 10n - 10 \rightarrow \mathcal{O}(n) \quad plocha = 14n^2 - 19n + 7 \rightarrow \mathcal{O}(n^2) \quad [4] \tag{4.5}$$

Wallaceův strom

Wallaceův strom neboli také zrychlené sčítání částečných součinů při násobení s uchováním přenosu, je metodou násobení, při níž jsou za použití sčítaček s uchováním přenosu představenými výše, sčítány částečné součiny v jiném (vhodnějším) pořadí, než je tomu u násobičky s uchováním přenosu. V případě násobičky s uchováním přenosu je v každé úrovni přičten jeden dílčí součin, naopak v případě Wallaceho stromu jsou již v první úrovni sečteny všechny dílčí součiny a dále je prováděno sčítání těchto součinů a jejich přenosů. Porovnání těchto dvou přístupů můžeme vidět na obrázku 4.5, kde SUP značí blok sčítačky s uchováním přenosu a SPP sčítačku s postupným přenosem. Touto metodou tedy lze redukovat počet



Obrázek 4.4: Schéma kombinační násobičky [31]

úrovní potřebných pro sečtení dílčích součinů, a tak zrychlit výpočet. Pro n bitové operandy je zpoždění této násobičky a počet potřebných hradel (velikost na čipu) následující:

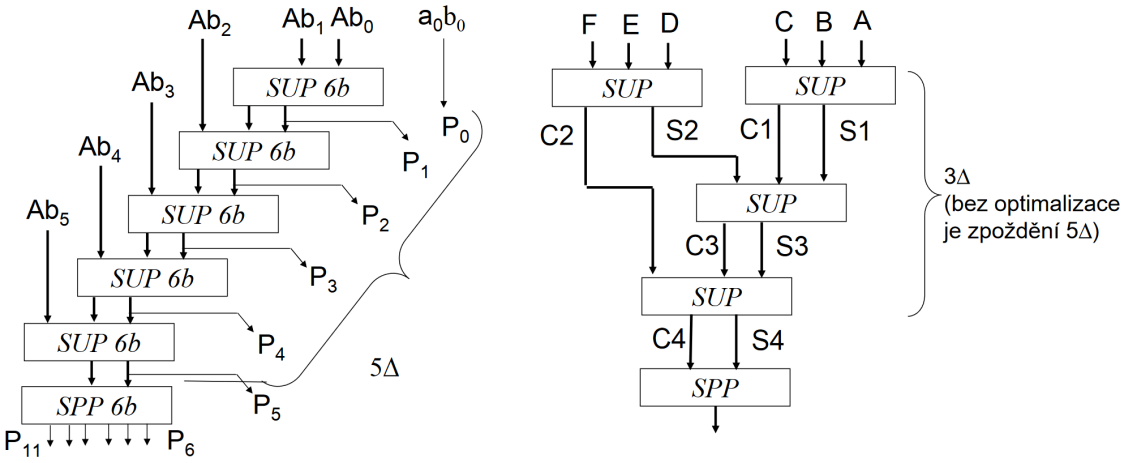
$$zpožděni = 8n + 6D(n) \rightarrow \mathcal{O}(n) \quad plocha = 26n^2 - 36n + 12 \rightarrow \mathcal{O}(n^2), \quad [4] \tag{4.6}$$

kde $D(n)$ je hloubkou Wallaceho stromu danou funkcí:

$$D(n) = \begin{cases} 0 & \text{pro } n \leq 2 \\ 1 & \text{pro } n = 3 \\ D\left(\left\lceil \frac{2n}{3} \right\rceil\right) + 1 & \text{pro } n \geq 4 \end{cases} \quad [4] \tag{4.7}$$

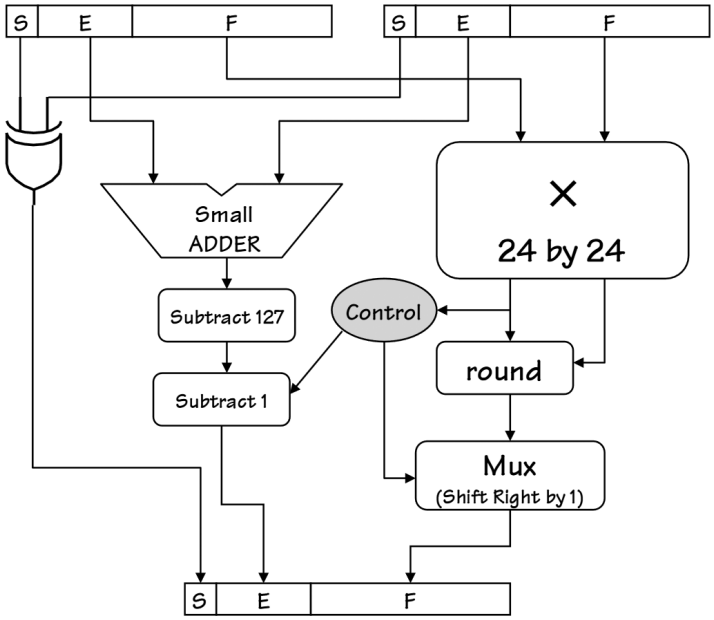
Násobení s pohyblivou řádovou čárkou

Obecně násobení čísel zapsaných ve vědecké notaci, a tedy také násobení čísel s pohyblivou řádovou čárkou podle IEEE 754-1985 [1], se provádí po částech v několika krocích. Konkrétně pro IEEE 754-1985 jsou tyto kroky následující. Nejprve se podle znamének operandů určí znaménko výsledku. Dále jsou sečteny exponenty operandů a je od nich odečteno 127, to proto že každý z exponentů je uložen jako $exp = e + 127$. Následně jsou mezi sebou vynásobeny mantisy operandů. Pokud má výsledná mantisa více než jednu číslici před řádovou



Obrázek 4.5: Porovnání násobičky s uchováním přenosu a Wallaceho stromu [31]

čárkou, je posunuta o 1 doprava a od exponentu výsledku je odečtena 1. Zjednodušené schéma takové násobičky můžeme vidět na obrázku 4.6, tato násobička neuvažuje speciální okrajové případy. Vidíme tedy, že násobička s pohyblivou řádovou čárkou v sobě obsahuje násobičku s pevnou řádovou čárkou, sčítačku a další prvky, proto je její plocha na čipu i zpoždění větší než u násobiček s pevnou řádovou čárkou. Kvůli tomu, ale také kvůli podobně vysoké ceně dalších aritmetických operací, se výpočty s pohyblivou řádovou čárkou v oblasti vestavěných systémů používají pouze, pokud je k tomu skutečný důvod, a tedy v oblasti neuronových sítí se téměř nepoužívají.



Obrázek 4.6: Zjednodušené schéma násobičky s pohyblivou řádovou čárkou [21]

4.3 Techniky optimalizace násobení

Tato sekce uvádí techniky používané pro optimalizaci násobení v neuronových sítích. Tedy s cílem sítě zrychlit, zmenšit jejich velikost na čipu, a to s co možná nejmenším vlivem na jejich úspěšnost. Následující techniky se zabývají optimalizací s použitím datového typu pevné řádové čárky, protože je nejčastěji používán v oblasti neuronových sítí ve vestavěných systémech.

Optimalizace bitové šířky

Základní a nejpoužívanější optimalizací je snižování bitové šířky vah, popřípadě i dalších čísel, se kterými je v rámci sítě nakládáno. Jak je ukázáno výše, velikost násobiček roste kvadraticky a jejich zpoždění lineárně v závislosti na velikosti operandů, tedy při jejich snížení je úspora místa i času znatelná. Při snížení velikosti datového typu je také potřeba méně místa pro uložení parametrů sítě, a to především méně místa pro uložení vah. Zmenšování datového typu můžeme provádět metodou pokus-omyl, dokud bude síť vykazovat potřebnou přesnost. V nedávné době však byly představeny algoritmy (framework Ristretto [11]), které jsou schopny pro konkrétní úlohu a architekturu určit optimální velikost datového typu. Například pro úlohu rozpoznávání čísel MNIST by měl stačit dvoubitový datový typ. Tato optimalizační technika se často používá v kombinaci s dalšími zmíněnými.

Aproximativní násobičky

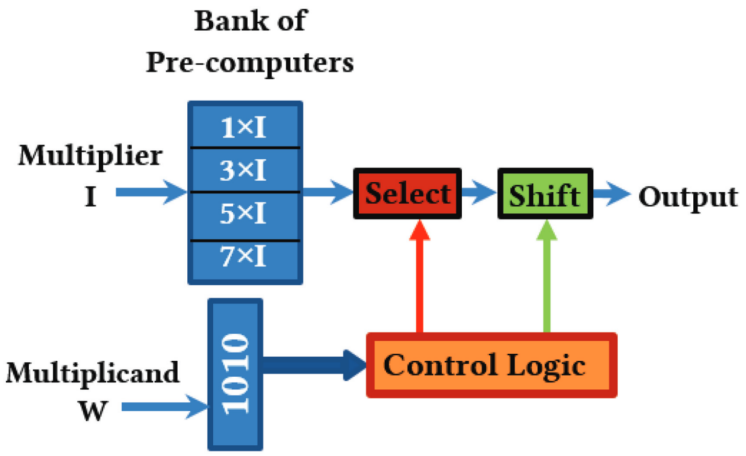
Pro vytváření aproximativních násobiček používáme techniku zvanou funkční aproximace. Při ní v plně definované funkci násobení záměrně označíme hodnoty pro některé (vhodné) kombinace vstupů za nedefinované a to tak, abychom z obvodové realizace této funkce mohli vypustit některé logické členy a tím zmenšit její velikost na čipu, popřípadě rychlost výpočtu. Další možností je kompletně změnit funkci násobení tak, že například nikdy nedostaneme přesný výsledek, avšak chyba bude vždy v jednom bitu. K návrhu takových obvodů se obvykle používají různé heuristiky [22]. Důležité parametry aproximativních obvodů jsou, s jakou pravděpodobností může chyba nastat a jak může být tato chyba velká. Například v aproximované 1 bitové sčítačce podle [35] je o jedno hradlo AND méně a používá pouze dvojvstupé hradlo OR namísto trojvstupého. V případě této sčítačky je aproximována část počítající přenos a chyba může nastávat s pravděpodobností 1/16 a její maximální velikost je 1. Další možnou aproximací sčítání by bylo ignorování některých LSB bitů. Z těchto aproximovaných sčítaček, popřípadě jejich kombinací se sčítačkami úplnými, můžeme skládat aproximované násobičky. Vytváření těchto stavebních bloků, či jejich skládání do větších aproximativních obvodů je v současné době předmětem výzkumu [9].

Použití aproximativních násobiček v oblasti neuronových sítí je umožněno především základní vlastností neuronových sítí, a to jejich schopností tolerovat nepřesnosti ve vstupních datech. Při použití aproximativních násobiček této schopnosti využíváme. Síť se během učení dokáže vypořádat s chybami v operaci násobení podobně, jako se vypořádá s nepřesnostmi ve vstupních datech. Jak velké bude snížení přesnosti celé sítě záleží na úloze, architektuře sítě a použité aproximativní násobičce. Tuto hodnotu je třeba zjistit experimentálně. Před praktickým použitím konkrétní aproximativní násobičky je tedy třeba zvážit její přínos a dopad na konkrétní neuronovou síť.

Násobení bez násobiček

V binární soustavě je možné násobit, popřípadě dělit násobky čísla 2 (2^i) pomocí operace levého, popřípadě pravého posunu. Této skutečnosti využívají násobičky bez násobení, kde je operace násobení nahrazena výběrem předpočítaného mezivýsledku násobení z tabulky předpočítaných hodnot, adresované násobencem a násobitelem a následným posunem o potřebný počet míst doleva. Tabulka je předpočítána při inicializaci systému a je sdílena napříč systémem. Tedy například pro 4b násobení bychom potřebovali předpočítat výsledky všech možných kombinací násobence a těchto hodnot násobitele $\{1, 3, 5, 7, 9, 11, 13, 15\}$, všechny ostatní hodnoty lze získat levým posunem (např. $12x = 3x << 2$) [28] [29].

Tento postup nám dává přesné výsledky násobení, ovšem podobně jako v aproximativním násobení i zde můžeme záměrně omezit přesnost násobení a některé kombinace označit za nedefinované. Učiníme tak omezením hodnot, které může nabývat násobitel (hodnoty vah v případě neuronových sítí), čímž docílíme zmenšení předpočítávané tabulky. Například pokud by tabulka byla předpočítána pouze pro hodnoty násobitele $\{1, 3, 5, 7\}$, pak by bylo možné vyjádřit pomocí posunů hodnoty $\{1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14\}$. Schéma násobení pro takovouto velikost tabulky můžeme vidět na obrázku 4.7 [28] [29].



Obrázek 4.7: Schéma 4b násobičky bez násobení [28]

Pokud váhy neuronové sítě omezíme pouze na násobky dvou, pak předpočítaná tabulka není vůbec potřeba a celé násobení lze provádět pouze pomocí operace posunu. Neuronová síť se i s touto chybou nějakým způsobem vypořádá. Zjistit, jaké to bude mít dopady na její rychlost, přesnost a velikost na čipu je cílem této práce.

Kapitola 5

Knihovny pro práci s neuronovými sítěmi

Na internetu je volně dostupná celá řada nástrojů pro práci s neuronovými sítěmi. V nich lze tvořit neuronové sítě různých architektur, vybírat z různě velké škály bázových i aktivačních funkcí, různých typů vrstev konvolučních sítí a některé z nich také nabízí různé podpůrné nástroje (např. podpora různých formátů vstupních dat). Pomocí nich lze o možnost práce s neuronovými sítěmi rozšířit celou řadu programovacích jazyků, nebo jiných nástrojů (např. Deep Learning Toolbox pro MATLAB[1]). V této kapitole budou představeny některé z nich a také bude vybrán jeden, který bude rozšířen o funkci zjednodušeného násobení a dále bude použit při provádění experimentů, jež jsou součástí této práce.

5.1 TensorFlow

TensorFlow [2] je open-source knihovna pro provádění matematických operací. Práce s neuronovými sítěmi je tedy pouze jednou z funkčností nabízených touto knihovnou. Byla vytvořena skupinou programátorů společnosti Google (skupinou Google Brain) a je zacílena především na vysoký výkon. Knihovna umožňuje provádění výpočtů jak na klasických procesorech (CPU), tak na grafických procesorech (GPU) a na proprietárních veřejně nedostupných procesorech společnosti Google (TPU – tensor processing unit). Části této knihovny jsou psány v jazycích Python, C++ a CUDA. Knihovna nabízí rozhraní pro jazyk Python a omezené rozhraní pro jazyk C++. Protože je tato knihovna poměrně nízko-úrovňová, její rozhraní je komplexní a pro nové uživatele může být práce s ním problematická. Pro její komplexnost a rychlost je tato knihovna pravděpodobně nejpoužívanější knihovnou svého druhu a k jejímu používání se kromě Googlu, jež ji stvořil, hlásí firmy Intel, AMD, či NVIDIA. Avšak kvůli její komplexnosti by bylo časově příliš náročné proniknout do jejího kódu a provézt úpravy vyžadované v této práci.

5.2 Keras

Keras [7] je open-source knihovna pro práci s neuronovými sítěmi. Jedná se o nadstavbu schopnou pracovat nad několika nízko-úrovňovými knihovnami nabízejícími podporu pro práci s neuronovými sítěmi. Jedná se o knihovny Microsoft Cognitive Toolkit, Theano,

[1] https://www.mathworks.com/products/deep-learning.html

a především výše zmíněný TensorFlow. Keras byl vytvořen především Françoisem Chollet-em, programátorem společnosti Google, pro zprostředkování uživatelsky přívětivějšího rozhraní nad výše zmíněnými knihovnami. Cenou za to, ale také požadovanou vlastností je, že knihovna zprostředkovává pouze práci s neuronovými sítěmi. Keras je napsán v jazyce Python a rozhraní nabízí také pouze pro jazyk Python. Keras se stal tak oblíbenou knihovnou pro práci s neuronovými sítěmi nad knihovnou TensorFlow, že byl v roce 2017 včleněn jako součást jejího API.

5.3 TypeCNN

TypeCNN [25] je open-source knihovna pro práci s klasickými a především s konvolučními neuronovými sítěmi a umožňuje tedy jejich návrh, trénování i použití. Tuto knihovnu v roce 2018 navrhl a implementoval Petr Rek v rámci své diplomové práce vytvořené na FIT VUT v Brně. Knihovna je psána objektově v jazyce C++ a hlavní důraz při jejím návrhu byl kladen na práci s datovými typy. Knihovna umožňuje použití až tří různých datových typů v jedné neuronové síti současně a to typ pro uložení vah, inferenci a trénování. Datový typ pro trénování je totožný pro všechny vrstvy, avšak datový typ pro váhy a inferenci je dokonce možné volit různý pro každou z vrstev sítě. Výchozím datovým typem knihovny je pohyblivá řádová čárka na 32 bitech. Autor také přikládá jím implementovaný datový typ pevné řádové čárky, u nějž je možné volit bitovou šířku před i za řádovou čárkou, společně s operacemi nutnými pro použití datového typu (aritmetické, logické atd.). Další datové typy společně s jejich operacemi může nadefinovat sám uživatel, popřípadě může upravit pouze některé operace v již existujícím datovém typu, jako například operace násobení v typu pevné řádové čárky. Kromě výše zmíněných funkčností knihovna nabízí také podpůrné nástroje jako je ukládání a načítání neuronové sítě a podpora různých formátů vstupních dat. V knihovně je možné rovnou pracovat s formáty často používaných datových sad, jako jsou MNIST či CIFAR-10.

Po konzultaci s vedoucím jsem pro provádění experimentů v rámci této diplomové práce zvolil právě tuto knihovnu. Důvodů této volby je několik a to implementační jazyk C++, zdokumentovaný návrh i implementace v rámci diplomové práce, využití práce vzniklé na naši fakultě, možnost konzultace úprav přímo s autorem či vestavěná podpora různých datových sad. Hlavním důvodem je však podpora různých datových typů a především různých operací nad těmito typy.

5.4 SimpleCNN

SimpleCNN [6] je open-source knihovna, která byla napsána Canem Bölükem za účelem vytvořit snadno použitelnou a hlavně snadno čitelnou knihovnu pro práci s konvolučními neuronovými sítěmi. Toho bylo bezesporu dosaženo. Knihovna je napsána přehledně v jazyce C++. Nicméně pro její jednoduchost je v ní implementována pouze jedna bázová, aktivační a ztrátová funkce a pouze jeden optimalizátor. To téměř znemožňuje její praktické použití. Z podpůrných nástrojů této knihovně bohužel chybí možnost uložit a načíst natrénovanou neuronovou síť (její strukturu a váhy) a chybí zde také podpora různých formátů vstupních dat. Pro použití knihovny v této práci by bylo potřeba tyto nástroje doimplementovat.

Kapitola 6

Návrh a implementace zjednodušené operace násobení

Pro zjednodušení operace násobení, které bude navrženo, implementováno a testováno v této práci jsem si zvolil techniku násobení bez násobiček popsanou v kapitole 4.3. Tato technika mě ze zkoumaných technik zaujala nejvíce, protože navrhuje celou náročnou operaci násobení nahradit za poměrně snadnou operaci posuvu. Návrh této techniky je zásadně inspirován těmito pracemi [28] [29].

6.1 Návrh

Standardní operace násobení bude tedy nahrazena za operaci levého posuvu vhodné hodnoty (mezivýsledku násobení) o potřebný počet bitů (dále jen zjednodušené násobení). Operace zjednodušeného násobení bude definována pouze pro kladná celá čísla libovolné bitové šířky. Protože operaci násobení upravuji pro použití v oblasti neuronových sítí, budu nadále násobence nazývat vstupní hodnotou (vstupem) a násobitele hodnotou váhy (váhou). Pro toto zjednodušené násobení je třeba znát bitovou šířku operandů N a zvolit takzvanou abecedu vah. Na jejich základě je sestavena tabulka předem vypočítaných mezivýsledků (dále jen tabulka mezivýsledků).

Abeceda vah je seznam vhodně zvolených (bázových) hodnot, z nichž je odvozen obor hodnot vah a sestavena tabulka posuvů. Bázové hodnoty je vhodné volit tak, že žádná z nich nelze získat bitovým posuvem některé jiné bázové hodnoty. Vhodnou abecedou vah může být např. $\{1, 3, 5\}$. Obor hodnot vah získáme levým posuvem jednotlivých bázových hodnot. Každá z nich je opakovaně posouvána, dokud by výsledek posuvu nebyl větší než největší zobrazitelné číslo $(2^N - 1)$. Během zjišťování oboru hodnot vah je vytvořena tabulka posuvů. Ta je adresována bázovou hodnotou a hodnotou váhy a určuje, o kolik bitů vlevo je třeba posunout bázovou hodnotu, abychom získali hodnotu váhy. Například pro bázovou hodnotu 3 a hodnotu váhy 12 by záznam tabulky vypadal následovně: $shift[3][12] = 2$.

Tabulka mezivýsledků je adresována bázovou a vstupní hodnotou a obsahuje předpočítaný výsledek násobení těchto hodnot. Například pro bázovou hodnotu 3 a vstupní hodnotu vstupu 4 by záznam tabulky vypadal následovně: $precomp[3][4] = 12$. Tabulka pro každou bázovou hodnotu z abecedy vah obsahuje výsledek jejího násobení se všemi možnými vstupními hodnotami. Tabulka mezivýsledků tedy obsahuje $(N^2 * pocet\_bazovych\_hodnot)$ N bitových hodnot. Pokud se v abecedě vah nachází bázová hodnota 1, je vzorec pro výpočet

velikosti tabulky mezivýsledků upraven ($N^2 * pocet\_bazovych\_hodnot - 1$), protože výsledek násobení číslem 1 není třeba uchovávat.

Vstupem upravené operace násobení je vstupní hodnota, jejíž rozsah není nijak omezen a váha, jejíž hodnota musí patřit do oboru hodnot vah odvozeného výše. Z hodnoty váhy je zjištěna bázová hodnota, k níž tato váha náleží ($vaha \mod baze = 0$). Následně je z tabulky mezivýsledků za pomocí bázové a vstupní hodnoty zjištěna hodnota mezivýsledku. Z tabulky posuvů je na základě bázové hodnoty a hodnoty váhy zjištěn potřebný počet bitů posuvu. Výsledkem upravené operace násobení je mezivýsledek posunutý doleva o potřebný počet bitů.

Například pro abecedu vah $\{1, 3\}$, vstupní hodnotu 4 a hodnotu váhy 12 by výpočet vypadal následovně:

$$\begin{aligned} baze &= 3 \quad (12 \mod 3 = 0) \\ mezivysledek &= precomp[3][4] = 12 \\ posuv &= shift[3][12] = 2 \\ vysledek &= 12 << 2 = 48 \end{aligned} \tag{6.1}$$

6.2 Implementace

Modul zjednodušeného násobení je stejně jako zbytek knihovny TypeCNN implementován v jazyce C++14. Tabulka mezivýsledků je implementována pomocí asociativní paměti obsahující pro každou bázovou hodnotu pole mezivýsledků.

$$std :: map < int32\_t, std :: vector < int32\_t >> precomp\_vals$$

Tabulka posuvů je implementována jako asociativní paměť, která pro každou bázovou hodnotu obsahuje další asociativní paměť.

$$std :: map < int32\_t, std :: map < int32\_t, int32\_t >> precomp\_shifts$$

Ta pro všechny hodnoty vah (které je možné získat jako levý posuv bázové hodnoty a které jsou menší než největší zobrazitelná hodnota) obsahuje počet bitů, o něž byla bázová hodnota posunuta, aby tato váha vznikla. Zjednodušeně tedy můžeme operaci zjednodušeného násobení popsat následujícím výrazem:

$$res = precomp\_vals[base][input] << precomp\_shifts[base][weight] \tag{6.2}$$

Protože se při načítání uložené sítě, nebo při výpočtu nové váhy v průběhu učení můžeme setkat s váhou, která je mimo obor hodnot vah, je třeba jí přiřadit hodnotu z toho oboru. Pro takovou váhu je z oboru hodnot nalezena maximální hodnota, která je menší než váha a minimální hodnota, která je větší než váha. Jako nová váha je použita hodnota, která je váze bližší. Pokud je jejich vzdálenost od váhy totožná, zvolí se jedna z nich náhodně.

Problémem tohoto přístupu je učení sítě, a to kvůli vzrůstajícímu rozdílu (vzdálenosti) dvou sousedních hodnot z oboru hodnot vah. Například při použití abecedy vah $\{1\}$ je rozdíl sousedních hodnot $2-1 = 1$ a rozdíl jiných sousedních hodnot $16-8 = 8$. Tento fenomén činí problém při volbě koeficientu učení standardních optimalizátorů. Při zvolení příliš malého koeficientu učení nemusí nikdy dojít k nastavení větší váhy a dojde tak k uváznutí sítě. Pokud je však koeficient učení zvolen příliš velký, váhy nižších hodnot se budou neustále měnit a nedojde k naučení sítě. Optimalizátor řešící tento problém není znám a může být předmětem dalšího výzkumu.

6.3 Integrace do TypeCNN

Takto navrženou a naimplementovanou zjednodušenou násobičku jsem integroval do knihovny TypeCNN [25]. Implementace modulu pro zjednodušené násobení se nachází v souboru *MultlessMult.cpp*, který je k nalezení na CD v adresáři *TypeCNN/src/Utils*. V této knihovně bude násobička použita společně s datovým typem pevné řádové čárky, jež je součástí knihovny. Protože je však operace zjednodušeného násobení definována pouze pro kladná celá čísla, je třeba před provedením této operace hodnotu vstupu a váhy upravit. Nejprve je zjištěno znaménko výsledku a dále se pracuje s absolutní hodnotou vstupu i váhy. Vstup i váha jsou posunuty o počet bitů za řádovou čárkou doleva. Tím získáme kladná celá čísla, která jsou vstupem upravené operace násobení. Po provedení násobení je výsledku nastaveno správné znaménko a je posunut o dvojnásobek počtu bitů za řádovou čárkou doprava. Vznikne tak správný výsledek v datovém typu, s nímž knihovna pracuje. Tato úprava je nutná pouze v softwarové simulaci, při použití v hardwaru nezáleží na tom, kam pomyslně umístíme řádovou čárku.

Přepnutí na zjednodušené násobení, a kontrolu a úpravu vah, která s tímto násobením souvisí, je provedeno definicí makra $MULTLESS\_MULT$ při překladu. Také je nutné nastavit abecedu vah prostřednictvím systémové proměnné $ALPHABET\_VALS$, která obsahuje jednotlivé bázové hodnoty oddělené čárkou (např. $ALPHABET\_VALS =$ "$1, 3, 5$").

Pro správnou funkci zjednodušeného násobení je třeba při načítání uložené sítě a úpravě vah při učení tyto váhy volit pouze z oboru hodnot vah. Proto je v modulu perzistence po načtení a ve všech optimalizátorech po spočítání váhy volána funkce pro její transformaci na povolené hodnoty ($mlmTransWeight()$). Protože zjednodušené násobení použijeme pouze při počítání vnitřního potenciálu neuronu je funkce $multlessMult()$ volána pouze z plně propojené a konvoluční vrstvy a ne např. z vrstvy aktivační.

Kapitola 7

Testování a experimenty

Jak bylo řečeno dříve, pro práci s konvolučními neuronovými sítěmi bude použita knihovna TypeCNN. V této kapitole budou představeny typické problémy používané pro hodnocení výkonnosti jak zvolené architektury konvoluční neuronové sítě, tak zvolené knihovny. Dále budou navrženy a popsány architektury neuronových sítí, které budou použity pro řešení těchto problémů. Následně popíši zvolené testy a experimenty, které budou provedeny. Poté budou představeny a diskutovány jejich výsledky.

7.1 Testovací sady

MNIST

Za základní a prakticky povinnou testovací sadu považujeme databázi MNIST [19] (Modified National Institute of Standards and Technology database). Jedná se o databázi černobílých (1 kanál 0-255 stupňů šedi) obrázků ručně psaných číslic rozsahu 0-9 o velikosti 28×28 pixelů. Řešenou úlohou je tedy klasifikace vstupního vzorku do jedné z deseti tříd. Databáze MNIST obsahuje 60 000 trénovacích a 10 000 testovacích vzorků, u nichž jsou známy třídy, do kterých spadají. Tuto databázi v roce 1998 vytvořili Yann LeCun, Corinna Cortes a Christopher J.C. Burges jako podmnožinu rozsáhlejších databází institutu NIST, konkrétně databází SD-1 a SD-3. Databáze SD-1 obsahuje 60 000 obrázků ručně psaných číslic pocházejících od 500 různých středoškolských studentů. Databáze SD-3 je velice podobná, s tím rozdílem, že je pořízena mezi úředníky. Čísla obsažená v databázi SD-3 jsou tedy úhlednější a jejich klasifikace je úspěšnější. Z důvodu zvětšení diverzity trénovacích a testovacích dat tedy došlo k vytvoření databáze MNIST. Trénovací datová sada MNIST tedy obsahuje 30 000 vzorků z databáze SD-1 a 30 000 vzorků z databáze SD-3 pocházejících přibližně od 250 pisatelů. Testovací sada MNIST je složena z 5 000 vzorků z databáze SD-1 a 5 000 vzorků z databáze SD-3. Autoři zaručují, že vzorky z trénovací a testovací sady jsou vzájemně disjunktní.

Výsledky klasifikace dosažené pro testovací sadu MNIST jsou pro účely porovnání součástí téměř každé práce z oblasti konvolučních neuronových sítí a proto tomu nebude jinak ani u této práce. V současnosti nejúspěšnější neuronová síť řešící tuto úlohu dosahuje přesnosti 99.79 %. Pro srovnání nejvyšší přesností v době vytvoření sady MNIST bylo 99.3 % dosažených sítí LeNet-4 [3].

Obrázek 7.1: Ukázka obrázků databáze MNIST [33]

CIFAR-10

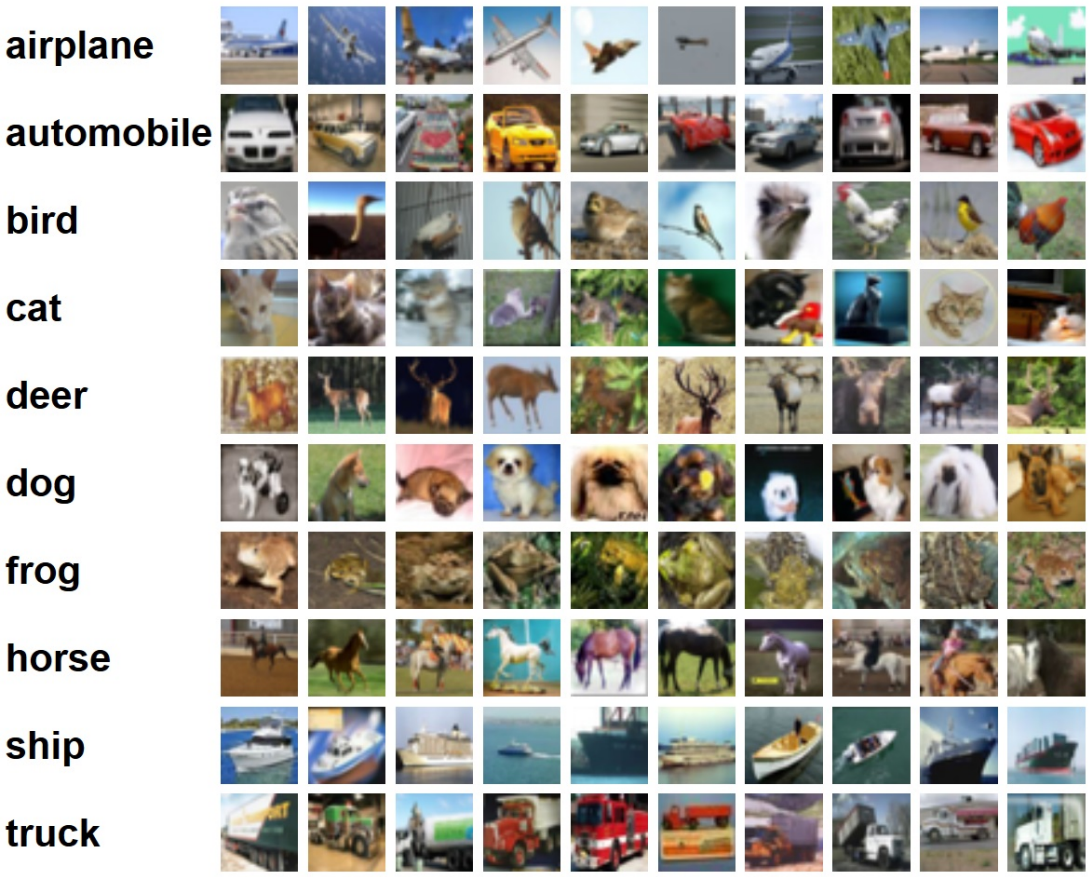
Další pro srovnání často používanou datovou sadou je CIFAR-10 [16]. Jedná se o databázi obsahující 60 000 barevných (3 kanály RGB z rozsahu 0-255) obrázků o velikosti 32×32 pixelů patřících do deseti tříd. Třídami jsou: letadla, automobily, ptáci, kočky, jeleni, psi, žáby, koně, lodě a nákladní automobily. Databáze je rozdělena na 50 000 trénovacích a 10 000 testovacích vzorků, u nichž jsou známy třídy, do kterých spadají. Řešenou úlohou je tedy také klasifikace vstupního vzorku do jedné z deseti tříd. Nicméně je to úkol o poznání složitější než klasifikace číslic a to především kvůli vnitrotřídní variabilitě vzorků, tedy dva objekty spadající do stejné třídy si nemusí být příliš podobné. Složitost úkolu dále zvyšuje různé pozadí objektů stejné třídy a možnost, že dva objekty různých tříd budou mít stejné pozadí. Databáze CIFAR-10 je podmnožinou mnohem větší databáze obsahující 80 000 000 obrázků, které byli sesbírány pány Alexem Krizhevskym, Vinodem Nairem a Geoffrym Hintonem, a skládá se z 6 000 obrázků z každé z výše zmíněných tříd. V současnosti nejúspěšnější neuronová síť řešící tuto úlohu dosahuje přesnosti 96.53 % [3].

Sesterskou databází je CIFAR-100, která stejně jako CIFAR-10 obsahuje 60 000 barevných obrázků o velikosti 32×32 pixelů, nicméně ty patří do 100 tříd, tedy databáze je složena ze 600 obrázků z každé třídy. Tím je úloha klasifikace opět ztížena a v současnosti nejúspěšnější neuronová síť řešící tuto úlohu dosahuje přesnosti 75.72 % [3].

7.2 Testovací architektury

BPN

Pro testovací úlohu klasifikace čísel testovací sady MNIST jsem zvolil architekturu BPN (Back Propagation Network) což je z pohledu dříve uvedených architektur plně propojená



Obrázek 7.2: Ukázka obrázků databáze CIFAR-10 [17]

dopředná síť. Konkrétně volím třívrstvou síť s jednou skrytou vrstvou. Tato síť je složena z vstupní (formální) vrstvy obsahující 784 neuronů (28×28, tedy pro každý pixel jeden), následuje skrytá vrstva obsahující 100 neuronů a výstupní vrstva obsahující 10 neuronů (pro každou třídu jeden). Vstupní a skrytá vrstva jsou rozšířeny o jeden formální neuron pro bias. Všechny neurony používají lineární bázové funkce. Skrytá vrstva jako aktivační funkci používá hyperbolický tangens a výstupní vrstva používá aktivační funkci SoftMax. Tato síť celkově obsahuje přibližně 80 000 vah, které budou učeny. Tuto architekturu i s konkrétními parametry jsem zvolil proto, že je použita ve studiích, z nichž budu dále čerpat, pod označením MPL (Multi Layer Perceptron), a umožní tak srovnání našich výsledků.

CNN

Pro testovací úlohu klasifikace obrázků testovací sady CIFAR-10 jsem zvolil architekturu konvoluční neuronové sítě (dále jen CNN), kterou použil Petr Rek při testování jím vytvořené knihovny TypeCNN [25], jež je použita při vytváření této práce. Tato síť se skládá ze vstupní, tří konvolučních, dvou poolingových, dvou drop-outových a dvou plně propojených vrstev. Ze vstupní 32×32×3 vrstvy vzniká konvoluční vrstva 28×28×12 vytvořená konvolucí s oknem 5×5×3. Dále je proveden pooling s oknem 2×2 a krokem 2 do vrstvy o velikosti 14×14×12. Nad ní je provedena další konvoluce s oknem 5×5×12 a je tak vytvořena vrstva o velikosti 10×10×24. Následuje pooling s oknem 2×2 a krokem 2, čímž je

vytvořena vrstva o velikosti 5×5×24. Poté je proveden drop-out a poslední konvoluce s oknem 5×5×24, čímž je vytvořena vrstva o velikosti 6×6×120. Ta je vstupem plně propojené vrstvy s 84 neurony, následuje další drop-out a konečně výstupní vrstva o 10 neuronech. Konvoluční i plně propojené vrstvy používají aktivační funkci Leaky ReLU. Poolingové vrstvy provádějí tzv. Max-pooling. Tato síť celkově obsahuje více než 90 000 vah, které budou učeny, což je o 10 000 vah více oproti první uvažované architektuře.

7.3 Metodika testování a experimentů

Testování bude probíhat na virtuálním stroji běžícím na mém osobním PC. Virtuální stroj má k dispozici 4 GB paměti RAM a 4 jádra procesoru Intel Core i5 750 o frekvenci 2.66 GHz. Na testovacím stroji je nainstalován 64-bitový operační systém Ubuntu verze 18.04 LTS.

Testování a experimenty můžeme rozdělit do tří fází. První fází jsou základní testy za použití standardní operace násobení a nejpřesnějšího knihovnou podporovaného datového typu pohyblivé řádové čárky na 32 bitech (dále jen float). Ty slouží nejen pro ověření funkčnosti knihovny, ale také pro ověření funkčnosti zvolených testovacích architektur pro konkrétní testovací úlohu. Další fází jsou experimenty se snižováním bitové šířky jak dopředného datového typu, tak datového typu pro uložení vah, za pomoci datového typu s pevnou řádovou čárkou (dále jen FX<počet bitů před řádovou čárkou, počet bitů za řádovou čárkou>). Jsou prováděny na sítích natrénovaných v prvním kroku a je zde stále používána standardní operace násobení. Poslední, a pro tuto práci nejdůležitější, fází jsou experimenty s násobením bez násobičky, a to s různými hodnotami abecedy vah. Tyto experimenty jsou prováděny na některých sítích vzniklých a natrénovaných v druhém kroku.

Základní testy

Pro architekturu BPN bude provedeno 10 nezávislých trénovacích běhů. Pro architekturu CNN bude provedeno pouze 5 nezávislých trénovacích běhů, a to z důvodu jejich časové náročnosti. Z nich je zaznamenán výstup programu, a uložen předpis sítě a hodnoty jejích natrénovaných vah. Každý takový běh je tvořen 10 epochami trénování a jednou epochou dotrénování. Po každé z epoch následuje validace sítě společně s výpisem úspěšnosti klasifikace sítě. Počáteční hodnoty vah sítě jsou voleny náhodně.

Pro obě úlohy jsem zvolil totožné parametry trénování i dotrénování. A sice optimalizátor SGDM (Stochastic Gradient Descent with Momentum) s koeficientem učení 0.005, koeficientem úbytku vah 0.001 a momentem 0.9 pro trénování a optimalizátor SGD s koeficientem učení 0.001 pro dotrénování. Jako ztrátovou funkci jsem pro všechny případy zvolil střední kvadratickou chybu.

Redukce bitové šířky

Při experimentech s redukcí bitové šířky jsou načteny jednotlivé sítě i s hodnotami jejich vah získané během základního testování. Tedy pro BPN se provádí 10 nezávislých běhů dotrénování a pro CNN se provádí 5 nezávislých běhů dotrénování, a to pro každý experiment pro danou architekturu. Z nich je opět uložen předpis sítě a hodnoty natrénovaných vah. Každý běh je tvořen 3 epochami dotrénování s použitím optimalizátoru SGD s koeficientem učení 0.005 a ztrátovou funkcí střední kvadratické chyby.

Pro architekturu BPN i CNN jsem zvolil pro počáteční snížení bitové šířky operandů datový typ $FX < 8, 8 >$ a to jak pro dopředné šíření, tak pro uložení vah. Následně je

bitová šířka dále redukována a redukovaný datový typ je opět použit pro dopředné šíření i uložení vah. Konkrétní datové typy jsou voleny experimentálně, až do snížení přesnosti klasifikace sítě o cca 5 %.

Pro architekturu BPN je zvolen a zafixován datový typ pro dopředné šíření. Typ je vybírán z výše testovaných datových typů a to tak, že je zvolen typ s nejmenší bitovou šířkou, jehož přesnost klasifikace není snížena o více než 0.5 %. Následně jsou provedeny experimenty s redukováním bitové šířky datového typu vah. Jako počáteční je zvolen typ dopředného šíření a jeho bitová šířka je dále redukována. Konkrétní datové typy jsou voleny experimentálně, až do snížení přesnosti klasifikace sítě o cca 5 %.

Násobení bez násobičky

Při experimentech s násobením bez násobiček jsou načteny zvolené sítě i s hodnotami jejich vah získané během experimentů s redukcí bitové šířky operandů. Váhy jsou převedeny na povolené hodnoty, závisející na zvolené abecedě vah. Následně je provedena validace sítě, jejíž výsledek je zaznamenán.

Pro architekturu BPN i CNN jsem zvolil sítě z počátečního experimentu snižování bitové šířky, dotrénované na datovém typu $FX < 8, 8 >$. Pro architekturu BPN byla navíc zvolena síť z experimentu se snižováním bitové šířky datového typu pro uložení vah, jejíž datový typ má nejmenší bitovou šířku a zároveň přesnost klasifikace není snížena o více než 1 %. Pro každou z těchto sítí jsou provedeny experimenty s následujícími abecedami vah $\{1, 3, 5, 7, 9\}$, $\{1, 3, 5, 7\}$, $\{1, 3, 5\}$, $\{1, 3\}$, $\{1\}$.

7.4 Výsledky testování a experimentů

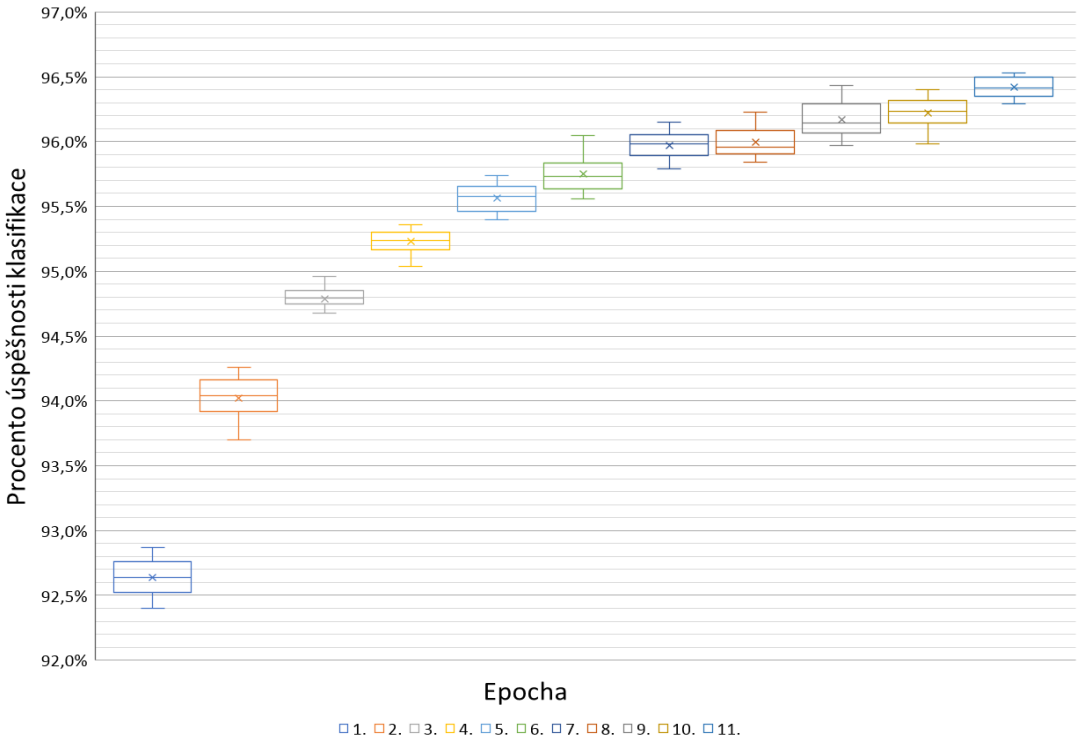
Scripty a programy (přeložené na školním serveru Merlin[1]) použité při testovaní jsou k nalezení na CD v adresáři *TypeCNN/juhanak* společně s výsledky všech testů a natrénovanými sítěmi, které se nachází v adresáři *TypeCNN/juhanak/bpn_res* resp. *TypeCNN/juhanak/cnn_res*. Všechny následující výsledky a hodnoty zanesené v grafech jsou získány validací dané sítě na příslušných testovacích datech, které jsou disjunktní s daty trénovacími.

Základní testy

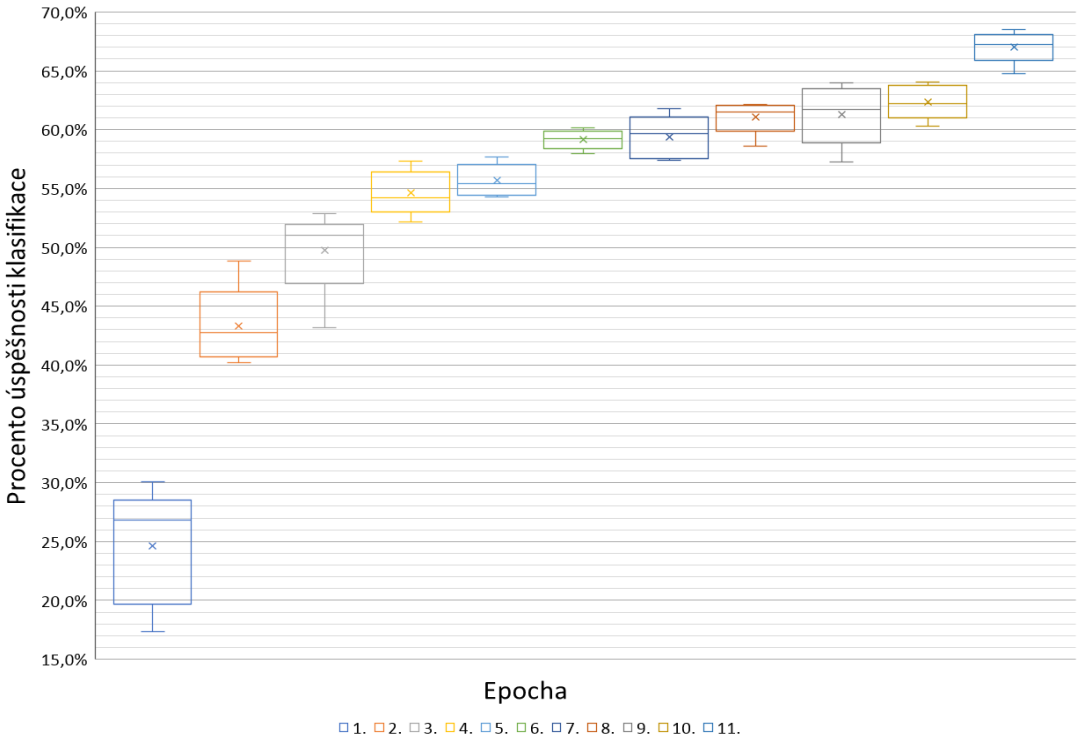
Jedna epocha trénování sítě BPN na datové sadě MNIST trvá cca 100 sekund a jeden trénovací běh trvá více než 15 minut. Celé testování této úlohy tedy trvalo téměř 3 hodiny. Jedna epocha trénování sítě CNN na datové sadě CIFAR-10 trvá přes 1 hodinu, což je čtyřikrát více než celý trénovací běh sítě BPN. Jeden testovací běh této úlohy tvá cca 14 hodin a celé testování trvalo 3 dny. Přestože by se mohlo zdát, že by doba trénování těchto dvou architektur mohla být podobná, protože architektura CNN trénuje pouze o 10 000 vah více než architektura BPN, není tomu tak z důvodu vyšší časové náročnosti konvolučních sítí na učení. Základní testy architektury CNN tedy celkově trvaly 24x déle než základní testy architektury BPN.

Graf 7.3 zobrazuje úspěšnost klasifikace v průběhu trénování po jednotlivých epochách trénování pro všechny trénovací běhy úlohy klasifikace nad datovou sadou MNIST sítí BPN. Graf nám může připomínat logaritmickou funkci. V prvních epochách se úspěšnost klasifikace rychle zvyšuje, naopak během posledních epoch trénování se úspěšnost zvyšuje pouze minimálně. U poslední epochy vidíme mírný skok ve zvýšení úspěšnosti klasifikace,

[1] merlin.fit.vutbr.cz



Obrázek 7.3: Úspěšnost klasifikace sítě BPN na úloze MNIST

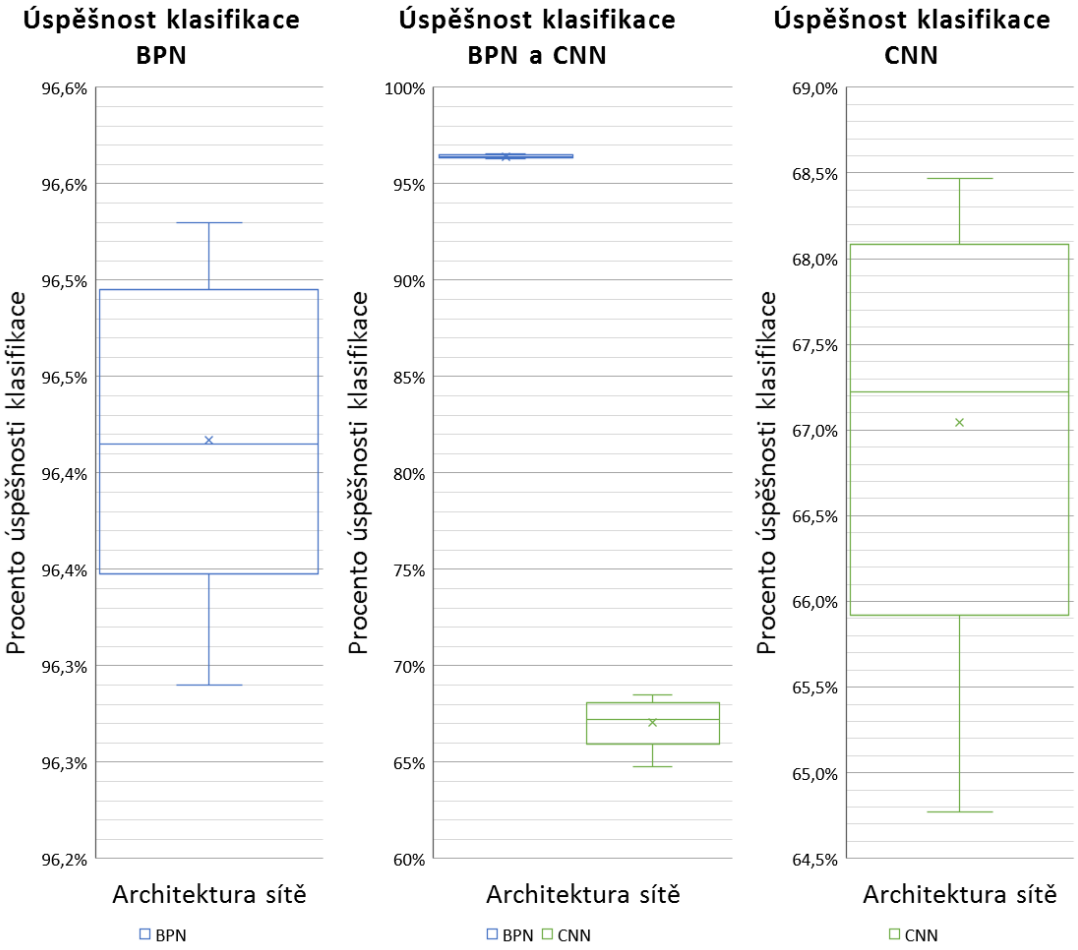


Obrázek 7.4: Úspěšnost klasifikace sítě CNN na úloze CIFAR-10

který je výsledkem jedné epochy jemnějšího dotrénování. Maximální medián úspěšnosti klasifikace se objevil v poslední epoše a má hodnotu 96.4 %, celkové maximum úspěšnosti klasifikace se také objevilo v poslední epoše a má hodnotu 96.5 %.

Totéž, ale pro úlohu klasifikace nad datovou sadou CIFAR-10 sítí CNN můžeme vidět na grafu 7.4. Graf nám opět může připomínat logaritmickou funkci přibližující se úspěšnosti 63 %. U poslední epochy nyní vidíme výraznější skok ve zvýšení úspěšnosti klasifikace, který je výsledkem jedné epochy jemnějšího dotrénování. Maximální medián úspěšnosti klasifikace se objevil v poslední epoše a má hodnotu 67.2 % a celkové maximum, které se objevilo v téže epoše má hodnotu 68.5 %. Výsledky klasifikace této úlohy dopadly hůře než výsledky úlohy předchozí, nicméně pro úlohu CIFAR-10 a použitou architekturu CNN je tento výsledek uspokojivý. Pro zvýšení úspěšnosti klasifikace by bylo nutné použít hlubší a větší konvoluční sítě, jejichž použití na osobním počítači je z důvodu časové náročnosti nepraktické.

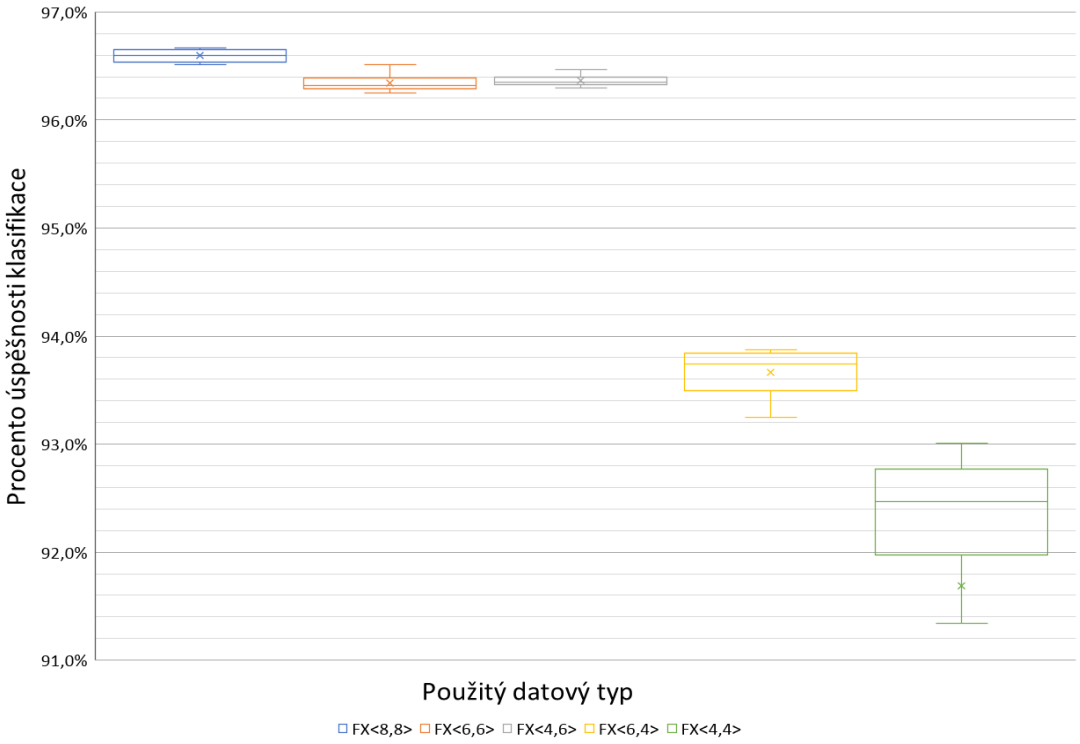
Na grafu 7.5 můžeme vidět srovnání úspěšnosti klasifikace posledních epoch trénování pro obě testovací úlohy a všechny trénovací běhy.



Obrázek 7.5: Porovnání úspěšnosti klasifikace sítě BPN na úloze MNIST a sítě CNN na úloze CIFAR-10

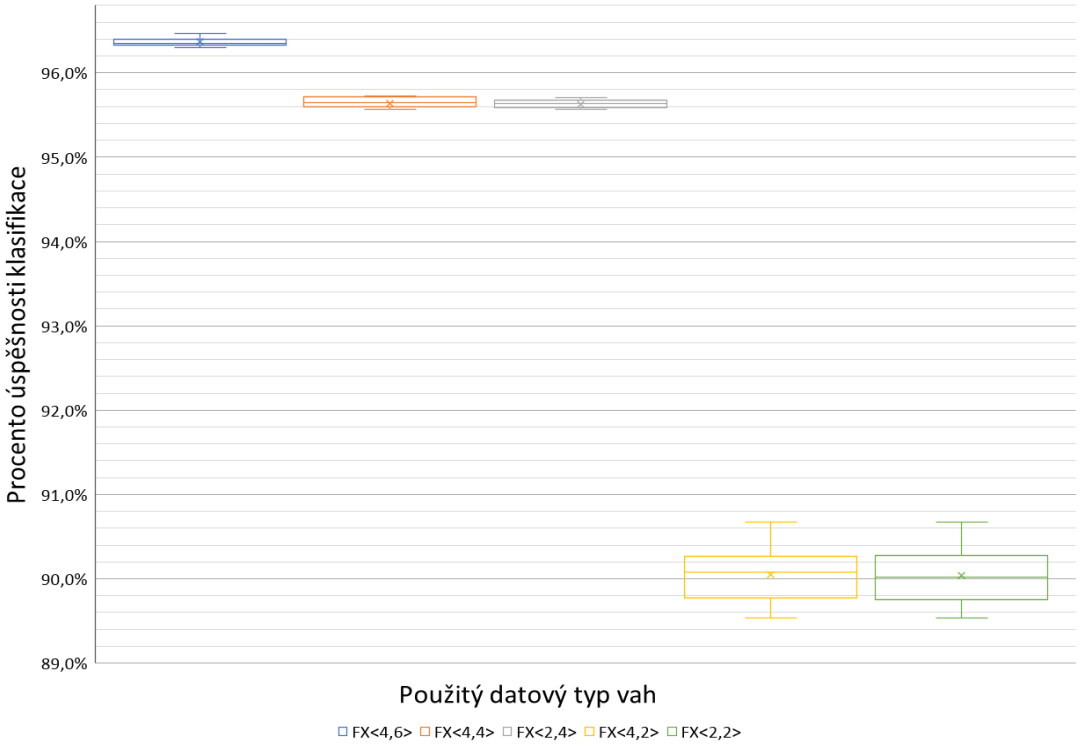
Redukce bitové šířky

Na grafu 7.6 je znázorněna výsledná úspěšnost klasifikace po dotrénování několika vybraných experimentů se snižováním bitové šířky operandů pro úlohu MNIST řešenou sítí BPN. Při těchto experimentech je zvolený datový typ použit jak pro dopředné šíření, tak pro uložení vah. Počáteční experiment s použitým datovým typem $FX < 8, 8 >$ s jeho mediánem úspěšnosti 96.6 % dosáhl dokonce o 0.2 % vyšší úspěšnosti než základní test s datovým typem float. Toto nepatrné zlepšení je pravděpodobně výsledkem lepšího zobecnění sítě díky nižší přesnosti použitého typu. U dalších experimentů již pozorujeme postupné snižování přesnosti. Za povšimnutí stojí především výsledky typů $FX < 4, 6 >$ a $FX < 6, 4 >$, které jsou oba 10 bitové, ale typ $FX < 4, 6 >$ má o 2.6 % vyšší úspěšnost. Je tak vidět důležitost počtu bitů za řádovou čárkou pro tuto konkrétní architekturu. Počet bitů před řádovou čárkou také nelze více snižovat, protože s jeho snižováním roste pravděpodobnost, že dojde k přetečení akumulátoru vnitřního potenciálu neuronu a tím ke snížení úspěšnosti klasifikace.



Obrázek 7.6: Úspěšnost klasifikace sítě BPN na úloze MNIST za použití datových typů s pevnou řádovou čárkou

Pro zafixování typu dopředného šíření sítě BPN jsem zvolil typ $FX < 4, 6 >$. Výsledky experimentů se snižováním bitové šířky typu vah vidíme na grafu 7.7, a můžeme pozorovat podobnost mezi tímto a předchozím grafem. Za povšimnutí opět stojí, že při porovnání úspěšnosti dvou typů s totožným celkovým počtem bitů si opět vede, tentokrát výrazněji, lépe ten s více bity za řádovou čárkou. Tedy s použitím typu vah $FX < 2, 4 >$ je úspěšnost sítě o 5.6 % vyšší než při použití typu $FX < 4, 2 >$. Medián úspěšnosti s typem $FX < 4, 6 >$ pro dopředné šíření a typem $FX < 2, 4 >$ pro uložení vah je 95.6 %.

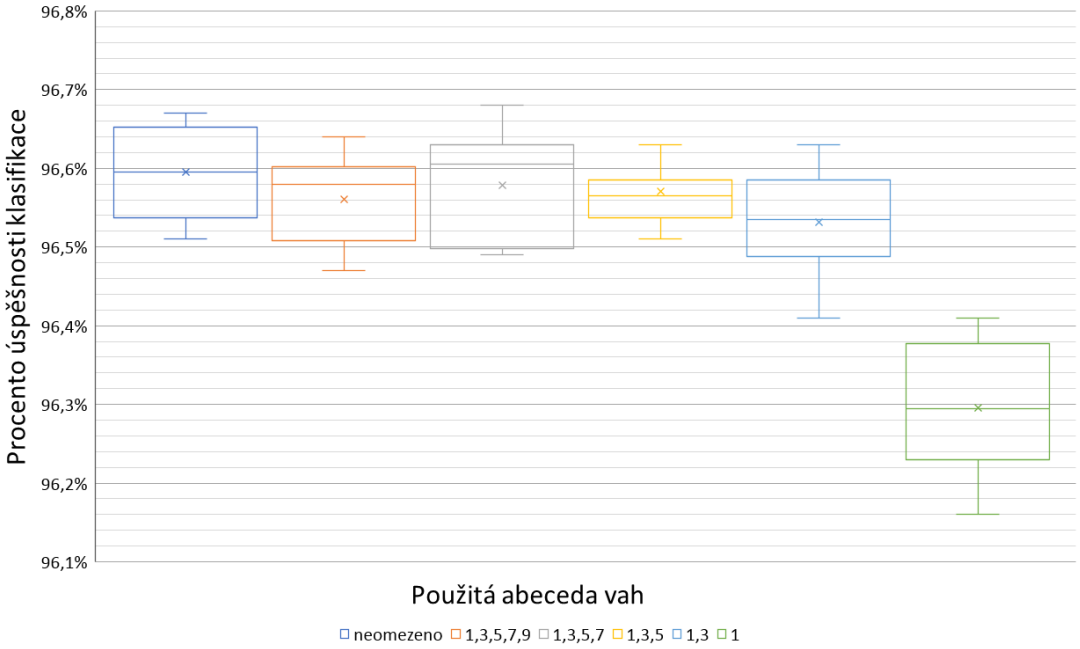


Obrázek 7.7: Úspěšnost klasifikace sítě BPN na úloze MNIST za použití datových typů s pevnou řádovou čárkou, dopředného typu $FX < 4, 6 >$

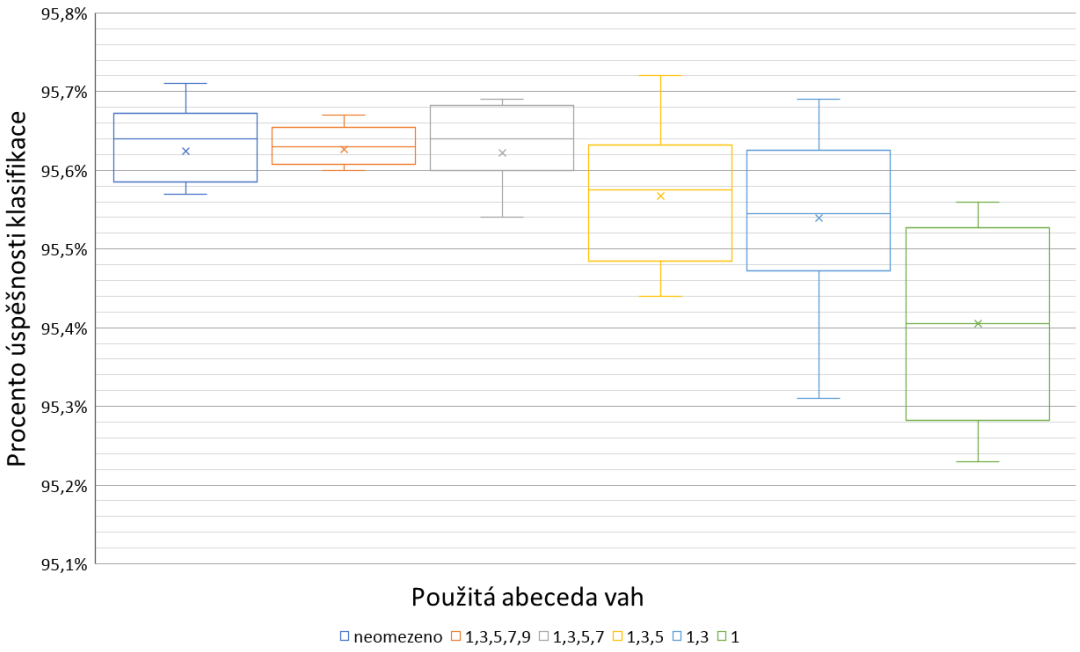
Počáteční experiment se snižováním bitové šířky operandů pro úlohu CIFAR-10 řešenou sítí CNN s použitým datovým typem $FX < 8, 8 >$ pro dopředné šíření i váhy opět dopadl lépe, než základní test s datovým typem float. Tentokrát datový typ $FX < 8, 8 >$ s jeho mediánem úspěšnosti 68.7 % dopadl o 1.5 % lépe, než datový typ float, což je ještě výraznější zlepšení, než u sítě BPN. I v tomto případě zlepšení přikládám lepšímu zobecnění sítě díky nižší přesnosti použitého typu. Bohužel se síť s dalším zmenšením bitové šířky operandů nedokázala vyrovnat a úspěšnost spadla na 10 %. Proto zde neuvádím samostatný graf tohoto experimentu a výsledek pro typ $FX < 8, 8 >$ je zobrazen až na grafu 7.10 spolu s výsledky experimentů násobení bez násobiček.

Násobení bez násobičky

Pro experimenty s násobením bez násobičky jsem kromě sítí z počátečního experimentu se snižováním bitové šířky operandů (sítě architektur BPN a CNN s typem $FX < 8, 8 >$ pro dopředné šíření i uložení vah) zvolil síť architektury BPN s typem $FX < 4, 6 >$ pro dopředné šíření a typem $FX < 2, 4 >$ pro uložení vah. Na grafu 7.8 jsou zobrazeny výsledky experimentu s násobením bez násobičky sítě BPN s typem $FX < 8, 8 >$ a to úspěšnosti klasifikace pro jednotlivé použité abecedy vah. Totéž, ale pro síť BPN s typem $FX < 4, 6 >$ pro dopředné šíření a typem $FX < 2, 4 >$ pro váhy, vidíme na grafu 7.9 a pro síť CNN s typem $FX < 8, 8 >$ na grafu 7.10.

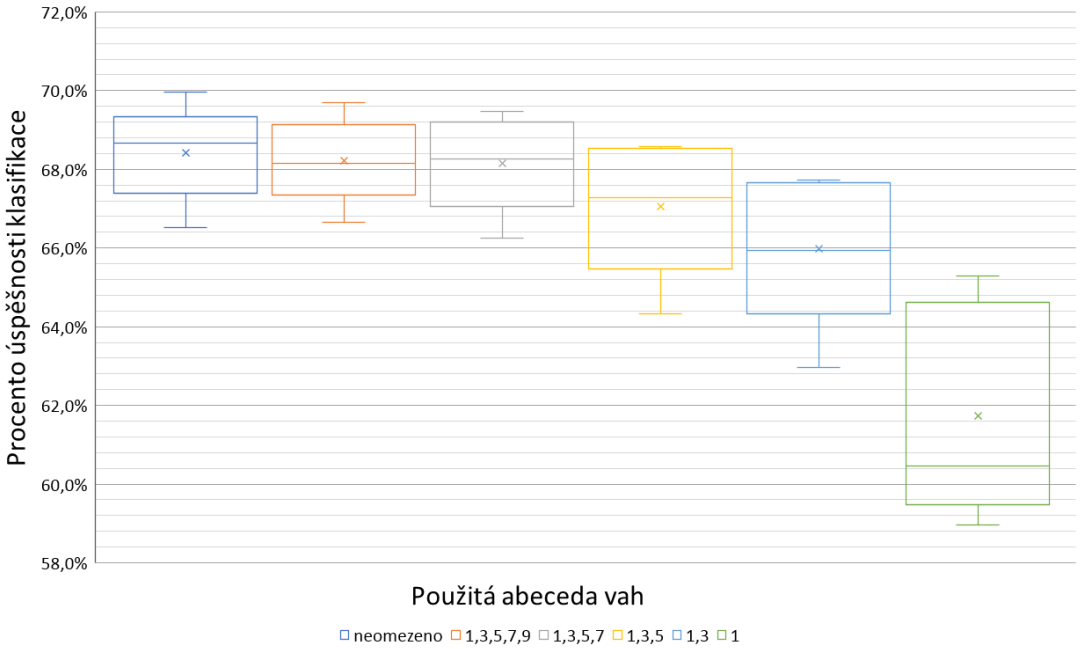


Obrázek 7.8: Úspěšnost klasifikace sítě BPN na úloze MNIST za použití násobení bez násobiček a typu $FX < 8, 8 >$



Obrázek 7.9: Úspěšnost klasifikace sítě BPN na úloze MNIST za použití násobení bez násobiček, dopředného typu $FX < 4, 6 >$ a typu vah $FX < 2, 4 >$

Tyto tři grafy vypadají velmi podobně, s postupným snižováním počtu prvků abecedy se procentuální úspěšnost klasifikace snižuje. Čím je počet prvků abecedy menší, tím výraznější je zhoršení úspěšnosti klasifikace. Přes podobný trend grafu je však rychlost snižování úspěšnosti sítě BPN a CNN rozdílná. Rozdíl úspěšnosti sítě BPN mezi neomezenou abecedou a abecedou obsahující pouze jeden prvek je 0.2 % (resp. 0.3 %), naopak rozdíl těchto úspěšností pro síť CNN je 8.2 %. Z toho vyplývá, že složitější problémy a rozsáhlejší sítě jsou citlivější na omezení oboru hodnot vah sítě.

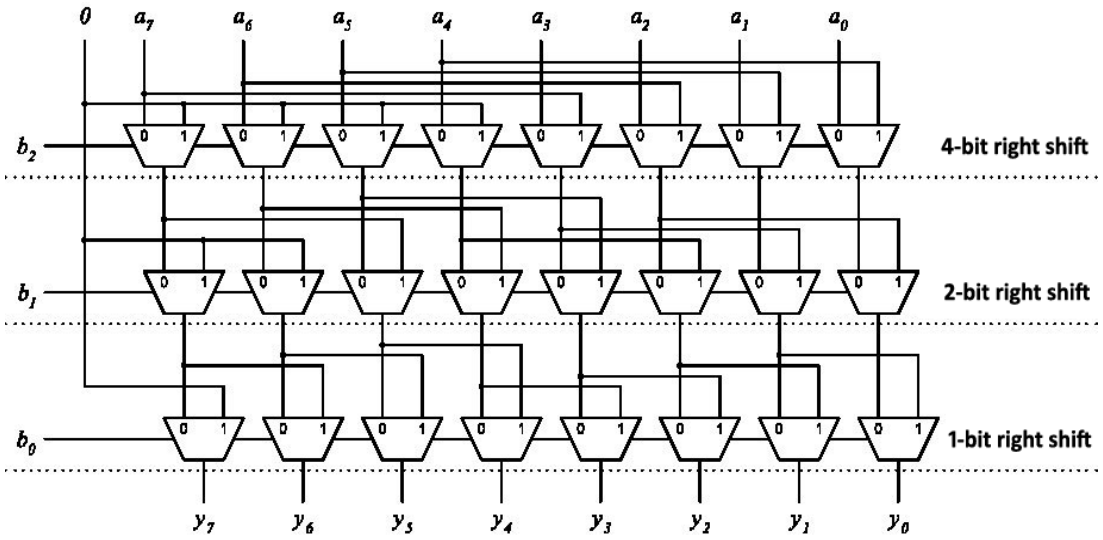


Obrázek 7.10: Úspěšnost klasifikace sítě CNN na úloze CIFAR-10 za použití násobení bez násobiček a typu $FX < 8, 8 >$

Kapitola 8

Odhad ceny hardwaru

V této kapitole provedu odhad ceny (velikosti na čipu a rychlosti) zjednodušeného násobení (násobení bez násobičky) a její porovnání s cenou konvenčního násobení. Odhad bude proveden pro proměnlivou šířku operandů (n) a demonstrován na výše použitém příkladu násobení s operandy o šířce 16 bitů. Cena konvenčního násobení je stanovena měřením hardwarové implementace operace násobení pro 45nm FreePDK technologii a byla získána nástrojem Synopsys Design Compiler. Toto měření provedl a hodnoty pro tuto práci poskytl Ing. Vojtěch Mrázek, Ph.D. Poskytnuté hodnoty jsou zaneseny v tabulce 8.1.



Obrázek 8.1: Schéma logaritmického válcového posouvače [27]

Pro odhad ceny násobení bez násobičky je nutné nejprve určit architekturu a cenu operace bitového posuvu. Dále tedy budu uvažovat použití logaritmického válcového posouvače (logarithmic barrel shifter). Schéma 8bitového logaritmického válcového posouvače můžeme vidět na obrázku 8.1. Jedná se o kombinační obvod, který za použití do vrstev uspořádaných multiplexorů provádí posuv n bitového čísla o 0 až $n-1$ bitů. Nejčastěji, a i v mém případě, jsou použity multiplexory $2-1$. Pro n bitové operandy je zpoždění tohoto posouvače a počet potřebných hradel (velikost na čipu):

$$\begin{aligned} zpoždění &= \log_2 n * zpoždění\_MUX &&\rightarrow \mathcal{O}(\log_2 n), \quad [27] \\ plocha &= n \log_2 n * plocha\_MUX &&\rightarrow \mathcal{O}(n \log_2 n), \quad [26] \end{aligned} \tag{8.1}$$

kde $zpoždění\_MUX = 0.06$ ns a $plocha\_MUX = 5.162$ µm^2. Podle předchozího vztahu má takový 16bitový posouvač zpoždění 0.24 ns a jeho velikost na čipu je 330 µm^2.

Obvod	Zpoždění [ns]	Plocha [µm^2]
Násobička 32b FP	6.12	6787
Násobička 16b FX	3.14	3020
Násobička 8b FX	1.35	688
Multiplexor 2-1	0.06	5.162

Tabulka 8.1: Cena obvodů pro 45nm FreePDK technologii

Dále určíme velikost dvou předem vytvořených tabulek a sice tabulky mezivýsledků a tabulky posuvů. Obsah obou tabulek je podrobněji popsán v kapitole 6.1. Velikost tabulek závisí na počtu bázových hodnot v abecedě vah, proto uvažujme neprázdnou abecedu vah, která vždy obsahuje bázovou hodnotu 1. Poté je počet bitů potřebných pro uložení tabulky mezivýsledku:

$$pocet\_bitu = n^2 * (pocet\_bazovych\_hodnot - 1) * n \tag{8.2}$$

Přesně vyjádřit velikost tabulky posuvů je problematické, protože by bylo třeba pro každou bázovou hodnotu z abecedy vah zjistit počet hodnot, které mohou být dosaženy levým posunem a jsou menší než největší zobrazitelné číslo. Počet takovýchto hodnot je však pro jakoukoliv bázovou hodnotu menší nebo rovno počtu takovýchto hodnot získaných pro hodnotu 1, pro kterou je tento počet roven n. Počet bitů potřebných pro uložení tabulky posuvů je tedy vždy menší nebo rovno:

$$pocet\_bitu <= n * pocet\_bazovych\_hodnot * n \tag{8.3}$$

8.1 Porovnání ceny hardwaru

Porovnání samotné operace násobení není závislé na velikosti abecedy vah, ani velikosti tabulek. Jak bylo ukázáno výše, velikost konvenční násobičky na čipu je 3020 µm^2 a její zpoždění je 3.14 ns. Válcový posouvač, který v operaci zjednodušeného násobení samotnou operaci provádí, na čipu zabere 330 µm^2 a jeho zpoždění je 0.24 ns. Tedy samotná operace násobení je provedena o 92 % rychleji a obsazená plocha na čipu je snížena o 89 %.

V reálné hardwarové implementaci však do času provedení zjednodušeného násobení musíme připočítat ještě časy přístupu do paměti. Pokud abeceda vah obsahuje pouze hodnotu 1, je proveden jeden přístup do paměti (zjištění posuvu), jinak jsou provedeny dva přístupy do paměti, a to zjištění mezivýsledku a posuvu. Doba přístupu do paměti závisí na typu a rychlosti paměti použité pro uložení těchto tabulek. Dále je potřeba do celkové ceny reálného hardwarového řešení započítat plochu na čipu a zpoždění řídicího obvodu, který na základě vstupu a váhy zajišťuje adresaci výše zmíněných tabulek. Návrh ani odhad ceny takového obvodu není cílem této práce.

Použijeme-li nejmenší možnou abecedu vah $\{1\}$, pak bude zjednodušené násobení provedeno pouze levým posuvem vstupní hodnoty a tabulka mezivýsledků tak nebude potřeba. Tabulka posuvů obsahuje šestnáct 16bitových hodnot a pro její uložení je tedy potřeba paměť o velikosti 256 bitů. To je oproti velikosti paměti potřebné pro uložení vah ($n * 80\ 000$ pro síť BPN) zanedbatelná hodnota. Zvolíme-li největší v této práci použitou abecedu $\{1, 3, 5, 7, 9\}$, pak budeme pro výpočet násobení potřebovat obě tabulky. Tabulka mezivýsledků obsahuje 1024 16bitových hodnot a pro její uložení je třeba paměť o velikosti 16 384 bitů. Tabulka posuvů obsahuje 80 16bitových hodnot a pro její uložení je třeba paměť o velikosti 1 280 bitů. I velikost obou těchto tabulek dohromady je v porovnání s velikostí vah sítě zanedbatelná. Součet velikostí obou tabulek nedosahuje ani 1.5 % velikosti paměti potřebné pro uložení vah.

Kapitola 9

Zhodnocení výsledků

V tabulce 9.1 vidíme vybrané výsledky testů jak pro architekturu BPN a úlohu MNIST, tak architekturu CNN a úlohu CIFAR-10. Tabulka obsahuje nastavení neuronové sítě, tedy použité datové typy, operaci násobení a zvolenou abecedu vah. Dále obsahuje experimentálně zjištěný medián úspěšnosti klasifikace sítě s tímto nastavením. Poslední skupinou, kterou tabulka obsahuje, jsou hodnoty ceny operace násobení spočítané na základě nastavení sítě. Cena operace násobení je složena ze zpoždění operace násobení [ns], její velikost na čipu [μm^2] a velikosti tabulek (mezivýsledků a posuvů) použitých při výpočtu. Odhad této ceny je proveden v předchozí kapitole.

Nastavení neuronové sítě					Medián	Cena operace násobení		
Architektura sítě	Dopředný datový typ	Datový typ vah	Operace násobení	Abeceda vah	úspěšnosti klasifikace	Zpoždění [ns]	Plocha [μm^2]	Velikost tabulek
BPN	float	float	standardní	-	96.42 %	6.12	6787	-
	FX<8,8>	FX<8,8>			96.60 %	3.14	3020	
	FX<4,6>	FX<2,4>			95.64 %	1.35	688	
	FX<8,8>	FX<8,8>	zjednodušené	1,3,5,7,9	96.58 %	0.24	330	17664b
				1,3,5	96.57 %			8960b
				1	96.30 %			256b
	FX<4,6>	FX<2,4>		1,3,5,7,9	95.63 %	0.18	124	4500b
				1,3,5	95.58 %			2300b
				1	95.41 %			100b
CNN	float	float	standardní	-	67.22 %	6.12	6787	-
	FX<8,8>	FX<8,8>			68.67 %	3.14	3020	
	FX<8,8>	FX<8,8>	zjednodušené	1,3,5,7,9	68.15 %	0.24	330	17664b
				1,3,5,7	68.27 %			13312b
				1,3,5	67.28 %			8960b
				1,3	65.93 %			4608b
				1	60.47 %			256b

Tabulka 9.1: Souhrné výsledky experimentů a výpočtů

Vidíme tedy, že rychlost a velikost na čipu samotné operace násobení (posouvače) závisí pouze na zvoleném datovém typu a velikost abecedy na ně nemá vliv. Pro datový typ $FX<8,8>$ je zjednodušené násobení oproti konvenčnímu zrychleno o 92 % a velikost na čipu je zmenšena o 89 %. Pro dopředný datový typ $FX<4,6>$ a typ vah $FX<2,4>$ je násobení zrychleno o 87 % a velikost je zmenšena o 82 %. Z tohoto pozorování plyne, že pro datový typ s větší bitovou šířkou dochází k výraznějšímu snížení ceny operace násobení.

Ke zpoždění a velikosti na čipu operace zjednodušeného násobení je však třeba ještě započítat zpoždění a plochu řídicího obvodu. Dále je třeba přičíst čas jednoho či dvou čtení z paměti (tabulky posuvů či mezivýsledků). Velikost abecedy ovlivňuje počet potřebných operací čtení a velikost tabulek, avšak jak bylo zmíněno dříve, tato velikost je v porovnání

s velikostí potřebnou pro uložení vah zanedbatelná. Pokud abeceda vah obsahuje pouze bázovou hodnotu 1, tak je potřeba pouze jedno čtení z tabulky posuvů. V tomto případě by bylo možné čtení z tabulky nahradit kombinační logickou sítí a ušetřit si tak čtení z paměti úplně.

Tabulka 9.2 obsahuje procento zhoršení mediánu úspěšnosti klasifikace, procento zrychlení a snížení velikosti na čipu zjednodušeného násobení v porovnání se stejnou sítí, avšak s použitím standardní operace násobení. Vidíme zde, že pro síť BPN je použití zjednodušeného násobení velice výhodné, a to i pokud je zvolena abeceda obsahující pouze hodnotu 1, kdy se zhoršení mediánu úspěšnosti pohybuje okolo 0.3 %. Se zvyšující se složitostí řešeného úkolu se snižuje výhodnost použití zjednodušeného násobení. Vidíme, že pro síť CNN jsou snížení úspěšnosti klasifikace znatelnější a pro abecedu obsahující pouze hodnotu 1 je zhoršení úspěšnosti klasifikace téměř 12 %, což znemožňuje její praktické použití.

Nastavení neuronové sítě				Zhoršení úspěšnosti klasifikace	Cena operace násobení	
Architektura sítě	Dopředný datový typ	Datový typ vah	Abeceda vah		Zrychlení	Zmenšení plochy
BPN	FX<8,8>	FX<8,8>	1,3,5,7,9	0.02 %	92.36 %	89.06 %
			1,3,5	0.03 %		
			1	0.31 %		
	FX<4,6>	FX<2,4>	1,3,5,7,9	0.01 %	86.67 %	81.99%
			1,3,5	0.06 %		
			1	0.24 %		
CNN	FX<8,8>	FX<8,8>	1,3,5,7,9	0.76 %	92.36 %	89.06 %
			1,3,5,7	0.58 %		
			1,3,5	2.02 %		
			1,3	3.99 %		
			1	11.94 %		

Tabulka 9.2: Porovnání úspěšnosti a ceny zjednodušeného násobení

Kapitola 10

Závěr

V rámci této práce jsme se seznámili s problematikou klasických i konvolučních neuronových sítí. Je zde popsána jejich funkčnost i hlavní problém jejich masivnějšího rozšíření ve vestavěných systémech. Dále jsme si nastínili problematiku hardwarového násobení čísel a možnosti jeho optimalizace. Představili jsme si volně dostupné nástroje pro práci s neuronovými sítěmi a datové sady pro testování jejich činnosti.

Jako nástroj pro práci s neuronovými sítěmi byla zvolena knihovna TypeCNN viz 5.3. Pro zjednodušení násobení byla zvolena technika násobení bez násobičky popsaná v kapitole 4.3. Byl proveden návrh a implementace modulu založeného na této technice. Ten byl integrován do knihovny TypeCNN. Navrhl jsem testovací architektury, metodiku testování a experimentů. Následně jsem provedl odhad ceny hardwarového provedení konvenčního i zjednodušeného násobení.

Po provedení experimentů a jejich zhodnocení jsem došel k závěru, že použití této techniky zjednodušeného násobení je velice výhodné pro jednodušší problémy a menší neuronové sítě. Pro síť BPN a datový typ $FX < 8, 8 >$ se standardní operací násobení je medián úspěšnosti klasifikace 96.6 %. Při použití operace zjednodušeného násobení je rychlost výpočtu zvýšena o 92 % a velikost na čipu zmenšena o 89 % za cenu snížení přesnosti klasifikace maximálně o 0.3 %. Pro síť CNN a datový typ $FX < 8, 8 >$ se standardní operací násobení je medián úspěšnosti klasifikace 68.7 %. Úspora místa a zvýšení rychlosti je totožné jako u sítě BPN, avšak přesnost je snížena až o 12 %. Takové snížení přesnosti je pro praktické použití nepřípustné.

Možným rozšířením této práce je návrh a implementace optimalizačního algoritmu, který by umožnil učení neuronové sítě používající tuto techniku násobení bez násobičky.

Literatura

[1] IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985*, 1985, doi:10.1109/IEEESTD.1985.82928.

[2] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, [Online; navštíveno 11.11.2018], Software available from tensorflow.org.
URL https://www.tensorflow.org/

[3] Benenson, R.: Classification datasets results. GitHub, 2016, [Online; navštíveno 11.11.2018].
URL https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[4] Bolješik, M.: *Generátor aritmetických obvodů*. Diplomová práce, FIT VUT v Brně, 2015, [Online; navštíveno 11.11.2018].
URL https://www.vutbr.cz/studenti/zav-prace/detail/88419

[5] Bulej, L.: Aritmetika v počítači. MFF UK v Praze, 2011, [Online; navštíveno 11.11.2018].
URL http://d3s.mff.cuni.cz/teaching/principles_of_computers/ar-20112012/04-aritmetika.pdf

[6] Bölük, C.: SimpleCNN. GitHub, 2016, [Online; navštíveno 11.11.2018].
URL https://github.com/can1357/simple_cnn

[7] Chollet, F.; aj.: Keras. 2015, [Online; navštíveno 11.11.2018].
URL https://keras.io

[8] Daubner, L.: *Deep learning*. Diplomová práce, FI MUNI v Brně, 2015, [Online; navštíveno 11.11.2018].
URL https://is.muni.cz/th/410034/fi_b/thesis.pdf

[9] Dvořáček, P.: *Evoluční návrh pro aproximaci obvodů*. Diplomová práce, FIT VUT v Brně, 2015, [Online; navštíveno 11.11.2018].
URL https://www.vutbr.cz/studenti/zav-prace/detail/88396

[10] Flídr, M.: Zobrazení čísel v počítači. Západočeská univerzita v Plzni, 2007, [Online; navštíveno 11.11.2018].
URL https://vendulka.zcu.cz/Vyuka/POS/slides/2007/zobrazeni_cisel.pdf

[11] Gysel, P.; Pimentel, J.; Motamedi, M.; aj.: Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks. *IEEE*

Transactions on Neural Networks and Learning Systems, ročník 29, č. 11, 2018: s. 5784–5789, ISSN 2162-237X, doi:10.1109/TNNLS.2018.2808319.

[12] Hradiš, M.: Konvoluční neuronové sítě. FIT VUT v Brně, 2015, [Online; navštíveno 11.11.2018].
URL https://openalt.cz/2015/data/Michal%20Hradis%20-%20Konvolucni%20neuronove%20site.pdf

[13] Karpathy, A.: Neural Networks Part 1. [Online; navštíveno 11.11.2018].
URL https://cs231n.github.io/neural-networks-1/

[14] Krajčovičová, M.: *Konvoluční neuronová síť pro zpracování obrazu*. Diplomová práce, FEKT VUT v Brně, 2015, [Online; navštíveno 11.11.2018].
URL https://www.vutbr.cz/studenti/zav-prace?zp_id=85389

[15] Kriesel, D.: *A Brief Introduction to Neural Networks*. 2007, [Online; navštíveno 11.11.2018].
URL http://www.dkriesel.com/en/science/neural_networks

[16] Krizhevsky, A.: *Learning multiple layers of features from tiny images*. 2009, [Online; navštíveno 11.11.2018].
URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[17] Krizhevsky, A.; Nair, V.; Hinton, G.: The CIFAR-10 dataset. 2009, [Online; navštíveno 11.11.2018].
URL https://www.cs.toronto.edu/~kriz/cifar.html

[18] Krizhevsky, A.; Sutskever, I.; Hinton, G.: ImageNet Classification with Deep Convolutional Neural Networks. 2012, [Online; navštíveno 11.11.2018].
URL http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

[19] LeCun, Y.; Cortes, C.; Burges, C. J.: THE MNIST DATABASE of handwritten digits. 1998, [Online; navštíveno 11.11.2018].
URL http://yann.lecun.com/exdb/mnist/

[20] Maďa, P.; Fontana, J.: Regulační mechanismy 2: Nervová regulace. [Online; navštíveno 11.11.2018].
URL http://fblt.cz/skripta/regulacni-mechanismy-2-nervova-regulace/4-synapticky-prenos/

[21] McMillan, L.: Floating-Point Arithmetic. UNC Computational Genetics, 2015, [Online; navštíveno 11.11.2018].
URL http://www.csbio.unc.edu/mcmillan/Comp411F15/Lecture13.pdf

[22] Mrazek, V.; Sarwar, S. S.; Sekanina, L.; aj.: Design of power-efficient approximate multipliers for approximate artificial neural networks. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, ISSN 1558-2434, s. 1–7, doi:10.1145/2966986.2967021.

[23] Neudert, L.: Historie neuronových počítačů a sítí. FI MUNI v Brně, 2000, [Online; navštíveno 11.11.2018].
URL https://www.fi.muni.cz/usr/jkucera/pv109/2000/xneudert.html

[24] Office of Communications: Neuroscience. 2018, [Online; navštíveno 11.11.2018].
URL https://www.nichd.nih.gov/health/topics/neuro/conditioninfo/parts

[25] Rek, P.: *Knihovna pro návrh konvolučních neuronových sítí*. Diplomová práce, FIT VUT v Brně, 2018, [Online; navštíveno 11.11.2018].
URL https://www.vutbr.cz/studenti/zav-prace/detail/114513

[26] S. B. Jondhale, S. S. P., T. S. Mulla: Design and Implementation of 8 Bit Barrel Shifter Using 2:1 Multiplexer in Verilog. *Journal of Advances in Science and Technology*, ročník 12, č. 25, 2016: s. 296–299, ISSN 2230-9659, [Online; navštíveno 15.05.2019].
URL http://www.ignited.in/File_upload/18300_88237588.pdf

[27] Sachan, P.; Katiyar, A.; Didal, A.; aj.: BARREL SHIFTER. *International Journal of Science, Engineering and Technology*, ročník 2, č. 7, 2014: s. 1434–1440, ISSN 2348-4098, [Online; navštíveno 15.05.2019].
URL http://www.ijset.in/wp-content/uploads/2014/09/IJSET.0920140903.1011.0909_PRAGATI-1434-1440.pdf

[28] Sarwar, S. S.; Venkataramani, S.; Ankit, A.; aj.: Energy-Efficient Neural Computing with Approximate Multipliers. *J. Emerg. Technol. Comput. Syst.*, 2018: s. 16:1–16:23, ISSN 1550-4832, doi:10.1145/3097264, [Online; navštíveno 11.11.2018].
URL http://doi.acm.org/10.1145/3097264

[29] Sarwar, S. S.; Venkataramani, S.; Raghunathan, A.; aj.: Multiplier-less Artificial Neurons Exploiting Error Resiliency for Energy-Efficient Neural Computing. *CoRR*, ročník abs/1602.08557, 2016, [Online; navštíveno 11.11.2018], 1602.08557.
URL http://arxiv.org/abs/1602.08557

[30] Schmid, M.: *Konvoluční neuronové sítě a jejich implementace*. Diplomová práce, MFF UK v Praze, 2011, [Online; navštíveno 11.11.2018].
URL https://is.cuni.cz/webapps/zzp/detail/93451

[31] Sekanina, L.: Studijní materiály kurzu Návrh počítačových systémů. FIT VUT v Brně, 2008, [Online; navštíveno 11.11.2018].
URL http://pub.eyim.net/ziraficka/inp/inp09mult.pdf

[32] So, H.: Introduction to Fixed Point Number Representation. University of California, Berkeley, 2006, [Online; navštíveno 11.11.2018].
URL http://www-inst.eecs.berkeley.edu/~cs61c/sp06/handout/fixedpt.html

[33] Steppan, J.: Sample images from MNIST test dataset. Wikipedia, 2017, [Online; navštíveno 11.11.2018].
URL https://en.wikipedia.org/wiki/MNIST_database#/media/File:MnistExamples.png

[34] Volná, E.: *Základy SoftComputingu*. Ostravská univerzita v Ostravě, 2012, [Online; navštíveno 11.11.2018].
URL http://www1.osu.cz/~volna/Zaklady_softcomputingu_skripta.pdf

[35] Yazdanbakhsh, A.; Thwaites, B.; Park, J.; aj.: Methodical Approximate Hardware Design and Reuse. Georgia Institute of Technology, 2014, [Online; navštíveno

11.11.2018].
URL https://www.cc.gatech.edu/~hadi/doc/paper/2014-wacas-hdl.pdf

[36] Zbořil, F.: Studijní materiály kurzu Soft Computing. FIT VUT v Brně, [Online; navštíveno 11.11.2018].
URL http://www.fit.vutbr.cz/study/courses/SFC/private/

Příloha A

Obsah CD

CD obsahuje následující soubory a adresáře:

`juhanak_DP.pdf`	- elektronická verze této práce
`text/`	- zdrojový kód v LaTeXu pro generování této práce
`TypeCNN/`	- celá knihovna TypeCNN
`TypeCNN/juhanak/`	- programy a scripty použité při testování
`TypeCNN/juhanak/*_res`	- natrénované sítě a výsledky testování
`TypeCNN/src/Utils/MultlessMult.cpp`	- implementace násobení bez násobičky

Příloha B

README

Pro použití zjednodušeného násobení (navrženého a implementovaného v této práci) v knihovně TypeCNN je potřeba:

1. `Získat knihovnu TypeCNN -` je přiložena na CD, jež je součástí této práce, popřípadě dostupná z `http://github.com/rekpet/TypeCNN`
2. `Nastavit makro` $MULTLESS\_MULT$ `při překladu`
3. `Nastavit systémovou proměnnou` $ALPHABET\_VALS$ `-` obsahuje jednotlivé bázové hodnoty oddělené čárkou (např. $ALPHABET\_VALS = "1, 3, 5"$)

Předpokladem použití je stroj s operačním systémem Linux schopný překladu C++14. Pokud byla knihovna získána z CD, je možné použít již přeložené (na školním serveru Merlin) aplikace $TypeCNN/juhanak/bpn\_mlm$, nebo $TypeCNN/juhanak/cnn\_mlm$ a tedy stačí nastavit systémovou proměnnou $ALPHABET\_VALS$.