VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE NEURONOVÝCH SÍTÍ V FPGA

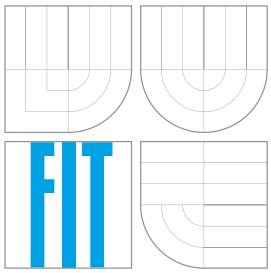
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KRČMA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE NEURONOVÝCH SÍTÍ V FPGA

ACCELERATION OF THE NEURAL NETWORKS IN THE FPGAS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KRČMA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KAŠTIL

BRNO 2014

Abstrakt

Tato práce se zabývá metodikami učení struktur FPNN. Zaměřuje se především na způsoby přímého převodu naučených neuronových sítí na FPNN, což je výhodné v situacích, kdy nejsou k dispozici trénovací data.

Abstract

This thesis deals with a training of the FPNN structures. It focuses on the ways of direct conversion of the pretrained artificial neural networks to FPNNs. This is useful when original training data set is not reachable.

Klíčová slova

neuronové sítě, FPNA, FPNN, optimalizace, strojové učení, FPGA.

Keywords

neual networks, FPNA, FPNN, optimalisation, machine learning, FPGA

Citace

Martin Krčma: Akcelerace neuronových sítí v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2014

Akcelerace neuronových sítí v FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Kaštila

.....
Martin Krčma
26. května 2014

Poděkování

Rád bych poděkoval Ing. Janu Kaštilovi za inspirativní odborné vedení, přátelský přístup, cenné rady, ochotu pomoci a poradit, a za poskytnutí přístupu k literatuře a prostředkům potřebným k vypracováním této práce. Také chci poděkovat své rodině za stálou a neochvějnou podporu.

Obsah

Kapitola 1

Úvod

V roce 1943 Warren McCulloch a Walter Pitts představili v článku *A Logical Calculus of Ideas Immanent in Nervous Activity* [17] matematický popis neuronu jako logického přepínače a dokázali, že sestavením sítí z takových neuronů je možné realizovat libovolné operace výrokové logiky. V roce 1949 definoval Donald Hebb ve své knize *The Organization of Behavior* [11] první učící algoritmus neuronových sítí. V roce 1957 Frank Rosenblat zobecnil model neuronu na *perceptron* [26], který počítal s reálnými čísly, zároveň definoval učící algoritmus, který v konečeném čase najde odpovídající váhový vektor nezávisle na počáteční kofiguraci. Na základě tohoto sestavil první neuropočítač *Mark Perceptron I*, který dovedl rozpoznávat znaky. Úspěšná prezentace tohoto počítače k neuronovým sítím přitáhla první vlnu zájmu. Na přelomu 50. a 60. let Karl Steinbuch vyvinul model *bibární asociativní sítě* [30]. V roce 1969 Marvin Minsky a Seymour Papert ve své knize *Perceptrons* [19] poukázali na nemožnost realizovat logickou funkci XOR jediným perceptronem. Tuto funkci sice bylo možné realizovat dvouvrtsvou sítí se třemi perceptrony, ale v té době nebyl znám algoritmus učení vícevrtsvých sítí. Z toho autoři nesprávně odvodili nemožnost takového algoritmu. To, spolu s vyčerpáním většiny nápadů, vedlo k úpadku zájmu o neuronové sítě a odliv financí z výzkumu tohoto oboru. Ten však v tichosti pokračoval a v 80. letech se znovu dostal do popředí. V roce 1982 představil John Hopfield nový model neuronové sítě fungující jako autoasociativní paměť [13]. Ve stejném roce vznikl také model samoorganizující se Kohonenovy sítě [14]. V roce 1986 David Rumelhart, Geo Rey Hinton a James McClelland publikovali [28] důležitý algoritmus učení založený na zpětném šíření chyby - *backpropagation*. V roce 1987 se v San Diegu konala první velká konference, *IEEE International Conference on Neural Networks*, výhradně zaměřená na neuronové sítě, na níž byla založena mezinárodní společnost pro výzkum neuronových sítí *INNS (International Neural Network Society)*. Od té doby se neuronové sítě stále těší zájmu odborné veřejnosti.

Tato práce navazuje na bakalářskou práci Akcelerace neuronových sítí v FPGA [15]. Ta pojednávala o aproximaci neuronových sítí pomocí struktur konceptu FPNA [6], jejich VHDL implementaci a výsledky běhu této implemnetace na hradlových polích FPGA. Tato diplomová práce se více zaměřuje na samotný koncept FPNA. Zabývá se učením FPNN a především přímým převodem neuronových sítí na FPNN. Kapitola 2 představuje úvod do neuronových sítí. Kapitola 2.3 je úvod do konceptu FPNA. Kapitola 4 pojednává o balíku aplikací PyFPNN, který byl představen již v bakalářské práci a od té doby se výrazně rozrostl. Kapitola 5 se týká samotného přímého převodu neuronových sítí na FPNN. Kapitola 8 shrnuje dosažené výsledky a navrhuje směr budoucího rozvoje práce.

Kapitola 2

Teorie

2.1 Úvod do neuronových sítí

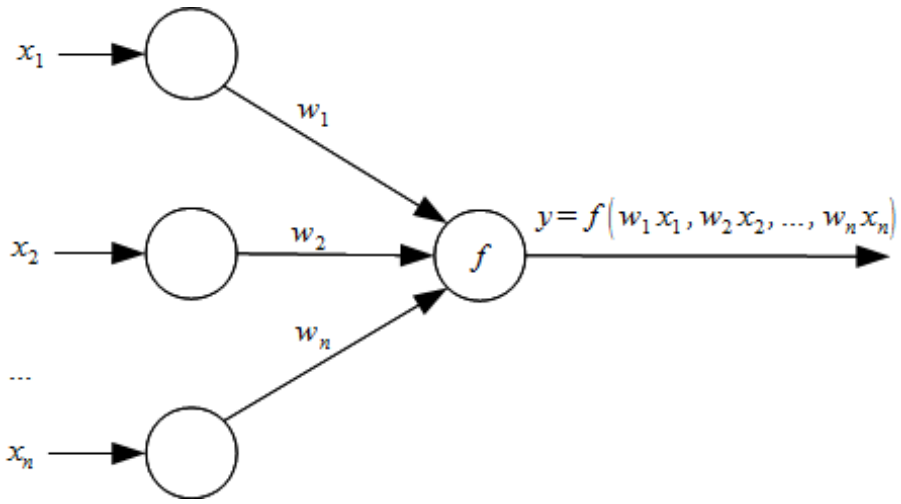
Neuronové sítě [20] se, podobně jako mozek, skládají z neuronů a spojů (synapsí), které je mezi sebou propojují. Topologie propojení neuronů synapsemi jsou pak definovány jednotlivými modely neuronových sítí, kterých existuje více.

2.1.1 Neuron

Na obrázku 2.1 představujícím obecný model neuronu, má každý neuron n vstupů x, které se s příslušnými vahami w uplatňují jako vstup funkce f, jejíž výstup je zároveň výstupem neuronu y. Ze vstupních impulsů x a vah w se počítá vážený součet, kterému se často říká *potenciál* (v angličtině se často označuje *net*). Ten je pak vstupem funkce f, které se říká nejčastěji *aktivační* nebo *přenosová* funkce. Definováno matematicky:

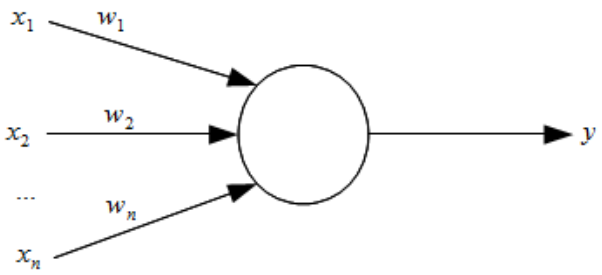
$$net = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

$$y = f(net) \tag{2.2}$$



Obrázek 2.1: Obecná struktura neuronu

Obrázek 2.2 zobrazuje schematickou značku neuronu používanou ve zbytku této práce.



Obrázek 2.2: Vnější pohled na neuron

2.1.2 Aktivační funkce

Jak bylo řečeno v sekci 2.1.1, základním prvkem výpočtů v neuronu je aktivační funkce, která se počítá z potenciálu a vytváří výstup neuronu. Konkrétní podoba této funkce se odvíjí od konkrétního modelu neuronové sítě, ale obecně se jedná o rostoucí, spojitou, derivovatelnou funkci jedné proměnné. Používají se také funkce, které nejsou spojité a na celém definičním oboru derivovatelné. Jejich problémem ale je, že k naučení sítě požívající tyto funkce nemůže být použit algoritmus uvedený v sekci 2.2.1. Často používanými aktivačními funkcemi jsou například funkce sigmoid (2.3), unipolární skoková funkce (2.4), bipolární skoková funkce (2.5), hyperbolický tangets (2.6), které jsou definovány následovně:

$$\text{sigmoid}(x) = \frac{1}{1+\epsilon^{-\theta x}} \tag{2.3}$$

$$\text{unipolar\_step}(x) = \begin{cases} 0 & \text{pro } x \leq 0 \\ 1 & \text{pro } x > 0 \end{cases} \tag{2.4}$$

$$\text{bipolar\_step}(x) = \begin{cases} -1 & \text{pro } x \leq 0 \\ 1 & \text{pro } x > 0 \end{cases} \tag{2.5}$$

$$\tanh(x) = \frac{2}{1+\epsilon^{-2x}} - 1 \tag{2.6}$$

$$\tag{2.7}$$

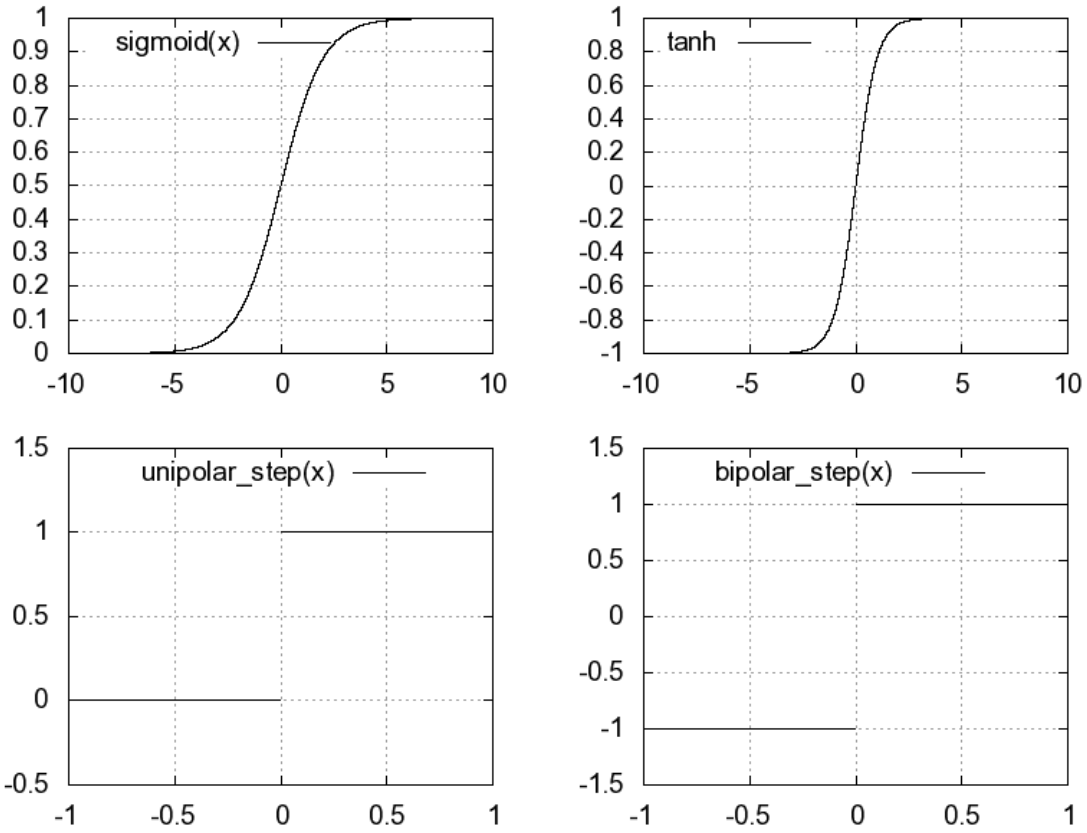
Průběhy těchto funkcí jsou znázorněny na obrázku 2.3

2.1.3 Práh

Práh je nástroj, který zvyšuje sílu neuronových sítí tím, že umožňuje posunout průběh aktivační funkce po ose x doprava nebo doleva. To je dosaženo přičtením hodnoty prahu k potenciálu neuronu, jak ukazuje rovnice (2.8), kde θ je práh.

$$y = f(net + \theta) \tag{2.8}$$

Pro implementaci prahu se často používá důmyslný trik, spočívající v přidání synapse s vahou rovnou hodnotě prahu vedoucí od virtuálního neuronu, jehož výstup je trvale roven 1. Tím je jednoduše zajištěno započítání prahu a zároveň je tím umožněno naučit prahy stejně jako váhy.



Obrázek 2.3: Grafy některých aktivačních funkcí

2.1.4 Vrstvy

Neurony jsou v neuronových sítích obyčejně organizovány do vrstev. Těchto vrstev může být různý počet a mohou obsahovat různé počty neuronů. Počty vrstev a neuronů v nich, propojení uvnitř vrstev a mezi vrstvami pak definují jednotlivé modely neuronových sítí. Neuronová síť pak může vypadat třeba jako na obrázku 2.4.

2.1.5 Typy a dělení neuronových sítí

Neuronové sítě můžeme dělit z různých hledisek. Prvním hlediskem může být počet vrstev - neuronové sítě sestávající z jedné vrstvy nazýváme *jednovrstvé*, ostatní *vrstvené*. Druhým hlediskem může být směr šíření signálů v síti - sítě, v nichž se signály šíří jen do následujících vrstev nazýváme *nerekuretní*, zatímco sítě, v nichž se signály vracejí do stejné vrstvy nebo předchozích vrstev nazýváme *rekuretní*. Dalším hlediskem může být způsob učení sítě - sítě, v jejichž procesu učení vystupuje nějaká entita hrající roli učitele nazýváme *sítě s učitelem*, sítě, které učitele nemají nazýváme *sítě bez učitele*, které se v některých případech nazývají *samoorganizující*.

2.1.6 Učení neuronových sítí

Učení neuronových sítí obecně pobíhá na principu modifikace vah synapsí (které jsou zpočátku nastaveny náhodně) - jak vyplývá ze sekce 2.1.1, modifikace vah způsobí při

stejném vstupu změnu výstupu sítě, takže při učení jsou váhy obecně nějakým způsobem modifikovány tak dlouho, dokud výstup neuronové sítě nespňuje nějaký požadavek na něj kladený. Toto probíhá na základě různých algoritmů, existují ale dva základní modely učení neuronových sítí.

Učení s učitelem probíhá za pomoci nějaké entity, nazývané učitel, která neuronové síti předkládá příklady z tzv. trénovací množiny, což je množina vstupních dat, na něž zná učitel požadovaný výstup, a hodnotí výstup sítě. Na základě tohoto hodnocení jsou pak modifikovány váhy.

Učení bez učitele neprobíhá za pomoci žádné entity, která by na učení neuronové sítě dohlížela. Není dopředu definován žádný cílový výstup sítě. Síť se učí sama na základě souvislostí v datech trénovací množiny, podle kterých se sama organizuje. Tomuto způsobu učení se tak často říká *samoorganizace*.

2.2 Modely neuronových sítí

Existuje mnoho různých modelů neuronových sítí, mající různé vlastnosti a určení. Tyto modely definují strukturu sítě z hlediska vrstev neuronů a jejich propojení, určují aktivační fukci a další aspekty relevantní pro dané modely. V tomto textu se podíváme jen na několik základních modelů.

2.2.1 Backpropagation model

Backpropagation model [20] je základní a nejpoužívanější model neuronových sítí. Jedná se o nerekuretní vrstvenou síť s jednou vstupní vrstvou, jednou výstupní vrstvou a jednou nebo více skrytými vrstvami. Na vstupy neuronů jsou přivedeny výstupy všech neuronů předchozí vsrtvy (viz. obrázek 2.4). Počet neuronů ve vstupní a výstupní vrstvě je dán velikostí vstupního a výstupního vektoru, počty neuronů ve skrytých vrstvách a počet skrytých vrstev může být různý. Backpropagation model využívá učení s učitelem, který síti předkládá příklady a kontroluje výstup sítě. Na základě odchylky od odčekávaného výstupu jsou pak modifikovány váhy. Modifikace vah se šíří od výstupní vrstvy přes skryté vrtsvy směrem k vrstvě vstupní, odtud se tento model nazývá *backpropagation* (zpětná propagace).

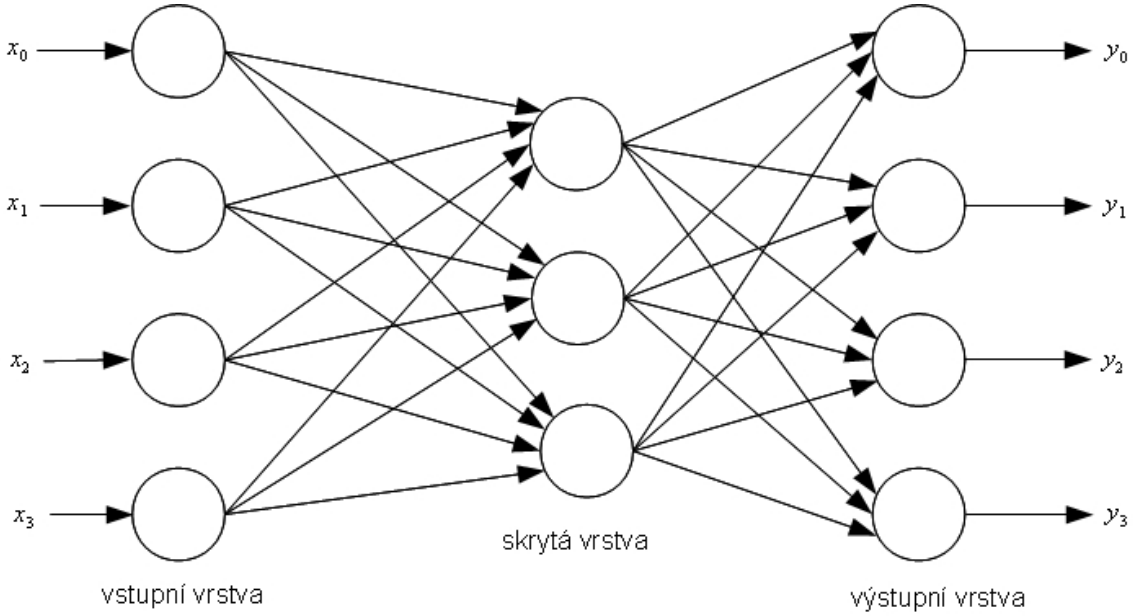
Učení v backpropagation modelu

Jak bylo řečeno v sekci 2.2.1, sítě tohoto modelu se učí za pomoci učitele, který síti předkládá j trénovacích vektorů x. Z výstupu sítě y_j a očekávaného výstupu sítě t_j je pak vypočtena chyba E, definovaná pomocí součtu čtverců všech odchylek:

$$E = \frac{1}{2}\sum_j (t_j - y_j)^2 \tag{2.9}$$

Při učení je pak nutno tuto odchylku snižovat, je tedy nutné, považujeme-li E za funkci vah, hledat minimum této chybové funkce. K tomu se používají různé metody jako např. simulované žíhání nebo častěji gradientní metody. Použitím gradientní metody odvodíme vzorce [20] pro modifikaci vah ve výstupní a skrytých vrstvách:

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} + \Delta w_{ij}^{(n)} \tag{2.10}$$



Obrázek 2.4: Backpropagation model

Kde:

$$\Delta w_{ij}^{(n)} = \eta \delta_j o_i + \alpha \Delta w_{ij}^{(n-1)} \tag{2.11}$$

Rovnice (2.10) popisuje změnu vah přičtením odchylky, která se vypočítá podle rovnice (2.11). V této rovnici η a α jsou kladná čísla, pro jejichž výpočet není definován vzorec. Proto se jejich hodnota stanoví libovolně a v průběhu učení se může měnit pro dosažení lepší konvergence.

Za předpokladu, že jako aktivační funkci používáme funkci sigmoid (2.3) s prahem, můžeme δ_j definovat [20] pro výstupní a skryté vrstvy takto:
Výstupní vrstva:

$$\delta_j = (t_j - o_j)\, o_j\, (1 - o_j) \tag{2.12}$$

Kde o_j je výstup výstupního neuronu j a t_j je požadovaný výstup tohoto neuronu.
Skryté vrstvy:

$$\delta_j = o_j\, (1 - o_j) \sum_k \delta_k w_{jk} \tag{2.13}$$

Kde bylo δ_k vypočteno v rovnici (2.12) a suma $\sum_k$ byla vypočítána přes neurony ve výstupní vrstvě.

2.2.2 Jiné modely

Kromě backpropagation modelu existuje řada dalších modelů neuronových sítí. *Shortcut perceptron* je speciálním případem backpropagation modelu, ve kterém jsou neurony propojeny se všemi neurony ve všech předchozích vrstvách, nikoliv jen s tou předchozí. Dalším známým modelem jsou *Hopfieldovy sítě* [22] [13], jednovrsvé rekurentní sítě používané například jako asociativní paměť. Dalším zajímavým zástupcem jsou *Kohonenovy sítě* [22] [14], což jsou jednovrtsvé plně propojené sítě vhodné ke kategorizaci.

2.3 Koncept FPNA

V této sekci představíme koncept aproximace neuronových sítí pomocí FPNA, který je pro tuto práci stěžejní. Na tomto konceptu je založen náš způsob implementace neuronových sítí v FPGA a na způsoby jeho vytvoření a naučení se celá tato práce zaměřuje.

2.3.1 FPNA

FPNA (Field Programmable Neural Array) [3] je koncept zjednodušující topologii neuronové sítě sloučením několika synapsí a jejich serializací. Vytváří tak síť s menším počtem synapsí, ale stejným počtem neuronů, aproximující původní neuronovou síť. Na FPNA můžeme nahlížet jako na orientovaný graf (N, E), kde N je uspořádaná množina uzlů a E je množina orientovaných hran:

N - uspořádaná množina uzlů – aktivátorů
E - množina orientovaných hran – spojů
Každý uzel má množinu předchůdců: $Pred(n) = \{p \in N, (p, n) \in E\}$
Každý uzel má množinu následníků: $Succ(n) = \{s \in N, (n, s) \in E\}$
Množina vstupních uzlů: $N_i = \{n \in N, Pred(n) = \emptyset\}$
Každý spoj $(p, n) \in E$ má definován afinní operátor: $\alpha_{(p,n)} = W_n(p)x + T_n(p)$
Každý nevstupní aktivátor $n \in N$ má definován iterační operátor i_n (pro sběr potenciálu)
Každý nevstupní aktivátor $n \in N$ má definován funkční operátor f_n (aktivační funkci)

Jak je uvedeno v definici výše, vrcholy grafu se nazývají aktivátory. Aktivátory aplikují na množinu svých vstupů přenosovou funkci. Aktivátory tedy reprezentují neurony. Hrany v grafu FPNA se pak nazývají spoje, které provádějí affinní trasformaci svých vstupů. Jak je z definice této transformace vidět, mohou spoje provádět (a provádějí) násobení vstupů vahami. Násobení vahami se tedy neprovádí v aktivátorech (neuronech), ale ve spojích a aktivátory tedy provádějí nikoliv vážený, ale obyčejný součet vstupních hodnot (za použití iteračního operátoru i_n), na nějž pak aplikují přenosovou funkci (funkční operátor f_n). Výrazným rozdílem mezi synapsemi a FPNA spoji je ten, že spoje neaproximují jednotlivou každou synapsi, ale aproximují nějakou část nějakého počtu synapsí, mezi dvěma aktivátory tedy může ležet několik spojů za sebou a násobění váhou se provádí po částech ve všech. Jak je vidět, FPNA nedefinuje konkrétní propojení, ani konkrétní hodnoty některých parametrů, které jsou potřeba pro konstrukci konkrétní sítě. FPNA tedy popisuje nikoliv jednu neurální síť, ale celou skupinu neurálních sítí. Pro popis konrétní sítě tedy pozřebujeme ještě něco - FPNN.

2.3.2 FPNN

FPNN (Field Programmable Neural Network) [3] je jedna z možných konfigurací nějakého FPNA. Udává konrétní propojení neurálních zdrojů a hodnoty všech parametrů. Pro nevstupní uzly a spoje definuje:

$\Theta_n \in \mathbb{R}$ - počáteční hodnota vnitřní iterační proměnné v aktivátoru (může sloužit jako práh)

$a_n \in \mathbb{N}$ - počet iterací (použití iteračního operátoru), po nichž aktivátor aplikuje funkční operátor

$W_n(p), T_n(p) \in \mathbb{R}$ - konkrétní hodnoty koeficientů afinního operátoru $\alpha_{(p,n)}$

$\forall p, p \in Pred(n) : r_n(p)$	-	binární příznak určující, zda jsou spoj (p, n) a aktivátor n propojeny
$\forall s, s \in Succ(n) : S_n(s)$	-	binární příznak určující, zda jsou aktivátor n a spoj (n, s) propojeny
$\forall p, s; p, s \in Pred(n), s \in Succ(n) : R_n(p, s)$	-	binární příznak určující, zda jsou spoje (p, n) a (n, s) propojeny

Pro každý vstupní uzel:

$c \in \mathbb{N}$	-	počet vstupů
$\forall s, s \in Succ(n) : S_n(s)$	-	binární příznak určující, zda jsou vstupní uzel n a spoj (n, s) propojeny

Jak vidíme v definici, FPNN binárními příznaky definuje lokální propojení aktivátorů se spoji a spojů se spoji. Definuje také koeficienty affinních operátorů spojů a přidává několik dalších parametrů.

Popis funkce

Jak bylo řečeno výše, neurální zdroje pracují autonomně a kontinuálně s ostatními. Jsou také (většinou) propojeny s některými svými předchůdci, od nichž získávají data ke zpracování, a s několika následníky, jimž předávají výsledky své práce. Toto předávání dat se děje formou požadavků. Vždy, když nějaký neurální zdroj dokončí výpočet a vystaví jeho výsledek na výstup, zároveň pošle svým připojeným následníkům zprávu, že má pro ně data – vygeneruje požadavky. Zde je problém, že následníci mohou být zrovna zaměstnáni a nepřevezmou si data hned. Proto následníci při převzetí požadavku posílají zpátky potvrzení o jeho převzetí. Každý neurální zdroj pak musí pozastavit svou činnost, dokud nedostane potvrzení od všech následníků. Teprve poté sám přijme další požadavek. Činnost neurálních zdrojů jde tedy shrnout do následujících bodů:

1. Neurální zdroj vybere ke zpracování jeden z požadavků od předchůdců čekajících na jeho vstupu. Pokud žádné požadavky na zpracování nečekají, neurální zdroj čeká, dokud nějaký nepřijde.

2. Pošle potvrzení předchůdci, který vybraný požadavek vznesl.

3. Zpracuje požadavek:
   Spoj: Na přijatá data aplikuje affinní operátor.
   Aktivátor: Na přijatá data aplikuje iterační operátor. Pokud se nachází v poslední iteraci – provedl parametrem a předepsaný počet iterací – na iteračním operátorem akumulovaná data aplikuje funkční operátor (přenosovou funcki) a pokračuje bodem 4. Pokud se v poslední iteraci nenachází, pokračuje bodem 1.

4. Vystaví výsledek na svůj výstup a vygeneruje požadavky pro připojené následníky.

5. Počká, dokud neobdrží potvrzení pro všechny požadavky.

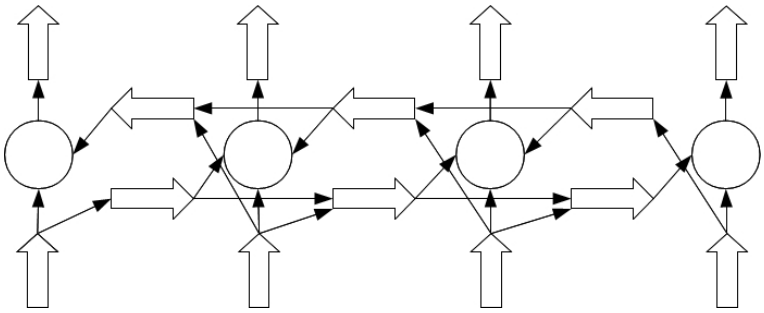
6. Pokračuje bodem 1.

V bodu 1 je důležitý způsob, jakým dochází k výběru požadavků. Pro správnou funkci FPNN je nutné, aby byly požadavky vybírány spravedlivě. Kdyby tomu tak nebylo, mohlo by docházet k zbytečným prodlevám práce některých částí sítě. Horším problémem by pak bylo, kdyby se některá část sítě zpozdila tak, že při dávkodém zpracování dat by aktivátory při iterování nad požadavky mohly počítat s daty požadavku patřícímu do jiné dávky dat,

což by zapříčinilo chybný výsledek celého výpočtu. Je tedy potřeba zvolit vhodnou techniku výběru požadavků, která těmto problémům předejde – například Round&Robin.

Mřížová struktura FPNN

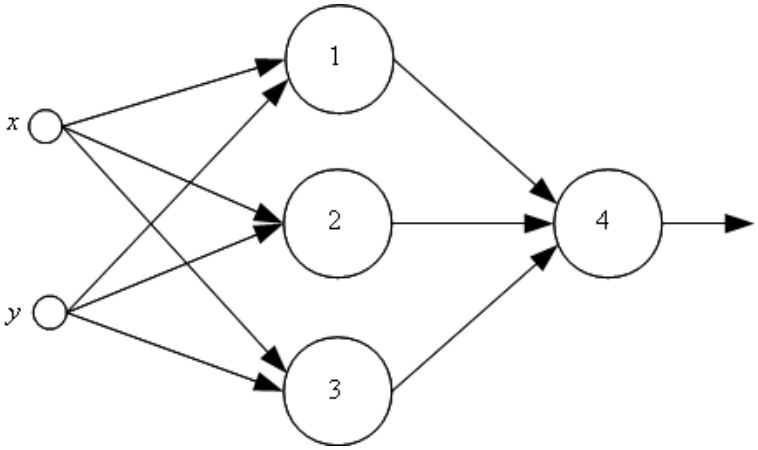
Jak bylo popsáno výše, koncept FPNA/FPNN zjednodušuje topologii propojení slučováním synapsí a jejich serializací. Dosahuje toho pomocí mřížové struktury propojení aktivátorů, jak znázorňuje obrázek 2.5. Na tomto obrázku kruhy představují aktivátory, tlusté šipky představují spoje a tenké šipky značí propojení jednotlivých jednotek signály. Na obrázku je vidět, že výstup každého aktivátoru je napojen na jeden spoj, který je připojen na aktivátor v následující vrstvě a také k řetězci spojů uvnitř vrtsvy, který realizuje propojení s ostatními aktivátory. Jednotlivé synapse (váhy) jsou tedy aproximovány sekvencí jednoho či více spojů. Konstrukce mřížového FPNN je popsána v [7].



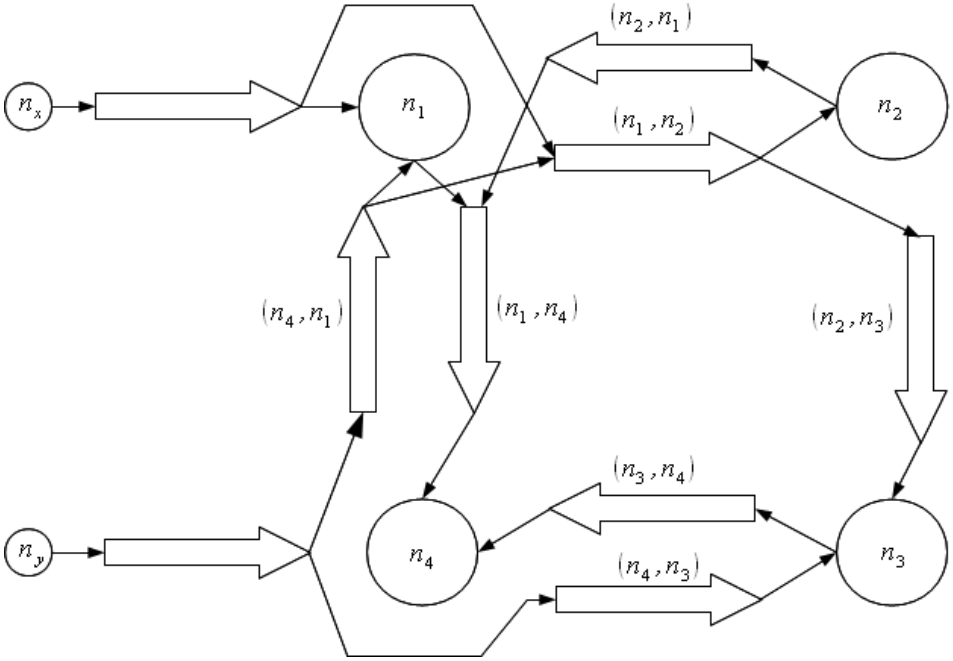
Obrázek 2.5: Mřížová struktura FPNN

Příklad

Činnost FPNN bude prakticky vysvětlena v následujícím příkladu, který vychází z příkladu uvedeném v [8]. Mějme neuronovou síť následující topologie:



Obrázek 2.6: Neuronová síť příkladu FPNN



Obrázek 2.7: Vzorové FPNN

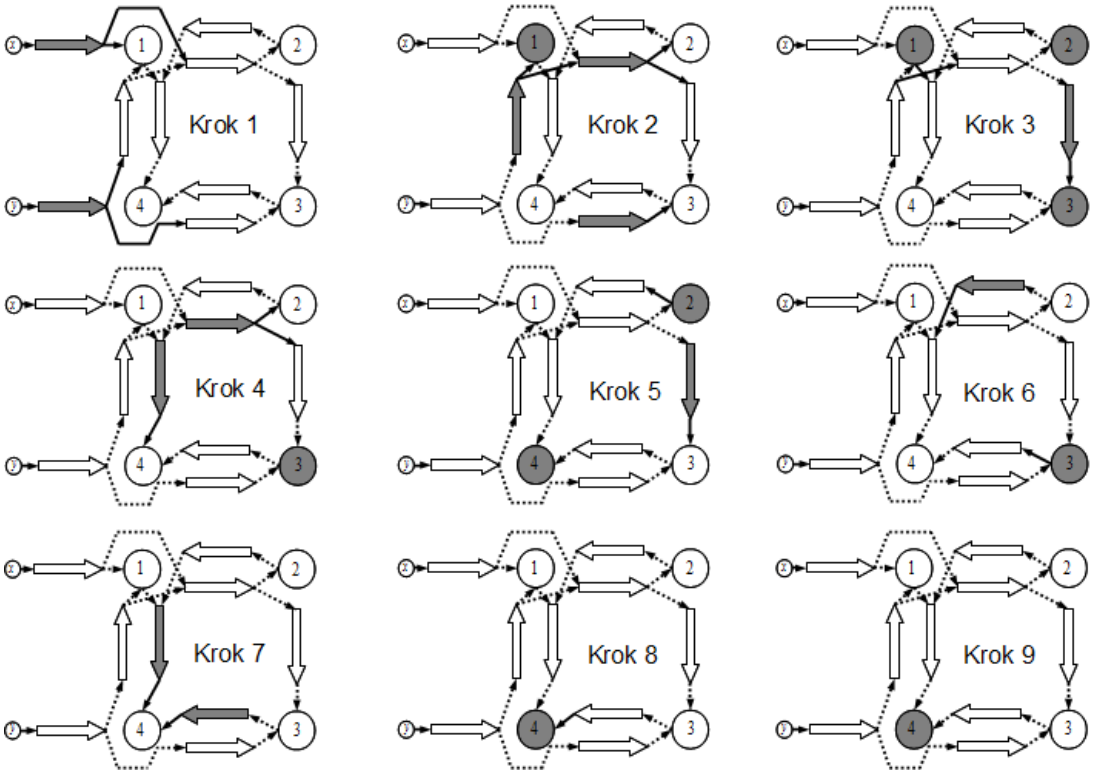
Příslušné FPNN by pak mohlo být definováno takto:

$N = \{n_x, n_y, n_1, n_2, n_3, n_4\}$
$E = \{(n_x, n_1), (n_y, n_4), (n_1, n_2), (n_2, n_1), (n_1, n_4), (n_4, n_1), (n_3, n_4), (n_4, n_3), (n_2, n_3)\}$
$i_{n1} = i_{n2} = i_{n3} = i_{n4} = ((x, x') \to x + x')$
$f_{n1} = f_{n2} = f_{n3} = f_{n4} = (x \to tanh(x))$
pro $n_1 : r_{n1}(n_x), r_{n1}(n_4), S_{n1}(n_4), R_{n1}(n_x, n_2), R_{n1}(n_2, n_4), R_{n1}(n_4, n_2)$
pro $n_2 : r_{n2}(n_1), S_{n2}(n_1), R_{n2}(n_1, n_3)$
pro $n_3 : r_{n3}(n_2), r_{n3}(n_4), S_{n3}(n_4)$
pro $n_4 : r_{n4}(n_1), r_{n4}(n_3), R_{n4}(n_y, n_1), R_{n4}(n_y, n_3)$
$a_1 = 2, a_2 = 2, a_3 = 2, a_4 = 3$
$c_{nx} = c_{ny} = 1$
Všechny ostatní hodnoty jsou nulové.

V definici vidíme, že všechny iterační operátory i jsou definované jako obyčejné sčítání. Všechny funkční operátory f, které v aktivátorech realizují přenosové funkce, jsou definovány jako hyperbolický tangents. Parametry aktivátorů a udávající počet iterací příslušnými operátory i, jsou nastavené v souladu s počtem synapsí, které do daného neuronu ve výchozí neuronové síti vedou. Parametry vstupů c říkají, že jsou na každém vstupním uzlu očekávána pouze jedna data. Pro všechny aktivátory jsou také definovány příznaky propojení s ostatními zdroji. Toto propojení je na obrázku 2.7.

Obrázek 2.8 ilustruje činnost vzorového FPNN v devíti krocích, v nichž se udává následující činnost:

1. Spoje $(n_x, n_1), (n_y, n_4)$ zpracovávají vstupní data a generují požadavky pro aktivátor n_1 a spoje $(n_1, n_2), (n_4, n_1)$ a (n_4, n_3).



Obrázek 2.8: Činnost vzorového FPNN

2. V druhém kroku pracují aktivátor n_1 a spoje $(n_1, n_2), (n_4, n_1)$ a (n_4, n_3) konkurentně zpracovávají požadavky a generují nové požadavky pro aktivátory n_1, n_2 a n_3 a spoje (n_1, n_2) a (n_2, n_3).

3. Ve třetím kroku pracují aktivátory n_1, n_2, n_3 a spoj (n_2, n_3). Aktivátor n_1 nyní také generuje požadavek, protože již zpracoval potřebný počet vstupních požadavků. Spoj (n_1, n_2) má na vstupu sice také čekající požadavek ze spoje (n_4, n_1), nicméně jej nemůže zpracovat, protože čeká na potvrzení převzetí požadavků od aktivátoru n_2 a spoje (n_2, n_3).

4. Nyní již spoj (n_1, n_2) svůj požadavek zpracovává zároveň se spojem (n_1, n_4), který také poprvé vygeneruje požadavek pro aktivátor n_4.

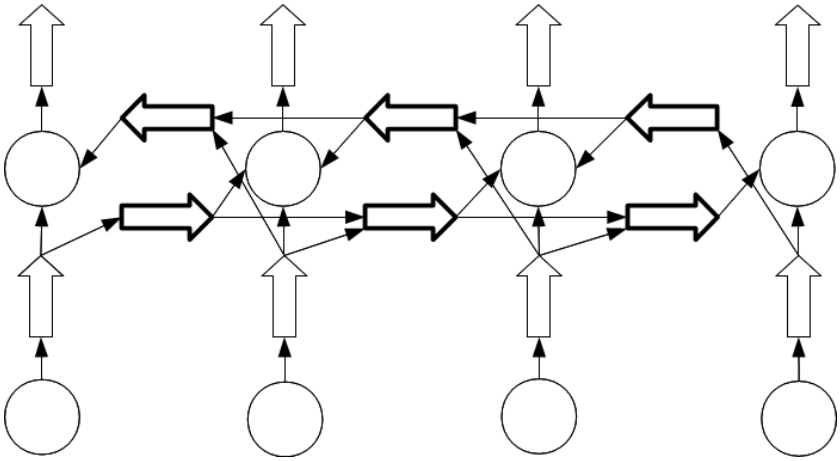
5. V tomto kroku své požadavky zpracovávají spoj (n_2, n_3) a aktivátory n_2 a n_4, který přitom zpracovává první ze tří očekávaných požadavků, ve vzorové neurální síti to jsou data od neuronu 1.

6. Spoj (n_2, n_1) a aktivátor n_3 zpracovávají požadavky.

7. Spoje (n_1, n_4) a (n_3, n_4) zpracovávají požadavky a generují zbývající dva požadavky pro aktivátor n_4.

8. Aktivátor n_4 zpracovává požadavek od spoje (n_1, n_4), která přišla přes spoj (n_2, n_1) od aktivátoru n_2, ve vzorové neurální síti by to tedy byla data od neuronu 2.

9. Aktivátor n_4 zpracovává poslední požadavek od spoje (n_3, n_4), ve vzorové neurální sítí to jsou data od neuronu 3. Výstup tohoto aktivátoru je výstupem celého FPNN.



Obrázek 2.9: Propojovací sled

Na 2.8 obrázku je patrné, jak data putují od vstupních uzlů až na výstup. Dobře viditelná je i konkurentost práce jednotlivých neurálních zdrojů. Z obrázků je také zřejmé, že k některým aktivátorům se data nedostávají přímo, ale procházejí v několika krocích několika spoji, či dokonce procházejí několika různými cestami přes různé posloupnosti spojů. Data tedy na cestě k ativátoru podstupují několik affinních transformací. Jak bylo řečeno výše, tyto affinní trasformace aproximují váhy jednotlivých synapsí mezi neurony. Protože parametr T je v tomto FPNN definován pro všechny spoje nulový, dochází pouze k samotnému násobení dat parametrem W. Je tedy zřejmé, že vážení jednotlivých dat je rozděleno do jednoho nebo několika násobení či dokonce součtu několika násobení (tento součet ale provádí aktivátor). Na základě tohoto výroku můžeme definovat aproximace konkrétních váh jako posloupnost jednotlivých násobení či součtu násobení ve vzorovém FPNN např. takto:

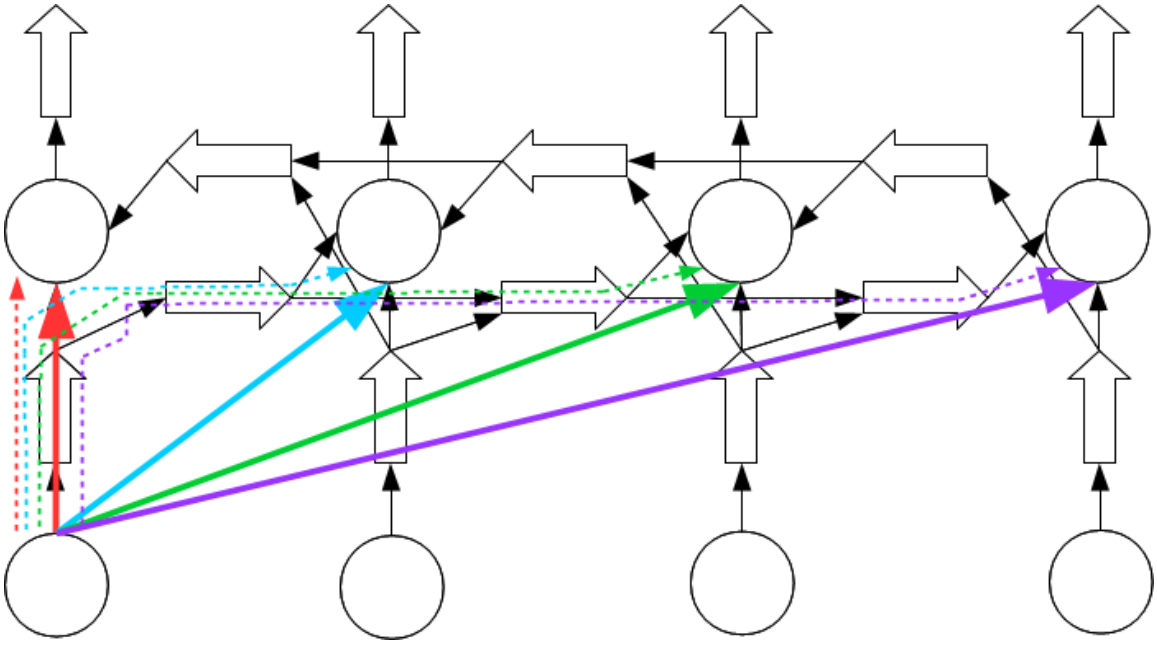
$$
\begin{aligned}
w_{x,1} &= W_{n_1}(n_x) \\
w_{x,2} &= W_{n_1}(n_x) \cdot W_{n_2}(n_1) \\
w_{x,3} &= W_{n_1}(n_x) \cdot W_{n_2}(n_1) \cdot W_{n_3}(n_2) \\
w_{y,1} &= W_{n_4}(n_y) \cdot W_{n_1}(n_4) \\
w_{y,2} &= W_{n_4}(n_y) \cdot W_{n_1}(n_4) \cdot W_{n_2}(n_1) \\
w_{y,3} &= W_{n_4}(n_y) \cdot W_{n_4}(n_3) + W_{n_4}(n_y) \cdot W_{n_1}(n_4) \cdot W_{n_2}(n_1) \cdot W_{n_3}(n_2) \\
w_{1,4} &= W_{n_4}(n_1) \\
w_{2,4} &= W_{n_1}(n_2) \cdot W_{n_4}(n_1) \\
w_{3,4} &= W_{n_4}(n_3)
\end{aligned}
$$

2.3.3 Řětězec a sled spojů

Pro potřeby dalšího textu zavedeme několik nových pojmů.

Definice 1 *Sledem spojů obecně rozumíme libovolnou posloupnost spojů propojených mezi sebou.*

Definice 2 *Propojovací sled je sled spojů propojující aktivátory skrze celou jednu vrstvu. Obrázek 2.9 silnými čarami znázorňuje propojovací sled.*



Obrázek 2.10: Řetězce

Definice 3 *Řetězec je sled, který aproximuje synaptický spoj. Na obou koncích řetězce je připojen aktivátor a přes řetězec pomyslně prochází původní synapse. Každý řetězec tak odpovídá jedné původní synapsi. Z každého aktivátoru vychází tolik řetězců, kolik z něj původně vycházelo synapsí. V každém aktivátoru končí tolik řetězců, kolik v něm končilo synapsí. Jednotlivé spoje jsou sdíleny mezi více řetězci - skrze různé části sledu procházejí různé počty řetězců.*

Pojem řetězec z definice 3 názorně ilustruje obrázek 2.10. Na obrázku jsou původní synapse vyvedené tlustými plnými barevnými čarami. Tečkované tenčí čáry stejné barvy pak označují řetězce, které tyto synapse aproximují. Je zřejmé že červený řetězec je shodný s původní synapsí, zatímco fialovou synapsi aproximuje řetězec čtyř spojů. Je také vidět, jak přes spoje propojovacího sledu vede více než jeden řetězec.

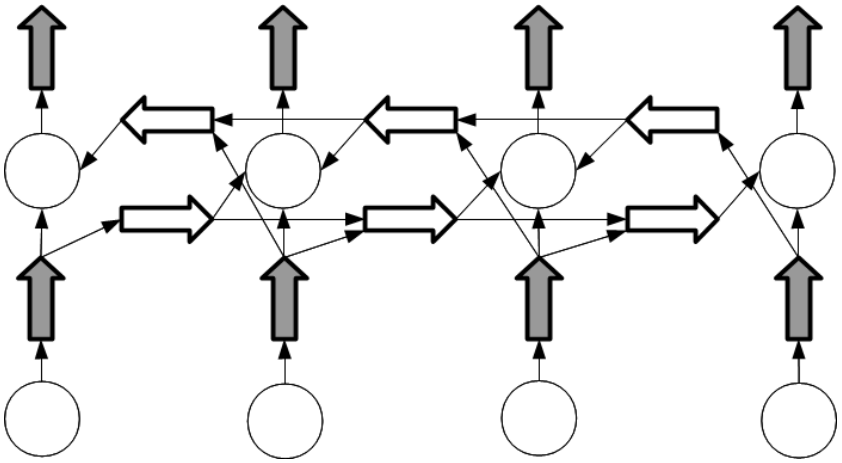
V rámci každého řetězce můžeme nalézt dva významné spoje. Pro potřeby textu jsme je pojmenovali initiál a (definice 4) a terminál (definice 5).

Definice 4 *Initál je první spoj v řetězci, přes který procházejí všechny řetězce vycházející z jednoho aktivátoru. Je to bezprostřední následovník aktivátoru. Initiály jsou zobrazeny šedě na obrázku 2.11.*

Definice 5 *Terminální spoj, nebo terminál, je poslední spoj, v kterém končí jeden nebo několik různých řetězců od různých aktivátorů předchozí vrtsvy. Terminály jsou bezprostředními předchůdci aktivátorů. Terminály jsou vyvedeny silnými čarami na obrázku 2.11.*

Definice 6 *Neterminální spoj, nebo neterminál, je spoj, který není terminál.*

V mřížové struktuře FPNN bude mít každý aktivátor obvykle dva terminály (zleva a zprava) a na jeho výstup bude připojen jeden initiál. V mřížové struktuře jsou všechny spoje v propojovacím sledu terminály. Všechny vertikální spoje jsou initiály a každý initiál je zároveň i terminál.



Obrázek 2.11: Initiály (šedě) a terminály (silně)

2.4 Algoritmus Nelder-Mead

V této kapitole bude představen pro tuto práci důležitý optimalizační algoritmus, který jsme použili v řadě experimentů popsaných v dalších kapitolách. Tento algoritmus je implementován v modulu `FPNNWeightsMapper`.

Algoritmus Nelder-Mead [23][2][4] je široce používaný algoritmus pro řešení optimalizačních problémů bez omezujících podmínek. Je to metoda minimalizace funkce o n proměnných použitím simplexu s $n + 1$ vrcholy.

Simplex je geometrický obrazec, který je konvexní obálkou množiny $n + 1$ bodů v n-rozměrném prostoru. Tímto útvarem metoda zkoumá tvar krajiny minimalizované funkce a hledá cestu k minimu. Jak simplex postupně klesá krajinou funkce k některému z lokálních minim, zmenšuje se až na předepsanou hraniční velikost, kdy algoritmus končí a minimum se vypočte jako poloha těžiště simplexu.

Z výše uvedeného vyplývá, že metoda obecně nachází lokální minima a není tedy optimální. Praxe však ukazuje, že poskytuje dobré výsledky a kromě toho je implementačně jednoduchá a výpočetně rychlá, což z ní činí dobře použitelnou metodu.

Algoritumus pracuje iterativně. V každé iteraci vybere nejhorší vrchol simplexu, tedy vrchol s nejvyšší funkční hodnotou a pokusí se najít nový lepší vrchol s nižšší hodnotou. K tomu používá celkem tři operace: *reflexi*, *expanzi* a *kontrakci*, která se dále dělí na vnitří a vnější. V případě, že se žádnou z operací nepodaří najít lepší vrchol, použije se čtvrtá operace - *smrštění*, která zmenší velikost simplexu.

Operace algorimu Nelder-Mead

Předpokládejme posloupnost vrcholů $x_0..x_n$ seřazenou podle jejich funkčních hodnot. Pak vrchol x_n je vrchol s nejvyšší funkční hodnotou. Bod $\overline{x}$ nazvěme *centroid*.

$$\overline{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i \tag{2.14}$$

Reflexe Tato operace promítá nejhorší vrchol na opačnou stranu centroidu. Nachází *reflexní bod* x_r, který leží na spojnici centroidu s vrcholem x_n, ale na opačné straně.

$$x_r = \overline{x} + \alpha(\overline{x} - x_n) \tag{2.15}$$

Parametr α je reflexní parametr, jehož hodnota je obvykle volena: $\alpha = 1$. V případě, že $f(x_0) \leq f(x_r) < f(x_{n-1})$, pak je vrchol x_n nahrazen novým vrcholem x_r. Jinak se pokračuje některou z následujících operací.

Expanze Pokud $f(x_r) < f(x_0)$ pak zřejmě stojí za to podívat se, jestli bychom ve stejném směru nenalezli ještě lepší vrchol. Protáhneme tedy simplex tímto směrem - nalezneme *expanzní bod* ležící na spojnici centroidu a reflexního bodu, ale leží ve β krát větší vzdálenosti. Expanzní parametr β je obvykle volen $\beta = 2$

$$x_e = \overline{x} + \beta(x_r - \overline{x}) \tag{2.16}$$

Pokud $f(x_e) < f(x_r)$, vrchol x_n je nahrazen vrcholem x_e, jinak x_r.

Kontrakce Pokud $f(x_r) > f(x_{n-1})$, tak reflexe nebyla úspěšná a třeba zjistit, zda posunutí nového vrcholu opačným směrem (blíže k centroidu) nenabídne lepší řešení. Tato operace je opakem expanze. Provádějí se dva druhy kontrakce:
Vnitřní kontrakce se provádí pokud $f(x_r) > f(x_n)$. Pak kontrakční bod x_e se vypočte:

$$x_{ic} = \overline{x} + \gamma(x_n - \overline{x}) \tag{2.17}$$

Jinak se provede *vnější* kontrakce a kontrakční bod se vypočte:

$$x_{ec} = \overline{x} + \gamma(x_r - \overline{x}) \tag{2.18}$$

Kontrakční parametr $0 < \beta < 1$ se obvykle volí $\beta = 0.5$. Pokud $f(x_{ic}) < f(x_n)$ nebo $f(x_{ec}) < f(x_n)$ pak je vrchol x_n příslušným bodem nahrazen. Jinak se pokračuje následující operací.

Smrštění Pokud selhala reflexe i kontrakce, zkusíme zmenšit rozměry simplexu v naději, že se dostaneme do příznivější krajiny průběhu funkce, nebo alespoň vysvobodíme vrchol s nejvyšší funkční hodnotou z nepříznivé oblasti, odkud jej ostatní operace nedostaly. Každý vrchol simplexu nahradíme novým vrcholem podle rovnice:

$$x_i^{novy} = x_0 + \delta(x_i - x_0) \tag{2.19}$$

Smršťovací parametr $0 < \delta < 1$ obvykle volíme $\delta = 0.5$.

Ukončení

Algoritmus by měl být ukončen, jakmile je splněno kritérium:

$$\sqrt{\frac{1}{n+1}\sum_{i=0}^{n}[f(x_i) - \overline{f}]^2} < \Upsilon \tag{2.20}$$

kde Υ je předem zvolená ukončující hodnota a

$$\overline{f} = \frac{1}{n+1}\sum_{i=0}^{n} f(x_i) \tag{2.21}$$

Kapitola 3

Implementace neuronových sítí v FPGA

Neuronové sítě jsou struktury s velkým potenciálem pro paralelizaci výpočtů. Jedním z možných způsobů, jak toho dosáhnout je implementovat je v hardware takovým způsobem, aby existovaly jako množina samostatných výpočetních obvodů a mohly tak počítat skutečně paralelně. O implementaci neuronových sítí v FPGA toho bylo napsáno mnoho [29], [27], [10], [9], [24], [3]. Tato kapitola představuje pouze základní úvod do problematiky implementace neuronových sítí a FPNN v obvodech FPGA. Dále představuje principy implementace FPNN v FPGA, kterou jsme navrhli již v naší předchozí práci [15]. V této diplomové práci plně využíváme naši již vyvinutou implementaci, nad kterou jsme neprovedli žádné další úpravy. Podrobnější popis problematiky a implementace je tedy možné najít v [15].

3.1 Reprezentace dat

Prvním problémem implementace neuronových sítí v FPGA je reprezentace dat. Na běžných CPU v softwarových implementacích neuronových sítí se obvykle využívá reprezentace dat v plovoucí řádové čárce (standart IEEE-754). Tato reprezentace je pro implementaci v FPGA nevýhodná protože výpočty v plovoucí řádové čárce vykonávají komplexní algoritmy, jejichž implementace v FPGA by zkonzumovala velké množství zdrojů. Proto je preferována reprezentace v pevné řádové čárce. V [12] bylo experimentálně dosaženo dobrých výsledků při 16-bitové reprezentaci vah, s 8-bitovu celou a 8-bitovou desetinnou částí.

Při použití plovoucí řádové čárky je množné použít například algoritmů CORDIC [18].

3.2 Aktivační funkce

Dalším problémem je implementace aktivační funkce. Některé používané aktivační funkce jsou složité nelineární funkce. Jejich přímá implementace v hardware by byla velmi náročná, vyžadovala by značné množství zdrojů a byla by pomalá. Proto se aktivační funkce obvykle aproximuje jinou funkcí, jako například rampovou funkcí, splajnem, nebo jinou nelineární funkcí s podobným průběhem, ale s nižšími implementačními nároky a prostorovou i časovou složitostí. Dalším výhodným způsobem je použití vyhledávacích tabulek, které nabízejí nejvyšší rychlost. Jejich nevýhodou je však potřebná kapacita tabulek a tedy vysoká prostorová složitost.

Pro potřeby této práce potřebujeme vhodně aproximovat především funkci sigmoid (2.3). V [16] byly definovány funkce sloužící k aproximaci rostoucích nelineárních aktivačních funkcí, především funkce sigmoid:

$$H_s(x) = \begin{cases} x\,(\beta + \theta x) & \text{pro } x \in \langle -L, 0\rangle \\ x\,(\beta - \theta x) & \text{pro } x \in (0, L\rangle \end{cases} \tag{3.1}$$

Tato funkce má průběh podobný hyperbolickému tangentu. β a θ pak určují sklon a zkosení funkce v její centrální části, tj. na intervalu $\langle -L, L\rangle$. S pomocí této funkce může být definována i funkce aproximující bipolární funkci sigmoid:

$$G_s(x) = \begin{cases} -1 & \text{pro } x \in (-\infty, -L\rangle \\ H_s(x) & \text{pro } x \in (-L, L) \\ 1 & \text{pro } x \in \langle L, \infty) \end{cases} \tag{3.2}$$

Pomocí předchozí funkce (3.2) lze také definovat funkci pro aproximaci běžné (unipolární) funkce sigmoid (2.3):

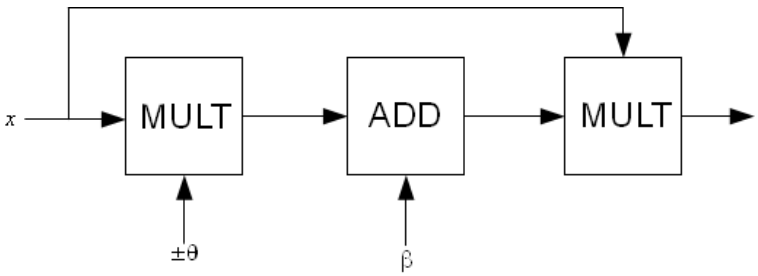
$$F_s(x) = \frac{1}{2} G_s(x) + \frac{1}{2} \tag{3.3}$$

V [16] také odvodili vzorce pro výpočet parametrů β a θ v závislosti na šířce středního intervalu L:

$$\theta = \frac{1}{L^2} \tag{3.4}$$

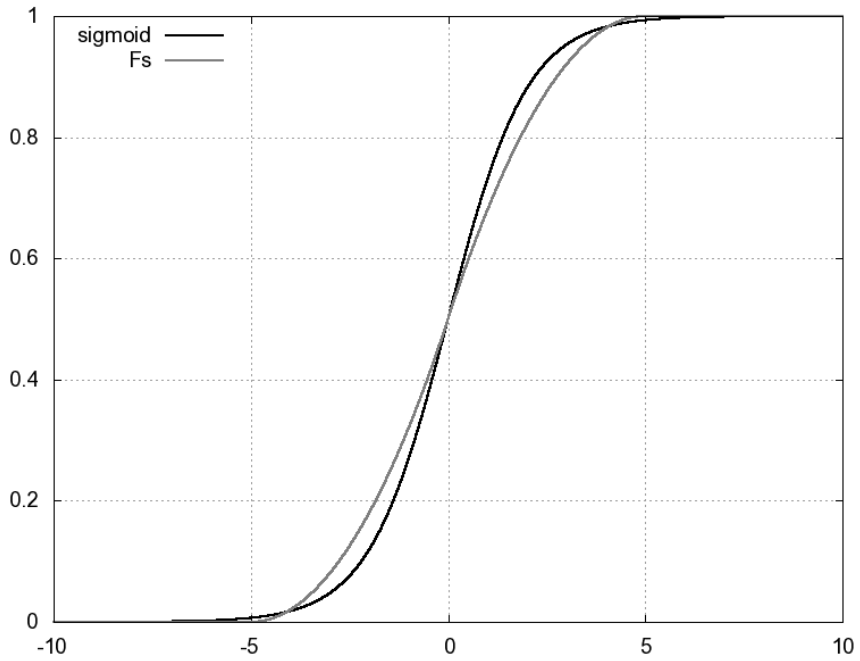
$$\beta = \frac{2}{L} \tag{3.5}$$

Výpočet funkce (3.1) lze tedy implementovat jako dvě násobení a jedno sčítání (viz obrázek 3.1). Nicméně, pokud bude interval L zvolen v mocninách dvou - 2^m , lze jedno násobení nahradit bitovým posunem, který je implementačně jednodušší. Jak vypadá výsledná aproximace funkce sigmoid (2.3) definovanou funkcí (3.3) je znázorněno na obrázku 3.2 a chybu tohoto způsobu aproximace zobrazuje obrázek 3.3. Interval byl zvolen $L = 4$, aby odpovídal centrální nelineární části funkce sigmoid (2.3). Chyba aproximace se pohybuje v intervalu $< -0.0216, 0.0216 >$. Při 8-bitové reprezentaci desetinné části bude chybou aproximace zatížen třetí nejnižší bit.



Obrázek 3.1: Implementace

Představená metoda poskytuje dobrou přesnost aproximace při nízké implementační náročnosti. V [15] bylo provedeno srovnání této metody s dalšími metodami aproximace.



Obrázek 3.2: Aproximace nelineární funkcí

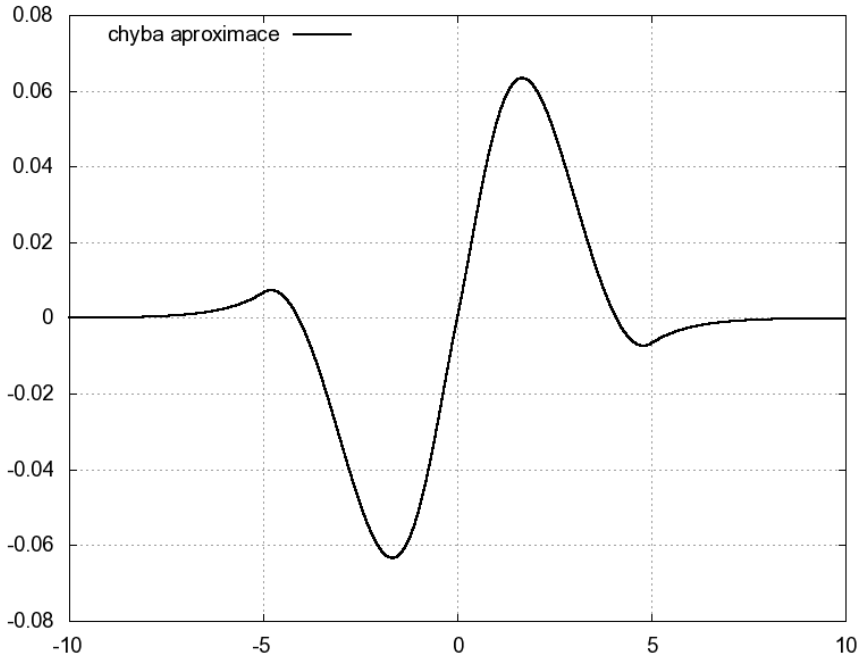
3.3 Implementace FPNN

V jazyce VHDL byla vytvořena implementace konceptu FPNN [5]. Spoje i aktivátory byly realizovány jako samostatné entity. Tato implementace není závislá na konkrétním zařízení FPGA a je ji tedy možné na libovolném, dostatečně vybaveném čipu použít.

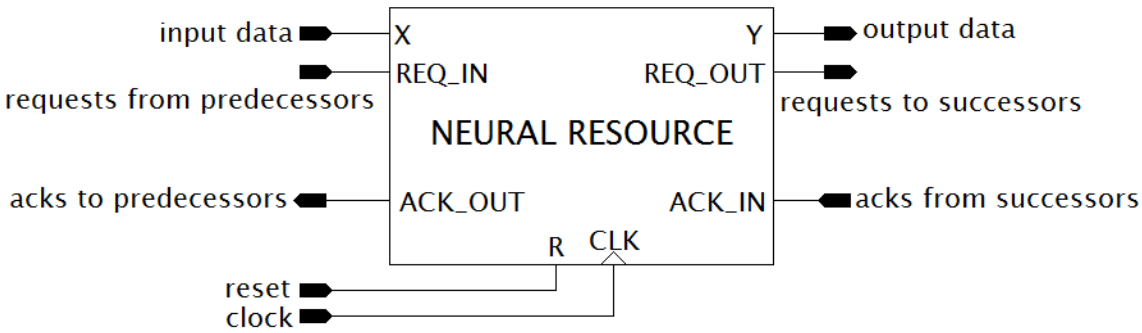
3.3.1 Neurální zdroje

Neurální zdroje byly implementovány jako plně generické samostatné entity LINK a AKTIVATOR. Obě tyto entity mají shodné rozhraní. Některé generické paramatery jsou shodné a některé rozdílné. Propojením instancí těchto entit signály je možné zkonstruovat FPNN. Obrázek 3.4 popisuje shodné rozhraní obou entit. [15] obsahuje podrobný popis implementace neurálních zdrojů včetně všech rozhraní a generických parametrů.

Rozhraní entit LINK a ACTIVATOR			
Jméno	Směr	Typ	Funkce
X	vstup	sběrnice	datový vstup
Y	výstup	sběrnice	datový výstup
REQ_IN	vstup	sběrnice	vstup požadavků od předchůdců
REQ_OUT	výstup	sběrnice	výstup požadavků pro následníky
ACK_IN	vstup	sběrnice	vstup potvrzení od následníků
ACK_OUT	výstup	sběrnice	bystup potvrzení pro předchůdce
R	vstup	vodič	nulovací signál
CLK	vstup	vodič	hodinový signál



Obrázek 3.3: Chyba aproximace nelineární funkcí



Obrázek 3.4: Entita neurálního zdroje

Kapitola 4

PyFPNN

I když je definován poměrně jednoduchý algoritmus převodu neuronové sítě na mřížovou sktrukturu FPNN, u větších sítí je to při ruční práci náročný úkol. Stejně tak sestavení popisu takového FPNN je kvůli složité definici náročná práce při níž lze velmi snadno udělat chybu. Sestrojení FPNN je tak nelehká činnost zatížená velkým prostorem pro dělání chyb. Stejně tak i ruční verifikace tohoto návrhu. Dalším náročným úkolem je také vytváření VHDL popisu implementujícího FPNN i neuronové sítě, pokud jsou realizovány pomocí oddělených obvodů pro neurony/aktivátory a synapse/spoje. Konstrukce takového obvodu vyžaduje vytvoření velkého množství komponent, jejich správné propojení pomocí signálů a správné nastavení generik u každé komponenty. Při ruční práci se tedy také jedná o náročný úkol s velkým potenciálem pro dělání chyb. Abychom se zbavili nutnosti těchto náročných ručních operací a nebezpečí chyb při nich vznikajících, vytvořili jsme projekt PyFPNN[15].

PyFPNN je sada aplikací napsaných v multiplatformním skriptovacím jazyce Python ve verzi 3, které slouží k usnadnění výše uvedených úkonů. Jedna z těchto aplikací umí převézt zadanou neruonovou síť na mřížové FPNN a vygenerovat jeho popis. Další umí podle tohoto popisu zkonstruovat VHDL kód implementující zadané FPNN pomocí námi vytvořených entit. Další dovede provést simulaci činnosti FPNN a umožňuje tak verifikovat jeho návrh. Poslední aplikace umí vytvořit VHDL popis neuronové sítě pomocí našich komponent naivní implementace. Všechny FPNN i neuronové sítě použité v této práci jsme vytvořili a implmentovali s pomocí těchto aplikací a ušetřili si tak mnoho práce.

4.1 FPNNGenerator

`FPNNGenerator` je aplikace sloužící ke konstrukci FPNN z normální neuronové sítě. Převádí běžnou vrstvenou síť struktury backpropagation či zkratkovitou síť do mřížové struktury FPNN. Jako vstup slouží argumenty, kterými je popsána struktura vzorové neuronové sítě. Výstupem je popis FPNN ve formátu uvedeném v kapitole B.1. Kromě FPNN v mřížové struktuře umí aplikace vygenerovat i FPNN ve struktuře odpovídající původní neuronové síti a provádí tak přimé mapování původní sítě na neurální zdroje.

4.2 FPNNSimulator

`FPNNSimulator` je aplikace umožňující softwarovou simulaci FPNN. Jako vstup bere soubor s popisem konkrétního FPNN ve formátu uvedeném v kapitole B.1. Jednotlivé neurální zdroje implementuje jako instance konkrétních tříd. FPNN tedy existuje jako množina ob-

jektů provázaných způsobem předepsaným v souboru. Simulace výpočtu probíhá podle principů diskrétní simulace. Tomuto FPNN mohou být předložena vstupní data a ono odsimuluje výpočet nad těmito daty a poskytne výsledek. Výpočet probíhá v buď ve vestavěném datovém typu pevné `float` nebo v řádové čárce, přičemž počty bitů celé a desetinné části jsou specifikovány v popisu FPNN ve zdrojovém souboru. Výpočet v pevné řádové čárce využívá knihovny `FixedPoint`.

Kromě samotné simulace umožňuje tato aplikace zobrazit vztahy mezi jednotlivými neurálními zdroji a verifikovat tak správnost struktury FPNN. Simulátor rovněž umí provést test FPNN nad předloženou datovou sadou, přičemž poskytuje výpis obsahující pro každý vstupní vektor hodnotu vypočteného výstupu, dále hodnotu cíleného (korektního) výstupu, hodnotu chyby, nebo u klasifikačních úloh úspěšnost klasifikace. Na konci výpisu je shrnutí celkové chyby nebo počtu správně a nesprávně klasifikovaných vektorů.

Modul také nabízí možnost experimentování s iterativními úpravami prahů. Aktivátory si dělají charakteristiky o potenciálech, kterých při průchodu dat nabývali a na základě těchto informací mohou několika různými metodami iterativně měnit své prahy před další iterací nad datovou sadou. Jde o jistou formu primitivní optimalizace, které funguje jako rozšíření metod popsaných v kapitole 5.6.

Simulátor umožňuje před začátkem simulace importovat nové hodnoty vah spojů z externího souboru. Soubor musí být ve formátu odřádkovaných přiřazení hodnot jménům spojů, tedy např. $n1n2 = 0.3$. Díky tomu je možné provádět optimalizace či úpravy FPNN pomocí externích programů a tyto úpravy pak otestovat bez nutnosti vytvářet nový soubor s popisem FPNN. Importovaná data jsou mapována na zadané FPNN, takže je nutné aby importovaná data pocházela z FPNN stejné struktury. Tuto funkcionalitu je možné použít nař. pro import výsledků optimalizačního skriptu programu Matlab, jehož generování nabízí modul FPNNWeightsMapper.

4.3 VHDLGenerator

`VHDLGenerator` je aplikace sloužící ke konstrukci VHDL kódu implementujícího FPNN. Je implementována pomocí stejných tříd jako `FPNNSimulator`, které podle své potřeby rozšiřuje. Při generování VHDL kódu používá námi vytvořené komponenty popsané v [15]. Jako vstup je brán soubor s popisem FPNN ve formátu uvedeném v kapitole B.1. Aplikace vygeneruje entitu pojmenovanou podle souboru s popisem a v ní deklaruje porty odpovídající vstupům a výstupům neurálních zdrojů na vstupu a výstupu FPNN. V architektuře vytvoří pro všechny neurální zdroje instance odpovídajících VHDL komponent a propojí je signály způsobem odpovídajícím struktuře vzorového FPNN. U každé komponenty také nastaví všechny potřebné generiky. Na začátek kódu také vloží odkazy na všechny potřebné balíčky, takže výsledný kód je možné rovnou vysyntetizovat. Pro tento kód umí vygenerovat i kostru testbenche se zkušebními signály.

Modul také nabízí volitelnou možnost vložit na vstupy a výstupy generovaných jednotek obvody pro komunikaci s počítačem přes rozhraní JTAG.

4.4 NaiveNNGenerator

`NaiveNNGenerator` nesouvisí s konceptem FPNN. Slouží ke konstrukci VHDL kódu naivní implementace neuronových sítí jak byla popsána v [15]. Podobně jako u `FPNNGeneratoru` jsou vstupem argumenty popisující strukturu sítě, nebo soubor ve formátu uvedeném v ka-

pitole B.2. Generování VHDL kódu pracuje podobně jako `VHDLGenerator`. Je vygenerována odpovídající entita a jednotlivé objekty neuronové sítě jsou instancovány jako komponenty popsané v [15], a propojeny signály tak, aby výsledná struktura odpovídala původní neuronové síti. I v této aplikaci je možné vygenerovat odpovídající testbench se zkušebními signály.

Modul také nabízí volitelnou možnost vložit na vstupy a výstupy generovaných jednotek obvody pro komunikaci s počítačem přes rozhraní JTAG.

4.5 FPNNWeightsMapper

`FPNNWeightsMapper` je aplikace sloužící k převodu naučené neuronové sítě na FPNN. Realizuje mapování vah synapsí sítě na příslušné parametry spojů FPNN. Podporuje celkem 10 metod mapování, z nichž všechny budou popsány v následujícím textu. Osm z těchto metod je také možné kombinavat po dvou, třech a čtyřech. Modul poskytuje také ještě další dvě metody mapování založené na matematické optimalizaci.

Dále nabízí možnost vyhodnotit a vypsat údaje o aproximaci původní neuronové sítě. Umí vypsat hodnotu chyby aproximace, vzorec chybové funkce, podle které byla chyba spočítána (u každého FPNN je jiná, viz. kapitola 5.2.1), a také seznam proměnných figurujících v této chybové funkci. Je možné vypsat charakteristiku aproximace na každém spoji. K této charakteristice je také poskytován přehled způsobu aproximace každé synapse, ve kterém je k ní vypsán řetězec, který ji aproximuje, posloupnost násobení realizovanou řetězcem a také vznikající chybu aproximace.

Další nabízenou funkcionalitou je duplikace některých spojů v propojovacím sledu. Tato metoda byla navžena pro snížení chyby aproximace za cenu strukturálních změn a nárůstu prostorové složitosti. Více tuto metodu rozvádí kapitola 7.

Modul umožňuje vygenerovat skript v programovacím jazyce programu Matlab vypočítávající chybové funkce (rovnice 5.3) spojů, sestavující a vyčíslující celkovou chybovou funkci v rámci všech spojů a aplikující matematickou optimalizaci pomocí funkce *fminseacrh*, kterou Matlab nabízí. Tento skript má podobu sady souborů obsahujících chybové funkce spojů a skript pro jejich postupné vyčíslování, sestavování chybové funkce a její optimalizaci. Kód tohoto skriptu je vypisován na výstup modulu FPNNWeightsMapper a soubory s chybovými funkcemi spojů jsou ukládány do aktuální složky.

Při spuštění skriptu v programu Matlab jsou na jeho výstup vypsány nové hodnoty všech spojů po optimalizaci ve formátu, který podporuje modul FPNNSimulator pro import. Pomocí této funkcionality je tedy možné provádět optimalizaci mapování použitím programu Matlab a testovat její výsledky v programu FPNNSimulator.

4.6 Teacher

`Teacher` je aplikace, která implementuje učení FPNN algoritmem *backpropagation*. Jeho vstupem je soubor s popisem FPNN a soubor s trénovacími daty. Umožňuje pomocí parametrů specifikovat učicí parametry *momentum rate* a *learning rate*. Také je možné zadat cílenou chybu, počet epoch a zakázat učení prahů. Výstupem programu je textový popis naučeného FPNN a případně hodnota dosažené chyby. Problematika učení algoritmem backpropagation je více rozebrána v kapitole 6.

Kapitola 5

Převod naučené neuronové sítě na FPNN

I když můžeme uvažovat o učení FPNN po vzoru neuronových sítí, nemusí to být vždy přístupná možnost. Trénovací data již nemusí existovat, nebo mohou být obtížně k sehnání. V takovém případě by nebylo možné FPNN naučit a provést implementaci zamýšlené sítě. Pro takový případ by bylo vhodné najít metodu, s níž bychom se mohli pokusit namapovat původní neuronovou síť na FPNN, přenést informace o vahách a prazích do parametrů FPNN tak, aby se co nejvíce podobalo původní síti a poskytovalo co nejpodobnější výsledky.

Tato kapitola popisuje metody, které jsme pro tuto úlohu navrhli. Představuje několik metod převodu neuronové sítě na FPNN, zabývá se jejich kombinacemi a rozšířeními, a prezentuje dosažené výsledky experimentů.

5.1 Rozbor

Bez trénovacích dát máme při převodu naučené neuronové sítě k dispozici pouze informace o její struktuře, vahách synaptických spojů a prazích. Informace o struktuře použijeme ke zkonstruování odpovídajícího mřížového FPNN. Prahy pak můžeme přenést přímo na parametry θ aktivátorů.

Většinu synaptických vah ale přímo přenést nelze. Váhu na parametr W (váhu spoje) můžeme přenést přímo pouze u těch spojů, které přímo propojují dva neurony a mají tak stejnou pozici jako původní synapse. Zbytek synapsí je ale aproximován řetězcem spojů, přičemž každým spojem prochází více než jedna původní synapse. Váhy tak není možné přenést přímo, ale je nutné je přepočítat v závislosti na struktuře FPNN.

Je nutné vzít v úvahu, že počet spojů FPNN je výrazně menší než počet synapsí původní sítě a aproximace několika synapsí jedním řetězcem spojů nutně vede ke snížení výpočetní síly FPNN oproti původní neuronové síti. Mapováním proto není ve většině případů možné dosáhnout naprosté přesnosti aproximace.

5.2 Přepočet vah - mapování

Pro zkoumaná FPNN mřížové struktury jsme navrhli několik metod mapování vah neuronové sítě na váhy spojů FPNN, všechny jsou založeny na stejném postupu. Tyto metody nepočítají s prahy, ty jsou přenášeny přímo z neuronů na aktivátory. Mapování vah probíhá tím způsobem, že se postupně dopočítávají parametry W spojů tak, aby jejich hodnota

byla v posloupnosti násobení co nejbližší původním vahám s tím, že jsou vzaty v úvahu všechny aproximované synapse. Každý řetězec spojů aproximující konkrétní synapsi je tedy rozpočítán na posloupnost násobení aproximující původní váhu synapse. Protože přes jeden spoj prochází více řetězců, z nichž každý na tomto spoji potřebuje svůj vlastní parametr (člen posloupnosti násobení), což je v rozporu s tím, že spoj disponuje pouze jediným parametrem W, je třeba učinit kompromis. Rozhodli jsme se vyzkoušet několik různých metod na získání kompromisu - aritmetický průměr, medián, vážený průměr s různými metodami určení vah.

Výpočet probíhá postupně směrem od neuronů nižší vrstvy a postupuje jednotlivými řetězci spojů až k aktivátorům následující vrstvy. Prvním vypočítávaným spojem $L1$ je tedy vždy initiál aktivátoru $n1$. Přes tento spoj procházejí všechny řetězce aproximující původní synapse neuronu $n1$. Vzhledem k mřížové struktuře FPNN je jeden z aktivátorů ($n2$) následující vrstvy ve stejném sloupci mříže jako aktivátor $n1$ a je proto přímo napojen na výstup spoje $L1$ a ten je tak zároveň terminálem. Synapse původní sítě mezi aktivátory $n1$ a $n2$ je tudíž aproximována řetězcem délky 1 tvořeným spojem $L1$. Tento spoj proto ponese hodnotu váhy původní synapse.

Další nejbližší aktivátor je s aktivátorem $n1$ propojen řetězcem o délce 2. Protože jeden spoj v řetězci už je daný, je nutné parametry následujícího spoje dopočítat tak, aby v součinu se spojem $L1$ měl hodnotu váhy původní aproximované synapse.

Aktivátor $n1$ a $n3$ jsou propojeny řetězecem o délce 3. Dva ze spojů v něm patří k řetězcům zmíněmým výše. Třetí spoj je terminálem toho řetězce a protože tento spoj může být i terminálem jiných řetězců vycházejích z jiných aktivátorů (např. $n2$), není možné dopočtené parametry součinů aproximujících synapse aplikovat přímo, protože jsou pro každý řetězec různé. Jako výsledná váha spoje se tedy dosadí některou z metod vypočtený kompromis mezi vypočtenými parametry všech řetězců končících v tomto terminálu.

Stejným způsobem se vypočtou parametry následujících spojů. Pokračuje se, dokud nejsou vypočteny parametry všech spojů.

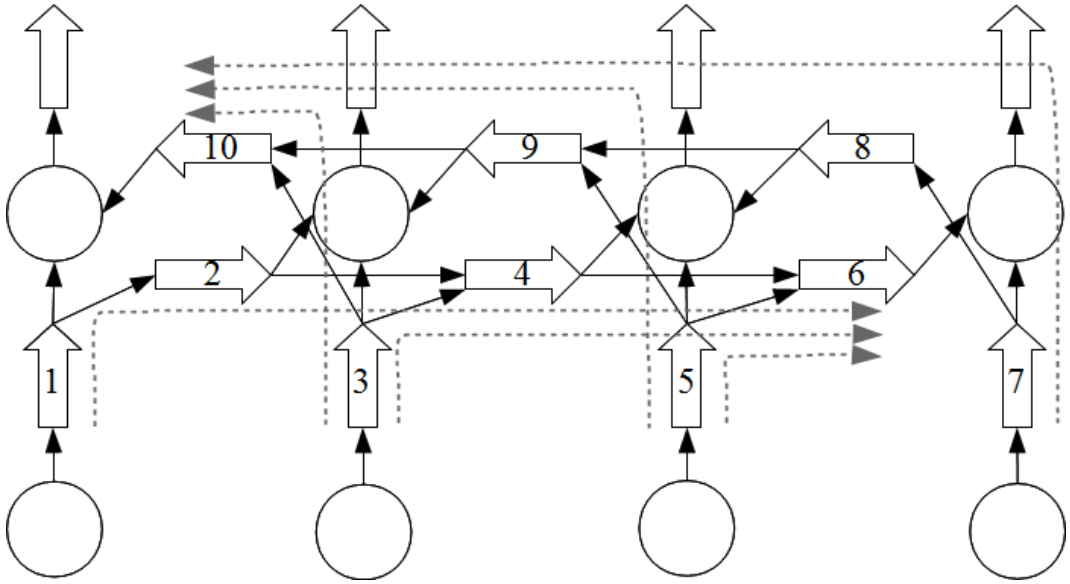
Postup mapování znázorňuje obrázek 5.1. Na tomto obrázku je přerušovanými šedými šipkami znázorněn postupný průběh mapování v rámci jedné vrstvy od prvního initiálu až po poslední terminál v propojovacím sledu. Čísla uvedená ve spojích ukazují pořadí jejich postupného mapovaní.

Povšimněme si nyní několika věcí. Jsou-li na sled spojů mapovány synapse vycházející jen z jednoho neuronu, je dosaženo přesné aproximace, protože každý terminál aproximuje jen jedinou synapsi. Dále každý propojovací sled se skládá ze dvou sledů jdoucích v opačných směrech vrstvou. Každý z těchto sledů může přesně aproximovat synapse jednoho neuronu. To znamená, že mapování propojení vrtsvy s dvěma neurony s následující vrstvou s libovolným počtem neuronů vede na přesnou aproximaci. Bereme-li v úvahu trojvrstvé sítě, znamená to, že dvouvstupé sítě mohou dosahovat velké přesnosti aproximace.

Vrstva, která má pouze jeden aktivátor pak postrádá propojovací sled, a aktivátor je propojen spojem s každým aktivátorem předchozí vrstvy. V tomto případě je tedy možné mapovat synapse na spoje přímo a dosáhnout tak maximální přesnosti. Z těchto faktů je zřejmé, že nehledě na počet neuronů skryté vrtsvy (u trojvrstvé sítě) budou sítě s dvěma vstupy a jedním výstupem mapovány na FPNN vždy s naprostou přesností.

5.2.1 Experimenty

Pro experimenty byly vybrány dvě klasifikační úlohy ze sady Proben1 [25]. Jejich vlastnosti shrnuje tabulka 5.1. Úloha Two Spiral splňuje podmínky na strukturu sítě, s níž lze



Obrázek 5.1: Postup mapování FPNN

dosáhnout přesné aproximace podle kapitoly 5.2. Budeme tedy u této úlohy očekávat nejlepší možné výsledky. Úloha Diabetes naopak podmínky nesplňuje a přesné aproximace u ní nemůže být FPNN mřížové struktury dosaženo.

Referenční sítě byly implementovány v jazyce C++ pomocí volně dostupné knihovny FANN [1]. Byly naučeny na trénovacích datech a otestovány na testovacích datech. Pro každou referenční síť bylo nástrojem `FPNNGenerator` vygenerováno FPNN. Pro převod parametrů referenční sítě na FPNN byl použit modul `FPNNWeightsMapper`. FPNN byla testována modulem `FPNNSimulator` nad trénovacími i testovacími daty.

Vzhledem k tomu, že se tato práce zabývá převodem neuronových sítí na FPNN, nebyla pro ni zajímavá samotná kvalita výsledků řešení zvolených úloh referenčními sítěmi a FPNN. Veličinou, kterou jsme sledovali byla shoda výsledků klasifikace referenčních sítí s příslušnými FPNN, z níž plynula přesnost aproximace. Měřili jsme tedy, kolik vstupních vektorů klasifikovalo FPNN stejně jako referenční síť a kolik odlišně.

Úloha	Vstupů	Skrytých neuronů	Výstupů	Trénovacích a testovacích vektorů
Two Spiral	2	25	1	193
Diabetes	8	16	2	384

Tabulka 5.1: Vlastnosti úloh a referenčních sítí

Chybové funkce

Pro potřeby měření kvality aproximace jsme definovali následující funkce:

$$chain\_poduct(X) = \prod w_i, i \in X \tag{5.1}$$

$$ApproxError = \frac{1}{|W|} \sum_{w \in W} (w - chain\_poduct(approx(w)))^2 \tag{5.2}$$

$$LinkApproxError = \sum_{w \in W\_link} (w - chain\_poduct(approx(w)))^2 = \tag{5.3}$$

$$= \sum_{w \in W\_link} chain\_poduct(x)^2 - 2 \times chain\_poduct(x) \times w + w^2 \tag{5.4}$$

Rovnice (5.1) popisuje funkci, která vrátí součin spojů v řetězci X. Rovnice (5.2) popisuje chybovou funkci celého FPNN, která je vyjádřena jako střední kvadratická odchylka hodnot aproximovaných vah a součinu řetězců, které je aproximují. V této rovnici funkce $approx(w)$ vrací řetězec, jímž je aproximována váha w. Kde W je množina vah původní sítě. Funkce (5.3) je chybová funkce jednoho spoje. W_link je zde množina synapsí končících v tomto spoji.

Výsledky experimentů budou uváděny formou tabulek, pro většinu z nich bude tvořit předlohu tabulka 5.2. Ve sloupci *Shoda* je počet vektorů, které FPNN klasifikovalo stejně jako referenční síť, ve sloupci *Neshoda* pak počet vektorů, u kterých ke shodě nedošlo. Sloupec *Shoda [%]* obsahuje procentuální míru shody s referenční sítí. Z více stejných výsledků složitějších metod budeme uvádět jeden, nebo ty dosahující zlepšení.

Metoda	Shoda	Neshoda	Shoda [%]
Název metody	Počet shod	Počet neshod	Procentuální shoda.

Tabulka 5.2: Předloha tabulky pro výsledky experimentů

5.2.2 Implementace úloh

V kapitole 3 byla naznačena VHDL implementace FPNN. Tato implementace byla vytvořena v [15]. V této práci jsme se zaměřili na problém převodu a mapování neuronových sítí na FPNN, přičemž jsme k experimentům využívali aplikací PyFPNN, které poskytly možnost tvorby i testování FPNN. Při našich experimentech jsme nijak neměnili strukturu FPNN ani jeho vnitřních komponent, takže zásahy do VHDL implementace nebyly nutné.

Pro dvě FPNN, které jsme v této práci používali k aproximaci neuronových sítí řešících úlohy Diabetes a Two spiral jsme pomocí nástroje `VHDLGenerator` z PyFPNN vytvořili VHDL kód jejich implementace. Tyto dvě jednotky byly vysyntetizovány nástroji firmy Xilinx pro FPGA XC6VLX75T rodiny Virtex6 od téže firmy. Byly provedny operace mapování a place&route. Tabulka 5.3 uvádí prostorovou náročnost obou designů po těchto operacích a jejich maximální pracovní frekvenci.

Úloha	Slice Registers	Slice LUT	DSP	Max. f [MHz]
Diabetes	3 489	12 356	164	70.932
Two spiral	4 182	16 117	230	71.088

Tabulka 5.3: Přehled náročnosti implementace v FPGA

5.3 Výsledky experimentů a aritmetickým průměrem

První vyzkoušenou metodou zisku kompromisu mezi parametry vypočtenými pro jednotlivé synapse byl artimetický průměr podle rovnice (5.5), kde P je množina vypočtených

parametrů. Tuto metodu budeme dále nazývat metoda ARIT.

$$W = \frac{\sum p}{|P|}, p \in P \tag{5.5}$$

Tabulka 6.1 obsahuje výsledky namapovaného FPNN pro obě úlohy nad trénovací i testovací datovou sadou.

Úloha	Datová sada	Shoda	Neshoda	Shoda [%]
Diabetes	Trénovací	250	133	65.274
Diabetes	Testovací	281	102	73.368
Two Spiral	Trénovací	192	0	100
Two Spiral	Testovací	192	0	100

Tabulka 5.4: Výsledky experimentů metody ARIT nad oběma úlohami a datovými sadami

Podle očekávání z kapitoly 5.2, úloha Two Spiral vykazuje díky vhodné struktuře sítě pro mapování naprostou přesnost aproximace. V příladě úlohy Diabetes ale experimenty ukazují, že mapování příliš dobrých výsledků nedosahuje. Nad testovací datovou sadou byla dosažena přesnost aproximace 53.525%, nad trénovací dokonce jen 44.125%.

Protože metoda ARIT nedosahuje pro úlohu Diabetes plánovaných výsledků, vyzkoušeli jsme i další metody výpočtu vah spojů, které rozebereme v následujících podkapitolách. Vzhledem k již přesnému namapování úlohy Two Spiral budeme v následujících experimentech uvažovat jen úlohu Diabetes.

5.4 Výsledky experimentů s mediánem

Další vyzkoušenou metodou zisku kompromisu mezi parametry vypočtenými pro jednotlivé synapse byl medián množiny vypočtených parametrů. Tuto metodu budeme dále nazývat metoda MED. Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

Výsledky ukazují, že metoda MED nepřináší vůči metodě ARIT žádné zlepšení.

5.5 Výsledky experimentů s váženým průměrem

Další vyzkoušenou metodou zisku kompromisu mezi parametry vypočtenými pro jednotlivé synapse byl vážený průměr vypočítávaný podle rovnice (5.6), v níž P je množina vypočtených parametrů. W je množina vah jednotlivých parametrů.

$$W = \frac{\sum w_i * p_i}{\sum w_i}, p \in P, w \in W, i \in \langle 1, |P| \rangle \tag{5.6}$$

Experimentovali jsme s různými metodami výpočtu vah jednotlivých parametrů. Vztahovali jsme jejich výpočet k informací, které částečně namapované FPNN poskytuje, nebo které poskytuje původní síť. Využívali jsme informací o délkách sledů mezi spoji a zdrojovými aktivátory, dále hodnot částečného součinu předchozích spojů ve sledu, původních hodnot vah synapsí referenční sítě a také hodnot samotných parametrů. Počítali jsme v přímé i nepřímé úměře. Vytvořené metody jsou rozebrány v následujících podkapitolách spolu s výsledky experimentů.

5.5.1 Váhy odvozené od vzdálenosti

První možností, kterou jsme vyzkoušeli bylo odvozovat váhy od vzdálenosti mezi počítaným spojem a zdrojovým aktivátorem řetězce, pro jehož parametr se váha počítá, kdy tato vzdálenost je rovna počtu spojů ve sledu vedoucím ze zdrojového aktivátoru k počítanému spoji. Tato metoda vychází z úvahy, že různě dlouhé sledy spojů v částech řetězců přicházejících do spoje by měly mít na aproximaci procházejících synapsí různý vliv. Tedy, že počet spojů aproximujících synapsi by měl mít vliv na váhu jejího parametru v počítaném spoji. Vyzkoušeli jsme přímou i nepřímou úměru vah parametrů a vzdálenosti.

Váhy přímo úměrné vzdálenosti

V prvním případě jsou váhy určeny vzdáleností právě počítaného spoje od zdrojového parametru synapse i pro kterou byl vypočítán parametr p_i. Delší synapse tedy mají vyšší váhu. Zobrazuje to rovnice (5.7), kde funkce *distance* vrací hodnotu této vzdálenosti jako počet spojů ve sledu vedoucím ke zdrojovému aktivátoru. P je množina vypočtených parametrů pro všechny aproximované synapse. w_i je váha parametru $p_i \in P$. Tuto metodu budeme dále nazývat metoda DIST_DP.

$$w_i = distance(p_i), i \in \langle 1, |P| \rangle \tag{5.7}$$

Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

Váhy nepřímo úměrné vzdálenosti

Ve druhém případě jsou váhy určeny blízkostí právě počítaného spoje od zdrojového parametru synapse i, pro kterou byl vypočítán parametr p_i. Kratší synapse tedy mají vyšší váhu. Zobrazuje to rovnice (5.8), kde funkce *closeness* vrací hodnotu této vzdálenosti jako o jedničku zvýšený rozdíl mezi maximální délkou všech přicházejících řetězců a délkou řetězce aproximující synapsi i. P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. Tuto metodu budeme dále nazývat metoda DIST_IP.

$$w_i = closeness(p_i), i \in \langle 1, |P| \rangle \tag{5.8}$$

Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

5.5.2 Váhy odvozené od součinu spojů v řetezcích

Další veličinou, od níž jsme zkoušeli dovozovat váhy byla hodnota součinu spojů v řetězcích aproximující váhy. Tato metoda vychází z úvahy, že části různých řetězců přicházejících do spoje nabývají v součinu různé hodnoty, podle kterých by se rozhodovalo, který řetězec upřednostnit a přiřadit do něj patřícímu parametru větší váhu a který naopak upozadit. Vyzkoušeli jsme přímou i nepřímou úměru vah parametrů a částečného součinu.

Váhy přímo úměrné součinu

Ve třetím případě jsou váhy parametru každé synapse i určeny hodnotou součinu vah spojů v řetězci, který synapsi i aproximuje. Jde tedy o hodnotu násobení bez posledního násobence realizovaného počítaným spojem. Čím větší je hodnota součinu, tím je větší příslušná váha.

Zobrazuje to rovnice (5.9). P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. X je množina spojů v řetězci aproximujícím synapsi i. $w_{i,x}$ je váha x-tého spoje v řetězci aproximujíxím synapsi i. Tuto metodu budeme dále nazývat metoda PROD_DP.

$$w_i = \prod w_{i,x}, i \in \langle 1, |P| \rangle, x \in \langle 1, |X| \rangle \tag{5.9}$$

Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

Váhy nepřímo úměrné součinu

Ve čtvrtém případě jsou váhy parametru každé synapse i určeny převrácenou hodnotou součinu vah spojů v řetězci, který synapsi i aproximuje. Jde tedy o hodnotu násobení bez posledního násobence realizovaného počítaným spojem. Čím větší je hodnota součinu, tím je meší příslušná váha. Zobrazuje to rovnice (5.10). P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. X je množina spojů v řetězci aproximujícím synapsi i. $w_{i,x}$ je váha x-tého spoje v řetězci aproximujíxím synapsi i. Tuto metodu budeme dále nazývat metoda PROD_IP.

$$w_i = \frac{1}{\prod w_{i,x}}, i \in \langle 1, |P| \rangle, x \in \langle 1, |X| \rangle \tag{5.10}$$

Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

5.5.3 Váhy odvozené od hodnot původních synapsí

Další veličinou, od níž jsme odvozovali váhy parametrů byly původní hodnoty vah aproximovaných synapsí. Tato metoda vychází z úvahy, že hodnota vah referenční sítě by měla mít vliv na rozhodování o tom, který řetězec bude upřednostněn a který upozaděn. Váha parametru bude tedy závislá na hodnotě původní váhy aproximované řetězcem, do kterého parametr náleží. Vyzkoušeli jsme přímou i nepřímou úměru vah parametrů a původních vah refereční sítě.

Váhy přímo úměrné hodnotám synapsí

V pátém případě jsou váhy parametru každé synapse i určeny hodnotou původní synapse. Čím měla původní synapse větší váhu, tím větší váhu má i parametr. Zobrazuje to rovnice (5.11). P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. $w_i^{original}$ původní hodnota váhy synapse i. Tuto metodu budeme dále nazývat metoda WEIG_DP.

$$w_i = w_i^{original}, i \in \langle 1, |P| \rangle \tag{5.11}$$

Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

Váhy nepřímo úměrné hodnotám synapsí

V tomto případě jsou váhy parametru každé synapse i určeny převrácenou hodnotou původní synapse. Čím měla původní synapse větší váhu, tím menší váhu má parametr. Zobrazuje to rovnice (5.12). P je množina vypočtených parametrů. w_i je váha parametru

$p_i \in P$. $w_i^{original}$ původní hodnota váhy synapse i. Tuto metodu budeme dále nazývat metoda WEIG_IP.

$$w_i = \frac{1}{w_i^{original}}, i \in \langle 1, |P| \rangle \tag{5.12}$$

Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

5.5.4 Váhy odvozené od hodnot vypočtených parametrů

Poslední veličinou, od níž jsme odvozovali váhy parametrů byly hodnoty parametrů částečných součinů. Tato metoda vychází z úvahy, že hodnota vypočteného parametru by měla mít na jeho váhu vliv, protože čím větší či naopak menší hodnotu parametr má, tím větší chybu vznikající v předchozí části řetězce kompenzuje. Vyzkoušeli jsme přímou i nepřímou úměru vah parametrů a hodnot vypočtených parametrů.

Váhy přímo úměrné hodnotám vypočtených parametrů

V tomto případě jsou váhy parametru každé synapse i určeny hodnotou parametru, který synapsi v daném spoji aproximuje. Čím má parametr větší hodnotu, tím větší má váhu. Hodnota váhy v tomto případě odpovídá pořadí daného parametru ve vzestupně uspořádané množině všech parametrů. Tuto metodu budeme dále nazývat metoda PVAL_DP. Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

Váhy nepřímo úměrné hodnotám vypočtených parametrů

V tomto případě jsou váhy parametru každé synapse i určeny hodnotou parametru, který synapsi v daném spoji aproximuje. Čím má parametr větší hodnotu, tím menší má váhu. Hodnota váhy v tomto případě odpovídá pořadí daného parametru ve sestupně uspořádané množině všech parametrů. Tuto metodu budeme dále nazývat metoda PVAL_IP. Tabulka 5.5 obsahuje výsledky namapovaného FPNN nad trénovací i testovací datovou sadou.

5.5.5 Shrnutí

Vyzkoušeli jsme tři různé způsoby výpočtu konečné váhy z parametrů jednotlivých řetězců (synapsí) - aritmetický průměr, medián a vážený průměr. V případě váženého průměru jsme váhy odvozovali od čtyřech různých veličin dostupných v FPNN, vždy v přímé i nepřímé úměře. Celkem jsme tedy použili deset různých metod výpočtu vah spojů. Tabulka 5.5 obsahuje výsledky všech těchto metod nad trénovacími i testovacími daty.

Metoda	Sada	Shoda	Neshoda	Shoda [%]
ARIT	Trénovací	250	133	65.274
ARIT	Testovací	281	102	73.368
MED	Trénovací	250	133	65.274
MED	Testovací	281	102	73.368
DIST_DP	Trénovací	237	146	61.879
DIST_DP	Testovací	215	168	56.135
DIST_IP	Trénovací	250	133	65.274
DIST_IP	Testovací	281	102	73.368
WEIG_DP	Trénovací	250	133	65.274

Metoda	Sada	Shoda	Neshoda	Shoda [%]
WEIG_DP	Testovací	280	103	73.107
WEIG_IP	Trénovací	250	133	65.274
WEIG_IP	Testovací	281	102	73.368
PROD_DP	Trénovací	251	132	65.535
PROD_DP	Testovací	277	106	72.323
PROD_IP	Trénovací	249	134	65.013
PROD_IP	Testovací	277	106	72.323
PVAL_DP	Trénovací	250	133	65.274
PVAL_DP	Testovací	281	102	73.368
PVAL_IP	Trénovací	250	133	65.274
PVAL_IP	Testovací	281	102	73.368

Tabulka 5.5: Porovnání metod

Nejlepších výsledků nad trénovací datovou sadou dosáhla metoda PROD_DP s 65.535 % shodou, s 0.261 % zlepšením oproti metodě ARIT. Všechny ostatní metody mají stejné nebo horší výsledky než metoda ARIT. Nad testovací datovou sadou dosáhly všechny metody stejných nebo horších výsledků než metoda ARIT. Zde zlepšení nebylo dosaženo.

5.5.6 Kombinace metod vážení

Kromě již zmíněných jednotlivých metod výpočtu vah při mapování váženým součinem, které nepřinesly téměř žádné zlepšení, jsme vyzkoušeli i všechny jejich kombinace. Kombinovali jsme pouze metody využívající k výpočtu různých veličin. Používali jsme kombinace dvou, tří i čtyř metod. Váha každého parametru byla vypočítána jako součet vah poskytnutých všemi metodami v kombinaci.

Kombinace 2 metod

Tato podkapitola obsahuje výsledky kombinování dvou metod, které jsou shrnuté v tabulkách C.1 a C.2. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Nejlepších výsledků nad trénovacími daty dosáhla kombinace metod PVAL_DP + DIST_DP s 72.845 % shody, tedy s o 7.310 % lepší shodou oproti jednoduchým metodám. Zlepšení dosáhly i další dvě kombinace metod. Nad testovacími daty nebylo zlepšení dosaženo.

Kombinace 3 metod

Tato podkapitola obsahuje výsledky kombinování tří metod, které jsou shrnuté v tabulkách C.3 a C.4. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Nejlepších výsledků nad trénovacími daty dosáhla kombinace metod PVAL_DP + PROD_IP + DIST_DP s 73.890 % shody, tedy s o 8.35533 % lepší shodou oproti jednoduchým metodám. Zlepšení dosáhly i další tři kombinace metod. Nad testovacími daty nebylo zlepšení dosaženo.

Kombinace 4 metod

Tato podkapitola obsahuje výsledky kombinování čtyř metod, které jsou shrnuté v tabulkách C.5 a C.6. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Nejlepších výsledků nad trénovacími daty dosáhla kombinace metod PVAL_DP + PROD_IP + WEIG_IP + DIST_DP s 73.107 % shody, tedy s o 7.57204 % lepší shodou oproti jednoduchým metodám. Zlepšení dosáhly i další tři kombinace metod. Nad testovacími daty dosáhla nejlepších výsledků kombinace metod PVAL_DP + PROD_DP + WEIG_DP + DIST_DP s 73.890 % shody, tedy s o 0.52233 % lepší shodou oproti jednoduchým metodám. Zlepšení nad testovacími daty dosáhly i další dvě kombinace metod.

Shrnutí kombinací

Tato podkapitola obsahuje výsledky kombinování všech metod, které jsou shrnuté v tabulkách 5.6 a 5.7. Do tabulke byly zahrnuty kombinace nehledě na počet členů. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Tabulky 5.8 a 5.9 obsahují ty metody, které dávaly lepší výsledky, než jakých bylo dosaženo v předchozí části s jednoduchými nekombinovanými metodami. Ve sloupci Nárůst [%] je v těchto tabulkách uvedeno zlepšení výsledků oproti nejlepšímu dosaženému výsledku předchozí části.

Nad trénovací datovou sadou dosáhla kombinace tří metod PVAL_DP + PROD_IP + DIST_DP celkové zlepšení o 8.35533 % ve srovnání s dosud nejúspěšnější jednoduchou metodou PROD_DP. Kromě této kombinace dosáhlo lepších výsledků dalších sedm kombinací metod. Nad testovací datovou sadou bylo dosaženo zlepšení o 0.52233 %. Celkem tři metody dosáhly zlepšení výsledků nad testovacími daty.

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_IP+DIST_DP	283	100	73.890
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	280	103	73.107
PVAL_DP+DIST_DP	279	104	72.845
PVAL_DP+WEIG_IP+DIST_DP	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP	259	124	67.624
PVAL_IP+PROD_DP	253	130	66.057
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	252	131	65.796
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	251	132	65.535
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	250	133	65.274
PVAL_DP+PROD_DP+WEIG_DP+DIST_IP	249	134	65.013
WEIG_IP+DIST_DP	237	146	61.879
PVAL_IP+PROD_IP+DIST_DP	230	153	60.052
PVAL_IP+PROD_IP+WEIG_IP+DIST_DP	227	156	59.268
PVAL_IP+WEIG_IP+DIST_DP	225	158	58.746
PVAL_IP+DIST_DP	223	160	58.224
PVAL_DP+PROD_IP	206	177	53.785
PVAL_DP+PROD_IP+WEIG_IP	204	179	53.263
PVAL_DP+WEIG_IP	159	224	41.514

Tabulka 5.6: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	283	100	73.890
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	282	101	73.629
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	281	102	73.368
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	280	103	73.107
PVAL_IP+PROD_DP+WEIG_DP+DIST_IP	278	105	72.584
PROD_DP+WEIG_DP+DIST_IP	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP	276	107	72.062
PVAL_DP+WEIG_IP+DIST_DP	275	108	71.801
PVAL_DP+DIST_DP	274	109	71.540
PVAL_DP+PROD_IP+DIST_DP	272	111	71.018
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	270	113	70.496
PVAL_IP+PROD_IP+WEIG_IP+DIST_DP	240	143	62.663
PVAL_IP+DIST_DP	238	145	62.140
PROD_IP+DIST_DP	233	150	60.835
PROD_IP+WEIG_IP+DIST_DP	230	153	60.052
PVAL_IP+WEIG_IP+DIST_DP	229	154	59.791
PVAL_DP+PROD_IP	224	159	58.485
PVAL_DP+PROD_IP+WEIG_IP	218	165	56.919
WEIG_IP+DIST_DP	213	170	55.613
PVAL_DP+WEIG_IP	140	243	36.553

Tabulka 5.7: Porovnání metod nad testovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_IP+DIST_DP	73.89033	8.35533
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	73.10704	7.57204
PVAL_DP+DIST_DP	72.84595	7.31095
PVAL_DP+WEIG_IP+DIST_DP	72.32375	6.78875
PVAL_IP+PROD_DP+WEIG_IP	67.62402	2.08902
PVAL_IP+PROD_DP	66.05744	0.52244
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	65.79634	0.26134
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	65.53524	0.00024

Tabulka 5.8: Zlepšení metod nad trénovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	73.89033	0.52233
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	73.62924	0.26124
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	73.36814	0.00014

Tabulka 5.9: Zlepšení metod nad testovacími daty

5.5.7 Experimenty se započítáním nekončících řetězců

Všechny předešlé experimenty počítaly pouze s řetězci, které v daném spoji končily. V tomto případě tedy nebyl brán ohled na procházející řetězce aproximující synapse vedoucí do vzdálenějších aktivátorů (neuronů). Tento přístup zvyšuje přesnost aproximace v jednom spoji, protože ten musí aproximovat méně synapsí, v celém FPNN by ale mohl vést ke zhoršení aproximace, protože na některých spojích, aproximujících mnoho synapsí velmi rozdílných vah by mohlo docházet ke vzniku velkých chyb. Rozprostření aproximace těchto synapsí na celý řetězec a tedy rozprostření chyby na více spojů by mohlo vést ke zlepšení celkové aproximace. Rozhodli jsme se proto také provést rozšíření původních metod, v nichž jsme do váženého průměru započítávali i nekončící řetězce. Parametry pro tyto řetězce byly vypočteny stejným způsobem jako pro končící řetězce. Při výpočtu se tedy nebraly v úvahu hodnoty vah následujících spojů, které ještě nebyly spočteny a byly započítány s hodnotu neutrálního prvku. Váhy pro tyto parametry byly vypočítány podobnými metodami jako váhy paramerů končících spojů.

Tabulka 5.10 opakuje zkratky metod mapování s váženým průměrem, na jejichž jednoduché i kombinované verze jsme navrhovaná rozšíření aplikovali.

Zkratka	Název metody
DIST_DP	Vážení přímo úměrné vzdálenosti
DIST_IP	Vážení nepřímo úměrné vzdálenosti
PROD_DP	Vážení přímo úměrné součinu řetězce
PROD_IP	Vážení nepřímo úměrné součinu řetězce
WEIG_DP	Vážení přímo úměrné původní váze
WEIG_IP	Vážení nepřímo úměrné původní váze
PVAL_DP	Vážení přímo úměrné hodnotě parametru
PVAL_IP	Vážení nepřímo úměrné hodnotě parametru

Tabulka 5.10: Zkratky metod vážení

Váhy nepřímo úměrné vzdálenosti

V tomto experimentu jsme váhy počítali jako převrácenou hodnotu vzdálenosti, jak to znázorňuje rovnice (5.13). Funkce *target_distance* vrací zbývající délku řetězce aproximující synapsi i. Jde tedy o počet spojů mezi počítaným spojem a cílovým aktivátorem řetězce. P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. Toto rozšíření budeme dále označovat suffixem F1 za názvem metody.

$$w_i = \frac{1}{target\_distance(i)}, i \in \langle 1, |P| \rangle \tag{5.13}$$

Váhy nepřímo úměrné desetinásobku vzdálenosti

V tomto experimentu jsme váhy počítali jako převrácenou hodnotu desetinásobku vzdálenosti, jak to znázorňuje rovnice (5.14). Funkce *target_distance* vrací zbývající délku řetězce aproximující synapsi i. Jde tedy o počet spojů mezi počítaným spojem a cílovým aktivátorem řetězce. P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. Toto

rozšíření budeme dále označovat suffixem F10 za názvem metody.

$$w_i = \frac{1}{10 \times target\_distance(i)}, i \in \langle 1, |P| \rangle \tag{5.14}$$

Váhy nepřímo úměrné čtverci vzdálenosti

V tomto experimentu jsme váhy počítali jako převrácenou hodnotu druhé mocniny vzdálenosti, jak to znázorňuje rovnice (5.15). Funkce *target_distance* vrací zbývající délku řetězce aproximující synapsi i. Jde tedy o počet spojů mezi počítaným spojem a cílovým aktivátorem řetězce. P je množina vypočtených parametrů. w_i je váha parametru $p_i \in P$. Toto rozšíření budeme dále označovat suffixem FPOW za názvem metody.

$$w_i = \frac{1}{target\_distance(i)^2}, i \in \langle 1, |P| \rangle \tag{5.15}$$

Srovnání

Zde srovnáme výsledky dosažené představenými rozšířeními. V tabulkách 5.11 a 5.12 jsou uvedeny výsledky kombinací metod vážení a rozšíření, které dosáhly zlepšení výsledků oproti jednoduchým nekombinovaným metodám. Pět rozšířených kombinací dosáhlo zlepšení výsledků nad trénovacími daty, z nichž byla nejlepší kombinace PVAL_DP + PROD_DP_F10 se zlepšením 1.56682 %. Nad testovacími daty dosáhly zlepšení tři kombinace s nejvyšším zlepšením o 1.04453 % v případě kombinace PVAL_DP + PROD_DP + WEIG_DP + DIST_DP_FPOW. Přestože bylo dosaženo zlepšení výsledků oproti jednoduchým nekombinovaným metodám, bylo nižší než zlepšení u nerozšířených kombinovaných metod. V případě většiny kombinovaných metod má tedy toto rozšíření negativní vliv na výsledky.

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_DP_F10	67.10182	1.56682
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP_FPOW	66.57963	1.04463
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP_FPOW	66.05744	0.52244
PVAL_IP+PROD_DP+WEIG_IP_F10_DESC	65.79634	0.26134
PVAL_DP+PROD_DP+WEIG_DP+DIST_IP_FPOW	65.53524	0.00024

Tabulka 5.11: Zlepšení metod nad trénovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP_FPOW	74.41253	1.04453
PVAL_IP+PROD_DP+WEIG_IP_F10	73.62924	0.26124
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP_F10	73.36814	0.00014

Tabulka 5.12: Zlepšení metod nad testovacími daty

5.6 Přepočet prahů

Doposud jsme při převodu neuronové sítě na FPNN přepočítávali pouze váhy synapsí na váhy spojů, prahy neuronů jsme na parametry θ aktivátorů přenášeli přímo. Protože mají

prahy na výslednou hodnotu potenciálu a tím i aktivační funkce velký vliv, nabízejí další možnost, jak zlepšit výsledky aproximace. V následujícím textu budeme v souvislosti s aktivátory hovořit o *prazích* místo o *parametrech* θ, ale ve stejném významu.

Při podrobnějším zkoumání činnosti skryté vrtsvy FPNN jsme přišli na to, že některé aktivátory drží svůj výstup pouze v jedné z krajních hodnot a téměř nikdy se nepřeklopí. Kobinace vah spojů v řetězcích vedoucích do takovýchto aktivátorů vedla k nashromáždění příliš velkého kladného nebo záporného potenciálu. Díky tomu FPNN nereagovalo na některé vstupní vektory správně a docházelo ke shoršení výsledků. Tento problém jsme se pokusili vyřešit manipulací s prahy a dosáhnout tak lepších výsledků.

Narozdíl od dříve představených metod, v nichž jsme použili pouze statické informace o vahách a prazích, v této metodě používáme dynamické informace získané za běhu FPNN nad trénovací datovou sadou. Navrhované metody jsou aplikovány na výsledky mapování dříve představenými metodami. Jedná se tedy o další rozšíření původních metod.

5.6.1 Použití střední hodnoty rozsahu potenciálu

V této metodě jsme u všech aktivátorů změřili, v jakém inetravalu se pohybovaly hodnoty nashromážděných potenciálů napříč celou datovou sadou. V těchto intervalech jsme nalezli střed a získané hodnoty jsme použili jako nové hodnoty prahů. Toto rozšíření budeme dále označovat suffixem TMID za názvem metody.

Toto rozšíření jsme aplikovali na úspěšné kombinace metod z předchozích částí a na metodu ARIT. Výpočty pro úpravu prahů jsme prováděli nad trénovací datovou sadou. S výsledným FPNN jsme experimentovali nad oběma datovými sadami. Tabulka 5.13 shrnuje výsledky experimentů nad trénovací datovou sadou, a tabulka 5.14 nad testovací datovou sadou. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod a rozšíření dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Tabulky 5.15 a 5.16 obsahují ty metody, jež dávaly lepší výsledky, než jakých bylo dosaženo v části s jednoduchými nekombinovanými metodami. Ve sloupci Nárůst [%] je v těchto tabulkách uvedeno zlepšení výsledků oproti nejlepšímu dosaženému výsledku předchozí části.

Tabulka 5.17 pak říká, jakého zlepšení dosáhly rozšířené metody oproti svým nerozšířeným variantám. Jsou vypsány jen hodnoty zlepšení, tam kde uvedeno není (symbol pomlčky) k žádnému zlepšení nedošlo.

Nad trénovací datovou sadou dosáhla rozšířená kombinace metod PVAL_DP + PROD_IP + DIST_DP + TMID celkové zlepšení o 8.35533 % ve srovnání s dosud nejúspěšnější jednoduchou metodou PROD_DP. Kromě této kombinace dosáhlo lepších výsledků dalších sedm rozšířených kombinací metod. Nad testovací datovou sadou bylo dosaženo zlepšení o 0.52233 %. Celkem tři metody dosáhly zlepšení výsledků nad testovacími daty.

Celkem deseti metodám přineslo rozšíření zlepšení výsledků nad trénovacími daty. Nejlepším dosaženým zlepšením je 3.99999 % u metody PVAL_DP + PROD_IP + WEIG_IP + DIST_DP. Nad testovací datovou sadou pomohlo rozšíření dosáhnout lepších výsledků celkem deseti metodám, přičemž nejvíce to bylo o 8.0 % u metody PVAL_DP + PROD_IP + WEIG_IP + DIST_DP.

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_IP+DIST_DP+TMID	283	100	73.890
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TMID	280	103	73.107

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+DIST_DP+TMID	279	104	72.845
PVAL_DP+WEIG_IP+DIST_DP+TMID	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP+TMID	259	124	67.624
PVAL_IP+PROD_DP+TMID	253	130	66.057
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TMID	252	131	65.796
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TMID	251	132	65.535
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TMID	250	133	65.274

Tabulka 5.13: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TMID	283	100	73.890
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TMID	282	101	73.629
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TMID	281	102	73.368
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TMID	280	103	73.107
PVAL_IP+PROD_DP+WEIG_IP+TMID	276	107	72.062
PVAL_DP+WEIG_IP+DIST_DPv	275	108	71.801
PVAL_DP+DIST_DP+TMID	274	109	71.540
PVAL_DP+PROD_IP+DIST_DP+TMID	272	111	71.018
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TMID	270	113	70.496

Tabulka 5.14: Porovnání metod nad testovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_IP+DIST_DP+TMID	73.89033	8.35533
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TMID	73.10704	7.57204
PVAL_DP+DIST_DP+TMID	72.84595	7.31095
PVAL_DP+WEIG_IP+DIST_DP+TMID	72.32375	6.78875
PVAL_IP+PROD_DP+WEIG_IP+TMID	67.62402	2.08902
PVAL_IP+PROD_DP+TMID	66.05744	0.52244
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TMID	65.79634	0.26134
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TMID	65.53524	0.00024

Tabulka 5.15: Zlepšení metod nad trénovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TMID	73.89033	0.52233
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TMID	73.62924	0.26124
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TMID	73.36814	0.00014

Tabulka 5.16: Zlepšení metod nad testovacími daty

Metoda	Tr. zlepšení[%]	Ts. zlepšení[%]
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	3.99999	8.00000

Metoda	Tr. zlepšení[%]	Ts. zlepšení[%]
PVAL_IP+PROD_DP+WEIG_IP	2.00000	0.00065
PVAL_DP+DIST_DP	0.00095	0.00045
PVAL_DP+WEIG_IP+DIST_DP	0.00075	0.00055
PVAL_IP+PROD_DP	0.00043	-
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	0.00033	3.99999
PVAL_DP+PROD_IP+DIST_DP	0.00033	0.00027
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	0.00024	0.00023
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	0.00015	0.00014
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	-	0.00033

Tabulka 5.17: Zlepšení dosažené rozšířením TMID nad trénovacími (Tr.) a testovacími (Ts.) daty

5.6.2 Použití aritmetického průměru potenciálů

V této mětodě jsme u všech aktivátorů vypočetli arimtetický průměr všech potenciálů, kterých aktivátory nabývaly a výsledné hodnoty jsme použili jako nové hodnoty prahů. Toto rozšíření budeme dále označovat suffixem TAVG za názvem metody.

Toto rozšíření jsme aplikovali na úspěšné kombinace metod z předchozích částí a na metodu ARIT. Výpočty pro úpravu prahů jsme prováděli nad trénovací datovou sadou. S výsledným FPNN jsme experimentovali nad oběma datovými sadami. Tabulka 5.18 shrnuje výsledky experimentů nad trénovací datovou sadou, a tabulka 5.19 nad testovací datovou sadou. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod a rozšíření dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Tabulky 5.20 a 5.21 obsahují ty metody, jež dávaly lepší výsledky, než jakých bylo dosaženo v části s jednoduchými nekombinovanými metodami. Ve sloupci Nárůst [%] je v těchto tabulkách uvedeno zlepšení výsledků oproti nejlepšímu dosaženému výsledku předchozí části.

Tabulka 5.22 pak říká, jakého zlepšení dosáhly rozšířené metody oproti svým nerozšířeným variantám. Jsou vypsány jen hodnoty zlepšení, tam kde uvedeno není (symbol pomlčky) k žádnému zlepšení nedošlo.

Nad trénovací datovou sadou dosáhla rozšířená kombinace metod PVAL_DP + PROD_IP + DIST_DP + TAVG celkové zlepšení o 8.35533 % ve srovnání s dosud nejúspěšnější jednoduchou metodou PROD_DP. Kromě této kombinace dosáhlo lepších výsledků dalších sedm rozšířených kombinací metod. Nad testovací datovou sadou bylo dosaženo zlepšení o 0.52233 %. Celkem tři metody dosáhly zlepšení výsledků nad testovacími daty.

Celkem deseti metodám přineslo rozšíření zlepšení výsledků nad trénovacími daty. Nejlepším dosaženým zlepšením je 3.99999 % u metody PVAL_DP + PROD_IP + WEIG_IP + DIST_DP. Nad testovací datovou sadou pomohlo rozšíření dosáhnout lepších výsledků celkem deseti metodám, přičemž nejvíce to bylo o 8.0 % u metody PVAL_DP + PROD_IP + WEIG_IP + DIST_DP.

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_IP+DIST_DP+TAVG	283	100	73.890
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TAVG	280	103	73.107
PVAL_DP+DIST_DP+TAVG	279	104	72.845

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+WEIG_IP+DIST_DP+TAVG	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP+TAVG	259	124	67.624
PVAL_IP+PROD_DP+TAVG	253	130	66.057
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TAVG	252	131	65.796
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TAVG	251	132	65.535
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TAVG	250	133	65.274

Tabulka 5.18: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TAVG	283	100	73.890
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TAVG	282	101	73.629
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TAVG	281	102	73.368
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TAVG	280	103	73.107
PVAL_IP+PROD_DP+WEIG_IP+TAVG	276	107	72.062
PVAL_DP+WEIG_IP+DIST_DP+TAVG	275	108	71.801
PVAL_DP+DIST_DP+TAVG	274	109	71.540
PVAL_DP+PROD_IP+DIST_DP+TAVG	272	111	71.018
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TAVG	270	113	70.496

Tabulka 5.19: Porovnání metod nad testovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_IP+DIST_DP+TAVG	73.89033	8.35533
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TAVG	73.10704	7.57204
PVAL_DP+DIST_DP+TAVG	72.84595	7.31095
PVAL_DP+WEIG_IP+DIST_DP+TAVG	72.32375	6.78875
PVAL_IP+PROD_DP+WEIG_IP+TAVG	67.62402	2.08902
PVAL_IP+PROD_DP+TAVG	66.05744	0.52244
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TAVG	65.79634	0.26134
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TAVG	65.53524	0.00024

Tabulka 5.20: Zlepšení metod nad trénovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TAVG	73.89033	0.52233
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TAVG	73.62924	0.26124
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TAVG	73.36814	0.00014

Tabulka 5.21: Zlepšení metod nad testovacími daty

Metoda	Tr. zlepšení[%]	Ts. zlepšení[%]
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	3.99999	8.00000
PVAL_IP+PROD_DP+WEIG_IP	2.00000	0.00065

Metoda	Tr. zlepšení[%]	Ts. zlepšení[%]
PVAL_DP+DIST_DP	0.00095	0.00045
PVAL_DP+WEIG_IP+DIST_DP	0.00075	0.00055
PVAL_IP+PROD_DP	0.00043	-
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	0.00033	3.99999
PVAL_DP+PROD_IP+DIST_DP	0.00033	0.00027
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	0.00024	0.00023
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	0.00015	0.00014
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	-	0.00033

Tabulka 5.22: Zlepšení dosažené rozšířením TAVG nad trénovacími (Tr.) a testovacími (Ts.) daty

5.7 Současný přepočet vah i prahů

Jako další metodu převodu neuronové sítě na FPNN jsme zvolili současný přepočet vah i prahů s ohledem na chování vzorových neuronů v referenční síti. Naším cílem bylo co nejvíce přiblížit odezvu aktivátoru na konkrétní vektor odezvě půdvoního vzorového neuronu na stejný vektor.

Pro získání daného chování jsme se rozhodli použít algoritmus Nelder-Mead popsaný v sekci 2.4. Jako chybovou funkci, kterou jsme algortimem minimalizovali, jsme zvolili odchylku výstupů aktivátorů od vzorových neuronů skryté vrstvy nad jednotlivými vektory trénovací datové sady. Jako výchozí bod jsme brali FPNN namapované úspěšnými metodami mapování vah popsaných v předchozích kapitolách. Experimentovali jsme se dvěma metodami popsanými v následujících podkapitolách. Navrhované metody jsou aplikovány na výsledky mapování dříve představenými metodami. Jedná se tedy o další rozšíření původních metod.

5.7.1 Přepočet celé vrtsvy najednou

Toto rozšíření budeme dále označovat suffixem TLAY za názvem metody. V této metodě jsme počítali celou skyrtou vrstvu najednou, tedy všechny příslušné váhy a také prahy všech aktivátorů. Simplex byl tvořen hodnotami všech vah spojů a všech prahů. Chybová funkce byla definována jako suma součtů odchylek výstupů všech aktivátorů od výstupů původních neuronů přes všechny vektory trénovací datové sady. Popisuje to rovnice (5.16), kde T je množina trénovacích vektorů, H je množina indexů neuronů skryté vrstvy (aktivátory mají stejné indexy), o_i^t je výstup aktivátoru i pro vektor t a d_i^t je výstup původního neuronu pro vektor t.

$$Err = \sum_{t \in T} \sum_{i \in H} o_i^t - d_i^t \tag{5.16}$$

Jako výchozí bod pro vytvoření počátečního simplexu byla použita FPNN namapovaná metodami, kterým se dodsud podařilo dosáhnout zlepšení, tedy metod uvedených v tabulkách 5.8 a 5.9.

Toto rozšíření jsme aplikovali na úspěšné kombinace metod z předchozích částí a na metodu ARIT. Výpočty pro úpravu prahů jsme prováděli nad trénovací datovou sadou. S výsledným FPNN jsme experimentovali nad oběma datovými sadami.

Tabulka 5.23 shrnuje výsledky experimentů nad trénovací datovou sadou, a tabulka 5.24 nad testovací datovou sadou. Tabulky jsou seřazené podle procentuální shody. Některé

kombinace metod a rozšíření dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Tabulky 5.23 a 5.24 shrnují výsledky experimentů. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod a rozšíření dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Tabulky 5.25 a 5.26 obsahují ty metody, jež dávaly lepší výsledky, než jakých bylo dosaženo v části s jednoduchými nekombinovanými metodami. Ve sloupci Nárůst [%] je v těchto tabulkách uvedeno zlepšení výsledků oproti nejlepšímu dosaženému výsledku předchozí části. Tabulka 5.27 uvádí zlepšení dosažené rozšířením oproti nerozšířeným verzím metod. Jsou vypsány jen hodnoty zlepšení, tam kde uvedeno není (symbol pomlčky) k žádnému zlepšení nedošlo.

Nad trénovací datovou sadou dosáhla rozšířená kombinace metod PVAL_IP+PROD_DP +WEIG_IP+TLAY celkové zlepšení o 3.13340 % ve srovnání s dosud nejúspěšnější jednoduchou metodou PROD_DP. Kromě této kombinace dosáhlo lepších výsledků dalších pět rozšířených kombinací metod. Nad testovací datovou sadou bylo dosaženo zlepšení o 8.61643 %. Celkem čtyři metody dosáhly zlepšení výsledků nad testovacími daty.

Celkem pěti metodám přineslo rozšíření zlepšení výsledků nad trénovacími daty. Nejlepším dosaženým zlepšením je 1.04440 % u metody PVAL_IP + PROD_DP + WEIG_IP. Nad testovací datovou sadou pomohlo rozšíření dosáhnout lepších výsledků celkem sedmi metodám, přičemž nejvíce to bylo o 3.13324 % u metody PVAL_DP + PROD_IP + WEIG_IP + DIST_DP.

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_IP+PROD_DP+WEIG_IP+TLAY	263	120	68.668
PVAL_DP+WEIG_IP+DIST_DP+TLAY	259	124	67.624
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TLAY	257	126	67.101
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TLAY	254	129	66.318
PVAL_IP+PROD_DP+TLAY	253	130	66.057
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TLAY	252	131	65.796
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TLAY	250	133	65.274

Tabulka 5.23: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TLAY	284	99	74.151
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TLAY	283	100	73.890
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TLAY	282	101	73.629
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TLAY	281	102	73.368
PVAL_IP+PROD_DP+WEIG_IP+TLAY	244	139	63.707
PVAL_IP+PROD_DP+TLAY	233	150	60.835

Tabulka 5.24: Porovnání metod nad testovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_IP+PROD_DP+WEIG_IP+ TLAY	68.66840	3.13340
PVAL_DP+WEIG_IP+DIST_DP+ TLAY	67.62402	2.08902
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TLAY	67.10182	1.56682

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TLAY	66.31853	0.78353
PVAL_IP+PROD_DP+TLAY	66.05744	0.52244
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TLAY	65.79634	0.26134

Tabulka 5.25: Zlepšení metod nad trénovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TLAY	74.15143	8.61643
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TLAY	73.89033	8.35533
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TLAY	73.62924	8.09424
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TLAY	73.36814	7.83314

Tabulka 5.26: Zlepšení metod nad testovacími daty

Metoda	Tr. zlepšení[%]	Ts. zlepšení[%]
PVAL_IP+PROD_DP+WEIG_IP	1.04440	-
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	0.26134	-
PVAL_IP+PROD_DP	0.00043	-
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	0.00033	0.78333
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	0.00015	0.00014
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	-	3.13324
PVAL_DP+PROD_IP+DIST_DP	-	2.61123
PVAL_DP+DIST_DP	-	2.35032
PVAL_DP+WEIG_IP+DIST_DP	-	2.08933
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	-	0.26143

Tabulka 5.27: Zlepšení dosažené rozšířením TLAY nad trénovacími (Tr.) a testovacími (Ts.) daty

5.7.2 Postupný přepočet

Toto rozšíření budeme dále označovat suffixem TACT za názvem metody. V této metodě jsme počítali každý aktivátor skryté vrstvy samostatně. Simplex byl tvořen vahami všech spojů v řetezcích vedoucích do aktivátoru a jeho prahem. Výsledné hodnoty vah spojů byly získány jako vážený průměr hodnot vypočtených pro každý aktivátor. Váhy byly vypočteny ze vzdálenosti aktivátoru a spoje. Chybová funkce byla definována jako suma součtů odchylek výstupů daného aktivátoru od výstupů původního neuronu přes všechny vektory trénovací datové sady. Popisuje to rovnice (5.17) pro aktivátor s indexem i (původní neuron má stejný index), kde T je množina trénovacích vektorů, o_i^t je výstup aktivátoru i pro vektor t a d_i^t je výstup původního neuronu pro vektor t.

$$Err = \sum_{t \in T} o_i^t - d_i^t \tag{5.17}$$

Toto rozšíření jsme aplikovali na úspěšné kombinace metod z předchozích částí a na metodu ARIT. Výpočty pro úpravu prahů jsme prováděli nad trénovací datovou sadou. S

výsledným FPNN jsme experimentovali nad oběma datovými sadami.

Tabulka 5.28 shrnuje výsledky experimentů nad trénovací datovou sadou, a tabulka 5.29 nad testovací datovou sadou. Tabulky jsou seřazené podle procentuální shody. Některé kombinace metod a rozšíření dosáhly vzájemně shodných výsledků. Tabulky obsahují z takových skupin vždy jen jednu.

Tabulky 5.30 a 5.31 obsahují ty metody, jež dávaly lepší výsledky, než jakých bylo dosaženo v části s jednoduchými nekombinovanými metodami. Ve sloupci Nárůst [%] je v těchto tabulkách uvedeno zlepšení výsledků oproti nejlepšímu dosaženému výsledku předchozí části. Tabulka 5.32 uvádí zlepšení dosažené rozšířením oproti nerozšířeným verzím metod. Jsou vypsány jen hodnoty zlepšení, tam kde uvedeno není (symbol pomlčky) k žádnému zlepšení nedošlo.

Nad trénovací datovou sadou dosáhla rozšířená kombinace metod PVAL_DP + DIST_DP + TACT celkové zlepšení o 9.92191 % ve srovnání s dosud nejúspěšnější jednoduchou metodou PROD_DP. Kromě této kombinace dosáhlo lepších výsledků dalších pět rozšířených kombinací metod. Nad testovací datovou sadou bylo dosaženo zlepšení o 7.83314 %. Celkem šest metod dosáhlo zlepšení výsledků nad testovacími daty.

Celkem šesti metodám přineslo rozšíření zlepšení výsledků nad trénovacími daty. Nejlepším dosaženým zlepšením zde je 4.69975 % u metody PVAL_IP + PROD_DP + WEIG_IP. Nad testovací datovou sadou pomohlo rozšíření dosáhnout lepších výsledků celkem sedmi metodám, přičemž nejvíce to bylo o 0.78374 % u metody PVAL_DP + DIST_DP.

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+DIST_DP+TACT	289	94	75.456
PVAL_DP+WEIG_IP+DIST_DP+TACT	288	95	75.195
PVAL_IP+PROD_DP+WEIG_IP+TACT	277	106	72.323
PVAL_IP+PROD_DP+TACT	259	124	67.624
PVAL_DP+PROD_IP+DIST_DP+TACT	258	125	67.362
ARIT+TACT	251	132	65.535
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TACT	250	133	65.274
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TACT	248	135	64.751

Tabulka 5.28: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TACT	281	102	73.368
ARIT+TACT	280	103	73.107
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TACT	279	104	72.845
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TACT	278	105	72.584
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TACT	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP+TACT	264	119	68.929
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TACT	248	135	64.751
PVAL_DP+PROD_IP+DIST_DP+TACT	245	138	63.968
PVAL_IP+PROD_DP+TACT	243	140	63.446

Tabulka 5.29: Porovnání metod nad testovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_DP+DIST_DP+TACT	75.45691	9.92191
PVAL_DP+WEIG_IP+DIST_DP+TACT	75.19582	9.66082
PVAL_IP+PROD_DP+WEIG_IP+TACT	72.32375	6.78875
PVAL_IP+PROD_DP+TACT	67.62402	2.08902
PVAL_DP+PROD_IP+DIST_DP+TACT	67.36292	1.82792
ARIT+TACT	65.53524	0.00024

Tabulka 5.30: Zlepšení metod nad trénovacími daty

Metoda	Shoda [%]	Nárůst [%]
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TACT	73.36814	7.83314
ARIT+TACT	73.10704	7.57204
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TACT	72.84595	7.31095
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TACT	72.58485	7.04985
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TACT	72.32375	6.78875
PVAL_IP+PROD_DP+WEIG_IP+TACT	68.92950	3.39450

Tabulka 5.31: Zlepšení metod nad testovacími daty

Metoda	Tr. zlepšení[%]	Ts. zlepšení[%]
PVAL_IP+PROD_DP+WEIG_IP	4.69975	-
PVAL_DP+WEIG_IP+DIST_DP	2.87282	0.52275
PVAL_DP+DIST_DP	2.61190	0.78374
PVAL_IP+PROD_DP	1.56701	-
ARIT	0.26108	-
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	0.00015	0.00014

Tabulka 5.32: Zlepšení dosažené rozšířením TACT nad trénovacími (Tr.) a testovacími (Ts.) daty

5.8 Srovnání

V předcházejících sekcích jsme představili několik metod mapování neuronové sítě na FPNN mřížové struktury. V této kapitole provedeme srovnání výsledků představených metod se všemi rozšířeními. Sloupec *Shoda [%]* obsahuje míru shody klasifikace FPNN a referenční neuronové sítě. Sloupec *Rozdíl [%]* udává rozdíl výsledné míry shody dané metody od míry shody metody ARIT, která byla navržena a implementována jako první, a proto ji použijeme pro srovnání zlepšení či zhoršení novějších metod. Experimentů bylo provedeno více než čtyři tisíce a jejich kompletní výpis zde není možný. Velmi mnoho kombinovaných metod však dosáhlo numericky stejných výsledků, proto byla z každé takové skupiny vybrána jediná jako reprezentant. Rozšířené metody byly zaneseny do tabulky jen pokud dosáhly zlepšení výsledků. Kompletní výpis experimentů je možné najít na přiloženém CD v souborech `/Results/complete_sum_table_train.pdf` a `/Results/complete_sum_table_test.pdf`.

Z tabulek vyplývá, že různé kombinace metod dosáhly znatelného zlepšení výsledků.

Nejlepší výsledek čistě kombinované metody je zlepšení o 8.616% nad trénovacími daty a 8.355% nad testovacími daty. Rozšíření pracující se společnými úpravami vah a prahů, TLAY a TACT, také přinesly jisté zlepšení oproti jednoduchým metodám i oproti kombinovaným metodám. Nejlepší výsledek nad trénovacími daty metody rozšířené o TACT je 10.182% a metoda rozšířená o TLAY dosáhla zlepšení 3.394%. Nad testovacími daty dosáhla nejlepších výsledků metoda rozšířena o TLAY se 8.87699% zlepšením a u rozšíření TACT bylo dosaženo nejlepšího zlepšení o 8.09399%. Rozšíření TLAY a TACT měly tedy obě pozitivní vliv na výsledky.

Dalším navrženým rozšířením byly TMID a TAVG založené na jednoduché úpravě prahů. I tyto metody přinesly zlepšení jak proti jednoduchým metodám, tak zlepšily výsledky i řady kombinovaných metod. Obě metody dosáhly shodného zlepšení výsledků. Nad trénovacími daty byl nejlepší výsledek z lepšení o 8.616%. Nad testovacími daty pak 8.616%.

Metoda	Shoda [%]	Rozdíl [%]
PVAL_DP+DIST_DP+TACT	75.456	10.18200
PVAL_DP+WEIG_IP+DIST_DP+TACT	75.195	9.92099
PVAL_DP+PROD_IP+DIST_DP+TAVG	73.89	8.616
PVAL_DP+PROD_IP+DIST_DP+TMID	73.89	8.616
PVAL_DP+PROD_IP+DIST_DP	73.89	8.616
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TAVG	73.107	7.83299
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TMID	73.107	7.83299
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	73.107	7.83299
PVAL_DP+DIST_DP+TAVG	72.845	7.57099
PVAL_DP+DIST_DP+TMID	72.845	7.57099
PVAL_DP+DIST_DP	72.845	7.57099
PVAL_DP+WEIG_IP+DIST_DP+TAVG	72.323	7.04899
PVAL_DP+WEIG_IP+DIST_DP+TMID	72.323	7.04899
PVAL_IP+PROD_DP+WEIG_IP+TACT	72.323	7.04899
PVAL_DP+WEIG_IP+DIST_DP	72.323	7.04899
PVAL_IP+PROD_DP+WEIG_IP+TLAY	68.668	3.39400
PVAL_IP+PROD_DP+WEIG_IP+TAVG	67.624	2.34999
PVAL_IP+PROD_DP+WEIG_IP+TMID	67.624	2.34999
PVAL_IP+PROD_DP+TACT	67.624	2.34999
PVAL_DP+WEIG_IP+DIST_DP+TLAY	67.624	2.34999
PVAL_IP+PROD_DP+WEIG_IP	67.624	2.34999
PVAL_DP+PROD_IP+DIST_DP+TACT	67.362	2.08799
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TLAY	67.101	1.82699
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TLAY	66.318	1.04399
PVAL_IP+PROD_DP+TAVG	66.057	0.78300
PVAL_IP+PROD_DP+TMID	66.057	0.78300
PVAL_IP+PROD_DP+TLAY	66.057	0.78300
PVAL_IP+PROD_DP	66.057	0.78300
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TAVG	65.796	0.52200
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TMID	65.796	0.52200
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TLAY	65.796	0.52200
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	65.796	0.52200
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TAVG	65.535	0.26099
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TMID	65.535	0.26099

Metoda	Shoda [%]	Rozdíl [%]
ARIT+TACT	65.535	0.26099
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	65.535	0.26099
PROD_DP	65.535	0.26099
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TAVG	65.274	0.0
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TMID	65.274	0.0
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TACT	65.274	0.0
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TLAY	65.274	0.0
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	65.274	0.0
ARIT	65.274	0.0
PVAL_DP+PROD_DP+WEIG_DP+DIST_IP	65.013	-0.26099
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TACT	64.751	-0.52299
WEIG_IP+DIST_DP	61.879	-3.39500
PVAL_IP+PROD_IP+DIST_DP	60.052	-5.22200
PVAL_IP+PROD_IP+WEIG_IP+DIST_DP	59.268	-6.006
PVAL_IP+WEIG_IP+DIST_DP	58.746	-6.52799
PVAL_IP+DIST_DP	58.224	-7.05000
PVAL_DP+PROD_IP	53.785	-11.48900
PVAL_DP+PROD_IP+WEIG_IP	53.263	-12.01100
PVAL_DP+WEIG_IP	41.514	-23.75999

Tabulka 5.33: Porovnání metod nad trénovacími daty

Metoda	Shoda [%]	Rozdíl [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TLAY	74.151	8.87699
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TAVG	73.89	8.616
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TMID	73.89	8.616
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TLAY	73.89	8.616
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	73.89	8.616
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TAVG	73.629	8.35500
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TMID	73.629	8.35500
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TLAY	73.629	8.35500
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	73.629	8.35500
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TAVG	73.368	8.09399
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TMID	73.368	8.09399
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TACT	73.368	8.09399
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP+TLAY	73.368	8.09399
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	73.368	8.09399
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TAVG	73.107	7.83299
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TMID	73.107	7.83299
ARIT+TACT	73.107	7.83299
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	73.107	7.83299
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP+TACT	72.845	7.57099
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP+TACT	72.584	7.31000
PVAL_IP+PROD_DP+WEIG_DP+DIST_IP	72.584	7.31000
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP+TACT	72.323	7.04899
PROD_DP+WEIG_DP+DIST_IP	72.323	7.04899

Metoda	Shoda [%]	Rozdíl [%]
PVAL_IP+PROD_DP+WEIG_IP+TAVG	72.062	6.78799
PVAL_IP+PROD_DP+WEIG_IP+TMID	72.062	6.78799
PVAL_IP+PROD_DP+WEIG_IP	72.062	6.78799
PVAL_DP+WEIG_IP+DIST_DP+TAVG	71.801	6.52700
PVAL_DP+WEIG_IP+DIST_DPv	71.801	6.52700
PVAL_DP+WEIG_IP+DIST_DP	71.801	6.52700
PVAL_DP+DIST_DP+TAVG	71.54	6.26600
PVAL_DP+DIST_DP+TMID	71.54	6.26600
PVAL_DP+DIST_DP	71.54	6.26600
PVAL_DP+PROD_IP+DIST_DP+TAVG	71.018	5.744
PVAL_DP+PROD_IP+DIST_DP+TMID	71.018	5.744
PVAL_DP+PROD_IP+DIST_DP	71.018	5.744
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TAVG	70.496	5.22199
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TMID	70.496	5.22199
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	70.496	5.22199
PVAL_IP+PROD_DP+WEIG_IP+TACT	68.929	3.65500
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP+TACT	64.751	-0.52299
PVAL_DP+PROD_IP+DIST_DP+TACT	63.968	-1.30599
PVAL_IP+PROD_DP+WEIG_IP+TLAY	63.707	-1.56700
PVAL_IP+PROD_DP+TACT	63.446	-1.82800
PVAL_IP+PROD_IP+WEIG_IP+DIST_DP	62.663	-2.61100
PVAL_IP+DIST_DP	62.14	-3.13400
PVAL_IP+PROD_DP+TLAY	60.835	-4.439
PROD_IP+DIST_DP	60.835	-4.439
PROD_IP+WEIG_IP+DIST_DP	60.052	-5.22200
PVAL_IP+WEIG_IP+DIST_DP	59.791	-5.48300
PVAL_DP+PROD_IP	58.485	-6.78900
PVAL_DP+PROD_IP+WEIG_IP	56.919	-8.35500
WEIG_IP+DIST_DP	55.613	-9.66100
PVAL_DP+WEIG_IP	36.553	-28.72100

Tabulka 5.34: Porovnání metod nad testovacími daty

Kapitola 6

Učení FPNN algoritmem backpropagation

FPNN zkoumaného mřížového typu mají do jisté míry shodné strukturální znaky jako dopředné vrstvené neuronové sítě - skládají se z vrstev a jsou nerekurentní. Protože dopředné vrstvené neuronové sítě jsou tradičně trénovány algoritmem backpropagation (viz. 2.2.1), je možné položit otázku, zda by bylo možné tímto algoritmem trénovat i FPNN. V případě, že by FPNN bylo zkonstruováno stejně jako sama neuronová síť, bez sdílení spojů, pak o tom není pochyb. V případě mřížové struktury ale narážíme právě na problém aproximace více synaptických propojení stejnými spoji.

Tradiční algoritmus backpropagation nepočítá se sdílením propojení. V tomto algoritmu jsou vypočtené chyby propagovány zpět do předcházející vrtsvy skrze všechny synapse a jsou váženy jejich váhami, přičemž se každou synapsí propaguje jen jedna chyba. Z hodnot takto akumulovaných chyb je pak pro váhu každé synapse zvlášť vypočítána její potřebná modifikace. V FPNN ale takový přístup koliduje se sdílením spojů mezi původními synapsemi. Při zpětné propagaci přes každý spoj prochází více než jen jedna odchylka. Procházející odchylky nejsou váženy jen jednou vahou, ale jsou násobeny vahou každého spoje jímž projdou, tedy několikrát za sebou. Pro každý spoj se počítá více modifikací, pro každou původní synapsi jedna. A tyto modifikace musejí být aplikovány společně.

Je zde tedy potřeba řešit stejný problém jako u mapování vah synapsí na váhy sdílených spojů v FPNN. Stejně jako u mapování je třeba najít kompromis mezi vypočtenými modifikacemi a tento kompromis použít jako výslednou modifikaci. Metod hledání kompromisu můžeme navrhnout více. Můžeme využívat stejných principů jako při mapování.

S ohledem na tuto úlohu jsme vytvořili nový modul `Teacher` a implementovali v něm algoritmus backpropagation tak jak je popsán v následující podsekci. Tato implementace počítá jen s jednou metodou hledání kompromisu, může být ale rozšířena o další. Tato úloha nabízí velký prostor pro výzkum a experimentování, to je ale již nad rámec této práce a jejího zaměření na mapování již naučených neuronových sítí.

6.1 Implementovaná verze algoritmu backpropagation

Algoritmus použitý pro učení pracuje podobně jako algoritmus užívaný k učení neuronových sítí. Byl ale přizpůsoben specifikům mřížové strultury FPNN. Zatímco pro každou synapsi je v originálním algoritmu vypočtena jediná odchylka, spoje v FPNN počítají s několika odchylkami. Pro průchodu algoritmu backpropagation má každý spoj vypočteno několik

modifikací, které spojuje do jedné aplikací aritmetického průměru. Po předložení vstupního vektoru a výpočtu výstupního vekoru vypočítají výstupní aktivátory j hodnotu δ_j:

$$\delta_j = (t_j - o_j)o_j(1 - o_j) \tag{6.1}$$

kde t_j je požadovaný výstup a o_j skutečný výstup. Aktivátory i ve skryté vrstvě vypočítají hodnotu δ_i:

$$w_{ij} = \prod_{w \in Chain_{ij}} w \tag{6.2}$$

$$\delta_i = o_i(1 - o_i) \sum_j \delta_j w_{ij} \tag{6.3}$$

kde o_i je výstup aktivátoru i, δ_j je hodnota vypočtená (a propagovaná zpět) aktivátorem j v následující vrtsvě, $Chain_{ij}$ je řetězec spojů mezi aktivátory i a j a w_{ij} (6.2) je jejich součin. Při zpětném šíření jsou hodnoty δ ukládány ve spojích, přes které jsou propagovány. Spoje pak vypočítají hodnoty změn parametru W v kroku n:

$$\Delta W_{ij}^n = \eta \delta_j o_i + \alpha w_{ij}^{n-1} \tag{6.4}$$

kde η je koeficient učení a α je koeficient momentu. Tyto hodnoty jsou vypočteny pro každý končící řetězec. Jejich aritmetický průměr je dosazen jako nová hodnota W.

6.2 Experimenty

Uvedený algoritmus jsme spustili na FPNN úlohy Diabetes s parametry $\eta = 0.9$ a $\alpha = 0.5$, což je podle [21] doporučená počáteční volba. Proběhlo 1000 epoch učení.

Úloha	Datová sada	Shoda	Neshoda	Shoda [%]
Diabetes	Trénovací	250	133	65.274
Diabetes	Testovací	248	135	64.752
Two Spiral	Trénovací	98	95	50.777
Two Spiral	Testovací	97	96	50.259

Tabulka 6.1: Výsledky experimentů učení nad oběma úlohami a datovými sadami

Kapitola 7

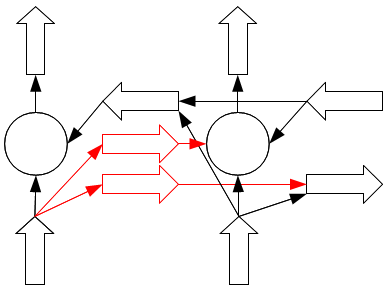
Zlepšení aproximace přidáním rendundatních zdrojů

Pro dosažení lepších výsledků se můžeme pokusit zvýšit výpočetní sílu FPNN porušením jeho struktury tak, že zavedeme jistou redundanci. V případě aktivátorů žádná redundance není možná, protože jejich počet je dán počtem neuronů původní sítě. Protože ale spojů je menší počet než synapsí referenční sítě a každý spoj aproximuje několik synapsí najednou, můžeme redundanci zavést právě u nich a docílit toho, že spoje budou aproximovat méně synapsí zároveň. Tato kapitola se zaměřuje na jednu z možností zavedení redundance do spojů FPNN, kterou jsme navrhli jako další možné rozšíření a námět pokračování práce.

7.1 Zdvojení spojů

Jendou z možností zavedení redundance je rozdvojení některých spojů. Vhodným kandidátem na zdvojení v rámci řetězce je terminál, protože v něm se tok výpočtu rozdvojuje - část řetězců zde končí a druhá část pokračuje sledem dále. Při zdvojení terminálu vytvoříme nový terminál, přes který půjdou všechny končící řetězce a původní spoj ve sledu bude přenášet řetězce, které ještě nekončí. Nebude tedy terminálem. Rozdvojení spojů ukazuje obrázek 7.1, kde byl původní spoj (první spoj v propojovacím sledu) nahrazen dvěma spoji.

Do modulu `FPNNWeightsMapper` byla implementována metoda zvojování spojů. Metoda umožňuje duplikovat více spojů najednou. Duplikace u některých spojů nemá smysl. Především spoje ležící na konci sledu, tedy spoje, které nemají jiný spoj jako následníka a mají pouze jednoho předchůdce, není třeba rozvojovat. Aplikace tedy umožňuje duplikovat množinu spojů z propojovacího sledu a znovu celé FPNN přemapovat. Celý proces proníhá



Obrázek 7.1: Zdvojený spoj

následovně:

1. FPNN se běžným postupem namapuje
2. Vypočítá se chyba aproximace
3. Spoje propojovacího sledu se seřadí sestupně podle chyby
4. Vybere se počet spojů předepsaný parametrem
5. Pro každý spoj se vyhodnotí, zda je možné jej rozdvojit, pokud ne, jsou vyřazeny.
6. Pro každý vybraný spoje je vytvořen duplikát. Pokud původní spoj:
   (a) Má jiný spoj jako následníka, tak je odpojen od aktivátoru a místo něj je připojen jeho duplikát. Na vstup duplikátu jsou připojeni všichni předchůdci původního aktivátoru. Původní spoj se stal neterminálem.
   (b) Nemá mezi následníky žádný spoj, tak zůstává připojen k aktivátoru duplikát je k němu připojen také. Polovina předchůdců původního spoje je převzata duplikátem. Ten tak přebírá jednu větev přicházejících řetězců.
7. Provede se nové mapování, jehož se neterminální spoje neúčastní.
8. Vypočtené parametry se propagují oběma směry propojovacím sledem.
9. Neterminály si vypočítají hodnoty všech řetězců, které přes ně procházejí.
10. Neterminály jsou po jednom namapovány postupně od spojů blíže začátku propojovacího sledu směrem ke vzdálenějším. Mapování probíhá následujícím způsobem:
    (a) Vyberou se ty řetězce, které směřují do nejbližšího aktivátoru a tedy končí v nejbližším terminálu. Pro tyto vybrané řetězce se vypočte mapování stejným způsobem jako v terminálech při běžném mapování. Neterminál se tedy chová stejně jako terminál s tím rozdílem, že terminál počítá s řetězci, které v něm končí a neterminál s řetězci končícími v nejbližším následujícím terminálu.
    (b) Neterminál propaguje nově vypočtený parametr W přes své následníky dále skrze celý zbytek sledu. Následující neterminály (které se teprve budou mapovat) si touto hodnotou aktualizují vypočtené řetězce.
11. Provede se ještě jedno mapování běžným způsobem, jehož se ale účastní pouze duplikáty a spoje na konci sledů, čímž dojde k započítání změn na neterminálech.

7.2 Závěr

Představená metoda mapování duplikovaných spojů vychází z metody ARIT a je to jen jedna z mnoha možných metod. Bylo by také možné založit mapování duplikátů na dalších metodách představených v této práci, stejně tak využití optimalizací. Vývoj těchto metod stejně jako důkladnější experimenty s navrženou metodou jsou však již nad rámec této práce, a budou náplní našeho budoucího výzkumu v této oblasti.

Kapitola 8

Závěr

V rámci této práce byly navrženy dva různé způsoby učení FPNN. Učení s učitelem pomocí algoritmu backpropagation bylo navrženo jako přirozené vyústění faktu, že FPNN jsou aproximací vrstvených dopředných sítí. Tato metoda byla implementována a byl nad ní proveden experiment, jenž u úlohy Diabetes dosáhl výsledné přesnosti 65.274% nad trénovacími daty a 64.752% nad testovacími daty. Učení FPNN úlohy Two Spiral dosáhlo nad trénovacími daty přesnost 50.777% a nad testovacimi daty 50.259%.

Dalším způsobem, na který jsme se v naší práci primárně zaměřili, bylo mapování parametrů neuronové sítě na parametry FPNN ve snaze se nabídnout možnost vytvořit FPNN z naučené neuronové sítě, aniž by byla potřeba originální trénovací datová sada. Navrhli jsme celkem deset metod mapování, se kterými jsme prováděli experimenty. Nejlepší dosažený výsledek byl 65.535% přesnost.

Pro dosažení lepších výsledků jsme osm metod založených na využití váženého průměru zkombinovali do mnoha kombinací a nad každou z nich jsme prováděli experimenty. Kombinování metod přineslo zlepšení o 8.356% s celkovou přesností 73.89% nad trénovacími daty a nad testovacími daty zlepšení o 0.522% s celkovou přesností 73.89%.

Navrhli jsme několik rozšíření původních metod a jejich kombinací. První rozšíření bylo založeno na započítávání nekončících řetězců do váženého průměru ve snaze o rozprostření chyby aproximace po celém řetězci. Navržené rozšíření nepřineslo zlepšení výsledků. Dalším rozšířením byly úpravy prahů aktivátorů na základě statistiky hodnot potenciálů, kterých nabývaly při výpočtu nad datovou sadou. Toto rozšíření přineslo zlepšení výsledků některým kombinacím metod, ale ke zlepšení celkových výsledků nedošlo. Dále jsme navrhli další dvě rozšiřující metody založené na optimalizačním algoritmu provádějící mapování a přepočet vah i prahů současně. Tato rozšíření přinesla některým kombinacím metod zlepšení v řádu procent a dosáhla zlepšení celkových výsledků o 10.182% s celkovou přesností 75.456% nad trénovacími daty. Nad testovacími daty zlepšení dosahovalo 8.877% s celkovou přesností 74.151%.

Celkově se nám tedy podařilo dosáhnout přesnost 75.456% nad trénovacími daty a 74.151% nad testovacími daty. Pro další zlepšení výsledků jsme navrhli metodu zavádění a mapování redundantních spojů do FPNN tak, aby převzaly část aproximovaných synapsí od spojů vykazujících vysoké chyby aproximace. Vývoj této metody a experimenty s ní by mohly být náplní další práce, jako metoda hledání kompromisu mezi přesností a prostorovou složitostí. Stejně tak učení algoritmem backpropagation by se mohlo stát předmětem dalšího výzkumu.

Literatura

[1] Fast Artificial Neural Network Library.
URL http://leenissen.dk/fann/wp/

[2] Byatt, D.: *Convergent Variants of the Nelder-Mead Algorithm: A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Mathematics at the University of Canterbury*. University of Canterbury, 2000.
URL http://books.google.cz/books?id=TNDLMAAACAAJ

[3] Ferreira, P.; Ribeiro, P.; Antunes, A.; aj.: Artificial Neural Networks Processor ? A Hardware Implementation Using a FPGA. In *Field Programmable Logic and Application*, *Lecture Notes in Computer Science*, ročník 3203, editace J. Becker; M. Platzner; S. Vernalde, Springer Berlin / Heidelberg, 2004, ISBN 978-3-540-22989-6, s. 1084–1086, 10.1007/978-3-540-30117-2-132.
URL http://dx.doi.org/10.1007/978-3-540-30117-2-132

[4] Gao, F.; Han, L.: Implementing the Nelder-Mead Simplex Algorithm with Adaptive Parameters. *Comput. Optim. Appl.*, ročník 51, č. 1, Leden 2012: s. 259–277, ISSN 0926-6003, doi:10.1007/s10589-010-9329-3.
URL http://dx.doi.org/10.1007/s10589-010-9329-3

[5] Girau, B.: FPNA: Applications and Implementations. In *FPGA Implementations of Neural Networks*, editace A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, s. 103–136, 10.1007/0-387-28487-7-4.
URL http://dx.doi.org/10.1007/0-387-28487-7-4

[6] Girau, B.: FPNA: Concepts and Properties. In *FPGA Implementations of Neural Networks*, editace A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, s. 63–101, 10.1007/0-387-28487-7-3.
URL http://dx.doi.org/10.1007/0-387-28487-7-3

[7] Girau, B.: FPNA: Concepts and Properties. In *FPGA Implementations of Neural Networks*, editace A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, s. 91–92, 10.1007/0-387-28487-7-3.
URL http://dx.doi.org/10.1007/0-387-28487-7-3

[8] Girau, B.: FPNA: Concepts and Properties. In *FPGA Implementations of Neural Networks*, editace A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, s. 81–85, 10.1007/0-387-28487-7-3.
URL http://dx.doi.org/10.1007/0-387-28487-7-3

[9] Gironés, R. G.; Gironés, R. G.; Palero, R. C.; aj.: FPGA Implementation of a Pipelined On-Line Backpropagation. *The Journal of VLSI Signal Processing*, ročník 40, 2005: s. 189–213, ISSN 0922-5773, 10.1007/s11265-005-4961-3.
URL http://dx.doi.org/10.1007/s11265-005-4961-3

[10] Gorgoń, M.; Wrzesiński, M.: Neural Network Implementation in Reprogrammable FPGA Devices ? An Example for MLP. In *Artificial Intelligence and Soft Computing ? ICAISC 2006*, *Lecture Notes in Computer Science*, ročník 4029, editace L. Rutkowski; R. Tadeusiewicz; L. Zadeh; J. Zurada, Springer Berlin / Heidelberg, 2006, ISBN 978-3-540-35748-3, s. 19–28, 10.1007/11785231-3.
URL http://dx.doi.org/10.1007/11785231-3

[11] Hebb, D.: *The Organization of Behavior: A Neuropsychological Theory*. L. Erlbaum Associates, 2002, ISBN 9780805843002.
URL http://books.google.cz/books?id=gUtwMochAI8C

[12] Holt, J.; Baker, T.: Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, ročník ii, jul 1991, s. 121 –126 vol.2, doi:10.1109/IJCNN.1991.155324.

[13] Hopfield, J. J.: Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, ročník 79, č. 8, Duben 1982: s. 2554–2558, ISSN 1091-6490.
URL http://www.pnas.org/content/79/8/2554.abstract

[14] Kohonen, T.: *Self-organization and associative memory*. Springer series in information sciences, Springer-Verlag, 1984, ISBN 9783540121657.
URL http://books.google.cz/books?id=LYZQAAAAMAAJ

[15] Krčma, M.: *Akcelerace neuronových sítí v FPGA*. Fakulta informačních technologií, Vysoké učení technické v Brně, 2011.
URL https://wis.fit.vutbr.cz/FIT/st/rp.php/rp/2011/BP/13719.pdf

[16] Kwan, H.: Simple sigmoid-like activation function suitable for digital hardware implementation. *Electronics Letters*, ročník 28, č. 15, 1992: s. 1379–1380, doi:10.1049/el:19920877.
URL http://link.aip.org/link/?ELL/28/1379/1

[17] McCulloch, W.; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, ročník 5, 1943: s. 115–133, ISSN 0092-8240, 10.1007/BF02478259.
URL http://dx.doi.org/10.1007/BF02478259

[18] Meyer-Bäse, U.; Meyer-Bäse, A.; Mellott, J.; aj.: A Fast Modified CORDIC?Implementation of Radial Basis Neural Networks. *The Journal of VLSI Signal Processing*, ročník 20, 1998: s. 211–218, ISSN 0922-5773, 10.1023/A:1008006030955.
URL http://dx.doi.org/10.1023/A:1008006030955

[19] Minsky, M.; Papert, S.: *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969, ISBN 9780262630221.
URL http://books.google.cz/books?id=4e5wPAAACAAJ

[20] Munakata, T.: Neural Networks: Fundamentals and the Backpropagation Model. In *Fundamentals of the New Artificial Intelligence*, editace T. Munakata, Texts in Computer Science, Springer London, 2007, ISBN 978-1-84628-839-5, s. 7–36, 10.1007/978-1-84628-839-5–2.
URL http://dx.doi.org/10.1007/978-1-84628-839-5--2

[21] Munakata, T.: Neural Networks: Fundamentals and the Backpropagation Model. In *Fundamentals of the New Artificial Intelligence*, editace T. Munakata, Texts in Computer Science, Springer London, 2007, ISBN 978-1-84628-839-5, s. 15–36, 10.1007/978-1-84628-839-5–2.
URL http://dx.doi.org/10.1007/978-1-84628-839-5--2

[22] Munakata, T.: Neural Networks: Other Models. In *Fundamentals of the New Artificial Intelligence*, editace T. Munakata, Texts in Computer Science, Springer London, 2007, ISBN 978-1-84628-839-5, s. 37–84, 10.1007/978-1-84628-839-5-3.
URL http://dx.doi.org/10.1007/978-1-84628-839-5-3

[23] Nelder, J. A.; Mead, R.: A Simplex Method for Function Minimization. *The Computer Journal*, ročník 7, č. 4, 1965: s. 308–313, doi:10.1093/comjnl/7.4.308, http://comjnl.oxfordjournals.org/content/7/4/308.full.pdf+html.
URL http://comjnl.oxfordjournals.org/content/7/4/308.abstract

[24] Ortigosa, E.; Ca?as, A.; Ros, E.; aj.: FPGA Implementation of a Perceptron-like Neural Network for Embedded Applications. In *Artificial Neural Nets Problem Solving Methods*, *Lecture Notes in Computer Science*, ročník 2687, editace J. Mira; J. Álvarez, Springer Berlin / Heidelberg, 2003, ISBN 978-3-540-40211-4, s. 1053–1053, 10.1007/3-540-44869-1-1.
URL http://dx.doi.org/10.1007/3-540-44869-1-1

[25] Prechelt, L. P.; Informatik, F. F.: — A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technická zpráva, Universitat Karlsruhe; 76128 Karlsruhe, Germany, 1994.

[26] Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, ročník 65, č. 6, Listopad 1958: s. 386–408.

[27] Rosselló, J.; Canals, V.; Morro, A.; aj.: Practical Hardware Implementation of Self-configuring Neural Networks. In *Advances in Neural Networks ? ISNN 2009*, *Lecture Notes in Computer Science*, ročník 5553, editace W. Yu; H. He; N. Zhang, Springer Berlin / Heidelberg, 2009, ISBN 978-3-642-01512-0, s. 1154–1159, 10.1007/978-3-642-01513-7-128.
URL http://dx.doi.org/10.1007/978-3-642-01513-7-128

[28] Rumelhart, D.; McClelland, J.; University of California, S. D. P. R. G.: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Foundations.* Computational Models of Cognition and Perception, Mit Press, 1986, ISBN 9780262680530.
URL http://books.google.cz/books?id=eFPqqMBK-p8C

[29] da Silva, R.; Nedjah, N.; de Macedo Mourelle, L.: Reconfigurable MAC-Based Architecture for Parallel Hardware Implementation on FPGAs of Artificial Neural

Networks Using Fractional Fixed Point Representation. In *Artificial Neural Networks ? ICANN 2009*, *Lecture Notes in Computer Science*, ročník 5768, editace C. Alippi; M. Polycarpou; C. Panayiotou; G. Ellinas, Springer Berlin / Heidelberg, 2009, ISBN 978-3-642-04273-7, s. 475–484, 10.1007/978-3-642-04274-4-50.
URL http://dx.doi.org/10.1007/978-3-642-04274-4-50

[30] Steinbuch, K.: Die Lernmatrix. *Biological Cybernetics*, ročník 1, č. 1, 1961: s. 36–45.

Příloha A

Obsah CD

Přiložené DVD obsahuje následující adresářovou strukturu:

/FPNN - soubory s uloženými FPNN

/FPNN VHDL - zdrojové kódy VHDL implementace FPNN

/Latex - zdrojové LaTeX kódy této technické zprávy a dokumentací

/technicka_zprava.pdf - tato technická zpráva

/fig - obrázky z této technické zprávy

/Naive NN - zdrojové kódy přímé VHDL implementace neuronových sítí

/Networks - soubory uložených referenčních sítí, vah a datové sady

/*.train - trénovací datové sady

/*.train_res - výsledky referenční sítě nad trénovací datovou sadou

/*.test - testovací datové sady

/*.test_res - výsledky referenční sítě nad tstovací datovou sadou

/*.net - uložené sítě pro softwarovou FANN implementaci

/*.rates - statistiky potenciálů referenční sítě

/*.results - výpis výstupů neuronů referenční sítě pro všechny vektory trénovací datové sady

/*.weights - váhy referenčních sítí

/Paper - článek na odbornou konferenci

/paper_draft.pdf - koncept článku shrnujícího výsledky práce pro odbornou konferenci

/tex - zdrojové kódy konceptu článku

/PyFPNN - zdrojové kódy projektu PyFPNN

/dokumentace - dokumentace PyFPNN ve formátu HTML

/uzivatelska_prirucka.pdf - uživatelská příručka k aplikacím PyFPNN

/Results - kompletní tabulky s výsledky experimentů a jejich zdrojové kódy

/Software - zdrojové kódy softwarové implementace neuronových sítí

- /uzivatelska_prirucka_software.pdf - uživatelská příručka k softwarové implementaci
- /ANN - zdrojové kódy programu na učení sítí
- /ANNTester - zdrojové kódy programu na testování sítí
- /FANN - zdrojové kódy knihovny FANN

Příloha B

Formáty souborů pro PyFPNN

B.1 Formát popisu FPNN

Aplikace PyFPNN používají textový popis FPNN, který vychází z definice FPNA/FPNN. Každý řádek tohoto popisu začíná návěštím, následovaným dvojtečkou. Návěští určuje, čeho se daný řádek týká. Může se jednat o některé předdefinované návěští, sloužící ke globálním nastavením, nebo se může odkazovat jménem na některý neurální zdroj, pak řádek slouží k nastavní konkrétního neurálního zdroje. Aktivátory mohou mít libovolná jména složená z alfanumerických znaků, spoje ale musí mít jména ve formátu (zdrojový aktivátor,cílový aktivátor).

Předdefinovaná návěští:
Ni - množina jmen vstupů
activators - globální nastavení všech aktivátorů
links - globální nastavení všech spojů
inputs - globální nastavení všech vstupů
number_format - nastavení datového typu

Za návěštím následuje výčet nastavení oddělených středníkem. Jedná se o zápisy *parametr* = *hodnota*, čímž jsou nastavovány konkrétní parametry. Kromě těchto nastavení uvádíme i jména neurálních zdrojů, se kterými je právě nastavovaný neurální zdroj propojen. Nastavení parametrů a jména připojených neurálních zdrojů můžou být na řádku libovolně promíchaná. Jména nastavitelných parametrů odpovídají jménem i významem definici FPNN, jedná se tedy o názvy: `W,T,i,f,theta,a,c`. Parametr `i` podporuje stejné hodnoty jako generický parametr `iter_op` entity `ACTIVATOR`, tedy `+,-,*,/` a parametr `f` podporuje stejné hodnoty jako generický parametr `func_name` entity `ACTIVATOR`, tedy `unipolar_sigmoid`, `bipolar_sigmoid`, `unipolar_step`, `bipolar_step`. V popisu je nutné uvézt množinu `Ni` a nastavení datového typu `number_format`. Lokální nastavení přepisují globální nastavení.

Příklad

Následující příklad definuje trojvrstvé FPNN se třemi aktivátory a dvěma spoji. Je uvedeno několik globálních nastavení, která jsou pak u některých neurálních zdrojů lokálně přepsána - např. spoje mají globálně nastavené parametry W na 1.0, ale u spoje $(n2, n3)$ je tento parametr přepsán na 0.453. Stejně tak u aktivátoru $n1$ je přenastaven parametr *theta*. Nastavení `FX` na řádku `number_format` deklaruje, že se bude používat pevná řádová čárka,

i_part pak určuje počet bitů celé části a *f_part* počet bitů desetinné části. Jména uvedená na řádcích neurálních zdrojů uvádějí propojení, takže např. aktivátor $n3$ je propojen se spojem $(n2, n3)$.

```
Ni: nx
activators: i = +;f = unipolar_sigmoid;theta = 0.0;a = 1
links: W = 1.0
inputs: c = 1
number_format: FX;i_part=8; f_part=8
n1: nx;(n1,n2);theta = 1.0;
n2: (n1,n2);(n2,n3)
n3: (n2,n3)
(n1,n2): n1;n2
(n2,n3): n2;n3;W=0.453
```

B.2 Formát popisu neuronové sítě pro NaiveNNGenerator

Formát textového popisu sítě pro NaiveNNGenerator je založen na řádkovém zadávání hodnot. Každý řádek je ve formátu `nastavení=hodnota`. Na levé straně rovnice je uvedeno jméno nastavení v daném formátu a na pravé číslo nebo řetězec jako hodnota. Tak je možné nastavit parametry neuronům, synapsím i vrstvám. Neurony se číslují od 1 od prvního neuronu ve vstupní vrstvě. Vrstvy se číslují od 1 od vstupní vrstvy.

Jména nastavení:

- structure - udává strukturu sítě jako čísla značící počet neuronů ve vrstvách oddělených znakem "x". Toto nastavení je povinné.
- activation_in_layerX - udává název aktivační funkce neuronů ve vrstvě číslo X. Podporovány jsou stejné hodnoty, které podporuje generický parametr `func\_name` entity `NEURON`, tedy unipolar_sigmoid, bipolar_sigmoid, unipolar_step a bipolar_step. Výchozí hodnota je unipolar_sigmoid.
- activation_param_in_layerX - udává parametr přenosové funkce pro neurony ve vrstvě číslo X. Toto nastavení odpovídá generickému parametru `L` entity `NEURON`. Výchozí hodnota je 5.0
- wX,Y - udává váhu synapse mezi neuronem číslo X a neuronem číslo Y.
- tresholdX - udává hodnotu prahu neuronu číslo X.

Příklad

Zde je uveden případ trojvrstvé sítě se třemi neurony, jejíž výstupní neuron používá jako aktivační funkci skokovou funkci (2.4) a má práh s hodnotou 3.0. Váhy jsou nastaveny na hodnoty svých indexů.

```
structure = 1x1x1
activation_in_layer3 = unipolar_step
activation_param_in_layer3 = 0.0
w1,2 = 1.2
w2,3 = 2.3
treshold3 = 3.0
```

Příloha C

Tabulky

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+DIST_DP	279	104	72.845
PVAL_IP+PROD_DP	253	130	66.057
PROD_DP+WEIG_IP	252	131	65.796
PVAL_IP+PROD_IP	250	133	65.274
PVAL_DP+PROD_DP	249	134	65.013
WEIG_IP+DIST_DP	237	146	61.879
PVAL_IP+DIST_DP	223	160	58.224
PVAL_DP+PROD_IP	206	177	53.785
PVAL_DP+WEIG_IP	159	224	41.514

Tabulka C.1: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_IP+PROD_DP	283	100	73.890
PVAL_IP+PROD_IP	281	102	73.368
PROD_IP+WEIG_DP	280	103	73.107
PROD_DP+WEIG_IP	276	107	72.062
PVAL_DP+DIST_DP	274	109	71.540
PROD_DP+DIST_IP	270	113	70.496
PVAL_IP+DIST_DP	238	145	62.140
PROD_IP+DIST_DP	233	150	60.835
PVAL_DP+PROD_IP	224	159	58.485
WEIG_IP+DIST_DP	213	170	55.613
PVAL_DP+WEIG_IP	140	243	36.553

Tabulka C.2: Porovnání metod nad testovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_IP+DIST_DP	283	100	73.890
PVAL_DP+WEIG_IP+DIST_DP	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP	259	124	67.624

Metoda	Shoda	Neshoda	Shoda [%]
PROD_IP+WEIG_IP+DIST_DP	251	132	65.535
PVAL_IP+PROD_IP+WEIG_IP	250	133	65.274
PVAL_DP+PROD_DP+WEIG_IP	249	134	65.013
PVAL_IP+PROD_IP+DIST_DP	230	153	60.052
PVAL_IP+WEIG_IP+DIST_DP	225	158	58.746
PVAL_DP+PROD_IP+WEIG_IP	204	179	53.263

Tabulka C.3: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_IP+PROD_IP+WEIG_IP	281	102	73.368
PVAL_DP+PROD_DP+WEIG_IP	280	103	73.107
PVAL_IP+PROD_DP+WEIG_DP	278	105	72.584
PROD_DP+WEIG_DP+DIST_IP	277	106	72.323
PVAL_IP+PROD_DP+WEIG_IP	276	107	72.062
PVAL_DP+WEIG_IP+DIST_DP	275	108	71.801
PVAL_DP+PROD_IP+DIST_DP	272	111	71.018
PVAL_IP+PROD_IP+DIST_DP	240	143	62.663
PROD_IP+WEIG_IP+DIST_DP	230	153	60.052
PVAL_IP+WEIG_IP+DIST_DP	229	154	59.791
PVAL_DP+PROD_IP+WEIG_IP	218	165	56.919

Tabulka C.4: Porovnání metod nad testovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	280	103	73.107
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	252	131	65.796
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	251	132	65.535
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	250	133	65.274
PVAL_DP+PROD_DP+WEIG_DP+DIST_IP	249	134	65.013
PVAL_IP+PROD_IP+WEIG_IP+DIST_DP	227	156	59.268

Tabulka C.5: Porovnání metod nad trénovacími daty

Metoda	Shoda	Neshoda	Shoda [%]
PVAL_DP+PROD_DP+WEIG_DP+DIST_DP	283	100	73.890
PVAL_DP+PROD_DP+WEIG_IP+DIST_IP	282	101	73.629
PVAL_IP+PROD_IP+WEIG_IP+DIST_IP	281	102	73.368
PVAL_IP+PROD_DP+WEIG_DP+DIST_DP	280	103	73.107
PVAL_IP+PROD_DP+WEIG_DP+DIST_IP	278	105	72.584
PVAL_DP+PROD_IP+WEIG_IP+DIST_DP	270	113	70.496
PVAL_IP+PROD_IP+WEIG_IP+DIST_DP	240	143	62.663

Tabulka C.6: Porovnání metod nad testovacími daty