
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# RBF NEURONOVÉ SÍTĚ

**RADIAL BASIS FUNCTION NEURAL NETWORK**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **LEOŠ NEVORAL**
**AUTHOR**

**VEDOUCÍ PRÁCE** **doc. Ing. FRANTIŠEK ZBOŘIL, CSc.**
**SUPERVISOR**



**BRNO 2024**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# Zadání bakalářské práce

154551

Ústav: Ústav inteligentních systémů (UITS)
Student: **Nevoral Leoš**
Program: Informační technologie
Název: **RBF neuronové sítě**
Kategorie: Umělá inteligence
Akademický rok: 2023/24

Zadání:

1. Prostudujte zadanou literaturu.
2. Navrhněte demonstrační program pro klasickou RCE neuronovou síť.
3. Navržený program implementujte.
4. Zvažte rozšíření tohoto programu pro eliptickou RBF neuronovou síť.
5. Proveďte potřebné experimenty (například na sadě MNIST).
6. Zhodnoťte dosažené výsledky.

Literatura:


- Moed, M.C., Lee, C. P.: Design of an Elliptical Neural Network with Application to Degraded Character Classification, IEEE Conference on Neural Networks, 1993
- Park, T. H., Cook, P. R.: Radial/Elliptical Basis Function Neural Network for Timbre Classification, ResearchGate, 2005
- Peng, H. W., Lee, S. J., Lee, C. H.: An oblique elliptical basis function network approach for supervised learning applications, Applied Soft Computing, 2017
- Samson, N, Deutsch, C. V.: Elliptical Radial Basis Function Neural Network, in CCG Annual Report 22, 2020


Při obhajobě semestrální části projektu je požadováno:
První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., CSc.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 3.10.2023

# Abstrakt

Práce se zabývá přiblížením problematiky RBF neuronových sítí a specificky RCE sítě pomocí demo aplikace. V demo aplikaci je hlavně vizualizovaný proces učení sítě, ale i stav neuronové sítě a různé tvary bázových funkcí. Dále je také zkoumáno využití EBF neuronových sítí. Klasické přístupy k EBF sítím jsou srovnány s novým návrhem OEBF sítě a následně jsou otestovány. Tento nový návrh je založený na vytváření eliptických oblastí pomocí euklidovské vzdálenosti od ohnisek elipsy. Nový návrh nejeví známky vylepšení vlastností těchto sítí a naopak spíše produkuje skoro stejné výsledky jako klasická RCE síť, které jsou ale řádově asi o jednotky procent méně přesné. Na závěr jsou navrženy možnosti pro budoucí úpravy tohoto přístupu.



# Abstract

The thesis focuses on explaining the field of RBF neural networks, specifically RCE networks, through a demo application. The demo application primarily visualizes the network learning process, as well as the state of the neural network and different shapes of basis functions. Additionally, the utilization of EBF neural networks is explored. Conventional approaches to EBF networks are compared and tested against new design of OEBF network. Which is based on deriving the elliptical areas from euclidean distance from both focal points of ellipse. The new design shows no signs of improving the properties of these networks and rather produces results almost identical to those of the classic RCE network, which are, however, several percentage points less accurate. Finally, methods for improving this solution in future are proposed.



# Klíčová slova

neuronové sítě, učící algoritmy, RBF (radiální bázová funkce), EBF (eliptická bázová funkce), OEBF (obecná eliptická bázová funkce), RCE (restricted coulomb energy), eliptická neuronová síť, demonstrační aplikace

# Keywords

neural networks, learning algorithms, RBF (radial basis function), EBF (elliptic basis function), OEBF (oblique elliptical basis function), RCE (restricted coulomb energy), elliptical neural network, demo application



# Citace

NEVORAL, Leoš. *RBF neuronové sítě*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, CSc.

# RBF neuronové sítě

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Františka Zbořila, CSc. V seznamu použité literatury jsem uvedl všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Leoš Nevoral

8. května 2024

## Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu panu doc. Ing. Františku Zbořilovi, CSc. za všechny konzultace a čas, který mi věnoval. Dále také patří velké poděkování všem lidem, kteří toto odvětví informatiky aktivně rozvíjí a z jejichž prací jsem mohl čerpat a inspirovat se.

# Obsah

# Seznam obrázků

# Kapitola 1

# Úvod

Neuronové sítě představují klíčový prvek moderního strojového učení a umělé inteligence. Tyto sítě, inspirované biologickým fungováním mozku, jsou schopny provádět širokou škálu úkolů, od klasifikace obrázků po předpověď cen akcií. Pro porozumění fungování neuronových sítí je zásadní znalost základních stavebních prvků, jako je samotný neuron. Ale i učících algoritmů sloužících pro vylepšování vlastností neuronových sítí. Proto v této práci bude stručný rozbor neuronů a jejich sestavení do neuronové sítě. Nejprve představíme základní strukturu neuronu a jeho klíčové charakteristiky, jako jsou různé typy bázových funkcí a aktivačních funkcí. Poté se budeme zabývat samotnými neuronovými sítěmi a jejich učením, včetně konkrétního příkladu RCE neuronové sítě.

Pro lepší pochopení prezentovaných konceptů a metodiky učení neuronových sítí bude demonstrována aplikace v praxi. Tato aplikace bude popisovat učení RCE neuronové sítě i s úpravami pro EBF a OEBF. V práci je popsaný postup implementace této aplikace i uživatelské rozhraní a ovládací prvky.

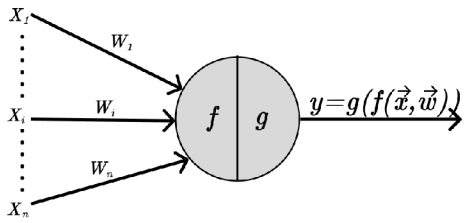
Nakonec se věnujeme testování a experimentům s aplikací neuronových sítí. Tyto experimenty budou zahrnovat testování různých algoritmů pro zmenšování vah neuronů a výběr ohnisek pro eliptickou bázovou funkci. Následně s použitím nejlepších nalezených algoritmů budou porovnány RCE neuronové sítě s neurony s různými bázovými funkcemi. Vybrané testy budou prováděny na známých datasetech jako je MNIST a FASHION MNIST.

Tato práce si klade za cíl poskytnout náhled do základů problematiky neuronových sítí, ale i vytvořit nástroj, který bude možné využít pro demonstraci učení RCE neuronové sítě. Na závěr se také pokusí navrhnout zlepšení nebo obměnu ke stávajícím algoritmům využívaným v oblasti neuronů s eliptickou bázovou funkcí.

# Kapitola 2

# Neuron

Tato kapitola pojednává o umělém neuronu, který je základní stavební jednotkou neuronových sítí. Je inspirovaný biologickými neurony v našem mozku, jejichž základní činností je přeměňovat vstupní signály na výstupní signály. V informatice využíváme umělý neuron, který je běžně popisován tímto matematickým modelem (Obrázek 2.1) [18, 14].




Obrázek 2.1: Model umělého neuronu

Činnost umělých neuronů by se dala shrnout do těchto kroků:

1. Na vstupu neuron přijímá data (ze vstupní vrstvy nebo jiných neuronů).
2. Vstupní data jsou zpracována pomocí bázové funkce.
3. Výsledek bázové funkce je vstupem aktivační funkce.
4. Výsledek aktivační funkce je předán výstupním spojům.

Pro manipulaci výsledků jednotlivých neuronů slouží váhy neuronu. Tyto váhy mohou být např. na vstupních spojích jako modifikátor toho, jak důležitý je daný vstup pro neuron. Dále např. neurony s radiální bázovou funkcí mají více vah, mohou mít váhy na vstupech, ale také lze upravovat jejich středový vektor nebo rozptyl aktivační funkce. Jak a proč se tyto váhy upravují bude později popsáno v kapitole 3.

## 2.1 Rozdělení podle bázové funkce

První z funkcí uvnitř neuronu, na kterou vstupní data narazí, je bázová funkce. Tato funkce zpracovává data na vstupu neuronu a svůj výsledek předává aktivační funkci. Výsledek této funkce budeme označovat symbolem $u$.

### Lineární bázová funkce (LBF)

Lineární bázová funkce, dále jen LBF, je matematicky definována jako

$$u = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

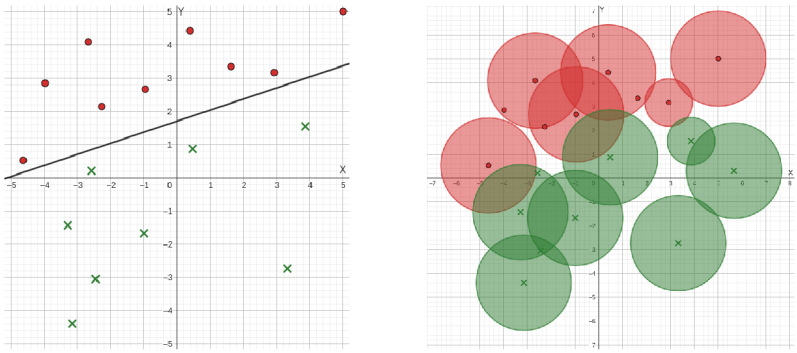
kde $w_i$ je váha spoje mezi i-tým vstupem a neuronem a $x_i$ je i-tý vstupní prvek. Výsledkem $u$ je vážená suma vstupních dat. S použitím této funkce lze data rozdělit do dvou tříd. Lze si to představit jako přímku, která rozděluje prostor do dvou polovin viz obrázek 2.2. Využití neuronů s touto aktivační funkcí je velmi široké a používají se prakticky ve všech oblastech neuronových sítí na celou škálu problémů. V této práci modely využívající tyto neurony nebudeme dopodrobna zkoumat[14, kapitola 18.6].

### Radiální bázová funkce (RBF)

Radiální bázová funkce, dále jen RBF, je funkce označující vzdálenost od středu neuronu nebo také jinak řečeno podobnost se středem neuronu. Hlavní výhodou této funkce je schopnost rozdělit prostor do konkávních útvarů na rozdíl od LBF. Toho můžeme dosáhnout při sjednocení podoblastí více neuronů viz obrázek 2.2. Díky těmto vlastnostem nám stačí jen jedna skrytá vrstva těchto neuronů pro rozdělení komplexních n-rozměrných datových prostorů. To snižuje složitost sítí s RBF neurony i jejich implementaci. Neurony s touto funkcí obsahují centrální vektor $\vec{c}$ a jednu další váhu, která reprezentuje rozptyl sítě, pro skokovou aktivační funkci půjde o průměr $r$ a pro Gaussovu aktivační funkci jde o směrodatnou odchylku $\sigma$. Tyto 2 parametry dohromady definují podoblast v n-rozměrném prostoru, kde jsou vektory klasifikovány do třídy neuronu s touto bázovou funkcí. Tento podprostor má ve 2D prostoru tvar kruhu, ve 3D prostoru tvar koule a v n-rozměrném prostoru se jedná o hyperkouli. Tato funkce je matematicky definována jako

$$u = \sqrt{\sum_{i=0}^{n} (x_i - c_i)^2} \tag{2.2}$$

kde $\vec{c}$ je středový vektor neuronu a $\vec{x}$ je vstupní vektor. Výsledkem $u$ je Euklidovská vzdálenost vstupního a středového vektoru [4, 14, 12, 6, 7].



Obrázek 2.2: Porovnání rozdělení stejného 2D prostoru pomocí LBF a RBF

## 2.2 Aktivační funkce neuronu

Tato funkce zpracovává výsledek bázové funkce neuronu a vytváří výstup neuronu. Také si můžeme tuto funkci představit jako hranici, která určuje, jestli je výstup bázové funkce dostatečně silný pro aktivaci neuronu. A také určuje, jak silná tato aktivace bude. Funkce rozdělujeme na spojité a nespojité (skokové). Spojité funkce mají běžně výstup na nějakém intervalu z $\mathbb{R}$. Kdežto skokové funkce mají jen 2 stavy, buď je neuron plně aktivní, nebo neaktivní. Mezi nejčastěji používané funkce pro LBF patří: [18]

- Skoková funkce: $y = \begin{cases} 1 & \text{pro } u > 0 \\ 0 & \text{pro } u \leq 0 \end{cases}$
- Sigmoidní funkce: $y = \frac{1}{1+e^{-u}}$
- Hyperbolický tangens: $y = tanh(u)$
- SoftPlus: $y = \ln(1 + e^u)$
- SoftMax: $y = \frac{e^{u_j}}{\sum_{i=0}^{n} e^{u_i}}$
- ReLU: $y = \begin{cases} u & \text{pro } u > 0 \\ 0 & \text{pro } u \leq 0 \end{cases}$

U RBF nejčastěji uvidíme: [18]

- Skoková funkce: $y = \begin{cases} 1 & \text{pro } u \leq R \\ 0 & \text{pro } u > R \end{cases}$
- Spojitá funkce (Gaussova funkce): $y = e^{-(\frac{u}{\sigma})^2}$

# Kapitola 3

# Neuronové sítě

Základní myšlenkou této práce je vytvořit nástroj, který bude schopen vizualizovat učení neuronové sítě. Proto je také důležité znát, co se snažíme zobrazit. V této kapitole se budeme věnovat důležitým základním pojmům neuronových sítí. Neuronová síť je definovaná jako matematický model inspirovaný biologickými sítěmi neuronů v našem mozku. Jedná se o nejčastější formu strojového učení, která se snaží generalizovat širokou škálu vstupních dat na odpovídající výstupní data. Formálně je neuronová síť definována jako uspořádaná trojice $(N, V, w)$, kde platí, že $N$ je množina neuronů, $V$ je množina $\{(i,j)|i,j \in \mathbb{N}\}$, jejíž prvky se nazývají spojení mezi neurony i, j. Funkce $w : V \to \mathbb{R}$ definuje váhy, $w((i,j))$ přiřazuje váhu v podobě reálného čísla spojením mezi neurony i, j, zkráceně jako $w_{ij}$ [4, str. 34].

### Architektura neuronových sítí

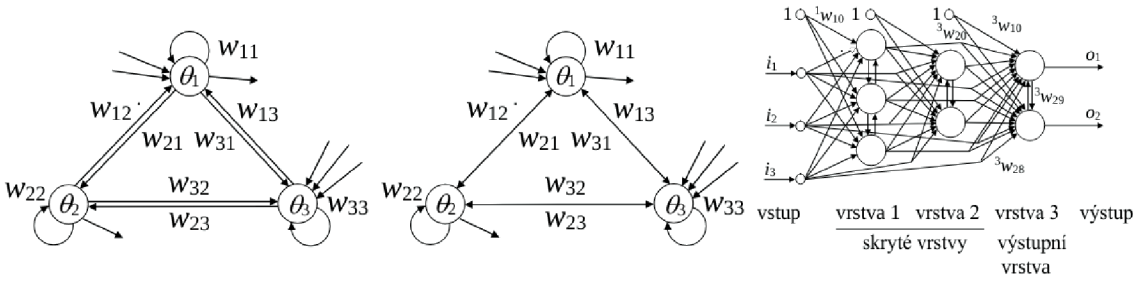
Architektura neuronové sítě je označení pro rozložení neuronů a spojů mezi neurony. Architektura neuronových sítí může být prakticky libovolná. Přehled diagramů nejznámějších neuronových sítí je dostupný v příloze 1. Ale mezi nejčastější architektury řadíme následující [18].

- **Rekurentní sítě viz obrázek 3.1:**

  **Plně propojené sítě:** Neurony jsou propojeny oběma směry, ale váhy každým směrem nabývají vlastních hodnot.

  **Plně propojené symetrické sítě:** Neurony jsou propojeny oběma směry stejnou váhou.

  **Vrstvová síť:** Neurony jsou propojeny s neurony ve stejné a následující vrstvě, ale neovlivňují neurony předchozích vrstev.

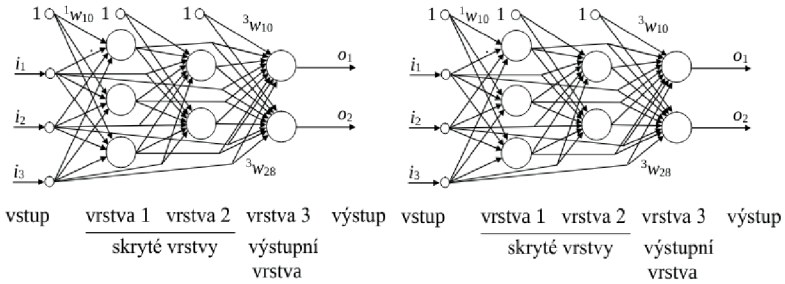



Obrázek 3.1: Rekurentní neuronové sítě převzato z [18]

- **Acyklické sítě viz obrázek 3.2:** (Výpočet odezvy je přímý a výrazně rychlejší, než u sítí rekurentních)

  **Acyklické:** Vrstvová síť, ve které neexistují vazby mezi neurony stejné vrstvy.

  **Dopředné acyklické sítě:** Acyklická síť, ve které existují vazby pouze mezi neurony sousedních vrstev




Obrázek 3.2: Acyklické neuronové sítě převzato z [18]

### Učení neuronové sítě

Pro vývoj a zlepšování vlastností neuronových sítí se používá proces, kterému se říká učení. Jako samotné neurony je i tento proces inspirovaný biologickými procesy v našem mozku. Takové učení je u neuronových sítí velmi podstatné, protože:

- Nemůžeme znát všechny budoucí změny dané vstupní množiny. Pokud se rozhodneme využít neuronovou síť pro predikce trhu, musí se program každý den adaptovat na nový stav trhu.
- Některé problémy ani ti nejschopnější z nás neumí naprogramovat jinak než pomocí učících algoritmů (např. rozpoznávání tváří) [14, str. 693].

Samotné učení může nabírat spoustu různých podob, vytváření/mazání/úpravu neuronů nebo spojů a jejich vah. Jakákoliv kombinace těchto procesů může vést k učení neuronové sítě. V této práci budeme využívat učení, kde se neuronová síť bude učit přidáváním nových neuronů a postupným kalibrováním vnitřních vah neuronu viz kapitola 3.2.

Učení se provádí nad vstupní množinou vektorů trénovacích dat. Každý vektor je předán na vstup neuronové síti a ta na základě svého aktuálního stavu vytvoří výstup. Na základě výstupu sítě a očekávaného výstupu se neuronová síť přizpůsobí. Stejnou vstupní množinu

můžeme použít i víckrát při učení neuronové sítě, ale časem můžeme narazit na situaci, kdy se neuronová síť přeučí a začne hůř generalizovat, protože se naučí trénovací množinu až moc dobře. Pro co nejlepší učení je vhodné mít velmi širokou trénovací sadu. např. pro rozpoznávání obrazu se také využívá metod, kdy se obrázky z trénovací sady upravují (otáčí, deformují atp.), aby se neuronová síť naučila rozpoznávat obrazy i při jiném natočení, a tak zároveň zvýšit velikost trénovací sady.

**Rozdělení neuronových sítí podle učících metod:** Jelikož algoritmů pro učení neuronových sítí existuje celá řada, tak ještě zmíníme nejčastější metody, jak rozdělujeme neuronové sítě podle učících algoritmů.

**Korelační** využívají Hebbova principu, současně aktivujícím neuronům se posilují váhy.

**Soutěživé** spočívá v soutěžení mezi neurony, výstupem je excitace pouze jednoho neuronu, jehož váha, a případně i váha několika topologicky souvisejících neuronů, je následně upravována.

**Adaptační** probíhá ve dvou fázích, nejdříve se přiloží vstupní vektor a porovnáním odezvy sítě a požadovaným výstupem se určí odchylka, následně se vypočítá podíl jednotlivých vah všech neuronů na těchto odchylkách a ten následně použijeme pro úpravu sítě.

Jak můžeme vidět, tak existuje široká škála neuronových sítí. Každá ale vyniká v jiné oblasti od rozpoznávání tváří, zvěře, zvuků, až po předvídání chování akciového trhu nebo výběr vhodných seriálů pro uživatele. Proto je důležité adekvátně volit model neuronové sítě, který se hodí pro řešení našich problémů.

## 3.1 LBF neuronové sítě

Nejznámější a nejčastěji využívané modely obsahují neurony s LBF. Tyto neuronové sítě byly vyvinuty dříve než RBF sítě. Mají různé topologie, ale na rozdíl od RBF se zde využívá i více vrstev neuronů ve skryté vrstvě, protože spoustu problémů nelze lineárně separovat tak, aby šly řešit jednovrstvou sítí jako RBF. Kvůli tomu často narůstá komplexnost topologii těchto sítí i implementační složitost těchto sítí.

Základním principem učících metod těchto sítí je minimalizování chybovosti sítě na dané trénovací množině pomocí úprav vah jednotlivých spojů uvnitř sítě. Nejznámějším a nejčastěji využívaným učícím algoritmem pro acyklické a dopředné sítě je **Backpropagation**.

Jedná se o algoritmus založený na sestupu gradientem. Cílem tohoto algoritmu je minimalizovat funkci často označovanou jako E (error), C (cost) nebo L (loss). Tato funkce je funkcí všech vah sítě a reprezentuje odchylku sítě od požadovaných hodnot. Pro funkčnost tohoto algoritmu je klíčové, aby všechny neurony využívaly diferencovatelnou aktivační funkci. Tento algoritmus spočívá ve vyhodnocení chyby učení porovnáním výstupu sítě a očekávaného výstupu všech trénovacích vektorů. Pro danou chybu je následně spočítán směrový vektor, který určuje, jak je potřeba váhy neuronové sítě upravit, abychom se přiblížili k nejbližšímu lokálnímu minimu této chybové funkce. Tento proces je opakován dokud není dosaženo požadované přesnosti sítě nebo požadovaného počtu epoch[1].

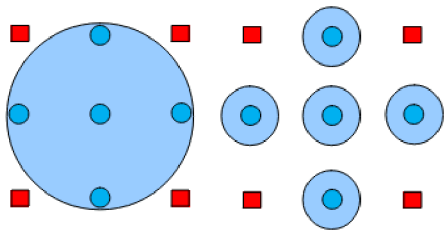
[1] Epocha označuje jeden průchod učení celou trénovací sadou.

## 3.2 RBF neuronové sítě

V této práci primárně využíváme neuronové sítě, které mají ve skryté vrstvě RBF neurony. Základní struktura těchto sítí se často skládá z jedné skryté vrstvy obsahující RBF neurony a výstupní vrstvy, která obsahuje neurony s LBF a často i lineární aktivační funkcí. Vstupní vrstvu většinou zanedbáváme, protože se jedná jen o vrstvu přenášející vstupní data do skryté vrstvy.

Velkou oblíbenost a dobré výsledky tyto neuronové sítě nacházejí např. v oblasti klasifikace (jmenovitě mluveného slova nebo obličejů [17, 15, 5]) nebo třeba funkční aproximátory [16] (především v n-dimenzionálním prostoru).

U tohoto druhu neuronových sítí narážíme na problém, jak vhodně zvolit počet neuronů ve skryté vrstvě. Při zvolení nízkého počtu neuronů může dojít ke špatné přesnosti klasifikace. Naopak při volbě moc velkého počtu neuronů může dojít k přeučení sítě a špatné generalizaci. Na obrázku 3.3 můžeme vidět, jak může správně zvolený počet neuronů ovlivnit výsledek učení.



Obrázek 3.3: Rozdíl mezi vhodně a špatně zvoleným počtem neuronů, převzato z [18]

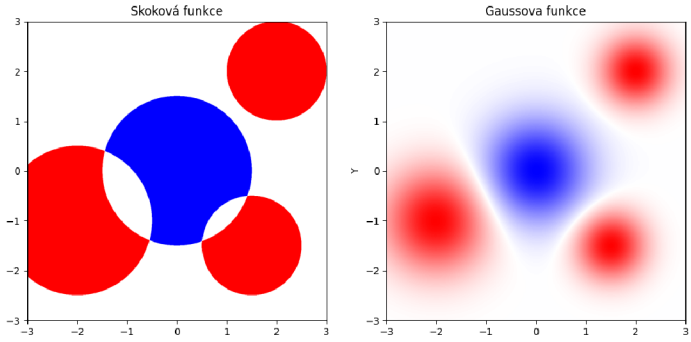
Proto se učení těchto neuronových sítí často rozkládá do dvou částí. V první části se vyberou vhodné středy a rozptyly středových funkcí neuronů a v druhé fázi dochází k učení vah spojů mezi neurony skryté a výstupní vrstvy. První část učení se může provést např. klasickými shlukovacími algoritmy, jako je např. k-means[2] nebo náhodným výběrem z trénovacích dat.

Následně druhá část se provádí např. pomocí soutěživého trénování nebo algoritmem backpropagation. Na druhou stranu tyto sítě přináší i výhody oproti klasickým neuronovým sítím. Velkou výhodou těchto neuronových sítí je jejich snazší topologie, a tak i snazší implementace. Díky vlastnostem neuronů ve skryté vrstvě lze řešit i problémy jako funkci XOR v jedné vrstvě neuronů, což u LBF neuronů nebylo možné.

### RBF sítě se spojitou aktivační funkcí

Jedná se o nejčastěji používaný druh RBF neuronových sítí. Jako aktivační funkci nejčastěji využívají Gaussovu funkci. Tato funkce přiděluje vektorům blíže ke středu silnější aktivaci než těm vzdáleným. Tento přístup má výhody v oblastech, kde je aktivních více neuronů. Můžeme vidět porovnání na obrázku 3.4, kde je vidět, že z výstupu skokové funkce není tak jednoznačné, kterou třídu neuronu vybrat. Na rozdíl od Gaussovy funkce, kde jde vidět postupný přechod z oblasti jednoho neuronu do druhého.

[2] Shlukovací algoritmus vytvářející $k$ shluků. Vstupní vektory přiřazuje do shluků podle nejbližšího těžiště shluku. Následně vypočítá pozici nového těžiště jako průměr bodů v shluku a opakuje proces, dokud dochází ke změnám shluků.




Obrázek 3.4: Rozdíl mezi aktivačními funkcemi

### RBF sítě se skokovou aktivační funkcí

Jak je vidět na obrázku výše, tak neurony uvnitř této sítě rozdělují n-rozměrný prostor na oblast vně a uvnitř neuronu. Takovou sítí je např. i RCE.

#### RCE neuronová síť

Jedná se o dvouvrstvou neuronovou síť. Nultá vrstva je vstupní a pouze předává vektor ze vstupu na výstup, který je napojený na všechny RBF neurony ve skryté vrstvě. Jako aktivační funkci mají neurony tuto skokovou aktivační funkci:

$$y = \begin{cases} 1 & \text{pro } u \leq r \\ 0 & \text{pro } u > r \end{cases}$$

Na výstupu se výsledek předá jednomu neuronu ve výstupní vrstvě. V poslední vrstvě se nachází neurony, které používají ke klasifikaci funkci OR. Tato síť vyniká svou schopností vždy se naučit trénovací množinu vektorů se **100%** úspěšností, ačkoliv za cenu velkého počtu neuronů a delšího času potřebného pro učení sítě. Síť se nehodí pro neseparovatelná data, protože dojde k vytvoření velkého počtu nejmenších možných neuronů.

**Topologie sítě:** Topologie této sítě se v průběhu učení zvětšuje díky rostoucím velikostem skryté a výstupní vrstvy. Skrytá vrstva se mění v závislosti na počtu neuronů, které jsou potřeba pro klasifikaci trénovací množiny. Velikost výstupní vrstvy je závislá na počtu klasifikovaných tříd. Vždy dochází pouze k expandování sítě, protože během učení se nikdy neurony ve skryté ani výstupní vrstvě neodstraňují.

**Algoritmus učení:** Učení této neuronové sítě spočívá v přidávání neuronů se středem v neklasifikovaných vstupních vektorech. Dále pokud je vstupní vektor neuronem klasifikován špatně, tak je podprostor daný tímto neuronem zmenšen.

**Algoritmus 1** Metoda `learn`

```
1: while modified do
2:     modified = false
3:     for  i to trainVectors.length do
4:         hit = false
5:         for neuron in hiddenLayer do
6:             if not neuron.contains(trainVectors[i]) then
7:                 Continue
8:             else if neuron.label != trainLabels[i] then
9:                 modified = true
10:                neuron.shrink(trainingVector)
11:            else
12:                hit = true
13:            end if
14:        end for
15:        if hit = false then
16:            modified = true
17:            addNewNeuron(trainVectors[i], trainLabels[i])
18:        end if
19:    end for
20: end while
```

Na začátku můžeme vidět funkci contains, která se volá nad jednotlivými neurony. Tato funkce není nic jiného než spočítání výsledku bázové funkce a předání tohoto výsledku aktivační funkci, která následně vrací, jestli je vektor obsažen nebo ne.

V průběhu je také volána funkce shrink nad jednotlivými neurony. Základním zmenšovacím algoritmem RBF neuronů v této sítí je zmenšení aktuální velikosti váhy $r$ na část vzdálenosti středu neuronu a neklasifikovaného vektoru. V literatuře se doporučuje zmenšení na 50 %. V této práci později otestujeme i různé jiné poměry zmenšení.

A také na konci můžeme vidět funkci addNewNeuron, která vytváří nové neurony ve skryté vrstvě a to tak, že využije předaný vektor jako střed nového neuronu a váhu neuronu nastaví na nějaké $R_{max}$, které je předem zvolené. Pokud ale zároveň neexistuje neuron ve výstupní vrstvě s třídou, která je funkci předána, tak je vytvořen i nový výstupní neuron.

**Výstup sítě:** Neurony stejné třídy jsou spojeny do jednoho neuronu ve výstupní vrstvě. Následně výstupy všech neuronů v poslední vrstvě jsou propojeny logickou funkcí OR. Tato funkce zajistí, že pokud je výstup alespoň jednoho neuronu aktivní, tak dochází ke klasifikaci vstupního vektoru. Při aktivaci pouze jedním neuronem jsou data klasifikovaná do třídy, která náleží danému neuronu. Pokud je aktivních více neuronů, tak vstupní vektor není klasifikován. Dodatečně lze síť rozšířit o možnost rozhodnutí podle nejbližšího neuronu, pokud je aktivních více výstupních neuronů. V takovém případě je tedy vektor klasifikován do třídy, která odpovídá nejbližšímu neuronu ve skryté vrstvě. Pokud žádný neuron není aktivní, tak vstupní vektor není klasifikován.

**Využití:** Nejběžnější využití této sítě je klasifikace dat, hlavně tedy tzv. single label classification [3].

## 3.3 EBF neuronové sítě

Tyto neuronové sítě využívají bázovou funkci inspirovanou RBF. V porovnání s RBF se stále jedná o rozptyl od středového vektoru, ale tento rozptyl už není symetrický. Využíváme eliptických tvarů, proto název eliptická bázová funkce. Ve 2D se jedná o elipsu, ve 3D o elipsoid a v n-dimenzionálním prostoru o hyperelipsoid. Dokonce z matematického hlediska je elipsa nadřazená kruhu, protože kruh je pouze speciální případ elipsy. Z toho plyne i větší flexibilita této funkce, což je její největší výhodou [12, 7].

### Definice bázové funkce

Na rozdíl od RBF zde existuje více způsobů, jak tuto funkci definovat. Tyto způsoby jsou odvozeny z vlastností elipsy a poslední využívá znalostí z oblasti statistiky.

#### Středová funkce elipsy

Tato funkce je matematicky definována jako

$$\sum_{i=0}^{n} \frac{(x_i - c_i)^2}{w_i^2} = u \tag{3.1}$$

kde $\vec{c}$ je středový vektor elipsy, $\vec{x}$ je vstupní vektor a $\vec{w}$ je vektor délek poloos elipsy. Výsledkem je $u$, což značí normalizovanou vzdálenost bodu od středu elipsy. Délka vektoru $\vec{w}$ je stejná jako počet prvků vstupního vektoru. Při aplikaci tohoto přístupu je potřeba rozlišovat ještě elipsy v normální a obecné poloze. Poloha elipsy je závislá na natočení hlavních os. Pokud jsou osy elipsy rovnoběžné s osami souřadného systému, tak hovoříme o normální poloze a pro výpočet nám stačí výše zmiňovaný vzorec pro středovou funkci elipsy. Obecná poloha elipsy se dá nazvat dalším rozšířením této funkce. V této poloze je elipsa libovolně natočená a např. středová funkce elipsy ve 2D se musí upravit na následující tvar [1].

$$\frac{[(x - x_0)\cos(\alpha) + (y - y_0)\sin(\alpha)]^2}{a^2} + \frac{[(x - x_0)\sin(\alpha) - (y - y_0)\cos(\alpha)]^2}{b^2} = u \tag{3.2}$$

V tomto vzorci zohledňujeme i natočení elipsy. Pro data s vyššími dimenzemi se pak dá využít rotačních matic, které umožní souřadnice vstupních vektorů přepočítat tak, aby stačil původní vzorec pro středovou funkci elipsy. S touto funkcí lze využít následující aktivační funkce.

- Skoková aktivační funkce:

$$y = \begin{cases} 1 & \text{pro } u \leq 1 \\ 0 & \text{pro } u > 1 \end{cases}$$

- Spojitá aktivační funkce (Gaussova funkce):

$$y = e^{-(\frac{u}{\sigma})^2}$$

[3] Výstupem klasifikace je pouze jedna třída. Při klasifikaci do skupiny tříd se jedná se o multi label classification.

### Oblast elipsy definovaná ohnisky

Matematicky je elipsa definována jako množina bodů, které mají konstantní součet vzdáleností od obou ohnisek elipsy. Z toho plyne vzorec

$$\sqrt{\sum_{i=0}^{n}(x_i - F_{1i})^2} + \sqrt{\sum_{i=0}^{n}(x_i - F_{2i})^2} = u \tag{3.3}$$

kde $\vec{F_1}$ a $\vec{F_2}$ jsou souřadnice ohnisek elipsy a $\vec{x}$ je vstupní vektor. Následně $n$ je velikost vstupního vektoru a ohnisek. Výsledkem je $u$ , což značí vzdálenost od obou ohnisek elipsy.

Tento přístup je blíže podobný RBF než ostatní zmíněné postupy, protože určujeme výsledek pouze z Euklidovské vzdálenosti bodů stejně jako u RBF. Ale na rozdíl od RBF potřebujeme dva body místo jednoho. Pro účely prozkoumání této bázové funkce byly navrženy algoritmy pro výběr ohnisek popsané později v této kapitole viz 3.3.

Manipulace s velikostí dané elipsy je také o to složitější, nestačí pouze upravit délku poloos. Po úpravě délek poloos je potřeba také v závislosti na nové excentricitě elipsy vypočítat nové souřadnice obou ohnisek. Excentricitu spočítáme pomocí vzorečku

$$e = \sqrt{a^2 - b^2} \tag{3.4}$$

kde $a$ a $b$ jsou délky hlavní a vedlejší poloosy neuronu. Následně pro přepočítání pozice ohnisek v n-dimenzionálním prostoru využijeme vzorec

$$F_1 = \vec{c} + \vec{v}\frac{e_{nove}}{e_{stare}}, F_2 = \vec{c} - \vec{v}\frac{e_{nove}}{e_{stare}} \tag{3.5}$$

kde $\vec{c}$ je středový vektor elipsy, $\vec{v}$ je směrový vektor od středu k původním ohniskům a $e_{nove}$ a $e_{stare}$ jsou původní a nová excentricita elipsy.

Zároveň funkce přináší jedno omezení, kdy nemůžeme po vytvoření zaměnit hlavní a vedlejší osu. Hlavní osa vždy musí zůstat delší, jinak by bylo potřeba přesunout ohniska na vedlejší osu, ale pro takovou operaci mi nejsou známy vzorečky, které by byly aplikovatelné ve více dimenzionálních prostorech. Tento problém je kompenzován případným zmenšením i vedlejší osy na délku hlavní osy, když by se hlavní osa zmenšila na kratší délku než je délka vedlejší osy.

Na druhou stranu má tento přístup výhodu v odlehčení potřeby počítat natočení elipsy, které je automaticky obsaženo v polohách obou ohnisek. Díky tomu můžeme říci, že funkce se automaticky řadí do kategorie OEBF, neboli eliptické bázové funkce v obecné poloze.

S touto funkcí lze využít následující aktivační funkce.

- Skoková aktivační funkce:

$$y = \begin{cases} 1 & \text{pro } u \leq 2a \\ 0 & \text{pro } u > 2a \end{cases}$$

- Spojitá aktivační funkce (Gaussova funkce):

$$y = e^{-(\frac{u-2E}{\sigma})^2}$$

Na rozdíl od aplikace klasické Gaussovy funkce, je zde potřeba odečíst od výsledku aktivační funkce $2E$, což značí délku excentricity elipsy. Jinak by funkce většinou nebyla ani ve středu aktivována naplno, protože vzdálenost od obou ohnisek může být nulová, jen v případě, že elipsa je zároveň i kružnice.

Na závěr je pro tento druh bázové funkce navíc potřeba vytvořit algoritmus pro výběr obou ohnisek elipsy. Ostatní přístupy se více podobají RBF v potřebě jen jednoho bodu jako

středu neuronu, ale zde potřebujeme i ohniska nebo střed a směrový vektor od středu k ohniskům. Jelikož tento přístup nebyl v žádné mně známé práci zkoumán, tak jsem algoritmy vytvořil sám. Algoritmy jsou založené na relativně jednoduché selekci z trénovacích vektorů, inspirované prací s RCE sítí.

- První algoritmus využívá první neklasifikovaný bod z trénovací množiny jako ohnisko a vybírá k němu z trénovacích vektorů druhé ohnisko podle těchto pravidel. Vybraný vektor nesmí být také ještě klasifikován neuronovou sítí. Střed mezi oběma vektory (střed elipsy) také nesmí být klasifikován žádným neuronem. Tato pravidla slouží pro omezení případů, kdy by se následně velikosti poloos zmenšily natolik, že by první vybraný vektor už nebyl klasifikován vytvořeným neuronem. Následně z množiny vhodných vektorů vybereme ten nejvzdálenější. Pokud žádný vektor nesplňuje parametry, tak obě ohniska jsou reprezentovaná aktuálním vektorem. Čímž je prakticky vytvořen klasický RBF neuron.

**Algoritmus 2** fociFitness v1

```
    Vstup: trainingVector
    Vstup: trainingLabel
    Výstup: center
    Výstup: focalPoints
 1: maxDistance = 0
 2: for Každý trenovací vektor, u kterého platí vector.label = trainingLabel do
 3:     newCenter = (trainingVector + vector) / 2
 4:     if Žádný neuron neobsahuje newCenter a vector then
 5:         vectorDistance = squaredDistance(trainingVector, vector)
 6:         if vectorDistance > maxDistance then
 7:             maxDistance = vectorDistance
 8:             center = newCenter
 9:             focalPoints = {trainingVector, vector}
10:         end if
11:     end if
12: end for
```

- Druhý algoritmus uvažuje první neklasifikovaný vektor jako střed elipsy, po vzoru RBF. Následně si směrový vektor pro ohniska určíme jako vektor mezi tímto vektorem a nejvzdálenějším vektorem z trénovací sady, který spadá do stejné třídy. U tohoto algoritmu není vybíráno podle žádných speciálních pravidel jako předtím, protože zvolením onoho vektoru jako středu je zaručena klasifikace tohoto vektoru vytvořeným neuronem i po libovolné redukci velikostí poloos tohoto neuronu.

**Algoritmus 3** fociFitness v2

**Vstup:** trainingVector
**Vstup:** trainingLabel
**Výstup:** center
**Výstup:** focalPoints

```
maxDistance = 0
newCenter = trainingVector
for vector in trainingVectors where vector.label = trainingLabel do
    vectorDistance = distance(trainingVector, vector)
    if vectorDistance > maxDistance then
        maxDistance = vectorDistance
        focisVector = newCenter - vector
    end if
end for
focalPoints = {newCenter + focisVector, newCenter - focisVector}
```

### Mahalanobisova vzdálenost od středu

Jedná se o vzdálenost, kterou zavedl ve statistice pan Prasanta Chandra Mahalanobis. Využívá se k vyhodnocení podobnosti dvou objektů ve více kategoriích naráz. V našem případě tedy nalezení podobnosti se středy neuronů. Vzorec pro výpočet této vzdálenosti je následující

$$u = \sqrt{(\vec{x} - \vec{c})^T C^{-1} (\vec{x} - \vec{c})} \tag{3.6}$$

kde $\vec{x}$ je vstupní vektor, $\vec{c}$ je střed neuronu a $C^{-1}$ je kovariantní matice. Právě touto kovariantní maticí je brán v potaz tvar a orientace distribuce reprezentované neuronem. Využití této funkce lze najít např v pracích [12, 13]. Jako aktivační funkce může být znovu použita Gaussova i skoková funkce.

## Optimalizace velikosti elipsy

Během učení těchto neuronových sítí využíváme podobných principů jako pro RBF neuronové sítě. Jedním z problémů proto je, jak zvolit ideální velikosti elips. Tohle, stejně jako u RBF, většinou řešíme nějakými klasickými metodami, jako jsou např. shlukovací algoritmy. Pro tuto práci je ale zásadní algoritmus pro zmenšování elipsy, protože během učení v RCE síti dochází ke zmenšování rozptylu od středu v neuronech. V rámci efektivního učení, je proto potřeba elipsu zmenšovat tak, aby neobsahovala body, které patří do jiné třídy, ale zároveň si udržela co největší velikost, aby nebylo potřeba vytvářet další neurony. V práci je využito poznatku z práce [7], kde je dokázáno, že pro získání optimálního objemu stačí upravit velikost jedné poloosy. V práci je následně otestováno a porovnáno víc variant algoritmů pro zmenšování poloos. Jedná se o tyto algoritmy.

**Zmenšení podle nejvzdálenější osy:** Najdeme osu, ve které se vstupní vektor nachází nejblíž k okraji, následně tuto osu zkrátíme na část vzdálenosti středu neuronu a vstupního vektoru v této ose. Ostatní délky poloos ponecháme nezměněny. Tento algoritmus je inspirovaný algoritmem pro zmenšování RBF neuronů a je aplikován na EBF neurony se středovou funkcí elipsy.

**Algoritmus převzatý z [7]:** Pro nalezení indexu poloosy, kterou je ideální zmenšit, spočítáme nové délky poloos tak, aby bod ležel za hranicí elipsy a zvolíme variantu, kde je produkt délek os největší. Tomuto postupu odpovídá tento vzorec

$$\delta = argmax(i) \left[ \prod_{j=0, j\neq i}^{k-1} w_j + \frac{c_i - x_i}{\sqrt{1 - \sum_{j=0, j\neq i}^{k-1} (c_j - x_j)^2/(w_j)^2}} \right] \tag{3.7}$$

Následně velikost této poloosy spočítám pomocí vzorce

$$w_\delta = \frac{c_\delta - x_\delta}{\sqrt{1 - \sum_{j=0, j\neq \delta}^{k-1} (c_j - c_j)^2/(w_j)^2}} \tag{3.8}$$

V obou vzorcích platí, že $\vec{w}$ je vektor velikostí poloos, $\vec{x}$ je vstupní vektor a $\vec{c}$ je středový vektor elipsy. Konstanta $k$ je velikost zmíněných vektorů. Výsledek $\delta$ je index poloosy, kterou je optimální zmenšit. Délku této poloosy nahradíme novou délkou a ostatní délky poloos ponecháme nezměněny. Tento postup je aplikovatelný na EBF neurony se středovou funkcí.

**Zmenšení elipsy definované ohnisky:** Tuto optimalizační metodu jsem navrhl sám. Při vytváření jsem využil modelu elipsy v programu Geogebra[4] a porovnával jsem svůj algoritmus s výsledky, které produkoval výše zmíněný algoritmus pro optimalizaci středové funkce elipsy. Výsledky byly srovnatelné, proto jsem se rozhodl tento algoritmus následně i otestovat. Tento algoritmus je založený na binárním vyhledávání v intervalu $< puvodni\_delka\_poloosy, 0 >$

1. Vybereme si jednu poloosu, kterou budeme zkracovat.
2. Vytvoříme si nové poloosy a vybranou poloosu, zkrátíme na polovinu.
3. Dopočítáme si ostatní osy, novou excentricitu a vypočítáme si nová ohniska, pomocí vzorečků z 3.3
4. Otestujeme jestli bod stále patří do elipsy.
5. Pokud ano, tak délku poloosy znovu zkrátíme na polovinu a vrátíme se o krok zpět.
6. Pokud ne, tak zkontrolujeme kolikrát je větší vzdálenost od ohnisek než $2a$. Jestliže je vzdálenost větší o méně než 1 % z $2a$, tak jdeme na další krok, jinak zvětšíme délku osy o polovinu délky minulého posunu a vrátíme s do kroku 3.
7. Pokud jsme zkusili zkrátit pouze jednu poloosu, tak si zvolíme druhou poloosu, vrátím se do kroku 2 a postup zopakujeme.
8. Zvolíme řešení, které má největší produkt délek poloos.
9. Nahradíme staré délky os a ohniska novými.

### Navržená síť

Už máme definované všechny potřebné podpůrné části pro aplikaci eliptických bázových funkcí. Proto si na závěr představíme, jak tyto části zkombinujeme do celku. Po vzoru RBF sítí budeme využívat neuronové sítě s jednou skrytou vrstvou, přesněji síť jako RCE.

[4] Geogebra je matematický software sloužící pro výuku matematiky. Více informací na https://www.geogebra.org/about

Učící algoritmus RCE sítě viz 3.2 zachováme ve stejné podobě.

Při tvorbě nových neuronů, např. pro OEBF založenou na ohniskách, ještě musíme projít trénovací vektory jednou z funkcí fociFitness viz 3.3. Poté si určíme délku hlavní osy jako vzdálenost ohnisek a k tomu přičteme parametr $R_{max}$ a délku vedlejší osy si určíme jako $R_{max}$.

Pro neurony se středovou eliptickou bázovou funkcí, zachováme stejně výběr aktuálního neklasifikovaného vektoru jako středu elipsy a následně délky všech poloos nastavíme na $R_{max}$.

Tento přístup využití co nejvíc identické neuronové sítě je pro nás výhodný hlavně pro férové porovnávání vlastností neuronů a ne učícího algoritmu.

# Kapitola 4

# Demo aplikace

V této kapitole se budu věnovat tvorbě demo aplikace, která slouží pro zobrazení učení RCE neuronové sítě. Základním požadavkem je pochopitelná vizualizace učení neuronové sítě a role, kterou neurony plní. Pro splnění těchto požadavků je součástí aplikace okno, do kterého pomocí kreslícího nástroje uživatel zanese trénovací data. Tato data se dělí do učících tříd pomocí barev. Nevybarvené pozadí zůstane jako prázdný datový prostor. Po zadaní všech trénovacích dat je možné spustit učení sítě. Během učení sítě aplikace zvýrazňuje bod, který se aktuálně učí rozpoznávat a i neuron, se kterým ho porovnává. Jednotlivé neurony jsou v aplikaci zobrazené daným 2D útvarem, který je reprezentuje (RBF - kruh, EBF - elipsa, OEBF - elipsa s obecnou polohou). Učení si uživatel ovládá sám pomocí tlačítek a může si při učení zvolit od posunu po jednotlivých krocích až k úplnému naučení sítě naráz. Dále může uživatel měnit i maximální délku poloos neuronů

## 4.1 Zvolené technologie

Aplikace bude implementovaná pomocí webových technologií HTML, CSS a TypeScript. Tyto technologie byly zvoleny díky jejich přenositelnosti a snadnému použití pro uživatele. Zároveň jsou tyto technologie velmi jednoduché. Pro vývoj není potřeba implementace serverové strany, takže jsou dostačující. Po stažení aplikace už nedochází k žádné komunikaci po síti a aplikace běží pouze v prohlížeči u uživatele.

**HTML** [10]- Je základním stavebním kamenem většiny webových stránek. Je to značkovací jazyk pro tvorbu webových stránek. Tvoří uživatelské rozhraní, do kterého jsou následně vloženy výsledky z aplikace.

**CSS** [9]- Jazyk sloužící pro popis vzhledu webových stránek, tvoří design a formát uživatelského rozhraní.

**Typescript** [11]- Jedná se o nadstavbu jazyka JavaScript, která přidává zejména typovou kontrolu, ale i různé další funkcionality s cílem zvýšit čitelnost kódu a údržbu aplikací. Tvoří logiku aplikace a reaguje na komunikaci s uživatelem.

## 4.2 Popis architektury

Jak již bylo zmíněno, vstupní data budou získaná z kreslícího plátna. Toto plátno je implementováno pomocí HTML elementu canvas [1]. Zároveň je canvas využit i pro vizualizaci neuronů, protože jeho API nabízí metodu ellipse, která slouží k vykreslování elips všech tvarů a barev. Na závěr je důležité zmínit, že data i vizualizace mají oddělené canvasy, které jsou překryty přes sebe tak, aby se při implementaci oblast vybarvená neurony nemohla zaměnit se vstupními daty. V praxi to následně znamená, že ovládání probíhá jen pomocí vrchního canvasu, kde se nachází vizualizace neuronů a nakreslená oblast je přenášena do spodního canvasu. Ovládání těchto pláten je uzavřeno uvnitř třídy jménem Vizualizer. Následně aplikace obsahuje třídu RCENN, která slouží jako rozhraní pro přístup k neuronové síti. Pomocí této třídy a jejích metod lze učení sítě ovládat. Všechny tyto části následně dohromady spojuje hlavní ovládací smyčka aplikace.

## 4.3 Implementace

Všechny části kódu jsou implementovány v jazyce Typescript a následně zkompilovány do jazyka Javascript. Pro kompilaci byl ještě vytvořen skript napsaný v jazyce Python, který všechny soubory v jazyce Typescript spojí do jednoho souboru a náležitě odstraní odkazy do jiných souborů, kvůli problémům s kompatibilitou modulových systémů jazyka Javascript v prohlížeči.

### Vizualizer

Celé ovládání kreslícího plátna a zobrazování neuronů je zaobaleno ve třídě s názvem Vizualizer. Tato třída obsahuje následující atributy.

**board, front** - Canvas elementy, které slouží pro vykreslování vstupních dat a neuronů.

**boardCtc, frontCtx** - Kontexty canvasů, které slouží pro manipulaci obsahu. Z optimalizačních důvodů je datový prostor boardCtx 3x menší a zároveň je každý pixel v tomto canvasu 3x větší. Na vzhledu to není tolik vidět a zároveň je omezen maximální datový prostor, který síť musí pojmout. V aktuální konfiguraci (šířka 333px x výška 167px) musí síť maximálně zpracovat oblast 55 611 vstupních vektorů. Už při tomto počtu vstupních vektorů a několika různých třídách se síť začne učit relativně pomalu a výsledek může trvat až několik minut (u mě čas nepřesáhl 3 minuty). Prostor by šel ještě zmenšit a prakticky tak anulovat delší čas učení, ale degradace vzhledu kreslícího plátna je až moc viditelná. Navíc není od věci, když uživatel vidí, jak s rostoucím počtem dat roste i komplexita a čas potřebný na učení.

**isDrawing** - Proměnná sloužící pro kontrolu, zda uživatel kreslí. Aktivuje se při stisknutí tlačítka myši v oblasti plátna. Deaktivuje se při uvolnění tlačítka myši nebo opuštěni oblasti plátna. Díky tomu nedochází k vykreslování pokud uživatel jen přejíždí po plátně.

**usedColors** - Pole obsahující barvy, které byly použity pro vykreslení vstupních dat. Každá barva je reprezentována jako pole tří hodnot RGB. Toto pole slouží pro následnou normalizaci vstupních dat.

[1] Jedná se o grafický element, který umožňuje vykreslování obrázků i animací. Obsahuje rozsáhlé API do javascriptu, kterým se dá ovládat [8].

Dále tato třída obsahuje následující metody.

**startDrawing** - Metoda pro inicializaci kreslení. Aktivuje se stisknutím tlačítka myši v oblasti plátna. Dojde k přepnutí proměnné isDrawing na true. A zavolá se metoda draw.

**draw** - Metoda pro vykreslení na aktuální pozici kurzoru. Aktivuje se pohybem po plátně.

**Algoritmus 4** draw

```
    Vstup: mouseMovement
 1: if isDrawing = false then
 2:     return
 3: end if
 4: rect = front.getBoundingClientRect();
 5: x, y = round((e.clientX - rect.left) / 3), round((e.clientY - rect.top) / 3)
 6: boardCtx.strokeStyle = (document.getElementById("colorPicker") as HTMLInputEle-
    ment).value
 7: boardCtx.lineWidth = (document.getElementById("size") as HTMLInputEle-
    ment).value
 8: boardCtx.lineTo(x, y);
 9: boardCtx.stroke();
10: boardCtx.beginPath();
11: boardCtx.moveTo(x, y);
```

**stopDrawing** - Metoda pro ukončení kreslení. Aktivuje se uvolněním tlačítka myši nebo opuštěním oblasti plátna. Vytvoříme nový kreslící kurzor uvnitř canvasu tak, aby při dalším volání funkce draw mohla tento kurzor rovnou přesunout na aktuální pozici myši a malovat. Kdybychom to neudělali, tak malování naváže na poslední bod, kde jsme přestali.

**drawEllipse** - Metoda pro vykreslení elipsy na plátně. Aktivuje se automaticky v průběhu učení. Během kreslení se samostatně vykreslí černý okraj elipsy a následně je elipsa i vyplněna barvou.

**clearBoard, clearFront** - Metody pro smazání obsahu plátna. Aktivují se při změnách ve skryté vrstvě nebo při aktivaci tlačítkem Clear.

**tresholdColors** - Metoda pro normalizaci barev na plátně. Jednou z vlastností kreslení na canvas je, že v oblastech, kde se barvy potkávají, dochází k jejich mixování. Při vykreslování se objevují ale i další nehodící se pixely bez známé příčiny. Pro naši síť je důležité mít data rozdělena do přesných tříd, protože jinak každý barevně odchýlený pixel bude zařazen do své vlastní třídy a všechny ostatní neurony se okolo něj budou muset stáhnout. Toho dosáhneme tak, že po každém kreslení na plátno znormalizujeme každý pixel na plátně, který neodpovídá žádné použité barvě, na poslední použitou barvu.

**Algoritmus 5** tresholdColors

```
1: for Každý pixel v plátně do
2:     if Barva pixelu není v usedColors a pixel nemá průhlednou barvu then
3:         boardCtx.fillStyle = color;
4:         boardCtx.fillRect(x, y, 1, 1);
5:     else if Barva pixelu je v usedColors, ale pixel má průhlednou barvu then
6:         boardCtx.fillStyle = color;
7:         boardCtx.fillRect(x, y, 1, 1);
8:     else
9:         continue
10:    end if
11: end for
```

### RCENN

Zde je přehled klíčových atributů, které třída obsahuje (seznam není implementačně kompletní).

**hiddenLayer** - Pole obsahující neurony ze skryté vrstvy. Každý neuron zároveň obsahuje třídu, ke které je přidělen a tímto způsobem je nahrazena poslední vrstva neuronů.

**neuronType** - Typ neuronu, který je použit ve skryté vrstvě.

**trainVectors** - Pole obsahující vstupní vektory trénovací sady.

**trainLabels** - Pole obsahující popisky tříd trénovací sady.

**axisesRange** - Hodnota značící maximální délky poloos neuronů.

**trainingCounter** - Počítadlo, které určuje, na kolika trénovacích vektorech už bylo provedeno učení sítě.

**checkedNeuronCounter** - Počítadlo, které určuje, kolik neuronů už bylo zkontrolováno pří učení aktuálního trénovacího vektoru.

**modified** - Proměnná značící, jestli v průběhu epochy došlo ke změně sítě.

**hit** - Proměnná značící, jestli byl trénovací vektor klasifikován správně.

**learningState** - Proměnná značící, co se stalo v posledním učícím intervalu sítě.

**epoch** - Počítadlo reprezentující, kolik epoch bylo provedeno.

### Ovládání neuronové sítě

Pro hlavní ovládání a manipulaci se sítí má třída následující metody (seznam není implementačně kompletní).

**changeMaxRange** - Metoda měnící změnu maximální délky poloos neuronů. Po změně se síť restartuje do původního stavu a musí se znovu učit.

**changeNeuronType** - Metoda měnící typ neuronu ve skryté vrstvě. Aby nedocházelo k mixování různých typů neuronů uvnitř skryté vrstvy, tak se neuronová síť restartuje.

**loadTrainData** - Slouží k invokaci načtení trénovacích dat z kreslícího plátna v aplikaci do sítě. To provádí instance třídy inputData pomocí metody readData. Tato metoda vrací celý obsah kreslícího plátna v podobě 2 polí. Jedno obsahuje X,Y souřadnice na plátně a druhé jednotlivé třídy těchto bodů. Tyto data jsou rozdělena na data vhodná a nevhodná pro učení. Data nevhodná pro učení mají v třídě na pozici alpha kanálu dané barvy hodnotu jinou než 255. Neboli jedná se o částečně nebo úplně průhledný bod.

**learn** - Metoda spouštějící učení sítě. Při volání metody lze omezit učení jen na daný počet vstupních vektorů, nebo neuronů, které se mají projít. Bližší popis a vysvětlení funkcionality této metody poskytuje pseudokód níže. Uvnitř metody je také volána metoda **checkHiddenLayer**, která zajišťuje kontrolu skryté vrstvy. Díky tomu, že je tento algoritmus jen verze učícího algoritmu RCE sítě upravená o ovládací prvky. Tak nebudou znovu popsány např. funkce jako addNewNeuron. Jejich vysvětlení lze najít v kapitole 3.2.

### Pseudokód učícího algoritmu

**Algoritmus 6** Metoda `learn`

```
    Vstup: vectorLimit
    Vstup: neuronLimit
 1: if Učení sítě už bylo dokončeno then Return
 2: end if
 3: for  i to vectorLimit do
 4:     checkHiddenLayer(trainingVectors[i], trainingLabels[i], neuronLimit)
 5:     if Učením neprošla celá skrytá vrstva then Return
 6:     else if hit = false then
 7:         modified = true
 8:         addNewNeuron(trainVectors[i], trainLabels[i])
 9:     end if
10:     hit = false
11: end for
12: if Učení ještě neprošlo všechny vstupní vektory then
13:     Return
14: else if modified = true then
15:     learningState = "epochFinished"
16:     modified = false
17:     epoch  1
18: else if modified = false then
19:     learningState = "finished"
20:     epoch  1
21: end if
```

**Algoritmus 7** Metoda checkHiddenLayer

**Vstup:** trainingVector
**Vstup:** trainingLabel
**Vstup:** neuronLimit

```
1:  for k to neuronLimit do
2:      neuron = hiddenLayer[k]
3:      if not neuron.contains(trainingVector) then
4:          learningState = "notContained"
5:          Continue
6:      else if neuron.label ≠ trainingLabel then
7:          learningState = "collision"
8:          modified = true
9:          neuron.shrink(trainingVector)
10:     else
11:         learningState = "found"
12:         hit = true
13:     end if
14: end for
```

### Neurony

Všechny 3 (RBF, EBF, OEBF) druhy neuronů využité v aplikaci mají svou vlastní třídu, která je reprezentuje. Základní rozhraní neuronů je navrženo ve třídě OEBF, ze které následně ostatní třídy dědí. Toto rozhraní obsahuje následující atributy.

**center** - Vektor obsahující střed neuronu.

**semiAxes** - Vektor obsahující délky jednotlivých poloos neuronu.

**rotationAngle** - Hodnota značící natočení elipsy neuronu.

**label** - Třída neuronu nahrazující poslední vrstvu sítě.

Dále také obsahuje tyto metody pro práci s neurony.

**contains** - Metoda, která aplikuje bázovou a aktivační funkci neuronu a vrací jejich výsledek. V tomto případě na reprezentaci EBF a OEBF neuronu používáme středovou funkci elipsy v normální a obecné poloze viz 3.3. A jako aktivační funkci využíváme skokovou aktivační funkci, také viz 3.3. Pro RBF neurony využívám také skokovou aktivační funkci, jak je popsáno v 3.2.

**shrinkNeuron** - Pomocí této metody dojde ke zmenšení neuronu tak, aby už neuron neobsahoval vektor, který je funkci předán. V případě EBF a OEBF využíváme algoritmus převzatý z [7] a pro RBF využíváme klasický algoritmus pro zmenšování neuronů v RCE sítí viz 3.2.

Díky této implementaci můžeme využít toho, že je EBF neuron pouze speciálním případem OEBF a to takovým, kdy je natočení elipsy 0. Proto nemusíme EBF nijak dále implementovat a pouze v konstruktoru této třídy vytvoříme OEBF neuron s nulovým natočením. Pro RBF neuron sice můžeme dědit z OEBF atributy, ale tentokrát je potřeba změnit metodu contains i shrinkNeuron tak, aby odpovídala specifikacím výše.

### Uživatelské ovládání

Jelikož je uživatelské rozhraní relativně dlouhé a implementačně nijak zajímavé, protože jen propojuje všechny vytvořené části, tak nebudeme zacházet do implementačních detailů. Jako např. které metody jsou kde provolávány a jenom si popíšeme základní části tohoto ovládání a co vše která řeší.

**Ovládání kreslícího plátna:** Vytvoří se nová instance třídy Vizualizer a pro akce myši jako pohyb nebo stisknutí tlačítka nad plátnem se nastaví reakce od vizualizéru.

**Ovládání parametrů sítě:** Ke každému tlačítku pro výběr typu neuronu se nastaví adekvátní reakce změna sítě i uživatelského rozhraní. Navíc při změně hodnoty v poli Rmax se nová délka také propíše do neuronové sítě.

**Ovládání učení:** Na každé tlačítko je nastaveno volání metody learn s jinými parametry tak, aby bylo dosaženo požadovaného učení.

**Popisky učení a statistiky:** Při každé změně stavu neuronové sítě se aktualizují hlášky, které se mají uživateli zobrazit. Zároveň se aktualizují i statistiky o stavu sítě.

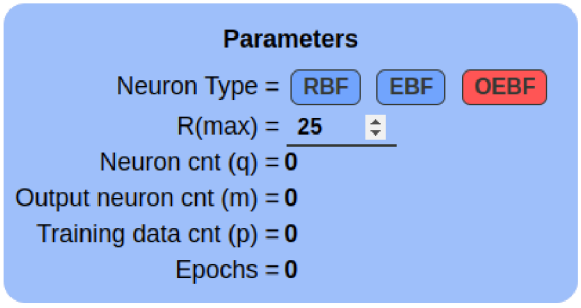
## 4.4 Popis grafického uživatelského rozhraní

Uživatelské rozhraní je rozděleno do pěti hlavních částí.

**Kreslící plátno** - Uživatel zde vkládá trénovací data. V horním levém rohu si uživatel zvolí barvu, která označuje třídu dat a velikost štětce, kterým kreslí. Dále může data smazat pomocí tlačítka "Clear", které smaže vše, viz obrázek 4.1.


Obrázek 4.1: Kreslící plátno

**Parametry sítě** - V této části si uživatel může nastavit maximální délku poloos neuronů a typ neuronu ve skryté vrstvě. Dále zde vidí, kolik skrytá vrstva obsahuje neuronů, kolik je neuronů ve výstupní vrstvě, kolik je nakreslených trénovacích dat a počet epoch, které již proběhly, viz obrázek 4.2. Toto okno se nachází buď napravo od kreslícího plátna nebo pod ním v závislosti na velikosti okna prohlížeče.



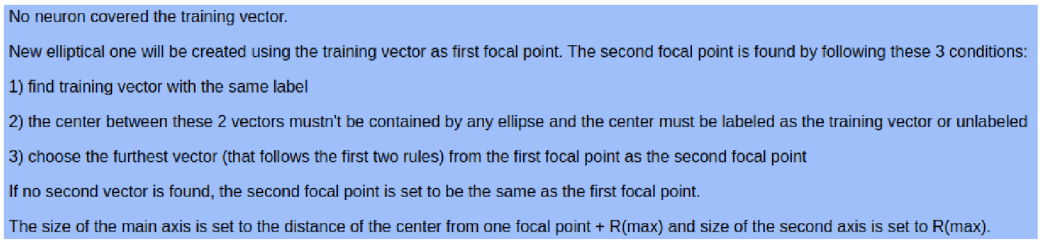

Obrázek 4.2: Nastavení a parametry sítě

**Ovládání učení** - Jedná se o set tlačítek, která na pozadí volají metodu learn s různými parametry a různým množstvím opakování. Tímto způsobem je dosaženo toho, že uživatel může učení sítě ovládat, viz obrázek 4.3.

- Next Step - Provede se učení pouze na jednom vstupním vektoru a zkontroluje se jeden neuron ve skryté vrstvě.
- Finish Vector - Provede se učení jednoho trénovacího vektoru (tj. zkontrolují se všechny vektory ve skryté vrstvě).
- Finish Epoch - Jednou se provede učení všech trénovacích vektorů.
- Learn - Spustí se kompletní učení sítě.
- Step by step - Postupně se provede kompletní učení celé sítě, ale učení je zpomaleno tak, aby uživatel mohl sledovat, co se děje.
- Stop - Slouží pro zastavení učení sítě v průběhu.
- Filter NN - Navíc tlačítko, které vyfiltruje skrytou vrstvu jen na unikátní neurony a smaže neurony, jejichž plocha je celá pokryta jinými neurony.


Obrázek 4.3: Tlačítka pro ovládání učení sítě

**Popisky** - Pod ovládáním učení je napsáno, co se v aktuálním kroku stalo a co se bude dít. Na obrázku můžete vidět ukázku textu, která vyskakuje pokud síť vytváří nový neuron, viz obrázek 4.4.

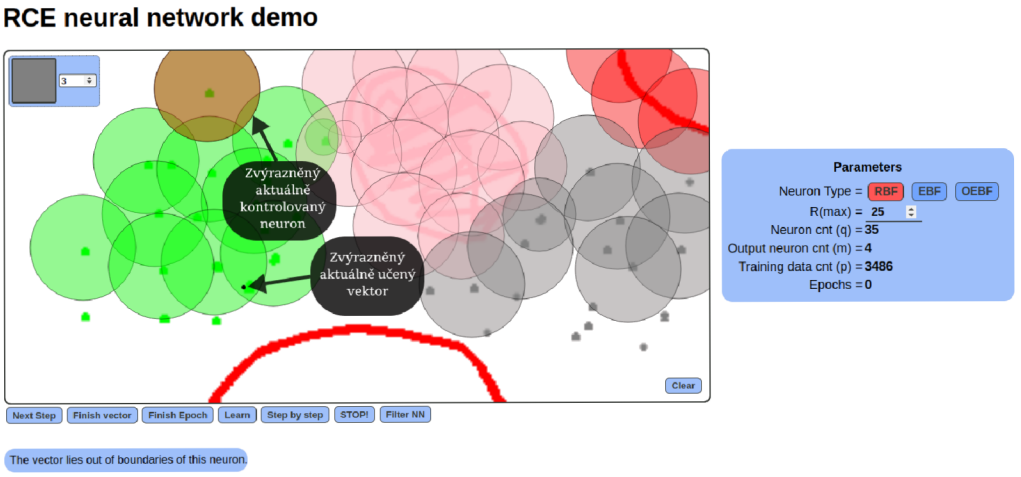



Obrázek 4.4: Popisky k učení neuronové sítě

**Nápověda** - Na spodku stránky se nachází nápověda, jak aplikaci používat. Informace, které jsou zde vypsané odpovídají informacím v této kapitole, proto zde nebudu přikládat obrázek.

## Vzhled celé aplikace

Na obrázku 4.5 vidíme kompletní design a rozpoložení jednotlivých prvků v grafickém uživatelském rozhraní




Obrázek 4.5: Celá aplikace

# Kapitola 5

# Testovací aplikace

Při implementaci jsem ze začátku pracoval s jazykem Python a knihovnou Numpy pro všechny matematické operace. Ale i přesto že jsem se snažil výpočty co nejvíce optimalizovat, tak větší testování bylo relativně zdlouhavé. Proto jsem zkusil jednoduchou verzi implementovat v jazyce C++ a po viditelném zrychlení jsem přešel do jazyka C++. Proto nadále budu popisovat jen verzi aplikace v jazyce C++. V této kapitole popíši implementaci RCE neuronové sítě s různými druhy neuronů a ostatních potřebných funkcionalit aplikace. Výsledkem bude aplikace spustitelná v terminále s parametry, které chceme otestovat. Aplikace nám vypíše průběh učení a výsledky na testovací sadě.

## 5.1 Implementace

Jelikož jsem vyvíjel demo i testovací aplikaci současně, tak se strukturálně programy moc neliší. Obě aplikace jsou založené na objektovém návrhu díky tomu, že to oba jazyky umožňují. Ale jinak se aplikace liší v implementačních detailech podle schopností daného jazyka. Tato verze aplikace má také jednodušší rozhraní RCE sítě, protože se sítí není potřeba za provozu pracovat. Nyní se v práci zaměříme na implementaci jednotlivých částí, a to v pořadí, v jakém jsou dané komponenty potřeba při provozu aplikace.

### Načítání trénovacích a testovacích dat

Data se načítají z datasetů, kde se nachází v binární podobě. Při překladu je potřeba nastavit tvar vstupních dat, a proto je při změně datasetu nutné kód upravit a aplikaci zkompilovat. Toto řešení je v aktuální verzi aplikace dostačující, protože množství testovaných datasetů není tak signifikantní. Data jsou načítána dvěma způsoby. Jedním je metoda, která data načte v původní podobě a druhým je metoda, která je načte a následně každou oblast o velikost NxM zesumarizuje na jeden bod (vytvoří součet všech hodnot uvnitř oblasti). Druhý způsob je přidaný pro testování neuronové sítě nad více kondenzovanými datasety, což nám umožňuje hlouběji porovnat prostorovou flexibilitu jednotlivých bázových funkcí.

### Klasifikace neuronovou sítí

V rozhraní sítě se nachází metoda evaluate, která na vstupu očekává testovací vektor a následně ho klasifikuje do třídy. Na rozdíl od základní RCE sítě je zde přidaný algoritmus, který umožní klasifikovat i vektory klasifikované do více tříd (v klasické RCE síti by neuron

**Algoritmus 8** Sumarizace dat

```
Vstup: vector
Vstup: squareHeight
Vstup: squareWidth
Výstup: condensedVector
for h = 0 to vectorHeight do
    for w = 0 to vectorWidth do
        heightIndex = floor(h / squareHeight);
        widthIndex = floor(w / squareWidth);
        squareIndex = heightIndex * floor(vectorWidth/squareWidth) + widthIndex;
        condensedVector[squareIndex]  vector[h *vectorWidth + w];
    end for
end for
```

nebyl klasifikován vůbec). Moje řešení v takovém případě vybere třídu aktivního neuronu, který je nejblíže vstupnímu vektoru.

**Algoritmus 9** evaluate

```
Vstup: testVector
Výstup: outputNeurons
for neuron in hiddenLayer do
    if neuron.contains(testVector) then
        outputNeuron[neuron.label] = 1
        if neuron je nejblíže k testVector then
            closestNeuron = neuron
        end if
        Continue
    end if
end for
if Více než jeden neuron ve výstupní vrstvě je aktivní then
    clearOutputNeurons()
    outputNeuron[closestNeuron.label] = 1
end if
Return outputNeurons
```

## Neurony

Jelikož bude v aplikaci potřeba testovat více neuronů, je praktické, aby skrytá vrstva mohla obsahovat kterýkoliv typ neuronů a zároveň bylo možné nad nimi provolávat správné metody. V C++ toho lze dosáhnout vytvořením společného rozhraní pro všechny třídy, které budou reprezentovat neurony. Následně pro každý druh neuronu můžeme libovolně přidávat metody a atributy. K přechodu na tento druh implementace došlo při implementaci eliptických neuronů v normální poloze, které byly implementované jako v pořadí druhé po klasických RBF neuronech. Proto je jako hlavní rozhraní zvoleno rozhraní EBF neuronů, ačkoliv by se dalo argumentovat, že OEBF jsou ještě obecnější. Implementačně v tom ale není žádný problém. Toto rozhraní se skládá z těchto komponent.

### Atributy

**center** - Středový vektor neuronu.

**focis** - Není součástí hlavního rozhraní, ale jen neuronů s EBF založenou na vzdálenosti od ohnisk.

**semiAxes** - Délky poloos bázové funkce.

**label** - Třída neuronu nahrazuje výstupní vrstvu neuronů.

**Metody**

**contains** - Slouží pro určení, jestli vektor, který ji předáme je obsažen v neuronu.

**shrink** - Zmenší délky poloos neuronu tak, aby nezahrnoval vektor, který ji předáme.

**getLabel** - Vrací třídu neuronu.

**toString** - Vytvoří textový popis neuronu.

### Matematické funkce

Jelikož C++ v základu neobsahuje matematickou knihovnu pro vektory, vytvořil jsem několik základních funkcí, které budu potřebovat. Pro optimalizaci výpočtů a vyhnutí se chybám při výpočtech s desetinou čárkou, jsou poloosy neuronů nechané v druhé mocnině a díky tomu není potřeba při výpočtech euklidovských vzdáleností použít odmocninu. Knihovna obsahuje následující funkce.

**squaredDistance** - Vypočítá druhou mocninu euklidovské vzdálenosti dvou vektorů.

$$\sum_{i=0}^{n}(x_i - y_i)^2 \tag{5.1}$$

**normalizedDistance** - Vypočítá normalizovanou vzdálenost od středu elipsy.

$$\sum_{i=0}^{n}\frac{(x_i - y_i)^2}{w_i^2} \tag{5.2}$$

**axisDistance** - Vypočítá druhou mocninu euklidovské vzdálenosti dvou vektorů, pro každou osu samostatně.

$$d_i = (x_i - y_i)^2 \tag{5.3}$$

## 5.2 Ovládání přes konzoli

Aplikace nemá aktivní uživatelské rozhraní, ale ovládá se pomocí vstupních parametrů. Zde je seznam parametrů a jejich výchozích hodnot.

**-d** Zapne výpis ladících informací. (false - vypnuto)

**-o** Vypne výpis výstupních informací. (false - vypnuto)

**--df** Nastaví výstupní soubor pro ladící informace. (stdout)

**--of** Nastaví výstupní soubor pro výstupní informace. (stdout)

**-r** Nastaví maximální délku poloos neuronů. (5 000 000)

**-n** Nastaví typ neuronu. Možné hodnoty jsou RBF, EBF a OEBF. (RBF)

**--trainLength** Nastaví počet trénovacích vstupů, které se mají projít. (60 000)

**--testLength** Nastaví počet trénovacích vstupů, na kterých se má síť otestovat. (10 000)

**--traind** Nastaví cestu k trénovacím datům. (MNIST/train-images.idx3-ubyte)

**--trainl** Nastaví cestu k trénovacím třídám. (MNIST/train-labels.idx1-ubyte)

**--testd** Nastaví cestu k testovacím datům. (MNIST/t10k-images.idx3-ubyte)

**--testl** Nastaví cestu k testovacím třídám. (MNIST/t10k-labels.idx1-ubyte)

Příklad spuštění aplikace s výchozími parametry.

```
./RCE -d -o --df debugInfo.txt --of RCEoutput.txt -r 5000000 -n RBF
--trainLength 60000 --testLength 10000 --traind MNIST/train-images.idx3-ubyte
--trainl MNIST/train-labels.idx1-ubyte --testd MNIST/t10k-images.idx3-ubyte
--testl MNIST/t10k-labels.idx1-ubyte
```

## 5.3 Příklad výstupních dat aplikace

```
Training images/labels: 60000 | Max range: 4000000
Learning process started...
Epoch - 1
images -> neurons | 60000 -> 4288
Epoch - 2
images -> neurons | 60000 -> 5839
Epoch - 3
images -> neurons | 60000 -> 6445
Epoch - 4
images -> neurons | 60000 -> 6647
Epoch - 5
images -> neurons | 60000 -> 6714
Epoch - 6
images -> neurons | 60000 -> 6719
Epoch - 7
images -> neurons | 60000 -> 6719
Epoch - 8
images -> neurons | 60000 -> 6719
Learning process finished...
RBF - 0| EBF - 6719| OEBF - 0

--------------------------------
| Guessed Right | Guessed Wrong |
| 9213          | 787           |
--------------------------------
```

# Kapitola 6

# Experimenty

V této kapitole se budeme věnovat testování různých neuronových sítí, které byly v této práci popsány. V první části budou otestovány různé parametry těchto sítí. Pomocí těchto testů se pokusíme najít parametry, které se blíží co nejvíc optimálnímu nastavení jednotlivých sítí. Poté s těmito co nejvíce optimálními parametry jednotlivé sítě porovnáme. Parametry, které budeme nastavovat, jsou zmenšovací algoritmy neuronů, algoritmy pro výběr ohnisek elips a maximální velikost poloos jednotlivých neuronů.

## 6.1 Hledání optimálních délek jednotlivých poloos

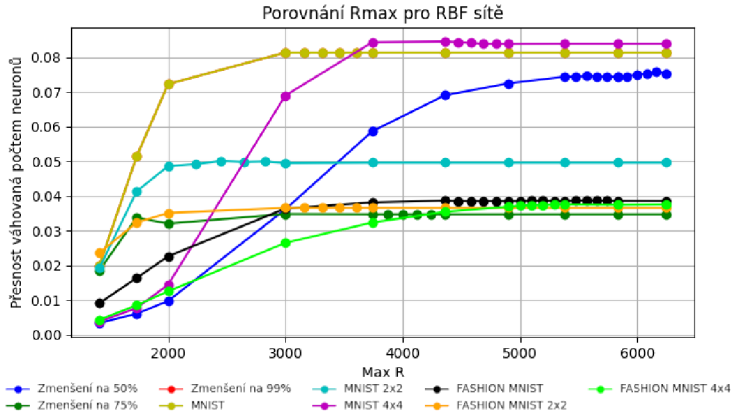
Tyto testy mají za úkol zaručit pro všechny typy neuronů, aby jejich porovnávání bylo co nejvíc fér. Testování bylo provedeno vždy na 5000 obrázcích z trénovací sady. Nejprve byly otestovány různé maximální délky s větším rozptylem. Z výsledků těchto testů byly vybrány 2 nejlepší výsledky. Na intervalu mezi nejlepším a druhým nejlepším výsledkem, byly následně provedeny další podrobnější testy. Číselně se jednalo v prvním testování o hodnoty z intervalu $(1000, 6500)$ s různými inkrementy podle toho, jak velká byla aktuální hodnota. Navíc hodnoty jsou programu předávány v druhé mocnině, kterou program využívá pro různé optimalizace. V druhé části nejde jednoznačně říct, o jaký interval se jednalo, protože pro každý druh testování se měnil. Ale inkrementy v druhé mocnině byly v této části po 1 000 000.

**RBF** Na těchto testech bylo vypozorováno, že většinou od nějaké různé hodnoty už zvětšování tohoto parametru nezlepšovalo vlastnosti dané sítě viz graf 6.1. Pro jednotlivé další testování byly vybrány tyto maximální délky viz tabulka 6.1.

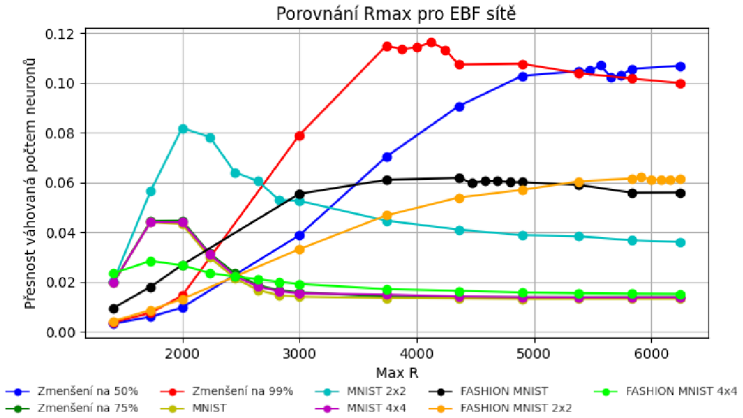
| **Zmenšování na 50 %** | **Zmenšování na 75 %** | **Zmenšování na 99 %** |
|---|---|---|
| 3000 | 2450 | 3000 |
| **MNIST** | **MNIST 2x2** | **MNIST 4x4** |
| 3000 | 4360 | 6165 |
| **FASHION MNIST** | **FASHION MNIST 2x2** | **FASHION MNIST 4x4** |
| 3320 | 5570 | 5400 |

Tabulka 6.1: Vybrané hodnoty parametru $R_{max}$ pro RBF na další testování

**EBF** Tyto sítě na rozdíl od RBF a OEBF většinou měly jednoznačnou hodnotu, která produkovala nejlepší výsledky viz graf 6.2. Zároveň tato hodnota byla většinou menší než




Obrázek 6.1: Výsledek testování RBF při hledání optimálních $R_{max}$




Obrázek 6.2: Výsledek testování EBF pri hledání optimálních $R_{max}$

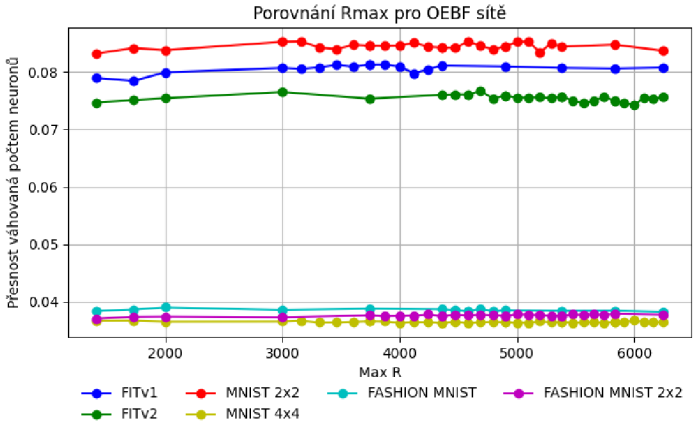
hodnota u RBF a OEBF, nejspíš díky velkému počtu jednotlivých os. Pro jednotlivé další testování byly vybrány tyto maximální délky viz tabulka 6.2.

| **Zmenšování 50 %** | **Zmenšování 75 %** | **Zmenšování 99 %** | **Zmenšování 3.3** |
|---|---|---|---|
| 1730 | 1730 | 2000 | 2000 |

| **MNIST** | **MNIST 2x2** | **MNIST 4x4** |
|---|---|---|
| 2000 | 4120 | 5570 |
| **FASHION MNIST** | **FASHION MNIST 2x2** | **FASHION MNIST 4x4** |
| 1730 | 4360 | 5920 |

Tabulka 6.2: Vybrané hodnoty parametru $R_{max}$ pro EBF na další testování

**OEBF** Zde bylo nejtěžší vybrat nějaký optimální parametr. Hlavně proto, že jednotlivé funkce měly skoro neměnný průběh viz graf 6.3. Nejspíš tomu tak je díky vytváření délky hlavní osy ze vzdálenosti obou ohnisek. Vybrané optimální parametry $R_{max}$ viz tabulka 6.3.




Obrázek 6.3: Výsledek testování OEBF při hledání optimálních $R_{max}$

| **fociFitness v1 algoritmus viz 3.3** | **fociFitness v2 algoritmus viz 3.3** |
|---|---|
| 4580 | 3870 |

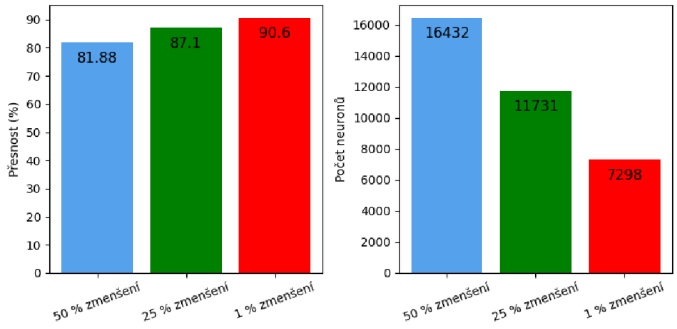
| **MNIST** | **MNIST 2x2** | **MNIST 4x4** |
|---|---|---|
| 3870 | 3160 | 4690 |
| **FASHION MNIST** | **FASHION MNIST 2x2** | **FASHION MNIST 4x4** |
| 6000 | 2000 | 5830 |

Tabulka 6.3: Vybrané hodnoty parametru $R_{max}$ pro OEBF na další testování

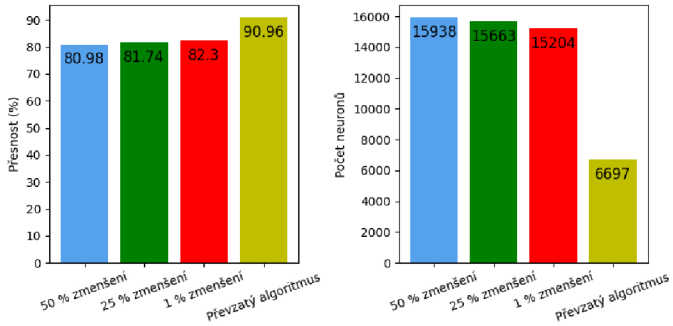
## 6.2 Testování zmenšovacích algoritmů

Tento parametr je klíčové určit co nejlepší, protože učení našich neuronových sítí je založeno na učícím algoritmu RCE sítě viz 3.2. Během tohoto učení jsou váhy neuronů, které klasifikují některý vstupní neuron špatně, zmenšovány. Pokud se budou váhy neuronu zmenšovat moc, může dojít k vytvoření velkého množství neuronů, které pokrývají malý prostor. Pokud by se ale zároveň neurony nezmenšovaly dostatečně, tak by v nejhorším případě učící proces nemohl být úspěšně ukončen a nebo by se váhy neuronu zmenšovaly v závislosti na stejném vstupním vektoru víckrát. Algoritmy budeme porovnávat v přesnosti, kterých budou dané sítě dosahovat a množství použitých neuronů. Všechny testy byly provedeny na sadě MNIST.

**RBF** Máme jeden základní zmenšovací algoritmus pro RBF neurony v RCE síti. Díky tomu, že mají jen jednu váhu, nemusíme řešit, kterou z vah zmenšit. V původních materiálech, ze kterých bylo čerpáno, se doporučuje zmenšovat váhu na 50 % vzdálenosti vstupního vektoru a středového vektoru. Dále budeme testovat i algoritmy zmenšující velikost váhy na jiné poměry vzdálenosti vstupního a středového vektoru. Jmenovitě půjde o 75 % a 99 %. Na grafu 6.4 můžeme vidět, že pro vyšší přesnost je velmi důležité udržet podoblast neuronu co nejvyšší. Algoritmus s nejmenším zmenšením, neboli algoritmus, kdy zmenšujeme váhu neuronu na 99 % vzdálenosti středového a vstupního vektoru, je v obou kategoriích značně dokonalejší.




Obrázek 6.4: Výsledek testování zmenšovacích algoritmů pro RBF



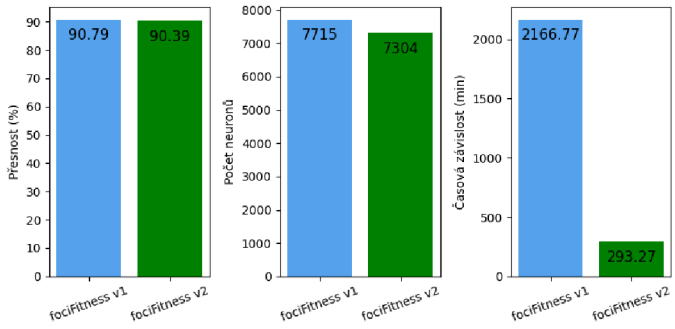

Obrázek 6.5: Výsledek testování zmenšovacích algoritmů pro EBF

**EBF** Byly porovnány 2 algoritmy. První inspirovaný zkracováním RBF neuronů, s tím rozdílem, že byla nejdřív vybrána poloosa neuronu, kde se vstupní vektor blížil nejvíce okraji. Tato poloosa byla následně zkrácena na podíl (50 % nebo 75 % nebo 99 %) vzdálenosti tohoto vektoru od středu neuronu v dané ose. Druhý algoritmus převzatý z [7], kde byly navrženy vzorečky pro výběr zkracované poloosy a výpočtu její nové délky. Z výsledného grafu 6.5 je znovu ihned jasné, že algoritmus převzatý z [7] je mnohem optimálnější než námi navržené algoritmy. V obou kategoriích se ale i tyto ostatní algoritmy zlepšují se zmenšující se redukcí vah neuronu, takový trend byl vidět i u RBF.

**OEBF** Po inspiraci předchozími výsledky nebyly prováděny testy různých velikostí zmenšení, protože je znatelné, že optimální přístup je zachovávat váhy neuronů co největší. Proto při tvorbě zmenšovacích algoritmů byl navrhnut algoritmus, který zmenšuje oblast neuronu tak, aby se nechtěné body nacházely těsně za hranicemi elipsy.

## 6.3 Testování algoritmů pro výběr ohnisek

V této sekci bylo zjišťováno, který algoritmus pro výběr ohnisek je efektivnější. V tomto případě byla porovnávána nejen přesnost a množství potřebných neuronů, ale i časová závislost jednotlivých algoritmů. První algoritmus byl založený na výběru dvou vektorů z trénovací sady jako ohnisek viz 3.3. Druhý algoritmus naopak volil vektor z trénovací sady jako střed elipsy a následně našel od něj nejvzdálenější vektor z trénovací sady. Druhý vektor byl následně použit pro vypočtení směrových vektorů ohnisek a vytvoření obou ohnisek viz 3.3.




Obrázek 6.6: Výsledek testování algoritmů pro výběr ohnisek

Z výsledků na grafu 6.6 můžeme vidět velmi podobné výsledky obou algoritmů v prvních dvou kategoriích, ale v časové závislosti se velice liší. Z tohoto důvodu byl pro další testování zvolen druhý navržený algoritmus. Navíc při testech prvního algoritmu ještě na sadě FASHION MNIST bylo odhaleno, že těžce selhává na takové datové sadě, jak v přehnaně dlouhém čekání na výsledky, tak i v neuronové závislosti, která se ještě více zhoršila.

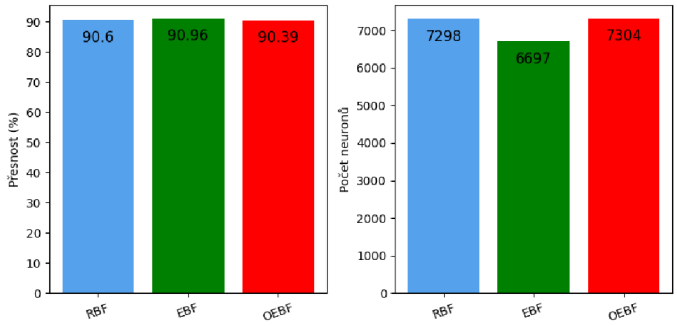
## 6.4 Sada MNIST

Jedná se o jednu z nejznámějších datových sad. Obsahuje 70 000 ručně psaných číslic, naskenovaných a uložených v bytové podobě. Každý obrázek má velikost 28x28 pixelů s hodnotami pixelů od 0 do 255, reprezentující barvu na černobílé škále. Dále sada obsahuje soubory s názvy tříd, každý obrázek má přidělenou jednu třídu reprezentovanou jedním bytem. Třídy jsou 0–9 podle toho o jakou číslici se jedná. Čísla jsou reprezentovaná přímo jejich hodnotou v decimální soustavě. Jsou tak i rovnou převedeny do hexadecimální soustavy, takže pro reprezentaci se nepoužívá jejich hodnota v ASCII tabulce. Tato sada je rozdělena do trénovací a testovací sady. Trénovací obsahuje 60 000 obrázku a testovací zbylých 10 000. Na obrázku 6.13 dále můžete vidět jednotlivé ukázky obrázků ze sady MNIST.

Po otestování všech výše zmíněných algoritmů byly vybrány ty s nejlepšími výsledky. Takže budeme testovat RBF neurony se zmenšovacím algoritmem, který zmenšuje na 99 % vzdálenosti středového vektoru a vstupního vektoru. EBF neurony, převzaté z [7]. OEBF neurony s bázovou funkcí založenou na ohniskách elipsy a využívající fociFitness v2 (3.3) algoritmus pro výběr ohnisek.

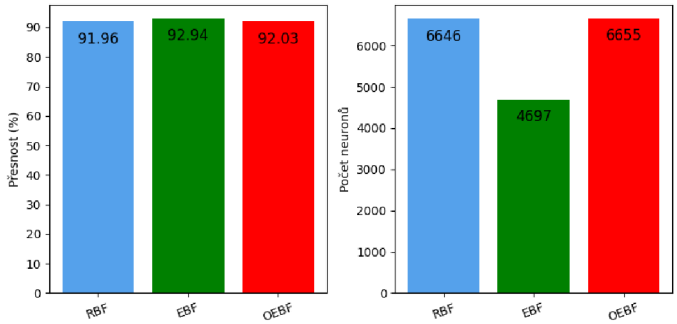
Při porovnání bylo využito i sumarizovaných verzí tohoto datasetu. V jednom případě byly všechny 2x2 oblasti pixelů sečteny do jedné hodnoty. V druhém případě byly všechny 4x4 velké oblasti sečteny. Tyto testy byly zvoleny, aby se změnil počet rozměrů vstupních dat z 28x28 na 14x14 nebo 7x7. Zároveň ale i obor hodnost jednotlivých rozměru z 0–255 na 0–1025 a 0–4080. Můžeme tak určit závislosti jednotlivých sítí na velikosti vstupních vektorů nebo rozptylu hodnot v jednotlivých osách. Protože data v těchto sumarizovaných datasetech jsou pořád stejné, ale jen reprezentované jinou podobou.

**Původní sada MNIST** Z výsledků na grafu 6.7 můžeme vidět, že všechny tři neuronové sítě mají velmi podobnou finální přesnost. Jediný větší rozdíl je v počtu neuronů. EBF neuronová síť tedy dosahuje nejlepší přesnosti i s nejmenším počtem neuronů. Překvapivě i OEBF má o něco horší výsledky než klasická RBF.



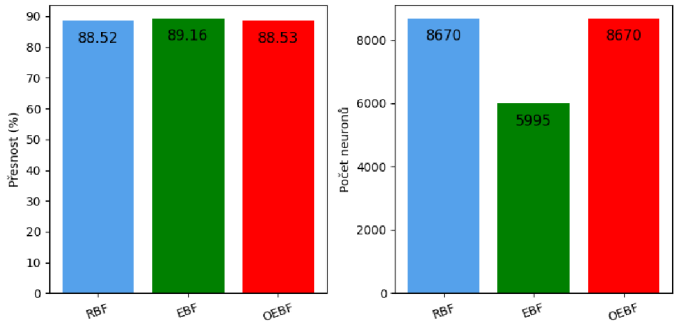

Obrázek 6.7: Výsledky testování na sadě MNIST

**Zmenšená (2x2) sada MNIST** Ve výsledcích u této sady viz 6.8 můžeme vidět podobný trend, jako na minulé sadě. Zároveň ale vidíme větší odskočení EBF neuronové sítě. Což naznačuje pozitivní vliv buď zvětšení intervalu jednotlivých rozměrů nebo snížení počtu rozměrů nebo jejích kombinace. Zároveň se dá říct že OEBF neuronové síti také tato změna prospěla, protože ačkoliv jen velmi minimálně, tak má už aspoň lepší přesnost než RBF.




Obrázek 6.8: Výsledky jednotlivých testování na 2x2 sumarizované sadě MNIST

**Zmenšená (4x4) sada MNIST** Zde na obrázku 6.9 můžeme vidět, že další větší sumarizace datasetu už EBF neuronové síti neprospěla a OEBF a RBF se naopak skoro absolutně zarovnaly, ačkoliv mají také ale horší výsledky než na předchozích sadách. Tento fakt by se mohl odůvodnit až přehnanou deformací obrázků a ztrátě informací z obrázku nebo také tím, že zlepšení na předchozí verzi sumarizovaného datasetu MNIST bylo čistě náhodné. Kvůli tomu je potřeba provést další testy.




Obrázek 6.9: Výsledky jednotlivých testování na 4x4 sumarizované sadě MNIST

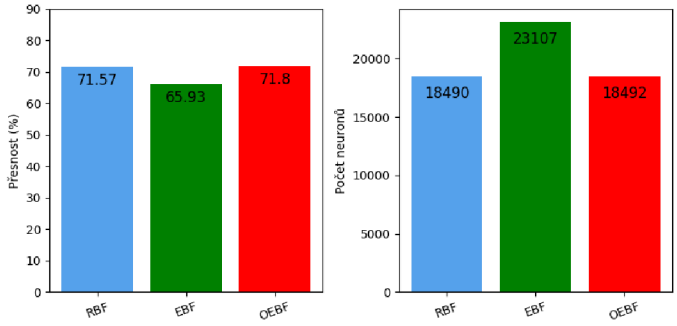
## 6.5 Sada FASHION MNIST

Jelikož všechny předchozí testy byly provedeny na jedné datové sadě, tak byla zvolená i další podobná sada pro porovnání výsledků. Tato sada má stejné parametry jako MNIST. Obsahuje 10 tříd a každý obrázek je reprezentován jako 28x28 pixelů v černobílé škále. Rozdíl je v zaplnění jednotlivých obrázků, na rozdíl od klasické MNIST sady obsahuje více zaplněné obrázky, to se hodí protože z testů na sumarizovaných sadách MNIST usuzujeme, že v takových oblastech EBF vyniká a touto sadou bychom si tuto teorii mohli ověřit. Také ale některé třídy jsou si velmi podobné. Na obrázku 6.13 dále je ukázka jednotlivých tříd uvnitř datasetu. To určitě ztíží rozpoznávání jednotlivých tříd a nejspíše povede i vytváření větších neuronových sítí s neurony s malými podprostory.

Znovu také zkusíme testovat sítě i na sumarizovaných verzích datasetu, abychom ověřili zda snížení počtu rozměrů datasetu a zvětšení jejích oborů hodnot má pozitivní vliv na výstupy neuronových sítí.

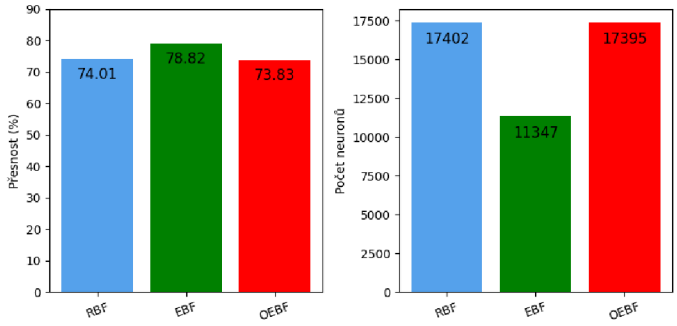
Zároveň je důležité zmínit, že testování i nadále provádíme se stejnými neuronovými sítěmi jako v podkapitole 6.4.

**Původní sada FASHION MNIST** Jak můžeme vidět viz 6.10, tak hned první testy se staví do cesty našemu tvrzení, že EBF má lepší výsledky na datech, která jsou tzv. více zaplněná, neboli v tomto případě mají méně prázdných pixelů. Zároveň RBF i OEBF pořád vykazují téměř identické výsledky.



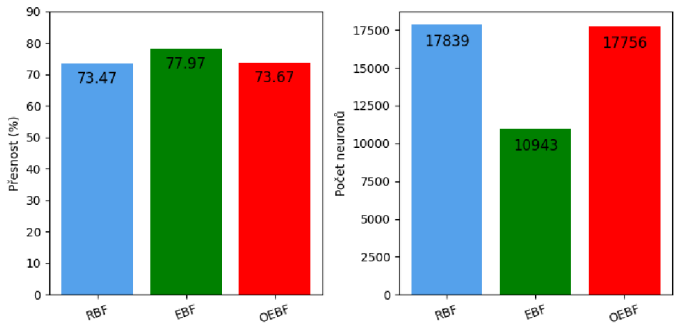

Obrázek 6.10: Výsledky jednotlivých testování sadě FASHION MNIST

**Zmenšená (2x2) sada FASHION MNIST** Na dalších testech viz 6.11 na sumarizované sadě ale zase můžeme vidět velmi výrazný skok u EBF sítě při srovnání s ostatními. Tento fakt dále potvrzuje naši myšlenku, že EBF sítě jsou silněji závislé na počtu rozměrů daného prostoru nebo oboru hodnot jednotlivých prostorů než ostatní sítě.

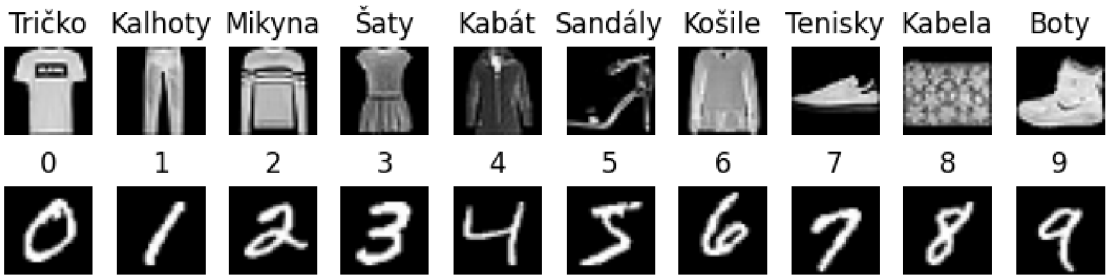



Obrázek 6.11: Výsledky jednotlivých testování na 2x2 sumarizované sadě FASHION MNIST

**Zmenšená (4x4) sada FASHION MNIST** Z posledních výsledků testů viz 6.12 vidíme, že větší sumarizace tentokrát přinesla snížení potřebného množství neuronů u EBF sítě, ale za cenu přesnosti sítě. Ostatní sítě na tuto změnu datasetu zareagovaly stejně jako u prvního datasetu.




Obrázek 6.12: Výsledky jednotlivých testování na 4x4 sumarizované sadě FASHION MNIST




Obrázek 6.13: Ukázky datasetů
Ukázka jednotlivých tříd, které datasety obsahují

## 6.6 Zhodnocení

V testech, které byly provedeny, bylo testováno několik různých vlastností těchto sítí. U některých vlastností jsme schopni z dat vyčíst i definitivní závěry.

Např. v návaznosti na práci [7], kde byl navržen algoritmus pro optimální zmenšení elipsy, byly v této práci navrženy další algoritmy pro zmenšení elips. Všechny algoritmy se inspirovaly zmenšováním RBF neuronů a hlavně oblast zmenšovaly více než algoritmus z [7]. Tyto algoritmy byly navrženy s myšlenkou, že vznikne méně oblastí, kde se podoblasti neuronů překrývají a neuronové sítě nemohou jednoznačně určit třídu vstupních dat. Opak byl ale pravdou. Testováním zmenšovacích algoritmů RBF a EBF sítí bylo dokázáno že zachováním co největších podoblastí neuronů dojde ke zvýšení přesnosti neuronové sítě a hlavně k velkému poklesu potřebných neuronů. Řádově tedy např. u EBF sítí došlo ke zlepšení přesnosti o 8 % a snížení počtu neuronů o 56 % s využitím tohoto optimálního algoritmu.

Dále byl testováním vybrán vhodnější z algoritmů pro výběr ohnisek. Bohužel k tomuto algoritmů nemáme žádné srovnání.

Následně jsme vybrané nejlepší sítě testovaly na sadách MNIST a FASHION MNIST. Cílém bylo ukázat vylepšené vlastnosti eliptických bázových funkcí. Toto se nám podařilo s klasickým eliptickým řešením s elipsami v normální poloze, kde jako bázovou funkci využíváme středovou funkci elipsy. Toto řešení bylo převzato z [7], kde byly ty neurony využity v neuronové síti sestrojené pro rozpoznávání všech písmen číslic a speciálních znaků. V porovnání s jejich řešením mají ale neuronové sítě trénované na podobný problém sadou MNIST, v této práci výrazně nižší přesnost v průměru o 8 %. Toto bude zapříčiněno tím, že v té práci bylo pro učení využito více vektorů derivovaných z každého obrázku. Např. jedním z nichž byly i sumarizované vektory. U těchto vektorů jsem prokázaly výrazné zlepšení eliptických funkcí. Nejspíš v závislostí na snížení počtu vah, které následně každý takový neuron obsahuje, protože při porovnání výsledků na sadě MNIST a FASHION MNIST by se dalo s jistotou říct, že větší zaplnění vstupních vektorů nemělo pozitivní vliv na výkon této sítě.

Případně bychom naše výsledky mohli srovnat ještě s prací [12], kde byly porovnávány RBF a EBF neuronové sítě na klasifikaci hudebních nástrojů. Kde např. RBF neuronové sítě dosahovaly lepších výsledků než EBF neuronové sítě. A to v obou testovaných kategoriích. Takovýchto podobných výsledků jsme dosáhli s naším řešením OEBF, což bylo relativně překvapivé.

Např. v práci [13], kde byla navržena OEBF neuronová síť a následně i byla otestována na široké škále datasetů a porovnána s výsledky MLP[1], RBF sítí, EBF sítí a dalšími mechanizmy strojového učení. Tak tato neuronová síť dosahovala v průměru nejlepších výsledků. Dále také v práci [3] bylo dosaženo eliptického tvaru bázové funkce neuronů pomocí algoritmu geometrické analýzy a následně hybridním trénovacím algoritmem. Díky tomu že obě tyto práce zaznamenaly zlepšení minimálně vůči RBF sítím a v případě [13] i vůči EBF sítím se dá předpokládat, že náš navržený způsob pro dosažení OEBF nenaplňuje potenciál OEBF.

Rozdíl mezi RBF a OEBF řešením v této práci je minimální, což naznačuje že ve finále se během učení vytvořené bázové funkce neuronů smrskly zpět do podoby hyperkoule. Díky tomu usuzuji, že tento problém mohl zapříčinit algoritmus pro výběr ohnisek elipsy. Jelikož v definici této funkce by chyba být neměla. Zároveň je i kompletně možné že dosažení nějakých

[1] Neboli Multilayer perceptron je neuronová síť, která obsahuje několik vrstev LBF neuronů a často se používá práce pro klasifikaci.

smysluplných výsledků touto cestou není možné. Zatím ale věřím, že se zlepšením výběru ohnisek by tento přístup mohl představovat určitý krok vpřed. Minimálně v oblastech jako např. v práci [12], kde RBF sítě měly větší úspěch, protože naše řešení se zpracováním více podobá.

# Kapitola 7

# Závěr

Konstantním problémem oblasti neuronových sítí je vysvětlení procesů a algoritmů využívaných pro jejich funkci. Tato práce měla za cíl vytvořit stručný přehled neuronových sítí s důrazem na oblast RBF neuronových sítí. Tohoto cíle bylo dosaženo i za pomocí vytvoření demonstrační aplikace, která uživateli přibližuje základní princip učení RCE neuronové sítě. Zároveň uživateli umožňuje učení do určité míry parametrizovat a nechat ho tak porovnávat chování sítě s měnícími se parametry.

Dále se také práce věnovala méně známým EBF neuronovým sítím. Byly představeny některé známé přístupy, ale zároveň byl navržen i nový přístup založený na ohniskách elipsy. Bylo vytvořeno několik podpůrných algoritmů pro úpravy a vytváření neuronů s touto OEBF a následně byly otestovány a porovnány s klasickými přístupy.

Při testování byly dokázány lepší vlastnosti EBF neuronových sítí v klasifikaci různých obrázku. Bohužel ale také byly odhaleny nepříznivé výsledky pro naše řešení. Toto řešení s malými odchylkami produkuje stejné výsledky jako RBF neuronové sítě, což naznačuje zpětnou degradaci tvaru bázové funkce těchto neuronů během učení.

Dále ale věřím, že navržený přístup může dosáhnout lepších výsledků po vytvoření přesnějších algoritmů pro selekci ohnisek elipsy. Např. znovu inspirací z přístupu využívaných u RBF sítí využít klasické shlukovací algoritmy pro vytvoření základní definice shluků, které chceme proložit.

# Literatura


[1] DENNIS andikat. *What is the general equation of the ellipse that is not in the origin and rotated by an angle?* Mathematics Stack Exchange. 2020. Dostupné z: https://math.stackexchange.com/q/434482. Author: (https://math.stackexchange.com/users/82597/andikat-dennis).

[2] FJODOR VAN VEEN, S. L. *THE NEURAL NETWORK ZOO*. 2019. Dostupné z: https://www.asimovinstitute.org/neural-network-zoo.

[3] JI XIANG DU, Z.-F. W. A Novel Elliptical Basis Function Neural Networks Model Based on a Hybrid Learning Algorithm. *Advances in Neural Networks - ISNN 2007*. 1. vyd. Springer Berlin, Heidelberg, 2007, č. 1, s. 1153–1161.

[4] KRIESEL, D. *A Brief Introduction to Neural Networks*. 2007. Dostupné z: http://www.dkriesel.com.

[5] MAK, M.; ALLEN, W. a SEXTON, G. Speaker identification using radial basis functions. In: *1993 Third International Conference on Artificial Neural Networks*. 1993, s. 138–142.

[6] MEHROTRA, K.; MOHAN, C. a PREFACE, S. *Elements of Artificial Neural Nets*. The MIT Press, 1997. ISBN 9780262133289.

[7] MOED, M. a LEE, C.-P. Design of an elliptical neural network with application to degraded character classification. In: *IEEE International Conference on Neural Networks*. 1993, s. 1576–1582 vol.3.

[8] MOZILLA. *Mdm Web Docs. <canvas>: The Graphics Canvas element* online. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas. [cit. 2024-04-10].

[9] MOZILLA. *Mdm Web Docs. CSS: Cascading Style Sheets* online. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS. [cit. 2024-04-10].

[10] MOZILLA. *Mdm Web Docs. HTML: HyperText Markup Language* online. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML. [cit. 2024-04-10].

[11] MOZILLA. *Mdm Web Docs. TypeScript* online. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/TypeScript. [cit. 2024-04-10].

[12] PARK, T. H. a COOK, P. Radial/Elliptical basis function neural networks for timbre classification. In: Association Française d'Informatique Musicale and Centre de recherche en Informatique et Création Musicale. *Journées d'Informatique Musicale*

*2005*. Saint-Denis, France: [b.n.], červen 2005. Dostupné z: https://univ-paris8.hal.science/hal-03114906.

[13] Peng, H.-W.; Lee, S.-J. a Lee, C.-H. An oblique elliptical basis function network approach for supervised learning applications. *Applied Soft Computing*, 2017, sv. 60, s. 552–563. ISSN 1568-4946. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1568494617304301.

[14] Russell, S. a Norvig, P. *Artificial Intelligence: A Modern Approach*. 3. vyd. Prentice Hall, 2010. Dostupné z: https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf.

[15] Sato, K.; Shah, S. a Aggarwal, J. Partial face recognition using radial basis function networks. In: Storms, P., ed. *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*. 1998, s. 288–293. ISBN 0-8186-8344-9.

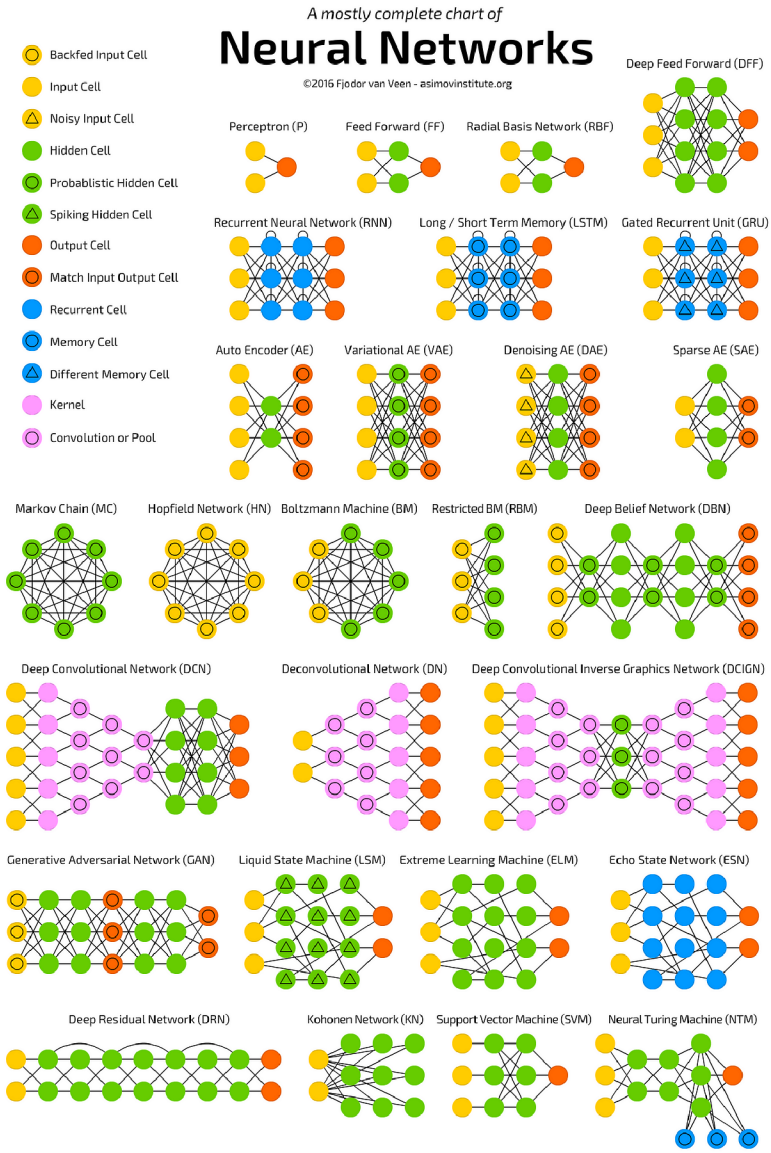
[16] Schilling, R.; Carroll, J. a Al Ajlouni, A. Approximation of nonlinear systems with radial basis function neural networks. *IEEE Transactions on Neural Networks*. 1. vyd., 2001, sv. 12, č. 1, s. 1–15.

[17] Thomaz, C.; Feitosa, R. a Veiga, A. Design of radial basis function network as classifier in face recognition using eigenfaces. In: *Proceedings 5th Brazilian Symposium on Neural Networks (Cat. No.98EX209)*. 1998, s. 118–123.

[18] Zbořil, F. *Neuronové sítě RBF a RCE. Materiály k přednáškám předmětu Softcomputing na FIT VUT v Brně.*

# Příloha

Přehled nejznámějších neuronových sítí




Obrázek 1: Přehled neuronových sítí
převzat z [2] beze změny