Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Alternativní techniky trénování vrstevnatých neuronových sítí

Plzeň 2022 Jindřiška Reismüllerová

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

Studijní program: Inženýrská informatika
Forma studia: Prezenční
Obor/kombinace: Medicínská informatika (MDInp)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení: **Bc. Jindřiška REISMÜLLEROVÁ**
Osobní číslo: **A21N0001P**
Adresa: Slavíčkova 1681, Sokolov, 35601 Sokolov 1, Česká republika

Téma práce: Alternativní techniky trénování vrstevnatých neuronových sítí
Téma práce anglicky: Alternative Approaches to Layered Neural Network Training

Vedoucí práce: Ing. Kamil Ekštein, Ph.D.
Katedra informatiky a výpočetní techniky

Zásady pro vypracování:

1. Seznamte se s používanými technikami trénování vrstevnatých neuronových sítí (zejména MLP), prostudujte odbornou literaturu z předmětné oblasti, zejména publikace pokusů o návrhy alternativních technik učení (k technice Backpropagation).
2. Navrhněte alespoň jednu inovativní alternativní techniku učení vrstevnaté neuronové sítě na jiném základě než je Backpropagation. Pokuste se využít poznatky z oblasti fyziologie nervové soustavy živých organismů.
3. Implementujte navrženou techniku, demonstrujte její použitelnost na trénování vícevrstvého perceptronu. Při trénování využijte běžné testy učení (např. trénink funkce XOR) a veřejně dostupné trénovací sady dat.
4. Vyhodnoťte dosažené výsledky, zejména obecně schopnost techniky natrénovat neuronovou síť, rychlost trénování, stabilitu konvergence, atp. Porovnejte výsledky se standardní technikou Backpropagation. Vše důkladně popište.

Seznam doporučené literatury:

Rámcově Nicholls, J. G., Robert Martin, A., Wallace, B. G., Fuchs, P. A.: Od neuronu k mozku, Academia 2013, ISBN 978-80-200-2155-7. Další dodá vedoucí práce dle potřeby.

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 19. května 2022

Jindřiška Reismüllerová

Prohlášení o ochranných známkách

Názvy programových produktů, technologií, firem apod. použité v textu mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Poděkování

Ráda bych poděkovala Ing. Kamilu Ekšteinovi, Ph.D. za odborné vedení této práce, věcné připomínky a cenné rady.

Abstract

The main goal of this master's thesis is to design and implement an innovative technique of neural network training, an alternative to backpropagation. The idea is to design a method more corresponding to real biological processes of the central nervous system of living organisms, than the backpropagation technique is. This way, faster neural network training could be achieved even at the expense of mild decrease of training accuracy. The described process is preceded by a thorough analysis of existing solutions to the problem. The result of this thesis should be an implementation of the proposed technique and it's comparison with backpropagation and other existing techniques of neural network training.

Abstrakt

Cílem této diplomové práce je navrhnout a implementovat novou techniku učení neuronových sítí alternativní k metodě backpropagation. Smyslem je zkusit navrhnout takovou metodu, která by lépe reflektovala skutečné biologické procesy centrální nervové soustavy živých organismů než technika backpropagation, čímž by se dosáhlo větší rychlosti učení i za cenu mírného snížení úspěšnosti učení neuronové sítě. Celé práci předchází důkladná analýza existujících řešení. Výsledkem by měla být implementace navrženého řešení a její porovnání s technikou backpropagation a dalšími existujícími technikami.

Obsah

1 Úvod

Učení je přirozenou vlatností všech živých organismů. „Učení jako celoživotní proces ve vývoji lidstva i jedince je základnou pro vytváření nejvyšších hodnot člověka“ [38]. Důležitým prvkem v procesu učení je snaha poučit se z vlastních i cizích chyb a neopakovat je. Centrální nervová soustava člověka je uzpůsobena k procesu učení, otázkou ale zůstává, jak tuto schopnost popsat, či dokonce přenést na neživý stroj?

Díky neustálému vývoji technologií si lidstvo může usnadňovat práci použitím nejrůznějších strojů. K jejich správnému fungování je však zapotřebí jasný popis práce, kterou mají vykonávat. Co ovšem když budeme chtít, aby stroj reagoval na situace, jež mu nebyly popsány? V takovém případě se snažíme vytvořit umělou inteligenci. Ač by se myšlenka umělé inteligence mohla zdát poměrně novou, není tomu zcela tak. Však už Karel Čapek ve své knize R.U.R. v roce 1920 [6], ještě před vznikem prvních počítačů (první samočinné programovatelné počítače byly vyrobeny až ve 30. letech 20. století) [3], popsal jakousi umělou bytost, která vykazovala známky inteligence.

Technologie pro řešení úloh, které musel dříve zastávat člověk, se postupně vyvinuly od pravidlově řízených k takovým, kterým dnes říkáme umělá inteligence. Jejich výhodou je, že po absolvování fáze učení jsou schopné korektně reagovat na vstupy, které nikdy předtím neviděly. Jednou z těchto technologií umělé inteligence jsou neuronové sítě. Jejich vznik se datuje do druhé poloviny 20. století [42], avšak jejich největší rozmach je patrný v posledních dvou desítkách let. Dnes již na neuronových sítích stojí nezanedbatelná část aplikací používaných širokou veřejností. Avšak své zastoupení mají neuronové sítě i ve strojním průmyslu, medicíně, meteorologii a v celé řadě dalších odvětví. Ačkoli se struktura neuronových sítí liší podle druhu úlohy, kterou řeší, v naprosté většině případů probíhá jejich učení na principu dopředného (forward propagation) a zpětného šíření (backpropagation). Algoritmus zpětného šíření s sebou však přináší řadu problémů, jež budou dále v této práci popsány. A právě vytvořením nového algoritmu, který by vyřešil problémy zpětného šíření, se tato práce bude zabývat.

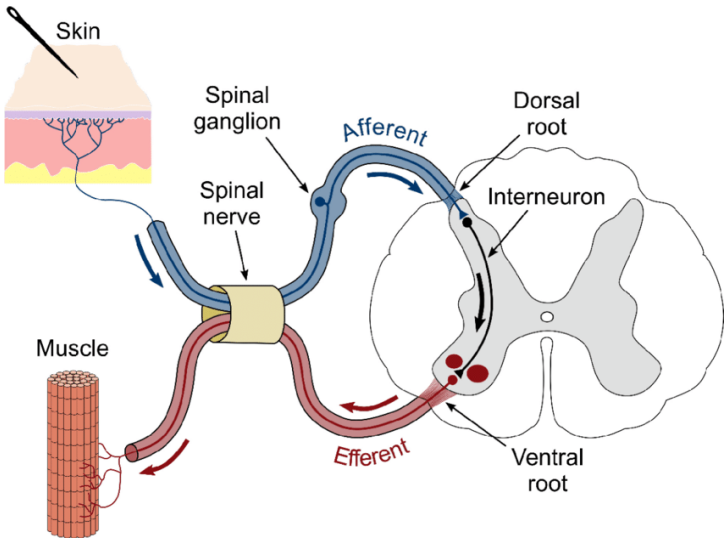
2 Biologické základy učení

Cílem umělé inteligence je imitovat lidské myšlení, a přirozeně se tedy snaží inspirovat fungováním centrální nervové soustavy člověka. Z tohoto důvodu je třeba nastínit základní biologické procesy v mozku člověka. V této kapitole proto budou stručně popsány základní fyziologické struktury a procesy, které mají zásadní vliv na učení člověka.

2.1 Reflexy

Saladin [33] definuje reflexy jako „rychlé, mimovolní, stereotypní reakce žláz nebo svalů na stimulaci.“ Viscerální reflexy, které řídí fungování vnitřních orgánů, jsou esenciální pro správné fungování lidského organismu a vytváří se ještě před narozením nebo krátce po něm. Druhým druhem reflexů jsou somatické reflexy, které zajišťují motorické reakce v kosterních svalech.

Z fyziologického hlediska jsou reflexy uskutečňovány reflexním obloukem (viz 2.1). Reflexní oblouk je nervová dráha vedoucí z kosterního svalstva do míchy a nazpět. Přes aferentní nervy se nejprve předá signál v rámci reakce na podnět do míchy, která následně vyvolá přes eferentní nervy odpověď. Díky reflexnímu oblouku tak dochází jen k minimálním synaptickým zpožděním.



Obrázek 2.1: Reflexní oblouk (převzato z [34]).

Mnohem zajímavější pro tuto práci je však psychologický pohled na navozování podmíněných reflexů. Zde se totiž setkáváme, jak vysvětluje Sovák [38], doplněním Pavlovovy klasické reflexní teorie, s učením jakožto „navazováním podmíněných reflexů (v činnostech, chování a jednání) a podmíněných spojů (v nabývání poznatků, tvoření asociací, v myšlení).“

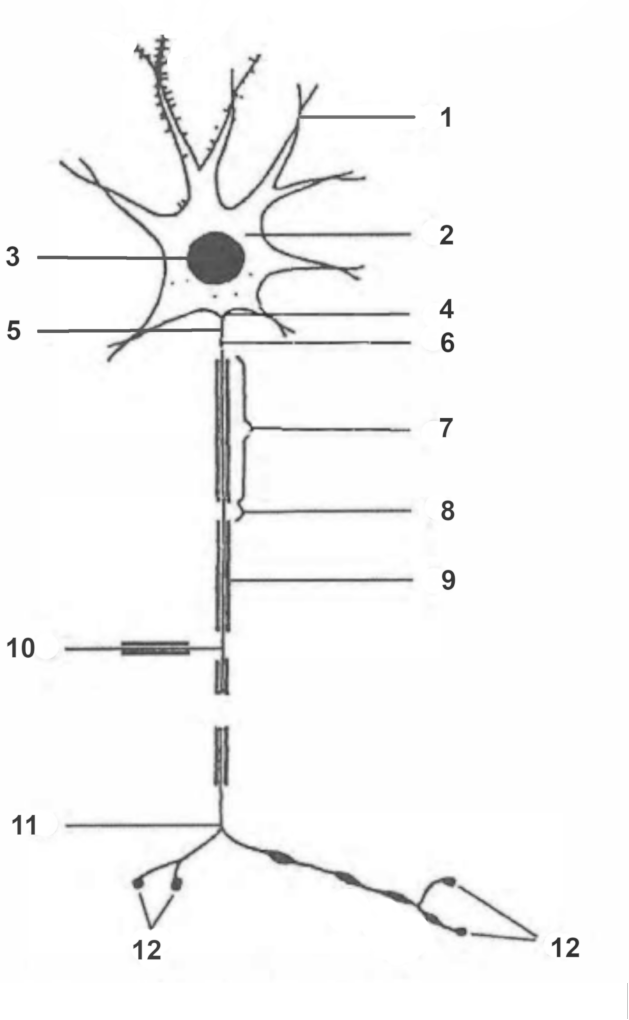
2.2 Centrální nervová soustava

U člověka se proces učení váže na funkce centrální nervové soustavy. Ta se skládá z mozku a míchy. Za učení je zodpovědná část mozku zvaná hipokampus, jež je umístěná v mozku ve střední části spánkového laloku. Hipokampus hraje rozhodující roli při vytváření paměti tím, že zajišťuje mozku časoprostorový rámec, díky kterému lze později vědomě vyvolat vzpomínku na určitou událost [23].

2.3 Neuron

Komunikace v nervové soustavě je uskutečňována prostřednictvím nervových buněk neboli neuronů [33]. Neuronů je několik druhů, avšak pro následné pochopení matematického modelu neuronu je důležitý klasický, multipolární neuron. Tento neuron je znázorněn na obrázku 2.2, přičemž směr postupu signálu je od dendritů přes soma, axon až k synapsím.

1. **Dendrity** – slouží k přijímání informací dostředivě.
2. **Soma** – tělo buňky.
3. **Jádro**
4. **Axonový hrbolek** – k šíření akčního potenciálu (odstředivě).
5. **Axolemma** (Mauthnerova pochva)
6. **Iniciální segment**
7. **Internodium**
8. **Ranvierův zářez** – zvýšená koncentrace iontových kanálů.
9. **Myelinová pochva** – izolační vlastnosti.
10. **Kolaterála** – boční větev axonu, umožňuje síťové propojení neuronu.



Obrázek 2.2: Model biologického neuronu. Založeno na [40].

11. **Telodendrion** – větevnaté zakončení axonu, zvyšuje komunikační schopnost.

12. **Synapse** – napojení na dendrity dalšího neuronu.

3 Neuronové sítě

V této kapitole budou popsány obecné principy neuronových sítí. Zvláštní pozornost bude věnována technice zpětného šíření (backpropagation), jejíž pochopení je zásadní pro celou tuto práci.

3.1 Principy

Neuronové sítě jsou jedním z nejrozšířenějších mechanismů umělé inteligence. Dají se využít jak k učení bez učitele, kdy v procesu učení nejsou známy správné výsledky, a nelze tedy na jejich základě ověřit správnost fungování neuronové sítě, tak k učení s učitelem. Tato práce se bude soustředit na klasifikační úlohy učení s učitelem, kdy je předem známý počet klasifikačních tříd. Takováto neuronová síť prochází několika fázemi.

1. **Struktura neuronové sítě** – První fází je určení struktury neuronové sítě (kolik bude mít vrstev, kolik bude v každé vrstvě neuronů, výběr somatické funkce, implicitní nastavení vah).

2. **Učení** – Druhou fází je proces učení neuronové sítě, při kterém se nastaví veškeré váhy v síti tak, aby síť správně klasifikovala všechny prvky vstupní trénovací množiny. Učení má zpravidla následující dvě fáze:

   - dopředné šíření
   - zpětné šíření.

   Tyto fáze budou podrobněji popsány v kapitole 4.

3. **Klasifikace** – Takto vytvořený model neuronové sítě by měl být schopen klasifikovat jakékoli vstupy do daných tříd s dostatečnou přesností.

Neuronové sítě se používají k řadě úloh, a to jak klasifikačních, tak regresních.

- **Klasifikační úloha** je takový druh úlohy, ve kterém je každému ze vstupních vektorů x_i (z množiny všech vstupů X) přiřazena právě jedna z klasifikačních tříd. Obvykle je tato třída označena přirozeným číslem, v případě učení pomocí neuronových sítí je však zvykem příslušnost k dané třídě reprezentovat pomocí tzv. one-hot vektoru.

Tento vektor má velikost shodnou s počtem možných klasifikačních tříd a obsahuje nuly na všech pozicích kromě jedné, která odpovídá dané třídě. Na této pozici je hodnota 1. Tato reprezentace je vhodná zejména k realizaci algoritmu zpětného šíření, který je popsán v kapitole 4.

Typickým příkladem klasifikačních úloh jsou např. bankovní problémy, ve kterých je každý klient klasifikován dle jeho schopnosti splácet dluh, a lze se tedy rozhodnout, zda mu banka poskytne úvěr. Rovněž lze do této skupiny zařadit i diagnostické problémy z lékařství, ve kterých je pacientovi dle jeho příznaků doporučeno podrobné vyšetření. Dalším typickým problém je detekce spamu, tedy klasifikace elektronické pošty na žádanou a nevyžádanou, a další [41].

- **Regresní úloha** je takový druh úlohy, ve které je každému ze vstupních vektorů x_i přiřazeno reálné číslo. Význam tohoto čísla je závislý na problému. Typickým příkladem regresní úlohy je jakákoliv predikce. Může jít například o predikci vývoje ceny ropy, predikce venkovní teploty a jiných spojitých veličin.

V neposlední řadě lze neuronové sítě použít k rozpoznávání vzorů, aproximaci funkcí, filtraci signálu a dalším úlohám.

3.2 Druhy neuronových sítí

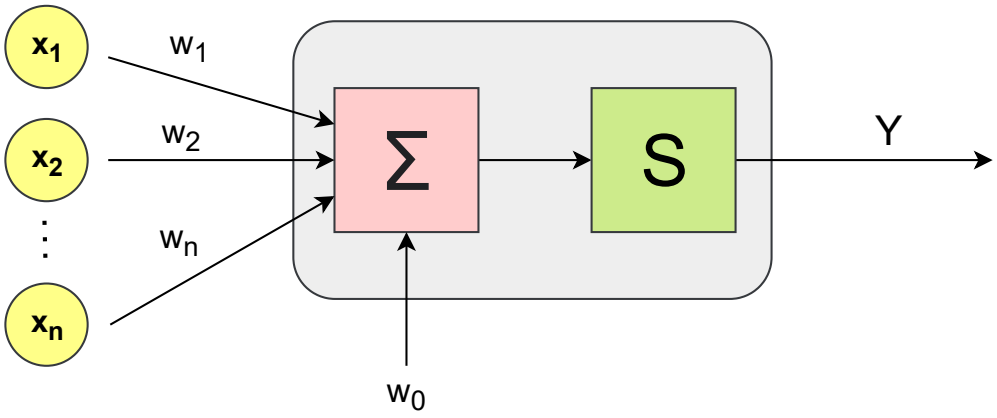
Neuronové sítě lze dělit na dopředné a rekurentní [37]. Mezi dopředné se řadí jednovrstvý perceptron, vícevrstvý perceptron nebo konvoluční neuronové sítě (CNN – Convolutional Neural Network). Rekurentní pak můžeme dělit na klasické rekurentní a takzvané LSTM (Long Short-Term Memory) neuronové sítě.

V této kapitole budou výše zmíněné druhy neuronových sítí stručně popsány. Je ovšem třeba podotknout, že se jedná pouze o podmnožinu druhů neuronových sítí, vybranou k vytvoření lepší představy o rozmanitosti této problematiky.

3.2.1 Perceptron

Perceptron je základní stavební jednotkou neuronové sítě. Jedná se o matematický model vycházející ze struktury biologického neuronu. Jedním z nej-

rozšířenějších je McCulloch-Pittsův model umělého neuronu. McCulloch-Pittsův model umělého neuronu je zobrazen na schématu 3.1.



Obrázek 3.1: Model umělého neuronu. Založeno na [10].

Složky vstupního vektoru jsou označeny jako x_1 až x_n. Každá tato složka vektoru je váhována pomocí příslušné synaptické váhy w_1 až w_n. Vstupy jsou příslušnými synaptickými vahami přenásobeny a následně se provede zvolená matematická operace nad výsledky. V případě McCulloch-Pittsova neuronu je tato operace suma, avšak obecně se může používat i jiná operace. Skalár w_0 slouží k prahování. Výsledek této naprahované sumy je vstupem do aktivační funkce S (někdy též somatická nebo přenosová funkce). Y znázorňuje výstup neuronu. Tento model lze popsat rovnicí 3.1.

$$Y = S(\sum_{i=1}^{N}(w_i x_i) + w_0) \tag{3.1}$$

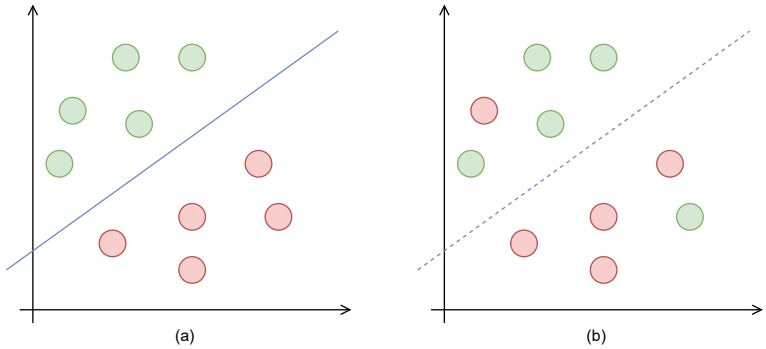
Aktivační funkce S se zpravidla implementace od implementace liší. Avšak nejčastější podoby této funkce jsou logistická sigmoida, po částech lineární funkce, skoková funkce, nebo hyperbolický tangens.

- **skoková funkce**
  - předpis: $S = 0$ pro $x < \Theta$, $S = 1$ pro $x \geq \Theta$
  - obor hodnot: $H(S) = \{0, 1\}$
- **logistická sigmoida**
  - předpis: $S = \frac{1}{1+e^{-kx}}$
  - obor hodnot: $H(S) = (0; 1)$

- **hyperbolický tangens**
  - předpis: $S = \frac{2}{1+e^{-kx}} - 1$
  - obor hodnot: $H(S) = (-1; 1)$

3.2.2 Jednovrstvý perceptron

Jednou ze základních architektur neuronové sítě je jednovrstvý perceptron. Vzniká paralelním uspořádáním perceptronů do jedné vrstvy. Počet perceptronů odpovídá počtu výstupů. Tuto architekturu lze použít pro řešení lineárně separabilních úloh, tedy takových, kde od sebe lze vstupní data jednotlivých tříd oddělit přímkou. Příklad a protipříklad takové úlohy lze vidět na obrázku 3.2.



Obrázek 3.2: Příklad lineárně separabilní (a) a lineárně neseparabilní (b) množiny dat.

K řešení úloh, u kterých není předem známý charakter vstupních dat, a tedy jejich rozložení v prostoru, však není jednovrstvý perceptron vhodný. Pro takové úlohy je vhodnější použití vícevrstvého perceptronu, či jiných hlubokých neuronových sítí, které problém žádoucí nelinearity rozhodovacích hranic řeší.

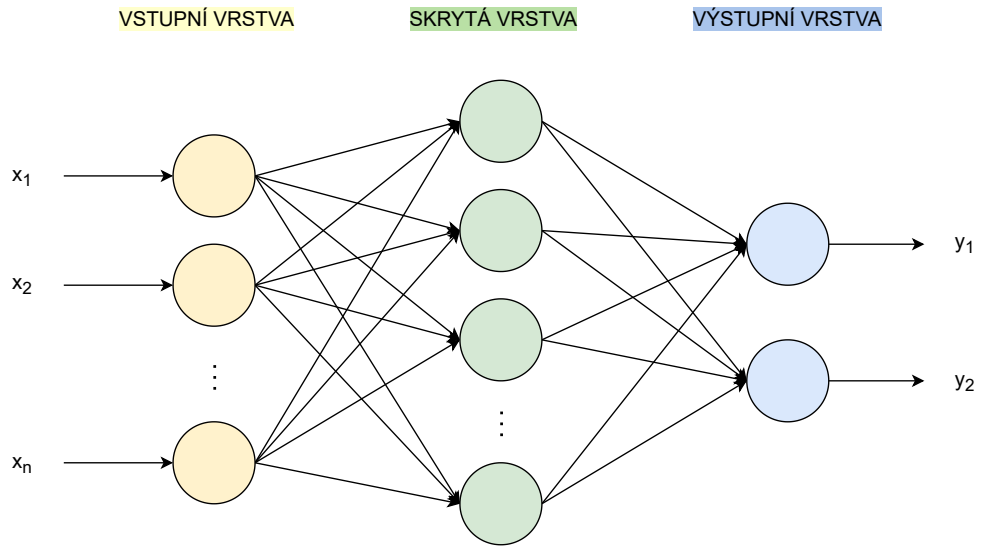
3.2.3 Vícevrstvý perceptron

Vícevrstvý perceptron je jedním ze zástupců dopředných neuronových sítí. Sestává alespoň ze tří vrstev, přičemž první je vrstva vstupní, poslední je výstupní a zbylé jsou vrstvy tzv. skryté. Aktivace každého neuronu dané vrstvy je navázána na všechny vstupy neuronů vrstvy následující. Postupným přidáváním skrytých vrstev lze zvyšovat komplexitu rozhodovacích hranic jednotlivých klasifikačních tříd.

Rozhodovací hranice je tvořena lineárními segmenty, které vznikají postupným dělením prostoru. Přidáním dostatečného množství skrytých neuronů lze libovolně komplexní rozhodovací hranici aproximovat [8].

Dle některých studií [31] lze však dosáhnout ekvivalentních výsledků za použití pouze jedné skryté vrstvy s dostatečným množstvím neuronů.

Na obrázku 3.3 je znázorněn vícevrstvý perceptron s jednou skrytou vrstvou.



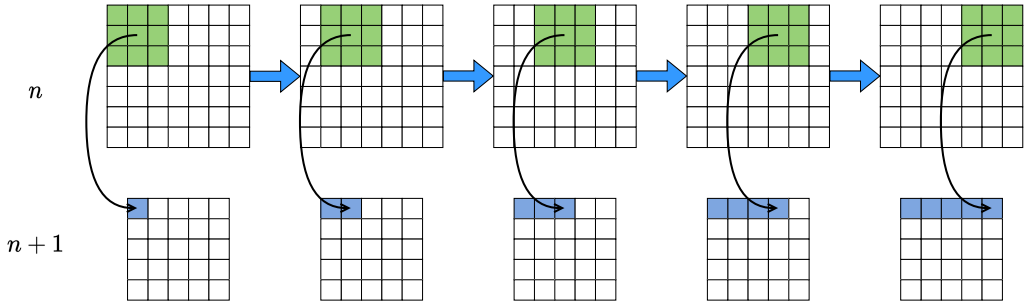
Obrázek 3.3: Architektura vícevrstvého perceptronu dle [12].

3.2.4 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou zvláštním druhem neuronových sítí a využívají se především k analýze prostorově uspořádaných dat (např. obrázků, signálu) [1]. Jejich hlavním znakem je použití speciálních druhů vrstev. Mezi tyto vrstvy patří například vrstva konvoluční, podvzorkovací, ReLU vrstva a další.

Konvoluční vrstva je fundamentálním stavebním kamenem tohoto druhu sítě. Jednotky této vrstvy detekují vzory ve vstupních datech za použití konvolučního jádra a konvoluční operace (viz obrázek 3.4). Konvoluce je druh operace, při které je konvoluční jádro postupně aplikováno na všechny položky vstupních dat, čímž generuje data pro další vrstvu. Vrstva podvzorkovací redukuje dimenzi vstupních dat za účelem snížení výpočetní náročnosti a zrychlení učení, přičemž se snaží zachovat všechny důležité detekované

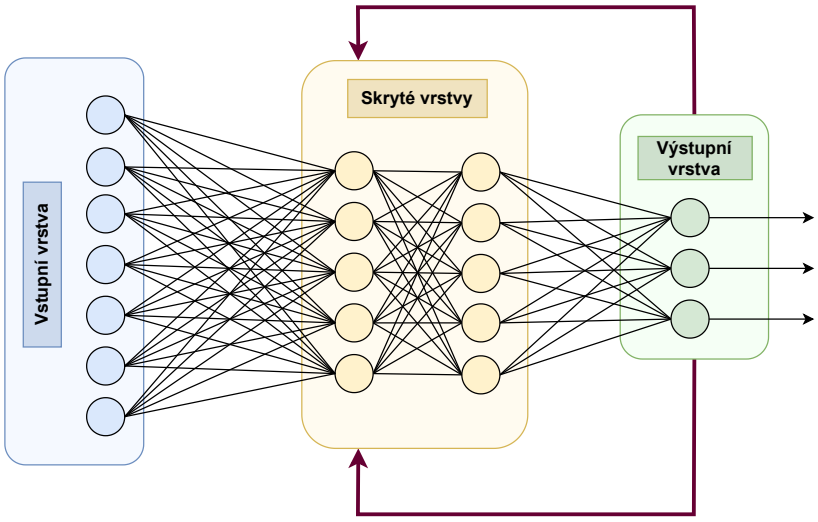
příznaky. Vrstva ReLU (Rectified Linear Unit) pak pouze svými jednotkami jednotlivé aktivace zdola ořezává tak, aby kladné aktivace nezměnila a záporné aktivace nastavila na 0.



Obrázek 3.4: Příklad použití konvoluce na vstupní data. Převzato z [1].

3.2.5 Rekurentní neuronové sítě

Rekurentní neuronová síť (RNN) je typ umělé neuronové sítě, která pracuje nad sekvenčními daty nebo časovými řadami. Tento typ neuronových sítí se běžně používá pro řešení časově nebo logicky uspořádaných problémů, jako je překlad jazyka, zpracování přirozeného jazyka, rozpoznávání řeči nebo tvorba titulků k obrázkům [20].



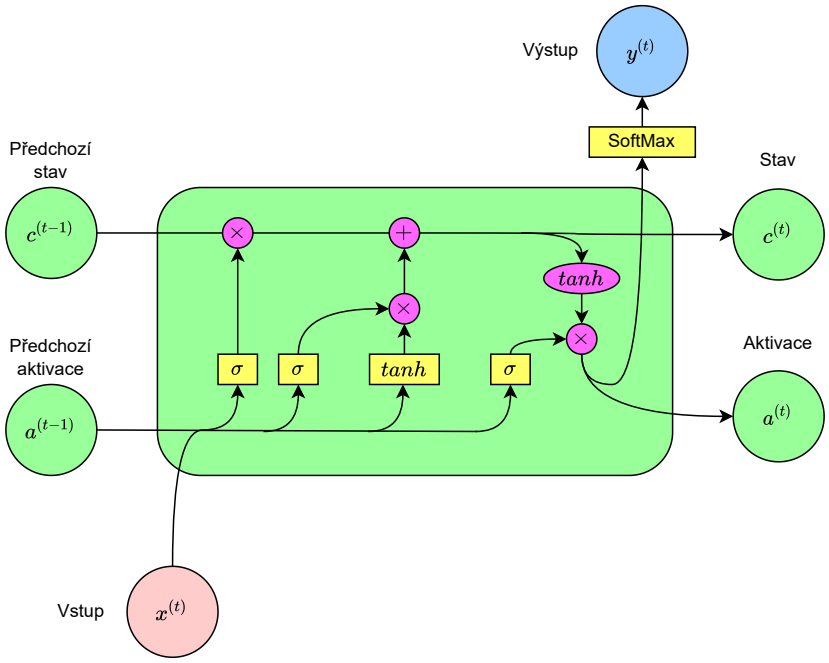
Obrázek 3.5: Příklad rekurentní neuronové sítě.

Principem rekurentních neuronových sítí je předání aktivací výstupní vrstvy

po průchodu jednoho vzorku na vstup neuronové sítě pro průchod dalšího vzorku touto sítí. Architektura tohoto druhu sítě vždy zakládá na jiném druhu neuronové sítě (např. vícevrstvý perceptron), který obohacuje o rekurentní krok. Model rekurentní neuronové sítě je znázorněn na obrázku 3.5.

3.2.6 Neuronové sítě Long Short-Term Memory

Long-Short Term Memory (LSTM) síť je druhem rekurentní neuronové sítě. Oproti již zmíněné tzv. klasické RNN je každý prvek této sítě, takzvaná buňka, rozšířena o jednotky vstupní, výstupní a zapomínací. Každá buňka má tedy stav, který je při každém průchodu pozměněn dle hodnot aktivace jednotlivých jednotek. Vstupní jednotka určuje, jakou mírou si má buňka zapamatovat současný vstup, a tedy jak obohatit svou paměť na základě vstupu a současného stavu. Výstupní jednotka určuje aktivaci buňky na základě vstupu a současného stavu. Zapomínací jednotka určuje, jak bude současný stav buňky ovlivňovat stav následující. Schéma takové neuronové sítě je znázorněno na obrázku 3.6



Obrázek 3.6: Schematické znázornění LSTM buňky. Převzato z [14].

Sítě typu LSTM tedy oproti klasickým RNN dovedou systematicky uchovávat v paměti stav ovlivněný nejen předchozím vstupem, ale celou sekvencí vstupů, které předcházely. Tato vlastnost se hodí například pro kontextové překlady textu, analýzu sentimentu a jiné úlohy, ve kterých je zapotřebí analyzovat větší kontext. Oproti RNN ale řeší zejména problém tzv. mizejícího gradientu, tedy jevu, při kterém metody učení založené na gradientním sestupu dojdou do stavu, kdy nemohou dále efektivně učit síť kvůli hodnotě gradientu blížící se nule. Při gradientním sestupu jsou váhy sítě pozměněny o násobek jednotlivých složek gradientu (parciálních derivací). Pokud je tento gradient dostatečně malý, kvůli reprezentaci čísel s plovoucí desetinnou tečkou může dojít k tomu, že jeho reprezentace splyne s nulou, a tedy nebude mít na korekci vah vliv. Tím dojde k zastavení učení sítě i přesto, že nebylo nalezeno minimum.

4 Učení hlubokých neuronových sítí

Pochopení principu učení neuronových sítí je základem k chápání dále navržených metod. V této kapitole budou podrobně vysvětleny dva algoritmy, které jsou dnes již standardem v oblasti učení neuronových sítí. Jedná se o algoritmy dopředného a zpětného šíření (forward propagation a backpropagation).

4.1 Dopředné šíření – forward propagation

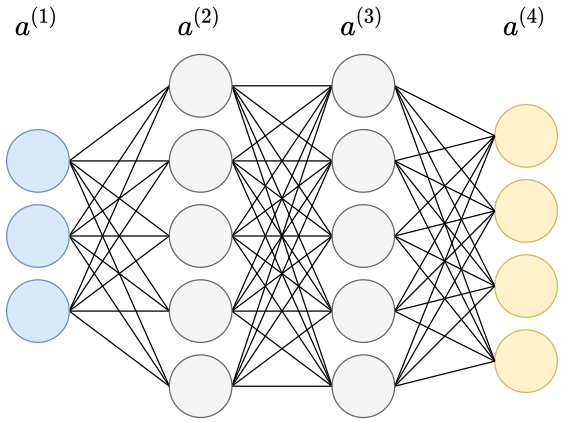
Metoda dopředného šíření je prvním krokem trénování neuronových sítí. S její pomocí dostaneme z daných vstupů konkrétní výstupy při aktuálním nastavení vah neuronové sítě. Tyto výstupy lze posléze porovnat s očekávaným výsledkem – odpovědí učitele. Jedná se v podstatě o dopředný průchod neuronovou sítí od vstupní k výstupní vrstvě.

Metoda dopředného šíření provede pro každý neuron skryté a výstupní vrstvy neuronové sítě dva kroky.

1. Předaktivace (z) – jedná se o vážený součet vstupů (aktivací neuronů předchozí vrstvy).

2. Aktivace (a) – vypočtená předaktivace je předána aktivační funkci. Aktivační funkce je matematická funkce, která do sítě vnáší nelinearitu, viz [35]. Aktivací vstupní vrstvy je samotný vstupní vektor.

Matematická podstata dopředného šíření je popsána rovnicemi 4.1 pro případ neuronové sítě z obrázku 4.1.



Obrázek 4.1: Ukázková neuronová síť pro znázornění dopředného šíření.

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \\
z^{(3)} &= \theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \\
z^{(4)} &= \theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\theta(x) = g(z^{(4)})
\end{aligned}
\tag{4.1}
$$

- $\mathbf{a^{(x)}}$ – vektor aktivací neuronů x-té vrstvy
- $\mathbf{z^{(x)}}$ – vektor předaktivací neuronů x-té vrstvy
- $\mathbf{x}$ – vstupní vektor
- $\boldsymbol{\theta^{(x)}}$ – váhy vstupující do x-té vrstvy
- $g(z^{(x)})$ – aktivační funkce neronů x-té vrstvy
- $h_\theta(x)$ – hypotéza

4.2 Zpětné šíření – backpropagation

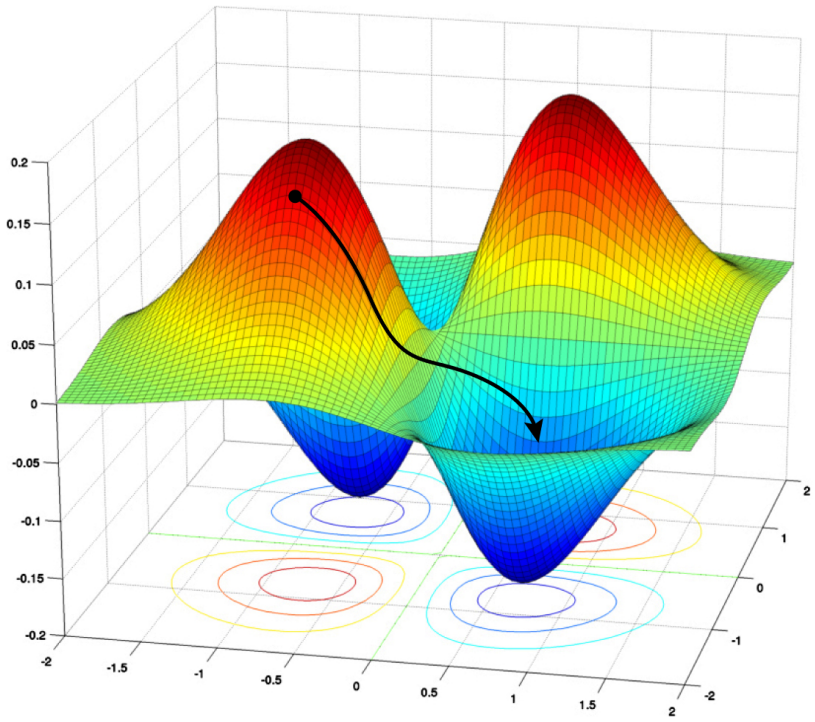
Podstatná většina neuronových sítí dnes ke svému trénování používá algoritmu zpětného šíření. Jedná se o matematický postup minimalizace cenové

funkce neuronové sítě za pomoci algoritmu gradientního sestupu. Algoritmem zpětného šíření se iterativně upravují váhy neuronové sítě, čímž se snaží zvýšit spolehlivost modelu.

V této podkapitole bude popsán algoritmus gradientního sestupu. Dále bude nastíněna cenová funkce neuronové sítě a bude popsán samotný algoritmus zpětného šíření.

4.2.1 Gradientní sestup

Gradientním sestupem se rozumí iterativní optimalizační algoritmus používaný k hledání lokalního minima (respektive maxima) dané funkce. Aby mohl algoritmus gradientního sestupu fungovat, musí být funkce diferencovatelná. Příklad gradientního sestupu pro konkrétní funkci je zobrazen na obrázku 4.2.



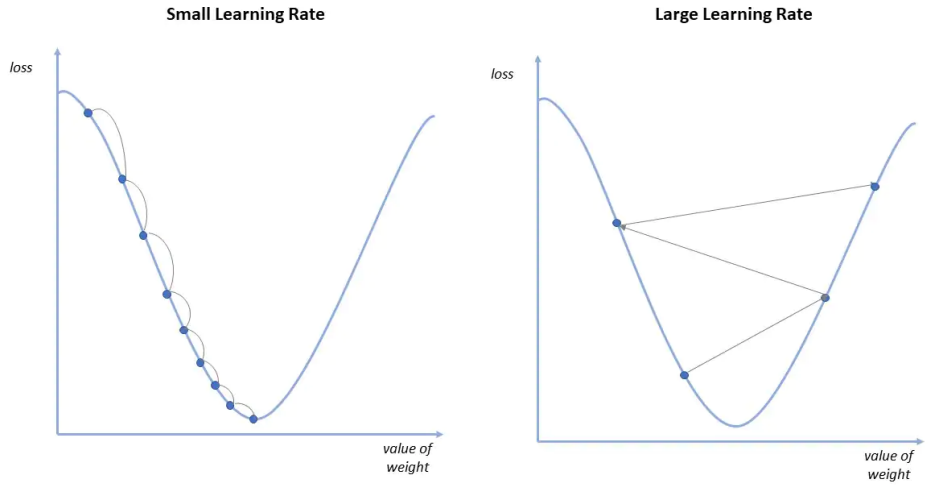
Obrázek 4.2: Příklad vizualizace gradientního sestupu. Převzato z [2].

Algoritmus gradientního sestupu začíná volbou výchozího bodu. Tento krok rozhodne o tom, ve kterém z lokálních minim algoritmus skončí. Z tohoto bodu pak algoritmus sestupuje zvoleným krokem ve směru proti gradientu. Gradientem se rozumí směr největšího růstu.

Matematicky pak lze zapsat jako:

$$\nabla f(x_1, x_2, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \ldots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

Zmíněný krok je třeba zvolit před zahájením algoritmu tak, aby algoritmus konvergoval k některému z lokálních minim. Při zvolení příliš malého kroku bude algoritmus konvergovat moc pomalu, naopak při zvolení příliš velkého kroku může algoritmus oscilovat, případně divergovat, viz obrázek 4.3.



Obrázek 4.3: Gradientní sestup s malým a velkým krokem. Převzato z [19].

Potenciálně nežádoucí vlastností gradientního sestupu může být neschopnost nalezení globálního minima. Pro konvexní funkce nalezne tento algoritmus globální minimum vždy, avšak u konkávních funkcí je nalezení globálního minima závislé na správné volbě výchozího bodu. Ve většině případů však takovýto výchozí bod nelze s jistotou určit.

4.2.2 Cenová funkce

Pro výpočet chyby je nejprve potřeba určit předpis cenové funkce neuronové sítě. Ta je složena z chybové funkce logistické regrese (v případě logistické

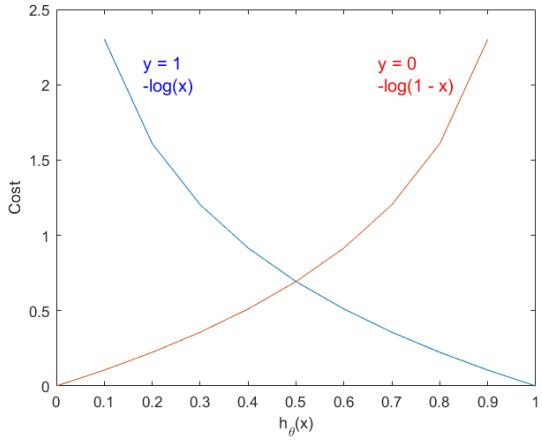
sigmoidy jako aktivační funkce), avšak bude zde vysvětlena tak, aby bylo možné ji pochopit i bez předchozí znalosti logistické regrese. Cenová funkce neuronové sítě je popsána rovnicí 4.2.

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - (h_\theta(x^{(i)}))_k)] \quad (4.2)$$

- θ = vektor vah neuronové sítě
- m = počet vzorků trénovací množiny
- K = počet neuronů výstupní vrstvy (počet tříd)
- $y_k^{(i)}$ = odpověď učitele
- $h_\Theta(x^{(i)})$ = hypotéza

Levá strana rovnice říká, že se jedná o celkovou chybu neuronové sítě, pro konkrétní nastavení vah θ. Pravá strana pak říká, že chybu počítáme přes všechny vzorky trénovací množiny a přes všechny neurony sítě, přičemž každý neuron provádí logistickou regresi.

Logistická regrese je klasifikační algoritmus, který slouží ke klasifikaci do dvou tříd, 0 a 1. K tomu využívá, ve většině případů, jako funkci hypotézy sigmoidu, neboli $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$. Cenová funkce logistické regrese je pak logaritmická a její graf lze vidět na obrázku 4.4. Dle odpovědi učitele (0 nebo 1) se pak jeden z jejích členů vynuluje.



Obrázek 4.4: Graf cenové funkce logistické regrese.

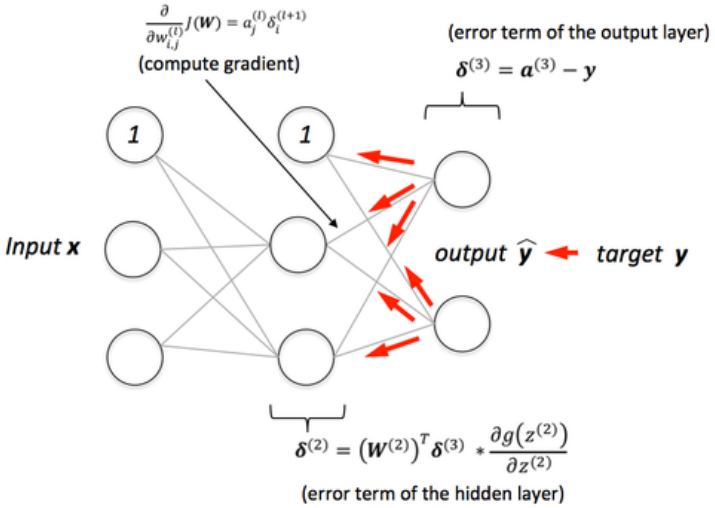
Výsledkem cenové funkce je skalár, který udává souhrnnou chybu neuronové sítě pro konkrétní nastavení vah. Snahou algoritmu zpětného šíření je tedy tuto funkci minimalizovat.

4.2.3 Algoritmus zpětného šíření

Samotný algoritmus spočívá ve výpočtu parciálních derivací cenové funkce podle všech vah v jednotlivých vrstvách sítě. Výpočty v jednotlivých vrstvách jsou prováděny následovně:

1. Pro **poslední vrstvu** platí, že chyba každého jejího neuronu je vypočtena z rozdílu aktivace neuronu po průchodu sítí dopředným šířením a odpovědí učitele v podobě tzv. one-hot vektoru (viz 3.1). Výpočet chyby j-tého neuronu v poslední (n-té) vrstvě je tedy $\delta_j^n = (h_\Theta(x))_j) - y_j$.
2. Pro všechny **skryté vrstvy** je pak chyba vypočtena jako součin chyby předchozí vrstvy, vah vstupujících do dané vrstvy a derivace aktivační funkce neuronů předchozí vrstvy. Matematicky pak daná chyba vypadá následovně: $\delta^{(L-1)} = (\Theta^{(L-1)})^T \delta^{(L)} \cdot \frac{\partial g(z^{(L-1)})}{\partial z^{(L-1)}}$.
3. Pro **vstupní vrstvu** nelze chybový vektor vypočítat, neboť aktivací vstupní vrstvy je vstupní vektor.

Algoritmus je pro lepší přehlednost znázorněn schématem 4.5.

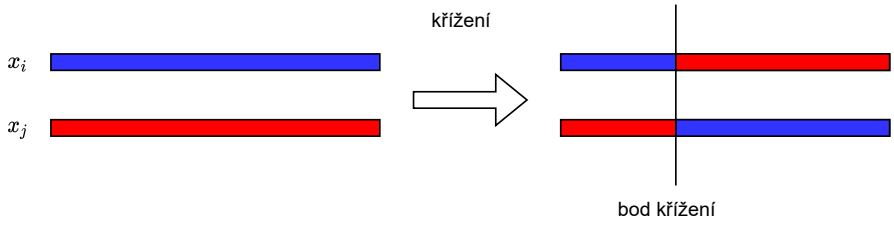


Obrázek 4.5: Schéma algoritmu backpropagation. Převzato z [32].

4.3 Genetické algoritmy

Genetický algoritmus je stochastická optimalizační metoda, která se zakládá na teorii evoluce [4]. Algoritmus pracuje s tzv. populací, tedy polem kandidátních řešení (tzv. chromozomů), z nichž každé nese svou tzv. genetickou informaci pevné délky. Každý prvek populace je ohodnocen pomocí tzv. fitness funkce (funkce vhodnosti), tedy takové funkce, která přiřazuje prvku populace reálné číslo dle toho, jak dobře řeší daný problém. Čím nižší je toto číslo, tím lepší daný prvek je. Analogií v teorii evoluce je princip přirozeného výběru, který silnější členy populace upřednostňuje před slabšími. V případě živých organismů jde o geneticky dané vlastnosti, které jedince činí vhodnějším pro další reprodukci, v případě genetických algoritmů je to schopnost řešit problém dle vhodně vybrané metriky.

Algoritmus pracuje v iteracích zvaných generace. Každá generace sestává z několika fází. První je křížení, při kterém je s určitou pravděpodobností (*crossover rate*, CR) zvolena n-tice prvků populace, které si mezi sebou vymění část své genetické informace. Algoritmus náhodně vybere místo, ve kterém ke křížení dojde, a poté část genetické informace za tímto místem mezi vybranými prvky vymění (znázorněno na obrázku 4.6). Druhou fází je mutace. V této fázi algoritmus s určitou pravděpodobností nahradí malou část genetické informace prvku jinou, např. náhodně vygenerovanou. Třetí fází je ohodnocení všech prvků populace již zmíněnou funkcí vhodnosti.



Obrázek 4.6: Znázornění klasického křížení dvou prvků populace x_i a x_j.

Obvykle je základní forma genetického algoritmu ještě obohacena o další kroky. Typickou technikou je tzv. elitářství, při které algoritmus uchovává nejlepší kandidátní řešení, a v případě, že vlivem křížení nebo mutace dojde ke zhoršení nejlepšího člena aktuální populace, je na konci iterace tento uchovaný prvek vrácen zpět namísto jiného prvku s horším ohodnocením.

Genetický algoritmus je standardně parametrizován velikostí populace, po-

čtem generací, pravděpodobností křížení a pravděpodobností mutace. Tyto parametry je nutné nastavit tak, aby byla populace dostatečně velká a dostatečně rozmanitá. Právě rozmanitost populace je klíčovou vlastností pro úspěšnou exploraci, tj. prohledávání prostoru za účelem konvergence k nejlepšímu řešení [18].

Oproti algoritmu zpětného šíření, který je založen na existenci gradientu, je genetický algoritmus na gradientu nezávislý [4]. To může být potenciální výhodou, pokud cenová funkce řešeného problému není spojitě diferencovatelá. Stejně jako algoritmus zpětného šíření však konvergence genetického algoritmu závisí na počátečním stavu, resp. celé počáteční populaci. Ta obvykle bývá generována náhodně.

Hypotéza stavebních bloků (Building Blocks Hypothesis, BBH) je jednou z důležitých hypotéz, která se pokouší vysvětlit schopnost genetických algoritmů řešit komplexní problémy [13]. Tato hypotéza říká, že genetické algoritmy fungují díky tomu, že napříč generacemi propagují tzv. stavební bloky, tedy části genetické informace, které lze hypoteticky optimalizovat samostatně. V přeneseném smyslu pak lze hypotézu přeformulovat tak, že genetické algoritmy jsou schopné dobře řešit takové problémy, které lze efektivně rozdělit na menší, samostatně řešitelné podproblémy.

Genetické algoritmy trpí dvěma hlavními problémy. Prvním je takzvaná předčasná konvergence. To je jev, při kterém populace v prvních krocích přijde o svou rozmanitost a ustrne v lokálním minimu funkce vhodnosti. Druhým problémem je stagnace. Ta se projevuje tak, že prvky populace jsou sice dostatečně rozmanité, ale žádný z kroků genetického algoritmu není schopen ze současné populace vytvořit prvek lepší.

4.3.1 Diferenciální evoluce

Jednou z variant genetického algoritmu je diferenciální evoluce (DE). Genetický algoritmus je navíc parametrizován diferenciálním parametrem F, který je použit jako faktor pro diferenciální křížení. Každá generace tedy navíc kromě klasického křížení (nebo místo něj) provádí ještě diferenciální křížení některým z možných způsobů, obvykle označovaných zkratkami. Diferenciální křížení pro prvek x_i populace P může mít například tyto podoby [15, 39]:

- `DE/rand/1` – vybírá náhodně jeden výchozí prvek populace, který po-

užije jako základ pro diferenciální křížení s daným prvkem populace

$$v_i = x_{r1} + F \cdot (x_{r2} - x_{r3}), \tag{4.3}$$

kde v_i značí aktuání prvek, $r1$, $r2$ a $r3$ značí náhodná čísla označující indexy v populaci; tato čísla jsou navzájem různá;

- `DE/best/1` – základem bude nejlepší prvek populace, popř. elitní prvek za předpokladu použití techniky elitářství

$$v_i = e + F \cdot (x_{r1} - x_{r2}), \tag{4.4}$$

kde v_i značí aktuání prvek, e značí elitní prvek (může, ale nemusí být aktuálně přítomen v populaci), $r1$ a $r2$ náhodná čísla označující indexy v populaci; tato čísla jsou navzájem různá;

- `DE/current-to-best/1` – základem je vždy daný prvek populace, který je diferenciálně křížen s nejlepším (elitním) prvkem

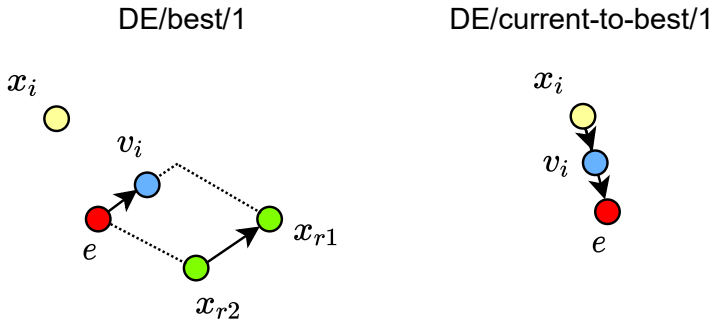
$$v_i = x_i + F \cdot (e - x_i), \tag{4.5}$$

kde v_i značí aktuání prvek, x_i téže prvek v minulé generaci, e značí elitní prvek;

- `DE/current-to-rand/1` – totéž, ale s náhodně vybraným prvkem

$$v_i = x_i + F \cdot (x_{r1} - x_i), \tag{4.6}$$

kde v_i značí aktuání prvek, x_i téže prvek v minulé generaci a $r1$ značí náhodný index v populaci.



Obrázek 4.7: Vizualizace diferenciálního křížení s parametrem $F = 0.5$ způsobem `DE/best/1` a `DE/current-to-best/1`

Hlavní výhodou diferenciální evoluce je její efektivita a jednoduchost. Oproti klasickému křížení má diferenciální křížení tu vlastnost, že změny v genetické informaci jsou systematičtější a ve své podstatě aproximují pohyb ve spojitém prostoru [4]. Diferenciální evoluce je vhodná pouze pro řešení problémů, jejichž genetická informace je kódována reálnými čísly, a tedy jde o spojité veličiny.

4.3.2 Další modifikace

Mezi často používané varianty genetických algoritmů, resp. diferenciální evoluce patří například algoritmus JADE [44], který zavádí strategii diferenciálního křížení označovanou jako `DE/current-to-pbest/1`. Ta pro diferenciální křížení neuvažuje pouze nejlepší prvek, ale p nejlepších prvků. To dovoluje částečně zamezit předčasné konvergenci, jelikož není zaručeno, že jediný nejlepší prvek v prvních generacích je nejblíže skutečnému (globálnímu) minimu funkce vhodnosti.

Dále je často diferenciální evoluce doplněna o formu adaptivity diferenciálního parametru. Takovou metodou je například metoda SADE (Self-Adapting DE) [45], která upravuje diferenciální parametr a pravděpodobnost křížení na základě předchozí úspěšnosti diferenciálního křížení. Parametry F a CR jsou součástí každého prvku populace, a v každé generaci jsou vygenerovány znovu. Parametr F je vygenerován s Cauchyho rozdělením se střední hodnotou μ_F a rozptylem $0,1$, parametr CR je vygenerován s normálním rozdělením se střední hodnotou μ_{CR} a rozptylem $0,1$. Střední hodnoty jsou na konci každé generace upraveny tak, aby reflektovaly průměrnou hodnotu odpovídajících parametrů těch prvků, které přežily do další generace. Nejprve je vypočtena střední hodnota s_{CR} jako aritmetický průměr

$$s_{CR} = \frac{\sum_{i=0}^{N_s} CR_i}{N_s} \tag{4.7}$$

a střední hodnota s_F jako Lehmerův průměr [5]

$$s_F = \frac{\sum_{i=0}^{N_s} F_i^2}{\sum_{i=0}^{N_s} F_i}, \tag{4.8}$$

kde N_s je počet přeživších prvků, i je index přeživšího prvku a F_i (resp. CR_i) je diferenciální parametr F (resp. CR) daného prvku s indexem i. Přeživší prvek je takový, jehož hodnota funkce vhodnosti je v této generaci lepší, než byla v generaci předchozí.

Poté je nová střední hodnota pravděpodobnostních rozdělení stanovena podle následující rovnice:

$$\mu = (1 - c) \cdot \mu + c \cdot s, \tag{4.9}$$

kde μ je příslušná aktuální střední hodnota rozdělení (μ_F a μ_{CR}), s je spočtená střední hodnota z populace (s_F a s_{CR}) a c je konstanta adaptability z intervalu $(0; 1\rangle$.

Další možností je kombinace více způsobů diferenciálního křížení pro každý prvek, kdy je do výsledné populace vybrán právě jeden prvek s nejlepší výslednou hodnotou funkce vhodnosti. Ustálenou variantou tohoto způsobu je metoda CODE (Composite DE) [43]. Ta použije strategie `DE/rand/1`, `DE/rand/2` a `DE/current-to-rand/1` s vybranými, empiricky určenými parametry F a CR. Dohromady generuje 3 potenciální prvky populace, z nichž vybere nejlepší.

4.3.3 Učení neuronových sítí genetickými algoritmy

V případě neuronových sítí představuje genetickou informaci jednoho kandidátního řešení sada vah všech synaptických spojení. Fitness funkce musí být zvolena tak, aby dostatečně dobře vyhodnocovala míru úspěšnosti klasifikace (popř. regrese). Takovou funkcí může být například průměrná kvadratická odchylka, nebo jiná, standardně používaná metrika.

Za předpokladu takového kódování vah synaptických spojení lze genetický algoritmus použít. Problémem ale může být počet vah, tedy dimenze problému. Spolu s faktem, že jde o neohraničený problém co se týče hodnoty jednotlivých vah, může být genetický algoritmus bez dodatečných úprav neefektivní a konvergence značně pomalá.

Genetické algoritmy jsou často používány i pro jiné úkoly související s neuronovými sítěmi. Vybrané studie pojednávají například o optimalizaci struktury neuronové sítě pomocí genetického algoritmu [21].

5 Existující alternativy k backpropagation

Algoritmus zpětného šíření, jak již bylo zmíněno, s sebou přináší určitá úskalí. Není tak vyjímkou, že vznikají publikace, ve kterých se jejich autoři snaží vymyslet algoritmus učení neuronových sítí, který by problémy zpětného šíření nepřinášel, a zároveň se s ním mohl svou rychlostí i přesností měřit. V této kapitole bude několik takových existujících algoritmů stručně popsáno.

5.1 Propagace rozdílových cílů [2015]

Pro neuronové sítě obsahující tzv. autoenkodéry byla navržena metoda propagace rozdílových cílů (*difference target propagation*) [24]. Autoenkodér je entita, která sestává z enkodéru a dekodéru. Enkodér je vrstva nebo soustava vrstev, která převádí vstupní vektor do kódu. Dekodér je vrstva nebo soustava vrstev, která převádí kód do výstupního vektoru identické délky jako je původní vstupní vektor. Entity tohoto typu se hodí například pro odstranění šumu nebo sjednocení drobných odlišností ve vstupních datech patřících do jedné třídy.

Hlavní myšlenkou rozdílových cílů je lineární korekce vah autoenkodérů. Tyto cíle jsou, stejně jako gradienty v algoritmu zpětného šíření, propagovány zpětně, tj. od poslední vrstvy k první. Narozdíl od zpětného šíření tyto změny nezávisí na diferencovatelnosti problému, a tedy tuto metodu lze použít i na problémy, na které se zpětné šíření nehodí. Rovněž lze tuto metodu použít i pro stochastické neuronové sítě. Původní studie uvádí, že pro takové sítě a problémy jde o efektivní alternativu [24].

5.2 Oddělená neuronová rozhraní s využitím syntetických gradientů [2017]

Další částečnou alternativou k algoritmu zpětného šíření je učení pomocí tzv. syntetických gradientů [22]. Tato metoda byla navržena zejména pro možnost učit neuronové sítě v distribuovaném prostředí. Umožňuje totiž učit jednotlivé vrstvy samostatně, zatímco ve zbytku vrstev stále ještě probíhá

dopředné šíření. Toto učení je možné díky tomu, že každá vrstva obsahuje model syntetických gradientů, které gradient odhadují už v momentě dopředného šíření. Zpětné šíření ale stále probíhá, jen je použito pro úpravu parametrů modelu syntetických gradientů.

Tento typ učení je možné použít pro takřka libovolnou architekturu neuronové sítě. Jeho účel však není být přesnější, než zpětné šíření, ale umožnit efektivně paralelizovat algoritmy dopředného a zpětného šíření s minimální ztrátou přesnosti a schopnosti konvergence. Dle výsledků uvedených v odkazované studii došlo k výraznému zrychlení učení neuronových sítí velkých rozměrů [22].

5.3 Trénování neuronových sítí pomocí lokálních chybových signálů [2019]

Metoda popsaná v této studii navrhuje učit jednotlivé skryté vrstvy pouze lokálně, za pomocí tzv. lokálních chybových signálů [30]. Tyto signály jsou vygenerovány v jednovrstvé neuronové síti, která je přidružena každé skryté vrstvě. Tato jednovrstvá síť je použita ke stanovení hned dvou chybových vektorů, z nichž je každý spočten na základě jiné metriky. Výsledný chybový vektor, který je použit k úpravě vah dané vrstvy, je výsledkem agregace těchto dvou vektorů.

Problémem však je, jak získat chybový vektor bez znalosti odpovědi učitele lokálně ve vrstvě, ke které se za použití běžného zpětného šíření dostane chybový vektor postupně z vrstvy poslední. Odpověď učitele je pro potřeby lokálních chyb transformována pomocí různých metod tak, aby vektor této aproximace odpovědi měl stejnou velikost, jako je počet neuronů dané vrstvy, a potažmo vrstvy lokální neuronové sítě.

Experimenty ukázaly, že výsledky této metody jsou srovnatelné s běžným algoritmem zpětného šíření pro klasifikaci obrazových informací. Tato metoda rovněž odstraňuje problém závislosti vrstev při zpětném šíření, a tedy je možné ji použít pro učení v distribuovaném nebo heterogenním prostředí.

5.4 Genetické algoritmy

Nejdiskutovanější alternativou k algoritmu zpětného šíření jsou bez pochyby genetické algoritmy. Problémem trénování neuronových sítí pomocí genetic-

kých algoritmů se zabývá celá řada publikací. Jejich závěry se však často rozchází.

Například Gupta et al. ve svém článku *Comparing backpropagation with a genetic algorithm for neural network training* [16] konstatují, že genetický algoritmus je efektivnější, účinnější a snažší k použití než algoritmus zpětného šíření.

Výsledky jiných publikací však zase ukazují, že genetické algoritmy nejsou vždy tím lepším řešením. Například Lee et al. ve svém článku *Optimising neural network weights using genetic algorithms: a case study* [25] poukazují na nižší výkonnost genetického algoritmu oproti standardnímu algoritmu zpětného šíření, a to hlavně pro větší neuronové sítě.

Popis fungování genetických algoritmů je k nalezení v sekci 4.3.

5.5 Další metody

Mezi další metody patří například Hilbert-Schmidtovo kritérium nezávislosti [26]. To se snaží urychlit a zlepšit učení neuronových sítí s minimální ztrátou přesnosti. Pro učení je použita aproximace tzv. *informačního bottlenecku*, což minimalizuje závislost vnitřních reprezentací na skutečných vstupech.

Další potenciálně úspěšnou metodou je metoda založená na pomocných proměnných [7]. Metoda učí neuronovou síť pomocí sady tzv. *mikro dávek*, tedy malých podmnožin trénovací množiny. Jako již uvedené metody, tak i tato se snaží odbourat závislost vrstev při zpětném šíření, ovšem tentokrát použitím pomocné proměnné, která odpovídá hodnotě aktivace neuronů dané vrstvy. Tato metoda řeší problém mizejícího gradientu a rovněž ji lze použít na problémy, které nejsou diferencovatelné.

Problémy zpětného šíření jsou známé a popsané, což vyúsťuje ve vývoj metod jiných, které se tyto problémy pokouší odbourat. Tyto pokusy jsou však často vázány na konkrétní druh neuronové sítě nebo konkrétní druh problémů.

6 Navržené metody

Metoda zpětného šíření je jednou z nejpoužívanějších metod učení neuronových sítí, avšak přináší s sebou řadu problémů. Prvním z těchto problémů je schopnost učení vycházející z počátečního nastavení vah neuronové sítě. Standardně se tyto váhy před započetím učení nastavují na náhodné hodnoty. Algoritmus zpětného šíření pak z tohoto výchozího bodu sestupuje ve směru lokálního gradientu do nejbližšího minima. Počáteční nastavení vah má tak přímý vliv na konvergenci ke konkrétnímu lokálnímu minimu. Není však zárukou, že toto nalezené minimum bude zároveň globálním minimem.

Dalším problémem metody zpětného šíření je neurčitost rychlosti konvergence. Tuto rychlost konvergence lze pro konkrétní počáteční nastavení vah optimalizovat pomocí úpravy parametru velikosti kroku. Z důvodu rozmanitosti směru gradientu v různých místech cenové funkce je třeba při jiném počátečním nastavení vah tento parametr velikosti kroku znovu upravit tak, aby metoda zpětného šíření stabilně a dostatečně rychle konvergovala.

Jedním z výrazných problémů algoritmů zpětného šíření je to, že rychlost konvergence bývá pomalá, a tak je potenciálně provedeno spousty kroků algoritmu jen pro malou změnu chyby. Navíc je každý krok relativně výpočetně náročný vzhledem k nutnosti vypočítat gradient nad velkou datovou množinou a každou vrstvou neuronové sítě. Pro rozlehlé architektury sítí a velké datové sady toto může být problém.

Jedním z dalších nedostatků algoritmu zpětného šíření je velice omezená schopnost neuronovou síť postupně učit. Jakmile je neuronová síť natrénovaná pomocí jedné datové množiny, nelze efektivně síť dotrénovat dalšími vzorky, aniž by byla k dispozici i původní datová množina. Datová množina definuje podobu cenové funkce, která může po změně této množiny kompletně změnit podobu. Další učení bez použití celé datové sady tedy může naopak degradovat schopnost sítě klasifikovat příznakové vektory, které by s použitím původní trénovací datové sady klasifikovala správně.

Následující podkapitoly se zabývají návrhem metod, které si dávají za cíl alespoň některé z těchto problémů řešit. Celkem byly navrženy 4 metody.

6.1 Metoda ReiStein Blitz

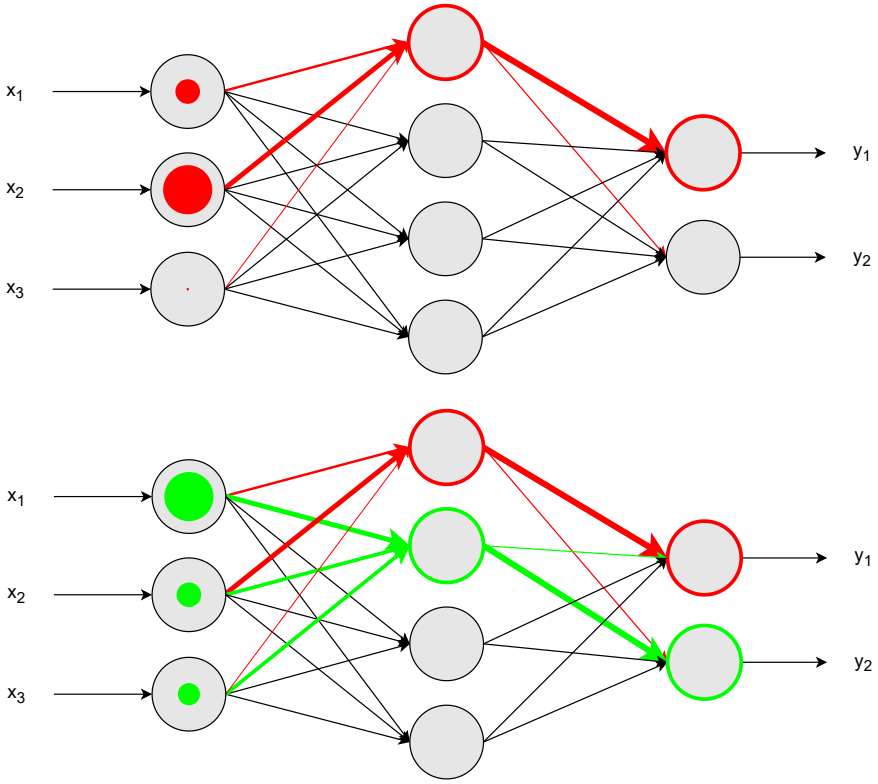
První z navržených metod vychází z principu vývoje reflexního oblouku centrální nervové soustavy. Centrální nervová soustava při svém vzniku produkuje řádově vyšší počet neuronů, než je ve výsledku použit pro výsledný účel. Při jejím vývoji dochází postupně k tzv. buněčné smrti, při které nadbytečné neurony postupně umírají [29]. Tím se částečně inspiruje tato metoda.

Metoda nejprve z trénovací množiny vybere reprezentanta každé klasifikační třídy. Poté pro každého z nich vybere právě jeden neuron skryté vrstvy. Tomuto neuronu nastaví všechny vstupní váhy na hodnoty odpovídající prvkům vektoru vybraného reprezentanta. Následně nastaví všechny výstupní váhy z tohoto skrytého neuronu na 0, kromě váhy synapse vedoucí do výstupního neuronu odpovídající dané třídě, kterou nastaví na 1. Tento postup je opakován pro všechny reprezentanty vybrané v prvním kroku. Veškeré váhy, které nebyly nastaveny tímto postupem, jsou inicializovány na dostatečně malé náhodné číslo. Tím je umožneno navazující učení sítě s účastí odpovídajících synaptických cest, a zároveň je minimalizována pravděpodobnost, že by takto vznikla aktivace náhodných neuronů výstupní vrstvy číselně větší, než aktivace vybudovaná v první fázi této metody.

Na obrázku 6.2 je znázorněno fungování metody na neuronové síti, klasifikující do dvou tříd. Červenou barvou jsou označeny váhy a neurony ovlivněné reprezentantem první třídy, zeleně pak reprezentantem druhé třídy. Váhy, které zůstávají vyznačené černou barvou, jsou po skončení průchodu obou reprezentantů sítí nastaveny na náhodná malá čísla.

Z výše uvedeného popisu je zřejmé, že metoda vyžaduje, aby skrytá vrstva obsahovala alespoň stejný počet neuronů, jako vrstva výstupní. Je však žádoucí, aby neuronů skryté vrstvy bylo více.

Tato metoda sama o sobě nemůže poskytnout dostatečnou míru učení. Z podstaty jejího fungování nastavuje váhy pouze užitím jednoho reprezentanta z každé třídy, což pochopitelně nereflektuje možnou variabilitu vstupních dat. Může ale poskytnout dostatečně dobré počáteční nastavení pro další algoritmus učení.



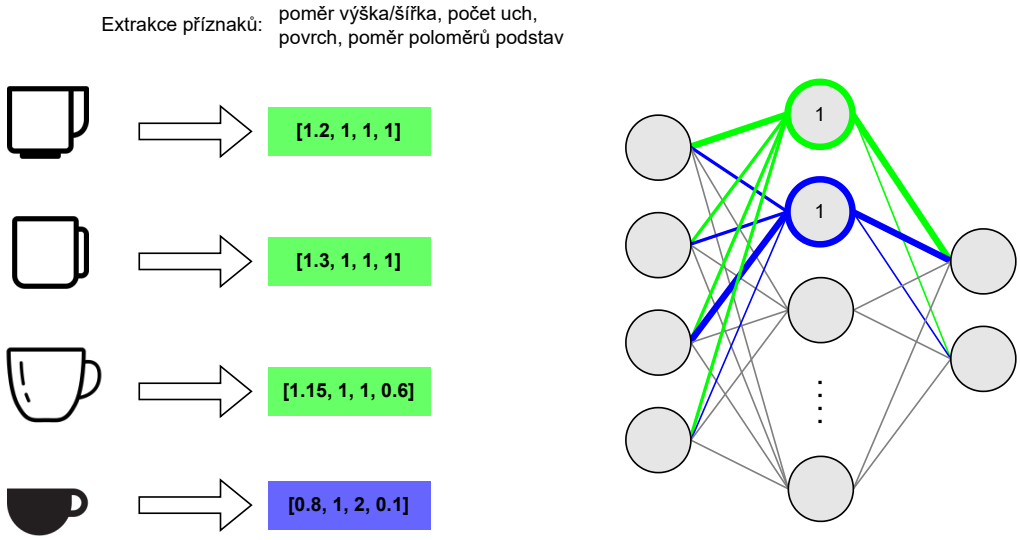
Obrázek 6.1: Znázornění fungování metody ReiStein Blitz. Tloušťka čáry odpovídá číselné hodnotě dané váhy.

6.2 Metoda Freely Growing

Metoda Freely Growing je založena na principu fungování vytváření synapsí v centrální nervové soustavě. Tento princip lze vysvětlit například na tom, jak se lidský mozek učí rozpoznávat běžné každodenní předměty na základě vizuální informace. Vezmeme-li například množinu čajových hrnků a množinu hrnců, a budeme se snažit naučit člověka rozpoznávat, který předmět patří do které skupiny. Komplexní nervová soustava jako je ta lidská má tendenci relativně automaticky extrahovat z vizuální informace příznaky, a tedy v konečném důsledku redukuje celou vizuální informaci na souhrn příznaků. Na příkladu výše zmíněných předmětů to může být například výška, poměr výšky a šířky, počet uch, povrchová úprava a podobné vlastnosti, které lze z vizuální informace extrahovat. Pokud budeme postupně předkládat předměty subjektu a poskytovat příslušnou doprovodnou informaci, o jaký předmět se jedná, jeho centrální nervová soustava si nejprve připodobní nové informace k tomu, co si již v minulosti zapamatovala, a pokud jsou tyto informace dostatečně podobné, pouze se ujistí v tom, že jsou dané vzory v podobě navázaných synaptických spojení správné – dojde tedy k dalšímu

posílení těchto spojení. Pokud však dostatečně podobné nejsou, je nutné se začít učit vzor, který doposud nebyl v předkládaných předmětech (a odpovídajících vizuálních informacích, respektive extrahovaných příznacích) zachycen.

Během učení se tedy postupně vytvářejí synaptické dráhy pro doposud neznámé vzory, popřípadě se posilují synaptické dráhy vzorů, které už byly spatřeny. Postupně tedy množina neuronů účastnící se rozpoznávání předmětů této domény roste s rozmanitostí vzorů ve vstupních datech. Nutno dodat, že tato metoda se nezabývá extrakcí příznaků, pouze procesem učení, který následuje s již extrahovanými příznaky.



Obrázek 6.2: Znázornění fungování metody Freely Growing na rozpoznávání čajových hrnků dle vybraných příznaků. Demonstrováno na prvcích jedné třídy, princip fungování pro více tříd je analogický.

Metoda postupně vybírá všechny prvky všech klasifikačních tříd. Pro každý prvek ověří, zda již byl dané třídě přiřazen nějaký neuron skryté vrstvy. Pokud nebyl, vybere doposud nepřiřazený a nastaví všechny vstupní váhy na čísla odpovídající prvkům vstupního vektoru, identicky jako v metodě Blitz. Pokud již byl nějaký neuron přiřazen, ověří postupně u všech již přiřazených neuronů dané třídě, zda jsou váhy dostatečně podobné jako prvky vstupního vektoru. První neuron, který této podmínce vyhovuje je vybrán, a jeho váhy jsou rozšířeny o tento daný prvek tak, že je s jeho vstupním vektorem

proveden vážený aritmetický průměr dle následujícího vzorce:

$$w_i = \frac{w_i * n + x_i}{n + 1}, \tag{6.1}$$

kde w_i je váha neuronu na indexu i, x_i je prvek vstupního vektoru na indexu i a n je počet prvků, které se podílejí na vahách daného neuronu. Pokud však žádný neuron nemá váhy dostatečně podobné, přiřadí se neuron další, doposud nepřiřazený. Pokud není k dispozici žádný nepřiřazený neuron, prvek se nepoužije.

Podobně jako v metodě Blitz, i tato metoda nastavuje váhy nevyužitých neuronů na dostatečně malá náhodná čísla. Tato čísla musí být malá, aby jejich příspěvek v aktivaci příslušných neuronů nepřevýšil aktivace vzniklé rozpoznanými vzory.

Tato metoda může být podobná metodě Blitz za předpokladu, že jsou prvky v rámci tříd všechny navzájem dostatečně podobné. Oproti metodě Blitz však pracuje nad všemi prvky trénovací množiny, a tak umožňuje síť naučit více vzorů, které se v dané třídě vyskytují.

6.3 Diferenciální evoluce

Tato metoda učení ve svém základu odpovídá klasické diferenciální evoluci. Navíc oproti standardní diferenciální evoluci obsahuje vybrané mechanismy pro zlepšení konvergence.

Prvním z nich je generování počáteční populace. Obvyklé způsoby inicializují počáteční populaci náhodně. To je poměrně univerzální způsob, kterým lze obvykle dosáhnout dobrých výsledků, ovšem konvergence pak může být značne pomalá vzhledem k dimenzi problému a tomu, že žádné náhodně vygenerované řešení velice pravděpodobně nebude nikde blízko globálnímu minimu. Tato modifikace nageneruje pouze část populace náhodně. Zbylou populaci inicializuje pomocí metody Blitz a Freely Growing, konkrétně ve variantách, ve kterých jsou nevyužité váhy nastaveny náhodně, a tedy je zaručena alespoň základní míra rozmanitosti. Tento způsob generování staví na hypotéze stavebních bloků, jelikož klasifikaci prvků každé třídy, respektive rozpoznávání patřičných vzorů lze chápat jako samostatný podproblém. Metody Blitz a Freely Growing pak slouží ke generování těchto stavebních kamenů.

Druhou modifikací je způsob, jakým je vybrán segment pro křížení. Neuronová síť obvykle v jednotlivých vrstvách, jejich neuronech a vahách konzervuje určité vzory, které ve vstupních vzorcích nachází během učení. Ekvivalentní neuronové sítě však mohou mít vybrané neurony a jejich váhy v dané vrstvě pouze libovolně permutovány, aniž by se tyto sítě lišily ve schopnosti řešit daný problém. Prosté křížení takovýchto ekvivalentních neuronových sítí by pak způsobilo nahrazení jednoho rozpoznaného vzoru jiným, který už je ale reprezentován neuronem jiným. Křížení by pak nebylo efektivní a prvky populace by takto přicházely o rozpoznané vzory. Namísto toho je pro křížení vybrán neuron a jeho váhy tak, aby u všech prvků účastnících se křížení byla kvadratická odchylka minimální. Tento způsob křížení by měl rozpoznávané vzory postupně zpřesňovat, aniž by došlo k zapomenutí vzorů jiných.

Další modifikací je přidání strategie křížení, která kromě jednoho neuronu a odpovídajících vstupních vah zahrnuje i váhy výstupní. Tato strategie by mohla vylepšit konvergenci v případě, že je vzor správně rozpoznán, ale nesprávně klasifikován.

Poslední modifikací je způsob, jakým je diferenciální evoluce řízena. Ke každému prvku populace kromě vektoru vah všech synaptických spojení je navíc přidán diferenciální parametr a způsob diferenciálního křížení. Tyto parametry jsou generovány v průběhu evoluce a jsou přejímány hůře se vyvíjejícími prvky populace. Tato změna by měla zlepšit schopnost adaptace algoritmu na různé druhy problémů.

6.4 Diferenciální evoluce se zpětným šířením

Další, příbuznou metodou učení je kombinace výše uvedené metody s metodou zpětného šíření. Jedním z hlavních problémů metody zpětného šíření je nalezení vhodného parametru míry učení, a potažmo i parametru setrvačnosti tak, aby bylo učení dostatečně rychlé, ale aby metoda nedivergovala. Nalezení optimálních parametrů může být obtížné i vzhledem k zastavovací podmínce, která je obvykle uvolněna, aby dovolila pomalejší konvergenci.

Tato metoda přidává metodu zpětného šíření jako jednu z fází algoritmu. Kromě křížení a mutace tedy může dojít s určitou pravděpodobností i k několika iteracím zpětného šíření. Za předpokladu, že jsou správně vybrány

parametry míry učení a setrvačnosti, by tato fáze měla kandidátní řešení vždy zlepšit.

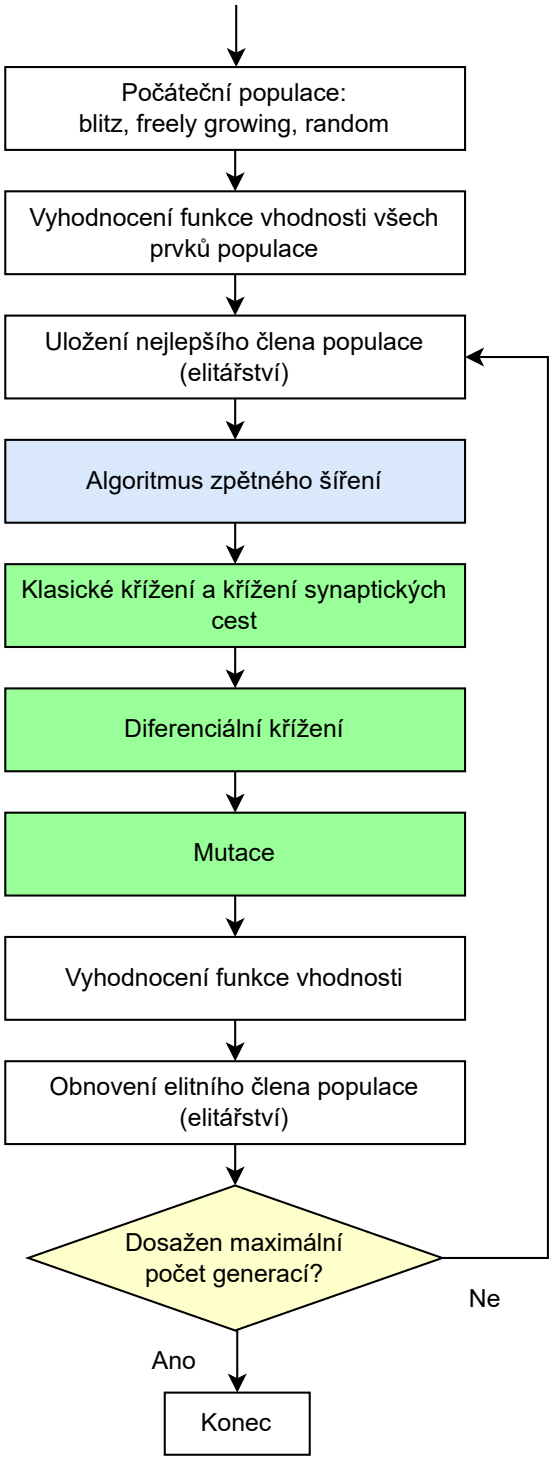
Kromě toho je ke každému prvku populace navíc přidán i parametr míry učení, setrvačnosti a počtu iterací zpětného šíření. Navíc je prvek ohodnocen ještě jednou metrikou, která udává úspěšnost učení pomocí zpětného šíření. Pokaždé, když je fáze zpětného šíření spuštěna, je vyhodnocena funkce vhodnosti před a po této fázi. Úspěšnost učení je pak prostým podílem nové a staré hodnoty funkce vhodnosti. S určitou pravděpodobností pak prvky populace přejímají tyto parametry zpětného šíření od prvků s větší úspěšností. Pokud se žádný prvek zpětným šířením dále nezlepšuje, je vygenerována nová sada parametrů. Tato modifikace umožňuje dynamické ladění těchto parametrů v průběhu učení. Není tedy potřeba je odhadovat.

Generování parametrů míry učení a setrvačnosti respektuje stejný postup, jako v algoritmu JADE pro diferenciální parametry F a CR. Zde jsou ale obě pravděpodobnostní rozdělení Gaussovská s rozptylem $0,5$, oříznutá na interval $(0, 1\rangle$. Střední hodnoty se počítají z aritmetických průměrů jedinců, kteří se algoritmem zpětného šíření zlepšují.

6.4.1 Metaheuristický algoritmus zpětného šíření

Tato varianta algoritmu je pouze modifikací výše uvedeného, pouze s tím rozdílem, že se nepoužívají prvky diferenciální evoluce pro křížení a mutaci samotných parametrů. Zůstává tak pouze algoritmus zpětného šíření a adaptivní ladění jeho parametrů.

Tato metoda byla navržena pouze jako referenční, aby bylo možné potvrdit či vyvrátit hypotézu, že prvky diferenciální evoluce dále zlepšují konvergenci. Pokud je tato hypotéza nesprávná, metody budou vykazovat velice podobný charakter učení a výsledná odchylka bude rovněž podobná.



Obrázek 6.3: Schéma fungování algoritmů diferenciální evoluce. Modrou barvou je označen krok vykonaný pouze ve druhé variantě algoritmu. Zelenou barvou jsou označeny kroky, které jsou vykonány pouze s určitou pravděpodobností.

7 Implementace

Architektura neuronové sítě, algoritmus zpětného šíření a výše uvedené algoritmy byly implementovány v jazyce C++ ve standardu C++17. Tento jazyk byl zvolen vzhledem k tomu, že jde o jazyk kompilovaný, a tedy jeho překladač potenciálně produkuje velmi výkonný strojový kód.

Pro potřeby této práce byla zvolena architektura vícevrstvého perceptronu [17], jejíž neurony jsou aktivovány pomocí standardní logistické sigmoidy.

Základem celé implementace je reprezentace neuronové sítě, respektive jejích vah. Ta je uložena ve struktuře `NETWORK`, jejímiž členy je atribut `inputs`, označující počet příznaků vstupních vektorů, a atribut `layers`. To je pole, které reprezentuje vrstvy sítě. Atribut `layer` je vektorem prvků typu `LAYER`, což je přepravka[1], která obsahuje pouze vektor neuronů `neurons`. Neuron je reprezentován třídou `NEURON`. Ta obsahuje dvě pole reálných čísel – `weights`, tedy aktuální váhy a `last_weights`, tedy váhy poslední iterace, které je potřeba ukládat kvůli metodě setrvačnosti.

Program nejprve načte parametry příkazové řádky, které jsou blíže popsány v příloze A. Mezi tyto parametry patří například počet vrstev a jejich neuronů, název vstupního a výstupního souboru, parametry zpětného šíření a genetického algoritmu a režim učení. Na základě parametrů vytvoří neuronovou síť a s tou dále pracuje. Pokud byla zvolena možnost načíst váhy ze souboru, soubor otevře a váhy načte. Poté dle zvoleného módu přejde buď k učení sítě zvolenou metodou, k validaci nebo průběžné klasifikaci.

7.1 Formáty dat

Program rozeznává dva formáty dat. Jeden pro množinu vstupních (trénovacích či validačních) dat, a jeden pro uložené váhy neuronové sítě. Oba tyto formáty jsou textové a ukládají pouze číselné hodnoty ve formátu s plovoucí desetinnou tečkou pro váhy sítě a hodnoty příznaků, a celočíselné hodnoty pro počty prvků.

[1] Přepravka je návrhový vzor, podle kterého je několik informací sloučeno do jednoho objektu, což usnadňuje opakovanou práci s touto skupinou informací.

Trénovací a validační data

Formát vstupních dat trénovací nebo validační množiny začíná vždy počtem prvků, které následují, a to na samostatné řádce. Každá další řádka pak označuje jeden prvek dané množiny, respektive jeho vektor příznaků a příslušnou odpověď učitele.

Příkladem může být výňatek z datové sady Iris1988, která je popsána v kapitole 8.1:

Výpis kódu 7.1: Výňatek z datové sady, ukázka formátu dat

```
6
5.1 3.5 1.4 0.2 1 0 0
4.9 3.0 1.4 0.2 1 0 0
5.6 3.0 4.5 1.5 0 1 0
5.8 2.7 4.1 1.0 0 1 0
6.9 3.2 5.7 2.3 0 0 1
5.6 2.8 4.9 2.0 0 0 1
```

Tato sada byla výrazně zredukována pro potřeby uvedeného příkladu. Počet příznaků a počet klasifikačních tříd je převzat z příkazové řádky jako parametr programu – pro zmíněnou datovou sadu je to číslo 4 pro počet příznaků a číslo 3 pro počet klasifikačních tříd. Lze tedy vidět, že na každém řádku jsou kódovány 4 příznaky prvku trénovací množiny a následně trojmístná odpověď učitele. Lze pozorovat, že pro potřeby klasifikace je odpověď učitele kódována jako one-hot vektor.

Formát dat pro klasifikaci

Pokud je vyžadován režim klasifikace bez následné validace, je vstupem soubor, který neobsahuje na prvním řádku počet prvků, ani neobsahuje odpovědi učitele na každém řádku. Jde tedy čistě o soubor, ve kterém jsou na každém řádku zvlášť zapsány příznakové vektory prvků ke klasifikaci.

Váhy neuronové sítě

Váhy sítě jsou serializovány po neuronech v jednotlivých vrstvách od váhy (neuronu, vrstvy) s nejmenším indexem po ten s indexem největším. Každá váha je uložena na vlastní řádce. Tyto váhy lze načíst v libovolném z režimů

– jak v režimu učení, kdy je načtená sada vah použita jako počáteční odhad, tak pochopitelně v režimu validace a průběžného učení, kdy slouží jako jediná parametrizace modelu pro klasifikaci vstupů.

7.2 Dopředné šíření

Po vytvoření sítě, a volitelně i načtení vah ze souboru, je možné síť použít k jejímu primárnímu účelu, kterým je v případě této implementace klasifikace vzorku. Každá klasifikace vyžaduje, aby byly spočteny aktivace neuronů všech vrstev až do poslední, výstupní. Tato výstupní vrstva, respektive aktivace jejích neuronů, slouží jako výstup dopředného šíření, a tedy kódovaný odhad klasifikace prvku.

Dopředné šíření lze vyvolat voláním funkce `calculateNetworkActivations` s parametrem neuronové sítě a vzorku, pro který mají být aktivace spočteny. Tato funkce používá dvojrozměrné vektory `tlPreactivations` a `tlActivations`, které jsou uloženy jako `thread_local`, tedy každé vlákno má svou kopii. Prvním rozměrem (indexem) je vrstva, druhým je neuron dané vrstvy. To z toho důvodu, aby byly alokovány pouze jednou, na začátku používání a nebylo je nutné vytvářet a ničit pokaždé, když je vyžadováno spočtení aktivací. Vzhledem k tomu, že vektor předaktivace a aktivace každé vrstvy je stále stejně velký díky neměnnosti sítě, je tento buffer až do konce stejně velký. Neustálá alokace a dealkoace by přinesla další režii navíc a celý program by zbytečně zpomalila.

Spočítání aktivací celé sítě sestává vždy z inicializace vyrovnávací paměti, pokud už k tomu nedošlo v minulosti, a následně se spočte aktivace první (skryté) vrstvy. V té je použit jako vstup klasifikační vzorek namísto neuronů sítě, a tak jde o výjimku z obecného chování kódu. Nejprve je spočtena předaktivace každého neuronu vrstvy jako součet součinů prvků vstupního vektoru a odpovídajících vah. Následně je voláním funkce `sigmoid` spočtena aktivace. V každé vrstvě kromě poslední je připojena jedna, „virtuální“ aktivace s hodnotou -1, která slouží jako *bias*. Tato hodnota je přidávána, aby se střední hodnota aktivací pohybovala blíže nule, respektive aby nedocházelo k narůstající odchylce této střední hodnoty od nuly postupem šíření aktivací skrze vrstvy. To je nutné zajistit u aktivačních funkcí, u kterých toto není možné předpokládat implicitně – např. logistická sigmoida, která má obor hodnot $\langle 0, 1\rangle$, a tak tento posun střední hodnoty hrozí.

7.3 Zpětné šíření

Algoritmus zpětného šíření je implementován ve funkci `backpropagation`, která dále využívá funkce `getDelta` a `adjustWeights`.

Funkce `backpropagation` je obecně volána s parametrem sítě, pole vzorků, indexu do pole vzorků pro začátek dávky pro učení a počtu prvků, pole odpovědí učitele, dále parametr míry učení a setrvačnosti a maximální počet epoch. V této funkci je nejprve inicializován vektor pro výpočet gradientů, který je rovněž uložen jako `thread_local`, aby bylo možné algoritmus spouštět paralelně. Poté je spočtena střední kvadratická odchylka, která se pak použije pro výpočet míry zlepšení.

Následuje cyklus, jehož jedna iterace odpovídá jedné epoše učení algoritmem zpětného šíření. V tomto algoritmu je nejprve pro všechny prvky z trénovací množiny (respektive podmnožiny, dle hodnot parametrů volání funkce) spočten gradient ve všech vrstvách a neuronech voláním funkce `getDelta` a na základě těchto gradientů je vždy provedena úprava vah voláním funkce `adjustWeights`. Poté, co je tento proces opakován pro každý prvek trénovací množiny, je znovu vyhodnocena střední kvadratická odchylka. Pokud je tato odchylka menší, než interně stanovené `EPSILON` (nastaveno na $5 \cdot 10^{-8}$), algoritmus končí. Rovněž končí, pokud byl nastaven limit počtu epoch a dokončením této iterace byl dosažen.

Funkce `getDelta` pro výpočet gradientů nejprve spočítá aktivace neuronů celé sítě pro daný prvek z trénovací množiny. Následně dle algoritmu popsaného v 4.2 spočítá gradienty nejprve pro aktivace poslední vrstvy, jelikož jde opět o výjimku z obecného chování díky odpovědím učitele, a poté gradienty zbytku sítě.

Funkce `adjustWeights` na základě předaných gradientů, spočtených funkcí `getDelta`, upraví váhy všech neuronů. Kromě algoritmu zpětného šíření je přidána ještě setrvačnost, která k aktuální úpravě váhy přidá ještě určitou část rozdílu proti vahám v předchozí iteraci, čímž simuluje setrvačnost. Parametr setrvačnosti ovlivňuje, jakou mírou setrvačnost ovlivňuje tento posun.

7.4 ReiStein Blitz

Metoda Blitz je implementována v souboru `blitz.cpp` ve funkci `blitzInit`. Tato funkce přejímá jako parametr neuronovou síť, trénovací množinu, od-

povědi učitele a režim fungování. Algoritmus Blitz funguje ve dvou režimech.

Na začátku je pro všechny třídy vybrán reprezentant. Ten je vybrán jako první nalezený prvek, který dané třídě náleží. Za předpokladu náhodného rozmíchání prvků trénovací množiny před samotným učením je to v podstatě prvek náhodný. Následně jsou inicializovány všechny váhy dané sítě na dostatečně malé číslo. Poté jsou provedeny průchody sítí, a pro každou třídu je vybrán jeden neuron první skryté vrstvy. Jeho váhy jsou nastaveny na čísla odpovídajícím příslušným příznakům vstupního vektoru reprezentanta. Poté je vybrán příslušný neuron výstupní, odpovídající dané klasifikační třídě. Jeho vstupní váhy jsou nastaveny všechny na 0 vyjma váhy, která vede do příslušného reprezentativního neuronu předchozí vrstvy. Ta jediná je nastavena na číslo 1.

Metoda Blitz má dva režimy. První režim je již popsaný plnohodnotný algoritmus, druhý režim přeskakuje fázi nastavení vah na náhodná čísla. To se může hodit např. při hledání vah genetickým algoritmem, kdy chceme síť pouze obohatit o nějaký vzor, a tedy dodat stavební blok do daného kandidátního řešení.

7.5 Freely Growing

Metoda Freely Growing je implementována v souboru `freely_growing.cpp` ve funkci `freelyGrowingInit`. Tato funkce rovněž přejímá jako parametr neuronovou síť, trénovací množinu, odpovědi učitele a režim fungování. Algoritmus funguje ve třech režimech.

Na začátku algoritmu jsou všechny váhy nastaveny na 0 nebo dostatečně malé číslo (režim 3). Poté následuje cyklus, ve kterém jsou implementovány postupy popsané v kapitole 6.2 – pro každý prvek je vybrán neuron skryté vrstvy, který má váhy dostatečně podobné příznakům vstupního vektoru a případně je váženým průměrem neuron o tento prvek „obohacen“. Pokud se žádný dostatečně podobný neuron (jeho váhy) nenalezne, přiřadí se nový a váhy jsou nastaveny přímo na vstupní příznakový vektor. Algoritmus končí, pokud byly vypotřebovány všechny prvky trénovací množiny, nebo byl dosažen horní limit počtu neuronů skryté vrstvy.

Popsaný algoritmus odpovídá režimu 1 fungování této metody. Režim 2 na-

víc přidává normalizaci vah do rozmezí $\langle 0; 1\rangle$. Režim 3, jak bylo zmíněno výše, v první fázi nenastavuje váhy na 0, ale na dostatečně malá čísla.

7.6 Genetický algoritmus

Implementace genetického algoritmu se nachází v hlavičkovém souboru `genetic_algorithm.h`. Jelikož jde o generickou implementaci, byl použit šablonový systém pro odlišení verze se zpětným šířením a bez něj. Genetický algoritmus se nad danou neuronovou sítí spouští voláním funkce `genetic_algorithm<true>` pro verzi se zpětným šířením, popř. `genetic_algorithm<false>` pro verzi bez něj.

Pro potřeby genetického algoritmu byla vytvořena struktura `Genome`, která obsahuje následující prvky:

- `n` – neuronová síť, respektive strukturovaně uložené váhy synaptických spojení;
- `next_n` – parametrizace neuronové sítě, která je vytvořena během generace;
- `fitness` – aktuální hodnota funkce vhodnosti (prvku `n`);
- `next_fitness` – hodnota funkce vhodnosti parametrizace `next_n`;
- `diff_F` – hodnota diferenciálního parametru F;
- `diff_P` – hodnota diferenciálního parametru P pro křížení metodou `DE/cur-to-pbest/1`;
- `diff_CR` – hodnota parametru diferenciálního křížení CR;
- `diff_strategy` – strategie křížení, hodnota z výčtu `CrossoverStrategy`;
- `batch_size` – velikost trénovací podmnožiny;
- `batch_start` – první index do pole trénovacích prvků, značící první prvek trénovací podmnožiny.

Výčtový typ `CrossoverStrategy` pak obsahuje prvky `Best_1`, `Rand_1`, `Cur_To_Best_1` a `Cur_To_Rand_1` odpovídající strategiím křížení uvedeným v kapitole 6.3.

Struktura pro diferenciální evoluci se zpětným šířením `Genome_Backprop` od této třídy dědí a rozšiřuje ji o prvky:

- `backprop_rate` – míra učení zpětného šíření;
- `backprop_momentum` – míra setrvačnosti zpětného šíření;
- `backprop_epochs` – počet iterací zpětného šíření;
- `bp_fitness` – míra zlepšování pomocí algoritmu zpětného šíření.

Zvolené konstanty pravděpodobnosti fází genetického algoritmu jsou (pro verzi bez zpětného šíření a pro verzi s ním):

- `p_crossover_weights = 0.15` a `0.05` – pravděpodobnost klasického křížení;
- `p_crossover_pathway = 0.15` a `0.05` – pravděpodobnost křížení celých synaptických cest;
- `p_crossover_diff = 0.35` a `0.3` – pravděpodobnost diferenciálního křížení;
- `p_mutation = 0.05` a `0.05` – pravděpodobnost náhodné mutace jedné váhy pro jeden prvek populace;
- `p_backprop = 0` a `0.4` – pravděpodobnost spuštění algoritmu zpětného šíření;
- `p_meta_crossover = 0` a `0.1` – pravděpodobnost diferenciálního křížení parametrů algoritmu zpětného šíření (míry učení a setrvačnosti);
- `p_blitz = 0.002` a `0.002` – pravděpodobnost přegenerování prvku populace metodou Blitz;
- `p_freely = 0.002` a `0.002` – pravděpodobnost přegenerování prvku populace metodou Freely Growing;
- `p_randomgen = 0.002` a `0.01` – pravděpodobnost přegenerování prvku populace s náhodnými vahami;
- `p_elitereinit = 0.01` a `0.01` – pravděpodobnost nahrazení prvku prvkem elitním;
- `p_rate_change = 0.2` a `0.15` – pravděpodobnost vygenerování nových parametrů zpětného šíření a diferenciálních parametrů.

Konstanty pravděpodobnosti přegenerování celých prvků populace nebo nahrazení elitou jsou v platnosti pouze pro druhou polovinu seřazené populace, tedy pro prvky s horší hodnotou funkce vhodnosti.

Algoritmus nejprve vygeneruje prvky populace, přičemž prvních 5 prvků vygeneruje metodami Blitz (jeho dvěma variantami) a Freely Growing (jeho třemi variantami) a poslední prvek přejme ze vstupního parametru funkce. Tento prvek může být například pole vah, které byly výsledkem předchozích průběhů optimalizací, čímž je umožněno pokračování v evoluci se zachováním nejlepšího prvku z předchozího spuštění. Následně spočítá funkce vhodnosti a první polovinu pole seřadí funkcí `std::partial_sort`. Částečné řazení je důležité jen pro první polovinu, jelikož obsahuje prvky nejlepší. Pokud má nějaký prvek hodnotu funkce vhodnosti nižší tak, že spadá do druhé poloviny, automaticky se stává kandidátem na nahrazení a na pořadí již nezáleží. Pro první polovinu je rovněž řazení důležité, jelikož prvek s indexem 0 je vždy prvkem nejlepším.

Po úvodním vygenerování populace a jejím ohodnocení přichází hlavní smyčka algoritmu. Zde jsou implementovány odpovídající fáze algoritmu popsané v kapitole 6.3 a 6.4. Jako první dojde ke spuštění algoritmu zpětného šíření, a to vždy paralelně nad všemi prvky. Tato fáze je mezi prvky z hlediska pořadí provádění nezávislá, a tak není třeba dodatečná synchronizace. Při implementaci bylo zjištěno, že tato fáze dosahuje větší efektivity, pokud je v každé generaci provedena vždy jedna iterace algoritmu zpětného šíření a s určitou pravděpodobností je pak spuštěna s počtem iterací větším (odpovídajícím obsahu atributu `backprop_epochs` daného prvku populace). Následně je vyhodnocena míra zlepšování algoritmem zpětného učení jako aritmetický průměr předchozí míry zlepšování a prostý podíl aktuální chyby po zpětném šíření (`cur_error`) a před ním (`cur_fitness`):

```
bp_fitness = (bp_fitness + cur_error / cur_fitness) * 0.5
```

Následuje fáze, ve které může s určitou pravděpodobností dojít ke křížení. Vyhodnotí se pravděpodobnost spuštění této fáze a podle té se spustí buď křížení normální, křížení synaptických cest, diferenciální křížení, nebo se křížení v této generaci vynechá. Křížení probíhá klasicky dle algoritmu JADE [44].

Poté je s určitou pravděpodobností provedena náhodná mutace, která změní právě jednu váhu aktuálního neuronu na novou, náhodně vygenerovanou. Nakonec je pouze pro spodní polovinu populace dle funkce vhodnosti roz-

hodnuto, zda bude prvek přegenerován metodou Blitz, Freely Growing, náhodně, a nebo zda bude nahrazen elitním prvkem. Tyto pravděpodobnosti jsou vůči ostatním velmi malé, a tak k těmto jevům dochází velice zřídka.

Poté je vyhodnocena funkce vhodnosti nově vygenerovaných prvků populace a nadchází úprava středních hodnot rozdělení, které generují hodnoty parametrů F a CR. Dle postupu uvedených v kapitole 6.4 jsou spočteny střední hodnoty parametrů F a CR přeživších prvků a na jejich základě jsou upraveny hodnoty středních hodnot odpovídajících pravděpodobnostních rozdělení.

Až v tento moment je provedena selekce, tedy akceptace nebo zamítnutí nových genetických informací všech prvků populace. Poté je populace částečně seřazena až do poloviny a celý proces se opakuje, dokud není dosažen maximální počet generací.

8 Dosažené výsledky

Algoritmy byly otestovány na čtyřech datových sadách o různých počtech vstupních příznaků a klasifikačních tříd. Pro testování byly vybrány čistě klasifikační úlohy, ale algoritmy by mělo být s minimálními úpravami možné použít i pro problémy regresní. Datové sady byly získány ze stránek repozitáře datových sad Kalifornské univerzity – University of California Irvine [11] `https://archive.ics.uci.edu/ml`.

U algoritmů byla vyhodnocována zejména rychlost konvergence vzhledem k reálnému času. Pro algoritmus zpětného šíření byla zvolena zastavovací podmínka tak, že je algoritmus ukončen, pokud je rozdíl kvadratických odchylek mezi iteracemi menší než $\delta = 5 \cdot 10^{-8}$. Pro algoritmy založené na evolučních strategiích byla zastavovací podmínka stanovena na dosažení požadovaného počtu generací. Tento počet zadává uživatel jako vstup při spouštění programu.

Pro všechny datové sady byl zvolen vícevrstvý perceptron s jednou skrytou vrstvou. Velikost skryté vrstvy se pro každou sadu liší a byla empiricky stanovena dle charakteru a dimenze problému. Velikost výstupní vrstvy pak standardně odpovídá počtu klasifikačních tříd.

Úspěšnost klasifikace byla ověřena pouze na trénovací množině vzhledem k tomu, že se tato práce věnuje hlavně míře konvergence a alternativním přístupům k učení. Předpokládá se, že datová sada je dostatečně reprezentativní, neobsahuje chyby a globální minimum cenové funkce odpovídá parametrům, se kterými je možné úspěšně klasifikovat s požadovanou přesností.

8.1 Datové sady

K testování byly použity datové sady, které jsou shrnuty v tabulce 8.1

Iris1988

Jedná se o datovou sadu, ve které jsou vstupy tvořeny příznakovým vektorem vlastností květů kosatců [11]. Jsou tvořeny čtyřmi příznaky – všechny jsou zaznamenané délky a šířky částí květenství. Klasifikačními třídami jsou pak tři různé druhy kosatce. Datová sada vznikla již v roce 1936, do struk-

Název	Počet příznaků	Počet klas. tříd	Počet vzorků
Iris1988	4	3	150
Leaf2014	14	36	340
PenDigits1998	16	10	7495
Biodegrad2013	41	2	1055

Tabulka 8.1: Tabulka použitých datových sad včetně základních vlastností

turované formy pro strojové učení však byla převedena až v roce 1988.

Pro tuto datovou sadu byla zvolena velikost skryté vrstvy 20 neuronů vzhledem k malému počtu vstupních příznaků a pouze třem klasifikačním třídám.

Genetický algoritmus byl parametrizován velikostí populace 100 a počtem iterací 500. Míra učení zpětného šíření byla nastavena na 0,01, míra setrvačnosti na 0,2.

Leaf2014

Tato datová sada obsahuje příznaky získané z vlastností listů vybraných rostlin [36]. Dohromady jde o 16 příznaků, které zahrnují různé délky, velikosti, větvení, excentricitu, barevný kontrast a další vlastnosti. Klasifikačními třídami jsou pak různé druhy rostlin, kterých je dohromady 36. Učení na této datové sadě může vykazovat anomálie vzhledem k tomu, že pro 6 tříd nejsou v datech přítomny žádné vzorky.

Pro tuto datovou sadu byla zvolena velikost skryté vrstvy 50 neuronů. Pro tuto velikost vykazovaly všechny algoritmy nejlepší schopnst učení. Empiricky byla stanovena vzhledem k tomu, že architektura a výsledný model neuronové sítě není hlavním tématem této práce.

Genetický algoritmus byl parametrizován velikostí populace 50 a počtem iterací 200. Míra učení zpětného šíření byla nastavena na 0,02, míra setrvačnosti na 0,2.

PenDigits1998

Tato datová sada vznikla jako alternativa k datové sadě MNIST (Modified National Institute of Standards and Technology) [9]. Jedná se o sadu ručně psaných číslic, které byly zaznamenány v rozlišení 500×500 pixelů, ale byly

podvzorkovány specializovaným algoritmem na 4×4 pixely za zachování rozumného množství informace v poměru k velikosti vektoru příznaků [11]. Klasifikačními třídami jsou pak číslice od 0 do 9.

Pro tuto datovou sadu byla zvolena velikost skryté vrstvy 50 neuronů. Vzhledem k omezenému množství příznaků a jejich prostorově podvzorkovanému charakteru je takový počet více než dostačující.

Genetický algoritmus byl parametrizován velikostí populace 50 a počtem iterací 200. Míra učení zpětného šíření byla nastavena na 0,1, míra setrvačnosti na 0,2.

Biodegrad2013

Datová sada obsahuje vybrané příznaky molekulární struktury a klasifikuje je dle toho, zda jsou biologicky rozložitelné nebo ne [27]. Vstupní příznakový vektor sestává z 41 příznaků, které odpovídají různým biochemickým vlastnostem dané molekuly. Klasifikační třídy jsou pouze dvě – molekula je či není biologicky rozložitelná.

Pro tuto datovou sadu byla rovněž zvolena velikost skryté vrstvy 50 neuronů. V tomto případě lze argumentovat poměrem vstupních příznaků a počtu klasifikačních tříd a tím, že vybrané příznakové veličiny mohou korelovat.

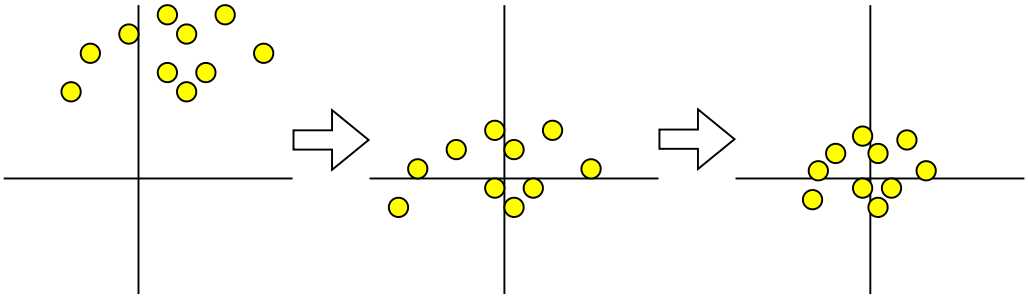
Genetický algoritmus byl parametrizován velikostí populace 50 a počtem iterací 200. Míra učení zpětného šíření byla nastavena na 0,01, míra setrvačnosti na 0,2.

8.2 Předzpracování dat

Data bylo nutné po získání převést do požadovaného vstupního formátu implementovaného řešení. K tomu byl vytvořen jednoduchý program, který je schopen převést různé reprezentace dat do jednotného výstupního formátu.

Kromě toho je v předzpracování dat pro strojové učení dobrou praktikou data normalizovat, normalizovat rozptyl ve všech příznacích a posunout střední hodnotu do počátku souřadnic. Učení by pak mělo vykazovat lepší výsledky [28]. Tento proces je znázorněn na obrázku 8.1.
Rovněž je vhodné vstupní data randomizovat. Randomizací se rozumí takový proces, kdy jsou vzorky náhodně zamíchány, aby se v procesu učení jednot-

livé třídy dostatečně často střídaly, a tedy nedošlo k cílenému přeučení ve prospěch jedné z klasifikačních tříd.



Obrázek 8.1: Znázornění posunutí střední hodnoty (první transformace) a normalizace rozptylu (druhá transformace).

8.3 Výsledky

V této podkapitole budou shrnuty a krátce diskutovány naměřené výsledky. Pro testování byl použit počítač s následujícími parametry:

- CPU Intel Core i5-4570 @ 3.20 GHz (4 jádra)
- 16 GB DDR3 RAM paměti
- OS MS Windows 10 Home 64-bit
- Kompilátor: MS Visual Studio 2022, kompilováno pro instrukční sadu AVX2 s maximální mírou optimalizace

Algoritmy jsou v tabulce zkráceny pomocí následujících zkratek:

- DE+BP – diferenciální evoluce s algoritmem zpětného šíření;
- BP – pouze algoritmus zpětného šíření;
- MetaBP – metaheuristická varianta diferenciální evoluce pro algoritmus zpětného šíření;
- BP+BL – algoritmus zpětného šíření s inicializací metodou Blitz;
- BP+FG – algoritmus zpětného šíření s inicializací metodou Freely Growing.

V grafech se ve vybraných místech určité veličiny zpožďují – to kvůli tomu, že např. genetický algoritmus poskytl první hodnotu (dokončil první generaci) až po daleko delší době, než algoritmus zpětného šíření. Vodorovná osa má logaritmické měřítko, aby byly lépe vidět rozdíly v křivkách. Svislá osa je přizpůsobena aktuálním hodnotám kvadratických odchylek.

8.3.1 Iris1988

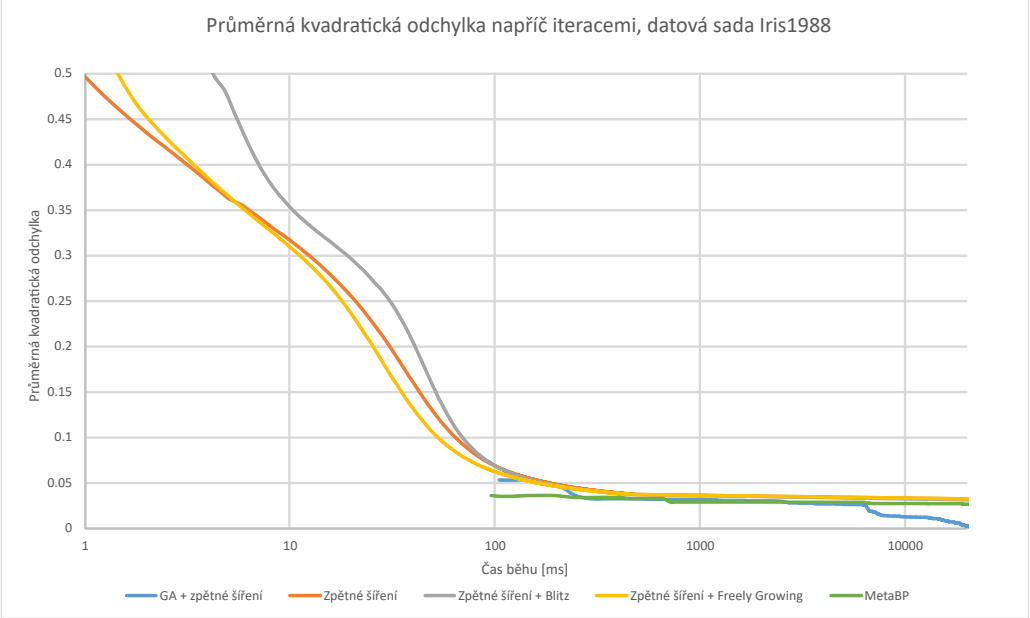
Výsledky měření jsou v tabulce 8.2, graf konvergence je na obrázku 8.2. Z výsledků je patrné, že algoritmus zpětného šíření a všechny jeho varianty postupně konvergují do přibližně stejné hladiny průměrné kvadratické odchylky bez výrazných rozdílů. Pouze metoda Blitz a Freely Growing v tomto případě neposkytla lepšího výchozího kandidáta, a tak konvergenci naopak zpomalila.

Algoritmus	Úspěšnost klasifikace	Prům. kvadr. odchylka
DE+BP	**100 %**	**0.00126**
BP	99.3 %	0.032
MetaBP	98.67 %	0.02527
BP+BL	99.3 %	0.0368
BP+FG	99.3 %	0.03692

Tabulka 8.2: Tabulka výsledků algoritmů pro datovou sadu Iris1988.

Oproti tomu genetické algoritmy sice konvergovaly do podobného minima, ale varianta s diferenciální evolucí nakonec nejspíše vlivem křížení a mutací nalezla minimum daleko lepší, potenciálně globální. To potvrzuje i úspěšnost klasifikace 100 % na trénovací množině.

Rychlost konvergence je do jisté míry po většinu času srovnatelná, genetický algoritmus však zvládl nalézt lepší řešení v závěru evoluce. Tento přelom ve variantě MetaBP nenastal nejspíše z toho důvodu, že tato varianta pouze adaptivním způsobem mění parametry zpětného šíření. Ke zlepšení tedy mohlo dojít pouze tehdy, pokud by parametry zpětného šíření byly nastaveny tak, že by lokální minimum a jeho oblast konvergence daný člen populace opustil a začal konvergovat k jinému extrému. To je však velice nepravděpodobné vzhledem k hornímu omezení parametru míry učení na 1, a také k tomu, že se evoluce nejspíš pohybovala naopak v číslech řádově nižších, jak je patrné z grafu.



Obrázek 8.2: Graf konvergence pro datovou sadu Iris1988.

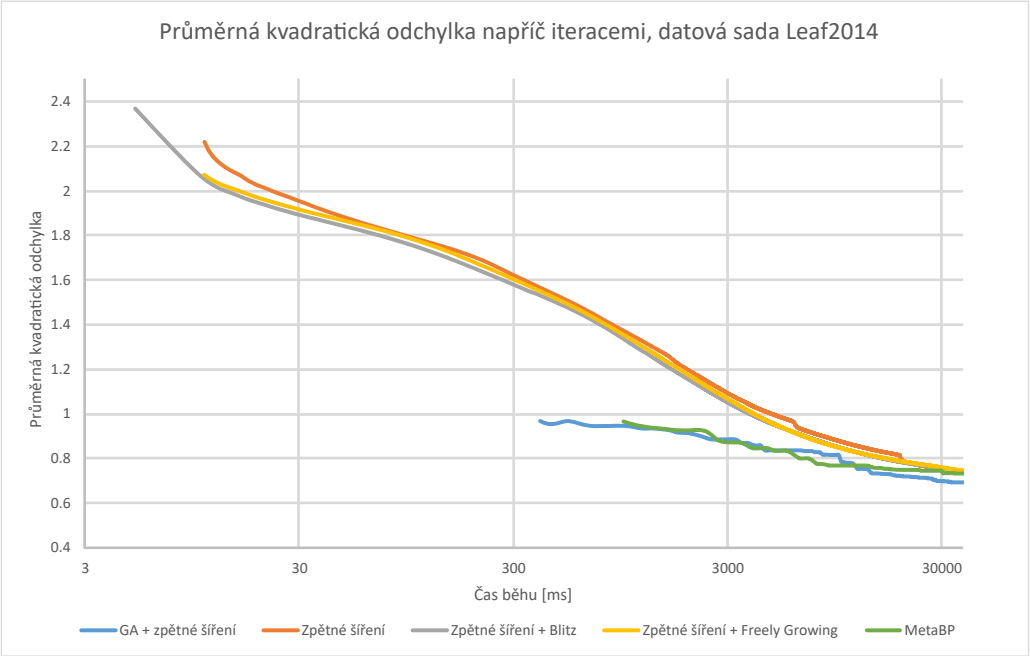
8.3.2 Leaf2014

Výsledky měření jsou v tabulce 8.3, graf konvergence je na obrázku 8.3. Na této datové sadě v konvergenci jasně dominuje genetický algoritmus. Díky úspěšné exploraci prostoru bylo vhodné řešení nalezeno už v počáteční populaci a první generaci, což je téměř jistě způsobeno tím, že se již v úvodní populaci nacházel nějaký prvek blízký dobrému minimu. Konvergence diferenciální evoluce je pak sice pomalejší, konverguje ale k lepšímu výsledku o něco dříve. Co se týče metody Blitz a FreelyGrowing, v tomto případě již na

Algoritmus	Úspěšnost klasifikace	Prům. kvadr. odchylka
DE+BP	**95.29 %**	**0.69257**
BP	92.64 %	0.74276
MetaBP	94.12 %	0.70316
BP+BL	92.65 %	0.73608
BP+FG	91.76 %	0.74248

Tabulka 8.3: Tabulka výsledků algoritmů pro datovou sadu Leaf2014.

začátku poskytly lepšího kandidáta, než náhodná inicializace. Konvergence je o něco rychlejší, než u běžného algoritmu zpětného šíření a nalezené minimum má srovnatelné vlastnosti, jako to, které nalezl náhodně inicializovaný algoritmus.



Obrázek 8.3: Graf konvergence pro datovou sadu Leaf2014.

V tomto případě tedy opět diferenciální evoluce se zpětným šířením nalezla lepší řešení rychleji. Varianta MetaBP však ustrnula opět v podobném minimu, jako algoritmus zpětného šíření. To odpovídá předchozímu případu, a odůvodnění je tím pádem identické – z nalezeného lokálního minima se již nelze dostat do jiného jen pomocí zpětného šíření.

8.3.3 PenDigits1998

Výsledky měření jsou v tabulce 8.4, graf konvergence je na obrázku 8.4. Zde je opět zvýrazněna explorativní schopnost genetických algoritmů a diferenciální evoluce. Již první generace poskytla dostatečně dobrou parametrizaci sítě pro úspěšnou klasifikaci.

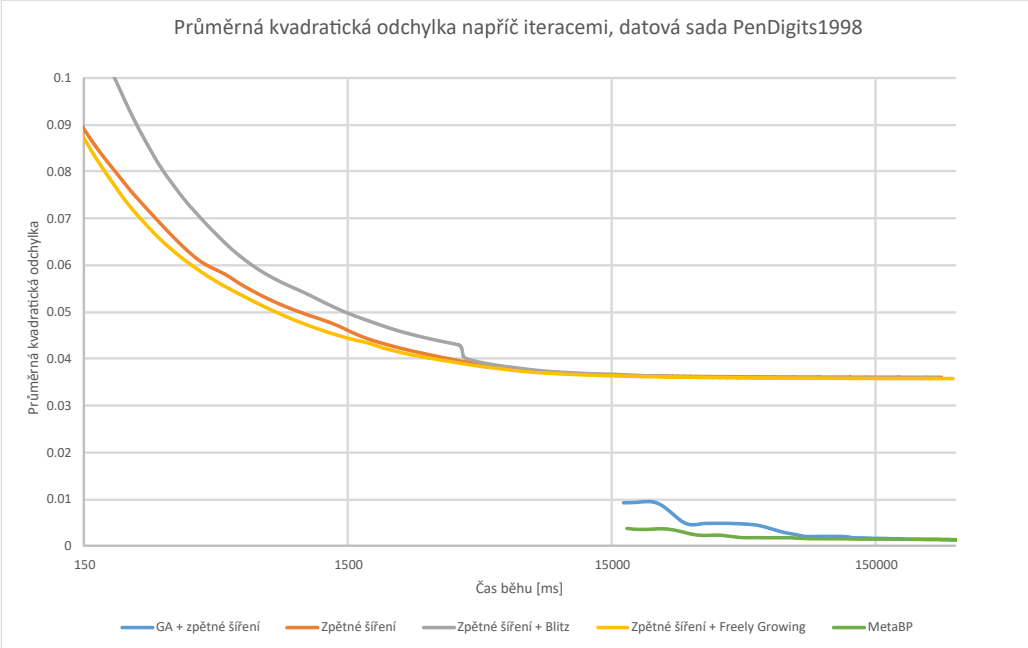
Algoritmus zpětného šíření zde očividně ustrnul v lokálním minimu, ze kterého se již nedostal. Inicializace metodou Blitz poskytla horší počáteční parametrizaci, než metoda Freely Growing. To lze všeobecně očekávat vzhledem k tomu, že metodu Freely Growing lze z definice za určitých okolností považovat za evoluci metody Blitz, byť je jejich původ zdůvodněn jinak.

Diferenciální evoluce opět poskytla nejlepší řešení v nejlepším čase. Tentokrát je ale míra konvergence lepší v případě metody MetaBP. To lze přisoudit

částečně náhodě, a částečně faktu, že díky oproštění algoritmu o diferenciálně evoluční kroky nad vahami synapsí je výpočtně méně náročný, a tak tedy k výsledku konverguje o něco rychleji.

Algoritmus	Úspěšnost klasifikace	Prům. kvadr. odchylka
DE+BP	**99.92 %**	**0.00102**
BP	96.42 %	0.03605
MetaBP	99.91 %	0.00115
BP+BL	96.44 %	0.03591
BP+FG	96.44 %	0.03578

Tabulka 8.4: Tabulka výsledků algoritmů pro datovou sadu PenDigits1998.



Obrázek 8.4: Graf konvergence pro datovou sadu PenDigits1998.

8.3.4 Biodegrad2013

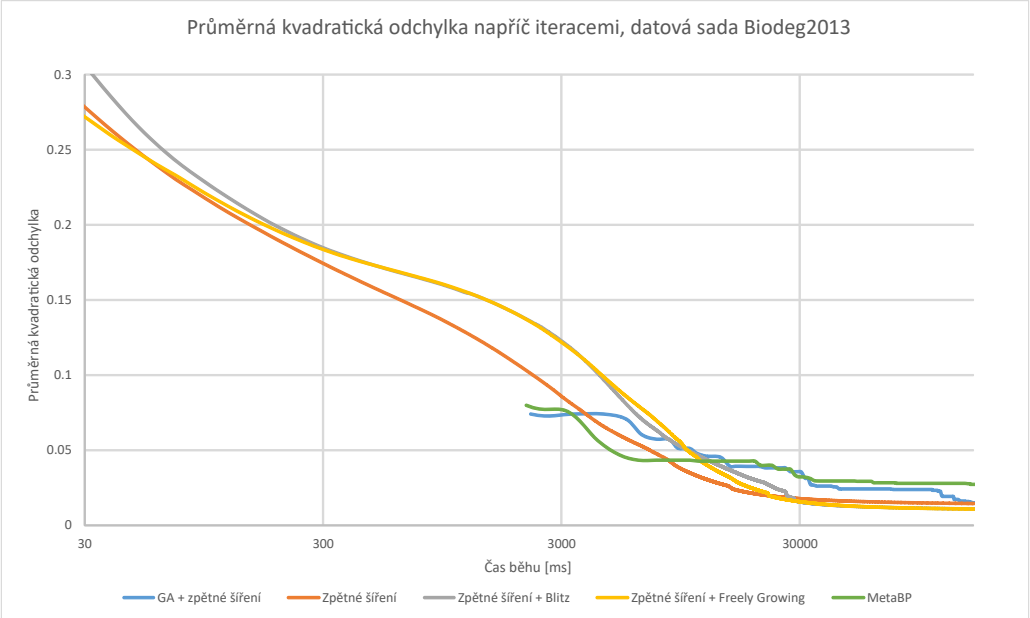
Výsledky měření jsou v tabulce 8.5, graf konvergence je na obrázku 8.5. Zde všechny algoritmy vykazují velice podobnou míru konvergence i úspěšnost klasifikace. Počátek byl sice příznivější pro diferenciální evoluci a její variantu, ovšem jejich výpočetní náročnost zde konvergenci naopak zpomalila. Úspěšnost klasifikace ale i pro algoritmus vykazující nejhorší výsledky přesahuje 98 %.

Diferenciální evoluce se zpětným šířením vykazuje téměř nejlepší výsledky, v tomto případě však tento algoritmus předčilo zpětné šíření inicializované metodami Blitz a Freely Growing.

Tento jev lze odůvodnit tím, že díky charakteru prvků trénovací množiny existuje jedno výrazné globální minimum, a tak metaheuristické vlastnosti diferenciální evoluce a modifikace MetaBP pomohou pouze minimálně, ovšem za cenu větší výpočetní náročnosti.

Algoritmus	Úspěšnost klasifikace	Prům. kvadr. odchylka
DE+BP	99.43 %	0.014491
BP	99.34 %	0.014371
MetaBP	98.67 %	0.027145
BP+BL	**99.53 %**	**0.010683**
BP+FG	**99.53 %**	**0.010693**

Tabulka 8.5: Tabulka výsledků algoritmů pro datovou sadu Biodegrad2013.



Obrázek 8.5: Graf konvergence pro datovou sadu Biodegrad2013.

8.4 Shrnutí

Z naměřených výsledků je patrné, že algoritmus diferenciální evoluce se zpětným šířením má nemalý potenciál pro učení neuronových sítí. Pochopitelně ověření proběhlo pouze na relativně malé sadě testovacích množin, a tak je nutné provést další, výpočetně mnohem náročnější testy, aby se tato hypotéza potvrdila nebo vyvrátila.

Kombinace diferenciální evoluce a metaheuristiky pro zpětné šíření se ukázala být relativně silnou. Zpětné šíření je vhodné pro lokální optimalizaci cenové funkce, kdy je do značné míry zajištěna konvergence za předpokladu správného nastavení parametrů učení. Diferenciální evoluce je pak nástrojem pro globální prohledávání a optimalizaci, a tedy poskytuje daleko lepší možnosti explorace prostoru, než obyčejné zpětné šíření inicializované náhodnou sadou vah.

Algoritmy Blitz a Freely Growing pro inicializaci poskytly rovněž relativně kvalitní počáteční odhad parametrizace vah neuronové sítě. Samy o sobě však síť naučit nedovedou, a tak je vždy potřeba algoritmus další, který z počátečního odhadu dovede najít dostatečně dobré minimum. Rovněž inicializace prvků populace genetických algoritmů těmito metodami přinesla určité zlepšení.

9 Závěr

V úvodu práce je popsána biologická podstata učení. Jsou zde vysvětleny základní stavební jednotky centrální nervové soustavy, které se na učení podílejí. Na tyto poznatky dále navazuje analytická část práce. Po úvodní, biologicky orientované části práce následuje kapitola věnující se teorii umělých neuronových sítí a kapitola věnující se jejich učení. Následuje analýza existujících alternativních metod učení neuronových sítí k metodě zpětného šíření. Na základě těchto poznatků jsou pak navrženy 3 metody učení, které na zpětném šíření nezakládají, nebo jej kombinují s metodami jinými za účelem potlačení známých problémů. Navržené metody byly implementovány a implementace je popsána v další kapitole. Výsledky jsou v závěru práce následně ověřeny a diskutovány. Zadání bylo splněno v celém rozsahu.

Navržené metody byly ověřeny na čtyřech datových sadách, přičemž bylo zjištěno, že nejlepších výsledků na většině datových sad dosahuje algoritmus, který kombinuje diferenciální evoluci, zpětné šíření i metody ReiStein Blitz a Freely Growing. Jediná datová sada, na které se tento algoritmus neumístil na prvním místě je Biodegrad2013, avšak i zde dosahovala tato nová metoda srovnatelných výsledků. Nejlepší přesnosti klasifikace na této datové sadě pak dosáhly neuronové sítě učené metodami zpětného šíření, inicializovaných jak metodou ReiStein Blitz, tak metodou Freely Growing.

Všechny navržené metody úspěšně trénují neuronovou síť s architekturou vícevrstvého perceptronu. Rychlost trénování je vzhledem k reálnému výpočetnímu času celkově lepší, než u obyčejné metody zpětného šíření. Stabilita konvergence však může být u diferenciální evoluce mírně horší vzhledem k tomu, že jde o stochastickou metodu. Její kombinace s metodou zpětného šíření však tento problém částečně potlačuje. Pro další validaci těchto závěrů je však nutné algoritmy otestovat na rozmanitější množině trénovacích sad.

Implementace a myšlenka metod ReiStein Blitz a Freely Growing je však zatím omezena pouze na jednu skrytou vrstvu architektury vícevrstvého perceptronu. Možným pokračováním této práce by jistě mohlo být zobecnění myšlenky těchto metod jak na sítě s větším počtem skrytých vrstev, tak i jiné architektury umělých neuronových sítí. Zajímavým kandidátem na toto rozšíření je pak architektura neuronové sítě obsahující neurony s jiným druhem aktivační funkce, např. radiální bázovou funkcí.

Seznam obrázků

Literatura

[1] Albawi, S. – Mohammed, T. A. – Al-Zawi, S. Understanding of a Convolutional Neural Network. In *2017 International Conference on Engineering and Technology (ICET)*, s. 1–6. IEEE, 2017.

[2] Anil Ananthaswamy. *Researchers Build AI That Builds AI* [online]. Quanta magazine, 2022. [cit. 12.3.2022]. Dostupné z: `https://www.quantamagazine.org/researchers-build-ai-that-builds-ai-20220125`.

[3] Blatný, J. et al. *Číslicové počítače, 1. vydání*. 1980. Dostupné z: `https://www.fit.vut.cz/research/publication/5687`.

[4] Brabazon, A. – O'Neill, M. – McGarraghy, S. *Natural Computing Algorithms*. 554. Springer, 2015.

[5] Bullen, P. S. *Handbook of Means and Their Inequalities*. 560. Springer Science & Business Media, 2013.

[6] Čapek, K. *R.U.R.: Rossumś universal robots*. Praha: Artur, 1920. ISBN 80-86216-46-2.

[7] Choromanska, A. et al. Beyond Backprop: Online Alternating Minimization With Auxiliary Variables. In *International Conference on Machine Learning*, s. 1193–1202. PMLR, 2019.

[8] Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of control, signals and systems*. 1989, 2, 4, s. 303–314.

[9] Deng, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*. 2012, 29, 6, s. 141–142.

[10] Doherty, S. *Control of pH in Chemical Processes Using Artificial Neural Networks*. PhD thesis, School of Engineering Liverpool, John Moores University, 1999.

[11] Dua, D. – Graff, C. UCI Machine Learning Repository, 2017. Dostupné z: `http://archive.ics.uci.edu/ml`.

[12] Fath, A. H. – Madanifar, F. – Abbasi, M. Implementation of Multilayer Perceptron (MLP) And Radial Basis Function (RBF) Neural Networks to Predict Solution Gas-Oil Ratio of Crude Oil Systems. *Petroleum*. 2020, 6, 1, s. 80–91.

[13] Forrest, S. – Mitchell, M. Relative Building-Block Fitness and the Building-Block Hypothesis. In *Foundations of genetic algorithms*, 2, s. 109–126. Elsevier, 1993.

[14] Gago, J. J. et al. Sequence-to-Sequence Natural Language to Humanoid Robot Sign Language. *10th EUROSIM Congress on Modelling and Simulation*. 2019.

[15] Georgioudakis, M. – Plevris, V. A Comparative Study of Differential Evolution Variants in Constrained Structural Optimization. *Frontiers in Built Environment*. 2020, 6, s. 102.

[16] Gupta, J. N. – Sexton, R. S. Comparing Backpropagation With a Genetic Algorithm for Neural Network Training. *Omega*. 1999, 27, 6, s. 679–684.

[17] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994.

[18] Hussain, A. – Muhammad, Y. S. Trade-off Between Exploration and Exploitation With Genetic Algorithm Using a Novel Selection Operator. *Complex & intelligent systems*. 2020, 6, 1, s. 1–14.

[19] IBM Cloud Education. *Gradient Descent* [online]. IBM, 2020. [cit. 12.1.2022]. Dostupné z: `https://www.ibm.com/cloud/learn/gradient-descent`.

[20] IBM Cloud Education. *Recurrent Neural Networks* [online]. IBM, 2020. [cit. 8.1.2022]. Dostupné z: `https://www.ibm.com/cloud/learn/recurrent-neural-networks`.

[21] Idrissi, M. A. J. et al. Genetic Algorithm for Neural Network Architecture Optimization. In *2016 3rd International Conference on Logistics Operations Management (GOL)*, s. 1–4. IEEE, 2016.

[22] Jaderberg, M. et al. Decoupled Neural Interfaces Using Synthetic Gradients. In *International conference on machine learning*, s. 1627–1635. PMLR, 2017.

[23] Knierim, J. J. The Hippocampus. *Current Biology*. 2015, 25, 23, s. R1116–R1121.

[24] Lee, D.-H. et al. Difference Target Propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, s. 498–515. Springer, 2015.

[25] Lee, K. – Lam, H. Optimising Neural Network Weights Using Genetic Algorithms: A Case Study. In *Proceedings of ICNN'95-International Conference on Neural Networks*, 3, s. 1384–1388. IEEE, 1995.

[26] Ma, W.-D. K. – Lewis, J. – Kleijn, W. B. The HSIC Bottleneck: Deep Learning Without Back-Propagation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 34, s. 5085–5092, 2020.

[27] Mansouri, K. et al. Quantitative Structure – Activity Relationship Models for Ready Biodegradability of Chemicals. *Journal of chemical information and modeling*. 2013, 53, 4, s. 867–878.

[28] Nawi, N. M. – Atomi, W. H. – Rehman, M. Z. The Effect of Data Pre-processing on Optimized Training of Artificial Neural Networks. *Procedia Technology*. 2013, 11, s. 32–39.

[29] Nicholls, J. G. et al. *From Neuron to Brain*. 271. Sinauer Associates Sunderland, MA, 2001. ISBN 1605354392.

[30] Nøkland, A. – Eidnes, L. H. Training Neural Networks With Local Error Signals. In *International conference on machine learning*, s. 4839–4850. PMLR, 2019.

[31] Panchal, G. et al. Behaviour Analysis of Multilayer Perceptrons With Multiple Hidden Neurons and Hidden Layers. *International Journal of Computer Theory and Engineering*. 2011, 3, 2, s. 332–337.

[32] Raschka, S. *Machine Learning* [online]. 2020. [cit. 20.2.2022]. Dostupné z: `https://sebastianraschka.com/faq/docs/visual-backpropagation.html`.

[33] Saladin, K. *Anatomy & Physiology: The Unity of Form and Function*. New York: McGraw-Hill, 2015.

[34] Santuz, A. *Extracting Muscle Synergies From Human Steady and Unsteady Locomotion: Methods and Experiments*. PhD thesis, Humboldt-Universität zu Berlin, 2018.

[35] Sharma, S. – Sharma, S. – Athaiya, A. Activation Functions in Neural Networks. *Towards data science*. 2017, 6, 12, s. 310–316.

[36] Silva, P. F. – Marcal, A. R. – Silva, R. M. Evaluation of Features for Leaf Discrimination. In *International Conference Image Analysis and Recognition*, s. 197–204. Springer, 2013.

[37] Šíma, J. – Neruda, R. *Teoretické otázky neuronových sítí*. Praha: Matfyzpress, 1996. ISBN 80-85863-18-9.

[38] Sovák, M. *Biologické základy učení*. Státní pedagogické nakladatelství, Praha, Czech Republic, 1985. ISBN 14-316-85.

[39] Storn, R. – Price, K. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces. *Journal of global optimization*. 1997, 11, 4, s. 341–359.

[40] Trojan, S. *Lékařská fyziologie*. Grada Publishing a.s., 2008. ISBN 80-247-0512-5.

[41] Uldrich, M. – Jurczyk, T. *Neuronové sítě a jejich využití* [online]. 2014. [cit. 4.2.2022]. Dostupné z: `https://www.systemonline.cz/clanky/neuronove-site-a-jejich-vyuziti-1.htm`.

[42] Volná, E. *Neuronové sítě 1* [online]. Ostrava: Ostravská univerzita v Ostravě. Vydání: druhé, 2008. [cit. 12.3.2022]. Dostupné z: `https://web.osu.cz/~Volna/Neuronove_site_skripta.pdf`.

[43] Wang, B.-C. et al. Composite Differential Evolution for Constrained Evolutionary Optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. 2018, 49, 7, s. 1482–1495.

[44] Zhang, J. – Sanderson, A. C. JADE: Adaptive Differential Evolution with Optional External Archive. *IEEE Transactions on evolutionary computation*. 2009, 13, 5, s. 945–958.

[45] Zheng, F. – Zecchin, A. C. – Simpson, A. R. Self-Adaptive Differential Evolution Algorithm Applied to Water Distribution System Optimization. *Journal of Computing in Civil Engineering*. 2013, 27, 2, s. 148–158.

A Uživatelská příručka

V této příloze bude popsáno použití vytvořeného programu včetně jeho různých režimů fungování.

Softwarové a hardwarové požadavky

Pro spuštění přiloženého programu je nutné mít nainstalovaný operační systém MS Windows verze 7 nebo vyšší. Dále je potřeba mít nainstalovaný balíček Visual Studio Redistributable, který je přibalený v archivu této práce, nebo je možné jej stahnout na následující adrese: `https://aka.ms/vs/17/release/vc_redist.x64.exe`.

Přiložený program je optimalizovaný pro novější procesory, a je tedy dále nutné, aby byl procesor vybaven instrukční sadou AVX2. Toto kritérium splňují například procesory Intel mikroarchitektury Haswell a novější (např. Intel Core i3-4130), nebo procesory AMD mikroarchitektury Excavator a novější (např. AMD Athlon X4 845). Dále je potřeba mít dostatečně velkou operační paměť. Konkrétní velikost se odvíjí od datové sady, která má být použita pro učení neuronové sítě. Pro přiložené datové sady stačí, aby měl program k dispozici 512 MB operační paměti.

Spuštění

Program je spouštěn příkazem `mlp.exe`. Bez dalších dodaných parametrů program pouze vypíše nápovědu.

Mezi parametry programu patří:

- `-i <#cislo>` – velikost vstupního příznakového vektoru, např. `-i 12`;
- `-o <#cislo>` – počet klasifikačních tříd, velikost výstupní vrstvy, např. `-o 3`;
- `-l <#pocet> <#vrstva1> ... <#vrstvaN>` – počet vrstev a počet neuronů v každé vrstvě včetně výstupní, např. `-l 2 20 3`;
- `-s` – prvky budou před učením náhodně zamíchány;

- `-v` – spouští režim validace nad vstupním souborem;
- `-c` – spouští režim klasifikace nad vstupním souborem;
- `-fi <cesta>` – cesta k souboru, který obsahuje prvky trénovací nebo validační množiny, např. `-fi train.txt`;
- `-fm <cesta>` – cesta k souboru s vahami sítě; tento soubor je použit jako výstupní v režimu učení a jako vstupní při validaci a klasifikaci, např. `-fm model.txt`;
- `-b <#rezim>` – neuronová síť bude inicializována metodou Blitz v daném režimu, např. `-b 1`;
- `-q <#rezim>` – neuronová síť bude inicializována metodou Freely Growing v daném režimu, např. `-q 1`;
- `-r <parametr>` – nastavení parametru míry učení, např. `-r 0.1`;
- `-m <parametr>` – nastavení parametru setrvačnosti, např. `-m 0.2`;
- `-w` – načte váhy ze zadaného souboru; typické je použití pro režim validace a klasifikace, jelikož jinak k načtení nedojde, je ale možné použít tento parametr i pro načtení vah jako počáteční odhad vah k dalšímu učení;
- `-g<rezim> <#populace>,<#generace>` – neuronová síť bude místo zpětného šíření učena variantou genetického algoritmu, možné varianty jsou následující:
  - `-gn <#populace>,<#generace>` – diferenciální evoluce (bez zpětného šíření), např. `-gn 50,200`
  - `-gb <#populace>,<#generace>` – diferenciální evoluce se zpětným šířením, např. `-gb 50,200`
  - `-gm <#populace>,<#generace>` – metaheuristická varianta backpropagation (MetaBP), např. `-gm 50,200`;
- `-x` – síť už nebude dále učena zpětným šířením nebo jinou metodou.

Příklad spuštění

Chceme-li například naučit neuronovou síť o 20 skrytých neuronech genetickým algoritmem se zpětným šířením na datové sadě Iris1988, musíme si

nejprve zjistit, kolik vstupních příznaků a kolik klasifikačních tříd tato datová sada má. Z kapitoly 8.1 se dozvíme, že jsou to 4 vstupní příznaky a 3 klasifikační třídy. Rovněž budeme určitě chtít náhodně zamíchat prvky trénovací množiny. Parametry genetického algoritmu stanovme na velikost populace 100 prvků a 500 generací. Data načítáme ze souboru `iris.txt` a výsledné váhy uložíme do souboru `iris_model.txt`.

Pro učení tedy můžeme spustit příkaz (jedná se o jeden dlouhý příkaz, lomítko je vloženo pro zdůraznění, že nejde o konec příkazu):

```
mlp.exe -i 4 -o 3 -l 2 20 3 -s -fi iris.txt \
        -fm iris_model.txt -gb 100,500
```

Výstupem pak bude výpis procesu učení zakončený výstupem, ve kterém je uvedena průměrná kvadratická odchylka po dokončení učení.

Pokud je výpočet příliš dlouhý, varianta s genetickými algoritmy průběžně ukládá váhy do souboru i během učení, a to po dokončení každé generace, pokud od posledního uložení uběhla více než jedna minuta, a nebo každou 50. generaci. Tyto váhy jsou uloženy do souboru pojmenovaného stejně jako výstupní, ovšem navíc s příponou `.tmp`, např. `iris_model.txt.tmp`.

Pokud chceme zjistit, jak úspěšný byl proces učení, a tedy jak dobře jsme schopni klasifikovat prvky trénovací množiny, můžeme spustit následující příkaz:

```
mlp.exe -i 4 -o 3 -l 2 20 3 -fi iris.txt -fm iris_model.txt \
        -v -w
```

Pozor, je potřeba kromě daného režimu (validace, `-v`) dodat i parametr pro načtení vah ze souboru (`-w`), jinak síť nebude klasifikovat dle nalezených vah.

Pokud chceme tentýž proces učení opakovat s metodou zpětného šíření, lze první příkaz obměnit spustit bez parametru `-gb`. Typicky ale chceme upravit míru učení a setrvačnosti, čehož docílíme parametry `-r` a `-m`:

```
mlp.exe -i 4 -o 3 -l 2 20 3 -s -fi iris.txt \
        -fm iris_model.txt -r 0.1 -m 0.15
```

K ověření úspěšnosti klasifikace pak můžeme použít stejný příkaz.