Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Implementace neuronových sítí v objektově orientovaném jazyce

Diplomová práce

Autor: Bc. Jan Trejbal
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Richard Cimler

Hradec Králové duben 2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 29. 4. 2016

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Richardu Cimlerovi za metodické vedení práce, dále Mgr. et Mgr. Rafaelovi Doležalovi Ph.D. a Ing. Agátě Milanov Ph.D. za konzultace v průběhu výzkumu a rodině a přátelům za velkou podporu.

Anotace

Neuronové sítě, díky vývoji výpočetních kapacit moderních počítačů, zaznamenali značné rozšíření do spousty oblastí lidské činnosti. Přístupy k implementaci takových sítí jsou velmi rozmanité. Cílem této práce je analyzovat neuronové sítě a objektový přístup a navrhnout implementační přístup neuronových sítí tak, aby využíval všech předností objektově orientovaného programování, a následně pomocí tohoto přístupu vytvořit knihovnu zaměřenou na experimentování s jednotlivými typy neuronových sítí.

V prvních kapitolách jsou rozebrány biologické a umělé neuronové sítě a objektově orientované programování, které jsou základem pro navrhované řešení. V těchto částech je provedena analýza každé oblasti, nutná pro navržení vlastního řešení, skládající se z popisu hlavní myšlenky a jednotlivých funkcí. Dále jsou v práci popsány již existující přístupy k implementaci neuronových sítí ve formě softwaru ve formě implementačních přístupů a vybraných knihoven zaměřených na neuronové sítě. Následuje detailní rozbor autorova vlastního návrhu spojení neuronových sítí a objektového přístupu s popisem základní myšlenky a výčtem entit, které byly v rámci modelu vytvořeny.

V závěrečných kapitolách je popsána implementace tohoto modelu ve formě knihovny napsané v programovacím jazyce Java – NeuralUHK – s popisem nejdůležitějších tříd, jejich odpovědností, metod a jejich umístěním v rámci celého systému. Následně jsou vyjmenovány a zhodnoceny experimenty, které byly v rámci vývoje aplikace provedeny k otestování aplikace na vybraných datech.

Na závěr je shrnut výsledek celého výzkumu a vyjmenovány možné směry dalšího vývoje aplikace a směry použití, pro které tento návrh není vhodný.

Annotation

Title: Implementation of Neural Networks in Object Oriented Programming language.

Neural networks have spread in many domains of human activities thanks to the evolution of performance of modern computers. Approaches to implement these networks are various. Goal of this research is to analyse neural networks and object oriented programming (OOP) and propose implementation model which utilizes benefits of OOP and then with this model create library for experimenting with different types of neural networks.

In first chapters are described biological and artificial neural networks and object oriented programming forming basis for proposed solution, which contains description of main idea and individual functions. These parts contain analysis of every part necessary for designing of model. Next chapter contains description of existing approaches to implement neural network in forms of approaches and selected libraries dedicated to neural networks. It follows detail analysis of authors own concept of merging neural networks and OOP with description of basic idea and list of entities which were created for model.

Next chapters contains implementation of this model in form of library written in Java Programming Language – NeuralUHK – with description of important classes, their responsibilities and their deployment in whole app. Follows list of experiments which were done to test application on selected data sets.

In last part is summarized result of whole research and are listed possible ways of future application development and ways of usage for which this model is not suited.

Obsah

1 Úvod

Umělá inteligence vždy lákala autory Science Fiction. Stroj, který by napodoboval lidskou inteligenci, je jedním z nejoblíbenějších témat tohoto žánru. Ale co to umělá inteligence je? Jedná se o takový stroj, který vykazuje známky inteligentního chování – rozumu. Dovede uvažovat, učit se, plánovat atd. Výzkumem umělé inteligence a její aplikace v praxi se zabývá celý jeden obor matematické informatiky.

Jedním z příkladů umělé inteligence jsou neuronové sítě (NS). Jedná se o matematický model, který se inspiruje neurony v lidském mozku. Myšlenka neuronových sítí, není nová, právě naopak. Bohužel, praktická aplikace v počítačových programech byla omezena technologii doby, protože učení NS je výpočetně velmi náročná úloha. Až v současnosti je, díky pokroku v HW technologiích, možné tuto myšlenku prakticky využít a aplikovat ji v praxi. Procesor totiž dříve nestačil tak velké množství operací vykonat v rozumném čase.

Tato práce je zaměřena na jejich implementaci – tj. využít myšlenku neuronových sítí a na jejím základě vytvořit aplikaci ve zvoleném programovacím jazyce. Implementace NS by se dala rozdělit na dvě kategorie: pro praktické nasazení a pro testování různých typů NS. Oba dva přístupy jsou důležité, ale jsou od sebe značně odlišné. Praktickým nasazením je myšlena aplikace NS v konkrétním odvětví, například rozpoznávání obličeje, pro které je primární co nejvýkonnější neuronová síť. Je to způsobeno tím, že učení neuronové sítě je, výpočetně, velmi náročné a často se proto využívají například grafické karty, které jsou pro podobné výpočty uzpůsobeny. Implementací pro testování rozumíme takový přístup, kdy se vytvoří program, ve kterém se testují různé struktury a typy neuronových sítí z toho důvodu, aby se rozhodlo, který typ sítě je pro danou problematiku nejvhodnější. Proto je dobré, aby takový program umožňoval snadnou změnu neuronové sítě s minimálním zásahem do kódu.

Tato práce je primárně zaměřena na druhou formu implementace (testování), konkrétně se jedná o implementaci NS v Objektově Orientovaném Programování (OOP) jazyce. Cílem je analyzovat neuronové sítě a možnosti OOP, navrhnout model, který by využíval všechny výhody, vyplývající z tohoto přístupu,

a implementovat jej ve formě aplikace, pomocí které by se tento přístup otestoval. Jako jazyk pro tuto implementaci byla zvolena Java – tato knihovna by se nazývala NeuralUHK.

Téma práce bylo vybráno na základě zkušeností, získaných v rámci projektu zaměřeného na klasifikaci léčiv. Cílem projektu bylo ověřit možnosti použití různých typů NS k určení účinnosti jednotlivých léků, na základě jejich složení a dalších vlastností. Dále bylo třeba zhodnotit, který typ neuronové sítě bude nejvhodnější. Průběh vývoje požadované aplikace a rešeršní činnost vedli k závěru, že je třeba navrhnout implementační přístup, který by umožňoval snadnou tvorbu takových sítí (včetně různých zapojení apod.) a změnu jejich kódu. Vzhledem k přednostem objektově orientovaného přístupu - tedy znovu použitelnosti kódu, snadné úpravě a modularitě, a zkušenostem získaným s implementací tohoto paradigmatu v jazyce Java jsem se rozhodl navrhnout model, který by tento přístup využíval.

Práce je členěna následovně: V první části práce jsou teoreticky popsány neuronové sítě z hlediska biologie a matematiky. Je zde zjednodušeně popsáno, jak fungují neurony v lidském mozku, jak je tato myšlenka převedena do matematického modelu, jaké jsou vlastnosti neuronových sítí a proč jsou využívány. V této části se také nachází konkrétní popis implementace NS v oboru biomedicína, konkrétně klasifikace léčiv. V druhé části práce je popsáno objektově orientované programování, jeho základní myšlenka, jeho pravidla, principy, jaké má výhody a nevýhody a jeho konkrétní použití v jazyce Java – jazyk, ve kterém bude testovací aplikace implementována. V další kapitole jsou popsány již existující řešení ve formě několika přístupů k implementaci NS a příklady některých existujících knihoven. Následující, čtvrtá, část se týká navrženého řešení. Zde je popsaná hlavní myšlenka celé práce, tj. nachází se zde návrh objektově orientovaného modelu, který by měl využívat poznatky, získané v teoretické analýze problematiky (předchozí kapitoly). Návrh je popsán jak teoreticky, tak pomocí UML (unified modeling language) a jiných diagramů. V páté kapitole je rozebrána implementace tohoto modelu v jazyce Java, jsou v ní popsány komponenty programu a jeho plánované využití.

Na závěr jsou rozebrány a ohodnoceny experimenty, které byly na síti provedeny v rámci vybraného oboru (klasifikace léčiv), a je slovně shrnut výsledek celé práce.

2 Cíl

Cílem této práce je analyzovat problematiku neuronových sít a objektově orientovaného programování a navrhnout výsledný model, který bude spojovat myšlenku neuronových sítí s výhodami plynoucími s využití objektově orientovaného programování. Výsledný model je třeba poté implementovat ve formě knihovny NeuralUHK, otestovat na vybraných datech a zhodnotit její účinnost. Primárním účelem knihovny, vytvořené v rámci této práce, je testování různých typů a nastavení neuronových sítí pro aplikaci v oblasti biomedicíny, konkrétně klasifikace léčiv. Z tohoto důvodu je nutné ji otestovat na příslušných datech.

Hlavním důvodem tvorby nového vlastního řešení jsou velmi specifické požadavky tohoto oboru – datové sety se obvykle vyznačují velmi malým množstvím vzorků (často řádově méně než je vstupních parametrů). Je potřeba do knihovny dodávat specifické funkcionality, které mohou být součástí i stěžejních tříd. Z těchto důvodů bylo nutné navrhnout vlastní knihovnu, která by se vyznačovala vysokou úrovní upravitelnosti, z důvodu flexibilnosti funkční části kódu. Jedním z hlavních důvodů, proč není možné použít již vytvořené modely je funkční problém, kdy je třeba oddělit samotné propojení a strukturu neuronové sítě od vnitřní funkčnosti neuronu – zpracování signálu. Cílem samotného návrhu je tedy model, reprezentující samotnou neuronovou síť se všemi jejími entitami tak, aby byl snadno rozšiřitelný a odděloval samotnou strukturu a propojení sítě od výpočtů prováděných uvnitř neuronu. Výsledkem by měla být knihovna, která by implementovala navržený model a sloužila pro výzkum v oblasti biomedicíny.

3 Neuronové sítě

Tato kapitola se věnuje neuronovým sítím jako celku. Nejprve jsou zde popsány biologické neuronové sítě (BNS), které představují hlavní část centrálního nervového systému – mozku. Lidský mozek je nesmírně komplexní výtvor a v současnosti se jedná pravděpodobně o jeden z nejsložitějších objektů ve známém vesmíru. Dovede se učit, adaptovat na nové situace, vyvolávat emoce a mnoho dalších věcí. Mozek není ještě kompletně prozkoumaný a některé jeho procesy nejsou zcela zmapovány, přesto, že se výzkumu tohoto orgánu věnuje spousta úsilí a času. Důležitým faktem je, že jeho komplexnost není způsobena složitostí neuronů, jednotlivých buněk, které ho tvoří. Neurony jsou naopak relativně jednoduché buňky, jejichž primárním úkolem je vysílat impulsy. Schopnost mozku, řešit složité problémy, je docílena množstvím neuronů a toho, kolik jednotlivých propojení mezi sebou tyto neurony navzájem mají.

Není tedy divu, že už před 80 lety vznikly první pokusy neuron (a první neuronovou síť) modelovat uměle. První neuronové sítě byly primitivní a zvládaly řešit jen jednoduché úlohy. Časem, jak se vyvíjela výpočetní technologie a algoritmy, se tyto sítě stávaly více použitelnými a v současnosti se již používají v mnoha odvětvích lidské činnosti. Po BNS jsou v další podkapitole popsány umělé neuronové sítě, které se těmi biologickými inspirují.

Umělé neuronové sítě jsou sice výjimečně modelovány jako propojení fyzických jednotek, kde každý neuron odpovídá jedné, ale v drtivé většině se jedná o počítačový model, kde neuron představuje spíše shluk matematických funkcí, než entitu. Vzhledem k zaměření práce jsou v této části popsány pouze počítačové modely umělých neuronových sítí a nikoliv ty fyzické.

3.1 Biologické Neuronové Sítě

Tato kapitola je zaměřena na Biologické Neuronové Sítě (BNS). Umělé Neuronové Sítě se z velké části inspirují biologickými. Pro pochopení celé problematiky je vhodné BNS alespoň obecně popsat. V této části práce se nachází výčet principů, na kterých jsou BNS založeny, seznam jednotlivých komponent a funkcí, které v rámci systému plní. Pro replikaci funkčnosti neuronových sítí

v počítačovém programu není třeba znát kompletní biologickou funkčnost samotného neuronu. Stačí abstrahovat (viz. 4.1.2) jeho základní funkčnost a princip a převést je do funkčního modelu. Tento popis je pro potřeby diplomové práce méně obsáhlý a je zaměřen spíše na popsání základních funkcí, než na kompletní popis buněk a dalších funkcí. Více informací o této problematice se nachází například zde [42].

3.1.1 Úvod

Všechny živé organismy, které mají schopnost se adaptovat na své prostředí, potřebují určitou formu kontrolní jednotky, která je schopna se učit. Lidé a vysoce vyvinutá zvířata používají velmi komplexní síť specializovaných neuronů, které jim adaptibilitu na prostředí umožňují[6]. Tato síť se nachází v jejich mozku a bývá označována pojmem Biologická Neuronová Síť (BNS). BNS je soustava neuronů-buněk (u lidí je jich až 10^{11} [7][6]), navzájem propojených pomocí synapsí a axiomů, které spolu navzájem interagují pomocí zasílání elektrochemických signálů. Každý neuron je propojen až s 10 000 dalšími neurony a samotná síť je založená na binárním principu (viz dále). Její jednotlivé komponenty jsou velmi jednoduché a komplexnost těchto sítí je zapříčiněna množstvím neuronů, které jsou její součástí a díky nim je, i přes zdánlivou jednoduchost, schopna řešit velmi složité problémy[8].

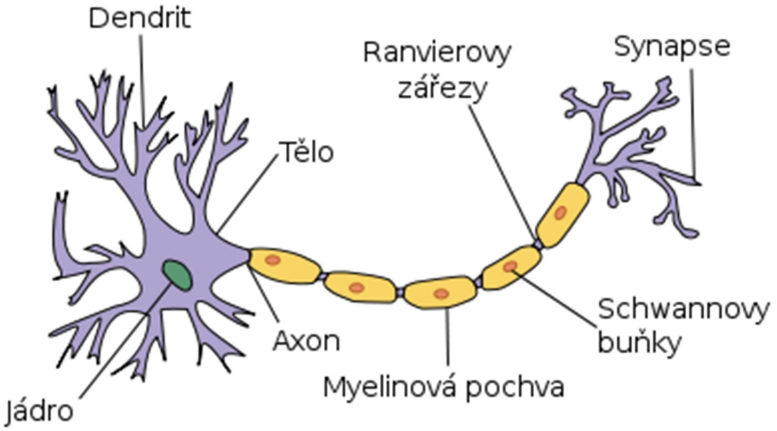
BNS jsou založeny na principu šíření signálů skrz jednotlivě propojené neurony. Signál se dostane na vstupní neurony, které ho dále distribuují do dalších synapsemi propojených neuronů, které je opět posílají k dalším neuronům. Signál (neboli impuls) prochází napříč celou sítí a vytváří určitou cestu skrz tuto síť. Podoba této cesty ovlivňuje výsledné zpracování vstupních hodnot. Tento signál se šíří binárně, neuron impuls vyšle nebo nevyšle, existují tedy jen dva možné výsledky v rámci jednoho neuronu. O tom, jestli je impuls vyslán rozhoduje neuron na základě jeho hraniční hodnoty – podle množství vstupní signálů.

Obrázek 1 - ilustrace biologické neuronové sítě [43].

V následujících podkapitolách jsou popsány hlavní komponenty - neuron a synapse. U neuronu jsou popsány dendrit a axiom, které jsou jeho součástí a reprezentují vstup a výstup z neuronu.

3.1.2 Neuron

„Hlavní funkcí nervového systému je řídit organizmus. Základem této funkce je schopnost systému zpracovávat informace. Informace se v NS přenášejí ve formě změn membránového potenciálu nervových buněk, neuronů.“[10]



Obrázek 2 - struktura biologického neuronu (přeloženo z [44])

Neuron je elektricky drážditelná buňka, která zpracovává informace skrz elektrické a chemické signály. Tyto neurony představují základní jednotky nervového systému (a BNS), které komunikují mezi sebou navzájem a s rozlišnými efektorovými orgány. Neuron je buňka, která se specializuje na odesílání a přijímání chemicky přenášených elektrických signálů (viz. Obrázek 2 a[11]).

Aby mohl neuron plnit svojí primární funkci, musí přijímat signály od ostatních buněk a šířit je dál. Vnímaví povrch neuronu (vertebrate) se z toho důvodu skládá z dendritů a axonu, které odpovídají za vnější interakci[9]. Dendrity v rámci neuronu plní úlohu receptoru, přijímají tedy vstupní signály. Většina neuronů má těchto dendritů několik a tím pádem jsou multipolární[11].[11] Dále každý neuron obsahuje axon, pomocí kterého přenáší signál dál do sítě. Jedná se o dlouhý výběžek neuronu, jehož součástí je axonový terminál, na který jsou pomocí synapsí napojeny dendrity sousedních neuronů. Pokud je překročen práh neuronu, signál je vyslán skrz axon do terminálu.

Základní chování většiny těchto buněk, což je přenášení signálů, funguje v několika fázích. Primárně mezi ně patří:

- **Přijetí signálu** – buňka postupně na dendritech přijímá elektrické impulsy
- **Zhodnocení signálů** – signály postupně zvyšují napětí na buňce, a pokud překročí určitý práh, neuron vyšle signál
- **Vyslání signálu** – neuron vyšle elektrický impuls na svůj axonový terminál, který se poté skrz synapse dostane na dendrity připojených neuronů

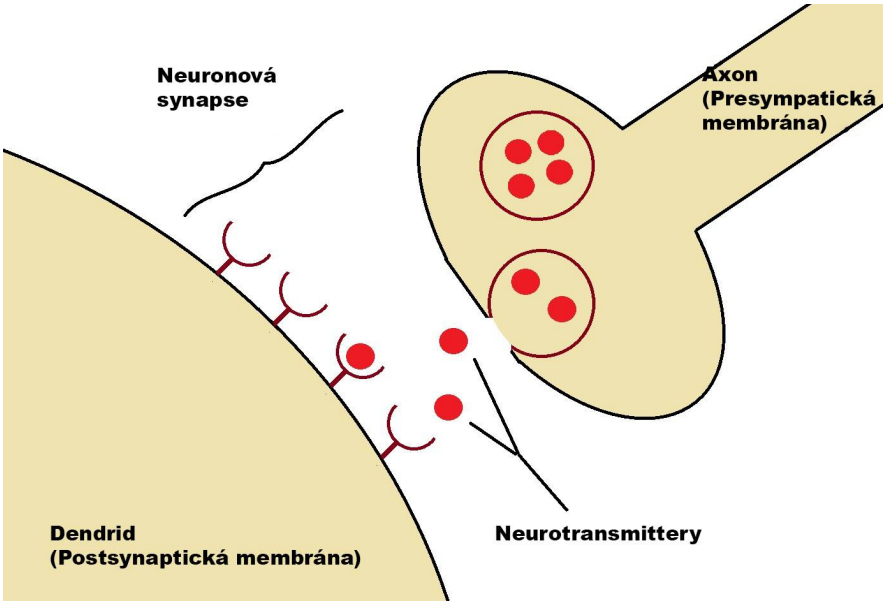
Tento model obecně neplatí pro všechny neurony, ale platí to pro ty, ve kterých se umělé neuronové sítě inspirovaly, je zde proto jako jediný rozebrán.

3.1.3 Synapse

Neurony vytvářejí funkční spojení v místě těsného kontaktu axonu jedné buňky s membránou jiné buňky. Toto spojení se nazývá synapse (lat. těsně se objímat). Na jednom neuronu mohou být stovky, tisíce a na některých až desítky tisíc synapsí[10]. Synapse představuje propojení dvou neuronů, pomocí kterého se

v neuronové síti šíří signál. Synapse je spíše teoretický pojem používaný pro vymezení takového spojení, protože jednotlivé součásti synapse jsou součástí buď zdrojové, nebo cílové buňky.

Toto spojení se skládá ze tří částí, membrány presynaptického terminálu, synaptické štěrbiny a postsynaptické membrány. První část odpovídá axonu neuronu, ze kterého se šíří signál, resp. jeho terminálu. Druhá část je synaptická štěrbina, přes kterou je napojena poslední součást - dendrit neuronu, který signál zdroje přijímá (viz. Obrázek 3 a [10]).



Obrázek 3 - synapse mezi dvěma neurony (přeloženo z [45])

Jedna synapse odpovídá propojení výběžku axonového terminálu ve zdrojovém neuronu a dendritu v neuronu cílovém. Toto propojení umožňuje zdrojovému neuronu vyslat signál všem neuronům, které jsou na něj napojeny.

Synaptické váhy a učení

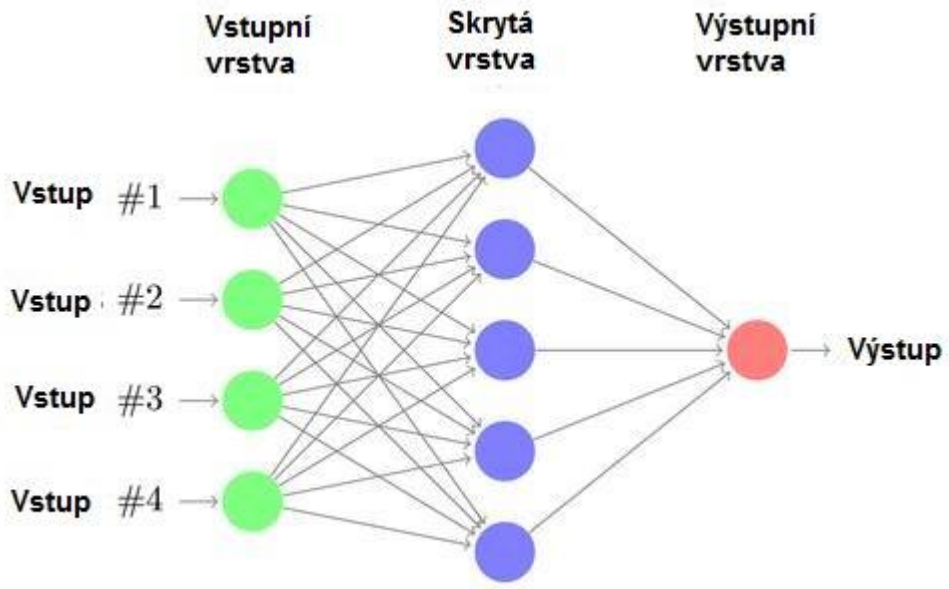
V oboru neurologie je takto označováno ohodnocení spojení dvou uzlů (neuronů). Každý dendrit má v rámci neuronu chemickou propustnost, která by se dala vyložit jako „důležitost signálu“. Díky prahové hodnotě (viz. zhodnocení signálu v neuronu), kterou neuron využívá k rozhodnutí, jestli vyslat signál, je možné pomocí této propustnosti ovlivnit, jakým způsobem budou různá připojení překročení tohoto prahu ovlivňovat.

Tato propustnost je měnitelná a díky tomu je možné naučit neuron reagovat na stejné situace, signály ze stejných neuronů, jiným způsobem. Toto je hlavní princip učení v biologických neuronových sítích, který se využívá i v umělých neuronových sítích.

3.2 Umělé Neuronové Sítě

„Umělé neuronové sítě mohou být nejpřesněji charakterizovány jako ‚výpočetní modely' s vlastnostmi adaptace a učení, využívanými ke generalizaci, shlukování, organizaci dat a operacím založeným na paralelním zpracování." [15]

Umělé Neuronové Sítě, neboli Artificial Neural Networks – ANN, jsou velmi zjednodušené modely struktury biologických neuronových sítí (viz. 3.1), které se používají k aproximaci funkcí, obsahujících velké množství obvykle neznámých parametrů. ANN se skládá z množství navzájem propojených výpočetních jednotek – neuronů (viz. Obrázek 4), jejichž propojení je ohodnoceno váhami, které je možno měnit. Toto dělá ANN adaptivními na vstupy a umožňuje jim se učit [12].



Obrázek 4 – Neuronová síť – typ Feedforward (přeloženo z [46])

ANN představují abstrakci biologických neuronových sítí, přejímají některé jejich charakteristiky a vlastnosti a odstiňují od nich samotnou biologickou implementaci v neuronu. Model ANN abstrahuje základní funkčnost BNN, což je

zpracování informací pomocí šíření impulsů (informací) napříč sítí skrz navzájem propojené neurony - samostatně se rozhodující, jednoduché, řídící jednotky - a možnost se učit pomocí úpravy vah na jednotlivých propojeních, od jejich složité chemické struktury a procesů, které v nich probíhají. Využívá jen základní myšlenku řešení komplexních úloh pomocí zapojení velkého množství velmi jednoduchých řídících jednotek a adaptace skrz „váhy“, ale její implementaci v podobě chemických procesů již ne.

V následujících podkapitolách je nejdříve popsána historie ANN v několika hlavních bodech, následně je rozebrán „umělý neuron“ a funkce neuronových sítí, poté jejich druhy, další podkapitola se věnuje technikám učení neuronových sítí a nakonec je zde zhodnoceno jejich použití v různých odvětvích lidské činnosti.

3.2.1 Historie

Počátek vývoje umělých neuronových sítí odpovídá době, kdy se poprvé člověk pokusil modelovat neuron. Prvním krokem byl rok 1943, kdy Warren McCulloh a Walter Pitts napsali článek o tom, jak mohou neurony fungovat a modelovali jednoduchou neuronovou síť pomocí elektrických obvodů[17][16][14].

V roce 1949 Donald Hebb napsal knihu *Organizace Chování*, ve které poukázal na fakt, že neurální cesty (viz. 3.1) zesilují pokaždé, když jsou použity. Tento koncept je zásadní pro způsob, kterým se lidé učí. Argumentoval, že pokud dva neurony vyšlou impuls ve stejný čas, propojení mezi nimi se posílí[16][13]. Dále byl v roce 1954 Marvinem Minskym vyvinut stroj, který se učil tak, že uměl upravovat sílu jednotlivých spojení, ale byl to Rosenblatt, který v roce 1958 navrhl model perceptronu, který uměl upravit váhy na spojení podle pravidel učení[12].

Na konci 50. let (1959) Widrow a jeho skupina navrhla modely ADALINE a MANDALINE pro zpracování dat pomocí neuronových sítí. ADALINE byl vyvinut k rozpoznávání binárních vzorců tak, že při čtení z telefoní linky byl schopen predikovat následující bit. MANDALINE představoval první neuronovou síť použitou na řešení reálného problému, použití adaptivního filtru k eliminaci ozvěn na telefonní lince. Tento systém byl v roce 2000 stále v komerčním použití. Dále v roce 1962 navrhli ještě LMS (least mean square) algoritmus k úpravě vah na jednotlivých spojeních[12][16].

V roce 1972 Kohonen a Anderson, nezávisle na sobě, navrhli podobnou neuronovou síť. Oba dva použili maticovou matematiku k popisu jejich nápadů, kterým bylo pole analogových ADELINE obvodů. První vícevrstvá síť byla vytvořena v roce 1975 - síť bez dohledu[16].

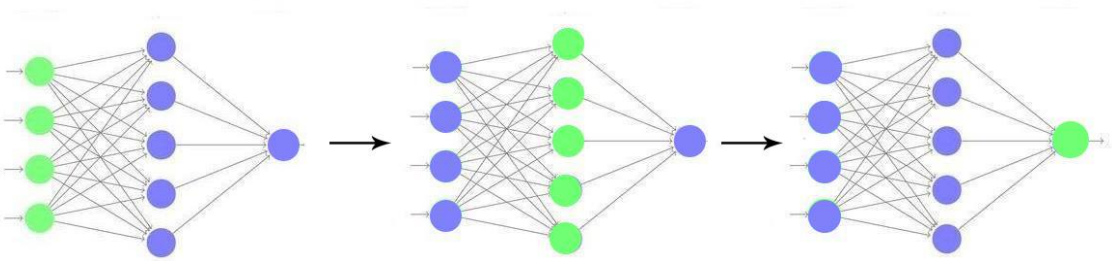
Rok 1982 byl rok, ve kterém několik událostí způsobilo znovuoživení zájmu o tuto technologii. Prvním je energická analýza zpětné vazby neuronových sítí od Johna Hopfielda vydaná v roce 1982. Jeho přístup byl nejen o zjednodušeném modelu mozku, ale také o vytvoření užitečných zařízení. Analýza ukázala, že existence rovnovážného stavu ve zpětnovazební síti dokazuje, že síť má symetrické váhy a úprava jejího stavu je prováděna asynchronně. Ve stejném roce Reailly a Cooper použili hybridní síť s několika vrstvami, přičemž každá vrstva používá jinou strategii řešení problémů[12][16].

V roce 1986 Rumelhart ukázal, že je možné upravit váhy vícevrstvé neuronové sítě symetricky pro naučení mapování setů párů vstupů a výstupů. Toto učící pravidlo je nazýváno *generalizované delta pravidlo* nebo také učení pomocí zpětné propagace. Základní myšlenkou tohoto pravidla je zpětná propagace chyby skrz síť – je vypočítána chyba očekávaného a skutečného výstupu NS a ta je distribuována zpětně do sítě[13][12][16].

V současnosti jsou neuronové sítě používány v širokém spektru odvětví. Toho je převážně dosaženo díky výzkumu lidí zmíněných v této kapitole. Historie je zde popsána jen pro upřesnění, více znalostí o historii vývoje ANN je možné získat z použitých zdrojů - [12][10][13][16][17].

3.2.2 Funkce

Umělá Neuronová Síť představuje množinu neuronů, které jsou mezi sebou určitým způsobem propojeny. Tyto neurony jsou uspořádány do vrstev, které reprezentují šíření signálu skrz celou síť. Každá neuronová síť má tyto vrstvy alespoň dvě – vstup a výstup. Signál se nejprve dostane na vstupní vrstvu, poté na vrstvu, která je na ní napojena a postupně prochází sítí, dokud se nedostane na výstup (viz. Obrázek 5).



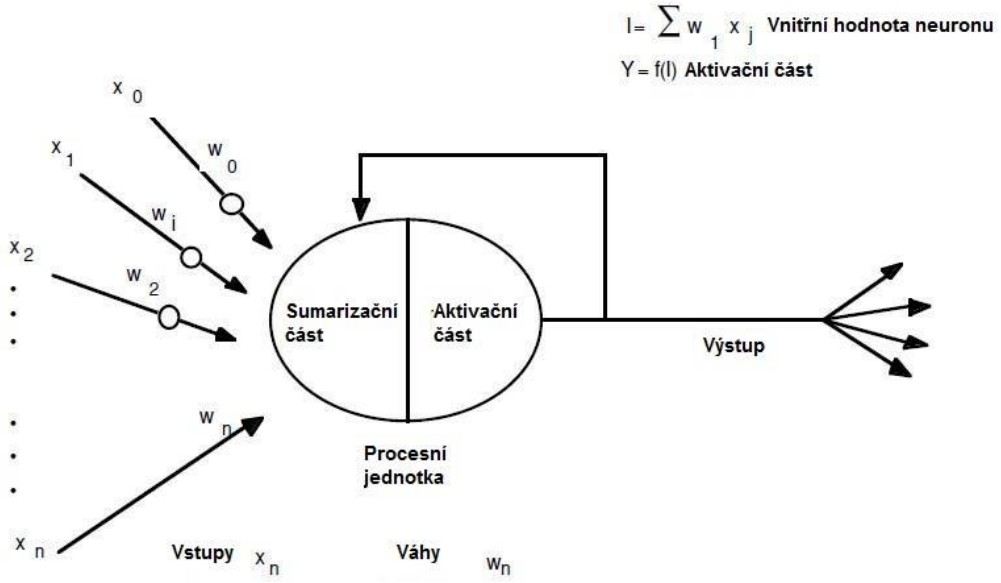
Obrázek 5 - Šíření signálu (zelená barva) skrz NS (upraveno z [46])

Vždy, když se na vrstvu dostane signál, všechny neurony na ní se aktivují (viz. 3.2.3 a Obrázek 5) a pomocí spojení s neurony předchozí vrstvy vypočtou vlastní výstup. Síla signálu je na jednotlivých spojeních upravována pomocích hodnot jejich vah. Tyto váhy určují, jak se bude intenzita signálu měnit při prostupování sítí a na jejich nastavení závisí výsledná hodnota na výstupu. Úpravou těchto vah je možné neuronové sítě učit (viz. 3.2.5).

3.2.3 Umělý „Neuron“

Neuron je základní výpočetní jednotkou umělých neuronových sítí. Jeho hlavní a jedinou funkcí je výpočet jeho výstupu, tedy jako hodnotu pošle dál do sítě, na základě všech jeho vstupů. Jedná se o model vytvořený na základě abstrakce biologických neuronů – přejímá jejich obecné funkce, ale abstrahuje konkrétní implementaci.

Jedno z hlavních témat v historii neuronových sítí souvisí s modelováním samotné neuronové buňky. [10]Postupným vývojem v této oblasti bylo dosaženo obecného modelu neuronu[10]. Ten se skládá ze sumarizační části a výstupu. Sumarizační část obsahuje několik vstupů, kde každý vstup představuje propojení se sousedním neuronem (konkrétně s jeho výstupem) a váhy tohoto propojení. Dále je její součástí funkce, která ze všech vstupů a jejich vah vypočítá hodnotu neuronu, jež je předána výstupní části. Ta obsahuje aktivační funkci, které je tato hodnota předána, ta následně vypočte výstupní hodnotu neuronu, která je předána na poslední část – výstup. Tato část pak představuje vstup pro další neurony v síti, které jsou na tento neuron napojeny (viz. Obrázek 6 [12][10]).



Obrázek 6 - model umělého neuronu (přeloženo z [47])

Vstup – jeden vstup do neuronu se skládá z výstupu sousedního neuronu a jeho přidružených váh. Tyto váhy představují významnost vstupu a ovlivňují, jak moc se tento vstup promítne do celkové vnitřní hodnoty neuronu. Váha obvykle bývá jedna, nicméně v několika speciálních případech jich může být i několik – například MF ART Map (viz. [29]).

Vnitřní funkce neuronu – tato funkce vypočítá vnitřní hodnotu neuronu na základě všech jeho vstupů. Obvykle se jedná o funkci:

$$\sum x_i w_i$$

kde x představuje vstupní hodnotu sousedního neuronu a w představuje jeho váhu. Jak bylo zmíněno, ne všechny neurony obsahují jednu váhu a tato funkce se může lišit, ale toto je nejpoužívanější varianta[18][17].

Aktivační funkce – jedná se o funkci, která z vnitřní funkce neuronu vypočítá jeho výstupní hodnotu. Těchto funkcí je několik, nejčastější jsou [10][17]:

- **Sigmoid**

$$\boldsymbol{S(t) = \frac{1}{1 + e^{-t}}}$$

- **Lineární**

$$\boldsymbol{L(t) = t}$$

- **Hyperbolický tangent**

$$\boldsymbol{tanhx = \frac{1 - e^{-2x}}{1 + e^{-2x}}}$$

- **Těžký limitní práh**

$$\boldsymbol{f(x) = \begin{cases} 0\ pokud\ x < 0 \\ 1\ pokud\ x \geq 1 \end{cases}}$$

Výstup – výstupní hodnota neuronu. V rámci neuronu není nijak dále upravována a slouží napojeným neuronům jako jejich vstupní hodnota. Na tuto část jsou napojeny následující neurony a je součástí jejich vstupu.

Základní funkce neuronu se nazývá *„aktivace“*. Tato aktivace v biologickém neuronu odpovídá stavu, kdy napětí na membráně překročí určitý práh a neuron v tu chvíli vyšle signál (impuls). V počítačovém modelu umělého neuronu není tento proces implementován přesně z důvodu výpočetní náročnosti. Aktivace je zde vyvolána vždy v určitém čase a probíhá následovně:

- Pro každý vstup (tj. výstup z předchozího neuronu) je vypočítána reálná hodnota pomocí přiřazených vah
- Je vypočítána vnitřní hodnota neuronu a předána výstupní části
- Pomocí aktivační funkce je z této hodnoty vypočítán výstup neuronu
- Ten je nastaven jako výstupní hodnota, která slouží jako vstup napojeným neuronům

Umělý neuron tedy přesně nekopíruje chování toho buněčného, nicméně se mu blíží.

3.2.4 Druhy

Za více jak 70 let vývoje neuronových sítí (viz. 3.2.1) vznikla celá řada druhů neuronových sítí. Ty se liší vnitřní strukturou neuronu, jejich zapojením, učícím algoritmem a dalšími vlastnostmi. Každý druh je specializovaný na jiný typ problematiky (viz. 3.2.6 a tabulka). Vzhledem k množství těchto sítí a zaměření

této práce není nutné popisovat všechny typy, ale budou zde vyjmenovány alespoň některé:

Feedforward, Backpropagation Network (dopředná neuronová síť se zpětnou propagací chyby) – Jeden z nejstarších, nejrozšířenějších a pravděpodobně nejjednodušších typů, který byl vyvinut v 70. letech několika autory nezávisle na sobě. Představuje složení struktury feedforward a učícího algoritmu backpropagation (viz. 3.2.5). Jedná se o velmi efektivní model používaný pro komplexní, více vrstvé sítě, jehož největší síla je v nelineárních řešeních pro špatně definované problémy. Tato síť se skládá z několika vrstev, v nichž jsou neurony napojeny jen v jednom směru – od vstupu k výstupu (viz. Obrázek 5). Zpracování signálu probíhá ve směru vstup -> výstup a učení probíhá ve směru opačném (výstup -> vstup). Tyto sítě jsou vhodné pro běžné klasifikační problémy, hledání vzorců apod. Vzhledem k absenci jakékoliv formy paměti nejsou vhodné pro sekvenční data, tedy taková, u kterých záleží na předchozích vstupech[13][15].

Rekurentní neuronová síť- jedná se o typ sítě, ve které nejsou neurony zapojeny ve vrstvách jen v jednom směru (vstup -> výstup), ale mohou být napojeny i na vrstvu, která se nachází před nimi, nebo sami na sebe. Toto chování umožňuje získávat zpětnou vazbu z předchozích cyklů ANN, výsledek pro jednu množinu vstupních dat je ovlivněn i předchozími daty a umožňuje predikovat výsledky na základě předchozích hodnot – například vývoj na trhu[15][17]. Tento typ je, díky vlivu předchozích výstupů, vhodný pro zpracování sekvenčních dat. Nicméně je méně efektivní, než vytvoření tzv. LSTM (long short-term memory – krátkodobá pamět) bloku a proto se, například pro zpracování hlasu, kde je zapotřebí delší paměť, často dává přednost právě jim[27].

MF-ART map – neuronová síť používající tzv. učení bez dozoru (bez učitele). Tato síť v sobě spojuje vlastnosti ART (Adaptive Resonance Theory) a Fuzzy systémů k řešení klasifikačních úloh. Cílem tohoto spojení je využít klasifikační schopnosti neuronových sítí ART a větší míry informací ve formě příslušnosti vstupu k fuzzy třídám[29][30]. Srdcem ART (a tím pádem i MF ART Map) je dvojice

vysoce propojených vrstev – spodní a vrchní rezonanční vrstvy, které se nachází mezi vstupem a výstupem. Každý vzorec vstupující do spodní rezonanční vrstvy vyvolá odeslání očekávaného vzorce z horní rezonanční vrstvy k ovlivnění následujícího vstupu. Toto vytváří rezonanci mezi spodní a vrchní vrstvou za účelem umožnění síti přijmout nový vzorec[13].

V následující tabulce (viz. Tabulka 1) se nachází ještě výčet dalších typů ANN a jejich využití.

Tabulka 1 - druhy a použití neuronových sítí, přeloženo z [13]

Typ	Sítě	Využití
Predikce	Back-propagation Delta Bar Delta Extended Delta Bar Delta Higher order Neural Networks Directed Random Search Self Organizanizing Map into Back-propagation	Použití vstupních hodnot k predikci výstupu (například vývoj počasí, identifikace lidí s rizikem rakoviny apod.)
Klasifikace	Learning vector quantization Counter-propagation Propabilistic Neural Network	Použít vstupní hodnoty ke klasifikaci (je vstup písmeno A, je datový soubor video letadla atd.)
Asociace dat	Hopfield Boltzmann Machine Hamming network Bidirectional associative memory Spatlo-temporal pattern recognition	Podobné jako klasifikace, ale mohou určit i kdy se vyskytuje v datech chyba (nejen určit, které znaky byly oskenovány, ale i že scanner nefungoval)
Konceptualizace dat	Adaptive resonance Network Self organizing map	Analyzovat vstupy tak, že mohou být určeny vztahy mezi skupinami (vybrat z databáze taková jména, která si pravděpodobně koupí vybraný produkt)
Filtrovaní dat	Recirculation	Vyladění vstupního signálu (odstranit hluk z telefonního signálu)

3.2.5 Učení

„Úprava tendencí chování na základě získaných zkušeností“ [19].

Vlastnost, která získala neuronovým sítím největším pozornost, je právě schopnost se učit, neboli trénovat. Tato schopnost jim umožňuje se adaptovat na nově vzniklé problémy a dává výhodu oproti běžným programům, které je většinou nutné upravit zdrojový kód. Trénováním NS se zabývá celé jedno odvětví strojového učení a existují dvě hlavní paradigmata, která se inspirují učením lidského mozku – s učitelem a bez učitele.

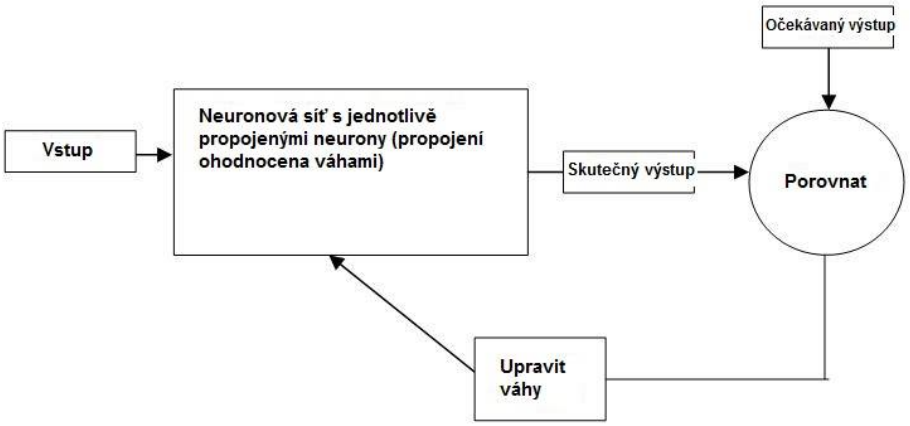
Neuronová síť v sobě v podstatě obsahuje funkci $\boldsymbol{f}$, která je definována počtem neuronů, jejich propojením a hodnotou vah těchto konexí. Pokud existuje nějaký úkol, k jehož řešení vede skupina funkcí $\boldsymbol{F}$, pak učení odpovídá aproximaci funkce $\boldsymbol{f}$ tak, aby byla nalezena funkce $\boldsymbol{f}^* \in \boldsymbol{F}$, která je schopna problém vyřešit. Tato aproximace probíhá pomocí úprav vah na jednotlivých propojeních mezi všemi neurony pomocí vybraného neurálního algoritmu. Zjednodušeně řečeno se jedná o úpravu neuronové sítě takovým způsobem, aby pro daný vstup bylo dosaženo, alespoň přibližného, očekávaného výstupu[15][13].

Existují dva základní typy učení (paradigmata) neuronových sítí. Jedná se o učení s učitelem a bez učitele. Tento přístup funguje i v lidské mysli. Člověk se může učit pomocí pozorování (bez učitele) bez nějakého cílového vzoru nebo může mít učitele, který mu ukáže vzor, který má sledovat. Z hlediska ANN může například učitel představovat množinu dat nebo pozorovatele, který hodnotí výsledky sítě. Bez učitele se musí systém sám organizovat pomocí nějakých vnitřních kritérii zakomponovaných do samotné sítě. Rozdíl těchto paradigmat převážně závisí na relevanci cílového vzoru a liší se problém od problému[13][17][20].

3.2.5.1 Učení s učitelem

Většina řešení v oblasti neuronových sítí byla trénována (učena) pomocí této metody. Při tomto typu učení jsou poskytnuty jak vstupy, tak výstupy. Tato data mohou být poskytnuta vnějším učitelem, nebo systémem, který obsahuje

samotnou síť (tzv. self-supervising – systém, který je sám sobě učitelem). Síť potom zpracuje vstupy a porovná jí dosažené výstupy s těmi očekávanými (poskytnutými). Chyby, vypočítané z tohoto porovnání, jsou poté propagovány zpět do systému, způsobující úpravu vah v neuronové síti (viz. Obrázek 7). Cílem těchto učících algoritmů je postupně minimalizovat chybu na všech procesních jednotkách (neuronech). Tohoto globální snížení chybovosti je docíleno pomocí soustavné úpravy vah na všech spojeních, dokud není dosaženo přijatelné přesnosti[20][13][21][15].

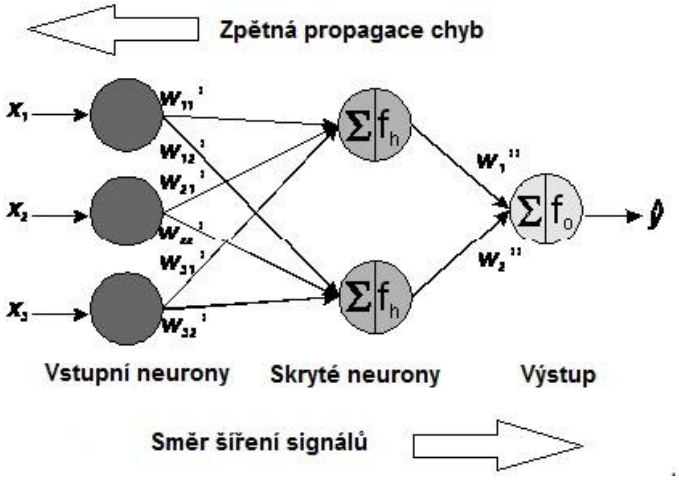


Obrázek 7 - Princip učení s učitelem (přeloženo z [48])

S použitím učení s učitelem musí být síť nejprve natrénována, než bude použitelná. Ve většině aplikací je k tomu třeba použít skutečná data a tento proces je časově velmi náročný. Bez dostatečné výpočetní kapacity může trvat až několik týdnů. Učení se považuje za kompletní, až v momentě kdy síť dosáhne předdefinované statistické přesnosti a produkuje vyžadované výstupy pro zadanou sekvenci vstupů. Po tomto procesu se obvykle váhy na NS zmrazí, nicméně některé sítě pokračují v neustálém učení (i když v menším měřítku), v operačním stavu. To pomáhá k adaptaci na měnící se podmínky (viz. Obrázek 7 a [13]).

Back-propagation - Mezi hlavní algoritmy s učitelem patří algoritmus Back-propagation (zpětná propagace), neboli backward-error propagation (zpětná propagace chyby). Tento algoritmus byl poprvé navržen v 60. letech (viz. 3.2.1) a vzhledem k jeho účinnosti je stále používán a zdokonalován. Hlavním principem

tohoto algoritmu je, že chyby ve skrytých vrstvách jsou určeny pomocí zpětné propagace chyby z neuronů na výstupní vrstvě – šíří se opačně proti směru šíření signálu (viz Obrázek 8). Tato metoda může být také považována za zobecnění delta pravidla pro nelineární aktivační funkce a vícevrstvé sítě[15]. Tento algoritmus využívá metody klesajícího gradientu – hledá globální minimum chybové funkce.



Obrázek 8 - Směr šíření signálu a propagace a chyby u back-propagation (přeloženo z [49])

Pro potřeby této práce je zde tento algoritmus popsán pouze obecně, nicméně je detailněji rozebrán v použitých zdrojích ([15][25][26]). Neuronová síť se v tomto popisu skládá z vrstev $L_1 \rightarrow L_n$ kde první je vstupní a poslední výstupní vrstva. Jedná se o popis pro jednu množinu dat:

1. Na vstup nastavit nový datový set
2. Provést aktivace pro všechny neurony na vrstvách $L_2 \rightarrow L_n$ (od vstupu k výstupu)
3. Pro všechny neurony na vrstvách $L_n \rightarrow L_2$ vypočítat chybu (v opačném pořadí, než v kroku 2)
4. Spočítat dočasné váhy Δw pro všechny neurony na vrstvách $L_n \rightarrow L_2$ pomocí chyb z předchozího kroku
5. Upravit neuronům váhy pomocí spočítaných Δw

3.2.5.2 Bez učitele

Toto učení probíhá bez znalosti výstupů dat, která neuronová síť obdrží, a jedná se o přístup, při němž se síť učí jen na základě vlastností vstupních dat. Tento přístup je velkým příslibem do budoucna, který by znamenal, že počítač se bude schopen učit sám bez jakéhokoliv vnějšího zásahu. Na rozdíl od učení s dohledem, zde není nijak zadán set kategorii, do kterých jsou rozčleňovány vstupy, a síť si musí vytvořit vlastní systém kategorizace. V současnosti ([13]) tento přístup používají jen sítě, známé jako samo-organizované mapy, které nejsou moc prakticky využívané. Převážně se jedná o akademickou literaturu. Nicméně v některých specifických případech jsou velmi užitečné, což prokazuje užitečnost jejich výzkumu[13][15].

3.2.6 Použití

Neuronové sítě v této dekádě (2010+) zaznamenaly prudký nárůst použití převážně díky pokroku ve vývoji hardwaru. Dříve nebylo možné ve smysluplné době natrénovat dostatečně komplexní neuronovou síť, která by umožňovala řešit náročné úlohy – jednoduché sítě nejsou na většinu problémů použitelné.

Vzhledem k množství různých druhů neuronových sítí (viz. 3.2.4), které se používají, je jejich využití značně rozsáhlé a každý typ se hodí na různé oblasti lidské činnosti. Zde jsou některé z oblastí, v nichž jsou neuronové sítě využívané:

- **Rozpoznání písma/znaků** – využití v programech využívajících ruční písmo nebo u skenování a následném rozpoznávání textu.
- **Rozpoznávání obrazu** – díky schopnosti rozeznávat a klasifikovat vzory je neuronová síť velmi účinná při klasifikaci obrazových souborů (např. určit, jestli se na fotce nachází slunce apod.)[23].
- **Medicína** – neuronová síť je schopna, pokud je správně natrénována, predikovat vývoj nemoci u pacienta, popřípadě určit jeho náchylnost touto nemocí onemocnět[17].
- **Marketing** – neuronové sítě, které používají metodu učení bez učitele, jsou účinné pro svou schopnost najít v datech nějaký vzor. Proto se

dají například použít pro cílený marketing na základě charakteristik zákazníka[17].

- **Rozpoznání hlasu** – ANN mající nějakou schopnost si pamatovat (například LTMS jednotka/rekurentní ANN) jsou velmi efektivní pro rozpoznání hlasu a používají se například pro ovládání hlasem některých zařízení nebo pro překládání hlasového vstupu do jiného jazyka[24][27].
- **Biomedicínská data -** Jedním z dalších možných využití, díky kterému tato práce vznikla, je aplikace v biomedicíně při klasifikaci léčiv. Konkrétně se jedná o použití ANN na ohodnocení kvality léků na základě jejich složení, vlastností a dalších charakteristik – klasifikace. Cílem je otestovat, jestli je možné pomocí ANN nalézt v těchto charakteristikách nějaké vzorce, díky kterým by se dala určit účinnost daného léku. Výzkum, který vedl k napsání této práce, se v současnosti věnuje testování různých struktur a napojení neuronových sítí a jejich testování na těchto látkách. Hlavním důvodem tohoto zaměření je fakt, že namíchání jednoho přípravku je časově i finančně velmi náročnou záležitostí a pokud by se podařilo najít v těchto charakteristikách nějaký vzorec, znamenalo by to značný pokrok ve výrobě takových léčiv.

4 Objektově Orientované Programování

Objektově Orientované Programování (OOP) je jedním z programovacích paradigmat, které nahlíží na systém jako na množinu objektů, které spolu navzájem interagují. Jedna z hlavních předností objektově orientovaného přístupu k vývoji softwaru je poskytnutí silných a významných konceptů, které jsou tak účinné, že mohou sloužit i mimo jejich původně cílenou oblast – programování. Tyto koncepty (principy), mezi které patří třídy, zasílání zpráv, dědění apod., jsou využitelné v mnohem obecnějším smyslu – navrhování systémů, jejich modelování a celkově pro lepší přemýšlení o systémech[4].

Tento přístup umožňuje rozdělit chod systému na jednotlivé úlohy, za jejichž plnění odpovídají konkrétní objekty. Systém je navržen tak, aby byly jednotlivé části jeho funkčnosti vyčleněny do tříd, které je pak budou poskytovat svému okolí. Výsledkem je program, ve kterém je kód rozdělen mezi jednotlivé objekty, které mezi sebou komunikují a navzájem využívají své služby. Výhody tohoto přístupu jsou následující:

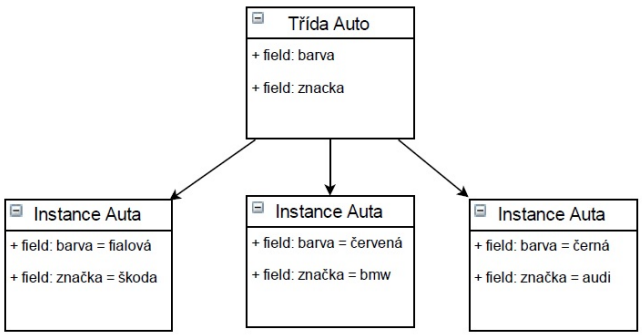
- **Přehlednost kódu** – kód je rozdělen do tříd s úzce definovanou funkčností, to ho odděluje od zbytku programu a činní ho lépe čitelným
- **Modularita** – vzhledem k tomu, že každá část programu tvoří uzavřený celek, je snadné přidat nové funkce/odebrat staré
- **Znovu použitelnost** – možnost použít již nadefinované třídy v jiných projektech, které jsou podobně zaměřené

4.1 Principy

Objektově orientované programování, jak již plyne z názvu, využívá objekty. Ale ne všechny principy a techniky, které jsou spjaté s OOP, jsou přímo podporované ve všech jazycích, které o sobě tvrdí, že jsou objektové. V této kapitole jsou vyjmenovány hlavní principy, patřící do tohoto paradigmatu.

4.1.1 Objekt a třída

Jádrem OOP je objekt, který představuje jakoukoliv entitu, která vystupuje v systému. A naopak, každá entita, která nějak vystupuje v systému, je reprezentována formou objektu. Ten obsahuje atributy, datovou část a metody, procedurální část, a s ostatními objekty komunikuje pomocí zasílání zpráv. Ve většině programovacích jazyků je objekt instancí třídy. Třída v tomto případě představuje šablonu objektů, které mají stejné procedury a atributy, a podle ní jsou tvořeny jednotlivé objekty. Takový objekt tedy odpovídá jedné entitě určitého typu (třídy) v systému, přičemž každý má svůj stav, který je reprezentován hodnotou jeho atributů, a chování, které definují jeho metody. Jako příklad lze uvést třídu Auto, která má dva atributy barvu a značku a z této třídy (šablony) jsou vytvořeny tři instance, které vždy odpovídají jednomu konkrétnímu autu, viz Obrázek 9.

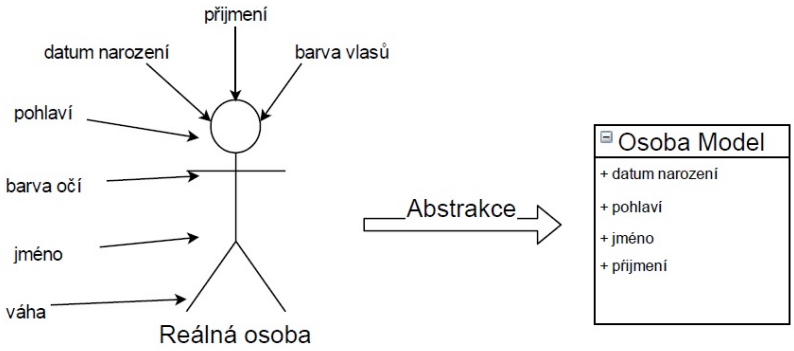


Obrázek 9 - vztah třída – objekt [autor]

4.1.2 Abstrakce

„Abstrakce je mechanismus, který umožňuje reprezentovat komplexní oblast reálného světa pomocí zjednodušeného modelu.“[1] Úkolem abstrakce je vytvořit z reálného objektu model tak, že výsledný model bude obsahovat jen ty vlastnosti, které jsou pro funkčnost modelu relevantní. Pomocí tohoto mechanismu je možné reprezentovat i velmi komplexní objekty, protože většinu vlastností lze odstínit – abstrahovat a z komplexního objektu udělat jednodušší. Například, pokud je úkolem vytvořit model osoby pro potřeby personálního oddělení, provede se analýza objektu (osoby) a do výsledného systému jsou zahrnuty jen ty vlastnosti, které jsou pro systém důležité, tj. datum narození, pohlaví, jméno, příjmení atd., a

ty zahrneme do modelu, zatímco vlastnosti, které pro systém nejsou třeba (barva očí, vlasů, váha) odstíníme a do výsledného modelu nedáme (viz Obrázek 10).



Obrázek 10 - Abstrakce reálné osoby na model [autor]

4.1.3 Zapouzdření

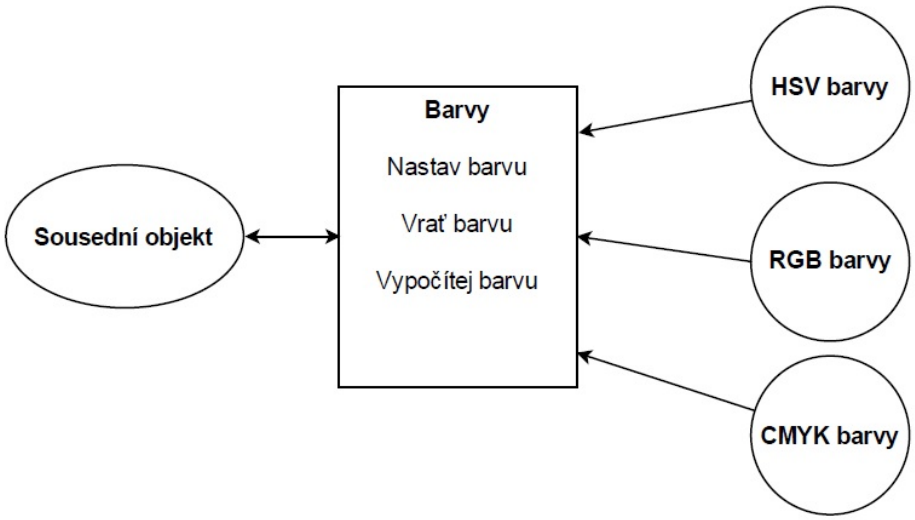
Tento koncept je blízce spojený s modularitou. Zjednodušeně, zapouzdření znamená sloučit metody a atributy do objektu (třídy), vytvořeného při abstrakci, a skrýt jejich implementaci před okolními objekty[2]. Díky této vlastnosti není struktura objektu ani implementace jeho metod viditelná pro zbytek systému. Zapouzdření zajišťuje, že interní detaily objektu jsou skryty před systémem a každý objekt udržuje svou vlastní identitu a stav[1]. Jako příklad se dá uvést třída Automobil, která má pro okolí několik metod (např. nastartuj, zastav, otevři dveře atd.), ale okolní objekty neznají jeho atributy, stav, ani jak jsou jeho metody konkrétně implementovány.

4.1.4 Dědičnost

Tento princip definuje vztah mezi několika třídami. Stanoví, že dvě třídy jsou ve vztahu rodič-potomek, kde potomek dědí všechny vlastnosti (atributy, metody) předka[3]. Jedná se tedy o vztah dvou tříd, kde dědící třída (potomek) obsahuje všechny atributy a metody třídy děděné (předek) a přidává některé své vlastnosti. Jako příklad může být uvedena třída Osoba, která obsahuje atributy jako jméno, příjmení a adresa, od ní mohou dědit třídy Zaměstnanec a Zákazník, které každé mají atributy a metody osoby, ale přidávají některé vlastní.

4.1.5 Polymorfismus

Díky polymorfismu mohou objekty různých tříd odpovídat na stejné zprávy (tj. vykonávat stejné metody), to umožňuje tvorbu generického softwaru, který posiluje možnosti údržby kódu[2]. V podstatě je možno říct, že různé třídy mohou mít definovány společné metody v určitém rozhraní, ale každá bude na metodu reagovat jinak (stejný vstup a výstup, ale odlišný způsob zpracování). Jako příklad může být uvedeno rozhraní Barvy. Pro okolí objektu není důležité, jak objekt tohoto rozhraní implementuje (tj. jak bude probíhat výpočet v daných metodách) vybrané metody, záleží jen na vstupu a výstupu. Toto rozhraní může používat jak RGB model (aditivní míchání), tak CMYK (subtraktivní míchání), a okolí to nijak neovlivní (viz Obrázek 11). Každý model se počítá jinak, má jiné atributy, ale pro okolí vystupuje stejnými metodami.



Obrázek 11 - princip polymorfismu u barev [autor]

4.2 Java

Java je obecně využitelný programovací jazyk, který využívá principů objektového přístupu (viz. 4). Jeho největší předností je jeho platformní nezávislost – kromě Javy pro Android – která znamená, že kompilovaný kód je možné spustit na prakticky jakékoliv platformě, na které jde Java spustit. Pro toto využívá Java Virtual Machine (JVM), což je prostředí, které představuje vrstvu mezi operačním systémem zařízení a samotnou aplikací.

Jedná se o jeden z nejvíce rozšířených programovacích jazyků [5], který za svou popularitu vděčí zejména již zmíněné přenositelnosti a jeho široké využitelnosti v mnoha odvětvích IT.

Tento jazyk má, ostatně jako každý jiný, vlastní implementaci OOP principů (některé ne-zcela naplňuje, viz dále):

- **Polymorfismus** - realizován skrz rozhraní (interface). Třídy, které rozhraní implementují, musí implementovat všechny jeho metody, které ale provádí po svém. Je možné namísto rozhraní použít jakoukoliv třídu, která ho implementuje
- **Zapouzdření** - pomocí modifikátorů přístupu (public/private/protected), které určují, kdo může k metodám a atributům přistupuvat
- **Dědičnost** – možnost rozšířit jakoukoliv třídu, navíc oproti OOP umožňuje rozšířit, popř. kompletně přepsat metody předka, tj. při zavolání metody může být vykonán odlišný kód

Tento jazyk bývá často označován jako ne zcela objektový. Je to ze dvou hlavních důvodů (více na [28]):

- **Ne vše je objekt** – Java obsahuje primitivní datové typy
- **Objekt může měnit stav jiného objektu** – v Javě je možné pomocí modifikátorů přístupu přistupovat k atributům (stavu) jiných objektů

5 Existující řešení

Neuronové sítě byly za dlouhou dobu, po jakou jsou vyvíjeny, implementovány v mnoha formách a podobách. Některé, kde neuron představoval jednu funkční součástku – hardwarové sítě, a jiné, kde je celá síť napsána formou nějakého zdrojového kódu – softwarové sítě. U prvních jmenovaných se jedná o zastaralý model, který sloužil převážně v raných počátcích pro výzkum tohoto oboru.

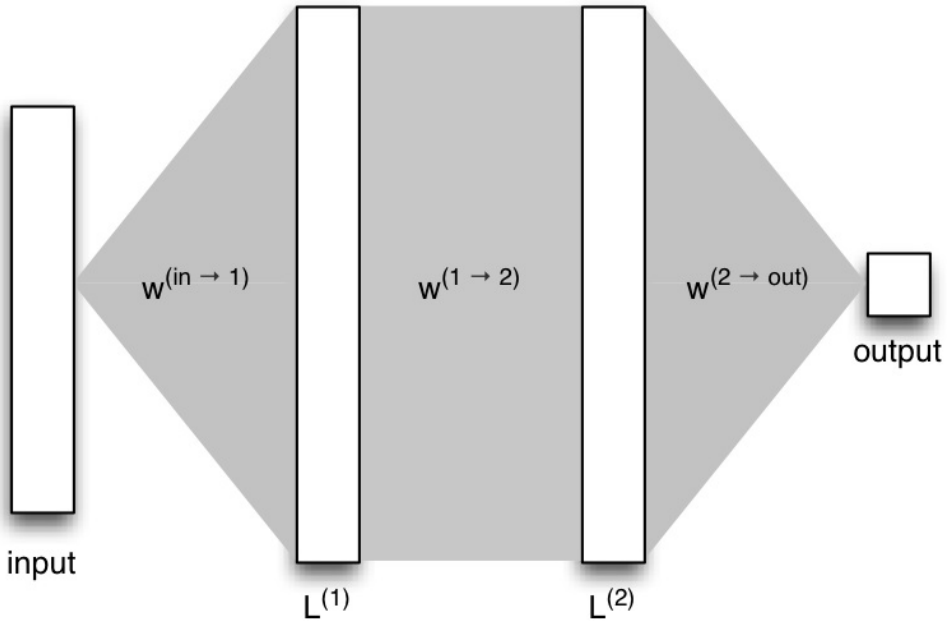
Tato práce se věnuje jen softwarovým sítím, takže zde budou vyjmenovány příklady tohoto typu. Přístupů jak model sítě implementovat existuje několik. V této práci budou vyjmenovány dva, se kterými se autor v rámci výzkumu setkal nejvíce. Jsou jimi maticové a pseudo-objektové.

5.1 Přístupy v implementaci

Zde jsou popsány vybrané způsoby, které jsou uplatněny při vytvoření programové neuronové sítě na základě matematického modelu NS. Jsou uvedeny dva nejobecnější – matice a pseudo-objektový přístup.

5.1.1 Matice

Jak vyplývá z názvu, toto řešení využívá matic. Matice představují dvourozměrné pole hodnot, které se skládá ze sloupců a řádků. Tato síť se pak skládá z několika matic, kde každá matice odpovídá jedné vrstvě a šíření signálu je pak implementováno pomocí násobení těchto matic. Matic může být několik druhů, jedna může představovat váhy na určité vrstvě, druhá výstupní hodnoty všech neuronů na vrstvě, další hodnoty předchozích iterací (u rekurentních sítí) a tak dále. Na Obrázek 12 je ukázána taková feedforward síť, kde vždy šedá oblast **w** představuje matici vah pro přechod mezi vrstvami (například první šedá matice odpovídá vahám ze vstupní vrstvy na vrstvu skrytou) a sloupce **L** představují matici hodnot výstupů všech neuronů v této vrstvě. Více o tomto přístupu je možné se dočíst například zde [31].



Obrázek 12 - neuronová síť implementovaná ve formě matic [31]

Toto řešení je ideální, pokud jde o výkon neuronové sítě. Je to převážně díky možnosti využití grafických karet pro provádění výpočtů, ty jsou vysoce specializované právě na práci s maticemi (díky paralelizaci výpočtů) – obraz není nic jiného, než matice, přičemž každý bod představuje pixel – a proto jsou na práci s takovými sítěmi oproti běžným procesorům řádově efektivnější. Problémem u tohoto přístupu je špatná upravitelnost kódu - u velkých sítí se jedná o značně nepřehledný kód, který se jen velmi těžko mění. Navíc pro komplexnější topologie sítí (například LSTM) není moc použitelná. Její výhodou je rychlost zpracování dat, nevýhodou pak, že není moc univerzální.

5.1.2 Pseudo-objektový

Tento přístup využívá v nějaké formě objektový design. Převážně se jedná o využití tohoto paradigmatu k zpřehlednění kódu využívajícího matice. Nejedná se tedy o zcela odlišný postup, ale spíše o úpravu předchozí formy implementace tak, aby se lépe upravovala a byla o něco přehlednější. Těchto forem je několik, ale obvykle využívají to, že všechny matice, které odpovídají jedné vrstvě, jsou zapouzdřeny do speciálního objektu, který pak představuje tuto vrstvu. Tato vrstva dostane vstup, pomocí její matice vah, pak tento vstup transformuje a nastaví jako

svůj výstup. Ve své podstatě se jedná o jednovrstvovou maticovou neuronovou síť, kterých je spojeno více dohromady.

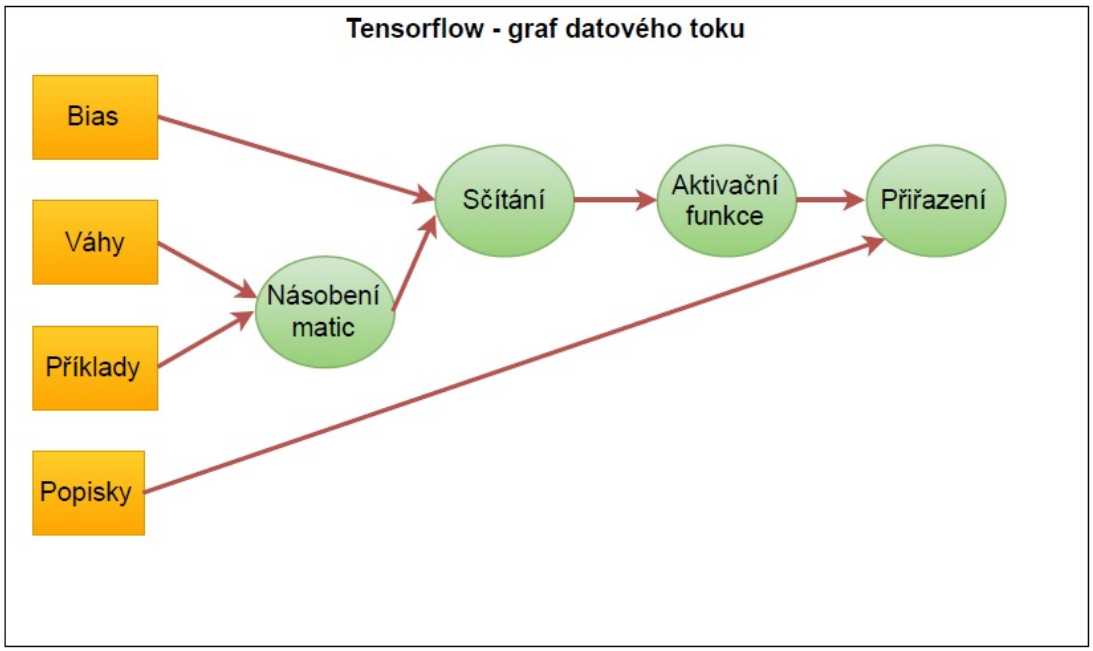
5.2 Existující knihovny

Zde jsou popsány některé již existující frameworky, které jsou zaměřeny na práci s neuronovými sítěmi. Nejedná se o všechny řešení, ale jsou popsány tři populární knihovny, kde každá využívá jiný princip implementace – TensorFlow (Grafy datových toků), Encog (Pseudo-objektový přístup s maticemi) a Neuroph (Objektový přístup zaměřený na Neuron jako základní jednotku). Tyto knihovny jsou dále v práci porovnány s navrhovaným řešením (viz. 6.5).

5.2.1 TensorFlow

TensorFlow je OpenSource knihovna pro numerické výpočty, využívající grafy. Byla vyvinuta výzkumníky a inženýry z Google Brain Teamu, spadajícího pod organizaci pro Vývoj Strojové Inteligence, pro účely strojového učení a výzkumu hlubokých neuronových sítí, přičemž systém je dost obecný na to, aby šel použít i v jiných oblastech. Tato knihovna je napsána v jazycích Python a C++[35].

Hlavní myšlenkou tohoto systému je využití matematických grafů, přesněji diagramu datových toků se stavem, jako neuronové sítě (viz. Obrázek 13). Vrcholy takového grafu představují jednotlivé operace (na obrázku zeleně), mezi které může patřit násobení matic, aktivační funkce nebo sčítání. Hrany reprezentují samotná data, tedy vícerozměrná datová pole, tzv. tensory (na obrázku červeně) [34][35].

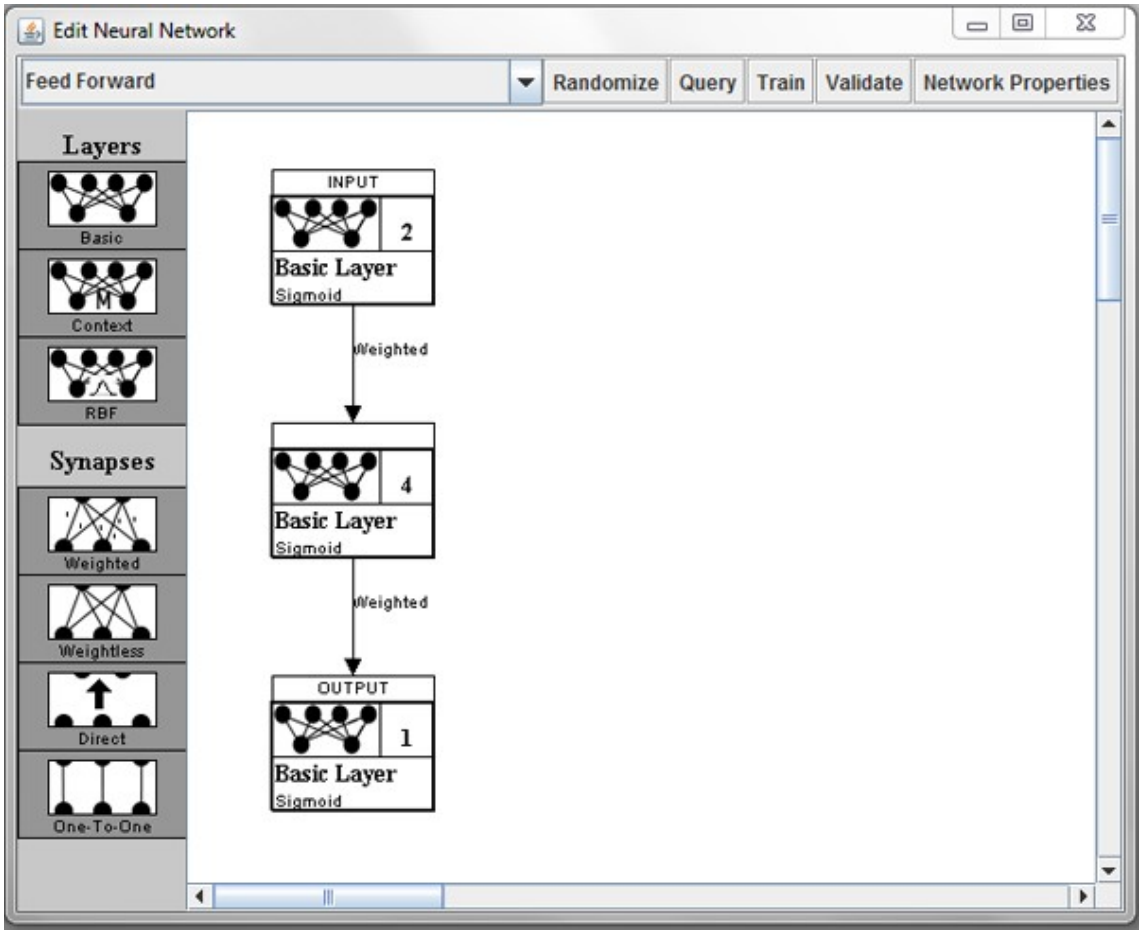


Obrázek 13 - ukázka grafu použitého v knihovně TensorFlow [autor]

5.2.2 Encog

Jedná se o Framework pro strojové učení, který je dostupný v jazycích Java, .NET a C++ vyvinutý v roce 2008. Encog obsahuje třídy pro širokou škálu sítí a podpůrné metody na normalizaci a zpracování dat pro tyto sítě. Encog trénuje sítě s použití více vláknové odolné propagaci, která umožňuje efektivní učení i velmi komplexních sítí[37][36]. Tato knihovna využívá principu vrstev.

Základním stavebním prvkem sítě není neuron, ale vrstva. Ta má pevně daný počet neuronů, výpočet pro výstup a aktivační funkci. Každá vrstva se skládá z několika druhů matic - váhy, výstupy atd. Funguje tedy na principu násobení matic spojeného s objektovým přístupem (viz. 5.1.2 a Obrázek 14).



Obrázek 14 - Rozhraní Encog [50]

5.2.3 Neuroph

Neuroph je objektově orientovaný ANN Framework napsaný v jazyce Java. Poskytuje knihovnu a GUI nástroj easyNeurons pro vytváření a trénování neuronových sítí. Jedná se o Open Source projekt, hostovaný službou SourceForge pod licencí Apache[38]. Tato knihovna je plně objektová a její základní třídy odpovídají konceptům neuronových sítí. Obsahuje různé druhy neuronů, vrstvy, synapse, váhy, přenosné a vstupní funkce, učící pravidla apod.

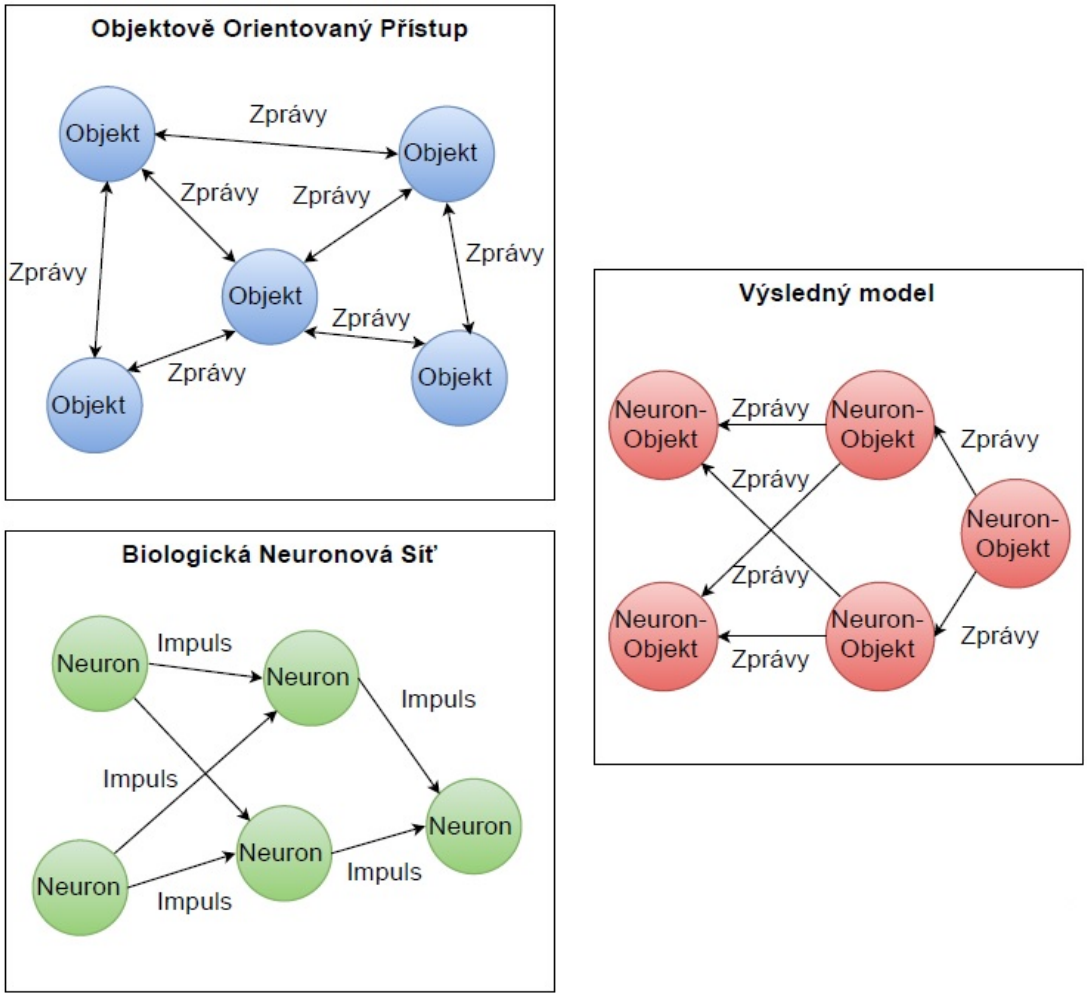
6 Návrh vlastního řešení

V této kapitole se nachází autorovo vlastní řešení v oblasti implementace neuronových sítí. Na základě teoretického rozboru obou problematik z předchozích kapitol je zde popsán princip objektového přístupu a biologických neuronových sítí a rozebrána možnost spojení obou těchto principů do výsledného návrhu tak, aby co nejvíce využíval výhod objektového přístupu a zároveň se co nejvěrněji držel BNN, ale neporušil základní pravidla, kterými se umělé neuronové sítě jako obor řídí.

Nejdříve je popsána základní myšlenka vedoucí k vytvoření této práce a v následující kapitole je popsán výsledný model s nejdůležitějšími entitami, které jsou jeho součástí. Výsledkem této kapitoly by měl být model, který potom bude implementován ve formě knihovny NeuralUHK.

6.1 Základní myšlenka návrhu

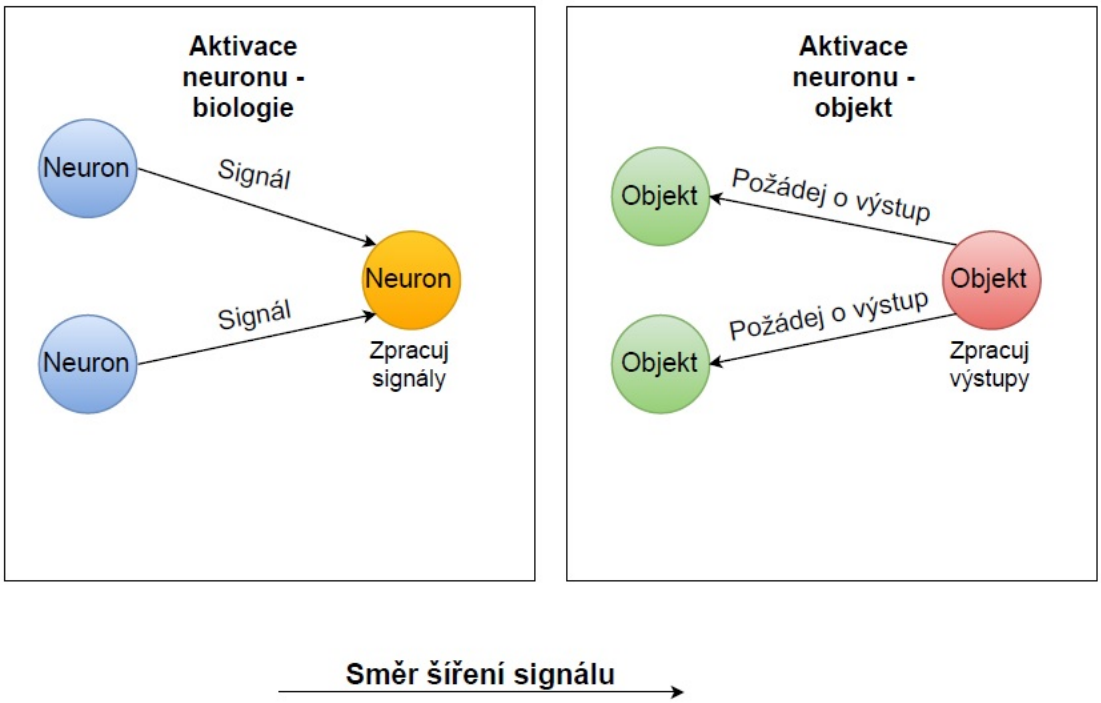
V objektovém principu probíhá interakce mezi jednotlivými objekty pomocí vzájemného zasílání zpráv. Objekt sám mění svůj stav na základě zpráv přijatých od ostatních objektů. Biologické neuronové sítě fungují na principu šíření signálu napříč celou sítí s tím, že záleží na samotném neuronu, jak bude signál interpretovat. Pomocí dendritů (vah) určuje významnost přijatých signálů, a to způsob, jak na tyto signály bude reagovat (vyšle impuls, nebo nevyšle). Existuje značná podobnost mezi těmito dvěma principy a bylo by výhodné se je pokusit spojit do jednoho modelu (viz. Obrázek 15).



Obrázek 15 - myšlenka spojení Biologické neuronové sítě a Objektové orientovaného programování [autor]

Pokud by se funkčnost s objektovým přístupem a BNN srovnala, pak by mohl neuron odpovídat objektu a šíření signálu by odpovídalo zasílání zpráv (viz. Obrázek 15), kde by se objekt dotázal na výstup neuronu, na který je napojen. Bylo by na samotném objektu, jak by takový výstup (signál) interpretoval.

Neuronů v mozku člověka existuje několik druhů a každý mají jiný úkol, ale signál se napříč všemi druhy šíří stejně. Bylo by tedy žádoucí se toto chování pokusit nasimulovat ve výchozím modelu a využít naplno výhody OOP. Jednou z možností je navrhnout základní neuron, který by odpovídal právě za spojení s ostatními neurony a skupinu rozšířených neuronů, které by doplňovali toto spojení o svoje specifické úkoly.



Obrázek 16 - Biologický neuron vs. Navrhovaný model [autor]

Takto navržený neuron by se ale lišil v několika věcech. Biologický neuron se aktivuje ve chvíli, kdy je překročen jeho práh, zatímco ten umělý se aktivuje v určitý čas. Dále se signál šíří z výstupu směrem na vstup následujícího neuronu – v objektovém přístupu by se musel cílový neuron zeptat na výstup zdrojového (viz. Obrázek 16). Je to z toho důvodu, že v objektovém principu funguje interakce mezi objekty na principu zasílání zpráv (viz. 4). Neuron se tedy musí při aktivaci dotázat neuronů, na které je připojen, jakou hodnotu mají v tomto momentě na výstupu.

Aktivace neuronu v mozku je vyvolána přijetím dostatečného množství impulzů (viz. 3.1.2). Poté je vyslán impuls do všech neuronů, které jsou na něj napojeny pomocí synapsí. Tento přístup v mozku zajišťuje paralelní zpracování všech dat. V objektovém přístupu by toto šlo nasimulovat, nicméně by to neodpovídalo teorii umělých neuronových sítí tak, jak se jim věnuje tento obor v softwarové implementaci. Je to z toho důvodu, že podle definice umělého neuronu se neuron neaktivuje při překročení prahu. Na místo toho se v danou chvíli vypočte hodnota ze všech jeho vstupů (ve většině případů ještě ovlivněna váhami) a ta je dále předána aktivační funkci, která představuje práh neuronu (viz. 3.2.2). Tím je simulována aktivace přijetím dostatečného množství signálů – pokud

jsou všechny vstupy malé (nebyl překročen práh), pak i neuron na svůj výstup nastaví skrz aktivační funkci malou hodnotu – nevyslal tedy v podstatě signál (malá hodnota neovlivní výstup u napojených neuronů). Je tedy potřeba, aby výsledný model obsahoval objekt, jenž by reprezentoval vrstvy, které určují, v jakém pořadí budou neurony aktivovány. V tomto se od BNN výsledný návrh liší. Takový objekt by představoval vrstvu, jejímž úkolem je aktivace všech svých neuronů tak, aby odpovídala paralelní aktivaci v určitém čase – je třeba aktivovat všechny neurony ve vrstvě ve stejnou chvíli.

Dále musí model obsahovat entitu synapse, která by v sobě obsahovala informace o tom, jaký neuron je zdrojový a jaký neuron je cílový – z jakého neuronu se na jaký šíří signál. Jak bylo zmíněno v kapitole o biologických NS, z biologického hlediska se spíše jedná o popis funkce výměny informací mezi dvěma neurony, než o samostatný objekt. Nicméně z objektového hlediska je třeba takovou entitu mít a moci jí identifikovat. Ve výsledném návrhu tedy takový objekt bude.

Poslední nutnou součástí by měla být samotná síť. Ta by měla obsahovat všechny ostatní prvky a tvořit zapouzdřující objekt odpovědný za interakci s okolním systémem. Stejně jako v OOP, tak i v BNN tvoří spíše zapouzdřující objekt - mozek sám o sobě není jedním celkem a stejně i tak tento objekt by neměl mít nějaké další vlastnosti. Měl by pouze jako entita, která v sobě obsahuje neuronovou síť tak, aby se před okolím tvářila jako jeden celek. Měla by přijímat na svém vstupu nějaká data, která zpracuje a okolí vrátí výsledné hodnoty.

V následující kapitole jsou popsány entity, které vznikly na základě této myšlenky.

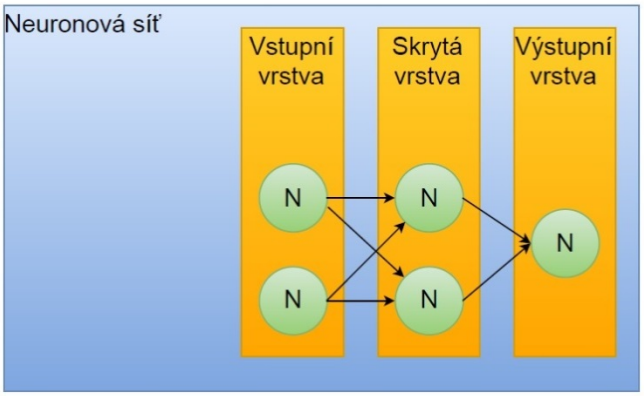
6.2 Entity

V této části jsou popsány jednotlivé entity, které byly vytvořeny na základě abstrakce a dekompozice z biologických neuronových sítí. Hlavním účelem tohoto postupu je rozdělení celého systému do co nejmenších, snadno upravitelných bloků, které budou každý mít minimum úkolů, což by mělo zajišťovat snadnou upravitelnost modelu a údržbu kódu. Zároveň je třeba zachovat co nejvíce funkčnosti z BNN. Každý takhle vytvořený objekt představuje v OOP přístupu třídu,

která má vlastní metody a atributy. V následujících podkapitolách jsou jednotlivé třídy popsány, graficky znázorněny a zde je vysvětlena jejich základní funkčnost. Mezi tyto třídy patří síť, vrstva, neuron a synapse.

6.2.1 Síť – hlavní objekt

Neuronová síť představuje hlavní řídící objekt celého modelu. Skládá se z vrstev, ve kterých jsou umístěny jednotlivé neurony, které jsou, stejně jako v biologických neuronových sítích, navzájem propojeny pomocí jednoduchých synapsí (viz. Obrázek 17). Toto propojení ale nezávisí na pořadí ve vrstvách a samotná vrstva jen reprezentuje šíření signálu (viz. 6.2.2 a 6.2.3). Objektově by se tato entita dala popsat jako jednoduchá zapouzdřující třída, obsahující jen minimum funkčního kódu a proměnných, jejímž účelem je v určité sekvenci volat metody jiných tříd, které jsou její součástí - v tomto případě se jedná převážně o vrstvy, ze kterých se síť skládá.



Obrázek 17 - navrhovaný model neuronové sítě [autor]

Základní funkcí sítě je směrování signálu skrz jednotlivé vrstvy a neurony, které tyto vrstvy obsahují, od vstupu až na výstup. Tato funkčnost je implementována ve třech základních metodách zodpovídajících za zpracování signálu – **nastav vstup, aktivuj síť, získej výsledek**. Algoritmus tohoto šíření signálu by se dal napsat následovně (více o šíření signálu je v následující kapitole):

1. **Na výstup vstupní vrstvy nastav množinu vstupních dat**
2. **Najdi vrstvu umístěnou za vstupní**
3. **Pro všechny další vrstvy opakuj, dokud existuje další vrstva**

   a. **Aktivuj vrstvu**
   b. **Najdi následující vrstvu**
4. **Z poslední vrstvy vezmi výstup a zprostředkuj ho systému**

Síť dále obsahuje ještě manažerské metody pro správu sítě. Jedná se například o metody pro uložení stavu sítě a všech jejich vah a metody pro znovu nahrání sítě z předem uloženého stavu.

6.2.2 Vrstva – šíření signálu

Tato entita sama o sobě nepředstavuje žádný reálný objekt v BNN, ale spíše reprezentuje šíření signálu napříč celou sítí. Důvodem tohoto řešení je to, že v reálných sítích se šíří signál z neuronu na neuron paralelně – signály se větví a shlukují a není efektivní ani moc smysluplné se pokusit toto chování simulovat na počítači. Maticové řešení NS umí simulovat šíření signálu také – několik dvourozměrných matic, představující váhy a výstupy, se bude násobit zleva doprava. Problémem tohoto řešení je například u rekurentních sítí. Nelze totiž simulovat napojení na předchozí vrstvy, protože šíření takového „impulsu“ probíhá jen zleva doprava – maticový přístup detailněji popsán na [31]. Tento problém lze obejít vytvořením dalších matic, které by simulovali hodnoty neuronu v čase $t-1$ nebo jiném. U komplexnějších sítí je pak velmi komplikované takovou strukturu jakkoliv upravovat a síť se tak stává velmi obtížně udržovatelnou a velmi těžko upravitelnou. Navíc některé komplexnější napojení takto řešit nelze.

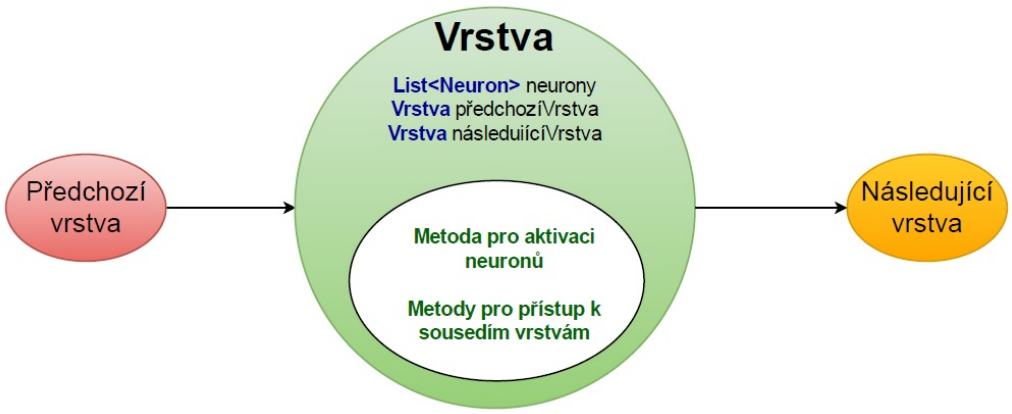
Vlastním návrhem řešení je vytvoření objektu, který by neovlivňoval zapojení neuronů ani jejich funkci, jen by určoval, v jakém pořadí budou neurony aktivovány – představoval by impuls, který se šíří sítí ze vstupu až na výstup. Tento přístup umožňuje simulovat vstup impulsu (množiny dat) do sítě a jeho průchod napříč sítí až k výstupu s tím, že díky možnosti nezávislého napojení je možné simulovat napojení neuronů oproti směru šíření signálu (rekurentní napojení). Tato vrstva by měla obsahovat, jak neurony, tak znalosti tom, která vrstva je před ní a která se nachází za ní - odkud signál přichází a kam směřuje. Naopak nemá informace o tom, jak jsou neurony napojeny (viz. Obrázek 18). Dále by měla být její

součástí aktivační funkce, která by měla být částečnou implementací následujícího algoritmu šíření signálu (vyznačeno tučně):

- **Proveď aktivaci**
  - **Pro všechny neurony ve vrstvě: aktivuj se**
  - **Pro všechny neurony ve vrstvě: nastav výstup**
- Aktivuj následující vrstvu

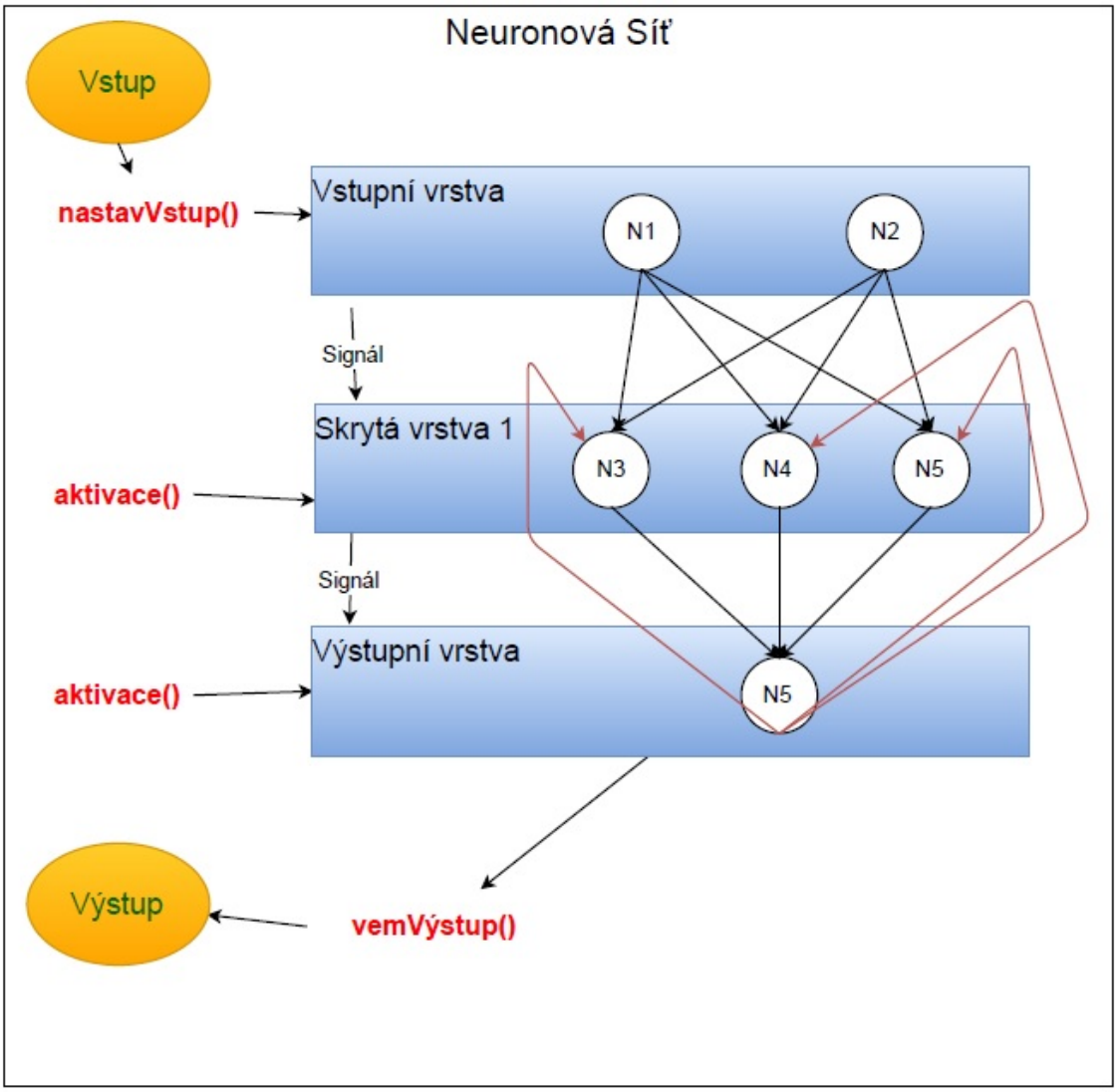
Z důvodu simulace paralelního šíření signálu – aktivace všech neuronů v jedné vrstvě by měla probíhat ve stejný čas – je potřeba nejdříve provést tuto aktivaci pro všechny objekty ve vrstvě a teprve po proběhnutí této aktivace se její výsledky nastaví na výstup neuronu.

Tento přístup využívá principu zapouzdření - vrstva nemá přístup k vnitřní struktuře neuronu a netuší, jaké má neuron připojení. Její jediný úkol je provést aktivaci všech jejich neuronů, -a to je vše.



Obrázek 18 - návrh entity vrstva [autor]

Jak je vidět z obrázku, (viz. Obrázek 19) signál se šíří skrz síť nezávisle na propojení neuronů. Na vstupní vrstvu je nastavena množina dat a provede se aktivace pro všechny následující vrstvy. Data projdou skrze NS, kde jsou upravena na konkrétní výstup, v tomto případě je předchozí výsledek neuronové sítě promítnut na první skrytou vrstvu – díky tomu, že neurony na první skryté vrstvě mají přístup k hodnotám na výstupu, které ve chvíli, kdy je signál na 1. vrstvě, odpovídají výsledku pro předchozí množinu dat (neuronová síť je ovlivněna jejím předchozím výsledkem).



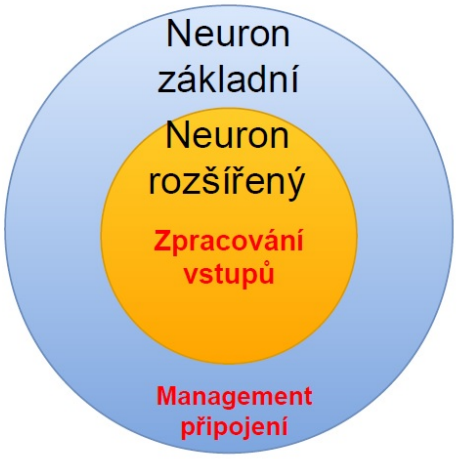
Obrázek 19 - šíření signálu pomocí vrstev [autor]

6.2.3 Neuron – zpracování

Neuron je v tomto modelu, stejně jako v BNN, základní funkční jednotkou, celé sítě a odpovídá za většinu výpočtů, které v tomto modelu probíhají. Představuje abstrakci základní funkčnosti biologického neuronu od jeho biochemické implementace.

Neuron se v tom přístupů skládá ze dvou základních částí. Jsou jimi základní neuron – neboli manažer připojení, a rozšířený neuron – manažer výpočtů (viz. Obrázek 20). Základní neuron je pevně daná třída (entita), která odpovídá za správu všech připojení. Disponuje znalostmi o všech neuronech, které jsou k němu připojeny jako vstup – tj. všechny neurony ze kterých přijímá výstup. Rozšířený

neuron je spíše obecný model, protože se nejedná o konkrétně implementovanou třídu ale spíše o skupinu tříd. Jak vyplývá z názvu, tato třída rozšiřuje základní neuron o výpočetní funkce a potřebné atributy. Výsledkem tohoto spojení je celek, který využívá základní neuron k přístupu k sousedním neuronům a svojí vlastní implementaci k zpracování signálu.

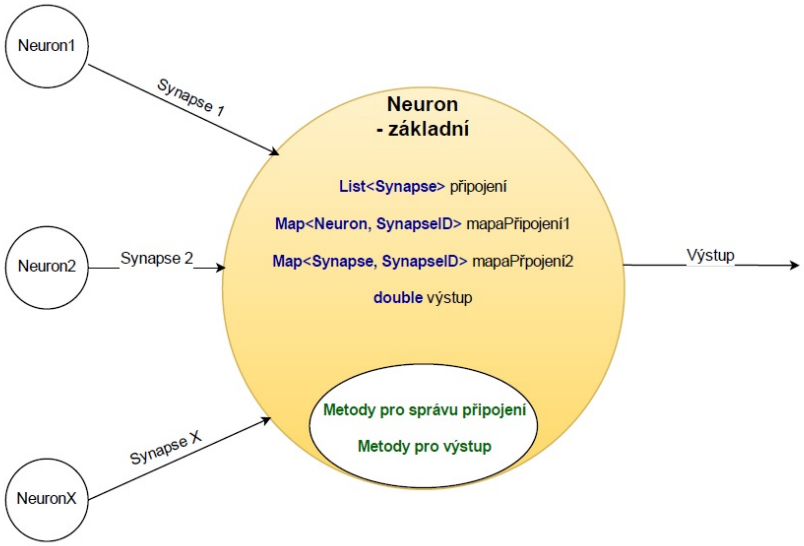


Obrázek 20 - navržený model kompletního neuronu vzniklého spojením základního a rozšířeného [autor]

Tento přístup využívá principu dědičnosti a polymorfismu. Dědičnost (viz. 4.1.4 a Obrázek 20) je implementována tak, že základní neuron představuje předka a rozšiřující neuron je potomek, který dědí jeho schopnosti správy připojení a přidává funkce pro výpočet výstupu. To umožňuje snadno měnit výpočetní funkce neuronu bez rizika zásahu do základní funkcionality, kterou je správa propojení neuronů. Ačkoliv se toto spojení logicky tváří jako dva rovnocenné celky spojené dohromady jedná se o jednu třídu, která jen dědí vlastnosti jejího předka.

Polymorfismus (viz. 4.1.5) je dalším důležitým principem v tomto modelu. Je zde implementován tak, že neuronová síť a její jednotlivé vrstvy, obsahují jen třídu základního neuronu - předka. Z definice dědičnosti vyplývá, že je možné použít potomka všude, kde lze použít předka. Díky tomu je možné do neuronové sítě vložit jakoukoliv třídu rozšiřujícího neuronu, která si výpočet implementuje vlastním způsobem, a reaguje na něj vždy jinak. V této formě se nejedná se o tradiční polymorfismus ve smyslu rozhraní, ale odpovídá jeho definici - volání stejné metody proběhne jinak v různých implementacích.

Základní neuron – manažer připojení. Z hlediska objektového přístupu se jedná o předka všech rozšiřujících neuronů, obsahujícího seznam všech připojených neuronů a metody pro práci s nimi, které od něj tyto schopnosti a vlastnosti dědí a rozšiřují je o metody pro výpočet výstupu. Z hlediska neuronu jako logického celku skládajícího se z manažerů připojení a výpočtů se jedná o část, jejíž jedinou funkcí je spravovat jednotlivá připojení k neuronu pro potřeby manažera výpočtů. Skládá se z několika jednoduchých částí, které jsou graficky zobrazeny na (viz. Obrázek 21) a detailněji popsány níže.



Obrázek 21 - základní neuron [autor]

- **Připojení** – seznam všech synapsí tohoto neuronu – tj. list všech neuronů, k jejichž výstupu je tento neuron připojen.
- **Mapy připojení** – jedná se o dvě mapy, které jsou seznamem připojení, ve kterých se dá vyhledávat pomocí neuronu nebo synapse (klíče). Např. pokud se jako klíč použije instance neuronu, mapa by měla vrátit, na které pozici v seznamu **připojení** se synapse, která je k němu připojena, nachází.
- **Výstup** – obsahuje hodnotu, která odpovídá výstupní hodnotě neuronu v daném čase.

- **Metody pro práci s připojeními** – metody, které umožňují práci s těmito připojeními. Jedná se o přidání nové synapse (neuronu), vyhledání v mapách atd.

Jednou ze základních funkcí základního neuronu je napojení neuronu na výstup jiného neuronu. To probíhá následujícím algoritmem:

1. Připoj se k neuronu
2. Vytvoř pro toto připojení synapsi
3. Vlož synapsi do seznamu připojení
4. Vlož pořadové číslo synapse (počet synapsí) do map s oběma klíči (neuronem a synapsí)
5. Inkrementuj počet synapsí

Sám o sobě se tento neuron v síti použít nesmí – kromě vstupní vrstvy. Je to proto, že tato třída neobsahuje žádnou možnost výpočtu jeho výstupu. Je nutné ji rozšířit o tyto funkce a vlastnosti nějakou jinou třídou. Je to z toho důvodu, že výpočetní část a váhy se v různých typech ANN mohou lišit, ale metody a správa propojení zůstává vždy stejná. Proto je tato funkce odstíněna do této třídy, aby nebylo potřeba je zvlášť, pro každý typ neuronu, implementovat.

Rozšiřující neuron – manažer výpočtů. V objektovém přístupu se jedná o potomka základního neuronu, z logického hlediska se jedná o část neuronu, která zodpovídá za výpočet výstupu. Nejedná se o jednu třídu, spíše o koncept tříd. Každá z těchto tříd musí přepsat dvě hlavní metody základního neuronu – vypočítat výstup a nastavit výstup. Toto je rozděleno z důvodu paralelního šíření signálu, zmíněného v předchozí kapitole. Vzhledem k faktu, že neuronů existuje několik druhů, které se liší podle počtu a typu vah, výpočetních funkcí, aktivačních funkcí a dalších věcí, je třeba pro každý typ neuronu vytvořit novou rozšiřující třídu základního neuronu a všechny tyto atributy v ní specifikovat. Správu připojení není třeba díky dědičnosti řešit a stačí využívat metody předka. Díky tomu rozšiřující neuron bude vždy jen obsahovat atributy a metody nutné pro

výpočet výstupu a učení a při změnách v jeho kódu nehrozí zásah do systému propojení neuronů.

Rozšiřující neuron přidává do tohoto konceptu neuronu funkce vah a výpočtů.

Jako příklad je zde uveden obyčejný neuron se standardními váhami – tj. pro každou synapsi existuje jedna váha a výstup je vypočítán jako suma výstupů připojených neuronů a jim přiřazených vah. Je tedy třeba vytvořit pro tento neuron třídu, která bude rozšiřovat základní neuron, a implementovat tyto algoritmy (viz. Obrázek 22). Vzhledem k faktu, že správa sousedních neuronů je vyřešena v předku, stačí vytvořit funkci pro výpočet vnitřního stavu neuronu a výstupu. Tato třída by tedy obsahovala funkci pro výpočet vnitřního stavu, kde a_i představuje výstup i-tého neuronu a w_i představuje váhu tohoto spojení:

$$y = \sum_{0}^{n} a_i w_i$$

Dále obsahuje aktivační funkci, která bude mít podobu sigmoidu, kde t představuje hodnotu vnitřní stavu neuronu a S je výstup pro tuto hodnotu a představuje celkový výstup z neuronu:

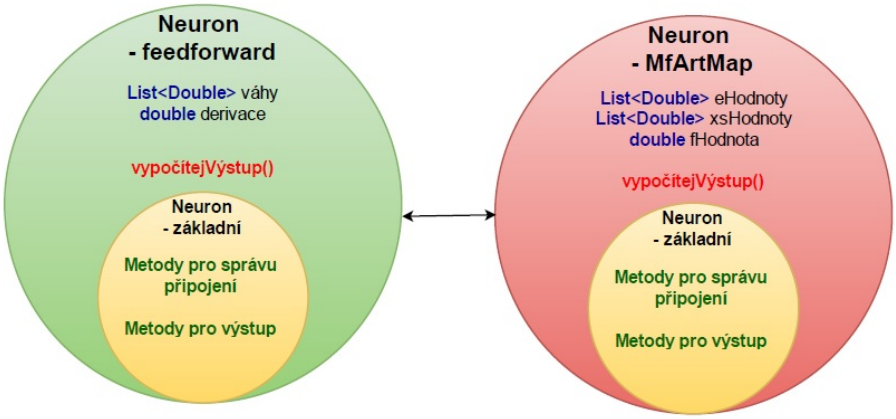
$$S(t) = \frac{1}{1 + e^{-t}}$$

Algoritmus pro aktivaci takového neuronu by vypadal následovně:

1. Aktivuj neuron
2. Pomocí metod základního neuronu získej všechny připojené neurony
3. Pro každý neuron opakuj
    a. Vynásob výstup neuronu s jeho přiřazenou váhou
    b. Přičti je k celkovému součtu
4. Celkový součet předej aktivační funkci
5. Výsledek nastav na dočasný výstup

Vzhledem k principu, popsaném v předchozí kapitole (viz. 6.2.2), není při aktivaci neuronu výsledek aktivační funkce předán na výstup hned, ale v rámci zachování paralelního zpracování je toto provedeno v metodě, která je zavolána až ve chvíli, kdy proběhne aktivace všech neuronů ve vrstvě.

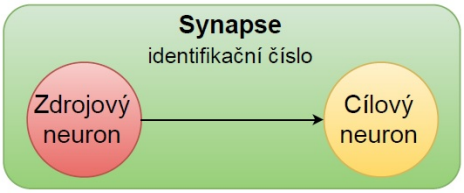
Dále je na Obrázek 22 vyobrazen vykonstruovaný neuron (vlevo) s jeho vlastnostmi a vpravo je pro ukázku neuron pro síť typu MF ArtMap. Je zde vidět, že feedforward dědí vlastnosti základního neuronu – správu připojení a rozšiřuje je o svojí implementaci metody pro výpočet výstupu a přidává jako atributy váhy a derivaci, zatímco MF ArtMap přidává eHodnoty, xsHodnoty a fHodnoty – její obdoba vah.



Obrázek 22 - rozšířené neurony [autor]

6.2.4 Synapse – spojení

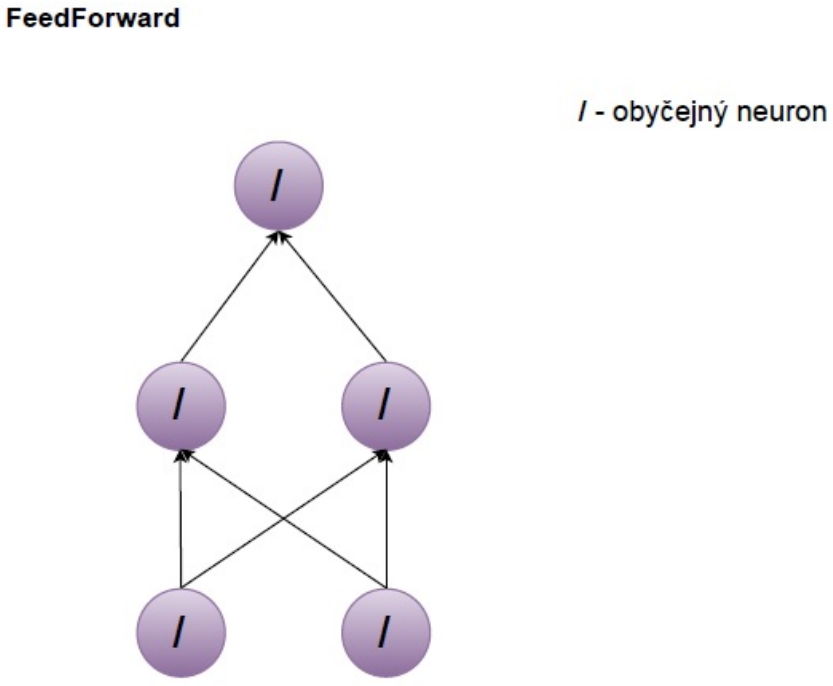
Synapse, stejně jako v biologických neuronových sítích, zde představuje propojení dvou neuronů – na rozdíl od biologických, kde se jedná spíše o formální nazvání tohoto spojení a synapse je fyzicky součástí jednoho nebo druhého neuronu, se zde jedná o samostatnou entitu, která vymezuje konkrétní spojení a slouží k jednoznačné identifikaci tohoto spojení pomocí svého identifikačního čísla (id). Synapse je jednoduchá třída, která obsahuje zdrojový neuron, cílový neuron a své id. Pokud se neuron připojuje k jinému neuronu, vytvoří se objekt této třídy. Připojovaný neuron se označí jako zdrojový a neuron, ke kterému se připojuje jako cílový. Cílový neuron je napojen na výstup ze zdrojového a využívá jeho výstupní hodnotu, viz Obrázek 23.



Obrázek 23 – synapse [autor]

6.3 Příklady sítí

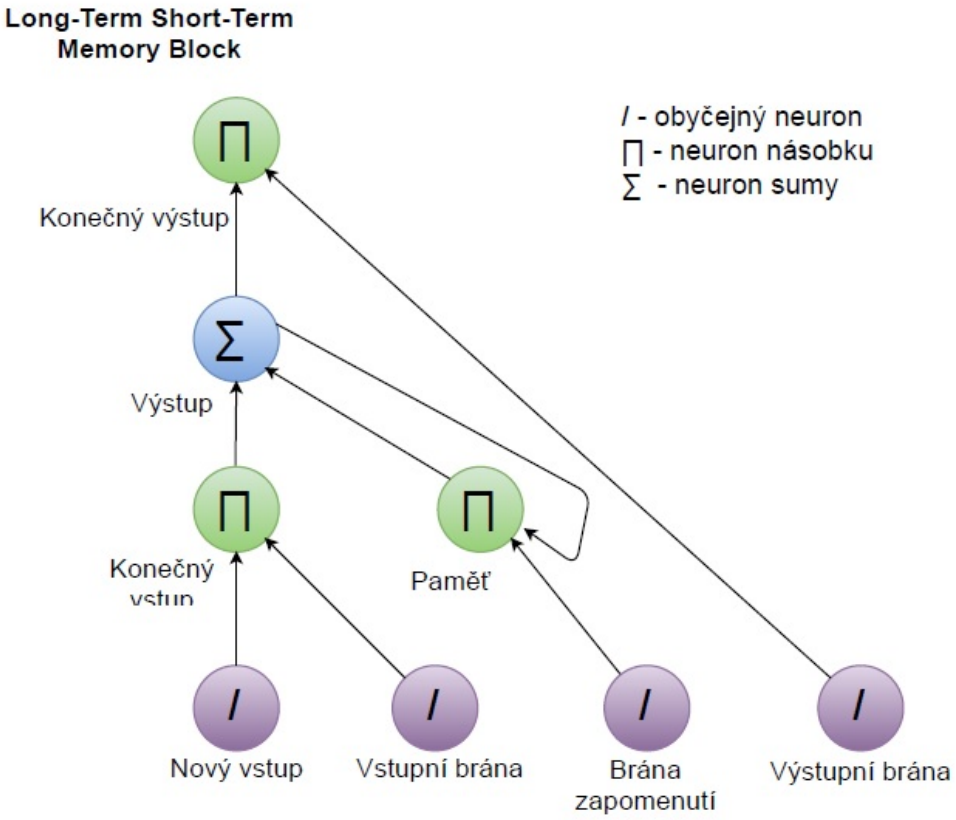
Zde jsou uvedeny tři příklady s použitím této myšlenky. Šipky znázorňují napojení neuronů. Prvním je tradiční feedforward (dopředná) síť (viz. 3.2.4 a Obrázek 24), ve které se šíří signál jen dopředu na následující vrstvu. Všechny neurony ve vrstvě jsou zde napojeny na všechny neurony ve vrstvě následující a obsahují jednu váhu na každé připojení. Průběh výpočtů a učení takového typu sítě je snadno modelovatelné pomocí matic a průběh výpočtu je nakonec prakticky totožný. Nicméně je důležité předvést, že jdou stejným způsobem modelovat jak jednoduché, tak složitější sítě.



Obrázek 24 - příklad Feedforward sítě [autor]

Dalším příkladem je Long Term Short Memory (LTSM) síť. Jedná se o typ architektury rekurentních neuronových sítí (viz. 3.2.4 a Obrázek 25), který

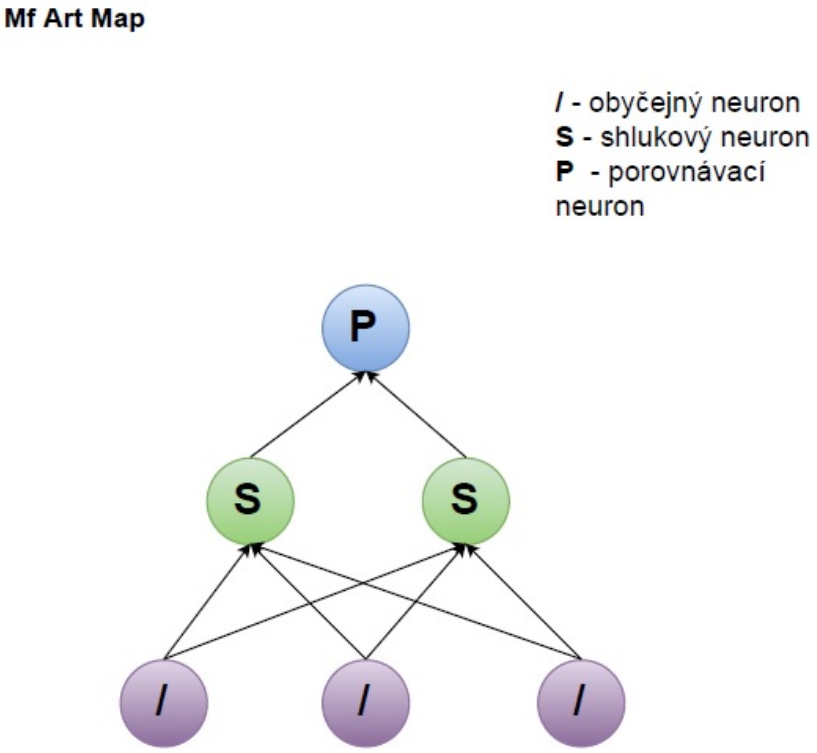
reprezentuje krátkodobou paměť. Ten se skládá z tří druhů neuronů ve čtyřech vrstvách. Prvním je tradiční neuron s jednou váhou na každém spojení a výstup představuje sumu násobků vstupu a jeho přiřazené váhy (na obrázku reprezentován jako „Obyčejný neuron“. Následují dva typy – „Násobek“ a „Suma“. Výstup násobku je výsledkem pro násobení všech jeho vstupů a výstupem sumy je jejich součet, neobsahují tedy žádné váhy. Hlavní myšlenkou je pomocí několika vstupních bran (první vrstva) simulovat paměť neuronu – tj. brány ovlivňují, jestli si hodnotu „zapamatovat“, „vzpomenout“, „zapomenout“, „hned použít“. Více o tomto druhu neuronových sítí se dá dočíst zde [27].



Obrázek 25 - příklad LSTM sítě [autor]

Poslední je neuronová síť MF ART Map. Tato síť je založena (v tomto modelu) na třech vrstvách se třemi druhy neuronů. Prvním je obyčejný neuron na vstupní vrstvě, dalším je neuron reprezentující shluk, který obsahuje tři váhy (viz. 3.2.4 a Obrázek 26) a výpočet probíhá pomocí speciální funkce. Posledním je neuron, který vyhodnocuje, který neuron nejvíce odpovídá vstupním hodnotám. Tato síť

má stejné propojení neuronů jako feedforward. Více o této architektuře se dá zjistit například zde [29].



Obrázek 26 - příklad Mf Art Map [autor]

6.4 Výhody, nevýhody a použitelnost řešení

Jak bylo zmíněno v úvodu celé práce, cílem je implementace neuronových sítí pro testování, ne pro ostrý provoz. Primární je snadná upravitelnost samotných sítí a jejich kódu. Optimalizace je až na druhém místě. Nicméně je třeba shrnout výhody a nevýhody tohoto přístupu a upozornit všechny, kteří z této práce budou chtít vycházet, k čemu je toto řešení použitelné.

Výhody tohoto řešení vyplývají z výhod objektového přístupu. Ten co nejvíce odráží lidské vnímání reálného světa. V tomto paradigmatu je vše objekt, který má nějaký stav, metody a nabízí svému okolí nějaké služby. Umožňuje tak lépe pochopit vztahy mezi entitami a jejich funkce v rámci celého systému a pracovat s nimi. Jak bylo zmíněno v předchozích kapitolách, neuronové sítě se jeví jako velmi komplexní věc, ale ve skutečnosti se jedná o spoustu jednoduchých objektů, které spolu navzájem interagují. Na rozdíl od maticového přístupu, kde je síť jeden komplexní celek, je možné v OOP tento princip věrně napodobit a tím pádem přesně vystihnout hlavní myšlenku biologických neuronových sítí – síť je i zde tvořena jednoduchými objekty, které mezi sebou komunikují pomocí zasílání zpráv. Další výhodou je možnost snadné upravitelnosti a rozšiřitelnosti kódu – každá funkce v síti je delegována jedné jednoduché jednotce (objektu), kterou je možné snadno upravit nebo rozšířit (viz. Obrázek 15).

Největší nevýhoda tohoto přístupu je výkon. Maticemi tvořenou neuronovou síť je možné snadno vypočítat na grafické kartě a tento výpočet paralelizovat. Z toho důvodu je tento přístup pro praktické použití u opravdu komplexních neuronových sítí neefektivní a je nutné se mu vyhnout bez předchozí optimalizace. Nicméně je možné zkoušet několik typů neuronových sítí tak, že každé vlákno bude zodpovídat za jednu neuronovou síť – což se ideálně hodí pro zkoušení několika typů najednou.

Hlavní oblastí, ve které se toto řešení dá aplikovat, je výzkum účinku typů neuronových sítí na různé typy dat. Přesněji, pokud konkrétní výzkum využívá určitá data, pak by tento model umožňoval jednoduše měnit strukturu neuronové sítě a použité typy neuronů tak, aby bylo možné zkoumat, jak sestavená síť je pro

danou problematiku ideální, bez komplikovaných změn v kódu. Jednoduše by pomocí tohoto modelu bylo snadnější nalézt optimální neuronovou síť.

Další možnou aplikací tohoto přístupu by mohlo být využití při vzdělávání. Na rozdíl od implementace v podobě matic, které si laik jen velmi těžko asociuje s matematickým modelem neuronu a sítí jako celku, toto řešení je mnohem blíže jak BNN, tak modelům neuronových sítí a mohlo by sloužit pro vysvětlení základů této problematiky s následnou implementací v podobě matic v následujících krocích, kde již bude potřeba výkon a určitá forma optimalizace.

6.5 Srovnání s existujícími knihovnami

V této kapitole jsou porovnány existující knihovny s modelem, navrženým v rámci práce. Účelem tohoto srovnání je nastínit rozdíly mezi navrhovaným řešením a několika nejznámějšími knihovnami zaměřenými na neuronové sítě a poukázat na výhody, které tento přístup oproti nim má.

6.5.1 TensorFlow

Přesto, že cíl obou knihoven je podobný – modelovat různé typy neuronových sítí – tak se oba dva přístupy liší prakticky ve všech prvcích. Základní jednotkou návrhu je neuron, zatímco u TensorFlow (TF) jsou základními jednotkami Vrcholy a Hrany – operace a datová pole (viz. 5.2.1). Návrh práce je postaven na struktuře velmi podobné Biologické Neuronové Síti – jednotlivě propojené neurony -, přičemž TF se pokouší NS promítnout jako datový tok – jednotlivě propojené operace. Práce se tedy pokouší dosáhnout stejných nebo podobných výsledků s použitím jiných principů.

Návrh popsaný v této práci více odpovídá reálnému modelu neuronové sítě – jednoduché objekty, které mezi sebou komunikují a díky množství jejich zapojení vytváří síť schopnou řešit i komplexní problémy (viz. Obrázek 20 a kapitola 6.4). TensorFlow sice využívá OOP, nicméně se nejedná o objekty vytvořené abstrakcí neuronové sítě (tedy Neuron, Synapse, atd.), ale o objekty z teorie grafů (uzel, hrana, …), inspiruje se tedy na rozdíl od navrženého řešení jen samotnou základní funkcí, ale strukturou již ne. Jednou z výhod je, že navržená knihovna tedy více

odpovídá biologickým neuronovým sítím a je tedy pro pochopení principu neuronových sítí výhodnější.

Přesto, že je tato knihovna výborným nástrojem pro práci s neuronovými sítěmi, má několik problémů. Například, jak je zmíněno například zde [41] (jedná se o subjektivní názor, se kterým se autor práce ztotožňuje), zdrojový kód této knihovny není moc přehledný. Dalším problémem je samotná distribuce výpočtů, na čemž ale tým stojící za touto knihovnou usilovně pracuje (viz. [41]). Primární výhodou navrženého řešení oproti této knihovně je tedy snadná upravitelnost a rozšiřitelnost kódu, což je jedním z primárních požadavků, vytyčených v úvodu práce.

6.5.2 Encog

Nejpodstatnějším rozdílem frameworku Encog a knihovny NeuralUHK je jejich základní jednotka. Encog využívá Vrstvu, zatímco druhý jmenovaný Neuron. První řešení je v podstatě maticový přístup – každá vrstva obsahuje jednotlivé matice, které představují neurony se všemi jejich váhami apod. (v jedné vrstvě může být například matice vah, matice výstupů a matice minulých výstupů – pro rekurentní vrstvu). Naproti tomu NeuralUHK je tvořen Neurony, které obsahují svoje synapse, aktivační funkci apod. (viz. 6.2). Z hlediska výkonu má Encog náskok vzhledem k faktu, že matice jsou pro výpočty NS efektivnější. Nicméně přístup navrhovaný v této práci je přehlednější, snadněji udržovatelný, upravitelný a rozšiřitelný, což je pro využití ve vybrané oblasti (pro testování různých typů neuronových sítí a úpravě jejich kódu) důležitější, než samotná rychlost.

Důkazem výhod objektového přístupu s důrazem na Neuron jako základní prvek, je snaha spojit framework Neuroph a Encog tak, aby se dala dohromady efektivita výpočtů knihovny Encog s výhodami frameworku Neuroph, který tento přístup využívá [40].

6.5.3 Neuroph

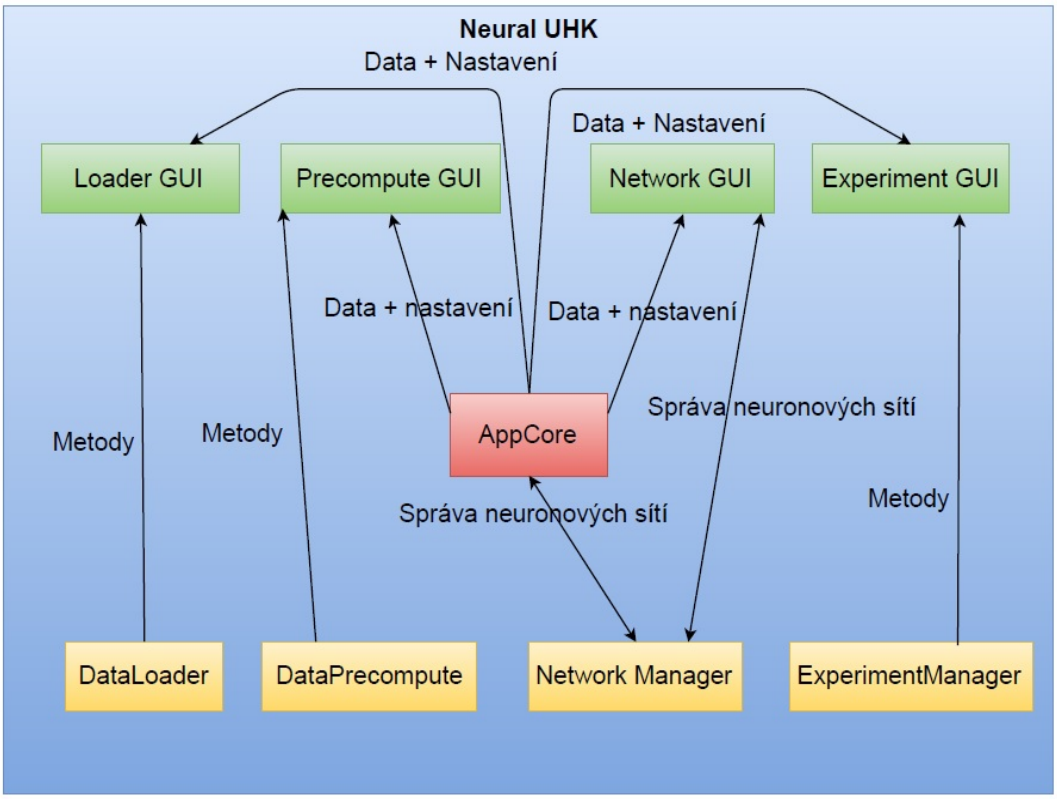
Tato knihovna je velmi podobná návrhu popsaném v této práci. Obsahuje třídy, které odpovídají neuronům, synapsím i vrstvám, s možností jejich rozšíření. Základním prvkem je opět Neuron, z jehož instancí je složena samotná síť. Tento

přístup zajišťuje snadnou upravitelnost a rozšiřitelnost kódu za cenu pomalejších výpočtů – existuje tedy snaha spojit tento a maticový přístup do jednoho funkční modelu (viz. [40]).

Nicméně oproti Neuroph, kde je funkce delegována na několik dalších tříd – například synapse obsahuje váhy, vrstva další funkce atd. – je u návrhu této práce kladen důraz na neuron jako základní funkční jednotku, která se sama stará o výpočet všech hodnot a vrstvy slouží jen k aktivaci neuronu ve zvolený čas. Dalším rozdílem návrhu je oddělení samotných výpočtů (vnitřní struktury neuronu) od správy jeho připojení – periferii tak, že základní prvky (neuron, vrstva, synapse a síť) se starají o přenos signálu napříč sítí a rozšířené o zpracování takového signálu uvnitř neuronu.

7 Implementace řešení v jazyce JAVA

V této kapitole je popsána implementace samotného návrhu ve formě aplikace - NeuralUHK. Ta slouží k testování neuronových sítí pro různé typy dat a dále nabízí dodatečné funkce, které jsou pro tuto funkci potřeba – správa datových setů, úprava a zpracování dat, vytváření neuronových sítí apod. Jedná se o standardní desktop aplikaci s uživatelským rozhraním napsanou v programovacím jazyce Java verze 8 (viz. [33]). Primární částí aplikace je knihovna NeuralManager, která nabízí všechny funkce a třídy pro práci se samotnými neuronovými sítěmi. Zbytek programu slouží jako prostředek k interakci s touto knihovnou a jejímu používání. V současné verzi obsahuje program jako celek možnost vytvářet feedforward neuronové sítě s různou strukturou, nicméně samotná knihovna podporuje několik dalších struktur (LTSM, rekurentní, Mf Art Map, ..).



Obrázek 27 - rozdělení vytvořené aplikace [autor]

Aplikace se skládá z několika modulů, které jsou na sobě nezávislé, a každý odpovídá za odlišnou funkci v rámci celého systému. V současné verzi Neural UHK

jsou všechny moduly součástí jednoho kódu s možností jejich rozdělení do menších knihoven. Je to z toho důvodu, že v současnosti na aplikaci pracuje jediný vývojář a bylo by zbytečné je zatím rozdělovat – k čemuž dojde v Beta verzi samotné aplikace. Tyto moduly jsou vyjmenovány v tabulce (viz. Tabulka 2), graficky zobrazeny (viz. Obrázek 27) a detailněji popsány v následujících podkapitolách.

Tabulka 2 - seznam komponent aplikace

Modul	Účel
AppCore	Primární řídící modul
DataLoader	Nahrávání datových setů ze souboru
DataPrecompute	Úprava nahraných datových setů
NetworkManager	Vytváření a správa neuronových sítí
ExperimentManager	Provádění experimentů

Kromě AppCore a NetworkManager se každý modul skládá z GUI části, která zodpovídá za interakci s uživatelem, a z funkční části, která obsahuje metody pro její funkce. Grafické rozhraní modulů využívá AppCore pro přístup k datům a nastavením aplikace a svou funkční část pro zpracování těchto dat. Tato část představuje sadu statických metod, které plní veškerou funkčnost modulu (viz. obrázek). Dala by se ještě rozdělit do tří logických částí:

- **Jádro** – obsahuje data a funkce pro hlavní chod aplikace – nabízí zbytku aplikace tyto základní funkce a data
- **GUI** – funkce pro ovládání celé aplikace – využívá jádro pro přístup k datům a funkční část pro metody na zpracování těchto dat
- **Funkční část** – statické funkce pro zpracování dat

Součástí programu je několik pracovních složek, ve kterých se ukládají vytvořené neuronové sítě – ve vlastním formátu, nahrané a upravené datové sety, výsledky experimentů a nastavení samotné aplikace. Z těchto složek jsou při startu nahrána veškerá data, která se v nich nachází (neslouží k trvalému ukládání dat). Tímto je umožněno používat program ze stejného stavu, v jakém byl ukončen.

Aplikace prošla několika změnami a až v poslední verzi implementovala návrh popsaný v kapitole 5. Předchozí verze obsahují maticové řešení, které se ukázalo jako složitě upravitelné, náročné na údržbu a kód byl pro ne-autora značně nepřehledný. V následující podkapitole jsou popsány jednotlivé moduly.

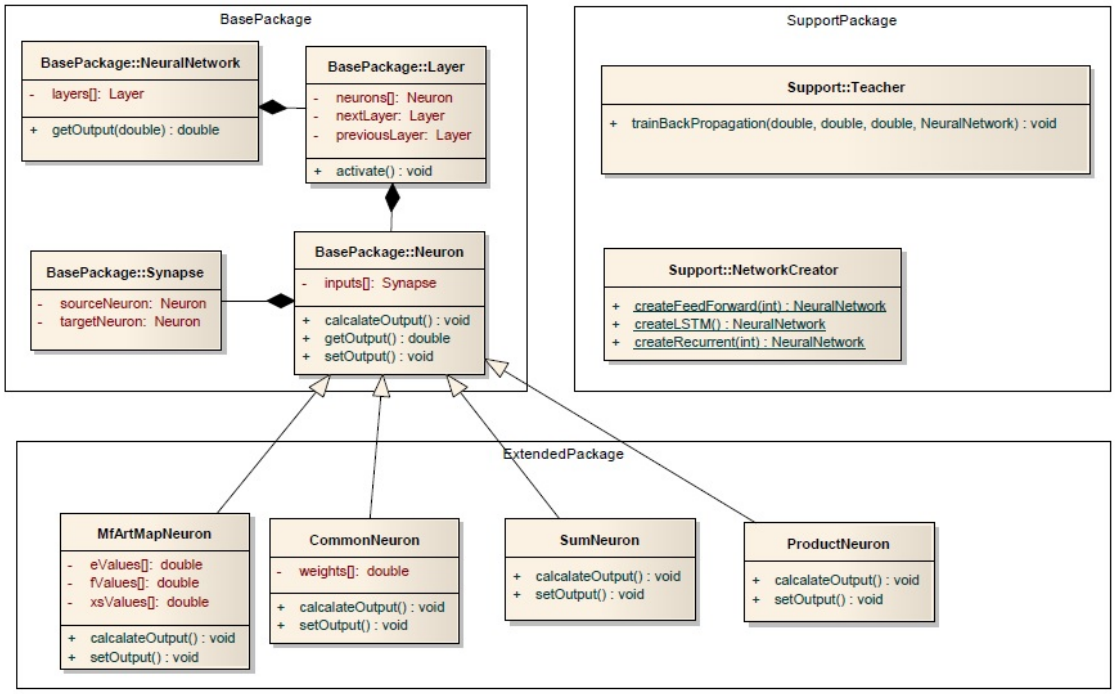
7.1 Moduly

V této kapitole se nachází podrobný popis modulů. Tyto moduly jsou umístěny jako balíčky v rámci jedné zastřešující aplikace. Navzájem jsou, mimo AppCore, úplně nezávislé, což umožňuje snadnou úpravu kódu – program stejně jako neuronová knihovna, jež pro něj byla vytvořena, využívá všech možností objektového přístupu. Jednotlivé moduly je možné upravovat každý jako osamocený Java projekt a nehrozí zásah do dalších částí kódu. Výhoda, která pomáhá rozdělení práce na projektu mezi jednotlivé vývojáře spočívá v tom, že každý zodpovídá za jednotlivé části kódu bez rizika změny jiných modulů.

7.1.1 NetworkManager

Tento modul je hlavní knihovnou celé aplikace. Byl vytvořen podle návrhu řešení, které bylo popsané v předchozí kapitole, a vychází z entit, které v ní byly navrženy. Cílem je, aby tato část fungovala jako samostatný projekt, který je možné implementovat do jakékoliv aplikace zaměřené na zkoumání neuronových sítí. Obsahuje více částí, než běžné moduly.

Součástí této knihovny není grafické rozhraní, ale je přidáno k modulu jako funkčnímu celku v rámci programu. V tomto GUI se nachází seznam neuronových sítí s možností definování nových pomocí zadaných parametrů (počet vrstev, typ, počet neuronů). V současnosti umí pouze vytvářet předdefinované neuronové sítě, ale v dalším vývoji by v rámci knihovny mělo vzniknout ještě rozhraní pro skládání vlastních neuronových sítí.



Obrázek 28 - Diagram tříd Neuronové knihovny [autor]

Knihovna samotná obsahuje kompletní neuronovou síť se všemi objekty, ze kterých je tvořena, metodami pro její vytváření, učení apod. (viz. **Chyba! Nenalezen zdroj odkazů.**). Tyto entity jsou ještě rozděleny do tří základních balíčků z důvodu logického členění knihovny. Ta je rozdělena do základního, který obsahuje samotnou síť a základní prvky, rozšířeného, který obsahuje rozšiřující neurony a podpůrného, jehož součástí jsou učící algoritmy a nástroje pro vytváření neuronových sítí. Tyto balíčky jsou:

Základní neuronová síť – tento balíček obsahuje základní síť, všechny její objekty a její základní funkcí je správa propojení samotné sítě a šíření signálu skrz ni. Skládá se z:

- **Základní neuron** – třída, která obsahuje informace o všech svých sousedních neuronech a metody pro přístup k nim (viz. 6.2.3)
- **Synapse** – tato třída v sobě má referenci zdrojového a cílového neuronu a identifikační číslo (viz. 6.2.4).
- **Vrstva** – v této entitě jsou všechny její neurony a metoda pro jejich aktivaci (viz. 6.2.2).

- **Neuronová síť** – Základní částí tohoto objektu (třídy) je seznam jednotlivých vrstev, ve kterých jsou uloženy navzájem propojené neurony – má v sobě tedy, skrz vrstvy, zapouzdřeny všechny neurony, které se v té síti nachází. Dále obsahuje metody pro nastavení hodnot na vstup a šíření signálu napříč celou sítí a vrácení výsledných hodnot (viz. 6.2.1).

Rozšířené neuronové sítě – zde se v jednotlivých podadresářích (např. feedforward, MF ARTMAP) nachází rozšiřující neurony. Ty rozšiřují základní neuron o jejich implementaci výpočtu aktivace a přidávají svoje vlastní váhy, v současnosti mezi ně patří:

- **CommonNeuron** – nejzákladnější implementace neuronu s jednou váhou na každou synapsi, výpočtem vnitřní hodnoty pomocí sumy vstupů a jejich vah a vybranou aktivační funkcí.
- **ProductNeuron** – tento neuron neobsahuje žádné váhy, výsledkem je pouze násobek všech vstupů. Použitelný v LSTM NS.
- **SumNeuron** – opět nemá váhy, sečte pouze všechny vstupy. Opět LSTM NS.
- **MFArtMapNeuron –** neuron pro tento typ sítě – jedná se o neuron druhé vrstvy – odpovědný za shluky - , který obsahuje 3 váhy a výpočet výstupu provádí přes několik funkcí.

Support – podpůrná část. Metody pro tvorbu a práci s NS, třídy v tomto balíčku jsou:

- **NetworkCreator** – tvorba neuronové sítě na základě zadaných argumentů. Pouze statické metody, které vrací instanci nově vytvořené sítě.
- **Teacher** – třída zodpovědná za učení neuronových sítí. V této verzi obsahuje algoritmy Backpropagation a MFArt map.

7.1.2 AppCore

AppCore představuje funkční jádro celé aplikace, které využívá singleton pattern. Tento přístup umožňuje mít stejná data napříč celou aplikací, aniž by bylo nutné, aby mezi sebou měli ostatní moduly nějakou referenci[32]. Jedná se o hlavní modul celého projektu, jehož funkce a uložená data využívají ostatní moduly pro vykonávání jejich specifických metod. Jeho úkoly jsou následující:

- **Udržovat v paměti všechny objekty, které jsou nutné pro funkčnost aplikace** – tato data spravuje a poskytuje ostatním modulům. Patří mezi ně neuronové sítě, datové sety, neprováděné experimenty apod.
- **Spravovat nastavení programu** – odpovídá za stav a nastavení celé aplikace tak, aby se tato data po vypnutí programu uložila a nebylo třeba používanou konfiguraci znovu nastavovat. Tato data dále dává k dispozici ostatním modulům.
- **Dodávat k dispozici vlákna pro provádění experimentů (budoucí funkce, v současné verzi neimplementována)** - všechna vlákna, ve kterých se mají provádět experimenty, se nachází v tomto modulu, ty jsou pouze „vypůjčeny“ pro potřeby provedení. Jejich správa je ale uložena v tomto modulu.

7.1.3 DataLoader

Úkolem tohoto modulu je nahrávat data ze souborů do jádra aplikace tak, aby je dále mohli používat ostatní části aplikace. V současnosti je jediným podporovaným formátem CSV, tedy soubor, který představuje dvourozměrnou tabulku, kde jsou buňky odděleny čárkou (nejedná se o pevně dané pravidlo, je možné soubory oddělovat). Budoucí plány pro aplikaci obsahují načítání excelových a jiných souborů. Obsahuje dvě části:

- **GUI –** grafická část, obsahuje seznam všech datových setů s informacemi o nich: počet sloupců, počet výstupů a název setu. Dále

rozhraní pro samotné nahrávání souboru se všemi potřebnými parametry.

- **Funkční část** – metody pro načtení souboru a jeho převedení do objektu datového setu.

7.1.4 DataPrecompute

Tento modul umožňuje uživateli upravovat, již nahrané datové sety, slouží pro tzv. předzpracování dat předtím, než jsou tyto sety předány neuronové síti (nebo jiným metodám – např. evoluční algoritmy). Veškerá modifikace dat se nachází v tomto modulu, nicméně je možné do aplikace již nahrát zpracovaná data, bez nutnosti jakékoliv úpravy. Tento modul není pro hlavní funkčnost aplikace nutný, slouží jen pro ulehčení samotného používání aplikace – aby kvůli triviálním úpravám nebylo nutné spouštět další program (např. Excel). Skládá se ze dvou částí:

- **GUI** – obsahuje opět seznam všech datových setů, jejich název a počet vstupů a výstupů. Pro potřeby zpracování dat má rozhraní se všemi možnostmi úpravy dat. Tyto sety získává z jádra aplikace, která je v sobě udržuje. Data jsou upravena pomocí funkční části a opět uložena do jádra.
- **Funkční část** – statické metody pro různé statistické úpravy dat. Datový set je dodán na vstupu a upravený výsledek je na výstupu metody. Mezi metody patří: standardizace dat, různé druhy transformací a rozdělení setu na více částí.

7.1.5 ExperimentManager

V Tento modul se skládá stejně jako DataLoader a DataPrecompute z GUI a funkční části. Jejím úkolem je pomocí vláken z AppCore vytvářet a provádět jednotlivé experimenty.

- **GUI** – skládá se z několika propojených oken. V hlavním se nachází seznam vytvořených experimentů a údaje o nich. Dále možnost vytváření experimentů, které je poté provedeno v několika

jednotlivých oknech, kde probíhá výběr neuronové sítě, typ učení a datové sety, které se k danému experimentu použijí.

- **Funkční část** – metody pro vytváření experimentů a řízení jejich průběhu (správa vláken pro tyto experimenty v budoucnu přes metody AppCore).

8 Experimenty ve vytvořené aplikaci

V rámci testování výsledné knihovny a navrhovaného modelu byly provedeny dva experimenty pro otestování funkčnosti modelu pro obecné problémy a pro jeho uplatnitelnost ve zvoleném oboru. Pro oba experimenty byly použity dva typy neuronových sítí – Feedforward a Mf Art Map. Nastavení bylo zvoleno dle standardních hodnot používaných v rámci výzkumu biomedicínských dat a informace o informace o jednotlivých parametrech lze nalézt v Tabulka 3 (význam těchto parametrů je možné pro Feedforward nalézt na [15] a Mf Art Map na [29]). Experimenty byly provedeny za použití knihovny neuronových sítí a metod ostatních modulů – nebylo využito grafické rozhraní.

Tabulka 3 - nastavení obou neuronových sítí

Feedforward		Mf Art Map	
Skryté vrstvy	75,25	**Práh**	0,65
Počet iterací učení	10000	**E hodnota**	0,05
Momentum	0,9	**F hodnota**	1
Charakteristika	0		
Rychlost učení	0,1		

První experiment - byl zaměřen na otestování funkčnosti vytváření různých typů neuronových sítí, při zachování jednotného modelu neuronové sítě a její vrstvy, s rozdílem pouze v učícím algoritmu, vnitřní struktuře neuronů a jejich vzájemného propojení. NS byly testovány na problematice kruhu ve čtverci. Jedná se o čtverec, který obsahuje vepsanou kružnici a úkolem NS je určit, jestli se určitý bod nachází v nebo mimo kružnici. Datový set použitý pro experiment obsahuje 10.000 jednotlivých bodů. Z těchto hodnot bylo náhodně vybráno 1.000, které byly použity pro učení samotné sítě – pro obě dvě NS stejné, a následně celý set pro otestování, jak se obě dvě adaptovali. Výsledky těchto experimentů se nachází ve dvou kontingenčních tabulkách (viz. Tabulka 4 a Tabulka 5).

Tabulka 4 - výsledky experimentu s daty "kruh ve čtverci" - Feedforward

Feedforward	Reálná data (V kruhu)	Reálná data (Mimo kruh)
Výstup sítě (V kruhu)	4950	56
Výstup sítě (Mimo kruh)	53	4941

Tabulka 5 - výsledky experimentu s daty "kruh ve čtverci" - Mf Art Map

Mf Art Map	Reálná data (V kruhu)	Reálná data (Mimo kruh)
Výstup sítě (V kruhu)	4863	182
Výstup sítě (Mimo kruh)	140	4815

Jak vyplývá z výsledků, oba dva typy NS se podařilo úspěšně natrénovat (Feedforward úspěšnost 98,9% a MF Art 96,7%) a bylo prokázáno, že aplikace vycházející z navrhovaného modelu je funkční pro různé druhy NS a záleží jen na stylu zapojení neuronu a jejich vnitřní struktuře.

Druhý experiment - měl za cíl otestovat použití neuronových sítí ve vybraném oboru – biomedicína. Konkrétně se jedná o snahu využít neuronové sítě k odhadu účinnosti léků na základě jejich vlastností. Pro tento experiment byl vybrán datový set „Tuberkulóza“, obsahující 134 různých léků se 114ti charakteristikami a jejich účinnost při léčbě TBC. Z tohoto setu bylo náhodně vybráno 65 léků pro učení a následně byla pomocí zbytku – 69 - otestována míra naučení NS. V rámci tohoto experimentu bylo provedeno deset nezávislých měření pro Feedforward a jedno pro MF Art Map (bez prohození pořadí dat bude výsledek učení MF Art Map vždy stejný). Při každém započatém měření byla neuronová síť vynulována. Výsledky jednoho z měření jsou zobrazeny v kontingenčních tabulkách (viz. Tabulka 6 a Tabulka 7) a jednotlivé výsledky pro Feedforward jsou v Tabulka 8. Podle očekávání bylo u druhého experimentu prokázáno, že malý počet vstupních hodnot ovlivnil negativně schopnost neuronové sítě se úspěšně naučit a úspěšnost obou sítí oproti prvnímu experimentu nižší: průměrná Feedforward úspěšnost 70,39% a MF Art 59,4%.

Tabulka 6 - výsledky experimentu s daty "Tuberkulóza" - Feedforward

Feedforward	Reálná data (Účinný)	Reálná data (Neúčinný)
Výstup sítě (Účinný)	26	7
Výstup sítě (Neúčinný)	12	24

Tabulka 7 - výsledky experimentu s daty "Tuberkulóza" – Mf Art Map

Mf Art Map	Účinný	Neúčinný
Účinný	26	16
Neúčinný	12	15

Tabulka 8 - jednotlivé úspěšnosti Feedforward sítě (v procentech)

Feedforward	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Úspěšnost	71	69,5	72,4	66,6	66,6	71	68	73,9	73,9	71

Výsledky obou experimentů dokázaly funkčnost celé aplikace. Malý datový set u druhého experimentu byl vybrán záměrně. U biomedicínských dat, která budou v následném výzkumu vyhodnocována, se často vyskytují sety, které obsahují pouze omezený počet řádků. I pro takto malý set, který může být problémem pro vyhodnocení pomocí NS, se ukázalo, že i přes nižší úspěšnost než v prvním experimentu, kde byla vybrána data, která jsou vhodná pro NS, může použití NS dávat zajímavé výsledky, se kterými se dá dále pracovat. Cílem dalšího výzkumu bude najít taková nastavení a typy NS, která budou nejvhodnější pro daný typ dat a velikost setu zejména Biomedicínských dat.

9 Závěry a doporučení

Vytvořená knihovna splňuje všechny nadefinované vlastnosti přesně podle požadavků stanovených v úvodu práce. Jak vyplývá z výsledků obou experimentů, provedených v rámci výzkumu, spojení výhod objektového přístupu a neuronových sítí umožnilo vytvořit jednoduchou, přístupnou a snadno upravitelnou knihovnu, která umožňuje provádět výzkum vlastností různých typů neuronových sítí s možností experimentovat se zdrojovým kódem samotné aplikace bez většího rizika poškození základní funkčnosti – propojení neuronů a šíření signálu pomocí vrstev.

V práci byly nejdříve detailně rozebrány neuronové sítě a objektový přístup, přičemž cílem práce bylo spojení těchto dvou do funkčního návrhu. U Neuronových sítí se jedná zejména o biologický model neuronových sítí – tedy popis jejich funkčnosti - a umělé neuronové sítě, kde je popsána historie jejich vzniku a základní myšlenka inspirace těmi biologickými včetně matematického modelu takových sítí. Poté následoval rozbor již existujících řešení, včetně některých přístupů k implementaci a popisu vybraných existujících knihoven. Další kapitola se zabývala samotným návrhem, vycházejícím z propojení neuronových sítí s objektovým přístupem. Primárně obsahuje podrobný popis základní myšlenky celého modelu a vymezení hlavních entit, které byly vytvořeny. V této části se ještě nachází srovnání výsledného modelu s již existujícími knihovnami. Následující odstavce práce jsou věnovány popisu implementace navrženého modelu – převedení návrhu do vybraného programovacího jazyku (Java) - se seznamem vytvořených tříd, metod a jejich popisem. Poté jsou v práci ukázány a okomentovány výsledky experimentů, které byly provedeny na Biomedicínských datech, což je oblast, kvůli které primárně začal výzkum vedoucí k sepsání této práce. Nakonec je shrnut výsledek této práce a popsány směry, kterými by se další vývoj knihovny měl ubírat.

Objektový přístup k vytvoření NS má oproti maticovému řešení není vhodný. Pro použití při klasifikaci rozsáhlejší datových setů je dobré nejprve využít tento model pro určení, který typ a nastavení neuronové sítě je pro daný problém

vhodný a následně takto získané informace aplikovat na NS využívající maticový přístup.

Dalšími směry vývoje by mělo být učení samotné knihovny. V současném stavu je učení velmi specifické a záleží na zvolené struktuře. Bylo by tedy vhodné vytvořit co nejadaptibilnější objekt odpovědný za učení takové sítě, popřípadě se pokusit učení zahrnout již do rozšiřujícího neuronu. Druhou možností je zlepšení výpočetních vlastností celého programu v podobném stylu o jaký se pokouší iniciativa spojení knihoven Encog a Neuroph.

Výsledkem je tedy aplikace, jejíž součástí je knihovna, která je vhodná pro experimenty s neuronovými sítěmi i k snadnému pochopení tohoto oboru a jejich matematického modelu. Na rozdíl od knihoven Encog a TensorFlow, zaměřených na výkon a praktické použití, nabízí přehledný a především snadno upravitelný kód, který je ideální pro použití ve zvolené oblasti - výzkum typů a nastavení neuronových sítí pro jejich použití v biomedicíně. Oproti knihovně Neuroph je více zaměřena na samotný neuron jako základní funkční prvek a pomocí principu dědičnosti odděluje vnější funkčnost neuronu – šíření signálu a správu propojení – od vnitřní, což je zpracování samotného signálu na jeho výstup. Dalším krokem v tomto výzkumu je samotná aplikace knihovny v tomto oboru.

10 Seznam použité literatury

[1] WAMPLER, Bruce E. *The essence of object-oriented programming with Java and UML*. Boston, MA: Addison-Wesley, 2002. ISBN 0201734109.

[2] POO, Danny C, Derek Beng Kee KIONG a Swarnalatha ASHOK. *Object-oriented programming and Java*. 2nd ed. London: Springer, c2008. ISBN 9781846289620.

[3] Aleks Rudenko. *Do You Speak Java ?: Java Language Fundamentals for Beginners.* : CreateSpace Independent Publishing Platform, February 10, 2016. ISBN 978-1522760214.

[4] PEDRONI, Michela a Bertrand MEYER. *Object-oriented modeling of Object-Oriented Concepts: A Case Study in Structuring an Educational Domain* [online]. Chair of Software Engineering, ETH Zurich, Switzerland [cit. 2016-03-18]. Dostupné z: http://se.ethz.ch/~meyer/publications/teaching/oomodeling.pdf

[5] Feature ComputingSoftware The 2015 Top Ten Programming Languages: New languages enter the scene, and big data makes its mark. *IEEE Spectrum* [online]. 2015 [cit. 2016-03-18]. Dostupné z: http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages

[6] Biological Neural Networks. *TECO research group* [online]. 2000 [cit. 2016-03-23]. Dostupné z: http://www.teco.edu/~albrecht/neuro/html/node7.html

[7] Neural Networks: Biological Neuron. *Stanford University* [online]. 2000 [cit. 2016-03-23]. Dostupné z: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Biology/index.html

[8] KURZWEIL, Ray. *How to create a mind: the secret of human thought revealed*. New York: Viking, 2012. ISBN 978-067-0025-299.

[9] BY A.G. BROWN. *Nerve Cells and Nervous Systems an Introduction to Neuroscience*. Second edition. London: Springer London, 2001. ISBN 9781447102373.

[10] KVASNIČKA, Vladimír. *Úvod do teórie neurónových sietí*. Slovenská republika: IRIS, 1997. ISBN 80-88778-30-1.

[11] KIERNAN, J a Nagalingam RAJAKUMAR. *Barr's the human nervous system: an anatomical viewpoint*. 10th ed. Philadelphia: Wolters Kluwer, c2014. ISBN 978-1-4511-7327-7.

[12] YEGNANARAYANA, B. *Artificial neural networks*. New Delhi: Prentice-Hall of India, 1999. ISBN 8120312538.

[13] ANDERSON, Dave a George MCNEILL. *ARTIFICIAL NEURAL NETWORKS TECHNOLOGY: A DACS State-of-the-Art Report*. Kaman Sciences Corporation, 1992.

[14] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics* **5** (4): 115–133.doi:10.1007/BF02478259

[15] KROSE, Ben a Patrick SMAGT. *An Introduction to Neural Networks* [online]. University of Amsterdam, 1996 [cit. 2016-04-01]. Dostupné z: https://books.google.cz/books?id=M7FTNQAACAAJ

[16] Neural Networks: History. *Stanford University* [online]. 2000 [cit. 2016-03-23]. Dostupné z: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

[17] SOUZA, Alan a Fabio SOARES. *Neural Network Programming with Java: Create and unleash the power of neural networks by implementing professional Java code*. Packt Publishing, 2016. ISBN 978-1-78588-494-8.

[18] Neural Networks: Perceptron. *Stanford University* [online]. 2000 [cit. 2016-03-23]. Dostupné z: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html

[19] *Merriam-Webster dictionary: Learning* [online]. An Encyclopædia Britannica Company [cit. 2016-04-04]. Dostupné z: http://www.merriam-webster.com/dictionary/learning

[20] Artificial Neural Networks Technology: Training an Artificial Neural Network. *University of Toronto*[online]. Toronto, www.utoronto.ca [cit. 2016-04-04]. Dostupné z: http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural3.html

[21] WILSON, Robert A a Frank C KEIL (eds.). *The MIT enyclopedia of the cognitive sciences*. 1st MIT Press pbk. ed. Cambridge, Mass.: MIT Press, 2001. ISBN 978-0-262-73144-7.

[22] Neural Networks: Applications of neural networks. *Stanford University* [online]. 2000 [cit. 2016-03-23]. Dostupné z: http://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/neural-networks/Applications/index.html

[23] Inceptionism: Going Deeper into Neural Networks. *Google Research Blog* [online]. 2015 [cit. 2016-04-04]. Dostupné z: http://googleresearch.blogspot.cz/2015/06/inceptionism-going-deeper-into-neural.html

[24] The neural networks behind Google Voice transcription. *Google Research Blog* [online]. 2015 [cit. 2016-04-04]. Dostupné z: http://googleresearch.blogspot.cz/2015/08/the-neural-networks-behind-google-voice.html

[25] Backpropagation Learning Algorithm. *University of Texas at El Paso* [online]. 2000 [cit. 2016-04-04]. Dostupné z: http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node22.html

[26] Backpropagation Algorithm. *Stanford University: Computer Science Department* [online]. [cit. 2016-04-04]. Dostupné z: http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm

[27] Speech Recognition with Neural Networks. *Andrew Gibiansky: Blog* [online]. 2014 [cit. 2016-04-04]. Dostupné z: http://andrew.gibiansky.com/blog/machine-learning/speech-recognition-neural-networks/

[28] Is Java a Pure Object Oriented Programming Language? *Blogspot: Java67* [online]. 2014 [cit. 2016-04-05]. Dostupné z: http://java67.blogspot.cz/2014/03/is-java-pure-object-oriented-programming-language.html

[29] The MF-ARTMAP neural network. BODNÁROVÁ, Agáta. *Latest Trends in Applied Informatics and Computing*. s. 267-269. ISBN 978-1-61804-130-2.

[30] MF ARTMap. *Neuron A*/[online]. [cit. 2016-04-05]. Dostupné z: http://neuron-ai.tuke.sk/hric/mfartmap/teoria.html

[31] Artificial Neural Networks: Matrix Form (Part 5). *Brian Dolhansky* [online]. 2014 [cit. 2016-04-05]. Dostupné z: http://briandolhansky.com/blog/2014/10/30/artificial-neural-networks-matrix-form-part-5

[32] Singleton Pattern. *Object Oriented Design* [online]. [cit. 2016-04-12]. Dostupné z: http://www.oodesign.com/singleton-pattern.html

[33] Java SE: Overview. *Oracle* [online]. [cit. 2016-04-12]. Dostupné z: http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html

[34] Tensor. *Wolfram: Mathworld* [online]. [cit. 2016-04-21]. Dostupné z: http://mathworld.wolfram.com/Tensor.html

[35] Tensor flow Overview. *Slideshare* [online]. [cit. 2016-04-21]. Dostupné z: http://www.slideshare.net/pavelyerofeyev/tensor-flow-overview

[36] Encog Machine Learning Framework: Project Summary. *Openhub* [online]. [cit. 2016-04-21]. Dostupné z: https://www.openhub.net/p/encog

[37] Techniques for Multi-Threaded Backpropagation for Encog. *Heaton research* [online]. 2009 [cit. 2016-04-21]. Dostupné z: http://www.heatonresearch.com/encog/mprop/compare.html

[38] Neuroph: Java Neural Network Framework. *SourceForge* [online]. [cit. 2016-04-21]. Dostupné z: http://neuroph.sourceforge.net/

[39] Neuroph: Java Neural Network Framework. *SourceForge* [online]. [cit. 2016-04-21]. Dostupné z: http://neuroph.sourceforge.net/improving_performance.html

[40] Encog and Neuroph Collaboration Announcement. *Source Forge: Neuroph: Java Neural Network Framework* [online]. 2010 [cit. 2016-04-23]. Dostupné z: http://neuroph.sourceforge.net/NeurophEncogCollaboration.html

[41] TensorFlow Disappoints: Google Deep Learning falls shallow. *Kdnuggets* [online]. 2015 [cit. 2016-04-23]. Dostupné z: http://www.kdnuggets.com/2015/11/google-tensorflow-deep-learning-disappoints.html

[42] *Biological neural networks: hierarchical concept of brain function*. S.l.: Birkhauser, 2012. ISBN 978-146-1286-523.

[43] Artificial neural network. *Slideshare* [online]. 2013 [cit. 2016-04-26]. Dostupné z: http://www.slideshare.net/vzoghiyan/artificial-neural-network-28960844

[44] SISTEM SARAF MANUSIA. *Almansyahnis* [online]. [cit. 2016-04-26]. Dostupné z: http://www.almansyahnis.com/2015/03/sistem-saraf.html

[45] Synapse: Definition & Transmission. *Study* [online]. [cit. 2016-04-26]. Dostupné z: http://study.com/academy/lesson/synapse-definition-transmission-quiz.html

[46] Using a Neural Network to Model the S&P 500. *Inovancetech* [online]. 2014 [cit. 2016-04-26]. Dostupné z: https://inovancetech.com/ann.html

[47] *Artificial Neural Networks Technology.* University Toronto [online]. [cit. 2016-04-26]. Dostupné z: http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html

[48] Explaining Neural Networks. *CVUT* [online]. [cit. 2016-04-26]. Dostupné z: http://radio.feld.cvut.cz/matlab/toolbox/nnet/preface.html

[49] Principles of Neural Networks. *Dr. Frank Dieterle* [online]. 2006 [cit. 2016-04-26]. Dostupné z: http://www.frank-dieterle.de/phd/2_7_1.html

[50] An Introduction to Encog Neural Networks for C#. *Codeproject* [online]. [cit. 2016-04-26]. Dostupné z: http://www.codeproject.com/Articles/54575/An-Introduction-to-Encog-Neural-Networks-for-C

11 Seznam ilustrací

12 Seznam tabulek