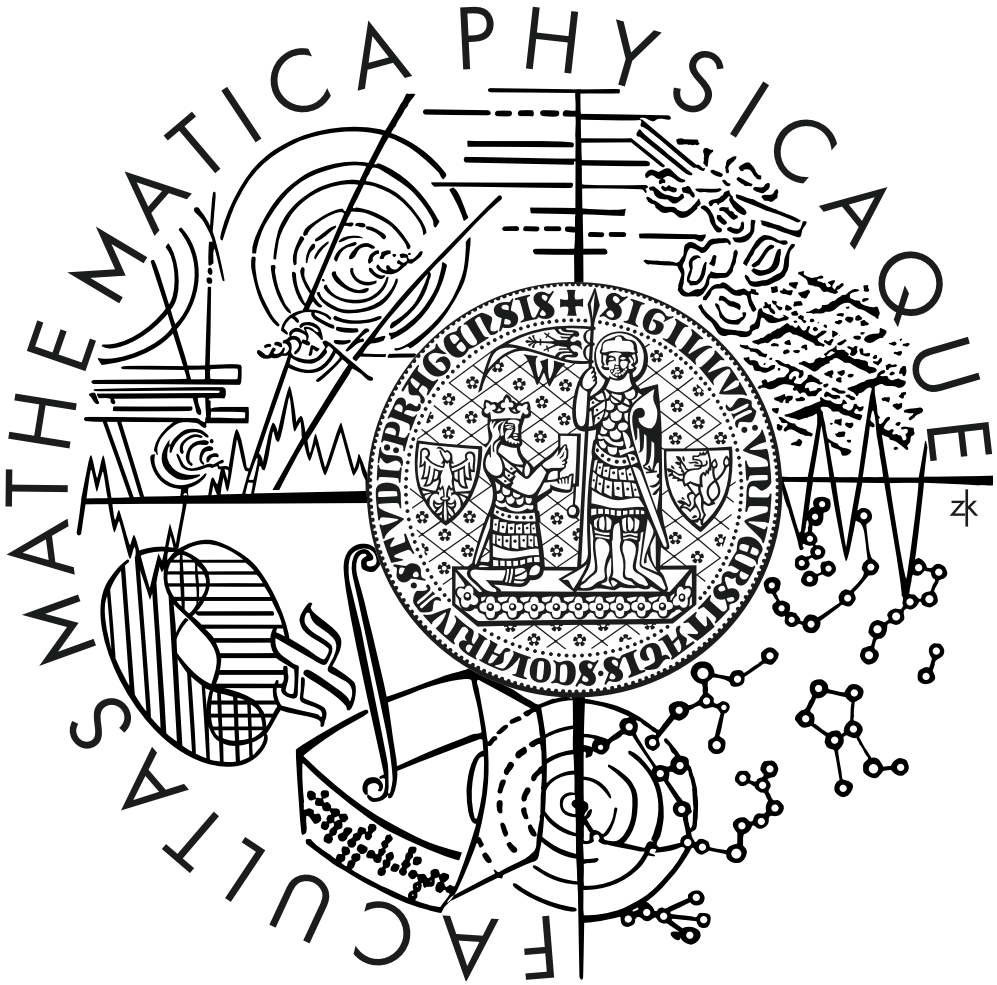
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Lukáš Hošek

Gradientní učení pro sítě hladce pulzních neuronů

Katedra softwarového inženýrství
Vedoucí diplomové práce: Doc. RNDr. Jiří Šíma, DrSc.
Studijní program: Informatika, teoretická informatika

Rád bych poděkoval vedoucímu mé diplomové práce doc. RNDr. Jiřímu Šímovi, DrSc. za ochotu a veškerou pomoc.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 4. 8. 2010 Lukáš Hošek

Obsah

Název práce: Gradientní učení pro sítě hladce pulzních neuronů
Autor: Lukáš Hošek
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: doc. RNDr. Jiří Šíma, DrSc.
e-mail vedoucího: sima@cs.cas.cz
Abstrakt: Sítě pulzních (spiking) neuronů představují biologicky více plausibilní alternativu k perceptronovým sítím mající velký potenciál pro zpracování časových řad. Nicméně doposud pro ně nebyl znám prakticky použitelný učící algoritmus. SpikeProp založený na gradientní metodě a jeho modifikace mají problém s principiální nespojitostí vzniku a zániku pulsů. Tuto otázku se snaží vyřešit nový netriviální gradientní učící algoritmus pro model hladce pulzních neuronů. Cílem práce je implementace a testování tohoto modelu a případné navržení jeho dalších vylepšení.
Klíčová slova: neuronová síť, spiking neuron, gradientní učící algoritmus

Title: Gradient learning for networks of smoothly spiking neurons
Author: Lukáš Hošek
Department: Department of Software Engineering
Supervisor: doc. RNDr. Jiří Šíma, DrSc.
Supervisor's e-mail: sima@cs.cas.cz
Abstract: Networks of spiking neurons present a biologically more plausible alternative to perceptron networks, having great potential for processing time series. However, as of now, no practically usable learning algorithm has been known. SpikeProp, based on a gradient descent method, and its modifications have a fundamental problem with discontinuity of spike creation and deletion. A new nontrivial gradient learning algorithm for a model of smoothly spiking neurons is proposed as a possible way to solve this problem. The goal of this work is to implement and test this model and eventually propose further improvements.
Keywords: neural network, spiking neuron, gradient learning algorithm

1 Úvod

Umělé neuronové sítě jsou výpočetní modely inspirované biologickými procesy a představují jeden z přístupů ke strojovému učení. Teoretický základ neuronových sítí položili už v roce 1943 Warren McCulloch a Walter Pitts (1), nicméně první prakticky použitelný model byl až model perceptronu, který navrhl Frank Rosenblatt (2) a odvodil pro něj algoritmus pro učení s učitelem.

Jednoduchý perceptron s algoritmem pro učení s učitelem byl prvním představitelem třídy modelů neuronových sít, do které patří i zřejmě nejúspěšnější model, perceptrony se sigmoidální přenosovou funkcí (dále jen sigmoidální neurony) organizované do vrstevnaté sítě učené pomocí algoritmu zpětného šíření chyby (back-propagation) (3) (4). Tento model byl detailně prozkoumán, byla vyvinuta celá škála jeho modifikací a učících heuristik a díky jeho snadné aplikovatelnosti na široké spektrum úloh se s ním dnes můžeme běžně setkat v praktických aplikacích, jako jsou klasifikační úlohy (automatizovaná kontrola kvality, filtrování spamu), předpovídání časových řad (předpovídání sluneční aktivity nebo vývoje ekonomických indexů) nebo v takových aplikacích jako rozpoznávání psaného písma.

Činnost biologických neuronů byla po mnoho let detailně zkoumána, a proto máme dnes poměrně dobrou představu o mechanismech, kterými probíhá přenos signálu mezi jednotlivými neurony. Jsme tak schopni sestavit velmi složité modely simulující biofyzikální procesy uvnitř neuronů. To ale není vždy žádoucí – pokud chceme sestavit model vhodný k aplikacím v oblasti umělé inteligence, pro které se dá snadno navrhnout učící algoritmus, volíme modely s jednoduchou dynamikou. Představitelem takového modelu jsou právě sítě jednoduchých perceptronů nebo perceptronů se sigmoidální přenosovou funkcí. Ty se inspirují biologickými neurony v tom smyslu, že jsou poskládané z jednoduchých jednotek, které samy o sobě mají velmi omezenou výpočetní kapacitu a složitější úlohy řeší, až pokud jsou organizovány do větších celků, ale od biologického originálu se odchylují ve většině konkrétních rysů – pro naši práci bude důležitý hlavně rozdíl v kódování informací. Opačným extrémem jsou modely pro simulování biofyzikálních procesů.

Pulzní (spiking) neurony, kterými se tato práce zabývá, představují model, který je dostatečně jednoduchý na to, aby byl použitelný v aplikacích pro umělou inteligenci a aby se dala snadno teoreticky zkoumat jeho dynamika, ale zároveň nabízí biologicky plausibilnější model reprezentace a přenosu signálu.

První model pulzních neuronů navrhl v roce 1952 Alan L. Hodgkin a Andrew Huxley (5). Tento model rozšiřuje simulaci o časovou doménu – informace se synapsemi mezi neurony nepřenášejí jako jedna spojitá veličina, ale jako série událostí (pulzů) nastávajících v daných časech. Dlouhou dobu panovalo obecné přesvědčení, že v

biologických neuronech je podstatná informace kódována frekvencí s jakou k pulsům dochází. Proto byl sigmoidální model považován za dostatečný; spojité hodnoty přenášené synapsemi se daly interpretovat jako zakódování frekvence pulzů. Nedávné poznatky v neurofyziologii ale ukazují, že přesné časování pulzů je také nositelem informace – experimenty na krysách ukazují, že podstatná informace v signálu může být kódována jako rozdíl v časování oproti signálu s konstantní frekvencí generovanému v hipokampu (6). Experimenty s neuroprostetikou pro obnovení pohyblivosti paralyzovaných končetin ukazují, že bez přesného časování pulsů není možné dosáhnout plynulých pohybů (7).

Díky těmto výsledkům se v posledních patnácti letech začali badatelé věnovat pulzním neuronům s obnoveným zájmem. V roce 2002 byl S. M. Bohtem navržen algoritmus *SpikeProp* (8) pro učení učitelem v upravené verzi modelu SRM_0 (9) pomocí adaptace vah v synaptických terminálech[1]. Algoritmus *SpikeProp* byl podrobněji zkoumán v (10), tato práce se mimo jiné zabývala rolí volby počátečních parametrů sítě a otázkou konvergence sítě s inhibičními vahami. Diplomová práce Karla Berkovce (11) se zabývá testováním modelu *SpikeProp* a zespojitěním vybavovací dynamiky sítě. Tato úprava umožňuje řádově urychlit učící proces. Jiný přístup představuje učení s učitelem se statistickým učícím kritériem (12) nebo učení pomocí evolučních metod (13). *SpikeProp* a jeho modifikace obvykle nedovolují více než jeden pulz na neuron, což zbavuje tento model možnosti použít frekvenční kódování.

Tato diplomová práce se zabývá modelem navrženým J. Šímou v (14). Tento model byl vytvořen se záměrem odstranit principiální nespojitost spojenou s procesem vytváření a zániku pulsů. Tato upravená verze modelu SRM_0 používá spojitou aproximaci schodové funkce vyjadřující čas posledního pulzu. Pulzy vznikají a zanikají spojitě sloučením s předchozím (v upravené verzi následujícím) pulzem. Tento model přirozeně umožňuje, aby neuron generoval v průběhu simulačního intervalu více než jeden pulz a umožňuje tak frekvenčně kódovat signál. K modelu je analyticky odvozen netriviální učící algoritmus. Tímto modelem se již zabývala bakalářská práce T. Janíka (15), ta ale narazila na problém s *nultým pulzem*, kvůli kterému se nebyla síť schopna naučit ani nejjednodušší úlohy. V prvním experimentu naší práce jsme tento problém zreplikovali a navrhujeme opravu modelu, která tento problém eliminuje.

Druhá kapitola shrnuje základní poznatky, které máme o biologických neuronech – jejich fyziologii a mechanismus excitace. Tato kapitola čerpá především z přehledu vypracovaného v (11).

Třetí kapitola popisuje předchozí přístupy k pulzním neuronům, věnuje se hlavně modelu SRM_0, jehož modifikací vznikl model, kterým se zabývá hlavní část této práce.

[1] Bohteho model používá synaptické terminály – každé dva neurony v sousedících vrstvách jsou spojeny více spoji, každý spoj má dva parametry – zpoždění, které je předem fixované, a váhu, která je proměnná a upravováním její hodnoty se síť učí.

Čtvrtá kapitola představuje model sítě hladce pulzních neuronů popsaný v (13), popisuje jeho architekturu a kompletní analytické odvození učícího pravidla.

V páté kapitole popisujeme provedené experimenty a navrhujeme několik modifikací, které mají řešit nedostatky modelu, které se během testování projevily.

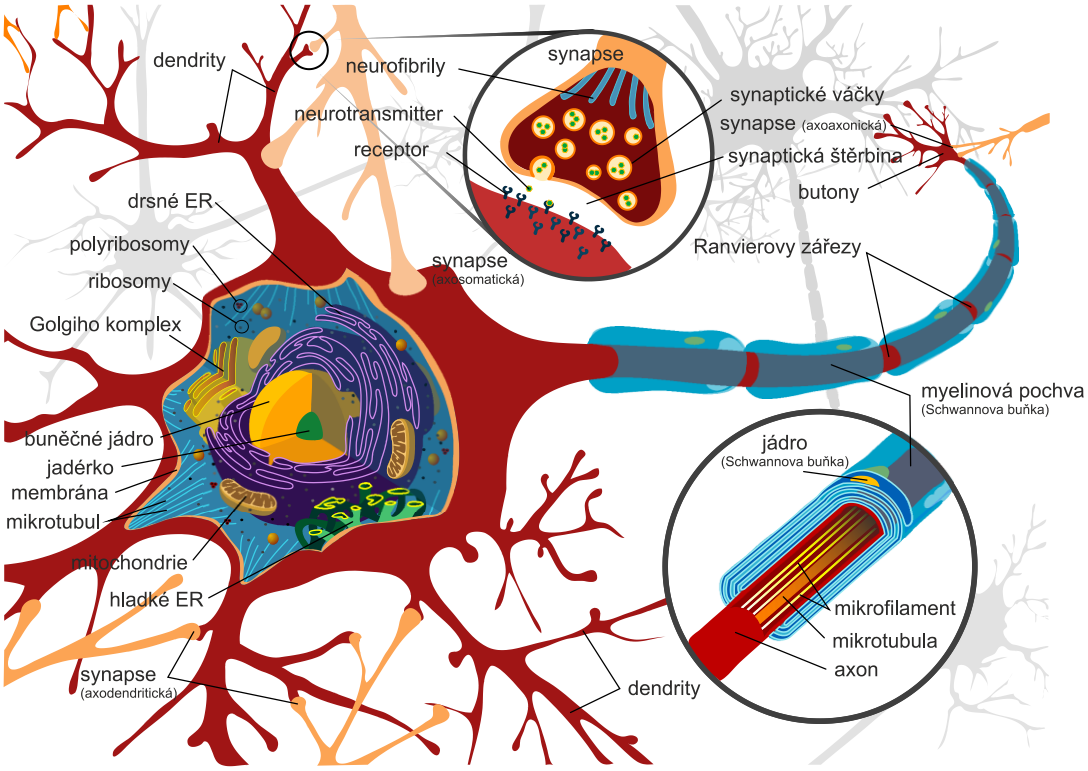
Šestá kapitola shrnuje dosažené výsledky a poznatky získané touto prací.

K diplomové práci je přiloženo CD s elektronickou verzí tohoto dokumentu a zdrojovými kódy programů, které byly použity při testování.

2 Biologický neuron

Neuron je typ buňky vyskytující se v těle většiny živočichů (všech s výjimkou houbovců a několika málo dalších primitivních živočišných druhů). Neurony jsou charakteristické hlavně elektrickou vzrušivostí a přítomností *synapsí*, což jsou spoje, kterými jsou neurony propojeny do sítí. U vyšších živočichů tvoří síť neuronů *centrální nervovou soustavu* – mozek, míchu a ganglie[2].

Existuje několik specializovaných typů neuronů: *Senzorické* neurony reagují na dotek, zvuk, světlo nebo jiné stimuly. Takové neurony jsou *unipolární* – mají jen jeden výběžek – axon, kterým předávají vzruchy dalším neuronům, ale samy žádné vzruchy nepřijímají. *Motorické* neurony reagují na přijímané vzruchy a kontrolují kontrakce svalů nebo ovlivňují činnost žláz. A konečně *interneurony* propojují ostatní neurony.



Obrázek 1: Stavba neuronu

Neurit neboli *axon*, je výběžek, kterým neuron předává vzruchy dalším neuronům. Po celé své délce s výjimkou konce je obalen *Schwannovou pochvou* a u myelinizovaných axonů i *myelinovou pochvou*, která funguje jako izolant. V periferní nervové soustavě je pochva přerušována Ranvierovými zářezy; v těchto místech může docházet k větvení axonů. Na konci se axon rozvětvuje a jednotlivá vlákna přecházejí do takzvaných *butonů*.

[2] Ganglie jsou nervová zauzlení, která u nižších živočichů tvoří hlavní část centrálního nervového systému (mozkový ganglion). U lidí jsou ganglie součástí periferní nervové soustavy; vyskytují se nejčastěji ve stěnách orgánů.

Dendrity jsou krátké výběžky, vystupují z těla neuronu typicky v oblasti neurocytu a nemají myelinovou pochvu. V místě odstupu od těla jsou tlusté a postupně se větví. Slouží k přijímání vstupních informací (nervových vzruchů). Dendrity jsou zakončené krátkými výběžky – *trny*. Těch může být řádově víc než dendritů a mohou se vytvářet a zanikat během celého života neuronu. Jejich význam zatím nebyl dostatečně prozkoumán, ale je pravděpodobné, že jejich počet a velikost souvisí s mechanismem učení.

Synapse, neboli mezineuronové spojení, je místo, kde se stýká buton větvící se z axonu neuronu, který vysílá vzruchy (*presynaptického neuronu*) s trnem dendritu neuronu, který vzruchy přijímá (*postsynaptického neuronu*). Mezi butonem a trnem dendritu je synaptická štěrbina o velikost 10 – 40 nm. Elektrický signál šířící se tělem neuronu spouští na synapsi sekreci *neurotransmiteru*. Neurotransmitery se váží na membráně postsynaptického neuronu na receptory, které podle typu neurotransmiteru otevírají různé iontové kanály a tím selektivně zvyšují permeabilitu membrány vůči specifickým iontům. Podle toho jestli synapse otevírá iontové kanály zvyšující nebo snižující elektrochemický potenciál buňky, rozdělujeme synapse na *excitační* a *inhibiční*.

Klidový potenciál buňky je přibližně -70 mV. Jeho hodnota závisí na podílu jednotlivých iontů uvnitř a vně membrány buňky. Přibližnou hodnotu klidového potenciálu můžeme spočítat Goldmannovo rovnicí (16) z koncentrací tří hlavních iontů Na^+, K^+ a Cl^- uvnitř a vně membrány:

$$\Delta\phi = \frac{RT}{F}\ln\frac{[K]_i + [Na]_i + [Cl]_e}{[K]_e + [Na]_e + [Cl]_i}$$

V hranatých závorkách jsou koncentrace iontů, indexy *i* a *e* označují koncentrace uvnitř, resp. vně buňky. *R* je univerzální plynová konstanta, *T* absolutní teplota a *F* Faradayův náboj.

2.1 Mechanismus vzniku nervového impulsu

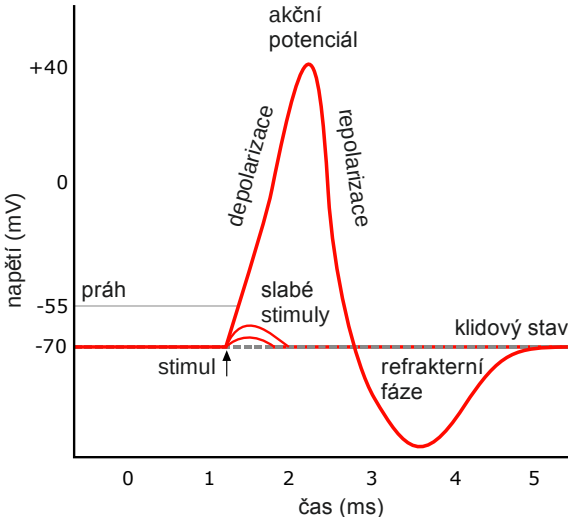
Při rovnovážném stavu jsou ionty K^+ puzeny ven z buňky chemickou silou; to je vyvažováno elektrickou silou, která ionty K^+ přitahuje dovnitř. Ionty Na^+ jsou do buňky přitahovány chemickou i elektrickou silou. Elektrochemický potenciál je udržován v rovnovážném stavu mechanismem *sodíko-draselné pumpy* (zvané též Na^+/K^+ ATPázy), která přesouvá buněčnou membránou ionty sodíku a draslíku proti koncentračnímu gradientu za spotřeby ATP.

Jak již bylo v předchozím textu zmíněno, excitační a inhibiční neurotransmitery mají schopnost navázat se na receptory v membráně a změnit její konformaci tak, že otevírají kanál propustný pro kationty nebo anionty. Tímto mechanismem jsou impulsy z presynaptického neuronu schopné ovlivňovat permeabilitu membrány. Průnikem

kladných iontů Na^+ do buňky v klidovém stavu vzniká *depolarizační* (též *excitační*) *potenciál*. Naopak průnik Cl^- má tlumivý efekt.

Depolarizace otevírá sodíkové a draselné kanály, kterými ionty proudí dovnitř, resp. ven z buňky. Pokud depolarizace nepřesáhne prahovou hodnotu (například při zvýšení potenciálu z -70 mV na -60 mV), ionty draslíku proudící ven vyváží proud sodíkových iontů dovnitř a membrána se repolarizuje zpět na klidovou hodnotu.

Pokud je ale depolarizace dostatečně silná, zvýšení potenciálu způsobené přílivem sodíkových iontů převáží inhibiční efekt odlivu draselných iontů (potenciál potřebný k překonání této hranice je přibližně -45 mV). Mechanismus pozitivní zpětné vazby otevře elektrosenzitivní sodíkové kanály a příliv sodíkových iontů způsobí, že potenciál vzroste k hodnotě blízké sodíkovému ekvilibriu (cca. +55 mV). Tato prudká změna elektrochemického potenciálu tvoří náběžnou hranu pulsu. Díky mechanismu pozitivní zpětné vazby dosáhne maximální potenciál během pulsu (zvaný též *akční potenciál*) vždy stejné absolutní hodnoty.



Obrázek 2: Nervový impuls

Neuron, který právě vyslal puls, není schopen během refrakterní fáze, kdy se sodíkové kanály ještě nevrátily do svého klidového stavu, vyslat další puls. V momentě, kdy se již část kanálů zotavila, je neuron schopen vyslat další puls, ale prahová hodnota, kterou musí elektrochemický potenciál překonat, aby nedošlo k předčasné repolarizaci je vyšší (kolem -30 mV).

V momentě, kdy jsou již všechny sodíkové kanály otevřeny, přestává působit pozitivní zpětná vazba, potenciál se již dále nezvyšuje, navíc stejný nárůst napětí, který sodíkové kanály otevřel, způsobí, že póry sodíkových kanálů se začnou pomalu uzavírat – sodíkové kanály se *inaktivují*. Současně s tím se otevřou elektrosenzitivní draslíkové kanály, které způsobí *repolarizaci* - prudký pokles potenciálu, sestupnou hranu pulsu.

Po skončení sestupné fáze pulzu nastává hyperpolarizace – během sestupné fáze se otevřelo mnohem víc kanálů zvyšujících permeabilitu draselných iontů než je otevřených za klidového stavu a ne všechny se zavřou okamžitě po návratu ke klidovému potenciálu. Neuron se tak těsně po vyslání pulsu dostane do hyperpolarizovaného stavu, ve kterém je hodnota elektrochemického potenciálu nižší než klidová a blíží draslíkovému ekvilibriu.

2.2 Šíření signálu a synaptický přenos

Akční potenciál se membránou axonu šíří pomocí pozitivní zpětné vazby sodíkových kanálů; každý sodíkový kanál funguje jako zesilovač, který na malou depolarizaci reaguje tak, že umožní tok iontů a tím vyvolá velkou depolarizaci. Tímto mechanismem se depolarizace šíří celým tělem axonu rychlostí až několik desítek metrů za sekundu.

U obratlovců se rychlost šíření signálu zvyšuje pomocí myelinové pochvy. Myelin je lipoprotein tvořící jednu ze základních složek bílé hmoty mozku a míchy a je příčinou jejich bílé barvy. Elektrický proud depolarizuje tím větší část membrány, čím méně náboje váže jednotka jejího povrchu. Protože myelin je izolant, zvyšuje dosah působení akčního potenciálu podél vlákna. Rozdíl v rychlosti vedení signálu mezi nemyelinizovaným a myelinizovaným vláknem je až o dva řády ve prospěch myelinizovaných.

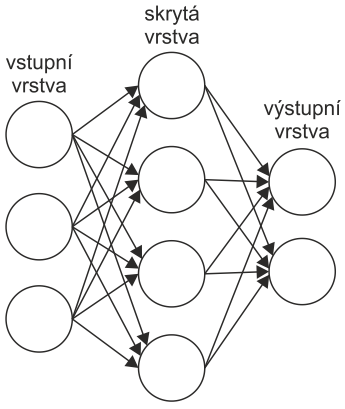
Když elektrický signál dorazí k zakončení axonu, způsobí vylití neurotransmiteru. Rychlost přenosu podráždění na synapsi je přibližně 1 ms. Neurotransmiter se v malé míře v nepravidelných intervalech vylévá i z nepodážděných zakončení.

3 Modely pulzních neuronů

Definující vlastnost, kterou se pulzní neurony liší od klasických modelů perceptronů, je rozšíření do časové domény. V klasických modelech probíhá předávání signálů mezi neurony v jednom okamžiku, u pulzních neuronů je předávaná informace funkcí času. Existují složité modely popsané diferenciálními rovnicemi snažící se věrně simulovat biologické neurony. Pro praktické aplikace je však vhodné zavést několik zjednodušení – například není vhodné povolit kódovat vstup jako libovolnou funkci času. Chování biologických neuronů je sice charakterizováno složitou dynamikou otevírání iontových kanálů a spojitých změn elektrochemického potenciálu v čase, ale s trochou představivosti můžeme od tohoto komplexního chování abstrahovat a představit si neurony jako jednotky předávající si události – pulzy, přičemž jeden pulz je charakterizován právě a jenom časem, kdy nastal (tj. nebudeme uvažovat žádnou sílu pulzu nebo dobu trvání pulzu). Toto zjednodušení umožňuje, aby se uživateli navenek jevila síť pulzních neuronů jako černá skřínka, do které zadává časy pulzů a dostává opět časy (a nikoliv složitě kódované libovolné funkce času). Některé modely zavádějí další zjednodušení, například model použitý v algoritmu *SpikeProp* umožňuje během simulace předat jednou synapsí maximálně jeden pulz.

3.1 Popis modelu SRM_0

Model SRM_0 je zjednodušená verze *Spike Response Modelu*[3]. SRM_0 samotné popisuje chování jednoho pulzního neuronu; pro naše potřeby budeme uvažovat o neuronu jako o jedné výpočetní jednotce v neuronové síti. Většinou pracujeme se sítěmi organizovanými do vrstev – jedna vstupní vrstva, nula nebo více skrytých vrstev a jedna výstupní vrstva; každý neuron je propojen se všemi neurony



Obrázek 3: Dopředná neuronová síť s jednou skrytou vrstvou

V modelu SRM_0 jsou informace zakódovány do akčních potenciálů neboli pulzů. Pro každý neuron j definujeme množinu Γ_j jeho bezprostředních předchůdců (*„presynaptických neuronů“*). Dále pro každý neuron j máme posloupnost $t_{j,1} \dots t_{j,s_j}$ časů, ve kterých nastávají pulzy (*„firing times“*). Tato posloupnost může být i nulové délky. Pro vstupní neurony je tato posloupnost zadána přímo uživatelem, pro

[3] SRM_0 se od obecného SRM liší absencí závislosti aktivační funkce na času od posledního pulzu.

všechny ostatní neurony je tato posloupnost definována jako posloupnost časů, ve kterých jejich vnitřní stavová proměnná zvaná potenciál neuronu (značíme u_j) zdola přesáhne práh ϑ_j. Potenciál neuronu pak je:

$$u_j(t) = \sum_{k=1\ldots s_j} \eta_j(t - t_{j,k}) + \sum_{i\in\Gamma_j} \sum_{k=1\ldots s_i} w_{ij}\varepsilon_{ij}(t - t_{i,k}) \tag{3.1}$$

První suma představuje inhibiční reakci neuronu při refrakterní fázi. Předpokládá se, že funkce η_j je nulová, kromě okamžiku, kdy její argument je malé kladné číslo – to odpovídá situaci, kdy je neuron j těsně po vyslání pulzu; v tom okamžiku by měla funkce n_j nabývat záporných hodnot a stlačovat tak potenciál neuronu j. Ve většině modelů se tento člen opomíjí, v Bohteho modelu (8) z toho důvodu, že povoluje maximálně jeden pulz na neuron: pokud potenciál překročí práh, pak je $t_{j,1}$ definováno jako čas, kdy u_j poprvé překročí ϑ_j a žádný další pulz již nenastává.

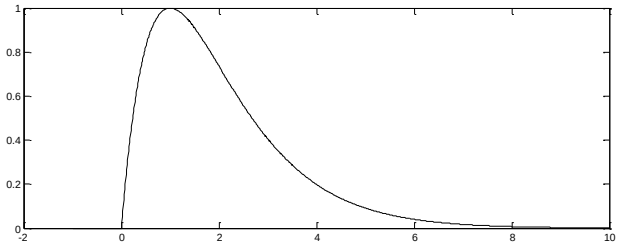
Druhá sada sum představuje reakci neuronu na pulzu přicházející z presynaptických neuronů. ε_{ij} je aktivační funkce, která reaguje, pokud je její argument malé nezáporné číslo, to jest těsně poté, co z presynaptického neuronu přišel pulz. Člen w_{ij}, kterým se násobí aktivační funkce, se nazývá *váha*. Oindexování nám umožňuje použít pro každou synapsi zcela libovolnou aktivační funkci, ale původní záměr, proč je aktivační funkce na každé synapsi rozdílná, je zavést do modelu simulaci *zpoždění* při šíření signálu axonem.

Pokud zanedbáme refrakterní fázi a předpokládáme, že aktivační funkce se liší pouze ve zpoždění, dostáváme zjednodušenou verzi vzorce (3.1):

$$u_j = \sum_{i\in\Gamma_j} \sum_{k=1..s_i} w_{ji}\varepsilon(t - t_{i.k} - d_{ji}) \tag{3.2}$$

Často používaná aktivační funkce je

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{1-\frac{t}{\tau}} & pro\ t \geq 0 \\ 0 & pro\ t < 0 \end{cases} \tag{3.3}$$



Obrázek 4: Graf aktivační funkce (3.3)

3.2 Vlastnosti modelu SRM_0

Pulzní neuron dokáže simulovat práci jednoduchého perceptronu s binárně kódovanými vstupy a Heavisideovo přechodovou funkcí (ostrou nelinearitou): mějme pulzní neuron j, předpokládejme, že všechny jeho aktivační funkce ε_{ji} jsou stejné, nezáporné a nabývají globálního maxima (označíme ε_{max}) a všechny presynaptické neurony, které vyšlou pulz, ho vyšlou ve stejný okamžik T_{input}. Původní binární vstupy zakódujeme tak, že presynaptický neuron vysílá, resp. nevysílá pulz v čase T_{input}, pokud původní vstup byl 1, resp. 0. Původní váhy jednoduchého preceptronu označíme α_{ji} a původní práh ϑ. Práh pulzního neuron volíme stejný, váhy pulzního neuronu přepočteme vzorcem $w_{ji} = \alpha_{ji}/\varepsilon_{max}$.

Protože jsou všechny aktivační funkce ε_{ji} stejné, nabývají maxima ve stejný čas. Proto je podmínka pro pulzní neuron

$$\sum_{i\ vyšle\ signál} w_{ji}\varepsilon_{max} \geq \vartheta$$

ekvivalentní podmínce pro perceptron

$$\sum_{na\ i-tém\ vstupu\ je\ 1} \alpha_{ji} \geq \vartheta$$

Pulzní neuron proto vysílá signál právě tehdy, když na výstupu perceptronu je 1.

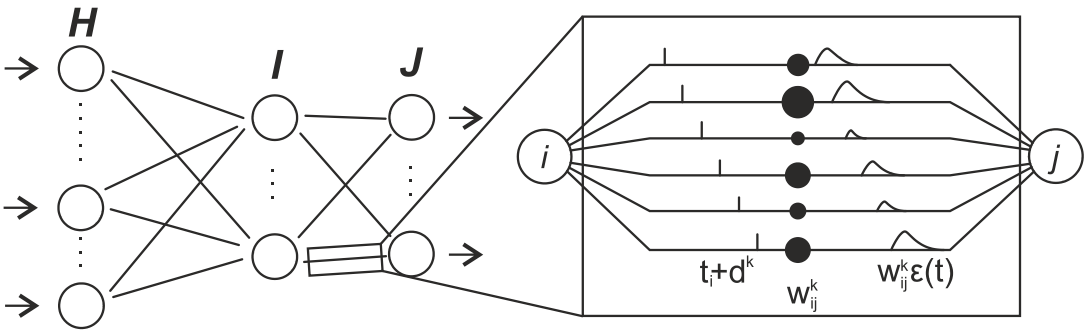
Simulace vrstevnaté sítě je ale nepoměrně komplikovanější úkol – pokud jsme simulovali jeden perceptron, spoléhali jsme se na to, že všechny vstupy přijdou ve stejný čas. O času výstupního pulzu nemáme ale zaručeno nic; jeden neuron může vysílat výstupní pulzy v různé časy v závislosti na tom, jak silné jsou vstupní stimuly a jak brzo překročí potenciál prahovou hodnotu. Pro simulaci vrstevnatých perceptronových sítí je třeba použít nějaký synchronizační mechanismus. Ten se dá vytvořit přidáním dalších neuronů do sítě, případně úpravou modelu tak, aby potenciály neuronů reagovaly na nosný signál. Podrobněji viz (17) (18).

O pulzních neuronech máme hodně teoretických poznatků: jak jsme předvedli v předchozích odstavcích, dokáží nahradit jednoduché perceptrony. V (19) je dokázáno, že problém konzistence neuronu (nalezení parametrů sítě takových, že neuron bude dávat výsledky konsistentní se všemi trénovacími vzory) je NP-úplný a navíc, pokud $RP \neq NP$, potom jeden pulzní neuron není PAC-naučitelný. Přesto jsou ale sítě pulzních neuronů schopny naučit se širokou škálu funkcí. Byly vyvinuty metody pro učení s učitelem i bez učitele, založené na lineárně algebraických (20), statistických (21) (12) nebo evolučních (13) metodách, metody založené na Hebbovském učení (22) (23), nebo metody pro učení speciálních typů úloh, jako je rozpoznávání shod.

3.3 SpikeProp

SpikeProp je metoda navržená v (8) pro učení s učitelem založená podobně jako back-propagation (3) (4) na explicitním vyhodnocování gradientu a zpětném šíření chyby. Tato metoda zavádí několik modifikací a omezení modelu SRM_0: zaprvé, neuron může během simulačního intervalu vyslat maximálně jeden pulz. Zadruhé, během učení se adaptují váhy; pro zpoždění žádné adaptační pravidlo neexistuje.

Nemožnost upravovat zpoždění během učení je kompenzována zavedením synaptických terminálů:



Obrázek 5: Síť se synaptickými terminály

Jedno spojení propojující presynaptický neuron i s postsynaptickým neuronem j sestává z předem stanoveného počtu m synaptických terminálů. Každý terminál k slouží jako samostatná komponenta spojení, která má vlastní váhu w_{ji}^k a zpoždění d_{ji}^k. Počet terminálů v jednom spojení je typicky stejný pro všechny spojení v síti a zpoždění v jednotlivých synaptických terminálech jednoho spojení jsou odstupňována v pravidelných intervalech, *SpikeProp* však s tímto předpokladem explicitně nepočítá.

Nevážený příspěvek jednoho synaptického terminálu k potenciálu neuronu je dán vzorcem

$$y_{ji}^k(t) = \varepsilon(t - t_i - d_{ji}^k)$$

v případě, že presynaptický neuron vysílá pulz. V případě, že pulz nevysílá, je příspěvek po čas celého simulačního intervalu nulový. ε je aktivační funkce uvedená ve vzorci (3.3). Upravená verze vzorce (3.2) pro sítě se synaptickými terminály potom je

$$u_j(t) = \sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ji}^k y_{ji}^k(t)$$

Učící algoritmus je odvozen pro plně propojenou vrstevnatou dopřednou síť. Vstupní vrstvu označíme H, skryté vrstvy I a výstupní vrstvu J. Cílem je naučit síť tak, aby po předložení vstupního vzoru $[t_1, \dots t_h]$, kde t_i označuje pro vstupní neuron $i \in H$ jeden čas jeho pulzu, vydala na výstupních neuronech výstupy $[t_1^d, \dots t_j^d]$, kde t_i^d

označuje požadovaný čas pulzu i - tého výstupního neuronu. Označíme si $[t_1^a, \dots t_j^a]$ skutečné výstupy (Problém může nastat v případě, že výstupní neuron pulz vůbec nevyšle. To se dá vyřešit vysláním implicitního pulzu vyslaného v čase větším než je délka simulačního intervalu.).

Z požadovaných a skutečných časů pulzů výstupních neuronů vyjádříme chybu na předkládaném učícím vzoru:

$$E = \frac{1}{2}\sum_{j \in J}\left(t_j^a - t_j^d\right)^2$$

Je možné použít jiné chybové funkce, například entropii.

S váhou každého synaptického terminálu se pracuje samostatně a adaptuje se následujícím pravidlem, které mění váhu v protisměru lokálního gradientu:

$$w_{ji}^k(T+1) = w_{ji}^k(T) - \eta \frac{\partial E}{\partial w_{ji}^k}$$

T zde označuje číslo iterace učícího algoritmu. η je parametr učení (učící konstanta). Derivace chyby podle váhy se dá rozepsat podle pravidla o derivování složených funkcí:

$$\frac{\partial E}{\partial w_{ji}^k} = \frac{\partial E}{\partial t_j}\left(t_j^a\right)\frac{\partial t_j}{\partial u_j(t)}\left(t_j^a\right)\frac{\partial u_j(t)}{\partial w_{ji}^k}\left(t_j^a\right)$$

Ve faktorech na pravé straně je t_j vyjádřeno jako funkce potenciálu u_j v okolí $t = t_j^a$. Protože druhý člen se nedá analyticky spočítat, pro zjednodušení výpočtu gradientu se funkce u_j aproximuje lineární funkcí, takže $\partial t_j / \partial u_j$ se považuje za konstantu. Odvozené vzorečky vypadají následovně:

$$\frac{\partial E}{\partial w_{ji}^k} = y_{ji}^k(t)\delta_j$$

kde δ_j se pro neurony ve výstupní vrstvě spočítá:

$$\delta_j = \frac{-(t_j^a - t_j^d)}{\sum_{i \in \Gamma_j}\sum_k w_{ji}^k \frac{\partial y_{ji}^k(t)}{\partial t}}$$

a pro skryté neurony:

$$\delta_j = \frac{\sum_{i \in \Gamma^j}\delta_i \sum_k w_{ji}^k \frac{\partial y_{ji}^k(t)}{\partial t}}{\sum_{i \in \Gamma_j}\sum_k w_{ji}^k \frac{\partial y_{ji}^k(t)}{\partial t}}$$

Algoritmus *SpikeProp* byl testován v (10) (24). Ukazuje se, že výkon algoritmu kriticky závisí na inicializaci vah. V původní verzi algoritmu se počítá pouze s kladnými

váhami; (10) ukazuje, že i se zápornými vahami může algoritmus konvergovat. V (24) je navržena modifikace, která přidává do adaptačního pravidla setrvačnost. V (25) je zobecněn algoritmus SpikeProp pro rekurentní sítě. V (11) je navržena modifikace se spojitou aproximací času posledního pulzu, která výrazně urychluje konvergenci algoritmu.

Jedním z principiálních problémů algoritmu *SpikeProp* je umírání neuronů – v momentě, kdy neuron nevydává žádný pulz, se již jeho směrem nepropaguje žádná chyba, váhy se přestanou adaptovat a neuron se již z tohoto stavu nikdy nezotaví. To je způsobeno nespojitostí procesu vytváření a zanikání pulzu.

Navíc *SpikeProp* počítá pouze s časem prvního pulzu. To drasticky limituje schopnost sítě pracovat s časově kódovanými daty. Jak bylo vysvětleno v úvodu, informace v biologických neuronech je kódována mimo jiné frekvencí pulsů. Omezení na jeden pulz frekvenční kódování znemožňuje. Bohte (8) místo toho navrhuje kódovat spojité veličiny pomocí populací neuronů metodou takzvaných Gaussovských receptivních polí. V následující kapitole popisujeme upravený model a učící algoritmus, který se snaží většinu zmíněných problémů přirozeně řešit.

4 Sítě hladce pulzních neuronů

Model sítě hladce pulzních neuronů byl navržen v (14). Základním rozdílem, kterým se tento model liší od modelu používaného ve *SpikeProp*, je kompletně spojitá a diferencovatelná dynamika sítě, včetně procesu vytváření a zanikání pulzů adaptací parametrů. To umožňuje odvodit učící pravidlo pro adaptaci parametrů pomocí explicitního analytického výpočtu gradientu. Touto architekturou se zatím zabývala pouze bakalářská práce (15), ve které je odvozeno modifikované učící pravidlo pomocí metody sdružených gradientů. Model se však nebyl schopen naučit ani nejjednodušší vzory kvůli problému s formálně definovaným nultým pulzem, který se z každého neuronu propaguje dál do výstupů. V následujícím textu mimo jiné navrhujeme řešení tohoto problému.

4.1 Popis modelu

Neurony jsou organizovány do dopředné vrstevnaté neuronové sítě. Množinu všech neuronů označíme V. Některé z těchto neuron slouží jako *vstupní* (označíme $X \subseteq V$) nebo jako *výstupní* ($Y \subseteq V$). Zbylé neurony nazýváme *skryté*. Pro každý neuron j označíme jako $j_{\leftarrow}$ množinu všech *presynaptických* neuronů, ze kterých vede spoj do j. Podobně označíme $j_{\rightarrow}$ množinu všech *postsynaptických* neuronů, do kterých vede spoj z j. Každý spoj má přiřazenu váhu w_{ji} (j je postsynaptický neuron, i presynaptický) a zpoždění d_{ji}. Kromě toho má každý neuron j přiřazen *bias* w_{j0}.

Dále si nadefinujeme pomocnou spojitou, dvakrát diferencovatelnou neklesající funkci, které bude sloužit jako aproximace schodové funkce. Hledáme funkci σ tří parametrů $\alpha \leq \beta$ a $\delta > 0$, pro kterou platí $\sigma(x) = \alpha$ pro $x \leq 0$, $\sigma(x) = \beta$ pro $x \geq \delta$ a první a druhé derivace splňují $\sigma'(0) = \sigma'(\delta) = \sigma''(0) = \sigma''(\delta)$. Pro normalizované $\sigma_0 = \sigma(0,1,\delta)$ na intervalu $[0;\delta]$ tyto podmínky splňuje například primitivní funkce

$$\int Ax^2(x-\delta)^2 dx = A\left(\frac{x^5}{5} - 2\delta\frac{x^4}{4} + \delta^2\frac{x^3}{3}\right) + C$$

Z $\sigma_0(0) = 0$ a $\sigma_0(\delta) = 1$ vychází konstanty $C = 0$ a $A = 30/\delta^5$. Obecné σ na intervalu $[0;\delta]$ získáme ze σ_0 vzorcem $\sigma(\alpha,\beta,\delta;x) = (\beta-\alpha)\sigma_0(x) + \alpha$. Kompletní definice funkce je:

$$\sigma(\alpha,\beta,\delta;x)=\begin{cases}\alpha & pro\ x<0\\ (\beta-\alpha)\left(\left(\frac{6x}{\delta}-15\right)\frac{x}{\delta}+10\right)\left(\frac{x}{\delta}\right)^3+\alpha & pro\ 0\le x\le\delta\\ \beta & pro\ x>\delta\end{cases} \tag{4.1}$$

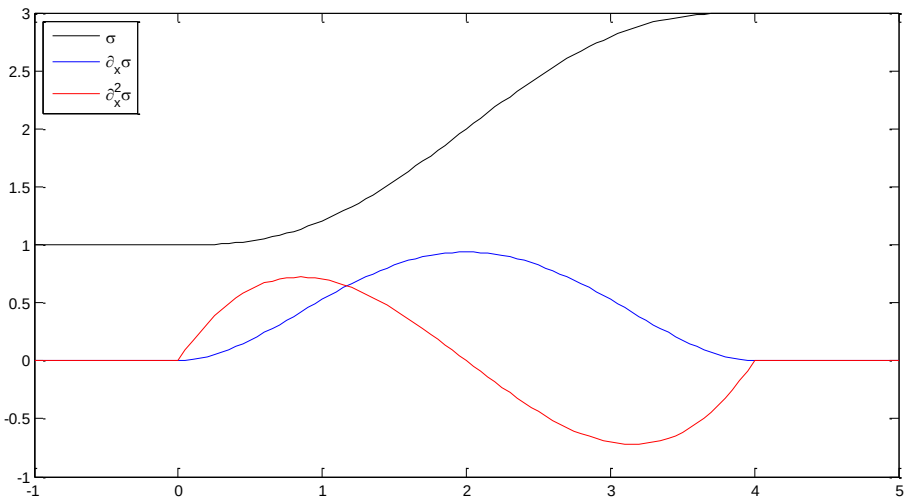
Parciální derivace funkce σ jsou:

$$\sigma'(x)=\frac{\partial}{\partial x}\sigma(x)=\begin{cases}\frac{30}{\delta}(\beta-\alpha)\left(\left(\frac{x}{2}-2\right)\frac{x}{\delta}+1\right)\left(\frac{x}{\delta}\right)^2 & pro\ 0\le x\le\delta\\ 0 & jinak\end{cases} \tag{4.2}$$

$$\sigma''(x)=\frac{\partial^2}{\partial x}\sigma(x)=\begin{cases}\frac{60}{x^2}(\beta-\alpha)\left(\left(\frac{2x}{\delta}-3\right)\frac{x}{\delta}+1\right)\frac{x}{\delta} & pro\ 0\le x\le\delta\\ 0 & jinak\end{cases} \tag{4.3}$$

$$\frac{\partial}{\partial\alpha}\sigma(x)=\begin{cases}1 & pro\ x<0\\ 1-\left(\left(\frac{6x}{\delta}-15\right)\frac{x}{\delta}+10\right)\left(\frac{x}{\delta}\right)^3 & pro\ 0\le x\le\delta\\ 0 & pro\ x>\delta\end{cases} \tag{4.4}$$

$$\frac{\partial}{\partial\beta}\sigma(x)=\begin{cases}0 & pro\ x<0\\ \left(\left(\frac{6x}{\delta}-15\right)\frac{x}{\delta}+10\right)\left(\frac{x}{\delta}\right)^3 & pro\ 0\le x\le\delta\\ 1 & pro\ x>\delta\end{cases} \tag{4.5}$$



Obrázek 6: Graf funkce $\sigma(1{,}3{,}4;x)$ a jejích derivací

Logistická sigmoida je definována vzorcem

$$P(\lambda;x)=\frac{1}{1+e^{-\lambda x}} \tag{4.6}$$

λ je parametr určující strmost přechodu. V naší implementaci používáme $\lambda = 4$. Derivace funkce P je definována následovně:

$$P'(x) = \lambda P(x)\big(1 - P(x)\big) \tag{4.7}$$

Každý neuron vysílá posloupnost p_j pulzů v časech $0 < t_{j1} < t_{j2} < \cdots < t_{jp_j} < T$ (T označuje délku simulačního intervalu). K tomu formálně dodefinujeme pulzy $t_{j0} = 0$ a $t_{jp_j+1} = T$. Pro vstupní neurony ($j \in X$) jsou pulzy dány uživatelem, u ostatních neuronů jsou pulzy definovány jako časy, kdy během simulačního intervalu excitační funkce neuronu ξ_j překročí zdola 0, tj,:

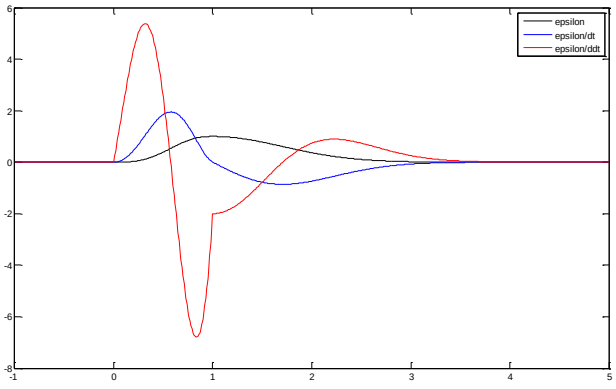
$$\{0 \le t \le T | \xi_j(t) = 0 \,\&\, \xi_j'(t) > 0\} = \left\{t_{j1}, t_{j2}, \ldots t_{jp_j}\right\}$$

Excitační funkce je definována jako vážená suma odezev z presynaptických neuronů zpožděných o čas d_{ji}. Funkční hodnotu funkce $\xi_j(t)$ budeme nazývat *potenciál* neuronu j v čase t. Je definována následovně:

$$\xi_j(t) = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji}\varepsilon\left(t - d_{ji} - \tau_i\big(t - d_{ji}\big)\right) \tag{4.8}$$

ε je funkce modelující odezvu neuronu na příchozí pulz. Její parametr si můžeme představit jako uplynulý čas od posledního pulzu a její funkční hodnota je nevážený přírůstek k potenciálu. Její definice je:

$$\varepsilon(t) = e^{-(t-1)^2}\,\sigma_0(t) \tag{4.9}$$



Obrázek 7: Graf funkce ε a jejích derivací

Funkce $\tau_i(t)$ z vzorečku (4.8) je spojitá (a dvakrát diferencovatelná) aproximace schodové funkce vyjadřující čas posledního pulzu předcházejícího času t. Před tím, než

jí popíšeme, musíme ještě definovat transformaci časů pulzů. Posloupnost časů $t_{j1} < t_{j2} < \cdots < t_{jp_j}$ pro každý neuron j transformujeme na posloupnost $\widetilde{t_{j0}} = 0 < \widetilde{t_{j1}} < \widetilde{t_{j2}} < \cdots < \widetilde{t_{jp_j}} < T = \widetilde{t_{j,p_j+1}}$ postupně od prvního po p_j – tý vzorcem

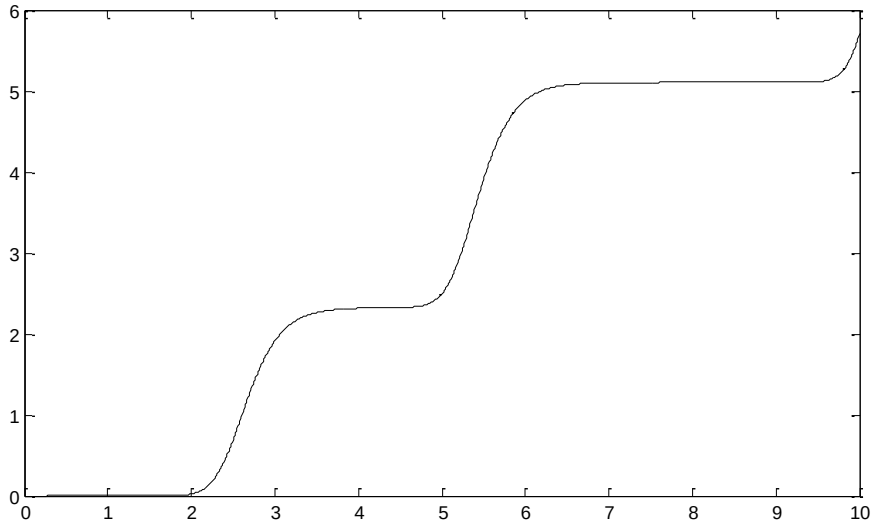
$$\widetilde{t_{js}} = \begin{cases} t_{js} & pro\ j \in X \\ \sigma\left(\widetilde{t_{j,s-1}}, t_{js}, \delta; \xi_j'(t_{js})\right) & pro\ j \notin X \end{cases} \tag{4.10}$$

Cílem této transformace je zespojitění procesu tvorby a zániku pulzů – pokud budeme během procesu například zmenšovat váhu, abychom odstranili nežádoucí pulz, nestane se, že by pulz najednou po snížení váhy zmizel. Místo toho se bude postupně přesouvat k předchozímu pulzu, protože bod, ve kterém funkce ξ bude protínat nulu, bude stále blíže jejímu lokálnímu extrému, ve kterém je hodnota funkce ξ' nulová. Podobně vytvářející se pulz se místo skokového objevení plynule oddělí od předcházejícího pulzu.

Nyní máme všechno potřebné pro definování funkce τ_j:

$$\tau_j(t) = \sum_{s=1}^{p_j+1} \left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right) P^C\left(t - \widetilde{t_{js}}\right) \tag{4.11}$$

Exponent C v tomto vzorečku je konstanta, kterou můžeme regulovat strmost schodových přechodů (v naší implementaci používáme $C = 4$).



Obrázek 8: Funkce τ pro neuron s pulzy nastávajícími v časech $\widetilde{t_1} = 3$ a $\widetilde{t_2} = 6$.

Zbývající rovnice potřebné pro kompletní popsání aktivní dynamiky sítě jsou

$$\xi_j'(t) = \sum_{i \in j_{\leftarrow}} w_{ji} \varepsilon'\left(t - d_{ji} - \tau_i\left(t - d_{ji}\right)\right)\left(1 - \tau_i'\left(t - d_{ji}\right)\right) \tag{4.12}$$

$$\varepsilon'(t) = e^{-(t-1)^2}\left(\sigma_0'(t) - 2(t-1)\sigma_0(t)\right) \tag{4.13}$$

$$\tau_i'(t) = \frac{\partial}{\partial t}\tau_i(t) = C\lambda \sum_{s=1}^{p_i+1} \left(\widetilde{t_{is}} - \widetilde{t_{i,s-1}}\right) P^C\left(t - \widetilde{t_{is}}\right)\left(1 - P\left(t - \widetilde{t_{is}}\right)\right) \tag{4.14}$$

4.2 Učení

Trénovací vzor přiřazuje každému vstupnímu neuronu $i \in X$ posloupnost vstupních pulzů $0 = t_{i0} < t_{i1} < t_{i2} < \cdots < t_{i,p_i} < T = t_{i,p_i+1}$ a každému výstupnímu neuronu $j \in Y$ posloupnost *požadovaných výstupů* $0 = \varrho_{j0} < \varrho_{j1} < \varrho_{j2} < \cdots < \varrho_{j,q_j} < \varrho_{j,q_j+1} = T$. Cílem učení je upravit parametry sítě – váhy a zpoždění – tak, aby se skutečné výstupy co nejvíce podobaly požadovaným výstupům. Odchylka sítě na jednom trénovacím vzoru je vyjádřena chybou

$$E(w,d) = \frac{1}{2}\sum_{j\in Y}\sum_{s=0}^{q_j}\left(\hat{o}_j\left(\varrho_{j,s+1}\right) - \varrho_{js}\right)^2 \tag{4.15}$$

Chyba na trénovacím vzoru je funkce vektoru všech vah sítě w a vektoru všech zpoždění sítě d. $\tau_j\left(\varrho_{j,s+1}\right)$ je aproximace času posledního pulzu předcházejícího času $\varrho_{j,s+1}$.

Učící pravidlo se snaží minimalizovat chybu pomocí gradientního sestupu. Pro každý nevstupní neuron j se v jednom kroku algoritmu upraví váhy a zpoždění následujícím způsobem:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} - \alpha\frac{\partial E}{\partial w_{ji}}\left(w^{(t-1)}, d^{(t-1)}\right) pro\ i \in j_{\leftarrow} \cup \{0\} \tag{4.16}$$

$$d_{ji}^{(t)} = d_{ji}^{(t-1)} - \alpha\frac{\partial E}{\partial d_{ji}}\left(w^{(t-1)}, d^{(t-1)}\right) pro\ i \in j_{-} \leftarrow \tag{4.17}$$

$w^{(i)}$, resp. $d^{(i)}$ značí vektor vah a zpoždění během i-té iterace učícího algoritmu. $w^{(0)}$ a $d^{(0)}$ jsou vhodně zvolené počáteční váhy (co to znamená „vhodně“ rozebíráme v kapitole věnované experimentům). α je učící konstanta.

Na rozdíl od klasického algoritmu back-propagation, ve kterém se z každého neuronu šíří zpět pouze jedna hodnota δ, potřebujeme v našem algoritmu šířit několik parciálních derivací v určitých časových okamžicích. Proto zavedeme pro každý nevstupní neuron j seznam P_j délky m_j trojic $(\pi_{jc}, \pi'_{jc}, u_{jc})$, kde $c = 1, \ldots m_j$.

π_{jc} a π'_{jc} označuje hodnoty parciálních derivací v čase u_{jc} takové, že z nich lze vypočítat parciální derivace chyby podle parametrů sítě:

$$\frac{\partial E}{\partial w_{ji}} = \sum_{c=1}^{m_j} \left(\pi_{jc} \cdot \frac{\partial}{\partial w_{ji}} \tau_j(u_{jc}) + \pi'_{jc} \cdot \frac{\partial}{\partial w_{ji}} \tau'_j(u_{jc}) \right) \; pro \; i \in j_{\leftarrow} \cup \{0\} \tag{4.18}$$

$$\frac{\partial E}{\partial d_{ji}} = \sum_{c=1}^{m_j} \left(\pi_{jc} \cdot \frac{\partial}{\partial d_{ji}} \tau_j(u_{jc}) + \pi'_{jc} \cdot \frac{\partial}{\partial d_{ji}} \tau'_j(u_{jc}) \right) \; pro \; i \in j_{\leftarrow} \tag{4.19}$$

Pokud máme dvě trojice $(\pi_{jc_1}, \pi'_{jc_1}, u_{jc})$ a $(\pi_{jc_2}, \pi'_{jc_2}, u_{jc})$ nastávající okamžik u_{jc}, můžeme je sloučit do jedné trojice $(\pi_{jc_1} + \pi_{jc_2}, \pi'_{jc_1} + \pi'_{jc_2}, u_{jc})$. Pokud máme trojici $(\pi_{jc}, \pi'_{jc}, u_{jc})$, kde $\pi_{jc} = \pi'_{jc} = 0$, můžeme tuto trojici vynechat.

Pro výstupní neurony $j \in Y$ můžeme odvodit výpočet seznamu P_j přímo z (4.15):

$$P_j = \{(\tau_j(\varrho_{j,s+1}) - \varrho_{js};\, 0;\, \varrho_{j,s+1}) |\, s = 0,1, \dots q_j\} \tag{4.20}$$

Pro ostatní neurony $j \in V \backslash X$ ve skrytých vrstvách odvodíme proceduru pro rekurzivní výpočet seznamu ze seznamů P_i neuronů z následující vrstvy ($i \in j_{\rightarrow}$) pomocí parciálních derivací

$$\frac{\partial}{\partial w_{il}} \tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) \cdot \frac{\partial \widetilde{t_{js}}}{\partial w_{il}} \tag{4.21}$$

$$\frac{\partial}{\partial w_{il}} \tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) \cdot \frac{\partial \widetilde{t_{js}}}{\partial w_{il}} \tag{4.22}$$

A z (4.7), (4.11) a (4.14) odvodíme:

$$\frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) = P^C\left(t - \widetilde{t_{js}}\right) \left(1 - C\lambda\left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right)\left(1 - P\left(t - \widetilde{t_{js}}\right)\right)\right) - P^C\left(t - \widetilde{t_{j,s+1}}\right) \tag{4.23}$$

$$\begin{aligned} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) = C\lambda \Bigg(& \left(\left(1 - C\lambda\left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right)\left(1 - P\left(t - \widetilde{t_{js}}\right)\right)\right)\right. \\ & \left. + \lambda\left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right) P\left(t - \widetilde{t_{js}}\right)\right) P^C\left(t - \widetilde{t_{js}}\right)\left(1 - P\left(t - \widetilde{t_{js}}\right)\right) \\ & - P^C\left(t - \widetilde{t_{j,s+1}}\right)\left(1 - P\left(t - \widetilde{t_{j,s+1}}\right)\right)\Bigg) \end{aligned} \tag{4.24}$$

Z (4.10) a (4.12) odvodíme:

$$\frac{\partial \widetilde{t_{js}}}{\partial w_{il}} = \frac{\partial \widetilde{t_{js}}}{\partial \widetilde{t_{j,s-1}}} \cdot \frac{\partial \widetilde{t_{j,s-1}}}{\partial w_{il}} + \left(\frac{\partial \widetilde{t_{js}}}{\partial t_{js}} \cdot \frac{\partial t_{js}}{\partial \tau_i} + \frac{\partial \widetilde{t_{js}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau_i} \right) \cdot \frac{\partial \tau_i}{\partial w_{il}} + \frac{\partial \widetilde{t_{js}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau'_i} \cdot \frac{\partial \tau'_i}{\partial w_{il}} \quad (4.25)$$

Ve vzorečku (4.25) se vyskytují parciální derivace

$$\frac{\partial \widetilde{t_{js}}}{\partial \widetilde{t_{j,s-1}}} = \begin{cases} \frac{\partial}{\partial \alpha} \sigma\left(\xi'_j(t_{js})\right) & pro\ s > 1 \\ 0 & pro\ s = 1 \end{cases} \quad (4.26)$$

$$\frac{\partial \widetilde{t_{js}}}{\partial t_{js}} = \left(\frac{\partial}{\partial \beta} + \xi''_j(t_{js}) \frac{\partial}{\partial x} \right) \sigma\left(\widetilde{t_{j,s-1}}, t_{js}, \delta; \xi'_j(t_{js})\right) \quad (4.27)$$

Derivace (4.12) je

$$\xi''_j(t) = \sum_{i \in j_{\leftarrow}} w_{ji} (\varepsilon''\left(t - d_{ji} - \tau_i(t - d_{ji})\right)\left(1 - \tau'_i(t - d_{ji})\right)^2 - \varepsilon'\left(t - d_{ji} - \tau_i(t - d_{ji})\tau''_i(t - d_{ji})\right) \quad (4.28)$$

Derivace (4.13) a (4.14) jsou

$$\varepsilon''(t) = e^{-(t-1)^2}\left(\sigma''_0(t) - 4(t-1)\sigma'_0(t) + (4(t-1)^2 - 2)\sigma_0(t)\right) \quad (4.29)$$

$$\tau''_i(t) = C\lambda \sum_{s=1}^{p_i+1} \left(\widetilde{t_{is}} - \widetilde{t_{i,s-1}}\right) P^C(t - \widetilde{t_{is}})\left(1 - P(t - \widetilde{t_{is}})\right)\left(C\left(1 - P(t - \widetilde{t_{is}})\right) - P(t - \widetilde{t_{is}})\right) \quad (4.30)$$

Ve (4.25) se vyskytují výrazy

$$\frac{\partial \widetilde{t_{js}}}{\partial \xi'_j} = \sigma'\left(\widetilde{t_{j,s-1}}, t_{js}, \delta; \xi'_j(t_{js})\right) \quad (4.31)$$

$$\frac{\partial \xi'_j}{\partial \tau'_i} = -w_{ji}\varepsilon''\left(t_{js} - d_{ji} - \tau_i(t_{js} - d_{ji})\right)\left(1 - \tau'_i(t_{js} - d_{ji})\right) \quad (4.32)$$

$$\frac{\partial \xi'_j}{\partial \tau'_i} = -w_{ji}\varepsilon'\left(t_{js} - d_{ji} - \tau_i(t_{js} - d_{ji})\right) \quad (4.33)$$

$$\frac{\partial t_{js}}{\partial \tau_i} = \frac{\frac{\partial t_{js}}{\partial w_{il}}}{\frac{\partial \tau_i}{\partial w_{il}}} = \frac{-\frac{\partial \xi_j}{\partial \tau_i}}{\xi'_j(t_{js})} = \frac{w_{ji}\varepsilon'\left(t_{js} - d_{ji} - \tau_i(t_{js} - d_{ji})\right)}{\xi'_j(t_{js})} \quad (4.34)$$

Vzorec (4.25) nyní použijeme v rovnicích (4.22) a (4.23) a rozepíšeme rekurentní vyjádření závislosti $\frac{\partial \widetilde{t_{js}}}{\partial w_{il}}$ na $\frac{\partial \widetilde{t_{j,s-1}}}{\partial w_{il}}$:

$$\frac{\partial}{\partial w_{il}}\tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \right) \times \left(\left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial \tau_i} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau_i} \right) \cdot \frac{\partial \tau_i}{\partial w_{il}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau'_i} \cdot \frac{\partial \tau'_i}{\partial w_{il}} \right) \tag{4.35}$$

$$\frac{\partial}{\partial w_{il}}\tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau'_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \right) \times \left(\left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial \tau_i} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau_i} \right) \cdot \frac{\partial \tau_i}{\partial w_{il}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau'_i} \cdot \frac{\partial \tau'_i}{\partial w_{il}} \right) \tag{4.36}$$

$0 \leq n_{js} \leq s-1$ je index definován jako nejmenší index, pro který platí

$$\frac{\partial \widetilde{t_{j,s-n_{js}}}}{\partial \widetilde{t_{j,s-n_{js}-1}}} = 0 \tag{4.37}$$

Sčítance ze vzorců (4.35) a (4.36) použijeme k výpočtu seznamu P_i (seznamy se vytvářejí po vrstvách neuronů od výstupní vrstvy směrem k nižším, takže při výpočtu P_i máme už k dispozici všechny seznamy P_j pro $j \in i_{\rightarrow}$):

$$P_i = \left(f_{jcsr} \cdot \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial \tau_i} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau_i} \right), f_{jcsr} \cdot \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j}, t_{jr} - d_{ji} \right) \tag{4.38}$$

$$f_{jcsr} = \left(\pi_{jc} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(u_{jc}) + \pi'_{jc} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(u_{jc}) \right) \prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \tag{4.39}$$

Seznam P_i se tvoří vzorcem (4.38) cyklem přes všechny $j \in i_{\rightarrow}$, $c = 1, \dots m_j$, $s = 1, \dots p_j$ a $r = s - n_{js}, \dots s$. S vytvořeným seznamem můžeme pomocí vzorečků (4.18) a (4.19) vypočítat parciální derivace chybové funkce vzhledem k vahám, resp. zpožděním. Parciální derivace funkce τ a τ' vyskytující se v (4.18) a (4.19) jsou

$$\frac{\partial}{\partial w_{ji}}\tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial w_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial w_{ji}} \right) \tag{4.40}$$

$$\frac{\partial}{\partial w_{ji}}\tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau'_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial w_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial w_{ji}} \right) \tag{4.41}$$

$$\frac{\partial}{\partial d_{ji}}\tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial d_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial d_{ji}} \right) \tag{4.42}$$

$$\frac{\partial}{\partial d_{ji}}\tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}}\tau'_j(t) \sum_{r=s-n_{js}}^{s} \left(\prod_{q=r+1}^{s} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q-1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial d_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial d_{ji}} \right) \tag{4.43}$$

Závislost t_{jr} na w_{ji} a d_{ji}je vyjádřena pouze implicitně vzorcem $\xi_j(t_{jr}) = 0$. Z věty o derivaci implicitní funkce dostáváme

$$\frac{\partial t_{jr}}{\partial w_{ji}} = -\frac{\frac{\partial \xi_j}{\partial w_{ji}}}{\frac{\partial \xi_j}{\partial t_{jr}}} = \begin{cases} -\dfrac{1}{\xi_j(t_{jr})} & pro\ i = 0 \\ -\dfrac{\varepsilon\left(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji})\right)}{\xi'_j(t_{jr})} & pro\ i \in j_{\leftarrow} \end{cases} \tag{4.44}$$

$$\frac{\partial t_{jr}}{\partial d_{ji}} = -\frac{\frac{\partial \xi_j}{\partial d_{ji}}}{\frac{\partial \xi_j}{\partial t_{jr}}} = \frac{w_{ji}\varepsilon'\left(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji})\right)\left(1 - \tau'_i(t_{jr} - d_{ji})\right)}{\xi'_j(t_{jr})} \tag{4.45}$$

Poslední dvě parciální derivace vyskytující se ve vzorečcích (4.40), (4.41), (4.42) a (4.43) jsou

$$\frac{\partial \xi'_j}{\partial w_{ji}} = \begin{cases} 0 & pro\ i = 0 \\ \varepsilon'\left(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji})\right)\left(1 - \tau'_i(t_{jr} - d_{ji})\right) & pro\ i \in j_{\leftarrow} \end{cases} \tag{4.46}$$

$$\begin{aligned} \frac{\partial \xi'_j}{\partial d_{ji}} = w_{ji}\Big(&\varepsilon'\left(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji})\right)\left(1 - \tau'_i(t_{jr} - d_{ji})\right) \\ &- \varepsilon''\left(t_{jr} - d_{ji} - \tau_i(t_{jr} - d_{ji})\right)\left(1 - \tau'_i(t_{jr} - d_{ji})\right)^2\Big) \end{aligned} \tag{4.47}$$

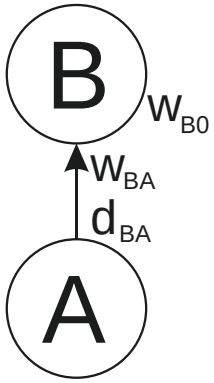
Tím je učící pravidlo kompletně definováno.

5 Experimentální výsledky

Testování probíhalo ve dvou fázích – v první byl algoritmus implementován v MATLABu pro ověření správnosti odvozeného algoritmu a prozkoumání základních vlastností jeho chování. Následně byl celý algoritmus implementován v C++ a testován na jednoduchých úlohách. V průběhu testování bylo provedeno několik úprav odstraňujících nedostatky architektury, které testy odhalily.

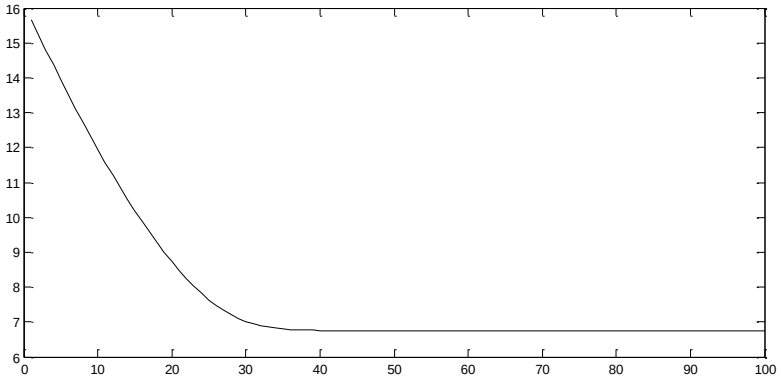
5.1 Konstantní zpoždění

Cílem tohoto jednoduchého testu je naučit jeden neuron, aby jeden pulz, který dostane na svém jediném vstupu, vracel s konstantním zpožděním. Použitý simulační interval je $[0; 10]$. Parametry byly inicializovány následovně: bias neuronu B je w_{B0}=-5, váha spoje je $w_{BA} = 10$, zpoždění je $d_{BA} = 3$.



Obrázek 9: Architektura sítě použitá v experimentu

Síti je předkládán jediný učící vzor – vstupem neuronu A je jediný pulz v čase $t_{A1} = 3$, jediný požadovaný výstup neuronu B je $\varrho_{B1} = 5$. Snažíme se tedy naučit neuron B vrátit pulz se zpožděním 2, zatímco původní zpoždění je nastaveno na 3. Protože váha a bias výstupního neuronu jsou již inicializovány tak, aby klidový potenciál byl záporný a vstupní pulz vytvořil odezvu, při které stoupne hodnota potenciálu nad nulu, zajímá nás hlavně adaptace váhy. Z tohoto důvodu byly učící konstanty nastaveny asymetricky – pro adaptaci vah a biasu se používala hodnota $\alpha = 0{,}005$ a pro adaptaci zpoždění $\alpha = 0{,}01$.

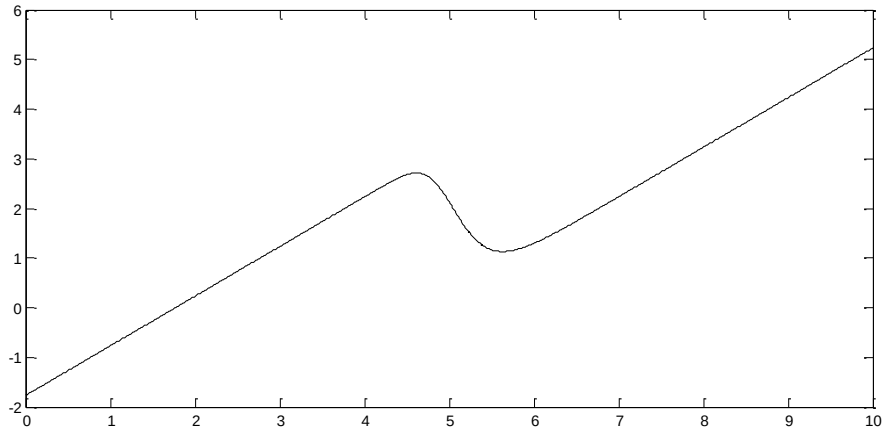


Obrázek 10: Vývoj chyby během průběhu učení

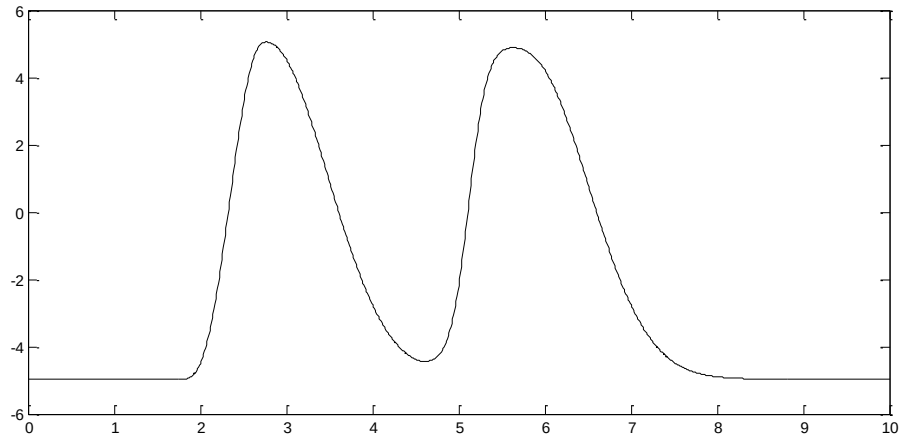
Bylo provedeno 100 iterací učícího algoritmu. Chyba během učení konverguje k hodnotě 6,7513. Proti veškerým očekáváním neuron B dává na výstupu dva pulzy:

2,3249	5,1056

Druhý pulz, který skutečně vzniká jako odezva na vstupní pulz, se síť naučila poměrně dobře. Kromě toho se ale na výstupu objevil pulz, který je v podstatě reakcí na nultý pulz, který jsme si formálně nadefinovali pro každý neuron jako $t_{j0} = 0$. Mechanismus jeho vzniku si nyní objasníme:



Obrázek 11: Graf argumentu funkce ε



Obrázek 12: Graf funkce ξ_B

Obrázek 11 ukazuje graf výrazu $t - d_{ji} - \tau_{ji}(t - d_{ji})$, což je hodnota, kterou ve vzorci (4.8) dostává funkce ε jako argument. Tento graf je vlastně spojitou aproximací času uplynulého od posledního pulzu. Připomeňme, že odezva funkce ε je největší, pokud její argument je malé kladné číslo a její maximum v bodě 1 (viz obrázek 7). Po-

kud zpožděný pulz dorazil synapsí v čase 5, zcela podle očekávání se krátce poté přiblíží hodnota argumentu ε hodnotě 0 a tím roste i nevážený přírůstek k potenciálu.

Z grafu ale vidíme, že argument je pro časy $t < d_{BA}$ záporný. V těchto místech je hodnota τ_A (která představuje čas posledního pulzu) nulová. Pokud tuto hodnotu „zpozdíme" o $d_{BA} = 2$, a následně jí odečteme od momentálního simulačního času, zjistíme, že naše aproximace času uplynulého od posledního pulzu dává na intervalu $[0; d_{BA}]$ záporné hodnoty! V momentě, kdy se simulační čas bude rovnat zpoždění, bude hodnota argumentu nulová a bude dále růst a funkce ε nám tak vytvoří v potenciálu pulz, který je vlastně obrazem nultého pulzu předchozího neuronu.

Nultý pulz kazí výpočet chybové funkce, učící algoritmus očekává $\tau_j(\widetilde{t_{j1}}) \cong 0$, ale místo toho dostává hodnotu přibližně se rovnající d_{ji}. Kvůli tomu není síť schopna naučit se ani nejjednodušší úlohy. Proto provedeme v modelu následující změnu:

5.2 Odstranění nultého pulzu

Jak je z obrázku 11 vidět, obraz nultého pulzu vznikl jiným mechanismem než obraz regulérního pulzu – u regulárního pulzu se argument funkce ε přiblíží k nule shora, nultý pulz nulovou osu protne. Ideou následující úpravy je vytvářet tímto mechanismem až první regulární pulz.

Pro přepočet t_{js} na $\widetilde{t_{js}}$ se místo vzorce (4.10) použije následující vzorec:

$$\widetilde{t_{js}} = \begin{cases} t_{js} & pro\ j \in X \\ \sigma\left(t_{js}, \widetilde{t_{s+1}}, \delta; \delta - \xi'(t_{js})\right) & pro\ j \notin X \end{cases} \tag{5.1}$$

Seznam $\widetilde{t_{js}}$ se nyní musí generovat pozpátku, tj. v posloupnosti od p_j do 1. Vzorec (4.11) funkce τ_j je upraven následujícím způsobem:

$$\tau_j(t) = \widetilde{t_{j1}} + \sum_{s=2}^{p_j+1} \left(\widetilde{t_{js}} - \widetilde{t_{j,s-1}}\right) P^C\left(t - \widetilde{t_{js}}\right) \tag{5.2}$$

Tato úprava si vyžaduje přepočítání dalších vzorečků. Vzorec pro τ' z (4.14) bude nahrazen vzorcem

$$\tau_i'(t) = \frac{\partial}{\partial t}\tau_i(t) = C\lambda \sum_{s=2}^{p_i+1} \left(\widetilde{t_{is}} - \widetilde{t_{i,s-1}}\right) P^C(t - \widetilde{t_{is}})\left(1 - P(t - \widetilde{t_{is}})\right) \tag{5.3}$$

To se promítne ve vzorečcích učícího pravidla. Vzorec (4.25) je třeba upravit pro zohlednění úpravy z (5.1):

$$\frac{\partial \widetilde{t_{js}}}{\partial w_{il}} = \frac{\partial \widetilde{t_{js}}}{\partial \widetilde{t_{j,s+1}}} \cdot \frac{\partial \widetilde{t_{j,s+1}}}{\partial w_{il}} + \left(\frac{\partial \widetilde{t_{js}}}{\partial t_{js}} \cdot \frac{\partial t_{js}}{\partial \tau_i} + \frac{\partial \widetilde{t_{js}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau_i} \right) \cdot \frac{\partial \tau_i}{\partial w_{il}} + \frac{\partial \widetilde{t_{js}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial \tau'_i} \cdot \frac{\partial \tau'_i}{\partial w_{il}} \tag{5.4}$$

n_{js}, původně definovaný v (4.37), nyní předefinujeme jako nejmenší index $s + 1 \leq n_{js} \leq p_j$, pro který platí

$$\frac{\partial \widetilde{t_{j,s+n_{js}}}}{\partial \widetilde{t_{j,s+n_{js}+1}}} = 0 \tag{5.4}$$

Vzorec pro výpočet faktoru f_{jcsr} z (4.39) nyní bude

$$f_{jcsr} = \left(\pi_{jc} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(u_{jc}) + \pi'_{jc} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(u_{jc}) \right) \prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \tag{5.5}$$

Vzorec (4.38) pro výpočet trojic tvořících seznam P_i bude stejný, ale cyklus vytvářející seznam nyní bude probíhat přes všechna $j \in i_{\rightarrow}, c = 1, \dots m_j, s = 1, \dots p_j$ a $r = s, \dots s + n_{js}$. Zbývá ještě úprava vzorečků (4.40), (4.41), (4.42) a (4.43):

$$\frac{\partial}{\partial w_{ji}} \tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial w_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial w_{ji}} \right) \tag{5.6}$$

$$\frac{\partial}{\partial w_{ji}} \tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial w_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial w_{ji}} \right) \tag{5.7}$$

$$\frac{\partial}{\partial d_{ji}} \tau_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial d_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial d_{ji}} \right) \tag{5.8}$$

$$\frac{\partial}{\partial d_{ji}} \tau'_j(t) = \sum_{s=1}^{p_j} \frac{\partial}{\partial \widetilde{t_{js}}} \tau'_j(t) \sum_{r=s}^{s+n_{js}} \left(\prod_{q=s}^{r-1} \frac{\partial \widetilde{t_{jq}}}{\partial \widetilde{t_{j,q+1}}} \right) \left(\frac{\partial \widetilde{t_{jr}}}{\partial t_{jr}} \cdot \frac{\partial t_{jr}}{\partial d_{ji}} + \frac{\partial \widetilde{t_{jr}}}{\partial \xi'_j} \cdot \frac{\partial \xi'_j}{\partial d_{ji}} \right) \tag{5.9}$$

Vzorce (4.35) a (4.36) je také třeba upravit, jejich plné znění zde ale neuvádíme, protože jsou to pouze mezikroky nevyskytující se v implementaci a jejich úprava je analogická úpravě ve vzorcích (5.6) a (5.7).

Nakonec je potřeba udělat drobnou technickou úpravu ve výpočtu chybové funkce (4.15) a v rutině pro vytváření seznamu P_i pro výstupní neurony $i \in Y$ (4.20): Pro tyto výpočty přiřadíme formálně definovanému požadovanému nultému pulzu každého výstupního neuronu $j \in Y$ hodnotu $\varrho_{j0} := \varrho_{j1}$.

5.3 Alternativní chybová funkce

Doposud používaná chybová funkce má několik nevýhod: Zaprvé, pokud při učení chyba konverguje k nule, zjistíme, že skutečné pulzy $\widetilde{t_{j1}}, \dots \widetilde{t_{j,p_j}}$ výstupních neuronů nenastávají přesně v požadovaných časech $\varrho_{j1}, \dots \varrho_{j,q_j}$, ale jsou mírně zpožděné. To je způsobeno tím, že správnost umístění pulzu počítáme pomocí funkce τ - v čase, kdy chceme, aby nastával požadovaný pulz ϱ_{js} se koukneme na hodnotu $\tau_j(\varrho_{js})$, která má udávat čas posledního pulzu předcházejícího času ϱ_{js}. Ten by ideálně měl být $\varrho_{j,s-1}$. Jenomže, protože τ čas posledního pulzu pouze spojitě aproximuje, i když budou všechny skutečné pulzy $\widetilde{t_{jk}}$ nastávat přesně v požadovaných časech ϱ_{jk}, hodnota $\tau_j(\varrho_{js})$ bude někde mezi ϱ_{js} a $\varrho_{j,s-1}$. Částečným řešením je zvýšení parametru C používaného ve funkci P, to ale snižuje gradient používaný v adaptačním pravidle a zpomaluje konvergenci.

Druhým problémem je, že úpravy parametrů, které by postupně vedly ke konvergenci k (z našeho intuitivního pohledu) lepším výsledkům chybová funkce, tak jak jí máme definovanou, vyhodnocuje jako úpravy zvyšující chybu. Praktický příklad: Máme výstupní neuron j, jehož požadované výstupy jsou $\varrho_{j1} = 2, \varrho_{j2} = 6$ a skutečné výstupy jsou $\widetilde{t_{j1}} = 2, \widetilde{t_{j2}} = 4, \widetilde{t_{j3}} = 6$. Máme tedy dva pulzy tam, kde je chceme mít a potřebujeme se zbavit pouze prostředního nežádoucího. Dejme tomu, že prostřední pulz by se dal eliminovat snížením nějaké váhy w_{ji} a přitom bychom neovlivnili časy dvou správných pulzů. Pokud začneme snižovat váhu w_{ji}, začne klesat derivace $\xi_j(t_{j2})$, což eventuelně povede k tomu, že se $\widetilde{t_{j2}}$ začne přesouvat doprava (zpožďovat) a nakonec se sloučí s následujícím pulzem. Přesně tak jak bylo zamýšleno. Ale pokud se koukneme na graf zobrazující průběh chybové funkce během této operace, zjistíme, že chyba roste. Roste totiž vzdálenost mezi $\tau_j(\varrho_{j2}) \cong t_{j2}$ a ϱ_{j1}. Můžeme obrátit logiku přepočtu t_{js} na $\widetilde{t_{js}}$, použít původní vzoreček (4.10), ve kterém se pulzy slučují místo následujících s předchozími, ale pak zase místo zániku nastane opačný problém se vznikem pulzů.

Proto navrhujeme jinou chybovou funkci, ve které se pro výpočet nepoužívá τ a která oceňuje mechanismus vzniku a zániku pulzů popsaný v předchozím odstavci.

Máme-li pro každý výstupní neuron $j \in Y$ vypočtenou posloupnost *skutečných výstupních pulzů* $\widetilde{t_{j0}} = 0 < \widetilde{t_{j1}} < \widetilde{t_{j2}} < \cdots < \widetilde{t_{j,p_j}} < \widetilde{t_{j,p_j+1}} = T$ a danou posloupnost *požadovaných výstupních pulzů* $\varrho_{j0} = 0 < \varrho_{j1} < \varrho_{j2} < \cdots < \varrho_{j,q_j} < \varrho_{j,q_j+1} = T$, potom odchylku sítě na jednom trénovacím vzoru můžeme vyjádřit chybou

$$E_2(w,d) = \sum_{j \in Y} \left(\sum_{a=1}^{p_j} \min_{b=0,\dots q_j+1} \left(\widetilde{t_{ja}} - \varrho_{jb}\right)^2 + \sum_{b=1}^{q_j+1} \min_{a=0,\dots p_j+1} \left(\widetilde{t_{ja}} - \varrho_{jb}\right)^2 \right) \tag{5.10}$$

Tato funkce má ale jednu podstatnou nevýhodu – není ve všech bodech diferencovatelná. Body, kde není gradient funkce E_2 definován, jsou konkrétně ty, ve kterých buďto

- Pro nějaké $a \in \{1, \dots p_j\}$ existují dva indexy $b_1, b_2 \in \{0, \dots q_j + 1\}$ takové, že $\left(\widetilde{t_{ja}} - \varrho_{jb_1}\right)^2 = \left(\widetilde{t_{ja}} - \varrho_{jb_2}\right)^2 = \min_{b=0,\dots q_j+1}\left(\widetilde{t_{ja}} - \varrho_{jb}\right)^2$
- Nebo pro nějaké $b \in \{1, \dots q_j\}$ existují dva indexy $a_1, a_2 \in \{0, \dots p_j + 1\}$ takové, že $\left(\widetilde{t_{ja_1}} - \varrho_{jb}\right)^2 = \left(\widetilde{t_{jb_2}} - \varrho_{jb}\right)^2 = \min_{a=0,\dots p_j+1}\left(\widetilde{t_{ja}} - \varrho_{jb}\right)^2$.

Proto budeme váhy a zpoždění adaptovat heuristickou metodou

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} - \alpha \Delta w_{ji}^{(t)} \tag{5.11}$$

$$d_{ji}^{(t)} = d_{ji}^{(t-1)} - \alpha \Delta w_{ji}^{(t)} \tag{5.12}$$

kde členy $\Delta w_{ji}^{(t)}$ a $\Delta w_{ji}^{(t)}$ se vypočítají pomocí

$$\begin{aligned} \Delta w_{ji}^{(t)} = \Big(& E_2\left(w^{(t-1)}\left[w_{ji} \backslash w_{ji}^{(t-1)} + \epsilon\right], d^{(t-1)}\right) \\ & - E_2\left(w^{(t-1)}\left[w_{ji} \backslash w_{ji}^{(t-1)} - \epsilon\right], d^{(t-1)}\right)\Big) / 2\epsilon \end{aligned} \tag{5.13}$$

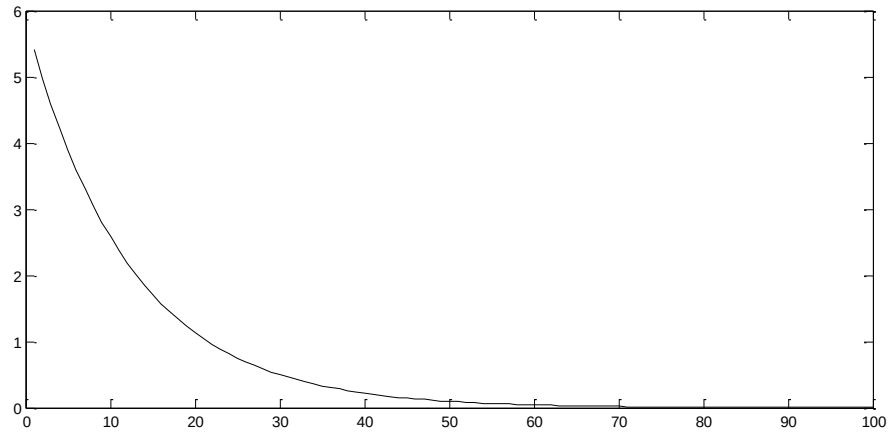
$$\begin{aligned} \Delta d_{ji}^{(t)} = \Big(& E_2\left(w^{(t-1)}, d^{(t-1)}\left[d_{ji} \backslash d_{ji}^{(t-1)} + \epsilon\right]\right) \\ & - E_2\left(w^{(t-1)}, d^{(t-1)}\left[d_{ji} \backslash d_{ji}^{(t-1)} - \epsilon\right]\right)\Big) / 2\epsilon \end{aligned} \tag{5.14}$$

Ve výrazech (5.13) a (5.14) je ϵ malé kladní číslo, zápis $w^{(t-1)}\left[w_{ji} \backslash w_{ji}^{(t-1)} + \epsilon\right]$ značí vektor vah, ve kterém je váha w_{ji} nahrazena hodnotou výrazu $w_{ji}^{(t-1)} + \epsilon$.

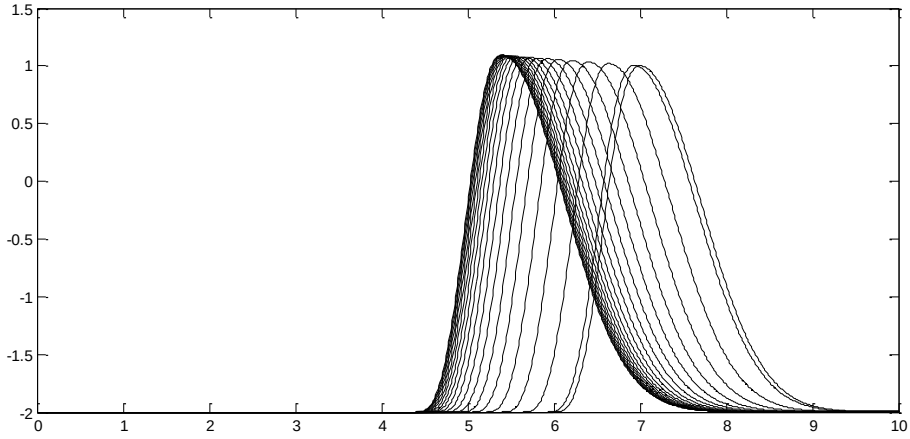
5.4 Konstantní zpoždění v upraveném modelu

Náš model vybavený upravenou dynamikou, novou chybovou funkcí a novým učícím pravidlem jsme znovu otestovali na úloze popsané v kapitole 5.1. Byla použita stejná architektura, iniciální parametry byly nastaveny následovně: $w_{B0} = -2, w_{BA} = 3,$ $d_{BA} = 3$. Adaptace váhy w_{B0} byla vypnuta, učící konstanta byla pro váhy $\alpha_w = 0{,}005,$ pro zpoždění $\alpha_D = 0{,}01$. Po 100 iteracích algoritmu dává sít na vstup $t_{A1} = 3$ odpo-

věď 5,028, chyba je 0,00157. Parametry sítě se upravily na hodnoty $w_{BA} = 3{,}09244$, $d_{BA} = 1{,}393025$.



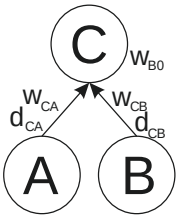
Obrázek 13: Vývoj chyby E_2 v průběhu učení úlohy 5.4



Obrázek 14: vývoj průběhu funkce x_B v průběhu učení úlohy 5.4
Pulz se přesunoval zprava doleva.

5.5 Úloha AND s jednoduchým kódováním vstupů

Pote, co jsme si předchozím experimentem ověřili základní funkčnost, můžeme zkusit úlohu, která je jen nepatrně složitější: Naučit síť logickou funkci AND. Pro tuto úlohu zvolíme síť se dvěma vstupními a jedním výstupním neuronem. Vstupy neuronů budeme kódovat jedním pulzem s konstantním časem 3, který buď nastává nebo nenastává podle toho, jestli je logickým vstupem neuronu *true* nebo *false*.

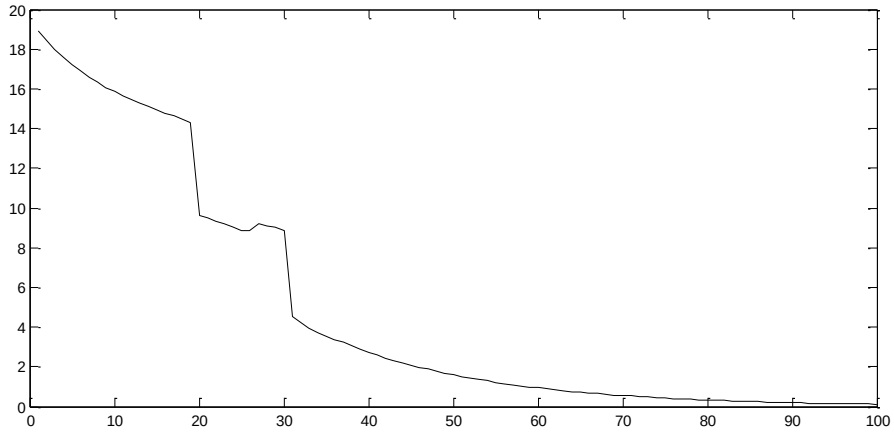


Obrázek 15: Architektura použitá v úloze 5.5

Simulační interval je opět $T = 10$. Iniciální parametry byly zvoleny následovně: $w_{C0} = -2, w_{CA} = 2{,}5, w_{CB} = 2{,}4, d_{CA} = 3{,}3, d_{CB} = 3{,}3$. Síť je tak vlastně na začátku nastavena na počítání funkce OR a cílem učení je kromě adaptace zpoždění také snížení vah tak, aby potenciál přesáhl nulu pouze v případě, že pulz přichází od obou vstupních neuronů.

#	Logický význam	Vstup neuronu A	Vstup neuronu B	Očekávaný výstup neuronu C
1	0 & 0 = 0	[0, 10]	[0, 10]	[0, 10]
2	0 & 1 = 0	[0, 10]	[0, 3, 10]	[0, 10]
3	1 & 0 = 0	[0, 3, 10]	[0, 10]	[0, 10]
4	1 & 1 = 1	[0, 3, 10]	[0, 3, 10]	[0, 6, 10]

Tato úloha je první, ve které byl předkládán více než jeden učící vzor. Adaptace parametrů probíhala po dávkách – napřed byly předloženy všechny vzory a pro každý spočteno Δw a Δd a poté byly parametry sítě aktualizovány průměrem ze všech vypočtených vzorů.

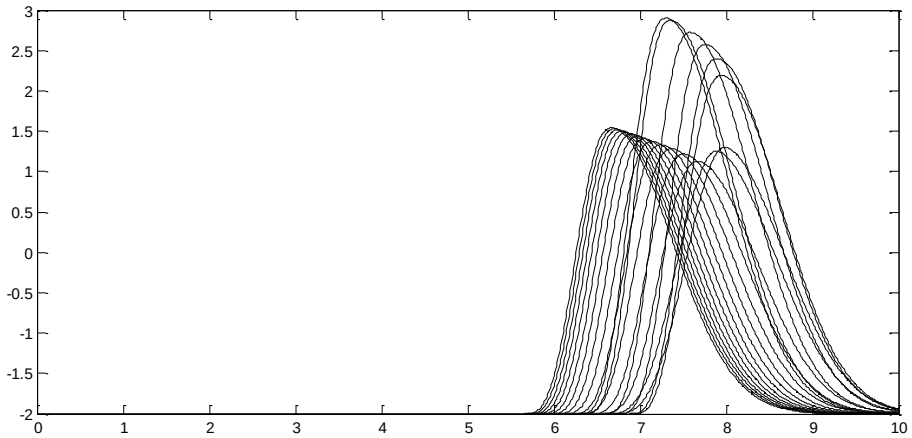


Obrázek 16: Vývoj kumulativní chyby všech vzorů v průběhu učení

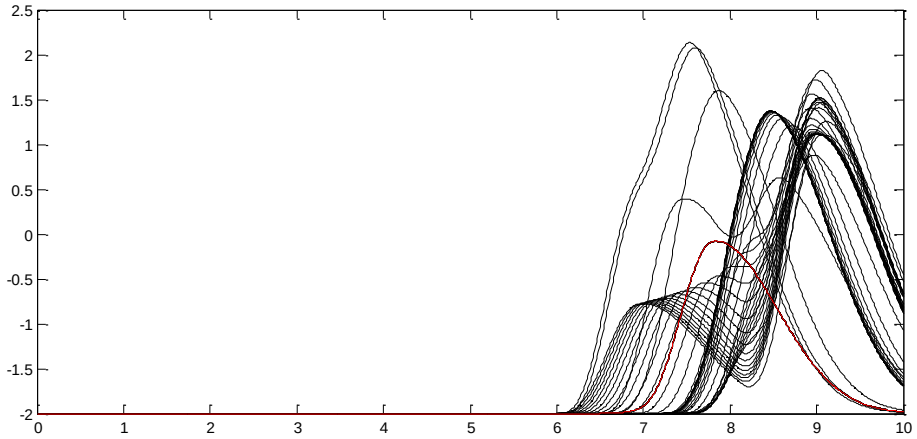
Obrázek 17 znázorňuje, jak se vyvíjel v čase průběh funkce ξ_C. Začátek je v nejvyšším pulzu. V reakci na to, že na čtvrtém učícím vzoru se na výstupu objevuje pulz, který tam být nemá, se síť snaží jednak stlačit váhu obou spojení a zadruhé posunout pulz tak, aby se slil s následujícím pulzem nastávajícím v čase 10. V momentě, kdy jsou váhy sníženy natolik, že funkce ξ_C překračuje nulovou hodnotu pouze pokud jsou oba vstupy aktivní, síť začíná zmenšovat zpoždění, aby se u čtvrtého vzoru kryl skutečný čas pulzu s časem požadovaným.

Pokud budeme zkoušet upravovat počáteční váhy a zpoždění, zjistíme, že výkon algoritmu v této úloze na inicializaci parametrů kriticky závisí. Pokud nastavíme zpož-

dění ze vstupních neuronů nesymetricky a dostatečně daleko od sebe, síť nedokáže oba příchozí pulzy spárovat a snaží se nechtěných pulzů na vzorech 2 a 3 zbavit snižováním vah. Pokud bychom nechali běžet algoritmus déle, síť se eventuálně dostane do stavu, kdy se váhy obou spojů sníží natolik, že síť nebude generovat na výstupu žádný pulz. V ten moment se učení zastaví, protože všechny Δw a Δd budou vycházet nulové. Příklad průběhu takového učení ukazuje obrázek 18.



Obrázek 17: Vývoj průběhu funkce ξ_C na čtvrtém učícím vzoru.



Obrázek 18: Pokud nastavíme váhy nesymetricky, síť konverguje k patologickému stavu. Průběh funkce ξ_C na 4. učícím vzoru. 1000 iterací, vykreslena je každá 20. Konečný stav je zvýrazněn červeně.

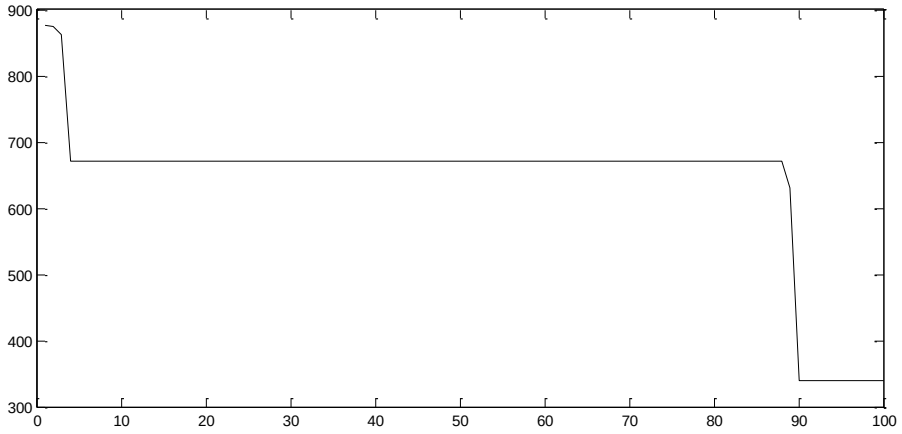
5.6 Úloha AND s frekvenčním kódováním

Výsledky předchozí úlohy, kde se síť nedokáže správně naučit požadované výstupy, pokud nemá vhodně inicializované váhy, vedou k myšlence, že učení by se dalo síti výrazně ulehčit vhodným kódováním vstupů a výstupů. Náš učící algoritmus má oproti většině ostatních algoritmů pro učení pulzních sítí výhodu v tom, že umí přiro-

zeně pracovat s více než jedním pulzem na neuron. V této úloze zkusíme rovnoměrně rozprostřít vstupní informaci do celého simulačního intervalu. Použijeme síť se stejnou architekturou jako v předchozí úloze (viz obrázek 15). Simulační interval byl prodloužen na $T = 20$. Kódování dat je zvoleno tak, že pokud je logickým vstupem *true*, vstupem neuronu jsou ekvidistantní pulzy po celé délce simulačního intervalu se vzdáleností 2. Výstup je kódován identicky.

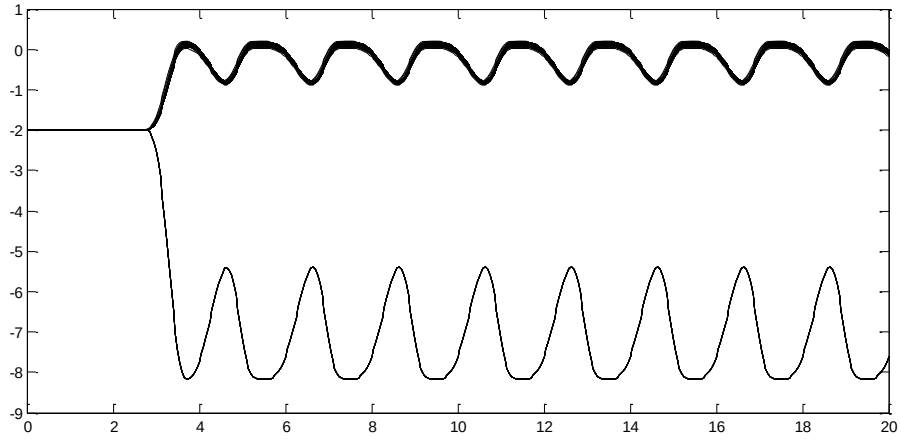
#	Logický význam	Vstup neuronu A	Vstup neuronu B	Požadovaný výstup
1	0 & 0 = 0	[0, 20]	[0, 20]	[0, 20]
2	0 & 1 = 0	[0, 20]	[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]	[0, 20]
3	1 & 0 = 0	[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]	[0, 20]	[0, 20]
4	1 & 1 = 1	[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]	[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]	[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

Učící konstanta je pro váhy i zpoždění nastavena identicky $\alpha = 0{,}005$. Bias neuronu C je nastaven na $w_{C0} = -2$ a adaptace biasu je vypnuta. Iniciální parametry jsou $w_{CA} = 2{,}1$, $w_{CB} = 2{,}2$, $d_{CA} = 0{,}1$, $d_{CB} = 0{,}6$. Váhy jsou tedy podobně jako v předchozí úloze nastaveny přibližně tak, aby na začátku síť počítala funkci OR. Bylo provedeno 100 iterací učení.

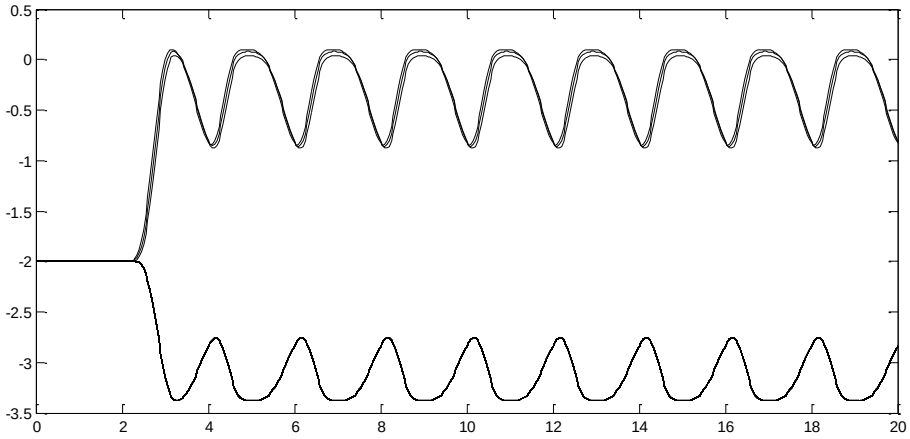


Obrázek 19: Vývoj kumulativní chyby všech vzorů v průběhu učení

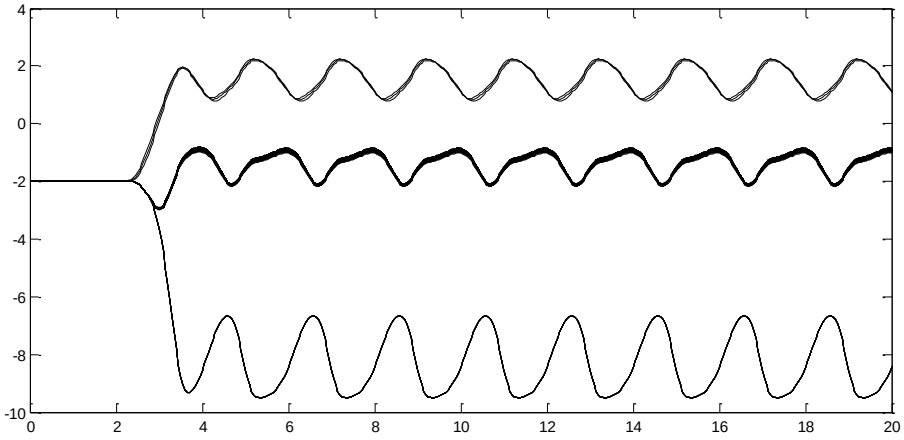
Po 88. iteraci se chyba ustálí na hodnotě 340. Po této iteraci jsou obě váhy nastaveny na zápornou hodnotu, síť nevydává žádný výstupní pulz a dále se již neučí.



Obrázek 20: Vývoj průběhu funkce ξ_C na učícím vzoru 2



Obrázek 21: Vývoj průběhu funkce ξ_C na učícím vzoru 3



Obrázek 22: Vývoj průběhu funkce ξ_C na učícím vzoru 4

Obrázky 20, 21 a 22 podrobněji ilustrují průběh učení. Algoritmus se snažil eliminovat pulzy, které se ze začátku tvořily na výstupu u vzorů 2 a 3 snižováním vah. Po 9. iteraci se skokově snížila váha w_{CB} na zápornou hodnotu, čímž se eliminovaly nechtěné pulzy na výstupu pro vzor 3. Eliminovat nechtěné pulzy na vstupním vzoru 2 se síti povedlo po 88. iteraci, v ten moment byly ale všechny váhy záporné. Na výstupu 4 síť nedávala po čas celého průběhu učení ani jednou správný výsledek, pouze prvních 8 iteraci dávala jeden pulz polem času 3 před tím, než se potenciál neuronu C přesaturoval.

5.7 Resilient propagation

Výsledky předchozího experimentu ukazují na jeden problém: v momentě, kdy jsou váhy jednoho neuronu j nastaveny tak, že potenciálová funkce ξ_j překračuje nulu ve chvílích, kdy se téměř blíží svým lokálním extrémům (tak jak je to vidět na obrázcích 20 a 21), prudce stoupá absolutní hodnota členu Δw_{ji} pro některá i (ta, která svými pulzy lokální extrémy generují), protože kvůli přepočtu t_{jk} na $\widetilde{t_{jk}}$ se časy pulzů prudce přesouvají.

Jednou z možností, jak tento problém řešit, je použití upraveného pravidla pro adaptaci vah. Pravidlo. které navrhujeme, je inspirováno heuristikou pro adaptaci vah v perceptronových sítích označovanou jako *Resilient propagation*, zkráceně *RPROP* (26). Tato metoda stále využívá členy Δw_{ji} a Δd_{ji} tak, jak jsou definované v (5.13) a (5.14), ale mění se způsob, jakým jsou používány pro adaptaci parametrů.

Každému neuronu j nyní přidáme sadu proměnných $\overline{w_{ji}}, i \in j_{\leftarrow} \cup \{0\}$ a $\overline{d_{ji}}, i \in j_{\leftarrow}$. Těmto proměnným budeme říkat *aktualizační hodnoty*. Kromě toho budeme mít ještě tři globální konstanty: *počáteční aktualizační hodnota* η_0 bude malé kladné číslo, *přírůstkový faktor* $\eta_{inc} > 1$ a *úbytkový faktor* $\eta_{dec} \in (0; 1)$. t-tý krok adaptace váhy bude probíhat následovně:

- vypočte se hodnota $\Delta w_{ji}^{(t)}$
- pokud probíhá první iterace nebo pokud $\delta\, w_{ji}^{(t-1)} = 0$, nastaví se $w_{ji}^{(t)} = sgn\left(\Delta_{ji}^{(t)}\right) \cdot \eta_0$
- v ostatních iteracích
    - pokud $sgn\left(\Delta w_{ji}^{(t)}\right) = sgn\left(\Delta w_{ji}^{(t-1)}\right)$, potom se zvětší aktualizační hodnota příslušné váhy: $\overline{w_{ji}}^{(t)} = \overline{w_{ji}}^{(t-1)} \cdot \eta_{inc}$
    - v opačném případě se aktualizační hodnota nastaví na $\overline{w_{ji}}^{(t)} = -sgn\left(-\Delta w_{ji}^{(t)}\right) \cdot \eta_{dec}$

- Samotná váha se pak aktualizuje přičtením aktualizační hodnoty: $w_{ji}^{(t)} = w_{ji}^{(t-1)} + \overline{w_{ji}}^{(t)}$

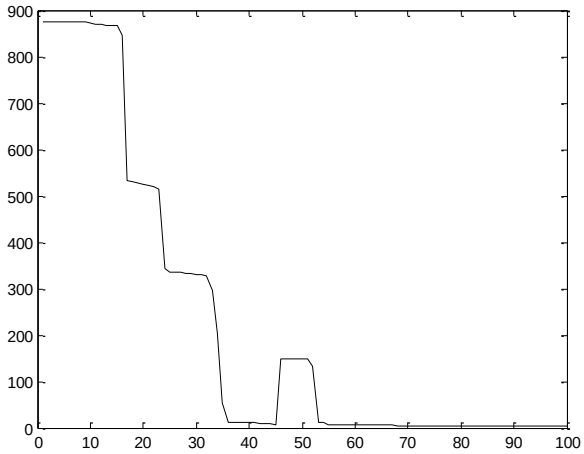
Zpoždění se aktualizují analogicky. Tuto upravenou verzi adaptačního pravidla budeme označovat RP.

Touto metodou se parametry aktualizují daleko předvídatelnějším a plynulejším způsobem, bez strmých výkyvů doprovázejících vznik a zánik pulzů. Další výhodou (která byla hlavním designovým záměrem původního algoritmu *RPROP*) je rychlejší konvergence v oblastech, kde se chybová funkce mění monotónně, ale jen velmi pozvolně.

5.8 Úloha AND s frekvenčním kódováním a adaptačním pravidlem RP

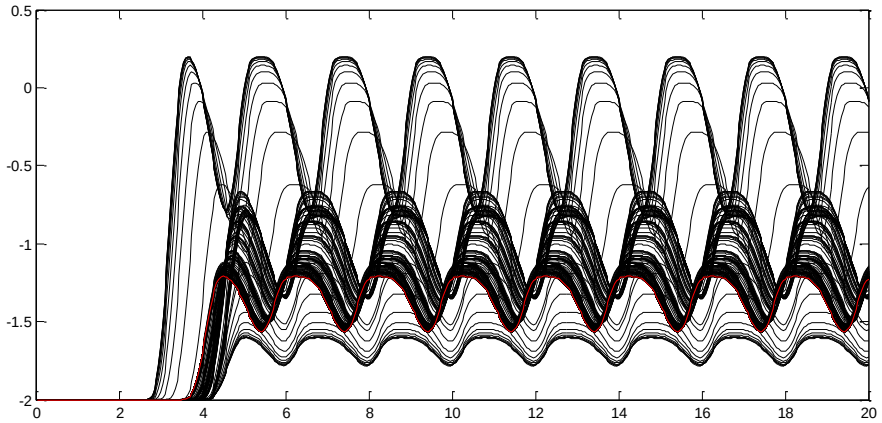
Použijeme síť se stejnou architekturou a stejnými učícími vzory jako v úloze z kapitoly 5.6. Pro váhy i zpoždění byly zvoleny parametry $\eta_0 = 0{,}0001, \eta_{inc} = 1{,}3$.

Síť se tentokrát dokázala téměř kompletně naučit všechny vzory, výjimkou je první pulz ve čtvrtém vzoru, jehož absence na výstupu indikuje, že synapse mají příliš velké zpoždění, nicméně kódování vstupů a výstupů bylo schválně zvoleno tak, aby mělo dostatečně velkou redundanci a přeskočení prvního pulzu na výstupu tak nepředstavuje žádnou zásadní ztrátu informace.

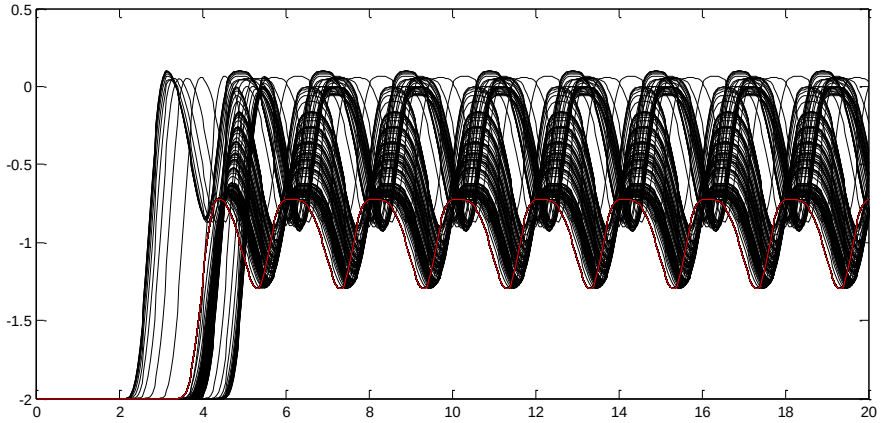


Obrázek 23: Vývoj kumulativní chyby všech vzorů v průběhu učení

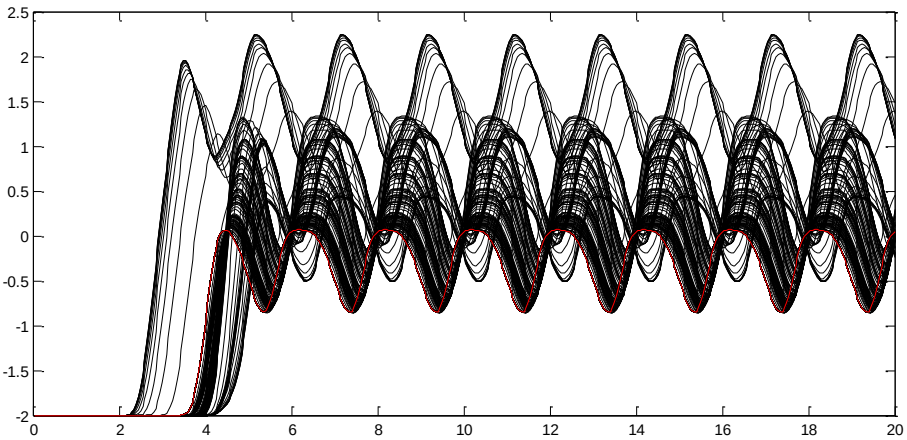
#	Výstup neuronu C
1	[0; 20]
2	[0; 20]
3	[0; 20]
4	[0; 4.46702; 5.96287; 7.9637; 9.9637; 11.9637; 13.9637; 15.9637; 17.9637; 20]



Obrázek 24: Vývoj průběhu funkce ξ_C na učícím vzoru 2. Konečný stav je znázorněn červeně



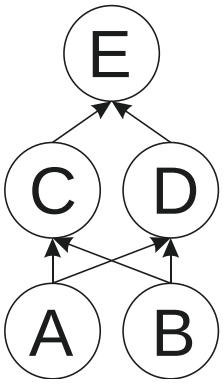
Obrázek 25: Vývoj průběhu funkce ξ_C na učícím vzoru 3. Konečný stav je znázorněn červeně



Obrázek 26: Vývoj průběhu funkce ξ_C na učícím vzoru 4. Konečný stav je znázorněn červeně

5.9 Úloha XOR ve vícevrstvé síti

XOR je klasická testovací úloha pro neuronové sítě. Pro tento test jsme zvolili vícevrstvou architekturu – Síť má kromě dvou vstupních neuronů a jednoho výstupního navíc jednu skrytou vrstvu se dvěma neurony. Simulační interval byl nastaven na $T = 30$. Parametry adaptačního algoritmu RP byly pro váhy i zpoždění nastaveny na $\eta_0 = 0{,}001$, $\eta_{inc} = 1{,}5$ a $\eta_{dec} = 1/3$. Kódování dat je stejné jako v předchozí úloze – pokud je logický vstup *true*, na vstupu neuron jsou po celou dobu simulačního intervalu ekvidistantní pulzy ve vzdálenosti 2; pokud je logický vstup *false*, na vstupu neuronu nejsou žádné pulzy.

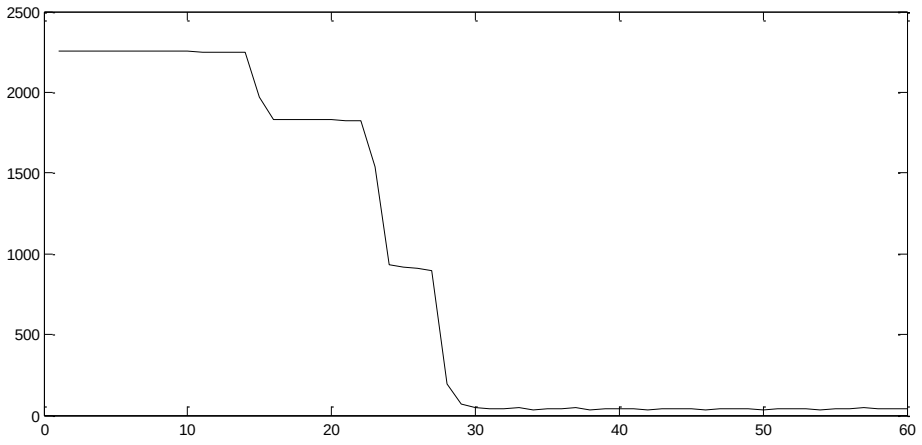


Obrázek 27: Použitá architektura sítě

V první iteraci tohoto pokusu jsme místo náhodné inicializace vah nastavili iniciální váhy následovně:

$$w_{CA} = 1{,}5 \quad w_{CB} = 1{,}4$$
$$w_{DA} = 2{,}1 \quad w_{DB} = 2{,}4$$
$$w_{EC} = 1{,}5 \quad w_{ED} = 1{,}4$$

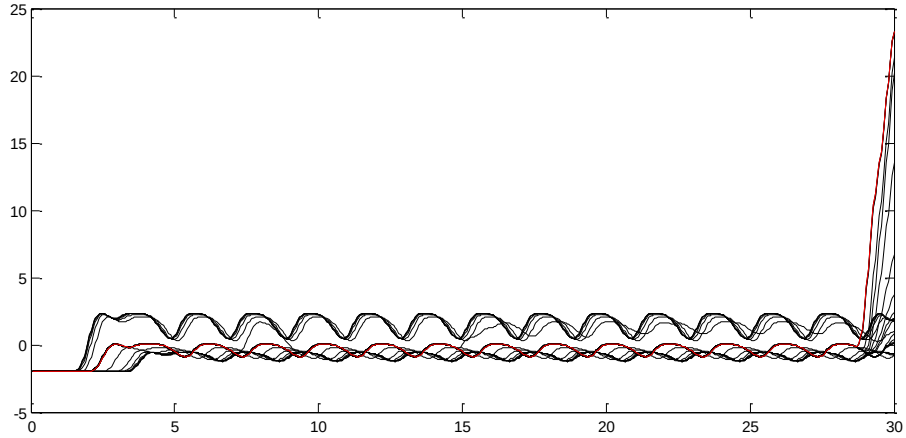
Bias všech neuronů byl nastaven na -2. Zpoždění byla náhodně zvolená malá čísla z intervalu $[0; 0{,}4]$. V této konfiguraci počítá neuron C přibližně logickou funkci OR a neuron D logickou funkci AND a jednou z možností, jak by se síť mohla naučit XOR je adaptovat váhy tak, aby se v neuronu E odečítal výstup D od A.



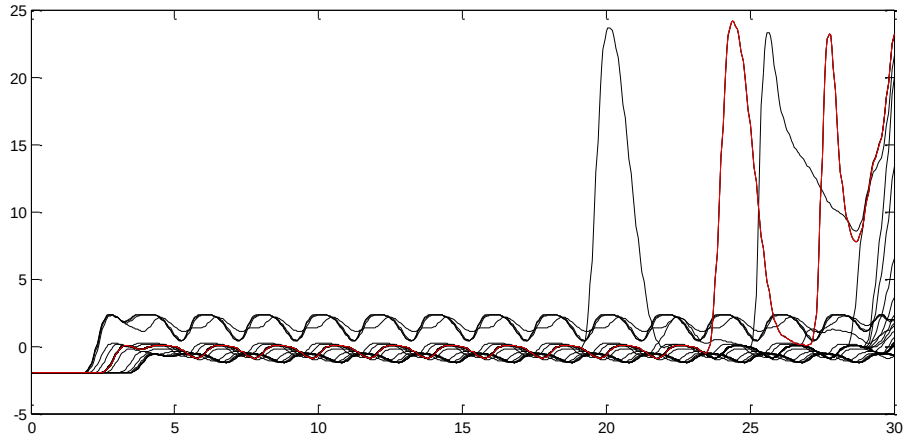
Obrázek 28: Vývoj kumulativní chyby všech vzorů v průběhu učení

Síť po 60 epochách učení skončila s chybou 40,3078. Experimentování s učícími parametry η_0 a η_{inc} ukázalo, že výkon algoritmu je na jejich volbě kriticky závislý; při volbě příliš nízkých hodnot se učení zastaví v suboptimálním lokálním minimu, při vol-

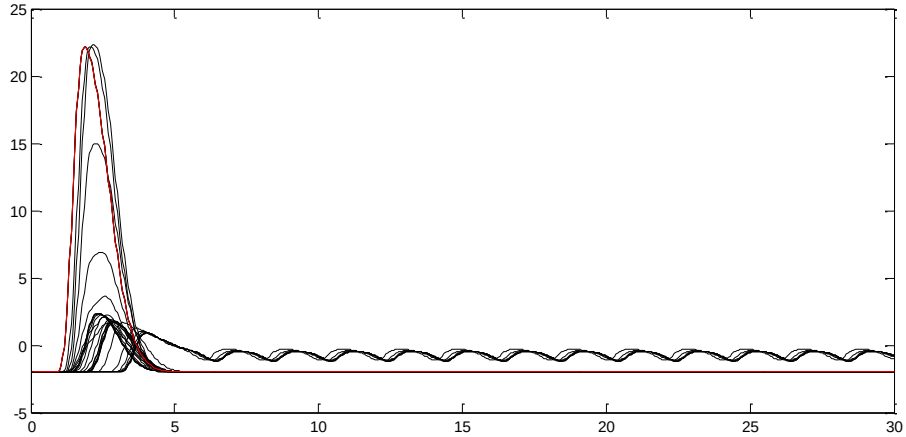
bě vysokých hrozí odumření neuronu nebo stav, kdy neuronová síť osciluje na rozhranní, kdy drobná úprava váhy jednoho neuronu způsobuje vznik nebo zánik pulzů.



Obrázek 29: Vývoj průběhu funkce ξ_E pro učící vzor 2



Obrázek 30: Vývoj průběhu funkce ξ_E pro učící vzor 3

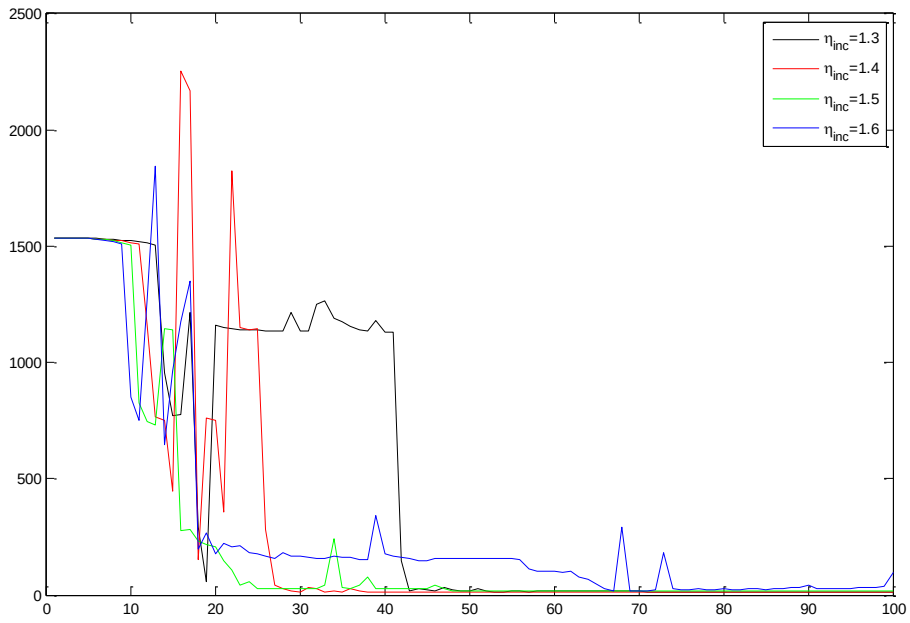


Obrázek 31: Vývoj průběhu funkce ξ_E pro učící vzor 4

Na počítači s procesorem Core 2 Quad Q6600 (4x2,4GHz) trvalo 60 iterací 16 sekund. Výstupy naučené sítě pro jednotlivé vzory vypadají následovně:

#	Výstup neuron E
1	[0; 29.1469; 30]
2	[0; 4.34046; 9.57925; 11.9101; 13.823; 15.7109; 17.5628; 19.3673; 21.109; 22.7679; 24.3175; 25.7226; 26.9369; 27.8992; 28.5288; 28.7189; 30]
3	[0; 4.62877; 9.66329; 12.0066; 13.8213; 15.5772; 17.2548; 18.829; 20.2666; 21.5239; 22.5431; 23.2477; 23.5368; 30]
4	[0; 1.10667; 30]

Další iterací tohoto pokusu je naučit síť počítat XOR, aniž bychom jí předem pomáhali vhodnou inicializací vah. Předpokládáme, že pokud máme úlohu, o které předem nic nevíme, je nejlepší inicializovat biasy zápornými čísly (tak, jak jsme to činili dosud) a váhy náhodně zvolenými čísly většími než absolutní hodnota biasu. Tak máme zaručeno, že pokud neuron dostává na vstupu aspoň jeden pulz, bude mít aspoň jeden pulz na výstupu, což je nutná podmínka pro to, aby se neuron dokázal učit. Následující grafu ukazuje průběhy učení pro síť s náhodně inicializovanými parametry pro různá nastavení učících parametrů:

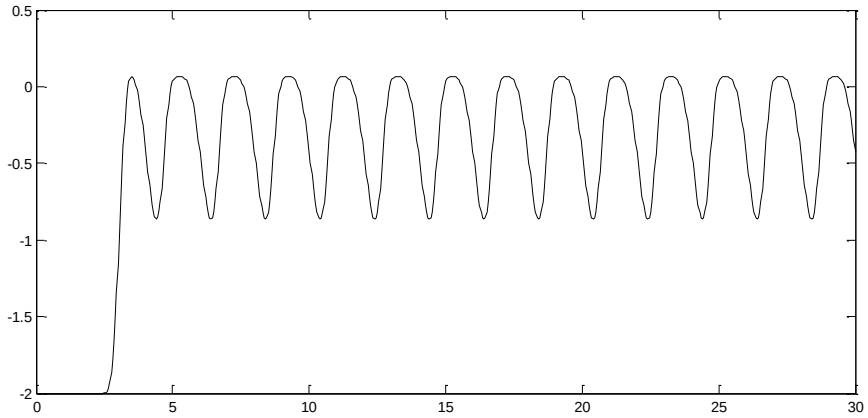


Obrázek 32: Vývoj kumulativní chyby všech vzorů v průběhu učení pro síť s náhodně nastavenými vahami

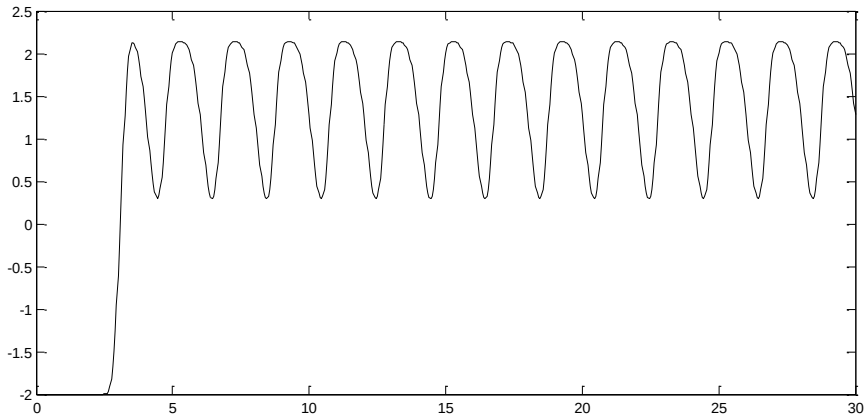
Síť s náhodně inicializovanými vahami paradoxně oproti očekávání konvergovala k nižší chybě než při prvním pokusu, při kterém jsme váhy přednastavili. S $\eta_{inc} = 1{,}5$ konvergovala síť nejrychleji, po 100 epochách učení byla chyba 15,5638. S nastavením $\eta_{inc} = 1{,}6$ už začíná být algoritmus nestabilní.

5.10 Úloha XOR s jedním neuronem

Síť používaná v algoritmu *SpikeProp* je schopna naučit se úlohu XOR pouze s jedním výpočetním neuronem. Připomeňme, že v architektuře sítí používaných pro *SpikeProp* jsou zavedeny synaptické terminály – dva neurony jsou propojeny více spoji s různými zpožděními. *SpikeProp* bez synaptických terminálů se není XOR schopen s jedním neuronem naučit (11), což je zapříčiněno omezenými možnostmi kódování – *SpikeProp* podporuje pouze jeden výstupní pulz na neuron. Naše architektura sice nemá synaptické terminály, ale máme možnost pracovat s více pulzy na neuron a zvolit frekvenční kódování.

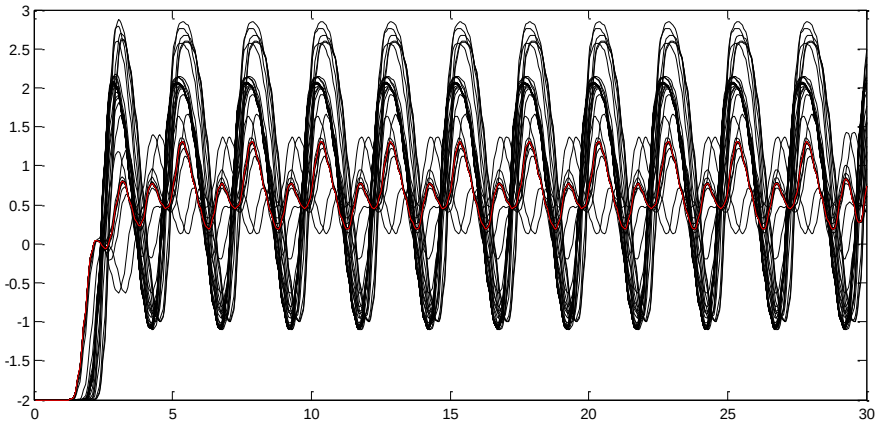


Obrázek 33: Funkce ξ jednoho neuronu řešícího $1 \oplus 0$

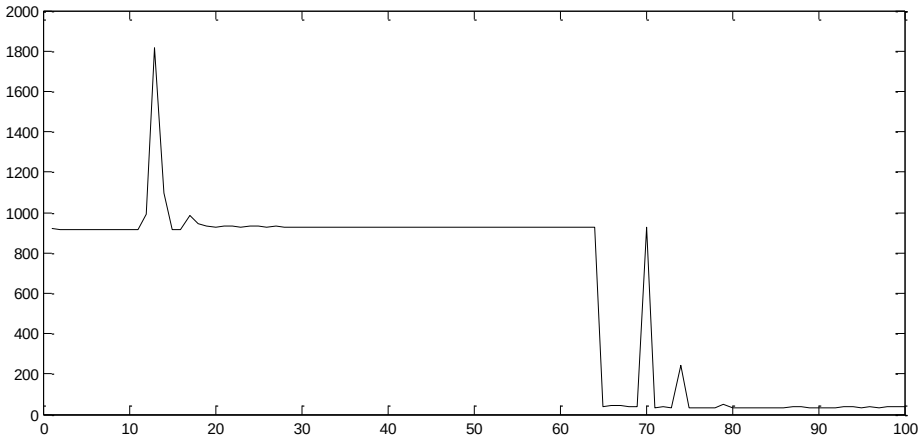


Obrázek 34: Funkce ξ jednoho neuronu řešícího $1 \oplus 1$

Úspěch učení poměrně výrazně závisí na zvolených parametrech kódování. Pokud pro kódování logického *true* zvolíme ekvidistantní pulzy se vzdáleností 2, síť je prakticky naučená již od začátku. Obrázek 33 ilustruje, jak neuron úlohu XOR řeší: pokud přichází signál pouze z jednoho spoje, neuron tento signál prakticky beze změny kopíruje na výstup. Pokud přichází výstup z obou spojů, Na výstupu se objeví jeden pulz, po zbytek simulačního intervalu ale bude potenciál neuronu přesaturovaný a pod nulu již nikdy neklesne. V síti s parametry $C = 3, \lambda = 4$ (připomeňme, že tyto parametry kontrolují strmost schodů ve funkce τ) a s pulzy šířky 2 je ve skutečnosti spíše problém najít počáteční váhy a zpoždění takové, aby neuron už od začátku funkci XOR nepočítal. Jakmile začneme vzdálenosti pulzů zvětšovat, začne být učení komplikovanější, protože pro potlačení pulzů pro vstup $1 \oplus 1$ bude potřeba měnit zpoždění vstupů, aby se vyplnily díry, ve kterých se funkce ξ propadá pod nulu, a to se děje na úkor přesnosti časování výstupních pulzů pro vstupy $1 \oplus 0$ a$1 \oplus 1$. Následující obrázky ilustrují průběh učení pro rozteč pulzů 2,5:



Obrázek 35: Vývoj průběhu funkce ξ na výstupním neuronu pro vzor $1 \oplus 1$. Konečný stav je znázorněn červeně.



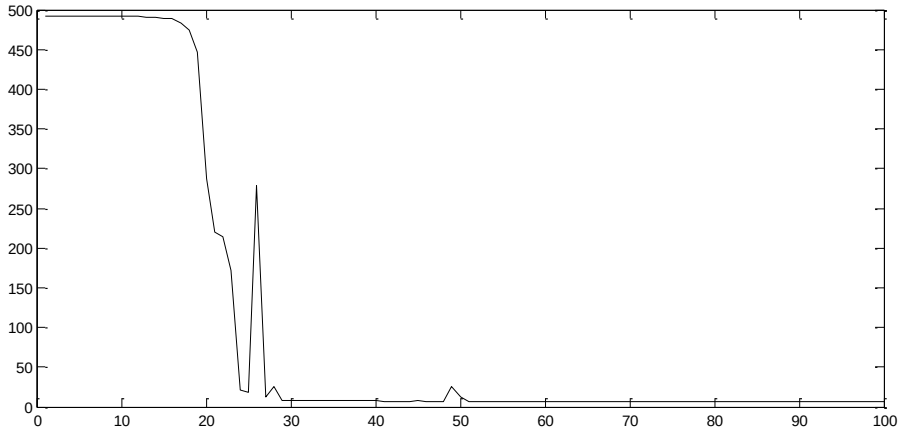
Obrázek 37: Vývoj kumulativní chyby všech vzorů v průběhu učení

Pokud zakódujeme data do pulzů s roztečí 3, síť už nekonverguje k řešení, které by správně řešilo vstup $1 \oplus 1$, přestože takové řešení existuje: stačí posunout zpoždění tak, aby periodický signál z prvního vstupu byl oproti signálu z druhého vstupu v opačné fázi. Díky tomu by lokální minima funkce ξ výstupníno neuronu pro vzor $1 \oplus 1$ neklesala pod nulu. Tak ale vzroste chyba na výstupu aspoň jednoho ze vzorů $1 \oplus 0$ a $0 \oplus 1$, kterou se učící algoritmus bude snažit zmenšit posunem vah zpět. Při posunování vah bude klesat chyba na vzorech $1 \oplus 0$ a $0 \oplus 1$, zatímco na vzoru $1 \oplus 1$ se chyba měnit nebude - do té chvíle než se posouváním fází vstupů zpět sníží lokální minima potenciálu výstupního neuronu pod hodnotu 0. V tuto chvíli už se síť nedokáže vrátit do stavu s posunutými fázemi a vzor $1 \oplus 1$ se už nikdy nenaučí.

5.11 Dolní frekvenční propust

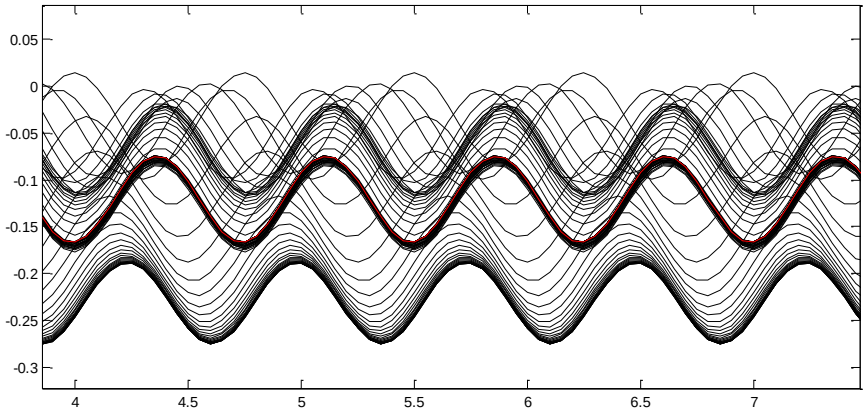
Následující experiment je úloha, která je přímo přirozeně frekvenčně kódovaná – naučit neuron chovat se jako dolní frekvenční propust. Zadání je následující: Máme síť s jedním vstupem a jedním výstupem (tedy stejná architektura jako na obrázku 9). Na vstupu máme po celý simulační čas ekvidistantní pulzy s frekvencí f. Úkolem je beze změny kopírovat pulzy na výstup, ale pouze pokud $f < f_h$, kde f_h je horní mezní frekvence. V případě, že $f \geq f_h$, nemá síť na výstupu vydávat žádný pulz.

Pokus byl prováděn se síti se simulačním intervalem $T = 15$, síti bylo předkládáno 10 učících vzorů s periodou 1 až 3,25 (v krocích po 0,25), cílem bylo naučit síť propouštět pulzy, pouze pokud perioda bude větší než 2,5. Bias výpočetního neuronu B byl nastaven na $w_{B0} = -2$, váha byla inicializována náhodným číslem větším než 2.

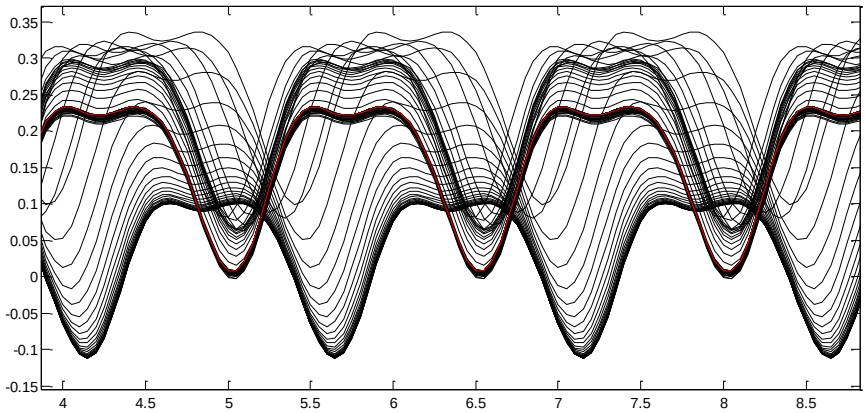


Obrázek 38: Vývoj kumulativní chyby všech vzorů v průběhu učení

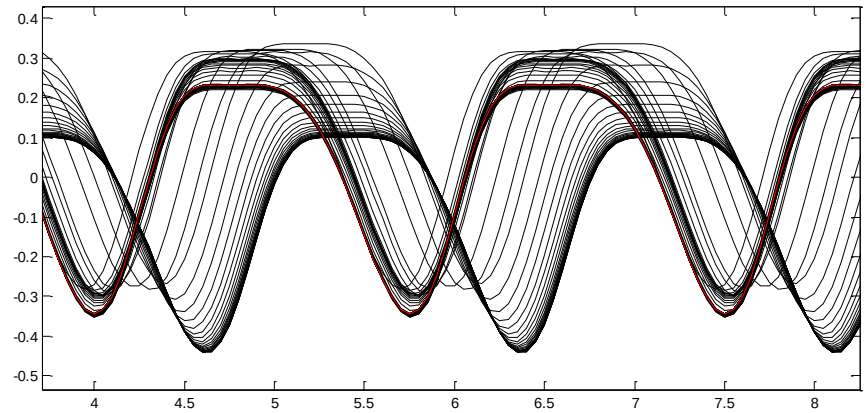
Po 100 iteracích je kumulativní chyba na všech vzorech 7,23777. Pokud pomineme jeden počáteční pulz na výstupu pro vyšší frekvence, síť se všechny vzory naučila korektně.



Obrázek 39: Výřez z grafu vývoje průběhu funkce ξ_B pro vstup s periodou 1,5



Obrázek 40: Výřez z grafu vývoje průběhu funkce ξ_B pro vstup s periodou 2 (nejnižší frekvence, která nevydává žádné výstupy)

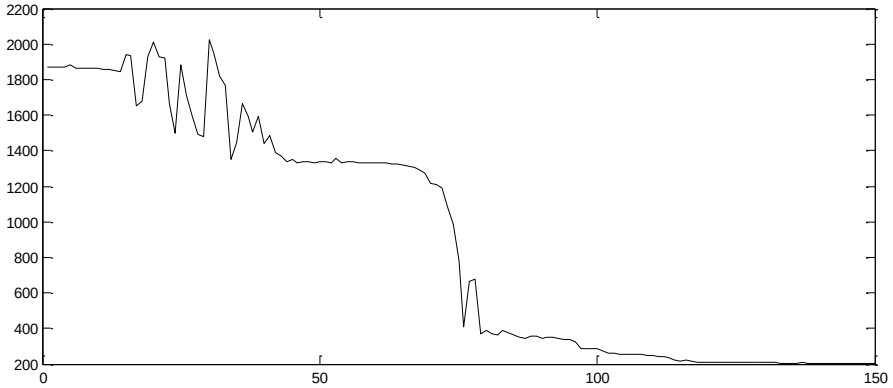


Obrázek 41: Výřez z grafu vývoje průběhu funkce ξ_B pro vstup s periodou 2,25 (nejvyšší frekvence, která vydává výstupy)

5.12 Horní frekvenční propust

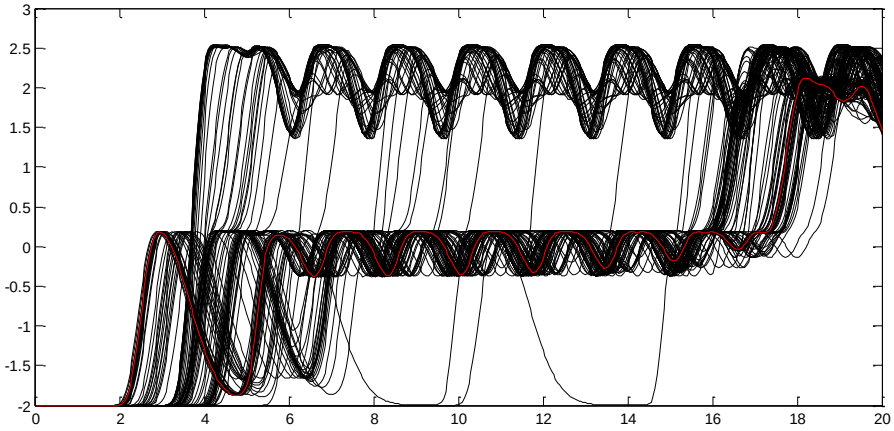
Poslední experiment, který zde uvádíme, je opakem úlohy předchozí – vstupem je série ekvidistantních pulzů s frekvencí f a úkolem je, aby síť zkopírovala pulzy na výstup, pokud $f \geq f_l$. Tuto úlohu není síť schopna řešit jedním neuronem, zvolili jsme architekturu se dvěma skrytými neurony (viz obrázek 27). Váhy byly opět inicializovány malými náhodnými čísly většími než opačná hodnota k biasu. Síti jsme předkládali deset učících vzorů s periodou 2 – 4,25 v krocích po 0,25. Síť měla reagovat na všechny vstupy s periodou $\frac{1}{f} \leq 3$. Simulační interval byl nastaven na $T = 20$, bylo provedeno 150 iterací učícího algoritmu. Naučená síť dává následující výstupy:

$\frac{1}{f}$	Výstupy
2	[0; 4.05232; 12.6865; 17.5903; 19.3816; 20]
2.25	[0; 2.72327; 5.38718; 9.4215; 10.4117; 14.0107; 17.9437; 20]
2.5	[0; 2.70671; 5.46252; 6.88108; 8.61892; 10.3455; 12.0504; 13.7132; 15.2915; 17.7602; 20]
2.75	[0; 2.77961; 6.85247; 8.41023; 10.3428; 12.2286; 14.032; 15.6886; 17.5814; 20]
3	[0; 2.92771; 10.8781; 12.7856; 14.5437; 16.0707; 20]
3.25	[0; 3.16726; 14.144; 20]
3.5	[0; 3.44436; 17.6572; 20]
3.75	[0; 3.56208; 20]
4	[0; 3.56208; 20]
4.25	[0; 3.56208; 20]

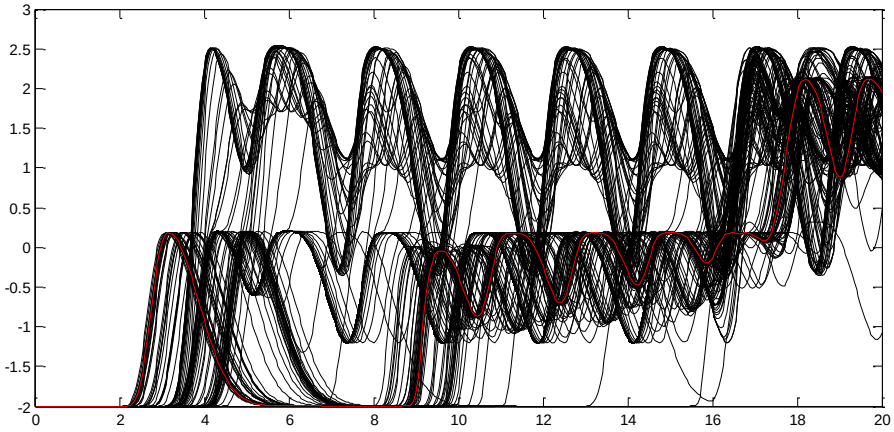


Obrázek 42: Vývoj kumulativní chyby všech vzorů v průběhu učení

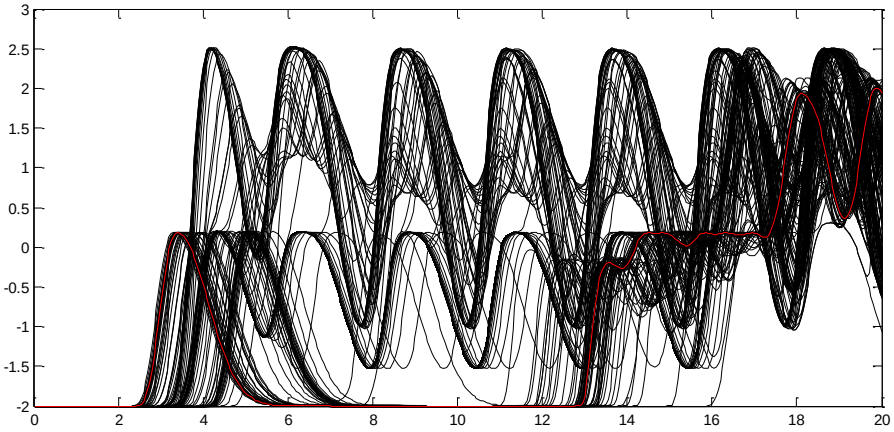
Algoritmus sice konverguje (kumulativní chyba po 150 iteracích učícího algoritmu byla 204,949), ale naučená funkce odpovídá spíše pásmovému filtru.



Obrázek 43: Vývoj průběhu funkce ξ_B pro vstup s periodou 2,5



Obrázek 44: Vývoj průběhu funkce ξ_B pro vstup s periodou 3



Obrázek 44: Vývoj průběhu funkce ξ_B pro vstup s periodou 3,25

6 Závěr

V práci byl popsán model sítě hladce pulzních neuronů s gradientním učícím algoritmem. Bylo navrženo několik modifikací modelu, které výrazně zvyšují výkon algoritmu a urychlují konvergenci. V experimentech byla podrobně zkoumána adaptační dynamika a schopnost sítě učit se jednoduché úlohy. Ukazuje se, že díky schopnosti přirozeně pracovat s více pulzy na jeden neuron se s použitím hladce pulzních neuronů otevírají možnosti řešení celé třídy úloh, ve kterých se přirozeně pracuje s frekvenčním kódováním. Sítě pulzních neuronů zatím obecně nedosahují tak dobrých výsledků jako tradiční modely a i do budoucna je nepravděpodobné, že by pulzní neurony mohly konkurovat schopnosti perceptronů se sigmoidální přenosovou funkcí adaptovat se na širokou škálu typů úloh; potenciální aplikace však spočívají ve třídě úloh, které jsou přirozeně definované v časové doméně, ve kterých je u tradičních modelů problém s efektivní reprezentací dat.

Další výzkum modelu hladce pulzních neuronů by se mohl týkat podrobnějšího zkoumání schopnosti neuronu pracovat s frekvenčním kódováním a úpravou učícího pravidla. Učící pravidlo s námi navrženými modifikacemi je sice schopné učit se jednoduché úlohy, ale u složitějších úloh se začíná projevovat numerická nestabilita celého systému. Jednou z možností by bylo vyzkoušet alternativní učící pravidla využívající genetické algoritmy nebo pokročilejší metody nelineární optimalizace.

7 Obsah CD

K práci je přiloženo CD, které obsahuje elektronickou verzi tohoto textu a zdrojové kódy programů používaných v experimentech, které by měly umožnit reprodukovat dosažené výsledky. Programy nejsou určeny pro běžné použití – v případě implementace v MATLABu jsou některé konstanty nastavené přímo ve zdrojovém kódu; v případě implementace napsané v C++ je hlavní knihovna napsána obecně, ale všechny přiložené programy, které jí používají, jsou neinteraktivní, učící vzory generují samy a pouze exportují výsledky a informace o průběhu učení.

7.1 MATLAB

Ve složce *matlab* jsou zdrojové kódy první referenční implementace algoritmu popsaného v kapitole 4.

Samotná síť je implementována jako objekt typu *snNetwork* a vytváří se funkcí stejnojmennou funkcí:

```
net = snNetwork([2 2 1], 30)
```

Vytvoří síť se dvěma vstupními, dvěma skrytými a jedním výstupním neuronem

```
result = snSim(net, {[5 7]; [6]})
```

Provede simulaci výpočtu sítě pro časy pulzů prvního vstupního neuronu nastavené na [5 7] a druhého na [6] (nultý a poslední formální pulz se doplní automaticky). Výstupem je buňka, která pro každý výstupní neuron obsahuje vektor časů pulzů.

```
[net error] = snTrain(net, inputs, outputs, epochs)
```

Provede *epochs* kroků učení se zadanými vzory. *inputs* a *outputs* jsou dvourozměrné buňky, kde ve kterých *i*-tý řádek, *j*-tý sloupec obsahuje vektor časů pulzů pro *j*-tý učící vzor, *i*-tý vstupní/výstupní neuron. Výstupem je síť s upravenými vahami a vektor v jednotlivých epochách

```
layers = snGetLayers(net)
```

Pro prozkoumání vnitřního stavu sítě je možné vyexportovat si z objektu *snNetwork* buňku se samotnými neurony. *i*-tý řádek v buňce *layers* představuje *i*-tou vrstvu a

j-tý řádek *j*-tý neuron ve vrstvě. Samotné neurony jsou objekty typu *snInputNeuron* nebo *snRegularNeuron*.

7.2 SpikeLib

Ve složce *spikelib* se nachází implementace všech popsaných algoritmů pro učení pulzních neuronů v C++, celkem ve třech verzích:

- *spikelib* bez přívlastku je implementací původního modelu popsaného v kapitole 4
- *spikelib-zero* je upravený model implementující změny popsané v kapitolách 5.2 a 5.3
- *spikelib-rp* je upravená verze *spikelib-zero* implementující úpravy popsané v kapitole 5.7

Pro každou verzi je na CD vlastní podadresář se *solution* pro *Visual Studio 2010*. V každé *solution* je projekt *SpikeLib* – statická knihovna obsahující samotnou implementaci algoritmu a několik dalších projektů implementujících samotné experimenty jako neinteraktivní konzolové aplikace.

- *spikelib* obsahuje projekt
  - *ConstDelayTest* použitý v kapitole (5.1). Program generuje tři výstupní soubory: *error.csv* s vývojem chyby a soubory *Layer 0 Neuron 0.csv* a *Layer 1 Neuron 0.csv*, obsahujících vývoj funkce ξ v příslušných neuronech
- *spikelib-zero* obsahuje projekty
  - *ConstDelayTest* použitý v kapitole (5.4). Program generuje soubory *error.csv* (s průběhem chyby E_2) a *Layer 1 Neuron 0.csv* s vývojem funkce ξ výstupního neuronu během učení.
  - *AndTestSimple* použitý v kapitole (5.5). Program generuje opět soubory *error.csv* a *Layer 1 Neuron 0.csv* – ten obsahuje vývoj funkce ξ na čtvrtém učícím vzoru.
  - *AndTestFreq* použitý v kapitole (5.6). Program generuje soubor *error.csv* a čtyři soubory *Output Pattern 0-3.csv* s průběhy funkce ξ pro příslušné učící vzory.
- *spikelib-rp* obsahuje projekty
  - *AndTest* použitý v kapitole (5.8). Program generuje soubor *error.csv* a čtyři soubory *Output Pattern 0-3.csv* s průběhy funkcí ξ pro příslušné vzory.
  - *XorTest* použitý v kapitole (5.9), stejné výstupy jako u *AndTest*.

- *TestXorSingle* použitý v kapitole (5.10), stejné výstupy jako u *AndTest*.
- *FreqTest* použitý v kapitole (5.11). Program generuje soubor *error.csv* a celkem deset souborů s vývojem funkce ξ výstupního neuronu pro všechny učící vzory.
- *HighpassFilter* použitý v úloze (5.12). Program generuje stejné výstupy jako *FreqTest*.

8 Seznam literatury

1. **McCulloch, Warren a Pitts, Walter.** A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5*. 1943.

2. **Rosenblatt, Frank.** The perceptron: A probabilistic model for information storage and organization in the Brain. *Psychological Review v65*. 1958.

3. **Bryson, Arthur a Ho, Yu CHi.** *Applied Optimal Control.* Washington, DC : Hemisphere, 1975.

4. **Rumelhart, David E. a Hinton, Geoffrey E.** Learning internal representation by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge : MIT Press, 1986.

5. **Hodgkin, Alan a Huxley, Andrew.** A quantitative description of membrane current and its applications to conduction and excitation in nerve. *The Journal of Physiology* . 1952.

6. **Mehta, M. R., Lee, A. K. a Wilson, M. A.** Role of experience of oscillations in transforming a rate code into a temporal code. *Nature, v417*. 2002.

7. **Popović, Dejan B. a Sinkjaer, Thomas.** *Control of Movement for the Physically Disabled.* London : Springer, 2000.

8. **Bohte, Sander m, Kok, Joost N. a Poultré, Han La.** Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing, v48*. 2002.

9. **Moore, Simon Christian.** *Back-Propagation in Spiking Neural Networks. M.Sc. thesis.* místo neznámé : University of Bath, 2002.

10. **Berkovec, Karel.** Učení sítí se spiking neurony, diplomová práce. 2005.

11. **Pfister, Jean Pascal, Toyoizumi, Taroi a Gerstner, Wulfram.** Optimal spike-timing dependent plasticity for precise action potential firing in supervised learning. *Neural Computation, v18*. 2006.

12. **Belatreche, Ammar, a další.** A Method for Supervised Training of Spiking Neural Networks. *Proceedings of IEEE Cybernetics Intelligence - Challenges and Advances (CICA).* 2003.

13. **Šíma, Jiří.** *Gradient Learning in Networks of Smoothly Spiking Neurons (Revised Version).* 2009.

14. **Janík, Tomáš.** *Implementace gradientního učícího algoritmu pro sítě hladce pulzních neuronů.* Brno : Masarykova univerzita, Fakulta informatiky, 2009.

15. **Goldman, David E.** Potential, impedance, and rectification in membranes. *The Journal of General Physiology.* místo neznámé : Rockefeller University Press, 1943.

16. **Maas, Wolfgang.** Lower bounds for the computational power of networks of spiking neurons. *Neural Computation, v8*. 1996.

17. —. Paradigms for Computing with Spiking Neurons. *Models of neural networks, v4*. New York : Springer, 2002.

18. **Šíma, Jiří a Sgall, Jiří.** On the non-learnability of a single spiking neuron. *Neural Computation, v17*. 2005.

19. **Carnell, Andrew, Richardson, Dan.** *Linear algebra for time series of spikes.* 2004.

20. **Pfister, Jean Pascal, Barber, David a Gerstner, Wulfram.** Optimal Hebbian Learning: a Probabilistic Point of View. *Lecture notes in Computer Science.* Berlin : Springer, 2003.

21. **Ruf, Berthold a Schmitt, Michael.** Learning temporally encoded patterns in networks of spiking neurons. *Neural Processing Letters, v5*. 1997.

22. **Ponulak, Filip.** *ReSuMe - new supervised learning method for spiking neural networks - tech. rep.* Poznan University : Institute of Control and Information Engineering, 2005.

23. **Xin, J. a Embrechts, M.J.** Supervised Learning with Spiking Neuron Networks. *Proceedings of IEEE International Joint Conference on Neural Networks.* Washington, DC : autor neznámý, 2001.

24. **Tino, Peter a Mills, Ashley J.S.** Learning Beyond Finite Memory in Recurrent Networks of Spiking Neurons. [autor knihy] Lecture Notes in Computer Science. Berlin : Springer, 2005.

25. **Riedmiller, Martin a Braun, Heinrich.** A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *IEEE International Conference on Neural Networks.* 1993.

26. **Gerstner, Wulfram.** Time structure of activity in neural network models. *Physical Review.* 1995.