TECHNICKÁ UNIVERZITA V LIBERCI
FAKULTA MECHATRONIKY A MEZIOBOROVÝCH INŽENÝRSKÝCH STUDIÍ

FONETICKÁ TRANSKRIPCE ČEŠTINY POMOCÍ TŘÍVRSTVÉ NEURONOVÉ SÍTĚ

Výzkumná zpráva č. ISRN-TUL-KES-T-PZ-00-005-C1-CZ

Ing. Dana NEJEDLOVÁ

Liberec **Listopad 2000**

1. Teoretický úvod

V teoretickém úvodu popisuji některé druhy neuronových sítí a problematiku fonetické transkripce češtiny, což je jedna z úloh řešitelných neuronovou sítí.

1.1. Co je neuronová síť [1]

Oblast umělých neuronových sítí představuje biologií inspirovaný přístup k řešení problémů. Neuronové sítě jsou matematické modely zpracování informací, které poprvé publikovali neurofyziolog Warren McCulloch a matematik Walter Pitts v roce 1943 v práci [2]. Jejich model vystřídal v roce 1962 model perceptronu Franka Rosenblatta, viz [4], který vzbudil více zájmu, protože dokázal řešit jednoduché klasifikační problémy. Tento zájem upadl po roce 1969, kdy Marvin Minsky a Seymour Papert v práci [13] matematicky dokázali omezení perceptronu. V 80. letech se zájem o neuronové sítě obnovil, protože byly vynalezeny nové architektury neuronových sítí a nové matematické metody jejich učení. Tento zájem je dnes umocňován rychlým rozvojem výpočetní techniky.

Na rozdíl od tradičního způsobu sekvenčního zpracování informací v jednoprocesorových počítačích, výpočty v neuronových sítích probíhají paralelně ve velkém počtu jednoduchých jednotek zvaných neurony. Informace je distribuována do celé sítě namísto toho, aby měla svou adresu v paměti počítače, a proto se tento způsob výpočtů někdy nazývá paralelně distribuované zpracování informací (*parallel distributed processing*). Mohou být definovány jednoduché učící algoritmy, pomocí kterých se upravují spoje mezi neurony tak, aby reagovaly na zkušenost představovanou zpracovávanými daty. Z neurofyziologie víme, že růst a zánik synapsí mezi neurony v mozcích organismů je důsledkem jejich učení, a tak nám neuronové sítě poskytují vhled do problémů řešených kognitivními vědami, například, jak miliardy interakcí mezi biologickými neurony vyústí v globální chování jedince. Kromě teoretického zkoumání se neuronové sítě s úspěchem využívají i v praktických komerčních aplikacích. Bylo vyvinuto mnoho druhů neuronových sítí, z nichž některé přibližuji v této výzkumné zprávě.

1.2. Jak neuronová síť pracuje [1]

Neuronová síť je systémem vzájemně propojených jednotek zvaných neurony. Každý neuron má svoji **přenosovou funkci** (*transfer function*), která vypočte vstup do neuronu tak, že sečte součiny výstupních signálů s vahami na příslušných spojích ze všech neuronů, od kterých do daného neuronu vede spojení, a od tohoto součtu odečte prahovou hodnotu patřící danému neuronu. Signály vycházející z neuronů, váhy na spojích mezi neurony a prahové hodnoty jsou reálná čísla. Výstup přenosové funkce je vstupem **aktivační** neboli **prahové funkce** neuronu (*activation/threshold function*), která jej transformuje na výstupní signál daného neuronu, který má shora i zdola omezenou hodnotu typicky od 0 do 1. Aktivační funkce může mít různé podoby, například schod (výstupem neuronu je 0, když součet vstupů nepřekročí prahovou hodnotu, nebo 1 v opačném případě) nebo sigmoida (zaoblený schod), kde výstupem neuronu je hodnota z intervalu od 0 do 1.

1.3. Jak se neuronová síť učí [1]

Existují dva hlavní způsoby učení neuronových sítí:

1. **učení s dohledem** (*supervised learning*),
2. **učení bez dohledu** (*unsupervised learning*).

Při 1. způsobu učení jsou síti předkládány příklady a správné výsledky pro každý z nich a síť se postupně úpravou svých vah učí vracet pro každý příklad správný výsledek. Modifikací tohoto postupu je, že místo výsledku je síti sdělováno pouze to, zda se mýlí či ne. Při 2. způsobu učení jsou síti předkládána data a síť upravuje své váhy tak, aby splnila nějaké své vnitřní kritérium pro úpravu vah. Při tom dochází k takzvané samoorganizaci vah a z dat se postupně vytvářejí předem neznámé kategorie. Data určená k učení se nazývají **vzory**. Vzorem je většinou vektor čísel vstupujících do vstupní vrstvy sítě. **Vrstva** sítě je skupina jejích neuronů, ve kterých se mohou signály zpracovávat paralelně, tedy ve stejný okamžik.

1.4. Bližší popis některých neuronových sítí

1.4.1. The Interactive Activation and Competition Network neboli síť pro interaktivní aktivaci a soutěžení [1]

Síť pro interaktivní aktivaci a soutěžení, dále zmiňovaná jako síť IAC, byla poprvé publikována Američany Jay McClellandem a Dave Rumelhartem v roce 1981 v práci [3]. Tato síť má mnoho vlastností, které dělají z neuronových sítí užitečné modely pro zpracování informací.

Síť IAC se skládá ze skupin vzájemně soutěžících jednotek (neuronů), kde každá jednotka představuje nějakou mikrohypotézu nebo vlastnost. Jednotky ve společné skupině představují vzájemně se vylučující vlastnosti a váhy, kterými jsou mezi sebou propojeny, jsou proto záporné. Pro spoje mezi jednotkami patřícími do různých skupin platí, že jsou na nich kladné váhy, pokud vlastnosti představované jimi propojenými jednotkami se vzájemně nevylučují. Všechny spoje jsou obousměrné.

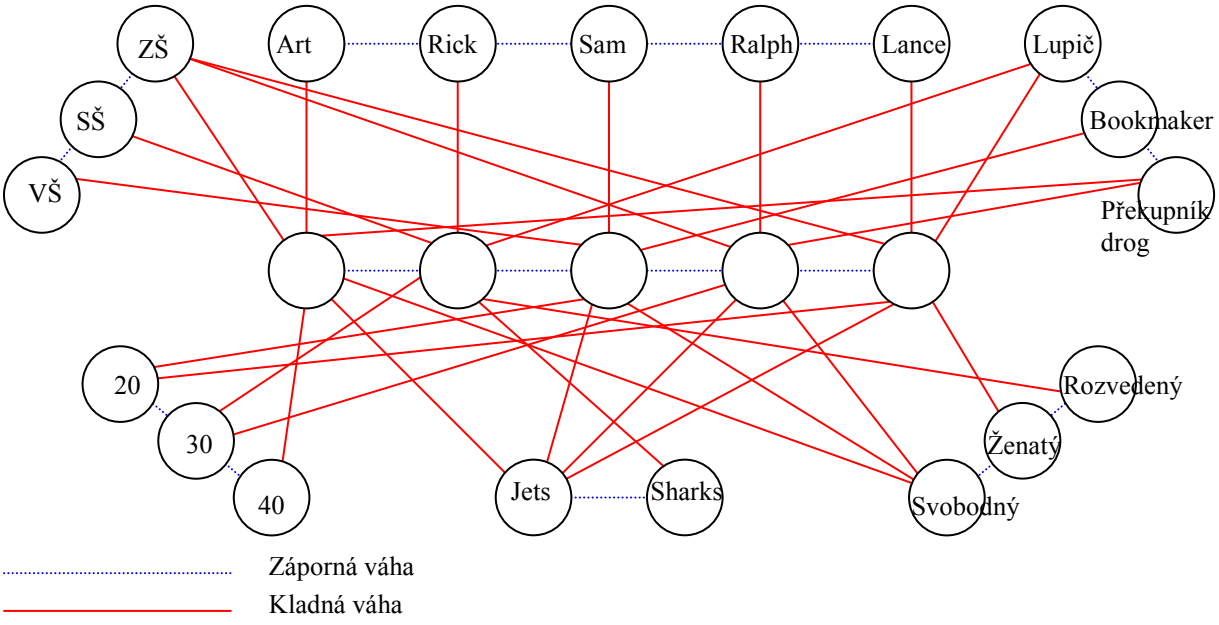
Následuje příklad, jak lze do sítě IAC zakódovat informace o dvou soupeřících ganzích. Jsou známy následující informace o členech gangů: jméno (Art, Rick, Sam, Ralph, Lance), způsob obživy (lupič, bookmaker, překupník drog), rodinný stav (svobodný, ženatý, rozvedený), příslušnost k gangu (Jets, Sharks), přibližný věk (20, 30, 40-cátník), a dosažené vzdělání (ZŠ, SŠ, VŠ). Skupina pěti jednotek uprostřed sítě na obrázku č. 1 představuje členy gangů. Od každé jednotky z této skupiny vede jeden spoj s kladnou váhou do každé okolní skupiny jednotek, která představuje vlastnost patřící danému členu gangu. Spoje mezi jednotkami uvnitř stejné skupiny mají záporné váhy. Jednotky ve stejné skupině jsou propojeny každá s každou, což z obrázku není vidět. Tabulka č. 1 obsahuje informace zakódované do sítě.

Tabulka č. 1: Informace o členech gangů

Jméno	**Gang**	**Přibližný věk**	**Vzdělání**	**Rodinný stav**	**Způsob obživy**
Art	Jets	40	ZŠ	svobodný	překupník drog
Rick	Sharks	30	SŠ	rozvedený	lupič
Sam	Jets	20	VŠ	svobodný	bookmaker
Ralph	Jets	30	ZŠ	svobodný	překupník drog
Lance	Jets	20	ZŠ	ženatý	lupič

Pramen: [1]

Obrázek č. 1: Síť IAC představující informace o členech gangů



Pramen: [1]

Hodnota patřící určité jednotce se nazývá její aktivace. Jednotka s vysokou aktivací je aktivní. Vytvořenou síť lze použít k vyvolávání do ní uložených informací. Chceme-li například zjistit vše o Artovi, přiřadíme jednotce pro jméno Art hodnotu 1 a všem ostatním jednotkám v síti hodnotu 0 a síť necháme cyklit. Při cyklení jednotky napojené kladnými vahami na aktivní jednotky se také stávají aktivními, zatímco aktivace jednotek napojených zápornými vahami na aktivní jednotky se snižuje. Po dostatečném počtu cyklů se v síti zaktivují jednotky představující vlastnosti v 1. řádku tabulky č. 1.

Mechanismus IAC je následující. Aktivace každé jednotky může být považována za míru důvěry v hypotézu nebo vlastnost, kterou tato jednotka představuje. Váhy mezi jednotkami v síti indikují, jak silně víra v jednu hypotézu implikuje víru v jinou hypotézu. Mechanismus IAC je cyklický proces aktualizace víry v hypotézu v závislosti na aktuálních důkazech.

Pro všechny jednotky v síti vypočteme jejich aktivaci ze vstupů od ostatních jednotek a hodnot na vahách mezi těmito jednotkami. Hodnota vstupu do *i*-té jednotky bude

$net_i = \Sigma w_{ij}\, a_j$,

kde w_{ij} je váha na spoji mířícím od *j*-té jednotky do *i*-té jednotky a a_j je aktivace *j*-té jednotky.

Jakmile jsou spočítány hodnoty vstupů do všech jednotek v síti, jsou aktualizovány aktivace jednotek podle následující rovnice:

když je ($net_i > 0$), potom $\Delta a_i = (max - a_i)\ net_i - decay\ (a_i - rest)$,
v ostatních případech $\Delta a_i = (a_i - min)\ net_i - decay\ (a_i - rest)$,
kde Δa_i je hodnota, o kterou změníme aktivaci i-té jednotky.

Typicky volíme $max > 0 >= rest >= min$. Konstanta *decay* je kladné číslo v intervalu (1, 0).

Konstanta *max* je rovna maximální možné aktivaci jednotky. Konstanta *min* je rovna minimální možné aktivaci jednotky. Členy ($max - a_i$) a ($a_i - min$) zaručují, že aktivace jednotek se bude snažit zůstávat v rozmezí od *min* do *max*. Odečtení členu *decay* ($a_i - rest$) nutí aktivaci vrátit se do hodnoty *rest*, pokud je vstup do i-té jednotky nulový.

V síti, která cyklicky upravuje aktivace svých jednotek podle těchto pravidel, lze pozorovat dva hlavní jevy. Jednotky napojené klanými vahami na nejaktivnější jednotku zvyšují svoji aktivaci a jednotky napojené zápornými vahami na nejaktivnější jednotku své aktivace snižují.

Na síti IAC lze demonstrovat některé vlastnosti typické pro všechny neuronové sítě. Těmito vlastnostmi jsou:

1. **adresovatelnost informace podle obsahu** (*content addressability*),
2. **odolnost vůči šumu v informacích** (*robustness to noise*),
3. **schopnost zobecnění** (*generalization*),
4. **schopnost určit důvěryhodné hodnoty pro chybějící informace** (*default assignment*).

1. vlastnost znamená, že neuronové sítě hledají požadovanou informaci podobně jako lidé podle jejího obsahu nebo kontextu a nikoliv podle adresy v paměti počítače nebo primárního klíče v databázi. Když například v síti na obrázku č. 1 aktivujeme jednotky pro věk okolo 30 a gang Jets, síť při cyklení zaktivuje také všechny ostatní jednotky patřící do 4. řádku tabulky č. 1, protože pouze ten má danou dvojici hodnot 30 a Jets.

2. vlastnost znamená, že neuronové sítě dokáží u nepřesného nebo částečně chybného vzoru určit jeho správnou hodnotu nebo kategorii. Když například v síti na obrázku č. 1 aktivujeme jednotky patřící určitému řádku tabulky č. 1 a uděláme přitom chybu jako třeba, že zapomeneme nějakou jednotku zaktivovat nebo zaktivujeme nějakou nesprávnou jednotku, síť při cyklení naši chybu opraví.

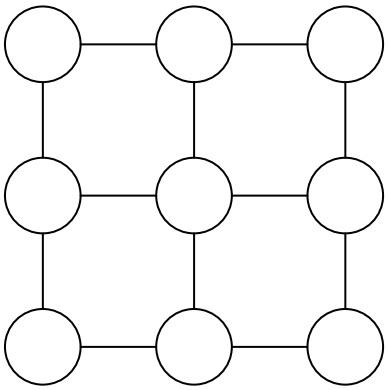
3. vlastnost znamená, že neuronové sítě dovedou z množiny příkladů určit, co v této množině platí obecně. Když například v síti na obrázku č. 1 aktivujeme jednotku pro gang Sharks, v ostatních skupinách jednotek pro jiné vlastnosti se zaktivují jednotky patřící Rickovi, který jediný patří do tohoto gangu.

4. vlastnost znamená, že neuronové sítě dovedou rekonstruovat chybějící informaci ve vzoru tak, že za ni dosadí to, co se většinou nachází na jejím místě v podobných vzorech.

1.4.2. Self-Organizing Map (SOM) neboli Samoorganizujícíse mapa [1]

V samoorganizujícíse mapě, dále označované jako SOM, kterou poprvé publikoval finský vědec Teuvo Kohonen v roce 1982 v článku [9] probíhá učení bez dohledu. SOM se skládá ze vstupní vrstvy, která je plně propojená s výstupní vrstvou. Znamená to, že z každého neuronu ve vstupní vrstvě vede synapse do všech neuronů ve výstupní vrstvě. Neurony ve výstupní vrstvě jsou mezi sebou obvykle propojeny do mřížky. Když vstupní vrstva vyšle do výstupní vrstvy vzor, neurony ve výstupní vrstvě soutěží mezi sebou o to, který z nich bude tento vzor reprezentovat. Vítězí neuron, jehož vektor vah je nejpodobnější vzoru. Vítěz a jeho sousední neurony v mřížce upraví své váhy tak, aby byly vzoru ještě podobnější. Protože se upravují i váhy neuronů blízko vítězů, při trénování se postupně na mřížce vytvoří oblasti reagující na podobné vzory. Čím jsou vzory odlišnější, tím vzdálenější jsou oblasti mřížky, které na ně reagují.

Obrázek č. 2: Síť neuronů Kohonenovy mapy



Pramen: [6]

Algoritmus, podle kterého se upravují váhy v SOM, se nazývá **soutěživý** (*competitive*). Podle tohoto algoritmu se porovná vzor s vektorem vah každého neuronu ve výstupní vrstvě a vybere se neuron s nejpodobnějším vektorem vah. Vyhraje-li i-tý neuron, musí platit, že
$|\mathbf{w}_i - \mathbf{x}| < |\mathbf{w}_j - \mathbf{x}|$ pro všechna j,
kde $\mathbf{w}_j$ je váhový vektor j-tého neuronu a $\mathbf{x}$ je právě čtený vzor.

Vítězný neuron a jeho sousedé si upravují váhy podle vzorce
$w_{jk} = w_{jk} + \Delta w_{jk}$, kde $\Delta w_{jk} = r \cdot N(i, j)\ (x_k - w_{jk})$,
kde
w_{jk} je váha na synapsi vedoucí ze vstupní jednotky k do jednotky mřížky j,
r je parametr rychlosti učení,
$N(i, j)$ je míra blízkosti jednotky j a vítězné jednotky i,
x_k je k-tá složka vstupního vektoru $\mathbf{x}$.

Funkce $N(i, j)$ dává nejvyšší hodnotu pro tu samou jednotku ($i = j$) a klesá, když se zvyšuje fyzická vzdálenost neuronů v mřížce. Může mít například následující podobu:

$N(i, j) = 1/(1 + d(i, j) / (20^2 * n^2))$
$d(i, j) = (x_i - x_j)^2 + (y_i - y_j)^2$,
kde x_j je x-ová souřadnice jednotky j, y_j je ypsilonová souřadnice jednotky j,
20 je šířka jednotky v pixelech a
n je parametr ovlivňující rychlost poklesu funkce s růstem vzdálenosti jednotek.

Za sousedy vítězného neuronu je vhodné považovat neurony, jejichž vzdálenost od něj je nižší než nějaká stále se snižující hodnota. Pro negrafickou verzi sítě je možné stanovovat míru blízkosti jednotek podle počtu spojů, kterými je na mřížce potřeba projít od jednotky k jednotce.

Sítě podobné SOM byly používány jako modely mapování vjemů v biologických mozcích.

1.4.3. Hebbova síť [1]

Hebbova síť může sloužit k jednoduché reprezentaci faktů. Skládá se ze dvou vrstev neuronů neboli jednotek. Neurony v sousedních vrstvách jsou propojeny každý s každým. Hebbovou sítí budeme reprezentovat vztah mezi dvěma entitami například, že nějaké zvíře jí nějakou potravu. Chceme-li do sítě uložit 4 různá zvířata, musí mít vstupní vrstva sítě alespoň 4 jednotky. Jedí-li tato zvířata 3 různé druhy potravy, musí mít výstupní vrstva alespoň 3 výstupní jednotky. Zvířata a potravu budeme reprezentovat jako ortogonální vektory, například

rosnička	(1, 0, 0, 0) jí	mouchy	(0, 1, 0),
šimpanz	(0, 1, 0, 0) jí	banány	(1, 0, 0),
vlk	(0, 0, 1, 0) jí	maso	(0, 0, 1),
pavouk křižák	(0, 0, 0, 1) jí	mouchy	(0, 1, 0).

Tento soubor faktů můžeme uložit do sítě tak, že hodnoty složek vektoru reprezentujícího zvíře vstoupí do vstupních jednotek sítě a hodnoty složek vektoru reprezentujícího jeho potravu vstoupí do výstupních jednotek sítě. Poté se vypočtou hodnoty vah na všech spojích podle vzorce pro Hebbovo učení:
$w_{ij} = w_{ij} + \Delta w_{ij}$, kde $\Delta w_{ij} = a_i \cdot a_j$. Počáteční váhy jsou rovny nule.
a_i je aktivace i-té jednotky vstupní vrstvy.
a_j je aktivace j-té jednotky výstupní vrstvy.

Hebbovo učení je pojmenováno po kanadském psychologovi Donaldu Oldingu Hebbovi, který v roce 1949 v díle [10] formuloval pravidlo pro učení synapse neuronu říkající, že průchodnost synapse mezi neurony se zvyšuje, když jeden neuron vysílá signál ke druhému, na kterém tento signál překročí prahovou hodnotu a neuron vypálí signál dál. Jednodušeji řečeno, váha na spoji mezi neurony se zvyšuje, když je aktivace na obou neuronech vysoká.

Hodnoty přírůstku vah po přečtení jedné dvojice zvíře-potrava je možno zapsat jako matici, která vznikne tenzorovým součinem vektorů pro zvíře (sloupcový vektor) a jeho potravy (řádkový vektor). Když sečteme takovéto matice za všechny dvojice zvíře-potrava, dále jen vzory, dostaneme matici zaznamenávající konečný stav vah po Hebbově učení všech vzorů. Z konečného stavu vah lze pro každé v síti uložené zvíře vyvolat informaci o jeho potravě tak, že do vstupních jednotek sítě vložíme hodnoty složek vektoru zvířete a hodnoty výstupních jednotek se vypočtou podle vzorce:
$a_j = \Sigma w_{ij}\, a_i$,
který se používá i u dalších druhů neuronových sítí.

Použijeme-li vektorový zápis, dostaneme pro ukládání faktů do vah sítě vzorec
$\mathbf{M} = \mathbf{M} + \mathbf{a}_{\text{zvíře}} \cdot \mathbf{a}_{\text{potrava}}^{\text{T}}$, kde $\mathbf{M}$ je matice vah, která má na začátku všechny prvky rovny nule.

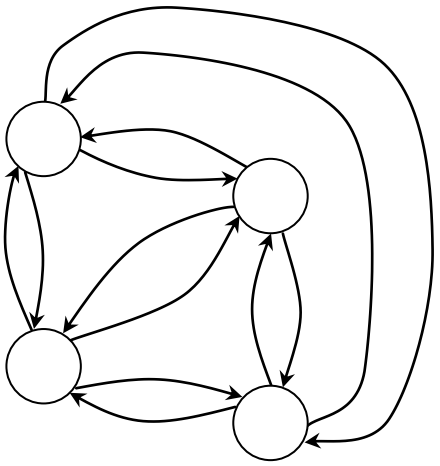
Pro vyvolání informace o tom, co jí určité zvíře, použijeme potom vzorec
$\mathbf{a}_{\text{potrava}}^{\text{T}} = \mathbf{a}_{\text{zvíře}}^{\text{T}} \cdot \mathbf{M}$, kde $\mathbf{a}_{\text{potrava}}^{\text{T}}$ nebo $\mathbf{a}_{\text{zvíře}}^{\text{T}}$ je transponovaný neboli řádkový vektor.

To, že jsme zvířata nebo potravu reprezentovali jako vektor, který měl všechny složky až na jedinou rovny nule se dá interpretovat jako to, že máme 4 třídy zvířat a 3 třídy potravy a pro každou ze tříd má Hebbova síť zvláštní jednotku. Takovýto způsob **reprezentace** se nazývá **lokální**. Jejím protikladem je **distribuovaná reprezentace**.

Distribuovaná reprezentace vznikne, když reprezentujeme entitu (v našem příkladě zvíře nebo potravu) jako vektor s libovolnými složkami namísto výběru určité jednotky, která má být jediná aktivována. Tento způsob reprezentace nám umožní používat naši síť pro další zvířata, aniž bychom museli přidávat nové jednotky. Stačí pouze dodržet pravidlo, že podobná zvířata a podobné potraviny budou reprezentovány podobnými vektory. Vytvoříme například vzor ropucha (0.9, 0, 0.1, -0.1), pro který nám síť vrátí na výstupní vrstvě vektor velmi blízký vektoru pro mouchy.

1.4.4. Hopfieldova síť [1]

Obrázek č. 3: Hopfieldova síť



Pramen: [6]

Hopfieldova síť byla poprvé publikována v roce 1982 v práci [11] a 1984 v práci [12] americkým laureátem Nobelovy ceny za fyziku Johnem Hopfieldem. Spolu s výzkumníkem z AT&T Davidem Tankem vymyslel Hopfield množství sítí s fixními vahami a měnitelnými aktivacemi neuronů. Nejdůležitější vlastností těchto sítí je **adresovatelnost informace podle obsahu** (*content addressability*, viz kapitola 1.4.1.). Hopfieldova síť dokáže skladovat množinu vzorů. Vložený vzor z ní může být získán zpracováním jeho částečně poškozené verze. Typickými vzory pro tuto úlohu jsou dvojbarevné obrázky.

V Hopfieldově síti je každý neuron propojen obousměrně se všemi ostaními. Má-li síť uchovávat obrázky, jsou neurony uspořádány do rastru, a potom každý z nich odpovídá jednomu pixelu na obrázku. Na rozdíl od ostatních zde probíraných sítí je každý neuron Hopfieldovy sítě aktivován hodnotou +1 nebo –1. Přenosová funkce do *j*-tého neuronu je opět

$a_j = \Sigma w_{ij} \cdot a_i$,

tedy vstup do neuronu a_j se vypočte jako suma součinů aktivací všech neuronů v síti s příslušnými vahami, které od neuronů s těmito aktivacemi míří do neuronu a_j. Aktivační funkce dává neuronu aktivaci s hodnotou +1, když je hodnota jeho přenosové funkce větší než nula a v opačném případě je výslednou aktivací neuronu –1. Prahovou hodnotou neuronů je tedy 0. Pořadí, v jakém se aktivace neuronů při obnovování poškozeného vzoru aktualizují, může významně tento proces ovlivnit. Existují dvě hlavní **metody aktualizace: synchronní a asynchronní**. Při synchronní aktualizaci se aktualizují v každém kroku všechny neurony najednou. Při asynchronním způsobu se v každém kroku aktualizuje náhodně vybraný neuron. Asynchronní aktualizace je rychlejší, protože se každý neuron aktualizuje podle stavu po aktualizaci předchozího vybraného neuronu, kdežto při synchronní aktualizaci se každý neuron v jednom kroku aktualizuje podle stavu na začátku tohoto kroku.

Ve fázi vkládání vzorů využívá Hopfieldova síť Hebbovo učení

$\Delta w_{ij} = a_i \cdot a_j$

známé z předchozí kapitoly.

Používání Hopfieldovy sítě má tedy fázi učení, kdy se síti předkládají vzory a po každém z nich síť přičte ke svým zpočátku vynulovaným vahám hodnotu $\Delta w_{ij} = a_i \cdot a_j$, kde hodnotou neuronu a je buďto +1 nebo –1 například podle toho, zda je příslušný pixel ukládaného obrázku bílý nebo černý. Druhou fází používání sítě je vyvolávání v ní uložených vzorů. Při ní se do neuronů sítě načte poškozený vzor a potom síť synchronně nebo asynchronně cyklí, neboli aktualizuje aktivace neuronů z počátečního stavu daného aktivacemi podle poškozeného vzoru pomocí přenosové a aktivační funkce. Vzhledem k tomu, že $w_{ij} \cdot a_i$ z přenosové funkce pro a_j se rovná w_{ij} / a_i, které lze odvodit pro a_j u Hebbova učení používaného při ukládání vzorů, bude se při cyklické aktualizaci neuronů v síti stav aktivací neuronů přibližovat aktivacím určeným uloženým vzorem, který je nejbližší obnovovanému poškozenému vzoru. V případě, kdy je v síti uložen jediný vzor, platí, že pro jeho vyvolání musí být na začátku jeho obnovování alespoň polovina neuronů aktivována jako tento vzor. Bude-li alespoň polovina neuronů aktivována jako tento vzor ale inverzně, síť vrátí uložený vzor inverzní, protože $+1 \cdot +1 = -1 \cdot -1$.

Kapacita Hopfieldovy sítě je omezená. Hertz, Krogh a Palmer v roce 1991 dokázali, že pro náhodně vytvořené vzory platí, že výkon Hopfieldovy sítě bude dobrý, pokud počet vzorů bude menší než 0,138 krát počet jejích neuronů.

Hopfield v roce 1984 dokázal, že jeho síť při obnovování vzoru konverguje do stabilního stavu aktivace neuronů, pomocí Lyapunovy funkce energie zde označované jako *E*.

$$E = -\frac{1}{2}\sum_i \sum_j a_i a_j w_{ij}$$

Funkce *E* má tu vlastnost, že nikdy nestoupá. Po aktualizaci *i*-tého neuronu o hodnotu Δa_i se energie sítě změní o

$$\Delta E = -\frac{1}{2} \cdot \Delta a_j \sum_i a_i w_{ij}$$

Podle přenosové funkce do *j*-tého neuronu platí $a_j = \Sigma a_i \cdot w_{ij}$. Podle aktivační funkce platí, že když je $\Sigma a_i \cdot w_{ij} > 0$, tak se a_j změní buď o nulu nebo z –1 na +1 tedy o +2. Ve vzorečku pro ΔE je tedy $\Delta a_j = +2$ a

$\Sigma a_i \cdot w_{ij} > 0$. Z toho plyne, že ΔE vyjde záporné a energie se tedy sníží. Kdyby $\Sigma a_i \cdot w_{ij} < 0$, tak by se a_j změnilo buď o nulu nebo z +1 na –1 tedy o –2. Ve vzorečku pro ΔE by tedy $\Delta a_j = -2$ a $\Sigma a_i \cdot w_{ij} < 0$. Z toho plyne, že ΔE by opět vyšlo záporně.

Po zavedení funkce energie je možné si představit vzory uložené v síti jako důlky. Tyto důlky se odborně nazývají **atraktory**. Poškozený vzor, pro který síť při cyklení hledá nejpodobnějšího v ní uloženého zástupce, se jakoby pohybuje z místa s vyšší energií do nejbližšího důlku. Představa přesunu bodu o souřadnicích daných buďto měnitelnými aktivacemi neuronů, jako je tomu u Hopfieldovy sítě, nebo měnitelnými vahami, jako je tomu například u sítě MLP z následující kapitoly, do místa s nižší energií je v teorii neuronových síti často užitečná.

1.4.5. The BackPropagation Network neboli Multilayer Perceptron (MLP) neboli vícevrstvá neuronová síť

[1]

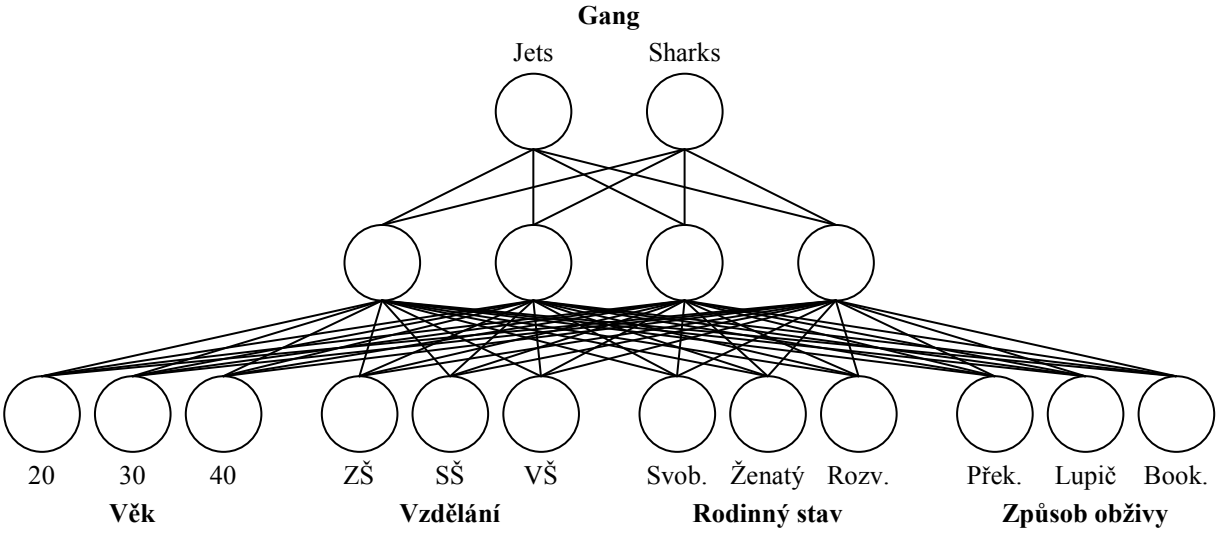
V mnoha reálných situacích musíme zpracovávat neúplná nebo nepřesná data a činit předpovědi o tom, co v dostupných informacích schází. To může být obzvlášť obtížné, pokud chybí dobrá teorie, podle které by mohla být nedostupná data rekonstruována. V těchto situacích může být užitečná vícevrstvá neuronová síť, dále označovaná jako MLP.

MLP se skládá alespoň ze 3 vrstev: vstupní vrstvy, alespoň jedné skryté (prostřední) vrstvy a výstupní vrstvy. Na rozdíl od sítí IAC a Hopfieldových jsou váhy v MLP jednosměrné a vedou ze vstupní vrstvy do skryté a ze skryté vrstvy do výstupní. Neurony v sousedních vrstvách jsou většinou propojeny každý s každým.

Výstup MLP můžeme označit jako klasifikační rozhodnutí. Na rozdíl od sítě IAC, kam se musí informace zakódovat natvrdo pomocí námi stanovených vah, MLP své váhy přizpůsobuje automaticky pomocí algoritmu zvaného *back-propagation* datům, které mu předložíme k naučení.

Obrázek č. 4 ukazuje, jak by se problém, řešený na obrázku č. 1 pomocí sítě IAC, řešil pomocí MLP. Zde zobrazená síť je schopna se naučit určovat u členů příslušnost do gangu Jets nebo Sharks podle jejich ostatních charakteristik. Učení spočívá v postupném přizpůsobování zpočátku náhodných hodnot vah na spojích mezi neurony tak, aby transformace vstupu do sítě na výstup ze sítě dávala správné výsledky.

Obrázek č. 4: MLP pro klasifikaci členů podle jejich příslušnosti ke gangům



Pramen: [1]

Tabulka č. 2 obsahuje data, podle kterých je MLP na obrázku č. 4 schopen se učit.

Tabulka č. 2: Informace o členech gangů převedené do dat pro MLP

Jméno	**Přibližný věk**			**Vzdělání**			**Rodinný stav**			**Způsob obživy**			**Gang**	
	20	30	40	ZŠ	SŠ	VŠ	svo-bodný	ženatý	rozve-dený	překupník drog	lupič	book-maker	Jets	Sharks
Art	0	0	1	1	0	0	1	0	0	1	0	0	1	0
Rick	0	1	0	0	1	0	0	0	1	0	1	0	0	1
Sam	1	0	0	0	0	1	1	0	0	0	0	1	1	0
Ralph	0	1	0	1	0	0	1	0	0	1	0	0	1	0
Lance	1	0	0	1	0	0	0	1	0	0	1	0	1	0

Pramen: [1]

Po fázi učení neboli trénování lze síť použít jako klasifikátor přiřazující vstupní vzory do kategorií.

Trénování sítě probíhá tak, že vytvoříme soubor trénovacích dat, a síť tento soubor několikrát po sobě zpracovává. **Trénovací soubor** se skládá z dvojic vstupních a výstupních vzorů. Výstupní vzor může být považován za kategorii, do které má být správně zařazen vstupní vzor. Síť přijme na vstupní vrstvě vstupní vzor, ten sítí projde přes skryté vrstvy na vrstvu výstupní, na které se objeví jako výstupní vzor. Tento výstup je porovnán se správným výstupním vzorem načteným z trénovacího souboru. Informace o odchylce je sítí šířena směrem od výstupní vrstvy k vrstvě vstupní a jsou podle ní upravovány váhy sítě tak, aby až síť bude stejnou dvojici vzorů číst příště, byla odchylka na její výstupní vrstvě menší.

Po té, co se síť správně naučí trénovací soubor, může být testována na jiném souboru dvojic vstupních a výstupních vzorů, abychom zjistili, jak dobře se síť naučila zobecňovat. U některých typů dat není výhodné aby se síť naučila trénovací soubor co nejpřesněji, protože potom si pamatuje trénovací data příliš specificky a to jí brání v zobecňování neznámých dat. V takových úlohách se obvykle při trénování sítě sleduje i její **výkon**, měřený například sumou čtverců odchylek na výstupní vrstvě, pro **testovací soubor** a trénink se ukončí v okamžiku, kdy po období poklesu odchylek nastává jejich růst, viz obrázek č. 15.

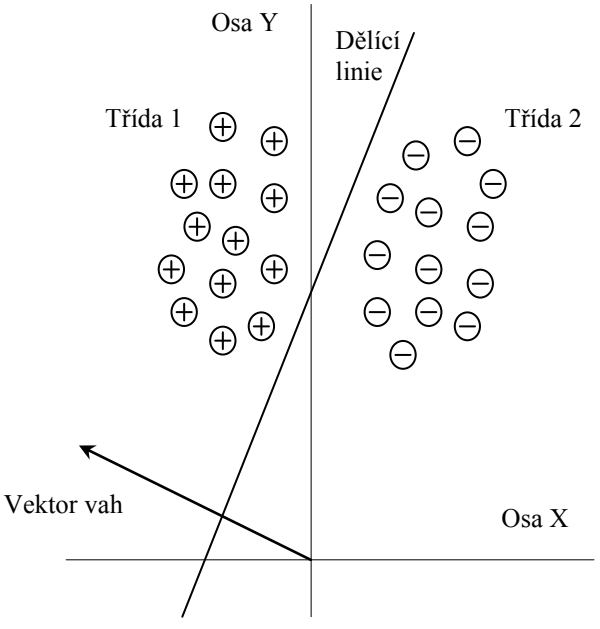
Aby bylo možné porovnávat **výkon** sítí s různým počtem výstupních jednotek zpracovávajících různé úlohy, je zvykem sumu čtverců odchylek na výstupní vrstvě převádět na odmocninu z průměrného čtverce odchylky připadajícího na jeden výstupní neuron. V angličtině se tato míra nazývá *Root Mean Square error* (*RMS error*), viz například strana 464 z knihy [5].

Ve zbytku této kapitoly bude problematika MLP vysvětlena podrobněji.

Jednou z nejstarších učících se sítí je **perceptron** amerického vědce Franka Rosenblatta z roku 1957 publikovaný v práci [4]. Úkolem perceptronu bylo naučit se správně rozdělovat binární, tj. složené z řady nul a jedniček, vektory do dvou kategorií. Perceptron se skládá z vrstvy vstupních jednotek a z jediné výstupní jednotky. Výstup perceptronu je počítán porovnáním vstupu do výstupní jednotky rovného součtu součinů vstupů ze vstupních jednotek a vah na spojích vedoucích z nich do výstupní jednotky s prahovou hodnotou θ. Vektory se kategorizují podle toho, zda je jejich skalární součin s vektorem vah větší nebo menší než práh θ. Učení perceptronu spočívá v hledání takových vah pro jeho spoje, aby byly vektory kategorizovány správně. Aby mohlo být toto učení zautomatizováno, musí být definována funkce kvantifikující chybu odpovědi perceptronu na určitý vektor a procedura, která podle vypočtené chyby upraví váhy tak, aby při příštím čtení stejného vektoru byla chyba menší. Procedura vhodná pro perceptron je velmi jednoduchá. Když je vektor zařazen správně, potom chyba se rovná 0 a s vahami se nic nedělá. Pokud je odpovědí perceptronu na vektor 0 a správná odpověď má být 1, potom se k váhám tento vstupní vektor přičte. Pokud je odpovědí perceptronu na vektor 1 a správná odpověď má být 0, potom se od váhového vektoru vstupní vektor odečte [5]. Důležitou vlastností perceptronu je, že řešení vždycky najde, pokud existuje.

K pochopení rozdílu mezi úlohami, které řešení pomocí perceptronu mají a které ne, je potřeba vysvětlit problém takzvané **lineární separability**. Dvě skupiny bodů v prostoru jsou lineárně separabilní, pokud jdou od sebe oddělit rovinou, případně přímkou, pokud jsou v dvourozměrném prostoru. Představme si vektor vycházející z počátku souřadnic, který je kolmý na takovou dělící rovinu. Dále si představme kolmé průměty bodů na tento vektor. Může nám přitom pomoci obrázek č. 5. Pokud lze mezi 2 skupiny bodů položit přímku, obecně rovinu o určitém rozměru, potom průměty bodů na vektor vah, který je na ni kolmý, tvoří nepřekrývající se skupiny a skalární součiny tohoto vektoru s kategorizovanými vstupními vektory by patřily pro obě skupiny do nepřekrývajících se intervalů a tudíž by se snadno mohlo najít číslo analogické prahu θ, se kterým by se skalární součiny porovnaly a výsledek by byl kladný pro jednu skupinu a záporný pro druhou. Perceptron tedy dokáže najít řešení, pokud lze mezi dvě kategorie n-rozměrných vektorů položit $n - 1$ rozměrnou rovinu.

Obrázek č. 5: Lineární klasifikátor



Pramen: [6]

Nejznámějším příkladem lineárně neseparabilního problému je takzvaný XOR problém. Existence lineárně neseparabilních úloh byla důvodem kritiky perceptronu v práci [13] a dočasného poklesu zájmu o neuronové sítě, který po jejím vydání následoval. XOR je jedna z logických funkcí, její tabulka je

1. vstup	**2. vstup**	**Výstup**
0	0	0
0	1	1
1	0	1
1	1	0

Budou-li hodnoty sloupků „1. vstup“ a „2. vstup“ souřadnicemi bodů v rovině, uvidíme, že mezi body s výstupem rovným 0 a výstupem rovným 1 nelze vložit rovnou čáru.

Abychom mohli vyřešit problém XOR pomocí perceptronu, musíme do funkce XOR přidat další sloupek, například funkci AND:

1. vstup	**2. vstup**	**AND**	**XOR**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Jiným způsobem, jak vyřešit pomocí perceptronů lineárně neseparabilní problém, je přidání další vrstvy perceptronů do sítě mezi vstupní a výstupní jednotky. Máme-li v této **skryté vrstvě** dostatečný počet jednotek, dokážeme libovolně složitý problém převést na lineárně separabilní a tak jej vyřešit. Jednotky ve skryté vrstvě v průběhu učení sítě začnou mít lineárně separovatelné hodnoty. Tyto hodnoty jsou pro síť **vnitřní reprezentací vzorů**. Dosud uvedené pravidlo pro učení perceptronu nám však nepomůže najít správné váhy pro spoje vedoucí od vstupní do skryté vrstvy, aby se mohla vytvořit lineárně separabilní reprezentace vzorů na skryté vrstvě, protože nemáme přímou informaci o chybě vznikající na skryté vrstvě.

Nejznámějším algoritmem, který dokáže vyřešit tento problém je *back-propagation*. Ten byl poprvé objeven Paulem Werbosem v roce 1974, ale začal se užívat až po jeho znovuobjevení Rumelhartem, McClellandem a Williamsem v roce 1986, viz časopis [7]. Řešíme-li tedy lineárně neseparabilní problém pomocí perceptronů, můžeme postupovat tak, že vytvoříme třívrstvou síť, jejímiž jednotkami neboli neurony ve skryté a výstupní vrstvě budou perceptrony (odtud název multilayer perceptron) a upravujeme její váhy pomocí algoritmu *back-propagation*. Aby tento algoritmus dobře fungoval, je nezbytné změnit aktivační funkci v perceptronech ve skryté a výstupní vrstvě. Aktivační funkce v původním perceptronu byla takzvaně **schodovitá**, což znamená, že vracela číslo 0, když vážený vstup do perceptronu byl nižší než práh θ, a v opačném případě vracela číslo 1. Nová aktivační funkce je **sigmoidální**, což znamená, že vypadá jako zaoblený schod. Takováto funkce vrací čísla v intervalu hodnot od 0 do 1. Argumentem této funkce je skalární součin výstupů z předchozí vrstvy a vektoru vah mínus prahová hodnota patřící danému neuronu. Protože i prahová hodnota v neuronech se v průběhu učení mění podobně jako váhy, lze ji realizovat jako zvláštní jednotku z předchozí vrstvy s konstantním výstupem rovným –1, který je vážen měnitelnou vahou. Potom je vstupem do aktivační funkce pouze jeden skalární součin.

Odvození algoritmu *back-propagation* uvádí mnoho publikací, je k dispozici i na internetové stránce [1], v této zprávě uvádím svou interpretaci odvozování podle literatury [5].

Název *back-propagation* by se dal česky přeložit jako **učení neuronové sítě zpětným šířením signálu**. Tento algoritmus je založen na stoupání ve směru nejprudšího svahu funkce vah. Tento směr se nazývá gradient, je to tedy stoupání po gradientu, v angličtině „*gradient ascent*“.

Při odvozování pravidel učení vícevrstvé neuronové sítě budeme používat řetězové pravidlo „*chain rule*“, určující derivaci složené funkce, a parciální derivaci.

Složená funkce může vypadat například tak, že f je funkcí g a tato funkce g je zároveň funkcí proměnné x. Odvození derivace funkce f podle proměnné x ukazuje vzorec (1).

$$\begin{aligned}
&\frac{df[g(x)]}{dx} = \\
&= \lim_{h\to 0} \frac{1}{h}[f[g(x+h)] - f[g(x)]] = \\
&= \lim_{h\to 0} \frac{1}{h}[f[g(x) + h\cdot g'(x)] - f[g(x)]] = \\
&= \lim_{h\to 0} \frac{1}{h}[f[g(x)] + h\cdot g'(x)\cdot f'[g(x)] - f[g(x)]] = \\
&= g'(x)\cdot f'[g(x)] = \frac{dg}{dx}\cdot\frac{df}{dg}
\end{aligned} \tag{1}$$

Pokud bude nějaká z funkcí funkce více proměnných, nahradí se symbol *d* symbolem ∂ značícím parciální derivaci.

Odvození parciální derivace složené funkce *f* podle proměnné *x*, kde *f* je funkcí několika funkcí g_i, z nichž každá je funkcí stejné proměnné *x*, ukazuje vzorec (2).

$$\begin{aligned}
&\frac{df[g_1(x), g_2(x), \ldots, g_n(x)]}{dx} = \\
&= \lim_{h\to 0} \frac{1}{h}[f[g_1(x+h), g_2(x+h), \ldots, g_n(x+h),] - f[g_1(x), g_2(x), \ldots, g_n(x)]] = \\
&= \lim_{h\to 0} \frac{1}{h}[f[g_1(x) + h\cdot g_1'(x), g_2(x) + h\cdot g_2'(x), \ldots, g_n(x) + h\cdot g_n'(x)] - f[g_1(x), g_2(x), \ldots, g_n(x)]] = \\
&= \lim_{h\to 0} \frac{1}{h}\left[f[g_1(x), g_2(x), \ldots, g_n(x)] + h\cdot\sum_{i=1}^{n} g_i'(x)\cdot\frac{\partial f}{\partial g_i(x)} - f[g_1(x), g_2(x), \ldots, g_n(x)]\right] = \\
&= g_1'(x)\cdot\frac{\partial f}{\partial g_1(x)} + g_2'(x)\cdot\frac{\partial f}{\partial g_2(x)} + \ldots + g_n'(x)\cdot\frac{\partial f}{\partial g_n(x)} = \sum_{i=1}^{n}\frac{dg_i}{dx}\cdot\frac{\partial f}{\partial g_i}
\end{aligned} \tag{2}$$

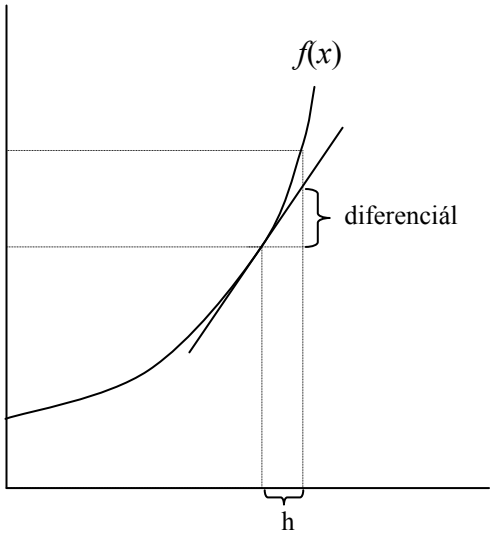
Při odvozování vzorců (1) a (2) byl použit diferenciál znázorněný obrázkem č. 6.

Obrázek č. 6: Diferenciál $h \cdot f'(x)$ umožňující aproximovat funkci $f(x+h)$ pomocí funkce $f(x)$ a její derivace v okolí bodu x

$$f(x+h) \approx f(x) + h\cdot f'(x)$$
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

když

$$h \to 0$$



Pramen: Vlastní výzkum

Tolik tedy nezbytná matematická teorie na úvod a vrátíme se k neuronové síti. Úkolem je najít takové váhy neuronové sítě, aby byl maximalizován její výkon *P* (*performance*).

Výkon neuronové sítě se měří vzorcem (3).

$$P = -\sum_{s}\sum_{z}\left(d_{sz} - o_{sz}\right)^2 \tag{3}$$

P - výkon neuronové sítě (*performance*)
s - počet trénovacích vzorů (*samples*)
z - počet výstupních uzlů neuronové sítě
d - správný výstup sítě (*desired output*)
o - skutečný výstup sítě (*output*)

Výkon je také funkcí všech vah neuronové sítě, viz vzorec (4). Správné váhy budeme hledat pomocí stoupání po gradientu (směru nejvyššího stoupání) funkce *P* všech vah.

$$P = f\left(w_i\right) \tag{4}$$

Budeme postupně měnit hodnoty vah w_i tak, aby hodnota *P* rostla co nejrychleji. Potom změna každého w_i musí být proporciální parciální derivaci *P* podle daného w_i. Tedy

$$\Delta w_i \approx \frac{\partial P}{\partial w_i} \tag{5}$$

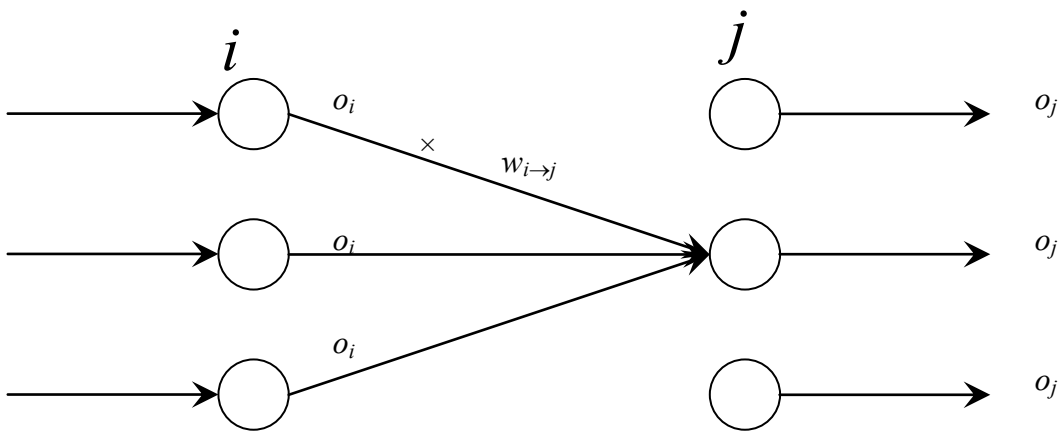
Takováto změna w_i je stoupáním po gradientu funkce *P*.

Hodnotu funkce *P* budeme měřit za každý vzor zvlášť a potom změny w_i sečteme za všechny vzory v trénovacím souboru. Výkon sítě pro jeden vzor ukazuje vzorec (6).

$$P = -\sum_{z}\left(d_{sz} - o_{sz}\right)^2 \tag{6}$$

Naše neuronová síť má vrstvy označené popořadě *i*, *j*, *k* a výstupní vrstvu *z*.

Obrázek č. 7: Účinek $w_{i\to j}$ na *P* je zprostředkován výstupem o_j na *j*-té vrstvě.



Pramen: Vlastní výzkum

Vzorec (7) vyjadřuje myšlenku zobrazenou obrázkem č. 7.

$$P = f\left[o_j\left(w_{i\to j}\right)\right] \tag{7}$$

Využijeme „*chain rule*“ (1), abychom vypočetli parciální derivaci výkonu sítě P podle váhy $w_{i\to j}$, což znamená váhu mířící ze vstupní vrstvy i do následující vrstvy j a dostaneme vzorec (8).

$$\frac{\partial P}{\partial w_{i\to j}} = \frac{\partial P}{\partial o_j} \cdot \frac{\partial o_j}{\partial w_{i\to j}} \tag{8}$$

Neuron v j-té vrstvě počítá vážený součet σ_j svých vstupů z neuronů i-té vrstvy.
Jak ukazuje vzorec (9), σ_j je funkcí vah.

$$\sigma_j = \sum_i o_i \cdot w_{i\to j} = f\left(w_{i\to j}\right) \tag{9}$$

Výstup neuronu v j-té vrstvě o_j se rovná

$$o_j = t\left(\sum_i o_i \cdot w_{i\to j}\right) = t\left(\sigma_j\right) = t\left[\sigma_j\left(w_{i\to j}\right)\right], \tag{10}$$

kde t (*threshold*) je prahová funkce.

Konec vzorce (8) je parciální derivací výstupu neuronu z j-té vrstvy podle váhy mezi i-tou a j-tou vrstvou. Můžeme jej rozepsat podle vzorce (10) s pomocí „*chain rule*“ (1) a výsledkem je vzorec (11).

$$\frac{\partial o_j}{\partial w_{i\to j}} = \frac{do_j}{d\sigma_j} \cdot \frac{\partial \sigma_j}{\partial w_{i\to j}} \tag{11}$$

Parciální derivace váženého součtu vstupů podle váhy ze vzorce (11) je rovna vstupu, který se touto vahou násobí.

$$\frac{\partial \sigma_j}{\partial w_{i\to j}} = o_i \tag{12}$$

Nyní zkombinujeme vzorce (11) a (12)

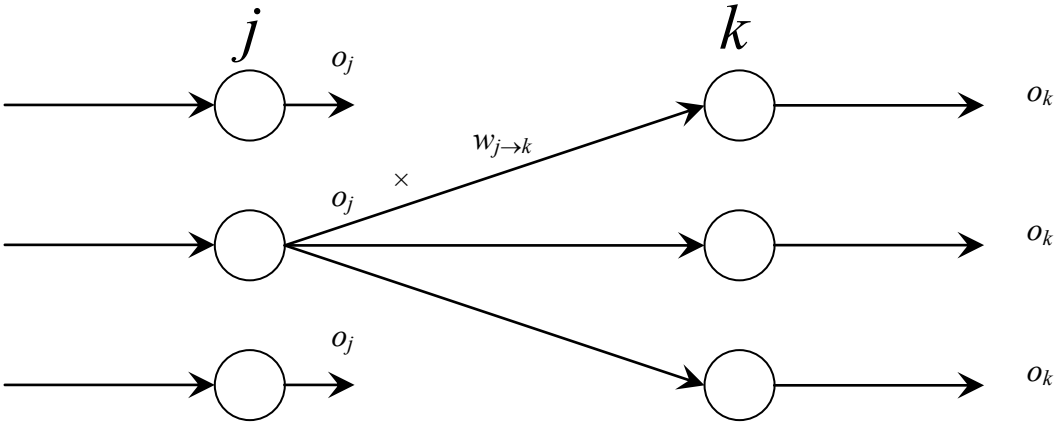
$$\frac{\partial o_j}{\partial w_{i\to j}} = \frac{do_j}{d\sigma_j} \cdot o_i \tag{13}$$

a výsledek dosadíme do vztahu (8).

$$\frac{\partial P}{\partial w_{i\to j}} = \frac{\partial P}{\partial o_j} \cdot \frac{do_j}{d\sigma_j} \cdot o_i \tag{14}$$

Dále budeme odvozovat části pravé strany rovnice (14).

Obrázek č. 8: Účinek jednoho z výstupů o_j na P je zprostředkován výstupy o_k neuronů v k-té vrstvě.



Pramen: Vlastní výzkum

Vzorec (15) vyjadřuje myšlenku zobrazenou obrázkem č. 8. Výkon sítě P je funkcí mnoha výstupů z k-té vrstvy o_k, z nichž každý je funkcí stejného výstupu z j-té vrstvy o_j.

$$P = f\left[o_k\left(o_j\right)\right] \tag{15}$$

Využijeme „*chain rule*“ (2), abychom vypočetli parciální derivaci výkonu sítě P podle výstupu z j-té vrstvy o_j a dostaneme vzorec (16).

$$\frac{\partial P}{\partial o_j} = \sum_k \frac{\partial P}{\partial o_k} \cdot \frac{\partial o_k}{\partial o_j} \tag{16}$$

Neuron v k-té vrstvě počítá vážený součet σ_k svých vstupů z neuronů j-té vrstvy.
Jak ukazuje vzorec (17), σ_k je funkcí vah.

$$\sigma_k = \sum_j o_j \cdot w_{j\to k} = f\left(w_{j\to k}\right) \tag{17}$$

Výstup neuronu v k-té vrstvě o_k se rovná

$$o_k = t\left(\sum_j o_j \cdot w_{j\to k}\right) = t\left(\sigma_k\right) = t\left[\sigma_k\left(w_{j\to k}\right)\right], \tag{18}$$

kde t (*threshold*) je prahová funkce.

Ze vzorce (17) vypočteme parciální derivaci σ_k podle výstupu z j-té vrstvy o_j, který je jejím vstupem. Parciální derivace váženého součtu vstupů podle jednoho z těchto vstupů je rovna váze vstupu, která se tímto vstupem násobí.

$$\frac{\partial \sigma_k}{\partial o_j} = w_{j\to k} \tag{19}$$

Součet vstupů na k-té vrstvě σ_k je nejen funkcí vah $w_{j\to k}$, viz vzorec (17), ale i funkcí výstupů z j-té vrstvy o_j, viz pravá strana vzorce (20) pro výstup neuronu z k-té vrstvy.

$$o_k = t\left[\sigma_k\left(o_j\right)\right] \tag{20}$$

Parciální derivace (21) výstupu neuronu z *k*-té vrstvy ze vzorce (20) podle výstupu neuronu z *j*-té vrstvy využívá „*chain rule*“ (1) a vztah (19).

$$\frac{\partial o_k}{\partial o_j} = \frac{do_k}{d\sigma_k} \cdot \frac{\partial \sigma_k}{\partial o_j} = \frac{do_k}{d\sigma_k} \cdot w_{j\to k} \tag{21}$$

Vzorec (21) dosadíme do parciální derivace výkonu sítě podle výstupu na *j*-té vrstvě z rovnice (16).

$$\frac{\partial P}{\partial o_j} = \sum_k \frac{\partial P}{\partial o_k} \cdot \frac{do_k}{d\sigma_k} \cdot w_{j\to k} \tag{22}$$

Tím jsme došli k závěru, že parciální derivace výkonu sítě *P* podle výstupu na určité vrstvě je funkcí parciální derivace *P* podle výstupu na následující vrstvě. Odtud pochází termín „*back propagation*“ jako šíření informace o opravách vah sítě směrem od jejího konce k počátku.

Jako první musíme spočítat parciální derivaci *P* podle výstupu o_z na výstupní vrstvě, viz vzorec (6).

$$\frac{\partial P}{\partial o_z} = \frac{\partial\left[-\left(d_z - o_z\right)^2\right]}{\partial o_z} = 2\left(d_z - o_z\right) \tag{23}$$

Zvolíme vhodnou prahovou funkci *t* převádějící vážený součet σ vstupů do uzlu sítě na výstup *o* v rozsahu od 0 do 1.

$$t(\sigma) = \frac{1}{1+e^{-\sigma}} = o \tag{24}$$

Odvodíme derivaci prahové funkce *t* podle σ.

$$\begin{aligned} \frac{dt(\sigma)}{d\sigma} &= \frac{d}{1+e^{-\sigma}} \cdot \frac{1}{d\sigma} = \frac{-1}{\left(1+e^{-\sigma}\right)^2} \cdot (-1) \cdot e^{-\sigma} = \\ &= \frac{1}{1+e^{-\sigma}} \cdot \left[1 - \frac{1}{1+e^{-\sigma}}\right] = t(\sigma) \cdot \left[1 - t(\sigma)\right] = o(1-o) \end{aligned} \tag{25}$$

Vidíme, že díky výhodnému tvaru prahové funkce *t* se dá tato derivace vyjádřit pomocí výstupu *o*.

Do vzorce (22) zavedeme substituci

$$\frac{\partial P}{\partial o_j} = \beta_j \tag{26}$$

Do vzorce (14) zavedeme substituci

$$\frac{\partial P}{\partial w_{i\to j}} = \Delta w_{i\to j} \tag{27}$$

A konečně můžeme zapsat všechny vzorce pro učení sítě.

Podle vztahu (23) a substituce (26) sestavíme míru, s jakou by se měl měnit výstup z výstupní vrstvy sítě.

$$\beta_z = d_z - o_z \tag{28}$$

Podle vztahů (22), (25) a substituce (26) sestavíme míru, s jakou by se měl měnit výstup z ostatních vrstev sítě.

$$\beta_j = \sum_k \beta_k \cdot o_k \left(1 - o_k\right) \cdot w_{j \to k} \tag{29}$$

Podle vztahů (14), (25) a substituce (27) sestavíme míru, s jakou by se měly měnit jednotlivé váhy sítě.

$$\Delta w_{i \to j} = \beta_j \cdot o_j \left(1 - o_j\right) \cdot o_i \cdot r \tag{30}$$

Konstanta r ve vzorci (30) je **parametr rychlosti učení**, který do sebe absorboval konstantu 2 vynechanou ve vzorci (28).

Pro trénování neuronové sítě platí následující postup, který se opakuje, dokud síť nedosáhne žádoucího výkonu:

Za každý vzor

1. Vypočteme nejprve výstup sítě o_z.
2. Vypočteme β_z na výstupní vrstvě.
3. Vypočteme ostatní hodnoty β podle hodnot β na následujících vrstvách.
4. Vypočteme všechny hodnoty Δw.

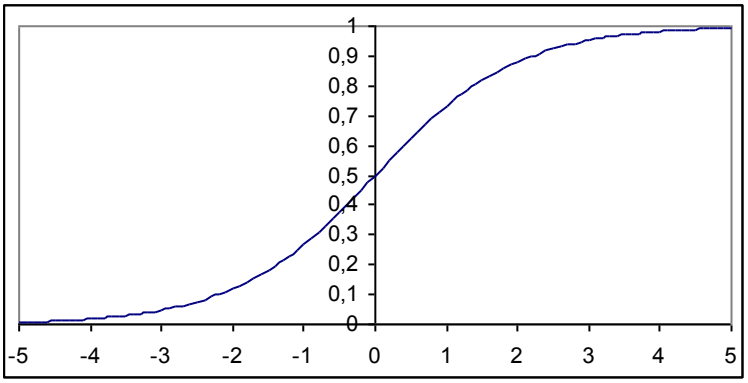
Sečteme hodnoty Δw za všechny vzory.
Přičteme součty Δw k původním hodnotám vah w.

Rychlost konvergence vah ke stavu s minimální sumou čtverců odchylek lze zrychlit zavedením různých zdokonalení.

Jak určit počáteční váhy

Kdyby počáteční váhy byly všechny stejné, učení by se tím značně zpomalilo, protože chyba zpětně šířená sítí je úměrná vahám a tak by se váhy upravovaly o stejný přírůstek a zůstávaly by stejně velké, kdežto správně naučená síť má různé váhy. Správným řešením tedy je dát počátečním vahám náhodné hodnoty. Další otázkou je, v jak velkém intervalu mají být počáteční váhy inicializovány. Správnou odpověď pomůže najít úvaha o aktivační neboli prahové funkci neuronů. Jak bylo napsáno výše, aktivační funkce je nejčastěji sigmoidální. Průběh aktivační funkce (24) ukazuje obrázek č. 9.

Obrázek č. 9: Sigmoidální aktivační funkce neuronu



Pramen: Vlastní výzkum

Učení neuronové sítě probíhá rychle, pokud vstupy do aktivačních funkcí neuronů jsou takové, aby výstupem aktivačních funkcí byla hodnota okolo 0,5. V této hodnotě má totiž maximální hodnotu derivace aktivační funkce, které je úměrná změna vah při učení. Nejběžnější je inicializovat váhy náhodnými hodnotami v intervalu od +0,5 do –0,5. Populárním algoritmem pro inicializaci vah je *Nguyen-Widrow initialization* popsaný například v literatuře [8].

Lokální minima

Algoritmus *back-propagation* klouže po povrchu sumy čtverců chyb ve směru nejprudšího srázu. U dvouvrstvých sítí (bez skryté vrstvy) má tento povrch tvar poháru a algoritmus proto konverguje bez problémů až do nejnižšího bodu zvaného globální minimum. Přidáním skryté vrstvy do sítě se většinou změní tvar povrchu

sumy čtverců chyb tak, že kromě globálního začne mít i mnohá lokální minima. Algoritmus do nich může sklouznout a zůstat v nich. Obecně platí, že čím více je neuronů ve skryté vrstvě, tím menší je pravděpodobnost uvíznutí v lokálním minimu, protože, i když je potom povrch ze sumy čtverců chyb složitější, přidání dalších rozměrů rozmnoží únikové cesty z lokálních minim. Pokud učení uvízlo v lokálním minimu, můžeme se z něho dostat také zavedením drobných poruch do vah (*white noise*) a pokračováním v učení.

Rychlost učení a momentum

Podle algoritmu *back-propagation* jsou změny vah úměrné derivaci chyby. Čím je vyšší parametr rychlosti učení *r*, tím se zvyšuje i rychlost učení, ale při příliš vysokém parametru *r*, může dojít k vykročení špatným směrem, protože svah, podle něhož se směr vykročení řídil, byl příliš krátký. Při učení potom dochází k oscilacím výkonu sítě. Parametr rychlosti učení by tedy měl být největší možný před tím, než začne docházet k oscilacím.

Učení sítě se většinou zrychlí, použijeme-li parametr **momentum**. Váhy se potom upravují podle vzorce

$w(t) = \mathrm{d}w(t) + momentum \cdot \mathrm{d}w(t-1)$,

kde *t* je pořadí učícího cyklu.

Momentum nutí váhy pohybovat se ve váhovém prostoru ve směru, kterým se vydaly v minulosti. *Momentum*, vysvětlované například v literatuře [8], využívá následující myšlenku. Hledané řešení jako bod v tolika rozměrném prostoru, kolik má síť vah, plus jeden rozměr pro hodnotu sumy čtverců chyb, se koulí po povrchu roviny, přiřazující každé kombinaci vah sumu čtverců chyb. Rychlost pohybu je úměrná prudkosti svahu, na kterém se právě bod nachází. Když se tento bod dostane do míst, která jsou plochá, jeho pohyb se velmi zpomalí, a pokud se dostane do místa s lokálním minimem, může tam uvíznout. *Momentum* je analogií z fyziky, kdy chceme, aby náš bod měl i setrvačnost získanou na svahu vedoucím ze stavu vah na začátku učení, která jej bude pohánět na plochých místech a případně i způsobí jeho vyklouznutí z oblastí lokálních minim. *Back-propagation* se v tomto případě liší od své standardní podoby v tom, že každá váha se upravuje také o zlomek hodnoty, o kterou se upravovala v předchozím kroku učení.

Aktualizace vah

Váhy mohou být upravovány buďto po zpracování každého vzoru, nebo po zpracování dávky vzorů. Rozdíl mezi rychlostí těchto dvou metod se zvyšuje s růstem parametru rychlosti učení, protože algoritmus *back-propagation* je odvozen za předpokladu, že váhy se aktualizují po zpracování všech vzorů v trénovacím souboru.

Vývoj vnitřních reprezentací vzorů

Učení třívrstvé neuronové sítě probíhá ve dvou fázích. V první fázi síť vytváří na neuronech ve skryté vrstvě lineárně separovatelnou reprezentaci vzorů a při tom se její výkon měřený poklesem sumy čtverců chyb zvyšuje velmi pomalu. Ve druhé fázi výkon stoupá rychle, protože síť řeší lineárně separovatelný problém mezi skrytou a výstupní vrstvou. Správné řešení, tj. nalezení globálního minima sumy čtverců chyb, není jedinou možnou kombinací vah a záleží na počátečních hodnotách vah.

1.5. Fonetická transkripce češtiny [14]

Fonetická transkripce je určena k přesnému a nedvojznačnému zápisu zvuků mluvené řeči. Fonetická transkripce se využívá při rozpoznání řeči počítačem a při hlasové syntéze z psaného textu. V obou těchto případech je nutné zabezpečit automatický přepis libovolného psaného textu na odpovídající řetězec fonémů. Protože si nelze zapamatovat všechny tvary výslovnosti pro každou promluvu, je nutné hledat obecná pravidla, podle nichž by bylo možné fonetický přepis automaticky vytvářet. Tato obecná pravidla mohou být formulována jako produkční pravidla a nazývají se **fonologická pravidla**. Protože většina fonémových změn může být vysvětlena levým a pravým kontextem daného fonému, lze definovat obecné produkční pravidlo ve tvaru

JESTLIŽE řetězci znaků A bezprostředně předchází řetězec znaků C a je bezprostředně následován řetězcem znaků D
PAK se A přepíše na řetězec znaků B.

Pro jednoduchost budeme v dalším výkladů toto pravidlo zapisovat ve tvaru

$A \rightarrow B / C \_ D$.

Pro zápis fonémů je třeba využít nějakou fonetickou abecedu. Mezinárodní fonetická abeceda (*International Phonetic Alphabet – IPA*) [15] se pro národní účely může nahradit abecedami lépe vystihujícími místní fonetická pravidla. V případě češtiny budu při popisu řešení problému fonetické transkripce používat fonetickou abecedu pro češtinu (*Phonetic Alphabet for Czech – PAC*) navrženou v článku [16].

Tabulka č. 3: Symboly pro české fonémy dle PAC, které jsou použity při řešení úlohy v této výzkumné zprávě

Číslo	Foném vyjádřený českými hláskami	Foném dle PAC	Příklad	Číslo	Foném vyjádřený českými hláskami	Foném dle PAC	Příklad
1	„a“	a	tát**a**	21	„m“	m	má**m**a
2	„á“	á	t**á**ta	22	„m“	M	tra**m**vaj
3	„b“	b	bá**b**a	23	„n“	n	ví**n**o
4	„c“	c	o**c**el	24	„n“	N	ba**n**ka
5	„dz“	C	le**c**kde	25	„ň“	ň	ko**ň**ě
6	„č“	č	**č**ichá	26	„o“	o	k**o**lo
7	„dž“	Č	rá**dž**a	27	„ó“	ó	**ó**da
8	„d“	d	je**d**en	28	„p“	p	pu**p**en
9	„ď“	ď	**ď**ělat	29	„r“	r	be**r**e
10	„e“	e	l**e**v	30	„ř“	ř	mo**ř**e
11	„é“	é	m**é**ně	31	„ř“	Ř	ke**ř**
12	„f“	f	**f**auna	32	„s“	s	**s**ud
13	„g“	g	**g**uma	33	„š“	š	du**š**e
14	„h“	h	a**h**a	34	„t“	t	du**t**ý
15	„ch“	X	**ch**udý	35	„ť“	ť	ku**ť**il
16	„i“ nebo „y“	i	b**i**l, b**y**l	36	„u“	u	d**u**še
17	„í“ nebo „ý“	í	v**í**tr, l**ý**ko	37	„ú“ nebo „ů“	ú	r**ů**že
18	„j“	j	do**j**at	38	„v“	v	lá**v**a
19	„k“	k	**k**upec	39	„z“	z	ko**z**a
20	„l“	l	dě**l**á	40	„ž“	ž	rů**ž**e

Pramen: [16]

Následují nejzákladnější fonologická pravidla pro český jazyk. Tato pravidla využívají zkratky vysvětlené v tabulce č. 4.

Tabulka č. 4: Dělení českých hlásek

Samohlásky (SA)	a, á, e, é, i, í, o, ó, u, ú									
Znělé párové souhlásky (ZPS)	b	d	ď	g	z	ž	v	h	dz (C)	dž (Č)
Neznělé párové souhlásky (NPS)	p	t	ť	k	s	š	f	ch (X)	c	č
Jedinečné souhlásky (znělé) (JS)	m, n, ň, l, j, r, ř									

Pramen: [14]

Písmeno „w“ se přepisuje na [v]: w → v / _ .
Písmeno „q“ se přepisuje na [kv]: q → kv / _ .
Následuje-li „ě“ po „b“, „p“, „f“, „v“, přepisuje se na [je]: ě → je / <b, p, f, v> _ .
Spojení „dě“, „tě“, „ně“ přepisujeme na [ďe], [ťe], [ňe]: <d, t, n>ě → <ď, ť, ň>e / _ .
Spojení „mě“ přepisujeme na [mňe]: ě → ňe / m _ .
Spojení „di“, „ti“, „ni“ přepisujeme na [ďi], [ťi], [ňi]: <d, t, n> → <ď, ť, ň> / _ <i, í>.
Jestliže „x“ stojí před znělou souhláskou, přepisuje se na [gz]: x → gz / _ <ZPS, JS>.
Jestliže „x“ stojí před neznělou souhláskou, či na konci slova, přepisuje se na [ks]: x → ks / _ <NPS, kon. sl.>.
Jestliže „x“ stojí na počátku slova před samohláskou, přepisuje se na [ks]: x → ks / poč. sl. _ SA.
Jestliže „x“ stojí mezi samohláskami, přepisuje se na [ks]: x → ks / SA_1 _ SA_2.
Jestliže na počátku slova je „ex“ a následuje-li samohláska, přepisuje se na [egz]: ex → egz / poč. sl _ SA.

Při spojování souhlásek dochází velmi často ke změnám, které jsou výsledkem neustálého přestavování mluvicích orgánů z jedné artikulační podoby do druhé. Nejčastější změnou je tzv. asimilace čili spodoba, která je dvojího druhu – spodoba znělostní a spodoba artikulační.

Spodoba znělosti

Spodoby znělosti se zúčastňují jen souhlásky ze skupiny souhlásek párových. Spojení takovýchto dvou souhlásek je buď celé znělé, nebo celé neznělé podle toho, je-li poslední souhláska znělá či neznělá.

Spodoba znělosti se může určitým způsobem projevit i přes hranice slov. Její účinek může nastat jen při plynulém vyslovení příslušného slovního spojení bez zřetelné pauzy mezi slovy. Platí zde základní pravidlo, že znělá souhláska ztrácí na konci slova znělost a může ji nabýt jen tehdy, když po ní následuje v počátku následujícího slova znělá souhláska párová.

Obdobné pravidlo platí i pro neznělou párovou souhlásku na konci slova, začíná-li následující slovo znělou párovou souhláskou a nebo souhláskou „ř".

Z pravidel o podobě znělosti existují tyto výjimky:

a) Vícehláskové předložky zakončené znělou párovou souhláskou (před, pod, nad, bez , ob, od) si zachovávají svou znělost, začíná-li následující slovo znělou párovou nebo jedinečnou souhláskou.
b) Předložky „z" a „v" před znělou párovou či jedinečnou souhláskou zůstávají znělé.
c) Předložka „k" se před nepárovými souhláskami a souhláskou „v" nemění v souhlásku znělou.
d) Znělá souhláska „v" se spodobuje, ale sama spodobu nezpůsobuje.
e) Spojení souhlásek „s" a „h" se spodobuje, jestliže mezi „s" a „h" je zřetelný prefixový šev (předěl mezi předponou a základem slova).

Zejména v Čechách dochází ve spojení „sh" k postupné asimilaci: h → X / s _ .

Souhláska „ř" se v postavení před souhláskou párovou řídí základním pravidlem o spodobě znělosti.

V postavení po párové souhlásce podléhá „ř" postupné asimilaci: ř → Ř / NPS _ .

Spodoba artikulační

Při spojení dvou souhlásek s rozdílnou artikulací se vyrovnává rozdíl mezi jejich výslovností artikulační spodobou. Rozeznáváme přitom asimilaci postupnou (předcházející souhláska ovlivňuje následující) a asimilaci zpětnou (následující souhláska ovlivňuje souhlásku předcházející). Nejdůležitější z těchto pravidel jsou:

a) Jestliže nazála „n" stojí před okluzívami „k" nebo „g", spodobuje se v [N]: n → N / _ <k, g>.
b) Jestliže nazála „m" stojí před frikativami „v" nebo „f", spodobuje se v [M]: m → M / _ <v, f>.
c) Jestliže nazála „n" stojí před okluzívami „ť" nebo „ď", spodobuje se často v [ň]: n → ň / _ <ť, ď>.
d) Jestliže nazála „ň" stojí za souhláskami „d" nebo „t", dochází k jejich spodobě na [ď] nebo [ť]:
   d → ď / _ ň,
   t → ť / _ ň.
e) Připouští se zjednodušená výslovnost závěrových souhlásek „t", „d" ve spojení s úžinovými „s", „z", „š", „ž".
   Tato výslovnost může být realizována pomocí polozávěrových protějšků [c], [č], popřípadě [C], [Č]:
   <ts, ds> → c / _ ,
   <tš, dš> → č / _ ,
   <dz> → C / _ ,
   <dž> → Č / _ .

V případě prefixového či mezislovního švu ve spojení „ts", „tš", „ds", „dš", „dz", „dž" se dává přednost zachování výslovnosti obou souhlásek. Výslovnost obou souhlásek se zachová i na hranici předložek a jmen. Přitom ve spojení „ds", „dš" se obvykle uplatní pravidlo o spodobě znělosti.

Dvě stejné souhlásky „cc", „čč", „dd", „jj", „kk", „ll", „nn", „mm", „ss", „šš", „tt", „zz", které se nacházejí na prefixovém švu (předělu mezi předponou a základem slova) či sufixovém švu (předělu mezi základem slova a příponou), (což je naprostá většina případů), se při vyslovení převážně redukují na souhlásku jedinou. Pouze chce-li řečník zdůraznit šev, nebo se zdvojená souhláska nachází na mezislovním švu, vysloví souhlásku zdvojenou. Podle tohoto pravidla lze postupovat i v případě, kdy vedle sebe stojí dvojice hlásek buďto neznělá párová a její znělý protějšek nebo znělá párová a její neznělý protějšek. Většinou zde dochází k spodobě znělosti a vedle sebe jsou dvě stejné znělé či neznělé párové souhlásky, u kterých může, ale nemusí dojít ke splynutí v hlásku jedinou.

Slovní přízvuk a ráz

Ráz se v češtině vytváří automaticky a pravidelně po každé delší pauze, pokud další promluva začíná samohláskou. Ráz jsem v níže popisované úloze neuvažovala. Slovní přízvuk je v mluvené češtině v zásadě vázán na první slabiku přízvukového taktu. Přízvukový takt je úsek promluvy s jedním přízvukovým vrcholem. Přízvukový takt má v češtině tyto vlastnosti:

a) Přízvučná slabika je zpravidla první slabikou přízvukového taktu.
b) V neutrální promluvě leží hranice taktů v místě hranic slov.
c) Přízvukový takt může obsahovat několik slov.
d) Hranice mezi jednotlivými přízvukovými takty je obvykle charakterizována kontrastem stupně přízvučnosti (nepřízvučná – přízvučná).

Některá dílčí upřesnění a časté výjimky:

- Původní předložky jednoslabičné, jako je „bez", „na", „do", „ke", „o", „od", „pod", „po", „přes", „u" apod., přejímají obvykle přízvuk následujícího slova a tvoří s ním jeden takt.
- Předložky nepůvodní, např. „blíž", „dle", „kol", „krom", „skrz" apod., přízvuk následujícího slova nepřijímají.
- Některá slova, obvykle jednoslabičná, přízvuk nemají a vytvářejí se slovem předcházejícím jednoslabičný takt.

- Některá slova nemají přízvuk zcela pravidelně a jejich základní podoba je nepřízvučná. Tato slova se nazývají příklonkami a jsou to například zájmena „se“, „si“, „mně“, „mi“, „ho“, „mu“, částice „-li“ apod.

Podle výše vyjmenovaných nejzákladnějších pravidel fonetického přepisu češtiny jsem sestavila tabulku č. 5.

Tabulka č. 5: Možnosti přepisu hlásek na fonémy

Číslo	Hlásky	Fonémy	Číslo	Hlásky	Fonémy	Číslo	Hlásky	Fonémy	Číslo	Hlásky	Fonémy
1	-	.	24	ě	je	47	k	g	70	t	t
2		_	25	ě	ňe	48	k	.	71	t	d
3		.	26	f	f	49	l	l	72	t	ť
4	a	a	27	g	g	50	l	.	73	t	.
5	á	á	28	g	k	51	m	m	74	ť	ť
6	b	b	29	h	h	52	m	M	75	ts	.c
7	b	p	30	h	X	53	m	.	76	tš	.č
8	c	c	31	ch	X	54	n	n	77	u	u
9	c	.	32	i	i	55	n	N	78	ú	ú
10	č	č	33	í	í	56	n	ň	79	ů	ú
11	č	.	34	ia	ija	57	n	.	80	v	v
12	d	d	35	iá	ijá	58	ň	ň	81	v	f
13	d	t	36	ie	ije	59	o	o	82	w	v
14	d	ď	37	ié	ijé	60	ó	ó	83	x	gz
15	ď	ď	38	ii	iji	61	p	p	84	x	ks
16	ď	ť	39	ií	ijí	62	q	kv	85	y	i
17	ds	.c	40	io	ijo	63	r	r	86	ý	í
18	dš	.č	41	ió	ijó	64	ř	ř	87	z	z
19	dz	.C	42	iu	iju	65	ř	Ř	88	z	s
20	dž	.Č	43	iů	ijú	66	s	s	89	z	.
21	e	e	44	j	j	67	s	z	90	ž	ž
22	é	é	45	j	.	68	š	š	91	ž	š
23	ě	e	46	k	k	69	š	.	92	ž	.

Vysvětlivky:
"." - Prázdný znak znamenající, že přepisovaná hláska se vynechá bez náhrady
" " - Mezera mezi slovy
"_" - Pauza v promluvě
"-" - Pomlčka před částicí "li"
Pramen: Vlastní výzkum

Expertní systémy pro fonetickou transkripci češtiny obvykle postupují takto:
- daný text zpracovávají znak po znaku zleva doprava;
- u každého znaku nejprve zjišťují, zda u něj nelze uplatnit nějakou výjimku; pokud ano, výjimka se prioritně uplatní;
- pokud nelze na daný text uplatnit některou z výjimek, aplikuje se vhodné základní pravidlo;
- jestliže nelze na daný znak (písmeno) aplikovat žádnou z výjimek ani žádné z pravidel, znak se jednoduše opíše do vytvářeného fonetického řetězce.

Podle tabulky č. 5 existuje přibližně 92 možností, jak přepsat řetězec psaného textu na řetězec fonémů. Na přepis textu na fonémy se dá nahlížet jako na klasifikační problém, ve kterém se znaky textu klasifikují do kategorií patřících určitým fonémům nebo řetězcům fonémů. Tam, kde je více možností, jak přepsat daný znak na foném, se rozhodujeme podle levého a pravého kontextu znaku ve slově.

V kapitole 1.4.5. jsme se seznámili s neuronovou sítí typu MLP, jejíž výstup můžeme označit jako klasifikační rozhodnutí. MLP je proto vhodný nástroj na řešení fonetické transkripce vedle tradičního expertního systému používajícího soubor pravidel a tabulku výjimek. MLP na rozdíl od tradičního expertního systému se učí soubor, ve kterém jsou (pokud možno rovnocenně) zastoupeny všechny možné příklady přepisu znaků textu na řetězce fonémů.

Poprvé použili neuronovou síť typu MLP pro fonetickou transkripci, konkrétně anglického textu, Američané Sejnowski a Rosenberg, viz články [17] a [18]. Svoji síť pojmenovali **NETtalk** (mluvící síť), protože byla napojena na řečový syntezátor, takže bylo možné sledovat postup učení sítě. Zajímavé bylo, že se při učení

síť chovala jako malé dítě, které začíná mluvit. Jako první se síť naučila rozlišovat mezi samohláskami a souhláskami, i když ještě nepoužívala správné samohlásky a souhlásky, což znělo jako žvatlání. Ve druhé fázi se síť naučila rozeznávat hranice slov, takže před tím, než dosáhla srozumitelné a z velké části správné výslovnosti, produkovala pseudoslova. Vstupem sítě byl řetězec 7 znaků textu a výstupem foném pro prostřední znak vstupního řetězce a přízvuk nebo hranice slabik. Písmena vstupního textu byla reprezentována lokálně v rámci každé ze 7 skupin jednotek ve vstupní vrstvě neuronové sítě. Vstupní vrstva se skládala ze 7 x 29 jednotek. V oněch 29 jednotkách v každé ze 7 skupin bylo 26 jednotek pro písmena anglické abecedy a 3 jednotky pro rozdělovací znaménka a hranice slov. Fonémy byly reprezentovány distribuovaně, protože každý foném byl vyjádřen dvojicí jednotek ve výstupní vrstvě. Jedna z nich patřila do skupiny 21 jednotek pro fonémy a druhá byla vybírána ze skupiny 5 jednotek pro 3 druhy přízvuků a 2 znaky pro hranice slov. Na souboru 20 012 slov síť dosáhla výkonu kolem 90 % správně určených dvojic foném a jeho přízvuk či hranice slabik pro každý znak textu. Ve své době tento pokus prokázal dobrou praktickou využitelnost neuronových sítí a dodnes je velmi populární.

O tom, jak vhodný nástroj je MLP pro fonetickou transkripci češtiny, pojednává praktická část této výzkumné zprávy.

2. Praktická část

Cílem mého výzkumu bylo zjistit, zda neuronová síť je schopna se naučit výslovnost českých slov.

Několik pokusů, které popisuji níže, svědčí to tom, že neuronová síť je schopna se naučit nejčastěji používaná pravidla české výslovnosti.

2.1. Podrobnosti o neuronové síti

Síť měla 3 vrstvy. Jednotky v sousedních vrstvách byly propojeny každá s každou. Na začátku učení jsem váhy inicializovala náhodnými hodnotami v rozmezí od –0,5 do +0,5, protože tento interval doporučuje literatura [8] na straně 297. Sejnowski a Rosenberg inicializovali váhy své sítě NETtalk v rozmezí –0,3 do +0,3. Učení probíhalo podle standardního algoritmu *back-propagation*. Kritériem pro aktualizaci vah byla minimalizace čtverců chyb na jednotkách výstupní vrstvy. Váhy byly aktualizovány po zpracování každého vzoru, tj. sekvenci 5 znaků. Učinila jsem i pokus, kdy se váhy aktualizovaly až po zpracování celého slova. Výsledkem bylo podstatné zpomalení postupu učení. Při učení jsem použila parametr rychlosti učení rovný 2, kromě 2. pokusu, kdy jsem použila parametr rovný 1. Parametr rovný 2 jsem vybrala proto, že při trénování souborů s přibližně 10 slovy dával nejlepší výsledky. To, že aktualizace vah až po zpracování slova dává tak špatné výsledky, neznamená, že je to špatná metoda, pouze pro ní není vhodný parametr rychlosti učení rovný 2. Při parametru rychlosti učení rovném 1 tato metoda dala po prvním průchodu trénovacím souborem z 1. pokusu sumu čtverců chyb 16 583 a při parametru rychlosti učení rovném 0,1 dala po prvním průchodu sumu čtverců chyb 7 098. Sejnowski a Rosenberg používali právě aktualizaci vah až po každém slově, která při mých pokusech dopadala hůře než aktualizace vah po každém vzoru.

Obrázek č. 10: Porovnání rychlosti učení sítě při různých způsobech aktualizace vah

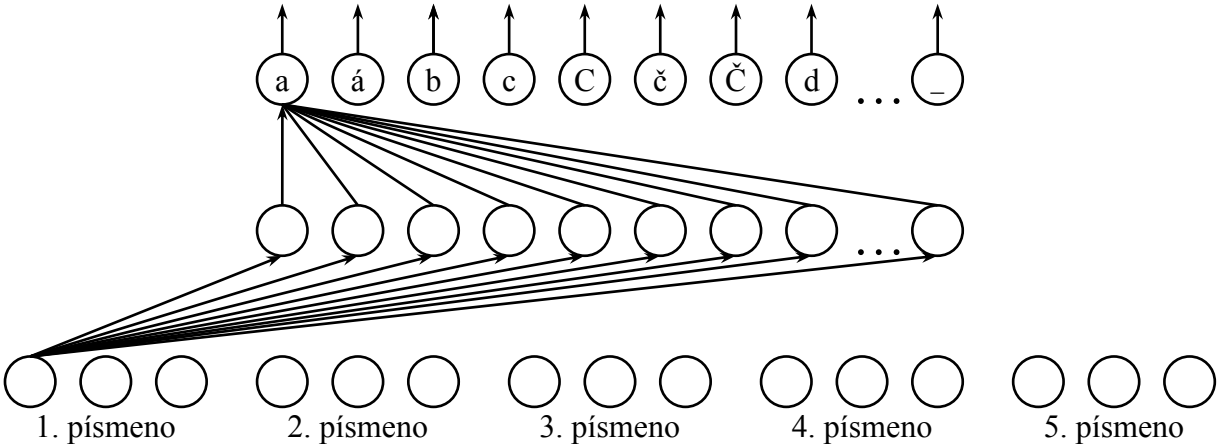
Suma čtverců chyb	
1786	35552
921	35101
571	29168
507	28475
457	28446
461	28431
433	28427
420	28426
400	28419
401	32376

Pramen: Vlastní výzkum

1. vrstva měla počet jednotek rovný počtu všech možných znaků vyskytujících se ve zpracovávaném souboru slov (44) krát 5. Písmeno *ch* mělo samostatnou jednotku. Číslo 5 znamená, že pro určení fonému patřícího určitému znaku je využit kontext 2 předchozích a 2 následujících znaků ve slově. Počet jednotek na vstupní vrstvě tedy byl 44 x 5 = 220. Na obrázku č. 11 je každých 44 jednotek pro jeden znak textu znázorněno třemi kolečky ve vstupní (zde spodní) vrstvě neuronové sítě typu MLP. Síť po přečtení každého vzoru musela upravit hodnotu 15 568 vah. Tento počet se dostane výpočtem $(220 + 1) \cdot 56 + (56 + 1) \cdot 56 = 15\,568$. Do tohoto

počtu jsou zahrnuty i váhy mířící z pomyslných jednotek s konstantním vstupem –1 reprezentujících prahovou hodnotu na neuronech ve skryté a výstupní vrstvě. Na obrázku č. 11 jsou znázorněny jen váhy mířící z jednoho vstupního neuronu a do jednoho výstupního neuronu sítě.

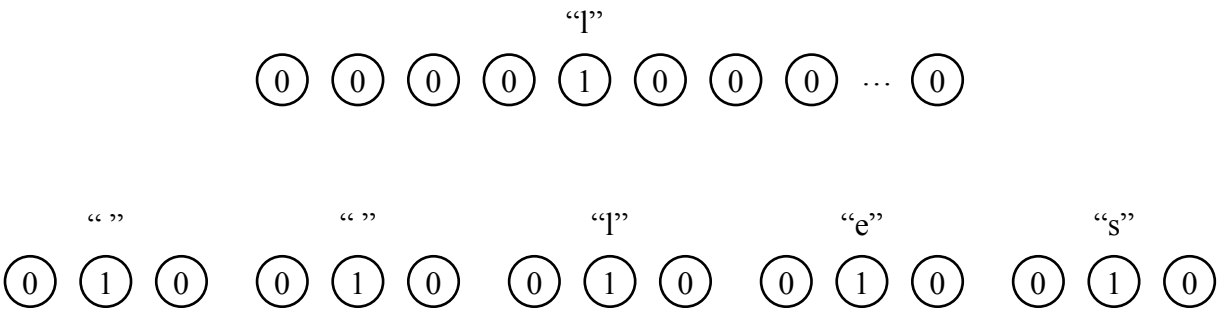
Obrázek č. 11: MLP pro fonetickou transkripci češtiny



Pramen: Vlastní výzkum

Vzor vzniká způsobem znázorněným na obrázku č. 12. Chceme například zpracovat slovo *les*. Nejprve se zpracovávané slovo obalí z každé strany dvěma mezerami. 1. vzor vzniklý ze slova *les* bude sekvencí znaků ‘ ’, ‘ ’, ‘l’, ‘e’, ‘s’. Vstupními signály na 1. vrstvě budou samé nuly až na 5 jedniček, z nichž první bude na jednotce z prvního sektoru 44 jednotek vstupní vrstvy, která patří mezeře, druhá bude na jednotce pro mezeru ve druhém sektoru, třetí bude ve třetím sektoru na jednotce patřící znaku *l*, čtvrtá bude ve čtvrtém sektoru na jednotce patřící znaku *e* a pátá bude na jednotce pro znak *s* ležící v pátém sektoru. Správnou hodnotou pro tento vzor bude foném *l* přiřazený znaku *l*, což znamená, že výstupním vzorem jsou samé nuly až na jedničku na jednotce výstupní vrstvy patřící fonému *l*. Po úpravě vah sítě se okénko 5 znaků posune a dalším vzorem bude sekvence znaků ‘ ’, ‘l’, ‘e’, ‘s’, ‘ ’ a prostřednímu znaku *e* se přiřadí foném *e*. Nakonec se bude zpracovávat sekvence ‘l’, ‘e’, ‘s’, ‘ ’, ‘ ’ a foném *s*. Takže každé slovo je síti prezentováno tolika vzory, kolik má znaků pro fonémy.

Obrázek č. 12: Reprezentace prvního vstupního a výstupního vzoru pro slovo „les“ na vstupní a výstupní vrstvě MLP

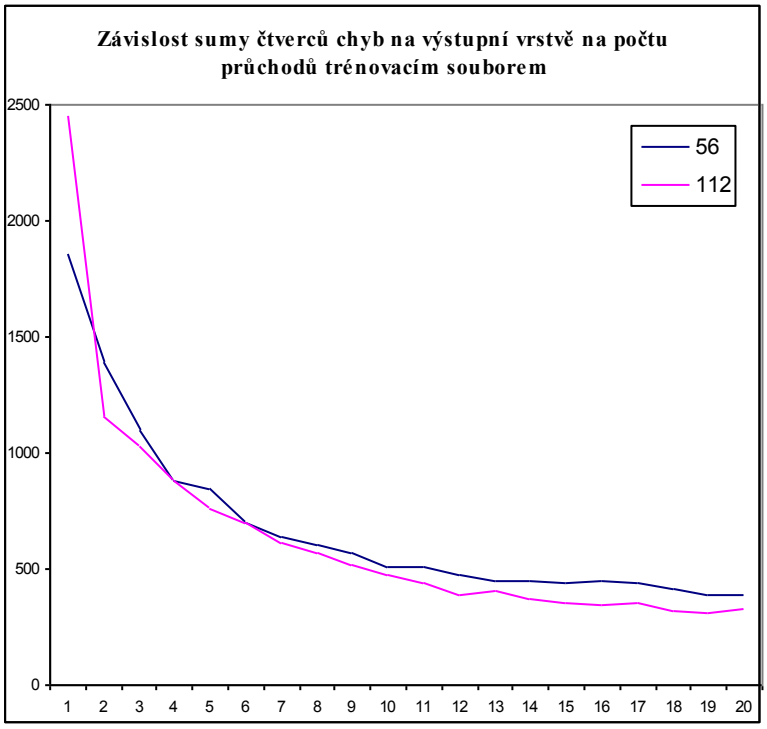


Pramen: Vlastní výzkum

2. vrstva má libovolný počet jednotek. Při trénování jsem vyzkoušela 56 a 112 jednotek s přibližně stejnou rychlostí konvergence, jak ukazuje obrázek č. 13. Pokud není v následně popsaných pokusech uvedeno jinak, skrytá vrstva použité neuronové sítě má 56 jednotek neboli neuronů.

Obrázek č. 13: Porovnání rychlosti učení neuronových sítí s různým počtem jednotek v prostřední vrstvě

Počet jednotek ve 2. vrstvě	
56	**112**
1851	2450
1390	1154
1095	1023
880	877
849	756
694	698
642	613
607	573
565	520
510	471
505	443
474	391
446	401
444	373
436	357
446	343
439	351
414	320
392	313
385	330



Pramen: Vlastní výzkum

Podle výzkumu neuronových sítí, který je přiblížen například literaturou [8] na straně 304, přibližně platí, že s růstem počtu jednotek v prostřední vrstvě klesá počet průchodů trénovací dávkou nezbytný k dosažení požadovaného výkonu sítě, od určitého počtu jednotek však již nezbytný počet průchodů klesá zanedbatelným tempem. Z toho se dá odvodit, že pro zde popisovanou síť je počet jednotek prostřední vrstvy 56 již dostatečně vysoký.

3. vrstva má tolik jednotek, kolik chceme použít různých fonémů. Při tomto způsobu zpracování slov bylo třeba, aby každému znaku ve slově byl přiřazen právě jeden foném. Případy, kdy fonémů ve slově je méně než znaků, jako například u slova *vonná* s fonetickým přepisem *voná*, jsem řešila použitím znaku pro vypuštěný foném. Případy, kdy fonémů ve slově je více než znaků, jako například u slova *město* s fonetickým přepisem *mňesto*, jsem řešila přidáním znaků pro dvojice fonémů do souboru jednotlivých fonémů podle PAC z kapitoly 1.5. Soubor všech mnou použitých znaků pro fonémy je vidět na obrázku č. 19. V 1. a 2. pokusu měla výstupní vrstva 55 jednotek, ve 3. a následujících pokusech to bylo 56 jednotek, protože jsem přidala znak pro dvojici fonémů *kv*. Při výpočtu výkonu sítě byla jednak počítána suma čtverců chyb na výstupní vrstvě, jednak počet chybně určených fonémů. Po průchodu vzoru sítí byl na výstupní vrstvě vybrán foném patřící jednotce s nejvyšší hodnotou. Na rozdíl od původní sítě NETtalk jsou v mé síti reprezentovány jak jednotlivá písmena tvořící slova tak jednotlivé fonémy lokálně, protože foném je určen výběrem pouze jediné jednotky z výstupní vrstvy. Na obrázku č. 11 jsou v kroužcích znázorňujících výstupní (zde horní) vrstvu sítě znaky pro fonémy.

2.2. Pokusy provedené na neuronové síti

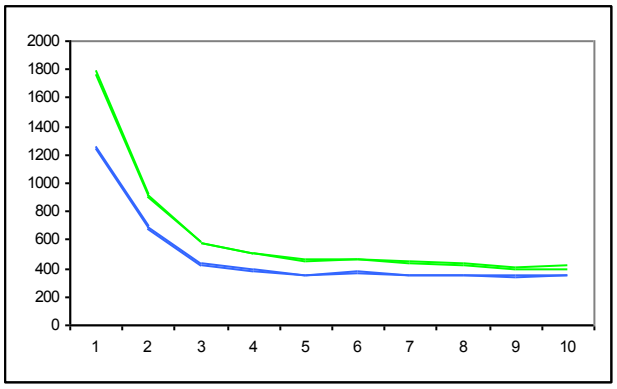
1. pokus

Síť se učila 5 167 slov (35 556 vzorů) a zároveň byl její výkon monitorován na testovacím soboru jiných 5 167 slov (35 904 vzorů), která se však v mnoha případech lišila od trénovaných slov jen tvarem. Slova byla vybrána podle četnosti jejich výskytu v denním tisku. Po 10 průchodech trénovacím souborem síť dosáhla výkonu 0,99 % chybně určených fonémů v trénovacím souboru a 0,97 % v testovacím souboru. Tato síť po vytrénování uměla správně vyslovovat většinu typicky českých slov. Tento výkon je lepší než byl výkon sítě NETtalk pro angličtinu vyčíslený na konci kapitoly 1.5. jako 90 % správně určených fonémů. To má důvod jednak v tom, že angličtina má více pravidel než čeština, a potom v tom, že Sejnowski a Rosenberg reprezentovali fonémy distribuovaně. Jejich síť vybírala 2 jednotky z výstupní vrstvy 26 neuronů. Bylo zde na výběr 21 x 5 možných dvojic foném a jeho přízvuk nebo hranice slov. Moje síť vybírala jedinou jednotku z 56.

Síť se nenaučila přiřazovat správné fonémy méně frekventovaným českým písmenům, jako jsou *ď*, *ť* a *ň*, a nenaučila se rozlišovat mezi slovy vyslovovanými podle českých fonologických pravidel a výjimkami. Například přejatému slovu *disk* síť přiřadila fonetický přepis *ďisk* namísto správného *disk* a typicky čekému slovu *divadlo* síť přiřadila fonetický přepis *divadlo* namísto správného *ďivadlo*, protože byla popletená výjimkami, kdy slabika *di* se vyslovuje jako *di* a ne jako *ďi*.

Obrázek č. 14: Porovnání průběhu trénování sítě na trénovacím a testovacím souboru při 1. pokusu

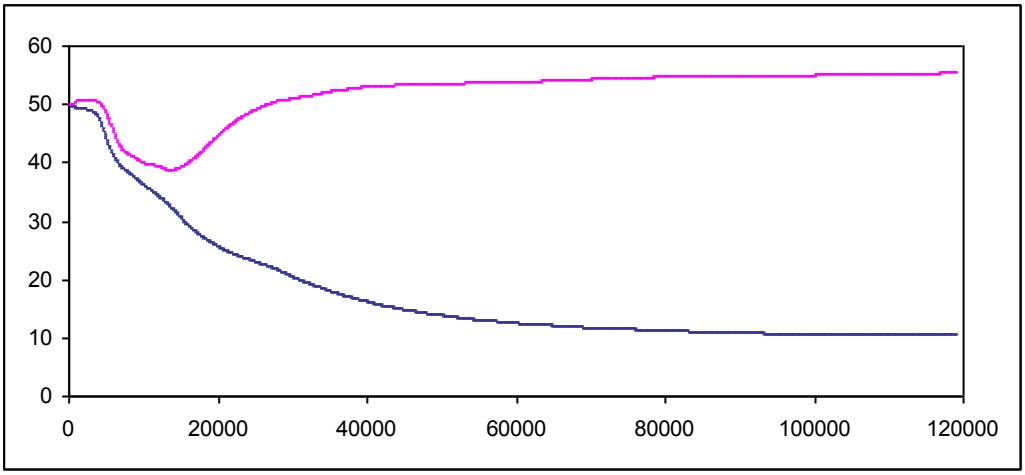
Suma čtverců chyb a počet chybných fonémů			
Trénovací soubor		Testovací soubor	
1786	1256	1759	1233
921	690	895	671
571	430	571	421
507	388	512	385
457	356	467	352
461	375	470	368
433	357	448	359
420	350	439	353
400	345	414	346
401	353	416	350



Pramen: Vlastní výzkum

Obrázek č. 14 ilustruje to, že výkon sítě na trénovacích i testovacích datech byl téměř stejný. Znamená to, že při řešení fonetické transkripce má síť velmi dobrou generalizační schopnost, neboli síť umí zobecnit to, co se naučila v trénovacím souboru na testovací slova. Veliká shoda výkonu na trénovacích a testovacích datech dále svědčí o tom, že v trénovacím a testovacím souboru jsou zastoupeny stejné druhy fonetických pravidel s přibližně stejným počtem výskytů. Při řešení jiných úloh pomocí MLP, kdy se sleduje výkon na trénovacím i testovacím souboru, bychom dostali průběh sumy čtverců chyb podobnější tomu, co znázorňuje obrázek č. 15.

Obrázek č. 15: Typický průběh výkonu sítě MLP na trénovacím a testovacím souboru



Pramen: Vlastní výzkum

Obrázek č. 15 znázorňuje závislost sumy čtverců chyb na počtu průchodů trénovacím souborem u testovacích a trénovacích dat. Daty byly v tomto případě 4 parametry, podle kterých se kreslil fraktál zvaný King's Dream Clifforda Pickovera. Síť se učila rozhodovat, zda vstupní parametry povedou k fraktálu vypadajícímu jako shluk teček nebo k tomu, že se tečky po malém počtu iterací začnou kreslit do stejného místa, případně do omezeného počtu míst nebo kroužků. V trénovacím i testovacím souboru byly parametry pro 100 nekonvergujících a 100 konvergujících fraktálů. Oba soubory byly samozřejmě disjunktní. Dolní křivka na obrázku č. 15 znázorňuje sumu čtverců chyb na trénovacím souboru. Horní křivka znázorňuje sumu čtverců chyb na testovacím souboru. Vidíme zde, že na rozdíl od obrázku č. 14 mají tyto křivky velmi odlišný průběh. V tomto případě by bylo vhodné ukončit učení sítě podle pravidla uvedeného v kapitole 1.4.5., tedy v okamžiku, kdy začne horní křivka po svém poklesu opět stoupat. Úloha znázorněná na obrázku č. 15 se od problému

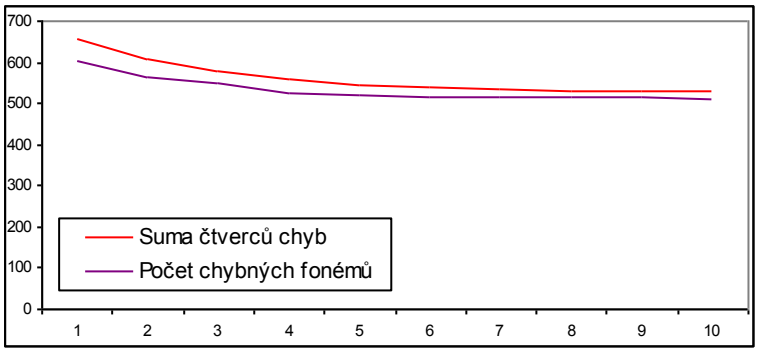
fonetické transkripce liší v tom, že počet pravidel je zde daleko vyšší a trénovací a testovací soubor obsahuje odlišná pravidla, pouze několik čtveřic parametrů je podobných a patří do stejné kategorie a proto křivka testovacích dat alespoň chvíli klesá.

2. pokus

Síť s vahami vytrénovanými 1. pokusem se učila pouze ta slova z trénovacího i testovacího souboru, která se nenaučila v 1. pokusu. Výsledkem bylo zhoršení výkonu sítě z původních 679 chybných slov na 724. V oněch 724 slovech byla většina stejných jako v souboru 679 slov trénovaných, některá z trénovaných slov ubyla (síť se je naučila), ale více jich přibylo, takže se dá říci, že při tomto „doučování" síť spíše zapomínala, než aby se učila nová slova.

Obrázek č. 16: Průběh pokračování učení sítě na souboru slov nenaučených při 1. pokusu

Doučování sítě	
Suma chyb[2]	Chybné fonémy
654	605
608	566
580	549
558	525
546	518
538	516
535	516
532	514
531	513
530	512

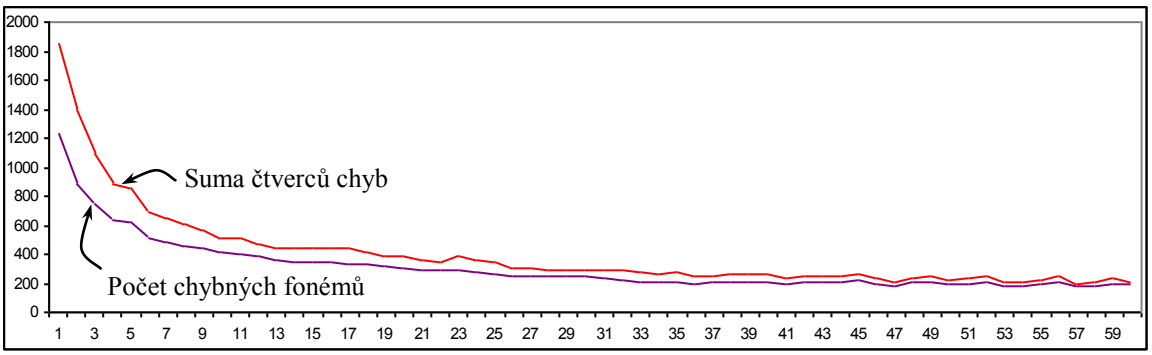


Pramen: Vlastní výzkum

3. pokus

Síť byla učena od začátku (z náhodně inicializovaných vah) soubor 1 360 slov (9 507 vzorů), který se snažil zachytit všechny jevy vyskytující se ve fonetice češtiny pokud možno tak, aby v něm byly tyto jevy zastoupeny rovnějším dílem, než je jejich výskyt v běžných textech. Chtěla jsem tak dosáhnout toho, aby se síť naučila i méně frekventovaná česká fonologická pravidla, která se nenaučila v 1. pokusu. Výsledkem bylo pouze mírné zlepšení výkonu sítě. Zatímco síť vytrénovaná v 1. pokusu se nenaučila 679 slov z trénovacího i testovacího souboru z 1. pokusu, síť vytrénovaná ve 3. pokusu se ze stejných souborů nenaučila 517 slov. Tento jen o málo lepší výsledek byl navíc dosažen až po 60 průchodech trénovacím souborem 1 360 slov.

Obrázek č. 17: Průběh učení sítě při 3. pokusu



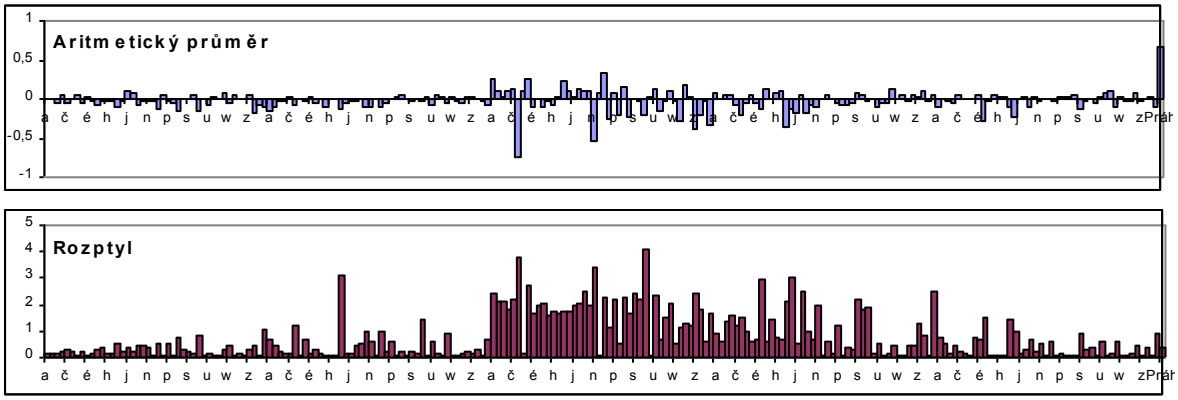
Pramen: Vlastní výzkum

Obrázek č. 17 nevypovídá pouze o tom, že s přibývajícími průchody trénovacím souborem se síť začíná učit velmi neochotně. Vidíme zde také to, že křivka sumy čtverců chyb se postupně přibližuje ke křivce počtu chybných fonémů. To se dá vyložit tak, že u chybně určených vzorů síť v pozdní fázi učení produkuje na výstupních jednotkách samé nuly.

2.2.1. Jednoduchá analýza vah naučené neuronové sítě

Při učení neuronové sítě zpravidla dochází u trénovaných vah k jejich divergenci od počáteční hodnoty. Z toho je možné vyvodit, že váhy které byly upravovány nejvíce, mají nejvyšší absolutní hodnoty. Jako zdroj dat pro obrázky č. 18 a 19 byly použity váhy neuronové sítě po 3. pokusu.

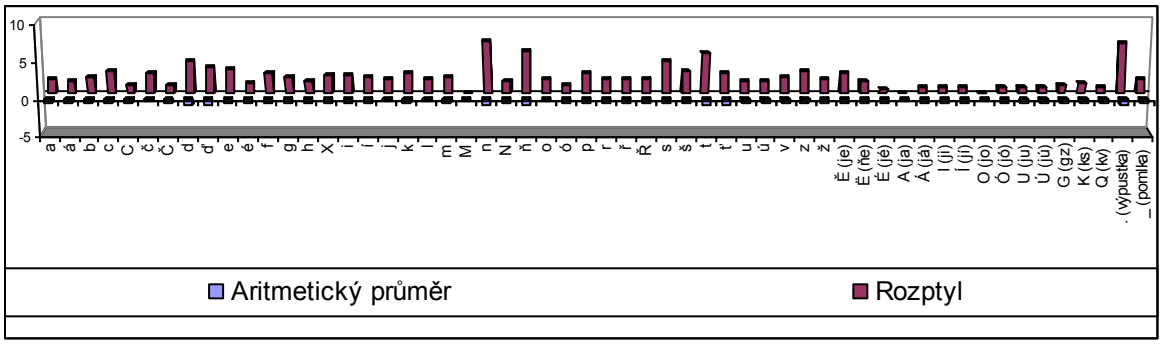
Obrázek č. 18: Aritmetický průměr a rozptyl vah vedoucích z 1. do prostřední vrstvy



Pramen: Vlastní výzkum

Z dolního grafu na obrázku č. 18 lze usoudit, že na určení fonému měl rozhodující vliv znak z prostředku zpracovávaného okénka a po něm většinou znaky následující, což je pro češtinu typické.

Obrázek č. 19: Aritmetický průměr a rozptyl vah vedoucích z prostřední do 3. vrstvy



Pramen: Vlastní výzkum

Ve 3. pokusu se síť nenaučila tyto převody znaků na foném: *ď* → *d*', *ť* → *t*', *ň* → *ň*, *cc* → *.c*, *nn* → *.n*, *kk* → *.k*, *m* → *M*, *ia* → *ija*, *io* → *ijo*. Jsou to relativně méně časté jevy v české fonetice, ale některá ještě méně frekventovaná pravidla se síť naučila, například *q* → *kv*. Tyto výsledky do jisté míry korespondují s rozptylem vah naučené neuronové sítě.

Když srovnáme písmena české abecedy od nejmenšího do největšího podle sumy rozptylů všech vah, které z jim patřících jednotek míří do prostřední vrstvy, dostaneme toto pořadí: *ň*, *ť*, *q*, *-*, *ú*, *w*, *ď*, *x*, *ó*, *ů*, *y*, *ý*, *ř*, *ch*, *h*, *f*, *j*, *é*, *g*, *r*, *ž*, *á*, *b*, *č*, *v*, *u*, *z*, *c*, *m*, *p*, *š*, *e*, *a*, *o*, *l*, *k*, *s*, *í*, (mezera), *d*, *n*, *ě*, *t*, *i*. Váhy s nejnižším rozptylem byly asi nejméně trénovány a to odpovídá skutečnosti, že síť se nenaučila *ď* → *d*', *ť* → *t*', *ň* → *ň*, i když například *q* → *kv*, *-* → *.* (vypuštění fonému neboli výpustka), *ú* → *ú*, *w* → *v* se síť naučila.

Když srovnáme fonémy od nejmenšího do největšího podle rozptylu všech vah mířících k jejich jednotkám z prostřední vrstvy, dostaneme toto pořadí: *M*, *O* (jo), *A* (ja), *É* (jé), *Í* (jí), *Q* (kv), *Á* (já), *Ó* (jó), *Ú* (jú), *I* (ji), *U* (ju), *ó*, *Č*, *C*, *G* (gz), *K* (ks), *é*, *Ë* (ňe), *h*, *á*, *N*, *u*, *ú*, *j*, *r*, *ž*, *ř*, *_* (pomlka), *l*, *o*, *Ř*, *a*, *g*, *m*, *v*, *í*, *b*, *X*, *i*, *č*, *k*, *f*, *Ě* (je), *p*, *ť*, *c*, *z*, *š*, *e*, *ď*, *d*, *s*, *t*, *ň*, *.* (výpustka), *n*. Vidíme, že 3 fonémy s nejnižším rozptylem také patří k tomu, co se síť nenaučila.

Vypadá to, jakoby při učení neuronové sítě docházelo k tomu, že některé jednotky jsou trvale odříznuty od možnosti trénovat své váhy. To se dá interpretovat také tak, že síť uvízla na lokálním a nikoli globálním minimu sumy čtverců chyb na výstupní vrstvě, což je u neuronových sítí častý problém.

2.2.2. Hledání lepších metod učení neuronové sítě

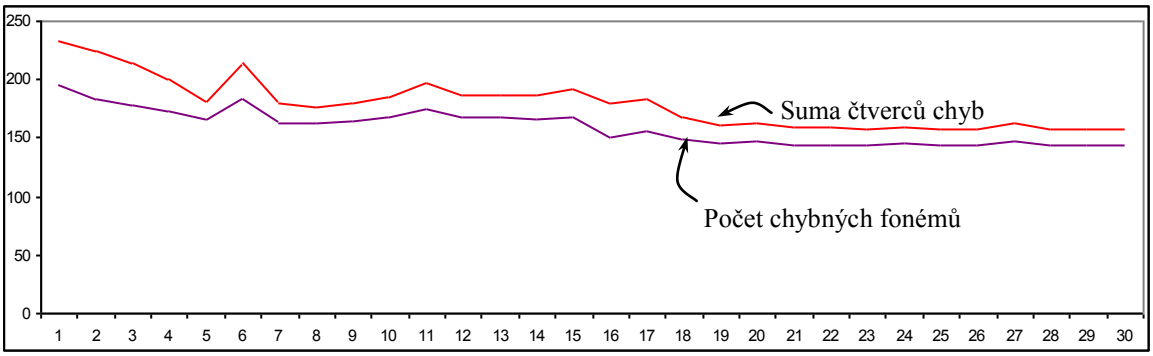
Po analýze vah jsem učinila několik pokusů o zlepšení výkonu neuronové sítě.

4. pokus

Trénování vycházelo z konečného stavu vah po 3. pokusu, trénovací soubor byl také ten samý jako ve 3. pokusu. Váhy mezi 1. a 2. vrstvou, které mířily ze znaků *ď*, *ň* a *ť*, a váhy mezi 2. a 3. vrstvou, které mířily do

fonémů *M*, *A* (ja) a *O* (jo) byly upravovány s několikanásobně vyšším parametrem rychlosti učení než ostatní váhy, ty byly upravovány s parametrem rovným 2. Výsledkem bylo to, že výkon sítě zůstával na dosavadní úrovni. Toto pozorování by mohlo být vysvětleno tím, že některé váhy se nemění, protože součet jejich přírůstků od začátku do konce trénovacího souboru dává nulu. Proto v dalším pokusu o zlepšení výkonu sítě byly opět ty samé vybrané váhy trénovány s vyšším parametrem rychlosti učení rovným 4, zatímco ostatní s parametrem 2, ale tentokrát jen v případě, že právě čtený znak je *ď*, *ň* nebo *ť*, nebo právě čtený foném je *M*, *A* (ja) nebo *O* (jo). Po 30 průchodech souborem s 1 360 slovy se výkon sítě zlepšil z původních 517 nenaučených slov na 441 slov ze souborů z 1. pokusu, což byl v mých pokusech nejlepší dosažený výkon. Síť se přitom naučila fonémy *A* (ia) a *M*. Vztáhneme-li výkon sítě k 1 360 trénovacím slovům, dostaneme zlepšení výkonu sítě z původních 0,97 % na 0,51 % chybně určených fonémů. Právě popsaný postup je také zmíněn v literatuře [8] na straně 306 v kapitole „*Adaptive Learning Rates*“ jako možná metoda trénování, když v trénovacím souboru existují vzory patřící do nějakých podstatně méně zastoupených kategorií. Jiným možným řešením tam doporučovaným je obohacení trénovacího souboru o další zástupce patřící do řídčeji obsazených kategorií.

Obrázek č. 20: Průběh učení sítě při zlepšování výkonu ve 4. pokusu



Pramen: Vlastní výzkum

5. pokus

Síť se učila od začátku (z náhodně inicializovaných vah) soubor 1 360 slov (9 507 vzorů), který byl poprvé použit ve 3. pokusu. 5. pokus se od 3. pokusu lišil v tom, že v něm bylo použito *momentum* rovné 0,5. Když bylo při stejném parametru rychlosti učení rovném 2 použito *momentum* rovné 0,9, došlo při výpočtech rychle k přetečení. Při parametru rychlosti učení rovném 1 a *momentu* rovném 0,9 byla po prvním průchodu trénovacím souborem téměř 100 % chybovost určených fonémů. *Momentum* je popisováno například v knížce [8] na straně 305.

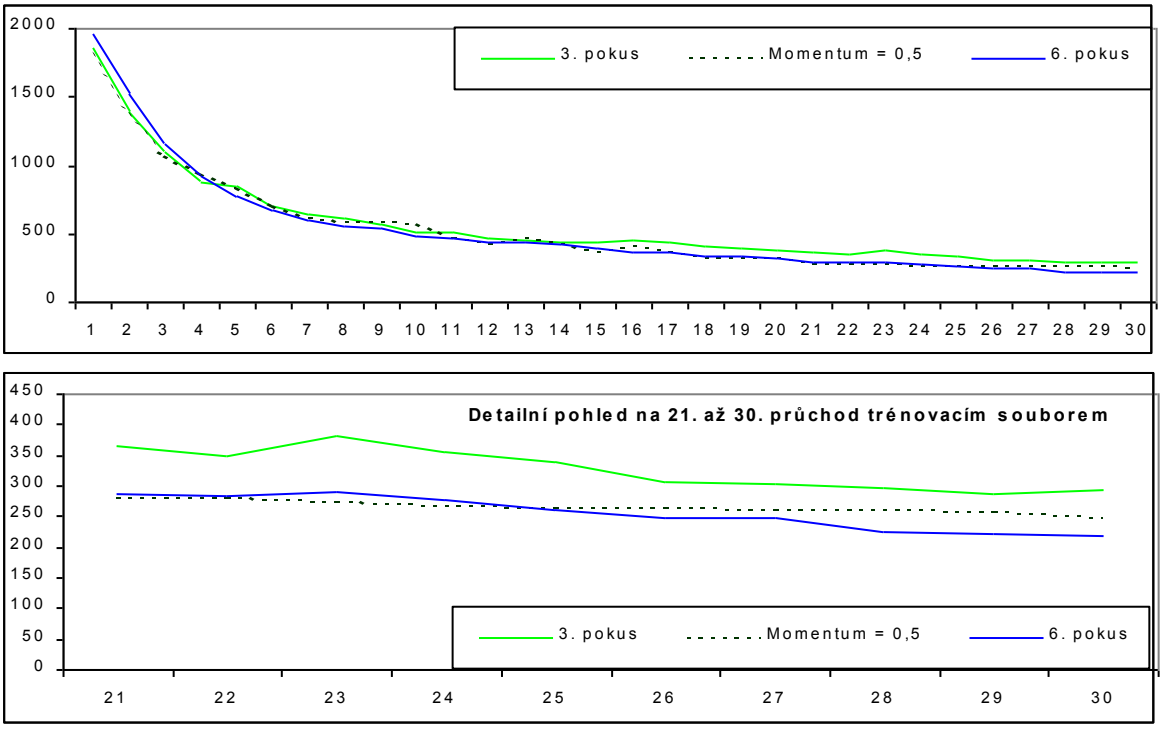
Dále jsem vyzkoušela metodu učení neuronové sítě zvanou *delta-bar-delta*, viz strana 307 v knížce [8]. Při ní se sleduje vývoj každé váhy po 3 období a každá váha má svůj vlastní parametr rychlosti učení. Aby se ušetřilo místo v paměti počítače, minulé a předminulé období se ukládá do jedné proměnné jako vážený průměr. Pokud váha po sledované období vykazuje stálý růst nebo stálý pokles, její parametr rychlosti učení se zvýší o nějakou malou hodnotu. Pokud váha ve sledovaném období osciluje, její parametr rychlosti učení se sníží vynásobením nějakým vhodným koeficientem. Tato metoda se při této úloze neosvědčila, pravděpodobně proto, že při aktualizaci vah po každém vzoru docházelo tak často k úpravě parametrů rychlosti učení, že nebylo možné najít správnou hodnotu koeficientu pro zvyšování parametru rychlosti učení váhy, který se k němu přičítá, a koeficientu pro jeho snižování, kterým se násobí tak, aby nedocházelo k nežádoucímu vychýlení celého systému vah z optimální střední hodnoty.

Další postup, který se u této úlohy neosvědčil, byla *Nguyen-Widrowova inicializace* počátečních vah mezi vstupní a skrytou vrstvou, viz strana 297 v knížce [8]. Tato metoda má urychlit vytvoření vnitřní reprezentace vzorů na skryté vrstvě sítě. Tento postup vedl často k přetékání výpočtu a k vysoké chybovosti určených fonémů po prvním průchodu trénovacím souborem.

6. pokus

Nakonec jsem vyzkoušela metodu pokoušející se kombinovat výhody *momenta* a metody *delta-bar-delta*. Tato metoda spočívá v tom, že u každé váhy se sleduje historie jejího vývoje stejně, jako je tomu u metody delta-bar-delta. Pokud váha ve sledovaném období pouze roste nebo pouze klesá, je k ní připočteno *momentum*, pokud váha ve sledovaném období osciluje, její učení proběhne bez *momenta*. Při této metodě byl použit v prvních 30 průchodech trénovacím souborem parametr rychlosti učení rovný 2 a *momentum* 0,9. Srovnání průběhu učení ve 3. pokusu, učení s *momentem* v 5. pokusu a průběhu učení v 6. pokusu přináší obrázek č. 21.

Obrázek č. 21: Porovnání průběhu učení sítě vyjádřené sumou čtverců chyb při 3. pokusu, při použití momenta v 5. pokusu a v 6. pokusu



Pramen: Vlastní výzkum

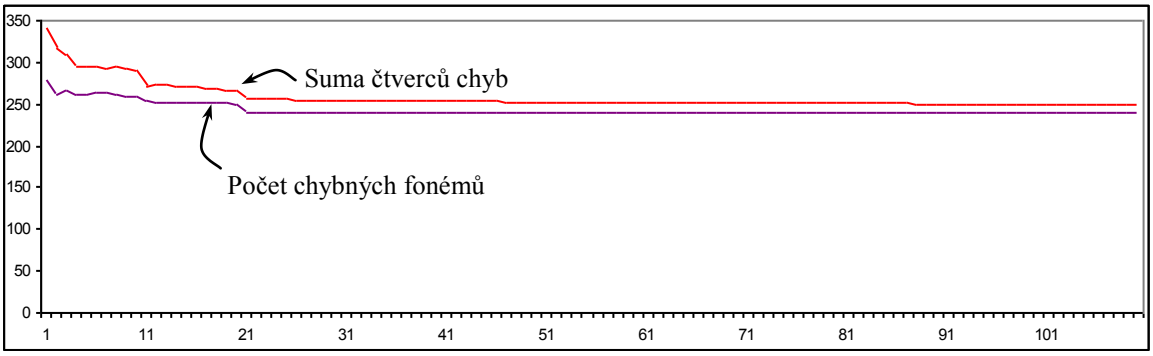
Obrázek č. 21 ukazuje, že učení sítě s *momentem* je o něco málo úspěšnější než standardní *back-propagation* a ještě o něco lepší než standardní *momentum* je používání *momenta*, jen když váha neosciluje. Protože metoda učení použitá v 6. pokusu se ukázala být nejlepší, využila jsem ji v následujících pokusech.

2.2.3. Učení neuronové sítě s přihlédnutím k fonologickým pravidlům

7. pokus

Zjistila jsem, že v 6. pokusu se síť nenaučila po 30 průchodech trénovacím souborem 10 fonologických pravidel úplně a 4 částečně. Výskyt těchto 14 jevů jsem v trénovacím souboru zdvojnásobila přidáním jiných slov s těmito jevy, takže nyní tento soubor obsahoval 1 500 slov tvořících 10 504 vzorů. Při následném učení rozšířeného trénovacího souboru z počátečního stavu vah po 6. pokusu byl v prvních 40 průchodech trénovacím souborem parametr rychlosti učení rovný 1, od 41. průchodu 0,5 a od 51 průchodu 0,1. *Momentum* bylo stále 0,9. Po 110 průchodech rozšířeným trénovacím souborem se síť naučila pouze ta 4 v předchozím pokusu částečně nenaučená pravidla. Z toho plyne závěr, že doučování sítě na souboru se zvýšeným počtem dříve nenaučených vzorů není příliš účinné.

Obrázek č. 22: Průběh pokračování učení sítě v 7. pokusu



Pramen: Vlastní výzkum

8. pokus

V češtině jsem identifikovala 136 fonologických pravidel, podle kterých jsem vytvořila trénovací soubor poprvé použitý ve 3. pokusu. Těchto 136 pravidel obsahovalo všech 92 pravidel z tabulky č. 5, ke kterým jsem přidala v češtině často používané přípony a předpony a další frekventovanější spojení písmen. Z rozšířeného trénovacího souboru ze 7. pokusu jsem vybrala 100 slov zachycujících všech 136 pravidel. Těchto 100 slov (703 vzorů) se neuronová síť učila od začátku, tj. z náhodně inicializovaných vah. Učení jsem zastavila po 160 průchodech 100 slovy, protože počet chybných fonémů rovný 14 se již nesnižoval. Těchto 14 chyb odpovídalo 14 fonologickým pravidlům. Učinila jsem i pokus se sítí se 112 skrytými neurony, ale rychlost učení se tím nezvýšila.

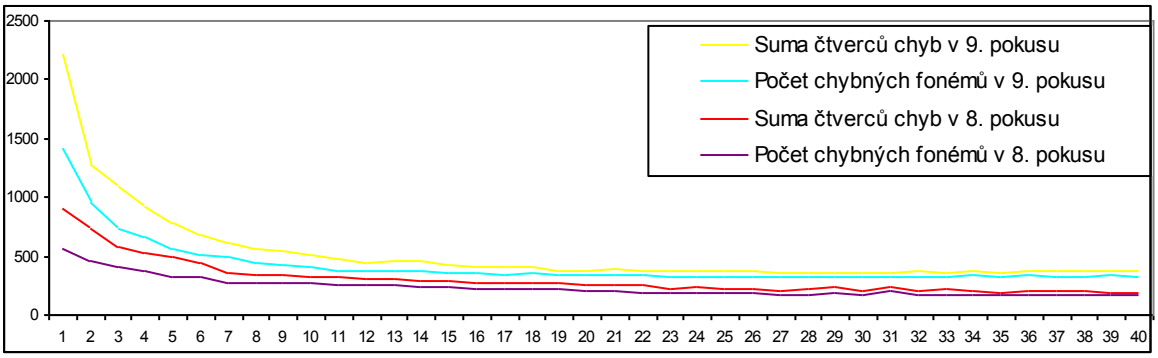
Z tohoto počátečního stavu síť 40-krát prošla souborem 1 500 slov ze 7. pokusu. Výsledkem bylo, že síť z toho, co předtím uměla, zapomněla 2 pravidla, a naučila se 5 nových, takže 9 dříve nenaučených pravidel stále neuměla.

9. pokus

V 8. pokusu se síť z trénovaných 100 nenaučila 14 slov. K těmto 14 slovům jsem přidala dalších 6 slov zachycujících alespoň 6 dalších fonologických pravidel. Vzniklý soubor 20 slov (148 vzorů) se síť po 111 průchodech naučila úplně. Soubory takto malého počtu slov je síť schopna se naučit kompletně, rychlost učení závisí na počátečním stavu vah. Po 120 průchodech jsem učení ukončila.

Z tohoto počátečního stavu síť 40-krát prošla souborem 1 500 slov ze 7. pokusu. Po tomto učení síť neuměla 18 pravidel, z nichž žádné nebylo obsaženo ve 20 původně naučených slovech. Tudíž to, co se síť naučí na začátku, již většinou nezapomene. Můžeme si proto na začátku učení vybrat několik pravidel a naučit je síť přednostně. Tato možnost je však vykoupena tím, že potom se síť naučí méně pravidel, než by se naučila, kdyby se učila všechna pravidla najednou. V 8. pokusu se síť nenaučila 11 pravidel, ale v 9. pokusu se jich nenaučila 18.

Obrázek č. 23: Průběh pokračování učení sítě po předchozím naučení se menších souborů



Pramen: Vlastní výzkum

Z obrázku č. 23 je vidět, že síť, která se předtrénuje na souboru s omezeným počtem fonologických pravidel, se potom hůře učí nová pravidla. Čím méně je pravidel, na kterých se síť předtrénuje, tím větší je pravděpodobnost, že si je síť bude pamatovat i při učení rozšířeného souboru se všemi pravidly, ale tím méně nových pravidel se síť v rozšířeném souboru poté naučí.

10. pokus

V tomto pokusu jsem síť předtrénovala na souboru s jediným pseudoslovem obsahujícím všechny možnosti přepisu písmen na fonémy:

```
úbábccčddódídďošszžegyhůhkkkll-liaáeéiíoóuůjjmměmfěnnéng ňněqurřssšštttťtířvvwxaxpýzzzžž žšch
úbáp.c.č.dóďítďtocčCČegihúX.kg.l.liAÁĚÉIÍOÓUÚ.j.mĚMfĚ.néNk_ňňeQurřsz.š.tdťtíŘvfvGaKpí.zsžš..šX
```

Toto slovo představuje 94 vzorů a síť se jej naučila po 140 přečteních.

Dále síť pokračovala v učení se souboru ze 7. pokusu. Výkon takto vytrénované sítě byl lepší než v předchozích pokusech, ale nikoli na trénovacím a testovacím souboru z 1. pokusu. Znamená to, že síť se sice naučila více pravidel než kdy dřív, (nenaučila se jich 6), ale ta nenaučená patřila bohužel k těm relativně užívanějším v běžných textech.

Dále jsem zkusila předtrénovanou síť učit trénovací soubor obsahující 1 450 slov (10 027 vzorů), ve kterém jsem odstranila výjimky, ve kterých se slabiky *di*, *ti*, *ni* vyslovují jako *di*, *ti*, *ni*.

Abych porovnala úspěšnost učení sítě za těchto různých podmínek, využila jsem míru výkonu sítě zvanou *Root Mean Square error* (*RMS error*), viz kapitola 1.4.5. Z obou posledně popsaných pokusů jsem si vybrala sumu čtverců chyb po 30. průchodu trénovacím souborem, vypočítala z ní *RMS error* a výsledek jsem porovnala

se stavem učení sítě po 30. kroku v 6. pokusu, protože metoda učení sítě v 10. pokusu je s tímto pokusem shodná.

6. pokus:

suma čtverců chyb = 219, počet vzorů = 9 507, *RMS error* = $(219 : (9\,507 \cdot 56))^{½} = 0{,}02028179$.

10. pokus s výjimkami:

suma čtverců chyb = 166, počet vzorů = 10 504, *RMS error* = $(166 : (10\,504 \cdot 56))^{½} = 0{,}01679897$.

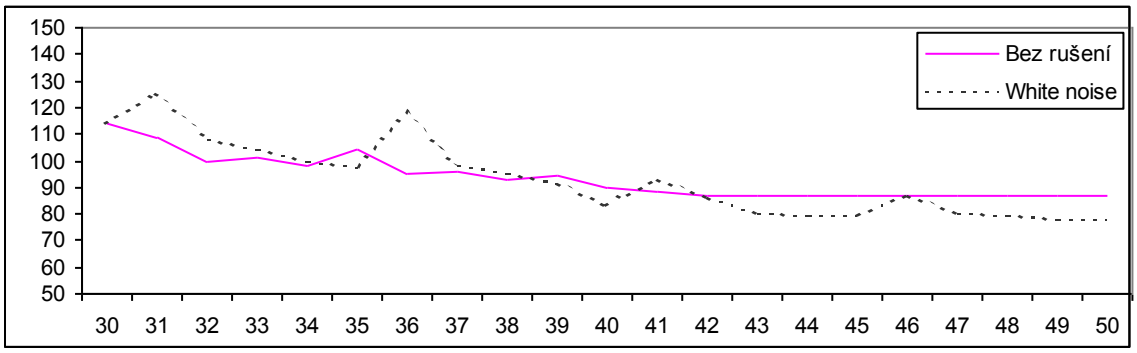
10. pokus bez výjimek:

suma čtverců chyb = 114, počet vzorů = 10 027, *RMS error* = $(114 : (10\,027 \cdot 56))^{½} = 0{,}014248623$.

Míra *RMS error* ukázala, že předtrénování sítě na souboru obsahujícím všechny možné přepisy písmen na fonémy zlepšuje výkon sítě při učení skutečným slovům. Míra *RMS error* dále ukázala, že učí-li se síť soubor bez výjimek, její výkon na tomto souboru je lepší než výkon, jaký by síť dosáhla při učení na souboru s výjimkami.

Na konci kapitoly 1.4.5. navrhuji zavedení drobných poruch do vah, takzvaný *white noise,* pro zvýšení pravděpodobnosti úniku stavu vah z lokálních minim, což by mělo urychlit učení sítě v jeho pozdní pomalé fázi. Užitečnost této rady jsem ověřila na učení sítě na souboru bez výjimek v 31. až 50. průchodu. Obrázek č. 24 porovnává, jak se vyvíjelo učení sítě měřené sumou čtverců chyb bez přidávání poruch do vah a s přidáním poruch do vah na začátku 31., 36., 41. a 46. průchodu trénovacím souborem. Poruchy v intervalu od –0,5 do +0,5 byly přičítány ke každé váze v síti.

Obrázek č. 24: Porovnání učení sítě bez zavádění poruch do vah a s aplikací šumu na všechny váhy



Pramen: Vlastní výzkum

Obrázek č. 24 svědčí o užitečnosti zavádění poruch do vah pro urychlení učení, avšak ani tato metoda není výrazně efektivní oproti ostatním mnou vyzkoušeným metodám učení.

2.3. Závěr

Tato studie představuje podle mých znalostí první experiment s fonetickou transkripcí češtiny pomocí neuronové sítě. Ukazuje, že tradiční metody fonetické transkripce využívající soubor pravidel a tabulku výjimek jsou úspěšnější než neuronová síť, protože neuronové síti se obvykle nepodaří naučit se všechna pravidla a výjimky. Tradiční expertní systém dovede přijímat nové poznatky relativně snadno. Je k tomu třeba jen přidat nová pravidla do jeho programu a nové výjimky do jeho tabulky výjimek. Neuronová síť by dosáhla naučení se nového pravidla jen po nekonečném počtu průchodů trénovacím souborem obsahujícím jak naučené tak dosud nenaučené vzory. Nejlepší výsledek, kterého jsem svou sítí dosáhla (ve 4. pokusu) byl 95,73 % správně přepsaných slov ze souboru 10 334 nejfrekventovanějších slov. Účinnost tradičních expertních systémů je při fonetické transkripci téměř 100%.

Neuronová síť tedy není nejlepším nástrojem při řešení problémů, u kterých umíme definovat pravidla pro klasifikaci vzorů, může však být nenahraditelná pro řešení problémů, kde tato pravidla neznáme.

Obsah

Literatura

[1] http://www.itee.uq.edu.au/~cogs2010/cmc/index.html (zkráceno a volně přeloženo)
[2] W. S. McCulloch and W. H. Pitts (1943/1965). A Logical Calculus of Ideas Immanent in Nervous Activity. In W. S. McCulloch (Ed.), Embodiments of Mind, pp. 19-39. Cambridge, MA: MIT Press.
[3] J. L. McClelland and D. E. Rumelhart (1981). An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings. Psychological Review, 88, pp. 375-407; D. E. Rumelhart & J. L. McClelland (1982). An Interactive Activation Model of Context Effects in Letter Perception: Part 2. The Contextual Enhancement Effect and Some Tests and Extensions of the Model. Psychological Review, 89, pp. 60-94.
[4] F. Rosenblatt (1957). The Perceptron: A Perceiving and Recognizing Automaton. Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory.
[5] P. H. Winston: Artificial Intelligence, Third Edition, Addison Wesley, Reading, MA, 1992, ISBN 0-201-53377-4.
[6] R. Beale, T. Jackson: Neural Computing: An Introduction, IOP Publishing Ltd., Bristol, Philadelphia, 1992.
[7] D. E. Rumelhart, G. E. Hinton, R. J. Williams Learning Internal Representations by Error Propagation. In: D. E. Rumelhart, J. L. McClelland (Eds.) Parallel Distributed Processing, Volume 1, MIT Bradford Press, Cambridge, MA, pp. 318-362, 1986.
[8] L. Fausett: Fundamentals of Neural Networks, Architectures, Algorithms and Applications, Prentice Hall International, Inc., New Jersey, 1994, ISBN 0-13-334186-0.
[9] T. Kohonen (1982). Self-Organized Formation of Topologically Correct Feature Maps. In Biological Cybernetics, 43, pp. 59-69.
[10] D. O. Hebb (1949). The Organization of Behavior: A Neuropsychological Theory. New York: Wiley. Partially reprinted in Anderson and Rosenfeld (1988).
[11] J. J. Hopfield: Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In Proceedings of the National Academy of Sciences of the USA, Volume 79, pages 2554-2588, 1982.
[12] J. J. Hopfield: Neurons with Graded Response Have Collective Computational Properties Like Those of Two-Sate Neurons. In Proceedings of the National Academy of Sciences of the USA, Volume 81, pages 3088-3092, 1984.
[13] M. Minsky, S. Papert (1969). Perceptrons: Introduction to Computational Geometry. Cambridge, MA: MIT Press. Expanded edition reprinted in MIT Press. ISBN 0262631113. (May 1988).
[14] J. Psutka: Komunikace s počítačem mluvenou řečí, Academia, Praha, 1995, ISBN 80-200-0203-0.
[15] The International Phonetic Alphabet. Journal of the Phonetic Association, vol. 19, no. 12, Dec. 1989.
[16] J. Nouza, J. Psutka, J. Uhlíř: Phonetic Alphabet for Speech Recognition of Czech. Radio Engineering, vol. 6, no. 4, December 1997, pp. 16-20.
[17] T. J. Sejnowski and C. R. Rosenberg (1986). NETtalk: a Parallel Network That Learns to Read Aloud. In Cognitive Science, 14, pp. 179-211.
[18] T. J. Sejnowski and C. R. Rosenberg (1987). Parallel Networks That Learn to Pronounce English Text. In Complex Systems, 1, pp. 145-168.