Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Nástroj na interaktívne experimentovanie s umelými neurónovými sieťami

Diplomová práca

2014 Bc. Milan Lajtoš

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Nástroj na interaktívne experimentovanie s umelými neurónovými sieťami

Diplomová práca

| | |
|---|---|
| Študijný program: | Kognitívna veda |
| Študijný odbor: | 9.2.11 Kognitívna veda |
| Školiace pracovisko: | Centrum pre kognitívnu vedu |
| Školiteľ: | RNDr. Kristína Rebrová PhD. |

Bratislava 2014 Bc. Milan Lajtoš

# Zadanie

## Anotácia

Umelé neurónové siete sú jedným z najčastejšie používaným modelovacím nástrojom v kognitívnej vede. Patria medzi povinnú literatúru aj v doméne umelej inteligencie. Často ich však ľudia považujú za niečo príliš zložité až mysteriózne, pretože pre ich kompletné porozumenie je potrebné ovládať zložitú matematiku a pre ich implementáciu zas programovanie na určitej úrovni.

Napriek tomu, že v modernej kognitívnej vede dominuje prístup spoznávania pomocou skúsenosti, paleta grafických demonštračných nástrojov na výuku neurónových sietí je stále pomerne chudobná. Ideálnym médiom pre vytvorenie platformy pre demonštráciu neurónových sietí je teraz už všade dostupný web. S pokrokom webových technológií a programovacích jazykov prichádza aj dobrá možnosť pre vznik výučbového prostredia pre online vytváranie a trénovanie neurónových sietí.

## Cieľ

V teoretickej časti práce, študent priblíži model umelého neurónu a typy umelých neurónových sietí, v ktorých sa takýto neurón používa. V súlade s nadobudnutými poznatkami z kognitívnej psychológie, študent navrhne softvérové prostredie, v ktorom možno interaktívne experimentovať s takýmito sieťami. Následne v praktickej časti práce, študent naimplementuje navrhnutý softvérový nástroj schopný demonštrácie rôznych typov sietí na vybranom probléme.

## Literatúra


1. Rojas, *Neural Networks: A Systematic Introduction*
2. Johnson, *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*
3. Victor, *Media for Thinking the Unthinkable*

# Abstrakt

LAJTOŠ, Milan: Nástroj na interaktívne experimentovanie s umelými neurónovými sieťami [Diplomová práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Centrum pre kognitívnu vedu. Vedúci diplomovej práce: RNDr. Kristína Rebrová PhD., Bratislava: FMFI UK, 2014. 45 s.

Táto práca sa zaoberá biologicky-inšpirovaným výpočtovým modelom – umelými neurónovými sieťami. Argumentujeme, že hlboké neurónové siete sú perspektívny výpočtový model a poukazujeme na fakt, že jeho využitie je silne obmedzené nástrojmi, ktoré poskytujú túto funkcionalitu. Tvrdíme, že treba nové nástroje, ktoré nám pomôžu vytvárať a porozumieť systémom, ktoré sú založené na umelých neurónových sieťach. Popisujeme tri princípy, ktoré by mali byť dodržané pri vytváraní takéhoto prostredia a predstavujeme prototyp, v ktorom sú tieto princípy implementované.

Kľúčové slová: strojové učenie, hlboké neurónové siete, vizuálne programovanie

# Abstract

LAJTOŠ, Milan: Interactive Experimentation Tool for Artificial Neural Networks [Master Thesis]. Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics; Centre for Cognitive Science. Master Thesis Supervisor: RNDr. Kristína Rebrová PhD., Bratislava: FMFI UK, 2014. 45 p.

This thesis deals with biologically-inspired computational model – artificial neural networks. We argue that deep neural networks are a prospective computational model, and we point out that its application is heavily limited due to tools, which provide this functionality. We claim there is a need for new tools which can help us create and understand systems based on the neural networks. We define three principles, which should be adhered to when conceiving such environment, and we present a prototype which implements these principles.

Keywords: Machine Learning, Deep Neural Networks, Visual Programming

# Predhovor

Príchod strojovej inteligencie je len otázkou času. Aby sme tento čas skrátili, potrebujeme nástroje, ktoré nám umožnia *vytvárať a porozumieť systémom*, ktoré sú medzistupňom k skutočnej strojovej mysli.

Táto záverečná práca pojednáva o takomto nástroji a princípoch, ktoré by mal spĺňať. Tiež predstavuje prototyp, ktorý vznikol aplikáciou spomínaných princípov na oblasť umelých neurónových sietí, ktoré sú v oblasti strojového učenia najperspektívnejší model ľudskej kognície.

Toto dielo by nevzniklo bez podpory, ktorej sa mi za uplynulý čas dostalo. Chcel by som sa poďakovať RNDr. Kristíne Rebrovej PhD. za ničím neobmedzený priestor pri tvorbe tejto práce. Rovnako by som sa chcel poďakovať aj Slnku môjho života, ktoré mi rozjasňuje každý jeden deň a aj každú jednu noc, a teda mi nedá spávať. Avšak najväčšia vďaka patrí mojej rodine, bez ktorej by nič z tohto nebolo možné a ani nutné.

> What I cannot create, I do not understand.
>
> Richard Feynman

# Obsah

# Zoznam obrázkov

# Úvod

Cieľom výskumu v oblasti umelej inteligencie, je vytvoriť umelú myseľ. Táto myseľ sa musí správať ako dokonalá čierna skrinka – žiadne nastaviteľné parametre, žiadne ladenie a žiadne vnútorné zásahy. Skutočne samostatný *mysliaci stroj*. Samozrejme, toto potrvá ešte nejaký čas... Medzitým potrebujeme nástroje, ktoré nám umožnia vytvárať a porozumieť systémom, ktoré sa stanú predchodcami k skutočnej strojovej mysli.

Nástroje, ktoré nám umožňujú vytvárať systémy imitujúce inteligentné správanie, si zvyčajne vyžadujú akademické znalosti na hranici ľudského poznania. Tieto vedomosti sú častokrát nedostupné alebo zabalené vo forme, ktorá nie je prístupná väčšine ľudí. Ak chceme výskum v oblasti umelej inteligencie urýchliť, potrebujeme nástroje, ktoré umožňujú experimentovanie s takýmito systémami len s minimálnymi znalosťami.

Oblasť hlbokých neurónových sietí je sľubná cesta, ktorá poskytuje mimoriadne výsledky na mnohých úlohách strojového učenia ako napr. rozpoznávanie vizuálnych vnemov, rozpoznávanie reči, predikcia molekulárnej aktivity a ďalších. Avšak na použitie tohto výpočtového modelu sú potrebné matematické znalosti a schopnosť programovať.

Tieto prekážky sa dajú obísť, keď budeme používať nástroj, ktorý nám umožní vytvárať a porozumieť neurónovým sieťam prirodzenejším spôsobom ako symbolickými reprezentáciami, ktoré používa matematika a programovanie. Tento nástroj musí klásť dôraz na všetky tri stupne mentálnych reprezentácií – na enaktívne, ikonické a symbolické. Teda musí poskytovať rozhranie pre akcie, obrazy a symboly.

Vizuálne programovanie umožňuje používateľom vytvárať programy grafickou manipuláciou prvkov, namiesto písania zdrojového kódu programu. Takéto prostredie je založené na myšlienke „krabíc a šípok“, kde krabice predstavujú entity a šípky vzťahy medzi nimi. Paradigma vizuálneho programovania je častokrát používaná na návrh komplexných systémov a teda je vhodnou voľbou na vytváranie neurónových sietí.

Vizuálne programovanie trpí rovnakým problémom ako písanie zdrojového kódu – človek iba slepo narába so symbolmi. Vizuálne programovanie neprináša žiadnu pridanú hodnotu oproti tradičnému programovaniu. Tento problém môže byť odstránený zobrazením dát, ktoré prúdia cez sieť. Rozdiel v týchto dvoch prístupoch je porovnateľný s rozdielom medzi MRI a fMRI zobrazovacími technikami. Porozumenie architektúre nie je dostatočné na vysvetlenie dejov, odohrávajúcich sa v tomto dynamickom systéme.

### Členenie práce

Táto práca je delená na 3 kapitoly. V prvej kapitole popisujem historické snaženie o vytvorenie umelej mysle. V druhej kapitole sa nachádzajú teoretické základy ohľadom konekcionistického modelovania kognitívnych procesov. V tretej kapitole predstavené prostredie na vytváranie neurónových sietí a aj popisané možné pokračovanie týchto myšlienok.

# Kapitola 1

# Model mysle

## 1.1 Výpočet

Historicky najhlbší vhľad do podstaty myslenia nám ponúkol v 17. storočí René Descartes. Tvrdil, že myšlienky sú len *symbolické reprezentácie* analogické k matematickým reprezentáciám, t.j. mentálne reprezentácie a objekty, ktoré reprezentujú môžu byť oddelené. V tomto rámci uvažovania myslenie nie je nič iné ako manipulácia symbolických reprezentácií – *výpočet*.[1]

Thomas Hobbes, Descartov súčasník, tiež došiel k záveru, že myslenie má matematickú podstatu. Opísal ju nasledovne:

> „Ak niekto myslí, nerobí nič iné ako že si predstavuje celkový súčet sčítaním častí alebo zbytok odčítaním jedného súčtu od iného.“[2]

Ak sa tejto definície pridržíme, otázka *umelej mysle*, resp. stroja vykonávajúceho myslenie, príde sama od seba.

## 1.2 Mysliace stroje

Idea, že myslenie sa dá redukovať na sériu výpočtov, je z rôznych pohľadov veľmi zaujímavá. Ak predpokladáme jej pravdivosť, tak by bolo možné zostrojiť mechanizmus, ktorý by takýto výpočet vykonával – *mysliaci stroj*.

Využitie takéhoto stroja je pomerne neobmedzené, pretože môže vykonávať akúkoľvek *myšlienkovú* činnosť, ktorú zvládne človek. Prvá možnosť ako by sa takýto stroj dal využiť je, že by mohol myslieť za nás. Oslobodil by nás tak od námahy, ktorú musíme denne podstupovať.

---

[1] Petrů, *Fyziologie mysli: Úvod do kognitivní vědy*, str. 31.

[2] Hobbes, *Leviathan neboli látka, forma a moc státu církevního a občanského*, kap. 5, O rozumu a vědě.

Na druhej strane, vo svetle Descartovho výroku *„Myslím teda som“*, naberá predstava stroja, ktorý myslí za nás, zvláštny rozmer. Ak by sme už nemuseli myslieť, prestali by sme existovať? Tento názor nás paradoxne núti zamyslieť sa a *byť* ešte viac.

Najvhodnejším spôsobom ako sa pozerať na mysliace stroje je považovať ich za predĺženie ľudskej mysle. Podobne ako nám rôzne nástroje umožňujú znásobiť vrodené schopnosti, tak nám umelá myseľ môže otvoriť dvere, o ktorých sme ani nevedeli, že existujú...

## 1.3 Počítacie stroje

### 1.3.1 História počítacích strojov

História počítacích strojov obsahuje množstvo osobností a ich objavy, ktoré posunuli poznanie podstaty výpočtu veľkými skokmi dopredu.

#### Počítacie hodiny

Úplne prvý počítací stroj zostrojil Wilhelm Schickard v roku 1623. Schickardov stroj používal ozubené kolieska pôvodne určené pre hodiny, preto sa jeho vynález niekedy označuje ako „počítacie hodiny“. Aj napriek faktu, že tento stroj vedel iba sčítavať a odčítavať čísla, našiel si praktické využitie – Johanes Kepler ho použil pri svojich astronomických výpočtoch.

Schickard prišiel s myšlienkou počítacieho stroja ako prvý, avšak ako to už býva zvykom, byť prvý častokrát nestačí. Tam kde všetko začalo, aj všetko skončilo.

#### Pascalina

Nezávisle od Schickarda, Blaise Pascal v roku 1642 zhotovil počítací stroj nazvaný Pascalina. Rovnako ako jeho predchodca, tento stroj vedel iba sčítavať a odčítavať. Pascal však urobil jeden zásadný krok, ktorý ho odlíšil od Schickarda.

Pascalov dizajn počítacieho stroja bol natoľko bezchybný, že sa rozhodol rozšíriť svoj vynález medzi ľudí, čo malo rôzne následky. Spoľahlivosť Pascaliny ohromila úradníkov až do takej miery, že sa začali obávať o stratu svojho zamestnania.

Pascal mal viac šťastia ako Schickard, jeho dielo inšpirovalo ďalšie generácie, ktoré myšlienku počítacieho stroja posunuli vpred.

#### Leibnizov projekt

Ďalším krok v histórii počítacích strojov učinil Wilhelm Gottfried von Leibniz v roku 1694. Leibniz zdokonalil Pascalov dizajn a vytvoril počítací stroj, ktorý navyše vedel násobiť a aj deliť.

Leibniz mal však rozpracovaný väčší plán, ktorý mal priniesť skutočný mysliaci stroj, nie iba pomôcku na počítanie. Aj napriek tomu že neuspel, zanechal svojím pokračovateľom základy, ktoré sa stali kľúčové pri plnení jeho sna.

### Analytický stroj

Pokračovateľom Leibnizovej myšlienky bol Charles Babbage, ktorý zostrojil Diferenčný stroj. Ten umožňoval počítať s polynomiálnymi funkciami, čo mnohonásobne zvyšovalo použitie stroja v rôznych oblastiach matematiky.

Babbage cítil obmedzenia svojho vynálezu, no vytrval a v roku 1837 opísal stroj, ktorý by umožňoval vypočítať čokoľvek – Analytický stroj. Myšlienka tohto stroja predstihla možnosti svojej doby a kvôli technickým problémom Babbage svoju prácu nikdy nedokončil.

### Elektronická kalkulačka

Predchádzajúce stroje uskutočňovali výpočet čisto mechanicky – otáčali ozubenými kolesami. George Stibitz toto zmenil, keď v roku 1937 dokončil prvý číslicový počítací stroj, ktorý používal elektromagnetické spínače. Vďaka elektronickým súčiastkam bol Stibitzov stroj podstatne rýchlejší ako dovtedajšie pokusy o počítacie stroje.

### Turingov stroj

V roku 1937 bol predstavený ďalší počítací stroj, ktorý však nemal fyzickú formu. Tento teoretický stroj vymyslel Alan Turing, ktorý priamo nadviazal na Babbageov Analytický stroj.

Hlavnou charakteristikou Turingovho stroja je zapisovateľná *páska*, *hlava*, ktorá sa pohybuje po páske a *strojová tabuľka* s pravidlami. Činnosť Turingovho stroja sa dá popísať nasledovne:

1. stav stroja a hodnota z aktuálneho políčka pásky udávajú vstupnú konfiguráciu
2. strojová tabuľka obsahuje mapovanie zo vstupnej konfigurácie na výstupnú konfiguráciu
3. výstupná konfigurácia udáva novú hodnotu zapísanú na pásku, posun pásky v istom smere a zmenu stavu

Na prvý pohľad sa zdá, že tento stroj je príliš jednoduchý na to, aby mohol vykonávať akúkoľvek zmysluplnú činnosť, nie to ešte myslieť. Avšak Turing dokázal, že jeho stroj je schopný akéhokoľvek výpočtu.[3]

[3]Turing, "On Computable Numbers with an Application to the Entscheidungsproblem".

Turing navyše ukázal, že je možné zostrojiť stroj s takou tabuľkou, ktorý dokáže simulovať všetky iné stroje. Tento *univerzálny Turingov stroj*, prečíta z pásky strojovú tabuľku iného stroja a vykoná operácie presne tak ako by ich vykonal popísaný stroj.

Turing svojím objavom otvoril Pandorinu skrinku. Na uskutočnenie umelej mysle už nebolo potrebné konštruovať zložité mechanizmy, stačilo by nájsť tú *správnu* strojovú tabuľku.

#### ENIAC a pokračovatelia

Prvý čisto elektrický počítací stroj, ktorý bol schopný vykonať akúkoľvek vypočítateľnú funkciu, bol ENIAC v roku 1946.[4] Od tohto momentu sa výpočtový svet zmenil už iba málo.

Poslednú veľkú zmenu do počítacích strojov priniesol EDVAC, na ktorého vývoji sa podieľal aj John Von Neumann. Narozdiel od ENIAC-u používal dvojkovú sústavu namiesto desiatkovej. EDVAC priniesol aj zmenu v nastavovaní strojovej tabuľky. Kým ENIAC musel byť programovaný použitím prepínačov a spojovacích káblov, EDVAC mal strojovú tabuľku uloženú v pamäti, teda podobne ako univerzálny Turingov stroj. Architektúra zavedená Von Neumannom je používaná dodnes.

#### Súčasnosť

Za posledných sedem desaťročí sa udialo vo výpočtovom svete mnoho objavov, ktoré posúvajú hranice stále ďalej. Ako tvrdí Moorov zákon, exponenciálny rast výpočtovej sily stále pokračuje a nezdá sa, že by sa v tomto ohľade malo niečo zmeniť. Výpočtové stroje budú rýchlejšie, menšie, úspornejšie a lacnejšie. Dôsledkom tohto trendu bude vytvorenie paralelne distribuovaného systému vytvoreného z jednoduchých výpočtových jednotiek, ktoré sú extrémne efektívne.

Počítacie stroje majú za sebou dlhú históriu počas ktorých prešli podstatnými zmenami, avšak v ich najhlbšej podstate stále ostávajú rovnocenné univerzálnemu Turingovmu stroju. A to sa týka nielen výpočtovej sily, ale aj problémov s nastavovaním ich strojovej tabuľky...

### 1.3.2 Programovanie

Počítací stroj, resp. počítač, ktorý dokáže vykonať akúkoľvek vypočítateľnú funkciu, je v mnohých smeroch výhrou. Avšak kde sa jeden problém vyrieši, tam vznikne ďalší, doplnkový. Tento problém sa týka nastavenia strojovej tabuľky výpočtového stroja, teda *problém programovania* počítača.

[4] Babbageov Analytický stroj by bol prvým mechanickým. Prvý elektro-mechanický postavil Konrad Zuse už v roku 1941.

Návod ako vypočítať nejakú funkciu – algoritmus, musí vložiť do stroja človek – programátor. Táto úloha si vyžaduje programátorov vhľad do riešeného problému, ale aj schopnosť previesť riešenie do formy zrozumiteľnej pre počítač. Na obe tieto činnosti musí človek vynaložiť námahu, ktorej nás mali počítače zbaviť.

### Spôsob programovania

Prvé počítače boli programované na tej najnižšej možnej úrovni. Na zmenu ich činnosti boli používané *prepínače* a *spojovacie káble*. Nastavenie počítača a vyriešenie problému boli dve odlišné veci. Programátor napísal program na papier, ktorý bol predaný operátorom počítača a tí nastavili počítač do želaného stavu.

Príchodom *uloženého programu* sa programovanie podstatne zjednodušilo. Úloha programátora a operátora počítača sa zlúčila, čím došlo k užšiemu prepojeniu medzi programátorom a programovaným. Programátor zapísal sled inštrukcií na úložné médium a počítač ich po prečítaní vykonal. Takýto program musel byť napísaný v strojovom kóde počítača, teda programátor si musel prispôsobiť myslenie strojovým reprezentáciám.

Veľkú zmenu priniesol špeciálny program, ktorý vedel vytvárať iné programy – *kompilátor*. Kompilátor, resp. prekladač, dokáže preložiť program z formy zrozumiteľnej človeku do strojového kódu. Program mohol nadobúdať rôznu podobu, ktorá však musela dodržiavať pravidlá, ktoré diktoval *programovací jazyk*. Programátor už nie je nútený prispôsobiť sa stroju a môže sa vyjadrovať spôsobom, ktorý je bližší ľudskému mysleniu.

Ako poznáme množstvo vzorov myslenia, tak existujú aj rôzne štýly programovania. Napríklad, dlho sa verilo, že rozmýšľame v logických krokoch, preto vznikli programovacie jazyky, ktoré využívali predikátovú logiku ako stavebnú jednotku. Alebo objektovo-orientované programovanie popisuje svet ako objekty interagujúce na viacerých úrovniach opisu. Zlatým pravidlo však je, že *paradigma programovania odzrkadľuje paradigmu myslenia*.

V súčastnosti existujú tisíce programovacích jazykov delené do mnohých skupín podľa rôznych kritérií, avšak jedna vlastnosť je zdieľaná medzi všetkými. Podobne ako prirodzený jazyk, tak aj programovacie jazyky používajú *textovú reprezentáciu*. A to platí nielen pre ukladanie programov, ale aj pre proces ich tvorby...

## 1.4 Učiace sa stroje

Problém programovania sa už pri začiatkoch počítacích strojov javil ako prekážka. Už Ada Lovelace, Babbageova spolupracovníčka a prvá programátorka, označila počítacie stroje za neschopné učenia sa. Mala pravdu. Čiastočne.

Turing, ako zakladateľ hnutia umelej inteligencie, mal veľké ambície – chcel uskutočniť ideu mysliaceho stroja. Avšak samotný počítač na umelú myseľ nestačí – to čo počítač vykoná je priamo dané programátorom. Turing si uvedomoval toto obmedzenie a navrhol, aby sa počítače začali správať inteligentnejšie, teda ako ľudia. Táto myšlienka sa stala stredobodom v oblasti výskumu umelej inteligencie.

### 1.4.1 Neusporiadaný stroj

Samotný Turing sa pokúsil o vyriešenie problému programovania. V roku 1948 predstavil teoretický stroj, ktorý sa skladá z množstva navzájom prepojených jednoduchých prvkov. Stroj *typu A*, ako ho Turing nazval, používal logický operátor NAND ako výpočtovú jednotku. Takýto stroj mal na začiatku náhodné nemeniteľné prepojenia a jeho správanie bolo chaotické. Turing ho popísal ako *neusporiadaný stroj*.[5]

Nevýhodou stroja typu A je samozrejme jeho fixný program, ktorý je uložený v kombinácií spojení medzi jednotkami. Turing upravil stroj a zmenil mu spojenia, tak aby toto spojenie bolo modifikovateľné. Tento *menič spojenia* bol vytvorený z rovnakých jednotiek ako samotný stroj a umožňoval prepustiť signál, zrušiť ho alebo striedavo prepúšťať a rušiť. Stroj *typu B* je teda schopný zmeny svojho vnútorného programu a teda aj úpravu svojho správania.

Aby neusporiadaný stroj začal vykazovať želané správanie, musí byť podmienený výchove, podobne ako dieťa. Turing popísal metódu učenia stroja *typu P*, ktorá obsahovala dva signály – *potešenie* a *potrestanie*. Týmto režimom bol Turing schopný doviesť stroj do stavu, kedy sa začal správať ako univerzálny stroj.

Tento sľubný prístup k strojom schopných učenia nezískal veľa nasledovníkov. Dokonca aj samotný Turing bol odradený od pokračovania výskumu neusporiadaných strojov a preorientoval sa na výskum umelého života.

### 1.4.2 Prahová logická jednotka

Iný spôsob k vytvoreniu stroja schopného učenia zvolil neurovedec Warren McCulloch a logik Walter Pitts. V roku 1943 navrhli model biologického neurónu, nazvaný *Prahová logická jednotka*.[6] Tento umelý neurón prijíma niekoľko binárnych hodnôt cez váhové spojenia a všetky integruje do jedného výsledku. Ak výsledok prekročí istý prah, výstupná hodnota bude pozitívna, inak negatívna.

Vhodne nastavené hodnoty váh a prahu umožňujú simulovať základné logické operácie a ich spojením môžeme vyjadriť akúkoľvek logickú operáciu. Sieť takýchto neurónov s cyklami, predstavuje dynamický systém s pamäťou, čo je pre stroje schopných učenia veľmi zaujímavá vlastnosť.

[5] Turing, "Intelligent Machinery".
[6] McCulloch and Pitts, "A logical calculus of the ideas immanent in nervous activity".

Takýto model mal však jednu veľkú nevýhodu – nevedel sa učiť. Hoci bolo zrejmé, že mechanizmus učenia bude pracovať so zmenou váh a prahu, neexistovala metóda, ktorá by to bola schopná vykonať automaticky.

### 1.4.3 Perceptrón

Frank Rosenblatt pokračoval v práci McCullocha a Pittsa a v roku 1957 predstavil zlepšený model biologického neurónu – *Perceptrón*. Zlepšenie spočívalo v možnosti učenia, čo bolo dosiahnuté vďaka použitiu reálnych hodnôt pre váhy spojení.

Učenie prebiehalo nasledovne: zaznamenáme aktiváciu neurónu na istý podnet. Ak je odpoveď správna nemodifikujeme váhy. Pri nesprávnej odpovedi modifikujeme tie váhy, ktoré boli aktívne nasledovným spôsobom: ak neurón mal byť aktívny a nebol, tak váhy zväčšíme o malú hodnotu. V opačnom prípade váhy zmenšíme.

Perceptrón bol schopný učenia, čo mu zabezpečilo sľubnú budúcnosť. Na tú si však musel počkať. Problém nastal v roku 1969 kedy Marvin Minsky a Seymour Papert poukázali na isté nedostatky Perceptrónu[7] a odporučili skúmať iné smery v umelej inteligencii. Týmto krokom odradili celú komunitu od skúmania perceptrónov na najbližšiu dekádu.

### 1.4.4 Spätné šírenie chyby

Po dlhej prestávke nastal o učiace sa stroje opäť veľký záujem. Ten bol vyvolaný možnosťou skladania perceptrónov do sietí, ktoré mohli mať viacero vrstiev. Viacvrstvové neurónové siete boli známe už dávno, avšak ich učenie predstavovalo problém, ktorý bol vyriešený až v roku 1986 objavom algoritmu *spätného šírenia chyby*.[8]

Učenie pomocou tohto algoritmu prebieha veľmi podobne ako učenie Perceptrónu, avšak je zovšeobecnené pre celú sieť. Zaujímavou vlastnosťou sietí učených týmto algoritmom je hierarchické rozdelenie rozpoznávania. Neuróny vo vnútorných vrstvách siete sú naučené rozpoznávať pravidelnosti vo vstupoch, ktoré prijímajú. Komplexný proces rozpoznávania je teda distribuovaný a vzniká interakciou medzi jednoduchými výpočtovými jednotkami.

### 1.4.5 Hlboké neurónové siete

Vďaka obnovenému záujmu o neurónové siete sa zistilo mnoho vlastností, ktoré vyplývajú z ich architektúry. Napríklad siete, ktoré majú aspoň jednu skrytú vrstvu sa

[7] Ich kritika spočívala v tom, že Perceptrón dokáže počítať iba lineárne separovateľné funckie, napr. nedokáže sa naučiť binárnu operáciu XOR. Zdá sa, že ich kritika bola opodstatnená, pretože biologický neurón dokáže počítať aj lineárne neseparovateľné funkcie. (Cazé et al., "Passive dendrites enable single neurons to compute linearly non-separable functions.")

[8] Rumelhart et al., *Learning representations by back-propagating errors*.

dokážu naučiť akúkoľvek spojitú funkciu, t.j. neurónové siete sú *univerzálnym aproximátorom*.[9] Toto tvrdenie nehovorí nič o počte neurónov na skrytej vrstve alebo o trvaní učenia, avšak tento objav podporil dôveru v neurónové siete.

Ukázalo sa, že hĺbka siete je podstatným faktorom pre efektívne učenie. Algoritmus spätného šírenia chyby však produkoval slabé výsledky kvôli tzv. *problému miznúceho gradientu chyby*.[10] Tento problém sa podarilo čiastočne vyriešiť troma technikami:

1. lepšie modely neurónov a neurónových sietí vďaka výskumu biologických systémov
2. mnoho trénovacích dát kvôli digitalizácii informácií
3. rýchlejšie a efektívnejšie počítače najmä kvôli miniaturizácii výpočtových jednotiek

Tieto faktory posunuli neurónové siete na prvú pozíciu vrámci strojového učenia v rôznych oblastiach ako napr. rozpoznávanie obrazu, rozpoznávanie reči, predikcia molekulárnej aktivity, atď.[11]

[9] Cybenko, "Approximation by Superpositions of a Sigmoidal Function".
[10] Bengio et al., "Learning Long-Term Dependencies with Gradient Descent is Difficult".
[11] Schmidhuber, "Deep Learning in Neural Networks: An Overview".

# Kapitola 2

# Konekcionistické modelovanie

Konekcionistický prístup k modelovaniu kognitívnych procesov si berie za vzor biologické neurónové siete. Tieto siete sa skladajú z obrovského počtu relatívne jednoduchých výpočtových jednotiek – neurónov.

## 2.1 Biologický neurón

Biologický *neurón*, znázornený na obrázku 2.1, sa skladá z troch hlavných štrukturálnych častí:

**Dendrit** (z gr. *déndron*, strom) je dostredivý výbežok neurónu. Jeho stromová štruktúra slúži na integráciu prijímaných signálov z iných neurónov.

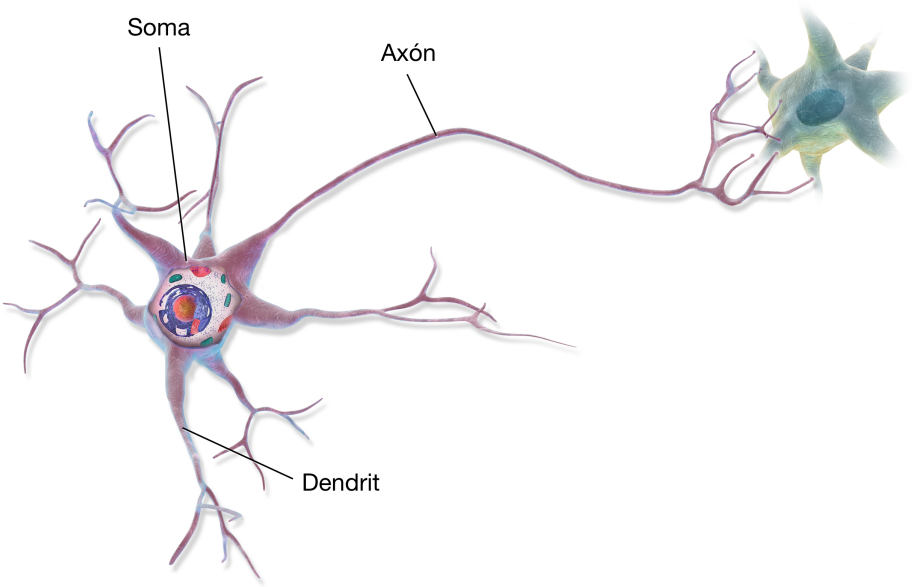
**Soma** (z gr. *sóma*, telo) je hlavnou časťou neurónu. Telo je zodpovedné za metabolizmus bunky a vyhodnotenie integrovaného signálu.

**Axón** (z gr. *axón*, os) je dlhý odstredivý výbežok neurónu, ktorý vykonáva prenos integrovaného signálu k iným neurónom.

### 2.1.1 Neurotransmisia

Neuróny medzi sebou komunikujú chemickým procesom, ktorý sa odohráva v medzere medzi neurónmi, v *synapse*. Presynaptický neurón vylúči z konca axónu chemický mediátor, nazývaný *neurotransmiter*, a ten sa naviaže na dendrit postsynapticého neurónu.

Množstvo vylúčeného mediátora priamo ovplyvňuje správanie postsynaptického neurónu. Ak *súhrnný výsledok* prijatých neurotransmiterov presiahne istý *prah*, neurón vygeneruje *akčný potenciál*, ktorý zapríčiní vylúčenie mediátorov na axóne. Signál sa tak prenesie na ďalšie neuróny.

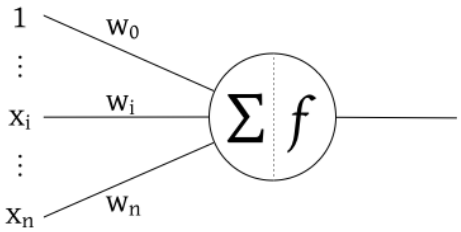



Obr. 2.1: Schéma biologického neurónu

Poznáme dva typy mediátorov, *excitačné* a *inhibičné*. Excitačné mediátory zvyšujú šancu, že postsynaptický neurón bude aktívny. Naopak inhibičný mediátor znižuje aktivitu a tým klesá súhrnný výsledok, čo môže zapríčiniť neaktivitu postsynaptického neurónu.

## 2.2 Umelý neurón

Abstrahovaním funkcií biologického neurónu dostaneme umelý neurón, ktorý si zachováva výpočtové vlastnosti a zanedbáva nezaujímavé vlastnosti ako napr. metabolizmus. Schématický popis umelého neurónu je možné vidieť na obrázku 2.2.




Obr. 2.2: Schéma umelého neurónu

### 2.2.1 Váhový vektor

Neurón obsahuje niekoľko vstupov, ktoré nadobúdajú reálne hodnoty a predstavujú hodnoty, ktoré neurón chce spracovať:

$$\mathbf{x} = (x_0, x_1, x_2, \ldots, x_n); \mathbf{x} \in \mathbb{R}^{n+1}$$

Každý z týchto vstupov má svoju vlastnú váhu, ktorá odpovedá sile spojenia medzi neurónmi:

$$\mathbf{w} = (w_0, w_1, w_2, \ldots, w_n); \mathbf{w} \in \mathbb{R}^{n+1}$$

Ak je hodnota váhy kladná, hovoríme o excitačnom spojení. Ak je hodnota záporná, tak o inhibičnom. Nulová váha vyjadruje neexistujúce spojenie medzi neurónmi.

Sila signálu pre jeden vstup teda závisí na veľkosti vstupu ako aj na váhe spojenia:

$$y_i = x_i \times w_i \tag{2.1}$$

Nultý člen oboch vektorov slúži ako špeciálna hodnota pre neurón, tzv. *bias*. Vstup $x_0$ je fixne nastavený na hodnotu 1 a váha $w_0$ nadobúda rôzne hodnoty. Úlohou biasu je škálovať ostatné vstupy.

### 2.2.2 Sumačná funkcia

Integrácia signálu je vykonaná sčítaním všetkých váhovaných vstupov dokopy[1], teda lineárna kombinácia vstupov a váh:

$$y = \sum_{i=1}^{n} w_i x_i \tag{2.2}$$

Ak si zvolíme vektorovú reprezentáciu vstupných hodnôt a váh, tak integráciu môžeme vyjadriť ako skalárny súčin týchto vektorov:

$$y = w \cdot x \tag{2.3}$$

### 2.2.3 Prenosová funkcia

Prenosová funkcia, často nazývaná aj aktivačná funkcia, má z hľadiska spracovania informácie dôležité postavenie. Jej úlohou je vyhodnotenie integrovaného signálu a propagácia do ďalších neurónov. Existuje viacero funkcií, ktoré sa používajú s väčšou, či menšou úspešnosťou.

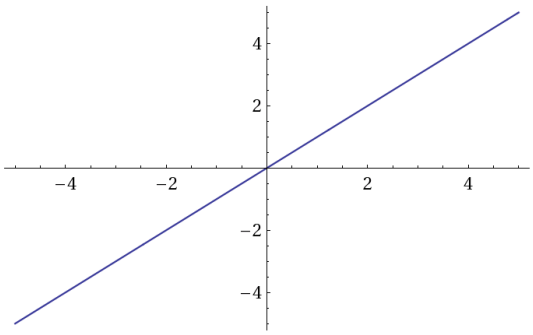
[1] Akoby sa zabúdalo na to, že topológia dentritu je hierarchická.

### Funkcia identity

Najjednoduchšou prenosovou funkciou je identita, teda neurón pošle ďalej iba nezmenený integrovaný signál:

$$f(y) = y \tag{2.4}$$

Táto funkcia má veľkú nevýhodu – počíta iba lineárnu transformáciu vstupov. Ak by sme teda poskladali sieť z vrstiev, ktoré by obsahovali iba takéto neuróny, na výstupe by sme opäť dostali lineárnu transformáciu. A tú vieme vypočítať aj jednou vrstvou.

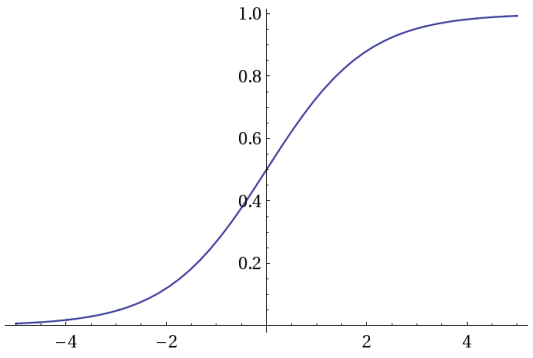



Obr. 2.3: Funkcia identity

### Sigmoidálna funkcia

Na vyriešenie problému s lineárnosťou výstupu sa začala používať nelineárna prenosová funkcia. Vhodným kandidátom bola jednoduchá sigmoidálna funkcia:

$$f(y) = \sigma(y) = \frac{1}{1 + e^{-y}} \tag{2.5}$$




Obr. 2.4: Sigmoidálna funkcia

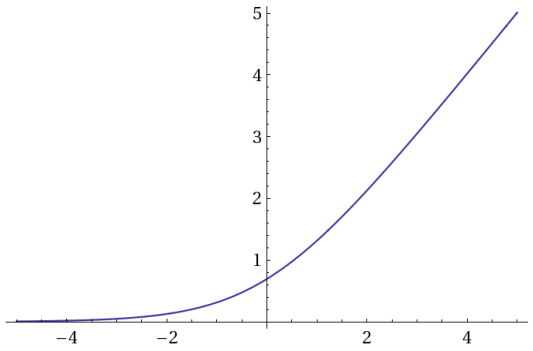
Táto prenosová funkcia sa dlhú dobu používala ako štandard, ktorý poskytoval uspokojivé výsledky. Miernou nevýhodou tejto funkcie je jej derivácia, ktorá pri veľmi vysokých a veľmi nízkych hodnotách sa blíži k nule, čo spomaľuje učenie.

### Softplus funkcia

Neurón používajúci *softplus* prenosovú funkciu má veľmi zaujímavé správanie – simuluje výpočet nekonečného počtu sigmoidálnych neurónov.[2]

Všetky kópie zdieľajú váhy, avšak každý neurón má mierne posunutý prah aktivácie o istú konštantu. Ak sú tieto konštanty rovné $-0{,}5$, $-1{,}5$, $-2{,}5$, atď., tak výsledný súčet sa dá aproximovať uzavretou formou:

$$f(y) = \sum_{i=1}^{\infty} \sigma(y - i + 0{,}5) \approx \log_e(1 + e^y) \tag{2.6}$$




Obr. 2.5: Softplus funkcia

Hoci má *softplus* aktivačná funkcia veľmi zaujímavé vlastnosti, vysoká výpočtová náročnosť znemožňuje jej použitie v praxi.

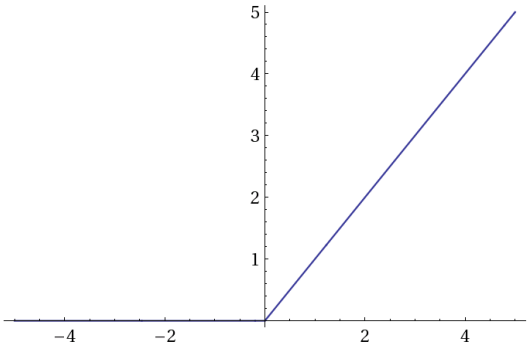
### Funkcia max

Pre softplus funkciu existuje aproximácia, ktorá si zachováva vynikajúce vlastnosti, avšak je výpočtovo nenáročná. Definovaná je nasledovne:

$$f(y) = \max(0, y) = \begin{cases} 0, & y \leq 0 \\ y, & y > 0 \end{cases} \tag{2.7}$$

Táto funkcia je až zarážajúco jednoduchá – pre nulu a záporné hodnoty vracia nulu a pre kladné hodnoty vracia nezmenený vstup, t.j. funkcia vracia väčší prvok z dvojice $(0, x)$. Vďaka výpočtovej nenáročnosti je funkcia $\max(0, y)$ veľmi vhodnou prenosovou funkciou pre hlboké siete s miliónmi, či dokonca miliardami spojení.

[2]Nair and Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines".




Obr. 2.6: Funckia $\max(0, y)$

Ďalšou pozitívnou vlastnosťou tejto funkcie je jej derivácia definovaná nasledovne:

$$f'(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases} \tag{2.8}$$

Vidíme, že derivácia pre kladné hodnoty je vždy rovná jednej, čo znamená, že zmena váh bude vždy konštantná. Táto črta je obzvlášť pozitívna – neurón netrpí tzv. problémom miznúceho gradientu chyby, ktorý komplikuje učenie hlbokých sietí.[3]

Potenciálny problém nastáva až v druhom prípade, keď je aktivácia neurónu nulová, teda keď je vstup záporný alebo rovný nule. Keďže pri takýchto vstupoch je derivácia nulová, zmena váh nenastane. Tento fakt môže zapríčiniť tzv. *smrť neurónu*, t.j. situáciu, kedy váhy neurónu produkujú nulovú aktiváciu, ktorá spätne nemá žiaden vplyv na zmenu jeho váh. V tomto prípade váhy ostávajú fixované bez možnosti akejkoľvek zmeny a neurón sa tak stáva zbytočným pretože nenastáva propagácia aktivácie.[4]

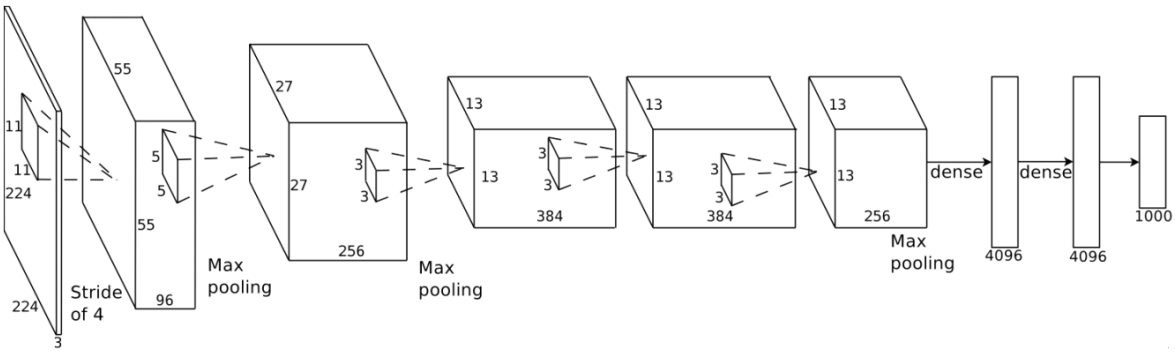
Nulová aktivácia sa na prvý pohľad môže zdať ako vážny problém, avšak opak je pravdou. Problém so stratou informácie sa vyrieši nasledovne: ak sa informácia prenesie aspoň cez jeden neurón vo vrstve, tak problém prestáva existovať. Tento jav podporuje distribuované a riedke reprezentácie, čo činí takéto neuróny biologicky prijateľnými.[5]

## 2.3 Neurónové siete

Spojením popísaných neurónov vytvoríme umelú *neurónovú sieť*. Existuje mnoho spôsobov ako ich usporiadať aby vykonávali nejaký výpočet. Jednu z najpoužívanejších techník usporiadania neurónov predstavujú dopredné neurónové siete, ktorým sa budeme ďalej venovať.

*Dopredné neurónové* siete nemajú spätnú väzbu, t.j. signál sa propaguje len jedným

[3] Bengio et al., "Learning Long-Term Dependencies with Gradient Descent is Difficult".
[4] Maas et al., "Rectifier Nonlinearities Improve Neural Network Acoustic Models".
[5] Glorot et al., "Deep Sparse Rectifier Neural Networks".




Obr. 2.7: Architektúra neurónovej siete Supervision

smerom cez vrstvy rôzneho typu. Každá vrstva zachytáva opakujúce sa pravidelnosti vo výstupoch predchádzajúcej vrstvy, čím vzniká hierarchická interakcia medzi neurónmi.

Hoci majú dopredné siete obmedzenú výpočtovú schopnosť kvôli chýbajúcej spätnej väzbe, stále ostávajú dobrou voľbou na mnoho problémov. Výborné výsledky podávajú najmä pri probléme klasifikácii vizuálnych vnemov, teda rozpoznávaní objektov na obrázkoch.

#### Supervision

Vynikajúce výsledky v rozpoznávaní objektov dosiahla konvolučná neurónová sieť *Supervision*,[6] ktorej architektúra je zobrazená na obrázku 2.7. Táto sieť je tvorená z nasledujúcich typov vrstiev: vstupná vrstva, konvolučná vrstva, max-pooling vrstva, úplne prepojená vrstva a výstupná vrstva. Každú si teraz stručne popíšeme.

### 2.3.1 Vstupná vrstva

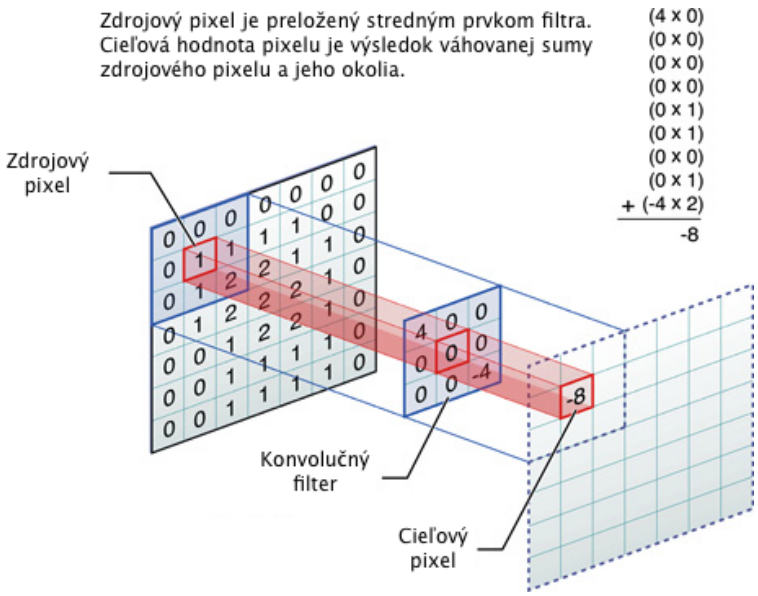
Vstupná vrstva neurónovej siete reprezentuje *senzorické neuróny*, ktoré signalizujú prítomnosť (zvyčajne) externej informácie.

V biologických systémoch to je napr. *sietnica* v oku, kde neuróny detegujú prítomnosť fotónov istej vlnovej dĺžky. V umelých systémoch je sietnica reprezentovaná neurónmi, ktoré si zachovávajú dvojrozmernú štruktúru vizuálneho vnemu.

Hodnoty vstupných neurónov reprezentujú intenzitu farby v danom bode obrazu. Ak máme iba čierno-biely vstup, tak neurón vyjadruje jas bodu – nízke hodnoty vyjadrujú nedostatok informácie (málo svetla), a opačne. Pri farebných obrázkoch to platí pre každý kanál zvlášť.

Podstatnou vlastnosťou vstupnej vrstvy by mala byť normalizácia vstupných hodnôt, t.j. hodnoty by mali spadať do istého rozsahu. V biologických systémoch túto úlohu zohráva *zrenica*, ktorá zamedzuje prílišnému príjmu svetla na sietnicu.

[6] Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks."




Obr. 2.8: Konvolúcia

### 2.3.2 Konvolučná vrstva

Konvolučná vrstva[7] je biologicky inšpirovaná výskumom vizuálneho kortexu cicavcov, kde boli identifikované neuróny, ktoré sú vysoko citlivé na malé výseky vizuálneho vstupu. *Receptívne polia*, ako sa tieto oblasti nazývajú, pokrývajú celé vizuálne pole a sú špecializované na detekciu vzorov zvyčajne podobných hranám.[8]

Konektivita týchto neurónov je simulovaná procesom *konvolúcie*. Konvolúcia (viď. obr. 2.8) je matematická operácia, ktorá produkuje prekryv dvoch funkcií, v tomto prípade dvoch obrázkov. Prvý obrázok je výrez zo vstupnej vrstvy neurónovej siete a druhý obrázok je receptívne pole registrujúce nejakú črtu. Výsledkom prekryvu týchto obrázkov je hodnota, ktorá odpovedá odozve neurónu na daný stimul.

Výhodou konvolučnej vrstvy je zdieľanie parametrov – váhy neurónu, ktorý detekuje istú črtu, sú použité na celý vizuálny vstup. Napr. na detekciu 45° hrany bude potrebný iba jeden špecializovaný neurón, ktorý bude použitý na celý vstup, nielen na vybranú oblasť.

Konvolučná vrstva má niekoľko nastaviteľných parametrov, ktoré ovplyvňujú jej správanie a veľkosť výstupu:

1. počet filtrov
2. veľkosť filtra, napr. $3 \times 3$
3. veľkosť kroku konvolúcie v oboch smeroch, napr. $1 \times 1$

[7] LeCun et al., "Gradient-based learning applied to document recognition".
[8] Hubel and Wiesel, "Receptive fields and functional architecture of monkey striate cortex".

### 2.3.3 Pooling vrstva

Podobne ako konvolučná, tak aj pooling vrstva je biologicky inšpirovaná vizuálnym kortexom cicavcov. Bunky, ktoré sú modelované pooling vrstvou majú väčšie receptívne pole a sú lokálne invariantné na stimuly.

Pooling vrstva zvyčajne nasleduje za konvolučnou vrstvou a znižuje veľkosť vstupnej informácie procesom *podvzorkovania*. Neurón v max-pooling vrstve propaguje iba salientné črty, ktoré konvolučná vrstva zachytila. Výstupom takejto vrstvy sú teda maximálne aktivácie z predchádzajúcej vrstvy. Výpočet závisí iba na vstupoch – nepoužíva sa žiaden váhový vektor.

Rovnako ako konvolučná vrstva, tak aj pooling vrstva pracuje s výrezmi vstupov, ktoré sú parametrizované nasledujúcimi hodnotami:

1. veľkosť okna, napr. $5 \times 5$
2. veľkosť kroku v oboch smeroch, napr. $2 \times 2$

### 2.3.4 Úplne prepojená vrstva

Úplne prepojená vrstva neobmedzuje konektivitu v žiadnom smere – všetky neuróny v takejto vrstvy sú spojené meniteľnými váhami s každým neurónom z predchádzajúcej vrstvy. Kvôli tomuto je počet spojení vysoký, čím je učenie siete pomalšie.

#### Regularizácia

Pri veľkej vrstve zvykne dochádzať k situácii, kedy trénovacie dáta sú naučené príliš dobre. V takomto prípade neurónová sieť slabo generalizuje, čo sa prejaví ako zlé výsledky na testovacích dátach.

Možnosťou ako sa vyhnúť tomuto javu je použitie novej regularizačnej techniky, tzv. *dropoutu*.[9] Táto technika zabraňuje koadaptácii váh neurónov vnesením šumu do aktivácie neurónov.

Každý neurón v úplne prepojenej vrstve má pravdepodobnosť aktivácie rovnú $p = 0{,}5$. To znamená, že pri každom trénovacom prípade používame iba polovicu neurónov a vždy inú sadu. Na dropout sa teda môžeme pozerať ako na trénovanie $2^n$ rôznych modelov zdieľajúcich váhy, pričom každý model je trénovaný len na jednom prípade.

### 2.3.5 Výstupná vrstva

Neurónová sieť, ktorá je určená na klasifikáciu vnemov, zvyčajne produkuje pravdepodobnosť do akej kategórie patrí nejaký stimul. Na výpočet týchto pravdepodobností sa používa tzv. *softmax* funkcia definovaná nasledovne:

[9] Hinton et al., "Improving neural networks by preventing co-adaptation of feature detectors".

$$p_j = \frac{e^{y_j}}{\sum_{k=1}^{n} e^{y_k}} \tag{2.9}$$

Index maximálnej hodnoty z vektoru pravdepodobností $p$ nám udáva kategóriu, ktorá produkuje najvyššiu mieru istoty pre nejaký stimul.

# Kapitola 3

# Interaktívne experimentovanie

Človek sa učí na základe skúseností s prostredím, ktoré mu poskytuje odozvu na jeho akcie. Ak je táto odozva nedostatočná, učenie môže prebiehať iba ťažko. Nástroj, ktorý umožňuje jednoduché použitie neurónových sietí, musí poskytovať prostredie, kde je možné pocítiť dôsledok vykonaných akcií okamžite.

## 3.1 Súčasné riešenia

Väčšina nástrojov, ktoré poskytujú funkcionalitu neurónových sietí, neumožňuje *aktívne* spoznávať princíp ich fungovania. Tieto nástroje sú určené pre ľudí, ktorí ich používajú na riešenie špecifických problémov, nie na pochopenie ich správania. Príkladom takého nástroja môže byť Caffe[1], ConvNetJS[2], Cuda-ConvNet[3], EBLearn[4], Pylearn2[5], Torch[6] a ďalšie.

Všetky tieto nástroje obsahujú možnosť naprogramovania želanej neurónovej siete použitím rôznych programovacích jazykov. Od programátora sa teda vyžaduje len absolútne minimum – vedieť programovať a rozumieť neurónovým sieťam.

### 3.1.1 Špecializované jazyky

Isté nástroje poskytujú veľmi šikovné riešenie pre prvý problém. Keďže učenie neurónovej siete vyžaduje len popis architektúry a niekoľko ďalších parametrov, ovládať všeobecný programovací jazyk je zbytočné.

Riešením tohoto problému sú popisné jazyky, ktoré špecifikujú iba potrebné parametre pre vytvorenie a učenie neurónovej siete. Ako takýto jazyk vyzerá, je možné

[1] http://caffe.berkeleyvision.org/
[2] http://cs.stanford.edu/people/karpathy/convnetjs/
[3] https://code.google.com/p/cuda-convnet/
[4] http://eblearn.sourceforge.net/
[5] http://deeplearning.net/software/pylearn2/
[6] http://torch.ch/

vidieť na nasledovnej ukážke:

```
name: "network"
input: "data"
input_dim: 64
input_dim: 1
input_dim: 28
input_dim: 28

layers {
  name: "conv"
  type: CONVOLUTION
  bottom: "data"
  top: "conv"
  convolution_param {
    num_output: 9
    kernel_size: 5
    stride: 1
  }
}

layers {
  name: "relu"
  type: RELU
  bottom: "conv"
  top: "pool"
}

layers {
  name: "pool"
  type: POOLING
  bottom: "relu"
  top: "pool"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layers {
  name: "ip"
  type: INNER_PRODUCT
  bottom: "pool"
  top: "ip"
  inner_product_param {
    num_output: 10
  }
}

layers {
  name: "prob"
  type: SOFTMAX
  bottom: "ip"
  top: "prob"
}
```

Formáty, ktoré sa používajú na tento druh popisu zahŕňajú *XML*,[7] *JSON*[8] a *Protocol Buffers*[9]. Ako reprezentácia väčšiny programovacích jazykov, aj tieto formáty sú primárne určené na *uchovávanie obsahu*, nie na priamu interakciu s nimi.

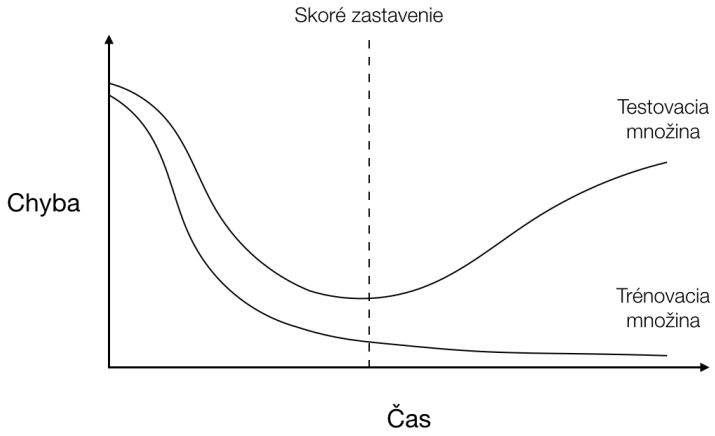
### 3.1.2 Vizualizácia

Väčšina spomínaných nástrojov ponúka možnosť čiastočnej vizualizácie neurónových sietí. Tieto vizualizácie pomáhajú pochopiť správanie sa siete.

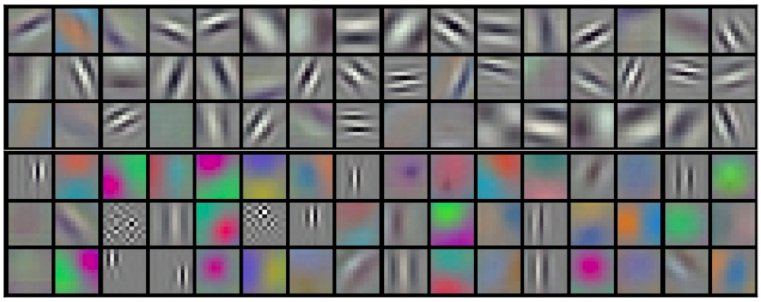
Najpoužívanejšou zobrazovacou technikou je monitorovanie priebehu učenia v čase, viď. obrázok 3.1. Táto jednoduchá vizualizácia nám pomáha zistiť, či sa sieť skutočne učí. Veľkou pomôckou je aj pri rozhodovaní o tom, kedy treba zastaviť trénovanie. Ak by sme zastavili trénovanie neskoro, sieť by mala výborné výsledky na trénovacej množine, avšak na testovacích dátach by pohorela.

Ďalším štandardom vo vizualizácii neurónových sietí je možnosť zobrazenia filtrov, resp. váh neuronálnych spojení. Na obrázku 3.2 sú zobrazené filtre z prvej vrstvy siete Supervision.[10] Podrobným skúmaním filtrov z celej siete sme schopní zistiť o vnútor-

[7] Beran et al., "ViNNSL - the Vienna Neural Network Specification Language."
[8] http://cs.stanford.edu/people/karpathy/convnetjs/
[9] http://caffe.berkeleyvision.org/
[10] Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks."




Obr. 3.1: Ukážka vizualizácie trénovania neurónovej siete



Obr. 3.2: Vizualizácia filtrov z prvej vrstvy siete Supervision

ných reprezentáciách systému omnoho viac.[11]

## 3.2 Navrhované riešenie

Predpokladajme, že existuje lepšie riešenie ako tie, ktoré sme si doteraz popísali. Čím je iné? Čo by malo spĺňať aby mohli neurónové siete používať aj ľudia, ktorí nevedia programovať?

### 3.2.1 Princípy

V nasledujúcej časti si popíšeme princípy, ktoré sú esenciálne pre navrhované riešenie nástroja na interaktívne experimentovanie s neurónovými sieťami.

#### Reprezentácie

Jerome Brunner popísal tri stupne reprezentácií, ktoré používame pri vytváraní si nových koncepcií.[12] Popíšeme si ich na príklade kvadratickej funkcie.

[11] Zeiler and Fergus, "Visualizing and Understanding Convolutional Neural Networks".
[12] Bruner, *Toward a Theory of Instruction*.

**Enaktívne reprezentácie** pozostávajú z akcií, ktoré môžeme vykonávať *priamou manipuláciou* s objektami. Napr. dieťa hrajúce sa s kockami môže kvadratickú funkciu vyjadriť rozmiestnením kociek do štvorcovej formácie.

**Ikonické reprezentácie** vyjadrujú *vizuálne vlastnosti* konkrétnych situácií naučených v enaktívnej fáze. Napr. akýkoľvek obrázok štvorca alebo graf kvadratickej funkcie spadajú do tejto kategórie.

**Symbolické reprezentácie** popisujú abstraktné informácie o naučených konceptoch cez *dohodnuté symboly*. Kvadratická funkcia je symbolicky vyjadrená ako $a \times a$ alebo $a^2$, kde symboly popisujú entity a aj manipulácie s nimi.[13]

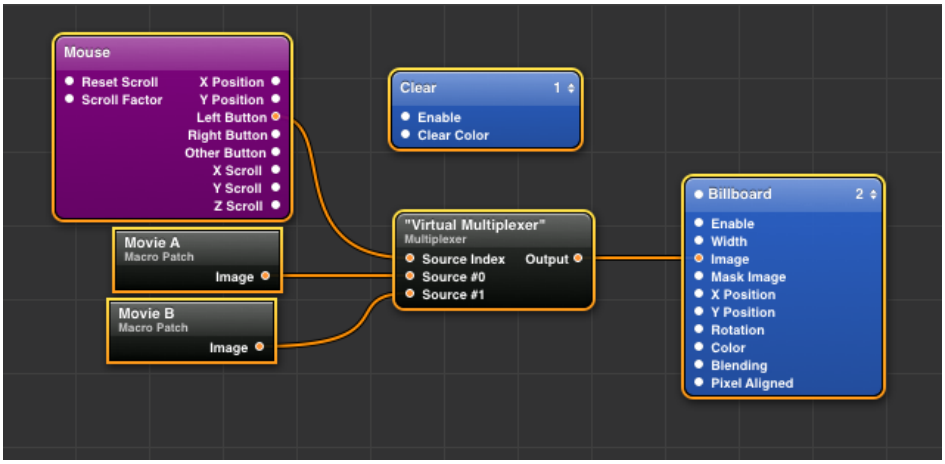
Ktoré reprezentácie treba použiť aby bol zabezpečený čo najlepší prenos a uchovanie informácie? Enaktívne, ikonické alebo symbolické? Ideálne všetky naraz. Použitím všetkých získava človek intuíciu o tom, aké vzťahy existujú medzi jednotlivými časťami systému na rôznych úrovniach opisu.

### Vizuálne programovanie

Ako vyzerá prostredie, ktoré by poskytovalo použitie všetkých troch druhov reprezentácií? Textový editor, kde napíšeme zdrojový kód neurónovej siete, to určite nie je.

*Vizuálne programovanie* (obr. 3.3) sa ponúka ako vhodná paradigma, ktorá je používaná s väčšou, či menšou úspešnosťou v rôznych oblastiach návrhu komplexných systémov.

Paradigma vizuálneho programovania používa *plátno*, na ktorom sú umiestnené *objekty*. Každý objekt obsahuje vstupy, ktoré transformuje na výstupy. Tieto výstupné hodnoty sú potom použité ako vstupné hodnoty do ďalšej transformácie.




Obr. 3.3: Ukážka vizuálneho programovania

[13] Zaujímavosťou je fakt, že symbolické reprezentácie môžu vyjadrovať aj inštancie situácií, ktoré nemôžeme poňať v enaktívnej fáze, napr. „štvorec so zápornou dĺžkou strany“.

Každý objekt predstavuje transformáciu dát, ktoré prúdia po spojeniach medzi nimi. Táto vizuálna forma je teda veľmi vhodná na zobrazovanie architektúry neurónových sietí a poslúži nám ako základ pre ďalšie úpravy.

#### Dáta ≫ Model

Veľmi dôležitá skutočnosť, na ktorú sa často zabúda je, že *dáta sú dôležitejšie ako model*. Aj ten najlepší model na svete je bezvýznamný bez dát, ktoré by mohol modelovať.

Problémom vizuálneho programovania je absencia akýchkoľvek dát. Objekty na ploche reprezentujú iba transformácie, avšak samotné dáta nie sú zobrazené. Vizuálna paradigma je týmto redukovaná na slepú manipuláciu so symbolmi, čo silne obmedzuje použitie tohto prístupu.

Najjednoduchším riešením tohto obmedzenia je *zobrazenie transformovaných dát*. Ak zobrazíme dáta v každom kroku transformácie, používateľovi pomáhame pochopiť jednotlivé časti systému. Experimentovaním si používateľ osvojí časti systému, čím sa mu otvára možnosť k pochopeniu neurónovej siete ako celku.[14]

### 3.2.2 Popis riešenia

#### Problém

Zaveďme si ilustratívny problém, ktorý chceme vyriešiť. V oblasti strojového učenia to zvyčajne býva *MNIST* – problém rozpoznávania rukou písaných číslic.[15] Tento dataset je relatívne malý a pozostáva zo 60 000 trénovacích a 10 000 testovacích príkladov.

Každý jeden vstupný vzor je $28 \times 28$ pixelový obrázok v odtieňoch šedej. Požadovaným výstupom má byť zaradenie obrázku do správnej kategórie. Vybrané ukážky z dátovej množiny je možno vidieť na obrázku 3.4.

Obr. 3.4: Ukážka príkladov z MNIST

[14] Victor, *Media for Thinking the Unthinkable*.
[15] LeCun et al., "Gradient-based learning applied to document recognition".

### Riešenie

Problém, ktorý chceme vyriešiť, je tvorený zo 70 tisíc menších problémov. Redukujme si teda náš problém a teda aj dátovú množinu na jeden príklad, na ktorý sa budeme sústrediť.

Vložíme na plochu jeden obrázok, ktorý prostredie automaticky rozpozná ako dátový zdroj:


Obr. 3.5: Dátový zdroj

Tento objekt má tri vlastnosti, ktoré definujú jeho existenciu na ploche. Prvou vlastnosťou, ktorú môžeme vidieť, je jeho *vizuál*. Objekt zapuzdruje vnútorné dáta a farbou vyjadruje svoj typ.

Ďalšou vlastnosťou je *symbolický popis*. Okrem prednastaveného mena máme k dispozícii aj textový opis dát, ktorý stručne charakterizuje zobrazené dáta.

Poslednou vlastnosťou je *priama manipulácia*. Objekt môžeme ťahaním premiestňovať po ploche, čím mu môžeme nastaviť polohu, ktorá nám vyhovuje. Rovnako mu môžeme zmeniť aj jeho ďalšie vlastnosti.

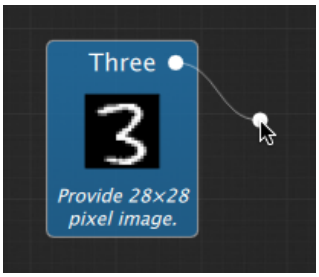
Priama manipulácia s objektom zahŕňa aj zmenu prednastaveného mena:


Obr. 3.6: Premenovanie dátového zdroja

Premenovanie objektu nám umožňuje priradiť mu deskriptor, ktorý viac vyhovuje našim požiadavkám. V tomto prípade sme premenovali dátový zdroj na meno vyjadrujúce popis dát, ktoré ponúka, teda číslo „tri“.

Nenápadný kruh na pravej strane komponentu predstavuje výstupnú hodnotu objektu. V tomto prípade sa jedná o vnútorné dáta, ktoré objekt poskytuje, t.j. obrázok čísla tri.

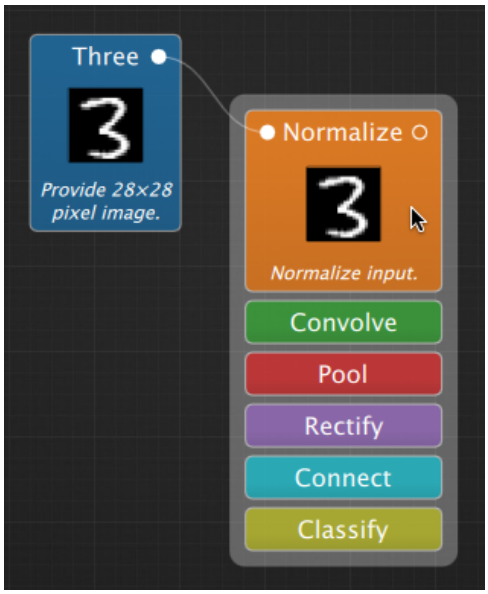
Aby sme mohli použiť tieto dáta v nejakej transformácii, musíme tento komponent spojiť s iným komponentom. V súlade s koncepciou priamej manipulácie je tento kruh *ťahateľný*. Ťahaním sa nám zobrazí *konektor*, ktorý spája objekt s voľným uzlom, ktorý pripneme k inému objektu:




Obr. 3.7: Ťahanie výstupného konektoru dátového zdroja

Nastal však problém. Na ploche existuje iba jeden objekt a ten chceme na nejaký napojiť – nemáme žiadne ďalšie objekty, ktoré by sme mohli použiť. Potrebujeme vytvoriť nový objekt, ktorý použijeme.

Konektor pustíme na prázdnu plochu. Zobrazí sa nám *ponuka*, z ktorej si môžeme vybrať komponent, ktorý nám vyhovuje:




Obr. 3.8: Ponuka transformácií s okamžitým náhľadom

Ponuka obsahuje všetky transformácie, ktoré môžeme pridať na plochu. Vidíme, že názvy transformácií používajú *slovesný tvar*. To zdôrazňuje fakt, že komponenty vykonávajú istú *funckiu*. Tento prístup je vhodnejší ako názov, ktorý implikuje iba pasívnu úlohu, napr. „normalizácia“ verzus „normalizovať“ alebo „normalizuj“.

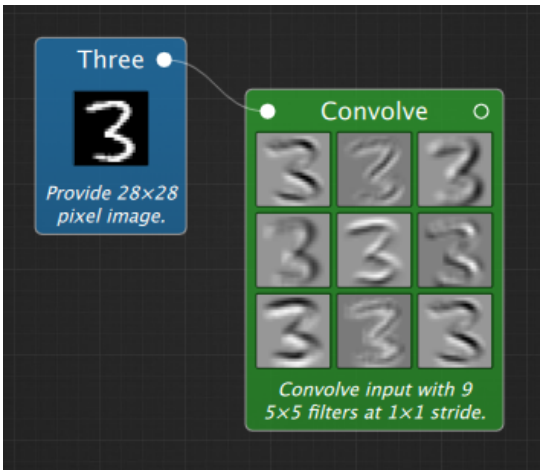
Dôležitou vlastnosťou ponuky komponentov je *okamžitý náhľad*. Ten môžeme vidieť na príklade normalizačnej transformácie. Vďaka vizuálnemu zobrazeniu a textovému

popisu vieme, čo máme očakávať od danej transformácie. Vďaka tejto vlastnosti sa môžeme vyhnúť zbytočnej manipulácii s objektami na ploche.

Máme predvolenú normalizačnú vrstvu, ktorá nám upraví rozsah vstupných hodnôt na požadované rozpätie. Túto úlohu zvyčajne vykonáva *vstupná vrstva* siete, avšak názov „vstupná vrstva“ nám nepovie nič o jej funkcii.

Zdá sa však, že výstup z dátového zdroja a výstup z normalizačnej transformácie sú úplne rovnaké – akoby transformácia nemala žiaden vplyv na dáta. A to je aj pravda. Dáta, ktoré používame, sú poskytnuté v normalizovanej forme – ďalšia normalizácia nebude mať žiaden efekt. Okamžitý náhľad nám pomohol zbaviť sa nepotrebnej transformácie ešte predtým ako sme ju vôbec použili.

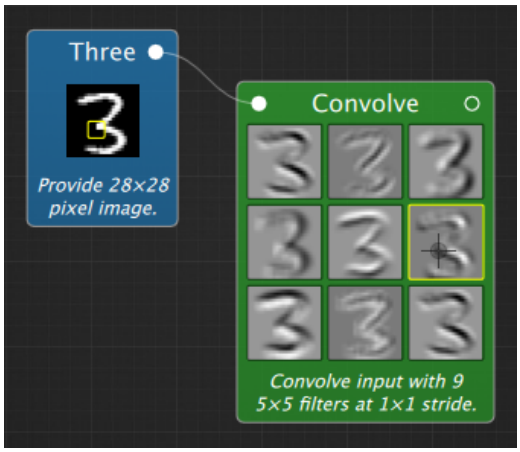
Zvoľme teda zaujímavejšiu vrstvu, ktorá nám pomôže pri našom probléme rozpoznávania číslic. Konvolučná vrstva je vhodnou voľbou:




Obr. 3.9: Konvolučná vrstva

Konvolučná vrstva produkuje viacero výstupov a každý jeden z nich môžeme *vidieť*. Vďaka textovému popisu vieme *povedať* čo daná transformácia vykonáva. Avšak až vďaka priamej manipulácii si môžeme začať budovať koncepcie o tom ako táto transformácia skutočne funguje.

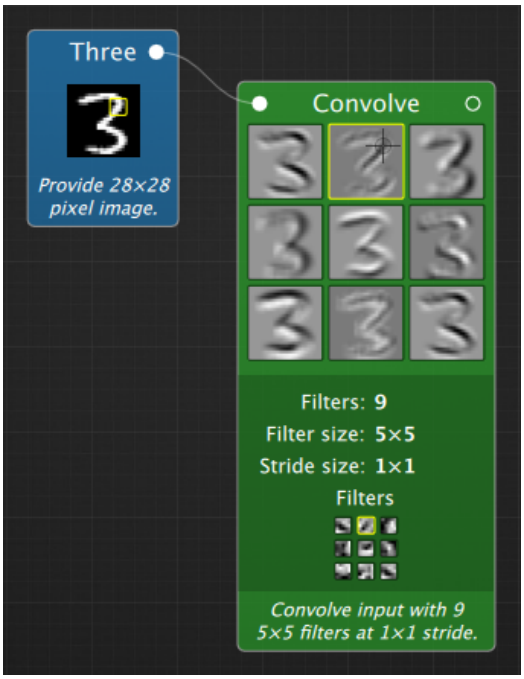
Pri prechádzaní ponad výstup konvolučnej vrstvy sa nám na dátovej vrstve zobrazí výrez obrázku, ktorý bol použitý na vypočítanie hodnoty danej aktivácie:




Obr. 3.10: Spojené reprezentácie

Z tejto interakcie je nám jasné, že každá jedna aktivácia na konvolučnej vrstve je asociovaná s istou časťou zdrojového obrázka. Avšak prečo transformácia produkuje 9 výstupov? Na túto otázku nám čiastočne odpovedá textový popis, v ktorom sa spomína „9 filtrov veľkosti $5 \times 5$“. Vidíme, že počet výstupov bude závislý od počtu filtrov, ktoré použijeme. Ale čo sú filtre? Navyše odkiaľ sa zobrali takéto parametre? Žiaden z nich sme nenastavovali.

Kliknutím na textový opis odkryjeme dôležitú časť tejto transformácie – skryté parametre:

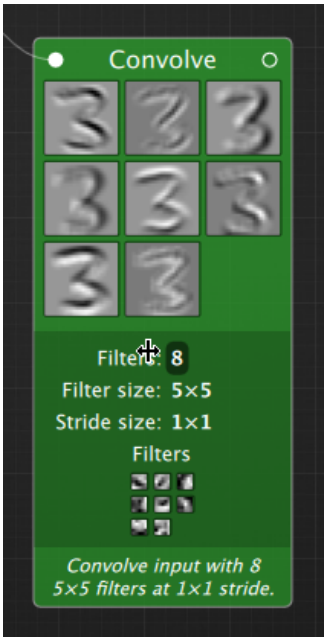



Obr. 3.11: Nastaviteľné parametre konvolučnej vrstvy

Prvé čo si všimneme sú vizuálne reprezentácie filtrov, ktorých je skutočne deväť. Ďalej vidíme tri parametre, ktoré ovplyvňujú správanie transformácie. Tieto parametre

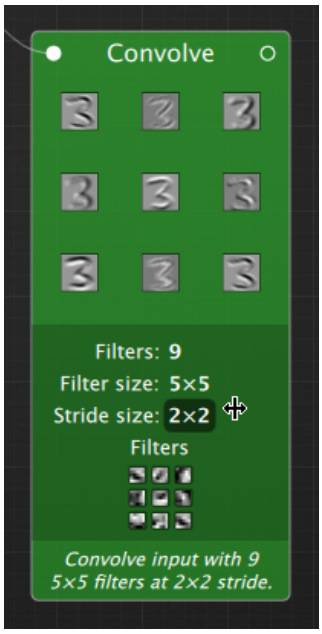
sú meniteľné ťahaním – chytíme číselný parameter a potiahnutím doľava alebo doprava meníme jeho hodnotu. Všetky zmeny sa prejavia okamžite, čím sa pocit slepej manipulácie so symbolmi úplne rozplynie.

Ukážme si takúto zmenu na príklade počtu filtrov. Miernym potiahnutím doľava znížime hodnotu parametra z 9 na 8. Táto zmena sa prejaví na ďalších troch rôznych miestach – počet výstupov transformácie, počet filtrov a nakoniec aj v textovom popise:




Obr. 3.12: Zmena počtu filtrov konvolučnej vrstvy

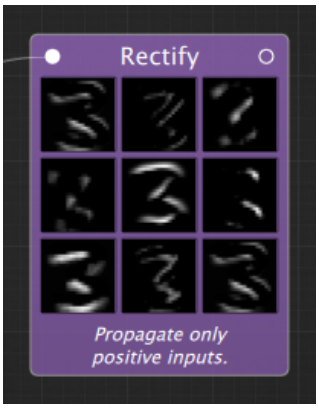
Táto zmena je celkom zjavná a očakávaná – počet filtrov je vo vzájomnom vzťahu s počtom výstupov. Čo však ďalšie parametre? Skúsme zmeniť veľkosť kroku z hodnoty $1 \times 1$ na $2 \times 2$. Táto zmena má celkom neočakávané správanie:




Obr. 3.13: Zmena kroku konvolučnej operácie

Veľkosti jednotlivých výstupov sa zmenšili na polovicu. Po vyskúšaní niekoľkých ďalších hodnôt zistíme, že veľkosť kroku konvolúcie funguje ako škálovací operátor, ktorý redukuje veľkosť výstupov. Symbol „krok konvolúcie“ žiadne takéto správanie neobjasňuje. Interaktívna manipulácia parametrov nám pomohla rýchlejšie pochopiť skryté súvislosti, ktoré sa len ťažko vysvetľujú matematickým formalizmami, či zdrojovým kódom nejakého počítačového programu.

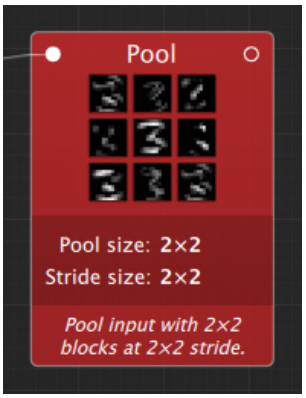
Pridajme ďalšiu transformáciu, ktorá bude spracúvať výstup z konvolučnej vrstvy:




Obr. 3.14: Jednoduchá transformácia

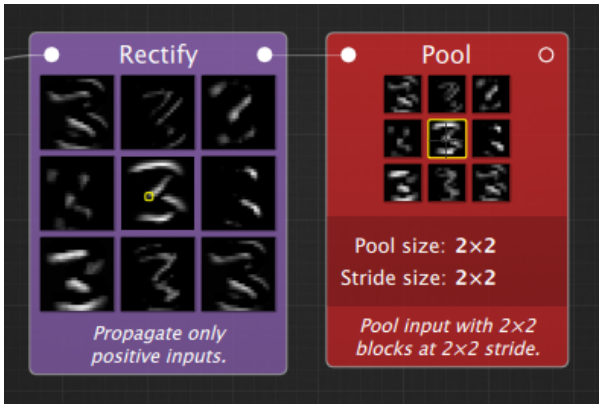
Transformácia rectify funguje ako prechodová funkcia neurónu. Táto inštancia prechodovej funkcie ignoruje záporné aktivácie z predchádzajúcej vrstvy a posiela ďalej iba kladné hodnoty. Keďže táto transformácia nemá žiadne nastaviteľné parametre dá sa jej venovať iba málo.

Pridajme pooling vrstvu, ktorá združuje aktivácie z predchádzajúcej vrstvy a berie z nich iba maximálne hodnoty, tzv. *max-pooling* vrstva:



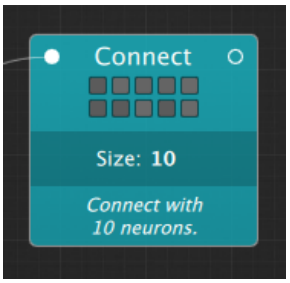

Obr. 3.15: Max-pooling vrstva

Táto transformácia je veľmi podobná konvolučnej vrstve, avšak akoby jej funkcia bola o niečo obmedzená. Pooling vrstva nepoužíva žiadne filtre a zvyšné dva parametre sa správajú rovnako ako parametre konvolučnej vrstvy, čo môžeme vidieť na príklade spojených reprezentácií:



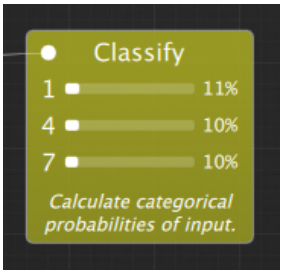

Obr. 3.16: Spojené reprezentácie na max-pooling vrstve

Zdá sa, že pooling vrstva iba zmenšuje výstup z rectify vrstvy. Možno teda vôbec nezáleží na ich poradí... Avšak skúsme pokračovať pridaním plne prepojenej vrstvy. Tej nastavíme veľkosť na desať, teda počet tried, do ktorých chceme zaradiť naše dáta:




Obr. 3.17: Úplne prepojená vrstva

Vidíme, že aktivácie na tejto vrstve sú takmer totožné a všetky sa držia približne v strednej hodnote. Avšak táto vrstva je iba medzistupňom k tomu aby sme skutočne videli to, čo si neurónová sieť *myslí* o danom vstupe. Pridajme teda poslednú vrstvu, ktorá nám túto informáciu sprostredkuje:




Obr. 3.18: Klasifikačná vrstva

Klasifikačná vrstva nám zobrazuje tri najpravdepodobnejšie tipy vrátane miery istoty, ktorá je vyjadrená vizuálne a aj číselne. Všetky triedy majú mieru istoty okolo 10%, čo nám hovorí, že neurónová sieť si svojimi tipmi nie je vôbec istá. Toto je dobré znamenie, pretože neurónová sieť ešte nevie nič o rozpoznávaní číslic, takže by nemala preferovať žiadnu triedu.

Posledná vec, ktorá nám ostáva urobiť, je natrénovať sieť, aby sa naučila rozpoznávať číslice.

*Avšak to je už iný príbeh...*

## 3.3 Budúce riešenia

Prístup k vytváraniu a chápaniu komplexných systémov, založený na princípe prepojených akcií, vizuálov a symbolov sa javí ako správna cesta. Vizuálne programovanie doplnené o zobrazenie dát je teda veľmi vhodná paradigma, ktorá nám ponúka široké možnosti pri implementácií neurónových sietí.

Keď si človek zvykne na priestorové rozmiestnenie transformácií na ploche, rýchlo prídu myšlienky, ktoré využívajú tento štýl rozmýšľania naplno. Môžeme zabudnúť na

lineárne usporiadanie transformácií, do ktorých sme nútený pri používaní lineárneho média ako je textový editor. Výpočet sa môže vetviť v rôznych bodoch architektúry, čo nám umožní porovnať vetvy, ktoré sa líšia v nastavených parametroch a vybrať tú lepšiu, prípadne ich môžeme nechať súperiť.

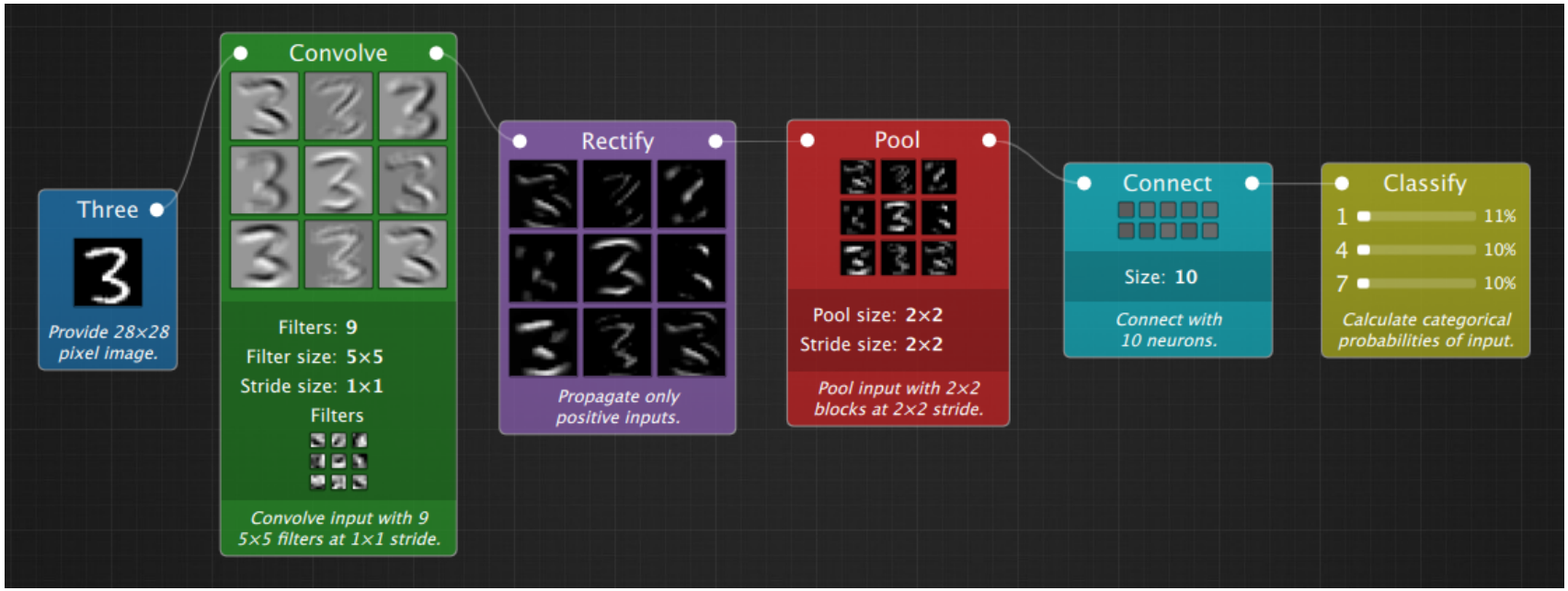
S touto reprezentáciou môžeme experimentovať aj s rôznymi dátovými zdrojmi. Napríklad dátový zdroj, ktorý vie prijímať dáta z používateľského vstupu, napr. z kamery, nám umožňuje vidieť výpočet v reálnom čase. Alebo napríklad pri rozpoznávaní číslic, môžeme vyskúšať sieť na našom vlastnom písme. Prípadne môžeme urobiť kombináciu nejakých dát a výstupu neurónovej siete ako dátový zdroj pre ďalšiu sieť – týmto by sme simulovali sprostredkované učenie neurónových sietí, kde by materská sieť učila dcérsku.

Spomínané princípy môžu byť aplikované aj na ďalšiu dimenziu – čas. Ak by sme mohli manipulovať s časom, otvorili by sa nám ďalšie možnosti, ktoré by znásobovali použitie takéhoto prostredia. Triviálnym príkladom je zobrazovanie neuronálnych váh v čase. Praktickým využitím by mohla byť manipulácia parametrov transformácií v čase a ich následné porovnanie.

### Rebrík

Existuje mnoho smerov, ktorými by sa mohla táto myšlienka uberať. Avšak predstavené prostredie má defekt, ktorý zabraňuje jeho ďalšiemu použitiu. Týmto problémom je zastaralé médium, ktoré používa. Človek potrebuje oveľa užšie prepojenie so svojím výtvorom ako mu to použitý typ počítacieho stroja umožňuje.

Tento výtvor svojím vznikom poprel dôvod svojej podstaty, a stal sa z neho iba užitočný nezmysel, ktorý treba teraz odhodiť, lebo už nie je viac potrebný...




Obr. 3.19: Užitočný nezmysel

# Záver

Cieľom výskumu v oblasti umelej inteligencie, je vytvoriť umelú myseľ. Táto myseľ sa musí správať ako dokonalá čierna skrinka – žiadne nastaviteľné parametre, žiadne ladenie a žiadne vnútorné zásahy. Skutočne samostatný *mysliaci stroj*.

Aby sme takýto stroj mohli vytvoriť, potrebujeme nástroje, ktoré nám pomôžu vytvárať a pochopiť systémy založené na umelých neurónových sieťach. Takéto prostredie nám musí ponúkať tri druhy reprezentácií – enaktívne, ikonické a symbolické. Tieto reprezentácie musia byť vzájomne prepojené, aby poskytovali čo najbohatší obraz o vytváranom systéme.

Vizuálne programovanie doplnené o zobrazovanie spracovávaných dát je prostredie, ktoré nám umožní široké spektrum experiementovania s neurónovými sieťami. Vďaka takémuto prostrediu budeme môcť vidieť, čo neurónová sieť robí a ako sa správa. Nebudeme si ju musieť simulovať v hlave na inej neurónovej sieti.

Na to aby sme vytvorili skutočnú čiernu skrinku, najskôr potrebujeme do nej jasne vidieť. Až potom ju môžeme natrieť na čierno.

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> Alan Turing

# Literatúra


Bengio, Yoshua, P Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *Neural Networks, IEEE* (1994). URL: http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf.

Beran, Peter Paul et al. "ViNNSL - the Vienna Neural Network Specification Language." In: *IJCNN*. IEEE, 2008, pp. 1872–1879. URL: http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2008.html%5C#BeranVSW08.

Bruner, J S. *Toward a Theory of Instruction*. Books that live. Belknap Press of Harvard University Press, 1966. URL: https://encrypted.google.com/books?id=xR44nQEACAAJ.

Cazé, Romain Daniel, Mark Humphries, and Boris Gutkin. "Passive dendrites enable single neurons to compute linearly non-separable functions." In: *PLoS computational biology* 9.2 (Jan. 2013), e1002867. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1002867. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3585427/.

Cybenko, George. "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of control, signals and systems* (1989), pp. 303–314. URL: http://deeplearning.cs.cmu.edu/pdfs/Cybenko.pdf.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP* 15 (2011), pp. 315–323. URL: http://eprints.pascal-network.org/archive/00008596/01/glorot11a.pdf.

Hinton, Geoffrey E. et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv: ...* (2012), pp. 1–18. arXiv: arXiv:1207.0580v1. URL: http://arxiv.org/abs/1207.0580.

Hobbes, Thomas. *Leviathan neboli látka, forma a moc státu církevního a občanského*. Knihovna n. Praha: OIKOYMENH, 2009, 513 s.

Hubel, DH and TN Wiesel. "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* (1968), pp. 215–243. URL: http://jp.physoc.org/content/195/1/215.short.

Johnson, Jeff. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN: 012375030X, 9780123750303.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *NIPS* (2012), pp. 1–9. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

LeCun, Yann et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* (1998). URL: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *Proceedings of the ICML* 28 (2013). URL: http://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

McCulloch, WS and W Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133. URL: http://link.springer.com/article/10.1007/BF02478259.

Nair, Vinod and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *International Conference on Machine Learning (ICML-10)* 3 (2010). URL: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf.

Petrů, Marek. *Fyziologie mysli: Úvod do kognitivní vědy*. Filosofie. Praha: Triton, 2007, p. 385. ISBN: 978-80-7254-969-6.

Rojas, Raul. *Neural Networks: A Systematic Introduction*. Berlin: Springer-Verlag, 1996. URL: http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/pmwiki/pmwiki.php?n=Books.NeuralNetworksBook.

Rumelhart, DE, Geoffrey E. Hinton, and RJ Williams. *Learning representations by back-propagating errors*. 1988. URL: http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf.

Schmidhuber, Juergen. "Deep Learning in Neural Networks: An Overview". In: (Apr. 2014), p. 66. arXiv: 1404.7828. URL: http://arxiv.org/abs/1404.7828.

Turing, Alan M. "Intelligent Machinery". In: *Machine Intelligence*. Ed. by Bernard Meltzer and Donald Michie. Vol. 5. Edinburgh, UK: Edinburgh University Press, 1969. Chap. 1, pp. 3–23.

— "On Computable Numbers with an Application to the Entscheidungsproblem". In: *J. of Math 58 (1936): 345-363.* 58 (1936), pp. 345–363. URL: http://classes.soe.ucsc.edu/cmps210/Winter11/Papers/turing-1936.pdf.

Victor, Bret. *Media for Thinking the Unthinkable*. 2013. URL: http://worrydream.com/MediaForThinkingTheUnthinkable/.

Zeiler, MD and Rob Fergus. "Visualizing and Understanding Convolutional Neural Networks". In: *arXiv preprint arXiv:1311.2901* (2013). arXiv: arXiv:1311.2901v3. URL: http://arxiv.org/abs/1311.2901.

# Príloha

Túto publikáciu, zdrojové kódy prototypu a dodatočné materiály je možné nájsť na webovej adrese http://github.com/mlajtos/MasterThesis.



Tento digitálny nosič je prázdny. Slúži iba ako pripomienka, že všetky médiá majú obmedzenú životnosť.