



**Gymnázium**  
**České Budějovice**  
**Jírovцова 8**

## **MATURITNÍ PRÁCE**

**Neuronová síť**

**Jonáš Havelka**

**vedoucí práce: Dr. rer. nat. Michal Kočer**


# Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně s vyznačením všech použitých pramenů.


V Českých Budějovicích dne ..... podpis .....

Jonáš Havelka

# Abstrakt





Neuronové sítě se dnes objevují všude, ať už jde o vyhledávání, překládání nebo třeba jen zpracovávání dat. Mnoho programovacích jazyků má své knihovny pro práci s umělou inteligencí, ale zrovna  Kotlin, který je mým oblíbeným programovacím jazykem a lze použít skoro kdekoliv (webové stránky, servery, mobily), takovou knihovnu postrádá. Proto jsem se rozhodl svoji práci koncipovat jako snahu o implementování takové knihovny.

## Klíčová slova

Neuronové sítě,  Kotlin, Umělá inteligence, Multiplatformní knihovna

# Poděkování

Poděkování patří hlavně mému učiteli informatiky, který je zároveň vedoucím mé práce, za skvělou výuku na hodinách a velkou trpělivost při kontrole našich prací. Také nesmím zapomenout na Alžbětu Neubauerovou, která mě celý rok podporovala a několikrát provedla korekturu mé práce.

Dále bych rád poděkoval všem komunitám, jejichž nástroje jsem používal, tj. JetBrains, v jejichž programovacím jazyce  Kotlin programuji a jejichž prostředí IntelliJ k tomu využívám,  Gradle, který používám ke kompilaci,  $\LaTeX$ , ve kterém píšu text a dále  git a  GitHub, jež uchovávají má data, ať už text nebo knihovnu.

# Obsah

<b>I</b>	<b>Teoretická část</b>	<b>8</b>
<b>1</b>	<b>Strojové učení a umělá inteligence</b>	<b>9</b>
<b>2</b>	<b>Neuronové sítě</b>	<b>10</b>
2.1	Laický náhled . . . . .	10
2.1.1	Neuron . . . . .	10
2.1.2	Sítě . . . . .	10
2.1.3	Dopředná propagace a zpětná propagace . . . . .	11
2.2	Formální náhled . . . . .	11
<b>II</b>	<b>Praktická část</b>	<b>13</b>
<b>3</b>	<b>Struktura knihovny</b>	<b>14</b>
3.1	core . . . . .	14
3.1.1	ActivationFunctions . . . . .	14
3.1.2	INeuralNetwork . . . . .	15
3.1.3	BasicNeuralNetwork . . . . .	15
3.1.4	ConvolutionalNetwork . . . . .	16
3.2	mnistDatabase . . . . .	16
3.2.1	Databáze MNIST . . . . .	16
3.2.2	Databáze EMNIST . . . . .	16
<b>4</b>	<b>Používání knihovny</b>	<b>18</b>
	<b>Apendix</b>	<b>18</b>
	<b>Bibliografie</b>	<b>20</b>

<b>Slovníček pojmů</b>	<b>21</b>
<b>Přílohy</b>	<b>24</b>
Fotky z pokusů . . . . .	24
Vlastní program . . . . .	CD
Dokumentace . . . . .	CD
Testovací data . . . . .	CD

# Úvod

Neuronové sítě jsou v poslední době velmi skloňované téma. Nikdo vlastně pořádně neví, jak fungují, ale fungují<sup>1</sup>. Cílem této práce však nebude zkoumat neuronové sítě, ale implementovat je v co největším rozsahu (ať už struktury bez širšího využití jako asociativní paměť nebo často používané konvoluční sítě na rozpoznávání obrázků).

Kotlin je ideální programovací jazyk pro vývoj knihovny, protože umožňuje tuto knihovnu používat jak pro JVM, tak i v prohlížeči nebo v programech kompilovaných přímo do binárního kódu.

Celá maturitní práce je k dispozici na GitHubu, text včetně zdrojového LaTeXu na adrese [https://github.com/JoHavel/Maturitni-Seminarni-Prace/tree/my\\_work](https://github.com/JoHavel/Maturitni-Seminarni-Prace/tree/my_work) a knihovna samotná pak na <https://github.com/JoHavel/NeuralNetwork>.

---

<sup>1</sup>Tady se hodí podotknout, že alespoň malou představu máme, přece jenom matematicky je to pouze sestup po gradientu, ale překvapivě to dokáže velmi mnoho

# Část I

## Teoretická část



# Kapitola 1

## Strojové učení a umělá inteligence

## Kapitola 2

### Neuronové sítě

#### 2.1 Laický náhled

##### 2.1.1 Neuron

Počítačové neuronové sítě nejsou jen výmysl lidí, jejich základ nalezneme v nervových soustavách živočichů. Základní stavební jednotka takové soustavy (stejně tak i neuronové sítě) je neuron. Neuron funguje tak, že přes dendrity přijímá elektrické (přesněji iontové) signály od jiných neuronů a když součet signálů přeteče určitou danou mez, vyšle neuron signál přes axony dál do dalších neuronů.

Přenos signálu z axonu do dendritu se odehrává v malých prostorách mezi nimi zvaných synapse. Vodivost synapsí je ovlivněna jejich chemickým složením, a proto se domníváme, že proces učení probíhá měněním těchto chemických spojů (Brookshear et al. 2013, s. 491).

Náš umělý neuron tedy bude mít seznam dendritů (nesoucích informaci z jakého neuronu vedou signál a jak ho mění synapse), tzv. aktivační funkci (tedy jak silný signál posílá dále v závislosti na součtu vstupních signálů) a výstupní signál. Často navíc bude obsahovat základní hodnotu (angl. bias), která reprezentuje hladinu iontů v neuronu.

##### 2.1.2 Sítě

Jelikož nahodilé neurony by se těžko udržovaly v paměti a operace na nich by byly velmi pomalé, potřebujeme síť nějak uspořádat. Nejjednodušším uspořádáním jsou vrstvy. Každý neuron z nějaké vrstvy má dendrity ze všech neuronů z vrstvy minulé. Tak se předejde cyklům, které jsou složité na výpočty, a navíc si nemusíme u každého neuronu pamatovat, ze kterých neuronů do něj vede signál.

### 2.1.3 Dopředná propagace a zpětná propagace

Dopředná propagace (častěji se používá anglický výraz forward propagation) je jednoduše spočítání signálů ve všech neuronech. Tedy u každého neuronu se sečtou vstupní signály (popř. přičte bias) a spočítá se funkční hodnota aktivační funkce v tomto bodě.

Naopak zpětná propagace (častěji se používá anglický výraz backward propagation či backpropagation) je na základě chyby, kterou spočítáme z výstupu neuronové sítě a předpokládaného výstupu, určit, které proměnné hodnoty (synapse a biasy) se na ní nejvíce podílejí. Potom tyto hodnoty posuneme odpovídajícím způsobem (stejně jako příroda mění chemické vlastnosti synapse)

## 2.2 Formální náhled

Označme  $\nu = (N, W, F)$  neuronovou síť,  $N$  je množina všech jejích neuronů,  $W: N \times N \rightarrow \mathbb{R}$  jsou váhy (angl. weights) udávající sílu synapse mezi dvěma neurony (v případě, že mezi neurony synapse není, je  $W$  rovno 0) a  $F: \mathbb{R}^{|N_v|} \rightarrow \mathbb{R}$  je chybová funkce udávající velikost chyby podle rozdílu reálných hodnot od chtěných hodnot výstupních neuronů ( $N_v$ ).

Nechť  $n \in N$ ,  $n = (N_{in}, N_{out}, f, b, v, \varepsilon)$  je neuron, kde  $N_{in} = \{n_x \in N | W(n_x, n) \neq 0\}$  je množina neuronů, které vysílají signál do  $n$ ,  $N_{out} = \{n_x \in N | W(n, n_x) \neq 0\}$  je množina neuronů, které přijímají signál od  $n$ ,  $f: \mathbb{R} \rightarrow \mathbb{R}$  je aktivační funkce,  $b \in \mathbb{R}$  je bias,  $v \in \mathbb{R}$  je signál vycházející z  $n$  a  $\varepsilon$  je chyba (parciální derivace chybové funkce podle  $f^{-1}(v)$ <sup>1</sup>). Potom dopředná propagace (tedy spočítání  $v$ ) vypadá takto:

$$v = f \left( b + \sum_{n_x \in N_{in}, v_x \in n_x} v_x \cdot W(n_x, n) \right)$$

To lze při označení

$$\vec{v} = (v_1, v_2, \dots)$$

$$\vec{w} = (w_1, w_2, \dots)$$

$$(\forall n_x \in N_{in}) (\exists i \in \mathbb{N}) (v_i \in n_x \wedge w_i = W(n_x, n))$$

zapsat vektorově jako:

$$v = f(b + \vec{w} \cdot \vec{v})$$

Případně můžeme do vektorů „zakomponovat“ i bias<sup>2</sup>:

$$\vec{v} = (1, v_1, v_2, \dots)$$

---

<sup>1</sup>Derivace aktivačních funkcí se často snadno spočítá z funkční hodnoty, proto uvádím, že hledám derivaci v bodě, kde je daná funkční hodnota, značím přitom  $f^{-1}(y) = x \Leftrightarrow f(x) = y$

<sup>2</sup>To jsem v knihovně nepoužil z důvodu netriviálního přidávání prvku do vektoru.

$$\vec{w} = (b, w_1, w_2, \dots)$$

$$(\forall n_x \in N_{in}) (\exists i \in \mathbb{N}) (v_i \in n_x \wedge w_i = W(n_x, n))$$

$$v = f(\vec{w} \cdot \vec{v})$$

Při zpětné propagaci je důležitý vzorec pro derivaci složené funkce, někdy také znám jako „řetízkové pravidlo“:

$$\begin{aligned} \frac{dy}{dx} &= \frac{dz}{dx} \frac{dy}{dz} \\ \frac{\delta y}{\delta x} &= \sum_z \frac{\delta z}{\delta x} \frac{\delta y}{\delta z} \end{aligned}$$

Pro jednoduchost předpokládejme, že neurony jsou nezávislé. Potom můžeme  $\varepsilon$  spočítat jako součet derivací  $\varepsilon$  ostatních neuronů podle  $f^{-1}(v)$

$$\varepsilon = \frac{\sum_{\varepsilon \in N_{out}, \varepsilon_x \in n_x} d\varepsilon_x}{df^{-1}(v)} = f'(f^{-1}(v)) \left( \sum_{\varepsilon \in N_{out}, \varepsilon_x \in n_x} \varepsilon \cdot W(n, n_x) \right)$$

Obdobně jako v předchozím případě definujeme vektory

$$\varepsilon =$$

## Část II

### Praktická část

# Kapitola 3

## Struktura knihovny

Knihovna je rozdělena do dvou částí:

- První, a ta hlavní, je `core` (česky jádro), které obsahuje definice neuronových sítí (tj. konvoluční neuronovou síť, obyčejnou neuronovou síť, asociativní paměť) a definice pro ně potřebné (například aktivační funkce).
- Druhá je `mnistDatabase`, která se stará o učení neuronových sítí na datových databázích.

### 3.1 core

#### 3.1.1 ActivationFunctions

Enumerate (česky výčet) funkcí, jež se používají jako aktivační funkce v neuronech:

- `TODO`(funkce s odkazy)

Funkce lze zavolat s parametrem typu `Double`, což nám dá hodnotu funkce v tomto bodě, popřípadě lze obdobně zavolat jejich 2 metody `xD` a `yD` udávající v pořadí hodnotu derivace v bodě `x` a v bodě, kde je funkční hodnota rovna parametru.<sup>1</sup>

Některé jsou označeny jako překonané (anglicky deprecated), jelikož u funkcí, které nejsou všude hladké, neexistuje všude derivace. Taktéž u funkcí, jež nejsou prosté, nelze vždy určit derivaci podle funkční hodnoty.

---

<sup>1</sup>Druhá hodnota se používá, jelikož při zpětné propagaci máme k dispozici i funkční hodnotu, a derivace se u těchto funkcí často jednodušeji spočítá právě z této hodnoty.

### 3.1.2 INeuralNetwork

Rozhraní (anglicky interface), které implementuje základní funkce neuronových sítí, které mají jako vstup i výstup `Double` (typ pro přesnější desetinné číslo) vektor. Obsahuje funkce:

- `run(vstupní vektor)`, která je koncipována tak, aby ze vstupního vektoru spočítala vektor výstupní (tedy většinou udělala dopřednou propagaci). Jako vstupní vektor lze dát jak `Matrix<Double>` z knihovny `koma`, tak `DoubleArray`, které je převedeno na `Matrix<Double>`, následně se zavolá funkce `run` s tímto typem a výstup se převede zpět na `DoubleArray`.<sup>2</sup>

Navíc (hlavně kvůli konvolučním neuronovým sítím) může být vstup i dvourozměrný, v tomto případě je pak nutno u `DoubleArray` uvést i šířku řádku.

- `train(vstupní vektor, chtěný výstupní vektor)` resp. `train(vstupní vektory, chtěný výstupní vektor)`, která je koncipována tak, aby nejdříve provedla `run(vstupní vektor)`, výsledek porovnála s chtěným a přepočítala váhy v neuronové síti tak, aby se výstup `run(vstupní vektor)` přiblížil chtěnému výstupnímu vektoru. Kromě verze s parametry typu `DoubleArray` je funkce implementována i pro typ `Array<DoubleArray>`, tedy trénovací vstupy a výstupy lze vložit i všechny najednou.

### 3.1.3 BasicNeuralNetwork

Tato třída rozhraní `INeuralNetwork` implementuje nejčastěji používanou neuronovou síť, kde neurony jsou uspořádány do vrstev a ovlivňují se pouze jedním směrem. Parametry, které lze nastavit, jsou:

- `numberOfHiddenLayers`, neboli počet skrytých vrstev (tj. ty, jež jsou mezi vstupní a výstupní vrstvou). Čím více vrstev je nastaveno, tím hůře se síť učí, většinou je proto třeba nastavit pouze jednu skrytou vrstvu nebo nastavit velmi malou hodnotu `learning rate` (proměnná, jež není v konstruktoru, udává rychlost změn vah).
- `activationFunctions`, česky aktivační funkce, musí být vybrána z třídy `ActivationFunction`. Při použití funkcí, které nejsou hladké, se neurony mohou chovat nepředvídatelným způsobem.

---

<sup>2</sup>`DoubleArray` je použito, protože je to typ Kotlinu samotného, ale jelikož matematika v Neuronových sítích je implementována pomocí `Matrix<Double>`, musí se převést mezi typy.

TODO(FILTER)

Obrázek 3.1: Ilustrace filtru z třídy `ConvolutionalNetwork`

### 3.1.4 `ConvolutionalNetwork`

Tato třída rozhraní `INeuralNetwork` implementuje konvoluční neuronové sítě. Její konstruktor přijímá dva parametry typu `BasicNeuralNetwork`, první je filtr, druhá je samotná neuronová síť. Dalším parametrem je logická hodnota, zda se má i filtr učit (to se ale téměř nepoužívá, takže je tato hodnota při neuvedení nastavena na `false`).

Companion object této třídy navíc obsahuje příklad takového filtru (jednoduchý filtr detekující hrany viz obrázek 3.1)

## 3.2 `mnistDatabase`

### 3.2.1 Databáze MNIST

„Dataset MNIST, dataset ručně psaných číslic dostupná na stránkách <http://yann.lecun.com/exdb/mnist/> obsahuje 60 000 tréninkových a 10 000 ověřovacích příkladů. MNIST vychází z databáze spravované NIST (National Institute of Standards and Technology). Číslice mají normalizovanou velikost a jsou vycentrované v obrázcích shodné velikosti.“ (LeCun et al. 2019, přeloženo) Ukázku takových obrázků vidíme na obrázku 3.2.

Tuto databázi jsem použil pro první testování `BasicNeuralNetwork`, jelikož má pro první testování dostačující velikost. Pro pozdější testování využívám převážně EMNIST.

### 3.2.2 Databáze EMNIST

„Databáze MNIST se stala standardem pro učení umělého vidění. Databáze MNIST je odvozená z databáze NIST Special Database 19, která obsahuje ručně psané číslice a velká i malá písmena. EMNIST (Extended MNIST), varianta celé databáze NIST, přebírá uspořádání z databáze MNIST<sup>3</sup>.“ (Cohen et al. 2017, přeloženo)

Tato databáze obsahuje více příkladů než MNIST, navíc obsahuje i sety s písmeny, proto jsem po prvních pokusech s MNIST přešel na tuto databázi.

---

<sup>3</sup>Má však prohozené řádky a sloupce obrázků.





Obrázek 3.2: Příklad obrázků z datasetu MNIST

TODO(<https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png>)

## Kapitola 4

### Používání knihovny

## Závěr

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

# Bibliografie

- Brookshear, J. Glenn, David T. Smith a Dennis Brylow (2013). *Informatika*. Czech. Přel. English Jakub Goner. 1. vyd. Brno, CZ: Computer Press. 608 **pagetotals**. ISBN: 978-80-251-3805-2.
- Cohen, Gregory et al. (2017). “EMNIST: an extension of MNIST to handwritten letters”. In: *arXiv preprint arXiv:1702.05373*.
- LeCun, Yann, Corinna Cortes a Christopher J.C. Burges (15.pros. 2019). *THE MNIST DATABASE*. of handwritten digits. English. URL: `\url{http://yann.lecun.com/exdb/mnist/}`.

# Slovníček pojmů

**axon** výběžek vedoucí signál z neuronu. 10, 21

**dendrit** výběžek vedoucí signál do neuronu. 10, 21

**interface** česky rozhraní je v objektově orientovaném programování zabalení funkcí a vlastností třídy, které by měla každá třída z nějaké skupiny mít (např. každá fronta by měla mít funkci pro přidání a odebrání prvku a jedna z jejích vlastností je velikost). 15

**JVM** Java Virtual Machine je virtuální stroj, který umožňuje běh Java Bytecodu, kódu, do kterého se překládá Java a Kotlin. 7

**Kotlin** programovací jazyk vyvíjený firmou JetBrains, založen na Javě. 3, 4

**synapse** spojení (mezera) mezi axonem a dendritem, jež podle svých chemických vlastností zesílí nebo zeslabí signál předávaný z axonu do dendritu. 10, 11

**třída** anglicky class je základní prvek objektově orientovaného programování. Obsahuje funkce a vlastnosti, které bude mít objekt, který se vytvoří z dané třídy (popřípadě třídy, jež budou z této třídy dědit). 21

## Seznam obrázků

3.1	Ilustrace filtru z třídy ConvolutionalNetwork . . . . .	16
3.2	Příklad obrázků z datasetu MNIST TODO( <a href="https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png">https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png</a> ) . . . . .	17

## Seznam tabulek

# Přílohy

1. Fotky z pokusů
2. Vlastní program
3. Dokumentace
4. Testovací data