



Gymnázium
České Budějovice
Jírovцова 8

MATURITNÍ PRÁCE

Neuronové sítě

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 právě  Kotlin, který je mým oblíbeným programovacím jazykem a který lze použít skoro kdekoli (webové stránky, servery, mobily), takovou knihovnu postrádá. Proto jsem se rozhodl svoji práci koncipovat jako snahu o implementování takové knihovny.

Klíčová slova

Neuronové sítě, Neuron, Umělá inteligence, Aktivační funkce,  Kotlin, Multiplatformní knihovna, Java, Javascript

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, L^AT_EX, ve kterém píšu text a dále  git a  GitHub, jež uchovávají má data, ať už text nebo knihovnu.

Obsah

I	Teoretická část	8
1	Laický náhled na neuronové sítě	9
1.1	Neuron	9
1.2	Aktivační funkce	10
1.3	Sítě	10
1.4	Dopředná propagace a zpětná propagace	10
1.5	Využití neuronových sítí	11
2	Formální náhled	12
2.1	Aktivační funkce	13
II	Praktická část	17
3	Struktura knihovny	18
3.1	core	18
3.1.1	ActivationFunctions	18
3.1.2	INeuralNetwork	18
3.1.3	BasicNeuralNetwork	19
3.1.4	ConvolutionalNetwork	19
3.2	mnistDatabase	20
3.2.1	Databáze MNIST	20
3.2.2	Databáze EMNIST	20
4	Používání knihovny	22
4.1	Nastavování hodnot	22

Apendix	22
Slovníček pojmů	24
Bibliografie	25
Seznam obrázků	26
Přílohy	27
Fotky z pokusů	27
Vlastní program	CD
Dokumentace	CD
Testovací data	CD

Úvod

Neuronové sítě jsou v poslední době velmi skloňované téma. Nikdo pořádně neví, jak to, že fungují tak dobře. 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 je interoperabilní s Javou, Javascriptem i C, a tak umožňuje tuto knihovnu používat jak pro JVM, tak i v prohlížeči nebo v programech kompilovaných přímo do binárního kódu.

V textu jsou použita slova ze stavby biologického neuronu, objektově orientovaného programování, Kotlinu, ... Tyto slova jsou vysvětlena na konci práce.

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

Část I

Teoretická část

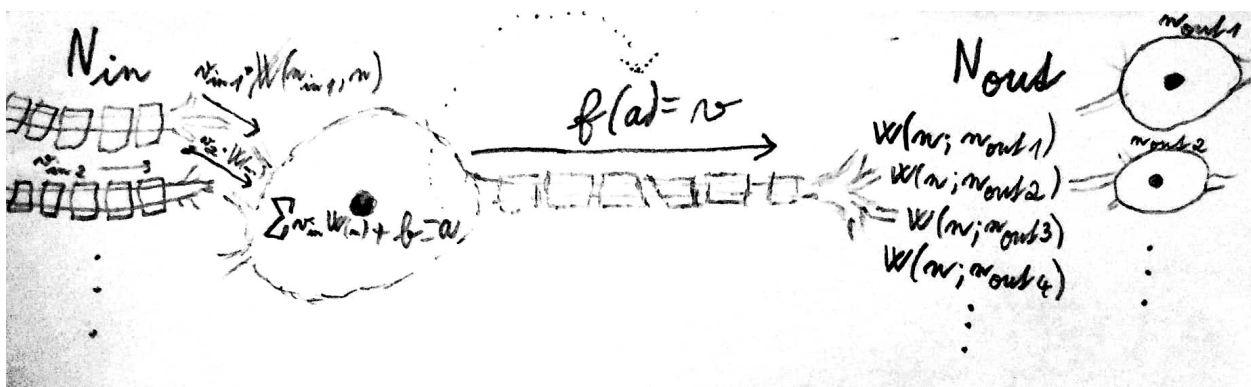
1 Laický náhled na neuronové sítě

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 [1, 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 (viz dále) a výstupní signál. Často navíc bude obsahovat základní hodnotu (angl. bias), která reprezentuje mez, při jejímž překročení začne neuron vysílat signál. Jinak řečeno posouvá aktivační funkci ve směru osy x .



Obrázek 1.1: Analogie umělého a biologického neuronu.

1.2 Aktivační funkce

Jak už bylo zmíněno, přírodní neuron funguje na principu toho, že když součet vstupních signálů nepřekračuje určitou mez, nevysílá neuron žádný (nebo téměř žádný) signál. Když je však tato mez překonána, neuron vyšle signál. V podstatě tedy vysílá buď 0 nebo 1. Pro účely umělého neuronu je 0 a 1 nedostačující, jelikož při procesu učení potřebujeme měnit hodnoty jemně, abychom nerozbili již naučené znalosti.

Proto se jako aktivační funkce (tedy to, co určuje jaký má být výstup v závislosti na součtu vstupů, v případě přírody tedy funkce zobrazující interval $-\infty$ až mez (bias) na 0 a zbylá čísla na 1 viz *binární krok* v sekci) používají funkce co nejvíce podobné právě tomuto binárnímu kroku, které jsou ale spojitě a mají co „nejhezčí“ derivace (protože při zpětné propagaci právě podle derivace určíme, jak moc daný neuron ovlivňuje výsledek).

1.3 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 jedné 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.

TODO(Konvoluční síť)

TODO(Asociativní paměť)

1.4 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, určení, které proměnné hodnoty (synapse a biasy) se na ní nejvíce podílejí. Potom tyto hodnoty posuneme odpovídajícím způsobem (stejně jako příroda mění chemické vlastnosti synapse).

1.5 Využití neuronových sítí

Než se pustíme do matematiky, která stojí za fungováním neuronových sítí, ještě si řekneme, kde a jaké neuronové sítě využíváme. Jedno z nejviditelnějších využití je rozpoznávání obrázků, protože takovou úlohu jen stěží zvládnou běžné algoritmy. Mezi rozpoznávání obrázku patří jak strojové čtení textů, tak třeba rozpoznávání tváře nebo klasifikace, zda je na obrázku morče, nebo slon. K tomu se používají hlavně konvoluční sítě, jelikož filtr rozezná hrany a různé útvary a neuronová síť podle toho určí dané rozřazení (znak, člověka, zvíře...).

Další oblastí je překlad. Překládat slova zvládneme jednoduše podle slovníků, ale aby věta dávala smysl a slovo bylo přeloženo v kontextu věty, potřebujeme něco více. Pro to se používá vektorový prostor slov, tedy všem slovům přiřadíme určitý vektor (to musíme udělat vždy, protože neuronová síť nemá jiný vstup) a poté na vzorovém textu učíme neuronovou síť odhadovat slovo podle několika okolních slov. Při tom ale neupravujeme jen hodnoty neuronové sítě, ale i vektorů slov. Tím dostaneme vektorový prostor slov, na kterém se překládající neuronová síť (jiná než ta, co vyrobila vektorový prostor) naučí překládat velmi lidsky. Stejný vektorový prostor se dá použít i na neuronovou síť generující text.

Když už bylo zmíněno generování, umělé neuronové sítě jsou schopny i generovat obrázky, hudbu, atd.¹ K tomu se používají dvě sítě, kdy jedna generuje a druhá dostane dvojici objekt vytvořený člověkem (resp. skutečností v případě fotek) a objekt vygenerovaný první sítí a má za úkol určit, který je který. Tyto sítě se učí spolu a výsledkem jsou relativně pěkná díla.

¹Stále je to však na základě nějakého datasetu obrázků nebo hudby.

2 Formální náhled

Označme $\nu = (N, W, F)$ neuronovou sítí, 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 ε je chyba (parciální derivace chybové funkce podle $f^{-1}(v)$ ¹). Potom dopředná propagace (tedy spočítání v) vypadá takto:

$$v = f \left(b + \sum_{n_x \in N_{in}, v_x \in n_x} v_x \cdot W(n_x, n) \right) \quad (2.1)$$

To lze při označení

$$\vec{v} = (v_1, v_2, \dots) \quad (2.2)$$

$$\vec{w} = (w_1, w_2, \dots) \quad (2.3)$$

$$(\forall n_x \in N_{in}) (\exists i \in \mathbb{N}) (v_i \in n_x \wedge w_i = W(n_x, n)) \quad (2.4)$$

zapsat vektorově jako:

$$v = f(b + \vec{w} \cdot \vec{v}) \quad (2.5)$$

Případně můžeme do vektorů „zakomponovat“ i bias²:

$$\vec{v} = (1, v_1, v_2, \dots) \quad (2.6)$$

$$\vec{w} = (b, w_1, w_2, \dots) \quad (2.7)$$

$$(\forall n_x \in N_{in}) (\exists i \in \mathbb{N}) (v_i \in n_x \wedge w_i = W(n_x, n)) \quad (2.8)$$

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

²To v knihovně není použito z důvodu netriviálního přidávání prvku do vektoru.

$$v = f(\vec{w} \cdot \vec{v}) \quad (2.9)$$

Při zpětné propagaci je důležitý vzorec pro derivaci složené funkce, někdy také znám jako „řetízkové pravidlo“ (pro funkci jedné proměnné platí (2.10), pro více pak (2.11)):

$$\frac{dy}{dx} = \frac{dz}{dx} \frac{dy}{dz} \quad (2.10)$$

$$\frac{\delta y}{\delta x} = \sum_z \frac{\delta z}{\delta x} \frac{\delta y}{\delta z} \quad (2.11)$$

Potom můžeme ε neuronu spočítat jako součet derivací ε neuronů, do kterých posílá signál, podle $f^{-1}(v)$:

$$\varepsilon = \frac{\sum_{\varepsilon \in N_{out}, \varepsilon_x \in n_x} \delta \varepsilon_x}{\delta f^{-1}(v)} = f'(f^{-1}(v)) \sum_{n_x \in N_{out}, \varepsilon_x \in n_x} \varepsilon_x \cdot W(n, n_x) \quad (2.12)$$

Obdobně jako v předchozím případě definujeme vektory

$$\vec{\varepsilon} = (\varepsilon_1, \varepsilon_2, \dots) \quad (2.13)$$

$$\vec{w} = (w_1, w_2, \dots) \quad (2.14)$$

$$(\forall n_x \in N_{out}) (\exists ! i \in \mathbb{N}) (\varepsilon_i \in n_x \wedge w_i = W(n, n_x)) \quad (2.15)$$

$$v = f'(f^{-1}(v)) \cdot (\vec{w} \cdot \vec{\varepsilon}) \quad (2.16)$$

2.1 Aktivační funkce

Jelikož neurony mají bias, není nutné udávat aktivační funkce obecně, stačí je jen udat tak, že $x = 0$ odpovídá mezi v pomyslném biologickém neuronu. Mezi aktivační funkce³ patří:

- *Binary step*

$$f(x) = \begin{cases} 0, & \text{když } x < 0 \\ 1, & \text{když } x \geq 0 \end{cases} \quad (2.17)$$

$$f'(x) = \begin{cases} 0, & \text{když } x \neq 0 \\ +\infty, & \text{když } x = 0 \end{cases} \quad (2.18)$$

(česky *binární krok*), již zmíněná funkce, jež odpovídá reálnému neuronu, ale není použitelná pro učení na základě gradientu, jelikož má derivaci 0 všude kromě bodu $x = 0$, kde je nespojitá.

³Funkce jsem čerpal převážně z [2]

- *Identity*

$$f(x) = x \quad (2.19)$$

$$f'(x) = 1 \quad (2.20)$$

(česky *identita*) odpovídá stavu, jako kdyby tam žádná funkce nebyla. Její derivace je 1, tedy se velmi snadno určí v libovolném bodě.

- *Sigmoid* (značí se σ)

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.21)$$

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x) \cdot (1 - \sigma(x)) \quad (2.22)$$

je jedna z nejznámějších aktivačních funkcí. Je to vlastně takový hladký přechod mezi 0 a 1. Také je na σ dobře vidět, proč se často počítá derivace z funkční hodnoty, místo počítání exponenciální funkce a dělení si vystačíme s násobením a odčítáním.

- Nesmíme zapomenout na *sigmoidě* podobnou a také často používanou funkci *hyperbolický tangens* (\tanh) [3]:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1 = 2 \cdot \sigma(2x) - 1 \quad (2.23)$$

$$\tanh'(x) = \frac{1}{\cosh^2(x)} = \frac{\cosh^2(x) - \sinh^2(x)}{\cosh^2(x)} = 1 - \tanh^2(x) \quad (2.24)$$

$$\tanh'(x) = 4 \cdot \sigma'(2x) \quad (2.25)$$

Největší rozdíl oproti σ je, že může nabývat i záporných hodnot, což sice moc neodpovídá přírodnímu neuronu, ale když si rozmyslíme, že stačí zvětšit biasy u neuronů, do kterých neuron s aktivační funkcí \tanh vysílá signál, dospějeme k výsledku, že tato funkce také funguje.

- Další funkce s vazbou na *sigmoidu* je funkce *swift*:

$$f(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}} \quad (2.26)$$

Nepodařilo se mi ale najít derivaci za pomoci funkční hodnoty. *Sigmoida* se také používá ve spojení s ostatními funkcemi, většinou $\sigma(x)$ pro kladné a druhá funkce pro záporné.

- Ukazuje se, že identita jako taková se v podstatě použít nedá, ale hojně využívaná je její „upravená“ verze *rectified linear unit* (česky něco jako *napravená přímá úměrnost*), která záporná čísla převádí na nulu a v kladných se chová jako *identita*:

$$f(x) = \begin{cases} 0, & \text{když } x < 0 \\ x, & \text{když } x \geq 0 \end{cases} \quad (2.27)$$

$$f'(x) = \begin{cases} 0, & \text{když } x < 0 \\ 1, & \text{když } x > 0 \\ \text{neexistuje,} & \text{když } x = 0 \end{cases} \quad (2.28)$$

Trochu připomíná biologický neuron, protože pro záporné hodnoty nevysílá, ale na rozdíl od něj má variabilní hodnotu vysílaného signálu. Často se například používá ve filtrech, jelikož chceme detekovat, zda je někde hrana, ale nechceme vysílat záporný signál, když někde hrana není, protože může být o pixel vedle.

Kromě této verze je v knihovně ještě *leaky* (děravá či prosakující) *rectified linear unit*, která v záporných hodnotách nedává nulu, ale *přímou úměrnost*. K těmto funkcím můžeme přiřadit i *hard hyperbolic function*, která je *identitou* pouze na intervalu $(-1, 1)$, tedy odpovídá biologickému neuronu asi nejvíce z těchto „lineárních funkcí“.

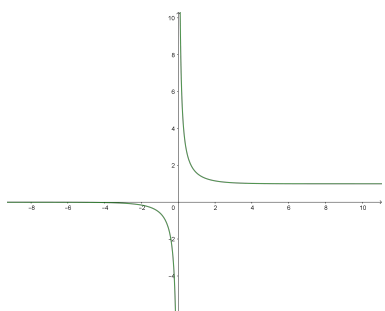
Rectified unit není hladká (nemá derivaci v bodě nula), ale to lze napravit, když použijeme funkci *soft plus* ($\ln(1 + e^x)$). Podobnou úpravu lze udělat i s funkcí *signum* (*znaménko*, často se značí *sign*), což je téměř *binární krok*⁴, akorát v záporných hodnotách nabývá funkční hodnoty -1 místo 0. *Signum* se dá zapsat jako podíl x a $|x|$, tudíž tato úprava (*soft sign*) vypadá následovně:

$$f(x) = \frac{x}{|x| + 1} \quad (2.29)$$

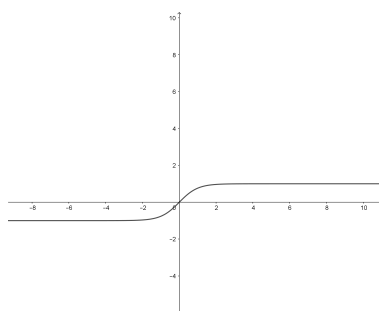
- Jednou skupinou funkcí, se kterými se sice experimentuje, ale stěží najdete nějaké využití, jsou ty, které nejsou monotonní⁵, jako sinus, kosinus, *Gaussova funkce* (e^{-x^2}), apod. Vzhledem k jejich mizivému využití je implementován pouze *sinus*.

⁴Z důvodu téhle podobnosti není ani implementována.

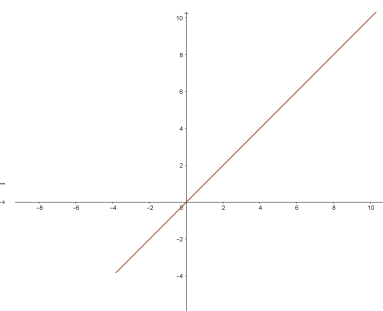
⁵Můžeme si všimnout, že téměř všechny předchozí funkce jsou neklesající, většina dokonce rostoucí na celém definičním oboru.



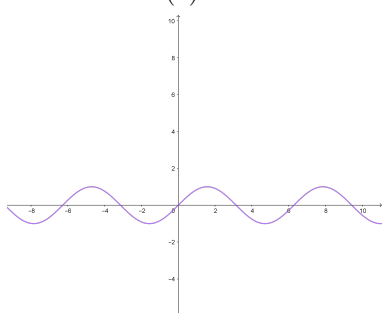
(a) σ



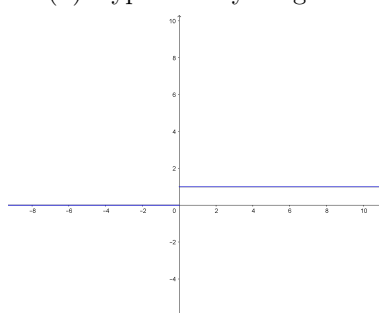
(b) Hyperbolický tangens



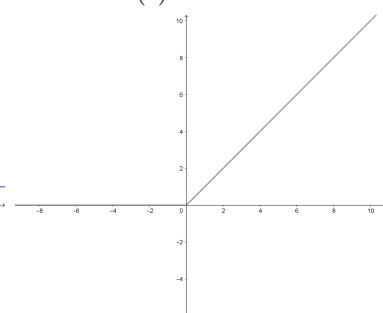
(c) Identita



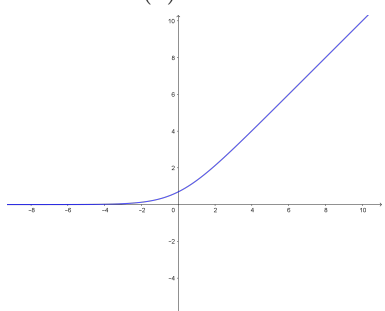
(d) Sinus



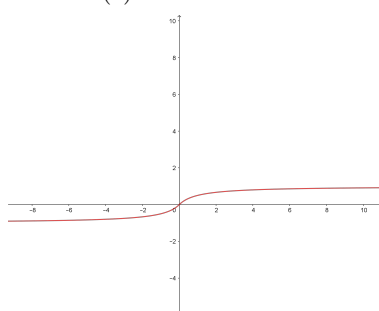
(e) Binární krok



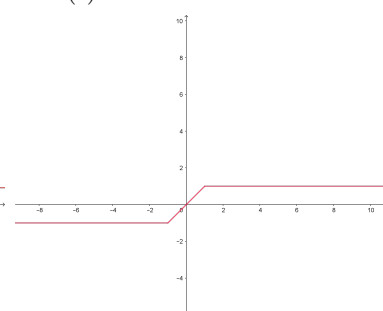
(f) Rectified linear unit



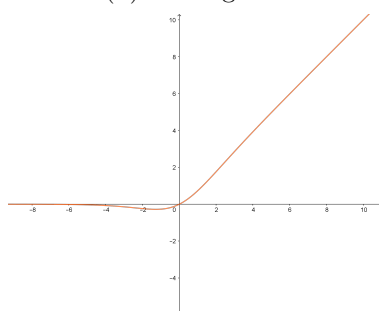
(g) Soft plus



(h) Soft signum



(i) Hard hyperbolic function



(j) Swift

Obrázek 2.1: Aktivační funkce (vyrobeny v programu Geogebra)

Část II

Praktická část

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 datech z databází ve formátu MNIST.

3.1 `core`

3.1.1 `ActivationFunctions`

Enumerate (česky výčet) funkcí, jež se používají jako aktivační funkce v neuronech. Funkce lze zavolat s parametrem typu `Double`, což nám dá hodnotu funkce v tomto bodě, popřípadě lze obdobně zavolat jejich dvě metody `xD` a `yD` udávající v pořadí hodnotu derivace v bodě `x` a v bodě, kde je funkční hodnota rovna parametru.

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.

Implementovány jsou všechny funkce uvedené v kapitole 2.1

3.1.2 `INeuralNetwork`

Rozhraní (anglicky interface), které implementuje základní funkce neuronových sítí, které mají jako vstup i výstup vektor `Double`. 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`.¹

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, chtěné výstupní vektory)`, která je koncipována tak, aby nejdříve provedla `run(vstupní vektor)`, výsledek porovnala 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, která 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.

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 neu-

¹`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

ronová 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

Pro otestování knihovny je potřeba nějaký dataset. K tomuto účelu je v knihovně implementována třída `TODO`, která umí přečíst data z databáze MNIST a EMNIST. Poté poskytuje vždy jedno zadání (obrázek číslice / písmena) a jeho řešení (ve formě vektoru, kde pouze na správném místě je 1, jinak je všude 0).

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.“ [4, přeloženo] Ukázku takových obrázků vidíme na obrázku 3.2.

Tuto databázi jsem použil pro první testování své 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 MNIST².“ [5, 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.

²Má však prohozené řádky a sloupce pixelů v obrázcích.



Obrázek 3.2: Příklad obrázků z datasetu MNIST

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

4 Používání knihovny

4.1 Nastavování hodnot

Neuronová síť má mnoho hodnot, které lze nastavit. Knihovnu jsem zkoušel na rozpoznávání čísel v databázích MNIST a EMNIST a zjistil jsem, že vhodné nastavení hodnot je asi:

- Learning rate je třeba nastavit na cca 0.1 a pomalu snižovat.
- Počet skrytých vrstev musí být právě jedna (dvě už se nenaučí propojit vstup s výstupem a bez skryté vrstvy vůbec nefunguje). Pokud byste potřebovali učit síť s více skrytými vrstvami, musíte nastavit learning rate na daleko nižší hodnotu.
- Počet neuronů ve skryté vrstvě je hodně variabilní, ideálně mezi hodnotami 100 a 300.
- Jako aktivační funkce stačí třeba sigmoida, jiné jsem nepoužíval.

Závěr

Povedlo se mi implementovat neuronovou síť do takové míry, aby byla schopna rozeznávat číslíce (TODO(znaky/číslíce)) (ukázka je na stránkách moznabude.cz). Dále bych mohl pokračovat například implementováním lepšího ukládání do souboru (ukládání typu `Double` jako textového řetězce není moc efektivní), implementování nějakých genetických algoritmů, či naprogramování konvoluční sítě tak, aby filtry mohly pracovat n rozměrně.

Slovníček pojmů

axon výběžek vedoucí signál z neuronu. 9, 24

dendrit výběžek vedoucí signál do neuronu. 9, 10, 24

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). 18

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. 9, 10

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). 24

Bibliografie

- [1] J. Glenn Brookshear, David T. Smith a Dennis Brylow. *Informatika*. Czech. Přel. English Jakub Goner. 1. vyd. Brno, CZ: Computer Press, 2013, s. 608. ISBN: 978-80-251-3805-2.
- [2] Wikipedia contributors. *Activation function* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 9-January-2020]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=933057521 (cit. 09.01.2020).
- [3] Farnoush Farhadi. *Learning activation functions in deep neural networks*. Université De Montréal (école Polytechnique De Montréal), 2017.
- [4] Yann LeCun, Corinna Cortes a Christopher J.C. Burges. *THE MNIST DATABASE of handwritten digits*. English. 1998. URL: <http://yann.lecun.com/exdb/mnist/> (cit. 15.12.2019).
- [5] Gregory Cohen et al. “EMNIST: an extension of MNIST to handwritten letters”. In: *arXiv preprint arXiv:1702.05373* (2017).

Seznam obrázků

1.1	Analogie umělého a biologického neuronu.	9
2.1	Aktivační funkce (vyrobeny v programu Geogebra)	16
3.1	Ilustrace filtru z třídy ConvolutionalNetwork	20
3.2	Příklad obrázků z datasetu MNIST TODO(https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png)	21

Přílohy

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