VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

DLOUHODOBÉ PREDIKTIVNÍ MODELOVÁNÍ NELINEÁRNÍCH DYNAMICKÝCH SYSTÉMŮ POMOCÍ KONVOLUČNÍCH NEURONOVÝCH SÍTÍ

LONG-TERM PREDICTIVE MODELLING OF NONLINEAR DYNAMICAL SYSTEMS USING CONVOLUTIONAL NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Vojtěch Ptáčník

AUTHOR

VEDOUCÍ PRÁCE — Ing. Jiří Kovář, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Vojtěch Ptáčník**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**Ing. Jiří Kovář, Ph.D.**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijníma zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Dlouhodobé prediktivní modelování nelineárních dynamických systémů pomocí konvolučních neuronových sítí

Stručná charakteristika problematiky úkolu:

Vytvoření počítačového modelu pro nelineární systém je možné dosáhnout použitím konvoluční neuronové sítě. Úkol je o to težší, pokud vytvořený model má být dlouhodobě přijatelně přesný. Cílem práce je prozkoumání a realizace použitím neuronové sítě pro tyto účely.

Cíle bakalářské práce:

1) Proveďte rešerši zadané problematiky a zhodnoťte jednotlivá řešení s uvážením vlastností sledované technické soustavy
2) Navrhněte softwarové řešení na základě provedené rešerše
3) Proveďte test vytvořeného softwarového řešení na reálných datech získaných z měření sledované technické soustavy – obráběcího stroje

Seznam doporučené literatury:

BURKOV, A., The Hundred-Page Machine Learning Book, 2019, ISBN-13 978-1999579500

GERON, A., Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts,Tools, and Techniques to Build Intelligent Systems, 2019, ISBN-13: 978-1492032649

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Tato práce se zabývá vytvořením počítačového modelu konvoluční neuronové sítě. Úkol je o to obtížnější, protože se jedná o nelineární systém, který požaduje dlouhodobou přijatelnou přesnost.

Hlavní částí je rešeršní výzkum na dané téma, který rozvede jednotlivá řešení neuronových sítí a zaměří se na konvoluční neuronové sítě, které mohou být vhodnou volbou pro řešení nelineárních systémů.

Ke splnění cílů práce je zapotřebí vytvořit výpočetní model, který bude následně předpovídat nelineární systém. Bude demonstrována a analyzována schopnost konvolučních sítí předpovídat tyto systémy.

Abstract

This work deals with creating a computer model of a convolutional neural network. The task is more difficult because it involves a nonlinear system that requires long-term acceptable accuracy.

The main part consists of research on the topic, which explores various solutions of neural networks and focuses on convolutional neural networks, which can be a suitable choice for solving nonlinear systems.

To achieve the goals of the work, it is necessary to create a computational model that will subsequently predict the nonlinear system. The ability of convolutional networks to predict these systems will be demonstrated and analyzed.

Klíčová slova

Neuronové sítě, Konvoluční neuronové sítě, Nelineární systém, Dlouhodobé předpovídání

Keywords

Neural networks, Convolutional neural networks, Nonlinear system, Long-term prediction.

Bibliografická citace

PTÁČNÍK, Vojtěch. *Dlouhodobé prediktivní modelování nelineárních dynamických systémů pomocí konvolučních neuronových sítí.* Brno, 2023. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/145817. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Kovář.

Čestné prohlášení

Prohlašuji, že jsem bakalářskou práci na téma „Dlouhodobé prediktivní modelování nelineárních dynamických systémů pomocí konvolučních neuronových sítí“ vypracoval samostatně s použitím odborné literatury a literatury uvedené v seznamu použitých zdrojů na konci dokumentu.

26. května 2023 ……

Vojtěch Ptáčník

Poděkování

Děkuji tímto Ing. Jiřímu Kovářovi, Ph.D. za cenné rady, připomínky a trpělivost při práci na bakalářské práci.

Obsah

1. Úvod 10
2. Strojové učení 11
2.1 Učení pod dohledem (Supervised learning) 11
2.2 Učení bez dohledu (Unsupervised learning) 12
2.3 Posilovací učení (Reinforcement learning) 13
3. Neuronové sítě 14
3.1 Neuron 14
3.1.1 Zkreslení a Rozhodovací Hranice 15
3.2 Případ s více výstupy 15
3.3 Vícevrstvý perceptron (Multilayer Perceptron) 17
3.4 Rekurentní neuronová síť 17
3.4.1 Jednoduchá rekurentní neuronová síť 18
3.4.2 Long Short-Term Memory (LSTM) 18
3.4.3 Gated Recurrent Unit (GRU) 19
3.5 Konvoluční neuronové sítě (Convolutional Neural Networks – CNN) 20
4. Struktura výpočtu konvolučních neuronových sítí 21
4.1 Konvoluční vrstva 21
4.2 Sdružovací vrstva (Pooling layer) 22
4.3 Aktivační funkce 22
4.3.1 Sigmoid 23
4.3.2 Tanh (hyperbolický tangent) 23
4.3.3 Softmax funkce 23
4.3.4 ReLU (rectified linear unit) 23
4.3.5 Swish funkce 24
4.4 Ztrátová funkce 24
4.4.1 Binární ztrátová funkce Cross-Entropy 24
4.4.2 Ztrátová funkce cross-entropy pro více třídovou klasifikaci 25
4.4.3 $\mathcal{L}p$ Norma 25
4.4.4 Střední kvadratická chyba (Mean Squared Error) 26
4.4.5 Střední absolutní chyba (Mean Absolute Error) 26
4.5 Optimalizační algoritmy 26
4.5.1 Stochastic Gradient Descent (SGD) 27
4.5.2 RMSProp (Root Mean Square Propagation) 27
4.5.3 Adam (Adaptive Moment Estimation) 28

4.5.4 Adagrad 29
4.5.5 Adadelta 30
4.5.6 Adamax 31
4.6 Dropout 31
4.7 Batch normalization 32
5. 1D Konvoluční neuronové sítě (1D CNN) 33
5.1 Dopředná a zpětná propagace v CNN vrstvách 35
5.2 Konvoluční Autoenkodér (CAE) 38
6. Realizace 39
6.1 Výpočetní Hardware 39
6.2 Knihovny Python 39
6.3 Charakterizace dat a normalizace 39
6.4 Zvolení výpočetního algoritmu 40
6.5 Popis algoritmu 41
6.6 Volba nejvhodnějšího nastavení modelu 41
6.7 Trénování algoritmu 41
6.7.1 Volba filtrů 41
6.7.2 Volba velikosti kernelů 43
6.7.3 Volba optimalizačního algoritmu 44
6.7.4 Volba ztrátové funkce 45
6.7.5 Volba aktivačních funkcí 46
6.7.6 Dodatečná regularizace 48
6.8 Predikce 50
7. Závěr 51

1. Úvod

Dynamické systémy jsou složité a nelineární systémy, které se vyvíjejí v průběhu času, což je dělá složitými na předpovídání. Studium dynamických systémů je stěžejní pro většinu technických oborů. Předpovídání chování dynamických systémů je důležité pro mnoho aplikací, jako řízení, optimalizace a rozhodování.

Konvoluční neuronové sítě jsou výkonnou třídou algoritmů hlubokého učení, které dosáhly obdivuhodných výsledků v různých aplikacích, včetně rozpoznávání obrazu a řeči, zpracování přirozeného jazyka a analýzy časových řad. Konvoluční neuronové sítě jsou navrženy tak, aby se naučily prostorové a časové vzorce z vstupních dat a ty následně použily k předpovídání časových řad.

V této práci bude navrhnut přístup k dlouhodobému prediktivnímu modelování nelineárních dynamických systémů pomocí konvolučních neuronových sítí. Cílem je prozkoumat jejich potenciál.

Hlavním cílem této práce je vyvinout rámec pro co nejpřesnější předpovídání budoucího chování nelineárních dynamických systémů. K dosažení tohoto cíle budou prozkoumány různé architektury a strategie trénování pro konvoluční neuronové sítě.

2. Strojové učení

Strojové učení spadá pod počítačové vědy. Zabývá se vytvářením algoritmů, které se řídí pomocí předem určeného souboru dat. Strojové učení je typ umělé inteligence, která umožňuje výpočetní technice učit se a zlepšovat se bez vnějšího zásahu programátora. Zahrnuje vytváření algoritmů a modelů, které umí automaticky určit daný problém a předpovědět, či rozhodnout o budoucím průběhu.

Ve strojovém učení je počítačový systém trénován na velkém datovém souboru, díky kterému je schopný se učit pomocí schémat a vztahů mezi daty. V tomto oboru jsou hojně využívány znalosti ze statistiky, které pomáhají odhalit vzory řešení potřebné pro tvorbu předpovědí nebo rozhodnutí pro nová, počítačem dosud neznámá data.

Jednotlivé body sledované strojovým učením jsou rozděleny různými vlastnostmi. Tyto vlastnosti je možné rozdělit do dvou skupin: rysy a štítky. Rysy jsou vlastnosti jednoduše počitatelné nebo měřitelné formou automatických procesů. Štítky jsou vlastnosti, které nelze změřit jednoduše, a je zapotřebí přítomnost vnějšího subjektu – experta. [1]

Strojové učení se rozděluje podle stylu učení do následujících skupin: učení pod dohledem, učení bez dohledu, posilovací učení. [2]

2.1 Učení pod dohledem (Supervised learning)

Učení pod dohledem je typ učení, které se pokouší předpovědět funkci nebo vztah založený na popsaných trénovacích datech a následně tuto získanou dovednost použít na nepopsaná data. Záleží také na množství dat, která byla obdrženy pro testování, což může ovlivnit přesnost sledované soustavy. Hlavním znakem tohoto typu učení jsou předem dané vstupní a výstupní hodnoty sledovaného systému. [3]

Určeme vstup učení jako *X* a výstup jako *Y*. V učení pod dohledem je *Y* vždy definované. Vyskytují se zde dva hlavní typy učení pod dohledem: klasifikace a regrese.

V *klasifikaci* $Y \in \{0, 1, \dots, c-1\}$, kde c je počet tříd. Proměnná *Y* má v tomto případě označení jako štítek, protože se nejedná o hodnotu s číselným významem, ale pouze o rozdělení kategorií. Například pro binární klasifikaci nabývá hodnota $c = 2$, kde se následně kontroluje chyba pomocí *ztráty z nesprávné klasifikace*. [4]

V *regresi* proměnná *Y* nabývá hodnot numericky validních, což znamená, že se může jednat o systém diskrétní nebo spojitý. Základní ztrátová funkce pro řešení tohoto problému je *střední kvadratická chyba* (mean-squared error). [4]

2.2 Učení bez dohledu (Unsupervised learning)

Na rozdíl od učení pod dohledem, kde jsou proměnné X a Y známé, v učení bez dohledu známe pouze rozdělení $\boldsymbol{X}$. Tudíž se zde neprovádí předpověď a její ztrátová chyba. Není zde ani přímočaré určení výsledku výpočtu. Učení bez dohledu primárně určuje rozdělení a vztahy v rozdělení $\boldsymbol{X}$. [4]

Na základě požadovaného výsledku sledované soustavy určuje výpočetní mechanismus potřebné rozhodnutí, předpověď budoucích vstupů nebo efektivní předání informace dalším výpočetním strojům. Hlavní myšlenkou učení bez dohledu je hledání vzorů v uskupení dat, které by běžně působily jako nahodilý šum. [5]

K řešení problému pomocí učení bez dohledu lze přistupovat těmito způsoby: odhad hustoty; shlukování; dimenzionální redukce, reprezentace, získávání vlastností; generování nových vzorků.

Odhad hustoty se zaměřuje přímo na určení pravděpodobnostního rozdělení, které může být vhodné pro návrh kompresních algoritmů, nebo detekci odlehlých hodnot.

Shlukování sleduje všechny body v testované množině a ty následně rozdělí do skupin s podobnými vlastnostmi.

Dimenzionální redukce, reprezentace a získávání vlastností jsou tři na sobě závislé úkony, které představují každý bod sledované soustavy v jiném prostoru. Tento způsob řešení problému lze aplikovat na systémy s menší dimenzionální velikostí za účelem zvýraznění nezávislých veličin, zjednodušení vizualizace, interpretace nebo realizace budoucích úkonů.

Generování nových vzorků má za úkol přidat ke stávajícím vzorkům, o určitém pravděpodobnostním rozdělení, nové body, které jsou podobné stávajícím datům. [6]

Autoenkoder (AE) využívá učení k rekonstrukci vstupního signálu na výstup sítě. Základní struktura AE obsahuje jednu skrytou vrstvu. Proces mezi vstupem a skrytou vrstvou se nazývá *enkoder*. Úloha enkoderu je namapovat vstupní vektor $X = (x_1, x_2, \dots, x_n)$ na nižší reprezentaci $H = (h_1, h_2, \dots, h_n)$ ve skryté vrstvě, která je označena jako vysoce-úrovňová vlastnost vstupu. Znázorněno následovně:

$$H = f(WX + b). \tag{1}$$

Podobně probíhá i proces mezi skrytou a výstupní vrstvou, který se nazývá *dekoder*. V tomto úkonu dekoder rekonstruuje nižší reprezentaci H na vstupní data vektoru X. Popsáno rovnicí:

$$Z = g(W'H + b'). \tag{2}$$

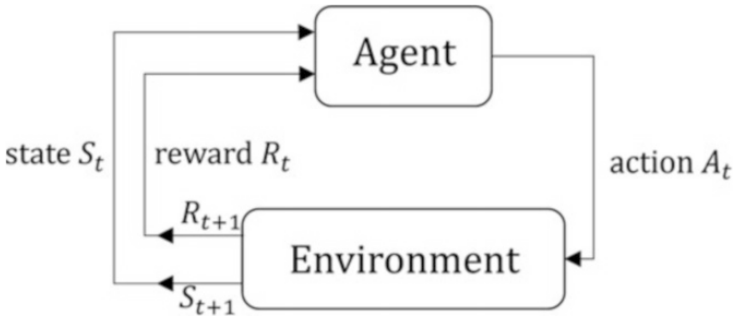
V těchto rovnicích jsou f a g nelineární aktivační funkce. W a b jsou váhy a zkreslení vektoru enkoderu. Podobně, W' a b' jsou váhy a zkreslení vektoru dekoderu. Tyto parametry AE jsou označeny jako $\theta = \{W, W', b, b'\}$ a jsou optimalizovány během procesu trénování minimalizací rekonstrukční chyby. [7]

2.3 Posilovací učení (Reinforcement learning)

Stejně jako u učení bez dohledu, posilovací učení nemá přístup k výstupním datům. Tudíž nelze použít ztrátovou funkci pro kontrolu a úpravu dat, jako je tomu u učení pod dohledem. Avšak je možné použít ztrátovou funkci na jednotlivé body, kdy je použita forma odměny. Pro příklad je tento jev možné vidět v autonomním řízení vozidla, kdy sledovaná soustava je pohyb automobilu, který disponuje senzorem a ztrátová funkce vyhodnocuje vzdálenost automobilu od překážky. [1]

Hlavními členy posilovacího učení jsou **agent** a **prostředí**. Prostředí je subjekt, se kterým komunikuje a interaguje agent. Cílem tohoto učení je naučit agenta správnému chování v daném prostředí pod určitým předdefinovaným cílem. Agent dostává **odměnu** *R*.

Vztah mezi agentem a prostředím lze popsat pomocí Obr. 1. V libovolném časovém kroku *t*, agent jako první pozoruje stav prostředí S_t, a jemu náležící hodnotu odměny R_t. Agent se následně rozhoduje o budoucím kroku na základě stavu prostředí a jeho odměně. Akce, kterou agent provádí, A_t, vstupuje do prostředí, které následně reaguje uvedením nových stavů S_{t+1} a R_{t+1}.



Obr. 1. Popis schématu fungování vztahu Agent-Prostředí. [8, s. 48]

Pro poskytnutí odměny agentovi, získané z prostředí, vytváří odměnová funkce R okamžitou odměnu R_t, která bývá poslána agentovi v každém časovém kroku výpočtu. Může nastat případ kdy je odměnová funkce závislá pouze na momentálním stavu prostředí, poté nastává $R_t = R(S_t)$.

Sekvence stavů, akcí a odměn je definována jako **trajektorie** $\tau = (S_0, A_0, R_0, S_1, A_1, R_1, \dots.)$, která zaznamenává, jak agent interaguje s prostředím, kde S_0 je náhodně zvolený stav vybraný z počátečního stavového rozdělení.

Přechody mezi stavy můžou být buď **deterministický přechodový proces** nebo **stochastický přechodový proces**. Pro deterministický přechod je určen další stav deterministickou funkcí, oproti stochastickému přechodu, kde je další stav popsán pravděpodobnostním rozdělením.

Pokud je trajektorie popsána od počátečního stavu až po jeho konečný stav, označuje se, jako **epizoda**.

Využití a **průzkum** jsou dva pojmy, které umožňují přístup k řešení problému. Proces využití je konzervativní, co se řešení týče a snaží se provádět nejlepší předpovídané výsledky na aktuálně existující informaci o soustavě. Proces průzkumu je o riskování a

navyšování existujících informací o prostředí i za cenu horších výsledků. Zájmem posilovacího učení je najít **kompromis** mezi procesy využití a průzkumu tak, aby se maximalizovala výpočetní efektivita systému. [8] [9]

3. Neuronové sítě

Neuronové sítě jsou silný nástroj v odvětví umělé inteligence a strojového učení. Jsou modelovány podle předlohy struktury a funkcí lidské mysli. Stejně jako je základní stavební jednotkou lidského mozku neuron, je tomu tak i v počítačových neuronových sítích. Tato jednoduchá výpočetní jednotka je propojena v síti a pracuje, spolu s dalšími jednotkami, na výpočtech informací a předpovědí.

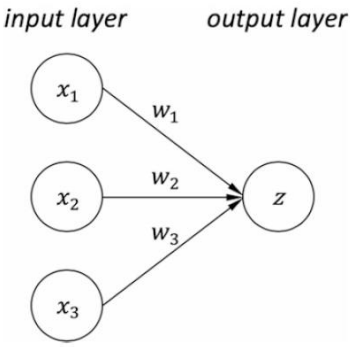
3.1 Neuron

V neuronových sítích má neuron obvykle několik vstupních parametrů. Každý vstupní parametr má svou váhu, která určuje sílu spojení. Neuron kombinuje vážené vstupy a aplikuje na ně aktivační funkci, která produkuje výstupní signál, který je odesílán do dalších neuronů, nebo určen k vytvoření rozhodnutí.

Aktivační funkce neuronu je důležitá část výpočetní operace, která předurčuje, jestli má být neuron aktivován či ne, na základě vstupních hodnot. Základní aktivační funkce použity v neuronových sítích jsou sigmoid, ReLU (Rectified Linear Unit) a tanh (Hyperbolic Tangent).

Neurony v neuronových sítích mohou být organizovány do jednotlivých vrstev, které můžou obsahovat několik neuronů, produkujících vstupní signály z předešlých vrstev. Výstup poslední vrstvy neuronové sítě je celkový výstup neuronové sítě, který může být použit pro budoucí potřebné úkony.

Uvažujme neuronovou síť složenou z jednoho neuronu. Tato síť disponuje alespoň dvěma vrstvami, které jsou pojmenovány jako vstupní a výstupní.



Obr. 2. Neuronová síť, složená z jednoho neuronu se dvěma vrstvami. [10, s. 6]

Síť na Obr. 2 je popsána rovnicí:

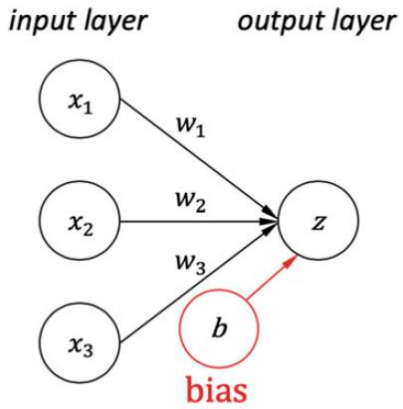
$$z = w_1x_1 + w_2x_2 + w_3x_3. \tag{3}$$

Jednotlivé váhy w mají své hodnoty závislé na důležitosti dopadu rozhodnutí. Pokud má mít rozhodnutí větší dopad, má hodnota w větší absolutní hodnotu. Oproti tomu, faktor

s menší důležitostí, by měl mít váhu w s hodnotou menší absolutní hodnotě. Pokud nastane případ, kdy je váha rovna nule, příslušný vstupní parametr je zanedbán a v procesu výpočtu nebo rozhodování je nepodstatný. Tento druh neuronové sítě je taktéž nazýván jako jednovrstvá neuronová síť nebo jako **perceptron**. [9] [10]

3.1.1 Zkreslení a Rozhodovací Hranice

Zkreslení je přidaná hodnota ve formě skaláru, která je použita ve struktuře neuronu, aby posunula hodnotu výstupu. Aplikaci zkreslení lze vidět v Obr. 3.



Obr. 3. Neuronová síť s přidaným zkreslením. [10, s. 6]

Neuronová síť v Obr. 3. je matematicky znázorněna:

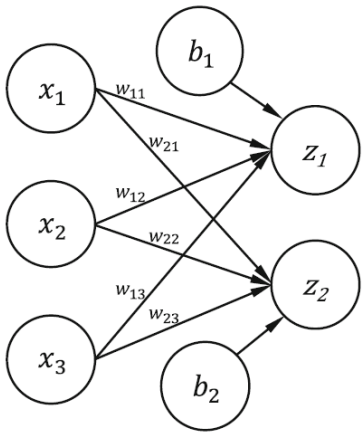
$$z = w_1x_1 + w_2x_2 + w_3x_3 + b, \tag{4}$$

kde b je zkreslení. Zkreslení pomáhá neuronové síti s přizpůsobením dat.

Rozhodovací hranice nabývá několika podob. Nejjednodušší je rozhodovací hranice pro systém se dvěma vstupy, kdy je rozhodovací hranice znázorněna přímkou, která odděluje sledovaný problém na dvě části. Pokud nastane případ s třemi a více vstupy, mění se rozhodovací hranice z přímky na plochu, která je v tomto odvětví nazývána jako **hyperplocha.** [10]

3.2 Případ s více výstupy

Jednovrstvá neuronová síť může mít několik neuronů, což znázorňuje Obr. 4, kde má jednovrstvá neuronová síť dva výstupy, které jsou popsány rovnicemi (5). Propojení každého výstupu se všemi vstupy se nazývají **hustá vrstva**, nebo **plně propojená vrstva**:



Obr. 4. Neuronová síť se třemi vstupními neurony a dvěma výstupními neurony. [10, s. 8]

$$\begin{aligned} z_1 &= w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1 \\ z_2 &= w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2 \end{aligned} \tag{5}$$

Obecným vztahem by hustá vrstva mohla být prezentována jako maticový součin:

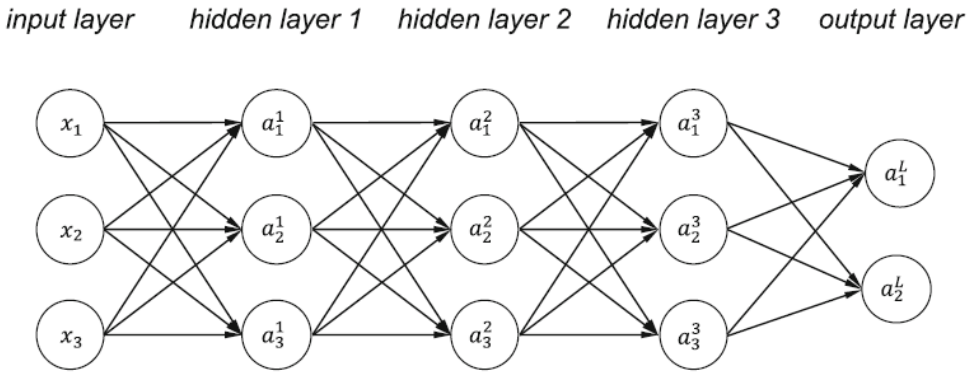
$$z = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{6}$$

kde $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ je matice znázorňující váhy a $z \in \mathbb{R}^m$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$ jsou sloupcové vektory zastupující výstupy, vstupy a zkreslení. V příkladu rovnice (5) je $m = 2$ a $n = 3$. [10]

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{7}$$

3.3 Vícevrstvý perceptron (Multilayer Perceptron)

Přidáním dvou a více hustých vrstev vzniká vícevrstvý perceptron. Obr. 5 znázorňuje vícevrstvý perceptron o čtyřech hustých vrstvách.



Obr. 5. Vícevrstvý perceptron se třemi skrytými vrstvami a jednou výstupní vrstvou. Neurony jsou značeny a_i^l, kde l je index vrstvy, a i je výstupní index. [10, s. 9]

Tři vrstvy nacházející se mezi vstupní a výstupní vrstvou jsou **skryté vrstvy**. Skryté vrstvy bývají obvykle nepřístupné vnějšímu zásahu mimo síť. Porovnáním s jednoduchou hustou sítí, vícevrstvý perceptron dokáže zpracovávat komplexnější data, jinými slovy má větší učící potenciál. [10]

Při práci s více skrytými vrstvami záleží na volbě počtu skrytých vrstev. Není pravidlo, že čím více skrytých vrstev, tím lepší výkone neuronové sítě. Stačí jedna skrytá vrstva a příslušná aktivační funkce (sigmoid nebo tanh) a systém dokáže vytvořit model, který se přibližuje jakékoliv Borelovsky měřitelné funkce. Avšak v praxi může být taková neuronová síť neforemná k učení a může být obtížné vyvarovat se *overfittingu*, který může být způsoben nadměrnou velikostí skryté vrstvy. Tudíž hluboké neuronové sítě disponující vícevrstvým perceptronem obvykle mívají více skrytých vrstev. [10] [11]

3.4 Rekurentní neuronová síť

Rekurentní neuronové sítě jsou typ umělé neuronové sítě, která je zaměřená na zpracování sekvenčních dat. Na rozdíl od dopředných neuronových sítí, které zpracovávají vstupní signál v jediném průchodu, jsou rekurentní neuronové sítě schopny uchovat paměť předešlých vstupů a tuto vlastnost využít při složitějších operacích.

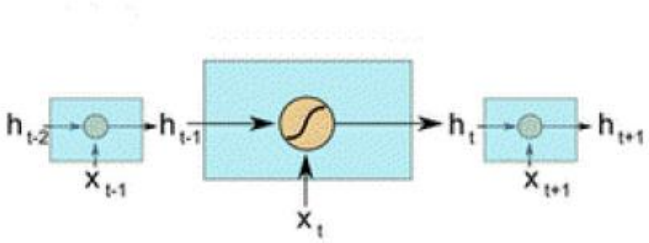
Klíčová vlastnost rekurentních neuronových sítí je užití rekurentního členu, který umožňuje přechod informace z jednoho časového kroku do dalšího. Jinými slovy, výstup sítě v čase *t* je funkce nejen vstupu v čase *t*, ale i výstupu v čase *t*-1. Ovšem může nastat problém při trénování těchto sítí nazývající se **problém mizejícího gradientu**, který je způsoben aktivitou gradientu, který aktualizuje váhy sítě a může nabývat velmi malých hodnot, které se propisují zpětně v čase a degradují výpočet. K zamezení tohoto problému byly navrhnuty modifikace rekurentních neuronových sítí a to: **long short-term memory** (LSTM) a **gated recurrent unit** (GRU).

3.4.1 Jednoduchá rekurentní neuronová síť

Jednoduchá rekurentní síť je v podstatě soubor propojených neuronových sítí, které přenášejí mezi sebou informace. Tyto sítě skladují do paměti vědomosti o datech, se kterými přišly do styku, avšak jejich paměť je krátkodobá a nedokáže udržet dlouhodobé časové sekvence. Jednoduchá rekurentní neuronová síť má pouze jednu vnitřní paměťovou jednotku - h_t:

$$h_t = g(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}_f h_{t-1} + \boldsymbol{b}), \tag{8}$$

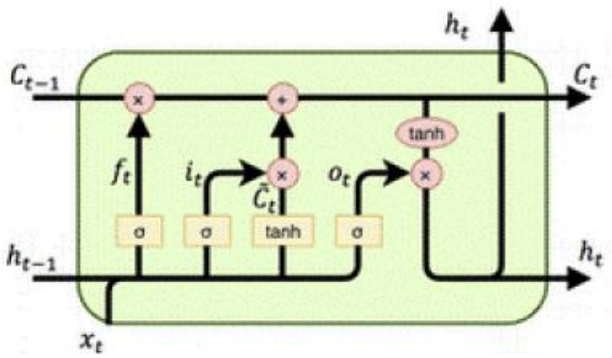
kde g je aktivační funkce, $\boldsymbol{U}$ a $\boldsymbol{W}$ jsou váhové matice h-té vrstvy, $\boldsymbol{b}$ je zkreslení a $\boldsymbol{x}$ je vstupní vektor. [12] [13]



Obr. 6. Základní struktura jednoduché rekurentní neuronové sítě. [12, s. 8]

3.4.2 Long Short-Term Memory (LSTM)

LSTM používá speciální kombinaci skrytých vrstev, násobení prvků a součtu mezi jednotkami za účelem vnoření bran, které kontrolují *paměťové buňky*. Tyto buňky mají za úkol udržet informaci po dlouhý časový úsek bez jakékoliv modifikace. Hlavní funkce LSTM je její schopnost naučit se dlouhodobou závislost systému, která není možná u jednoduchých rekurentních sítí. Pro předpověď dalšího kroku je potřeba aktualizovat váhy hodnot, což požaduje ponechávání informace z předešlých kroků.



Obr. 7. Struktura Long-Short Term Memory (LSTM) sítě. [12, s. 8]

LSTM má tři brány: vstupní, bránu zapomínání a výstupní. Brána zapomínání označuje, jak moc si předešlá paměť pamatuje a jak moc zapomněla. Pro LSTM se skrytý stav h_t počítá takto:

$$i_t = \sigma(W_i X_t + U_i h_{t-1} + b_i) \quad (9)$$

$$f_t = \sigma(W_f X_t + U_f h_{t-1} + b_f) \quad (10)$$

$$O_t = \sigma(W_O X_t + U_O h_{t-1} + b_O) \quad (11)$$

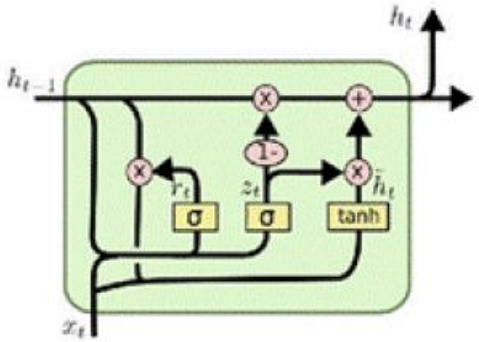
$$\widetilde{C_t} = \tanh(W_C X_t + U_C h_{t-1} + b_C) \quad (12)$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \quad (13)$$

$$h_t = \tanh(C_t) * O_t, \quad (14)$$

kde i_t je vstupní brána, f_t je brána zapomínaní a O_t je výstupní brána v čase t. W_i, W_f, W_O a W_C jsou vstupní skryté váhy bran. U_i, U_f, U_O a U_C jsou výstupní skryté váhy bran. $b_i, b_f . b_0$ a b_C jsou vektory. C_t a h_t jsou výstupy buňky a vrstvy. [12] [14]

3.4.3 Gated Recurrent Unit (GRU)



Obr. 8. Struktura Gated recurrent unit (GRU) sítě. [12, s. 8]

GRU je zjednodušený řešič vycházející z LSTM. Rozdíl mezi LSTM a GRU je, že GRU slučuje vstupní bránu a bránu zapomínání a převádí je na aktualizační bránu. Z čehož vyplývá, že má GRU méně parametrů než LSTM, což znamená snadnější trénování. Pro GRU je výstupní hodnota h_t vypočítaná takto:

$$z_t = \sigma(W_z X_t + U_z h_{t-1} + b_z) \quad (15)$$

$$r_t = \sigma(W_r X_t + U_r h_{t-1} + b_r) \quad (16)$$

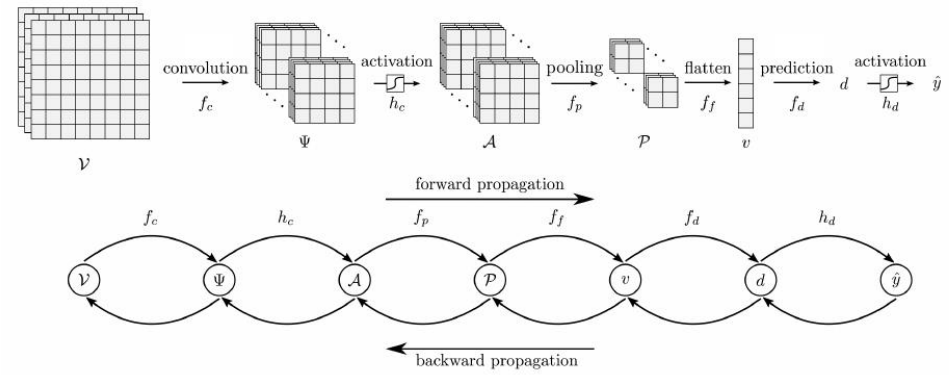
$$\widetilde{h_t} = \tanh(W_h X_t + (r_t * h_{t-1}) U_h) \quad (17)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t + z_t * \widetilde{h_t}, \quad (18)$$

kde *r* je resetovací brána a z zastupuje aktualizační bránu. Resetovací brána označuje, jak může být nový vstup kombinován s dřívější pamětí. Aktualizační brána také indikuje, jak moc je předešlá paměť uchována. Pokud je aktualizační brána rovna 1, předešlá paměť je plně uchována a pokud je 0, předešlá paměť je kompletně zapomenuta. V LSTM je brána zapomínání, která automaticky předurčí, kolik je předešlé paměti uchováno, zatímco v GRU všechny předešlé paměti jsou buď zachovány nebo kompletně zapomenuty. Pro určité problémy je GRU schopno poskytnout výkon porovnatelný s LSTM, s výhodou využití méně paměti. [12] [14]

3.5 Konvoluční neuronové sítě (Convolutional Neural Networks – CNN)

Konvoluční neuronové sítě označují velkou část strojového učení, které využívá speciální konvoluční bloky k získání funkční informace z datových objektů. Původně byly konvoluční neuronové sítě navrženy k získání dat ze síťových objektů, ale aktuálně jsou používány i k získávání informací z grafů. Hlavní a klíčovou úlohou konvolučních neuronových sítí je **konvoluce**. Cíl konvolučních neuronových sítí je získání optimálních filtrů nebo operátorů, které maximalizují výstupní informaci sledovaného systému s předem danými vstupními daty.



Obr. 9. Popis struktury konvoluční neuronové sítě. Na vstupní data jsou postupně použity sekvence konvoluce, aktivace a sdružovacích operací, které v konečném výsledku vytvoří výstupní předpověď. Parametry operací jsou učeny pomocí minimalizace ztrátové funkce. Derivace/gradient ztrátové funkce je počítán zpětnou propagací. [15, s. 8]

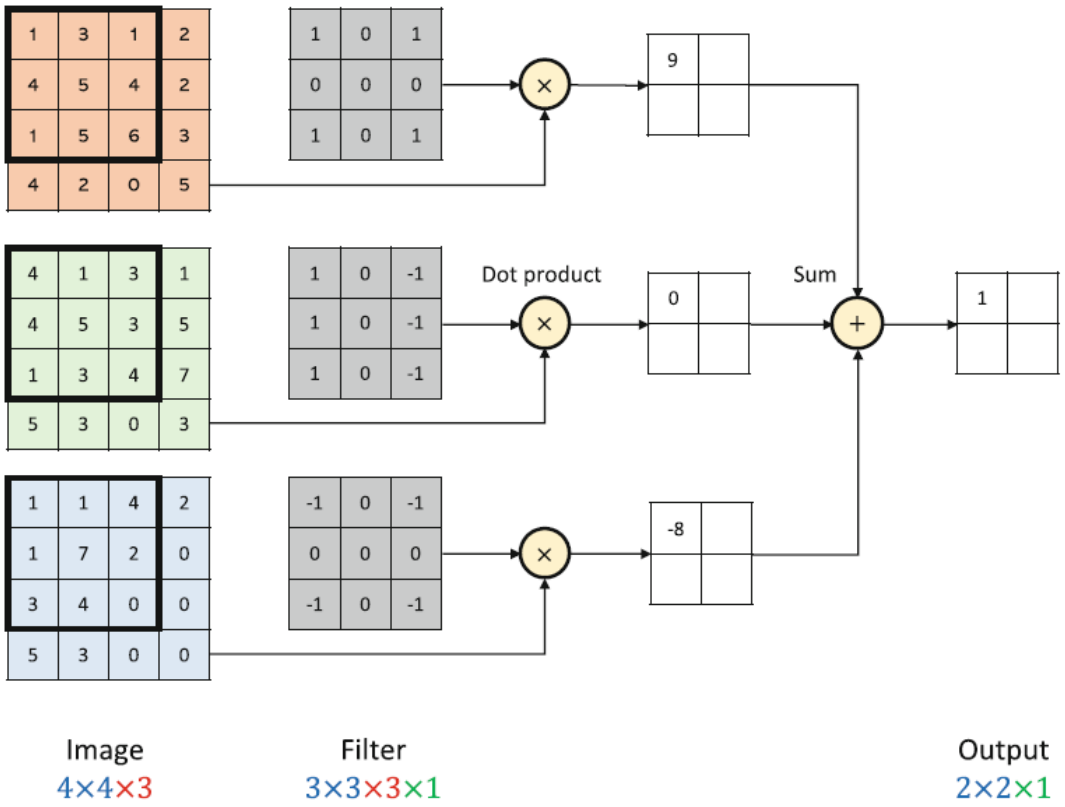
Struktura konvolučních neuronových sítí je dopředná propagace vstupních dat přes vrstvy konvoluce, aktivace a sdružovacích operací, jejichž parametry jsou učeny pomocí minimalizace ztrátové funkce. Zpětná propagace je použita pro výpočet gradientu ztrátové funkce s ohledem na parametry. [15]

Následující kapitola bude popisovat jednotlivé kroky řešení konvolučních neuronových sítí, spolu s matematickým vyjádřením.

4. Struktura výpočtu konvolučních neuronových sítí

4.1 Konvoluční vrstva

Proces konvoluce je vykonáván konvolučním operátorem, který bývá menší než vstupní matice dat. Konvoluční operátor je postupně používán a posouván přes celé vstupní data. Tato operace není dobře definovatelná, tudíž je zapotřebí přidat výplň kolem okrajů sledované soustavy, která obsahuje nuly, nazýváme *nulová výplň*. Konvoluční operátor pronásobuje vstupní data a vytváří jejich vážený výsledek. Tyto operátory jsou trénovány podle potřeby výstupních dat. [15]



Obr. 10. Ilustrace konvoluční operace. Filtr o velikosti 3×3×3 je aplikován na vstupní data o velikosti 4×4 se třemi vstupními kanály. Je vypočítán skalární součin mezi vstupními daty a filtry. Hodnoty získané skalárním součinem jsou sečteny a tvoří výstup v levém horním rohu. [10, s. 28]

Výpočet velikosti výstupu v Obr. 10 je vypočítán pomocí vzorce:

$$\left[\frac{W - F + 2P}{S} + 1\right], \tag{19}$$

kde je vstupní matice o velikosti výšky/šířky W. F je velikost filtru. S znázorňuje krok vzdálenosti jednotlivých vstupních bloků. P označuje velikost nulové výplně. [10]

Uvažujme případ z Obr. 10. Zde je vstupní matice o velikosti 4×4 se třemi vstupními kanály a jeden filtr o velikosti 3×3×3 s krokem $S = 1$ a výplní $P = 0$. Podle rovnice (19) je výstupní matice o velikosti 2×2:

$$\frac{4 - 3 + 0}{1} + 1 = 2.$$

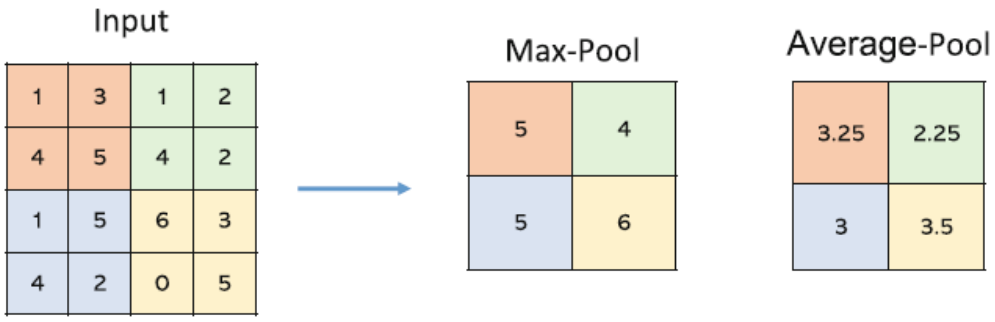
4.2 Sdružovací vrstva (Pooling layer)

Sdružovací vrstva cílí k postupné redukci rozměru sledované soustavy. Je redukován počet parametrů, a tudíž i výpočetní komplexita modelu.

Sdružovací vrstva rozlišuje dva hlavní druhy: maximální sdružovací vrstva (max-pooling layer) a průměrová sdružovací vrstva (average-pooling layer). Podle názvu lze odvodit, že se jedná u maximální sdružovací vrstvy o algoritmus, který z dané oblasti vybere maximální hodnotu, zatímco u průměrové sdružovací vrstvy se jedná o průměr celé sledované oblasti.

Vzhledem k destruktivní povaze sdružovací vrstvy bývá obvykle volen rozměr sdružovacího filtru jako 2×2 a krok posuvu je 2.

Dále lze použít **překrývající sdružování**, které má krok posuvu 2 a rozměr filtru 3×3. Vzhledem k redukční povaze sdružovací vrstvy, použití většího filtru, než je 3×3, znamená snížení výkonu modelu. [16]



Obr. 11. Sdružovací proces s filtrem 2×2 a krokem posuvu 2. Maximální sdružovací vrstva vybírá ze sledované oblasti maximální hodnotu. Průměrová sdružovací vrstva počítá průměr ze sledované oblasti. [10, s. 28]

4.3 Aktivační funkce

Pro lineární modely je schopnost učit se velmi omezená, protože mají problém s aproximací nelineárních funkcí. Většinu reálných systému nelze zjednodušit na lineární rovinu, tudíž je zapotřebí uvést do modelu nelinearitu. Nelinearita je do neuronové sítě uvedena právě pomocí aktivační funkce. Aktivační funkce je potřeba i v případě, kdy je zapotřebí získat vektor pravděpodobnosti namísto vektoru s libovolnou hodnotou.

Na výběr je mnoho aktivačních funkcí a nedá se říct, která je nejlepší. Každá aktivační funkce je specifická pro určitý problém, tudíž volba závisí na konkrétní podobě řešené úlohy. Základní aktivační funkce jsou: sigmoid, tanh, ReLU a softmax.

4.3.1 Sigmoid

Aktivační funkce sigmoid má výstup v rozpětí mezi 0 a 1. Tato funkce může být použita pro klasifikační rozdělení, například jako binární rozdělení, kdy se rozhoduje mezi výstupní hodnotou 0 a 1 a hranice rozeznání je pro tento případ 0,5. [17]

$$f(z) = \frac{1}{1 + e^{-z}} \tag{20}$$

4.3.2 Tanh (hyperbolický tangent)

Hyperbolický tangent je podobný funkci sigmoid. Jeho výstupní hodnoty se pohybují v rozmezí -1 a 1. Funkce tanh se používá ve skrytých funkcích, kdy vnáší do systému nelinearitu. Oproti funkci sigmoid je tanh symetrický okolo nuly, tudíž má výhodu v tom, že výstup může být kladný i záporný. [17]

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{21}$$

4.3.3 Softmax funkce

Softmax funkce počítá normalizaci mezi všemi výstupy z výstupů z předešlých vrstev. Jako první vypočítá exponenciální funkci e^{z} a poté každý vstup normalizuje dělením. V praxi je softmax funkce obvykle používaná ve výstupní vrstvě, aby normalizovala výstupní vektor z na pravděpodobnostní vektor, kde každý vstup je kladný a vstupy jsou v součtu rovny jedné. Tudíž je softmax funkce využívaná pro klasifikaci. [18] [19]

$$f(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{22}$$

4.3.4 ReLU (rectified linear unit)

Usměrňovací lineární jednotka, také známa jako usměrňovač, je definovaná rovnicí (23). Výhodou oproti funkcím sigmoid a tanh je jednoduchost funkce ReLU. Její jednodušší zařazení a výpočet systému, kdy porovnává vstup s nulou, a buď je výstup rovný 0 nebo vstupní hodnotě. Kdežto ve funkcích sigmoid a tanh jsou exponenciální funkce složité na výpočet v případě rozsáhlých sítí. Funkce ReLU se blíží linearitě, tudíž je pro ni jednodušší výpočet derivace. Derivace aktivního neuronu je vždy rovna jedné, za což u funkcí sigmoid a tanh může nastat případ mizejícího gradientu, pokud se aktivační hodnoty blíží limitně k jejím hranicím.

$$f(z) = \begin{cases} 0 \text{ když } z \leq 0 \\ z \text{ když } > 0 \end{cases} \tag{23}$$

Avšak výstup negativních hodnot rovný 0 může vést ke ztrátě informací, což může být způsobeno nevhodným nastavením učení, nebo záporným zkreslením. Tento problém může být eliminován vytvořením *leaky ReLU* (unikající usměrňovací lineární jednotka) popsána vzorcem (22).

$$f(z) = \begin{cases} \alpha z \text{ když } z \leq 0 \\ z \text{ když } > 0 \end{cases} \quad (24)$$

Skalár α je malá pozitivní hodnota kontrolující sklon (číselně 0,01 nebo 0,02), aby byla uchována malá informace o negativní hodnotě.

Parametric ReLU (parametrická usměrňovací lineární jednotka) je podobná leaky ReLU, avšak považuje hodnotu α, jako parametr, který se dá trénovat. [20] [21]

4.3.5 Swish funkce

Swish je poměrně nová aktivační funkce. Je definována rovnicí (25), kde σ je aktivační funkce sigmoid, díky které můžeme rozepsat aktivační funkci Swish rovnicí (26):

$$f(z) = z \cdot \sigma(\beta z) \quad (25)$$

$$f(z) = \frac{z}{1 + e^{(-\beta z)}}, \quad (26)$$

kde β může být konstanta nebo trénovatelný parametr. Pro různé hodnoty parametru β nabývá Swish funkce jednotlivé podoby. Pokud $\beta = 0$, má Swish funkce tvar:

$$f(z) = \frac{z}{2},$$

tudíž se z ní stává lineární funkce. Ale pokud uvažujeme $\beta \to \infty$, potom se komponent sigmoid blíží k 0-1 funkci, což u aktivační funkce Swish vytváří stejné chování, jako u aktivační funkce ReLU. Funkce Swish je obsáhlá v knihovně TensorFlow. [22]

4.4 Ztrátová funkce

Ztrátová funkce posuzuje chybu mezi předpovědí a cílem. Ztrátová hodnota je používaná jako předmět optimalizace parametrů neuronových sítí, jako jsou váhy nebo zkreslení. V následující části budou popsány jednotlivé základní ztrátové funkce.

4.4.1 Binární ztrátová funkce Cross-Entropy

V binární klasifikaci pro každý vstup x_i s cílem y_i potřebuje model předpovědět pravděpodobnost každého z kandidátů tříd $\hat{y}_{i,1}$ a $\hat{y}_{i,2}$. Počítá se zde s pravděpodobností a jedná se o binární systém, tudíž platí $\hat{y}_{i,1} + \hat{y}_{i,2} = 1$. Předpověď může být zapsána jako $\hat{y}_i$, což reprezentuje pravděpodobnost jedné třídy a pravděpodobnost druhé třídy bude $1 - \hat{y}_i$. A proto má neuronová síť pro binární klasifikaci obvykle jeden výstupní neuron a následující definici cross-entropy funkce:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)), \quad (27)$$

kde N zastupuje celkový počet vzorků dat. Vzhledem k tomu, že se jedná o binární klasifikaci, cíl y_i může být buď 0 nebo 1. Což znamená uchování pouze jedné volby z $y_i \log \hat{y}_i$ nebo $(1 - y_i)\log(1 - \hat{y}_i)$ pro jednotlivé vzorky dat. Pokud pro každé i, $y_i = \hat{y}_i$, je ztrátová funkce cross-entropy rovna nule. [18]

4.4.2 Ztrátová funkce cross-entropy pro více třídovou klasifikaci

Ve více třídové klasifikaci je každý vzorek dat x_i přiřazen do jedné ze tří a více kandidátních tříd. Model předpovídá pravděpodobnost každé třídy $\{\hat{y}_{i,1}, \hat{y}_{i,2}, \dots, \hat{y}_{i,M}\}$, kde $M \geq 3$ a $\sum_{j=1}^{M} \hat{y}_{i,j} = 1$. Cíl jednotlivých vzorků dat je označen jako c_i, což je celé číslo v intervalu mezi $[1, M]$ a může být převeden na vektor $y_i = [y_{i,1},\ y_{i,2}, \dots, y_{i,M}]$, kde může být pouze $y_{i,c_i} = 1$ a ostatní hodnoty rovny 0. Poté může být napsána rovnice cross-entropy pro více třídou klasifikaci jako: [18]

$$\begin{aligned} \mathcal{L} &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} \log \hat{y}_j = \\ &= -\frac{1}{N}\sum_{i=1}^{N}(0 + \cdots + y_{i,c_i} \log \hat{y}_{c_i} + \cdots + 0) = \\ &= -\frac{1}{N}\sum_{i=1}^{N} \log \hat{y}_{c_i}. \end{aligned} \tag{28}$$

4.4.3 $\mathcal{L}_p$ Norma

Zadaný vektor $\boldsymbol{x}$ má p-normu, která měří jeho velikost tak, že vektor s většími hodnotami má větší velikost. P-norma je definovaná následovně, kde p je číslo větší nebo rovno jedné:

$$\|x\|_p = \left(\sum_{i=1}^{N} |x_i|^p\right)^{1/p}, \tag{29}$$

což může být zapsáno jako:

$$\|x\|_p^p = \sum_{i=1}^{N} |x_i|^p \tag{30}$$

V hlubokém učení může být p-norma použita při měření rozdílu vzdáleností mezi dvěma vektory. Značeno jako $\mathcal{L}_p$, kde $\boldsymbol{y}$ je cíl a $\hat{\boldsymbol{y}}$ je predikce.

$$\mathcal{L}_p = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_p^p = \sum_{i=1}^{N} |y_i - \hat{y}_i|^p. \tag{31}$$

4.4.4 Střední kvadratická chyba (Mean Squared Error)

Střední kvadratická chyba je průměrná $\mathcal{L}_2$ norma definována rovnicí (32). Střední kvadratická chyba může být použita v regresních problémech, ve kterých je výstup neuronových sítí spojitá hodnota.

$$\mathcal{L} = \frac{1}{N}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{32}$$

kde *N* je počet vzorků dat, y_i je cíl a $\hat{y}_i$ je predikce. [18]

4.4.5 Střední absolutní chyba (Mean Absolute Error)

Stejně jako střední kvadratická chyba může být použita střední absolutní chyba pro regresní problémy. A je definována jako průměrná $\mathcal{L}_1$ norma:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \tag{33}$$

kde *N* je počet vzorků dat, y_i je cíl a $\hat{y}_i$ je predikce.

Střední kvadratická chyba a střední absolutní chyba minimalizují rozdíl mezi cílem a predikcí. Střední kvadratická chyba nabízí lepší matematické řešení kvůli parciálním derivacím, které jsou potřeba pro algoritmus *gradient descent*, který není možný u střední absolutní chyby, protože absolutní hodnota v případě $y_i = \hat{y}_i$ nelze derivovat. V případě, kdy je rozdíl mezi hodnotami y_i a $\hat{y}_i$ větší než 1, má kvůli kvadratickému členu střední kvadratická chyba větší hodnotu chyby než střední absolutní chyba, což může způsobit rychlejší optimalizaci sítě. [18]

4.5 Optimalizační algoritmy

Optimalizační algoritmy jsou v neuronových sítích používány především pro úpravu vah, zkreslení a parametrů. Cíl optimalizačního algoritmu je minimalizace ztrátové funkce, která je popsána v *kapitole 4.4.*

Mezi základní optimalizační algoritmy, které budou následně popsány, patří: Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSProp), Adam, Adagrad a Adadelta. Volba jednotlivých optimalizačních algoritmů závisí na specifickém problému řešené soustavy.

4.5.1 Stochastic Gradient Descent (SGD)

Algoritmus Stochastic Gradient Descent je stochastická aproximace algoritmu *gradient descent* [23] a iterativní metody pro minimalizaci nebo maximalizaci cílové funkce. Na rozdíl od algoritmu gradient descent, SGD nepočítá s celým setem dat, ale pouze s částmi (dávkami) nebo jednotlivci. Díky tomuto je systém méně náročný na výpočetní výkon a zrychluje čas výpočtu. Kromě toho může SGD dynamicky upravovat odhad matic prvního a druhého řádu gradientu každého parametru podle ztrátové funkce, což redukuje možnost konvergence modelu na lokální minimum/maximum.

Uvažujme soubor dat *S*. $x_i \in R^n$ je *n*-dimenzionální vektor. $y_i \in \{1,\ m-1\}$ je kategorie *i*-tého trénovacího vzorku. Jako první je přiřazen nulový vektor k hodnotě váhy W_1 a náhodně zvolen trénovací vzorek (x_i, y_i) z celého trénovacího setu, kde $i_t \in \{1, \dots, m\}$ je cíl zvolného trénovacího vzorku v *t*-té iteraci. Objektivní funkce je:

$$\min(W) = \frac{\lambda}{2}\|W\|^2 + f(W, (x_i, y_i)). \tag{34}$$

Dalším krokem je výpočet gradientu rovnice (34), které může vypadat takto:

$$\nabla_t = \lambda W_t - \alpha_t y_{i_t} x_{i_t}, \tag{35}$$

kde $\alpha_t = \begin{cases} 1, & \text{jestli } y_{i_t}\langle W_t, x_{i_t}\rangle < 1 \\ 0, & \text{v opačném případě} \end{cases}$

Aktualizační rovnice matice *W* je:

$$W_{t+1} = W_t - \eta_t \alpha_t, \tag{36}$$

kde $\eta_t = \frac{1}{(\lambda t)}$.

Poté může být aktualizace vah matice *W* získána pomocí rovnic (35) a (36) vzorcem:

$$W_{t+1} = \left(1 - \frac{1}{t}\right) W_t + y_{i_t} x_{i_t}. \tag{37}$$

Rovnice (37) je použita pro hledání minima či maxima pomocí iterací. [24]

4.5.2 RMSProp (Root Mean Square Propagation)

Algoritmus RMSProp je založený na algoritmu SGD. RMSProp rychle snižuje ztrátovou funkci upravováním rychlosti učení SGD. Problém u tohoto algoritmu je v oscilaci, která může vytvářet šum a v pomalé konvergenci, kdy mají gradienty velký rozptyl.

Algoritmus SGD, ze kterého následně vychází RMSProp, může být znázorněn rovnicí:

$$\theta_{i,t} = \theta_{i,t-1} - \alpha \nabla f\left(\theta_{i,t-1}; x_{t-1}\right), \tag{38}$$

kde je cíl minimalizovat ztrátovou funkci $f(\theta)$ iterativní aktualizací θ pomocí minisérií vzorků. Kde α je míra učení, $\theta_{i,t}$ je *i*-tý element parametrového vektoru v čase *t*, x_{t-1} je vzorek minisérie v čase $t-1$ a $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ je gradient prvního řádu, který dbá na *i*-tý parametr získaný z x_{t-1}.

RMSProp používá aktualizační pravidlo:

$$v_{i,t} = \beta v_{i,t-1} + (1-\beta) \nabla f\left(\theta_{i,t-1}; x_{t-1}\right)^2 \tag{39}$$

$$\theta_{i,t} = \theta_{i,t-1} - \frac{\alpha}{\sqrt{v_{i,t}} + \epsilon} \nabla f\left(\theta_{i,t-1}; x_{t-1}\right), \tag{40}$$

kde $v_{i,t}$ je pohyblivý průměr necentrovaného rozptylu posledního gradientu prvního řádu $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$, β je míra úbytku pro výpočet $v_{i,t}$ a ϵ je malá hodnota pro stálý výpočet.

RMSProp dělí míru učení α velikostí $\sqrt{v_{i,t}}$ posledního gradientu prvního řádu $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$. Tudíž, pokud má *i*-tý parametr velkou hodnotu $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ v minulosti, RMSProp produkuje malou míru učení, protože je v rovnici (40) výraz $\sqrt{v_{i,t}}$ velký. Tato vlastnost efektivně redukuje ztrátovou funkci neuronové sítě. [25]

4.5.3 Adam (Adaptive Moment Estimation)

Adam (Adaptivní odhad momentů) vychází z algoritmu SGD. Je vhodný pro zpracování velkého množství dat a optimalizaci nestacionárních cílových funkcí, které disponují šumem a řídkými gradienty.

Algoritmus Adam provádí postupné optimalizace náhodné objektivní funkce. Adam používá aktualizační pravidlo:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \tag{41}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{42}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{43}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{44}$$

kde θ_t a θ_{t+1} jsou objektivní funkce, t je časový parametr, $\beta_1, \beta_2 \in \langle 0,1)$ představuje rychlost úpadku pohyblivého průměrného indexu, η je míra učení, ε je konstantní parametr, $\hat{m}_t$ a $\hat{v}_t$ jsou odhady momentů prvního a druhého řádu po úpravě gradientu. Pokud jsou m_t a v_t inicializovány jako nulový vektor, budou posunuty na nulu. Je zapotřebí opravit tuto odchylku rovnicemi (44). [26] [27]

4.5.4 Adagrad

Adagrad je algoritmus založený na SGD. Upravuje míru učení vůči parametrům, provádějící velké změny pro nepříliš časté parametry a malé změny pro frekventované parametry. Díky tomu je tento algoritmus výhodný pro řídká data.

Adagard používá různé míry učení pro každý parametr θ_i v každém časovém kroku t. Uvažujme $g_{t,i}$ jako gradient objektivní funkce vzhledem k parametru θ_i v časovém kroku t:

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{45}$$

SGD algoritmus aktualizuje každý parametr θ_i v každém časovém kroku rovnicí:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \tag{46}$$

V tomto aktualizačním pravidlu, Adagrad upravuje hlavní míru učení η v každém časovém kroku t pro každý parametr θ_i založený na předešlých gradientech, které byly vypočítány pro θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}, \tag{47}$$

kde $G_t \in \mathbb{R}^{d \times d}$ je diagonální matice, kde každý diagonální element i, i je součet čtverců gradientů parametrů θ_i až do časového kroku t, ϵ je vyrovnávací člen, který zabraňuje dělením nulou.

Tím, že G_t obsahuje součet čtverců předešlých gradientů všech parametrů θ podél jeho diagonály, je možné provést násobení matice a vektoru po jednotlivých prvcích:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{48}$$

Hlavní nevýhoda algoritmu Adagard je hromadění čtverců gradientů ve jmenovateli, jelikož je každý přidaný výraz pozitivní, tak součet neustále narůstá v průběhu trénování. Díky tomu se může míra učení zmenšovat až do takové úrovně, kdy algoritmus přestane mít schopnost dále se učit. Algoritmus Adadelta tuto vadu umí vyřešit. [28]

4.5.5 Adadelta

Algoritmus Adadelta vychází z algoritmu Adagrad. Jeho cílem je vylepšení dvou hlavních problémů algoritmu Adagrad: 1) neustály pokles míry učení během trénování a 2) potřeba manuálního zvolení míry učení.

V metodě Adagrad jmenovatel akumuluje čtverce gradientů z každé iterace vzniklé v průběhu učení, které následně zmenšují míru učení. Místo toho Adadelta používá fixní okno o velikosti w, kde se nachází určitý počet předešlých gradientů (namísto velikosti t, která značí aktuální iteraci v algoritmu Adagrad). S touto úpravou ve formě shlukování do okna nemůže jmenovatel Adagradu sčítat gradienty donekonečna, ale stane se lokálním, používajícím aktuální gradienty. Toto ochrání budoucí pokračování progresu i po mnoha aktualizačních iteracích. [29]

Místo skladování w předešlých druhých mocnin gradientů, je součet gradientů rekurzivně definován jako klesající průměr předchozích druhých mocnin gradientů. Běžící průměr $E[g^2]_t$ v časovém kroku t záleží pouze na předešlém průměru a současném gradientu.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2, \tag{49}$$

kde γ je ztrátová konstanta. Parametrový aktualizační vektor $\Delta\theta_t$ převzatý z algoritmu SGD:

$$\Delta\theta_t = -\eta \cdot g_{t,i}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Výměnou diagonální matice G_t algoritmu Adagrad rovnice (48), klesajícím průměrem předchozích druhých mocnin gradientů $E[g^2]_t$ obdržíme rovnici:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{50}$$

Jelikož je výraz $\sqrt{E[g^2]_t + \epsilon}$ kritérium střední kvadratické chyby (RMS) gradientu, můžeme jej nahradit zkratkou kritéria:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \tag{51}$$

V [28] byla otevřena otázka ohledně nesrovnalosti v jednotkách algoritmu, kdy by měl mít aktualizační člen stejné jednotky jako parametr. Tímto byl definován nový klesající průměr, který obsahoval místo čtverců gradientů, čtverce parametrů:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \tag{52}$$

Root mean squared error (RMS) v překladu odmocnina průměrného čtverce chyby je:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \tag{53}$$

Hodnota $RMS[\Delta\theta]_t$ je neznámá, tudíž je přibližována pomocí RMS aktualizací parametrů až do předchozího časového kroku. Výměnou míry učení η v předešlém aktualizačním pravidlu, hodnotou $RMS[\Delta\theta]_{t-1}$, vede k aktualizačnímu pravidlu Adadelta:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{54}$$

U algoritmu Adadelta není třeba nastavovat výchozí hodnotu míry učení, protože byla vyřazena z aktualizačního pravidla v rovnici (54).

4.5.6 Adamax

V optimalizačním algoritmu Adam je aktualizační pravidlo pro jednotlivé váhy určeno tak, aby byly jejich gradienty nepřímo úměrně přizpůsobeny L^2 normě jejich momentálních a předešlých gradientů. V algoritmu Adamax je generalizováno L^2 normované aktualizační pravidlo na L^p normované aktualizační pravidlo. Pro velkou hodnotu p začínají být algoritmy numericky nestabilní, avšak pro případ, kdy $p \to \infty$, nastává změna v stabilitě algoritmu. Uvažujme časový krok t nepřímo úměrný k $v_t^{1/p}$, kde:

$$v_t = \beta_2^p v_{t-1} + \left(1 - \beta_2^p\right)|g_t|^p, \tag{55}$$

$$v_t = \left(1 - \beta_2^p\right) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p. \tag{56}$$

Uvažujme $p \to \infty$ a definujme $u_t = \lim_{p\to\infty} (v_t)^{1/p}$, kde:

$$u_t = \lim_{p\to\infty} (v_t)^{1/p} = \lim_{p\to\infty} \left(\left(1 - \beta_2^p\right) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p \right)^{1/p}, \tag{57}$$

kde po úpravách vzniká jednoduchá rekurzivní formule:

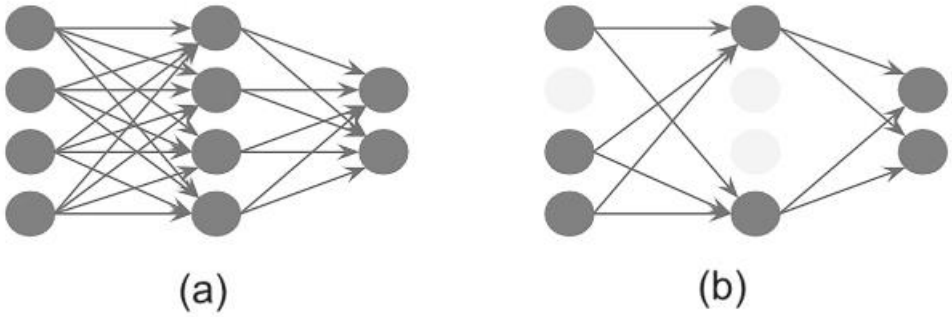
$$u_t = \max(\beta_2 \cdot u_{t-1}, |g_t|), \tag{58}$$

s počáteční hodnotou $u_0 = 0$. [30]

4.6 Dropout

Dropout je technika k zamezení overfittingu. Hlavní myšlenka je náhodné odebírání jednotek z každé trénovací dávky modelu, u kterého již došlo k overfittingu. Opakujícím se odebíráním náhodných vzorků, nabývá model na robustnosti.

Dropout rate je parametr, který udává počet dat, které budou eliminovány. Pro vstupní vrstvu bývá dropout rate 0,1 a pro vnitřní vrstvy bývá mezi 0,5 a 0,8. [31]



Obr. 12. Dropout funkce náhodně odebere jednotky z plně propojené sítě (a) a vzniká sub-model (b) [31]

4.7 Batch normalization

Batch normalization je použit k minimalizaci overfitingu, zvýšení generalizace a pomáhá modelu rychleji konvergovat, díky tomu je možné zvýšit rychlost učení, což zapříčiňuje zkrácení času výpočtu.

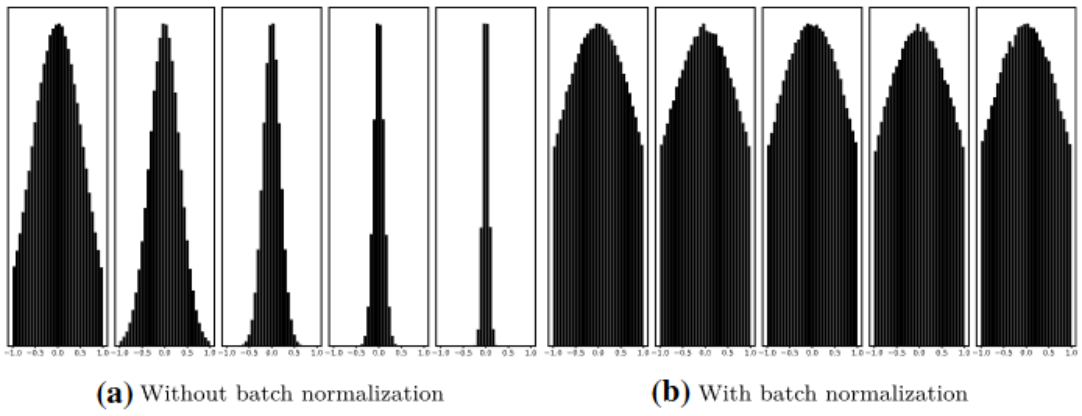
Batch normalization je vhodný pro menší velikosti dávek dat. Ošetřuje proměnlivost sledované soustavy (explodující a zanikající problém) a normalizuje hodnoty všech vrstev neuronové sítě.

V obr. 13 lze pozorovat v případě (a), že síť konverguje k nule. S pomocí batch normalization (b) každá vrstva znovu nastaví rozdělení vstupů do aktuální vrstvy.

Batch normalization funkce je umisťována před aktivační funkci. K realizaci je potřeba výpočet průměru a směrodatné odchylky, poté jsou vypočítány výsledky standardizace. Po standardizaci je vypočten batch normalization pomocí rovnice:

$$\hat{z}^i = \gamma * \tilde{z}^i + \beta(x), \tag{59}$$

kde γ a β jsou hyperparametry, které se dají upravovat za účelem lepších výsledků. [32] [33]



Obr. 13. Průběh neuronové sítě znázorněn v jednotlivých vrstvách postupujících zleva doprava, kdy (a) není použit batch normalization, (b) batch normalization je použit [32]

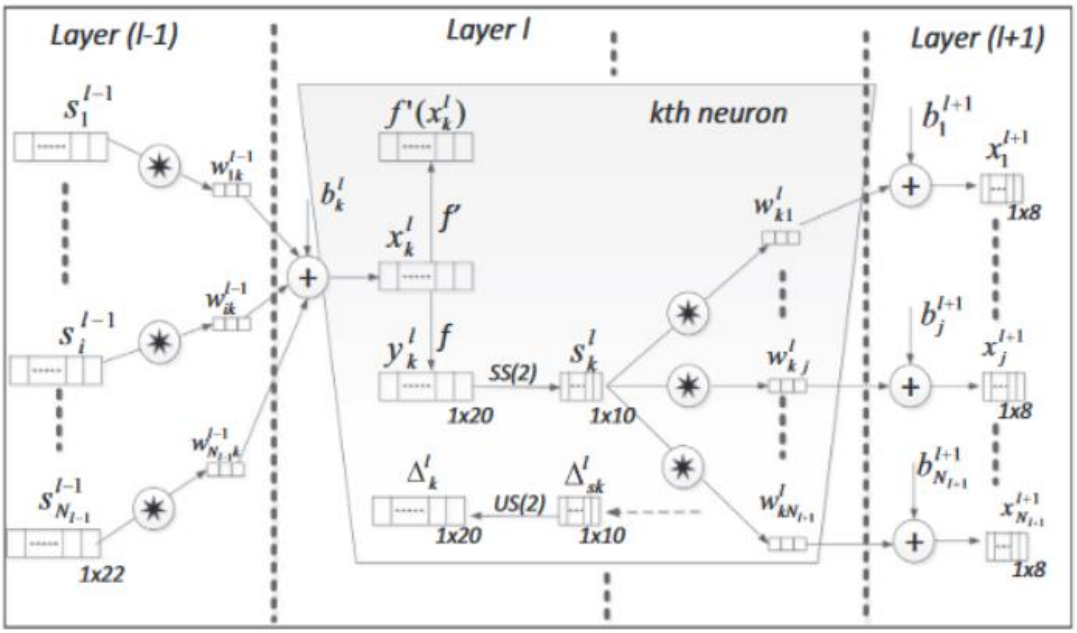
5. 1D Konvoluční neuronové sítě (1D CNN)

Běžné konvoluční neuronové sítě jsou konstruovány pro 2D signál, jako může být například obrazový signál či video signál. Modifikovaná verze 2D konvolučních sítí jsou 1D konvoluční neuronové sítě. Jejich výhoda je ve výpočetní náročnosti a menší náročnost na počet parametrů.

V [34] je 1D konvoluční neuronová síť popsána dvěma typy vrstev: 1) konvoluční vrstva, která obsahuje konvoluci, aktivační funkci a sdružovací mechanismus, 2) plně propojená vrstva, která může být jako vícevrstvý perceptron.

Konfigurace 1D konvoluční neuronová síť je formována těmito hyper-parametry:

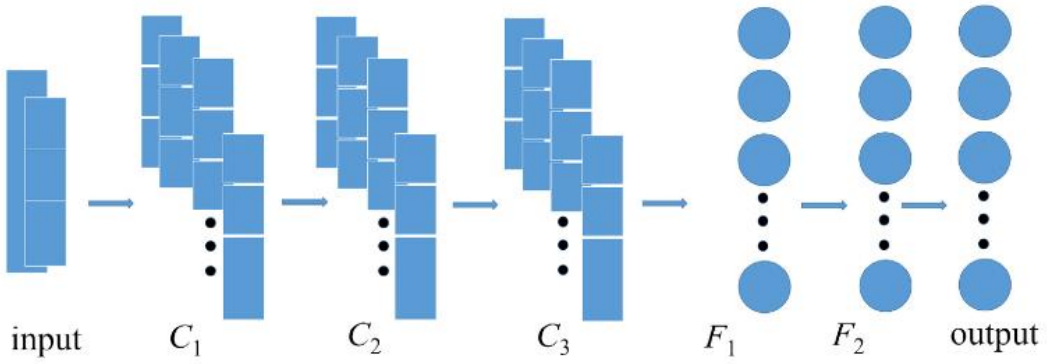
1) Počet skrytých konvolučních neuronových sítí a vícevrstvých perceptronů/neuronů.
2) Velikostí filtru (kernelu) v každé konvoluční vrstvě.
3) Sdružovací faktor.
4) Volba sdružovacích a aktivačních funkcí.



Obr. 14. Tři po sobě jdoucí skryté vrstvy typu CNN v 1D CNN. [35]

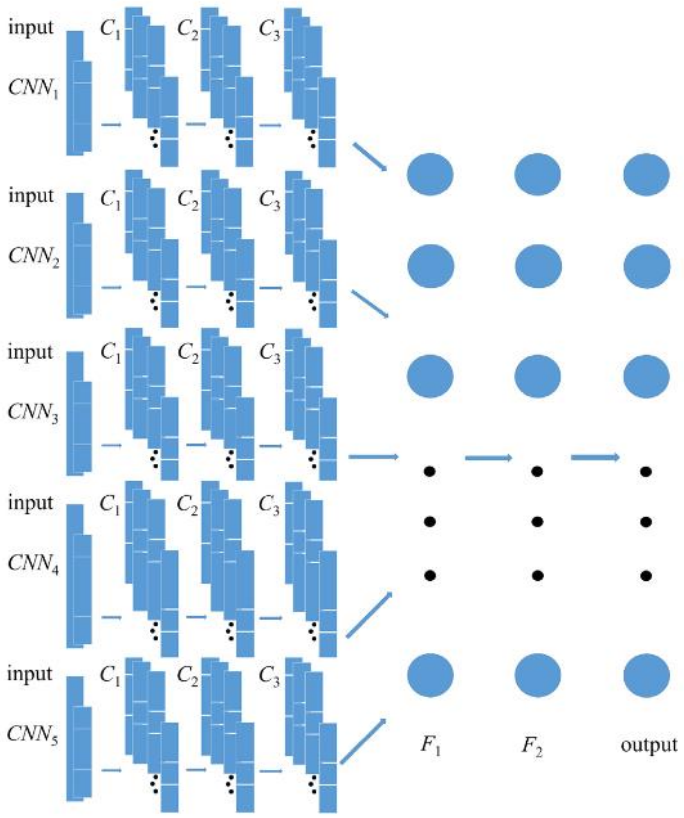
Stejně jako v 2D konvolučních neuronových sítích, tak i v 1D konvolučních neuronových sítích je vstupní vrstva pasivní, která přijímá syrová data a výstupní vrstva je vícevrstvý perceptron obsahující stejný počet neuronů jako tříd. V Obr. 14 jsou zobrazeny tři po sobě jdoucí skryté vrstvy CNN. 1D filtr má velikost 3 a velikost faktoru pro *sub-sampling* je 2, kde *k*-tý neuron skryté CNN vrstvy *l*, jako první provede sekvenci konvolucí jejichž součet prochází aktivační funkcí *f* a následnou sdružovací operací.

V [36] je navržena 1D konvoluční neuronová síť obsahující tři konvoluční vrstvy (C_1, C_2, C_3) a dvě plně propojené vrstvy (F_1, F_2). Vstupní vrstva je postupně následovaná 1D C_1, C_2, C_3 s C_3 připojenou k F_1 následovanou F_2 s výstupní vrstvou na konci. Výstupní vrstva je softmax vrstva obsahující počet neuronů stejný, jako počet tříd.

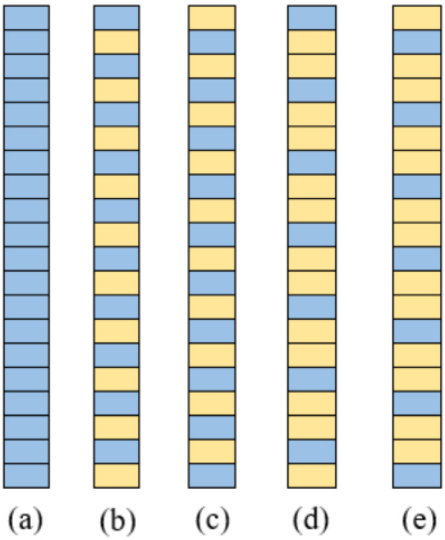


Obr. 15. Architektura jednoduché konvoluční neuronové sítě. [36]

Pokud se jedná o soubor dat, který není závislý na pořadí, je možné vstupní signál rozdělit do tříd a ty řešit jednotlivě pomocí samostatných konvolučních sítí. V [36] je řešená soustava rozdělena do 5 konvolučních sítí (CNN_1, CNN_2, CNN_3, CNN_4, CNN_5), každá z nich obsahuje architekturu jednoduché konvoluční sítě z Obr. 15. Vstup pro CNN_1 je stejný jako pro jednoduchou konvoluční síť. Vstup pro CNN_2 bere ze vstupní vrstvy hodnoty s přírůstkem dva, začínající první hodnotou. Podobně vstup pro CNN_3 bere ze vstupní vrstvy hodnoty s přírůstkem dva začínající druhou hodnotou. Vstup CNN_4 a CNN_5 jsou navrženy podobně, akorát s přírůstkem tři, začínajícím první a druhou hodnotou vstupní vrstvy, znázorněno v Obr. 16. Toto rozdělení do jednotlivých CNN urychluje výpočet a zapříčiňuje přesnější výsledky výstupů.



Obr. 16. Architektura multi-konvoluční neuronové sítě. [36]



Obr. 17. Vstupy do multi CNN. (a) vstupní hodnoty. (a), (b), (c), (d), a (e) znázorňují vstupy do CNN_1, CNN_2, CNN_3, CNN_4, CNN_5. Modrá barva značí použité hodnoty vstupu, oranžová barva značí vynechané hodnoty vstupu. [36]

5.1 Dopředná a zpětná propagace v CNN vrstvách

V každé CNN vrstvě je popsána 1D dopředná propagace jako:

$$x_k^l = b_k^l \sum_{i=1}^{N_{l-1}} \mathrm{con}v1\mathrm{D}(w_{ik}^{l-1}, s_i^{l-1}), \tag{60}$$

kde x_k^l je vstup, b_k^l je zkreslení k-tého neuronu vrstvy l, s_i^{l-1} je výstup i-tého neuronu vrstvy $l-1$, w_{ik}^{l-1} je kernel i-tého neuronu vrstvy $l-1$ k-tého neuronu vrstvy l. $\mathrm{con}v1\mathrm{D}(.,.)$ je pro provedení 1D konvoluce bez nulové výplně (zero-padding). Tudíž je rozměr vstupního pole x_k^l menší, než rozměr výstupních polí s_i^{l-1}. Mezivýstup y_k^l lze znázornit použitím aktivační funkce $f(.)$ na vstup x_k^l:

$$y_k^l = f(x_k^l) \text{ a } s_k^l = y_k^l \downarrow ss, \tag{61}$$

kde s_k^l je výstup k-tého neuronu vrstvy l a "$\downarrow ss$" je operace dekomprese vzorkování skalárním faktorem ss.

Algoritmus zpětné propagace začíná chybou výstupu vícevrstvého perceptronu. Uvažujme $l = 1$ pro vstupní vrstvu a $l = L$ pro výstupní vrstvu. N_L je počet tříd v databázi. Vstupní vektor p a jeho cílový vektor $\boldsymbol{t}^p$ a výstupní vektory $[y_1^L, \dots, y_{N_L}^L]$. Ve výstupní vrstvě pro vstup p je střední kvadratická chyba E_p popsána vzorcem:

$$E_p = \mathrm{MSE}\left(\boldsymbol{t}^p, [y_1^L, \dots, y_{N_L}^L]'\right) = \sum_{i=1}^{N_L} \left(y_i^L - t_i^p\right)^2 \tag{62}$$

Pro nalezení derivace E_p každým parametrem sítě, je zapotřebí vypočítat chybu delta $\Delta_k^l = \frac{\partial E}{\partial x_k^l}$. Toto je třeba pro aktualizaci rozptylu neuronu a všech vah předešlých vrstev. Je použito řetězové pravidlo derivací:

$$\frac{\partial E}{\partial w_{ik}^{l-1}} = \Delta_k^l y_i^{l-1} \text{ a } \frac{\partial E}{\partial b_k^l} = \Delta_k^l \tag{63}$$

Z prvního vícevrstvého perceptronu do poslední CNN vrstvy je regularizovaná (skalární) zpětná propagace popsána rovnicí:

$$\frac{\partial E}{\partial s_k^l} = \Delta s_k^l = \sum_{i=1}^{N_{l+1}} \frac{\partial E}{\partial x_i^{l+1}} \frac{\partial x_i^{l+1}}{\partial s_k^l} = \sum_{i=1}^{N_{l+1}} \Delta_i^{l+1} w_{ki}^l \tag{64}$$

Jakmile je provedena zpětná propagace z vrstvy $l+1$ do aktuální vrstvy l, je možné pokračovat k vstupu delta Δ_k^l vrstvy l. $\mathrm{us}_k^l = \mathrm{up}(s_k^l)$ je zvýšená vzorkovací mapa nulového řádu, poté je chyba delta znázorněna jako:

$$\Delta_k^l = \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial \mathrm{us}_k^l} \frac{\partial \mathrm{us}_k^l}{\partial y_k^l} f'(x_k^l) = \mathrm{up}(s_k^l) \beta f'(x_k^l), \tag{65}$$

kde $\beta = (ss)^{-1}$. Zpětná propagace chyby delta $\left(\Delta s_k^l \overset{\Sigma}{\leftarrow} \Delta_i^{l+1}\right)$ je popsána rovnicí:

$$\Delta s_k^l = \sum_{i=1}^{N_{l+1}} \mathrm{conv1Dz}(\Delta_i^{l+1}, rev(w_{ki}^l)), \tag{66}$$

kde $rev(.)$ je použito k otočení pole a $\mathrm{conv1Dz}(.,.)$ je použito k vykonání plné 1D konvoluce s nulovou výplní. Citlivost vah a zkreslení je:

$$\frac{\partial E}{\partial w_{ik}^l} = \mathrm{conv1D}(s_k^l, \Delta_i^{l+1}) \text{ a } \frac{\partial E}{\partial b_k^l} = \sum_n \Delta_k^l(n) \tag{67}$$

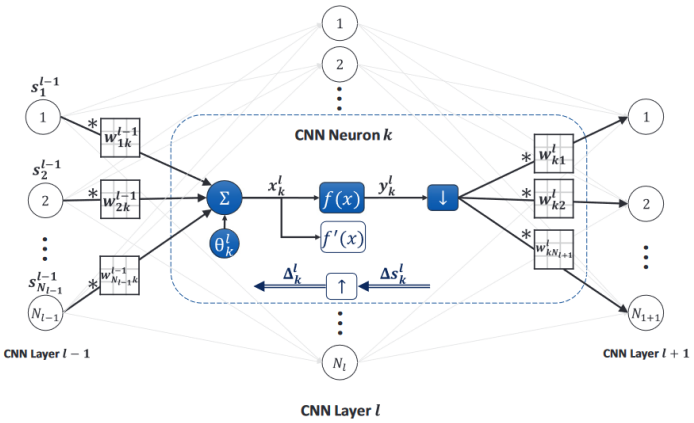
Jakmile je vypočítaná citlivost vah a zkreslení, může být použita k aktualizaci vah a zkreslení pomocí faktoru učení ε:

$$w_{ik}^{l-1}(t+1) = w_{ik}^{l-1}(t) - \varepsilon \frac{\partial E}{\partial w_{ik}^{l-1}} \text{ a } b_k^l(t+1) = b_k^l(t) - \varepsilon \frac{\partial E}{\partial b_k^l} \tag{68}$$

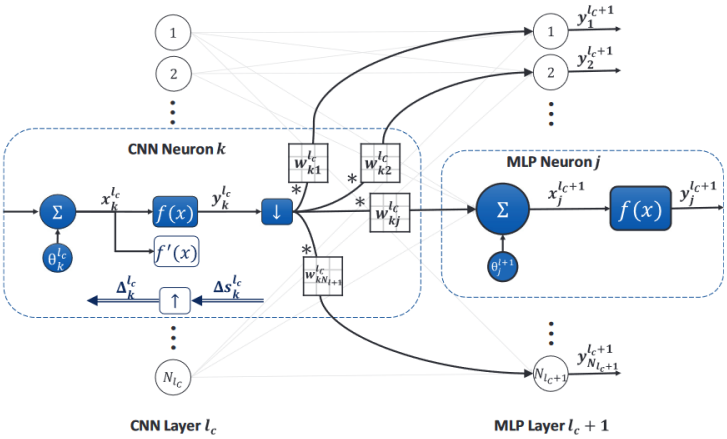
Dopředná a zpětná propagace ve skryté CNN vrstvě je zobrazena v Obr. 18. Výstupní citlivost *k*-tého neuronu ve vrstvě l, $\mathbf{\Delta s_k^l}$, je vytvořeno zpětnou propagací všech chyb delta Δ_i^{l+1} nacházejících se ve vrstvě $l+1$ pomocí rovnice (66). Dopředná a zpětná propagace mezi poslední skrytou vrstvou CNN a první skrytou vrstvou vícevrstvého perceptronu je pospána v Obr. 19.

Iterativní tok pro zpětnou propagaci 1D syrového signálu v trénovací fázi může být znázorněn jako:

1) Inicializace vah a zkreslení sítě (náhodně v rozpětí $(-0{,}1; 0{,}1)$).
2) Pro každou iteraci zpětné propagace:
    a. Pro každý vstupní vzorek souboru dat:
        i. Dopředná propagace z vstupní vrstvy do výstupní vrstvy pro získání výstupů každého neuronu každé vrstvy s_i^l, $\forall i \in [1, N_l]$ a $\forall l \in [1, L]$.
        ii. Výpočet chyby delta ve výstupní vrstvě a zpětně propagovat do první skryté vrstvy pro výpočet chyb delta Δ_k^l, $\forall k \in [1, N_l]$ a $\forall l \in [1, L]$.
        iii. Následné zpracování výpočtu citlivosti vah a zkreslení pomocí rovnice (67).
        iv. Aktualizace vah a zkreslení pomocí citlivosti a faktoru učení ε rovnicí (68). [34]



Obr. 18. Dopředná a zpětná propagace ve skrytých CNN vrstvách. [34]



Obr. 19. Dopředná a zpětná propagace mezi poslední skrytou vrstvou CNN a první skrytou vrstvou vícevrstvého perceptronu. [34]

5.2 Konvoluční Autoenkodér (CAE)

CAE kombinuje výhody konvolučních sítí a autoenkodéru k dosáhnutí silné reprezentace vlastností sledované soustavy. CAE umožňuje sdílení vah mezi vstupy a zajišťuje zachování jejich vlastností. Díky tomuto se počet parametrů redukuje, což snižuje výpočetní náročnost a urychluje výpočet. Tímto je CAE s využitím konvolučních vrstev upřednostňováno před klasickým autoenkodérem s plně propojenými vrstvami. Pro enkodování je použita konvoluční vrstva a pro dekodovací proces je použita dekonvoluční vrstva.

Pro 1D CAE lze použít sérii konvolučních vrstev, které zefektivní a zpřesní průběh trénování. Skryté vrstvy jsou znázorněny jako:

$$H^l = f(H^{l-1} * W^l + b^l), \tag{69}$$

kde $l \in (1, 2, \dots, L)$, pokud je použito L konvolučních vrstev a H^0 označuje originální vstupní vektor X.

Podobně je nastavena série dekonvolučních vrstev v dekodovacím procesu za účelem rekonstrukce originálního vstupu sítě z H^L. Toto je znázorněno rovnicí:

$$Z^l = g(Z^{l-1} * W^{(L-l+1)T} + b'^l), \tag{70}$$

kde Z^l je vstup první dekodovací vrstvy, což je H^L, výstup poslední enkodovací vrstvy. Jako výstup poslední dekodovací vrstvy Z^L je rekonstrukce originálního vstupního vektoru X.

1D CAE je navržena k měření střední kvadratické chyby k výpočtu rekonstrukční chyby mezi originálním vstupním vektorem X a rekonstruovaným vektorem Z. [7]

V [37] je navržen 1D CAE algoritmus, který ošetřuje vstup a výstup algoritmu, aby se pokaždé jednalo o sudý počet vstupních a výstupních signálů. V enkoderu je použita nulová výplň a v dekoderu ořezávací funkce. Enkoder se skládá z kombinace konvolučních a sdružovacích vrstev. Dekoder tyto vrstvy musí kopírovat.

	Layer type	Activation function	kernel size	features filter
Encoder	-	-	-	$b, 1$
	1D Zero padding	-	-	$\tilde{b} = \begin{cases} b, & \text{if } b \text{ even} \\ b+1, & \text{if } b \text{ odd} \end{cases}, 1$
	1D Convolutional	ϕ_1= Leaky ReLU	11	$\tilde{b}, 64$
	1D MaxPooling	-	2	$\lfloor \tilde{b}/2 \rfloor, 64$
	1D Convolutional	ϕ_1= Leaky ReLU	11	$\lfloor \tilde{b}/2 \rfloor, 32$
	1D MaxPooling	-	2	$\hat{b}, 32$
	1D Convolutional	ϕ_1= Leaky ReLU	9	$\hat{b}, 16$
	1D Convolutional	ϕ_1= Leaky ReLU	7	$\hat{b}, 1$
Decoder	1D Convolutional	ϕ_1= Leaky ReLU	7	$\hat{b}, 16$
	1D Convolutional	ϕ_1= Leaky ReLU	9	$\hat{b}, 32$
	1D UpSampling	-	2	$\hat{b} \cdot 2, 32$
	1D Convolutional	ϕ_1= Leaky ReLU	11	$\hat{b} \cdot 2, 64$
	1D UpSampling	-	2	$\hat{b} \cdot 4, 64$
	1D Convolutional	ϕ_2= sigmoid	11	$\hat{b} \cdot 4, 1$
	Cropping1D	-	-	$b = \begin{cases} \hat{b}-1, & \text{if } b \text{ even} \\ \hat{b}, & \text{if } b \text{ odd} \end{cases}, 1$

Obr. 20. Struktura 1D konvolučního autoenkoder modelu [37]

6. Realizace

6.1 Výpočetní Hardware

CPU: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, 2304 Mhz, jádra: 4, logické procesory: 8

RAM: 8 GB

GPU: NVIDIA GeForce GTX 1050 s výpočetní schopností 6,1 potřebnou pro CUDA toolkit

6.2 Knihovny Python

Pandas byla použita pro načtení csv souboru, kde se nacházely všechny potřebná data. [38]

Numpy byla použita na práci a úpravu vektorů a matic. [39]

Matplotlib byla použita pro tvorbu grafů. [40]

Tensorflow byla použita pro rozhrání neuronové sítě a jejího modelu, konkrétně trénovací části a predikce. [41]

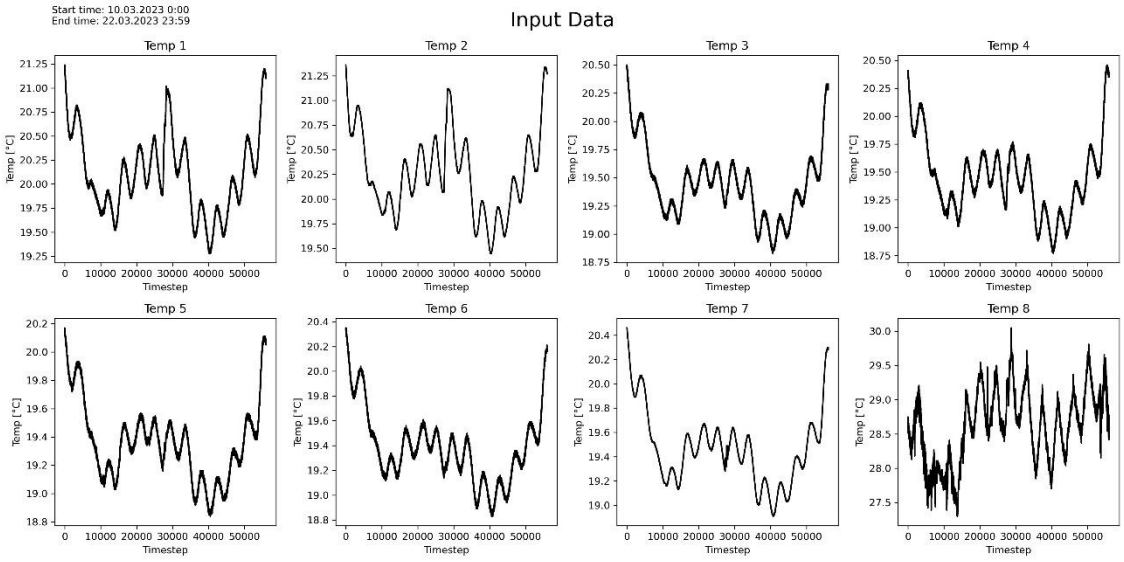
Keras byla použita pro: aktivační funkce, optimalizační algoritmy, konkrétní vrstvy neuronové sítě, callback brzkého přerušení, a regulátory [42]

Sklearn byla použita pro preprocessing dat (normalizaci) a pro rozdělení datasetu na testovací a validační sety. [43]

Json byla použita pro uložení historie modelů pro pozdější načtení a zpracování. [44]

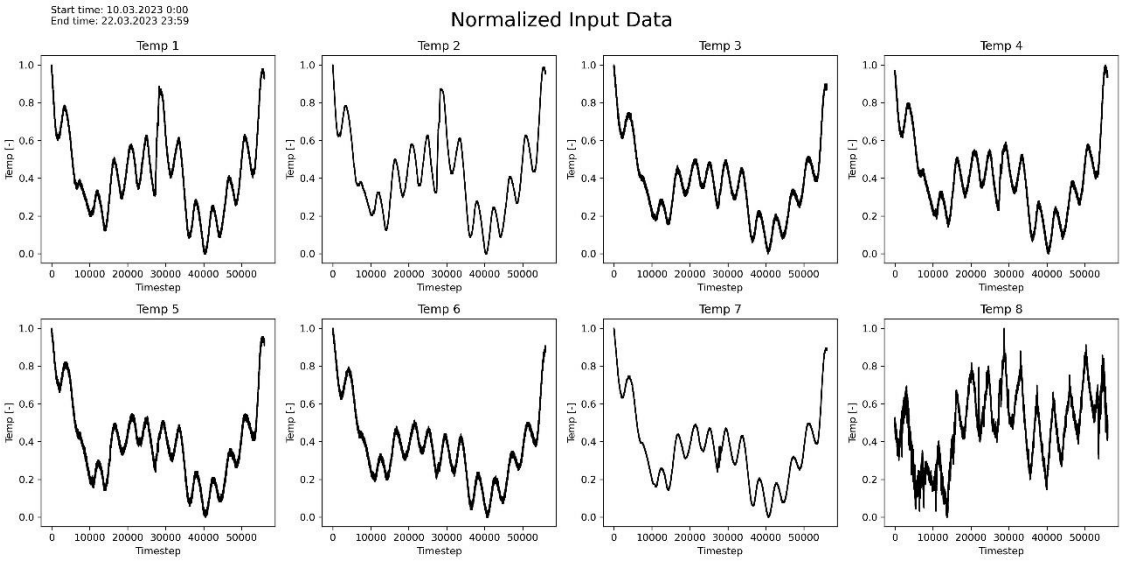
6.3 Charakterizace dat a normalizace

Byl obdržen dataset s osmi 1D signály znázorňující teploty. Jedná se o dataset vypovídající délku 56013 časových kroků.



Obr. 21. Vstupní data obsahující 8 1D signálů znázorňující jednotlivé průběhy teplot.

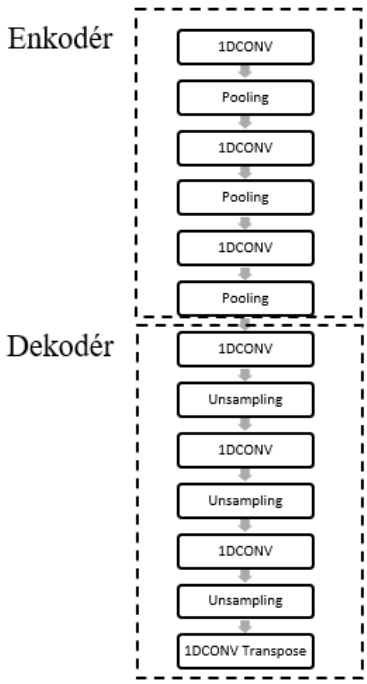
Pro modelový výpočet bylo potřeba data normalizovat. Data byly normalizovány algoritmem MinMax scaling.



Obr. 22. Normalizovaná vstupní data obsahující 8 1D signálů znázorňující jednotlivé průběhy teplot.

6.4 Zvolení výpočetního algoritmu

Sledovaná soustava má pouze jeden vstup, tudíž se jedná o učení bez dozoru (Unsupervised learning). Pro případ spojitých dat byl zvolen algoritmus autoenkodéru, konkrétně **konvoluční autoenkodér**. Z *kapitoly 5.2* byl inspirován model autoenkodéru, použit pro výpočet soustavy.



Obr. 23. Schéma konvolučního autoenkodéru.

6.5 Popis algoritmu

V modelu jsou nastavitelné parametry – počet filtrů, velikost kernelů a aktivační funkce v konvolučních vrstvách a ve sdružovacích vrstvách velikost sdružovacího filtru. Toto platí pro enkodér i dekodér, avšak v dekodéru je namísto sdružovací vrstvy, vrstva fungující opačně – unspampling vrstva.

Volba filtrů byla navrhnuta tak, aby se zrcadlila v enkodéru a dekodéru. Poslední vrstva dekodéru se nazývá Conv1DTranspose, což uvádí rozměr výstupu na původní vstupní dimenzi.

6.6 Volba nejvhodnějšího nastavení modelu

Na základě rešeršní práce byly zvoleny následující parametry pro počáteční nastavení modelu:

- Optimalizační algoritmus: Adamax
- Ztrátová funkce: MSE
- Aktivační funkce konvolučních vrstev (Aktivační funkce 1): ReLU
- Aktivační funkce poslední konvoluční vrstvy (transpoziční – Aktivační funkce 2): Sigmoid
- Rozvržení filtrů: 2,4,8 (zrcadlově v dekodéru), transpoziční konvoluční vrstva: 8
- Velikost kernelů: 3,3,3 (zrcadlově v dekodéru), transpoziční konvoluční vrstva: 3
- Velikost filtrů ve sdružovacích a unsampling vrstvách zůstal na hodnotě 2.

Dále byl navrhnut postup získání ideální kombinace parametrů, funkcí a algoritmu:

1) Volba filtrů a jejich rozvržení
2) Volba velikosti kernelů
3) Volba optimalizačního algoritmu
4) Volba ztrátové funkce
5) Volba aktivačních funkcí
6) Dodatečná regularizace

6.7 Trénování algoritmu

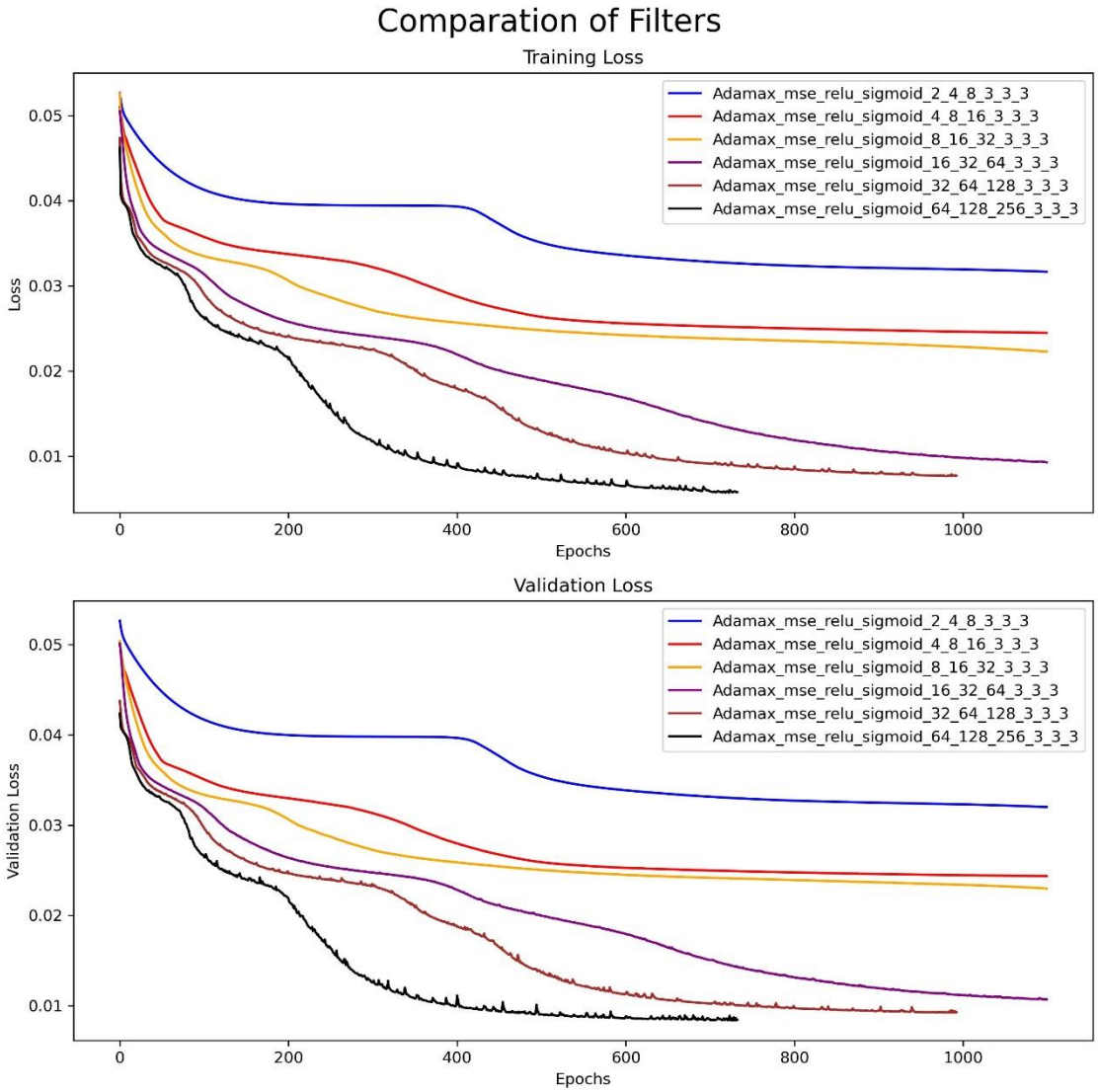
Vstupní dataset byl rozdělen na trénovací a validační sety dat v poměru 8:2. Trénování bylo nastaveno na 1100 epoch s přidaným callbackem o předčasné ukončení výpočtu, pokud se ztrátová hodnota nebude zlepšovat do 10 kroků.

6.7.1 Volba filtrů

Bylo zvoleno 6 kombinací rozložení filtrů a ty byly následně natrénovány a mezi sebou porovnány, jak z hlediska hodnot ztrátových a validačních ztrátových hodnot, tak i ze schopnosti předpovídat testovací signál.

Tabulka 1. Kombinace rozvržení filtrů. Optimalizační algoritmus: Adamax, Ztrátová funkce: MSE, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení kernelů: 3,3,3.

Vzorek	Filtry	Počet parametrů	Min. ztrátová hodnota	Min. val. ztrátová hodnota
1	2,4,8	564	0,032	0,032
2	4,8,16	1984	0,024	0,024
3	8,16,32	7416	0,022	0,023
4	16,32,64	28648	0,009	0,011
5	32,64,128	112584	0,008	0,009
6	64,128,256	446344	0,006	0,008



Obr. 24. Porovnání ztrátových a validačních ztrátových průběhů pro různé zvolení rozvržení filtrů.

Na Obr. 24 lze vidět porovnání ztrátových a validačních ztrátových průběhů. První tři vzorky disponují velkou hodnotou ztráty, což následně zapříčiňuje nízkou schopnost přizpůsobení se sledované soustavě a její možnost predikovat.

Ze zbylých tří vzorků byly porovnány jejich predikce vůči testovacím datům a do budoucího testování postupuje model s rozvržením filtrů: 32,64,128.

Více výsledků v Příloze 1.

6.7.2 Volba velikosti kernelů

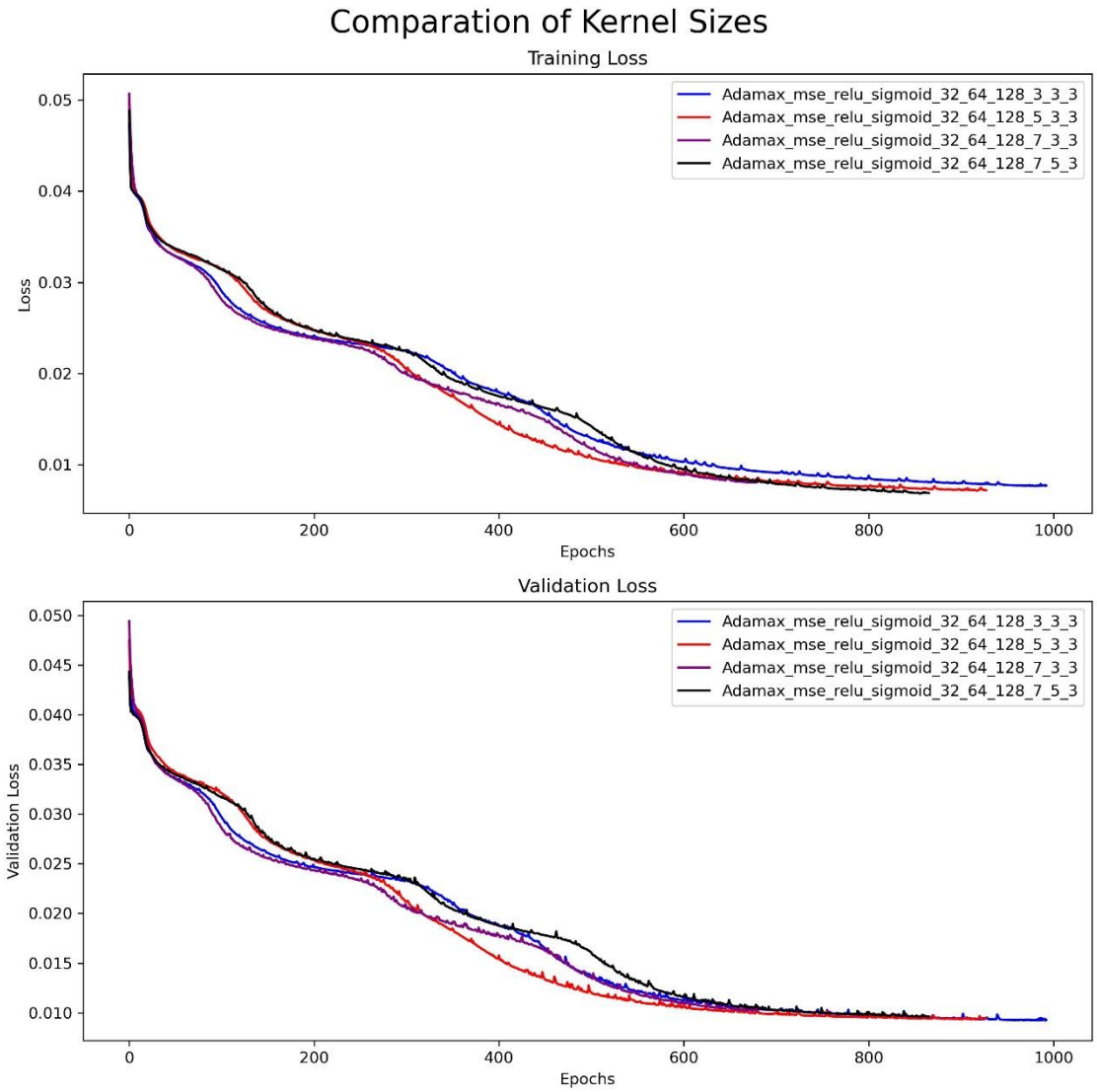
Pro volbu kernelů byly zvoleny čtyři modely, které byly následně vypočítány a porovnány.

Tabulka 2. Kombinace rozvržení velikostí kernelů. Optimalizační algoritmus: Adamax, Ztrátová funkce: MSE, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení filtrů: 32,64,128.

Vzorek	Velikost kernelů	Počet parametrů	Min. ztrátová hodnota	Min. val. ztrátová hodnota
1	3,3,3	112584	0,008	0,009
2	5,3,3	117192	0,007	0,009
3	7,3,3	121800	0,008	0,010
4	7,5,3	142280	0,007	0,010

Sledovaná soustava je členitého charakteru, což lze vidět na vstupních datech na Obr. 21. Lze tedy předpokládat, že model bude přizpůsobivější s menšími velikostmi kernelů, což se také potvrdilo při porovnání ztrát na Obr. 25 a následném porovnání predikcí, kde nejlepší výsledky (nejnižší hodnotu MSE mezi reálnými daty a predikcí), mělo rozvržení kernelů: 3,3,3.

Více výsledků v Příloze 2.



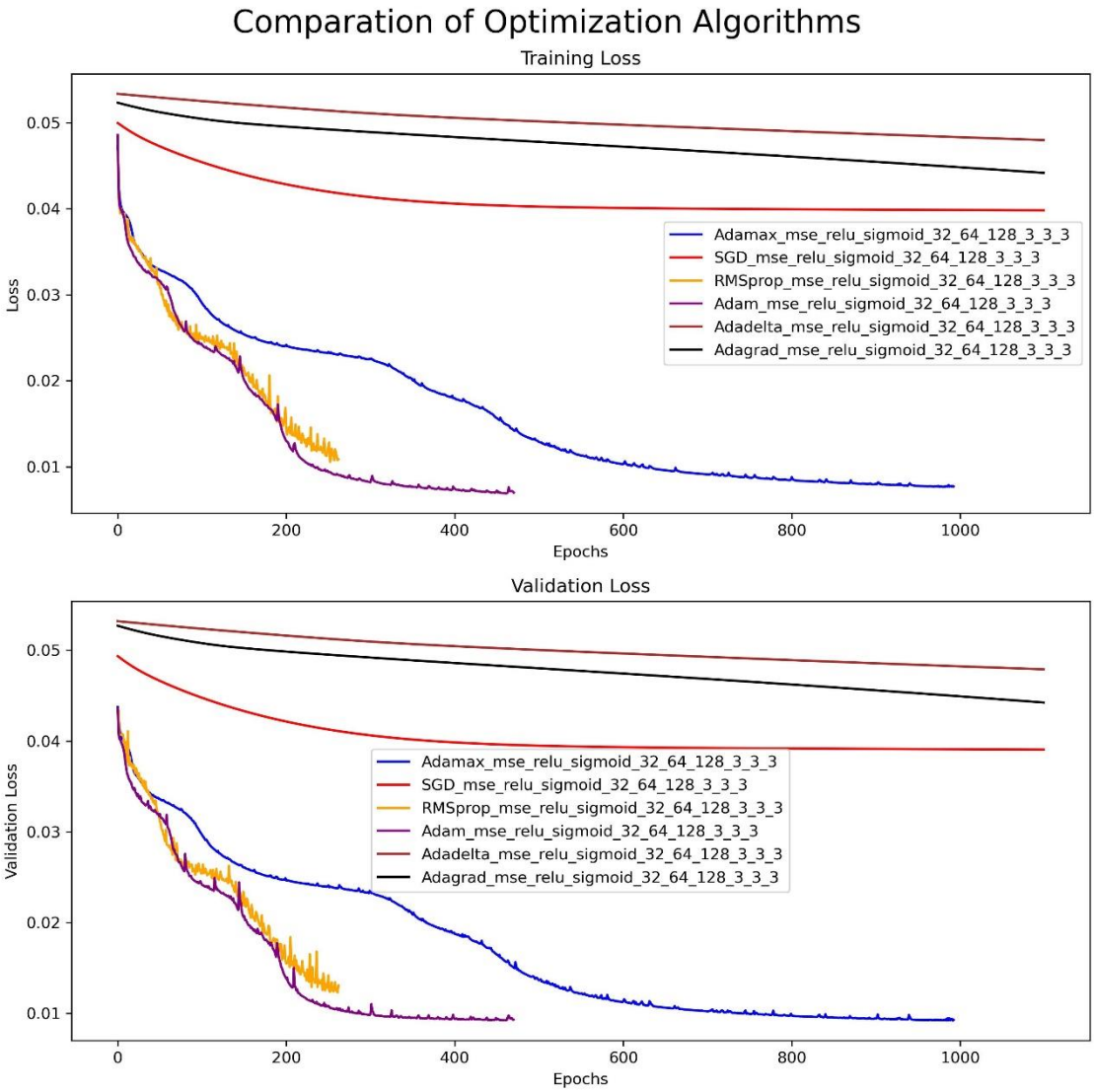
Obr. 25. Porovnání ztrátových a validačních ztrátových průběhů pro různé zvolení velikostí kernelů.

6.7.3 Volba optimalizačního algoritmu

Bylo porovnáno 6 různých optimalizačních algoritmů, znázorněných v Tabulce 3.

Tabulka 3. Kombinace optimalizačních algoritmů. Ztrátová funkce: MSE, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení kernelů: 3,3,3. Rozvržení filtrů: 32,64,128.

Vzorek	Optimalizační algoritmus	Počet parametrů	Min. ztrátová hodnota	Min. val. ztrátová hodnota
1	Adamax	112584	0,008	0,009
2	SGD	112584	0,040	0,039
3	RMSprop	112584	0,011	0,012
4	Adam	112584	0,007	0,009
5	Adadelta	112584	0,048	0,048
6	Adagrad	112584	0,044	0,044



Obr. 26. Porovnání ztrátových a validačních ztrátových průběhů pro různé optimalizační algoritmy.

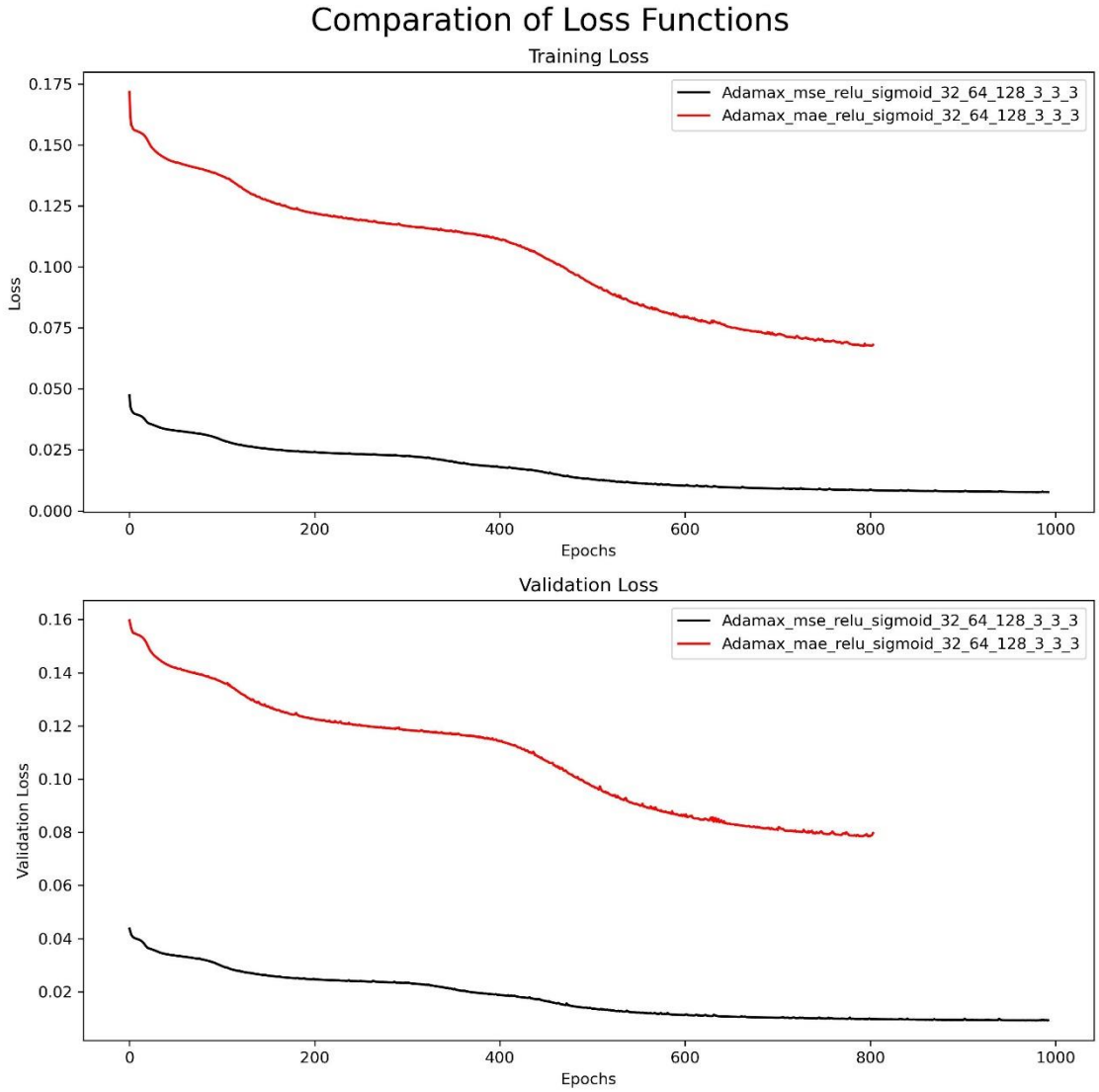
Podle hodnot ztrát z Tabulky 3 a průběhů z Obr. 26 je možné vidět, že vhodnými algoritmy jsou pouze Adam a Adamax. Avšak po porovnání predikcí těchto dvou optimalizačních algoritmů vychází opět lépe algoritmus Adamax, což také potvrzuje hodnota MSE, která je pro algoritmus Adamax menší.

Více výsledků v Příloze 3.

6.7.4 Volba ztrátové funkce

Tabulka 4. Kombinace ztrátových funkcí. Optimalizační algoritmus: Adamax, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení kernelů: 3,3,3, Rozvržení filtrů: 32,64,128.

Vzorek	Ztrátová funkce	Počet parametrů	Min. ztrátová hodnota	Min. val. ztrátová hodnota
1	MSE	112584	0,008	0,009
2	MAE	112584	0,068	0,078



Obr. 27. Porovnání ztrátových a validačních ztrátových průběhů pro různé ztrátové funkce.

Z Tabulky 4 a Obr. 27 je jasné, že ztrátová funkce s lepšími výsledky je MSE.

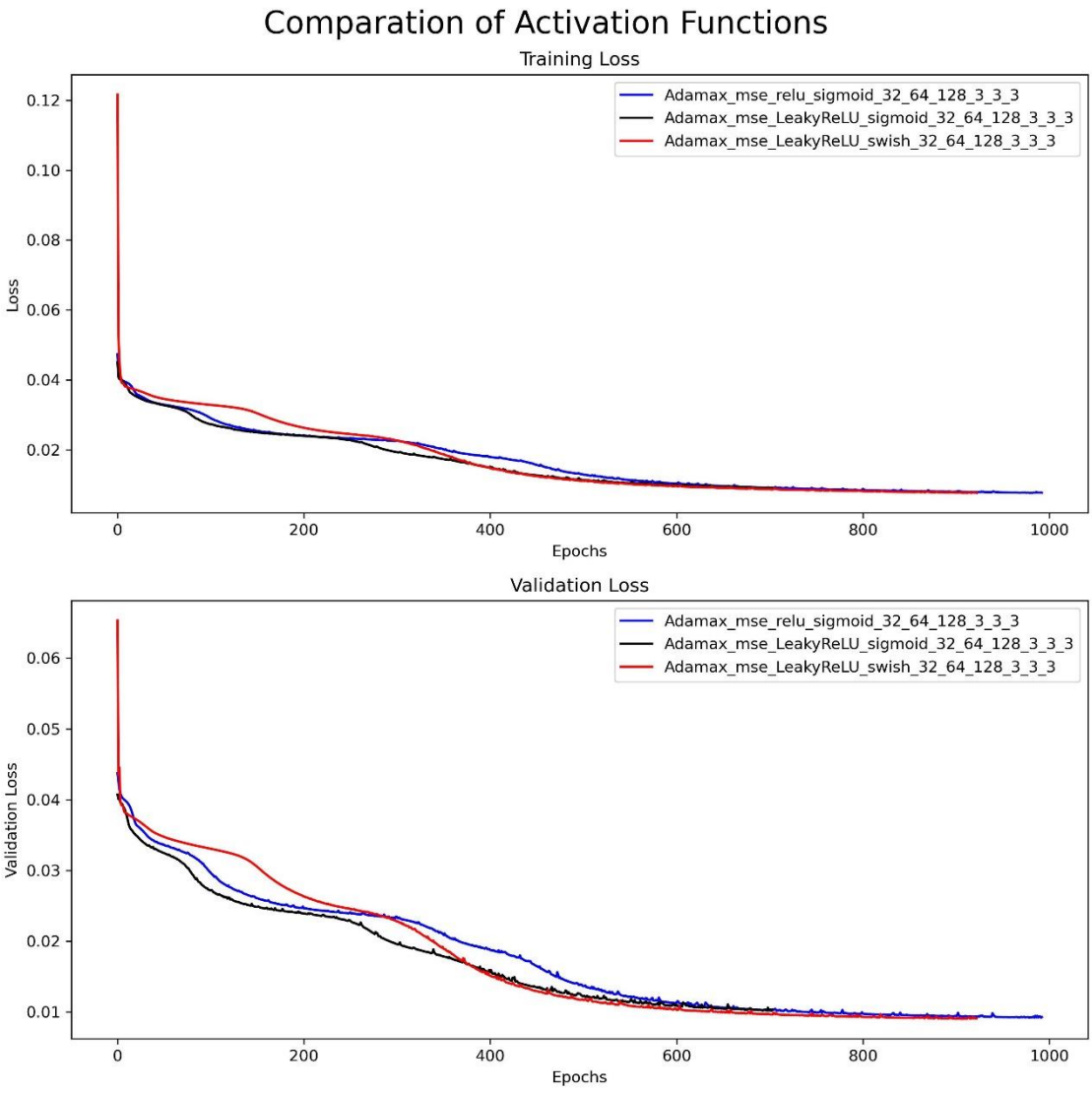
Více výsledků v Příloze 4.

6.7.5 Volba aktivačních funkcí

Vzhledem k obstojným výsledkům modelů s počátečně zvolenými aktivačními funkcemi, bylo uváženo aktivační funkce ponechat a otestovat jejich nadstavby, jmenovitě Leaky ReLU a Swish.

Tabulka 5. Kombinace aktivačních funkcí. Optimalizační algoritmus: Adamax, Rozvržení kernelů: 3,3,3, Rozvržení filtrů: 32,64,128, počet parametrů: 112584, Ztrátová funkce: MSE.

Vzorek	Aktivační funkce 1	Aktivační funkce 2	Min. ztrátová hodnota	Min. val. ztrátová hodnota
1	ReLU	Sigmoid	0,008	0,009
2	LeakyReLU	Sigmoid	0,009	0,010
3	LeakyReLU	Swish	0,007	0,009



Obr. 28. Porovnání ztrátových a validačních ztrátových průběhů pro kombinace aktivačních funkcí.

Podle grafů v Obr. 28 nebo Tabulky 5 je viditelná podobnost modelů, co se týče ztrátových hodnot. Avšak hlavní rozdíly lze pozorovat v predikcích, kde nejlepších výsledků dosáhla kombinace aktivační funkce 1: Leaky ReLU a aktivační funkce 2: Swish. Tato kombinace dosáhla na hodnotu $\mathrm{MSE} = 0{,}011$ znázorňující rozdíl mezi reálnými hodnotami a predikcí.

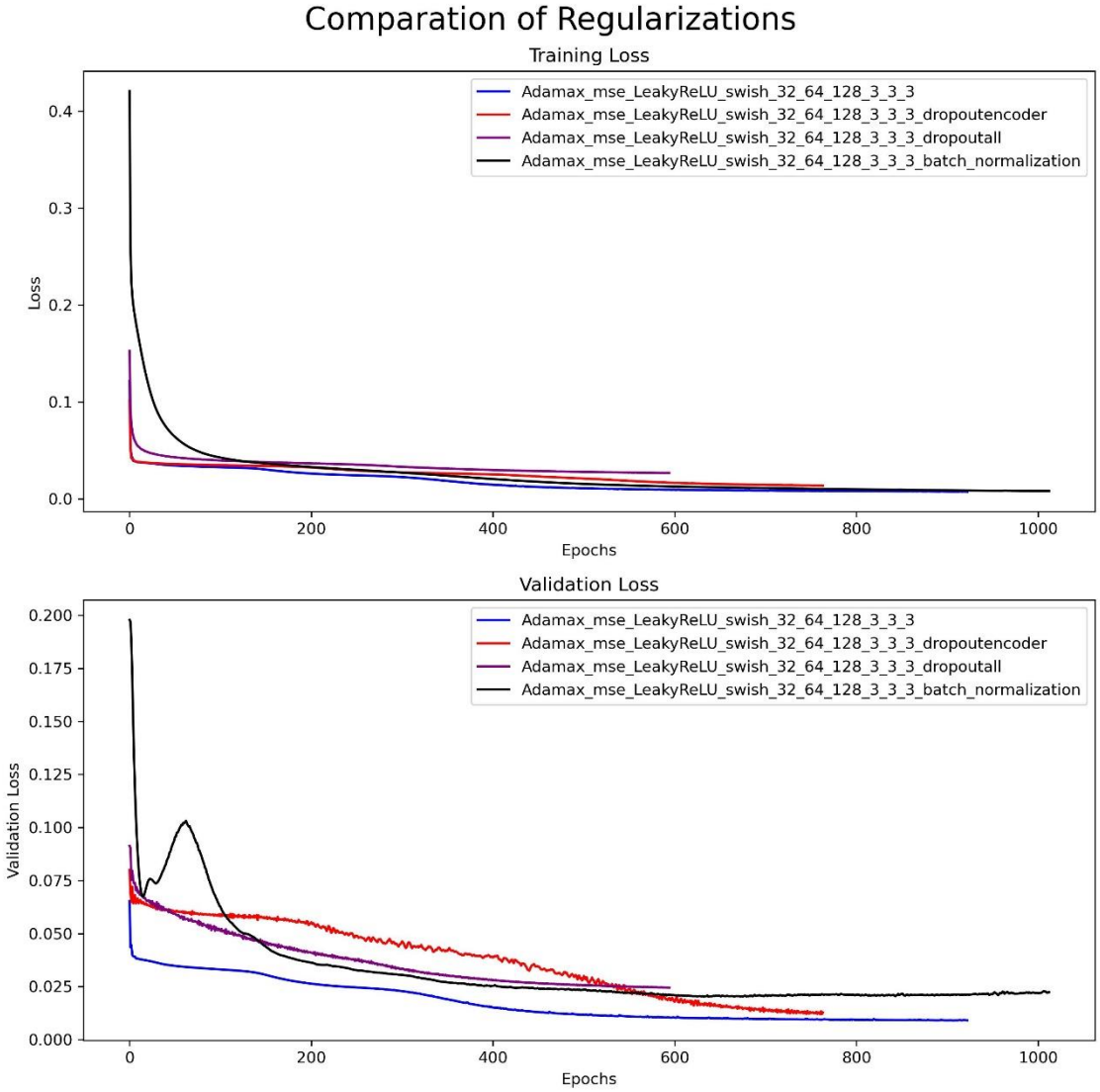
Více výsledků v Příloze 5.

6.7.6 Dodatečná regularizace

Pro dodatečnou regularizaci byly navrženy algoritmy Dropout a Batch Normalization, kdy byl Dropout testován na použití pouze v enkodéru a poté v celé síti.

Tabulka 6. Volba dodatečných regularizací. Optimalizační algoritmus: Adamax, Aktivační funkce 1: Leaky ReLU, Aktivační funkce 2: Swish, Rozvržení kernelů: 3,3,3, Rozvržení filtrů: 32,64,128, Ztrátová funkce: MSE.

Vzorek	Regularizace	Počet parametrů	Min. ztrátová hodnota	Min. val. ztrátová hodnota
1	-	112584	0,007	0,009
2	DropoutEncoder	112584	0,014	0,012
3	DropoutAll	112584	0,027	0,024
4	Batch Normalization	114376	0,008	0,020



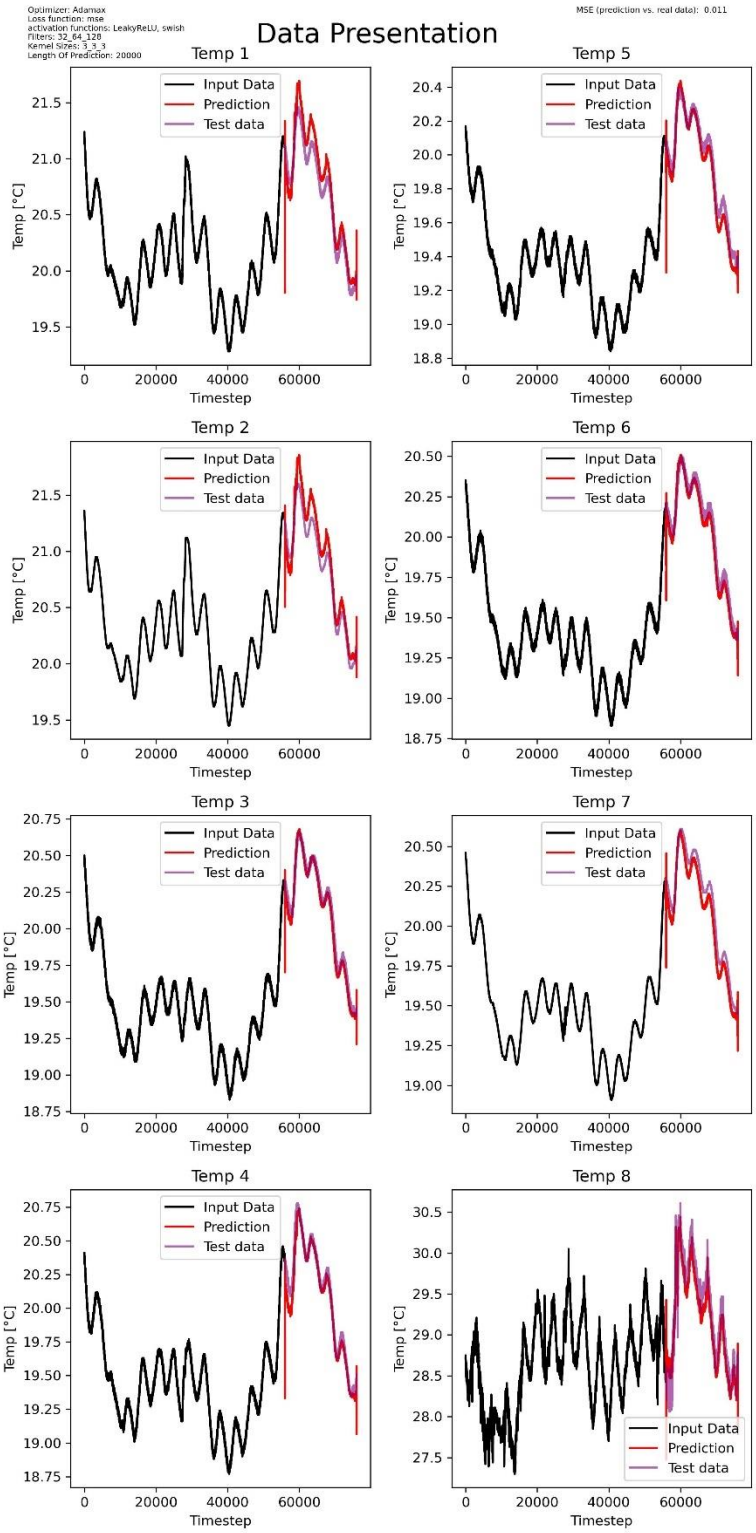
Obr. 29. Porovnání ztrátových a validačních ztrátových průběhů pro dodatečné regularizace.

Porovnáním hodnot ztrát a predikcí generovaných těmito modely, číselně vyšel nejlépe model bez dodatečných regularizací. Toto lze odůvodnit správným natrénováním původního modelu, kdy nedochází k tzv. overfittingu.

Více výsledků v Příloze 6.

6.8 Predikce

Predikce byla prováděna na délku 20000 kroků. Vstup byly data, na kterých model nebyl trénován.



Obr. 30. Predikce. Optimalizační algoritmus: Adamax, Ztrátová funkce: MSE, Aktivační funkce 1: LeakyReLU, Aktivační funkce 2: Swish, Rozložení kernelů: 3,3,3, Rozložení filtrů: 32,64,128.

7. Závěr

Základní část je tvořena rešeršním výzkumem, kdy bylo zapotřebí pochopit do hloubky danou problematiku. Byla zjištěna základní odvětví a ta byla dopodrobna rozvedena.

Hlavním cílem této práce bylo vytvoření modelu na základě rešeršní části. Model musel být dostatečně přesný, aby dokázal předpovídat dlouhodobě nelineární systémy.

Jako vstup byly obdrženy průběhy 8 teplot, které byly zaznamenány v průběhu 12 dní. Následně model predikoval vývoj teplot po délku zhruba 4 dní. Parametry modelu byly prvotně zvoleny na základě rešeršní práce, ale později, navrženým postupem úprav parametrů, změněny na parametry vyhovující chování systému s co nejmenší hodnotou střední kvadratické chyby (MSE) predikce, konkrétně $\mathrm{MSE} = 0{,}011$.

Výpočty byly realizovány na sestavě zmíněné v *kapitole 6.1*. Trénování probíhalo s rozdělenými daty do několika podskupin, což mohlo snížit výpočetní přesnost. Tomuto by bylo možné předejít zvolením výkonnější výpočetní techniky. Lepšího výsledku by bylo možné dosáhnout i zvětšením testovacích dat, které by však opět zvýšilo výpočetní náročnost.

V této práci byl navrhnut jeden postup zvolení potřebných parametrů pro funkčnost modelu. Komplexností a počtem parametrů v modelu by bylo velmi náročné prozkoumat všechny možné kombinace jednotlivých parametrů, které by však mohly vést k lepším výsledkům. Tímto se otevírá otázka nových cest voleb parametrů, nebo dokonce jejich automatické úpravy v průběhu trénovací fáze.

1D konvoluční neuronové sítě disponují malou výpočetní náročností na poli neuronových sítí všeho druhu. Proto bylo možné model trénovat i na ne zcela optimální výpočetní soustavě. Navržený postup získání konkrétních parametrů vyústil v úspěšné namodelování a predikování nelineárního systému.

Bibliografie

[1] JUNG, Alexander. *Machine Learning*. Singapore: Springer, 2022. ISBN 9789811681929.

[2] BURKOV, Andriy. *Hundred-Page Machine Learning Book*. 2019. ISBN 978-1999579500.

[3] BALADRAM, Mohammad Samy, Atsushi KOIKE a Kazunori D YAMADA. Introduction to Supervised Machine Learning for Data Science. *Interdisciplinary information sciences* [online]. Sendai: The Editorial Committee of the Interdisciplinary Information Sciences, 2020, **26**(1), 87-121 [cit. 2023-03-05]. ISSN 1340-9050. Dostupné z: doi:10.4036/iis.2020.A.03

[4] BRAGA-NETO, Ulisses. *Fundamentals of Pattern Recognition and Machine Learning*. Cham: Springer International Publishing AG, 2020. ISBN 9783030276553.

[5] GHAHRAMANI, Zoubin. Unsupervised Learning. *Advanced Lectures on Machine Learning* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, 72-112 [cit. 2023-05-11]. ISBN 9783540231226. ISSN 0302-9743. Dostupné z: doi:10.1007/978-3-540-28650-9_5

[6] SIMEONE, Osvaldo. A Very Brief Introduction to Machine Learning With Applications to Communication Systems. *IEEE transactions on cognitive communications and networking* [online]. PISCATAWAY: IEEE, 2018, **4**(4), 648-664 [cit. 2023-03-06]. ISSN 2332-7731. Dostupné z: doi:10.1109/TCCN.2018.2881442

[7] BINBUSAYYIS, Adel a Thavavel VAIYAPURI. Unsupervised deep learning approach for network intrusion detection combining convolutional autoencoder and one-class SVM. *Applied intelligence (Dordrecht, Netherlands)* [online]. New York: Springer US, 2021, **51**(10), 7094-7108 [cit. 2023-05-12]. ISSN 0924-669X. Dostupné z: doi:10.1007/s10489-021-02205-9

[8] DING, Zihan, Yanhua HUANG, Hang YUAN a Hao DONG. Introduction to Reinforcement Learning. In: *Deep Reinforcement Learning*. Singapore: Springer Singapore, 2020, s. 47-123. ISBN 9811540942. Dostupné z: doi:10.1007/978-981-15-4095-0_2

[9] SEWAK, Mohit. *Deep Reinforcement Learning*. Singapore: Springer Singapore, 2019. ISBN 9811382840. Dostupné z: doi:10.1007/978-981-13-8285-7

[10] DONG, Hao, Zihan DING a Shanghang ZHANG. *Deep Reinforcement Learning*. Singapore: Springer Singapore Pte. Limited, 2020. ISBN 9811540942. Dostupné z: doi:10.1007/978-981-15-4095-0

[11] MACUKOW, Bohdan. Neural Networks – State of Art, Brief History, Basic Models and Architecture. In: *Computer Information Systems and Industrial Management*

[online]. Cham: Springer International Publishing, 2016, , s. 3-14 [cit. 2023-03-08]. ISBN 9783319453774. ISSN 0302-9743. Dostupné z: doi:10.1007/978-3-319-45378-1_1

[12] APAYDIN, Halit, Hajar FEIZI, Mohammad Taghi SATTARI, Muslume Sevba COLAK, Shahaboddin SHAMSHIRBAND a Kwok-wing CHAU. Comparative analysis of recurrent neural network architectures for reservoir inflow forecasting. *Water (Basel)* [online]. Basel: MDPI AG, 2020, **12**(5), 1500 [cit. 2023-03-13]. ISSN 2073-4441. Dostupné z: doi:10.3390/w12051500

[13] SALEM, Fathi M. *A Basic Recurrent Neural Network Model* [online]. 2016 [cit. 2023-03-14]. Dostupné z: doi:10.48550/arxiv.1612.09022

[14] REZK, Nesma M., Madhura PURNAPRAJNA, Tomas NORDSTROM a Zain UL-ABDIN. Recurrent Neural Networks: An Embedded Computing Perspective. *IEEE access* [online]. Ithaca: IEEE, 2020, **8**(1), 57967-57996 [cit. 2023-03-14]. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2020.2982416

[15] JIANG, Shengli, Shiyi QIN, Joshua L PULSIPHER a Victor M ZAVALA. *Convolutional Neural Networks: Basic Concepts and Applications in Manufacturing* [online]. 2022 [cit. 2023-03-08]. Dostupné z: doi:10.48550/arxiv.2210.07848

[16] O'SHEA, Keiron a Ryan NASH. *An Introduction to Convolutional Neural Networks* [online]. 2015, **2015** [cit. 2023-03-18]. Dostupné z: doi:10.48550/arxiv.1511.08458

[17] ZHU, Jianhao a Zong CHEN. Comparative analysis of various new activation functions based on convolutional neural network. *Journal of Physics: Conference Series* [online]. Bristol: IOP Publishing, 2020, **1676**(1), 12228 [cit. 2023-03-19]. ISSN 1742-6588. Dostupné z: doi:10.1088/1742-6596/1676/1/012228

[18] DING, Zihan, Yanhua HUANG, Hang YUAN a Hao DONG. Introduction to Deep Learning. In: *Deep Reinforcement Learning*. Singapore: Springer Singapore, 2020, s. 11-13. ISBN 9811540942. Dostupné z: doi:10.1007/978-981-15-4095-0_2

[19] KOURETAS, Ioannis a Vassilis PALIOURAS. Hardware Implementation of a Softmax-Like Function for Deep Learning. *Technologies (Basel)* [online]. MDPI AG, 2020, **8**(3), 46 [cit. 2023-03-19]. ISSN 2227-7080. Dostupné z: doi:10.3390/technologies8030046

[20] VARGAS, Victor Manuel, Pedro Antonio GUTIERREZ, Javier BARBERO-GOMEZ a Cesar HERVAS-MARTINEZ. Activation Functions for Convolutional Neural Networks: Proposals and Experimental Study. *IEEE transaction on neural networks and learning systems* [online]. IEEE, 2021, (34), 1-11 [cit. 2023-03-19]. ISSN 2162-237X. Dostupné z: doi:10.1109/TNNLS.2021.3105444

[21] MAGUOLO, Gianluca, Loris NANNI a Stefano GHIDONI. Ensemble of convolutional neural networks trained with different activation functions. *Expert*

systems with applications [online]. New York: Elsevier Ltd, 2021, **166**, 114048 [cit. 2023-03-19]. ISSN 0957-4174. Dostupné z: doi:10.1016/j.eswa.2020.114048

[22] RAMACHANDRAN, Prajit, Barret ZOPH a Quoc V LE. *Searching for Activation Functions* [online]. 2017 [cit. 2023-03-19]. Dostupné z: doi:10.48550/arxiv.1710.05941

[23] BOTTOU, Léon. Stochastic Gradient Descent Tricks. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 7700. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, , s. 421-436. ISBN 9783642352881. ISSN 0302-9743. Dostupné z: doi:10.1007/978-3-642-35289-8_25

[24] YANG, Jing a Guanci YANG. Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer. *Algorithms* [online]. Basel: MDPI AG, 2018, **11**(3), 1-15 [cit. 2023-03-22]. ISSN 1999-4893. Dostupné z: doi:10.3390/a11030028

[25] IDA, Yasutoshi, Yasuhiro FUJIWARA a Sotetsu IWAMURA. Adaptive learning rate via covariance matrix based preconditioning for deep neural networks. In: *IJCAI International Joint Conference on Artificial Intelligence* [online]. Ithaca: Cornell University Library, arXiv.org, 2016, s. 1923-1929 [cit. 2023-03-25]. ISBN 9780999241103. ISSN 1045-0823. Dostupné z: doi:10.24963/ijcai.2017/267

[26] WANG, Youming, Zhao XIAO a Gongqing CAO. A convolutional neural network method based on Adam optimizer with power-exponential learning rate for bearing fault diagnosis. *Journal of Vibroengineering* [online]. JVE International, 2022, **24**(4), 666-678 [cit. 2023-03-26]. ISSN 1392-8716. Dostupné z: doi:10.21595/jve.2022.22271

[27] KOHLI, Himani, Jyoti AGARWAL a Manoj KUMAR. An improved method for text detection using Adam optimization algorithm. *Global Transitions Proceedings* [online]. Elsevier B.V, 2022, **3**(1), 230-234 [cit. 2023-03-26]. ISSN 2666-285X. Dostupné z: doi:10.1016/j.gltp.2022.03.028

[28] RUDER, Sebastian. *An overview of gradient descent optimization algorithms* [online]. 2016 [cit. 2023-03-27]. Dostupné z: doi:10.48550/arxiv.1609.04747

[29] ZEILER, Matthew D. *ADADELTA: An Adaptive Learning Rate Method* [online]. 2012 [cit. 2023-03-27]. Dostupné z: doi:10.48550/arxiv.1212.5701

[30] KINGMA, Diederik P a Jimmy BA. *Adam: A Method for Stochastic Optimization* [online]. 2014 [cit. 2023-05-21]. Dostupné z: doi:10.48550/arxiv.1412.6980

[31] GARBIN, Christian, Xingquan ZHU a Oge MARQUES. Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia tools and applications* [online]. New York: Springer US, 2020, **79**(19-20), 12777-12815 [cit. 2023-05-12]. ISSN 1380-7501. Dostupné z: doi:10.1007/s11042-019-08453-9

[32] CHEN, Huan, Yue-hsien WANG a Chun-hung FAN. A convolutional autoencoder-based approach with batch normalization for energy disaggregation. *The Journal of supercomputing* [online]. New York: Springer US, 2021, **77**(3), 2961-2978 [cit. 2023-05-12]. ISSN 0920-8542. Dostupné z: doi:10.1007/s11227-020-03375-y

[33] GERVASI, Osvaldo, Beniamino MURGANTE, Sanjay MISRA, Ana Maria A. C ROCHA a Chiara GARAU. Improved CNN Based on Batch Normalization and Adam Optimizer. In: *Computational Science and Its Applications - ICCSA 2022 Workshops*. 13381. Switzerland: Springer International Publishing AG, 2022, , s. 593-604. ISBN 9783031105470. Dostupné z: doi:10.1007/978-3-031-10548-7_43

[34] KIRANYAZ, Serkan, Onur AVCI, Osama ABDELJABER, Turker INCE, Moncef GABBOUJ a Daniel J. INMAN. 1D convolutional neural networks and applications: A survey. *Mechanical systems and signal processing* [online]. Berlin: Elsevier Ltd, 2021, **151**, 107398 [cit. 2023-03-31]. ISSN 0888-3270. Dostupné z: doi:10.1016/j.ymssp.2020.107398

[35] KIRANYAZ, Serkan, Adel GASTLI, Lazhar BEN-BRAHIM, Nasser AL-EMADI a Moncef GABBOUJ. Real-Time Fault Detection and Identification for MMC Using 1-D Convolutional Neural Networks. *IEEE transactions on industrial electronics (1982)* [online]. PISCATAWAY: IEEE, 2019, **66**(11), 8760-8771 [cit. 2023-03-31]. ISSN 0278-0046. Dostupné z: doi:10.1109/TIE.2018.2833045

[36] HARBOLA, Shubhi a Volker COORS. One dimensional convolutional neural network architectures for wind prediction. *Energy conversion and management* [online]. OXFORD: Elsevier Ltd, 2019, **195**, 70-75 [cit. 2023-04-03]. ISSN 0196-8904. Dostupné z: doi:10.1016/j.enconman.2019.05.007

[37] KUESTER, J., W. GROSS a W. MIDDELMANN. 1D-Convolutional autoencoder based hyperspectral data compression. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* [online]. Gottingen: Copernicus GmbH, 2021, , s. 15-21 [cit. 2023-05-12]. ISSN 1682-1750. Dostupné z: doi:10.5194/isprs-archives-XLIII-B1-2021-15-2021

[38] MCKINNEY, Wes. Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference* [online]. 2010, (445), 56-61 [cit. 2023-05-22]. Dostupné z: doi:10.25080/Majora-92bf1922-00a

[39] HARRIS, Charles R, K Jarrod MILLMAN, Stéfan J VAN DER WALT et al. Array programming with NumPy. *Nature (London)* [online]. England: Nature Publishing Group, 2020, **585**(7825), 357-362 [cit. 2023-05-22]. ISSN 0028-0836. Dostupné z: doi:10.1038/s41586-020-2649-2

[40] HUNTER, John D. Matplotlib: A 2D Graphics Environment. *Computing in science & engineering* [online]. New York: IEEE, 2007, **9**(3), 90-95 [cit. 2023-05-22]. ISSN 1521-9615. Dostupné z: doi:10.1109/MCSE.2007.55

[41] ABADI, Martín, Paul BARHAM, Jianmin CHEN et al. *TensorFlow: A system for large-scale machine learning* [online]. 2016 [cit. 2023-05-22]. Dostupné z: doi:10.48550/arxiv.1605.08695

[42] CHOLLET, F. et al. *Keras* [online]. In: . 2015 [cit. 2023-05-22]. Dostupné z: https://keras.io

[43] PEDREGOSA, Fabian, Gaël VAROQUAUX, Alexandre GRAMFORT et al. Scikit-learn: Machine Learning in Python. *Journal of machine learning research* [online]. Microtome Publishing, 2011, (12), 2825-2830 [cit. 2023-05-22]. ISSN 1532-4435. Dostupné z: doi:10.5555/1953048.2078195

[44] PEZOA, F., J. L. REUTTER, F. SUAREZ, M. UGARTE a D. VRGOČ. Foundations of JSON schema. *Proceedings of the 25th International Conference on World Wide Web* [online]. 2016, 263-273 [cit. 2023-05-22]. Dostupné z: doi:10.1145/2872427.2883029

Seznam Obrázků

Obr. 1. Popis schématu fungování vztahu Agent-Prostředí. [8, s. 48] 13
Obr. 2. Neuronová síť, složená z jednoho neuronu se dvěma vrstvami. [10, s. 6] 14
Obr. 3. Neuronová síť s přidaným zkreslením. [10, s. 6] 15
Obr. 4. Neuronová síť se třemi vstupními neurony a dvěma výstupními neurony. [10, s. 8] 16
Obr. 5. Vícevrstvý perceptron se třemi skrytými vrstvami a jednou výstupní vrstvou. Neurony jsou značeny ail, kde l je index vrstvy, a i je výstupní index. [10, s. 9] 17
Obr. 6. Základní struktura jednoduché rekurentní neuronové sítě. [12, s. 8] 18
Obr. 7. Struktura Long-Short Term Memory (LSTM) sítě. [12, s. 8] 18
Obr. 8. Struktura Gated recurrent unit (GRU) sítě. [12, s. 8] 19
Obr. 9. Popis struktury konvoluční neuronové sítě. Na vstupní data jsou postupně použity sekvence konvoluce, aktivace a sdružovacích operací, které v konečném výsledku vytvoří výstupní předpověď. Parametry operací jsou učeny pomocí minimalizace ztrátové funkce. Derivace/gradient ztrátové funkce je počítán zpětnou propagací. [15, s. 8] 20
Obr. 10. Ilustrace konvoluční operace. Filtr o velikosti 3×3×3 je aplikován na vstupní data o velikosti 4×4 se třemi vstupními kanály. Je vypočítán skalární součin mezi vstupními daty a filtry. Hodnoty získané skalárním součinem jsou sečteny a tvoří výstup v levém horním rohu. [10, s. 28] 21
Obr. 11. Sdružovací proces s filtrem 2×2 a krokem posuvu 2. Maximální sdružovací vrstva vybírá ze sledované oblasti maximální hodnotu. Průměrová sdružovací vrstva počítá průměr ze sledované oblasti. [10, s. 28] 22
Obr. 12. Dropout funkce náhodně odebere jednotky z plně propojené sítě (a) a vzniká sub-model (b) [31] 32
Obr. 13. Průběh neuronové sítě znázorněn v jednotlivých vrstvách postupujících zleva doprava, kdy (a) není použit batch normalization, (b) batch normalization je použit [32] 32
Obr. 14. Tři po sobě jdoucí skryté vrstvy typu CNN v 1D CNN. [35] 33
Obr. 15. Architektura jednoduché konvoluční neuronové sítě. [36] 34
Obr. 16. Architektura multi-konvoluční neuronové sítě. [36] 34
Obr. 17. Vstupy do multi CNN. (a) vstupní hodnoty. (a), (b), (c), (d), a (e) znázorňují vstupy do $CNN1, CNN2, CNN3, CNN4, CNN5$. Modrá barva značí použité hodnoty vstupu, oranžová barva značí vynechané hodnoty vstupu. [36] 35
Obr. 18. Dopředná a zpětná propagace ve skrytých CNN vrstvách. [34] 37
Obr. 19. Dopředná a zpětná propagace mezi poslední skrytou vrstvou CNN a první skrytou vrstvou vícevrstvého perceptronu. [34] 37
Obr. 20. Struktura 1D konvolučního autoenkoder modelu [37] 38
Obr. 21. Vstupní data obsahující 8 1D signálů znázorňující jednotlivé průběhy teplot. 39
Obr. 22. Normalizovaná vstupní data obsahující 8 1D signálů znázorňující jednotlivé průběhy teplot. 40
Obr. 23. Schéma konvolučního autoenkodéru. 40
Obr. 24. Porovnání ztrátových a validačních ztrátových průběhů pro různé zvolení rozvržení filtrů. 42
Obr. 25. Porovnání ztrátových a validačních ztrátových průběhů pro různé zvolení velikostí kernelů. 44

Obr. 26. Porovnání ztrátových a validačních ztrátových průběhů pro různé optimalizační algoritmy. 45
Obr. 27. Porovnání ztrátových a validačních ztrátových průběhů pro různé ztrátové funkce. 46
Obr. 28. Porovnání ztrátových a validačních ztrátových průběhů pro kombinace aktivačních funkcí. 47
Obr. 29. Porovnání ztrátových a validačních ztrátových průběhů pro dodatečné regularizace. 48
Obr. 30. Predikce. Optimalizační algoritmus: Adamax, Ztrátová funkce: MSE, Aktivační funkce 1: LeakyReLU, Aktivační funkce 2: Swish, Rozložení kernelů: 3,3,3, Rozložení filtrů: 32,64,128. 50

Seznam Tabulek

Tabulka 1. Kombinace rozvržení filtrů. Optimalizační algoritmus: Adamax, Ztrátová funkce: MSE, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení kernelů: 3,3,3. 42
Tabulka 2. Kombinace rozvržení velikostí kernelů. Optimalizační algoritmus: Adamax, Ztrátová funkce: MSE, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení filtrů: 32,64,128. 43
Tabulka 3. Kombinace optimalizačních algoritmů. Ztrátová funkce: MSE, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení kernelů: 3,3,3. Rozvržení filtrů: 32,64,128. 44
Tabulka 4. Kombinace ztrátových funkcí. Optimalizační algoritmus: Adamax, Aktivační funkce 1: ReLU, Aktivační funkce 2: Sigmoid, Rozvržení kernelů: 3,3,3, Rozvržení filtrů: 32,64,128. 45
Tabulka 5. Kombinace aktivačních funkcí. Optimalizační algoritmus: Adamax, Rozvržení kernelů: 3,3,3, Rozvržení filtrů: 32,64,128, počet parametrů: 112584, Ztrátová funkce: MSE. 47
Tabulka 6. Volba dodatečných regularizací. Optimalizační algoritmus: Adamax, Aktivační funkce 1: Leaky ReLU, Aktivační funkce 2: Swish, Rozvržení kernelů: 3,3,3, Rozvržení filtrů: 32,64,128, Ztrátová funkce: MSE. 48

Seznam příloh

Příloha 1 – volba_filtru (grafy trénovací fáze a predikce jednotlivých modelů)
Příloha 2 – volba_kernelu (grafy trénovací fáze a predikce jednotlivých modelů)
Příloha 3 – volba_optimalizacniho_algoritmu (grafy trénovací fáze a predikce jednotlivých modelů)
Příloha 4 – volba_ztratove_funkce (grafy trénovací fáze a predikce jednotlivých modelů)
Příloha 5 – volba_aktivacnich_funkci (grafy trénovací fáze a predikce jednotlivých modelů)
Příloha 6 – volba_regularizace (grafy trénovací fáze a predikce jednotlivých modelů)
Příloha 7 – main.py (kód s neuronovou sítí a predikcí)
Příloha 8 – train_progress_comparation.py (kód pro vykreslování trénovacích porovnávání)