ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní - Ústav mechaniky, biomechaniky a mechatroniky

DIPLOMOVÁ PRÁCE

SYSTÉM PRO AUTOMATICKY OPTIMALIZOVANOU PREDIKCI ČASOVÝCH ŘAD NEURONOVÝMI SÍTĚMI

System for Automatically Optimized Time Series Prediction by Neural Networks

Patrik Zach

2021/2022

Prohlašuji, že jsem tuto práci vypracoval(a) samostatně s použitím literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací.

Datum: ....

podpis

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zach** Jméno: **Patrik** Osobní číslo: **473565**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Průmysl 4.0**

Studijní obor: **bez oboru**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Systém pro automaticky optimalizovanou predikci časových řad neuronovými sítěmi

Název diplomové práce anglicky:

System for Automatically Optimized Time Series Prediction by Neural Networks

Pokyny pro vypracování:

1) Proveďte rešerši metod predikce časových řad metodami neuronových sítí a metod pro nalezení/optimalizaci vstupního vektoru příznaků (kdy vstupem pro predikci časové řady je jedna nebo více časových řad), přičemž důležitým ohledem je i relativně omezené množství trénovacích dat (N<1E5 vzorků časové řady).
2) Popište stručně typy neuronových sítí a metody jejich učení se zaměřením na parametry dále použité v navženém systému. (HONU, MLP, ELM, FRVL, LSTM)
3) Kromě manuální volby vzorkování a prvků vstupního vektoru příznaků, navrhněte a implementujte i automatickou extrakci příznaků pomocí konvoluční vrstvy pro vámi použité neuronové sítě, řešení popište.
4) Navrhněte a naprogramujte SW systém pro trénování a testování predikce časových řad, který bude umožňovat trénovat a testovat predikce různých typů neuronových sítí pro zadaný soubor časových řad.
5) Systém otestujte na datasetu umělých i reálných časových řad.
délka práce: min.50 stran bez příloh
grafický obsah 50% max.

Seznam doporučené literatury:

Např.:
[1] JAKUB ČEPELA. Studie využití neuronových sítí pro predikci spotřeby komplexu budov. B.m.: Fakulta Strojní, ČVUT v Praze. Diplomová práce 2014. 00000
[2] BUKOVSKY, Ivo. Deterministic behavior of temperature field in turboprop engine via shallow neural networks. Neural Computing and Applications [online]. 2021, 33(19), 13145–13161. ISSN 0941-0643, 1433-3058. Dostupné z: doi:10.1007/s00521-021-06013-7
[3] TEALAB, Ahmed. Time series forecasting using artificial neural networks methodologies: A systematic review. Future Computing and Informatics Journal [online]. 2018, 3(2), 334–340. ISSN 23147288. Dostupné z: doi:10.1016/j.fcij.2018.10.003
[4] LIM, Bryan a Stefan ZOHREN. Time Series Forecasting With Deep Learning: A Survey [online]. 2020 [vid. 2022-04-11]. Dostupné z: doi:10.48550/ARXIV.2004.13408
[5] LARA-BENÍTEZ, Pedro, Manuel CARRANZA-GARCÍA a José C. RIQUELME. An Experimental Review on Deep Learning Architectures for Time Series Forecasting. International Journal of Neural Systems [online]. 2021, 31(03), 2130001. ISSN 0129-0657, 1793-6462. Dostupné z: doi:10.1142/S0129065721300011

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Ivo Bukovský, Ph.D. U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **22.04.2022** Termín odevzdání diplomové práce: **15.08.2022**

Platnost zadání diplomové práce: _____

doc. Ing. Ivo Bukovský, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Michael Valášek, DrSc. podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Anotace

Diplomová práce se zabývá vytvořením systému pro automatickou predikci časových řad. V teoretické části jsou představeny neuronové sítě, jejich historie, struktury, aktivační funkce a optimalizační algoritmy. Dále je představeno vývojové prostředí Django, v kterém je systém naprogramován. V části praktické jsou pak popsány vybrané funkce systému. Frontend webové aplikace je naprogramován pomocí jazyka HTML, CSS, JavaScript a backend webové aplikace pomocí Python. Jednotlivé struktury sítě jsou poté otestovány na umělých i reálných datasetech.

Klíčová slova: Neuronová síť, LNU, QNU, CNU, HONU, MLP, RVFL, ELM, LSTM, predikce časových řad, systém pro predikci časových řad

Abstract

The master's thesis deals with the creation of a system for automatic prediction of time series. In the theoretical part, neural networks, their history, structures, activation functions and optimization algorithms are presented. The Django development environment in which the system is programmed is also presented. Selected functions of the system are then described in the practical part. The frontend of the web application is programmed using HTML, CSS, JavaScript, and the backend of the web application is programmed using Python. Individual network structures are then tested on artificial and real datasets.

Keywords: Neural network, LNU, QNU, CNU, HONU, MLP, RVFL, ELM, LSTM, time series prediction, system for time series prediction

Poděkování

Děkuji doc. Ing. Ivu Bukovskému, Ph. D. za vedení diplomové práce.

Obsah

1 Úvod

Neuronové sítě spadají do kategorie supervizorovaného učení neboli učení s učitelem. Předpoklad naučení správného výstupu je datový vzorek, kde jsou obsažena jak data vstupní, tak data výstupní. Výstupním formátem dat je poté buď hodnota binární - klasifikace nebo spojitá - regrese. Naučené strojové učení po správném natrénování dokáže predikovat výstupní hodnoty.

S neuronovými sítěmi se dnes setkáme téměř všude, ať už se jedná o konvoluční neuronovou síť sloužící k rozpoznání objektu nebo obličeje, prediktivní údržbu či předpověď počasí. V této diplomové práci se budu zabývat dopřednými a rekurentními sítěmi pro predikci časových řad. Cílem je vytvoření systému, který bude zahrnovat několik architektur neuronových sítí, vybrané aktivační funkce, optimalizační algoritmy a volbu zpracování dat. Obsluha tohoto systému pak pomocí rozhraní stanoví kritéria neuronové sítě a tento systém se pak časovou řadu pokusí predikovat.

V teoretické části bude nastíněna teorie dopředných sítí - HONU (Higher Order Neural Unit), MLP (Multilayer Perceptron), ELM (Extreme Learning Machine), RVFL (Random Vector Functional Link) a rekurentních sítí - LSTM (Long Short Term Memory). Dále budou v teorii zahrnuty optimalizační algoritmy neuronových sítí, a to jak gradientní prvního i druhého řádu, tak i negradientní metody.

V následující části bude představeno vývojové prostředí Django, pomocí kterého je systém naprogramován. Postupně bude popsán vývoj této webové aplikace a její funkce. Lehce bude zmíněna i syntaxe programovacích jazyků.

V závěru práce bude tento systém otestován na umělých i reálných časových řadách.

2 Neuronové sítě

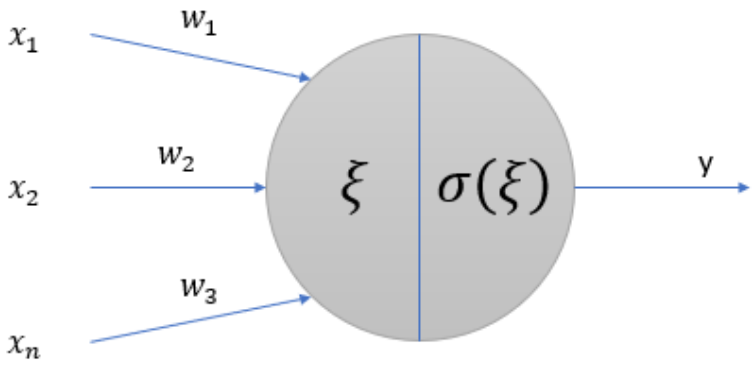
Neuronové sítě jsou výpočetní modely, které jsou inspirované biologickou strukturou neuronových sítí. Jsou tvořeny několika neurony, které jsou uspořádany ve vrstvách. Neurony z jednotlivých vrstev jsou vzájemně propojeny s vrstvami sousedními. Tato propojení jsou charakterizována pomocí tzv. vah, které slouží buď k zesílení nebo zeslabení vstupu na následující neuron a během procesu učení se aktualizují, aby došlo k minimalizaci nákladové funkce. Každý neuron transformuje vstupy pomocí aktivační funkce.

2.1 Historie neuronových sítí

Základní principy neuronových sítí byly formulovány roku 1943 Warrenem MuCullochem a Walterem Pittsem. Do neuronu vstupovaly binární hodnoty, byla provedena agregace a na základě agregované hodnoty bylo rozhodnuto o binárním výstupu. Tento neuron uměl základní booleovské operátory, např. AND, OR, NOT. [1] [2]

V roce 1949 se do vývoje neuronových sítí zapsal Donald Hebb a jeho *"Hebbian Learning Law"*. Tento postulát pochází z jeho knihy *"The Organization of Behavior"* a tvrdí, že synapse mezi neurony je posílena, pokud dochází k jejich korelaci a naopak zeslabena, pokud ke korelaci nedochází. [1]

Dalším důležitým milníkem v historii neuronových sítí bylo představení Perceptronu Frankem Rosenblattem v roce 1958. Jednalo se o dopřednou síť, která se skládala z jednoho neuronu. Tento neuron je binárním klasifikátorem a umí řešit pouze lineárně separovatelné úlohy, tzn. neumí vyřešit binární funkci XOR. Do neuronu vstupuje vektor reálných čísel , který je násobený váhami a na základě aktivační funkce, v tomto případě skokové funkce, je určen binární výstup. Do neuronu může vstupovat i hodnota prahu. [1] [3]



Obr. 1: Perceptron

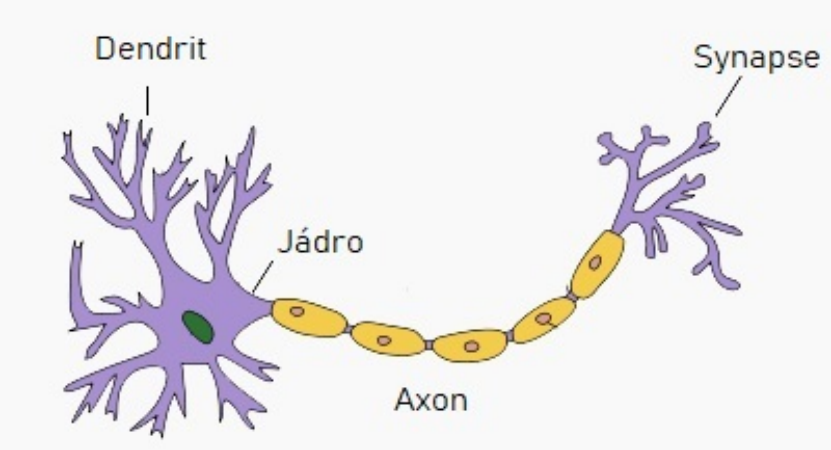
O dva roky později byla představena neuronová síť ADALINE Bernardem Widrowem, která byla velmi podobná Perceptronu. Rozdílem byl výstup sítě, kdy hodnota byla reálná a bylo tak možná realizace lineární funkce. [1] [4]

V roce 1986 nastal pro neuronové sítě přelom zveřejněním algoritmu zpětného šíření chyby Davidem Rumelhartem a dalšími. Pro aktualizaci vah využívá gradientní sestup a hledá globální minimum nákladové funkce. Algoritmus vypočítá chybu a ta je zpětně přepočítávána do předchozích vrstev a upravuje váhy. Dodnes patří k nejpoužívanějším algoritmům neuronových sítí. [1]

Za dva roky již byla představena síť MADALINE, která se skládala z několika neuronových sítí ADALINE. [1]

2.2 Matematický model neuronu

Vznik neuronový sítí byl silně inspirován funkcí biologického neuronu. Struktura biologického neuronu závisí na jeho roli a výskytu, ale téměř všechny obsahují 3 základní části, které jsou pro pochopení principu funkce neuronu dostačující. Buněčné jádro, které nese genetickou informaci, poskytuje energii pro řízení aktivit a udržuje strukturu neuronu. Dendrity, což jsou krátké výběžky z jádra, které přijímají signály. Dlouhý výběžek, tzv. axon, který slouží pro přenos informace mezi jednotlivými neurony. Axon je zakončen synapsí, která je spojena s dalším neuronem a přenáší vzruch, buď chemicky nebo elektricky. [5]



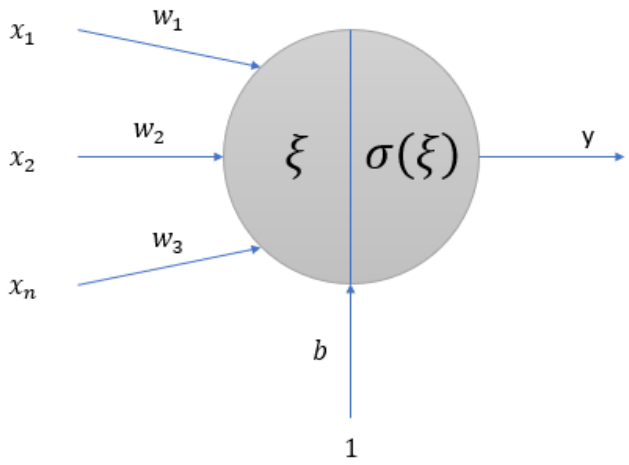
Obr. 2: Struktura nervové buňky, upraveno z [2]

Matematicky je tento model popsán tak, že do neuronu vstupuje vektor n reálných vstupů $X = [1, x_1, x_2, ...x_n]$, v biologické analogii dendrity, a ke vstupům je přidružen vektor synaptických vah $W = [b, w_1, w_2, ...w_n]$, jehož hodnoty nabývají i záporných hodnot. Vstup je tedy buď zesilován nebo utlumován. Součin těchto vektorů pak tvoří vnitřní potenciál neuronu. Bias b posouvá průběh aktivační funkce buď vlevo, nebo vpravo. Matematicky je

vnitřní potenciál neuronu vyjádřen:

$$\xi = b + \sum_{i=1}^{n} w_n \cdot x_n \tag{1}$$

Vnitřní potenciál neuronu poté vstupuje do aktivační funkce, která vnáší do systému nelineritu. Tento systém bývá označován jako jádro neuronu, výstupem jádra neuronu je axon. Existuje celá řada aktivačních funkcí, z nichž základní jsou představeny v následující kapitole.



Obr. 3: Neuron

2.2.1 Aktivační funkce neuronu

Skoková funkce

Aktivační skoková funkce se používala pro první modely neuronových sítí. Výstupem je binární hodnota. Dnes se již nepoužívá, protože není diferencovatelná.

Lineární funkce

Aktivační lineární funkce neboli žádná aktivace se používá pro lineární regresi. Nelze ji použít v hlubokých neuronových sítích, protože vstupní vrstva je lineární funkcí výstupní vrstvy. [5] [6]

ReLU - Rectified Linear Activation

Aktivační funkce ReLU je velmi populární pro konvoluční neuronové sítě a dopředné neuronové sítě. Její hlavní výhoda spočívá ve výpočetní efektivitě (je rychlejší než například sigmoida) a zejména v konstantní velikosti gradientu (derivace této funkce je konstantní), pokud je vstup do funkce > 0. To znamená, že v tomto intervalu nepodléhá tzv. *"vanish gradient"*. V záporném intervalu nastává problém tzv. *"Dying ReLU"*. Jelikož je v tomto

intervalu nulová derivace, optimalizační algoritmus, který je založený na gradient descent metodě, nebude upravovat váhy a dochází k umření neuronu. Proto existují další varianty ReLU např. Leaky ReLU, které tento problém odstraňuje - při vstupu záporných hodnot je výsledkem malá záporná hodnota. [7] [8]

Sigmoida

Aktivační funkce sigmoida je používána hlavně pro klasifikaci, kde je výstupní hodnota pravděpodobnost. Funkce je diferencovatelná na celém intervalu, avšak gradient této funkce velmi vzdálených bodů od počátku se blíží nule. To znamená, že v těchto bodech je učení pomalé a dochází tak k problému *"vanish gradient"*. Funkce se řadí také mezi výpočetně náročnější a nenabývá záporných hodnot. Pokud dochází ke klasifikaci, nemusí být součet všech tříd roven 1. [9]

Hyperbolický tangens

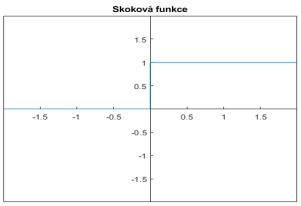
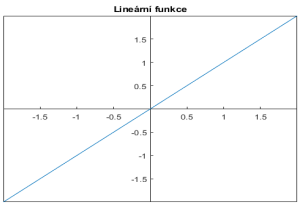
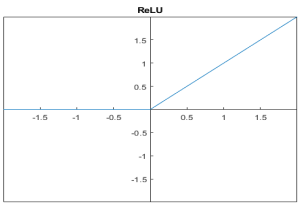
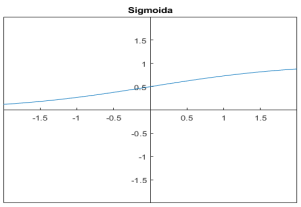
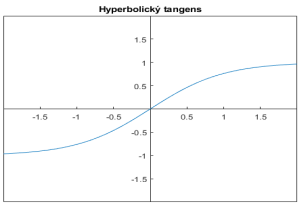
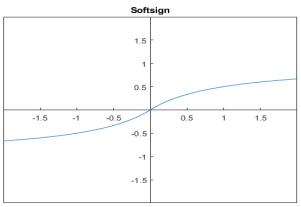
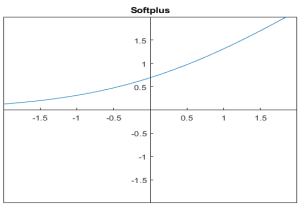
Aktivační funkce hyperbolický tangens je velmi podobná sigmoidě. Stejně jako sigmoida je výpočetně náročná, liší se větší derivací, protože se její výstupy pohybují v intervalu <-1 , 1>. Stejně jako u sigmoidy u vzdálených hodnot od počátku dochází k *"vanish gradient"*. [6]

Softsign

Aktivační funkce Softsign oproti hyperbolickému tangensu roste spíše polynomiálně než exponenciálně. Dochází tak k rychlejšímu učení a k méně častému *"vanish gradient"*. Na druhou stranu je výpočetně náročnější vzhledem ke složitější derivaci.

Softplus

Aktivační funkce Softplus je další variantou ReLU. Oproti Leaky ReLU se liší tím, že při vstupu záporných hodnot je výsledkem malá kladná hodnota.

Název aktivační funkce	Předpis aktivační funkce	Graf aktivační funkce
Skoková	$\sigma(\xi) = 1, \quad$ je-li $\quad \xi \geq 0$ $\sigma(\xi) = 0$, je-li $\xi < 0$	
Lineární	$\sigma(\xi) = \xi$	
ReLU	$\sigma(\xi) = max(0, \xi)$	
Sigmoida	$\sigma(\xi) = \frac{1}{1+e^{-a\xi}}$	
Hyperbolický tangens	$\sigma(\xi) = tanh(\xi)$	
Softsign	$\sigma(\xi) = \frac{\xi}{1+\lvert\xi\rvert}$	
Softplus	$\sigma(\xi) = log(1 + e^{\xi})$	

Tab. 1: Aktivační funkce

2.3 Modely dopředných neuronových sítí

Modely dopředných neuronových sítí jsou schopné predikování výstupu pouze pomocí aktuálních vstupů, vah a biasu. Aplikací tohoto druhu neuronových sítí existuje mnoho, například predikce časových řad [10] či klasifikace na základě tabulkových dat [11]. V následující kapitole jsou popsány principy vybraných dopředných neuronových sítí.

2.3.1 HONU - Higher Order Neural Unit

Jedná se o neuronovové jednotky, které vypočítávají výstup jako nelineární funkci lineárních kombinací vstupních neuronů prvního řádu. S rostoucím řádem polynomu roste účinnost neuronových jednotek, avšak roste počet optimalizovaných parametrů a tím stoupá časová náročnost. Dle stupňů polynomu dělíme HONU na LNU - Linear neural unit, QNU - Quadratic neural unit a CNU - Cubic neural unit. Obecný matematický popis pro vnitřní potenciál neuronu HONU pak je:

$$\xi = \sum_{i=0}^{n} \sum_{j=i}^{n} ... \sum_{...}^{n} (x_i x_j ... x) \cdot w_{i,j...} \tag{2}$$

Jelikož je aktivační funkce lineární, platí tento předpis i pro výpočet výstupu neuronu.
Rovnice (2) je pro jednotlivé řády vypadá následovně:
LNU - predikce pouze lineárních vztahů, polynom prvního řádu

$$\xi = \sum_{i=0}^{n} x_i \cdot w_i \tag{3}$$

QNU - predikuje nelineární vztahy, polynom druhého řádu

$$\xi = \sum_{i=0}^{n} \sum_{j=i}^{n} x_{i,j} \cdot w_{i,j} \tag{4}$$

CNU - predikuje nelineární vztahy, polynom třetího řádu

$$\xi = \sum_{i=0}^{n} \sum_{j=i}^{n} \sum_{k=j}^{n} x_{i,j,k} \cdot w_{i,j,k} \tag{5}$$

Jednotlivé matice z těchto rovnic mohou být převedeny do dlouhých vektorů $row(X)$ a $col(W)$, kdy dojde ke snížení optimalizačních parametrů, protože dojde ke snížení počtu potřebných vah. Dlouhé vektory $row(X)$ a $col(W)$ vypadají pro QNU takto:

$$row(X) = [x_{00}, x_{01}, x_{02}, ..., x_{0n}, ..., x_{nn}] \tag{6}$$

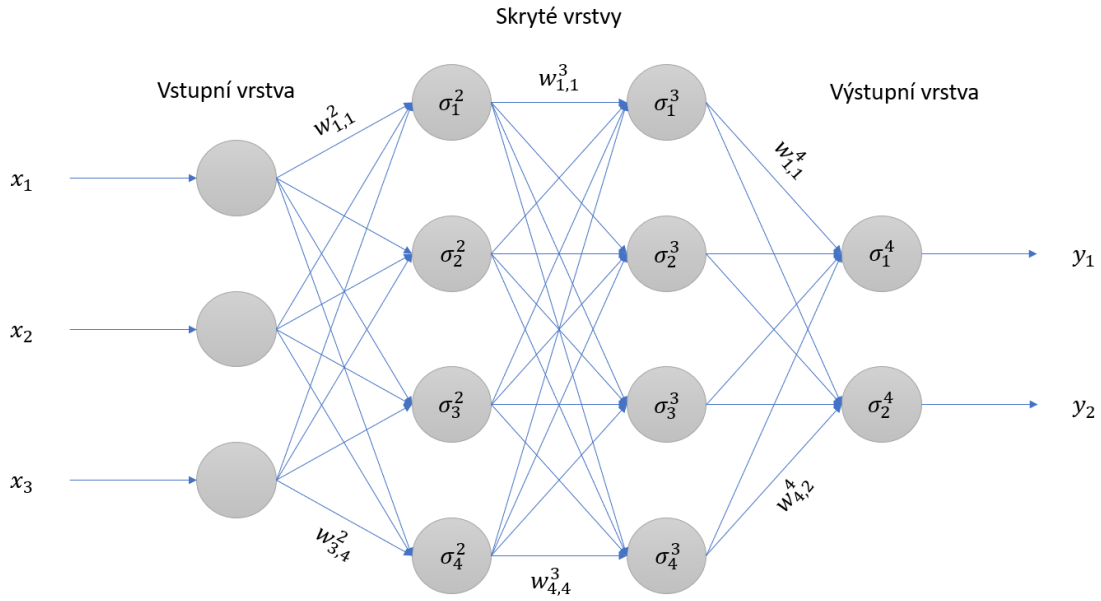
$$col(W) = [w_{00}, w_{01}, w_{02}, ..., w_{0n}, ..., w_{nn}]^T \tag{7}$$

A celkový vnitřní potenciál neuronu je roven:

$$\xi = row(X) \cdot col(W) \tag{8}$$

2.3.2 MLP - Multilayer Perceptron

Neuronová síť MLP je jednou z dalších dopředných sítí. Skládá se z minimálně 3 vrstev, z nichž jedna je vstupní, jedna výstupní a zbylé vrstvy se nazývají skryté. Do vstupní vrstvy vstupuje signál, který je zpracován a předán do každého neuronu skryté vrstvy. Ve skrytých vrstvách dochází k detekci příznaků a výstupní vrstva z těchto příznaků detekuje výstup. Struktura této sítě je zobrazena na následujícím obrázku.



Obr. 4: MLP neuronová síť se 2 skrytými vrstvami

Vstupní signál x_i je zesilován nebo zeslabován synaptickou vahou $w^2_{i,j}$. Výsledný součet všech upravených vstupních signálů je rozšířen o bias b^2_j. Aplikací aktivační funkce σ^2_j na vnitřní potenciál neuronu dostaneme poté výslednou hodnotu neuronu v 1. skryté vrstvě a^2_j:

$$a^2_j = \sigma^2_j(\sum_{i=1}^{n}(x_i \cdot w_{i,j}) + b^2_j), \tag{9}$$

kde n je počet vstupních signálů. Výslednou hodnotu v další skryté vrstvě získám obdobně:

$$a^3_k = \sigma^3_k(\sum_{j=1}^{o}(a^2_j \cdot w_{j,k}) + b^3_k), \tag{10}$$

kde o je počet neuronů v první skryté vrstvě. Výsledný výstup pak získám:

$$y_l = \sigma_l^4(\sum_{k=1}^{p}(a_k^3 \cdot w_{k,l}) + b_l^4), \tag{11}$$

kde p je počet neuronů v druhé skryté vrstvě. Celkový vztah pro výpočet výstupu pomocí vstupů je pak:

$$y_l = \sigma_l^4(\sum_{k=1}^{p}(\sigma_k^3(\sum_{j=1}^{o}(\sigma_j^2(\sum_{i=1}^{n}(x_i \cdot w_{i,j}) + b_j^2) \cdot w_{j,k}) + b_k^3) \cdot w_{k,l}) + b_l^4), \tag{12}$$

2.3.3 ELM - Extreme Learning Machine

Neuronová síť ELM je variací sítě MLP s jednou skrytou vrstvou. Využívá svého vlastního algoritmu a dosahuje menší výpočetní náročnosti díky menšímu počtu upravovaných parametrů. Může dokonce přinést i lepší schopnost generalizace. Trénování této sítě spočívá v inicializaci náhodných synaptických vah w_i a biasu b_i skryté vrstvy. Tyto parametry zůstavají během celého trénovacího procesu neměnné. Vektor vstupních hodnot vstupuje poté do náhodného prostoru, kde je na ně v každém skrytém neuronu aplikována aktivační funkce, nejčastěji se jedná o sigmoidu nebo hyperbolický tangens. Tyto hodnoty poté vstupují do výstupní vrstvy, kde jsou na ně aplikovány výstupní váhy β_i, které jsou jsou definovány na začátku trénovacího procesu, ale během tréninku jsou už měnné. Výstup neuronové sítě pak lze popsat:

$$y_j = \sum_{i=1}^{N} \beta_i \cdot \sigma(w_i \cdot x_j + b_i), j = 1, ..., M \tag{13}$$

kde N je počet skrytých neuronů. Konstantní část rovnice (8) se pro zjednodušení přepisuje do matice H o rozměru NxM, což je matice, která určuje hodnotu výstupu neuronu ze skryté vrstvy:

$$H = \begin{bmatrix} \sigma(w_1 \cdot x_1 + b_1) & \dots & \sigma(w_N \cdot x_1 + b_N) \\ \vdots & \vdots & \vdots \\ \sigma(w_1 \cdot x_M + b_1) & \dots & \sigma(w_N \cdot x_M + b_N) \end{bmatrix} \tag{14}$$

Poté lze výstupní hodnoty vyjádřit:

$$Y = H \cdot \beta, \tag{15}$$

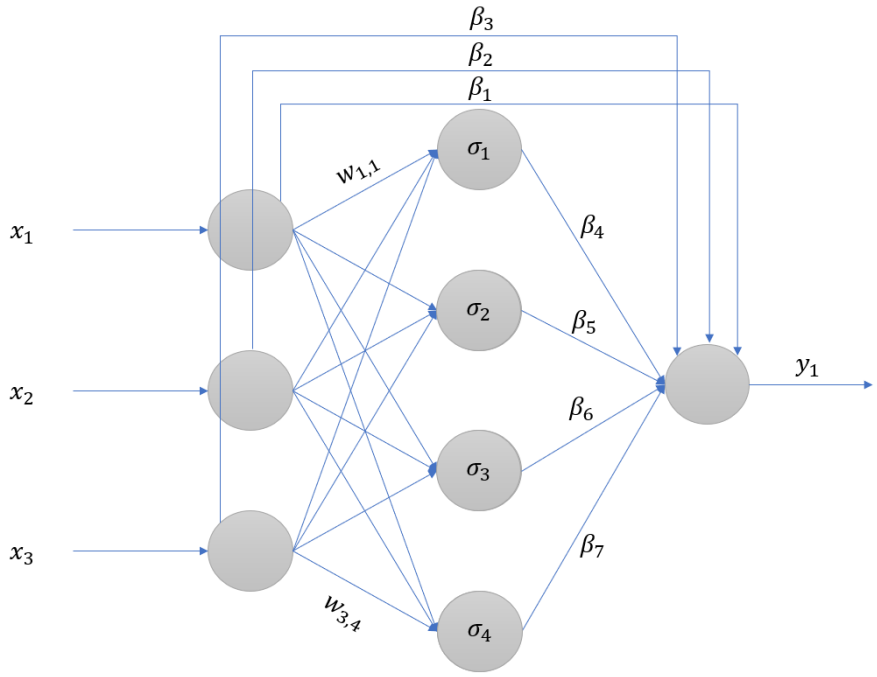
Z matice H je poté vypočtena Moore-Penroseova pseudoinverzní matice H^+. Poté platí

$$\beta = H^+ \cdot Y \tag{16}$$

Veškeré matematické vztahy byly převzaty z publikací [12] [13]. Volitelné parametry této sítě jsou pouze velikost skryté vrstvy a druh aktivační funkce. Velikost skryté vrstvy je zásadní navrhnout tak, abychom dosáhli požadované přesnosti predikce a časové náročnosti.

2.3.4 RVFL - Random Vector Functional Link

Poslední zmíněnou dopřednou neuronovou sítí je RVFL. Jedná se opět o variaci sítě MLP s jednou skrytou vrstvou, ale existují i odvozené sítě dRVFL, které skrytých vrstev mají více. Oproti síti ELM bývají méně složité - jsou menší. [14]



Obr. 5: RVFL neuronová síť

Tak jako u neuronové sítě ELM i zde jsou inicializovány váhy $w_{i,j}$ a biasy ve skryté vrstvě b_j, které po celou dobu tréninku zůstavají neměnné. Na sumu těchto vstupů a bias je v každém neuronu skryté vrstvy aplikována aktivační funkce σ_j. Váhy β_k jsou poté použity jak pro násobení neupravených vstupních signálů, což je tedy lineární komponenta výstupu, tak pro násobení neuronů ve skryté vrstvě, což je nelineární komponenta výstupu. K této sumě je nakonec přičten bias výstupní vrstvy β_l. Výstup neuronové sítě lze poté popsat:

$$y_l = \sum_{i=1}^{m} \beta_{i,l} \cdot x_i + \sum_{i=1}^{n} (\beta_{m+i,l} \cdot \sigma_i (\sum_{j=1}^{m} (x_j \cdot w_{j,i}) + b_i)) + \beta_l, \tag{17}$$

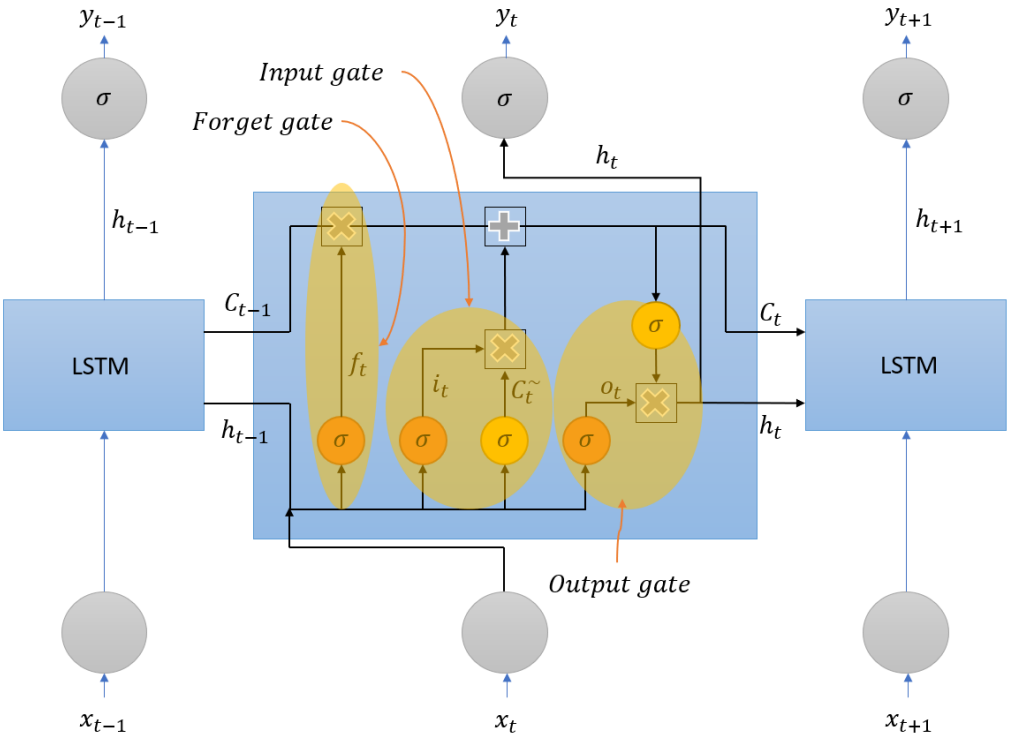
kde m je počet vstupních neuronů a n je počet neuronů ve skryté vrstvě. Konstantní část lze zapsat, stejně jako u sítě ELM, do matice H a postup je obdobný.

2.4 Modely rekurentních neuronových sítí

Modely rekurentních neuronových sítí počítají nejen s aktuálním vstupem, váhami a biasem, ale také s předchozími vstupy, které jsou uloženy do paměti. Aplikací pro rekurentní sítě je celá řada, například klasifikace fonémů [15], predikce časových řad [16] či detekce anomálií [17]. V následující kapitole je vysvětlen princip jedné z nejznámějších rekurentních sítí.

2.4.1 LSTM - Long Short Term Memory

Neuronová sít LSTM si pamatuje předchozí vstupní data a na základě těchto dat ovlivňuje výpočet aktuálního výstupu. Využívá se zejména pro sekvenční data, jelikož předchozí data v sekvenci ovlivňují aktuální stav. Oproti klasickým RNN se odlišuje délkou zapamatování sekvence, jelikož předchozí data je schopna filtrovat pomocí bran *"gates"*. To umožňuje zapamatování pouze podstatných informací, které do sítě vstoupily i před 1 000 kroky a nyní ovlivňují výstup. Následující obrázek zobrazuje, jak data prochází buňkou LSTM.



Obr. 6: LSTM neuronová síť

V horní části obrázku se nachází stav předchozí LSTM buňky C_{t-1}, do tohoto stavu mohou být zapisovány nebo odebírany informace pomocí bran. Do následující LSTM buňky vstupuje stav C_t. LSTM buňka obsahuje 3 brány:

- Forget gate
- Input gate
- Output gate

Forget gate

Jak již z názvu vypovídá, tato brána se stará o zapomínání informací, které již není nutné v daném kontextu znát. Do této brány vstupuje aktuální hodnota vstupu x_t a skrytý stav předchozí buňky h_{t-1} a platí vztah:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{18}$$

kde f_t je výstup, který se pohybuje v rozmezí od 0 - smazání dané informace do 1 - zanechání informace. [18] Rozsah výstupu udržuje aktivační funkce sigmoida σ, která je v celém intervalu diferencovatelná. W_f jsou pak váhy této brány a b_f bias této brány.

Input gate

Tato brána rozhoduje, která informace vstoupí do stavu LSTM buňky. Do brány opět vstupuje aktuální hodnota vstupu x_t a skrytý stav předchozí buňky h_{t-1} a nejprve se pomocí aktivační funkce rozhodne, které hodnoty se budou aktualizovat. V následujícím kroku je vytvořen vektor nových kandidátních hodnot C_t^-, které by mohly být přidány do stavu LSTM buňky. [18] Platí pak vztahy:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{19}$$

kde i_t je výstup, na který byla aplikována aktivační funkce sigmoida, tudíž se pohybuje v rozmezí 0 - do stavu nevstupuje tato hodnota do 1 - do stavu vstupuje tato hodnota. W_i jsou pak váhy této brány a b_i bias této brány.

$$C_t^- = \sigma(W_c \cdot [h_{t-1}, x_t] + b_c), \tag{20}$$

kde σ je aktivační funkce hyperbolického tangensu - větší rozsah a je taktéž diferencovatelný. W_c jsou pak váhy této brány a b_c bias této brány.
Následně aktualizujeme starý stav buňky C_{t-1} na nový stav buňky C_t pomocí následujícího vztahu:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t^- \tag{21}$$

Output gate

Poslední branou je tzv. Output gate, která rozhoduje o následujícím výstupu. Nejprve použijeme sigmoidní aktivační funkci, která rozhoduje, jaké části stavu buňky budou brány v potaz.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{22}$$

kde o_t je výstup pohybující se v rozmezí od 0 - tato hodnota nebude zahrnuta ve výstupu do 1 - tato hodnota bude zahrnuta ve výstupu. W_o jsou pak váhy této brány a b_o bias této brány. Následně vypočítáme skrytý stav aktuální buňky h_t:

$$h_t = o_t \cdot \sigma(C_t), \tag{23}$$

kde σ je aktivační funkce hyperbolického tangensu.

2.5 Algoritmy učení neuronové sítě

Algoritmy učení neuronové sítě jsou nezbytné pro správně předpovídání výstupu v závislosti na vstupních příznacích. Fungují v tzv. epochách, neboli cyklech, kdy postupně iterují synaptické váhy a biasy. Hledají tak nejlepší řešení pro trénovací data. Nejlepší řešení je vyhodnocené pomocí nákladové funkce, která může být definována několika způsoby. Pro představu zde uvadím 2 nejčastěji používané.

– Mean Squared Error - Průměrná čtvercová chyba

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^n)^2, \tag{24}$$

kde i je index výstupu, n je počet výstupů, y_i je skutečná hodnota a y_i^n je predikovaná hodnota neuronovou sítí.

– Mean Absolute Error - Průměrná absolutní chyba

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - y_i^n \mid \tag{25}$$

Rozdíl těchto hodnot y_i a y_i^n je poté definován proměnnou e, která následně figuruje u výpočtu nových vah a biasů u gradientových metod.
V této kapitole budou představeny gradientní metody učení neuronových sítí prvního a druhého řádu. Následně bude představeno zpětné šíření chyby, bez něhož by nebylo možné naučení hluboké neuronové sítě (neuronové sítě se skrytou vrstvou). Nakonec zde bude popsána i metoda genetického algoritmu, který dokáže natrénovat síť i bez použití gradientu.

2.5.1 Optimalizační algoritmy prvního řádu

Optimalizační algoritmy prvního řádu vyžadují alespoň jednu derivaci/gradient prvního řádu. V následujích řádcích stručně představím nejzákladnější algoritmy.

Batch Gradient Descent

Batch Gradient Descent patří mezi nejjednodušší optimalizační algoritmy. [19] Základní myšlenkou tohoto algoritmu je dosažení minimální hodnoty nákladové funkce pomocí vypočítávání směru, kterým by měly váhy být změněny. Matematicky lze tento algoritmus vyjádřit jako:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla J(\theta_t), \tag{26}$$

kde θ_{t+1} je matice vah a biasů následujícího kroku, θ_t je matice vah a biasů aktuálního kroku, α je rychlost učení a $J(\theta_t)$ je matice prvních derivací chyby vůči váhám a biasům. Pro jednotlivé váhy vypadá rovnice následovně:

$$w_{t+1}^i = w_t^i - \frac{1}{2} \cdot \alpha \cdot \frac{\partial e^2}{\partial w_t^i}, \tag{27}$$

kde i je index váhy a e chyba předpokládaného výstupu neuronové sítě a skutečného výstupu. Hlavní výhodou tohoto algoritmu je velmi jednoduchá implementace a snadné porozumnění. Oproti tomu, velmi často se stává, že optimalizace uvízne v lokálním minimu. Mezi další nevýhodu patří také výpočetní čas, vzhledem k tomu, že ke změně vah dochází až po výpočtu gradientu na celém datasetu.

Stochastic Gradient Descent

Vzhledem k výpočetní náročnosti algoritmu Batch Gradient Descent byla odvozena pravděpodobnostní aproximace Stochastic Gradient Descent. Jeho rychlost se projeví zejména u velkých souborů dat, kdy místo výpočtu gradientu pro celý soubor dat je náhodně zvolen bod v každé iteraci. Vzhledem k tomu, že gradient není počítán pro všechna data, dochází k vyššímu kolísání nákladové funkce, což ve výsledku může ztěžovat dosažení minima. [20] Oproti tomu zvýšená kolísavost může nákladovou funkci vymanit z lokálního minima a může dojít k nalezení minima globálního. Tento algoritmus má stejný předpis jako Batch Gradient Descent, pouze s rozdílem, že do něj vstupují náhodné body místo celého datasetu.

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla J_i(\theta_t), \tag{28}$$

kde i je index náhodného bodu.

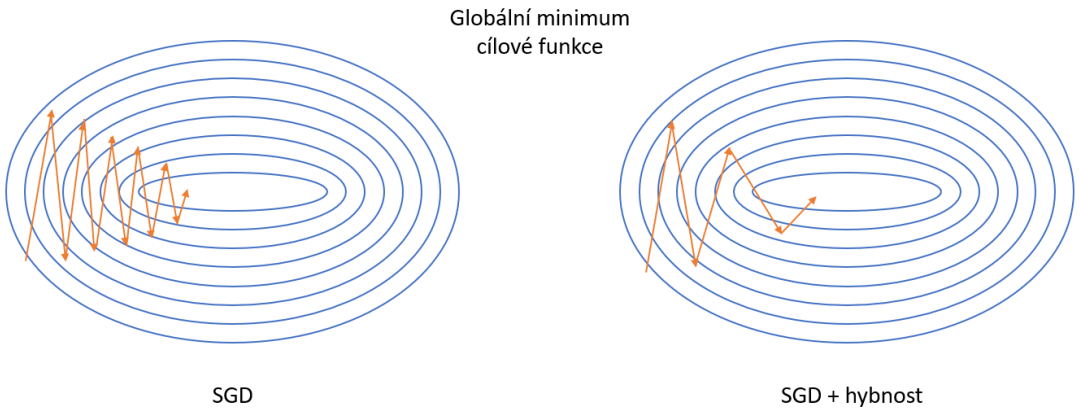
Mini Batch Gradient Descent

Další z variant gradientních optimilizačních algoritmů je Mini Batch Gradient Descent. Kombinuje oba algoritmy zmíněné dříve, kdy velký soubor dat rozdělí na menší dávky, tzv. *"mini-batch"*. Při správném rozdělení trénovacích dat pak dochází k menší kolísavosti nákladové funkce a oproti Batch Gradient Descent také k výráznému snížení výpočetní náročnosti. Oproti předešlým algoritmům je nutná konfigurace dalšího parametru - velikost dávky (*"mini-batch size"*). Tento algoritmus se řadí mezi nejpoužívanější pro trénování hlubokých neuronových sítí. [21] Matematicky lze vyjádřit obdobně, do Jacobiho matice vstupuje vždy jen *"mini-batch"*.

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla J_{B(i)}(\theta_t), \tag{29}$$

Stochastic Gradient Descent + momentum

Stochastic Gradient Descent s tzv. hybností. Hybnost urychluje gradientní sestup správným směrem na základě předchozí aktualizace vah a tím dochází k rychlejší konvergenci díky exponenciálně váženým průměrům.



Obr. 7: Porovnání SGD a SGD s hybností

K aktualizaci vah poté platí vztah

$$\theta_{t+1} = \theta_t - v_t, \tag{30}$$

kde v_t je hybnost, která je definována:

– podle Polyaka [22]

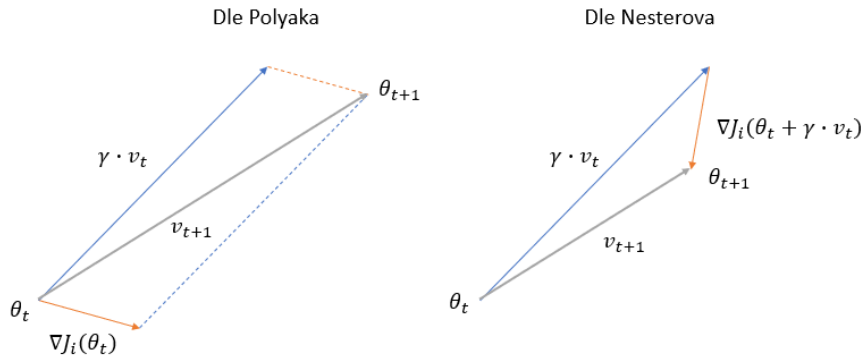
$$v_{t+1} = \gamma \cdot v_t + \alpha \cdot \nabla J_i(\theta_t), \tag{31}$$

kde γ je koeficient hybnosti.

– podle Nesterova [22]

$$v_{t+1} = \gamma \cdot v_t + \alpha \cdot \nabla J_i(\theta_t + \gamma \cdot v_t) \tag{32}$$

Výpočet momentu podle Nesterova se odlišuje tím, že výsledný krok gradientu počítáme z bodu již posunutého o krok hybnosti. Vizualizaci tohoto výpočtu jsem naznačil v následujícím obrázku.



Obr. 8: Hybnost podle Polyaka a podle Nesterova

Adaptive Gradient

Optimalizační algoritmus Adaptive Gradient, zkráceně AdaGrad, pracuje s proměnnou rychlostí učení α. Využívá se zejména pro práci s řídkými daty, jelikož pro méně časté příznaky zvětšuje rychlost učení a pro časté příznaky ji zmenšuje. [23] Největší výhodou tohoto algoritmu je, že parametr rychlosti učení můžeme ponechat výchozí, obvykle $\alpha = 0.01$. Předpis tohoto algoritmu je roven:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta_t), \tag{33}$$

kde ϵ je kladné číslo s nízkou hodnotou, obvykle 10^{-8} a G_t je diagonální matice součtu druhých mocnin předchozích gradientů.
Jelikož jsou kvůli matici G_t druhé mocniny ve jmenovateli, znamená to, že součet gradientů během iterací neustále roste. Rychlost učení se pak může blížit k nule a algoritmus se přestane učit.

AdaDelta

Optimalizační algoritmus AdaDelta je odvozen od již zmíněného algoritmu AdaGrad. Je o něco robustnější, jelikož během iterací neshromažďuje všechny minulé gradienty, ale toto nashromáždění je omezené. Parametr rychlosti učení stejně jako u AdaGrad lze ponechat

výchozí. Předpis tohoto algoritmu můžeme vyjádřit jako:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E_t + \epsilon}} \cdot \nabla J(\theta_t), \tag{34}$$

kde E_t je závislá na předchozí matici E_{t-1} a aktuálním gradientu:

$$E_t = \gamma \cdot E_{t-1} + (1 - \gamma) \cdot \nabla J^2(\theta_t), \tag{35}$$

kde γ je volitelný parametr, obvykle $\gamma = 0.9$ a ovlivňuje zahrnutí předchozí matice E_{t-1} a aktuálního gradientu do nově vzniklé matice.

Adam

Posledním představeným algoritmem prvního řádu je Adam, Adaptive moment estimation. Vychází z již zmíněného AdaDelta, kde pro výpočet jedné z hybností v_t také používá exponenciálně klesající průměr minulých kvadratických gradientů. Tento algoritmus je rozšířen o další hybnost m_t, která zahrnuje exponenciálně klesající průměr minulých gradientů. [23]

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla J(\theta_t) \tag{36}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \nabla J^2(\theta_t) \tag{37}$$

kde β_1 je volitelný parametr, exponenciální míra poklesu pro první hybnost, obvykle se volí $\beta_1 = 0.9$ a β_2 je exponenciální míra poklesu pro druhou hybnost, obvykle $\beta_2 = 0.999$. [24] Následně jsou tyto odhady korigované dle autorů:

$$m_t^{\wedge} = \frac{m_t}{1 - \beta_1^t} \tag{38}$$

$$v_t^{\wedge} = \frac{v_t}{1 - \beta_2^t} \tag{39}$$

Předpis toho algoritmu je vyjádřen následující rovnicí:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t^{\wedge}} + \varepsilon} \cdot m_t^{\wedge} \tag{40}$$

Tento algoritmus je ze všech představených nejvíce sofistikovaný a nejvíce výpočetně efektivní s ohledem na nízké nároky na paměť. Je vhodný jak pro řídká data, tak data obsáhlá. Všechny parametry obvykle nejsou náročné na ladění.

2.5.2 Optimalizační algoritmy druhého řádu

Optimalizační algoritmy druhého řádu používají derivace gradientů druhého řádu, aby rychleji konvergovaly k minimu.

Levenberg-Marquardt

Optimalizační algoritmus Levenberg-Marquardt je dávkovým algoritmem druhého řádu, který funguje bez přesného výpočtu Hessovy matice, což je čtvercová matice druhých parciálních derivací. Uvažuje se nákladová funkce, která zahrnuje čtvercovou chybu všech tréninkových dat. Gradient chybové funkce je definován jako:

$$g = J^T \cdot e, \tag{41}$$

kde J je Jacobiho matice prvních derivací chyby vůči vahám a biasu a e je vektorem chyb. Hessova matice je poté aproximována jako:

$$H \approx J^T \cdot J + \lambda I, \tag{42}$$

kde I je jednotková matice a λ je tzv. faktor tlumení. Pokud je faktor tlumení $\lambda = 0$, učení se chová podle metody Newton-Gauss. Pokud je faktor tlumení příliš velký, algoritmus se začne chovat jako Gradient Descent s malým krokem. Změna vah je poté dána následující rovnicí.

$$\theta_{t+1} = \theta_t - (J_t^T \cdot J_t + \lambda_t I)^{-1} \cdot (J_t^T \cdot e_t) \tag{43}$$

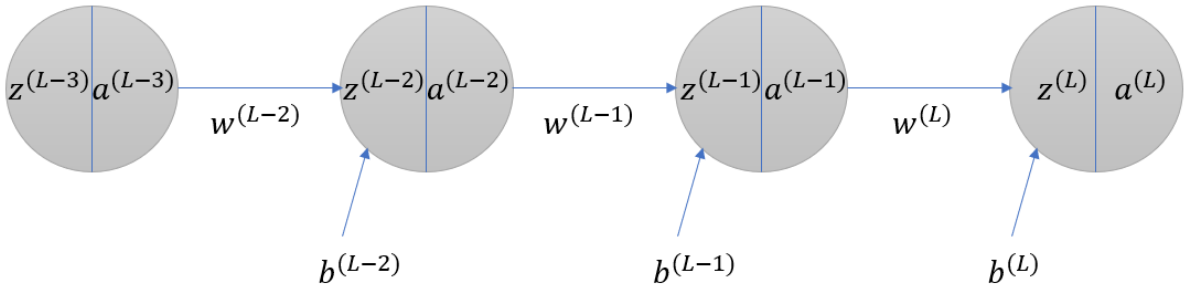
Algoritmus je inicializován s větším faktorem λ tak, aby k prvním aktualizacím vah došlo pomocí malých kroků ve směru gradientu. Postupně je snižován, aby se Levenberg-Marquadt choval podle metody Newton-Gauss. Tímto postupem se z algoritmu stává jeden z nejrychlejších algoritmů druhého řádu. [25]

2.5.3 Backpropagation - Zpětné šíření chyby

Algoritmus zpětného šíření chyby se používá u vícevrstvých neuronových sítí. Jedná se o speciální případ gradientního sestupu, kdy výpočet nových vah v síti se tzv. zpětně šíří, to znamená, že je nejprve vypočten gradient finální vrstvy, následně se postupuje směřem k vrstvě vstupní. Tento algoritmus funguje efektivněji, než výpočet gradientu každé vrstvy zvlášť. [26] V následujících řádcích bude princip algoritmu vysvětlen.
Změny vah iterujeme podle již zmíněného algoritmu:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\partial E}{\partial \theta_t}, \tag{44}$$

kde E je nákladová funkce.
Následně je nutná definice neuronové sítě a aplikace řetězového pravidla. Princip řetězového pravidla bude vysvětlen na jednoduché neuronové síti, která je vyobrazena na následujícím obrázku.



Obr. 9: Neuronová síť pro vysvětlení řetězového pravidla

Tato neuronová síť je definována několika vrstvami, které obsahují vždy jeden neuron. Velikost nákladové funkce pak závisí na jednotlivých vahách a biasech, konkrétně $w^{(L-2)}, w^{(L-1)}, w^{(L)}, b^{(L-2)}, b^{(L-1)}$ a $b^{(L)}$, kde v horním indexu se nachází index vrstvy, nejedná se tedy o mocninu. [27], [28]
Pokud uvažujeme nejčastěji používanou nákladovou funkci MSE, tak její předpis je roven:

$$E = (y - a^{(L)})^2 \tag{45}$$

Výstup posledního neuronu $a^{(L)}$ je pak roven:

$$a^{(L)} = \sigma(w^{(L)} \cdot a^{(L-1)} + b^{(L)}) \tag{46}$$

Pomocí potenciálu vnitřního neuronu $z^{(L)}$ lze tuto rovnici vyjádřit jako:

$$a^{(L)} = \sigma(z^{(L)}) \tag{47}$$

Následně je zkoumáno, jak změny jednotlivých parametrů sítě ovlivňují nákladovou funkci - řetězové pravidlo.

$$\frac{\partial E}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial E}{\partial a^{(L)}} \tag{48}$$

Pro jednotlivé komponenty této rovnice pak platí:

$$\frac{\partial E}{\partial a^{(L)}} = 2(a^{(L)} - y) \tag{49}$$

Změna velikosti $a^{(L)}$ má vysoký vliv na velikosti nákladové funkce.

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}), \tag{50}$$

kde σ' je derivací aktivační funkce.

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \tag{51}$$

Pro bias pak platí:

$$\frac{\partial E}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial E}{\partial a^{(L)}} = \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y) \tag{52}$$

Pro neuron, který se nachází v předchozí vrstvě poté platí:

$$\frac{\partial E}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial E}{\partial a^{(L)}} = w^{(L)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y) \tag{53}$$

Tímto způsobem se iteruje přes všechny váhy až do první vrstvy. Můžeme tak zjistit, jak moc jednotlivé váhy a biasy předchozích vrstev ovlivňují nákladovou funkci. Derivace této nákladové funkce je pak průměrem přes všechna tréninková data.

2.5.4 Genetické algoritmy

Mezi další učící algoritmy neuronových sítí patří i genetické algoritmy, které jsou inspirovány evoluční biologií. Tento algoritmus je inicializován počáteční populací, z které každý jedinec představuje jedno řešení problému. Zdatnost těchto jedinců poté určuje fitness funkce, která je počítána pro všechny jedince během přechodu do nové generace. Tak jako u evoluční biologie, i zde je prováděno křížení, mutace a reprodukce.
Princip tedy spočívá v tom, že pro novou generaci jedinců je vypočtena fitness funkce a je vybráno několik zdatných jedinců. Poté jsou generováni další jedinci následující generace a to pomocí:

- Křížení - kdy dva zdatní jedinci vytvoří dva nové - buď aritmeticky, nebo pomocí prostého křížení
- Mutace - kdy s určitou pravděpodobností se část jedince změní na náhodné číslo z předem definovaného intervalu
- Reprodukce - kdy se zdatný jedinec přesune do následující generace beze změny

Následuje další výpočet fitness funkce a výpočet končí tehdy, kdy je dosáhnuto předem definované kvality řešení nebo maximálního počtu generací. Jedinec s nejvyšší zdatností pak představuje řešení problému.
Výhodou tohoto algoritmu je, že v hlubokých neuronových sítí není potřeba zpětného šíření chyby. Mezi další výhody patří i to, že výpočet není závislý na gradientu, což může u některých aktivačních funkcí v případě nulového gradientu znamenat, že se gradientní metodou již nebudou učit. U aktivační funkce ReLU může nastat, že vnitřní potenciál neuronu je záporný a neuron tzv. "umírá". To u negradientních metod nehrozí. Nevýhodou je pak časová náročnost výpočtu, která je ovlivněna jak složitostí sítě, tak počtu dat, vzhledem k tomu, že pracuje na celém vzorku dat.

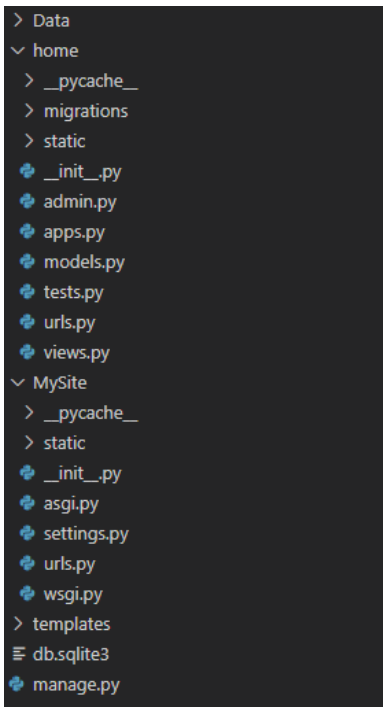
3 Vývojové prostředí - Django

3.1 Důvod volby a instalace

Webové prostředí bylo vytvořené za pomoci komplexního frameworku Django založeném na programovacím jazyce Python. Tato metoda byla zvolena z několika důvodů:

- Vývojové prostředí je open-source, navíc využívá jazyku Python, což patří mezi jednodušší jazyky a je také velmi dobře čitelný. Python se navíc používá pro implementaci jak věděckých výpočtů, tak i pro umělou inteligenci, která je podporována několika knihovnami - např. TensorFlow, PyTorch, OpenCV.
- Pomocí tohoto frameworku lze vytvářet robustní a bezpečné webové stránky, jsou i dobře škálovatelné a rychlé. Vzhledem k těmto výhodám je použit i velmi známými společnostmi jako je YouTube a Instagram. [29]
- Django je vytvářeno webovými vývojářemi, jsou tak vyřešeny všechny obecné problémy a obsahuje obrovské množství funkcí. [30] Vzhledem k této vlastnosti, je zde předdefinováno několik proměnných, s kterými je nutno se seznámit. Také je nutno dodržovat některá pravidla, např. dodržovat strukturu souborů.

Instalace Djanga byla provedena příkazem přes příkazový řádek. Po instalaci byl vytvořen projekt opět přes příkazový řádek. Tento projekt byl následně otevřen ve Visual Studio Code, což je open-source pracovní prostředí. Struktura tohoto projektu vypadá následovně.



Obr. 10: Adresářová struktura projektu

3.2 Struktura projektu

Na předchozí ilustraci byla znázorněna struktura Django projektu, v následujících řádcích budou jednotlivé soubory stručně představeny.

- admin.py - Administrace jednotlivé aplikace.
- apps.py - Konfigurace jednotlivé aplikace.
- models.py - Používá se pro tvorbu databází, např. PostgreSQL, lze se napojit i na MySQL.
- tests.py - Testování aplikace.
- settings.py - Obsahuje nastavení pro správné fungování databáze, nastavení pro hledání statických souborů, nastavení pro webové aplikace a jiné.
- urls.py - Definice url adres a také definice nasměrování na část aplikace, tzv. routování.
- views.py - Hlavní výpočetní jádro celého webu, přebírá požadavky http (request) a vrací odpověď (response) HTML dokumentům.
- wsgi.py - Rozhraní brány webového serveru, specifikuje jak webový server komunikuje s webovými aplikacemi.
- manage.py - Jedná se o nástroj příkazového řádku, který se používá pro spouštění různých příkazů, nejčastěji pro spouštění serveru, migrování databáze a spouštění aplikací.
- Složka Data - V této složce se nachází data určená pro trénování neuronových sítí (vstupy a výstupy) ve formátu csv, npy.
- Složka static - Zde se nachází soubory, které jsou použity na webovém rozhraní, ať už se jedná o png, jpg, gif.
- Složka templates - Ve složce templates jsou uloženy všechny šablony potřebné pro webové rozhraní ve formátu html.

3.3 Tvorba frontendu

Frontend webové aplikace byl naprogramován pomocí jazyků HTML (HyperText Markup Language), CSS (Cascading Style Sheets) a JavaScriptu. HTML je značkovací jazyk, který je používán pro tvorbu struktury stránek v systému World Wide Web, což umožňuje publikování webového rozhraní na internet. CSS popisuje způsob zobrazení elementů definovaných v HTML. JavaScript pak umožňuje webové rozhraní udržovat interaktivní. V následujícím kódu je naprogramováno velmi jednoduchý frontend.

```
1  <!-- Importováni header šablony -->
2  {% include 'header.html' %}
3  <!-- Importováni složky static -->
4  {% load static %}
5  <!-- Informace o typu dokumentu -->
6  <!doctype html>
7  <!-- Informace o jazyku, kterým je psán -->
8  <html lang="en">
9  <!-- Metadata -->
10   <head>
11     <!-- Kódování -->
12     <meta charset="utf-8">
13     <!-- Škálování -->
14     <meta name="viewport" content="width=device-width, initial-scale=1">
15     <title>Vzor pro diplomovou práci</title>
16     <!-- Deklarace stylu v programovacím jazyku CSS -->
17     <style>
18         p, h1   {color: aliceblue;
19             }
20         #ID-selector {background-color:  aqua;
21             }
22         .class-selector {padding-left: 0.1rem;
23             }
24         p:hover {color:  red;
25             }
26     </style>
27   </head>
28 <!-- Definuje tělo dokumentu -->
29 <body>
30 <div class="class-selector" id="ID-selector">
31     <p>Example</p>
32 </div>
33 </body>
34 <!-- Definuje Javascript -->
35 <script>
36     var exampleElement = document.getElementById('ID-selector');
37     exampleElement.style.color = "green";
38 </script>
39 </html>
```

Obr. 11: Zdrojový kód: Definice základního frontendu

V kódu lze vidět aplikaci těchto jazyků, jedná se pouze o ukázku řešení, tento skript nekomunikuje s backendem a jedná se tak čistě o vizualizaci. Na řádcích 17 - 26 lze vidět zapsané styly jednotlivých elementů pomocí jazyka CSS, postupně se jedná o element selektor, poté je definován ID selektor, selektor třídy a element selektor s akcí. Na řádcích 29

- 33 lze vidět tělo dokumentu, které je psané HTML, je zde znázorněna deklarace ID a třídy elementu. Na řádcích 35 - 38 je definována změna barvy elementu pomocí JavaScriptu. Ten se používá spíše pro tvorbu funkcí, např. po kliknutí. Skript je záměrně velmi jednoduchý a v podstatě téměř nic nedělá. Cílem této ilustrace byla zejména názornost struktury dokumentu a syntaxe programovacích jazyků.

3.4 Tvorba backendu

Backend webové aplikace je naprogramován v jazyce Python. Ve složce views.py jsou definovány všechny potřebné funkce, které běží na serveru. K přesměrovávání url adres slouží skript urls.py. Pro ilustraci zde uvádím velmi jednoduchý backend.

```
1   #ulrs.py
2   from django.contrib import admin
3   from django.urls import path, include
4   from home import views
5
6   urlpatterns = [
7       path('admin/', admin.site.urls),
8       path('', views.home, name='home'),
9       path('results/', views.result, name='results'),
10  ]
11
12  #views.py
13  from django.shortcuts import render, HttpResponse
14
15  def home(request):
16      if request.method == 'GET':
17          return render(request, 'home.html')
18      if request.method == 'POST':
19          context = {}
20          return render(request, 'results.html', context)
```

Obr. 12: Zdrojový kód: Definice základního backendu

Na řádku 1 - 10 je naznačena struktura urls.py skriptu, kde je nutné definovat jednotlivé url adresy a k ním musí být přiřazena funkce, která je pro názornost znázorněna na řádcích 15 - 20. Na řádcích 16 - 17 je znázorněno, jaký frontend se má zobrazit při *GET requestu*. V případě *POST requestu* (řádky 18 - 20) se do funkce dostávají data z frontendu a zde může být definována další funkce, která s daty pracuje a výsledek zasílá pomocí dictionary na frontend - context.
Na základě předchozího popisu lze říci, že požadavek *HTTP GET* je používán pro vyžádání webové stránky, zatímco požadavek *HTTP POST* se používá pro odesílání dat na server, např. pomocí webového formuláře.

4 Systém pro trénování a testování neuronových sítí

V této kapitole bude představen naprogramovaný systém pro trénování a testování neuronových sítí.

4.1 Frontend webové aplikace

Nejprve bude představena vizuální stránka této aplikace. Všechny skripty ve formátu *html* jsou uloženy ve složce *templates*.

4.1.1 Použité knihovny

Pro usnadnění vývoje projektu byly použity 2 knihovny, které jsou v následujících odstavcích stručně představeny.

Bootstrap

Jedná se o frontendovou sadu nástrojů, která obsahuje definované CSS styly, což usnadňuje vývoj responzivních webů. Bootstrap umožňuje vytvářet stránky rychleji, jelikož při správném použití není nutné definovat základní funkce a příkazy. Vzhledem k tomu, že se jedná o populární framework, existuje velmi dobře zpracovaná dokumentace. Pomocí tohoto nástroje je uživatel schopen rychle vytvořit layout webové stránky, formátovat obrázky, tabulky a grafy.

Chart.js

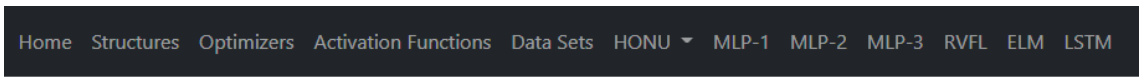
Jedná se o knihovnu JavaScriptu, která umožňuje vizualizaci dat pomocí grafů. Tato knihovna disponuje rychlým renderováním dat ve všech moderních prohlížečích a dokáže přizpůsobovat graf velikosti stránky.

4.1.2 Programování softwaru

Složka *templates* obsahuje několik souborů ve formátu *html*.

header.html

Tento skript obsahuje navigační lištu, pomocí které se lze překlikávat na ostatní naprogramované stránky. V tělu tohoto skriptu se nachází jednotlivé sekce, které obsahují odkazy. Při najetí myši na odkaz je zde upravený kurzor myši a pole se zvýrazní.



Obr. 13: Navigační lišta

```
<body>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
    data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
    aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link " aria-current="page" href="/">Home</a>
        </li>
```

Obr. 14: Zdrojový kód: Část definice navigační lišty pomocí knihovny Bootstrap

home.html

Na této stránce se nachazí kontejner, který definuje jednotlivá pole do mřížkové struktury. V této sekci je pomocí gifů stručně představeno, co webové rozhraní obsahuje.

structures.html

V této záložce jsou představeny jednotlivé modely neuronových sítí s popisem a ilustrací uspořádáné do mřížkové struktury.

optimizers.html

Zde se nachází představení optimalizačních algoritmů v mřížkové struktuře.

activationfunctions.html

V tomto skriptu se uživatel může dozvědět informace o jednotlivých aktivačních funkcích, každá funkce je doplněna grafem funkce.

datasets.html

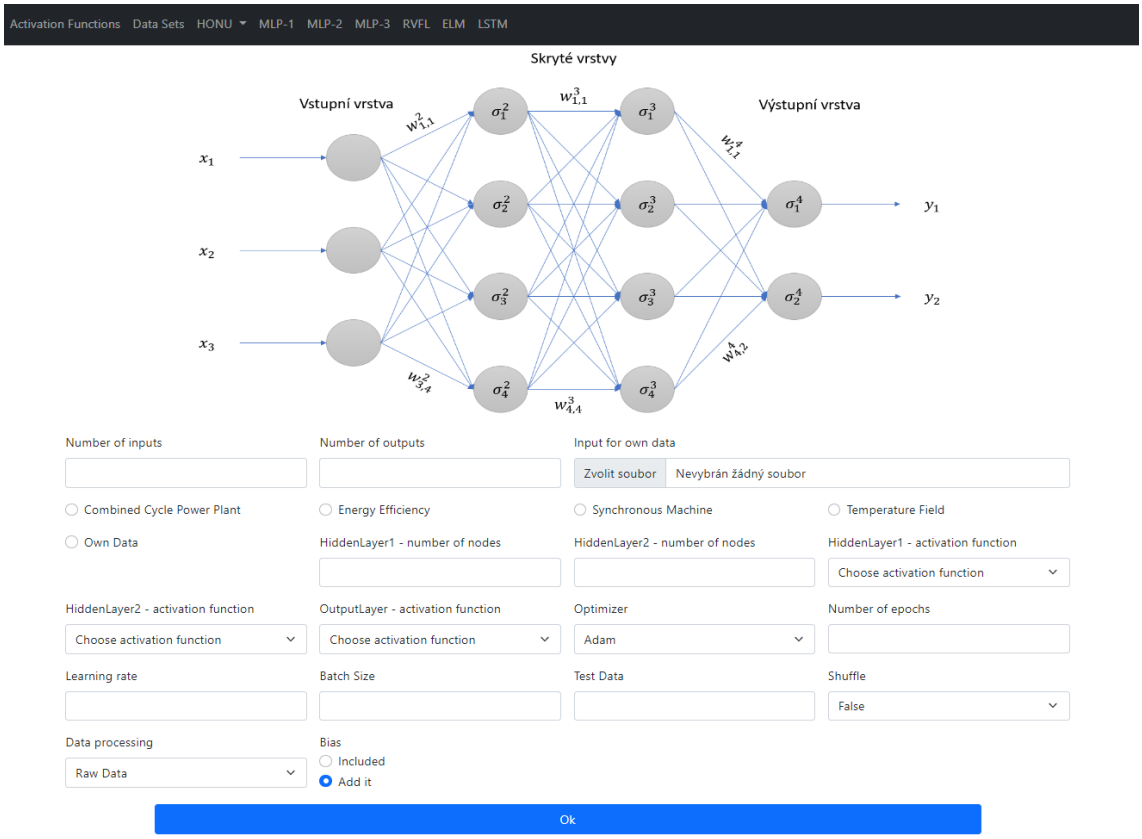
Tento skript obsahuje informace o nahraných datech na serveru, které uživatel může použít pro učení a ověření funkčnosti neuronových sítí. Rozhraní je ale rozšířeno i o nahrání vlastních dat a uživatel není tedy vůbec limitován.

LNU.html, QNU.html, CNU.html

V záložce HONU se po kliknutí zobrazí rozbalovací menu, pomocí kterého se uživatel může přesunout na jednotlivý neuronový model. V těchto skriptech se nachází formulář, pomocí kterého si uživatel může zvolit data, které chce pro trénování neuronové sítě použít, nebo zda-li chce použít data vlastní. Dále je zde nutné zvolit počet epoch, koeficient učení, zpracování dat a přidání biasu. Po volbě parametrů a stisknutí tlačítka jsou data odeslána na server.

MLP1.html, MLP2.html, MLP3.html, RVFL.html, ELM.html, LSTM.html

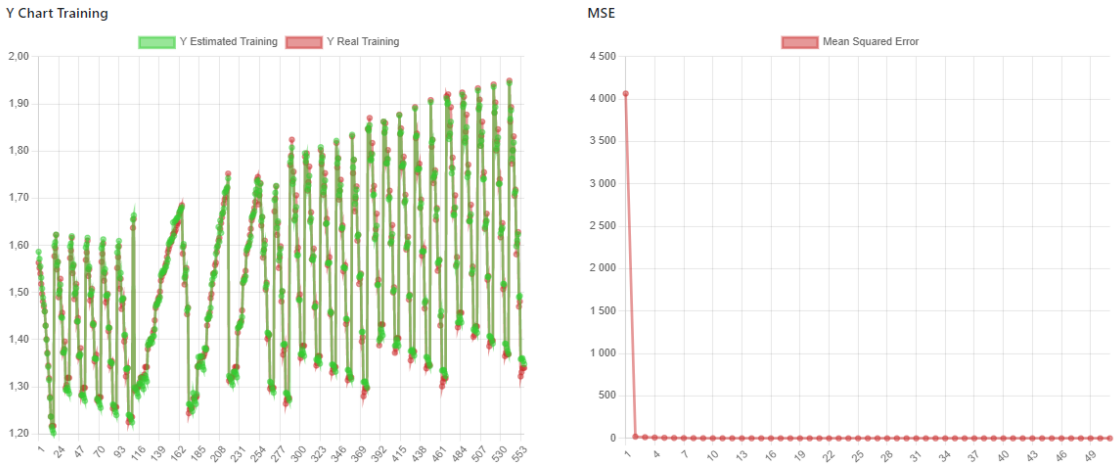
V těchto skriptech se opět nachází formuláře, pomocí nichž si uživatel navolí parametry neuronové sítě. Kromě předchozích zmíněných se zde nachází počet neuronových uzlů ve skrytých vrstvách, druh aktivační funkce, algoritmus, kterým se má neuronová síť učit, velikost dávky, procentuální zastoupení testovacích dat a možnost data promíchat, což není vhodné pro časové řady.



Obr. 15: Vizualizace formuláře MLP se 2 skrytými vrstvami

results.html

Po odeslání formuláře na server jsou data získané od uživatele použita v neuronové síti. Výsledky této neuronové sítě jsou poté interpretovány pomocí grafů použité pomocí knihovny Chart.js. Na rozhraní se nachází 4 grafy - zobrazení reálných hodnot a predikovaných hodnot během procesu trénování neuronové sítě, velikost čtvercové chyby v jednotlivých epochách, zobrazení reálných hodnot a predikovaných hodnot během procesu testování neuronové sítě a velikost vah během jednotlivých epoch.



Obr. 16: Vizualizace dat pomocí Chart.js

4.2 Backend webové aplikace

V této kapitole budou představeny vybrané funkce, které běží na straně serveru.

4.2.1 Použité knihovny

Na serveru je importováno několik knihoven. V následujících řádcích jsou představeny některé vybrané.

Keras

Jedná se o vysokoúrovňové API, které běží na TensorFlow frameworku. Využívá se zejména pro zrychlenou tvorbu neuronových sítí poměrně jednoduchou syntaxí, kde nejprve je definována struktura sítě, dále je model kompilován a pomocí tréninkových a validačních dat jsou váhy modelu přizpůsobovány definovaným algoritmem.

NumPy

NumPy je knihovna, která se používá zejména pro výpočty. Pomocí této knihovny je možné vytvořit vícerozměrné pole (matice) s pevnou velikostí a stejným datovým typem. Dále lze s poli zjednodušeně pracovat, jelikož NumPy definuje i několik pokročilejších matematických, logických i tvarových operací. V porovnání s listy definované pomocí jazyku Python jsou tyto operace několikrát rychlejší.

Dále byla použita knihovna pandas, pomocí které byla importovaná data, knihovna scikit-learn, která byla použita pro PCA (Analýza hlavních komponent), knihovna math, která obsahuje matematické operace a konstanty a knihovna random, která generuje náhodné čísla.

4.2.2 Programování softwaru

Funkce GetDataFromForm

Tato funkce importuje data ze složky *Data* na základě volby uživatele. Také zpracovává data uživatelem nahrané pomocí formuláře ve formátu *csv*, kde data jsou oddělena středníkem a desetinná hodnota je oddělena čárkou.

Funkce DatasetProcessing

V této funkci je definováno několik forem zpracování dat, konkrétně se jedná o:

- normalizaci dat pomocí minimálních a maximálních hodnot v datasetu, což způsobí lepší rozložitelnost dat,
- standardizaci dat pomocí metody *Z-scoring*, kdy hodnota průměrné hodnoty v datasetu je rovna 0 a hodnota posunuta o +1 standartní odchylku je rovna +1,
- analýzu hlavních komponent, kdy je redukován počet vstupních veličin s minimální ztrátou informace,
- konvoluční vrstvu, která redukuje délku dat a může redukovat i počet vstupních veličin. Jedná se o tzv.*pooling*, kdy jsou vybrána dvou rozměrná pole o několika prvcích a tato pole jsou zredukována do 1 prvku. Například je pole nahrazeno průměrnou hodnotou všech prvků zahrnutých v poli nebo největší hodnotou prvku v poli.
- posun dat, kdy se posouvají hodnoty vstupních dat od výstupních. Tento princip lze využít např. u časově zpožděného výstupu, kdy dojde ke zjištění, že vstupní hodnoty dosahují největší korelace s výstupními hodnotami o posun (*lag*)
- přidání biasu

Funkce ELMDataPreparation, RVFLDataPreparation

Pomocí této funkce jsou vytvářeny vstupní veličiny do sítí typu ELM a RVFL, kdy je vypočítáván vnitřní potenciál neuronu ve skryté vrstvě.

Funkce nazvané typem modelu

Funkce nesoucí název typu modelu vykresluje uživateli grafické rozhraní. Při requestu POST pak zpracovává data z formuláře, řídí informační tok a nakonec získaná data vykresluje uživateli. Do tohoto odstavce patří funkce LNU, QNU, CNU, MLP1, MLP2, MLP3, RVFL, ELM, LSTMpage (zde je název pozměněn vzhledem k importu neuronu typu LSTM z knihovny Keras, aby nedocházelo ke kolizi).

Funkce nazvané typem modelu + Model

Tyto funkce obsahují modely neuronových sítí definované pomocí knihovny Keras. Do těchto funkcí patří MLP1Model, MLP2Model, MLP3Model, RVFLModel, ELMModel a LSTMModel.

Funkce GraphDataProcessing Tato funkce převádí data získané neuronovou sítí do slovníku (*dictionary*), který je definován tak, aby hodnoty byly správně vykresleny na frontendu (názvy a styl).

Dále je definováno několik funkcí pro výpočet neuronové sítě typu HONU a další funkce pro zobrazení html.

```
949  def MLP1Model(hidden_layer1, colsX, hidden_layer1_func,
950                colsY, output_layer_func,
951                optimizer, learning_rate, X, Y, batch_size,
952                epochs, shuffle, test_data):
953
954      model = Sequential([
955          Dense(units=hidden_layer1, input_shape=(colsX,),
956              activation=hidden_layer1_func),
957          Dense(units=colsY, activation=output_layer_func)
958      ])
959      model.summary()
960      model.compile(optimizer=Adam(learning_rate=learning_rate),
961      loss='mean_squared_error',
962      metrics=['mean_absolute_percentage_error'])
963      history = model.fit(x=X, y=Y, batch_size=batch_size,
964                          validation_split=test_data, epochs=epochs,
965                          shuffle=shuffle, verbose=2)
966      predictions = model.predict(X)
967      loss = history.history['loss']
968
969      return predictions, loss
```

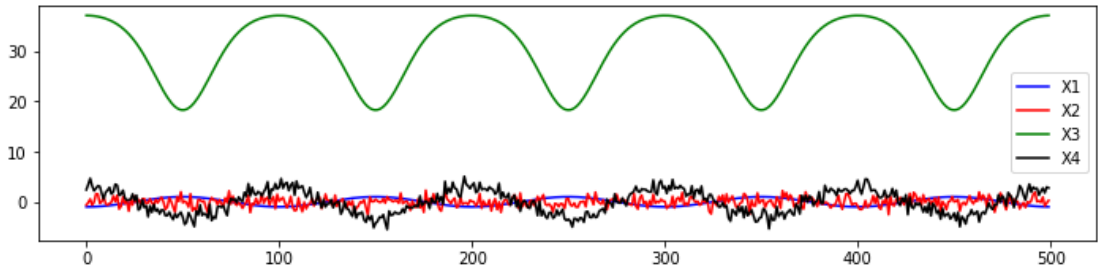
Obr. 17: Zdrojový kód: Funkce MLP1Model

5 Použitá data

V následujících podkapitolách budou představeny data, která byla použita pro trénování a validaci neuronových sítí.

5.1 Umělá data

Pro testování webové rozhraní byla vytvořena umělá data. Data jsou vytvořena tak, že mezi některými proměnnými existuje souvislost bez časového posunu, nebo s časovým posunem. Na následujícím obrázku jsou tato data zobrazena.



Obr. 18: Průběh vytvořených umělých dat

Předpis těchto uměle vytvořených dat vypadá následovně:

$$X1(t) = cos(\frac{2 \cdot \pi}{t}) \tag{54}$$

$$X2 = randn(N) \tag{55}$$

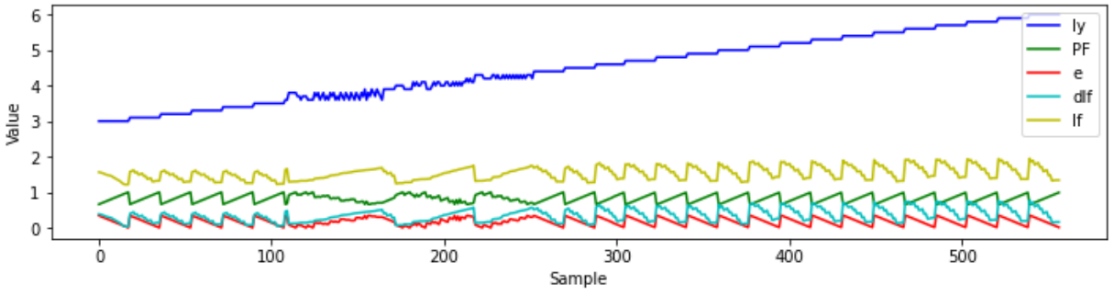
$$X3(t) = 40 - 8 \cdot e^{X1} \tag{56}$$

$$X4(t) = X2 - 3 \cdot X1[t-1]) \tag{57}$$

Tyto rovnice byly vytvořeny pro t = 50, 51 ... N, kde N = 550. Funkce $randn$ generuje náhodná čísla normálního rozdělení. Čísla v hranatých závorkách pak znázorňují časový posun. Z předpisů lze vidět souvislosti mezi jednotlivými proměnnými. Lze si povšimnout, že mezi proměnnými $X1$ a $X2$ neexistuje souvislost žádná. Predikování hodnoty $X1$ pomocí hodnoty $X2$ by tak při správném fungování neuronové sítě měla být mizivá. Pokud se bude jednat o složitou neuronovou síť, tréninková data by mohla být předpovídána se slušnou přesností, avšak validační set dat už nikoliv. Umělá data jsou úmyslně navržena tak, aby funkce $X2$ simulovala šum, který je pak zahrnut ve funkci $X4$.

5.2 Synchronní motor

Tento dataset obsahuje hodnoty měřených veličin zátěžového proudu Iy, účiníku PF, chybu účiníku e a změnu budícího proudu synchronního stroje dIf. Jako výstup je pak měřena hodnota budícího proudu If. Všechna data jsou měřena na reálném stroji v reálném čase v experimentálním provozním prostředí. Tato data byla získána z archivu UCI. [31]
Průběh těchto dat je zobrazen na následujícím grafu.

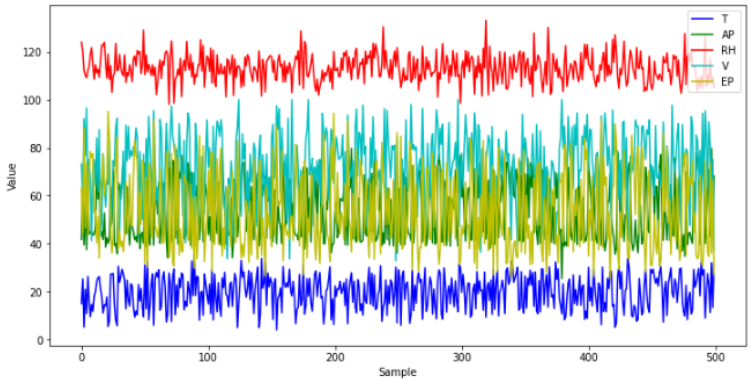


Obr. 19: Průběh dat z měření synchronního motoru

Z grafu je patrné, že mezi veličinami e, dIf a If probíhá velmi silná korelace.

5.3 Elektrárna s kombinovaným cyklem

Tento dataset obsahuje data získaná z měření, které bylo zaznamenáváno po dobu 6 let s frekvencí 1 hodiny. Dataset obsahuje průměrnou teplotu T, okolní tlak AP, relativní vlhkost RH a vyčerpání vakua V. Tyto hodnoty pak slouží k predikci hodinového výstupu elektrické energie EP. Tato data byla získána z archivu UCI, kde je měření detailněji popsáno. [32]

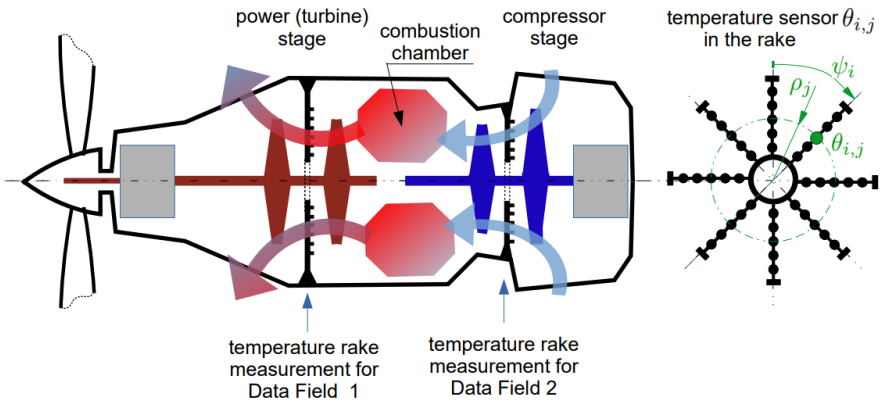


Obr. 20: Průběh dat z měření elektrárny

V předchozím grafu bylo vykresleno prvních 500 hodnot datasetu. Hodnoty relativní vlhkosti byly sníženy o hodnotu 900 a hodnota hodinového výstupu elektrické energie o 400. .

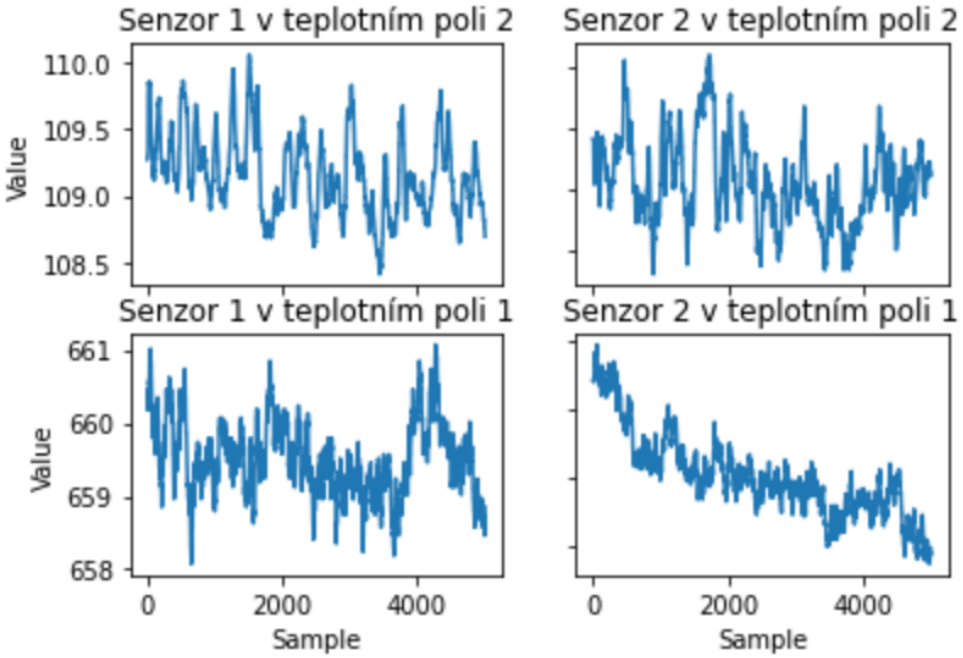
5.4 Teplotní pole v turbomotoru

Tento dataset obsahuje měření 2 teplotních polí v turbomotoru. Teplotní pole jsou měřena hřebenovými sondami s frekvencí 100 Hz. První teplotní pole se nachází za spalovací komorou, kde se nachází 7 hřebenových sond, kde každá z nich nese 5 termočlánků na odlišných průměrech. Druhé teplotní pole je měřeno před spalovací komorou, teplotu zaznamenávají 4 hřebenové sondy opět s 5 termočlánky na 5 odlišných průměrech. Měření je zaznamenáváno 440 sekund. Schéma měření je zobrazeno na následujícím obrázku.



Obr. 21: Schéma měření teplotních polí v turbomotoru. Převzato z [33]

Celkem je tedy zaznamenáno 35 průběhů teplot v teplotním poli 1 a 20 průběhů teplot v teplotním poli 2. Tato data byla poskytnuta vedoucím práce. V následujícím grafu je vykreslen průběh vybraných senzorů prvních 5 000 hodnot.



Obr. 22: Průběh dat z měření teplotních polí turbomotoru

6 Predikce časových řad pomocí webové aplikace

V této kapitole bude provedena predikce časových řad pomocí vybudovaného rozhraní. Nejprve bude ověřena funkčnost rozhraní na předem vytvořených umělých datech, dále bude následovat aplikace na data reálná.

6.1 Validace funkčnosti neuronových sítí

Funkčnost neuronových sítí jsem se rozhodl otestovat na předem definovaných umělých datech v kapitole 5.1. Jelikož některé proměnné jsou od sebe časově posunuté, musela být tato skutečnost zachycena buď posunem vstupů od výstupů, nebo vytvořením časového okna, které by zahrnovalo i vstupy v předchozích krocích, např. k předpovídání výstupu na základě 10 předchozích vstupech. Jelikož se jedná o data umělá, hodnota posunu vstupu a výstupu neuronové sítě je předem známá a vytvoření časového okna zde není nutností. V případě reálných dat je nutné prostudování zpoždění výstupu na vstup, k tomu může dopomoci prostudování děje nebo vykreslení grafů korelace v závislosti na vzájemném posunu.

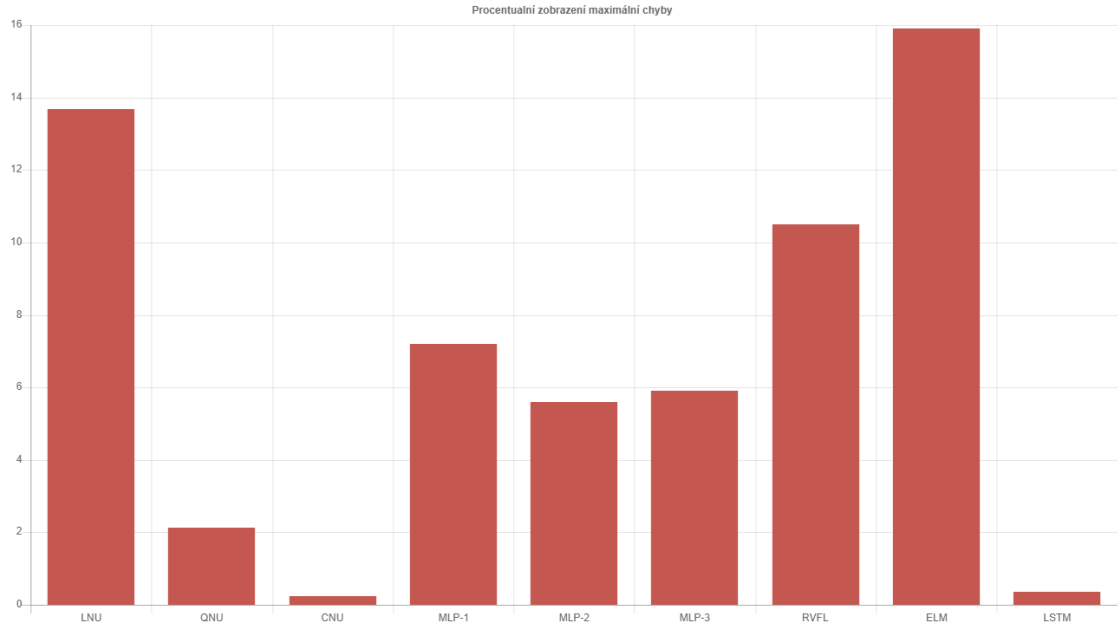
Nejprve jsem se pokusil o predikci hodnoty $X3$ pomocí hodnoty $X1$. Tyto rovnice jsou závislé pomocí exponentu a nejsou od sebe časově posunuté. Definice jednotlivých sítí jsou pak znázorněny v následující tabulce.

Neuronová síť	HONU	MLP-1	MLP-2	MLP-3	RVFL	ELM	LSTM
Rychlost učení	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Počet epoch	200	200	200	200	200	200	200
Počet neuronů	1, 2, 3	1-2	1-2-2	1-2-2-2	20	30	20
Aktivační funkce	lin.	sigm.	sigm.	sigm.	sigm.	lin.	lin.
Opt. algoritmus	LM	Adam	Adam	Adam	Adam	Adam	Adam
Data	/	Norm.	Norm.	Norm.	Norm.	/	/

Tab. 2: Definice neuronových sítí, které byly použity pro umělá data

Dosažené výsledky byly zaneseny do sloupcového grafu, kde faktorem je největší procentuální chyba, vždy se jednalo o hodnotu na konci intervalu hodnot. Jelikož data vstupovala do sítí jinak zpracována, funkčnost dané sítě nemohla být znázorněna chybou MSE.

V případě sítě LNU se predikování výstupu $X3$ nepodařilo zcela přesně zachytit, v některých místech předpovídaná hodnota kopíruje hodnotu skutečnou, avšak v hodnotách na krajích intervalu se předpovídaná hodnota odlišuje. To je dáno zejména tím, že 1 neuron nedokáže zcela zachytit exponenciální charakteristiku. Lepších výsledků dosáhla síť QNU, kde při stejných parametrech bylo dosaženo několikanásobně lepšího výsledku. Síť CNU už výstup



Obr. 23: Maximální procentuální chyba předpovědi

předpovídala téměř bez chyby.Všechna data, která vstupovala do neuronové sítě HONU byla použita bez úpravy, standardizovaná, normalizovaná. Ve všech případech datového předzpracování došlo k velmi podobnému výsledku.

Výsledky sítě MLP s jednou skrytou vrstvou pak velmi závisela na zvolených parametrech, u lineární aktivační funkce funguje stejně jako LNU, zde byla tedy nutná úprava aktivační funkce. Jako nejlepší řešení se jevilo využití aktivační funkce sigmoida s normalizovanými daty. Tímto krokem byla dosaženo lepší přesnosti. Lehké zlepšení výsledků pak nastalo u MLP se 2 skrytými vrstvami a se 3 skrytými vrstvami. U sítě RVFL došlo k nejlepšímu výsledku s obdobnými parametry.

Síť ELM nedokázala dostatečně predikovat pomocí jiných zvolených funkcí, než lineární, a to i přesto, že data vstupovala po standardizaci či normalizaci. Nejlepšího výsledku dosáhla beze změny vstupních dat, chová se tedy podobně jako LNU. Nutno však podotknout, že pro správné fungování této sítě musí vstupovat více proměnných. Síť LSTM pak zvládla predikovat výstup téměř přesně, bez úprav parametrů.

V dalším kroku jsem provedl testování neuronových sítí na vstupu, který je zatížen šumem, jelikož reálná data bývají zašuměná. Pokusil jsem se o predikci hodnoty $X1$ pomocí hodnoty $X4$, která obsahuje složku $X1$ a $X2$, což simuluje šum. Hodnoty $X1$ jsou zároveň posunuté o jedno měření, bude tedy nutné vstupní data posunout od výstupu, nebo vytvořit časové okno. Abych síť udělal více složitou, rozhodl jsem se ke vstupu přidat i 5 předchozích měření. Na následujících grafech lze vidět, že sítě jsou schopny predikovat výstup i z dat, které jsou zatížená šumem. V prvním grafu lze vidět neuronovou síť CNU, kde parametry této sítě jsou podobné, jako v předem definované tabulce. Změna je počet neuronů, jelikož

vstupních proměnných je nyní 6 a v tomto testu byla data standardizovaná. V druhém grafu je znázorněna predikce sítě LSTM.

Obr. 24: Neuronová síť CNU - predikce ze zašuměných dat

Obr. 25: Neuronová síť LSTM - predikce ze zašuměných dat

Postupně jsem predikci, která je zatížená šumem, vyzkoušel pro ostatní neuronové sítě. Data byla tentokrát standardizovaná pro sítě typu HONU a LSTM a normalizovaná pro sítě MLP a RVFL. U sítí se skrytou vrstvou byl pak počet neuronů navýšen. Pod tímto odstavcem se nachází tabulka, která zobrazuje, jak se dané sítě vypořádaly se šumem, jako ukazatel je tentokrát průměrná čtvercová chyba.

Neuronová síť	LNU	QNU	CNU	MLP-1	MLP-2	MLP-3	RVFL	LSTM
MSE	0.086	0.063	0.027	0.08	0.08	0.06	0.08	0.04

Tab. 3: Průměrná čtvercová chyba v závislosti na druhu sítě

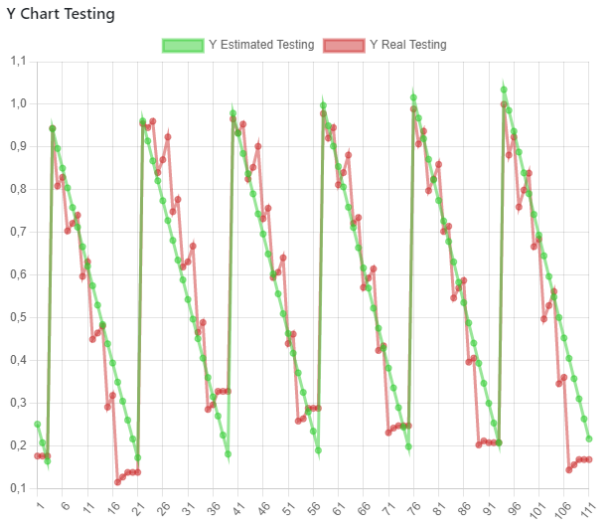
Všechny struktury neuronových sítí jsou tak schopné predikovat i zašuměná data.

6.2 Aplikace neuronových sítí na reálná data

V této kapitole budou postupně řešeny jednotlivé reálné datasety.

6.2.1 Synchronní motor

Predikce budícího proudu u synchronního motoru byla velmi snadno dosažena pomocí nejjednodušší sítě LNU, to je dáno zejména velmi silnou korelací chyby účiníku a změny budícího proudu s predikovaným výstupem. Jelikož vstup a ani výstup není časově opožděn a data nejsou příliš zašuměná, pokusím se predikovat výstup bez chyby účiníku a změny budícího proudu.
Ze všech testovaných struktur sítí se LSTM projevila jako nejvhodnější pro tento problém. Tato síť zvládla se slušnou přesností predikovat výstup. Do natrénované sítě byly pak vloženy testovací data, která ověřovala, zda je neuronová síť naučena správně. Výsledek predikce testovacích dat je zobrazen v následujícím obrázku.



Obr. 26: Neuronová síť LSTM - predikce budícího proudu

Výsledky této sítě se nelišily ani po standardizaci či normalizaci dat. Učení sítě bylo nastaveno na 0.01, počet LSTM buněk na 20, optimalizační algoritmus Adam a počet epoch 200. Testovací data se skládala z 20 % všech dat. Zvětšením počtu buněk bylo dosáhnuto podobných výsledků, zmenšením pak došlo ke zhoršení predikce. Slušných výsledků dosahovaly i sítě MLP a RVFL.

Aplikací touto neuronovou sítí můžeme dosáhnout vcelku obstojných dat bez skutečného měření budícího proudu, změny budícího proudu a chyby účiníku.

6.2.2 Elektrárna s kombinovaným cyklem

Vzhledem k povaze dat, které byly odečítány s frekvencí hodiny se nejedná o časově zpožděný děj. Hodnoty těchto měření nabývají různě velikých hodnot a ideální zpracování

dat se jeví jako standardizace, vzhledem k tomu, že nás zajímají výkyvy hodnot od obvyklých stavů. Metodou Z-scoring jsem tedy všechna data standardizoval. Následně jsem se pokusil najít ideální strukturu sítě a parametry této sítě.

Nejprve jsem se pokusil nastavit ideální parametry pro síť HONU, zde se vyskytuje jediný proměnný parametr - rychlost učení. Pokud docházelo ke zvyšování parametru, suma čtvercové chyby dosahovala hodnoty až 0.063 pro QNU a CNU. Tato síť byla řešena algoritmem Levenberg-Marquardt.

Následně byla otestována struktura MLP s jednou skrytou vrstvou, jako výstupní aktivační funkce byla zvolena lineární. Postupně byly ověřovány vlivy počtu neuronů ve skrytých vrstvách a vliv aktivační funkce. Vzhledem k tomu, že optimalizační algoritmus byl Adam, pracoval jsem pouze s jednou rychlostí učení. Nejlepších výsledků z hlediska trénovacích dat bylo dosaženo neuronovou sítí s aktivační funkcí sigmoid a 10 skrytými neurony, bohužel se zde objevil lehký overfitting a testovací data tato síť už tak dobře nepředpovídala. S rostoucím počtem neuronů ve skryté vrstvě se zlepšovala procentuální chyba, avšak stoupala suma čtvercové chyby. Z hlediska testovacích dat nejlépe ve skryté vrstvě fungovala aktivační funkce hyperbolického tangensu, zde bylo dosaženo sumy čtvercové chyby 0.0713 při 5 skrytých neuronech. Zvyšování počtu neuronů zvyšovalo chybu.

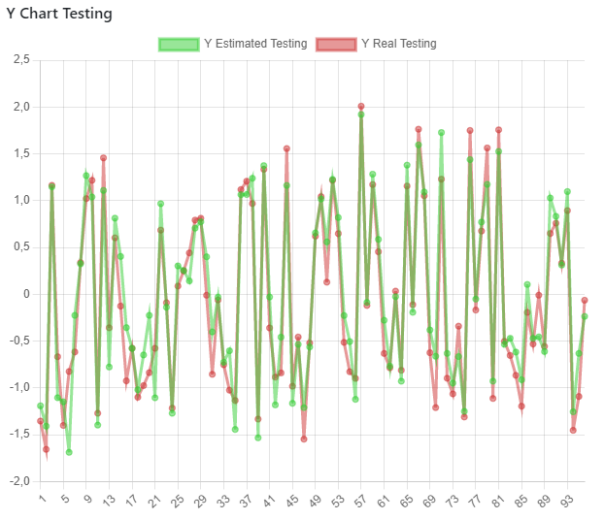
Při přidání další skryté vrstvy do sítě MLP s 20 skrytými neurony v první vrstvě a 20 skrytými neurony ve vrstvě druhé s aktivační funkcí sigmoida docházelo k nejlepším výsledkům. Suma čtvercové chyby u validačních dat se pohybovala pod hodnotou 0.062. Změnou aktivační funkce na hyperbolický tangens nebo na linearní jsem dosáhl lepších výsledků, než u MLP s jednou skrytou vrstvou. Při snížení počtu neuronů docházelo ke zmenšení chyby tréninkových dat, avšak u dat validačních došlo k nárůstu chyby.

Trénováním sítě MLP se 3 skrytými vrstvami se mi nepodařilo dosáhnout lepších výsledků než u MLP se dvěmi skrytými vrstvami.

Sítí RVFL jsem se dokázal přiblížit k výsledkům MLP se 2 skrytými vrstvami. Nejlepších výsledků jsem dosáhl se 30 neurony ve skryté vrstvě. Suma čtvercových chyb byla obdobná, avšak došlo zde k lehkému nárůstu chyby u dat validačních.

Neuronová struktura ELM dosahovala největších chyb. Rekurentní síť LSTM dosahovala obdobných výsledků jako struktura QNU, CNU. Při zvyšování počtu buněk klesala suma čtvercové chyby a při větším počtu dosahovala výsledků MLP se dvěmi skrytými vrstvami, avšak oproti této síti trvalo učení násobně déle.

Všechny struktury byly otestovány na 200 epochách. U všech sítí kromě HONU byl využit optimalizační algoritmus Adam s velikostí dávky 32.



Obr. 27: Neuronová síť RVFL - predikce hodinového výstupu elektrické energie

Jako optimální síť pro tuto predikci byla tedy s ohledem na validační data vybrána síť MLP se 2 skrytými vrstvami s počtem neuronů 20 v každé vrstvě s aktivační funkcí hyperbolického tangensu. Ve výstupní vrstvě se pak nacházela aktivační funkce lineární. Predikování hodinového výstupu elektrické energie není úplně přesné, jelikož se zde objevují i další vlivy, které mají na tento výstup vliv.

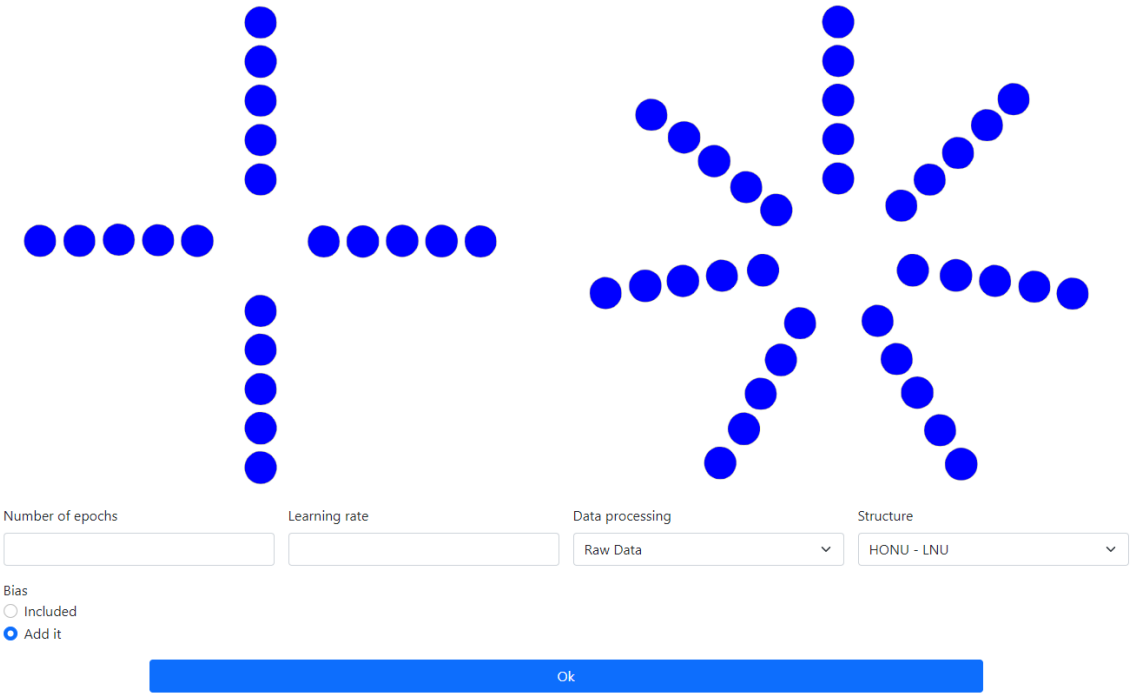
6.2.3 Teplotní pole v turbomotoru

Na hledání optimální neuronové sítě pro tato data bylo vytvořeno speciální rozhraní. Na výběr teplotních senzorů jsem vytvořil obrázek ve formátu SVG, což je škálovatelná vektorová grafika, která byla naprogramována do html. Rozhraní tedy umožňuje výběr vstupních senzorů, výběr predikovaného výstupu a výběr struktury sítě, kde při výběru dané sítě jsou pomocí JavaScriptu zobrazeny parametry vybrané sítě.

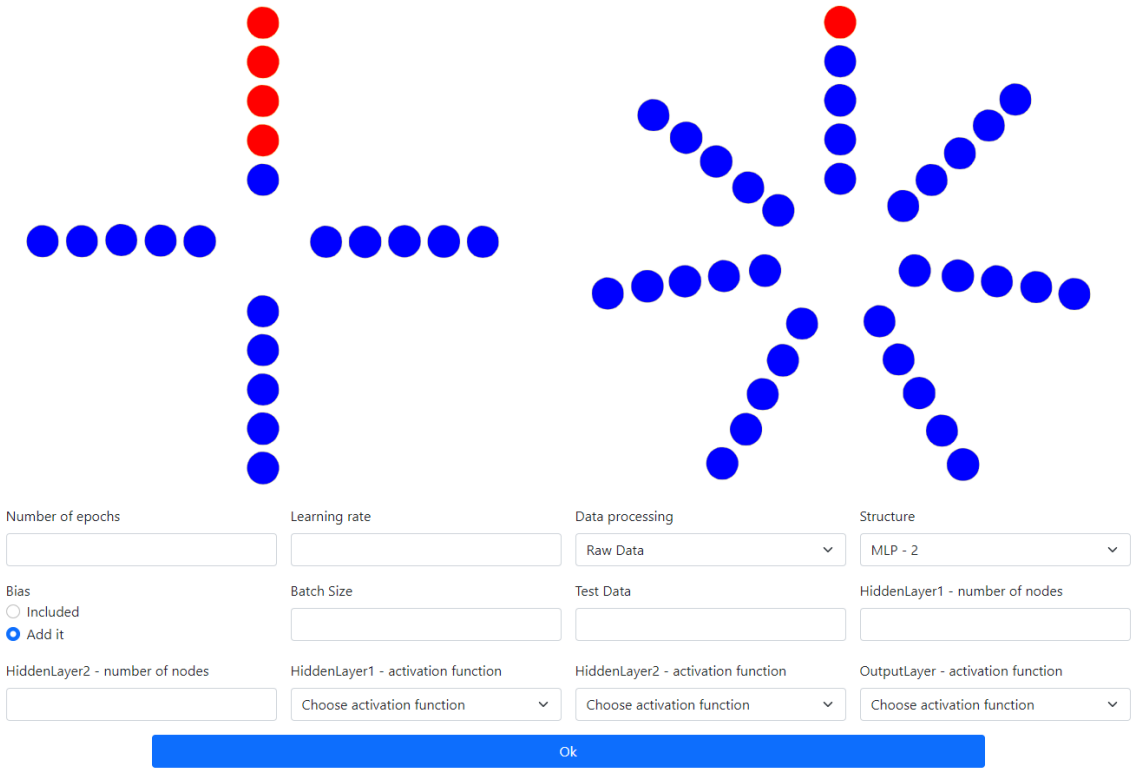
```
107   <div class="mapdiv col-md-6 position-relative" id="sensors2">
108     <svg version="1.0" xmlns="http://www.w3.org/2000/svg"
109   width="923.000000pt" height="844.000000pt" viewBox="0 0 923.000000 844.000000"
110   preserveAspectRatio="xMidYMid meet">
111
112   <g transform="translate(0.000000,844.000000) scale(0.050000,-0.050000)"
113   fill="#000000" stroke="none">
114   <path id = "1" class="output" d="M4499 8377 c-93 -35 -146 -87 -180 -177 -24 -64 -24 -125 1 -192 26
115   -70 66 -117 131 -153 46 -26 65 -30 129 -30 64 0 83 4 129 30 65 36 105 83
116   131 153 66 176 -56 366 -244 378 -36 3 -75 -1 -97 -9z"/>
117   <path id = "2" class="output" d="M4510 7701 c-86 -28 -141 -75 -181 -151 -35 -66 -33 -176 3 -245 53
118   -101 134 -150 248 -150 114 0 195 49 248 150 22 41 26 63 26 125 0 116 -50
119   198 -154 250 -48 24 -147 35 -190 21z"/>
```

Obr. 28: Zdrojový kód: Ukázka naprogramovaných dvou senzorů

Jednotlivé senzory mají své vlastní definované id, které se po kliknutí zbarví na červenou barvu a jsou uloženy do listu, který je po POST requestu odeslán se server. Na straně backendu pak běží funkce, která z tabulky importuje data pomocí definovaných id. Na následujících obrázcích je pak zobrazen náhled uživatelského rozhraní.



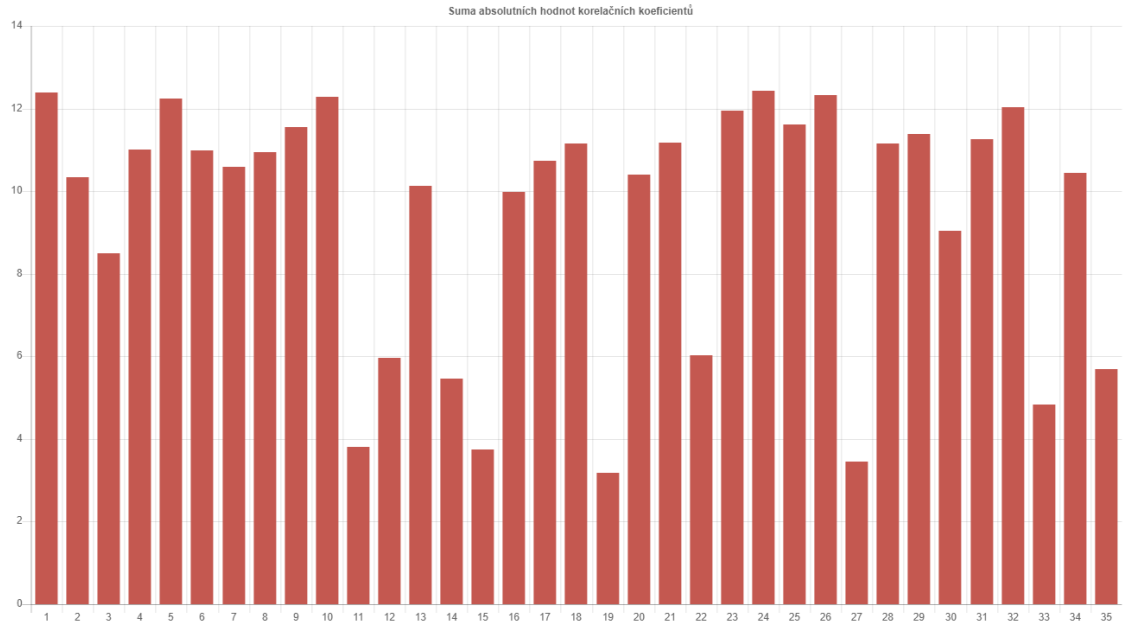
Obr. 29: Uživatelské rozhraní pro teplotní pole v turbomotoru



Obr. 30: Uživatelské rozhraní pro teplotní pole v turbomotoru po navolení senzorů a struktury sítě

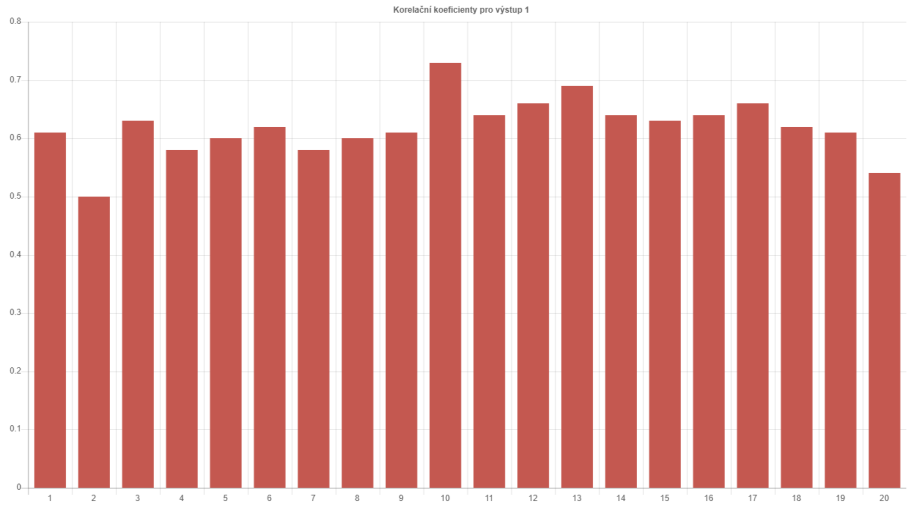
V tomto datasetu se tedy budu snažit najít ideální vstupní senzory a strukturu sítě k predikci zvoleného výstupu. Jelikož jsou měření zaznamenávána s frekvencí 100 Hz, lze předpokládat časově zpožděný děj. Vstupní data i výstupní data se pak s takto vysokou frekvencí měření téměř nemění, bude tedy nutné i převzorkování, což do jisté míry pomůže i s velikostí šumu. K tomu bude použita konvoluční vrstva.

Na data byla aplikována konvoluční vrstva, kdy došlo k převzorkování ze 100 Hz na 10 Hz. Nejprve byla použita konvoluční vrstva, kde 10 vzorků bylo průměrováno do vzorku jediného, na těchto datech jsem vyzkoušel korelaci vstupů a výstupů. Došlo i k naprogramování vzájemných posunů časových řad, kdy největší korelace byla při časovém posunu 0. Dále jsem se pokusil aplikovat konvoluční vrstvu, kde jsem 10 vzorků nahradil maximální hodnotou vzorku jediného, tímto způsobem došlo ke zhoršení korelace, což jsem předpokládal. Na následujícím grafu zobrazuji sumu korelačních koeficientů všech vstupů na jednotlivých výstupech, kde byla aplikována konvoluční vrstva, kdy 10 vzorků z měření bylo průměrováno do vzorku jediného.



Obr. 31: Suma korelačních koeficientů všech vstupů v závislosti na jednotlivých výstupech

Z grafu lze vyčíst, že suma maximálních korelačních koeficientů se pohybovala lehce přes 12, u některých výstupů pak nedosahovala ani velikosti 4. Tyto sumy mohou napovědět', jak složité bude daný výstup predikovat, avšak toto tvrzení nemusí být pravdivé, protože pokud alespoň jeden vstupní senzor dosahuje vysokého korelačního koeficientu, nemusíme ostatní senzory potřebovat. Na následujícím grafu jsou proto zobrazeny jednotlivé korelační koeficienty vstupů pro první výstup.



Obr. 32: Korelační koeficienty jednotlivých vstupů na prvním výstupu

V dalším kroku se pokusím o predikování hodnot pro první výstup.

Neuronová síť LNU

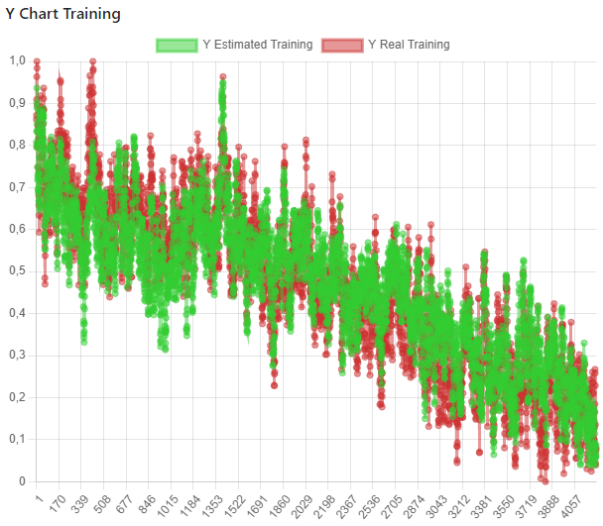
Nejprve jsem se pokusil o naučení datasetu na nejjednodušší strukturu sítě. Učení sítě jsem vyzkoušel na datech bez úpravy, datech po normalizaci i standardizaci. Vždy jsem dosáhl podobného výsledku. Vzhledem k tomu, že u ostatních neuronových struktur budu pracovat s normalizací dat, rozhodl jsem se chybu sítě zaznamenávat pomocí MSE.

Koeficient učení	Data	Počet senzorů	MSE	MSE val.
0.1	Normalizovaná	20	0.008	0.013
0.1	Normalizovaná	10	0.032	0.079

Tab. 4: Průměrná čtvercová chyba v závislosti na parametrech sítě LNU

Z tabulky lze vyčíst, že neuronová síť při vstupu všech vstupních dat se dokázala naučit s průměrnou čtvercovou chybou 0.008, což je u normalizovaných dat vcelku vysoké číslo. Při validaci dat došlo k nárůstu chyby na 0.013. Abych přiblížil velikost chyby, tak normalizovaná data se pohybují v intervalu od 0 do 1. Jedná se tedy průměrně o 9 % chybu u dat tréninkových a 11 % chybu u dat validačních (přepočteno z MSE).

Po zredukování počtu vstupů na 10 už neuronová síť ani nekopírovala trend poklesu výstupních hodnot.



Obr. 33: Predikce výstupu termočlánku 1 v 1. teplotním poli pomocí LNU

Neuronová síť QNU

Trénování této sítě jsem vyzkoušel opět s různými úpravami dat, opět vycházely velmi podobně. V následující tabulce jsou zobrazeny parametry vyzkoušených QNU sítí.

Koeficient učení	Data	Počet senzorů	MSE	MSE val.
0.1	Normalizovaná	20	0.008	0.012
0.1	Normalizovaná	10	0.015	0.063

Tab. 5: Průměrná čtvercová chyba v závislosti na parametrech sítě QNU

Aplikací této sítě jsem dostal o něco lepší výsledky. U trénování i validace došlo ke zlepšení velikosti čtvercové chyby u zmenšeného počtu vstupních veličin. Průběh predikované hodnoty vypadá téměř stejně jako u sítě LNU.

Neuronová síť CNU

Dále byla otestována struktura sítě CNU. Průběhy grafů i velikosti chyb byly téměř identické jako u QNU. Obrovské rozdíly byly pak ve výpočetním času.

Koeficient učení	Data	Počet senzorů	MSE	MSE val.
0.1	Normalizovaná	20	0.008	0.012
0.1	Normalizovaná	10	0.021	0.042

Tab. 6: Průměrná čtvercová chyba v závislosti na parametrech sítě CNU

Při redukci počtu vstupů na polovinu došlo ke zlepšení predikci validačních dat oproti QNU, avšak chyba byla stále obrovská.

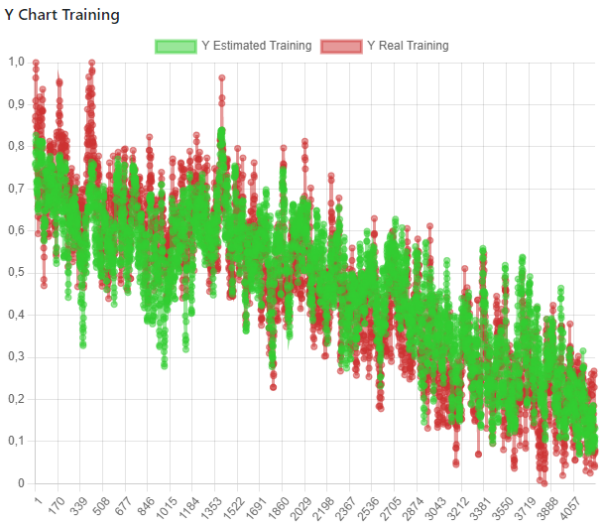
Neuronová síť MLP s jednou skrytou vrstvou

Všechny neuronové sítě MLP byly trénovány pomocí optimalizačního algoritmu Adam po dobu 200 epoch s velikostí dávky 32. Všechna data byla normalizovaná.

Počet senzorů	Počet skytých neuronů	Aktivační funkce	MSE	MSE val.
20	40	tanh	0.01	0.0171
20	40	sigm	0.01	0.0127
20	80	tanh	0.011	0.0156
20	80	sigm	0.01	0.0116

Tab. 7: Průměrná čtvercová chyba v závislosti na parametrech sítě MLP-1

Z tabulkových hodnot si lze povšimnout, že oproti sítím typu HONU došlo k lehkému zlepšení průměrné čtvercové chyby u validačních hodnot při nastavení správných parametrů. Testování proběhlo i se změnou velikosti dávky na polovinu, ovšem výsledky byly velmi podobné. Tréninková data pak vyšly hůře než u sítě typu HONU. Při redukci počtu senzorů došlo pak k lehkému zlepšení oproti sítím typu HONU, avšak jen tím, že byl lépe kopírován trend poklesu. Zvětšením velikosti konvoluční vrstvy, konkrétně průměrováním 20 hodnot jsem dosáhl horších výsledků. Zmenšením velikosti konvoluční vrstvy, konkrétně na průměrování 5 hodnot došlo k lehkému zlepšení výsledků u trénovací sady dat, u hodnot validačních pak průměrná čtvercová chyba zůstala stejná.



Obr. 34: Predikce výstupu termočlánku 1 v 1. teplotním poli pomocí MLP (20, 80, sigm)

Neuronová síť MLP se dvěmi skrytými vrstvami

Další testovanou strukturou byla MLP se dvěmi skrytými vrstvami s následujícími parametry.

Počet senzorů	Počet skytých neuronů	Aktivační funkce	MSE	MSE val.
20	40, 40	tanh, tanh	0.0097	0.0084
20	40, 40	sigm, sigm	0.0097	0.0116
20	80, 80	tanh, tanh	0.0095	0.009
20	80, 80	sigm, sigm	0.0096	0.0142

Tab. 8: Průměrná čtvercová chyba v závislosti na parametrech sítě MLP-1

Trénování MLP sítě se dvěmi skrytými vrstami dosahujeme lepších výsledků s použitím aktivační funkce hyperbolického tangensu. Testovány byly i ostatní aktivační funkce, ale lepší výsledky nepřinesly. Dvě skryté vrstvy předpovídání termočlánku lehce pomohly.

Neuronová síť MLP se třemi skrytými vrstvami

Sítí MLP se třemi skrytými vrstvami jsem lepších výsledků nedosáhl.

Neuronová síť RVFL

Další testovanou strukturou na těchto datech byla síť RVFL. V následující tabulce lze vidět parametry této sítě. Pro všechna nastavení v následující tabulce byl použit učící parametr 0.1, velikost dávky 32, optimalizační algoritmus Adam a počet epoch 200.

Počet senzorů	Počet skytých neuronů	Aktivační funkce	MSE	MSE val.
20	100	tanh	0.0115	0.0126
20	100	sigm	0.0099	0.0114
20	200	tanh	0.0121	0.0118
20	200	sigm	0.0103	0.0180

Tab. 9: Průměrná čtvercová chyba v závislosti na parametrech sítě RVFL

Vyšším počtem skrytých neuronů se výsledek postupně zhoršoval, zejména pak u aktivační funkce sigmoida, kde chyba rostla hlavně u validačních dat. Ve srovnání se sítí LNU, která neobsahuje skrytou vrstvu, dokázala predikovat validační data lépe.

Neuronová síť ELM

U této sítě je nutnost definovat co největší počet neuronů ve skryté vrstvě. To se pak projevuje zejména na výpočetním čase. U této sítě se mi nepodařilo odladit parametry tak, aby síť vypočítala publikovatelné výsledky. Dosahoval jsem 3 - 4 násobné chyby než u sítí zmiňovaných v předchozích paragrafech.

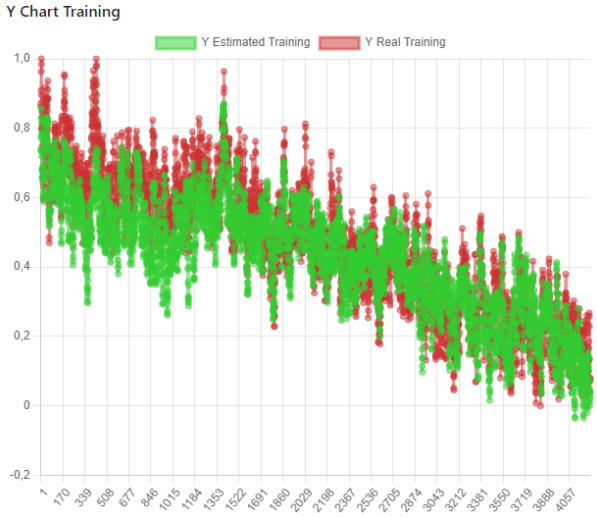
Neuronová síť LSTM

Poslední testovanou sítí byla struktura LSTM. U této sítě jsem očekával lepší výsledky, než byly získané. Síť byla otestována těmito parametry.

Počet senzorů	Počet LSTM buněk	MSE	MSE val.
20	5	0.0105	0.0140
20	10	0.0105	0.0151
20	15	0.0100	0.0195

Tab. 10: Průměrná čtvercová chyba v závislosti na parametrech sítě LSTM

U této sítě jsem se rozhodl postupně zvyšovat počet LSTM buněk. Jak počet buněk rostl, rostla složitost sítě a začalo docházet k zvětšení velikosti průměrné čtvercové chyby u validačních dat.



Obr. 35: Predikce výstupu termočlánku 1 v 1. teplotním poli pomocí LSTM

Nejlépší síť i s ohledem na validační data byla typu MLP se dvěmi skrytými vrstvami s aktivační funkcí hyperbolického tangensu s počtem neuronů 80 v každé skryté vrstvě.

7 Závěr

Cílem této diplomové práce bylo naprogramování systému pro automaticky optimalizovanou predikci časových řad. V rešeršní části byla shrnuta historie neuronových sítí a vysvětlen princip fungování jednoduchého perceptronu. Tato problematika byla rozšířena o představení aktivačních funkcí neuronových sítí, na které bylo navázáno shrnutím struktur sítě dopředných a rekurentních neuronových sítí. Postupně byly představeny principy jednotlivých optimalizačních algoritmů a to jak metod gradientních, tak i metod negradientních. V této kapitole byl vysvětlen i princip zpětného šíření chyby, který se používá u struktur sítě MLP.

V následující kapitole bylo představeno vývojové prostředí Django, které bylo použito pro naprogramování systému. Zde byly vysvětleny důvody volby a představena základní

struktura projektu. Stručně je ukázána i syntaxe daného projektu.

Dále byla popsáno naprogramování samotného systému. Nejprve byly představeny jednotlivé frontendy, které tvoří uživatelské rozhraní - client. Programovací jazyk pro tvorbu těchto skriptů byl HTML, CSS a JavaScript. Tvorba kódu byla usnadněna importováním knihoven Bootstrap a Chart.js. Dále byly představeny vybrané funkce, které běží na straně serveru a jsou psané v Pythonu. Pro tvorbu neuronových modelů byla použita knihovna Keras.

Rozhraní umožňuje výběr optimalizačního algoritmu, navolení počtu epoch, stanovení učícího koeficientu, výběr vstupních i výstupních dat. Představená data jsou uložena na serveru a pomocí přepínačů je lze navolit, nebo lze nahrát své vlastní data. Tato data systém zpracovává podle požadovaného výběru, a to buď metodou normalizace, standardizace, PCA nebo konvoluční vrstvou, kde je nutné navolení i dalších parametrů. Vzhledem k vyšší časové náročnosti výpočtu neuronových sítí jsem rozhodnutí o zpracování dat ponechal na uživateli, než aby vhodné parametry byly vypočítávány, avšak struktura tohoto projektu je velmi přehledná díky refaktorizaci kódu, a to i přes vysoký počet řádků kódu, a rozšíření o automatický výpočet parametrů sítě už není takový problém, avšak časová optimalizace by se pohybovala v řádech hodin (dle počtu dat). U stochastického algoritmu lze pak navolit i velikost jednotlivé dávky. U neuronových sítí se skrytou vrstvou lze pak navolit typ aktivační funkce a počet neuronů ve skryté vrstvě. U rekurentní architektury lze poté navolit počet LSTM buněk. Tato data zadaná uživatelem jsou poté zpracována serverem, který optimalizuje síť navrženou uživatelem. Po výpočtu se pak zobrazí rozhraní s výsledky, kde lze vidět, jak je neuronová síť schopna předpovídat tréninková a validační data. V pravé části tohoto rozhraní je pak zobrazen vývoj průměrné čtvercové chyby na jednotlivých epochách a vývoj jednotlivých vah v závislosti na epochách.

V následující části byla stručně představena data, která byla použita pro testování rozhraní. Následně bylo rozhraní na těchto datech úspěšně otestováno. Nejprve byl systém podroben zkoušce pomocí umělých dat, kde byl testován, jestli se zvládne vypořádat s přidaným šumem. Systém zvládl predikci budícího proudu u synchronního motoru, nejlépe podle struktury sítě LSTM. Dále byl tento systém otestován na predikci hodinového výstupu elektrické energie u elektrárny s kombinovaným cyklem, zde se jako nejlepší zvolená síť osvědčila síť MLP se dvěmi skrytými vrstvami. Posledním testovaným datasetem byla predikce termočlánku v teplotním poli pomocí vstupních teplot v teplotním poli druhém. Zde bylo nutné data zpracovat pomocí konvoluční vrstvy. Nejlepší výsledek s ohledem na validační data zvládla síť MLP se dvěmi skrytými vrstvami. Pro tento dataset bylo vytvořeno rozhraní, které pomocí vektorové grafiky umožňovalo navolení jednotlivých senzorů, které vstupovaly a vystupovaly ze sítě.

V závěru této práce bych chtěl zhodnotit jednotlivé struktury sítí. Síť typu LNU funguje z daných sítí nejrychleji, avšak je schopna predikovat pouze lineární vztahy. Odolnost proti šumu této sítě vycházela nejhůře. Síť typu QNU bere v úvahu již nelinearitu. Pomocí této sítě jsem byl schopen predikce časově zpožděného děje exponenciální charakteristiky, predikci elektrické energie i předpověď teplot v turbomotoru, kde bylo potřeba přidání konvoluční vrstvy - převzorkování z 100 Hz na 10 Hz. Síť CNU fungovala vždy stejně nebo lépe než ostatní HONU. Také byla odolnější vůči šumu. Došlo zde ale k velkému časovému nárůstu trénování sítě. Sítě typu HONU, zejména pak QNU a CNU, se stávají populární pro mnoho technických aplikací, protože jsou schopné vystihnout nelineární chování a učení této sítě má stále konvexní povahu. Další výhodou pak je, že tyto sítě jsou lineární v parametrech, dá se pak dosáhnout stejných výsledků i z odlišných počátečních vah. Všechny sítě tohoto typu byly trénovány optimalizačním algoritmem Levenberg-Marquardt.

Síť MLP s jednou až třemi skrytými vrstvami je o něco složitější správně nastavit, jelikož je zde mnoho parametrů. Nejlepších výsledků jsem dosahoval u MLP se dvěmi skrytými vrstvami, kde tato síť dosahovala u predikce elektrické energie a teplot v turbomotoru sice horší chybu během tréninku, avšak u testovacích datech jsem dosáhl někdy i lepších výsledků než u HONU. Tato síť se neosvědčila v predikci umělých dat, kdy měla předpovídat exponenciální charakteristiku. Reálná data do této sítě vstupovaly po normalizaci, u turbomotoru byla využita i konvoluční vrstva. Nejlepších výsledků jsem dosahoval s aktivačními funkcemi sigmoid a hyperbolický tangens, optimalizačním algoritmem byl Adam.

Síť RVFL dosahovala při správném nastavení lepších výsledků než LNU. Tato síť kombinuje vlastnosti LNU a MLP. Ovšem výsledky nebyly reprodukovatelné z důvodu počáteční inicializace vah, které se během tréninku neměnní. Solidních výsledků jsem dosáhl u reálných dat, zejména u predikce teplot v turbomotoru, kde byla použita konvoluční vrstva, a predikce elektrické energie. Nejlepších výsledků jsem dosahoval s aktivačními funkcemi sigmoid a hyperbolický tangens s optimalizačním algoritmem Adam.

Síť ELM byla testována jak na umělých, tak i reálných datech. Bylo provedeno mnoho změn, ať už v nastavení parametrů sítě, ve zpracování dat, ale nikdy se mi nepodařilo dosáhnout lepších výsledků než u sítí, které byly zmíněné. Tato síť je ze všech nejvíce závislá na počátečních vahách, a pokud nejsou inicializovány správně, síť není schopna se naučit.

Nejvíc komplexní sítí je síť typu LSTM, jelikož jsem vždy pomocí této sítě dosáhl slušných výsledků. Tato síť společně s CNU dokázala nejlépe predikovat umělá data a byly nejvíce odolné vůči šumu. Ze všech testovaných struktur dosáhla nejlepší předpovědi u synchronního motoru, kde predikovala budící proud. Do této sítě vstupovala normalizovaná data, kde byl počet LSTM buněk nastaven na 20. Úspěch této sítě u této aplikace je dán zejména tím, že se jedná o periodický děj. U predikce hodinového výstupu elektrické energie fungovala obdobně jako QNU a CNU. Predikci teplotních polí v turbomotoru zvládla tato

síť vcelku slušně, avšak u testovacích datech se objevil nárůst chyby. Vstupním vektorem byla převzorkovaná data pomocí konvoluční vrstvy. Optimalizačním algoritmem byl Adam, počet LSTM buněk se lišil v závislosti na aplikaci.

Seznam použitých značek a symbolů

1. HONU - Higher Order Neural Unit
2. MLP - Multilayer Perceptron
3. ELM - Extreme Learning Machine
4. RVFL - Random Vector Functional Link
5. LSTM - Long Short Term Memory
6. ReLU - Rectified Linear Activation
7. LNU - Linear neural unit
8. QNU - Quadratic neural unit
9. CNU - Cubic neural unit
10. RNN - Recurrent neural network
11. MSE - Mean Squared Error
12. MAE - Mean Absolute Error
13. AdaGrad - Adaptive Gradient
14. Adam - Adaptive moment estimation
15. HTML - HyperText Markup Language
16. csv - Comma-separated values
17. png - Portable Network Graphics
18. jpg - Joint Photographic Group
19. CSS - Cascading Style Sheets
20. HTTP - Hypertext Transfer Protocol
21. API - Application Programming Interface
22. PCA - Principal component analysis
23. Hz - Hertz
24. svg - Scalable Vector Graphic

Seznam použité literatury a zdrojů

[1] GRAUPE, Daniel. *PRINCIPLES OF ARTIFICIAL NEURAL NETWORKS*. 2nd Edition. USA: World Scientific Publishing Co. Pte., 2007. ISBN 981-270-624-0.

[2] CHANDRA, Akshay L. *McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron*. Towards Data Science [online]. 24.7.2018 [cit. 2022-06-24]. Dostupné z: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[3] PILLAR, Anna *The Perceptron*. Turning Magazine [online]. 26.12.2020 [cit. 2022-06-24]. Dostupné z: https://www.turningmagazine.com/blog/2020/12/26/the-perceptron/

[4] ŠÍMA, Jiří a Roman NERUDA. *Teoretické otázky neuronových sítí*. Praha: MATFYZPRESS, 1996. ISBN 80-85863-18-9.

[5] VANDERGRIENDT, Carly a Rachel ZIMLICH *An Easy Guide to Neuron Anatomy with Diagrams*. Healthline [online]. 26.12.2020 [cit. 2022-06-24]. Dostupné z: https://www.healthline.com/health/neurons

[6] BAHETI, Pragati *Activation Functions in Neural Networks*. V7Labs [online]. 21.6.2022 [cit. 2022-06-24]. Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions#: :text=The% 20linear%20activation %20function%2C%20also,the%20value%20it%20was%20given.

[7] DORSAF, Sebai *Comprehensive synthesis of the main activation functions pros and cons*. Analytics Vidhya [online]. 2.5.2020 [cit. 2022-06-24]. Dostupné z: https://medium.com/analytics-vidhya/comprehensive-synthesis-of-the-main-activation -functions-pros-and-cons-dab105fe4b3b

[8] BROWNLEE, Jason *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Machine Learning Mastery [online]. 9.1.2019 [cit. 2022-06-24]. Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep -learning-neural-networks/#: :text=Key%20among%20the%20limitations%20of,as%20 a%20%E2%80%9Cdying%20ReLU%E2%80%9C.

[9] MASS, L. Andrew, HANNUN, Awni Y., NG, Andrew Y. *Rectifier Nonlinearities Improve Neural Network Acoustic Models*[online]. USA, 2013 [cit. 2022-06-24]. Dostupné z: http://ai.stanford.edu/ amaas/papers/relu_hybrid_icml2013_final.pdf. Stanford University

[10] ZHOU, Tianxiang. JIANG, Zhaobing. LIU, Xujian. TAN, Kun *Research on the long-term and short-term forecasts of navigable river's water-level fluctuation based on the adaptive multilayer*

perceptron[online]. USA, China 2020 [cit. 2022-06-24]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0888327021010682#f0025. University of California, Sanjiang University, Nanjing Tech University

[11] ZHANG, Yudong. WANG, Shuihua. JI, Genlin. PHILLIPS, Preetha *Fruit classification using computer vision and feedforward neural network*[online]. USA, China 2014 [cit. 2022-06-24]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S026087741400291X. Shepherd University, Nanjing Normal University

[12] HUANG, Guang-Bin. ZHU, Qin-Yu, SIEW, Chee-Kheong *Extreme learning machine: Theory and applications*[online]. Singapore, 2006 [cit. 2022-06-24]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0925231206000385. School of Electrical and Electronic Engineering, NanyangTechnological University

[13] WANG, Jian. LU, Siyuan, WANG, Shui-Hua, ZHANG, Yu-Dong *A review on extreme learning machine*[online]. Multimed Tools Appl, 2021 [cit. 2022-06-24]. Dostupné z: https://link.springer.com/content/pdf/10.1007/s11042-021-11007-7.pdf

[14] SHI, Qiushi. KATUWAL, Rakesh. SUGANTHAN, P.N.. TANVEER, M. *Random vector functional link neural network based ensemble deep learning*[online]. Singapore, India 2020 [cit. 2022-06-24]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0031320321001655. Nanyang Technological University, Indian Institute of Technology Indore

[15] GRAVES, Alex. SCHMIDHUBER, Jürgen *Framewise phoneme classification with bidirectional LSTM and other neural network architectures*[online]. Switzerland, Germany, 2005 [cit. 2022-06-24]. Dostupné z: https://reader.elsevier.com/reader/sd/pii/S0893608005001206?token=DCAECD7DE592D88D01CB124CAFDEE8BFA9F25369BD674B247277F3AF7183E7157FE1BFE5D5C497FF0626105FC3ADBBBF&originRegion=eu-west-1&originCreation=20220704125533. IDSIA, TU Munich

[16] HEWAMALAGE, Hansika. BERGMEIR, Christoph. BANDARA, Kasun *Recurrent Neural Networks for Time Series Forecasting: Current status and future direction*[online]. Australia, 2021 [cit. 2022-06-24]. Dostupné z: https://reader.elsevier.com/reader/sd/pii/S0169207020300996?token=68EEAE1F3AC0B42BE122E02AA309FB9F45A20C576A9A29C236A4B99918FC8B4411AA11D250F89C8A8D35A149B7AF95C0&originRegion=eu-west-1&originCreation=20220704130525. Monash University

[17] VOS, Kilian. PENG, Zhongxiao. JENKINS, Christopher *Vibration-based anomaly detection using LSTM/SVM approaches*[online]. Australia, 2022 [cit. 2022-06-24]. Dostupné z:

https://www.sciencedirect.com/science/article/pii/S0888327021010682#f0025. UNSW Sydney

[18] MUNGALPARA, Jaimin *What is LSTM , peephole LSTM and GRU?*. Nerd For Tech [online]. 5.2.2021 [cit. 2022-06-24]. Dostupné z: https://medium.com/nerd-for-tech/what-is-lstm-peephole-lstm-and-gru-77470d84954b

[19] DOSHI, Sanket *Various Optimization Algorithms For Training Neural Network*. Towards Data Science [online]. 13.1.2019 [cit. 2022-08-2]. Dostupné z: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[20] BENTO, Carolina *Stochastic Gradient Descent explained in real life*. Towards Data Science [online]. 2.6.2021 [cit. 2022-08-02]. Dostupné z: https://towardsdatascience.com/stochastic-gradient-descent-explained-in-real-life-predicting-your-pizzas-cooking-time-b7639d5e6a32#: :text=Compared%20to%20Gradient%20Descent%2C%20Stochastic,updates%20have%20a%20higher%20variance.

[21] BROWNLEE, Jason *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*. Deep Learning [online]. 21.7.2017 [cit. 2022-08-02]. Dostupné z:https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[22] SUTSKEVER, Ilya. MARTENS, James, DAHL, George, HINTON, Geoffrey *On the importance of initialization and momentum in deep learning*[online]. Canada, 2013 [cit. 2022-08-02]. Dostupné z: http://proceedings.mlr.press/v28/sutskever13.pdf.University of Toronto

[23] RUDER, Sebastian *An overview of gradient descent optimization algorithms*. Ruder [online]. 19.1.2016 [cit. 2022-08-02]. Dostupné z: https://ruder.io/optimizing-gradient-descent/index.html#adagrad/.

[24] BROWNLEE, Jason *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Deep Learning Performance [online]. 3.7.2017 [cit. 2022-08-02]. Dostupné z: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[25] QUESADA, Alberto *5 algorithms to train a neural network*. Artelnics [online] [cit. 2022-08-02]. Dostupné z: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network# Levenberg-Marquardt

[26] MCGONAGLE, John, SHALKOUSKL, George and others *Backpropagation*. Brilliant [online]. [cit. 2022-08-02]. Dostupné z: https://brilliant.org/wiki/backpropagation/

[27] NIELSEN, Michael *How the backpropagation algorithm works*.

Deep Learning [online]. 12.2019 [cit. 2022-08-02]. Dostupné z: http://neuralnetworksanddeeplearning.com/chap2.html

[28] 3Blue1Brown *Backpropagation calculus | Chapter 4, Deep learning*. YouTube [online]. 3.11.2017 [cit. 2022-08-02]. Dostupné z: https://www.youtube.com/watch?v=tIeHLnjs5U8

[29] PATEL, Jeel *List of 10 Best Web Frameworks for Web App Development in 2022*. Monocubed [online]. 9.11.2021 [cit. 2022-08-02]. Dostupné z: https://www.monocubed.com/blog/most-popular-web-frameworks/

[30] DataFlair *Django Advantages and Disadvantages – Why You Should Choose Django?*. DataFlair [online]. [cit. 2022-08-02]. Dostupné z: https://data-flair.training/blogs/django-advantages-and-disadvantages/

[31] BAYINDIR, Ramazan. KAHRAMAN, Hamdi Tolga *Synchronous Machine Data Set Data Set*. Machine Learning Repository [online]. [cit. 2022-08-02]. Dostupné z: https://archive.ics.uci.edu/ml/datasets/Synchronous+Machine+Data+Set

[32] TUFEKCI, Pinar. KAYA, Heysem *Combined Cycle Power Plant Data Set*. Machine Learning Repository [online]. [cit. 2022-08-02]. Dostupné z: https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant

[33] BUKOVSKY, Ivo. *Deterministic behavior of temperature field in turboprop engine via shallow neural networks.* Neural Computing and Applications [online]. 2021, 33(19), 13145–13161. ISSN 0941-0643, 1433-3058. Dostupné z: doi:10.1007/s00521-021-06013-7

Seznam obrázků a tabulek

Seznam obrázků

Seznam tabulek

Seznam použitého SW

– Texmaker, MiKTeX (LaTeX), Python, HTML, CSS, JavaScript, Django, Visual Studio Code, Microsoft Excel, Microsoft PowerPoint

Seznam příloh

Příloha 1: Vypracované skripty ve formátu .py (CD)

Příloha 2: Vypracované skripty ve formátu .html (CD)

Příloha 3: Data pro učení neuronových sítí ve formátu .npy, .csv (CD)