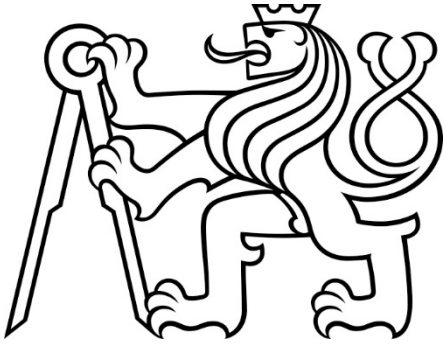
**ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE**

**FAKULTA STROJNÍ**



# DIPLOMOVÁ PRÁCE

**2019**

**RADEK HOFÍREK**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hofírek** Jméno: **Radek** Osobní číslo: **438959**

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:

**Algoritmy výpočetní inteligence pro rekonstrukci teplotních polí v turbomotoru**

Název diplomové práce anglicky:

**Computational Intelligence Algorithms for Temperature Field Reconstruction.**

Pokyny pro vypracování:

Proveďte rešerši problematiky měření teplot v řezu malých leteckých turbomotorů. Proveďte rešerši vhodných supervizorovaných algoritmů pro aproximaci a predikci (např. HONU, MLP, ELM, RVFL, LSTM,...).
Analyzujte měřená data. Navrhněte metodu rekonstrukce teplotního pole v řezu a odhadu isotermických čar v řezu z minimáního počtu bodově měřených hodnot s využitím neuronových sítí a supervizorovaných algoritmů. Vyberte metody, implementujte na umělých datech s různou úrovní šumu i na měřených datech a vyhodnoťe přesnost a robustnost učících algoritmů.
Rozsah práce min. 50 stran přílohy.

Seznam doporučené literatury:

[1] BUKOVSKY, I. a N. HOMMA. An Approach to Stable Gradient-Descent Adaptation of Higher Order Neural Units. IEEE Transactions on Neural Networks and Learning Systems [online]. 2017, 28(9), 2022–2034. ISSN 2162-237X. Dostupné z: doi:10.1109/TNNLS.2016.2572310
[2] DAVID B FOGEL, DERONG LIU a JAMES M KELLER. Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation. Piscataway, NJ: IEEE Press, 2016. ISBN 978-1-119-21434-2.

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: **29.04.2019** Termín odevzdání diplomové práce: **16.08.2019**

Platnost zadání diplomové práce: _____

| doc. Ing. Ivo Bukovský, Ph.D.<br>podpis vedoucí(ho) práce | prof. Ing. Milan Růžička, CSc.<br>podpis vedoucí(ho) ústavu/katedry | prof. Ing. Michael Valášek, DrSc.<br>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

# Anotace

| | |
|---|---|
| Autor: | Radek Hofírek |
| Název DP: | Algoritmy výpočetní inteligence pro rekonstrukci teplotních polí v turbomotoru |
| Rozsah práce: | 68 stran |
| Školní rok vyhotovení: | 2018-2019 |
| Škola: | ČVUT v Praze – Fakulta strojní |
| Ústav: | Ú 12105 Ústav mechaniky, biomechaniky a mechatroniky |
| Vedoucí DP: | doc. Ing. Ivo Bukovský, Ph.D. |
| Zadavatel tématu: | ČVUT v Praze – Fakulta strojní |
| Datum odevzdání: | 12. Srpna 2019 |
| Klíčová slova: | Neuronové sítě, turbomotory, LSTM, MLP. strojové učení |

# Annotation

| | |
|---|---|
| Author: | Radek Hofírek |
| Title of master thesis: | Computational Intelligence Algorithms for Temperature Field Reconstruction. |
| Extent: | 68 pages |
| Školní rok vyhotovení: | 2018-2019 |
| University: | CTU in Prague – Faculty of Mechanical Engineering |
| Department: | Ú 12105 Department of Mechanics, Biomechanics and Mechatronics |
| Supervisor: | doc. Ing. Ivo Bukovský, Ph.D. |
| Submitter of Theme: | ČVUT v Praze – Fakulta strojní |
| Date of handover: | 12th of August 2019 |
| Keywords: | Neural networks, turboengines, MLP, LSTM, machine learning |

# Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a že jsem uvedl v přiloženém seznamu veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací, vydaným ČVUT v Praze 1. 7. 2009.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu S 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 13.8.2019 ……..……

podpis

Děkuji doc. Ing. Ivo Bukovskému, Ph.D. za odborné vedení, rady a velkou trpělivost při tvorbě této práce. Za všechen čas, který mi věnoval a za vše co jsem se od něj naučil. Hlavní dík však patří mé rodině, která mě celé studium i mimo něj vždy podporovala. Pak také mé partnerce a přátelům. Bez jejich morální i jiné podpory by nic z toho nebylo možné.

# Obsah

# Seznam obrázků

## Seznam tabulek



# Seznam použitých zkratek a jednotek

| | |
|---|---|
| **ML** | Machine learning |
| **UI** | Umělá inteligence |
| **MLP** | Multi-layer Perceptron |
| **HONU** | Higher order neural unit |
| **ELM** | Extreme learning machine |
| **LSTM** | Long-Short term memory |
| **PCA** | Principal Components Analysis |
| **A(NN)** | Artifical (Neural Network) |
| **BP** | BackPropragation |
| **MAE** | Mean absolute error |
| ***s / ms*** | sekunda / milisekunda |
| **°C** | stupeň Celsia |

# 1 Úvod

Díky modernizaci a digitalizaci průmyslu a jiných odvětví, během posledních několika let, strmě stoupá množství dat, která z procesů v těchto jednotlivých odvětvích získáváme. Následkem zmenšování počítačů a růstu jejich výpočetního výkonu a díky vývoji v senzorické technice a možnostech rozlišitelnosti jednotlivých vstupů, máme nyní přístup k enormnímu datovému vzorku popisující dané děje. Tato nezpracovaná data však sama o sobě povětšinou nemají velkého užitku. Přidaná hodnota vzniká při analýze originálních dat a následným aplikačním využitím. Proto dnes vědní obory jako statistika či datová analýza zažívají velký boom [1]. V době, kdy snížení nákladů či preventivní zamezení ztráty, znamenají kýžený nárůst efektivity podniku ve stále více konkurenčním prostředí, je možnost účinně získávat a následně správně analyzovat data nedocenitelná. Využití jako prediktivní údržba či návrh logistiky a optimalizace materiálových cest, vyhodnocování dat z testů a další, jsou jedny z mála věcí, které jsme schopni dnes za pomoci datové analýzy dosáhnout.

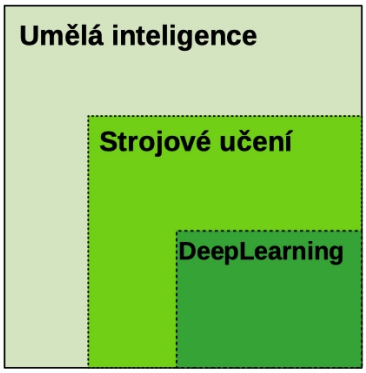
Jednou z hojně využívaných technik pro účinnou analýzu dat a informací je i strojové učení [2]. Metody strojového učení spočívající ve využití neuronových sítí a jejich schopnosti přizpůsobovat se podle vnějších stimulů (vstupních dat) a chovat se podobně jako struktura lidského mozku jsou jednou z nejvíce rozvíjejících se oblastí informační technologií.

Cílem této práce je provést rešerši dané problematiky a popsat některé z modelů neuronových sítí a jejich možnosti využití. V následujících kapitolách bude popsán princip fungování modelů **MLP** (angl. *Multi Layer Perceptron*[3]), **HONU** (angl. *Higher Order Neural Unit*[4]), **ELM** (*Extreme Learning Machine*) a **LSTM** ***(****Long Short Term Memory*[5]*).* U těchto metod budou zhodnoceny jejich výhody a nevýhody při využití s reálnými daty. Dalším cílem je vytvoření některých z těchto neuronových sítí v programovacím jazyce Python[6]. Bude také proveden, krátký průzkum možností měření teplot v jednotlivých řezech turbodmychadla. Následně budou vytvořené neuronové sítě aplikovány na data z reálného turbodmychadla [7] a využity na predikci vývoje teploty vzduchu v jednotlivých oblastech motoru v konkrétním řezech. Cílem je

zjistit zda je možné z hodnot jedné konkrétní teploty v jednom bodě dopočítat teplotu ve všech ostatních.. Porovnat výsledky jednotlivých typů sítí a zjistit jejich přesnost. Následně získané závislosti využít na predikci na jiném umělém vzorku dat, kde budeme záměrně přidávat množství šumu a otestujeme tak robustnost učících algoritmů.

# 2 Strojové učení

Strojové učení (angl. *Machine learning*) je podobor oblasti vývoje, kterou dnes nazýváme umělou inteligencí (viz. Obr. 2.1) . Jedná se o soubor metod, jenž jsou schopny automaticky vytvářet analytické modely, které pak můžeme využít pro datovou analýzu. Pomocí algoritmů, které se postupně učí a adaptují na poskytnutá data je ML schopno poskytnout náhled na vnitřní závislosti v datech bez toho, aby byl algoritmus předem instruován, kde hledat[2, 8]. Obecně základní technikou ML je prohledávání stavového prostoru. Mezi běžné rysy ML lze zařadit využívání znalostí, práce se symbolickými nebo strukturovanými proměnnými či aplikace poznatků z nestandardních logik. Nejtypičtější úlohou strojového učení je pomoc při získávání znalostí z dat, tzv. *Data mining*.




*Obr. 2.1: Ukázka zařazení ML jako podoboru umělé inteligence*

## 2.1 Vývoj ML

Historicky raný vývoj ML započal už na začátku 19. století s příchodem využití lineární algebry v oblasti statistiky. V té době byla uveřejněna práce Adrien-Marie Legendra a Carl Friedrich Gausse popisující *metodu nejmenších čtverců*, která

využívala rané formy lineární regrese[8]. Později bylo toto odvětví posunuto také Pierre-Simon Laplacem, jenž rozvinul práci Thomas Bayese[1] a definoval to čemu dnes říkáme Bayesův teorém.

S příchodem výpočetní techniky nastává velký rozvoj ML. Již v roce 1950 Alan Turing navrhuje možnost „učícího se stroje“, který by mohl být považován za umělou inteligenci[2]. Už v roce 1951 poté přišli Marvin Minsky a Dean Edmonds s prvním strojem obsahujícím neuronovou síť schopnou se učit, *SNARC*[3]. Vývoj pokračoval dál a s rostoucím výkonem počítačů rostlo i tempo objevů nových teorií i možností implementace ML. Počítače jsou díky ML schopny hrát piškvorky nebo šachy a provádět náročné výpčoty. Při implementaci senzorů je stroj se schopen pohybovat a vyhýbat překážkám. Jsou postulovány nové metody jako *Perceptron* nebo rekurentní neuronové sítě. V roce 1997 je poté počítač od společnosti IBM, pojmenovaný *Deep Blue*, schopen porazit světového šampiona v šachu Garry Kasparova.

S příchodem 21. století je ML a hlavně UI věnována veliká pozornost a jejich vývoj je součástí inovačních strategií většiny vyspělých zemí[2, 8, 9].

## 2.2 Rozdělení ML

Velice obecně rozlišujeme celkem 3 základní metody ML. Tyto 3 metody se odlišují hlavně podobou vstupů a výstupů, čímž se následně odlišuje i jejich využití. Jedná se o:

### 2.2.1 Unsupervised learning

V této práci není unsupervised learning („učení bez učitele“) využito, tudíž se pouze krátce zmiňuje o používaných aplikacích a jejich využití.

Unsupervised learning je metoda strojového učení, kdy učícímu se systému dáme pouze vstupní data a nedáme mu k dispozici požadované (preferované) výsledky. Data nejsou nijak označena ani rozdělena do kategorií. Od metody očekáváme, že sama dokáže najít závislosti v datovém vzorku a postupně se na ně adaptovat. Následně je

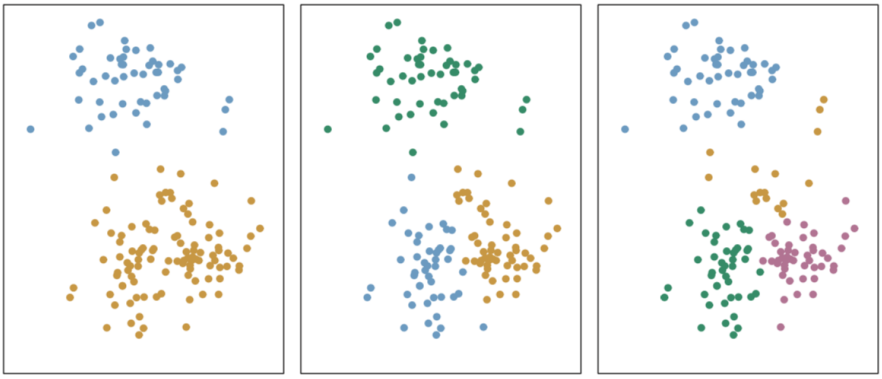
1 *Solving a Problem in the Doctrine Chances*
2 Turingův test – dnes základním kritériem UI
3 Stochastic neural analog reinforcement calculator

schopna vyhodnotit zda se tyto závislosti vyskytují i v dalších testovaných vzorcích. Jedná se o náročnější metodu, jelikož výsledky mohou být subjektivně interpretovány a neexistuje žádný univerzální způsob jejich ověření třeba křížovou validací[9]. Většina hlavních aplikací je z oblasti odhadu hustoty výskytu ze statistiky. Snahou je co nejlepší interpretace dat bez ztráty přesnosti.

### *Používané aplikace*

- **PCA** neboli *Principal Components Analysis* je aplikace, kterou využíváme pokud máme k dispozici větší množství korelovaných dat. Pomocí PCA jsme schopni udělat kompresi daných dat a vytvořit interpretaci dat, které má menší rozměr než data původní[8–10].

- **Clustering** je název skupiny aplikací, které dokáží nacházet podskupiny neboli shluky v datech. Tyto shluky se snaží tvořit tak, aby jednotlivé elementy shluku měli co největší podobnost, aniž by nás zajímal význam jednotlivých elementů. Jedna z typických metod shlukování je K-Means Clustering[8, 9]. Na Obr. 2.2 vidíme postupnou tvorbu výsledných shluků.



*Obr. 2.2: Ukázka postupného Clusteringu náhodných dat [8]*

### *Využití*

Pomocí unsupervised learning sice nedosahujeme výsledků co se týče predikce, ale zato vynikají v nalezení skrytých vzájemných vztahů mezi daty. Velké úspěchy se daří na poli výzkumu rakoviny prsu, kdy je ML aplikace schopna nalézt společné prvky u pacientů s rakovinou[8]. Dále se hodí jako příprava dat pro neuronové sítě nebo jiné metody supervised learningu[9]. Jsou sice časově a strojově náročnější, ale do budoucna je do nich vkládán velký potenciál, protože stále roste množství dat, která

jsou třeba analyzovat a unsupervised learning metody dokážou nacházet souvislosti, které by člověk odhalit nedokázal[11, 12].

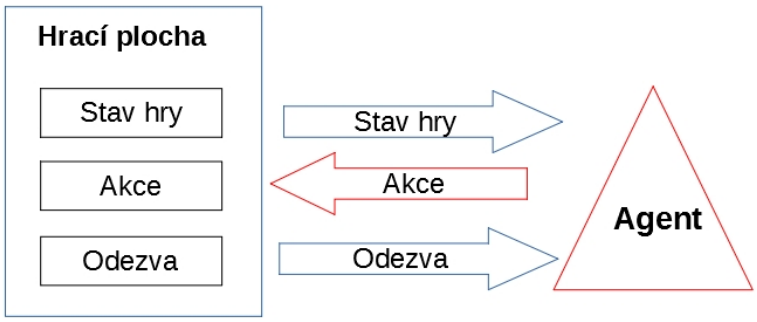
### 2.2.2 Reinforcement learning

Ani reinforcement learning není využit v praktické části této práce a tudíž následující popis je pouze povrchní shrnutí informací.

Česky přeloženo „zpětnovazební učení“ je metoda strojového učení, která je inspirována behavioristickou psychologií a je částečně podobná způsobu, jakým se učí malé děti[10]. Oproti ostatním metodám ML je tato rozdílná v tom, že je aplikována ve formě **agentů**, kteří jsou vloženi do **prostředí**[4], ve kterém se musejí sami adaptovat a vytvořit si systém **pravidel chování** na základě **odezvy** z prostředí a následně je aplikovat pro dosažení maximálního užitku. Toto je asi hlavní rozdíl oproti ostatním metodám ML, které se zaměřují na nalezení vzorů v datech nebo predikci. Tato metoda využívá přístupu *pokus*-omyl. Výhodou také je, že si vystačí s menším vzorkem dat oproti ostatním metodám[9, 13].

Příkladem může být UI podpořený počítačový software *AlphoGo*, který byl vytvořen panem Fan Hui. Této UI byly předány pouze základní principy hry a to, že cílem hráče je obklopit svými kameny větší území než soupeř. *AlphaGo* bylo schopno se naučit a vytvořit si systém pravidel a akcí pro všechny možné kombinace[5], které dokázalo zužitkovat při hře proti světovému šampionovi, jehož porazilo 4:1 na hry. V tomto příkladu **agent** byl počítač *AlphaGo* a **prostředí** byla hrací plocha s kameny, počítač zkusí táhnout (**akce**) a následně vyhodnotí pozitivní či negativní dopad (**odezvu**) na celkový stav hry viz. Obr. 2.3.

4 Někdy nazýváno Markovův rozhodovací proces
5 Až $10^{761}$ kombinací




*Obr. 2.3: Princip zpětné vazby v učení hry Go*

### *Používané aplikace*

- **Kritérium optimality** (*Criterion of optimality)* je algoritmus, který mapuje možnosti chování agenta a postupným zužováním výběru následující akce, dosahuje nejoptimálnějšího řešení.

- **Aplikace hrubé síly** (*Brute Force*) je svým přístupem relativně jednoduchá metoda. Spočívá v rozdělení možných postupů k dosažení výsledku a jejich následným testováním s tím, že jako ideální je zvolena metoda s největším užitkem. Nevýhodou tohoto přístupu je z logiky věci to, že možné cesty k dosažení kýženého výsledku mohou být v některých případech takřka nekonečně a tudíž převyšující dostupnou výpočetní kapacitu.

- **Užitkové funkce** (*Valuation function)* se snaží dosáhnout optimálního výsledku tím, že má přednastaveny očekávané reakce a jednotlivé postupy s nimi porovnává. Ten jež se bude nejvíce shodovat je poté zvolen.

### *Využití*

Nejpříhodnějším využitím reinforcement learning metody je pro aplikace, kde se snažíme optimalizovat reakce systému z dlouhodobého hlediska a nehledíme na krátkodobé nezdary. Velké uplatnění nachází v simulačních modelech, kde známe přesný popis prostředí, ale nikoliv analytické řešení problému. Příkladem aplikace může být i možnost robota, který se sám naučí vykonávat jistou činnost (otevřít dveře, něco uchopit). Můžeme jej nechat dlouho vykonávat tuto činnost, dokud nepřijde s optimální řešením → otevřou se dveře, uchopí předmět[8, 13, 14].

### 2.2.3 Supervised learning

Česky přeloženo *„učení s učitelem“* je oproti předchozím dvěma metodám rozdílné v tom, že algoritmu předkládáme datový vzorek, kde je obsažena jak množina vstupů (X), tak i množina jim příslušících výstupů (Y). Cílem je, aby byl algoritmus schopen nalézt vazby uvnitř datového vzorku, a poté je využít pro predikci dalšího průběhu. Data pro algoritmus mohou být poskytována člověkem nebo automaticky, kdy tak jednají právě jako „učitel“[9]. Neuronové sítě použité později v této práci jsou příkladem využití ML se supervised learning, kde máme ať už reálná nebo umělá data a hledáme v nich závislosti, které chceme následně využít pro predikci dalších dat. [9]

$$Y = f(X) \tag{2.1}$$

Obecně se metody supervised learning rozdělují do dvou kategorií:

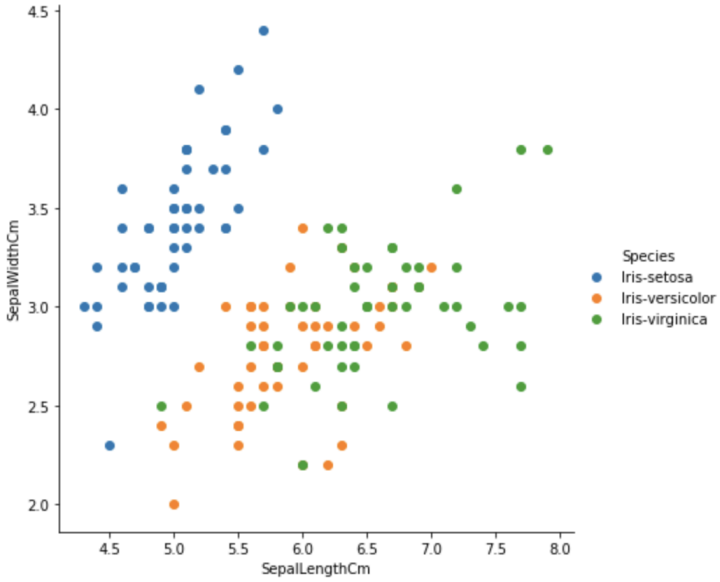
1. **Klasifikace**
2. **Regrese**

**Klasifikační** ML nám umožňuje na základě vstupních dat rozhodnout do, které kategorie bude spadat výstup. Pro ilustraci si lze představit, že po poskytnutí trénovacích dat běžné lidské populace (výška, váha, věk), může algoritmus odhadovat, zda daný vzorek spadá do skupiny žena (1) nebo muž (0). Výsledkem pak může být třeba graf obsahující dělící linii nebo barevné rozdělení na skupiny, podle které spadají data do jedné z těchto kategorií. Na Obr. 2.4 můžeme vidět klasifikační rozdělení druhů orchidejí podle velikosti jejich okvětních lístků[6]. Tato metoda se hodí pro typy úloh, kde neřešíme kvantitativní výsledek, ale právě kvalitativní. I když v některém z kroků řešíme i kvantitativní problém, kdy řešíme pravděpodobnost jednotlivých kategorií, proto částečně připomíná regresní modely[8, 9].

6 Klasický datový vzorek pro ML dostupný z: https://archive.ics.uci.edu/ml/datasets/iris

Nejčastěji používané metody klasifikační ML jsou:

- **Logistická regrese**
- **Analýza lineárního diskriminantu**
- **K-Nearest neighbors**



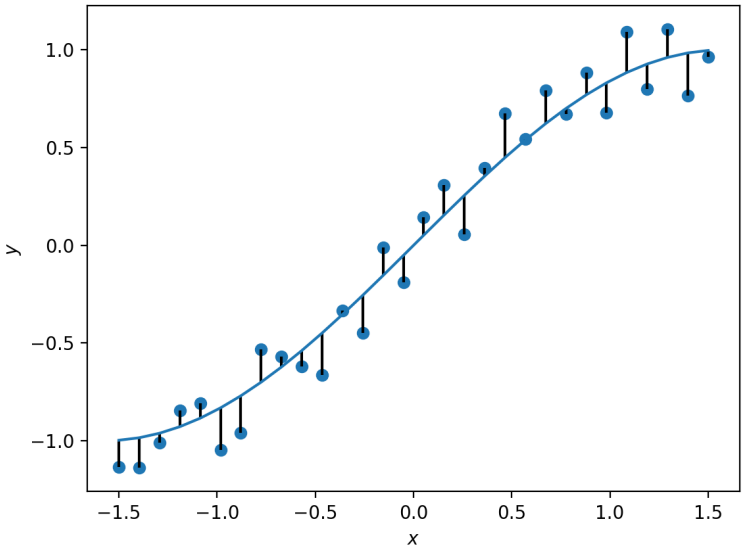

*Obr. 2.4: Příklad klasifikace typů orchidejí podle šířky a délky okvětních lístků*

**Regresní** ML zakládá na kvantitativním odhadováním výsledku. Využíváme různých typů regrese a výpočtů. Jednoduché využití může být ku příkladu odhad platu zaměstnance podle délky zaměstnaneckého poměru. Na Obr. 2.5 vidíme regresní křivku vytvořenou proložením náhodně vytvořených bodů. Algoritmus se snaží vytvořit křivku tak aby vzdálenost jednotlivých bodů od křivky byla co nejmenší. Snažíme se „nafitovat“ křivku na dané body za pomocí využití chybových funkcí jako SSE – sum of squared erros (suma chyb na druhou). Rovnice (2.2) popisuje funkci pro vykreslení regresní přímky. Zde v této rovnici je $\beta_0$ posunutí počátku křivky a $\beta_1$ je sklon křivky ,někdy jsou také nazývány jako koeficienty regrese [8, 9, 12].

$$Y \approx \beta_0 + \beta_1 X \tag{2.2}$$

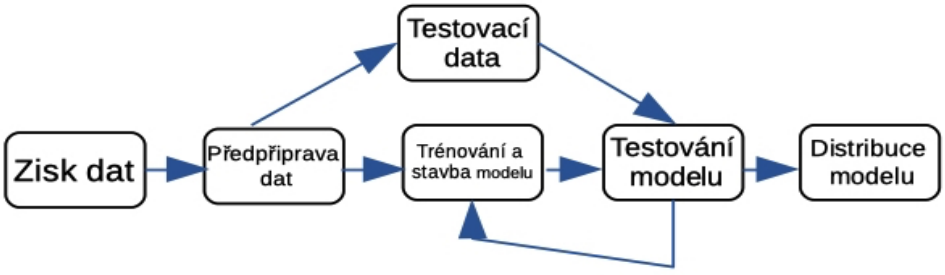
Nejčastěji používané metody regresní ML:

- Lineární regrese
- Polynominální regrese
- Support Vector




*Obr. 2.5: Ukázka proložení bodů pro získání regresní křivky*

## 2.3 Postup tvorby ML




*Obr. 2.6: Obecný postup tvorby modelu ML*

Pokud bychom zobecnili běžný postup při tvorbě ML modelu, dostali bychom přibližně proces jako na Obr. 2.6. Jako první krok musí být samozřejmě získání dat. Ta by měla následně projít nějakou předpřípravou. V této fázi se většinou uplatňuje z-scoring[7] či jiné škálování dat a také třeba odstraňování odlehlých hodnot. Tyto metody

7 Vychází ze standardního rozdělení podél průměru

mají za cíl to, aby následný proces učení byl co nejefektivnější. Poté rozdělíme připravená data na trénovací a testovací sadu. Účelem je, abychom mohli model ověřit na datech u kterých známe správný výsledek a tudíž můžeme určit přesnost modelu. Po rozdělení dat se model učí na trénovacích datech požadovanou činnost pomocí metod popsaných výše. Následně využijeme testovacích dat k ověření výsledku. Snažíme se zabránit takzvanému *overfittingu*. Podle potřeby model upravíme tak, abychom dosáhli co největší přesnosti. Následně model distribuujeme a nasazujeme na další data[8, 15].

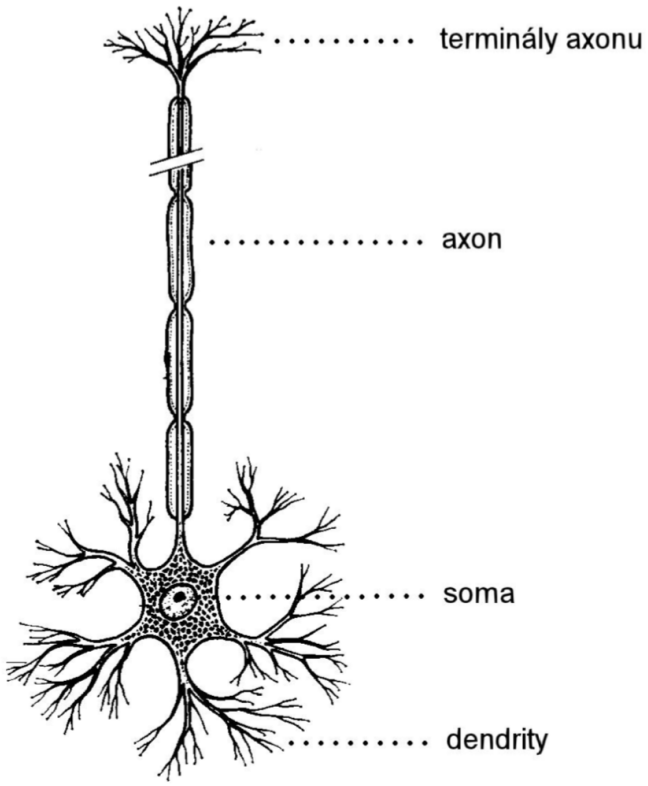
# 3 Neuronové sítě

Historicky bychom počátek neuronových sítí (ANN, Artificial Neural Network) mohli datovat do roku 1943. Tehdy Walter Pittse a Warren McCulloch uvěřejnili jeden z prvních matematických modelů umělého neuronu. Dali tak podnět ke vzniku nového odvětví. V roce 1949 je následoval Donald Hebb s knihou *„The Organization of Behaviour“*,ve které postuloval pravidlo pro učení synapsí neuronů[8]. Poznatky čerpal z myšlenky, že vlastnosti jednotlivých neuronů se dají pozorovat jako podmíněné reflexi u všech živočichů[14, 15]. Další pokrok přinesl Frank Rosenblatt, jenž zobecnil Pittsův a McCullochův neuron a navrhl pro něj učící algoritmus *Perceptron*[9]. Vydal také jednu z prvních knih o neurovýpočtech[9], a proto bývá také někdy považován za zakladatele tohoto oboru. V následujících letech oblíbenost celého oboru značně kolísala, jelikož nebyly vyřešeny problémy jako, jak na učení více vrstev spojených neuronů v neuronových sítích. Bez tohoto totiž nebyl samotný perceptron schopen vyřešit jednoduchou funkci X**OR** (vylučovací disjunkce), což bylo tomuto oboru značně vyčítáno. Přelom přišel v roce 1986, kdy David Rumelhart, Geoffrey Hinton a Ronald Williams uveřejnili článek v němž popsali algoritmus pro zpětné šíření chyby ve vícevrstvých neuronových sítích – *Backpropagation.* Od té doby opět narůstal zájem o tento vědní obor a zůstává tak až do dnes. Začaly se objevovat nové druhy neuronových sítí a nové možnosti učících algoritmů. Studium neuronových sítí je nyní na vrcholu vlny zájmu o tento obor a je otázkou, jak bude dále pokračovat[11]. Co lze říct je, že neuronové sítě a na nich postavené aplikace jsou dnes součástí skoro,

8 Hebbovo učení
9 *Principles of Neurodynamics*

každého analytického programu a pomalu se dostávají do stále běžnější oblasti jako telefony, televize, autonomní vozidla, vyhledávací algoritmy a jiné.

Základní myšlenkou NN je přiblížit se struktuře podobné lidskému mozku. Neuronová síť biologických neuronů je tvořena jednotlivými neurony. Biologický neuron (Obr. 3.1) je tvořen tělem neboli *soma*, poté vstupními a výstupními kanály – *dendrity* a *axony.* Axony mají na konci terminály, kterými se připojují na dendrity dalších neuronů a tím tvoří neuronovou síť. Přenos informací mezi neuronu umožňuje chemická vazba *synapse*. [11, 15, 16]

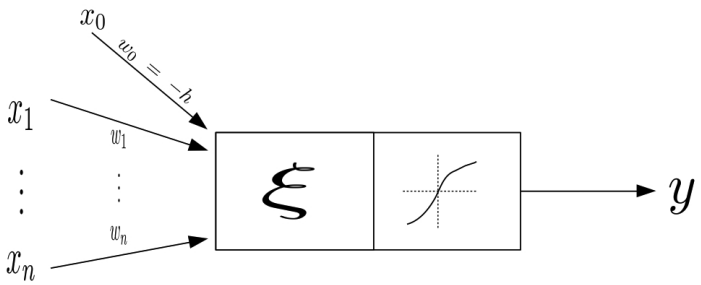



*Obr. 3.1: Nákres biologického neuronu [11]*

## 3.1 Formální neuron

Základním kamenem neuronové sítě je matematický model biologického neuronu – formální neuron, také nazývaný *perceptron*. Jeho struktura je schématicky znázorněna na Obr.3.2 . Neuron má obecně *n* vstupů $x_1, \ldots, x_n$, které simulují dendrity.

Těmto vstupům jsou přidělovány synaptické váhy $w_1, \ldots, w_n$. Tyto váhy určují důležitost a vliv jednotlivých vstupů na spuštění neuronu.




*Obr. 3.2: Schéma formálního neuronu*

Váhy mohou být i záporné a tudíž fungovat jako blokátor jednotlivých vstupů. Fungují tedy jako vstupní filtr pro vstupy z okolí nebo z jiných neuronů. Suma součinů vah a vstupů tvoří *vnitřní potenciál* neuronu.

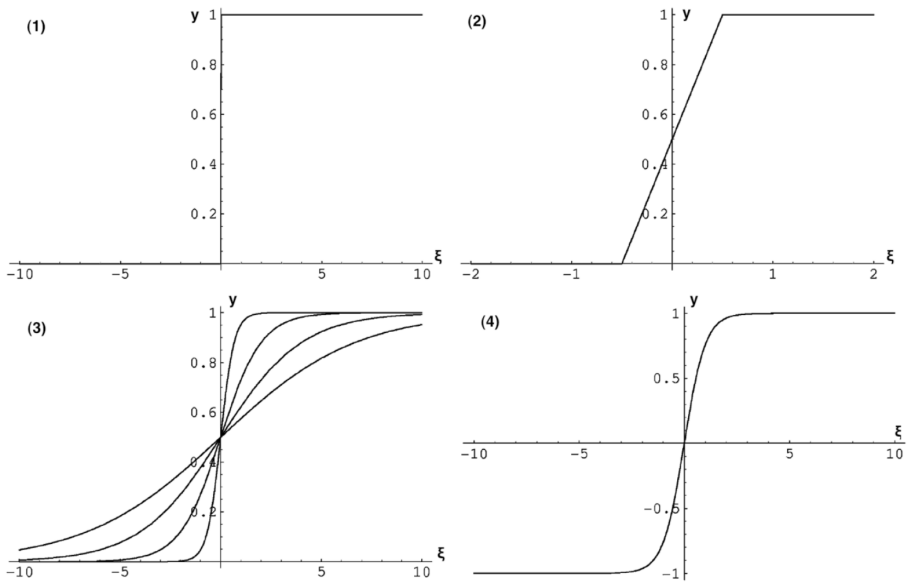
$$\xi = \sum_{i=0}^{n} w_i x_i \tag{3.1}$$

Součástí vstupní matice může být i *bias*. Jedná se obrácenou prahovou hodnotu *h*. Je začleněn do vstupní matice a má i vlastní váhu. Při překročení prahové hodnoty je neuron excitován a vyšle impuls (výstupní hodnotu). Cílem *biasu* je také vymanění funkce z lokálního minima do globálního minima.[11, 14, 15, 17].

$$\xi = w_0 + \sum_{i=0}^{n} w_i x_i = b + \sum_{i=0}^{n} w_i x_i \tag{3.2}$$

Nelineární nárůst výstupní hodnoty $y = \sigma(\xi)$ při dosažení prahové hodnoty potenciálu h je dán aktivační (přenosovou) funkcí. Nejjednodušším typem *aktivační funkce* je tzv. ostrá nelinearita. Dalšími typy aktivační funkce může být spojitá linearita, standardní sigmoida, hyperbolický tangens. Jejich průběh je znázorněn na Obr. 3.3. Možné stavy pro ostrou nelinearitu pak můžeme zapsat takto:

$$\sigma(\xi) = \begin{cases} 1 \; jestliže \; \xi \geq 0 \\ 0 \; jestliže \; \xi < 0 \end{cases} \tag{3.3}$$




*Obr. 3.3: Průběh aktivačních funkcí - (1)skoková, (2)spojitá lineární, (3)standartní sigmoida, (4)hyperbolický tangens; [11]*

Formálně lze tedy zapsat, že neuronové modely jsou funkcí:

$$y = f_{(X,W)} \tag{3.4}$$

Kde **X** je vektorem vstupů včetně biasu a **W** je vektorem synaptických vah.

## 3.2 Neuronové architektury

Neuronové sítě jsou tvořeny spojením formálních neuronů tak, že výstup jednoho je vstupem do jednoho či více dalších neuronů. Počet neuronů a jejich vzájemné propojení určuje *topologii* sítě. Tyto jednotlivě propojené neurony rozdělujeme do vrstev. Nejčastější rozdělení vrstev je na *vstupní*, *skryté* a *výstupní*. V jednotlivých vrstvách může být různý počet neuronů. Stavy všech neuronů v síti určují *stav neuronové sítě* a synaptické váhy představují *konfiguraci neuronové sítě*. [15, 18]

### 3.2.1 Dynamické změny

Neuronová síť se časem adaptuje a mění jednotlivé váhy a spojení. Tyto změny můžeme rozdělovat podle jejich účinku do 3 kategorií:

- **Organizační změna –** jedná se o změnu topologie sítě. Síť se může rozšiřovat o další neurony, případně spoje. Většinou však síť zůstává stálá. Organizaci sítě však může rozlišit na *rekurentní* a *dopřednou*.

- **Aktivní změna** – neurony jsou nastaveny na základě počátečních vstupů, následně v čase probíhají výpočty a mění se stavy jednotlivých neuronů (*sekvenční výpočet)* nebo více neuronů najednou *(paralelní výpočet)*. Na základě toho zda mění svůj stav centrálně nebo nezávisle na sobě rozlišujeme ještě *centrální* a *asynchronní* modely.

- **Adaptivní změna –** na počátku se nastaví základní váhy jednotlivých spojů. Adaptováním se v čase postupně mění konfigurace sítě, tedy váhy jednotlivých spojení. Předpokládáme, že změna vah je spojitou funkcí v čase. Cílem této změny je najít takovou konfiguraci, která bude nejlépe realizovat předepsanou funkci a minimalizace chyb ve váhové prostoru [11, 15, 17].

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)} \tag{3.5}$$

# 4 Neuronové modely

## 4.1 MLP − Vícevrstvá neuronová síť

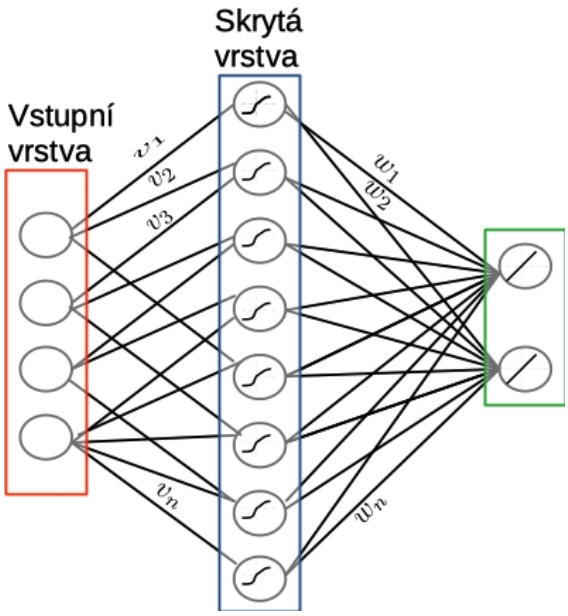
Jedná se o jeden z nejstarších modelů vícevrstvé neuronové sítě. Organizační strukturou se jedná o *dopřednou* síť. Jejím cílem je aproximace nějaké funkce $f^*$. Snažíme se namapovat určitý vstup na výstup a po naučení ji používat na predikci dalších hodnot. Tvoří základ většiny běžně využívaných aplikací strojového učení a se svými různými obměnami se jedná o nejpoužívanější typ NN vůbec.[3, 9, 11]

Na Obr. 4.1 vidíme schéma jednoduché MLP sítě s 1 skrytou vrstvou. Vstupní vrstva reprezentuje vektor vstupů do sítě $x_0, \ldots, x_n$. Neurony v této vrstvě jsou následně excitovány a poté jsou zesíleny či zeslabeny pomocí synaptických vah $v_1 \ldots v_n$ a přivedeny do další vrstvy. Neurony v následující skryté vrstvě provedou součet k nim přivedených vazeb podle již známého vztahu $\sum_{i=0}^{n} v_{ij} x_{ij}$ a následně je využit v aktivační

funkci daného neuronu, která určuje excitaci tohoto neuronu. Skryté vrstvy obyvkle obsahují neurony, které mají nelineární aktivační funkci. Na schématu je využita aktivační funkce *sigmoida.* Takto je neuron ve skryté vrstvě excitován a informaci posílá do další vrstvy. V našem schématu následuje již vrstva výstupní. Do této vrstvy, opět vazbami posílenými či zeslabenými synaptickými vahami $w_1 \dots w_n$ , vstupují informace ze všech neuronů. Ty jsou opět sumarizovány a využity v aktivační funkci, v případě výstupní vrstvy funkci *linearní.* Poté nastává fáze „učení“, kdy se aktualizují hodnoty vah a biasu, které při výstupu neprodukovali požadované výsledky[14, 17]. Výstup sítě je tedy možno sumarizovat následující rovnicí:

$$y = \vec{w} \cdot \sigma(\vec{v}) = \vec{\nu} \cdot \sigma(\vec{V} \cdot \vec{x}) \tag{4.1}$$

- kde $\vec{w}$ je vektor vah výstupního neuronu
- $\sigma(\vec{v})$ je aktivační funkce skryté vrstvy
- a $\vec{V} \cdot \vec{x}$ je součin vektorů vstupů a vektor vah skryté vrstvy




*Obr. 4.1: Schéma MLP s 1 skrytou vrstvou*

### 4.1.1 Aktivační funkce

***Standardní sigmoida***

Jedná se o často využívanou aktivační funkci v neuronových sítích. Její výhodou je spojitost a hladkost v celém průběhu a z toho její vyvozená

diferencovatelnost. Umožňuje nám spojitě aproximovat ostrou nelinearitu.[11] Na rovnici (4.2) vidíme její matematický popis.

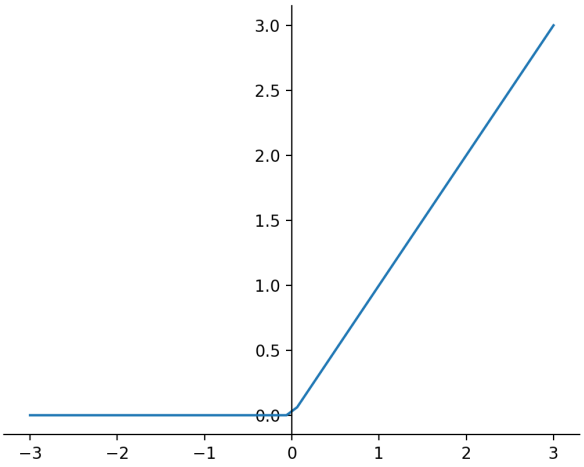
$$\sigma = \frac{1}{1 + e^{-\lambda x}} \tag{4.2}$$

Diferencovatelnost standardní sigmoidy je také výhodou i pro následné učení za pomocí *back propagation*. Parametr *strmosti* $\lambda$ je reálná složka funkce a mění nelineární nárůst funkce v okolí nuly, tj. určuje míru „rozhodnosti“ neuronu. Vliv strmosti na funkci lze vidět na Obr. 3.3. Pokud bychom vzali $\lambda \to \infty$ dostáváme ostrou nelinearitu. Povětšinou je ale využíváno $\lambda = 1$, může se však lišit pro každou vrstvu a každý jednotlivý neuron ve vrstvě.

### *ReLU – Rectified Linear Unit*

Jedná se o lineární aktivační funkci, která dává výstup pokud se hodnoty vstupu pohybují nad 0, jinak vrací 0. Matematicky je popsána rovnicí (4.3).

$$y = x^{+} = max(0, x) \tag{4.3}$$




*Obr. 4.2: Zobrazení aktivační funkce ReLU*

A graficky přibližně jako na Obr. 4.2Výhodou ReLU je její nenáročnost na výpočetní kapacitu. Oproti sigmoidě nebo hyperbolickému tangensu může být až o polovinu rychlejší. Dále také netrpí problémem saturace funkce jako sigmoida nebo

tanges[10], u kterých probíhá rozlišení pouze v úzké oblasti okolo 0,5 a větší vstupy či velice malé vstupy spadnou do oblastí 0 nebo 1. Díky tomu je docíleno také menšího množství excitovaných neuronů najednou, což může být žádaná vlastnost pro některé funkce, připomínající trochu lidský mozek, kde také né všechny neurony vystřelují informace najednou. Díky tomu mají ReLU menší tendenci k *overfittingu* a lepší výsledky pro predikci.

Nevýhoda ReLU vychází, jak je možné vidět z grafu, z toho, že pro veškeré záporné vstupu je výstupní hodnota rovna 0. Tento problém bývá někdy nazývám jako *dying ReLU* neboli „umírající“ ReLU. Jelikož i strmost funkce v záporné části spektra je také rovná nula, je časté, že jakmile jednou neuron dosáhne záporné hodnoty, nemusí se již být schopen vrátit zpět. V důsledku tohoto problému, je možné při praktických aplikacích skončit s neuronovou sítí v níž bude spoustu „mrtvých“ neuronů. Často se to stává pokud je příliš vysoký *learnign rate* nebo velký záporný *bias.*

Tyto problémy lze většinou zmenšit použitím jiného learning rate nebo bias, případně využitím existujících variant původního ReLU[11][11, 19, 20].

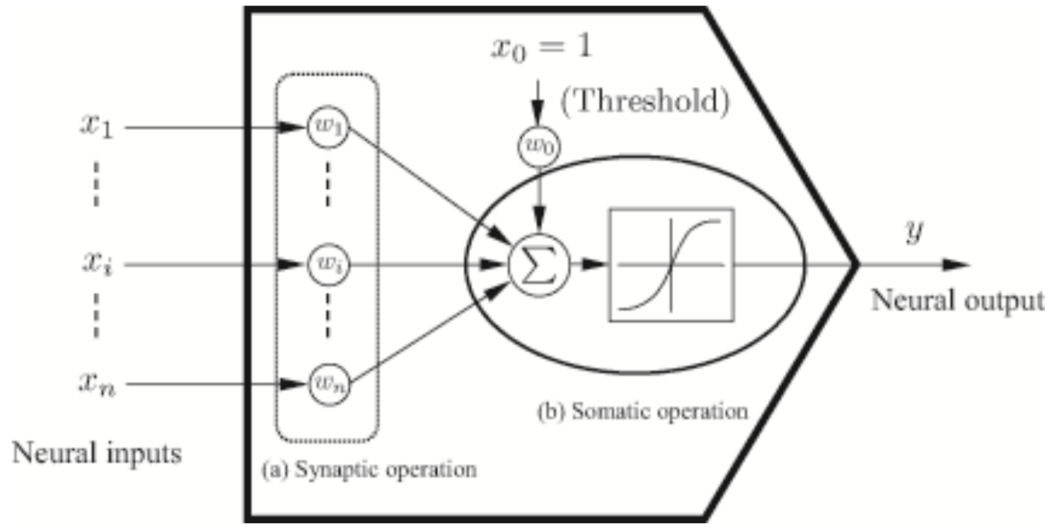
## 4.2 ELM – Extreme learning machine

Jedná se o variaci na MLP. Stojí na bázi SLFN (*Single layer feedforward network),* kdy neuronová síť obsahuje pouze jednu skrytou vrstvu, avšak o **extrémním** množství neuronů. Využívá navíc učící algoritmus nazvaný **ELM**. Hlavním rozdílem, že při dávkování dat do neuronové sítě probíhá pouze prvotní náhodné zvolení synaptických vah skryté vrstvy a bias vstupu, pokud je aktivační funkce $\sigma$ nekonečně diferencovatelná (sigmoida, sinus, exponenciála, atd.). Výhodou je až tisíckrát menší výpočetní náročnost aplikace této sítě. Tato síť je schopna menší chyby na tréninkových datech a také lepší generalizace oproti běžným učícím algoritmům jako *back propagation.* Následně proběhne výpočet uvnitř skryté vrstvy a za pomocí výstupních vah jsou výsledky předány do výstupní vrstvy, kde může být již standardní lineární aktivační funkce[21, 22].

10 Problém „mizejícího“ gradientu (Vanishing gradient)
11 Leaky ReLU nebo Exponential LU

## 4.3 HONU – Higher Order Neural Unit

HONU jsou nelineárním modelem neuronové sítě a jsou v přímé analogii k Taylorovu rozvoji. U HONU je možné měnit řád polynomu a proto lze nastavovat kvalitu nelineární aproximace. Ovšem s rostoucím stupněm polynomu stoupá i časová a výpočetní náročnost na hardware. Běžnými prostředky jsme dnes schopni umožnit strojové učení až do 3. stupně polynomu. Jednotlivé typy HONU se můžou lišit svou vnitřní agregační funkcí ($\sum$). Pro LNU se jedná o běžnou lineární funkci avšak u HONU vyšších řádů jde o funkci nelineární[23].




*Obr. 4.3: Schéma HONU modelu neuronu[4]*

Na Obr. 4.3 vidíme schéma HONU neuronu. Jako u ostatních modelů i zde máme vektor vstupů $\vec{x} = [x_1 \dots x_n]^T$ a vektor synaptických vah $\mathbf{W} = [\mathbf{w_1} \dots \mathbf{w_n}]^\mathbf{T}$. Vnitřní agregační funkce poté obecně vypadá takto[23]:

$$y = \sum_{i=0}^{n} \sum_{j=i}^{n} \cdots \sum_{\dots}^{n} (x_i x_j \cdots x_{\dots}) * w_{i,j,\dots} \tag{4.4}$$

kde bias $x_0 = 1$,

$n$ odpovídá množství vstupů do neuronů $x_i$.

Tuto rovnici následně upravíme do tvaru pro zvolený řád polynomu. Pro typ LNU,QNU, CNU, jež jsou v této práci použity vypadají upravené rovnice následovně.

Pro **LNU** (Linear Neural Unit), jak již bylo zmíněno je rovnice lineární:

$$y = \sum_{i=0}^{n} x_i w_i \tag{4.5}$$

Pro **QNU** (*Quadratic Neural Unit*) je již funkce nelineární a rovna:

$$\sum_{i=0}^{n} \sum_{j=i}^{n} x_i x_j w_{ij} \tag{4.6}$$

Pro **CNU** (*Cubic Neural Unit*) je funkce také nelineární a odpovídá:

$$\sum_{i=0}^{n} \sum_{j=i}^{n} \sum_{k=j}^{n} x_i x_j x_k w_{ijk} \tag{4.7}$$

### 4.3.1 Optimalizace HONU

Kombinací vyššího řádu vstupů a vah, jsme schopni dosahovat efektivních výsledků z hlediska výstupu z neuronu, což bylo potvrzeno při aplikacích od adaptivního řízení po klasifikační problémy. Ovšem jednou z překážek, která se objevila v minulých výzkumech byl problém z nárůstem parametrů nutných pro učení. Optimalizace této problematiky byla tématem některých prací[4]. Cílem bylo zjednodušení výpočtů se zachování účinnosti modelu.

Jako příkladem naznačíme zjednodušení QNU. Základem je zápis původní funkce v maticovém tvaru s vektorem ***x*** jako vektorem vstupů a rozšířenou maticí ***W*** synaptických vah.

$$\sum_{i=0}^{n} \sum_{j=i}^{n} x_i x_j w_{ij} = [\ (x_0 = 1)\ x_1\ \cdots\ x_n\ ] * \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ 0 & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & w_{n,n} \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{4.8}$$

Vzhledem k tomu, že rozšířená matice vah i vektor vstupů mají stejný rozměr, je pro naše výpočty dostačující využít pouze horní trojúhelníkovou matici, čímž vyloučíme nulové prvky. Pro další zjednodušení výpočtů odstraníme nutnost tvorby sumací součinů prvků matice a vektoru, jejich převedením do podoby tzv. d*louhých vektorů*.

$$y = \mathbf{rowx} \cdot \mathbf{colW} = [x_0\ x_1\ \ldots x_i x_j\ \ldots x_n x_n] \cdot [w_{0,0}\ w_{0,1}\ \ldots w_{i,j}\ \ldots w_{n,n}]^T \quad (4.9)$$

kde délka ***rowX*** a ***colW***, vyplývá vnitřní agregační funkce a liší se pro jednotlivé stupně polynomu[4, 24]. Pomocí této optimalizace dosahují HONU dobrých výsledků a byly třeba využity k výzkumu pro hledání plicního nádoru[24].

## 4.4 LSTM – Long Short Term Memory

Oproti předešlým modelů spadá LSTM do skupiny *rekurentních* neuronových modelů někdy zkracováno jako RNN (*Recurrent Neural Network*).. Rozdíl oproti předchozím modelům, které jsou dopředné je ten, že rekurentní modely na začátku v $t = 1$ neberou ve výpočtech v potaz pouze hodnotu vstupů, vah a biasu, ale také předchozí hodnoty získané ve vrstvách v $t - 1$. Jinými slovy akce rekurentní sítě mají dva druhy vstupů: *aktuální hodnoty vstupů* a minulé vstupy neboli *paměť*. Tato paměť je součástí sítě ve formě skryté vrstvy, která je částečně oddělená.

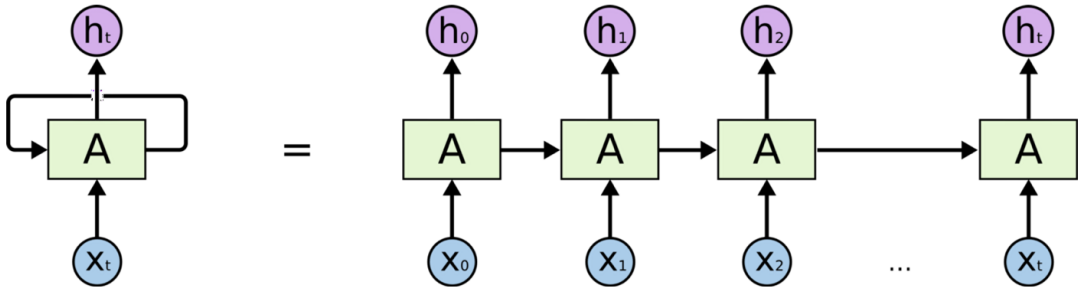
$$y_t = \sigma(W x_t + U y_{t-1}) \quad (4.10)$$

kde *W* je matice vah vstupů jako u dopředné sítě

*U* je přechodná matice pro zakomponování předchozích stavů $y_{t-1}$

Vnitřní funkcí $\sigma$ může být standardní sigmoida nebo tangenta. Avšak musí splňovat úplnou diferencovatelnost.

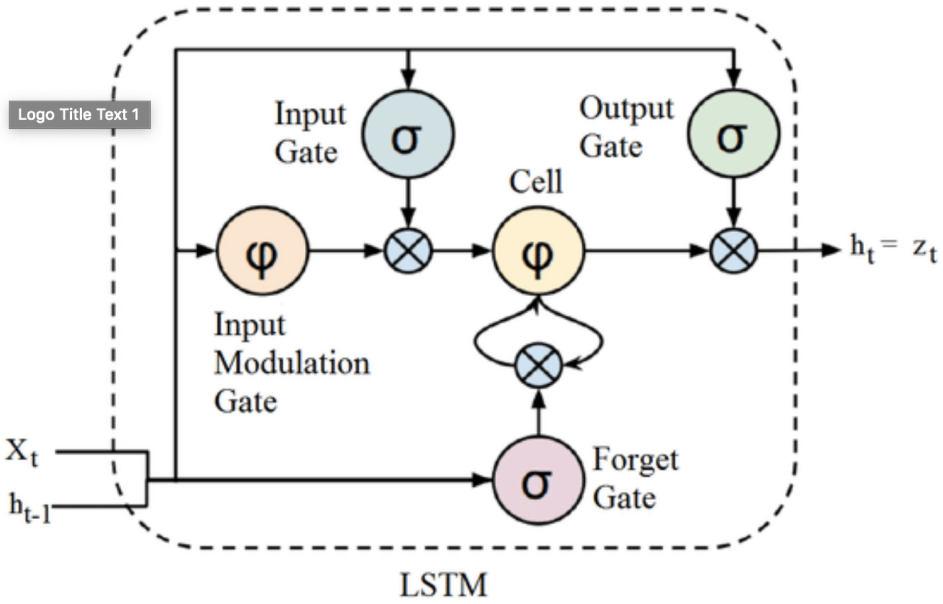
Tyto modely jsou hojně využívané pro predikci a učení na sekvenční datech jako třeba text či video, také překlad textu. Nejedná se pouze o body jako třeba v časových řadách, kdy nám jde pouze o zjištění další hodnoty, ale zde bereme v potaz celou datovou řadu jako celek. Nejčastěji při využití učícího algoritmu *backpropagation.* A stejně jako ostatní modely využívající *backpropagation* i u tohoto se postupně objevil problém mizejícího či explodujícího gradientu. LSTM byly navrženy jako částečné řešení tohoto problému. LSTM byly navrženy tak, aby dokázaly zachovat chybu vznikající při výpočtech a poslat ji dál. Tím pádem se vzniklá chyba v průběhu času nenásobí a nevzniká již zmíněný problém běžných RNN[5, 25]. Na Obr. 4.4 lze vidět jak funguje celý princip RNN. Předchozí stav je vždy předáván dále do sítě.




*Obr 4.4: Design RNN [32]*

### 4.4.1 Struktura LSTM

Aby bylo docíleno funkčních vlastností LSTM, je tento model tvořen specifickou strukturou. Využívá takzvaných *paměťových buněk***.** Každá z těchto buněk má specifickou vnitřní strukturu. Tato struktura je nejčastěji tvořena 3 regulátory v literatuře nazývány *gate* neboli „brány“. Tyto jsou: vstupní, výstupní a brána „zapomenutí“. Tyto brány se chovají jako klasické uzly v neuronové síti a blokují nebo posílají dál vstupy, které do nich vstoupí na základě určených podmínek.




*Obr 4.5: Struktura paměťové buňky LSTM[32]*

Na Obr. 4.5 vidíme strukturu paměťové buňky LSTM. Na tomto obrázku je navíc ještě součástí struktury brána pro modulaci vstupů. Jedná se o jednu z variací

LSTM. Nejběžnější variantou je však s 3 branami. Každá z bran obsahuje vlastní aktivační funkci[26]. Stav paměťové buňky z Obr. 4.5 je poté určen jako:

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t \tag{4.11}$$

kde $c_t$ je stav paměťové buňky v čase *t* (resp. *t-1)*

$f_t$ určuje stav brány zapomenutí neboli *forget gate*

$i_t$ je vstupní brána a $g_t$ je brána modulace vstupů

Výstupem skrytého stavu v neuronové síti je poté:

$$h_t = o_t \circ tanh(c_t) \tag{4.12}$$

Kde $h_t$ představuje skrytý stav a $o_t$ je výstupní brána. Každá z těchto bran má jedinečnou funkci v paměťové buňce LSTM. **Vstupní** brána *input gate*, v této bráně se rozhoduje zda vstup (informace) smí vstoupit do paměťové buňky. Její aktivační funkcí je povětšinou sigmoida s rozsahem [0,1].

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{4.13}$$

Jak již bylo zmíněno, vstupní brána bývá někdy rozšířena o tzv. **modulaci** vstupů *Modulation input gate*. Ta umožňuje klasické vstupní bráně to, že využitím tangenciální aktivační funkce o rozsahu [-1, 1], je vstupní bráně umožněno „zapomenout“ předchozí stav paměťové buňky.

$$g_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{4.14}$$

Výstupní brána *output gate* je někdy také nazývána „ohniskovým“ vektorem. Rozhoduje o tom, které informace budou zachovány a použity v dalším skrytém stavu buňky.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{4.15}$$

Brána pro zapomenutí *forget gate*, pak už jak z názvu vypovídá rozhoduje, které informace z předchozího stavu v čase *t-1* budou zapomenuty.

$$f_t = \sigma(W_f[h_{t-1}, x_f] + b_f) \tag{4.16}$$

Spolupráce těchto bran v paměťové buňce LSTM a jejich konstrukce jsou tím rozdílem, kvůli kterému bývají upřednostňovány při tvorbě RNN.

# 5 Učící algoritmy

Základním kamenem všech neuronových sítí, je také jejich schopnost naučit se podle vstupů na výstupy a najít závislosti v trénovacích datech.

## 5.1 Backpropagation

Algoritmy využívající Backpropagation, znamenali revoluci ve využívání a tvorbě neuronových sítí. Do té doby mnohé problémy, které se zdáli neřešitelné pomocí pouze dopředných sítí a bez korekce chyb, nebylo možno vyřešit. Zpětné šíření chyby neboli b*ackpropagation* předurčilo další rozvoj NN. Hlavní podstatou BP je, že na výstupní vrstvě neuronové sítě sledujeme chybu (odchylku) $\boldsymbol{e}$ vypočtené hodnoty $\boldsymbol{y}$ a reálné (naměřené hodnoty) $\boldsymbol{y_r}$.

$$e = y_r - y \tag{5.1}$$

Podle toho se následně přizpůsobují váhy v nižších vrstvách a také bias. BP využívá různých optimalizací ať už se jedná o gradientního spádu nebo třeba metody nejmenších čtverců jako Levenberg – Marquardt[11, 16]. Celým systém učení spočívá v optimalizace chybové funkce (angl. c*ost function* nebo také *loss function)*:

$$E = C(y_r - y)^2 \tag{5.2}$$

$$E = \frac{1}{2n} \sum_x \|y_r(x) - y(x)\|^2 \tag{5.3}$$

Ta je závislá na vstupním vektoru dat $[x_0 \dots x_n]$, synaptických vahách $[w_0, \dots, w_n]$ a reálných datech. Pokud vezmeme vstupní data a reálná data jako stálá tak to co nám mění vypočtené hodnoty ze sítě jsou synaptické váhy a bias. Optimalizace chybové funkce tedy spočívá v hledání takového ustanovení synaptických vah, aby její výsledek byl co nejmenší. Tedy aby se funkce nacházela ve svém **globálním minimu**. Pro tento účel využijeme gradient chybové funkce.

$$\bigtriangledown E = \frac{\partial E}{\partial w} \tag{5.4}$$

Na základě gradientu hledáme ono minimum a v každém kroku přizpůsobujeme synaptické váhy[12]. Úpravu vah můžeme popsat pro každý časový okamžik.

$$\triangle w = -\frac{1}{2}\mu\frac{\partial e^2}{\partial w} \tag{5.5}$$

A jak již bylo zmíněno v 3.2.1 co se týče adaptivních změn:

$$w = w + \triangle w \tag{5.6}$$

***C*** je poté zvolená konstanta, která se obvykle volí $\frac{1}{2}$ pro usadnění výpočtů[9, 15].

## 5.2 Gradient Descent

GD je často využívaný algoritmus pro optimalizaci neuronových sítí. Bývá součástí i ostatních pokročilejších optimalizační algoritmů včetně BP. Funkční popis adaptace vah v konkrétním čase zapíšeme:

$$w_i(t+1) = w_i(t) - \mu\frac{\partial E}{\partial w_i} \tag{5.7}$$

Následně dosadíme chybovou funkci:

$$w_i(t+1) = w_i(t) - \mu\frac{1}{2}\frac{\partial e^2}{\partial w_i} \tag{5.8}$$

nebo:

$$w_i(t+1) = w_i(t) - \bigtriangledown C(\vec{w_i}) \tag{5.9}$$

Jako základní algoritmus pro optimalizaci je GD dostačující, avšak trpí některými nedostatky, které poté vynahrazují vylepšené metody optimalizace. Jednou z nevýhod je již jednou zmíněné ustálení v lokálním minimu místo globálního minima. Tato problematika je ovlivněna i zvolenou rychlostí učení *learning rate* $\mu$. Při správně zvolené rychlosti učení, můžeme dosáhnout lepšího výsledku.

## 5.3 Levenberg − Marquardt

Optimalizace pomocí algoritmu LM byl souběžně objeven svými vynálezci Levenbergem a Marquardtem. Princip spočívá v postupném (iteračním) hledání globálního minima chybové funkce za pomocí standardní metody nejmenší čtverců a svými parametry tedy spadá do oblasti mezi metody Gauss – Newtona a GD. Jedná se o robustnější metodu oproti Gauss-Newtonovi, díky čemuž je schopen najít optimalní řešení i pokud začíná z počátku iterovat mnohem dál[9, 27].

Původním cílem LM bylo dosažení druhého trénovací rychlosti druhého řádu bez toho, abychom byli nuceni počítat Hessovu matici[12]. Místo toho je její velikost aproximována jako:

$$\mathbf{H} = \mathbf{J^T J} \tag{5.10}$$

Poté můžeme hledání minima považovat za sumu kvadrátu chyb:

$$E = \frac{1}{2}\sum_{k=0}^{N} e^2(t) \tag{5.11}$$

Gradient chybové funkce je pak definován jako:

$$g = J^T e, \tag{5.12}$$

kde ***e*** je vektor chyb sítě a ***J*** je matice jakobiánů první derivace chyby vůči synaptickým vahám a biasu. Následně poté LM aproximuje Hessovu matici,

$$w = w - [J^T J + \mu \mathbf{I}]^{-1} \mathbf{J^T e}, \tag{5.13}$$

kde ***I*** je jednotková matice a $\mu$ je rychlost učení. Podle nastavení rychlosti učení se poté LM chová více jako Gauss-Newton nebo GD.

# 6 Měření teploty v řezu

Pro tvorbu algoritmu byla využita data získaná při experimentálním měření vývoje teploty při běhu turbomotoru. Kvůli zpřesnění údajů je pro získávání našich dat

12 Matice druhých parciálních derivací funkce

využito větší množství čidel než je běžné v klasické produkční výrobě. Jedná se tedy spíše o měření během výzkumné a vývojové fáze u turbomotoru.

## 6.1 Termočlánky

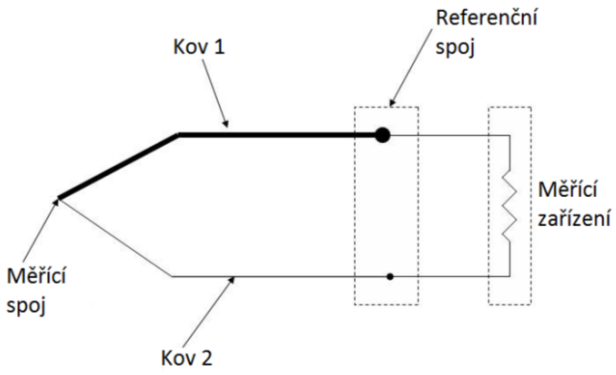
Princip fungování termoelektrických článků (termočlánků) je založen na Seebeckově jevu. V zásadě mají nosiče náboje v teplejší části vodiče větší energii a přesouvají se tedy do chladnějších částí vodiče. Obráceně nosiče z chladnější části se přesouvají do oblasti teplejší, avšak v menším počtu. Proto v průběhu dochází ke vzniku převahy nosičů s kladným nebo záporným[28]. Závislost teploty na termoelektrickém napětí termočlánku lze vyjádřit rovnicí

$$E = \sum x_t \; t' \qquad (\mu V) \tag{6.1}$$

Z napětí následně určíme teplotu pomocí rovnice [29]

$$t = \sum y_j E' \qquad (^{\circ}C) \tag{6.2}$$

Konstrukčně jsou termočlánky tvořeny vždy dvěma vodiči (viz. Obr. 6.1), jenž volíme tak abychom zajistili postačující nelineární závislost termoelektrického napětí na teplotě a zároveň zajistili odolnost vůči vnějším vlivům chemický či mechanickým. V průmyslu se nejčastěji využívá kombinace železo – měďnikl (FE-CuNi) nebo kombinace s niklem (NiCr-Ni). Avšak pro aplikace s vysokými teplotami se používají články ze vzácných kovů jako platina rhodium – platina (PtRh10-Pt). Podle složení rozdělujeme termočlánky na různé typy.

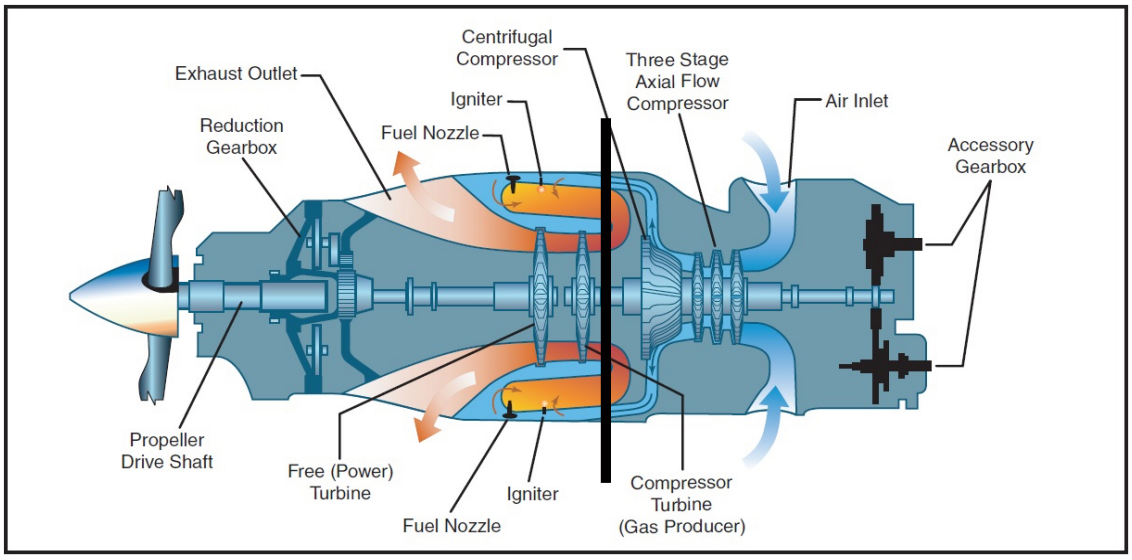



*Obr 6.1: Schéma termočlánku [28]*

Pro měření teplot v turbomotorech a podobných aplikací se využívá **typ K** se složením NiCr – NiAl s rozsahem měřitelných teplot pro dlouhodobé měření 0°C až 1100°C pro dlouhodobé měření, nebo **typ R** se složením PtRh13-Pt s rozsahem 0°C až 1600°C a odolný proti oxidaci a korozi.

## 6.2 Zapojení termočlánků

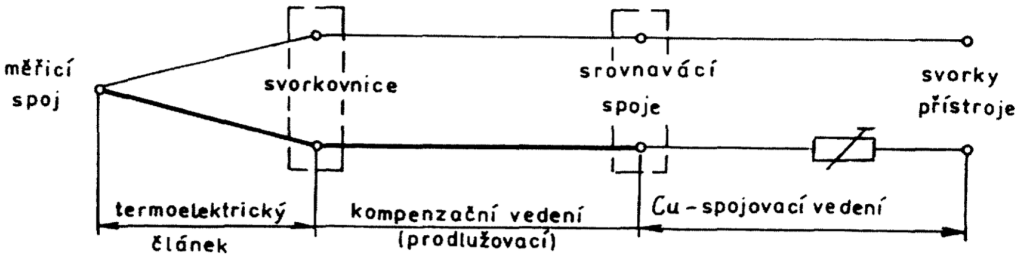
Měření probíhá na turbomotoru od okamžiku zapnutí po předem určenou dobu. Pro co nejpřesnější data je na motoru umístěno vícero termočlánku na různém průměru a obvodovém úhlu (viz. Obr.7.2). Na Obr. 6.2 lze vidět schéma turbomotoru od firmy Pratt & Whitney model Pt6, funkcionálně je toto schéma dostatečně podobné pro představu umístění měřících článků na námi zkoumané jednotce. Na obrázku je přibližné místo řezu zaznačeno černou čarou.




*Obr 6.2: Schéma turbomotoru PT6 [33]*

V tomto řezu jsou rozmístěny speciální nástavce, které jsou osazeny jednotlivými termočlánky. Z těch jsou následně získávána data. Na Obr. 7.1 vidíme schématicky zapojení jednoho termočlánku i s názvoslovím. Takto jsou následně zapojeny všechny termočlánky na nástavcích tvořící **hřebenovou sondu**. Jak již bylo zmíněno výše tento způsob je využívám nejvíce ve vývoji. V klasickém produkčním testování je většinou využito pouze pár termočlánků osazených po obvodu celého

motoru. V této práci se zabýváme jedním konkrétním řezem avšak během testů je takto umístěných hřebenových sond samozřejmě více na mnoha řezech.




*Obr 7.1: Schéma zapojení termočlánků[29]*

# 7 Algoritmus pro rekonstrukci teplotních polí v turbomotoru

Jak již bylo napsáno v úvodu, cílem této práce je nalezení algoritmu, který by dokázal rekonstruovat teplotní pole v řezu turbomotoru za poskytnutí co nejmenšího množství dat. V následující kapitole budou popsány jak testovaná data, tak jednotlivé neuronové modely použité pro predikci a následně jejich srovnání.

## 7.1 Datové vzorky

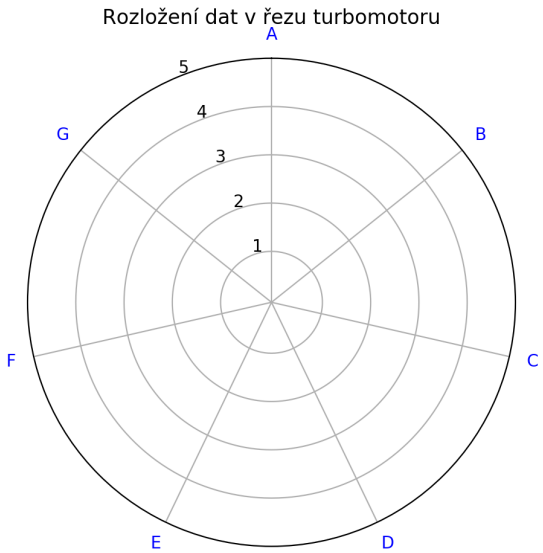
Algoritmu budou poskytnuty dva druhy dat – reálná a umělá. Cílem je nejprve algoritmus naučit a poté demonstrovat jeho funkci a úspěšnost na umělých datech, která vytvoříme.

### 7.1.1 Reálná data

Reálná data byla získaná, za pomocí metod popsaných v kapitole 6. Jedná se o anonymizovaná data z řezu turbomotoru přibližně v oblasti, kde dochází k průchodu vzduchu o největší teplotě za kompresorovou turbínou. Data byla odečítána od rozběhu do ustáleného běhu motoru a se stejnou frekvencí odečtů.

Jedná se tedy o data *ekvidistantní deterministické* časové řady, kdy hodnoty teploty *T* byly odečítány vždy po $\triangle t = 0.01s$. Časová řada obsahuje celkem **44 404** vzorků tedy od $t = 0.00$ až $t = 440.04$.

Data použitá v práci jsou již normalizovaná, což bude popsáno v následujících kapitolách a značení bylo zvoleno, aby nepřipomínalo žádný z existujících turbomotorů. Na Obr. 7.2 vidíme rozložení dat v řezu. Vidíme také značení, které bude použito po zbytek této práce. Po obvodu je rozmístěno 7 hřebenových sond – *A, B, C, D, E, F, G.* Každá z nich má čidlo na průměru 1 až 5. .




*Obr. 7.2: Schématický náznak rozložení dat z řezu turbomotoru*

### 7.1.2 Umělá data

Umělá data slouží jako datový vzorek, který využíváme k ověření funkčnosti a aplikovatelnosti naučené neuronové sítě pro náš problém. Byla vytvořena tak, aby napodobila data reálná svým rozložením a hodnotami. Pro jejich vytvoření bylo využito původních dat $\mathbf{Y_{real}}$, ta byla následně proložena funkcí sinus a prodloužena do požadovaného rozměr.

Pro další testovaní upravíme umělá data tak, abych záměrně zvětšily šum ve vzorku a tím dále otestovali robustnost řešení.

Využijeme tedy 3 různé datové vzorky:

- Umělá lineární data

$$y_u = 2 * x + 12 \tag{7.1}$$

kde *x* je číslo náhodně zvoleno v intervalu (0,1)

- Umělá data s minimem šumu

$$y_{u+1} = 4 * sin(y_u) + 5 \tag{7.2}$$

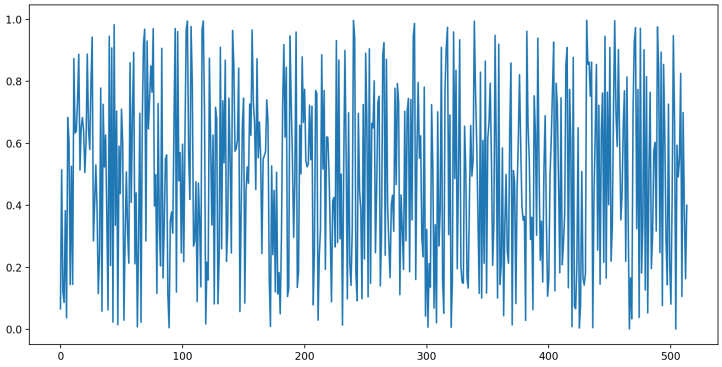
- Umělá data s větším množstvím šumu

$$y_{u+1} = 4 * sin(y_u) + 5 + x_{rand} \tag{7.3}$$

kde $x_{rand}$ je náhodně zvolené číslo z intervalu (0,1)

### 7.1.3 Náhodná data

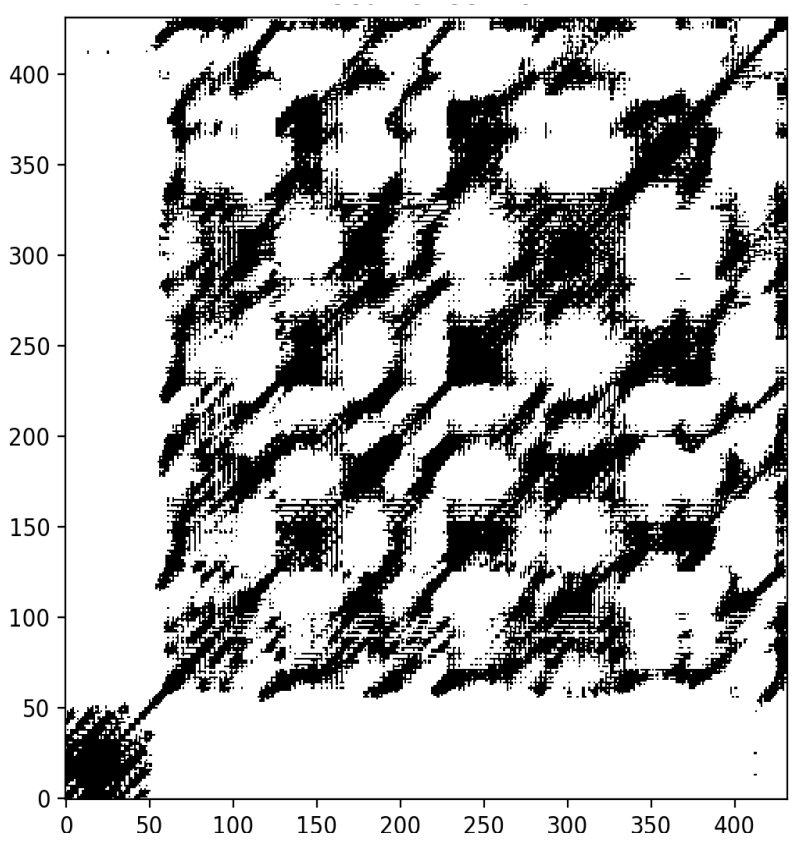
Náhodná data byla vytvořena za pomocí funkce, která vytváří náhodný datový vzorek ze zvoleného rozsahu. Využití tohoto vzorku je pouze pro ověření, že neuronová síť se neučí pouze na vstupní data[13]. Pokud bude tvorba NN úspěšná, měla by být schopnost predikce na této NN mizivá.




*Obr 7.3: Zobrazení náhodně vygenerovaného datového vzorku*

13 „overfitting“

## 7.2 Analýza reálných dat



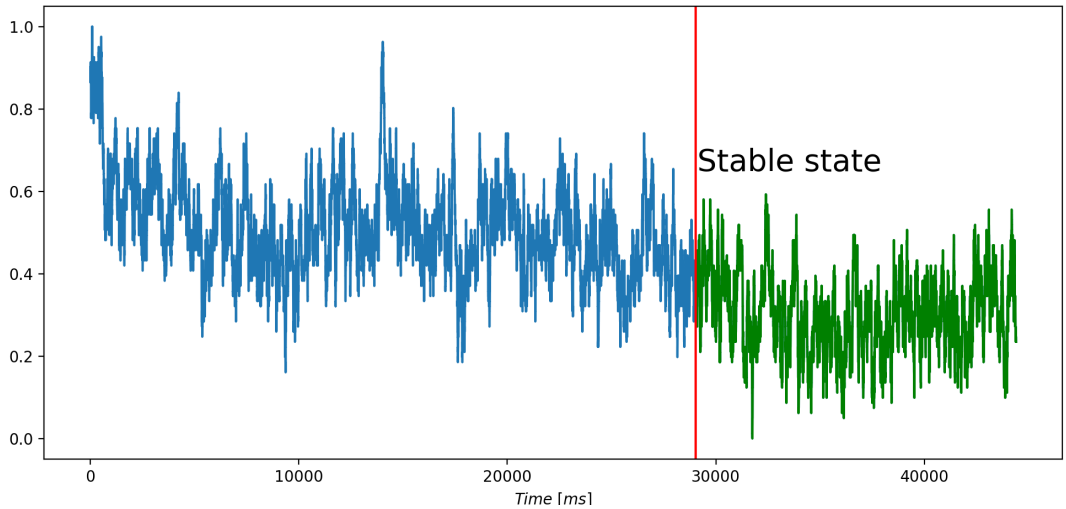

*Obr 7.4: Rekurentní zobrazení jedné z měřených teplot. Tmavější místa vyznačují podobnost mezi hodnotami v daném čase.*

Aby mohl algoritmus náležitě pracovat, je nutné nejprve analyzovat vstupní data. Jak již bylo zmíněno výše získaný datový vzorek obsahuje naměřené hodnoty od chvíle rozběhu motoru až po jeho ustálení. Abychom předešli komplikovanému učení algoritmu na neustálený stav, jenž by vnášel značné chyby do celého systému, bylo nutné zvolit od jakého časového úseku budeme data považovat za ustálená a tedy vhodná k využití pro náš algoritmus.

Pro grafické znázornění je na Obr.7.4 vidět vytvořený rekurentní graf jedné měřených hodnot. Tmavší místa na grafu znázorňují podobnost hodnot v čase. Z grafu je možné vyčíst, že největší podobnost hodnot je v oblasti levého horního kvadrantu, což odpovídá hodnotám přibližně poslední třetiny časové řady viz. Obr.7.5. Pro učení

našeho modelu a následné testování bude využito hodnot od $t = 293.07s$. Tato volba byla následně zkonzultována s odborníkem[14] přes turbmotory a schválena.

Za pozornost stojí zmínit, že i pohled na určenou množinu dat v graf naznačuje, že data jsou kolísavá, a tedy je možné předpokládat sníženou přesnost výsledku našeho modelu.




*Obr 7.5: Názorné zobrazení rozdělení dat na neustálený (modrá) a ustálený stav (zelená).*

## 7.3 Příprava dat – Data Preprocessing

V kapitole 2.3 již bylo naznačeno, že před tvorbou a učením NN je nutné alespoň minimálně připravit data pro jejich využití[30]. Většinou se jedná o metody, které zmenšují rozměr vstupních dat (*PCA*) případně je škálují (*standardizace dat*), abychom obdrželi bezrozměrné veličiny. Pokud nám v datovém vzorku některá data chybí můžeme je doplnit případně odstranit tzv. o*utliers* neboli vzorky, které svou hodnotou vysoce vybočují z ostatních. Správná volba metod a příprava dat nám může leckdy velice usnadnit následnou implementaci NN. V následující podkapitole jsou popsány metody přípravy dat, které byli využity na nám poskytnutá data.

### 7.3.1 Z-Scoring

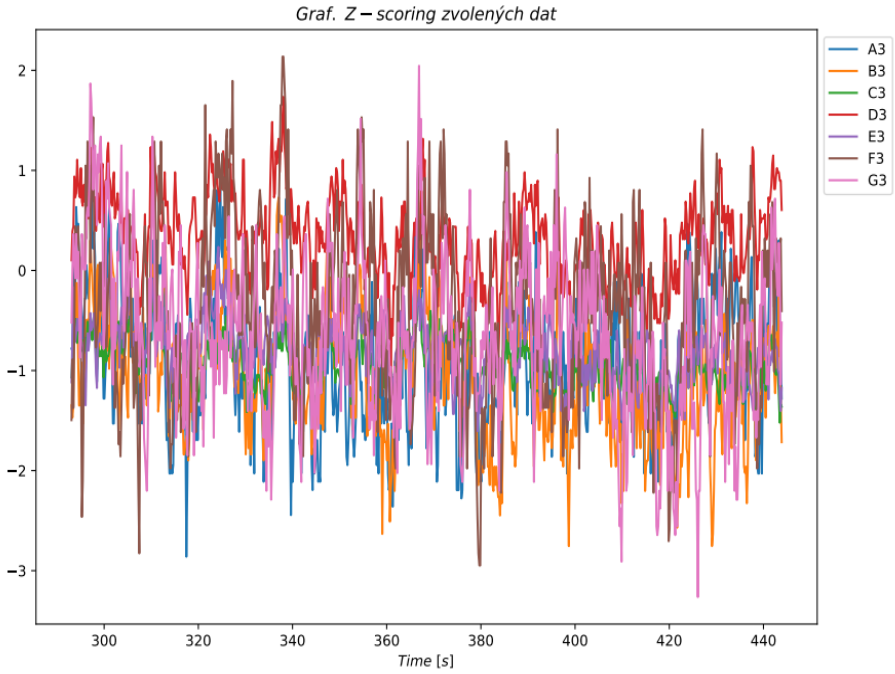
Původně naměřené hodnoty jsou přetransformovány, do takové podoby aby výsledné rozložení bylo o průměru 0 a standardizované odchylce 1. Výsledkem je

14 Ing. Michal Čížek

zjištění o kolik standardních odchylek, jsou jednotlivé data posunuty oproti průměru. Předpokladem pro tohoto postupu je normální rozdělení původních hodnot. Výpočet z-score hodnot vypadá takto:

$$z = \frac{x - \mu}{\sigma} \ , \tag{7.4}$$

kde $x$ je datový bod, $\mu$ je průměr hodnot a $\sigma$ je směrodatná odchylka. Pokud vychází hodnota z větší než 0, tak se jedná o vzorek nad průměrem a opačně pokud menší než 0 tak pod průměrem ostatních hodnot. Pokud by hodnota výrazně přesáhla $\pm 3$, jedná se o hodnotu, které by se mělo více věnovat ať už úpravou nebo odstraněním, jakožto outlieru.

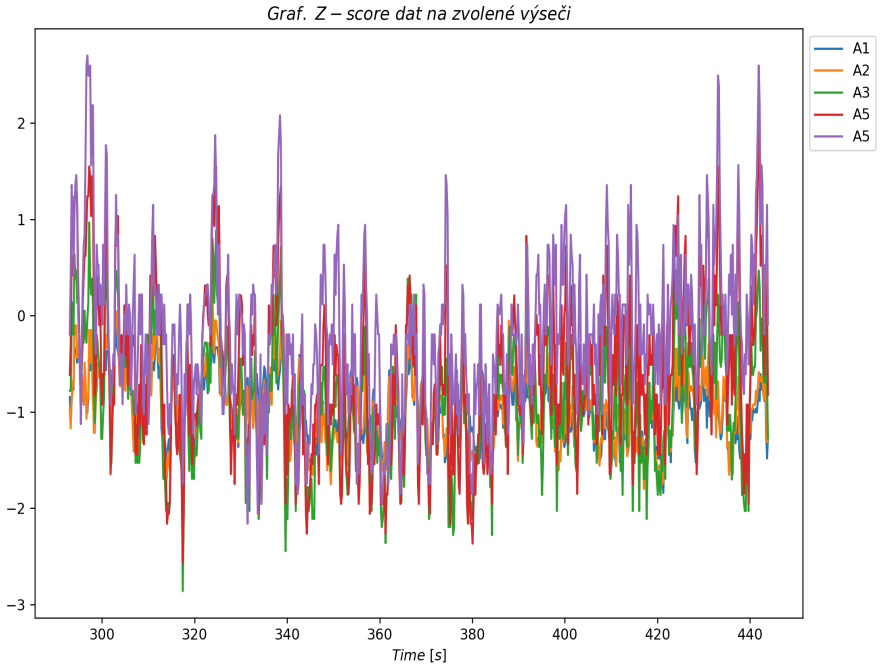



*Obr. 7.6: Z-Score dat na zvoleném průměru (C) vzorkování 20*

Na Obr. 7.6 vidíme data z již zmíněného středního průměru, na která byla aplikována metoda Z-Score. Pro lepší zobrazení bylo využito zobrazení po 20 datech. Lze si již na tomto grafu povšimnout, že u středního průměru data v průběhu času značně kolísají, ale mezi většinou ze 7 úhlů lze vidět podobný průběh a tedy i pravděpodobně společnou závislost.

Obr. 7.7 znázorňuje upravená data pouze na jedné výseči (A). Na tomto grafu je patrné, jak teploty na jedné výseči na různých průměrech kopírují podobný trend. Již od pohledu tedy můžeme usuzovat, že z jedné teploty bychom měli být schopni dopočítat ostatní.

Z obou grafů je patrné, že během sledované periody teplota kolísá a tudíž není mnoho hodnot, které by leželi blízko u průměru.



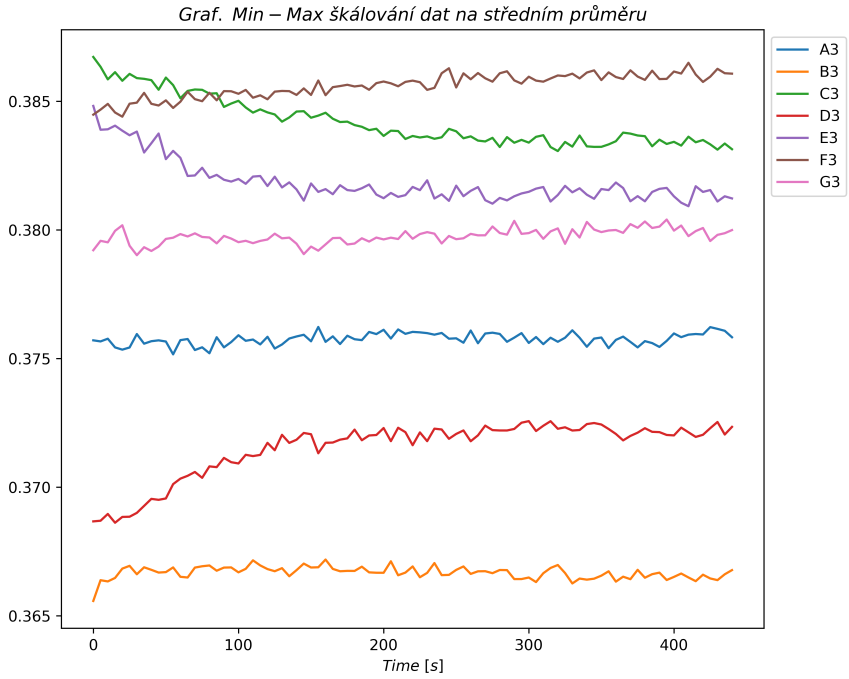

*Obr. 7.7: Z-Score dat na jedné výseči (A) vzorkování 20*

### 7.3.2 Normalizace dat

Metoda sloužící k přeškálování dat. S cílem aby data byla rozmístěna v intervalu [0,1] případně [-1,1] v závislosti na naší potřebě. Motivací pro tento proces je pro případy klasifikaci, či jiných aplikací, kde nás zajímá spíše vzdálenost mezi jednotlivými body než jejich vlastní hodnota. Existuje více druhů normalizace: Min-Max škálování, škálování na jednotkový vektor, normalizace podle průměru. Rovnice 7.5 popisuje normalizaci za využití Min-Max.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{7.5}$$

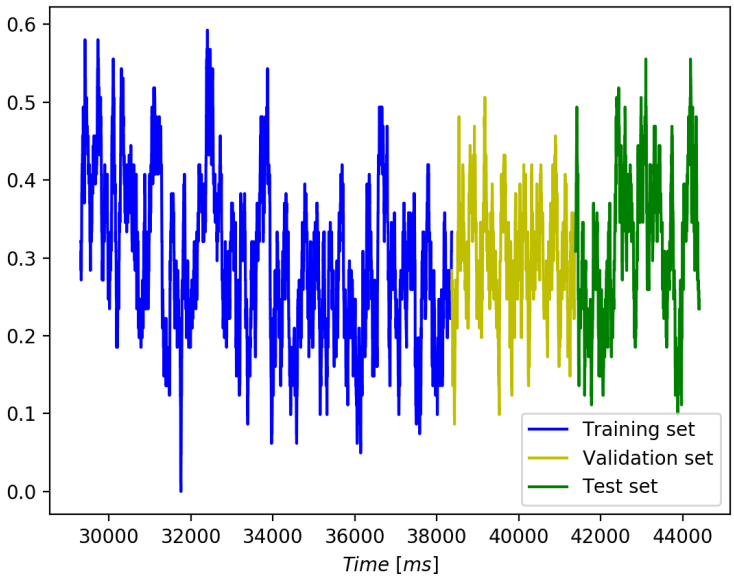
Na Obr. 7.8 vidíme graf upravených hodnot na středním průměru pomocí Min-Max škálování.




*Obr. 7.8: Graf normalizovaných dat na středním průměru*

### 7.3.3 Trénovací, testovací a validační sada dat

Po aplikaci normalizačních metod na data je dalším krokem jejich rozdělení do 3 sad. Jak jejich jednotlivé názvy napovídají slouží nám pro přípravu NN. Na **trénovací** sadě se NN naučí najít nejvhodnější nastavení vah a *biasu*, tak aby dosahovali požadovaných výsledků. Během trénování si model na **validační** sadě, která je části sady trénovací, ověřuje vývoj chybové funkce a upravuje učící rychlost na základě úspěšnosti. Po fázi učení testujeme model na **testovací** sadě, abychom zjistili jak dobře se dokázal naučit predikovat naše data.




*Obr 7.9: Rozdělení na trénovací validační a testovací sadu dat*

V této práci, je jak už bylo zmíněno výše, využito přibližně poslední třetiny poskytnutých dat. Tato třetina je z 66% využita jako trénovací sada z čehož je 20% použito jako validační a zbytek je je sadou testovací. Znázornění rozdělení dat lze vidět na Obr. 7.9.

## 7.4 Python, Keras

### 7.4.1 Python

Veške vytvořené grafy i tvorba algoritmu probíhá v programovacím jazyku Python[6]. Jedná se o vysokoúrovňový skriptovací jazyk. Jedná se o *open-source* projekt vytvořený Guido van Rossem v roce 1991. Existují různé implementace postavené na základech různých jazyků. Mezi nejznámější patří implementace na základech jazyka C – CPython, na základě jazyka Java – Jython a jazyka C# - IronPython. V dnešní době existují paralelně dvě verze Pythonu – 2.x 3.x. V následujích letech (cca. 2020) bude ukončena podpora verze 2.x a bude existovat pouze verze 3.x. Algoritmus v této práci je vyvíjen ve verzi 3.7.1. Jedna z předních výhod jazyka Python je jeho velká veřejná databáze dostupných knihoven a modulů vytvářených skupinami či jednotlivými uživateli, usnadňující vývoj pro ostatní.

### 7.4.2 Keras

Keras je vysokoúrovňové API[15] pro tvorbu neuronových sítí napsané v jazyce Python, které na pozadí využívá back- end z TensorFlow. Hlavní myšlenkou je urychlení vývoje při tvorbě neuronových modelů. To je dosaženo značným zjednodušením syntaxe pro tvorbu NN v Pythonu. Kde v originálním programování za pomocí TensoFlow je zapotřebí větších znalostí a pochopení matematických souvislostí, Keras umožňuje rychle vytvořit jednoduché NN modely, a následně je adaptovat do požadované podoby[31].

Keras umožňuje upravovat vytvořené modely na základě potřeby a dále využívat i ostatní externí knihovny. Hlavním problémem při tvorbě za pomocí Keras bylo v konečném důsledku pouze správná úprava dat před vložením do modelu, kdy bylo nutné dodržovat některé základní principy maticových výpočtů.

## 7.5 MLP

V následující podkapitole bude vytvořen optimální model NN za využití MLP pro aplikaci na získaná data. Bude zjištěno optimální nastavení parametrů pro učení NN a následné aplikování na datové sady.

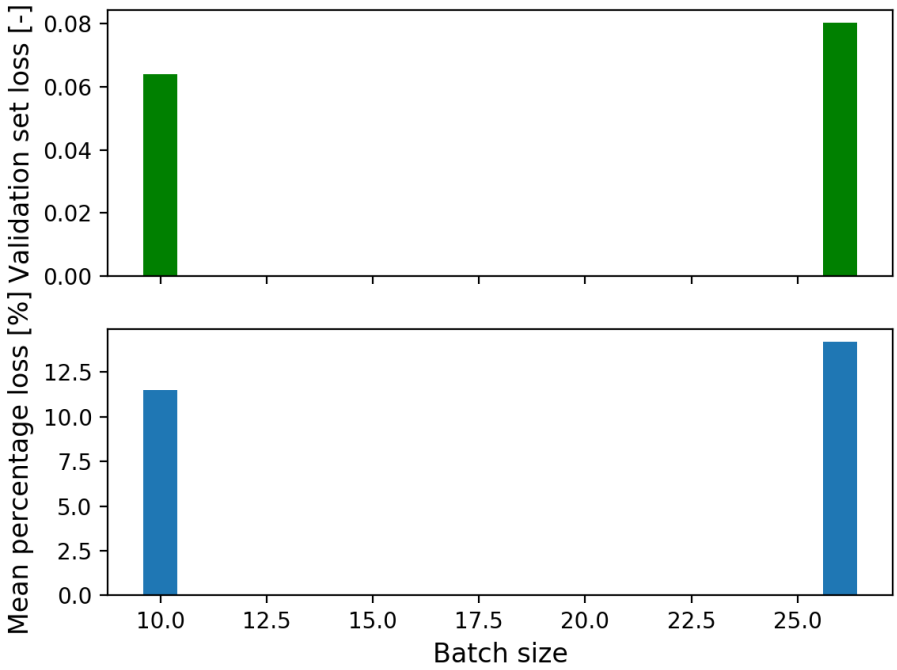
Pro tvorbu byla využita výše zmíněná knihovna Keras. neuronová síť takto vytvořená bude obsahovat 4 vrstvy – 1 vstupní, 2 skryté a 1 výstupní. Byly využity základní moduly knihovny Keras jako *Sequential* a *Dense*. Aby se zabránilo zbytečnému overfittingu byla při programování využito tzv. *Earlystopping*. Tato funkce sleduje vývoj chybové funkce na validační části dat a pokud po určitou dobu chybová funkce neklesá tak zastaví další průběh učení modelu. Dále tako jako optimalizační mechanismus bylo využito algoritmu *Adam*. Veškerý postup bude znázorněn pomocí grafů predikce jedné z teplot. Vyhodnocení bude probíhá pomocí MAE – *pruměrná absolutní chyba* (čím menší, tím lepší) jejíž výpočet vypadá takto:

$$MAE = \frac{\sum_{i=1}^{n} |y_{pred} - y_{real}|}{n} \tag{7.6}$$

15 Application Programming Interface

### 7.5.1 Velikost dávky

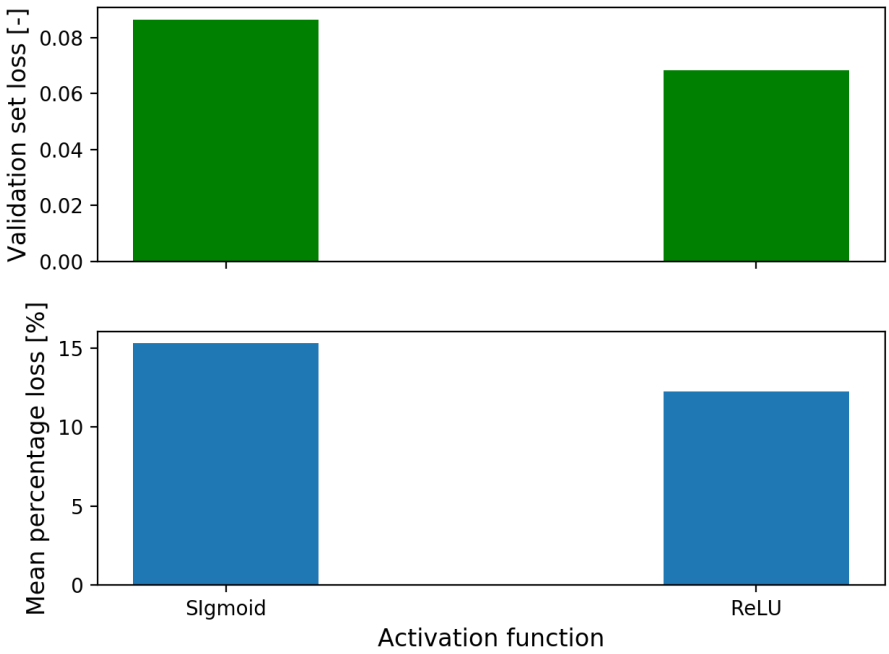
Změněním velikosti dávky umožňuje měnit podvzorek dat, na kterém se model snaží učit. Správně zvolená velikost dávky může výrazně ovlivnit výsledek predikce NN.




*Obr 7.10: Porovnání velikosti dávky 10 a 26. Nižší hodnota je lepší*

Z grafu na obr. 7.11 vidíme že dobrých výsledků dosahujeme při velikosti dávky do 10 a poté v bodě kdy je velikost dávky 26. Proto bude provedeno další porovnání pro tyto dvě hodnoty. Z obr. 7.10 vidíme, že nejlépe pro model MLP nám vychází velikost dávky 10. Tato bude tedy použita dále v této práci.

### 7.5.2 Typ aktivační funkce




*Obr 7.12: Rozdíl v přesnosti mezi aktivačními funkcemi. Menší hodnota je lepší*

Při tvorbě modelu se musíme rozhodnout o typu použité aktivační funkce. Bude se rozhodovat mezi *ReLU* a *Sigmoidou*. Rozdíly mezi těmito funkcemi byly popsány výše. Z grafu na Obr. 7.12 je znázorněn rozdíl přesnosti predikce mezi aktivačními funkcemi. Vychází nám z něj, že aktivační funkce *ReLU* dosahuje lepších výsledků. Pro další aplikaci tefy bude použita funkce **ReLU**

### 7.5.3 Shrnutí

Neuronový model použitý pro další aplikaci tedy vypadá následovně:

*Table 1: Specifikace MLP modelu*

| Počet skrytých vrstev | Počet neuronů ve vrstvě | Aktivační funkce | Velikost dávky | Optimalizační algoritmus |
|---|---|---|---|---|
| **2** | **32** | **ReLU** | **10** | **Adam** |

### 7.5.4 Otestování na umělých datech a náhodných datech

Poté co jsme zvolili ideální strukturu celého modelu, je nutné jej otestovat zda je model schopen najít závislosti na umělých datech a zároveň ověřit na náhodných datech, že se neučí pouze na poskytnutá data.

*Table 2: Výsledky aplikace na umělé datové vzorky*

| Kritérium přesnosti | Náhodná data | Lineární data | Umělá data | Umělá data se šumem |
|---|---|---|---|---|
| **MAE** | **99,84** | **0,001** | **0,03** | **0,08** |

Z tabulky 2 vyplývá, že vytvořený model je schopen úspěšně predikovat umělá data se známou závislostí a to i v případě vneseného šumu. Dále také nebyl schopen predikovat náhodná data, čímž je ověřena jeho funkčnost.

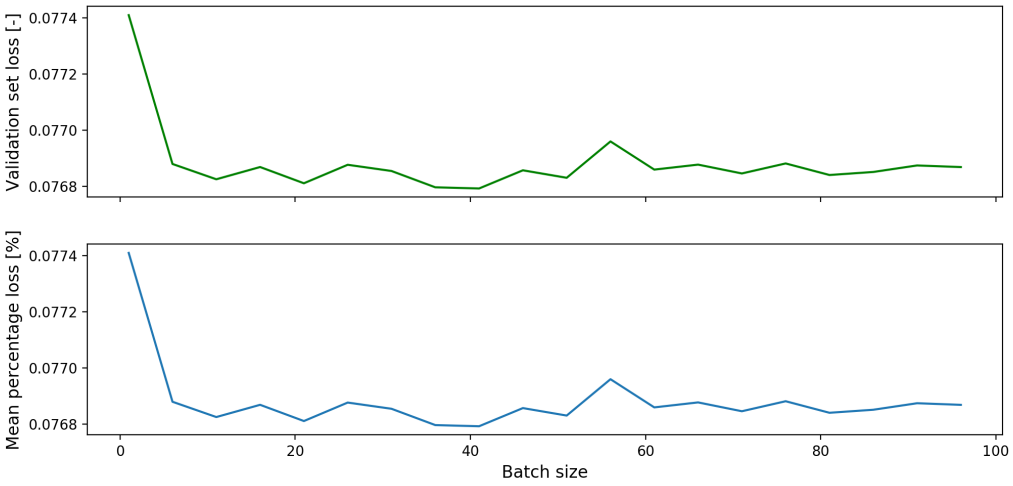
## 7.6 LSTM

V následující podkapitole bude vytvořen optimální model NN za využití LSTM pro aplikaci na získaná data. Bude zjištěno optimální nastavení parametrů pro učení NN a následné aplikování na datové sady.

Stejně jako u MLP je pro tvorbu tohoto modelu využito knihovny Keras. V této knihovně se nachází již předem připravená funkce přidávající do našeho modulu *Sequential* vrstvy neuronové sítě *LSTM*. V následujících odstavcích bude opět optimalizováno nastavení dávkování a aktivační funkce pro náš model. Opět to bude 4 vrstvý model – 1 vstupní vrstva, 2 skryté vrstvy LSTM a 1 výstupní vrstva. Stejně jako u MLP i zde bylo využito metod pro zabránění overfittingu. Pro vyhodnocení bude opět použito ***MAE***.

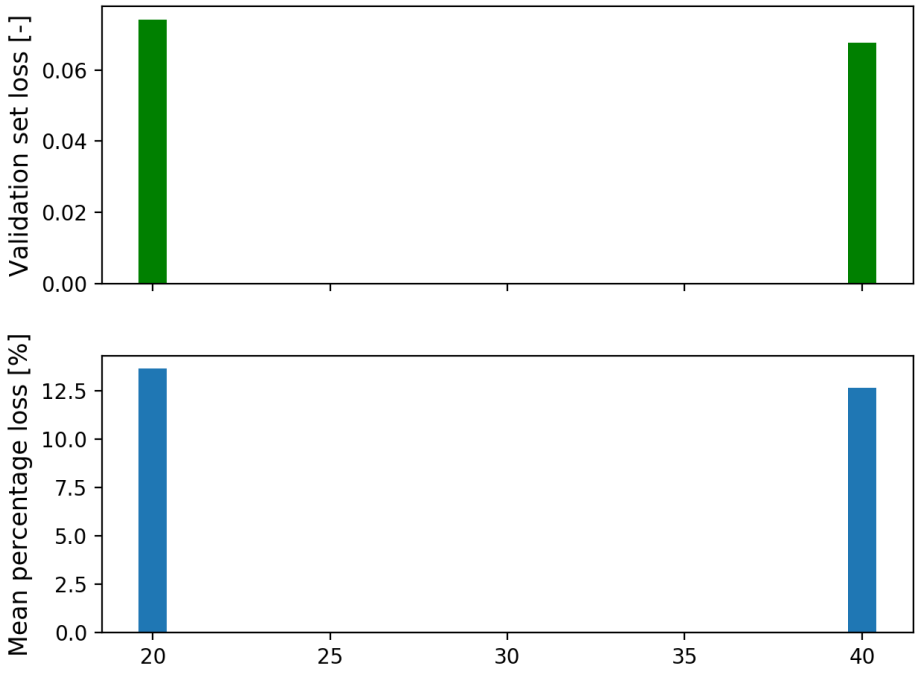
### 7.6.1 Velikost dávky

Pro zjištění ideální velikosti dávky pro model LSTM bude užito stejného postupu jako u MLP.




*Obr 7.13: Vývoj chybové funkce u LSTM při různé velikosti dávky*

Na Obr.7.13 vidíme že nejlepších výsledků dosahujeme při 20 vzorcích na dávku a při 40 vzorcích na dávku. Proto pro další zkoumání zvolíme jen tyto 2 možnosti.

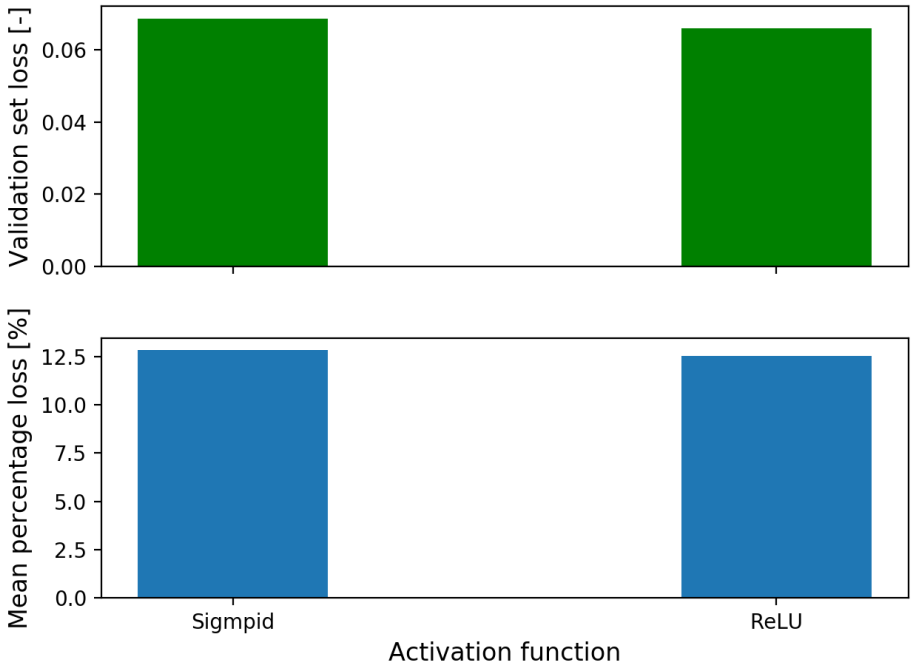



*Obr 7.14: Porovnání přesnosti dávek 20 a 40*

Z grafu na Obr.7.14 vidíme že přesnost velikosti dávek je takřka totožná a proto byla zvolena pro další implementaci velikost 20, která má větší potenciál v tom postihnout v sobě menší nuance v datovém vzorku zatímco u velikosti 40 bychom

mohli riskovat, že některé hodnoty budou zanedbány. Nevýhoda velikosti dávky 20 je, že si vyžádá více výpočetního času.

### 7.6.2 Typ aktivační funkce




*Obr 7.15: Rozdíl v aktivačních funkcí u LSTM*

Z grafu na Obr. 7.15 vidíme, že o něco lepší výsledky dosahuje algoritmus pokud je jako aktivační funkce použita ReLU. Ta je tedy použita dále v této práci, čímž poskytuje i výhodu menší náročnosti na výpočetní výkon.

### 7.6.3 Shrnutí

Neuronový model LSTM zvolený pro aplikaci na reálná data tedy vypadá následovně:

*Table 3: Shrnutí optimalního nastavení LSTM modelu*

| Počet skrytých vrstev | Počet neuronů ve skryté vrstvě | Aktivační funkce | Velikost dávky | Optimalizační algoritmus |
|---|---|---|---|---|
| **2** | **4** | **ReLU** | **20** | **Adam** |

### 7.6.4 Otestování na umělých datech a náhodných datech

Poté co jsme zvolili ideální strukturu celého modelu, je nutné jej otestovat zda je model schopen najít závislosti na umělých datech a zároveň ověřit na náhodných datech, že se neučí pouze na poskytnutá data.

*Table 4: Výsledky optimalizovaného modelu LSTM*

| Kritérium přesnosti | Náhodná data | Lineární data | Umělá data | Umělá data se šumem |
|---|---|---|---|---|
| **MAE** | **98,41** | **0,001** | **0,01** | **0,02** |

# 8 Aplikace algoritmů na experimentálním data

Poté co byly vytvořeny optimalizované modely MLP a LSTM, jsme připraveni aplikovat tyto algoritmy na reálný datový vzorek. Cílem v této kapitole je nalézt za pomocí naučení NN a vyhodnocení NN čidlo v řezu, z kterého jsme nejpřesněji schopni predikovat zbytek čidel.

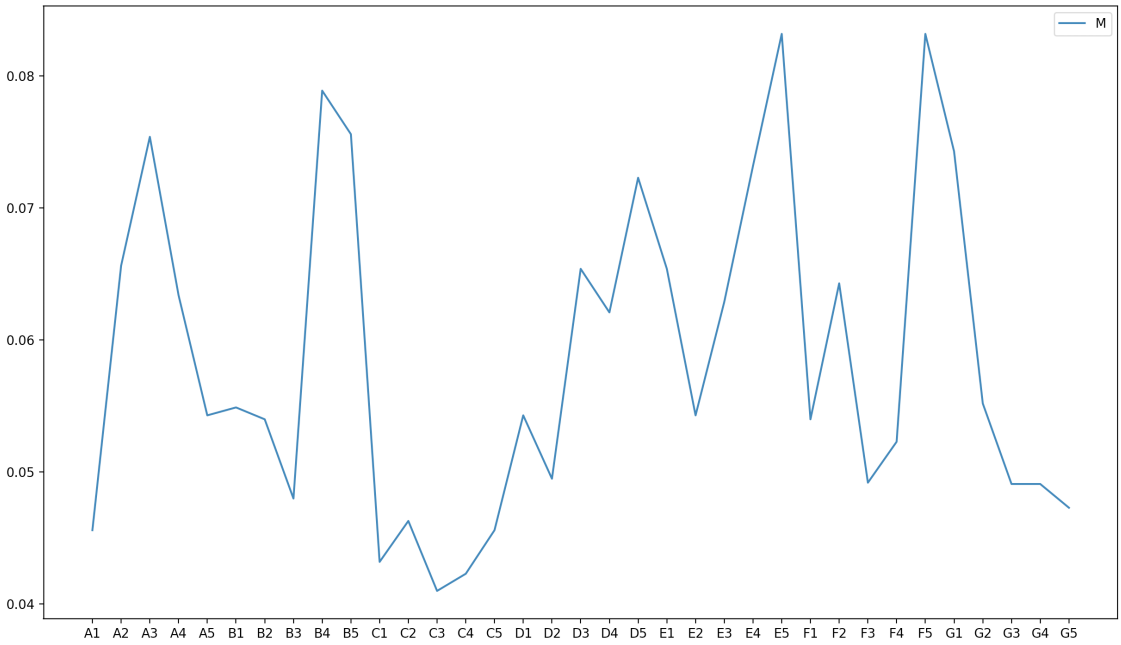
Program je nastaven tak, aby ve smyčce vždy zvolil jednu z teplot a pomocí předem připravené struktury NN modelu zjištěné v minulé kapitole zkusil predikovat ostatní teploty. Následně hodnotu úspěšnosti uložil do pole. Z tohoto pole bude následně vypočtena průměrná hodnota úspěšnosti predikce pro danou teplotu. Z ní zjistíme, jaká z teplot (čidel) se nejvíce hodí pro predikci ostatních.

## 8.1 MLP

Před připravenému modelu MLP, jsme poskytli veškerá nutná data pro tvorbu vnitřní struktury a naučení se predikovat naše hodnoty. Následně byl spuštěn cyklus, který v každém kroku nejprve zvolil jednu z teplot, a následně se pokusil naučit na predikci zbývajících 34. Po fázi naučení byla vytvořena predikce a porovnána s reálnou hodnotou pro všechny body každé z teplot. Průměrné absolutní odchylky (MAE) byly uloženy do pole a pokračovalo se další teplotou.

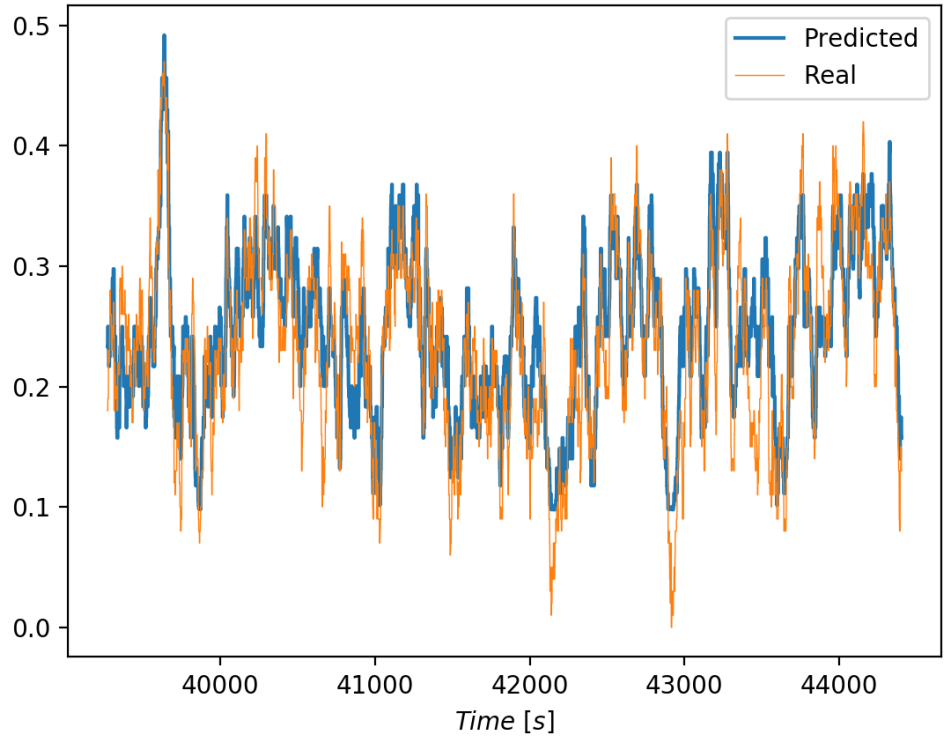
Z těchto odchylek pro každou teplotu byla následně vytvořena průměrná hodnota pro odchylku při predikci s využitím dané teploty. Tyto odchylky byly zapsány

do tabulky z níž pro lepší názornost byl vytvořen graf. Z něj lze vyčíst, že nejvhodenšjí teplota pro predikci je C3.



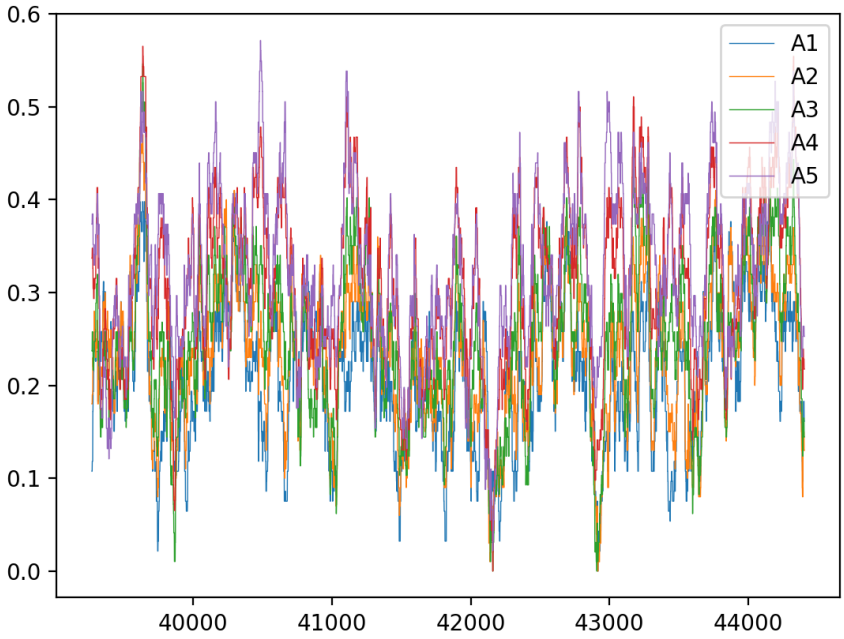

*Obr 8.1: Graf velikosti chyby v závislosti na využité teplotě pro predikci pomocí MLP*

### 8.1.1 Ukázky využití

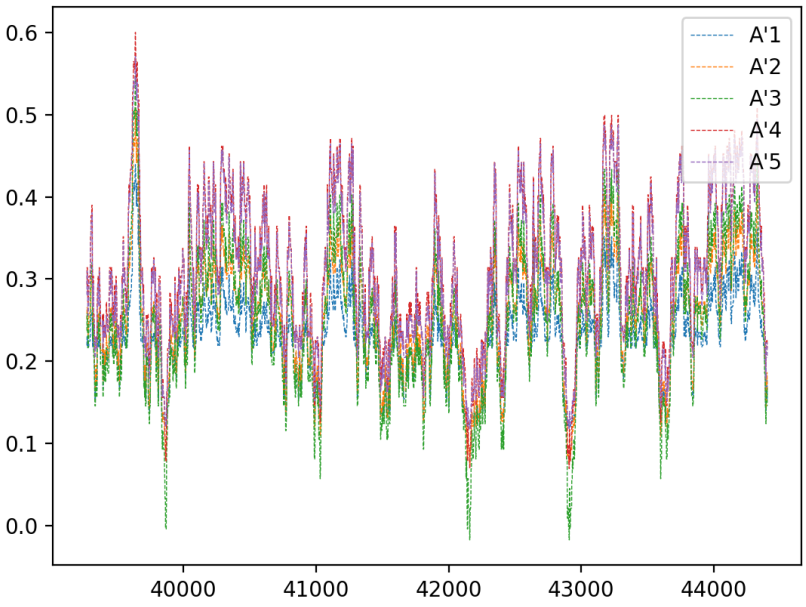



*Obr 8.2: Ukázka předpovědi vývoje teploty D2 z teploty C3*

Na Obr.8.2 vidíme ukázku předpovědi jiné teploty z námi určené jako vhodné pro predikci. Procentuální chyba této předpovědi je 12.8%. Na Obr.8.3 a 8.4 můžeme vidět srovnání predikování vývoje všech teplot na jednom úhlu za pomocí naučeného modelu pomocí teploty C3. Průměrná odchylka u predikce všech teplot z jedné byla 13%.




*Obr 8.3: Reálná data z úhlu (A) pro porovnání*

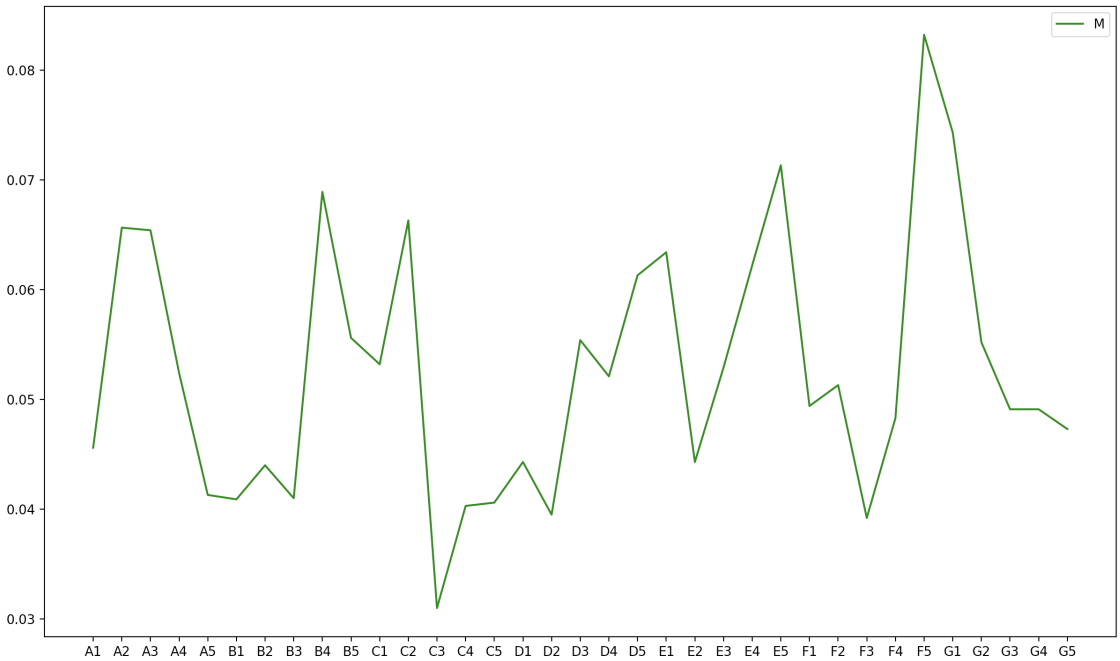



*Obr 8.4: Predikovaná data na úhlu (A) z teploty C3*

## 8.2 LSTM

Před připravenému modelu LSTM, jsme poskytli veškerá nutná data pro tvorbu vnitřní struktury a naučení se predikovat naše hodnoty. Následně byl spuštěn cyklus, který v každém kroku nejprve zvolil jednu z teplot, a následně se pokusil naučit na predikci zbývajících 34. Po fázi naučení byla vytvořena predikce a porovnána s reálnou hodnotou pro všechny body každé z teplot. Průměrné absolutní odchylky (MAE) byly uloženy do pole a pokračovalo se další teplotou.

Z těchto odchylek pro každou teplotu byla následně vytvořena průměrná hodnota pro odchylku při predikci s využitím dané teploty. Tyto odchylky byly zapsány do tabulky z níž pro lepší názornost byl vytvořen graf.

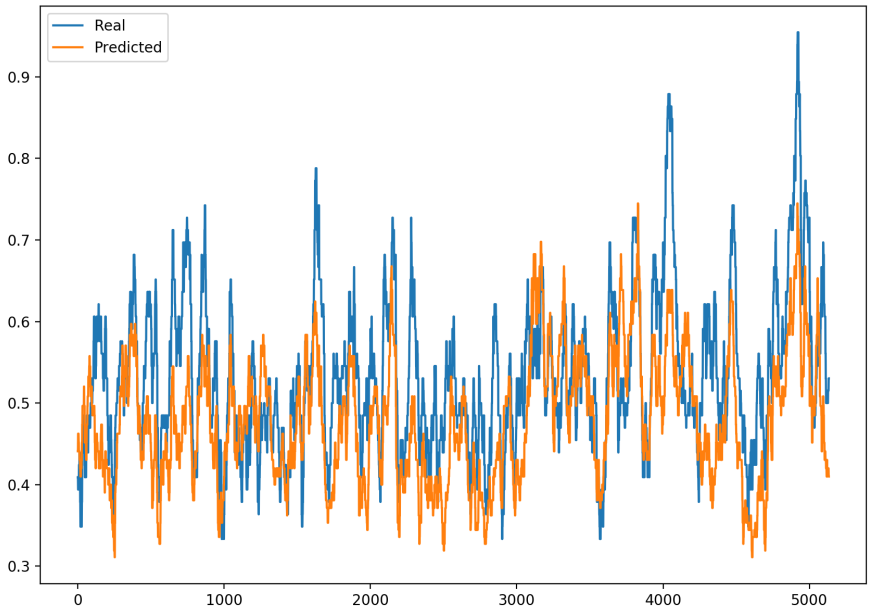



*Obr 8.5: Graf velikosti chyby v závislosti na využité teplotě pro predikci pomocí LSTM*

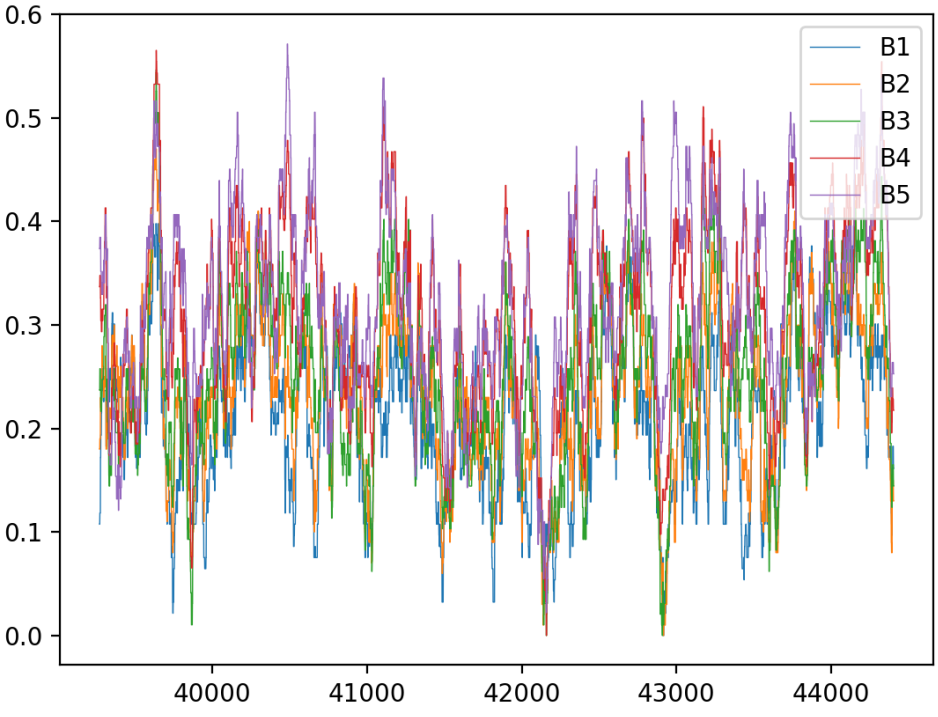
### 8.2.1 Ukázky využití

Na Obr. 8.6 vidíme předpověď teploty za pomocí vyhodnocené C3. U této teploty docházelo k větším rozdílům, ale i tak byla průměrná odchylka od reálného

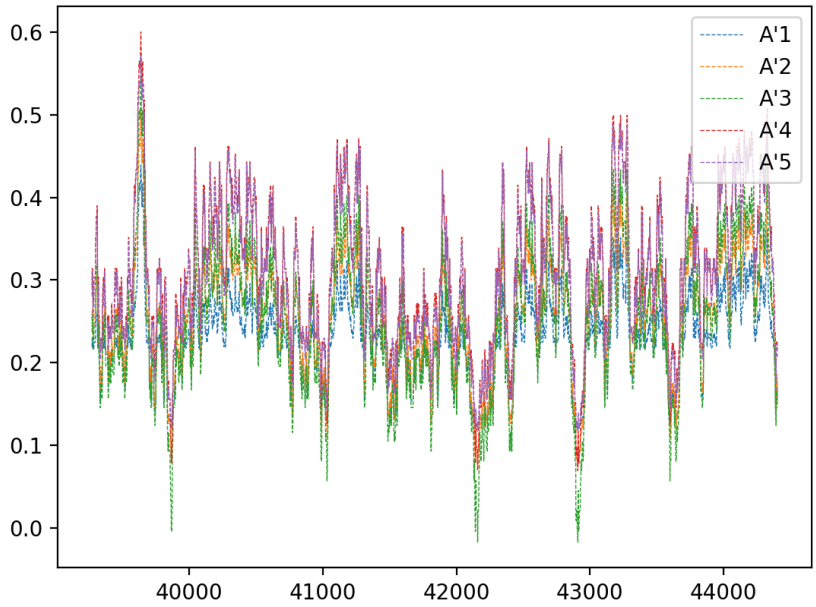
vzorku 12,1%. Na Obr.8.7 a 8.8 můžeme srovnat predikované teploty na jednom úhlu vůči reálným teplotám.




*Obr 8.6: Predikce jedné teploty pomocí LSTM za využít teploty C3*




*Obr 8.7: Reálné hodnoty teplot na úhlu (A)*




*Obr 8.8: Predikované teploty za pomocí LSTM*

## 8.3 Shrnutí

Z výsledných grafů vidíme, že pro predikci ostatních teplot se jako nejvhodnější jeví teplota označená jako C3. Dále také z výsledků vyplývá, že model LSTM je schopen dosahovat vyšších přesností, což bylo dopředu předpokládáno.

# 9 Závěr

V průběhu práce v kapitole 2 byl popsán vývoj a aktuální stav strojového učení. Následně byla provedena rešerše neuronových sítí v kapitole 3. Dále byly v kapitole 4 popsány některé neuronové modely – jejich specifikace, výhody a nevýhody a jejich matematické pozadí. K nim bylo také povrchně vysvětleno fungování některých z učících algoritmů.

V kapitole 6 bylo zevrubně naznačeno, jakým způsobem je možné získávat data z řezu turbomotoru, použitá senzorika a její zapojení pro experimentální potřebu. Což nám umožnilo schématicky naznačit rozmístění jednotlivých čidel (viz. Obr. 7.2).

Následně v kapitole 7, byla provedena analýza datových vzorků, na kterých byl aplikován vytvořený algoritmus. Byla popsána metoda přípravy těchto dat pro dosažení lepších výsledků. Na vytvořeném vzorku umělých dat, byly optimalizovány dva modely neuronových síti **MLP** a **LSTM** v programovacím jazyce Python. Tyto optimalizované modely byly otestovány na umělých datech, abychom potvrdili jejich funkčnost a na náhodných datech, abychom dokázali, že nedochází k overfittingu.

Tyto modely byly následně v kapitole 8 využity pro vyhledání nejvhodnějšího čidla pro předpověď dat z ostatních. Model byl vždy postupně učen na jednotlivé data a následně byla vypočtena průměrná úspěšnost u všech predikcí a z těch byla zvolena ta nejlepší. Dále také ukázána predikce pomocí této teploty.

Vhodným pokračováním této práce by bylo ověřit zda i v jiných řezech turbomotoru je dosahováno nejlepších výsledků v témže místě a případně vytvořit celistvou aplikaci s uživatelským rozhraním umožňující uživateli efektivní práci s nastavením modelů a zpracování výsledků. Dále také možnost využití jiných modelů neuronových sítí, obzvláště rekurentního stylu jako LSTM, jenž dosahují lepších výsledků díky jejich struktuře.

# Bibliografie

[1] . .HRONOVÁ, Markéta. Datový analytik jako povolání budoucnosti: Firmy se bez nich neobejdou, školy jich ale vychovávají málo. *Hospodářské noviny* [online]. 19. říjen 2016 [vid. 2019-04-07]. Dostupné z: https://byznys.ihned.cz/c1-65484080-datovy-analytik-jako-povolani-budoucnosti-firmy-se-bez-nich-neobejdou-skoly-jich-ale-vychovavaji-malo
[2] *Machine learning* [online]. 2019 [vid. 2019-04-06]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=890697907
[3] A Beginner's Guide to Multilayer Perceptrons (MLP). *Skymind* [online]. [vid. 2019-04-06]. Dostupné z: http://skymind.ai/wiki/multilayer-perceptron
[4] .....GUPTA, Madan M., Ivo BUKOVSKÝ, Homma NORIYASU, ASHU M. G. SOLO a ZENG-GUANG HOU. *Artificial Higher Order Neural Networks for Modeling and Simulation:* [online]. B.m.: IGI Global, 2013 [vid. 2019-04-05]. ISBN 978-1-4666-2175-6. Dostupné z: doi:10.4018/978-1-4666-2175-6
[5] HOCHREITER, Sepp a Jürgen SCHMIDHUBER. Long Short-Term Memory. *Neural Computation* [online]. 1997, **9**(8), 1735–1780. ISSN 0899-7667, 1530-888X. Dostupné z: doi:10.1162/neco.1997.9.8.1735
[6] Python Programming Language. *GeeksforGeeks* [online]. [vid. 2019-04-06]. Dostupné z: https://www.geeksforgeeks.org/python-programming-language/
[7] *GE H-Series Turboprop Engines* [online]. B.m.: GE Aviation. Dostupné z: https://www.geturboprops.com/cs/download/iauzszpp-h-series-data-sheet-12-2018-a4-preview.pdf
[8] …JAMES, Gareth, Daniela WITTEN, Trevor HASTIE a Robert TIBSHIRANI, ed. *An introduction to statistical learning: with applications in R*. 7th edition. New York: Springer, 2017. Springer texts in statistics, 103. ISBN 978-1-4614-7137-0.
[9] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016. Adaptive computation and machine learning. ISBN 978-0-262-03561-3.
[10] ….KURFÜRSTOVÁ, Jana. Strojové učení kouzla zbavené. *EDTECH KISK* [online]. 16. duben 2018 [vid. 2019-04-23]. Dostupné z: https://medium.com/edtech-kisk/strojov%C3%A9-u%C4%8Den%C3%AD-kouzla-zbaven%C3%A9-e066d79ebe51
[11] . ŠÍMA, Jiří a Roman NERUDA. *Teoretické otázky neuronových sítí*. Praha: Matfyzpress, 1996. ISBN 978-80-85863-18-5.
[12] . .NIELSEN, Michael A. Neural Networks and Deep Learning [online]. 2015 [vid. 2019-04-06]. Dostupné z: http://neuralnetworksanddeeplearning.com
[13] .....A Beginner's Guide to Deep Reinforcement Learning. *Skymind* [online]. [vid. 2019-04-25]. Dostupné z: http://skymind.ai/wiki/deep-reinforcement-learning
[14] . .VONDRÁK, Ivo. *Umělá inteligence a neuronové sítě: Určeno pro posl. 4. roč. Fak. elektrotechniky a informatiky*. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 1994. ISBN 978-80-7078-259-0.

[15] VOLNÁ, Eva. *NEURONOVÉ SÍTĚ 1* [online]. nedatováno. Dostupné z: http://www1.osu.cz/~volna/Neuronove_site_skripta.pdf
[16] FILIP MOLČÍK. *STUDIE VYUŽITÍ NEURONOVÝCH SÍTÍ A ADAPTIVNÍHO MONITOROVÁNÍ PRO ALGO-TRADING*. Praha, nedatováno. ČVUT v Praze.
[17] .....ŠTENCL, Ing Michael. Predikce a optimalizace reálných dat pomocí algoritmů umělé inteligence. nedatováno, 108.
[18] ..*(6) But what *is* a Neural Network? | Deep learning, chapter 1 - YouTube* [online]. [vid. 2019-04-06]. Dostupné z: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
[19] LIU, Danqing. A Practical Guide to ReLU. *TinyMind* [online]. 30. listopad 2017 [vid. 2019-05-07]. Dostupné z: https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7
[20] .....BROWNLEE, Jason. A Gentle Introduction to the Rectified Linear Unit (ReLU) for Deep Learning Neural Networks. *Machine Learning Mastery* [online]. 8. leden 2019 [vid. 2019-05-07]. Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
[21] HUANG, Guang-Bin, Qin-Yu ZHU a Chee-Kheong SIEW. Extreme learning machine: Theory and applications. *Neurocomputing* [online]. 2006, **70**(1–3), 489–501. ISSN 09252312. Dostupné z: doi:10.1016/j.neucom.2005.12.126
[22] TANG, Jiexiong, Chenwei DENG a Guang-Bin HUANG. Extreme Learning Machine for Multilayer Perceptron. *IEEE Transactions on Neural Networks and Learning Systems* [online]. 2016, **27**(4), 809–821. ISSN 2162-237X, 2162-2388. Dostupné z: doi:10.1109/TNNLS.2015.2424995
[23] BUKOVSKÝ, Ing Ivo. Nekonvenční neuronové architektury a jejich výhody pro technické aplikace. nedatováno, 33.
[24] BUKOVSKY, Ivo, Noriyasu HOMMA, Kei ICHIJI, Matous CEJNEK, Matous SLAMA, Peter M. BENES a Jiri BILA. A Fast Neural Network Approach to Predict Lung Tumor Motion during Respiration for Radiation Therapy Applications. *BioMed Research International* [online]. 2015, **2015**, 1–13. ISSN 2314-6133, 2314-6141. Dostupné z: doi:10.1155/2015/489679
[25] . .BROWNLEE, Jason. Multi-step Time Series Forecasting with Long Short-Term Memory Networks in Python. *Machine Learning Mastery* [online]. 9. květen 2017 [vid. 2019-04-08]. Dostupné z: https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/
[26] .....CHEN, Gang. A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation. *arXiv:1610.02583 [cs]* [online]. 2016 [vid. 2019-05-17]. Dostupné z: http://arxiv.org/abs/1610.02583
[27] *Levenberg-Marquardt backpropagation - MATLAB trainlm* [online]. [vid. 2019-05-20]. Dostupné z: https://www.mathworks.com/help/deeplearning/ref/trainlm.html
[28] .....VAJDÁK, Martin. *Metody měření teploty spalin před turbodmychadlem*. Brno, 2014. Bachelor. VÚT v Brně.
[29] .....NOVÁK, Martin, ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE a STROJNÍ FAKULTA. *Technická měření*. 2018. ISBN 978-80-01-06388-0.

[30] .....KUŹNIAR, Krystyna a Maciej ZAJĄC. Some methods of pre-processing input data for neural networks. nedatováno, 11.
[31] *Home - Keras Documentation* [online]. [vid. 2019-08-09]. Dostupné z: https://keras.io/
[32] .CULURCIELLO, Eugenio. The fall of RNN / LSTM. *Towards Data Science* [online]. 13. duben 2018 [vid. 2019-05-13]. Dostupné z: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0
[33] ……How does the Pratt & Whitney Canada PT6 differ from other turboprop engines? *Aviation Stack Exchange* [online]. [vid. 2019-07-24]. Dostupné z: https://aviation.stackexchange.com/questions/22151/how-does-the-pratt-whitney-canada-pt6-differ-from-other-turboprop-engines

# Seznam příloh

# Příloha 1 – Zdrojový kód třídy pro import dat

```
import re

import pandas as pd
from sklearn import preprocessing


class DataImport:
    """Class to imporrt data from csv file and split them into
separated variables

    """
    db = pd.DataFrame()
    temperatureDB = pd.DataFrame()
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))

    def splitdata(self, data):
        """Splits data into temperature DB and Time
        :param str data: CSV file format
        :return: tuple of Time DB and Temperature DB
        """
        DataImport.db = pd.read_csv(data)
        time = DataImport.db["TIME"]
        DataImport.temperatureDB = DataImport.db.iloc()[:, 1:]
        columns = DataImport.temperatureDB.columns
        scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
        DataImport.temperatureDB =
scaler.fit_transform(DataImport.temperatureDB.to_numpy())
        DataImport.temperatureDB =
pd.DataFrame(DataImport.temperatureDB, columns=columns)

        return time, DataImport.temperatureDB

    def azimut(self, name,base):
        """Returns specific azimut DataFrame.
        :param str name: Specification of concrete azimut we are
looking for.
        :return: DataFrame DB with conrete azimut
        """
        myregex = r"{base}" + re.escape(name) + r"."
        azimut = DataImport.temperatureDB.filter(regex=myregex)
        return azimut


    def radius(self, name, base):
        """Returns specific radius DataFrame.
        :param str name: Specification of concrete radius we are
looking for.
        :return: DataFrame DB with conrete radius
        """
        myregex = r"{base}" + r"." + re.escape(name) + "$"
        radius = DataImport.temperatureDB.filter(regex=myregex)
        return radius
```

# Příloha 2 – Zdrojový kód modulu pro tvorbu NN modelů

```
from keras.layers import Dense, LSTM
from keras.models import Sequential


class NeuralNetworks:
    """Class for creating specific NN models"""
    pass


def MLP_Regressor(units=(32, 64, 64), activation="relu",
loss_func="mae"):
    """Function for creating MLP regression model uisng keras
    :param str activation: Activation function ("relu", "elu",
"sigmoid")
    :param tuple units: Specifies number of neurons in 1st and other
layers, default (32.64.64)
    :param str loss_func: Specifies loss func for compilation, default
("mae")
    :return: Compiled model
    """
    regressor = Sequential()
    regressor.add(Dense(units=units[0], input_dim=1))
    regressor.add(Dense(units=units[1],activation=activation))
    regressor.add(Dense(units=units[2], activation=activation))
    regressor.add(Dense(1))
    regressor.compile(optimizer="adam", loss=loss_func,
metrics=["mape"])
    return regressor


def lstm_regresor(units=(10, 4, 4), activation="relu",
loss_func="mae"):
    """Function for creating MLP regression model uisng keras
        :param str activation: Activation function ("relu", "elu",
"sigmoid")
        :param tuple units: Specifies number of neurons in 1st and
other layers, default (32.64.64)
        :param str loss_func: Specifies loss func for compilation,
default ("mae")
        :return: Compiled model
        """
    module = Sequential()
    module.add(LSTM(units=units[0], input_shape=(1, 1),
return_sequences=True))
    module.add(LSTM(units=units[1], activation=activation,
return_sequences=True))
    module.add(LSTM(units=units[2], activation=activation))
    module.add(Dense(1))
    module.compile(optimizer='adam', loss=loss_func,
metrics=["mean_absolute_percentage_error"])
    return module
```

```
def mlp_multi(units=(68,68,68),activation = "relu",
loss_func="mae",output= 1):
    """Function for creating MLP regression model for whole dataset
uisng keras
        :param str activation: Activation function ("relu", "elu",
"sigmoid")
        :param tuple units: Specifies number of neurons in 1st and
other layers, default (32.64.64)
        :param str loss_func: Specifies loss func for compilation,
default ("mae")
        :param int output: Number of outputs from model
        :return: Compiled model
        """
    regressor = Sequential()
    regressor.add(Dense(units=units[0], input_dim=1))
    regressor.add(Dense(units=units[1], activation=activation))
    regressor.add(Dense(units=units[2], activation=activation))
    regressor.add(Dense(output))
    regressor.compile(optimizer="adam", loss=loss_func,
metrics=["mse"])
    return regressor
def lstm_multi(units = (34,4,4), activation = "relu", loss_func="mae",
output = 1):
    """Function for creating MLP regression model uisng keras
            :param str activation: Activation function ("relu", "elu",
"sigmoid")
            :param tuple units: Specifies number of neurons in 1st and
other layers, default (32.64.64)
            :param str loss_func: Specifies loss func for compilation,
default ("mae")
            :param int output: Number of outputs from model
            :return: Compiled model
            """
    module = Sequential()
    module.add(LSTM(units=units[0], input_shape=(1, 1),
return_sequences=True))
    module.add(LSTM(units=units[1], activation=activation,
return_sequences=True))
    module.add(LSTM(units=units[2], activation=activation))
    module.add(Dense(units=output))
    module.compile(optimizer='adam', loss=loss_func, metrics=["mse"])
    return module
```

# Příloha 3 – CD

- Elektronická verze práce
- Grafy
- Části kódu použitelné univerzálně