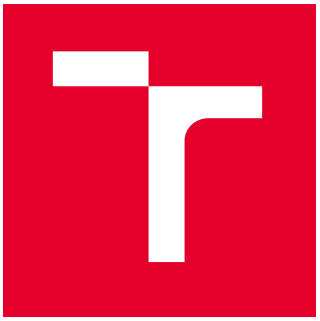


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

DLOUHODOBÉ PREDIKTIVNÍ MODELOVÁNÍ NELINEÁRNÍCH DYNAMICKÝCH SYSTÉMŮ POMOCÍ REKURENTNÍCH NEURONOVÝCH SÍTÍ

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

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Tomáš Pluskal

AUTHOR

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

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

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

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

Dlouhodobé prediktivní modelování nelineárních dynamických systémů pomocí rekurentních neuronových sítí

Stručná charakteristika problematiky úkolu:

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

Cíle bakalářské práce:

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

Seznam doporučené literatury:

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

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

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

V Brně, dne

L. S.

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

Abstrakt

Tato bakalářská práce se zabývá zkoumáním rekurentních neuronových sítí za účelem dlouhodobé predikce nelineárních dynamických systémů pomocí rekurentních neuronových sítí. Cílem je návrh a otestování softwarového řešení neuronové sítě na reálných datech pocházejících z měření teplot obráběcího stroje.

Summary

This bachelor thesis investigates recurrent neural networks for long-term prediction of nonlinear dynamic systems using recurrent neural networks. The aim is to design and test a neural network software solution on real data coming from machine tool temperature measurements.

Klíčová slova

strojové učení, rekurentní neuronové sítě, RNN, LSTM, GRU, TensorFlow

Keywords

machine learning, recurrent neural networks, RNN, LSTM, GRU, TensorFlow

Bibliografická Citace

PLUSKAL, T. *Dlouhodobé prediktivní modelování nelineárních dynamických systémů pomocí rekurentních neuronových sítí.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2023. 46 s., Vedoucí bakalářské práce: Ing. Jiří Kovář, Ph.D.

Prohlašuji, že předložená bakalářská práce je původní, vypracoval jsem ji samostatně a že jsem všechny prameny, ze kterých jsem čerpal, řádně uvedl v seznamu použité literatury.

Tomáš Pluskal

Brno . . . .

Tímto bych chtěl poděkovat svému vedoucímu Ing. Jiřímu Kovářovi Ph.D. za jeho rady při řešení této práce.

Dále bych chtěl poděkovat své rodině za podporu.

Tomáš Pluskal

Obsah

1 Úvod

Tato práce se zabývá průzkumem poznatků o rekurentních neuronových sítích, spadajících do oboru strojové učení, a jejich využitím za účelem dlouhodobé predikce dynamických nelineárních systémů s dosažením přijatelné přesnosti.

Dynamické systémy jsou takové systémy, jejichž stav v určitém časovém bodu závisí na vnějších i vnitřních vlivech, stejně jako na předešlých stavech daného systému. Jedná-li se o reálnou dynamickou soustavu, není v takovém případě mnohdy jednoduché vytvořit dostatečně přesný matematický model. Řešením může být využití strojového učení, pomocí kterého lze na základě znalosti předešlých a současných průběhů pozorovaných vlastností předpovídat jejich další vývoj, bez nutnosti znát jejich matematický popis.

Sledovanou soustavou je pro účely této práce obráběcí stroj. Jeho monitorováním a předpovídáním kritických stavů s předstihem je v průmyslu možné zvýšit efektivitu výroby, ale také omezit zvýšené náklady spojené s nastáním kritického stavu, jak na samotném obráběcím stroji, tak na obráběné součásti, a také omezit ztráty způsobené přerušením činnosti stroje. V této práci budou pro predikci použita data z teplotních senzorů.

2 Rešerše

2.1 Strojové učení

Strojové učení (*machine learning*) je podoblastí umělé inteligence (*artificial intelligence*), oboru informatiky zabývajícím se tvorbou počítačových modelů se snahou napodobit lidské myšlení, rozhodování a chování a se schopností se vyvýjet a učit z nabytých zkušeností. Samotné strojové učení se potom zabývá tvorbou počítačových modelů sloužících k interpretaci dat, zpravidla ke klasifikaci nebo regresi, na základě souboru dat, pomocí kterého byl model trénován. [1] To umožňuje (zejména v případě složitějších, tzv. „hlubokých“, modelů) řešit úlohy, jejichž řešení by jinak pouze algoritmicky bylo buď nemožné, nebo by alespoň bylo velmi obtížné na popis.

Různými typy modelů strojového učení jsou např. lineární regrese, logistická regrese, rozhodovací stromy, metoda podpůrných vektorů (*support vector machine*) a algoritmus k-nejbližších sousedů a neuronové sítě. [2, s. 9]

2.1.1 Typy strojového učení

Existuje několik různých způsobů, jakými lze modely strojového učení trénovat. Jejich výběr je ovlivněn zejména podobou souboru dat, pomocí kterého má být model trénován, ale také i volbou samotného trénovacího algoritmu. [3]

Učení s učitelem (*supervised learning*)

Trénovací algoritmy spadající do kategorie učení s učitelem vyžadují anotovaná data, to znamená, že se vzorek dat skládá z dvojic vstupních, které se nazývají vektory vlastností, a k nim odpovídajících výstupních hodnot, tzv. anotací. Cílem procesu učení je potom naučit model na vybraném vzorku dat vztahy mezi vstupy a výstupy tak, aby libovolná nová vstupní data dokázal na základě těchto závislostí vhodně generalizovat na odpovídající výstupy. [4, s. 4]

Učení s učitelem je výhodné zejména v situacích, kde existuje jasný vztah mezi vstupy a výstupy, a kde existuje dostatek reprezentativních a vhodně anotovaných dat. Obstarat vyhovující vzorek dat může být ale v mnoha případech jak časově, tak finančně náročné. V případě, že není možné takový vzorek dat obstarat, není možné učení s učitelem použít.

Tento typ učení se používá zejména pro klasifikaci a regresi.

Učení bez učitele (*unsupervised learning*)

Algoritmy učení bez učitele používají ke trénování modelů neanotovaná data. Cílem tohoto učení je tedy nalézt závislosti mezi jednotlivými prvky vektoru vlastností během procesu trénování. [5, s. 4]

Tento typ učení je výhodný zejména v situacích kdy je dostupné velké množství dat, které by ale bylo časově i finančně velmi náročné anotovat, případně v situacích, kdy je velmi obtížné popsat vlastnosti vstupních dat.

Takto naučené modely se používají zejména ke klastrování, detekci anomálií, vizualizaci dat na základě pozorovaných vlastností, redukci dimenzí vstupních dat a učení asociačních pravidel. Z důvodu absence anotací v souboru dat není možné modely učené tímto způsobem využít k regresi. Další nevýhodou může být to, že výstupní data mohou v některých případech vyžadovat další zpracování.

Částečné učení s učitelem (*semi-supervised learning*)

Tento typ učení je kombinací učení s učitelem a učení bez učitele. Soubor dat obsahuje větší množství neoznačených vektorů vlastností a pouze malé množství jich je s anotacemi. Algoritmus potom během trénování dokáže seskupit neoznačená data na základě společených znaků a následně k jednotlivým skupinám přiřadit anotaci. Výhodou je zejména to, že je možné tímto způsobem dosáhnout výsledků srovnatelných s učením s učitelem, bez nutnosti obstarat kompletně anotovaný soubor dat. Nutností ale je, aby byla anotována alespoň část dat reprezentující všechny možné charakteristické skupiny obsažené ve vektoru vlastností. [5, s. 4]

Zpětnovazební učení (*reinforcement learning*)

Zpětnovazební učení je druh strojového učení, u kterého není potřeba mít před započetím trénování modelu nashromážděná data. Učící se systém se nazývá agent a vykonává akce (zpočátku náhodné po inicializaci parametrů modelu), za které je odměňován nebo penalizován dle stanovených pravidel, jež je před započetím trénování nutné definovat. Tímto iterativním způsobem se agent snaží maximalizovat možnou odměnu. [2, s. 609]

Toto učení je výhodné pro aplikace, u kterých je velmi složité popsat celé, mnohdy měnící se, prostředí, ale naopak je možné vytvoření jasných pravidel na základě žádaných výstupů. Zpětnovazební učení se často používá v robotice, při učení modelů hrát hry, nebo při vytváření kontrolních systémů.

2.1.2 Příprava souboru dat před učením

Před započetím tvorby a trénování modelu je nezbytné obstarat trénovací data. Jelikož surová naměřená data mohou obsahovat plno chyb, množství nerelevantních informací apod., je nutné taková data nejprve pročistit a následně transformovat.

Čištění dat zahrnuje:

- odstranění chybějících dat,
- opravení chybných dat
- odstranění duplikovaných dat,
- odstranění nerelevantních dat. [6]

Data mohou mít vlivem jiného měřítka rozdílnou váhu, což snižuje výkon a stabilitu modelu. V takovém případě je nutné ještě provést transformaci dat. [7]

Normalizace slouží k transformaci dat do jednotného měřítka škálováním na rozsah daný minimální a maximální hodnotou v souboru dat. [8] Popsána je rovnicí (2.1).

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{2.1}$$

Standardizace slouží k transformaci dat na normální rozdělení. Pro použití této metody nesmí být v souboru dat příliš mnoho odlehlých hodnot. [9] Popsána je rovnicí

$$x' = \frac{x - \mu}{\sigma}, \tag{2.2}$$

kde μ je průměrná hodnota dat a σ je směrodatná odchylka.

Oříznutí slouží k odstanění příliš odlehlých hodnot v souboru dat nahrazením všech hodnot mimo stanovený rozsah jeho hraničními body. [7]

2.2 Umělé neuronové sítě

Jak již název napovídá, vznik umělých neuronových sítí jako matematických modelů byl původně inspirován nervovou soustavou živočichů. I přesto, že se obě skládají ze vzájemně propojených neuronů, umělé neuronové sítě jsou pouze modelem a nejde je tedy z funkčního hlediska považovat za podobné např. lidskému mozku. [2, s. 279-282]

Neuronové sítě jsou zpravidla složitějšími, tzv. hlubokými, modely strojového učení, v nichž jsou neurony seskupeny po vrstvách, které mouhou mít podle použití sítě libovolný počet neuronů. Samotný neuron je potom nejmenší funkční jednotkou. Jeho podoba závisí na použité architektuře neuronové sítě. Aby mohl být model klasifikován jako hluboký, musí obsahovat alespoň tři vrstvy. Souhrnně se obor zabývající se hlubokými umělými neuronovými sítěmi nazývá hluboké učení (*deep learning*).

První vrstva se nazývá vstupní a vstupuje do ní vektor vlastností. Tato vrstva je nezbytná a počet neuronů v ní je shodný s počtem pozorovaných vlastností. Následuje jedna nebo více skrytých vrstev, ve kterých probíhají hlavní výpočetní operace sloužící k transformaci dat vstupních na výstupní a k identifikaci pozorovaných vlastností. Každá skrytá vrstva může mít libovolný počet neuronů. Cílem návrhu neuronové sítě je ale zvolit takový počet neuronů a vrstev, který bude dosahovat nejlepších výsledků. Poslední vrstvou je výstupní vrstva, jež slouží k reprezentaci výstupních dat a jejíž počet neuronů se shoduje s počtem žádaných výstupů. Výstupní vrstva je opět nezbytná.

2.2.1 Architektury neuronových sítí

Existuje mnoho různých architektur neuronových sítí a jejich variací, lišících se zejména svou strukturou a způsobem použití.

Dopředné neuronové sítě (*feed-forward neural networks*)

Dopředné neuronové sítě (FFNN) jsou nejzákladnějším a nejjednodušším typem neuronových sítí. Vyznačují se tím, že v nich informace putuje pouze jedním směrem, a to směrem od vstupní k výstupní vrstvě. Nejčastěji jsou využívány pro klasifikaci a regresi. [2, s. 289]

Rekurentní neuronové sítě (*recurrent neural networks*)

Hlavním rozdílem rekurentních neuronových sítí (RNN) oproti dopředným neuronovým sítím je, že rekurentní neuronové sítě dokáží zohlednit vliv předešlých vstupů na ty následující. Jejich využití je proto vhodné zejména ke zpracovávání sekvenčních dat, která mohou být anotována, klasifikována nebo na základě nich mohou být generovány nové sekvence. Příkladem sekvenčních dat mohou být například řetězce textů nebo časové řady. [10]

Konvoluční neuronové sítě (*convolutional neural networks*)

Konvoluční neuronové sítě (CNN) byly specificky navržené pro zpracování obrazu, je ale možné je použít také na rozpoznávaní hlasu, zpracovávání přirozeného jazyka, regresi aj.

Tyto neuronové sítě se skládají z konvolučních vrstev, jež jsou složeny z filtrů (masek), které, v případě zpracování obrazu, vždy zpracovávají pouze určitý sektor obrazu a postupně se po něm posouvají. Jednotlivé filtry slouží k detekci různých znaků, např. hran nebo textur. [11]

2.2.2 Vícevrstvý perceptron (*multilayer perceptron*)

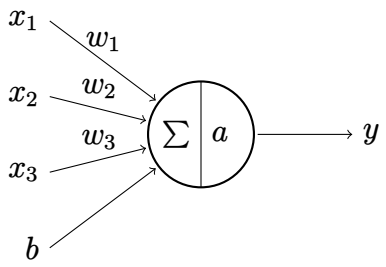
Nejjednodušším možným typem neuronové sítě je perceptron. Jedná se o dopřednou neuronovou síť o jedné vrstvě s jedním neuronem, která byla vyvinuta v 50. letech 20. století. [12]

Perceptron přijímá jako vstup vektor vlastností $\mathbf{x}$, který je vynásoben vektorem vah $\mathbf{w}$. K tomuto váženému součtu je následně přičten práh $\mathbf{b}$ (*bias*). Tento součet je potom vstupem do aktivační funkce a, jejíž význam bude podrobněji popsán v následující podkapitole. Výstup z

aktivační funkce je skalár a je zároveň i výstupem perceptronu. Každý perceptron má pouze jeden výstup. Jednotlivé váhy (u naučeného modelu) udávají, jak velký vliv má konkrétní vstup na žádaný výstup, přičemž práh stanovuje hranici, při které má být neuron aktivován. U složitějších neuronových sítí s více neurony udává práh význam celého neuronu na žádaný výstup. [13]

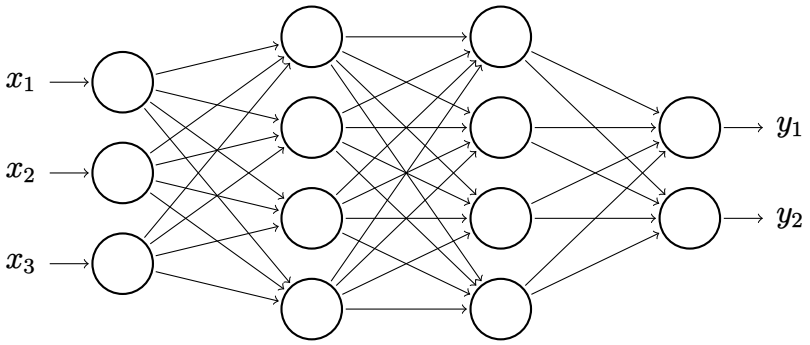
Jeho struktura je znázorněna na obrázku 2.1 a matematický popis je vyjádřen rovnicí

$$y(\mathbf{x}) = a(\mathbf{w}\mathbf{x} + \mathbf{b}). \tag{2.3}$$



Obrázek 2.1: Struktura perceptronu

Samotný perceptron kvůli své jednoduchosti nemá v praxi významné využití. Jeho uspořádáním do vrstev je však možné vytvářet složitější struktury zvané vícevrstvé perceptrony (*multilayer perceptron*). Struktura vícevrstvého perceptronu je znázorněna na obrázku 2.2. [13, s. 129] Spojení mezi jednotlivými neurony se nazývají synapse.



Obrázek 2.2: Struktura vícevrstvého perceptronu

Matematický popis jedné vrstvy vícevrstvého perceptronu je dán rovnicí

$$\mathbf{y}^{(i)}(\mathbf{x}) = a(\mathbf{W}^{(i)}\mathbf{x}^{(i)} + \mathbf{b}^{(i)}), \tag{2.4}$$

přičemž i je index vrstvy, $\mathbf{W}$ je matice vah, $\mathbf{x}$ je vektor vstupů (shodný s vektorem výstupů předchozí vrstvy $\mathbf{y}^{(i-1)}$), $\mathbf{b}$ je vektor prahů, a je aktivační funkce a $\mathbf{y}$ je vektor výstupů.

2.2.3 Aktivační funkce (*activation function*)

Aktivační funkce slouží k určení výstupu neuronu, jehož hodnota reprezentuje jeho aktivaci.

Existuje mnoho různých druhů aktivačních funkcí, jejichž volbou lze mimo jiné ovlivnit, jestli bude neuronová síť sloužit k regresi nebo klasifikaci (volba ve výstupní vrstvě). Podmínkou je, aby funkce byla spojitá a hladká, respektive alespoň po částech hladká, tedy aby existovala její derivace v celém definičním oboru, respektive vyjma jednotlivých bodů.

Aktivační funkce také bývá zpravidla nelineární. Jelikož jejím vstupem je lineární funkce (posunutý vážený součet vstupů), jakákoliv kombinace lineárních funkcí, v po sobě jdoucích vrstvách, by byla stále lineární funkcí. Důsledkem by bylo, že by ke všem těmto vrstvám existovala jedna ektivalentní vrstva. Všechny následující vrstvy by tedy byly redundantní a model by nedokázal aproximovat složitější a nelineární úlohy, které jsou v praxi velmi časté. [14]

Sigmoid

Sigmoid (také zvaný logistická funkce) slouží k transformaci vstupů z definičního oboru $\mathbb{R}$ do intervalu $(0; 1)$. Díky tomu je u výstupní vrstvy výhodný zejména pro binární klasifikační úlohy, ve kterých výstupní hodnota z intervalu $(0; 1)$ představuje pravděpodobnost sledovaného jevu. [15]

Sigmoid byl dříve hojně využíván, jelikož se nejvíce podobá způsobu aktivace neuronů v lidském mozku, později byl ale překonán vlastnostmi nově popsaných aktivačních funkcí.

Matematicky je dán rovnicí

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

Hyperbolický tangens

se tvarem podobá funkci sigmoid (zároveň je jeho lineární transformací) s tím rozdílem, že jeho výstupní hodnoty jsou z intervalu $(-1; 1)$. [15]

Jeho funkční předpis je dán rovnicí (2.6)

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{2.6}$$

Díky tomu, že je střední hodnota průběhu této funkce nulová, je možné během trénování neuronových sítí rychleji dosáhnout konvergence.

ReLU (*rectified linear unit*)

ReLU je jednou z aktivačních funkcí, která není derivovatelná v celém svém definičním oboru, konkrétně v bodě 0. Díky jejím výhodám, jako jsou zejména nízká náročnost na výpočet, a díky tomu, že není saturována pro kladné vstupy, se stala jednou z nejpoužívanějších aktivačních funkcí. [15]

Její matematický popis je dán rovnicí

$$\text{ReLU}(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0. \end{cases} \tag{2.7}$$

Nevýhodou této funkce je, že způsobuje tzv. „umírání“ neuronů v případě, že je vážený součet vstupů neuronu záporný. Výstup ReLU je potom vždy roven nule a daný neuron se přestává podílet na výstupu neuronové sítě. Z toho důvodu byly vytvořeny její variace „leaky“ ReLU a ELU, které tento problém řeší.

„Leaky“ ReLU

„Leaky“ ReLU vychází z ReLU, pouze s tím rozdílem, že pro vstupní hodnoty menší než nula

je výstupem lineární funkce se sklonem daným hyperparametrem[1] α, čímž eliminuje problém tzv. „umírajících“ neuronů. Stále je ale pouze po částech hladkou funkcí. [15]

Její funkční předpis je dán rovnicí

$$\text{„Leaky“ ReLU}(x) = \begin{cases} \alpha x, & x \leq 0 \\ x, & x > 0. \end{cases} \tag{2.8}$$

ELU (*exponential linear unit*)

ELU opět vychází z ReLU, s tím rozdílem, že pro záporné vstupní hodnoty je použit exponenciální člen. Kvůli tomu je výpočtově náročnější. Na druhou stranu ale díky svým vlastnostem umožňuje rychlejší učení modelu. [15]

Její funkční předpis je dán rovnicí

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x, & x > 0. \end{cases} \tag{2.9}$$

Její výhodou je, že pro hyperparametr $\alpha = 1$ je hladkou funkcí.

Jelikož platí, že

$$\lim_{x \to -\infty} \alpha(e^x - 1) = -\alpha, \tag{2.10}$$

hyperparametr α také určuje hodnotu dolní saturace.

Průběhy všech výše zmíněných aktivačních funkcí jsou znázorněny na obrázku 2.3.

2.2.4 Rekurentní neuronové sítě

Narozdíl od dopředných neuronových sítí, ve kterých závisí výstupní hodnoty výhradně na aktuálních vstupech, rekurentní neuronové sítě jsou navrženy tak, aby dokázaly zpracovávat sekvenční data, jako jsou například časové řady nebo řetězce textů a zvuků, u kterých může mít celý průběh sekvence zásadní význam pro interpretaci dat. To je umožněno díky tomu, že v rekurentních neuronových sítích má neuron v každém kroku uloženou hodnotu stavu (odpovídající výstupu v tomtéž kroku), která se v následujícím časovém kroku stává jedním z jeho vstupů, čímž jsou schopny uchovávat a předávat informaci. [16]

Model neuronu rekurentní neuronové sítě je znázorněn na obrázku 2.4.

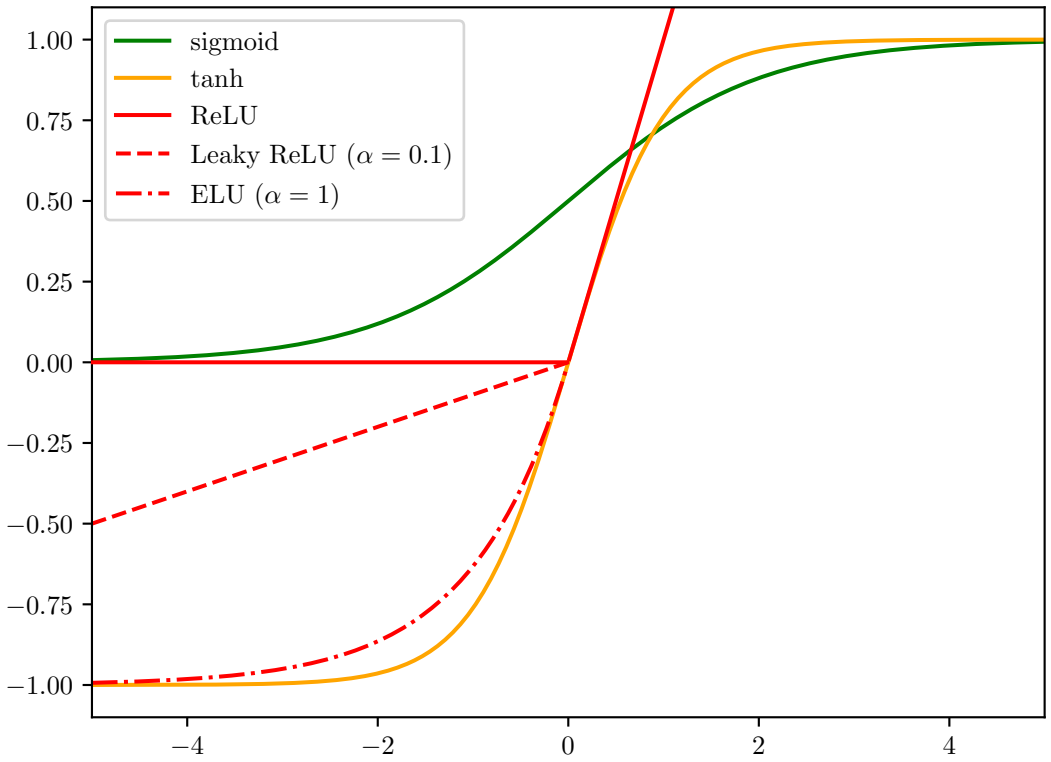
Průchod rekurentní neuronovou sítí lze vypočítat pomocí rovnice (2.11) postupnou iterací všemi časovými kroky, přičemž i je index vrstvy, t je časový krok, $\mathbf{W_x}$ je matice vah vstupů, $\mathbf{x}^t$ je vektor vstupů pro daný časový krok (shodný s vektorem výstupů předchozí vrstvy $\mathbf{y}^{(i-1),t}$), $\mathbf{W_h}$ je vektor vah aktivací neuronů v předchozím časovém kroku $t-1$, $\mathbf{b}$ je vektor prahů, a je aktivační funkce a $\mathbf{y}^t$ je vektor výstupů v daném časovém kroku.

$$\mathbf{y}^{(i),t}(\mathbf{x}) = a(\mathbf{W}_x^{(i)}\mathbf{x}^{(i),t} + \mathbf{W}_h^{(i)}y^{(i),t-1} + \mathbf{b}^{(i)}) \tag{2.11}$$

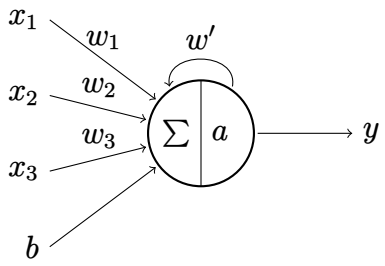
Pro první vzorek vstupní sekvence v kroku $t = 0$ je nutné stanovit hodnoty stavů jednotlivých neuronů jako nulové.

Ačkoliv umožňují rekurentní neuronové sítě zpracovávat sekvenční data, s ohledem na dlou-

[1] Hyperparametry jsou takové parametry, které nemohou být upraveny učením a musí tedy být stanoveny před jeho začátkem. Příkladem dalších hyperparametrů jsou například počet vrstev a počet neuronů v nich nebo volba aktivační funkce.



Obrázek 2.3: Průběhy aktivačních funkcí sigmoid, tanh, ReLU, Leaky ReLU a ELU



Obrázek 2.4: Neuron v rekurentní neuronové sítí

hodobé uchovávání informací u u nich nastávají dva problémy označované jako mizející a explodující gradient, které budou podrobněji popsány v podkapitole 2.3.3. [16]

2.2.5 Rekurentní neuronové sítě s bránami

Rekurentní neuronové sítě s bránami jsou rekurentními neuronovými sítěmi upravenými pro dosažení lepších výsledků při zpracovávání dlouhých sekvencí dat. Toho je dosaženo přidáním tzv. bran, prvků, které jsou na základě vlastních parametrů otevírány a zavírány, čímž přímo ovlivňují tok dat a tedy i stav neuronu uchovávající potřebnou informaci po libovolně dlouhou dobu. [17] Samotný neuron, jakožto základní funkční prvek, je v těchto neuronových sítích nahrazen paměťovým blokem, jehož vnitřní stav je řízen bránami.

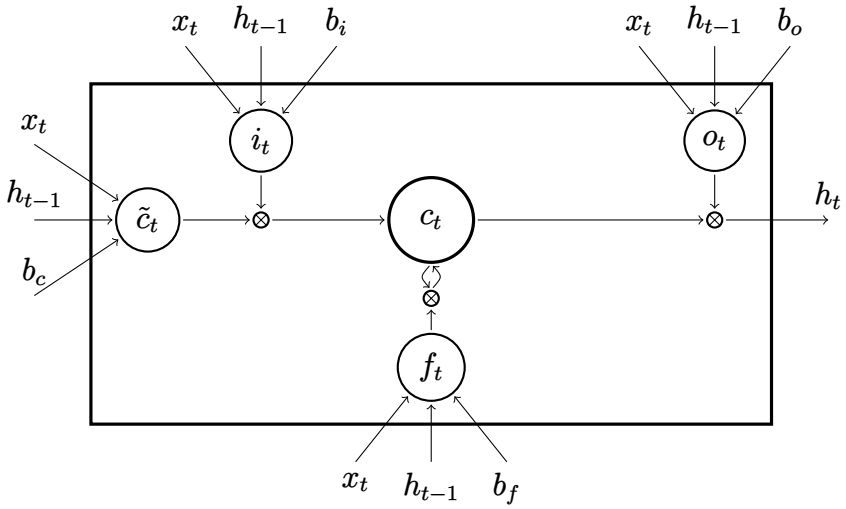
LSTM (*long short-term memory*)

LSTM je rekurentní neuronovou sítí využívající bran představenou v roce 1997 S. Hochreiterem a J. Schmidhuberem. V době první publikace disponovala dvěma bránami, vstupní a výstupní. [18] Později byla z důvodu nemožnosti LSTM zpracovávat nepřetržitý tok dat, ale pouze sekvence omezené délky, přidána zapomínací brána, která umožňuje vymazat již nadále nerelevantní informace. [19] Ve své aktuální podobě tedy paměťový blok LSTM obsahuje tři

brány a jednu paměťovou buňku.

Jelikož stav bran může nabývat dvou extrémních stavů – otevřeno a zavřeno, je pro jejich výpočet použita výhradně logistická aktivační funkce, neboť má výstupy v intervalu $(0; 1)$.

Paměťový blok LSTM je znázorněn na obrázku 2.5.



Obrázek 2.5: Paměťový blok LSTM

Vstupní brána (*input gate*) určuje, která informace má být uložena do vnitřního stavu paměťové buňky. Jejím vstupem v každém časovém kroku je vstupní vektor $\mathbf{x}_t$ vynásobený maticí příslušných vah $\mathbf{W}_{ix}$, vektor výstupů z předchozího časového kroku $\mathbf{h}_{t-1}$, opět vynásobený příslušnými vahami $\mathbf{W}_{ih}$ a vektor prahů $\mathbf{b}_i$. Rovnice pro výpočet vstupní brány i_t je

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i). \tag{2.12}$$

Zapomínací brána (*forget gate*) rozhoduje, zda má být stav paměťové buňky zachován, respektive použit v dalším časovém kroku. Tato brána byla později přidána do původního návrhu LSTM, jelikož bez ní docházelo k postupnému nárůstu hodnoty stavu paměťové buňky, což znemožňovalo použití LSTM pro zpracovávání kontinuálních sekvencí dat. [20] Její výstupní hodnota je označena $\mathbf{f}_t$ a je vyjádřena rovnicí

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f), \tag{2.13}$$

přičemž $\mathbf{x}_t$ je vektor vstupů v daném časovém kroku t, $\mathbf{W}_{fx}$ je matice vah vstupního vektoru, $\mathbf{h}_{t-1}$ je vektor stavů paměťových buněk v předchozím časovém kroku, $\mathbf{W}_{hf}$ je matice jemu odpovídajících vah a $\mathbf{b}_t$ je vektor prahů v daném časovém kroku.

Stav paměťové buňky (*cell state*) tvoří vnitřní paměť LSTM bloku, uchovávající potřebné vlastnosti dat po neomezeně dlouhou dobu. Jeho hodnota je ovládána vstupní a zapomínací branou, přičemž je nejprve vypočten tzv. vstupní kandidát

$$\tilde{\mathbf{c}}_t = g(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c), \tag{2.14}$$

kde g je zvolená aktivační funkce, $\mathbf{W}_{cx}$ je matice vah vstupního vektoru, $\mathbf{W}_{ch}$ je matice vah minulých výstupů a $\mathbf{b}_c$ je vektor prahů. Poté tento vstupní kandidát prochází vstupní branou a

společně s hodnotou minulého stavu, která projde přes zapomínací bránu, tvoří stav paměťové buňky

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t. \tag{2.15}$$

Výstupní brána (*output gate*) řídí, jaká část stavu paměťové buňky se dostane na výstup samotného bloku. Je popsána rovnicí

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o), \tag{2.16}$$

kde $\mathbf{W}_{ox}$ je matice vah vstupního vektoru, $\mathbf{W}_{oh}$ je matice vah minulých výstupů a $\mathbf{b}_o$ je vektor prahů.

Samotný výstup je poté vypočítán pomocí rovnice

$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t), \tag{2.17}$$

kde $\mathbf{o}_t$ je výše zmíněný výstup výstupní brány, g je zvolená výstupní aktivační funkce a $\mathbf{c}_t$ je stav paměťové buňky.

GRU (*gated recurrent units*)

GRU byla představena v roce 2014 a využívá zjednodušenou architekturu inspirovanou LSTM. Místo tří bran má pouze dvě, aktualizační a resetovací. Díky tomuto zjednodušení je její implementace snažší a její učení je výpočetně méně náročné. [21] I přesto však v modelech se stejným množstvím parametrů může dosahovat srovnatelných výsledků jako LSTM a zároveň rychleji konverguje. [22]

Aktualizační brána (*update gate*) určuje, v jakém poměru se ve výstupu bloku GRU projeví nový vstup a výstup z předchozího časového kroku. Její výpočet je dán rovnicí

$$\mathbf{z}_t = \sigma(\mathbf{W}_{zx}\mathbf{x}_t + \mathbf{W}_{zh}\mathbf{h}_{t-1} + \mathbf{b}_z), \tag{2.18}$$

kde $\mathbf{W}_{zx}$ je matice vah vstupního vektoru, $\mathbf{W}_{zh}$ je matice vah výstupů z předchozího časového kroku a $\mathbf{b}_z$ je vektor prahů. [21]

Resetovací brána (*reset gate*) rozhoduje o tom, jaká část z předchozího výstupu bude zapomenuta nebo bude použita pro další výpočet. Je definována rovnicí

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rx}\mathbf{x}_t + \mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{b}_r), \tag{2.19}$$

kde $\mathbf{W}_{rx}$ je matice vah vstupního vektoru, $\mathbf{W}_{rh}$ je matice vah výstupů z předchozího časového kroku a $\mathbf{b}_r$ je vektor prahů. [21]

Skrytý stav (*hidden state*) Skrytý stav GRU bloku se počítá stejně jako vstupní kandidát LSTM bloku, s tím rozdílem, že je míra přenosu předchozích výstupů řízena resetovací bránou. Jeho výpočet je dán rovnicí

$$\tilde{h}_t = f(\mathbf{W}_{hx}x_t + \mathbf{W}_{hh}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h), \tag{2.20}$$

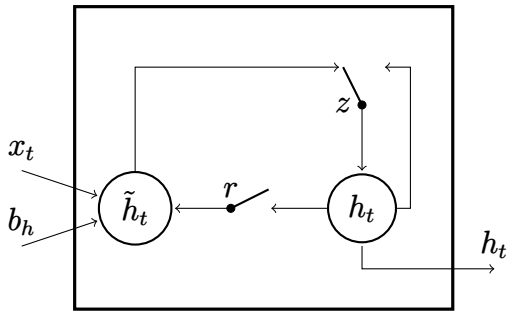
kde $\mathbf{W}_{hx}$ je matice vah vstupního vektoru, $\mathbf{W}_{hh}$ je matice vah výstupů z předchozího časového kroku, $\mathbf{b}_h$ je vektor prahů a $\mathbf{r}_t$ představuje stav resetovací brány. [21]

Samotný výstup vrstvy GRU bloků je potom dán rovnicí

$$\mathbf{h}_t = \mathbf{z}_t\mathbf{h}_{t-1} + (1 - z_t)\tilde{\mathbf{h}}_t. \tag{2.21}$$

Z rovnice (2.21) je patrné, že, oproti rovnici (2.15) u LSTM, není možné nezávisle ovlivňovat, jaké množství informací z předchozích časových kroků a jaké z aktuálního vstupu se dostanou do aktuálního výstupu. Obě hodnoty jsou svázány stavem aktualizační brány. [22]

Struktura bloku GRU je na obrázku 2.6.



Obrázek 2.6: Blok GRU

2.3 Trénování neuronových sítí

Trénování neuronových sítí slouží k naučení modelu správně vyhodnocovat vstupní data. Během tohoto procesu dochází k optimalizaci parametrů sítě za pomoci trénovacího algoritmu. Míra naučení a celková schopnost modelu plnit zadání se vyhodnocuje pomocí ztrátové a nákladové funkce, vysvětlené v podkapitole 2.3.1, s cílem je minimalizovat. [13, s. 168-169]

Před započetím učení modelu je vhodné soubor dat rozdělit na tři části – trénovací data, validační data a testovací data. Validační soubor a testovací soubor bývají zpravidla stejně velké a zároveň mnohem menší než trénovací soubor. Cílem tohoto rozdělení je možnost ověření kvality výstupních dat modelu (v případě učení s učitelem). Pokud by validace a testování probíhaly na stejném datasetu jako trénování, mohlo by se stát, že by si model data „zapamatoval“, spíše než dokázal rozklíčovat vzájemné vztahy mezi pozorovanými vlastnostmi. Nedokázal by tedy generalizovat na nová data, ale přesto by model mohl být uveden do reálného provozu, protože by při finálním testování na souboru dat, na kterém byl i učen, poskytoval uspokojivé výsledky. Validační set se proto využívá pro průběžné ověřování výkonnosti modelu po jednotlivých trénovacích cyklech (epochách). [5, s. 64] Testovací set se potom využívá k finálnímu testování před uvedením modelu do provozu.

Dalším nutným krokem před započetím učení je rovněž inicializace parametrů modelu.

2.3.1 Ztrátová a nákladová funkce

Ztrátová funkce *loss function* vyjadřuje chybu mezi aktuální výstupní hodnotou neuronové sítě a skutečnou hodnotou, tedy anotací, pro jeden konkrétní vstup. [5]

Nákladová funkce *cost function* vychází ze ztrátové funkce, ale určuje chybu neuronové sítě pro všechny vektory vlastností v celém souboru dat. Nákladová funkce tak může být například průměrnou hodnotou výstupů ztrátové funkce pro všechny vektory vlastností. [5]

Ztrátová, a tedy i nákladová, funkce musí, stejně jako aktivační funkce, být spojitá a ale-

spoň po částech hladká. Jelikož oba typy funkcí porovnávají aktuální výstup s tím skutečným, používají se pouze při učení s učitelem.

Kvadratická chyba

Kvadratická chyba, je jednou z nejčastěji používaných ztrátových funkcí.

Její obdoba nákladové funkce je střední kvadratická chyba *mean squared error*. Jedná se o střední hodnotu druhou mocninou umocněného rozdílu skutečné hodnoty y_i (anotace) a aktuálního výstupu $\hat{y}_i$ pro všech n vektorů vlastností. [23] Tento vtah je matematicky popsán rovnicí

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{2.22}$$

Ztrátová funkce potom dána rovnicí

$$L(x) = (y_i - \hat{y}_i)^2. \tag{2.23}$$

Absolutní chyba

Střední absolutní chyba *mean absolute error* se od MSE liší pouze tím, že rozdíl mezi skutečným y_i a aktuálním výstupem $\hat{y}_i$ není umocněn, ale pro zajištění kladného výstupu funkce je použita absolutní hodnota. Výhodou absolutní chyby oproti kvadratické chybě je její menší citlivost na chyby vzniklé výrazně odlehlými hodnotami y_i, které se mezi anotace mohly dostat například chybou měření. Zároveň z toho ale plyne nevýhoda, jelikož má vyšší citlivost na blízké body a tím pádem nedochází k postupnému zmenšování gradientu, a tedy i kroku při úpravě parametrů, s přibližováním se minimu funkce. [23]

MAE je dána rovnicí

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{2.24}$$

Ztrátovou funkcí je výraz

$$L(x) = |y_i - \hat{y}_i|. \tag{2.25}$$

Huberova ztrátová funkce

Huberova ztrátová funkce je kombinací kvadratické a absolutní chyby, čímž sdružuje výhody obou těchto funkcí. Je tedy odolnější vůči odlehlejším hodnotám než kvadratická chybová funkce a zároveň konverguje rychleji než absolutní chybová funkce. [24]

Je definována rovnicí

$$L(x) = \begin{cases} \frac{1}{2\delta}x^2 + \frac{1}{2}\delta, & x \leq \delta \\ |x|, & x > \delta, \end{cases} \tag{2.26}$$

přičemž hyperparametr δ udává přechod mezi oběma funkcemi.

2.3.2 Zpětná propagace (*backpropagation*)

Zpětná propagace je algoritmus využívaný při učení neuronových sítí sloužící k výpočtu gradientu nákladové funkce, tedy ke zjištění vlivu jednotlivých parametrů modelu na výsledkou chybu výstupu. Zpětná propagace pro tento výpočet využívá řetězového pravidla pro průchod jednotlivými vrstvami neuronové sítě. [25]

Během procesu učení je nejprve provedena dopředná propagace za účelem zjištění aktuálního výstupu z neuronové sítě, který slouží k určení chyby pomocí ztrátové funkce. Následně je zpětnou propagací určen vliv parametrů výstupní vrstvy na tuto chybu a pomocí řetězového pravidla se přes skryté vrstvy postupuje až ke vstupní vrstvě.

Pokud je chyba výstupu $L(\hat{y})$ vyjádřena rovnicí

$$L(\hat{y}) = (y - \hat{y})^2, \tag{2.27}$$

kde y je skutečná a $\hat{y}$ aktuální hodnota výstupu neuronové sítě, přičemž

$$\hat{y}(z(x)) = a^{(l)}(z(a^{(l-1)})) = a^{(l)}(wa^{(l-1)} + b), \tag{2.28}$$

kde l udává index poslední skryté vrstvy, a je výstup aktivační funkce neuronu, w je příslušná váha a b práh, potom lze parciální derivaci chyby $L(\hat{y})$ podle dané váhy $w^{(l)}$ zapsat pomocí řetězového pravidla jako

$$\frac{\partial L}{\partial w^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w^{(l)}} \tag{2.29}$$

a parciální derivaci chyby $L(\hat{y})$ podle prahu $b^{(l)}$ jako

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial b^{(l)}}. \tag{2.30}$$

Jelikož je výstup aktivační funkce $a^{(l)}(a^{(l-1)})$ funkcí výstupu předchozí vrstvy a ta je zase funkcí $a^{(l-2)}$, je pro zjištění parciálních derivací podle dalších parametrů nutné řetězové pravidlo aplikovat opakovaně. Například pardiální derivace $L(\hat{y})$ podle váhy v předposlední vrstvě $l-1$ je vyjádřena rovnicí (pro přehlednost je vynechán mezivýpočet váženého součtu z)

$$\frac{\partial L}{\partial w^{(l-1)}} = \frac{\partial L}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial a^{(l-1)}} \frac{\partial a^{(l-1)}}{\partial w^{(l-1)}}. \tag{2.31}$$

Zpětná propagace v čase (*backpropagation through time*)

Zpětná propagace v čase je rozšířením standardní zpětné propagace pro použití v rekurentních neuronových sítích s dynamickými vstupními daty. Nejprve je, stejně jako u samotné zpětné propagace, vypočítán výstup sítě dopřednou propagací a poté je řetězovým pravidlem prováděna zpětná propagace v čase všemi vrstvami i časovými kroky. [26]

Průběh výpočtu je možné přiblížit derivací rovnice (2.11) podle váhy skryté vrstvy $W_{h,j}^{(i)}$. Potom

$$\frac{\partial y_j^{(i),t}}{\partial W_{h,j}^{(i)}} = \frac{\partial a_j^{(i),t}}{\partial z_j^{(i),t}} \frac{\partial z_j^{(i),t}}{\partial W_{h,j}^{(i)}}, \tag{2.32}$$

kde

$$\frac{\partial z_j^{(i),t}}{\partial W_{h,j}^{(i)}} = y_j^{(i),t-1} + W_{h,j}^{(i)} \frac{\partial y_j^{(i),t-1}}{\partial W_{h,j}^{(i)}}, \tag{2.33}$$

přičemž t je časový krok, i je index vrstvy a j je index neuronu v dané vrstvě. Rovnice (2.32) a (2.33) jsou opakovány, dokud není $t = 0$. Pro ostatní parametry je postup obdobný.

2.3.3 Mizející a explodující gradient

Mizející a explodující gradient nastávají nejčastěji při dopředné i zpětné propagaci u rekurentních a konvolučních neuronových sítí při zpracovávání sekvenčních dat. V důsledku exponenciálního charakteru výpočtů po sobě jdoucích vrstev ale mohou nastávat i v případě hlubokých dopředných neuronových sítí.

Mizející gradient (*vanishing gradient*)

nastává, když se vypočtený gradient stává velmi malým To má za následek, že následná úprava parametrů v dřívějších vrstvách není dostatečná a model se není schopen efektivně naučit významné souvislosti a vzorce v souboru dat v dlouhodobém časovém horizontu, což může vést k nevyhovujícímu výkonu modelu. [27]

Explodující gradient (*exploding gradient*)

je přesně opačným problémem a nastává, když gradient při průchodu neuronovou sítí nepřiměřeně roste. To následně způsobuje velké změny při úpravě parametrů sítě. Důsledkem může být, že během učení není možné dosáhnout konvergence k optimálnímu řešení. [27]

Mizející a explodující gradient je možné omezit vhodnou volbou aktivační funkce a optimalizačního algoritmu, z nichž některé byly specificky vytvořeny za tímto účelem. Dále je možné využít některé z regularizačních technik popsaných v podkapitole 2.3.7. Například aktivační funkce hyperbolický tangens a logistická funkce jsou nevhodné pro použití v hlubokých neuronových sítích, jelikož mají v celém definičním oboru obor hodnot v intervalu $(-1; 1)$, respektive $(0; 1)$, proto u nich může docházet k mizejícímu gradientu a je tedy výhodnější použít ReLU, „Leaky“ ReLU nebo ELU. U rekurentních neuronových sítí ale při použití nesaturované aktivační funkce ReLU a jejich variací zase může častěji docházet k explodujícímu gradientu, čemuž by saturovaná aktivační funkce zabránila.

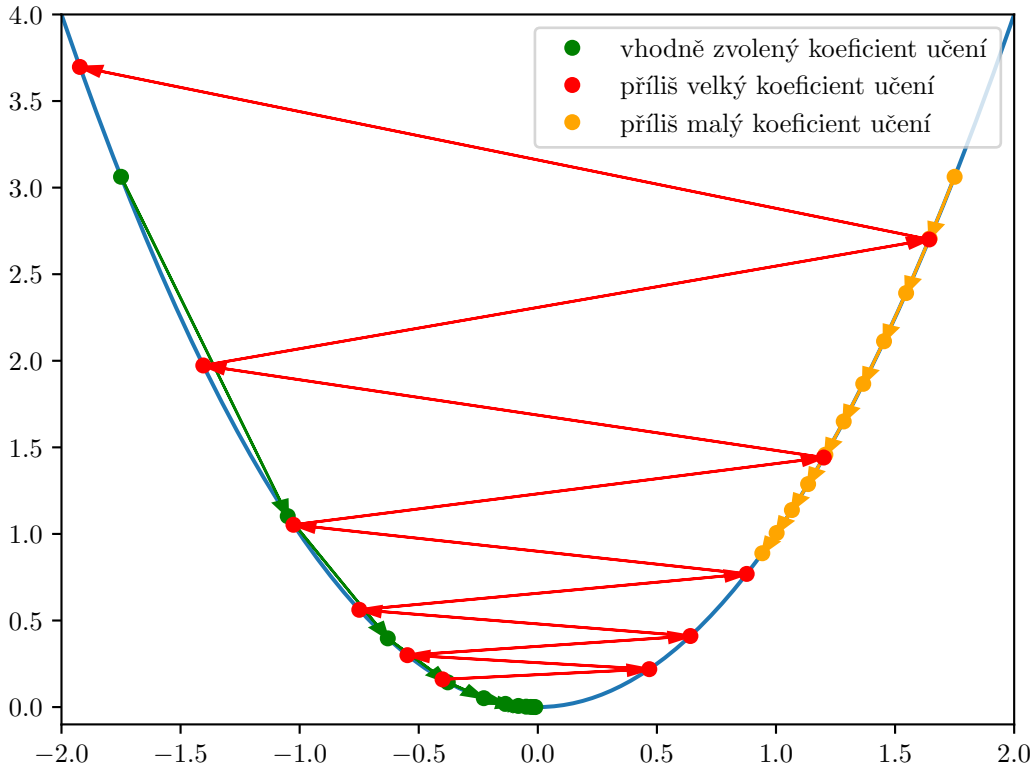
Jelikož velikost vypočtené chyby při zpětné propagaci závisí exponenciálně na velikostech vah, standardní rekurentní neuronové sítě nejsou schopny zohlednit souvislosti vstupních dat vzdálené od sebe více než pět až deset časových kroků. [20]

2.3.4 Koeficient učení (*learning rate*)

Koeficient učení je jedním ze stěžejních hyperparametrů, který udává, jak moc mají být parametry modelu upraveny po jednom trénovacím kroku.

Jeho volbou je možné přímo ovlivnit, jak rychle bude model během učení konvergovat k optimálnímu řešení. Příliš malé hodnoty mají za následek velmi pomalé učení a v krajním případě je až nemožné takový model optimalizovat, nicméně vždy bude mít tendenci konvergovat. Naopak příliš vysoké hodnoty mohou způsobovat nestabilitu a vést k divergenci. [2, s. 325]

Vliv volby velikosti koeficientu učení je znázorněn na obrázku 2.7.



Obrázek 2.7: Koeficient učení

2.3.5 Optimalizační algoritmy

Optimalizace parametrů modelu neuronové sítě minimalizací ztrátové funkce je možná pouze v případě, že je možné minimum ztrátové funkce určit. Toho je dosaženo pomocí tzv. optimalizačních algoritmů.

Numerickou optimalizací pomocí nákladové funkce nelze zajistit, že je nalezené minimum globální. V praxi je však mnohdy postačující nalézt i lokální minimum.

Gradientní sestup (*gradient descent*)

Gradientní sestup slouží k výpočtu gradientu ztrátové funkce podle všech parametrů, které neuronová síť má. Jelikož gradient je vektor představující směr největšího růstu dané funkce, je provedena úprava všech parametrů v jeho opačném směru, čímž dojde ke zmenšení výstupu nákladové funkce a tedy optimalizaci celého modelu. [28]

Určení gradientního sestupu pro komplexní modely je výpočetně velmi náročné, protože je počítán pro všechny parametry modelu a pro všechny vzorky v datovém souboru. Proto jsou častěji používány jeho varianty, pomocí jichž je možné dosáhnout konvergence během učení rychleji.

Gradientní sestup je popsán rovnicí

$$\theta_{i+1} = \theta_i - \alpha \nabla L(\theta_i), \tag{2.34}$$

přičemž hyperparametr α je zvolený koeficient učení.

Stochastický gradientní sestup (*stochastic gradient descent*)

Stochastický gradientní sestup využívá, narozdíl od gradientního sestupu, v každém kroku výpočtu gradientu pouze jeden náhodně vybraný vzorek z trénovacího datasetu a je tedy mnohem rychlejší na výpočet. Nevýhodou však je, že místo stálého snižování výstupu nákladové funkce se může celková chyba v jednotlivých krocích měnit v obou směrech, pouze s postupnou tendencí klesat. Není tedy možné tímto algoritmem dosáhnout absolutního minima, ale pouze jeho blízkosti. Postupným snižováním koeficientu učení je ale možné toto blízké okolí zmenšovat. [28]

Díky své nahodilosti má stochastický gradientní sestup vyšší šanci překonat lokální minima a oblasti s nulovou derivací ztrátové funkce a tím nalézt oblast globálního minima. Na druhou stranu, pomocí stochastického gradientního sestupu je obtížné nalézt minimum nákladové funkce v oblastech připomínajících tvar rokle, neboť může v tomto případě docházet k oscilačnímu pohybu ve směru kolmém na pohyb vedoucí k minimalizaci nákladové funkce. [28]

Matematický popis stochastického gradientního sestupu vychází z gradientního sestupu a je dán rovnicí

$$\theta_{i+1} = \theta_i - \alpha \nabla L(\theta_i) \tag{2.35}$$

Hybnost (*momentum*)

Hybnost rozšiřuje gradientní sestup o vlastnost analogickou k mechanické hybnosti. Hybnost využívá klouzavý průměr gradientů z předchozích časových kroků společně s aktuálně vypočteným gradientem, čímž pomáhá zabránit oscilujícím změnám parametrů modelu a tím přispívá k rychlejší konvergenci během učení. Zároveň je díky hybnosti možné překonat lokální minima a roviny v průběhu optimalizačního kritéria.

Hybnost upravuje rovnici pro výpočet (stochastického) gradientního sestupu na rovnice (2.36) a (2.37), přičemž μ je hyperparametr představující hodnotu hybnosti, b_t je gradient se započtenou hybností pro časový krok t, α je koeficient učení a θ je vektor parametrů modelu. [29]

$$b_t = \mu b_{t-1} + g_t \tag{2.36}$$
$$\theta_t = \theta_{t-1} - \alpha b_t \tag{2.37}$$

Adam (*adaptive moment estimation*),

Adam, v překladu odhad adaptivních momentů, je dalším optimalizačním algoritmem vycházejícím ze stochastického gradientního sestupu a spojujícím výhody dalších algoritmů, jako jsou Adagrad a RMSprop.

Pro výpočet využívá exponenciálních klouzavých průměrů daných gradientů

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \text{ a} \tag{2.38}$$

kvadratických gradientů

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2.39}$$

přičemž míra jejich exponenciálních úpadků je ovlivněna hyperparametry β_1 a β_2, náležícími do intervalu $\langle 0; 1)$. [30]

Jelikož jsou m_t a v_t inicializovány jako nulové vektory, na počátku procesu učení by docházelo k výpočtu odhadů momentů blížících se nule. Z tohoto důvodu je nutná jejich korekce pomocí rovnic

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \text{ a} \tag{2.40}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2.41}$$

jejichž význam pro pozdější časové kroky prakticky zaniká, jelikož je při každé iteraci z počátku nulový parametr t inkrementován.

Následná úprava parametrů modelu je vypočtena pomocí rovnice

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{2.42}$$

Účinnost algoritmu lze dále vylepšit nahrazením rovnic (2.40), (2.41) a (2.42) rovnicemi (2.43), (2.44) a (2.45). [30]

$$\alpha_t = \alpha \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \tag{2.43}$$

$$\hat{\epsilon} = \epsilon \sqrt{1 - \beta_2^t} \tag{2.44}$$

$$\theta_t = \theta_{t-1} - \alpha_t \frac{m_t}{\sqrt{v_t} + \hat{\epsilon}} \tag{2.45}$$

Dle jeho autorů je vhodná volba výchozích hyperparametrů následující: $\alpha = 0{,}001$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ a $\epsilon = 10^{-8}$. [30]

2.3.6 Nedoučení a přeučení

Model neuronové sítě je považován za správně navržený a naučený, pokud dokáže vhodně generalizovat na nová data, tedy že je dokáže správně vyhodnocovat na základě trénovacího vzorku, přesto, že se jedná o zcela rozdílná data. Z pohledu schopnosti modelu se přizpůsobit vstupním datům mohou nastat dvě situace, kdy model na konci trénovacího procesu nevyhovuje požadavkům.

Nedoučení (*underfitting*)

Nedoučení nastává, když model není schopen dostatečně charakterizovat vlastnosti vstupních dat. Tato situace může nastat, pokud je model příliš jednoduchý. Příkladem může být snaha aproximovat nelineární funkci lineárním modelem. V tomto případě je nutné zvětšit komplexitu modelu přidáním dalších vrstev nebo zvýšením počtu neuronů v těch stávajících. [31]

Další možnou příčinou je, že byl model trénován s malým nebo nedostatečně reprezentativním vzorkem dat na to, aby v něm byly obsaženy všechny vzájemné vztahy mezi pozorovanými vlastnostmi, případně nebyl model dostatečně naučen. Řešením je obstarání více dat, zvýšení komplexity vektoru vlastností tak, aby byl reprezentativnější, nebo prodloužení a optimalizace procesu učení.

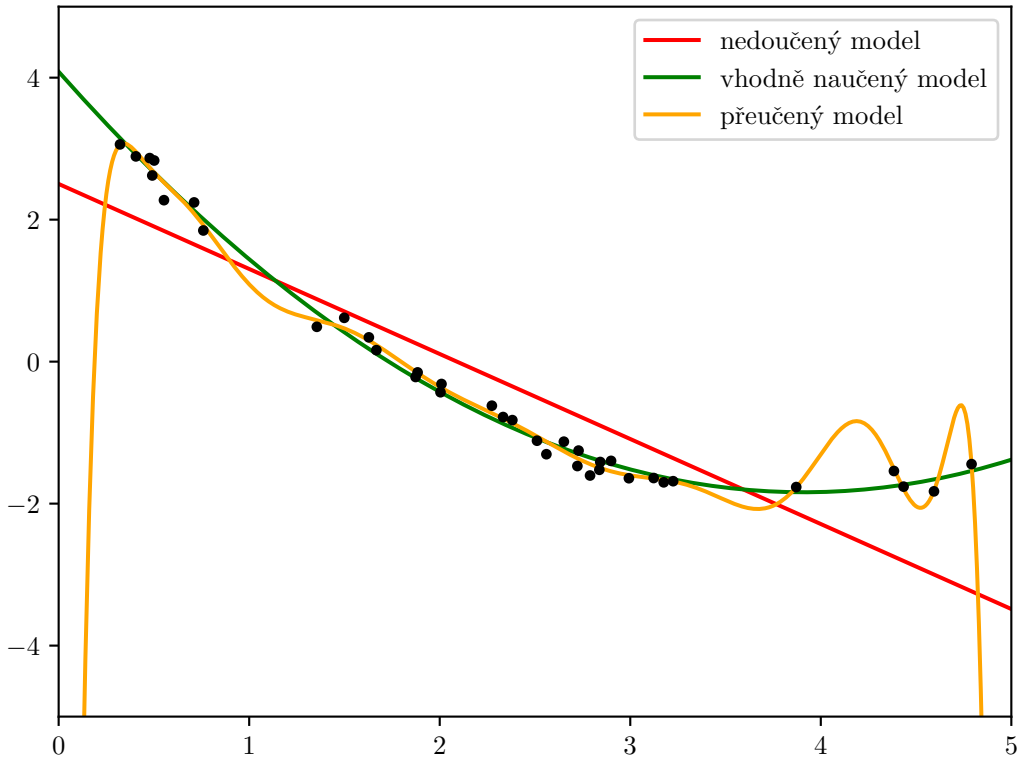
Přeučení (*overfitting*)

Přeučení nastává pokud je model neuronové sítě příliš složitý na pozorované vlastnosti v souboru dat. Model se potom příliš přizpůsobí trénovacím datům, která mohou být mnohdy i zarušená, a poté již nedokáže dostatečně generalizovat na nová data. [32] Přeučení ale může nastat i u lineárního modelu, zejména pokud je pozorováno velké množství vlastností na malém souboru dat. Jedná se tedy o přesně opačný problém jakým je nedoučení.

Přeučený model se vyznačuje nízkou chybou nákladové funkce na trénovacích datech, ale zároveň špatným vyhodnocováním validačních, případně testovacích, vzorků.

Řešením může být obstarání většího trénovacího souboru dat, použití jednoduššího modelu (snížení počtu skrytých vrstev a neuronů v nich) nebo redukce dimenzí vektoru vlastností. Nejčastější metodou pro zamezení přeučení je ale regularizace. Tento termín zahrnuje mnoho různých způsobů omezování modelu, z nichž některé zde budou podrobněji popsány v následující podkapitole 2.3.7.

Na obrázku 2.8 jsou graficky znázorněny příznaky nedoučení a přeučení. Je možné pozorovat, že v oblastech s vyšší koncentrací bodů se výstup přeučeného modelu příliš neliší od výstupu vhodně naučeného modelu, což dokazuje význam dostatečného množství dat pro trénování jako jednoho z možných prostředků proti přeučení.



Obrázek 2.8: Příklad nedoučeného, vhodně naučeného a přeučeného modelu

2.3.7 Regularizace

Regularizační techniky slouží k omezení složitosti modelu, čímž se dá efektivně zabránit přeučení, ale také i k zamezení mizejícího a explodujícího gradientu.

Jako cíl použití regularizace je také možné označit snahu o snížení rozptylu (*variance*) výstupních hodnot chybové funkce, jejíž průvodním jevem je ale zvětšení zkreslení (*bias*). Vysoký rozptyl chyby predikce modelu je příznakem přeučení modelu. Naopak vysoké zkreslení je způ-

sobeno nedoučením. Je proto důležité vhodně navrženou regularizací nalézt kompromis při snižování rozptylu a zvyšování zkreslení tak, aby výsledná chyba predikce byla co nejnižší. [33]

L_1 regularizace

L_1 regularizace, nazývaná také Lasso regularizace, vnáší do optimalizačního kritéria, tedy ztrátové funkce, penalizační prvek, čímž dochází k vynulování některých parametrů a tím se zlepšuje schopnost modelu generalizovat.

Díky absolutní hodnotě je L_1 regularizace, stejně jako kvadratická chyba, odolná vůči výrazně odlehlým hodnotám v trénovacím datasetu. Její další výhodou, která vyplývá z vynulování některých parametrů, je to, že napomáhá potlačit vliv nerelevantních vlastností vstupních dat na výstup modelu. [33] V případě, že se ale v trénovacích datech nacházejí korelované vlastnosti, může tento jev být nežádoucí, jelikož po L_1 regularizaci zůstanou nenulové parametry reprezentující pouze jednu z nich, což může vést k horším výsledkům modelu. [34]

Matematický popis penalizačního prvku L_1 regularizace je dán rovnicí

$$L_1 = \lambda \sum_{i=1}^{N} |\theta_i| \,, \tag{2.46}$$

kde θ představuje parametry modelu a λ je hyperparametr udávající míru regularizace.

L_2 regularizace

L_2 regularizace, také zvaná Ridge regularizace, funguje na stejném principu jako L_2 regularizace, tedy přičítání penalizačního prvku k optimalizačnímu kritériu, s tím rozdílem, že jsou v něm parametry modelu umocněny. To je činí náchylnější vůči výrazněji odlehlým hodnotám v trénovacím datasetu. Narozdíl od L_1 regularizace dochází u L_2 regularizace pouze ke zmenšování parametrů, nikoliv vynulování. Tato regularizace proto není vhodná v případě, že je nutné vyseparovat relevantní vlastnosti z trénovacích dat. [33]

Matematický popis penalizačního prvku L_2 regularizace je dán rovnicí

$$L_2 = \lambda \sum_{i=1}^{N} \theta_i^2, \tag{2.47}$$

kde θ představuje parametry modelu a λ je opět hyperparametr udávající míru regularizace.

L_2 regularizace není efektivní při použití společně s optimalizačním algoritmem Adam. [35]

Kombinací L_1 a L_2 regularizace vznikne regularizační technika zvaná Elastic Net. [36]

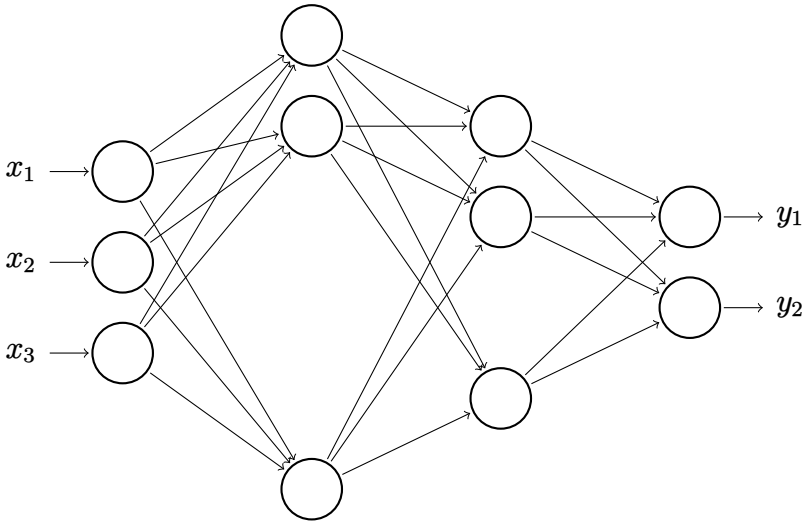
Dropout

Dropout spočívá v náhodném vynechávání některých neuronů během učení na základě stanovené pravděpodobnosti jejich výskytu p. S každou iterací učení tak vzniká nová, „zeštíhlená“, síť. Výsledná naučená síť je potom kombinací všech těchto dílčích sítí. To má za následek rovnoměrnější využití všech parametrů modelu, zamezení přeučení a lepší generalizaci, neboť se model nemůže pouze na vybrané synapse. [37]

Během testování modelu jsou pak aktivní všechny neurony, ale jejich příslušné váhy jsou vynásobeny pravděpodobností p. To je nutné, jelikož během trénování byl průměrný vážený součet vstupů do neuronu $p$krát menší, od kterého se odvíjela hodnota aktivace neuronu. [37]

Jejikož původní návrh nepočítal s použitím v rekurentních neuronových sítích, v [38] je doporučeno použít dropout jak na vstupní, tak na rekurentní synapse. Dále je doporučeno použít stejnou dropout masku pro celou sekvenci a ne jen pro jednotlivé časové kroky.

Příklad struktury neuronové sítě s topologií 3-6-6-2 s aplikovanou dropout regularizací skry-



Obrázek 2.9: Struktura neuronové sítě s aplikovanou dropout regularizací

tých vrstev s pravděpodobností výskytu neuronů v nich $p_h = 0{,}5$ je na obrázku 2.9.

Předčasné ukončení *(early stopping)*

Regularizace předčasným ukončením vychází z úvahy, že u dostatečně složitého modelu dojde po určité době trénování k přeučení. Během učení je proto sledována chyba nákladové funkce na trénovacím i validačním datasetu a ve chvíli, kdy chyba validačních dat začne růst, dojde k ukončení trénování. Díky tomu může být použit složitější model, který je celkově robustnější a méně náchylný na nesprávně anotovaná data, a přitom nejdojde k jeho přeučení. [39]

Oříznutí gradientu *(Gradient clipping)*

Oříznutí gradientu spočívá v omezení maximální absolutní hodnoty vypočtených gradientů. To je přínosné pro zamezení přílišné změny parametrů modelu na základě odlehlých hodnot v datech a celkově to zvyšuje jeho stabilitu. Zároveň může omezit explodující gradient. [27]

2.4 Zhodnocení provedené rešerše

V provedené rešerši byly rozebrány základní pojmy z oblasti strojového učení a umělých neuronových sítí.

S ohledem na zadání práce, *Dlouhodobé prediktivní modelování nelineárních dynamických systémů pomocí rekurentních neuronových sítí*, se jeví jako vhodné použití rekurentních neuronových sítí s bránami. Pro učení rekurentních neuronových sítí bude použita zpětná propagace v čase.

Ačkoliv existují profesionálně vyvýjené softwarové implementace strojového učení, jako jsou např. TensorFlow[2] nebo PyTorch[3], vlastní implementace na úrovni matematického popisu může být svou názorností přínosná pro ověření platnosti provedené rešerše. Pro nasazení modelu do provozu je naopak vhodnější použít existující řešení.

[2] `https://www.tensorflow.org`
[3] `https://pytorch.org`

3 Praktická část

V této části práce bude s využitím teorie popsané v rešeršní části proveden návrh softwarového řešení se zaměřením na dlouhodobou predikci. Následně bude provedena analýza obdržených dat z měření teplot obráběcího stroje a v závěru práce bude proveden návrh a ověření přesnosti modelů vytvořených s využitím popsané implementace i existujícího frameworku TensorFlow.

3.1 Vypracování vlastního softwarového řešení

Tato podkapitola slouží pro popis vypracované implementace neuronové sítě v jazyku Python na základě provedené rešerše. Pro maticové operace je použit modul numpy. Hlavní motivací pro vytvoření vlastní implementace neuronové sítě bylo aplikování a hlubší pochopení teorie zpracované v rámci rešeršní části této práce. Použití takéví implementace je ale za cenu předpokládaných horších výsledků v oblasti dlouhodobé predikce, než by bylo možné dosáhnout s existujícím softwarovým řešením.

Jelikož byla tato implementace vytvořena před získáním souboru naměřených dat, bylo nutné, aby s ohledem na zadání této práce splňovala následující podmínky:

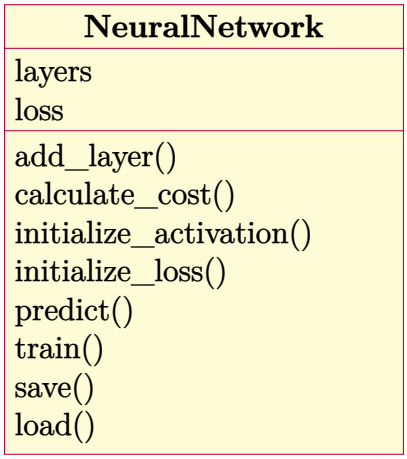
- příjem a zpracování sekvenčních dat
- možnost vytvořit model s libovolnou topologií sítě v kombinaci s volbou implementovaných architektur, aktivačních funkcí, ztrátových funkcí, optimalizačních algoritmů a regularizačních technik
- získání sekvence predikovaných výstupních hodnot

Na základě rešerše byly implementovány následující prvky:

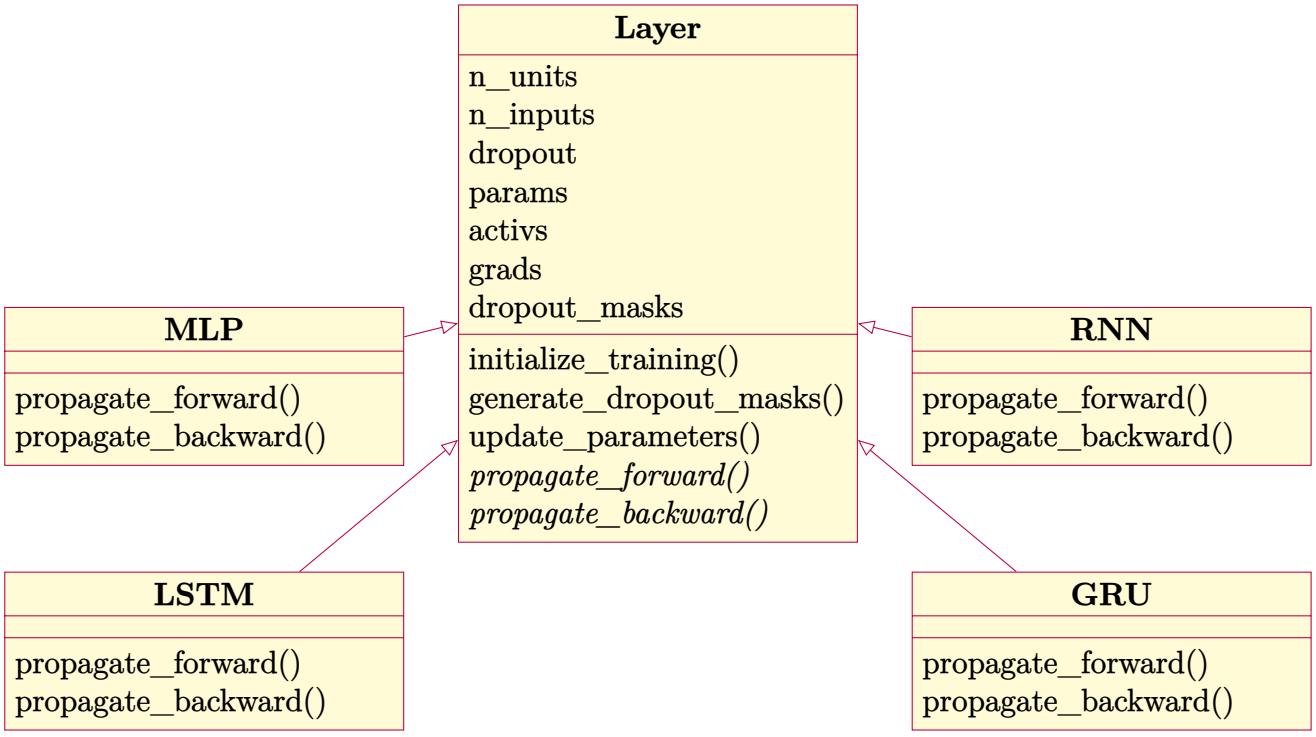
- architektury vrstev – MLP, RNN, LSTM, GRU
- aktivační funkce – sigmoid, hyperbolický tangens, ReLU, LReLU, ELU
- ztrátové funkce – kvadratická chyba, absolutní chyba, Huberova funkce
- optimalizační algoritmy – (stochastický) gradientní sestup, Adam
- regularizační techniky – oříznutí gradientu, dropout

Samotný model neuronové sítě je reprezentován třídou `NeuralNetwork`, jejíž diagram je znázorněn na obrázku 3.1. Pro jeho inicializaci je zavolán konstruktor třídy s povinnými argumenty udávajícími počet pozorovaných vlastností (a tím počet neuronů ve vstupní vrstvě), množství a architektury skrytých vrstev a architekturu výstupní vrsty, včetně počtu neuronů v nich a jejich příslušných aktivačních funkcí. Následně jsou zavolány metody `initialize_activation`, `initialize_loss` a `add_layer`, přičemž jsou inicializované vrstvy, jejichž třídy jsou zobrazené na diagramu 3.2, uloženy v parametru `layers`.

Dopředná propagace je zajištěna metodou `predict` třídy `NeuralNetwork`, která postupně iteruje přes všechny časové kroky vstupní sekvence (metoda ale přijímá i nedynamický vstup) a volá metodu `propagate_forward` potomků třídy `Layer`, přičemž je postupným průchodem všemi vrstvami předáván výstup předchozí vrstvy následující vrstvě jako vstup. Návratovou



Obrázek 3.1: Třída reprezentující model neuronové sítě



Obrázek 3.2: Diagram zobrazující třídy jednotlivých vrstev architektur neuronových sítí

hodnotou metody `predict` může být buď predikce pozorovaných vlastností v následujícím časovém kroku nebo jejich klasifikace, v závislosti na charakteru vstupních dat. Opakovaným voláním metody `predict` v cyklu a přidáním argumentu `continuing=true`, pro uchování paměti rekurentních neuronů, může být dosáhnuto predikce pro libovolný počet časových kroků.

Pro trénování slouží metoda `train` třídy `NeuralNetwork`. Jelikož bylo uvažováno pouze učení s učitelem, přijímaný soubor trénovacích dat je ve formátu seznamu jednotlivý vzorků, přičemž jeden vzorek má podobu $[[x_1, \ldots, x_n], [y_1, \ldots, y_m]]$. Vektor $[x_1, \ldots, x_n]$ představuje sekvenci vstupů, která slouží k predikci hodnoty y_1 a zároveň k vytvoření historie v rekurentních neuronech, přičemž platí $n \geq 1$. Vektor $[y_1, \ldots, y_m]$ je jeho pokračováním a slouží jako anotace, přičemž také platí $m \geq 1$. Po dopředné propagaci vstupního vektoru je vypočítána derivace ztrátové funkce $L'(y_1, \hat{y}_1)$, kde $\hat{y}_1$ je predikovaná hodnota, a následuje iterativní volání metody `propagate_backward` potomků třídy `Layer` v opačném pořadí, přičemž jsou vypočtené gradienty uloženy. Následně je opět provedena dopředná propagace, tentokrát s hodnotou $\hat{y}_1$ a argumentem `continuing=true` a jako anotace slouží y_2. Tento proces je opakován, dokud

nejsou vypočteny gradienty pro všechny prvky ve vektoru anotací. Poté je pokračováno s dalším vzorkem.

Hyperparametr `batch_size` udává velikost trénovací dávky. Po dokončení celé dávky jsou vypočtené gradienty zprůměrovány a pomocí metody `update_parameters` třídy `NeuralNetwork` je provedena aktualizace parametrů modelu pomocí zvoleného optimalizačního algoritmu v kombinaci s halšími hyperparametry, jako je např. koeficient učení.

Za předpokladu, že vstupní vrstva obsahuje více neuronů, vstupy a anotace jsou seskupeny do matic, kde sekvence jednotlivých vlastností tvoří řádky matice a sloupce jednotlivé časové kroky.

Pro regularizaci je možné použít oříznutí gradientů, které je implementováno pomocí dekorátoru při volání metody `update_parameters` potomků třídy `Layer`, nebo nastavením dropout při inicializaci vstev. Dropout je implementován na základě článků [37] a [38]. Během testování jsou tedy příslušné parametry kompenzovány pravděpodobností výskytu příslušných neuronů. Naopak framework TensorFlow využívá tzv. obrácený dropout, který kompenzuje parametry už při trénování, čímž je ušetřeno výpočetní operace během provozu modelu. U malých nebo jednorázově využívaných modelů je však přínos obráceného dropoutu zanedbatelný.

Všechny aktivační i ztrátové funkce jsou implementovány jako samostatné třídy. Díky tomu si mohou samy uchovat nastavení hyperparametrů. Při dopředné propagaci je na nich volána metoda `calc`, která vrací jejich funkční výstup. Během zpětné propagace je pak volána metoda `diff`, která jejich derivací.

Metody `save` a `load` třídy `NeuralNetwork` slouží k uložení a načtení naučeného modelu. K tomu je využito Python modulu h5py.

Zásadní nevýhodou této implementace je doba nutná pro trénování modelů, jelikož veškeré výpočty probíhají pouze na jednom procesorovém jádru. Rozdělením jednotlivých trénovacích dávek do vláken by bylo možné výrazně urychlit proces učení, jelikož aktualizace parametrů modelu probíhá vždy po dokončení celé dávky. V tom případě by ale bylo nutné ošetřit výpočet inkrementačních parametrů, jakým je např. parametr t u optimalizačního algoritmu Adam. Další možnou úpravou by bylo využití paralelních výpočtů pomocí grafické karty.

3.2 Analýza souboru dat

Na obráběcím stroji probíhá měření pomocí teplotních senzorů na různých místech v pravidelných časových intervalech s cílem monitorovat stav stroje.

Obdržená naměřená data obsahují informace o průbězích teplot během osmi po sobě jdoucích dnů v lednu a během pěti po sobě jdoucích dnů v březnu tohoto roku. Měření probíhalo na osmi různých místech stroje se vzorkováním dvacet sekund. Celkem tedy každý teplotní signál obsahuje 56013 hodnot. Bližší informace o původu těchto dat jsou utajeny.

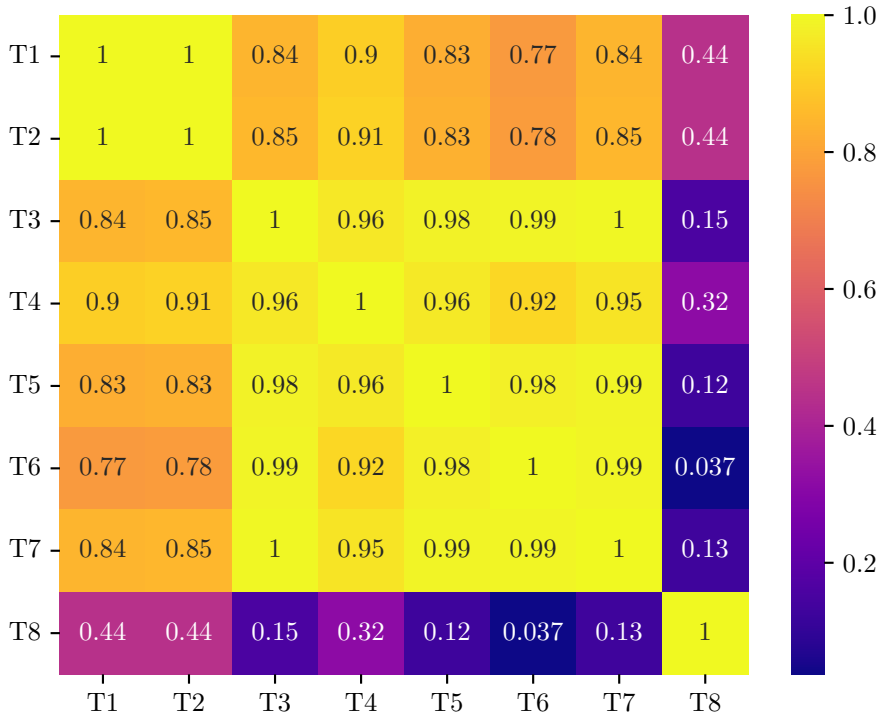
Průběhy teplot jsou znázorněny na obrázku 3.3, z kterého je patrné, že v čase $t = 192\,\mathrm{h}$ na sebe tyto měření nenavazují. Zároveň jsou jejich jejich střední hodnoty a rozsahy rozdílné. Z toho důvodu budou pro další zpracování standardizovány.

Z průběhů 3.3 je patrné, že data ze senzorů 1 a 2, stejně tak 3 až 4, jsou vysoce korelovaná, což také potvrzuje teplotní mapa na obrázku 3.4. To bude pravděpodobně ovlivněno jejich fyzickým umístěním. Díky takto vysoké korelaci mezi jednotlivými pozorovanými vlastnostmi je možné využití relativně jednoduchého modelu, který i přesto bude schopen zachytit všechny potřebné informace.

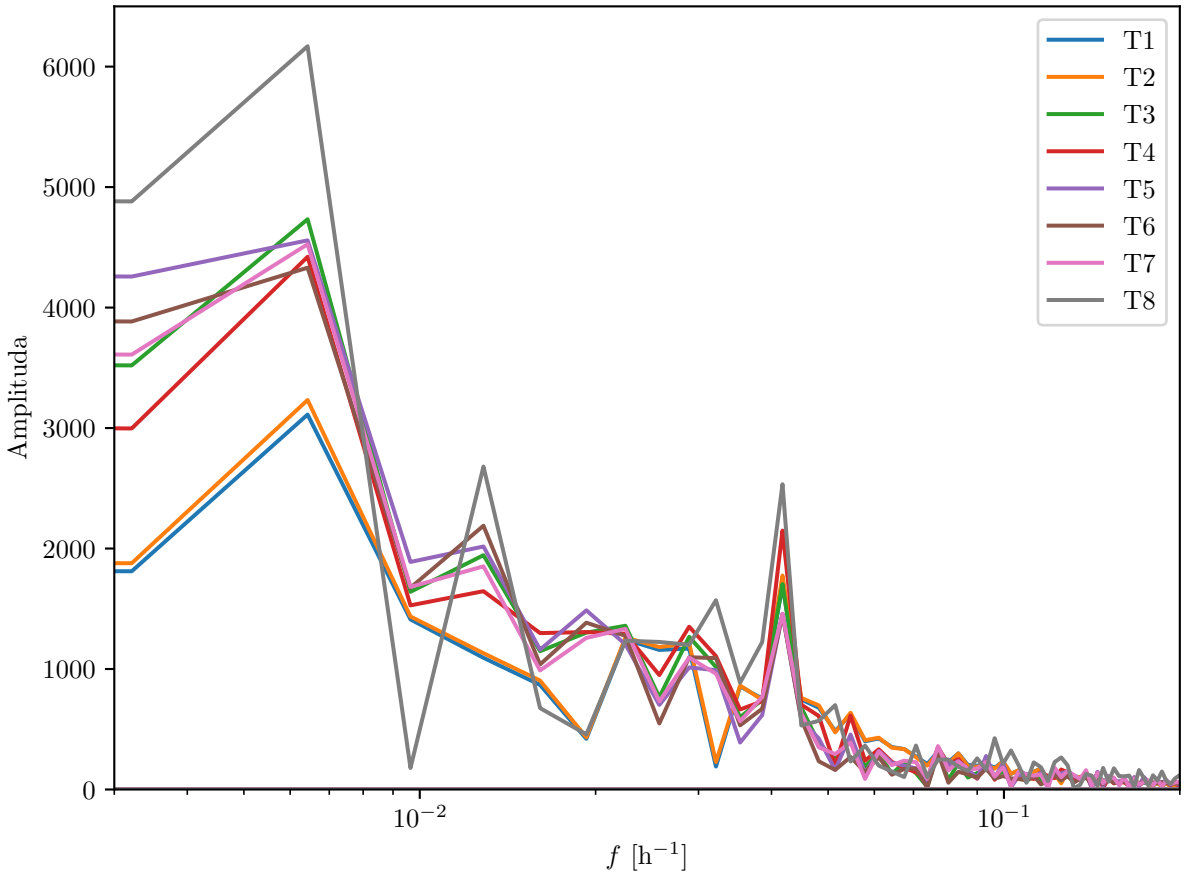
Převedením teplotních průběhů z časové oblasti do frekvenční oblasti pomocí Fourierovy transformace je možné pozorovat zastoupené frekvence. Z obrázku 3.5, kde je frekvenční průběh zobrazen v logaritmickém měřítku, jsou patrná zvýšení amplitudy u několika frekvencí. Nejjednotnější zvětšení je při frekvenci $f = 0.0418\,\mathrm{h}^{-1}$, která délkou periody odpovídá jednomu

Obrázek 3.3: Průběh teplot na osmi místech obráběcího stroje

dni. Maximální amplituda je při frekvenci přibližně $f = 0.0064\,\mathrm{h}^{-1}$, která svou délkou periody odpovídá šesti a půl dni, tedy necelému týdnu. Další významnější periodicitu vykazuje teplota ze senzoru 8 a to přibližně po třech dnech a sedmi hodinách.



Obrázek 3.4: Teplotní mapa zobrazující korelaci mezi jednotlivými signály



Obrázek 3.5: Frekvence obsažené v průbězích teplot

3.3 Tvorba a učení modelů neuronové sítě

Před započetím tvorby a učení modelů je nutné, aby byla data, popsaná v kapitole 3.2, rozdělena na trénovací a validační. Testovací dataset je opomenut jelikož uvedené modely nebudou v této podobě nikdy nasazeny v provozu. Pro testování tedy bude použit validační soubor, který je pro porovnávání výkonnosti jednotlivých modelů dostatečný. Před reálným použitím by ale bylo nutné testovací dataset obstarat.

Jelikož se jedná o sekvenční data, trénovací i validační dataset je pro použití učení s učitelem nutné rozdělit na jednotlivé vzorky tak, aby každý vzorek obsahoval vstupní sekvenci a anotaci. Anotace je tvořena sekvencí stejné délky, jako je ta vstupní, přičemž obě vzniknou rozdělením sekvence dvojnásobné délky. Anotace je tedy pokračováním vstupní sekvence.

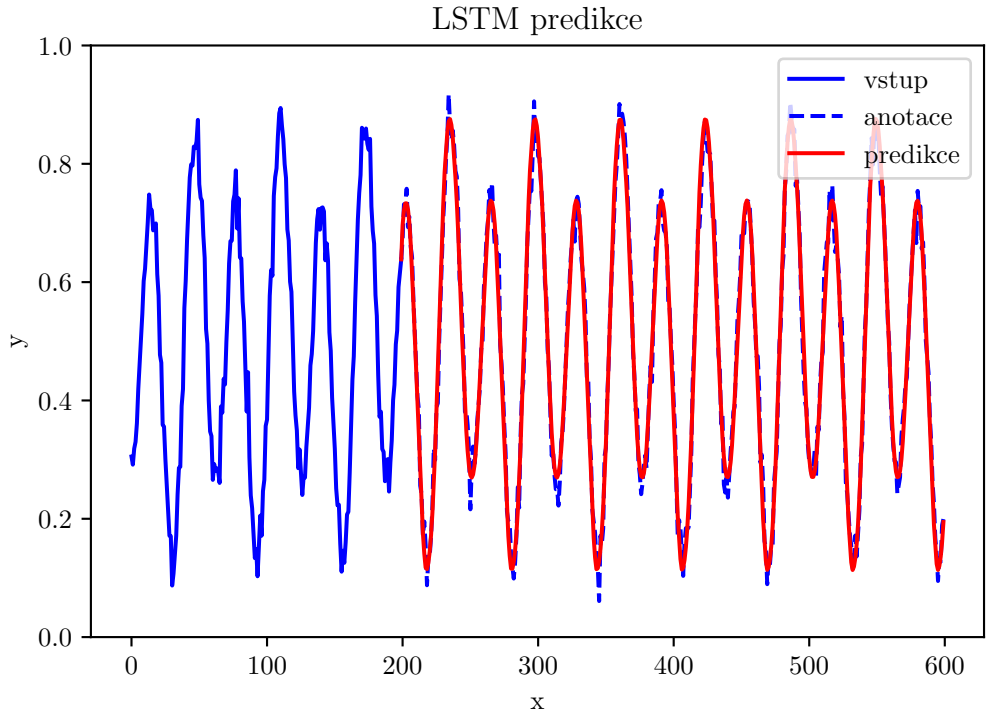
Délka vstupů i anotací byla zvolena 80 hodnot, přičemž jednotlivé vzorky jsou vůči sobě posunuty o 20 hodnot. Díky tomu jsou využita všechna data v trénovacím souboru (kromě prvních 80 hodnot nerozdělené trénovací sekvence) k úpravě parametrů. Pokud by se vzorky nepřekrývaly, nebyla by na hodnotách obsažených pouze ve vstupních sekvencích počítána chyba predikce a tedy by se na úpravě parametrů tyto části nepodílely, sloužily by pouze k vytvoření historie pro predikci anotací.

Tím, že byla měření provedena v různá období, a tedy na sebe ani po standardizaci nenavazují, nesmí se jejich přechod objevit v žádném vzorku a zároveň musí být validační soubor vytvořen ze souvislé sekvence, aby bylo možné pomocí něj model testovat. Je tedy možné trénovací a validační dataset rozdělit buď v poměru $8:5$, v tom případě by ale poměr trénovacího souboru byl menší, než je obvyklé. Zároveň, v případě, že by se charakteristika průběhů změnila na základě ročního období, nebyl by trénovací soubor dostatečně reprezentativní. Z toho důvodu budou datasety rozděleny opět v poměru $7:3$, ale bude zajištěno, aby nebyl přechod mezi měřeními uprostřed žádného vzorku.

3.3.1 Využití vlastní implementace

Během vytváření byla tato implementace průběžně úspěšně testována na jednoduchých generovaných signálech pro ověření její funkčnosti. Při snaze o trénování na reálných datech z měření teplot obráběcího stroje se ale ukázalo, že implementace v nich není během trénování schopna zachytit potřebné vzorce a i přes výrazné snížení chyby nákladové funkce během učení není model schopen konvergovat k optimálnímu řešení a predikovat přijatelně přesný výstup. Tomu nepřispívá ani fakt, že je celková doba nutná k učení velmi dlouhá, jelikož celý výpočet běží pouze na jednom procesorovém jádru. Z tohoto důvodu je také velmi obtížné použít dropout, jelikož by model musel být dostatečně složitý, aby vlivem dropout regularizace nedocházelo k nedoučení. Přidáváním dalších neuronů do vrstvy ale exponenciálně roste počet parametrů modelu a tím i doba učení.

Příklad predikovaného jednoduchého zašuměného signálu se vzorkováním 0.2 (jelikož se jedná o generovaný signál, nejsou uvedeny jednotky) je na obrázku 3.6. Takto naučený model dosahoval střední kvadratické chyby predikce všech časových kroků přibližně 0.001 (nutno podotknout, že jediný rozdíl mezi trénovacím a validačním souborem byl vlivem šumu a náhodného fázovéhu posuvu). Zvolenou architekturou byla LSTM se dvěma skrytými vrstvami po 32 neuronech. Aktivační funkce pro vstup a výstup LSTM bloků byla hyperbolický tangens a pro výpočet aktivací bran byla zvolena logistická funkce. Výstupní vrstva byla tvořena jedním perceptronem, také s logistickou funkcí jakožto aktivační funkcí. Parametry byly inicializovány náhodně s rovnoměrným rozložením v intervalu $(-0.1; 0.1)$. Tento natrénovaný model je obsažen v archivu přiloženém k této práci.



Obrázek 3.6: Predikce jednoduchého signálu pomocí vlastní implementace neuronové sítě

Příkladem inicializace modelu může být např.:

```
model = NeuralNetwork(n_inputs=8, loss_type='se', init_par_range=[-0.1, 0.1],
  layers_params=[
  {"Layer_type": 'lstm', "n_units": 32,
      "activations": {"activ_out":'tanh', "activ_in":'tanh', "activ_gate":'sigmoid'}}
  {"Layer_type": 'gru', "n_units": 32,
    "activations": {"activ_out":'tanh', "activ_gate":'sigmoid'}},
  {"Layer_type": 'mlp', "n_units": 8,
    "activations": {"activ_out":'sigmoid'}}])
```

kde je vytvořen model s osmi neurony ve vstupní vrstvě, první skrytou LSTM vrstvou s 32 neurony, druhou skrytou vrstvou GRU také s 32 neurony a 8 perceptrony ve výstupní vrstvě. Všechny vrstvy mají také specifikovány aktivační funkce. Architektury LSTM i GRU jsou v jednom modelu obsaženy pouze pro příklad. Nicméně takový model je funkční.

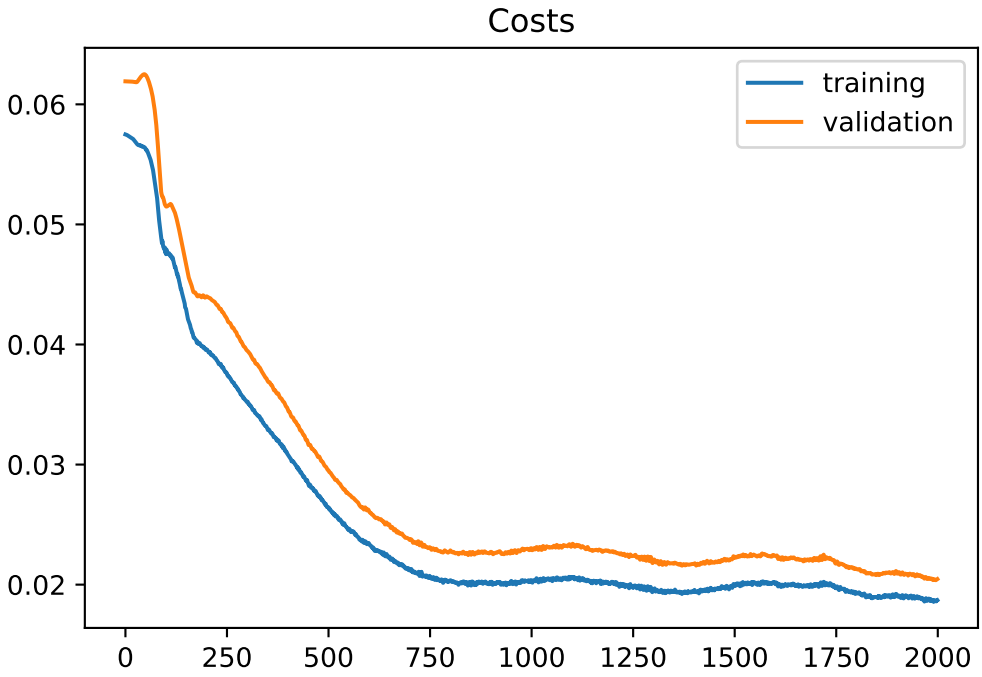
Příkladem zahájení trénování modelu je:

```
model.train(data_tra, data_val, epochs=300, optimization_alg="adam",
  learning_rate=0.001, batch_size=4, display_update=10)
```

kde `data_tra` a `data_val` jsou trénovací a validační dataset upravené do podoby popsané na začátku podkapitoly 3.3. Dále je zvolen počet epoch, optimalizační algoritmus, koeficient učení, velikost dávky – ta je nízká z důvodu výpočetní doby (zvětšování dávky zlepšuje stabilitu trénování, ale úměrně zpomaluje aktualizaci parametrů a tím zvyšuje dobu nutnou k naučení modelu) a počet epoch po kterých se vypíše průběžný stav učení.

Při použití implementace na reálných datech se po několika desítkách časových kroků predikce zastaví na konstantní hodnotě. Implementace zřejmě přes vysoké frekvence šumu není schopna postihnout nižší frekvence, které mají charakter změn teplot.

Na obrázku 3.7 je vudět průběh vývoje nákladu.



Obrázek 3.7: Průběh vývoje nákladu během trénování s použitím vlastní implementace

3.3.2 Využití frameworku TensorFlow

Z důvodu nemožnosti plně optimalizovat vlastní implementaci vlivem časového omezení při zpracovávání práce aby byla použitelná pro predikci reálných dat, bude v této části práce provedena predikce pomocí frameworku TensorFlow, pomocí kterého budou natrénovány a porovnány modely architektur LSTM a GRU. Pro porovnání bude uveden i model jednoduché rekurentní neuronové sítě. Jelikož má každá architektura jiný počet parametrů v neuronech, bude jako ukazatel složitosti použit celkový počet parametrů, namísto pouhého počtu neuronů.

V tabulce 3.1 je uvedeno porovnání modelů architektur RNN, LSTM a GRU s jednou skrytou vrstvou a různým počtem neuronů. Trénování trvalo patnáct epoch a velikost dávky byla 32. Jako optimalizační algoritmus byl na základě rešerše zvolen Adam s koeficientem učení 0,001, pro výpočet ztráty byla použita Huberova ztrátová funkce a aktivační funkce u všech modelů byla je hyperbolický tangens. Výpočet chyby predikce byl proveden výpočtem střední kvadratické chyby mezi skutečným a predikovaným průběhem pro všech osm teplot, které byly následně zprůměrovány. Jako výstupní prstva je použita architektura Dense, která odpovídá perceptronu.

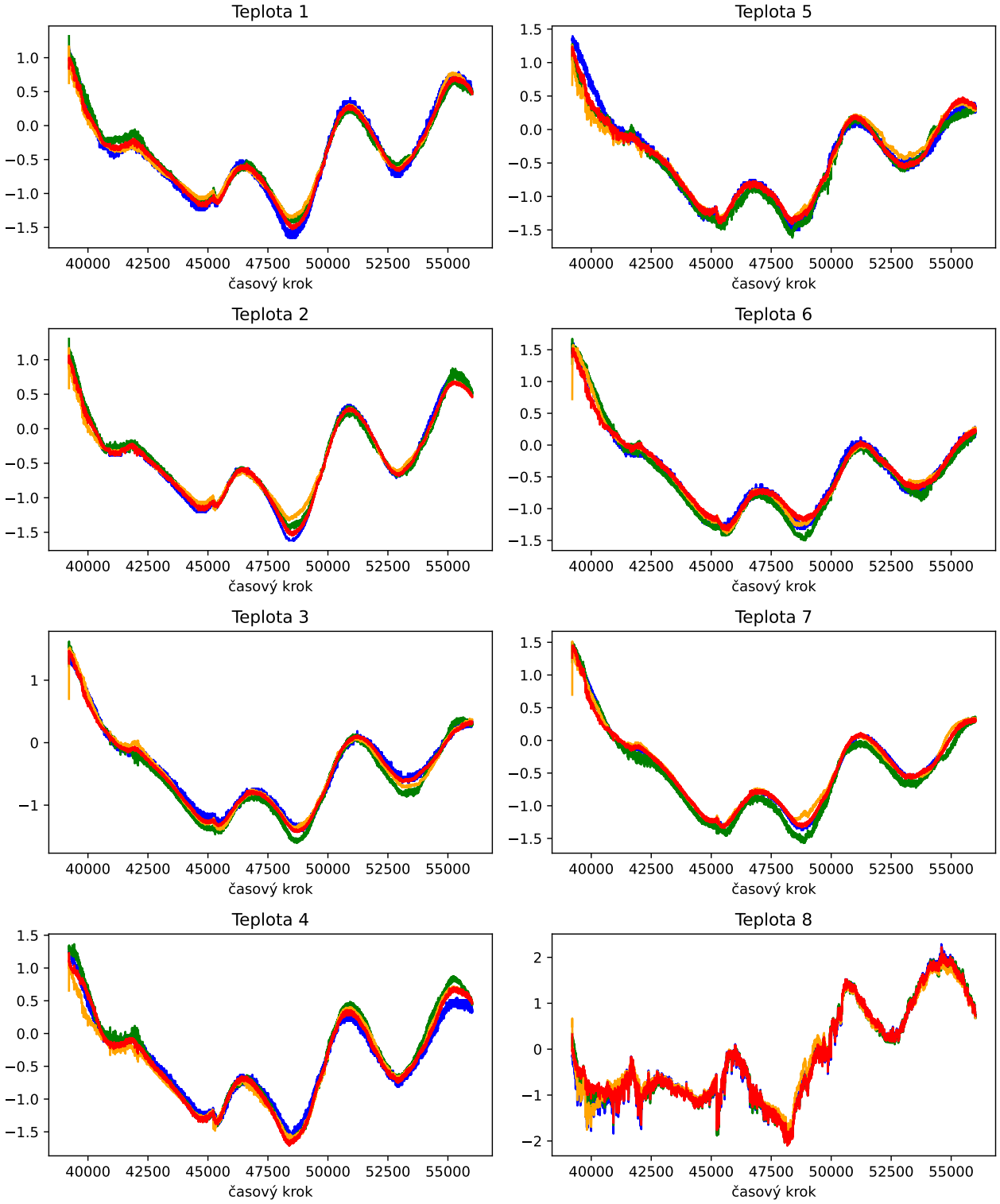
Z tabulky 3.1 je dále patrné, že nejlepších výsledků dosahuje architektura GRU. To je do značné míry způsobeno tím, že oproti LSTM má méně parametrů a tedy při relativně nízkém počtu epoch konvergovalo k optimálnímu řešení rychleji.

Na obrázku 3.8 je zobrazena predikce modelů se dvěma skrytými vrstvami. Optimalizační algoritmem použitým pro trénování byl opět Adam a ztrátovou funkcí Buberova funkce. Aktivační funkcí je ReLU, jelikož při větším počtu vrstev je neuronová síť náchylnější k mizejícímu

Tabulka 3.1: Porovnání modelů s jednou skrytou vrstvou

architektura	p. neuronů	p. parametrů	náklad (učení)	náklad (validace)	chyba predikce
RNN	8	208	0,0188	0,0237	0,025186
LSTM	8	616	0,0140	0,0189	0,030250
GRU	8	504	0,0121	0,0171	0,034832
RNN	16	536	0,0057	0,0068	0,009511
LSTM	16	1736	0,0049	0,0063	0,016893
GRU	16	1384	0,0035	0,0044	0,012432
RNN	32	1576	0,0022	0,0026	0,006059
LSTM	32	5512	0,0024	0,0032	0,012127
GRU	32	4296	0,0014	0,0016	0,008504
RNN	64	5192	0,00079	0,00087	0,002144
LSTM	64	19208	0,00160	0,00220	0,00392
GRU	64	14728	0,00064	0,00074	0,001512

gradientu.



Obrázek 3.8: Predikce standardizovaných teplot obráběcího stroje s modely se dvěma skrytými vrstvami se 32 neurony (modrá – anotace, zelená – RNN, oranžová – LSTM, červená – RNN)

4 Závěr

Tato bakalářská práce se zabývala tématem dlouhodobé predikce nelineárních dynamických systémů pomocí rekurentních neuronových sítí.

V rámci této práce byla vytvořena vlastní implementace neuronové sítě v jazyce Python pomocí teorie a matematického popisu uvedeného v provedené rešerši. Ačkoliv má tato implementace limity v rychlosti výpočtu a nepodařilo se pomocí ní natrénovat model na reálných datech pocházejících z měření teplot obráběcího stroje, s použitím jednodušších signálů bylo prokázáno, že je implementace funkční. Dalším pokračováním této práce by tak mohla být optimalizace a rozšíření této implementace. Je ale nutné si uvědomit, že vlastní softwarové řešení s použitím pouze těch nejzákladnějších knihoven pro matematické operace se svým výkonem nemůže rovnat s existujícíc profesionálně vyvýjenou knihovnu strojového učení. Přesto je však vytvoření vlastní implementace přínosné zejména pro pochopení vnitřní struktury neuronových sítí.

V další části byla k predikci použita knihovna TensorFlow. Z uvedených výsedků se jeví jako nejvhodnější použití architektury GRU. To je ale z části ovlivněno nižším počtem epoch během trénování a proto došlo ke konvergenci architektur s nižším počtem parametrů rychleji. Nicméně je z uvedených grafů zřejmé, že bylo možné dosáhnout přijatelné přesnosti.

Jelikož použitá data neobsahovala informace o provozním stavu obráběcího stoje, pouze na základě času a historických průběhů nelze predikovat skutečný vývoj teplot. Pokračováním této práce může být další zkoumání predikce s využitím těchto dodatečných informací.

Seznam použitých zkratek

MLP	multilayer perceptron
RNN	recurrent neural network
LSTM	long short-term memory
GRU	gated recurrent unit
MAE	mean absolute error
MSE	mean square error

Seznam obrázků

Seznam tabulek

Seznam použité literatury

1. *Machine learning, explained | MIT Sloan* [online]. MIT Sloan, 2023 [cit. 2023-02-15]. Dostupné z: `https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained`.
2. GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems.* Second edition. Beijing: O'Reilly, 2019. ISBN 978-1492032649.
3. GUPTA, Vratika; MISHRA, Vinay Kumar; SINGHAL, Priyank; KUMAR, Amit. An Overview of Supervised Machine Learning Algorithm [online]. 2022, s. 87–92 [cit. 2023-02-15]. Dostupné z DOI: `10.1109/SMART55829.2022.10047618`.
4. GRAVES, Alex. *Supervised sequence labelling with recurrent neural networks* [online]. Preprint. New York: Springer, 2012 [cit. 2023-02-15]. ISBN 3642247962. Dostupné z: `https://www.cs.toronto.edu/~graves/preprint.pdf`.
5. BURKOV, Andriy. *The hundred-page machine learning book.* Quebec: Andriy Burkov, 2019. ISBN 978-1999579500.
6. D'AGOSTINO, Andrea. *How To Prepare Data For Machine Learning* [online]. Towards Data Science, 2023 [cit. 2023-04-30]. Dostupné z: `https://towardsdatascience.com/how-to-prepare-data-for-machine-learning-eb9d9973832f`.
7. *Normalization | Machine Learning | Google Developers* [online]. Google for Developers, 2022 [cit. 2023-04-30]. Dostupné z: `https://developers.google.com/machine-learning/data-prep/transform/normalization`.
8. Data Preparation in Neural Network Data Analysis. In: *Foreign-Exchange-Rate Forecasting With Artificial Neural Networks* [online]. Boston, MA: Springer US, 2007, s. 39–62 [cit. 2023-03-06]. ISBN 978-0-387-71719-7. Dostupné z DOI: `10.1007/978-0-387-71720-3_3`.
9. VIDIYALA, Ramya. *Normalization vs Standardization* [online]. Towards Data Science, 2020 [cit. 2023-04-30]. Dostupné z: `https://towardsdatascience.com/normalization-vs-standardization-cb8fe15082eb`.

10. SCHMIDT, Robin M. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview [online]. 2019 [cit. 2023-02-20]. Dostupné z arXiv: 1912.05911.

11. LECUN, Yann; BENGIO, Yoshua. Convolutional Networks for Images, Speech, and Time Series. In: *The Handbook of Brain Theory and Neural Networks* [online]. Cambridge, MA, USA: MIT Press, 1998, s. 255–258 [cit. 2023-02-20]. ISBN 0262511029.

12. ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* [online]. 1958, roč. 65 6, s. 386–408 [cit. 2023-02-18].

13. ZHANG, Aston; LIPTON, Zachary C.; LI, Mu; SMOLA, Alexander J. *Dive into Deep Learning* [online]. 2023. [cit. 2023-02-20]. Dostupné z arXiv: 2106.11342.

14. SHARMA, Sagar; SHARMA, Simone; ATHAIYA, Anidhya. Activation functions in neural networks. *Towards Data Sci* [online]. 2017, roč. 6, č. 12, s. 310–316 [cit. 2023-02-19].

15. NWANKPA, Chigozie; IJOMAH, Winifred; GACHAGAN, Anthony; MARSHALL, Stephen. Activation Functions: Comparison of trends in Practice and Research for Deep Learning [online]. 2018 [cit. 2023-02-19]. Dostupné z arXiv: 1811.03378.

16. SALEHINEJAD, Hojjat; SANKAR, Sharan; BARFETT, Joseph; COLAK, Errol; VALAEE, Shahrokh. Recent Advances in Recurrent Neural Networks [online]. 2018 [cit. 2023-02-25]. Dostupné z arXiv: 1801.01078.

17. KRISHNAMURTHY, Kamesh; CAN, Tankut; SCHWAB, David J. Theory of Gating in Recurrent Neural Networks. *Phys. Rev. X* [online]. 2022, roč. 12, s. 011011 [cit. 2023-02-22]. Dostupné z DOI: 10.1103/PhysRevX.12.011011.

18. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. *Neural Computation* [online]. 1997-11-01, roč. 9, č. 8, s. 1735–1780 [cit. 2023-02-28]. ISSN 0899-7667. Dostupné z DOI: 10.1162/neco.1997.9.8.1735.

19. SAK, Haşim; SENIOR, Andrew; BEAUFAYS, Françoise. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition [online]. 2014-02-01 [cit. 2023-02-28]. Dostupné z DOI: 10.48550/arXiv.1402.1128.

20. GERS, F.A. Learning to forget: continual prediction with LSTM. In: *9th International Conference on Artificial Neural Networks: ICANN '99* [online]. Edinburg: IEE, 1999, s. 850–855 [cit. 2023-03-05]. ISBN 0-85296-721-7. ISSN 0537-9989. Dostupné z DOI: 10.1049/cp:19991218.

21. CHO, Kyunghyun; MERRIENBOER, Bart van; GULCEHRE, Caglar; BAHDANAU, Dzmitry; BOUGARES, Fethi; SCHWENK, Holger; BENGIO, Yoshua. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation [online]. 2014 [cit. 2023-03-01]. Dostupné z arXiv: 1406.1078.

22. CHUNG, Junyoung; GULCEHRE, Caglar; CHO, KyungHyun; BENGIO, Yoshua. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [online]. 2014 [cit. 2023-03-01]. Dostupné z arXiv: 1412.3555.

23. ALLWRIGHT, Stephen. *MSE vs MAE, which is the better regression metric?* [online]. Stephen Allwright, 2023 [cit. 2023-02-27]. Dostupné z: https://stephenallwright.com/mse-vs-mae/.

24. GOKCESU, Kaan; GOKCESU, Hakan. Generalized Huber Loss for Robust Learning and its Efficient Minimization for a Robust Statistics [online]. 2021 [cit. 2023-03-01]. Dostupné z arXiv: 2108.12627.

25. LEUNG, H.; HAYKIN, S. The complex backpropagation algorithm. *IEEE Transactions on Signal Processing* [online]. 1991, roč. 39, č. 9, s. 2101–2104 [cit. 2023-02-22]. Dostupné z DOI: 10.1109/78.134446.

26. WERBOS, P.J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* [online]. 1990, roč. 78, č. 10, s. 1550–1560 [cit. 2023-03-05]. Dostupné z DOI: 10.1109/5.58337.

27. PASCANU, Razvan; MIKOLOV, Tomas; BENGIO, Yoshua. On the difficulty of training Recurrent Neural Networks [online]. 2013 [cit. 2023-05-08]. Dostupné z arXiv: 1211.5063.

28. RUDER, Sebastian. An overview of gradient descent optimization algorithms [online]. 2017 [cit. 2023-02-23]. Dostupné z arXiv: 1609.04747.

29. LEE, Thomas; GASSWINT, Greta; HENNING, Elizabeth. *Momentum - Cornell University Computational Optimization Open Textbook - Optimization Wiki* [online]. Cornell University Computational Optimization Open Textbook, 2021 [cit. 2023-04-04]. Dostupné z: https://optimization.cbe.cornell.edu/index.php?title=Momentum.

30. KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization [online]. 2017 [cit. 2023-04-10]. Dostupné z arXiv: 1412.6980.

31. JABBAR, H; KHAN, Rafiqul Zaman. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices* [online]. 2015, roč. 70, s. 163–172 [cit. 2023-02-27].

32. YING, Xue. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series* [online]. 2019, roč. 1168, č. 2, s. 022022 [cit. 2023-02-27]. Dostupné z DOI: 10.1088/1742-6596/1168/2/022022.

33. TEWARI, Ujwal. *Regularization — Understanding L1 and L2 regularization for Deep Learning* [online]. Medium, 2021 [cit. 2023-04-10]. Dostupné z: https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf.

34. VUNGARALA, Seshu Kumar. *Lasso and Ridge Regression — Regularization Techniques Explained for Beginners* [online]. Medium, 2023 [cit. 2023-04-10]. Dostupné z: https://medium.com/@seshu8hachi/lasso-and-ridge-regression-regularization-techniques-explained-for-beginners-be568ac6c0a3.

35. LOSHCHILOV, Ilya; HUTTER, Frank. Decoupled Weight Decay Regularization [online]. 2019 [cit. 2023-04-10]. Dostupné z arXiv: 1711.05101.

36. ZOU, Hui; HASTIE, Trevor. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology* [online]. 2005, roč. 67, č. 2, s. 301–320 [cit. 2023-04-10]. ISSN 1369-7412. Dostupné z DOI: 10.1111/j.1467-9868.2005.00503.x.

37. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* [online]. 2014, roč. 15, č. 56, s. 1929–1958 [cit. 2023-05-07]. Dostupné z: http://jmlr.org/papers/v15/srivastava14a.html.

38. GAL, Yarin; GHAHRAMANI, Zoubin. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks [online]. 2016 [cit. 2023-05-07]. Dostupné z arXiv: 1512.05287.

39. LI, Mingchen; SOLTANOLKOTABI, Mahdi; OYMAK, Samet. Gradient Descent with Early Stopping is Provably Robust to Label Noise for Overparameterized Neural Networks. In: CHIAPPA, Silvia; CALANDRA, Roberto (ed.). *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* [online]. PMLR,

2020, sv. 108, s. 4313–4324 [cit. 2023-05-08]. Proceedings of Machine Learning Research. Dostupné z: http://proceedings.mlr.press/v108/li20j/li20j.pdf.