VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY

INSTITUTE OF MATHEMATICS

VYUŽITÍ NEURONOVÝCH SÍTÍ PRO PREDIKCI HODNOT A DETEKCI ANOMÁLIÍ V SÍŤOVÝCH DATECH

USING NEURAL NETWORKS FOR FORECASTING AND DETECTION OF ANOMALY DATA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Zdeněk Fiala

AUTHOR

VEDOUCÍ PRÁCE — Ing. Pavla Sehnalová, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav matematiky
Student:	**Bc. Zdeněk Fiala**
Studijní program:	Matematické inženýrství
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Pavla Sehnalová, Ph.D.**
Akademický rok:	2023/24

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 diplomové práce:

Využití neuronových sítí pro predikci hodnot a detekci anomálií v síťových datech

Stručná charakteristika problematiky úkolu:

Cílem práce je aplikování neuronových sítí na data ze síťového provozu za účelem predikce hodnot. Návrh jednoduché NN s učením a následným testováním na reálných datech v počítačové síti bude implementován jako jednoduchý program a otestován na různých typech dat. Pro srovnání výsledků bude pro původní data provedena i regresní analýza.

Cíle diplomové práce:

Výstupem práce je program/prototyp, který bude implementovat jednoduchou neuronovou síť, která pro vstupní data ve formátu časová značka+hodnota predikuje následující hodnotu.
Součástí prototypu bude i porovnání skutečné a predikované hodnoty a vyhodnocení odchylky jako anomálie.

Seznam doporučené literatury:

BROCKWELL, P. J. and DAVIS, R. A. Introduction to time series and forecasting. Springer Cham, 2016. eBook ISBN 978-3-319-29854-2.

HAMILTON, J. D. Time series analysis. Princeton University Press, 1994. ISBN 978-0-691-04289-3.

HAHN, G. J. and SHAPIRO, S. S. Statistical models in engineering. John Wiley & Sons, 1994. ISBN 0-471-04065-7.

KONG, W., DONG, Z. Y., JIA, Y., HILL, D. J., XU, Y. and ZHANG, Y. Short-term residential load forecasting based on LSTM recurrent neural network. IEEE Transactions on Smart Grid, 10(1), pp. 841-851, 2017.

RAFF, E. Inside Deep Learning: Math, Algorithms, Models. Manning Publications, 2022. ISBN 9781617298639.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

doc. Mgr. Petr Vašík, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

Abstrakt

Diplomová práce se zabývá predikcí dat pomocí neuronové sítě a detekcí anomálií v síťových datech. V práci je sestaven model neuronové sítě pro predikci časových řad, který je otestován na reálných datech. Následně je predikce využita při detekování anomálií v síťových datech. Výsledky neuronové sítě jsou následně porovnány s regresní analýzou dat.

Abstract

The thesis deals with data forecasting using neural network and anomaly detection in network data. In this thesis, a neural network model for time series forecasting is constructed and tested on real data. Subsequently, the forecasting is used in detecting anomalies in network data. The neural network results are then compared with regression analysis of the data.

Klíčová slova

neuronová síť, LSTM, predikce časových řad, detekce anomálií, regresní analýza, Python

Keywords

neural nework, LSTM, time series forecasting, anomaly detection, regression analysis, Python

FIALA, Zdeněk. *Využití neuronových sítí pro predikci hodnot a detekci anomálií v síťových datech* [online]. Brno, 2024 [cit. 2024-05-23]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/154109. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav matematiky. Vedoucí práce Pavla Sehnalová.

Prohlašuji, že jsem diplomovou práci *Využití neuronových sítí pro predikci hodnot a detekci anomálií v síťových datech* vypracoval samostatně pod vedením Ing. Pavly Sehnalové, Ph.D. s použitím materiálů uvedených v seznamu použité literatury.

Zdeněk Fiala

Děkuji své vedoucí Ing. Pavle Sehnalové, Ph.D. za cenné rady, připomínky a čas věnovaný konzultacím k této diplomové práci.

Zdeněk Fiala

Obsah

1 Úvod

Popularita umělé inteligence každým rokem roste a uplatňuje se v stále širším okruhu odvětví. Jedním z důvodů je nárůst dostupných dat a rychlý vývoj výpočetní kapacity, což vytváří ideální podmínky pro vývoj v oblasti umělé inteligence a strojového učení. Jednou z nejvýznamnějších technologií umělé inteligence jsou neuronové sítě, inspirované strukturou a funkcí lidského mozku. Jejich schopnost učit se rozpoznávat složité vzory v datech přináší revoluční možnosti v širokém spektru aplikací.

Obecně dělíme neuronové sítě do dvou skupin na konvoluční neuronové sítě CNN a rekurentní neuronové sítě RNN. Konvoluční neuronové sítě jsou navrženy pro práci s tabulkovými nebo mřížovými daty a nejvíce se využívají u analýzy, klasifikace a rozpoznávání obrazu. Rekurentní neuronové sítě slouží k práci se sekvenčními daty jako jsou časové řady nebo text. Jejich charakteristikou je uchovávání dlouhodobých závislostí a vzorů chování v sekvenčních datech. V této práci se budeme zabývat predikcí reálných síťových dat pomocí rekurentních neuronových sítí. Následně využijeme predikce vygenerované neuronovou sítí pro detekci anomálií v síťových datech. Dalším předmětem práce bude porovnání výsledků dosažených umělou inteligencí s výsledky regresní analýzy dat.

S rostoucím výpočetním výkonem také narůstají nároky na komplexnost řešení a obecnost modelů neuronové sítě. Proto se vytváří velké a složité obecné modely, které nejsou konstruovány pro určité aplikace, ale jako počáteční forma pro pozdější využití na konkrétní problematiku. S tím souvisí využívání složitých modelů na problémy, které by mohly být řešeny jednodušším modelem s konstrukcí přizpůsobenou konkrétnímu problému. V této práci se pokusíme sestavit jednoduchý model konstruovaný pro řešení konkrétní problematiky, který bude splňovat naše požadavky s co nejnižší časovou a výpočetní náročností.

Práce je rozdělena do čtyř hlavních částí. V první části uvedeme úvod k neuronovým sítím, popíšeme základní strukturu a fungování neuronové sítě. Dále představíme pokročilejší model LSTM neuronové sítě, který vyhovuje naší aplikaci. Druhá část obsahuje využití teoretických poznatků při tvorbě neuronové sítě pro predikci síťových dat. Následuje implementace v programovacím jazyce Python a následné testování jednotlivých parametrů pro konstrukci optimálního modelu neuronové sítě.

Ve třetí části využíváme navržený model neuronové sítě, který použijeme při detekci anomálií v jednotlivých datasetech. Zde popíšeme dvě vytvořené metody pro detekci anomálií, jednu zaměřující se na úsporu časové náročnosti a druhou na co nejvyšší přesnost při detekci anomálií. V poslední části nadefinujeme potřebnou teorii regresní analýzy a predikce dat pro porovnání s neuronovou sítí. Následně provedeme predikci časové řady a detekci anomálií pomocí statistických metod a porovnáme výsledky a celkový proces obou přístupů.

2 Neuronová síť

Neuronová síť (Neural Network), často nazývaná také umělá neuronová síť (Artificial Neural Network), označuje matematický model inspirovaný fungováním lidského mozku. Tato modelová struktura je navržena tak, aby simulovala způsob, jakým biologické neurony vzájemně spolupracují při zpracování informací. Tyto sítě mají schopnost učit se a rozpoznávat vzory v chování, což umožňuje řešení složitých úloh, které by jinak byly obtížné nebo nemožné pro tradiční algoritmy. Poznatky v této kapitole vychází z [1], [2], [4], [5], [7].

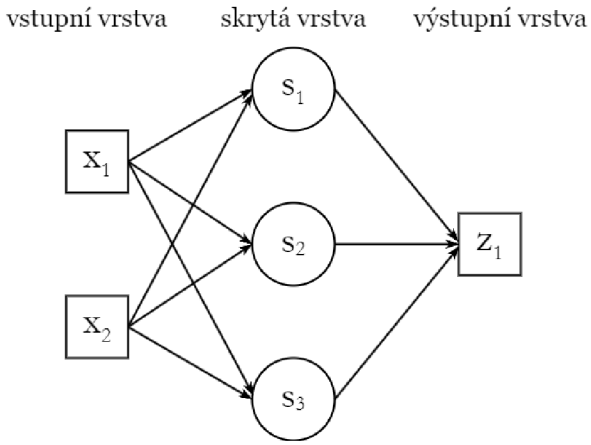
Obecně dělíme neuronové sítě na konvoluční neurnové sítě CNN a rekurentní neuronové sítě RNN. V této kapitole vysvětlíme základy stavby a principy fungování neuronových sítí, využívané ve všech složitějších konstrukcích neuronových sítí, které budeme v dalších kapitolách rozvíjet.
Neuronovou síť si můžeme představit jako orientovaný graf, kde každý uzel představuje matematickou operaci a každá orientovaná hrana určuje vstupy a výstupy mezi jednotlivými uzly. V jazyce neuronových sítí se základní struktura skládá ze dvou stavebních částí:

- neuronů,
- vazeb.

Kombinací neuronů a jejich vzájemných vazeb tvoříme strukturu neuronové sítě, kdy je zvykem, že neuronová síť tvoří k-partitní úplný graf. To znamená, že neuronová síť obsahuje k disjunktních množin tak, že neurony v jednotlivých množinách mezi sebou nemají žádné vazby, ale jsou propojeny se všemi ostatními neurony z ostatních množin. Tyto disjunktní množiny označujeme jako vrstvy, které dělíme do 3 skupin:

- vstupní vrstva,
- skrytá vrstva,
- výstupní vrstva.

Obecně do neuronové sítě zadáváme n vstupních hodnot, které jsou reprezentovány vstupní vrstvou. Vstupy transformujeme ve skrytých vrstvách a výsledkem je m hodnot obsažených ve výstupní vrstvě. Obrázek 2.1 ukazuje jednoduchou neuronovou síť se dvěma vstupy, jednou skrytou vrstvou a jedním výstupem. Skryté vrstvy získaly svůj název díky jejich neuronům, které během výpočtů své výstupy neukazují. Počet skrytých vrstev lze rozšířit na základě požadavků modelu. S rostoucím počtem a velikostí vstupních hodnot je pro složitější zadání nutné rozšířit množství vrstev a neuronů, aby neuronová síť byla schopna popsat vzájemné vztahy a vzorce chování ve vstupních datech. Počet skrytých vrstev se označuje jako hloubka modelu. Neuronové sítě s hloubkou větší než 3 se označují jako *hluboké učení* (Deep Learning).



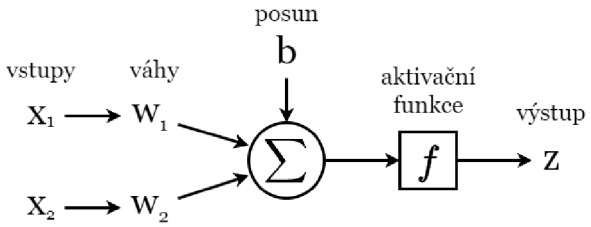
Obrázek 2.1: Jednoduchá neuronová síť

2.1 Neuron

Neuron je základní stavební jednotkou neuronové sítě. Tvoří vrstvy neuronové sítě a provádí výpočetní operace, které zpracovávají vstupní hodnoty vektoru $\boldsymbol{x} = (x_1, \ldots, x_n)$ do výstupního vektoru $\mathbf{z} = (z_1, \ldots, z_m)$ následujícím způsobem

$$z_k = f\left(\sum_{i=1}^{n}(w_{ik}x_i) + b_k\right),$$

kde matice $W = (w_{ik})$ pro $k = 1, \ldots, m$ označuje váhy příslušné vektoru vstupních hodnot $\boldsymbol{x}$, $\mathbf{b} = (b_1, \ldots, b_m)$ označuje vektor posunutí a f označuje výstupní aktivační funkci. Neuron může mít více vstupů, ale pouze jeden výstup. Schematický proces fungování neuronu je zobrazen na Obrázku 2.2.



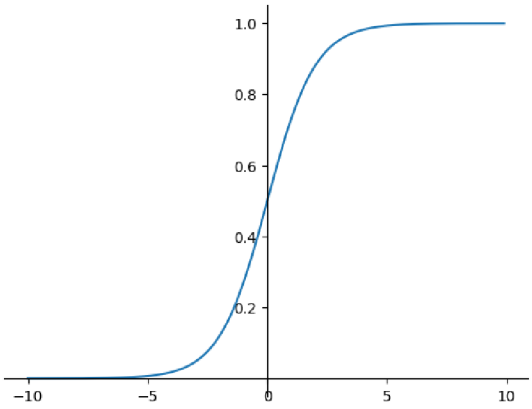
Obrázek 2.2: Model neuronu

2.1.1 Aktivační funkce

Výstupní aktivační funkce slouží k zavedení nelinearity do výstupu neuronů. Bez této úpravy by neuronová síť nedokázala řešit složitější nelineární problémy, byla by omezena pouze na úlohy s lineárním řešením. Pro různé aplikace používáme různé aktivační funkce, které mohou ovlivnit přesnost a fungování neuronové sítě.

Sigmoida
Jednou z nejpoužívanějších aktivačních funkcí je sigmoida, kterou budeme v následujícím textu značit $\sigma(x)$ a je dána předpisem

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad x \in \mathbb{R}. \tag{1}$$

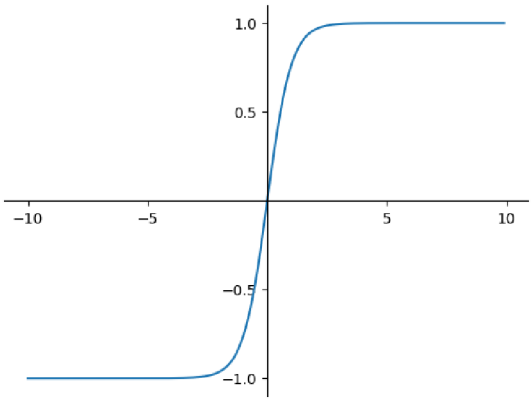


Obrázek 2.3: Sigmoida

Jak je vidět na obrázku 2.3, obor hodnot této funkce je omezený na intervalu $(0, 1)$. Této vlastnosti se využívá pro filtrování dat, kdy velkým záporným hodnotám připisujeme malou váhu a velké kladné hodnoty značně neovlivní řešení.

Tangens hyperbolický
Další běžně používaná aktivační funkce je tangens hyperbolický daný předpisem

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \quad x \in \mathbb{R}. \tag{2}$$



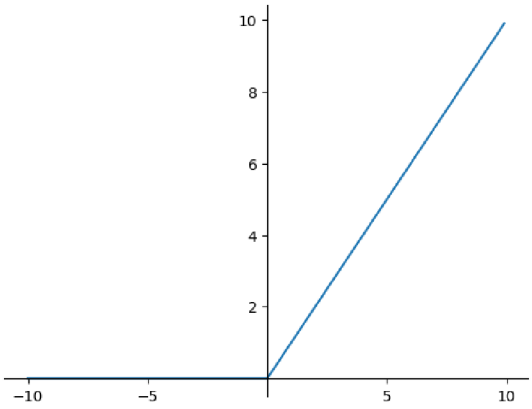
Obrázek 2.4: Tangens hyperbolický

ReLU
Jiným příkladem může být usměrňovací aktivační funkce zvaná ReLU (Rectified Linear Unit)

$$f(x) = max(0, x), \quad x \in \mathbb{R}, \tag{3}$$

která slouží k eliminaci vlivu záporných hodnot při řešení. Aktivační funkci ReLU můžeme vidět na obrázku 2.5.

Tyto funkce jsou využívány kvůli svým vlastnostem, hlavně kvůli existenci a omezenosti jejich derivací, což je klíčové pro optimalizaci parametrů neuronové sítě. Detailnější analýzu aktivačních funkcí najdete v [3]. V našem případě budeme v rámci neuronové



Obrázek 2.5: Usměrňovací lineární funkce ReLU

sítě využívat aktivační funkci sigmoidu a tangens hyperbolický, které budou klíčové pro architekturu neuronové sítě, kterou popíšeme v kapitole 2.2.1.

2.1.2 Backpropagation

Backpropagation (nebo také backward propagation) je algoritmus využívaný pro učení neuronových sítí. Učením rozumíme úpravu neznámých parametrů vah $\mathbf{w}$ a posunů $\mathbf{b}$ modelu. Jelikož tyto parametry při startu modelu neznáme, jsou voleny náhodně a metodou backpropagation se tyto parametry optimalizují. Následující algoritmus vychází z [8] a [9].

Předpokládejme neuronovou síť s N vrstvami, kde vrstva 0 je vstupní vrstva a vrstva N je výstupní vrstva. Algoritmus funguje následujícím způsobem:

1. Dopředný chod (Forward Pass):

   Pro každou vrstvu $a = 1, \ldots, N$ se spočítají výstupy $\mathbf{z}^{(a)}$ pro každý neuron ve vrstvě následujícím způsobem:

   $$\mathbf{z}^{(a)} = f(W^{(a)}\mathbf{x}^{(a)} + \mathbf{b}^{(a)})$$

   Výstup neuronové sítě $\mathbf{z}^{(N)}$ budeme pro zjednodušení značit $\mathbf{z}$.

2. Výpočet chyby ve výstupní vrstvě:

   Spočítá se chyba ve výstupní vrstvě N. V našem případě chybovou funkci E představuje střední kvadratická chyba:

   $$E = \frac{1}{m}\sum_{i=1}^{m}(z_i - \hat{z}_i)^2, \tag{4}$$

   kde z_i , pro $i = 1, \ldots, m$ představuje výstupy neuronové sítě a $\hat{z}_i$, pro $i = 1, \ldots, m$ označuje validační hodnoty.

3. Výpočet gradientů chybové funkce vzhledem k vahám w a posunům b:

   Pro každou vrstvu $a = 1 \ldots, N$ se vypočítají gradienty pro váhy a posuny následujícím způsobem:

   $$\frac{\partial E}{\partial \mathbf{w}^{(a)}} = \frac{\partial E}{\partial \mathbf{z}^{(a)}}\frac{\partial \mathbf{z}}{\partial \mathbf{w}^{(a)}} = \frac{\partial E}{\partial \mathbf{z}^{(a)}}\mathbf{x}^{(a)}$$
   $$\frac{\partial E}{\partial \mathbf{b}^{(a)}} = \frac{\partial E}{\partial \mathbf{z}^{(a)}}\frac{\partial \mathbf{z}}{\partial \mathbf{b}^{(a)}} = \frac{\partial E}{\partial \mathbf{z}^{(a)}}.$$

4. Aktualizace vah a posunů pomocí algoritmu optimalizace (např. metody gradientů):

   Pro každou vrstvu $a = 1 \ldots, N$ se aktualizují váhy a posuny následujícím způsobem:

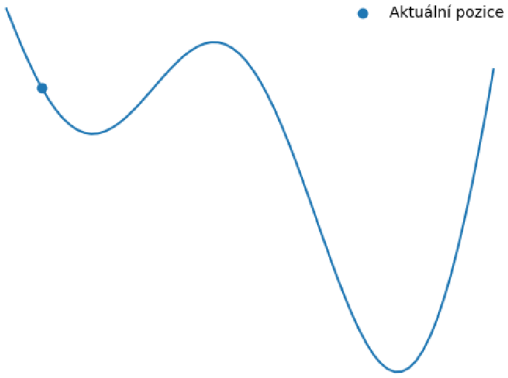
$$\mathbf{w}^{(a)} = \mathbf{w}^{(a)} - \alpha \frac{\partial E}{\partial \mathbf{w}^{(a)}}$$

$$\mathbf{b}^{(a)} = \mathbf{b}^{(a)} - \alpha \frac{\partial E}{\partial \mathbf{b}^{(a)}},$$

   kde $\alpha \in (0, 1)$ je míra učení (learning rate).

Volba parametru α záleží na dané úloze, bývá zvykem volit $\alpha = 0.01$ nebo $\alpha = 0.001$. Tímto postupem iteračně vypočítáme gradienty a aktualizujeme parametry sítě $\mathbf{w}$ a $\mathbf{b}$ tak, aby se minimalizovala chybová funkce E, což nazýváme učením neuronové sítě.

Na Obrázku 2.6 je ukázán příklad gradientní funkce neznámého parametru. Při volbě malého parametru α je učení sítě pomalé, a navíc algoritmus může skončit v lokálním minimu a globálního minima nikdy nedosáhne. Volbou velkého parametru α učení urychlíme, ale můžeme globální minimum úplně minout.



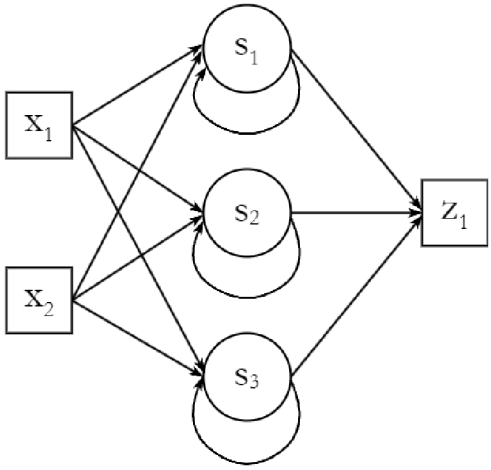
Obrázek 2.6: Příklad funkce gradientu neznámého parametru

2.2 Rekurentní neuronová síť

Neuronové sítě se obecně dělí na dvě velké skupiny, konvoluční neuronové sítě a rekurentní neuronové sítě. Konvoluční neuronové sítě se běžně používají pro řešení problémů obsahujících prostorová nebo tabulková data jako například obrázky. Rekurentní sítě se více hodí pro sekvenční a časové posloupnosti, například časové řady, protože na rozdíl od konvolučních sítí dokáží zachovávat dlouhodobější závislosti v chování dat. Navíc konvoluční modely mají přesně daný formát vstupních a výstupních dat, zatímco rekurentní sítě dovolují variabilitu vstupů, což znamená možnost libovolně měnit jejich dimenzi. Vzhledem k charakteru našich dat budeme aplikovat na náš problém rekurentní neuronovou síť, která je vhodná pro práci s časovými řadami.

Základní neuronové sítě využívají pouze dopředný chod, tedy informace jsou transportovány sítí pouze jedním směrem ze vstupní vrstvy na výstupní vrstvu. Tento přístup je vhodný v případech, kdy jsou výstupní data nezávislá na předchozích vstupech. Avšak tento model neumožňuje uchovávat dlouhodobější informace z minulých vstupů.

Rekurentní neuronové sítě se odlišují schopností využívat předešlé výstupní hodnoty jako vstupní data při dalším běhu sítě. Tímto způsobem neuronové sítě uchovávají předešlé



Obrázek 2.7: Rekurentní neuronová síť

informace a jsou schopny rozeznávat dlouhodobější vzory v chování dat. Na obrázku 2.7 je ukázán příklad rekurentní neuronové sítě.
Dopředný chod neuronu rekurentní neuronové sítě můžeme popsat následujícím vztahem:

$$z_k^t = f\left(\sum_{i=1}^{n}(w_{ik}x_i^t) + \sum_{i=1}^{m}(v_{ik}z_i^{t-1}) + b_k\right),$$

kde m označuje velikost výstupního vektoru $\mathbf{z}^t = (z_1^t, \ldots, z_m^t)^\mathsf{T}$ a $V = (v_{ik})$ je matice vah vektoru $\mathbf{z}^t$.

Díky předávání předešlých informací při běhu sítě má tento model schopnost uchovávat kontext předchozích dat, avšak rozsah kontextu, který dokáže neuronová síť uchovat, je značně omezený. Vliv vstupů exponenciálně roste nebo klesá v závislosti na počtu cyklů, kterými projde neuronovou sítí. Tento problém nastává při metodě backpropagation. Následující odvození vychází z [10].

Budeme uvažovat model rekurentní neuronové sítě v odlišném tvaru, který je ekvivalentní s naším původním předpisem.

$$\mathbf{z}^t = W\boldsymbol{f}(\mathbf{x}^t) + V\boldsymbol{f}(\mathbf{z}^{t-1}) + \mathbf{b},$$

kde $\boldsymbol{f}(\mathbf{g})$ pro vektor $\mathbf{g}$ s délkou k chápeme jako $\boldsymbol{f}(\mathbf{g}) = (f(g_1), \ldots, f(g_k))$. Opět se snažíme obecně minimalizovat chybovou funkci $E(\mathbf{z}^t)$ vzhledem k neznámým parametrům, které budeme značit $\boldsymbol{\theta} = (W, V, \mathbf{b})$. Když daný model rozložíme do tvaru tradiční neuronové sítě, můžeme chybovou funkci zapsat jako součet chyb v jednotlivých cyklech, kde T je celkový počet cyklů.

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \sum_{t=1}^{T}\frac{\partial E^t}{\partial \boldsymbol{\theta}}.$$

Zaměříme se na minimalizaci chyby v kroku t. Užitím řetězového pravidla dostáváme

$$\frac{\partial E^t}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{t}\left(\frac{\partial E^t}{\partial \mathbf{z}^t}\frac{\partial \mathbf{z}^t}{\partial \mathbf{z}^k}\frac{\partial \mathbf{z}^k}{\partial \boldsymbol{\theta}}\right).$$

Člen $\frac{\partial \mathbf{z}^t}{\partial \mathbf{z}^k}$ nám určuje, jak výstupy z předchozích cyklů ovlivňují chybu v cyklu t. Rozepsá-

ním tohoto členu získáme

$$\frac{\partial \mathbf{z}^t}{\partial \mathbf{z}^k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{z}^i}{\partial \mathbf{z}^{i-1}} = \prod_{i=k+1}^{t} V \frac{\partial f(\mathbf{z}^{i-1})}{\partial \mathbf{z}^{i-1}} = \prod_{i=k+1}^{t} V \begin{pmatrix} \frac{\partial f(\mathbf{z}_1^{i-1})}{\partial \mathbf{z}_1^{i-1}} & \cdots & \frac{\partial f(\mathbf{z}_1^{i-1})}{\partial \mathbf{z}_m^{i-1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{z}_m^{i-1})}{\partial \mathbf{z}_1^{i-1}} & \cdots & \frac{\partial f(\mathbf{z}_m^{i-1})}{\partial \mathbf{z}_m^{i-1}} \end{pmatrix} =$$

$$= \prod_{i=k+1}^{t} V \begin{pmatrix} f'(\mathbf{z}_1^{i-1}) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & f'(\mathbf{z}_m^{i-1}) \end{pmatrix} = \prod_{i=k+1}^{t} V \ \mathrm{diag}(f'(\mathbf{z}^{i-1})) \tag{5}$$

Nyní si ukážeme, jak nám vzdálenost $t-k$ ovlivňuje učení sítě a jaký vliv na to má matice vah V. Předpokládáme, že derivace aktivační funkce $f'(x)$ je omezená číslem $\gamma \in \mathbb{R}$. Když budeme uvažovat euklidovskou normu matice, dostáváme

$$\left\| \mathrm{diag}(f'(\mathbf{z}^{i-1})) \right\| \leq \gamma.$$

Budeme uvažovat $\lambda_1 < \frac{1}{\gamma}$ jako absolutní hodnotu největšího vlastního čísla matice V. Vezmeme-li libovolné $k < t$, parciální derivace $\frac{\partial \mathbf{z}^{k+1}}{\partial \mathbf{z}^k}$ vyjádříme jako

$$\frac{\partial \mathbf{z}^{k+1}}{\partial \mathbf{z}^k} = V \mathrm{diag}(f'(\mathbf{z}^k)).$$

Využijeme-li základní vlastnosti normy matice, můžeme tento člen odhadnout

$$\left\| \frac{\partial \mathbf{z}^{k+1}}{\partial \mathbf{z}^k} \right\| \leq \|V\| \left\| \mathrm{diag}(f'(\mathbf{z}^k)) \right\| < \frac{1}{\gamma}\gamma = 1.$$

Odtud je vidět, že existuje $\eta \in \mathbb{R}$ splňující $\frac{\partial \mathbf{z}^{k+1}}{\partial \mathbf{z}^k} \leq \eta < 1$. Dosazením do rovnice (5) dostaneme

$$\frac{\partial \mathbf{z}^t}{\partial \mathbf{z}^k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{z}^i}{\partial \mathbf{z}^{i-1}} \leq \eta^{t-k-1}. \tag{6}$$

Tedy při rostoucí vzdálenosti $t-k$ se tato hodnota exponenciálně blíží 0 a tedy s rostoucím t klesá množství informací, které síť uchovává o minulých hodnotách. Tento nedostatek se nazývá problém mizejících gradientů. V případě aktivační funkce sigmoidy je $\gamma = \frac{1}{4}$ a pro funkci tangens hyperbolický platí $\gamma = 1$. Tedy obě funkce splňují náš předpoklad omezenosti první derivace a hodnota gradientu tak závisí na matici V. Analogicky pokud uvažujeme $\lambda_1 > \frac{1}{\gamma}$, dostaneme problém explodujících gradientů, kdy při rostoucím $t-k$ nastává $\frac{\partial \mathbf{z}^t}{\partial \mathbf{z}^k} \to \infty$.
Z tohoto důvodu pro uchovávání dlouhodobějších závislostí je třeba využít pokročilejší model neuronové sítě.

2.2.1 LSTM neuronová síť

Long Short Term Memory (LSTM) síť je speciální druh rekurentní neuronové sítě. Je speciálně navržena, aby vyřešila problém s dlouhodobým uchováváním závislostí, který se vyskytuje u tradiční rekurentní sítě. Pro uchování více informací má tento model komplexnější strukturu. Klasické neurony jsou nahrazeny bloky, které obsahují více částí. Schéma LSTM bloku můžeme vidět na obrázku 2.11. Teorie popsaná v této kapitole vychází z [6], [11] a [17].
Každý blok obsahuje tři hlavní komponenty, takzvané brány (Gates):

- Forget Gate,
- Input Gate,
- Output Gate.

Další klíčová součást LSTM bloku je funkce zachovávající dlouhodobé závislosti nazývaná stav buňky (Cell State). V další části si detailněji popíšeme dopředný chod a strukturu LSTM sítě.

Cell State

Na začátku je třeba zmínit, jak funguje stav buňky. Tento vektor budeme značit C^t, kde t představuje index cyklu LSTM sítě. Představuje dlouhodobou paměť uchovávající spojitosti a závislosti v předešlých datech. V průběhu chodu sítě je tento stav upravován pomocí bran na základě nových dat. Jelikož není v průběhu chodu sítě upravován pomocí vah, nenastává v tomto případě problém s mizejícím/explodujícím gradientem během učení neuronové sítě.

Forget Gate

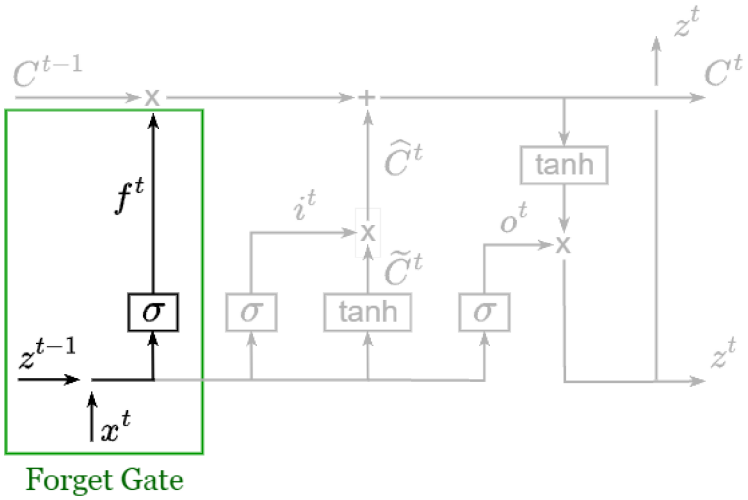
Prvním krokem v algoritmu LSTM sítě je rozhodnout, které informace z minulosti zachovat a naopak, které již nejsou důležité. Tento problém řeší Forget Gate. Aktivační funkce sigmoida, která nabývá hodnoty pouze v intervalu $(0, 1)$, slouží jako „filtrovací“ funkce. S jejím využitím dostaneme hodnotu $\boldsymbol{f}^t$ pomocí následujícího vztahu

$$\boldsymbol{f}^t = \sigma\big(W_f \mathbf{x}^t + V_f \mathbf{z}^{t-1} + \mathbf{b}_f\big),$$

kde funkcí $\sigma(\mathbf{r})$ rozumíme $\sigma(\mathbf{r}) = (\sigma(r_1), \ldots, \sigma(r_n))$. Pomocí této hodnoty upravíme stav buňky C^{t-1} z předchozího cyklu

$$\overline{C}^t = \boldsymbol{f}^t \odot C^{t-1},$$

kde $\odot$ chápeme jako násobení po složkách vektoru.



Obrázek 2.8: Forget Gate

Input Gate

V následujícím kroku vybereme, které vstupní informace uložíme do dlouhodobé paměti. Tento postup se skládá ze dvou částí. V první části se znovu pomocí funkce sigmoida určí hodnota i^t, která vyfiltruje důležité informace ze vstupních hodnot.

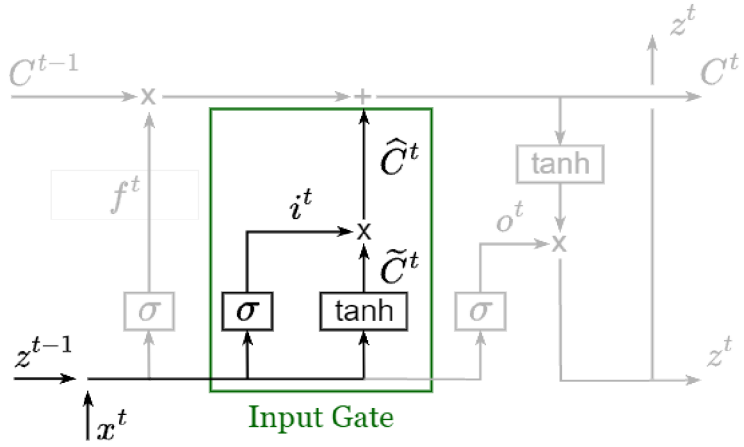
$$\boldsymbol{i}^t = \sigma\big(W_i \mathbf{x}^t + V_i \mathbf{z}^{t-1} + \mathbf{b}_i\big)$$

Druhá část převede pomocí aktivační funkce tangens hyperbolický vstupní data do hodnoty $\widetilde{C}^t$, která svým formátem odpovídá stavu buňky C^t.

$$\widetilde{C}^t = \tanh\big(W_C \mathbf{x}^t + V_C \mathbf{z}^{t-1} + \mathbf{b}_C\big)$$

Spojením těchto dvou částí dostaneme výslednou hodnotu $\widehat{C}^t$, která představuje nové informace, o které bude doplněn stav buňky.

$$\widehat{C}^t = \boldsymbol{i}^t \odot \widetilde{C}^t$$



Obrázek 2.9: Input Gate

Pomocí Forget Gate a Input Gate můžeme upravit stav buňky z předchozího cyklu a přidat k němu nové důležité informace, čímž vytvoříme nový stav buňky, pomocí kterého můžeme předpovídat nové výstupní hodnoty. Ve výsledku tedy dostáváme nový stav buňky C^t následujícím předpisem

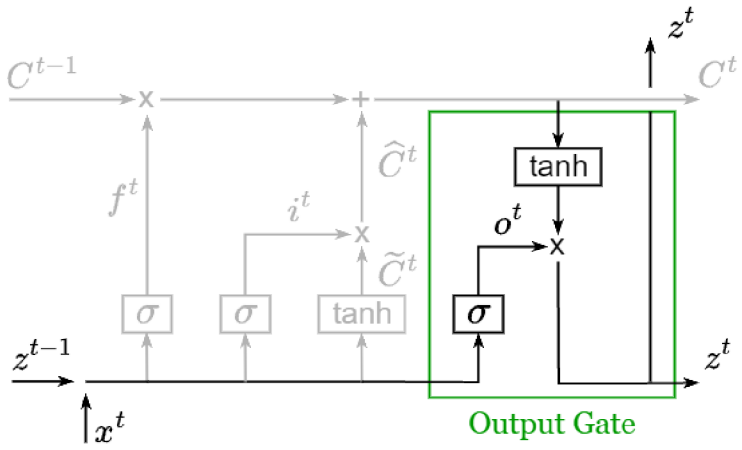
$$C^t = \overline{C}^t + \widehat{C}^t = \boldsymbol{f}^t \odot C^{t-1} + \boldsymbol{i}^t \odot \widetilde{C}^t.$$

Output Gate

Posledním krokem LSTM algoritmu je využití nového stavu buňky a vstupních hodnot pro vytvoření výstupu. Stejně jako v předchozích krocích se znovu použije funkce sigmoid pro filtraci vstupních hodnot. Tuto funkci budeme značit o^t. Poté už pouze použijeme tangens hyperbolický na stav buňky, abychom dostali výstup v intervalu $(-1, 1)$, a obě hodnoty spolu vynásobíme

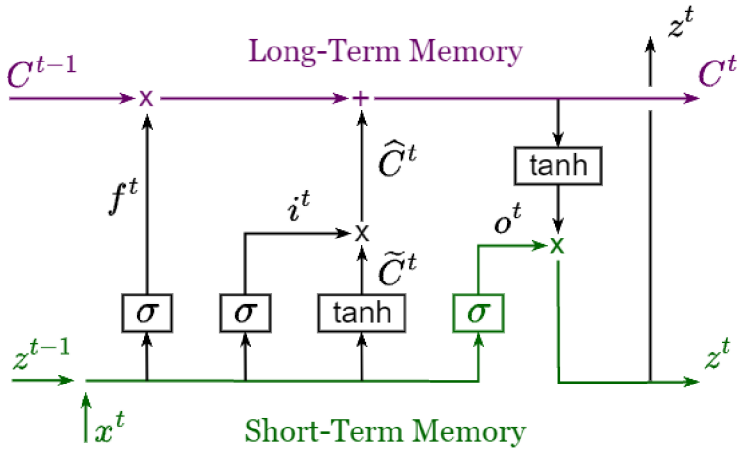
$$\boldsymbol{o}^t = \sigma\big(W_o \mathbf{x}^t + V_o \mathbf{z}^{t-1} + \mathbf{b}_o\big),$$

$$\mathbf{z}^t = \tanh(C^t) \odot \boldsymbol{o}^t.$$



Obrázek 2.10: Output Gate

Na Obrázku 2.2.1 je graficky znázorněna celá buňka LSTM sítě. Navíc je zde znázorněna část buňky, která zachovává dlouhodobé závislosti (Long-Term Memory) a krátkodobé závislosti (Short-Term Memory). Můžeme vidět, že krátkodobé souvislosti jsou uchovány ve výstupu z předchozího cyklu sítě $\mathbf{z}^{t-1}$ a dlouhodobé závislosti uchovává stav buňky C^t.



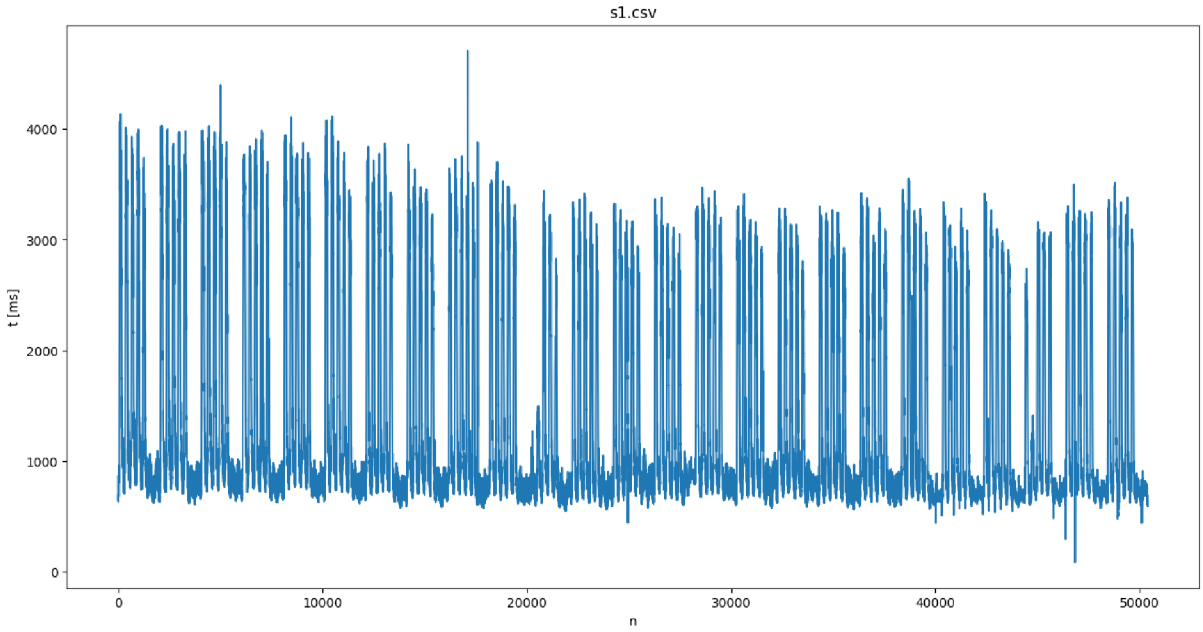
Obrázek 2.11: Long Short Term Memory síť

3 Model neuronové sítě

V této sekci se zaměříme na samotný model neuronové sítě. Naším cílem bude nalézt co nejobecnější model neuronové sítě, který bude predikovat budoucí hodnoty reálných dat, jejichž charakter bude různorodý. Pro tvorbu modelu použijeme programovací jazyk Python, který vyhovuje našemu účelu díky jednoduché syntaxi, ale hlavně díky rozsahu knihoven, které jsou uzpůsobené pro tvorbu neuronových sítí a také pro zpracování dat. Primárně budeme využívat knihovnu TensorFlow pro stavbu modelu neuronové sítě a knihovnu Scikit-learn pro přípravu dat.

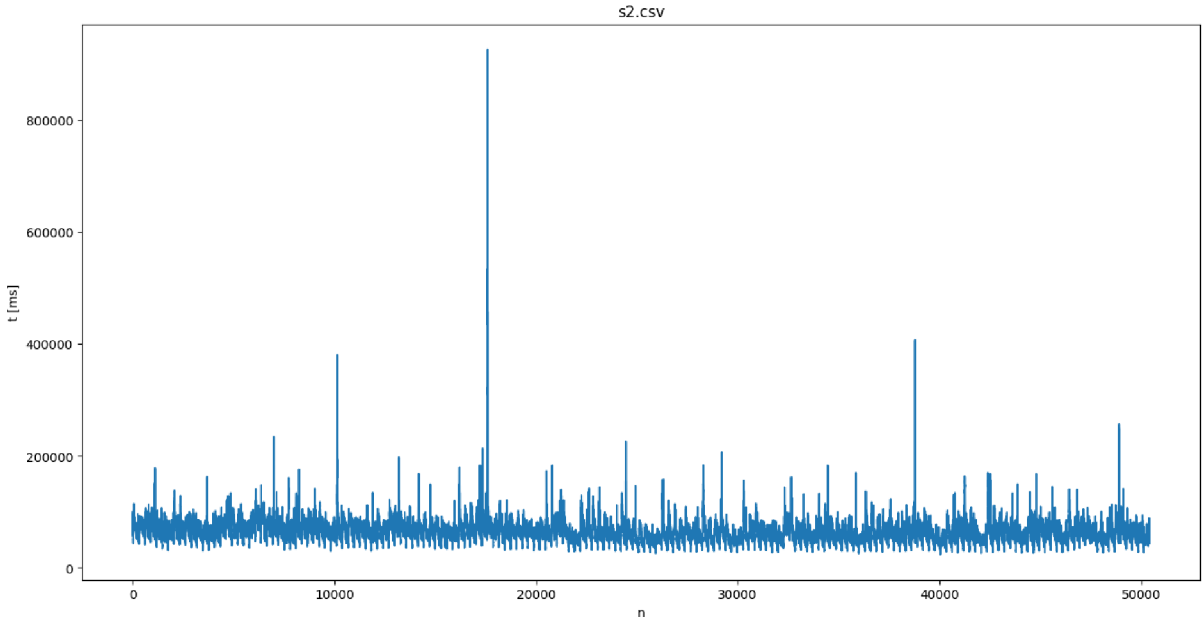
3.1 Popis dat

Náš model budeme aplikovat na reálná data. K dispozici máme dva rozdílné datasety. Jedná se o časovou odezvu serverů síťového provozu. Data jsou ve formátu .csv, obsahují informaci o čase měření ve formátu timestamp a časovou délku odezvy serveru v milisekundách. Každý dataset obsahuje hodnoty měření za půl roku, kdy měření se opakuje jednou za 5 minut. Budeme pracovat s daty od 26. dubna 2021 2:00 do 18. října 2021 1:55 o celkové délce 50400 hodnot. Data můžeme reprezentovat jako časové řady a pro zjednodušení nahradíme formát času indexem určujícím pořadí hodnot. Na obrázcích 3.1 a 3.2 můžete vidět použité datasety.



Obrázek 3.1: Dataset s1.csv

Naším cílem je na základě našich dat využít neuronovou síť k predikci následující hodnoty. Řešení rozdělíme na dva typy problémů. První případ bude popisovat situaci, kdy v reálném provozu dostaneme jednu hodnotu za 5 minut a na základě dosavadních hodnot budeme predikovat následující hodnotu. V našem případě není nutné predikovat více hodnot do budoucnosti, jelikož na validaci jedné hodnoty čekáme 5 minut a můžeme novou hodnotu použít pro predikci další hodnoty, čímž máme přesnější vstup, než kdybychom predikovali následující 2 hodnoty a čekali na validaci 10 minut. Hlavní myšlenkou je použití natrénované LSTM neuronové sítě, která je schopná s přijatelnou přesností predikovat následující hodnotu a v momentě změření reálné hodnoty použijeme predikci



Obrázek 3.2: Dataset s2.csv

pro porovnání. Pokud se budou tyto hodnoty odlišovat o předem stanovenou hodnotu, prohlásíme reálnou hodnotu za anomálii.

Druhý problém bude zahrnovat zpětnou analýzu celé časové řady a označení možných anomálií na základě stejného principu jako v předchozím případě. Budeme se primárně soustředit na detekci anomálií na celém datasetu, tedy zpětnou analýzu poskytnutých reálných dat. Řešení této úlohy budeme moci aplikovat později s menšími úpravami i na první problém v reálném provozu.

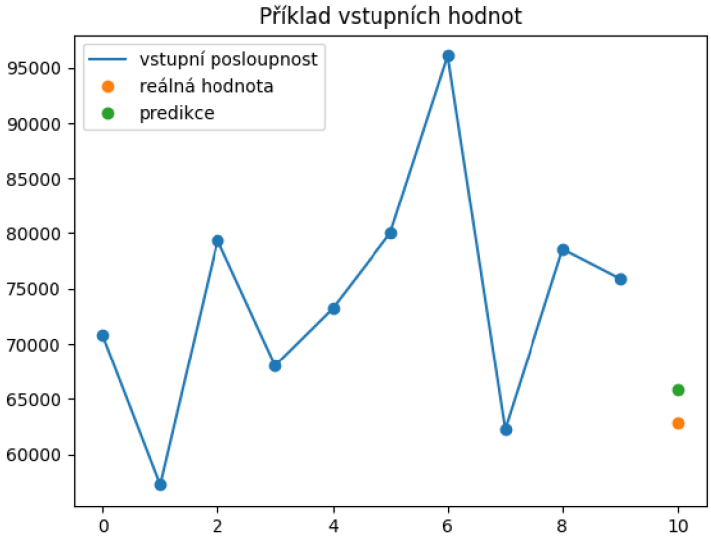
3.2 Preprocessing

V první řadě budeme věnovat pozornost zpracování časových řad a jejich formátování pro vstup do neuronové sítě. Základní model LSTM neuronové sítě pracuje s aktivačními funkcemi sigmoida a tangens hyperbolický, jejichž obor hodnot je $(0, 1)$ a $(-1, 1)$. Funkce sigmoida má funkci filtru a přímo netransformuje vstupní data, budeme se tedy hlavně soustředit na funkci tangens hyperbolický. Z průběhu této funkce zobrazené na obrázku 2 vidíme, že funkční hodnoty vzdálené od nuly se zobrazují na hodnoty blízké jedničce a pokud bychom jako vstupy využili naše původní data, všechny hodnoty by se zobrazily na skoro stejné hodnoty blízké jedničce a neuronová síť by neměla šanci zachytit charakter dat, protože daná data by byla reprezentovaná téměř konstantní řadou. Z tohoto důvodu bude potřeba data normalizovat. Jelikož naše data nabývají pouze kladných hodnot, transformujeme naše data do intervalu $\langle 0, 1\rangle$ následujícím způsobem

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}, \qquad \forall x \in X.$$

Tímto zajistíme správnou funkčnost neuronové sítě. Proces učení bude probíhat následujícím způsobem. Časová řada se před vstupem do neuronové sítě normalizuje. Z normalizovaných vstupů síť predikuje normalizovanou hodnotu, která se následně transformuje do původního měřítka. Jelikož výpočet chyby probíhá uvnitř neuronové sítě, musí být normalizovaná i validační data. Z tohoto důvodu budou hodnoty chybové funkce velmi malé, jelikož se počítají z normalizovaných dat.

Při učení neuronové sítě se využívají dva hlavní postupy, trénování s učitelem a bez učitele. Trénování bez učitele znamená, že neuronová síť nemá data označená a musí v nich sama hledat skryté vzory nebo struktury. My zvolíme přístup učení neuronové sítě s učitelem, tedy neuronové síti připravíme ke každé predikci validační reálnou hodnotu, pro ověření přesnosti predikce. Pro učení neuronové sítě budeme potřebovat matici vstupních hodnot a vektor reálných validačních hodnot. Velikost matice vstupních hodnot je určená velikostí vstupu neuronové sítě, což znamená, kolik předchozích hodnot bude využito pro predikci následující hodnoty. Vstup ve formátu časové řady vždy posuneme o jednu hodnotu dopředu při predikci následující hodnoty. Každá vstupní řada bude představovat jeden řádek ve vstupní matici X a k ní bude náležet jedna následující reálná hodnota ve validačním vektoru $\hat{\mathbf{z}}$. Příklad vstupních hodnot můžete vidět na obrázku 3.3.



Obrázek 3.3: Příklad vstupních hodnot pro predikci

Obecně uvažujme časovou řadu s celkovým počtem N hodnot. Uvažujme délku vstupní posloupnosti k, vstupní matice X bude mít rozměry $N-k \times k$ a vektor reálných hodnot $\hat{\mathbf{z}}$ bude mít velikost $N-k$. Tímto způsobem si připravíme vstupní posloupnosti pro celou řadu, kdy neuronová síť provede predikce pro každý řádek matice X, výsledkem bude vektor predikcí $\mathbf{z}$ o velikosti $N-k$.

Pro přípravu dat slouží metoda `create_dataset()`. Vstupem do této metody je časová řada k predikci a velikost vstupního vektoru $\mathbf{x}_t$, tedy kolik hodnot z minulosti se použije k predikci. Metoda postupně prochází celý dataset a přidává řádky do matice X a vektoru $\mathbf{z}$. Nakonec se výsledné hodnoty převedou do typu pole pro pozdější využití funkcí knihovny Pandas.

Metoda 1 Vytvoření vstupů neuronové sítě

```
def create_dataset(dataset, input_size):
        dataX, dataZ = [], []
        for i in range(len(dataset)-input_size):
                x_t = dataset[i:(i+input_size), 0]
                dataX.append(x_t)
                dataZ.append(dataset[i + input_size, 0])
        return np.array(dataX), np.array(dataZ)
```

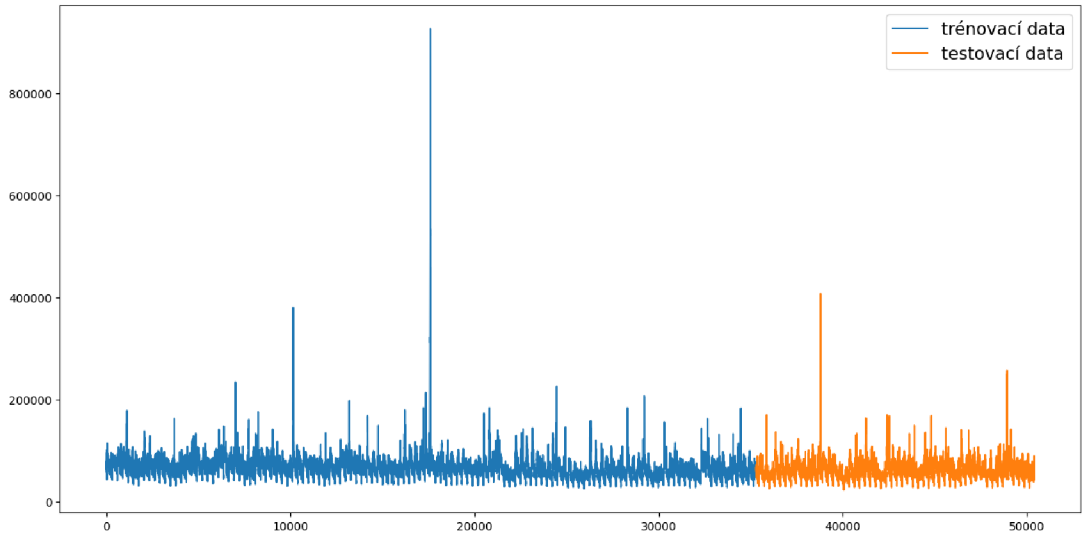
Na obrázku 3.4 je zobrazen příklad vstupních hodnot neuronové sítě pro prvních 10 hodnot souboru s1.csv.

	x_1	x_2	x_3	x_4	x_5	z
0	70778.687500	57231.960938	79374.859375	68029.671875	73192.382812	80004.234375
1	57231.960938	79374.859375	68029.671875	73192.382812	80004.234375	96065.312500
2	79374.859375	68029.671875	73192.382812	80004.234375	96065.312500	62237.812500
3	68029.671875	73192.382812	80004.234375	96065.312500	62237.812500	78576.984375
4	73192.382812	80004.234375	96065.312500	62237.812500	78576.984375	75884.906250

Matice X Vektor $\mathbf{z}$

Obrázek 3.4: Vstupní hodnoty neuronové sítě

Pro trénování neuronové sítě se vstupní data dělí na dva datasety, trénovací a testovací data. Trénovací data se používají při trénování sítě, pomocí kterých se minimalizuje chybová funkce. Testovací data následně slouží k ověření natrénovaných parametrů neuronové sítě, jelikož tato data neměla síť při trénování k dispozici. Bývá zvykem použít 70-80 % celkových dat pro trénování a zbylá data využít při testování sítě. V našem případě využijeme 70 % dat pro trénování a 30 % dat pro testování modelu. Rozdělení dat u datasetu s2 můžete vidět na obrázku 3.5.



Obrázek 3.5: Rozdělení souboru s2

3.3 Tvorba modelu neuronové sítě

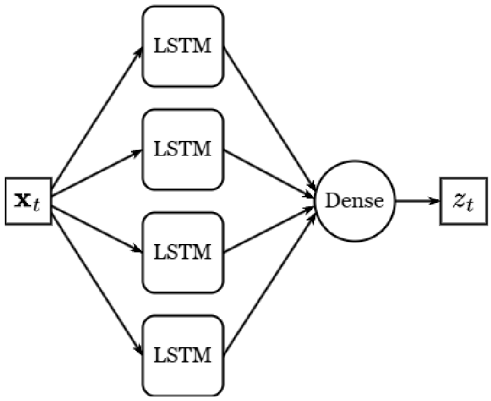
V této sekci se budeme věnovat postupnému sestavení modelu rekurentní neuronové sítě. Základní model bude vycházet z [12] a [2] a postupně budeme model rozšiřovat na základě získaných výsledků a teorie popsané v předchozí části práce.

V dnešní době je stále častější při nasazení umělé inteligence použít již předtrénované modely, které slouží jako forma pro co nejobecnější využití. Tímto si můžeme usnadnit práci při navrhování a trénování sítě, protože se nemusíme zabývat vhodnou architekturou modelu, ale na druhou stranu využíváme velkou síť s vyšší výpočetní a časovou náročností, než by bylo potřeba u specializované neuronové sítě. Pokusíme se vytvořit menší model specializovaný na náš problém a vyhneme se použití velkého předtrénovaného modelu. Náš

primární cíl bude vytvořit co nejjednodušší model, který bude co nejpřesněji predikovat budoucí hodnoty našich datasetů a tím minimalizovat celkovou náročnost neuronové sítě.

3.3.1 Struktura modelu

Nyní můžeme začít konstruovat samotnou strukturu neuronové sítě. Máme připravená vstupní data i validační hodnoty a můžeme se zabývat jednotlivými parametry a komponenty, ze kterých se skládá samotná neuronová síť. Mimo vstupů máme jasný i výstup, pouze jedinou výstupní hodnotu reprezentující predikci. Ze začátku se omezíme na velice jednoduchou strukturu neuronové sítě, na které otestujeme, jaké parametry budou pro náš účel nejvhodnější. Základem naší neuronové sítě budou LSTM bloky. Budeme uvažovat pouze model s jedinou skrytou LSTM vrstvou, která bude obsahovat pro začátek menší počet bloků. Pro nižší časovou náročnost při testování zvolíme pouze 4 LSTM bloky. Počet neuronů (bloků v případě LSTM) v jedné skryté vrstvě budeme označovat jako šířku vrstvy a jak už bylo zmíněno dříve, hloubkou modelu budeme označovat počet skrytých vrstev. Poslední skrytou vrstvou bude vrstva Dense, která označuje obyčejnou vrstvu obsahující základní neuron popsaný v sekci 2.1, který hodnoty z předchozí LSTM vrstvy transformuje do jedné výstupní hodnoty. Graf základního modelu je zobrazen na obrázku 3.6.



Obrázek 3.6: Graf modelu neuronové sítě

Než přejdeme k samotnému testování parametrů, musíme rozhodnout, jaké základní komponenty bude síť obsahovat a jaké principy bude využívat při trénování. Jelikož naše data jsou ve tvaru časové řady, a cílem je co nejpřesnější predikce, budeme chybu predikce posuzovat na základě rozdílu vzdálenosti přesné a predikované hodnoty. Z tohoto důvodu využijeme jako chybovou funkci (loss function) střední kvadratickou chybu E popsanou rovnicí (4). Jelikož hodnota střední kvadratické chyby se při trénování sítě nehodí pro představu, jaké přesnosti model dosahuje, knihovna Tensorflow nabízí navíc možnost si u modelu zvolit metriku, pomocí které se spočítá chyba predikce, ale tato funkce není využita při trénování modelu, slouží pouze jako informativní hodnota. Z tohoto důvodu při trénování sítě zvolíme jako metriku průměrnou absolutní odchylku.

Dalším důležitým aspektem modelu neuronové sítě je takzvané přetrénování. K tomuto jevu dochází, když se váhy a posuny neuronové sítě nastaví tak, že dochází k přesné aproximaci testovacích dat. Tím neuronová síť přichází o obecnost predikovat odlišná data. Obvykle se tomuto jevu snažíme zabránit. Přetrénování sítě se dá zmírnit zakomponováním Dropout vrstvy do modelu. Jak už její název napovídá, tato vrstva náhodně vynechává při trénování určité procento vstupů, čímž zabraňuje modelu se přesně přizpůsobit trénovacím datům a zachová si větší obecnost.

V některých případech může být přetrénování výhodné pro daný problém. Může pomoci neuronovým sítím zachytit velmi jemné a komplexní vzory v časových řadách, které by bez přetrénování nebyly zachyceny. Také se využívá při malém vzorku dat pro získání největšího množství informací. Možné benefity přetrénování jsou zmíněny v [19]. V našem případě, kdy nám při predikci jde o co nejpřesnější zachování charakteru dat, může být přeučení neuronové sítě cesta k přesnější predikci. V kapitole 3.5 otestujeme, zda přetrénování modelu bude mít pozitivní vliv na naše výsledky.

3.3.2 Optimizer

Jedním ze stěžejních faktorů neuronové sítě je optimizer. Úzce souvisí s učením neuronové sítě, které jsme popsali v teoretické části práce. Optimizer je metoda využívaná pro minimalizaci chybové funkce E. Metoda popsaná v teoretické části je jednou z nejjednodušších optimizerů a také velice vhodnou pro pochopení. Avšak daná metoda není nejrychlejší a je výpočetně poměrně náročná. Z toho důvodů vzniklo více verzí optimizerů, z nichž většina vychází z gradientní metody a přidává další parametry nebo nastavuje adaptivně míru učení pro urychlení algoritmu. Jedním z nejefektivnějších algoritmů je optimizer Adam. Skládá se ze dvou gradientních metod. Následující informace vychází z [13].

První metoda využívá momenty pro urychlení metody gradientů (SGD), která využívá klouzavých průměrů gradientů. Algoritmus určující váhu w_t v čase $t+1$ funguje následovně.

$$m_t = \beta m_{t-1} + (1-\beta)\frac{\partial E}{\partial w_t},$$

$$w_{t+1} = w_t - \alpha m_t,$$

kde m_t označuje moment v čase t a $\beta \in \langle 0, 1)$, popisuje parametr klouzavého průměru, který určuje, jakou váhu mají předchozí gradienty.

Druhá metoda se nazývá Root Mean Square Propagation (RMSP). Jedná se o adaptivní metodu, která v každém kroku upravuje míru učení α. Tento algoritmus je podobný předchozímu, akorát upřednostňuje více gradient v daném kroku.

$$v_t = \beta v_{t-1} + (1-\beta)\left(\frac{\partial E}{\partial w_t}\right)^2,$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \varepsilon}}\frac{\partial E}{\partial w_t},$$

kde v_t označuje druhý moment v čase t a ε je konstanta blízká nule zajišťující numerickou stabilitu v prvním kroku, kdy $v_0 = 0$.
Spojením předchozích algoritmů a označením $\frac{\partial E}{\partial w_t} = g(t)$ dostáváme první a druhý moment

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g(t),$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g(t)^2.$$

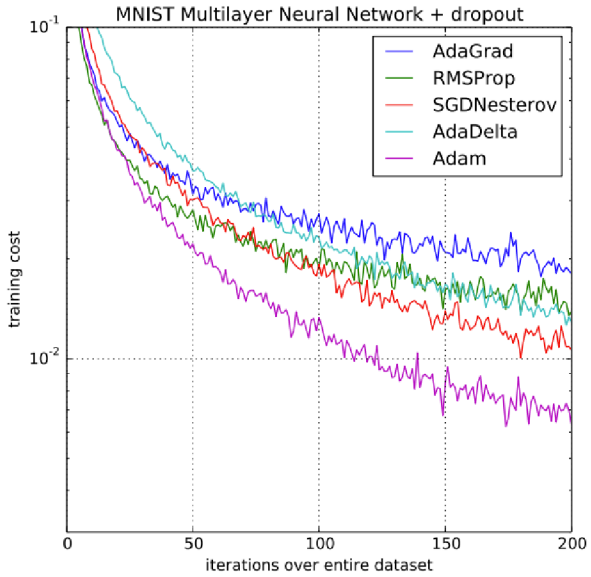
Počáteční hodnoty volíme $m_0 = 0$, $v_0 = 0$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ a $\varepsilon = 10^{-8}$. V kroku t spočítáme momenty m_t a v_t následně provedeme korekci biasu, což nám dává lepší odhad pro klouzavé průměry

$$\widehat{m}_t = \frac{m_t}{(1-\beta_1^t)}, \qquad \widehat{v}_t = \frac{v_t}{(1-\beta_2^t)}.$$

V posledním kroku spočítáme samotné váhy w_t

$$w_t = w_{t-1} - \alpha \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}.$$

Algoritmus opakujeme, dokud není dosaženo požadovaného počtu iterací, které budeme v rámci neuronových sítí nazývat epochy. Stejným algoritmem budeme počítat také posuny **b** neuronové sítě. Na obrázku 3.7, který je převzatý z [13] je porovnání minimalizace chyby s ostatními optimizery.



Obrázek 3.7: Porovnání efektivity minimalizace chyby neuronové sítě

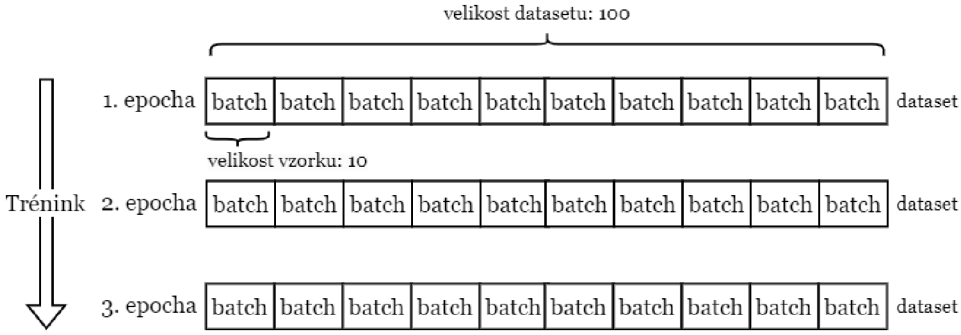
3.3.3 Epocha a velikost vzorku

Abychom měli model neuronové sítě kompletní, je potřeba zmínit význam epochy a velikosti vzorku (batch size). Pro názornost si tyto pojmy vysvětlíme na vstupní matici X, která bude obsahovat 100 řádků. Každý řádek obsahuje jednu vstupní posloupnost hodnot o délce k.

Velikost vzorku udává, kolik vstupních hodnot vstoupí při trénování do neuronové sítě, než dojde k úpravě neznámých parametrů vah a posunů. Tedy pokud zvolíme velikost vzorku 10, model predikuje 10 hodnot a poté dojde ke srovnání s přesnými hodnotami a úpravě vah a posunů. Pokud velikost vzorku nedělí celkovou velikost datasetu, poslední velikost vzorku je menší a obsahuje zbylé hodnoty.

Počet epoch určuje, kolikrát v rámci učení neuronová síť projde celý trénovací dataset. V rámci jedné epochy má každá hodnota trénovacího datasetu možnost upravit model. Pokud nastavíme 10 epoch, velikost vzorku 10 a máme celkem 100 vstupních hodnot, v každé epoše dojde k 10 úpravám parametrů, tedy parametry modelu budou přeučeny stokrát.

Počet epoch představuje počet iterací, kolikrát v rámci učení síť použila celý dataset a velikost vzorku snižuje náročnost učení neuronové sítě, ale značně snižuje rychlost konvergence k nejmenší hodnotě chyby. Každá epocha obsahuje minimálně jeden vzorek (v případě, kdy k přeučení sítě dojde po predikci celého datasetu). Počet epoch ovlivňuje



Obrázek 3.8: Epocha vs batch size

velikost chyby neuronové sítě. S rostoucím počtem epoch chybová funkce neuronové sítě konverguje ke svému minimu.

Větší hodnoty velikosti vzorku se využívají u velkých trénovacích datasetů a neuronových sítích s velkou hloubkou nebo šířkou. Epochy i velikost vzorku významně ovlivňují jak časovou, tak i výpočetní náročnost modelu. Půjde o další důležité parametry, které je třeba optimalizovat pro efektivitu trénování neuronové sítě.

3.4 Implementace modelu neuronové sítě

Nyní, když známe strukturu a komponenty našeho modelu, můžeme začít s implementací modelu LSTM sítě. Jak už bylo zmíněno na začátku kapitoly, program budeme implementovat v programovacím jazyce Python. Model neuronové sítě budeme tvořit pomocí knihovny TensorFlow, k práci s daty a preprocessingu budeme využívat knihovny scikit-learn, Pandas a NumPy. Vizualizaci výsledků zajišťuje knihovna Matplotlib.

Jako první vytvoříme metodu pro tvorbu modelu neuronové sítě. Tato metoda má název `set_model()` a používá vstupní parametr `input_length`, který označuje délku k vstupního vektoru $\mathbf{x}$ a parametr `neurons`, který určuje šířku modelu, tedy počet neuronů v jedné skryté vrstvě. Uvnitř metody se inicializuje model neuronové sítě, následně se do modelu přidá skrytá LSTM vrstva obsahující zadaný počet neuronů a nastaví se velikost vstupního vektoru. Následně se přidá vrstva Dense, která zkombinuje výstupy z LSTM vrstvy do jediné požadované hodnoty. Nastavíme chybovou funkci, v našem případě použijeme střední kvadratickou chybu, jako optimizer zvolíme metodu Adam, a nakonec nastavíme metriku jako průměrnou absolutní odchylku.

Metoda 2 Vytvoření modelu neuronové sítě

```
def set_model(input_length,neurons):
    model = Sequential()
    model.add(LSTM(neurons, input_shape=(1, input_length)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error',optimizer='adam',
                  metrics="mean_absolute_error")
    return model
```

S vytvořeným modelem můžeme přejít k trénování neuronové sítě. Trénink parametrů nám obstará metoda `NNTrain()`. Vstupními parametry této metody jsou parametry `trainX` a `trainZ`, které reprezentují matici vstupních hodnot X a validační vektor $\hat{\mathbf{z}}$. Následují parametry určující počet neuronů, délku vstupních posloupností, počet epoch a velikost vzorku. Uvnitř metody se nejdříve inicializuje model neuronové sítě pomocí metody

`set_model()` a poté se nastaví takzvaný `ModelCheckpoint()`, který slouží k průběžnému ukládání modelu.

Díky průběžnému ukládání nemusíme model při každém spuštění programu znovu trénovat. Navíc, pokud dojde k chybě nebo model nedosahuje požadované přesnosti, můžeme v trénování pokračovat v místě, kde bylo přerušeno. V rámci checkpointu je třeba nastavit umístění v paměti pro ukládání. Budeme ukládat pouze váhy a posuny modelu místo modelu celého pro ušetření místa v paměti, navíc jsme kdykoliv schopni snadno znovu vytvořit model díky metodě `set_model()`. Následně se určí chybová funkce na základě, které se budou ukládat váhy. Váhy budeme ukládat na základě nejmenší hodnoty chybové funkce, navíc uložíme pouze váhy, které budou vykazovat menší chybu než v předchozích epochách. Samotné trénování se spouští pomocí `model.fit()`, který vyžaduje vstupní a validační hodnoty a nastaví počet epoch, velikost vzorku a průběžné ukládání. Nakonec uložíme získané hodnoty chybové funkce, které budeme potřebovat pro pozdější porovnání jednotlivých modelů a uložíme je ve formátu csv.

Metoda 3 Trénink neuronové sítě

```
def NNTrain(trainX, trainZ, neurons, input_length, epochs, batch_size):
    model = set_model(input_length,neurons)
    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
        filepath='.\weights\weight_{}_ep_{:03d}_neur_{}_lb_{}'
                  .format(filename,epochs,neurons,input_length),
        save_weights_only=True,
        monitor='loss',
        mode='min',
        save_best_only=True)
    history=model.fit(trainX, trainY, epochs=epochs,batch_size=batch_size,
            verbose="auto",callbacks=[model_checkpoint])
    loss=history.history['loss']
    data=pd.DataFrame({"epochs":np.arange(1,epochs+1),"loss":loss})
    data.to_csv('.\loss\loss_{}_ep_{:03d}_neur_{}_lb_{}.csv'
                 .format(filename,epoch,neurons,look_back),index=False)
    return model
```

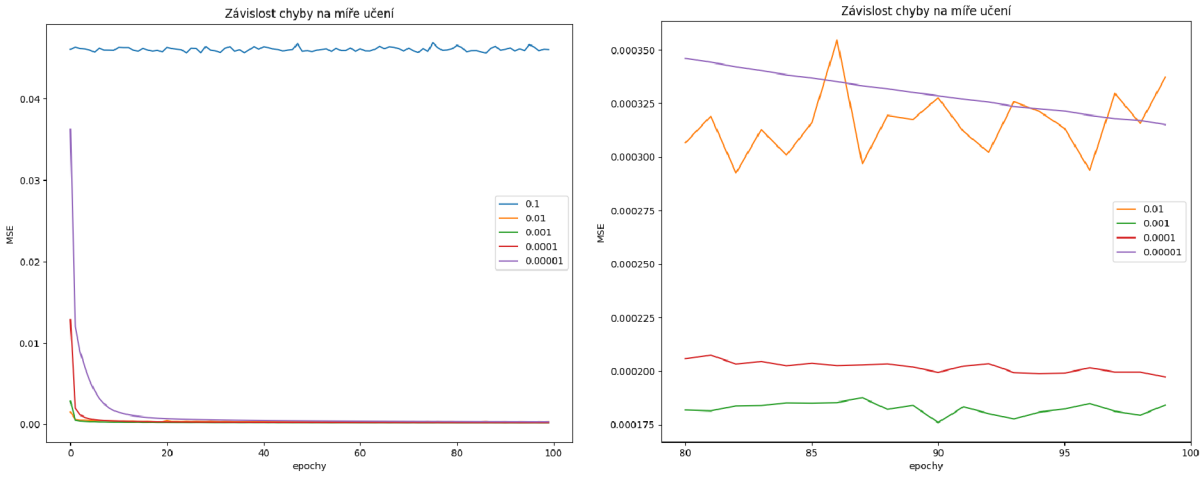
3.5 Trénování modelu

Popsali jsme ty nejdůležitější metody, které budeme využívat při tvorbě a trénování neuronových sítí a nyní se v této kapitole budeme věnovat časově nejnáročnější části, trénování modelů neuronových sítí, jejich porovnání a vylepšení na základě získaných výsledků. V této sekci nebudeme uvádět přesné hodnoty časové náročnosti algoritmů, jelikož se jedná pouze o prototypy a jazyk Python není tak striktní při běhu kódu jako například C++.

3.5.1 Míra učení

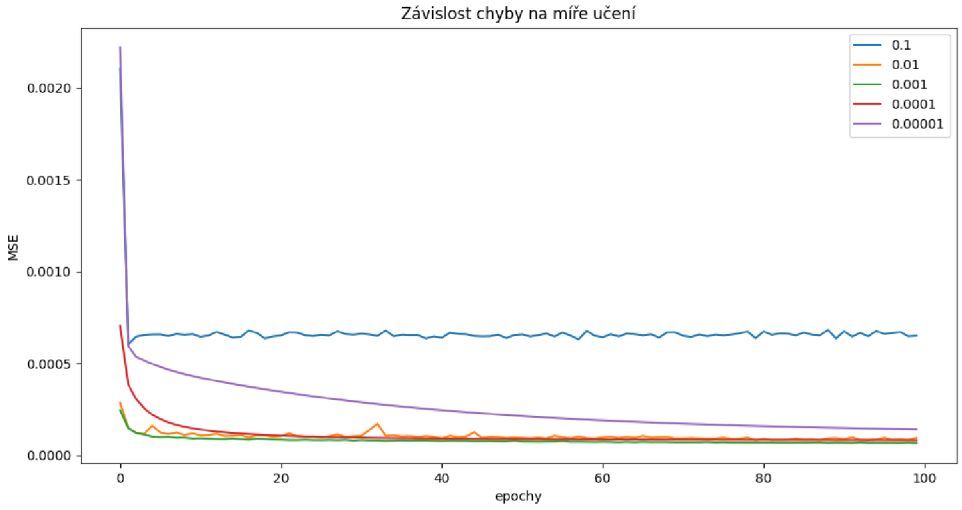
V první řadě se podíváme, jakou nejvhodnější hodnotu zvolíme jako míru učení. Otestujeme hodnoty α v rozmezí 0.1 až po 0.00001 a hodnoty budeme postupně zmenšovat o jeden řád. Používáme sice optimizer Adam, který využívá adaptivní míru učení, což je popsáno v sekci 3.3.2, ale její počáteční hodnota výrazně ovlivňuje průběh učení. Jednotlivé hodnoty budeme porovnávat na základě chybové funkce E. Jako vstupní hodnoty zvolíme vektor o délce 288 hodnot, což v pětiminutových intervalech představuje jeden den. Pro trénink bylo zvoleno 100 epoch. Na obrázcích 3.9 a 3.10 můžeme vidět vývoj

chyby u datasetů s1 a s2 s rostoucím počtem epoch. Mějme na paměti, že pracujeme s normalizovanými daty, proto nabývá chybová funkce takto nízkých hodnot.



Celý dataset Detail posledních 20 epoch

Obrázek 3.9: Závislost chyby na míře učení souboru s1



Obrázek 3.10: Závislost chyby na míře učení souboru s2

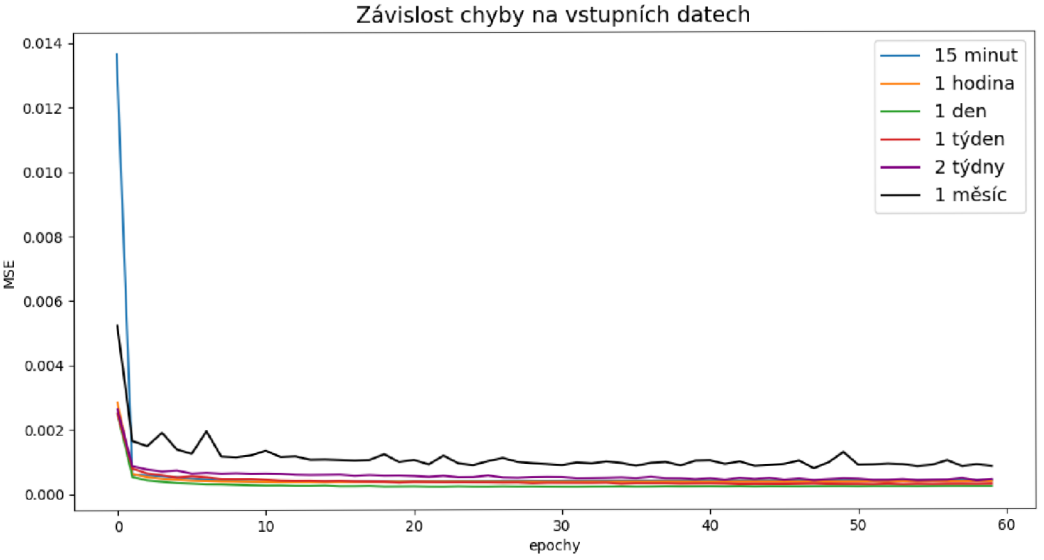
Z obrázků je vidět, že u obou datasetů hodnota $\alpha = 0.1$ byla zvolena příliš velká a chybová funkce nekonverguje ke globálnímu minimu. Menší hodnoty již konvergují všechny, ale liší se rychlostí konvergence. U hodnoty $\alpha = 0.01$ chyba konverguje, ale na průběhu funkce u obou datasetů je vidět, že vzhledem k normalizovaným datům je tato hodnota stále relativně velká, což se projevuje výkyvy v chybové funkci.

Nejlépe se chová chybová funkce při volbě $\alpha = 0.001$, kdy funkce konverguje ve stejném čase nejrychleji k nejnižší hodnotě v obou případech. U menších α chybová funkce začíná na velkých hodnotách a rychlost konvergence je menší než u 0.001. Tedy u těchto hodnot by byl pro stejnou přesnost nutný větší počet epoch. Z obou grafů je vidět, že zvolená hodnota epoch byla pro $\alpha = 0.001$ zbytečně velká. V posledních 40-50 epochách docházelo k minimálnímu poklesu chybové funkce.

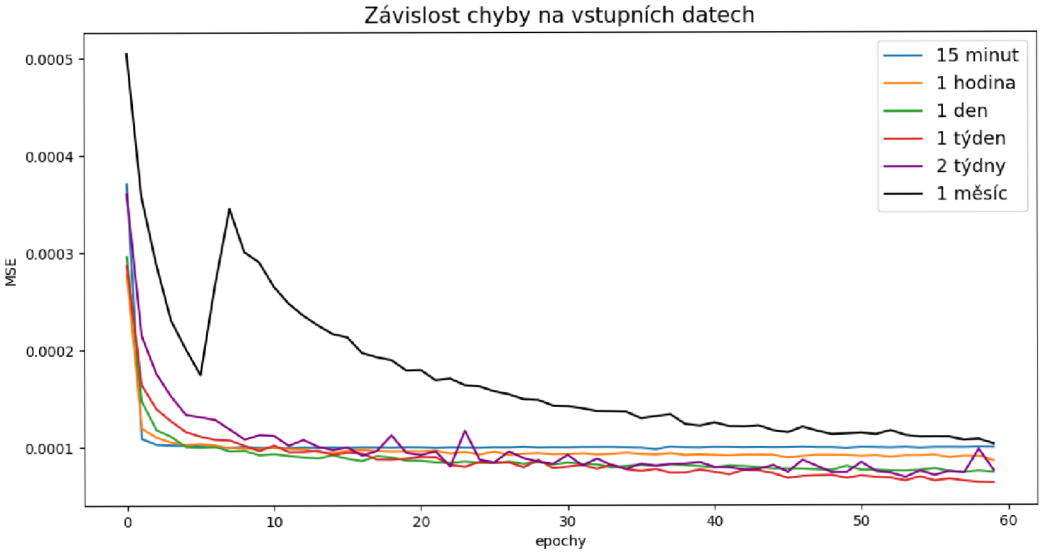
3.5.2 Délka vstupní posloupnosti

Dalším krokem v tvorbě modelu bude zvolení ideální délky vstupních posloupností. Tato hodnota nám určuje, z kolika předešlých hodnot se bude predikovat následující hodnota. LSTM síť sice obsahuje i určité množství informací o předchozích vstupech, ale značný podíl informací neuronová síť získává z nedávných hodnot. Pravděpodobně bude mít vliv na výsledky mimo velikosti vstupu i šířka modelu, aby mohla síť využít více informací z větších vstupů. Zatím se omezíme pouze na síť se čtyřmi bloky pro získání prvotních informací, poté se budeme zabývat i vlivem šířky modelu.

Budeme testovat vstupní posloupnosti o délce 15 minut, 1 hodina, 1 den, 1 týden, 2 týdny a 1 měsíc. Z výsledků v předešlé kapitole snížíme počet epoch na 60 pro zmenšení časové i výpočetní náročnosti. Výsledné hodnoty chybové funkce v průběhu učení jsou zobrazeny na obrázku 3.11 a 3.12.



Obrázek 3.11: Závislost chyby na šířce neuronové sítě datasetu s1

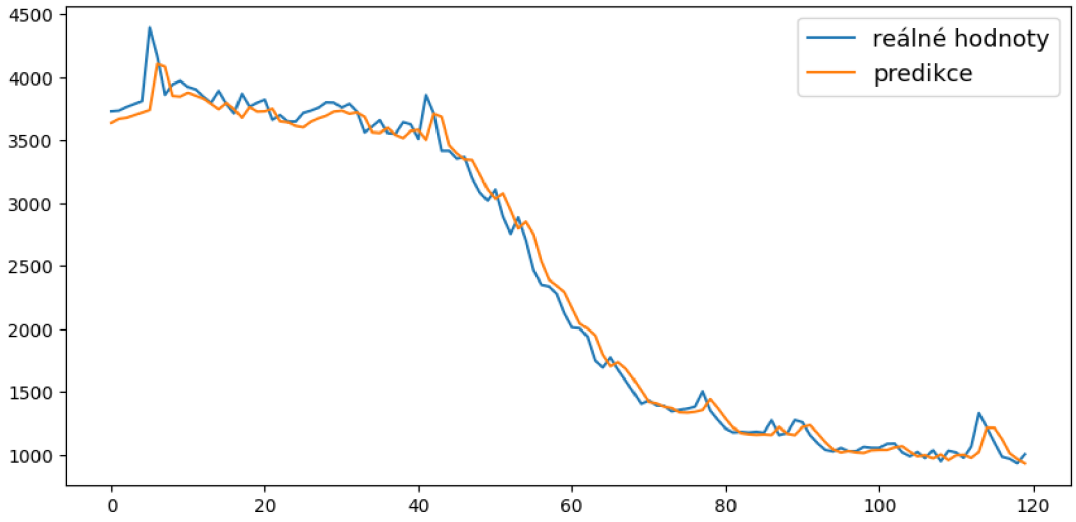


Obrázek 3.12: Závislost chyby na šířce neuronové sítě datasetu s2

Z výsledných grafů se v případě datasetu s1 ukazuje jako nejlepší výsledná posloupnost jeden den, v případě datasetu s2 nejlépe dopadl jeden týden. V obou případech se ukázal jeden měsíc jako zbytečně velká informace. Navíc se v případě datasetu s2 chybová funkce rapidně zvýšila v prvotní fázi trénování. Těžko se odhaduje, co způsobilo tento neočekávaný výkyv, jestli je to způsobeno vlivem optimizeru nebo tento jev vznikl díky výskytu velkých hodnot v datasetu.

V dalších modelech budeme uvažovat jako výchozí hodnotu jeden týden. K tomuto rozhodnutí vede několik faktorů. Dataset s1 je jednodušší na predikci díky svojí periodicitě dat a výskytu menšího množství extrémních hodnot a abnormalit v charakteru dat. Dá se tedy předpokládat, že k predikci tohoto datasetu potřebujeme menší množství vstupních informací než v případě datasetu s2, který nemá tak dobré vlastnosti.

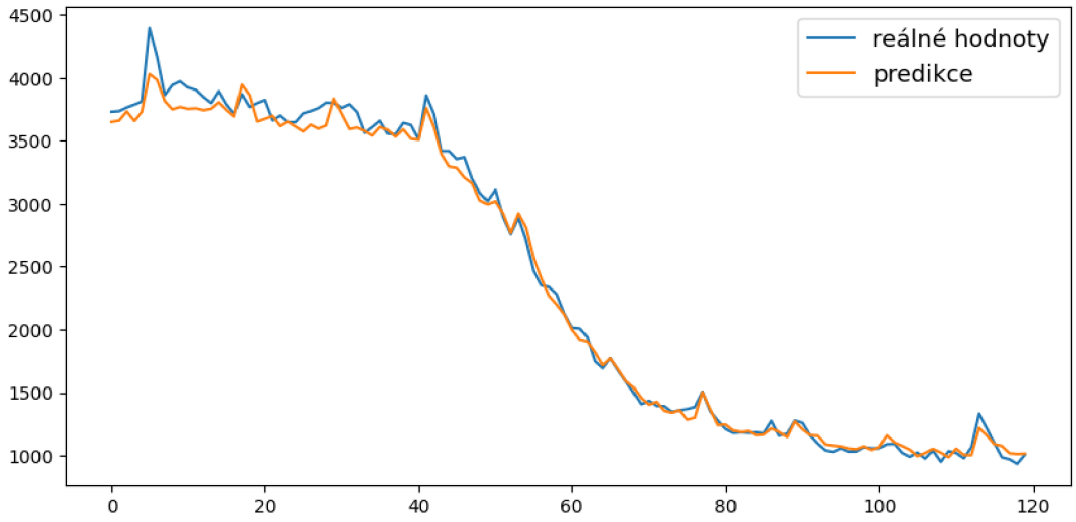
Dalším faktorem při určení optimální délky vstupní posloupnosti je posuzování výsledné predikce. Když se znovu vrátíme k výsledným grafům chyby, rozdíl mezi 15 minutami a jedním dnem se nezdá být natolik výrazný. Navíc zmenšením vstupu také snížíme výpočetní náročnost. Způsob, jakým neuronová síť predikuje následující hodnotu při nedostatku informací, je vidět na obrázku 3.13, který ukazuje část predikce datasetu s1, která je predikovaná na základě předchozích 15 minut.



Obrázek 3.13: Predikce datasetu s1 se vstupem 15 minut

Ve skutečnosti je predikce jen upravená původní číselná řada, která je posunutá o jednu pozici dopředu. Neuronová síť tedy při nedostatku informací jako predikci určí poslední známou hodnotu, kterou upraví pomocí předchozích hodnot. Tento výsledek nelze považovat za užitečnou predikci, jelikož jejím výsledkem je nám známá hodnota.

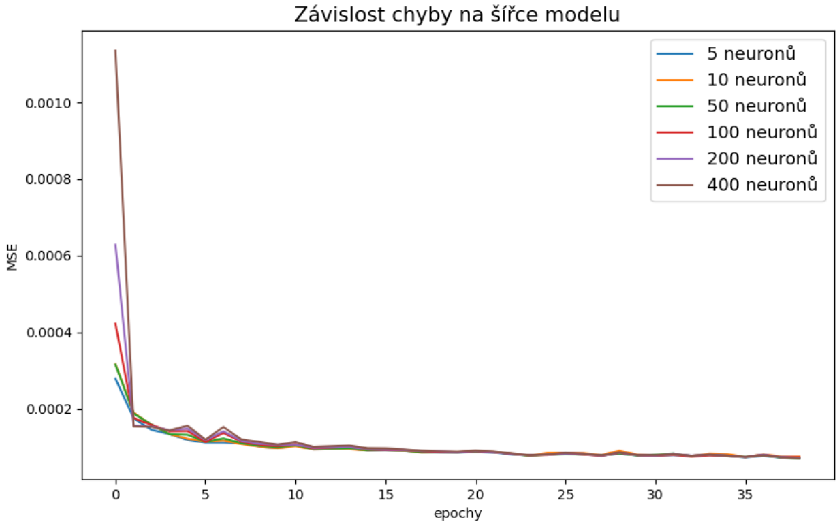
Oproti tomu, když si zobrazíme část predikce datasetu s1 se vstupní posloupností jeden den, dostaneme odlišné výsledky. Tento případ je zobrazen na obrázku 3.14. Nyní predikce již není posunutá a kopíruje původní data. Tento problém jsme si ukázali pouze u datasetu s1, protože je názornější než u s2. V datasetu s2 dochází ke stejnému problému, ale s malou šířkou modelu nevymizí, jako v případě datasetu s1. Z toho důvodu použijeme vstup s lepšími výsledky u datasetu s2.



Obrázek 3.14: Predikce datasetu s1 se vstupem 1 den

3.5.3 Šířka modelu

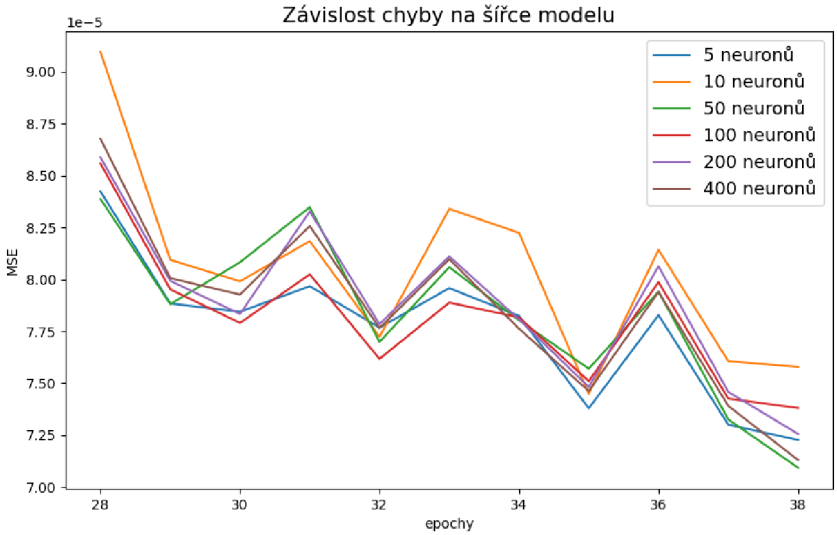
Jako vstup do našeho modelu jsme určili posloupnost o délce jednoho týdne. V této sekci otestujeme, jestli přidání většího množství LSTM bloků vylepší naše dosavadní predikce. Jelikož dataset s1 dosahoval dostatečně přesných výsledků v modelu se čtyřmi LSTM bloky, budeme v této kapitole věnovat pozornost datasetu s2. Pro úsporu času budeme model trénovat pouze 39 epoch. Tento počet byl určen empiricky v průběhu experimentů jako nejlepší poměr přesnosti a časové náročnosti.



Obrázek 3.15: Závislost chyby na šířce modelu datasetu s2

Na obrázku 3.15 můžeme vidět chyby modelů neuronové sítě s 5, 10, 50, 100, 200 a 400 bloky v jedné skryté LSTM vrstvě. Takto vysoké hodnoty byly otestovány kvůli délce vstupních posloupností. Cílem bylo otestovat, jestli pro takto velká vstupní data nebude potřeba velká šířka modelu neuronové sítě pro přesnější predikce. Původním záměrem bylo otestovat až 2016 bloků, stejný počet jako velikost vstupních dat pro jeden týden, což z důvodu časové náročnosti nebylo uskutečněno.

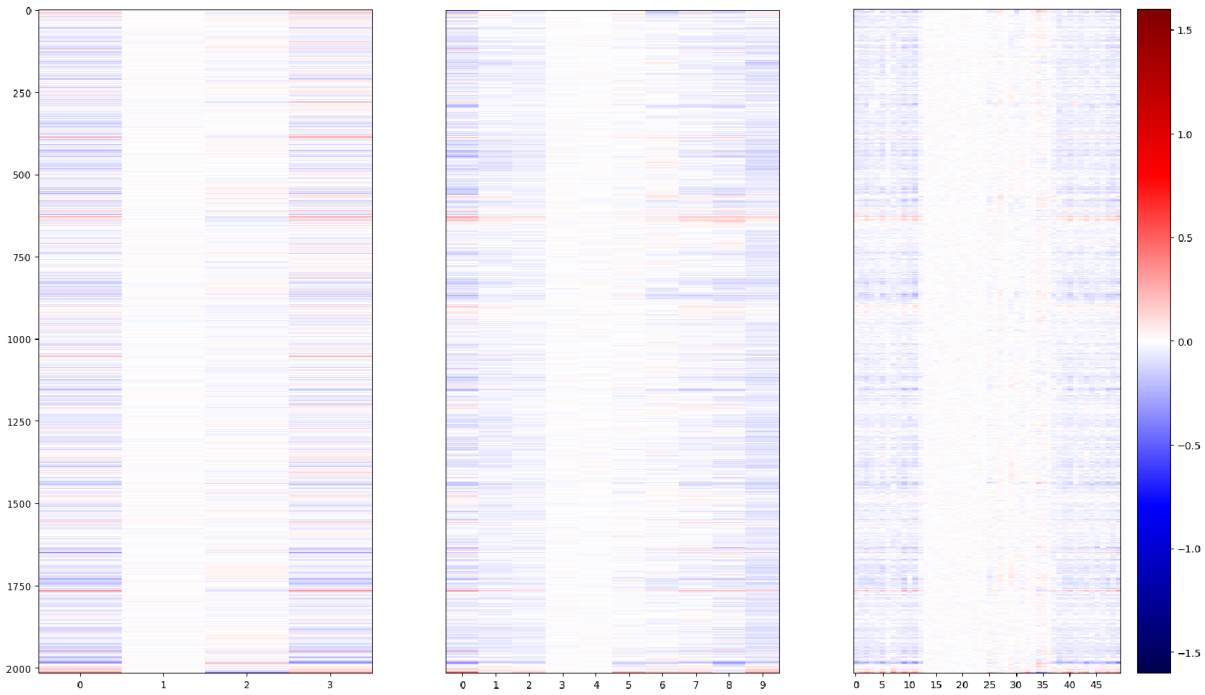
Jak je vidět z detailu posledních 10 epoch na obrázku 3.16, tak s šířkou modelu se



Obrázek 3.16: Detail závislosti chyby na šířce modelu datasetu s2

nezvyšovala přesnost neuronové sítě. Tento závěr je překvapivý a bylo do určité míry předpokládáno, že šířka modelu vylepší dosavadní výsledky. Otázkou je, zda je toto chování způsobeno naším cílem predikovat pouze jedinou hodnotu z velkého množství předcházejících hodnot. Vysvětlením může být, že model pro takto jednoduché zadání nebude benefitovat z rostoucí šířky modelu. Druhou možností může být normalizace dat. Jelikož původní data nabývají velkých hodnot a následná normalizace je převede na interval $\langle 0, 1\rangle$, mohlo vlivem normalizace dojít ke částečné ztrátě informací o původních datech.

Na obrázku 3.17 jsou zobrazeny matice vah modelů s šířkou 4, 10 a 50, které transformují vstupní data. Na základě jejich podobnosti se nejspíše bude jednat o první případ,



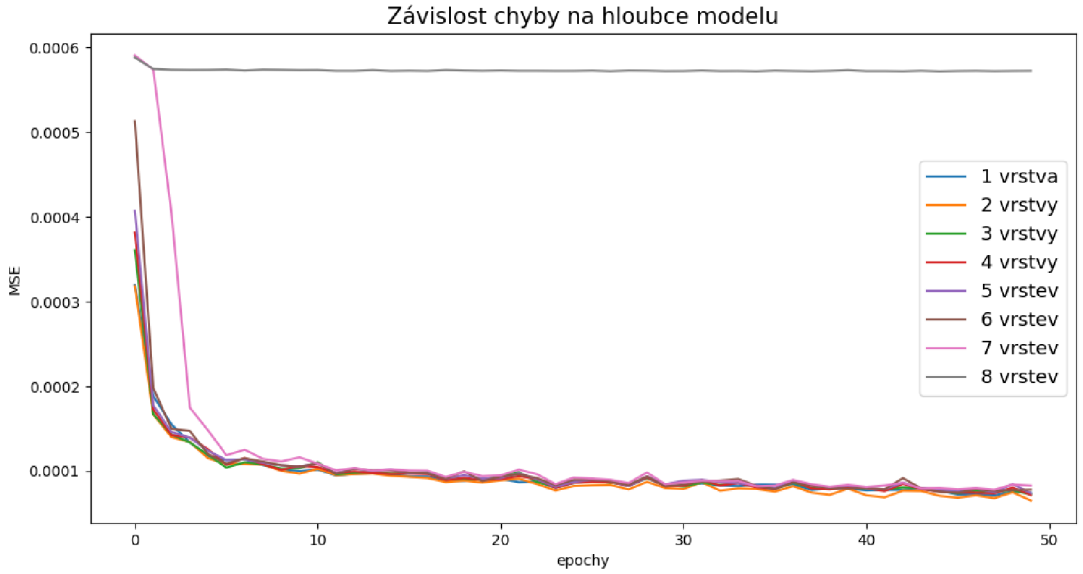
Šířka modelu 4 Šířka modelu 10 Šířka modelu 50

Obrázek 3.17: Porovnání váhových matic vstupů

kdy model nebenefitoval z rostoucí šířky modelu. Přesto se v pozdější části práce budeme věnovat modelu bez normalizace, abychom zjistili, jestli normalizace dat má za následek ztrátu informací. Z výsledků našich pozorování plyne, že šířka modelu neovlivňuje přesnost neuronové sítě. Budeme se model snažit dále vylepšit pomocí rozšíření neuronové sítě do hloubky.

3.5.4 Hloubka modelu

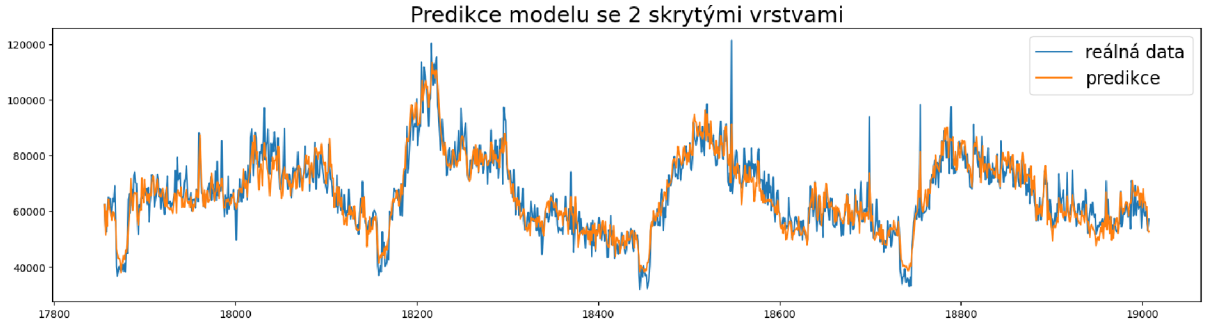
V předchozí sekci rozšiřování modelu do šířky nepřineslo požadované výsledky. Proto se v této části zaměříme na rozšiřování modelu do hloubky. Jako výchozí parametry modelu jsme zvolili 50 epoch a 50 neuronů. Necháváme délku vstupní posloupnosti jednoho týdne, tedy 2016 vstupních bodů. Začneme na modelu s jednou skrytou LSTM vrstvou a postupně budeme vrstvy přidávat. Na obrázku 3.18 jsou zobrazeny průběhy chybových funkcí modelu.



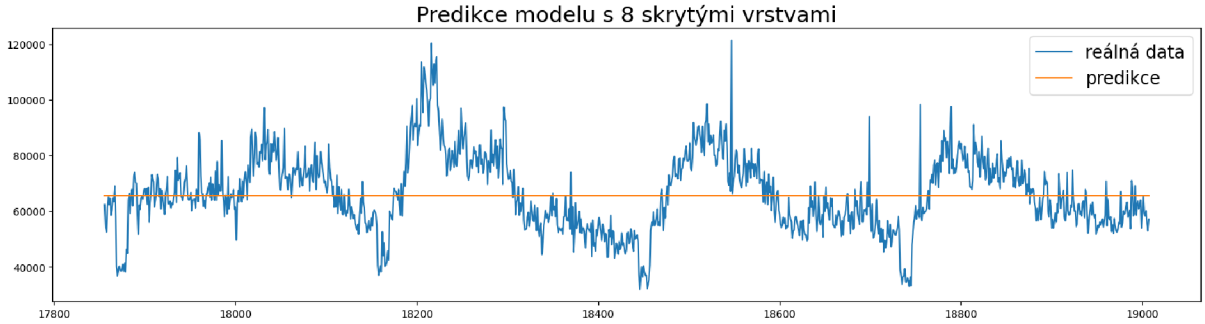
Obrázek 3.18: Závislost chyby na hloubce modelu datasetu s2

Z výsledků vidíme, že hloubka modelu nemá výrazný vliv na výslednou chybu. Nejlépe ve výsledku dopadl model se 2 skrytými LSTM vrstvami, jehož predikce můžeme vidět na obrázku 3.19. Zajímavé chování se projevuje u modelu se 7 a 8 skrytými vrstvami. Oba modely v prvních iteracích mají větší chybu než ostatní modely. Model se 7 vrstvami nakonec začne konvergovat jako ostatní modely, ale model s 8 vrstvami se s rostoucím počtem epoch téměř neučí. Stejný výsledek dostaneme i pro modely s hloubkou větší než 8 vrstev. Predikce tohoto modelu můžeme vidět na obrázku 3.20. Výsledkem predikce je pouze jedna hodnota, která je blízká střední hodnotě trénovacích dat, tedy hodnota blízká minimu střední kvadratické chyby pro průměr. Z předchozích pozorování se nabízí možnost, že hodnota míry učení α je pro tento model již příliš velká. Testování tuto teorii potvrdilo a model začal konvergovat při $\alpha = 0.0001$, ale trvalo 30 epoch, než začala chyba modelu postupně klesat. Avšak model nedosahoval lepší přesnosti a jeho časová náročnost se více než zdvojnásobila.

Jelikož rozšíření sítě mohlo změnit její chování, otestovali jsme znovu různé délky vstupních posloupností a také různé šířky modelu. Stále nejlepší výsledky neuronová síť vykazovala při vstupní posloupnosti s délkou jednoho týdne a šířkou 50 neuronů.

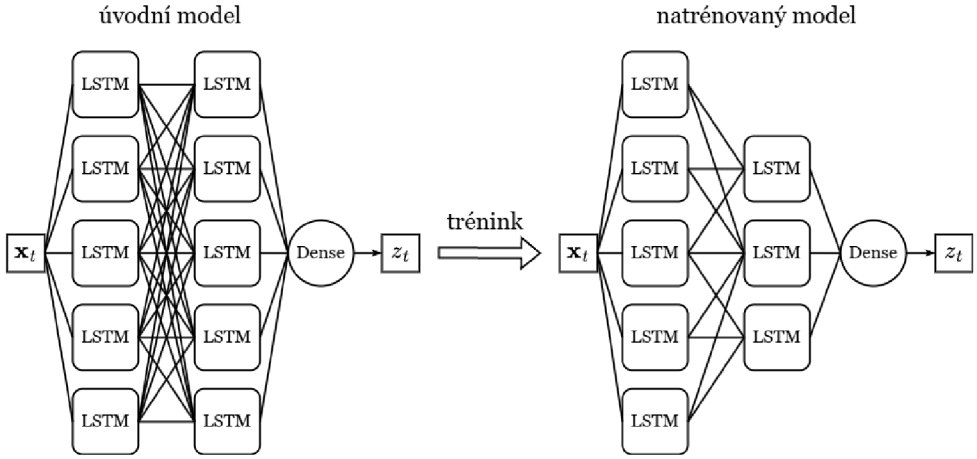


Obrázek 3.19: Predikce modelem se 2 vrstvami datasetu s2



Obrázek 3.20: Predikce modelem s 8 vrstvami datasetu s2

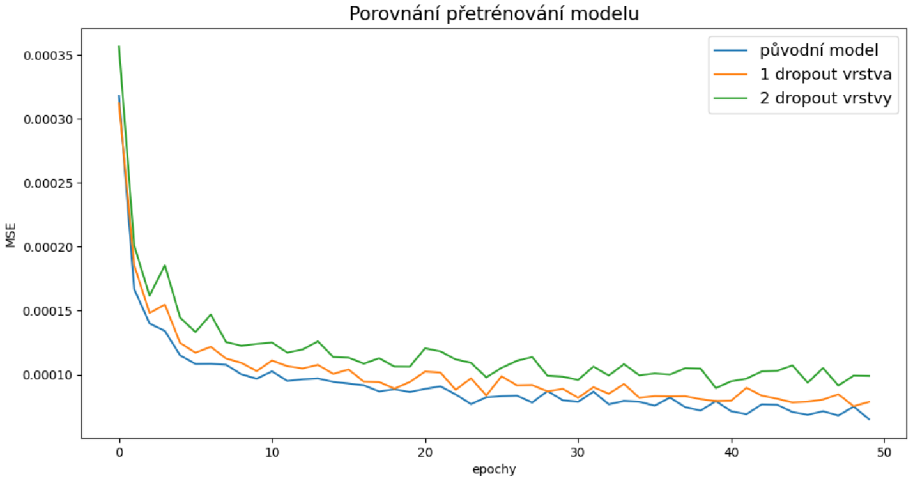
Ve fázi testování se nabízela možnost manuální úpravy jednotlivých vrstev neuronové sítě. Hlavní myšlenou bylo, jestli postupným zmenšováním skrytých vrstev nebude model dosahovat lepší přesnosti, když predikuje pouze jednu hodnotu, ale do modelu vstupují tisíce vstupních hodnot. Tato teorie byla otestována, ale nepřinesla lepší výsledky. Když se nad touto situací zamyslíme, dobereme se k závěru, že neuronová síť dokáže tyto změny struktury provádět sama pomocí nastavení jednotlivých vah modelu. V rámci učení může neuronová síť nastavit váhy blízké nule u nadbytečných cest. Manuálním zmenšením modelu sice můžeme zkrátit dobu učení, ale na druhou stranu omezujeme možnosti celého modelu.



Obrázek 3.21: Změna struktury modelu v průběhu trénování

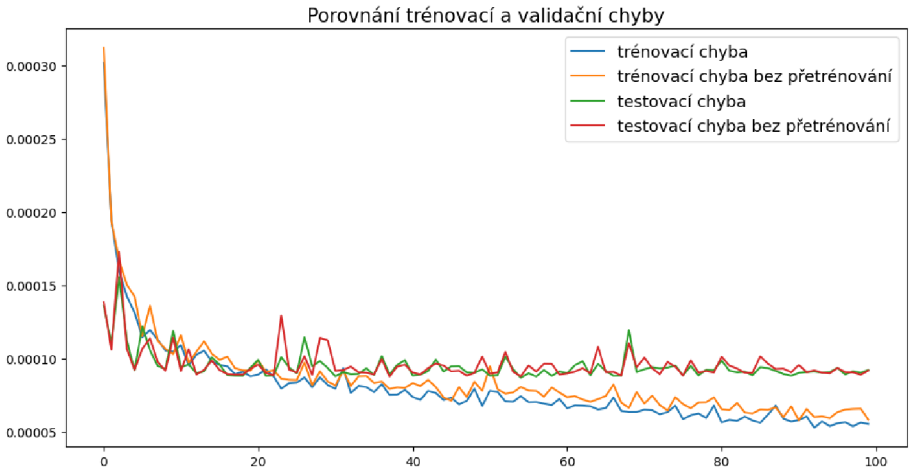
3.5.5 Přetrénování a velikost vzorku modelu

Než prohlásíme náš model za finální, zbývá otestovat několik posledních věcí. V sekci 3.3.1 jsme popsali jev přetrénování neuronové sítě, který má běžně negativní vliv na obecnost modelu. Jelikož máme celkem jednoduchý model, můžeme si dovolit síť znovu natrénovat na konkrétních datech, která chceme predikovat, aby zachytila konkrétní vzory v chování. Stačí nám obecnost modelu vzhledem ke struktuře, nepotřebujeme obecnost predikcí pro různá data se stejnými parametry.



Obrázek 3.22: Porovnání modelů bez přetrénování

Na obrázku 3.22 můžeme vidět porovnání chyby tří modelů. První model je náš doposud nejpřesnější model a další dva modely jsou rozšířené o jednu a dvě Dropout vrstvy. První upravený model náhodně vynechává 10 % vstupů do Dense vrstvy a druhý model má navíc stejnou Dropout vrstvu mezi LSTM vrstvami. Z obrázku můžeme vidět, že upravené modely nedosahují stejné přesnosti při trénování sítě jako původní model. Navíc chyba predikce testovacích dat je větší než u původního modelu o 1 % a 3 %. Přetrénování modelu, které má škodlivý vliv na výsledky predikcí, můžeme pozorovat při vykreslení chyby trénovacích dat a validačních dat v průběhu trénování. Když s klesající chybou trénovacích dat roste chyba testovacích dat, potom má přetrénování negativní vliv na predikování jiných než trénovacích dat.

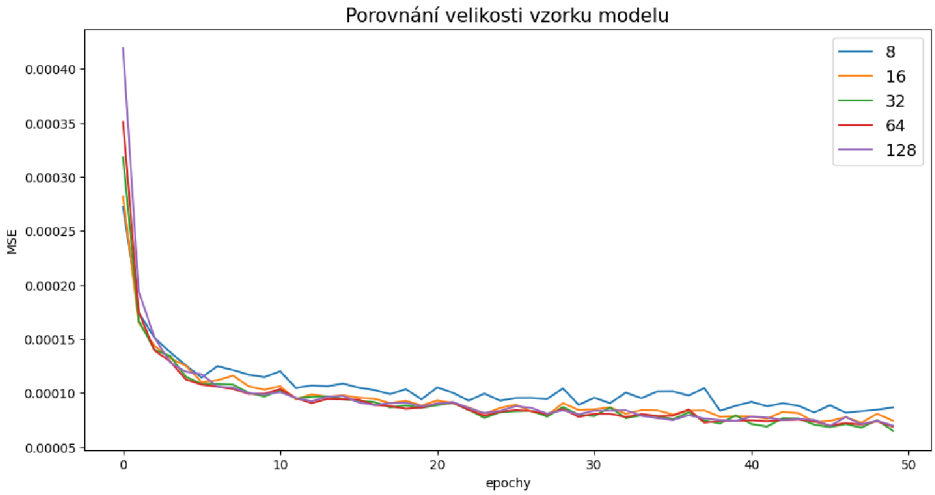


Obrázek 3.23: Porovnání chyb testovacích a trénovacích dat

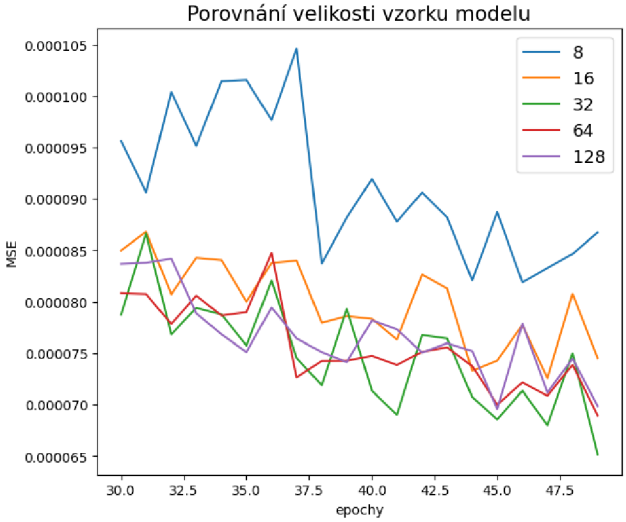
Na obrázku 3.23 můžeme vidět porovnání chyb trénovacích a testovacích chyb v průběhu trénování neuronové sítě s chybami modelu s jednou Dropout vrstvou. Vidíme, že původní model dosahuje lepší přesnosti na trénovacích datech a chyba na testovacích datech je téměř totožná s modelem bránícím přeučení. Jasně vidíme, že síť nevykazuje lepší hodnoty bez přeučení ani nepřichází o schopnost přesné predikce na testovacích datech. Díky tomuto závěru nebudeme v našem modelu využívat Dropout vrstvy a zůstaneme u našeho původního modelu.

Dalším důležitým faktorem v průběhu trénování neuronové sítě je velikost vzorku modelu. V sekci 3.3.3 jsme popsali vliv velikosti vzorku na učení neuronové sítě a časovou náročnost trénování. V rámci testování se budeme snažit naleznout nejlepší poměr přesnosti modelu a časové náročnosti trénování. Jako velikost vzorku využijeme mocniny 2 od 8 do 128. Tyto hodnoty nesouvisí s modelem nebo daty, využívají se čistě z programátorského hlediska pro využití co největšího výpočetního výkonu výpočetních jader.

Na obrázku 3.24 můžeme vidět porovnání chybových funkcí pro různé velikosti vzorku a obrázek 3.25 zobrazuje detailněji posledních 10 epoch trénování. V přiložené tabulce jsou zobrazeny časy trénování pro jednotlivé velikosti vzorku.



Obrázek 3.24: Porovnání chyby modelů v závislosti na velikosti vzorku

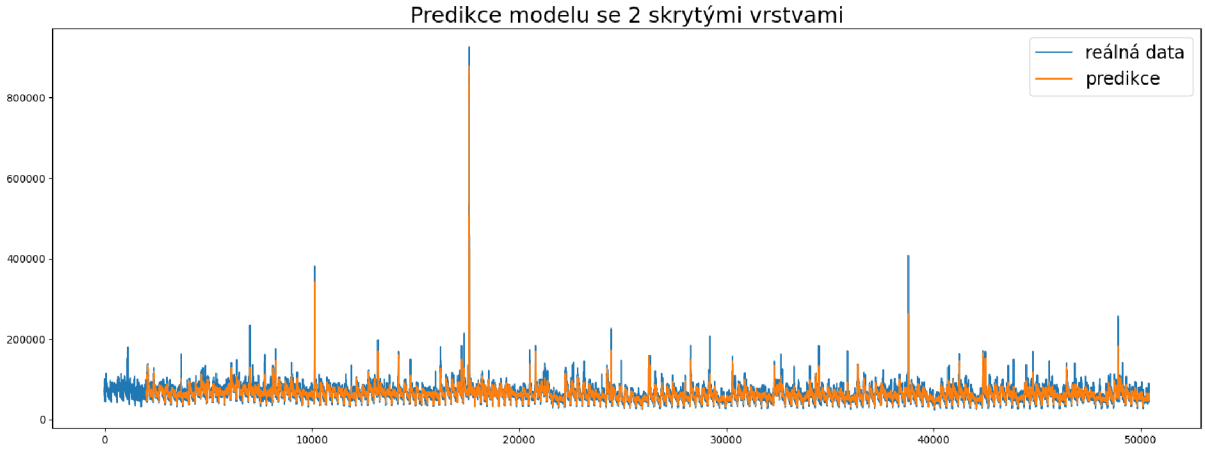


Velikost vzorku	t [min]
8	9
16	5
32	2.25
64	1.66
128	1

Obrázek 3.25 & Tabulka 1: Porovnání přesnosti a časové náročnosti vzorku modelu

Na detailním obrázku můžeme vidět, že nejlepší přesnosti dosahoval model s velikostí vzorku 32 a nejvíce se blíží model s velikostí 64. Model s velikostí 64 je sice o 35 vteřin rychlejší než model s velikostí 32, ale časová náročnost modelu s velikostí 32 je stále poměrně krátká doba vzhledem k pětiminutovým intervalům v našich datech. V tomto případě dáme přednost větší přesnosti před ušetřením času, jelikož trénování sítě se provádí často pouze jednou, ale menší přesnost predikce se projeví s každou predikovanou hodnotou. V případě, že by bylo nutné síť přetrénovat v průběhu predikování v reálném čase, máme stále časovou rezervu pro natrénování i větších datasetů, než používáme k testování. Pokud by u velkých datasetů došlo k překročení 5 minut při trénování, potom budeme volit velikost vzorku 64. Z toho důvodu budeme v našem modelu pro predikci dat používat velikost vzorku 32.

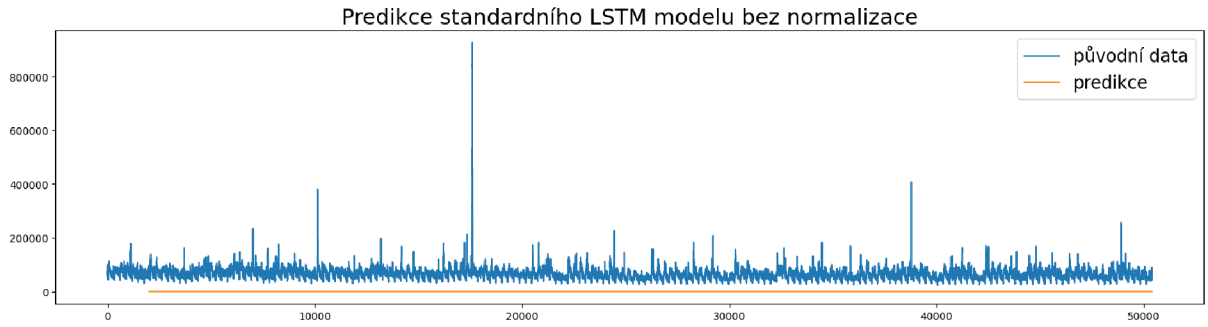
Na závěr shrneme poznatky z testování LSTM neuronové sítě. Náš model bude využívat míru učení $\alpha = 0.001$, která vzhledem k rychlosti konvergence a přesnosti vykazovala nejlepší výsledky a fungovala až pro modely se 7 LSTM vrstvami. Jako vstup jsme zvolili týden dat, který obsahuje 2016 hodnot. Tato posloupnost dosahovala nejlepších výsledků v datasetu s2 a v datasetu s1 byl nepatrný rozdíl s posloupností jednoho dne, která v datasetu s2 dosahovala horších výsledků než jeden týden v datasetu s1. Šířka modelu měla překvapivě minimální vliv na výsledky modelu, ale u modelu s větší hloubkou vykazovala lepší výsledky síť s 50 neurony, byť minimální. Pro náš model jsou dostačující 2 skryté LSTM vrstvy a pro náš účel nebudeme využívat prostředky zamezující přetrénování, naopak použijeme vysoký počet epoch v našem případě si můžeme dovolit 100 epoch a velikost vzorku 32. V případě potřeby můžeme využít větší velikost vzorku pro snížení časové náročnosti. Na obrázku 3.26 můžeme vidět výsledné predikce spočítané našim modelem. Výsledná chyba našeho modelu je 7.4 % z průměru predikovaných hodnot.



Obrázek 3.26: Predikce datasetu s2

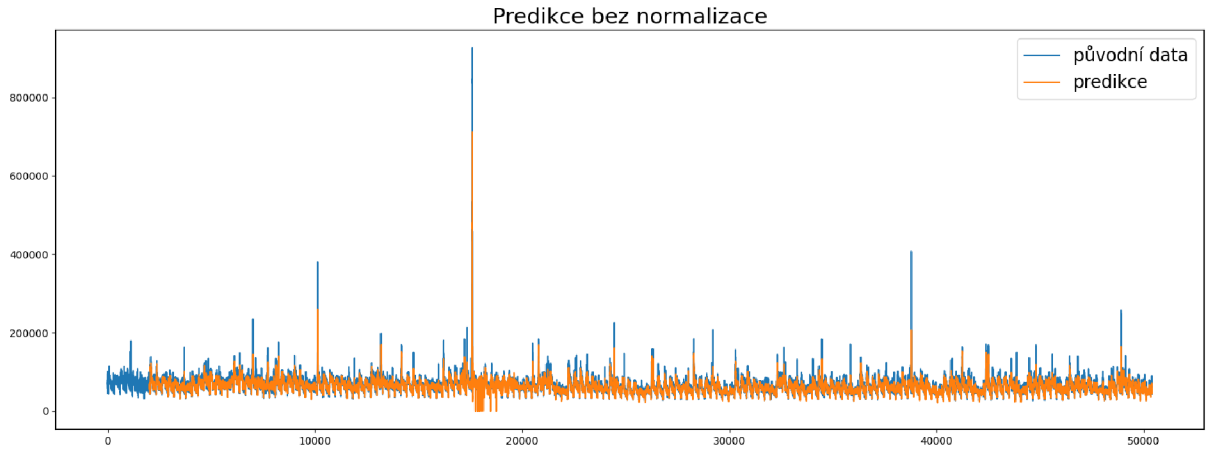
3.5.6 Model bez normalizace

Předtím, než se začneme věnovat detekci anomálií, prozkoumáme, jestli normalizace dat nezkresluje predikce neuronové sítě. Abychom mohli náš model využít, bude potřeba lehce pozměnit LSTM vrstvy. Jak už bylo zmíněno v sekci 3.4, pokud bychom se pokusili využít stejný model na původní data, predikce nebude fungovat, díky aktivační funkci tangens hyperbolický (2). Výsledek takového modelu můžeme vidět na obrázku 3.27. Model nebyl schopný si poradit se vstupními daty a vrací pouze nulové hodnoty.



Obrázek 3.27: Predikce datasetu s2 standardním LSTM modelem bez normalizace

Abychom mohli použít model se stejnou stavbou jako v předchozí sekci, budeme muset upravit aktivační funkci LSTM vrstev. Potřebujeme funkci, která zobrazí vstupní hodnoty do $\mathbb{R}$. Jelikož naše data neobsahují záporné hodnoty a jedná se o časové záznamy, nebude obsahovat záporné hodnoty ani v budoucnu. Z toho důvodu se nabízí aktivační funkce ReLU (3), která odpovídá indentitě na $\mathbb{R}_0^+$. Avšak při změně aktivační funkce model po natrénování predikoval pouze jednu hodnotu. Stejně jako v případě modelů s hloubkou větší než 8 bylo třeba zmenšit o jeden řád míru učení α. Po této úpravě model začal konvergovat jako obvykle. Ovšem zmenšení míry učení způsobilo snížení přesnosti o 28 % oproti normalizovanému modelu. Predikce spočítané tímto modelem můžeme vidět na obrázku 3.28



Obrázek 3.28: Predikce datasetu s2 upraveným modelem bez normalizace

Z obrázku vidíme, že tento model predikuje nižší hodnoty v místě anomálií, což by bylo užitečné v pozdější fázi při detekci anomálií. Jenže aktivační funkce ReLU nefunguje tak dobře jako filtr vysokých vstupních hodnot a toto chování je vidět na predikcích za největší anomálií v datasetu. Zhruba od hodnoty 17000 epoch je vidět, že jeden týden predikcí po největší anomálii predikuje nepřesné hodnoty. Z důvodu snížené přesnosti a výkyvům v predikcích vzhledem k anomáliím budeme pro predikci dále používat model sestavený v sekci 3.5.

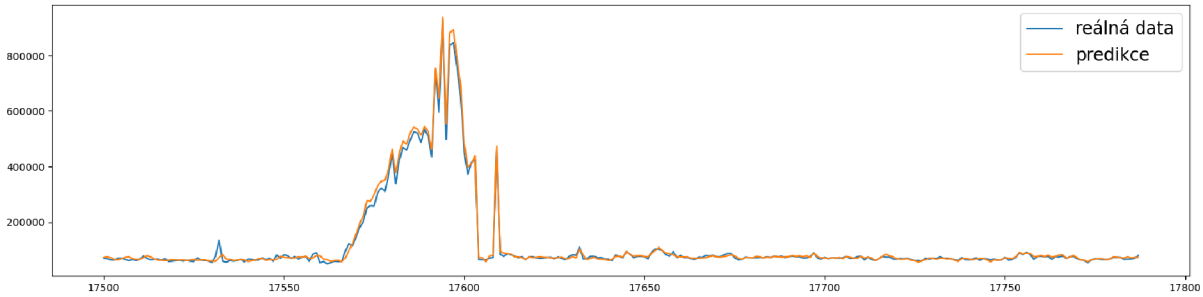
4 Detekce anomálií

V předchozí části práce jsme se soustředili na vytvoření modelu neuronové sítě a sestavili jsme vhodný model pro predikci časových řad. Nyní použijeme tento model k detekci anomálií v datasetu. Využijeme predikce vygenerované naším modelem, které budeme porovnávat s původními hodnotami, a na základě rozdílu jejich vzdáleností určíme, zda se jedná o anomálii, či nikoliv. Hraniční hodnotu nebo také časté označení threshold označíme $\varkappa$. Množinou anomálií rozumíme

$$A(X_t) = \{x \in X_t, |x - P(x)| > \varkappa\},$$

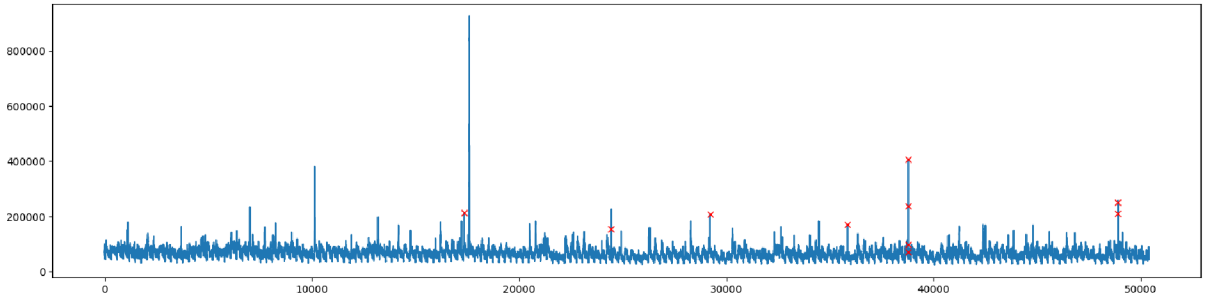
kde $P(x)$ označuje predikci prvku x.

Pokud bychom nyní použili náš model a nastavili pevně danou hranici $\varkappa$ určující anomálie, nedostaneme výsledek, se kterým bychom byli spokojeni. Z výsledných predikcí v kapitole 3.5, které můžeme vidět na obrázku 3.26, vychází paradoxně jako největší problém přesnost predikcí. Jak se dá očekávat, když jsme trénovali model na datech s anomáliemi, bude tento model anomálie i predikovat. Úsek jednoho dne predikcí v místě s největšími anomáliemi datasetu s2 je zobrazen na obrázku 4.1.

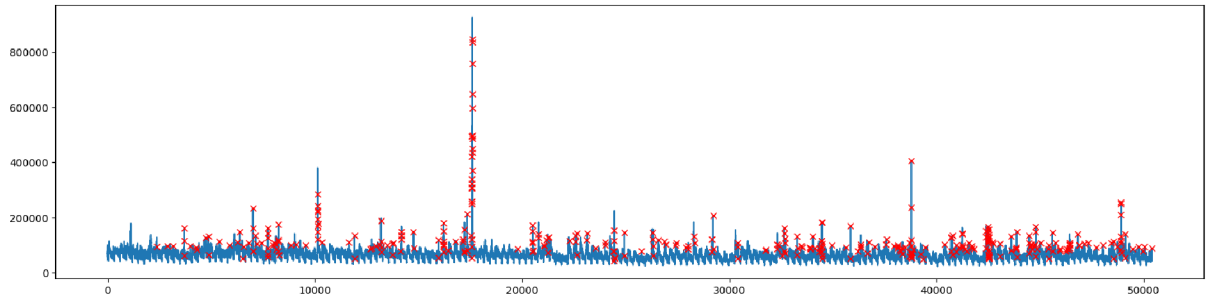


Obrázek 4.1: Predikce anomálie datasetu s2

Vidíme, že predikce velmi přesně kopírují hodnoty anomálií. Při zvolení vysokého $\varkappa$ vzhledem k měřítku našeho datasetu dostaneme výsledek zobrazený na obrázku 4.2. Výsledkem je malý počet bodů, ale tyto body nepředstavují největší anomálie. Na druhou stranu při zvolení menšího $\varkappa$ získáme vysoký počet potencionálních anomálií. Část vybraných bodů se dá považovat za opravdové anomálie, avšak neobsahuje všechny anomální hodnoty. Naopak obsahuje i spoustu bodů, které anomálie s nejvyšší pravděpodobností nejsou. Anomálie vybrané pomocí nízkého $\varkappa$ můžeme vidět na obrázku 4.3. Tyto výsledky nemůžeme považovat za validní detekování anomálií.



Obrázek 4.2: Detekce anomálií datasetu s2 pro velké $\varkappa$



Obrázek 4.3: Detekce anomálií datasetu s2 pro malé $\varkappa$

Přesné predikování anomálií nás nutí znovu pozměnit náš dosavadní model neuronové sítě. Přesněji řečeno nebudeme měnit stavbu modelu, ta funguje výborně na predikování budoucích hodnot, ale zaměříme se na trénovací data. Jelikož náš trénovací dataset obsahoval anomální hodnoty, parametry modelu počítají s budoucími anomáliemi, a budou je predikovat. Abychom mohli predikce považovat za směrodatnou informaci, zda se jedná o anomálii či nikoliv, potřebujeme náš model natrénovat na datech bez anomálií.

Abychom získali dataset, který nebude obsahovat anomální hodnoty, budeme si muset vypomoci statistikou pro jejich určení. Pozorný čtenář se může ptát, proč se v této práci zabýváme umělou inteligencí pro detekci anomálií, když v průběhu musíme použít statistické metody, abychom z datasetu odstranili anomálie. Potom bychom mohli k detekci anomálií rovnou použít statistické metody. Tuto otázku si zodpovíme v kapitole 5.7.

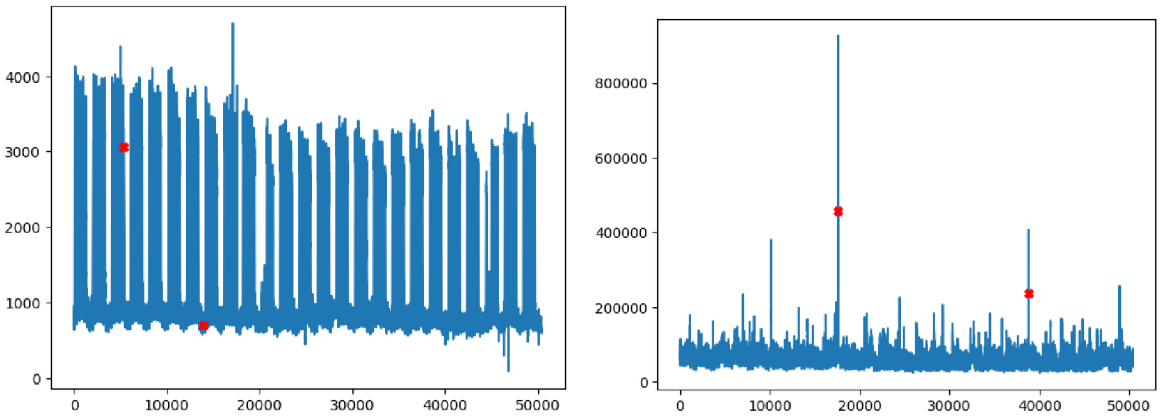
Neuronová síť je schopna dostatečně zachytit charakter dat i z menšího úseku datasetu. Z toho důvodu nemusíme provádět detekci anomálií na celém datasetu, ale pouze na vybraném úseku. Pokusíme se vybrat úsek s co nejméně anomáliemi, abychom museli filtrovat co nejméně hodnot. Z tohoto úseku odebereme anomálie a použijeme jej pro natrénování neuronové sítě. Díky tomu bude neuronová síť predikovat pouze hodnoty bez anomálií a můžeme její predikce využít pro detekci anomálií v původním datasetu.

Největším problémem je nalezení vzorku datasetu, který obsahuje co nejméně anomálií a zároveň obsahuje dostatek informací o chování dat. První možností je, že úsek dat budeme muset vybrat ručně. Pokud se vrátíme k problému, že detekujeme anomálie v reálném čase, budeme mít k dispozici pouze počáteční dataset, který budeme muset využít jako vzorový. Ten pouze očistíme od anomálií a použijeme k natrénování neuronové sítě.

Druhou možností je vymyslet algoritmus, který vybere úsek dat za nás. Bohužel z různorodosti dat je téměř nemožné navrhnout algoritmus, který bude fungovat pro všechny datasety s různým typem chování. V datasetu s2 se největší anomálie projevují jako vysoké hodnoty. Můžeme proto vytvořit algoritmus, který nalezne největší hodnoty v daném datasetu a poté vybere úsek předem zadané délky s nejmenším počtem vysokých hodnot. Bohužel v případě datasetu s1 se nepodařilo nalézt vyhovující metodu, která by vybrala vhodný úsek dat pro trénování. Pokoušeli jsme se detekovat anomálie jak na původním datasetu, tak i na reziduích získaných pomocí dekompozice datasetu, ale nezískali jsme požadované výsledky.

U datasetu s1 jsme ručně vybrali vhodný vzorek dat. Vybírali jsme data s délkou dvou měsíců u datasetu s2 a jedním měsícem u datasetu s1, což se během testování ukázalo jako dostatečný vzorek pro natrénování neuronové sítě. Na následujících obrázcích můžeme vidět vybrané úseky s délkou jednoho měsíce. Červené body na obrázku značí hraniční body, které do datasetu nezahrnujeme.

Nyní, když máme vybrané vhodné úseky dat, pokusíme se tyto datasety vyčistit od anomálií. Protože potřebujeme anomální hodnoty nejen nalézt, ale také nahradit, nemá



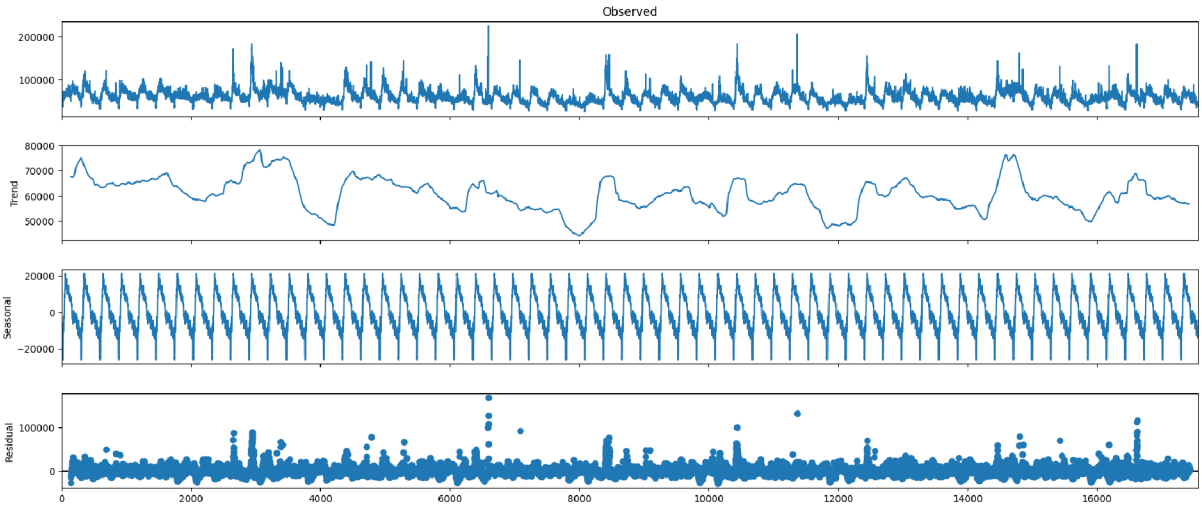
Dataset s1 Dataset s2

Obrázek 4.4: Nalezení trénovacího datasetu nejmenšími anomáliemi

smysl hledat anomálie přímo ve vzorovém datasetu, protože potom nevíme, jakými hodnotami máme anomálie nahradit, abychom nevytvořili změny v chování dat. Z toho důvodu použijeme dekompozici časové řady, která řadu X_t rozloží následujícím způsobem

$$X_t = T_t + S_t + Y_t,$$

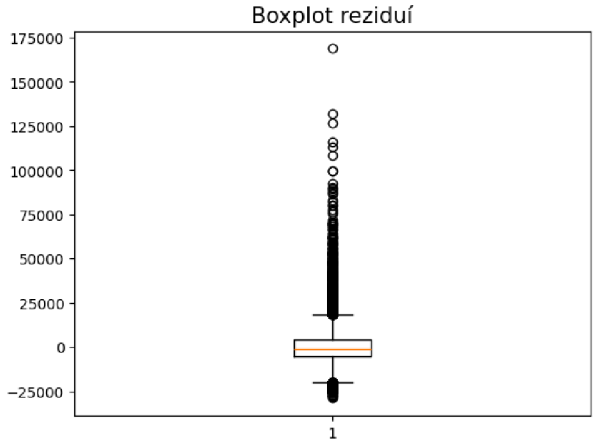
kde T_t je trendová složka, S_t je sezónní složka a Y_t jsou rezidua. Dekompozice časové řady je detailněji popsána v kapitole 5. Dekompozici vzorového datasetu s2 můžeme vidět na obrázku 4.5.



Obrázek 4.5: Dekompozice vzorového datasetu s2

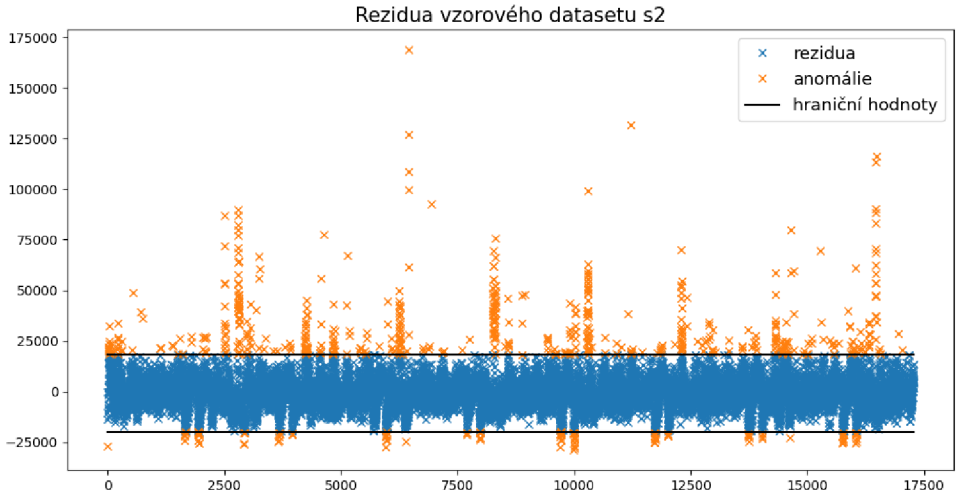
Trendová složka a sezónní složka zachovávají charakter dat, proto budeme anomálie detekovat na reziduích. Jelikož chceme náš model co nejobecnější, nemáme žádné předpoklady na data samotná. Z toho důvodu se k detekci anomálií nabízí pouze neparametrické metody. Využijeme metodu Boxplot fungující na základě kvartilů. Kvartil se určuje pomocí p-kvantilu, který definujeme jako reálné číslo $x_p = \inf\{x \in Z, F(x) \geq p\}$, kde Z označuje základní soubor a $F(x)$ je distribuční funkce Z. Horním a dolním kvartilem rozumíme hodnotu p-kvantilu pro $p = 0.75$ a $p = 0.25$. Hodnota kvantilu pro $p = 0.5$ se označuje jako medián. Rozdíl horního a dolního kvartilu se nazývá kvartilová odchylka (IQR).

Metoda box plot najde horní a dolní kvartil, které v grafu vykreslí jako obdélník a přidá linii značící medián. Následně vytvoří takzvané vousy, které rozšiřují interval mezi kvartily na každou stranu o délku 1.5 IQR. Pokud vousy nabývají hodnot mimo maximum nebo minimum datasetu, jsou omezeny těmito hodnotami, proto vousy Boxplotu nemusí být stejně dlouhé. Hodnoty mimo interval vymezený vousy Boxplotu považujeme za anomálie (outliery). Hodnoty anomálií jsou následně vykresleny v Boxplot grafu.



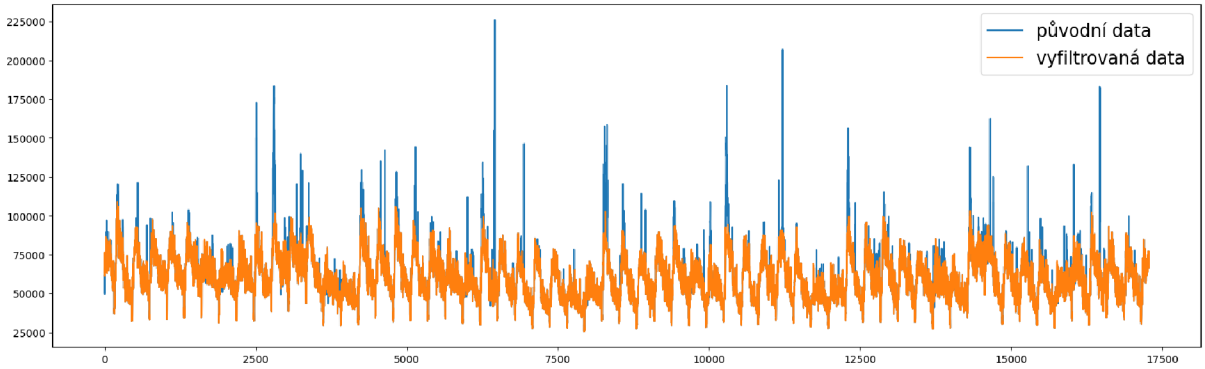
Obrázek 4.6: Boxplot reziduí

Na rezidua použijeme metodu Boxplot, jejíž výsledky můžeme vidět na obrázku 4.6. Zde vidíme hodnoty kvartilů, mediánu a anomálií, ale není zde vidět, o jaké anomálie se jedná. Graf s rezidui, kde jsou vykresleny anomálie a hraniční hodnoty Boxplotu, můžeme vidět na obrázku 4.7. Rezidua prohlášená za anomálii nahradíme nulovou hodnotou, jelikož předpokládáme, že rezidua mají nulovou střední hodnotu. Vyfiltrovaný vzorový dataset získáme zpětným sečtením trendové, sezónní a upravené reziduální složky.



Obrázek 4.7: Anomálie reziduí vzorového datasetu s2

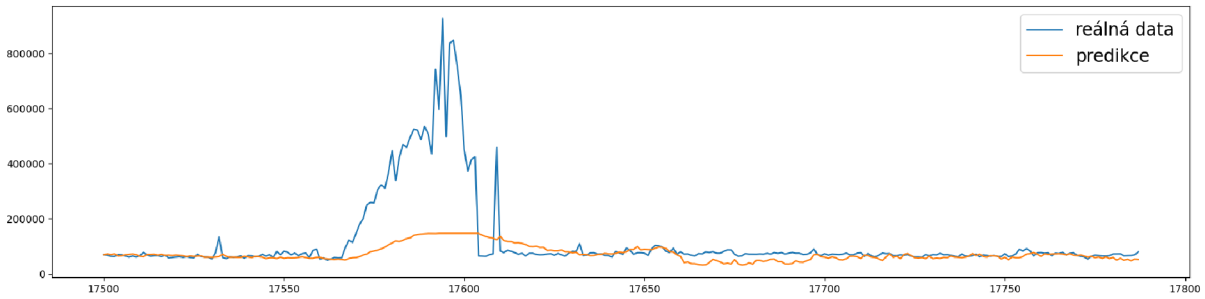
Tímto procesem jsme získali vhodná data pro natrénování neuronové sítě. Výsledná očištěná data můžeme vidět na obrázku 4.8. Stejným způsobem jsme vytvořili trénovací data pro dataset s1. Nyní můžeme tato data použít k natrénování nového modelu, který nebude predikovat anomálie. Použijeme stále stejný model, pouze změníme trénovací data. Výsledek natrénovaného modelu je velmi podobný jako u celého datasetu, predikce mají průměrnou chybu 6.5 %, což je o procento lepší výsledek než u celého datasetu.



Obrázek 4.8: Vzorový dataset s2

Použitím vzorového datasetu na trénink nastává problém s normalizací dat. Jelikož neuronovou síť jsme natrénovali na datech, která mají jiné maximum a minimum než celý dataset. Tím pádem mají jiné měřítko než původní dataset. Budeme si muset zvolit, které měřítko budeme používat. Ve výsledku by to neměl být rozdíl, pouze když zvolíme menší měřítko ke transformaci, dostaneme po normalizaci původních dat vstupní hodnoty větší než 1. To nebude problém díky aktivačním funkcím tangens hyperbolický a sigmoida, které větším hodnotám přiřadí téměř stejnou váhu jako hodnotám v okolí 1. V našem případě dává smysl zvolit menší měřítko, které lépe reflektuje hodnoty bez anomálií.

Samozřejmě u extrémních hodnot se projeví vliv na predikce, ale tyto hodnoty budou s největší pravděpodobností anomálie, které nechceme používat jako vstupy do neuronové sítě z důvodu ovlivnění predikcí. Když se vrátíme k původní myšlence použít rozdíl vzdálenosti mezi predikcí a reálnou hodnotou, ukážeme si predikci stejného úseku jako na obrázku 4.1, ale použijeme nový model. Výsledek můžeme vidět na obrázku 4.9. Z obrázku je patrné, že predikce již nekopíruje anomální hodnoty, ale vliv anomálií je stále značný. Vysoké hodnoty anomálií způsobují nárůst predikcí a jejich následný pokles. Toto chování by způsobovalo označení validních hodnot za anomálie.



Obrázek 4.9: Predikce anomálie datasetu s2 novým modelem

Z tohoto důvodu je potřeba při predikování hodnot upravovat vstupy. Ideální scénář by vypadal následujícím způsobem. V průběhu predikování v momentě, kdy je hodnota označena za anomálii, bychom její hodnotu nahradili hodnotou její predikce ve všech ostatních vstupech. Potom by již hodnota anomálie nezasahovala do dalších predikcí. Bohužel toto není možné, protože narazíme na omezení způsobená výpočetním výkonem. Jak bylo popsáno v kapitole 3.2, naše vstupní vektory mají délku 2016 (jeden týden) a obsahují po sobě jdoucí hodnoty, které se s každým dalším vstupním vektorem posunou o jednu hodnotu dopředu. Tím pádem jedna vstupní hodnota je obsažena v 2016 vstupních vektorech (když nepočítáme prvních 2015 hodnot datasetu), které musíme změnit. Samotná změna

nepředstavuje problém, ale narazíme na problém při použití funkce `model.predict()` v knihovně TensorFlow. Tato funkce je přizpůsobená na predikování velkých datasetů v případě, že do funkce vložíme všechny vstupní hodnoty najednou, v našem případě matici X, a funkce vrátí všechny predikce najednou a nelze v průběhu predikování měnit její vstupní data. Avšak spouštění této funkce je časově náročné, přímo v dokumentaci se nedoporučuje přes tuto funkci iterovat. Pro náš problém by bylo ideální predikovat hodnoty po jednom a na základě výsledku predikce změnit další vstupy. Tento problém by se dal vyřešit implementací naší neuronové sítě například v jazyce C++, ale to je nad rámec této práce.

4.1 Metoda Double-pass

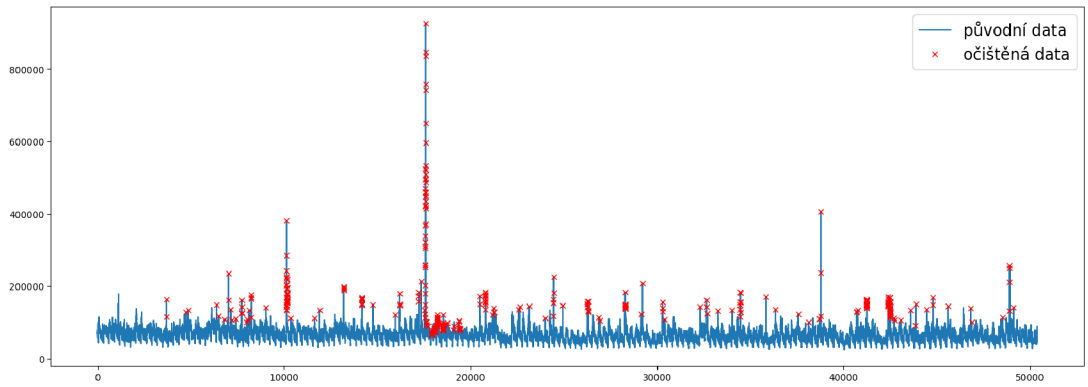
Budeme se snažit náš problém vyřešit jiným způsobem. Jako první vytvoříme metodu, která bude cílit na nízkou časovou náročnost. Tuto metodu budeme nazývat Double-pass. Její princip je založen na získaných výsledcích z minulých predikcí. Jak můžeme vidět na obrázku 4.9, predikce mají tendenci snižovat hodnoty anomálií. Této vlastnosti využijeme a provedeme predikci celého datasetu.

Jelikož považujeme predikce jako „správné“ hodnoty, tak po detekci anomálií pomocí thresholdu $\varkappa$ označíme za anomálie i validní hodnoty. Tyto nepřesnosti zatím ignorujeme a nahradíme detekované anomálie příslušnými predikcemi. Takto nám vznikne nový dataset, který obsahuje nižší hodnoty anomálií, ale také vyšší hodnoty ovlivněné anomáliemi při predikci. Proto tento dataset znovu použijeme pro predikování. Získáme nové predikce, které jsou méně ovlivněné anomáliemi a znovu provedeme porovnání pomocí thresholdu $\varkappa$ s původním datasetem. Dostaneme novou množinu anomálií, kterou prohlásíme za finální anomálie. Tímto způsobem do určité míry snížíme vliv anomálií na predikci a pomocí thresholdu $\varkappa$ můžeme ovlivnit tolerance, podle kterých se budou detekovat anomálie. Velikost $\varkappa$ je zapotřebí určit empiricky v závislosti na datasetu a požadované přesnosti, s jakou se budou detekovat anomálie. Navíc výstupem z tohoto algoritmu není pouze množina anomálií, ale také očištěný dataset, který byl anomálií zbaven.

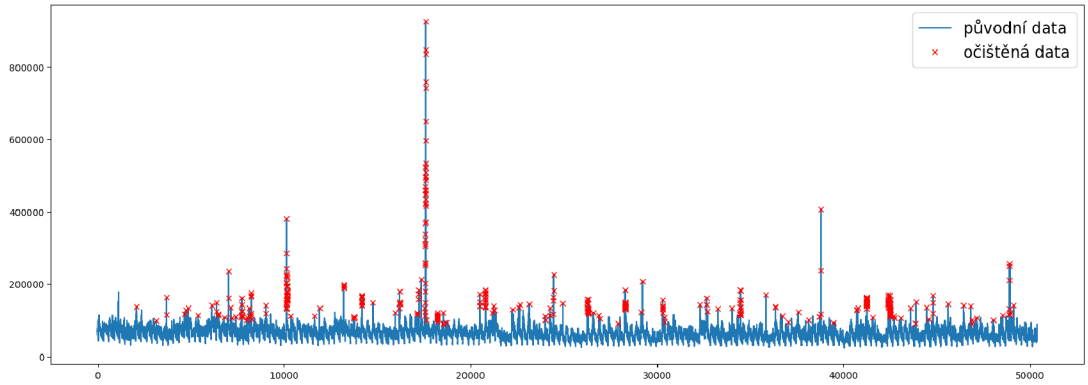
Metoda 3 Double-pass

```
def double_pass(dataset, input_length, neurons, kappa):
    model=set_model(input_length, neurons)
    model.load_weights()

    for i in range(2):
        X = create_input(scaler.transform(dataset),input_size = input_length)
        prediction=scaler.inverse_transform(model.predict(X))
        anomaly_index=[]

        for i in range(len(prediction)):
            if(abs(prediction1[i]-dataset[i+input_length])>kappa):
                dataset[i+input_length]=prediction1[i]
                anomaly_index.append([i+input_length])
    return dataset, anomaly_index
```

Na obrázku 4.10 jsou znázorněny výsledky jednotlivých fází algoritmu u datasetu s2. Můžeme vidět, že v první fázi došlo k silnému ovlivnění predikcí, proto došlo k označení většího počtu anomálií, speciálně za největší anomálií v datasetu. Druhý průchod odstranil většinu falešných anomálií, ale stále některé anomálie bychom mohli považovat za falešné. Celkový počet detekovaných anomálií pomocí této metody je 300. Detekce trvala přibližně 5 vteřin.



(a) První průchod



(b) Druhý průchod

Obrázek 4.10: Průběžné výsledky metody Double-pass datasetu s2

4.2 Metoda se změnou vstupů

V této metodě se pokusíme přiblížit ideálnímu případu, kdy v případě detekce anomálie se anomální hodnota nahradí predikcí při vstupu do neuronové sítě, aby neovlivňovala výsledky dalších predikcí. Našemu modelu trvá predikce celého datasetu přibližně 2 vteřiny. Pro představu pomocí předchozí metody jsme detekovali 300 anomálií u datasetu s2. Pokud bychom znovu predikovali při detekci anomálie, detekce trvala přes 10 minut, a to jsme navíc detekovali pouze nejvíce výrazné anomálie. Abychom tento proces urychlili, budeme muset snížit množství predikcí, které se v průběhu detekování provedou.

Využijeme vlastnosti neuronové sítě, kterou můžete vidět v levé části obrázku 4.9, kde se vyskytuje jedna výraznější anomálie, která zásadně neovlivnila následující predikce. K ovlivnění došlo, až po výskytu několika větších anomálií po sobě. Z tohoto důvodu se zaměříme na po sobě jdoucí anomálie. Budeme predikovat dataset X_t a velikost vstupního vektoru zvolíme k, predikce datasetu X_t označíme $P(X_t)$. K nové predikci dojde až v případě, když bude detekováno p po sobě jdoucích anomálií. Stejně jako v minulé metodě budeme anomálie detekovat pomocí thresholdu $\varkappa$. Do té doby bude průběžně vytvářen nový dataset $\widehat{X}_t$, který bude obsahovat stejné hodnoty jako X_t, až na anomálie, které budou nahrazeny predikcemi. V iteraci t_A, kdy bude detekováno p po sobě jdoucích anomálií, dojde k nové predikci. Nyní musíme vytvořit nový dataset Y_t následujícím způsobem

$$Y_t = \begin{cases} \widehat{X}_t, & t - k - p < t < t_A \\ X_t, & t \geq t_A. \end{cases}$$

Následně predikujeme dataset Y_t a novými predikcemi nahradíme p minulých predikcí,

které ovlivňují výsledky a nahradíme i budoucí predikce. Tento algoritmus se opakuje pokaždé, co je detekováno p po sobě jdoucích hodnot.

$$P(X_t) = P(Y_t), \quad t > t_A - p.$$

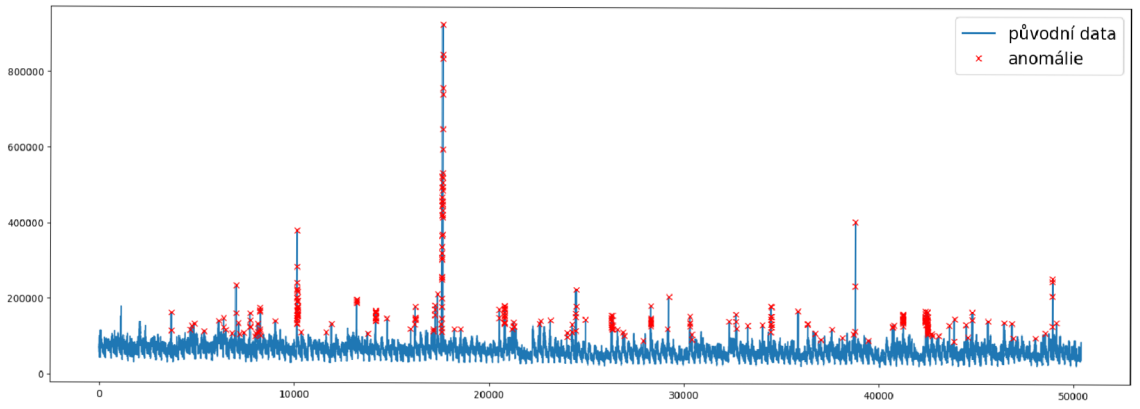
Všimněme si, že dataset Y_t má menší velikost než X_t, jelikož k predikci v čase t potřebujeme pouze $t - k$ předchozích hodnot. Jelikož chceme opravit i predikce ovlivněné p anomáliemi potřebujeme k predikcím $t - k - p$ předchozích hodnot. Tímto algoritmem získáme predikce $P(X_t)$, které jsou méně ovlivněné anomáliemi a s jejich využitím můžeme určit množinu anomálií $A(X_t)$.

Metoda 4 Se změnou vstupů

```
def double_pass(dataset, input_length, neurons, kappa, p):
    model=set_model(input_length, neurons)
    model.load_weights()
    X = create_input(scaler.transform(dataset),input_size = input_length)
    prediction=scaler.inverse_transform(model.predict(X))
    anomaly_index=[]
    anomaly_count=0

     for i in range(len(prediction)):
        if(abs(prediction1[i]-dataset[i+input_length])>kappa):
            anomaly_count+=1
            if(anomaly_count==p):
                new_dataset=np.zeros((len(dataset)-i+anomaly_count,1))
                new_dataset[0:input_length+p]=clean_data[i-p:i+input_length]
                new_dataset[input_length+p:]=dataset[i+input_length:]
                X = create_input(scaler.transform(new_dataset),look_back= input_length)
                prediction[i-p:]=scaler.inverse_transform(model.predict(X))
                clean_data[i-p+input_length:i+input_length]=prediction[i-p:i]
                anomaly_count=0
            clean_data[i+input_length]=prediction[i]
        else:
            clean_data[i+input_length]=dataset[i+input_length]
            anomaly_count=0

        for i in range(len(prediction)):
            if(abs(prediction[i]-dataset[i+input_length])>kappa):
                anomaly_index.append([i+input_length])
                clean_data[i+input_length]=prediction[i]
            else:
                clean_data[i+input_length]=dataset[i+input_length]
    return clean_data, anomaly_index
```

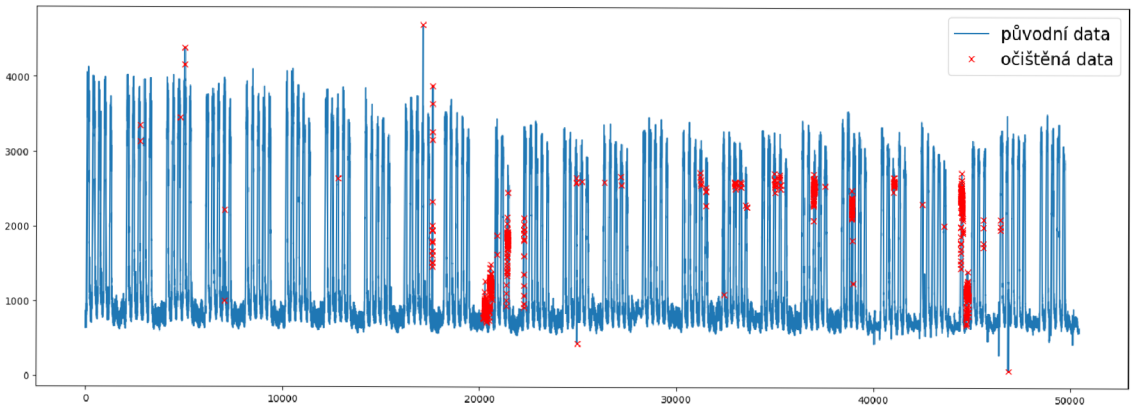
Výsledek získaný touto metodou je zobrazen na obrázku 4.11. Hodnota thresholdu $\varkappa$ byla zvolena stejná jako u předešlé metody a bylo zvoleno $p = 5$. Detekce trvala přibližně 50 vteřin. Z výsledku vidíme, že všechny vysoké hodnoty byly detekovány jako anomálie a počet bodů, které pravděpodobně nejsou anomálie, se zmenšil. Celkový počet anomálií zachycených touto metodou je 195. Se zmenšujícím se p se bude minimalizovat vliv anomálií na predikce, ale také značně vzroste časová náročnost. K nastavení parametru p můžeme využít předchozí metodu, která nám poskytne prvotní odhad počtu anomálií, pomocí kterého můžeme určit parametr p.

Podíváme se i na výsledky anomálií dataseu s1, kde detekování anomálií bylo paradoxně složitější než u datasetu s2, i když má mnohem pravidelnější chování. Problém nastává v místě, kde se rozptyl časové řady sníží od indexu 20 000, což můžete vidět na

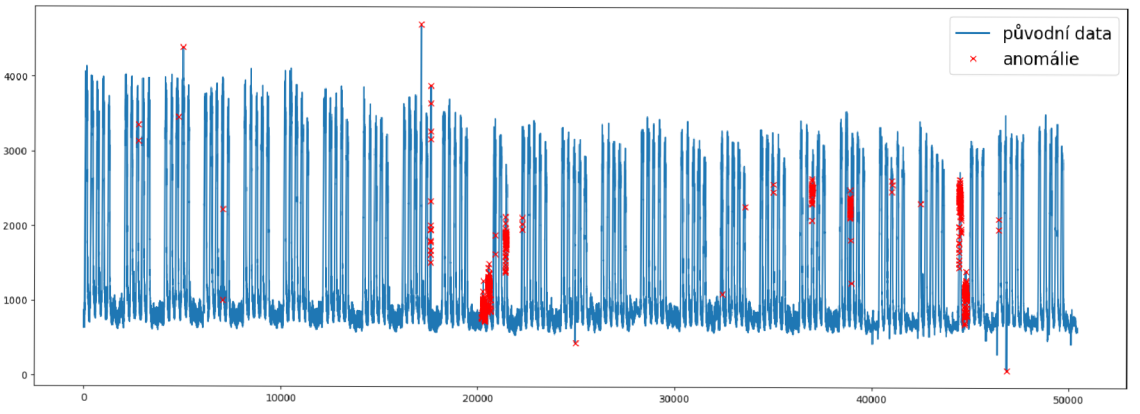


Obrázek 4.11: Anomálie datasetu s2 pomocí metody se změnou vstupů

obrázku 3.1. Náš trénovací dataset s1 pro detekci anomálií byl zvolen v úseku s větším rozptylem. Z toho důvodu neuronová síť předpokládá stejný rozptyl u celé časové řady a bude označovat za anomálie hodnoty s menším rozptylem. V tomto případě se nám podařilo tomuto chování zamezit díky vhodné volbě $\varkappa$. V opačném případě bychom tento problém řešili přetrénováním neuronové sítě v průběhu detekování. Další možností by bylo natrénování neuronové sítě na dvou vzorových datasetech s vyšším a nižším rozptylem. Výsledky obou metod u datasetu s1 můžeme vidět na obrázcích 4.12 a 4.13.

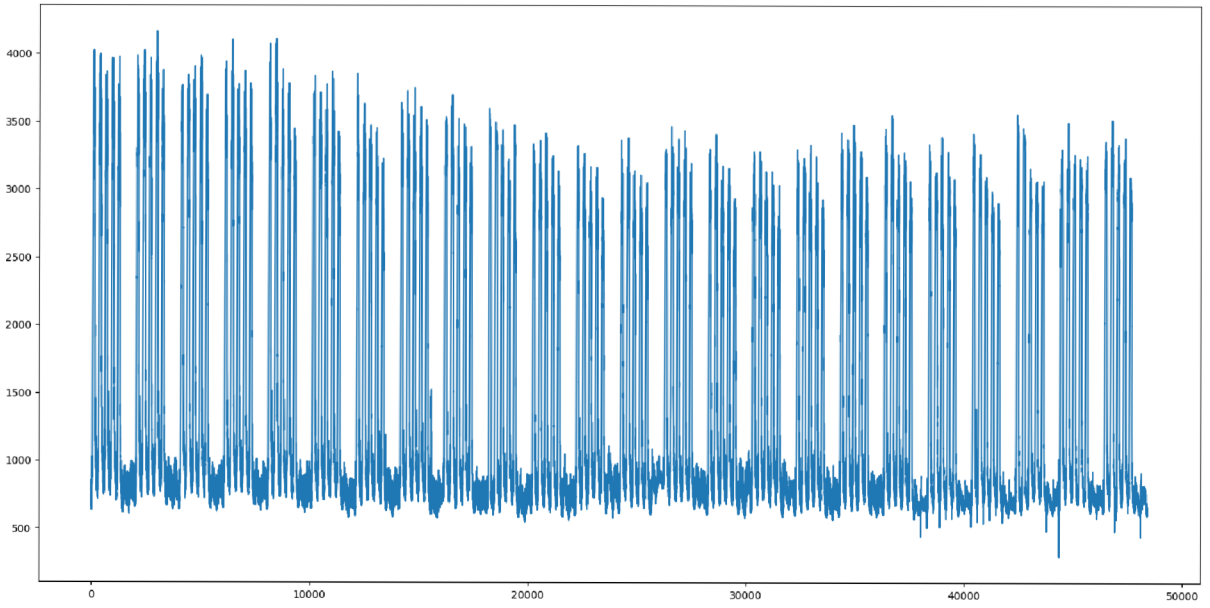


Obrázek 4.12: Detekce anomálií datasetu s1 pomocí metody Double-pass

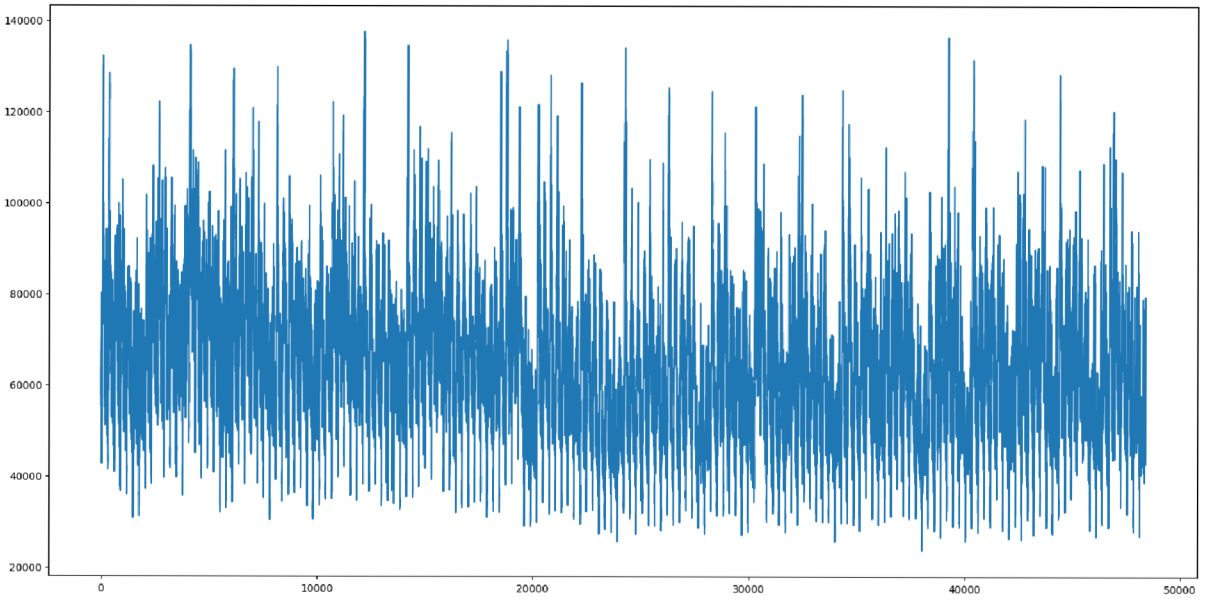


Obrázek 4.13: Detekce anomálií datasetu s1 pomocí metody se změnou vstupů

Výsledkem obou metod je nejen množina anomálií $A(X_t)$, ale také dataset $\widehat{X}_t$ vyčištěný od anomálií, který můžeme použít pro budoucí trénování nebo predikování. Výsledné vyfiltrované datasety, získané pomocí metody se změnou vstupů, můžeme vidět na obrázcích 4.14 a 4.15. U těchto datasetů jsme vynechali první týden dat, který je potřeba k první predikci a nemáme možnost ho pomocí neuronové sítě vyfiltrovat.



Obrázek 4.14: Vyfiltrovaný dataset s1



Obrázek 4.15: Vyfiltrovaný dataset s2

5 Regresní analýza časových řad

Výsledky vytvořené neuronovými sítěmi budeme porovnávat se statistickými metodami pro predikci časových řad. V následující sekci popíšeme teorii a základní vlastnosti potřebné pro predikci našich dat, které budeme uvažovat ve tvaru časové řady. Předpokládáme základní znalosti statistiky a budeme se hlavně věnovat problematice spojené s naším problémem. Následující teorie vychází z [14], [16], [15] a [18]

Definice 5.1. Časovou řadou $\{X_t\}$ (též stochastický proces) je množina náhodných veličin $\{X_t : t \in T\}$, $T \subset \mathbb{Z}$ definovaných na tomtéž pravděpodobnostním prostoru (Ω, A, P), $T \neq \emptyset$.

V našem případě uvažujeme množinu T jako podmnožinu celých čísel, jelikož náš dataset jsou diskrétní hodnoty v čase.

Definice 5.2. Nechť $\{X_t\}$ je časová řada s konečnými druhými momenty ($EX_t^2 < \infty$). Pro tento stochastický proces definujeme funkci středních hodnot

$$\mu_X(t) = EX_t$$

a kovarianční funkci

$$\gamma_X(r, s) = C(X_r, X_s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))],$$

kde r, s, $t \in \mathbb{N}$.

Důležitou vlastností pro analýzu časových řad je stacionarita. Udává do jisté míry stabilitu časové řady, tedy jestli je tato řada předvídatelná a její hodnoty nejsou závislé na čase.

Definice 5.3. Řekneme, že $\{X_t\}$ je stacionární (slabě stacionární), jestliže

- $\mu_X(t)$ nezávisí na t,
- $\gamma_X(t + h, t)$ nezávisí na t pro každé $h \in \mathbb{N}$.

Uvedeme příklady stacionárních procesů, které budou používány v následující části.

Definice 5.4. Jestliže v $\{X_t\}$ jsou všechny X_t stochasticky nezávislé a stejně rozdělené se střední hodnotou μ a rozptylem σ^2 značíme $X_t \sim IID(\mu, \sigma^2)$.

Definice 5.5. Jestliže v $\{X_t\}$ jsou všechny X_t nekorelované se stejnou střední hodnotou μ a rozptylem σ^2 značíme $X_t \sim WN(\mu, \sigma^2)$. Tento proces je označován jako bílý šum (White Noise). Přidáme-li podmínku, že $\{X_t\}$ je IID proces, dostaneme Gaussovský bílý šum $X_t \sim WN(0, \sigma^2)$.

Mezi další důležité vlastnosti časových řad patří autokorelační funkce. Určuje lineární závislost dvou prvků časové řady v různých časových okamžicích. V případě stacionárních řad jsou tyto funkce závislé pouze na vzdálenosti h daných časových okamžiků.

Definice 5.6. Nechť $\{X_t\}$ je stacionární časová řada, pak pro každé $h \in \mathbb{Z}$ definujeme

- autokovarianční funkci
$$\gamma_X(h) = C(X_{t+h}, X_t),$$

- autokorelační funkci ACF

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} = R(X_{t+h}, X_t).$$

Poznámka 5.7. Operátor zpětného posunu budeme značit B

$$BX_t = X_{t-1}.$$

Operátor diference na zpoždění 1 budeme značit ∇

$$\nabla X_t = X_t - X_{t-1} = (1 - B)X_t.$$

Opakované použití operátoru zavedeme následujícím způsobem

$$B^j X_t = B^{j-1} B X_t = X_{t-j}.$$

5.1 MA a AR procesy

V této části se budeme věnovat stacionárním procesům s nulovou střední hodnotou. MA proces popisuje časovou řadu, jejíž současná hodnota závisí na chybách predikce z předešlých časových hodnot.

Definice 5.8. Řekneme, že $\{X_t\}$ je MA (moving average) proces řádu q, jestliže platí

$$X_t = Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \cdots + \theta_q Z_{t-q}.$$

Tvar pomocí operátoru zpětného posunu

$$X_t = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q) Z_t = \left(1 + \sum_{i=1}^{q} \theta_i B^i\right) Z_t = \theta(B) Z_t,$$

$$\theta(B) = 1 + \theta_1 B + \cdots + \theta_q B^q,$$

kde $Z_t \sim WN(0, \sigma^2)$ a $\theta_1, \ldots, \theta_q \in \mathbb{R}$ jsou konstanty.

Na rozdíl od MA procesu, AR proces určuje současnou hodnotu časové řady pomocí předešlých hodnot.

Definice 5.9. Řekneme, že $\{X_t\}$ je AR (autoregression) proces řádu p, jestliže splňuje

$$X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + Z_t.$$

Tvar pomocí operátoru zpětného posunu

$$X_t = (\phi_1 B + \phi_2 B^2 + \cdots + \phi_p B^p) X_t + Z_t \Rightarrow Z_t = \left(1 - \sum_{i=1}^{p} \phi_i B^i\right) X_t = \phi(B) X_t,$$

$$\phi(B) = 1 - \phi_1 B - \cdots - \phi_p B^p,$$

kde $Z_t \sim WN(0, \sigma^2)$, $\phi_1, \ldots, \phi_p \in \mathbb{R}$ jsou konstanty a platí $C(X_r, Z_t) = 0$ pro $r < t$.

5.2 ARMA procesy

Spojením předešlých dvou procesů dostaneme obecnější proces nazývaný ARMA proces.

Definice 5.10. Řekneme, že $\{X_t\}$ je ARMA(p,q) proces $(p, q \geq 0)$, je-li stacionární a pro každé t platí

$$X_t - \phi_1 X_{t-1} - \cdots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \cdots + \theta_q Z_{t-q}.$$

Tvar pomocí operátoru zpětného posunu

$$\phi(B)X_t = \theta(B)Z_t,$$

kde $Z_t \sim WN(0, \sigma^2)$ a polynomy $\phi(B)$ a $\theta(B)$ nemají společné kořeny.

Pro získání lepších predikcí ARMA procesů využijeme následující vlastnost zvanou kauzalita, která popisuje vliv proměnné na budoucí hodnoty.

Věta 5.11. ARMA(p,q) je kauzální proces právě tehdy, když

$$\phi(z) \neq 0, \quad \forall |z| \leq 1,$$

tedy polynom $\phi(z)$ má kořeny mimo jednotkový kruh.

Následující věta zajišťuje, že řešení rovnice ARMA procesu existuje, a navíc je pouze jediné.

Věta 5.12. Stacionární řešení $\{X_t\}$ rovnice ARMA(p,q) existuje právě tehdy, když

$$\phi(z) \neq 0, \quad \forall |z| = 1,$$

tedy když kořeny polynomu $\phi(z)$ neleží na jednotkové kružnici. Toto řešení je navíc jediné.

V praxi všechny modely nesplňují podmínku kauzality. Pokud v časové řadě existuje z splňující podmínku $\phi(z) = 0$, potom se taková řada označuje jako nestacionární se stochastickým trendem nebo se v ní objevuje sezónnost či se v průběhu mění rozptyl řady, nemůžeme daný proces modelovat pomocí ARMA procesu. K tomu použijeme nestacionární modely stochastických procesů. K odstranění proměnného rozptylu využíváme Box-Coxovu transformaci stabilizující rozptyl.

5.2.1 Box-Coxova transformace

Nechť náhodná veličina X má $EX^2 < \infty$ a její směrodatná odchylka je funkcí střední hodnoty $\sigma(\mu) = \sigma_0 \mu^\theta$, $\sigma_0 > 0$. Položme $\lambda = 1 - \theta$, pak

$$g(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda}, & \lambda \neq 0 \\ \ln |x|, & \lambda = 0. \end{cases}$$

5.3 ARIMA procesy

Pro predikci nestacionárních časových řad se stochastickým trendem se využívají ARIMA procesy. Jedná se o procesy, které se pomocí opakovaného diferencování převedou na ARMA proces.

Definice 5.13. Nechť $d \in \mathbb{N}_0$, pak řekneme, že $\{X_t\}$ je ARIMA(p, d, q) proces, jestliže $Y_t = (1 - B)^d X_t$ je kauzální ARMA(p, q) proces. Tedy

$$\phi(B)(1 - B)^d X_t = \theta(B)Z_t, \quad Z_t \sim WN\left(0, \sigma^2\right),$$

kde $\phi(z)$ je polynom řádu p, $\theta(z)$ je polynom řádu q a $\phi(z) \neq 0$ pro $z \leq 1$.

Tedy pokud chápeme $\phi(B)(1 - B)$ jako polynom, potom p kořenů tohoto polynomu leží mimo jednotkový kruh a d kořenů leží na jednotkové kružnici. Na druhou stranu, pokud je kořen MA polynomu v ARMA modelu blízký 1, znamená to, že byla řada přediferencovaná.

5.4 SARIMA procesy

Předešlé modely byly schopné popsat pouze nesezónní data. Proto využijeme SARIMA model, který využívá periodičnost dat s.

Definice 5.14. Nechť d, $D \in \mathbb{N}_0$, pak řekneme, že $\{X_t\}$ je SARIMA $(p, d, q) \times (P, D, Q)_s$ proces s periodou s, jestliže $Y_t = (1-B)^d \left(1 - B^s\right)^D X_t$ je kauzální ARMA proces splňující rovnici

$$\phi(B)\Phi\left(B^s\right) Y_t = \theta(B)\Theta\left(B^s\right) Z_t, \quad Z_t \sim WN\left(0, \sigma^2\right),$$

kde $\phi(z)$ je polynom řádu p, $\Phi(z)$ je polynom řádu P, $\theta(z)$ je polynom řádu q, $\Theta(z)$ je polynom řádu Q. Pro splnění kauzality musí proces splňovat $\phi(z) \neq 0$ a $\Phi(z) \neq 0$ pro $z \leq 1$.

5.5 Predikce ve stacionárních časových řadách

Hledáme nejlepší lineární predikci $P_n X_{n+h}$, $h > 0$ ve stacionární řadě se známými parametry μ, $\gamma(h)$.

$$P_n X_{n+h} = a_0 + \sum_{i=1}^{n} a_i X_{n+1-i}.$$

Snažíme se tedy minimalizovat chybu $S(a_0, \ldots, a_n) = E(X_{n+h} - P_n X_{n+h})^2$. Celé odvození je popsáno v [14]. Minimalizací této hodnoty získáme výsledný tvar nejlepší predikce

$$P_n X_{n+h} = a_0 + \sum_{i=1}^{n} a_i X_{n+1-i} = \mu\left(1 - \sum_{i=1}^{n} a_i\right) + \sum_{i=1}^{n} a_i X_{n+1-i}, \quad \boldsymbol{\Gamma}_n \mathbf{a} = \boldsymbol{\gamma}_n(h),$$

kde

$$\boldsymbol{\gamma}_n(h) = \begin{pmatrix} \gamma(h) \\ \gamma(h+1) \\ \vdots \\ \gamma(h+n-1) \end{pmatrix}, \quad \boldsymbol{\Gamma}_n = \begin{pmatrix} \gamma(0) & \gamma(1) & \cdots & \gamma(n-1) \\ \gamma(1) & \gamma(0) & \cdots & \gamma(n-2) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma(n-1) & \gamma(n-2) & \cdots & \gamma(0) \end{pmatrix}.$$

5.5.1 Predikce v ARIMA modelu

Je-li X_t ARIMA(p, d, q), pak Y_t je kauzální ARMA(p, q), přičemž

$$(1 - B)^d X_t = Y_t$$

Proto

$$X_t = Y_t - \sum_{j=1}^{d} \binom{d}{j} (-1)^j X_{t-j}, \quad t = 1, 2, \ldots$$

Aplikací operátoru predikce dostáváme

$$P_n X_{n+h} = P_n Y_{n+h} - \sum_{j=1}^{d} \binom{d}{j} (-1)^j P_n X_{n+h-j}, \quad t = 1, 2, \ldots$$

kde $P_n Y_{n+h}$ získáme z predikce z ARMA modelu.

5.5.2 Predikce v SARIMA modelu

Je-li X_t SARIMA$(p, d, q) \times (P, D, Q)_s$, pak Y_t je kauzální ARMA(p, q), přičemž

$$(1 - B)^d (1 - B^s)^D X_t = Y_t$$

Proto

$$X_{n+h} = Y_{n+h} + \sum_{j=1}^{d+Ds} a_j X_{n+h-j}, \quad t = 1, 2, \ldots$$

Aplikací operátoru predikce dostáváme

$$P_n X_{n+h} = P_n Y_{n+h} + \sum_{j=1}^{d+Ds} a_j P_n X_{n+h-j}, \quad t = 1, 2, \ldots$$

kde $P_n Y_{n+h}$ získáme z predikce z ARMA modelu.

5.6 Dekompozice časové řady

Při detekci anomálií v časové řadě budeme využívat dekompozici časové řady. Tímto způsobem budeme schopni detekovat anomálie na reziduích časové řady. Mějme časovou řadu X_t, kterou můžeme rozložit následujícím způsobem

$$X_t = T_t + S_t + Y_t,$$

kde $EY_t = 0$, $S_{t+d} = S_t$ a platí $\sum_{j=1}^{d} S_j = 0$. T_t označuje trendovou složku časové řady, S_t označuje sezónní složku a Y_t představuje rezidua časové řady. Předpokládejme, že máme pozorování $\{x_1, ..., x_n\}$. Trend se nejprve odhadne použitím klouzavého průměru speciálně zvoleného tak, aby eliminoval sezónní složku a potlačil šum. Je-li perioda d sudá, řekněme $d = 2q$, pak použijeme

$$\widehat{T}_t = (0.5x_{t-q} + x_{t-q+1} + \cdots + x_{t+q-1} + 0.5x_{t+q})/2q, \quad q < t \leq n - q.$$

Je-li perioda lichá, například $d = 2q + 1$, pak použijeme jednoduchý klouzavý průměr

$$\widehat{T}_t = \frac{1}{2q+1} \sum_{j=-q}^{q} X_{t-j}, \quad q + 1 \leq t \leq n - q.$$

Druhým krokem je odhad sezónní složky. Pro každé $k = 1, \ldots, d$, vypočítáme průměr w_k odchylek $\{(X_{k+jd} - \widehat{T}_{k+jd}),\ q < k + jd \leq n - q\}$. Protože tyto průměrné odchylky se nemusí nutně rovnat nule, odhadujeme sezónní složku S_k jako

$$\widehat{S}_k = w_k - \frac{1}{d}\sum_{i=1}^{d} w_i, \quad k = 1, \ldots, d.$$

a $\widehat{S}_k = \widehat{S}_{k-d}$, pro $k > d$. Nesezónní data jsou pak definována jako původní řada s odstraněnou odhadnutou sezónní složkou

$$D_t = X_t - \widehat{S}_t, \quad t = 1, ..., n.$$

Nakonec znovu odhadneme trend z nesezónních dat pomocí klouzavého průměru. Z hlediska tohoto přehodnoceného trendu a odhadnuté sezónní složky je pak řada reziduí dána vztahem

$$Y_t = X_t - \widehat{T}_t - \widehat{S}_t, \quad t = 1, \ldots, n.$$

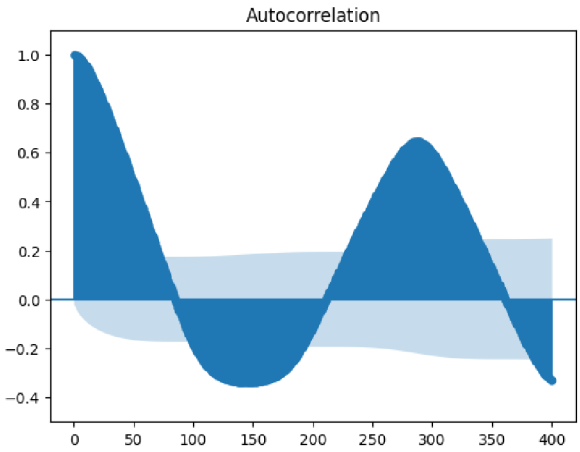
Dekompozici budeme využívat při detekci anomálií. Jelikož u dekompozice se předpokládá homoskedasticita, což znamená, že časová řada má v čase konstantní rozptyl, použijeme Box-Coxovu transformaci stabilizující rozptyl.

5.7 Porovnání regresní analýzy s umělou inteligencí

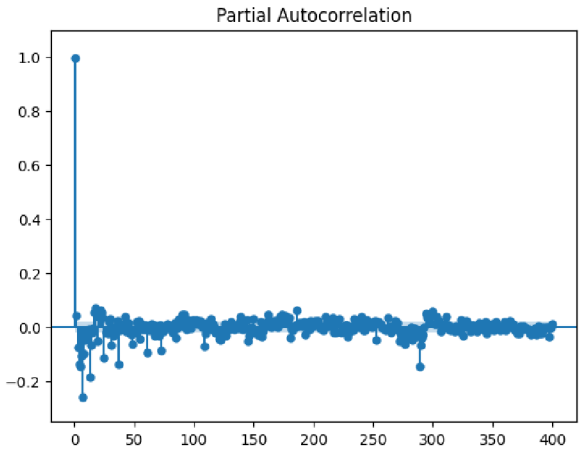
V této kapitole provedeme predikci dat pomocí statistických metod, pro pozdější porovnání s neuronovou sítí. Následně aplikujeme statistické metody pro detekci anomálií a výsledky srovnáme s našimi výsledky neuronové sítě. Regresní analýzu budeme provádět v jazyce Python s využitím knihoven Statsmodels, Scipy a Pmdarima.

Nejprve ověříme, zda jsou data stacionární. Vzhledem k našim datům nepředpokládáme, že naše časové řady budou stacionární. K testu využijme ADF (Augmented Dickey-Fuller) test a KPSS test. Hned v tomto případě dostáváme rozpor, jelikož ADF test zamítl nestacionaritu našich datasetů na hladině významnosti 0.05, na druhou stranu KPSS zamítl stacionaritu. Jelikož se v obou našich datasetech vyskytuje sezónnost, nebudeme naše data považovat za stacionární.

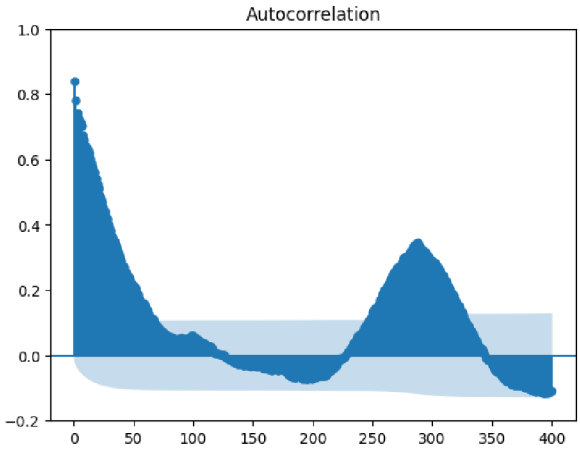
K tomuto závěru nasvědčují i autokorelační funkce a parciální autokorelační funkce datasetů. Z periodicity ACF vidíme jasnou sezónnost dat a pomalé konvergování k nule ACF zase naznačuje, že je zde přítomný trend.



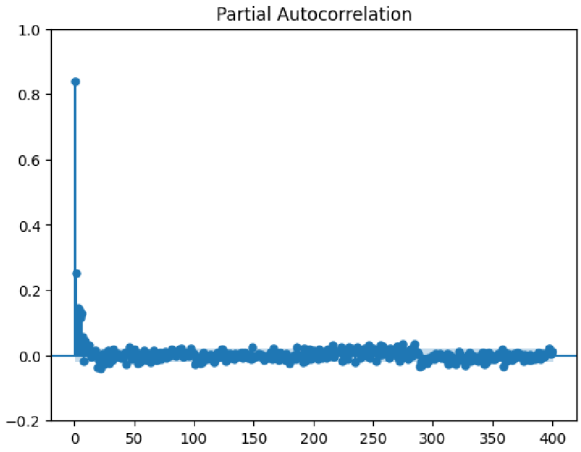
ACF datasetu s1



PACF datasetu s1



ACF datasetu s2



PACF datasetu s2

Následně zkontrolujeme, jestli oba naše datasety nemají proměnný rozptyl, což znamená, že rozptyl nezávisí na střední hodnotě. Tento aspekt budeme testovat pomocí parametru θ. Podmínkou je, že časová řada musí mít všechny hodnoty kladné. Naše datasety tuto podmínku splňují, nemusíme k řadám přičítat kladnou konstantu pro odstranění záporných hodnot. Následně vybereme k úseků délky 4-12 a odhadneme jejich μ a σ pomocí

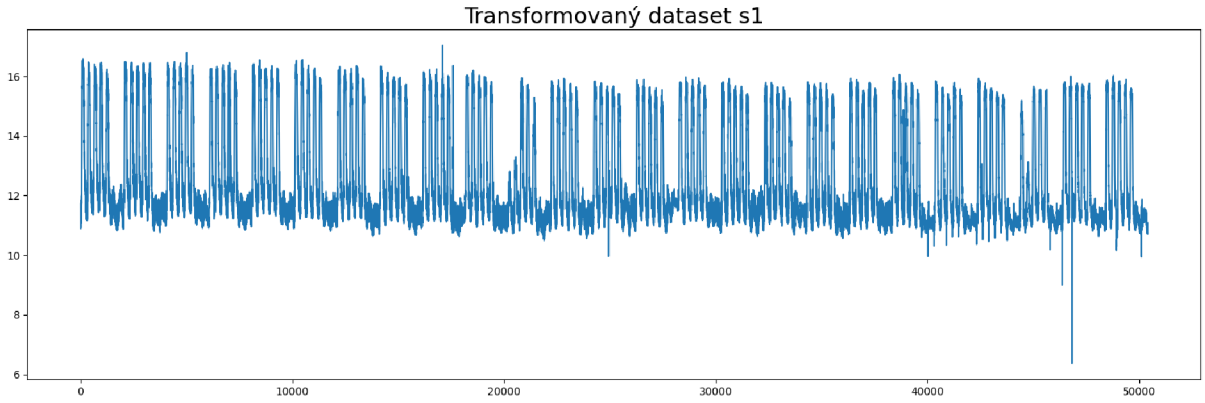
$\overline{x}_j$ a $r_j = \max x_j - \min x_j$, $j = 1, \ldots, k$. Parametr θ určuje závislost rozptylu na střední hodnotě následujícím způsobem

$$\sigma(\mu) = \sigma_0 \mu^\theta, \quad \ln \sigma(\mu) = \ln \sigma_0 + \theta \ln \mu.$$

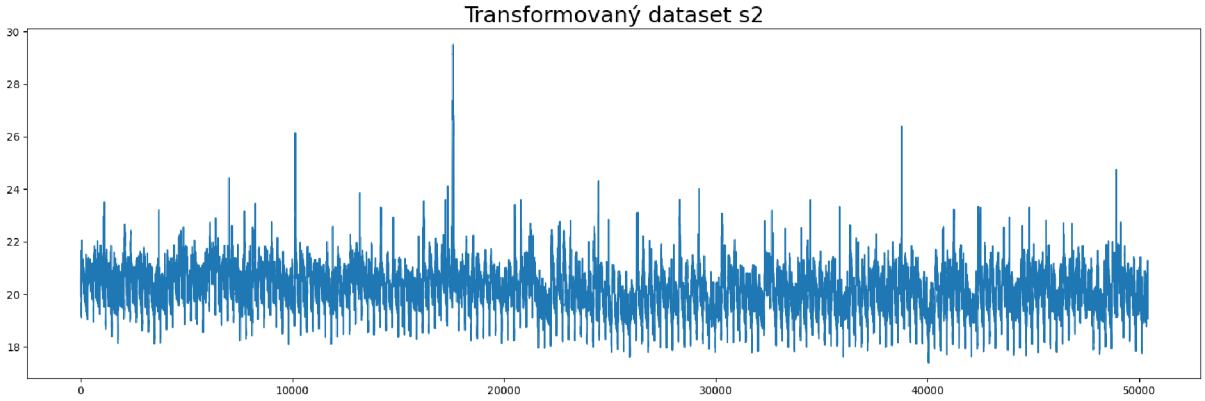
Následně tento parametr odhadneme pomocí regrese

$$\ln r_j = \ln \sigma_0 + \theta \ln \overline{x}_j.$$

Ke stanovení vhodné hodnoty θ využijeme interval spolehlivosti pro θ. Pokud bude interval spolehlivosti obsahovat 0, potom data mají konstantní rozptyl a nemusíme provádět transformaci stabilizující rozptyl. Jelikož oba naše datasety měly parametr θ větší než nula, provedli jsme Box-Coxovu transformaci stabilizující rozptyl.



Obrázek 5.3: Transformovaný dataset s1

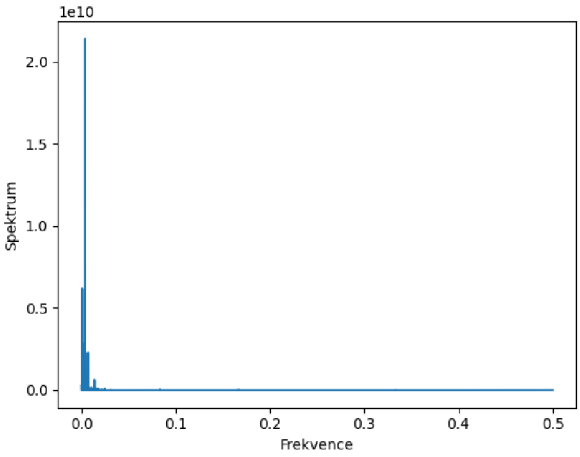


Obrázek 5.4: Transformovaný dataset s2

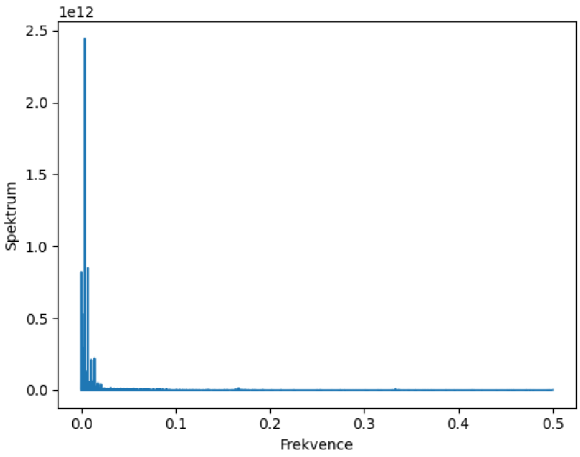
Tímto máme připravená data, která můžeme použít pro predikce. Určili jsme, že máme nestacionární data s trendem a sezónností, tím pádem k predikci využijeme SARIMA metodu popsanou v kapitole 5.4. Než začneme s predikcemi, stojí za to zmínit rozdíly v přípravě dat než u neuronové sítě. Vidíme, že jsme museli ověřit značné množství parametrů, transformovat data a vybrat vhodnou metodu pro predikce. U neuronové sítě stačilo pouze data normalizovat a připravit vstupní a validační data. Tento proces je plně automatický a nevyžaduje naši pozornost u rozdílných dat. Je tedy mnohem obecnější a nemusíme kontrolovat žádné požadavky, protože neuronová síť sama rozpozná vzory v chování dat. Oproti tomu ve statistické části musíme ověřit podmínky u každého nového datasetu a proces přípravy dat se může pokaždé lišit v závislosti na vstupních datech.

5.7.1 Predikce pomocí algoritmu SARIMA

Jelikož naše data obsahují sezónní složku, musíme určit periodu datasetů, abychom je mohli diferencovat, čímž eliminujeme sezónnost. Z chování dat můžeme soudit, že perioda obou datasetů bude 1 den, což v pětiminutových intervalech odpovídá 288 hodnotám. K potvrzení použijeme periodogram, který určí periodu datasetu. Zavedení periodogramu je popsáno v [14]. Periodogramy obou datasetů jsou zobrazeny na následujících obrázcích. Periodu datasetu získáme jako převrácenou hodnotu frekvence s nejvyšší hodnotou spektra. V obou případech vyšla perioda 288, což odpovídá jednomu dni.

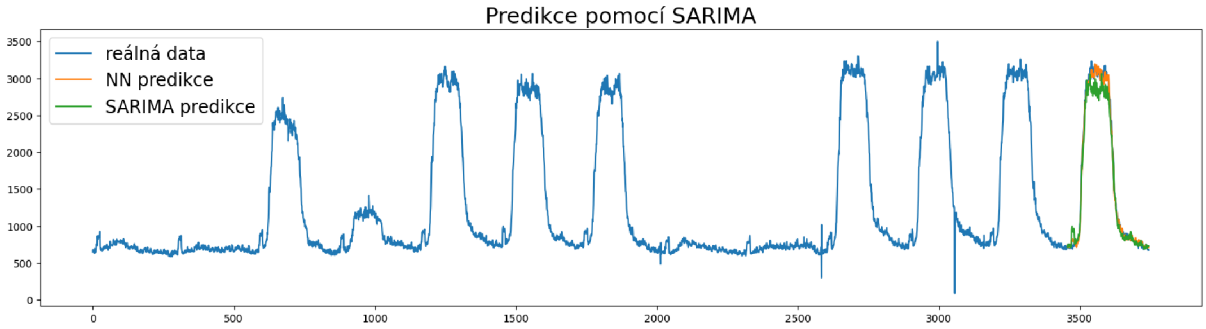


Periodogram datasetu s1



Periodogram datasetu s2

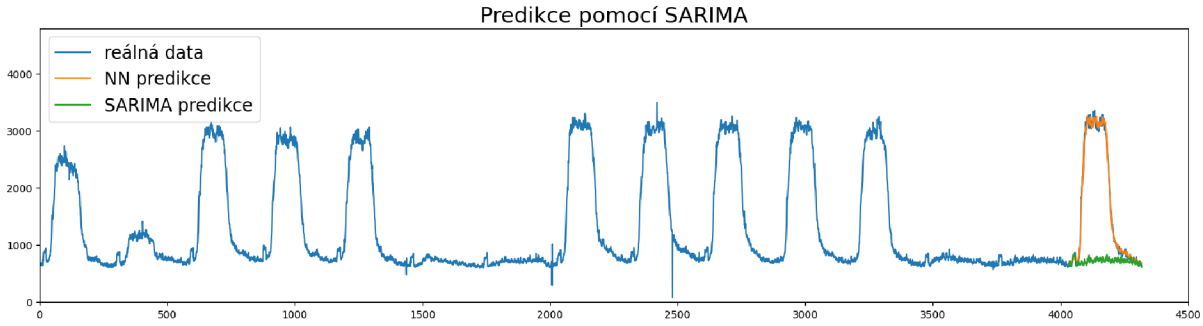
Následujícím krokem je diferencování časové řady pro zbavení se sezónnosti a následné určení parametrů SARIMA algoritmu. Jelikož se snažíme náš proces co nejvíce automatizovat, nebudeme zkoušet parametry nalézt osobně. Pro osobní určení parametrů se využívají ACF a PACF pro určení parametrů AR a MA procesu pro sezónní i nesezónní část časové řady. V našem případě necháme nalezení modelu na automatické detekci. Tento algoritmus počítá hodnotu AIC pro různé parametry modelu SARIMA a vybere ten s nejnižší hodnotou. Popis algoritmu můžete nalézt v [14]. Naše datasety jsou příliš velké pro využití tohoto algoritmu na celém datasetu, proto se omezíme pouze na poslední 3 týdny dat. První dva týdny využijeme pro určení modelu a poslední týden využijeme na otestování predikcí. Pro porovnání do obrázku přidáme i predikce vytvořené neuronovou sítí. Na obrázku 5.6 je zobrazena predikce datasetu s1.



Obrázek 5.6: Predikce datasetu s1 pomocí SARIMA algoritmu

Predikovali jsme dopředu 288 hodnot. Jako nejlepší model byl nalezen SARIMA(0,0,5)$(0,1,0)_{288}$. Tato metoda není stavěná na postupné predikování, jako to děláme v případě

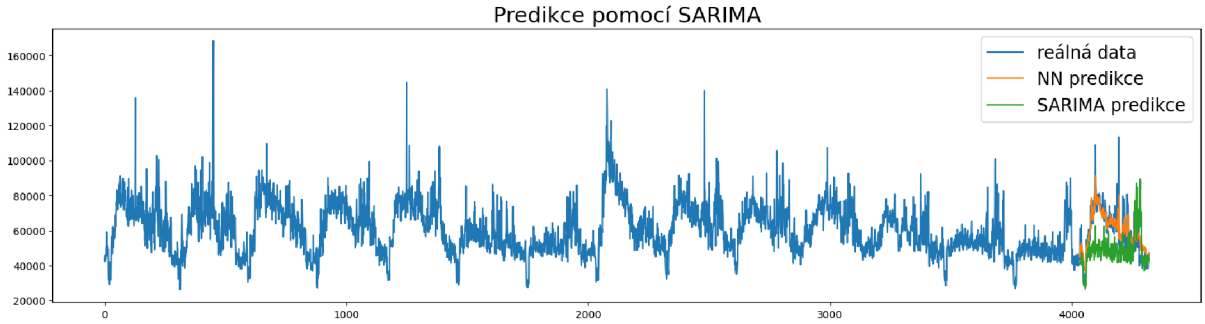
neuronové sítě, ale k predikci více hodnot najednou. Je potřeba zmínit, že nastavení parametrů u čtrnácti dní trvá kolem 20 minut a kdybychom to praktikovali s každou novou hodnotou, nedočkali bychom se výsledku. Můžeme použít model se stejnými parametry pro další predikce, ale vytvoření modelu se známými parametry pro určitá data trvá kolem 5 minut. V tomto má velkou výhodu neuronová síť, které predikce trvají vteřiny a stačí ji natrénovat pouze jednou. Predikce SARIMA algoritmu je méně přesná než u neuronové sítě. Navíc si můžeme všimnout, že predikce silně kopíruje předchozí data o délce jedné periody. Pokud chceme predikovat delší časový úsek, tak se predikce periodicky opakuje. Navíc zde nastává problém znázorněný na obrázku 5.7.



Obrázek 5.7: Predikce datasetu s1 pomocí SARIMA algoritmu

SARIMA predikce nebere v potaz týdenní periodičnost a pouze navazuje na nejbližší data. Možná je to způsobeno periodou pro diferencování řady, momentálně používáme periodu jeden den. Pokud zkusíme zvolit periodu dat jeden týden, jsou potřeba k určení parametrů modelu data s minimální délkou odpovídající dvěma periodám. Bohužel s tímto nastavením se model nikdy nedostal ke spočítání hodnoty AIC ani pro jednu sadu parametrů. Na naše množství dat není tento algoritmus svou výkonností stavěný. Bohužel se nám nepodařilo upravit nastavení modelu, aby bral v úvahu i týdenní periodicitu.

U datasetu s2 byl vyhodnocen jako nejlepší model SARIMA$(5,1,1)(0,1,0)_{288}$. Výsledky predikce můžeme vidět na obrázku 5.8. Model vykazuje podobné vlastnosti jako v předchozím případě. Věrně kopíruje poslední periodu, která neodpovídá reálným datům. V tomto případě dosahuje neuronová síť opět lepších výsledků. Na těchto příkladech můžeme vidět nesporné výhody neuronových sítí v umění najít vzory v chování dat a využít je v predikci. Navíc neuronová síť analyzovala celý dataset, což jsme si v případě SARIMA algoritmu nemohli z časových důvodů dovolit.



Obrázek 5.8: Predikce datasetu s2 pomocí SARIMA algoritmu

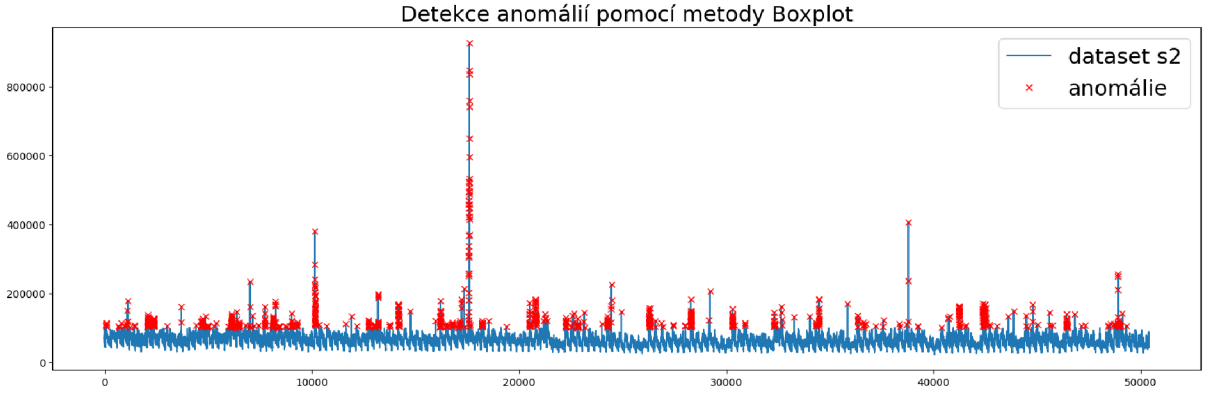
V tomto ohledu má umělá inteligence spoustu benefitů, jak v rychlosti a přesnosti, ale také ve své univerzálnosti. Ač máme dva rozdílné datasety, tak pro jejich predikování

jsme použili stejný model neuronové sítě. Pouze jsme natrénovali parametry, jejichž určení zabralo zlomek času oproti určení SARIMA modelu. Sice neuronová síť predikuje pouze jednu hodnotu do budoucnosti, přičemž SARIMA predikovala 288 hodnot, bohužel použít stejný přístup jako u neuronové sítě je z hlediska časové náročnosti u SARIMA algoritmu téměř nemožný.

5.7.2 Detekce anomálií

V této kapitole se pokusíme detekovat anomálie našich datasetů pomocí statistických metod a porovnáme složitost a postup řešení s neuronovou sítí.

Nejprve se zaměříme na dataset s2, který bude představovat menší problém na detekování anomálií. Je to způsobeno vysokými hodnotami anomálií a tím, že řada nemá rostoucí trend. Nejvýraznější anomálie vyskytující se v datech jsou hodnoty výrazně vzdálené od mediánu. Nabízí se nám použít metodu Boxplot, která označí hodnoty vzdálené o 1.5 kvartilové odchylky od mediánu za anomálie. Výsledek metody můžeme vidět na obrázku 5.9. Vidíme, že největší anomálie byly zachyceny, ale je označeno i značné množství bodů, které se nezdají býti anomáliemi. Tento přístup navíc nerozezná případy, kdy dojde ke změně chování, které se neprojeví vysokými nebo nízkými hodnotami, což neuronová síť zaznamená. Tento postup slouží spíše ke zpětné analýze dat, ale v případě, kdy potřebujeme s daty dále pracovat, nemáme anomálie čím nahradit, pouze víme, které body byly označeny za anomálie. Na druhou stranu se jedná o rychlou metodu, pro označení odlehlých anomálií, která nemá žádné předpoklady.



Obrázek 5.9: Anomálie datasetu s2 získané metodou Boxplot

Nyní se zaměříme na dataset s1, který bude na detekci obtížnější. Jelikož v datasetu s1 je minimum hodnot, které jsou odlehlé, nebudeme používat metodu Boxplot. Místo ní využijeme kritérium zvané Z-Score.

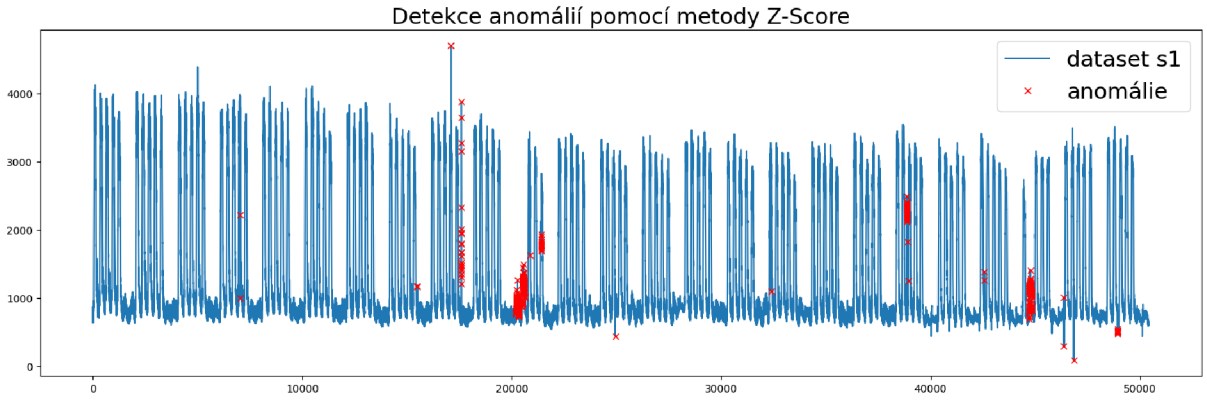
Z-Score

Předpokladem pro určení Z-Score je normální rozdělení datasetu X_t. Hodnotu z-score získáme pomocí vzorce

$$z = \frac{x - \mu}{\sigma}, \quad x \in X_t.$$

Za anomálii považujeme bod s hodnotou $|z| > 3$. Tato hodnota vychází z vlastnosti normálního rozdělení, kdy Z-Score má rozdělení $N(0, 1)$ a 99.7 % hodnot leží ve 3 směrodatných odchylkách od střední hodnoty. Jelikož předpokladem je normální rozdělení, použijeme na dataset Box-Coxovu transformaci, jejíž vedlejší vlastností je, že transformovaný dataset se přibližuje normálnímu rozdělení.

Použití metody Z-Score nemá vyhovující výsledky, když se použije přímo na transformovaný dataset. Nezaznamená totiž změny v chování. Využijeme dekompozici časové řady, protože dataset s1 je značně periodický. Tím oddělíme sezónní a trendovou složku a poté můžeme detekovat anomálie pomocí metody Z-Score přímo na reziduích, které by měli mít normální rozdělení. Tímto způsobem zachytíme jako anomálie nejen odlehlé hodnoty, ale i výrazné změny v chování. Výsledek můžeme vidět na obrázku 5.10. Jedná se znovu o rychlou metodu, ale musíme data nejdříve připravit. Přesnost detekce také souvisí s dekompozicí, protože ne vždy jsou výsledkem dekompozice normální residua, což snižuje přesnost metody.



Obrázek 5.10: Anomálie datasetu s1 detekované pomocí Z-Score

Z výsledků vidíme, že můžeme dosáhnout solidních výsledků při detekování anomálií i pomocí statistiky. Existuje i mnoho dalších metod na detekci anomálií, ale využili jsme ty, které vychází hlavně ze statistických poznatků. Nechtěli jsme využívat metody machine learningu, které jsou do určité míry velmi blízké našemu detekování pomocí neuronové sítě. Sice musíme určit tu nejvhodnější metodu pro detekci anomálií, ale podobným procesem si projdeme i u neuronové sítě. K natrénování neuronové sítě musíme vybrat vhodný úsek dat, což může být problematické samo o sobě, a následně tento dataset očistit od anomálií, což svým způsobem zahrnuje stejný proces jako v této kapitole. Na druhou stranu díky neuronové síti můžeme nahradit hodnoty anomálií pomocí predikcí, které zachovávají charakter dat. Oba postupy mají své výhody a nevýhody a pro nejlepší možný výstup bude vhodné zkombinovat oba přístupy.

6 Závěr

Cílem diplomové práce bylo aplikování neuronových sítí na reálná síťová data za účelem predikce hodnot. Dalším úkolem bylo využití vygenerovaných predikcí neuronovou sítí pro detekci anomálií a následné porovnání výsledků s regresní analýzou.

V první části jsme zavedli základní konstrukci neuronové sítě a následně ji rozšířili o LSTM architekturu. Druhá část obsahovala navržení a implementaci neuronové sítě v programovacím jazyce Python. Testovali jsme vliv míry učení, šířku vstupního vektoru a hloubku neuronové sítě. Dále jsme se zabývali vlivem velikosti vzorku na trénování neuronové sítě a otestovali jsme vliv přetrénování na výsledný model. Dosáhli jsme predikcí, jejichž chyba dosahovala přibližně 7 %.

Ve třetí části jsme upravili původní model pomocí vzorových dat, abychom dosáhli predikcí bez anomálií, které sloužily jako kritérium pro detekci anomálií. Následně jsme vytvořili dvě metody na detekci anomálií. První je metoda Double-pass, která využívá dvojí predikce pro zmírnění chyby způsobené anomáliemi s co nejmenší časovou náročností. Druhá je metoda se změnou vstupů, která v průběhu predikování nahrazuje anomální hodnoty ve vstupech a minimalizuje tak chybu predikcí, která se projeví v detekci anomálií.

Ve čtvrté části jsme uvedli definice potřebné pro regresní analýzu časových řad, kterou jsme potřebovali v závěrečném porovnání. Následně jsme provedli predikci dat statistickou metodou SARIMA, která slouží k predikci dat s trendem a sezónností. Bohužel tato metoda dosahovala mnohem horších výsledků než neuronová síť z hlediska přesnosti i časové náročnosti. Na druhou stranu statistické metody vykazovaly dobré výsledky při detekování anomálií, které byly srovnatelné s neuronovou sítí. V tomto ohledu má každý způsob svoje pozitiva i negativa a ideálním výsledkem je do detekce anomálií zapojit jak statistické metody, tak i neuronovou síť.

Tento proces nabízí širokou škálu možností, která by v budoucnu stála za otestování nebo vylepšení. Jako další postup se nabízí využití i jiných architektur neuronových sítí a jejich vzájemné porovnání. Možnost k vylepšení představuje vývoj adaptivních metod pro určování thresholdu $\varkappa$ pro přesnější detekci anomálií. Dalším aspektem vhodným k inovaci je výběr a filtrování testovacích dat, které jsou klíčové pro natrénování neuronové sítě sloužící k detekci anomálií. Ideálním scénářem by bylo navržení obecné metody, která by fungovala pro různé typy dat. Nabízí se i další množství témat, která by stála za uvážení, jelikož se jedná o stále poměrně mladý obor, který má vysoký potenciál pro zdokonalení.

7 Zdroje

[1] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE, 2016. Deep Learning [online]. MIT Press [cit. 2024-04-13]. Dostupné z: http://imlab.postech.ac.kr/dkim/class/csed514˙2019s/DeepLearningBook.pdf

[2] RAFF, E. Inside Deep Learning: Math, Algorithms, Models. Manning Publications, 2022. ISBN 9781617298639

[3] CABELLO, Julia, 2022. Mathematical Neural Networks [online]. [cit. 2024-04-13]. Dostupné z: https://www.mdpi.com/2075-1680/11/2/80

[4] KABBAY, Harcharan, 2022. Artificial Neural Network Concepts and Examples. Dostupné také z: https://irl.umsl.edu/cgi/viewcontent.cgi?article=1432&context=thesis. Thesis. University of Missouri-St. Louis.

[5] HAYKIN, Simon, 1999. Neural networks and Learning Machines [online]. Third edition. [cit. 2024-04-13]. ISBN 0-13-147139-2. Dostupné z: https://lps.ufrj.br/˜caloba/Livros/Haykin2009.pdf

[6] HOCHREITER, Sepp a Jürgen SCHMIDHUBER. Long short-term memory [online]. [cit. 2024-04-13]. Dostupné z: https://www.bioinf.jku.at/publications/older/2604.pdf

[7] ZHOU, Victor. Machine Learning for Beginners: An Introduction to Neural Networks [online]. [cit. 2024-04-13]. Dostupné z: https://victorzhou.com/blog/intro-to-neural-networks/

[8] Mathematics behind the Neural Network [online]. [cit. 2024-04-13]. Dostupné z: https://studymachinelearning.com/mathematics-behind-the-neural-network/

[9] HAMMER, Barbara, 2003. Mathematical Aspects of Neural Networks [online]. [cit. 2024-04-13]. Dostupné z: https://www.researchgate.net/profile/Thomas-Villmann/publication/221165237˙Mathematical˙Aspects˙of˙Neural˙Networks/links/0c96051fb93db28a94000000/Mathematical-Aspects-of-Neural-Networks.pdf

[10] PASCANU, Razvan, Tomas MIKOLOV a Yoshua BENGIO, 2013. On the difficulty of training Recurrent Neural Networks [online]. [cit. 2024-04-13]. Dostupné z: https://arxiv.org/pdf/1211.5063.pdf

[11] Understanding LSTM Networks [online]. [cit. 2024-04-13]. Dostupné z: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[12] BROWNLEE, Jason, 2020. Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras [online]. [cit. 2024-04-13]. Dostupné z: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

[13] KINGMA, Diederik P. a Jimmy Lei BA, 2015. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION [online]. [cit. 2024-04-17]. Dostupné z: https://arxiv.org/pdf/1412.6980.pdf

[14] BROCKWELL, P. J. and DAVIS, R. A. Introduction to time series and forecasting. Springer Cham, 2016. eBook ISBN 978-3-319-29854-2.

[15] HAMILTON, J. D. Time series analysis. Princeton University Press, 1994. ISBN 978-0-691-04289-3.

[16] HAHN, G. J. and SHAPIRO, S. S. Statistical models in engineering. John Wiley & Sons, 1994. ISBN 0-471-04065-7.

[17] KONG, W., DONG, Z. Y., JIA, Y., HILL, D. J., XU, Y. and ZHANG, Y. Short-term residential load forecasting based on LSTM recurrent neural network. IEEE Transactions on Smart Grid, 10(1), pp. 841-851, 2017.

[18] ANALYZE AND ECONOMIC TIME SERIES FORECASTING BY USING SELECTED STATISTICAL METHODS [online], 2019. Brno [cit. 2024-04-13]. Dostupné z: https://www.vut.cz/www˙base/zav˙prace˙soubor˙verejne.php?file˙id=193675. Master's thesis. VUT.

[19] ROMERO, Enrique, Josep M. SOPENA, Gorka NAVARRETE a René ALQUÉZAR, 2003. Feature Selection Forcing Overtraining May Help to Improve Performance [online]. [cit. 2024-05-17]. Dostupné z: https://ieeexplore.ieee.org/document/1223746