Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Detekce nestandardní spotřeby vody

Bakalářská práce

2021 Benjamin Ranš

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Benjamin Ranš**
Osobní číslo:	**I17132**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Detekce nestandardní spotřeby vody**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce bude návrh a implementace „chytrého" systému pro detekci nestandardní spotřeby vody v domácnosti. Student v teoretické části popíše nejčastěji používané metody pro detekci neobvyklých situací (detekce odlehlých datových bodů) a v praktické části na základě teoretických znalostí navrhne vhodný algoritmus (případně skupinu algoritmů), který implementuje ve zvoleném programovacím jazyce. Systém by měl na základě dat o spotřebě vody umět vytvořit model spotřeby vody pro libovolnou ubytovací jednotku a detekovat spotřebu vody, která se svou charakteristikou příliš liší od obvyklé spotřeby. Funkčnost systému student ověří na testovacích datech.

Rozsah pracovní zprávy: **min. 30**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

AGGARWAL, Charu C. *Outlier analysis*. New York: Springer, 2013. ISBN 978-1461463955.
HAWKINS, D. M. *Identification of Outliers* [online]. Dordrecht: Springer Netherlands, 1980 [cit. 2019-10-30]. DOI: 10.1007/978-94-015-3994-4. ISBN 978-94-015-3996-8.
KAZMIER, Leonard J. *Schaum's outline of theory and problems of business statistics*. 3rd ed. New York: McGraw-Hill, c1996. ISBN 0070340269.

Vedoucí bakalářské práce: **Ing. Jan Merta**
Katedra řízení procesů

Datum zadání bakalářské práce: **15. listopadu 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

Univerzita Pardubice Česká republika
L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 17. prosince 2019

Prohlašuji:

Práci s názvem Detekce nestandardní spotřeby vody jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 9. 5. 2021

Benjamin Ranš

PODĚKOVÁNÍ

Rád bych zde poděkoval především Ing. Janu Mertovi, který přátelsky, odborně, a hlavně trpělivě vedl zpracování této bakalářské práce. Díky jeho radám a doporučením jsem měl přístup k nejmodernějším metodikám strojového učení. Zároveň zde chci poděkovat své rodině, blízkým a přátelům za podporu a trpělivost při studiu i při vypracovávání této bakalářské práce.

ANOTACE

Cílem bakalářské práce bude návrh a implementace "chytrého" systému pro detekci nestandardní spotřeby vody v domácnosti. Student v teoretické části popíše nejčastěji používané metody pro detekci neobvyklých situací (detekce odlehlých datových bodů) a v praktické části na základě teoretických znalostí navrhne vhodný algoritmus (případně skupinu algoritmů), které implementuje ve zvoleném programovacím jazyce. Systém by měl na základě dat o spotřebě vody umět vytvořit model spotřeby vody pro libovolnou ubytovací jednotku a detekovat spotřebu vody, která se svou charakteristikou příliš liší od obvyklé spotřeby. Funkčnost systému student ověří na testovacích datech.

KLÍČOVÁ SLOVA

Strojové učení, detekce uniku vody, model spotřeby vody

TITLE

Detection of non-standard water consumption

ANNOTATION

The main objective of the bachelor thesis is going to be design and implementation of "smart" system for detection of non-standard usage of water in the household. During the theoretical part of the thesis, the student is going to describe the most commonly used methods for detection of unusual situations (data deviating from the model result) and during the practical part, the student is going to design suitable algorithm (or multiple algorithms) based on theoretical knowledge, which he is going to implement in chosen programming language. Based on water consumption data, the system should be able to create a model of water consumption for any accommodation unit and detect the usage of water, which differs in characteristics of usual water consumption. Based on collected test data, the student verifies system functionality.

KEYWORDS

Machine learning, water leakage detection, data modeling

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

LSTM	Long short-term memory
MAD	Mean absolute deviation
PDF	Portable document format
RNN	Recurrent neural network

ÚVOD

Chceme-li detekovat únik vody ve vodovodním potrubí, můžeme využít řadu přístupů. Použijeme-li například kouřový plyn můžeme zjistit, kde se ve zkoumaném potrubí nachází netěsnosti a patřičně na ně reagovat. Jelikož metoda hledání netěsností pomocí kouřového plynu je příliš invazivní k domácnosti a zároveň se toto řešení nedá považovat za stálé, je nutné přijít se způsobem, který bude šetrný a bude pro uživatele stálým řešením. Mezi dlouhodobá řešení lze považovat užití neuronových sítí, které v datech hledají neobvyklosti. Dalším aspektem je samozřejmě životnost vodoměru a zařízení, které digitalizuje data z vodoměru.

Cílem této bakalářské práce bude navrhnout algoritmus, pomocí kterého bude možné v datech detekovat anomálie. Vhodný algoritmus bude vybrán ze čtyř různých topologií, na kterých budou prováděny trénovací experimenty, které jsou blíže popsány v praktické části. Díky těmto experimentům budeme schopni vybrat nejvhodnější topologii pro detekci anomálií v datech o spotřebě vody v domácnosti.

První část bakalářské práce se skládá z teorie strojového učení a možnými přístupy k učení. Ke každému z přístupů je uveden jeho popis. Modely použité v této práci jsou primárně řízené učitelem, nebo samořízené.

Hluboké neuronové sítě slouží jako základ pro většinu algoritmů uvedených v praktické části. V textu je popsáno jejich fungování, tedy výpočet odezvy modelu na vstupní data. S výpočtem odezvy modelu je úzce spojený i výběr aktivační funkce. Správný výběr aktivační funkce může výrazně ovlivnit výpočet odezvy. Přesnost odezvy je v průběhu učení modelu zpřesňována díky algoritmu klesání podle gradientu, který k výpočtu směru k lokálnímu minimu využívá algoritmu zpětného šíření chyby.

Dalším typem neuronových sítí jsou rekurentní sítě. V této bakalářské práci je využito LSTM sítí. Hlavní výhodou rekurentních neuronových sítí je schopnost udržet závislosti mezi datovými body. V textu je obecně popsáno fungování těchto sítí a také jejich základní výhody a nevýhody. V další části jsou rozebrány možné přístupy k detekci neobvyklých stavů s tím, že v praktické části jsou využity pouze některé z nich. Využití dalších může být tématem příštích bakalářských či diplomových prací.

V praktické části je zdokumentovaná implementace přípravy dat pro natrénování modelů a způsob detekce anomálií pomocí auto-enkodérů a regresních modelů. Příloze je přidán soubor, kde je detekce anomálií implementována v jazyce Python.

1 STROJOVÉ UČENÍ

Strojové učení je věda zabývající se skupinou algoritmů, které jsou na základě zkušenosti schopny se učit. Často zkušeností myslíme vstupní data, která algoritmus sleduje za účelem nalezení pravidel pro nová data z množiny, která definují nějaké chování. Důležitým předpokladem je, že algoritmus je schopen pravidla z množiny dat a odpovědí najít sám a odhadnout tak chování budoucích dat. Nevýhodou strojového učení je, že data, která model zpracovává, musí být předem připravena. Hledáme v datech vlastnosti (příznaky), které je nejlépe popisují a potom sestavíme model, který je na těchto datech trénován. V klasickém programování vycházíme z předpokladu, že na základě vstupních dat a předem pevně definovaných pravidel dostaneme očekávaný výsledek. Problém však nastane ve chvíli, kdy máme širokou škálu pravidel definujících chování vstupních dat. Není v lidských silách pojmout všechna možná pravidla. Příkladem může být rozpoznání ručně napsaných číslic, písmen nebo rozpoznávání obličejů, kdy je skoro nemožné definovat sadu pravidel, podle které například poznáme obličej, písmeno nebo číslici. V této oblasti jsou algoritmy strojového učení výhodné, jelikož pravidla na základě už definovaných odpovědí se umí naučit samy ([5], [7]).

Obrázek č. 1 - Strojové učení, zdroj: ([5], s. 22).

1.1 Učení s učitelem

Algoritmus strojového učení s učitelem funguje na principu učení se ze zkušenosti. Cílem modelu je z množiny vstupních dat vyprodukovat výsledek, který je následně porovnán s očekávaným výstupem pro daný vstup. Na základě chyby, kterou získáme rozdílem mezi očekávaným výstupem a odezvou modelu na vstup, hledáme optimální nastavení parametrů modelu za účelem nalezení co nejpřesnější aproximace cílové funkce ([5], [7]). Algoritmus hledání optimálního nastavení parametrů se jmenuje sestup podle gradientu. Nejprve však musíme vypočítat vliv jednotlivých parametrů na výslednou chybu pomocí algoritmu zpětného šíření chyby. Proces učení opakujeme do chvíle, než se model naučí mapovat vstup na výstup. Množinu dat je nutné rozdělit na nejméně dvě podmnožiny. Jedna z množin slouží k natrénování modelu a

druhá množina slouží k testování přesnosti modelu. Trénovací množina bývá větší než testovací množina, protože pro aproximaci cílové funkce je potřeba větší množství dat. Testovací množinou zjišťujeme, jak je model přesný na nových datech neboli jak dobře umí generalizovat. Nevýhodou učení s učitelem je, že data musí být člověkem předem označena. Model se pak na těchto datech učí mapovat vstupy na výstupy.

1.2 Učení bez učitele

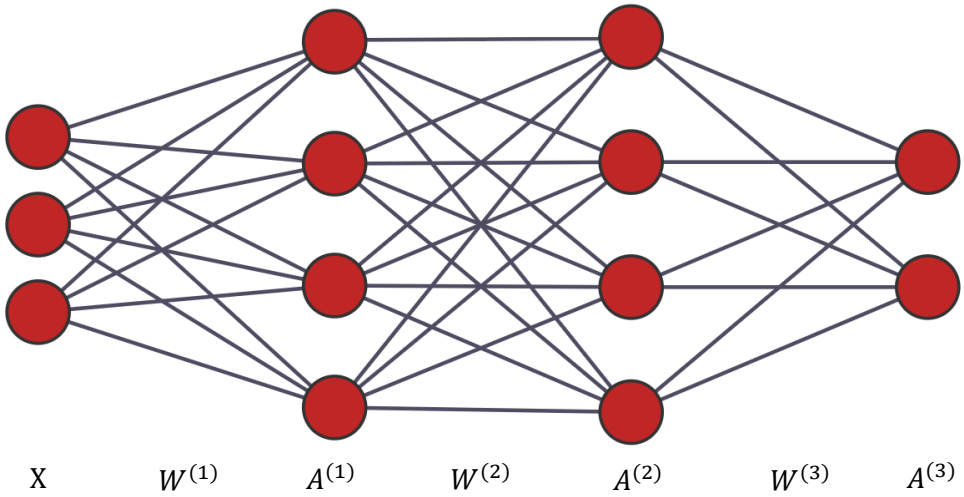
Model strojového učení bez učitele hledá ve vstupních datech zajímavé vzory bez zásahu učitele ([7], s. 9). Vstupní data nejsou označená, což představuje značnou výhodu nad přístupem učení s učitelem, kde data musí člověk předem ručně označit. Model strojového učení bez učitele například slouží k detekci odlehlých bodů tím, že pomocí shlukování data automaticky rozdělí do několika podmnožin. Pokud nová data nespadají do jedné z těchto množin nebo od jejich středu jsou až příliš vzdálená, model je vyhodnotí jako nežádoucí.

1.3 Samořízené učení

Model se učí ze zkušenosti a v průběhu trénování zvyšuje svoji přesnost. Rozdíl mezi samořízeným učením a učením s učitelem je, že data nutně nemusí být označená člověkem. Hezkým příkladem samořízeného učení jsou auto-enkodéry ([1], [6]). Model se skládá ze dvou částí. První část, kodér, vstupní data redukuje do podoby, která je obvykle menší než původní. Druhá část, dekodér, výstupní data z kodéru rekonstruuje do původní podoby. Výslednou chybou je rozdíl mezi původním a zrekonstruovaným vstupem. Další proces model upraví podle vzniklé chyby. Cílem je najít redukovanou reprezentaci původních dat, ze které jsme schopni původní data znovu vytvořit.

2 NEURONOVÉ SÍTĚ

Základními komponentami neuronových sítí jsou neurony a cesty mezi nimi. Ke každé cestě se váže váha neboli hodnota, která určuje vliv aktivace neuronu v předchozí vrstvě na neuron ve vrstvě následující. Neuron na vstupu přijímá signál. Ve vstupní vrstvě se jedná o vstupní data, která vstupní neurony nijak neupravují. V ostatních vrstvách se jedná o výstup z agregační funkce, která sečte hodnoty aktivačních funkcí v předchozí vrstvě vynásobené s vahami, které vedou k neuronu ve vrstvě následující podle vztahu (1) a následně na hodnotu z agregační funkce aplikuje aktivační funkci. Ve výstupní vrstvě se hodnota aktivace neuronu porovnává s očekávaným výstupem a je vypočtena chyba, pomocí které se pak model učí. Hlavním cílem neuronových sítí je naučit se mapovat vstupy na výstupy hledáním pravidel a správně tak klasifikovat, aproximovat nebo predikovat výstupy na nových vstupních datech.



Obrázek č. 2 – Neuronová síť, zdroj: autor

Neuronová síť se třemi vrstvami, kde X je vektor vstupních dat, $W^{(l)}$ označuje tenzor druhého řádu l-té vrstvy. Vektor $A^{(l)}$ označuje hodnoty aktivací v l-té vrstvě.

2.1 Výběr komplexnosti modelu

Při tvorbě modelu je důležité rozmyslet, kolik parametrů bude náš model mít. Parametry modelu jsou myšleny váhy a prahy. Počet parametrů je ovlivněn počtem neuronu v jednotlivých vrstvách a počtem skrytých vrstev. Obecně platí, že kapacita modelu je dána jeho velikostí. Správný výběr velikosti modelu může ovlivnit, jestli náš model dokáže dostatečně správně ge-

neralizovat na nových datech. Pokud vytvoříme model, který bude příliš velký a naše data nebudou tolik komplexní, model se na datech přeučí a jeho výkonnost na testovacích datech bude malá oproti výkonosti na trénovacích datech. Pokud však vytvoříme model, který bude mít příliš malé rozměry, nebudeme schopni najít dobrou reprezentaci jak pro nová, tak i pro trénovací data. Model bude mít příliš malou kapacitu. Optimální cestou je najít takový model, jehož kapacita bude co nejpřesněji odpovídat komplexnosti dat a zároveň bude umět dobře generalizovat na nových datech ([6], s. 110).

2.2 Trénování neuronové sítě

Trénování modelu je proces hledání optimálního nastavení parametrů neuronové sítě. Zároveň očekáváme, že model s optimálně nastavenými parametry bude umět generalizovat na nových datech ([1], [6], [7]). Prvním krokem trénování je vytvoření modelu neuronové sítě (volba počtu parametrů, neuronů a vrstev), výběr aktivačních funkcí v závislosti na umístění (skrytá nebo výstupní vrstva), nastavení koeficientu rychlosti učení a výběr chybové funkce. Druhým krokem trénování modelu je hledání optimálních hodnot parametrů neuronové sítě pomocí algoritmu klesání podle gradientu, kdy rychlost klesání určuje zvolený koeficient rychlosti učení. Optimalizace probíhá tak, že vypočítáváme průměrnou chybu na vzorku dat (postupně tímto způsobem několikrát projdeme celou trénovací množinu dat) a na základě této chyby pak počítáme gradient v bodech aktuálního nastavení jednotlivých parametrů pomocí algoritmu zpětného šíření chyby. Vypočtený gradient pak použijeme pro aktualizaci jednotlivých parametrů tak, že k aktuálním hodnotám parametrů přičteme záporný gradient vynásobený koeficientem rychlosti učení. Třetím krokem je testování modelu, které spočívá v tom, že na vstup natrénovaného modelu postupně přivedeme všechna data z testovací množiny a v každém kroku vypočítáme chybu odezvy modelu. Z výsledné hodnoty vypočítáme průměrnou chybu, kterou porovnáme s průměrnou chybou modelu na trénovacích datech ([1], [8], [10]). Pátým krokem je rozhodnutí, zdali je náš model na testovacích datech dostatečně přesný, což pro nás znamená úspěšně zvolený model nebo pokud zvolený model nedostatečně generalizuje na testovacích datech, musíme se vrátit zpět na první krok a zvolit jiný počet parametrů, vrstev a neuronů a proces trénování neuronové sítě opakovat.

2.3 Agregační funkce

Každý neuron, vyjma vstupních neuronů, má přidruženou váženou agregační funkci (1), jejíž účel je aktivační hodnoty vynásobené vahami vedoucími z neuronů v předchozích vrstvách sečíst. K výsledku je následně přičtena hodnota prahu neuronu, která je pro vstup do jednotlivých neuronů konstantní, mění se pouze v rámci učení neuronové sítě. Výstup z této funkce je pak předán aktivační funkci neuronu, jejíž výstup je pak propagován dále do modelu – buď ztrátové funkci nebo společně s přidruženou váhou agregační funkci v další vrstvě.

$$Z_j^{(l)} = \sum_{i=0}^{n} w_{ji}^{(l)} a_i^{(l-1)} \tag{1}$$

kde:

$Z_j^{(l)}$ – hodnota agregační funkce použitá v aktivační funkci j-tého neuronu v l-té vrstvě,

n – počet neuronů v předchozí vrstvě,

$w_{ji}^{(l)}$ – hodnota váhy v l-té vrstvě vedoucí z i-tého neuronu do j-tého neuronu; v případě, že $i = 0$ se jedná o práh neuronu,

$a_i^{(l-1)}$ – hodnota aktivace i-tého neuronu v předchozí vrstvě; v případě, že $i = 0$ je hodnota aktivace rovna jedné.

Agregační funkci lze také zobecnit pro jednotlivé vrstvy podle vztahu (2) (v kontrastu se vztahem (1), který počítá agregační funkci pro jednotlivé neurony zvlášť). Výstup z této funkce je vektor, který je v aktivační funkci neuronu modulován a propagován do dalších vrstev.

$$Z^{(l)} = W^{(l)} A^{(l-1)} + B^{(l)} \tag{2}$$

kde:

$W^{(l)}$ – tenzor (druhého řádu) vah v l-té vrstvě,

$A^{(l-1)}$ – vektor aktivací jednotlivých neuronů v předchozí vrstvě,

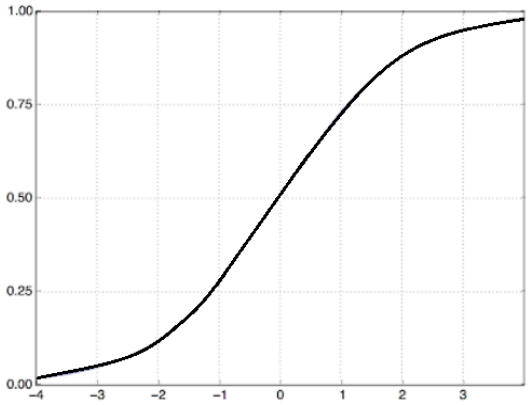
$B^{(l)}$ – vektor hodnot prahů l-té vrstvy.

Práh neuronu slouží k nastavení citlivosti aktivační funkce na vstupy z předchozí vrstvy. Je to jeden z druhů parametrů, které jsou v procesu učení optimalizovány. Pokud je hodnota prahu záporná, snižujeme citlivost neuronu na vstup. Když je hodnota prahu kladná, citlivost neuronu na vstup je vyšší. Jedná se o umělý vstup, který ovlivňuje celkovou odezvu sítě ([10], s. 17).

2.4 Aktivační funkce

Reakci neuronu na vstup definuje aktivační funkce. Přivedením signálu z agregační funkce (1) na vstup aktivační funkce neuronu se v závislosti na výběru aktivační funkce moduluje hodnota vstupu aktivační funkce do intervalu $(0, 1)$ pro sigmoid, $(-1,\ 1)$ pro tanh, $\langle 0, 1\rangle$ pro skokovou funci nebo $\langle 0, \infty)$ pro ReLU. Hodnota aktivace se předává jako vstup agregačním funkcím neuronů v dalších vrstvách do chvíle, kdy signál dorazí do výstupní vrstvy. Signál je cestou agregován s přidruženými váhami a modulován aktivačními funkcemi.

Existují dva typy aktivačních funkcí. Lineární a nelineární. Lineární funkce je polynomická funkce prvního stupně. Její hodnota buď konstantně roste, anebo klesá. Je definovaná po celém definičním oboru a na celém oboru hodnot. Derivace lineární funkce je konstanta. Hlavní nevýhodou lineárních aktivačních funkcí je, že jejich kombinací získáme pouze lineární funkci, což znamená, že nejsme schopni aproximovat nelineární funkce. Nelineární aktivační funkce je každá funkce, která není lineární. Díky nelinearitě jsme schopni reprezentovat i komplexní systémy.



Obrázek č. 3 – Sigmoid, zdroj: ([5], s. 74)

Aktivační funkci vybíráme podle několika kritérií. Jedno z kritérií je, v jaké vrstvě se aktivační funkce nachází. Pro výstupní vrstvu musíme rozhodnout, jestli neuronovou síť budeme využívat k regresi nebo klasifikaci. U regresních modelů se aktivační funkce ve výstupní vrstvě nevyužívá. Zastupuje jí agregační funkce, která je lineární a její obor hodnot je v intervalu $(-\infty,\ \infty)$, čehož lze v regresi využít. Pro klasifikační problémy používáme několik funkcí. Například funkce $softmax$ slouží pro klasifikaci do mnoha tříd. Funkce $sigmoid$ (Obrázek č. 3) slouží pro určení pravděpodobnosti výskytu nějakého jevu. Funkce $tanh$ je vhodná pro klasifikaci do dvou tříd. Obecně platí, že aktivační funkce musí být diferencovatelná (v každém

jejím bodě existuje derivace) a zároveň je doporučeno vybírat takové aktivační funkce, jejichž derivace nejsou náročné na výpočet.

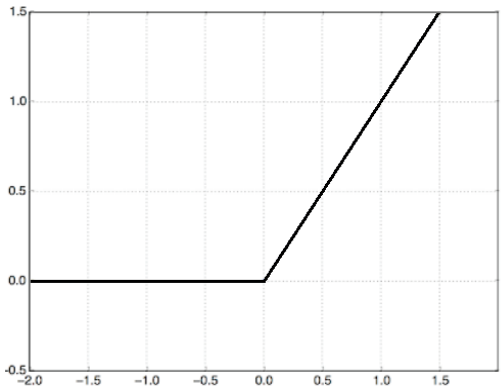
$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

Pro skryté vrstvy můžeme použít libovolnou, výše zmíněnou funkci, avšak jednou z nejvyužívanějších aktivačních funkcí ve skrytých vrstvách je ReLU (Obrázek č. 4). Vychází z lineární funkce, avšak výsledek funkce pro záporné hodnoty je nulový. Nevýhodou je, že neumí reagovat na záporné vstupy a není omezená shora, což z ReLU (4) dělá funkci nevhodnou pro neurony ve výstupní vrstvě. Výhodou je rychlost výpočtu její reakce na vstup. Dále neguje výskyt mizení gradientu tím, že není omezená shora. Jedná se o jev, kdy se parametry ve vrstvách (blíže ke vstupní vrstvě) aktualizují o mizivé hodnoty.

$$ReLU(x) = \max(0,\ x) \tag{4}$$

kde:

$max(0, x)$ – vrací maximum z hodnot 0 a x.



Obrázek č. 4 – ReLU, zdroj: ([5], s. 74).

2.5 Odezva neuronové sítě

Neuronovou síť lze reprezentovat složenou funkcí, kde každá funkce reprezentuje jednu vrstvu v neuronové síti. Vstupní vrstva se v kontextu neuronových sítí nepočítá do celkového počtu vrstev, protože na svůj vstup neaplikuje aktivační funkci. První skrytá vrstva daná funkcí (6), pracuje se vstupními daty, které je vhodné předem zpracovat. Na základě empirických rozhodnutí je vhodné vstupní data transformovat do intervalu <-1;1> a rozdělit na dvě oddělené podmnožiny, jednu trénovací a druhou na testování výkonnosti modelu ([10], s. 30). Vnější

funkce $f^{(2)}$ a $f^{(3)}$ pracují s výsledky z předchozích vrstev. Při procesu učení je výstup z modelu ve ztrátové funkci porovnán s očekávaným výstupem za účelem optimalizace parametrů modelu pomocí gradientu chybové funkce.

$$f(x;\theta) = f^{(3)}(f^{(2)}(f^{(1)}(x,\theta^{(1)}),\theta^{(2)}),\ \theta^{(3)}) \tag{5}$$

kde:

$$f^{(l)}(x;\ \theta^{(l)}) = A^{(l)} = \ \phi(Z^{(l)}); \tag{6}$$

$A^{(l)}$ – vektor aktivací jednotlivých neuronů ve vrstvě l,

$Z^{(l)}$ – vektor hodnot agregačních funkcí v l-té vrstvě vycházející ze vztahu (2),

θ – tenzor všech parametrů modelu,

$\theta^{(l)}$ – parametry l-té vrstvy.

K výpočtu odezvy sítě je výhodné používat tenzory prvního (např. (8)) a druhého řádu (např. (7)), kde řád reprezentuje dimenzi tenzoru.

$$W^{(l)} = \begin{bmatrix} w_{01} & \cdots & w_{0i} \\ \vdots & \ddots & \vdots \\ w_{j1} & \cdots & w_{ji} \end{bmatrix} \tag{7}$$

kde:

w_{ji} – váha vedoucí do cílového neuronu j ze zdrojového neuronu i.

$$A^{(l)} = \begin{bmatrix} a_0 \\ \vdots \\ a_i \end{bmatrix} \tag{8}$$

kde:

a_i – reprezentuje výstup z i-tého neuronu.

Jedním z důvodů, proč neuronové sítě používají tenzory je, že jsme schopni výpočty paralelizovat. S oblibou se tak využívají grafické karty Nvidia, které obsahují mnoho CUDA jader, se kterými umí pracovat knihovny pro neuronové sítě. Každé z těchto jader má na starosti jednu část výpočtu jak při dopředném, tak při zpětném průchodu neuronovou sítí.

2.6 Chybová funkce

Jedno z nejdůležitějších rozhodnutí je výběr chybové funkce, která je jedním ze základních stavebních kamenů procesu učení modelu neuronové sítě. V této bakalářské práci bude použita střední kvadratická chyba (vztah (10), kde ztrátu na trénovací vstup vypočteme podle vztahu (9)). Chybová funkce měří, jak moc se odezva sítě liší od očekávané odezvy na vstup a podle toho „penalizuje“ neuronovou síť. Z pohledu výpočetní náročnosti je výhodné měřit průměrnou chybu na určitém vzorku modelem zpracovaných dat s tím, že v procesu učení průběžně projdeme všechna trénovací i testovací data. Velikost vzorku může výrazně ovlivnit konvergenci modelu k optimálnímu nastavení parametrů a ideálně nalezení globálního minima chybové funkce. Zvolíme-li velikost vzorku příliš malou, parametry modelu budeme aktualizovat velmi často, což proces učení zpomaluje, protože pokaždé musíme znovu vypočítat gradient v aktuálním nastavení parametrů modelu. Když velikost vzorku bude rovna velikosti trénovací množiny, proces učení bude také pomalý, protože po každé aktualizaci parametrů modelu budeme muset znovu projít všechna trénovací data, což v závislosti na množství dat zpomalí konvergenci k optimálnímu nastavení parametrů modelu. Optimální velikost vzorku dat vybíráme jak proporcionálně k velikosti trénovací množiny, tak ze zkušenosti, kdy musíme metodou pokus omyl zjistit, jaká je optimální velikost vzorku pro výpočet průměrné chyby. Podle vztahu (9) vypočítáme chybu pro i-tý trénovací vstup a podle vztahu (10) vypočítáme průměrnou chybu na n datech, kde n je menší nebo rovno velikosti procházené množiny dat.

$$L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2 \tag{9}$$

kde:

$\hat{y}_i$ – odezva modelu na i-tý prvek z trénovací množiny,

y_i – Předpokládaná odezva modelu na i-tý prvek.

$$J(X;\theta) = \frac{1}{n}\sum_{i=1}^{n} L(f(x_i;\theta), y_i) \tag{10}$$

kde:

J – chybová funkce,

X – tenzor vstupních dat,

θ – parametry modelu,

n – velikost vzorku dat,

$L(f(x_i;\theta),\hat{y}_i)$ – vrací hodnotu chyby pro i-tý trénovací příklad,

$f(x_i;\theta)$ – odezva modelu na vstupní vektor dat x_i na základě parametrů θ modelu neuronové sítě,

y_i – očekávaná hodnota výstupu modelu.

2.7 Zpětné šíření chyby

Algoritmus zpětného šíření chyby je založen na výpočtu gradientu ztrátové funkce podle parametrů modelu neuronové sítě, tedy všech vah a prahů. Jednotlivé položky vektoru gradientu jsou tvořeny parciálními derivacemi chybové funkce podle všech parametrů modelu ([6], s. 82). Předpokladem je, že funkce, ze které počítáme gradient, je diferencovatelná, tedy v každém jejím bodě existuje derivace. Gradient nám slouží pro nalezení lokálního maxima z libovolného bodu ztrátové funkce. Cílem však není najít maximum chybové funkce, ale její minimum. Musíme se tedy vydat ve směru záporného gradientu. Jméno algoritmu zpětného šíření chyby vychází z postupu výpočtu jednotlivých derivací. Výpočet parciálních derivací chybové funkce podle parametrů modelu začínáme ve výstupní vrstvě a pokračujeme zpět neuronovou sítí až do první skryté vrstvy. V každé vrstvě vypočteme derivace jednotlivých aktivací (pro aktivace neuronů ve skrytých vrstvách platí vztah (12), pro neurony ve výstupní vrstvě je derivace podle výrazu (13)), ze kterých pak vycházíme při výpočtu parciálních derivací chybové funkce podle parametrů modelu neuronové sítě. Parciální derivace pro neurony ve skrytých vrstvách jsou ovlivněny všemi parciálními derivacemi neuronů, které jsou ve vrstvách vedoucích od aktuální směrem k výstupní vrstvě. Parciální derivace chybové funkce podle aktivační funkce neuronu v L-té vrstvě je rovna součtu parciálních derivací chybové funkce podle aktivačních funkcí neuronů ve vrstvě $L+1$. Jednotlivé parametry gradientu pak vypočítáme podle vztahu (11).

$$\frac{\partial J}{\partial \theta_{ji}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial \theta_{ji}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial J}{\partial a_j^{(L)}} \tag{11}$$

kde:

$\frac{\partial J}{\partial \theta_{ji}^{(L)}}$ – parciální derivace chybové funkce podle parametru v L-té vrstvě vedoucí z i-tého neuronu v předchozí vrstvě do j-tého neuronu v L-té vrstvě,

$\frac{\partial z_j^{(L)}}{\partial \theta_{ji}^{(L)}}$ – parciální derivace agregační funkce j-tého neuronu v L-té vrstvě podle parametru v L-té vrstvě vedoucí z i-tého neuronu v předchozí vrstvě do j-tého neuronu v L-té vrstvě,

$\partial \theta_{j0}^{(L)}$ – pokud je hodnota na indexu $i = 0$, jde o práh j-tého neuronu,

$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$ – parciální derivace aktivační funkce j-tého neuronu v L-té vrstvě podle agregační funkce j-tého neuronu v L-té vrstvě,

$\frac{\partial J}{\partial a_i^{(L)}}$ – parciální derivace chybové funkce podle aktivační i-tého neuronu v L-té vrstvě, kterou vypočítáme podle vztahu (12) ve skryté vrstvě a podle vztahu (13) ve výstupní vrstvě.

$$\frac{\partial J}{\partial a_i^{(L)}} = \sum_{j=1}^{n_{L+1}} \frac{\partial z_j^{(L+1)}}{\partial a_i^{(L)}} \frac{\partial a_j^{(L+1)}}{\partial z_j^{(L+1)}} \frac{\partial J}{\partial a_j^{(L+1)}} \tag{12}$$

kde:

$\frac{\partial J}{\partial a_i^{(L)}}$ – parciální derivace chybové funkce podle aktivační funkce i-tého neuronu v L-té vrstvě,

j – index neuronů ve vrstvě $L + 1$,

n_{L+1} – počet neuronů ve vrstvě $L + 1$,

$\frac{\partial z_j^{(L+1)}}{\partial a_i^{(L)}}$ – parciální derivace agregační funkce j-tého neuronu ve vrstvě $L + 1$ podle aktivační funkce i-tého neuronu v L-té vrstvě,

$\frac{\partial a_j^{(L+1)}}{\partial z_j^{(L+1)}}$ – parciální derivace aktivační funkce j-tého neuronu ve vrstvě $L + 1$ podle agregační funkce j-tého neuronu ve vrstvě $L + 1$,

$\frac{\partial J}{\partial a_j^{(L+1)}}$ – parciální derivace chybové funkce podle aktivační funkce j-tého neuronu ve vrstvě L+1.

$$\frac{\partial J}{\partial a_i^{(L)}} \tag{13}$$

kde:

$\frac{\partial J}{\partial a_i^{(L)}}$ – parciální derivace chybové funkce podle aktivační funkce i-tého neuronu v L-té vrstvě.

2.8 Klesání podle gradientu

Derivace dvourozměrné funkce v bodě nám říká, jak se sledovaná funkce ve směru osy x mění. Pokud je hodnota derivace v bodě rovna nule, jedná se o kritický bod. Pokud je pro tento kritický bod derivace druhého řádu kladná, jedná se o lokální minimum. Na základě této znalosti můžeme vyhledat všechny tyto kritické body a podle derivací druhého řádu určit, zda se jedná o lokální minimum a ze všech těchto bodů vybrat ten, který je nejníže položen, tedy globální minimum. Musíme však vycházet ze skutečnosti, že rozměr chybové funkce je udáván počtem všech parametrů, kterých v modelu neuronové sítě může být nespočet. Díky této skutečnosti není efektivní vyhledávat globální minimum výběrem nejníže položeného kritického bodu, protože hledání takových bodu v mnoharozměrném prostoru zabere hodně času. Proto bylo důležité přijít s algoritmem, který nalezne minimum iterativně.

Algoritmus klesání podle gradientu je iterativní optimalizační algoritmus, jehož cílem je nalézt minimum chybové funkce ([6], s. 82). Tedy nalézt takové hodnoty parametrů modelu, které minimalizují její výstup. K nalezení minima musíme rozhodnout, kdy budeme upravovat hodnoty jednotlivých parametrů. Existuje několik možností. První upravuje parametry modelu po každém vypočtení odezvy na prvek z trénovací množiny (online). Nevýhodou však je, že musíme pokaždé znovu vypočítat gradient chybové funkce v bodech aktuálního nastavení parametrů pomocí algoritmu zpětného šíření chyby, což je samo o sobě časově nákladná operace. Druhou variantou je upravovat parametry modelu po vypočtení průměrné odezvy na všech prvcích z trénovací množiny (full-batch). Tento přístup nevyžaduje častý výpočet gradientu chybové funkce, avšak nalezení minima nám zabere spoustu času, jelikož musíme po každém kroku směrem k minimu projít celou trénovací množinu. Třetím způsobem, který je z pohledu časové náročnosti optimální, je upravovat hodnoty parametrů po malých dávkách. Vypočítáme průměr výkonnosti modelu na pevně definovaných částech trénovací množiny a upravíme hodnoty parametrů modelu na základě průměrné chyby na vzorku trénovacích dat (mini-batch). Rychlost učení je pak ovlivněna koeficientem rychlosti učení a velikosti jednotlivých dávek. Další důležitou volbou je hodnota koeficientu rychlosti učení. Vybereme-li hodnotu příliš vysokou, může se stát, že při procesu optimalizace nenalezneme v rozumné době minimum funkce. Může se dokonce stát, že minimum nenajdeme vůbec. Pokud hodnota koeficientu bude příliš malá, k minimu nedojdeme v rozumném čase ([6], [8]).

$$W = W - \propto \nabla_{\theta} J(\theta) \tag{14}$$

kde:

W – vektor všech vah modelu,

$\propto$ – koeficient rychlosti učení,

θ – vektor všech parametrů modelu,

$\nabla_{\theta} J(\theta)$ – gradient chybové funkce podle parametrů modelu v bodě aktuálních hodnot parametrů.

3 HLUBOKÉ UČENÍ

Algoritmy hlubokého učení patří do množiny algoritmů strojového učení. Na rozdíl od ostatních algoritmů strojového učení umí algoritmy hlubokého učení automaticky hledat v datech příznaky procesem učení. Algoritmy hlubokého učení využívají různě uspořádané neuronové sítě s tím, že musí mít minimálně jednu skrytou vrstvu (proto „hluboké učení“). Existují dva typy modelů hlubokého učení. Regresní modely, které slouží pro predikci spojitých hodnot například predikce vývoje venkovní teploty na další dny. Dalším typem jsou klasifikační modely, které slouží pro zařazení vstupu do jedné z klasifikačních tříd například klasifikace koček a psů na základě obrázkových dat. Hluboké učení spadá do kategorie učení s učitelem, protože data, která modely hlubokého učení zpracovávají, musí být předem označena.

3.1 Regresní modely

Úkolem regresních modelů je aproximovat funkci, který produkuje spojité výstupy. Množina trénovacích a testovacích dat obsahuje vstup a předpoklad, který je reprezentován spojitou veličinou. Od klasifikačních modelů se liší tím, že výstup není diskrétní veličina, ale spojitá veličina. Po natrénování regresního modelu předpovídáme chování spojité funkce na základě vstupních dat. Na výstupech regresních modelu používáme lineární aktivační funkci, protože na výstupu očekáváme hodnoty v intervalu $(-\infty, \infty)$. Jiné aktivační funkce jejich vstup obvykle modulují, což je pro výstup z regresních modelů nežádoucí.

3.2 Klasifikační modely

Úkolem klasifikačních modelů je aproximovat funkci, která mapuje vstupy na diskrétní veličiny. Jednotlivé diskrétní veličiny reprezentují třídy, do kterých se model neuronové sítě učí mapovat vstupy. Trénovací a testovací množiny dat obsahují vstup a předpoklad, který je reprezentován diskrétní veličinou. Na výstupu klasifikačních modelů používáme aktivační funkce v závislosti na počtu tříd, do kterých vstup můžeme klasifikovat. Pokud klasifikujeme do více jak dvou tříd, je vhodné použít funkci $softmax$, která výstupy z jednotlivých neuronu, které reprezentují jednotlivé třídy, získá distribuci pravděpodobnosti. Tedy s jakou pravděpodobností spadá vstup do jednotlivých kategorií.

4 REGULARIZACE

Při procesu trénování modelu neuronové sítě se může stát, že se model na trénovacích datech přeučí. To znamená, že model v datech najde vlastnosti specifické pouze pro trénovací množinu. V tomto případě bude výkonnost modelu na trénovací množině výrazně vyšší než na testovací množině. Problém přeučení lze řešit natrénováním modelu na větším množství dat nebo použitím jedné nebo více technik regularizace. Cílem regularizace je model penalizovat za složitost. Složitostí myslíme příliš mnoho vrstev s příliš mnoha neurony. Čím je model složitější, tím je větší pravděpodobnost, že se model na datech přeučí, nevyužijeme-li alespoň jedné z technik regularizace ([6], [8]).

4.1 Redukce velikosti sítě

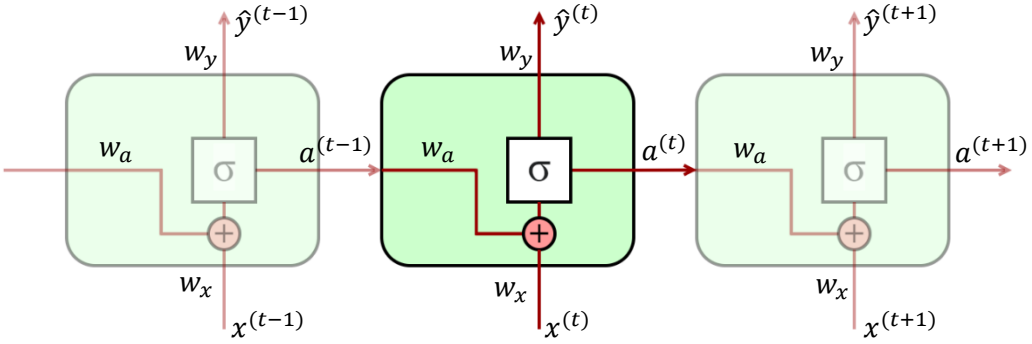
Nejjednodušší technikou regularizace je redukce velikosti sítě. Velikost sítě je často spojována s její kapacitou pojmout data, na kterých model trénujeme. Pokud naše data nejsou příliš komplexní, není nutné, aby náš model měl mnoho vrstev s mnoha neurony. Pokud aproximujeme funkci, která reprezentuje komplexní data, je na začátku vhodné zvolit větší model. Princip redukce velikosti sítě spočívá v tom, že na začátku zvolíme model s o něco větší kapacitou, než je komplexnost dat a sledujeme, jak se model na datech dokáže učit. Když výkonost na testovacích datech je přijatelná, můžeme zvolit menší model a sledovat jeho výkonnost. Proces opakujeme do chvíle, než najdeme co nejmenší model. Výhodou menších modelů je jednak rychlejší odezva, ale také schopnost lépe generalizovat na nových datech.

4.2 Dropout

Regularizační metoda dropout spočívá ve vynechání určitého počtu neuronů v jednotlivých vrstvách v procesu učení za účelem donucení modelu lépe generalizovat. Tím, že v průběhu učení náhodně vynecháme některé neurony v jednotlivých vrstvách, donutíme model neuronové sítě hledat v datech jiné, případně i lépe popisující příznaky, což v konečném důsledku znamená lepší výkonnost modelu na testovacích datech.

5 REKURENTNÍ NEURONOVÉ SÍTĚ

Dalším typem algoritmů strojového učení jsou rekurentní neuronové sítě označovány zkratkou RNN. Nevýhodou nerekurentních neuronových sítí je, že nemají paměť, tedy neumí uchovávat závislosti v datech neboli kontext. Každý vstup do takové sítě je zpracován nezávisle na stavu neuronové sítě při výpočtu odezvy na předchozích vstupních datech ([5], s. 185). Chceme-li však zpracovávat data, které jsou kontextově závislá, musíme použít jeden z typů RNN podle toho, jak dlouhodobé závislosti v datech očekáváme. Pokud závislosti v datech nejsou dlouhodobé, ale i přesto záleží na kontextu, je vhodné použít klasické rekurentní neuronové sítě (Obrázek č. 5). Pokud však v datech očekáváme dlouhodobé závislosti, klasické RNN nepřichází v úvahu, protože z důvodu mizícího gradientu neumí udržet dlouhodobé závislosti v datech. Tento problém je například řešen rekurentními neuronovými sítěmi typu LSTM.



Obrázek č. 5 – RNN, zdroj: ([11], s. 2)

Stavy rekurentní neuronové sítě v časovém intervalu

$$\langle t-1, t+1 \rangle.$$

RNN pro zachování kontextu využívají takzvaný skrytý stav neuronu RNN, který je průběžně předáván pro výpočet následujících stavů neuronu. Na začátku je skrytý stav roven nule. Při každém dalším vstupu se hodnota skrytého stavu změní v závislosti na hodnotě aktuálního vstupu, vah a hodnoty aktivace skrytého stavu v předchozím kroku podle vztahu (15). Výstup z neuronu je vypočten podle vztahu (16), kde figuruje výpočet aktuálního skrytého stavu, tedy kontextu.

$$a^{(t)} = \sigma(w_a a^{(t-1)} + w_x x^{(t)} + b_a) \tag{15}$$

kde:

$a^{(t)}$ – aktivace skrytého stavu neuronu v čase t,

σ – aktivační funkce,

w_a – hodnota váhy vedoucí z předchozího stavu neuronu do aktivační funkce,

$a^{(t-1)}$ – aktivace skrytého stavu neuronu v čase $t-1$,

w_x – hodnota váhy vedoucí ze vstupu do aktivační funkce,

$x^{(t)}$ – vstup do neuronu v čase t,

b_a – hodnota prahu pro hodnotu aktivace skrytého stavu.

$$\hat{y}^{(t)} = \Phi(w_y a^{(t)} + b_y) \tag{16}$$

kde:

$\hat{y}^{(t)}$ – výstup z neuronu v čase t,

Φ – aktivační funkce pro výstup z neuronu,

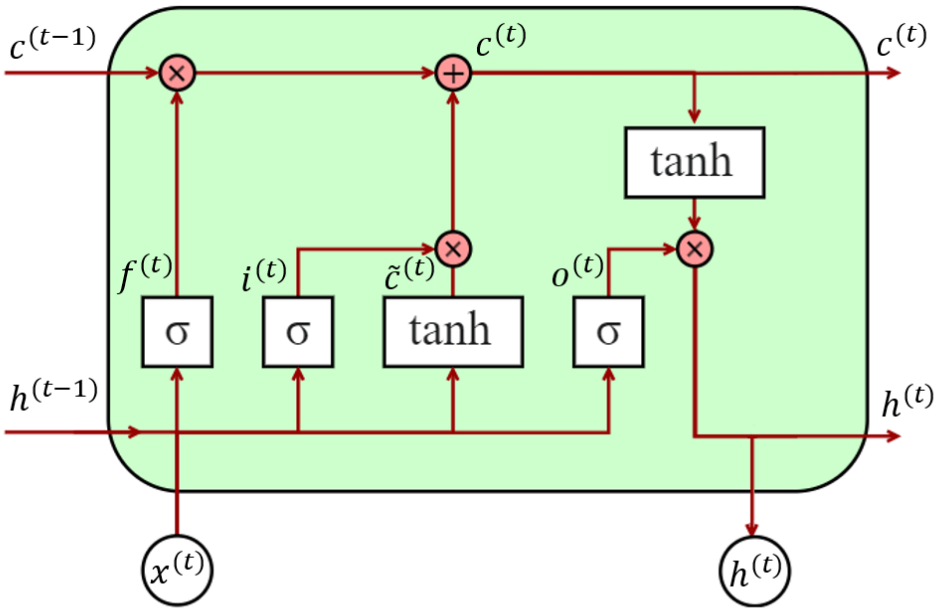
w_y – hodnota váhy pro výstup z neuronu,

$a^{(t)}$ – hodnota aktivace skrytého stavu v čase t, podle vztahu (15),

b_y – hodnota prahu pro výstup z neuronu.

5.1 Long short-term memory

Model long short-term memory (Obrázek č. 6), označovaný zkratkou LSTM, je založen na vyřešení hlavní nevýhody základních RNN, tedy na neschopnosti udržení dlouhodobějších závislostí. Součástí LSTM jsou čtyři brány, kterými řídíme vnitřní stav buňky ([6], s. 408). První brána $f^{(t)}$ slouží pro řízené zapomínání. Ovlivňuje, jaké informace se z paměti v předchozím stavu $c^{(t-1)}$ zachovají a jaké se zapomenou. Lze ji vypočítat podle vztahu (19). Druhá brána $\tilde{c}^{(t)}$ slouží k zavedení aktuálního vstupu $x^{(t)}$ a předchozího skrytého stavu neuronu $h^{(t-1)}$do paměti s tím, že pomocí třetí brány $i^{(t)}$ se vyberou informace, které se uloží do paměti aktuálního stavu $c^{(t)}$. Stav paměti $c^{(t)}$ v čase t je následně vypočten sečtením tenzorů modifikovaného vstupu $\tilde{c}^{(t)}$ a upraveného předchozího stavu paměti $c^{(t-1)}$ podle vztahu (17). Čtvrtá brána $o^{(t)}$ společně s modulovaným stavem paměti buňky $c^{(t)}$ tvoří skrytý stav buňky $h^{(t)}$ podle vztahu (18), který drží závislosti lokálního charakteru a slouží jako vstup do dalšího stavu neuronu. Zároveň lze skrytý stav $h^{(t)}$ postupně ukládat do nezávislé proměnné pro pozdější využití. Optimální nastavení jednotlivých parametrů buňky je řízeno algoritmem sestupu podle gradientu.



Obrázek č. 6 – LSTM, zdroj: ([11], s. 5)

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + \tilde{c}^{(t)} \circ i^{(t)} \tag{17}$$

kde:

$f^{(t)}$ – brána řízeného zapomínání,

$c^{(t-1)}$ – předchozí stav paměti neuronu,

$\tilde{c}^{(t)}$ – kandidátní data pro uložení do paměti,

$i^{(t)}$ – filtrovací brána pro výběr z kandidátních dat.

$$h^{(t)} = o^{(t)} \circ tanh\,(c^{(t)}) \tag{18}$$

kde:

$h^{(t)}$ – skrytý stav neuronu v čase t,

$o^{(t)}$ – výstup z neuronu,

$c^{(t)}$ – stav paměti neuronu v čase t.

$$g^{(t)} = w_{gx} \cdot x^{(t)} + w_{gh} \cdot h^{(t-1)} + b_g \quad (19)$$

kde:

$g^{(t)}$ – obecná hodnota reprezentující aktivaci jedné z bran ($f^{(t)}$, $\tilde{c}^{(t)}$, $i^{(t)}$, $o^{(t)}$),

w_{gx} – hodnota váhy vedoucí ze vstupu x do obecné brány g,

$x^{(t)}$ – vstup do neuronu v čase t,

w_{gh} – hodnota váhy vedoucí ze skrytého stavu v předchozím kroku,

$h^{(t-1)}$ do obecné brány g,

b_g – práh neuronu pro obecnou pránu g.

6 DETEKCE NEOBVYKLÝCH STAVŮ

Stavy, které nesplňují charakteristiky standardního chování se nazývají neobvyklé stavy nebo anomálie ([4], s. 1). Chování, které se v datech často opakuje lze označit jako normální, ale nemusí tomu tak vždy být. V kontextu hodnot se jedná o body, které jsou příliš odlehlé.

6.1 Detekce založená na klasifikaci

Základním předpokladem pro detekci anomálií založené na klasifikaci modely neuronových sítí je mít velké množství označených dat. To znamená projít co nejvíce dat a ručně označit, jedná-li se o anomálii nebo ne. Je to jak časově, tak i finančně velmi nákladná činnost a často vyžaduje profesionální znalost domény. Přesnost modelu úzce souvisí s množstvím správně označených dat. Čím více, tím lepší přesnost. Zároveň je nutné, aby v trénovacích i testovacích datech bylo množství normálních stavů a anomálií podobné, abychom model dokázali natrénovat rozpoznat jednotlivé stavy. Velkou nevýhodou je, že k natrénovaní klasifikačního modelu, abychom dosáhli přijatelné přesnosti při detekování, musíme mít opravdu velké množství dat. Pokud nemáme potřebné prostředky k obstarání takového množství označených dat, je lepší přistoupit k jiným metodám strojového nebo hlubokého učení ([1], [4], [6]).

6.2 Detekce pomocí regresních modelů

U regresních modelu, které jsou založeny na čistě dopředných neuronových sítí postrádáme kontext, který předchází novým spojitým datovým bodům, tudíž výkonnost těchto modelu bude nízká, pokud jsou všechny hodnoty závislé na kontextu. Vyšší výkonnosti můžeme dosáhnout pomocí rekurentních modelů neuronových sítí, například využitím LSTM buněk. LSTM jsou oproti klasickým rekurentním neuronovým sítím schopny udržet dlouhodobější kontext. Toho lze při regresi využít, jelikož jsme pak schopni ovlivnit výslednou reakci modelu na základě datových bodů, které se ve spojité funkci nacházejí dříve a mohou mít určitý vliv na nové body spojité funkce. Anomálie pak můžeme detekovat tak, že pokud se nový datový bod příliš liší od predikce, znamená to, že s velkou pravděpodobností se jedná právě o anomálii.

6.3 Detekce pomocí auto-enkodérů

Auto-enkodéry patří do samořízených algoritmů hlubokého učení. Od klasického řízeného hlubokého učení se auto-enkodéry liší tím, že nepotřebujeme označená data. Pomocí auto-enkodérů hledáme nízko-dimenzionální reprezentaci původních dat, kterou pak rekonstruujeme do původní podoby. Chybová funkce je vyjádřena rozdílem mezi zrekonstruovanými a původními daty. Pro detekci anomálií musíme auto-enkodéry naučit rekonstruovat dostatečné množství normálních datových bodů, abychom při nasazení modelu do běhu detekovali anomálie tím, že model nebude schopen zrekonstruovat neobvyklá data z vytvořené nízko-dimenzionální reprezentace zpět do původní podoby. Jinými slovy, model se učí rekonstruovat normální data, a když narazí na data, která jsou anomáliemi, není je schopen zrekonstruovat s rozumně nízkou chybou. To znamená, že chyba mezi vstupem a výstupem modelu bude signifikantně velká, což indikuje neobvyklá data.

6.4 Detekce založená na shlukování

Shlukování je metoda hledání shluků na základě podobnosti jednotlivých bodů v datech. V závislosti na komplexnosti dat lze očekávat větší počet shluků neboli klastrů. Pokud existuje bod, který není přiřazen do jednoho ze shluků, velmi pravděpodobně se jedná o anomálii. Anomálie však může být i shluk, který je tvořen výrazně méně body než ostatní shluky.

6.4.1 Shlukování podle k-means

Metoda, díky které jsme na neoznačených datech schopni hledat podobnosti mezi jednotlivými datovými body a na základě těchto podobností vytvářet shluky. Cílem je vybrat takový počet shluků, pro které platí, že rozptyl bodů od středu (průměru) shluku je co nejmenší[1].

6.5 Chyby při testování hypotéz

V kontextu detekce anomálií je platná hypotéza nalezení anomálie. Pokud máme model, který v datech nachází chyby I. druhu (falešně pozitivní výsledek), tedy model určí, že se jedná o anomálii. Skutečností však je, že hypotéza není platná. V konečném důsledku se nic nestalo,

[1] pro více informací vizte [6], s. 150.

avšak z lidského pohledu může dojít k tomu, že uživatel systém přestane používat, protože mu jednoduše nevěří.

Chyby II. druhu (falešně negativní výsledek) jsou v porovnání s chybou I. druhu horší. Jednak uživatel může systému přestat věřit, a navíc model neúspěšně detekuje neobvyklou spotřebu vody, což muže mít i fatální následky. Cílem je najít takový model, který bude co nejpřesněji detekovat neobvyklé stavy a zároveň korektně určovat, že chování spotřeby vody je obvyklé.

7 POPIS ŘEŠENÍ

K problematice detekce neobvyklé spotřeby vody lze přistoupit mnoha způsoby. V této bakalářské práci je použita regrese a auto-enkodéry pomocí hlubokých neuronových sítí a rekurentních neuronových sítí LSTM. U všech typů modelů budeme sledovat rozdíl předpovědi a aktuálního stavu s tím, že pomocí koeficientem vynásobeného mediánu absolutních vzdáleností od mediánu rozdílů trénovací množiny a predikcí modelu na trénovací množině lze určit, jestli se jedná o neobvyklý stav nebo ne. Všechen zdrojový kód je napsán v programovacím jazyce Python.

Nejprve je nutné připravit data do patřičného formátu a vytvořené modely na nich natrénovat. U regresních modelů, jak hlubokých neuronových sítí, tak i LSTM sítí, aproximujeme cílovou funkci, tedy model spotřeby pro danou domácnost. Na výstupy regresních modelů máme pouze očekávanou spotřebu s tím, že vstupy se liší. U hlubokých neuronových regresních modelů je použita cyklická reprezentace časových hodnot[2] jako vstup a u rekurentních neuronových sítí LSTM je na vstupu použita upravená forma spotřeby s tím, že každé hodnotě předchází několik datových bodů.

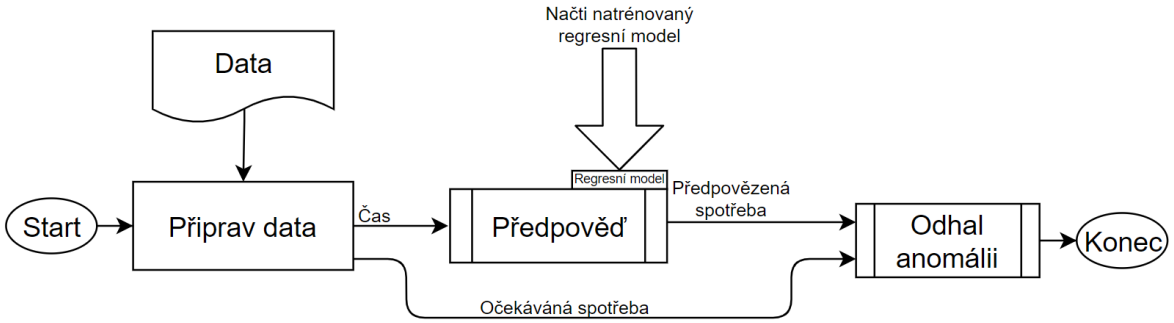
U auto-enkodérů je cílem najít nízko-dimenzionální reprezentaci vstupu a poté zrekonstruovat vstupní data co nejpřesněji do původní podoby. Idea u auto-enkodéru je, že pokud data naučíme z větší části na normálních datech, ve kterých se nacházejí neobvyklé stavy v menším množství, model pak bude tyto odlehlé body rekonstruovat s větší chybou než body normální. Pro LSTM auto-enkodéry se na vstupu používá stejná forma dat, jako pro LSTM regresní modely s rozdílem, že na výstupu očekáváme to samé, co na vstupu. Pro auto-enkodéry hlubokých neuronových sítí také na výstupu očekáváme to samé co na vstupu. Chybou pak je rozdíl mezi vstupem a výstupem, jinými slovy, jak se modelu povedla rekonstrukce vstupních dat.

Detekce u všech výše zmíněných modelů je založena na rozdílu očekávané spotřeby a modelem predikované spotřeby. Z těchto rozdílů neboli chyb vypočítáme medián, protože v datech (Obrázek č. 10) se velmi pravděpodobně nacházejí anomálie. Na základě vypočteného mediánu získáme medián absolutních odchylek (MAD) chyb predikce modelu (21). Vynásobíme-li MAD koeficientem citlivosti na přítomnost anomálií získáme práh, pomocí kterého budeme určovat, zdali se konkrétní bod příliš neliší od modelem predikované hodnoty.

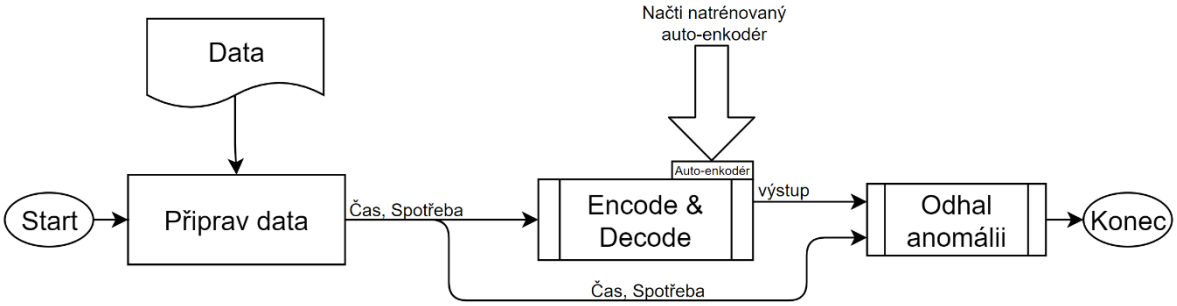
[2] Více o cyklické reprezentaci dat vizte [12], s. 1.

7.1 Architektura řešení

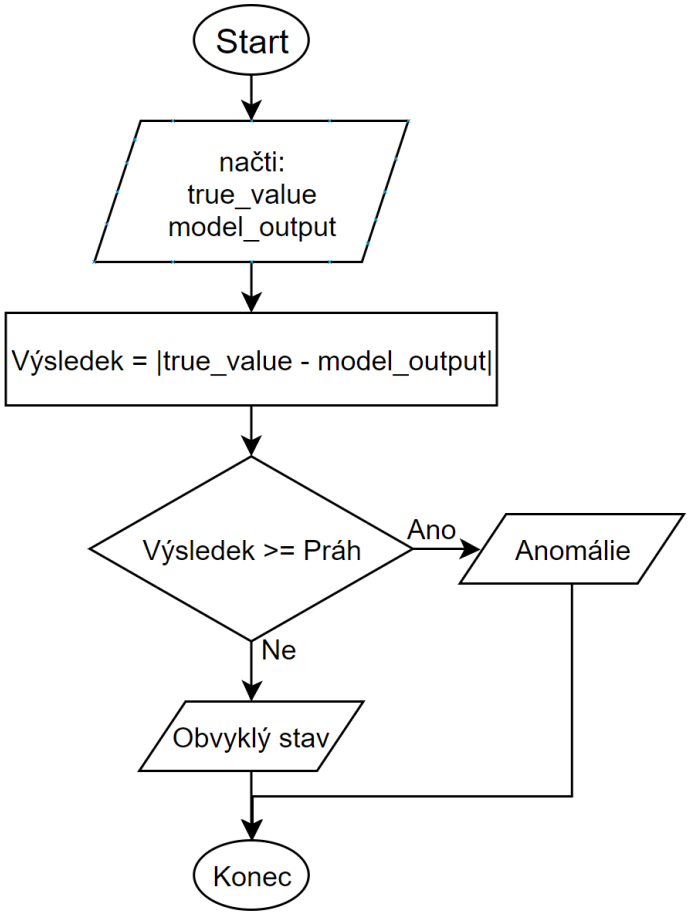
Algoritmus detekce neobvyklé spotřeby vody je složen ze dvou základních částí. První částí je model hlubokého učení. Pro regresní modely produkujeme očekávanou spotřebu na základě cyklicky upravených hodnot času, kterou porovnáváme se skutečností (Obrázek č. 7). U auto-enkodérů je skutečná spotřeba společně s cyklicky reprezentovanými informacemi o čase přivedena na vstup s tím, že stejné hodnoty následně očekáváme i na výstupu (Obrázek č. 8). Druhou částí je samotná detekce, která v obou případech spočívá v tom, že získáváme rozdíl mezi skutečností a produktem modelu. Výsledný rozdíl následně porovnáváme s vypočteným prahem (8.2). Pokud bude hodnota spotřeby větší nebo rovna prahu, stav klasifikujeme jako anomálii. V opačném případě se jedná o standardní spotřebu (Obrázek č. 9).



Obrázek č. 7 – Detekce anomálií regresním modelem, zdroj: autor



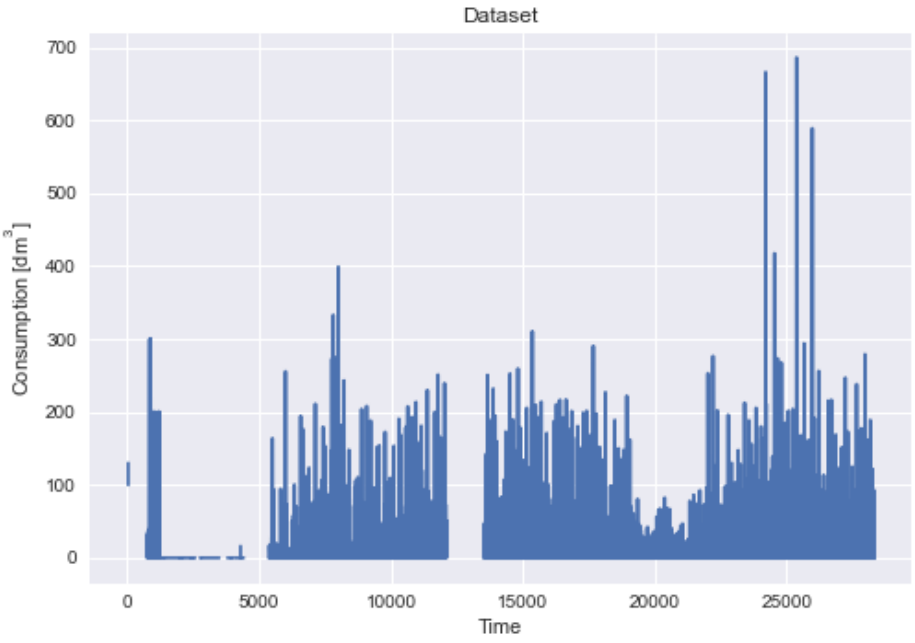
Obrázek č. 8 – Detekce anomálií auto-enkodérem, zdroj: autor



Obrázek č. 9 – Algoritmus detekce anomálií, zdroj: autor

8 PŘÍPRAVA DAT

Jelikož se málokdy stane, že data, která se mají zpracovat budou čistá, tak i náš případ není výjimkou. V tomto případě byly v datech mezery ve formě chybějících měření v konkrétní datové úseky. Důležité rozhodnutí bylo, jakým způsobem se vypořádat s chybějícími daty. Jelikož spotřeba vody, je ovlivněna více faktory než jenom samotným předpokladem, že se v konkrétní hodinu pravidelně spotřebovává voda, nelze jednoduše doplnit chybějící body průměrnou hodnotou spotřeby. Jak lze vidět na grafu spotřeby (Obrázek č. 10), v datech se nacházejí signifikantně velké mezery, jejichž doplnění by znamenalo přílišné zkreslení pro modelování modely neuronových sítí. Data byla podle největších mezer rozdělena na dvě části s tím, že menší část byla rozdělena na testovací a validační data. V rozdělených úsecích se také nacházejí chybějící měření, ale lze předpokládat, že v tu dobu byla spotřeba nulová, protože velikost těchto úseků je v rozmezí jednoho dne až několika málo dní.



Obrázek č. 10 – Původní data, zdroj: autor

Surová data po doplnění chybějících časových úseků.

8.1 Normalizace připravených dat

Zkoumaná data je nutné před zpracováním neuronovými sítěmi normalizovat (viz [5], s. 103), protože tím zajistíme nejen rychlejší konvergenci k optimálnímu nastavení sítě, ale sjednotíme tak formát vstupních dat. Pro časové hodnoty je před normalizací vhodné převést je do jejich cyklického vyjádření ([12], s. 1). Tato transformace je nutná, protože model tak naučíme, že například vzdálenost mezi nedělí a pondělím může být zároveň jeden den (za předpokladu, že pondělí následuje po neděli) a sedm dní, tedy stejná jako například mezi čtvrtkem a pátkem. Pokud bychom pro hodnoty v týdnu zvolili interval ⟨1,7⟩, potom model učíme, že vzdálenost mezi pondělím a nedělí je sedm dní. Ten samý problém se týká času (hodin) a měsíců v roce. Následující kód slouží pro transformaci dat do jejich cyklické podoby.

```
def __encode(self, data, col_name, col_df, max_val):
    data[col_name + '_sin'] = np.float32(np.sin(2 * np.pi * col_df /
max_val) + 1).astype(np.float16)
    data[col_name + '_cos'] = np.float32(np.cos(2 * np.pi * col_df /
max_val) + 1).astype(np.float16)
```

Co se týče anomálií ve spotřebě vody, velice silně ovlivňují normalizovanou podobu spotřeby vody. Pro tento problém lze použít i jiné metody škálování například užití mediánu pro normalizaci, protože hodnota mediánu není příliš ovlivněna odlehlými body. V této bakalářské práci jsou konečné reprezentace dat normalizovány podle jejich průměru.

Normalizovaná data je vhodné zpět denormalizovat jednak pro pozdější použití, tak i pro validaci, že detekované anomálie pasují na původní data. Pro tento účel lze využít funkce třídy „MinMaxScaler“ z projektu „Scikit-learn“ ([13], s. 1).

8.1.1 Implementace normalizace a příprava dat

Nejprve bylo nutné připravit původní data pomocí třídy *TimeSeriesDataPreprocessing*, kde pomocí metody *prepare_data()* jsou data rozdělena na trénovací, testovací a validační množiny a doplněna o chybějící měření. Třída *DataFetcher* a její metoda *get_data()* zde slouží pouze pro načtení dat do programu. Předtím, než jsou data dále použita v programu, jsou pomocí metody *get_normalized_data()* normalizována s tím, že v parametru této metody je možné specifikovat, jaká část dat bude normalizována. Za použití výchozího parametru jsou normalizována všechna data.

8.2 Výpočet prahu a detekce

Pro výpočet prahu (20) je nutné vypočítat medián absolutní odchylky (21), který poté vynásobíme koeficientem, který určuje citlivost algoritmu na přítomnost anomálií. Koeficient citlivosti volíme na základě předpokladu, že k neobvyklým stavům nedochází často a spíše nás zajímají hodnoty, které jsou příliš odlehlé. Tedy velké rozdíly očekávaných stavu oproti modelem predikovaných stavů podle vzorce (23). Hodnota prahu (20) je při detekci porovnána s chybou modelu na nových datech a pokud hodnota chyby je rovna nebo práh překročí, jedná se o neobvyklý stav.

$$Práh = k * MAD \tag{20}$$

kde:

$$MAD = median(|error_i - \tilde{x}|)_{i=1}^{n}, \tag{21}$$

$$\tilde{x} = median(Error), \tag{22}$$

$$Error = \left|\hat{Y} - Y\right|, \tag{23}$$

$\hat{Y}$ – množina predikcí modelu,

Y – množina očekávaných hodnot,

k – koeficient citlivosti na přítomnost anomálií.

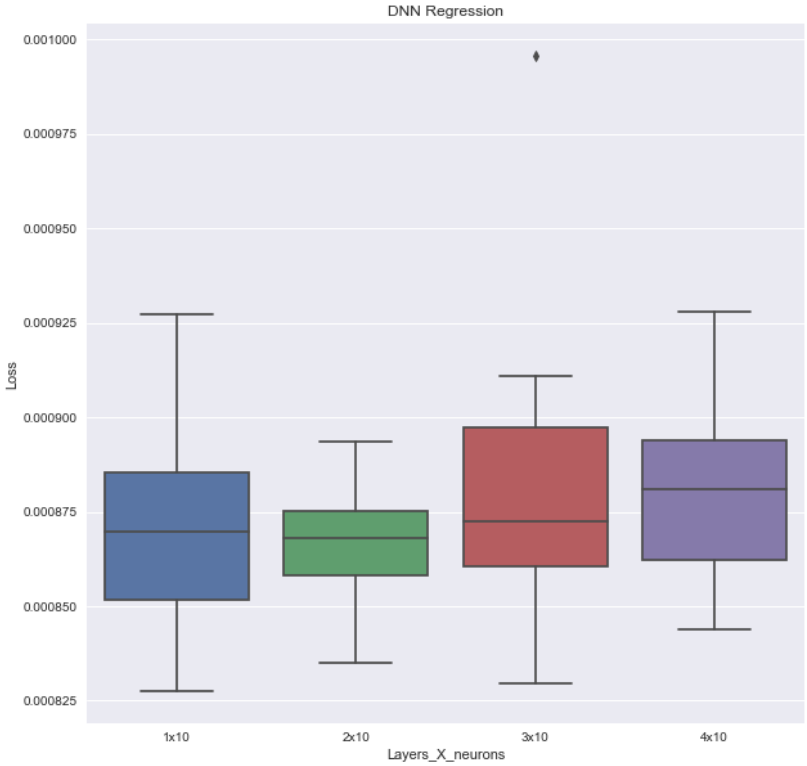
9 REGRESNÍ MODELY

Při tvorbě regresních modelů je vhodné vycházet ze dvou přístupů. Jednak regrese pomocí klasických hlubokých neuronových sítí nebo pomocí rekurentních neuronových sítí. Při hledání vhodného modelu je důležité mít připraveno několik možných topologií a otestovat, jaká jsou nejlepší pro problematiku detekce nestandardní spotřeby vody. Samotná detekce je založena na tom, že předpovídáme průběh spotřeby vody v budoucí hodinu nebo den a poté na základě rozdílu předpovědi a aktuálního stavu se rozhodujeme, zdali se jedná o nestandardní spotřebu vody díky nastavenému prahu na trénovacích datech.

9.1 Regrese pomocí hluboké neuronové sítě

Regrese pomocí hlubokých neuronových sítí vyžaduje úpravu a výběr vstupních vlastností z původních dat. Vstupními parametry do hluboké neuronové sítě jsou upravené reprezentace hodin, dnů a měsíců[3]. Ke každému záznamu ve vstupních datech se váží očekávané výstupy, tedy spotřeba vody v konkrétní chvíli. Po natrénování modelu je výstupem očekávaná hodnota spotřeby v konkrétní hodinu, den a měsíc. Protože framework Keras každé učení začíná s jinými hodnotami vah za účelem vyšší pravděpodobnosti nalezení globálního minima chybové funkce neuronové sítě, bylo nutné provést experiment hledání správného nastavení vah několikrát. Obrázek č. 11 ukazuje vykreslené jednotlivé experimenty, kde na ose *x* vidíme popis jednotlivých topologií.

[3] V kapitole 8.1 je popsána cyklická úprava hodin, dnů a měsíců

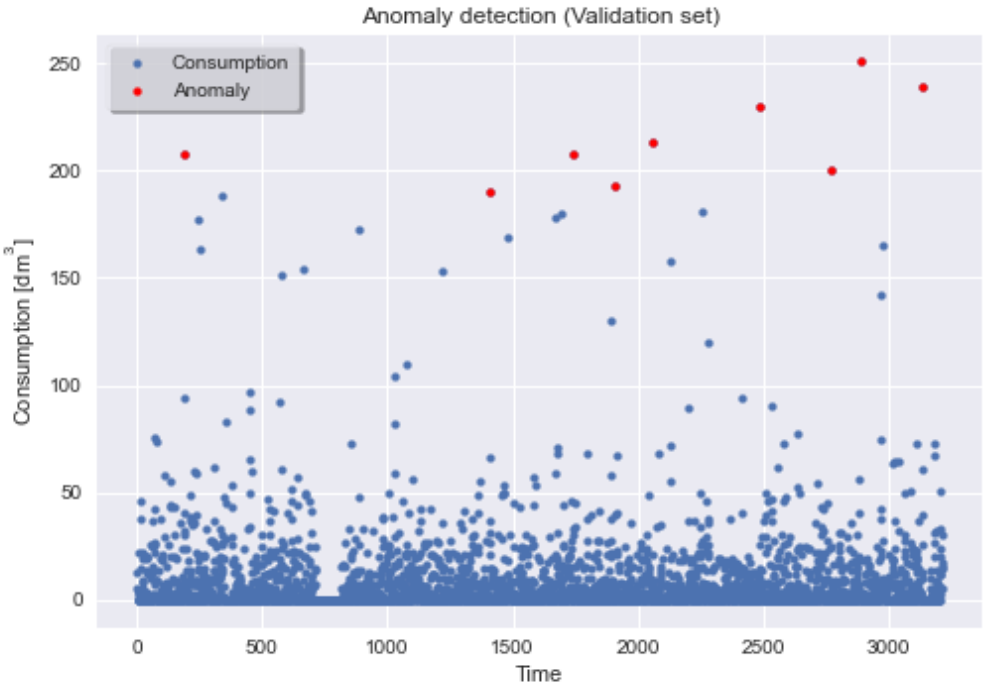


Obrázek č. 11 – Hustoty ztráty DNN regrese, zdroj: autor

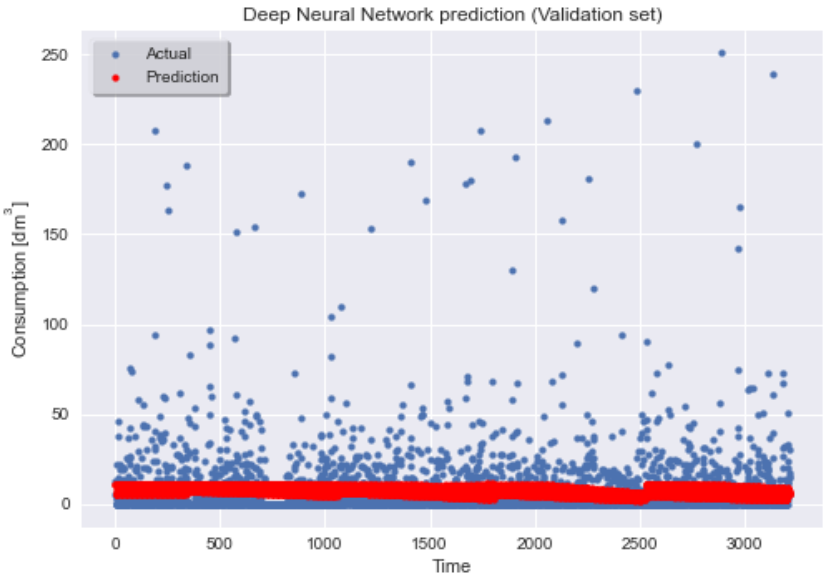
Grafy hustoty hodnot ztráty jednotlivých experimentů pro různé topologie jednotlivých hlubokých neuronových sítí.

Cílem je najít takovou topologii, která bude mít nejmenší mezikvartilní rozpětí a nejnižší průměr. V tomto případě se jedná o model s dvěma skrytými vrstvami po deseti neuronech, tedy druhý typ modelu v pořadí (Obrázek č. 11). Vybereme takové nastavení vah, které má pro nalezenou topologii nejmenší chybu, čemuž odpovídá minimální hodnota na druhém krabicovém grafu (Obrázek č. 11).

Samotná detekce pak probíhá tak, že počítáme chybu modelu na nových datech a porovnáváme ji s prahem, který získáme na trénovacích datech (vizte kapitolu 8.2). Detekce modelem je zdánlivě podobná použití lineární regrese, protože nastavením vyšší hodnoty prahu pouze zvyšujeme hranici hodnoty spotřeby, za kterou jsou body klasifikovány jako anomálie (Obrázek č. 12).



Obrázek č. 12 – Detekce DNN regrese, zdroj: autor



Obrázek č. 13 – Predikce DNN regrese, zdroj: autor

9.1.1 Implementace DNN a detekce anomálií

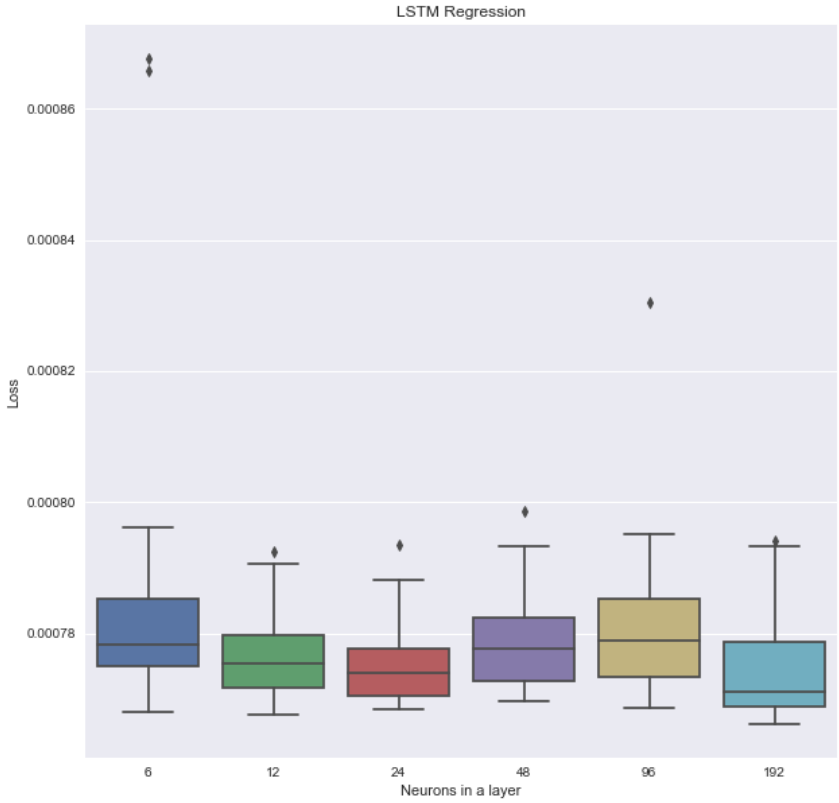
Implementaci trénování modelu lze najít v souboru *Models/dnn_models/DeepNeuralNetwork.py*, kde pomocí metody *train_model* lze model sestavit a natrénovat s tím, že je z této metody vrácena natrénovaná síť a historie průběhu ztráty při trénování. Třídá *DeepNeuralNetwork* je použita v soboru *notebooks/training_dnn_notebook.ipynb*, kde je zároveň implementováno natrénování více topologií modelů. V soboru *notebooks/model_selection_dnn_notebook.ipynb* je z natrénovaných modelů vytvořen graf hustoty ztráty různých topologií modelů na validačních datech a na konci je na základě ručního výběru vypsán i nejlepší model z vybrané topologie. Nakonec v souboru *notebooks/detection_dnn_notebook.ipynb*, nalezneme použitou třídu ze stejnojmenného souboru *Models/dnn_models/DeepNeuralNetworkDetection.py*, která zajišťuje nastavení prahu a samotnou detekci anomálií na základě vstupních dat do metody *is_anomaly_detected*.

9.2 Regrese pomocí LSTM sítě

U rekurentních neuronových sítí LSTM se předpokládá, že v datech umí hledat časové závislosti ([6], s. 411). Tohoto efektu lze docílit také tím, že připravíme data do patřičného formátu. Vstupní data upravujeme do tenzoru rozměrů $(počet\ dat, časové\ okno, počet\ vlastností)$, kde $počet\ dat$ odpovídá velikosti neupravené vstupní množiny dat. $Časové\ okno$ znázorňuje, kolik hodnot konkrétnímu datovému bodu předchází. Následující kód slouží pro vytváření vstupních sekvencí pro LSTM, kde se data transformují do výše zmíněného formátu.

```
def create_sequence(self, X, y=None, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X.iloc[i:(i + time_steps)].values)
        if y is not None:
            ys.append(y.iloc[i + time_steps])
    if y is not None:
        return np.array(Xs), np.array(ys)
    else:
        return np.array(Xs)
```

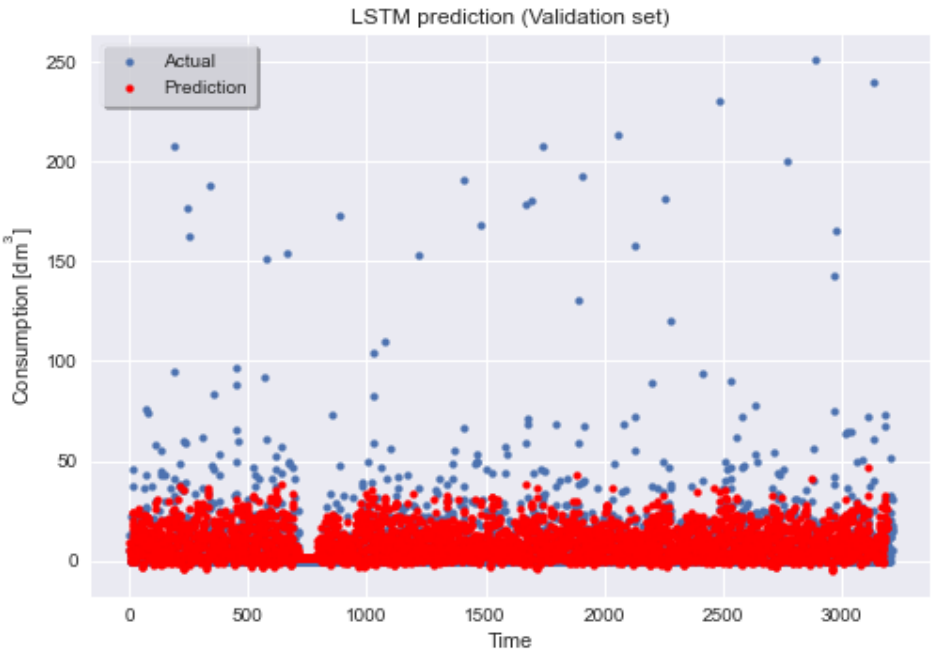
Dalším krokem je vytvořit jednovrstvou LSTM síť a vybrat počet neuronů, který na množinu dat nejlépe pasuje (Obrázek č. 14). Model s 24 neurony má sice nejmenší inter-kvartální rozsah, avšak model s 192 neurony má nejnižší průměrnou ztrátu, tudíž vybereme model se 192 neurony. Obrázek č. 15 porovnává predikci modelu a očekávaných hodnot validační monžiny. Oproti regresi pomocí hluboké neuronové sítě se model LSTM mnohem lépe napasoval na spotřebu. Některé hodnoty však jsou menší než nula.



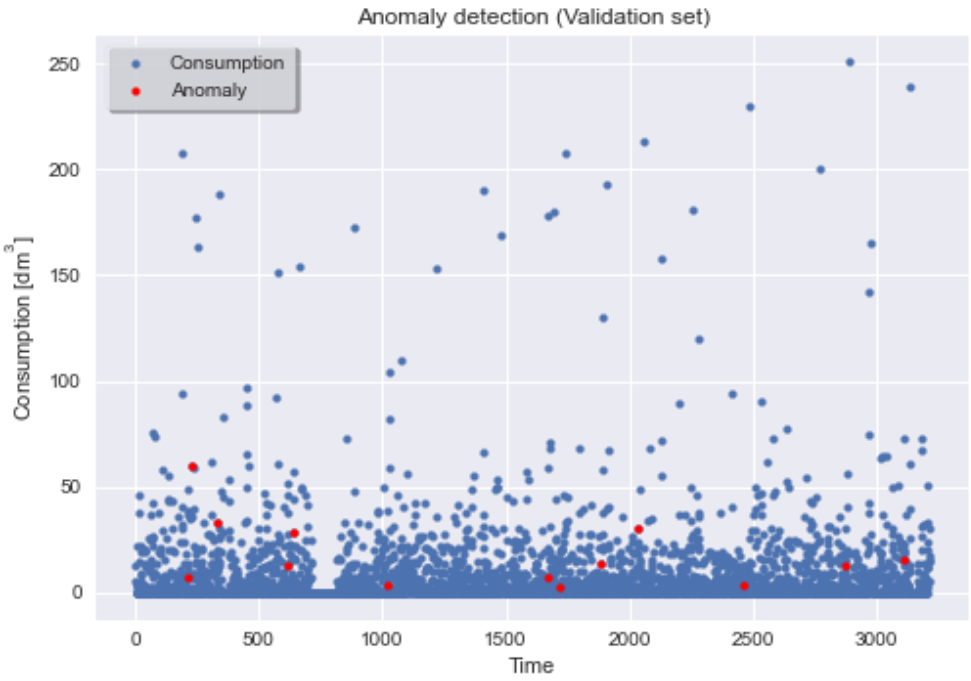
Obrázek č. 14 – Hustoty ztráty LSTM regrese, zdroj: autor

Grafy hustoty hodnot ztráty jednotlivých experimentů
pro různé topologie jednotlivých LSTM sítí.

Princip detekce anomálií u LSTM sítí funguje stejně jako u hlubokých neuronových sítí s tím rozdílem, že při trénování na datech hledáme časové závislosti mezi jednotlivými hodnotami spotřeby. Model na validačních množině dat predikuje lépe než model hluboké neuronové sítě (Obrázek č. 15), avšak přesnost detekce anomálií bude vykazovat více chyb I. druhu (Obrázek č. 16), protože anomálie je buď příliš vysoká nebo nízká hodnota spotřeby. Model v kontrastu detekuje spíše hodnoty blízko průměru (Obrázek č. 16).



Obrázek č. 15 – Predikce LSTM regrese, zdroj: autor



Obrázek č. 16 – Detekce LSTM regrese, zdroj: autor

9.2.1 Implementace LSTM a detekce anomálií

Implementaci trénování modelu lze najít v souboru *Models/lstm_models/LTSMModel.py*, kde pomocí metody *train_model* lze model sestavit a natrénovat s tím, že je z této metody vrácena natrénovaná síť a historie průběhu ztráty při trénování. Třídá *LSTMNetwork* je použita v soboru *notebooks/training_lstm_notebook.ipynb*, kde je zároveň implementováno natrénování více topologií modelů. V soboru *notebooks/model_selection_lstm_notebook.ipynb* je z natrénovaných modelů vytvořen graf hustoty ztráty jednotlivých topologií modelů na validačních datech a na konci je na základě ručního výběru vypsán i nejlepší model z vybrané topologie. Nakonec v souboru *notebooks/detection_lstm_notebook.ipynb*, nalezneme použitou třídu ze stejnojmenného souboru *Models/lstm_models/LSTMModelDetection*.py, která zajišťuje nastavení prahu a samotnou detekci anomálií na základě vstupních dat do metody *is_anomaly_detected*.

10 AUTO-ENKODÉRY

Jak se při řešení této bakalářské práce ukázalo, nejlepší modely z výběru pro detekci anomálií jsou právě auto-enkodéry. Je to vlastně forma samořízeného učení, kdy se snažíme model naučit na trénovacích datech vyjímat jejich zajímavé reprezentace a poté data znovu konstruovat do původní podoby. Samořízené učení zde probíhá formou porovnávání výstupu se vstupem za účelem zlepšení schopnosti rekonstrukce modelu.

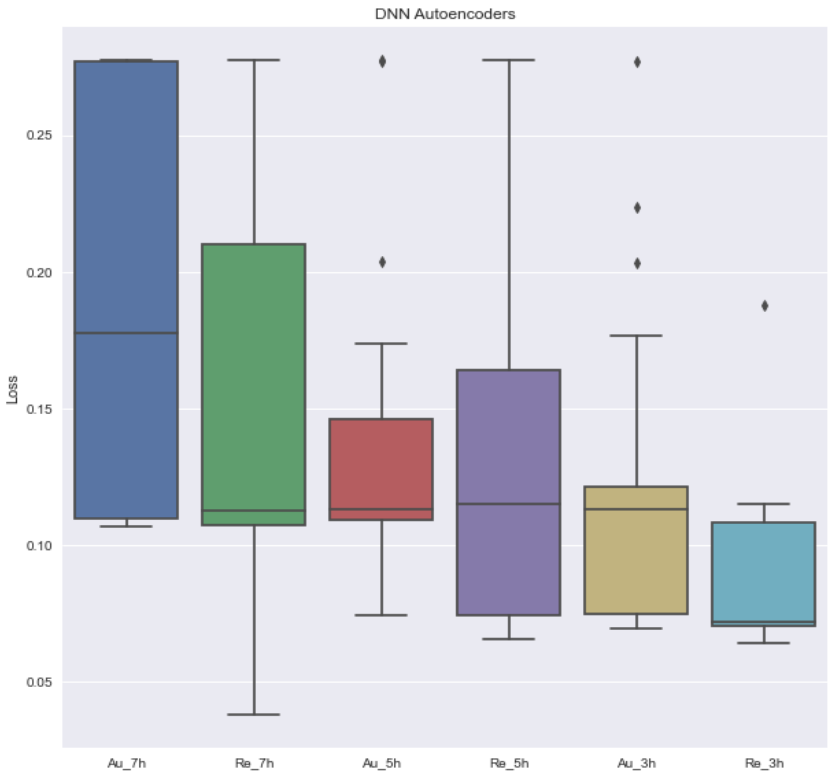
Zajímavou vlastností auto-enkodérů je, že se dokážou naučit detekovat anomálie i na datech, které v sobě obsahují neobvyklé stavy. Nejlepší samozřejmě je, pokud data v sobě anomálie neobsahují, ale nelze vždy s určitostí tvrdit, že data jsou v tomto kontextu čistá ([6], s. 502).

Auto-enkodéry dokážou v datech hledat jejich redukované reprezentace, čehož lze využít například při regresi, kdy na vstupu regresního modelu máme právě výstup z první části auto-enkodéru (enkodér). Předpokladem samozřejmě je, že auto-enkodér prvně naučíme po nalezení redukované reprezentace dat rekonstruovat je do původní podoby. Detekce pak probíhá stejně jako u regresních modelů.

10.1 Auto-enkodér hluboké neuronové sítě

V této bakalářské práci na vstupu auto-enkodérů hlubokých neuronových sítí je využíváno vlastností časových údajů (čas, den v roce, měsíc) společně se spotřebou. Cílem auto-enkodéru je vstupní data rekonstruovat s tím, že pokud se rekonstrukce příliš liší od vstupních dat, lze tento datový bod považovat za anomálii. Jinými slovy, pokud je chyba mezi vstupními daty a modelem zrekonstruovanými daty vyšší než na trénovacích datech zvolený práh (8.2), lze tento bod považovat za neobvyklý.

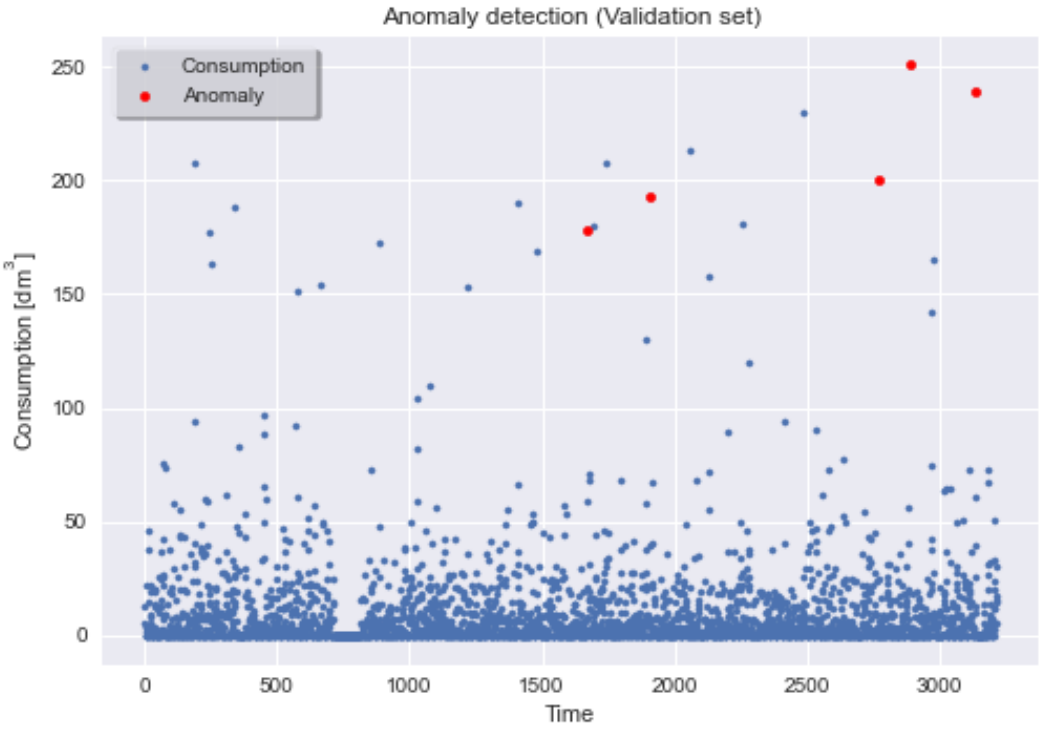
Po provedení několika experimentů s šesti topologiemi hlubokých auto-enkodérů (Obrázek č. 17) vyhrála výběr šestá topologie v pořadí. K této topologii přiřazený krabicový graf má zdaleka nejmenší průměr oproti ostatním topologiím hlubokých auto-enkodérů, ale zároveň i mezikvartilní rozpětí, které je obdobné jako u třetí topologie. Avšak průměr šesté topologie je výrazně nižší než třetí topologie.



Obrázek č. 17 – Hustoty ztráty DNN auto-enkodérů, zdroj: autor

Grafy hustoty hodnot ztráty jednotlivých experimentů pro různé topologie jednotlivých auto-enkodérů hlubokých neuronových sítí.

Obrázek č. 18 ukazuje detekci pomocí hlubokých auto-enkodérů. Jak je vidět, jednotlivé detekované body spotřeby lze s celkem vysokou jistotou považovat za neobvyklé, protože oproti průměru mají body vyšší hodnotu spotřeby. Avšak mezi body s vyššími hodnotami spotřeby se nacházejí i takové body, které modelem nebyly označeny jako anomálie. Mezi těmito body mohou být i anomálie, které model chybně klasifikoval jako obvyklé. Jedná se tedy o chybu II. druhu. Falešně negativní detekce může mít i fatální následky, kdy dotčené ubytovací jednotce při nepřítomnosti ubytovaných může dlouho a nekontrolovatelně unikat voda. Pokud chceme minimalizovat falešně negativní detekce a předejít tak fatálním následkům, můžeme snížit práh modelu, tedy citlivost modelu na detekci neobvyklých stavů. Snížení prahu lze docílit nastavením nižšího koeficientu citlivosti (20). Model pak bude detekovat větší počet anomálií, ale mezi těmito body bude více chybných měření I. druhu, tedy falešně pozitivní měření.



Obrázek č. 18 – Detekce DNN auto-enkodérů, zdroj: autor

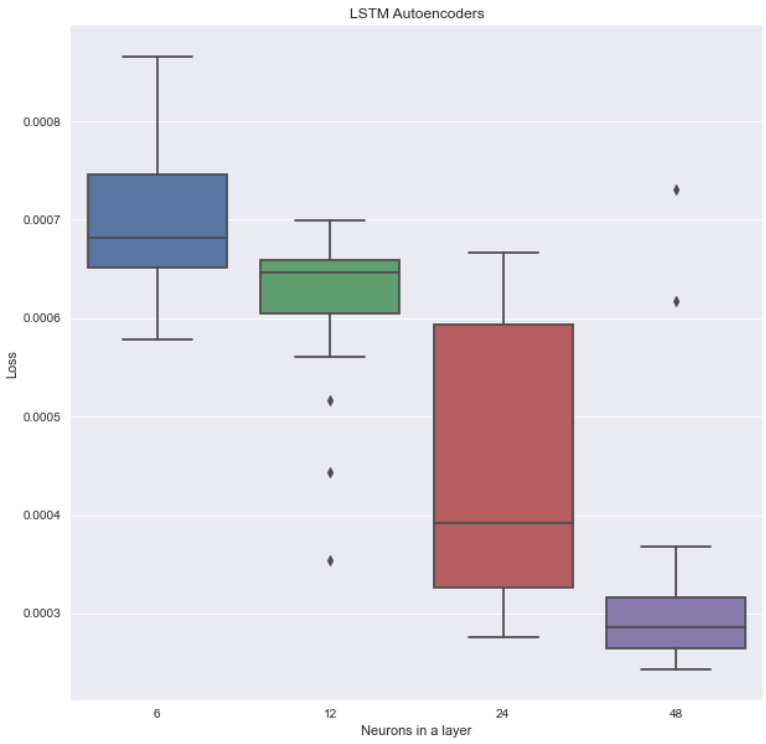
10.1.1 Implementace auto-enkodéru hluboké sítě a detekce anomálií

Implementace trénování modelu lze najít v souboru *Models/dnn_models/DeepAutoencoder.py*, kde pomocí metody *train_model* lze model sestavit a natrénovat s tím, že je z této metody vrácena natrénovaná síť a historie průběhu ztráty při trénování. Třídá *DeepAutoencoder* je použita v soboru *notebooks/ training_dnn_autoencoder_notebook.ipynb*, kde je zároveň implementováno natrénování více topologií modelů. V soboru *notebooks/model_selection_dnn_autoencoder_notebook.ipynb* je z natrénovaných modelů vytvořen graf hustoty ztráty jednotlivých topologií modelů na validačních datech a na konci je na základě ručního výběru vypsán i nejlepší model z vybrané topologie. Nakonec v souboru *notebooks/detection_dnn_autoencoder_notebook.ipynb* nalezneme použitou třídu ze stejnojmenného souboru *Models/dnn_models/DeepAutoencoderDetection.py*, která zajišťuje nastavení prahu a samotnou detekci anomálií na základě vstupních dat do metody *is_anomaly_detected*.

10.2 Auto-enkodér LSTM sítě

Jako u auto-enkodéru hluboké neuronové sítě je hlavním cílem zrekonstruovat vstupní data. Data jsou LSTM auto-enkodéru podávána na vstup v sekvenčně upraveném formátu s tím, že cílem modelu je upravená data znovu rekonstruovat po „vytažení“ jejich nízko-dimenzionální reprezentace.

Výběr topologie modelu (Obrázek č. 19) byl v toto případě jednoduchý, protože čtvrtá topologie v pořadí má jednak nejmenší mezikvartilové rozpětí, ale zároveň průměr přesnosti jednotlivých experimentů je výrazně nižší než průměr experimentů ostatních topologií.

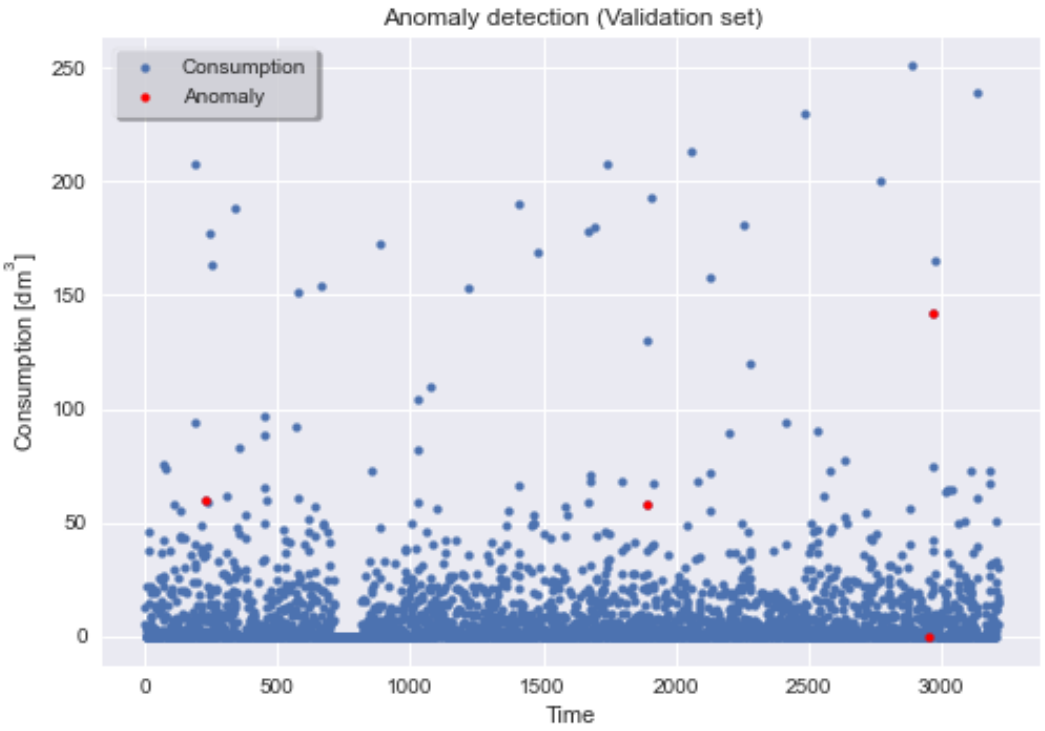


Obrázek č. 19 – Hustoty ztráty LSTM auto-enkodérů, zdroj: autor

Grafy hustoty hodnot ztráty jednotlivých experimentů
pro různé topologie jednotlivých LSTM auto-enkodérů.

Můžeme pozorovat, že počet detekovaných anomálií modelem LSTM auto-enkodéru na validační množině je menší, než počet detekovaný auto-enkodérem hluboké sítě. Tato skutečnost je ovlivněna také nižším ručně nastaveným prahem, protože model detekoval body umístěné spíše v dolní části grafu spotřeby na validační množině. Pokud chceme detekovat anomálie a minimalizovat falešně negativní měření, můžeme vybrat jinou topologii, ladit nastavení prahu

za účelem nalezení optimálního počtu anomálií, které budou detekovány převážně ve vrchní části grafu spotřeby nebo model natrénovat na větším množstvím dat. Další možností, jak najít optimální topologii, je provést více experimentů s rozdílným nastavením vah na jednotlivých zvolených topologií.



Obrázek č. 20 – Detekce LSTM auto-enkodérů, zdroj: autor

Grafy hustoty hodnot ztráty jednotlivých experimentů
pro různé topologie jednotlivých LSTM auto-enkodérů.

10.2.1 Implementace LSTM auto-enkodéru a detekce anomálií

Implementace trénování modelu lze najít v souboru *Models/lstm_models/LSTMAutoencoder.py*, kde pomocí metody *train_model* lze model sestavit a natrénovat s tím, že je z této metody vrácena natrénovaná síť a historie průběhu ztráty při trénování. Třída *LSTMAutoencoder* je použita v soboru *notebooks/training_lstm_autoencoder_notebook.ipynb*, kde je zároveň implementováno natrénování více topologií modelů. V soboru *notebooks/model_selection_lstm_autoencoder_notebook.ipynb* jsou z natrénovaných modelů vytvořeny grafy hustoty ztráty jednotlivých topologií modelů na validačních datech a na konci je na základě ručního výběru vypsán i nejlepší model z vybrané topologie. Nakonec v souboru *notebooks/de-*

tection_lstm_autoencoder_notebook.ipynb, nalezneme použitou třídu ze stejnojmenného souboru *Models/lstm_models/LSTMAutoencoderDetection*.py, která zajišťuje nastavení prahu a samotnou detekci anomálií na základě vstupních dat do metody *is_anomaly_detected*.

ZÁVĚR

Cílem této bakalářské práce bylo najít algoritmus, pomocí kterého budeme schopni detekovat neobvyklé hodnoty spotřeby vody. První částí algoritmu je model, který v případě regrese dokáže modelovat data o spotřebě vody nebo v případě auto-enkodérů rekonstruovat vstupní data s určitou přesností. Ve druhé části figuruje na trénovacích datech vypočtený práh, díky kterému dokážeme o konkrétním záznamu o spotřebě vody říct, zda je či není obvyklý. Způsob výpočtu prahu je u všech modelů podobný. Nejprve jsou vypočítány chyby mezi produktem modelu a očekávaným výstupem. Z chyb následně získáme medián, ze kterého pak počítáme medián absolutních odchylek (MAD), který po vynásobení zvolenou konstantou citlivosti tvoří práh.

Při použití regresního modelu hluboké neuronové sítě pro modelování spotřeby, nebylo dosaženo dostatečně přesných predikcí. Výsledná detekce zdánlivě připomíná pevné nastavení maximální hodnoty běžné spotřeby, jejíž překročení znamená, že model bod klasifikuje jako anomálii. Regrese za použití LSTM sítí model spotřeby dokázala vymodelovat s vysokou přesností. Avšak některé predikované hodnoty na validačních datech jsou záporné a samotná detekce označuje anomálie blíže k průměru spotřeby.

V případě použití auto-enkodéru hluboké neuronové sítě detekoval algoritmus anomálie s vyšší hodnotou spotřeby, avšak neoznačoval všechny body za určitou hranicí, jako tomu je při použití regresního modelu hluboké neuronové sítě. Bude-li algoritmus detekce označovat všechny anomálie za určitou hranicí, může se stát, že pokud si například napustíte bazén, algoritmus akci vyhodnotí jako neobvyklou spotřebu. Auto-enkodér pomocí LSTM sítí v kontrastu detekoval neobvyklé hodnoty spotřeby blíže k průměru. Problémem však bylo, že výsledky na trénovacích a validačních datech byly až příliš rozdílné. Při výběru nízkého koeficientu citlivosti na přítomnost anomálií se na validační množině neobjevily žádné anomálie s tím, že na trénovacích datech jich bylo výrazně více. Obdobně tomu bylo při výběru vyšší hodnoty koeficientu.

Jelikož byla k dispozici data, kde nejsou označeny anomálie, byl algoritmus navržen tak, aby označoval anomálie značně vzdálené od průměru všech měření. Tento záměr v algoritmu detekce anomálií nejlépe plnil auto-enkodér hluboké neuronové sítě, jelikož nejvěrněji označoval anomálie vzdálené od průměru měření.

Zajímavým pokračováním této bakalářské práce může být výzkum, kde hledáme nejvhodnější soubor modelů strojového učení pro detekci anomálii ve spotřebě vody. Výhodou nalezeného souboru je, že soustředíme pozornost jednotlivých modelů na širší spektrum vlastností,

které popisují obvyklost spotřeby. Příkladem může být obvyklá délka kontinuální spotřeby. Překročení určitého intervalu by znamenalo, že model klasifikuje spotřebu jako neobvyklou. Lze zároveň sledovat, kolik vody se za tento interval spotřebuje. Důležité může být, zda někdy naměříme nulovou spotřebu, protože pokud ne, může to znamenat, že voda uniká v malém množství nežádoucím způsobem. Lze modely, které sledují odlišné vlastnosti spotřeby, různými způsoby kombinovat, což je také vhodné pokračování této práce. Základním předpokladem pro tuto detailní analýzu je mít data s kratšími intervaly mezi měřeními v řádu minut či sekund s označenými nežádoucími úniky vody, chceme-li sledovat výkon nalezených souboru modelů.

POUŽITÁ LITERATURA

[1] AGGARWAL, Charu C. *Outlier analysis*. New York: Springer, 2013. ISBN 978-1461463955.

[2] HAWKINS, D. M. *Identification of Outliers*. Dordrecht: Springer Netherlands, 1980. DOI: 10.1007/978-94-015-3994-4. ISBN 978-94-015-3996-8.

[3] KAZMIER, Leonard J. *Schaum's outline of theory and problems of business statistics*. 3rd ed. New York: McGraw-Hill, c1996. ISBN 0070340269.

[4] CHANDOLA, Varun, Arindam BANERJEE a Vipin KUMAR. Anomaly detection. *ACM Computing Surveys*. 2009, 41(3), 1-58. DOI: 10.1145/1541880.1541882. Dostupné také z: https://dl.acm.org/doi/10.1145/1541880.1541882

[5] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

[6] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge: MIT Press, [2016]. Adaptive computation and machine learning (MIT Press). ISBN 978-0262035613.

[7] MURPHY, Kevin P. *Machine Learning: A Probabilistic Perspective*. Cambridge: The MIT press, 2012. ISBN 978-0262018029.

[8] ILYAS, Ihab F. a Xu CHU. *Data Cleaning*. Association for Computing Machinery, 2019. DOI: 10.1145/3310205. ISBN 9781450371520. **//doplnit vydavatele a místo vydání**

[9] SHARMA, Sagar. Activation Functions in Neural Networks. Towards Data Science [online]. 6. 9. 2017 [cit. 2020-10-11]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[10] DOLEŽEL, Petr. *Úvod do umělých neuronových sítí*. Pardubice: Univerzita Pardubice, 2016. ISBN 978-80-7560-022-6.

[11] OLAH, Christopher. Understanding LSTM Networks. Home – colah's blog [online]. August 27, 2015 [cit. 2020-10-30]. Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

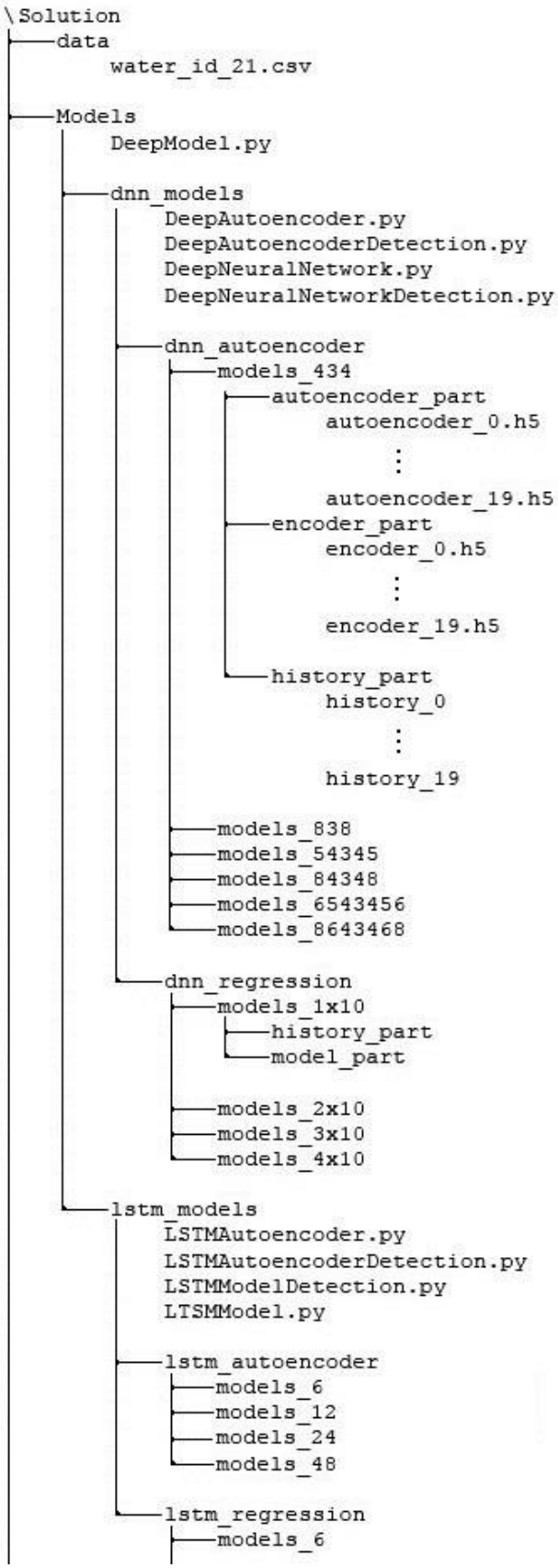
[12] VAN WYK, Andrich. Encoding Cyclical Features for Deep Learning. *Avanwyk* [online]. Pretoria, April 13, 2018 [cit. 2020-11-29]. Dostupné z: https://www.avanwyk.com/encoding-cyclical-features-for-deep-learning/

[13] MinMaxScaler [online]. scikit-learn, 2020 [cit. 2020-12-06]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[14] Project Jupyter [online]. Austin: NumFOCUS, 2021 [cit. 2021-5-7]. Dostupné z: https://jupyter.org/

PŘÍLOHY

Příloha A – Adresářová struktura řešení I. část



Příloha B – Adresářová struktura řešení II. část