VYSOKÁ ŠKOLA
CHEMICKO-TECHNOLOGICKÁ
V PRAZE

Fakulta chemicko-inženýrská

Ústav počítačové a řídicí techniky

Rekurentní neuronové sítě pro modelování časových řad

Semestrální projekt oboru Senzorika a kybernetika v chemii I

VYPRACOVAL	**Martin Vejvar**
VEDOUCÍ	**Prof. Ing. Jan Náhlík CSc.**
STUDIJNÍ PROGRAM	Procesní inženýrství a informatika
STUDIJNÍ OBOR	Senzorika a kybernetika v chemii
ROK	2018

1 Úvod

Práce se zabývá studiem a vývojem modelů rekurentních neuronových sítí pro predikci časových řad. První koncept rekurentních neuronových sítí s LSTM architekturou byl publikován již v roce 1997 [1]. V posledních letech se značně začíná zvyšovat zájem o jejich praktické využití pro zpracování rozmanitých sekvenčních dat zejména díky levné a veřejně dostupné výpočetní síle, rozvoji grafických karet a distribuovaným (cloudovým) výpočetním serverům. Cílem teoretické části práce je seznámení se s terminologií a problematikou rekurentních neuronových sítí, porozumění jejich funkci a aplikačním využitím. Praktická část zahrnuje návrh jednoduché rekurentní neuronové sítě s LSTM architekturou v programovacím jazyce Python3 s moduly NumPy a TensorFlow a otestování její funkčnosti pro jednokrokovou predikci zvolených časových řad.

2 Rekurentní neuronové sítě

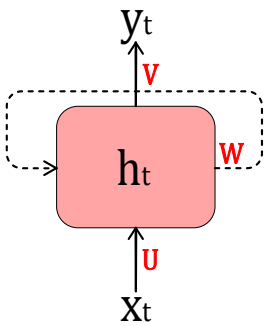
Rekurentní neuronové sítě (RNN) jsou speciální třídou umělých neuronových sítí specificky navrženou pro zpracování sekvenčních dat [2]. RNN zavádějí sdílení váhových koeficientů přes určitý počet (statický či dynamicky se měnící) časových (vzorkovacích) okamžiků, což jím umožňuje sledovat a uchovávat závislosti mezi vzorky sekvenčních dat. Jsou tedy schopny si "pamatovat"minulé události v datech a počítat s nimi při generování dat nových.

2.1 Rekurentní buňka

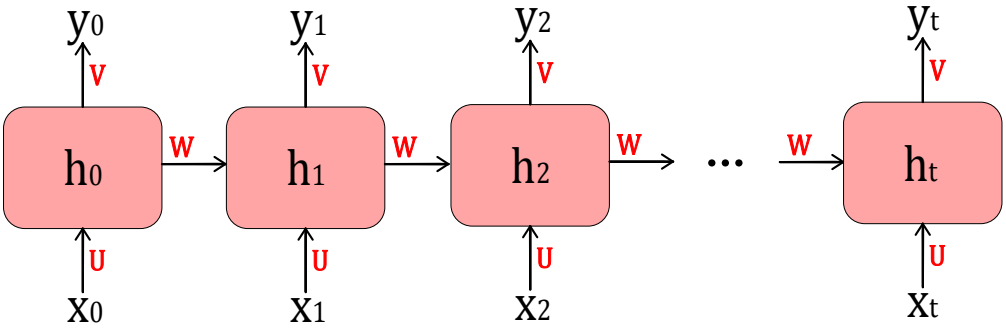
Elementárním prvkem RNN je buňka se zpětnou vazbou (recurrent cell), která v časovém okamžiku t parametrizuje vstup x_t přes matici váhových koeficientů U na skrytý stav h_t. Ze stavu h_t následně násobením maticí váhových koeficientů V generuje výstup y_t a přes matici váhových koeficientů W vypočítává výstupní stav, který je dále posílán na vstup buňky v dalším časovém okamžiku $t+1$ (viz obrázek 1). Skrytý stav h_t a výstup y_t tudíž nejsou závislé pouze na aktuálním vstupu x_t, ale zároveň na stavech buňky (a tedy i vstupech) z předchozích časových okamžiků. Funkční závislost skrytého stavu buňky v časovém okamžiku t lze tedy vyjádřit následovně:

$$h_t = f(x_t, h_{t-1}, h_{t-2}, ...h_0) \tag{1}$$

Rekurentní buňka v nerozvinuté podobě (obr. 1) předává vždy svůj stav v čase t (h_t) na vstup v následujícím časovém okamžiku $t+1$. Je tedy závislá na všech předchozích hodnotách vstupní sekvence. Při rozvinutí smyčky do jednotlivých časových okamžiků lze buňku reprezentovat obrázkem 2. Paměť buňky vždy dosahuje až na úplný počátek vstupních dat, což je kapacitně i výpočetně náročné zejména pro sekvence s velkým množstvím vzorků.



Obrázek 1: Nerozvinutá rekurentní buňka s maticí vstupních váhových koeficientů U, maticí výstupních váhových koeficientů V a maticí váhových koeficientů skrytého stavu W.

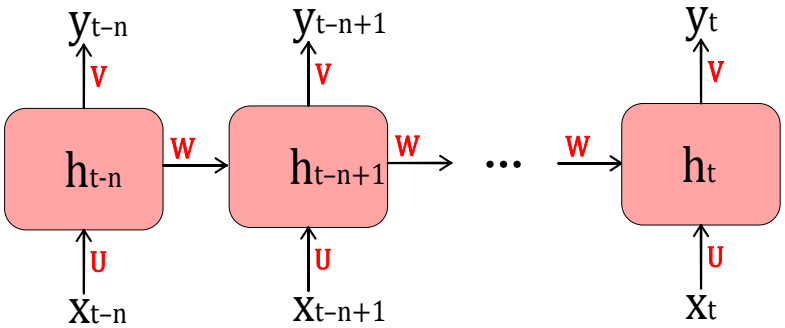


Obrázek 2: Rozvinutá rekurentní buňka s maticí vstupních váhových koeficientů U, maticí výstupních váhových koeficientů V a maticí váhových koeficientů skrytého stavu W.

Aby bylo zamezeno přílišné paměťové a výpočetní náročnosti, omezuje se rozvinutí rekurentní buňky na daný počet kroků do minulosti n. Všechny závislosti vzdálenější od buňky více než n časových okamžiků již buňka nevidí (viz obrázek 3). Funkční závislost skrytého stavu rekurentní buňky v okamžiku t je poté upravena následovně:

$$h_t = f(x_t, h_{t-1}, h_{t-2}, ...h_{t-n}) \tag{2}$$

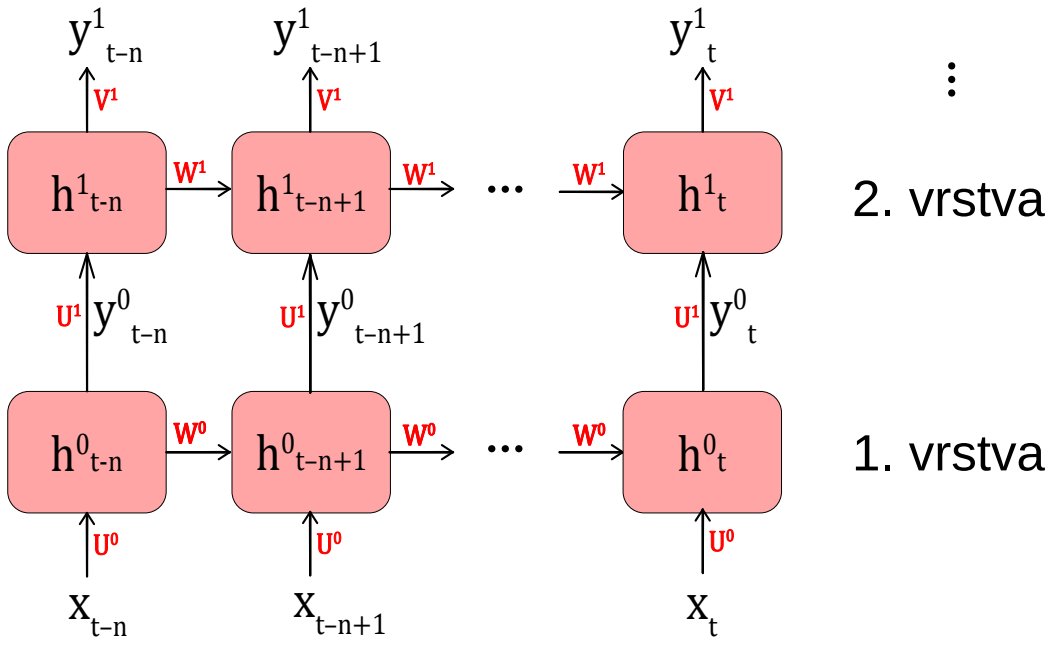
kde n může být dáno konstantou (staticky) nebo se může v čase dynamicky měnit.



Obrázek 3: Rozvinutá rekurentní buňka omezená na n hodnot do minulosti s maticí vstupních váhových koeficientů U, maticí výstupních váhových koeficientů V a maticí váhových koeficientů skrytého stavu W.

2.2 Zvýšení efektivity sítě

Samotná rekurentní buňka v rozvinuté podobě charakterizuje základní Rekurentní neuronovou síť. Pro zvýšení efektivity jsou zaváděny různé úpravy této základní verze mezi které patří např. zvýšení hloubky (přidání dodatečných vrstev) mezi vstupy a skryté stavy, mezi stavy a výstupy nebo mezi přechody stavů do dalšího časového okamžiku [3]. Dalším rozšířeným a jednoduše implementovatelným přístupem ke zvýšení komplexity sítě, který se ukázal být velmi efektivní (viz např. [4]), je vrstvení (stacking) několika rekurentních buněk nad sebe tím, že na vstup vyšší vrstvy x_t^1 je posílána hodnota výstupu buňky z vrstvy předchozí y_t^0 (viz obrázek 4). Takovéto struktury jsou označovány jako vrstvené rekurentní neuronové sítě (stacked RNN) [3].

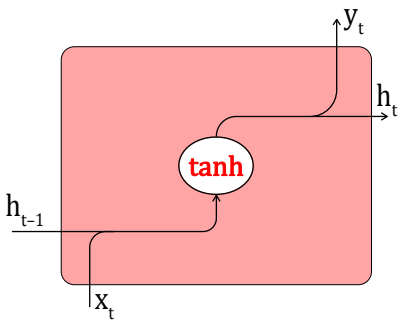


Obrázek 4: Vrstvená rekurentní neuronová síť s 2 vrstvami.

2.3 Long Short-Term Memory

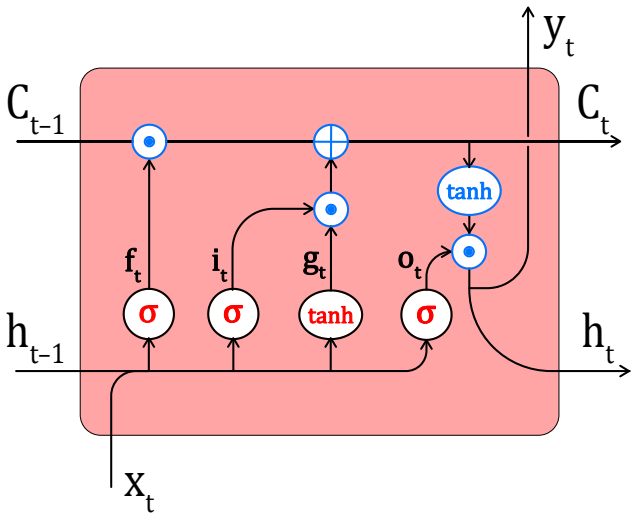
Dalším způsobem zvýšení efektivity sítě je celková změna vnitřní architektury rekurentní buňky. Mezi nejúspěšnější z architektur rekurentních buněk patří LSTM (Long Short-Term Memory) model, který byl navržen jako řešení problému mizejícího gradientu (vanishing gradient problem) [5].

Problém mizejícího gradientu nastává při trénování velmi hlubokých neuronových sítí, mezi které se řadí i rekurentní neuronové sítě se standardní architekturou rekurentní buňky (viz obrázek 5) a velkým počtem kroků rozvinutí do minulosti, gradientními optimalizačními metodami (Gradient descent, ADAGrad, Adam, atp. [6]) zejména při využití přenosových funkcí sigmoidální $\sigma(x)$ či hyperbolické tangenty $tanh(x)$. Problém lze stručně charakterizovat tak, že gradienty vypočtené na základě optimalizační funkce, která je složena z násobení nízkými hodnotami váhových koeficientů z jednotlivých vrstev, nabývají velmi nízkých hodnot, což vede k nedostatečné aktualizaci koeficientů sítě. Důsledkem je nutnost vysokého počtu trénovacích epoch než je dosaženo optimálních váhových koeficientů. Více o problému mizejícího gradientu při trénování rekurentních neuronových sítí viz např. [7].



Obrázek 5: Vnitřní architektura standardní rekurentní buňky.

LSTM model navržený Hochreiterem and Schmidhuberem v roce 1997 [1] zavádí speciální vnitřní architekturu buňky umožňující průchod gradientů velkým množstvím časových kroků bez jeho mizení (gradient jde k nule) či naopak explodování (gradient jde k nekonečnu) [5]. Stěžejním prvkem LSTM buňky, zvané též paměťová buňka, je vnitřní stav (internal cell state, resp. paměť) C_t, který je upravován pouze základními interakcemi a jeho hodnota může procházet buňkou, aniž by byla měněna, čímž je velikost gradientu při trénování zachována. Dalším důležitým prvkem buňky jsou tzv. brány (gates), které jsou charakteristické sigmoidální aktivační funkcí s výstupem v intervalu $(0, 1)$ sloužící k propuštění či naopak zamezení průchodu dat z některých ostatních částí buňky. Jsou obvykle značeny řeckým písmenem σ (sigma) v kruhovém ohraničení.



Obrázek 6: Vnitřní architektura Long Short-Term Memory rekurentní paměťové buňky ($\odot$ značí Hadamardův součin neboli součin po složkách [8]). Červeně jsou označeny dopředné vrstvy s příslušnou aktivační funkcí na výstupu. Modře jsou označeny operace, které jsou realizovány po složkách.

Architektura běžné LSTM buňky, jak je vyobrazena na obr. 6, se skládá z následujících elementů:

- **Brána zapomínání (f_t):** Určuje, která data z předchozího vnitřního stavu C_{t-1} mají být zachována a která zapomenuta. Buňka se tak může zbavit dat, která nejsou pro řešení daného problému relevantní a více se zaměřit na důležité úseky. Tato brána nebyla součástí originálního návrhu od Hochreitera and Schmidhubera [1]. Byla přidána až v roce 2000 ve zprávě *Learning to Forget* od Felixe A. Gerse a spol., kde bylo prokázáno, že její zavedení umožňuje řešení komplexnějších problémů [9]. Je možno ji popsat následujícím vztahem [10]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

- **Vstupní uzel (g_t):** Upravuje násobením maticí váhových koeficientů W_g a průchodem *tanh* aktivační funkcí vstupy z aktuálního časového okamžiku x_t a skrytý stav buňky z předchozího časového okamžiku h_{t-1}. Charakterizuje data, která mají být uložena do aktuálního vnitřního stavu C_t a lze ho popsat následující rovnicí [10]:

$$g_t = \tanh\left(W_g \cdot [h_{t-1}, x_t] + b_g\right) \tag{4}$$

- **Vstupní brána (i_t):** Určuje, která data ze vstupního uzlu g_t mají být použita pro aktualizaci vnitřního stavu buňky C_t. Obdobně jako brána zapomínání f_t, dovoluje buňce pomocí matice váhových koeficientů W_i různým částem dat dávat různou váhu nebo je zcela ignorovat. Vztah pro tuto bránu je ekvivalentní bráně zapomínání f_t s odlišnou maticí váhových koeficientů [10]:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{5}$$

- **Výstupní brána (o_t):** Definuje, která data z vnitřního stavu buňky C_t mají být odeslána na aktuální výstup y_t a skrytý stav h_t. Vztah je opět obdobný ostatním branám [10]:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{6}$$

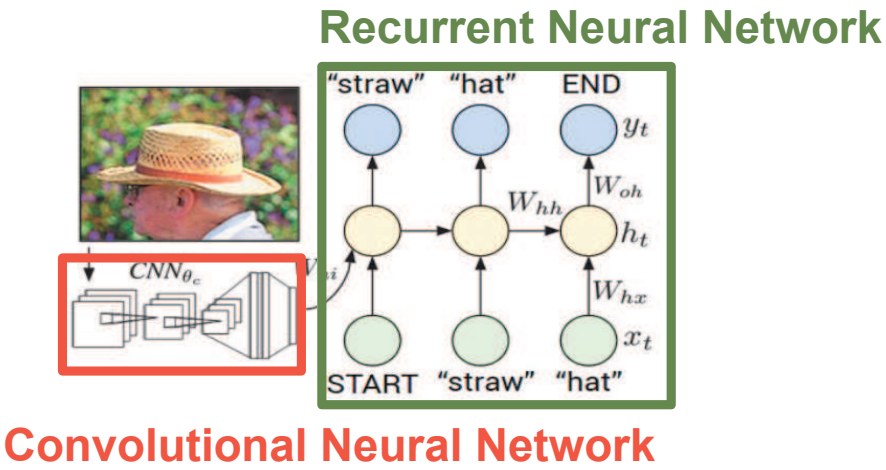
- **Výstup y_t (resp. skrytý stav h_t) buňky:** Je dán průchodem upraveného vnitřního stavu (C_t) tanh aktivační funkcí, jejíž výstup je navíc násoben výstupem brány o_t pro možnost odstranění (resp. snížení váhy) některé části, která je pro výstup nepodstatná. Výstup y_t (skrytý stav h_t) LSTM buňky v časovém okamžiku t lze tedy definovat vztahem [10]:

$$\begin{aligned} y_t &= o_t \odot \tanh(C_t) \\ C_t &= f_t \odot C_{t-1} + i_t \odot g_t \end{aligned} \tag{7}$$

  kde $\odot$ značí Hadamardův součin neboli součin po složkách [8].

2.4 Aplikace RNN

Rekurentní neuronové sítě mají díky svým vlastnostem velké množství aplikačních možností. Jejich využití je zejména vhodné v případech, kdy zpracovávaná data vykazují autokorelaci (vzájemná závislost mezi hodnotami vzorků dat) či časové závislosti (např. trendy a sezónní složky) a je nutno s nimi počítat jako např. při predikci časových řad. Zároveň RNN umožňují sekvenční data generovat. Příkladem může být popis obrázků pomocí sekvence slov (image captioning), kdy je cyklicky zpracováván výstup z konvoluční neuronové sítě (CNN) a v každém kroku je přidáno jedno slovo v závislosti na aktuálním výstupu CNN a již vygenerovaných slov z předchozích kroků (viz obrázek 7). Obdobným způsobem lze tuto architekturu využít i pro popis videosekvencí, kdy jsou na vstup CNN postupně posílány jednotlivé snímky sekvence. Více o aplikaci CNN a RNN pro popis obrázků [11] a pro popis videí [12].



Obrázek 7: Aplikace CNN a RNN pro popis obrázků [13].

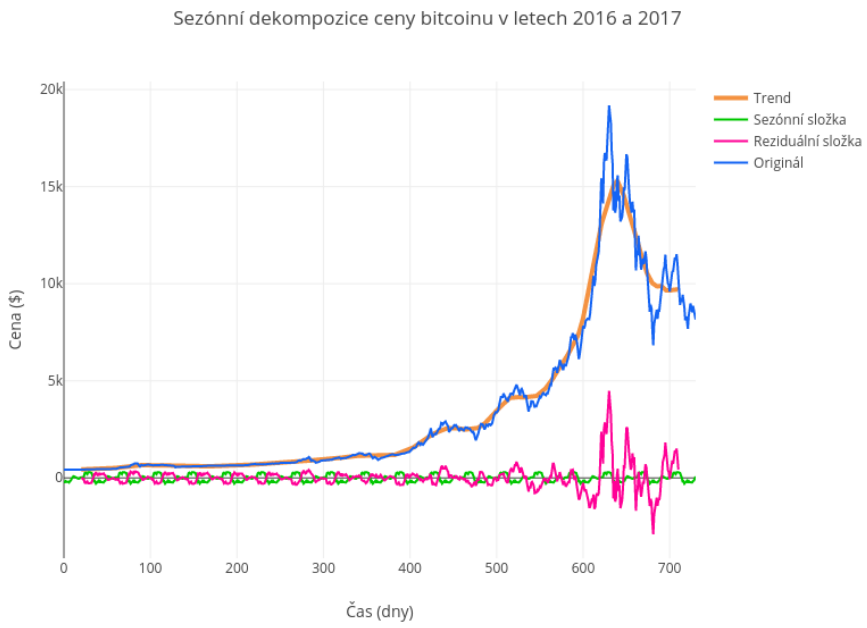
Velkým aplikačním odvětvím RNN je práce s textem. Význam slov a vět je značně podmíněný kontextem, ve kterém se vyskytují. Pro správné pochopení kontextu je nutno pracovat s textem jako sekvencí vzájemně závislých slov či symbolů. RNN jsou tudíž pro zpracování a generování textu vhodnou volbou. Mezi základní aplikace lze řadit např. generování textu, který se syntakticky podobá dané předloze, čehož lze dosáhnout posíláním sekvence znaků ze zdrojového textu na vstup sítě a požadovat při trénování na výstupu stejnou sekvenci znaků, avšak o jeden krok posunutou [14]. Tento postup principiálně odpovídá trénování RNN pro predikci časových řad. Dále je za pomocí RNN v tomto odvětví možno realizovat např. překlad textu z jednoho jazyka do jiného, kde se využívá architektury *Encoder-Decoder*, která se skládá ze dvou RNN. První RNN, Enkodér, převádí (enkóduje) vstupní sekvenci (např. text v anglickém jazyce) na určitou matematickou reprezentaci (vektor hodnot o fixní délce [15]). Druhá RNN, Dekodér, je naučena převádět

(dekódovat) tuto matematickou reprezentaci zpět na sekvenci slov (např. text v českém jazyce). Více o Encoder-Decoder architektuře např. zde [15]. Na této architektuře je postaven například Google systém pro strojový překlad textu (*Google's Neural Machine Translation System*), který se skládá z 8 vrstvého LSTM enkodéru a 8 vrstvého LSTM dekodéru [16]. Obdobně lze Encoder-Decoder architekturu využít pro vedení konverzace generováním odpovědí na základě aktuálních i minulých položených otázkách a odpovědích [17] či pro předpověď výstupu programu na základě vstupního zdrojového kódu [18].

Jak již bylo zmíněno, RNN lze efektivně využít pro predikci časových řad (*time series prediction* resp. *time series forecasting*). RNN, zejména s LSTM architekturou, mohou v časových řadách nalézt dlouhodobé závislosti, mezi které patří:

- **Trend:** Dlouhodobý pohyb nebo tendence v datech. Např. nárůst či pokles hodnot v průběhu určitého období.
- **Sezónní složka:** Periodická odchylka od trendové složky jejíž perioda je menší než celkový časový interval sledovaného období.
- **Náhodná (resp. reziduální) složka:** Náhodné kolísání časové řady, které zbývá po odstranění ostatních složek.

Pro názornost byla dle postupu [19] provedena sezónní dekompozice denního vývoje cen známé krypto-měny Bitcoin v průběhu let 2016 a 2017, kterou lze pozorovat na obrázku 8. Pokud je k dispozici dostatečné množství trénovacích dat, pak je kvalitně navržená rekurentní síť schopna tyto složky detekovat a predikovat jejich budoucí vývoj [19].



Obrázek 8: Sezónní dekompozice denního vývoje cen Bitcoinu v letech 2016 a 2017.

Mezi další aplikace RNN patří např. skládání hudby [20], rozpoznávání hlasu [21] nebo predikce epitopů B-lymfocytů (B buněk) v sekvenci antigenů [22].

3 Hyperparametry neuronových sítí

Pro dosažení žádoucích výsledků je při trénování a optimalizaci neuronových sítí důležité správné nastavení tzv. hyperparametrů. Mezi hyperparametry lze řadit veškeré parametry sítě, které nemohou být sítí automaticky odhadnuty z dat a jsou běžně manuálně nastavovány návrhářem sítě

[23]. Jedná se o parametry, které charakterizují strukturu sítě (např. počet vrstev, počet neuronů ve skryté vrstvě, volba aktivační funkce) a chování sítě při trénování (např. míra učení, počet trénovacích epoch). Mezi důležité hyperparametry rekurentních sítí, popsaných v kapitole 2, lze řadit:

- **Počet vrstev sítě**: Počet vertikálně vrstvených rekurentních buněk (viz obrázek 4).
- **Počet rekurentních jednotek buňky**: Určuje počet elementů ve vektoru, který uchovává vnitřní stav C_t LSTM buňky a tudíž i počet elementů vektoru skrytého stavu h_t a společně s velikostí vstupu v jednom časovém okamžiku x_t udává i velikost váhových matic (W_f, W_g, W_i, W_o) a vektorů posuvů (b_f, b_g, b_i a b_o) v branách sítě. Větší počet elementů umožňuje buňce realizovat komplexnější funkční závislosti mezi vstupy a výstupy a uchovávat v paměti více informací o datech na úkor rychlosti trénování. S rostoucí velikostí vnitřního stavu také roste riziko přeučení sítě (overfitting) na trénovacích datech.
- **Počet epoch**: Počet opakování (iterací) procesu učení sítě na trénovacích datech. Jedna epocha učení odpovídá jednomu průchodu celých trénovacích dat sítí. Příliš nízký počet epoch vede k nedoučení sítě (underfitting). Příliš vysoký počet epoch vede naopak k přeučení sítě (overfitting). Běžnou praktikou je tzv. *early stopping*, neboli zastavení procesu učení ve chvíli, kdy se za několik posledních epoch snižuje přesnost sítě (zvyšuje se hodnota optimalizační funkce) na testovacích (resp. validačních) datech [24].
- **Rozvinutí do minulosti (unrolling)**: Jak již bylo zmíněno v kapitole 2, jedná se o počet minulých časových okamžiků vstupních dat, ze kterých buňka počítá svůj stav a výstup v čase t (viz obrázek 3). S rostoucí délkou tohoto parametru je buňka schopna provádět předpovědi z delších úseků dat a lépe tak vystihnout dlouhodobé závislosti.
- **Účelová funkce (cost function resp. loss function)**: Funkce, která charakterizuje odchylku výstupů sítě od požadovaných hodnot výstupů. Cílem učení neuronové sítě je minimalizace této funkce úpravou hodnot váhových koeficientů. Volba účelové funkce je závislá na konkrétní aplikaci (tvaru vstupních a výstupních dat, architektuře sítě apod.). Mezi běžně používané účelové funkce lze řadit např. střední kvadratickou odchylku (*mean square error*, MSE), odmocninu ze střední kvadratické odchylky (*root mean square error*, RMSE) nebo křížovou entropii (cross-entropy), kterou lze popsat vztahem:

$$L_c = -\sum_{i=1}^{N} t_i \log y_i \tag{8}$$

  kde t_i jsou elementy vektoru požadovaných hodnot, y_i elementy výstupního vektoru sítě a N je počet elementů těchto vektorů.

  Křížovou entropii je vhodné využít v případě, kdy výstupem sítě je vektor pravděpodobností. Příkladem může být klasifikace vstupních dat do N tříd. Více o typech účelových funkcí např. zde [25].
- **Optimalizační algoritmus**: Určuje, jakým způsobem jsou z účelové funkce vypočteny hodnoty pro aktualizaci matic váhových koeficientů v průběhu učení sítě. Jsou založeny na výpočtu gradientu účelové funkce a provedení kroku (úprava hodnot váhových koeficientů) ve směru opačném než je tento směr nejvyššího růstu, čímž se algoritmus blíží k minimu účelové funkce. Mezi běžné optimalizační algoritmy patří např. *Gradient descent*, *Stochastic gradient descent* (SGD), *AdaGrad*, *RMSprop* nebo *Adaptive moment estimation* (Adam) [6].
- **Míra učení (learning rate)**: Upravuje velikost změn váhových koeficientů, kterou zavádí optimalizační algoritmus při učení sítě. Při příliš vysoké míře učení není optimalizační algoritmus schopen dosáhnout minima účelové funkce, jelikož váhové koeficienty jsou upravovány příliš agresivně a hodnoty účelové funkce oscilují kolem minima. Nízká hodnota účelové funkce naopak vede k nedostatečné aktualizaci váhových koeficientů a nutnosti vysokého počtu epoch pro dosažení minima účelové funkce. Z principu učení neuronových sítí je výhodné míru učení v průběhu epoch snižovat, aby se hodnota účelové funkce mohla více přiblížit k minimu. Některé optimalizační algoritmy (např. *AdaGrad*, *RMSprop* nebo *Adam*) míru učení adaptivně upravují. Pro ostatní algoritmy (např. SGD) je zaváděno globální snižování míry učení v průběhu epoch (např. exponential decay). Více o klesající míře učení (decaying learning rate) např. zde [26].

- **Velikost dávky (batch size)**: Z důvodů optimalizace paměťové a výpočetní náročnosti se jednotlivé vzorky vstupních dat posílají do sítě tzv. po dávkách. Dávka je sada vzorků vstupních dat, kterou síť zpracovává najednou. K optimalizaci váhových koeficientů dochází až po průchodu všech vzorků obsažených v dávce síťí. S rostoucí velikostí dávky se tedy zpravidla zvyšuje rychlost trénování sítě, jelikož roste perioda aktualizace váhových koeficientů, což při přílišné velikosti dávky může vést k celkovému snížení přesnosti sítě. Zvýšením velikosti dávky se však také zvyšuje hodnota účelové funkce a snižuje se rozptyl gradientů účelové funkce sloužících pro aktualizaci váhových koeficientů, jelikož jejich hodnota se počítá z většího množství vzorků a je tedy více stabilní. Důsledkem zmíněných jevů je možnost provádění agresivnějších (větších) kroků k minimu účelové funkce, avšak s nižší frekvencí. S vyšší velikostí dávky je tedy možno předpokládat vyšší množství trénovacích epoch potřebných pro naučení sítě s řádově kratším trváním jedné epochy. Okrajovým případem je velikost dávky odpovídající počtu trénovacích dat, kdy se gradient účelové funkce průměruje z celé sady trénovacích dat a k aktualizaci váhových koeficientů dochází jen jednou za epochu učení. Ve článku *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour* [27] je podrobně prozkoumán problém optimalizace váhových koeficientů při vysokých velikostech dávky, který se projevuje zejména v počátečních epochách učení. Pokud je však tento problém vyřešen (např. zavedením nízké velikosti dávky na několik počátečních epoch), vykazuje síť lepší generalizační schopnosti a dosahuje tedy celkově nižších hodnot účelové funkce validačních dat za kratší dobu než při malé velikosti dávky.
- **Dropout**: Jedná se o techniku běžně využívanou pro zabránění přeučení (overfitting) hlubokých neuronových sítí, zejména při malém množství trénovacích dat [28]. Spočívá v aplikaci masky na vstupy, výstupy či vnitřní stavy při učení sítě, která způsobí vynulování určitého procenta z nich (náhodně vybraných). Síť se tím učí pracovat s neúplnou reprezentací dat a je v nich schopna kvalitněji detekovat obecné závislosti.

4 Praktická část

Pro utvrzení znalostí o RNN a demonstraci jejich možností v oblasti predikce časových řad byla navržena rekurentní neuronová síť s LSTM architekturou buněk a volitelnými hyperparametry (viz kapitolu 3). K návrhu byl použit programovací jazyk Python (verze 3.6) s dodatečnými moduly pro matematické výpočty (zejména modul NumPy ze sady vědecko-výzkumných numerických nástrojů SciPy [29]). Pro definici struktury neuronové sítě a její výpočet byl využit Pythonovský modul knihovny TensorFlow [30], který zprostředkovává návrh a efektivní numerické výpočty matematických modelů. Program byl vytvořen ve webovém programovacím prostředí Jupyter Notebook [31]. Všechny zmíněné balíčky a prostředí byly nainstalovány prostřednictvím platformy Anaconda, otevřené distribuce Pythonu určené pro nauku o datech. Realizované programy a výsledky pokusů společně s elektronickou verzí zprávy jsou uloženy a volně k dispozici v online repositáři webové služby GitHub pod adresou `https://github.com/vejvarm/RNNs`. Aktuální verze hlavního programu k datu této zprávy je také vložena v příloze A.

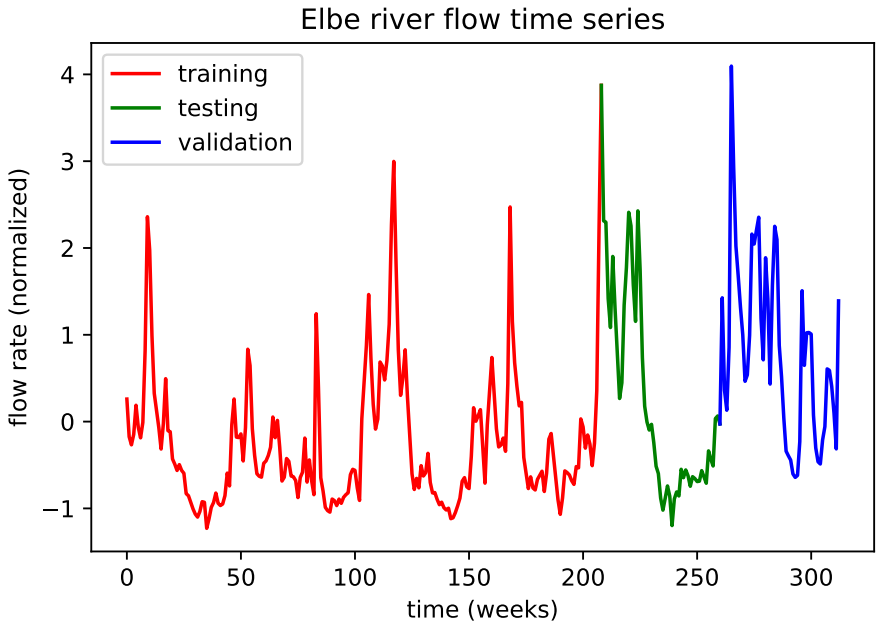
4.1 Datové sady

Trénování a testování sítě bylo prováděno na dvou časových řadách:

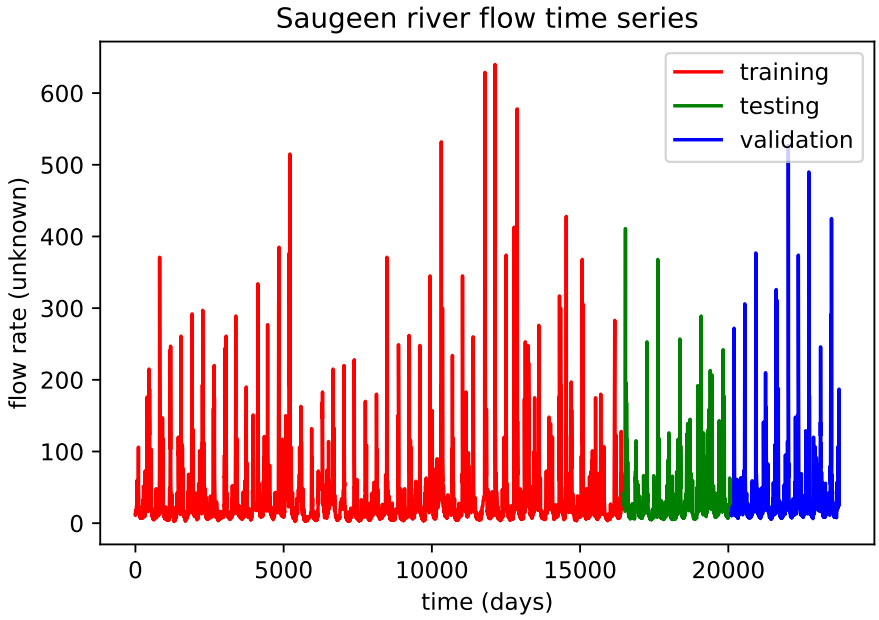
1. Průměrný týdenní průtok řeky Labe v průběhu 6.let (celkem 313 vzorků) poskytnutých prof. Ing. Alešem Procházkou, CSc
2. Průměrný denní průtok řeky Saugeen v období 1915 až 1979 (celkem 23741 vzorků) z databáze *Time Series Data Library* [32]

Obě časové řady byly rozděleny na trénovací, testovací a validační sadu v poměru (70%:15%:15%) a vykresleny do grafů na obrázcích 9 (týdenní průměr průtoků Labe) a 10 (denní průměr průtoků Saugeen). Toto rozdělení je zavedeno z důvodu možnosti otestování schopnosti sítě predikovat hodnoty časové řady, která nebyla součástí trénovací sady. Testovací část časové řady slouží pro kontrolu kvality predikce v průběhu trénování a identifikaci okamžiku, kdy síť dosahuje stavu přeučení (indikováno snižováním přesnosti predikce a tudíž zvyšováním hodnoty účelové funkce

na testovací části dat). Validační část časové řady poté simuluje využití již naučené sítě v praxi. Aplikuje se po ukončení trénovacího procesu a slouží k určení výsledné přesnosti predikce naučenou sítí.



Obrázek 9: Časová řada průměrných týdenních průtoků řeky Labe (normalizovaných na nulovou střední hodnotou a jednotkový rozptyl) v průběhu 6. let rozdělená na trénovací, testovací a validační část.



Obrázek 10: Časová řada průměrných denních průtoků řeky Saugeen v období 1915 až 1979 rozdělená na trénovací, testovací a validační část.

4.2 Pevně nastavené hyperparametry sítě

Za účelovou funkci (*loss*) byla zvolena odmocnina ze střední kvadratické chyby (RMSE), která je popsána následujícím vzahem:

$$loss = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2} \tag{9}$$

kde N je počet vzorků, y_i jsou výstupy sítě (predikce hodnot časové řady) a t_i jsou požadované výstupy sítě (skutečné hodnoty časové řady). Její hodnota charakterizuje směrodatnou odchylku mezi predikovanými výstupy sítě a požadovanými výstupy sítě.

Pro dodatečné vyhodnocení přesnosti predikce časové řady byla zvolena symetrická střední hodnota relativní chyby predikce (Symmetric mean absolute percentage error, neboli SMAPE) definovaná následovně:

$$SMAPE = \frac{2}{N}\sum_{i=1}^{N}\frac{|y_i - t_i|}{|y_i| + |t_i|} \tag{10}$$

jejíž výhodou oproti běžné střední relativní chybě (MAPE) je existence horní meze (více o MAPE a SMAPE např. zde [33]).

Pro výpočet úprav váhových koeficientů sítě v průběhu učení byl zvolen adaptivní gradientní optimalizační algoritmus *Adaptive moment estimation* (Adam), který byl publikován v roce 2014 ve článku *Adam: A Method for Stochastic Optimization* [34], kde bylo za jeho využití v úlohách strojového učení dosaženo příznivých výsledků vůči ostatním gradientním optimalizačním metodám. Často se proto využívá jako univerzální optimalizační algoritmus pro trénování rekurentních neuronových sítí. Mezi další při optimalizaci rekurentních sítí hojně využívané algoritmy lze řadit RMSProp a AdaGrad (více např. zde [6]). V této práci byl však testován pouze Adam.

Míra učení Adam optimalizačního algoritmu byla empiricky nastavena na hodnotu $0,001$ a v průběhu nastavování ostatních hyperparametrů již nebyla měněna.

Pro zabránění přeučení sítě na datech bylo implementováno předčasné zastavení učení sítě (early stopping) v případě, že hodnota účelové funkce (9) při predikci testovacích dat vykazovala po zvolený počet epoch neklesající charakter.

4.3 Jednokroková predikce

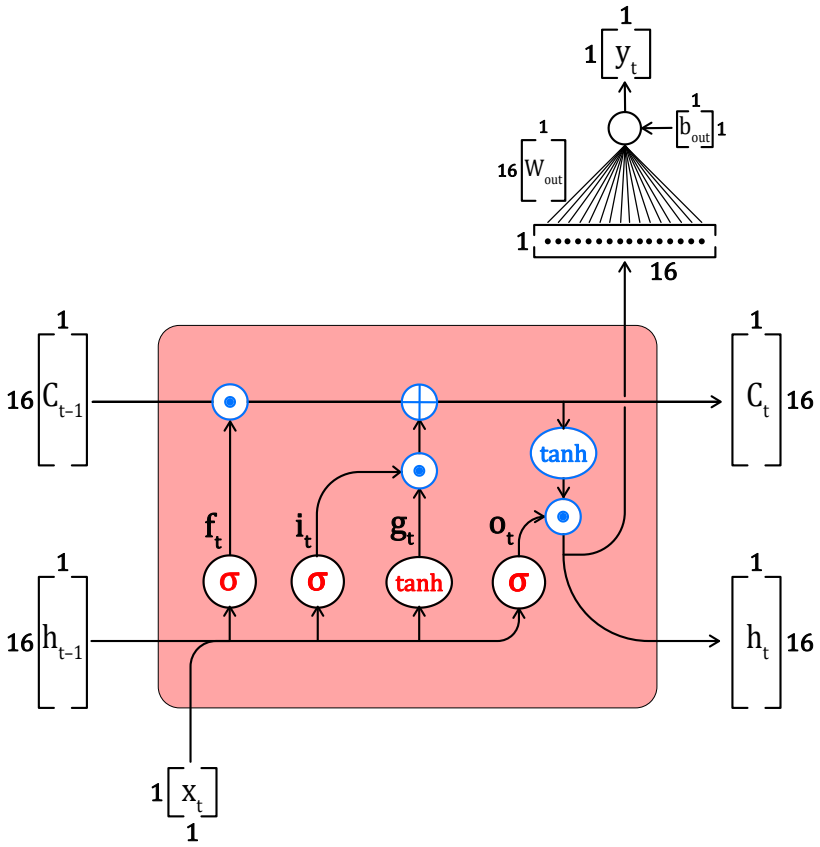
Praktická část je rozdělena na jednokrokovou predikci (výpočet jednu vzorkovací periodu do budoucnosti) pro velkou datovou sadu (průtoky řeky Saugeen) a malou datovou sadu (průtoky řeky Labe). Navržený program (příloha A) je schopen i vícekrokové predikce (výpočet o více jak jednu vzorkovací periodu do budoucnosti). Její testování a optimalizace je však náplní budoucí práce.

Jednokroková predikce spočívá ve výpočtu následující hodnoty sekvence z jedné či více hodnot předchozích. Jak pro týdenní průtok Labe (krátká sekvence), tak pro denní průtok Saugeen (dlouhá sekvence) byly testovány různé konfigurace hyperparametrů sítě (viz kapitoly 4.3.1 a 4.3.2). Základní hodnoty hyperparametrů sítě byly empiricky zvoleny vyhodnocením grafu a hodnot $SMAPE$ validačních dat průtoku řeky Saugeen a jsou shrnuty v tabulce 1. 14.

Tabulka 1: Základní zvolené hodnoty hyperparametrů sítě.

název	**hodnota**
počet vrstev LSTM buněk	2
počet rekurentních jednotek buňky	16
rozvinutí do minulosti	16
velikost dávky	4
maximální počet epoch	1000
epoch před zastavením kvůli neklesající účelové funkci	20
perioda kontroly účelové funkce (epoch)	10
dropout	*vypnuto*

Struktura LSTM buňky v čase t pro základní zvolené hyperparametry sítě v tabulce 1 je vyobrazena na obrázku 11. Výstup brány zapomínání f_t je pro tyto hyperparametry dán operací na obrázku 12. Výstupy ostatních bran se sigmoidální aktivační funkcí (vstupní brána i_t a výstupní brána o_t) jsou dány identickou operací s unikátními hodnotami váhových matic (W_i a W_o) a vektorů posuvů (b_i a b_o). Výstupní vektor vrstvy s tanh aktivační funkcí je získán dle vztahu na obrázku 13. Celkový výstup buňky y_t je dán průchodem skrytého stavu h_t buňky skrze běžnou dopřednou vrstvu s jedním neuronem, tedy násobením vektoru skrytého stavu h_t váhovou maticí W_{out} a přičtením posuvu b_{out}.



Obrázek 11: Struktura jedné LSTM buňky v síti se základními hyperparametry (viz tab. 1) s velikostí vstupů (x_t), stavů (h_t a C_t) a výstupů po průchodu lineární dopřednou vrstvou (y_t) v čase t ($\odot$ značí Hadamardův součin neboli součin po složkách [8]). Červeně jsou označeny dopředné vrstvy s příslušnou aktivační funkcí na výstupu. Modře jsou označeny operace, které jsou realizovány po složkách.

.

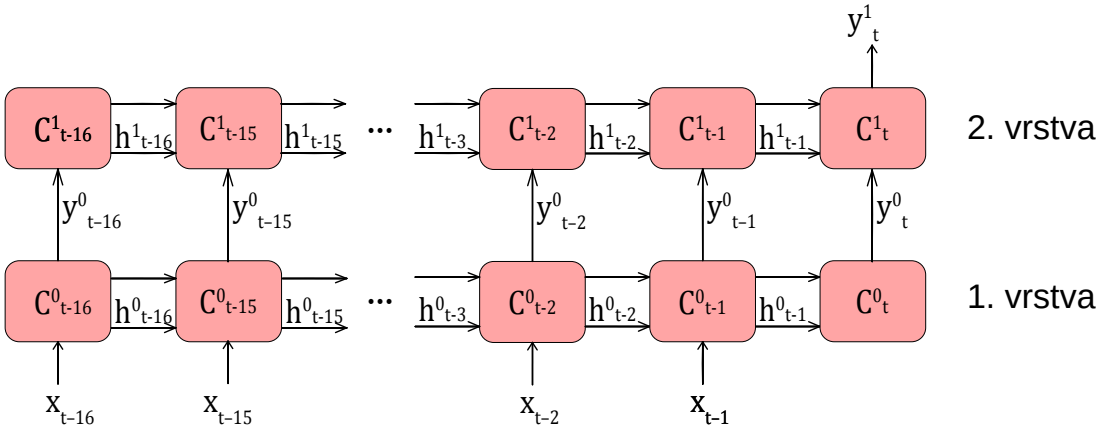
$$\overset{1}{\underset{16}{\left[f_t\right]}} = \sigma\left(\overset{17}{\underset{16}{\left[W_f\right]}} \cdot \overset{1}{\left[\begin{matrix} h_{t-1} \\ x_t \end{matrix}\right]_{17}} + \overset{1}{\underset{16}{\left[b_f\right]}}\right)$$

Obrázek 12: Výpočet výstupního sloupcového vektoru brány zapomínání f_t jehož počet elementů je roven počtu rekurentních jednotek buňky (velikosti vnitřního stavu C_t), což je v případě základních parametrů 16 (viz tab. 1).

$$\overset{1}{\underset{16}{\left[g_t\right]}} = \tanh\left(\overset{17}{\underset{16}{\left[W_g\right]}} \cdot \overset{1}{\left[\begin{matrix} h_{t-1} \\ x_t \end{matrix}\right]_{17}} + \overset{1}{\underset{16}{\left[b_g\right]}}\right)$$

Obrázek 13: Výpočet výstupního sloupcového vektoru vrstvy s tanh výstupem g_t jehož počet elementů je roven počtu rekurentních jednotek buňky (velikosti vnitřního stavu C_t), což je v případě základních parametrů 16 (viz tab. 1).

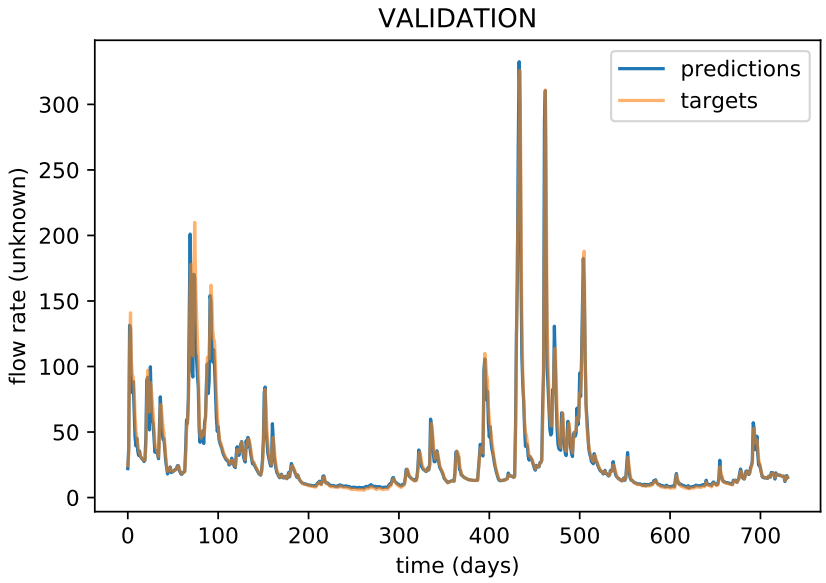
Celková struktura dle tab. 1 dvouvrstvé sítě s LSTM buňkami, které mají skalární výstup z dopředné vrstvy y_t (viz obr. 11), rozvinuté 16 hodnot do minulosti je poté vykreslena na obrázku 14.



Obrázek 14: Struktura v čase rozvinuté sítě se základními zvolenými hyperparametry (viz tab. 1), kde pro $i = \langle t-16 : t \rangle$ je C_i vektor vnitřního stavu buňky s 16 elementy, h_i vektor skrytého stavu buňky s 16 elementy, y_i výstupní hodnota buňky po průchodu dopřednou vrstvou s 1 neuronem s váhovými koeficienty W_{out} a posuvem b_{out} a x_i vstupní hodnota časové řady.

4.3.1 Průtok řeky Saugeen (velká datová sada)

Při trénování sítě na průtocích řeky Saugeen s hyperparametry v tabulce 1 bylo dosaženo nejnižší hodnoty účelové funkce v 70. epoše. Hodnota SMAPE při predikci validačních průtoků řeky Saugeen naučenou sítí byla $0,090773$, tedy cca. $9,1\,\%$. Porovnání predikovaných a skutečných hodnot pro tuto konfiguraci je možno pozorovat na obrázku 15.



Obrázek 15: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Saugeen v průběhu 2 let.

Následně byly dle poznatků v kapitole 3 upravovány hodnoty hyperparametrů a pozorován vliv změn na kvalitu predikce (hodnotu SMAPE validačních dat a epochu učení, při které bylo dosaženo nejnižší hodnoty účelové funkce). Výsledky jsou shrnuty v tabulce 2.

Nejprve byl měněn počet rekurentních jednotek buňky (délka vektoru vnitřního stavu buňky C_t). Z původní hodnoty 16 byla hodnota zvýšena nejprve na 32, dále na 64 a poté na 128. Tím by se měla zvýšit komplexnost funkčních závislostí, které je síť schopna realizovat. Z toho důvodu by síť měla umět lépe vystihnout trénovací data a dosáhnout minima hodnoty účelové funkce (i stavu přeučení) za menší množství trénovacích epoch. Proces trénování však bude výpočetně náročnější, jelikož je nutno optimalizovat větší množství váhových koeficientů sítě. Při porovnání výsledků (pokusy 0 1 2 a 3 v tab. 2) byly předpoklady potvrzeny. Zároveň bylo s rostoucím počtem rekurentních jednotek pozorováno mírné zvýšení hodnoty SMAPE validačních dat.

Dále byly prováděny změny délky rozvinutí do minulosti. Počáteční hodnota 16 časových okamžiků (dnů) do minulosti byla změněna na 30 (přibližně odpovídá jednomu měsíci dat), následně na 90 (jedno roční období) a posléze na 183 (půl roku). Hodnota SMAPE pro rozvinutí 30 a 183 dnů byla oproti původním datům cca. o 3 % vyšší. Při rozvinutí do minulosti odpovídající jednomu ročnímu období (90 dnů) byl však zaznamenán pokles hodnoty SMAPE o necelá 2 %.

Obdobně byla testována změna velikosti dávky s hodnotami 16, 64, 128, 256 a 512. Při vyšších hodnotách dávky se váhové koeficienty sítě aktualizují s nižší frekvencí (jednou za dávku), což vede ke znatelnému snížení výpočetní náročnosti trénování a tudíž i doby učení. Od příliš vysokých hodnot dávky lze však také očekávat snížení kvality predikce sítě (zvýšení hodnoty SMAPE). Dle výsledků pokusů 7, 8, 9, 10 a 11 v tabulce 2 je možno s rostoucí velikostí dávky pozorovat zvyšování počtu epoch potřebných k naučení sítě. Nejnižší hodnoty SMAPE bylo dosaženo při velikosti dávky 16 (7,9 %) a 64 (8,3 %). Při vyšších dávkách se SMAPE opět navrátilo do blízkého okolí 9 %. Lze tedy usoudit, že optimální velikost dávky sítě při základních hodnotách ostatních hyperparametrů se pohybuje v intervalu (16,64).

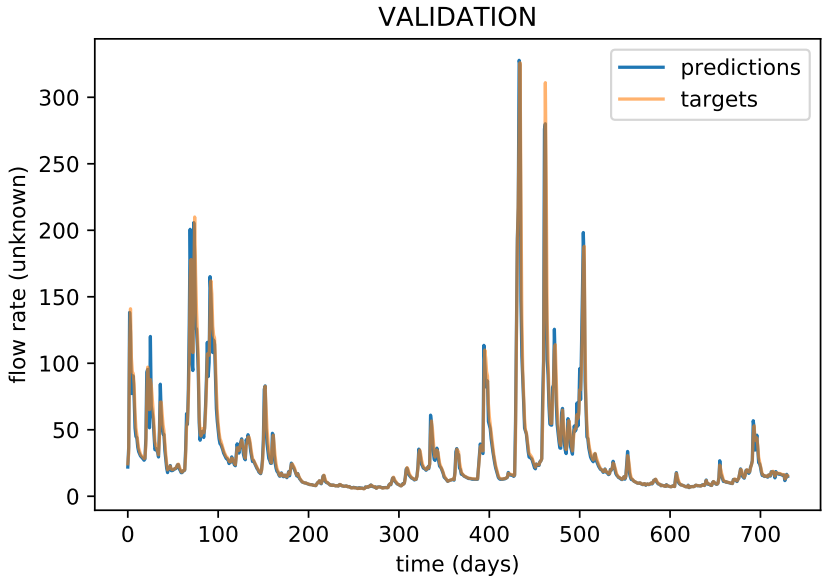
Následně byla provedena kombinace hyperparametrů, které individuálně vedly k nejnižším hodnotám SMAPE. Mezi tyto hodnoty hyperparametrů lze řadit 16 rekurentních jednotek, velikost dávky 16 a rozvinutí rekurentních buněk 90 dnů do minulosti. Pokus č. 12 v tabulce 2 zaznamenává nejlepší z několika provedených opakování pokusu s těmito hyperparametry. Hodnota SMAPE (8,3 %) se snížila vůči základnímu nastavení hyperparametrů o 0,8 %. V pokusu č. 13 v tabulce 2 byla velikost dávky zvýšena na 64, jelikož pro tuto hodnotu vykazovala hodnota SMAPE v předchozím pokusu (č. 8) přijatelné výsledky. SMAPE validačních dat se však ve zmíněné konfiguraci snížilo vůči základním hyperparametrům pouze o 0,13 %. Pozitivním přínosem je snížení doby

učení o 5 minut. V posledním pokusu (č. 14) bylo testováno, zda by síť byla schopna dosáhnout lepších výsledků než v pokusu č. 13 v případě vyššího množství rekurentních jednotek (aby měla více paměti pro možnost uchování dlouhodobých závislostí v průběhu 90 dnů). Výsledkem však bylo, obdobně jako při předchozích pokusech s množstvím rekurentních jednotek, snížení počtu epoch potřebných k naučení sítě a zvýšení SMAPE validačních dat o 0,48 %.

Tabulka 2: Výsledky jednokrokové predikce průtoku řeky Saugeen pro testované hodnoty hyperparametrů.

číslo pokusu	**rekur. jednotek**	**velikost dávky**	**rozvinutí do minulosti**	**epoch učení**	**SMAPE** (1)	**doba učení** (hod : min : s)
0	16	4	16	70	0,090773	0:23:43
1	32	4	16	40	0.095761	0:18:30
2	64	4	16	20	0,099052	0:16:19
3	128	4	16	20	0,107802	0:33:23
4	16	4	30	30	0,115417	0:24:19
5	16	4	90	30	0,088969	1:05:32
6	16	4	183	10	0,121210	1:13:45
7	16	16	16	40	0,078515	0:04:42
8	16	64	16	60	0,082546	0:01:52
9	16	128	16	60	0,090922	0:01:53
10	16	256	16	80	0,087125	0:01:59
11	16	512	16	100	0,091367	0:02:11
12	16	16	90	20	0,083098	0:24:53
13	16	64	90	70	0,089402	0:18:30
14	64	64	90	20	0,095532	0:17:44

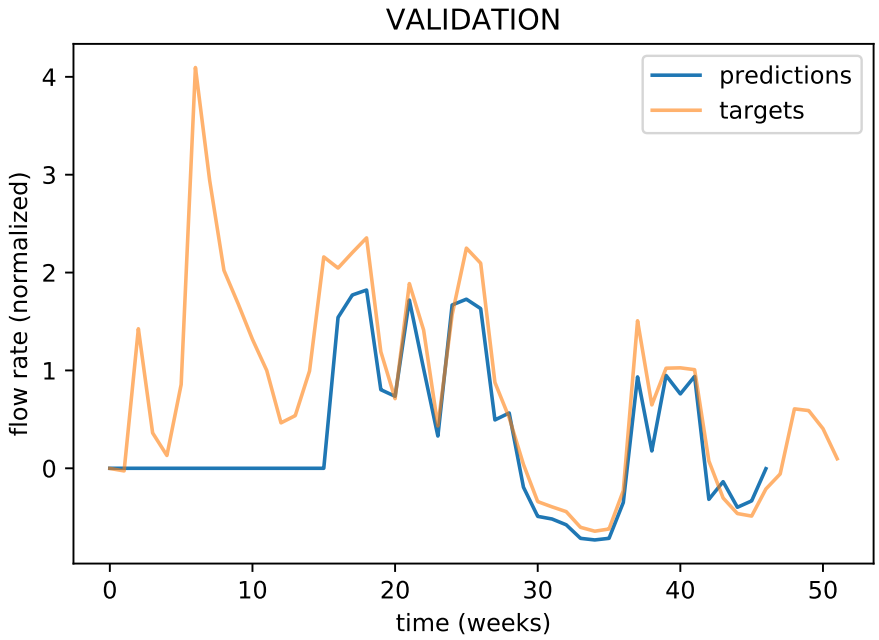
Nejlepších výsledků z hlediska hodnoty SMAPE bylo při jednokrokové predikci průtoků řeky Saugeen dosaženo pro 16 rekurentních jednotek, velikost dávky 16 a rozvinutí 16 dnů do minulosti v pokusu č. 7 s hodnotou 7,9 % (graf porovnání predikcí vůči skutečným hodnotám lze pozorovat na obrázku 16). Díky zvýšení velikosti dávky vůči základnímu nastavení hyperparametrů byla také značně snížena doba potřebná k naučení sítě z původních 24 minut na pouhých 5. Pro další snížení SMAPE a zrychlení procesu učení by mohlo být zavedeno zvýšení velikosti dávky po několika počátečních epochách učení, jak bylo zmíněno v kap. 3.



Obrázek 16: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Saugeen v průběhu 2 let při 16 rekurentních jednotkách, velikosti dávky 16 a rozvinutí 16 dnů do minulosti.

4.3.2 Průtok řeky Labe (malá datová sada)

Časová řada týdenních průtoků řeky Labe obsahuje pouze 313 vzorků, což je v oblasti trénování hlubokých sítí velmi malá sada. Lze tedy předpokládat, že výsledky predikce budou v porovnání s velkou datovou sadou (kap. 4.3.1) méně přesné, jelikož síť nemá k dispozici dostatek unikátních trénovacích vzorků na kterých by se mohla kvalitně naučit obecné závislosti v datech. Při nastavení stejných základních hyperparametrů jako při jednokrokové predikci průtoku řeky Saugeen (viz tab. 1) je hodnota SMAPE pro validační data $64,9\,\%$. Porovnání predikcí se skutečnými hodnotami pro tuto konfiguraci se nachází na obrázku 17.



Obrázek 17: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Labe v celém rozsahu dostupných hodnot.

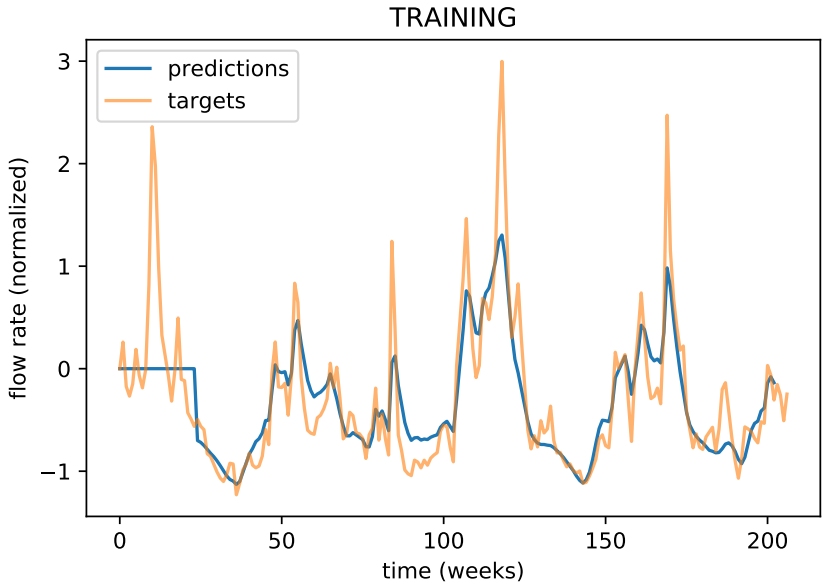
Predikce (a zároveň výpočet SMAPE) začíná až od 17. týdne, jelikož rozvinutí do minulosti je nastaveno na 16. týdnů a končí dříve, neboť je nastavena velikost dávky 4 a zbývají pouze 3 hodnoty. Tyto nedostatky by mohly být opraveny zavedením variabilního rozvinutí do minulosti pro počáteční hodnoty a proměnné velikosti dávky. V této práci však tyto úpravy nebyly zavedeny.

Následně byly, obdobně jako v předchozí kapitole (4.3.1), prováděny změny jednotlivých hyperparametrů a sledován vliv na hodnotu SMAPE, počet potřebných epoch pro naučení sítě a dobu trvání procesu učení. Jelikož je kvalita naučení sítě značně závislá na inicializaci váhových koeficientů, bylo každé unikátní nastavení hyperparametrů (pokus) sítě spuštěno 5 krát. Z těchto 5 instancí pokusu byla poté vybrána ta s nejnižší hodnotou SMAPE a uložena do tabulky 3 jako výsledek pokusu.

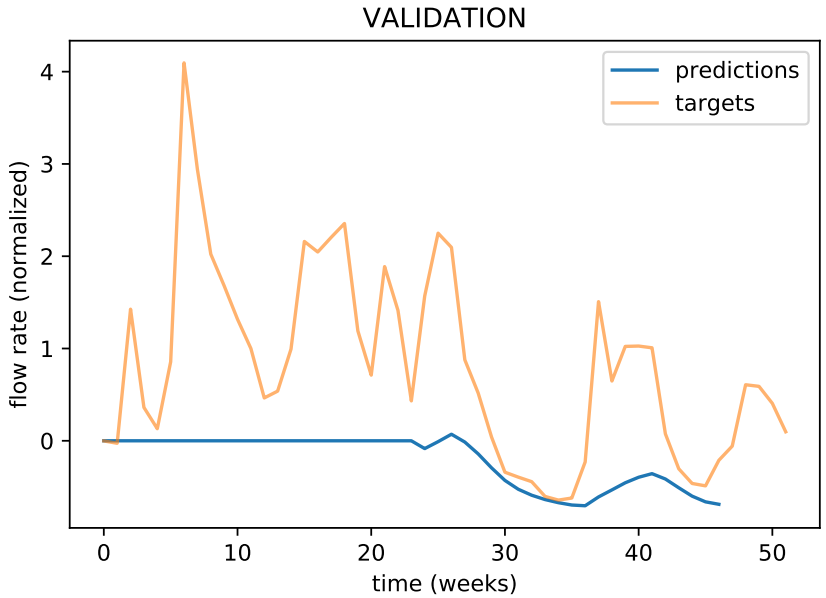
Prvním měněným parametrem byl počet rekurentních jednotek s hodnotami 4, 32, 64 a 128. V tabulce 3 tomu odpovídají výsledky pokusů 1 až 4. Opět lze s rostoucím počtem rekurentních jednotek pozorovat snížení potřebných epoch učení a zvyšování hodnot SMAPE. Nejnižší hodnoty SMAPE ($64,3\,\%$) bylo dosaženo pro 4 rekurentní jednotky výměnnou za delší dobu učení.

Rozvinutí do minulosti bylo testováno pro hodnoty 4 (jeden měsíc), 12 (jedno roční období) a 24 (půl roku) týdnů. Při vyšších hodnotách rozvinutí se snižoval počet epoch potřebných k naučení sítě. Příčinnou může být dostupnost vyššího množství historických dat, čímž by měl vnitřní stav buňky lépe reprezentovat průběh trénovacích dat. Zvyšováním rozvinutí se však také snižuje počet iterací v každé trénovací epoše, což při malé datové sadě může způsobovat snížení kvality naučení sítě a počet epoch učení naopak mírně zvyšovat. Nejlepších výsledků bylo dosaženo pro rozvinutí 4 hodnoty do minulosti s hodnotou SMAPE $63,2\,\%$ (o $1,72\,\%$ nižší než SMAPE při základních

hyperparametrech). Pro rozvinutí do 24 týdnů (pokus č. 7 z tabulky 3) již hodnota SMAPE značně narostla a v nejlepší z 6 iterací pokusu dosahovala hodnoty 119,0 %. Jedním z důvodů tohoto navýšení je pravděpodobně nedostatek validačních dat a jejich pohyb v hodnotách blízkých 0, což značně zvyšuje chybu počítanou vztahem SMAPE i při menších odchylkách. Hlavním důvodem, který je pozorovatelný na obrázku 18 je však neschopnost sítě se dostatečně naučit obecné závislosti v trénovacích datech než začne docházet k jejímu přeučení a předčasnému zastavení (early stopping). Při pohledu na graf porovnání predikcí se skutečnými hodnotami průtoku validačních dat při pokusu č. 7 se SMAPE 119,0 % lze vysokou odchylku jednoznačně pozorovat (viz obr. 19).

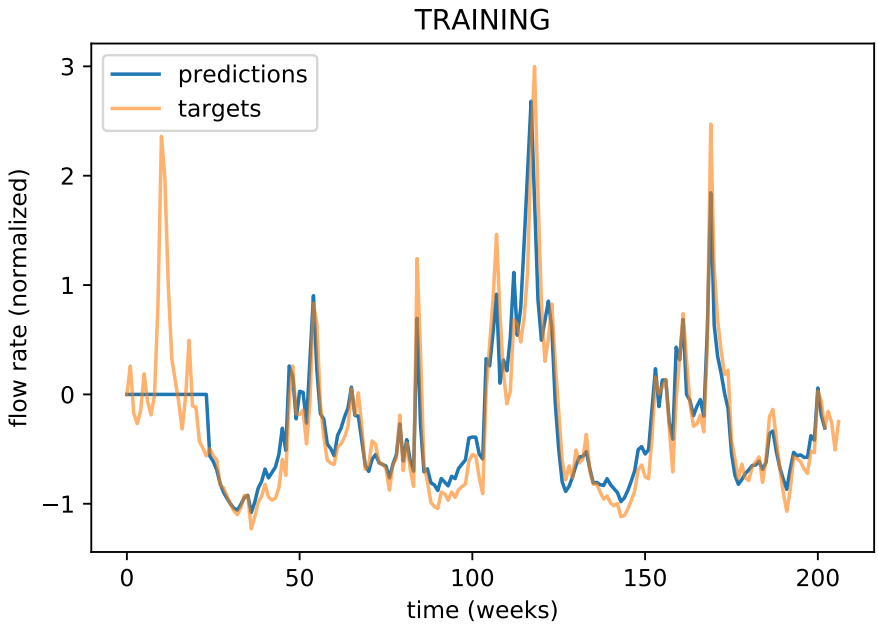


Obrázek 18: Porovnání předpovědí se skutečnými hodnotami trénovacích dat průtoku řeky Labe při rozvinutí sítě 24 týdnů do minulosti s 16 rekurentními jednotkami.

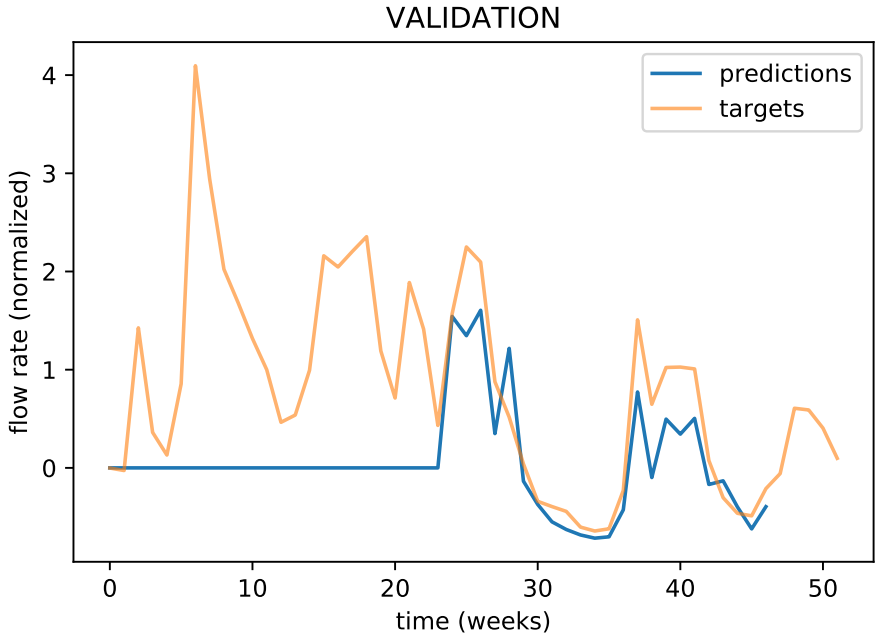


Obrázek 19: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Labe při rozvinutí sítě 24 týdnů do minulosti s 16 rekurentními jednotkami.

Tento jev je úkazem nedostatečné velikosti paměti buněk sítě a lze tedy eliminovat zvýšením počtu rekurentních jednotek (velikosti vnitřního stavu C) v buňkách sítě. V pokusu č. 8 (tab. 3) byl počet rekurentních jednotek zvýšen z 16 na 32 přičemž je možno pozorovat značné zvýšení přesnosti naučení trénovacích dat (viz obr. 20) a schopnosti predikovat data validační 21.



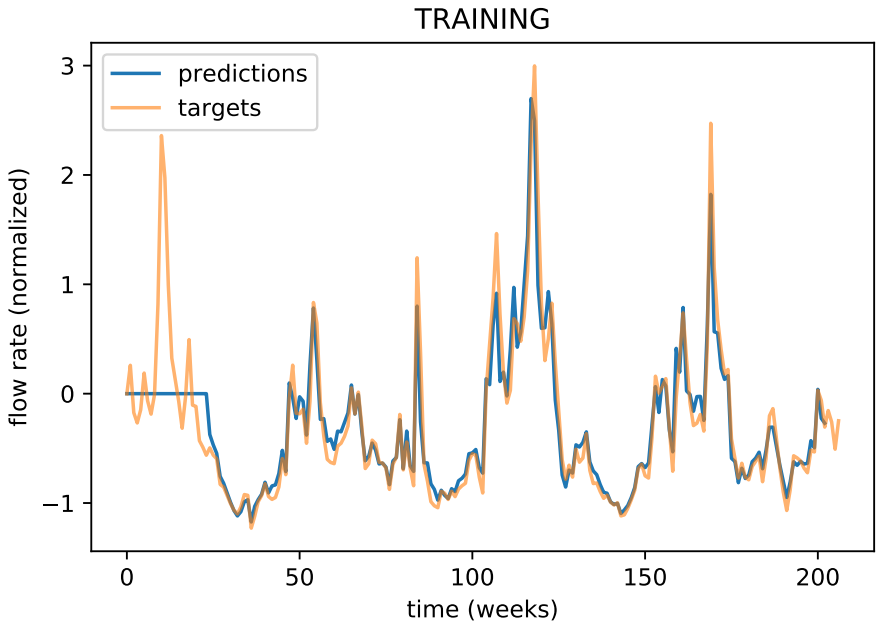
Obrázek 20: Porovnání předpovědí se skutečnými hodnotami trénovacích dat průtoku řeky Labe při rozvinutí sítě 24 týdnů do minulosti s 32 rekurentními jednotkami.



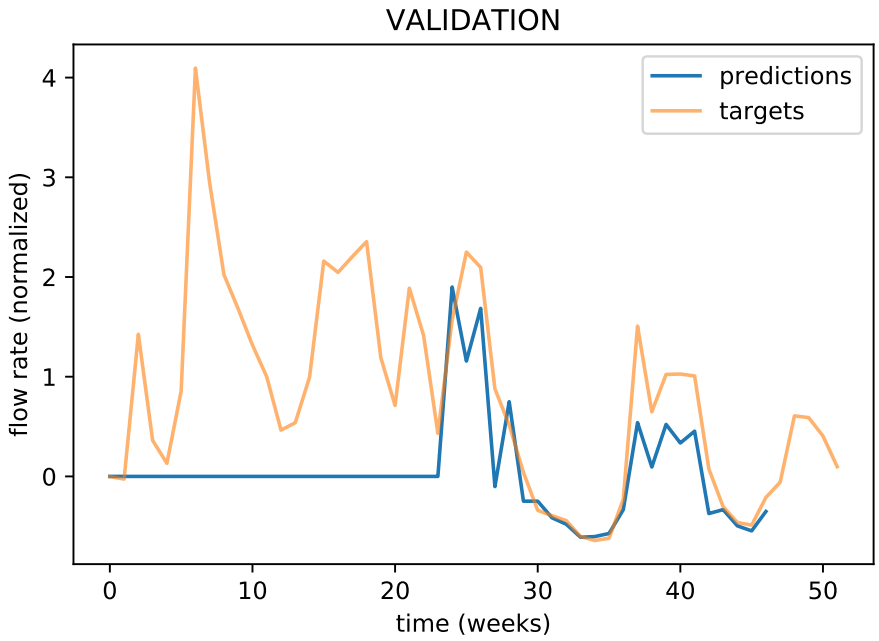
Obrázek 21: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Labe při rozvinutí sítě 24 týdnů do minulosti s 32 rekurentními jednotkami.

Přesnost naučení sítě a predikce validačních dat lze nadále mírně zlepšit zvýšením počtu rekurentních jednotek na 64 (viz pokus č. 9 v tab. 3 a obrázky 22 a 23). Je však nutno podotknout, že

hodnota SMAPE je stále o 14,1 % vyšší než při základních hyperparametrech (pokus č. 0 v tab. 3).



Obrázek 22: Porovnání předpovědí se skutečnými hodnotami trénovacích dat průtoku řeky Labe při rozvinutí sítě 24 týdnů do minulosti s 64 rekurentními jednotkami.

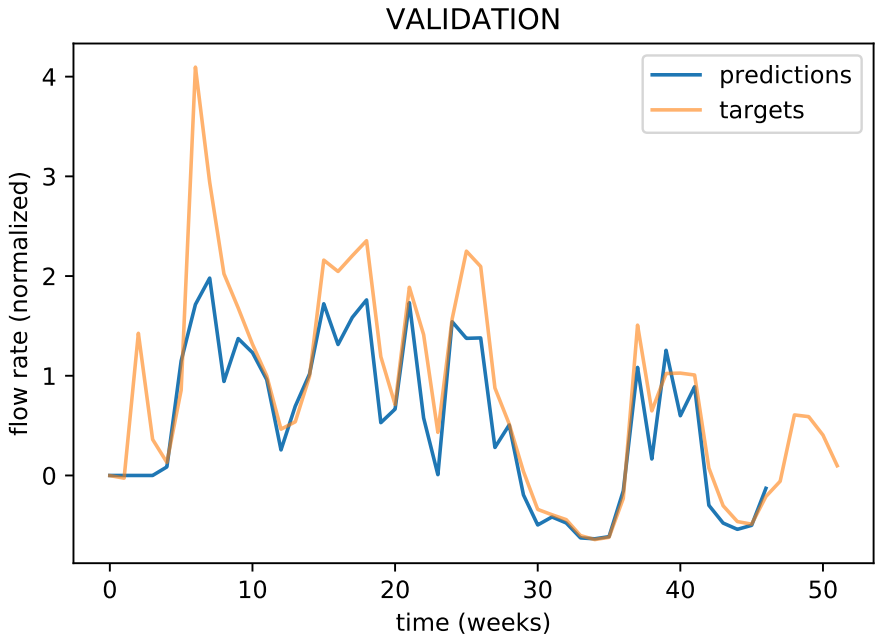


Obrázek 23: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Labe při rozvinutí sítě 24 týdnů do minulosti s 64 rekurentními jednotkami.

Posléze byly testovány změny velikosti dávky s hodnotami 2, 8, 16 a 24 (viz pokusy č. 10 až 13 v tab. 3). Velikost dávky byla měněna pouze pro trénovací data. Testovací a validační data byla stále posílána do sítě po dávkách o velikosti 4, aby bylo zajištěno získání hodnot účelové funkce a SMAPE z většiny dostupných testovacích a validačních dat a hodnoty byly srovnatelné s ostatními pokusy s velikostmi dávky 4. Z hodnot SMAPE v tabulce 3 je obdobně jako u velké sady trénovacích dat

v kapitole 4.3.1 vyvodit přímou úměrnost počtu učících epoch a nepřímou úměrnost doby učení s velikostí dávky. Na rozdíl od výsledků pro velkou datovou sadu je možno u predikce validačních dat průtoků Labe při zvyšování velikosti dávky pozorovat nárůst hodnot SMAPE. Nejnižší hodnoty (64,0 %) nabývalo SMAPE pro velikost dávky 2.

Závěrem byla provedena kombinace nejlépe dopadajících pokusů (z hlediska hodnoty SMAPE). Při pokusu č. 14 má síť 4 rekurentní jednotky, velikost dávky 2 a rozvinutí do minulosti 4 týdny. Nejnižší hodnota SMAPE z 5 opakování pokusu dosáhla 62,3 %, tedy hodnoty přibližně o 2,59 % nižší než pro základní hodnoty hyperparametrů (pokus č. 0). Jedná se o ze všech realizovaných pokusů na malé datové sadě nejnižší hodnotu SMAPE. Graf zobrazující porovnání předpovědí sítě se skutečnými hodnotami validační části časové řady je vykreslen na obrázku 24.



Obrázek 24: Porovnání předpovědí se skutečnými hodnotami validačních dat průtoku řeky Labe při 4 rekurentních jednotkách, velikosti dávky 2 a rozvinutí 4 týdnů do minulosti.

Tabulka 3: Výsledky jednokrokové predikce průtoku řeky Labe pro testované hodnoty hyperparametrů bez dropout.

číslo pokusu	**rekur. jednotek**	**velikost dávky**	**rozvinutí do minulosti**	**epoch učení**	**SMAPE** (%)	**doba učení** (hod : min : s)
0	16	4	16	70	0,648836	0:00:36
1	4	4	16	240	0,642717	0:01:58
2	32	4	16	30	0,655974	0:00:26
3	64	4	16	20	0,710103	0:00:26
4	128	4	16	10	0,720485	0:00:24
5	16	4	4	90	0,631591	0:00:21
6	16	4	12	50	0,653681	0:00:27
7	16	4	24	20	1,189776	0:00:16
8	32	4	24	90	0,822633	0:00:55
9	64	4	24	70	0,790182	0:01:08
10	16	2	16	30	0,640359	0:00:36
11	16	8	16	80	0,653946	0:00:34
12	16	16	16	80	0,689889	0:00:28
13	16	24	16	90	0,745502	0:00:27
14	4	2	4	240	0,622945	0:00:58

5 Závěr

Byla provedena studie rekurentních neuronových sítí, jejich základní struktury, důležitých vlastností a využití (viz kap. 2). V kapitole 2.1 byla prozkoumána funkcionalita rekurentní buňky, její rozvinutí do minulých hodnot časové řady a sdílení váhových koeficientů a hodnot skrytého stavu buňky v čase. Zaměřením práce byla zejména díky své efektivitě učení hojně využívaná Long Short-Term Memory (LSTM) architektura rekurentní buňky, jejíž podrobný popis včetně v ní probíhajících výpočetních vztahů se nachází v kapitole 2.3. Dále byl v kapitole 2.4 proveden stručný přehled rozmanitého aplikačního využití rekurentních neuronových sítí včetně podrobnějšího zaměření na predikci časových řad a identifikaci trendů a sezónních složek v časových řadách. Kapitola 3 uvádí do problematiky hyperparametrů sítě, vyčleňuje nejdůležitější z nich a krátce popisuje jejich vliv na chování sítě.

V praktické části (kap. 4) byla v programovacím jazyce Python3 s výpočetními moduly NumPy a TensorFlow v prostředí Jupyter Notebook realizována rekurentní neuronová síť s LSTM architekturou a nastavitelnými hyperparametry, mezi které patří počet vertikálních vrstev, množství rekurentních jednotek, rozvinutí do minulosti, velikost trénovací, testovací a validační dávky, maximální počet epoch a předčasné zastavení trénování při neklesající účelové funkci, míra učení, počet opakování pokusu a zavedení dropout (náhodného vynulování určitého procenta elementů při učení) vstupní, rekurentní či výstupní vrstvy sítě. Pro učení a testování sítě byly zvoleny dvě časové řady (viz kapitola 4.1); normalizovaný průměrný týdenní průtok řeky Labe v průběhu 6.let s 313 vzorky (viz obrázek 9) poskytnutý prof. Ing. Alešem Procházkou, CSc a průměrný denní průtok řeky Saugeen v období 1915 až 1979 s 23741 vzorky z *Time Series Data Library* [32] (viz obrázek 10). Některé hyperparametry navržené sítě byly pro snížení stupňů volnosti při testování sítě nastaveny fixně. Jedná se o účelovou funkci (RMSE), funkci pro vyhodnocení relativní chyby predikce (SMAPE), optimalizační algoritmus (Adam) a míru učení $(0,001)$. Více informací o těchto nastaveních se nachází v kapitole 4.2. Následně byly empiricky zvoleny počáteční hodnoty ostatních hyperparametrů sítě a provedeno porovnání přesnosti (hodnoty SMAPE při predikci validační části časové řady), počtu epoch potřebných k naučení sítě a rychlosti učení při změnách některých z nich (počet rekurentních jednotek, velikost dávky a rozvinutí do minulosti) pro obě vstupní časové řady. Průběh a výsledky testování pro časovou řadu denních průtoků řeky Saugeen je předmětem kapitoly 4.3.1. Zde bylo dosaženo nejpříznivějších výsledků (nejnižších hodnot SMAPE) pro 16 rekurentních jednotek, velikost dávky trénovacích, validačních i testovacích dat 16 a rozvinutí 16 dnů do minulosti s hodnotou SMAPE $7,9\,\%$ dosaženou po 40 epochách učení za 4 minuty a 42 sekund. Shrnutí výsledků všech testovaných nastavení hyperparametrů sítě pro tuto datovou sadu se nachází v tabulce 2. Obdobně bylo v kapitole 4.3.2 provedeno testování nastavení hyperparametrů sítě pro časovou řadu normalizovaných týdenních průtoků řeky Labe. Jelikož se jedná o velmi malou datovou sadu, nemá síť dostatek unikátních vzorků dat k přesnému naučení závislostí v datech a kvalita predikce je proto v porovnání s výsledky kapitoly 4.3.1 řádově nižší, což se projevuje ve zvýšení hodnot SMAPE. Nejlepších výsledků bylo v tomto případě dosaženo velmi malou sítí s počtem rekurentních jednotek 4, velikostí dávky 2 a rozvinutím do minulosti 4 s hodnotou SMAPE $62,3\,\%$ dosaženou po 240 epochách učení za 58 sekund.

V budoucím rozšíření práce je plánováno testování vícekrokové predikce, pro kterou je realizovaný program již připraven. Dále lze ověřit vliv zavedení dropout na kvalitu predikce testovacích a validačních dat, zejména pro malou datovou sadu průtoku řeky Labe. Následně je možno prozkoumat praktiky automatizované optimalizace hodnot hyperparametrů mezi které patří např. grid search (postupné nastavování všech možných permutací hodnot zvolených hyperparametrů) nebo random search (náhodné nastavování zvolených hyperparametrů) o nichž je možno zjistit více např. ve článku *Random Search for Hyper-parameter Optimization* [35].

Odkazy

[1] Hochreiter, Sepp a Schmidhuber, Jürgen. „Long Short-term Memory“. In: 9 (pros. 1997), s. 1735–80.

[2] Goodfellow, Ian, Bengio, Yoshua a Courville, Aaron. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[3] Pascanu, Razvan et al. „How to Construct Deep Recurrent Neural Networks“. In: *CoRR* abs/1312.6026 (2013). arXiv: 1312.6026. URL: http://arxiv.org/abs/1312.6026.

[4] Hermans, Michiel a Schrauwen, Benjamin. „Training and Analysing Deep Recurrent Neural Networks“. In: *Advances in Neural Information Processing Systems 26*. Ed. Burges, C. J. C. et al. Curran Associates, Inc., 2013, s. 190–198. URL: http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf.

[5] Lipton, Zachary Chase. „A Critical Review of Recurrent Neural Networks for Sequence Learning“. In: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[6] Ruder, Sebastian. „An overview of gradient descent optimization algorithms“. In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[7] Hochreiter, Sepp. „The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions“. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06.02 (1998), s. 107–116. DOI: 10.1142/S0218488598000094. eprint: https://www.worldscientific.com/doi/pdf/10.1142/S0218488598000094. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218488598000094.

[8] Million, Elizabeth. „The Hadamard Product“. In: (dub. 2007). URL: http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf.

[9] Gers, F. A., Schmidhuber, J. a Cummins, F. „Learning to forget: continual prediction with LSTM“. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*. Sv. 2. 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218.

[10] Olah, Christopher. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (cit. 29. 04. 2018).

[11] Andrej Karpathy, Fei-Fei Li. „Deep Visual-Semantic Alignments for Generating Image Descriptions“. In: (2015). URL: https://cs.stanford.edu/people/karpathy/sfmltalk.pdf.

[12] Venugopalan, Subhashini et al. „Sequence to Sequence - Video to Text“. In: *CoRR* abs/1505.00487 (2015). arXiv: 1505.00487. URL: http://arxiv.org/abs/1505.00487.

[13] Andrej Karpathy, Fei-Fei Li. *Automated Image Captioning with ConvNets and Recurrent Nets*. 2015. URL: https://cs.stanford.edu/people/karpathy/sfmltalk.pdf.

[14] Karpathy, Andrej. *The Unreasonable Effectiveness of Recurrent Neural Networks*. 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (cit. 29. 04. 2018).

[15] Cho, Kyunghyun et al. „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation“. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[16] Wu, Yonghui et al. „Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation“. In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144.

[17] Vinyals, Oriol a Le, Quoc V. „A Neural Conversational Model“. In: *CoRR* abs/1506.05869 (2015). arXiv: 1506.05869. URL: http://arxiv.org/abs/1506.05869.

[18] Zaremba, Wojciech a Sutskever, Ilya. „Learning to Execute“. In: *CoRR* abs/1410.4615 (2014). arXiv: 1410.4615. URL: http://arxiv.org/abs/1410.4615.

[19] Bobriakov, Igor. *Bitcoin price forecasting with deep learning algorithms*. 6. břez. 2018. URL: https://medium.com/activewizards-machine-learning-company/bitcoin-price-forecasting-with-deep-learning-algorithms-eb578a2387a3 (cit. 06. 05. 2018).

[20] Agarwala, Nipun, Inoue, Yuki a Sly, Axel. *Music Composition using Recurrent Neural Networks*. 11. dub. 2017. URL: https://github.com/yinoue93/CS224N_proj/raw/master/final_paper.pdf (cit. 06. 05. 2018).

[21] Graves, A., Mohamed, A. r. a Hinton, G. „Speech recognition with deep recurrent neural networks“. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Květ. 2013, s. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

[22] Sudipto, Saha a S., Raghava G. P. „Prediction of continuous B-cell epitopes in an antigen using recurrent neural network“. In: *Proteins: Structure, Function, and Bioinformatics* 65.1 (), s. 40–48. DOI: 10.1002/prot.21078. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.21078. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.21078.

[23] Brownlee, Jason. *What is the Difference Between a Parameter and a Hyperparameter?* 26. čvc 2017. URL: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/ (cit. 13. 05. 2018).

[24] Yao, Yuan, Rosasco, Lorenzo a Caponnetto, Andrea. „On Early Stopping in Gradient Descent Learning“. In: *Constructive Approximation* 26.2 (říj. 2007), s. 289–315. ISSN: 1432-0940. DOI: 10.1007/s00365-006-0663-2. URL: https://doi.org/10.1007/s00365-006-0663-2.

[25] community, GitHub. *ML Cheatsheet. Loss Functions*. 2017. URL: http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html (cit. 18. 05. 2018).

[26] Lau, Suki. *Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning*. 29. čvc 2017. URL: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1 (cit. 12. 05. 2018).

[27] Goyal, Priya et al. „Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour“. In: *CoRR* abs/1706.02677 (2017). arXiv: 1706.02677. URL: http://arxiv.org/abs/1706.02677.

[28] Gal, Yarin a Ghahramani, Zoubin. „A Theoretically Grounded Application of Dropout in Recurrent Neural Networks“. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, s. 1027–1035. ISBN: 978-1-5108-3881-9. URL: http://dl.acm.org/citation.cfm?id=3157096.3157211.

[29] Jones, Eric, Oliphant, Travis, Peterson, Pearu et al. *SciPy: Open source scientific tools for Python*. 2001. URL: http://www.scipy.org/ (cit. 13. 05. 2018).

[30] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software dostupný na tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[31] Kluyver, Thomas et al. „Jupyter Notebooks - a publishing format for reproducible computational workflows“. In: *ELPUB*. 2016.

[32] Hipel a McLeod. *Mean daily saugeen River flows, Jan 01, 1915 to Dec 31, 1979. Time Series Data Library*. 1994. URL: https://datamarket.com/data/set/235a/mean-daily-saugeen-river-flows-jan-01-1915-to-dec-31-1979#!ds=235a&display=line (cit. 13. 05. 2018).

[33] Hyndman, Rob J. *Errors on percentage errors*. 16. dub. 2014. URL: https://robjhyndman.com/hyndsight/smape/ (cit. 16. 05. 2018).

[34] Kingma, Diederik P. a Ba, Jimmy. „Adam: A Method for Stochastic Optimization“. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[35] Bergstra, James a Bengio, Yoshua. „Random Search for Hyper-parameter Optimization“. In: *J. Mach. Learn. Res.* 13 (ún. 2012), s. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2188395.

Přílohy

A Zdrojový kód realizovaného programu

```
## Prediction of river flow using RNN with LSTM architecture
import tensorflow as tf
from tensorflow.contrib import rnn
from tensorflow.python.tools import inspect_checkpoint as chkp # import the
    inspect_checkpoint library
import numpy as np
import scipy.io as sio # for working with .mat files
from openpyxl import load_workbook # for working with .xlsx files
import matplotlib.pyplot as plt # for plotting the data
from datetime import datetime # for keeping separate TB logs for each run
import os, sys
import textwrap


# define class for net parameters
class params:
    # initialization of instance variables
    def __init__(self,n_lstm_layers=2,hidden_size=10,delay=10,pred_step=1,
    train_batch_size=5,test_batch_size=5,val_batch_size=5,n_epochs=100,
    stop_epochs=20,check_every=10,init_lr=0.001,n_repeats=1,dropout=False,
    input_keepProb=1.0,output_keepProb=1.0,recurrent_keepProb=1.0):
        self.input_size = 1 # number of input features (we only follow one
    variable == flow)
        self.num_classes = 1 # number of output classes (we wan't specific
    value, not classes, so this is 1)
        self.target_shift = 1 # the target is the same time-series shifted by 1
     time-step forward
        self.n_lstm_layers = n_lstm_layers # number of vertically stacked LSTM
    layers
        self.hidden_size = hidden_size # hidden state vector size in LSTM cell
        self.delay = delay # the number of time-steps from which we are going
    to predict the next step
        self.pred_step = pred_step # the number of time-steps we predict into
    the future (1 == One-step prediction ; >1 == Multi-step prediction)
        self.train_batch_size = train_batch_size # number of inputs in one
    training batch
        self.test_batch_size = test_batch_size # number of inputs in one
    testing batch
        self.val_batch_size = val_batch_size # number of inputs in one
    validation batch
        self.n_epochs = n_epochs # number of epochs
        self.stop_epochs = stop_epochs # if the loss value doesn't improve over
     the last stop_epochs, the training process will stop
        self.check_every = check_every # how often to check for loss value in
    early stopping
        self.init_lr = init_lr # initial learning rate for Adam optimizer (
    training algorithm)
        self.n_repeats = n_repeats # number of repeats of the whole training
    and validation process with the same params
        self.net_unroll_size = self.delay + self.pred_step - 1 # number of
    unrolled LSTM time-step cells
        # FIGHTING OVERFITTING:
```

```
        self.dropout = dropout # if true the dropout is applied on inputs, outputs and recurrent states of cells
        self.input_keepProb = input_keepProb # (dropout) probability of keeping the input
        self.output_keepProb = output_keepProb # (dropout) probability of keeping the output
        self.recurrent_keepProb = recurrent_keepProb # (dropout) probability of keeping the recurrent state

    # representation of object for interpreter and debugging purposes
    def __repr__(self):
        return '''params(n_lstm_layers={:d},hidden_size={:d},delay={:d},pred_step={:d},train_batch_size={:d},test_batch_size={:d},val_batch_size={:d},n_epochs={:d},stop_epochs={:d},check_every={:d},init_lr={:f},dropout={},n_repeats={:d},input_keepProb={:f},output_keepProb={:f},recurrent_keepProb={:f})'''.format(self.n_lstm_layers,

    self.hidden_size,

    self.delay,

    self.pred_step,

    self.train_batch_size,

    self.test_batch_size,

    self.val_batch_size,

    self.n_epochs,

    self.stop_epochs,

    self.check_every,

    self.init_lr,

    self.n_repeats,

    self.dropout,

    self.input_keepProb,

    self.output_keepProb,

    self.recurrent_keepProb)

    # how will the class object be represented in string form (eg. when called with print())
    def __str__(self):
        answer = '''
Input size ..................... {:4d}
Number of classes .............. {:4d}
Target shift ................... {:4d}
Number of stacked LSTM layers ... {:4d}
Hidden state size .............. {:4d}
Delay .......................... {:4d}
Number of prediciton steps...... {:4d}
```

```
Training batch size ..... {:4d}
Testing batch size ..... {:4d}
Validation batch size ..... {:4d}
Maximum number of epochs ..... {:4d}
Early stopping epochs ..... {:4d}
Check (save) every (epochs)..... {:4d}
Initial learning rate ..... {:9.4f}
Dropout ..... {}'''.format(self.input_size
                                        ,self.num_classes
                                        ,self.target_shift
                                        ,self.n_lstm_layers
                                        ,self.hidden_size
                                        ,self.delay
                                        ,self.pred_step
                                        ,self.train_batch_size
                                        ,self.test_batch_size
                                        ,self.val_batch_size
                                        ,self.n_epochs
                                        ,self.stop_epochs
                                        ,self.check_every
                                        ,self.init_lr
                                        ,self.dropout)

        dropout_answer = '''
Input keep probability ..... {:7.2f}
Output keep probability ..... {:7.2f}
Recurrent keep probability ..... {:7.2f}'''.format(self.input_keepProb
                                                   ,self.output_keepProb
                                                   ,self.recurrent_keepProb)

        if self.dropout:
            return answer + dropout_answer
        else:
            return answer

# net and training parameter specification
par = params(n_lstm_layers = 10
             ,hidden_size = 16
             ,delay = 4
             ,pred_step = 1
             ,train_batch_size=16 # (https://machinelearningmastery.com/use-
    different-batch-sizes-training-predicting-python-keras/)
             ,test_batch_size=4
             ,val_batch_size=4
             ,n_epochs=1000
             ,stop_epochs=50 # if <= 0 then early stopping is disabled and check
     every sets the saving period
             ,check_every=10
             ,init_lr=0.001
             ,n_repeats=1
             ,dropout=True
             ,input_keepProb=1.0
             ,output_keepProb=0.75
             ,recurrent_keepProb=1.0)


# enable/disable L2 regularization
```

```
regularization = False # NOT IMPLEMENTED YET (go to: https://stackoverflow.com/questions/37869744/tensorflow-lstm-regularization)

# continue training the model with saved variables from previeous training
continueLearning = False

# decaying learning rate constants (for exponential decay)
# Adam already has adaptive learning rate for individual weights
# but it can be combined with decaying learning rate for faster convergence
decay_steps = par.n_epochs//10 # every "n_epochs//10" epochs the learning rate is reduced
decay_rate = 1 # the base of the exponential (rate of the decay ... 1 == no decay)


# select input data for the network
# 1) ... Weekly Elbe flow (normalized)
# 2) ... Daily Saugeen flow (unknown)
selected_data = 1


# ### 1) DATA from prof. A. Procházka:
# * **url:**: http://uprt.vscht.cz/prochazka/pedag/Data/dataNN.zip
# * **name**: Weekly Elbe river flow
# * **Provider source:** Prof. Ing. Aleš Procházka, CSc
# * **Span:** 313 weeks ~ 6 years of data
# * **Data size:** 313 values
# * **Already normalized** to 0 mean and 1 variance
if selected_data == 1:
    # load data from Q.mat
    filename = './datasets/Q.MAT'
    data = sio.loadmat(filename) # samples were gathered with period of one week

    # convert to np array
    data = np.array(data['Q'],dtype=np.float32)

    print(np.shape(data))

    # normalize the data to interval (0,1)
    min_data = np.min(data)
    max_data = np.max(data)
    # shift the data to start at 0.001 (for relative error counting purposes)
#    data = np.subtract(data,min_data)+0.001
    # data = np.divide(np.subtract(data,min_data),np.subtract(max_data,min_data)).flatten()
    # normalize the data to interval (-1,1) (cca 0 mean and 1 variance)
    mean_data = np.mean(data) # mean
    std_data = np.std(data) # standard deviation
    data = np.divide(np.subtract(data,mean_data),std_data).flatten()

    # divide the data into training, testing and validation part
    weeks_in_year = 52.1775
    years_in_data = 313/weeks_in_year

    years_in_train = int(years_in_data*0.7) # 70% of data rounded to the number of years
```

```
175     years_in_test = int(np.ceil(years_in_data*0.15)) # 15% of data rounded to
        the number of years

177     weeks_train = int(years_in_train*weeks_in_year) # number of weeks in
        training data
        weeks_test = int(years_in_test*weeks_in_year) # number of weeks in testing
        data
179
        end_of_train = weeks_train
181     end_of_test = weeks_train + weeks_test

183     x_train = data[:end_of_train]
        x_test = data[end_of_train:end_of_test]
185     x_validation = data[end_of_test:]

187     plot_start = 0
        plot_end = -1
189


191 # ### 2) DATA from Time Series Data Library:
    # * **url:** https://datamarket.com/data/set/235a/mean-daily-saugeen-river-
        flows-jan-01-1915-to-dec-31-1979#!ds=235a&display=line
193 # * **name:** Mean daily Saugeen River (Canada) flows
    # * **Provider source:** Hipel and McLeod (1994)
195 # * **Span:** Jan 01, 1915 to Dec 31, 1979
    # * **Data size:** 23741 values
197 if selected_data == 2:
        # load excel spreadsheet with openpyxl:
199     filename = './datasets/sugeen-river-flows.xlsx'
        xl = load_workbook(filename)
201
        # print sheet names:
203     print(xl.get_sheet_names())

205     # get sheet:
        sheet = xl.get_sheet_by_name('Mean daily saugeen River flows,')
207
        data = []
209
        # fill a list with values from cells:
211     for cell in sheet['B16:B23756']:
            data.append(cell[0].value)
213
        # convert list to numpy array and reshape to a column vector
215     data = np.array(data)
        data = np.reshape(data,(1,-1))
217
        # normalize the data to interval (0,1) <- DONT!
219     min_data = np.min(data)
        max_data = np.max(data)
221     # data = np.divide(np.subtract(data,min_data),np.subtract(max_data,min_data
        )).flatten()
        # !!! CENTERING data:
223     # normalize the data to interval (-1,1) (cca 0 mean and 1 variance)
        # data = data[0,:120]
225     mean_data = np.mean(data) # mean
        std_data = np.std(data) # standard deviation
227     data = np.divide(np.subtract(data,mean_data),std_data).flatten()
```

```
229     # divide the data into training, testing and validation part
        days_in_data = np.shape(data)[0]
231     days_in_year = 365.25
        years_in_data = days_in_data/days_in_year
233
        years_in_train = int(years_in_data*0.7) # 70% of data rounded to the number
         of years
235     years_in_test = int(np.ceil(years_in_data*0.15)) # 15% of data rounded to
        the number of years

237     days_train = int(years_in_train*days_in_year) # number of days in training
        data
        days_test = int(years_in_test*days_in_year) # number of days in testing
        data
239
        end_of_train = days_train
241     end_of_test = days_train + days_test

243     x_train = data[:end_of_train]
        x_test = data[end_of_train:end_of_test]
245     x_validation = data[end_of_test:]

247     plot_start = int(days_in_year*3)
        plot_end = int(days_in_year*5)
249


251 # define the shifted time-series (targets)
    y_train = np.roll(x_train, par.target_shift)
253 y_test = np.roll(x_test, par.target_shift)
    y_validation = np.roll(x_validation, par.target_shift)
255
    # delete the first elements of the time series that were reintroduced from the
        end of the timeseries
257 y_train[:par.target_shift] = 0
    y_test[:par.target_shift] = 0
259 y_validation[:par.target_shift] = 0


261
    # reset TensorFlow graph
263 tf.reset_default_graph()

265 # define tensorflow constants
    min_of_data = tf.constant(min_data, dtype=tf.float32, name='min_of_data')
267 max_of_data = tf.constant(max_data, dtype=tf.float32, name='max_of_data')
    mean_of_data = tf.constant(mean_data, dtype=tf.float32, name='mean_of_data')
269 std_of_data = tf.constant(std_data, dtype=tf.float32, name='std_of_data')

271 # define output weights and biases
    with tf.name_scope("output_layer"):
273     weights_out = tf.Variable(tf.random_normal([par.hidden_size,par.num_classes
        ]),name='weights_out')
        bias_out = tf.Variable(tf.random_normal([par.num_classes]),name='biases_out
        ')
275
    # define placeholders for the batches of time-series
277 x = tf.placeholder(tf.float32,[None, par.net_unroll_size, par.input_size],name=
        'x') # batch of inputs
```

```
y = tf.placeholder(tf.float32,[None, par.num_classes, par.pred_step],name='y')
    # batch of labels

# define placeholders for dropout keep probabilities
input_kP = tf.placeholder(tf.float32,name='input_kP')
output_kP = tf.placeholder(tf.float32,name='output_kP')
recurrent_kP = tf.placeholder(tf.float32,name='recurrent_kP')


# processing the input tensor from [par.batch_size,par.delay,par.input_size] to
     "par.delay" number of [par.batch_size,par.input_size] tensors
input=tf.unstack(x, par.net_unroll_size, 1, name='LSTM_input_list') # create
    list of values by unstacking one dimension


# function to create an LSTM cell:
def make_cell(hidden_size):
    return rnn.LSTMCell(hidden_size, state_is_tuple=True, activation=tf.tanh)

# define an LSTM network with 'par.n_lstm_layers' layers
with tf.name_scope("LSTM_layer"):
    lstm_cells = rnn.MultiRNNCell([make_cell(par.hidden_size) for _ in range(
    par.n_lstm_layers)], state_is_tuple=True)

    # add dropout to the inputs and outputs of the LSTM cell (reduces
    overfitting)
    lstm_cells = rnn.DropoutWrapper(lstm_cells, input_keep_prob=input_kP,
    output_keep_prob=output_kP, state_keep_prob=recurrent_kP)

    # create static RNN from lstm_cell
    outputs,_ = rnn.static_rnn(lstm_cells, input, dtype=tf.float32)


# generate a list of predictions based on the last "par.pred_step" time-step
    outputs (multi-step prediction)
prediction = [tf.matmul(outputs[-i-1],weights_out) + bias_out for i in (range(
    par.pred_step))] # newest prediction first
prediction = prediction[::-1] #reverse the list (oldest prediction first)

prediction = tf.reshape(prediction,[-1,par.num_classes,par.pred_step])


# define loss function with regularization
with tf.name_scope("loss"):
    loss = tf.sqrt(tf.losses.mean_squared_error(predictions=prediction,labels=y
    )) # RMSE (root mean squared error)

# exponential decay of learning rate with epochs
global_step = tf.Variable(1, trainable=False, name='global_step') # variable
    that keeps track of the step at which we are in the training
increment_global_step_op = tf.assign(global_step, global_step+1,name='
    increment_global_step') # operation that increments global step by one
# decayed_learning_rate = learning_rate * decay_rate ^ (global_step /
    decay_steps)
learning_rate = tf.train.exponential_decay(par.init_lr, global_step,
                                           decay_steps, decay_rate, staircase=
    True) # decay at discrete intervals
```

```
323 # define Adam optimizer for training of the network
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
325
327 # denormalize data (from (-1,1)):
    denormalized_prediction = mean_of_data + tf.multiply(prediction, std_of_data)
329 denormalized_y = mean_of_data + tf.multiply(y, std_of_data)

331 # calculate relative error of denormalized data:
    with tf.name_scope("SMAPE"):
333     SMAPE = 2*tf.reduce_mean(tf.divide(tf.abs(tf.subtract(
        denormalized_prediction,denormalized_y))
                                           ,tf.add(tf.abs(denormalized_y),tf.abs(
        denormalized_prediction))))
335

337 # TensorBoard summaries (visualization logs)
    # histogram summary for output weights
339 w_out_summ = tf.summary.histogram("w_out_summary", weights_out)

341
    def create_batch(x_data,y_data,batch_size,index):
343     """
        function for generating the time-series batches
345
        :param x_data: input data series
347     :param y_data: output data series
        :param batch_size: size of one batch of data to feed into network
349     :param index: index of the current batch of data
        :return: input and output batches of data
351     """

353     x_batch = np.zeros([batch_size,par.delay,par.input_size])
        x_pad = np.zeros([batch_size,par.pred_step-1,par.input_size])
355     y_batch = np.zeros([batch_size,par.num_classes,par.pred_step])

357     step = index*(batch_size*par.pred_step)

359     for i in range(batch_size):
            x_batch[i,:,:] = np.reshape(x_data[step+i*par.pred_step:step+i*par.
        pred_step+par.delay],(par.delay,par.num_classes))
361         y_batch[i,:] = np.reshape(y_data[step+par.delay+i*par.pred_step+1:step+
        par.delay+i*par.pred_step+1+par.pred_step],(1,par.num_classes,par.pred_step
        ))

363     # the last "par.pred_step - 1" columns in x_batch are padded with 0
        # because there are no inputs into the net at these time steps
365     x_batch = np.hstack((x_batch, x_pad))

367 #    print(x_batch)
    #    print('_________________')
369 #    print(y_batch)
    #    print('================')
371
        return x_batch, y_batch
373

375 def run_model(inputs,labels,batch_size,save=False,train=False):
```

```
    """
377     feeding the model with batches of inputs, running the optimizer for
    training and getting training and testing results

379     :param inputs: input data series
        :param labels: output data series
381     :param batch_size: the size of the data batch
        :param save: if True the predicted time series is returned as a list
383     :param train: if True the optimizer will be called to train the net on
    provided inputs and labels
        :return: loss (cost) and prediction error throughout the whole data set and
         if save = True: also returns the list of predicted values
385     """

387     prediction_list = [] # list for prediction results
        loss_val_sum = 0 # sum of the loss function throughout the whole data
389     error_val_sum = 0 # sum of the relative error function throughout the whole
         data
        error_val_mean = 0
391     prefix = ""

393     # number of batches == number of iterations to go through the whole dataset
         once
        n_batches = (len(inputs)-par.delay-2)//(batch_size*par.pred_step)
395     remainder = (len(inputs)-par.delay-2)%(batch_size*par.pred_step)

397     for i in range(n_batches):
            # get batch of data
399 #          if i == n_batches:
    #              # last batch
401 #              x_batch, y_batch = create_batch(inputs,labels,remainder,-1/par.
    pred_step)
    #          else:
403             # batches before last
                x_batch, y_batch = create_batch(inputs,labels,batch_size,i)
405


407             # dropout at training
                if par.dropout:
409                 feed_dict_train = {x: x_batch, y: y_batch, input_kP: par.
    input_keepProb,
                                       output_kP: par.output_keepProb, recurrent_kP:
    par.recurrent_keepProb}
411             else:
                    feed_dict_train = {x: x_batch, y: y_batch, input_kP: 1.0, output_kP
    : 1.0, recurrent_kP: 1.0}
413
                # no dropout at testing and validation
415             feed_dict_test = {x: x_batch, y: y_batch, input_kP: 1.0, output_kP:
    1.0, recurrent_kP: 1.0}

417             # train the net on the data
                if train:
419                 prefix = "Training_" # for summary writer
                    session.run(optimizer,feed_dict=feed_dict_train) # run the
    optimization on the current batch
421 #              loss_val, prediction_val, error_val = session.run(
```

```
#                (loss, denormalized_prediction, SMAPE), feed_dict=
    feed_dict_train)
        else:
            prefix = "Testing_" # for summary writer
#            loss_val, prediction_val, error_val = session.run(
#                (loss, denormalized_prediction, SMAPE), feed_dict=
    feed_dict_test)

        loss_val, prediction_val, error_val = session.run(
            (loss, denormalized_prediction, SMAPE), feed_dict=feed_dict_test)

        # prediction_val is a list of length "par.pred_step" with arrays of "
    batch_size" output values
        # convert to numpy array of shape (batch_size, par.pred_step)
        prediction_val = np.array(prediction_val)
        # reshape the array to a vector of shape (1, par.pred_step*batch_size)
        prediction_val = np.reshape(prediction_val, (1, par.pred_step*
    batch_size))

        loss_val_sum += loss_val # sum the losses across the batches
        error_val_sum += error_val # sum the errors across the batches

        # save the results
        if save:
            # save the batch predictions to a list
            prediction_list.extend(prediction_val[0,:])

    # the mean value of loss (throughout all the batches) in current epoch
    loss_val_mean = loss_val_sum/n_batches
    # the mean of relative errors
    error_val_mean = error_val_sum/n_batches

    # Create a new Summary object for sum of losses and mean of errors
    loss_summary = tf.Summary()
    error_summary = tf.Summary()
    loss_summary.value.add(tag="{}Loss".format(prefix), simple_value=
    loss_val_mean)
    error_summary.value.add(tag="{}Error".format(prefix), simple_value=
    error_val_mean)

    # Add it to the Tensorboard summary writer
    # Make sure to specify a step parameter to get nice graphs over time
    summary_writer.add_summary(loss_summary, epoch)
    summary_writer.add_summary(error_summary, epoch)

    return loss_val_mean, error_val_mean, prediction_list


def early_stopping(loss_val,epoch,stop_epochs,check_every):
    """
    Save the model coefficients if the data loss function value is better than
    the last loss function value.

    :param loss_val: current value of loss function
    :param epoch: current epoch
    :param stop_epochs: number of epochs after which the training is stopped if
     the loss is not improving
```

```
    :param check_every: define the period in epochs at which to check the loss value (and at which to save the data)
    :return: the epoch at which the best loss was and the value of the loss (ergo at which the last checkpoint was created)
    """

    stop_training = False

    # initialize function attributes
    if not hasattr(early_stopping,"best_loss"):
        early_stopping.best_loss = loss_val
        early_stopping.best_epoch = epoch

    # saving if the loss val is better than the last
    if stop_epochs > 0 and epoch % check_every == 0:
        # if loss val is better than best_loss save the model parameters
        if loss_val < early_stopping.best_loss:
            saver.save(session, './checkpoints/Multi-Step_LSTMforPredictingLabeFlow')
            early_stopping.best_loss = loss_val
            early_stopping.best_epoch = epoch
            print("Model saved at epoch {} \nwith testing loss: {}.".format(epoch,loss_val))
        else:
            print("Model NOT saved at epoch {} \nwith testing loss: {}".format(epoch,loss_val))

        print("_______________________________")

        # if the loss didn't improve for the last stop_epochs number of epochs then the training process will stop
        if (epoch - early_stopping.best_epoch) >= stop_epochs:
            stop_training = True

    # only saving
    if stop_epochs <= 0 and epoch % check_every == 0:
        saver.save(session, './checkpoints/Multi-Step_LSTMforPredictingLabeFlow')
        early_stopping.best_loss = -1.0
        early_stopping.best_epoch = -1
        print("Model saved at epoch {}".format(epoch))
        print("_______________________________")


    return early_stopping.best_loss, early_stopping.best_epoch, stop_training


# TRAINING THE NETWORK
# initializer of TF variables
init = tf.global_variables_initializer()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# define unique name for log directory (one-step and multi-step are in sepparate directories)
now = datetime.now()
if par.pred_step > 1:
```

```
    logdir = "./logs/multi-step/" + now.strftime("%Y%m%d-%H%M%S") + "/"
else:
    logdir = "./logs/one-step/" + now.strftime("%Y%m%d-%H%M%S") + "/"

print(now.strftime("%Y%m%d-%H%M%S"))

for repeat in range(par.n_repeats):
    with tf.Session() as session:

        # initialize helping variables
        stop_training = False
        best_epoch = par.n_epochs

        # reset instance variables
        early_stopping.best_loss = 10000
        early_stopping.best_epoch = 0

        # Restore variables from disk if continueLearning is True.
        if continueLearning:
            # restoring variables will also initialize them
            saver.restore(session, './checkpoints/Multi-
    Step_LSTMforPredictingLabeFlow')
            print("Model restored.")
        else:
            session.run(init) # initialize variables


        # Create a SummaryWriter to output summaries and the Graph
        # in console run 'tensorboard --logdir=./logs/'
        summary_logdir = logdir + "/" + str(repeat)
        summary_writer = tf.summary.FileWriter(logdir=summary_logdir, graph=
    session.graph)

        for epoch in range(par.n_epochs):
            # TRAINING
            loss_val, error_val, _ = run_model(x_train,y_train,par.
    train_batch_size,save=False,train=True)

            # TESTING
            loss_val_test, error_val_test, _ = run_model(x_test,y_test,par.
    test_batch_size,save=False,train=False)


            # increment global step for decaying learning rate at each epoch
            session.run(increment_global_step_op)

            # Printing the results at every "par.n_epochs//10" epochs
            if epoch % (par.n_epochs//10) == 0:
                print("Epoch: {}".format(epoch))
                print("TRAINING Loss: {}".format(loss_val))
                print("TRAINING Error: {}".format(error_val))
                print("TESTING Loss: {}".format(loss_val_test))
                print("TESTING Error: {}".format(error_val_test))
                # flush the summary data into TensorBoard
                # summary_writer.flush()

            # Checking the model loss_value for early stopping each "
    check_every" epochs
```

```
            # save the trained net and variables for later use if the test loss_val is better than the last saved one
575         best_loss, best_epoch, stop_training = early_stopping(loss_val_test,epoch,par.stop_epochs,par.check_every)

577         # Stop the training process
            if stop_training:
579             print("The training process stopped prematurely at epoch {}.".format(epoch))
                break
581


583     # Restoring the model coefficients with best results
        with tf.Session() as session:
585
            # restore the net coefficients with the lowest loss value
587         saver.restore(session, './checkpoints/Multi-Step_LSTMforPredictingLabeFlow')
            print('Restored model coefficients at epoch {} with TESTING loss val: {:.4f}'.format(best_epoch, best_loss))
589
            # run the trained net with best coefficients on all time-series and save the results
591         loss_val, error_val, prediction_list = run_model(x_train,y_train,par.train_batch_size,save=True,train=False)
            loss_val_test, error_val_test, prediction_list_test = run_model(x_test,y_test,par.test_batch_size,save=True,train=False)
593         loss_val_validation, error_val_validation, prediction_list_validation = run_model(x_validation,y_validation,par.val_batch_size,save=True,train=False)


595
        # printing parameters and results to console
597     results = '''Timestamp: {}
-----------------------------------------------
599 Net parameters:
    {}
601 -----------------------------------------------
    Results: \n
603 Best epoch ..................... {:4d}
    TRAINING Loss ................... {:11.6f}
605 TRAINING Error .................. {:11.6f}
    TESTING Loss .................... {:11.6f}
607 TESTING Error ................... {:11.6f}
    VALIDATION Loss ................. {:11.6f}
609 VALIDATION Error ................ {:11.6f}
    -----------------------------------------------'''.format(now.strftime("%Y%m%d-%H%M%S")
611                                                             ,par
                                                                ,best_epoch
613                                                             ,loss_val
                                                                ,error_val
615                                                             ,loss_val_test
                                                                ,error_val_test
617                                                             ,loss_val_validation
                                                                ,error_val_validation)
619
        print(results)
```

```
621
623     # saving results to log file in logdir
        file = "{}log{}.txt".format(logdir,repeat)
625     with open(file, mode='w') as f:
            f.write(results)
627
        # Shift the predictions "par.delay" time-steps to the right
629     prediction_train = np.array(prediction_list)
        prediction_test = np.array(prediction_list_test)
631     prediction_validation = np.array(prediction_list_validation)

633     print(np.shape(prediction_train))

635     prediction_train = np.pad(prediction_train,pad_width=((par.delay,0))
                                  ,mode='constant',constant_values=0) # pad with "
    par.delay" zeros at the start of first dimension
637     prediction_test = np.pad(prediction_test,pad_width=((par.delay,0))
                                 ,mode='constant',constant_values=0) # pad with "
    par.delay" zeros at the start of first dimension
639     prediction_validation = np.pad(prediction_validation,pad_width=((par.delay
    ,0))
                                 ,mode='constant',constant_values=0) # pad with "
    par.delay" zeros at the start of first dimension
641
        print(np.shape(prediction_train))
643

645     def denormalize(labels):
            """
647         Denormalize target values from interval (-1,1) to original values

649         :param labels: values of labels to be denormalized
            :return: denormalized values of labels
651         """

653         # denormalized_labels = min_data + labels*(max_data - min_data)
            # denormalize the labels from (-1,1)
655         denormalized_labels = mean_data + labels*std_data

657         return denormalized_labels

659     y_train_denorm = denormalize(y_train)
        y_test_denorm = denormalize(y_test)
661     y_validation_denorm = denormalize(y_validation)


663     # Plot the results
665     def plot_results(predictions, targets, selected_data):
            """
667         plot the predictions and target values to one figure

669         :param predictions: the predicted values from the net
            :param targets: the actual values of the time series
671         :param selected_data: selector of input data (1 == Elbe flow, 2 ==
    Saugeen flow)
            :return: plot of predictions and target values
673         """
```

```
        plt.plot(predictions)
        plt.plot(targets, alpha=0.6)
        plt.legend(['predictions', 'targets'])
        if selected_data == 1:
            plt.xlabel('time (weeks)')
            plt.ylabel('flow rate (normalized)')
        elif selected_data == 2:
            plt.xlabel('time (days)')
            plt.ylabel('flow rate (unknown)')
        plt.draw()

    f_training = plt.figure()
    plot_results(prediction_train[plot_start:plot_end], y_train_denorm[
    plot_start:plot_end], selected_data)
    plt.title('TRAINING')

    f_testing = plt.figure()
    plot_results(prediction_test[plot_start:plot_end], y_test_denorm[plot_start
    :plot_end], selected_data)
    plt.title('TESTING')

    f_validation = plt.figure()
    plot_results(prediction_validation[plot_start:plot_end],
    y_validation_denorm[plot_start:plot_end], selected_data)
    plt.title('VALIDATION')
    plt.show()

    # Save the figures:
    img_save_dir = "{}IMG".format(logdir)
    save_dir_path = os.path.join(os.curdir, img_save_dir)
    os.makedirs(save_dir_path, exist_ok=True)

    f_training.savefig(save_dir_path + "/training{}.pdf".format(repeat),
    bbox_inches='tight')
    f_testing.savefig(save_dir_path + "/testing{}.pdf".format(repeat),
    bbox_inches='tight')
    f_validation.savefig(save_dir_path + "/validation{}.pdf".format(repeat),
    bbox_inches='tight')

    print("Figures saved to: {}".format(save_dir_path))
```