Универзитет у Београду
Електротехнички факултет

Анализа модела машинског учења за предикцију хаотичних динамичких система

Мастер рад

Ментор:
Проф. др Бошко Николић

Кандидат:
Никола Јовановић 3037/2019

Београд, август 2021.

Захвалница

Истраживања описана у овом раду реализована су приликом више стручних пракси на Рајс универзитету у Хјустону. Ментори на пројекту су били проф. др Кришна Палем и проф. др Девика Субраманиан. Захвалност дугујем и осталим сарадницима, а то су др Адам Дурач, др Рајан Пејл и проф. др Анкит Пател.

САДРЖАЈ

1. Увод

Тематика овог рада уско је повезана са системима за предикцију временских услова, а главна мотивација за овај рад је истраживање и потенцијално допринос квалитету истих система. Временска прогноза је услуга коју данас узимамо здраво за готово, ипак корисно је осврнути се на њену историју и препознати њену важност.

Праћење и предвиђање времена, изузимајући сујеверје и астрологију, започело је релативно скоро [1]. Изум који је по први пут омогућио извештавање о временским условима био је телеграф 1835. године, који је уз информације о смеру ветра могао да се користи за предвиђање временских услова. До краја 19. века, државе попут Енглеске, Француске, Сједињених Америчких Држава и Норвешке увешће системе за упозорење од временских непогода и процену услова за пловидбу на рекама и језерима. Почетком 20. века, проналазак радија омогућава размену информација о времену између копна и бродова.

До средине 20. века, временска прогноза се заснивала на праћењу временских услова и што ефикаснијем обавештавању о тренутном стању. Лансирање сателита за праћење времена, почевши од 1960. године, омогућава добијање детаљне слике о времену на нивоу целе планете, што уз развој рачунара и разумевања структуре атмосфере отвара врата за коришћење нумеричких симулација за предвиђање времена. По први пут могуће је са прихватљивом прецизношћу израчунати стање времена у будућности, брже од саме промене времена на терену.

Данас најбољи модели за предвиђање времена користе комбинације (*ensemble*) више различитих модела за глобално предвиђање времена, чији су резултати затим кориговани бројним статистичким хеуристикама за појединачне географске локације [2]. Главни модели се заснивају на решавању комплексних система диференцијалних једначина, којима се као почетно стање проследи најскорије измерено стање. Главни проблеми овог приступа су недовољно улазних података и недовољна нумеричка прецизност рачуна, који су изнуђени због потребе за брзином израчунавања симулације.

Системи диференцијалних једначина који се примењују у предикцији времена спадају у класу хаотичних динамичких система [3]. Они се дефинишу као системи чија је промена стања кроз време изгледа потпуно насумично и непредвидиво иако су заправо потпуно детерминистички. Разлог за ово је велика осетљивост система на почетно стање, односно на изглед неприметна промена у почетном стању у веома малом броју итерација система доводи до потпуно другачијих стања.

Централни део овог рада је истраживање и анализа примера хаотичних динамичких система, као и примене модела машинског учења у њиховој евалуацији [4]. Главни циљ је да се, без великог губитка прецизности у односу на симулационе методе, направи знатно бржи и енергетски ефикаснији модел машинског учења за евалуирање хаотичних динамичких система.

Мотивација за примену модела машинског учења у ову сврху је велики напредак у развоју алгоритама машинског учења, као и рачунарског хардвера за њихово тренирање и евалуацију. Такође, модели машинског учења, а поготово неуралне мреже, показују способност да моделирају висок степен апстракције и генерализације, што се добро уклапа са непредвидивом природом хаотичних динамичких система и симулације временских услова.

У овом раду ће бити урађена анализа појма хаотичних динамичких система, као и више њихових примера. Након тога ће бити описани модели машинског учења одабрани за тестирање на овом проблему, као и специфичности овог проблема у односу на уобичајене проблеме који се решавају методама машинског учења. На крају ће бити спроведено тестирање одабраних модела и њихови резултати упоређени са другим методама предикције, као и анализа утицаја смањења прецизности нумеричког рачуна при евалуацији модела на резултате.

2. ХАОТИЧНИ ДИНАМИЧКИ СИСТЕМИ

2.1. Динамички системи

Динамички системи су математички појам који представља функцију која описује кретање тачке у простору кроз време [5]. У сваком тренутку динамички систем скупом вредности параметара описује стање система, које се може протумачити као координате тачке у вишедимензионалном простору. Функција промене која описује динамички систем узима тренутно стање система и израчунава промену сваког од параметара приликом преласка система у наредно стање.

$$\dot{x} = v(x)$$

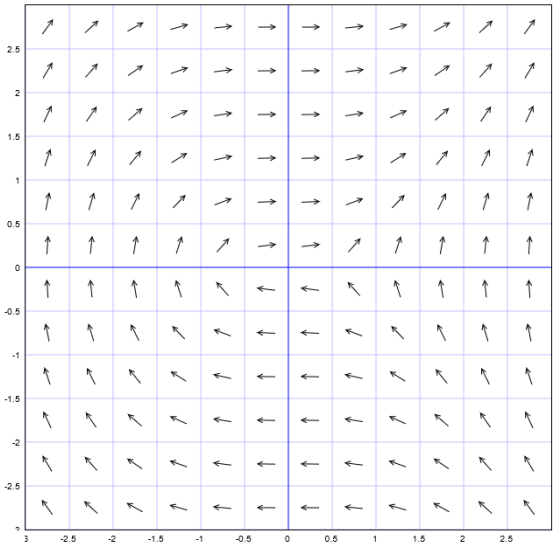
Најчешћи и најједноставнији облик динамичког система је линеарни динамички систем, који линеарном функцијом пресликава тренутно стање у информацију о трајекторији система у том тренутку.

$$\dot{x} = v(x) = Ax_n + b$$

На овај начин могуће је израчунати трајекторију система у било којој тачки у простору и тако дати интуитиван визуални опис дводимензионалног система, или система са два параметра.

У следећем примеру промена прве променљиве је директно сразмерна другој, док промена друге зависи од квадрата прве. На слици је лако уочити како се ове зависности осликавају на графичком приказу трајекторије динамичког система.

$$\frac{dx}{dt} = 2y$$
$$\frac{dy}{dt} = x^2$$



Слика 1. На слици су приказани смерови у којима се пружају трајекторије за простор око координатног почетка за изнад приказани динамички систем са две променљиве.

Поред стандардних континуалних функција промене, могуће је систем описати и функцијама које тренутно стање пресликавају у наредно, са фиксним временским размаком. Приликом коришћења рачунара за симулацију динамичких система, неопходно је дати динамички систем дискретизовати, при чему се уводе нумеричке грешке.

$$x_{n+1} = Ax_n + b$$

Функција промене динамичког система може бити и недетерминистичка, односно зависити од неког случајног догађаја, али се таквим системима нећемо бавити у овом раду.

Још једна особина динамичког система је периодичност. Систем је периодичан ако се после коначног броја корака враћа у почетно стање, из чега следи да ће се у исто стање периодично враћати заувек. Такође, конкретно почетно стање неког система може бити периодично, иако то не важи за било које стање тог система.

Систем може имати и тачке конвергенције у којима је вредност промене стања једнака нули, односно када стање система дође у ту тачку, у њој ће остати заувек. Исти систем може имати почетна стања која су периодична, конвергирају и дивергирају.

Динамички системи се често користе у природним наукама, како би се објасниле или описале различите природне појаве. Када се појава опише задовољавајућим динамичким системом, отвара се могућност коришћења добијеног система за предвиђање будућих стања тог система. Треба имати на уму да због природе динамичких система, није могуће једноставно израчунати вредност параметара система у било којем будућем тренутку, већ је потребно израчунати функцију промене довољан број пута док се не стигне од почетног до жељеног временског тренутка. Из угла математичара ово не представља велики проблем, док се из угла инжењера јављају проблеми акумулације нумеричке грешке израчунавања, као и времена и рачунарских ресурса потребних за жељена израчунавања.

Често приликом изучавања појава описаних динамичким системима немамо све потребне информације. Код сложенијих динамичких система, може се десити да нисмо у могућности да измеримо вредност неког од параметара који је потребан за одређивање почетног стања система. Такође, комплексност система нас може спречити да прецизно одредимо функцију промене система, што нам једино оставља могућност да покушамо што прецизније да је проценимо.

2.2. Теорија хаоса

2.2.1. Увод у теорију хаоса

За неке од проучаваних динамичких система је утврђено да испољавају занимљиве и специфичне особине. Неки детерминистички системи који би се састојали од релативно једноставних функција промене, испољавају понашање које изгледа потпуно насумично, иако је потпуно предодређено. Једна од последица овога је да за почетна стања чији су почетни параметри веома блиски, систем за веома кратко време потпуно дивергира. То значи и да су ови системи јако осетљиви на нумеричке грешке при израчунавању. Ови системи су названи хаотични динамички системи *CDS* (*Chaotical Dynamical Systems*).

Код подскупа ових система, пронађен је још један феномен. За било које почетно стање, систем би завршио у осцилацији између једне или више тачака у простору које су назване атрактори. Атрактори су фиксни за одређени *CDS* и разликују се од тачака конвергенције по томе што стање система никада не долази до њих, већ кружи око њих и наизглед насумично прескаче са једног атрактора на други.

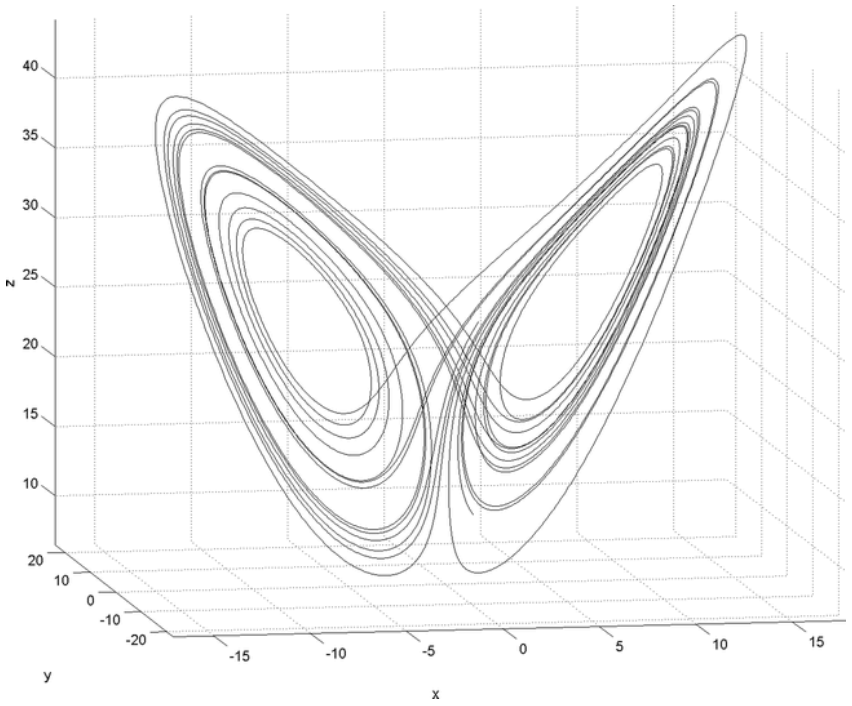
2.2.2. Лептир

Најпознатији представник ових система је такозвани Лоренцов атрактор, који је настао упрошћавањем модела који је био дизајниран да опише промене у атмосфери и одличан је пример за анализу особина ових система [6].

Баш овај феномен је инспирација за популарни ефекат лептирових крила који каже да махање крила лептира може да изазове ураган да другом крају света [7]. У популарној науци значење овог исказа је погрешно протумачено у буквалном смислу. Исказ заправо само жели да укаже како непознавање најситнијих почетних параметара система може потпуно да нам онемогући предвиђање будућих стања система.

Једно стање основног Лоренцовог система садржи три параметра и функција промене се састоји од три обичне диференцијалне једначине. Ове диференцијалне једначине су такође параметризоване са три параметра који се могу варирати σ, ρ и β, а посебно интересантно понашање испољавају за вредности параметара 10, $8/3$ и 28.

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

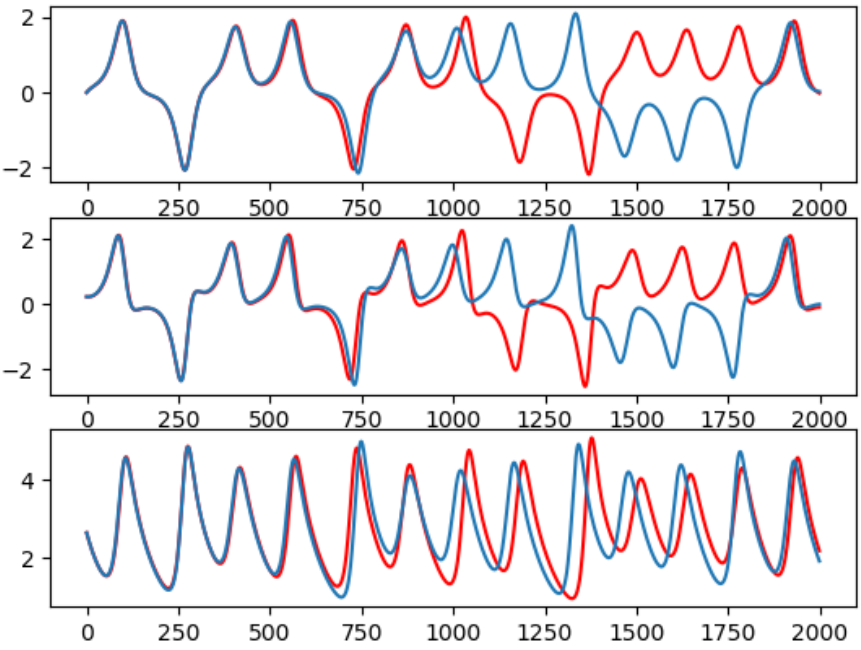


Слика 2. На слици је приказана једна могућа трајекторија коју описује основни Лоренцов систем. Све трајекторије започете из било ког иницијалног стања система конвергирају тако да описују приказану форму.

На слици се може видети путања коју описује тачка која се креће по правилима Лоренцовог система. Као што се може видети, тачка описује један или више кругова око једног атрактора, да би затим прешла на други и поновила исту акцију. Једноставност овог система у смислу једначина и тродимензионалног простора нам омогућава да интуитивно разумемо особине овог система. Код компликованијих система који не могу бити лепо графички приказани, можемо се ослонити само на статистичку анализу.

Једна од битних особина ових система је непериодичност. Иако ће тачка бесконачно осциловати између атрактора, она се никада неће два пута наћи на истом месту у простору, односно у истом стању. Такође, узимајући у обзир да ново стање система зависи искључиво од претходног стања, две трајекторије, из различитих почетних стања, се никада не могу пресећи.

Један од начина да се опише понашање Лоренцовог система за одређено почетно стање је бинарни низ нула и јединица, где нула значи да је систем направио круг око једног атрактора, а јединица око другог. На пример два круга око једног, три око другог и још два око првог би била представљена бројем 0011100. На овај начин се врло лепо може видети ефекат ситне промене у почетном стању, где за најситнију промену почетног стања, бинарне представе потпуно дивергирају већ после првог или другог круга. Ово је и одличан начин мерења успешности предвиђања будућих стања система, тако да се не мора обраћати пажња на ситне грешке у предвиђању појединачних стања, докле год модел за предвиђање, предвиђа круг око исправног атрактора.



Слика 3. На слици су плавим линијама приказане вредности све три променљиве основног Лоренцовог система на сегменту од 2000 временских корака. Црвеним линијама је приказан резултат предикције.

На графику се јасно могу видети осцилације по свакој од оса система и може се рећи да је предвиђање добро докле год оно не оде на потпуно супротну страну на некој од оса.

2.2.3. *Лоренц 96*

Поред оригиналног Лоренцовог система, постоје и многе модификоване верзије. Једна од најпознатијих је Лоренц 96 хаотични систем, који је добио име по години када је откривен. Овај систем има доста више променљивих, као и параметара система, што његово понашање чини доста сложенијим. Због својих карактеристика постао је стандардни систем за тестирање модела за предикцију хаотичног понашања.

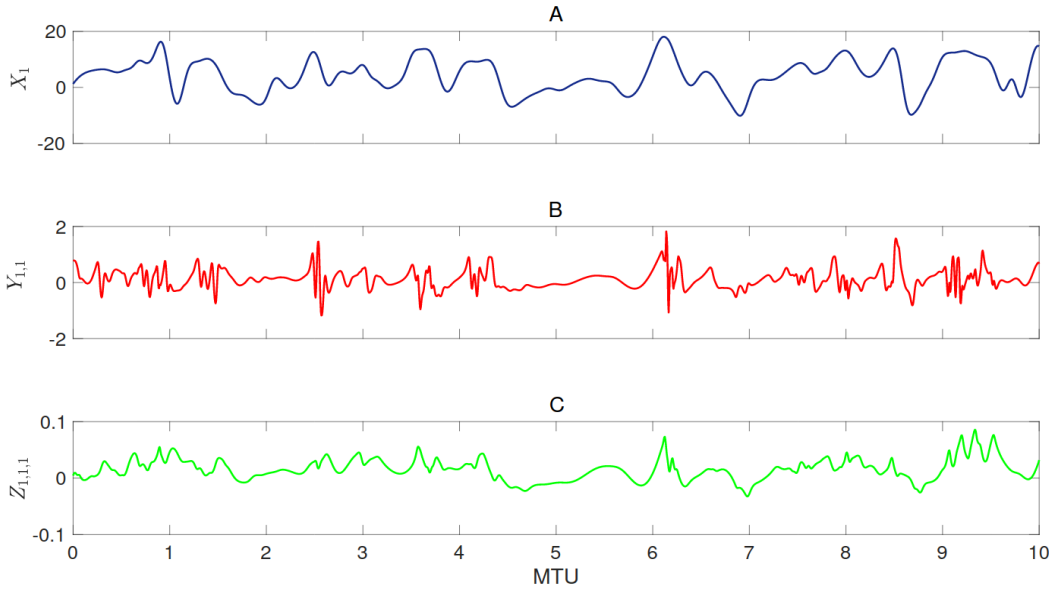
У наставку је дата варијанта овог система која је коришћена у овом раду.

$$\frac{dX_k}{dt} = X_{k-1}(X_{k+1} - X_{k-2}) + F - \frac{hc}{b}\sum_j Y_{j,k}$$

$$\frac{dY_{j,k}}{dt} = -cbY_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - cY_{j,k} + \frac{hc}{b}X_k - \frac{he}{d}\sum_i Z_{i,j,k}$$

$$\frac{dZ_{i,j,k}}{dt} = edZ_{i-1,j,k}(Z_{i+1,j,k} - Z_{i-2,j,k}) - geZ_{i,j,k} + \frac{he}{d}Y_{j,k}$$

У претходним формулама индекси i, j и k се крећу између 1 и 8, што значи да постоји 8 X, 64 Y и 512 Z променљивих. Како би хаотични системи из реалног света били веродостојније моделирани, након генерисања временских података за овај систем, задржане су и коришћене само вредности 8 X променљивих. На овај начин изгубљене су информације о остатку система и симулирана је немогућност мерења свих улазних параметара реалног система.

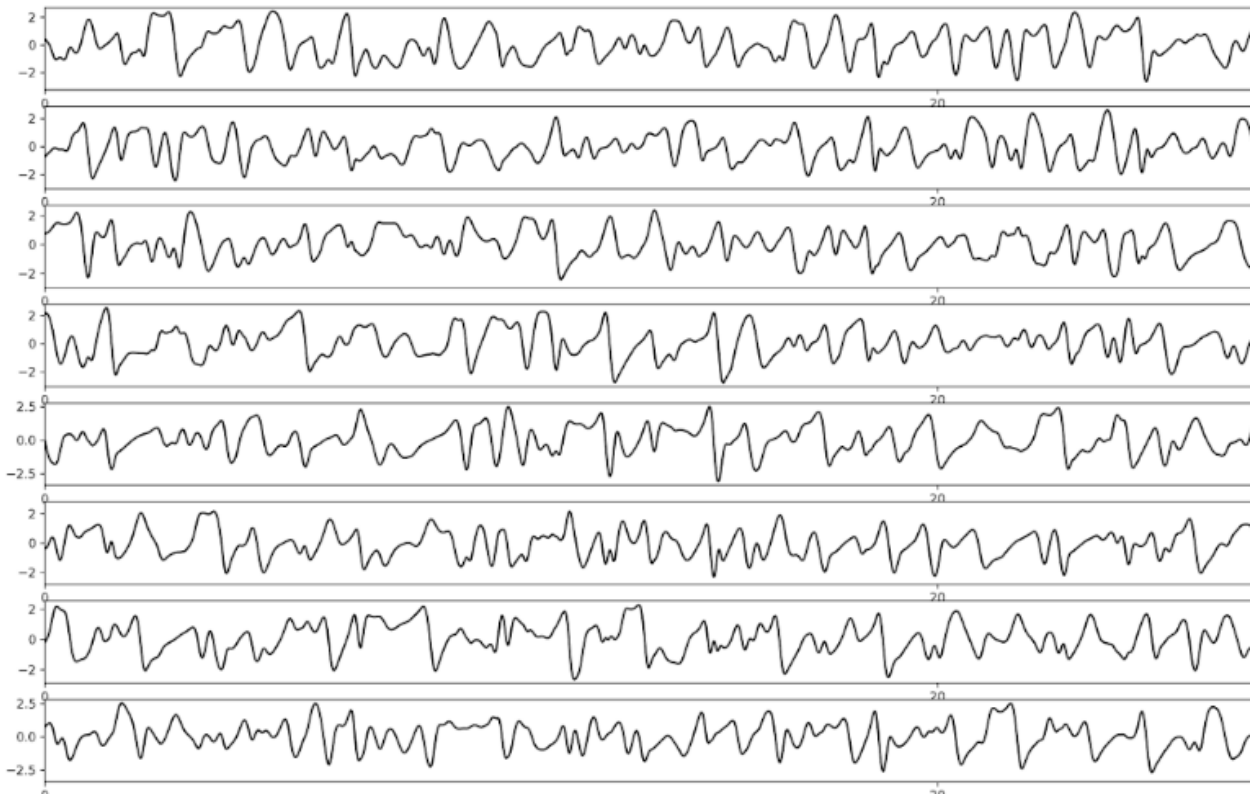


Слика 4. Вредности по једне од *X*, *Y* и *Z* променљивих на насумичном сегменту временске прогресије

На претходном графикону се може уочити да променљива X има правилније понашање, са већом амплитудом и дужим периодом осцилација, што указује на то да су њом представљене основне вредности система које су од највећег интереса за предвиђање. Са друге стране променљиве Y и Z представљају ситне локалне финесе унутар система.

На наредном графикону су приказане осцилације кроз време за 8 одабраних параметара система који су коришћени у овом раду. Ту се може приметити колико је теже уочити или

пратити било какве зависности између ових варијабли, у односу на једноставнији систем са три променљиве.



Слика 5. На слици су приказане вредност 8 основних променљивих динамичког система Лоренц 96. Може се уочити да је њихово понашање драстично компликованије од система са три променљиве и није могуће интуитивно извући никакве закључке.

3. МОДЕЛИ МАШИНСКОГ УЧЕЊА ЗА МОДЕЛОВАЊЕ ВРЕМЕНСКИХ ПРОГРЕСИЈА

Циљ овог рада је моделирање и предикција система представљених у претходном поглављу помоћу метода машинског учења. Временом су развијени разни приступи за моделирање ових система [8]. Ипак, још увек није било успешних покушаја да се ови системи квалитетно моделирају помоћу модела машинског учења, тако да овај рад покушава да донесе напредак на том пољу.

3.1. Основни концепти машинског учења

Машинско учење је област која се бави развијањем алгоритама чија се функционалност обраде улазних података и израчунавања излаза програмира посредно, аутоматским извлачењем правилности из великог борја унапред припремљених примера за тренирање [9]. Начин на који се ово постиже је тако што се дефинише функција која пресликава улаз жељеног облика на излаз жељеног облика и она представља модел. Ова функција треба да буде што генералнија и да се њено понашање ослања на мањи или већи број тежинских параметара, у зависности од тежине проблема који се решава.

Након функције пресликавања треба дефинисати и начин за евалуацију квалитета пресликавања које функција извршава. Ова вредност зависи од конкретног скупа тежинских параметара и података за тренирање модела. Модел се тренира подешавањем тежинских параметара тако да се минимализује акумулирана функција грешке над подацима за тренирање. У општем случају смањење грешке над подацима за тренирање указује да модел учи да генерализује фундаменталне зависности у подацима, што му касније омогућава да са великом прецизношћу евалуира улазне податке које није раније сретао међу подацима за тренирање.

Постоји велики број приступа за дефинисање функције пресликавања и функције грешке у зависности од конкретног проблема који се решава. Од сада ћемо се фокусирати на приступе који се могу применити на наш проблем предикције динамичких система.

3.2. Рекурентни модели машинског учења

У општем случају за проблеме у којима се моделира некаква прогресија вредности најбоље се показују такозвани рекурентни модели. Као што се може видети из природе наших података, ови модели на улазу примају претходно стање система, а на свом излазу дају стање система у следећем временском кораку. Из овога се може разумети атрибут „рекурентни“, односно модели у којима се излаз враћа на улаз и тако у круг. На основу тога се може приметити да они не моделују просто пресликавање улазних вредности на излазне, већ извесни ток тих вредности.

Одатле долази идеја да се предикција следећег стања може значајно унапредити ако би се моделу дао већи контекст од једне претходне вредности стања система, иако је она технички довољна приликом математичке евалуације оригиналног система. Једноставан начин да се унапреди модел је да се уместо једног претходног стања система, на улаз модела доведу неколико претходних стања. Ово би значајно унапредило контекст који модел има, али постоје

и проблеми као што су велико усложњавање модела због много већег броја улаза, или ограничење контекста на одабрани број претходних стања.

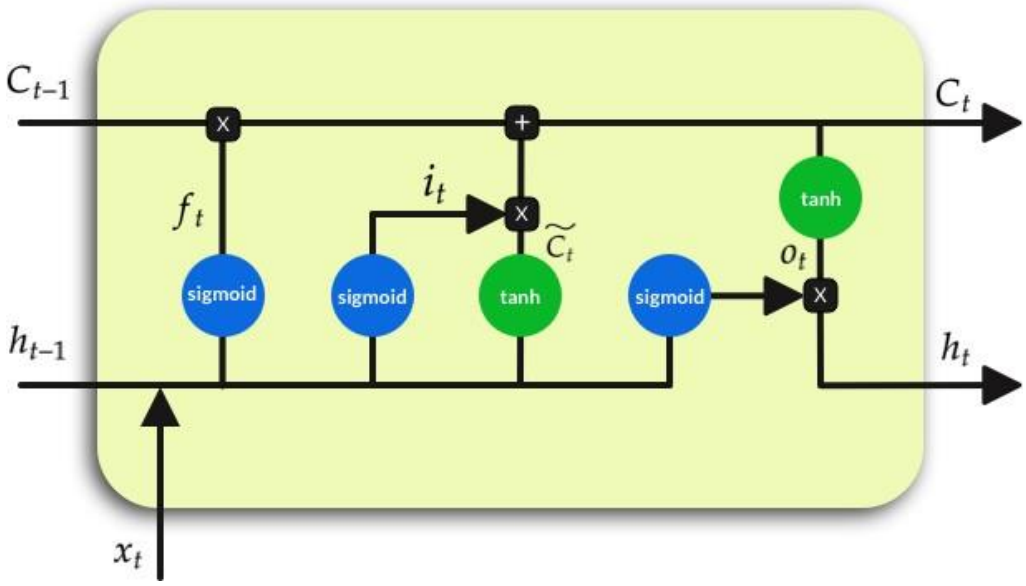
Решење које се показало као универзално успешно, је да модели поред истренираних и фиксираних тежинских параметара садрже и параметре тренутног стања модела који се ажурирају током евалуације модела како би при следећој евалуацији представљали тренутно стање модела. У општем случају, приликом сваке евалуације стари параметри стања модела мало избледе, а затим се ажурирају новим улазним подацима. Модел овако може да неки податак, који је приметио као битан, веома дуго пропагира кроз наредне евалуације.

Модели који су коришћени у нашим експериментима су верзије *LSTM* (*Long-Short Term Memory*) [10] и *ESN* (*Echo-State Networks*) [11]. У наредним поглављима биће објашњена њихова структура.

3.3. LSTM модели

LSTM рекурентне мреже су универзални индустријски стандард за рекурентне моделе и добро се показују у најразличитијим применама приликом предвиђања прогресија [12]. У овом раду користиће се стандардна верзија овог модела као основа за упоређивање резултата.

LSTM представља конкретизацију претходно објашњеног решења за рекурентне неуралне мреже и овде ће бити објашњен у контексту овог рада. Основни градивни блокови су улаз модела, стање модела, операција заборављања *(forget gate),* операција меморисања *(input gate),* генерисање излаза *(output gate)* и излаз модела.



Слика 6. Структура *LSTM* модела

Улаз модела је вектор који у нашем случају представља једно стање система који моделирамо. Он се иницијално поставља на неко познато стање система, а затим током евалуације узима вредности претходних излаза из мреже. Стање модела је такође вектор истих димензија који се у свакој евалуацији ажурира и користи при израчунавању излаза модела.

Први корак евалуације је фаза заборављања. Она представља засебну неуралну мрежу која пресликава улаз у низ коефицијената између 0 и 1, којима ће се скалирати стање система.

Односно, она у зависности од улазних података одлучује који елементи стања модела треба да сачувају, а који треба да драстичније умање своју вредност.

Други корак је фаза меморисања у којој се ефекат новог улаза додаје у стање модела. Ова фаза се састоји од две засебне неуралне мреже. Прва у зависности од улаза, за сваки елемент стања модела, израчунава вредност између -1 и 1 коју треба додати на тај елемент. Друга мрежа израчунава коефицијенте између 0 и 1 који скалирају излаз претходне мреже пре додавања на стање модела. Ове две мреже раде добро у пару зато што се прва фокусира на учење нумеричке промене стања у зависности од улаза, док друга учи да процени које елементе треба приоритизовати, а које игнорисати.

Након што стара меморија избледи и додају се нови подаци, остаје да се израчуна излазна вредност. Ово се постиже помоћу још једне неуралне мреже која од улазних података израчунава коефицијенте између 0 и 1 који се множе са елементима меморије модела и генеришу излаз. Односно, излаз се добија скалирањем меморије модела излазом излазне неуралне мреже.

На шеми Слика 6 је приказана структура *LSTM* модела. Евалуација овог модела кроз низ вредности у временској прогресији може се замислити као ређање истих блокова један за другим. Модел са шеме се може представити скупом једначина, где су параметри који се уче тренирањем тежински параметри означени са W и U и *bias* параметри b:

$$f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c_t} = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c_t}$$

$$h_t = o_t * \tanh(c_t)$$

3.4. *ESN* модели

ESN су још један приступ рекурентним неуралним мрежама. Доста је мање заступљен од *LSTM* мрежа зато што није толико универзалан, али има своје место у одређеним применама. Конкретно у сфери предвиђања динамичких система се показао обећавајуће, тако да ће се овај рад фокусирати на анализу овог приступа.

3.4.1. ESN архитектура

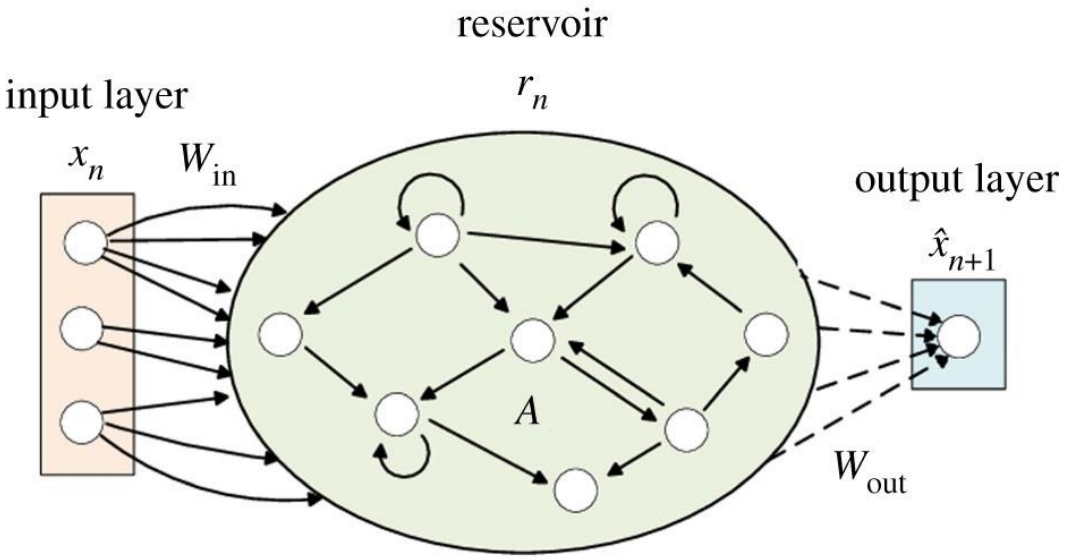
Други назив за *ESN* моделе је резервоарско израчунавање (*reservoir computing*), што долази од архитектуре ових модела, која се састоји од великог низа који представља стање модела. Приликом сваког корака евалуације модела, извршава се операција мапирања вредности између елемената тог низа, односно “претицање” између резервоара.

Конкретно, *ESN* се састоје од улазног низа, улазног мапирања, резервоара, излазног мапирања и излазног низа. Као и код *LSTM* мрежа, улазни и излазни подаци представљају претходно и наредно стање симулираног система и током предикције излазни низ се прослеђује назад на улаз.

Резервоар се састоји од жељеног броја неурона, од којих сваки носи једну вредност и заједно представљају стање модела. Ови неурони не представљају уобичајени слој (*layer*) неуралне мреже, већ скуп насумично повезаних вредности. Та повезаност представљена је усмереном матрицом пресликавања, која дефинише који неурони су међусобно повезани и са којим коефицијентом пресликавања. Приликом сваког корака евалуације ажурирају се вредности резервоара применом матрице пресликавања.

Како би се резервоар повезао са улазним и излазним подацима, постоје улазна матрица која пресликава низ из димензије улаза у димензију резервоара и излазна матрица која пресликава низ из димензије резервоара у димензију излаза.

Веома интересантна особина ових модела је количина насумично одабраних параметара који нису подложни тренингу модела. Заправо матрица за мапирање улаза, као и матрица пресликавања резервоара се одређују потпуно насумично, у одређеним оквирима прихватљивих вредности, док је матрица излаза једини део модела који се тренира. Још једна особина матрице мапирања која је експериментално утврђена је да се најбољи резултати добијају када је она веома слабо попуњена (*sparse matrix*), највероватније јер у том случају лепше моделира претицање између резервоара, а такође омогућава и коришћење знатно већег резервоара.



Слике 7. Шема структуре *ESN*

На претходној слици се може видети шема описане структуре *ESN* модела, укључујући улазни слој, рекурентни резервоар и излаз модела.

3.4.2. ESN предвиђање

За описивање модела користићемо следеће ознаке. Улазни подаци су представљени променљивом X_n, где n означава временски тренутак од почетка скупа познатих података. Улазна и излазна матрица су означене са W_{in} и W_{out}. Стање резервоара је означено са r_n, где n такође означава временски тренутак. Матрица пресликавања је представљена са D.

Први корак у израчунавању предвиђања је делинеаризација стања резервоара. Ова операција се показује као потребна зато што су све остале операције унутар модела линеарне трансформације. Делинеаризација се извршава тако што се свака друга вредност у низу стања резервоара помножи са својим претходником.

$$r_{aug\,n,i} = \begin{cases} r_{n,i}, & i \bmod 2 = 0 \\ r_{n,i} * r_{n,i-1}, & i \bmod 2 = 1 \end{cases}$$

Излаз модела се затим једноставно рачуна множењем делинеаризованог стања резервоара са наученом излазном матрицом.

$$output_n = W_{out} * r_{aug}$$

На крају, ажурира се стање резервоара применом матрице пресликавања на стање резервоара и улазне матрице на улазне податке. На добијени резултат се примењује хиперболички тангенс како би се вредности у резервоару држале у фиксном опсегу.

$$r_{n+1} = \tanh\left(A * r_n + W_{in} * output_n\right)$$

У случају *ESN*, стање резервоара је у сржи модела и за разумно предвиђање оно се мора довести у валидно стање пре почетка предвиђања. Овај процес се назива загревање резервоара (*warm-up).* Загревање се постиже извршавањем неколико модификованих корака предикције. За то је потребно знати одређени број претходних стања модела (не само једно). Предвиђање се извршава тако што се на улаз доводе познате улазне вредности једна по једна и ажурира се стање резервоара, а добијени излази се занемарују.

3.4.3. ESN тренирање

Грешка предвиђања или функције цене (*cost function*) над целим сетом података за тренирање се израчунава уобичајено, као сума квадрата разлике израчунатих и стварних вредности стања система на сету за тренирање, уз коришћење регуларизације.

$$cost(W_{out}) = \sum_{i=1}^{n} (W_{out} * r_{aug\,n})^2 + \alpha(W_{out})^2$$

Као што је већ поменуто, једини део модела који се тренира је излазна матрица W_{out} и она се проналази тако да се задовољи минимална регуларизована грешка предвиђања.

$$W_{out} = argmin_{W_{out}} \sum_{i=1}^{n} (W_{out} * r_{aug\,n})^2 + \alpha(W_{out})^2$$

Лепа особина ове архитектуре модела је постојање формуле за директно израчунавање оптималних вредности за матрицу излаза (*one-shot learning*).

Први корак у тренирању је насумично генерисање матрица W_{in} и A. Затим се извршава велико загревање резервоара проласком кроз цео сет података за тренирање, а сва стања резервоара кроз која је модел прошао се агрегирају у матрицу S_{aug}, у својој делинеаризованој форми. Након тога, употребљава се следећа формула за израчунавање W_{out}.

$$W_{out} = (S_{aug} * {S_{aug}}^T + \delta * I)^{-1} * S_{aug} * X$$

У претходној формули I представља јединичну матрицу, а X улазне податке. Иако је у питању учење у једном кораку, оно укључује споре операције над матрицама, тако да за резервоаре са великим бројем неурона тренинг може потрајати колико и тренинг уобичајених неуралних мрежа.

3.4.4. Агрегација резултата више модела

Приступ за унапређивање квалитета оваквих система је коришћење комбинације резултата ансамбла модела приликом предвиђања (*ensemble models*). На пример, могуће је истренирати више модела који се разликују у садржају сета података за тренирање, или просто са различито иницијализованом основом за насумичне делове модела, што је веома ефективно у случају *ESN* модела.

Експериментално је показано да није добро пуно варирати саме параметре архитектуре модела јер онда коначни модели неће имати исте генералне перформансе предвиђања и лошији модели ће само покварити квалитетније резултате бољих модела.

3.5. Закључак о моделима

Одабрани модели представљају одабир најуспешнијих метода из области машинског учења у решавању проблема сличних оном којим се овде бавимо.

Циљеви овог експеримента су да се утврде релативне перформансе између тестираних приступа, као и да се процени апсолутни квалитет модела машинског учења за примену на овој врсти проблема. Очекивање од примене модела машинског учења у овој области је велико унапређење у перформансама и у генерализацији приступа проблему, који је до сада био решаван локално и хеуристички. Са друге стране, биће веома тешко постићи резултате које даје тренутни приступ, апстрахованим приступом машинског учења.

У наредном поглављу биће представљени резултати постигнути приликом тестирања претходних модела.

4. ЕВАЛУАЦИЈА МОДЕЛА

У овом поглављу биће представљен начин евалуације модела и постигнути резултати. За тренирање и тестирање одабраних модела коришћен је програмски језик *Python.* За *LSTM* мреже је коришћена библиотека *Keras,* док су *ESN* модели имплементирани коришћењем основних структура језика *Python.* Рад је примарно фокусиран на испитивање *ESN* модела, док су *LSTM* модели коришћени за упоредну анализу.

4.1. Извршавање експеримената

Приступ евалуацији модела је био следећи. Прво су прикупљени или генерисани потребни подаци. Конкретно, овај рад се фокусира на анализу предвиђања на систему Лоренц 96 и за те потребе је употребом Рунге-Кута [13] интегратора генерисан сет података дужине једног милиона временских корака једне трајекторије. Овај сет података се испоставио као довољно велики да се издвоје подаци за тренирање и тестирање модела. Највећи сетови за тренирање који су испробани нису показали квалитетније резултате од нешто мањих сетова, а рачунарски ресурси потребни за њихову евалуацију су се приближили лимиту уобичајеног корисничког хардвера.

Након прикупљања података приступило се је имплементацији модела. Направљен је робустан и високо параметризован систем који омогућава подешавање параметара за тренинг и тестирање, као и параметара самог модела. Резултати свих експеримената су појединачно чувани и укључивали су велики број различитих визуализација, о којима ће бити речи касније у овом поглављу.

Коначне верзије *ESN* и *LSTM* модела које су произвеле резултате приказане у овом раду трениране су на подацима који су садржали 300.000 временских корака, док је величина резервоара *ESN* постављена на 5000 чворова.

С обзиром да је проблематика којом се бавимо у овом раду релативно нова и неистражена, највећи део времена је потрошен у детаљној анализи различитих експеримената како би се што боље разумели коришћени динамички системи, модели машинског учења, као и њихова интеракција [14]. Поред представљања добијених упоредних резултата, циљ овог рада је и да предложи систематику приступа евалуацији модела за предвиђање хаотичних динамичких система.

4.2. Временска јединица модела

Приликом представљања резултата евалуације модела динамичких система, временске интервале је могуће представити на више начина, од којих су два коришћена у овом раду.

Први и најједноставнији начин представљања времена су јединице коришћене приликом генерисања основног сета података. Овај начин представљања омогућава интуитивно разумевање временских интервала када их поредимо са вредностима као што су број временских тренутака који се налазе у сету за тренирање, време потребно за загревање резервоара или удаљеност сегмента на коме се врши предвиђање од краја сета за тренирање. Ипак, бројање дискретних временских корака приликом генерисања података интеграцијом диференцијалних једначина коришћеног система, не пружа увид у ширу слику о датом систему. На пример, не омогућава нам да проценимо понашање и брзину промене у самом

систему, што нам је потребно како би могли да проценимо апсолутне резултате наше предикције.

Други приступ за мерење временских интервала се назива временска јединица модела *MTU* (*Model Time Unit*) [15], који зависи од временског корака коришћеног при интеграцији. У нашем примеру коришћен је временски корак од $\Delta t = 0.005$, из чега произилази да је $1MTU = 200\Delta t$. Како би се одабрао задовољавајући временски корак, ослањамо се на концепт Лајапоновог експонента. Лајапонов експонент (λ) је вредност додељена неком динамичком систему која квантификује колико брзо дивергирају две трајекторије које почињу из веома блиских тачака, чија се удаљеност означава са $\delta \boldsymbol{Z}_0$.

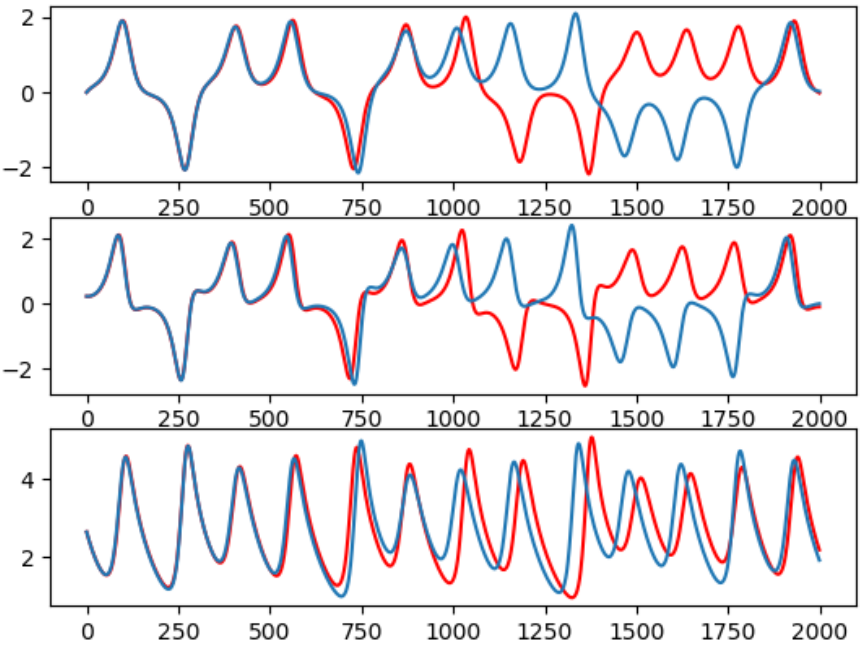
$$|\delta \boldsymbol{Z}(t)| \approx e^{\lambda t} |\delta \boldsymbol{Z}_0|$$

$$\lambda = \lim_{t \to \infty} \lim_{|\delta \boldsymbol{Z}_0| \to 0} \frac{1}{t} \ln \frac{|\delta \boldsymbol{Z}(t)|}{|\delta \boldsymbol{Z}_0|}$$

У нашем случају $1MTU \approx 4{,}5/\lambda$, што је еквивалентно временском интервалу од 2 до 4 дана у реалној атмосфери [15]. У наредним поглављима бројне вредности и графикони ће користити једну од ова два метода за означавање временских интервала.

4.3. Хоризонт предикције

На графикону Слика 8 је пример предикције на једноставном Лоренцовом систему са 3 промењљиве и на њему се могу уочити особине ових модела које се провлаче и кроз остале тестове. Наиме, може се приметити да је на почетку предвиђање веома прецизно, затим почиње мало да одступа од реалних података. У једном тренутку предвиђање дивергира у потпуно супротну страну од очекиваних вредности. Занимљиво је да чак и након дивергирања модел наставља да производи излаз у очекиваном опсегу и у облику који не наговештава да је дошло до грешке, све док се подаци не упореде са очекиваним вредностима.



Слика 8. Пример предвиђања на систему Лоренц 63

Као што је поменуто у претходној секцији, иако модели које користимо нису потпуно уобичајени, њихов приступ тренирању минимализацијом квадратне греше предикције у сваком појединачном кораку је потпуно уобичајен. Са друге стране за нашу примену, овај начин израчунавања грешке не описује наш циљ на оптималан начин. За разлику од стандардне употребе неуралних мрежа за апроксимацију функције пресликавања једног појединачног улаза у одређени излаз, у нашем случају потребан нам је што дужи низ прихватљиво прецизних предвиђања.

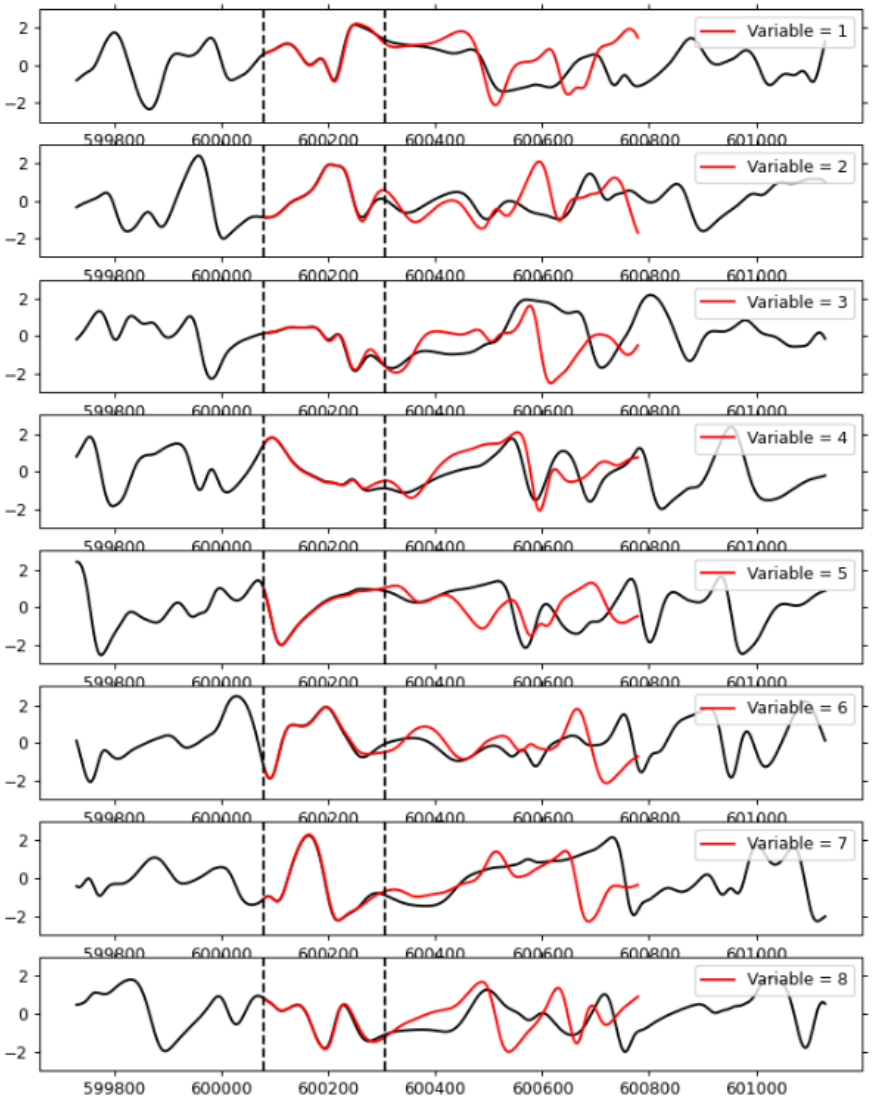
Управо број корака предикције са прихватљиво прецизним предвиђањима називамо хоризонт предикције (*prediction horizon*).

Гледајући графикон Слика 8, у нашем случају главни циљ модела би требао да буде да што дуже прати смер осцилације, док ситна одступања у појединачним стањима не представљају проблем сама по себи. Овде може да дође до проблема уколико локална грешка прекорачи задати лимит грешке за хоризонт предикције, иако модел и после тог корака наставља да задовољавајуће прати осцилације система.

4.3.1. Интерактивна анализа хоризонта предикције

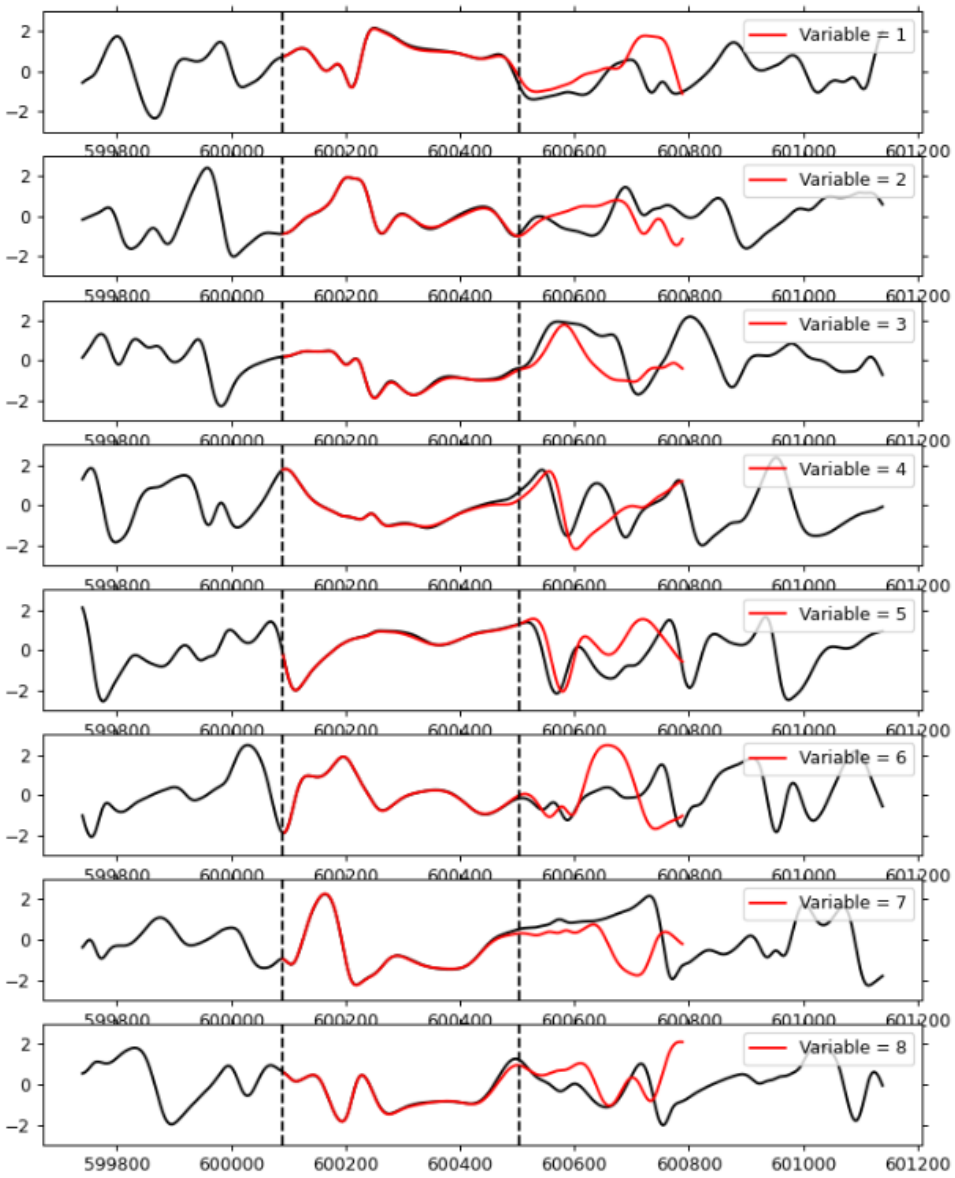
Један од првих приступа анализи резултата био је креирање интерактивне скрипте помоћу алата *jupyter notebook* који омогућава да се *Python* скрипте извршавају у веб претраживачу из делова, уместо целе скрипте од једном. Ово омогућава и креирање једноставног интерактивног интерфејса. Код скрипте је дат у прилогу „Програмски код скрипте за анализу направљених предикција“.

Скрипта функционише тако што учитава модел и основне податке са задатим параметрима, а затим извршава предикцију учитаним моделом почевши од одређеног временског тренутка унутар сета података за тестирање. Затим приказује како предвиђене вредности на почетку прате истините податке, а затим одступају од њих, омогућавајући детаљну анализу понашања модела на конкретном примеру. Скрипта омогућава и кретање кроз сет података за тестирање, као и сет података на којима је модел треniран један по један временски корак, што омогућава одличан увид у значајне разлике у квалитету предикције између веома блиских почетних вредности.



Слика 9. Предвиђање *ESN* модела у тест сету на удаљености од 300.080 од краја тренинг сета

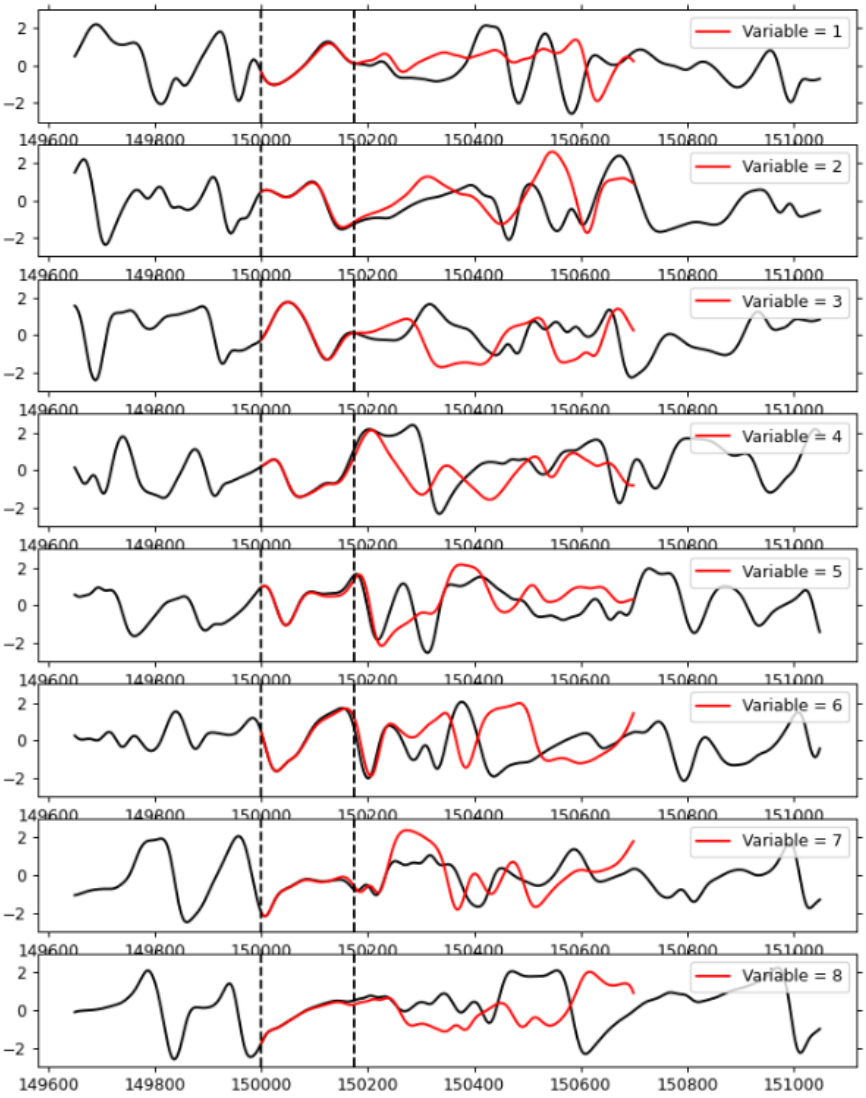
На графикону Слика 9 види се пример предикције на тест сету, почевши од 300.080 временских корака удаљености од краја сета коришћеног за тренирање у коме се налази првих 300.000 временских корака од почетка укупног сета података. Лева вертикална линија означава почетак предвиђања, а десна временски хоризонт предикције, односно тренутак када је грешка прекорачила задати лимит, који у овом случају има вредност 227 временских корака. Црвена линија представља 700 временских корака предвиђања и може се видети како временом све више губи сличност са основним подацима.



Слика 10. Предвиђање *ESN* модела у тест сету на удаљености од 300.090 од краја тренинг сета

На графикону Слика 10 може се видети еквивалентна представа примера предикције, која је извршена са почетком који се налази на само 10 временских корака разлике од претходне. Линија хоризонта предикције у овом примеру је на 414 временских корака, што значи да је предикција скоро дупло квалитетнија од претходне, иако су обе започете из врло блиских стања (на слици се практично не може приметити разлика у почетним стањима).

Овај феномен указује на то да постоје одређене особине стања из којих се предвиђање започиње, од којих зависи колико то предвиђање може да буде квалитетно, али у нашим истраживањима нисмо успели да дођемо до конкретног објашњења. Ово такође подвлачи хаотичну природу целог овог проблема и указује на комплексност детаљне анализе.



Слика 11. Предвиђање *ESN* модела на средини тренинг сета, на удаљености од 150.000 до краја тренинг сета

На графикону Слика 11 приказан је пример предикције извршене на самој средини сета за тренирање и као што се може видети квалитет предикције није виши од квалитета добијеног почевши од стања која су јако далеко од сегмента трајекторије коришћеног за тренирање. Иако један овакав пример не мора да представља правило, испоставља се да квалитет предикције статистички заиста не зависи од близине почетног стања сегменту на коме је тренирано.

Замислимо стање овог система слично понашању лептира описаног раније у тексту, односно као систем који никад не долази у стање у коме је био раније, али осцилује у трајекторијама које су веома блиске. Из тога се може закључити да модел веома лепо генерализује понашање система, бар у релативно кратким предикцијама, и да због тога удаљеност почетног стања од сегмента за тренирање не утиче на предикцију.

4.3.2. Загревање резервоара ESN модела

Помоћу претходно описаног алата урађена је анализа утицаја дужине секвенце за загревање модела на успешност предвиђања, у склопу процеса проналажења оптималних хипер параметара за коришћене моделе. Веома интересантан резултат овог експеримента је

била дужина од само три временска корака за загревање која је давала подједнако квалитетна предвиђања било којој вишој вредности броја корака за загревање. Исти резултат добијен је и тестирањем на *LSTM* моделима.

Око конкретно значи да уколико желимо да извршимо предикцију од почетног стања у временском тренутку t, довољно је да модел, који је трениран на 300.000 временских корака, загрејемо само са стањима у тренуцима $t-3$, $t-2$ и $t-1$, и одмах почнемо са предвиђањем прослеђујући у модел стање у тренутку t. Ово је веома интересантан резултат јер показује да је моделима неопходан контекст, али не превише велики.

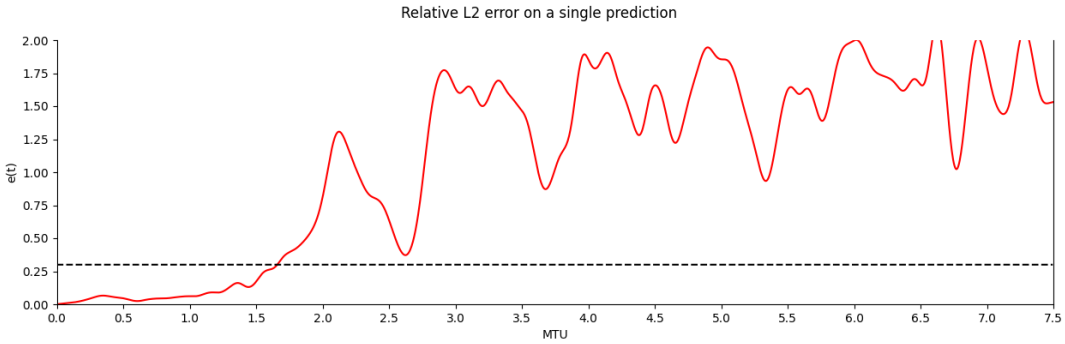
Када се у наставку рада каже да је предикција извршена од одређеног временског тренутка, подразумева се да је загревање модела извршено помоћу стања из три претходна временска тренутка.

4.4. Стандардна грешка предикције

Након што смо добили интуитивни увид у понашање наших модела, можемо прећи на основну квантитативну анализу резултата.

Почећемо од анализе грешке предвиђања у зависности од корака предикције. Наравно, овде очекујемо да у првим корацима предикције грешка буде близу нули, а да затим почне да расте, пређе задати лимит хоризонта предикције и на крају стигне до засићења које показује да је моћ предикције потпуно изгубљена.

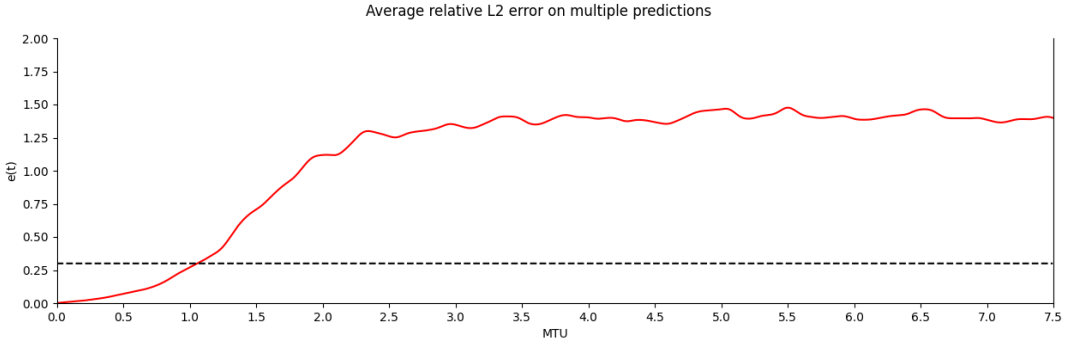
Занимљиво је почети од графикона грешке једне секвенце предвиђања, започете од насумично одабраног почетног тренутка.



Слика 12. Стандардна квадратна грешка *ESN* модела на једном насумичном предвиђању

На графикону Слика 12 може се видети да се грешка на почетку, до око 2 *MTU* понаша очекивано, односно полако расте и прелази хоризонталну линију која представља хоризонт предикције. Ипак, након тога може се видети да грешка драстично осцилује, али такође не излази из одређеног опсега. Ово се дешава зато што и излаз нашег модела остаје у валидном опсегу очекиваних стања, иако је потпуно изгубио корак са оригиналним подацима. Последица овога је да грешка остаје у разумном опсегу. Чак се и у одређеним тренуцима, када се вредност предикције потпуно случајно у својим осцилацијама приближи тачном стању система, грешка драстично смањује, стварајући осцилације.

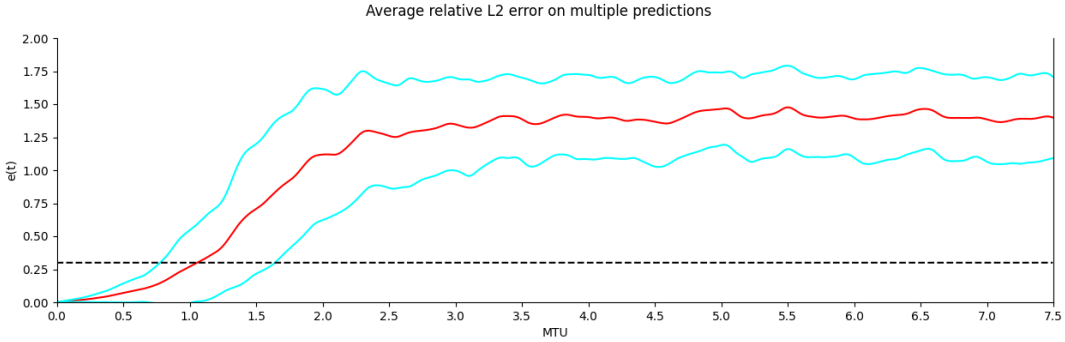
У конкретном примеру, са графикона се може видети да модел даје прихватљиве резултате 1,5 *MTU* од почетка предвиђања, након чега грешка драстично скаче.



Слика 13. Стандардна квадратна грешка *ESN* модела упросечена на 100 предвиђања започетих на разним деловима доступног сета података

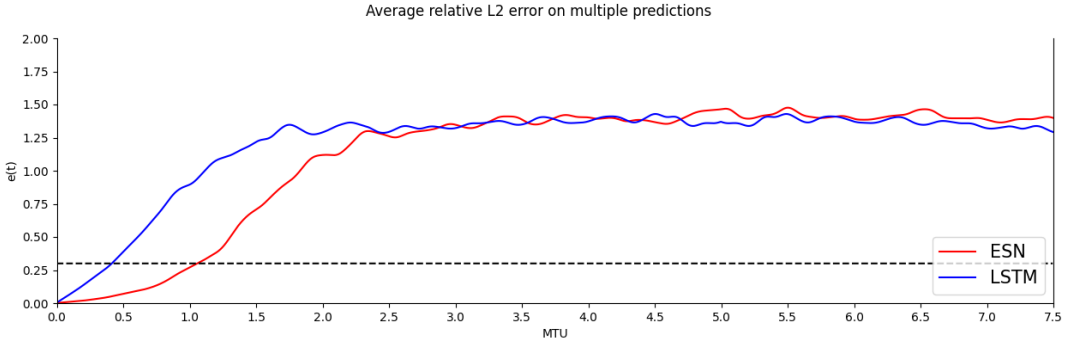
Наравно, графикон Слика 12 није само проблематичан због осцилације греше након што је она јасно прешла хоризонт предикције и модел више није употребљив, већ је проблем и коришћење само једне предикције за процену квалитета целог модела. На графикону Слика 13 се може видети иста врста грешке, усредњене на 100 предикција, што нам даје бољу слику о генералном квалитету модела.

Конкретно, просечно предвиђање прелази хоризонт предикције на 1 *MTU*, а затим просечна грешка лаганије расте, указујући да у неким случајевима квалитет предикције може бити и значајно бољи од ове средње вредности. Као што се може видети предикција са претходног графикона је значајно квалитетнија од просечне.



Слика 14. Стандардна квадратна грешка *ESN* модела упросечена на 100 предвиђања (црвена линија), са приказаном стандардном девијацијом (плаве линије)

Претходни графикон можемо проширити графиконом Слика 14, на коме се може видети и стандардна девијација грешке предикције око просечне грешке. Ово може да пружи мало бољу слику о томе како се модел понаша, али не пружа пуно корисних информација. Као што се може видети са графикона стандардна девијација грешке пре хоризонта предикције једнака је самој средњој вредности, указујући на то да постоје случајеви у којима је модел јако прецизан цео 1 *MTU*, као и случајева где се модел сналази много лошије. Након преласка хоризонта предикције сама грешка више нема сврху, али се помоћу стандардне девијације може очитати опсег осцилације грешке као последице случајног приближавања тачним вредностима.

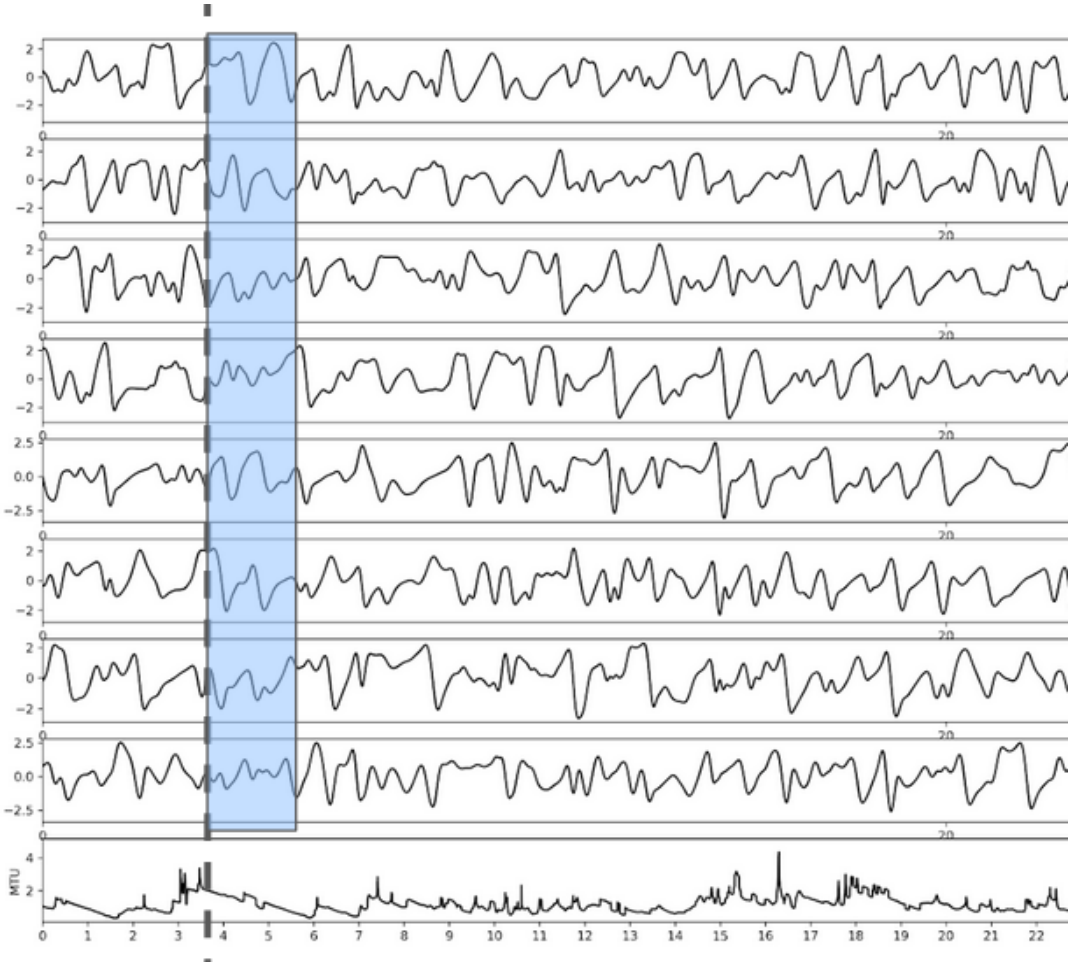


Слика 15. Стандардна квадратна грешка *ESN* и *LSTM* модела упросечена на 100 предвиђања започетих на разним деловима доступног сета података

За крај можемо да упоредимо раст просечне грешке *ESN* и *LSTM* модела. На графикону Слика 15 се јасно види да у просечном случају *ESN* модел доста дуже остаје испод лимита грешке који представља границу хоризонта предикције, али након што су оба модела достигла засићење у просечној грешци, настављају да се понашају еквивалентно.

4.5. Промене хоризонта предикције у узастопним предвиђањима

У претходном поглављу урађена је основна анализа квалитета модела као целине. У овом поглављу ћемо се бавити анализом промене вредности хоризонта предикције на сегментима предикције започетим у узастопним временским тренуцима.



Слика 16. Вредности хоризонта предикције *ESN* модела на узастопним временским тренуцима, у *MTU*

На графикону Слика 16 је комбиновани графикон који је доста коришћен приликом анализе понашања модела коришћених у овом раду и састоји се из два дела:

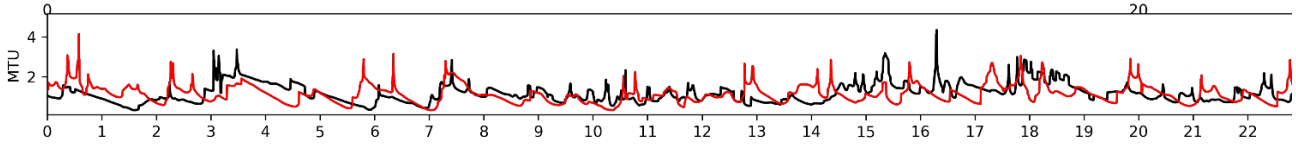
1) Првих осам редова графикона приказује вредности осам променљивих које представљају стање динамичког система кроз време. Хоризонтална оса графикона представља временски интервал од 0 до 22 *MTU* што обухвата доста велики временски интервал у контексту нашег система, с обзиром да 1 *MTU* садржи 200 временских корака.
2) Девети ред је графикон који представља дужину квалитетног предвиђања у *MTU*, пре него што грешка пређе хоризонт предикције, у зависности од временског тренутка када је предикција започета.

На слици је испрекиданом линијом означен произвољан временски тренутак. На првих осам графикона може се очитати тачно стање система у том тренутку, као и у његовој околини. Са последњег графикона може се очитати да је за одабрани временски тренутак хоризонт предикције на отприлике 2 *MTU*. Плавим правоугаоником је означена дужина једнака 2 *MTU* од одабраног тренутка од кога предикција почиње, односно, плави правоугаоник означава временски интервал у коме модел производи прецизну предикцију почевши од означеног тренутка.

Анализом последњег графикона можемо извући корисне закључке о понашању Лоренцовог динамичког система и коришћених модела. Тренд који се лако уочава је да хоризонт предикције полако и равномерно пада, често до веома ниских вредности, да би затим произвољно скочио. Ово се највероватније може објаснити неповољним стањима система која се налазе на крају тог пада, тако да модел није у стању да после тог тренутка настави да даје квалитетна предвиђања, све док не прође проблематични тренутак, или му се веома приближи. У краћој анализи овог феномена, није пронађена ни једна очигледна заједничка особина ових неповољних стања система, што остаје као простор за даљи рад.

На графикону се могу приметити и специфична иницијална стања која дају драстично боље предикције од своје околине. Најизраженији пример на сегменту приказаном на претходном графикону се налази у интервалу између 16 и 17 *MTU*, где хоризонт предикције из релативно стабилне околине са дужином око 1.5 *MTU* скаче на 4 *MTU* квалитетне предикције. Овакви примери показују потенцијални квалитет који модел може да достигне даљим унапређивањем, али са друге стране нема објашњења зашто и када се овај ефекат јавља.

У сваком случају, може се закључити да успешност предвиђања, односно тежина почетног стања предвиђања очигледно корелира са својом околином и да уколико знамо квалитет предвиђања у једном почетном стању, можемо очекивати сличну успешност и његовој околини. Са друге стране, дефинитивно постоје и необјашњене специфичности конкретних почетних стања која могу драстично да утичу на квалитет предвиђања.

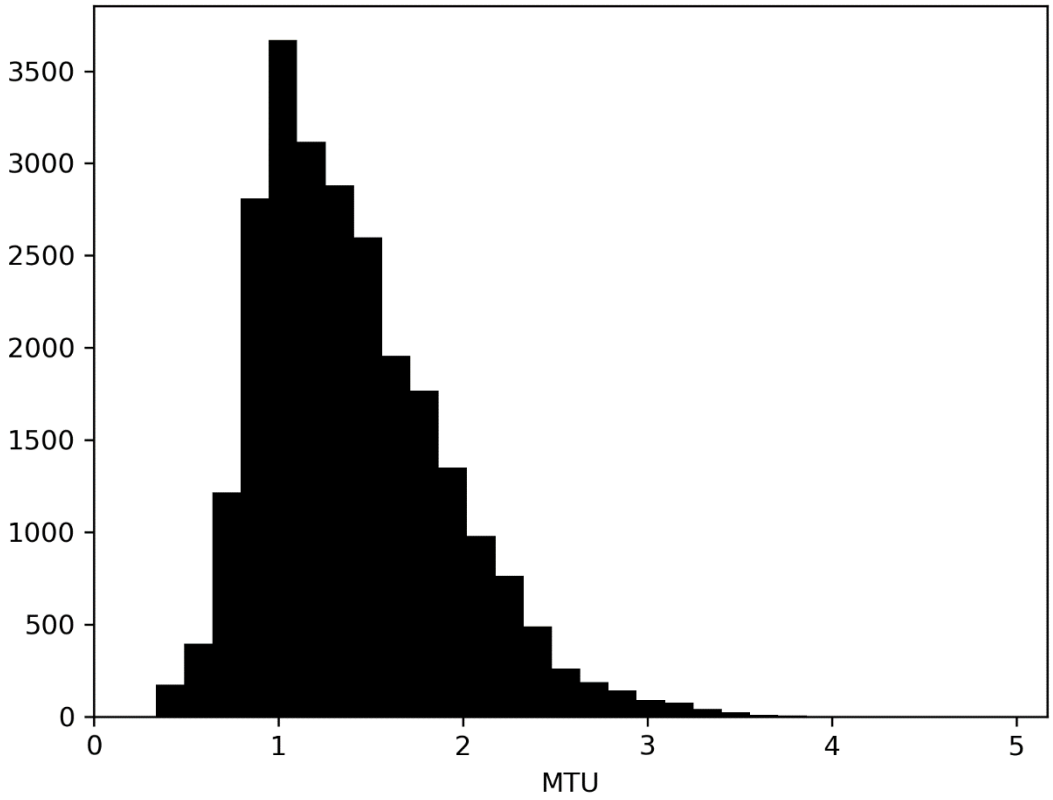


Слика 17. Вредности хоризонта предикције *ESN* (црна линија) и *LSTM* (црвена линија) модела на узастопним временским тренуцима, у *MTU*

На графикону Слика 17 представљен је исти интервал као и на претходном, али су уз резултате *ESN* модела приказани и резултати *LSTM* модела. Као што се може приметити, модели доста корелирају међусобно и искључујући специфичне изузетке, иста почетна стања се испостављају као једнако тешка за оба. Са друге стране, почетна стања са необичним скоковима у квалитету предикције се не поклапају и специфична су за сваки модел.

4.6. Разврставање предикција по квалитету

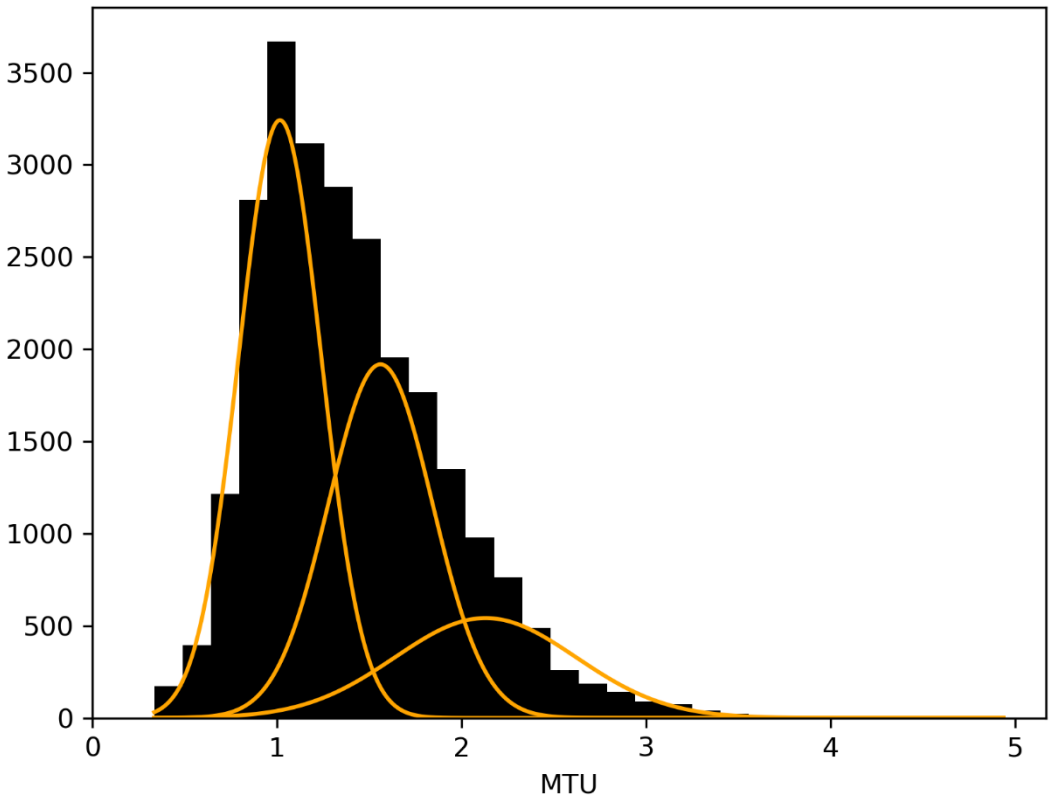
У претходном поглављу анализирана је промена хоризонта предикције на узастопним почетним стањима. У овом поглављу биће анализирана расподела дужина хоризонта предикције на целом сету података за тестирање. У ту сврху је одабрано 20.000 еквидистантних тренутака из сета за тестирање, израчунати су хоризонти предикције и приказани су на хистограму.



Слика 18. Хистограм дужина хоризонта предикције *ESN* модела

На графикону Слика 18 приказан је хистограм вредности хоризонта предикције на 20.000 предвиђања извршених кроз цео сет података за тестирање. За дужину стуба хистограма је одабрана 1/6 *MTU*, односно 33 временска корака. Иако расподела подсећа на нормалну дистрибуцију, графикон значајно тежи левој страни, односно мањим вредностима.

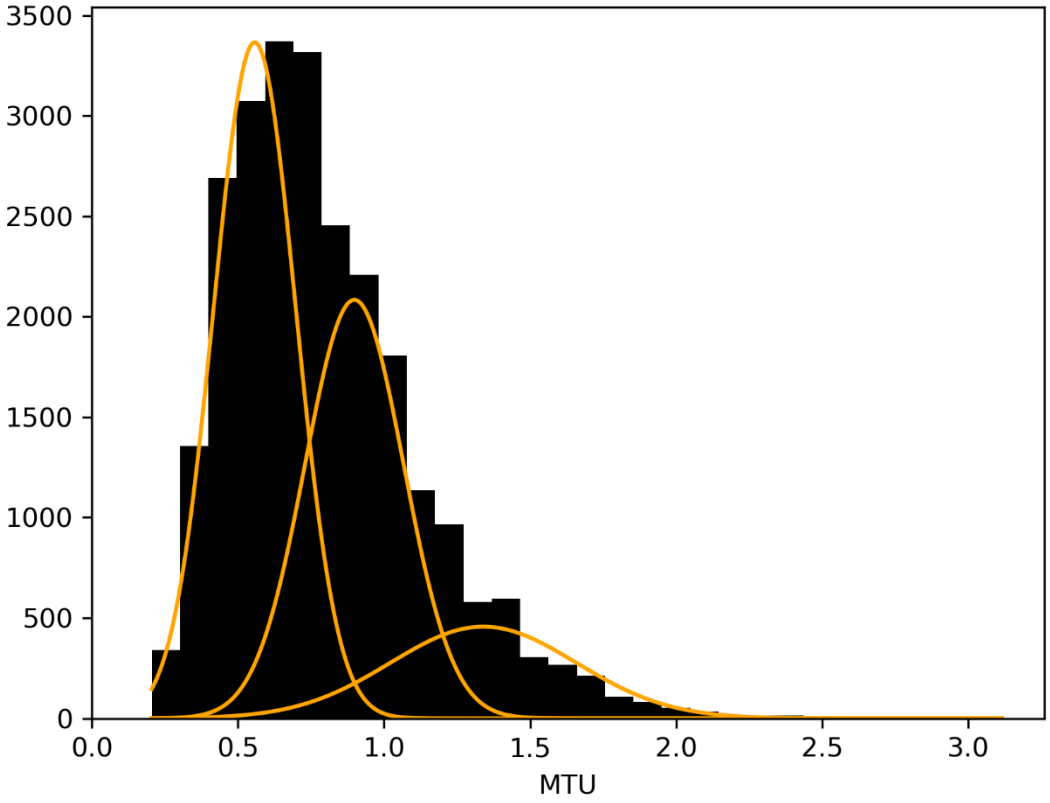
Због ове чињенице нажалост нисмо у могућности да урадимо једноставну статистичку анализу, као што је проналажење средње вредности и стандардне дивијације резултата, што би нам касније дало одличну основу за прецизно квантитативно упоређивање квалитета различитих модела за предикцију динамичких система. Сада се ипак морамо задовољити хеуристикама помоћу којих можемо да упоредимо одређене аспекте модела, углавном упоређујући различите графиконе приказане раније у раду.



Слика 19. Хистограм дужина хоризонта предикције *ESN* модела, са три криве нормалне дистрибуције на које се добијена расподела може поделити

На графикону Слика 19 приказана је иста дистрибуција на коју је примењена метода Гаусових мешавина *GMM* (*Gaussian Mixture Model*) која проналази скуп нормалних дистрибуција које у збиру што прецизније представљају дату расподелу. Одабрано је да се расподела подели на три нормалне, како би оне означавале прелазе између класа тешких, средњих и лаких почетних стања система, гледајући постигнути хоризонт предикције.

Као што се може видети са графикона, највећи број почетних стања спада у тешка, док у класу лаких спада најмањи број. Ипак, може се видети да практично нема предвиђања која спадају у два најнижа стуба на хистограму, што значи да се потпуно можемо ослонити на резултате нашег модела у привих 2/6, а са веома великом вероватноћом и у 4/6 *MTU* од почетка предвиђања. Са друге стране може се уочити да иако лаких предикција нема пуно, њихов распон је веома велики, а вредности достижу и до 3.5 *MTU*.



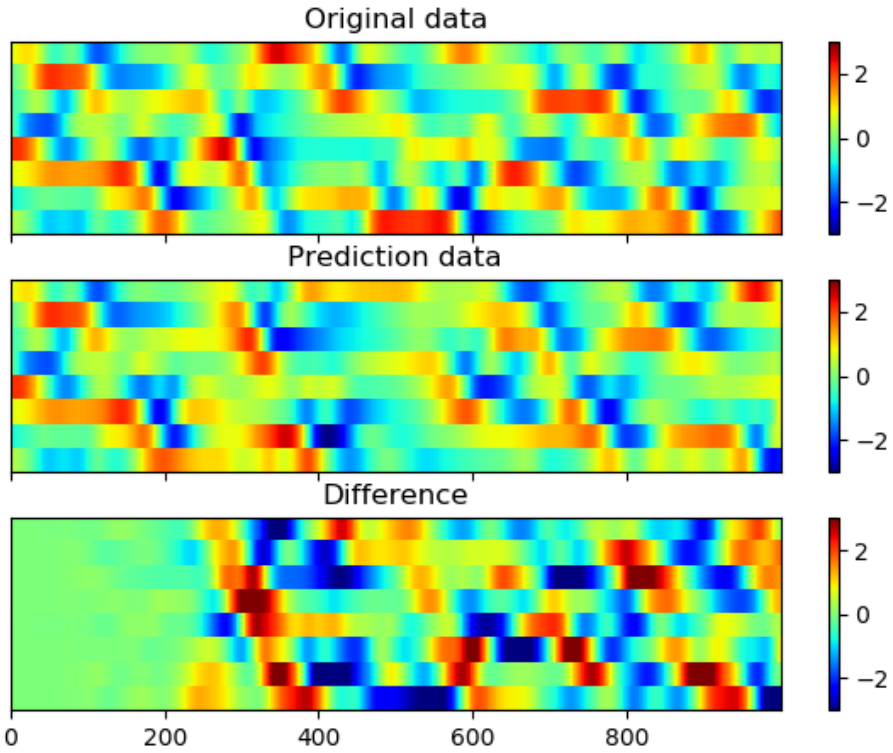
Слика 20. Хистограм дужина хоризонта предикције *LSTM* модела, са три *GMM* дистрибуције

На графикону Слика 20 може се видети хистограм добијен предвиђањима *LSTM* модела. Особине које овај хистограм показује су исте као и претходни, али треба приметити генерално лошији квалитет предвиђања, односно померај графикона на лево у односу на графиконе *ESN* модела.

4.7. Анализа резултата ансамбл модела

Као што је раније у тексту поменуто, ансамбл модели су веома широк појам који се односи на комбиновање резултата више модела на било који начин, како би коначни резултати били квалитетнији или поузданији. Како ово није био фокус рада већ успутно испитивање, експеримент је урађен једноставном методом која може бити детаљније анализирана и даље унапређена. Тренирано је неколико *ESN* модела исте архитектуре и хипер параметара, али са различитим основама за генерисање насумичних вредности које су многобројне у структури овог модела, а затим је при предикцији рачуната средња вредност излаза ових модела.

Овај приступ линеарно повећава количину компјутерских ресурса потребних за извршавање предикције са бројем модела у ансамблу, али то је заправо прихватљива последица, с обзиром да повећавање величине једног модела експоненцијално отежава његову евалуацију.

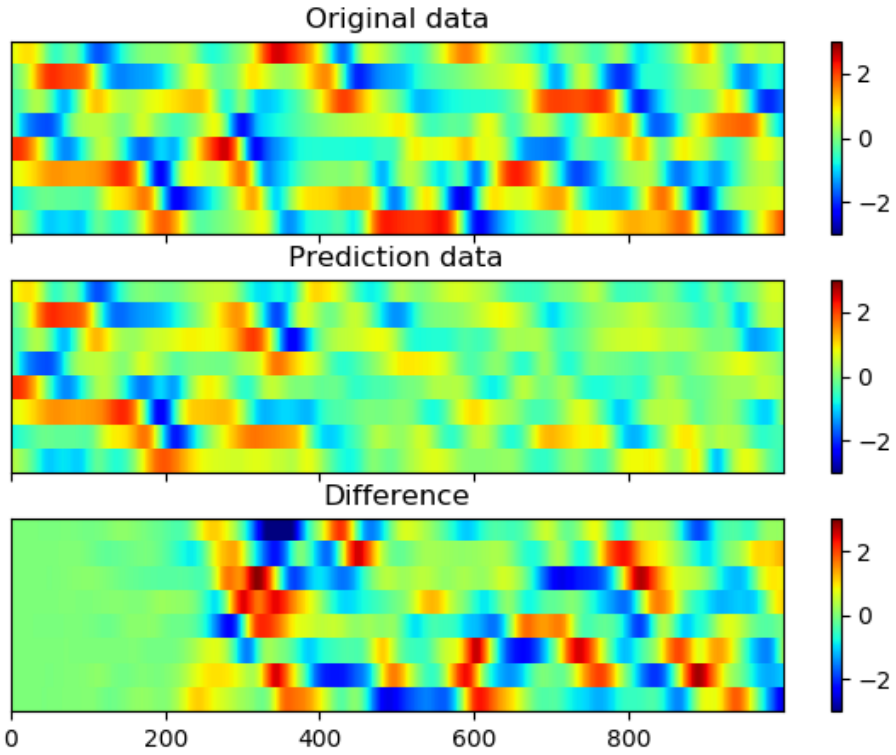


Слика 21. Графичка представа једног предвиђања једним *ESN* моделом

Шема Слика 21 има структуру са хоризонталном осом која представља корак предикције почевши од почетног стања у нули и три графикона који представљају:

1) Основне податке из сета података за тестирање
2) Података генерисаних једним ESN моделом
3) Разлику између основних и генерисаних података

На графиконима сваки ред представља вредност једне променљиве система, а промене боја њену осцилацију кроз време. Као што се може видети, у првих 200 корака предикција се поклапа са основним подацима, тако да је разлика на нули, означена зеленом бојом. Може се уочити да и након тога вредности предикције изгледају уреду, али када се одузму од основних података види се да су потпуно дивергирале. Такође, тренутак у коме предикција дивергира од основих података је веома јасан и нема лаганог повећања грешке.



Слика 22. Графичка представа једног предвиђања ансамблом *ESN* модела

Проблем са приступам ансамблима који смо одабрали у овом раду је то што чим један модел дивергира од основних података, он ће средњом вредности повући и остале, тако да коришћење ансамбла заправо не даје унапређење у самом квалитету предвиђања. Ипак, ансамбли могу дати допринос експерименту, што се може видети из шеме Слика 22.

За разлику од претходне шеме где је био коришћен само један *ESN* модел, овде се може уочити прилично равномерна зелена површина у генерисаним подацима са десне стране графикона. Ово је резултат потпуно насумичне дивергенције свих компонената ансамбла модела, што у случају нашег система чије вредности осцилују око нуле, за резултат има поништавање тих насумичних вредности и тежњу нули. Ова појава може бити корисна као ознака да се од тог тренутка више не можемо ослонити на резултате нашег модела. Ово је веома корисна информација, јер како је раније поменуто, самостални модели настављају да производе предвиђања која ни на који начин не наговештавају у којем тренутку су потпуно изгубио корак са оригиналним подацима.

5. УТИЦАЈ СМАЊЕЊА ПРЕЦИЗНОСТИ РЕАЛНЕ АРИТМЕТИКЕ

Претходна поглавља бавила су се проблемом анализе квалитета модела за предикцију динамичких система, као и проналажењем дизајна и тренирањем што квалитетнијих конкретних модела. У овом поглављу биће извршена додатна анализа утицаја смањења прецизности реалне аритметике при коришћењу модела машинског учења за предикцију динамичких система.

5.1. Потрошња ресурса за употребу модела машинског учења

Сви већи рачунарски системи захтевају значајну употребу различитих ресурса за исправно функционисање. Ти ресурси могу бити електрична енергија, време извршавања или цена и величина потребног хардвера. Дизајнер одређеног рачунарског система свесно прави компромис између улагања у различите ресурсе и квалитета резултата које систем производи.

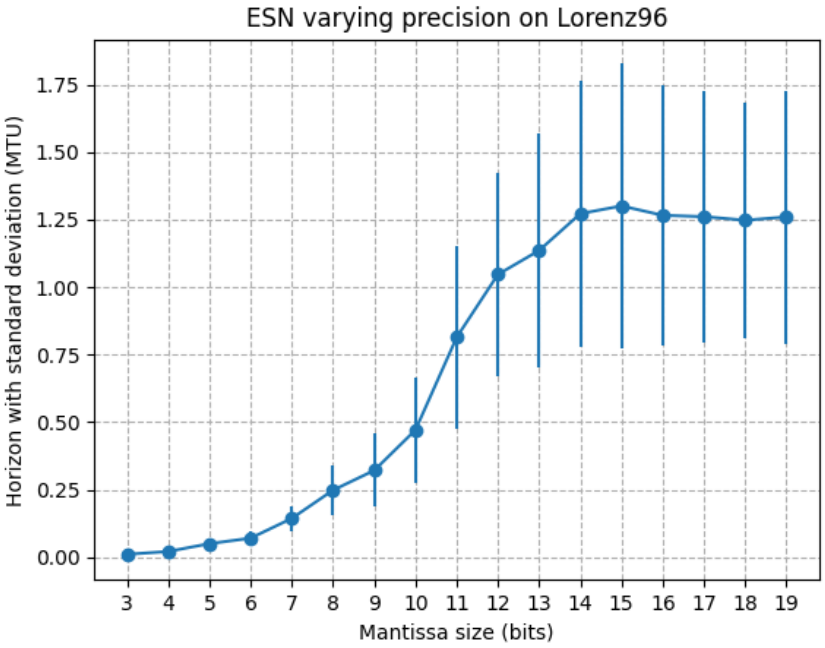
Први корак у поједностављивању система је примена метода машинског учена на одговарајуће проблеме, чија су решења другим методама неприхватљиво сложена у смислу потребних ресурса. Како и модели машинског учења постају све захтевнији, њихова употреба постаје све мање практична из угла цене потребног хардвера и времена потребног за тренирање и евалуацију. Један од водећих приступа за решавање овог проблема је приступ приближног израчунавања (*Inexact Computing* или *Approximate Computing)*. Основна идеја овог приступа је да се у замену за мали пад у квалитету модела постигне велика уштеда у утрошеним ресурсима. Ово се углавном постиже проналажењем дела модела у коме се могу користити бројне вредности мање прецизности без великог утицаја на резултате модела.

Једноставнији приступ овој техници је анализа функционисања модела и одређивање који од стандардних типова података је оптимално користити у којем кораку израчунавања. Показало се да стандардни типови података често имају превише велику разлику у прецизности, тако да коришћење мање прецизног типа података драстично квари квалитет модела. Други могући приступ који се примењује у озбиљнијим пројектима је теоријско одређивање оптималне прецизности, а затим дизајнирање и производња хардвера за ту специфичну потребу. Овако наменски произведен хардвер би смањио употребу свих ресурса у жељеној мери, уз задржавање прихватљивог квалитета израчунавања.

5.2. Смањење прецизности *ESN* модела

У нашем проблему предвиђања времена метод приближног израчунавања се често примењује са одличним резултатима [16] [17], где је показано да коришћење реалних бројева са 64, 32 и на крају 16 бита смањује потрошњу енергије и време извршавања. Како се добро показао у комбинацији са другим методама, испитан је и на нашим моделима.

Како би добили прецизније резултате од коришћења само стандардних типова података чија се прецизност драстично разликује, примењен је следећи принцип. Након тренирања, све вредности параметара модела су заокружене на вредности које могу бити представљене типовима различитих особина. У нашем примеру, варирана је мантиса реалних бројева између 3 и 23 бита, што одговара стандардним реалним бројевима укупне дужине 32 бита.



Слика 23. Просечан хоризонт предикције и стандардна девијација, у зависности од броја битова мантисе *ESN* модела

На графикону Слика 23 се могу видети просечне дужине хоризонта предикције за 100 насумичних покретања тренирања и евалуације, за различите прецизности реалних параметара модела. Као што се може видети, квалитет модела не опада смањењем броја битова мантисе са 24 бита, све до 14 бита. Уколико би за сврху евалуације овог модела био прављен посебан хардвер, ово смањење прецизности реалних бројева би значајно повећало његову ефикасност.

Упоредимо најбољи постигнути квалитет предвиђања са резултатима добијеним коришћењем стандардних реалних бројева укупне дужине 16 бита. Ови бројеви поседују мантису дужине 11 бита, што судећи по датом графикону подразумева значајније губитке у квалитету предвиђања. Идући даље низ графикон, хоризонт предикције веома брзо постаје све краћи и потпуно се губи предност у квалитету коју је *ESN* модел показао у односу на конкурентске моделе.

6.Закључак

Мотивација за овај рад била је жеља да се допринесе истраживачкој области предвиђања временских услова коришћењем метода машинског учења. Предвиђање времена се као дисциплина јако брзо развија, пратећи унапређење и повећану доступност рачунарских ресурса, као и све веће количине података добијених са временских станица на земљи, као и сателита. Ипак, симулације за предвиђање времена троше јако пуно рачунарских ресурса, што је препознато као један од највећих проблема за напредовање ове области. У овом раду су анализиране методе за унапређење овог проблема.

Евалуација свих експеримената је вршена помоћу Лоренцових хаотичних динамичких система који се у овој области сматрају за одличне моделе за тестирање решења за предвиђање временских услова. Разлог за то је наизглед хаотична природа осцилација променљивих унутар овог система, које су са једне стране потпуно одређене претходним стањем система, али такође врло тешко предвидиве. Кључна особина која везује временске услове и Лоренцове системе је њихова тенденција да уколико се почетно стање система измени само мало, у веома малом броју корака трајекторије ће се потпуно одвојити, одакле долази и концепт ефекта лептирових крила.

За моделирање система у овом раду су одабрани *LSTM* модели који генерално важе за индустријски стандард по питању предвиђања временских секвенци, и *ESN* модели који су се раније показали веома успешно у овој области. Урађена је оптимизација параметара оба модела и они су евалуирани на генерисаној трајекторији хаотичног динамичког система Лоренц 96. Оба модела су дала резултате на нивоу најбољих досадашњих објављених достигнућа, с тим да је *ESN* модел произвео значајно квалитетније предикције.

Током подешавања параметара модела и анализе направљених предикција, осмишљени су разни начини за представљање квалитета модела, специфичних за хаотичне динамичке системе, који су објашњени раније у тексту. Посебна пажње обраћена је на то да визуалне представе резултата модела буду што корисније за упоређивање квалитета различитих модела. С обзиром да различити модели испољавају веома различита понашање приликом предикције ових система, било би веома корисно осмислити конкретну метрику која би квантификовала ту разлику у квалитету. У овом раду конкретна метрика није предложена и то би био један од главних циљева у даљем раду на овој теми.

Након израде и анализе модела, урађена је анализа утицаја смањења прецизности реалне аритметике. Овај приступ анализи особина модела које се не односе само на чист квалитет резултата модела, већ и на потрошњу рачунарских ресурса, примењује се све чешће. У области дубоког машинског учења овакве анализе су све значајније јер модели који доносе импресивне резултате често захтевају значајну потрошњу скупих рачунарских ресурса. У складу са тиме, и овај рад садржи основну анализу овог типа.

Наставком унапређивања модела машинског учења, снаге рачунара и квалитета и количине података за тренирање модела, сложени проблеми погодни за решавање на овај начин добијају решења изврсног квалитета. У овом раду је дат допринос проблему предвиђања временских услова, првенствено агрегацијом најуспешнијих приступа решавању овог проблема, као и покушајем да се што интуитивније опишу особине система који се моделирају, као и коришћених модела.

Литература

[1] E. D. Craft, „Economic History of Weather Forecasting.,“ *EH. Net Encyclopedia,* 2001.

[2] T. Gneiting и A. E. Raftery, „Weather Forecasting with Ensemble Methods,“ *Science,* 2005.

[3] C. Guanrong и D. Xaiaoning, „From Chaos to Order - Perspectives and Methodologies in Controlling Chaotic Nonlinear Dynamical Systems,“ *International Journal of Bifurcation and Chaos,* 1993.

[4] P. D. Dueben и P. Bauer, „Challenges and design choices for global weather and climate models based on machine learning,“ *Geoscientific Model Development,* т. 11, 20018.

[5] A. Katok и B. Hasselblatt, „Introduction to the modern theory of dynamical systems,“ *Cambridge university press,* 1997.

[6] E. N. Lorenz, „Deterministic Nonperiodic Flow,“ *Journal of the Atmospheric Sciences,* p. 130–141, 1963.

[7] R. Abraham и Y. Ueda, The Chaos Avant-garde: Memories of the Early Days of Chaos Theory, 2000.

[8] R. Shaw, „Modeling Chaotic Systems.,“ *Chaos and Order in Nature. Springer Series in Synergetics,* т. 11, 1981.

[9] M. I. Jordan и T. M. Mitchell, „Machine learning: Trends, perspectives, and prospects,“ *Science,* т. 349, бр. 6245, pp. 255-260, 2015.

[10] F. A. Gers, J. Schmidhuber и F. Cummins, „Learning to forget: continual prediction with LSTM,“ у *9th International Conference on Artificial Neural Networks: ICANN '99,* Edinburgh, UK , 1999.

[11] L. Xiaowei, Y. Zehong и S. Yixu, „Short-term stock price prediction based on echo state networks,“ *Expert Systems with Applications,* т. 36, бр. 3, pp. 7313-7317, 2009.

[12] A. Sherstinsky, „Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network,“ *Physica D: Nonlinear Phenomena,* т. 404, 2020.

[13] C. Runge, „Ueber die numerische Auflösung von Differentialgleichungen,“ *Mathematische Annalen,* т. 46, p. pages 167–178, 1895.

[14] J. Pathak, B. Hunt, M. Girvan, Z. Lu и E. Ott, „Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach,“ *Physical Review Letters,* т. 120, 2018.

[15] T. Thornes, P. Düben и T. Palmer, „On the use of scale-dependent precision in Earth System modelling,“ *Quarterly Journal of the Royal Meteorological Society,* т. 143, бр. 703, pp. 897-908, 2017.

[16] P. D. Duben, J. Joven, A. Lingamneni, H. McNamara, G. De Micheli, K. V. Palem и T. N. Palmer, „On the use of inexact, pruned hardware in atmospheric modelling,“ *Philosophical transactions of the royal society A,* т. 372, бр. 2018, 2014.

[17] R. Pyle, N. Jovanovic, D. Subramanian, K. V. Palem и A. B. Patel, „Domain-driven models yield better predictions at lower cost than reservoir computers in Lorenz systems,“ *Philosophical transactions of the royal society A,* т. 379, бр. 2194, 2021.

Списак скраћеница

CDS	*Chaotical Dynamical Systems*
EM	*Ensemble Models*
ESN	*Echo State Networks*
FG	*Forget Gate*
GMM	*Gaussian Mixture Model*
IC/AC	*Inexact Computing /Approximate Computing*
IG	*Input Gate*
JSON	JavaScript Object Notation
LSTM	*Long-Short Term Memory*
MTU	*Model Time Unit*
OG	*Output Gate*
PH	*Prediction Horizon*
RC	*Reservoir Computing*
RNN	*Recursive Neural Networks*

Списак слика

A. Програмски код

У овом прилогу биће дат програмски код коришћених модела, ако и скрипти за приказивање предикција модела за одабране почетне услове и генерисање већине графикона датих у овом раду.

Параметри модела су чувани у посебним *JSON* фајловима, како би били одвојени од саме имплементације и могли да буду лако измењени и учитани приликом покретања модела.

A.1. Програмски код *LSTM* модела

```
from script.model import Model

from script.printer import Printer

import json
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.layers import LSTM
from keras.layers import CuDNNLSTM
from keras.models import load_model
import numpy as np
import os
import tensorflow as tf


# Model class representing LSTM with variable look-back

class ModelLstm(Model):

    # Initialize model parameters and variables

    def __init__(self, model_params, data_set_path):
        super().__init__(data_set_path)

        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        tf.Session(config=config)

        # Parameters
        self.n_inp = model_params['n_inp']
        self.tr_skip = model_params["tr_skip"]
        self.look_back = model_params['look_back']
        self.tr_len = model_params['tr_len']
        self.n_hidden = model_params['n_hidden']
        self.do_basis_exp = model_params['do_basis_exp']
        self.epochs = model_params['epochs']
        self.batch_size = model_params['batch_size']
        self.shuffle = model_params['shuffle']

        # State
        self.model = None
        self.predictions = {}

    def params_to_dict(self):
        return {
            'n_inp': self.n_inp,
            'tr_skip': self.tr_skip,
            'look_back': self.look_back,
            'tr_len': self.tr_len,
            'n_hidden': self.n_hidden,
            'do_basis_exp': self.do_basis_exp,
            'epochs': self.epochs,
            'batch_size': self.batch_size,
            'shuffle': self.shuffle,
        }
```

```
# Convert simple to stacked data for given LSTM look back

def stack_data(self, data):
    whole_data_len = data.shape[1]
    data_points = whole_data_len - self.look_back + 1

    x_temp = np.zeros((self.look_back, self.n_inp, data_points))
    for i in range(self.look_back):
        x_temp[i, :] = data[:, i:data_points + i]

    x = x_temp[0]
    for i in range(self.look_back - 1):
        x = np.vstack([x, x_temp[i + 1]])

    x = np.transpose(x)
    return x.reshape((data_points, self.look_back, self.n_inp))

# Generate model and train Wout

def train(self, data, verbose):

    Printer.add(verbose, 'training')

    self.model = Sequential()
    # self.model.add(LSTM(self.n_hidden, input_shape=(self.look_back, self.n_inp)))
    self.model.add(CuDNNLSTM(self.n_hidden, input_shape=(self.look_back, self.n_inp)))

    if self.do_basis_exp:
        delinearize_layer = Dense(self.n_hidden)
        built_layer = delinearize_layer(Input((self.n_hidden,)))
        built_layer.trainable = False
        weights = delinearize_layer.get_weights()
        weights[0] = np.zeros((self.n_hidden, self.n_hidden))
        for i in range(0, self.n_hidden, 2):
            weights[0][i, i] = 1
        for i in range(1, self.n_hidden, 2):
            weights[0][i, i] = 1
            weights[0][i, i - 1] = 1
        delinearize_layer.set_weights(weights)
        self.model.add(delinearize_layer)

    self.model.add(Dense(self.n_inp))
    self.model.compile(loss='mse', optimizer='adam')

    # Prepare data

    # Call stack data and pass the whole training set except for the last data point
    # The last data point is needed as a label for the last stacked training set entry?
    x = self.stack_data(data[:, :self.tr_len - 1])
    y = np.transpose(data[:, self.look_back:self.tr_len])

    # Fit network

    self.model.fit(x, y, epochs=self.epochs, batch_size=self.batch_size,
                   verbose=2, shuffle=self.shuffle)
    self.model._make_predict_function()

    Printer.rem(verbose)

# Load reservoir if it exists, else train it

def load_or_train(self, data, verbose):

    folder_name = self.dict_to_os_path_with_data(self.params_to_dict())
    try:

        Printer.print_log(verbose, 'Loading LSTM model... ')
        self.model = load_model(os.path.join(folder_name, 'lstm_model.h5'))
        self.model._make_predict_function()
        Printer.print_log(verbose, 'Loading LSTM model... Done')

    except OSError:
```

```
            self.train(data[:, self.tr_skip:self.tr_skip + self.tr_len], verbose)

            Printer.print_log(verbose, 'Saving LSTM model... ')
            if not os.path.exists(folder_name):
                os.mkdir(folder_name)
            with open(os.path.join(folder_name, 'params.json'), 'w') as out_file:
                json.dump(self.params_to_dict(), out_file)
            self.model.save(os.path.join(folder_name, 'lstm_model.h5'))
            Printer.print_log(verbose, 'Saving LSTM model... Done')

    # Predict function

    def predict(self, warm_up_input, run_params, pos_inside_run, verbose):

        if str(run_params) in self.predictions:
            return self.predictions[str(run_params)][pos_inside_run]
        else:
            try:
                file_prefix = os.path.join(self.data_set_path, Model.model_run_params_prefix(
                    self.params_to_dict(), run_params))
                self.predictions = np.load(file_prefix + 'pred.npy')
                return self.predictions[pos_inside_run]
            except OSError:
                pass

        Printer.add(verbose, 'predicting')

        output = np.zeros((run_params['pred_len'], self.n_inp))

        stacked_input = self.stack_data(warm_up_input)

        for i in range(run_params['pred_len']):
            pred = self.model.predict(stacked_input)
            output[i, :] = pred
            stacked_input[0, :-1, :] = stacked_input[0, 1:, :]
            stacked_input[0, -1, :] = pred

        Printer.rem(verbose)
        return output.T
```

A.2. Програмски код *ESN* модела

```
from script.model import Model

from script.printer import Printer

import json
import numpy as np
import os
import scipy.sparse as sparse


# Model class representing ESN

class ModelEsn(Model):

    # Initialize model parameters and variables

    def __init__(self, model_params, data_set_path):
        super().__init__(data_set_path)

        # Parameters
        self.n_inp = model_params["n_inp"]
        self.tr_skip = model_params["tr_skip"]
        self.tr_len = model_params["tr_len"]
        self.radius = model_params["radius"]
        self.degree = model_params["degree"]
        self.sigma = model_params["sigma"]
        self.beta = model_params["beta"]
        self.win_seed = model_params["win_seed"]
        self.a_seed = model_params["a_seed"]
        self.fun = model_params["fun"]
        self.size = model_params["size"]
```

```
        # State
        self.w_in = None
        self.a = None
        self.a_sparse = None
        self.w_out = None
        self.predictions = {}

        # Compute reservoir state update non-linearity
        if self.fun == 'tanh':
            self.r_s_u_n_l = np.tanh
        elif self.fun == 'norm':
            self.r_s_u_n_l = lambda vector: vector / np.linalg.norm(vector)
        else:
            print()
            print('Unknown reservoir state update non-linearity function')
            exit(1)

    def params_to_dict(self):

        return {
            'n_inp': self.n_inp,
            'tr_skip': self.tr_skip,
            'tr_len': self.tr_len,
            'radius': self.radius,
            'degree': self.degree,
            'sigma': self.sigma,
            'beta': self.beta,
            'win_seed': self.win_seed,
            'a_seed': self.a_seed,
            'fun': self.fun,
            'size': self.size,
        }

    # Generate reservoir sparse matrix

    def generate_reservoir(self, verbose, a_sparse=None):

        Printer.print_log(verbose, 'generating reservoir')
        if a_sparse is None:
            sparsity = self.degree / float(self.size)
            np.random.seed(self.a_seed)
            a_sparse = sparse.rand(self.size, self.size, density=sparsity)
        a = a_sparse.todense()
        values = np.linalg.eigvals(a)
        e = np.max(np.abs(values))
        a = (a / e) * self.radius
        return a, a_sparse

    # Generate the vector of reservoir states for each training data point and return as a matrix

    def reservoir_warm_up_history(self, input_data, verbose):

        Printer.add(verbose, 'generating reservoir state history')
        Printer.print_log(verbose)
        states = np.zeros((self.size, self.tr_len))
        for i in range(self.tr_len - 1):
            if i % int((self.tr_len - 1) / 100) == 0 or i == self.tr_len - 2:
                Printer.print_log(verbose, '{}%'.format(int(i * 100 / (self.tr_len - 1))))
            states[:, i + 1] = self.r_s_u_n_l(
                np.dot(self.a, states[:, i]) + np.dot(self.w_in, input_data[:, i]))
        Printer.rem(verbose)
        return states

    # Delinearize matrix of inputs by multiplying every even element by its predecessor
    # (introduce basis expansion)
    @staticmethod
    def basis_expansion_matrix(in_arr, verbose):

        Printer.print_log(verbose, 'introducing basis expansion')
        out_arr = in_arr.copy()
        for j in range(1, np.shape(in_arr)[0], 2):
            out_arr[j, :] = in_arr[j, :] * in_arr[j - 1, :]
        return out_arr
```

```
# Generate model and train Wout

def train(self, data, verbose):

    Printer.add(verbose, 'training')

    # Win
    Printer.print_log(verbose, 'generating Win')
    q = int(self.size / self.n_inp)
    self.w_in = np.zeros((self.size, self.n_inp))
    np.random.seed(self.win_seed)
    for i in range(self.n_inp):
        self.w_in[i * q: (i + 1) * q, i] = self.sigma * (-1 + 2 * np.random.rand(q))

    # A
    self.a, self.a_sparse = self.generate_reservoir(verbose)

    # Wout
    Printer.print_log(verbose, 'generating Wout')
    states = self.reservoir_warm_up_history(data, verbose)
    id_mat = self.beta * sparse.identity(self.size)
    expanded_states = self.basis_expansion_matrix(states, verbose)
    u = np.dot(expanded_states, expanded_states.transpose()) + id_mat
    self.w_out = np.dot(np.linalg.inv(u), np.dot(expanded_states, data.transpose())).transpose()

    Printer.rem(verbose)

# Load reservoir if it exists, else train it

def load_or_train(self, data, verbose):

    folder_name = self.dict_to_os_path_with_data(self.params_to_dict())
    try:

        Printer.print_log(verbose, 'Loading reservoir... ')
        self.w_in = np.load(os.path.join(folder_name, 'win.npy'))
        self.w_out = np.load(os.path.join(folder_name, 'wout.npy'))
        self.a_sparse = sparse.load_npz(os.path.join(folder_name, 'a_sparse.npz'))
        self.a, _ = self.generate_reservoir(verbose, self.a_sparse)
        Printer.print_log(verbose, 'Loading reservoir... Done')

    except FileNotFoundError:

        self.train(data[:, self.tr_skip:self.tr_skip + self.tr_len], verbose)

        Printer.print_log(verbose, 'Saving reservoir... ')
        if not os.path.exists(folder_name):
            os.mkdir(folder_name)
        with open(os.path.join(folder_name, 'params.json'), 'w') as out_file:
            json.dump(self.params_to_dict(), out_file)
        np.save(os.path.join(folder_name, 'win.npy'), self.w_in)
        np.save(os.path.join(folder_name, 'wout.npy'), self.w_out)
        sparse.save_npz(os.path.join(folder_name, 'a_sparse.npz'), self.a_sparse)
        Printer.print_log(verbose, 'Saving reservoir... Done')

# Reservoir warm up function

def reservoir_warm_up(self, warm_up_input, run_params):

    states = np.zeros((self.size, run_params['warm_up_size'] + 1))
    for i in range(run_params['warm_up_size']):
        states[:, i + 1] = self.r_s_u_n_l(
            np.dot(self.a, states[:, i]) + np.dot(self.w_in, warm_up_input[:, i]))
    return states[:, -1]

# Delinearize reservoir state
@staticmethod
def basis_expansion_array(in_arr):

    arr_shifted = np.insert(in_arr.copy()[:-1], 0, 1)
    arr_multiply = np.multiply(in_arr, arr_shifted)
    return np.dstack((in_arr[::2], arr_multiply[1:][::2])).flatten()
```

```
    # Predict function

    def predict(self, warm_up_input, run_params, pos_inside_run, verbose):

        if str(run_params) in self.predictions:
            return self.predictions[str(run_params)][pos_inside_run]
        else:
            try:
                file_prefix = os.path.join(self.data_set_path, Model.model_run_params_prefix(
                    self.params_to_dict(), run_params))
                self.predictions = np.load(file_prefix + 'pred.npy')
                return self.predictions[pos_inside_run]
            except OSError:
                pass

        Printer.add(verbose, 'predicting')
        res_state = self.reservoir_warm_up(warm_up_input, run_params)
        output = np.zeros((self.n_inp, run_params['pred_len']))
        for i in range(run_params['pred_len']):
            if (i * 100) % run_params['pred_len'] == 0:
                Printer.print_log(verbose, '{}/{}'.format(i + 1, run_params['pred_len']))
            x_aug = ModelEsn.basis_expansion_array(res_state)
            output[:, i] = np.squeeze(np.asarray(np.dot(self.w_out, x_aug)))
            x1 = self.r_s_u_n_l(np.dot(self.a, res_state) + np.dot(self.w_in, output[:, i]))
            res_state = np.squeeze(np.asarray(x1))
        Printer.rem(verbose)
        return output
```

A.3. Програмски код скрипте за анализу направљених предикција

```
%matplotlib nbagg

from script.model import Model, DataSet
from script.model_esn import ModelEsn
# from script.model_lstm import ModelLstm

import ipywidgets as widgets
from IPython.display import display
import matplotlib.pyplot as plt
import numpy as np
import threading

# Reservoir warmup function

def reservoir_warmup(A, Win, warmup_input, model_params, warmup_length):

    states = np.zeros((model_params['size'], warmup_length + 1))
    for i in range(warmup_length):
        a = np.dot(A, states[:, i])
        b = np.dot(Win, warmup_input[:, i])
        states[:, i + 1] = np.tanh(a + b)
    return states[:,-1]


# Delinearize reservoir state

def delinearize_array(in_arr):

    arr_shifted = np.insert(in_arr.copy()[:-1], 0, 1)
    arr_mult = np.multiply(in_arr, arr_shifted)
    return np.dstack((in_arr[::2], arr_mult[1:][::2])).flatten()


# Predict function

def predict_with_warmup(A, Win, model_params, warmup_input, Wout,
                        pred_length, thread_state, warmup_length):

    res_state = reservoir_warmup(A, Win, warmup_input, model_params, warmup_length)
    output = np.zeros((model_params['n_inp'], pred_length))
    for i in range(pred_length):
        if thread_state['running'] == False:
            return None
```

```
        x_aug = delinearize_array(res_state)
        output[:, i] = np.squeeze(np.asarray(np.dot(Wout, x_aug)))
        x1 = np.tanh(np.dot(A, res_state) + np.dot(Win, output[:, i]))
        res_state = np.squeeze(np.asarray(x1))
    return output

# Plot reference data and predicted output

def plot_lines(reference_data_start, reference_data_end, reference_data,
               vertical_line_time, prediction_start_time, prediction_end_time,
               prediction, horizon, model_params, red_col):

    if not prediction is None and \
        ((prediction_start_time > reference_data_start and \
        prediction_start_time < reference_data_end) or
        (prediction_end_time > reference_data_start and \
        prediction_end_time < reference_data_end)):

        time_skipped = prediction_start_time - model_params['tr_len']
        prediction_length = prediction_end_time - prediction_start_time

        if prediction_start_time < reference_data_start:
            left_cut = reference_data_start - prediction_start_time
            prediction_start_time += left_cut
            prediction = prediction[:, left_cut:]

        if prediction_end_time > reference_data_end:
            right_cut = prediction_end_time - reference_data_end
            prediction_end_time -= right_cut
            prediction = prediction[:, :-right_cut]

        plt.suptitle('ESN free prediction\nReservoir size: {}\nTraining length: {}\n'
                     'Time skipped: {}\nPrediction length: {}\nWarmup length: {}\n'
                     'Prediction horizon: {}'.format(
                         model_params['size'], model_params['tr_len'], time_skipped,
                         prediction_length, temp_params['old_warmup_size'], horizon / 200))

    else:
        plt.suptitle('ESN free prediction')

    for var in range(model_params['n_inp']):
        ax[var].clear()
        ax[var].plot(range(reference_data_start, reference_data_end), reference_data[var], 'black')
        if not prediction is None and \
            ((prediction_start_time > reference_data_start and \
            prediction_start_time < reference_data_end) or
            (prediction_end_time > reference_data_start and \
            prediction_end_time < reference_data_end)):
            ax[var].plot(range(prediction_start_time, prediction_end_time), prediction[var],
                    red_col, label='Variable = {}'.format(var + 1))
            ax[var].legend(loc=1)
            ax[var].axvline(x=vertical_line_time, color='black', linestyle='--')
            if horizon is not None:
                ax[var].axvline(x=prediction_start_time + horizon, color='black', linestyle='--')

        ax[var].set_ylim([-3, 3])
    plt.draw()


# Thread function for parallel computation of prediction output

def prediction_thread(A, Win, model_params, warmup_input, Wout, pred_length, thread_state,
                      reference_data_start, reference_data_end, reference_data,
                      prediction_start_time, prediction_end_time, warmup_length, graph_padding):

    prediction = predict_with_warmup(A, Win, model_params, warmup_input, Wout,
                                     pred_length, thread_state, warmup_length)
    if thread_state['running'] == True:
        temp_params['old_prediction'] = prediction
        temp_params['old_prediction_start'] = prediction_start_time
        temp_params['old_prediction_end'] = prediction_end_time

        comp_data = reference_data[:, graph_padding:graph_padding + pred_length]
        e = np.linalg.norm(prediction - comp_data, axis=0) / np.linalg.norm(comp_data, axis = 0)
```

```
        temp_params['old_horizon'] = np.argmax(e > 0.3) if (e > 0.3).any() else pred_length

        temp_params['old_warmup_size'] = warmup_length
        plot_lines(reference_data_start, reference_data_end, reference_data, prediction_start_time,
                   prediction_start_time, prediction_end_time, temp_params['old_prediction'],
                   temp_params['old_horizon'], model_params, 'red')

        temp_params['plot_thread'] = None

# Prepare the params and the model

temp_params = {}
temp_params['plot_thread'] = None
temp_params['plot_thread_state'] = None

temp_params['old_prediction'] = None
temp_params['old_prediction_start'] = None
temp_params['old_prediction_end'] = None
temp_params['old_horizon'] = None
temp_params['old_warmup_size'] = None

# Loading Lorenz data

data_set = DataSet('lorenz_96', 200, 1)
data = data_set.data

# Load default initialization parameters

model_params = Model.default_model_params('esn')
# model_params = Model.default_model_params('lstm')
run_params = Model.default_run_params()

# Load or train the model

model = ModelEsn(model_params, data_set.path_prefix())
# model = ModelLstm(model_params, data_set.path_prefix())
model.load_or_train(data, True)
predictions = np.zeros((run_params['num_i_c'], model_params['n_inp'], run_params['pred_len']))

# Predict

fig, ax = plt.subplots(model_params['n_inp'], figsize=(9, 1.5 * model_params['n_inp']), sharex=True)
plt.subplots_adjust(top=0.86)
plt.suptitle('ESN free prediction')
for var in range(model_params['n_inp']):
    ax[var].set_xlabel('$timestep$')
    ax[var].set_ylabel('$output$')


def update_plot(time_offset_1000, time_offset_10, pred_length, warmup_length):
    global model_params

    predict_time_offset = time_offset_1000 + time_offset_10
    prediction_start_time = predict_time_offset #  + model_params['tr_len']
    prediction_end_time = prediction_start_time + pred_length
    graph_padding = int(pred_length * 0.5)

    reference_data_start = prediction_start_time - graph_padding
    reference_data_end = prediction_end_time + graph_padding
    prediction_data_warmup_start = prediction_start_time - warmup_length

    reference_data = data[:, reference_data_start:reference_data_end]

    warmup_input = data[:, prediction_data_warmup_start:prediction_start_time]

    plot_lines(reference_data_start, reference_data_end, reference_data,
               prediction_start_time, temp_params['old_prediction_start'],
               temp_params['old_prediction_end'], temp_params['old_prediction'],
               temp_params['old_horizon'], model_params, 'lightcoral')

    # Prediction thread
    if temp_params['plot_thread'] != None:
        temp_params['plot_thread_state']['running'] = False
```

```
    temp_params['plot_thread_state'] = { 'running': True }
    temp_params['plot_thread'] = threading.Thread(
        target=prediction_thread,
        args=(model.a, model.w_in, model_params, warmup_input,
              model.w_out, pred_length, temp_params['plot_thread_state'],
              reference_data_start, reference_data_end, reference_data,
              prediction_start_time, prediction_end_time, warmup_length, graph_padding)
    )
    temp_params['plot_thread'].start()


time_offset_1000 = widgets.IntSlider(
    min=3000,
    max=data.shape[1] - 3000,
    step=1000, value=950000,
    description='Global skip:',
    continuous_update=True)

time_offset_10 = widgets.IntSlider(
    min=0, max=1000, step=10, value=0,
    description='Local skip:',
    continuous_update=True)

pred_length = widgets.IntSlider(
    min=100, max=1000, value=500,
    description='Pred length:',
    continuous_update=True)

warmup_length = widgets.IntSlider(
    min=1, max=10, value=3,
    description='Warmup:',
    continuous_update=False)

widgets.interactive(
    update_plot,
    time_offset_1000=time_offset_1000,
    time_offset_10=time_offset_10,
    pred_length=pred_length,
    warmup_length=warmup_length)
```

A.4. Програмски код скрипте за цртање разних графикона

```
# Analise the behaviour of prediction horizon on a consecutive set of initial conditions

from script.model import Model, DataSet
from script.model_ensemble_1 import ModelEnsemble1
from script.model_esn import ModelEsn
from script.model_lstm import ModelLstm
from script.model_lstm_2 import ModelLstm2
from script.printer import Printer

import json
import math
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
import threading
import time

from scipy.stats import norm
from sklearn.mixture import GaussianMixture


# Calculate prediction horizon

def get_horizon(e, threshold):
    return np.argmax(e > threshold) if (e > threshold).any() else len(e) - 1


# Min thread function

def thread_main(model_params, data, run_params, horizons, ind,
```

```
                threads, model, x_true_avg, predictions, errors):

    tasks = math.ceil(run_params['num_i_c'] / threads)
    if ind + 1 == threads:
        tasks = run_params['num_i_c'] - (threads - 1) * tasks

    ics = range(run_params['start_i_c'] + ind * tasks * run_params['i_c_distance'],
                run_params['start_i_c'] + (ind + 1) * tasks * run_params['i_c_distance'],
                run_params['i_c_distance'])

    verbose = ind == 0
    Printer.add(verbose, 'Thread: {}/{}'.format(ind + 1, threads))
    Printer.add(verbose, 'Predicting')
    Printer.print_log(verbose)

    start_time = time.time()
    threshold = model_params['n_inp'] * 0.0375
    for count, init_cond in enumerate(ics):
        Printer.add(verbose, 'IC: {}/{} - {}s'.format(
            count + 1, tasks, int(time.time() - start_time)))

        if model is None:
            prediction = predictions[ind * tasks + count]
        else:
            warm_up_input = data[:, init_cond - run_params['warm_up_size']:init_cond]
            prediction = model.predict(warm_up_input, run_params, ind * tasks + count, verbose)
            predictions[ind * tasks + count, :] = prediction
        Printer.print_log(verbose, 'calculating error...')
        comp_data = data[:, init_cond:init_cond + run_params['pred_len']]
        e = np.linalg.norm(prediction - comp_data, axis=0) / x_true_avg
        errors[ind * tasks + count, :] = e
        horizons[0, ind * tasks + count] = get_horizon(e, threshold)
        for inp in range(model_params['n_inp']):
            horizons[inp + 1, ind * tasks + count] =
                get_horizon(np.abs(prediction[inp] - comp_data[inp]), threshold)
        Printer.print_log(verbose, 'calculating error... Done')

        Printer.rem(verbose)


# Plotting prediction horizons

def plot_horizon(run_params, data, model_params, horizons, graph_folder, mtu_constant):
    Printer.print_log(True, 'Plotting prediction horizon graph... ')

    width = max(run_params['num_i_c'] / 300, 10)
    fig, ax = plt.subplots(model_params['n_inp'] + 1,
        figsize=(width, 1.4 * (model_params['n_inp'] + 1)), sharex=True)
    plt.suptitle('Prediction horizon')
    x_values_var = np.arange(run_params['num_i_c'] + run_params['pred_len']) / mtu_constant
    x_values = np.arange(run_params['num_i_c']) / mtu_constant

    for var in range(model_params['n_inp']):
        data_from = run_params['start_i_c']
        data_to = run_params['start_i_c'] +
            run_params['num_i_c'] * run_params['i_c_distance'] + run_params['pred_len']
        data_skip = run_params['i_c_distance']
        ax[var].plot(
            x_values_var,
            data[var, data_from:data_to:data_skip],
            'black', label='Variable: {}/{}'.format(var + 1, model_params['n_inp']))
        ax[var].set_xlim([0, x_values_var[-1]])
        ax[var].legend(loc=1)

    hor_id = model_params['n_inp']
    ax[hor_id].plot(x_values, horizons / mtu_constant, 'black', label='Horizons')
    ax[hor_id].set_xlabel('initial condition (shift from start in MTU)')
    ax[hor_id].set_ylabel('MTU')
    ax[hor_id].set_xlim([0, x_values_var[-1]])
    ax[hor_id].legend(loc=1)

    plt.savefig(os.path.join(graph_folder, 'pred_horizons.png'), dpi=300)
```

```
# Plotting colored map of error for each time-step of each initial condition

def plot_all_errors(run_params, errors, graph_folder, mtu_constant):
    Printer.print_log(True, 'Plotting colored map of all errors... ')

    errors = errors.T
    errors = np.array([errors[i, :] for i in range(errors.shape[0] - 1, 0, -1)])

    width = max(run_params['num_i_c'] / 300, 10)
    fig, ax = plt.subplots(1, figsize=(width, 10))
    plt.suptitle('Colored map of errors for each time-step and initial condition')

    ax.matshow(errors, aspect='auto', cmap=plt.cm.get_cmap('Blues'))
    ax.set_xlabel('initial condition (shift from start in MTU)')
    ax.set_ylabel('prediction time-step')
    # Flip y axis
    # Width more than one pixel

    ax.set_xticklabels([''] + list(np.arange(0, run_params['num_i_c'] + 0.1,
        run_params['num_i_c'] / 5) / mtu_constant))
    ax.set_yticklabels([''] + list(
        range(run_params['pred_len'], -1, -int(run_params['pred_len'] / 5))))

    plt.savefig(os.path.join(graph_folder, 'colored_errors.png'), dpi=300)


# Plotting prediction horizons per each variable

def plot_horizon_per_var(run_params, data, model_params, horizons, graph_folder, mtu_constant):
    Printer.print_log(True, 'Plotting prediction horizon graphs per variable... ')

    width = max(run_params['num_i_c'] / 300, 10)
    fig, ax = plt.subplots(model_params['n_inp'] * 2,
        figsize=(width, 2 * 1.4 * model_params['n_inp']), sharex=True)
    plt.suptitle('Prediction horizons per variable')
    x_values_var = np.arange(run_params['num_i_c'] + run_params['pred_len']) / mtu_constant
    x_values = np.arange(run_params['num_i_c']) / mtu_constant

    for var in range(model_params['n_inp']):
        data_row = var * 2
        hor_row = data_row + 1

        data_from = run_params['start_i_c']
        data_to = run_params['start_i_c'] +
            run_params['num_i_c'] * run_params['i_c_distance'] + run_params['pred_len']
        data_skip = run_params['i_c_distance']

        ax[data_row].plot(
            x_values_var,
            data[var, data_from:data_to:data_skip],
            'black', label='Variable: {}/{}'.format(var + 1, model_params['n_inp']))
        ax[data_row].legend(loc=1)
        ax[data_row].set_xlim([0, x_values_var[-1]])

        ax[hor_row].plot(x_values, horizons[var] / mtu_constant,
            'black', label='Variable {} horizons'.format(var + 1))
        ax[hor_row].set_xlim([0, x_values_var[-1]])
        ax[hor_row].set_xlabel('initial condition (shift from start in MTU)')
        ax[hor_row].set_ylabel('MTU')
        ax[hor_row].legend(loc=1)

    plt.savefig(os.path.join(graph_folder, 'pred_horizons_per_var.png'), dpi=300)


# Plot histogram of prediction horizons

def plot_histogram(horizons, graph_folder, mtu_constant):
    Printer.print_log(True, 'Plotting prediction horizon histogram... ')

    num_bins = 30

    plt.figure()
    plt.suptitle('Prediction horizon histogram')
    plt.hist(horizons / mtu_constant, bins=num_bins, color='black')
```

```
plt.xlim(xmin=0.0)
plt.xlabel('MTU')
plt.savefig(os.path.join(graph_folder, 'pred_horizons_histogram.png'), dpi=300)

curve_points = num_bins * 50

min_hor = np.min(horizons)
max_hor = np.max(horizons)
data = horizons.reshape(-1, 1)
gmm_model = GaussianMixture(n_components=3).fit(data)
total_curve = np.exp(gmm_model.score_samples(
    np.linspace(min_hor, max_hor, curve_points).reshape(-1, 1)))
total_curve /= sum(total_curve)
total_curve *= horizons.shape[0] / num_bins * curve_points

plt.figure()
plt.suptitle('Prediction horizon histogram with combined gmm')
plt.hist(horizons / mtu_constant, bins=num_bins, color='black')
plt.xlim(xmin=0.0)
plt.plot(np.linspace(min_hor, max_hor, curve_points) / mtu_constant,
    total_curve, color='orange')
plt.xlabel('MTU')
plt.savefig(os.path.join(graph_folder, 'pred_horizons_histogram_combined_gmm.png'), dpi=300)

plt.figure()
plt.suptitle('Prediction horizon histogram with separate gmm')
plt.hist(horizons / mtu_constant, bins=num_bins, color='black')

for i in range(3):
    mu = gmm_model.means_[i]
    sigma = math.sqrt(gmm_model.covariances_[i])
    x = np.linspace(min_hor, max_hor, curve_points)
    curve = norm.pdf(x, mu, sigma)
    curve = curve * horizons.shape[0] / sum(curve) /
        num_bins * curve_points * gmm_model.weights_[i]
    plt.plot(x / mtu_constant, curve, color='orange')

plt.xlim(xmin=0.0)
plt.xlabel('MTU')
plt.savefig(os.path.join(graph_folder, 'pred_horizons_histogram_separate_gmm.png'), dpi=300)

gmm_classes = gmm_model.predict(data)
means = [mean for mean in gmm_model.means_[:, 0]]
class_map = np.argsort(means)
name_map = ['hard', 'medium', 'easy']
gmm_classes = [name_map[np.argmax(class_map == bin_class)] for bin_class in gmm_classes]
plt.figure()
plt.suptitle('Easy, medium and hard initial conditions')
bin_values = [
    len([x for x in gmm_classes if x == 'hard']),
    len([x for x in gmm_classes if x == 'medium']),
    len([x for x in gmm_classes if x == 'easy'])
]
# plt.hist(gmm_classes, bins=num_bins, color='black')
plt.bar('hard', bin_values[0], color='black')
plt.bar('medium', bin_values[1], color='black')
plt.bar('easy', bin_values[2], color='black')
plt.xlabel('Difficulty')
plt.savefig(os.path.join(graph_folder, 'easy_medium_hard.png'))

plt.figure()
plt.suptitle('Distribution of probability')
x = np.linspace(min_hor, max_hor, curve_points)
p = gmm_model.predict_proba(x.reshape(-1, 1))
p = p.cumsum(1).T

plt.fill_between(x / mtu_constant, 0, p[0], color='gray', alpha=0.3)
plt.fill_between(x / mtu_constant, p[0], p[1], color='gray', alpha=0.5)
plt.fill_between(x / mtu_constant, p[1], 1, color='gray', alpha=0.7)
plt.xlim(xmin=0.0)
plt.ylim(0, 1)
plt.xlabel('MTU')
plt.ylabel('probability')
plt.savefig(os.path.join(graph_folder, 'probabilities.png'))
```

```
    return gmm_classes


# Calculate and draw error and std plots for each histogram bin

def calc_and_draw_per_bin(horizons, errors, graph_folder, difficulty_classes, mtu_constant,
threshold):
    Printer.print_log(True, 'Plotting error graphs per difficulty... ')

    fig, ax = plt.subplots(3)
    plt.suptitle('Mean error with standard deviation per difficulty')

    for i, difficulty in enumerate(['hard', 'medium', 'easy']):

        picked_horizons = [hor for i, hor in enumerate(horizons) \
            if difficulty_classes[i] == difficulty]
        picked_errors = [error for i, error in enumerate(errors) if \
            difficulty_classes[i] == difficulty]

        if len(picked_horizons) == 0:
            continue

        e = np.average(picked_errors, axis=0)
        std = np.std(picked_errors, axis=0)
        max_horizon = np.argmax(e > threshold)

        x_values = np.arange(e.shape[0]) / mtu_constant

        ax[i].set_title(difficulty)
        ax[i].plot(x_values, e + std, color='blue', alpha=0.4)
        ax[i].plot(x_values, e, color='red')
        ax[i].plot(x_values, e - std, color='green', alpha=0.4)
        ax[i].plot(x_values, [threshold for _ in range(e.shape[0])], 'r--', color='black')
        ax[i].set_xlim([0, max_horizon * 2 / mtu_constant])
        ax[i].set_ylim([0, threshold * 2])
        ax[i].set_ylabel('L2')

    fig.text(0.5, 0.04, 'MTU', ha='center')
    fig.tight_layout()
    fig.subplots_adjust(top=0.88, bottom=0.12)
    fig.savefig(os.path.join(graph_folder, 'error_std.png'), dpi=300)


# Main function starting the threads and summarizing the results

def main():

    # Loading Lorenz data

    data_set = DataSet('lorenz_96', 200, 1)
    data = data_set.data

    # Load default initialization parameters

    # model_params = Model.default_model_params('esn')
    model_params = Model.default_model_params('lstm')
    # model_params = Model.default_model_params('lstm2')
    # model_params = Model.default_model_params('ensemble_1')

    run_params = Model.default_run_params()

    # model = ModelEsn(model_params, data_set.path_prefix())
    model = ModelLstm(model_params, data_set.path_prefix())
    # model = ModelLstm2(model_params, data_set.path_prefix())
    # model = ModelEnsemble1(model_params, data_set.path_prefix())

    # Load existing prediction or load or train the model

    file_prefix = model.model_run_param_prefix_with_data(model.params_to_dict(), run_params)

    try:
        predictions = np.load(file_prefix + 'pred.npy')
        model = None
```

```
except OSError:
    Printer.add(True, 'training the model')
    model.load_or_train(data, True)
    predictions = np.zeros((run_params['num_i_c'],
        model_params['n_inp'], run_params['pred_len']))
    Printer.rem(True)

try:
    x_true_avg = np.load(file_prefix + 'x_true.npy')
except OSError:
    Printer.print_log(True, 'Computing average Xtrue... ')
    x_true_avg = np.zeros((run_params['pred_len'],))
    for init_cond in range(
            run_params['start_i_c'],
            run_params['start_i_c'] + run_params['num_i_c'] * run_params['i_c_distance'],
            run_params['i_c_distance']):
        for dt in range(run_params['pred_len']):
            x_true_avg[dt] += np.linalg.norm(data[:, init_cond + dt])
    x_true_avg /= run_params['num_i_c']
    try:
        np.save(file_prefix + 'x_true.npy', x_true_avg)
    except OSError:
        pass
    Printer.print_log(True, 'Computing average Xtrue... Done')

# Run threads

thread_count = 2
threads = []

horizons = np.zeros((model_params['n_inp'] + 1, run_params['num_i_c'], ))
errors = np.zeros((run_params['num_i_c'], run_params['pred_len']))

for i in range(thread_count):
    thread = threading.Thread(target=thread_main, args=(
        model_params, data, run_params, horizons, i, thread_count,
        model, x_true_avg, predictions, errors
    ))
    threads.append(thread)
    thread.start()
for thread in threads:
    thread.join()

# Save predictions if not already saved

try:
    np.save(file_prefix + 'pred.npy', np.array(predictions))
except OSError:
    pass

# Generate cumulative prediction horizon plot

Printer.clear(True)
np.random.seed()
graph_folder = os.path.join(data_set.path_prefix(), 'pred_horizon_') +
    str(np.random.random())[2:]
if not os.path.exists(graph_folder):
    os.mkdir(graph_folder)
with open(os.path.join(graph_folder, 'model_params.json'), 'w') as out_file:
    json.dump(model_params, out_file)
with open(os.path.join(graph_folder, 'default_run.json'), 'w') as out_file:
    json.dump(run_params, out_file)

threshold = model_params['n_inp'] * 0.0375

plot_horizon(run_params, data, model_params, horizons[0], graph_folder, data_set.dt_per_mtu)
print()

# Generate colored matrix

plot_all_errors(run_params, errors, graph_folder, data_set.dt_per_mtu)
print()

# Generate per variable prediction horizon plot
```

```
    plot_horizon_per_var(run_params, data, model_params,
        horizons[1:], graph_folder, data_set.dt_per_mtu)
    print()

    # Generate histogram

    difficulty_classes = plot_histogram(horizons[0], graph_folder, data_set.dt_per_mtu)
    print()

    # Generate histogram bin error graphs

    calc_and_draw_per_bin(horizons[0], errors, graph_folder,
        difficulty_classes, data_set.dt_per_mtu, threshold)
    print()


if __name__ == '__main__':
    main()
```