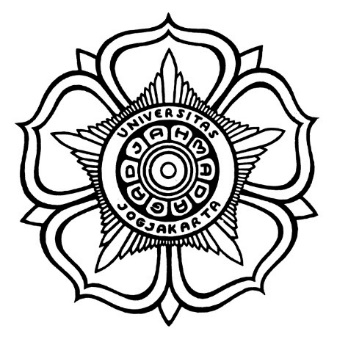
skripsi

*Unscented Kalman Filter* dan *recurrent Neural network* pada peramalan

*Unscented Kalman Filter* and *recurrent Neural network* for forcasting



ruli sastra putri

14/364140/pa/15914

PROGRAM STUDI ilmu komputer

DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS GADJAH MADA

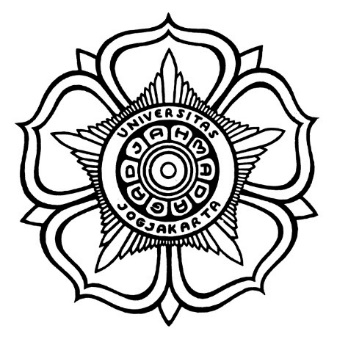
YOGYAKARTA

skripsi

*Unscented Kalman Filter* dan *recurrent Neural network* pada peramalan

*Unscented Kalman Filter* and *recurrent Neural network* for forcasting

Diajukan untuk memenuhi salah satu syarat memperoleh derajat



ruli sastra putri

14/364140/pa/15914

PROGRAM STUDI ilmu komputer

DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS GADJAH MADA

YOGYAKARTA

# HALAMAN PENGESAHAN

SKRIPSI

*Unscented Kalman Filter* dan *recurrent Neural network* pada peramalan

Telah dipersiapkan dan disusun oleh

RULI SASTRA PUTRI

14/364140/PA/15914

Telah dipertahankan di depan Tim Penguji

pada tanggal

Susunan Tim Penguji

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |

# PERNYATAAN

Dengan ini saya menyatakan bahwa Skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaan di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Yogyakarta,

Ruli Sastra Putri

HALAMAN MOTTO DAN PERSEMBAHAN

*Knowing is better than wondering, waking is better than sleeping, and even the biggest failure, even the worst, beats the hell out of never trying* (Meredith Grey)

*You belong wherever you want to belong* (Sophia Amoruso)

*Karya ini penulis persembahkan untuk orang tua, kakak-kakak, adik-adik, dan seluruh keluarga besar yang selalu memberikan harapan bagi penulis, juga segenap rekan-rekan serta pembaca sekalian.*

# PRAKATA

Puji syukur kehadirat Allah SWT atas limpahan anugrah, rahmat, karunia, serta petunjuk-Nya sehingga tugas akhir berupa penyusunan skripsi ini telah terselesaikan dengan baik.

Dalam penyusunan tugas akhir ini penulis telah banyak mendapatkan arahan, bantuan, serta dukungan dari berbagai pihak baik secara langsung maupun tidak langsung. Oleh karena itu pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Allah SWT dan Baginda Rasulullah SAW atas nikmat dan karunia yang telah diberikan pada penulis selama di dunia dan akhirat nanti.
2. Ibu dan Bapak yang dengan sabarnya menunggu, membimbing, memfasilitasi, dan memberikan kasih saying selama penulis menempuh pendidikan. Tidak lain keja keras ini hanya untuk membahagiakan kalian.
3. Bapak Agus Sihabuddin, S.Si., M.Kom., Dr. dan Ibu Faizah S.Kom., M.Kom. selaku dosen pembimbing tugas akhir yang senantiasa berkenan meluangkan waktunya untuk memberikan arahan dan bimbingan kepada penulis dalam penyusunan tugas akhir ini.
4. Ibu Aina Musdholifah, S.Kom., M.Kom., Ph.D. selaku dosen pembimbing akademik yang telah membimbing penulis selama menjalani masa perkuliahan di Ilmu Komputer UGM.
5. Ma Sist yang selalu ada saat ingin bercerita dengan motivasi dan kegigihannya menjadi tolak ukur kami adiknya, serta Kak Hendra kemudian Gendis yang dengan cerdasnya datang kedunia dan menceriakan Palembang dengan pipinya.
6. Mas Kibut dan Yuk Milda yang selalu ada untuk memberikan kepercayaan dan peluang untuk lebih mengeksplore diri. Terima kasih sudah melahirkan Babang yang menjadi sumber energi baru bagi penulis.
7. Saudari saudari dari Indoktang yang dengan teguh, strong menghadapi dunia beserta anak-anaknya yang dengan cerdas menjadi tumpuan pembelajaran baru bagi penulis.
8. Bapak/Ibu Dosen Ilmu Komputer UGM yang telah membimbing dan membantu penulis selama menjalani kuliah di Universitas Gadjah Mada
9. Teman-teman kuliah seluruh angkatan yang telah memberikan dukungan tiada habisnya. Menjadi tolak ukur keberhasilan yang kelihatannya tidak berusaha, padahal berjuang mati-matian.
10. Wiwik dan Dyah. Kalianlah yang menjadi pelita yang selalu ada. You are the only one.
11. Teman KKN Kemadang 2018 yang dengan komitmenya menjaga silaturahmi antar sesame.
12. Teman HIMAKOM UGM yang menjadi bagian manis dan pahit dalam berorganisasi.
13. Pihak-pihak lain yang tidak dapat disebutkan satu per satu.
14. Yang terakhir… Kamu. Terima kasih telah menjadi support utama di masa labil penulis. Kenangan manis terukir jelas di seluruh Jogja.

|  |  |
| --- | --- |
|  | Yogyakarta, 14 Juni 2019 Penulis    Ruli Sastra Putri |

# DAFTAR ISI

HALAMAN MOTTO DAN PERSEMBAHAN ii

1 BAB I PENDAHULUAN 12

1.1 Latar Belakang 12

1.2 Rumusan Masalah 13

1.3 Batasan Masalah 14

1.4 Tujuan Penelitian 14

1.5 Manfaat Penelitian 14

1.6 Metodologi Penelitian 15

1.7 Sistematika Penulisan 15

2 BAB II TINJAUAN PUSTAKA 17

3 BAB III LANDASAN TEORI 21

3.1 Bitcoin 21

3.2 Prediksi Data *Time Series* 21

3.3 *Neural Network* 22

3.4 Gaussian 30

3.5 Kalman Filter 30

3.6 Normalisasi 39

3.7 *Sliding Window* 39

3.8 Evaluasi 40

4 BAB IV ANALISIS DAN RANCANGAN SISTEM 43

4.1 Analilis Permasalahan 43

4.2 Rancangan Umum Sistem 43

4.3 Data 44

4.4 Normalisasi Data 45

4.5 Arsitektur RNN 47

4.6 *Unscented Kalman Filter* 48

4.7 RNN dan UKF 50

4.8 Prosedur *Training* 52

4.9 Rancangan Pengujian 55

5 BAB V IMPLEMENTASI 56

5.1 Lingkungan Implementasi 56

5.2 Data 56

5.3 Normalisasi Data 58

5.4 Pemisahan Data 58

5.5 Menentukan I/O JST 60

5.6 Implementasi RNN 62

5.7 Pengujian 71

6 BAB VI HASIL PENELITIAN DAN PEMBAHASAN 74

6.1 Pengujian Nilai Q dan R 75

6.2 Pengujian Arsitektur JST 77

6.3 Pengujian Ukuran *Sliding Window* 77

6.4 Perbandingan dengan RNN-Linear Kalman Filter 78

7 BAB VII KESIMPULAN 79

7.1 Kesimpulan 79

7.2 Saran 79

8 DAFTAR PUSTAKA 80

9 LAMPIRAN A SOURCE CODE 82

# DAFTAR TABEL

Tabel 1.1 Perbandingan Penelitian 19

Tabel 4.2 Cuplikan data yang sudah di normalisasi 46

Tabel 4.3 Parameter *training* 52

# DAFTAR GAMBAR

Gambar 3.1 Jaringan saraf biologis (Fausett, 1994) 22

Gambar 3.2 *Multilayer perceptron* (Fausett, 1994) 23

Gambar 3.3 Grafik fungsi identitas (Fausett, 1994) 25

Gambar 3.4 Grafik fungsi undak biner (Fausett, 1994) 25

Gambar 3.5 Grafik sigmoid biner (Fausett, 1994) 26

Gambar 3.6 Grafik hyperbolic tangent (Fausett, 1994) 27

Gambar 3.7 Distribusi normal (*gaussian*) 30

Gambar 3.8 Perbandingan a)Nilai awal, b)KF, c)UKF (Wan & Merwe, 2002) 33

Gambar 3.9 Persamaan KF dan UKF 34

Gambar 3.10 Ilustrasi *sliding window* 40

Gambar 4.11 Diagram alur penelitian secara umum 44

Gambar 4.12 Cuplikan data Bitcoin 45

Gambar 4.13 Visualisasi harga asli dan ternormalisasi 46

Gambar 4.14 Arsitektur RNN yang digunakan 47

Gambar 4.15 Ilustrasi sliding window 48

Gambar 4.16 Pseudocode UKF 50

Gambar 4.17 Flowchart gabungan RNN dan UKF 51

Gambar 4.18 Pseudocode forward pass 54

Gambar 5.19 Kode untuk membaca data.csv 56

Gambar 5.20 Data dibaca sebanyak 2359 *rows* x 1 *column* 57

Gambar 5.21 Kode untuk menormalisasi data 58

Gambar 5.22 Tabel normalisasi data dan contoh nilai [-1,1] 58

Gambar 5.23 Kode pemisahan data 59

Gambar 5.24 Variabel pemisahan data 59

Gambar 5.25 Kode input data 60

Gambar 5.26 Tabel input data dengan windowSize=1 61

Gambar 5.27 Tabel input data dengan windowSize=3 61

Gambar 5.28 Fungsi hyperbolic tangent dan turunannya 62

Gambar 5.29 Kode fungsi denormalisasi 63

Gambar 5.30 Penentuan banyaknya *neuron* jaringan 63

Gambar 5.31 Inisialisasi bobot jaringan 63

Gambar 5.32 Inisialisalisasi parameter dan jumlah bobot 64

Gambar 5.33 Forward pass 64

Gambar 5.34 Menghitung *error* dan *backward pass* 65

Gambar 5.35 Chain Rule 65

Gambar 5.36 Update bobot 65

Gambar 5.37 Penggabungan bobot 66

Gambar 5.38 Penentuan parameter titik sigma 66

Gambar 5.39 Penghitungan titik-titik sigma disekeliling mean 67

Gambar 5.40 Pembobotan titik-titik sigma 67

Gambar 41 Mean dan kovarian di measurement 68

Gambar 42 Kalman gain 69

Gambar 5.43 Update kovarian 70

Gambar 5.44 Innovation dan update bobot UKF 70

Gambar 5.46 Context layer baru 70

Gambar 5.47 Kode prediksi data uji 71

Gambar 5.47 Kode MAE 71

Gambar 5.49 Kode MSE 72

Gambar 5.49 Kode MAPE 72

Gambar 5.50 Kode RMSE 73

Gambar 5.51 Kode Dstat 73

Gambar 6.52 Grafik pemisahan data 74

Gambar 6.53 Grafik pemisahan data ternormalisasi 75

Gambar 6.54 Nilai Q 76

Gambar 6.55 Nilai R 77

# INTISARI

*Unscented Kalman Filter* dan *recurrent Neural network* pada peramalan

Ruli Sastra Putri

98/123919/PA/07519

Contoh untuk Intisari. Pada umumnya sistem perangkat lunak terdiri dari beberapa *concern*, premis dari masalah ini adalah sebaran *concern*, di mana kebutuhan rancangan tertentu cenderung memotong-melintasi grup inti fungsional modul. Teknik orientasi-objek yang menerapkan *concern* tersebut cenderung menghasilkan kode yang tersebar, daya baca yang sulit, serta susah untuk dikembangkan. Metodologi baru, *aspect-oriented programming* (AOP), memberikan fasilitas modularisasi pemotong-lintasan/*cross-cutting concern*. Dengan menggunakan AOP, terdapat cara untuk membuat penerapan sistem yang lebih mudah untuk dirancang, dipahami, dan dipelihara. Lebih jauh lagi, AOP menjanjikan produktivitas yang lebih tinggi, peningkatan kualitas, dan kemampuan lebih baik untuk menambahkan *feature* baru.

Contoh paragraf kedua. AspectJ adalah bahasa pemrograman yang digunakan secara luas untuk menerapkan program-program berorientasi aspek di Java. Namun demikian, AspectJ masih belum memiliki bahasa pemodelan yang dapat memenuhi perancangan program berorientasi aspek. Aspect Oriented Design Model (AODM), sebagai sebuah model perancangan baru pada pengembangan program dalam AspectJ, hanya memperluas konsep-konsep UML (Unified Modeling Language) yang telah ada dengan menggunakan mekanisme perluasan UML untuk memberikan konsep orientasi-aspek yang ada di dalam AspectJ. AODM menyediakan spesikasi model rancangan orientasi-aspek untuk ditransformasikan menjadi model rancangan UML biasa.

# ABSTRACT

*Unscented Kalman Filter* and *recurrent Neural network* for forcasting

Ruli Sastra Putri

98/123919/PA/07519

Example of Abstract. Most software systems consist several concerns, the premise of such thing is separation of concerns, where certain design requirements tend to cut across group of core functional modules. Object-oriented techniques for implementing such concerns result in systems that are invasive to implement, tough to understand, and difficult to evolve. The new aspect-oriented programming (AOP) methodology facilities modularization of crosscutting concerns. Using AOP, there is a way to create implementations that are easier to design, understand, and maintain. Further, AOP promises higher productivity, improved quality, and better ability to implement newer features.

Second paragraf. AspectJ is a well-established programming language that is widely used to implement aspect-oriented programs in Java. However, there is no modeling language available for the design aspect oriented programs in AspectJ. Aspect Oriented Design Model (AODM), as a new design model for development of AspectJ programs, extends existing UML (Unified Modeling Language) concepts using standard UML extension mechanisms to provide aspect-oriented concepts as in AspectJ. The AODM species how an aspect-oriented design model maybe transformed into an ordinary UML design model.

1. BAB I  
   PENDAHULUAN
   1. Latar Belakang

*Cryptocurrency* merupakan alat transaksi online alternatif yang fenomenal. Bersifat desentralisasi (tidak terpusat) untuk mengurangi pengaruh kuat dari pemerintah suatu negara (sistem moneter). *Cryptocurrency* tidak memiiki bentuk fisik (*virtual*) dengan diperoleh melalui “mining” oleh *user*. Faktor dimana tidak adanya kontrol dari pemerintah, jangkauan yang luas, serta penggunaannya yang bebas selama dikui kedua pihak dalam bertransaksi mengakibatkan *cryptocurrency* tidak stabil (fluktuasi). *Cryptocurrency* tidak dapat dinyatakan sebagai mata uang di Indonesia karena tidak memiliki karakteristik sebagai mata uang dibawah UU No. 7 Tahun 2011 (Kenawas & Sulistiowati, 2016). Walaupun kendala legalitas *cryptocurrency* masih dipertanyakan secara hukum di beberapa negara, sampai saat ini *cryptocurrency* masih dimiliki dan digunakan oleh banyak orang.

Bitcoin diklasifikasikan sebagai koin yang paling terkenal dan merupakan salah satu yang tertua dari teknologi *cryptocurrency*. Bitcoin ditemukan pada tahun 2008 oleh Satoshi Nakamoto (Nakamoto, 2009) yang sampai sekarang tidak diketahui identitas aslinya. Pada tahun 2013, Bitcoin sangatlah fenomenal. Pada awal tahun 2013, harga per-Bitcoin adalah $13 dan melambung menjadi $230 pada 9 April 2013 dengan potensial keuntungan mencapai hampir 1700% dalam kurun waktu kurang dari 4 bulan. Pada tahun yang sama, harganya meroket menjadi $395 pada 9 November 2013 dengan keuntungan mencapai 2900% sejak awal tahun 2013 (Kristoufek, 2013).

Pada awal bulan Mei 2018, harga Bitcoin turun hingga $9,674. Fluktuasi yang tinggi tersebut menjadi masalah dalam dunia *cryptocurrency.* Sama halnya dengan *trader*, penggguna *cryptocurrency* memiliki satu tujuan, yaitu mendapatkan keuntungan maksimal dengan kerugian seminimal mungkin. Oleh karenanya, permasalahan tersebut menarik banyak peneliti untuk mencoba memprediksi harga menggunakan banyak metode. Akan tetapi, kurangnya referensi sebagai dasar penelitian dan hasil akurasi yang tidak terlalu tinggi dibandingkan dengan data saham menjadi kendala untuk peneliti *cryptocurrency*.

Upaya peningkatan akurasi dapat dilakukan dengan mencari korelasi (hubungan) antara variabel bebas lainnya atau dengan melakukan prediksi dengan system dan algoritma yang tepat. Salah satu penelitian dilakukan (Kim, et al., 2016) dengan menggunakan komentar Twitter sebagai variabel bebas menggunakan algoritma *Machine Learning*. Penelitian tersebut menyatakan bahwa *cryptocurrency* dilakukan secara *online* dalam skala besar dimana banyak pengguna mengambil keputusan *hold*, *sell* dan *buy* bergantung dari informasi dari website*.* Pada hasil penelitian tersebut, akurasi yang didapatkan adalah sebesar ±70%. Peneitian (Martina, et al., 2015) menyatakanbahwa Twitter sebagai media jaringan sosial online dan *microblogging* telah menjadi alat yang penting untuk bisnis, komunikasi serta membagikan informasi dengan pertumbuhan yang cepat dan signifikan. Pada akhir penelitian, hasil yang didapatkan adalah *tweets* berpengaruh besar terhadap harga Bitcoin dalam beberapa hari dan Google Trends yang mampu untuk memprediksi Bitcoin dikarenakan nilai *cross-correlation* yang tinggi dan 0 *delay.*

Selain faktor, pemilihan model arsitektur sistem dan algoritma menjadi krusial dalam akurasi prediksi *cryprocurrency*.

* 1. Rumusan Masalah

Fluktuasi *cryptocurrency* yang tidak stabil dikarenakan sifatnya yang desentralisasi menjadi salah satu masalah pada prediksi harga Bitcoin. Sedikitnya referensi peneliti *cryptocurrency* dan hasil akurasi prediksi yang tidak terlalu tinggi dibandingkan penelitian menggunakan data saham, menjadi masalah lainnya. Upaya untuk mengatasi masalah ini adalah dengan menggunakan metode dan rancangan model prediksi lainnya yang memungkinkan untuk meningkatkan hasil akurasi prediksi.

Oleh karena itu, rumusan masalah penelitian ini adalah untuk membangun dan mengetahui hasil akurasi prediksi harga Bitcoin menggunakan metode *Unscented Kalman Filter Recurrent Neural Network*.

* 1. Batasan Masalah

Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Dataset harga Bitcoin diambil dari cyptodatadownload.com.
2. Penelitian ini mengggunakan metode *Unscented Kalman Filter* dan *Recurrent Neural Network.*
3. Data set Bitcoin merupakan harga BTC/USD diambil mulai dari 1 Januari 2011 sampai 31 Mei 2018 sebanyak 2359 data.
   1. Tujuan Penelitian

Tujuan penelitian ini adalah untuk membangun metode prediksi harga Bitcoin menggunakan R*ecurrent Neural Network* dan persamaan *Unscented Kalman Filter* sebagai update bobot. Penelitian ini juga bertujuan sebagai tolak ukur akurasi dan referensi peneliti lainnya.

* 1. Manfaat Penelitian

Manfaat yang dapat diambil dari penelitian ini, antara lain:

1. Menambah pengetahuan dan wawasan tentang model untuk memprediksi data *time series* harga Bitcoin.
2. Sebagai referensi peneliti lainnya di ruang lingkup masalah yang berhubungan agar menghasilkan metode prediksi yang lebih baik.
3. Dapat digunakan oleh pemain *cryptocurrency* dan pihak lainnnya berhubungan.
   1. Metodologi Penelitian

Adapun tahap-tahap penelitian ini dapat dijabarkan sebagai berikut:

1. Studi Literatur

Penelitina ini diawali dengan studi literatur mengenai topukk yang bersangkutan, yaitu prediksi data *time series.* Kemudian dilakukan pengumpulan teori, informasai, dan data pendukung mengenai prediksi data *time series* khususnya pada Bitcoin atau *cryptocurrency.* Prinsip kerja algoritma juga dipahami dan didalami.

1. Analisis dan Rancangan

Tahap ini, peneliti melakukan anlalisis terhadap model yang dibuat. Analisis dilakukan dengan mengindentifikassi masalah dan apa saja yang dibutunkan model adar dapat menyelesaaikan permasalahan prediksi Bitcoin. Model perancangan kemudian diteliti lagi dan diterapkan kedalam implementasi model.

1. Implementasi Model

Pada tahap ini, *Unscented Kalman Filter* kemudian diimplementasikan dengan *Recurrent Neural Network* kedalam program untuk mendapatkan nilai akurasi pada prediksi Bitcoin.

1. Pengujian

Pengujian dilakukan dengan beberapa metode evaluasi yang sesuai untuk pengukuran besar kecilnya *error* modelperancangan.

* 1. Sistematika Penulisan

Pada penulisan penelitian ini terdapat beberapa BAB yang masing-masing dapat ditulis menjadi:

BAB I PENDAHULUAN

BBAB ini memuat latar belakang, rumusan masalah, Batasan masalah, tujuan, manfaat, metodologi penelitian serta sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Tinjauan pustaka memuat beberapa penelitian terdahulu yang menjadi rujukan terkait topuk masalah, metode yang digunakan, dan menjadi bahan referensi dalam penelitian ini.

BAB III LANDASAN TEORI

BAB ini memuat teori-teori yang digunakan dalam penelitian ini.

BAB IV ANALISIS DAN RANCANGAN

Berisi benjelasan serta analisis permasalahan, system, data dan peranccangan terhadap sistem yang dibuat. Baik itu arsitektur, algoritma, dan metode pengujian.

BAB V IMPLEMENTASI

Memuat spesifikasi *hardware* dan *software* yangdigunakn dan hasil implementasi kode system yang dikembangkan berdasaarkan perancangan yang dilakukan beserta penjelasannya.

BAB VI HASIL PENELITIAN DAN PEMBAHASAN

Pada bab ini, dimuat rangkuman hasil penelitian dan pengujian berupa akurasi, waktu eksekuusi, dan permasalahan yang dihadapi berserta pembahasannya.

BAB VII PENUTUP

Berisi kesimpulan penelitian yang telah dilakukan disertai saran untuk penelitiana selanjutnya.

1. BAB II  
   TINJAUAN PUSTAKA

Berdasarkan penelitian (Kim, et al., 2016), terdapat korelasi (hubungan) antara komentar pengguna dan harga serta *volume* Bitcoin, Ethereum, dan Ripple. Pada penelitian ini, Kim dkk menggunakan metode *Machine Learning* menggunakan persamaan *Average One Dependence Estimator* sebagai validasi model yang digunakan. Komentar pengguna pada setiap forum koin dilakukan analisis sentiment secara manual. Hasilnya, metode tersebut berhasil digunakan dan didapatkan nilai akurasi ±70% untuk semua koin. Besarnya forum dan banyaknya data yang diolah juga mempengaruhi akurasi.

Metode peramalan juga dilakukan pada penelitian lain. Penelitian (Bisoi & Dash, 2014) memprediksi data trend saham menggunakan *Unscented Kalman Filter,* *Differential Evolution* dan DEUKF. Hasilnya didapatkan bahwa model *hybrid* DEUKF menghasilkan nilai MAPE lebih sedikit dibandingkan dengan UKF atau DE sendiri. Metode *hybrid* ini menghasilkan MAPE3.13 dengan 400 data IBM. Sedangkan UKF dan DE sebesar 4.62 dan 3.96.

Penelitian (Amjad & Shah, 2016) tentang perbandingan algoritma untuk prediksi Bitcoin menyatakan bahwa model ARIMA sangat buruk untuk hasil prediksi dan probabilitasnya. Penelitian tersebut mengunakan data dari 2014-2016 dengan perbandingan algoritma antara model *Empirical Conditional,* ARIMA, *Random Forest, Linear Descriminant Analysis* dan *Logistic Regression*. Hasil yang didapatkan algoritma RF (algoritma klasifikasi)menunjukkan performa yang lebih baik dengan akurasi (>60-70%) dan *Sharpe Ratio* (> 2.0)*.*

Prediksi Bitcoin (Albariqi, 2018) menggunakan *Multilayer Perceptron* dan RNN, menghasilkan bahwa MLP mengungguli RNN dengan *accuracy* 81,3%, *precision* 81% dan *recall* 94,7%. Data yang digunakan adalah data *Blockchain* pada tahun 2010-2017 dengan periode 2 hari. Sedangkan pada penelitian lain dilakukan prediksi mata uang USD menggunakan RNN dan *Extended Kalman Filter* (Hazazi, 2018)*.* Hasilnya didapatkan *Dstat* terbaik sebesar 73,06%, MSE 2425,64, RMSE 49,25 dan MAE 35,05. Data yang digunakan adalaj 70% data latih dan 30% data uji dengan model arsitektur 3-6-1.

Berdasarkan penelitian (Torres & Qiu, 2018), prediksi harga Bitcoin menggunakanmodel LSTM dan GRU menghasilkan RSME LSTM dan GRU sebesar 272,96 dan 274,02 dimana hasil tidak terlalu kentara. Tetapi pada performa jaringan, GRU lebih unggul 12% dibandingkan LSTM dikarenakan sistem GRU yang lebih sederhana. Namun pada penelitian yang dilakukan (Saptoro, 2012), digunakan model gabungan *Feedforward Neural Networks* dengan EKF dan UKF. Hasilnya, didapatkan nilai MAE pada UKF lebih unggul dengan peningkatan 2..45-21.48% (data pembelajaran)dan 8.35-29.15% (data uji) dibandingkan dengan EKF.

Salah satu penelitian menunjukkan bahwa EKF memiliki beberapa kekurangan dan melakukan penelitian pada metode alternative, yaitu UKF. Penelitian ini memodifikasi mean dan kovarian pada algoritma EKF. Hasil yang didapatkan adalah jelas, bahwa UKF menunjukkan kompleksitas dan performance yang terus menerus mengungguli EKF (Wan & Merwe, 2002). Sedangkan pada penelitian (Zhan & Wan, 2006), dilakukan perbandingan algoritma EKF, UKF, Particle Filter menggunakan arsitektur NN. Hasilnya, didapatkan bahwa metode gabungan PF-UKF menunjukkan akurasi dengan hanya 4% waktu komputasi yang membuat metode tersebut lebih cocok untuk dilakukan prediksi *real-time.*

Penelitian yang dilakukan pada penelitian ini menggunakan data *time series* harga Bitcoin sebagai *neuron input* model *Neural Network* dan UKF untuk pembobotan sebagai arsitektur model. Kemudian data di*preprocessing* dan normalisasi terlebih dahulu. Evalusi metode dilakukan penghitungan *Dstat,* RMSE, MAE, dan MAPE dalam mengetahui akurasi metode rancangan. Untuk perbandingan penelitian yang sudah ada dirangkum dalam **Error! Reference source not found.**

Tabel 1.1 Perbandingan Penelitian

| **No** | **Peneliti** | **Topik** | **Metode** | **Perbedaan** |
| --- | --- | --- | --- | --- |
| 1. | (Kim, et al., 2016) | *Predicting Fluctuations in Cryptocurrency Transaction Based on User Comments and Replies* | *Machine Learning* (persamaan *Average One Dependence Estimator*) | -Metode  -Data Bitcoin, Ethereum, dan Ripple  -Data dari forum web dan Twitter |
| 2. | (Bisoi & Dash, 2014) | *A hybrid evolutionary dynamic neural network for stock market trend analysis and prediction using unscented Kalman filter* | **-**UKF  -DE (Differential Evolution)  -DEUKF (hybrid) | **-**Metode  -Data (Data trend saham) |
| 3. | (Amjad & Shah, 2016) | *Trading Bitcoin and Online Time Series Prediction* | -ARIMA  -*Random Forest*  -*Logistic Regression*  -*Linear Discriminant Analysis*  -*Empirical Conditional* | -Perbandingan metode |

Tabel 2.1 Perbandingan penelitian (lanjutan)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Peneliti** | **Topik** | **Metode** | **Perbedaan** |
| 4. | (Albariqi, 2018) | *Prediksi Perubahan Harga Bitcoin Menggunakan Jaringan Syaraf Tiruan* | -*Multilayer Perceptron*  *-Recurrent Neural Networks.* | -Metode  -Data |
| 5. | (Saptoro, 2012) | *Extended and unscented kalman filters for artificial neural network modelling of a nonlinear dynamical system* | **-***Extended Kalman Filters*  *-Unscented Kalman Filters*  *-Feedforward Neural Networks* | -Metode  -Data  -Perbandingan |
| 6. | (Hazazi, 2018) | *Reccurent Neural Network dan Extended Kalman Filter untuk Peramalan Nilai Tukar Mata Uang* | -RNN  -EKF | -Metode  -Data |
| 7. | (Wan & Merwe, 2002) | *The Unscented Kalman Filter For Nonlinear Estimation* | -EKF  -UKF | -Data  -Arsitektur |
| 8. | (Zhan & Wan, 2006) | *Neural Network-Aided Adaptive Unscented Kalman Filter For Nonlinear State Estimation* | 1. NN-UKF 2. Particle Filter 3. PF-EKF 4. PF-UKF | -Data  -Metode |

1. BAB III  
   LANDASAN TEORI
   1. Bitcoin

Mata uang digital yang menggantungkan keamanannya dengan kriptografi. Bitcoin bersifat *Peer-to-peer* (P2P) *networking* atau *computing* merupakan aplikasi arsitektur sistem terdistribusi yang membagi-bagi pekerjaan ke setiap titik. Setiap node berfungsi baik sebagai penyedia maupun pengguna layanan. Berbeda dengan sistem *client-server*, *peer-to-peer* bersifat desentralisasi karena setiap titik memiliki hak yang serupa. Bitcoin juga menggunakan *hashcash* *proof-of-work* untuk keamanannya dalam bertransaksi. *Hashcash* adalah jenis *proof-of-work* yang selain diimplementasikan di Bitcoin, diaplikasikan juga untuk menyaring surel yang datang ataupun pesan yang ditujukan ke alamat IP (Internet Protocol). Metode ini dilakukan dengan menambahkan teks pada header pesan yang sudah memiliki bentuk, agar saat dilakukan fungsi *hash*, dapat dihasilkan *message digest* yang sesuai dengan kriteria; Bitcoin menggunakan SHA256 sebagai fungsi hash-nya (Mangan, 2013)

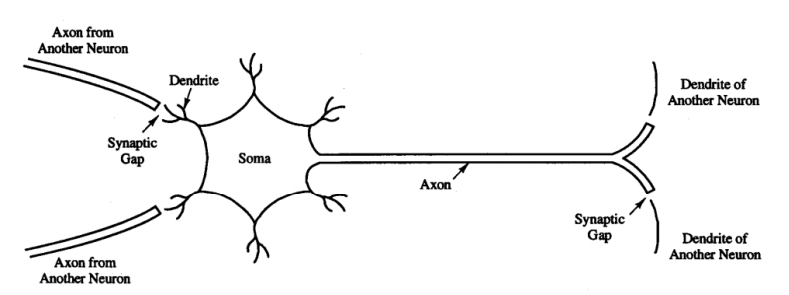
*Proof-of-work* adalah sebuah fungsi atau protokol yang diharapkan mampu mengagalkan *denial of service* ataupun berbagai penggunaan jasa berlebihan seperti *spam* dengan menuntut sebuah pekerjaan dilakukan oleh pengguna/pengaju jasa sebelum menggunakan jasa tersebut, biasanya mengakibatkan waktu proses saat dilakukan komputer. Kunci dari fungsi ini adalah asimetri; pekerjaan tersebut haruslah sulit (tetapi dapat dilakukan) dari pihak pengaju tetapi mudah diperiksa oleh pihak pemberi jasa. *Proof-of-work* berbeda dengan CAPTCHA, yang dimaksudkan untuk diselesaikan oleh manusia (Mangan, 2013).

* 1. Prediksi Data *Time Series*

Data *time series* merupakan nilai-nilai yang berurutan dan biasanya mempunyai atribut waktu. Data *time series* dapat ditemukan di berbagai ranah, contohnya keuangan, energi, klimatologi, dan meteorologi. Terdapat dua macam pengolahan untuk data *time series*, yaitu *time series analysis* dan *time series forecasting*. Analisis data *time series* bertujuan untuk menghitung sifat-sifat statistik data, sedangkan *time series forecasting* untuk mengetahui nilai pada masa mendatang (Gutierrez, et al., 2017)

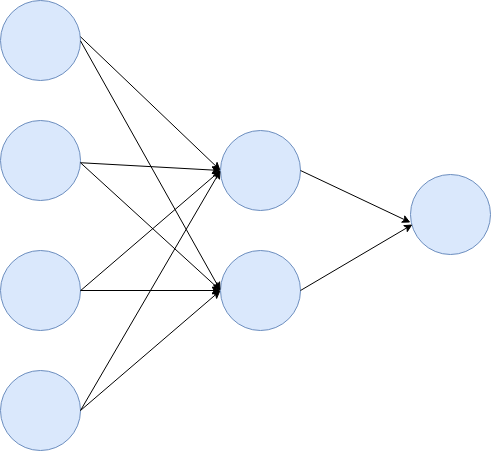
* 1. *Neural Network*

Jaringan saraf tiruan (JST) atau *Artificial Neural Network* (ANN) merupakan sistem pemrosesan informasi yang memiliki karakteristik seperti dengan jaringan saraf biologis (Fausett, 1994). NN dikembangkan sebagai bentuk generalisasi dari model matematika dari sistem pengetahuan manusia.



Gambar 3.1 Jaringan saraf biologis (Fausett, 1994)

Pada jaringan saraf biologis seperti pada **Error! Reference source not found.** neuron mendapat rangsang sebagai *input* ke dalam badan *neuron*, dan disampaikan ke *neuron* lain melalui akson dan dendrit. Pada NN, pola hubungan antar neuron disebit arsitektur jaringan. Sebagai contoh, diilustrasikan sebuah *multilayer perceptron* dengan 3 *neuron* pada lapisan *input*, 2 *neuron* dalam lapisan tersembunyi, dan 1 *neuron* pada lapisan *output* pada Gambar 3.2.



Gambar 3.2 *Multilayer perceptron* (Fausett, 1994)

Lapisan-lapisan penyusun NN dibagi menjadi tiga (Siang, 2009, p. 24), yaitu;

1. Lapisan *input*

*Neuron* di dalam lapisan *input* disebut neuron *input*. Neuron *input* menerima *input* dari luar, *input* yang diberikan merupakan penggambaran suatu permasalahan.

1. Lapisan *hidden*

*Neuron* di dalam lapisan initersembunyi. Output dari lapisan ini tidak dapat diamati secara langsung.

1. Lapisan *output*

*Neuron* di dalam lapisan *output* disebut neuron *output*. *Output* dari lapisan ini merupakan hasil dari *Neural Network* terhadap suatu permasalahan.

*Neuron-neuron* dalam NN berfungsi sebagai elemen pemroses yang berfungsi seperti halnya sebuah *neuron* pada hewan. Sejumlah sinyal *input* dikalikan dengan masing-masing bobot yang bersesuaian (penentuan nilai bobot disebut sebagai metode *training*). Kemudian dilakukan penjumlahan dari seluruh hasil perkalian tersebut dan *output* yang dihasilkan digunakan untuk mendapatkan *output* jaringan.

Berdasarkan (Fausett, 1994, pp. 12-15), *training* NN ada 2 macam, yaitu *training* terbimbing (*supervised training*) dan *training* tidak terbimbing (*unsupervised training*).

Secara garis besar *supervised training* merupakan model *training* dengan dataset yang telah dilabel. Tipenya adalah klasifikasi dan regresi(misalnya prediksi trend menggunakan data yang telah dilabel sebelumnya). Contoh algoritmanya adalah *Back Propagation.*

*Unsupervised training* tidak diberikan pengetahuan sepanjang *training*. Tipenya adalah klasterisasi (menemukan pola dan pengelompokan dari data tanpa pelabelan). Selama proses *training*, nilai bobot disusun dalam suatu *range* tertentu tergantung pada nilai *input* yang diberikan. Terdapat vektor *input* namun tidak ada vektor target. NN melakukan pembaruan bobot sehingga *input* dengan karakteristik serupa dapat digolongkan menjadi satu klaster. Contohnya adalah *self-organizing NN*.

* + 1. Komponen Neural Network

Seperti pada jaringan saraf biologis, pada NN terdapat *node* yang berperan seperti *neuron*. Pemrosesan informasi berlangsung pada *neuron*-*neuron* tersebut. *Neuron* terhubung dengan *neuron* lainnya menggunakan *connection link* dan memiliki bobot (*weight*) yang terus diperbarui nilainya. Bobot akhir ini yang digunakan apabila ada nilai *input* baru yang diproses untuk mengetahui nilai *output*nya. Ada tahap untuk melakukan 10 pembaruan bobot sampai perulangan yang ditentukan atau sampai nilai pembaruannya tidak berubah secara signifikan. (Fausett, 1994)

Fungsi Aktivasi

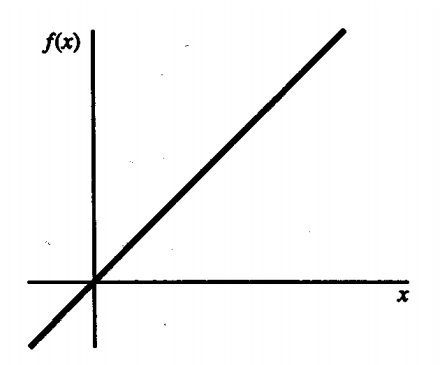
Fungsi aktivasi adalah nilai yang nantinya akan digunakan untuk *propagated* (membawa) nilai dari *layer* ke *layer* padapersamaan *forward pass*. Biasanya dilambangkan dengan (sigma kecil). Terdapat beberapa macam fungsi aktivasi yang ada pada *Neural Network*.

1. Fungsi linear (identitas)

Fungsi identitas tidak mengubah nilai yang masuk ke *neuron* tersebut. Fungsi linear dirumuskan pada pers.(3.1).**Error! Reference source not found.**

|  |  |
| --- | --- |
| untuk setiap x | (3.1) |

Kemudian pers. (3.1) dapat digambarkan seperti pada Gambar 3.3

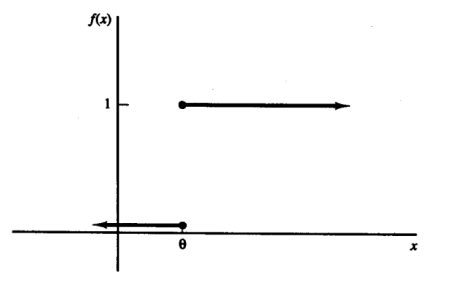


Gambar 3.3 Grafik fungsi identitas (Fausett, 1994)

1. Fungsi undak biner (*binary step function*)

*Neuron* yang mempunyai fungsi undak biner (*binary step function*) menghasilkan *output* 0 atau 1. Terdapat nilai ambang atau *threshold* (θ) yang menjadi batas apakah nilai yang dihasilkan adalah 0 atau 1. Persamaan untuk fungsi undak biner dituliskan dalam pers. (3.2).

|  |  |
| --- | --- |
|  | (3.2) |



Gambar 3.4 Grafik fungsi undak biner (Fausett, 1994)

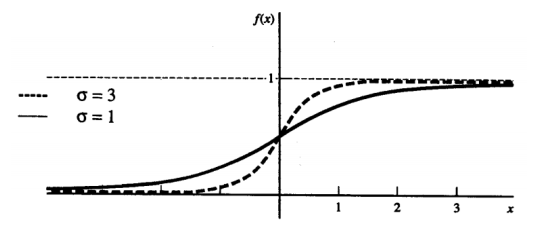
Pada Gambar 3.4, terlihat bahwa grafik yang dibentuk menghasilkan kurva yang lancip dan menunjukkan bahwa nilai yang dihasilkan dari fungsi aktivasi ini adalah 0 atau 1. Nilai *output* bernilai 1 apabila nilai yang masuk lebih besar atau sama dengan nilai ambang yang ditentukan dan akan bernilai 0 apabila nilai yang masuk ke dalam *neuron* lebih kecil dibanding nilai ambang.

1. Sigmoid biner

Fungsi aktivasi sigmoid biner mengubah nilai yang masuk ke dalam *neuron* menjadi antara 0 maupun 1. Sigmoid biner dirumuskan dalam pers.(3.3).

|  |  |
| --- | --- |
|  | (3.3) |

Pada pers.(3.3), semakin besar nilai *x* membuat menjadi 1 dan apabila nilai *x* semakin kecil membuat nilai menjadi 0. Terdapat parameter *σ* yang menentukan nilai yang dihasilkan yang disebut dengan *steepness parameter*. Kurva yang dihasilkan berbentuk seperti huruf S seperti pada Gambar 3.5.

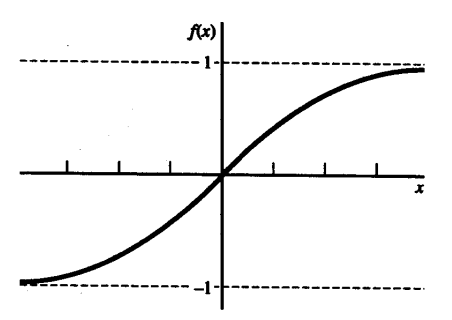


Gambar 3.5 Grafik sigmoid biner (Fausett, 1994)

Pada Gambar 3.5, kurva dengan garis putus-putus menggunakan *σ* = 3, sedangkan garis lurus menggunakan *σ* = 1.

1. Bipolar sigmoid dan *tanh* (*hyperbolic tangent*)

Bipolar sigmoid erat kaitannya dengan fungsi *tanh*. Kurva yang dihasilkan oleh fungsi aktivasi *hyperbolic tangent* juga berbentuk seperti huruf S. Yang membedakan dengan sigmoid biner adalah batas nilai minimal dari *hyperbolic tangent* bukan 0, melainkan -1, seperti pada Gambar 3.6. Oleh karenanya, jangkauan nilai yang dihasilkan dari fungsi aktivasi *hyperbolic tangent* adalah dari -1 sampai 1.



Gambar 3.6 Grafik hyperbolic tangent (Fausett, 1994)

Fungsi bipolar sigmoid dapat dipetakan dari fungsi pada pers.(3.3) sebelumnya dan menjadi pers.(3.4) dibawah.

|  |  |
| --- | --- |
|  | (3.4) |

Pers.(3.4) dapat membuat batas atas (1) dan batas bawah (-1). Kemudian, apabila pers.(3.3) disubstitusikan ke dalam pers.(3.4), diperoleh pers.(3.5).

|  |  |
| --- | --- |
|  | (3.5) |
|  | (3.6) |
|  | (3.7) |

Sedangkan fungsi *tanh* *h*(*x*) apabila diatur *σ* sebesar 1 dirumuskan dalam pers.(3.6). Pers.(3.7) merupakan persamaan *tanh* yang diturunkan. Dapat dilihat pada pers.(3.5) dan pers.(3.6) masing-masing merupakan *tanh* saat dan bipolar sigmoid saat akan memiliki hasil yang sama. Oleh karena itu, sigmoid bipolar dan *tanh* dapat dikatakan erat kaitannya*.*

***Learning Rate* (α)**

*Learning Rate* adalah salah satu parameter *training* untuk menghitung nilai koreksi bobot pada waktu proses *training.* Dengan kata lain sebagai parameter penentu ketelitian dan kecepatan sebuah arsitektur dalam pembelajaran. Nilai α berada pada *range* [0,1]. Semakin besarα,maka semakin cepat proses *training* dan *error*nya. Tetapi jika nilai α terlalu besar, maka proses *training* dapat melampaui keadaan optimal (keadaan *error* minimal).

**Bobot**

Bobot (*Weight*) merupakan nilai matematis yang mengekspresikan kekuatan antar *neuron* dari *layer* ke *layer*. Tiap hubungan mempunyai nilai bobot sendiri. Nilai ini biasanya menjadi parameter penentu dalam optimasi arsitektur *neural network* dalam suatu masalah akurasi.

* + 1. *Recurrent Neural Network*

*Recurrent Neural Network* (RNN) merupakan salah satu bentuk dari NN dimana pada RNN memungkinkan *output* dari lapisan tersembunyi dapat dijadikan *input* kembali sehingga dapat memperhitungkan riwayat data yang telah dilalui (Fausett, 1994).

Penelitian ini menggunakan arsitektur Elman RNN yang dikembangkan oleh Elman dimana *output* *unit* tersembunyi disimpan dan digunakan lagi untuk menjadi *input* data setelahnya. Pada Elman RNN, terdapat *context unit*, yaitu *neuron* yang berfungsi untuk menyimpan nilai *output* dari unit tersembunyi sebelumnya. Nilai yang disimpan di dalam *context unit* akan dilakukan perhitungan dengan nilai selanjutnya sebagai nilai *input* seperti pada pers.(3.9)

|  |  |  |
| --- | --- | --- |
|  | | (3.8) |
|  | | (3.9) |
|  | | (3.10) |
| Keterangan: | |  |
| : *neuron* pada *context layer* | : *neuron* pada *input layer* | |
| : *neuron* pada *hidden layer* | : *bias* pada *hidden layer* | |
| : *neuron* pada *output layer* *s* | :fungsi aktivasi | |

RNN menggunakan *forward pass* untuk “propagate” atau membawa *value* pada *input* melewati tiap *unit* (*neuron*) pada *layer* tersembunyi ke *output layer* yang nanti akan dihitung *error*nya.

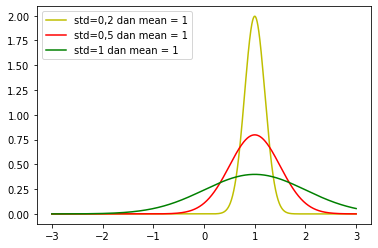
Selain forward pass, pada *Recurrent Neural Network,* juga dibutuhkan *backward pass* untuk mengecilkan nilai *loss* function yang biasa disebut dengan *backpropagation. Backpropagation* berisi perhitungan matematis turunan dari *forward pass*.Menggunakan *chained rule* untuk mencari gradiennya. Gradien dapat dihitung dengan mencari nilai *error* terkecil untuk setiap perulangan yang dijalankan pada arsitektur dengan pers.(3.11).

|  |  |  |
| --- | --- | --- |
|  | | (3.11) |
|  | | (3.12) |
|  | | (3.13) |
| Keterangan: | |  |
| : Nilai *error* di *output layer* | : *output* yang diharapkan | |
| : (phi) turunan fungsi aktivasi | : *transpose* dari bobot | |
| : (delta) pada *layers* |  | |

Pers.(3.13) menunjukkan turunan dari perhitungan *layer* ke *hidden layer* pada *backpropagation*. Dimana persaamaan didapatkan dari perhitungan gradien bobot setiap *layer* (dari kanan ke kiri) secara berurutan terhadap *error* yang didapatkan pada *output layer* pers.(3.14)

* 1. Gaussian

Dikenal juga sebagai distribusi normal, gaussian merupakan salah satu metode pengukuran kumpulan data. Dalam statistika, sebuah kumpulan data dikatakan berdistribusi normal akan menunjukkan bahwa probabilitasnya kontinyu. Terdapat dua parameter utama, yaitu *mean* () dan *variance* (). Dimana *mean* merupakan nilai rata-rata, sedangkan variasi menunjukkan seberapa besar pengukuran (*measurement*)bervariasi dari *mean* seperti pada Gambar 3.7.



Gambar 3.7 Distribusi normal (*gaussian*)

Sedangkan korelasi data merupakan metode statistika untuk melihat hubungan antara dua data (variabel). Ada hubungan, berarti satu variabel mempengaruhi variabel lainnya. Korelasi antara dua variabel akan bernilai positif (+) atau negatif (-) jika terdapat korelasi dan bernilai 0 jika tidak memiliki korelasi sama sekali. Sedangkan tanda positif atau negatif menunjukkan korelasi data yang berlawanan arah (-) atau sebaliknya (+). Nilai kovarian didapatkan dari nilai *variance* yang dibagi dengan banyaknya jumlah data.

* 1. Kalman Filter

*Kalman filter* merupakan persamaan matematis yang termasuk kedalam salah satu penemuan penting dalam prinsip teori kontrol dan diterbitkan pada artikel (Kalman, 1960). Paling banyak diaplikasikan pada sistem dinamis yang kompleks seperti proses *manufacturing*, penerbangan, perkapalan, dan pesawat luar angkasa (digunakan di sistem pemandu). Sekarang, *Kalman filter* tidak hanya digunakan pada mesin, tetapi juga pada bidang ekonomi, grafis dan lainnya.

*Kalman filter* sendiri terdapat dua komponen utama, yaitu *process* *equation* (prediksi)dan *measurement* (*update*). *State* vektor () didefinisikan sebagai himpunan data yang secara unik mendeskripsikan tingkah laku sistem yang dinamis, dimana *t* menunjukkan waktu diskrit. Sederhananya, *state vector* adalah vektor yang berisi semua variabel sistem yang relevan dari sistem (Haykin, 2004).

**Proses (*Process Equation*)**

Pada persamaan, terdapat lambang *subscript* (bawah) dan *superscript* (atas) untuk menandakan perbedaan waktu *value* yang yang digunakan. Pada tahap pertama Kalman Filter, dihitung proses *predict equation* ditunjukkan pada pers.(3.14)

|  |  |  |
| --- | --- | --- |
|  | | (3.14) |
|  | | (3.15) |
| Keterangan: | |  |
| : matriks transisi | : *state* pada *t.* | |
| : Kovarian *noise* dari *process* | : *expected value* (*mean*) | |

**Pengukuran (*Measurement Equation*)**

|  |  |  |
| --- | --- | --- |
|  | | (3.16) |
|  | | (3.17) |
| Keterangan: | |  |
| : *observable* pada waktu *t* | :matriks *measurement* | |
| : Kovarian *noise measurement* |  | |

*Noise* dari dan diasumsikan aditif dan *Gaussian* dengan *zero mean* dan memiliki kovarian dan dengan dan tidak saling berkorelasi (hubungan). melambangkan nilai yang diharapkan pada matriks tersebut, dalam hal ini adalah *mean* yang diasumsikan *zero mean*. Permasalahan Kalman *filter* terletak pada penggabungan solusi *Process* dan *Measurement equation* untuk *state* yang tidak diketahui dengan optimal. Semua data yang didapatkan harus digunakan, terdiri dari vektor untuk menentuan MSE dari *state* untuk semua *t ≥ 1.* Kalman *filter* bekerja dalam dua tahap berulang:

Kovarian sendiri secara umum merupakan ukuran kekuatan seberapa besar perubahan yang dihasilkan antara satu variabel dengan variabel lainnya. Secara sederhana, sebagai pengukur hubungan antar variabel secara linear. Dimana hubungan tersebut ditunjukkan dengan nilai (+) dan (-) yang berarti kedua variabel berhubungan (entah naik atau turun). Sedangkan kovarian dengan nilai 0 menunjukkan bahwa keduanya tidak berhubungan sama sekali (*zero covariance*).

***Time Update (predict)***

Disebut juga sebagai tahap prediksi dengan menghitung nilai *error covariance* pada pers. (3.18) dan pers. (3.19) untuk prediksi *state* *.*

|  |  |  |
| --- | --- | --- |
|  | | (3.18) |
|  | | (3.19) |
| Keterangan: | |  |
| :prediksi *state* | : fungsi matriks transisi | |
| :Kovarian *noise* dari *process* | :Kovarian prediksi | |

***Prediction Update (measurement)***

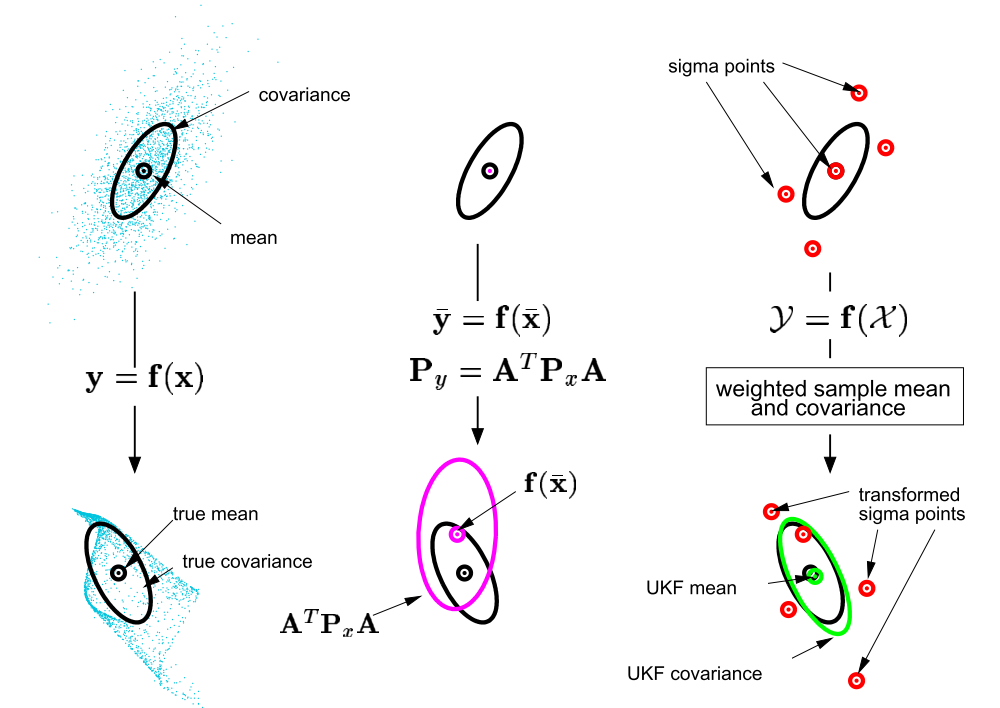
Dikenal juga dengan tahap koreksi (*measurement*), dimana dihitung *Kalman gain* yang digunakan untuk mengkoreksi prediksi dari *state* yang dihasilkan pada tahap *time update* berdasarkan *measurement* dari . Perhitungan pada tahap ini dijabarkan pada pers.(3.21) sampai pers.(3.24).

|  |  |  |
| --- | --- | --- |
|  | | (3.20) |
|  | | (3.21) |
|  | | (3.22) |
|  | | (3.23) |
|  | | (3.24) |
| Keterangan: | |  |
| : *noise measurement* | : H transpose | |
| *K* : Kalman *gain* | *'* :prediksi *state* | |
| :kovarian *error* | z:observable | |

Nilai *state* terbaru dihitung dengan pers.(3.23) dan kovarian diupdate dengan pers.(3.24) dimana sebelumnya dihitung *Kalman gain* dan residual *error* pengukuran pada pers.(3.20).

* + 1. Perbedaan KF dan UKF

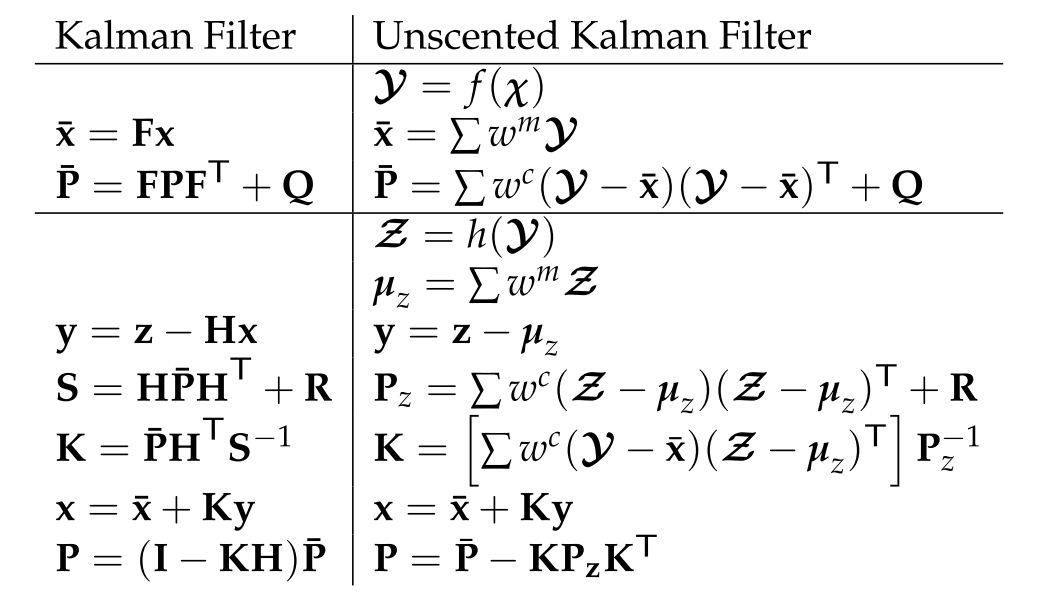
Ilustrasi perbedaan Linear Kalman Filter dan UKF dapat dilihat pada Gambar 3.8.



Gambar 3.8 Perbandingan a)Nilai awal, b)KF, c)UKF (Wan & Merwe, 2002)

Himpunan *sigma points* dipilih dan digunakan sebagai *sample* untuk *mean* () dan kovarian (P). Ketika himpunan tersebut diperbanyak melalui sistem *nonlinear* sesungguhnya, maka dapat menangkap *mean posterior* (sebelum) dan kovariannya untuk fungsi nonlinear.

Selain pada Gambar 3.8, perbedaan diantara keduanya dapat dilihat dari persamaan pada Gambar 3.9 tanpa *superscript* dan *subscript*. Persamaan UKF kemudian dijabarkan pada pers.(3.25) sampai pers.(3.40).



Gambar 3.9 Persamaan KF dan UKF

Pada gambar, masing-masing metode KF dan UKF memiliki persamaan prediksi (atas) dan *measurement* (bawah) yang dilambangkan dengan *subscript* (*z*). Dimana hasil akhir dari pembobotan UKF akan dihitung dari *Kalman gain* pers. dan *update* kovarian pada pers.(3.25) sampai pers.(3.40).

* + 1. *Unscented kalman filter*

UKF merupakan salah satu metode pengembangan kalman filter yang bertujuan untuk mengupdate bobot dengan menggunakan *sigma points*. Metode yang digunakan mirip dengan tipe metode Monte Carlo, dimana sejumlah sample dipilih secara acak yang juga diperbanyak melalui transformasi nonlinear. Di sisi lain, sigma points tidak dipilih secara acak, melainkan secara deterministik dan jumlah sigma points yang rendah

Perbedaan persaamaan KF dan UKF dapat dilihat pada Gambar 3.9. Sedangkan penjelasan persamaan UKF (Wan & Merwe, 2002) dapat dijabarkan sebagai berikut.

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **Menghitung titik-titik sigma** | | | |
|  | |  | (3.25) |
|  | |  | (3.26) |
|  | |  | (3.27) |
| Keterangan: | | |  |
| : (chi) titik-titik *Sigma* | :rata-rata (*mean*) | | |
| : matriks kovarian | :dimensi data | | |
| :parameter | :parameter | | |
| : parameter jauhnya penyebaran dari *mean* | | | |

(chi) menunjukkan matriks dari *sigma points* (titik-titik sigma) yang dicari dengan jumlah . Hal penting yang perlu diperhatikan adalah ukuran matriks tersebut. Dimana setiap kolom menunjukkan satu setpoin sigma. Misalkan pada 2 dimensi data, ukuran matriks menjadi 2 x 5. Hal ini dikarenakan setiap dimensi memiliki 5 titik sigma. Sedangkan jika , maka ukuran matriks menjadi 3 x 7. Maka, jika salah satu titik sigma adalah mean, maka sisanya dihitung berdasarkan pers.(3.26) dan pers.(3.27).

Menghitung *value* titik-titik sigma ke- dengan sebagai kolom ke- dari matriks akar kuadrat .

(parameter penyebaran) merupakan faktor skala yang memberi tahu harus seberapa jauh dalam pemilihan titik sigma. Semakin besar maka akan semakin jauh titik sigma dari *mean* tersebut. Terdapat beberapa versi untuk pemilihan parameter . Dimana menurut (J. Julier & Uhlmann, 1997), nilai optimal adalah . Sedangkan inisialisasi parameter dapat langsung ditentukan (Wan & Merwe, 2002) dengan dimana (kappa) sebagai parameter skala kedua setelah dan biasa di-*set* 0 dan .

|  |  |  |
| --- | --- | --- |
| 1. **Komputasi bobot untuk titik-titik sigma yang telah ditentukan** | | |
|  |  | (3.28) |
|  |  | (3.29) |
|  |  | (3.30) |
| Keterangan: |  |  |
| :bobot *mean* ke 0 | :bobot kovarian ke 0 | |

Titik-titik sigma tersebut memiliki bobot terhadap *mean* yang telah ditentukan sebelumnya. Pembobotan ( ke-0 dari titik-titk sigma tersebut memiliki persamaan yang berbeda dari sisa titik sigma yang lain . Hal menarik disini, jumlah semua bobot bernilai = 1 walaupun juga bisa lebih berdasarkan (Roger & Labbe, 2018)**.** Kemudian vektor sigma tersebut di “*propagated*” atau dibawa ke sistem fungsi nonlinier () pada pers.(3.31)

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **Tahap prediksi dengan perhitungan *mean* dan *kovarian*** | | | |
|  |  | | (3.31) |
|  | | | (3.32) |
|  | | | (3.33) |
| Keterangan: | | |  |
| :prediksi *mean* | | :fungsi non linier | |
| : prediksi kovarian | | :(chi) matriks titik sigma | |
| : *noise* di proses | | :sigma awal | |

Pada tahap prediksi, setiap *sigma points* dimasukkan kedalam fungsi non-linear . Kemudian dihitung *mean* prediksi pers.(3.32) dan kovarian prediksi (3.33), dengan menghitung perbedaan antar nilai *mean* dan kovarian yang diprediksi dengan nilai dari data sebenarnya. Perhitungan ini memiliki prosedur yang mirip dengan persamaan di Kalman Filter. Jadi yang dilakukan selanjutnya adalah mengambil *predicted state* ke *measurement state.*

Disini, terdapat pilihan dalam menghasilkan titik sigma lagi. Pilihan pertama adalah menentukan titik sigma dari awal lagi karena nilai prediksi *mean* dan varian yang berubah karena saling mempengaruhi satu sama lain. Pilihan lainnya adalah dengan menggunakan titik sigma yang telah dihasilkan. Untuk menggunakan titik sigma yang sebelumnya, maka diambil *state* dari *state space* ke *measurement state space* seperti pada tahap 4 pers.(3.34).

|  |  |
| --- | --- |
| 1. **Tahap update measurement** | |
|  | (3.34) |
|  | (3.35) |
|  | (3.36) |
| Keterangan: |  |
| :titik sigma yang ditransformasi ke dalam *measurement space* | |
| : *mean* di *measurement* *space* | |
| : kovariansi di *measurement* *space* | |
| : *noise* di *measurement space* | |
| : fungsi yang memetakan titik sigma yang ditetapkan sebelumnya ke *measurement* *space* | |

Penting: merupakan *measurement space,* yaitu pengukuranyang berasal dari sensor (sistem RNN). Jadi fungsi bertugas untuk mengubah *state space* menjadi *measurement space* sehingga dapat disamakan ke dalam satu unit.

Disini tidak perlu dihitung nilai Jacobian seperti pada Kalman Filter karena disini tidak akan dilakukan linierisasi fungsi (membuat garis fungsi). Untuk menghitung *Kalman gain*, perlu dilakukan perhitungan korelasi (hubungan) silang dengan dilakukan transformasi ( antara titik sigma di *state space* dan titik sigma di *measurement space.*

|  |  |
| --- | --- |
| 1. **Perhitungan Kalman Gain** | |
|  | (3.37) |
|  | (3.38) |
| Keterangan: |  |
| :(*cross covariance*) korelasi (hubungan) silang matriks antara *state space* dan *predicted space* | |
| : Kalman gain |  |

*Kalman gain* dihitung menggunakan pers.(3.38) dengan menghitung *cross covariace* terlebih dahulu seperti yang sudah dijabarkan sebelumnya. Persamaan untuk menghitung nilai *sama* dengandi Kalman Filter dan di EKF

|  |  |  |
| --- | --- | --- |
| 1. **Final state** | | |
|  | | (3.39) |
|  | | (3.40) |
| Keterangan: | |  |
| : *mean* data sebenarnya | : *mean* di *measurement space* | |
| : *mean state* terbaru | :prediksi *mean* | |
| : kovarian | : prediksi kovarian | |

Langkah terakhir adalah menghitung *state* terbaru menggunakan residual (sisa dari pengurangan) *Kalman gain* dengan pers.(3.38) dan kovarian terbaru dengan pers.(3.40). *Mean state* terbaru disini merupakan *state* (*x*) bobot yang di transformasi kembali seperti diawal pers.(3.25) karena UKF bertujuan *update bobot* pada penelitian UKF dan RNN.

* 1. Normalisasi

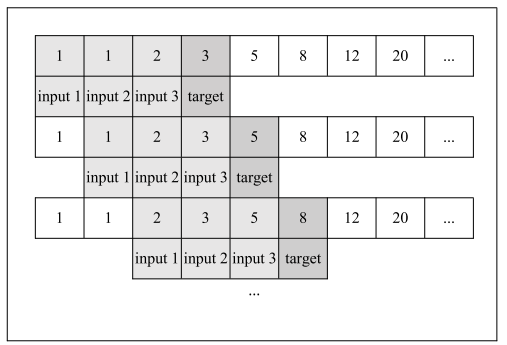
Normalisasi merupakan tahapan dimana data yang didapatkan disamakan jangkauannya. Hal ini dilakukan agar hasil yang didapatkan optimal dengan sistem yang dirancang. Biasanya, nilai data disamakan dengan jangkauan niai real [0,1] atau [-1,1]. Hal ini sangat diperlukan dan berguna apabila data berupa data mentah dengan jangkauan (perbedaan) yang sangat jauh. Dengan normalisasi, dapat dihitung jauhnya perbedaan dan hubungan nilai antar satu data dengan yang lainnya.

Sebelum di proses di arsitektur, data dilakukan normalisaasi terlebih dahulu agar jangkauannya sama. Normalisasi data dinyatakan seperti pada pers.(3.41). Pada tahap akhir sebelum evaluasi, data dikembalikan seperti data sebenarnya dengan denormalisasi pers.(3.42) untuk dapat dilihat grafik dari prediksi dan data sebenarnya.

|  |  |  |
| --- | --- | --- |
|  | | (3.41) |
|  | | (3.42) |
| Keterangan: | |  |
| : Data | : Batas bawah | |
| : Nilai minimum data | : Batas atas | |
| : Nilai maksimum data |  | |

* 1. *Sliding Window*

Digunakan untuk menentukan besarnya *sliding window* data model *time series*. *Sliding window* menentukan besarnya jumlah hari yang digunakan untuk menentukan nilai prediksi *output*. Pada pengujian dan analisis data latih, didapatkan hasil MAPE terkecil dan Dstat terbesar untuk menentukan ukuran *sliding window* yang diimplementasikan pada model RNN-UKF.



Gambar 3.10 Ilustrasi *sliding window*

Sebagai contoh dipilih *sliding window* dengan ukuran 3 untuk sebuah data *time-series*. Maka digunakan data untuk meramalkan nilai ke-t. Begitu seterusnya untuk meramalkan nilai , hingga nilai ke . Metode *sliding window* diilustrasikan seperti pada Gambar 3.10.

* 1. Evaluasi

Evaluasi dilakukan untuk mengetahui performa model yang dirancang. Dalam penelitian ini, dilakukan penghitunan MAE, RMSE dan MAPE serta Dstat. seperti pada pers.(3.43) sampai pers.(3.47).

* + 1. *Mean absolute error*

Salah satu kriteria pengukuran akurasi prediksi adalah menggunakan *Mean Absolute Error* (MAE). MAE menentukan rata-rata selisih antara nilai prediksi terhadap nilai sebenarnya untuk buah data seperti dirumuskan pada pers.(3.43).

|  |  |
| --- | --- |
|  | (3.43) |

* + 1. *Mean squared error*

MSE adalah suatu ukuran akurasi prediksi dengan mengkuadratkan masing-masing *error* untuk masing-masing pengamatan dalam sebuah susunan data dan kemudian memperoleh rata-rata jumlah kuadrat tersebut. MSE memberikan bobot yang lebih besar terhadap *error* dibandingkan dengan *error* yang kecil, sebab *error* dikudratkan sebelum dijumlahkan. MSE dapat dihitung dengan pers.(3.44).

|  |  |
| --- | --- |
|  | (3.44) |

* + 1. *Mean absolute percentage error*

Nilai MAE relatif tergantung jarak jangkauan nilai minimal dan maksimal data sehingga agar nilai akurasi dapat dijadikan ke dalam bentuk persen. Untuk mendapatkan nilai dalam bentuk persen, *Mean Absolute Percentage Error* (MAPE) digunkan, yaitu dengan membagi selisih dengan nilai sebenarnya agar diperoleh perbandingan terhadap nilai sebenarnya. MAPE dirumuskan seperti pada pers.(3.45).

|  |  |
| --- | --- |
|  | (3.45) |

* + 1. *Root mean squared error*

Root-mean-squared *error* (RMSE) adalah nilai akar pangkat 2 dari rata-rata *error* kuadratnya. RMSE dirumuskan seperti pada pers.(3.46).

|  |  |
| --- | --- |
|  | (3.46) |

* + 1. *Directional statistic*

Directional statistics atau Dstat merupakan tingkat akurasi yang lebih umum digunakan dalam bidang ekonomi khususnya finansial (Pangayom, 2017). Berbeda dengan perhitungan *error*, Dstat mengukur tingkat kebenaran naik atau turunnya nilai hasil prediksi terhadap nilai naik atau turun yang sebenarnya sehingga nilai Dstat yang lebih tinggi yang diharapkan. Nilai Dstat dirumuskan seperti pada pers.(3.47).

|  |  |
| --- | --- |
|  | (3.47) |

Dimana 𝑎𝑡 bernilai 1 apabila dan bernilai 0 apabila .

Dalam prediksi, nilai *error* yang diharapkan adalah nilai yang seminimal mungkin karena menunjukkan bahwa perbedaan nilai yang diramalkan dengan nilai sebenarnya cukup kecil atau hasil prediksi mendekati nilai yang sebenarnya. Untuk Dstat diharapkan nilai yang maksimal karena mendeskripsikan ketepatan arah prediksi.

1. BAB IV  
   ANALISIS DAN RANCANGAN SISTEM
   1. Analilis Permasalahan

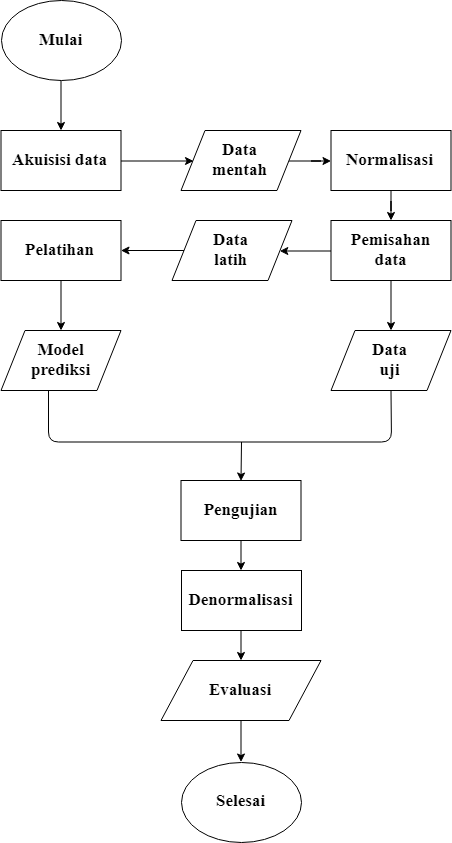
Pada penelitian ini dilakukan penghitungan untuk mencari akurasi dengan performa algoritma RNN-UKF dengan data *time series*. Data *time series* harga Bitcoin diambil dari 1 April 2017 sampai 1 Desember 2019 sebanyak 975 data. Data Bitcoin kemudian dilakukan normalisasi menggunakan pers.(3.41). Setelah dilakukan normalisasi data, data ternormalisasi dibagi menjadi data latih dan data uji dengan perbandingan (6:6), (7:3), atau (8:2). Untuk membangun model rancangan diperlukan data latih yang dari sini, data latih dibagi lagi menjadi data latih dan validasi untuk optimasi model perancangan. Data latih dan uji dengan perbandingan terbaik saat *training* dan kemudian dilakukan pengujian di model RNN-UKF. Setelah dilakukan pengujian dan implementasi model rancangan, dilakukan denormalisasi data dengan menggunakan kebalikan dari persamaan normalisasi pada awal data mentah. Setelah implementasi, evaluasi performa metode dilakukan dengan perhitungan Dstat, RMSE, MAE, MSE dan MAPEnya. Secara umum, langkah-langkah penelitian yang dilakukan dialurkan pada Gambar 4.11

* 1. Rancangan Umum Sistem

Rancangan umum sistem memiliki langkah-langkah yang dilakukan seperti:

1. Mengumpulkan data *time series* harga Bitcoin.
2. Melakukan praproses data dengan normalisasi data. Jangkauan data diubah menjadi skala yang lebih sesuai jangkuan nilai fungsi aktivasi pada *neuron*.
3. Memisahkan data menjadi dua bagian, yaitu data *training* dan data uji.
4. Melakukan *training* data pada data *training* untuk membuat model prediksi.
5. Melakukan update bobot dengan UKF dalam *training*.
6. Melakukan pengujian pada data uji dengan menggunakan model yang telah dilatih sebelumnya.
7. Menghitung akurasi dari model yang sudah dilatih terhadap data.

Rancangan dilakukan dengan tahapan tahapan seperti yang dilustrasikan pada diagram Gambar 4.11.

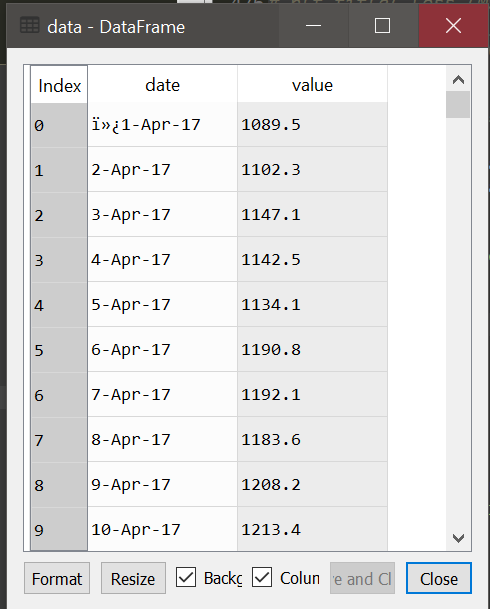


Gambar 4.11 Diagram alur penelitian secara umum

Analisis data juga dilakukan untuk menemukan performa terbaik dari sistem *neural network* yang dirancang dan algoritmanya dalam memprediksi *time-series* data. Penelitian ini menggunakan rancangan sistem dengan pengembangan algoritma *Neural Network.* Pengembangan tersebut adalah dengan menggunakan metode *Recurrent Neural Network* dan *Unscented Kalman Filter* (RNN-UKF).

* 1. Data

Pada Gambar 4.12, dapat dilihat data *time series* Bitcoin yang diperoleh dari cryptodatadownload.com. dan merupakan harga Bitcoin terhadap USD. Data tersebut terdiri dari dua variabel, yaitu tanggal dan harga *close* pada saat itu. Data yang diambil dan simpan dalam format .csv dibagi menjadi data latih dan uji.



Gambar 4.12 Cuplikan data Bitcoin

* 1. Normalisasi Data

Sebelum dilakukan *training*, dilakukan tahap praproses terlebih dahulu. Tahap praproses yang dilakukan adalah normalisasi jangkauan nilai atau melakukan normalisasi minimum-maksimum. Normalisasi dengan membuat skala jangkauan nilai menjadi lebih kecil diberlakukan pada variabel harga Bitcoin. Nilai yang besar diperkecil dengan normalisasi agar jangkauannya menjadi [-1,1].

|  |  |
| --- | --- |
|  |  |

Gambar 4.13 Visualisasi harga asli dan ternormalisasi

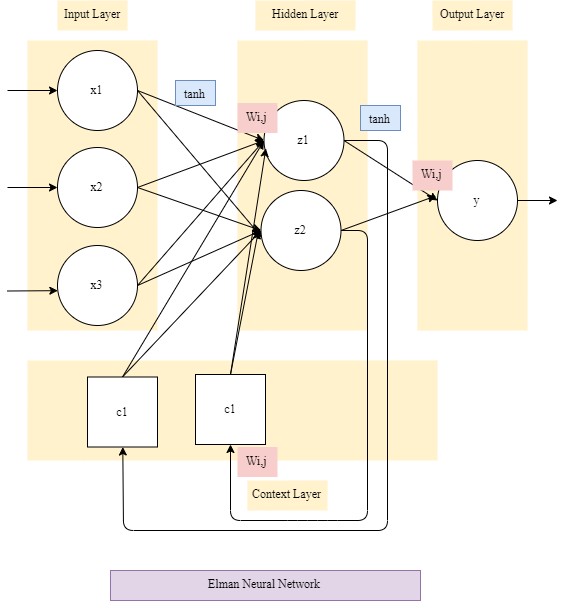
Visualisasi data seluruhnya (kiri) dan data yang sudah ditentukan jangkauan (kanan) dapat dilihat pada Gambar 4.13, sedangkan data *float* dapat dilihat pada Tabel 4.2 dengan jangkauan [-1,1]. Nilai jangkauan untuk menormalisasi data dilakukan dengan menggunakan pers.(3.41). Cuplikan jangkauan data dapat dilihat pada Gambar 4.13 pada setiap axis *y*.Tabel 4.2 Cuplikan data yang sudah di normalisasi

Tabel 4.2 Cuplikan data yang sudah di normalisasi

|  |  |  |
| --- | --- | --- |
| Date | Price (USD) | Normalisasi |
| 1-Apr-17 | 1,089.50 | -1 |
| 2-Apr-17 | 1,102.30 | -0.998599 |
| 3-Apr-17 | 1,147.10 | -0.993694 |
| 4-Apr-17 | 1,142.50 | -0.994198 |
| 5-Apr-17 | 1,134.10 | -0.995117 |
| 6-Apr-17 | 1,190.80 | -0.98891 |
| 7-Apr-17 | 1,192.10 | -0.988768 |
| 8-Apr-17 | 1,183.60 | -0.989698 |
| 9-Apr-17 | 1,208.20 | -0.987005 |
| … | … | … |
| 1-Dec-19 | 7,303.50 | -0.319703 |

* 1. Arsitektur RNN

Pada *Neural Network*, terdapat bermacam jenis dan arsitektur. Salah satunya adalah *Recurrent Neural Network*. *Neural Network* sendiri biasanya mempunyai 3 *layer* (*input, hidden, output*). Pada dasarnya, RNN menggunakan 3 *layers* tersebut. Pada beberapa arsitektur RNN, lebih dari 2 *hidden layer* digunakan. Tetapi penambahan *hidden layer* tidak menjamin akurasi pada prediksi. Terdapat banyak faktor untuk menjamin didapatkan nilai akurasi terbaik. Salah satunya adalah analisis data, performa algoritma dan rancangan sistem. Menurut (McNally, 2016) terlalu banyak *node* (50-100) dapat mempunyai performa, tetapi dapat meningkatkan kemungkinan *overfitting*. Maka karena itu, pada penelitian digunakan antara 10, 20, 30, 40 *neuron* pada *hidden layer*.

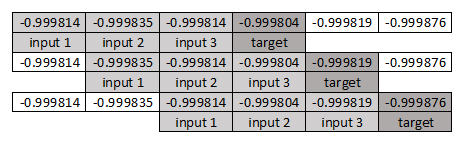


Gambar 4.14 Arsitektur RNN yang digunakan

Arsitekrut pada Gambar 4.14, digunakan *Elman Recurrent Neural Network* dimana selain tiga layer tersebut, digunakan juga *context layer* untuk menyimpan nilai dari *hidden layer* pada *timestep t-1* untuk digunakan kembali sebagai *input* pada *hidden layer* pada *timestep t*. Nilai pada *context layer* terus di-*update* hingga *training* selesai dengan jumlah *neuron* yang sama dengan *neuron* pada *hidden layer* seperti pada Gambar 4.14 Arsitektur RNN yang digunakanGambar 4.14.

Pada sistem ini, digunakan RNN dengan *neuron input* merupakan nilai dari variabel *input* yang digunakan pada *timestep t* yaitu nilai *float* dari Bitcoin. Jika menggunakan satu *neuron* *input*, maka data yang dimasukkan adalah data (kemarin) untuk memprediksi data (hari ini). Jika menggunakan tiga *neuron*, maka digunakan (tiga hari kebelakang).

Pada *neuron input*, *input* yang digunakan diilustrasikan seperti pada **Error! Reference source not found.**. Pada gambar, diilustrasikan *Sliding Window* yang digunakan adalah 3 data, dengan data ke-4 menjadi target. Kemudian begitu seterusnya untuk *neuron* *input* sampai *n*.



Gambar 4.15 Ilustrasi sliding window

Pada *hidden layer*, digunakan *neuron* dengan fungsi aktivasi *hyperbolic tangent*. Pengaruh jumlah *hidden layer*, diuji dengan variasi dari satu unit *neuron* hingga 10 *neuron*. Pada *output layer*, digunakan *many-to-one*. Yaitu satu *neuron* sebagai *output* yang ditetapkan oleh *hyperbolic tangent* agar niai yang keluar pada output tidak terjadi *exploding gradient* atau malah *vanishing gradient.*

Pada setiap *backward pass*

* 1. *Unscented Kalman Filter*

Algoritma pembelajaran yang digunakan pada penelitian ini merupakan algoritma gabungan dari *Recurrent Neural Network* dan update bobot *Unscented Kalman Filter*. Berdasarkan persamaan matematis (Wan & Merwe, 2002), *Unscented Kalman Filter* memiliki persamaan dan tahapan seperti seperti berikut:

1. Hitung kumpulan titik sigma.
2. Tetapkan bobot untuk setiap titik sigma.
3. Transformasikan titik tersebut melalui fungsi non-linier.
4. Hitung gaussian (*mean* dan kovarian) dari titik yang dibobot (*weighted*) dan ditransformasikan sebelumnya.
5. Hitung *mean* dan *kovarian* baru.

**Langkah Prediksi**

Maka terdapat langkah prediksi yang mirip dengan 5 langkah tadi, yaitu:

1. Hitung titik sigma (sama dengan langkah 1)
2. Hitung bobot titik sigma (sama dengan langkah 2)
3. Mengubah titik sigma dan hitung *mean* dan kovariana barudengan pertimbangan *noise* dari proses (**R**)

Yang pada tahap ini, pseudocode dari UKF dapat dirumuskan menjadi:

Gambar 4.16 Pseudocode UKF

MULAI :

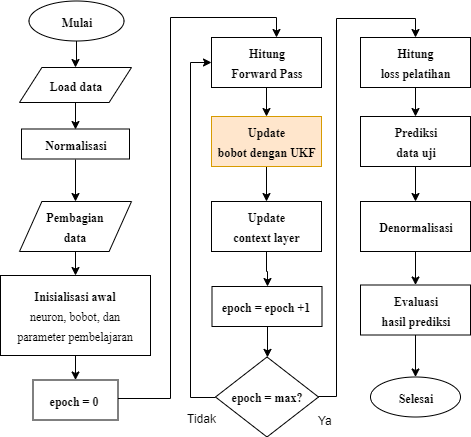
1. Input : Mean *µ*, kovarian, sigma points , weighted sigma points
2. Output : Kalman gain *K*, updated mean *µ*, updated kovarian *P*
3. *µ ←* concatenate(*wih, wh, who*)
4. *sigmas[0] ← µ*
5. *Sigma points* ← *sigmas[k+1], sigmas[n+k+1]*
6. *Wm, Wc* ← λ, *α, β, κ*
7. *K ← TutPz-1*
8. *µ ← µ’ + Kz*
9. *P ← P – (KPzKT)*

SELESAI

Gambar 4.17 menunjukkan lima tahap menghitung *value* pembobotan UKF ke RNN sesuai pers.(3.25) sampai pers.(3.40) dimana secara lebih rinci diperlihatkan pada implementasi kode dengan lima tahap UKF yang telah dijelaskan sebelumnya.

* 1. RNN dan UKF

Pada arsitektur rancangan dengan harapan dapat menaikkan akurasi dari prediksi data, maka penggabungan antara 2 metode dilakukan dapat diilustrasikan dengan ttahap sebagai berikut:



Gambar 4.17 Flowchart gabungan RNN dan UKF

Dapat dilihat pada Gambar 4.17 bahwa inisialisasi awal dilakukan sebelum perulangan (*epoch*) *training* dengan persiapan data sebelumnya. Kemudian persamaan *Unscented Kalman Filter* dilakukan dengan penjabaran pers.(3.25) sampai pers.(3.40), atau secara sederhana seperti yang telah disimpulkan pada Gambar 3.9 untuk mendapatkan *mean* dan kovarian baru terhadap semua bobot yang ada pada RNN selama *epoch training*.

Kemudian selama *epoch*, akan dilakukan analisis terhadap hasil data *training* yang ada untuk menentukan parameter terbaik yang akan dilakukan pada proses pengujian. Analisis ini dibantu dengan adanya nilai *loss* pada pelatihan. Besaran nilai *loss* diharapkan menurun selama *epoch* berlangsung untuk menunjukkan bahwa pelatihan berhasil dijalankan dalam usaha mendapatkan hasil akurasi terbaik. Setelah analisis dan pemilihan parameter terbaik dipilih, maka dilakukan *test* uji.

Proses pengujian dilakukan dengan sisa data yang belum tersentuh selama pelatihan yang telah dibagi sebelumnya. Hasil kemudian disimpan dalam *array* *value* dan dihitung evaluasinya dengan MAE, MSE, RMSE, MAPE, dan Dstat. dimana secara lebih rinci dijabarkan pada prosedur *training* dan rancangan pengujian.

* 1. Prosedur *Training*

Pada penelitian, dilakukan langkah-langkah yang terdiri dari beberapa tahap, yakni:

* + 1. Data latih

Persentase pemisahan data yang digunakan adalah 70% data dari data keseluruhan yang telah dinormalisasi. Data yang terhitung adalah data sebanyak 975 data.

* + 1. *Preprocessing*

Sebelum pembagian data latih dan uji, normalisasi dilakukan agar jangkauan nilai harga Bitcoin tidak terlalu jauh. Normalisasi dilakukan agar hasil prediksi yang diharapkan dapat maksimal. Normalisasi data akan menghasilkan nilai [-1,1] yang kemudian akan dilakukan pemilihan data dengan fungsi aktivasi *hyperbolic tangent*. Penghitungan dilakukan pada data dengan jumlah input yang telah ditentukan dengan metode *sliding window*.

* + 1. Inisialisasi bobot awal

Terdapat tiga bobot awal yang ditentukan secara random [-1,1] untuk digunakan dalam aristektur RNN yaitu sebagai betikut:

1. sebagai bobot dari *input* ke *context layer.*

2. sebagai bobot dari *context* ke *hidden layer.*

3. sebagai bobot dari *hidden* ke *output layer.*

* + 1. Inisialisasi parameter *training*

Parameter pembelajaran ditentukan pada awal sistem. Dalam sistem ini, parameter pembelajarannya adalah sebagai berikut:

Tabel 4.3 Parameter *training*

|  |  |  |
| --- | --- | --- |
| No. | Parameter | Penjelasan |
| 1. | *Process noise covariance Q* | Matriks berukuran , sebagai jumlah semua bobot yang ada pada semua *layers*. |
| 2. | *Measurement noise covariance R* | Matriks berukuran , sebagai jumlah *neuron* pada *output layer.* |
| 3. | Jumlah *epoch* | Diterapkan antara 50-100 *epoch* |
| 4. | Fungsi aktivasi | Fungsi aktivasi *tanh untuk forward pass.* |
| 5. | Parameter UKF | untuk penentuan titik-titik sigma dalam *update* bobot RNN |

Inisialisasi awal *Q* dan *R* memiliki nilai dimana merupakan banyak data dan merupakan matriks identitas sebagai *noise* dalam persamaan UKF yang dianggap aditif dan Gaussian dengan *zero mean*. Kemuadian dilakukan pengujian terhadap nilai untuk mengetahui pengaruh nilai *Q* dan *R* terhadap hasil prediksi.

Sedangkan parameter UKF di tentukan untuk mengetahui seberapa jauh sigma points dari *mean.* Parameter-parameter ini di-*set* dengan nilai *float,* Karena kecilnya nilai ini, maka terdapat dua rekomendasi utama parameter UKF dalam buku (Roger & Labbe, 2018), yaitu parameter berdasarkan penelitian (Wan & Merwe, 2002) dan penelitian yang dilakukan (J. Julier & Uhlmann, 1997) dimana keduanya akan dilakukan pada saat *training* data untuk melihat pengaruhnya.

* + 1. Proses update bobot dengan UKF

Algoritma UKF terdapat di dalamnya 2 bagian penting, yaitu tahap prediksi dan tahap *update* yang dipaparkan pada pers.(3.25) sampai pers.(3.40) dengan *pseudocode* pada Gambar 4.16 dan kelima tahapnya. Perlu digaris bawahi, bahwa *mean* awal untuk menentukan titik sigma lainnya merupakan gabungan dari *value* semua bobot yang ada pada jaringan. Titik-titik sigma itu sendiri berjumlah (2n+1) dimana salah satunya adalah *mean*. Titik-titik tersebut secara berurututan dihitung ke dalam matriks.

Hasil akhir dari perhitungan matematis UKF adalah nilai *mean* ter-*update* dan nilai *error* dari kovarian. Nilai pada *mean* dan kovarian tersebut didapatkan dari pencarian nilai *Kalman gain* pada pers.(3.38).

* + 1. Perhitungan *Forward Pass*

Perhitungan *forward pass* adalah langkah yang umum digunakan dalam JST. *Forward pass* adalah proses perhitungan maju (*propagated* untuk menghitung hasil *output* dari sistem RNN yang telah dibuat. Proses *forward pass* meliputi beberapa tahapan.

1. Mengambil nilai *neuron* *input* pada *input input* dengan *window size.*
2. Menghitung nilai *neuron* pada *hidden layer* sesuai dengan pers.(3.9).
3. Menghitung nilai *neuron* pada *output layer* sesuai dengan pers.(3.10).

Gambar 4.18 Pseudocode forward pass

MULAI :

1. Input : Bobot *wih*, Bobot *wch*, Bobot *wh*o, Nilai di lapisan masukan *Ui*,

nilai di *context layer Cc*

1. Output : Nilai di lapisan tersembunyi *hh*, Nilai di lapisan luaran *yo*
2. *hh* ← σ(*wihUi* + *wchCc*)
3. *yo* ← σ(*whohh*) SELESAI

*Forward pass* memerlukan fungsi aktivasi yang dalam penelitian ini digunakan fungsi *tanh* pada pers.(3.6). Penjelasan algoritmanya dijelaskan pada Gambar 4.18 yang merupakan *pseudocode* dari perhitungan *forward pass* (maju) setiap *layer* yang ada. Setelah tahap 3 dilakukan, dihitung nilai *error* dari nilai *output* prediksi dengan nilai aslinya.

* + 1. Perhitungan Prediksi

Prediksi data masing-masing dilakukan pada tahap pelatihandan uji. Pada tahap pelatihan ini, UKF menghitung *update* bobot dari inisialisasi bobot awal pada semua *layer*. Ketika bobot baru didapatkan, bobot tersebut diterapkan pada data uji dalam proses pengujian selanjutnya. Akan tetapi, sebelum masuk ke tahap pengujian pada data uji, *training* yang berhasil dijalankan dengan RNN-UKF, dihitung besar *errornya.* Pada tahap ini, akan jauh lebih baik jika pada setiap *epoch* nilai tersebut terus mengecil. Karena jika *error-*nya terus berkurang, hal ini menandakan bahwa arsitektur dan metode yang dirancang terus “belajar”.

Saat pengujian pada data uji selesai, data kemudian didenormalisasi kembali agar jangkauannya sama dengan data sebenarnya (harga Bitcoin dalam USD). Setelah didenormalisasi, dilakukan penghitungan akurasi MAE, MSE, MAPE, RMSE, Dstat untuk mengetahui kekuatan model rancangan dan besarnya *error* terhadap data sebenarnya.

* 1. Rancangan Pengujian

Membangun model prediksi *neural network* membutuhkan beberapa parameter *training* karena setiap arsitektur NN yang berbeda menghasilkan akurasi yang berbeda juga. Maka dilakukan pengujian yang meliputi:

1. Pengujian laju pembelajaran.
2. Pengujian arsitektur JST untuk perbandingan.
3. Pengujian ukuran *sliding window*.
4. Pengujian parameter UKF.
5. Pengujian akurasi MAE, MSE, MAPE, RMSE, Dstat untuk mengetahui kekuatan model rancangan.

Tujuan dari pengujian ini adalah untuk mendapatkan akurasi dengan *error* seminimal mungkin dengan melakukan persiapan dan rancangan sistem sebaik mungkin dengan teori dan penelitian sebelumnya

1. BAB V  
   IMPLEMENTASI
   1. Lingkungan Implementasi

Penelitian ini menggunakan spesifikasi sistem seperti berikut:

1. Laptop Asus VivoBook AMD Rayzen 5, 8 GB RAM

2. Sistem operasi Windows 10 64bit

3. Python 3.7.1 64bit

4. IDE Spyder

5. Library Pandas, Numpy, Matplotlib

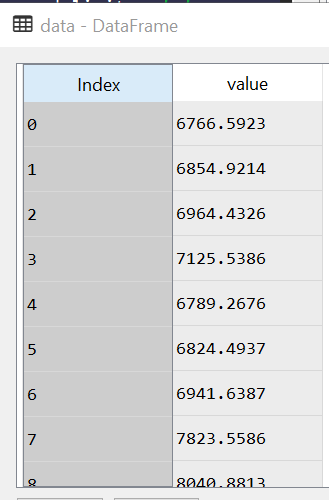
* 1. Data

Data diambil dari cyptodatadownload.com dalam file .csv. File data tersebut memiliki 2 kolom, yaitu kolom “Date” dan “Close Price” yang kemudian akan dilakukan proses implementasi pada sistem. Setelah dilakukan pembacaan data, maka dilakukan perubahan jenis data *value* sebagai data float32.

Gambar 5.19 Kode untuk membaca data.csv

1. import pandas as pd
2. import numpy as np
3. data = pd.read\_csv('data.csv',usecols=[1],
4. engine='python',
5. delimiter=',',
6. decimal=".",
7. thousands=',',
8. header=None,
9. names=['date','value'] )
10. data['value'] = data['value'].values
11. data['value'] = data['value'].astype('float32')

Cuplikan data dapat dilihat pada Gambar 5.20 dibawah. Data tersebut dibaca menggunakan *library* dari pandas sebagai *DataFrame* dan hanya menggunakan satu kolom, yaitu kolom harga. Data di-*import* dengan *pseudocode* dari Gambar 5.19 sebelumnya.



Gambar 5.20 Data dibaca sebanyak 2359 *rows* x 1 *column*

Setelah di*-import*, data kemudian masuk ke tahap *preprosessing* dengan menormalisasi jangkauan. Persiapan data dilakukan agar proses penelitian lebih rapi dan diharapkan memperoleh hasil akhir yang baik. Data yanag digunakan adalah data pada kolom ke-1 yang merupakan kolom *value* dan menghasilkan data *size* berukuran *n* x *m* untuk baris dan kolomnya.

* 1. Normalisasi Data

Untuk proses *preprosessing*, dilakukan normalisasi terlebih dahulu dengan jangkauan data [-1,1] menggunakan pers.(3.41). Data kemudian disimpan ke dalam variabel normalized. Fungsi normalize dirangkai sedemikian rupa dalam kode menggunakan variabel data yang diberikan dan scale sebagai skala yang diinginkan [-1,1] seperti kode pada Gambar 5.21.

Gambar 5.21 Kode untuk menormalisasi data

1. def normalize(data, scale):
2. normalized = []
3. for i in *range*(len(data)):
   * a = (min(scale))+(data[i]-min(data))\*(max(scale)-min(scale))/(max(data)-min(data))
4. normalized.append(a)
5. return np.array(normalized)

Data tersebut dinormaslisasi dengan *function* pada kode diatas yang menghasilkan data *float* dengan *scale* [-1,1]*.* Data ternormalisasi kemudian menjadi *value* data *array dengan* ukuran 2 dimensi (*m* x *n*) untuk baris x kolom seperti pada gambar dibawah.



Gambar 5.22 Tabel normalisasi data dan contoh nilai [-1,1]

Jika Gambar 5.21 menunjukkan kode normalisasi, maka Gambar 5.22 menunjukkan *value* dari *array* (pada variabel data\_raw dengan satu kolom (kebawah). Saat data sudah diimplementasikan ke *Recurrent Neural Network,* akan didapatkan data prediksi ter-*denormalisasi* yang kemudian baru diukur besarnya *error* antara data sebenarnya dan data prediksi. Perumusan *denormalisasi* dilakukan sesuai pers.(3.42) dan kodenya seperti pada Gambar 5.29.

* 1. Pemisahan Data

Pemisahan data dilakukan setelah normalisasi data sebagai salah satu bentuk persiapan data sebelum dimasukkan ke sistem RNN-UKF. Pada penelitian ini, data dipisahkan menjadi dua bagian utama, yaitu data latih dan data uji. Kode pemisahan data dapat dilihat pada Gambar 5.23. Data dipisah dengan rasio 6:4, 7:3 atau 8:2 untuk data latih dan data uji. Pada kode dibawah, diilustrasikan pembagian 8:2. Model yang dirancang kemudian akan menggunakan 70% dari keseluruhan data untuk *training* dan sisanya untuk diuji.

Gambar 5.23 Kode pemisahan data

1. def pisahData(data,a,b):
2. if((a+b)!=1):
3. print("pemisahan tidak valid")
4. else:
5. train = []
6. test = []
7. train\_size = int(len(data)\*a)
8. train = data[0:train\_size-1]
9. test = data[train\_size-1:len(data)]
10. return np.array(train),np.array(test)
11. train\_data, test\_data = pisahData(data\_raw, 0.8, 0.2)

Fungsi pembagian pisahData pada gambar diatas dapat dipanggil dengan menggunakan pisahData(data,a,b) dimana *input* *value-*nya diisi dengan data\_raw, 0.8, 0.2 dan disimpan ke train\_data dan test\_data.



Gambar 5.24 Variabel pemisahan data

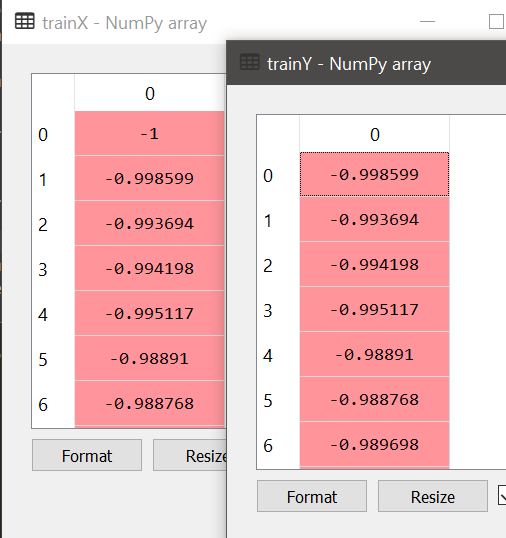
Pada Gambar 5.23 dapat dilihat bahwa kode pemisahan data diawali dengan membuat fungsi pisahData dengan parameter (data), persentase data train (a) dan test (b). Kemudian kode hanya berjalan jika data awal sama dengan 100% yang kemudian data latih disimpan dalam variabel train\_data dan data uji dalam test\_data. Setelah program dijalankan, didapatkan bahwa keseluruhan data berjumlah 975 data yang kemudian dipisah menjadi 779 data latih dan 196 data uji yang terlihat pada Gambar 5.24.

* 1. Menentukan I/O JST

Pada arsitektur NN penelitian ini, digunakan *many-to-one* arsitektur dimana *input* berupa banyak *nodes*. Sedangkan *output* berupa satu *output* saja untuk memprediksi data. Menentukan *input* dan *output* RNN dapat dianalisis menggunakan fungsi windowSize seperti pada Gambar 5.25. Awalnya, dibuat fungsi createDataset dengan variabel (data dan windowSize). Kemudian *value input* dimuat dalam *array* dataX dan target ke dalam dataY. Masing-masing *input* data latih dan uji dideklarasikan pada trainX dan trainY.

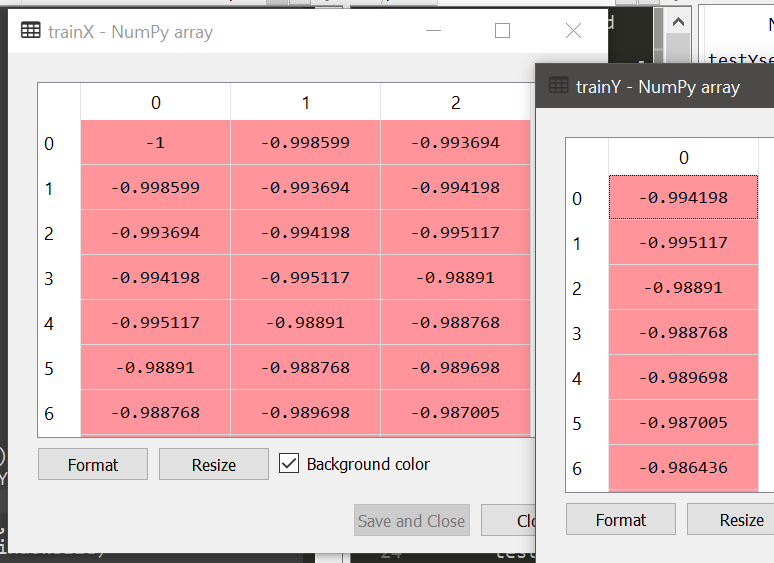
Gambar 5.25 Kode input data

1. def createDataset(data, windowSize):
2. dataX, dataY = [],[]
3. for i in *range*(len(data)-windowSize):
4. a = []
5. for j in *range*(i, i+windowSize):
6. a.append(data[j,0])
7. dataX.append(a)
8. dataY.append(data[i+windowSize,0])
9. return np.array(dataX), np.array(dataY)



Gambar 5.26 Tabel input data dengan windowSize=1

Jika windowSize=3, maka pada data ke 4 digunakan sebagai target. Sedangkan data ke-1, ke-2, dan ke-3 menjadi *input* pertama seperti pada Gambar 5.26 dan Gambar 5.27. Begitu pula seterusnya sampai data ke-*n*.



Gambar 5.27 Tabel input data dengan windowSize=3

* 1. Implementasi RNN

Arsitektur NN yang digunakan terbagi menjadi *input layer*, *hidden layer*, dan *output layer*. Pada penelitian ini, digunakan Elman *Recurrent Neural Network* dengan konsep *many-to-one*. Banyaknya *neuron* pada *input layer* ditentukan oleh besarnya *sliding window* yang dipilih. *Neuron* pada *hidden layer* menjadi bahan pengujian antara 1 unit sampai 10 unit. Sedangkan *output layer* berjumlah satu buah *neuron*. Variabel mse\_all digunakan untuk menyimpan nilai *loss* selama *training*.

1. def tanh(x):
2. return (1-np.exp(-2\*x))/(1+np.exp(-2\*x))
3. def dtanh(x):
4. return (1-tanh(x)\*\*2)

Gambar 5.28 Fungsi hyperbolic tangent dan turunannya

Fungsi aktivasi *hyperbolic tangent* digunakan untuk memfilter nilai yang masuk ke *neuron* setiap *layer* pada perhitungan *forward pass* dan *backward pass* dengan pers.(3.6) dan dengan kode seperti pada Gambar 5.28**Error! Reference source not found.**. Turunan *tanh* juga dihitung dengan variable *dtanh* seperti pada pers.(3.7).

1. def denormalize(normalized, data, scale):
2. denormalized = []
3. for i in *range*(len(normalized)):
4. a = ((normalized[i]-min(scale))\*(max(data)-min(data)))/(max(scale)-min(scale))+min(data)
5. denormalized.append(a)
6. return np.array(denormalized)
7. y\_pred = []
8. y\_pred = denormalize(np.reshape(y\_pred,(-1,1)), data['value'], (-1,1))
9. testY = denormalize(testY, data['value'], (-1,1))

Gambar 5.29 Kode fungsi denormalisasi

Pada akhir sistem, semua data dikembalikan nilainya seperti awal lagi, dimana data sebelumnya dinormalisasi ke dalam jangkauan [-1,1]. Maka data dikembalikan jangkauannya dengan pers.(3.42). Kode dari fungsi denormalisasi dijabarkan seperti pada Gambar 5.29.

Fungsi denormalisasi dengan tiga variabel yaitu normalized, data, scale dimana secara berurutan menunjukkan *value* yang didenormalisasii, data sebenarnya dan skala jangkauan. Kemudian dilakukan perhitungan dengan persamaan matematika dimana hasil denormalisasi untuk normalized disimpan dalam denormalized yang kemudian hasil dari denormalisasi untuk y\_pred disimpan dalam *array* y\_pred.

Gambar 5.30 Penentuan banyaknya *neuron* jaringan

1. input\_dim = windowSize
2. hidden\_dim = 6
3. output\_dim = 1

Salah satu unsur utama sistem *Neural Network* adalah *layer* dari sistem tersebut. *Layer* tersebut kemudian memiliki dimensi yang akan mentukan besaran, dan kompleksitas dari sebuah sistem NN. Pada sistem ini, dilakukan percobaan dengan menggunakan 2 sampai 10 *nodes* pada *hidden layer*. Besaran tersebut ditentukan dengan *input* variabel pada Gambar 5.30. input\_dim merupakan besarnya dimensi *input* yang bergantung pada besarnya *windowSize.* Sedangkan hidden\_dim dan output\_dim merupakan dimensi *hidden* *layer* dan *output* *layer* akhir yang dibutuhkan untuk prediksi hari berikutnya.

Gambar 5.31 Inisialisasi bobot jaringan

1. synapse\_0 = 2\*np.random.random((input\_dim,hidden\_dim)) - 1
2. synapse\_h = 2\*np.random.random((hidden\_dim,hidden\_dim)) - 1
3. synapse\_1 = 2\*np.random.random((hidden\_dim,output\_dim)) - 1

Bobot pada setiap *layer* digambarkan berturut-turut pada variabel synapse\_0, synapse\_h dan synapse\_1 untuk bobot jaringan *input* ke *hidden, hidden* ke *context*, dan *hidden* ke *output.* Pada Gambar 5.31, diinisialisasi bobot awal jaringan dengan interval [-1,1] secara *random*.

Gambar 5.32 Inisialisalisasi parameter dan jumlah bobot

1. jumlah\_w = (input\_dim\*hidden\_dim)+(hidden\_dim\*hidden\_dim)+
   * (hidden\_dim\*output\_dim)
2. Q = 0.01\*np.identity(jumlah\_w)
3. R = 1\*np.identity(output\_dim)
4. P = 1\*np.identity(jumlah\_w)

Gambar 5.32 menunjukkan inisialisasi sebelum *training* dengan jumlah\_w merupakan jumlah total perkalian atara satu *layer* dengan *layer* lainnya. Sedangkan Q, R dan P merupakan parameter wajib untuk *Unscented Kalman Filter* untuk memasuki tahap proses (*predict*) dan pengukuran (*measurement*). Dimana Q (kovarian *noise* dari *process*), R (kovarian *noise* dari *measurement*), dan P (*error* dari kovarian).

1. layer\_1 =

tanh(np.dot(X,synapse\_0)+np.dot(context\_layer,synapse\_h))

1. layer\_2 = tanh(np.dot(layer\_1,synapse\_1))
2. layer\_2\_value.append(layer\_2)

Gambar 5.33 Forward pass

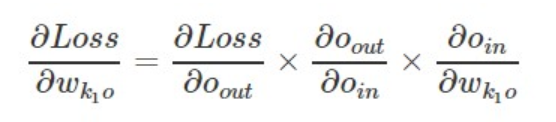
Untuk melakukan *forward pass,* dilakukan dengan perkalian matriks dan fungsi aktivasi *tanh* antara *layer* ke *layer* berikutnya seperti pada Gambar 5.33 diatas. layer\_1 untuk *forward pass* pada *hidden layer* dan layer\_2 menunjukkan *forward pass* untuk *output layer.* *Forward pass* dilakukan untuk membawa data “propagate” menuju *layer* selanjutnya yang dihitung dengan pers.(3.9)

Sebelum bobot di-*update*, harus dicari terlebih dahulu *error* yang didapatkan dari setiap *output layer* yang dilatih. Untuk mengitung *error*, dilakukan operasi pengurangan untuk mencari selisih *output* yang diharapkan dan output yang diprediksi dengan Gambar 5.34. Nilai pada setiap *layer* kemudian dihitung dengan fungsi aktivasi *dtanh* pada layer\_2 dan layer\_1 dengan persamaan *chain rule* pada Gambar 5.35.

Gambar 5.34 Menghitung *error* dan *backward pass*

1. layer\_2\_error = layer\_2 - Y[:,None]
2. layer\_2\_delta = layer\_2\_error\*dtanh(layer\_2)
3. layer\_1\_delta = (np.dot(layer\_h\_deltas,synapse\_h.T) +

np.dot(layer\_2\_delta,synapse\_1.T)) \* dtanh(layer\_1)



Gambar 5.35 Chain Rule

Dimana persamaan forward pass dan backward pass dapat ditemukan dalam pers.(3.8) sampai pers.(3.13). Kemudian bobot di*update* dengan Gambar 5.36 berdasarkan *backward pass* pada setiap bobot masing-masing *layer*.

Gambar 5.36 Update bobot

1. synapse\_1\_update =

np.dot(np.atleast\_2d(layer\_1).T,(layer\_2\_delta))

1. synapse\_h\_update =

np.dot(np.atleast\_2d(context\_layer).T,(layer\_1\_delta))

1. synapse\_0\_update = np.dot(X.T,(layer\_1\_delta)

*Update* variabel dilakukan seperti pada *pseudocode* pada Gambar 5.36 dengan menghitung besar *error* (selisih nilai prediksi yang diharapkan) setiap baris pada layer\_2 ke dalam variabel layer\_2\_*error*. Sedangkan layer\_2\_delta dan layer\_1\_delta secara berurutan merupakan perhitungan untuk *backward pass* balik dari *context layer* ke *hidden layer.* Dilakukan juga *update* matrikspada bobotjaringan dengan operasi numpy *dot product* dari matriks. Perhitungan *update* bobot tersebutdilakukan dengan synapse\_1\_update, synapse\_h\_update dan synapse\_0\_update.

Bobot kemudian digabung menjadi satu kolom *array* secara berurutan (kebawah) synapse\_0\_c, synapse\_h\_c, synapse\_1\_c pada Gambar 5.37 yang kemudian dicari bobot terbarunya dengan persamaan UKF. Seperti yang telah dijelaskan pada bab sebelumnya, *Unscented Kalman Filter* memerlukan perhitungan titik-titik sigma disekitar *mean*. Sigma tersebut memiliki parameter yang di-*set* menurut penelitian (Wan & Merwe, 2002) dan (J. Julier & Uhlmann, 1997) sebagai nilai terbaik seperti pada Gambar 5.37.

Gambar 5.37 Penggabungan bobot

1. synapse\_0\_c = np.reshape(synapse\_0,(-1,1))
2. synapse\_h\_c = np.reshape(synapse\_h,(-1,1))
3. synapse\_1\_c = np.reshape(synapse\_1,(-1,1))
4. w\_concat = np.concatenate((synapse\_0\_c,synapse\_h\_c,synapse\_1\_c), axis=0)

Kemudian besarnya dimensi titik-titik sigma diambil dari dimensi w\_concat dimana n = 2 untuk menentukan banyaknya titik sigma setiap satu *mean.* Kode pencarian titik-titik sigma itu sendiri dijabarkan pada Gambar 5.38.

Gambar 5.38 Penentuan parameter titik sigma

1. beta = 2.
2. kappa = 0
3. lambda\_ = 1
4. n = w\_concat.size

Kemudian bobot w\_concat ditransformasi menjadi titik sigma tengah (*mean* utama) dan ke-i dan seterusnya sampai *n* bobot yang ada di arsitektur RNN per *epoch.* Bobottersebut kemudian dicari titik-titik sigma disekitar *mean* utama dengan kode pada Gambar 5.39 dengan jumlah titik (2*n*+1). Dimana pada penelitian ini berarti terdapat 3 titik sigma yang salah satu diantaranya adalah *mean*. Kemudian titik sigma tersebut diprediksi dan didapatkan matriks baris dan kolom.

1. U = cholesky((n + lambda\_)\*P)
2. sigmas = np.zeros((2\*n+1, n))
3. sigmas[0] = X\_
4. for k in range(n):
5. sigmas[k+1] = np.subtract(X\_, -U[k])
6. sigmas[n+k+1] = np.subtract(X\_, U[k])

Gambar 5.39 Penghitungan titik-titik sigma disekeliling mean

Kemudian matriks titik-titik sigma tersebut dicari pembobotannya dengan pers.(3.30) yang dituliskan dengan kode pada Gambar 5.39 dengan menggunakan parameter yang telah diinisialisasi pada kode Gambar 5.38 dan dimensi dari w\_concat.

1. lambda\_ = alpha\*\*2 \*(n + kappa) - n
2. c\_ = .5 / (n + lambda\_)
3. Wm = np.full(2\*n+1, c\_)
4. Wc = Wm
5. Wc[0] = lambda\_ / (n + lambda\_) + (1 - alpha\*\*2 + beta)
6. Wm[0] = lambda\_ / (n + lambda\_)

Gambar 5.40 Pembobotan titik-titik sigma

Kemudian titik-titik sigma dan *mean* ditransformasi ke proses *measurement* dengan menghitung Mz seperti pada pers.(3.35) dan kovariannya Pz pada pers.(3.36). Keduanya dihitung dengan kode seperti pada Gambar 5.41.

Setelah ditransformasi, kemudian dihitung nilai *Kalman gain*-nya dengan pers.(3.38) pada Gambar 5.42. Satu persatu persamaan dijabarkan agar perhitungan *Kalman gain* tidak terjadi kesalahan. *Array* dari *Kalman gain* merupakan *array* satu kolom untuk dapat dioperasikan dengan nilai innovation pada Gambar 5.44.

Gambar 5.41 Mean dan kovarian di measurement

1. Mz = np.dot(Wm,sigmas)
2. kmax, n = sigmas.shape
3. Pz = np.zeros(n)
4. for k in range(kmax):
5. c = np.subtract(sigmas[k],Mz)
6. Pz = Wc[k] \* np.outer(c, c)
7. Pz += Q
8. Pz = Pz.sum()

Kemudian bobot di*update* dengan satu nilai, yaitu nilaiinnovation. *Value* dari innovation itu sendiri merupakan hasil dari jumlah selisih *error* yang dibagi dengan layer\_2\_error seperti kode pada Gambar 5.44.

1. Pxz = np.zeros(sigmas.shape)
2. for k in range(kmax):
3. cc = np.subtract(X\_,Mz)
4. Pxz = Wc[k] \* np.outer(cc, c)
5. Pxz = Pxz.sum()
6. K = np.dot(Pxz,inv(Pz))

Gambar 5.42 Kalman gain

Setelahnya, nilai bobot baru didapatkan dengan w\_concat\_new untuk *assign* kembali ke synapse\_0\_c, synapse\_h\_c, synapse\_1\_c.

Setelah didapatkan bobot baru, maka kovarian (P) di-*update* dengan pers.(3.40) dimana didapatkan *mean state* terbaru sesuai pers.(3.39) dengan kode pada Gambar 5.44.

Gambar 5.43 Update kovarian

1. P = P - np.dot(K,np.dot(Pz,K.T))

Gambar 5.44 Innovation dan update bobot UKF

1. innovation = ((Y-layer\_2).sum()/len(layer\_2\_error)) # hitung error pred dan yang diharapkan
2. synapse\_0 = w\_concat\_new[0:(input\_dim\*hidden\_dim),0]
3. synapse\_h = w\_concat\_new[(input\_dim\*hidden\_dim):(input\_dim\*hidden\_dim)+(hidden\_dim\*hidden\_dim),0]
4. synapse\_1 = w\_concat\_new[(input\_dim\*hidden\_dim)+(hidden\_dim\*hidden\_dim):w\_concat\_new.shape[0],0]

*Update* pada *context layer* kemudian dilakukan seperti pada Gambar 5.46 yang dilakukan terus menerus selama *epoch* pada sistem RNN. Nilai yang didapatkan kemudian akan digunakan untuk mendapatkan nilai prediksi *output* pada *output layer.*

1. layer\_h\_deltas = layer\_1\_delta
2. context\_layer = layer\_1

Gambar 5.45 Context layer baru

Pengujian dilakukan setelah dilakukan perhitungan sebanyak *epoch* yang ditentukan. Ketika *epoch* selessai, maka *training* pun selesai. Model dari hasil *training* digunakan untuk prediksi pada data uji seperti pada dimana hasil pengujian kemudian disimpan dalam variabel layer\_2p seperti pada Gambar 5.47.

* 1. Pengujian

Tahap akhir dalam penelitian adalah menguji sistem yang telah dirancang dan telah diimplementasikan datanya. Untuk menguji sistem rancangam, maka dihitung performa model dengan menghitung MAE, MSE, MAPE, RMAE dan Dstat.

1. context\_layer\_p = np.full((batch\_predict,hidden\_dim),0)
2. layer\_1p = tanh(np.dot(X,synapse\_0) +
   * np.dot(context\_layer\_p,synapse\_h))
3. layer\_2p = tanh(np.dot(layer\_1p,synapse\_1))

Gambar 5.46 Kode prediksi data uji

* + 1. MAE

*Mean* Absolute *Error* (MAE) merepresentasikan rata-rata kesalahan (*error*) absolut antara nilai prediksi dan sebenarnya. Persamaan MAE ditulis seperti pada Gambar 5.47Gambar 5.47 seperti pada pers.(3.43).

1. def mae(x,y):
2. mae = []
3. for i in *range*(len(y)):
4. a = abs(y[i]-x[i])
5. mae.append(a)
6. mae = float(sum(mae)/len(y))
7. return mae

Gambar 5.47 Kode MAE

* + 1. MSE

*Mean Squared* *Error* (MSE) merepresentasikan sebagai *error* dengan varian ditambah dengan kuadrat bias suatu model. Biasanya nilai MSE lebih besar dibandingkan dengan MAE karena dikuadratkan. Akibatnya nilai yang kecil semakin kecil dan yang besar semakin besar. Persaamaan MSE dituliskan seperti pada Gambar 5.49 dan pers.(3.44).

Gambar 5.48 Kode MSE

1. def mse(x,y):
2. mse = []
3. for i in *range*(len(y)):
4. a = (x[i]-y[i])\*\*2
5. mse.append(a)
6. mse = float((sum(mse)/len(y)))
7. return mse
   * 1. MAPE

Hampir sama dengan MAE, MAPE bertujuan untuk menunjukkan seberapa besar *error* yang didapat dalam persentase. Persamaan MAPE dituliskan dalam kode seperti pada Gambar 5.49 dan pers.(3.45).

1. def mape(x,y):
2. mape = []
3. for i in *range*(len(y)):
4. a = abs((x[i]-y[i])/x[i])
5. mape.append(a)
6. mape = float((sum(mse))/len(y))\*100
7. return mape

Gambar 5.49 Kode MAPE

* + 1. RMSE

RMSE dituliskan dalam kode seperti pada Gambar 5.50 untuk mendapatkan nilai akar dari MSE dan pers.(3.46).

1. def rmse(x,y):
2. rmse = []
3. for i in *range*(len(y)):
4. a = (x[i]-y[i])\*\*2
5. mse.append(a)
6. rmse = float((sum(rmse)/len(y))\*\*0.5)
7. return rmse

Gambar 5.50 Kode RMSE

* + 1. Dstat

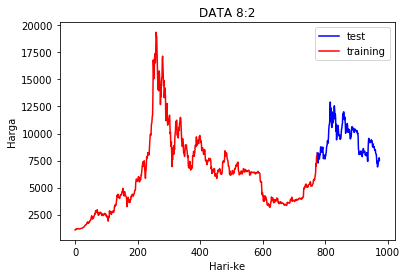
Kode Dstat dapat dituliskan seperti pada Gambar 5.51 dan pers. (3.47) dengan menghitung nilai persen dari *error*. Dstat membantu untuk melihat besarnya nilai akurasi karena berada dalam persen. Nilai Dstat digunakan untuk menentukan ukuran *sliding window* yang akan dilakukan pada pengujian. Nilai terbesar tersebut akan digunakan untuk *sliding window* tetap pada pengujian.

1. def dstat(x,y):
2. dstat = 0
3. n = len(y)
4. for i in *range*(n-1):
5. if(((x[i+1]-y[i])\*(y[i+1]-y[i]))>0):
6. dstat += 1
7. Dstat = (1/float(n-2))\*float(dstat)\*100
8. return float(Dstat)

Gambar 5.51 Kode Dstat

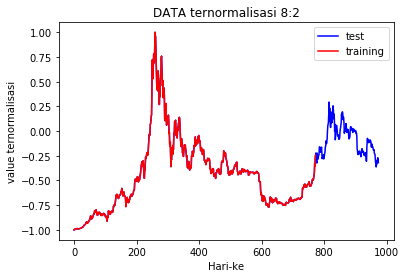
1. BAB VI  
   HASIL PENELITIAN DAN PEMBAHASAN

Pada bab ini, dipaparkan hasil pengujian implementasi NN-UKF dengan melihat hasil nilai hasil *loss* dan Dstat dalam persen untuk memudahkan analisis pelatihan. Arsitektur RNN yang digunakan adalah arsitektur *Elman* *Recurrent Neural Network* untuk memprediksi nilai *cryptocurrency* Bitcoin. Data *time series* yang dipakai adalah harga Bitcoin dengan nilai tukar USD dari 1 April 2017 sampai 1 Desember 2019 sebanyak 975 data.



Gambar 6.52 Grafik pemisahan data

Data diambil dari cryptodatadownload.com dan disimpan dalam file .csv. Kemudian data di *load* dengan dua kolom berbeda, yaitu *date* dan *value*. Kemudian kolom value dinormalisasi dengan persamaan 3.4 agar jangkauan data tidak terlalu jauh [-1,1]. Setelah dinormalisasi, data dipisah menjadi 2 bagian dengan rasio 8:2 antara data latih dan data uji. Grafik data untuk tiap bagiannya disajikan pada Gambar 6.52 untuk data sebelum dinormalisasi dan Gambar 6.53 yang telah dinormalisasi.



Gambar 6.53 Grafik pemisahan data ternormalisasi

* 1. Pengujian Nilai Q dan R

*Learning rate* merupakan laju pembelajaran, salah satu parameter yang berpengaruh pada arsitektur *Neural Network.* *Learning rate* biasanya di lambangkan dengan alpha (α) yang bernilai antara (0,1) menggambarkan kecepatan pembelajaran (*training*) dari sistem NN*.* Pada UKF, *learning rate* ditentukan oleh parameter R dan Q. *Learning rate* memiliki pengaruh terhadap kecepatan (waktu) pelatihan yang dibutuhkan untuk tercapainya target prediksi dengan *error* terkecil.

Jika nilai laju pembelajaran yang digunakan terlalu kecil maka *epoch* yang

dibutuhkan untuk mencapai nilai target yang diinginkan akan semakin banyak sehingga menyebabkan proses *training* membutuhkan waktu yang lama. Dengan nilai laju pembelajaran yang besar, maka proses *training* akan berjalan semakin cepat tetapi akan cenderung mengakibatkan jaringan yang tidak stabil dan menyebabkan nilai *error* berulang diantara nilai tertentu, sehingga mencegah *error* mencapai target yang diharapkan. Oleh karena itu pemilihan nilai variabel laju pembelajaran harus seoptimal mungkin agar didapatkan proses *training* yang cepat dengan jaringan yang stabil (Hermawan, 2006).

Nilai Q dan R masing-masing dicoba dengan 1, 0.1, 0.001, dan 0.001. Untuk mendapatkan nilai yang tetap, maka parameter lainnya di-*set* menjadi pembagian data rasio 8:3, *epoch* 1000, window size = 3, *hidden layer* = 7 *neuron,* R dengan *value* 1 dan sigma points dengan dan .

Gambar 6.54 Nilai Q

Dimana Q berubah, nilai R=1 agar tetap. Tetapi pada saat Q berubah, nilai Dstat berubah menjadi lebih baik yaitu secara berurutan menjadi 59%, 65%, 74%, dan 78% pada Gambar 6.54. Sedangkan untuk perubahan R, nilai Q di-*set* menjadi Q=1. Dapat dilihat pada gambar bahwa *loss* yang paling stabil adalah ketika Q = 1. Tetapi dalam persentase akurasi Dstat, akurasi pada data *training* lebih kecil dengan 59% disbanding yang lainnya.

Gambar 6.55 Nilai R

Pada pengujian R, dengan nilai berurutana 1, 0.1, 0.01, dan 0.001 seperti pada Gambar 6.55, didapatkan *loss* yang sangat tidak stabil. Tetapi nilai Dstat yang didapatkan juga adalah 74%, 71%, 73% dan 74%. Maka dari itu, dapat dianalisis bahwa nilai *loss* terbaik adalah saat Q = 0.001 dan R = 1 dengan Dstat bernilai 78%.

* 1. Pengujian Arsitektur JST

Pengujian arsitektur JST penelitian ini terletak pada banyaknya jumlah *nodes* (1-10 *nodes*) pada *hidden layer,* terlalu banyak *nodes* pada akan meningkatkan kemungkinan *overfitting* dan komputasi yang lebih lama walaupun mungkin akan menghasilkan akurasi yang baik. Maka dari itu, diharapkan pada proses pelatihan akan didapaktkan jumlah *nodes hideen layer* yanag tepat agar dapat meningkatkan hasil akurasi yang baik.

Jumlah *nodes* itu sendiri dicoba dengan parameter tetap seperti pada pengujian Q dan R kecuali nilai Q dan R itu sendiri yang akan ditetapkan menjadi Q = 1 dan R = 1*.* Kemudian arsitektur yang digunakan adalah arsitektur *Elman Recurrent Neural Network* dengan *context layer.*

* 1. Pengujian Ukuran *Sliding Window*

Pengujian dilakukan dengan *sliding window* (*window size*)satu sampai sepuluh *window.* Satu persatu digunakan dan dimuat dalam tabel untuk memudahkan analisis terhadap Dstat. Pengujian *window size* dan *hidden nodes* terhadap data *training* dapat dilihat dalam Tabel 6.4 menggunakan *epoch* (100) dan nilai *noise* ditetapkan menjadi Q=R=P dengan nilai 1.

Tabel 6.4 Hasil *training* *window size* dan *hidden nodes*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Window** | **Hidden** | **MAE** | **RMSE** | **Dstat (%)** |
| 1 | 3 | 0.068727026 | 0.084450054 | 57.86082474 |
| 1 | 5 | 0.991283427 | 1.081125566 | 53.09278351 |
| 1 | 7 | 0.636129526 | 0.70726959 | 52.57731959 |
| 1 | 9 | 0.170040283 | 0.198910993 | 50.25773196 |
| Average | |  |  | 53.44716495 |
|  | | | | |
| 2 | 3 | 0.672668086 | 0.741736908 | 54.58064516 |
| 2 | 5 | 0.630828004 | 0.692126957 | 48.64516129 |
| 2 | 7 | 1.097897536 | 1.185116883 | 52.77419355 |
| 2 | 9 | 1.163249887 | 1.244781261 | 54.96774194 |
| Average | |  |  | 52.74193548 |
|  | | | | |
| 3 | 3 | 0.403092596 | 0.434901074 | 47.93281654 |
| 3 | 5 | 0.166384039 | 0.218140616 | 57.49354005 |
| 3 | 7 | 0.15584539 | 0.205229047 | 55.42635659 |
| 3 | 9 | 1.219638058 | 1.307610504 | 52.84237726 |
| Average | |  |  | 53.42377261 |
|  | | | | |
| 4 | 3 | 1.022515868 | 1.106359889 | 53.29883571 |
| 4 | 5 | 0.208743002 | 0.237842284 | 53.29883571 |
| 4 | 7 | 0.822289548 | 0.878528569 | 49.41785252 |
| 4 | 9 | 0.636512643 | 0.687615498 | 52.39327296 |
| Average | |  |  | 52.10219922 |
|  | | | | |
| 5 | 3 | 0.362090536 | 0.434987817 | 54.40414508 |
| 5 | 5 | 0.130119082 | 0.162894834 | 54.40414508 |
| 5 | 7 | 0.059088863 | 0.081135364 | 62.56476684 |
| 5 | 9 | 1.195493153 | 1.272456211 | 51.94300518 |
| Average | |  |  | 55.82901554 |
|  | | | | |
| 6 | 3 | 0.526062457 | 0.563170585 | 48.11932555 |
| 6 | 5 | 0.412701858 | 0.51529648 | 56.67963684 |
| 6 | 7 | 1.327839428 | 1.399712284 | 56.16083009 |
| 6 | 9 | 0.271261334 | 0.36449538 | 57.9766537 |
| Average | |  |  | 54.73411154 |
|  | | | | |
| 7 | 3 | 0.51102752 | 0.569964667 | 48.18181818 |
| 7 | 5 | 0.522736734 | 0.571613879 | 52.20779221 |
| 7 | 7 | 0.378988663 | 0.420080146 | 53.8961039 |
| 7 | 9 | 0.234418342 | 0.26050265 | 54.28571429 |
| Average | |  |  | 52.14285714 |
|  | | | | |
| 8 | 3 | 0.527182333 | 0.58872538 | 53.3159948 |
| 8 | 5 | 1.002185032 | 1.079205461 | 51.23537061 |
| 8 | 7 | 0.539595777 | 0.60799265 | 53.05591678 |
| 8 | 9 | 0.107071932 | 0.134041636 | 58.51755527 |
| Average | |  |  | 54.03120936 |
|  | | | | |
| 9 | 3 | 0.155008235 | 0.188943205 | 53.125 |
| 9 | 5 | 0.416079961 | 0.478330428 | 51.43229167 |
| 9 | 7 | 1.23505256 | 1.29840343 | 51.69270833 |
| 9 | 9 | 0.156173684 | 0.199857462 | 53.125 |
| Average | |  |  | 52.34375 |
|  | | | | |
| 10 | 3 | 1.228949815 | 1.312639925 | 53.71577575 |
| 10 | 5 | 0.412128435 | 0.480693803 | 55.410691 |
| 10 | 7 | 0.2291132 | 0.258241377 | 52.67275098 |
| 10 | 9 | 1.272456126 | 1.343558415 | 51.36897001 |
| Average | |  |  | 53.29204694 |

Dapat dilihat pada hasil Tabel 6.4, bahwa nilai Dstat tertinggi pada data training didapatkana ketika *windowSize* 5 dan *hidden nodes* sebanyak 7 dengan Dstat sebesar 62%. Hal ini tidak menutup kemungkinan bahwa hasil akan meningkat pada pengujian menggunakan data uji yang sudah dibagi sebelumnya.

Sedangkan Dstat terendah didapatkan saat windowSize dan hidden nodes sebanyak 3 dan 3 sebesar hampir 48%. Maka dari itu, saat pengujian, akan digunakan *windowSize* sebesar 5 dan *hidden layer* dengan 7 *nodes*.

* 1. Perbandingan dengan RNN UKF menggunakan Gradient Descent

Pada akhir penelitian, hasil yang didapatkan pada RNN UKF dengan data uji menghasilkan nilai prediksi yang sudah besar, menandakan bahwa UKF dapat digunakan untuk *update* bobot pada jaringan *Neural Network.* Sebagai perbandingan bahwa rancangan arsitektur dan metode dapat dijalankan untuk memprediksi data *time series,* maka dilakukan percobaan lain menggunakan RNN-UKF-Gradient Descent.

*Gradient Descent* adalah salah satu metode pembelajaran yang paling populer untuk mendapatkaan *error* terkecil pada *Neural Network*. Menggunakan data training dan data uji, dilakukan percobaan dengan menggunakan kedua metode tersebut yang hasilnya dapat dilihat seperti pada Tabel 6.5.

Tabel 6.5 Perbandingan metode RNN-UKF-GD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluasi dengan** | **RNN-UKF** | | **RNN-UKF-GD** | |
| Training | Uji | Training | Uji |
| **MAE** | 0.05908885 | 404.7625356 | 1.032405892 | 366.4995843 |
| **RMSE** | 0.08113534 | 556.3166894 | 1.082601069 | 517.2792736 |
| **Dstat (%)** | 62.1761658 | 70.37037037 | 47.79792746 | 73.54497354 |
| runtime | 1.99614978 | 2.135160923 | 2.161162138 | 2.093158007 |

Pada Tabel 6.5, didapatkan bahwa nilai Dstat pengujian sangat besar dibandingkan dengan pelatihannya. Dstat pengujian didapatkan sebesar 70% saat pelatihan hanya sebesar 62% untuk RNN-UKF biasa. Sedangkana hal yang tak terduga didapatkan pada RNN-UKF-*Gradient Descent* dari pelatihan dengan nilai 47% menjadi 73% dengan pengujian.

Kedua metode dilakukan dengan 100 *epoch* menggunakan *windowSize* sebesar 5 dan *hidden layer* sebanyak 7 *nodes* karena jumlah tersebut menghassilkan Dstat terbaik pada pelatihhan yang dilakukan.

|  |  |
| --- | --- |
| **Loss RNN-UKF** | **Loss RNN-UKF-Gradient Descent** |
|  |  |
|  |  |

Gambar 6.56 Loss training RNN-UKF dibandingkan RNN-UKF-GD

Nilai *loss* yang didapatkan pada 100 dan 1000 epoch pada Gambar 6.56 menunjukkan RNN-UKF yang lebih stabil dibandingkan dengan RNN-UKF-*Gradient Descent*.

|  |  |
| --- | --- |
| **Nilai prediksi RNN-UKF** | **Nilai prediksi RNN-UKF-GD** |
|  |  |

Gambar 6.57 Prediksi RNN-UKF dibandingkan RNN-UKF-GD

Dapat dilihat pada Gambar 6.57, bahwa loss yang didapatkan pada RNN-UKF akhirnya lebih stabil dibandingkan dengan RNN-UKF-*Gradient Descent* dengan hanya menggunakan 100 *epoch.* Walaupun *loss* lebih stabil, nilai Dstat yang didapatkan pada RNN-UKF-GD lebih besar dengan 73% pada data uji.

1. BAB VII  
   KESIMPULAN
   1. Kesimpulan

Maka, setelah dilakukan penelitian, dapat disimpulkan bahwa:

1. *Recurrent Neural Network* - *Unscented Kalman Filter* terbukti dapat digunakan untuk *update* bobot pada RNN seperti penelitian-penelitian sebelumnya.
2. *Unscented* *Kalman Filter* tidak perlu mengitung matriks Jacobian seperti pada Kalman filter biasa. Oleh karenanya, UKF mengharuskan adanya pemillihan parameter yang masih tidak terlalu dimengerti jangkauan optimalnya pada penyelesaian masalah yang berbeda.
3. Data Bitcoin lebih fluktuatif dibandingkan dengan nilai tukar uang yang terbilang lebih stabil. Oleh karenanya, diperukan lebih banyak penelian tentang prediksi *cryptocurrencies* (terutama Bitcoin).
4. RNN-UKF dengan pelatihan *Gradient Descent* menghasilkan nilaia akurasi yang lebih baik dibandingkan RNN-UKF biasa dengan Dstat sebesatr 73%
   1. Saran

Adapun saran untuk penelitian selanjutnya dalam ruang lingkup yang sama, adalah:

1. Dapat dilakukan improvisasi dengan mengubah arsitektur *Neural Network* yang lebih baik seperti GRU atau LSTM dengan adanya “memori” dan “gerbang” untuk optimisasi.
2. Prediksi bobot juga dapat dilakukan dengan *Particle Filter* dimana *Particle Filter* merupakan pengembangan lanjutan dari Kalman Filter itu sendiri tetapi dengan komputasi yang lebih besar.
3. Kemudian jika ingin melakukan metode Kalman Filter, dapat diterapkan metode *smoothing* jika terdapat banyak noise dana adanya variabel tambahan yang mungkin mempengaruhi prediksi.

1. DAFTAR PUSTAKA

Albariqi, R., 2018. Prediksi Perubahan Harga Bitcoin Menggunakan Jaringan Syaraf Tiruan. *Skripsi.*

Amjad, M. J. & Shah, D., 2016. Trading Bitcoin and Online Time Series Prediction. *Proceedings of the Time Series Workshop at NIPS,* pp. 1-15.

Bisoi, R. & Dash, P., 2014. A hybrid evolutionary dynamic neural network for stock market trend analysis and prediction using unscented Kalman filter. *Applied Soft Computing Journal,* Volume 19, pp. 41-56.

Bisoi, R. & Dash, P. K., 2014. A hybrid evolutionary dynamic neural network for stock market trend analysis and prediction using unscented Kalman filter. *Applied Soft Computing Journal,* Volume 19, pp. 41-56.

Fausett, L. V., 1994. *Fundamentals of neural networks: Architectures, algorithms, and applications.* Englewood Cliffs(NJ): Prentice-Hall.

Gutierrez, G., Sesmero, M. P. & Sanchis, A., 2017. Forecasting time series by an ensemble of Artificial Neural Networks based on transforming the time series. *2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016 - Conference Proceedings,* pp. 4769-4774.

Haykin, S., 2001. *Kalman Filtering and Neural Networks.* New York, USA: John Wiley & Sons, Inc..

Haykin, S., 2004. *Kalman filtering and neural networks.* Canada: AWILEY-INTERSCIENCE PUBLICATION JOHN WILEY & SONS, INC..

Hazazi, M. A., 2018. Reccurent Neural Network dan Extended Kalman Filter untuk Peramalan Nilai Tukar Mata Uang. *Skripsi.*

Hermawan, A., 2006. *Jaringan Syaraf Tiruan: Teori dan Aplikasi.* Yogyakarta: Andi.

J. Julier, S. & Uhlmann, J. K., 1997. New extension of the Kalman filter to nonlinear systems. *Signal Processing, Sensor Fusion, and Target Recognition VI,* Volume 3068, p. 182.

Kalman, R., 1960. A New Approach to Linear Filtering and Prediction Problems. *ASME Journal of Basic Engineering,* Issue 82, pp. 35-45.

Kenawas, M. R. & Sulistiowati, 2016. Legalitas Bitcoin sebagai mata uang dan metode pembayaran di Indonesia. *Skripsi.*

Kim, Y. B. et al., 2016. Predicting Virtual World User Population Fluctuations with Deep Learning. *PLoS ONE,* 11(12), pp. 1-12.

Kristoufek, L., 2013. *BitCoin meets Google Trends and Wikipedia: Quantifying the relationship between phenomena of the Internet era,* s.l.: s.n.

Mangan, D. M., 2013. Bitcoin: Cara Kerja dan Perbandingannya dengan Mata Uang Konvensional. *Makalah Kriptografi,* pp. 1-5.

Martina, M., Lunesu, I. & Marchesi, M., 2015. Bitcoin Spread Prediction Using Social And Web Search Media. *UMAP Workshops.*

McNally, S., 2016. Predicting the price of Bitcoin using Machine Learning. *Doctoral dissertation, Dublin, National College of Ireland.*

Nakamoto, S., 2009. Bitcoin: A Peer-to-Peer Electronic Cash System.

Pangayom, G. R., 2017. *Model Peramalan Long Short-Term Memory Recurrent Neural Network untuk Peramalan Nilai Tukar Mata Uang Dolar Amerika-Rupiah Indonesia.* Yogyakarta: Skripsi UGM.

Roger, R. & Labbe, J., 2018. Unscented Kalman Filter. In: *Kalman and Bayesian Filters in Python.* MIT Licence: Github, pp. 531-387.

Saptoro, A., 2012. Extended and unscented kalman filters for artificial neural network modelling of a nonlinear dynamical system. *Theoretical Foundations of Chemical Engineering,* 46(3), pp. 274-278.

Siang, J. J., 2009. *Jaringan Syaraf Tiruan dan Pemrogramannya Menggunakan MATLAB.* Yogyakarta: Andi.

Torres, D. G. & Qiu, H., 2018. Applying Recurrent Neural Networks for Multivariate Time Series Forecasting of Volatile Financial Data. pp. 1-10.

Wan, E. A. & Merwe, R. v. d., 2002. The unscented Kalman filter for nonlinear estimation. *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000,* Issue 3, pp. 153-158.

Zhan, R. & Wan, J., 2006. Neural network-aided adaptive unscented Kalman filter for nonlinear state estimation. *IEEE Signal Processing Letters,* 13(7), pp. 445-448.

1. LAMPIRAN A  
   SOURCE CODE

Perhitungan Data

1. S
2. Vsv
3. sv