PERAMALAN PERSEDIAAN *SPARE PART* SEPEDA MOTOR MENGGUNAKAN ALGORITME *BACKPROPAGATION*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Danastri Ramya Mehaninda
NIM: 145150200111035

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PERAMALAN PERSEDIAAN *SPARE PART* SEPEDA MOTOR
MENGGUNAKAN ALGORITME *BACKPROPAGATION*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Danastri Ramya Mehaninda
NIM: 145150200111035

Skripsi ini telah diuji dan dinyatakan lulus pada
23 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Imam Cholissodin, S.Si, M.Kom	Ir. Sutrisno, M.T
NIK: 201201 850719 1 001	NIP: 19570325 198701 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T., Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 22 Juni 2018

Danastri Ramya Mehaninda

NIM: 145150200111035

KATA PENGANTAR

Alhamdulillah, puji syukur penulis panjatkan kepada Allah SWT karena atas segala rahmat dan karuniaNya penulis dapat menyelesaikan laporan skripsi berjudul "Peramalan Persediaan *Spare Part* Menggunakan Algoritme *Backpropagation*". Dalam pengerjaan skripsi dan penyusunan laporan skripsi banyak pihak yang membantu, membimbing, serta memberikan dukungan baik secara moril maupun materiil. Oleh karena itu, penulis mengucapkan terimakasih yang sebesar-besarnya kepada:

1. Mama Lilis Widayati dan papa Handaru Subekti selaku orang tua penulis.
2. Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang; Tri Astoto Kurniawan, S.T, M.T., Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang dan Agus Wahyu Widodo, S.T, M.Cs selaku Ketua Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
3. Imam Cholissodin, S.Si, M.Kom selaku dosen pembimbing 1 skripsi dan Ir. Sutrisno, M.T selaku dosen pembimbing 2 skripsi.
4. Luckman Warwandono selaku narasumber.
5. Kakak Prasidya Radintantya, eyang Marpah, eyang Sugijanto, eyang Soehartati, alm. eyang Soedjiamto dan seluruh keluarga.
6. Rayhan Tsani Putra selaku *partner* penulis dalam banyak hal.
7. Cindy, Enggar, Katon, Faykel, Fendy, Nisya, Sarah, Eka, Nisa, Adit, Danang dan Elha selaku teman dekat penulis.
8. Teman-teman dari Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya dan lainnya.
9. Seluruh pihak yang telah membantu dalam pengerjaan skripsi ini yang tidak dapat disebutkan satu persatu.

Penulis menyadari bahwa dalam penyusunan laporan skripsi ini masih banyak kekurangan, baik dari segi susunan maupun isi. Oleh karena itu, penulis dengan senang hati dan terbuka menerima kritik dan saran yang membangun sebagai pedoman perbaikan penulis.

Malang, 22 Juni 2018

Penulis

drmehaninda@gmail.com

ABSTRAK

Sepeda motor adalah transportasi darat yang paling banyak digunakan karena harganya lebih terjangkau dan lebih efisien. Sepeda motor memerlukan perawatan yang baik agar penggunaannya tetap nyaman dan menjaga kinerja sepeda motor sehingga dapat juga meminimalisir kecelakaan. Perawatan sepeda motor dapat dilakukan dengan cara penggantian *spare part* atau suku cadang secara berkala di bengkel. Guna menunjang perawatan sepeda motor, sebaiknya bengkel memberikan pelayanan perawatan yang terbaik termasuk memiliki persediaan *spare part* yang cukup agar dapat memenuhi kebutuhan pelanggan dalam perawatan sepeda motor. Jika bengkel memiliki persediaan *spare part* yang cukup, maka bengkel dapat meminimalisir biaya pemesanan dan dapat meminimalisir terjadinya kerusakan akibat penyimpanan terlalu lama sehingga perputaran *spare part* dinilai baik karena stok tidak terlalu banyak ataupun terlalu sedikit dan seluruh transaksi dapat terpenuhi. Terdapat banyak bengkel yang menyediakan pelayanan penggantian *spare part* seperti pada Pangestu Utomo Motordi Candi, Sidoarjo. Pada Pangestu Utomo Motortersebut mengalami kesulitan dalam menentukan persediaan *spare part* untuk bulan berikutnya. Peramalan persediaan dapat membantu untuk menentukan persediaan *spare part* pada Pangestu Utomo Motor. Penelitian ini menggunakan algoritme *backpropagation* untuk peramalan persediaan *spare part*. Arsitektur *backpropagation* yang terbaik adalah 9-7-1, yaitu 9 node *input*, 7 node *hidden* dan 1 node *output*. Input yang digunakan adalah *history* penjualan *spare part* bulan sebelumnya. Rata-rata *MSE* (nilai *error*) yang didapatkan dari hasil pengujian adalah 0.0094506 dan *MSE* terkecil yang didapatkan adalah 0.0085305 dengan rata-rata selisih nilai aktual dengan hasil peramalan adalah 6. Pada nilai *MSE* terkecil tersebut, hasil peramalan mendekati nilai aktualnya dan memiliki pola yang hampir sama.

Kata kunci: peramalan, prediksi, persediaan, *spare part*, sepeda motor, *backpropagation.*

ABSTRACT

Motorcycle are the most used roudways transportation because they are more affordable and more efficient. Motorcycle require good maintenance to keep comfortable uses and maintain motorcycle performance so as to minimize accidents. Motorcycle maintenance can be done by replacing spare parts regularly in the workshop. To support the maintenance of motorcycle, the workshop should provide the best care services including having spare part inventory to suffice customer who maintance of motorcycle. If the workshop has sufficient spare part, the workshop can minimize the cost of ordering and can minimize the damage caused by storage for too long so that spare part rotation is considered good because the stock is not too much or too little and all transactions can be fulfilled. There are many workshops that provide spare part replacement service such as Yamaha Motor in Candi, Sidoarjo. At Pangestu Utomo Motor is having difficulty in determining the spare part inventory for the next month. Inventory forecasting can help to determine the supply of spare part on Pangestu Utomo Motor. This research uses backpropagation algorithm for forecasting spare part inventory. The best backpropagation architecture is 9-7-1, which mean 9 input nodes, 7 hidden nodes and 1 output node. The input used is the history of spare part sales the previous month. The average MSE (error value) obtained from the test result is 0.0094506 and the smallest MSE obtained is 0.0085305 with the average difference of the actual value with the forecasting result is 6. At the smallest MSE value, the forecasting result approaches the actual value and has a pattern that almost the same.

***Keywords**: forecasting, prediction, inventory, spare part, motorcycle, backpropagation.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Transportasi digunakan untuk mempermudah kegiatan atau aktivitas sehari-hari. Transportasi adalah usaha dan kegiatan mengangkut atau membawa penumpang dan/atau barang dari suatu tempat ke tempat lainnya (Kadir, 2006). Transportasi memiliki beberapa unsur-unsur penting, salah satunya adalah kendaraan sebagai alat angkut atau alat transportasi. Berdasarkan unsur tersebut, transportasi diklasifikasikan menjadi transportasi udara, transportasi air dan trasportasi darat (Kadir, 2006). Salah satu jenis kendaraan pada transportasi darat adalah sepeda motor. Sepeda motor adalah kendaraan yang paling banyak disukai karena merupakan kendaraan roda dua yang menggunakan mesin penggerak sehingga lebih efisien dan harganya lebih terjangkau dibandingkan kendaraan roda empat (mobil) (Sudjarwo, 2013). *Performance* atau kinerja sepeda motor akan menurun dan terjadi terjadi kerusakan, meskipun cepat atau lambatnya hal tersebut terjadi bergantung pada pemeliharaan atau perawatan sepeda motor. Sepeda motor memerlukan perawatan yang baik agar penggunaannya tetap nyaman dan menjaga kinerja sepeda motor sehingga dapat juga meminimalisir kecelakaan. Perawatan sepeda motor dapat dilakukan dengan cara penggantian *spare part* atau suku cadang secara berkala di bengkel. Oleh karena itu, setiap bengkel sebaiknya mempunyai persediaan *spare part* yang cukup untuk memenuhi kebutuhan konsumennya guna menunjang perawatan sepeda motor dengan baik. Seperti pemaparan Ida Farida dan Moh. Nafis Rozini (2016) pada penelitiannya bahwa perusahaan memiliki risiko jika tidak dapat memenuhi keinginan para pelanggan, maka perusahaan harus memiliki konsep pengendalian persediaan yang baik agar mengurangi risiko tersebut (Farida & Rozini, 2016).

Masalah persediaan menjadi permasalahan berbagai perusahaan termasuk perusahaan jasa seperti bengkel motor (Dinata & Wigati, 2016). Hal tersebut terjadi karena persediaan dilakukan tanpa memperhitungkan perencanaan, sehingga dapat mempengaruhi biaya total persediaan. Jika bengkel memiliki persediaan *spare part* yang kurang untuk konsumennya maka dapat menghambat perawatan sepeda motor dan tidak dapat memenuhi keinginan permintaan konsumen. Apabila permintaan konsumen tidak terpenuhi maka menyebabkan ketidakpuasan pada konsumen dan menyebabkan konsumen akan beralih ke perusahaan lain (Farida & Rozini, 2016). Jika persediaan *spare part* terlalu banyak, juga dapat menyebabkan masalah untuk bengkel terutama ketika pembelian *spare part* tersebut sedikit. Hal itu menyebabkan penuhnya tempat penyimpanan dan kerusakan *spare part* atau perubahan kualitas pada *spare part*. Perusahaan sebisa mungkin harus memberikan pelayanan yang baik untuk konsumen. Selalu menyediakan suku cadang (*spare part*) dengan kualitas yang baik merupakan salah satu cara bengkel memberikan pelayanan yang baik (Farida & Rozini, 2016). Jika persediaan *spare part* dapat dikendalikan dalam artian perusahaan selalu mempunyai persediaan yang cukup, maka perusahaan dapat meminimalisir biaya pemesanan dengan melakukan pemesanan hanya ketika *reorder point* dan dapat

menghindari biaya penyimpanan serta menghindari kerusakan akibat persediaan *spare part* yang ada di gudang berlebih, perputaran *spare part* dinilai baik karena stok tidak terlalu banyak ataupun terlalu sedikit dan hampir seluruh transaksi dapat terpenuhi (Farida & Rozini, 2016).

Terdapat banyak bengkel yang menyediakan pelayanan penggantian *spare part* seperti pada salah satu *dealer* Yamaha Motor di Candi, Sidoarjo. Pangestu Utomo Motormerupakan salah satu *dealer* Yamaha yang memperjual belikan sepeda motor dan memberikan pelayanan perawatan sepeda motor yang meliputi penggantian *spare part* serta penjualan *spare part* sepeda motor Yamaha. Tidak semua pemilik kendaraan memiliki kesadaran akan pentingnya perawatan dan penggantian *spare part* secara berkala. Hal itu menyebabkan banyaknya pembelian sepeda motor tidak menjamin akan banyaknya penjualan *spare part* pada Pangestu Utomo Motor. Pangestu Utomo Motor harus selalu mempunyai persediaan *spare part* yang cukup untuk memenuhi kebutuhan konsumen dan tidak terlalu berlebihan agar tidak membuat gudang penyimpanan penuh serta menimbulkan kerusakan atau perubahan kualitas. Standarnya, Pangestu Utomo Motor membeli *spare* part pada distributor setiap 1 bulan sekali untuk menghemat biaya operasional pembelian dan mempermudah pencatatan pada pembukuan. Selama ini Pangestu Utomo Motor membeli *spare part* untuk persediaan berdasarkan perkiraan. Hal tersebut sering menimbulkan masalah, seperti terjadinya kekurangan persediaan sehingga tidak dapat memberikan pelayanan kepada konsumen dan harus segera membeli *spare part* sebelum waktunya. Selain itu, persediaan *spare part* terlalu banyak dapat menyebabkan penuhnya gudang penyimpanan dan jika *spare part* tersebut disimpan terlalu lama juga dapat menyebabkan kerusakan dan perubahan kualitas. Berdasarkan permasalahan tersebut, maka dibutuhkan peramalan untuk memperkirakan jumlah *spare part* yang dibutuhkan.

Penelitian tentang metode peramalan telah banyak dilakukan. Salah satunya penelitian mengenai peramalan permintaan produksi yang dilakukan oleh Febrina dan kawan-kawan (2013) yang berhasil menerapkan metode *backpropagation* untuk peramalan dengan 3 *neuron* pada lapisan *input* yang merupakan factor yang mempengaruhi permintaan, yaitu hasil penjualan, harga penjualan dan stok. Nilai *error* yang dihitung dengan *Mean Absolute Percentage Error* (MAPE) pada penelitian tersebut terbilang kecil, yaitu sebesar 5,7134%. Penelitian peramalan menggunakan *backpropagation* juga dilakukan oleh Razak dan Riksakomara (2017) untuk meramalkan jumlah produksi ikan nilai MAPE kurang dari 5%. Penelitian tersebut juga memaparkan kelebihan *backpropagation* yaitu memiliki sifat *adaptive* (dapat menyesuaikan terhadap dataset) dan *fault tolerance* (kesalahan error kecil) terhadap pemecahan masalah pada sistem (Razak & Riksakomara, 2017). Oleh karena itu, *backpropagation* dipilih untuk digunakan pada kasus yang diusulkan ini. Pada penelitian ini penulis akan menggunakan *backpropagation* untuk peramalan persediaan *spare part* sepeda motor pada Pangestu Utomo Motor.

1.2 Rumusan masalah

Pemaparan latar belakang mendasari perumusan masalah pada penulisan, rumusan masalah yang didapatkan adalah sebagai berikut:

1. Bagaimana penerapan algoritme *backpropagation* untuk peramalan persediaan *spare part* sepeda motor?
2. Bagaimana nilai *error* pada algoritme *backpropagation* untuk peramalan persediaan *spare part* sepeda motor?

1.3 Tujuan

Dari rumusan masalah yang telah dipaparkan, penelitian ini memiliki tujuan sebagai berikut:

1. Menerapkan algoritme *backpropagation* untuk peramalan persediaan *spare part* sepeda motor.
2. Menghitung dan mengetahui nilai *error* pada algoritme *backpropagation* untuk peramalan persediaan *spare part* sepeda motor.

1.4 Manfaat

Penelitian ini merupakan implementasi komputasi cerdas dalam kehidupan nyata. Penelitian ini diharapkan dapat bermanfaat bagi Pangestu Utomo Motoruntuk melakukan peramalan persediaan *spare part*. Penelitian ini juga diharapkan dapat bermanfaat bagi yang membutuhkan peramalan persediaan dan dapat dikembangkan menjadi lebih baik lagi dalam penelitian selanjutnya.

1.5 Batasan masalah

Batasan masalah pada penelitian ini bertujuan agar penelitian berfokus dan tidak meluas. Penelitian ini memiliki batasan masalah sebagai berikut:

1. Penelitian ini menggunakan data pada salah satu *dealer* Yamaha Motor, yaitu Pangestu Utomo Motor.
2. *Spare Part* yang akan diramalkan adalah oli *yamalube matic*.
3. *Input* yang akan digunakan adalah data *history* penjualan *spare part* beberapa bulan sebelumnya. *Output* yang didapat adalah jumlah *spare part* pada bulan berikutnya.
4. *Input* atau *fitur* (jumlah node *input layer*) dan jumlah node *hidden layer* pada penelitian ini dapat ditentukan, sedangkan *output* (jumlah node *output layer*) hanya satu.

1.6 Sistematika pembahasan

Penyusunan penelitian ini menggunakan kerangka pembahasan yang tersusun sebagai berikut:

BAB 1 PENDAHULUAN

Pendahuluan merupakan bab yang berisi latar belakang, rumusan masalah, tujuan, batasan masalah serta manfaat peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakaan digunakan untuk mendeskripsikan dan memaparkan dasar teori yang akan digunakan serta yang mendukung penelitian peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*. Landasan kepustakaan ini terdiri dari kajian pustaka, peramalan, persediaan, *spare part*, jaringan syaraf tiruan dan algoritme *backpropagation*.

BAB 3 METODOLOGI

Metodologi merupakan bab metodologi penelitian yang menguraikan metode dan langkah kerja yang dilakukan untuk peramalan persediaan *spare part*. Metodologi penelitian pada penelitian ini terdiri dari studi pustaka, pengumpulan data, analisis kebutuhan, perancangan, implementasi, pengujian, kesimpulan dan saran.

BAB 4 PERANCANGAN

Menganalisis kebutuhan sistem untuk peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.

BAB 5 IMPLEMENTASI

Mengimplementasikan dan membahas peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.

BAB 6 PENGUJIAN DAN ANALISIS

Pengujian dan analisis mendeskripsikan rencana pengujian serta menganalisis hasil pengujian pada peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* yang telah diimplementasikan.

BAB 7 PENUTUP

Bab ini memuat kesimpulan berdasarkan hasil penelitian peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* yang telah dilakukan dan memberikan saran yang dapat dipergunakan untuk penelitian berikutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Persediaan barang yang tidak tepat menyebabkan kerugian, kerugian tersebut bisa dari segi waktu dan biaya, maka untuk meminimalisir kerugian perlu dilakukan prediksi atau peramalan dengan sebuah sistem peramalan (Sundari, et al., 2015). Hal tersebut yang mendorong Sundari dan kawan-kawan (2015) untuk melakukan peramalan persediaan barang pada toko *The Kids* 24. Toko *The Kids* 24 merupakan toko yang menjual boneka, mainan, pakaian dan berbagai perlengkapan anak lainnya. Dari tahun ke tahun toko *The Kids* 24 mengalami perkembangan penjualan dan memiliki persaingan usaha yang cukup ketat. Toko *The Kids* 24 harus memiliki persediaan barang yang cukup agar dapat memenuhi kebutuhan serta keinginan konsumennya. *History* penjualan berpengaruh pada prediksi jumlah barang yang perlu dibeli untuk persediaan toko *The Kids* 24 (Sundari, et al., 2015). Metode yang digunakan pada penelitian tersebut adalah *Weighted Moving Average*. Hasil penelitian tersebut peramalan atau prediksi persediaan barang mempermudah toko untuk pembelian persediaan barang pada toko setiap bulannya sehingga membantu proses pelayanan toko.

Pengendalian persediaan perlu dilakukan agar dapat memiliki persediaan cukup dan pengendalian dapat dilakukan dengan meramalkan persediaan. Penelitian mengenai peramalan telah banyak dilakukan. Seperti penelitian yang dilakukan oleh Razak dan Riksakomara (2017) mengenai peramalan jumlah produksi ikan dengan menggunakan *backpropagation* pada Unit Pelaksana Teknisi Daerah (UPTD) Pelabuhan perikanan Banjarmasin. Pada penelitian tersebut permasalahan utama yang ada adalah setiap bulannya jumlah produksi ikan tidak menentu. Hal tersebut berdampak pada kegiatan yang berkaitan dengan proses produksi seperti penyaluran es pendingin, penyediaan air bersih dan kegiatan lainnya. Penulis memberikan solusi atas permasalahan tersebut, yaitu melakukan peramalan jumlah produksi ikan beberapa periode berikutnya berdasarkan data-data *history* produksi sebelumnya. Penulis menggunakan metode metode *backpropagation neural network* (BPNN). Aristektur Metode BPNN yang digunakan adalah dengan masukan satu dan dua. Masukan tersebut merupakan jumlah produksi ikan UPTD Pelabuhan Perikanan Banjarmasin pada periode sebelum sekarang. Hasil *error* yang dihasilkan peramalan berkisar 20% pada proses testing dan juga peramalan dengan menggunakan seluruh data *history* yang ada (Razak & Riksakomara, 2017).

Penelitian tentang peramalan juga dilakukan oleh Febrina dan kawan-kawan (2013) untuk meramalkan jumlah permintaan produksi menggunakan metode *backpropagation*. Pada penelitian tersebut menggunakan metode Jaringan Syaraf Tiruan (JST) *backpropagation* untuk meramalkan jumlah permintaan produk v-belt AJGG B-65. Fitur yang digunakan merupakan faktor terkait yaitu hasil penjualan, harga dan stok barang jadi. Arsitektur jaringan syaraf tiruan yang dapat digunakan untuk peramalan permintaan v-belt AJGG B-65 di PT.XYZ

adalah jaringan *multilayer feedforward* dengan struktur *neuron* 20-1 dengan 1 (satu) hidden layer, *learning rate* (lr) yang digunakan 0,1 dan momentum *constant* (mc) 0,2 (Febrina, et al., 2013). Nilai *Mean Square Error* (*MSE*) pelatihan jaringan sebesar 0,001 dan nilai MAPE pengujian data sebesar 5,7134% (Febrina, et al., 2013).

Ismail dan Jamaluddin (2008) juga melakukan penelitian menggunakan *backpropagation* untuk melakukan peramalan permintaan beban listrik yang mana permintaan dipengaruhi oleh beberapa variabel seperti cuaca, suhu, hari libur, hari dalam seminggu. Penelitian tersebut memperkenalkan sinyal error *backpropagation* yang lebih baik dengan menggunakan dua fungsi aktivasi, *sigmoid* dan *arctangent*, dalam meramalkan beban listrik maksimum setiap hari. Hasil penelitian menunjukkan bahwa tingkat konvergensi perbaikan *error* dengan fungsi *sigmoid* jauh lebih cepat dibandingkan dengan *error* yang diperbaiki dengan fungsi *arctangent*, sedangkan fungsi *arctangent* secara teoritis menghasilkan kecepatan konvergensi yang lebih cepat karena memiliki nilai derivatif yang lebih tinggi dibandingkan dengan fungsi *sigmoid* (Ismail & Jamaluddin, 2008).

Julpan dan kawan-kawan (2015) melakukan penelitian menganalisis fungsi aktivasi *sigmoid biner* dan *bipolar* pada *backpropagation* untuk memprediksi kemampuan siswa. Pada penelitian ini digunakan suatu model *neural network* (jaringan syaraf tiruan) untuk memprediksi kemampuan siswa, selanjutnya hasil yang diperoleh ditransformasikan dengan suatu pendekatan algoritme *backpropagation*. Hasil penelitian prediksi kemampuan siswa berdasarkan nilai raport dengan ini menunjukkan fungsi aktivasi *sigmoid biner* memiliki tingkat akurasi yang paling tinggi, sedangkan fungsi aktivasi *sigmoid bipolar* memiliki tingkat kecepatan (konvergen) lebih tinggi (Julpan, et al., 2015).

Peramalan menggunakan algoritme *backpropagation* dilakukan oleh Abdellah dan Djamel (2013) untuk meramalkan profil puncak beban Aljazair. Peramalan ini menggunakan model Time-Series, yaitu menggunakan data beban historis untuk ekstrapolasi untuk memprediksi beban di masa mendatang. Elaborasi perkiraan beban Aljazair menjadi semakin sulit karena ketidakpastian yang terkait dengan faktor-faktor yang digunakan dalam persiapannya, terutama yang terkait dengan kebiasaan konsumsi yang berubah (Abdellah & Djamel, 2013). Dataset yang digunakan pada penelitian ini menggunakan data selama 23 bulan. *Input* yang digunakan berupa matriks 664x5, *output* juga berupa matriks 664x1 dan *neuron hidden* 20. Nilai *error* yang dihasilkan dari penelitian tersebut rata-rata mendekati 0.

Tabel 2.1 Perbedaan Penelitian

Judul	**Objek**	**Metode**	**Keluaran**
	Masukan dan Parameter	**Proses**	**Hasil Penelitian**
Sistem Peramalan Persediaan Barang	1. Data *history* penjualan	*Weight Moving Average*	Jumlah barang untuk bulan berikutnya

Dengan *Weight Moving Average* Di Toko The Kids 24	barang 3 bulan sebelumnya 2. Data *history* penjualan 5 bulan sebelumnya		Rata-rata nilai *error* (*Mean Absolute Deviation* (MAD), *Mean Squared Error* (*MSE*), *Mean Absolute Persentage Error* (MAPE)), yaitu: 1. - *MAD* : 8,56 - *MSE* : 115,4 - *MAPE* : 25% 2. - *MAD* : 9,28 - *MSE* :129,8 - *MAPE* : 24%
Peramalan Jumlah Produksi Ikan dengan Menggunakan Backpropagation Neural Network (Studi Kasus UPTD Pelabuhan Perikanan Banjarmasin)	Data *History* jumlah ikan satu dan dua periode sebelum sekarang	*Backpropagation*	Jumlah produksi ikan Nilai *error* berkisar 20% pada proses testing dang peramalan
Peramalan Jumlah Permintaan Produksi Menggunakan Metode Jaringan Syaraf Tiruan (JST) Backpropagation	1. Harga dan stok barang jadi 2. Data *history* hasil penjualan	*Backpropagation*	Jumlah permintaan *v-belt* Arsitektur JST terbaik adalah 1 (satu) hidden layer, learning rate (lr) yang digunakan 0,1 dan momentum *constant* (mc) 0,2 dengan fungsi aktivasi *logsig* dan *purelin* Nilai Mean Square Error (MSE) pelatihan jaringan sebesar 0,001 dan nilai MAPE pengujian data sebesar 5,7134%
A Backpropagation Method for Forecasting	1. permintaan beban puncak	*Backpropagation*	Nilai *Mean Absolute Persentage Error* (MAPE) berkisar 5-8%

Electricity Load Demand	2. suhu rata-rata 3. suhu maksimum 4. suhu minimum 5. beban hari sebelumnya 6. beban 7 hari sebelumnya (hari yang sama minggu sebelumnya) dan 6 hari sebelumnya		Fungsi aktivasi sigmoid dan *arctangent* menunjukkan bahwa tidak banyak perbedaan dalam hal akurasi perkiraan
Analisis Fungsi Aktivasi *Sigmoid Biner* dan *Sigmoid Bipolar* dalam Algoritma *Backpropagation* pada Prediksi Kemampuan Siswa	Data nilai raport 6 mata pelajaran (6 *input*)	*Backpropagation*	*Output*nya adalah nilai UN
			Fungsi aktivasi *sigmoid biner* memiliki tingkat akurasi prediksi yang paling tinggi
			Fungsi aktivasi *sigmoid bipolar* memiliki tingkat kecepatan (konvergen) yang tinggi
Forecasting the Algerian Load Peak Profile Using Time Series Model Based On Backpropagation Neural Networks	Data beban historis	*Backpropagation*	Rata-rata nilai *error* mendekati 0
Peramalan Persediaan *Spare Part* Menggunakan Algoritme *Backpropagation*	Data *history* penjualan *spare part* beberapa bulan sebelumnya	*Backpropagation*	Jumlah persediaan *spare part*
			Nilai *error*

Penelitian saat ini menggunakan banyak referensi, seperti beberapa penelitian yang telah dilakukan. Penelitian yang dijadikan referensi memiliki karakteristik yang berbeda sesuai dengan kebutuhan pada penelitian ini. Unsur bahan yang dijadikan referensi memiliki kesamaan ataupun yang dapat menunjang penelitian

ini seperti topik, objek, algoritme dan juga konsep. Tabel 2.1 menunjukkan perbedaan setiap penelitian yang menjadi referensi dengan penelitian ini. Berdasarkan Tabel 2.1 diketahui bahwa peramalan berhasil dilakukan. *Backpropagation* untuk peramalan menghasilkan nilai *error* yang relatif kecil, sehingga *backpropagation* baik digunakan untuk melakukan peramalan.

2.2 *Spare Part*

Spare Part atau suku cadang adalah stok barang persediaan umum yang diperlukan untuk memelihara dan merawat peralatan (Hu, et al., 2018). *Spare Part* adalah suatu barang yang terdiri dari beberapa komponen yang membentuk satu kesatuan dan mempunyai fungsi tertentu (Purba, 2015). Berdasarkan definisi di atas, suku cadang merupakan salah satu faktor yang menentukan jalannya proses produksi atau aktivitas dalam suatu perusahaan. Sehingga dapat dikatakan suku cadang ini mempunyai peranan yang cukup besar dalam serangkaian aktivitas perusahaan.

Spare Part secara umum dapat dibagi menjadi dua, yaitu (Afrianto, 2014):

1. *Spare Part* baru yaitu *spare part* yang belum pernah digunakan sama sekali kecuali untuk pengetesan.
2. *Spare Part* bekas atau copotan yaitu *spare part* yang pernah dipakai untuk periode tertentu dengan kondisi:
   a. Masih layak pakai.
   b. Tidak layak pakai.

2.3 Persediaan

Persediaan adalah barang yang dimiliki perusahaan pada waktu tertentu dengan tujuan untuk dijual atau digunakan dalam siklus proses perusahaan (Yousida, 2013). Definisi persediaan menurut Indrajit dan Djokopranoto (2003) adalah barang-barang yang biasanya dapat dijumpai di gudang tertutup, lapangan, gudang terbuka, atau tempat-tempat penyimpanan lain. Persediaan dapat berupa bahan baku, barang setengah jadi, barang jadi, barang-barang untuk keperluan proses produksi, atau barang-barang untuk keperluan suatu proyek (Supit & Jan, 2015).

Persediaan yang terdapat dalam perusahaan dapat dibedakan berdasarkan fungsinya, jenis dan posisi barang dalam urutan pengerjaan produk, berikut ada macam-macam persediaan (Heripracoyo, 2009) :

1. Dilihat dari fungsinya
   a. *Batch stock* atau *lot inventory*
   b. *Fluctuation stock*
   c. *Anticipation stock*
2. Dilihat dari jenis dan posisi produk dalam urutan pengerjaan produk

a. Persediaan bahan baku (*raw material stock*)

   Persediaan yang telah dibeli, tetapi belum diproses. Dapat digunakan untuk melakukan *decouple* (memisahkan) pemasok dari proses produksi (Supit & Jan, 2015).

b. Persediaan bagian produk atau *parts* yang dibeli (*purchase parts/component stock*)

c. Persediaan bahan-bahan pembantu atau barang-barang perlengkapan (*supplier stock*)

d. Persediaan barang setengah jadi atau barang dalam proses (*work in process/progress stock*)

   Komponen atau bahan baku atau bahan mentah yang telah melewati beberapa proses perubahan, tetapi belum selesai diproses menjadi barang jadi (Supit & Jan, 2015).

e. Persediaan barang jadi (*finished goods stock*)

   Produk yang telah selesai dan tinggal menunggu pengiriman dan penggunaan. Barang jadi dimasukan ke dalam persediaan karena permintaan pelanggan di masa mendatang tidak diketahui (Supit & Jan, 2015).

Pencatatan persediaan biasa dilakukan untuk mengetahui mengenai persediaan. Terdapat berbagai macam pencatatan yang berhubungan dengan persediaan, diantaranya adalah (Heripracoyo, 2009):

1. Permintaan untuk dibeli (*purchase requisition*)
2. Laporan penerimaan (*receiving report*)
3. Catatan persediaan (*balances of stores record*)
4. Daftar permintaan bahan (*material requisition form*)
5. Perkiraan pengawasan (*control accounting*)

2.4 Peramalan

Peramalan adalah proses untuk memperkirakan atau memprediksi jumlah kebutuhan masa mendatang (Pakaja, et al., 2012). Kebutuhan tersebut meliputi berbagai aspek, yaitu kualitas, kuantitas, lokasi dan waktu. Peramalan merupakan seni ilmu yang menggunakan pendekatan-pendekatan matematis untuk memprediksi peristiwa-peristiwa masa mendatang dengan menggunakan data terdahulu yang diproyeksikan ke masa mendatang (Razak & Riksakomara, 2017). Peramalan adalah metode untuk memperkirakan atau memprediksi nilai pada masa mendatang berdasarkan data pada masa lampau (Wardah & Iskandar, 2016).

Peramalan dilakukan agar dapat melakukan beberapa hal berikut (Wardah & Iskandar, 2016):

1. Meminimalisir pengaruh ketidak pastian terhadap perusahaan.

2. Meminimalisir kesalahan meramal (*forecast error*) yang biasanya diukur dengan berbagai tolak ukur kesalahan, seperti MSE (*Mean Squared Error*), *MAE* (*Mean Absolute Error*) dan sebagainya .

Peramalan dapat memiliki kualitas atau mutu yang baik apabila prosedur atau langkah-langkah penyusunannya baik. Pada dasarnya ada beberapa langkah penting dalam peramalan, yaitu (Wardah & Iskandar, 2016):

1. Menganalisa data yang lalu (data *historis*) untuk mengetahui pola pada masa lalu.
2. Menentukan data yang dipergunakan.
3. Menentukan metode peramalan. Metode sangat mempengaruhi hasil peramalan, maka harus memilih metode yang dapat menghasilkan nilai peramalan yang tidak jauh beda dengan nilai aktualnya.
4. Data yang lalu (data *history*) diproyeksikan dengan mempertimbangkan faktor-faktor yang mempengaruhi menggunakan metode yang dipergunakan.

Terdapat beberapa hal yang harus diperhatikan dalam peramalan. Prinsip-prinsip yang perlu diperhatikan adalah sebagai berikut (Wardah & Iskandar, 2016):

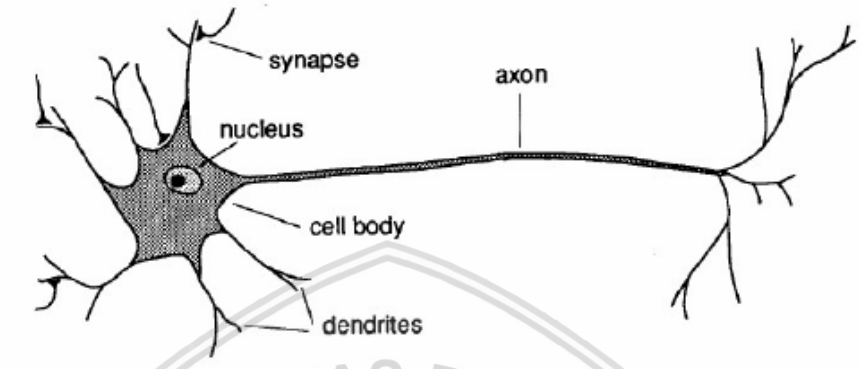
1. Peramalan tetap mempunyai kesalahan (*error*). Peramalan tidak menghilangkannya ketidakpastian, hanya mengurangi ketidakpastian.
2. Peramalan sebaiknya menggunakan tolak ukur kesalahan peramalan untuk mengetahui besar kesalahan. Tolak ukur kesalahan dapat dinyatakan dalam satuan unit atau persentase (*probability*) permintaan aktual akan jatuh dalam interval peramalan.
3. Peramalan jangka pendek lebih akurat dari pada peramalan jangka panjang. Hal ini disebabkan faktor yang mempengaruhi peramalan jangka pendek cenderung tidak banyak berubah atau tetap, sehingga peramalan jangka pendek lebih akurat.

2.5 Jaringan Syaraf Tiruan

Jaringan syaraf tiruan atau disebut juga *Artificial Neural Network* (ANN) adalah sistem pemrosesan informasi yang memiliki karakteristik kinerja sama dengan jaringan syaraf biologis (Fausett, 1993). Menurut Hecht-Nielsend (1988) sistem syaraf buatan adalah suatu struktur pemroses informasi yang terdistribusi dan bekerja secara paralel, dimana struktur tersebut terdiri atas elemen pemroses (elemen pemroses memiliki memori lokal dan beroperasi dengan informasi lokal) yang diinterkoneksi bersama dengan alur sinyal searah yang disebut koneksi. Setiap elemen pemroses memiliki koneksi keluaran tunggal yang bercabang (*fan out*) ke sejumlah koneksi kolateral yang diinginkan (setiap koneksi membawa sinyal yang sama dari keluaran elemen pemroses tersebut) (Pakaja, et al., 2012).

Jaringan syaraf terdiri dari sejumlah besar elemen pemrosesan sederhana yang disebut *neuron*, unit, sel, atau nodus (Fausett, 1993). Jaringan otak manusia tersusun dari 10^{13} *neuron* yang terhubung oleh sekitar 10^{15} dendrite (Pakaja, et

al., 2012). Jaringan syaraf dapat dilihat pada Gambar 2.1. Masing-masing bagian mempunyai fungsi yang berbeda. *Nucleus* adalah bagian inti dari suatu *neuron*, *dendrite* berfungsi menyampaikan sinyal dari *neuron* tersebut ke *neuron* yang terhubung dengannya, *axon* merupakan saluran keluaran dari *neuron*, dan *synapsis* berfungsi mengatur kekuatan hubungan antar *neuron* (Pakaja, et al., 2012).



Gambar 2.1 Jaringan Syaraf

Struktur dasar jaringan syaraf tiruan dapat dilihat pada Gambar 2.2. Setiap *neuron* terhubung ke *neuron* lain melalui hubungan komunikasi terarah, masing-masing memiliki bobot yang mewakili informasi yang digunakan oleh jaringan untuk memecahkan masalah (Fausett, 1993). Jaringan *neuron* buatan terdiri dari beberapa *neuron* yang memiliki lapisan (Pakaja, et al., 2012). Lapisan-lapisan tersebut adalah sebagai berikut (Razak & Riksakomara, 2017):

1. *Input Layer* (Lapisan Masukan)

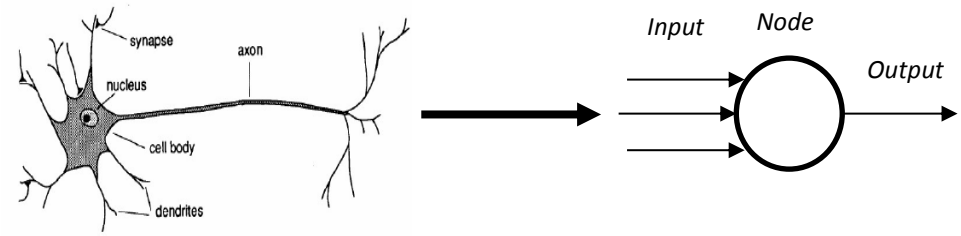
   Lapisan masukan merupakan lapisan yang menghubungkan sumber data ke jaringan pemrosesan. Variabel-variabel bebas merupakan representasi masukan yang mempengaruhi keluaran (*output*).

2. *Hidden Layer* (Lapisan Tersembunyi)

   Lapisan perambat variabel-variabel *input* untuk mendapatkan hasil *output* yang mendekati terbaik (hasil yang diinginkan). Suatu jaringan syaraf tiruan dapat tidak memiliki lapisan ini dan juga dapat memiliki satu *hidden layer* atau beberapa *hidden layer*.

3. *Ouput Layer* (Lapisan Keluaran)

   Lapisan ini merupakan hasil keluaran dari pemrosesan yang telah dilakukan jaringan syaraf tiruan. Keluaran yang didapatkan bergantung pada bobot, jumlah lapisan tersembunyi (*hidden layer*), dan fungsi aktivasi yang ditetapkan.



Gambar 2.2 Struktur Dasar Jaringan Syaraf Tiruan

Jaringan syaraf tiruan memiliki beberapa jenis *neural network*. Berdasarkan jenis *network*nya, *neural network* terdapat tiga jenis, yaitu (Pakaja, et al., 2012):

1. *Single-Layer Neural*

   Jaringan syaraf tiruan yang memiliki koneksi secara langsung dari jaringan *input* ke jaringan *output*.

2. *Multilayer Perceptron Neural Network*

   Jaringan syaraf tiruan yang mempunyai *hidden layer* yang berada di antara *input layer* dan *output layer*. *Hidden layer* ini bersifat variabel, dapat digunakan lebih dari satu *hidden layer*.

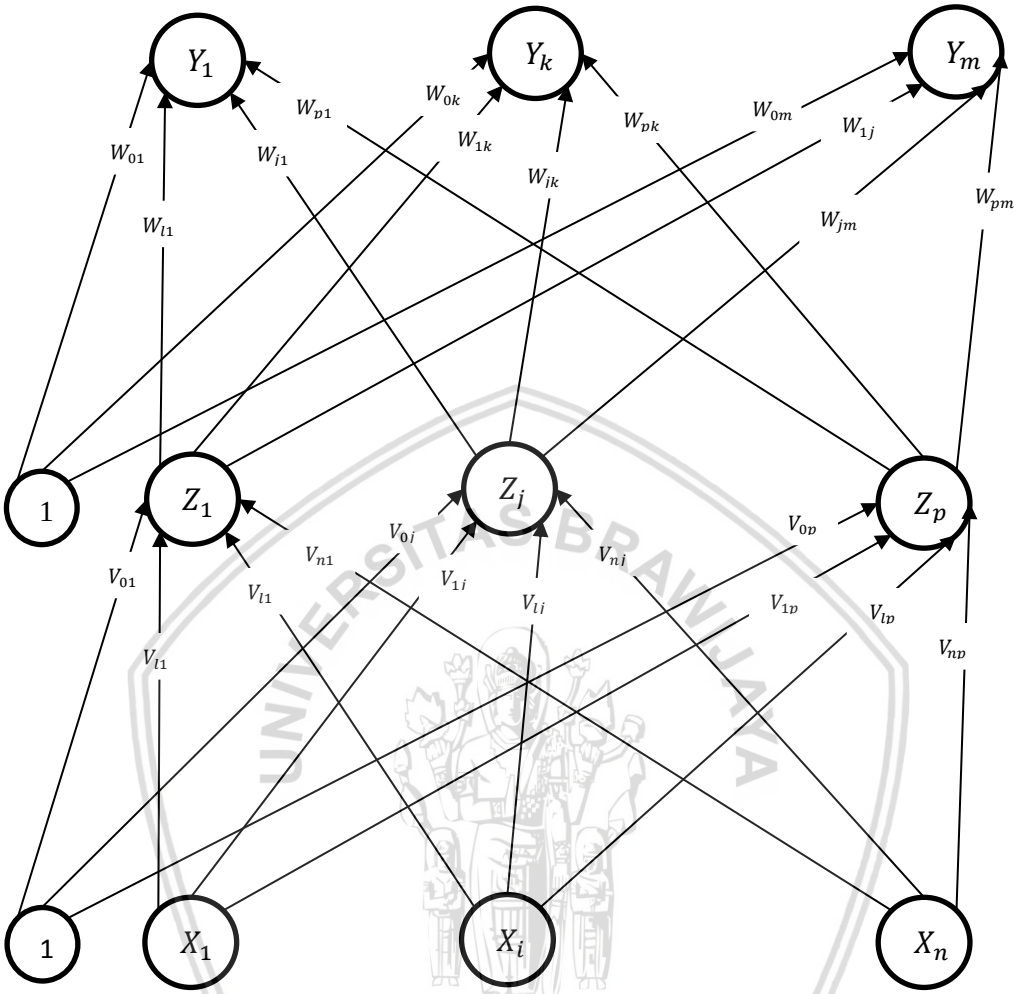
3. *Recurrent Neural Networks Neural Network*

   Jaringan syaraf tiruan yang memiliki koneksi umpan balik dari *output layer* ke *input layer*.

2.6 *Backpropagation*

Backpropagation merupakan salah satu algoritme Jaringan Syaraf Tiruan *Multilayer Perceptron Neural Network* dengan pembelajaran terawasi (nilai *output* sudah diketahui) dimana dilakukan penyesuaian bobot secara berulang untuk mendapatkan nilai *error* terendah antara hasil prediksi dengan target yang diinginkan (Razak & Riksakomara, 2017). *Backpropagation* memiliki kelebihan yaitu bersifat *adaptive* (dapat menyesuaikan terhadap dataset) dan *fault tolerance* (kesalahan *error* kecil) terhadap pemecahan masalah pada sistem (Razak & Riksakomara, 2017). Sifat yang umum dari pelatihan *backpropagation* berarti jaringan *backpropagation* dapat digunakan untuk memecahkan masalah di banyak area (Fausett, 1993). Pelatihan *backpropagation* melibatkan tiga tahap: *feedforward* dari pola pelatihan *input*, *backpropagation* dari kesalahan yang terkait, dan penyesuaian bobot. Setelah pelatihan, penerapan jaringan hanya melibatkan perhitungan fase *feedforward*. *Backpropagation* terdiri dari *input layer*, *hidden layer* dan *output layer* (Yohannes, et al., 2015). Arsitektur *backpropagation* dapat dilihat pada Gambar 2.3. Menurut Heaton (2005) satu hidden layer mampu menyelesaikan berbagai masalah, sedangkan penentuan jumlah node pada hidden layer harus memenuhi aturan sebagai berikut: berada dalam interval jumlah node *input* maupun *output layer*, jumlah node pada *hidden layer* adalah 2/3 dari jumlah node pada *input layer*, ditambah dengan jumlah node

pada *output layer* dan jumlah node pada *hidden layer* kurang dari dua kali jumlah node pada *input layer* (Yohannes, et al., 2015).



Gambar 2.3 Arsitektur *Backpropagation*

Penjelasan simbol yang digunakan pada algoritme *backpropagation* terdapat di Tabel 2.2. Selama tahap *feedforward* (maju), setiap node masukan (X_i) menerima sinyal masukan dan mengirim sinyal ini ke setiap node tersembunyi (*hidden* node) Z_1, …, Z_p. Setiap node tersembunyi menghitung aktivasi dan mengirim sinyalnya (Z_j) ke setiap node keluaran (*output* node). Setiap node keluaran (Y_k) menghitung aktivasi (y_k) untuk menunjukkan respon jaringan terhadap pola masukan yang diberikan (Fausett, 1993).

Selama pelatihan, setiap node keluaran membandingkan aktivasi y_k dihitung dengan nilai target t_k untuk menentukan kesalahan yang terkait dengan pola tersebut dengan node tersebut. Berdasarkan kesalahan ini, faktor δ_k (k = 1, ..., m) dihitung. δ_k digunakan untuk mendistribusikan kesalahan pada node keluaran (*output*) Y_k kembali ke semua node pada lapisan sebelumnya (node tersembunyi yang terhubung ke Y_k). Hal ini juga digunakan untuk memperbarui bobot antara *output* dan lapisan tersembunyi (*hidden layer*). Dengan cara yang sama, faktor δ_j

(j = 1, ..., p) dihitung untuk setiap node tersembunyi (Z_j). Hal ini tidak perlu untuk menyebarkan kesalahan kembali ke lapisan masukan (*input layer*), tapi δ_j digunakan untuk memperbarui bobot antara lapisan tersembunyi dan lapisan masukan (Fausett, 1993).

Setelah semua dari faktor δ telah ditentukan, bobot untuk semua lapisan disesuaikan secara bersamaan. Penyesuaian terhadap berat w_{jk} (dari node tersembunyi Z_j ke node keluaran Y_k didasarkan pada faktor δ_k dan aktivasi $Z_i$dari node tersembunyi Z_j. Penyesuaian dengan berat v_{ij} (dari node masukan X_i, ke node tersembunyi Z_j) didasarkan pada faktor δ, dan aktivasi x_i dari node masukan (Fausett, 1993).

Tabel 2.2 Parameter pada *Backpropagation*

Simbol	Keterangan
x	*Input training vector.*
t	*Output target vector*.
δ_k	Bagian penyesuaian koreksi kesalahan (*error*) untuk W_{jk}yang disebabkan oleh kesalahan pada node keluaran Y_k; Juga, informasi tentang kesalahan pada node Y_k yang disebarkan kembali ke node tersembunyi yang masuk ke node Y_k.
δ_j	Bagian penyesuaian koreksi kesalahan (*error*) untuk v_{ij} yang disebabkan oleh informasi kesalahan *backpropagation* dari lapisan keluaran ke node tersembunyi Z_j.
α	*Learning rate*. Parameter ini merupakan parameter umum dalam pembelajaran untuk mempercepat penemuan solusi optimal.
X_i	Node masukan *i*. Untuk node masukan, sinyal masukan sinyal keluaran sama, yaitu *x*.
v_{0j}	Bias pada node tersembunyi *j*
Z_j	Node tersembunyi *j* Masukan untuk Z_j dinotasikan z_in_j Sinyal keluaran dari Z_j dinotasikan z_j
w_{0k}	Bias node keluaran *k*
Y_k	Node keluaran *k* Masukan untuk Y_k dinotasikan y_in_k Sinyal keluaran dari Y_k dinotasikan y_k

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \quad (2.1)$$

$$z_j = f(z\_in_j) \quad (2.2)$$

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk} \quad (2.3)$$

$$y_k = f(y\_in_k) \quad (2.4)$$

$$f_1(x) = \frac{1}{1+\exp(-x)} \quad (2.5)$$

$$f_1' = f_1(x)\,[1 - f_1(x)] \quad (2.6)$$

$$f_2(x) = \frac{1}{1+\exp(-x)} - 1 \quad (2.7)$$

$$f_2' = \frac{1}{2}\,[1 - f_2(x)][1 - f_2(x)] \quad (2.8)$$

$$tahn(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.9)$$

$$\delta_k = (t_k - y_k) f'(y\_in_k) \quad (2.10)$$

$$\Delta w_{jk} = \alpha \delta_k z_j \quad (2.11)$$

$$\Delta w_{0k} = \alpha \delta_k \quad (2.12)$$

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk} \quad (2.13)$$

$$\delta_j = \delta\_in_j\, f'(z\_in_j) \quad (2.14)$$

$$\Delta v_{ij} = \alpha \delta_j x_i \quad (2.15)$$

$$\Delta v_{0j} = \alpha \delta_j \quad (2.16)$$

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} \quad (2.17)$$

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij} \quad (2.18)$$

Langkah-langkah untuk melakukan pelatihan pada algoritme *backpropagation* adalah sebagai berikut (Fausett, 1993):

Langkah 1 Inisialisasi bobot dan bias. Inisialisasi bobot dan bias bisa dilakukan dengan melakukan *random initialization*. Umumnya, menginisialisasi bobot (dan bias) ke nilai acak antara -0,5 dan 0,5.

Langkah 2 Melakukan langkah 3 hingga langkah 8 selama kondisi berhenti belum terpenuhi.

Feedforward

Langkah 3 Setiap node masukan (X_i, i = 1, …, n) menerima sinyal masukan x_i dan mengirim sinyal tersebut ke seluruh node pada lapisan berikutnya (*hidden layer* atau lapisan tersembunyi).

Langkah 4 Setiap node tersembunyi (Z_j, j = 1, …, p), menjumlahkan bobot sinyal input dengan Persamaan 2.1 dan keluarannya dihitung dengan menerapkan fungsi aktivasi yaitu Persamaan 2.2. Fungsi aktivasi sendiri dapat dihitung dengan Persamaan 2.5, Persamaan 2.7 dan Persamaan 2.9.

Langkah 5 Setiap node keluaran (Y_k, k = 1, …, m) memasukkan sinyal masukan yang memiliki bobot dengan Persamaan 2.3 dan keluarannya dihitung dengan menerapkan fungsi aktivasi yaitu Persamaan 2.4 Fungsi aktivasi sendiri dapat dihitung dengan Persamaan 2.5, Persamaan 2.7 dan Persamaan 2.9.

Backpropagation

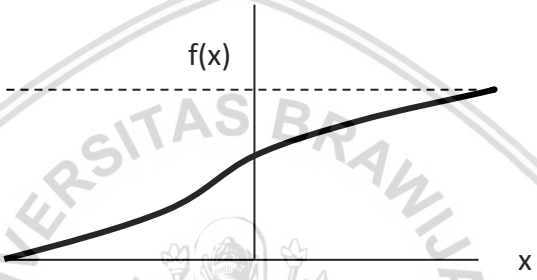
Langkah 6 Setiap node keluaran (Y_k, k = 1, …, m) menerima pola target yang sesuai dengan pola pelatihan masukan, menghitung koreksi *error*-nya dengan Persamaan 2.10, hitung koreksi bobotnya dengan Persamaan 2.11 dan hitung koreksi biasnya dengan Persamaan 2.12. δ_k dikirim ke lapisan berikutnya.

Langkah 7 Setiap node tersembunyi (Z_j, j = 1, …, p) menjumlahkan delta masukan (dari lapisan sebelumnya) dengan Persamaan 2.13 dikalikan dengan fungsi aktivasi agar mengetahui koreksi *error*-nya dengan Persamaan 2.14. Hitung koreksi bobotnya dengan Persamaan 2.15 dan koreksi biasnya dengan Persamaan 2.16.

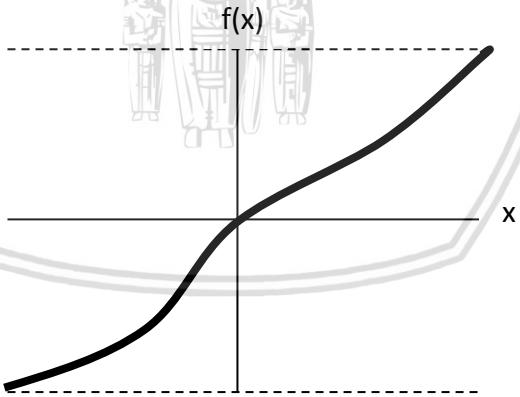
Update bobot dan bias

Langkah 8 Setiap node keluaran (Z_j, j = 1, …, p) update bias dan bobot (j = 0, …, p) dengan Persamaan 2.17. Setiap node tersembunyi (Z_j, j = 1, …, p) update bias dan bobot (i = 0, …, n) dengan Persamaan 2.18.

Fungsi aktivasi pada algoritme *backpropagation* memiliki karakteristik *kontinyu*, terdeferensial dan *monotonically non-decreasing* (tidak menurun secara monotis) (Fausett, 1993). Fungsi aktivasi yang paling sering digunakan adalah fungsi aktivasi *sigmoid biner* dan fungsi aktivasi *sigmoid bipolar*. Fungsi aktivasi *sigmoid bipolar* erat kaitannya dengan fungsi aktivasi *arctangent*. Fungsi aktivasi *sigmoid biner* dapat dilihat pada Persamaan 2.5 dan fungsi aktivasi *sigmoid bipolar* dapat dilihat pada Persamaan 2.7, sedangkan fungsi aktivasi *arctangent* dapat dilihat pada Persamaan 2.9. Fungsi aktivasi *sigmoid biner* memiliki nilai pada *range* 0 hingga 1, sedangkan fungsi aktivasi *sigmoid bipolar* memiliki nilai pada *range* -1 hingga 1. Fungsi aktivasi *sigmoid biner* digambarkan pada Gambar 2.4 dan fungsi aktivasi *sigmoid bipolar* pada Gambar 2.5. Fungsi aktivasi *sigmoid biner* digunakan untuk jaringan syaraf yang membutuhkan nilai output antara 0 hingga 1 dan 0 atau 1, dikarenakan mempunyai *range* 0 hingga 1 (Tanjung, 2014-2015).



Gambar 2.4 Fungsi Aktivasi *Sigmoid Biner* (0,1)



Gambar 2.5 Fungsi Aktivasi *Sigmoid Bipolar* (-1,1)

2.6.1 Normalisasi dan Denormalisasi data

Data yang akan digunakan pada perhitungan *backpropagation* harus memiliki nilai sesuai *range* fungsi aktivasi. Jika nilai pada data tidak sesuai *range*, maka perlu dilakukan normalisasi (Rachman, et al., 2018). Agar nilai pada data dapat bernilai 0 hingga 1, dapat menggunakan Persamaan 2.19 (Cynthia & Ismanto, 2017). Hasil

dari *backpropagation* juga perlu dilakukan denormalisasi agar menghasilkan nilai yang sesuai dengan data. Denormalisasi dilakukan dengan Persamaan 2.20.

$$x' = \frac{x - min}{\max - min} \quad (2.19)$$

$$y = y' (\max - \min) + min \quad (2.20)$$

Keterangan:

x' : nilai hasil normalisasi

x : nilai sebelum dinormalisasi

y' : nilai sebelum didenormalisasi

y : nilai hasil denormalisasi

$\max$: nilai maksimum yang terdapat pada dataset

min : nilai minimum yang terdapat pada dataset

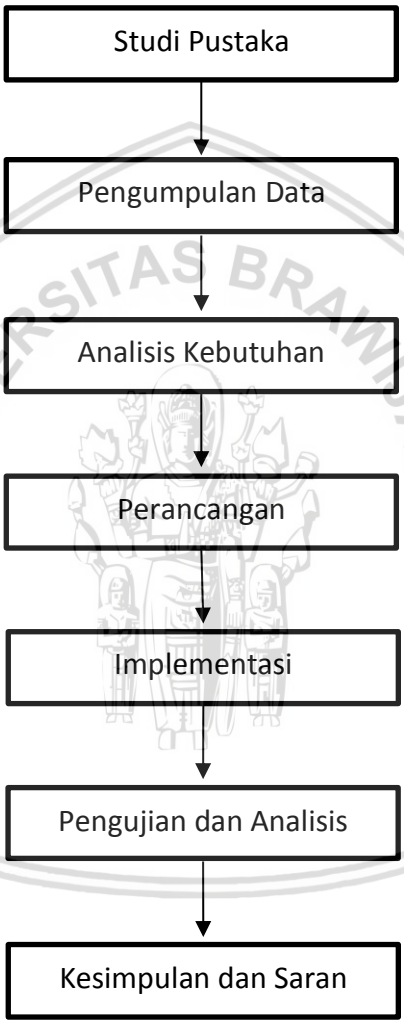
2.6.2 *Mean Squared Error* (MSE)

Mean square error (*MSE*) adalah salah satu metode yang digunakan untuk menguji kesalahan pada peramalan (Yohannes, et al., 2015). Untuk menghitung *Mean Squared Error* dapat dilihat pada Persamaan 2.21, dengan penjelasan F_i adalah peramalan (*forecast*) pada periode-*i*, X_i adalah permintaan aktual pada periode-*i* dan n adalah jumlah periode peramalan yang terlibat (Augustine & Manuharawati, 2017).

$$MSE = \frac{\sum_{i=1}^{n} |X_i - F_i|^2}{n} \quad (2.21)$$

BAB 3 METODOLOGI

Metodologi penelitian adalah suatu sistematik tahapan penelitian untuk menyelesaikan masalah yang ada pada penelitian. Metodologi penelitian yang dilakukan pada penelitian ini ada beberapa tahapan yaitu studi pustaka, pengumpulan data, analisis kebutuhan, perancangan, implementasi, pengujian dan analisis, serta kesimpulan dan sara. Metodologi penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 Metodologi Penelitian

3.1 Studi Pustaka

Studi pustaka adalah metode untuk mendapatkan teori pendukung yang berkaitan dengan penelitian dan menjadi dasar dalam penelitian ini. Pustaka pada penelitian dapat diperoleh dari buku, ebook , jurnal dan dokumentasi *project*. Teori yang ada pada studi pustaka penelitian ini adalah sebagai berikut:

a. *Spare Part*

b. Persediaan

c. Peramalan

d. Jaringan Syaraf Tiruan

e. Algoritme *Backpropagation*

3.2 Pengumpulan Data

Data yang digunakan pada penelitian ini didapatkan dari satu *dealer* Yamaha Motor, yaitu Pangestu Utomo Motoryang terletak di Candi, Sidoarjo. Data yang digunakan meliputi data *history* penjualan *spare part.* Data yang digunakan adalah perbulan dan terdapat 60 bulan data. Data tersebut sebagian akan digunakan sebagai data latih dan sebagian digunakan sebagai data uji.

3.3 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengetahui kebutuhan yang harus ada dalam merancang, mengimplementasi dan menguji. Analisis kebutuhan ini juga akan dilakukan pada Pangestu Utomo Motor. Analisis kebutuhan akan dilakukan dengan cara wawancara kepada pemilik dan karyawan Pangestu Utomo Motor. Hasil wawancara terdapat pada Lampiran B. Kebutuhan minimal yang digunakan dalam perancangan, implementasi dan pengujian adalah sebagai berikut:

1. Hardware yang dibutuhkan adalah laptop dengan spesifikasi sebagai berikut:
   - Prosesor Intel Quad Core / i3
   - RAM 4 GB
2. Software yang dibutuhkan, meliputi:
   - Sistem operasi, sistem operasi yang digunakan adalah windows
   - Netbean sebagai IDE bahasa pemrograman java
   - Microsoft Office untuk perhitungan manual dan penempatan data

3.4 Perancangan

Perancangan berisi rancangan kerja yang akan dibangun. Tujuan perancangan ini agar mempermudah implementasi dan pengujian algoritme. Langkah-langkah yang dilakukan dalam perancangan, yaitu:

1. Perancangan peramalan persediaan *spare part* menggunakan algoritme *backpropagation*.
2. Perancangan antarmuka untuk mempermudah pembuatan program. Perancangan antarmuka akan dijelaskan dalam bentuk gambar sederhana.

3.5 Implementasi

Implementasi peramalan persediaan *spare part* menggunakan algoritme *backpropagation* adalah menerapkan hal-hal yang telah didapat dalam proses

studi literatur dan representasi perancangan. Fase-fase yang ada di dalam implementasi antara lain:

1. Menerapkan algoritme *backpropagation* untuk peramalan persediaan *spare part* sepeda motor pada program dengan bahasa java menggunakan IDE Netbeans.
2. Membuat tampilan antarmuka agar lebih mudah digunakan dan dipahami.

3.6 Pengujian

Pengujian yang dilakukan adalah pengujian jumlah iterasi, jumlah node *input layer*, jumlah node *hidden layer* dan learning rate (α). Dari pengujian tersebut dapat diketahui nilai terbaik jumlah node *input layer*, jumlah node *hidden layer*, jumlah iterasi dan learning rate (α) yang dapat digunakan untuk melakukan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*. Diharapkan dengan menggunakan nilai terbaik dapat menghasilkan hasil peramalan yang lebih baik.

3.7 Kesimpulan dan Saran

Setelah melakukan semua tahapan penelitian, maka akan disimpulkan hasil dari penelitian yang diusulkan ini. Kesimpulan penelitian ini bisa berisi hasil peramalan menggunakan *backpropagation*, hasil nilai *error* peramalan menggunakan *backpropagation,* perbandingan nilai aktual dengan nilai hasil peramalan dan mengetahui kelebihan serta kekurangan dari penelitian ini. Saran didasarkan atas hasil temuan pada penelitian yang dilakukan. Saran berisi kekurangan yang ada penelitian ini dan memberikan penjelasan apa yang dapat diperbaiki untuk penelitian selanjutnya.

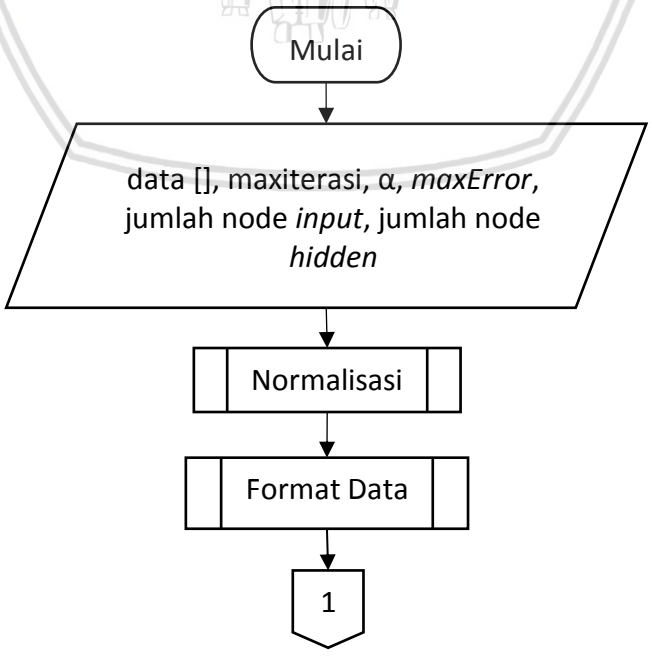
BAB 4 PERANCANGAN

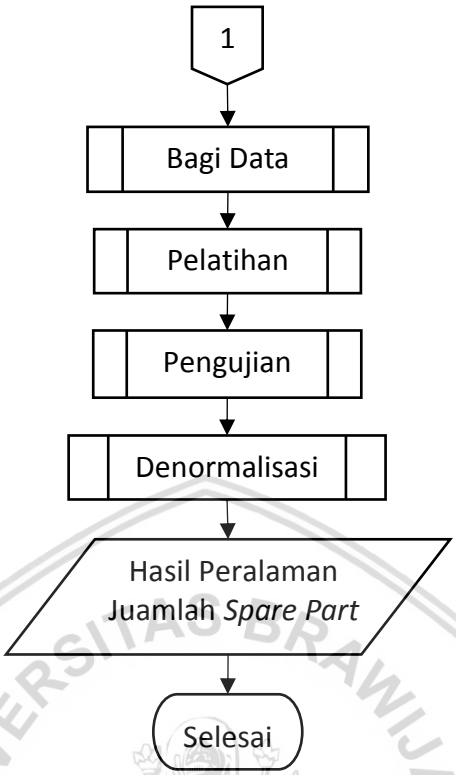
4.1 Formulasi Permasalahan

Permasalahan persediaan seringkali dialami oleh salah satu *dealer* Yamaha Motor di Candi Sidoarjo, yaitu Pangestu Utomo Motor. Dimana persediaan *spare part* yang dimiliki tidak sesuai dengan kebutuhan sehingga mengakibatkan kerugian secara materi maupun non materi. Permasalahan yang akan diselesaikan pada penelitian ini adalah peramalan persediaan *spare part* sepeda motor pada Pangestu Utomo Motor. *Spare part* yang akan diramalkan pada penelitian ini adalah *spare part* yang masuk dalam kategori *fast moving* pada Pangestu Utomo Motor, yaitu oli *yamalube matic*. *Input*-an yang digunakan merupakan data *history* beberapa bulan sebelumnya dan akan menghasilkan *output* jumlah persediaan *spare part* untuk bulan berikutnya.

4.2 Perancangan Peramalan Persediaan *Spare Part*

Algoritme yang akan digunakan untuk menyelesaikan permasalahan yang ada adalah *backpropagation*. Pada sub bab ini dijelaskan alur atau urutan proses peramalan persediaan *spare part* menggunakan algoritme *backpropagation* dalam penyelesaian permasalahan ini. Secara umum, perancangan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* dijelaskan dengan diagram alir pada Gambar 4.1. Berdasarkan diagram alir tersebut dapat dijelaskan bahwa data dan nilai-nilai yang diinputkan, diproses pada beberapa tahapan yaitu normalisasi; format data; bagi data; pelatihan; pengujian dan denormalisasi. Algortime *backpropagation* terdapat pada proses pelatihan dan pengujian.

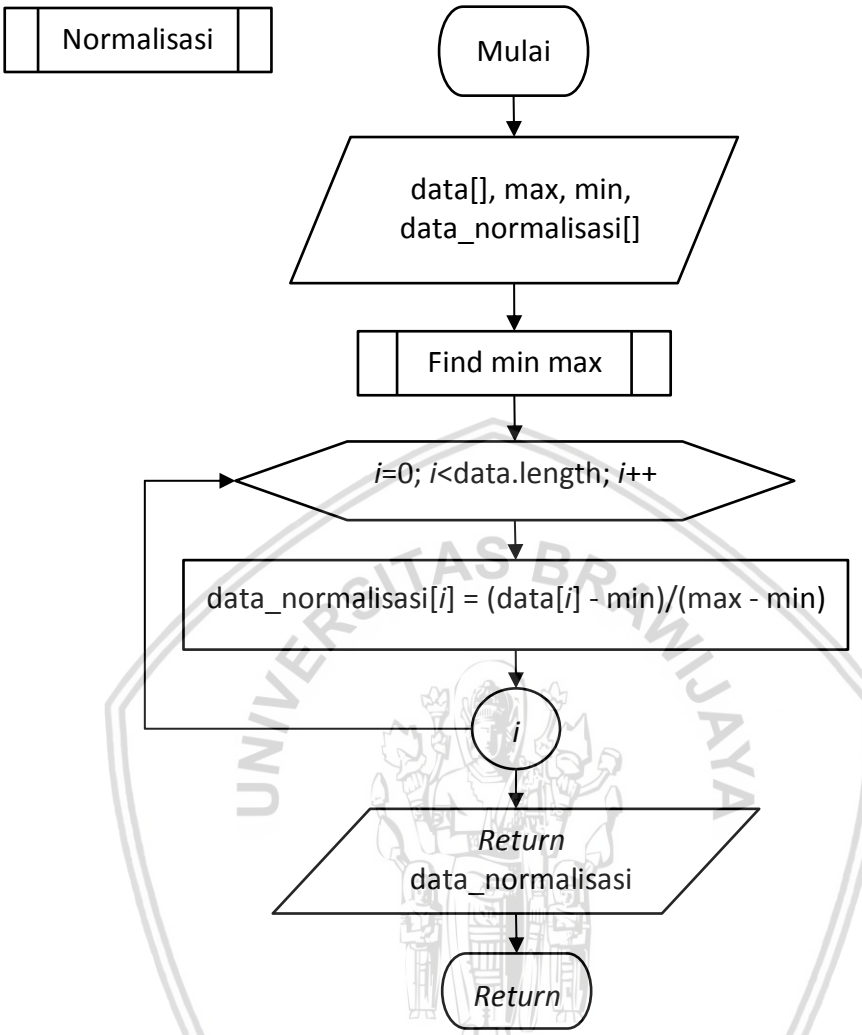




Gambar 4.1 Diagram Alir Perancangan Peramalan

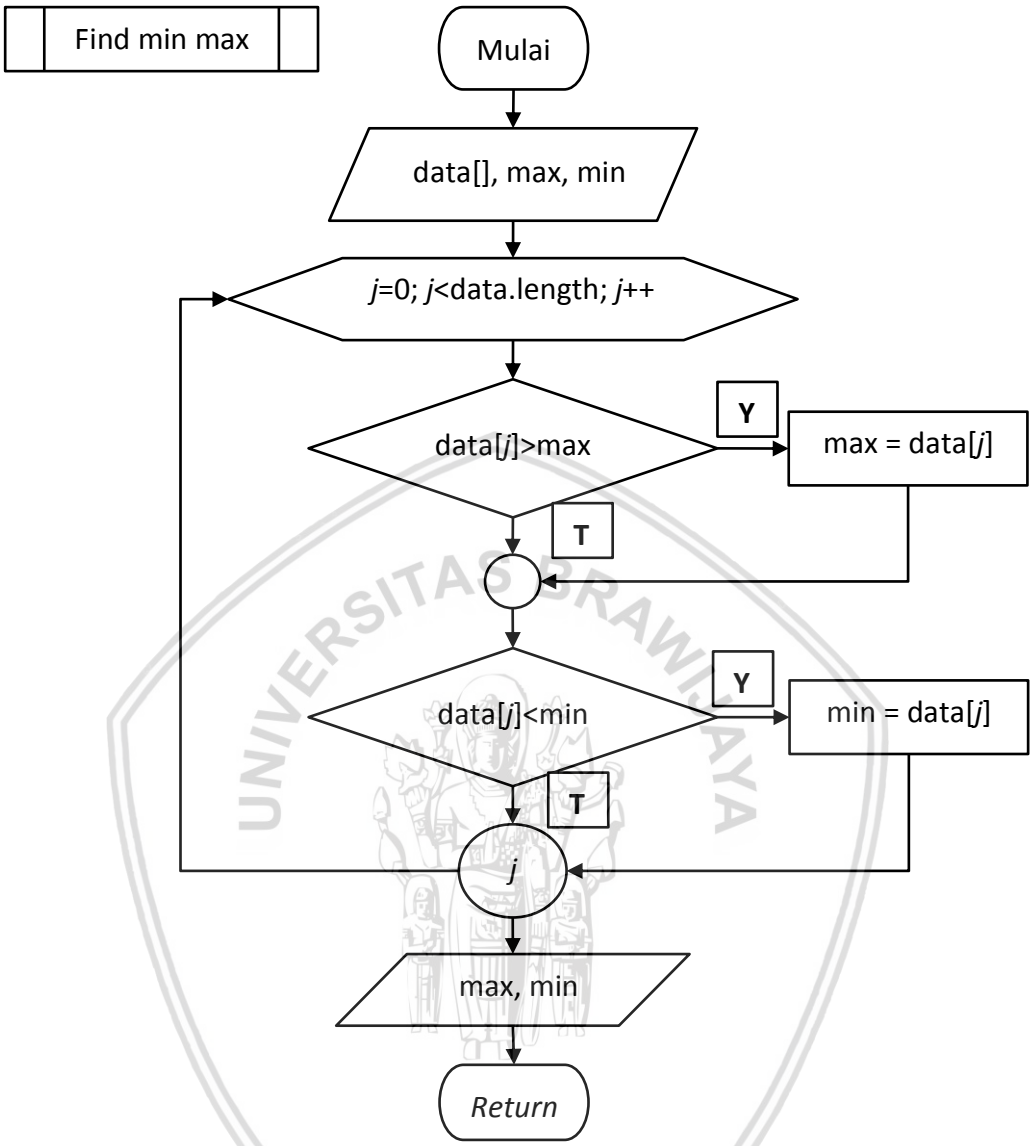
4.2.1 Normalisasi

Fungsi aktivasi yang digunakan pada peramalan ini adalah fungsi aktivasi *sigmoid biner*. *Fungsi* aktivasi *sigmoid biner* memiliki nilai pada *range* 0 hingga 1, sehingga data yang akan digunakan pada perhitungan peramalan pesediaan menggunakan *backpropagation* harus memiliki nilai sesuai *range* fungsi aktivasi. Nilai pada dataset peramalan ini belum memenuhi *range* aktivasi yang digunakan, oleh karena itu dilakukan normalisasi dengan Persamaan 2.19. Perancangan normalisasi digambarkan dengan diagram alir pada Gambar 4.2. Ketika fungsi *normalisasi* dijalankan, pertama yang dilakukan adalah menetukan variabel yang akan digunakan dalam proses yaitu dataset(*data*[]), nilai maximal pada dataset (*max*), nilai minimal pada dataset (*min*) dan panjang hasil normalisasi (*data_normalisasi*[]). Nilai maximal dan minimal yang digunakan pada normalisasi diperoleh dengan proses find min max. Kemudian menghitung nilai normalisasi dengan Persamaan 2.19 hingga banyaknya dataset.



Gambar 4.2 Diagram Alir Normalisasi

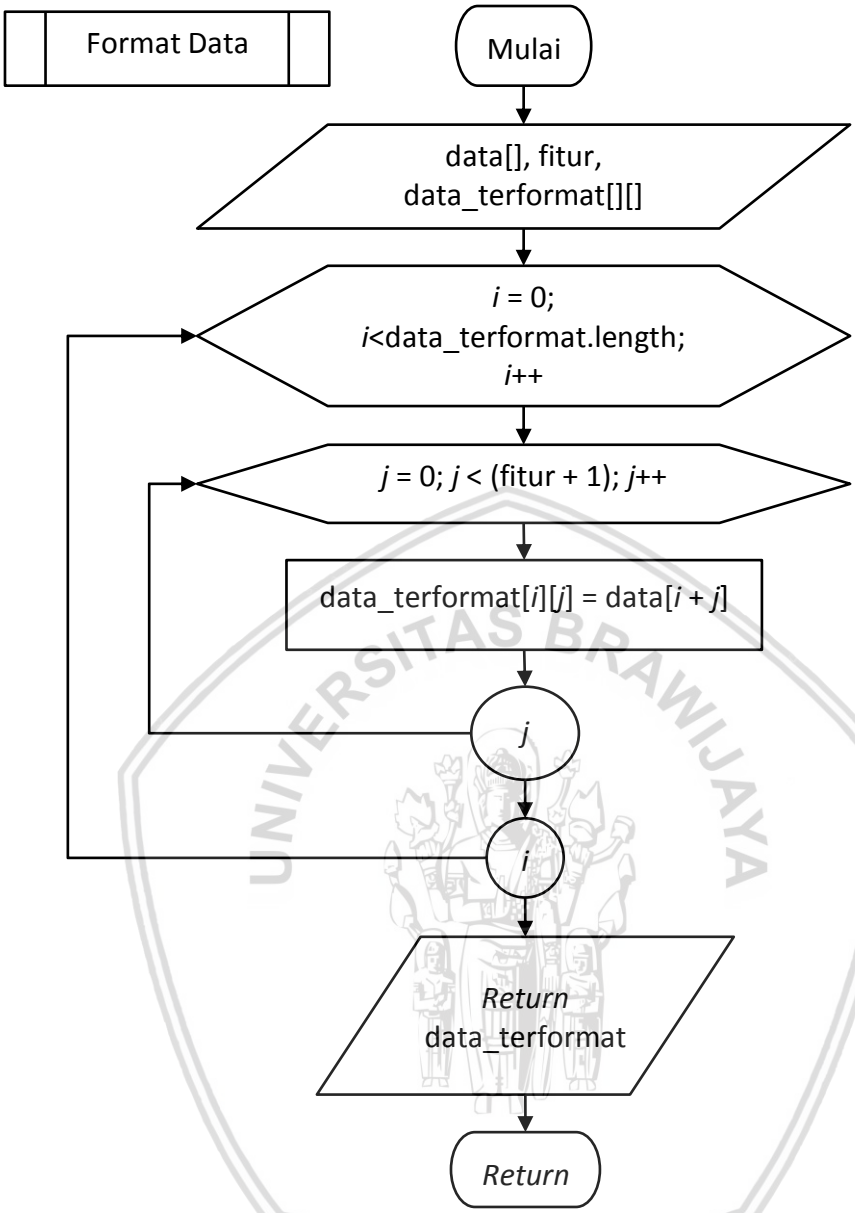
Nilai maximal dan nilai minimal yang digunakan pada proses normalisasi didapatkan dari nilai maximal dan nilai minimal pada dataset. Perancangan untuk menemukan nilai maximal dan minimal digambarkan dengan diagram alir pada Gambar 4.3. Melakukan perhitungan untuk menemukan nilai maximal dan minimal dengan *input*an dataset yang akan digunakan yaitu *data*. Kemudian menetapkan nilai *max* dan *min* awal. Perhitungan dilakukan pada semua data (dilakukan perulangan). Jika nilai data lebih besar dari nilai *max*, maka nilai *max* menjadi nilai data tersebut dan jika nilai data lebih kecil dari nilai *min*, maka nilai *min* menjadi nilai data tersebut.



Gambar 4.3 Diagram Alir Mencari Nilai Maximal dan Minimal

4.2.2 Format Data

Proses format data untuk mengolah dataset menjadi data yang memiliki sejumlah fitur yang telah ditentukan dan target serta menentukan nilai data tersebut. Perancangan format data digambarkan dengan diagram alir pada Gambar 4.4. Panjang *data_terformat* ditentukan berdasarkan banyaknya *data* dan *fitur*. Perulangan dilakukan selama *i* kurang dari panjang *data* dan *j* kurang dari nilai *fitur* ditambah 1. Nilai *data_terformat* pada elemen baris ke-*i* dan elemen kolom ke-*j* adalah nilai *data* ke-(*i* ditambah *j*).

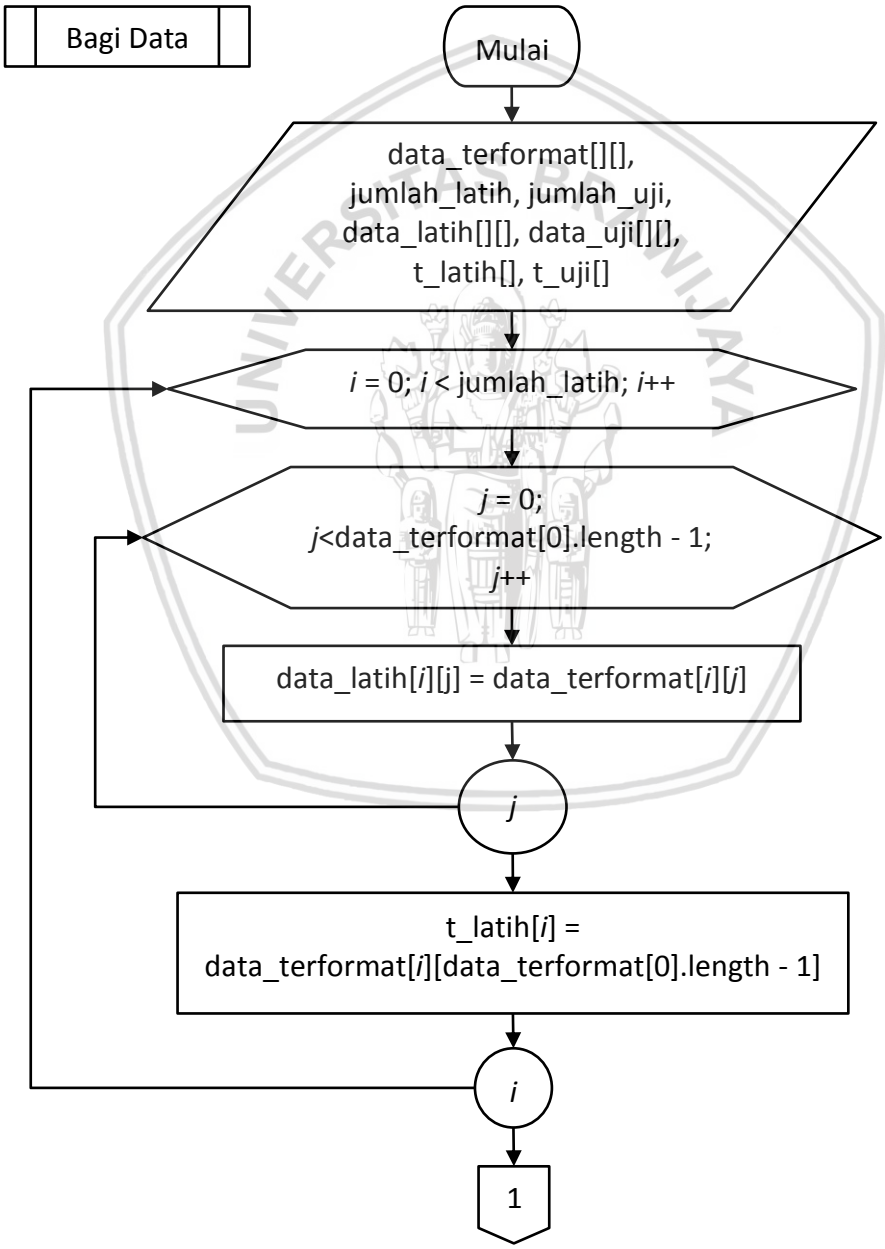


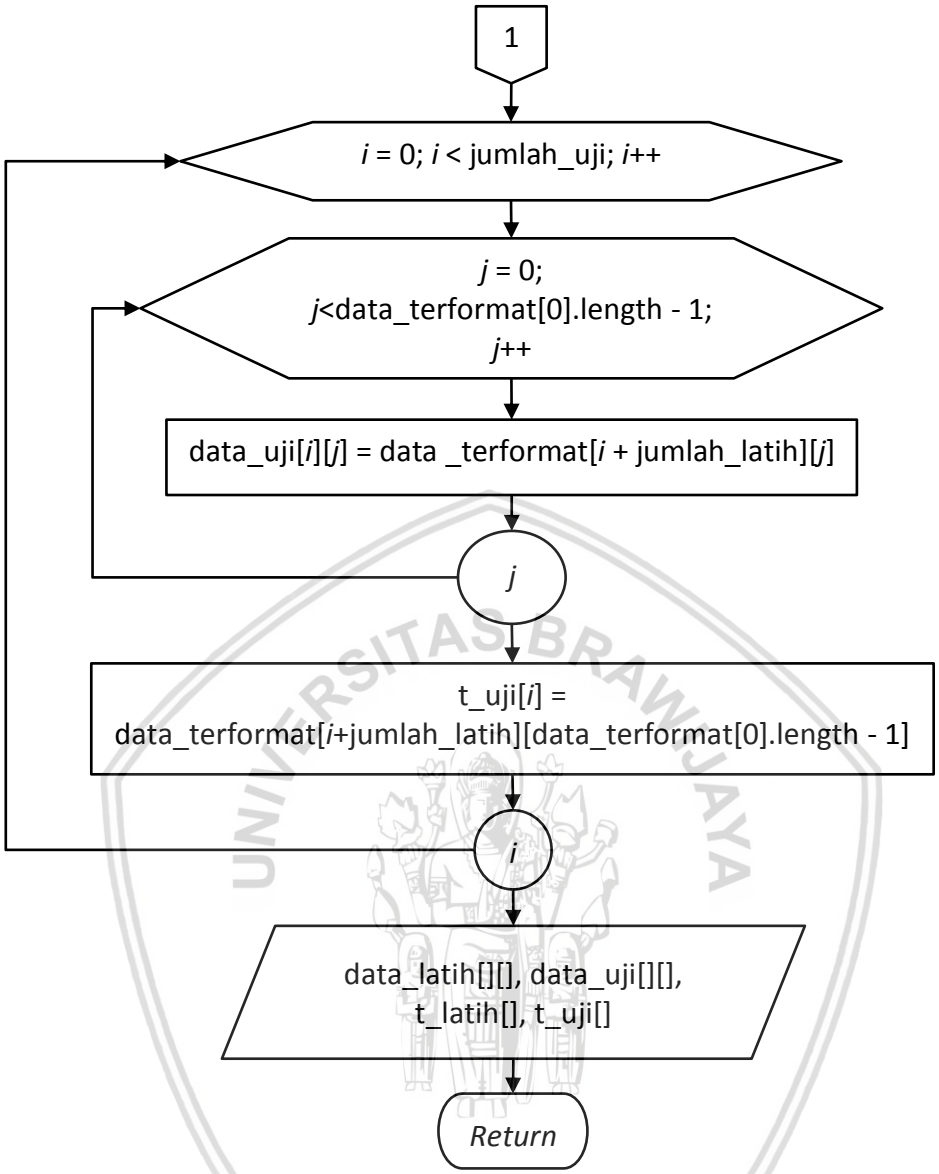
Gambar 4.4 Diagram Alir Format Data

4.2.3 Bagi Data

Proses bagi data untuk membagi data yang telah melalui proses data format menjadi data latih serta data uji. Perancangan bagi data digambarkan dengan diagram alir pada Gambar 4.5. Hasil pada proses format digunakan pada bagi data, *data_terformat* dibagi menjadi data latih dan data uji. Pada proses ini juga akan diketahui target pada setiap data. Panjang *data_latih*, *data_uji*, *t_latih* dan *t_uji* diketahui berdasarkan *jumlah_latih* dan *jumlah_uji* yang ditentukan. Untuk mendapatkan nilai *data_latih* dilakukan perulangan selama kondisi *i* kurang dari *jumlah_latih* dan *j* kurang dari panjang elemen baris *data_terformat* dikurangi satu, nilai *data_latih* pada elemen baris ke-*i* dan elemen kolom ke-*j* adalah nilai *data_terformat* pada elemen baris ke-*i* dan elemen kolom ke-*j*. Untuk

mendapatkan nilai *t_latih* dilakukan perulangan selama kondisi *i* kurang dari *jumlah_latih*, nilai *t_latih* ke-*i* adalah nilai *data_terformat* pada elemen baris ke-*i* dan elemen kolom ke- panjang elemen baris *data_terformat*. Untuk mendapatkan nilai *data_uji* dilakukan perulangan selama kondisi *i* kurang dari *jumlah_uji* dan *j* kurang dari panjang elemen baris *data_terformat* dikurangi satu, nilai *data_uji* pada elemen baris ke-*i* dan elemen kolom ke-*j* adalah nilai *data_terformat* pada elemen baris ke- *i* ditambah *jumlah_latih* dan elemen kolom ke-*j*. Untuk mendapatkan nilai *t_uji* dilakukan perulangan selama kondisi *i* kurang dari *jumlah_uji*, nilai *t_iji* ke-*i* adalah nilai *data_terformat* pada elemen baris ke- *i* ditambah *jumlah_latih* dan elemen kolom ke- panjang elemen baris *data_terformat*.



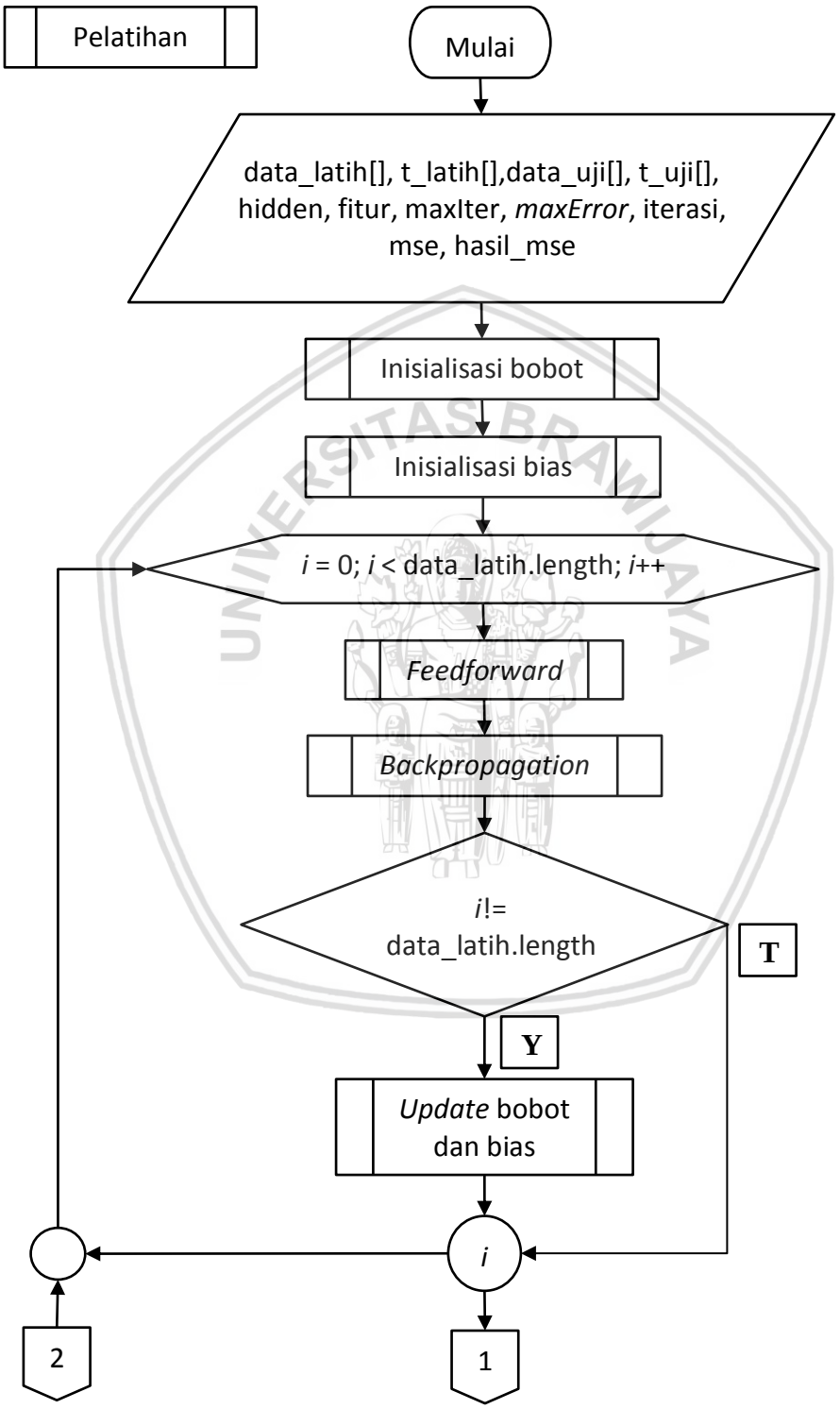


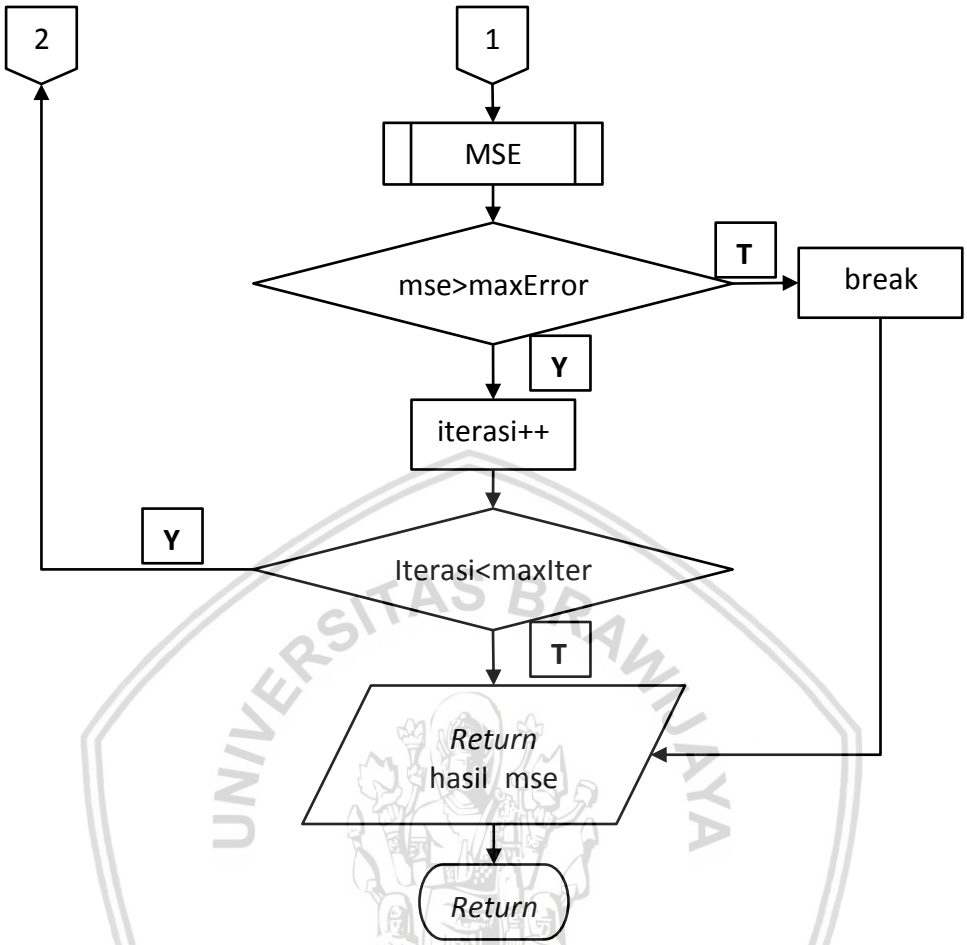
Gambar 4.5 Diagram Alir Bagi Data

4.2.4 Pelatihan

Pelatihan atau *training* dilakukan untuk melatih jaringan syaraf tiruan menggunakan data *input*-an. Pelatihan ini bertujuan untuk mendapatkan bobot yang optimal. Bobot hasil pelatihan digunakan untuk menghitung pada proses pengujian. *Backpropagation* memiliki 3 proses utama, yaitu *feedforward*, *backpropagation* dan *update* bobot dan bias. Perancangan pelatihan *backpropagation* digambarkan dengan diagram alir pada Gambar 4.6. Proses pelatihan dimulai dengan menentukan variabel dan parameter yang digunakan pada proses pelatihan, yaitu *data_latih*[], *t_latih*[], *data_uji*[], *t_uji*[], *hidden*, *fitur*, *maxIter*, *maxError*, *iterasi*, *mse*. Pada pelatihan, akan dilakukan inisialisasi bobot dan inisialisasi bias terlebih dahulu. Proses selanjutnya dilakukan selama belum mencapai *stop condition* (*mse*>*max_error* dan *iterasi*<*max_Iter*). Proses yang

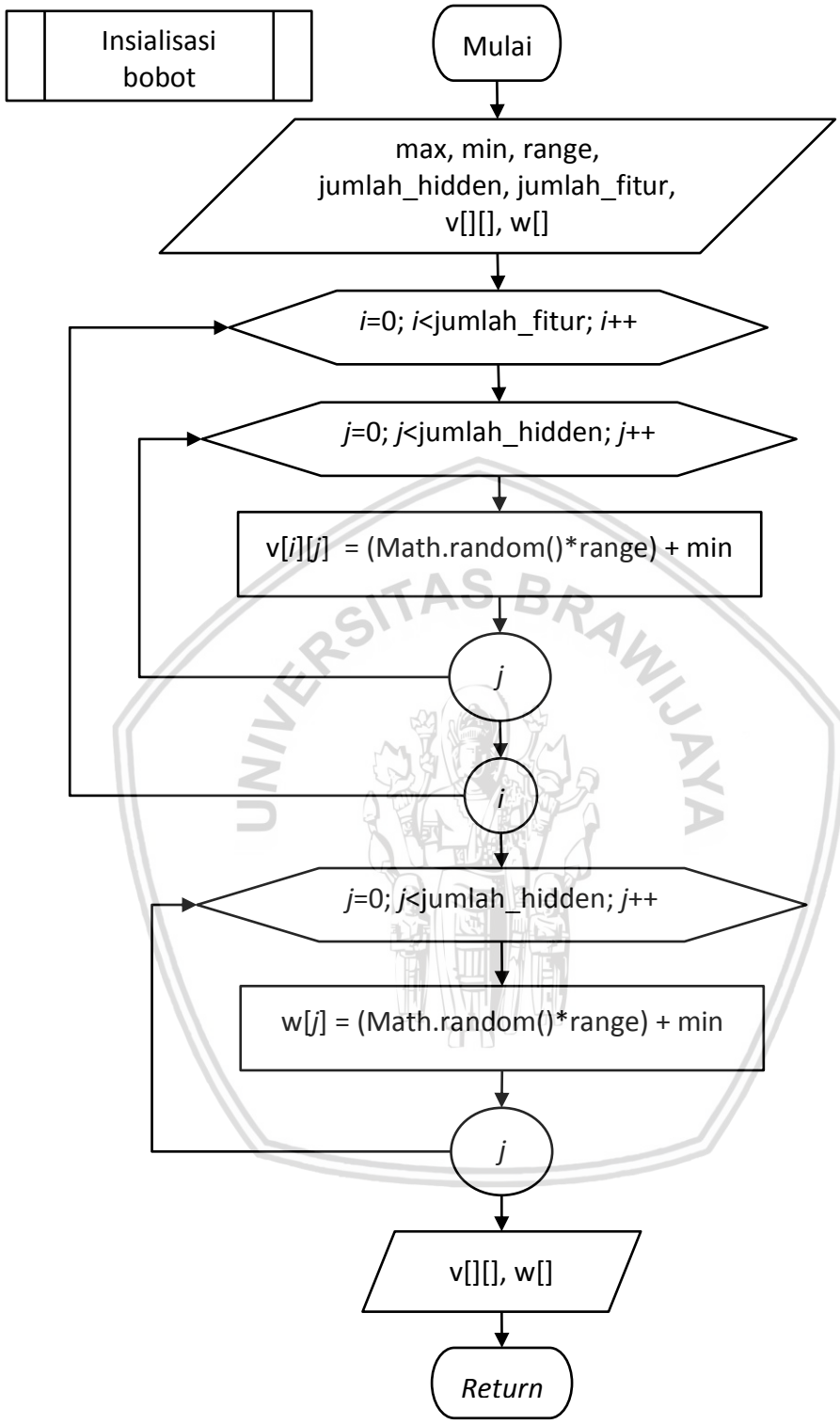
dilakukan adalah *feedforward*, *backpropagation* dan *update* bobot serta bias. Proses ini dilakukan sebanyak jumlah data latih dan *update* bobot serta bias dilakukan apabila belum mencapai data latih terakhir. Kemudian dihitung MSEnya.





Gambar 4.6 Diagram Alir Pelatihan

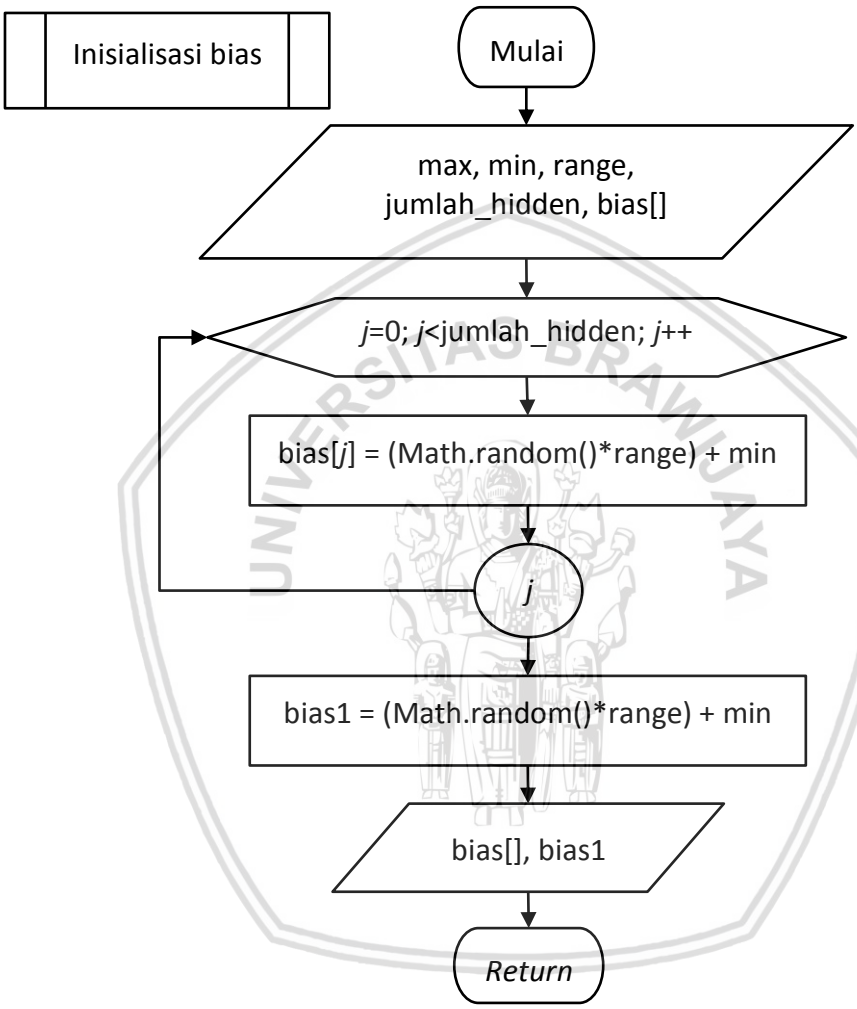
Bobot dan bias awal yang digunakan untuk pelatihan didapatkan dengan *random*. Perancangan untuk melakukan *random* bobot awal digambarkan dengan diagram alir pada Gambar 4.7. Inisialisasi bobot dilakukan untuk mendapatkan bobot antara *input layer* dengan *hidden layer* (*v*) dan bobot antara *hidden layer* denga *output layer* (*w*). Tentukan terlebih dahulu *max* (nilai maximal untuk bobot), *min* (nilai minimal untuk bobot), *range* (selisih nilai maximal dan nilai minimal), *jumlah_hidden*, *jumlah_fitur* dan panjang *v* serta *w* yang dapat diketahui berdasarkan *jumlah_hidden* dan *jumlah_fitur*. Untuk mendapatkan nilai *v* dilakukan perulangan selama *i* kurang dari *jumlah_ fitur* dan kondisi *j* kurang dari *jumlah_hidden*, nilai *v* pada elemen baris ke-*i* dan elemen kolom ke-*j* dihitung dengan (*Math.random*()**range*) + *min*. Untuk mendapatkan nilai *w* dilakukan perulangan selama *j* kurang dari *jumlah_ hidden*, nilai *w* ke-*j* dihitung dengan (*Math.random*()**range*) + *min*.



Gambar 4.7 Diagram Alir Inisialisasi Bobot

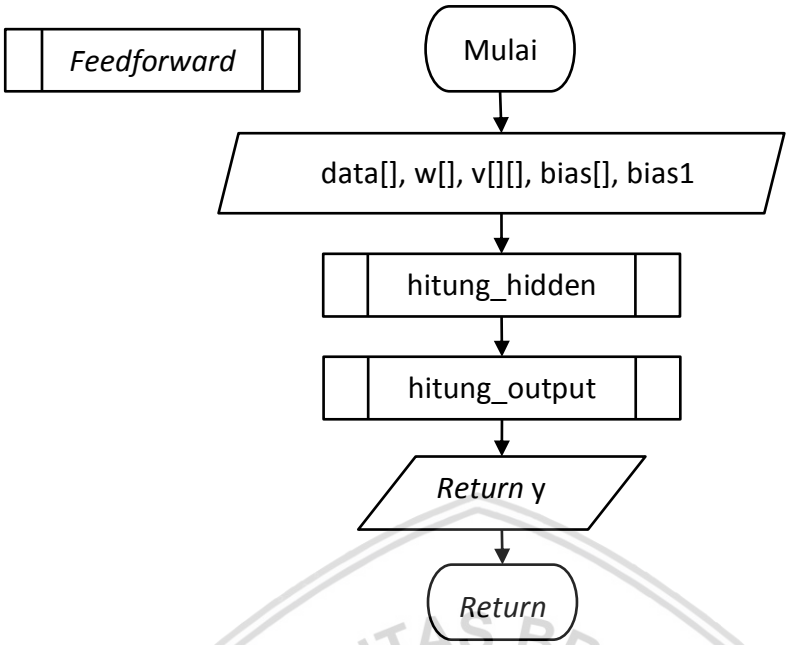
Perancangan untuk melakukan *random* bobot awal digambarkan dengan diagram alir pada Gambar 4.8. Inisialisasi bias dilakukan untuk mendapatkan bias pada *input layer* (*bias*) dan bias pada *hidden layer* (*bias1*). Tentukan terlebih

dahulu *max* (nilai maximal untuk bobot), *min* (nilai minimal untuk bobot), *range* (selisih nilai maximal dan nilai minimal), *jumlah_hidden* dan panjang *bias* yang dapat diketahui berdasarkan *jumlah_hidden*. Untuk mendapatkan nilai *bias* dilakukan perulangan selama *j* kurang dari *jumlah_ hidden*, nilai *bias* ke-*j* dihitung dengan (*Math.random*()**range*) + *min*. Untuk mendapatkan nilai *bias1* dihitung dengan (*Math.random*()**range*) + *min*.



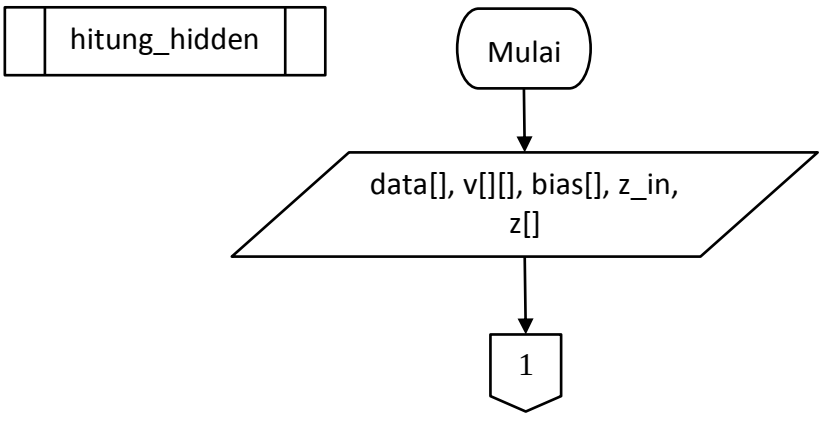
Gambar 4.8 Diagram Alir Inisialisasi Bias

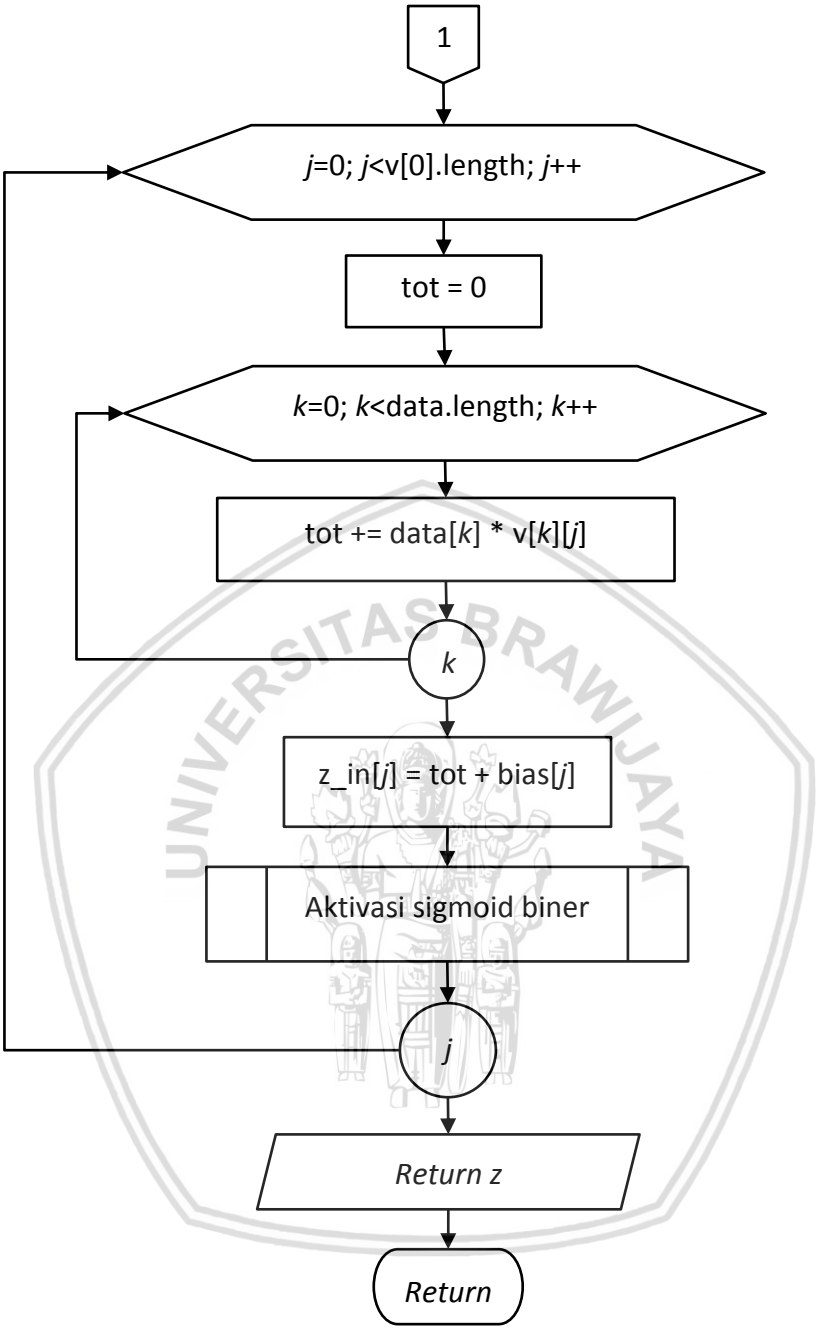
F*eedforward* merupakan proses awal dari algoritme *backpropagation*. Dari proses ini akan diketahui nilai setiap *hidden node* pada *hidden layer* dan nilai setiap *output node* pada *output layer*, serta nilai masukan dan keluaran. Perancangan *feedforward* dijelaskan dengan diagram alir pada Gambar 4.9. Variabel dan parameter yang diperlukan pada proses *feedforward* ini adalah *data*, *v, w bias* dan *bias1*. Pada proses *feedforward* ini akan dimulai dengan proses hitung *hidden* kemudian hitung *output*.



Gambar 4.9 Diagram Alir *Feedforward*

Fungsi hitung *hidden* adalah proses untuk mendapatkan nilai z_in dan z. Perancangan hitung *hidden* digambarkan dengan diagram alir pada Gambar 4.10. proses ini diawali dengan menentukan *data*, *v*, *bias* dan panjang *z_in* serta panjang *z*. Untuk mendapatkan nilai *z_in*, maka dilakukan perhitungan dengan Persamaan 2.1. Pada perancangan penelitian ini untuk mendapatkan nilai z_in dilakukan perulangan selama kondisi *j* kurang dari panjang elemen baris *v* dan nilai tot adalah 0, kemudian selama perulangan selama kondisi *k* kurang dari panjang *data* dilakukan penambahan nilai pada *tot* yaitu perkalian *data* ke-*k* dengan *v* pada elemen baris ke-*j* dan elemen kolom ke-*k*, setelah itu hitung nilai *z_in* ke-*j* dengan perhitungan nilai *tot* ditambah dengan nilai *bias* ke-*j*. Nilai *z_in* tersebut digunakan untuk perhitungan aktivasi sigmoid biner.

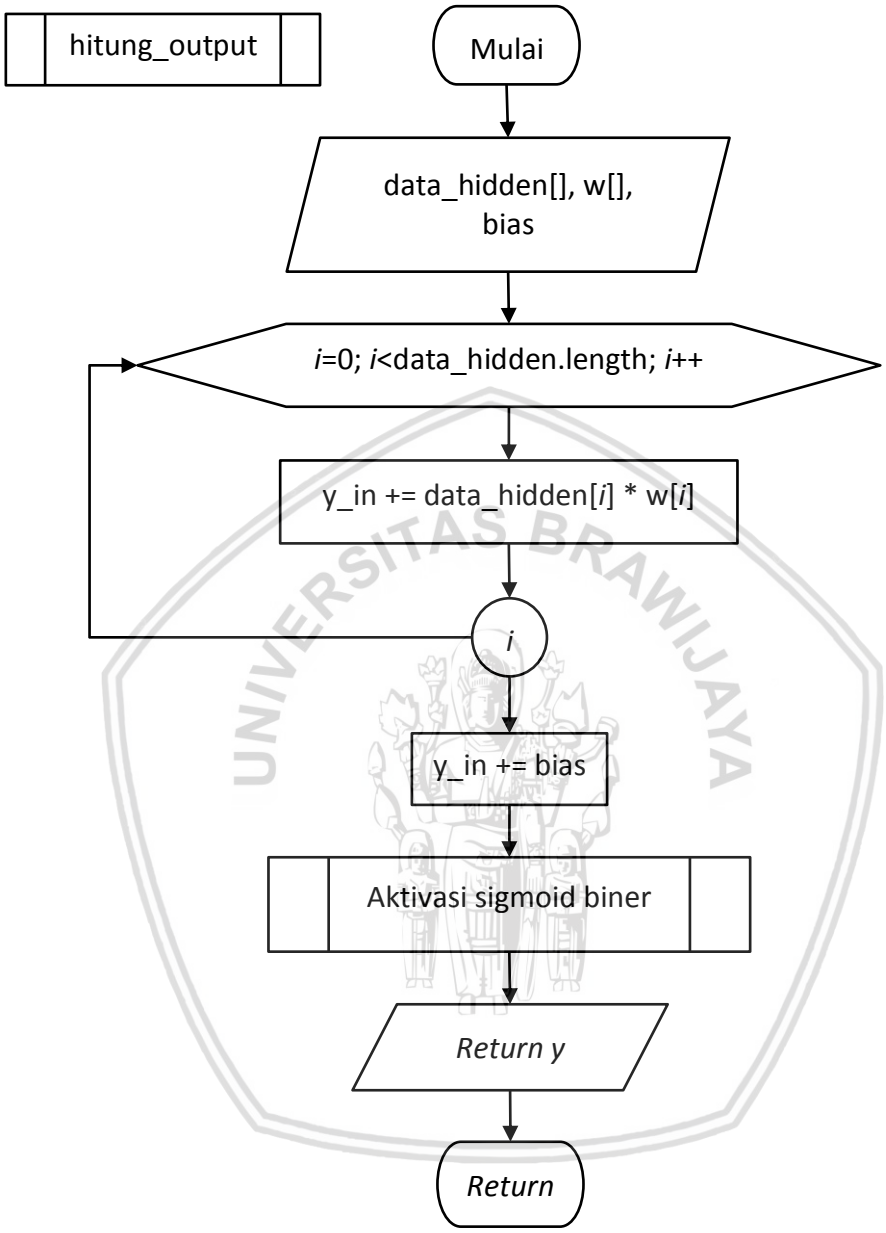




Gambar 4.10 Diagram Alir *Feedforward* pada *Hidden Layer*

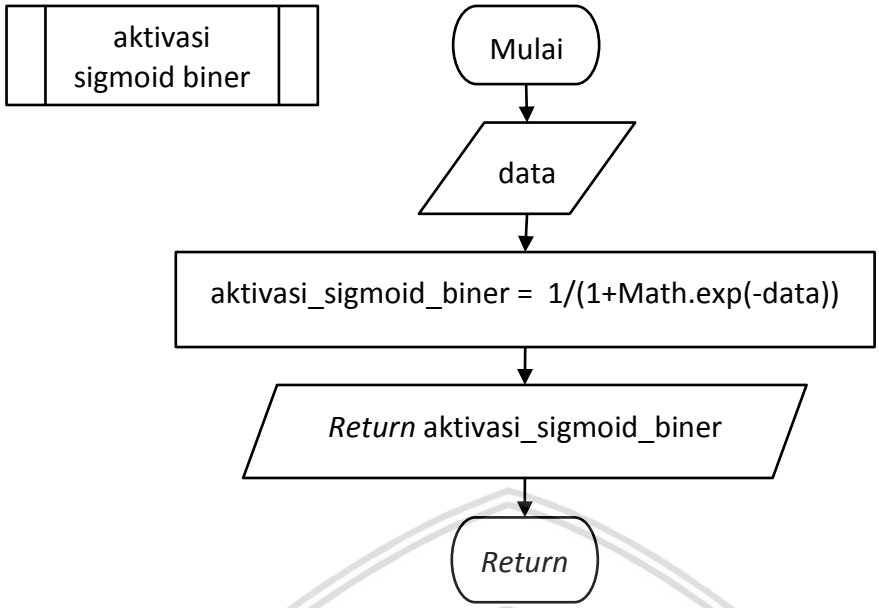
Fungsi hitung output adalah proses untuk mendapatkan nilai y_in dan y. Perancangan hitung *output* digambarkan dengan diagram alir pada Gambar 4.11. proses ini diawali dengan menentukan *data_hiddden*, *w* dan *bias1*. Untuk mendapatkan nilai *y_in*, maka dilakukan perhitungan dengan Persamaan 2.3. Pada perancangan penelitian ini untuk mendapatkan nilai *y_in* dilakukan perulangan selama kondisi *i* kurang dari panjang *data* dan nilai *y_in* ditambahkan dengan

perkalian antara *data_hidden* ke-*i* dan *w* ke-*i*. Kemudian *y_in* ditambahkan dengan. Nilai *y_in* tersebut digunakan untuk perhitungan aktivasi sigmoid biner.



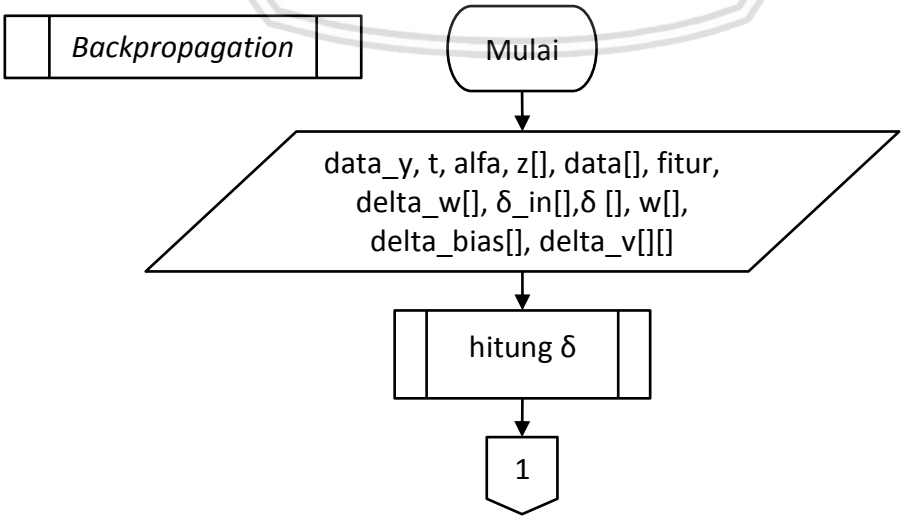
Gambar 4.11 Diagram Alir *Feedforward* pada *Output Layer*

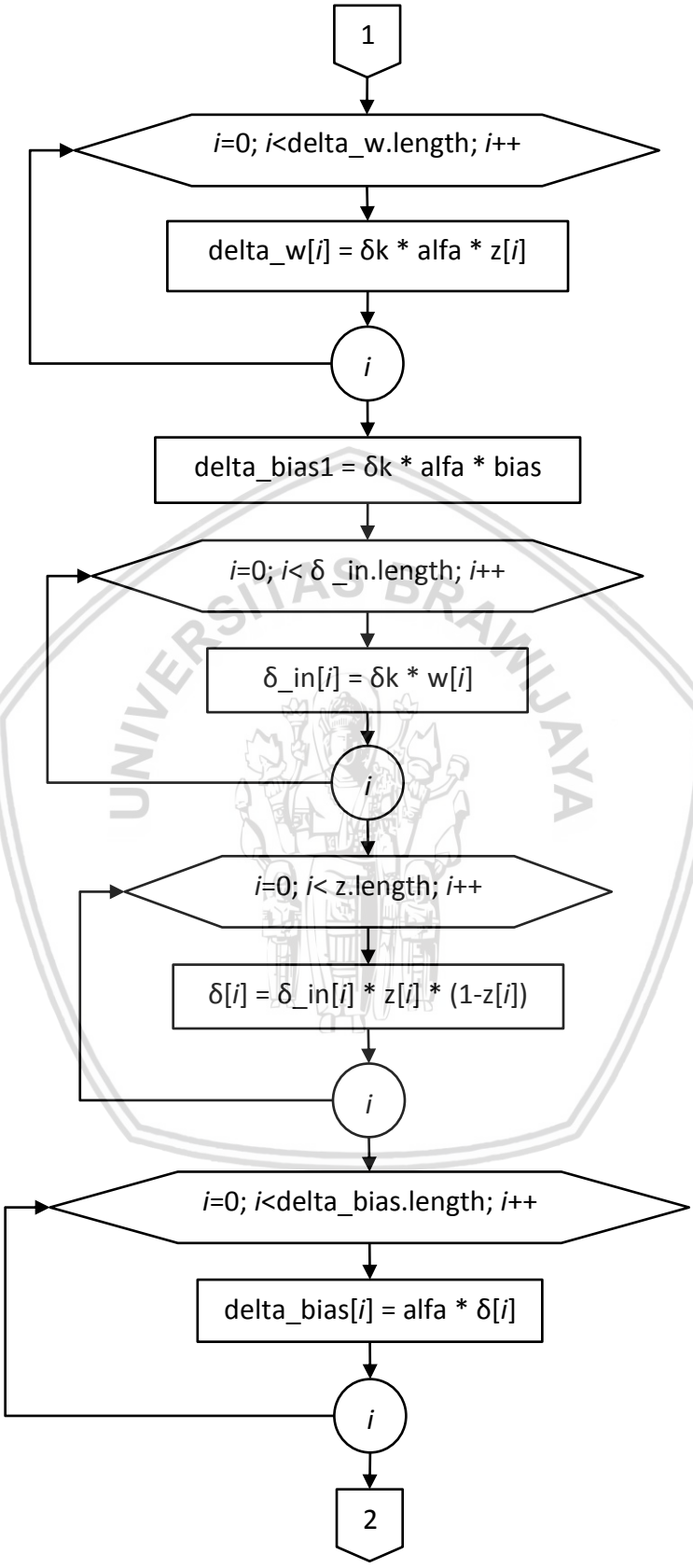
Proses aktivasi sigmoid biner diawali dengan menentukan *data* yang akan digunakan, yaitu data yang akan dihitung aktivasi sigmoid binernya. Kemudian dihitung dengan Persamaan 2.5. Perancangan aktivasi sigmoid biner digambarkan dengan diagram alir pada Gambar 4.12.

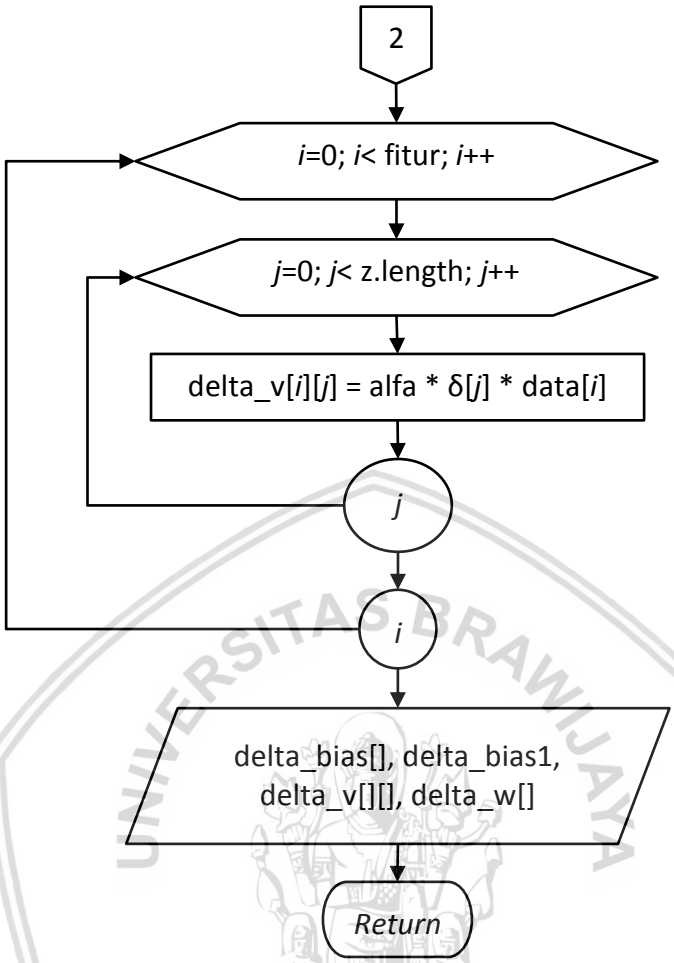


Gambar 4.12 Diagram Alir Fungsi Aktivasi *Sigmoid Biner*

Backpropagation dilakukan untuk mengkoreksi hasil perhitungan pada *feedforward* serta mengkoreksi bobot dan bias. Perancangan *backpropagation* digambarkan dengan diagram alir pada Gambar 4.13. *Backpropagation* dimulai dengan menentukan variabel dan parameter seperti pada diagram alir. Proses δ untuk menghitung δ_k. Hitung Δw_{jk} (*delta_w*) dengan Persamaan 2.11, perhitungan dilakukan hingga banyaknya delta_w. Hitung Δw_0 (*delta_bias1*) dengan Persamaan 2.12. Hitung δ_in$_j$ dengan Persamaan 2.13 hingga banyaknya δ_in$_j$, kemudian hitung δ_j dengan Persamaan 2.14 hingga banyaknya *z*. Hitung Δv_{oj} (*delta_bias*) dengan Persamaan 2.16 hingga banyaknya *delta_bias*. Hitung Δv_{ij} (*delta_v*) dengan Persamaan 2.15 hingga banyaknya fitur dan banyaknya *z*.

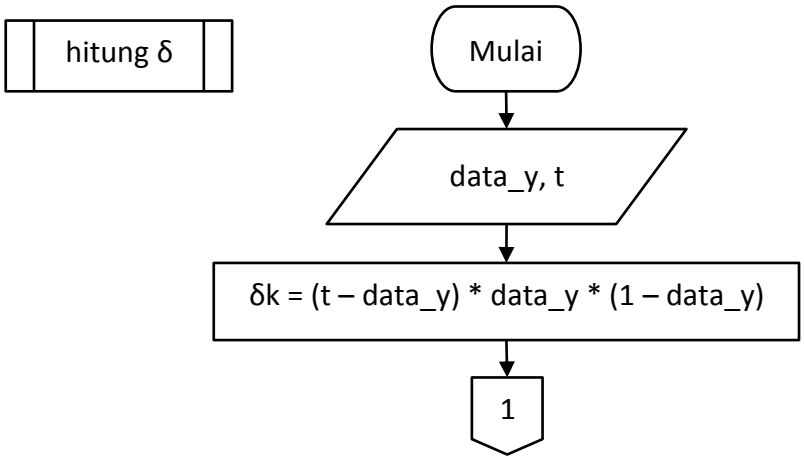


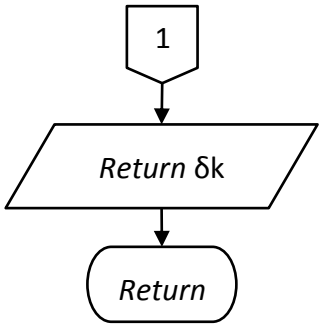

1
i=0; i<delta_w.length; i++
delta_w[i] = δk * alfa * z[i]
i
delta_bias1 = δk * alfa * bias
i=0; i< δ _in.length; i++
δ_in[i] = δk * w[i]
i
i=0; i< z.length; i++
δ[i] = δ_in[i] * z[i] * (1-z[i])
i
i=0; i<delta_bias.length; i++
delta_bias[i] = alfa * δ[i]
i
2



Gambar 4.13 Diagram Alir *Backpropagation*

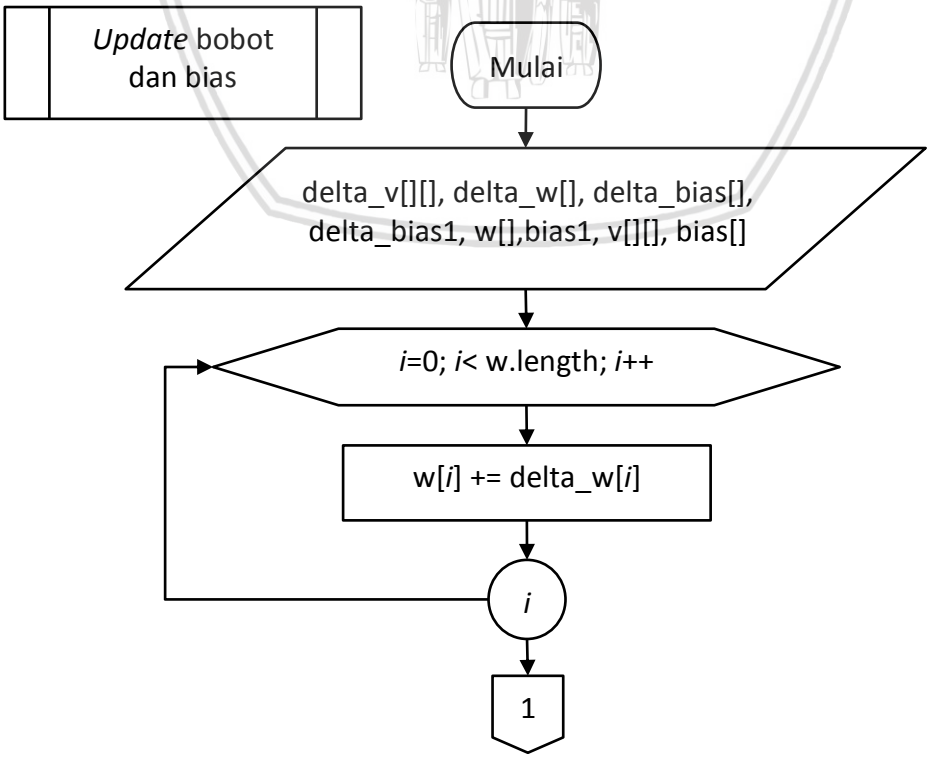
Proses hitung δ dilakukan untuk mendapatkan nilai δ_k. Tentukan data yang digunakan, yaitu target pada data *input*an (*t*) dan hasil nilai y (*data_y*), kemudian hitungan dengan Persamaan 2.10. Perancangan hitung δ digambarkan dengan diagram alir pada Gambar 4.14.

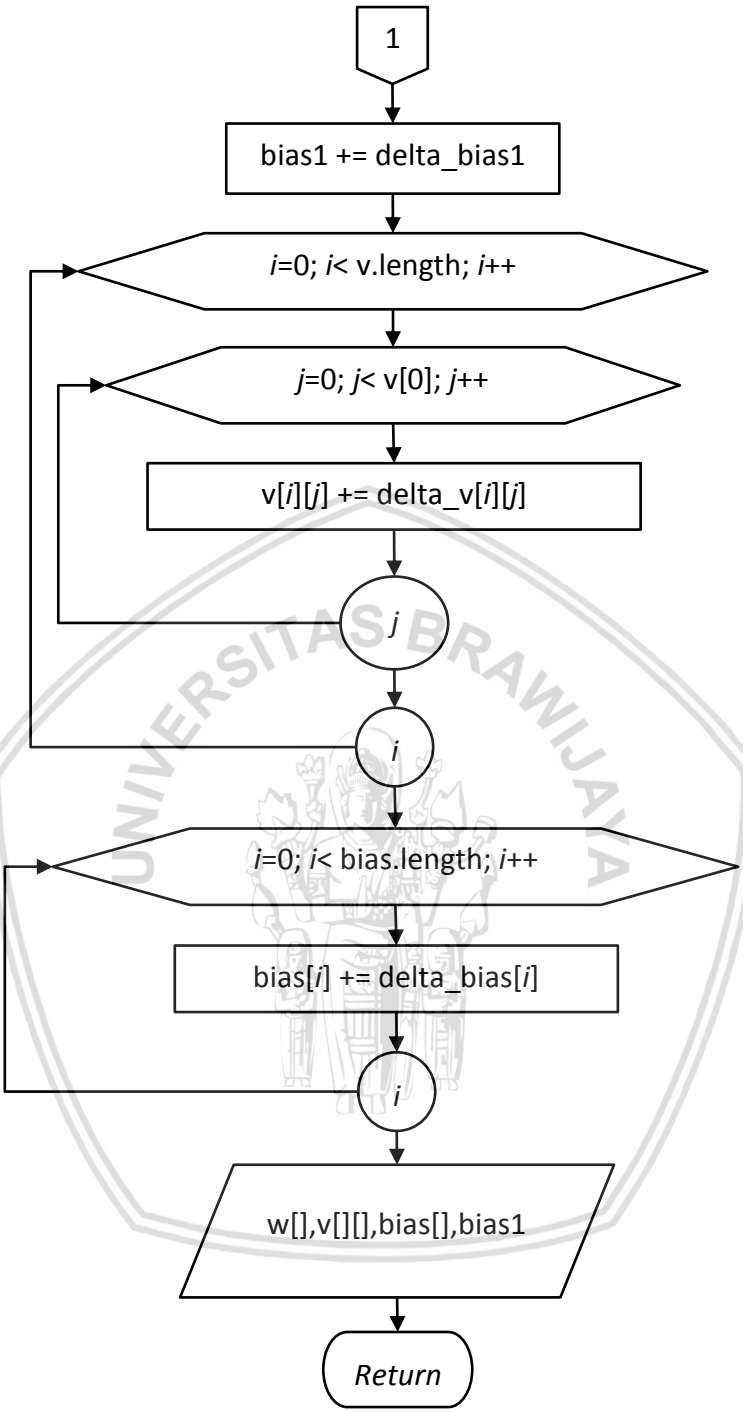




Gambar 4.14 Diagram Alir Koreksi Bobot W

Setelah menghitung *feedforward* dan *backpropagation* pada setiap data, dilakukan *update* bobot dan bias untuk perhitungan pada data selanjutnya. Perancangan *update* bobot dan bias digambarkan pada diagram alir pada Gambar 4.15. Tentukan *delta_v*, *delta_w*, *delta_bias* serta *delta_bias1* yang telah didapatkan dari proses *backpropagation* dan nilai *w*, *bias1*, *v*, *bias*. *Update* nilai *w* dengan Persamaan 2.17 yaitu menjumlahkan nilai *w* dengan *delta_w* hingga banyaknya *w*(perulangan dengan kondisi *i* kurang dari panjang *w*). *Update* nilai *bias1* dengan Persamaan 2.17 yaitu menjumlahkan nilai *bias1* dengan *delta_bias1*. *Update* nilai *v* dengan Persamaan 2.18 yaitu menjumlahkan nilai *v* dengan *delta_v* hingga banyaknya *v* dan elemen baris *v*(perulangan dengan kondisi *i* kurang dari panjang *v* dan kondisi *j* kurang dari panjang elemen baris *v*). *Update* nilai *bias* dengan Persamaan 2.19 yaitu menjumlahkan nilai *bias* dengan *delta_bias* hingga banyaknya *bias*(perulangan dengan kondisi *i* kurang dari panjang *bias*).

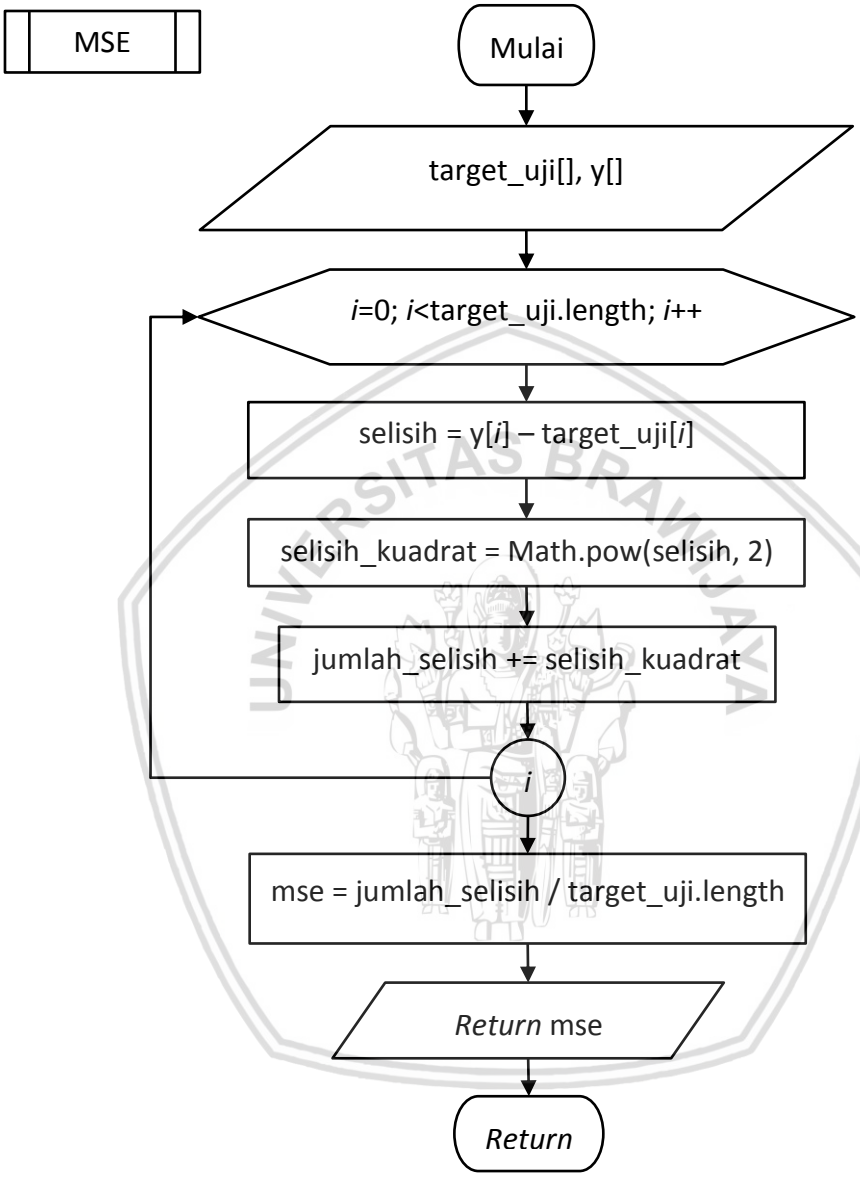




Gambar 4.15 Diagram Alir *Update* Bobot dan Bias

Pada setiap iterasi, setelah semua data dilakukan proses perhitungan sebelumnya maka dihitung *MSE*nya. Perhitungan *MSE* dilakukan untuk mengetahui seberapa baik perhitungan algoritme *backpropagation*, *MSE* ini juga digunakan untuk *stop condition* pada pelatihan. Perancangan *MSE* digambarkan dengan diagram alir pada Gambar 4.16. Tentukan *target_uji* dan *y* yang akan dihitung *MSE*nya. Selama kondisi *i* kurang dari panjang *target_uji* dilakukan

perulangan menghitung nilai selisih, *selisih_kuadrat* dan *jumlah*_selisih. Setelah mendapatkan nilai *jumlah_selisih*, maka nilai *MSE* dihitung dengan membagi nilai *jumlah_selisih* dengan panjang *target_uji*. Perhitungan *MSE* seperti pada Persamaan 2.21.

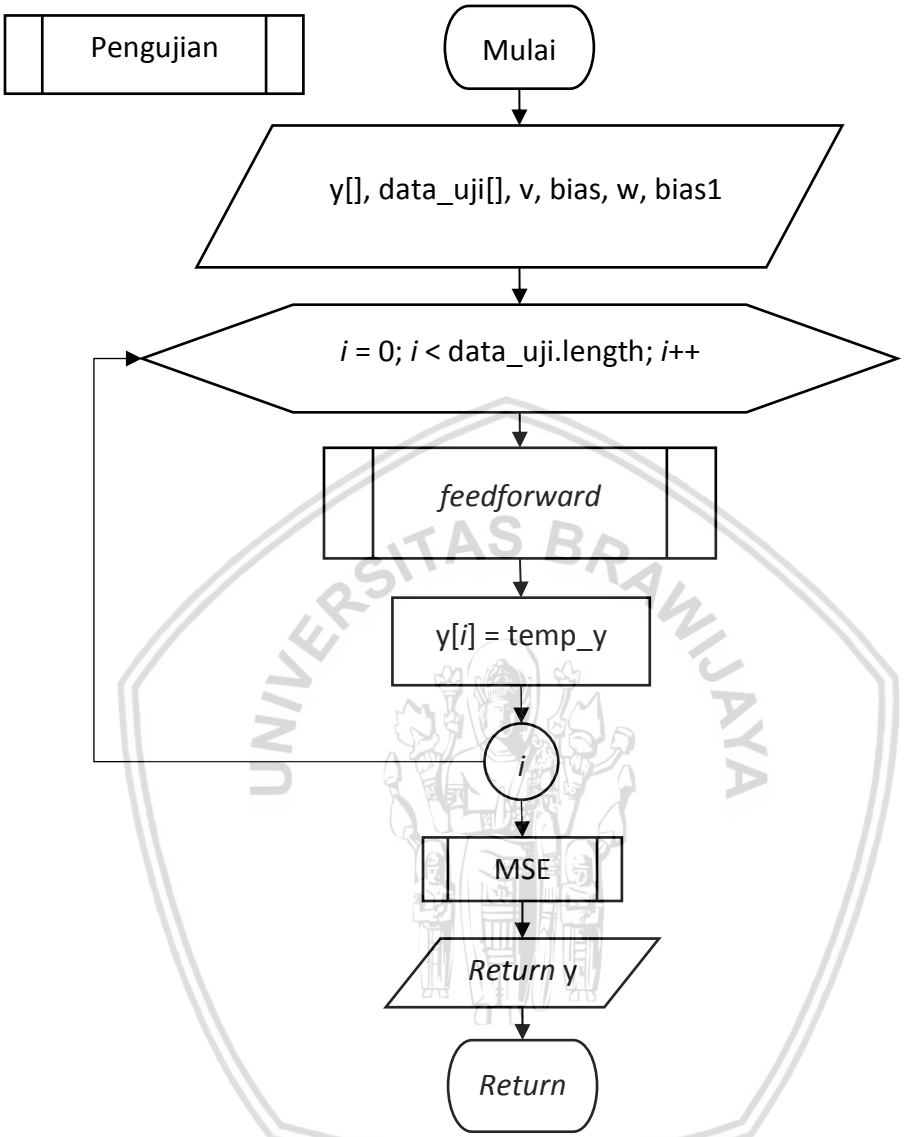


Gambar 4.16 Diagram Alir *MSE*

4.2.5 Pengujian

Setelah melakukan pelatihan, maka didapatkan bobot terbaik untuk mencapai hasil lebih optimal. Bobot tersebut digunakan sebagai bobot untuk pengujian. Pengujian ini akan mengetahui seberapa baik pelatihan beserta nilai-nilai pada variabel dan parameter yang digunakan untuk peramalan. Perancangan pengujian digambarkan dalam diagram alir pada Gambar 4.17. Tentukan *data_uji*, *v*, *bias*, *w*, *bias1* yang akan digunakan dan panjang *y* yang dapat diketahui berdasarkan *data_uji*. Kemudian melakukan proses *feedforward* hingga banyaknya *data_uji*

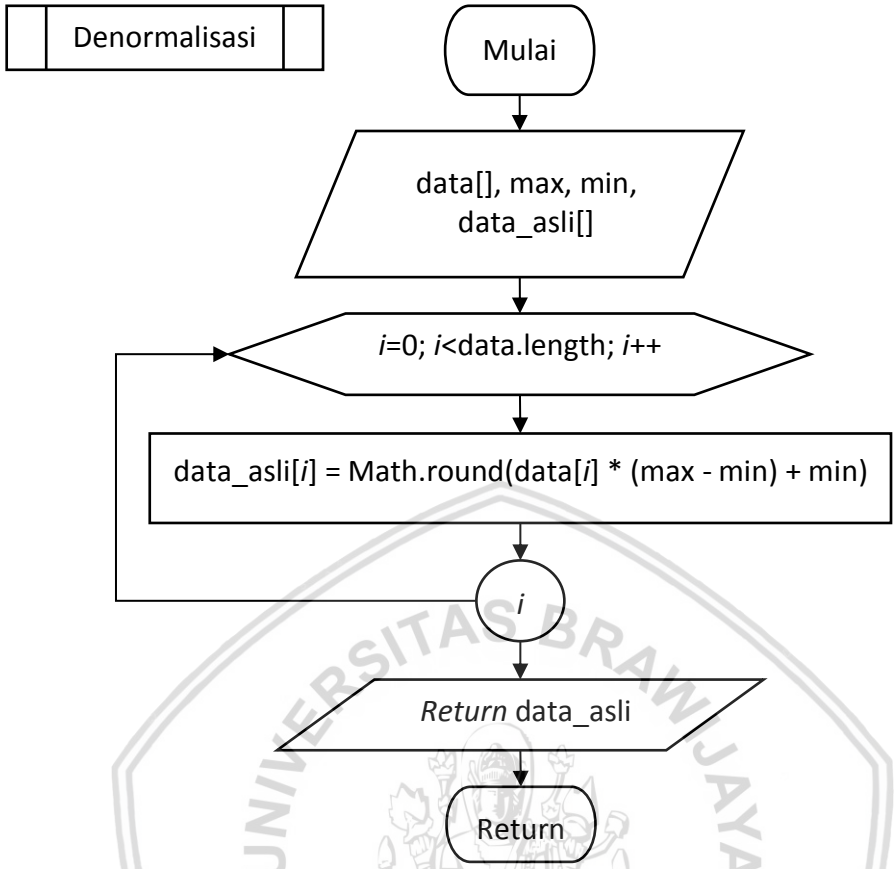
(perulangan selama kondisi *i* kurang dari panjang *data_uji*). Setelah itu, dihitung *MSE*nya dengan melakukan proses *MSE*.



Gambar 4.17 Diagram Alir Pengujian

4.2.6 Denormalisasi

Denormalisasi dilakukan untuk mengetahui nilai yang sebenarnya, karena data yang menjadi inputan sebelumnya dinormalisasi dengan *range* 0 hingga 1. Data yang didenormalisasi merupakan hasil peramalan data uji. Perancangan denormalisasi digambarkan dengan diagram alir pada Gambar 4.18. Ketika fungsi denormalisasi dijalankan, pertama yang dilakukan adalah menetukan *data*, nilai maximal pada dataset (*max*), nilai minimal pada dataset (*min*) dan banyaknya hasil denormalisasi (panjang *data_asli* yang dapat diketahui berdasarkan *data*). Kemudian menghitung nilai denormalisasi dengan Persamaan 2.20, yaitu nilai *data_asli* ke-*i* adalah hasil *Math.round* dikalikan dengan hasil perkalian *data* ke-*i* dan (*max* - *min*) + *min* yang dilakukan hingga banyaknya *data*.



Gambar 4.18 Diagram Alir Denormalisai

4.3 Contoh Perhitungan Manual

Setelah perancangan algoritme, dilakukan perhitungan manual untuk memperjelas alur algoritme. Pada contoh perhitungan manual dilakukan dengan data *spare part* oli *yamalube matic*. Jumlah node *input layer* yang digunakan adalah 2, jumlah node *hidden layer* adalah 3, nilai *learning rate* adalah 0.5, jumlah iterasi adalah 3 dan bobot awalnya *random*. Data pada contoh perhitungan manual merupakan sebagian data dari data *history* penjualan *spare part* pada salah satu *dealer* Yamaha Motor, yaitu Pangestu Utomo Motordi Candi, Sidoarjo. Dataset oli *yamalube matic* dapat dilihat pada Tabel 4.1. Data ke-1 hingga 3 digunakan sebagai data latih, sedangkan data ke 4 hingga 5 digunakan sebagai data uji. Langkah 3 hingga 6 dilakukan pada data latih terlebih dahulu hingga memenuhi *stop condition*. Jika perhitungan pada semua data latih (pelatihan) telah selesai, maka dilakukan perhitungan langkah 3 pada data uji (pengujian). Bobot yang digunakan pada pengujian merupakan hasil *update* bobot dan bias terakhir pada pelatihan (bobot hasil pelatihan).

Tabel 4.1 Dataset Oli *Yamalube Matic*

Data ke-	X1	X2	X3	Y
1	91	91	82	84

2	91	82	84	77
3	82	84	77	63
4	84	77	63	77
5	77	63	77	76

Langkah-langkah untuk melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan *algoritme backpropagation* adalah sebagai berikut:

Langkah 1: Normalisasi

Normalisasi data dilakukan agar nilai data mempunyai range 0 hingga 1. Normalisasi dihitung dengan Persamaan 2.19. Nilai maximal dan nilai minimal yang akan digunakan untuk perhitungan normalisasi didapatkan dari nilai maximal dan nilai minimal pada dataset. Nilai maximal pada dataset oli *yamalube matic* adalah 91 dan nilai minimalnya adalah 63. Hasil proses normalisasi dataset oli *yamalube matic* dapat dilihat pada Tabel 4.4.

$$x' = \frac{x - min}{\max - min}$$

Contoh perhitungan manual pada data ke-1:

Normalisasi fitur x1:

$x1' = \frac{91-63}{91-63}$ = 1

Normalisasi fitur x2:

$x2' = \frac{91-63}{91-63}$ = 1

Normalisasi fitur x3:

$x3' = \frac{82-63}{91-63}$ = 0.678571

Normalisasi fitur Y:

$Y' = \frac{84-63}{91-63}$ = 0.75

Tabel 4.2 Hasil Normalisasi Dataset Oli *Yamalube Matic*

Data ke-	X1	X2	X3	Y
1	1	1	0.678571	0.75
2	1	0.678571	0.75	0.5
3	0.678571	0.75	0.5	0
4	0.75	0.5	0	0.5
5	0.5	0	0.5	0.464286

Langkah 2: Inisialisasi bobot awal

Sebelum melakukan proses pelatihan algoritme *backpropagation*, dilakukan inisialisasi bobot awal yang nilainya didapatkan dengan *random*. Pada contoh perhitungan manual ini, bobot awal yang digunakan dapat dilihat pada Tabel 4.7.

Tabel 4.3 Insialisasi Bobot Awal

w11	w21	v11	v21	v31	v12	v22	v32
0.226622	0.220696	0.456789	0.234567	0.345678	0.234567	0.456789	0.345678

Tabel 4.4 Inisialisasi Bias Awal

w01	v01	v02
0.222222	0.123456	0.456654

Iterasi 1

Langkah 3: *Feedforward*

Fungsi aktivasi yang digunakan pada algoritme *backpropagation* ini adalah *binary sigmoid* yaitu pada Persamaan 2.5. Langkah-langkah pada proses *feedforward* adalah menghitung masukan untuk setiap node *hidden layer* dengan Persamaan 2.1, menghitung nilai aktivasi untuk setiap node *hidden layer*, menghitung masukan untuk setiap node *output layer* dengan Persamaan 2.3 dan menghitung nilai aktivasi untuk setiap node *output layer*. Hasil perhitungan proses *feedforward* pada iterasi 1 dan data ke-1 oli *yamalube matic* dapat dilihat pada Tabel 4.8.

Hitung masukan untuk setiap node *hidden layer*

$$z\_in_1 = v_{01} + ((x_1 v_{11}) + (x_2 v_{21}) + (x_3 v_{31}))$$

$$z\_in_1 = 0.123456 + ((1 \times 0.456789) + (1 \times 0.234567) + (0.678571 \times 0.345678))$$

$$z\_in_1 = 1.049379$$

$$z\_in_2 = v_{02} + ((x_1 v_{12}) + (x_2 v_{22}) + (x_3 v_{32}))$$

$$z\_in_2 = 0.456654 + ((1 \times 0.234567) + (1 \times 0.456789) + (0.678571 \times 0.345678))$$

$$z\_in_2 = 1.382577$$

Hitung nilai aktivasi setiap node *hidden layer*

$$z_1 = f(z\_in_1) = \frac{1}{1+\exp(-z\_in_1)}$$

$$z_1 = f(1.049379) = \frac{1}{1+ 2.71828183^{-1.049379}}$$

$$z_1 = 0.740656$$

$$z_2 = f(z\_in_2) = \frac{1}{1+\exp(-z\_in_2)}$$

$$z_2 = f(1.382577) = \frac{1}{1+ 2.71828183^{-1.382577}}$$

$$z_2 = 0.799405$$

Hitung masukan untuk setiap node *output layer*

$$y\_in_1 = w_{01} + ((z_1 w_{11}) + (z_2 w_{21}))$$

$$y\_in_1 = 0.222222 + ((0.740656 \times 0.226622) + (0.799405 \times 0.220696))$$

$$y\_in_1 = 0.566496$$

Hitung aktivasi untuk setiap node *output layer*

$$y_1 = f(y\_in_1) = \frac{1}{1+\exp(-y\_in_1)}$$

$$y_1 = f(0.566496) = \frac{1}{1+ 2.71828183^{-0.566496}}$$

$$y_1 = 0.637954$$

Tabel 4.5 Hasil *Feedforward* Oli *Yamalube Matic* Iterasi ke-1

Data ke-	Z_in1	Z_in2	z1	z2	Y_in1	y
1	1.049379	1.382577	0.740656	0.799405	0.566496	0.637954
2	1.000469	1.261903	0.731151	0.779354	0.587925	0.642889
3	0.781792	1.130853	0.686066	0.755996	0.537585	0.631251

Langkah 4: *Backpropagation*

Proses ini menghitung nilai *error* dan juga mengkoreksi bobor serta bias dari hasil yang didapatkan pada perhitungan *feedforward*. Awalnya menghitung koreksi *error* pada node *output* menggunakan Persamaan 2.10, dengan melakukan perhitungan menggunakan Persamaan 2.6 sebelumnya. Kemudian menghitung koreksi *error* pada node *hidden* menggunakan Persamaan 2.14, dengan terlebih dahulu melakukan perhitungan menggunakan Persamaan 2.6 dan Persamaan 2.13 sebelumnya. Setelah mengetahui hasil koreksi *error*, maka dilakukan koreksi bobot dan bias. Koreksi bobot dihitung menggunakan Persamaan 2.11 atau Persamaan 2.15. Koreksi bias dihitung menggunakan Persamaan 2.12 atau Persamaan 2.16. Hasil perhitungan proses *backpropagation* dapat dilihat pada Tabel 4.11 untuk hasil koreksi *error* dan Tabel 4.12 untuk hasil koreksi bobot dan bias.

Hitung $f(\text{y\_in})'$

$$f(y\_in_1)' = f(y\_in_1)\,[1 - f(y\_in_1)]$$

$$f(y\_in_1)' = 0.637954\ [1 - 0.637954\ ]$$

$$f(y\_in_1)' = 0.230969$$

Hitung koreksi *error* pada *output*

$$\delta_1 = (t_1 - y_1) f'(y\_in_1)$$

$\delta_1 = (\ 0.75\ -\ 0.638833)\ \times\ 0.230969$

$\delta_1 = 0.025879$

Hitung koreksi bobot dan bias

$\Delta w_{11} =\ \alpha\delta_1 z_1$

$\Delta w_{11} = 0.5\ \times\ 0.025879\ \times 0.740656$

$\Delta w_{11} =\ 0.0095837$

$\Delta w_{21} =\ \alpha\delta_1 z_2$

$\Delta w_{21} = 0.5\ \times\ 0.025879\ \times 0.799405$

$\Delta w_{21} =\ 0.0103439$

$\Delta w_{01} =\ \alpha\delta_1$

$\Delta w_{01} = 0.5\ \times\ 0.025879$

$\Delta w_{01} = 0.012939518$

Hitung δ_in

$\delta\_in_1 =\ \delta_1 w_{11}$

$\delta\_in_1 =\ 0.025879\ \times\ 0.226622$

$\delta\_in_1 =\ 0.005865$

$\delta\_in_2 =\ \delta_1 w_{21}$

$\delta\_in_2 =\ 0.025879\ \times\ 0.220696$

$\delta\_in_2 = 0.005711$

Hitung $f(\text{z\_in})'$

$f(z\_in_1)' =\ f(\text{z}\_in_1)\ [1 -\ f(z\_in_1)]$

$f(z\_in_1)' =\ 0.740656\ [1 -\ 0.740656]$

$f(z\_in_1)' =\ 0.192085$

$f(z\_in_2)' =\ f(z\_in_2)\ [1 -\ f(z\_in_2)]$

$f(z\_in_2)' =\ 0.799405\ [1 -\ 0.799405]$

$f(z\_in_2)' = 0.160357$

Hitung koreksi *error* pada *hidden*

$\delta_1 = \delta\_in_1\ f'(z\_in_1)$

$\delta_1 =\ 0.005865\ \times\ 0.192085$

$\delta_1 = 0.001126531$

$\delta_2 = \delta\_in_2\ f'(z\_in_2)$

$\delta_2 =\ 0.005711\ \times\ 0.160357$

$\delta_2 =\ 0.000915862$

Hitung koreksi bobot dan bias

$\Delta v_{11} = \alpha \delta_1 x_1$

$\Delta v_{11} = 0.5 \times 0.001126531 \times 1$

$\Delta v_{11} = 0.0005633$

$\Delta v_{21} = \alpha \delta_1 x_2$

$\Delta v_{21} = 0.5 \times 0.001126531 \times 1$

$\Delta v_{21} = 0.0005633$

$\Delta v_{31} = \alpha \delta_1 x_3$

$\Delta v_{31} = 0.5 \times 0.001126531 \times 0.678571$

$\Delta v_{31} = 0.0003822$

$\Delta v_{12} = \alpha \delta_2 x_1$

$\Delta v_{12} = 0.5 \times 0.000915862 \times 1$

$\Delta v_{12} = 0.0004579$

$\Delta v_{22} = \alpha \delta_2 x_2$

$\Delta v_{22} = 0.5 \times 0.000915862 \times 1$

$\Delta v_{22} = 0.0004579$

$\Delta v_{32} = \alpha \delta_2 x_3$

$\Delta v_{32} = 0.5 \times 0.000915862 \times 0.678571$

$\Delta v_{32} = 0.0003107$

$\Delta v_{01} = \alpha \delta_1$

$\Delta v_{01} = 0.5 \times 0.001126531$

$\Delta v_{01} = 0.00056327$

$\Delta v_{02} = \alpha \delta_2$

$\Delta v_{02} = 0.5 \times 0.000915862$

$\Delta v_{02} = 0.00045793$

Tabel 4.6 Hasil Koreksi *Error Backpropagation* Oli *Yamalube Matic* Iterasi ke-1

Data ke-	δ (y)	δ (z1)	δ (z2)
1	0.025879036	0.001126531	0.000915862
2	-0.032804842	-0.001523155	-0.001303336

Tabel 4.6 Hasil Koreksi *Error Backpropagation* Oli *Yamalube Matic* Iterasi ke-1

Data ke-	δ (y)	δ (z1)	δ (z2)
3	-0.146938267	-0.007095778	-0.005915868

Tabel 4.7 Hasil Koreksi Bobot *Backpropagation* Oli *Yamalube Matic* Iterasi ke-1

Data ke-	Δw11	Δw21	Δv11	Δv21	Δv31	Δv12	Δv22	Δv32
1	0.0095837	0.0103439	0.0005633	0.0005633	0.0003822	0.0004579	0.0004579	0.0003107
2	-0.011993	-0.012783	-0.000762	-0.000517	-0.000571	-0.000652	-0.000442	-0.000489
3	-0.050405	-0.055542	-0.002407	-0.002661	-0.001774	-0.002007	-0.002218	-0.001479

Tabel 4.8 Hasil Koreksi Bias *Backpropagation* Oli *Yamalube Matic* Iterasi ke-1

Data ke-	Δw01	Δv01	Δv02
1	0.01293952	0.00056327	0.00045793
2	-0.0164024	-0.0007616	-0.0006517
3	-0.0734691	-0.0035479	-0.0029579

Langkah 5: *Update* bobot dan bias

Bobot dan bias di*update* untuk perhitungan selanjutnya menggunakan bobot dan bias yang baru. *Update* bobot dan bias menggunakan Persamaan 2.17 dan Persamaan 2.18. Hasil *update* bobot dan bias dapat dilihat pada Tabel 4.17.

Hitung *update* bobot dan bias

$$w_{11}(new) = w_{11}(old) + \Delta w_{11}$$
$$w_{11}(new) = 0.222222 + 0.0095837$$
$$w_{11}(new) = 0.2362057$$
$$w_{21}(new) = w_{21}(old) + \Delta w_{21}$$
$$w_{21}(new) = 0.220696 + 0.0103439$$
$$w_{21}(new) = 0.2310399$$
$$w_{01}(new) = w_{01}(old) + \Delta w_{01}$$
$$w_{01}(new) = 0.222222 + 0.01293952$$
$$w_{01}(new) = 0.23516152$$
$$v_{11}(new) = v_{11}(old) + \Delta v_{11}$$
$$v_{11}(new) = 0.456789 + 0.0005633$$
$$v_{11}(new) = 0.4573523$$
$$v_{21}(new) = v_{21}(old) + \Delta v_{21}$$
$$v_{21}(new) = 0.234567 + 0.0005633$$
$$v_{21}(new) = 0.2351303$$
$$v_{31}(new) = v_{31}(old) + \Delta v_{31}$$
$$v_{31}(new) = 0.345678 + 0.0003822$$

$v_{31}(new) = 0.3460602$

$v_{12}(new) = v_{12}(old) + \Delta v_{12}$

$v_{12}(new) = 0.234567 + 0.0004579$

$v_{12}(new) = 0.2350249$

$v_{22}(new) = v_{22}(old) + \Delta v_{22}$

$v_{22}(new) = 0.456789 + 0.0004579$

$v_{22}(new) = 0.4572469$

$v_{32}(new) = v_{32}(old) + \Delta v_{32}$

$v_{32}(new) = 0.345678 + 0.0003107$

$v_{32}(new) = 0.3459887$

$v_{01}(new) = v_{01}(old) + \Delta v_{01}$

$v_{01}(new) = 0.123456 + 0.00056327$

$v_{01}(new) = 0.12401927$

$v_{02}(new) = v_{02}(old) + \Delta v_{02}$

$v_{02}(new) = 0.456654 + 0.00045793$

$v_{02}(new) = 0.45711193$

Tabel 4.9 Hasil *Update* Bobot Oli *Yamalube Matic* Iterasi ke-1

Data ke-	w11	w21	v11	v21	v31	v12	v22	v32
1	0.2362057	0.2310399	0.4573523	0.2351303	0.3460602	0.2350249	0.4572469	0.3459887
2	0.2242131	0.2182566	0.4565907	0.2346135	0.345489	0.2343733	0.4568047	0.3455

Tabel 4.10 Hasil *Update* Bias Oli *Yamalube Matic* Iterasi ke-1

Data ke-	w01	v01	v02
1	0.23516152	0.12401927	0.45711193
2	0.2187591	0.12325769	0.45646026

Langkah 6: *Mean Squared Error*

Mean squared error (*MSE*) digunakan untuk mengukur tingkat akurasi hasil peramalan. *MSE* juga digunakan untuk mengetahui *stop condition*. Selama *MSE*>maximal *error* yang ditentukan dan *stop condition* yang lain belum terpenuhi, maka perhitungan pelatihan *backpropagation* (*feedforward*, *backpropagation* dan *update* bobot serta bias) terus dilakukan. Perhitungan *MSE* dihitung di setiap iterasi dengan Persamaan 2.21. Untuk perhitungan *MSE*, perhitungan pelatihan *backpropagation* harus diselesaikan hingga semua data latih terlebih dahulu. Hasil perhitungan pelatihan *backpropagation* dapat dilihat

pada Tabel 2.20. Hasil perhitungan *MSE* pada semua iterasi dapat dilihat pada Tabel 4.23.

Tabel 4.11 Hasil Pelatihan *Backpropagation* Oli *Yamalube Matic* Iterasi ke-1

Data ke-	X1	X2	X3	Y	y
1	1	1	0.678571	0.75	0.637954
2	1	0.678571	0.75	0.5	0.642889
3	0.678571	0.75	0.5	0	0.631251

$$MSE = \frac{\sum_{i=1}^{n} |Y_i - y_i|^2}{n}$$

Hitung *MSE* pada iterasi ke-1

$$MSE = \frac{|Y_1 - y_1|^2 + |Y_2 - y_2|^2 + |Y_3 - y_3|^2}{3}$$

$$MSE = \frac{|0.75 - 0.637954|^2 + |0.5 - 0.642889|^2 + |0 - 0.631251|^2}{3}$$

$$MSE = 0.143816236$$

Tabel 4.12 Hasil *MSE* Pelatihan Oli *Yamalube Matic*

Iterasi ke-	*MSE*
1	0.143816236
2	0.143185313
3	0.142629673

Langkah 7: Denormalisasi

Setelah semua proses atau langkah-langkah perhitungan telah dilakukan, maka dilakukan denormalisasi data pada hasil peramalan untuk mengetahui nilai aslinya agar dapat dibandingkan dengan nilai aktualnya. Denormalisasi dilakukan dengan Persamaan 2.20. Nilai pada denormalisasi merupakan bilangan bulat, maka hasil perhitungan akan dibulatkan. Data yang akan digunakan pada contoh perhitungan denormalisasi adalah hasil peramalan data uji pada Tabel 2.26. Nilai maximal dan minimal yang digunakan sama seperti yang digunakan pada proses normalisasi yaitu nilai maximal dan minimal pada dataset. Nilai maximal pada dataset oli *yamalube matic* adalah 91 dan nilai minimalnya adalah 63. Hasil proses denormalisasi dataset oli *yamalube matic* dapat dilihat pada Tabel 4.26.

$$y = y' (\max - \min) + min$$

Hitung denormalisasi hasil peramalan (y) pada data ke 4

$$y = y' (\max - \min) + min$$

$$y = 0.623558 (91 - 63) + 63$$

$$y = 80$$

Tabel 4.13 Hasil Denormalisasi Oli *Yamalube Matic*

Data ke-	Hasil Peramalan (y)	Denormalisasi
4	0.623558	80
5	0.621624	80

Perbedaan hasil peramalan dengan nilai aktual dapat dilihat pada Tabel 4.14. Pada Tabel 4.14 juga dapat diketahui selisih dan rata-rata selisih antara hasil peramalan dan nilai aktual.

Tabel 4.14 Perbedaan Nilai Aktual dan Hasil Peramalan

Data ke-	Jumlah Aktual	Hasil Peramalan	Selisih	Rata-Rata Selisih
4	77	80	3	3.5
5	76	80	4	

4.4 Perancangan Pengujian

Pengujian dilakukan untuk mengetahui variabel-variabel terbaik agar menghasilkan hasil yang paling optimal. Pengujian yang akan dilakukan pada penelitian ini adalah pengujian nilai *learning rate*, pengujian jumlah node *input layer*, pengujian jumlah node *hidden layer* dan pengujian jumlah iterasi.

4.4.1 Pengujian Jumlah Iterasi

Pengujian jumlah iterasi dilakukan untuk mendapatkan jumlah iterasi yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Iterasi yang akan diujikan adalah 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500 dan 5000. Jumlah node *hidden layer*, jumlah node *input layer* dan *learning rate* yang akan digunakan pada setiap pengujian sama. Bobot dan bias awal adalah *random*, maka pengujian akan dilakukan dengan menjalankan program sebanyak 5 kali untuk setiap iterasi yang diujikan, hasil *MSE* setiap pengujian akan dicatat dan dihitung rata-rata *MSE*-nya sehingga diketahui jumlah iterasi yang menghasilkan hasil paling optimal. Perancangan pengujian jumlah iterasi dapat dilihat pada Tabel 4.15.

Tabel 4.15 Perancangan Pengujian Jumlah Iterasi

Iterasi	*MSE*					Rata-Rata *MSE*
	Percobaan ke-					
	1	2	3	4	5	
500						

1000						
1500						
2000						
2500						
3000						
3500						
4000						
4500						
5000						

4.4.2 Pengujian Jumlah Node *Input Layer*

Pengujian jumlah node *input layer* dilakukan untuk mendapatkan jumlah node *input layer* yang yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Jumlah node *input layer* yang akan diujikan adalah 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 dan 22. Nilai *Learning rate*, jumlah node *input layer* dan iterasi yang akan digunakan pada setiap pengujian sama. Bobot dan bias awal adalah *random*, maka pengujian akan dilakukan dengan menjalankan program sebanyak 5 kali untuk setiap jumlah node *input layer* yang diujikan, hasil *MSE* setiap pengujian akan dicatat dan dihitung rata-rata *MSE*-nya sehingga diketahui jumlah node *input layer* yang menghasilkan hasil paling optimal. Perancangan pengujian jumlah node *input layer* dapat dilihat pada Tabel 4.16.

Tabel 4.16 Perancangan Pengujian Jumlah Node *Input Layer*

Jumlah node i*nput layer*	MSE					Rata-Rata MSE
	Percobaan ke-					
	1	2	3	4	5	
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						

12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

4.4.3 Pengujian Jumlah Node *Hidden Layer*

Pengujian jumlah node *input layer* dilakukan untuk mendapatkan jumlah node *hidden layer* yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Jumlah node *hidden layer* yang akan diujikan adalah 1, 2, 3, 4, 5, 6, 7, 8, 9 dan 10. *Leaning rate*, jumlah node *hidden layer* dan iterasi yang akan digunakan pada setiap pengujian sama. Bobot dan bias awal adalah *random*, maka pengujian akan dilakukan dengan menjalankan program sebanyak 5 kali untuk setiap jumlah node *hidden layer* yang diujikan, hasil *MSE* setiap pengujian akan dicatat dan dihitung rata-rata *MSE*-nya sehingga diketahui jumlah node *hidden layer* yang menghasilkan hasil paling optimal.Perancangan pengujian jumlah node *hidden layer* dapat dilihat pada Tabel 4.17.

Tabel 4.17 Perancangan Pengujian Jumlah Node *Hidden Layer*

Jumlah node *hidden layer*	*MSE*					Rata-Rata *MSE*
	Percobaan ke-					
	1	2	3	4	5	
1						
2						
3						
4						
5						
6						
7						
8						
9						

10						

4.4.4 Pengujian Nilai *Learning Rate* (α)

Pengujian nilai *learning rate* dilakukan untuk mendapatkan nilai *learning rate* yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Nilai *learning rate* yang akan diujikan adalah 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4 dan 0.5. Jumlah node *hidden layer*, jumlah node *input layer* dan iterasi yang akan digunakan pada setiap pengujian sama. Bobot dan bias awal adalah *random*, maka pengujian akan dilakukan dengan menjalankan program sebanyak 5 kali untuk setiap nilai *learning rate* yang diujikan, hasil *MSE* setiap pengujian akan dicatat dan dihitung rata-rata *MSE*-nya sehingga diketahui nilai *learning rate* yang menghasilkan hasil paling optimal.Perancangan pengujian nilai *learning rate* dapat dilihat pada Tabel 4.18.

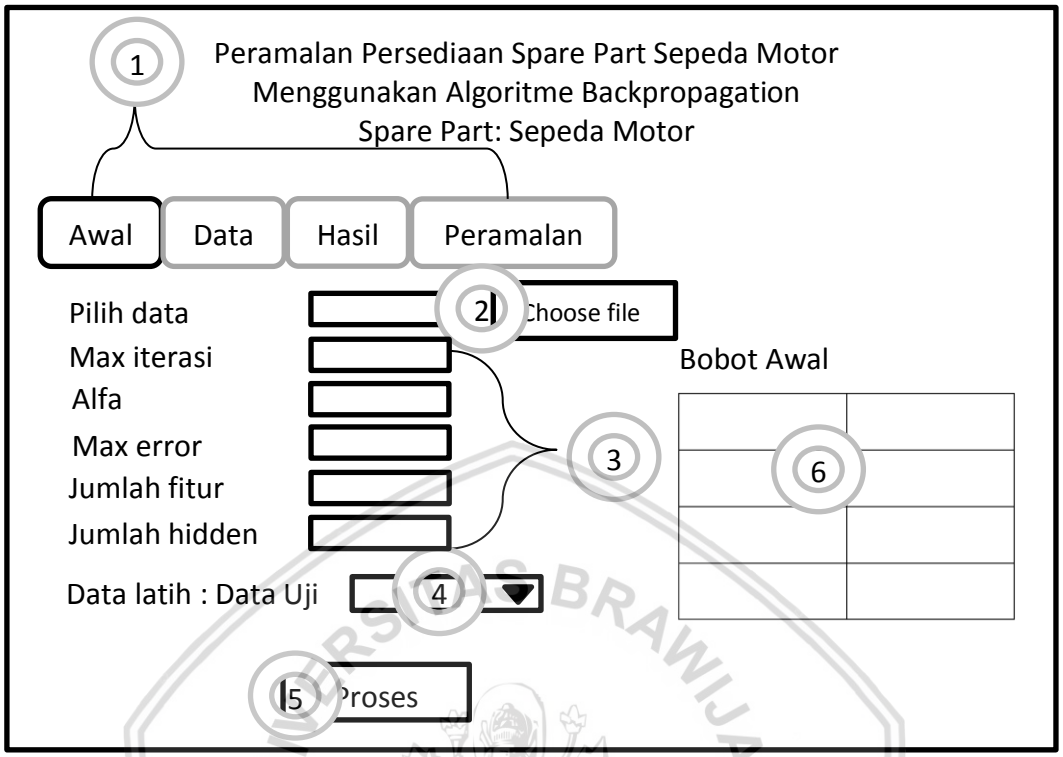
Tabel 4.18 Perancangan Pengujian Nilai *Learning Rate*

Learning Rate	*MSE*					Rata-Rata *MSE*
	Percobaan ke-					
	1	2	3	4	5	
0.05						
0.06						
0.07						
0.08						
0.09						
0.1						
0.2						
0.3						
0.4						
0.5						

4.5 Perancangan Antarmuka

Perancangan antarmuka merupakan gambaran dari tampilan sistem peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* yang nantinya dapat mempermudah implementasi antarmuka sistem.

4.5.1 Tampilan Awal

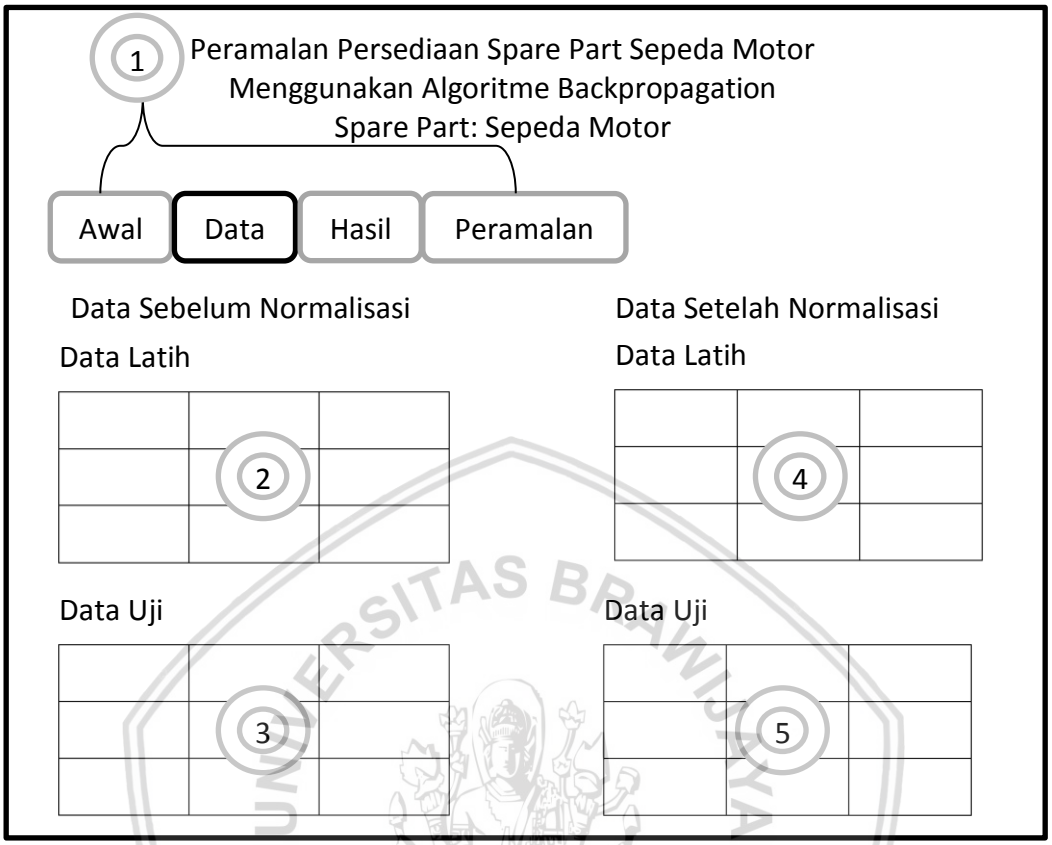


Gambar 4.19 Perancangan Antarmuka Tampilan Awal

Perancangan antarmuka tampilan awal dapat dilihat pada Gambar 4.9. Penjelasan dari perancangan antarmuka tersebut adalah sebagai berikut:

1. Menu untuk memilih yang ingin ditampilkan. Tampilan awal ketika program di*run* adalah menampilkan menu Awal. Dimana menu awal ini berfungsi untuk memasukkan nilai-nilai variabel.
2. *Input* file yang berisi data untuk melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
3. *Text area* yang dapat diisi *user* untuk masukan pada program guna melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
4. *Combo box* memiliki beberapa pilihan yang ditentukan untuk masukan pada program guna melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
5. *Button* untuk memproses atau mulai melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
6. Tabel yang akan menampilkan bobot awal setelah perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* diproses.

4.5.2 Data

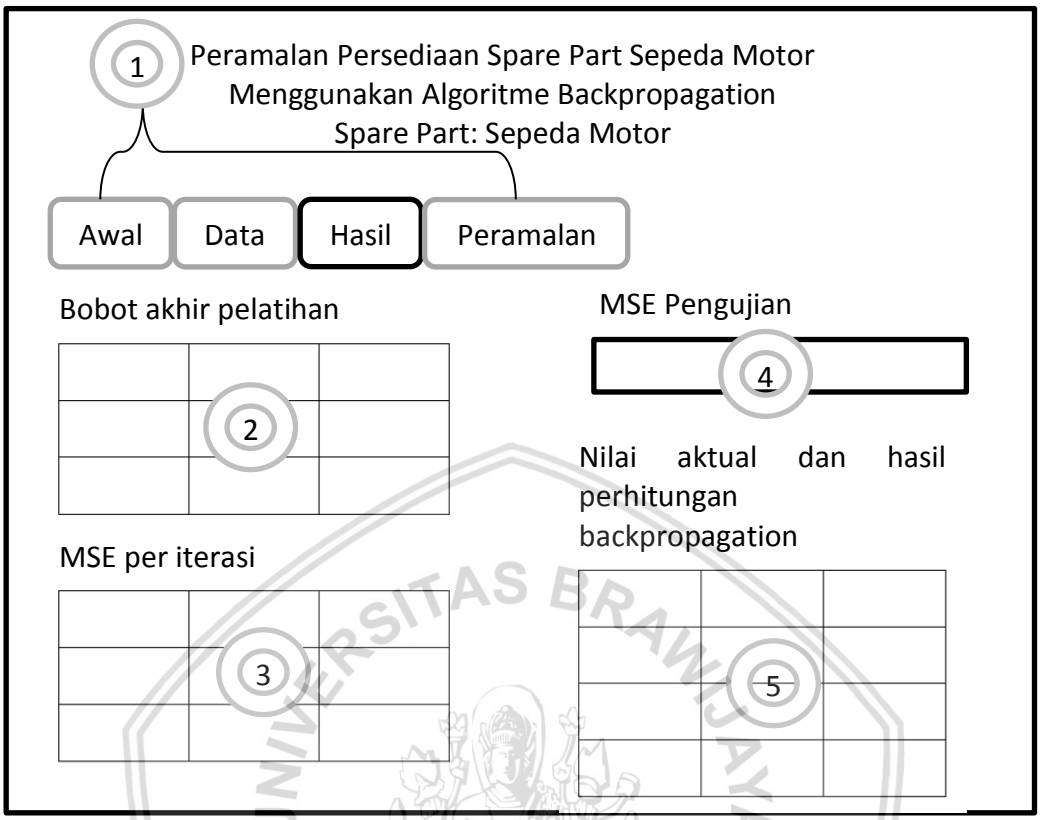


Gambar 4.20 Perancangan Antarmuka Data

Perancangan antarmuka data dapat dilihat pada Gambar 4.10. Penjelasan dari perancangan antarmuka tersebut adalah sebagai berikut:

1. Menu untuk memilih yang ingin ditampilkan. Menu Data ini berfungsi untuk menampilkan data setelah perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* diproses.
2. Tabel untuk menampilkan data latih yang telah dipilih dan proses pada menu Awal.
3. Tabel untuk menampilkan data uji yang telah dipilih dan proses pada menu Awal.
4. Tabel untuk menampilkan data latih yang telah dinormalisasi. Data ini yang digunakan untuk pelatihan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
5. Tabel untuk menampilkan data uji yang telah dinormalisasi. Data ini yang digunakan untuk pengujian pada perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.

4.5.3 Hasil

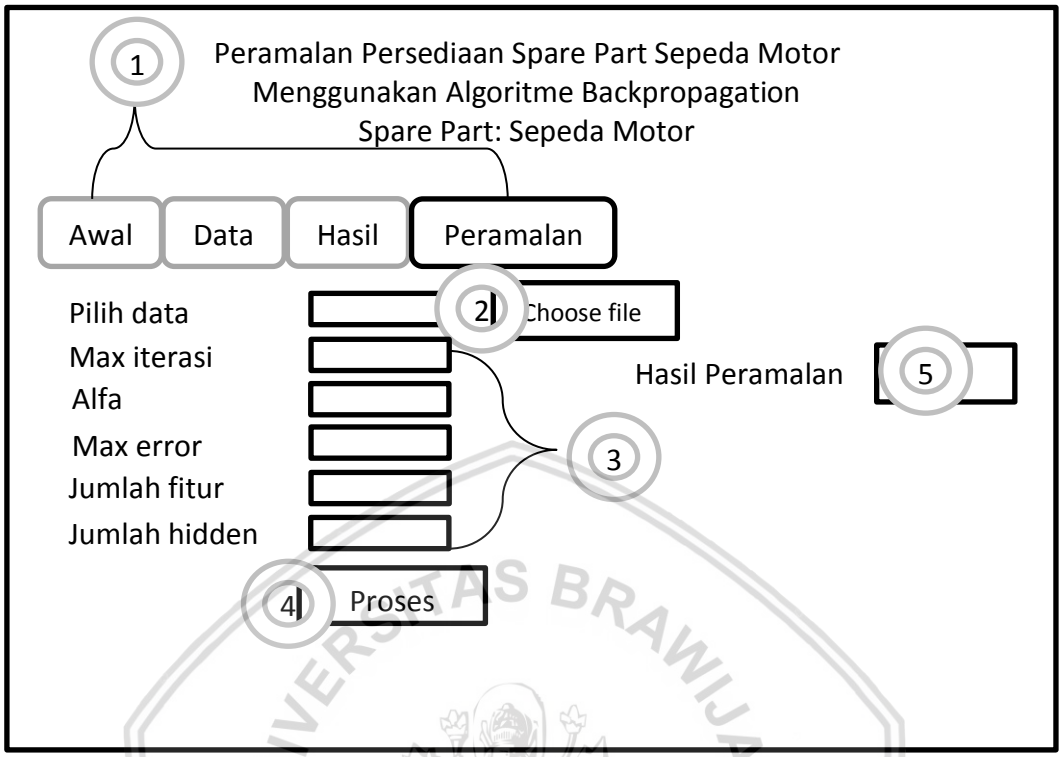


Gambar 4.21 Perancangan Antarmuka Hasil

Perancangan antarmuka hasil dapat dilihat pada Gambar 4.10. Penjelasan dari perancangan antarmuka tersebut adalah sebagai berikut:

1. Menu untuk memilih yang ingin ditampilkan. Menu Hasil ini berfungsi untuk menampilkan hasil setelah perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* diproses.
2. Tabel untuk menampilkan bobot akhir pada proses pelatihan.
3. Tabel untuk menampilkan *MSE* setiap iterasi pada proses pelatihan.
4. *Text area* untuk menampilkan *MSE* pengujian.
5. Tabel untuk menampilkan data nilai target aktual dan nilai hasil peramalan.

4.5.4 Peramalan



Gambar 4.22 Perancangan Antarmuka Peramalan

Perancangan antarmuka peramalan dapat dilihat pada Gambar 4.12. Penjelasan dari perancangan antarmuka tersebut adalah sebagai berikut:

1. Menu untuk memilih yang ingin ditampilkan. Menu peramalan ini berfungsi untuk melakukan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* bulan berikutnya. Pada menu ini terdapat tempat memasukkan nilai-nilai variabel dan menampilkan hasilnya.
2. *Input* file yang berisi data untuk melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
3. *Text area* yang dapat diisi *user* untuk masukan pada program guna melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
4. *Button* untuk memproses atau mulai melakukan perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation*.
5. *Text area* yang akan menampilkan hasil peramalan setelah perhitungan peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* diproses.

BAB 5 IMPLEMENTASI

5.1 Implementasi Peramalan Persediaan *Spare Part*

Implementasi peramalan persediaan *spare part* sepeda motor menggunakan algoritme *backpropagation* pada penelitian ini menggunakan bahasa java dan *netbeans* sebagai IDE. Pada subbab ini dijelaskan kode programnya. Terdapat kelas algoritma untuk kode program implementasi algoritmenya beserta fungsi pendukung lainnya. Algoritme berserta fungsi pendukung yang sudah dirancang pada bab perancangan dijelaskan pada tabel kode program tiap subbab implementasi berikut.

5.1.1 Normalisasi

Fungsi pendukung dan penyusun normalisasi berada pada kelas algoritma. Implementasi normalisasi sesuai pada perancangan yang telah dilakukan. Kode program normalisasi dapat dilihat pada Tabel 5.1 yang disesuaikan dengan perancangan pada Gambar 4.2.

Tabel 5.1 Kode Program Normalisasi

No	Kode Program
1	`double[] normalisasi(double[] data) {`
2	`    double max = this.getMax();`
3	`    double min = this.getMin();`
4	`    double data_normalisasi[] = new double[data.length];`
5	`    for (int i = 0; i < data.length; i++)   {`
6	`      data_normalisasi[i] = (data[i] - min) / (max - min);`
7	`    }`
8	`    return data_normalisasi;`
9	`}`

Penjelasan kode program pada Tabel 5.1.

Baris 1 : Deklarasi fungsi *normalisasi* dengan parameter data yang merupakan array bertipe double. Fungsi ini digunakan untuk mengubah nilai pada *data* menjadi nilai yang memiliki *range* 0 hingga 1.

Baris 2-3 : Deklarasi variabel *max* dan *min* yang betipe *double*, dimana nilai variabel tersebut didapatkan dengan memanggil fungsi *getMax* dan *getMin*. Dimana nilai *max min* tersebut diset pada fungsi *find_min_max*.

Baris 4 : Deklarasi variabel *data_normalisasi* yang merupakan *array* bertipe *double*, dimana panjang *array*nya sebanyak panjang *data*.

Baris 5-7 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *data*.

Baris 6 : Perhitungan nilai *data_normalisasi* ke-*i*.

Baris 8 : Mengembalikan nilai *data_normalisasi*.

Nilai maximal dan minimal yang digunakan pada normalisasi didapatkan dari fungsi find min max yang juga terdapat pada kelas algoritma. Nama method fungsi *min max* adalah find_min_max. Kode program find_min_max dapat dilihat pada Tabel 5.2 yang disesuaikan dengan perancangan pada Gambar 4.3.

Tabel 5.2 Kode Program *Find Min Max*

No	Kode Program
1	`public void find_min_max(double[] data) {`
2	`   double max = 0;`
3	`   double min = Double.MAX_VALUE;`
4	`   for (int j = 0; j < data.length; j++) {`
5	`     if (data[j] > max) {`
6	`         max = data[j];`
7	`     }`
8	`     if (data[j] < min) {`
9	`         min = data[j];`
10	`     }`
11	`   }`
12	`   this.setMax(max);`
13	`   this.setMin(min);`
14	`}`

Penjelasan kode program pada Tabel 5.2.

Baris 1 : Deklarasi fungsi *find_min_max* dengan parameter data yang merupakan *array* bertipe *double*. Fungsi ini digunakan untuk memcari nilai mkasimal dan nilai minimal pada dataset.

Baris 2-1 : Deklarasi variabel *max* dan *min* dengan tipe *double*. Variabel *max* didklarasikan dengan nilai 0, sedangkan nilai variabel *min* didapatkan dari *library* double.java dengan mengambil nilai *MAX_VALUE*.

Baris 4-11 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama j kurang dari dari panjang *data*.

Baris 5-7 : Jika kondisi nilai data ke-*j* lebih dari *max*, maka nilai *max* berubah menjadi nilai *data* tersebut.

Baris 8-10 : Jika kondisi nilai data ke-*j* kurang dari min, maka nilai *min* berubah menjadi nilai *data* tersebut.

Baris 12-13 : Mengeset nilai *max* dan *min* dengan nilai *max* dan *min* hasil proses *find_min_max*.

5.1.2 Format Data

Format data berfungsi untuk membagi dataset menjadi data yang memiliki sejumlah fitur yang ditentukan serta menentukan nilai data tersebut. Pada proses format data ini terdapat 1 *method* (fungsi) yang berada pada kelas algoritma, yaitu fungsi format_data untuk mengolah dataset menjadi data yang memiliki sejumlah fitur dan target yang telah ditentukan serta menentukan nilai data tersebut.

Kode program untuk fungsi format_data dapat dilihat pada Tabel 5.3 yang disesuaikan dengan perancangan pada Gambar 4.4.

Tabel 5.3 Kode Program Format Data

No	Kode Program
1	`double[][] format_data(double[] data, int fitur) {`
2	`   double[][] data_terformat = new double[data.length - fitur][(fitur + 1)];`
3	`   for (int i = 0; i < data_terformat.length; i++) {`
4	`       for (int j = 0; j < (fitur + 1); j++) {`
5	`            data_terformat[i][j] = data[i + j];`
6	`       }`
7	`   }`
8	`   return data_terformat;`
9	`}`

Penjelasan kode program pada Tabel 5.3.
Baris 1 : Deklarasi fungsi *format_data* dengan parameter data yang merupakan array bertipe double dan fitur bertipe integer.
Baris 2 : Deklarasi variabel *data_terformat* yang merupakan *double array* bertipe *double*. Pada variabel ini terdapat elemen baris sebanyak panjang *data* dikurangi 1 dan nilai elemen kolom sebanyak *fitur* ditambah 1.
Baris 3-7 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari dari panjang *data_terformat*.
Baris 4-6 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari *fitur* ditambah 1.
Baris 5 : Nilai *data_terformat* pada elemen baris ke-*i* dan elemen kolom ke-*j* merupakan nilai pada *data* ke-(*i* dijumlahkan *j*).
Baris 8 : Mengembalikan nilai data_terformat

5.1.3 Bagi Data

Data yang telah diproses pada format_data, data tersebut dibagi menjadi data latih dan data uji. Pada proses bagi data ini terdapat 1 *method* (fungsi) yang berada pada kelas algoritma, yaitu fungsi bagi_data untuk membagi data menjadi data latih serta data uji. Kode program untuk fungsi bagi_data dapat dilihat pada Tabel 5.4 yang disesuaikan dengan perancangan pada Gambar 4.5.

Tabel 5.4 Kode Program Bagi Data

No	Kode Program
1	`void bagi_data(double[][] data_terformat, double presentase_latih) {`
2	`   int jumlah_latih = (int) (presentase_latih * data_terformat.length);`
3	`   int jumlah_uji = data_terformat.length - jumlah_latih;`
4	`   double[][] data_latih = new double[jumlah_latih][data_terformat[0].length - 1];`
5	`   double[][] data_uji = new double[jumlah_uji][data_terformat[0].length - 1];`
6	`   double[] t_latih = new double[jumlah_latih];`
7	`   double[] t_uji = new double[jumlah_uji];`
8	`   for (int i = 0; i < jumlah_latih; i++) {`

```
 9          for (int j = 0; j < data_terformat[0].length - 1;
            j++) {
10              data_latih[i][j] = data_terformat[i][j];
11          }
12          t_latih[i] =
            data_terformat[i][data_terformat[0].length - 1];
13      }
14      for (int i = 0; i < jumlah_uji; i++) {
15          for (int j = 0; j < data_terformat[0].length - 1;
            j++) {
16              data_uji[i][j] = data_terformat[i +
                jumlah_latih][j];
17          }
18          t_uji[i] = data_terformat[i +
            jumlah_latih][data_terformat[0].length - 1];
19      }
20      setData_latih(data_latih);
21      setData_uji(data_uji);
22      setT_latih(t_latih);
23      setT_uji(t_uji);
24  }
```

Penjelasan kode program pada Tabel 5.4.

Baris 1 : Deklarasi fungsi *bagi_data* dengan parameter *data_terformat* yang merupakan *double array* bertipe *double* dan *presentase_latih* bertipe *double*.

Baris 2-7 : Deklarasi variabel beserta nilai dan/atau banyaknya kolom baris array.

Baris 8-13 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari *jumlah_latih*.

Baris 9-11 : Perluangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari dari panjang kolom *data_terformat* dikurangi 1.

Baris 10 : Nilai *data_latih* pada elemen baris ke-*i* dan elemen kolom ke-*j* merupakan nilai pada *data_terformat* elemen baris ke-*i* dan elemen kolom ke-*j*

Baris 12 : Nilai *t_latih* ke-*i* merupakan nilai pada *data_terformat* elemen baris ke-*i* dan elemen kolom ke-hasil panjang kolom *data_terformat* dikurangi 1.

Baris 14-19 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari *jumlah_uji*.

Baris 15-17 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang kolom *data* dikurangi 1.

Baris 16 : Nilai *data_uji* pada elemen baris ke-*i* dan elemen kolom ke-*j* merupakan nilai pada *data_terformat* elemen baris ke-hasil *i* dijumlah dengan *jumlah_latih* dan elemen kolom ke-*j*

Baris 18 : Nilai *t_latih* ke-*i* merupakan nilai pada *data_terformat* elemen baris ke-*i* dijumlah dengan *jumlah_latih* dan elemen kolom kehasil panjang kolom *data* dikurangi 1.

Baris 20-23 : Mengeset nilai variabel.

5.1.4 Pelatihan

Fungsi pendukung dan penyusun pelatihan berada pada kelas algoritma. Implementasi pelatihan sesuai pada perancangan yang telah dilakukan. Kode

program pelatihan data dilihat pada Tabel 5.5 yang disesuaikan dengan perancangan pada Gambar 4.6.

Tabel 5.5 Kode Program Pelatihan

No	Kode Program
1	`List pelatihan(double[][] data_latih, double t[], double alfa, double max_error, int max_iter, int fitur, int hidden) {`
2	`this.inisialisasi_bobot(hidden, fitur);`
3	`this.inisialisasi_bias(hidden);`
4	`int iterasi = 0;`
5	`List hasil_mse = new ArrayList();`
6	`do {`
7	`double y[] = new double[data_latih.length];`
8	`for (int i = 0; i < data_latih.length; i++) {`
9	`y[i] = this.feedforward(data_latih[i], this.getV(), this.getBias(), this.getW(), this.getBias1());`
10	`this.backpropagation(y[i], t[i], alfa, this.getZ(), data_latih[i], fitur);`
11	`if (i != data_latih.length - 1) {`
12	`this.update();`
13	`}`
14	`}`
15	`double mse = this.MSE(t, y);`
16	`hasil_mse.add(mse);`
17	`if (mse > max_error) {`
18	`iterasi++;`
19	`} else {`
20	`break;`
21	`}`
22	`} while (iterasi < max_iter);`
23	`return hasil_mse;`
24	`}`

Penjelasan kode program pada Tabel 5.5.

Baris 1 : Deklarasi fungsi *pelatihan* dengan parameter.

Baris 2-3 : Memanggil fungsi *inisialisasi_bobot* dan *inisialisasi_bias*.

Baris 4 : Deklarasi variabel *iterasi* bertipe *integer* dengan nilai 0.

Baris 5 : Deklarasi variabel *hasil_mse* yang merupakan *array list*.

Baris 6-22 : Melakukan baris 7-21 dan melakukan perulangan selama kondisi *iterasi* kurang dari *max_iter*.

Baris 7 : Deklarasi variabel *y* yang merupakan *array* bertipe *double*. Panjang *array* variabel *y* sama dengan panjang *data_latih*.

Baris 8-14 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *data_latih*.

Baris 9 : Nilai variabel *y* diisi dengan memanggil fungsi *feedforward*.

Baris 10 : Memanggil fungsi *backpropagation*.

Baris 11-13 : Jika kondisi *i* tidak sama dengan panjang *data_latih* dikurangi satu, maka memanggil fungsi *update*.

Baris 15 : Deklarasi variabel *mse* bertipe *double*, dimana nilainya diisi dengan memanggil fungsi *MSE*.
Baris 16 : Variabel *hasil _mse* diisi dengan nilai *mse*.
Baris 17-21 : Jika kondisi *mse* lebih besar dari *max_error*, maka nilai variabel *iterasi* ditambah. Jika tidak maka berhenti.
Baris 23 : Mengembalikan nilai *hasil _mse*.

Kode program inisialisasi_bobot dilihat pada Tabel 5.6 yang disesuaikan dengan perancangan pada Gambar 4.7. Fungsi ini digunakan untuk menginisialisasi bobot awal yang akan digunakan pada perhitungan pelatihan.

Tabel 5.6 Kode Program Inisialisasi Bobot

No	Kode Program
1	`void inisialisasi_bobot(int jumlah_hidden, int jumlah_fitur) {`
2	`   double max = 0.5;`
3	`   double min = -0.5;`
4	`   double range = (max - min);`
5	`   double v[][] = new double[jumlah_fitur][jumlah_hidden];`
6	`   double w[] = new double[jumlah_hidden];`
7	`   for (int i = 0; i < jumlah_fitur; i++) {`
8	`      for (int j = 0; j < jumlah_hidden; j++) {`
9	`         v[i][j]= (Math.random() * range) + min;`
10	`      }`
11	`   }`
12	`   for (int j = 0; j < jumlah_hidden; j++) {`
13	`      w[j] = (Math.random() * range) + min;`
14	`   }`
15	`   setV(v);`
16	`   setW(w);`
17	`}`

Penjelasan kode program pada Tabel 5.6.
Baris 1 : Deklarasi fungsi *inisialisasi_bobot* dengan parameter *jumlah_hidden* dan *jumlah_fitur* bertipe *integer*.
Baris 2-6 : Deklarasi variabel.
Baris 7-12 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari *jumlah_fitur*.
Baris 8-11 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari *jumlah_hidden*.
Baris 9 : Perhitungan nilai *v* pada elemen baris ke-*i* dan elemen kolom ke-*j*.
Baris 12-14 : Perulangan selama kondisi *j*=0 dan *j* kurang dari *jumlah_hidden*.
Baris 13 : Perhitungan nilai *w* pada elemen baris ke-*i*.
Baris 15-16 : Mengeset nilai variabel.

Kode program inisialisasi_bias dilihat pada Tabel 5.7 yang disesuaikan dengan perancangan pada Gambar 4.8. Fungsi ini digunakan untuk menginisialisasi bias awal yang akan digunakan pada perhitungan pelatihan.

Tabel 5.7 Kode Program Inisialisasi Bias

No	Kode Program
1	`void inisialisasi_bias(int jumlah_hidden) {`
2	`   double max = 0.5;`
3	`   double min = -0.5;`
4	`   double range = (max - min);`
5	`   double bias[] = new double[jumlah_hidden];`
6	`   double bias1;`
7	`   for (int j = 0; j < jum_hidden; j++) {`
8	`      bias[j]= (Math.random() * range) + min;`
9	`   }`
10	`   bias1 = (Math.random() * range) + min;`
11	`   setBias(bias);`
12	`   setBias1(bias1);`
13	`}`

Penjelasan kode program pada Tabel 5.7.
Baris 1 : Deklarasi fungsi *inisialisasi_bobot* dengan parameter *jumlah_hidden* bertipe integer.
Baris 2-6 : Deklarasi variabel.
Baris 7-10 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari *jumlah_hidden*.
Baris 8 : Perhitungan nilai *bias* ke-*j*.
Baris 10 : Perhitungan nilai *bias1*.
Baris 11-12 : Mengeset nilai variabel.

Kode program feedforward dilihat pada Tabel 5.8 yang disesuaikan dengan perancangan pada Gambar 4.9.

Tabel 5.8 Kode Program *Feedforward*

No	Kode Program
1	`double feedforward(double[] data, double[][] v, double[] bias, double[] w, double bias1) {`
2	`        double[] z = hitung_hidden(data, v, bias);`
3	`        setZ(z);`
4	`        double y = hitung_output(z, w, bias1);`
5	`        return y;`
6	`}`

Penjelasan kode program pada Tabel 5.8.
Baris 1 : Deklarasi fungsi *feedforward* dengan parameter.
Baris 2 : deklarasi variabel *z* yang merupakan *array* bertipe *double*. Nilai variabel *z* diisi dengan memanggil fungsi *hitung_hidden*.
Baris 3 : Mengeset nilai variabel.

Baris 4 : Deklarasi variabel *y* bertipe *double*. Nilai variabel *y* diisi dengan memanggil fungsi *hitung_output*.
Baris 5 : Mengembalikan nilai *y*.

Kode program hitung_hidden dilihat pada Tabel 5.9 yang disesuaikan dengan perancangan pada Gambar 4.10.

Tabel 5.9 Kode Program *Feedforward* pada *Hidden Layer*

No	Kode Program
1	`double[] hitung_hidden(double[] data, double[][] v, double[] bias) {`
2	`   double[] z_in = new double[v[0].length];`
3	`   double[] z = new double[z_in.length];`
4	`   double tot;`
5	`   for (int j = 0; j < v[0].length; j++) {`
6	`      tot = 0;`
7	`      for (int k = 0; k < data.length; k++) {`
8	`         tot += data[k] * v[k][j];`
9	`       }`
10	`       z_in[j] = tot + bias[j];`
11	`       z[j] = aktivasi_sigmoid_biner(z_in[j]);`
12	`   }`
13	`   return z;`
14	`}`

Penjelasan kode program pada Tabel 5.9.
Baris 1 : Deklarasi fungsi *hitung_hidden* dengan parameter.
Baris 2-4 : Deklarasi variabel.
Baris 5-12 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang elemen baris *v*.
Baris 6 : Pemberian nilai variabel *tot*=0.
Baris 7-9 : Perulangan dengan kondisi awal *k*=0 dan dilakukan perulangan selama *k* kurang dari panjang *data*.
Baris 8 : Perhitungan nilai *tot*.
Baris 10 : Perhitungan nilai variabel *z_in*.
Baris 11 : Perhitungan nilai variabel *z* diisi dengan memanggil fungsi *aktivasi_sigmoid_biner*.
Baris 13 : Mengembalikan nilai *z*.

Kode program hitung_output dilihat pada Tabel 5.10 yang disesuaikan dengan perancangan pada Gambar 4.11.

Tabel 5.10 Kode Program *Feedforward* pada *Output Layer*

No	Kode Program
1	`double hitung_output(double[] data_hidden, double[] w, double bias) {`
2	`   double y_in = 0;`
3	`   double y;`
4	`   for (int i = 0; i < data_hidden.length; i++) {`

```
5        y_in += data_hidden[i] * w[i];
6     }
7     y_in += bias;
8     y = aktivasi_sigmoid_biner(y_in);
9     return y;
10    }
```

Penjelasan kode program pada Tabel 5.10.
Baris 1 : Deklarasi fungsi *hitung_output* dengan parameter.
Baris 2-3 : Deklarasi variabel.
Baris 5-6 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang elemen baris *w*, menghitung nilai variabel *y_in*.
Baris 7 : Menghitung nilai variabel *y_in*.
Baris 8 : Perhitungan nilai *y* yang diisi dengan memanggil fungsi *aktivasi_sigmoid_biner*.
Baris 9 : Mengembalikan nilai *y*.

Kode program aktivasi_sigmoid_biner dilihat pada Tabel 5.11 yang disesuaikan dengan perancangan pada Gambar 4.12.

Tabel 5.11 Kode Program Aktivasi *Sigmoid Biner*

No	Kode Program
1	`double aktivasi_sigmoid_biner(double data) {`
2	`   double aktivasi_sigmoid_biner = 1 / (1 + Math.exp(-data));`
3	`   return aktivasi_sigmoid_biner;`
4	`}`

Penjelasan kode program pada Tabel 5.11.
Baris 1 : Deklarasi fungsi *aktivasi_sigmoid_biner* dengan parameter *data* bertipe *double*.
Baris 2 : Perhitungan nilai variabel *aktivasi_sigmoid_biner*.
Baris 3 : Mengembalikan nilai *aktivasi_sigmoid_biner*.

Pada pelatihan algoritme *backpropagation*, setelah proses *feedforward* adalah proses *backpropagation*. Kode program *backpropagation* dapat dilihat pada Tabel 5.12 yang disesuaikan dengan perancangan pada Gambar 4.13.

Tabel 5.12 Kode Program *Backpropagation*

No	Kode Program
1	`void backpropagation(double data_y, double t, double alfa, double[] z, double data[], int fitur) {`
2	`   double δ[] = new double[z.length];`
3	`   double[] delta_w = new double[z.length];`
4	`   double δk = hitung_δ(data_y, t);`
5	`   for (int i = 0; i < delta_w.length; i++) {`
6	`      delta_w[i] = δk * alfa * z[i];`
7	`   }`

```
 8      double delta_bias1 = δk * alfa;
 9      double δ_in[] = new double[z.length];
10      double w[] = getW();
11      for (int i = 0; i < δ_in.length; i++) {
12         δ_in[i] = δk * w[i];
13      }
14      for (int j = 0; j < z.length; j++) {
15         δ[j] = δ_in[j] * z[j] * (1 - z[j]);
16      }
17      double[] delta_bias = new double[z.length];
18      for (int j = 0; j < delta_bias.length; j++) {
19         delta_bias[j] = alfa * δ[j];
20      }
21      double delta_v[][] = new double[fitur][z.length];
22      for (int i = 0; i < fitur; i++) {
23         for (int j = 0; j < z.length; j++) {
24            delta_v[i][j] = alfa * δ[j] * data[i];
25         }
26      }
27      setDelta_bias(delta_bias);
28      setDelta_bias1(delta_bias1);
29      setDelta_v(delta_v);
30      setDelta_w(delta_w);
31  }
```

Penjelasan kode program pada Tabel 5.12.

Baris 1 : Deklarasi fungsi *backpropagation* dengan parameter.

Baris 2-3 : Deklarasi variabel.

Baris 4 : Deklarasi variabel δk yang nilainya diisi dengan memanggil fungsi *hitung_δ*.

Baris 5-7 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *delta_w*.

Baris 6 : Perhitungan nilai *delta_w* ke-*i*.

Baris 8 : Perhitungan nilai *delta_bias1*.

Baris 9-10 : Deklarasi variabel.

Baris 11-13 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *δ_in*.

Baris 12 : Perhitungan nilai *δ_in* ke-*i*.

Baris 14-16 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang *z*.

Baris 15 : Perhitungan nilai δ ke-*j*.

Baris 17 : Deklarasi variabel.

Baris 18-20 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang *delta_bias*.

Baris 19 : Perhitungan nilai *delta_bias* ke-*j*.

Baris 21 : Deklarasi variabel.

Baris 22-26 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari *fitur*.

Baris 23-25 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang *z*.

Baris 24 : Perhitungan nilai *delta_v* elemen baris ke-*i* dan elemen kolom ke*j*.
Baris 27-30 : Mengeset nilai variabel.

Kode program hitung_δ dapat dilihat pada Tabel 5.13 yang disesuaikan dengan perancangan pada Gambar 4.14.

Tabel 5.13 Kode Program Koreksi Bobot W

No	Kode Program
1	`double hitung_δ(double data_y, double t) {`
2	`   double δk = (t - data_y) * data_y * (1 - data_y);`
3	`   return δk;`
4	`}`

Penjelasan kode program pada Tabel 5.13.
Baris 1 : Deklrasi fungsi *hitung_*δ dengan parameter *data_y* dan *t* bertipe *double*.
Baris 2 : Deklarasi variabel δk dan perhitungan nilainya.
Baris 3 : Mengembalikan nilai δk.

Setelah menghitung *feedforward* dan *backpropagation* pada setiap data, dilakukan *update* bobot dan bias untuk perhitungan pada data selanjutnya. Kode program untuk update bobot dan bias dengan nama fungsi update, dapat dilihat pada Tabel 5.14 yang disesuaikan dengan perancangan pada Gambar 4.15.

Tabel 5.14 Kode Program *Update* Bobot dan Bias

No	Kode Program
1	`void update() {`
2	`   double delta_v[][] = getDelta_v();`
3	`   double delta_w[] = getDelta_w();`
4	`   double delta_bias[] = getDelta_bias();`
5	`   double delta_bias1 = getDelta_bias1();`
6	`   double w[] = getW();`
7	`   double bias1 = getBias1();`
8	`   for (int i = 0; i < w.length; i++) {`
9	`      w[i] += delta_w[i];`
10	`   }`
11	`   bias1 += delta_bias1;`
12	`   double v[][] = getV();`
13	`   double bias[] = getBias();`
14	`   for (int i = 0; i < v.length; i++) {`
15	`      for (int j = 0; j < v[0].length; j++) {`
16	`         v[i][j] += delta_v[i][j];`
17	`      }`
18	`   }`
19	`   for (int i = 0; i < bias.length; i++) {`
20	`      bias[i] += delta_bias[i];`
21	`   }`
22	`   setW(w);`
23	`   setBias1(bias1);`
24	`   setV(v);`
25	`   setBias(bias);`
26	`}`

Penjelasan kode program pada Tabel 5.14.
Baris 1 : Deklarasi fungsi *update*.
Baris 2-7 : Deklarasi variabel.
Baris 8-10 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *w*.
Baris 9 : Perhitungan nilai *w* ke-*i*.
Baris 11 : Perhitungan nilai *bias1*.
Baris 12-13 : Deklarasi variabel.
Baris 14-18 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *v*.
Baris 15-17 : Perulangan dengan kondisi awal *j*=0 dan dilakukan perulangan selama *j* kurang dari panjang elemen baris *v*.
Baris 16 : Perhtungan nilai *v* pada elemen baris ke-*i* dan elemen kolom ke-*j*.
Baris 19-21 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *bias*.
Baris 20 : Perhitungan nilai *bias*.
Baris 22-25 : Mengeset nilai variabel.

Pada setiap iterasi, setelah semua data dilakukan proses perhitungan sebelumnya maka dihitung MSEnya. Kode program MSE dapat dilihat pada Tabel 5.15 yang disesuaikan dengan Gambar 4.16.

Tabel 5.15 Kode Program MSE

No	Kode Program
1	`public double MSE(double[] target_uji, double[] y) {`
2	`   double tot = 0;`
3	`   for (int i = 0; i < target_uji.length; i++) {`
4	`      double selisih = y[i] - target_uji[i];`
5	`      double selisih_kuadrat = Math.pow(selisih, 2);`
6	`      jumlah_selisih += selisih_kuadrat;`
7	`   }`
8	`   double mse = jumlah_selisih / target_uji.length;`
9	`   return mse;`
10	`}`

Penjelasan kode program pada Tabel 5.15.
Baris 1 : Deklarasi fungsi *MSE* dengan parameter *target_uji* dan *y* yang merupakan *array* beertipe *double*.
Baris 2 : Deklarasi variabel.
Baris 3-7 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *target_uji*.
Baris 4 : Deklarasi variabel *selisih* dan perhitungan nilainya.
Baris 5 : Deklarasi variabel *selisih _kuadrat* dan perhitungan nilainya.
Baris 6 : Perhitungan nilai *jumlah_ selisih*.
Baris 8 : Deklarasi variabel *mse* dan perhitungan nilainya.
Baris 9 : Mengembalikan nilai *mse*.

5.1.5 Pengujian

Pengujian dilakukan untuk mengetahui seberapa baik pelatihan beserta nilai-nilai pada variabel dan parameter yang digunakan untuk peramalan. Fungsi pendukung dan penyusun pengujian berada pada kelas algoritma. Implementasi pengujian sesuai pada perancangan yang telah dilakukan. Kode program pengujian data dilihat pada Tabel 5.16 yang disesuaikan dengan perancangan pada Gambar 4.17.

No	Kode Program
1	`double [] pengujian(double [][]data_uji){`
2	`   double y[] = new double[data_uji.length];`
3	`   for (int i = 0; i < data_uji.length; i++) {`
4	`      double temp_y = this.feedforward(data_uji[i], this.getV(), this.getBias(), this.getW(), this.getBias1());`
5	`      y[i] = temp_y;`
6	`   }`
7	`   double mse = this.MSE(this.getT_uji(),y);`
8	`   return y;`
9	`}`

Penjelasan kode program pada Tabel 5.16.
Baris 1 : Deklarasi fungsi *pengujian* dengan parameter *data_uji* yang merupakan *double array* bertipe *double*.
Baris 2 : Deklarasi variabel.
Baris 3-6 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama *i* kurang dari panjang *data_uji*.
Baris 4 : Deklarasi variabel *temp_y*, yang nilainya diisi dengan memanggil fungsi *feedforward*.
Baris 5 : Pengisian nilai *y* ke-*i*.
Baris 7 : Deklarasi variabel *mse* yang nilainya diisi dengan memanggil fungsi *MSE*.
Baris 8 : Mengembalikan nilai *y*.

5.1.6 Denormalisasi

Denormalisasi dilakukan untuk mengetahui nilai yang sebenarnya, karena data yang menjadi inputan sebelumnya dinormalisasi dengan *range* 0 hingga 1. Data yang didenormalisasi merupakan hasil peramalan data uji. Fungsi pendukung dan penyusun denormalisasi berada pada kelas algoritma. Kode program denormalisasi dapat dilihat pada Tabel 5.17 yang disesuaikan dengan perancangan pada Gambar 4.18.

No	Kode Program
1	`public double[] denormalisasi(double data[]) {`
2	`   double max = getMax();`
3	`   double min = getMin();`
4	`   double[] data_asli = new double[data.length];`
5	`   for (int i = 0; i < data.length; i++) {`

```
6           data_asli[i] = Math.round(data[i] * (max - min) + min);
7       }
8       return data_asli;
9   }
```

Penjelasan kode program pada Tabel 5.17.

Baris 1 : Deklarasi fungsi *denormalisasi* dengan parameter *data* yang merupakan *array* bertipe *double*. Fungsi ini digunakan untuk mengubah nilai pada *data* yang mempunyai *range* 0 hingga 1 menjadi nilai yang sebenarnya.

Baris 2-3 : Deklarasi variabel *max* dan *min* yang betipe *double*, dimana nilai variabel tersebut didapatkan dengan memanggil fungsi *getMax* dan *getMin*.

Baris 4 : Deklarasi variabel *data_asli* yang merupakan *array* bertipe *double*, dimana panjang arraynya sebanyak panjang *data*.

Baris 5-7 : Perulangan dengan kondisi awal *i*=0 dan dilakukan perulangan selama i kurang dari panjang *data*.

Baris 6 : Perhitungan nilai *data_asli* ke-*i*.

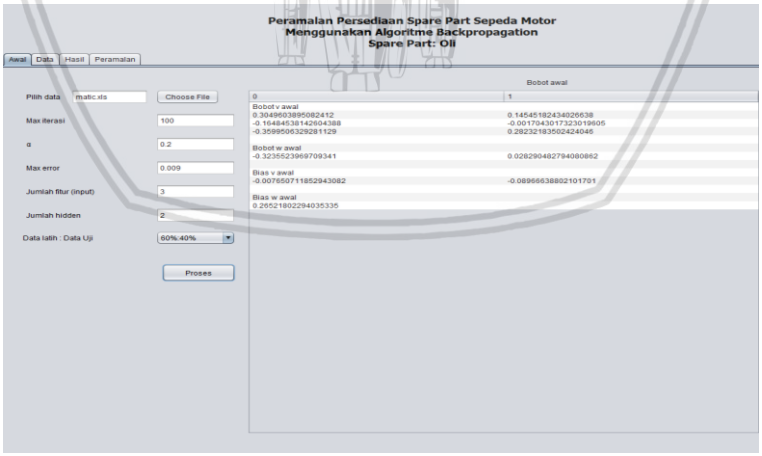
Baris 8 : Mengembalikan nilai *data_asli*.

5.2 Implementasi Antarmuka

Implementasi antarmuka dibuat sesuai dengan perancangan antarmuka yang telah dibuat. Antarmuka atau *Graphical User Interface (GUI)* dibuat menggunakan bahasa *java* dengan IDE NetBeans. Antarmuka ini dibuat untuk mempermudah pengguna memasukkan nilai dan mempermudah melihat hasilnya.

5.2.1 Tampilan Awal

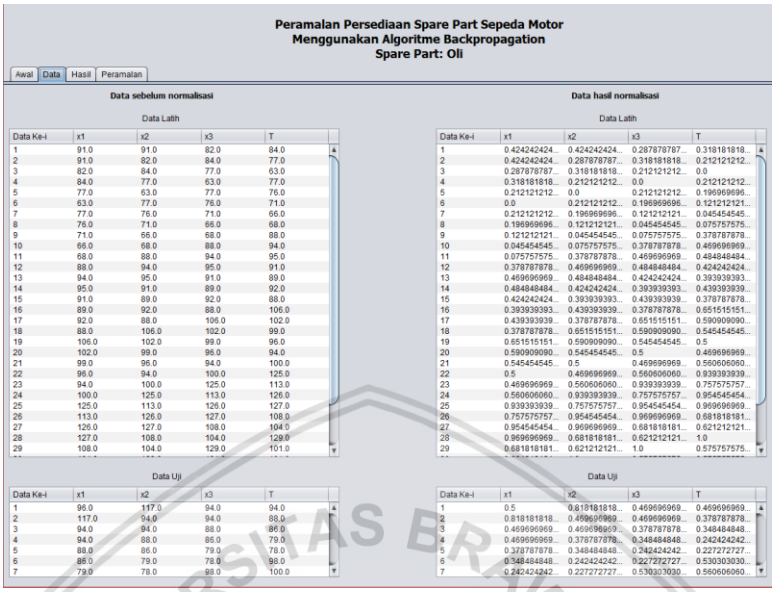
Implementasi antarmuka "tampilan awal" dapat dilihat pada Gambar 5.1.



Gambar 5.1 Implementasi Antarmuka Tampilan Awal

5.2.2 Data

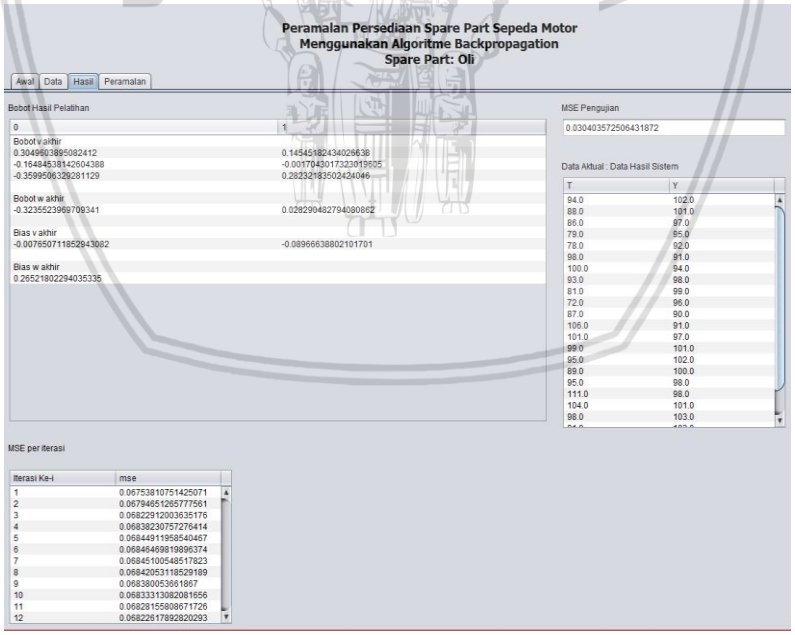
Implementasi antarmuka “data” dapat dilihat pada Gambar 5.2.



Gambar 5.2 Implementasi Antarmuka Data

5.2.3 Hasil

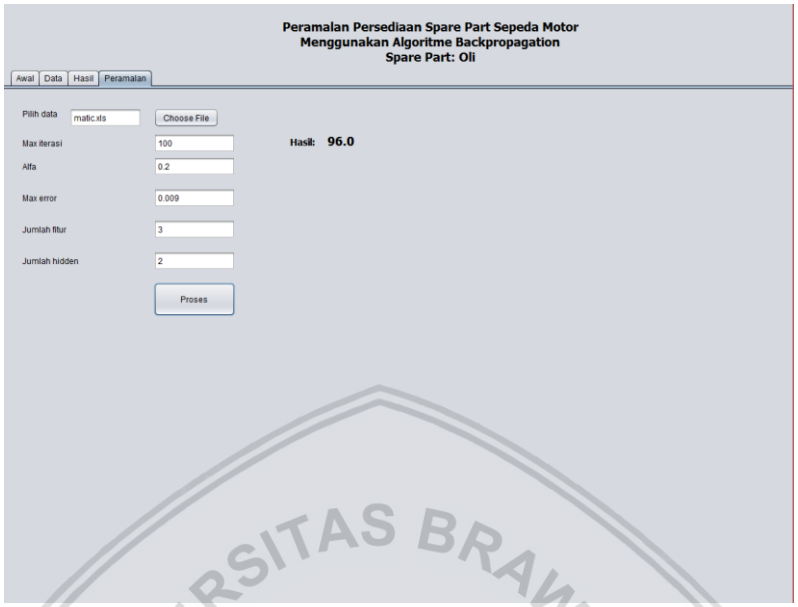
Implementasi antarmuka “hasil” dapat dilihat pada Gambar 5.3.



Gambar 5.3 Implementasi Antarmuka Hasil

5.2.4 Peramalan

Implementasi antarmuka “hasil” dapat dilihat pada Gambar 5.4.



Gambar 5.4 Implementasi Antarmuka Peramalan

BAB 6 PENGUJIAN DAN ANALISIS

6.1 Skenario Pengujian

Pengujian pada penelitian ini dilakukan untuk mengetahui nilai terbaik dari parameter sehingga peramalan persediaan *spare part* menggunakan algoritme *backpropagation* mendapatkan hasil yang optimal. Parameter mempengaruhi perubahan hingga tingkat akurasi, sehingga pemilihan parameter sangat mempengaruhi hasil (Saputra, et al., 2017). Data yang digunakan pada pengujian adalah *history* penjualan oli *yamalube matic* selama 60 bulan yang terdapat pada Lampiran A.1. Perbandingan data latih dan data uji pada pengujian ini adalah 80% dan 20% dengan pertimbangan penulis telah melakukan *trial and error*. Pengujian dengan 80% data latih dan 20% data uji dilakukan agar pengujian memiliki data yang variatif karena data bisa berkembang. Sehingga dengan menggunakan data yang bervariatif, proses pelatihan dan pengujian dapat lebih banyak mengenali data dan polanya. Pengujian yang dilakukan secara berurutan adalah pengujian jumlah iterasi, jumlah node *input layer*, jumlah node *hidden layer* dan nilai *learning rate* sesuai dengan perancangan pengujian pada bab perancangan. Nilai parameter yang digunakan untuk pengujian didapatkan dari referensi yang telah dibahas pada landasan kepustakaan. Setelah mendapatkan nilai terbaik dari parameter pada satu pengujian, maka nilai tersebut digunakan untuk pengujian selanjutnya apabila masih terdapat pengujian. Bobot dan bias awal yang digunakan adalah random, sehingga menghasilkan hasil yang berbeda. Oleh karena itu, pengujian dilakukan dengan menjalankan program sebanyak 5 kali untuk setiap iterasi yang diujikan dan dihitung rata-rata nilai *MSE*-nya. Semakin kecil nilai *MSE* atau nilai *error*-nya, maka menunjukkan semakin baik peramalan yang dilakukan.

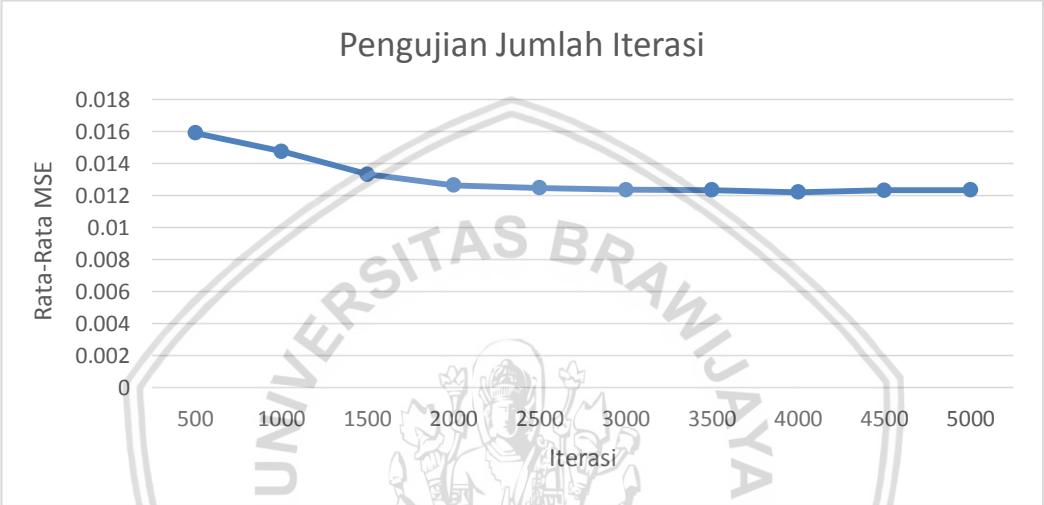
6.2 Pengujian Jumlah Iterasi

Pengujian jumlah iterasi dilakukan untuk mengetahui jumlah iterasi yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Iterasi yang diujikan adalah 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500 dan 5000. Jumlah node *hidden layer*, jumlah node *input layer* dan nilai *learning rate* yang digunakan pada setiap pengujian sama, yaitu mengacu pada penelitian Julpan dan kawan-kawan (2015). Jumlah node *hidden layer* adalah 3, jumlah node *input layer* adalah 6 dan nilai *learning rate* adalah 0.06. Hasil pengujian jumlah iterasi ditunjukkan pada Tabel 6.1.

Tabel 6.1 Pengujian Jumlah Iterasi

Iterasi	*MSE*					Rata-Rata *MSE*
	Percobaan ke-					
	1	2	3	4	5	
500	0.014946	0.015834	0.016606	0.016387	0.015753	0.015905
1000	0.015123	0.014727	0.014549	0.014795	0.014576	0.014754

1500	0.013977	0.012941	0.013459	0.013198	0.013056	0.013326
2000	0.012734	0.012747	0.012489	0.012824	0.01244	0.012647
2500	0.012551	0.012526	0.012657	0.01233	0.012343	0.012481
3000	0.012352	0.012272	0.012264	0.012525	0.012389	0.01236
3500	0.012584	0.012507	0.012283	0.012132	0.012159	0.012333
4000	0.012153	0.012144	0.012501	0.012146	0.012052	0.012199
4500	0.012387	0.012332	0.012138	0.012245	0.012485	0.012317
5000	0.012578	0.012277	0.012251	0.012484	0.012115	0.012341



Gambar 6.1 Pengujian Jumlah Iterasi

Gambar 6.1 merupakan grafik hasil pengujian jumlah iterasi. Dari grafik tersebut dapat dilihat bahwa mulai iterasi 2500 sudah menunjukkan konvergen dan jumlah iterasi terbaik adalah 4000 iterasi dengan rata-rata nilai *MSE* 0.012199.

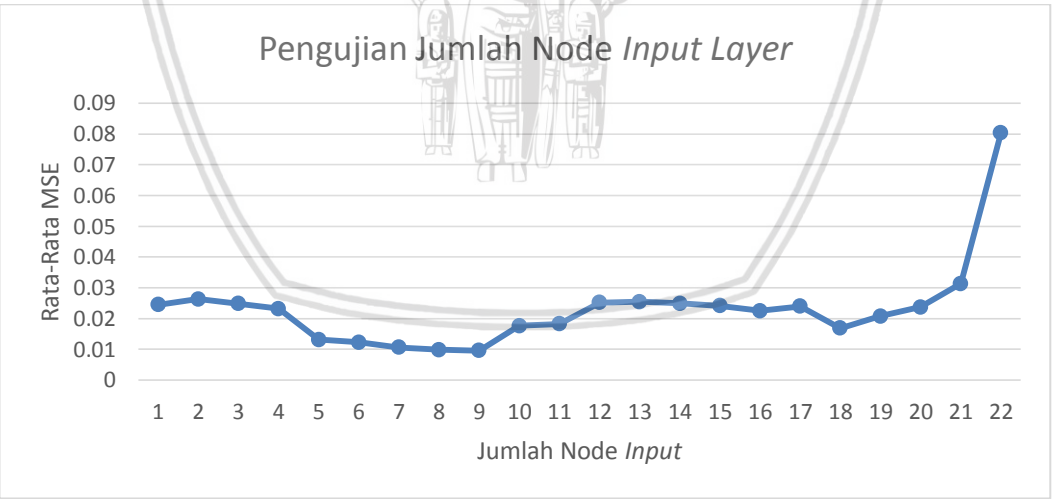
6.3 Pengujian Jumlah Node *Input Layer*

Pengujian node *input layer* yang yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Jumlah node *input layer* yang diujikan adalah 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 dan 22. Nilai *Learning rate*, jumlah node *hidden layer* dan iterasi yang digunakan pada setiap pengujian sama, yaitu mengacu pada penelitian Julpan dan kawan-kawan (2015) dan pengujian sebelumnya pada penelitian ini. Jumlah node *hidden layer* adalah 3, nilai *learning rate* adalah 0.06 dan jumlah iterasi 4000 yang telah didapatkan dari pengjuan sebelumnya pada penelitian ini. Hasil pengujian jumlah node *input layer* ditunjukkan pada Tabel 6.2

Tabel 6.2 Pengujian Jumlah Node *Input Layer*

Jumlah node input layer	*MSE*					Rata-Rata *MSE*
	Percobaan ke-					
	1	2	3	4	5	

1	0.0242753	0.0258981	0.0250315	0.0237911	0.0235581	0.0245108
2	0.0259775	0.0265797	0.0261069	0.0262123	0.0267745	0.0263302
3	0.024712	0.0246294	0.0247341	0.0248383	0.0250933	0.0248014
4	0.0228832	0.0230303	0.0237752	0.023086	0.0232707	0.0232091
5	0.0131003	0.0132643	0.0129427	0.0129142	0.0133378	0.0131118
6	0.0120594	0.0122824	0.0123953	0.0122207	0.0123275	0.012257
7	0.0106126	0.0106808	0.0104253	0.0106306	0.0105915	0.0105882
8	0.0094364	0.0097506	0.0099613	0.0103815	0.0096101	0.009828
9	0.00933	0.0093918	0.0101146	0.0095771	0.0095384	0.0095904
10	0.0183075	0.0175768	0.0175493	0.0179539	0.016917	0.0176609
11	0.0193492	0.0180831	0.017779	0.0180875	0.0178623	0.0182322
12	0.0244442	0.0276118	0.0235035	0.0247784	0.0254049	0.0251486
13	0.021414	0.0278945	0.0263435	0.0258992	0.0258427	0.0254788
14	0.0235829	0.0229159	0.0252874	0.0259189	0.0268749	0.024916
15	0.0232615	0.027342	0.0218767	0.0255927	0.0224853	0.0241116
16	0.0227186	0.0223725	0.0224719	0.0227263	0.0222662	0.0225111
17	0.0244948	0.0255009	0.0245551	0.0224857	0.0229628	0.0239998
18	0.0147666	0.0178458	0.019999	0.0160639	0.0155505	0.0168451
19	0.0197813	0.0259079	0.01988	0.0181184	0.0199993	0.0207374
20	0.0227695	0.0249988	0.0236634	0.0239479	0.0230761	0.0236911
21	0.0351104	0.0262054	0.0328325	0.0341592	0.0284832	0.0313581
22	0.0746687	0.0794657	0.0825811	0.0733082	0.0916313	0.080331



Gambar 6.2 Pengujian Jumlah Node *Input Layer*

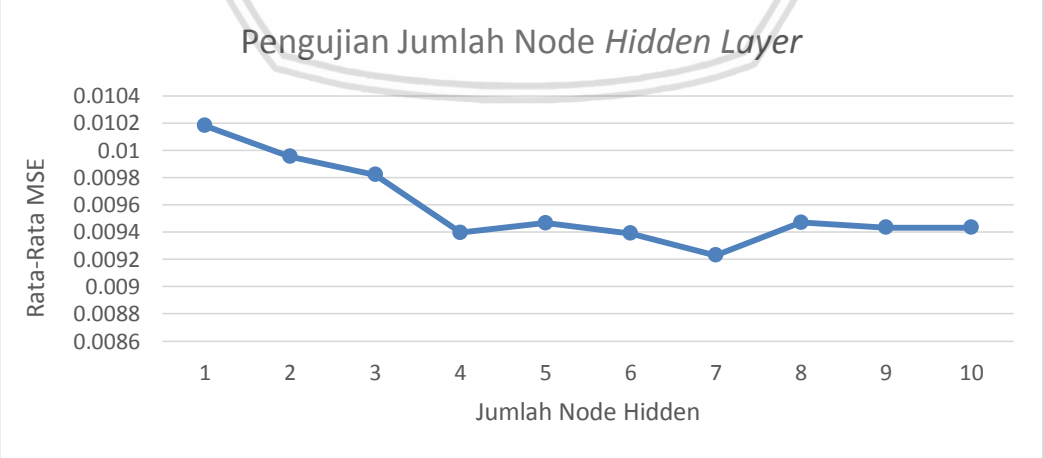
Gambar 6.2 merupakan grafik hasil pengujian jumlah node *input layer*. Jumlah node *input layer* terbaik adalah 9 dengan rata-rata nilai *MSE* 0.00959. Pola yang dimiliki adalah fluktuatif. Hal ini dapat disebabkan karena variabel *input*nya tidak merepresentasikan targetnya sehingga variabel *input* tersebut tidak berpengaruh baik pada jaringan. Merancang jaringan membutuhkan pilihan variabel *input* yang tepat untuk menghindari *overfitting* karena tidak semua variabel berpengaruh baik pada jaringan (Abdellah & Djamel, 2013).

6.4 Pengujian Jumlah Node *Hidden Layer*

Pengujian jumlah node *hidden layer* dilakukan untuk mendapatkan jumlah node *hidden layer* yang yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Jumlah node *hidden layer* yang diujikan adalah 1, 2, 3, 4, 5, 6, 7, 8, 9 dan 10. Nilai *Learning rate*, jumlah node *input layer* dan iterasi yang digunakan pada setiap pengujian sama, yaitu mengacu pada penelitian Julpan dan kawan-kawan (2015) dan pengujian sebelumnya pada penelitian ini. Nilai *learning rate* adalah 0.06 dan jumlah iterasi 4000 serta jumlah node *input layer* adalah 9 yang telah didapatkan dari pengujian sebelumnya pada penelitian ini. Semakin kecil nilai *MSE* atau nilai *error*-nya, maka menunjukkan semakin baik peramalan yang dilakukan. Hasil pengujian jumlah node *hidden layer* ditunjukkan pada Tabel 6.3.

Tabel 6.3 Pengujian Jumlah Node *Hidden Layer*

Jumlah node hidden *layer*	*MSE*					Rata-Rata *MSE*
	Percobaan ke-					
	1	2	3	4	5	
1	0.010203	0.010182	0.01013	0.010249	0.01014	0.010181
2	0.010334	0.009608	0.010301	0.010116	0.009402	0.009952
3	0.009252	0.009754	0.010043	0.009891	0.010157	0.009819
4	0.009518	0.009236	0.00938	0.009298	0.009548	0.009396
5	0.009198	0.009759	0.009302	0.009512	0.009564	0.009467
6	0.009077	0.009761	0.009258	0.009517	0.009335	0.00939
7	0.009233	0.009404	0.009195	0.009134	0.009182	0.00923
8	0.009483	0.009736	0.009284	0.00942	0.00943	0.009471
9	0.009382	0.009433	0.009422	0.009437	0.009487	0.009432
10	0.009429	0.009648	0.009324	0.009315	0.009449	0.009433



Gambar 6.3 Pengujian Jumlah Node *Hidden Layer*

Gambar 6.3 merupakan grafik hasil pengujian jumlah node *hidden layer*. Dari grafik tersebut dapat dilihat bahwa mulai jumlah node *hidden*nya 4 mengalami perubahan nilai *MSE* yang tidak signifikan sehingga dapat dikatakan konvergen dan jumlah node *hidden layer* terbaik adalah 7 dengan rata-rata nilai *MSE* 0.00923. Menurut Heaton (2005) jumlah node hidden layer yang baik adalah 2/3 dari jumlah node input layer ditambah dengan jumlah node output layer, selain itu jumlah node hidden layer harus kurang dari dua kali jumlah node input layer (Yohannes, et al., 2015).

6.5 Pengujian Nilai *Learning Rate*

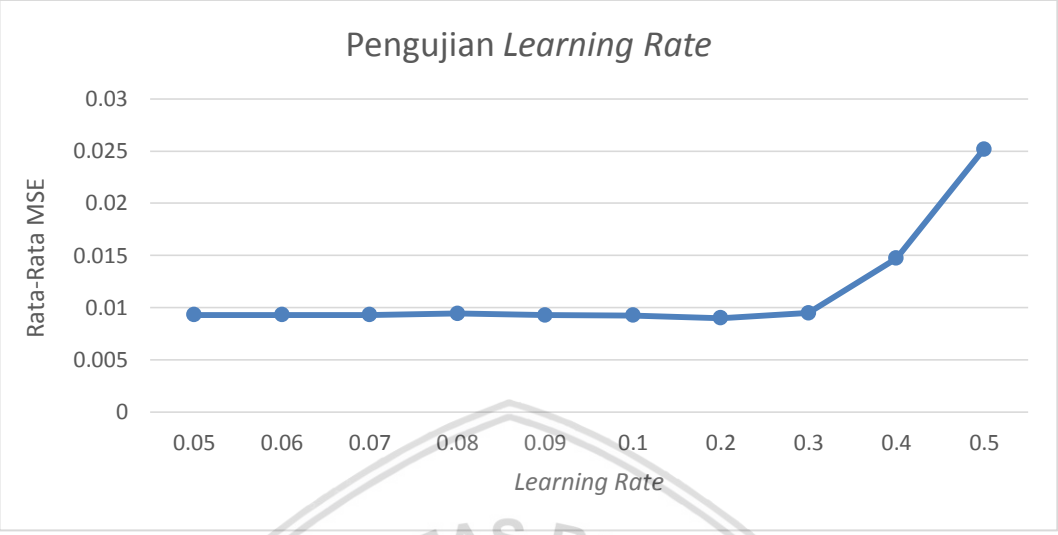
Pengujian nilai *learning rate* dilakukan untuk mendapatkan nilai *learning rate* yang terbaik agar hasil peramalan menghasilkan nilai *error* yang kecil. Nilai *learning rate* yang diujikan adalah 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4 dan 0.5. Jumlah node *hidden layer*, jumlah node *input layer* dan iterasi yang digunakan pada setiap pengujian sama, yaitu nilai terbaik yang telah didapatkan dari pengujian sebelumnya pada penelitian ini. Jumlah node *hidden layer* adalah 7, jumlah node *input layer* adalah 9 dan jumlah iterasi adalah 4000. Hasil pengujian jumlah iterasi ditunjukkan pada Tabel 6.4.

Tabel 6.4 Pengujian *Learning Rate*

learning rate	*MSE* Percobaan ke-					Rata-Rata *MSE*
	1	2	3	4	5	
0.05	0.009299	0.009223	0.009248	0.009376	0.009398	0.009309
0.06	0.009476	0.009159	0.009357	0.009219	0.00934	0.00931
0.07	0.009191	0.009396	0.009476	0.009365	0.00913	0.009311
0.08	0.009232	0.009455	0.009303	0.009648	0.009459	0.009419
0.09	0.009102	0.009152	0.009422	0.009192	0.009559	0.009285
0.1	0.009384	0.009254	0.00935	0.009192	0.009043	0.009245
0.2	0.008455	0.009088	0.008824	0.010036	0.008515	0.008984
0.3	0.008958	0.008742	0.009935	0.009935	0.009794	0.009473
0.4	0.01118	0.011216	0.021723	0.015037	0.01445	0.014721
0.5	0.025214	0.03784	0.025209	0.012364	0.025172	0.02516

Gambar 6.4 merupakan grafik hasil pengujian nilai *learning rate*. Dari grafik tersebut dapat dilihat bahwa mulai nilai *learning rate* 0.05 sudah menunjukkan konvergen dan nilai *learning rate* terbaik adalah 0.2 dengan rata-rata nilai *MSE* 0.008984. Nilai *MSE* pada *learning rate* 0.05 menuju ke *learning rate* 0.08 terus mengalami kenaikan, pada *learning rate* 0.09 mengalami penurunan hingga *learning rate* 0.2 yang menunjukkan nilai *MSE* terkecil, setelah itu nilai *MSE* mengalami kenaikan yg signifikan mulai *learning rate* 0.3. Semakin besar *learning rate* dapat menyebabkan ketelitian jaringan akan semakin berkurang dan semakin

kecil *learning rate* memungkinkan ketelitian jaringan akan semakin besar tetapi bertambah pula konsekuensi prosesnya (Widyaningrum & Romadhon, 2014).



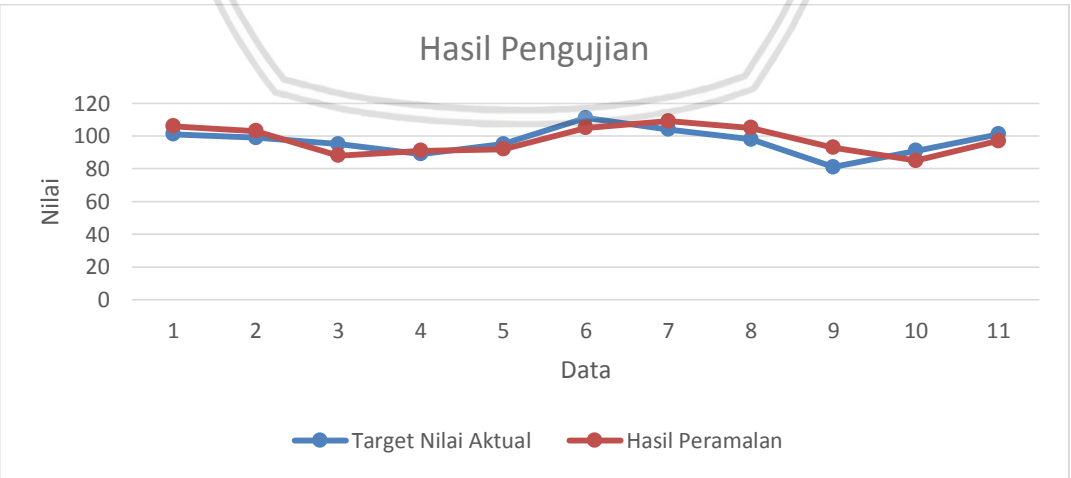
Gambar 6.4 Pengujian *Learning Rate*

6.6 Hasil Pengujian

Setelah mendapatkan nilai parameter terbaik, program dijalankan dengan nilai parameter terbaik untuk mengetahui seberapa baik hasil peramalan yang dilakukan. Hasil pengujian dapat dilihat pada Tabel 6.5.

Tabel 6.5 Hasil Pengujian

Percobaan ke-					Rata-Rata *MSE*
1	2	3	4	5	
0.0095835	0.0093691	0.0091437	0.0085305	0.0106263	0.0094506



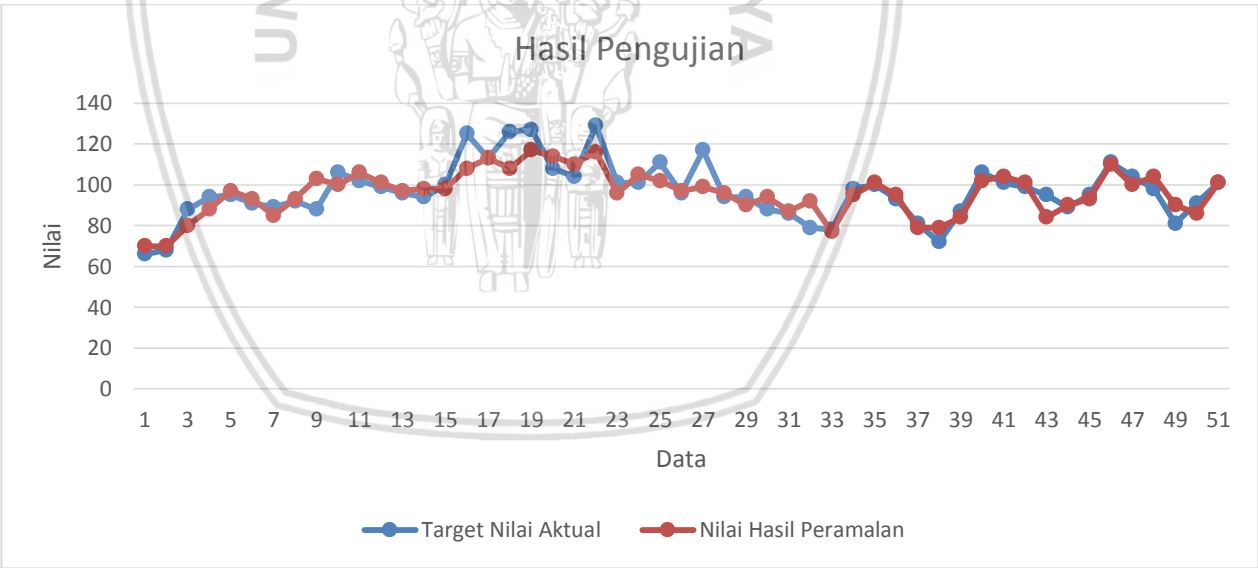
Gambar 6.5 Hasil Pengujian

Rata-rata *MSE* yang didapatkan adalah 0.0094506 dan *MSE* terkecil yang didapatkan adalah 0.0085305. Untuk mengetahui perbedaan nilai aktual dengan hasil peramalan, maka pada percobaan ke 4 yang memiliki nilai *MSE* terkecil dibuat grafik pada Gambar 6.5. Pada grafik tersebut menunjukkan bahwa hasil peramalan banyak mendekati nilai aktualnya, hanya ada beberapa yang memiliki perbedaan yang signifikan. Rata-rata selisih nilai aktual dengan nilai hasil peramalan adalah 6. Pola yang dimiliki antara nilai aktual dengan hasil peramalan juga memiliki kesamaan. Ketidaktepatan pola dapat disebabkan oleh data yang anomali dan belum didapatkannya bobot yang optimal.

Pengujian juga dilakukan terhadap data uji yang mempunyai data yang sama dengan data latih, yaitu 51 data. Hal ini dilakukan untuk mengetahui seberapa baik hasil peramalan yang dilakukan dengan data tersebut. Hasil pengujian dapat dilihat pada Tabel 6.5.

Tabel 6.6 Hasil Pengujian Data Uji sama dengan Data Latih

Percobaan ke-					Rata-Rata MSE
1	2	3	4	5	
0.0125155	0.0117653	0.0122308	0.0115223	0.0109775	0.0118023



Gambar 6.6 Hasil Pengujian Data Uji sama dengan Data Latih

Untuk mengetahui perbedaan nilai aktual dengan hasil peramalan, maka pada percobaan ke-5 yang memiliki nilai *MSE* terkecil dibuat grafik pada Gambar 6.6. Pada grafik tersebut menunjukkan bahwa hasil peramalan banyak mendekati nilai aktualnya, hanya ada beberapa yang memiliki perbedaan yang signifikan. Rata-rata selisih nilai aktual dengan nilai hasil peramalan adalah 5. Pola yang dimiliki antara nilai aktual dengan hasil peramalan juga memiliki kesamaan, meskipun ada

beberapa yang tidak sesuai. Ketidaktepatan pola dapat disebabkan oleh data yang anomali dan belum didapatkannya bobot yang optimal. Pada data uji yang mempunyai data yang sama dengan data latih juga ditunjukkan laju *MSE* pelatihan pada pengujian yang memiliki *MSE* pengujian terkecil, dapat dilihat pada Lampiran E. Berdasarkan Gambar 6.5 dan Gambar 6.6 dapat dilihat bahwa ketidaktepatan pola lebih banyak terjadi pada data dengan data uji yang mempunya data sama dengan data latih daripada data dengan 80% data latih dan 20% data uji. Hal tersebut dapat disebabkan karena data latih dan data uji tidak banyak variatif sehingga proses pelatihan dan pengujian kurang banyak mengenali data dan pola.

BAB 7 PENUTUP

7.1 Kesimpulan

Kesimpulan yang didapatkan dari penelitian ini adalah sebagai berikut:

1. Algoritme *backpropagation* berhasil diterapkan untuk meramalkan persedian *spare part* sepeda motor. Pelatihan *backpropagation* melibatkan tiga fase: *feedforward*, *backpropagation* dan *update* bobot. Setelah pelatihan, penerapan jaringan hanya melibatkan perhitungan fase *feedforward*. *Backpropagation* terdiri dari *input layer*, *hidden layer* dan *output layer* Arsitektur *backpropagation* yang terbaik dan digunakan adalah 9-7-1, yaitu 9 node *input*, 7 node *hidden* dan 1 node *output*. Pada penelitian ini data yang digunakan adalah *spare part* oli *yamalube matic* yang memiliki data sebanyak 60 bulan. Pengujian pada penelitian menghasilkan jumlah fitur (node *input*) terbaik adalah 9 fitur, sehingga dataset yang dimiliki adalah 51 data. Dari 51 data tersebut dibagi menjadi 40 data latih dan 11 data uji.
2. Rata-rata *MSE* (nilai *error*) yang didapatkan dari hasil pengujian adalah 0.0094506 dan *MSE* terkecil yang didapatkan adalah 0.0085305 dengan rata-rata selisih nilai aktual dengan nilai hasil peramalan adalah 6. Pada nilai *MSE* terkecil tersebut, hasil peramalan mendekati nilai aktualnya dan memiliki pola yang hampir sama. Ketika data uji memiliki data yang sama dengan data latihnya, rata-rata *MSE* yang didapatkan dari hasil pengujian adalah 0.0118023 dan *MSE* terkecil yang didapatkan adalah 0.0109775 dengan rata-rata selisih nilai aktual dengan nilai hasil peramalan adalah 5.

7.2 Saran

Saran yang diberikan dari penelitian ini untuk penelitian selanjutnya adalah sebagai berikut:

1. Perlu dilakukan penelitian yang lebih lanjut mengenai teknik optimasi penentuan nilai bobot dan bias awal serta interval nilai bobot dan bias awal pada algoritme *backpropagation* guna memberikan hasil bobot yang lebih baik pada pelatihan *backpropagation*.
2. Perlu dilakukan penelitian yang lebih lanjut mengenai teknik untuk mengetahui fluktuatif pola data hingga mencapai pola yang berulang guna memberikan hasil peramalan dengan pola yang lebih baik.
3. Pembagian data uji dan data latih pada penelitian selanjutnya dapat dilakukan secara dinamis atau dapat menggunakan *k-fold*, agar lebih mengenali pola *time-series*.
4. Penelitian selanjutnya dapat menggunakan data *spare part* lain yang lebih kompleks.

DAFTAR PUSTAKA

Abdellah, D. & Djamel, L., 2013. *Forecasting the Algerian Load Peak Profile Using Time Series Model Based On Backpropagation Neural Networks.* Istanbul, 4th International Conference on Power Engineering, Energy and Electrical Drives.

Afrianto, U. A., 2014. Sistem Informasi Jasa Layanan Service Sepeda Motor pada AHASS Motor Cahaya Sakti 871 Tlogosari Semarang. *UDiNus Repository.*

Augustine, A. & Manuharawati, 2017. Forecasting Fitness Gym Membership pada Pusat Kebugara "The Body Art Fitness, Aerobic & Pool" Menggunakan Metode Exponential Smoothing. *Jurnal Ilmiah Matematika,* Volume 31-7.

Cynthia, E. P. & Ismanto, E., 2017. *Jaringan Syaraf Tiruan Algoritma Backpropagation Dalam Memprediksi Ketersediaan Komoditi Pangan Provinsi Riau.* Pekanbaru, Seminar Nasional Teknologi Informasi, Komunikasi dan Industri (SNTIKI) 9.

Dinata, A. E. & Wigati, . S. S., 2016. *Analisis Sistem Persediaan Sparepart Motor di Bengkel Aneka Sakti.* s.l., Seminar Nasional IENACO.

Farida, I. & Rozini, M. N., 2016. Pengendalian Persediaan Spare Part dan Pengembangan dengan Konsep 80-20 (Analisis ABC) pada Gudang Suku Cadang PT. Astra International Tbk – Daihatsu Sales Operational Cabang Tegal. *SENIT,* pp. 163-169.

Fausett, L. V., 1993. *Fundamentals of Neural Network: Architecture, Algorithms, and Applications.* s.l.:Pearson.

Febrina, M., Arina, F. & Ekawati, R., 2013. Peramalan Jumlah Permintaan Produksi Menggunakan Metode Jaringan Syaraf Tiruan (JST) Backpropagation. *Jurnal Teknik Industri,* Volume 1, pp. 174-179.

Heripracoyo, S., 2009. *Analisis dan Perancangan Sistem Informasi Akuntansi Pembelian dan Persediaan pada PT. Oliser Indonesia.* Yogyakarta, Seminar Nasional Aplikasi Teknologi Informasi 2009 (SNATI 2009).

Hu, Q., Boylan, J. E., Chen, H. & Labib, A., 2018. OR in Spare Parts Management: A Review. *European Journal of Operational Research,* Volume 266, pp. 395-414.

Ismail, Z. & Jamaluddin, F. A., 2008. A Backpropagation Method for Forecasting Electricity Load Demand. *Journal of Applied Sciences,* Volume 8 (13), pp. 2428-2434.

Julpan, Nababan, E. B. & Zarlis, M., 2015. Analisis Fungsi Aktivasi Sigmoid Biner dan Sigmoid Bipolar dalam Algoritma Backpropagation pada Prediksi Kemampuan Siswa. *Jurnal Teknovasi,* Volume 2, pp. 103-116.

Kadir, A., 2006. Transportasi: Peran dan Dampaknya dalam Pertumbuhan Ekonomi Sosial. *Jurnal Perencanaan & Pengembangan Wilayah Wahana Hijau,* Volume 1, pp. 121-131.

Pakaja, F., Naba, A. & Purwanto, 2012. Peramalan Penjualan Mobil Menggunakan Jaringan Syaraf Tiruan dan Certainty Factor. *EECCISS,* Volume 6, pp. 23-28.

Purba, M., 2015. Rancang Bangun Pengolahan Data Penjualan Sparepart Alat Berat (Hose Hidrolik) pada PT. Sumatra Unggul Menggunakan Visual Basic 6.0. *Jurnal Teknik Informatika Politeknik Sekarayu ,* Volume 2, pp. 73-80.

Rachman, A. S., Cholissodin, I. & Fauzi, M. A., 2018. Peramalan Produksi Gula Menggunakan Metode Jaringan Syaraf Tiruan Backpropagation Pada PG Candi Baru Sidoarjo. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* Volume 2, pp. 1683-1689.

Razak, M. A. & Riksakomara, E., 2017. Peramalan Jumlah Produksi Ikan dengan Menggunakan Backpropagation Neural Network (Studi Kasus UPTD Pelabuhan Perikanan Banjarmasin). *Jurnal Teknik ITS,* Volume 6, pp. 142-148.

Saputra, W. et al., 2017. *Analysis Resilient Algorithm on Artificial Neural Network Backpropagation.* s.l., International Conference on Information and Communication Technology (IconICT).

Sudjarwo, 2013. *Pemeliharaan Mesin Sepeda Motor.* 1 ed. Jakarta: Kementerian Pendidikan & Kebudayaan.

Sundari, S. S., S. & Revianti, W., 2015. *Sistem Peramalan Persediaan Barang Dengan Weight Moving Average Di Toko The Kids 24.* Bali, Konferensi Nasional Sistem & Informatika 2015.

Supit, T. & Jan, A. H., 2015. Analisis Persediaan Bahan Baku pada Industr Mebel di Desa Leilem. *Jurnal EMBA,* Volume 3, pp. 1230-1241.

Tanjung, D. H., 2014-2015. Jaringan Saraf Tiruan dengan Backpropagation untuk Memprediksi Penyakit Asma. *Citec Journal,* Volume 2, pp. 28-38.

Wardah, S. & Iskandar, 2016. Analisis Peramalan Penjualan Produk Keripik Pisang Kemasan Bungkus (Studi Kasus : Home Industry Arwana Food Tembilahan). *Jurnal Teknik Industri,* Volume XI, pp. 135-142.

Widyaningrum, V. T. & Romadhon, A. S., 2014. *Pengaruh Pemberian Momentum pada Artificial Neural Network Backpropagation.* Jakarta, Seminar Nasional Sains dan Teknologi 2014.

Yohannes, E., Mahmudy, W. F. & Rahmi, A., 2015. Penentuan Upah Minimum Kota Berdasarkan Tingkat Inflasi Menggunakan Backpropagation Neural Network (BPNN). *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK),* Volume 2, pp. 34-40.

Yousida, I., 2013. Sistem Akuntansi Persediaan Barang UD Kartika Motor di Banjarmasin. *KINDAI,* Volume 9, pp. 101-113.