ANALISA PERBANDINGAN ALGORITMA PELATIHAN PROPAGASI BALIK DAN ALGORITMA PELATIHAN LEVENBERG-MARQUARDT (STUDI KASUS: PREDIKSI CUACA KOTA MAKASSAR)

Lisye[1], Stefany Yunita Bara'langi[2]
[1,2]Program Studi Teknik Informatika, Fakultas Teknologi Informasi, Universitas Atma Jaya Makassar
Alamat e-mail: lisyelisyee@gmail.com[1], fbaralangi@lecturer.uajm.ac.id[2]

ABSTRACT

Artificial Neural Network can solve a problem based on its experience and training. The purpose of this research is analyzing the comparison of two training that is Backpropagation Training (BP) and Levenberg-Marquardt (LM) Training to determine the speed and accuracy of each training. The case that be used is the weather prediction at Makassar City which is obtained from BMKG. The advantage of BP is the runtime execution to get SSE's value faster than LM that has difference 410.48974 ms. The advantage of LM is can obtain the smallest SSE's value on training process than BP, that is 0.0264.

Keywords: *Artificial Neural Network, Levenberg-Marquardt, weather prediction, Backpropagation*

1. PENDAHULUAN

Jaringan Syaraf Tiruan (JST) adalah algoritma yang memiliki cara kerja mirip dengan otak manusia. JST disebut tiruan karena jaringan syaraf ini diimplementasikan dengan program komputer yang mampu menyelesaikan proses perhitungan (Atiliani, 2013). JST dapat mempelajari pola latihan yang diberikan kepadanya sehingga ketika diberi masalah baru, dapat diselesaikan menggunakan pola latihan tersebut.

JST memiliki nilai tertentu yang menunjukkan seberapa besar koneksi antara *neuron*, nilai ini disebut bobot. JST memerlukan algoritma pelatihan untuk mendapatkan keseimbangan antara kemampuan jaringan untuk menanggapi secara benar pola-pola masukan pada saat pelatihan dan kemampuan untuk memberikan penilaian yang layak dari suatu pola masukan lain yang serupa (Purnamasari, 2013). Proses pelatihan tersebut menghasilkan suatu bobot yang akan digunakan pada masukan yang lain. Tanpa pelatihan yang tepat maka JST tidak akan mampu menyelesaikan masalah-masalah yang kompleks. Oleh karena itulah, JST membutuhkan pelatihan sebagai proses pembelajarannya.

Menurut Utami dan Ulama (2015), algoritma *Backpropagation* atau yang sering disebut dengan Propagasi Balik termasuk metode pelatihan terawasi dimana terdapat pasangan masukan-target serta didesain untuk operasi jaringan *feed forward* multilapis. Algoritma Levenberg-Marquardt (LM) merupakan pengembangan algoritma BP standar untuk mempercepat proses pelatihan dengan teknik optimasi numerik (Retnani, 2013). Algoritma BP memiliki kelemahan dan algoritma LM dikembangkan untuk mengatasi kelemahan algoritma BP namun algoritma LM juga memiliki kelemahan lain. Pemilihan jenis algoritma pelatihan yang tepat pada JST akan sangat mempengaruhi performa JST dari segi tingkat keakuratan hasilnya maupun dari segi kecepatan eksekusi program. Hal tersebut membuat algoritma BP dan algoritma LM layak untuk dilakukan analisa perbandingan.

Adapun studi kasus yang digunakan dalam penelitian ini adalah prediksi cuaca yang bersifat *time series*. Analisa perbandingan yang dilakukan terhadap kedua jenis algoritma pelatihan berdasarkan keakuratan dan kecepatannya. Tingkat keakuratan dinilai dengan melihat nilai SSE (*Sum of Squared Error*) yang dihasilkan kedua jenis algoritma pelatihan (Yuniar *et al.*, 2013). Nilai SSE dipengaruhi oleh beberapa parameter seperti jumlah *neuron*, nilai *learning rate*, nilai parameter Marquardt,

nilai faktor Tau, target *error* dan *epoch*. Semakin kecil nilai SSE maka semakin akurat hasilnya. Kecepatan algoritma juga diuji dalam beberapa penelitian yang dilakukan oleh Nugraha *et al.,* 2013 dan Retnani, 2013. Kecepatan pelatihan diukur dari lamanya waktu eksekusi program untuk mencapai konvergensi dan dapat dilihat dari hasil analisa Big-Oh. Selain itu terdapat penelitian berskala international yang dilakukan oleh Sharma dan Venugopalan pada tahun 2014 dimana penelitian ini menggunakan nilai SSE (keakuratan) dan nilai *epoch* (kecepatan) untuk membandingkan algoritma *Gradient Descent based backpropagation, Gradient Descent with momentum* dan *Resilence backpropogation*.

Pada penelitian ini juga dilakukan pengujian terhadap beberapa parameter yang digunakan pada algoritma pelatihan Propagasi Balik dan algoritma pelatihan Levenberg-marquardt. Pada algoritma pelatihan Propagasi Balik dilakukan uji coba untuk mengetahui pengaruh perubahan jumlah *neuron* lapisan tersembunyi terhadap nilai SSE dan waktu eksekusi program. Selain itu dilakukan juga uji coba perubahan nilai *learning rate* terhadap nilai SSE dan waktu eksekusi program. Pada algoritma pelatihan Levenberg-Marquardt dilakukan uji coba perubahan jumlah *neuron* lapisan tersembunyi dan parameter Marquardt terhadap nilai SSE dan waktu eksekusi program.

2. TINJAUAN PUSTAKA

2.1 Jaringan Syaraf Tiruan

Jaringan Syaraf Tiruan (JST) adalah paradigma pemrosesan suatu informasi yang terinspirasi oleh sistem sel syaraf biologi (Sukarno *et al.,* 2014). Hal yang ingin dicapai dengan melatih JST adalah untuk mencapai keseimbangan antara kemampuan memorisasi dan generalisasi. Yang dimaksud kemampuan memorisasi adalah kemampuan JST untuk mengambil kembali secara sempurna sebuah pola yang telah dipelajari. Kemampuan generalisasi adalah kemampuan JST untuk menghasilkan respon yang bisa diterima terhadap pola-pola yang sebelumnya telah dipelajari.

JST memiliki masalah utama yaitu pada proses pelatihan dan pembentukan model jaringan yang lama. Oleh karena itu diperlukan pemilihan konfigurasi jaringan yang tepat seperti pemilihan jumlah lapis tersembunyi, *neuron*, nilai momentum, *learning-rate* dan fungsi aktivasi. Algoritma pelatihan juga sangat mempengaruhi hasil dari JST nantinya, maka pemilihan algoritma pelatihan yang tepat sangat dibutuhkan.

Fungsi yang digunakan pada penelitian ini yaitu sigmoid biner karena memiliki nilai fungsi yang terletak antara 0 dan 1 sehingga sangat mudah untuk didiferensialkan (Yulianti, 2014). Fungsi aktivasi ini yang akan digunakan pada penelitian karena berdasarkan penelitian yang dilakukan oleh Redjeki pada tahun 2014 memperoleh bahwa tingkat rata-rata akurasi aktivasi sigmoid biner lebih baik dibandingkan dengan sigmoid bipolar.

$$f(x) = \frac{1}{1+e^{-x}} \qquad (1)$$

2.2 Propagasi Balik

Propagasi Balik atau *Backpropagation* (BP) merupakan suatu teknik pembelajaran atau pelatihan *supervised learning* yang paling banyak digunakan (Agustin, 2012). Metode ini merupakan salah satu metode yang sangat baik dalam menangani masalah pengenalan pola-pola kompleks. Secara rinci algoritma pelatihan jaringan backpropagation dapat diuraikan sebagai berikut :

1. Inisialisasi bobot (setiap bobot diberi nilai acak antara 0 – 1)
2. Selama kondisi berhenti belum terpenuhi lakukan langkah 3 – 4
3. Lakukan Langkah 4 – 14 sebanyak jumlah pelatihan yang diinginkan
4. Untuk keluaran lapisan dan setiap lapisan tersembunyi lakukan langkah 5 – 14
5. Hitung masukan setiap *node* pada lapisan tersembunyi

   $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i * v_{ij}$ (2)

   dimana: z_in_j = jumlah total masukan untuk *node* ke-j

   n = jumlah *node* pada lapisan tersembunyi sebelumnya, untuk lapisan tersembunyi yang pertama n adalah jumlah *node* masukan

   x_i = nilai *node* ke-i yang dikeluarkan oleh lapisan tersembunyi sebelumnya, untuk lapisan tersembunyi yang pertama xi adalah masukan yang diterima oleh sistem

w_{ij} = bobot yang menghubungkan antara *node* ke-i dan *node* ke-j

6. Hitung keluaran setiap *node* pada lapisan tersembunyi dengan fungsi aktivasi:
   $z_j = f(z\_in_j)$ *(3)*
   dimana: z_j = keluaran dari node ke-j
7. Hitung masukan setiap *node* pada keluaran lapisan
   $y\_in_k = w_{0k} + \sum_{i=1}^{p} z_i * w_{jk}$ *(4)*
   dimana: y_in_k = jumlah total masukan untuk *node* ke-k
   n = jumlah node pada lapisan tersembunyi sebelum keluaran lapisan
   w_{jk}= bobot yang menghubungkan antara *node* ke-j dan *node* ke-k
8. Hitung keluaran pada setiap *node* pada keluaran lapisan
   $y_k = f(y\_in_k)$ *(5)*
   dimana: y_k = keluaran dari *node* ke-k
9. Hitung *error* setiap *node* pada keluaran lapisan dengan fungsi deaktivasi.
   $\delta_k = (t_k - y_k) * f'(y\_in_k)$ *(6)*
   dimana: δ_k= *error* pada *node* ke-k
   t_k = target ke-k
10. Hitung perubahan bobot pada setiap *node* di setiap keluaran lapisan
    $\Delta w_{jk} = \alpha * \delta_k * z_j$ *(7)*
    dimana: Δw_{jk} = perubahan bobot yang menghubungkan *node* ke-j dan *node* ke-k
    α = *learning rate* yang merupakan nilai antara 0 – 1.
    Perubahan bias dengan rumus berikut,
    $\Delta w_{0k} = \alpha * \delta_k$ *(8)*
11. Hitung *error* setiap *node* pada lapisan tersembunyi dengan fungsi deaktivasi
    $\delta_j = (\sum_{k=1}^{m} \delta_k * w_{jk}) * f'(z\_in_j)$ *(9)*
    dimana: δ_j = *error* pada *node* ke-j
12. Hitung perubahan bobot pada setiap *node* pada setiap lapisan tersembunyi
    $\Delta v_{ij} = \alpha * \delta_j * x_i$ *(10)*
    dimana: Δv_{ij} = perubahan bobot yang menghubungkan *node* ke-i dan *node* ke-j
    Hitung perubahan bias dengan rumus berikut,
    $\Delta v_{0j} = \alpha * \delta_j$ *(11)*
13. *Update* bobot pada setiap *node* pada keluaran lapisan
    w_{jk} *(new)* = w_{jk} *(old)*+Δw_{jk} *(12)*
14. *Update* bobot pada setiap *node* pada setiap lapisan tersembunyi
    v_{ij} *(new)* = v_{ij} *(old)*+Δv_{ij} *(13)*

2.3 Levenberg-Marquardt

Antiliani (2013), Algoritma Levenberg-Marquardt merupakan salah satu jenis dari algoritma pelatihan jaringan syaraf tiruan *Backpropagation* dengan dua jenis perhitungan, yakni perhitungan maju dan perhitungan mundur. Parameter yang digunakan pada pelatihan LM terdiri dari parameter *Levenberg-Marquardt* (η), parameter faktor Tau (τ) dan maksimum *epoch*. Perhitungan Levenberg-Marquardt dijelaskan sebagai berikut:

1. Inisialiasasi bobot dan bias dengan bilangan acak, *epoch* maksimum, dan target minimal (target biasanya dihitung dengan menggunakan *Sum of Squared Error*/SSE).
2. Menentukan parameter yang dibutuhkan, antara lain:
   a. Inisialisasi *epoch* = 0
   b. Parameter *Levenberg-Marquardt* (η) yang nilainya harus lebih besar dari nol.
   c. Parameter faktor Tau (τ) yang digunakan sebagai parameter yang dikalikan atau dibagi dengan parameter *Levenberg-Marquardt*.
3. Perhitungan *Feedforward*/tiap-tiap unit masukan (X_i, i = 1, 2, 3, ..., n) menerima sinyal masukan dan meneruskan sinyal tersebut ke semua unit pada lapisan tersembunyi. Tiap-tiap unit lapisan tersembunyi (Z_j, j = 1, 2, ..., p) menjumlahkan sinyal-sinyal masukan berbobot (v_{ij}) dan bias ($b1_j$).
   $z_{in_j} = b1_j + \sum_{i=1}^{n} x_i v_{ij}$ *(14)*
   Gunakan fungsi aktivasi untuk menghitung sinyal keluaran.
   $z_j = f(z\_in_j)$ *(15)*
   Kemudian kirimkan sinyal tersebut ke semua unit di lapisan atasnya.
4. Tiap-tiap unit lapisan keluaran (Y_k, k = 1, 2, 3, ..., m) menjumlahkan sinyal-sinyal masukan berbobot (w_{jk}) dan bias ($b2_k$).
   $y_{in_k} = b2_k + \sum_{i=1}^{p} z_i w_{jk}$ *(16)*
   Gunakan fungsi aktivasi untuk menghitung sinyal keluaran.
   $y_k = f(y\_in_k)$ *(17)*
   Kemudian kirimkan sinyal tersebut ke semua unit di lapisan atasnya.
5. Menghitung *error* dan SSE. Rumus untuk *error*:
   $e_r = t_r - y_r$ *(18)*
   r = masukan ke-r

Rumus untuk menghitung SSE:

$SSE = \sum(e_r)^2$ *(19)*

6. Hitung *error neuron* untuk tiap unit lapisan keluaran (Y_k, k=1,2,3, ..., m)

$\delta 2_k = (t_r - y_r) f'(y_{in_k})$ *(20)*

$\varphi 2_{jk} = \delta 2_k z_j$ *(21)*

$\beta 2_k = \delta 2_k$ *(22)*

Kemudian hitung koreksi bobot untuk memperbaiki nilai W_{jk}

$\Delta w_{jk} = \varphi 2_{jk}$ *(23)*

Hitung juga koreksi bias untuk memperbaiki nilai $\beta 2_k$:

$\Delta b2_k = \beta 2_k$ *(24)*

7. Hitung *error neuron* untuk tiap unit lapisan tersembunyi (Z_j, j=1,2,3,... p)

$\delta in_j = \sum_{k=1}^{m} \delta 2_k w_{jk}$ *(25)*

$\delta 1_J = \delta_{in_j} f'\left(z_{in_j}\right) 1 - \left(z_{in_j}\right)$ *(26)*

$\varphi 1_{ij} = \delta 1_j x_j$ *(27)*

$\beta 1_j = \delta 1_j$ *(28)*

Kemudian hitung koreksi bobot (yang nantinya digunakan untuk memperbaiki nilai V_{ij} yaitu :

$\Delta V_{ij} = \varphi 1_{ij}$ *(29)*

Hitung juga koreksi bias yang nantinya digunakan untuk memperbaiki $b1_j$:

$\Delta b1_j = \beta 1_j$ *(30)*

8. Membentuk matriks jacobian J(x). X merupakan matriks yang berisi nilai bobot dan bias dari keseluruhan jaringan.

$j\ [\varphi 1_{11} \ldots \varphi 1_{np} \beta 1_1 \ldots\ \beta 1_p \varphi 2_{11} \ldots\ \varphi 2_{pm} \beta 2_1 \ \ldots \beta 2_m]$ *(31)*

9. Menghitung bobot baru

$w_{baru} = w_{lama} - [J^T j + \eta\ I]^{-} J^T e$ *(32)*

10. Menghitung SSE

Jika $SSE_{baru} <= SSE_{lama}$, maka

a. $\eta = \eta / \tau$ *(33)*

b. $epoch = epoch + 1$ *(34)*

c. Kembali ke langkah 3

Jika $SSE_{baru} > SSE_{lama}$, maka

a. $\eta = \eta * \tau$ *(35)*

b. Kembali ke langkah 9

11. Proses pelatihan berhenti jika *epoch* ≥ *epoch* maksimal atau *error* ≤ target *error*.

3. METODOLOGI PENELITIAN

Metode perancangan yang akan digunakan adalah metode terstruktur. Penulis memilih metode terstruktur karena metode ini memiliki cara kerja yang sistematis dan sesuai dengan penelitian yang dilakukan oleh penulis. Tahapan-tahapan dalam metode ini adalah:

1. Tahap Pengumpulan Dataset. Pada tahap ini dilakukan pengumpulan data-data yang akan digunakan dalam penelitian. Data ini terdiri dari data suhu udara, tekanan udara, kecepatan angin, kelembaban udara dan curah hujan. Data ini diperoleh dari BMKG Kota Makassar.
2. Tahap Perancangan Arsitekur JST. Tahap ini meliputi:
   a. Pembentukan arsitektur jaringan syaraf tiruan, seperti pembentukan jumlah lapisan dan *neuron*.
   b. Penentuan nilai bobot, bias, dan *learning rate* awal.
3. Tahap *Training* Algoritma. Pada tahap ini dilakukan proses *training* algoritma Propagasi Balik dan algoritma Levenberg-Marquardt untuk memperoleh bobot yang diinginkan.
4. Tahap *Testing* Algoritma. Pada tahap ini dilakukan proses *testing* dengan menggunakan bobot-bobot yang dihasilkan dari proses *training*.
5. Tahap Implementasi. Tujuannya untuk menguji dan mengimplementasikan program yang telah dibuat pada tahap sebelumnya.
6. Tahap Pengujian. Tahap ini dilakukan pengujian kedua jenis algoritma pelatihan dan menganalisa pengujian kedua algoritma tersebut.

4. HASIL DAN PEMBAHASAN

4.1 Dataset

Pada tahap awal ini, penulis mengumpulkan data masukan berupa data bulanan curah hujan, suhu udara, tekanan udara, kecepatan angin dan kelembaban udara. Data-data tersebut diperoleh dari Balai Besar Meteorologi, Klimatologi, dan Geofisika Wilayah IV Makassar berupa data tahun 2006 hingga tahun 2016. Data bulanan yang diperoleh penulis merupakan data yang diukur setiap hari lalu dirata-ratakan membentuk data bulanan. Adapun beberapa contoh dari data-data tersebut sebagai berikut:

Tabel 1 Curah hujan

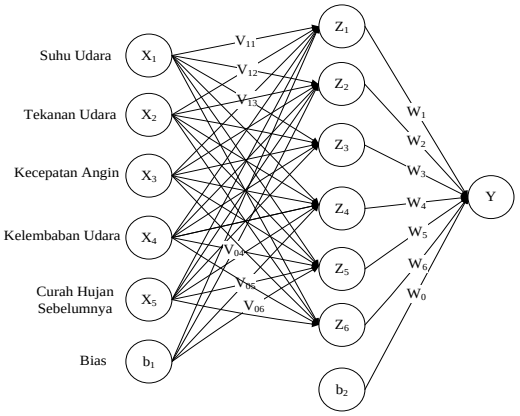
Tahun	Bulan											
	Jan	Feb	Mar	Apr	Mei	Jun	Jul	Aug	Sep	Okt	Nov	Des
2006	587	649	353	265	26	137	1	0	0	0	20	445
2007	693	486	283	197	36	130	4	3	-	16	215	869
2008	662	881	315	77	61	35	58	4	6	74	409	764

2009	955	740	197	72	50	36	41	0	-	16	119	474
2010	873	429	279	230	144	124	100	57	231	223	238	760
2011	562	529	595	386	162	8	1	0	-	40	183	858
2012	520	372	639	78	208	36	69	-	-	11	71	448
2013	982	418	336	270	137	275	94	1	2	24	203	675
2014	836	313	311	282	105	134	30	6	0	0	117	673
2015	962	355	306	204	9	55	0	0	0	0	148	619
2016	385	727	224	121	44	47	14	-	79	425	150	547

Data-data yang diperoleh penulis lalu dinormalisasi untuk menghasilkan nilai 0-1 menggunakan Normalisasi *Min-Max*. Selain data curah hujan, terdapat pula data suhu udara, tekanan udara, kecepatan angin dan kelembaban udara yang dinormalisasi. Data-data yang telah dinormalisasi tersebut akan digunakan sebagai masukan pada proses *training* dan *testing* algoritma Propagasi Balik dan algoritma Levenberg-Marquardt. Data *training* digunakan sebagai data latihan untuk melatih kedua algoritma agar dapat mengenali pola yang ada dan memperoleh bobot yang sesuai. Data *testing* digunakan sebagai data percobaan untuk mengetahui sebagus apa bobot yang didapatkan dari proses *training*. Data *training* diambil sebanyak 70% dari total data yang ada. Proses *training* menggunakan 88 data curah hujan yang diambil dari Januari 2006 hingga Desember 2010 dan Januari 2013 hingga Juni 2015. Proses *training* juga menggunakan 87 data suhu udara, tekanan udara, kecepatan angin, kelembaban udara yang diambil dari Februari 2006 hingga Desember 2010 dan Januari 2013 hingga Juni 2015. Data *testing* diambil sebanyak 30% dari total data yang ada. Proses *testing* menggunakan 38 data curah hujan, suhu udara, tekanan udara, kecepatan angin dan kelembaban udara yang diambil dari sisa data yang ada yang tidak digunakan pada proses *training*.

4.2 Perancangan Arsitektur Jaringan Syaraf Tiruan

Pada tahap proses perencanaan, dirancang arsitektur jaringan syaraf tiruan yang akan digunakan dalam proses *training*. Pada perancangan arsitektur jaringan syaraf tiruan, penulis menggunakan 3 lapisan yaitu sebuah lapisan masukan, sebuah lapisan tersembunyi dan sebuah lapisan keluaran.



Gambar 1 Arsitektur JST

Pada Gambar 1 terdapat bobot yang menghubungkan antar *neuron*. Bobot yang menghubungkan *neuron* lapisan masukan dan *neuron* lapisan tersembunyi disimbolkan dengan V_{11} dan seterusnya. Bobot yang menghubungkan *neuron* lapisan tersembunyi dan *neuron* lapisan keluaran disimbolkan dengan W_1 dan seterusnya. Fungsi aktivasi yang digunakan pada kedua algoritma *training* yaitu fungsi aktivasi sigmoid biner.

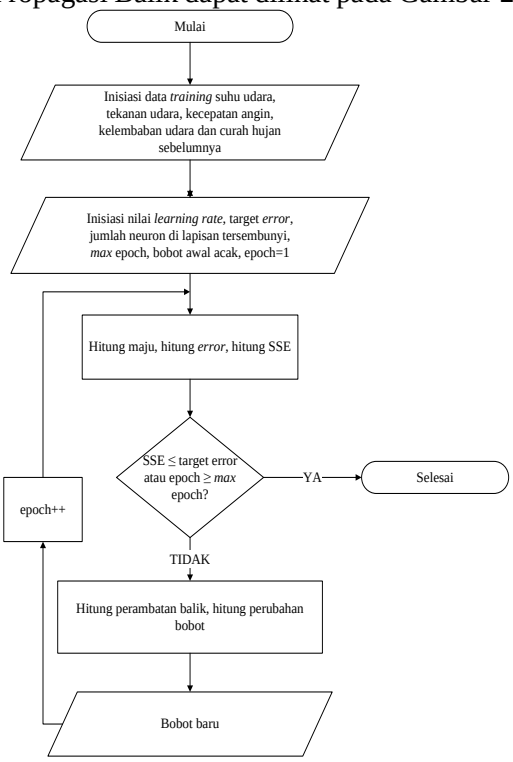
4.3 Perancangan Proses *Training* Algoritma Propagasi Balik dan Algoritma Levenberg-Marquardt

Proses *training* algoritma Propagasi Balik dan algoritma Levenberg-Marquardt dilakukan untuk mendapatkan bobot yang optimal berdasarkan *error* yang diinginkan. Selain itu, proses *training* bertujuan mendapatkan arsitektur jaringan dan parameter yang sesuai untuk menghasilkan nilai SSE terkecil. Proses *training* ini menggunakan 88 data curah hujan, 87 data suhu udara, tekanan udara, kecepatan angin, dan kelembaban udara.

4.3.1 Proses *Training* Algoritma Propagasi Balik

Proses *training* dengan keluaran berupa curah hujan ini menggunakan SSE dan jumlah *epoch* sebagai syarat berhentinya proses dan bobot terakhir yang diperoleh merupakan bobot optimal. Proses *training* akan berhenti apabila SSE yang diperoleh lebih kecil daripada target *error* yang ditentukan di awal. Selain itu, proses *training* juga akan berhenti apabila jumlah *epoch* telah melebihi maksimal *epoch*. *Epoch* dihitung ketika semua data masukan pada proses *training* telah digunakan dan akan diulang

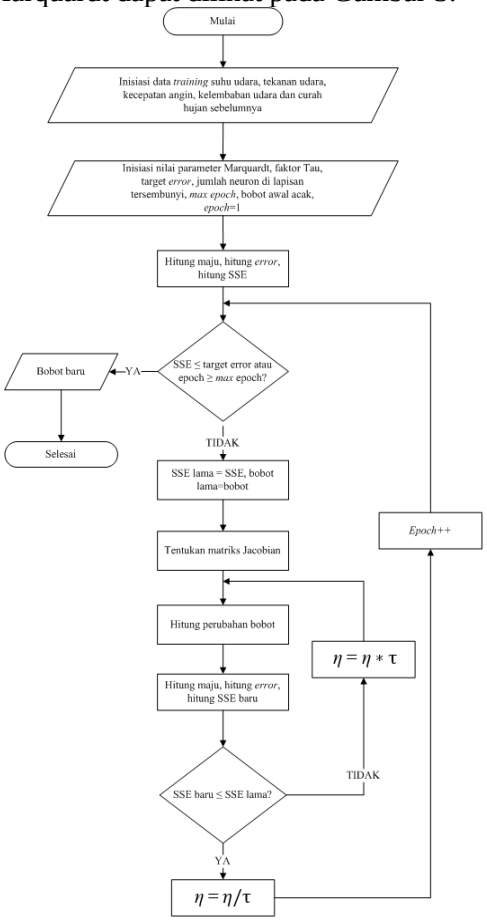
mulai dari data pertama lagi lalu *epoch* bertambah satu. Beberapa parameter yang mempengaruhi nilai SSE pada algoritma Propagasi Balik yaitu jumlah *neuron*, nilai *learning rate*, target *error* dan jumlah *epoch*. Pada proses *training*, kedua algoritma pelatihan akan diukur waktunya dalam satuan *milisecond* (ms). Parameter waktu akan dijadikan salah satu dari elemen penguji selain nilai SSE. Parameter waktu dalam pengujian tentunya sangat bergantung pada spesifikasi perangkat laptop yang digunakan penulis dalam melakukan proses *training* kedua algoritma pelatihan tersebut. Fungsi aktivasi yang digunakan pada proses *training* adalah fungsi aktivasi sigmoid biner. Secara umum gambaran proses *training* algoritma Propagasi Balik dapat dilihat pada Gambar 2.



Gambar 2 Proses *training* algoritma Propagasi Balik

4.3.2 Proses *Training* Algoritma Levenberg-Marquardt

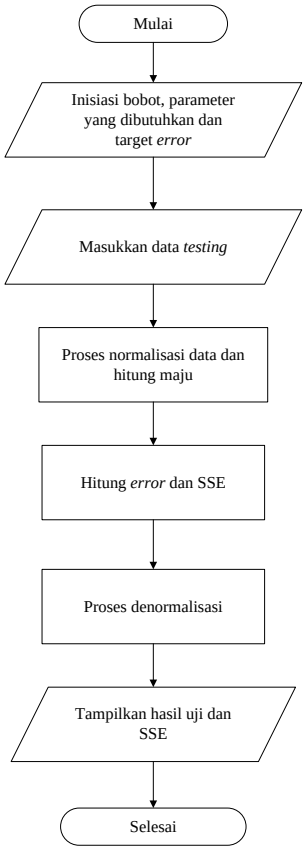
Proses *training* pada algoritma Levenberg-Marquardt bertujuan untuk melatih data dan melakukan pengenalan pola sehingga dapat memperoleh bobot optimal berdasarkan *error* yang diinginkan. Proses *training* ini menggunakan SSE dan jumlah *epoch* sebagai syarat berhentinya proses. Beberapa parameter yang mempengaruhi nilai SSE yaitu jumlah *neuron*, nilai parameter Marquardt, nilai faktor Tau, target *error* dan jumlah *epoch*. Proses *training* algoritma Levenberg-Marquardt hampir serupa dengan proses *training* algoritma Propagasi Balik dalam hal syarat berhentinya proses. Namun langkah-langkah dan rumus yang digunakan dalam memperoleh bobot optimal berbeda. Secara umum gambaran proses *training* algoritma Levenberg-Marquardt dapat dilihat pada Gambar 3.



Gambar 3 Proses *training* algoritma Levenberg-Marquardt

4.4 Proses *Testing* Algoritma Propagasi Balik dan Algoritma Levenberg-Marquardt

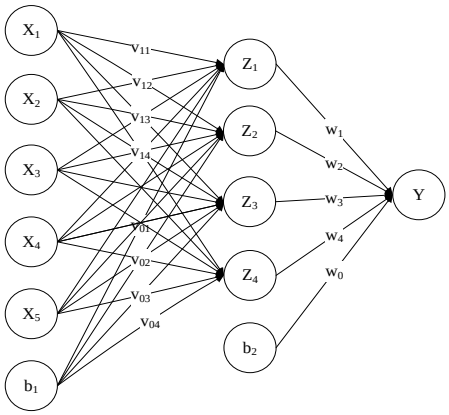
Proses *testing* bertujuan untuk mengetahui tingkat keakuratan bobot-bobot yang dihasilkan dari proses *training*. Nilai curah hujan yang akan dihasilkan pada proses *testing* akan diuji keakuratannya dengan menghitung selisih antara target dengan nilai curah hujan yang dihasilkan. Semakin kecil nilai selisih yang diperoleh maka semakin tinggi keakuratannya, demikian juga sebaliknya. Secara umum, proses *testing* dapat dilihat pada Gambar 4.



Gambar 4 Proses *testing*

4.5 Tahap Implementasi

Tahap ini bertujuan untuk mengimplementasikan program yang telah dirancang ke dalam studi kasus yang digunakan. Program dibuat menggunakan *software Netbeans 6.7* dengan bahasa pemograman *Java*. Penyimpanan data pada program menggunakan *array*. Penulis akan melakukan proses perhitungan manual pada tahap *training* algoritma Propagasi Balik dan algoritma Levenberg-Marquardt. Proses *training* menggunakan arsitektur jaringan seperti Gambar 5.



Gambar 5 Arsitektur jaringan implementasi proses *training*

Pada tahap *training* algoritma Propagasi Balik memiliki langkah-langkah sebagai berikut:

1. Penentuan parameter dan bobot awal yang akan digunakan pada proses *training*.
   Jumlah *neuron* pada lapisan tersembunyi: 4
   Nilai *learning rate* (α): 1
   Maksimum *epoch* : 1000
   Target *error* : 0.01
   Setelah itu akan ditentukan bobot awal setiap *neuron* dan bias pada lapisan masukan dan lapisan tersembunyi secara acak dengan *range* 0-1.
2. Tentukan data masukan yang akan digunakan dalam proses *training*. Penulis akan menggunakan 3 data yang sudah dinormalisasi sebagai percobaan.
3. Data pertama akan dilakukan proses hitung maju menggunakan Persamaan 2. Pada proses ini langkah pertama akan dihitung sinyal input yang masuk pada lapisan tersembunyi.
   z_in_1 = 0.002083561 + 0.322580645*0.757155337 + 0.41509434*0.88397507 + 0.8*0.838776825 + 0.833333333*0.58899791 + 0.597759674*0.474556173 = 2.058783861
   z_in_2 = 0.617413623 + 0.322580645*0.610795122 + 0.41509434*0.213190484 + 0.8*0.296751482 + 0.833333333*0.83781681 + 0.597759674*0.690430413 = 2.251231789
   z_in_3 = 0.94075132 + 0.322580645*0.60357807 + 0.41509434*0.85238618 + 0.8*0.21383946 + 0.833333333*0.86371502 + 0.597759674*0.84073685 = 2.882667266
   z_in_4 = 0.863151084 + 0.322580645*0.016459027 + 0.41509434*0.581409664 + 0.8*0.475967543 + 0.833333333*0.103984919 + 0.597759674*0.526763849 = 1.892106628
   Selanjutnya akan dihitung keluaran dari setiap *neuron* lapisan tersembunyi dengan fungsi aktivasi menggunakan Persamaan

3. Nilai e adalah konstanta matematika yang bernilai 2.71828182845904523536.

$z_1 = \frac{1}{1+e^{-z\_in_1}} = 0.8866832175$

$z_2 = \frac{1}{1+e^{-z\_in_2}} = 0.904756734$

$z_3 = \frac{1}{1+e^{-z\_in_3}} = 0.946982937$

$z_4 = \frac{1}{1+e^{-z\_in_4}} = 0.86899554$

Setelah itu hitung sinyal input yang masuk ke lapisan keluaran menggunakan Persamaan 4.

y_in = 0.356279169+0.8866832175*0.310081314+0.904756734*0.202613717 + 0.946982937*0.348562118 + 0.86899554*0.679713494 = 1.735335753

Lalu hitung keluaran dari *neuron* lapisan keluaran menggunakan Persamaan 5.

$y = \frac{1}{1+e^{-y\_in}} = 0.85009365$

4. Jika sudah mendapat nilai dari lapisan keluaran maka dapat dihitung *error* dan SSE. Hitung *error* dan SSE menggunakan Persamaan 18 dan 19.

   Error = 0.66089613 - 0.85009365 = -0.18919752

   $SSE = (-0.18919752)^2 = 0.035795702$
5. Lakukan pengecekan apakah SSE yang diperoleh lebih kecil sama dengan target *error*. SSE yang diperoleh tidak lebih kecil sama dengan 0.01 sehingga akan dilakukan tahap perambatan balik untuk memperbaiki bobot.
6. Tahap perambatan balik menggunakan Persamaan 6 hingga 11. Hitung *error* tiap *neuron* pada lapisan keluaran dengan Persamaan 6.

   δ = *error**y*(1-y) = -0.18919752*0.85009365*(1-0.85009365) = -0.024110279

   Kemudian hitung perubahan bobot pada *neuron* yang menghubungkan lapisan tersembunyi dan lapisan keluaran dengan Persamaan 7 dan 8.

   $\Delta w_1 = \alpha * \delta * z_1 = -0.021381771$

   $\Delta w_2 = \alpha * \delta * z_2 = -0.021813938$

   $\Delta w_3 = \alpha * \delta * z_3 = -0.022832023$

   $\Delta w_4 = \alpha * \delta * z_4 = -0.020951725$

   $\Delta w_0 = \alpha * \delta = -0.024110279$

   Setelah itu hitung *error* tiap *neuron* pada lapisan tersembunyi menggunakan Persamaan 9.

   $\delta_{in1} = \delta * w_1 * z_1 * (1 - z_1) = -0.000750313$

   $\delta_{in2} = \delta * w_2 * z_2 * (1 - z_2) = -0.000420956$

   $\delta_{in3} = \delta * w_3 * z_3 * (1 - z_3) = -0.00042193$

   $\delta_{in4} = \delta * w_4 * z_4 * (1 - z_4) = -0.001865657$

   Lalu hitung perubahan bobot tiap *neuron* yang menghubungkan lapisan masukan dan lapisan tersembunyi menggunakan Persamaan 10 dan 11.

   $\Delta v_{11} = \alpha * \delta_1 * x_1 = -0.000242036$

   Demikian juga dengan Δv_{12}, Δv_{13}, Δv_{14}.

   $\Delta v_{21} = \alpha * \delta_1 * x_2 = -0.000311451$

   Demikian juga dengan Δv_{22}, Δv_{23}, Δv_{24}.

   $\Delta v_{31} = \alpha * \delta_1 * x_3 = -0.00060025$

   Demikian juga dengan Δv_{32}, Δv_{33}, Δv_{34}.

   $\Delta v_{41} = \alpha * \delta_1 * x_4 = -0.000625261$

   Demikian juga dengan Δv_{42}, Δv_{43}, Δv_{44}.

   $\Delta v_{51} = \alpha * \delta_1 * x_5 = -0.000448507$

   Demikian juga dengan Δv_{52}, Δv_{53}, Δv_{54}.

   $\Delta v_{01} = \alpha * \delta_1 = -0.000750313$

   Demikian juga dengan Δv_{02}, Δv_{03}, Δv_{04}.
7. Hitung bobot baru berdasarkan perubahan bobot yang diperoleh dengan menggunakan Persamaan 12 dan 13.

   $v_{11(baru)} = v_{11(lama)} + \Delta v_{11} = 0.756913301$

   Demikian juga dengan $v_{12(baru)}$, $v_{13(baru)}$, $v_{14(baru)}$, $v_{21(baru)}$, $v_{22(baru)}$, $v_{23(baru)}$, $v_{24(baru)}$, $v_{31(baru)}$, $v_{32(baru)}$, $v_{33(baru)}$, $v_{34(baru)}$, $v_{41(baru)}$, $v_{42(baru)}$, $v_{43(baru)}$, $v_{44(baru)}$, $v_{51(baru)}$, $v_{52(baru)}$, $v_{53(baru)}$, $v_{54(baru)}$, $v_{01(baru)}$, $v_{02(baru)}$, $v_{03(baru)}$, $v_{04(baru)}$.

   $w_{1(baru)} = w_{1(lama)} + \Delta w_1 = 0.288699543$

   Demikian juga dengan $w_{2(baru)}$, $w_{3(baru)}$, $w_{4(baru)}$, $w_{0(baru)}$.
8. Untuk data kedua dilakukan operasi yang sama dengan data pertama. Bobot dan bias awal menggunakan nilai bobot dan bias baru yang didapatkan dari hasil perhitungan data pertama. Demikian seterusnya hingga data ketiga (1 *epoch*). Proses *training* ini akan terus berlanjut hingga syarat berhenti terpenuhi yaitu SSE lebih kecil sama dengan target *error* atau jumlah *epoch* lebih dari 1000.
9. Pada perhitungan data kedua, diperoleh SSE sebesar 0.22486826358. Nilai SSE dan jumlah *epoch* belum sesuai dengan syarat berhentinya proses *training*. Perhitungan data kedua ini menghasilkan bobot dan bias baru.
10. Proses *training* dilanjutkan dengan perhitungan data ketiga namun belum juga mencapai syarat terhentinya proses *training*. Ketika data ketiga telah dihitung maka *epoch* akan bertambah satu menjadi dua. Lalu kembali menggunakan data pertama dalam perhitungan maju. Data pertama ini telah mencapai SSE yang diharapkan yaitu 0.00489195093884142.

SSE tersebut lebih kecil dari 0.01 dan jumlah *epoch* tidak lebih dari 1000 sehingga proses *training* selesai dan mendapatkan bobot yang optimal.

Setelah perhitungan manual untuk proses *training* algoritma Propagasi Balik selesai, selanjutnya perhitungan manual untuk proses *training* algoritma Levenberg-Marquardt. Langkah-langkah proses *training* untuk algoritma Levenberg-Marquardt dapat diuraikan sebagai berikut:

1. Penentuan parameter dan bobot awal yang akan digunakan pada proses *training*.
   Jumlah *neuron* padalapisan tersembunyi: 4
   Nilai parameter Marquardt (η) : 0.1
   Faktor tau (τ) : 10
   Maksimum *epoch* : 2
   Target *error* : 0.01
   Setelah itu akan ditentukan bobot awal setiap *neuron* dan bias pada lapisan masukan dan lapisan tersembunyi secara acak dengan *range* 0-1.
2. Tentukan data masukan yang akan digunakan dalam proses *training*. Penulis akan menggunakan 3 data yang sudah dinormalisasi sebagai percobaan.
3. Data pertama akan dilakukan proses hitung maju menggunakan Persamaan 14. Pada proses ini langkah pertama akan dihitung sinyal input yang masuk pada lapisan tersembunyi.
   z_in_1 = 0.002083561 + 0.322580645*0.757155337 + 0.41509434*0.88397507 + 0.8*0.838776825 + 0.833333333*0.58899791 + 0.597759674*0.474556173 = 2.058783861
   z_in_2 = 0.617413623 + 0.322580645*0.610795122 + 0.41509434*0.213190484 + 0.8*0.296751482 + 0.833333333*0.83781681 + 0.597759674*0.690430413 = 2.251231789
   z_in_3 = 0.94075132 + 0.322580645*0.60357807 + 0.41509434*0.85238618 + 0.8*0.21383946 + 0.833333333*0.86371502 + 0.597759674*0.84073685 = 2.882667266
   z_in_4 = 0.863151084 + 0.322580645*0.016459027 + 0.41509434*0.581409664 + 0.8*0.475967543 + 0.833333333*0.103984919 + 0.597759674*0.526763849 = 1.892106628
   Selanjutnya akan dihitung keluaran dari setiap *neuron* lapisan tersembunyi dengan fungsi aktivasi menggunakan Persamaan 15. Nilai e adalah konstanta matematika yang bernilai 2.718281828459045235360287471352.
   $z_1 = \frac{1}{1+e^{-z\_in_1}} = 0.8866832175$
   $z_2 = \frac{1}{1+e^{-z\_in_2}} = 0.904756734$
   $z_3 = \frac{1}{1+e^{-z\_in_3}} = 0.946982937$
   $z_4 = \frac{1}{1+e^{-z\_in_4}} = 0.86899554$
   Setelah itu hitung sinyal input yang masuk ke lapisan keluaran menggunakan Persamaan 16.
   y_in = 0.356279169+0.8866832175*0.310081314+0.904756734* 0.202613717 + 0.946982937 * 0.348562118 + 0.86899554 * 0.679713494 = 1.735335753
   Lalu hitung keluaran dari *neuron* lapisan keluaran menggunakan Persamaan 17.
   $y = \frac{1}{1+e^{-y\_in}} = 0.85009365$
4. Jika sudah mendapat nilai dari lapisan keluaran maka dapat dihitung *error* dan SSE. Hitung *error* dan SSE menggunakan Persamaan 17 dan 18.
   Error = 0.66089613 - 0.85009365 = -0.18919752
   SSE = $(-0.18919752)^2$ = 0.035795702
5. Setelah mendapatkan SSE maka dibandingkan denga target *error*. SSE tersebut masih lebih besar daripada target *error* yang ditetapkan di awal proses *training* sehingga dilanjutkan tahap pembentukan matriks Jacobian. Untuk mendapatkan perubahan bobot, buat matriks Jacobian menggunakan Persamaan 20 hingga 31. Matriks Jacobian ini akan digunakan untuk mendapatkan bobot baru. Ukuran matriks Jacobian yang terbentuk adalah 1x29.
   $J = [-2.4203632768122409E-4 \ \ldots \ -0.024110279297537545]_{1x29}$
6. Langkah selanjutnya setelah mendapat matriks Jacobian adalah menghitung perubahan bobot menggunaka Persamaan 32. Pertama, buat *transpose* dari matriks Jacobian.

$$J^{T} = \begin{bmatrix} -2.42036327681E-4 \\ -1.3579240985E-4 \\ \vdots \\ -0.0241102792975 \end{bmatrix}_{29x1}$$

7. Setelah itu *transpose* dari matriks Jacobian dikalikan dengan matriks Jacobian menghasilkan matriks berikut:
$J^{T} J =$
$$\begin{bmatrix} 5.8581583917E-8 & \cdots & 5.8355634605E-6 \\ 3.2866696208E-8 & \cdots & 3.27399292815E-6 \\ 3.29426937202E-8 & \cdots & 3.28156336703E-6 \\ \vdots & \ddots & \vdots \\ 5.8355634605E-6 & \cdots & 5.8130556780E-4 \end{bmatrix}_{29x29}$$
8. Setelah itu buat matriks identitas sesuai ordo hasil kali *transpose* matriks Jacobian dengan matriks Jacobian. Matriks identitas sebagai berikut:
$$I = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{29x29}$$
9. Kemudian matriks identitas dikalikan dengan parameter Marquardt menjadi matriks berikut ini:
$$\eta I = \begin{bmatrix} 0.1 & 0 & 0 & \cdots & 0 \\ 0 & 0.1 & 0 & \cdots & 0 \\ 0 & 0 & 0.1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}_{29x29}$$
10. Lalu tambahkan hasil tersebut dengan perkalian *transpose* Jacobian dan Jacobian, hasilnya sebagai berikut:
$J^{T} J + \eta I =$
$$\begin{bmatrix} 0.1000000585 & 3.28666E-8 & \cdots & 5.83556E-6 \\ 3.2866696E-8 & 0.100000018 & \cdots & 3.273992E-6 \\ 3.29426937E-8 & 1.84822E-8 & \cdots & 3.28156E-6 \\ \vdots & \vdots & \ddots & \vdots \\ 5.835563E-6 & 3.27399E-6 & \cdots & 0.100581 \end{bmatrix}_{29x29}$$
11. Setelah mendapat hasil penjumlahan, maka dilakukan perhitungan *invers*, menghasilkan matriks berikut ini:
$(J^{T} J + \eta I)^{-1} =$
$$\begin{bmatrix} 9.9999942 & -3.2068E-6 & \cdots & -5.6939E-4 \\ -3.20689E-6 & 9.9999982 & \cdots & -3.1945E-4 \\ -3.21430E-6 & -1.8033E-6 & \cdots & -3.2019E-4 \\ \vdots & \vdots & \ddots & \vdots \\ -5.69391E-4 & -3.1945E-4 & \cdots & 9.94328043 \end{bmatrix}_{29x29}$$
12. Hasil perhitungan *invers* kemudian dikalikan dengan *transpose* matriks Jacobian, hasilnya sebagai berikut:
$(J^{T} J + \eta I)^{-1} J =$
$$\begin{bmatrix} -0.0023616142020403286 \\ -0.0013249634330595385 \\ -0.0013280271399452755 \\ \vdots \\ -0.2352505450306442 \end{bmatrix}_{29x1}$$
13. Hasil perkalian pada tahap ke 12 kemudian dikalikan dengan nilai *error* sehingga mendapatkan perubahan bobot seperti berikut ini:
$(J^{T} J + \eta I)^{-1} J e =$
$$\begin{bmatrix} 4.4681155008952106E-4 \\ 2.5067979555077057E-4 \\ 2.51259441295386E-4 \\ \vdots \\ 0.0445088196851688 \end{bmatrix}_{29x1}$$
14. Untuk mendapatkan bobot baru, bobot lama dikurangi dengan perubahan bobot yang diperoleh dari tahap ke 13.
$$w_{baru} = \begin{bmatrix} 0.7567085254499105 \\ 0.6105444422044491 \\ 0.6033268065587046 \\ \vdots \\ 0.3117703493148312 \end{bmatrix}_{29x1}$$
15. Setelah memperoleh bobot baru maka bobot baru tersebut akan menjadi bobot lama untuk perhitungan selanjutnya. Lalu lakukan hitung maju dengan menggunakan data pertama, gunakan Persamaan 14 hingga 17.
$z\_in_1 = 2.0546726673662357$
$z\_in_2 = 2.2489252396027126$
$z\_in_3 = 2.8803553834172444$
$z\_in_4 = 1.881884120900208$
$z_1 = 0.8864189149615563$
$z_2 = 0.9045577881513118$
$z_3 = 0.9468667458209455$
$z_4 = 0.8678273906115865$
$y\_in = 1.544933774444029$
$y = 0.8241808065833971$
16. Setelah mendapat keluaran dari hitung maju, lakukan perhitungan nilai *error* dan SSE baru.
Error = -0.16328467658339707
SSE = 0.02666188560694458
17. Lakukan pengecekan apakah SSE baru ≤ SSE lama. Jika ya, lakukan kembali perhitungan tahap 5 hingga tahap 16 dengan parameter Marquardt dibagi dengan faktor Tau.. Jika tidak, lakukan kembali perhitungan tahap 13 hingga tahap 16 dengan parameter Marquardt dikali dengan faktor Tau.
18. Perhitungan dilanjutkan dengan tahap 5 hingga tahap 16 karena SSE baru ≤ SSE lama. Perhitungan dilanjutkan dengan nilai parameter Marquardt yang telah berubah. Setelah itu diperoleh bobot baru sebagai berikut:
$w_{baru} =$
$$\begin{bmatrix} 0.7539685087673208 \\ 0.6091348988897222 \\ 0.6017764638691792 \\ \vdots \\ 5.679858239571178E-6 \end{bmatrix}_{29x1}$$

19. Setelah itu dilakukan hitung maju lagi menggunakan bobot baru yang diperoleh dan menggunakan data kedua lalu menghasilkan nilai SSE baru.
20. Hasil perhitungan memperoleh SSE baru > SSE lama sehingga dilanjutkan ke tahap 13 hingga tahap 16. Parameter Marquardt yang digunakan adalah parameter Marquardt yang dikalikan dengan faktor Tau. Tahap ini menghasilkan nilai SSE baru.
21. Perhitungan di atas menghasilkan SSE baru > SSE lama sehingga harus mengulang tahap 13 hingga tahap 16 dengan parameter Marquardt yang semakin besar karena dikalikan dengan faktor Tau. Percobaan yang dilakukan penulis, pengulangan tahap 13 hingga tahap 16 terjadi sebanyak 19 kali hingga mendapatkan SSE baru ≤ SSE lama. Kemudian proses *training* dilanjutkan menggunakan data ketiga. Hasil perhitungan maju menggunakan data ketiga menghasilkan nilai SSE.
22. Setelah data ketiga selesai maka kembali lagi menggunakan data pertama dan *epoch* telah menjadi 2. Syarat berhentinya proses *training* telah terpenuhi yaitu *epoch* ≥ maksimal *epoch* sehingga bobot akhir diperoleh pada tahap ini.

Setelah memperoleh bobot optimal dan arsitektur jaringan yang sesuai dari masing-masing algoritma pelatihan, maka selanjutnya dilakukan proses *testing*. Penulis akan melakukan perhitungan manual proses *testing*. Secara umum, proses *testing* kedua algoritma memiliki tahap-tahap yang serupa. Perbedaan hanya terletak pada bobot awal dan arsitektur jaringan yang akan digunakan pada tahap *testing*. Masing-masing algoritma pelatihan baik algoritma Propagasi Balik maupun algoritma Levenberg-Marquardt menghasilkan bobot optimal dan arsitektur jaringan yang berbeda pada proses *training*. Pada proses *testing* digunakan bobot awal yang diperoleh dari bobot optimal masing-masing algoritma. Penulis memilih menggunakan bobot optimal yang dihasilkan algoritma Propagasi Balik untuk dilakukan perhitungan manual proses *testing*. Pada proses *testing* ini digunakan data yang tidak digunakan pada proses *training*. Setelah memilih arsitektur jaringan yang ingin digunakan maka proses *testing* dapat dimulai. Tahap-tahap proses *testing* sebagai berikut:

1. Penentuan bobot-bobot yang akan digunakan pada proses *testing*.
2. Penentuan data masukan proses *testing*.
3. Data masukan tersebut kemudian dinormalisasi menggunakan metode normalisasi *Min-Max*. Setelah itu hitung maju menggunakan Persamaan 2 hingga 5. Hasil dari perhitungan maju adalah sebagai berikut:
   $z\_in_1 = 2.526001858$
   $z\_in_2 = 2.059225005$
   $z\_in_3 = 2.256779836$
   $z\_in_4 = 1.974248509$
   $z\_in_5 = 2.226048321$
   $z\_in_6 = 1.692698679$
   $z_1 = 0.925944663$
   $z_2 = 0.886876441$
   $z_3 = 0.905233748$
   $z_4 = 0.878066713$
   $z_5 = 0.902564393$
   $z_6 = 0.844578733$
   $y\_in = 1.954034843$
   $y = 0.875885934$
4. Setelah mendapat nilai keluaran berupa curah hujan, maka akan dihitung selisihnya dengan target keluaran.
   Error = 0.740325866 – 0.875885934 = -0.135560068
   $SSE = (-0.135560068)^2 = 0.018376532$
5. Setelah memperoleh nilai *error* dan SSE maka nilai keluaran curah hujan yang diperoleh lalu didenormalisasi sehingga menghasilkan nilai berikut:
   Nilai keluaran curah hujan: 860.1199876
   Berdasarkan kategori curah hujan maka cuaca yang dihasilkan adalah hujan sangat lebat.

4.6 Tahap Pengujian

Tahap ini terdiri dari analisa algoritma dan pengujian terhadap kedua algoritma pelatihan. Tujuan dari tahap evaluasi ini adalah untuk mengetahui perbedaan yang terdapat pada kedua jenis algoritma pelatihan yaitu algoritma Propagasi Balik dan algoritma Levenberg-Marquardt. Perbedaan kedua jenis algoritma pelatihan dapat dilihat dari segi efesiensi algoritma maupun dari segi keluaran yang dihasilkan. Kinerja kedua jenis algoritma dievaluasi menggunakan parameter SSE dan waktu eksekusi program. Beberapa parameter yang akan diuji

pengaruhnya terhadap nilai SSE dan waktu eksekusi program adalah jumlah *neuron* pada lapisan tersembunyi, nilai *learning rate*, dan nilai parameter Marquardt.

4.6.1 Analisa Algoritma

Analisa algoritma dapat dilihat dari segi efisiensi algoritma. Penulis menggunakan order waktu proses (Big-Oh) untuk menguji efisiensi algoritma Propagasi Balik dan algoritma Levenberg-Marquardt. Analisa Big-Oh menguji kompleksitas algoritma dan menekankan pada bagian operasi aktif yaitu operasi yang paling sering dieksekusi. Operasi pada algoritma Propagasi Balik dan algoritma Levenberg-Marquardt ada yang serupa namun ada juga yang berbeda. Penulis akan melakukan analisa Big-Oh pada bagian operasi yang berbeda diantara algoritma Propagasi Balik dan algoritma Levenberg-Marquardt. Algoritma Propagasi Balik memiliki order $O(n^2)$ dan algoritma Levenberg-Marquardt memiliki order $O(n^4)$. Algoritma Levenberg-Marquardt memiliki kompleksitas yang lebih besar daripada algoritma Propagasi Balik.

4.6.2 Pengujian Algoritma Propagasi Balik dan Algoritma Levenberg-Marquardt

Tahap pengujian algoritma Propagasi Balik dan algoritma Levenberg-Marquardt akan dilakukan dengan membandingkan nilai SSE yang diperoleh sesuai dengan arsitektur jaringan dan parameter algoritma masing-masing. Selain itu, akan dilakukan pengujian terhadap data *testing* menggunakan bobot-bobot optimal yang dihasilkan kedua algoritma pelatihan. Tingkat akurasi yang dihasilkan dari proses *testing* juga akan menjadi parameter pengujian bagi kedua algoritma pelatihan tersebut.

Untuk proses *training* algoritma Propagasi Balik, penulis melakukan beberapa percobaan dengan melakukan perubahan pada jumlah *neuron* lapisan tersembunyi mulai dari 3 *neuron* hingga 15 *neuron*. Tidak terdapat ketentuan mengenai jumlah *neuron* yang harus digunakan pada lapisan tersembunyi, maka dari itu penulis mencoba untuk melakukan pengujian terhadap perubahan jumlah *neuron* pada lapisan tersembunyi. Selain itu, pengujian juga dilakukan terhadap perubahan nilai target *error* dan *learning rate*. Target *error* yang akan digunakan sebesar 0.1, 0.01, 0.001. Nilai *learning rate* yang akan diuji coba adalah 0.01, 0.05, 0.1, 0.5 dan 1. Pengujian perubahan nilai *learning rate* akan menggunakan arsitektur jaringan yang menghasilkan nilai SSE terkecil dari setiap target *error*.

Tabel 2 Hasil proses *training* algoritma Propagasi Balik

Jumlah *neuron* lapisan tersembunyi	*Learning rate*	Target *error*	SSE (*mean*)	*Epoch* (*mean*)	Waktu (*mean*)
3	0.1	0.1	0.03686530	1	6.0333
15	0.1	0.01	0.0021932	1	124.2
6	0.1	0.001	1.83E-04	1	25.966

Berdasarkan Tabel 2 diperoleh nilai SSE minimum dari ketiga jenis target *error*. Setiap nilai minimum SSE diperoleh jumlah *neuron* pada lapisan tersembunyi yang berbeda-beda sehingga menghasilkan arsitektur jaringan yang berbeda pula. Setiap percobaan yang dihasilkan telah melalui 30 kali uji coba dan nilai SSE, *epoch*, waktu yang diambil adalah rata-rata dari 30 kali uji coba tersebut. Setelah memperoleh arsitektur jaringan untuk setiap target *error* maka akan diuji pengaruh nilai *learning rate* terhadap nilai SSE dan waktu yang dihasilkan. Nilai *learning rate* yang akan diuji adalah 0.01, 0.05, 0.1, 0.5, dan 1. Nilai *learning rate* yang berbeda-beda tersebut akan diuji pada setiap data dengan SSE minimum untuk ketiga jenis target *error*.

Untuk proses *training* algoritma Levenberg-Marquardt, penulis melakukan beberapa percobaan dengan merubah jumlah *neuron* lapisan tersembunyi mulai dari 3 *neuron* hingga 15 *neuron*. Hal ini dilakukan untuk mencari nilai SSE terkecil. Selain itu, pengujian juga dilakukan dengan mengubah nilai target *error* dan parameter Marquardt. Target *error* yang akan digunakan sebesar 0.1, 0.01, 0.001. Nilai parameter Marquardt yang akan diuji coba adalah 0.01, 0.05, 0.1, 0.5 dan 1. Pengujian perubahan nilai parameter Marquardt akan menggunakan arsitektur jaringan yang menghasilkan nilai SSE terkecil dari setiap target *error*.

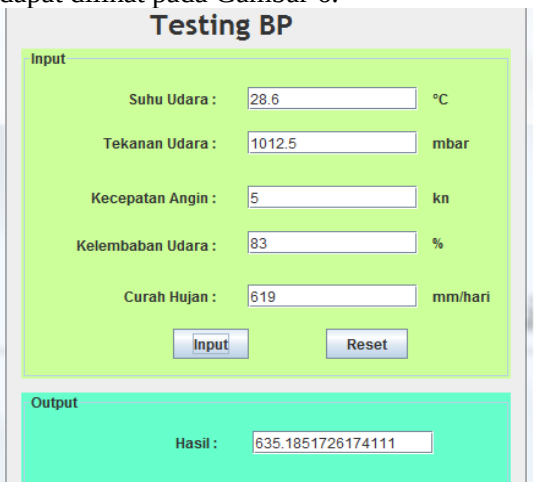
Pada Tabel 3 diperoleh nilai SSE minimum dari ketiga jenis target *error*. Setiap nilai minimum SSE diperoleh jumlah *neuron* pada lapisan tersembunyi yang berbeda-beda sehingga menghasilkan arsitektur jaringan

yang berbeda pula. Setiap percobaan yang dihasilkan telah melalui 30 kali uji coba dan nilai SSE, *epoch*, waktu yang diambil adalah rata-rata dari 30 kali uji coba tersebut. Setelah memperoleh arsitektur jaringan untuk setiap target *error* maka akan diuji pengaruh nilai parameter Marquardt terhadap nilai SSE dan waktu yang dihasilkan. Nilai parameter Marquardt yang akan diuji adalah 0.01, 0.05, 0.1, 0.5, dan 1. Nilai parameter Marquardt yang berbeda-beda tersebut akan diuji pada setiap data dengan SSE minimum untuk ketiga jenis target *error*.

Tabel 3 Hasil proses *training* algoritma Levenberg-Marquardt

Jumlah *neuron* lapisan tersembunyi	Parameter Marquardt	Faktor Tau	Target *Error*	SSE (*mean*)	*Epoch* (*mean*)	Waktu (*mean*)
3	0.1	10	0.1	0.04410357	1	8.43333
15	0.1	10	0.01	0.00190576	1	901.333
8	0.1	10	0.001	1.62E-04	1	616.867

Adapun tampilan program untuk proses *testing* algoritma Propagasi Balik dapat dilihat pada Gambar 6.



Gambar 6 Tampilan *testing* algoritma BP

Berdasarkan penelitian pada proses *training* dengan menggunakan 88 data dan proses *testing* dengan menggunakan 38 data, penulis memperoleh beberapa kesimpulan yaitu:

1. Jumlah *neuron* pada lapisan tersembunyi mempengaruhi nilai SSE dan waktu eksekusi program. Semakin banyak jumlah *neuron* maka waktu eksekusi program semakin lama dan nilai SSE semakin kecil untuk beberapa kasus.
2. Target *error* mempengaruhi jumlah *neuron* pada lapisan tersembunyi. Nilai target *error* yang besar dapat dicapai menggunakan arsitektur jaringan dengan jumlah *neuron* yang sedikit pada lapisan tersembunyi. Nilai target *error* yang lebih kecil menggunakan arsitektur jaringan dengan jumlah *neuron* yang lebih banyak pada lapisan tersembunyi.
3. Nilai *learning rate* dan parameter Marquardt dipengaruhi oleh jumlah *neuron* pada lapisan tersembunyi dan target *error*. Jika target *error* besar dan jumlah *neuron* sedikit maka nilai *learning rate* dan parameter Marquardt besar. Jika target *error* kecil dan jumlah *neuron* banyak maka nilai *learning rate* dan parameter Marquardt kecil.

5. KESIMPULAN

Kesimpulan yang diperoleh penulis dari analisa perbandingan algoritma pelatihan Propagasi Balik dan algoritma pelatihan Levenberg-Marquardt pada jaringan syaraf tiruan adalah:

1. Jumlah *neuron* pada lapisan tersembunyi mempengaruhi nilai SSE dan waktu eksekusi program. Semakin banyak jumlah *neuron* maka waktu eksekusi program semakin lama dan nilai SSE semakin kecil untuk beberapa kasus. Pengujian tersebut menggunakan konfigurasi jaringan dengan 5 *neuron* pada lapisan input dan 1-unit bias, 15 *neuron* pada lapisan tersembunyi dan 1 unit bias, serta 1 *neuron* pada lapisan keluaran.
2. Pada proses *training*, algoritma Levenberg-Marquardt dengan menggunakan parameter Marquardt 0.1 dan faktor Tau 10 menghasilkan nilai SSE yang lebih kecil yaitu 0.001905758 dibandingkan algoritma Propagasi Balik yang menggunakan *learning rate* 0.1 dengan nilai SSE 0.0021932 untuk target *error* 0.01. Namun waktu eksekusi program yang dihasilkan algoritma Propagasi Balik yaitu 124.2 ms lebih cepat daripada algoritma Levenberg-Marquardt yaitu 901.333 ms dalam memperoleh nilai SSE tersebut.

6. DAFTAR PUSTAKA

[1] Agustin, M. 2012. *Penggunaan Jaringan Syaraf Tiruan Backpropagation untuk Seleksi Penerimaan Mahasiswa Baru pada Jurusan Teknik Komputer di*

Politeknik Negeri Sriwijaya. Tesis. Semarang. Universitas Diponegoro.

[2] Atiliani, A. 2013. Pelatihan Jaringan Syaraf Tiruan *Multi Layer Perceptron* Menggunakan *Genetic Algorithm Levenberg Marquardt*. Skripsi. Surakarta. Universitas Sebelas Maret.

[3] Nugraha, P. A., Saptono, R. dan Sulistyo, M. E. 2013. Perbandingan Metode Probabilistik *Naive Bayesian Classifier* dan Jaringan Syaraf Tiruan Learning Vector Quantization dalam Kasus Klasifikasi Penyakit Kandungan. *Jurnal ITSMART*. 2/2: 20-33.

[4] Purnamasari, R. W. 2013. Implementasi Jaringan Syaraf Tiruan *Backpropagation* sebagai Sistem Deteksi Penyakit *Tuberculosis* (TBC). Skripsi. Semarang. Universitas Negeri Semarang.

[5] Retnani, W. I. 2013. Perbandingan Algoritma *Backpropagation Levenberg Marquardt* (LM) dengan *Backpropagation Gradient Descent Adaptive Gain* (BPGD/AG) dalam Prediksi Jumlah Pengangguran di Provinsi Jawa Tengah. Skripsi. Surakarta. Universitas Sebelas Maret.

[6] Sharma, B. and Venugopalan, K. 2014. Comparison of Neural Network Training Functions for Hematoma Classification in Brain CT Images. IOSR *Journal of Computer Engineering* (IOSR-JCE). 16/2: 31-35

[7] Utami, A. T. W. dan Ulama, B. S. S. 2015. Penerapan *Backpropagation* untuk Meningkatkan Efektivitas Waktu dan Akurasi pada Data *Wall-Following Robot Navigation*. *Jurnal Sains dan Seni ITS*. 4/2: 280.

[8] Yuniar, R. J., Rahadi, D. dan Setyawati, O. 2013. Perbaikan Metode Prakiraan Cuaca Bandara Abdulrahman Saleh dengan Algoritma *Neural Network Backpropagation*. *Jurnal EECCIS*. 7/1: 65-70.