Algoritma Pelatihan Levenberg-Marquardt Backpropagation Artificial Neural Network Untuk Data Time Series

Sylvia Jane Annatje Sunarauw[1]
Universitas Negeri Manado
e-mail: sylviasumarauw@unima.ac.id

ABSTRAK

Algoritma *Levenberg-Marquardt* merupakan salah satu algoritma *backpropagation Artificial Neural Network* yang dikembangkan sendiri oleh *Kenneth Levenberg* dan *Donald Marquardt*, memberikan solusi numerik untuk masalah meminimalkan fungsi *non-linear*. Algoritma ini memadukan metode *steepest descent* dan algoritma *Gauss-Newton*. yaitu kecepatan algoritma *Gauss-Newton* dan stabilitas metode *steepest descent.* Ide dasar dari algoritma *Levenberg-Marquardt* adalah melakukan proses pelatihan gabungan. Pada sekitar area dengan kelengkungan yang kompleks, algoritma *Levenberg-Marquardt* beralih ke algoritma *steepest descent*, sampai kelengkungannya tepat untuk membuat pendekatan kuadrat dan pendekatannya menggunakan algoritma *Gauss-Newton*, yang dapat mempercepat konvergensi secara signifikan. Dalam menerapkan algoritma *Levenberg-Marquardt* untuk pelatihan *neural network*, harus menyelesaikan dua masalah yaitu menghitung matriks *Jacobian*, dan bagaimana mengatur proses pelatihan iteratif untuk memperbarui bobot. Algoritma *Levenberg-Marquardt* memecahkan permasalahan yang ada di kedua metode *gradient descent* dan metode *Gauss-Newton* untuk pelatihan *neural-netwok*, dengan kombinasi dari dua algoritma maka algoritma ini dianggap sebagai salah satu algoritma pelatihan yang paling efisien.

Kata Kunci: *Levenberg-Marquardt Backpropagation, Artificial Neural Network*

ABSTRACT

The Levenberg-Marquardt algorithm is one of the backpropagation algorithms of Artificial Neural Network, which was developed by Kenneth Levenberg and Donald Marquardt, providing numerical solutions to the problem of minimizing non-linear functions. This algorithm combines the steepest descent method and the Gauss-Newton algorithm. namely the speed of the Gauss-Newton algorithm and the stability of the steepest descent method. The basic idea of the Levenberg-Marquardt algorithm is to carry out a joint training process. Around the area with complex curvature, the Levenberg-Marquardt algorithm switches to the steepest descent algorithm, until its curvature is appropriate to make a quadratic approach and the approach uses the Gauss-Newton algorithm, which can accelerate convergence significantly. In applying the Levenberg-Marquardt algorithm for neural network training, it must solve two problems, namely calculating the Jacobian matrix, and how to regulate the iterative training process to update weights. The Levenberg-Marquardt algorithm solves problems in both gradient descent and Gauss-Newton methods for neural-netwok training, with a combination of two algorithms, this algorithm is considered to be one of the most efficient training algorithms.

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

PENDAHULUAN

Model *artificial neural network* adalah model dengan struktur fungsi yang fleksibel. Hal ini mengakibatkan model *artificial neural network* cepat berkembang dan telah banyak diaplikasikan pada berbagai bidang (Abdalla dkk., 2011) dan (Sumarauw dkk., 2016). *Time series* atau runtun waktu adalah suatu himpunan pengamatan dengan urutan titik data yang dibangun secara berurutan dalam suatu interval waktu tertentu (Rosadi, 2011) dan (Box & Jenkins, 1971). Pada peramalan data

runtun waktu digunakan metode *backpropagation* yang merupakan metode pelatihan terawasi yang diterapkan pada *Artificial Neural Network.* Algoritma *Levenberg-Marquardt* adalah salah satu algoritma *backpropagation* yang dikembangkan sendiri oleh *Kenneth Levenberg* dan *Donald Marquardt*, memberikan solusi numerik untuk masalah meminimalkan fungsi *non-linear*. Algoritma ini cepat dan memiliki konvergensi yang stabil terutama pada data runtun waktu.

KAJIAN TEORI

Tiga hal yang sangat menentukan keandalan kinerja *neural network* adalah:

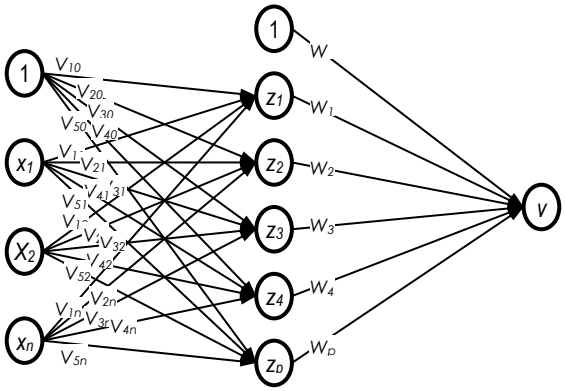
(1) Pola rangkaian neuron-neuron dalam jaringan yang disebut dengan arsitektur jaringan
(2) Metode penentuan bobot (*weight*) pada hubungan, disebut pelatihan (*training*), pembelajaran (*learning*), atau algoritma
(3) Persamaan fungsi untuk mengolah masukan yang akan diterima oleh neuron yang disebut dengan fungsi aktivasi (Fausett, 1994)

Beberapa hasil penelitian menyimpulkan bahwa kekuatan utama dalam metode *Artificial Neural Network* terletak pada kemampuannya untuk mengetahui hubungan linier yang melekat pada data, sedangkan model linier menggambarkan hubungan linier antara pengamatan saat ini dan waktu yang akan datang. *Artificial Neural Network* menggambarkan hubungan *non linear* antara keduanya (Samarasinghe, 2006). Hal ini dapat digambarkan sebagai;

$$y_t = f\left(y_{t-1}, y_{t-2}, \ldots y_{t-p}\right) + \varepsilon_t$$

Dimana, $f\left(y_{t-1}, y_{t-2}, \ldots y_{t-p}\right)$ adalah fungsi *non linear* yang memetakan serangkaian pengamatan *non linear* masa lalu dengan hasil berikutnya. Fungsi ini adalah model *neural network*. Komponen terakhir persamaan yaitu ε_t adalah kesalahan yang sebagai menjadi variabel acak dengan rata-rata 0 dan varians σ^2.

Pada peramalan data runtun waktu digunakan metode propagasi balik (*backpropagation*) yang merupakan metode pelatihan terawasi yang diterapkan pada *Artificial Neural Network*. Metode ini memiliki satu atau lebih lapisan tersembunyi (*hidden layer*). Gambar-1 menunjukkan contoh arsitektur *Artificial Neural Network* 3-*layer*. Lapisan masukan mempunyai *n* buah *node*, yaitu: X_1, X_2, X_3, …X_n. Lapisan tersembunyi mempunyai *p* buah *node* yaitu Z_1, Z_2, Z_3,…, Z_p. Lapisan luaran (*output layer*) memiliki 1 buah *node* yaitu *Y*. Bobot dari lapisan masukan ke lapisan tersembunyi dinyatakan dengan V_{pn} dengan *p* adalah *node* ke-*p* pada lapisan tersembunyi dan *n* adalah *node* ke *n* pada lapisan masukan, sedangkan V_{po} adalah bias yang masuk *node* ke-*p* pada lapisan tersembunyi. Bobot dari lapisan tersembunyi ke lapisan luaran dinyatakan dengan W_{mp} dengan *m* adalah node ke-*m* pada lapisan luaran dan *p* adalah *node* ke-*p* pada lapisan tersembunyi, sedangkan V_{mo} adalah bias yang masuk node ke-*m* pada lapisan luaran. Pada Arsitektur *Artificial Neural Network multilayer*, salah satu permasalahan yang dijumpai adalah menentukan jumlah neuron pada lapisan tersembunyi.



Gambar 1. Contoh gambar arsitektur ANN

Fungsi Aktivasi

Dalam metode *backpropagation*, fungsi aktivasi yang dipakai harus memenuhi beberapa syarat yaitu: kontinu, terdeferensiasi dengan mudah, dan merupakan fungsi yang tidak turun. Salah satu fungsi yang memenuhi ketiga syarat tersebut sehingga sering dipakai adalah fungsi *sigmoid biner* yang memiliki interval (0, 1) dan digunakan pada lapisan luaran, sesuai persamaan 1

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

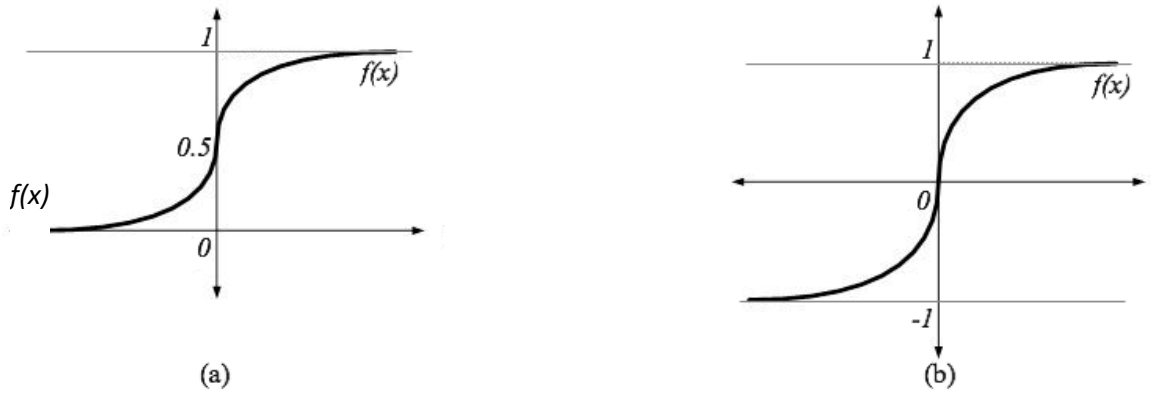
dengan $f'(x) = f(x)\big(1 - f(x)\big)$

Grafik fungsi persamaan 1 terlihat pada Gambar 2(a). Fungsi lain yang sering dipakai adalah fungsi *sigmoid bipolar* yang bentuk fungsinya mirip dengan fungsi *sigmoid biner*, tapi dengan interval yang antara -1 (minus satu) sampai dengan 1 (satu) (persamaan 2).

$$f(x) = \frac{2}{1+e^{x}} - 1 \quad (2)$$

dengan $f'(x) = \frac{(1+f(x))-(1-f(x))}{2}$

Grafik fungsi persamaan (2) terlihat pada Gambar 2(b). Fungsi *sigmoid* memiliki nilai maksimum = 1 (satu). Pada pola yang targetnya lebih besar dari 1 (satu), pola masukan dan luaran harus terlebih dahulu ditransformasi sehingga semua polanya memiliki interval sama seperti fungsi *sigmoid* yang dipakai. Alternatif lain adalah menggunakan fungsi aktivasi *sigmoid* hanya pada lapisan yang bukan luaran *layer*. Pada lapisan luaran, fungsi aktivasi yang dipakai adalah fungsi identitas, *f(x) = x*.



Gambar 2. (a) Grafik fungsi sigmoid biner (b) Bipolar (Fausett, 2004)

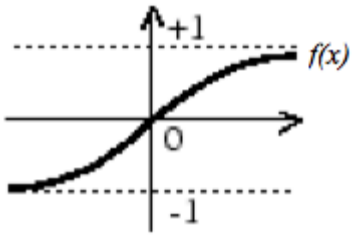
Fungsi lain yang mirip dengan *sigmoid bipolar* adalah fungsi *hyperbolic tangent* yang dirumuskan sebagai berikut

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (3)$$

atau, $$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \quad (4)$$

dengan $f'(x) = [1 + f(x)][1 - f(x)]$

Grafik fungsi persamaan (3) dan (4) adalah seperti pada Gambar Error! No text of specified style in document..



Gambar **Error! No text of specified style in document.**. Grafik fungsi tansig

Algoritma Pelatihan Levenberg-Marquardt Backpropagation

Algoritma *Levenberg-Marquardt*, yang dikembangkan sendiri oleh *Kenneth Levenberg* dan *Donald Marquardt*, memberikan solusi numerik untuk masalah meminimalkan fungsi *non-linear*. Algoritma ini cepat dan memiliki konvergensi yang stabil. Beberapa metode lain telah dikembangkan untuk pelatihan *neural-network*., seperti algoritma *steepest descent*, juga dikenal sebagai algoritma *error backpropagation* (EBP) dan algoritma *Gauss-Newton*, dengan kekurangan masing-masing (Yu dan Wilamowski 2016). Algoritma *Levenberg-Marquardt* memadukan metode *steepest descent* dan algoritma *Gauss-Newton*. yaitu kecepatan algoritma *Gauss-Newton* dan stabilitas metode *steepest descent* (Yu dan Wilamowski 2016) dan (Othman dan Musirin 2012).

Ide dasar dari algoritma *Levenberg-Marquardt* adalah melakukan proses pelatihan gabungan. Pada sekitar area dengan kelengkungan yang kompleks, algoritma *Levenberg-Marquardt* beralih ke algoritma *steepest descent*, sampai kelengkungannya tepat untuk membuat pendekatan kuadrat dan pendekatannya menggunakan algoritma *Gauss-Newton*, yang dapat mempercepat konvergensi secara signifikan. Algoritma *Levenberg-Marquardt* memecahkan permasalahan yang ada di kedua metode *gradient descent* dan metode *Gauss-Newton* untuk pelatihan *neural-netwok*, dengan kombinasi dari dua algoritma. Hal ini dianggap sebagai salah satu algoritma pelatihan yang paling efisien. Beberapa indeks yang digunakan pada Algoritma *Levenberg-Marquardt* adalah:

- *p* adalah indeks pola, dari 1 sampai *P*, dimana *P* adalah jumlah pola.
- *m* adalah indeks luaran, dari 1 sampai *M*, dimana *M* adalah jumlah luaran.
- *i* dan *j* adalah indeks bobot, (1 sampai *N*), dimana N adalah jumlah bobot.
- *k* adalah indeks iterasi.

Jumlah kuadrat galat (*sum squared error* (SSE)) didefinisikan untuk mengevaluasi proses pelatihan. Untuk semua pola pelatihan dan luaran jaringan, dihitung dengan persamaan (5)

$$E(x,w) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{p,m}^2 \qquad (5)$$

dimana, $\boldsymbol{x}$ adalah vektor masukan, dan $\boldsymbol{w}$ adalah vektor bobot
$\boldsymbol{e_{p,m}}$ adalah kesalahan pelatihan pada luaran $\boldsymbol{m}$ ketika menerapkan pola $\boldsymbol{p}$ dan didefinisikan menurut persamaan (6)

$$e_{p,m} = d_{p,m} - o_{p,m} \qquad (6)$$

dimana,
$\boldsymbol{d}$ adalah vektor luaran yang diinginkan
$\boldsymbol{o}$ adalah vektor luaran aktual

Algoritma EBP adalah algoritma orde pertama, dan algoritma *Gauss Newton* adalah algoritma orde kedua yang merupakan pengembangan dari algoritma *Newton*, dimana pada algoritma Newton digunakan matriks *Hessian* (H) (persamaan 7) sebagai derivatif orde kedua total fungsi galat yang merupakan proses perhitungan yang sangat rumit. Oleh karena itu, untuk menyederhanakan proses perhitungan pada algoritma *Gauss Newton* diperkenalkan matriks *Jacobian* (*J*) (persamaan 8)

$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_N^2} \end{bmatrix} \quad (7)$$

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_N} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \cdots & \frac{\partial e_{1,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\partial w_1} & \frac{\partial e_{1,M}}{\partial w_2} & \cdots & \frac{\partial e_{1,M}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,1}}{\partial w_1} & \frac{\partial e_{P,1}}{\partial w_2} & \cdots & \frac{\partial e_{P,1}}{\partial w_N} \\ \frac{\partial e_{P,2}}{\partial w_1} & \frac{\partial e_{P,2}}{\partial w_2} & \cdots & \frac{\partial e_{P,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,M}}{\partial w_1} & \frac{\partial e_{P,M}}{\partial w_2} & \cdots & \frac{\partial e_{P,M}}{\partial w_N} \end{bmatrix} \quad (8)$$

Aturan *update* algoritma *steepest descent* didefinisikan menurut persamaan (9)

$$w_{k+1} = w_k - \alpha g_i \quad (9)$$

dimana,

α adalah konstanta pembelajaran (*learning rate*)

g adalah gradien dan didefinisikan sebagai derivatif orde pertama total fungsi errror (persamaan (10))

$$g = \frac{\partial E(x,w)}{\partial w} = \left[\frac{\partial E}{\partial w_1} \quad \frac{\partial E}{\partial w_2} \quad \cdots \quad \frac{\partial E}{\partial w_N}\right] \quad (10)$$

Sedangkan aturan *update* algoritma *Gauss Newton* didefinisikan menurut persamaan 11

$$w_{k+1} = w_k - (J_k^T J_k)^{-1} J_k e_k \quad (11)$$

Algoritma *Levenberg-Marquardt* memperkenalkan pendekatan lain untuk matriks Hessian seperti pada persamaan 12

$$H = J^T J + \mu I \quad (12)$$

dimana,

μ selalu positif, yang disebut koefisien kombinasi

I adalah matriks identitas

Dari persamaan 12, satu hal yang perlu diperhatikan bahwa unsur-unsur pada diagonal utama matriks *Hessian* yang diperkirakan akan lebih besar dari nol. Oleh karena itu, dengan pendekatan ini (persamaan 12), itu dapat diyakini bahwa matriks *H* selalu dibalik.

Aturan update algoritma *Levenberg-Marquardt* adalah seperti pada persamaan 12

$$w_{k+1} = w_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \quad (13)$$

Sebagai kombinasi dari algoritma *steepest descent* dan algoritma *Gauss-Newton*, algoritma *Levenberg-Marquardt* beralih di antara kedua algoritma selama proses pelatihan. Ketika koefisien kombinasi μ sangat kecil (hampir nol), aturan *update* algoritma *Levenberg-Marquardt* (persamaan 13) mirip dengan aturan *update* pada algoritma *Gauss Newton* (persamaan 11)dan algoritma *Gauss Newton* digunakan. Ketika koefisien kombinasi μ sangat besar, aturan update algoritma *Levenberg-Marquardt* (persamaan 13) mirip dengan aturan *update* pada algoritma *steepest descent* (persamaan 9) dan algoritma *steepest descent* digunakan. Jika kombinasi koefisien μ pada persamaan 13 sangat besar, dapat diartikan sebagai koefisien pembelajaran dalam metode *steepest descent*, sehingga terdapat hubungan antara koefisien pembelajaran dengan μ seperti pada persamaan

$$\alpha = \frac{1}{\mu} \quad (14)$$

Dalam menerapkan algoritma *Levenberg-Marquardt* untuk pelatihan *neural network*, dua masalah harus diselesaikan: bagaimana cara menghitung matriks *Jacobian*, dan bagaimana mengatur proses pelatihan iteratif untuk memperbarui bobot.

PEMBAHASAN

Menghitung Matriks Jacobian

Pada sub bab ini, nilai ***j*** dan ***k*** yang digunakan sebagai indeks dari *neuron*, dari 1 sampai ***nn***, di mana ***nn*** adalah jumlah neuron yang terkandung dalam sebuah topologi; ***i*** adalah indeks dari masukan neuron, dari 1 sampai ***ni***, dimana ***ni*** adalah jumlah masukan dan mungkin berbeda untuk neuron yang berbeda.

Sebagai pengenalan konsep dasar pelatihan *neural network*, mari kita mempertimbangkan ***j*** neuron dengan masukan ***ni***, seperti yang ditunjukkan pada Gambar 4. Jika neuron ***j*** adalah di lapisan pertama, semua masukan yang akan dihubungkan ke masukan dari jaringan, jika tidak, masukan dapat dihubungkan ke luaran dari neuron lain atau ke jaringan masukan jika koneksi di seluruh lapisan diperbolehkan.

Node ***y*** merupakan konsep penting dan fleksibel. Ia dapat berupa $\boldsymbol{Y_{J,i}}$, berarti masukan ***i*** dari neuron ***j***. Hal ini juga dapat digunakan sebagai $\boldsymbol{y_j}$ untuk menentukan luaran dari neuron ***j***. Dalam derivasi berikut, jika simpul ***y*** memiliki satu indeks kemudian digunakan sebagai node luaran neuron, tetapi jika memiliki dua indeks (neuron dan masukan), ia adalah simpul masukan neuron.

Luaran node dari neuron ***j*** dihitung menggunakan

$$y_j = f_j(net_j) \quad (15)$$

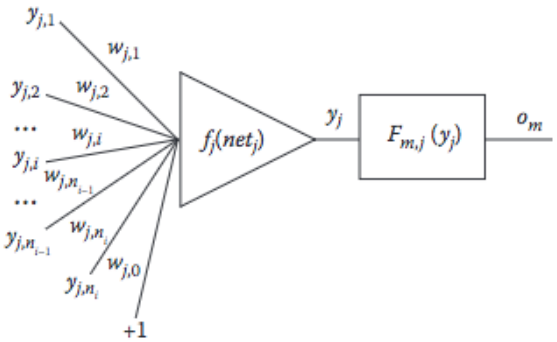
dimana f_j adalah fungsi aktivasi neuron j dan nilai bersih net_j adalah jumlah bobot node masukan neuron *j*:

$$net_j = \sum_{i=1}^{ni} w_{j,i} y_{j,i} + w_{j,0} \quad (16)$$

dimana

$\boldsymbol{y_{j,i}}$ adalah simpul masukan ***i*** dari neuron *j*, dengan bobot $\boldsymbol{w_{j,i}}$

$\boldsymbol{w_{j,0}}$ adalah bobot bias neuron ***j***



Gambar 4. Koneksi dari neuron *j* dengan seluruh jaringan. Node $y_{j,i}$ bisa mewakili masukan jaringan atau luaran dari neuron lainnya. $F_{m,j}(y_j)$ adalah hubungan

nonlinier antara luaran neuron simpul y_j dan luaran jaringan o_m.

Turunan dari $\boldsymbol{net_j}$ pada persamaan (), adalah

$$\frac{\partial net_j}{\partial w_{j,i}} = y_{j,i} \quad (18)$$

dan kemiringan (*slope*) $\boldsymbol{s_j}$ dari fungsi aktivasi $\boldsymbol{f_j}$ adalah

$$s_j = \frac{\partial y_j}{\partial net_j} = \frac{\partial f_j(net_j)}{\partial net_j} \quad (19)$$

Antara node luaran $\boldsymbol{y_j}$ dari neuron $\boldsymbol{j}$ tersembunyi dan luaran jaringan $\boldsymbol{o_m}$, ada hubungan non-linear yang kompleks (Gambar)

$$o_m = F_{m,j}(y_j) \quad (20)$$

dimana $\boldsymbol{o_m}$ adalah luaran ke-$\boldsymbol{m}$ dari jaringan.

Kompleksitas dari fungsi non-linear ini $\boldsymbol{F_{m,j}(y_j)}$ tergantung pada berapa banyak neuron lainnya adalah antara neuron $\boldsymbol{j}$ dan luaran jaringan $\boldsymbol{m}$. Jika neuron $\boldsymbol{j}$ adalah pada luaran jaringan $\boldsymbol{m}$, maka $\boldsymbol{o_m} = \boldsymbol{y_j}$ dan $\boldsymbol{F'_{mj}(y_j)} = 1$, di mana $\boldsymbol{F'_{mj}}$ adalah turunan dari hubungan non-linear antara neuron $\boldsymbol{j}$ dan luaran $\boldsymbol{m}$. Unsur-unsur dari matriks ***Jacobian*** pada persamaan 8 dapat dihitung sebagai

$$\frac{\partial e_{p,m}}{\partial w_{j,i}} = \frac{\partial (d_{p,m} - o_{p,m})}{\partial w_{j,i}} = -\frac{\partial o_{p,m}}{\partial w_{j,i}} = -\frac{\partial o_{p,m}}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{j,i}} \quad (21)$$

Penggabungan persamaan 15 ke persamaan 19, maka persamaan 20 dapat ditulis kembali sebagai

$$\frac{\partial e_{p,m}}{\partial w_{j,i}} = -F'_{mj} s_j y_{j,i} \quad (22)$$

dimana $\boldsymbol{F'_{mj}}$ adalah turunan fungsi non-linear antara neuron $\boldsymbol{j}$ dan luaran $\boldsymbol{m}$.

Perhitungan untuk matriks *Jacobian* dapat diatur sesuai dengan perhitungan *backpropagation* tradisional di urutan pertama algoritma (seperti algoritma EBP). Tetapi ada juga perbedaan diantara mereka. Pertama-tama, untuk setiap pola, dalam algoritma EBP, hanya satu proses *backpropagation* diperlukan, sedangkan dalam algoritma *Levenberg-Marquardt* proses *backpropagation* harus diulang untuk setiap luaran yang terpisah untuk mendapatkan baris berturut-turut dari matriks *Jacobian*. Perbedaan lain adalah bahwa konsep *backpropagation* dari parameter $\boldsymbol{\delta}$ harus diubah. Dalam algoritma EBP, galat luaran adalah bagian dari parameter $\boldsymbol{\delta}$, sedangkan dalam algoritma *Levenberg-Marquardt*, parameter $\boldsymbol{\delta}$ dihitung untuk setiap neuron $\boldsymbol{j}$ dan setiap luaran $\boldsymbol{m}$, secara terpisah

$$\delta_{m,j} = s_j F'_{mj} \quad (23)$$

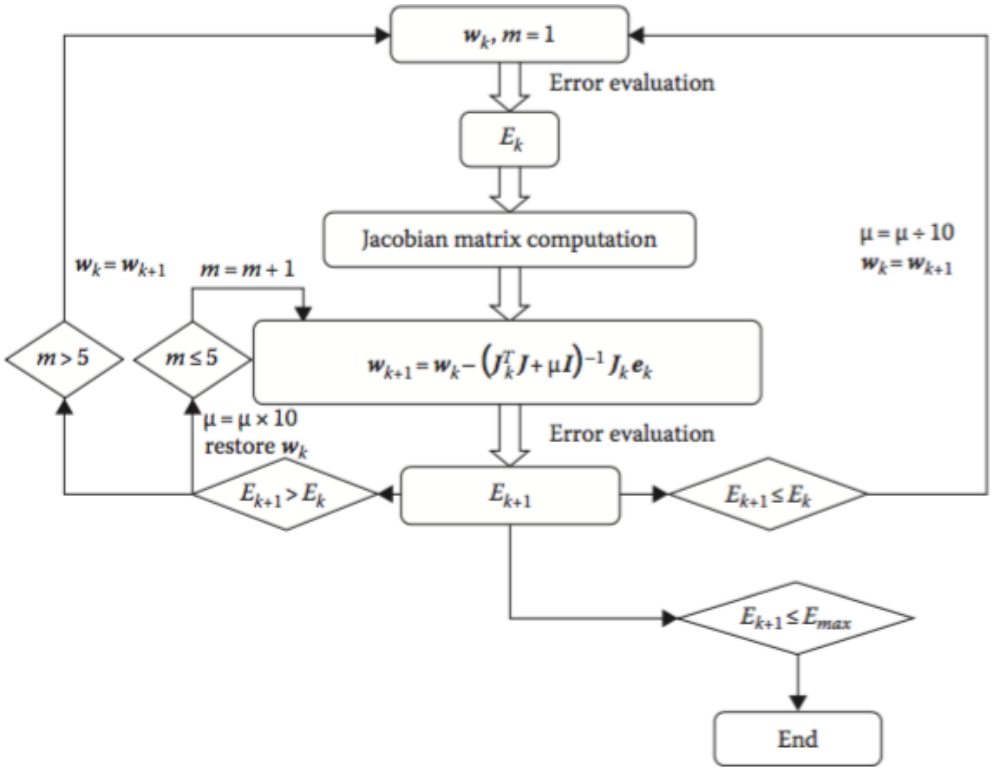
Dengan menggabungkan persamaan (22) dan (23), unsur-unsur dari matriks *Jacobian* dapat dihitung dengan

$$\frac{\partial e_{p,m}}{\partial w_{j,i}} = -\delta_{m,j} y_{j,i} \quad (24)$$

Ada dua faktor tidak diketahui dalam persamaan 11 untuk perhitungan matriks *Jacobian*. Masukan node, $\boldsymbol{y_{j,i}}$ dihitung dalam perhitungan maju (penyebaran sinyal dari masukan ke luaran); sementara $\boldsymbol{\delta_{m,j}}$ diperoleh dalam perhitungan mundur, yang dihimpun sebagai kesalahan *backpropagating* dari neuron luaran (lapisan luaran) ke masukan jaringan. Pada neuron luaran $\boldsymbol{m}$ ($\boldsymbol{j} = \boldsymbol{m}$), $\boldsymbol{\delta_{m,j}} = \boldsymbol{s_m}$.

Desain Proses Pelatihan

Implementasi algoritma *Levenberg-Marquardt* untuk pelatihan *neural network* adalah seperti diagram alir pada Gambar (Yu dan Wilamowski 2016).



Gambar 5 . Diagram untuk pelatihan menggunakan algoritma Levenberg-Marquardt: w_k adalah bobot saat ini, w_{k+1} adalah bobot berikutnya, E_{k+1} adalah total galat saat ini, dan E_k adalah total galat terakhir

Proses pelatihan menggunakan algoritma *Levenberg-Marquardt* dirancang sebagai berikut: (Yu dan Wilamowski 2016)

(i). Dengan bobot awal (secara acak), mengevaluasi total galat (SSE).
(ii). Melakukan *update* persamaan 13 untuk menyesuaikan bobot.
(iii). Dengan bobot baru, mengevaluasi kesalahan total.
(iv). Jika total kesalahan saat ini meningkat sebagai akibat dari *update*, kembali ke-langkah sebelumnya (seperti *reset* vektor bobot ke nilai yang lebih baik) dan meningkatkan koefisien kombinasi μ dengan faktor 10 atau oleh beberapa faktor lainnya. Kemudian lanjutkan ke langkah ii dan mencoba *update* kembali.
(v). Jika total kesalahan saat ini menurun sebagai akibat dari *update*, langkah ini diterima (*accepted* dan menjaga bobot vektor baru) dan menurunkan koefisien kombinasi μ dengan faktor 10 atau oleh faktor yang sama seperti langkah iv.
(vi). Lanjutkan ke langkah ii dengan bobot baru sampai kesalahan total saat ini lebih kecil dari nilai yang dibutuhkan.

PENUTUP

Algoritma *Levenberg-Marquardt* adalah proses pelatihan gabungan. Pada sekitar area dengan kelengkungan yang kompleks, algoritma *Levenberg-Marquardt* beralih ke algoritma *steepest descent*, sampai kelengkungannya tepat untuk membuat pendekatan kuadrat dan pendekatannya menggunakan algoritma *Gauss-Newton*, yang dapat mempercepat konvergensi secara signifikan. Algoritma

Levenberg-Marquardt memecahkan permasalahan yang ada di kedua metode *gradient descent* dan metode *Gauss-Newton* untuk pelatihan *neural-netwok*, dengan kombinasi dari dua algoritma maka algoritma ini dianggap sebagai salah satu algoritma pelatihan yang paling efisien**.**

DAFTAR PUSTAKA

Abdalla, M, A. M., Dress, S. dan Zaki, N., (2011), Detection of Masses in Digital Mammogram Using Second Order Statistics and Artificial Neural Network, *International Journal of Computer Science and Information Technology*, 3(3), pp.176–186.

Box, G. E. P. dan Jenkins, G. M., (1971). Time Series Analysis Forecasting and Control, *D. J. Bartholomew Operational Research*, Quarterly (1970-1977), Vol 22, pp.199–201.

Fausett, L., 1994, *Fundamental of Neural Networks, Algoritms and Aplications*,

Othman, M. M. dan Musirin, I., (2012), Short Term Load Forecasting Using Artificial Neural Network with Feature Extraction Method and Stationary Output, *Power Engineering and Optimization Conference (PEDCO)*, Melaka, Malaysia, 2012 Ieee International, (June), pp.6–7.

Rosadi, D., (2011), *Pengantar Analisis Runtun Waktu*, Buku Ajar Perkuliahan, Program Studi Statistika, FMIPA UGM Yogyakarta.

Samarasinghe, S., (2006). Neural Network for Applied Sciences and Engineering, *Journal of Chemical Information and Modeling*, 53(9), pp.1689–1699.

Sumarauw, S. J. A., Subanar, Winarko, E. dan Wardoyo, R., (2016), Weighting Customer's Data For More Accurate Short-Term Load Forecast, *Journal of Theoretical and Applied Information Technology*, 90(2).

Yu, H. dan Wilamowski, B. M., (2016), *Levenberg–Marquardt Training*, *In Intelligent System*. CRC Press, p. 16.

THIS PAGE IS INTENTIONALLY LEFT BLANK