PREDIKSI CURAH HUJAN KOTA SEMARANG DENGAN FEEDFORWARD NEURAL NETWORK MENGGUNAKAN ALGORITMA QUASI NEWTON BFGS DAN LEVENBERG-MARQUARDT

Budi Warsito[1)], Sri Sumiyati[2)]
[1)]Program Studi Statistika Universitas Diponegoro
[2)]Program Studi Teknik Lingkungan Universitas Diponegoro

Abstract

This paper will study the rainfall prediction at Semarang City as time series data with Feed Forward Neural Network (FFNN). The learning algorithm that be used are the Quasi Newton BFGS and Levenberg-Marquardt algorithm. The input unit is determined based on the best of ARIMA model. The computation is done with use Matlab 7.1 program with 1000 epoch, five unit of hidden layer, 100 replication and use input at lag variabel 1, 12 and 13, respectively. The result shows that the prediction is good in relatively, where Quasi Newton BFGS algorithm result the Mean Square Error (MSE) that more accurate.

***Key Words** : FFNN, Quasi Newton BFGS, Levenberg-Marquardt, rainfall*

1. Pengantar

Curah hujan mempunyai peran yang sangat penting. Berdasarkan data curah hujan dapat dilakukan penggolongan iklim menurut perbandingan antara jumlah rata-rata bulan kering dengan jumlah rata-rata bulan basah. Bulan kering terjadi jika curah hujan bulanan kurang dari 60 mm/bulan, sedangkan bulan basah terjadi jika curah hujan bulanan diatas 100 mm/bulan. Diantara bulan kering dan bulan basah tersebut terdapat bulan lembab yang terjadi apabila curah hujan bulanan antara 60-100 mm/bulan. Kondisi topografi kota Semarang yang terbagi atas Semarang atas dan Semarang bawah menjadikan hujan sebagai hal yang sangat vital terutama curah hujan yang tinggi dapat memberikan efek yang besar terutama pada Semarang bagian bawah. Untuk itu diperlukan prediksi curah hujan dengan presisi tinggi berdasarkan data masa lampau sehingga efek negatifnya dapat dicegah dengan tindakan preventif. Pada tulisan ini prediksi curah hujan sebagai data time series dilakukan dengan Feed Forward Neural Network (FFNN).

Pada tulisan ini proses pelatihan untuk menentukan bobot-bobot koneksi pada model FFNN digunakan algoritma Quasi Newton BFGS dan Levenberg-Marquardt dengan paket program Matlab 7.1. Sistematika penulisan dilakukan sebagai berikut. Pada bagian kedua membahas algoritma propagasi balik pada model FFNN, bagian ketiga tentang algoritma BFGS, bagian keempat tentang algoritma Levenberg Marquardt. Bagian kelima aplikasi pada data curah hujan kota Semarang dan bagian terakhir penutup.

2. Backpropagation

Pada dasarnya, Neural Network (NN) merupakan suatu kumpulan elemen-elemen pemrosesan sederhana yang saling berhubungan, yang disebut neuron (*unit, sel* atau *node*). Setiap neuron dihubungkan dengan neuron lain dengan link komunikasi langsung melalui pola hubungan yang disebut arsitektur jaringan [5]. Tiap-tiap hubungan tersebut mempunyai bobot koneksi (*weight*) yang dilatih untuk mencapai respon yang diinginkan. Sehingga dengan pelatihan terhadap data berdasarkan bobot-bobot koneksi tersebut diharapkan memperoleh output yang diinginkan. Metode yang digunakan untuk menentukan bobot koneksi tersebut dinamakan algoritma pelatihan (*training algorithm*). Tiap-tiap hubungan antar neuron mempunyai bobot koneksi yang dilatih untuk mencapai respon yang diinginkan dengan melakukan suatu proses pelatihan (*training*). Selama proses pelatihan, terjadi perubahan yang cukup berarti pada bobot-bobot yang menghubungkan antar neuron. Apabila ada rangsangan yang sama dengan rangsangan yang telah diterima oleh neuron sebelumnya, maka neuron akan memberikan reaksi dengan cepat. Akan

tetapi apabila ada rangsangan yang berbeda dengan apa yang telah diterima oleh neuron, maka neuron akan segera beradaptasi untuk memberikan reaksi yang sesuai.

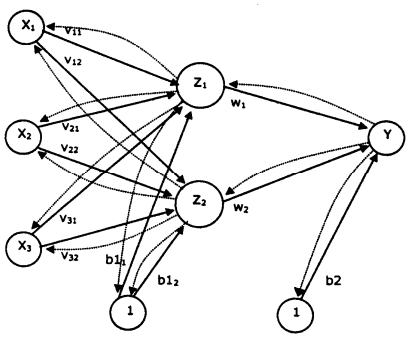
Nilai bobot akan bertambah, jika informasi yang diberikan oleh neuron yang bersangkutan tersampaikan, sebaliknya jika informasi tidak disampaikan oleh suatu neuron ke neuron yang lain, maka nilai bobot yang menghubungkan keduanya akan dikurangi. Pada saat pelatihan dilakukan pada input yang berbeda, maka nilai bobot akan diubah secara dinamis hingga mencapai suatu nilai yang cukup seimbang. Apabila nilai ini telah tercapai mengindikasikan bahwa tiap-tiap input telah berhubungan dengan output yang diharapkan.

Selama proses pelatihan, masing-masing bobot koneksi dipropagasikan ke seluruh unit atau node dengan metode pelatihan perambatan balik (*backpropagation*). Bobot-bobot diatur secara iteratif untuk meminimumkan fungsi kinerja jaringan. Fungsi kinerja yang sering digunakan adalah MSE, dimana fungsi ini akan mengambil kuadrat error yang terjadi antara output dan target. Ada beberapa metode pembelajaran yang dapat digunakan untuk mengatur bobot dalam rangka meminimumkan MSE. Pada tulisan ini dibatasi hanya untuk BFGS *quasi Newton* dan *Levenberg_Marquardt*. Seluruh proses pelatihan tersebut dapat langsung dikerjakan oleh komputer menggunakan aplikasi program MATLAB versi 7.1 dengan syntax `trainbfg` dan `trainlm`.

Pelatihan *backpropagation* dilakukan dalam rangka pengaturan bobot, sehingga pada akhir pelatihan akan diperoleh bobot-bobot yang optimal. Hal ini dapat dipenuhi dengan melakukan proses pelatihan dengan menyesuaikan parameter-parameter atau bobot dan bias koneksi sehingga untuk suatu input tertentu, jaringan dapat menghasilkan output yang sesuai dengan target. Proses penyesuaian dilakukan dengan memberikan sebuah *training set* (berupa suatu himpunan yang terdiri dari pasangan input dan output) pada jaringan. *Training set* tersebut dievaluasi dengan suatu algoritma pelatihan tertentu. Selama proses pelatihan berlangsung, *training set* akan dievaluasi berkali-kali secara iteratif untuk meminimumkan fungsi kuadrat errornya. Fungsi ini akan mengambil kuadrat error yang terjadi antara output dan target. Dengan metode ini kesalahan lokal tiap sel dapat dilihat sebagai bagian yang terkontribusi dalam menghasilkan kesalahan total pada lapisan output. Apabila kesalahan pada lapisan output dapat dipropagasikan kembali masuk ke lapisan tersembunyi, maka kesalahan lokal sel-sel pada lapisan tersebut dapat dihitung untuk mendapatkan perubahan bobot.

Backpropagation merupakan algoritma pelatihan terawasi dan biasanya digunakan oleh perceptron dengan banyak lapisan untuk mengubah bobot-bobot yang terhubung dengan neuron-neuron pada lapisan tersembunyi. Algoritma ini menggunakan error output untuk mengubah nilai bobot-bobotnya dalam arah mundur *(backward)*. Untuk mendapatkan error ini, tahap perambatan maju *(forward propagation)* harus dikerjakan terlebih dahulu.

Contoh arsitektur jaringan backpropagation seperti terlihat pada Gambar 1, dimana jaringan terdiri dari 3 neuron pada lapisan input yaitu x_1, x_2, dan x_3; 1 lapisan tersembunyi dengan 2 neuron, yaitu z_1 dan z_2; serta 1 unit pada lapisan output, yaitu y. v_{ij} adalah bobot yang menghubungkan neuron input ke-i ke neuron ke-j pada lapisan tersemunyi dan $b1_1$ adalah bobot bias yang menuju ke neuron ke-j pada lapisan tersembunyi. Bobot yang menghubungkan z_1 dan z_2 dengan neuron pada lapisan output adalah w_1 dan w_2 dan bobot bias b2 menghubungkan lapisan tersembunyi dengan lapisan output.



Gambar 2

Gambar 1. Arsitektur jaringan *Backpropagation* dengan satu lapisan tersembunyi, 3 unit input, 2 unit tersembunyi dan satu unit output.

Algoritma Pelatihan *Backpropagatian*

Pelatihan *backpropagation* menggunakan metode pancarian titik minimum untuk mencari bobot dengan

error minimum. Algoritma *backpropagation* menggunakan error output untuk mengubah nilai bobot-bobotnya dalam arah mundur (*backward*) [5]. Pelatihan backpropagation meliputi 3 fase yaitu fase maju, fase mundur dan fase perubahan bobot.

Fase I : *Forward Propagation* (Propagasi maju)
Selama propagasi maju, sinyal input (x_i) dipropagasikan ke lapisan tersembunyi. Sinyal-sinyal input terboboti yang masuk ketiap-tiap unit pada lapisan tersembunyi (Z_j, j=1,2,..,p), ditentukan dengan persamaan:

$$z\_in_j = b1_j + \sum_{i=1}^{n} x_i v_{ij} \quad (1)$$

Sinyal output dari lapisan tersembunyi (z_j) ditentukan menggunakan fungsi aktivasi yang telah ditentukan.

$$z_j = f(z\_in_j) \quad (2)$$

Karena fungsi aktivasi yang digunakan *BP* pada lapisan tersembunyi adalah fungsi logistik sigmoid maka persamaan (2) menjadi:

$$z_j = \frac{1}{1+e^{-(z\_in_j)}} = \frac{1}{1+e^{-\left(b1_j + \sum_{i=1}^{n} x_i v_{ij}\right)}}$$

Selanjutnaya z_j tersebut dipropagasikan maju lagi kelapisan output dengan persamaan :

$$y\_in = b2 + \sum_{j=1}^{p} z_j w_j$$

Sinyal outputnya ditentukan dengan :

$y = f(y\_in)$

Karena fungsi aktivasi yang digunakan BP pada lapisan output adalah fungsi identitas, maka persamaan (2.6.5) menjadi :

$$y = f(y\_in) = y\_in = b2 + \sum_{j=1}^{p} z_j w_{j1}$$

Berikutnya, output jaringan (y) dibandingkan dengan target yang harus dicapai (t). Selisih (t– y) adalah kesalahan (error) yang terjadi. Jika kesalahan ini lebih kecil dari batas toleransi yang ditentukan, maka *epoch* dihentikan. Akan tetapi apabila kesalahan masih lebih besar dari batas toleransinya, maka bobot setiap garis dalam jaringan akan dimodifikasi untuk mengurangi kesalahan yang terjadi.

Fase II : *Back Propagation* (Propagasi mundur).
Kesalahan (error) yang tejadi dipropagasikan mundur, dimulai dari garis yang berhubungan langsung dengan unit pada lapisan output.

Fase III : *Weight Update* (Perubahan bobot)
Modifikasi bobot dan bias untuk menurunkan kesalahan yang terjadi.

- Perubahan bobot dan bias pada garis yang menuju unit output:
  w_j (baru) = w_j(lama) + Δw_j
  b2 (baru) = b2(lama) + Δ b2

- Perubahan bobot dan bias pada garis menuju unit tersembunyi:
  v_{ij} (baru) = v_{ij}(lama) + Δv_{ij}
  $b1_j$ (baru) = $b1_j$(lama) + $\Delta b1_j$

Ketiga fase tersebut diulang-ulang terus hingga kondisi penghentian dipenuhi. Umumnya kondisi penghentian yang sering dipakai adalah jumlah *epoch* atau kesalahan. *Epoch* akan dihentikan jika jumlah *epoch* yang dilakukan sudah melebihi jumlah maksimum *epoch* yang ditetapkan, atau jika kesalahan yang terjadi sudah lebih kecil dari batas toleransi yang diijinkan.

Inisialisasi Bobot

Inisialisasi nilai bobot awal sangat mempengaruhi jaringan saraf dalam mencapai minimum global (atau mungkin hanya lokal saja) terhadap nilai error, serta cepat tidaknya proses pelatihan menuju konvergen. Apabila nilai bobot awal terlalu besar, maka input ke setiap lapisan tersembunyi atau lapisan output akan jatuh pada daerah dimana turunan fungsi sigmoidnya akan sangat kecil. Sebaliknya, apabila nilai bobot awal terlalu kecil, maka input ke setiap lapisan tersembunyi atau lapisan output akan sangat kecil, yang akan menyebabkan proses pelatihan akan berjalan sangat lambat.

Pelatihan BP dilakukan dalam rangka pengaturan bobot, sehingga pada akhir pelatihan akan diperoleh bobot-bobot yang optimal. Tujuannya adalah agar jaringan dapat menjalankan operasi sesuai dengan yang diinginkan. Hal ini dapat dipenuhi dengan melakukan proses pelatihan dengan menyesuaikan parameter-parameter atau bobot dan bias koneksi sehingga

untuk suatu input tertentu, jaringan dapat menghasilkan output yang sesuai dengan yang telah ditargetkan.

Proses penyesuaian dilakukan dengan memberikan sebuah *training set* (berupa suatu himpunan yang terdiri dari pasangan input dan output). *Training set* tersebut dievaluasi dengan suatu algoritma pelatihan tertentu. Selama proses pelatihan berlangsung, *training set* akan dievaluasi berkali-kali secara iteratif, tujuannya adalah untuk meminimumkan fungsi kuadrat errornya. Fungsi ini akan mengambil kuadrat error yang terjadi antara output dan target. Dengan metode ini kesalahan lokal tiap sel dapat dilihat sebagai bagian yang terkontribusi dalam menghasilkan kesalahan total pada lapisan output. Apabila kesalahan pada lapisan output dapat dipropagasikan kembali masuk ke lapisan tersembunyi, maka kesalahan lokal sel-sel pada lapisan tersebut dapat dihitung untuk mendapatkan perubahan bobot.

3. Algoritma Quasi Newton BFGS

Misalkan $\mathbf{w}_t$ adalah vektor bobot yang mengandung $w_j(t)$, $b2(t)$, $v_{ij}(t)$ dan $b1_j(t)$, konsep dasar metode Newton adalah :

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1} . \mathbf{g}_t \qquad (1)$$

dengan

- **w** : vektor bobot dan bias koneksi.
- $\mathbf{g}_t$: vektor gradien yang berisi $g_{j(t)}$, $g_{b2(t)}$, $g_{ij(t)}$ dan $g_{b1j(t)}$.
- **H** : matriks Hessian.

Matriks Hessian merupakan turunan kedua dari indeks kinerja terhadap bobot-bobot dan bias koneksi jaringan pada nilai ke-t, sehingga matriks Hessian didapatkan dengan cara yang lebih kompleks. Untuk menghindari penghitungan yang lebih kompleks tersebut matriks Hessian dapat ditentukan secara iteratif pada masing-masing *epoch* dengan memberikan inisial matriks Hessian pada awal pelatihan. Inisial matriks Hessian pada awal *epoch* harus bersifat simetrik dan definit positif. Matriks yang biasanya digunakan sebagai inisial matriks Hessian adalah matriks identitas. Hal ini disebabkan matriks identitas mudah didefinisikan serta merupakan matriks simetrik dan definit positif [1].

Salah satu algoritma perubahan bobot dengan metode newton adalah algoritma BFGS yang diperkenalkan oleh Broyden, Fletcher, Goldfarb dan Shanno. Algoritma pelatihan dengan metode *quasi Newton* adalah sebagai berikut [3]:

Langkah 0 :

a. Inisialisasi bobot awal dengan bilangan acak kecil
b. Inisialisasi Epoch 0, MSE $\neq 0$
c. Inisialisasi $\mathbf{H_0}$ yang merupakan matrik definit posotif simetrik.
d. Tetapkan Maksimum *Epoch* dan Target Error.

Langkah 1 :

Jika kondisi penghentian belum terpenuhi (*Epoch* < Maksimum *Epoch* atau MSE > Target Error), lakukan langkah berikutnya.

Langkah 2 :

Unit output Y menerima target pola yang berhubungan dengan pola input pelatihan. Kesalahan pada unit output didefinisikan sebagai :

$$e = (t - y)$$

Dengan :

- e : Kesalahan pada unit output
- *t* : Keluaran yang diinginkan (acuan/ target)
- *y* : Keluaran aktual

Fungsi jumlah kuadrat error didefinisikan dengan :

$$E = \frac{1}{2}(t-y)^2$$

Misalkan $\mathbf{w}_t$ adalah vektor bobot yang mengandung $w_j(t)$, $b2(t)$, $v_{ij}(t)$ dan $b1_j(t)$. Gradien fungsi kinerja terhadap nilai bobot dan bias koneksi pada waktu ke-t didefinisikan dengan :

$g_{j(t)} = -\ \delta 2\,(t).\ z_j\,(t)$

$g_{b2(t)} = -\ \delta 2\,(t)$

$g_{ij(t)} = -\ \delta 1_j\,(t)\ x_i$

$g_{b1j(t)} = -\ \delta 1_j\,(t)$

$\mathbf{g_t}$ merupakan vektor gradien yang berisi $g_{j(t)}$, $g_{b2(t)}$, $g_{ij(t)}$ dan $g_{b1j(t)}$.

Jika $\mathbf{g_t} = 0$ maka algoritma berhenti, jika tidak maka hitung :

$$\mathbf{d_t} = -\ \mathbf{H_t}.\mathbf{g_t}$$

Langkah 3 :

Tempatkan nilai α_t dengan menggunakan fungsi line search. Tujuannya adalah untuk meminimumkan error yang akan terjadi.

$$\alpha_t = arg\ min_{\ \alpha} \left[f\left(\boldsymbol{w}_t + \alpha_t \boldsymbol{d}_t\right)\right]$$

Perubahan vektor bobot dan bias yang terjadi adalah :

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \alpha_t . \mathbf{d_t}$$

Langkah 4 : Hitung perubahan bobot dan bias dengan persamaan berikut :

$$\Delta \mathbf{w_t} = \alpha_t . \mathbf{d_t}$$

Sedangkan perubahan arah pencarian adalah sebagai berikut :

$$\Delta \mathbf{g_t} = \mathbf{g_{t+1}} - \mathbf{g_t}$$

Maka didapatkan persamaan :

$$\mathbf{H_{t+1}}=\mathbf{H_t} + \left(1 + \frac{\Delta \boldsymbol{g}_t^{\,T} . \boldsymbol{H}_t . \Delta \boldsymbol{g}_t}{\Delta \boldsymbol{g}_t^{\,T} . \Delta \boldsymbol{w}_t}\right) . \frac{\Delta \boldsymbol{w}_t . \Delta \boldsymbol{w}_t^{\,T}}{\Delta \boldsymbol{w}_t^{\,T} . \Delta \boldsymbol{g}_t} - \frac{\boldsymbol{H}_t . \Delta \boldsymbol{g}_t . \Delta \boldsymbol{w}_t^{\,T} + (\boldsymbol{H}_t . \Delta \boldsymbol{g}_t . \Delta \boldsymbol{w}_t^{\,T})^T}{\Delta \boldsymbol{g}_t^{\,T} . \Delta \boldsymbol{w}_t}$$

Langkah 5 :
epoch = *epoch* + 1
Kembali ke langkah 2.

4. Algoritma Levenberg-Marquardt

Langkah dasar algoritma *Levenberg_Marquardt* adalah penentuan matriks Hessian untuk mencari bobot-bobot dan bias koneksi yang digunakan [1]. Matriks Hessian merupakan turunan kedua dari fungsi kinerja terhadap masing-masing komponen bobot dan bias. Untuk memudahkan proses komputasi, matriks Hessian diubah dengan pendekatan secara iteratif pada masing-masing *epoch* selama algoritma pelatihan berjalan. Proses perubahannya dilakukan dengan menggunakan fungsi gradien. Jika fungsi kinerja yang digunakan berbentuk jumlah kuadrat error (SSE), maka matriks Hessian dapat diestimasi dengan persamaan berikut:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} + \eta \mathbf{I} \qquad (2)$$

dimana :

- η : parameter Marquardt
- **I** : matriks identitas
- **J** : matriks Jakobian yang terdiri dari turunan pertama error jaringan terhadap masing-masing komponen bobot dan bias.

Matriks Jakobian dapat dikomputasikan melalui teknik backpropagation standar [4]. Matriks Jakobian tersusun dari turunan pertama fungsi error terhadap masing-masing komponen bobot dan bias koneksi jaringan. Nilai parameter Marquardt (η) dapat berubah pada setiap *epoch.* Jika setelah berjalan satu *epoch* nilai fungsi error menjadi lebih kecil, nilai η akan dibagi oleh faktor τ. Bobot dan bias baru yang diperoleh akan dipertahankan dan pelatihan dapat dilanjutkan ke *epoch* berikutnya. Sebaliknya, jika setelah berjalan satu *epoch* nilai fungsi error menjadi lebih besar maka nilai η akan dikalikan dengan faktor τ. Nilai perubahan bobot dan bias dihitung kembali sehingga menghasilkan nilai yang baru.

Algoritma pelatihan dengan metode *Levenberg_Marquardt* dapat dijabarkan sebagai berikut :

Langkah 0 :

- Inisialisasi bobot awal dengan bilangan acak kecil
- Inisialisasi Epoch 0, MSE $\neq 0$
- Tetapkan Maksimum *epoch*, parameter Levenberg_Marquardt ($\eta > 0$), faktor τ dan target Error

Langkah 1 :
Jika kondisi penghentian belum terpenuhi (*epoch* < Maksimum *epoch* atau MSE > target Error), lakukan langkah berikutnya.

Langkah 2 :

- *Epoch* = *epoch* + 1
- Untuk setiap pasangan data pelatihan, lakukan langkah 3 – 4.

Langkah 3 :
Unit output Y menerima target pola yang berhubungan dengan pola input pelatihan. Jika diberikan N pasangan input data pelatihan (x_r, t_r), r = 1,2,...,N, dengan x_r adalah input dan t_r target yang akan dicapai. Kesalahan pada suatu data pelatihan ke-r didefinisikan sebagai:

$$e_r = t_r - y_r$$

dengan :

- e_r : Kesalahan pada unit output
- t_r : Keluaran yang diinginkan (acuan/ target)
- y_r : Keluaran aktual.

e adalah vektor kesalahan berukuran Nx1 yang tersusun dari e_r, r = 1,2,...,N. **e** dapat dituliskan sebagai :

$$\mathbf{e} = \begin{bmatrix} e_1 & e_2 & \dots & e_N \end{bmatrix}^T$$

Misal bobot dan bias koneksi dinyatakan dalam vektor **w**, **w** merupakan vektor berukuran ((2+n)p+1)x1 dapat dituliskan sebagai :

$$\mathbf{w} = \begin{bmatrix} w_j & b2 & v_{ij} & b1_j \end{bmatrix}^T$$

Kesalahan suatu pelatihan jaringan oleh vektor bobot dan bias koneksi **w** pada suatu data pelatihan ke-r menjadi :

$e_r(\mathbf{w}) = (t_r - y_r)$
$= (t_r - f (x_r, \mathbf{w}))$

Vektor kesalahan oleh vektor bobot dan bias koneksi **w** menjadi **e(w)** berukuran Nx1 yang tersusun dari $e_r(\mathbf{w})$, dengan r = 1,2,...,N.

Hitung fungsi jumlah kuadrat error dengan persamaan :

$$E(\mathbf{w}) = \frac{1}{2}\ \mathbf{e}^T(\mathbf{w})\ \mathbf{e}(\mathbf{w})$$

Hitung matriks Jacobian untuk vektor bobot dan bias koneksi :

$$\mathbf{J}(\mathbf{w}) = \left[\frac{\partial e_r}{\partial \boldsymbol{w}}\right]_{Nx((2+n)p+1)},$$

untuk r = 1,2,...,N

a. Hitung matriks Hessian untuk vektor bobot dan bias koneksi.
$$\mathbf{H(w)} = \left[\boldsymbol{J^T(w)J(w)} + \eta \boldsymbol{I}\right]_{((2+n)p+1)x((2+n)p+1)}$$

b. Hitung perubahan vektor bobot dan bias dengan persamaan berikut :
$$\Delta \mathbf{w} = -\left[\left[\boldsymbol{H(w)}\right]^{-1} \boldsymbol{J^T(w)e(w)}\right]_{((2+n)p+1)x1}$$

c. Hitung vektor bobot dan bias baru.
w (baru) = w (lama) + Δ w

d. Hitung kesalahan yang terjadi oleh bobot dan bias koneksi yang baru.
$$E(\mathbf{w(baru)}) = \frac{1}{2}\ \mathbf{e(w\ (baru))}^T\ \mathbf{e(w\ (baru))}$$

e. Bandingkan E(**w**) dengan E(**w(baru)**).
- Jika E(**w**) <= E(**w(baru)**) maka didapatkan
  $\eta = \eta * \tau$
  dan kembali ke langkah a.
- Jika E(**w**) > E(**w(baru)**) maka didapatkan
  $\eta = \eta / \tau$
  w (t+1) **=** **w** (t) **+** Δ **w**

Kembali ke langkah 2.

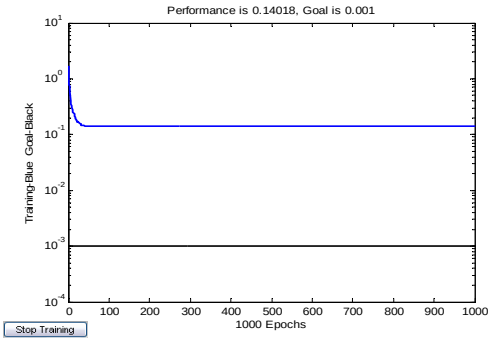
5. Prediksi Curah Hujan

Data yang digunakan adalah data bulanan curah hujan kota Semarang mulai Januari 1994 – Desember 2003 sebanyak 120 data dimana 100 data digunakan sebagai data training dan sisanya 20 data sebagai data testing. Berdasarkan pembentukan model ARIMA menggunakan fungsi autokorelasi dan autokorelasi parsial input yang dipilih adalah data lag 1, 12 dan 13. Perhitungan dilakukan menggunakan program Matlab 7.1 dengan jumlah epoch dibatasi sebanyak 1000, tiga unit input, 5 unit hidden dan masing-masing proses diulang sebanyak 100 kali. Hasil perhitungan disajikan pada tabel berikut.

Tabel 1. Hasil perhitungan model FFNN dengan metode Levenberg-Marquardt dan BFGS

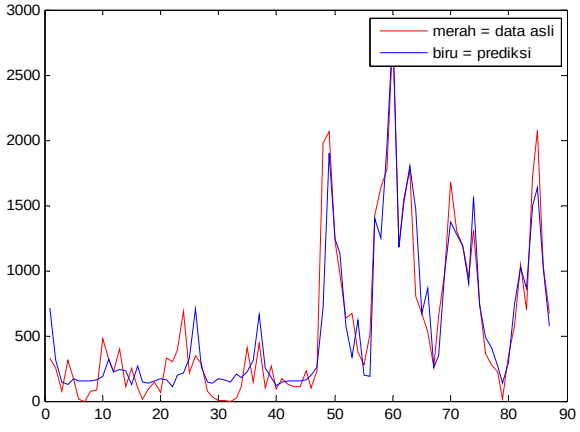
	Levenberg-Marquardt	BFGS
Unit input	Lag 1, 12 dan 13	Lag 1, 12 dan 13
Jumlah epoch	1000	1000
Learning rate	0.1	
Jumlah replikasi	100	100
MSE training	0.0609	0.0114
MSE testing	4.1123	1.8087
Jumlah unit hidden	5	5

Dari tabel 1 nampak bahwa metode BFGS memberikan keakuratan prediksi yang lebih baik pada training maupun testing. Proses iterasi menuju konvergen dari proses terakhir disajikan pada gambar 2.

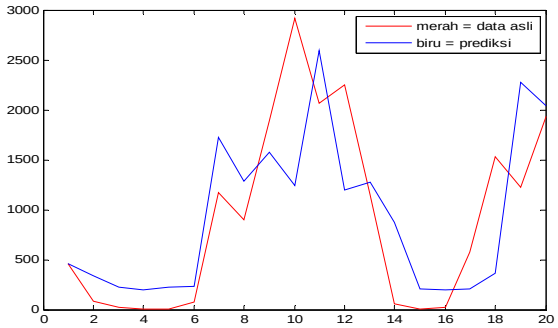


Gambar 2. Proses iterasi sampai 1000 epoch

Hasil prediksi *in sample* dan *out sample* pada proses terakhir disajikan pada gambar 3 dan grafik 4. Nampak bahwa model menghasilkan prediksi yang cukup baik.



Gambar 3. Grafik prediksi in-sample data curah hujan dengan metode BFGS



Gambar 4. Grafik prediksi out-sample data curah hujan dengan metode BFGS

6. Penutup

Berdasarkan pembahasan dapat disimpulkan bahwa pelatihan terhadap jaringan FFNN pada data curah hujan kota Semarang dengan menggunakan algoritma *Quasi Newton BFGS* lebih baik digunakan dalam rangka meminimumkan kesalahan yang terjadi jika dibandingkan dengan algoritma *Levenberg-Marquardt*.

Daftar Pustaka

[1] Al-Haik, M.S., Garmestani, H. and Navon, I.M., 2003, *Truncated-Newton Training Algorithm for Neurocomputational Viscoplastic Model*, Comput. Methods Appl. Mech. Engrg., No. 192, p. 2249-2267

[2] Bolat, S. and Kalenderli, O., 2003, *Electrode Contour Optimization by Artificial Neural Network with Levenberg-Marquardt Algorithm*, IJCI Proceeding of Int. XII Turkish Symposium on Artificial Intelligence and Neural Network, Vol. 1 No. 1.

[3] Bishop, Christopher, M., 1995, *Neural Networks for Pattern Recognition*, Oxford University Press, New York

[4] Chatfield, C. and Faraway, J., 1998, *Time Series Forecasting with Neural Networks: a Comparative Study Using the Airline Data*, Applied Statistics

[5] Fausett, L., 1994, *Fundamentals of Neural Networks; architectures, algorithms and applications*, Prentice-Hall Inc., Englewoods Cliffs, New Jersey

[6] Zhou, G. and Si, J., 1998, *Advanced Neural Network Training Algorithm with Reduced Complexity Based on Jacobian Deficiency*, IEEE Transactions on Neural Network, Vol. 9, No. 3, May