



UNIVERSITAS INDONESIA

**SISTEM BERBASIS PENGETAHUAN
LAPORAN PROYEK JARINGAN SYARAF TIRUAN UNTUK DIRECT
INVERSE CONTROL**

Hansel Matthew (1806194914)

**FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO**

**DEPOK
MEI 2021**

DAFTAR ISI

DAFTAR ISI.....	2
BAB I PENDAHULUAN.....	3
2.1 Latar Belakang.....	3
2.2 Rumusan Masalah	3
2.3 Tujuan Penelitian.....	4
BAB II DASAR TEORI.....	5
2.1 Artificial Neural Network	5
2.2 Backpropagation	6
A. Inisialisasi bobot.....	8
B. Data Preprocessing	9
C. Algoritma Pembelajaran	9
2.3 Sistem Kendali Berbasis Artificial Neural Networks	11
2.4 Direct Inverse Control.....	12
BAB III IDENTIKASI SISTEM DAN PENGENDALI MENGGUNAKAN ARTIFICIAL NEURAL NETWORK	14
3.1 Pembuatan Database	14
3.2 Proses Pelatihan	14
3.3 Proses Pengujian	19
BAB IV DESAIN PENGENDALI INVERS MENGGUNAKAN ARTIFICIAL NEURAL NETWORK	22
5.1 Pembuatan Database	22
5.2 Proses Pelatihan	22
5.3 Proses Pengujian	26
BAB V PENERAPAN DIRECT INVERSE CONTROL MENGGUNAKAN ARTIFICIAL NEURAL NETWORK	29
5.1 Kendali Metode Direct Inverse Control.....	29
BAB VI KESIMPULAN.....	33

BAB I

PENDAHULUAN

2.1 Latar Belakang

Sistem kendali adalah cabang ilmu yang fokus mempelajari pengendalian respon dari sebuah sistem. Sistem tersebut dapat datang dari banyak ruang lingkup seperti pada sistem proses kimia, produksi tenaga listrik, bidang manufaktur, produksi minyak dan gas, dan masih banyak lagi. Sistem kendali pertama yaitu sistem kendali klasik PID dikembangkan pada tahun 1940 untuk pengendali dalam proses industri. 90% dari lup pengendalian menggunakan pengendali PID (Proporsional – Integral – Diferensial). Sistem kendali klasik ini Sebagian besar masih bekerja secara manual dengan parameter ditentukan secara trial and error. Penalaan ulang dari parameter ini menghabiskan waktu yang lama serta memakan biaya yang tinggi. Sering sekali industri menggunakan pengendali dengan nilai parameter *default* sehingga memberikan kinerja yang tidak sama untuk perubahan dari titik operasi.

Seiring berkembangnya zaman, sistem kendali menjadi semakin dibutuhkan, hal itu dikarenakan kemampuan manusia tidak dapat lagi mengalahkan kemampuan mesin dalam hal mengendalikan dengan akurat dan presisi. Sistem kendali pun mengalami beberapa perkembangan menjadi berbagai jenis seperti optimal control, robust control, non linear control, dan lain lain. Salah satu perkembangannya adalah intelligent control, yaitu suatu pengendali adaptif yang berbasis pengetahuan. Terciptalah suatu sistem yang memiliki cara kerja mirip dengan sistem kerja jaringan saraf manusia yang dinamakan *Artificial Neural Network*. *Artificial Neural Network*. *Artificial Neural Network* (ANN) atau Jaringan Saraf Tiruan adalah suatu sistem adaptif yang dapat mengubah strukturnya untuk memecahkan masalah berdasarkan informasi eksternal maupun internal yang mengalir melalui jaringan tersebut. Sistem ini tidak menggunakan sebuah pemodelan matematis karena menggunakan pemodelan *blackbox*. Pemodelan ini dilakukan dengan cara menganalisa perilaku sistem jika diberikan variasi pasangan input dan output. Dari perilaku inilah maka jaringan saraf tiruan akan berusaha untuk menghasilkan output yang diinginkan.

2.2 Rumusan Masalah

- Bagaimana konsep dasar jaringan saraf tiruan?
- Bagaimana cara kerja metode pembelajaran jaringan saraf tiruan dengan menggunakan algoritma *backpropagation*?
- Bagaimana suatu jaringan saraf tiruan dapat mengklasifikasikan suatu dataset?

2.3 Tujuan Penelitian

Berdasarkan permasalahan yang telah dipaparkan, maka tujuan dari laporan ini adalah untuk menganalisis dan mempelajari cara kerja dari jaringan saraf tiruan pada dataset yang diberikan.

BAB II DASAR TEORI

2.1 Artificial Neural Network

Artificial Neural Network (ANN) atau Jaringan Saraf Tiruan lahir dari usaha manusia yang ingin memodelkan otak manusia untuk membuat suatu sistem yang paling sempurna. Model ANN pertama kali dikenalkan oleh Mc. Culloh dan Pitts sebagai komputasi dari aktivitas syarat. Hasil penelitian mereka menjadi dasar bagi penelitian di bidang ANN pada masa berikutnya.

Pada tahun 1958, Rosenblat, Widrow dan Hoff pertama kali menemukan aturan pembelajaran pada perceptrons. Minski dan Papert (1969) meyakini bahwa penggunaan perceptrons sangat terbatas yaitu hanya sebagai metode perhitungan dalam kehidupan nyata. Bernard Widrow menemukan neuron sederhana yang mirip dengan perceptron yang disebut ADALINE (neuron linier adaptif), dan jaringan multilayernya yang disebut MADALINE (multiple adaline). Selanjutnya, Widrow juga mengembangkan program pembelajaran terbimbing yang disebut dengan metode pembelajaran Least Mean Square (LMS) atau Widrow Hoff. Di era berikutnya, jaringan syaraf tiruan berkembang sedemikian rupa sehingga menemukan berbagai metode pembelajaran dan aturan pembelajaran.

ANN merupakan jaringan yang dibuat dengan meniru jaringan syaraf manusia dengan diilhami oleh struktur dan cara kerja otak dan sel syaraf manusia. Sebuah neuron memiliki sebuah badan sel, pengirim sinyal dan penerima sinyal. Neuron biologis pada manusia dapat dilihat pada gambar 2.1

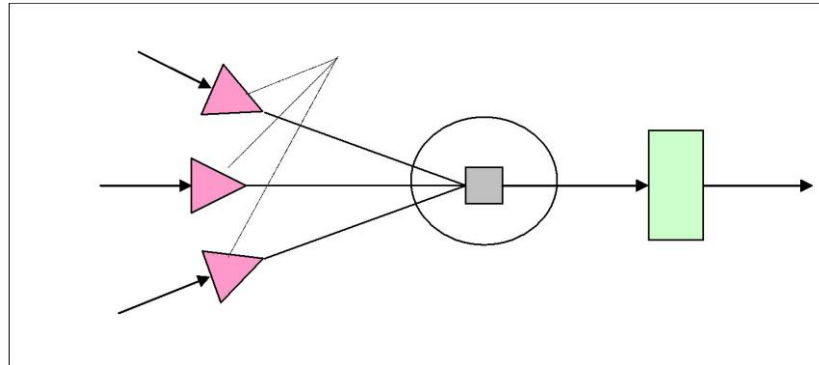


Gambar 2.1. Ilustrasi Neuron Biologis Manusia

Cara kerja dari sebuah neuron adalah akan bereaksi apabila potensial listrik mencapai suatu batasan tertentu. Neuron akan menjumlahkan sinyal yang masuk melalui dendrite yang dikalikan dengan pembobot sinapsis[1]. Proses pembelajaran terjadi dengan perubahann yang

terjadi pada sinapsis. Sinyal yang masuk akan dijumlahkan dan dikonversi dengan suatu fungsi aktivitas yang kemudian akan mengeluarkan suatu sinyal pemicu yang dialirkan ke neuron lain.

Prinsip kerja dari neuron ini kemudian dimodelkan secara matematis. Model matematik orde pertama dari neuron dapat dilihat pada Gambar 2.2



Gambar 2.2. Model Matematik dari Neuron

Pemodelan matematika inilah yang menjadi dasar dari ANN. Elemen dasar dari sebuah ANN adalah sebuah neuron. Neuron ini akan mengubah sinyal masukan menjadi sebuah keluaran. Setiap neuron mempunyai suatu inputan yang memiliki bobotnya masing – masing. Sinyal kemudian akan dikonversi menggunakan sebuah fungsi aktivasi. Fungsi aktivasi yang digunakan dapat beragam seperti *Relu*, *sigmoid*, *tanh* dan lain lain

2.2 Backpropagation

Backpropagation adalah suatu algoritma ANN yang memiliki kekuatan utama pada klasifikasi suatu pola atau disebut sebagai *pattern recognition*. Jaringan syaraf *backpropagation* dapat digunakan untuk memprediksi luaran dari sebuah sistem. Pada mulanya data akan diberikan pada sistem. Jika sistem menghasilkan luaran yang tidak sesuai dengan yang diharapkan, maka *backpropagation* akan memodifikasi bobot pada hubungan neuron.[2]

Backpropagation menggunakan gradient descent untuk mengoptimasi nilai dari bobot tersebut. Gradient descent bertujuan adalah untuk terus mengambil sampel gradien parameter model ke arah yang berlawanan berdasarkan bobot w serta memperbarui secara konsisten hingga ANN dapat mencapai fungsi minimum global $J(w)$. Gradient descent memiliki suatu parameter yaitu Learning Rate (α) yang mengatur seberapa besar perubahan bobot pada setiap epoch.

Pada gradient descent dikenal juga istilah momentum. Momentum adalah suatu metode agar mempertahankan arah dari gradient descent. Hal ini diperoleh dengan mengkombinasikan

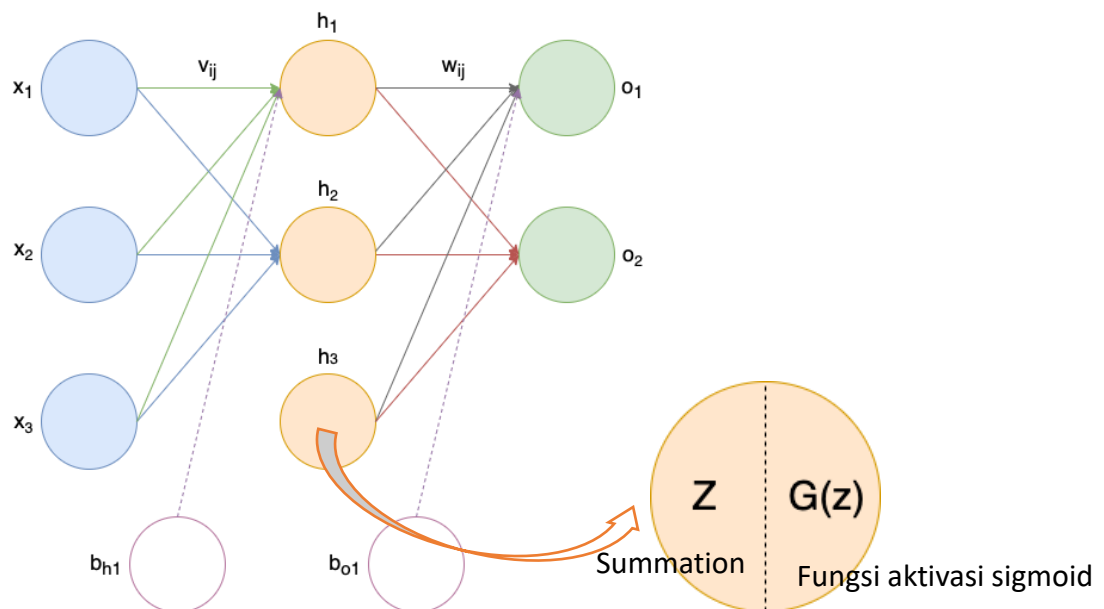
arah yang dikomputasi pada iterasi sekarang dengan iterasi sebelumnya. Besar pengaruh arah iterasi sebelumnya dapat diatur dengan mengatur koefisien momentum (μ).

ANN terdiri dari 3 layer, yaitu *input layer* yang memiliki neuron sebanyak I buah, *hidden layer* yang memiliki neuron sebanyak J buah, dan *output layer* yang memiliki neuron sebanyak K buah. Antara layer dihubungkan dengan bobot. Antara *input layer* dengan *hidden layer* dihubungkan dengan bobot w . Antara layer *hidden layer* dengan *output layer* dihubungkan dengan bobot v . Pada layer input dan hidden ditambah sebuah neuron bobot bias, yang berguna sebagai konstanta.[3]

Algoritma *backpropagation* dapat dibagi menjadi dua buah proses yaitu training dan testing. Dimana kedua proses tersebut dapat dibagi lagi menjadi:

1. Training
 - a. Proses pengolahan data input (*feedforward*).
 - b. Perhitungan error (*backpropagation*).
 - c. Pembaruan bobot.
2. Testing
 - a. Pengelompokkan nilai output (*Quantizing*).
 - b. Perhitungan *Recognition Rate*.

Diagram sederhana pemodelan jaringan saraf tiruan yang digunakan dapat dilihat pada Gambar 2.3.



Gambar 2.3. Struktur Neural Networks Sederhana

Keterangan:

∂ x \rightarrow Input layer

∂ h \rightarrow Hidden layer

∂_o	\rightarrow Output layer	∂_v	\rightarrow Bobot hidden layer
∂_i	\rightarrow Ukuran input layer	∂_w	\rightarrow Bobot output layer
∂_j	\rightarrow Ukuran hidden layer		

Berikut merupakan algoritma pemrograman *backpropagation*:

A. Inisialisasi bobot

Inisialisasi awal dari tiap bobot hubungan antar neuron dapat dilakukan menggunakan dua buah metode yaitu random dan nguyen widrow.

1. Random

Metode inisialisasi random adalah metode dimana bobot awal ditentukan secara acak. Hal ini dapat kita lakukan menggunakan fungsi random pada MATLAB. Inisialisasi awal yang acak dibutuhkan untuk menghasilkan symmetry breaking. Suatu strategi yang efektif untuk diterapkan pada inisialisasi random adalah menentukan batas atas dan batas bawah dari nilai acak tersebut. Hal ini dilakukan untuk memastikan bahwa bobot neuron tidak terlalu besar sehingga proses pembelajaran lebih efisien.

2. Nguyen Widrow

Metode nguyen widrow adalah sebuah algoritma modifikasi sederhana dari bobot yang mampu meningkatkan kecepatan proses pembelajaran. Algoritma nguyen-widrow adalah sebagai berikut:

- Menentukan besar faktor skala (β)

$$\beta = 0,7 * J^{\frac{1}{I}}$$

Dengan: J = Ukuran *hidden layer*; I = Ukuran *input layer*.

- Inisialisasi bobot secara random pada rentang -0.5 sampai 0.5
- Menghitung norm dari vektor bobot

$$\|v_j\| = \sqrt{\sum_{i=1}^I v_{ij}^2} \text{ dan } \|w_k\| = \sqrt{\sum_{j=1}^J w_{jk}^2}$$

- Memperbaharui bobot

$$v_{ij} = \frac{\beta v_{ij}}{\|v_j\|} \quad w_{jk} = \frac{\beta w_{jk}}{\|w_k\|}$$

- Mengatur bias sebagai bilangan acak antara β sampai $-\beta$

B. Data Preprocessing

Preprocessing pertama adalah normalisasi data yaitu proses mengubah data pada suatu feature agar mempunyai rentang yang sama yaitu diantara 0-1. Hal ini dilakukan agar seluruh feature mempunyai rentang yang sama sehingga tidak ada feature yang mendominasi satu dengan yang lainnya

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

C. Algoritma Pembelajaran

Algoritma untuk proses pembelajaran adalah sebagai berikut:
Selama kondisi stopping FALSE, maka untuk setiap pasangan pelatihan:

Proses *forward pass*

1. Komputasi *Input Layer*. Setiap x_i , $i = 1, 2, \dots, I$
 - Menerima input x_i
 - Mengirimkannya ke semua *Hidden Layer* di atasnya.
2. Komputasi *Hidden Layer*. Setiap h_j , $j = 1, 2, \dots, J$
 - $z_in_j = v_{0j} + \sum x_{pi} v_{ij}$
 - $z_j = \frac{1}{1 + e^{-z_in_j}}$
3. Komputasi *Output Layer*. Setiap o_k , $k = 1, 2, \dots, K$
 - $o_in_k = w_{0k} + \sum z_j w_{jk}$
 - $o = \frac{1}{1 + e^{-y_in_k}}$
 - Kuantisasi output
 - $0 \leq O_k < 0.3 \rightarrow O_k = 0$
 - $0.7 < O_k \leq 1 \rightarrow O_k = 1$
 - Else $\rightarrow O_k = O_k$.

Pada layer output diterapkan suatu threshold untuk membantu proses pembelajaran. Hal ini dilakukan karena sebuah output yang bernilai 1 atau 0 pada neuron output nyaris mustahil untuk tercapai tanpa adanya fungsi matematika tambahan. Threshold yang dipilih adalah 0.7 untuk batas atas dan 0.3 untuk batas bawah. Sehingga output yang bernilai lebih besar dari 0.7 akan dikonversi menjadi nilai biner 1 dan output yang bernilai lebih kecil dari 0.3 akan dikonversi menjadi nilai biner 0

Proses *backward pass*

1. Komputasi di *Output Layer*. Setiap o_k , $k = 1, 2, \dots, K$
 - Menghitung informasi error:
$$\delta_k = (t_{pk} - y_{pk}) \cdot f'(y_{in_k})$$
 - Menghitung besarnya koreksi bobot *output layer*:
$$\Delta w_{jk} = \alpha \delta_k z_j$$
 - Menghitung besarnya koreksi bias output:
$$\Delta w_{0k} = \alpha \delta_k$$
4. Komputasi di *Hidden Layer*. Setiap h_j , $j = 1, 2, \dots, J$
 - Menghitung semua koreksi error:
$$\delta_{in_j} = \sum \delta_k w_{jk}$$
 - Menghitung nilai aktivasi koreksi error :
$$\delta_j = \delta_{in_j} f'(z_{in_j})$$
 - Menghitung koreksi bobot unit hidden :
$$\Delta v_{ij} = \alpha \delta_j x_i$$
 - Menghitung koreksi error bias unit hidden :
$$\Delta v_{0j} = \alpha \delta_j$$

Proses memperbaharui bobot

1. Bobot ke *hidden layer*:
$$v_{ij}(\text{baru}) = v_{ij}(\text{lama}) + \Delta v_{ij}$$
$$v_{0j}(\text{baru}) = v_{0j}(\text{lama}) + \Delta v_{0j}$$
2. Bobot ke *output layer*:
$$w_{jk}(\text{baru}) = w_{jk}(\text{lama}) + \Delta w_{jk}$$
$$w_{0k}(\text{baru}) = w_{0k}(\text{lama}) + \Delta w_{0k}$$

Proses perhitungan error

Error dihitung dengan menggunakan fungsi kesalahan fungsi error kuadratis:

$$E = \frac{1}{2} \sum (t_{pk} - y_{pk})^2$$

D. Algoritma Pengujian

Setiap $p = 1$ sampai $p = P_u$:

Proses *feedforward*

3.1. Komputasi *Input Layer*. Setiap $x_{uji_{pi}}$, $i = 1, 2, \dots, I$

- Menerima input $x_{uji_{pi}}$
- Mengirimkannya ke semua *Hidden Layer*.

3.2. Komputasi *Hidden Layer*. Setiap Z_j , $j = 1, 2, \dots, J$

- $z_{in_uji_j} = v_{0j} + \sum x_{uji_{pi}} v_{ij}$
- $z_{uji_j} = \frac{1}{1 + e^{-z_{in_uji_j}}}$

3.3. Komputasi *Output Layer*. Setiap y_k , $k = 1, 2, \dots, K$

- $y_{in_uji_k} = w_{0k} + \sum z_{uji_j} w_{jk}$
- $y_{uji_{pk}} = \frac{1}{1 + e^{-y_{in_uji_k}}}$
- Kuantisasi output
 - $0 \leq y_{uji_{pk}} < 0.3 \rightarrow y_{rec_{pk}} = 0$
 - $0.7 < y_{uji_{pk}} \leq 1 \rightarrow y_{rec_{pk}} = 1$
 - Else $\rightarrow y_{rec_{pk}} = y_{uji_{pk}}$.

3.4. Menghitung *recognition rate*:

$$RR = \frac{\text{banyaknya data output yang benar}}{\text{banyaknya data pengujian}} \times 100\%$$

Output dikatakan benar jika nilai biner nya sama dengan binernya target.

2.3 Sistem Kendali Berbasis Artificial Neural Networks

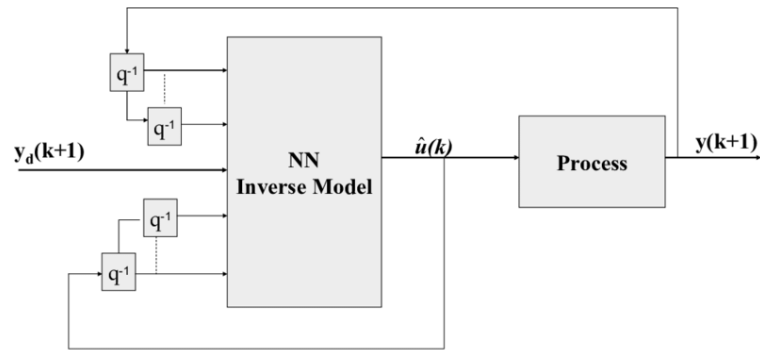
ANN dapat digunakan pada sistem kendali. Pengendali ini akan tidak berbasis pada sebuah model matematis[4]. Hal ini dilakukan karena terdapat beberapa plant yang sulit untuk dimodelkan secara matematis. Fungsi dari sebuah plant biasanya memiliki beberapa sifat yaitu:

- Non Linear
- Cross coupling
- Underactuated
- Tidak mampu ditulis model matematikanya

ANN akan digunakan untuk membuat sebuah fungsi transfer sebuah sistem rumit dimana sistem rumit adalah sistem yang belum tentu mempunyai fungsi transfer yang jelas. Fungsi transfer dapat berupa fungsi spatial dan fungsi waktu. Fungsi transfer ini disimpan dalam bentuk bobot dalam struktur arsitektur ANN tersebut.

Terdapat beberapa perbedaan yang harus kita buat ketika menggunakan ANN untuk sistem kendali. Komponen perubahan tersebut dapat dilihat pada tabel dibawah ini

dengan perubahan dalam dinamika sistem dan tahan terhadap ketidakpastian di lingkungannya. Teknik DIC yang digabungkan dengan jaringan syaraf tiruan dipilih sebagai pengontrol karena mudah diimplementasikan dalam batasan perangkat lunak komputer kontrol penerbangan dan pengontrol dapat disetel untuk mengubah parameter status menggunakan ANN. Berikut merupakan skema dari direct inverse control



BAB III

IDENTIKASI SISTEM DAN PENGENDALI MENGGUNAKAN ARTIFICIAL NEURAL NETWORK

3.1 Pembuatan Database

Data dari plant akan di generate dengan menghasilkan sebanyak 3010 angka random. 3010 angka tersebut kemudian kita masukkan kedalam persamaan plant sehingga menghasilkan sebanyak 3010 data output. Pasangan input dan output ini akan menjadi database untuk proses identifikasi sistem.

$$y(k) = \frac{1}{1 + y(k-1)^2} + 0.25 x(k) - 0.3 x(k-1)$$

Berikut merupakan program MATLAB untuk pembuatan database:

```
%% Generate Data
rng(42, 'twister');

% Input random
u = 2*rand(3010,1) - 1;

for k = 2:length(u)
    y(k,1) = (1/(1+y(k-1,1)^2)) + 0.25*u(k) - 0.3*u(k-1);
end
```

Program akan menghasilkan 3010 buah data untuk acak biasa pada variable y dan 3010 buah data acak sinusoid pada variable y_sinusoid. **uk** merupakan data input **u(k)** yang didapat melalui proses randomisasi yang telah di normalisasi. Sedangkan **yk** merupakan target **y(k)** yang didapat melalui fungsi

Data kemudian dinormalisasi menggunakan metode max-min agar mendapatkan nilai random pada range -1 dan +1.

```
%% Data Input
in = [y(2:2101) y(1:2100) u(3:2102) u(2:2101) u(1:2100)];
target = [y(3:2102)];
[in_row,in_col] = size(in);
[target_row,target_col] = size(target);
n_data = in_row;

%% Normalisasi
in = normalize(in);

target = normalize(target);
```

3.2 Proses Pelatihan

Pada proses learning identifikasi plant, data yang digunakan adalah data yang berasal dari generate data sebelumnya. Masukan dari sistem akan berjumlah 5 yaitu $y(k-1), y(k-2), x(k), x(k-2), x(k-1)$. Data dibagi menjadi 50:50 menjadi 2 bagian yaitu training dan testing.

Inisialisasi bobot menggunakan metode nguyen-widrow dengan program MATLAB sebagai berikut:

```
%% Inisialisasi

% Input layer ke hidden layer
beta = 0.7*n_hidden^(1/n_in);
v_ij = rand(n_in,n_hidden) - 0.5;
for i = 1:n_hidden
    norma(i) = sqrt(sum(v_ij(:,i).^2));
    v_ij(:,i) = (beta*v_ij(:,i))/norma(i);
end
v_0j = (rand(1,n_hidden) - 0.5)*beta*2;

% Hidden layer ke output layer
w_jk = rand(n_hidden,n_out) - 0.5;
w_0k = rand(1,n_out) - 0.5;
```

Kemudian program akan melakukan proses feedforward dan backpropagation yang akan mengupdate bobot sehingga sistem dapat menghasilkan luaran yang sama dengan persamaan plant yang menghasilkan data pada database.

Program MATLAB untuk feedforward dan backpropagation adalah sebagai berikut:

```
%% Hyperparameter & Variabel
n_in = in_col;           %jumlah unit input layer
n_hidden = 30;           %jumlah unit hidden layer
n_out = target_col;      %jumlah unit output layer

alpha = 0.001;
miu = 0.9;

rng(25, 'twister')

%% Inisialisasi

% Input layer ke hidden layer
beta = 0.7*n_hidden^(1/n_in);
v_ij = rand(n_in,n_hidden) - 0.5;
for i = 1:n_hidden
    norma(i) = sqrt(sum(v_ij(:,i).^2));
    v_ij(:,i) = (beta*v_ij(:,i))/norma(i);
end
v_0j = (rand(1,n_hidden) - 0.5)*beta*2;

% Hidden layer ke output layer
```

```

w_jk = rand(n_hidden,n_out) - 0.5;
w_0k = rand(1,n_out) - 0.5;

%% TRAINING

maxepoch = 20000;
targeterror = 0.00001;

stop = 0;
epoch = 1;
delta_wjk_old = 0;
delta_w0k_old = 0;
delta_vij_old = 0;
delta_v0j_old = 0;

%% NN

while stop == 0 && epoch <= maxepoch
    for n=1:n_data
        % Feedforward
        xi = in(n,:);
        ti = target(n,:);

        % Komputasi input layer ke hidden layer
        z_inj = xi*v_ij + v_0j;
        for j=1:n_hidden
            zj(1,j) = (1-exp(-z_inj(1,j))) / (1+exp(-z_inj(1,j)));
        end

        % Komputasi hidden layer ke output layer
        y_ink = zj*w_jk + w_0k;
        for k=1:n_out
            yk(n,k) = y_ink(k);
        end

        % Store error
        error(n,1) = 0.5*sum((yk(n,1) - ti).^2);

        % Backpropagation of error %

        % Komputasi dari output layer ke hidden layer
        d_ok = (yk(n,1) - ti);
        delta_wjk = alpha*zj'*d_ok + miu*delta_wjk_old;
        delta_w0k = alpha*d_ok + miu*delta_w0k_old;
        delta_wjk_old = delta_wjk;
        delta_w0k_old = delta_w0k;

        % Komputasi dari hidden layer ke input layer
        doinj = d_ok*w_jk';
        d_oj = doinj.*(1-zj.^2)/0.5;
        delta_vij = alpha*xi'*d_oj + miu*delta_vij_old;
        delta_v0j = alpha*d_oj + miu*delta_v0j_old;
        delta_vij_old = delta_vij;
        delta_v0j_old = delta_v0j;

        % Memperbarui bobot dan bias
        w_jk = w_jk - delta_wjk;

```



```

        w_0k = w_0k - delta_w0k;
        v_ij = v_ij - delta_vij;
        v_0j = v_0j - delta_v0j;
    end
    errorpereepoch(1,epoch) = sum(error)/n_data;

    if errorpereepoch(1,epoch) < targeterror
        stop = 1;
    end

    epoch = epoch+1;
end

%% Plot

epoch = epoch - 1;
figure;
plot(errorpereepoch);
ylabel('Error per epoch'); xlabel('Epoch')
disp("Error per epoch minimum = "+ min(errorpereepoch) *100 +" %");
disp("Error akhir = "+ errorpereepoch(1,epoch)*100+" %");

%% Plot MSE & RMSE
MSE_train = (sum((target-yk).^2))/n_data;
RMSE_train = sqrt(MSE_train);
disp("MSE training = "+ MSE_train); disp("RMSE training = "+
RMSE_train);

figure;
plot(yk, 'o');
hold on
plot(target, 'x');
xlim([0 n_data]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN', 'Output Plant');
hold off

```

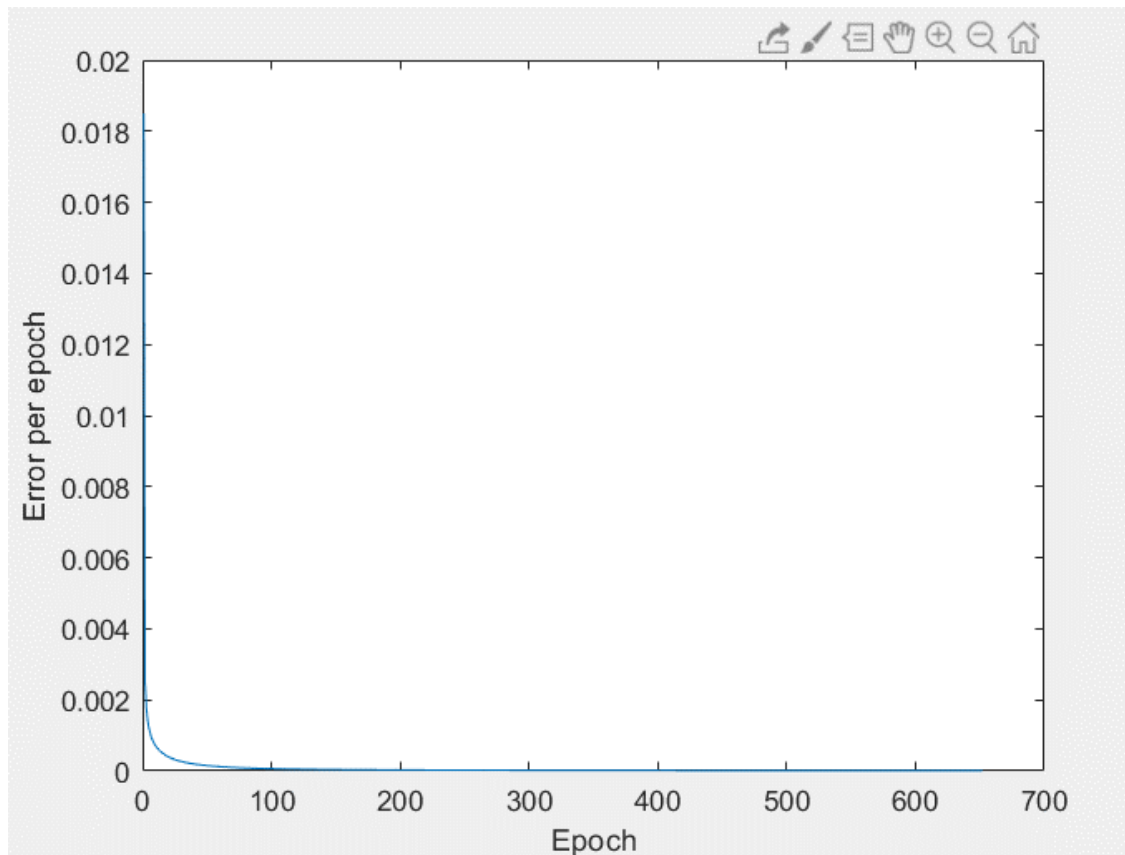
Didapatkan hasil dari proses pelatihan sebagai berikut:

```

Error per epoch minimum = 0.0009995 %
Error akhir = 0.0009995 %
MSE training = 1.999e-05
RMSE training = 0.004471

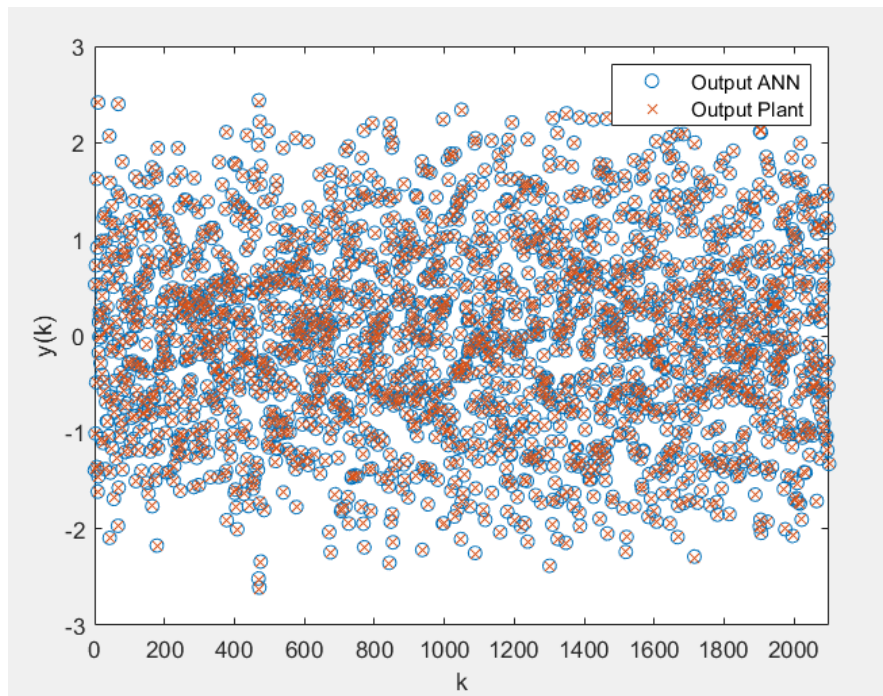
```

Pada setiap epoch, bobot pada jaringan akan diupdate. Semakin banyak neuron hidden layer yang digunakan, semakin kecil besar epoch yang dibutuhkan untuk mendapatkan nilai output yang sesuai dengan target. Hubungan antara error dan epoch kemudian di plot untuk dilihat penurunan nilai error pada jaringan. Berikut merupakan plot hubungan antara epoch dan error pada input acak.



Dapat dilihat dari grafik bahwa penurunan error sudah tidak mengalami perubahan yang signifikan setelah epoch 100. Sehingga dapat disimpulkan bahwa perubahan bobot lebih dominan terjadi pada epoch dibawah 100. Epoch yang digunakan berjumlah sedikit karena ketika jaringan tetap melakukan proses pelatihan namun error sudah kecil, maka nilai error dapat naik lagi karena jaringan akan keluar dari konvergensi dan menjadi divergen.

Luaran dari sistem juga di plot untuk melihat performa dari model pada dataset training, dan diperoleh hasil sebagai berikut



3.3 Proses Pengujian

Data dari 1506 sampai 3010 kemudian digunakan untuk proses testing. Hal ini dilakukan untuk mengetahui performa dari jaringan yang telah dibentuk pada proses pelatihan. Berikut merupakan program MATLAB untuk proses pengujian:

```
%% TESTING

% Data Input

input_test = [y(2102:3001) y(2101:3000) u(2103:3002) u(2102:3001)
u(2101:3000)];
target_test = [y(2103:3002)];
n_test = length(input_test);
input_test = normalize(input_test);
target_test = normalize(target_test);

%% NN

test_true = 0;
test_false = 0;
for n=1:n_test
    xi_test = input_test(n,:);
    ti_test = target_test(n,:);

    % Komputasi input layer ke hidden layer
    z_inj_test = xi_test*v_ij + v_0j;
    for j=1:n_hidden
        zj_test(1,j) = (1-exp(-z_inj_test(1,j))) / (1+exp(-
z_inj_test(1,j)));
    end
```

```

% Komputasi hidden layer ke output layer
y_in_k_test = zj_test*w_jk + w_0k;
for k=1:n_out
    yk_test(n,k) = y_in_k_test(1,k);
end

% Store error
error_test(1,n) = 0.5*sum((yk_test(n,1) - ti_test).^2);

end

%% Error

avg_error = sum(error_test)/n_test;
disp("Error average test = "+ avg_error *100 +" %");

n_test = length(target_test);
MSE_test = (sum((target_test-yk_test).^2))/n_test;
RMSE_test = sqrt(MSE_test);
disp("MSE testing = "+ MSE_test); disp("RMSE testing = "+
RMSE_test);

%% Plot

figure;
plot(yk_test,'o');
hold on
plot(target_test,'x');
xlim([0 n_test]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN','Output Plant');
hold off

```

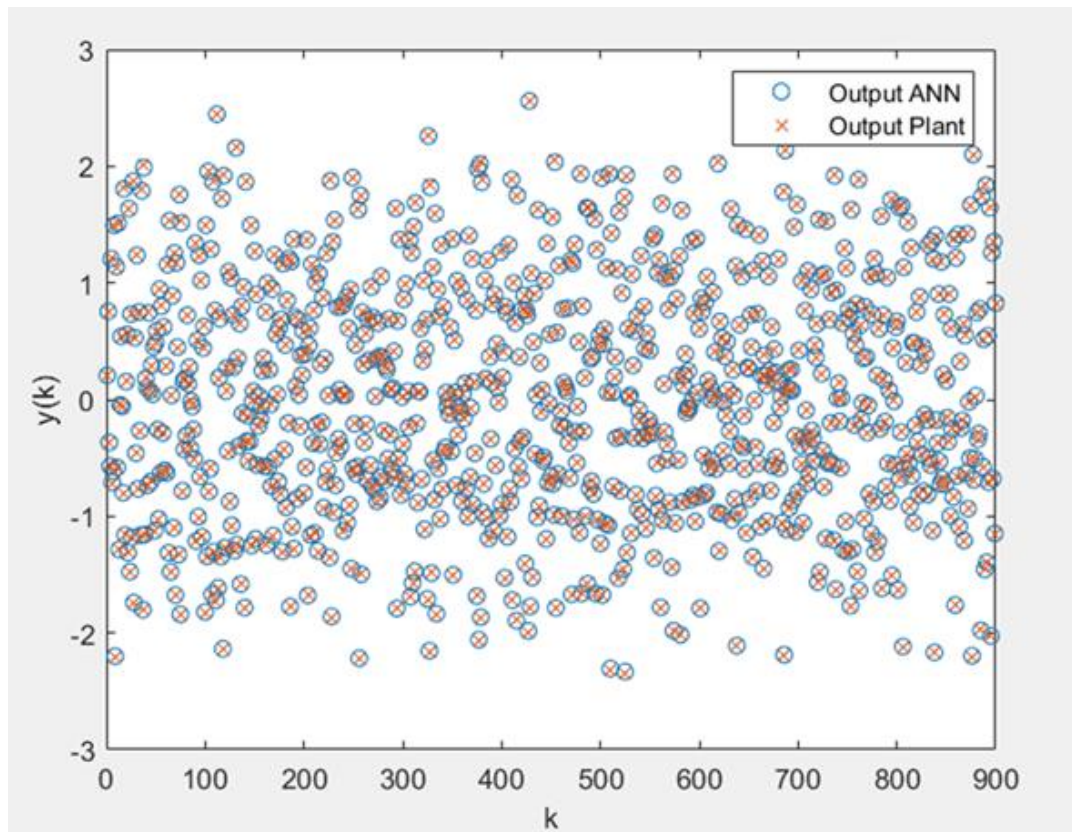
Didapatkan hasil dari proses pengujian sebagai berikut:

```

Error average test = 0.0011536 %
MSE testing = 2.3072e-05
RMSE testing = 0.0048033

```

Penulis juga membuat plot hubungan antara output ANN dengan output plant untuk melihat keluaran dari sistem kita. Berikut merupakan plot output dari ANN dan plant pada input acak.



BAB IV

DESAIN PENGENDALI INVERS MENGGUNAKAN ARTIFICIAL NEURAL NETWORK

5.1 Pembuatan Database

Data dari plant akan di generate dengan menghasilkan sebanyak 3010 angka random. 3010 angka tersebut kemudian kita masukkan kedalam persamaan plant sehingga menghasilkan sebanyak 3010 data output. Pasangan input dan output ini akan menjadi database untuk proses identifikasi sistem.

$$u(k-1) = f(u(k-2), u(k-3), y(k), y(k-1))$$

Berikut merupakan program MATLAB untuk pembuatan database:

```
% Generate Data
rng(42, 'twister');

% Input random
u = 2*rand(3010,1) - 1;

for k = 2:length(u)
    y(k,1) = (1/(1+y(k-1,1)^2)) + 0.25*u(k) - 0.3*u(k-1);
end
```

Program akan menghasilkan 3010 buah data untuk acak biasa pada variable y dan 3010 buah data acak sinusoid pada variable y_{sinusoid} . $u(k)$ merupakan data input $u(k)$ yang didapat melalui proses randomisasi yang telah di normalisasi. Sedangkan $y(k)$ merupakan target $y(k)$ yang didapat melalui fungsi

Data kemudian dinormalisasi menggunakan metode max-min agar mendapatkan nilai random pada range -1 dan +1.

```
% Data Input
input_control = [u(2:2100) u(1:2099) y(4:2102) y(3:2101) y(2:2100)];
target_control = [u(3:2101)];
[input_con_row, input_con_col] = size(input_control);
[target_con_row, target_con_col] = size(target_control);
n_con_data = input_con_row;

% normalisasi
input_control = normalize(input_control);
target_control = normalize(target_control);
```

5.2 Proses Pelatihan

Pada proses learning identifikasi plant, data yang digunakan adalah data yang berasal dari generate data sebelumnya. Masukan dari sistem akan berjumlah 4 yaitu $u(k-2)$, $u(k-3)$, $y(k)$, $y(k-1)$. Data dibagi menjadi 50:50 menjadi 2 bagian yaitu training dan testing.

Inisialisasi bobot menggunakan metode nguyen-widrow dengan program MATLAB sebagai berikut:

```
%% inisialisasi bobot

% Input layer ke hidden layer
beta = 0.7*n_hidden_con^(1/n_in_con);
v_ij_con = rand(n_in_con,n_hidden_con) - 0.5;
for i = 1:n_hidden_con
    norma(i) = sqrt(sum(v_ij_con(:,i).^2));
    v_ij_con(:,i) = (beta*v_ij_con(:,i))/norma(i);
end
v_0j_con = (rand(1,n_hidden_con) - 0.5)*beta*2;

% Hidden layer ke output layer
w_jk_con = rand(n_hidden_con,n_out_con) - 0.5;
w_0k_con = rand(1,n_out_con) - 0.5;
```

Kemudian program akan melakukan proses feedforward dan backpropagation yang akan mengupdate bobot sehingga sistem dapat menghasilkan luaran yang sama dengan persamaan plant yang menghasilkan data pada database.

Program MATLAB untuk feedforward dan backpropagation adalah sebagai berikut:

```
%% Hyperparameter & Variabel
n_in = in_col;           %jumlah unit input layer
n_hidden = 30;           %jumlah unit hidden layer
%% Training

maxepoch_con = 2000;
targeterror_con = 0.00001;

stop = 0;
epoch = 1;
delta_con_wjk_old = 0;
delta_con_w0k_old = 0;
delta_con_vij_old = 0;
delta_con_v0j_old = 0;

while stop == 0 && epoch <= maxepoch_con
    for n=1:n_con_data
        % %
        % Feedforward
        xi = input_control(n,:);
        ti = target_control(n,:);

        % Komputasi input layer ke hidden layer
        z_inj = xi*v_ij_con + v_0j_con;
        for j=1:n_hidden_con
            %zj(1,j) = 2/(1+exp(-z_inj(1,j))) - 1;
            zj(1,j) = (1-exp(-z_inj(1,j))) / (1+exp(-z_inj(1,j)));
        end
    end
end
```

```

% Komputasi hidden layer ke output layer
y_innk = zj*w_jk_con + w_0k_con;
for k=1:n_out_con
    yk(n,k) = y_innk(k);
end

% Store error
error(n,1) = 0.5*sum((yk(n,1) - ti).^2);

% Backpropagation of error %

% Komputasi dari output layer ke hidden layer
d_ok = (yk(n,1) - ti);
delta_wjk = alpha_con*zj'*d_ok + miu_con*delta_con_wjk_old;
delta_w0k = alpha_con*d_ok + miu_con*delta_con_w0k_old;
delta_con_wjk_old = delta_wjk;
delta_con_w0k_old = delta_w0k;

% Komputasi dari hidden layer ke input layer
doinj = d_ok*w_jk_con;
d_oj = doinj.*(1-zj.^2)/0.5;
delta_vij = alpha_con*xi'*d_oj + miu_con*delta_con_vij_old;
delta_v0j = alpha_con*d_oj + miu_con*delta_con_v0j_old;
delta_con_vij_old = delta_vij;
delta_con_v0j_old = delta_v0j;

% Memperbarui bobot dan bias
w_jk_con = w_jk_con - delta_wjk;
w_0k_con = w_0k_con - delta_w0k;
v_ij_con = v_ij_con - delta_vij;
v_0j_con = v_0j_con - delta_v0j;
end
errorpereepoch(1,epoch) = sum(error)/n_con_data;

if errorpereepoch(1,epoch) < targeterror_con
    stop = 1;
end

epoch = epoch+1;
end

%% plot error

epoch = epoch - 1;
figure;
plot(errorpereepoch);
ylabel('Error per epoch'); xlabel('Epoch')
disp("Error per epoch minimum = "+ min(errorpereepoch) *100 +" %");
disp("Error akhir = "+ errorpereepoch(1,epoch)*100+" %");

MSE_train_con = (sum((target_control-yk).^2))/n_con_data;
RMSE_train_con = sqrt(MSE_train_con);
disp("MSE training = "+ MSE_train_con); disp("RMSE training = "+
RMSE_train_con);

%% plot perbandingan output plant dan ANN

```

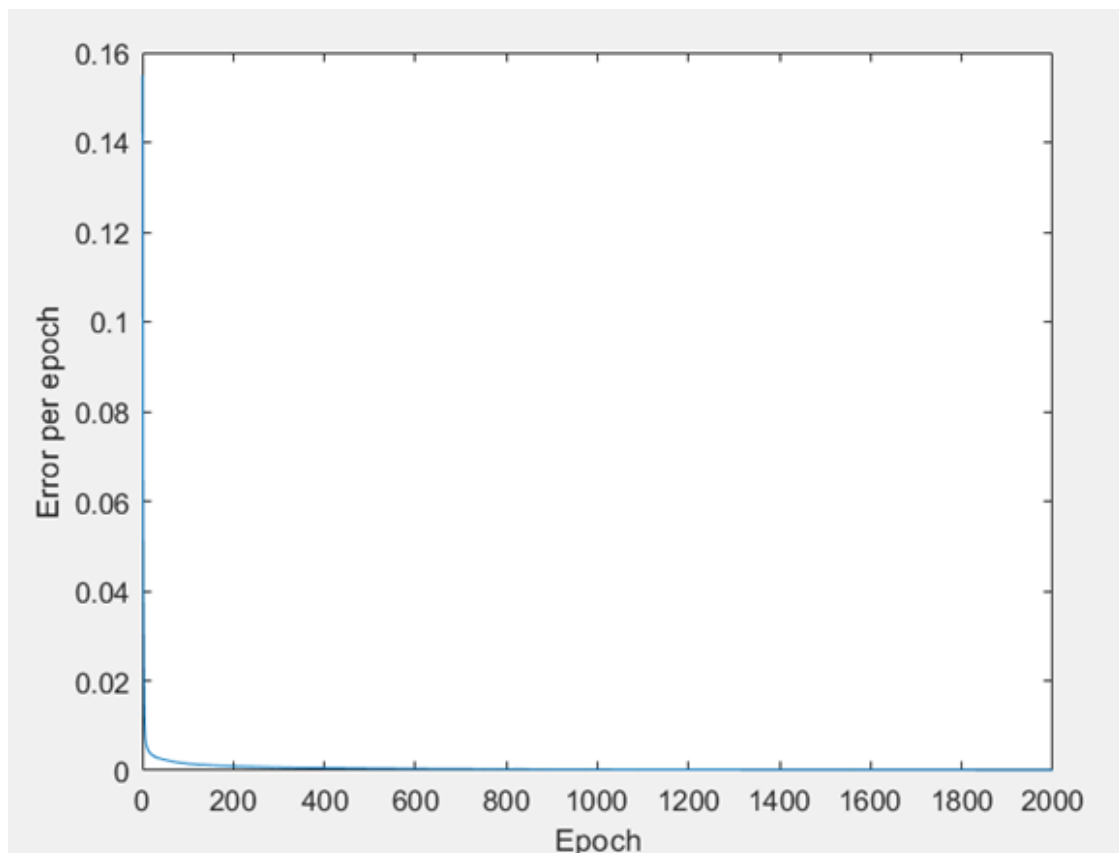


```
figure;
plot(yk, 'o');
hold on
plot(target_control, 'x');
xlim([0 n_con_data]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN', 'Output Plant');
hold off
```

Didapatkan hasil dari proses pelatihan sebagai berikut:

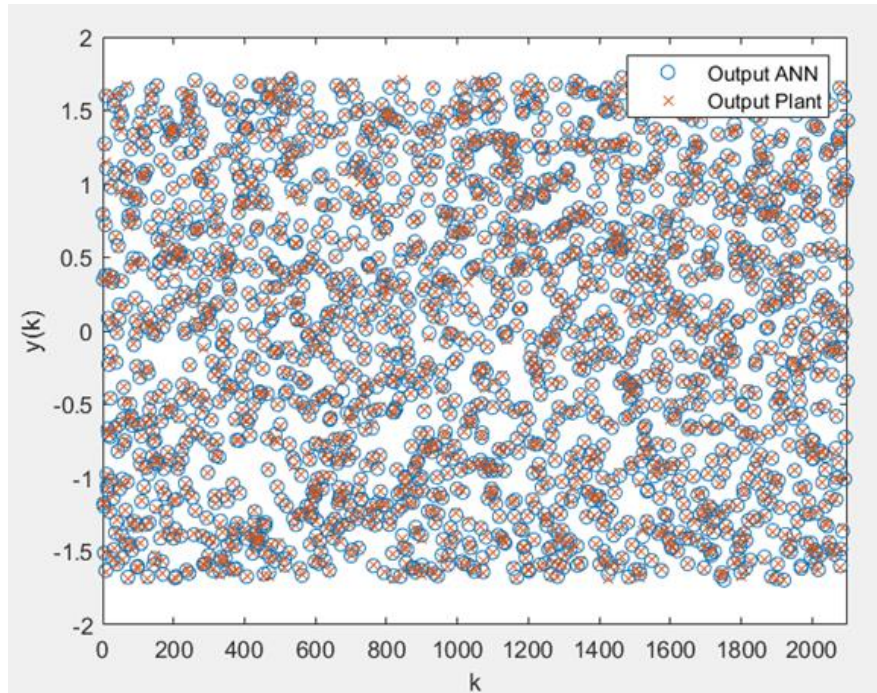
```
Error per epoch minimum = 0.010599 %
Error akhir = 0.010599 %
MSE training = 0.00021198
RMSE training = 0.01456
```

Pada setiap epoch, bobot pada jaringan akan diupdate. Semakin banyak neuron hidden layer yang digunakan, semakin kecil besar epoch yang dibutuhkan untuk mendapatkan nilai output yang sesuai dengan target. Hubungan antara error dan epoch kemudian di plot untuk dilihat penurunan nilai error pada jaringan. Berikut merupakan plot hubungan antara epoch dan error pada input acak.



Dapat dilihat dari grafik bahwa penurunan error sudah tidak mengalami perubahan yang signifikan setelah epoch 100. Sehingga dapat disimpulkan bahwa perubahan bobot lebih dominan terjadi pada epoch dibawah 100. Epoch yang digunakan berjumlah sedikit karena ketika jaringan tetap melakukan proses pelatihan namun error sudah kecil, maka nilai error dapat naik lagi karena jaringan akan keluar dari konvergensi dan menjadi divergen.

Luaran dari sistem juga di plot untuk melihat performa dari model pada dataset training, dan diperoleh hasil sebagai berikut



5.3 Proses Pengujian

Data dari 1506 sampai 3010 kemudian digunakan untuk proses testing. Hal ini dilakukan untuk mengetahui performa dari jaringan yang telah dibentuk pada proses pelatihan. Berikut merupakan program MATLAB untuk proses pengujian:

```
%Data Inserting

input_test_control = [u(2101:3000) u(2100:2999) y(2103:3002)
y(2102:3001) y(2101:3000)];
target_test_control = [u(2102:3001)];
n_test_con = length(input_test_control);
input_test_control = normalize(input_test_control);
target_test_control = normalize(target_test_control);

%Feedforward
test_true = 0;
test_false = 0;
for n=1:n_test_con
    xi_test = input_test_control(n,:);
```

```

ti_test = target_test_control(n,:);

% Komputasi input layer ke hidden layer
z_inj_con_test = xi_test*v_ij_con + v_0j_con;
for j=1:n_hidden_con
    zj_con_test(1,j) = (1-exp(-z_inj_con_test(1,j))) / (1+exp(-
z_inj_con_test(1,j)));
end

% Komputasi hidden layer ke output layer
y_ink_con_test = zj_con_test*w_jk_con + w_0k_con;
for k=1:n_out_con
    yk_con_test(n,k) = y_ink_con_test(1,k);
end

% Store error
error_test_con(1,n) = 0.5*sum((yk_con_test(n,1) - ti_test).^2);

end

avg_error = sum(error_test_con)/n_test_con;
disp("Error average test = "+ avg_error *100 +" %");

%Test MSE & RMSE
n_test_con = length(target_test_control);
MSE_test_con = (sum((target_test_control-
yk_con_test).^2))/n_test_con;
RMSE_test_con = sqrt(MSE_test_con);
disp("MSE testing = "+ MSE_test_con);
disp("RMSE testing = "+ RMSE_test_con);

%Plot actual output and ANN output
figure ();
plot(yk_con_test,'o');
hold on
plot(target_test_control,'x');
xlim([0 n_test_con]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN','Output Plant');
hold off

```

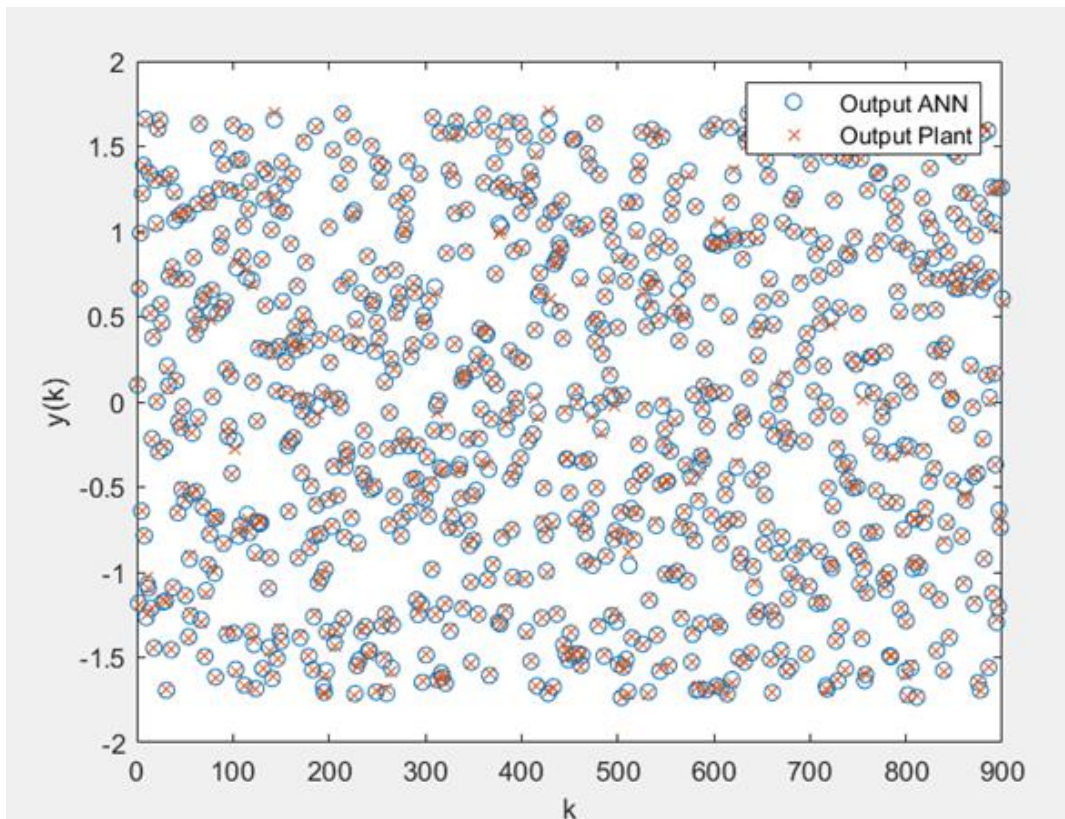
Didapatkan hasil dari proses pengujian sebagai berikut:

```

Error average test = 0.0090533 %
MSE testing = 0.00018107
RMSE testing = 0.013456

```

Penulis juga membuat plot hubungan antara output ANN dengan output plant untuk melihat keluaran dari sistem kita. Berikut merupakan plot output dari ANN dan plant pada input acak.



Hasil yang didapatkan sudah cukup baik dimana nyaris Sebagian besar dari lingkaran biru dan oranye sudah tumpang tindih walaupun masih terdapat beberapa pasangan data yang masih belum menghasilkan hasil yang sama

BAB V

PENERAPAN DIRECT INVERSE CONTROL MENGUNAKAN ARTIFICIAL NEURAL NETWORK

5.1 Kendali Metode Direct Inverse Control

Setelah telah berhasil membuat pengendali dan plant yang ingin dikendalikan secara baik. Hal ini dilakukan dengan menggunakan bobot bobot yang telah didapatkan dari proses pelatihan pengendali dan identifikasi sistem dan Menyusun program forward pass dimulai dari pengendali kemudian masuk ke plant yang telah diidentifikasi untuk menghasilkan luaran dari plant. Pada dasarnya, DIC dimulai dengan memberikan output reference kepada model inversi. Selanjutnya model inversi akan menghasilkan input yang sesuai dengan output tersebut untuk diberikan kepada hasil sistem identifikasi. Seelanjutnya sistem identifikasi akan memproses input tersebut untuk menghasilkan output dari plant. Output actual inilah yang akan dibandingkan dengan output reference apakah sesuai atau tidak

Program Direct Inverse Control dilakukan dengan menggunakan kode MATLAB sebagai berikut;

```
norm_ref = y(1:3010); %r(k) = y(k)
norm_ref = normalize(norm_ref);

[n_in_row,n_in_col] = size(norm_ref);
[n_out_row,n_out_col] = size(norm_ref);

for n = 1:n_in_row
    %-----INV CONTROL-----
    if n==1
        in = [0 0 norm_ref(n,1) 0 0];
    elseif n==2
        in = [yk_con_inv(n-1,:) 0 norm_ref(n,1) yk_id_inv(n-1,:) 0];
    else
        in = [yk_con_inv(n-1,:) yk_con_inv(n-2,:) norm_ref(n,1)
            yk_id_inv(n-1,:) yk_id_inv(n-2,:)];
    end

    in = normalize(in);

    target_inv = norm_ref(n,:);

    %Input to Hidden
    z_inj_con_inv = in*v_ij_con + v_0j_con;
    for j=1:n_hidden_con
        %zj_test(1,j) = 2/(1+exp(-z_inj_test(1,j))) - 1;
        zj_con_inv(1,j) = (1-exp(-z_inj_con_inv(1,j))) / (1+exp(-
            z_inj_con_inv(1,j)));
    end

    %Hidden to Output
```

```

y_ink_con_inv = zj_con_inv*w_jk_con + w_0k_con;
for k=1:n_out_con
    %yk_test(n,k) = 2/(1+exp(-y_ink_test(1,k))) - 1;
    %yk_test(n,k) = (1-exp(-y_ink_test(1,k))) / (1+exp(-
y_ink_test(1,k)));
    yk_con_inv(n,k) = y_ink_con_inv(1,k);
end

%-----SYSTEM ID-----

if n==1
    %in_id = [yk_con_inv(n,:) 0 0 0 0];
    in_id = [0 0 yk_con_inv(n,:) 0 0];
elseif n==2
    %in_id = [yk_con_inv(n,:) yk_con_inv(n-1,:) 0 yk_id_inv(n-
1,:) 0];
    in_id = [yk_id_inv(n-1,:) 0 yk_con_inv(n,:) yk_con_inv(n-
1,:) 0];
else
    %in_id = [yk_con_inv(n,:) yk_con_inv(n-1,:) yk_con_inv(n-
2,:) yk_id_inv(n-1,:) yk_id_inv(n-2,:)];
    in_id = [yk_id_inv(n-1,:) yk_id_inv(n-2,:) yk_con_inv(n,:)
yk_con_inv(n-1,:) yk_con_inv(n-2,:)];
end

in_id = normalize(in_id);

z_inj_id_inv = in_id*v_ij + v_0j;
for j=1:n_hidden
    %zj_test(1,j) = 2/(1+exp(-z_inj_test(1,j))) - 1;
    zj_id_inv(1,j) = (1-exp(-z_inj_id_inv(1,j))) / (1+exp(-
z_inj_id_inv(1,j)));
end

y_ink_id_inv = zj_id_inv*w_jk + w_0k;
for k=1:n_out
    %yk_test(n,k) = 2/(1+exp(-y_ink_test(1,k))) - 1;
    %yk_test(n,k) = (1-exp(-y_ink_test(1,k))) / (1+exp(-
y_ink_test(1,k)));
    yk_id_inv(n,k) = y_ink_id_inv(1,k);
end

error_inv(1,n) = 0.5*sum((target_inv - yk_id_inv(n,:)).^2);
%-----

end

MSE_DIC = sum(error_inv)/n_in_row

figure ();
plot(yk_id_inv,'o');
hold on
plot(norm_ref,'x');
xlabel('k'); ylabel('y(k)'); legend('Output ANN','Reference');
hold off

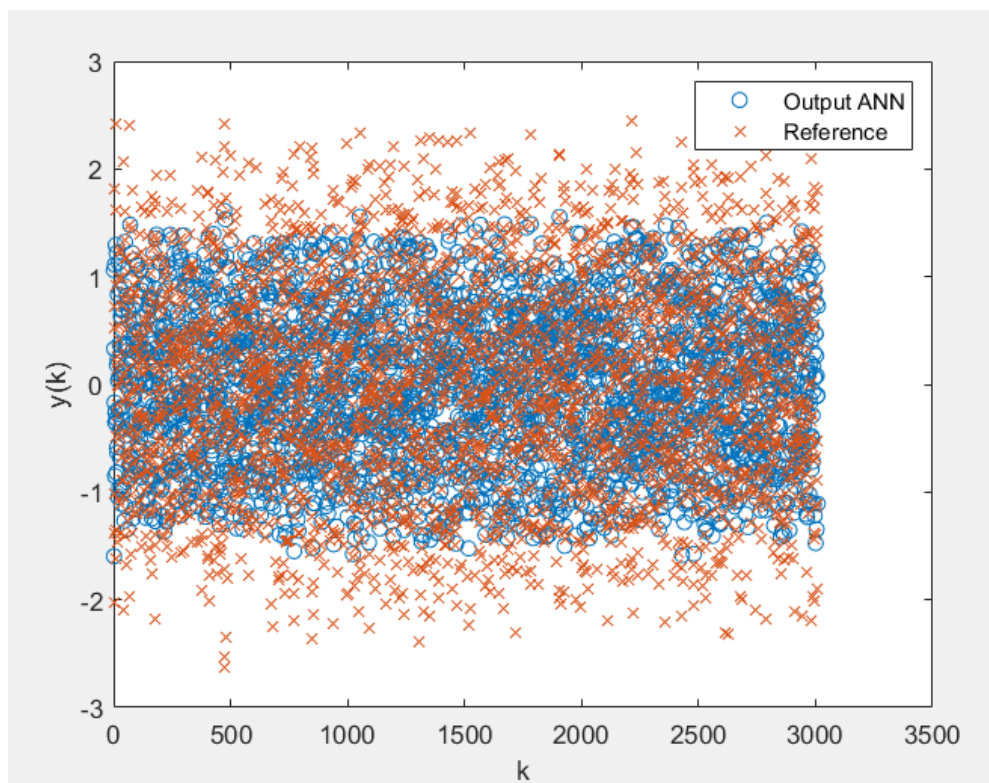
```


Diperoleh hasil dari DIC sebagai berikut:

```
MSE_DIC =  
  
1.1204
```

Terlihat bahwa error yang didapat tidak sedikit. Namun, hasil ini adalah hasil terbaik yang dapat diperoleh setelah beberapa upaya dengan mengubah penundaan input dan output. Nilai MSE dari DIC yang tidak terlalu besar dapat disebabkan oleh delay yang tidak tepat atau arsitektur yang tidak tepat.

Penulis juga membuat plot hubungan antara output ANN dengan output plant untuk melihat luaran dari sistem kita. Berikut merupakan plot output dari sistem DIC.



Seperti yang Anda lihat, ada beberapa lingkaran oranye dan biru yang berdekatan, hampir tumpang tindih. Hal ini menunjukkan bahwa sebagian data keluaran yang dihasilkan juga merupakan keluaran yang diinginkan. Namun masih terdapat pasangan data yang masih belum bisa menyamai dari reference yang diberikan.

BAB VI

KESIMPULAN

Setelah melakukan eksperimen identifikasi sistem dengan input acak dan input sinusoidal, diperoleh kesimpulan sebagai berikut:

1. *Backpropagation* merupakan metode ANN yang merupakan metode pembelajaran sistem yang mengubah nilai bobot dari neuron relatif terhadap error yang diperoleh
2. ANN dapat digunakan untuk mengidentifikasi sistem dan pengendali pada sistem kendali
3. Nilai masukan yang lebih berpola teratur menghasilkan jaringan yang lebih baik dibandingkan nilai masukan yang acak
4. Jumlah neuron hidden layer berpengaruh terhadap hasil luaran ANN dimana semakin banyak jumlah neuron akan memberikan hasil yang lebih baik namun perlu diwaspadai pengaruh dari jumlah neuron yang terlalu banyak karena dapat menyebabkan overfitting
5. Sistem identifikasi dapat dilakukan menggunakan artificial neural network
6. Desain kendali inversi dapat dilakukan menggunakan artificial neural network
7. Penggabungan sistem identifikasi dan kendali inversi akan membentuk metode direct inverse control yang akan memperbolehkan kita mengendalikan sistem non linear

DAFTAR REFERENSI

- [1] D. Wahyuningsih, I. Zuhroh, and - Zainuri, "Prediksi Inflasi Indonesia Dengan Model Artificial Neural Network," *J. Indones. Appl. Econ.*, vol. 2, no. 2, pp. 2–2008, 2008, doi: 10.21776/ub.jiae.2008.002.02.7.
- [2] D. B. Kusumoputro, "BACKPROPAGATION," 2021.
- [3] "Artificial_Neural_Network @ Www.Saedsayad.Com." [Online]. Available: http://www.saedsayad.com/artificial_neural_network.htm.
- [4] P. D. B. Kusumoputro and U. Indonesia, "Penggunaan Artificial Neural Networks dalam Sistem Kendali."

LAMPIRAN

Kode Sumber Simulasi Direct Inverse Control

```
clear;
clc;
close all;

y(1) = 0;

%% Generate Data
rng(42, 'twister');

% Input random
u = 2*rand(3010,1) - 1;

for k = 2:length(u)
    y(k,1) = (1/(1+y(k-1,1)^2)) + 0.25*u(k) - 0.3*u(k-1);
end

%% Data Input
in = [y(2:2101) y(1:2100) u(3:2102) u(2:2101) u(1:2100)];
target = [y(3:2102)];
[in_row,in_col] = size(in);
[target_row,target_col] = size(target);
n_data = in_row;

%% Normalisasi
in = normalize(in);

target = normalize(target);

%% Hyperparameter & Variabel
n_in = in_col;           %jumlah unit input layer
n_hidden = 30;           %jumlah unit hidden layer
n_out = target_col;      %jumlah unit output layer

alpha = 0.001;
miu = 0.9;

rng(25, 'twister')

%% Inisialisasi

% Input layer ke hidden layer
beta = 0.7*n_hidden^(1/n_in);
v_ij = rand(n_in,n_hidden) - 0.5;
for i = 1:n_hidden
    norma(i) = sqrt(sum(v_ij(:,i).^2));
    v_ij(:,i) = (beta*v_ij(:,i))/norma(i);
end
v_0j = (rand(1,n_hidden) - 0.5)*beta*2;

% Hidden layer ke output layer
w_jk = rand(n_hidden,n_out) - 0.5;
w_0k = rand(1,n_out) - 0.5;
```

```

%% TRAINING

maxepoch = 20000;
targeterror = 0.00001;

stop = 0;
epoch = 1;
delta_wjk_old = 0;
delta_w0k_old = 0;
delta_vij_old = 0;
delta_v0j_old = 0;

%% NN

while stop == 0 && epoch <= maxepoch
    for n=1:n_data
        % Feedforward
        xi = in(n,:);
        ti = target(n,:);

        % Komputasi input layer ke hidden layer
        z_inj = xi*v_ij + v_0j;
        for j=1:n_hidden
            zj(1,j) = (1-exp(-z_inj(1,j))) / (1+exp(-z_inj(1,j)));
        end

        % Komputasi hidden layer ke output layer
        y_ink = zj*w_jk + w_0k;
        for k=1:n_out
            yk(n,k) = y_ink(k);
        end

        % Store error
        error(n,1) = 0.5*sum((yk(n,1) - ti).^2);

        % Backpropagation of error %

        % Komputasi dari output layer ke hidden layer
        d_ok = (yk(n,1) - ti);
        delta_wjk = alpha*zj'*d_ok + miu*delta_wjk_old;
        delta_w0k = alpha*d_ok + miu*delta_w0k_old;
        delta_wjk_old = delta_wjk;
        delta_w0k_old = delta_w0k;

        % Komputasi dari hidden layer ke input layer
        doinj = d_ok*w_jk';
        d_oj = doinj.*(1-zj.^2)/0.5;
        delta_vij = alpha*xi'*d_oj + miu*delta_vij_old;
        delta_v0j = alpha*d_oj + miu*delta_v0j_old;
        delta_vij_old = delta_vij;
        delta_v0j_old = delta_v0j;

        % Memperbarui bobot dan bias
        w_jk = w_jk - delta_wjk;
        w_0k = w_0k - delta_w0k;
        v_ij = v_ij - delta_vij;
    end
    epoch = epoch + 1;
end

```

```

        v_0j = v_0j - delta_v0j;
    end
    errorpereepoch(1,epoch) = sum(error)/n_data;

    if errorpereepoch(1,epoch) < targeterror
        stop = 1;
    end

    epoch = epoch+1;
end

%% Plot

epoch = epoch - 1;
figure;
plot(errorpereepoch);
ylabel('Error per epoch'); xlabel('Epoch')
disp("Error per epoch minimum = "+ min(errorpereepoch) *100 +" %");
disp("Error akhir = "+ errorpereepoch(1,epoch)*100+" %");

%% Plot MSE & RMSE
MSE_train = (sum((target-yk).^2))/n_data;
RMSE_train = sqrt(MSE_train);
disp("MSE training = "+ MSE_train); disp("RMSE training = "+
RMSE_train);

figure;
plot(yk,'o');
hold on
plot(target,'x');
xlim([0 n_data]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN','Output Plant');
hold off

%% TESTING

% Data Input

input_test = [y(2102:3001) y(2101:3000) u(2103:3002) u(2102:3001)
u(2101:3000)];
target_test = [y(2103:3002)];
n_test = length(input_test);
input_test = normalize(input_test);
target_test = normalize(target_test);

%% NN

test_true = 0;
test_false = 0;
for n=1:n_test
    xi_test = input_test(n,:);
    ti_test = target_test(n,:);

    % Komputasi input layer ke hidden layer
    z_inj_test = xi_test*v_ij + v_0j;
    for j=1:n_hidden

```

```

        zj_test(1,j) = (1-exp(-z_inj_test(1,j))) / (1+exp(-
z_inj_test(1,j)));
    end

    % Komputasi hidden layer ke output layer
    y_ink_test = zj_test*w_jk + w_0k;
    for k=1:n_out
        yk_test(n,k) = y_ink_test(1,k);
    end

    % Store error
    error_test(1,n) = 0.5*sum((yk_test(n,1) - ti_test).^2);

end

%% Error

avg_error = sum(error_test)/n_test;
disp("Error average test = "+ avg_error *100 +" %");

n_test = length(target_test);
MSE_test = (sum((target_test-yk_test).^2))/n_test;
RMSE_test = sqrt(MSE_test);
disp("MSE testing = "+ MSE_test); disp("RMSE testing = "+
RMSE_test);

%% Plot

figure;
plot(yk_test,'o');
hold on
plot(target_test,'x');
xlim([0 n_test]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN','Output Plant');
hold off

clearvars -except v_0j v_ij w_0k w_jk u y n_hidden n_out

input_control = [u(2:2100) u(1:2099) y(4:2102) y(3:2101) y(2:2100)];
target_control = [u(3:2101)];
[input_con_row,input_con_col] = size(input_control);
[target_con_row,target_con_col] = size(target_control);
n_con_data = input_con_row;

%% normalisasi
input_control = normalize(input_control);

target_control = normalize(target_control);

%% parameter
n_in_con = input_con_col;           %jumlah unit input layer
n_hidden_con = 30;                 %jumlah unit hidden layer
n_out_con = target_con_col;        %jumlah unit output layer

alpha_con = 0.001;
miu_con = 0.3;

```

```

rng(30, 'twister');

%% inisialisasi bobot

% Input layer ke hidden layer
beta = 0.7*n_hidden_con^(1/n_in_con);
v_ij_con = rand(n_in_con,n_hidden_con) - 0.5;
for i = 1:n_hidden_con
    norma(i) = sqrt(sum(v_ij_con(:,i).^2));
    v_ij_con(:,i) = (beta*v_ij_con(:,i))/norma(i);
end
v_0j_con = (rand(1,n_hidden_con) - 0.5)*beta*2;

% Hidden layer ke output layer
w_jk_con = rand(n_hidden_con,n_out_con) - 0.5;
w_0k_con = rand(1,n_out_con) - 0.5;

%% Training

maxepoch_con = 2000;
targeterror_con = 0.00001;

stop = 0;
epoch = 1;
delta_con_wjk_old = 0;
delta_con_w0k_old = 0;
delta_con_vij_old = 0;
delta_con_v0j_old = 0;

while stop == 0 && epoch <= maxepoch_con
    for n=1:n_con_data
        % %
        % Feedforward
        xi = input_control(n,:);
        ti = target_control(n,:);

        % Komputasi input layer ke hidden layer
        z_inj = xi*v_ij_con + v_0j_con;
        for j=1:n_hidden_con
            %zj(1,j) = 2/(1+exp(-z_inj(1,j))) - 1;
            zj(1,j) = (1-exp(-z_inj(1,j))) / (1+exp(-z_inj(1,j)));
        end

        % Komputasi hidden layer ke output layer
        y_ink = zj*w_jk_con + w_0k_con;
        for k=1:n_out_con
            yk(n,k) = y_ink(k);
        end

        % Store error
        error(n,1) = 0.5*sum((yk(n,1) - ti).^2);

        % Backpropagation of error %

        % Komputasi dari output layer ke hidden layer
        d_ok = (yk(n,1) - ti);
        delta_wjk = alpha_con*zj'*d_ok + miu_con*delta_con_wjk_old;
    end
    epoch = epoch + 1;
end

```

```

delta_w0k = alpha_con*d_ok + miu_con*delta_con_w0k_old;
delta_con_wjk_old = delta_wjk;
delta_con_w0k_old = delta_w0k;

% Komputasi dari hidden layer ke input layer
doinj = d_ok*w_jk_con';
d_oj = doinj.*(1-zj.^2)/0.5;
delta_vij = alpha_con*xi'*d_oj + miu_con*delta_con_vij_old;
delta_v0j = alpha_con*d_oj + miu_con*delta_con_v0j_old;
delta_con_vij_old = delta_vij;
delta_con_v0j_old = delta_v0j;

% Memperbarui bobot dan bias
w_jk_con = w_jk_con - delta_wjk;
w_0k_con = w_0k_con - delta_w0k;
v_ij_con = v_ij_con - delta_vij;
v_0j_con = v_0j_con - delta_v0j;
end
errorpereepoch(1,epoch) = sum(error)/n_con_data;

if errorpereepoch(1,epoch) < targeterror_con
    stop = 1;
end

epoch = epoch+1;
end

%% plot error

epoch = epoch - 1;
figure;
plot(errorpereepoch);
ylabel('Error per epoch'); xlabel('Epoch')
disp("Error per epoch minimum = "+ min(errorpereepoch) *100 +" %");
disp("Error akhir = "+ errorpereepoch(1,epoch)*100+" %");

MSE_train_con = (sum((target_control-yk).^2))/n_con_data;
RMSE_train_con = sqrt(MSE_train_con);
disp("MSE training = "+ MSE_train_con); disp("RMSE training = "+
RMSE_train_con);

%% plot perbandingan output plant dan ANN

figure;
plot(yk, 'o');
hold on
plot(target_control, 'x');
xlim([0 n_con_data]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN', 'Output Plant');
hold off

%Data Inserting

input_test_control = [u(2101:3000) u(2100:2999) y(2103:3002)
y(2102:3001) y(2101:3000)];
target_test_control = [u(2102:3001)];
n_test_con = length(input_test_control);

```



```

input_test_control = normalize(input_test_control);
target_test_control = normalize(target_test_control);

%Feedforward
test_true = 0;
test_false = 0;
for n=1:n_test_con
    xi_test = input_test_control(n,:);
    ti_test = target_test_control(n,:);

    % Komputasi input layer ke hidden layer
    z_inj_con_test = xi_test*v_ij_con + v_0j_con;
    for j=1:n_hidden_con
        zj_con_test(1,j) = (1-exp(-z_inj_con_test(1,j))) / (1+exp(-
z_inj_con_test(1,j)));
    end

    % Komputasi hidden layer ke output layer
    y_ink_con_test = zj_con_test*w_jk_con + w_0k_con;
    for k=1:n_out_con
        yk_con_test(n,k) = y_ink_con_test(1,k);
    end

    % Store error
    error_test_con(1,n) = 0.5*sum((yk_con_test(n,1) - ti_test).^2);

end

avg_error = sum(error_test_con)/n_test_con;
disp("Error average test = "+ avg_error *100 +" %");

%Test MSE & RMSE
n_test_con = length(target_test_control);
MSE_test_con = (sum((target_test_control-
yk_con_test).^2))/n_test_con;
RMSE_test_con = sqrt(MSE_test_con);
disp("MSE testing = "+ MSE_test_con);
disp("RMSE testing = "+ RMSE_test_con);

%Plot actual output and ANN output
figure ();
plot(yk_con_test, 'o');
hold on
plot(target_test_control, 'x');
xlim([0 n_test_con]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN', 'Output Plant');
hold off

norm_ref = y(1:3010); %r(k) = y(k)
norm_ref = normalize(norm_ref);

[n_in_row,n_in_col] = size(norm_ref);
[n_out_row,n_out_col] = size(norm_ref);

for n = 1:n_in_row
    %-----INV CONTROL-----

```

```

if n==1
    in = [0 0 norm_ref(n,1) 0 0];
elseif n==2
    in = [yk_con_inv(n-1,:) 0 norm_ref(n,1) yk_id_inv(n-1,:) 0];
else
    in = [yk_con_inv(n-1,:) yk_con_inv(n-2,:) norm_ref(n,1)
yk_id_inv(n-1,:) yk_id_inv(n-2,:)];
end

in = normalize(in);

target_inv = norm_ref(n,:);

%Input to Hidden
z_inj_con_inv = in*v_ij_con + v_0j_con;
for j=1:n_hidden_con
    %zj_test(1,j) = 2/(1+exp(-z_inj_test(1,j))) - 1;
    zj_con_inv(1,j) = (1-exp(-z_inj_con_inv(1,j))) / (1+exp(-
z_inj_con_inv(1,j)));
end

%Hidden to Output
y_ink_con_inv = zj_con_inv*w_jk_con + w_0k_con;
for k=1:n_out_con
    %yk_test(n,k) = 2/(1+exp(-y_ink_test(1,k))) - 1;
    %yk_test(n,k) = (1-exp(-y_ink_test(1,k))) / (1+exp(-
y_ink_test(1,k)));
    yk_con_inv(n,k) = y_ink_con_inv(1,k);
end

%-----SYSTEM ID-----

if n==1
    %in_id = [yk_con_inv(n,:) 0 0 0 0];
    in_id = [0 0 yk_con_inv(n,:) 0 0];
elseif n==2
    %in_id = [yk_con_inv(n,:) yk_con_inv(n-1,:) 0 yk_id_inv(n-
1,:) 0];
    in_id = [yk_id_inv(n-1,:) 0 yk_con_inv(n,:) yk_con_inv(n-
1,:) 0];
else
    %in_id = [yk_con_inv(n,:) yk_con_inv(n-1,:) yk_con_inv(n-
2,:) yk_id_inv(n-1,:) yk_id_inv(n-2,:)];
    in_id = [yk_id_inv(n-1,:) yk_id_inv(n-2,:) yk_con_inv(n,:)
yk_con_inv(n-1,:) yk_con_inv(n-2,:)];
end

in_id = normalize(in_id);

z_inj_id_inv = in_id*v_ij + v_0j;
for j=1:n_hidden
    %zj_test(1,j) = 2/(1+exp(-z_inj_test(1,j))) - 1;
    zj_id_inv(1,j) = (1-exp(-z_inj_id_inv(1,j))) / (1+exp(-
z_inj_id_inv(1,j)));
end

```

```

y_ink_id_inv = zj_id_inv*w_jk + w_0k;
for k=1:n_out
    %yk_test(n,k) = 2/(1+exp(-y_ink_test(1,k))) - 1;
    %yk_test(n,k) = (1-exp(-y_ink_test(1,k))) / (1+exp(-
y_ink_test(1,k)));
    yk_id_inv(n,k) = y_ink_id_inv(1,k);
end

error_inv(1,n) = 0.5*sum((target_inv - yk_id_inv(n,:)).^2);
%-----

end

MSE_DIC = sum(error_inv)/n_in_row

figure ();
plot(yk_id_inv, 'o');
hold on
plot(norm_ref, 'x');
xlabel('k'); ylabel('y(k)'); legend('Output ANN', 'Reference');
hold off

```

Lampiran 1.

Berikut merupakan kode lengkap MATLAB untuk eksperimen berikut:

```
clc;
clear;
close all;

%% Random Input
rng(0,'twister');
u = (1-(-1)).*rand(3010,1)-1;

y=0;
for i = 2:length(u)
    if i == 1 then
        y(i)=0
    end
    y(i)=(1/(1+y(i-1)^2)) + 0.25*u(i) - 0.300*u(i-1);
end

%% Sinusoidal Input
u_sinusoid = sind(0:(length(u)-1));

y_sinusoid = 0;
for i = 2:length(u_sinusoid)
    if i == 1 then
        y_sinusoid(i)=0
    end
    y_sinusoid(i)=(1/(1+y_sinusoid(i-1)^2))+ 0.250*u_sinusoid(i) -
    0.300*u_sinusoid(i-1);
end

%% Data Splitting

input = [y(2:1505)' y(1:1504)' u(3:1506) u(2:1505) u(1:1504)];
target = y(3:1506)';
[input_row,input_col] = size(input);
[target_row,target_col] = size(target);
n_train = size(input);
n_train = n_train(2);

%% Normalisasi
input = normalize(input)

%% Hyperparameter & Declare Variable

%Hyperparameter
x_size = input_col; %Input layer
z_size = 7; %Hidden layer
y_size = target_col; %Output layer

alpha = 0.1;
epoch = 400;
miu = 0.3;

%Variable
```

```

stop = 0;
error_target = 0.00001;
data_count = input_row;
epoch_count = 1;

delta_zy_old = 0;
delta_zy_bias_old = 0;
delta_xz_old = 0;
delta_xz_bias_old = 0;
%% Initalisasi

%Random Init
rng(200) %Seed
epsilon_init = 0.5; %Range random number

beta = 0.7 * z_size^(1/x_size);
weight_xz = rand(x_size, z_size) - epsilon_init;
weight_zy = rand(z_size, y_size) - epsilon_init;
for i = 1:z_size
    norma(i) = sqrt(sum(weight_xz(:,i).^2));
    weight_xz(:,i) = beta*(weight_xz(:,i))/norma(i));
end

bias_xz = (rand(1, z_size) - epsilon_init) * beta;
bias_zy = rand(1, y_size) - epsilon_init;

%% Backpropagation
while stop == 0 && epoch_count <= epoch
    for n = 1:data_count
        %Forward Pass
        %Input -> Hidden
        xi = input(n,:);
        ti = target(n,:);

        z_in = bias_xz + xi*weight_xz;

        for m=1:z_size
            z(1,m) = 1/(1+exp(-z_in(1,m)));
        end

        %Hidden -> Output
        y_in = bias_zy + z*weight_zy;

        for l=1:y_size
            yk(n,l) = 1/(1+exp(-y_in(1,l)));
        end

        %Backward Pass

        %Momentum calculation
        momentum_zy = miu*delta_zy_old;
        momentum_xz = miu*delta_xz_old;

        %Output->Hidden
        for l=1:y_size
            do_k = (yk(n,l) - ti).*(1+yk(n,l)).*(1-yk(n,l)).*0.5;
        end
    end
end

```

```

delta_zy = alpha.*z'.*do_k + momentum_zy;
delta_zy_old = delta_zy;
delta_zy_bias = alpha.*do_k + miu*delta_zy_bias_old;
delta_zy_bias_old = delta_zy_bias;

%Hidden->Input
sigma_j = do_k * weight_zy';
do_j = sigma_j.* z.*(1-z).*0.5;

delta_xz = alpha.* xi' .* do_j + momentum_xz;
delta_xz_old = delta_xz;
delta_xz_bias = alpha .* do_j + miu*delta_xz_bias_old;
delta_xz_bias_old = delta_xz_bias;

%Weight Update
weight_zy = weight_zy - delta_zy;
weight_xz = weight_xz - delta_xz;
bias_zy = bias_zy - delta_zy_bias;
bias_xz = bias_xz - delta_xz_bias;

%Cost function
error(n,1) = 0.5*sum((yk(n,1)-ti).^2);

%Momentum update
delta_zy_old = delta_zy;
delta_xz_old = delta_xz;
end
error_per_epoch(1,epoch_count) = sum(error)/input_row;
if error_per_epoch(1,epoch_count) < error_target
    stop = 1;
end

epoch_count = epoch_count+1;

end
avgererrortrain = sum(error)/n_train;
plot(error_per_epoch);
%% ANN Input Acak (testing)

% Data Input
input_testing = [y(1506:3009)' y(1505:3008)' u(1507:3010)
u(1506:3009) u(1505:3008)];
target_testing = y(1507:3010)';
[input_row_test,input_col_test] = size(input);
[target_row_test,target_col_test] = size(target);
n_test = size(input_testing);
n_test = n_test(2);

3:1506 %y(k)
2:1505 %y(k-1)
1:1504 %y(k-2)

1507:3010 %y(k)
1506:3009 %y(k-1)
1505:3008 %y(k-2)

```

```

%% Normalisasi
input = normalize(input)
%% Hyperparameter & Declare Variable

%Variable
data_count_test = input_row_test;
true_count = 0;

for n = 1:data_count_test
    %Forward Pass
    %Input -> Hidden
    x_test = input_testing(n,:);
    t_test = target_testing(n,:);

    z_in_test = bias_xz + x_test*weight_xz;

    for m=1:z_size
        z_test(1,m) = 1/(1+exp(-z_in_test(1,m)));
    end

    %Hidden -> Output
    y_in_test = bias_zy + z_test*weight_zy;

    for l=1:y_size
        y_test(n,l) = 1/(1+exp(-y_in_test(1,l)));
    end

    error_test(1,n) = 0.5*sum((y_test(n,1) - t_test).^2);

end

avgerrorestest = sum(error_test)/n_test;
MSE_test = (sum((t_test-y_test).^2))/n_test;
disp("Error average test = "+ avgerrorestest *100 +" %");
disp("MSE testing = "+ MSE_test*100 +" %");

figure;
plot(y_test,'o');
hold on
plot(target_testing,'x');
xlim([0 n_test]); xlabel('k'); ylabel('y(k)'); legend('Output
ANN','Output Plant');
hold off

```