Algoritma Klasifikasi *Dataset Iris* dengan *Backpropagation Neural Networks* (BPNN) pada Matlab

Azzam Muhammadi Rizqun Karima
Universitas Indonesia
azzam.muhammadi@ui.ac.id

ABSTRAK

Penelitian ini menerapkan algoritma Backpropagation Neural Networks (BPNN) untuk klasifikasi dataset Iris menggunakan Matlab. Tujuan utama adalah mengklasifikasikan tiga spesies bunga Iris berdasarkan fitur-fitur seperti panjang dan lebar sepal serta petal. Hasil eksperimen menunjukkan akurasi pelatihan rata-rata 97.8% dengan standar deviasi 0.22222% dan akurasi pengujian mendekati 93%. Analisis epoch pelatihan menunjukkan rata-rata sebesar 2138.84 dengan standar deviasi 497.7648. Kesalahan klasifikasi yang signifikan terjadi ketika objek dengan true class 3 diprediksi sebagai kelas 2. Meskipun BPNN menunjukkan efektivitas tinggi untuk dataset ini, analisis menekankan pentingnya pra-pemrosesan dan mempertimbangkan kompleksitas model. Kesimpulan menunjukkan potensi BPNN untuk tugas klasifikasi lain dengan pertimbangan tertentu.

Kata Kunci : *Backpropagation Neural Networks*, Dataset Iris, Klasifikasi, Matlab, Pembelajaran Mesin.

1. Pendahuluan

Jaringan Saraf Tiruan (JST) atau *Artificial Neural Networks* (ANN) telah menjadi salah satu metode paling menonjol dalam bidang kecerdasan buatan dan pembelajaran mesin. Kemampuannya untuk memodelkan dan mengklasifikasikan data kompleks telah menghasilkan aplikasi yang meluas dalam berbagai bidang, mulai dari pengenalan gambar, analisis teks, hingga prediksi finansial. Dalam konteks ini, algoritma *Backpropagation Neural Networks* (BPNN) menonjol sebagai salah satu pendekatan dasar yang paling sering digunakan dalam JST. Algoritma ini menawarkan kombinasi antara efisiensi komputasi dan akurasi klasifikasi, menjadikannya pilihan yang ideal untuk mereka yang baru memulai dalam bidang *Neural Networks*.

Dataset Iris, yang diperkenalkan oleh Ronald A. Fisher pada tahun 1936, telah lama menjadi benchmark dalam literatur pengenalan pola. Dataset ini menawarkan tiga kelas dari spesies bunga iris dengan empat fitur, menjadikannya dataset yang sederhana namun informatif untuk eksperimen awal dalam pembelajaran mesin. Menggunakan dataset ini sebagai titik awal, penelitian ini bertujuan untuk menerapkan algoritma BPNN pada Matlab, memberikan pemahaman mendalam tentang bagaimana algoritma bekerja dan bagaimana mengoptimalkannya untuk hasil yang lebih baik.

Alasan utama memilih BPNN sebagai titik awal dalam eksplorasi Neural Networks adalah karena sifatnya yang intuitif dan struktur jaringannya yang relatif sederhana. Dengan memahami BPNN, peneliti dan praktisi dapat membangun dasar yang kuat sebelum beralih ke model JST yang lebih kompleks.

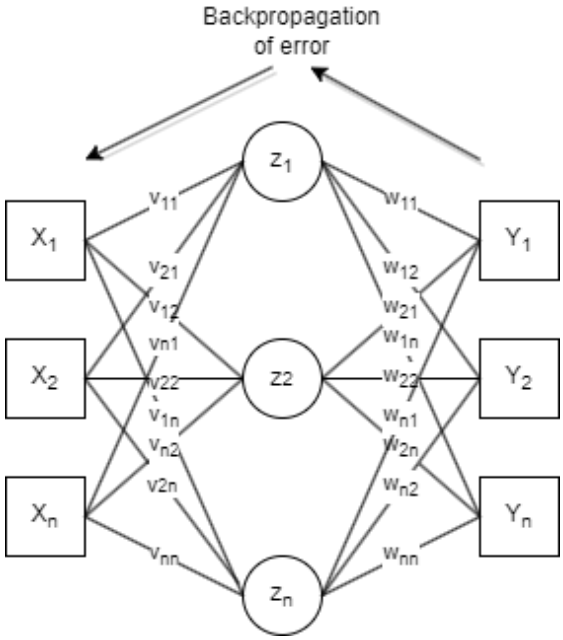
Struktur artikel ini adalah sebagai berikut: Bagian I, yang dibaca saat ini, memberikan pendahuluan umum tentang JST, alasan memilih BPNN, dan pentingnya dataset Iris. Bagian II memberikan tinjauan pustaka tentang BPNN dan dataset Iris. Bagian III menjelaskan metodologi yang digunakan. Bagian IV menyajikan hasil dan diskusi. Bagian V menyimpulkan dan memberikan arah untuk penelitian masa depan.

2. Tinjauan Pustaka

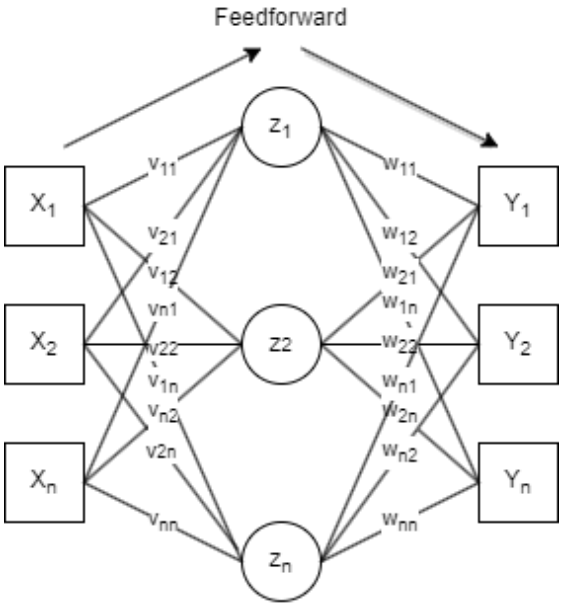
2.1 *Backpropagation Neural Networks*

Backpropagation Neural Networks (BPNN) adalah salah satu metode pembelajaran dalam jaringan saraf tiruan yang paling populer. BPNN diperkenalkan oleh Rumelhart et al. pada tahun 1986 dan sejak itu telah menjadi dasar bagi banyak penelitian dalam bidang pembelajaran mesin [1]. Algoritma ini menggunakan metode gradien turun untuk meminimalkan kesalahan antara keluaran yang diharapkan dan keluaran aktual dari jaringan.

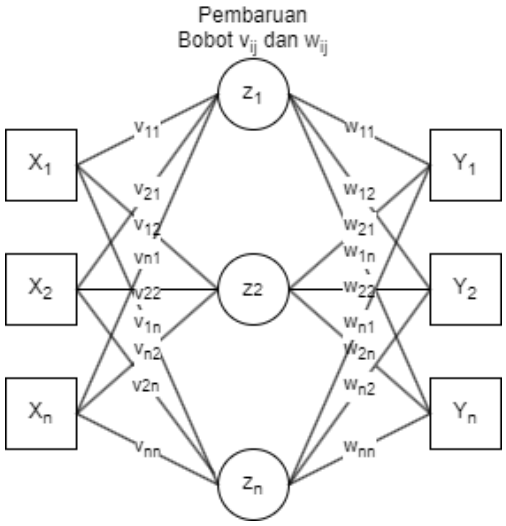
Dalam BPNN, kesalahan dikomputasi di setiap neuron dan disebarkan kembali ke neuron sebelumnya, sehingga bobot dan bias dapat diperbarui untuk meminimalkan kesalahan tersebut. Proses ini diulang hingga kesalahan mencapai nilai minimum yang diinginkan atau setelah jumlah iterasi tertentu [2].



Gambar 2. Proses *backpropagation* dimana dilakukan perhitungan balk dari Y ke X dimana kesalahan prediksi Y dianalisis untuk menghitung gradien pada setiap bobot.



Gambar 1. Proses *feedforward* yang digambarkan sebuah diagram yang menunjukkan data masukan dari input X ke output Y, data melewati lapisan Z dengan bobot v dan w, menghasilkan prediksi.



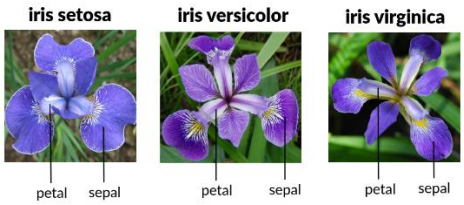
Gambar 3. Proses pembaruan bobot dimana bobot v dan w diperbarui berdasarkan gradien untuk meningkatkan akurasi di iterasi berikutnya.

2.2 Dataset Iris

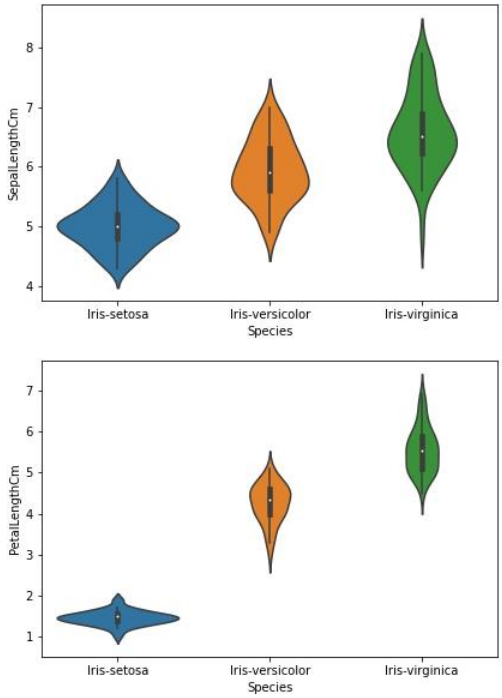
Dataset Iris, yang diperkenalkan oleh Ronald A. Fisher pada tahun 1936, adalah salah satu dataset paling terkenal dalam literatur pembelajaran mesin [3]. Dataset ini

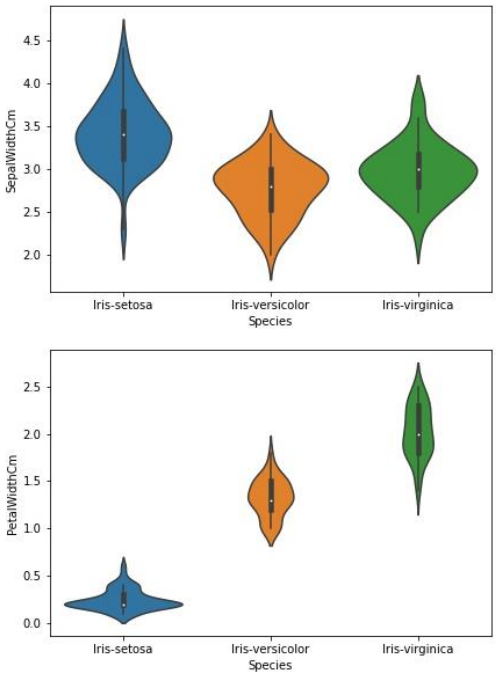
terdiri dari 150 sampel dari tiga spesies iris, yaitu: Iris setosa, Iris versicolour, dan Iris virginica. Setiap spesies memiliki 50 sampel, dan setiap sampel memiliki empat fitur, yaitu: panjang sepal, lebar sepal, panjang petal, dan lebar petal.

Dataset Iris telah digunakan dalam berbagai penelitian untuk menguji efektivitas algoritma klasifikasi. Karena sifatnya yang sederhana dan struktur datanya yang jelas, dataset ini sering digunakan sebagai titik awal dalam pembelajaran mesin dan analisis data [4].



Gambar 4. 3 kelas *iris setosa*, *versicolor*, dan *virginica.* dibedakan dari panjang petal dan sepal.





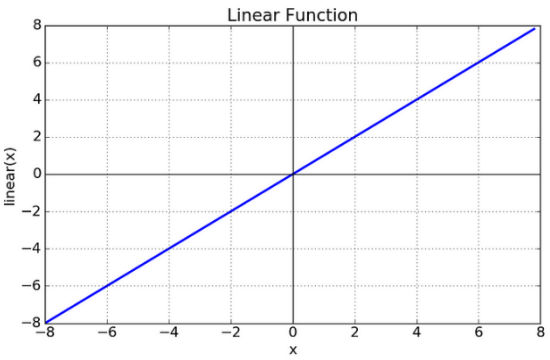
Gambar 5. Distribusi kelas dari tiap jenis iris yang akan diuji[5].

2.3 Fungsi Aktivasi

Fungsi aktivasi memainkan peran krusial dalam arsitektur jaringan saraf tiruan. Fungsi ini bertanggung jawab untuk mengubah input dari sebuah neuron menjadi sebuah output, yang kemudian akan menjadi input bagi neuron berikutnya dalam jaringan.

2.3.1 Fungsi Aktivasi Linear

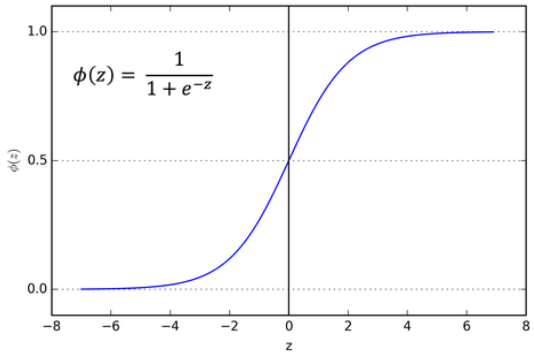
Fungsi aktivasi linear, atau fungsi identitas, menghasilkan output yang sama dengan inputnya. Meskipun sederhana, fungsi ini lebih sesuai untuk masalah regresi. Namun, pada lapisan tersembunyi, fungsi ini kurang efektif karena tidak dapat menangkap non-linearitas.



Gambar 6. Fungsi aktivasi linear dimana f(x) = x

2.3.2 Fungsi Sigmoid (Logistik)

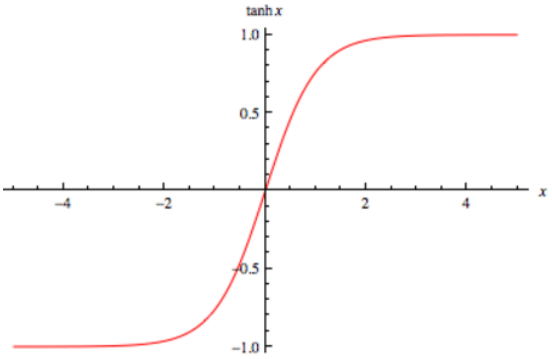
Fungsi sigmoid menghasilkan output antara 0 dan 1. Cocok untuk lapisan output pada tugas klasifikasi biner. Namun, ia rentan terhadap masalah vanishing gradient, yang dapat memperlambat konvergensi selama pelatihan.



Gambar 7. Fungsi Sigmoid dengan range 0 sampai 1.

2.3.3 Fungsi Tangen Hiperbolik (Tanh)

Mirip dengan sigmoid, tetapi menghasilkan output antara -1 dan 1. Lebih efektif daripada sigmoid dalam menangkap non-linearitas, tetapi masih rentan terhadap masalah vanishing gradient.



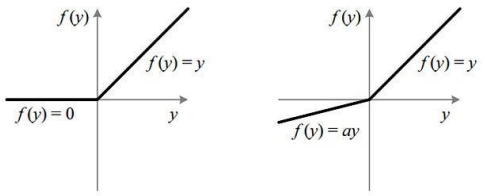
Gambar 8. Fungsi Tanh(x) , mirip sigmoid namun dengan range -1 sampai 1.

2.3.4 Fungsi ReLU (Rectified Linear Unit)

ReLU menghasilkan output yang sama dengan input untuk semua nilai positif dan menghasilkan output nol untuk semua nilai negatif. Cocok untuk lapisan tersembunyi karena efisiensi komputasional dan kemampuannya menangkap non-linearitas. Namun, ia mungkin rentan terhadap "neuron mati" yang tidak aktif selama pelatihan.

2.3.5 Fungsi Leaky ReLU

Varian dari ReLU, menghasilkan output kecil untuk input negatif, bukan nol. Hal ini membantu mengatasi masalah "neuron mati" yang mungkin terjadi dengan ReLU.



Gambar 9. Fungsi ReLU dan leaky ReLU. Dengan perbedaan utama pada fase x < 0 untuk leaky terdapat konstanta a untuk mencegah fenomena “neuron mati”.

2.3.6 Fungsi Softmax

Softmax mengonversi vektor nilai kelas menjadi vektor probabilitas. Sangat sesuai untuk lapisan output pada tugas klasifikasi multikelas.

$$\text{softmax}(x) = \begin{bmatrix} \frac{e^{x1}}{\sum_{i=1}^{n} e^{x_i}} \\ \frac{e^{x2}}{\sum_{i=1}^{n} e^{x_i}} \\ \cdots \\ \frac{e^{xn}}{\sum_{i=1}^{n} e^{x_i}} \end{bmatrix}$$

Gambar 10. Fungsi Softmax. Untuk hubungan hidden layer terakhir ke layer output. Dengan nilai probabilistik.

Pemilihan fungsi aktivasi tergantung pada jenis masalah yang dihadapi dan arsitektur jaringan yang digunakan. Berikut adalah pemilihan berdasarkan masalah yang dihadapi.

- **Klasifikasi Biner**: Sigmoid atau Tanh untuk lapisan output.
- **Klasifikasi Multikelas**: Softmax untuk lapisan output.
- **Lapisan Tersembunyi**: ReLU atau variasinya (seperti Leaky ReLU) karena efisiensi dan kemampuan menangkap non-linearitas.
- **Regresi**: Fungsi aktivasi linear untuk lapisan output.

2.4 Inisialisasi Bobot

Inisialisasi bobot (disini bobot v dan w) merupakan langkah krusial dalam proses pembelajaran jaringan saraf tiruan. Menentukan nilai awal bobot dengan tepat dapat mempercepat konvergensi dan meningkatkan akurasi model. Sebuah inisialisasi yang buruk dapat mengakibatkan pelatihan yang lambat atau bahkan gagal mencapai solusi yang optimal.

2.4.1 Randomisasi

Dalam metode ini, bobot diinisialisasi dengan nilai acak kecil yang diambil dari distribusi seragam atau normal. Pendekatan ini memastikan bahwa setiap neuron memiliki perilaku awal yang berbeda, mencegah stagnasi selama fase awal pelatihan. Meskipun sederhana, randomisasi seringkali cukup efektif untuk banyak aplikasi.

2.4.2 Metode Nguyen-Widrow

Metode ini dirancang khusus untuk meningkatkan inisialisasi bobot pada jaringan dengan fungsi aktivasi sigmoid. Nguyen-Widrow mengusulkan suatu teknik yang menyesuaikan rentang nilai bobot berdasarkan jumlah neuron pada lapisan input dan lapisan tersembunyi. Dengan demikian, distribusi bobot awal lebih sesuai dengan bentuk fungsi aktivasi, yang pada gilirannya meningkatkan kecepatan konvergensi selama pelatihan.

Metode Inisialisasi Bobot Nguyen Widrow:

Menentukan faktor skala (fungsi dari arsitektur jaringan):

β : faktor skala

$\beta (V_{ij}) = 0.7 (m)^{1/n}$

$\beta (W_{jk}) = 0.7 (l)^{1/m}$

Algoritma :

- **Inisialisasi bobot secara random**

  V_{ij} = bilangan acak antara -0.5 sampai +0.5

  W_{jk} akan diperlakukan sama

- **Hitung :** $\| v_j \| = \sqrt{\{\sum_{i=1}^{p} (v_{ij})^2\}}$ p total semua bobot
- **Update bobot** $v_{ij} = \frac{\beta v_{ij}}{\| v_j \|}$
- **Set bias** v_{0j} **: bilangan acak antara -** β **sampai** β

Gambar 11. Metode inisialisasi Nguyen-widrow dimana bobot awal disesuaikan berdasarkan factor skala dan parameter *neural-network*.

3. Metodologi Penelitian

3.1 Algoritma Backpropagation Neural Networks (BPNN)

BPNN adalah salah satu metode dalam jaringan saraf tiruan yang digunakan untuk pembelajaran terawasi. Algoritma ini terdiri dari dua fase: propagasi maju dan propagasi mundur.

3.1.1 Propagasi Maju (*Feedforward*)

Dalam fase ini, input diberikan ke jaringan, dan informasi tersebut bergerak dari lapisan input ke lapisan keluaran melalui lapisan tersembunyi. Setiap neuron pada lapisan berikutnya menerima informasi dari neuron pada lapisan sebelumnya, mengalikannya dengan bobot yang sesuai, dan meneruskannya setelah menerapkan fungsi aktivasi.

3.1.2 Propagasi Mundur (*Backpropagation*)

Setelah fase propagasi maju, kesalahan antara keluaran yang diharapkan dan keluaran aktual dihitung. Kesalahan ini kemudian dipropagasikan kembali dari lapisan keluaran ke lapisan input untuk memperbarui bobot dan bias.

3.2 *Dataset Iris*

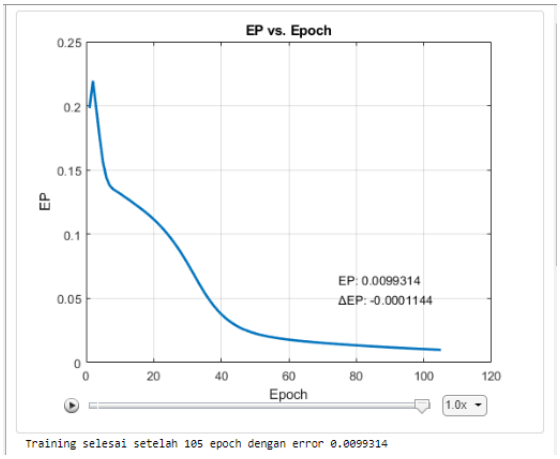
Dataset Iris terdiri dari 150 sampel dari tiga spesies iris dengan empat fitur untuk setiap sampel. Dataset ini dibagi menjadi dua bagian: data pelatihan dan data pengujian. Data pelatihan digunakan untuk melatih model, sedangkan data pengujian digunakan untuk menguji kinerja model.

3.3 Implementasi pada Matlab

Matlab adalah lingkungan pemrograman yang populer untuk analisis data dan pembelajaran mesin. Dalam penelitian ini, BPNN diimplementasikan pada Matlab dengan menggunakan fitur Neural Network Toolbox. Proses meliputi:

- Pra-pemrosesan Data: Data dinormalisasi untuk memastikan bahwa semua fitur memiliki skala yang sama.
- Pembentukan Jaringan: Jaringan dengan satu lapisan tersembunyi dibuat, dan jumlah neuron di lapisan tersembunyi ditentukan berdasarkan eksperimen.
- Penentuan parameter awal dan fungsi : Penentuan parameter awal seperti bobot awal menggunakan metode Nguyen-widrow. Karena data bersifat 0-1 fungsi sigmoid digunakan sebagai fungsi aktivasi. Rasio antara data training dan testing adalah 0.7 : 1.
- Pelatihan Jaringan: Jaringan dilatih dengan menggunakan data pelatihan dan algoritma backpropagation. Proses training dilakukan menggunakan perkalian matriks untuk parameter dan beban agar siap diimplementasikan pada sistem kendali. Proses training dan pengujian diiterasi sebanyak 100 kali.
- Evaluasi Model: Setelah pelatihan, model dievaluasi dengan menggunakan data pengujian untuk mengukur jumlah Epoch tiap iterasi, akurasi, confusion matrix, dan distribusi error.

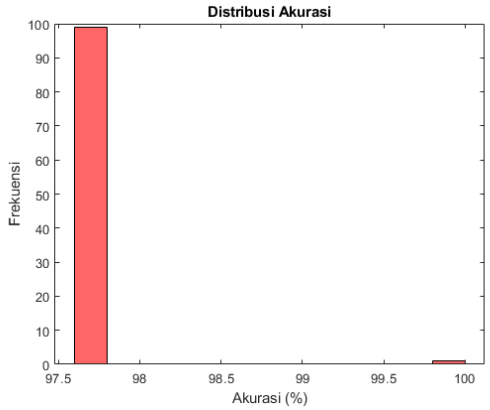
4. Hasil



Gambar 12. Contoh grafik EP vs Epoch dalam proses training.

Dalam penelitian ini, eksperimen dilakukan dengan berbagai konfigurasi untuk menentukan parameter terbaik dari model BPNN. Hasil dari eksperimen tersebut adalah sebagai berikut:

4.1 Akurasi Model

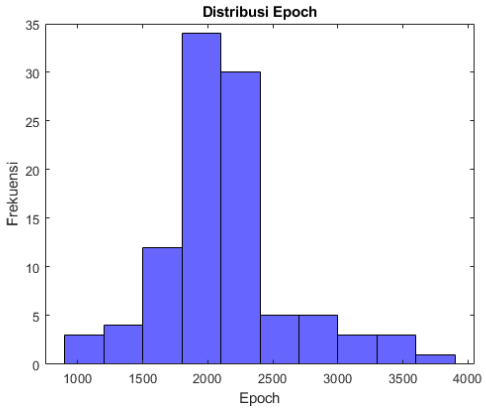



```
Statistik Deskriptif untuk Akurasi:
Rata-rata: 97.8%
Median: 97.7778%
Modus: 97.7778%
Max: 100%
Min: 97.7778%
Standar Deviasi: 0.22222%
Rentang: 2.2222%
```

Gambar 13. Hasil analisis akurasi model

Akurasi pelatihan menunjukkan performa yang konsisten dan tinggi. Rata-rata akurasi adalah 97.8% dengan variasi yang sangat kecil, ditunjukkan oleh standar deviasi sebesar 0.22222%. Meskipun akurasi maksimum mencapai 100%, median dan modus menunjukkan bahwa sebagian besar iterasi memiliki akurasi sekitar 97.7778%. Rentang akurasi hanya 2.2222%, menunjukkan konsistensi dalam performa model..

4.2 Evaluasi Epoch Pelatihan



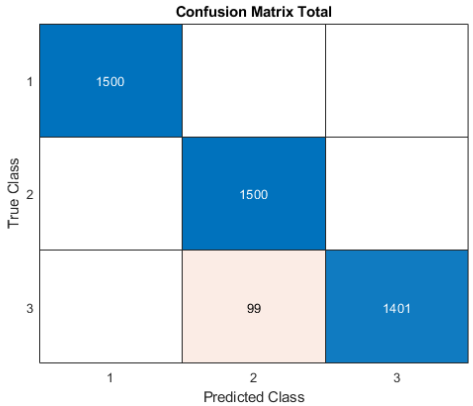

```
Statistik Deskriptif untuk Epoch:
Rata-rata: 2138.84
Median: 2065
Modus: 1903
Max: 3803
Min: 1093
Standar Deviasi: 497.7648
Rentang: 2710
```

Gambar 14. Hasil analisis Epoch Pelatihan.

Dari hasil pelatihan, epoch memiliki rata-rata sebesar 2138.84 dengan variasi yang cukup signifikan, ditunjukkan oleh standar deviasi sebesar 497.7648. Rentang epoch adalah 2710, dengan nilai minimum dan maksimum masing-masing 1093 dan 3803. Median menunjukkan bahwa setengah dari epoch berada di bawah 2065, sedangkan modus menunjukkan bahwa epoch paling sering muncul adalah 1903

4.3 Analisis Kesalahan

Dari hasil testing selama 100 iterasi, terdapat kesalahan klasifikasi yang konsisten dimana objek dengan true class 3 diprediksi sebagai kelas 2. Namun, kesalahan ini adalah satu-satunya kesalahan yang terdeteksi, menunjukkan bahwa model memiliki tingkat ketepatan yang tinggi dalam klasifikasi objek lainnya.



Gambar 15. Confusion Matrix dari Model

5. Diskusi

Hasil dari eksperimen menunjukkan bahwa BPNN adalah metode yang efektif untuk klasifikasi dataset Iris. Meskipun demikian, ada beberapa poin penting yang perlu diperhatikan:

Kompleksitas Model:

Meskipun BPNN dapat mencapai akurasi yang tinggi, model tersebut memiliki banyak parameter yang perlu dioptimalkan. Hal ini dapat menyebabkan model menjadi kompleks dan memerlukan waktu pelatihan yang lebih lama.

Pentingnya Pra-pemrosesan:

Normalisasi data sangat penting untuk memastikan konvergensi yang cepat selama pelatihan. Tanpa normalisasi, model mungkin memerlukan lebih banyak iterasi untuk konvergen atau bahkan mungkin tidak konvergen sama sekali.

Analisis Kesalahan:

Meskipun model memiliki akurasi yang tinggi, masih ada beberapa sampel yang diklasifikasikan dengan salah. Analisis kesalahan dapat memberikan wawasan tentang batasan model dan bagaimana meningkatkannya di masa depan

6. Kesimpulan

Dari penelitian ini, dapat disimpulkan bahwa BPNN adalah metode yang kuat untuk klasifikasi dataset Iris. Dengan akurasi yang tinggi dan metrik evaluasi lainnya yang menjanjikan, BPNN menunjukkan potensinya sebagai alat untuk tugas klasifikasi lainnya. Namun, penting untuk mempertimbangkan kompleksitas model dan kebutuhan pra-pemrosesan saat menerapkannya pada dataset lain atau masalah yang lebih kompleks.

Daftar Pustaka

[1] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[2] Haykin, S. (1999). Neural networks: a comprehensive foundation. Prentice Hall PTR.

[3] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. Annals of eugenics, 7(2), 179-188.

[4] Dua, D., & Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[5] G. Chauhan, “Iris dataset project from UCI Machine Learning Repository,” Machine Learning HD, https://machinelearninghd.com/iris-dataset-uci-machine-learning-repository-project/ (accessed Sep. 14, 2023).

Dataset dapat diakses di :
Iris Dataset | Kaggle
https://www.kaggle.com/datasets/vikrishnan/iris-dataset

Lampiran Program Matlab

```
% Membaca dataset
data = readtable('iris.data.csv');
input = table2array(data(:, 1:end-1)); % Asumsikan kolom terakhir adalah
label
labels = data{:, end}; % Gunakan kurung kurawal untuk mengakses data dalam
tabel


% Konversi label teks menjadi angka
uniqueLabels = unique(labels);
target = zeros(height(data), 1);
for i = 1:length(uniqueLabels)
    target(strcmp(labels, uniqueLabels{i})) = i;
end


% Normalisasi data
input = (input - min(input)) ./ (max(input) - min(input));


% Membagi dataset menjadi training dan testing dengan Stratified Sampling
ratio = 0.7;
uniqueTargets = unique(target);
trainIdx = [];
testIdx = [];


for i = 1:length(uniqueTargets)
    classIdx = find(target == uniqueTargets(i)); % Indeks dari kelas
tertentu
    numTrainClass = round(ratio * length(classIdx));

    randClassIdx = randperm(length(classIdx));

    trainClassIdx = classIdx(randClassIdx(1:numTrainClass));
    testClassIdx = classIdx(randClassIdx(numTrainClass+1:end));

    trainIdx = [trainIdx; trainClassIdx];
    testIdx = [testIdx; testClassIdx];
end


inputTrain = input(trainIdx, :);
targetTrain = target(trainIdx);
inputTest = input(testIdx, :);
targetTest = target(testIdx);


% Inisialisasi
alpha = 0.4; %kecepatan learning
miu = 0.75; %momentum learning
n = size(inputTrain, 2); % Jumlah fitur
```

```
m = 10; % Jumlah neuron di hidden layer
l = length(unique(targetTrain)); % Jumlah kelas
max_epoch = 10000;
EP = inf;


% Inisialisasi bobot dan bias dengan Nguyen-Widrow
beta = 0.7 * m^(1/n);
v = rand(n, m) * 2 - 1;
normV = sqrt(sum(v.^2));
v = beta * v ./ normV;
v0 = (rand(1, m) * 2 - 1) * beta;
w = rand(m, l) * 2 - 1;
w0 = rand(1, l) * 2 - 1;


%inisialiasi momentum perubahan bobot
deltaWPrev = zeros(size(w));
deltaVPrev = zeros(size(v));
deltaW0Prev = zeros(size(w0));
deltaV0Prev = zeros(size(v0));


epoch = 0;
previousEP = 0; %untuk kebutuhan grafik realtime
errorList = []; % Untuk menyimpan error pada setiap epoch
figure; % Membuka jendela figure untuk grafik




while EP > 0.01 && epoch < max_epoch
    EP = 0;
    for p = 1:size(inputTrain, 1)
        xi = inputTrain(p, :);
        tk = zeros(1, l);
        tk(targetTrain(p)) = 1;


        % Feedforward
        z_in = zeros(1, m);
        z = zeros(1, m);
        for j = 1:m
            z_in(j) = xi * v(:, j) + v0(j);
            z(j) = 1 / (1 + exp(-z_in(j)));
        end

        y_in = zeros(1, l);
        y = zeros(1, l);
        for k = 1:l
            y_in(k) = z * w(:, k) + w0(k);
            y(k) = 1 / (1 + exp(-y_in(k)));
        end
```

```
        % Backpropagation
        delta_k = zeros(1, l);
        for k = 1:l
            delta_k(k) = (tk(k) - y(k)) * y(k) * (1 - y(k));
        end

        delta_j = zeros(1, m);
        for j = 1:m
            sum_delta_w = 0;
            for k = 1:l
                sum_delta_w = sum_delta_w + delta_k(k) * w(j, k);
            end
            delta_j(j) = sum_delta_w * z(j) * (1 - z(j));
        end


        % Update bobot dan bias dengan momentum
        for k = 1:l
            for j = 1:m
                deltaW = alpha * delta_k(k) * z(j);
                w(j, k) = w(j, k) + deltaW + miu * deltaWPrev(j, k);
                deltaWPrev(j, k) = deltaW;
            end
            deltaW0 = alpha * delta_k(k);
            w0(k) = w0(k) + deltaW0 + miu * deltaW0Prev(k);
            deltaW0Prev(k) = deltaW0;
        end

        for j = 1:m
            for i = 1:n
                deltaV = alpha * delta_j(j) * xi(i);
                v(i, j) = v(i, j) + deltaV + miu * deltaVPrev(i, j);
                deltaVPrev(i, j) = deltaV;
            end
            deltaV0 = alpha * delta_j(j);
            v0(j) = v0(j) + deltaV0 + miu * deltaV0Prev(j);
            deltaV0Prev(j) = deltaV0;
        end




        EP = EP + sum((tk - y).^2)/2;


    end
    EP = EP / size(inputTrain, 1);
    epoch = epoch + 1;
    errorList = [errorList, EP]; % Menambahkan error ke list


    % Plotting grafik Error per Epoch (EP)
    plot(errorList, 'LineWidth', 2);
    title('EP vs. Epoch');
```

```
    xlabel('Epoch');
    ylabel('EP');
    grid on;

   % Menampilkan nilai EP saat ini
    strEP = ['EP: ' num2str(EP)];
    hTextEP = annotation('textbox', [0.6, 0.25, 0.3, 0.1], 'String', strEP,
'EdgeColor', 'none', 'VerticalAlignment', 'top');

    % Menampilkan perubahan EP
    deltaEP = EP - previousEP;
    strDeltaEP = ['ΔEP: ' num2str(deltaEP)];
    hTextDeltaEP = annotation('textbox', [0.6, 0.2, 0.3, 0.1], 'String',
strDeltaEP, 'EdgeColor', 'none', 'VerticalAlignment', 'top');

    drawnow; % Memperbarui grafik secara real-time
    previousEP = EP;
end


disp(['Training selesai setelah ' num2str(epoch) ' epoch dengan error '
num2str(EP)]);


% Evaluasi pada dataset testing
correct = 0;
for p = 1:size(inputTest, 1)
    xi = inputTest(p, :);
    tk = zeros(1, l);
    tk(targetTest(p)) = 1;


    % Feedforward
    z_in = zeros(1, m);
    z = zeros(1, m);
    for j = 1:m
        for i = 1:n
            z_in(j) = z_in(j) + xi(i) * v(i, j);
        end
        z_in(j) = z_in(j) + v0(j);
        z(j) = 1 / (1 + exp(-z_in(j)));
    end

    y_in = zeros(1, l);
    y = zeros(1, l);
    for k = 1:l
        for j = 1:m
            y_in(k) = y_in(k) + z(j) * w(j, k);
        end
        y_in(k) = y_in(k) + w0(k);
        y(k) = 1 / (1 + exp(-y_in(k)));
    end
```

```
    [~, predicted] = max(y);
    predictedLabels(p) = predicted;
    if predicted == targetTest(p)
        correct = correct + 1;
    end
end
    figure; % Membuka jendela figure baru
    confusionchart(targetTest, predictedLabels);
    title('Confusion Matrix');

    % Visualisasi Kesalahan
    errors = targetTest - predictedLabels;
    figure; % Membuka jendela figure baru
    histogram(errors);
    title('Distribution of Errors');
    xlabel('Error');
    ylabel('Number of Samples');



accuracy = correct / size(inputTest, 1) * 100;
disp(['Akurasi pada dataset testing: ' num2str(accuracy) '%']);
```