Kitchen Ordering Application with ANN Backpropagation for Order Prediction

Muhammad Said Idris[1], Rismayanti[2]
[1,2] Engineeringand Computer Science, Informatic Engineering, Universitas Harapan Medan, Indonesia
[1]said.idris011@gmail.com,[2]risma.stth@gmail.com

Article Info

Article history:

Received September 24, 2025
Revised November 11, 2025
Accepted November 26, 2025

Keywords:

Artificial Neural Network
Backpropagation Algorithm
Culinary Industry
Kitchen Inventory Management
Order Prediction

ABSTRACT

Raw material inventory management is an important aspect of kitchen operations, especially in the culinary industry, which depends on the timely and efficient availability of stock. Inaccurate raw material ordering can result in excess costs, stockpiling, or shortages that affect service quality. Therefore, an intelligent system is needed that can accurately and automatically predict raw material requirements. This study aims to design a kitchen raw material application using the Artificial Neural Network (ANN) method and the Backpropagation algorithm as a method for automatically predicting order quantities. By using historical raw material data as input to train the model, analysis data obtained from the mean squared error, mean absolute error, and R squared error can be used to study demand patterns over time. The test results show that the ANN model with the backpropagation algorithm is capable of providing fairly accurate predictions of kitchen raw material requirements. The application created can also simplify the raw material ordering process by providing recommendations for effective purchase quantities. Thus, this system can help manage inventory more efficiently and based on data.

Corresponding Author:

Muhammad Said Idris
Universitas Harapan Medan
Email: said.idris011@gmail.com

1. INTRODUCTION

Efficiency in managing raw material inventory is a fundamental factor that determines the sustainability of a culinary business. An inaccurate raw material procurement process has the potential to cause serious problems such as excess stock, which leads to waste, or stock shortages, which hinder customer service. Inventory management in the culinary business is often done manually and periodically, rather than continuously, which risks causing operational inefficiencies[1].

In practice, most businesses still rely on intuition or experience to determine the amount of raw materials to order. This method is subjective and prone to error, especially when there are fluctuations in customer demand that are difficult to predict. Data with irregular patterns, such as raw material usage data, is difficult to predict accurately using conventional methods[2]. Therefore, an approach that can utilize historical data to produce more accurate predictions is needed.

With the advancement of information technology, various data-based intelligent methods have been developed. One promising approach is Artificial Neural Network (ANN), a branch of artificial intelligence that mimics the way the human brain recognizes patterns and learns from historical data. With the advancement of information technology, various data-based intelligent methods have been developed. One promising approach is Artificial Neural Network (ANN), a branch of artificial intelligence that mimics the way the human brain recognizes patterns and learns from historical data. ANN has the ability to model complex non-linear relationships and has proven effective in producing highly accurate predictions. ANN can represent complex input-output relationships, making it suitable for solving prediction problems in dynamic data[3].

The Backpropagation algorithm is one of the most widely used training methods in artificial neural networks. This algorithm works by gradually improving the network weights to minimize the error between the predicted output and the target. In a publication in Nature, this algorithm was introduced as an efficient directed learning method for training multilayer networks. In its development, Backpropagation has proven to be flexible and applicable in various fields, including pattern recognition, data forecasting, and complex system control. The International Conference on Computer Engineering also confirmed the effectiveness of this algorithm for supervised learning[4].

The application of the ANN method with the Backpropagation algorithm in kitchen ingredient prediction systems can provide innovative solutions to procurement problems. By utilizing historical data, the system can learn demand patterns and generate recommendations for optimal order quantities. This not only helps reduce the risk of overstocking or understocking, but also supports data-driven decision making. Machine learning can improve the effectiveness of supply chain management through more accurate demand predictions[5]. ANN-based prediction systems are also in line with the needs of the business world, which increasingly demands speed and efficiency in decision making. In the context of the culinary industry, the system's ability to provide automatic ordering recommendations can save time, energy, and operational costs. In the Operations Research Journal, structured inventory management can reduce waste and increase customer satisfaction[6].

This study focuses on designing a kitchen ingredient ordering application using ANN with the Backpropagation algorithm to predict the needs of two main ingredients, namely salmon and beef. The application was developed using the Python programming language with the Streamlit framework, which enables a web-based and easily accessible system. With this approach, it is hoped that the system can provide practical contributions to culinary businesses in managing inventory automatically, accurately, and efficiently. Thus, this research has a high urgency to support technological innovation in raw material inventory management. The contribution is not only practical but also academic, namely by proving the effectiveness of the ANN and Backpropagation methods in solving real problems in the culinary industry.

2. METHOD

2.1 Artificial Neural Network

Artificial neural networks are one of the artificial representations of the human brain that attempt to mimic the learning processes that occur in the brain. The term "artificial" here refers to the fact that these neural networks are implemented through computer programs capable of performing various computational processes during learning. Artificial neural networks are powerful data modeling tools capable of capturing and representing complex input-output relationships[3]. Python is a highly recommended programming language for data analysis. In recent years, the continuous development of Python libraries, especially Pandas, has made it a powerful choice for data analysis. Pandas makes it easy for users to process, analyze, and visualize data, simplifying the data analysis process[7]. The activation function is used to calculate the output value based on the total input received and is very important in determining the ability of artificial neural networks to solve problems. The following are the types of artificial neural network architectures:

1.Single layer network

Consists of a single output layer that is directly connected to the input. There is no hidden layer. Suitable for simple and linear problems.

2.Multi Layer Network

Has one or more hidden layers between the input and output. Capable of handling complex and non-linear problems. The activation function used is ReLU (Rectified Linear Unit):

$$output = max(0, \mathrm{W} \cdot \mathrm{X} + \mathrm{b} \ldots (1)$$

3.Competitive Layer Network

There is a competitive layer where neurons compete to be the winner in the learning process.

2.2 Backpropagation

Backpropagation is an algorithm in a system designed to mimic the human brain's ability to solve problems by learning through weight modification. This algorithmic method in supervised learning can reduce errors in the results produced. This method is applied to train the network's ability to respond accurately to input samples using the data used during the training process[4].

Backpropagation is widely applied in various domains to solve complex prediction and classification problems. Backpropagation is widely recognized for its success in training multilayer neural networks, as well as being the main foundation for various applications in artificial intelligence. The backpropagation algorithm works by adjusting the output of the neural network to reduce errors in predictions[8].

The principle of backpropagation is to reduce prediction errors in artificial neural networks by calculating and using the gradient of the loss function for each network parameter (weights and biases), then gradually updating those parameters to reduce the error. Based on the quote[9]. The following are the main stages of the backpropagation working principle:

1. Feedforward

Input data is fed into the network and processed through all layers to produce a predicted output. This output is then compared to the actual target to calculate the error using a loss function such as Mean Squared Error or Cross-Entropy. The first step in the feedforward process is calculating the input to the hidden layer.

$$\mathrm{Zh} \;=\; \mathrm{X}\,.\,\mathrm{Wih} + \mathrm{bh} \ldots (2)$$

Step 2: Sigmoid activation :

$$\mathrm{a}_h = \sigma(z_h) = \frac{1}{1+e^{-zh}} \ldots (3)$$

Steps to calculate error:

$$MSE \;= \frac{1}{n}\sum_{i=1}^{n}\left(y_{true} - y_{pred}\right)^2 \ldots . (4)$$

2. Backward Propagation: The resulting error is calculated by each neuron using the chain rule of calculus, then this error is propagated backward from the output layer to the previous layers. This process calculates the gradient of the loss function for each Weight and bias. With the first calculation step, calculate the delta of the sigmoid activation function output layer against the quadratic error function using the formula:

$$\delta = \left(y_{pred} - y_t\right)\cdot\, y_{pred}\cdot\left(1 - y_{pred}\right) \ldots (5)$$

Second, continue calculating the hidden layer delta:

$$\delta_{h1} = \left(w_{ho} - \delta_0\right)\cdot\, a_h \cdot (1 - a_h) \ldots . (6)$$

3.Weight Update (Gradient Descent):Based on the calculated gradient, the weights and biases are updated using optimization methods such as gradient descent. The weights are changed gradually to reduce the error as much as possible. This process is repeated until the error reaches the lowest value or meets the convergence criteria.For the gradient weight formula: Weight update formula (output layer):

$$\mathrm{W}_{\mathrm{ho1baru}} = \left(w_{ho1lama} - \eta\,.\,\delta_0\,.\,a_{hq}\right) \ldots .(7)$$

Bias update formula:

$$b_{obaru} = b_{olama-}\,\eta\,.\,\delta_0 \;\; \ldots .(8)$$

The core of the backpropagation working principle is a learning method based on gradient-based optimization, in which artificial neural networks repeatedly adjust weights by utilizing error backpropagation.

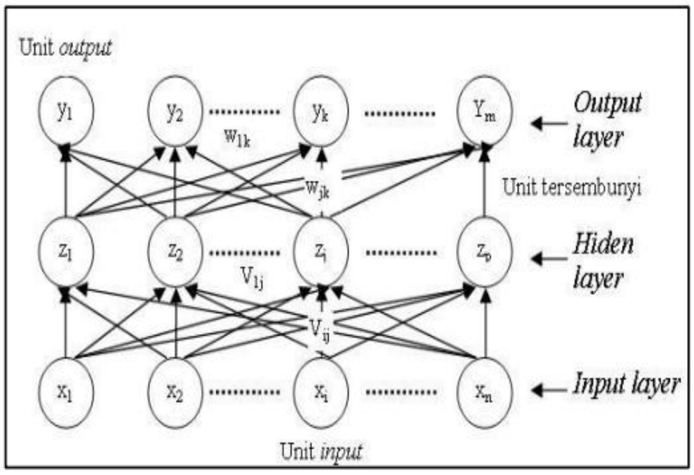


Figure 1. Basic Architecture of Backpropagation

The figure depicts a feedforward neural network architecture consisting of an input layer, a hidden layer, and an output layer. The symbols X_1 to X_n represent input units (data features) that serve as gateways for information entering the neural network. Each input unit represents an attribute or characteristic of the data to be processed by the network. The hidden layer also exists in the backpropagation architecture, consisting of z_1 to z_p. This hidden layer plays an important role in processing and transforming the representation of data from input into a form that is more useful for prediction tasks. Each neuron in the hidden layer receives weighted inputs from all neurons in the previous layer, performs a non-linear transformation through an activation function, and passes the result to the next layer[10].

3. RESULTS AND DISCUSSION

This research resulted in a web-based kitchen ingredient ordering application using the Python programming language and the Streamlit framework. The application is designed to help culinary businesses predict the amount of kitchen ingredients needed for the next period. The system development process was carried out through several implementation stages, namely training and prediction (testing) implementation. The first stage was to train the ANN model using historical kitchen ingredient data. The dataset used contained data on initial stock, arrivals, usage, final stock, and orders. The data was first normalized to a range of 0–1 so that each variable had a comparable scale.

Table 1. Kitchen raw material usage dataset (salmon and beef)

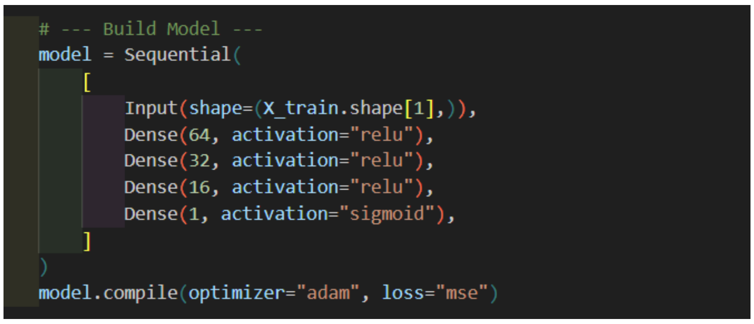
No	stock_awal	Pemakaian	stock_akhir	Pembelian_di terima	order
1	*141.33*	*44.4*	221.93	125	*0*
2	89.87	*5048*	39.38	0	*0*
3	89.87	*50.48*	150	20	*300*
4	200	*20*	80	10	*200*
5	250	*10*	100	15	*250*

3.1 Training Implementation

In this study, the author used a sequential model training method. Sequential in backpropagation usually refers to sequential learning, where network weights are updated each time a piece of data is processed. There are several steps in implementing the sequential method, as explained below:

1. Initialization

Determine the network architecture (number of neurons per layer), initialize weights and biases with small random values, and set training parameters such as learning rate and momentum.

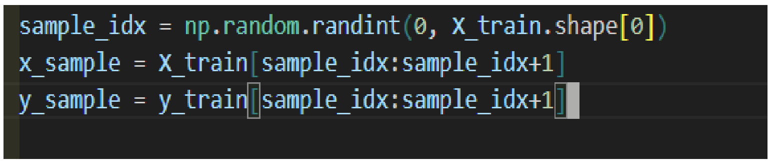


```
# --- Build Model ---
model = Sequential(
    [
        Input(shape=(X_train.shape[1],)),
        Dense(64, activation="relu"),
        Dense(32, activation="relu"),
        Dense(16, activation="relu"),
        Dense(1, activation="sigmoid"),
    ]
)
model.compile(optimizer="adam", loss="mse")
```

Figure 2. Code implementation Initialization steps

2. Take one input data

Take one data pair (x, t), where x = input vector and t = expected target/output.



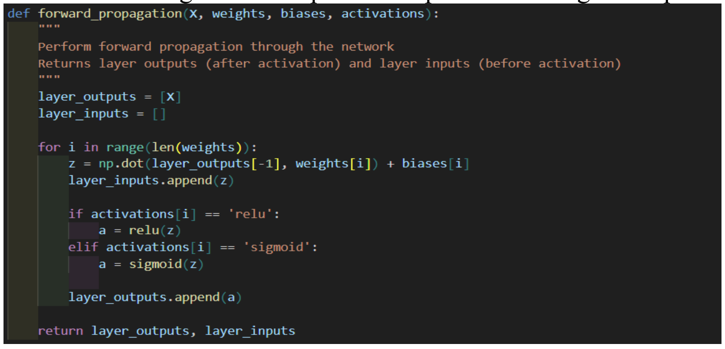
```
sample_idx = np.random.randint(0, X_train.shape[0])
x_sample = X_train[sample_idx:sample_idx+1]
y_sample = y_train[sample_idx:sample_idx+1]
```

Figure 3. Code implementation Step: take one input

The process of selecting a random sample from the training data (X_train and Y_train) for prediction or further analysis. First, sample_idx generates a random index within the range of the training data using np.random.randint().

3. Forward Propagation

Calculating the output of each layer from the input Calculating the output of a layer based on the input. The forward_propagation function in Figure 4.3 implements the forward pass in an artificial neural network by calculating the output of each layer through linear (np.dot) and non-linear (ReLU/Sigmoid activation) operations, storing intermediate results (layer_inputs and layer_outputs) for optimization purposes. This process forms the foundation for training models to predict outputs based on given inputs.

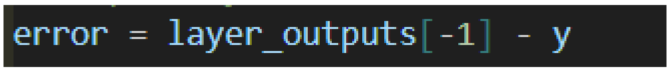


```python
def forward_propagation(X, weights, biases, activations):
    """
    Perform forward propagation through the network
    Returns layer outputs (after activation) and layer inputs (before activation)
    """
    layer_outputs = [X]
    layer_inputs = []

    for i in range(len(weights)):
        z = np.dot(layer_outputs[-1], weights[i]) + biases[i]
        layer_inputs.append(z)

        if activations[i] == 'relu':
            a = relu(z)
        elif activations[i] == 'sigmoid':
            a = sigmoid(z)

        layer_outputs.append(a)

    return layer_outputs, layer_inputs
```

Figure 3. Implementation of code to calculate the output of each layer

4. Calculation error

Calculation of the error from the difference between the actual target and the prediction to reduce the prediction value error.



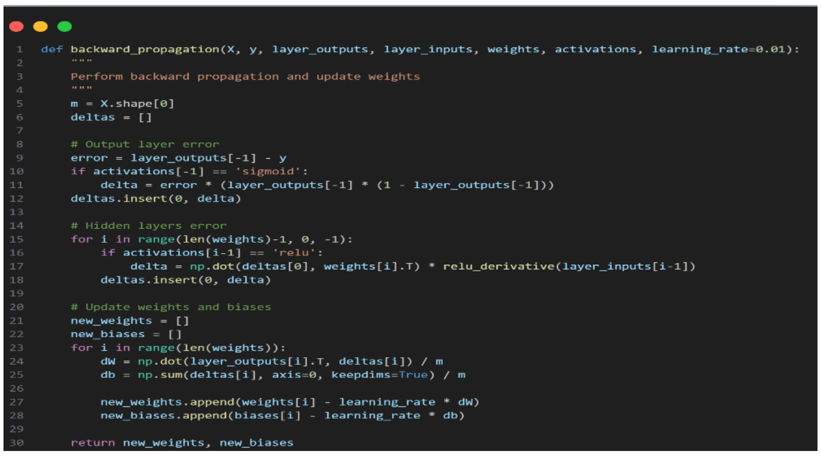
```python
error = layer_outputs[-1] - y
```

Figure 4. Implementation of error calculation code

Calculation of the error between the model's predicted output (layer_outputs[-1]) and the actual target value (y). The variable layer_outputs[-1] represents the final output of the artificial neural network after passing through all layers, while y is the expected ground truth value. The error is calculated simply by subtracting the predicted value from the target value (error = layer_outputs[-1] - y).

5. Backward Propagation

Manually updating parameters by calculating delta and gradient



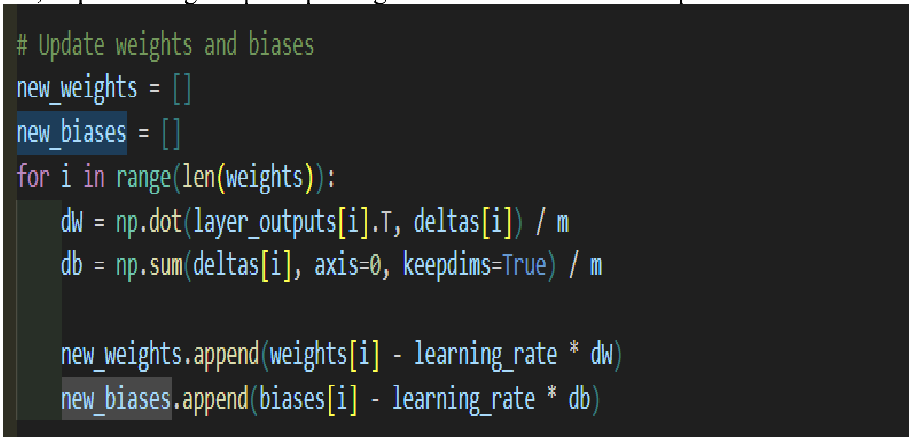
```python
def backward_propagation(X, y, layer_outputs, layer_inputs, weights, activations, learning_rate=0.01):
    """
    Perform backward propagation and update weights
    """
    m = X.shape[0]
    deltas = []

    # Output layer error
    error = layer_outputs[-1] - y
    if activations[-1] == 'sigmoid':
        delta = error * (layer_outputs[-1] * (1 - layer_outputs[-1]))
    deltas.insert(0, delta)

    # Hidden layers error
    for i in range(len(weights)-1, 0, -1):
        if activations[i-1] == 'relu':
            delta = np.dot(deltas[0], weights[i].T) * relu_derivative(layer_inputs[i-1])
        deltas.insert(0, delta)

    # Update weights and biases
    new_weights = []
    new_biases = []
    for i in range(len(weights)):
        dW = np.dot(layer_outputs[i].T, deltas[i]) / m
        db = np.sum(deltas[i], axis=0, keepdims=True) / m

        new_weights.append(weights[i] - learning_rate * dW)
        new_biases.append(biases[i] - learning_rate * db)

    return new_weights, new_biases
```

Figure 5. Implementation of error calculation code

Implementing the backward propagation algorithm to manually update model parameters. This process consists of three main stages: (1) calculating the error in the output layer by comparing the prediction (layer_outputs[-1]) with the actual value (y), where for sigmoid activation, the derivative sigmoid*(1-sigmoid) is used; (2) calculating the error in the hidden layer recursively from the output to the input using the chain rule, utilizing the derivative of the ReLU function for the hidden layer; (3) updating the weights and biases based on the calculated gradient, adjusting the learning rate.

6. Sequential Update

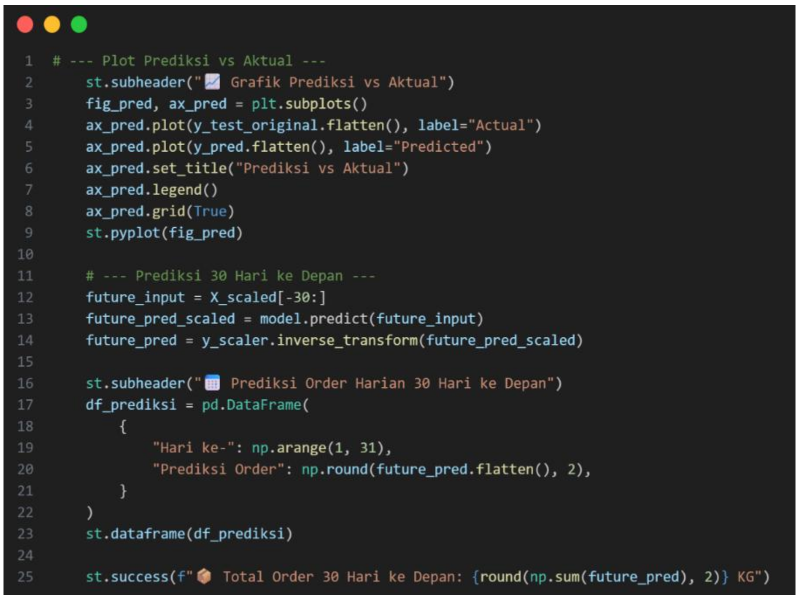
Final update The final stage of backpropagation is the updating of model parameters (weights and biases). This process is carried out by calculating the gradient for each parameter: (1) the weight gradient (dW) is calculated by multiplying the dot product between the transpose of the previous output layer (layer_outputs[i].T) and the current delta error (deltas[i]), then normalized by the number of samples (m); (2) the bias gradient (db) is calculated by summing the delta error along axis 0 and normalizing it. of the backpropagation algorithm; the update is performed immediately after one sample is processed. The two parameters are then updated by subtracting the current value from the result of multiplying the learning_rate by each gradient, implementing the principle of gradient descent for model optimization.



```python
# Update weights and biases
new_weights = []
new_biases = []
for i in range(len(weights)):
    dW = np.dot(layer_outputs[i].T, deltas[i]) / m
    db = np.sum(deltas[i], axis=0, keepdims=True) / m

    new_weights.append(weights[i] - learning_rate * dW)
    new_biases.append(biases[i] - learning_rate * db)
```

Figure 6. Implementation of weight update code

After implementing the sequential method, proceed to the prediction implementation stage, where the model will predict results based on the available dataset. This can be implemented into code as follows:



```python
# --- Plot Prediksi vs Aktual ---
    st.subheader("📈 Grafik Prediksi vs Aktual")
    fig_pred, ax_pred = plt.subplots()
    ax_pred.plot(y_test_original.flatten(), label="Actual")
    ax_pred.plot(y_pred.flatten(), label="Predicted")
    ax_pred.set_title("Prediksi vs Aktual")
    ax_pred.legend()
    ax_pred.grid(True)
    st.pyplot(fig_pred)

    # --- Prediksi 30 Hari ke Depan ---
    future_input = X_scaled[-30:]
    future_pred_scaled = model.predict(future_input)
    future_pred = y_scaler.inverse_transform(future_pred_scaled)

    st.subheader("📅 Prediksi Order Harian 30 Hari ke Depan")
    df_prediksi = pd.DataFrame(
        {
            "Hari ke-": np.arange(1, 31),
            "Prediksi Order": np.round(future_pred.flatten(), 2),
        }
    )
    st.dataframe(df_prediksi)

    st.success(f"📦 Total Order 30 Hari ke Depan: {round(np.sum(future_pred), 2)} KG")
```

Figure 7. Prediction code implementation

There are two main stages in model analysis, namely performance evaluation and future prediction. In the evaluation stage, the model makes predictions on test data (X_test) which produces standardized output (y_pred_scaled), then converted to the original scale using y_scaler.inverse_transform to facilitate interpretation. Evaluation metrics such as MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error) are calculated by comparing the predicted values (y_pred) to the actual values (y_test_original), providing a measure of the model's accuracy in actual unit scale. At the prediction stage, the model is used to forecast 30 periods ahead by taking the last 30 samples from the scaled data (X_scaled[-30:]). The prediction results are then transformed back to the original scale and displayed in a dataframe table using Streamlit, complete with information on the total order predictions for 30 days. This visualization allows users to clearly see prediction trends while understanding model performance through quantitative metrics.

4. CONCLUSION

After implementing all steps of the Backpropagation algorithm during the research process, it can be concluded that.The system that was built is capable of processing data on the consumption and supply of salmon and beef to produce predictions of demand up to 30 days in advance. This shows that the Backpropagation algorithm can be adapted well to cases of time series data-based predictions such as raw material demand. The evaluation results show that the model can provide output in the form of prediction tables and can be visualized through graphs. This prediction model can be used directly in the kitchen raw material procurement process. Both show effective convergence with a significant decrease in loss in the early epoch (Training Loss for beef fell 65.4% from 0.081 to 0.028 in the first epoch; salmon decreased by 56.3% from 0.16 to around 0.07), and ended with stable and low final loss values (beef: 0.024, salmon: 0.06–0.07), confirming the model's success in minimizing error and its readiness for use in prediction. And for MSE: 30.81 RMSE: 41.17 R2 scored: 0.18

ACKNOWLEDGEMENTS

First of all, the author would like to express gratitude for completing this article and research. The author would like to express his deepest gratitude to his supervisor for providing guidance and motivation so that the author could complete this research well. The author realizes that this research still has many shortcomings and needs to be developed further.

REFERENCES

[1] J. Juwita and F. Rahmiyatun, "Penerapan Metode Economic Order Quantity (EOQ) Dan Reorder Point (ROP) Pada Pengendalian Persediaan Bahan Baku Di UMKM Dapur Bunga Berbintang," *J. Maneksi*, vol. 12, no. 4, pp. 818–827, 2023, doi: 10.31959/jm.v12i4.1833.

[2] F. P. Hanifi, S. Syafriandi, and N. Amalita, "Artificial Neural Network Model for Forecasting Inflation Rate in Indonesia Using Backpropagation Algorithm," 2025.

[3] R. Gusriva and Y. M. Putra, "Model Prediksi Kerusakan Sepeda Motor Matic Menggunakan Jaringan Saraf Tiruan dan Metode Hebb ' s Rule," vol. 6, no. 2, pp. 585–592, 2025.

[4] F. Nailah, D. I. Larasati, S. Siswanto, and A. Kalondeng, "Optimasi Metode Jaringan Saraf Tiruan Backpropagation Untuk Peramalan Curah Hujan Bulanan Di Kota Denpasar," *MATHunesa J. Ilm. Mat.*, vol. 12, no. 1, pp. 134–140, 2024, doi: 10.26740/mathunesa.v12n1.p134-140.

[5] V. I. Sunarko, D. L. S. Dimara, P. S. E. Siagian, D. Manalu, and F. T. Anggraeny, "Implementasi K-Fold Dalam Prediksi Hasil Produksi Agrikultur Pada Algoritma K-Nearest Neighbor (KNN)," *INTEGER J. Inf. Technol.*, vol. 10, no. 1, pp. 10–16, 2025, doi: 10.31284/j.integer.2024.v10i1.6892.

[6] A. P. Sheriva, C. Rahmadhini, E. Angelia, G. Perwinta, and B. Ginting, "Implementasi Sistem ERP (Enterprise Resource Planning) Dalam Manajemen Persediaan di PT Indofood Sukses Makmur," vol. 2, no. 2, pp. 83–92, 2024.

[7] M. R. Qisthiano and A. O. Pratiwi, "MENGGUNAKAN ALGORITMA SOBEL DAN PREWITT," vol. 13, no. 2, pp. 1115–1122, 2025.

[8] N. T. Khair, I. Afrianty, F. Syafria, E. Budianita, and S. K. Gusti, "Penerapan Information Gain Untuk Seleksi Fitur Pada Klasifikasi Jenis Kelamin Tulang Tengkorak Menggunakan Backpropagation," vol. 5, no. 4, pp. 666–678, 2025, doi: 10.47065/bulletincsr.v5i4.637.

[9] N. Tri *et al.*, *Deep Learning : Teori , Algoritma , dan Aplikasi*, no. March. 2025.

[10] R. 2020 Mirfiza, "Implementasi Backpropagation Berdasarkan Particle Swarm Optimization Untuk Memprediksi Jumlah Penumpang Kereta Api," p. 91, 2020.