EAI Endorsed Transactions
on Transportation Systems and Ocean Engineering Research Article EAI.EU

Study on IoT application and development of artificial neural networks in vehicle status diagnosis

Hoang-Minh Luu[1,*]

[1]Artificial Intelligent in Transportation, University of Transport Ho Chi Minh City

Abstract

In the contemporary landscape, characterized by the robust advancements of IoT and AI in automation system control and monitoring, vehicle condition diagnostic systems, crucial for ensuring safe and efficient vehicle operation and management, are fully aligned with this technological trajectory. This paper outlines a proposed hardware system designed to facilitate the collection of vehicle parameters for IoT applications and the development of a vehicle condition recognition model utilizing Artificial Neural Network (ANN). The model incorporates two distinct training methodologies: the weight detection method and the learning rate adjustment method. A comparative analysis of these two methods will serve as the foundation for selecting a model that demonstrates superior quality, high accuracy, and optimal efficiency in terms of both time and training resources.

Keywords: IoT system, Artificial Neural Network, Weight detection, Learning rate.

Received on 09 November 2025, accepted on 10 December 2025, published on 12 December 2025

doi: 10.4108/_______________

1. Introduction

Vehicle management plays a key role in ensuring business operations run efficiently. Strict control of the status of vehicles helps to optimize fuel and maintenance costs, improve operation efficiency, and increase the quality of customer service.

Nowadays, with the development of the Industrial Revolution 4.0, it is extremely necessary to combine artificial intelligence (AI) and the Internet of Things (IoT) to improve efficiency in management [1]. In this paper, we propose to build a vehicle condition monitoring system that applies a combination of IoT and AI to enhance the ability to track and manage vehicle operations. IoT devices play a role in continuously collecting data from the physical environment, storing it in a separate database, and then importing it into a central processing system. By applying machine learning models, especially ANN [2], the data are analyzed to carry out the classification process, helping to recognize and evaluate the condition of vehicles.

However, for the system to operate with high accuracy and stable predictability, optimizing the training process is a key factor. This study focuses on comparing the optimal weighting detection method (using the Optuna library) or the method of fine-tuning the learning rate. From there, the right method can be selected to help the model converge faster, avoid overfitting or slow learning, and improve the accuracy of the training model.

The above approach not only enables the system to classify the operating status of vehicles in a more stable and reliable way, but also improves the efficiency of enterprises' fleet management.

The structure of the article consists of 4 parts:

(i) Introduction: the study aims to apply AI and IoT to monitor the condition of vehicles in real time, and at the same time find the most effective method of ANN training for this system.

(ii) Overview of CANBUS and system hardware: Collect real-time vehicle operation data through the CANBUS system via the OBD-II interface.

*Corresponding author. Email: minh.luu@ut.edu.vn

(iii) ANN Model Training Methods: Design and train an ANN model. Evaluate and compare two optimization weights for ANN training.
(iv) Conclusion: conclusions about the system, and future work of the study.

2. Overview of CANBUS and system hardware

2.1. On-vehicle CANBUS system

The vehicle's parameters are collected through the CANBUS (Controller Area Network BUS) communication network. It is a serial networking technology designed for embedded solutions, supporting microcontrollers and devices that can communicate with each other efficiently over a single pair of wires [3]. To obtain data on parameters from the vehicle's CANBUS system, communication modules can be used that connect directly to the vehicle's built-in OBD-II diagnostic port [4].

Figure 1. OBD-II Port

The CANBUS data frame provided by the vehicle follows the SAE J1979 standard format [5]. For example, with a data plan:
ID: 0x7E8 – Data: 0x04 0x41 0x0C 0x19 0x64 0xAA 0xAA 0xAA

From the above data frame: the response ID from the engine ECU is 7E8, the length of the frame consists of 4 valid bytes of data, the 0x41 mode responds to the current data viewing request, the PID 0C represents the motor speed (RPM) parameter, and the next two data bytes, $Byte_1$= 0x19 and $Byte_2$ =0x64, are used to calculate the physical value of the RPM according to the SAE standard formula J1979 [6]:

$$\text{RPM} = (Byte_1 * 256 + Byte_2)/4 \qquad (1)$$

2.2. System Hardware

The hardware used in this study includes a CANBUS Shield module and an Arduino microcontroller to establish a connection to the vehicle's CANBUS system via the OBD-II port.

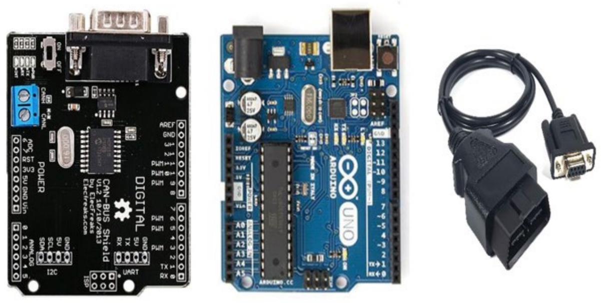

Figure 2. CANBUS Data Acquisition System via OBD-II Port

In addition, the system is also integrated with a 4G SIM module, which improves the ability to connect and exchange data between devices and central servers. With the advantage of high-speed mobile networks, the collected data can be transmitted continuously and stably without interruption. As a result, the system's scope of application is expanded, and the flexibility and efficiency of vehicle monitoring in a variety of real-world conditions are increased.

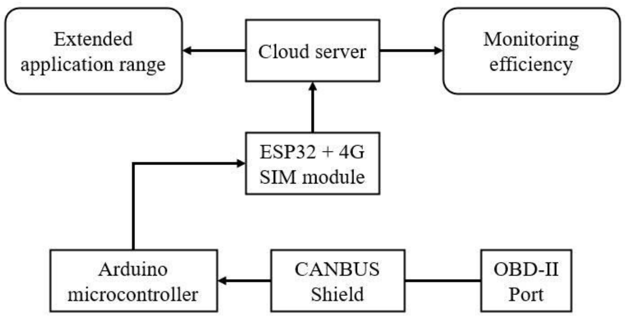

Figure 3. The system diagram

3. ANN Model Training Methods

3.1 System Process

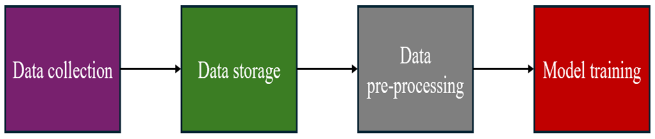

Figure 4. System Operation Process

Figure 4 shows the system implementation process, which consists of four steps: Collect vehicle data as a data source to train the AI model.

(i) Store data in Excel format for original data management, facilitating easy import into analysis tools.
(ii) Pre-process the data to select necessary parameters, standardize them on the same scale, and divide them into two sets: training (80%) and testing (20%).
(iii) Train the vehicle's condition classification model, issue warnings to help minimize incidents, and create maintenance plans.

3.2 ANN design

In this study, after processing CAN messages to convert the data into physical values for each vehicle parameter, an ANN model was trained on Google Colab [^1]. Specifically, the model is designed to classify input data into three groups: Normal, Warning, and Danger, using the SoftMax activation function [^2] at the output layer.

The data is stored in a CSV file and consists of five OBD-II parameters: Load, Mass Air Flow, Speed, RPM, and Throttle Position. It includes more than 27,000 samples collected from the field over approximately 26 hours through an IoT system. The raw data was originally recorded as text strings and contained wildcards representing measurement units (such as “%”, “g/s”, “km/h”, and “RPM”), as described in Table 1.

Table 1. The raw data

Load	Mass Air Flow	Speed	RPM	Throttle Position
60.00%	28.55g/s	27km/h	1994RPM	83.10%
60.00%	24.08g/s	27km/h	1994RPM	45.10%
63.10%	24.08g/s	27km/h	1994RPM	45.10%
63.10%	15.69g/s	28km/h	1355RPM	32.90%
63.10%	15.69g/s	28km/h	1355RPM	32.90%

The data pre-processing pipeline includes removing redundant characters, converting data to real numerical form, assigning labels, splitting the dataset, checking and standardizing data using MinMaxScaler for model training. The dataset is divided into an 80% training set and a 20% testing set.

The model architecture consists of three consecutive hidden layers with 32, 16, and 8 neurons, respectively. All three layers use the ReLU activation function to enhance nonlinear capabilities and efficient feature extraction [7]. Finally, the SoftMax output layer is responsible for classifying the data pattern into one of the vehicle's three operating states.

The model's classification criteria are evaluated at one time. If no more than one parameter exceeds the permissible threshold, it is classified as Normal. If two parameters exceed the threshold, they are classified in the Warning group. The remaining cases are included in the Danger group. The parameters' thresholds are set as follows:

$$\begin{cases} 30 \le Load \le 70 \\ 5 \le Mass\ Air\ Flow \le 17 \\ Speed \le 65 \\ RPM \le 2800 \\ 30 \le Throttle\ Position \le 70 \end{cases} \quad (2)$$

The above thresholds are determined based on practical experience, reflecting the stable operating status of the engine. Specifically, the engine operates at a moderate load, which helps reduce fuel consumption; the fuel system works stably; vehicle speed remains at an average level in urban areas; RPM is limited to reduce mechanical wear; and the throttle opening angle is within the optimal range, which is suitable for normal operating conditions.

The input setting consists of the values of five parameters, and the output comprises labels normalized to *to_categorical* (Normal: 1-0-0; Warning: 0-1-0; Danger: 0-0-1). Initially, training the model with 200 epochs and a batch size of 32 did not help it clearly distinguish between classes.

Two methods to optimize the model are proposed. The first is to detect the weight of each output label using the Optuna library, which involves hyperparameter optimization [8]. The second method is the learning rate adjustment method (Learning rate).

3.2.1. Weight detection method - Optuna

Optuna's search mechanism is based on the results of previous tests, building a probabilistic model that describes the relationship between the parameters and the output. Based on this model, Optuna predicts parameter regions that are likely to be more efficient, and then intelligently selects new values for the next attempt. This process is repeated until optimal results are achieved, or the set number of tests is reached [9]. In this study, the optimal criteria were selected as the F1 mean—an index that reflects the conditioned average of Precision (the reliability of the prediction) and Recall (the ability to fully detect the correct patterns) of the classification model, which helps to balance accuracy and the ability not to miss samples.

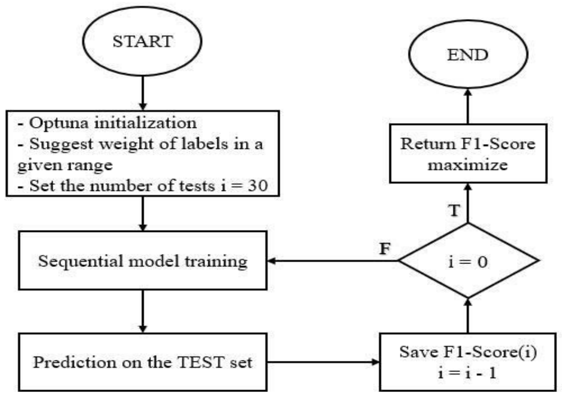
START
- Optuna initialization
- Suggest weight of labels in a given range
- Set the number of tests i = 30
Sequential model training
Prediction on the TEST set
Save F1-Score(i)
i = i - 1
F
i = 0
T
Return F1-Score maximize
END

Figure 5. Weighting detection algorithm flowchart

The process of formulating this weighted detection method is described according to the algorithm flowchart in Figure 5.

The following is the code part of the algorithm above:

```
def objective(trial):
    # Suggest values for class weights using Optuna's parameter search
    w0 = trial.suggest_float('w0', 0.6, 1.0)
    w1 = trial.suggest_float('w1', 0.6, 1.0)
    w2 = trial.suggest_float('w2', 1.0, 1.5)
    # Store the weights in a dictionary for training
    class_weights = {0: w0, 1: w1, 2: w2}
    # Initialize a Sequential neural network model
    model = Sequential([
        Dense(32, activation='relu', input_shape=(5,)),  # first hidden layer with 32 neurons
        Dense(16, activation='relu'),                   # second hidden layer with 16 neurons
        Dense(8, activation='relu'),                    # third hidden layer with 8 neurons
        Dense(3, activation='softmax')                  # output layer with 3 classes (softmax for classification)
    ])

    # Compile the model with optimizer, loss function, and evaluation metric
    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    # Train the model with training data, validation data, and class weights
    model.fit(
        X_train_sc, y_train,
        epochs=100, batch_size=32,
        validation_data=(X_test_sc, y_test),
        class_weight=class_weights,
        verbose=0   # suppress training output
    )
    # Predict on test data
    y_pred = model.predict(X_test_sc)
    # Convert predicted probabilities to class labels
    y_pred_class = np.argmax(y_pred, axis=1)
    # Convert true one-hot labels to class labels
    y_true_class = np.argmax(y_test, axis=1)
    # Return the macro-average F1-score as the optimization target
    return report['macro avg']['f1-score']
# Create an Optuna study to maximize F1-score
study = optuna.create_study(direction='maximize')
# Run optimization for 30 trials
study.optimize(objective, n_trials=30)
```

This method determined that the optimal weighting for the model was to reduce the Normal class to 0.68, reduce the Warning class to 0.87, and increase the Danger class to 1.06. After this fine-tuning, the neural network architecture was set up for the training model illustrated in Figure 6.

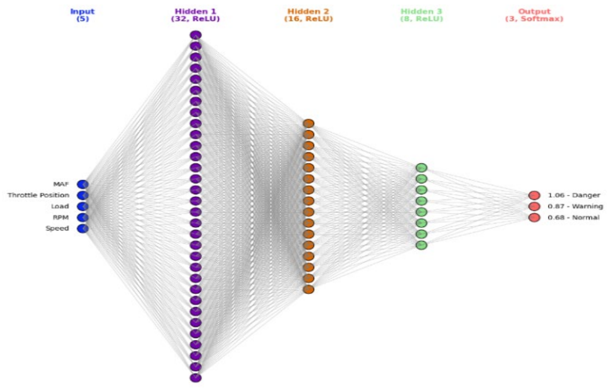

Figure 6. ANN Configuration After Adjustment

After setting the new weights for the layers, the predicted result is shown in Figure 7.

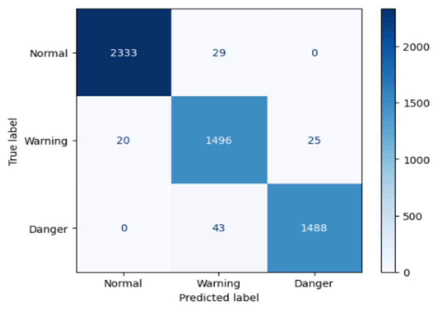

Figure 7. Predictive labels and True labels after weight adjustment

Table 2. Performance of the classification model after weight adjustment

	Precision	Recall	F1-Score	Support
Normal	0.99	0.99	0.99	2362
Warning	0.95	0.97	0.96	1541
Danger	0.98	0.97	0.98	1531
Accuracy			0.98	5434
Macro avg	0.98	0.98	0.98	5434
Weighted avg	0.98	0.98	0.98	5434

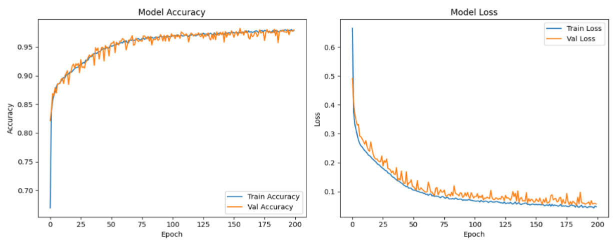

Figure 8. Accuracy and loss function of the model after weight adjustment

Results with an accuracy of up to 98% indicate stable and highly accurate predictions. The Precision, Recall, and F1-score indicators of the three classes (Normal, Warning, and Danger) all ranged from 0.95 to 0.99, proving that the model has greatly improved compared to the initial results. The Model Accuracy graph shows that the training and validation curves are almost identical, converging around the 0.97–0.98 level, indicating that the model learns well without overfitting. Similarly, the Model Loss decreases steadily with the number of epochs, indicating that the optimization process is efficient. Overall, the application of Optuna to adjust weights has helped the model achieve high performance, stability, and balance between data layers. This result demonstrates that automating the parameter selection process can result in a more optimal model than default or manual fine-tuning.

3.2.2. Learning rate adjustment method

Learning speed is one of the factors that directly affects the convergence speed and training quality of the model [10, 11]. The specified loss value is tracked on the training set. After five consecutive epochs without improvement, the learning speed is reduced to 80% of the original value, and the minimum value of the learning rate is set to 10-6 to avoid excessive degradation of the model's learning ability. Performing model training with the number of training rounds (epochs = 200) and the number of data samples per training round (batch_size = 32), the prediction results are depicted in Figure 9.

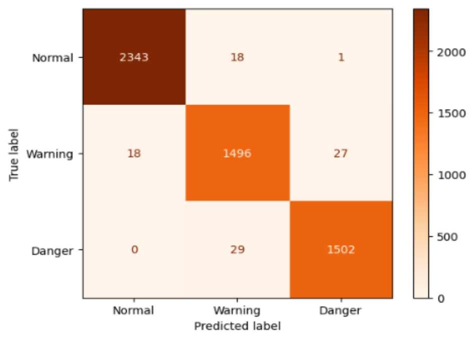

Figure 9. Predictive labels and True labels after learning rate adjustment

Table 3. Performance of the classification model after learning rate adjustment

	Precision	Recall	F1-Score	Support
Normal	0.99	0.99	0.99	2362
Warning	0.97	0.97	0.97	1541
Danger	0.98	0.98	0.98	1531
Accuracy			0.98	5434
Macro avg	0.98	0.98	0.98	5434
Weighted avg	0.98	0.98	0.98	5434

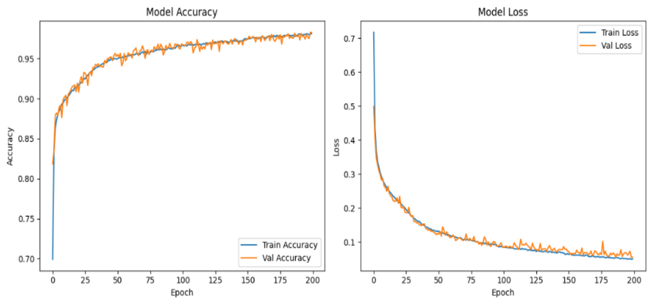

Figure 10. Accuracy and loss function of the model after learning rate adjustment

Statistical results show that accuracy is 98%, reflecting the model's accuracy. The Precision, Recall, and F1-score indices of each class all fluctuated between 0.97–0.99, indicating that the model maintains a balance between correct detection and limiting misclassification errors. The Model Accuracy and Model Loss charts illustrate the stable training process: The accuracy of both the training and testing sets increased steadily and converged around 0.97–0.98, indicating that the model learned well without overfitting. The loss curve decreased rapidly at an early stage and gradually stabilized to a small value (<0.1), indicating that the model had reached a state of convergence. Overall, adjusting the learning rate has helped the model improve generalization, especially with low-frequency data layers, while maintaining training stability and high performance across the entire dataset.

4. Conclusion

Through testing, the IoT system can operate in harsh environments with strong vibrations, such as in cars. It also boasts a fast data collection speed, with a sample extraction time of 300ms for each parameter. However, due to the use of 4G SIM cards, storing data in a private database may not ensure stability in locations where the telecommunications signal is weak or interfered with.

The two methods of adjusting weights using the Optuna library and adjusting the learning rate are both aimed at optimizing model performance. In terms of efficiency, both methods help the model achieve 98% accuracy. The Optuna-optimized model provides fast convergence and a good balance between data layers, while the re-learning rate adjustment model makes the training process smoother, reduces fluctuations at the end of the training phase, and maintains stability between training set and inspection set.

Experimental results show that the process of detecting weights with Optuna takes more than one hour, while adjusting the learning speed only takes about 20 minutes. Therefore, applying learning speed adjustment before weighting is considered a more efficient approach in terms of training time and resources.

The future work of this study is to apply these two models to the actual system to accurately evaluate their effectiveness and find a suitable model for diagnosing the vehicle's condition.

References

[1] U. Khadam, P. Davidsson, and R. Spalazzese, "Exploring the role of artificial intelligence in Internet of Things systems: A systematic mapping study," Sensors, vol. 24. 2024.

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, H. Arshad, "State-of-the-art in artificial neural network applications: A survey," Heliyon, 2018.

[3] C. Aciti, M. Urraco, E. Todorovich. "OBD-II vehicle data capture and monitoring system prototype," XXIV Congreso Argentino de Ciencias de la Computación 2018, Argentina, 2018.

[4] H. M. Luu, T. B. N. Nguyen, T. D. Nguyen, "Research on IoT application in controlling and monitoring the CANBUS system on cars," Proceedings of the 6th National conference on transportation science and technology 2025, Vietnam, Transport pulishing house, pp. 218-221,2025

[5] SAE International, "SAE J1979: E/E Diagnostic Test Modes," Surface Vehicle Standard, America, May 2007.

[6] S. Roksic, "Controller Area Network (CAN) Bus Simulator and Data-logger for In-Vehicle Infotainment Testing," Senior Project Report, Electrical Engineering Department, California Polytechnic State University, 2020.

[7] Y. Bai, "RELU-Function and Derived Function Review," SHS Web of Conferences, 2022.

[8] N. A. Rahmi, S. Defit, Okfalisa, "The Use of Hyperparameter Tuning in Model Classification: A Scientific Work Area Identification," JOIV: Int. J. Inform, Visualization, 2024.

[9] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," July 2019.

[10] Leslie N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay," USNaval Research Laboratory Technical Report 5510-026, April 2018.

[11] A. Crăciun, D. Ghoshdastidar, "On the Convergence of Gradient Descent for Large Learning Rates," Cornell University Library, 2024, doi: https://doi.org/10.48550/arXiv.2402.13108.