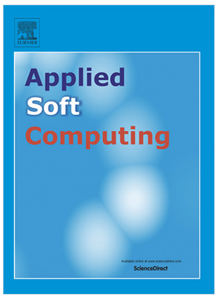


Event prediction algorithm using neural networks for the power management system of electric vehicles

Ki-sung Koo, Manimaran Govindarasu, Jin Tian

PII: S1568-4946(19)30490-9
DOI: https://doi.org/10.1016/j.asoc.2019.105709
Reference: ASOC 105709

To appear in: *Applied Soft Computing Journal*

Received date : 22 March 2019
Revised date : 8 July 2019
Accepted date : 9 August 2019

Event prediction algorithm using neural networks for the power management system of electric vehicles

Ki-sung Koo[a,*], Manimaran Govindarasu[a], Jin Tian[b]

[a]*Department of Electrical and Computer Engineering, Iowa State University, Iowa, USA*
[b]*Department of Computer Science, Iowa State University, Iowa, USA*

Abstract

The power management system for electronic vehicles selectively activates Electronic Control Units (ECUs) in the electronic control system according to time-series vehicle data and predefined operation states. However, at an operation state transition, the energy overheads used for the selective ECU activation could be higher than the energy saved by deactivating ECUs. To prevent these energy-inefficient state transitions, we apply two main ideas to our proposed algorithm: (A) unacceptable state transitions and (B) adaptive training speed. For the unacceptable transitions, our energy model evaluates the breakeven time where energy saving equals to energy overheads. Based on the breakeven time, our algorithm classifies training dataset as unacceptable and acceptable event sets. Especially when the algorithm trains neural networks for the two event sets, the adaptive training speed expedites its training speed based on a history of training errors. Consequently, without violating in-vehicle time constraints, the algorithm could provide real-time predictions and save energy overheads by avoiding unacceptable transitions. In the simulation results on real driving datasets, our algorithm improves the energy dissipation of the electronic control system by 5% to 7%.

Keywords: event prediction, neural networks, power management system, electric vehicles, neural network training, real-time systems, time-series data

1. Introduction

Electronic Vehicles (EVs) could be one of the most promising next-generation cars. First of all, the driving cost of EVs is lower than the Internal Combustion Engine (ICE) vehicles because EVs are much more efficient than the ICE vehicles. Based on the U.S. Department of Energy, gasoline vehicles only use 14% - 30% of fuel energy [1] for wheel rotation, while EVs utilize 74% - 94% of battery

*Corresponding author
Email addresses: kskoo@iastate.edu (Ki-sung Koo), gmani@iastate.edu (Manimaran Govindarasu), jtian@iastate.edu (Jin Tian)

energy [2]. Since the United Nations [3, 4] and European countries [5] regulate the emission of the ICE vehicles and plan to ban the use of gasoline and diesel cars, the ICE vehicles could be phased out and replaced with zero-emission EVs.

However, the current technologies of EV batteries support a relatively shorter driving distance and require a longer recharging time than gasoline and diesel cars. Although a few models have a driving range from 200 to 300 miles [6], typical EVs can only drive from 60 to 120 miles on a full charge. Recharging a battery pack usually takes between 4 and 8 hours [7]. To overcome the limitation of the current EV batteries, our research improves the energy consumption of automotive operations by enhancing the power management system for EV batteries.

The present power management system for EV batteries selectively activates necessary Electronic Control Units (ECUs) in the automotive's electronic control system based on time-series vehicle data and predefined operation states. By switching off ECUs that do not have a workload, the power management system can save the battery energy in electric vehicles when the vehicle operating system changes from one operation state to another. However, at an operation state transition, the operational energy used for the selective ECU activation could be higher than the energy that the power management system saves by deactivating ECUs.

Continuous energy-inefficient transitions rapidly increase the amount of minus energy balance between energy savings and energy overheads in the power management system. Inevitably, these state transitions quickly degrade EVs' energy efficiency. The state transitions motivate our research to enhance the power management system. In this paper, we prevent such transitions by applying two main ideas to our proposed algorithm: (A) unacceptable state transitions and (B) adaptive training speed. For the unacceptable transitions, our energy model evaluates the breakeven time where energy saving equals to energy overheads. Based on the breakeven time, our algorithm classifies training dataset as unacceptable and acceptable event sets. An unacceptable event is an energy-inefficient transition where the operational energy overhead is greater than energy savings because the changed state does not last for at least the breakeven time. When the algorithm trains neural networks for the unacceptable and acceptable event sets, the adaptive training speed expedites its training speed based on a history of training errors. Consequently, without violating in-vehicle time constraints, the algorithm could provide real-time predictions and save energy overheads by avoiding unacceptable transitions. In the simulation results on driving datasets [8, 9, 10, 11], our algorithm improves the energy dissipation of the automotive's electronic control system by 5% to 7%.

1.1. Relevant Work

There are two main concepts about the power management systems in automotive electrical/electronic architectures. First, partial networking only acti-

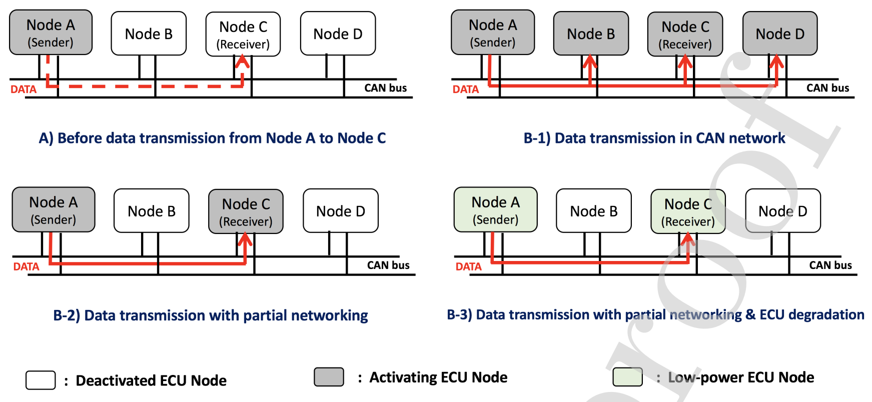

Node A (Sender)
Node B
Node C (Receiver)
Node D
DATA
CAN bus
A) Before data transmission from Node A to Node C
B-1) Data transmission in CAN network
B-2) Data transmission with partial networking
B-3) Data transmission with partial networking & ECU degradation
: Deactivated ECU Node
: Activating ECU Node
: Low-power ECU Node

Figure 1: Example of partial networking and ECU degradation on CAN (Controller Area Network) bus.

vates necessary ECUs in Controller Area Networks (CAN) [12] by using Wake-Up Frames (WUFs) [13] as shown in Figure 1. The concept derives from International Organization for Standardization (ISO) 11898-5 [14] and German car manufacturers [15]. Second, ECU degradation could reduce the energy consumption of ECUs by adjusting power-manageable components without violating performance requirements and constraints. Dynamic Power Management (DPM) [16, 17] provides a practical solution such as Adaptive Voltage Scaling [18] for the low energy consumption of ECUs. However, the existing methods do not take the energy overhead for the selective ECU activation into account. Notably, there is a problem when the energy saved by deactivating unnecessary ECUs is less than the energy overhead dissipated in turning off the ECUs.

To study about the energy imbalance between the energy savings and the energy overheads associated with power mode changes at the system-level, Lu [19, 20], Devadas [21], and Zhao [22] use the concept of *breakeven time*. Before switching off a subsystem in an embedded system, the power management system considers how much energy the subsystem consumes for turning on. Namely, if the power management system reactivates the subsystem too quickly after turning it off, the energy overheads of the power mode changes would offset the energy savings of deactivating the subsystem. Hence, the duration from the deactivation to the reactivation must be long enough to compensate for the operational energy overhead. The breakeven time refers to the energy balance point where the energy overheads of the mode changes become equal to the energy savings of the deactivated subsystem. Therefore, to make efficient power mode alterations, the power management system requires forecasting whether a deactivated period will be larger than the breakeven time or not.

Research about predicting a specific period against the breakeven time for the vehicle power management systems can be divided into two primary models: (A) the estimation models to calculate the breakeven time, and (B) the algorithm models to forecast whether a particular period is longer than the estimated breakeven time. Because the breakeven time depends on the energy overheads and the energy savings, the estimation models need to compute the energy consumption of both cases. Schmutzler [23] and Hong [24] assess the energy savings by adjusting the power mode of vehicle Electronic Control Units (ECUs) without affecting their performance. However, present estimation models regarding the energy consumption of ECUs do not evaluate the operational energy overhead used for ECU mode changes. In contrast, our model estimates both the energy savings and the energy overheads according to ECU operation procedures [25], ECU parameters [26, 27, 28, 29], in-vehicle network delays [30, 31, 32, 33], and network transceivers [34, 35].

The algorithm models predict events whose time series values remain above or below the threshold for at least the breakeven time. Weiss and Hirsh [36] proposed a system to predict events based on two steps: (A) identifying temporal and sequential patterns within time-series data, and (B) generating an ordered list of prediction patterns from step A. Xi and Song [37] suggested an algorithm that perceives unusual variations in time-series values and classifies them as events. Guralnik and Srivastava [38] approached the event problem as change-point detection in time-series data. The authors proposed two versions of the algorithms: the batch version receives all data before processing, whereas the incremental version processes new data points one at a time. Unlike other research works, the proposed algorithm in this study learns events according to the breakeven time and predefined event conditions. The algorithm predicts events by using the correlation between the learned events' data and current time-series data.

Power management using machine learning, existing methods train an agent for previous operation policies by collected data regarding workloads or user patterns. Based on the present data, the agent can select the best policy for the most energy savings. The authors in [39, 40, 41] use a supervised learning algorithm to discover an optimal policy by using performance data such as processor runtime statistics. Based on evaluating the performance of previous policies, the learning algorithm can choose the next policy. To maximize energy savings, the authors of [42] propose that the operating system learns which policy performs the best for the current workload by using machine learning. For recognizing workloads at runtime, the system records hard disk accesses and monitors I/O-related parameters. Jung and Pedram [43] present a supervised learning-based power management algorithm by Bayesian Classification. The algorithm learns to predict the system performance state from some readily available input features and then uses this predicted state to look up the optimal power management action from a precomputed policy lookup table.

The paper [44] shows a power management technique based on reinforcement learning. Without previous information, this learning algorithm dynamically adapts to the environment to achieve autonomous power management. However, in this paper, we use supervised learning instead of reinforcement learning. This is because we assume that the given inputs-outputs (time-series sensor data and vehicle operation states) are independent and identically distributed. More specifically, since vehicle sensor data are mostly changed by drivers and outside driving conditions, operation states made by the power management do not directly relate to variations of time-series sensor data. Consequently, we cannot use the power management-based feedback loop between time-series data and vehicle operation transitions for reinforcement learning.

1.2. Contribution

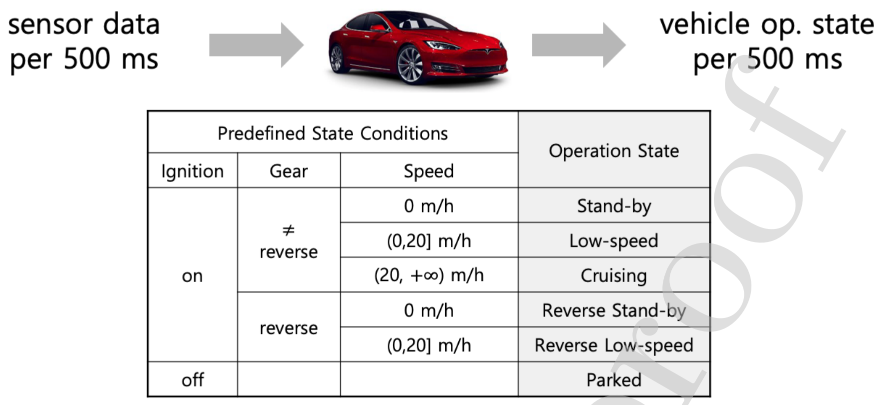
The contributions of this paper are fourfold:

1. We propose an energy model of an automotive's electronic control system. The model estimates the energy dissipation of vehicle operation states using parameters of Electronic Control Units (ECUs) and predefined state conditions.
2. We suggest a supervised learning algorithm based on neural networks to forecast whether the current vehicle state transition is energy-profitable or not.
3. We propose an adaptive training algorithm that tailors the network parameters in the neural networks based on the training history such that the network training can be done within the in-vehicle time constraint (0.1s).
4. We conduct a feasibility study on the proposed algorithm. To consider the in-vehicle communication cycles and delays, we evaluate the algorithm execution time as well as the average training error and the correct prediction rate in the feasibility study.

The rest of the paper is organized as follows: In Section 2, we will define a research problem and show our research object for our proposed algorithm. In Section 3, we will formulate the defined problem by using our system model. In Section 4, we propose to address unacceptable state transitions where operational energy overheads are higher than energy savings by developing a prediction algorithm. In Section 5, we will present evaluation results. Finally, Chapter 6 gives the conclusion.

2. Problem Statement

As shown in Figure 2, the vehicle operating system selects the current operation state based on the received sensor data and the predefined state conditions. When the vehicle speed rises or falls from a threshold value, the vehicle operating system changes the present operation state according to the speed variation.



Predefined State Conditions			Operation State
Ignition	Gear	Speed	
on	≠ reverse	0 m/h	Stand-by
		(0,20] m/h	Low-speed
		(20, +∞) m/h	Cruising
	reverse	0 m/h	Reverse Stand-by
		(0,20] m/h	Reverse Low-speed
off			Parked

Figure 2: Diagram of operation state selections based on sensor data and the predefined state conditions[24].

At the operation state transition, the power management system in the automotive operating system turns off the ECUs that do not have a workload. However, when the operation state changes many times during a short interval, the energy overhead used for reactivating the ECUs after the deactivation could be higher than the energy saved by switching off the ECUs. Because short-term frequent state transitions continuously consume more energy overheads than the energy savings, serial energy-inefficient transitions can radically lower EVs' energy efficiency. In this section, we define these state transitions as our research problem by using the concept of unacceptable state transitions and show our research object for our proposed algorithm.

2.1. Breakeven time (T_{be})

To justify operation state transitions with energy overhead, the power management system must consider the balance between energy consumed and saved in the state transitions before making the operation state alteration. More specifically, as the vehicle operating system changes from one operation state to another, the ECU without a workload becomes inactive as shown in Figure 3. The amount of energy savings is determined by the inactive duration (T_{sleep}) of the ECU. On the other hand, the energy overhead dissipated in the two ECU mode alterations depends on the ECU parameters. Therefore, to make energy-efficient operation state transitions, the power management system needs to forecast whether T_{sleep} is long enough to compensate for the energy overhead associated with the ECU mode changes.

Figure 4 shows the threshold time when the ECU mode alteration becomes profitable. If the inactive duration (T_{sleep}) is larger than the breakeven time (T_{be}), taking the operational energy overhead into account, the energy balance benefits from the ECU deactivation. To be specific, Figure 4 represents two different cases. One line increases linearly over T_{sleep} because the ECU continued

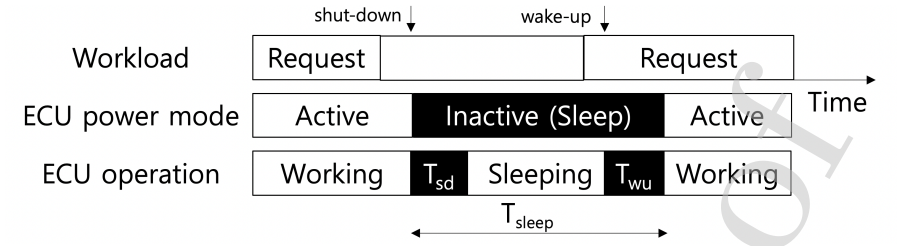


Figure 3: Power management from the workload, ECU power mode, and ECU operation points of view [20]. (T_{sd} and T_{wu} refer to shut-down delay and wake-up delay respectively.)

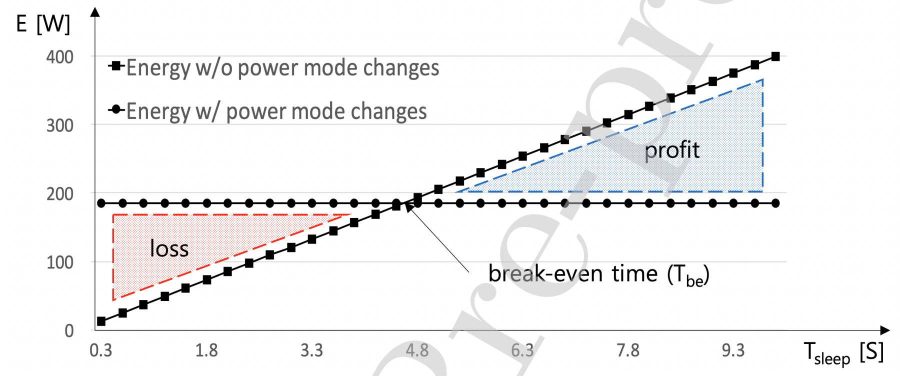


Figure 4: Breakeven time (T_{be}) for ECU mode changes [26]

to dissipate energy without power mode changes. The other line indicates ECU energy consumption when the ECU switches the power mode. The line value mainly expresses the energy overheads of both the deactivation and reactivation since the deactivated ECU consumes less than 1% of the energy overheads based on the ECU datasheet [26]. After the two lines cross as shown in Figure 4, the energy with ECU mode changes becomes less than the energy without ECU mode alterations. Consequently, the inactive duration (T_{sleep}) where the two lines intersect denotes the breakeven time (T_{be}) between energy with and without ECU mode changes.

T_{be} is calculated in terms of the parameters of an individual ECU. The left side in Figure 5 describes the energy consumption of an active ECU (E_{active}) according to active power (P_{active}). The right side defines the energy dissipation of an inactive ECU (E_{sleep}) and the energy overheads (E_{trans}= E_{sd}+E_{wu}) based on inactive power (P_{sleep}) and the time overhead (T_{trans}= T_{sd}+T_{wu}). When the two energy consumption values are equal, T_{be} is computed by using Equation 1.

$$T_{be} = \frac{E_{trans} - P_{sleep} * T_{trans}}{P_{active} - P_{sleep}} \tag{1}$$

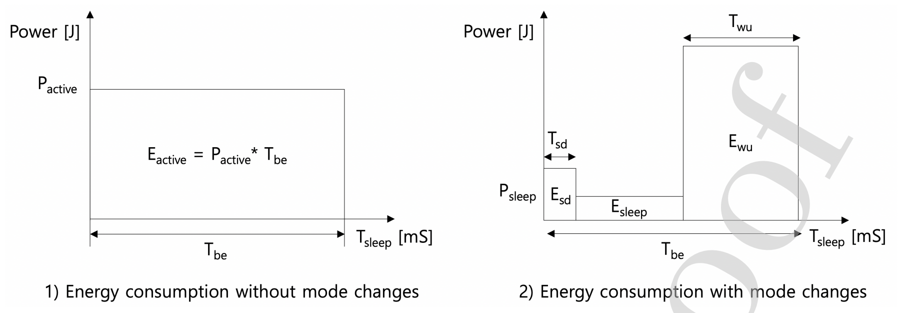

Power [J]
P_active
E_active = P_active* T_be
T_be
T_sleep [mS]
1) Energy consumption without mode changes
Power [J]
T_wu
T_sd
E_wu
P_sleep
E_sd
E_sleep
T_be
T_sleep [mS]
2) Energy consumption with mode changes

Figure 5: $T_{be} = T_{sleep}$ when the same energy consumption of two different cases [16, 20]

2.2. Unacceptable State Transition

As shown in Figure 6, when a vehicle enters a low-speed zone such as a roundabout, the driver will reduce the vehicle speed gently to less than 20 mph. Consequently, the vehicle operating system will make a state transition from cruising to low-speed. After passing the low-speed zone, the driver will recover the speed to more than 20 mph. Then, the vehicle operation system will make a second state transition from low-speed to cruising. As a result, ECU A will have an inactive duration (T_{sleep}) for 2 seconds between the two transitions. If T_{sleep} is less than the breakeven time (T_{be}), energy dissipation with the two transitions could be higher than energy consumption without the transitions. Namely, the operational overhead could be higher than the energy saved by the transitions. To avoid an energy-inefficient state transition where T_{sleep} is smaller than T_{be}, we define such a transition as an *unacceptable* state transition. Table 1 also shows that the ECU energy consumption with an unacceptable state transition is larger than the ECU energy dissipation without the state transition when the Figure 6 scenario is applied to an electronic control system model. In particular, consecutive unacceptable transitions can momently consume a large amount of energy overheads. As a result, these successive transitions can suddenly drop energy efficiency. The current state-based power management system cannot detect any unacceptable transitions and prevent them. This is because the power management system is not able to perceive T_{sleep} and T_{be}.

To prevent wasting energy overheads from unacceptable state transitions, our proposed algorithm determines whether the current state transition is acceptable or not based on comparing the forecasted inactive duration (T_{sleep}) with the breakeven time (T_{be}). Thus, if an unacceptable transition is predicted, the present operation state is retained without making a state transition. As shown in both Figure 6 and Table 1, by maintaining the current operation state, the state-based power management system with the proposed algorithm does not consume energy overhead.

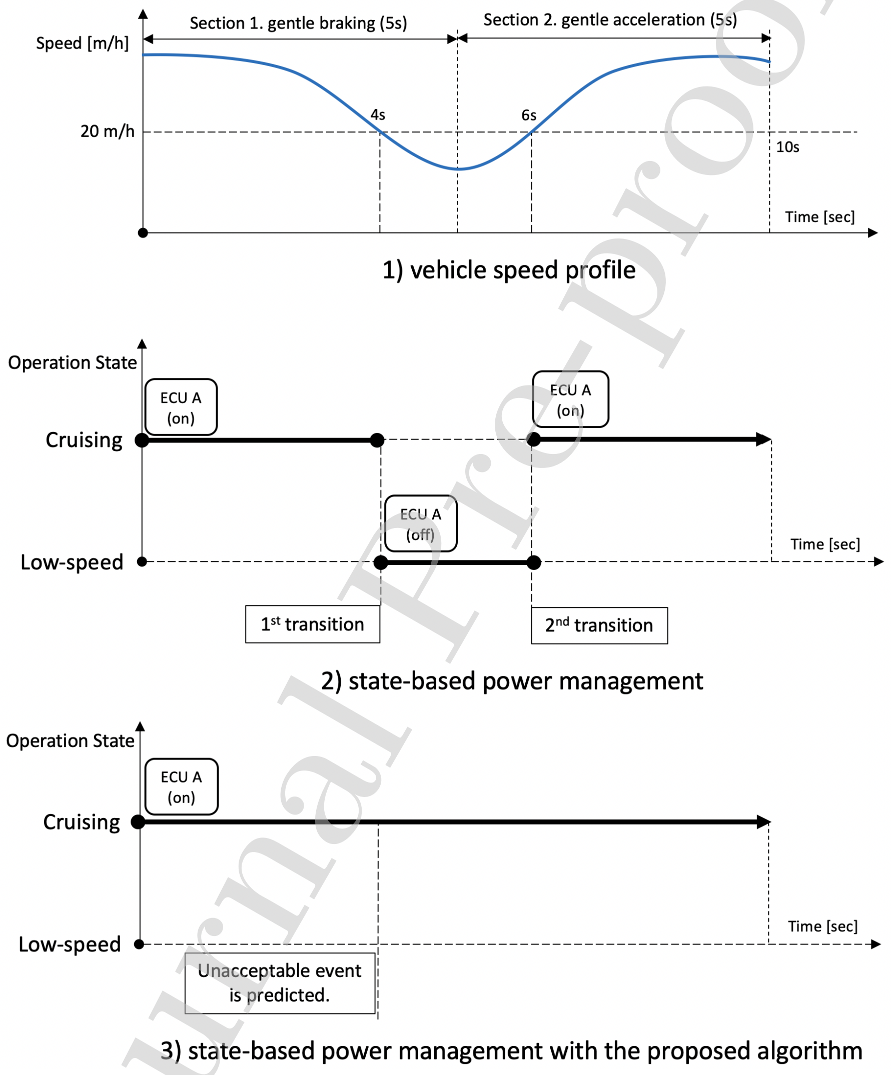


Figure 6: Example of an unacceptable state transition

	state-based power management	state-based power management with the proposed algorithm
ECU energy consump.	[cruising] 40W*2EA*8s [low-speed] 40W*1EA*2s [common] 40W*5EA*10s + 5.6W*2EA*10s + 0.9W*2EA*10s + 0.3W*25EA*10s = 2.9kJ	[cruising] 40W*7EA*10s + 5.6W*2EA*10s + 0.9W*2EA*10s + 0.3W*25EA*10s = 3kJ
Overhead	[3 ECU mode changes] 0.2kJ*3EA = 0.6kJ	
Total	3.5kJ	3kJ

Table 1: comparison of energy consumption by two different solutions (EA: each) [23, 24, 26]

3. Problem Formulation

The research objective is to overcome the limitation of electric vehicle (EV) batteries by minimizing the energy consumption of non-safety-critical Electronic Control Unit (ECU) applications[1] in the vehicle's electronic control system. To be specific, the power management system in the automotive operating system can make unacceptable state transitions that consume more overheads used for the ECU operations than the energy saved by turning off the ECUs in the electronic control system. Our proposed algorithm forecasts these unacceptable transitions and prevents them. Consequently, by saving energy overheads used for the unacceptable transitions[2], the algorithm improves EVs' energy efficiency.

To evaluate the actual energy cost of operation state transitions in the electronic control system, we formulate a model of the electronic control system.

[1]We bound our research scope to automotive applications in non-safety-critical ECUs because the proposed algorithm cannot accurately predict all state transitions and some mispredictions can be fatal to safety-critical ECU applications. More specifically, if the algorithm forecasts an unacceptable state transition as an acceptable transition, it is not a big problem because the automotive systems have this problem before the algorithm runs. However, we take account of the case when the algorithm predicts an acceptable state transition as an unacceptable transition. Because our prediction algorithm creates this problem and cannot avoid 100% of mispredictions regarding acceptable state transitions, we apply the proposed algorithm only to non-safety-critical ECU applications.

[2]Based on the parameters of ECUs[26, 27, 28, 29], the proposed algorithm can conserve the energy overhead (185J) per ECU by preventing an unacceptable transition with consuming about 0.1s and 0.01J as time and energy overheads for the algorithm execution[45]. Especially, since our vehicle model requires at most 0.4s for both ECU operations and data communications per instruction cycle (0.5s), the algorithm must complete the prediction within the time constraint (0.1s).

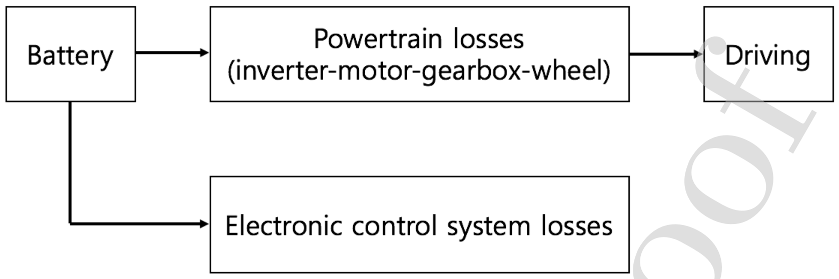

Battery
Powertrain losses (inverter-motor-gearbox-wheel)
Driving
Electronic control system losses

Figure 7: Energy consumption flow of an electric vehicle

Our model estimates the total energy consumption of all ECUs per state transition. In the model, the energy cost function h^* evaluates energy consumption of ECUs according to existing operation conditions. The other cost function c^* assesses energy dissipation of ECUs based on the breakeven time (T_{be}) as well as the existing conditions. Notably, at an unacceptable transition, c^* is smaller than h^*. This is because c^* does not consider the overheads used for the state transition when the predicted state duration (T_{state}) is less than T_{be}. By selecting the minimum cost between c^* and h^* per state transition, our model can avoid unacceptable state transitions. As a result, the model minimizes energy consumption of the electronic control system by saving overheads used for the unacceptable transitions.

3.1. Automotive Energy Consumption Model

The energy consumption model of EV batteries consists of two main parts: (A) the vehicle powertrain model, and (B) the electronic control system model. The primary energy flows from the EV batteries to each wheel through the vehicle powertrain as shown in Figure 7. The powertrain delivers about 70% of total battery energy to the driving with about 20% energy losses [46]. The electronic control system consumes the next most significant amount of vehicle battery energy. With high-end driving systems such as advanced driving assistance functions, the automotive's electronic control system can consume up to about 10%[3] of the vehicle battery energy[47].

3.2. Electronic Control System Model

To minimize the energy consumption of ECU operations in the electronic control system, we model the system by using the concept of the sequential decision process (SDP) [48]. Our SDP D estimates the energy cost of state-based ECU operations with and without consideration of condition $\mathbb{A}$ ($T_{state} \geq$

[3] In consideration of different electric cars, vehicle speed, and vehicle battery temperature except energy regenerated from braking, the total power consumption of vehicle electrical control units is about 2kW out of 20kW (total battery capacity of typical electrical vehicles).

T_{be}), and selects the better energy cost for energy-efficient state transitions. D is a finite automaton M with the cost functions $c*$ and h^* associated with state transitions in the electronic control system. M is described as $M = (Q, \Sigma, \delta, q_0, Q_F)$, where

- Q is a finite nonempty set of vehicle operation states;
- Σ is a finite input alphabet;
- $\delta : Q \times \Sigma \to Q$ is a state transition function;
- $q_0 \in Q$ is the initial state;
- $Q_F \subset Q$ is a set of final states.

A finite sequence of input alphabets is called an input string. Let Σ^* denote the set of all strings generated by Σ. δ is extended to δ^*: $Q \times \Sigma^* \to Q$ by using $\delta^*(q_0,\ \text{ab}) = \delta(\delta(q_0,\ \text{a}),\ \text{b})$ for ab $\in \Sigma^*$ and a & b $\in \Sigma$. $F(M) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in Q_F\}$ is the set of input strings accepted by the M.

According to the input string in $F(M)$, the sequential decision process D can evaluate the energy dissipation of a state transition. D is a system (M, h, ζ_0, c), where

- M is a finite automaton;
- $h : R \times Q \times \Sigma \to R$, where R denotes the set of real numbers, is an energy cost function without condition $\mathbb{A}$;
- $\zeta_0 \in R$ is the initial energy cost of q_0;
- $c : R \times Q \times \Sigma \to R$, is an energy cost function with condition $\mathbb{A}$.

Based on the extended transition function Σ^*, h can be expanded to $h^* : R \times Q \times \Sigma^* \to R$ by using $h^*(\zeta_0,\ q_0,\ \text{ab}) = h(h(\zeta_0,\ q_0,\ \text{a}),\ \delta^*(q_0,\ \text{a}),\ \text{b})$ for ab $\in \Sigma^*$ and a & b $\in \Sigma$. c can also be extended to $c^* : R \times Q \times \Sigma^* \to R$ by using $c^*(\zeta_0,\ q_0,\ \text{ab}) = c(c(\zeta_0,\ q_0,\ \text{a}),\ \delta^*(q_0,\ \text{a}),\ \text{b})$ for ab $\in \Sigma^*$ and a & b $\in \Sigma$. Because $F(M)$ is the set of accepted input strings, $h^*(\zeta_0, \delta^*(q_0, F(M)) \equiv h^*(\zeta_0, q_0, F(M))$, and $c^*(\zeta_0, \delta^*(q_0, F(M)) \equiv c^*(\zeta_0, q_0, F(M))$ are used for convenience. $G(D) = \{\delta^*(q_0, x) \in Q_F \mid c^*(\zeta_0, \delta^*(q_0, x)) = h^*(\zeta_0, \delta^*(q_0, x))\}$ represents the set of vehicle state transitions accepted by D.

$G(D)$ determines whether the current state transition is acceptable or not based on the SDP D. D evaluates the energy costs c^* and h^* regarding the present operation state transition according to received sensor data, Table 3, and condition $\mathbb{A}$. Then, D makes the state transition by comparing the cost c^* with the cost h^*. More specifically, the finite automaton M in D has three modules: an input interface, a read head, and a state controller. In one state transition, M reads the input symbol i_1, moves the read head one symbol to the right, and changes the current state q_0 to $\delta(q_0,\ i_1)$. When the read head moves

Table 2: list of inputs, operation states, and active ECUs [24]

input Σ^*	final operation state Q_F	active ECUs
i_0	Q_{F0} (Parked)	
$s_0g_0i_1$	Q_{F1} (Stand-by)	ECU 1, 2, 5, 6
$s_1g_0i_1$	Q_{F2} (Low-speed)	ECU 1, 2, 5-8
$s_2g_0i_1$	Q_{F3} (Cruising)	ECU 1-4, 6-8
$s_0g_1i_1$	Q_{F4} (Reverse stand-by)	ECU 5, 6
$s_1g_1i_1$	Q_{F5} (Reverse low-speed)	ECU 5

(i_0 : ignition off, i_1 : ignition on, g_0 : not reverse, g_1 : reverse
s_0 : 0 m/h, s_1 : (0, 20 m/h], s_2 : (20m/h, $+\infty$))

off the end of the input string, the accepted input string set $F(M)$ can give an input string to D. Then D estimates the final energy costs $c^*(\zeta_0,\ q_0,\ s_2g_0i_1)$ and $h^*(\zeta_0,\ q_0,\ s_2g_0i_1)$. Because the accepted transition set $G(D)$ requires that c^* is equal to h^* in the case of an acceptable state transition, only if $c^* = h^*$, M will change the operation state from q_0 to Q_{F3} based on Table 2.

3.3. Energy Cost Functions h^* and c^*

The energy cost functions h^* and c^* evaluate the energy dissipation of each ECU operation in the electronic control system per vehicle operation state. h^* represents the ECU energy consumption of the electronic control system according to the collected sensor data and the predefined ECU activation conditions (Table 3). Based on ECU power modes in [25], we model that h^* has three values: E_{sleep}, E_{active}, and E_{tran}. E_{sleep} indicates the energy consumption of an inactive ECU for the predicted inactive duration (T_{state}). E_{active} denotes the energy dissipated by an active ECU during T_{state}. E_{tran} is energy overhead used to switch off an active ECU or turn on an inactive ECU.

To consider unacceptable state transitions in modeling energy cost, we develop c^* from h^* with condition $\mathbb{A}$ ($T_{state} \geq T_{be}$). More specifically, c^* estimates the energy consumption based on the sensor data, the ECU activation conditions, and condition $\mathbb{A}$. Due to condition $\mathbb{A}$, c^* has different values from h^* at unacceptable state transitions. For example, if s^{th} state transition is predicted as an unacceptable transition, our model for electronic control system provides $h{*}^s$ and $c{*}^s$ for estimating energy consumption of all ECUs in the model at the s^{th} state transition.

For i^{th} ECU, when the given sensor data does not satisfy activation conditions, $h{*}^{is}$ shows E_{tran} for deactivating and reactivating the i^{th} ECU. On the other hand, $c{*}^{is}$ displays E_{active} for the i^{th} ECU at the s^{th} state transition. Since ${T_{state}}^s$ does not meet condition $\mathbb{A}$, the s^{th} state transition is forecasted as an unacceptable transition. Namely, $c{*}^{is}$ does not consider the s^{th} state transition as acceptable and energy overheads (E_{tran}) used for the s^{th} state transition. For all the ECUs in the model at the s^{th} state transition, $h{*}^s$ evaluates the total ECU consumption based on E_{active}, E_{sleep}, and E_{tran}. However, c^* estimates the total energy dissipation by using E_{active} and E_{sleep} without E_{tran}. This is

Table 3: ECU symbols, applications, and activation input conditions [24]

symbol	application	input conditions
ECU1	Adaptive Cruise Control	i_1, g_0, $\{s_0, s_1, s_2\}$
ECU2	Automatic Emergency Braking	i_1, g_0, $\{s_0, s_1, s_2\}$
ECU3	Lane Change Assistance	i_1, g_0, s_2
ECU4	High Beam Assistance	i_1, g_0, s_2
ECU5	Parking Assistance	i_1, $\{s_0, s_1\}$
ECU6	Tire Pressure Monitoring	a) i_1, g_0 b) i_1, g_1, s_0
ECU7	Traffic Light Recognition	i_1, g_0, $\{s_1, s_2\}$
ECU8	Electronic Stability Program	i_1, g_0, $\{s_1, s_2\}$

(The automotive electronic control model consists of 48 ECUs.
The other 40 ECUs are turned on when the vehicle ignites.)

because $c{*}^s$ verifies that the s^{th} state transition is unacceptable based on the ${T_{state}}^s$ and condition $\mathbb{A}$. In other words, $c{*}^s$ does not take account of any ECU mode changes at the s^{th} state transition.

3.4. Total Energy Consumption Minimization

The following descriptions and Table 4 show how our electronic control system model minimizes energy consumption. Based on the hypothesis, the breakeven time (T_{be}) derives from the the state duration (T_{state}) when the ECU energy consumption without the power mode changes becomes equal to the energy dissipation with the power mode changes. According to the predicted T_{state}, the calculated T_{be}, and the ECU parameters, our model estimates two kinds of total energy consumption (c^* and h^*) of the electronic control system per vehicle operation state. Generally, c^* and h^* have the same value. However, c^* is smaller than h^* when the model evaluates the energy dissipation of unacceptable state transitions. In the case of an unacceptable state transition, T_{state} does not satisfy condition $\mathbb{A}$. Based on the hypothesis regarding T_{be}, such T_{state} is not long enough to compensate for the sizeable operational overhead (E_{tran}) with the energy savings by deactivating the ECU. Because c^* uses condition $\mathbb{A}$ for energy estimation, c^* does not consider the unacceptable transition and becomes smaller than h^* due to the absence of E_{tran}.

Hypothesis:

$$P_{active} * T_{state} = E_{tran} + P_{sleep} * (T_{state} - T_{tran}) \ if \ T_{state} = T_{be}$$

Parameters:

$$P_{active},\ P_{sleep},\ E_{tran},\ T_{be},\ T_{tran}$$

Cost function $h*$:

$$h*^{is} = \begin{cases} E_{active}{}^{i} & if\ i^{th} \text{ ECU is active and the given inputs satisfy Table 3.} \\ E_{tran}{}^{i} & if\ i^{th} \text{ ECU is active and the given inputs do not satisfy Table 3.} \\ & if\ i^{th} \text{ ECU is inactive and the given inputs satisfy Table 3.} \\ E_{sleep}{}^{i} & if\ i^{th} \text{ ECU is inactive and the given inputs do not satisfy Table 3.} \end{cases}$$

$$E_{active}{}^{i} = P_{active}{}^{i} * T_{state}{}^{s}$$
$$E_{sleep}{}^{i} = P_{sleep}{}^{i} * T_{state}{}^{s},\ for\ i \in \mathbb{I},\ s \in \mathbb{S}$$

Cost function $c*$:

$$c*^{is} = \begin{cases} E_{active}{}^{i} & if \quad i^{th} \text{ ECU is active and the given inputs satisfy Table 3.} \\ & if \quad i^{th} \text{ ECU is active and the given inputs do not satisfy Table 3.} \\ & \quad\ \ T_{state}{}^{s} \text{ does not meet condition } \mathbb{A}. \\ E_{tran}{}^{i} & if \quad i^{th} \text{ ECU is active and the given inputs do not satisfy Table 3.} \\ & \quad\ \ T_{state}{}^{s} \text{ meets condition } \mathbb{A}. \\ & if \quad i^{th} \text{ ECU is inactive and the given inputs satisfy Table 3.} \\ & \quad\ \ T_{state}{}^{s} \text{ meets condition } \mathbb{A}. \\ E_{sleep}{}^{i} & if \quad i^{th} \text{ ECU is inactive and the given inputs satisfy Table 3.} \\ & \quad\ \ T_{state}{}^{s} \text{ does not meet condition } \mathbb{A}. \\ & if \quad i^{th} \text{ ECU is inactive and the given inputs do not satisfy Table 3.} \end{cases}$$

$$E_{active}{}^{i} = P_{active}{}^{i} * T_{state}{}^{s}$$
$$E_{sleep}{}^{i} = P_{sleep}{}^{i} * T_{state}{}^{s},\ for\ i \in \mathbb{I},\ s \in \mathbb{S}$$

Condition $\mathbb{A}$: $T_{state}{}^{s} \geq T_{be}$

$$T_{be} = \max_{\forall i \in \mathbb{I}} T_{be}{}^{i}$$

$$T_{be}{}^{i} = \frac{E_{tran}{}^{i} - P_{sleep}{}^{i} * T_{tran}{}^{i}}{P_{active}{}^{i} - P_{sleep}{}^{i}} \quad \text{if } i^{th} \text{ ECU is active at } (s-1)^{th} state.$$

$$i \in \mathbb{I},\ s \in \mathbb{S}, s \neq 1.$$

Goal:

$$\underset{T_{state}}{\text{minimize}} \sum_{i}^{\mathbb{I}} \sum_{s}^{\mathbb{S}} c*^{is}$$

$$subject\ to \sum_{i}^{\mathbb{I}} c*^{is} \leq \sum_{i}^{\mathbb{I}} h^{is},\ for\ i \in \mathbb{I},\ s \in \mathbb{S}$$

Table 4: Symbols and Meanings

symbol	meaning
ECU	Electronic Control Unit
active	ECU power mode that all elements inside ECU are fully functional
inactive	ECU power mode that all elements except memory are turned off
shut-down	power mode change from active to inactive
wake-up	power mode change from inactive to active
$h*^{is}$	model 1: energy consumption of i^{th} ECU at s^{th} state (transition)
$c*^{is}$	model 2: energy consumption of i^{th} ECU at s^{th} state (transition)
${E_{active}}^{i}$	energy consumption of i^{th} active ECU
${E_{sleep}}^{i}$	energy consumption of i^{th} inactive ECU (sleep mode)
${E_{sd}}^{i}$	energy consumption to shut down i^{th} ECU
${E_{wu}}^{i}$	energy consumption to wake up i^{th} ECU
${E_{tran}}^{i}$	total i^{th} ECU transition energy, i.e. ${E_{tran}}^{i} = {E_{sd}}^{i} + {E_{wu}}^{i}$
${T_{state}}^{s}$	duration from s^{th} state transition to the predicted next transition
${T_{be}}^{i}$	breakeven time for i^{th} ECU
${T_{sd}}^{i}$	transition time to shut down i^{th} ECU
${T_{wu}}^{i}$	transition time to wake up i^{th} ECU
${T_{tran}}^{i}$	total i^{th} ECU transition time, i.e. ${T_{tran}}^{i} = {T_{sd}}^{i} + {T_{wu}}^{i}$
${P_{active}}^{i}$	power consumption of i^{th} active ECU
${P_{sleep}}^{i}$	power consumption of i^{th} inactive ECU
$\mathbb{I}$	the set of all the electronic control units in the vehicle energy model
$\mathbb{S}$	the set of all the state transitions in a single driving cycle

For condition $\mathbb{A}$, the electronic control system model takes advantage of the largest T_{be}. When the automotive operating system selects the current operation state, all ECUs are active or inactive. Since the energy savings depend on the deactivated duration of active ECUs, the energy model computes the energy breakeven point (T_{be}) for the active ECUs in an operation state. More specifically, to make an energy-efficient state transition, the energy model requires verification that the predicted state duration (T_{state}) is longer than each T_{be} of all the active ECUs before making the state transition. Hence, the biggest T_{be} among the ECUs represents T_{be} in condition $\mathbb{A}$.

Our energy model minimizes the energy dissipation of the electronic control system by avoiding the consumption of operational overheads used for unacceptable state transitions. Namely, for an unacceptable state transition, total energy consumption of ECUs in the electronic control system without making the unacceptable state transition is less than total energy dissipation of overheads used for the ECU mode changes in the unacceptable transition. Therefore, by selecting the smaller energy estimation between the two cost functions $c*$ and $h*$ per vehicle operation state, the energy model accumulates the energy savings by avoiding unacceptable state transitions.

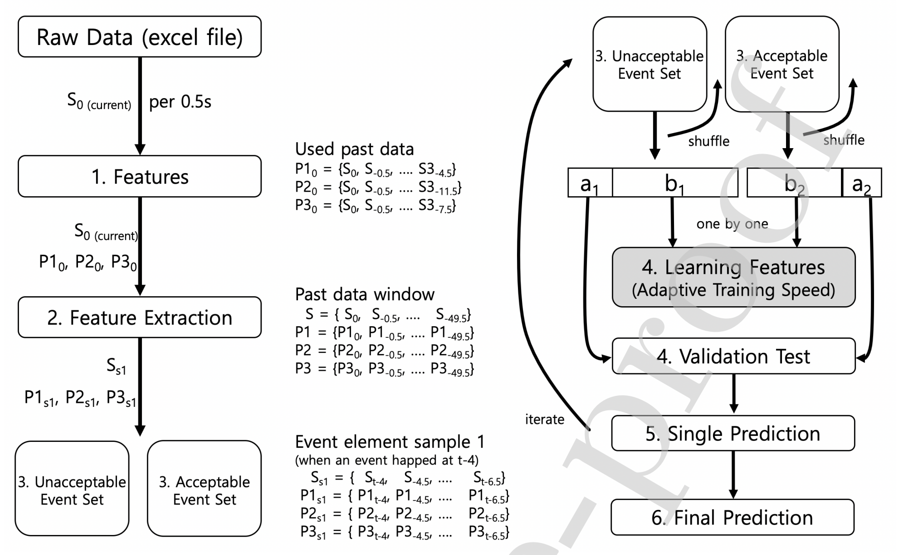

Raw Data (excel file)
$S_{0\,(current)}$ per 0.5s
1. Features
$S_{0\,(current)}$
$P1_0$, $P2_0$, $P3_0$
2. Feature Extraction
S_{s1}
$P1_{s1}$, $P2_{s1}$, $P3_{s1}$
3. Unacceptable Event Set
3. Acceptable Event Set
Used past data
$P1_0$ = {S_0, $S_{-0.5}$, ... $S3_{-4.5}$}
$P2_0$ = {S_0, $S_{-0.5}$, ... $S3_{-11.5}$}
$P3_0$ = {S_0, $S_{-0.5}$, ... $S3_{-7.5}$}
Past data window
S = { S_0, $S_{-0.5}$, ... $S_{-49.5}$}
P1 = {$P1_0$, $P1_{-0.5}$, ... $P1_{-49.5}$}
P2 = {$P2_0$, $P2_{-0.5}$, ... $P2_{-49.5}$}
P3 = {$P3_0$, $P3_{-0.5}$, ... $P3_{-49.5}$}
Event element sample 1
(when an event happed at t-4)
S_{s1} = { S_{t-4}, $S_{-4.5}$, ... $S_{t-6.5}$}
$P1_{s1}$ = { $P1_{t-4}$, $P1_{-4.5}$, ... $P1_{t-6.5}$}
$P2_{s1}$ = { $P2_{t-4}$, $P2_{-4.5}$, ... $P2_{t-6.5}$}
$P3_{s1}$ = { $P3_{t-4}$, $P3_{-4.5}$, ... $P3_{t-6.5}$}
3. Unacceptable Event Set
3. Acceptable Event Set
shuffle
shuffle
a_1
b_1
b_2
a_2
one by one
4. Learning Features (Adaptive Training Speed)
4. Validation Test
iterate
5. Single Prediction
6. Final Prediction

Figure 8: Diagram of the proposed algorithm for one event prediction (S_t and $P1_t$: the speed and the first preprocessed value at t-sec, S_{s1} and $P1_{s1}$: a sample set of time-series feature data regarding S_t and $P1_t$.)

4. Proposed Event Prediction Algorithm using Neural Networks

We propose to address unacceptable state transitions where the state period (T_{state}) does not meet condition $\mathbb{A}$ by developing a prediction algorithm. The proposed prediction algorithm forecasts an unacceptable state transition based on vehicle sensor data. Before making a state transition, the present sensor data provide crucial clues to determine whether T_{state} satisfies condition $\mathbb{A}$ or not. As shown in Figure 8, when the vehicle operating system receives data from multiple sensors per 0.5 seconds, the algorithm extracts features from the recorded data. The proposed algorithm searches past state transitions in the data and regards each previous transition as an event. If T_{state} is smaller than T_{be}, the algorithm defines the transition as an unacceptable event. As a result, there are features from both unacceptable and acceptable event sets. The algorithm (re)trains the neural networks for the event features. Especially, by adjusting the network parameters in the neural networks based on the training history, our algorithm can complete the network training within the given in-vehicle time constraint (0.1s). Then, the neural networks scan the event features associated with the current sensor data. According to the correlation between the present data and the past event features, the algorithm can predict the coming state transition. The proposed algorithm iterates from the feature extraction to the prediction per 0.5 seconds.

4.1. Features

The proposed algorithm takes advantage of time-varying indexes [49, 50] because the indexes are widely used for analysis and prediction of time-series data such as vehicle sensor data and stock price. As shown in Table 5, three preprocessed features (weighted moving average, stochastic D%, and relative strength index) are used as input values for the neural networks.

Table 5: Selected technical indexes and the formulas [49, 50]

Index	*Formulas*
P1. weighted 5-sec moving average	$\frac{5*S_0+...0.5*S_{-4.5}}{(5+...+0.5)}$
P2. 12-sec stochastic D%	$\frac{\sum(K_0+K_1+K_2)}{3}$
P3. 8-sec relative strength index	$\frac{AU}{AU+AD}*100$

(S_t: speed at t-sec, K_a: 4-sec stochastic K% about the a^{th} sample in last 12-sec speed data, $K_a=\frac{S_{-4a}-S_{L4}}{S_{H4}-S_{L4}}*100$ regarding the a^{th} sample ({ $S_{(-4a+0)}, S_{(-4a-0.5)}, ..., S_{(-4a-3.5)}$}), S_{L4}: lowest speed in the sample, S_{H4}: highest speed in the sample. For t={0, -0.5, ..., -7.5}, $if\ S_{(t-0.5)} > S_t \rightarrow$ Down = $(S_{(t-0.5)} - S_t)$, Average Down (AD) = $\frac{\sum Down}{n}$, $if\ S_t > S_{(t-0.5)} \rightarrow$ Up = $(S_t - S_{(t-0.5)})$, Average Up (AU) = $\frac{\sum Up}{n}$.)

4.2. Feature extraction

As shown in Figure 8, when the vehicle operating system receives a sensor value (S_0) per 0.5 seconds, the proposed algorithm generates the three features ($P1_0, P2_0, P3_0$) based on the previous sensor data. Our past data window stores the feature data including the sensor value and the generated features for last 50 seconds. The algorithm searches the data window for both previous state transitions and the feature data that were recorded before the transitions occurred because the algorithm assumes that the feature data can represent particular characteristics of past state transitions. For example, if the algorithm finds two serial state transitions in the past data window and if the duration between the transitions is less than the breakeven time (T_{be}), the algorithm regards the first transition as an unacceptable event. Based on when the event occurred, a set of time-series feature data for the last three seconds[4] becomes features of the unacceptable event as the event sample 1 in Figure 8. Consequently, the feature extraction produces the elements of both unacceptable and acceptable event sets to train the neural networks and to check the validity of the neural networks as shown in Figure 9.

[4]When the proposed algorithm trained the neural networks for the input-output relationship from the relationship between feature data and previous state transitions, we gradually reduced the size of time-series feature data from T_{be} to three seconds. The three-second was the longest size of feature data. Based on the sum of the training errors between the network outputs and the target outcomes for a certain period of time-series feature data, the algorithm adjusted the weight parameters in the neural network training. From three-second, the training errors could be stably reduced by performing the weight parameter adjustments.

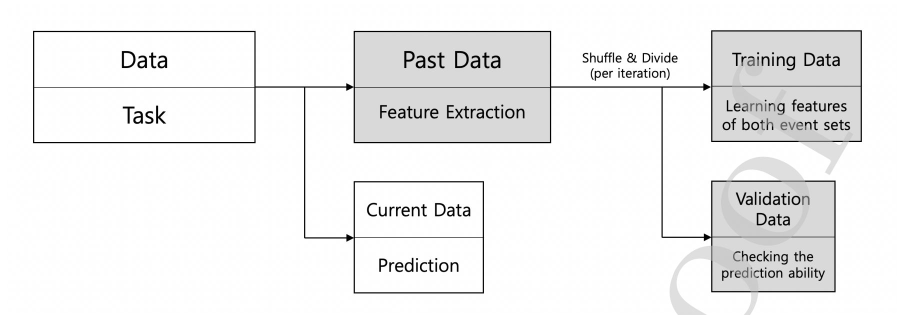

Data
Task
Past Data
Feature Extraction
Shuffle & Divide (per iteration)
Training Data
Learning features of both event sets
Current Data
Prediction
Validation Data
Checking the prediction ability

Figure 9: Data processing flow for training, validation, and prediction.

4.3. Shuffle the event element sequence[5]

To effectively train the neural networks in the proposed algorithm, the algorithm shuffles the element sequence of both the unacceptable and the acceptable event sets and trains the neural networks according to the rearranged element order. First, mixing the elements between old events and recent events provides fair learning opportunities to the algorithm. Because the newly created event elements are located at the end of the element sequence and have relatively less learning opportunities than the others, it can be more advantageous for the algorithm to minimize the training errors on the neural network training. Second, blending the elements between troublesome events and normal events can successfully save the training time. For example, when the vehicle speed is mostly zero during a long period of traffic jams, the features of collected events are also zero or very small. Namely, the network inputs are mostly zero. However, the target outputs are not zero. It is challenging for the algorithm to train the neural networks to make non-zero outputs by using the event element with zero network inputs. Consequently, the network training time for serial difficult elements is relatively longer than the training time for normal elements. Hence, we randomly rearrange the elements between laborious and general events and give more training time to the elements of general events.

4.4. Neural Network Training

Based on the reordered sequence of event elements, the proposed algorithm starts to train the neural networks. When the algorithm provides one event element to the neural networks, each element consists of time-series inputs and outputs. Figure 8 shows the inputs of an event element sample. The outputs are an operation state of Table 2 selected by the given inputs. However, the last output at the state transition is determined by whether the transition is

[5]The difference between the proposed algorithm and stochastic gradient descent is selecting event elements randomly without and with the replacement when an algorithm trains the neural networks. The difference is similar to the difference between permutation and combination.

unacceptable or not. More specifically, in the case of an unacceptable transition, the last output is the same as the previous output(s) because the unacceptable state transition should not occur.

Figure 10 illustrates the inputs and the output of two neural network models[6]. Both models have four network inputs $(x_{1t}, x_{2t}, x_{3t}, x_{4t})$ and one network output $(\widehat{y_{1t}})$. When the proposed algorithm offers an event element to both the inputs and the output of the Artificial Neural Networks (ANNs) in Figure 10-1, the neural network training starts. Each event element has four types of time-series inputs including one sensor value (S_t) and three preprocessed values $(P1_t, P2_t, P3_t)$, and one kind of time-series target outcomes (y_{1t}). After the algorithm maps from the inputs to the network inputs, the neural networks produce time-series network outputs $(\widehat{y_{1t}})$. To be specific, the feature extraction generates $S_t, P1_t, P2_t, P3_t,$ *and* y_{1t} for three seconds. The vehicle collects S_t per 0.5 seconds. As a result, one event element consists of four sets of six inputs (S_t, $P1_t$, $P2_t$, $P3_t$ for t={1,2, ..., 6}) and one set of six outputs (y_{1t} for t={1,2, ..., 6}). Hence, when the neural networks produce six network outputs ($\widehat{y_{1t}}$ for t={1,2, ..., 6}) based on the event element, the algorithm has six errors between the network outputs and the target outcomes. Based on the sum of the six errors, the algorithm adjusts the network parameters for the neural networks to learn event features regarding the given event element.

The network parameter adjustment begins to modify the output parameters[7] where they are located between the network output $(\widehat{y_{1t}})$ and the hidden neurons (h_{1t}, h_{2t}, ..., h_{8t}) by using algorithm 1 (discussed in Section 4.5) and the backpropagation algorithm. Then, the input parameters between the network inputs $(x_{1t}, x_{2t}, x_{3t}, x_{4t})$ and the hidden neurons are adjusted. After updating all the network parameters, the algorithm retrains the ANNs for the given inputs and recalculates the sum of the training errors between the network outputs and the target outcomes. Until the error sum reaches the threshold within the predefined training constraints[8], the neural network training is iterated. After

[6]To study the most suitable model of neural networks, the proposed algorithm trains artificial or recurrent neural networks with a single hidden layer because of limited execution time. More specifically, the algorithm needs to provide an event prediction per 0.5s and the neural network training takes the most algorithm execution in the proposed algorithm. Due to large execution times, our research cannot consider multiple hidden layers or complex neural networks such as Long Short-Term Memory.

[7]The range for initial output parameters is (-0.2, 0.2). Input and hidden parameters have their initial value between -0.1 and 0.1. Initial weight parameters in neural networks are randomly selected by using random number generators. As a result, the network training can begin with bad values. Because of the poor values, the algorithm could not pass a validation test and make an event prediction. However, the initial network training can improve the bad parameters for the next network training and even prediction. For example, as shown in Figure 13, adaptive training speed has the average training error lower than the threshold (1.5). Namely, the algorithm decreases training errors gradually on the network training.

[8]The maximum execution time and the maximum training number for (re)training the neural networks about one event element are 0.1s and 250. The threshold is 1.5.

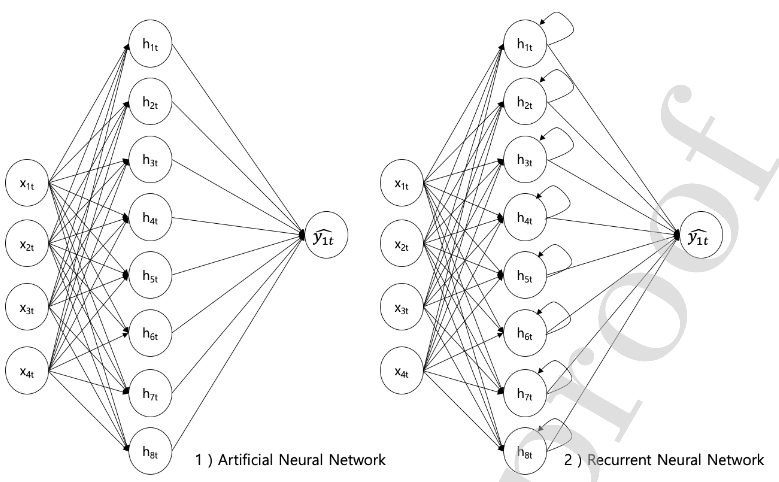


Figure 10: Models of Neural Networks (x: input neuron, h: hidden neuron, $\hat{y}$: output neuron)

completing the network training about the given event element, the algorithm provides the next event element to the ANNs according to the shuffled element order. The ANNs generate the network outputs based on the received event element. The algorithm repeats the network parameter adjustment according to the produced network outputs. In this way, once the parameter adjustment for the remaining elements is finished, the neural network training is completed.

To train Recurrent Neural Networks (RNNs), we take the network inputs of the hidden neurons (h_{1t}, h_{2t}, ..., h_{8t}) into account. As shown in Figure 10-2, each hidden neuron has one more input than artificial neural networks. Because RNNs assume that the previous output can become the present input, the hidden neurons send their output to both the network output ($\widehat{y_{1t}}$) and themselves. For instance, when the algorithm delivers the third input data $(S_3, P1_3, P2_3, P3_3)$ of an event element from training data set to the network inputs, the first hidden neuron (h_{13}) in the networks makes the third output based on the third network inputs $(x_{13}, x_{23}, x_{33}, x_{43})$ and the previous output of h_{12}. Then, the first hidden neuron (h_{14}) receives the fourth network inputs $(x_{14}, x_{24}, x_{34}, x_{44})$ and the third output of the first hidden neuron (h_{13}). After the RNNs generate the network outputs ($\widehat{y_{11}}$, $\widehat{y_{12}}$, ..., $\widehat{y_{16}}$) regarding the given event element, the proposed algorithm computes the training errors between the network outputs and the target outcomes (y_{11}, y_{12}, ..., y_{16}).

To minimize the training errors in the neural network training, the proposed algorithm modifies the weight parameters of the RNNs. By repeating the process of adjusting the parameters and reducing the errors, the algorithm (re)trains the RNNs to learn the event features about the given event element. More specifically, based on the sum of training errors, the algorithm adjusts from the output parameters between the network output ($\widehat{y_{1t}}$) and the hidden neurons

Algorithm 1 Adaptive Training Speed

```
Input: Initial parameter $\alpha, \beta, \mu$, $k$=0.
Input: Learning rate $l$, error history $h$, weight parameter $w$
  while stopping criteria not met do
    Compute the error by $k^{th}$ element ($x_k$, $y_k$) of event set:
    $L_k \leftarrow +L(\hat{y_k}(x_k; w_k), y_k)$
    Update the minimum training error:
    if $L_k < \mu$, $\mu \leftarrow L_k$
    Update the learning rate:
    $l_k \leftarrow \alpha l_{k-1} + (1-\alpha)L_k$
    Update the error history:
    $h_k \leftarrow \beta h_{k-1} + (1-\beta)L_k$
    Compute gradient estimate:
    $\hat{g_k} \leftarrow +\nabla_w L(\hat{y_k}(x_k; w_k), y_k)$
    Update weight parameter: $w_k \leftarrow w_k - l_k(h_k/\hat{g}_k)$
  end while
  Update element number: $k \leftarrow k + 1$
```

x: training input, y: target output, $\hat{y}$: training outcome,
L: error function, α = threshold/μ, μ: min. training error, $\beta = 0.1$

(h_{1t}, h_{2t}, ..., h_{8t}) to the weight parameters of the hidden neurons and the input parameters in order by using algorithm 1 and the backpropagation algorithm. For all the event elements, our algorithm iterates the parameter modification regarding one event element until the sum of training errors meets the minimum values within the training constraints[9].

4.5. Adaptive training speed

In the network parameter adjustment, the proposed algorithm considers the in-vehicle communication protocols. In particular, the vehicle operating system (OS) in our automotive research model selects the current operation state per 0.5 seconds according to real-time sensor data. Before the next sensor data are updated, the algorithm has to complete the training in 0.1 seconds. Based on the in-vehicle network protocols, the OS requires the maximum 0.4 seconds for ECU operations when the OS turns on an inactive ECU from the wake-up command to the ECU activation with consideration of the end-to-end communication delay[10]. To meet the limited training constraint (0.1s), our algorithm modifies the training speed by using the adaptive training speed when the proposed algorithm adjusts the weight parameters in the neural network training.

[9]The training constraints are the same as the ANN training, i.e. the maximum execution time (0.1s), the maximum training number (250), and the threshold (1.5s).

[10]In a worst case, the 0.4s derives from the wake-up control command (0.1s)[32], the ECU wake-up time (0.2s)[33], and the bus communication delay (0.1s)[31, 34, 35]

The main idea of the adaptive training speed is to accelerate the training speed based on training history. By applying previous results to the next training in the neural network training, the adaptive training speed could effectively escape the training errors from diverged or fluctuated situations in the network training and easily converge the errors to the threshold (1.5). To be specific, as shown in algorithm 1, the adaptive training speed begins with initial parameter α, β, μ, and k. α refers to a parameter used for updating the learning rate. α is computed by the ratio of the threshold to μ. μ is the minimum value of training errors in network training. As training errors are close to the threshold, α converges to 1. As a result, the learning rate is not changed. β is a constant used for updating the error history. Since the previous error history is getting less effective when the error history is exponential calculated, we set β for the previous error history as 0.1. k indicates the number of event element which algorithm 1 is currently used.

Next, each weight parameter[11] is updated by the three main parts: (A) the gradient of the training error, (B) the error history, and (C) the learning rate. For k^{th} event element $(x_k,\ y_k)$, the gradient of the training error $(\nabla_w L(x_k, y_k))$ provides the direction of the weight parameter adjustment. The ratio of error history (h_k) to $\nabla_w L(x_k, y_k)$ determines the amount that the weight parameter (w_k) is modified. The learning rate (l_k) controls how fast the weight parameter is changed based on the ratio. Both h_k and l_k are updated by the exponential moving average of training errors. Consequently, when the training errors regarding the k^{th} event element are not stabilized in series, the accumulated errors will rapidly increase h_k and l_k. The increased h_k and l_k, and $\nabla_w L$ could drastically reduce the next training errors to the threshold by adjusting the weight parameters in the networks.

4.6. Event Prediction

After the neural network training, the proposed algorithm verifies the validity of the neural networks for final event predictions. To be specific, when the event elements are shuffled, 70% of the rearranged elements are used for the network training. The remaining elements are utilized for the validation check. If the neural networks correctly output more than half of the events in the validation test, the algorithm applies the current sensor data to the neural networks for obtaining a final prediction. Especially, to prevent event predictions made by the discrepancy between past and current data sets, the algorithm iterates an event prediction several times as shown in Figure 8. Each prediction iteration including the network training and the validation test is executed by using different training and validation datasets after shuffling the previous datasets.

[11] Initial parameters are randomly selected by using random number generators. The range for initial output parameters and for both initial input and hidden parameters are (-0.2, 0.2) and (-0.1, 0.1) respectively.

Table 6: Descriptions of Training Algorithms in the Feasibility Study

	Step 1	Step 2
Adaptive Training Speed	$l_t \leftarrow \alpha l_{t-1} + (1-\alpha)L_t$ $h_t \leftarrow \beta h_{t-1} + (1-\beta)L_t$	$w_{t+1} \leftarrow w_t - l_t \frac{h_t}{\nabla_w L_t}$
Adaptive Moment[51]	$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\nabla_w L_t$ $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)(\nabla_w L_t)^2$	$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{(\frac{v_t}{1-\beta_2^t})+\epsilon}}(\frac{m_t}{1-\beta_1^t})$
RMSProp[52]	$G_t \leftarrow \gamma G_{t-1} + (1-\gamma)(\nabla_w L_t)^2$	$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{G_t+\epsilon}}\nabla_w L_t$
Adaptive Gradient[53]	$G_t \leftarrow G_{t-1} + (\nabla_w L_t)^2$	$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{G_t+\epsilon}}\nabla_w L_t$
Gradient Descent		$w_{t+1} \leftarrow w_t - \eta\nabla_w L_t$
Momentum[55]	$m_t \leftarrow \gamma m_{t-1} + \eta(\nabla_w L_t)$	$w_{t+1} \leftarrow w_t - m_t$
Adaptive Delta[54]	$G_t \leftarrow \gamma G_{t-1} + (1-\gamma)(\nabla_w L_t)^2$	$w_{t+1} \leftarrow w_t - \frac{\sqrt{s_t+\epsilon}}{\sqrt{G_t+\epsilon}}\nabla_w L_t$

(Note that Adaptive Delta updates s_{t+1} by using $s_{t+1} \leftarrow \gamma s_t + (1-\gamma)(\frac{\sqrt{s_t+\epsilon}}{\sqrt{G_t+\epsilon}}\nabla_w L_t)^2$.
For t^{th} training, l_t: learning rate, h_t: error history, w_t: network weight parameter, L_t: training error, m_t: moment term, v_t: 2nd moment term, G_t: gradient history. Parameter $\alpha, \beta, \beta_1, \beta_2, \gamma, \eta \in (0,1)$)

The algorithm collects these predictions and makes a final event prediction according to the event that more than half of the collected predictions indicate. Namely, if the predictions do not show a certain event, the algorithm will not make a final prediction.

4.7. Feasibility study

To assess the practical performance of algorithm 1, we conduct a feasibility study about algorithm 1 from four perspectives: (A) correct prediction rate, (B) average execution time, (C) average training errors, and (D) convergence rate. Based on these four perspectives, we verify that algorithm 1 meets the requirements for a solution to the research problem. Next, by comparing the four types of performance evaluation between algorithm 1 and alternatives, we study the advantages and the disadvantages of algorithm 1.

For the alternatives, we use a prediction algorithm using neural networks with first derivative-based methods as shown in Table 6: adaptive momentum[51], RMSprop[52], adaptive gradient[53], gradient descent, adaptive delta[54], and momentum[55]. Since driving conditions are dynamically changed, the current event is not always correlated with past events. Thus, the prediction algorithm using neural networks can forecast unacceptable state transitions only if the algorithm discovers strong relevance between past and current event data. Accordingly, algorithms that forecast events by generalizing past events such as clustering, decision boundary, and regression analysis cannot effectively predict unacceptable transitions. For example, in this study, a logistic regression method provides inconsistent event predictions per performance evaluation. Next, due to limited execution times, we do not consider multiple hidden layers and high-order methods such as conjugate gradient and Levenberg-Marquardt algorithms in the feasibility study.

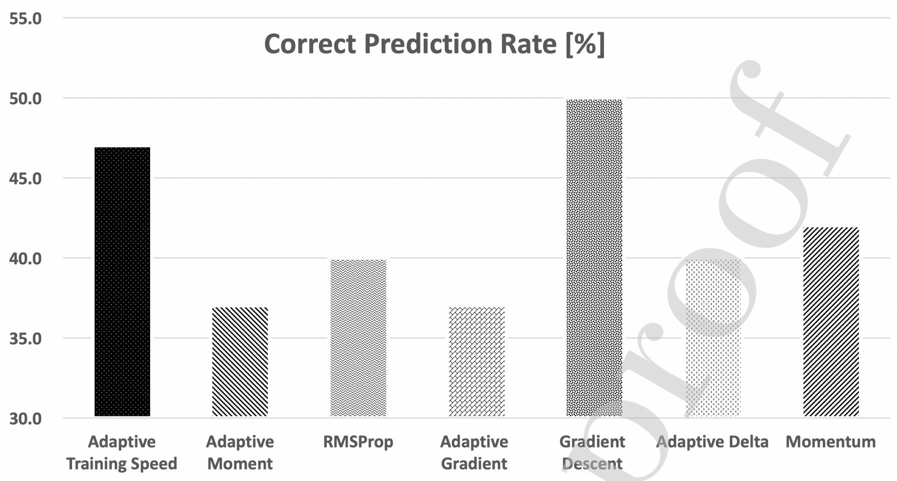


Figure 11: Benchmark test of correct prediction rates

For evaluating correct prediction rates, we define the rates based on the percentage of unacceptable transitions that each algorithm predicts from the evaluation driving cycle (EU in Table 8). Figure 11 shows that the gradient descent has the highest predictability among the training algorithms in the feasibility study. Algorithm 1 (adaptive training speed) comes in the second[12]. To check these algorithms' availability for vehicle ECU operations with in-vehicle time constraints, we consider the average execution time. To be specific, if these algorithms consume more execution time than the system's instruction cycle (0.5s), the predicted events have already occurred when the algorithms forecast events. Consequently, the system cannot use the algorithms for predicting events. To check this validity for these algorithms as well as the others in the feasibility study, we measure the average execution time.

For the average execution time in Figure 12, we measure each execution time when algorithms predict a state transition. As shown in Figure 8, to make a final prediction, an algorithm takes steps from the feature extraction to the final prediction when our vehicle model collects new data. We measure the algorithm's execution times for this duration and average them. Based on the average execution time with the instruction cycle (0.5s) and the system

[12]Because unacceptable state transitions is closely related to the energy savings in vehicle ECU operations, we consider the unacceptable transitions when we define the correction prediction rates. In the case of the correct prediction rate for acceptable state transitions, the results are as the following: Adaptive Training Speed (75%), Adaptive Moment (89%), RMSPRop (82%), Adaptive Gradient (75%), Gradient Descent (71%), Adaptive Delta (75%), and Momentum (75%). In the driving cycle, both unacceptable and acceptable transitions are randomly distributed. The total number of unacceptable and acceptable transitions in the benchmark test are 40 and 28.

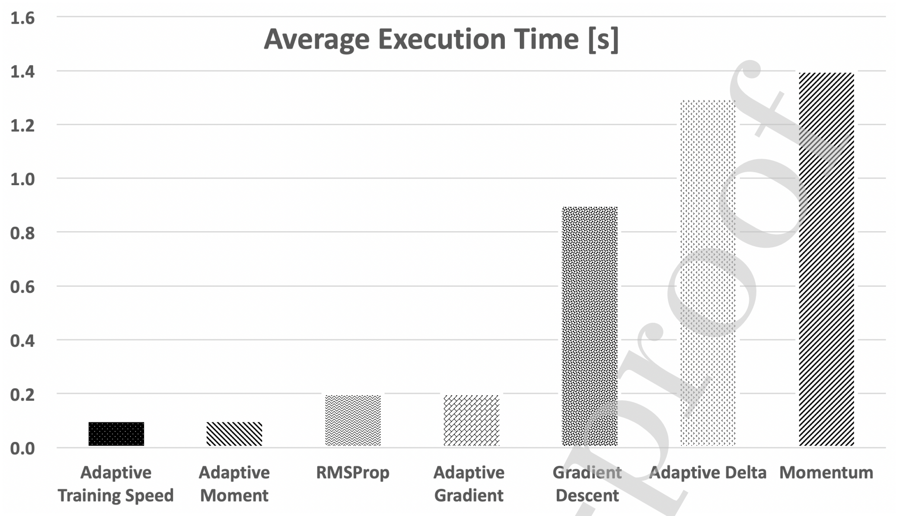


Figure 12: Benchmark test of average execution times
(execution time: total time from the feature extraction to the final prediction)

overheads (0.4s)[13], we can check the validity of algorithms. As shown in Figure 12, both algorithm 1 (adaptive training speed) and adaptive moment can be used for providing valid predictions to vehicle ECU operations. Since the vehicle OS requires at most 0.4s for ECU operations per the instruction cycle, algorithms have the maximum 0.1s for predicting a state transition. In Figure 12, all algorithms except adaptive training speed and adaptive moment consume more than 0.1s used for predicting an event. Thus, because of their large execution times, our model can use predictions made only by both adaptive training speed and adaptive moment.

To verify a training performance of algorithms with the average execution time in Figure 13, we collect values of training error. When an algorithm trains neural networks for a given element, the algorithm will reduce the training errors between the network outputs and the target values. If the algorithm effectively competes the neural network training, the training errors will be gradually decreased. For example, algorithm 1 (adaptive training speed), adaptive moment, RMSPRop, and adaptive gradient in Figure 13 have relatively smaller variations of training error than the others. Namely, when these algorithms train neural networks, their training errors have continuously reduced. Besides, with consideration of the average execution times in Figure 12, we can verify the algorithms' training performance. For example, algorithm 1 has the minimum

[13]It is the worst case's delay from the wake-up control command (0.1s)[32], the ECU wake-up time (0.2s)[33], and the bus communication delay (0.1s)[31, 34, 35].

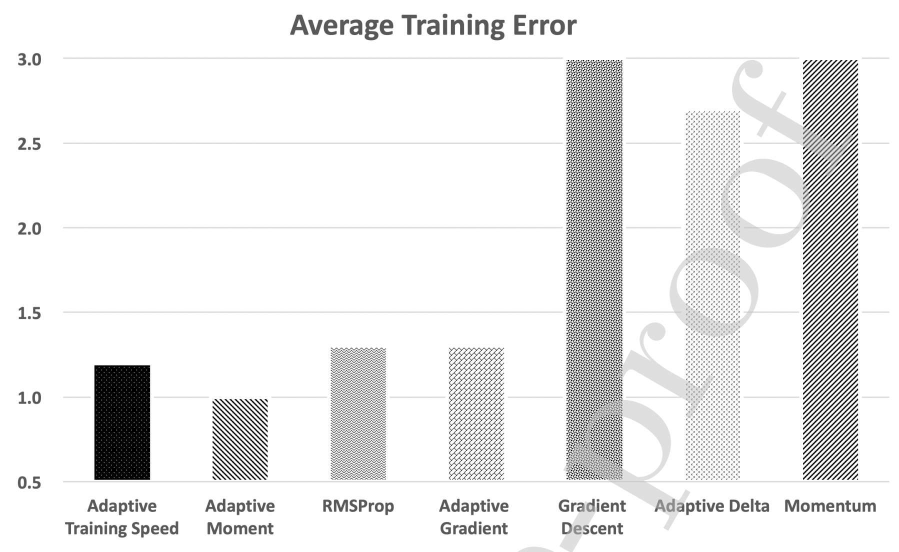


Figure 13: Benchmark test of average training errors

Table 7: Types of the Algorithm convergence rate

	convergence rate type
Adaptive Training Speed	linear
Adaptive Moment	linear[58]
RMSProp	linear[58]
Adaptive Gradient	linear [56]
Gradient Descent	sublinear [57]
Momentum	linear, sublinear [57]
Adaptive Delta	linear [59]

execution time and the second smallest average of training error. These values indicate that algorithm 1 takes the fewest steps for neural network training without big fluctuating or diverging cases.

Lastly, to evaluate the training speed of algorithms when the error function L is convex, we study the algorithm's convergence rate. To be specific, when the k^{th} training error L_k is defined by the difference (x_k-x^*) between the network output x_k and the target outcome x^*, the convergence rate comes from the ratio of L_{k+1} to L_k by increasing the k to the infinity. Based on the reference papers [56, 57, 58, 59], Table 7 shows the convergence rate type of each algorithm. Most algorithms including algorithm 1 (adaptive training speed) have a linear convergence rate.

Table 8: Descriptions of Actual Driving Data used in the Evaluation

Symbol	EU[10]	US1[8]	US2[9]	OZ[11]
Data subject	Urban driving cycle, European Artemis	New York City Cycle (NYCC)	EPA Urban Dynamometer Driving Schedule (UDDS)	Light duty petrol vehicle emissions testing
Duration	16 min.	10 min.	23 min.	30min.
Distance	3 mile	1.2 mile	7.45 mile	12 mile
Average speed	28.3 m/h	7.1 m/h	19.6 m/h	60.6 m/h
No. of unacceptable state transitions	40	27	36	24
No. of acceptable state transitions	28	21	36	23

5. Evaluation

The primary evaluation purpose is to determine how much the proposed algorithm with algorithm 1 (adaptive training speed) can improve the energy consumption of the electronic control system with consideration of time overheads and the algorithm side-effects. First, since the total energy consumption is enhanced by predicting unacceptable transitions and preventing them, we start to evaluate how many unacceptable transitions the proposed algorithm forecasts. Second, since our research model cannot use predictions that the proposed algorithm executes for more than 0.1s, we measure each execution time to check the validity for the algorithm's predictions and average them. Third, we calculate energy improvement by using two types of energy consumption. When we apply an evaluation scenario to our research model, we measure the total energy consumption of the model without and with using the proposed algorithm. Based on the difference between the two energy consumptions, we estimate how much energy the algorithm improves.

When we evaluate the performance of the proposed algorithm with algorithm 1 from the three perspectives, we compare the results to the performance of the algorithm with gradience descent. Based on the feasibility study, only gradient decent had better prediction performance than algorithm 1. To further performance comparison between algorithm 1 and gradient descent, we evaluate these algorithms from the three perspectives with two neural network models (artificial and current neural networks in Figure 10) and different evaluation datasets. For the evaluation datasets, we use four actual driving scenarios that Europe, America, and Australia utilize for the automotive performance tests. Table 8 gives the particular information about the four scenarios.

5.1. Correct prediction rate

The proposed algorithm predicts unacceptable or acceptable state transitions. Since the algorithm cannot correctly predict all the vehicle state transitions, we evaluate two types of correct prediction rate in the evaluation. First, we estimate the correct prediction rate about unacceptable state transitions based on the ratio of the forecasted unacceptable transitions to the total number of unacceptable transitions. By predicting unacceptable transitions and preventing them, the algorithm saves the energy consumption of our system model. Namely, the correct prediction rate about unacceptable transitions directly relates to the amount of energy consumption saved by the algorithm.

Second, we evaluate the correct prediction rate about acceptable state transitions. If the algorithm mispredicts an acceptable state transition as an unacceptable state transition, the algorithm creates a side effect. Because our system does not make the mispredicted unacceptable transition (actually acceptable transition), the energy consumption without the state transition consumes more energy than the energy dissipation with the state transition. As seen in Figure 4, since the state duration (T_{sleep}) for the acceptable transition is longer than the breakeven time (T_{be}), the energy saved by the state transition is larger than the operational energy used for the state transition. To estimate the adverse effect of these mispredictions about acceptable state transitions, we estimate the correct prediction rate based on the number of the predicted acceptable transitions.

As shown in Figure 14, there are 16 test cases according to two training algorithms, two neural network models, and four driving scenarios. Each test case has two types of correct prediction rate: (A) the acceptable state transitions and (B) the unacceptable state transitions. Notably, the correct prediction rates about the acceptable transitions are mostly higher than the correct prediction rates regarding the unacceptable transitions. When the proposed algorithm cannot forecast a state transition based on given vehicle data, the algorithm predicts an acceptable state transition as a default prediction. Consequently, the prediction results about the unacceptable state transitions are relatively fewer. That is, the correct prediction rates of the unacceptable transitions are less than the other.

For the correct prediction rates about acceptable transitions, most rates are more than 70% as shown in Figure 14. Namely, the misprediction rates about the acceptable transitions are less than 30%. Based on the misprediction rates, we conclude that the given algorithm cases do not produce significant side-effects to our system model. For further verification, we check practical effects of the mispredictions about acceptable transitions when we evaluate the energy improvement rates in Figure 16.

For the correct prediction rates about unacceptable transitions, the rates in Figure 14 are generally between 30% and 60%. Since an algorithm with artificial neural networks (ANNs) has a higher prediction rate than the same algorithm with recurrent neural networks (RNNs), ANNs is more effective for predicting unacceptable transitions than RNNs. Especially, the gradient descent with ANN has the highest predictability for all driving scenarios. Adaptive

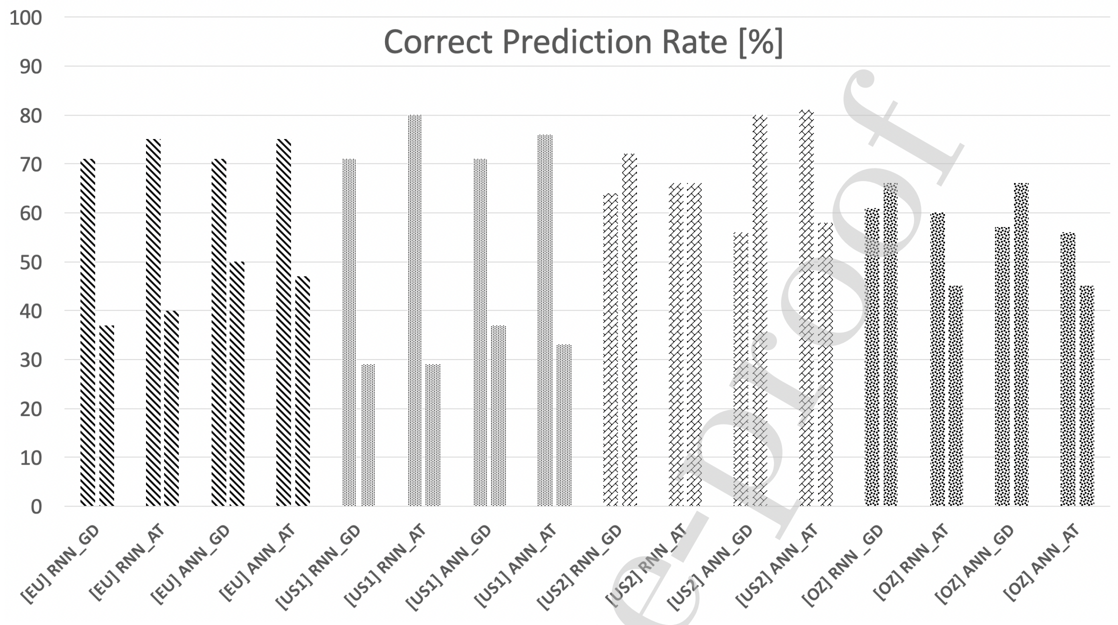


Figure 14: Comparison of correction prediction rates of acceptable (left) and unacceptable (right) transitions (EU: Europe, OZ: Australia, GD: gradient descent, AT: Adaptive Training Speed, ANN: Artificial Neural Networks, RNN: Recurrent Neural Networks)

speed training with ANNs produces the second highest performance. Namely, like the feasibility study, gradient descent shows better prediction performance than adaptive training speed. To further the performance comparison between the gradient descent and the adaptive training speed with the time constraint, we measure the algorithms' execution times in Figure 15.

In this study, the following issues limit the proposed algorithm to achieve high prediction rates. First, dynamic driving environments make the goal more difficult. The proposed algorithm predicts events based on the events that the algorithm previously experienced. That is, if the algorithm meets a new event that is not relevant to the experienced events, the algorithm cannot predict the event. The second constraint for the high prediction rates is the randomness of events. According to the analysis of our prediction results, even though the algorithm predicts an event based on high probabilistic relevance between the present event and the previous events, the same past event does not always occur due to event randomness[14].

[14]Based on the analysis of Student's t-test regarding the evaluation data, about 20% of both acceptable and unacceptable transitions have similar prediction clues. As a result, a prediction can be wrong even though the prediction is made by deterministic clues.

5.2. Average execution time

To check the validity of an algorithm with in-vehicle communication protocols, we measure each execution time when the algorithm predicts an event based on newly-collected sensor data and average them. In our research model, the vehicle OS selects an operation state per 0.5s according to the new sensor data. In the instruction cycle (0.5s), since the vehicle OS requires at most 0.4s[15] for completing ECU operations, the proposed algorithm can have at most 0.1s for predicting a state transition. Thus, based on comparing the average of collected execution times with this 0.1s, we can verify whether an algorithm is applicable for vehicle ECU operations in our research model.

Figure 15 displays each case's average execution time used for predicting a state transition. All the cases except the adaptive training speed with ANNs consume more than the time constraint (0.1s). Consequently, these cases cannot save energy consumption because their predictions are expired when the predictions are applied for ECU operations. Thus, we conclude that the adaptive training speed can produce the most energy savings among the given cases even though the gradient descent with ANNs marks the highest prediction performance. To check this assumption, we estimate the energy improvement rates with consideration of the two kinds of the correct prediction rates in Figure 14 and the average execution times in Figure 15.

5.3. Energy improvement rate

Figure 16 shows energy improvement rates for each case. The rates derive from two kinds of energy consumption. To be specific, when we simulate our system model by using a driving scenario in Table 8, we measure the total energy consumption of our system model without and with using an algorithm case in Figure 16. Based on the difference between the two energy consumptions, we evaluate how much energy consumption is improved by this case in the selected driving scenario. Especially, in evaluating energy improvement rates, we take account of two main examples. First, if an algorithm correctly predicts an unacceptable transition but its execution time exceeds the time constraint, we do not include the energy saved by the prediction in the energy improvement rate. Second, if an algorithm mispredicts an acceptable transition within the time constraint, the energy improvement rate for the algorithm will be degraded[16].

[15]The vehicle OS sends commands to ECU transceivers for activating/deactivating ECUs over the in-vehicle networks. After the ECU transceivers receive these commands, the transceivers will turn on/off ECUs. In this paper, we estimate that the maximum time overheads from the OS command to the ECU operation is 0.4s. The 0.4s derives from the communication service rate (0.1s)[32], the maximum communication delay (0.1s)[31, 34, 35], the maximum time for ECU activation (0.2s)[33].

[16]Because of an misprediction for an acceptable transition, our system model does not make the state transition. Since the mispredicted transition is actually an acceptable transition, the state duration (T_{sleep}) is larger than the breakeven time (T_{be}). Namely, based on the descriptions in Figure 4, energy consumption without the mispredicted transition is larger than energy dissipation with the mispredicted transition.

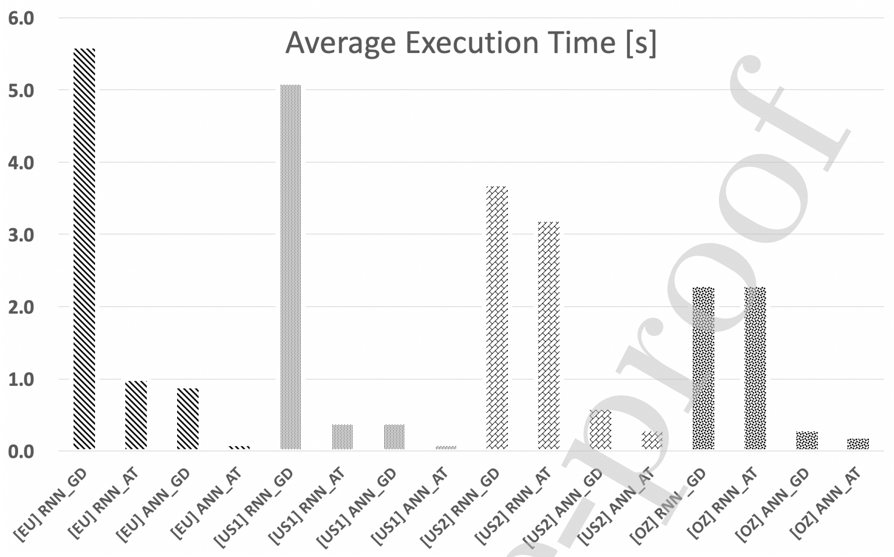


Figure 15: Comparison of average execution times
(EU: Europe, OZ: Australia, GD: gradient descent, AT: Adaptive Training Speed, ANN: Artificial Neural Networks, RNN: Recurrent Neural Networks)

With consideration of these examples, Figure 16 shows energy improve rate for each algorithm case. The adaptive training speed with artificial neural networks achieves the most energy improvement (5%-7%) among the algorithm cases. Because the adaptive training speed provides the second highest performance in the correct prediction rate and the average execution time does not exceed the time constraint (0.1s) for the most driving scenarios, its predicted transitions are used for most energy savings in the electronic control system.

6. Conclusion

Based on the feasibility study and the evaluation, the proposed algorithm with the adaptive training speed using artificial neural networks (ANNs) is the best solution to avoid energy-inefficient ECU operations for the automotive's electronic control system. Even though the adaptive training speed marked less problem-solving performance (correct prediction rate for unacceptable events) than the gradient descent, the adaptive training speed enhanced more energy of the electronic control system. The adaptive training speed completed the neural network training without violating the time constraint. Namely, the trained ANNs produced many valid forecasts. On the other side, the gradient descent generated numerous expired predictions because the gradient descent was exhaustive for training the ANNs and the comprehensive training consumed more execution time than the time constraint. Besides, the adaptive training

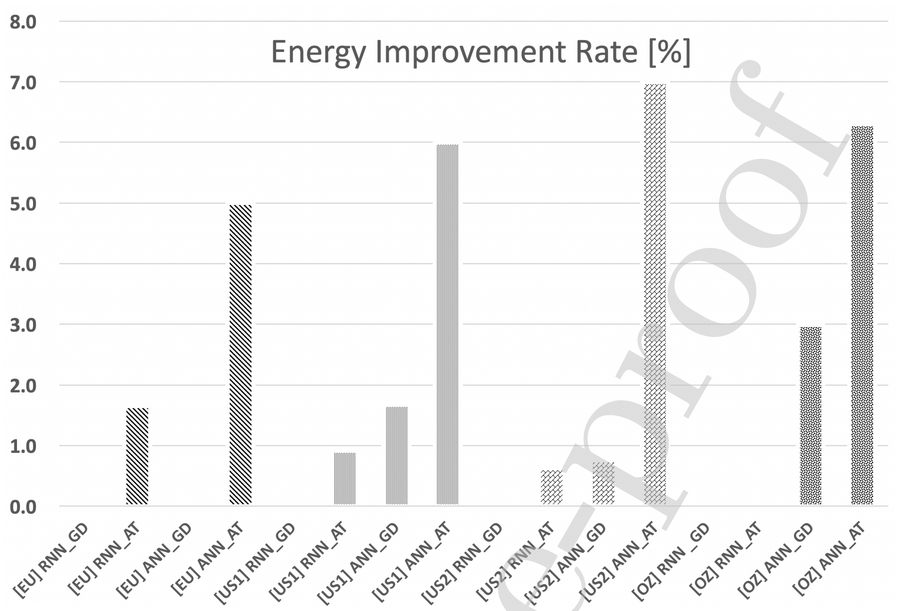


Figure 16: Comparison of energy improvement rates
(EU: Europe, OZ: Australia, GD: gradient descent, AT: Adaptive Training Speed,
ANN: Artificial Neural Networks, RNN: Recurrent Neural Networks)

speed created the minimum side-effects (incorrect prediction rate for acceptable events) in the evaluation. Consequently, by preventing energy-inefficient vehicle events from the valid predictions without critical side-effects, the proposed algorithm with the adaptive speed training using ANNs provides the most substantial energy improvement rates to the vehicle's electronic control system without any operational problems.

For further research work, we consider two directions: (A) performance evaluation with datasets that have numerous event instances and long-term driving, (B) prototype-based research. In the feasibility study and the evaluation, we use real driving data that have at most 72 instances and 12-mile driving distance. For in-depth performance verification regarding the proposed algorithm, we plan to evaluate the algorithm by using real data with a large number of event elements and several-hour driving distance. In addition, we study for the given research problem by using our system model. However, there can be discrepancies between the system model and electric vehicles' real operations. Hence, we plan to apply the proposed algorithm to a prototype electric vehicle for developing the algorithm with consideration of substantial vehicle conditions.

References

[1] U.S. Department of Energy, Where the energy goes: Gasoline vehicles, [Online; accessed 24-December-2018].
URL https://www.fueleconomy.gov/feg/atv.shtml

[2] U.S. Department of Energy, Where the energy goes: Electric cars, [Online; accessed 24-December-2018].
URL https://www.fueleconomy.gov/feg/atv-ev.shtml

[3] United Nations Climate Change, Kyoto protocol - targets for the first commitment period, [Online; accessed 24-December-2018].
URL http://unfccc.int/kyoto_protocol/items/2830.php

[4] United Nations Climate Change, The paris agreement, [Online; accessed 24-December-2018].
URL http://unfccc.int/paris_agreement/items/9485.php

[5] Paul A. Eisenstein, These countries want to ban all vehicles that run on gas or diesel, [Online; accessed 24-December-2018].
URL https://www.nbcnews.com/business/autos/these-countries-want-ban-all-vehicles-run-gas-or-diesel-n781431

[6] U.S. Department of Energy, Fuel economy data, [Online; accessed 24-December-2018].
URL https://www.fueleconomy.gov/feg/download.shtml

[7] U.S. Department of Energy, All-electric vehicles, [Online; accessed 24-December-2018].
URL https://www.fueleconomy.gov/feg/evtech.shtml

[8] U.S. Environmental Protection Agency, The new york city cycle (nycc), [Online; accessed 24-December-2018].
URL https://www.epa.gov/vehicle-and-fuel-emissions-testing/dynamometer-drive-schedules

[9] U.S. Environmental Protection Agency, Epa urban dynamometer driving schedule (udds), [Online; accessed 24-December-2018].
URL https://www.epa.gov/emission-standards-reference-guide/epa-urban-dynamometer-driving-schedule-udds

[10] ARTEMIS: Assessment and reliability of transport emission models and inventory systems, The artemis european driving cycle, urban, [Online; accessed 24-December-2018].
URL http://www.car-engineer.com/the-different-driving-cycles/

[11] Australian Government Department of Environment, Light duty petrol vehicle emissions testing, [Online; accessed 24-December-2018].
URL https://www.environment.gov.au/archive/transport/publications/nise2.html

[12] BOSCH, Can specification - version 2.0, [Online; accessed 24-December-2018] (1991).
URL `http://esd.cs.ucr.edu/webres/can20.pdf`

[13] AUTOSAR (AUTomotive Open System ARchitecture), Requirements on can, [Online; accessed 24-December-2018].
URL `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SRS_CAN.pdf`

[14] International Organization for Standardization, Iso 11898-5:2007, road vehicles – controller area network (can) – part 5: High-speed medium access unit with low-power mode, [Online; accessed 24-December-2018] (2007).
URL `https://www.iso.org/standard/41284.html`

[15] T. Liebetrau, U. Kelling, T. Otter, M. M. Hell, Energy saving in automotive e / e architectures.

[16] L. Benini, A. Bogliolo, G. D. Micheli, A survey of design techniques for system-level dynamic power management, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 8 (3) (2000) 299–316. `doi:10.1109/92.845896`.

[17] P. Bogdan, R. Marculescu, S. Jain, Dynamic power management for multidomain system-on-chip platforms: An optimal control approach, ACM Trans. Design Autom. Electr. Syst. 18 (2013) 46:1–46:20. `doi:10.1145/2504904`.

[18] M. Elgebaly, M. Sachdev, Variation-aware adaptive voltage scaling system, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 15 (5) (2007) 560–571. `doi:10.1109/TVLSI.2007.896909`.

[19] Y.-H. Lu, E.-Y. Chung, T. Simunic, T. Benini, G. de Micheli, Quantitative comparison of power management algorithms, Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537) (2000) 20–26`doi:10.1109/DATE.2000.840010`.

[20] Y.-H. Lu, G. D. Micheli, Comparing system level power management policies, IEEE Design Test of Computers 18 (2) (2001) 10–19. `doi:10.1109/54.914592`.

[21] V. Devadas, H. Aydin, On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications, Proceedings of the 8th ACM International Conference on Embedded Software (2008) 99–108`doi:10.1145/1450058.1450073`.

[22] B. Zhao, H. Aydin, Minimizing expected energy consumption through optimal integration of dvs and dpm, 2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers (2009) 449–456`doi:10.1145/1687399.1687484`.

[23] C. Schmutzler, A. Kruger, F. Schuster, M. Simons, Energy efficiency in automotive networks: Assessment and concepts, 2010 International Conference on High Performance Computing Simulation (2010) 232–240doi:10.1109/HPCS.2010.5547131.

[24] W. Hong, O. Hucke, A. Burger, A. Viehl, O. Bringmann, W. Rosenstiel, State-based power optimization using mixed-criticality filter for automotive networks, 2015 IEEE Intelligent Vehicles Symposium (IV) (2015) 773–778doi:10.1109/IVS.2015.7225778.

[25] AUTOSAR (AUTomotive Open System ARchitecture), Specification of ecu state manager with fixed state machine (autosar cp release 4.3.1), [Online; accessed 24-December-2018].
URL https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_ECUStateManagerFixed.pdf

[26] NXP Semiconductors, S32v234: Vision processor for front and surround view camera, machine learning and sensor fusion applications, [Online; accessed 24-December-2018].
URL https://www.nxp.com/docs/en/data-sheet/S32V234.pdf

[27] NXP Semiconductors, Mpc5554 microcontroller data sheet, [Online; accessed 24-December-2018].
URL https://www.nxp.com/docs/en/data-sheet/MPC5554.pdf

[28] Renesas Semiconductor, User's manual of v850es/jg3-l 32-bit single-chip microcontrollers, [Online; accessed 24-December-2018].
URL https://media.digikey.com/pdf/Data%20Sheets/Renesas/uPD70F3737,38,92,93.pdf

[29] Renesas Semiconductor, Rsk m16c29 user's manual, [Online; accessed 24-December-2018].
URL https://www.renesas.com/en-us/doc/products/tool/001/reg10j0004_rskm16c29_um.pdf

[30] R. I. Davis, A. Burns, R. J. Bril, J. J. Lukkien, Controller area network (can) schedulability analysis: Refuted, revisited and revised, Real-Time Systems 35 (3) (2007) 239–272. doi:10.1007/s11241-007-9012-7.

[31] M.-M. Hell, The new wake-up pattern for a robust system, 16th international CAN Conference.

[32] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, L. Kilmartin, Intra-vehicle networks: A review, IEEE Transactions on Intelligent Transportation Systems 16 (2) (2015) 534–545. doi:10.1109/TITS.2014.2320605.

[33] M.-M. Hell, Power saving in can applications, 11th international CAN Conference.

[34] NXP Semiconductors, Tja1043 high-speed can transceiver, [Online; accessed 24-December-2018].
URL https://www.nxp.com/docs/en/data-sheet/TJA1043.pdf

[35] Infineon, High speed can transceivers application note, tle6250g, [Online; accessed 24-December-2018].
URL https://www.infineon.com/dgdl/High+speed+CAN-AN.pdf?fileId=db3a304320896aa20120c3442faf2391

[36] G. M. Weiss, H. Hirsh, Learning to predict rare events in event sequences, Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining KDD'98 (1998) 359–363.
URL http://dl.acm.org/citation.cfm?id=3000292.3000360

[37] F. Xie, A. Song, V. Ciesielski, Event detection in time series by genetic programming, 2012 IEEE Congress on Evolutionary Computation (2012) 1–8doi:10.1109/TITS.2014.2320605.

[38] V. Guralnik, J. Srivastava, Event detection from time series data, Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '99 (19) (1999) 33–42. doi:10.1145/312129.312190.

[39] G. Dhiman, T. S. Rosing, Dynamic power management using machine learning (2006) 747–754doi:10.1109/ICCAD.2006.320115.

[40] G. Dhiman, T. S. Rosing, Dynamic voltage frequency scaling for multi-tasking systems using online learning (2007) 207–212doi:10.1145/1283780.1283825.

[41] N. AbouGhazaleh, A. Ferreira, C. Rusu, R. Xu, F. Liberato, B. Childers, D. Mosse, R. Melhem, Integrated cpu and l2 cache voltage scaling using machine learning (2007) 41–50doi:10.1145/1254766.1254773.
URL http://doi.acm.org/10.1145/1254766.1254773

[42] A. Weissel, F. Bellosa, Self-learning hard disk power management for mobile devices, Proc. 2nd IWSSPS, Seoul, Korea.

[43] H. Jung, M. Pedram, Supervised learning based power management for multicore processors, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29 (9) (2010) 1395–1408. doi:10.1109/TCAD.2010.2059270.

[44] H. Shen, Y. Tan, J. Lu, Q. Wu, Q. Qiu, Achieving autonomous power management using reinforcement learning, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013) 24:1–24:32. doi:10.1145/2442087.2442095.
URL http://doi.acm.org/10.1145/2442087.2442095

[45] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, P. Cunha, Energy consumption and execution time estimation of embedded system applications, Microprocess. Microsyst. 35 (4) (2011) 426–440. doi:10.1016/j.micpro.2010.08.006.

[46] J. Thomas, Drive cycle powertrain efficiencies and trends derived from epa vehicle dynamometer results, SAE International Journal of Passenger Cars. Mechanical Systems (Online) 7 (4). doi:10.4271/2014-01-2562.

[47] K. S. Richard Barney Carlson, Jeffrey Wishart, On-road and dynamometer evaluation of vehicle auxiliary loads, SAE 2016 World Congress and Exhibition (2016) 9doi:10.4271/2016-01-0901.

[48] T. Ibaraki, Finite automata having cost functions, Information and Control 31 (2) (1976) 153 – 176. doi:10.1016/S0019-9958(76)80005-8.

[49] J. Patel, S. Shah, P. Thakkar, K. Kotecha, Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques, Expert Systems with Applications 42 (1) (2015) 259 – 268. doi:10.1016/j.eswa.2014.07.040.

[50] Y. Kara, M. A. Boyacioglu, Ömer Kaan Baykan, Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange, Expert Systems with Applications 38 (5) (2011) 5311 – 5319. doi:10.1016/j.eswa.2010.10.027.

[51] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980.

[52] G. Hinton, Lecture 6a overview of mini-batch gradient descent.
URL http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[53] Y. S. John Duchi, Elad Hazan, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (2011) 2121–2159.
URL http://jmlr.org/papers/v12/duchi11a.html

[54] M. D. Zeiler, Adadelta: An adaptive learning rate method.
URL https://arxiv.org/abs/1212.5701

[55] N. Qian, On the momentum term in gradient descent learning algorithms, Neural Networks 12 (1) (1999) 145 – 151. doi:https://doi.org/10.1016/S0893-6080(98)00116-6.
URL http://www.sciencedirect.com/science/article/pii/S0893608098001166

[56] R. Ward, X. Wu, L. Bottou, Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization, CoRR abs/1806.01811.

[57] H. Kim, J. Kang, W.-M. Park, S. Ko, Y.-H. Choi, D. Yu, Y. Song, J. Choi, Convergence analysis of optimization algorithms, CoRR abs/1707.01647.

[58] A. Basu, S. De, A. Mukherjee, E. Ullah, Convergence guarantees for rmsprop and adam in non-convex optimization and their comparison to nesterov acceleration on autoencoders.

[59] D. Zhou, Y. Tang, Z. Yang, Y. Cao, Q. Gu, On the convergence of adaptive gradient methods for nonconvex optimization, CoRR abs/1808.05671.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

*I used a template for "Declaration of Interest Statement" for the link (https://www.elsevier.com/journals/applied-soft-computing/1568-4946/guide-for-authors). If there is a problem for my document, please let me know.