Available online at www.sciencedirect.com
ScienceDirect

Neural Networks 21 (2008) 458–465

Neural Networks

www.elsevier.com/locate/neunet

2008 Special Issue

Toyota Prius HEV neurocontrol and diagnostics[☆]

Danil V. Prokhorov[*]

Toyota Technical Center, A division of Toyota Motor Engineering and Manufacturing North America (TEMA), Ann Arbor, MI 48105, United States

Received 10 August 2007; received in revised form 20 November 2007; accepted 11 December 2007

Abstract

A neural network controller for improved fuel efficiency of the Toyota Prius hybrid electric vehicle is proposed. A new method to detect and mitigate a battery fault is also presented. The approach is based on recurrent neural networks and includes the extended Kalman filter. The proposed approach is quite general and applicable to other control systems.
© 2008 Elsevier Ltd. All rights reserved.

Keywords: RNN; Neurocontrol; Battery diagnostics; Fault mitigation; HEV; Control; NN model; NN controller; EKF

1. Introduction

Hybrid powertrains have been gaining popularity due to their potential to improve fuel economy significantly and reduce undesirable emissions. Control strategies of the hybrid electric vehicle (HEV) are more complex than those of the internal combustion engine-only vehicle because they have to deal with multiple power sources in sophisticated configurations. The main function of any control strategy is power management. It typically implements a high-level control algorithm which determines the appropriate power split between the electric motor and the engine to minimize fuel consumption and emissions, while staying within specified constraints on drivability, reliability, battery charge sustenance, etc.

Computational intelligence techniques have previously been applied to HEV power management by various authors. A rule-based control was employed in Baumann, Washington, Glenn, and Rizzoni (2000). Fuel economy improvement with a fuzzy controller was demonstrated in Salman, Schouten, and Kheir (2000) and Schouten, Salman, and Kheir (2002), relative to other strategies which maximized only the engine efficiency. Another system for improving fuel economy in the form of fuzzy rule-based advisor was proposed in Syed, Filev, and Ying (2007). The advisor either lets the driver inputs through intact (accelerator and brake positions), or adjusts its limits to provide advantageous corrections even for a fuel efficiency minded driver.

An intelligent controller combining neural networks and fuzzy logic which could adapt to different drivers and drive cycles (profiles of the required vehicle speed over time) was studied in Baumann, Rizzoni, and Washington (1998). Recently a neurocontroller was employed in a hybrid electric propulsion system of a small unmanned aerial vehicle which resulted in significant energy saving (Harmon, Frank, & Joshi, 2005).

The references cited above indicate a significant potential for improving HEV performance through more efficient power management based on application of computational intelligence (CI) techniques. Though the Toyota HEV Prius efficiency is quite high already, there is a potential for further improvement, as illustrated in this paper.

Unlike traditional hybrid powertrain schemes, series or parallel, the Prius hybrid implements what is called the power split scheme. This scheme is quite innovative and has not been studied extensively yet from the standpoint of application of CI techniques. The Prius powertrain uses a planetary gear mechanism to connect an internal combustion engine, an electric motor and a generator. A highly efficient engine can simultaneously charge the battery through the generator and propel the vehicle (Fig. 1). It is important to be able to set the engine operating point to the highest efficiency possible and at sufficiently low emission levels of undesirable exhaust gases such as hydrocarbons, nitrogen oxides and carbon monoxide.

[☆] An abbreviated version of some portions of this article appeared in Prokhorov (2007) as part of the IJCNN 2007 Conference Proceedings, published under IEE copyright.

[*] Tel.: +1 734 9951017.

Best Paper Award of IJCNN 2007

0893-6080/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.12.043

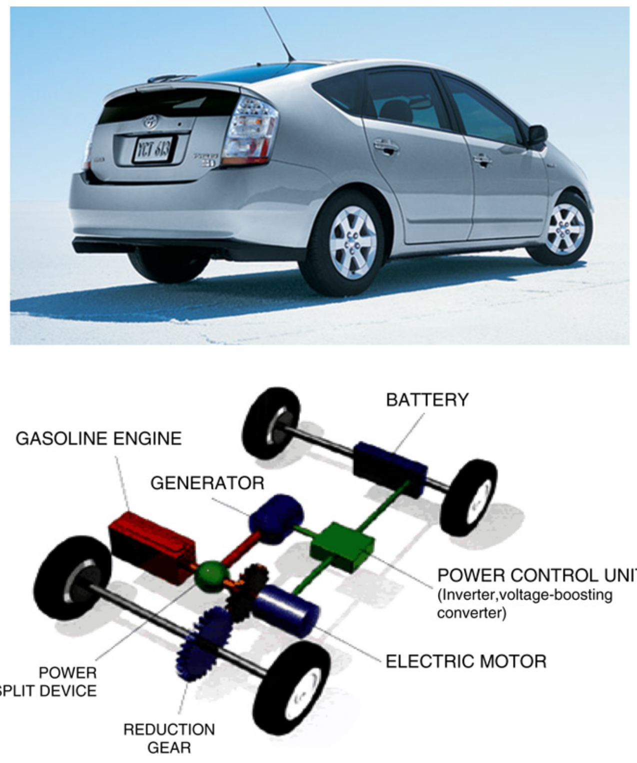

Fig. 1. The Prius car and the main components of the Toyota hybrid system.

The motor is physically attached to the ring gear. It can move the vehicle through the fixed gear ratio and either assist the engine or propel the vehicle on its own for low speeds. The motor can also return some energy to the battery by working as another generator in the regenerative braking mode.

As in the previous work (Prokhorov, 2006; Prokhorov, Puskorius, & Feldkamp, 2001), I employ recurrent neural networks (RNN) as controllers and train them for robustness to parametric and signal uncertainties (known bounded variations of physical parameters, reference trajectories, measurement noise, etc.). I intend to deploy the trained neurocontroller with fixed weights. It is still desirable to have a possibility to influence the closed-loop performance in case some degree of adaptivity is needed, e.g., when an intermittent fault in the system occurs which temporarily makes significant changes in its performance (until repairs are made). It may be not helpful to adapt weights of the controller because (1) it would compromise its already trained weights, i.e., its long-term memory, which is undesirable in the intermittent fault case, and (2) adaptation in strongly nonlinear systems can cause bifurcations. It may be safer to augment the fixed-weight RNN controller by simpler means for adaptation.

This paper is structured as follows. In next section 2 describe main elements of the off-line training. The approach permits me to create an RNN controller which is ready for deployment with fixed weights. I describe my control experiments in Section 3. I then propose a NN for battery diagnostics and discuss ways to mitigate a battery fault in Section 4. Battery fault mitigation is carried out by influencing inputs or outputs of the fixed-weight NN controller, with a diagnostic NN continuously monitoring the closed-loop performance.

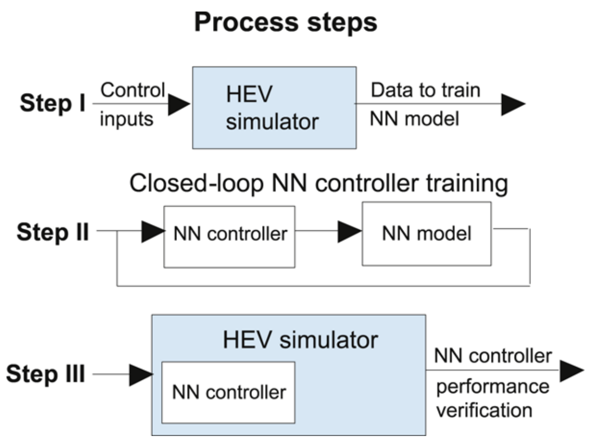

Fig. 2. Steps of my process for NN controller training and verification.

2. Off-line training

I adopt the approach of indirect or model-based control development for off-line training. The Prius simulator is a highly complex, distributed software which makes training a neurocontroller directly in the simulator difficult. I implemented an approach in which the most essential elements of the simulator are approximated sufficiently accurately by a neural network model. The NN model is used to train a neurocontroller by effectively replacing the simulator; this configuration is also known as the *parallel* identification scheme, as in Eaton, Prokhorov, and Wunsch (2000) and Narendra and Parthasarathy (1990). The trained neurocontroller performance is then verified in the simulator (Fig. 2).

The use of differentiable NN for both model and controller makes possible application of the industrially proven training method. I describe here only the main elements of the method, referring the reader to other references for its more comprehensive account (Prokhorov et al., 2001; Puskorius, Feldkamp, & Davis Jr., 1996).

Truncated backpropagation through time (BPTT(h), where h stands for the truncation depth) offers potential advantages relative to forward methods for obtaining sensitivity signals in NN training problems. First, the computational complexity scales as the product of h with the square of the number of nodes (for a fully connected NN). The required storage is proportional to the product of the number of nodes and the truncation depth h. Second, BPTT(h) often leads to a more stable computation of dynamic derivatives than do forward methods because its history is strictly finite. Third, the use of BPTT(h) permits training to be carried out asynchronously with the RNN execution. This feature enabled testing a BPTT-based approach on a real automotive hardware as described in Puskorius et al. (1996).

After the derivatives are computed via BPTT(h), I can update the NN weights. Unlike weight update methods that originate from the field of differentiable function optimization, the extended Kalman filter (EKF) method treats supervised learning of a NN as an optimal filtering problem. The NN weights $\mathbf{w}$ are interpreted as states of the trivially evolving dynamic system, with the measurement equation described by the NN function $\mathbf{h}$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \boldsymbol{\nu}(t) \quad (1)$$

$$\mathbf{y}^d(t) = \mathbf{h}(\mathbf{w}(t), \mathbf{i}(t), \mathbf{v}(t-1)) + \boldsymbol{\omega}(t) \quad (2)$$

where $\mathbf{y}^d(t)$ is the desired output vector, $\mathbf{i}(t)$ is the external input vector, $\mathbf{v}$ is the RNN state vector (internal feedback), $\boldsymbol{\nu}(t)$ is the process noise vector, and $\boldsymbol{\omega}(t)$ is the measurement noise vector. The weights $\mathbf{w}$ may be organized into g mutually exclusive weight groups. This trades off performance of the training method with its efficiency; a sufficiently effective and computationally efficient choice, termed node decoupling, has been to group together those weights that feed each node. Whatever the chosen grouping, the weights of group i are denoted by $\mathbf{w}_i$. The corresponding derivatives of network outputs with respect to weights $\mathbf{w}_i$ are placed in N_{out} columns of $\mathbf{H}_i$.

To minimize at time step t a cost function cost $= \sum_t \frac{1}{2}\boldsymbol{\xi}(t)^{\mathrm{T}}\mathbf{S}(t)\boldsymbol{\xi}(t)$, where $\mathbf{S}(t) > 0$ is a weighting matrix and $\boldsymbol{\xi}(t)$ is the vector of errors, $\boldsymbol{\xi}(t) = \mathbf{y}^d(t) - \mathbf{y}(t)$, where $\mathbf{y}(t) = \mathbf{h}(\cdot)$ from (2), the decoupled EKF equations are as follows (Puskorius et al., 1996):

$$\mathbf{A}^*(t) = \left[\frac{1}{\eta(t)}\mathbf{I} + \sum_{j=1}^{g} \mathbf{H}_j^*(t)^{\mathrm{T}}\mathbf{P}_j(t)\mathbf{H}_j^*(t)\right]^{-1} \quad (3)$$

$$\mathbf{K}_i^*(t) = \mathbf{P}_i(t)\mathbf{H}_i^*(t)\mathbf{A}^*(t) \quad (4)$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \mathbf{K}_i^*(t)\boldsymbol{\xi}^*(t) \quad (5)$$

$$\mathbf{P}_i(t+1) = \mathbf{P}_i(t) - \mathbf{K}_i^*(t)\mathbf{H}_i^*(t)^{\mathrm{T}}\mathbf{P}_i(t) + \mathbf{Q}_i(t). \quad (6)$$

In these equations, the weighting matrix $\mathbf{S}(t)$ is distributed into both the derivative matrices and the error vector: $\mathbf{H}_i^*(t) = \mathbf{H}_i(t)\mathbf{S}(t)^{\frac{1}{2}}$ and $\boldsymbol{\xi}^*(t) = \mathbf{S}(t)^{\frac{1}{2}}\boldsymbol{\xi}(t)$. The matrices $\mathbf{H}_i^*(t)$ thus contain scaled derivatives of network (or the closed-loop system) outputs with respect to the ith group of weights; the concatenation of these matrices forms a global scaled derivative matrix $\mathbf{H}^*(t)$. A common global scaling matrix $\mathbf{A}^*(t)$ is computed with contributions from all g weight groups through the scaled derivative matrices $\mathbf{H}_j^*(t)$, and from all of the decoupled approximate error covariance matrices $\mathbf{P}_j(t)$. A user-specified learning rate $\eta(t)$ appears in this common matrix. (Components of the measurement noise matrix $\mathbf{R}$ are inversely proportional to $\eta(t)$.) For each weight group i, a Kalman gain matrix $\mathbf{K}_i^*(t)$ is computed and is then used in updating the values of the group's weight vector $\mathbf{w}_i(t)$ and in updating the group's approximate error covariance matrix $\mathbf{P}_i(t)$. Each approximate error covariance update is augmented by the addition of a scaled identity matrix $\mathbf{Q}_i(t)$ that represents additive data deweighting.

I employ a multi-stream version of the algorithm above. A concept of multi-stream was proposed in Feldkamp and Puskorius (1994) for improved training of RNN via EKF. It amounts to training N_s copies (N_s streams) of the same RNN with N_{out} outputs. Each copy has the same weights but different, separately maintained states. With each stream contributing its own set of outputs, every EKF weight update is based on information from all streams, with the total effective number of outputs increasing to $M = N_s N_{\text{out}}$. The multi-stream training may be especially effective for heterogeneous data sequences because it resists the tendency to improve local performance at the expense of performance in other regions.

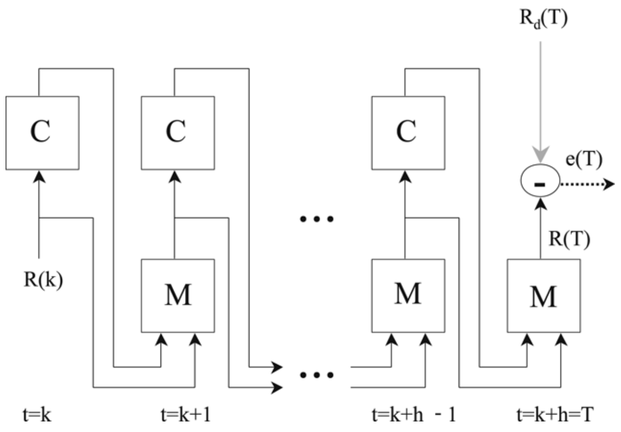

Fig. 3. Pictorial representation of unfolding the closed-loop system consisting of neurocontroller C and plant model M. The variables predicted by the model R are compared with their desired values R_d, generating the error e. While this is done only for the last step of the unfolding, the unfolding process itself is repeated many times, starting from $t = k+1, k+2, \ldots$.

Fig. 3 illustrates how the neurocontroller is trained in my off-line training. After the NN model of the simulator (or plant in general) is trained, we carry out the following procedure:

1. Initiate states of the NN model and NN controllers, assuming RNN is used. Generate a trajectory $\{\mathbf{i}(t), \mathbf{y}^d(t)\}$, $t = k, k+1, k+2, \ldots, N$.

2. Run the closed-loop system forward from time step k to step $k+h$, where $1 \ll h \leq N$, and compute the cost function. The pair NN model controller, or the entire closed-loop system, is run forward for h time steps, thereby generating its h copies (Fig. 3). It may be helpful to think of the current time step as step $k+h$, rather than step k.

3. Backpropagate through the temporal chain of the closed-loop system copies from step $k+h$ to step k while computing derivatives of the relevant outputs with respect to NN controller weights, i.e., carry out BPTT(h).

4. Adjust the weights according to the EKF algorithm with the goal of incremental minimization of the cost in the mean square sense.

5. Move forward one time step (run the closed-loop system forward from step $k+h$ to step $k+h+1$), then repeat the procedure beginning from step 3, etc., until the end of trajectory is reached.

6. Optionally, generate a new trajectory and resume training from step 1.

The training is stopped when sufficiently small values of the cost are obtained.

The described procedure is similar to both model predictive control (MPC) with receding horizon (see, e.g., Allgoewer and Zheng (2000)) and optimal control based on the adjoint (Euler–Lagrange/Hamiltonian) formulation (Stengel, 1994). The most significant differences are that this scheme uses a parametric nonlinear representation for controller (NN) and that updates of RNN weights are incremental, not "greedy" as in the receding-horizon MPC.

3. Control experiments

I first train a NN model to enable off-line training the neurocontroller as discussed in Section 2. To do supervised training of the NN model in Fig. 4, I gather the input–output pairs from 20 diverse drive cycles generated in the Prius simulator. I trained a 25-node structured RNN for 3000 epochs using the multi-stream EKF with $g = 1$ in (3) (Prokhorov et al., 2001) and attained the training root mean squared error (RMSE) of 5×10^{-4} (the largest generalization RMSE was within 20% of the training RMSE). One part of the structured RNN approximates fuel rate, whereas another part does SOC. The parts are decoupled.

The closed-loop control system for training the NN controller is shown in Fig. 4. The converter determines the required values of the speed ω_r^d and the torque T_r^d at the ring gear of the planetary mechanism to achieve the desired vehicle speed specified in the drive cycle. This is done on the basis of the Prius model of motion. The constraint verifier assures satisfaction of various constraints which must hold for the engine, the motor and the generator speeds and torques in the planetary gear mechanism, i.e., ω_e and T_e, ω_m and T_m, ω_g and T_g, respectively.

The first control goal is to minimize the average fuel consumed by the engine. However, fuel minimization only is not feasible. The Prius nickel-metal hydride battery is the most delicate nonlinear component of the system with long-term dynamics due to discharging, charging and temperature variations. It is important to avoid rapid and deep discharges of the battery which can drastically reduce its life, requiring costly repairs or even battery replacement. Thus, the second goal of the HEV control is to maintain the battery State Of Charge (SOC) throughout the drive cycle in the safe zone. The SOC can vary between 0.0 (fully discharged) and 1.0 (fully charged), but the safe zone is typically above 0.4.

I combine the two control goals to obtain $\text{cost}(t) = \lambda_1 sf^2(t) + \lambda_2(t)(\text{SOC}^d(t) - \text{SOC}(t))^2$, where $sf(t)$ stands for specific fuel or fuel rate consumed by the engine at time t, $\lambda_1 = 1$, and $\lambda_2(t) \in [10, 50]$ due to about one order of magnitude difference between values of sf and those of SOC. The desired $\text{SOC}^d(t)$ is constant in my experiments for simplicity. I encourage the controller to pay approximately the same level of attention to both sf and SOC, although the optimal balance between λ_1 and λ_2 is yet to be determined. I also penalize reductions of the SOC below SOC^d five times heavier than its increases to discourage the controller from staying in the low-SOC region for long. Thus, $\lambda_2(t) = 10$ if $\text{SOC}(t) \geq \text{SOC}^d$, and $\lambda_2(t) = 50$ if $\text{SOC}(t) < \text{SOC}^d$.

Ultimately, I would also like to minimize emissions of the harmful gases. In this paper emission reduction is influenced indirectly through reducing the fuel consumption because they are often correlated.

The RNN controller has 5-10R-2 architecture of the recurrent multi-layer perceptron (RMLP), i.e., five inputs, ten

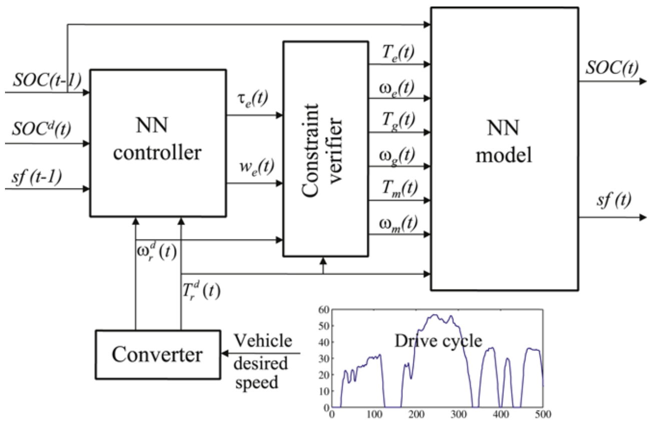

Fig. 4. Block diagram of the closed-loop system for training the NN controller. The converter determines the required values of speed ω_r^d and torque T_r^d at the ring gear of the planetary mechanism to achieve the desired vehicle speed profile. The constraint verifier makes sure not only that the torques and speeds are within their specified physical limits but also that they are consistent with constraints of the planetary gear mechanism. The trained NN model takes care of the remaining complicated dynamics of the plant. The feedback loop is closed via SOC and the fuel rate sf, but the required ω_r^d and T_r^d are guaranteed to be achieved through the appropriate design of the constraint verifier.

recurrent nodes in the fully recurrent hidden layer, and two bipolar sigmoids as output nodes. The RNN receives as inputs the required output drive speed ω_r^d and torque T_r^d, the current engine fuel rate sf, the current SOC and the desired SOC, SOC^d (see Fig. 4; the desired fuel rate is implicit, and it is set to zero). Additional, potentially useful inputs could be the coolant or the battery temperatures, to be added in the future.

The RNN produces two control signals normalized in the range of ± 1. The first output is the engine torque τ_e, and the second output is the engine speed w_e which become T_e and ω_e, respectively, after passing through the constraint verifier.

The RNN controller is trained off-line using the multi-stream EKF algorithm described in Section 2. I train according to the 5-stream EKF algorithm and BPTT(h) with $h = 20$ in which each stream is assigned to a particular instantiation of the NN model. Each stream has a slightly different copy of the NN model to imitate parametric and signal uncertainties in the Prius system. More specifically, the NN models differ in their output node weights only.

Every stream is assigned to its own 50-point segment of the reference trajectory (drive cycle), with the starting point chosen at random. I also choose $\text{SOC}^d(0)$ randomly from the range $[0.5, 0.8]$ and keep it constant for the entire drive cycle. The NN models and drive cycles are redrawn randomly every 20 training epochs, each epoch consisting of processing all 250 points for all streams. I train for 1200 epochs total (about 60 000 weight updates) with $\eta = 0.01$ and $\text{diag}(\mathbf{Q}) = 10^{-4}\mathbf{I}$. These are reasonably effective values for the training control parameters, although they are not fine-tuned.

When training of the NN controller from Fig. 4 is finished, I can deploy it inside the high-fidelity simulator which approximates well the behavior of the real Prius and all its powertrain components (Step III in Fig. 2). As expected, I observed some differences between the neurocontroller

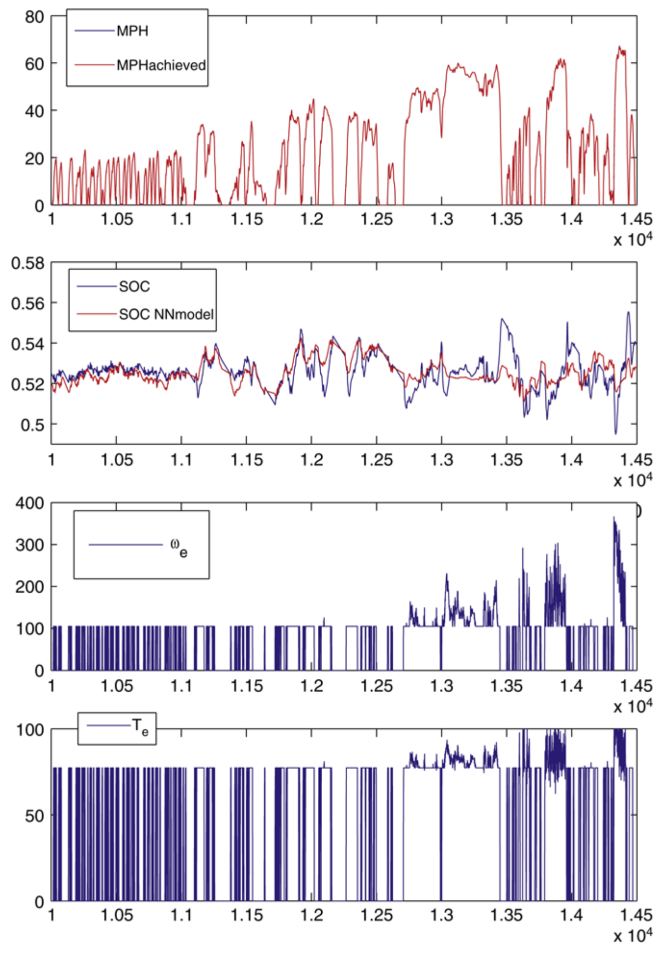

Fig. 5. Illustration of the NN model accuracy in matching the simulator and NN controller performance on a fragment of a very long drive cycle. The SOC is matched accurately enough to produce a good NN controller, and the simulator MPG = 49.6 vs. the NN model prediction of 49.0. MPH and MPG stand for speed in miles per hour and fuel consumption in miles per gallon, respectively. The engine torque (the bottom panel) and speed are in N m and rad/s, respectively.

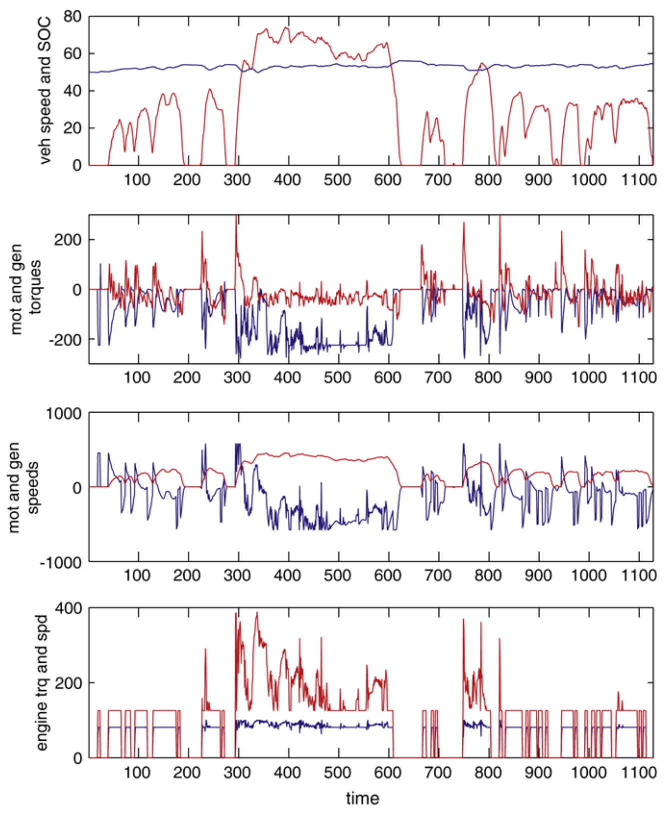

Fig. 6. Illustrative performance of the neurocontroller on a typical drive cycle. The top panel shows the vehicle speed and SOC (in percent), the second panel shows the motor (red) and the generator (blue) torques, T_m and $10 \times T_g$, respectively; the third panel shows the motor ω_m (red) and the generator ω_g (blue) speeds, and the bottom panel shows the engine torque T_e (blue) and speed ω_e (red). Note that $\omega_m = \omega_r^d$ due to the system design constraint. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

performance in the closed loop with the NN model and its performance in the high-fidelity simulator because the NN model and the verifier only approximate the simulator's behavior (see Fig. 5). My results below pertain to the simulator, rather than its NN approximation.

The basic idea of the current Prius HEV control logic is discussed in Hermance (1999). When the power demand is low and the battery SOC is sufficiently high, the motor powers the vehicle. As the power demand and vehicle speed increase, or the SOC reduces below a threshold, the engine is started. The engine power is split between propelling the vehicle and charging the battery through the generator. As the power demand continues to grow, the engine might not be able to stay within its efficiency limits. In those cases the motor can provide power assistance by driving the wheels to keep the engine efficiency reasonably high, as long as the battery can supply the required power. During decelerations the motor is commanded to operate as a generator to recharge the battery, thereby implementing regenerative braking. It is hard to make this *baseline* strategy optimal for such a complex powertrain. A strategy based on a data-driven learning system presents an opportunity to improve over the baseline strategy because of its ability to discern differences in driving patterns and take advantage of them for improved performance.

I compare my RNN controller trained for robustness with the baseline control strategy of the Prius on many drive cycles including both standard cycles (required by government agencies) and nonstandard cycles (e.g., random driving patterns). The RNN controller is better by 17% on average than the baseline controller in terms of fuel efficiency. It also reduces variance of the SOC over the drive cycles by 35% on average.

Fig. 6 shows an example of the neurocontroller results, which should be compared with Fig. 7. The latter depicts the baseline controller results on the same drive cycle. The NN controller advantage appears to be in more efficient usage of the engine, e.g., longer idling at higher torque values. The engine efficiency is 37% vs. 31% for the baseline controller. An even bigger improvement is achieved in the generator efficiency: 72% vs. 49%; other component efficiencies remain basically unchanged.

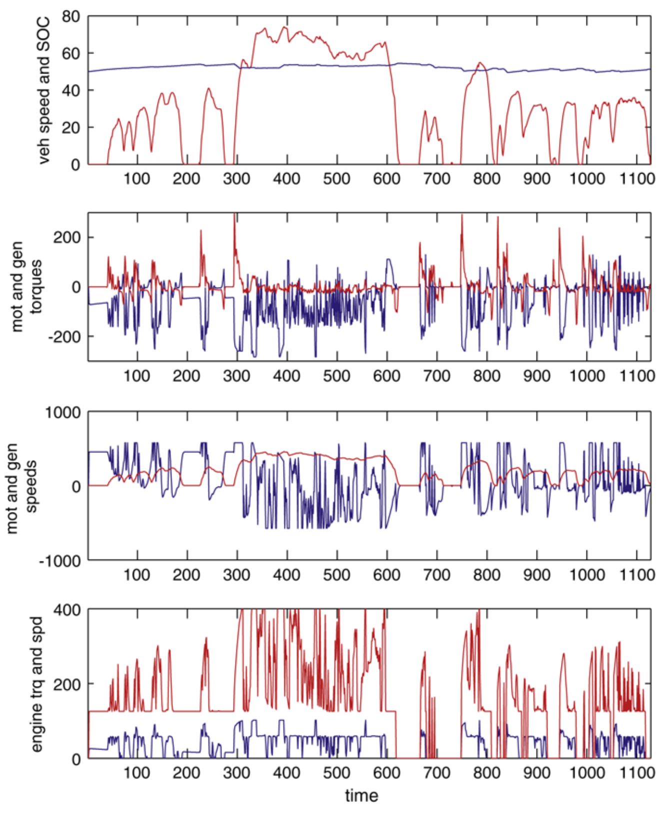

Fig. 7. Illustration of the baseline controller performance. The drive cycle and notation are the same as in Fig. 6.

I also test robustness of the NN controller to measurement noise. Practically, the most critical parameter is the SOC measurement noise because of long delays in the SOC dynamics. Interestingly, my NN controller exhibits very graceful degradation to the significant SOC measurement uniform noise up to 1% in magnitude, resulting in just under 5% reduction of fuel efficiency on average.

4. Battery diagnostics and fault mitigation

The Prius nickel-metal hydride battery is another energy source in the HEV system. Due to current technology limitations, it is important to strive for battery charge sustenance, as large variations of the battery SOC can drastically reduce its life requiring costly repairs or even a replacement.

In this section, I wish to discuss a method to diagnose a battery fault by observing battery inputs and its output (SOC) and comparing simple statistics of the output with the statistics of the normally operating battery. Fig. 8 contrasts behaviors of the normal and the faulty battery. The faulty battery does not hold its charge as long as the normal battery does. I henceforth refer to such faulty battery as leaky battery. It is important not only to detect this fault reliably but also try to mitigate it until the HEV is properly serviced.

To diagnose the battery, I employ a neural network based approach using the same NN training method as described in Section 2. My diagnostic NN inputs are the past SOC, the current and several past values of the required speed ω_r^d

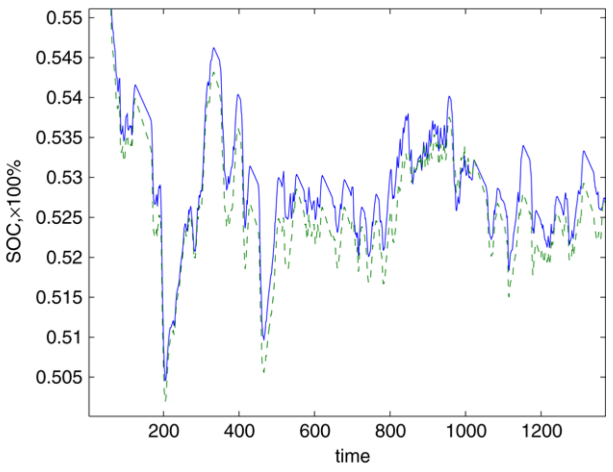

Fig. 8. Evolution of SOC for two different battery states, normal (solid blue) and leaky (dashed green, discharge 40% faster than normal), for a typical drive cycle. Values of the leaky battery SOC are higher occasionally than those of the normal SOC due to feedback control, but the overall MPG is lower for the leaky battery, as expected. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

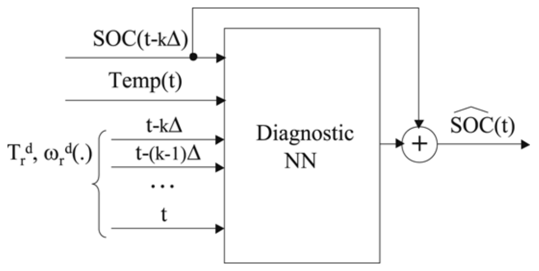

Fig. 9. Block diagram of the diagnostic NN. The required speed ω_r^d and torque T_r^d from the converter in Fig. 4 are fed via the tapped-delay line from the current (t) to the earliest ($t - k\Delta$) time step. The current battery temperature *Temp* and the past SOC are also provided.

and torque T_r^d, and the battery temperature. The diagnostic NN puts out its estimate of the future SOC of the battery. Fig. 9 illustrates a slightly different arrangement in which the diagnostic NN predicts the difference between the past and the future SOC, which I used in experiments of this section.

I intend to compare the RMSE between the output of the diagnostic NN trained on the normal battery data with the SOC observed from the system. The SOC is to be estimated by the NN with as high accuracy as possible because the estimation accuracy determines the fault detection threshold. I carried out several experiments to determine a suitable NN architecture and its inputs, as well as the required measurement accuracy of the SOC.

Initially, I trained a feedforward NN with 10 hidden nodes equipped with the current SOC input, 20 time-delay inputs of the ring gear torque and speed demand with delay between adjacent inputs as 1 sec. For the 9 s ahead SOC target, I obtained the test RMSE of 3.5×10^{-3} on a drive cycle of approximately 2000 s long after 3000 epochs of training. The NN should be trained on a sufficiently rich training set, representing various drive cycles, controller behaviors and other important variabilities.

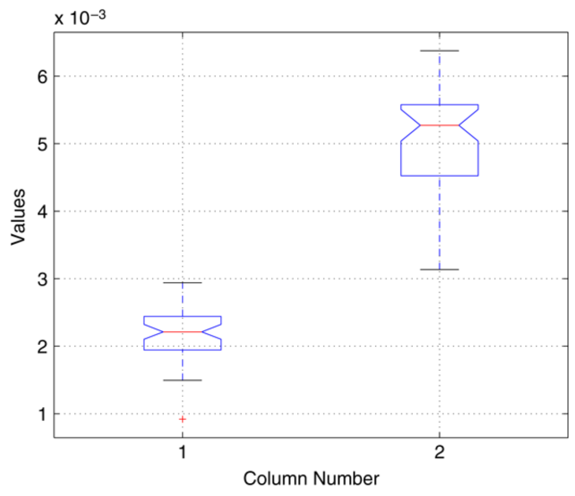

Fig. 10. Distributions of the 9 s ahead SOC RMSE values for the normal (left) and the leaky (right) batteries. The RMSE values are taken over the 100 s windows.

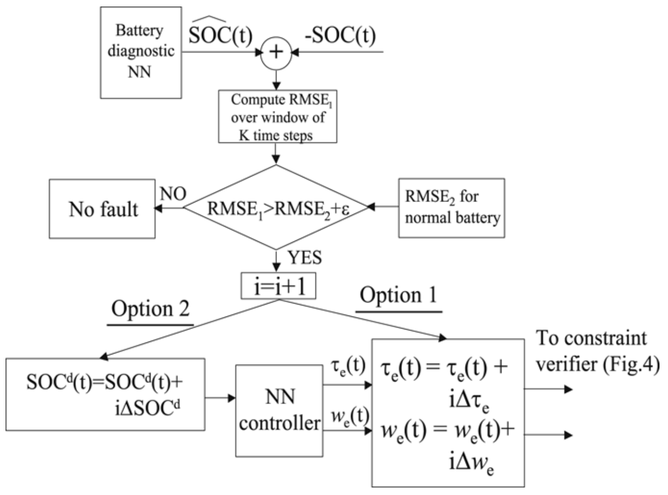

Fig. 11. Block diagram of the battery fault mitigation algorithm. The two alternative options for fault mitigation are shown together: the one which affects the NN controller input SOC^d, and another which affects its outputs.

Compared with the feedforward NN, an RMLP with just five hidden nodes in its only recurrent layer and the same inputs in essentially the same training process demonstrated the test RMSE of 2.3×10^{-3}, which is substantially better than that of the feedforward NN. While it is probably possible to improve the results of the feedforward NN, the RNN is known to be generally better for time series predictions (see, e.g., Mandic and Chambers (2001)). And of course the size of RNN can always be increased if higher prediction accuracy is required because the amount of training data is not a hard restriction in this problem.

Fig. 10 illustrates the diagnostic value of the RNN output. The distribution of the 9 s ahead SOC RMSE values on the left is for the normal battery, and the distribution on the right is for a leaky battery The RMSE values are taken over the 100 s windows, rather than over the entire 10 000 s drive cycle. The 100 s window represents a reasonable balance between diagnostic speed and statistical significance.

It is useful to discuss a way to mitigate the battery fault by adapting or reconfiguring the controller. While the common practice is to resort to NN weight adaptation for this purpose, I prefer to keep the NN weights fixed and rely on adapting components external to the NN controller. My rationale for this is two-fold:

- the NN controller weights are its long-term memory, not desirable to change especially in the case of transient (fixable) fault,
- the NN controller would try to compensate for a fault of some severity even with its weights fixed due to the nature of feedback, especially the RNN controller.

My proposal is supposed to help the fixed-weight NN controller to combat faults which are more severe than those the NN controller can cope with on its own. Fig. 11 illustrates my proposal implemented in this paper. The system constantly compares the diagnostic NN output with the observed SOC and computes the RMSE in a window. This RMSE is compared with the value for the normally operating battery. The mitigating action is initiated as soon as the first RMSE exceeds the second RMSE by a calibratable threshold. I envision two types of mitigating action. First, I can increment the NN controller outputs, the engine torque and the speed, by some amount. Second, I can increment the relevant NN input, i.e., SOC^d. In either case, the engine must clearly work harder when the battery is leaky.

Several comments are in order. The amounts to incrementing the NN input or the outputs are to be calibrated based on engine calibration maps to keep the fuel consumption as low as possible. (Such maps are in common usage among engine control engineers. They link steady state values of engine variables such as fuel, torque, speed, etc.) As far as incrementing the outputs is concerned, I recommend to increment both the torque and the speed because sometimes either of these two variables may be at its upper limit. The battery fault mitigation is quicker when both the variables are incremented, if at all possible.

The algorithm of Fig. 11 is initialized with $i = 0$. The counter i gets incremented every K time steps until the mitigation process succeeds. The counter is reset once the battery is repaired or replaced.

The algorithm of Fig. 11 is verified on several drive cycles. I used $K = 100$, $\epsilon = 10^{-3}$, $RMSE_2 = 2.5 \times 10^{-3}$ and either $\Delta \text{SOC}^d = 0.005$ or $\Delta\tau_e = 0.5$, $\Delta w_e = 1.0$ which were added three times to the input or the outputs, respectively, at the beginning of the drive cycle when the fault was detected. The RMSE$_1$ was $4.8 \times 10^{-3} \pm 0.5 \times 10^{-3}$ before the application of mitigating actions, and it reduced to $3.1 \times 10^{-3} \pm 0.3 \times 10^{-3}$ after the application. The RMSE reduction was achieved at the expense of 10% decrease in MPG, as expected.

The length K of the window is also to be calibrated. While I mentioned above that $K = 100$ seems reasonable, much longer windows might be required because of variability in drive cycles, changes in the NN controller response to mitigating actions, measurement noise, and simplicity of the RMSE as a statistical diagnostic measure. Recall that the battery diagnostic NN is not trained on examples of the closed-loop system behavior with the mitigated fault because such examples are hard to procure in a real world. (Even if examples of the mitigated fault were available, they would be unlikely to match the richness of the normal behavior examples.)

In principle, the RNN controller can be trained to be robust to a battery fault in the same process as described in Section 2. However, it would be of little diagnostic value because of its nontransparency. The explicit diagnostic system proposed here is clearly advantageous.

It is probably possible to learn a mapping between the increments $i\Delta\text{SOC}^d$, or $i\Delta\tau_e$, $i\Delta w_e$ and other relevant variables. However, the faulty system does not need to be fine-tuned. It just has to be properly reconfigured and has to last long enough until necessary repairs are made (sometimes this is called "limp home" capability). That is why I adopted the simpler approach of relatively coarse increments instead of a more complex learnable function approximator.

5. Conclusions

The contribution of this paper is two-fold. First, I propose to use an effective method for off-line NN training in the three-step process to obtain a good NN controller for the Prius HEV high-fidelity simulator. The approach may be applicable to many real-world control problems. Second, I propose a virtual sensor to detect a defective battery and a method to mitigate the battery fault without changing the NN controller weights.

In this paper I used a neural network model of the plant. This enabled the use of multi-stream EKF as the off-line training method. It is also possible to employ an existing nonneural model of a plant and a *nonlinear* Kalman filter method for neurocontrol as described in Prokhorov (2006), however proper care must be taken for adequate simplification of that substantially more complex method.

Miscellaneous issues remain to be clarified in future work. The fault mitigation method provides two options, the SOC correction or the torque and the speed corrections. The full potential of each option needs to be explored further, as well as the diagnostic utility of using statistics other than RMSE. Furthermore, while the battery SOC behavior in different states estimated by the diagnostic NN demonstrates a clear diagnostic value, further research is required to fine-tune this promising diagnostic method.

References

Allgoewer, F., & Zheng, A. (2000). *Nonlinear model predictive control*. Basel: Birkhauser Verlag.

Baumann, B., Rizzoni, G., & Washington, G. (1998). Intelligent control of hybrid vehicles using neural networks and fuzzy logic, SAE technical paper 981061. In *SAE int. cong. and exposition*.

Baumann, B., Washington, G., Glenn, B., & Rizzoni, G. (2000). Mechatronic design and control of hybrid electric vehicles. *IEEE/ASME Transactions on Mechatronics*, *5*(1), 58–72.

Eaton, P., Prokhorov, D., & Wunsch, D. (2000). Neurocontroller alternatives for 'fuzzy' ball-and-beam systems with nonlinear, nonuniform friction. *IEEE Transactions on Neural Networks*, *11*(2), 423–435.

Feldkamp, L. A., & Puskorius, G. V. (1994). Training controllers for robustness: Multi-stream dekf. In *Proceedings of the IEEE international conference on neural networks* (pp. 2377–2382).

Harmon, F., Frank, A., & Joshi, S. (2005). The control of a parallel hybrid-electric propulsion system for a small unmanned arial vehicle using a cmac neural network. *Neural Networks*, *18*(June/July), 772–780.

Hermance, D. (1999). Toyota hybrid system. In *Sae toptec conference proc*.

Mandic, D., & Chambers, J. (2001). *Recurrent neural networks for prediction*. Wiley.

Narendra, K. S., & Parthasarathy, P. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*(1), 4–27.

Prokhorov, D. V. (2006). Training recurrent neurocontrollers for robustness with derivative-free kalman filter. *IEEE Transactions on Neural Networks*, *17*(6), 1606–1616.

Prokhorov, D. V. (2007). Toyota Prius HEV neurocontrol. In *Proceedings of the international joint conference on neural networks*.

Prokhorov, D. V., Puskorius, G. V., & Feldkamp, L. A. (2001). Dynamical neural networks for control. In J. Kolen, & S. Kremer (Eds.), *A field guide to dynamical recurrent networks* (pp. 257–289). IEEE Press.

Puskorius, G. V., Feldkamp, L. A., & Davis, L. I., Jr. (1996). Dynamic neural network methods applied to on-vehicle idle speed control. *Proceedings of the IEEE*, *84*(10), 1407–1420.

Salman, M., Schouten, N., & Kheir, N. (2000). Control strategies for parallel hybrid vehicles. In *Proc. American control conference* (pp. 524–528).

Schouten, N., Salman, M., & Kheir, N. (2002). Fuzzy logic control for parallel hybrid vehicles. *IEEE Transactions on Control Systems Technology*, *10*(3), 460–468.

Stengel, R. (1994). *Optimal control and estimation*. Dover.

Syed, F., Filev, D., & Ying, H. (2007). A rule-based fuzzy driver advisory system for fuel economy improvement in a hybrid electric vehicle. In *Proceedings of North American fuzzy information processing society conference* (pp. 178–183).