Priority Neuron: A Resource-Aware Neural Network for Cyber-Physical Systems

Maral Amir, *Graduate Student Member, IEEE*, and Tony Givargis, *Senior Member, IEEE*

***Abstract*—Advances in sensing, computation, storage, and actuation technologies have entered cyber-physical systems (CPSs) into the smart era where complex control applications requiring high performance are supported. Neural networks (NNs) models are proposed as a predictive model to be used in model predictive control (MPC) applications. However, the ability to efficiently exploit resource hungry NNs in embedded resource-bound settings is a major challenge. In this paper, we propose priority neuron network (PNN), a resource-aware NNs model that can be reconfigured into smaller subnetworks at runtime. This approach enables a tradeoff between the model's computation time and accuracy based on available resources. The PNN model is memory efficient since it stores only one set of parameters to account for various subnetwork sizes. We propose a training algorithm that applies regularization techniques to constrain the activation value of neurons and assigns a priority to each one. We consider the neuron's ordinal number as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number in the layer. This imposes a relatively sorted order on the activation values. We conduct experiments to employ our PNN as the predictive model of a vehicle in MPC for path tracking. To corroborate the effectiveness of our proposed methodology, we compare it with two state-of-the-art methods for resource-aware NN design. Compared to state-of-the-art work, our approach can cut down the training time by 87% and reduce the memory storage by 75% while achieving similar accuracy. Moreover, we decrease the computation overhead for the model reduction process that searches for n neurons below a threshold, from $O(n)$ to $O(\log n)$.**

***Index Terms*—Cyber-physical system, model predictive control (MPC), neural networks (NNs), resource-aware.**

I. Introduction

CYBER-PHYSICAL systems (CPSs) are composed of cyber and physical components in a feedback loop, where physical processes affect computations and vice versa [1]–[3]. With the recent developments in CPS, cloud computing, machine learning, and artificial intelligence technologies, it is just a matter of time before autonomous drivers replace humans on the road. Vehicles are now embedded with intelligent devices that enable the vehicle to respond to various factors and obstacles, sudden acceleration or braking, etc., in real-time. The control and prediction of system dynamics are important factors in autonomous driving [4], [5]. Model predictive control (MPC), also known as receding horizon control, is an advanced control method. MPC makes explicit use of a model of the physical system to estimate its behavior for a given stream of inputs in a predetermined prediction horizon. The predicted outputs depend on the past inputs/outputs, and the future control signals [6]. As shown in Fig. 1, these future control signals are calculated by the optimizer taking into account the cost function and enforced constraints. The cost function usually takes the form of a quadratic function of errors between the predicted output signal and the reference trajectory. In the standard approach, ordinary differential equations (ODEs) are employed as the predictive model to represent the dynamic behavior of a physical system. Iterative methods to approximate a solution for nonlinear ODEs have introduced challenges in the design of embedded MPCs in terms of scalability, performance, and power consumption [7].

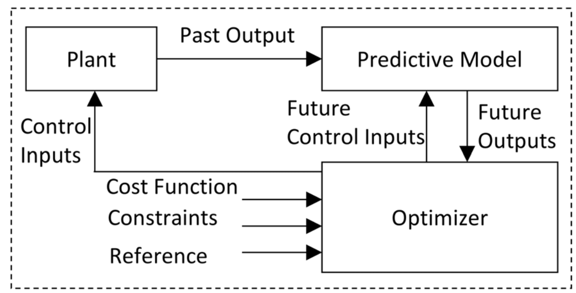

Fig. 1. MPC loop.

The computational overhead in traditional MPC grows exponentially with the length of the prediction horizon [8]. Research shows that a stable MPC controller requires a sufficiently large prediction horizon [9]. On the other hand, short prediction horizons are preferred for improved prediction accuracy of predictive models. This is because harmful effects of the poor estimates are amplified over a long prediction horizon time. Here, the problem is addressed by proposing an MPC approach that uses an adaptive prediction horizon with respect to quality measures [10]. However, the numerical effort needed in order to solve the optimal control problem for a long prediction horizon still remains significant. One

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. This work was supported by the National Science Foundation under NSF Grant 1563652. This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2018 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Maral Amir.)*

The authors are with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: mamir@uci.edu; givargis@uci.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857319

approach to overcome the computational burden of long horizon predictions is by implementing multirate prediction. In this approach, each look-ahead has a separate weight in the estimation of the steering input, where the furthest look-ahead point has the lowest weight [8].

Another method that is proposed to handle the computational issue associated with MPC systems is to use accelerated predictive models of the physical system. Different variants of neural networks (NNs) (e.g., recurrent NNs (RNNs) [11]) hold promising performance for time-series prediction as they can easily be built to predict multiple steps ahead all at once. These models are well-known to have the ability to learn linear and nonlinear relations between input and output variables without prior knowledge [12]. However, the use of NN models for long prediction horizon MPC problems could raise scalability and computational complexity challenges. The state-of-the-art methodologies are focused on reducing the size of the NN models without significantly affecting the performance [13]–[15]. These methodologies leverage the intrinsic error tolerance property of the NN models due to their parallel and distributed structure. Therefore, model reduction schemes could be exploited to employ the NN as the predictive model in the MPC loop. Several recent studies have focused on rescaling the size of the NN to adjust the resource usage on the embedded platform with respect to response time, power, and accuracy targets [16]. In other words, several sizes of the NN are available at runtime to manage resources for inference time-, safety-, and energy-constrained tasks. Moreover, continuous learning of NNs in data-driven modeling [17], transfer learning techniques [18], and adaptive modeling [19] impose significant training-time constraints at *runtime*.

A. Our Contribution

In this paper, we propose priority neuron network (PNN), a novel NN model that is featured with a reconfigurable architecture. Our objective is to design a resource-aware reconfigurable NN model that not only computes the future outputs as time series data in constant time, but is also memory efficient. The summary of our contributions in this paper are as follows.

1) We develop a reconfigurable NN model to fit the dynamic behavior of the physical systems for multistep-ahead prediction in receding horizon problems. Our resource-aware NN model can be reconfigured to various network sizes at runtime while storing only *one set of weight parameters* for memory efficiency.
2) We propose a training algorithm that controls the priority of each neuron in the computation of the model's output. We regulate the priority of each neuron using regularization techniques enforced on weight parameters. We consider the neuron's ordinal number as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number. We can reconfigure our NN model to smaller sizes by eliminating low priority neurons. This approach allows the tradeoff between the model's computation time and accuracy in resource-constrained systems.
3) We implement our reconfigurable NN model that contains multiple subnetworks using one-time training, hence reducing overall training time.
4) Our priority-based training algorithm enforces a sorted distribution on activation values of neurons. This helps to reduce the computation complexity of the model reduction process when searching for n neurons below the pruning threshold, from $O(n)$ to $O(\log n)$. It needs to be pointed out that we are not proposing a pruning methodology, but a memory efficient NN model that can be reconfigured to smaller sizes with less computation complexity at runtime.
5) We apply our method to train a three-layer fully connected NN model to be employed as the predictive model of a vehicle in MPC for path tracking application. We conduct closed-loop simulation of MPC using ODE predictive models to collect the training data. To evaluate the efficacy of our methodology, we compare it with two state-of-the-art approaches-Inc [20] and Big/Little [16]—that are targeted for resource-aware NN design in embedded systems. We show that our proposed PNN model outperforms the BL method with 89% reduction in training time and 78% saving in memory storage. The PNN model shows similar results to Inc method in terms of memory and model reduction complexity. However, we show that PNN follows a single training process to adjust weight parameters as opposed to Inc method that is based on multiple retraining. Therefore, the PNN model can cut down the training time by 86% with respect to Inc method while maintaining a better prediction performance from 0.25% to 0.21%.

The rest of this paper is organized as follows. In Section II, we summarize the state-of-the-art approaches to solve the computational complexity of MPC systems and design resource-efficient NN models. We describe our proposed method in Section III. We demonstrate the effectiveness of our framework for path following application in Section IV. Finally, we give our conclusions in Section V.

II. Background and Related Work

Advanced control methodologies have emerged for path planning and path following applications in modern vehicles. Nonlinear MPC is leveraged to develop path following control systems while handling model uncertainties, constraints and nonlinearities. A predictive model of the physical plant is used to estimate the future outputs for a prediction horizon within a window of time and with respect to known input and output values (Fig. 1). Mathematical descriptions in the form of ODEs are used to model the linear/nonlinear behavior of the physical system [21]. ODE solvers are applied to estimate solutions that converge to the exact solution of an equation or system of equations [22]. A runtime optimization routine is evaluated as a parametric quadratic function to calculate a set of future control inputs subject to constraints enforced by the environment and system dynamics. These routines are computationally intensive, and for nonlinear physical models, the computational overhead grows with complexity of the model [23].

One of the challenges in classic MPC is that the computational overhead increases with the length of the prediction horizon [8]. One approach to overcome the computational burden of long horizon predictions is by implementing a multirate prediction control strategy, where the prediction horizon is sampled in nonequidistant way [24]. In this approach, for a determined prediction horizon of n time steps, the initial steps have a shorter sampling period than the ones in the more distant future. In other words, fine tuning the control in such a way as to reduce the importance of predictions that contribute to time steps further in the future. Novel approaches are proposed for nonlinear dynamic system modeling and identification, where the NN realizes the behavior of a set of ODEs with smaller computation overhead [12], [25]. Moreover, data-driven NNs are increasingly in demand. Data-driven NNs are based on direct use of input-output observations collected from various real-world processes to perform system optimization, control and/or modeling [26]. Classic NNs have a three-layer structure, namely input, hidden, and output layers. Each layer contains a set of neurons with edges to pass the information. The edges entering the neurons are associated with weight parameters. The weight parameters are adjusted in a training algorithm (e.g., by back propagation) so that the difference between the network's prediction and the target output is minimized.

Developing resource-efficient NNs for embedded systems with limited hardware resources is a challenging task. To solve the memory complexity of NN models, many model compression approaches are proposed based on the claim that NN models have natural error tolerance because NNs usually contain more neurons than necessary to solve a given problem [27]. Many network pruning and model reduction techniques are proposed in the previous work with promising results [28]–[30]. However, finding an optimal pruning solution is NP-hard and requires a costly retraining process [31]. Many works have focused on selecting weight parameters for pruning based on criteria such as magnitude of the weight, activation value for the respective neuron, and increase in training error [32]–[34]. Han *et al.* [35] proposed an iterative pruning method that removes all neuron connections whose weight is lower than a certain threshold. This approach converts a dense fully connected layer into a sparser layer. The pruning is followed by a retraining process to boost the performance of the trimmed NN. A common approach to reduce the size of the "parameter intensive" fully connected layers is to reduce the magnitude of the overall weight parameters by including regularization terms in the model's cost function. Pan *et al*. [15] exploited regularization terms during the training process to simplify the NN model. At the end of the training, the NN is trimmed by dropping neurons below a certain threshold.

Another approach to address resource-constrained deployment of NNs for embedded systems is to adapt the size of the NN model to the performance requirements. Park *et al.* [16] addressed the energy complexity of NNs using a novel big/little implementation, whereby a score margin metric is employed to select between the two sizes. This approach is memory intensive such that it requires storing separate sets of weights for different sizes of NNs. Tann *et al.* [20] addressed the memory complexity problem by proposing a multistep incremental training algorithm such that the weights trained in earlier steps are fixed. In this method, multiple subnetworks with different sizes are formed while storing and using only one sets of weight parameters. Although this approach is close to ours, our proposed method is more computationally flexible in generating multiple subnetwork sizes and does not suffer from a time-consuming retraining process. In the following section, we describe PNN, our proposed reconfigurable NN model and its training algorithm.

III. Method

A. Application of Neural Networks in Model Predictive Control

MPC exploits a predictive model of the physical system to produce an optimized control input sequence. The predictive model computes the output of the system, a number of time steps into the future based on the current output and future control input values. Therefore, the predictive model to estimate future outputs at time k in the next n time steps—$Y(k+n|k)$—can be formulated as a time series prediction function f of future control inputs $I(k+n|k)$ and a vector of current state variables $S(k|k)$ for $S = [S_0, S_1, \ldots, S_{N_s}]$. Time-series data is a sequence of time-ordered values as measurements of some physical process [36]

$$Y(k+n|k) = f(\mathbf{S}(\mathbf{k}|\mathbf{k}), I(k+n|k)). \tag{1}$$

The prediction function in (1) can be fitted in a multiple input multiple output NN model with future control inputs and current state of the physical system as its input features and the future outputs in the next n time steps as its target outputs. Once the function is learned, the acyclic NN model computes the future outputs as a time-series data in *constant computing time* [12]. We use a three-layer fully connected feed-forward NN (FFNN) to fit (1) and approximate the dynamic behavior of the physical system. The FFNN is a class of NNs, where the input signal feeds forward through the network layers to the output in a single direction. Here, each layer of the network consists of computing neurons with edges that typically have a weight parameter. The output $\hat{y}_i$ of the NN model can be computed as follows given x_k input features for $i \in \{1 \cdots N_o\}$ and $k \in \{1 \cdots N_i\}$:

$$\hat{y}_i = \sum_{j=1}^{N_h}\left[w_{ji}^2\, \sigma\left(\sum_{k=1}^{N_i} w_{kj}^1 \mathbf{x}_\mathbf{k} + \theta_j^1\right) + \theta_i^2\right] \tag{2}$$

where N_i, N_h, and N_o denote the numbers of input-layer, hidden-layer, and output-layer neurons, respectively. The parameters w_{kj}^1 and w_{ji}^2 are weights connecting the first layer to hidden layer and connecting the hidden layer to the output layer, respectively, and are adjusted in the learning process. The threshold offsets for the hidden and output layers are represented as θ^1 and θ^2. The function $\sigma(.)$ represents an activation functions, e.g., sigmoid, or rectified linear unit (ReLU), that limits the variation to output values with respect to changes in NN parameters.

B. Architecture of Priority Neuron Neural Network As Predictive Model in MPC

We propose PNN, a resource-aware reconfigurable NN such that the full model can be reconfigured to smaller sizes for less computation time and relatively comparable accuracy. Here, we deploy our proposed NN model for multistep ahead time-series prediction in constant time for an MPC application. However, the proposed NN model can be generalized for other prediction applications, e.g., computer vision. As stated in Section III-A, the nonlinear model in (1) is used by MPC to compute future behavior of the physical system can be fitted into a three-layer fully connected FFNN. The future control inputs and current state of the physical system are given as the input features to the FFNN to approximate the future outputs in the next n time steps. The proposed NN model can be described as in (2) for N_i = (# of state variables(N_s) + N_o) and $N_h = N_o$ = (# of time steps in the prediction horizon(n)). The value for N_h is set empirically equal to N_o. We have two weight matrices W^1 and W^2 with sizes $(N_i \times N_h)$ and $(N_h \times N_o)$ containing connecting weights of our hidden and output layers, respectively. We use the ReLU activation function which is one of the most widely used activation functions and is defined as

$$\sigma(z) = \max(0, z). \tag{3}$$

During the prediction process of the NN, we would ideally want a few neurons in the network to not activate, thereby making the activations sparse and efficient. The ReLU activation function gives us the ability to design a sparser NN model because it outputs 0 for negative input values and imposes no constraint on the positive inputs. Equation (2) is broken down into (4a) and (4b) to compute the outputs of hidden and output neurons, respectively. Here, for brevity, the bias parameters are deleted

$$h_j = \sigma\left(\sum_{k=1}^{N_i} w^1_{kj}\mathbf{x_k}\right) \tag{4a}$$

$$\hat{y}_i = \sum_{j=1}^{N_h}\left(w^2_{ji}\ h_j\right). \tag{4b}$$

Hereafter, we are seeking a methodology for an architecture of an NN that stores one set of weight parameters yet can be reconfigured to smaller sizes of the NN with small drop in accuracy. To adopt the reconfigurability feature in our model, we exploit the multirate prediction idea suggested by [8] that assigns lower accent to further look-ahead points in the computation of the future dynamic behavior of the system. Therefore, the proposed PNN model follows a sequential priority-based architecture. This means we consider the neurons' ordinal numbers as our priority criteria such that the priority of each neuron is inversely proportional to its ordinal number in the given layer. Therefore, the model can be reduced starting from the neuron with the highest ordinal number. Our goal is to synchronize the priority level of the output and hidden neurons so that the model reduction process is more computationally efficient for runtime applications. We will elaborate more on this in Section III-D. In Fig. 2, we show the architecture of the proposed PNN as a three-layer FFNN where higher priority neurons are colored darker. We can deploy PNN as a resource-aware predictive model for closed-loop MPC to estimate the future outputs $[Y_0, Y_1, \ldots, Y_{N_h}]$. Here, we use the future control inputs $[I_0, I_1, \ldots, I_{N_h}]$ and current state variables $[S_0, S_1, \ldots, S_{N_s}]$ as input features. In the following section, we describe our proposed training algorithm and the associated cost function to develop the priority-based NN model.

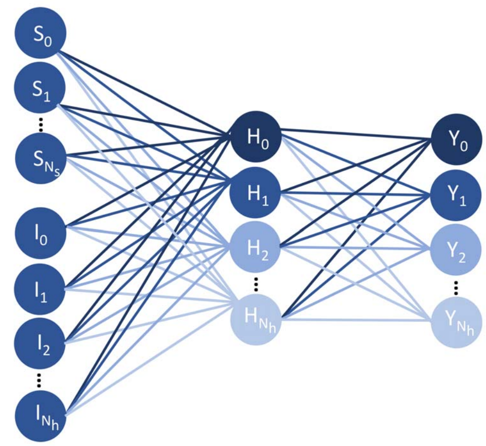

Fig. 2. PNN model.

C. Training Algorithm to Prioritize Neurons

During the training process of an NN, an optimization algorithm is exploited to minimize an objective function $E_0(.)$, which is simply a mathematical function based on the model's learning parameters (e.g., weights and biases). We might use sum of the squared deviations of our neuron's output $\hat{y}_i$ from the target output y_i as the loss function for N_o number of outputs denoted as

$$E_0(w, b) = \frac{1}{2N_o}\sum_{i=1}^{N_o}\left(y_i - \hat{y}_i\right)^2. \tag{5}$$

The learning parameters are optimized and updated in an iterative training process toward a solution that minimizes the loss function. A learning rate η is assigned to the training algorithm that determines the size of the steps we take at each iteration to reach a (local) minimum. For a convex optimization problem like this, we use derivatives of the loss function ∇E. Therefore, the following updating rule is formulated for the weight parameters to be updated after $(t+1)$th update iteration:

$$w^{t+1} \leftarrow w^t - \eta \nabla E_0. \tag{6}$$

For our optimization algorithm, we employ a variant of gradient descent called adaptive moment estimation (Adam) [37] which computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. In the proposed PNN model, the priority of the neuron determines how important the value of that neuron is in the overall performance of the NN. In order to control the priority of each neuron, we enforce constraints on the computation of its output value. This can be done through regularization techniques that restrain the growth of weight parameters. From (4), we see that the weight parameters used to compute the hidden neuron h_j are $W^1[:, j] = [w^1_{1j}, w^1_{2j}, \ldots, w^1_{N_ij}]$. The output neuron $\hat{y}_i$ is computed using weight parameters $W^2[:, i] = [w^2_{1i}, w^2_{2i}, \ldots, w^2_{N_hi}]$. We call the weight parameters of each neuron its *associated weights*.

1) Regularization: A common approach to reduce the complexity and size of NN models is to constrain the magnitude of the overall weight parameters by including regularization terms in the model's cost function. The $L1$ norm is one of the most commonly used regularization techniques that penalizes weight values by adding the sum of their absolutes to the error term. Therefore, the cost function E with the $L1$ regularization term is

$$E(w, b) = E_0(w, b) + \frac{1}{2}\lambda \sum_{l=1}^{2}\sum_{i=1}^{N_l} |W_i^l| \qquad (7)$$

where λ is the weight decay coefficient for which larger values lead to larger cost, and causes the training algorithm to generate small weight values. Existing work sets the same weight decay coefficient for all layers to avoid the computational costs required to manually fine-tune each coefficient. However, to train our priority-based NN model, we penalize each weight with a specific weight decay coefficient so that the value of the corresponding weight is constrained to grow up only to a desired threshold point. Hence, the activation of each neuron is governed by the weight decay coefficients of its *associated weights*. As shown in Algorithm 1, we use a new cost function for our three-layer fully connected feed-forward PNN

$$E(w, b) = E_0(w, b) + \frac{1}{2}\sum_{k=1}^{N_i}\sum_{j=1}^{N_h} |\lambda^1_{kj} w^1_{kj}| + \frac{1}{2}\sum_{j=1}^{N_h}\sum_{i=1}^{N_o} |\lambda^2_{ji} w^2_{ji}| \qquad (8)$$

for $\lambda^1 \in \Lambda^1$ and $\lambda^2 \in \Lambda^2$, where Λ^1 and Λ^2 are two weight decay matrices of our hidden and output layers, respectively. Therefore, the new updating rule for weight parameters is

$$w^{t+1} \leftarrow w^t - \eta\Big(\nabla E_0 + \Lambda^1 W^1 + \Lambda^2 W^2\Big). \qquad (9)$$

In the following section, we describe our heuristic algorithm used to assign values to weight decay coefficients such that a sorted priority-based architecture is enforced on the proposed NN model.

D. Model Reconfiguration of PNN Model

In PNN, we want to force a priority onto each neuron during the computation of model output so that the *accuracy is maintained* after reconfiguring the network to smaller subnetworks by removing low priority neurons. Therefore, we consider larger weight decay coefficients for *associated weights* of neurons that are desired to have lower level of priority and vice versa. We are following the multirate prediction scheme that allocates less stress on accuracy of further look-ahead points. We design our weight decay matrices so that a sorted priority-based architecture for our PNN is developed during the training process. The intuition behind the sorted priority-based architecture of the PNN is to reduce the complexity of the model reconfiguration and reduction process. Model pruning approaches to constrain the complexity of NN models by applying regularization techniques, have been around for a while [28], [38]. These approaches are based on an exhaustive search process to remove neurons with activation values below a certain threshold. In our proposed priority-based architecture, we enforce a sorted priority on hidden neurons to compute the overall performance of the model. This helps reduce the time complexity for searching neurons below a certain activation value as we can employ a binary search algorithm. Therefore, the worst-case time complexity for the model pruning process in our PNN model with n number of hidden neurons is $O(\log n)$ as opposed to standard architectures that require $O(n)$ worst-case time complexity to prune the network. Moreover, the model can be reduced to smaller subnetworks at constant time $O(1)$ due to its reconfigurability feature that is adopted throughout the training process.

There is always a tradeoff between the number of subnetworks and the accuracy of the model. We assign the same level of priority to the number of neurons that are deleted at each level of model reduction. We call this number the *priority size* and denote it as p. Fig. 3 illustrates the reconfiguration process of the original NN model where neurons are sorted and colored in terms of priority and importance. At each level of reconfiguration, p number of hidden neurons with the least level of priority are deleted from the end of the hidden layer. Hence, their input and output weight connections are also removed from the weight space of the NN. These

Algorithm 1: Priority Neuron Training Algorithm

Input: input features - x
Input: output targets - y
Output: trained NN - *PNN*
Output: estimated outputs - $\hat{y}$

```
// initialize NN weights
```
init_random W
```
// estimate outputs given W weights
```
$\hat{y} = PNN\ (x)\ [W]$
```
// evaluate residual error
```
$err = \sum_{i=0}^{N_o} (y_i - \hat{y}_i)^2$
```
// evaluate regularization penalty
```
$reg = \sum |\Lambda^1_{N_i \times N_h}.W^1_{N_i \times N_h}| + \sum |\Lambda^2_{N_h \times N_o}.W^2_{N_h \times N_o}|$
```
// evaluate loss function
```
$loss = err + reg$
```
// optimize W weights for minimal loss
```
$\overline{W}$ = **AdamOptimizer** (*loss*)
```
// estimate outputs given optimal W̄
```
$\hat{y} = PNN\ (x)\ [\overline{W}]$
return $[PNN, \hat{y}]$

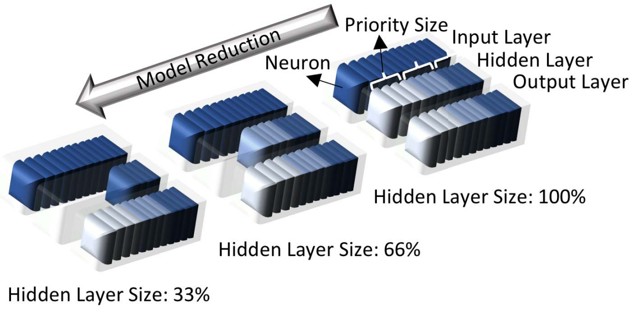

Fig. 3. Model reduction process for a three-layer fully connected NN with priority size $p = 4$.

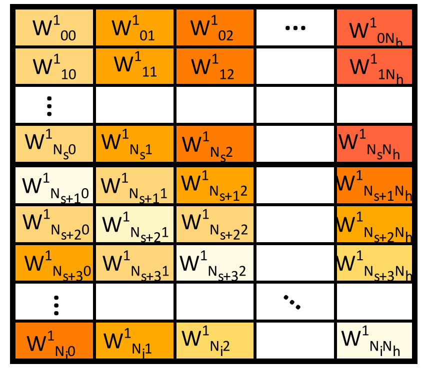

Fig. 4. Weight parameters of hidden layer.

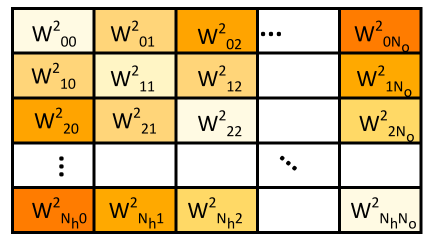

Fig. 5. Weight parameters of output layer.

subnetworks can be deployed separately while reducing the memory complexity to a single network. In other words, only one set of weight parameters are stored for multiple subnetworks of different sizes. We consider neuron's ordinal number as our priority criteria which can be mapped into index values for neuron's associated weights. Therefore, the weight decays vary with respect to row and column indices of the weight matrix where r and c denote the row and column indices, respectively. Equations (10) and (11) are expanded from (4). In (11), we see N_o number of output formulas that are used to estimate the future output behavior of the physical system in the next N_o time steps, hence the size of the prediction horizon is N_o. It needs to be noted that, here we do not include the bias terms for simplification purposes

$$h_0 = w^1_{00}s_0 + w^1_{10}s_1 + \cdots + w^1_{N_i0}I_{N_i} \tag{10a}$$
$$h_1 = w^1_{01}s_0 + w^1_{11}s_1 + \cdots + w^1_{N_i1}I_{N_i} \tag{10b}$$
$$\cdots$$
$$h_{N_h} = w^1_{0N_h}s_0 + w^1_{1N_h}s_1 + \cdots + w^1_{N_iN_h}I_{N_h} \tag{10c}$$

$$y_0 = w^2_{00}h_0 + w^2_{10}h_1 + \cdots + w^2_{N_h0}h_{N_h} \tag{11a}$$
$$y_1 = w^2_{01}h_0 + w^2_{11}h_1 + \cdots + w^2_{N_h1}h_{N_h} \tag{11b}$$
$$\cdots$$
$$y_{N_o} = w^2_{0N_o}h_0 + w^2_{1N_o}h_1 + \cdots + w^2_{N_hN_o}h_{N_h}. \tag{11c}$$

Let us assume that the model is trained for a priority-based architecture where the priority of neurons decreases inversely with their ordinal number. For a pretrained model with priority size $p = 1$, we want to reduce the size of the model by removing hidden neuron h_{N_h} with the least priority level from the hidden layer. While removing the hidden neuron h_{N_o}, its associated weight connections $W^1[:, N_h] = [w^1_{0N_h}, w^1_{1N_h}, \ldots, w^1_{N_iN_h}]$ and $W^2[N_h, :] = [w^2_{N_h1}, w^2_{N_h2}, \ldots, w^2_{N_h(N_o-1)}]$ are removed from W^1 and W^2, respectively. In the next section, we describe the selection of weight decay coefficients to enforce a sorted priority on hidden and output neurons. For a simple implementation we use the same number of hidden and output neurons. Therefore, the W^2 weight matrix is squared.

E. Decay Matrix

A graphical illustration of our W^1 and W^2 weight matrices for hidden and output layers with $p = 1$ is shown in Figs. 4 and 5, respectively. The weight matrices in Figs. 4 and 5 are darker colored based on the value of their corresponding weight decay coefficients. This helps to visualize the selected distribution pattern for weight decay coefficients where a priority-based architecture for our PNN model is developed. In order to maintain the accuracy of the model after the removal of hidden neuron h_{N_h} [computed in (10c)], we want the model reduction to affect the least number of output neurons possible. Therefore, we seek to adjust the weight parameters so that removing the hidden neuron h_{N_h} mostly impacts the least priority output neuron y_{N_o}. Hence, we select weight decay coefficients for the weight parameters in the vector $[w^2_{N_h1}, w^2_{N_h2}, \ldots, w^2_{N_hN_o}]$ in a descending order so that the least weight decay value is assigned for $w^2_{N_hN_o}$. Smaller weight decay coefficients push the training algorithm to assign greater values for the weight parameters. In this method, we try to zero out $[w^2_{N_h1}, w^2_{N_h2}, \ldots, w^2_{N_h(N_o-1)}]$ as much as possible such that the removal of h_{N_h} has minimal impact on the values $[y_1, y_2, \ldots, y_{(N_o-1)}]$.

To expand this idea to other neurons in the hidden layer, we should change the weight decay coefficients above the main diagonal of W^2, in descending order per column and in ascending order per row, so that the least weight decay coefficients are placed on the main diagonal. Moreover, we should adjust the weight decay coefficients below the main diagonal of W^2 in ascending order per column and in a descending order per row. We use ascending order per column so that the priority level of output neurons decreases for larger ordinal numbers and descending order per row forces the weight parameters on the diagonal to contribute the most to the computation of their corresponding output neuron. We propose (12) to compute the weight decay coefficient for each weight parameter in order to regulate the sorted priority order of PNN neurons. Here, r and c denote the row and column index of the weight matrix, respectively. The parameter p stands for the number of neurons deleted at each model reduction process, hence the *priority size*

$$f(x) = \begin{cases} \left[\lambda_{rc} : \lambda_{r(c+p)}\right] = \beta f\left(\dfrac{r}{c}\right), & r \geq c \\ \left[\lambda_{rc} : \lambda_{(r+p)c}\right] = \beta f\left(\dfrac{c}{r}\right), & r < c. \end{cases} \tag{12}$$

Here, $f(.)$ can be considered as a linear, exponential, or logarithmic, etc. growth function considering the target application. The type of function $f(.)$ determines the variance of the priority distribution among various neurons at each layer. The greater the variance of the priority distribution is, the more ways the original NN can be reconfigured into subnetworks. That means less neurons (p) are deleted per model reconfiguration (reduction) process. Larger variance for the priority order of neurons decreases the model accuracy as it enforces more constraints on weight parameters. Therefore, the function $f(.)$ is assigned based on design requirements of the target application and the tradeoff between the model accuracy and number of subnetworks embedded in one NN model. The parameter β maps the computed value of weight decay from (12) to a range as $\lambda \in [\lambda_{\min}:\lambda_{\max}]$. This range is empirically selected based on the tradeoff between the model accuracy and the number of hidden neurons deleted per reconfiguration of the model-priority size. For our future work, we plan to automate the optimal selection of ranges for the weight decay coefficient.

F. Other Types of Neural Networks

The proposed priority-based approach is applied to a fully connected FFNN architecture. This is because state-of-the-art methods proposed fully connected FFNN as a predictive model to approximate dynamic behavior of physical systems in an MPC application. Previous state-of-the-art approaches has mostly focused on reducing the size of the fully connected layers in other NN architectures because these layers are well known to be parameter intensive and occupy more than 90% of the model size [15]. Another popular architecture of NNs for time series forecasting is RNN which is distinguished from FFNN by having signals traveling in both directions and introducing loops in the network. The RNN architecture can be converted into an FFNN by unfolding over time [11]. Therefore, in our future work, we plan to expand our method to other NN architectures. Although we evaluate the effectiveness of our methodology for MPC applications, it can be generalized to other applications of NN models.

IV. Experimental Results

A. Experimental Setup

Our implementation is based on the TensorFlow framework [39] executed on a PC with a quad-core Intel Core i7 and 16 GB of DDR3 RAM. The MPC formulation is implemented in software using the ACADO Toolkit framework [40], which is open source software written in C++ for automatic control and dynamic optimization. To evaluate the efficacy of our proposed methodology, we exploit the PNN as a predictive model in an MPC system for the path following application. We describe the process on how we collect our training dataset in the following section.

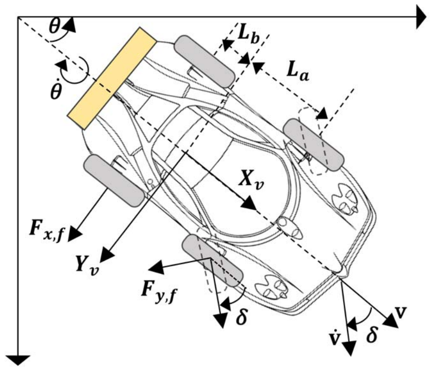

Fig. 6. Schematic of the vehicle model.

B. Simulation to Collect Training Data

As mentioned in Section II, the dynamic behavior of a physical system formulated as ODE can be fitted into a fully connected FFNN. The future control inputs and current state of the physical system are fed as the input features to the FFNN in order to predict the future outputs in the next n time steps. To collect the training dataset, we exploit the following ODE model of a vehicle [41] as shown in (13) and Fig. 6 to conduct offline simulation of MPC for a path following application:

$$\dot{s} = \begin{bmatrix} v\sin(\theta) \\ v\cos(\theta) \\ \cos(\delta)a - \dfrac{2}{m}F_{y,f}\sin(\delta) \\ \phi \\ \dfrac{1}{J}(L_a(ma\sin(\delta) + 2F_{y,f}\cos(\delta)) - 2L_bF_{y,r}) \\ \omega \end{bmatrix}. \tag{13}$$

Here, $s = [x, y, v, \theta, \phi, \delta]$ is the vector of state variables with acceleration a and steering angular speed ω as control inputs. The variables x and y stand for longitudinal and lateral positions, and v and θ are velocity and the azimuth. The variables δ and ϕ represent the steering angle and speed, respectively. The distance from sprung mass center of gravity to the front and rear axles are denoted as L_a and L_b, respectively, and J is the angular momentum. The variables $F_{y,f}$ and $F_{y,r}$ stand for front and rear tire lateral forces. These forces are computed from the following equations:

$$F_{y,f} = C_y\left(\delta - \frac{L_a\phi}{v}\right) \tag{14a}$$

$$F_{y,r} = C_y\left(\frac{L_b\phi}{v}\right) \tag{14b}$$

where C_y is the lateral tire stiffness. We applied real-world parameters of a 2011 Ford Fusion as $L_a = L_b = 1.5$ m, mass $m = 1700$ kg, and tire stiffness data for our experiments. The MPC formulation to follow the reference path x^r, y^r is the

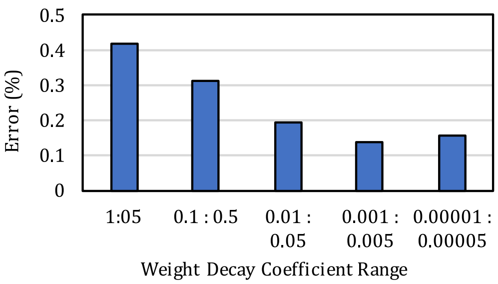

Fig. 7. Performance of PNN for different ranges of weight decay coefficients.

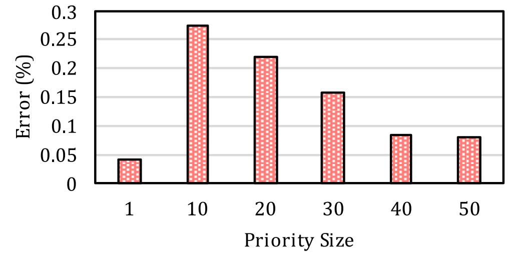

Fig. 8. Performance of PNN for different priority sizes.

solution to the following optimization problem:

$$\begin{aligned} \min_{x,y}.\quad & \sum_{t=0}^{T_p} \|\hat{x}(k+1|k) - x^r(k+1|k)\|_{Q_c}^2 && \text{(15a)} \\ & + \|\hat{y}(k+1|k) - y^r(k+1|k)\|_{Q_c}^2 && \text{(15b)} \\ \text{s.t.}\quad & \delta_{\min} \le \delta \le \delta_{\max} && \text{(15c)} \\ & \omega_{\min} \le \omega \le \omega_{\max} && \text{(15d)} \\ & a_{\min} \le a \le a_{\max}. && \text{(15e)} \end{aligned}$$

We simulate the MPC to predict 101 time steps in the future with time intervals of 5.05 s for a vehicle with an average speed of $v = 10$ (m/s). The appropriate value for the prediction horizon and step size is bounded by some factors such as stability and accuracy requirements and it varies based on plant dynamic characteristics. We implement an FFNN with input size $N_i = 6 + 102$ for six values of current state variables and future control inputs in the next 101 time steps. We select $N_o = 102$ as the output size for our NN to predict the future output of the physical system in the next 101 time steps. The number of hidden neurons in our three-layer FFNN are $N_h = N_o$.

C. PNN Training

In order to fine tune the range of weight decay coefficients $\lambda \in [\lambda_{\min}:\lambda_{\max}]$ and select an appropriate value for the constant factor β in (12), we empirically pick the values that yield the best performance on a held-out dataset. Therefore, we conducted experiments based on five different ranges of coefficients. Fig. 7 shows the error rate of the PNN model with respect to variations in the range of weight decay coefficients. The optimal range of weight decay coefficients for each layer may change with respect to the size of the next layer. In back propagation training, the gradient term in (9) is scaled with the size of the next layer [42]. Therefore, to compensate for the rescaling in the gradient term of the update rule, the optimal range for weight decay coefficients might change. These results are derived for priority size of $p = 10$, which denotes the number of hidden neurons that are removed at each reconfiguration of the model to a smaller subnetwork. Greater values of p restrict the original NN model to be reconfigured to less number of subnetworks. Naturally, there is always a tradeoff between the accuracy of the model and the number of subnetworks as shown in Fig. 8. Considering this tradeoff, the user might select an optimal priority size based on the design requirements for the target application. The error values in this figure are collected while reducing the size of the NN to 50% of its original size. A tradeoff still remains between the number of subnetworks with acceptable error values and the percentage at which the size of the model is reduced. With respect to the application and design requirements, the user may select the appropriate value for the hyper parameter p.

D. Comparison to State-of-the-Art Methodologies

We evaluate the performance of our methodology in training a resource-aware NN model with two state-of-the-art approaches that are proposed as solutions to implement resource efficient NN in embedded system. By using the notation *resource-aware NN model*, we are implying that these NN models are targeted for systems that monitor the resource usage and dynamically manage the allocated resources to the NN model with respect to runtime constraints. The results are collected for a three-layer fully connected NN of 108×102 and 102×102 inputs to its hidden and output layers, respectively. The *Big/Little* approach [16], suggests multiple implementations of an NN model with small to bigger sizes. In the *Incremental* method [20], which is the most similar to ours, the NN is trained based on an iteratively incremental training algorithm where the weights computed in the earlier steps are fixed. The *Big/Little* approach would require separate memory storage to hold model parameters of different sizes. Moreover, a retraining process is mandatory to generate multiple sizes for the NN model. The *Inc* method is more memory efficient such that only one set of model parameters are stored to implement an NN model that can be reconfigured into subnetworks with different sizes. However, this approach suffers from the retraining overhead per increment of size. In today's embedded systems, where runtime continuous learning of NNs is required, retraining process overhead is prohibitive [17]. Our proposed PNN model is memory efficient such that only one set of weights are computed for multiple subnetworks. Furthermore, we compute the model parameters for PNN in a single-training process. Throughout the examples, we use the following abbreviation to indicate the three models: 1) PNN: priority-based; 2) Inc: Incremental; and 3) BL: Big/Little.

Emerging research is based on developing approaches to estimate the number of neurons and hidden layers required for an NN [43]. However, these approximations also depend on the type of the database samples for which the network is designed. Therefore, it is still challenging to determine a good network topology for different applications. Therefore, exhaustive pruning and model reduction methodologies are in demand

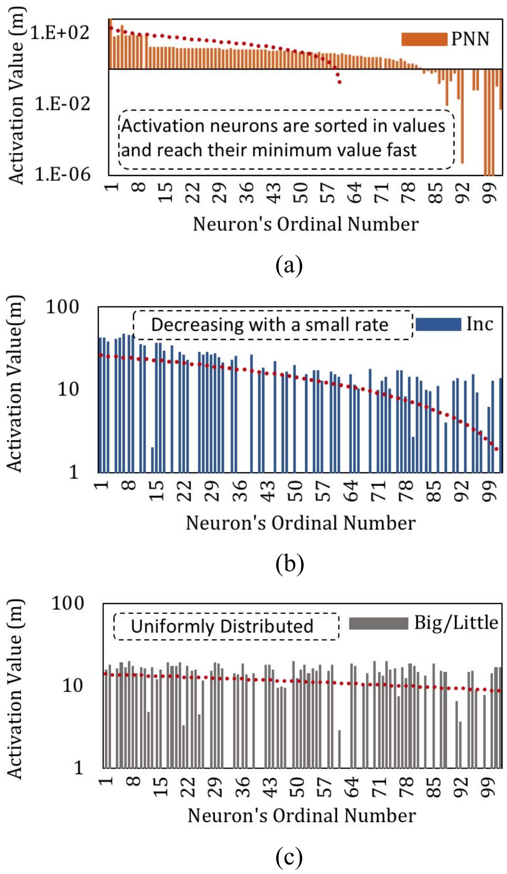

Fig. 9. Comparing activation values of neurons with respect to their ordinal number. Activation values for neurons in (a) PNN, (b) Inc., and (c) Big/Little.

to reduce the over-sized NN models. One advantage of our proposed priority-based training algorithm is that it enforces a relatively sorted distribution to the activation values. We compare the activation value of hidden neurons for our proposed PNN model with respect to the *incrementally* trained model and the *Big/Little* model that is trained with no constraint on its weight parameters in Fig. 9. For fairness of comparison, all experiments are conducted with the same size for all three models. The ordinal number of the neuron denotes the position of the respective neuron in the layer. The dotted red line shows the trend for linear changes in activation values with respect to ordinal number of the neuron. As shown in Fig. 9(a), the activation values for the hidden neurons in PNN with priority size $p = 10$ is following a sorted order. The trend line shows that the density of the model is mostly populated throughout the first neurons and the activation values for the neurons further in the end of the layer are forced to be very small. This is as opposed to the two other methods that show a more uniform distributions of activation values for the neurons. The incremental approach in Fig. 9(b) also shows slight sorted order among activation values. However, as represented by the trend line, the rate of change for neuron's activation value with respect to its ordinal number is very slow compared to PNN method. In other words, in incremental approach, the weight parameters are adjusted more uniformly throughout the network. This decreases the number of subnetworks and the number

TABLE I
COMPARING THE TRAINING PROCESS

Model	# of Sub-Networks	Retrain	# of Retrain	Train Time (s)
PNN	6	No	0	2627
Inc	6	Yes	6	21534
Big/Little	6	Yes	6	25020

of hidden neurons that can be pruned from the model without major drop in accuracy.

Table I compares the training process for a three layer fully connected FFNN using the three aforementioned methods. The data is collected to train six separate subnetworks of various sizes using the three methods. As we can see in the table, our proposed method can generate six separate subnetworks in single training process. This is as opposed to the two other methods that require retraining for each of the subnetworks. The performance of these six subnetworks is evaluated in Fig. 10(a) and (b) where the x-axis represents the number of hidden neurons at each subnetwork. The retraining process imposes additional computation complexity to retune the parameters and hyper parameters. We can see that our proposed model reduces the computation overhead for the training process substantially. The training time is a critical matter especially in embedded systems for CPS applications where many NN models are trained on the fly.

In Fig. 10(a), we show the prediction time values over six different subnetwork sizes. The results show similar performance for all three approaches in terms of runtime prediction overhead which increases for larger network size. As shown in the figure, by reducing the number of hidden neurons to half of its original size, we can improve the computation overhead by 30%. However, this saving in computation time comes as a tradeoff for model accuracy. Fig. 10(b) shows the percentage prediction error values for different subnetwork sizes. The results for the BL [16] method that trains the subnetworks separately with no additional constraints show that after a certain point the model error does not change with growth in the NN size. This justifies the over-parameterization phenomena in training the NN that urges pruning and model reduction methodologies. Moreover, the mean of prediction error for six different subnetworks using our proposed PNN method and Inc. [20] are 0.2% and 0.25%, respectively. Therefore, our proposed PNN method outperforms the Inc approach for better prediction performance with no additional retraining process needed.

In order to evaluate the comparability of model accuracy among the three methods, we also show the probability distribution of prediction error values in Fig. 10(c). These results are collected for a full-size NN with no model reduction process performed. We can see in the figure that the low variation in prediction errors using our proposed PNN model, confirms its stable performance in prediction of various test data. Moreover, the average of prediction errors for the PNN model is very close to that of BL method. This experiment ensures that our proposed model is validated as a memory

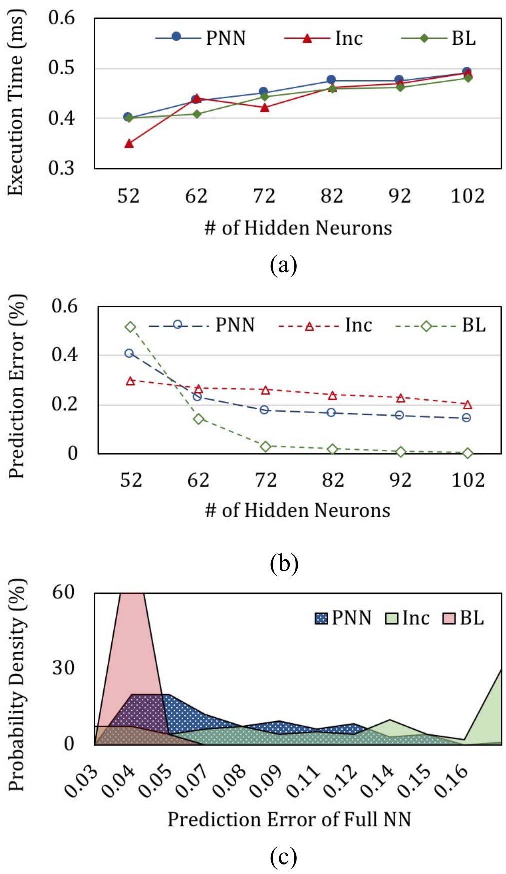

Fig. 10. Performance comparison of three resource-aware approaches. (a) Execution time. (b) Prediction error. (c) Probability distribution of prediction error for full-size NN.

TABLE II
COMPARING MEMORY REDUCTION WITH RESPECT TO ERROR

Model	# of Sub-Networks	# of Parameters	Memory Reduction	Mean Error (%)
PNN	6	21522	78%	0.2
Inc	6	21522	78%	0.25
Big/Little	6	87292	-	0.125

efficient architecture for NN models with small drop in accuracy and comparable performance can be acquired using all three methods.

We compare the efficiency of the three resource-aware methods in terms of memory requirements and model reduction complexity in Table II. The PNN and Inc methods are both memory efficient in that they need one set of weight parameters to store multiple subnetwork sizes. This is as opposed to the BL method that requires separate memory to store each subnetwork. Therefore, we can achieve 78% saving in memory to store six subnetworks with very small loss in accuracy.

To summarize, our proposed PNN model outperforms the BL method with 89% reduction in training time and 78% saving in memory storage. Moreover, the computation complexity of the model reduction process to search for n neurons below the pruning threshold is improved from $O(n)$ to $O(\log n)$. The PNN model shows similar results to Inc method in terms of memory and model reduction complexity. However, we show that PNN follows a single training process to adjust weight parameters as opposed to Inc method that is based on multiple retraining. Therefore, The PNN model can cut down the training time by 86% with respect to Inc method while maintaining a better prediction performance from 0.25% to 0.21%.

V. CONCLUSION

In this paper, we proposed PNN, a resource-aware NN model with a reconfigurable architecture. We proposed a training algorithm to exploit regularization constraints on each neuron based on their ordinal number at a given layer. This enforces a sorted order distribution for the activation value of the neurons. We implemented our model for a three-layer fully connected NN architecture to be employed as the predictive model of a vehicle in MPC for path tracking application. To corroborate the effectiveness of our proposed methodology, we compared it with two state-of-the-art methods for resource-aware NN design. We showed that compared to current state-of-the-art, our approach achieves 75% reduction in memory usage and 87% less training time with no significant drop in accuracy. Moreover, we improve the computational complexity of the model reduction process in order to prune n number of neurons, from $O(n)$ to $O(\log n)$.

REFERENCES

[1] F. Cicirelli, L. Nigro, and P. F. Sciammarella, "Model continuity in cyber-physical systems: A control-centered methodology based on agents," *Simulat. Model. Pract. Theory*, vol. 83, pp. 93–107, Apr. 2017.

[2] K. Vatanparvar and M. A. A. Faruque, "Eco-friendly automotive climate control and navigation system for electric vehicles," in *Proc. ACM/IEEE 7th Int. Conf. Cyber. Phys. Syst. (ICCPS)*, Vienna, Austria, 2016, pp. 1–10.

[3] K. Vatanparvar and M. A. A. Faruque, "Design and analysis of battery-aware automotive climate control for electric vehicles," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 4, p. 74, 2018.

[4] M. Amir and T. Givargis, "Hybrid state machine model for fast model predictive control: Application to path tracking," in *Proc. IEEE 36th Int. Conf. Comput.-Aided Design*, Irvine, CA, USA, 2017, pp. 185–192.

[5] M. Amir and T. Givargis, "HES machine: Harmonic equivalent state machine modeling for cyber-physical systems," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Santa Cruz, CA, USA, 2017, pp. 31–38.

[6] E. F. Camacho and C. Bordons, *Model Predictive Control in the Process Industry*, 2nd ed. London, U.K.: Springer-Verlag, 2007.

[7] A. N. Gorban and D. Roose, *Coping With Complexity: Model Reduction and Data Analysis*, vol. 75. Heidelberg, Germany: Springer, 2011.

[8] J. S. Barlow, "Data-based predictive control with multirate prediction step," in *Proc. IEEE Amer. Control Conf. (ACC)*, Baltimore, MD, USA, 2010, pp. 5513–5519.

[9] A. Jadbabaie and J. Hauser, "On the stability of receding horizon control with a general terminal cost," *IEEE Trans. Autom. Control*, vol. 50, no. 5, pp. 674–678, May 2005.

[10] G. Droge and M. Egerstedt, "Adaptive time horizon optimization in model predictive control," in *Proc. IEEE Amer. Control Conf. (ACC)*, San Francisco, CA, USA, 2011, pp. 1843–1848.

[11] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, *An Overview and Comparative Analysis: Recurrent Neural Networks for Short Term Load Forecasting*. New York, NY, USA: Springer, 2017.

[12] J. Fojdl and R. W. Brause, "The performance of approximating ordinary differential equations by neural nets," in *Proc. 20th IEEE Int. Conf. Tools Artif. Intell. (ICTAI)*, vol. 2. Dayton, OH, USA, 2008, pp. 457–464.

[13] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 6071–6079.

[14] A. Mujika, F. Meier, and A. Steger, "Fast-slow recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5915–5924.
[15] W. Pan, H. Dong, and Y. Guo, "DropNeuron: Simplifying the structure of deep neural networks," *arXiv:1606.07326 [cs, stat]*, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.07326
[16] E. Park *et al.*, "Big/little deep neural network for ultra low power inference," in *Proc. 10th Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Amsterdam, The Netherlands, 2015, pp. 124–132.
[17] K. Vatanparvar and M. A. A. Faruque, "ACQUA: Adaptive and cooperative quality-aware control for automotive cyber-physical systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Irvine, CA, USA, 2017, pp. 193–200.
[18] W. Hu, Y. Qian, F. K. Soong, and Y. Wang, "Improved mispronunciation detection with deep neural network trained acoustic models and transfer learning based logistic regression classifiers," *Speech Commun.*, vol. 67, pp. 154–166, Mar. 2015.
[19] W. He, Y. Chen, and Z. Yin, "Adaptive neural network control of an uncertain robot with full-state constraints," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 620–629, Mar. 2016.
[20] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth. (CODES+ISSS)*, Pittsburgh, PA, USA, 2016, pp. 1–10.
[21] K. Vatanparvar and M. A. A. Faruque, "Battery lifetime-aware automotive climate control for electric vehicles," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2015, pp. 1–6.
[22] C. T. Kelley, *Solving Nonlinear Equations With Newton's Method*. Philadelphia, PA, USA: SIAM, 2003.
[23] P. Krauthausen and U. D. Hanebeck, "A model-predictive switching approach to efficient intention recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Taipei, Taiwan, 2010, pp. 4908–4913.
[24] U. Halldorsson, M. Fikar, and H. Unbehauen, "Nonlinear predictive control with multirate optimisation step lengths," *IEE Proc. Control Theory Appl.*, vol. 152, no. 3, pp. 273–284, May 2005.
[25] N.-B. Hoang and H.-J. Kang, "Neural network-based adaptive tracking control of mobile robots in the presence of wheel slip and external disturbance force," *Neurocomputing*, vol. 188, pp. 12–22, May 2016.
[26] D. Solomatine, L. M. See, and R. J. Abrahart, "Data-driven modelling: Concepts, approaches and experiences," in *Practical Hydroinformatics*. Heidelberg, Germany: Springer, 2009, pp. 17–30.
[27] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.
[28] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.
[29] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.
[30] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, 2016, pp. 243–254.
[31] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4860–4874.
[32] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[33] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, vol. 2, 2017, pp. 1389–1397.
[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.
[35] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[36] Z. Tang and P. A. Fishwick, "Feedforward neural nets as models for time series forecasting," *ORSA J. Comput.*, vol. 5, no. 4, pp. 374–385, 1993.
[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.
[38] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.
[39] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16. Savannah, GA, USA, 2016, pp. 265–283.
[40] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—An open source framework for automatic control and dynamic optimization," *Opt. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, 2011.
[41] K. Zhang, J. Sprinkle, and R. G. Sanfelice, "Computationally aware control of autonomous vehicles: A hybrid model predictive control approach," *Auton. Robots*, vol. 39, no. 4, pp. 503–517, 2015.
[42] M. Ishii and A. Sato, "Layer-wise weight decay for deep neural networks," in *Proc. Pac.-Rim Symp. Image Video Technol.*, Wuhan, China, 2017, pp. 276–289.
[43] S. Karsoliya, "Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture," *Int. J. Eng. Trends Technol.*, vol. 3, no. 6, pp. 714–717, 2012.

Maral Amir (GS'15) received the first M.S. degree in embedded electrical and computer systems from San Francisco State University, San Francisco, CA, USA, in 2014, and the second M.S. degree in computer science from the University of California at Irvine, Irvine, CA, USA, in 2018, where she is currently pursuing the Ph.D. degree in computer science.

Her current research interests include model reconfiguration/compression methods for resource-aware cyber-physical systems and machine learning applications.

Tony Givargis (SM'98) received the B.S. and Ph.D. degrees in computer science from the University of California at Riverside, Riverside, CA, USA, in 1997 and 2001, respectively.

He is a Professor of Computer Science with the School of Information and Computer Sciences, University of California at Irvine, Irvine, CA, USA. He has authored over 100 peer reviewed papers. He is a named inventor on 11 U.S. patents and has co-authored two popular textbooks on embedded system design. His current research interests include embedded systems with an emphasis on embedded and system software.

Prof. Givargis was a recipient of numerous teaching, service, and research awards, including the Frederick Emmons Terman Award, presented annually to an outstanding young electrical engineering educator by the Electrical and Computer Engineering Division of the American Society for Engineering Education.

Priority Neuron: A Resource-Aware Neural Network for Cyber-Physical Systems

Maral Amir, *Graduate Student Member, IEEE*, and Tony Givargis, *Senior Member, IEEE*

Abstract*—Advances in sensing, computation, storage, and actuation technologies have entered cyber-physical systems (CPSs) into the smart era where complex control applications requiring high performance are supported. Neural networks (NNs) models are proposed as a predictive model to be used in model predictive control (MPC) applications. However, the ability to efficiently exploit resource hungry NNs in embedded resource-bound settings is a major challenge. In this paper, we propose priority neuron network (PNN), a resource-aware NNs model that can be reconfigured into smaller subnetworks at runtime. This approach enables a tradeoff between the model's computation time and accuracy based on available resources. The PNN model is memory efficient since it stores only one set of parameters to account for various subnetwork sizes. We propose a training algorithm that applies regularization techniques to constrain the activation value of neurons and assigns a priority to each one. We consider the neuron's ordinal number as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number in the layer. This imposes a relatively sorted order on the activation values. We conduct experiments to employ our PNN as the predictive model of a vehicle in MPC for path tracking. To corroborate the effectiveness of our proposed methodology, we compare it with two state-of-the-art methods for resource-aware NN design. Compared to state-of-the-art work, our approach can cut down the training time by 87% and reduce the memory storage by 75% while achieving similar accuracy. Moreover, we decrease the computation overhead for the model reduction process that searches for n neurons below a threshold, from $O(n)$ to $O(\log n)$.

***Index Terms*—Cyber-physical system, model predictive control (MPC), neural networks (NNs), resource-aware.**

I. Introduction

CYBER-PHYSICAL systems (CPSs) are composed of cyber and physical components in a feedback loop, where physical processes affect computations and vice versa [1]–[3]. With the recent developments in CPS, cloud computing, machine learning, and artificial intelligence technologies, it is just a matter of time before autonomous drivers replace humans on the road. Vehicles are now embedded with intelligent devices that enable the vehicle to respond to various factors and obstacles, sudden acceleration or braking, etc., in real-time. The control and prediction of system dynamics are important factors in autonomous driving [4], [5]. Model predictive control (MPC), also known as receding horizon control, is an advanced control method. MPC makes explicit use of a model of the physical system to estimate its behavior for a given stream of inputs in a predetermined prediction horizon. The predicted outputs depend on the past inputs/outputs, and the future control signals [6]. As shown in Fig. 1, these future control signals are calculated by the optimizer taking into account the cost function and enforced constraints. The cost function usually takes the form of a quadratic function of errors between the predicted output signal and the reference trajectory. In the standard approach, ordinary differential equations (ODEs) are employed as the predictive model to represent the dynamic behavior of a physical system. Iterative methods to approximate a solution for nonlinear ODEs have introduced challenges in the design of embedded MPCs in terms of scalability, performance, and power consumption [7].

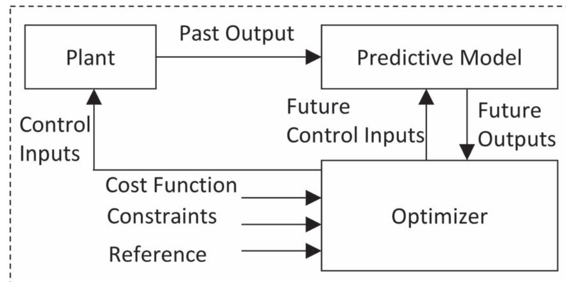

Fig. 1. MPC loop.

The computational overhead in traditional MPC grows exponentially with the length of the prediction horizon [8]. Research shows that a stable MPC controller requires a sufficiently large prediction horizon [9]. On the other hand, short prediction horizons are preferred for improved prediction accuracy of predictive models. This is because harmful effects of the poor estimates are amplified over a long prediction horizon time. Here, the problem is addressed by proposing an MPC approach that uses an adaptive prediction horizon with respect to quality measures [10]. However, the numerical effort needed in order to solve the optimal control problem for a long prediction horizon still remains significant. One

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. This work was supported by the National Science Foundation under NSF Grant 1563652. This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2018 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Maral Amir.)*

The authors are with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: mamir@uci.edu; givargis@uci.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857319

approach to overcome the computational burden of long horizon predictions is by implementing multirate prediction. In this approach, each look-ahead has a separate weight in the estimation of the steering input, where the furthest look-ahead point has the lowest weight [8].

Another method that is proposed to handle the computational issue associated with MPC systems is to use accelerated predictive models of the physical system. Different variants of neural networks (NNs) (e.g., recurrent NNs (RNNs) [11]) hold promising performance for time-series prediction as they can easily be built to predict multiple steps ahead all at once. These models are well-known to have the ability to learn linear and nonlinear relations between input and output variables without prior knowledge [12]. However, the use of NN models for long prediction horizon MPC problems could raise scalability and computational complexity challenges. The state-of-the-art methodologies are focused on reducing the size of the NN models without significantly affecting the performance [13]–[15]. These methodologies leverage the intrinsic error tolerance property of the NN models due to their parallel and distributed structure. Therefore, model reduction schemes could be exploited to employ the NN as the predictive model in the MPC loop. Several recent studies have focused on rescaling the size of the NN to adjust the resource usage on the embedded platform with respect to response time, power, and accuracy targets [16]. In other words, several sizes of the NN are available at runtime to manage resources for inference time-, safety-, and energy-constrained tasks. Moreover, continuous learning of NNs in data-driven modeling [17], transfer learning techniques [18], and adaptive modeling [19] impose significant training-time constraints at *runtime*.

A. Our Contribution

In this paper, we propose priority neuron network (PNN), a novel NN model that is featured with a reconfigurable architecture. Our objective is to design a resource-aware reconfigurable NN model that not only computes the future outputs as time series data in constant time, but is also memory efficient. The summary of our contributions in this paper are as follows.

1) We develop a reconfigurable NN model to fit the dynamic behavior of the physical systems for multistep-ahead prediction in receding horizon problems. Our resource-aware NN model can be reconfigured to various network sizes at runtime while storing only *one set of weight parameters* for memory efficiency.
2) We propose a training algorithm that controls the priority of each neuron in the computation of the model's output. We regulate the priority of each neuron using regularization techniques enforced on weight parameters. We consider the neuron's ordinal number as our priority criteria in that the priority of the neuron is inversely proportional to its ordinal number. We can reconfigure our NN model to smaller sizes by eliminating low priority neurons. This approach allows the tradeoff between the model's computation time and accuracy in resource-constrained systems.
3) We implement our reconfigurable NN model that contains multiple subnetworks using one-time training, hence reducing overall training time.
4) Our priority-based training algorithm enforces a sorted distribution on activation values of neurons. This helps to reduce the computation complexity of the model reduction process when searching for n neurons below the pruning threshold, from $O(n)$ to $O(\log n)$. It needs to be pointed out that we are not proposing a pruning methodology, but a memory efficient NN model that can be reconfigured to smaller sizes with less computation complexity at runtime.
5) We apply our method to train a three-layer fully connected NN model to be employed as the predictive model of a vehicle in MPC for path tracking application. We conduct closed-loop simulation of MPC using ODE predictive models to collect the training data. To evaluate the efficacy of our methodology, we compare it with two state-of-the-art approaches-Inc [20] and Big/Little [16]—that are targeted for resource-aware NN design in embedded systems. We show that our proposed PNN model outperforms the BL method with 89% reduction in training time and 78% saving in memory storage. The PNN model shows similar results to Inc method in terms of memory and model reduction complexity. However, we show that PNN follows a single training process to adjust weight parameters as opposed to Inc method that is based on multiple retraining. Therefore, the PNN model can cut down the training time by 86% with respect to Inc method while maintaining a better prediction performance from 0.25% to 0.21%.

The rest of this paper is organized as follows. In Section II, we summarize the state-of-the-art approaches to solve the computational complexity of MPC systems and design resource-efficient NN models. We describe our proposed method in Section III. We demonstrate the effectiveness of our framework for path following application in Section IV. Finally, we give our conclusions in Section V.

II. Background and Related Work

Advanced control methodologies have emerged for path planning and path following applications in modern vehicles. Nonlinear MPC is leveraged to develop path following control systems while handling model uncertainties, constraints and nonlinearities. A predictive model of the physical plant is used to estimate the future outputs for a prediction horizon within a window of time and with respect to known input and output values (Fig. 1). Mathematical descriptions in the form of ODEs are used to model the linear/nonlinear behavior of the physical system [21]. ODE solvers are applied to estimate solutions that converge to the exact solution of an equation or system of equations [22]. A runtime optimization routine is evaluated as a parametric quadratic function to calculate a set of future control inputs subject to constraints enforced by the environment and system dynamics. These routines are computationally intensive, and for nonlinear physical models, the computational overhead grows with complexity of the model [23].

One of the challenges in classic MPC is that the computational overhead increases with the length of the prediction horizon [8]. One approach to overcome the computational burden of long horizon predictions is by implementing a multirate prediction control strategy, where the prediction horizon is sampled in nonequidistant way [24]. In this approach, for a determined prediction horizon of n time steps, the initial steps have a shorter sampling period than the ones in the more distant future. In other words, fine tuning the control in such a way as to reduce the importance of predictions that contribute to time steps further in the future. Novel approaches are proposed for nonlinear dynamic system modeling and identification, where the NN realizes the behavior of a set of ODEs with smaller computation overhead [12], [25]. Moreover, data-driven NNs are increasingly in demand. Data-driven NNs are based on direct use of input-output observations collected from various real-world processes to perform system optimization, control and/or modeling [26]. Classic NNs have a three-layer structure, namely input, hidden, and output layers. Each layer contains a set of neurons with edges to pass the information. The edges entering the neurons are associated with weight parameters. The weight parameters are adjusted in a training algorithm (e.g., by back propagation) so that the difference between the network's prediction and the target output is minimized.

Developing resource-efficient NNs for embedded systems with limited hardware resources is a challenging task. To solve the memory complexity of NN models, many model compression approaches are proposed based on the claim that NN models have natural error tolerance because NNs usually contain more neurons than necessary to solve a given problem [27]. Many network pruning and model reduction techniques are proposed in the previous work with promising results [28]–[30]. However, finding an optimal pruning solution is NP-hard and requires a costly retraining process [31]. Many works have focused on selecting weight parameters for pruning based on criteria such as magnitude of the weight, activation value for the respective neuron, and increase in training error [32]–[34]. Han *et al.* [35] proposed an iterative pruning method that removes all neuron connections whose weight is lower than a certain threshold. This approach converts a dense fully connected layer into a sparser layer. The pruning is followed by a retraining process to boost the performance of the trimmed NN. A common approach to reduce the size of the "parameter intensive" fully connected layers is to reduce the magnitude of the overall weight parameters by including regularization terms in the model's cost function. Pan *et al*. [15] exploited regularization terms during the training process to simplify the NN model. At the end of the training, the NN is trimmed by dropping neurons below a certain threshold.

Another approach to address resource-constrained deployment of NNs for embedded systems is to adapt the size of the NN model to the performance requirements. Park *et al.* [16] addressed the energy complexity of NNs using a novel big/little implementation, whereby a score margin metric is employed to select between the two sizes. This approach is memory intensive such that it requires storing separate sets of weights for different sizes of NNs. Tann *et al.* [20] addressed the memory complexity problem by proposing a multistep incremental training algorithm such that the weights trained in earlier steps are fixed. In this method, multiple subnetworks with different sizes are formed while storing and using only one sets of weight parameters. Although this approach is close to ours, our proposed method is more computationally flexible in generating multiple subnetwork sizes and does not suffer from a time-consuming retraining process. In the following section, we describe PNN, our proposed reconfigurable NN model and its training algorithm.

III. Method

A. Application of Neural Networks in Model Predictive Control

MPC exploits a predictive model of the physical system to produce an optimized control input sequence. The predictive model computes the output of the system, a number of time steps into the future based on the current output and future control input values. Therefore, the predictive model to estimate future outputs at time k in the next n time steps—$Y(k+n|k)$—can be formulated as a time series prediction function f of future control inputs $I(k+n|k)$ and a vector of current state variables $S(k|k)$ for $S = [S_0, S_1, \ldots, S_{N_s}]$. Time-series data is a sequence of time-ordered values as measurements of some physical process [36]

$$Y(k+n|k) = f(\mathbf{S}(\mathbf{k}|\mathbf{k}), I(k+n|k)). \quad (1)$$

The prediction function in (1) can be fitted in a multiple input multiple output NN model with future control inputs and current state of the physical system as its input features and the future outputs in the next n time steps as its target outputs. Once the function is learned, the acyclic NN model computes the future outputs as a time-series data in *constant computing time* [12]. We use a three-layer fully connected feed-forward NN (FFNN) to fit (1) and approximate the dynamic behavior of the physical system. The FFNN is a class of NNs, where the input signal feeds forward through the network layers to the output in a single direction. Here, each layer of the network consists of computing neurons with edges that typically have a weight parameter. The output $\hat{y}_i$ of the NN model can be computed as follows given x_k input features for $i \in \{1 \cdots N_o\}$ and $k \in \{1 \cdots N_i\}$:

$$\hat{y}_i = \sum_{j=1}^{N_h} \left[w_{ji}^2 \, \sigma\left(\sum_{k=1}^{N_i} w_{kj}^1 \mathbf{x}_\mathbf{k} + \theta_j^1 \right) + \theta_i^2 \right] \quad (2)$$

where N_i, N_h, and N_o denote the numbers of input-layer, hidden-layer, and output-layer neurons, respectively. The parameters w_{kj}^1 and w_{ji}^2 are weights connecting the first layer to hidden layer and connecting the hidden layer to the output layer, respectively, and are adjusted in the learning process. The threshold offsets for the hidden and output layers are represented as θ^1 and θ^2. The function $\sigma(.)$ represents an activation functions, e.g., sigmoid, or rectified linear unit (ReLU), that limits the variation to output values with respect to changes in NN parameters.

B. Architecture of Priority Neuron Neural Network As Predictive Model in MPC

We propose PNN, a resource-aware reconfigurable NN such that the full model can be reconfigured to smaller sizes for less computation time and relatively comparable accuracy. Here, we deploy our proposed NN model for multistep ahead time-series prediction in constant time for an MPC application. However, the proposed NN model can be generalized for other prediction applications, e.g., computer vision. As stated in Section III-A, the nonlinear model in (1) is used by MPC to compute future behavior of the physical system can be fitted into a three-layer fully connected FFNN. The future control inputs and current state of the physical system are given as the input features to the FFNN to approximate the future outputs in the next n time steps. The proposed NN model can be described as in (2) for N_i = (# of state variables(N_s) + N_o) and $N_h = N_o$ = (# of time steps in the prediction horizon(n)). The value for N_h is set empirically equal to N_o. We have two weight matrices W^1 and W^2 with sizes $(N_i \times N_h)$ and $(N_h \times N_o)$ containing connecting weights of our hidden and output layers, respectively. We use the ReLU activation function which is one of the most widely used activation functions and is defined as

$$\sigma(z) = \max(0, z). \tag{3}$$

During the prediction process of the NN, we would ideally want a few neurons in the network to not activate, thereby making the activations sparse and efficient. The ReLU activation function gives us the ability to design a sparser NN model because it outputs 0 for negative input values and imposes no constraint on the positive inputs. Equation (2) is broken down into (4a) and (4b) to compute the outputs of hidden and output neurons, respectively. Here, for brevity, the bias parameters are deleted

$$h_j = \sigma\left(\sum_{k=1}^{N_i} w_{kj}^1 \mathbf{x_k}\right) \tag{4a}$$

$$\hat{y}_i = \sum_{j=1}^{N_h}\left(w_{ji}^2\ h_j\right). \tag{4b}$$

Hereafter, we are seeking a methodology for an architecture of an NN that stores one set of weight parameters yet can be reconfigured to smaller sizes of the NN with small drop in accuracy. To adopt the reconfigurability feature in our model, we exploit the multirate prediction idea suggested by [8] that assigns lower accent to further look-ahead points in the computation of the future dynamic behavior of the system. Therefore, the proposed PNN model follows a sequential priority-based architecture. This means we consider the neurons' ordinal numbers as our priority criteria such that the priority of each neuron is inversely proportional to its ordinal number in the given layer. Therefore, the model can be reduced starting from the neuron with the highest ordinal number. Our goal is to synchronize the priority level of the output and hidden neurons so that the model reduction process is more computationally efficient for runtime applications. We will elaborate more on this in Section III-D. In Fig. 2, we show the architecture of the proposed PNN as a three-layer FFNN where higher priority neurons are colored darker. We can deploy PNN as a resource-aware predictive model for closed-loop MPC to estimate the future outputs $[Y_0, Y_1, \ldots, Y_{N_h}]$. Here, we use the future control inputs $[I_0, I_1, \ldots, I_{N_h}]$ and current state variables $[S_0, S_1, \ldots, S_{N_s}]$ as input features. In the following section, we describe our proposed training algorithm and the associated cost function to develop the priority-based NN model.

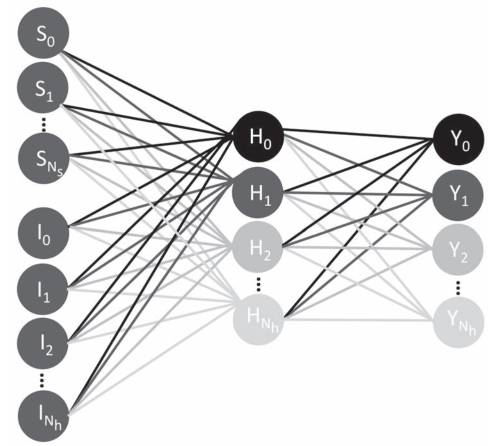

Fig. 2. PNN model.

C. Training Algorithm to Prioritize Neurons

During the training process of an NN, an optimization algorithm is exploited to minimize an objective function $E_0(.)$, which is simply a mathematical function based on the model's learning parameters (e.g., weights and biases). We might use sum of the squared deviations of our neuron's output $\hat{y}_i$ from the target output y_i as the loss function for N_o number of outputs denoted as

$$E_0(w, b) = \frac{1}{2N_o}\sum_{i=1}^{N_o}\left(y_i - \hat{y}_i\right)^2. \tag{5}$$

The learning parameters are optimized and updated in an iterative training process toward a solution that minimizes the loss function. A learning rate η is assigned to the training algorithm that determines the size of the steps we take at each iteration to reach a (local) minimum. For a convex optimization problem like this, we use derivatives of the loss function ∇E. Therefore, the following updating rule is formulated for the weight parameters to be updated after (t+1)th update iteration:

$$w^{t+1} \leftarrow w^t - \eta \nabla E_0. \tag{6}$$

For our optimization algorithm, we employ a variant of gradient descent called adaptive moment estimation (Adam) [37] which computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. In the proposed PNN model, the priority of the neuron determines how important the value of that neuron is in the overall performance of the NN. In order to control the priority of each neuron, we enforce constraints on the computation of its output value. This can be done through regularization techniques that restrain the growth of weight parameters. From (4), we see that the weight parameters used to compute the hidden neuron h_j are $W^1[:, j] = [w^1_{1j}, w^1_{2j}, \ldots, w^1_{N_{ij}}]$. The output neuron $\hat{y}_i$ is computed using weight parameters $W^2[:, i] = [w^2_{1i}, w^2_{2i}, \ldots, w^2_{N_h i}]$. We call the weight parameters of each neuron its *associated weights*.

1) Regularization: A common approach to reduce the complexity and size of NN models is to constrain the magnitude of the overall weight parameters by including regularization terms in the model's cost function. The $L1$ norm is one of the most commonly used regularization techniques that penalizes weight values by adding the sum of their absolutes to the error term. Therefore, the cost function E with the $L1$ regularization term is

$$E(w, b) = E_0(w, b) + \frac{1}{2}\lambda \sum_{l=1}^{2} \sum_{i=1}^{N_l} |W_i^l| \tag{7}$$

where λ is the weight decay coefficient for which larger values lead to larger cost, and causes the training algorithm to generate small weight values. Existing work sets the same weight decay coefficient for all layers to avoid the computational costs required to manually fine-tune each coefficient. However, to train our priority-based NN model, we penalize each weight with a specific weight decay coefficient so that the value of the corresponding weight is constrained to grow up only to a desired threshold point. Hence, the activation of each neuron is governed by the weight decay coefficients of its *associated weights*. As shown in Algorithm 1, we use a new cost function for our three-layer fully connected feed-forward PNN

$$E(w, b) = E_0(w, b) + \frac{1}{2}\sum_{k=1}^{N_i}\sum_{j=1}^{N_h} |\lambda^1_{kj} w^1_{kj}| + \frac{1}{2}\sum_{j=1}^{N_h}\sum_{i=1}^{N_o} |\lambda^2_{ji} w^2_{ji}| \tag{8}$$

for $\lambda^1 \in \Lambda^1$ and $\lambda^2 \in \Lambda^2$, where Λ^1 and Λ^2 are two weight decay matrices of our hidden and output layers, respectively. Therefore, the new updating rule for weight parameters is

$$w^{t+1} \leftarrow w^t - \eta\Big(\nabla E_0 + \Lambda^1 W^1 + \Lambda^2 W^2\Big). \tag{9}$$

In the following section, we describe our heuristic algorithm used to assign values to weight decay coefficients such that a sorted priority-based architecture is enforced on the proposed NN model.

D. Model Reconfiguration of PNN Model

In PNN, we want to force a priority onto each neuron during the computation of model output so that the *accuracy is maintained* after reconfiguring the network to smaller subnetworks by removing low priority neurons. Therefore, we consider larger weight decay coefficients for *associated weights* of neurons that are desired to have lower level of priority and vice versa. We are following the multirate prediction scheme that allocates less stress on accuracy of further look-ahead points. We design our weight decay matrices so that a sorted priority-based architecture for our PNN is developed during the training process. The intuition behind the sorted priority-based architecture of the PNN is to reduce the complexity of the model reconfiguration and reduction process. Model pruning approaches to constrain the complexity of NN models by applying regularization techniques, have been around for a while [28], [38]. These approaches are based on an exhaustive search process to remove neurons with activation values below a certain threshold. In our proposed priority-based architecture, we enforce a sorted priority on hidden neurons to compute the overall performance of the model. This helps reduce the time complexity for searching neurons below a certain activation value as we can employ a binary search algorithm. Therefore, the worst-case time complexity for the model pruning process in our PNN model with n number of hidden neurons is $O(\log n)$ as opposed to standard architectures that require $O(n)$ worst-case time complexity to prune the network. Moreover, the model can be reduced to smaller subnetworks at constant time $O(1)$ due to its reconfigurability feature that is adopted throughout the training process.

There is always a tradeoff between the number of subnetworks and the accuracy of the model. We assign the same level of priority to the number of neurons that are deleted at each level of model reduction. We call this number the *priority size* and denote it as p. Fig. 3 illustrates the reconfiguration process of the original NN model where neurons are sorted and colored in terms of priority and importance. At each level of reconfiguration, p number of hidden neurons with the least level of priority are deleted from the end of the hidden layer. Hence, their input and output weight connections are also removed from the weight space of the NN. These

Algorithm 1: Priority Neuron Training Algorithm

Input: input features - x
Input: output targets - y
Output: trained NN - *PNN*
Output: estimated outputs - $\hat{y}$

```
// initialize NN weights
```
init_random W
```
// estimate outputs given W weights
```
$\hat{y} = PNN\ (x)\ [W]$
```
// evaluate residual error
```
$err = \sum_{i=0}^{N_o} (y_i - \hat{y}_i)^2$
```
// evaluate regularization penalty
```
$reg = \sum |\Lambda^1_{N_i \times N_h} . W^1_{N_i \times N_h}| + \sum |\Lambda^2_{N_h \times N_o} . W^2_{N_h \times N_o}|$
```
// evaluate loss function
```
$loss = err + reg$
```
// optimize W weights for minimal loss
```
$\overline{W} =$ **AdamOptimizer** (*loss*)
```
// estimate outputs given optimal W̄
```
$\hat{y} = PNN\ (x)\ [\overline{W}]$
return $[PNN, \hat{y}]$

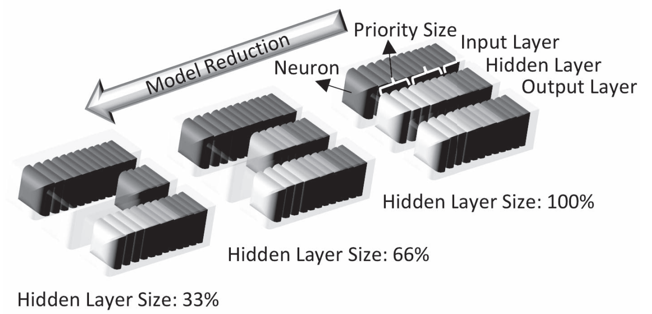

Fig. 3. Model reduction process for a three-layer fully connected NN with priority size $p = 4$.

W^1_{00}	W^1_{01}	W^1_{02}	$\cdots$	$W^1_{0N_h}$
W^1_{10}	W^1_{11}	W^1_{12}		$W^1_{1N_h}$
$\vdots$				
$W^1_{N_s0}$	$W^1_{N_s1}$	$W^1_{N_s2}$		$W^1_{N_sN_h}$
$W^1_{N_{s+1}0}$	$W^1_{N_{s+1}1}$	$W^1_{N_{s+1}2}$		$W^1_{N_{s+1}N_h}$
$W^1_{N_{s+2}0}$	$W^1_{N_{s+2}1}$	$W^1_{N_{s+2}2}$		$W^1_{N_{s+2}N_h}$
$W^1_{N_{s+3}0}$	$W^1_{N_{s+3}1}$	$W^1_{N_{s+3}2}$		$W^1_{N_{s+3}N_h}$
$\vdots$			$\ddots$	
$W^1_{N_i0}$	$W^1_{N_i1}$	$W^1_{N_i2}$		$W^1_{N_iN_h}$

Fig. 4. Weight parameters of hidden layer.

W^2_{00}	W^2_{01}	W^2_{02}	$\cdots$	$W^2_{0N_o}$
W^2_{10}	W^2_{11}	W^2_{12}		$W^2_{1N_o}$
W^2_{20}	W^2_{21}	W^2_{22}		$W^2_{2N_o}$
$\vdots$			$\ddots$	
$W^2_{N_h0}$	$W^2_{N_h1}$	$W^2_{N_h2}$		$W^2_{N_hN_o}$

Fig. 5. Weight parameters of output layer.

subnetworks can be deployed separately while reducing the memory complexity to a single network. In other words, only one set of weight parameters are stored for multiple subnetworks of different sizes. We consider neuron's ordinal number as our priority criteria which can be mapped into index values for neuron's associated weights. Therefore, the weight decays vary with respect to row and column indices of the weight matrix where r and c denote the row and column indices, respectively. Equations (10) and (11) are expanded from (4). In (11), we see N_o number of output formulas that are used to estimate the future output behavior of the physical system in the next N_o time steps, hence the size of the prediction horizon is N_o. It needs to be noted that, here we do not include the bias terms for simplification purposes

$$h_0 = w^1_{00}s_0 + w^1_{10}s_1 + \cdots + w^1_{N_i0}I_{N_i} \tag{10a}$$
$$h_1 = w^1_{01}s_0 + w^1_{11}s_1 + \cdots + w^1_{N_i1}I_{N_i} \tag{10b}$$
$$\cdots$$
$$h_{N_h} = w^1_{0N_h}s_0 + w^1_{1N_h}s_1 + \cdots + w^1_{N_iN_h}I_{N_h} \tag{10c}$$

$$y_0 = w^2_{00}h_0 + w^2_{10}h_1 + \cdots + w^2_{N_h0}h_{N_h} \tag{11a}$$
$$y_1 = w^2_{01}h_0 + w^2_{11}h_1 + \cdots + w^2_{N_h1}h_{N_h} \tag{11b}$$
$$\cdots$$
$$y_{N_o} = w^2_{0N_o}h_0 + w^2_{1N_o}h_1 + \cdots + w^2_{N_hN_o}h_{N_h}. \tag{11c}$$

Let us assume that the model is trained for a priority-based architecture where the priority of neurons decreases inversely with their ordinal number. For a pretrained model with priority size $p = 1$, we want to reduce the size of the model by removing hidden neuron h_{N_h} with the least priority level from the hidden layer. While removing the hidden neuron h_{N_o}, its associated weight connections $W^1[:, N_h] = [w^1_{0N_h}, w^1_{1N_h}, \ldots, w^1_{N_iN_h}]$ and $W^2[N_h, :] = [w^2_{N_h1}, w^2_{N_h2}, \ldots, w^2_{N_h(N_o-1)}]$ are removed from W^1 and W^2, respectively. In the next section, we describe the selection of weight decay coefficients to enforce a sorted priority on hidden and output neurons. For a simple implementation we use the same number of hidden and output neurons. Therefore, the W^2 weight matrix is squared.

E. Decay Matrix

A graphical illustration of our W^1 and W^2 weight matrices for hidden and output layers with $p = 1$ is shown in Figs. 4 and 5, respectively. The weight matrices in Figs. 4 and 5 are darker colored based on the value of their corresponding weight decay coefficients. This helps to visualize the selected distribution pattern for weight decay coefficients where a priority-based architecture for our PNN model is developed. In order to maintain the accuracy of the model after the removal of hidden neuron h_{N_h} [computed in (10c)], we want the model reduction to affect the least number of output neurons possible. Therefore, we seek to adjust the weight parameters so that removing the hidden neuron h_{N_h} mostly impacts the least priority output neuron y_{N_o}. Hence, we select weight decay coefficients for the weight parameters in the vector $[w^2_{N_h1}, w^2_{N_h2}, \ldots, w^2_{N_hN_o}]$ in a descending order so that the least weight decay value is assigned for $w^2_{N_hN_o}$. Smaller weight decay coefficients push the training algorithm to assign greater values for the weight parameters. In this method, we try to zero out $[w^2_{N_h1}, w^2_{N_h2}, \ldots, w^2_{N_h(N_o-1)}]$ as much as possible such that the removal of h_{N_h} has minimal impact on the values $[y_1, y_2, \ldots, y_{(N_o-1)}]$.

To expand this idea to other neurons in the hidden layer, we should change the weight decay coefficients above the main diagonal of W^2, in descending order per column and in ascending order per row, so that the least weight decay coefficients are placed on the main diagonal. Moreover, we should adjust the weight decay coefficients below the main diagonal of W^2 in ascending order per column and in a descending order per row. We use ascending order per column so that the priority level of output neurons decreases for larger ordinal numbers and descending order per row forces the weight parameters on the diagonal to contribute the most to the computation of their corresponding output neuron. We propose (12) to compute the weight decay coefficient for each weight parameter in order to regulate the sorted priority order of PNN neurons. Here, r and c denote the row and column index of the weight matrix, respectively. The parameter p stands for the number of neurons deleted at each model reduction process, hence the *priority size*

$$f(x) = \begin{cases} \left[\lambda_{rc} : \lambda_{r(c+p)}\right] = \beta f\left(\frac{r}{c}\right), & r \geq c \\ \left[\lambda_{rc} : \lambda_{(r+p)c}\right] = \beta f\left(\frac{c}{r}\right), & r < c. \end{cases} \quad (12)$$

Here, $f(.)$ can be considered as a linear, exponential, or logarithmic, etc. growth function considering the target application. The type of function $f(.)$ determines the variance of the priority distribution among various neurons at each layer. The greater the variance of the priority distribution is, the more ways the original NN can be reconfigured into subnetworks. That means less neurons (p) are deleted per model reconfiguration (reduction) process. Larger variance for the priority order of neurons decreases the model accuracy as it enforces more constraints on weight parameters. Therefore, the function $f(.)$ is assigned based on design requirements of the target application and the tradeoff between the model accuracy and number of subnetworks embedded in one NN model. The parameter β maps the computed value of weight decay from (12) to a range as $\lambda \in [\lambda_{\min}:\lambda_{\max}]$. This range is empirically selected based on the tradeoff between the model accuracy and the number of hidden neurons deleted per reconfiguration of the model-priority size. For our future work, we plan to automate the optimal selection of ranges for the weight decay coefficient.

F. Other Types of Neural Networks

The proposed priority-based approach is applied to a fully connected FFNN architecture. This is because state-of-the-art methods proposed fully connected FFNN as a predictive model to approximate dynamic behavior of physical systems in an MPC application. Previous state-of-the-art approaches has mostly focused on reducing the size of the fully connected layers in other NN architectures because these layers are well known to be parameter intensive and occupy more than 90% of the model size [15]. Another popular architecture of NNs for time series forecasting is RNN which is distinguished from FFNN by having signals traveling in both directions and introducing loops in the network. The RNN architecture can be converted into an FFNN by unfolding over time [11]. Therefore, in our future work, we plan to expand our method to other NN architectures. Although we evaluate the effectiveness of our methodology for MPC applications, it can be generalized to other applications of NN models.

IV. Experimental Results

A. Experimental Setup

Our implementation is based on the TensorFlow framework [39] executed on a PC with a quad-core Intel Core i7 and 16 GB of DDR3 RAM. The MPC formulation is implemented in software using the ACADO Toolkit framework [40], which is open source software written in C++ for automatic control and dynamic optimization. To evaluate the efficacy of our proposed methodology, we exploit the PNN as a predictive

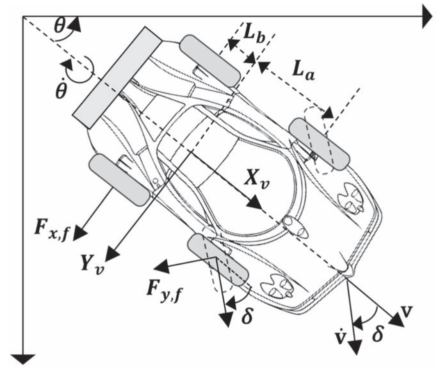

Fig. 6. Schematic of the vehicle model.

model in an MPC system for the path following application. We describe the process on how we collect our training dataset in the following section.

B. Simulation to Collect Training Data

As mentioned in Section II, the dynamic behavior of a physical system formulated as ODE can be fitted into a fully connected FFNN. The future control inputs and current state of the physical system are fed as the input features to the FFNN in order to predict the future outputs in the next n time steps. To collect the training dataset, we exploit the following ODE model of a vehicle [41] as shown in (13) and Fig. 6 to conduct offline simulation of MPC for a path following application:

$$\dot{s} = \begin{bmatrix} v\sin(\theta) \\ v\cos(\theta) \\ \cos(\delta)a - \frac{2}{m}F_{y,f}\sin(\delta) \\ \phi \\ \frac{1}{J}(L_a(ma\sin(\delta) + 2F_{y,f}\cos(\delta)) - 2L_bF_{y,r}) \\ \omega \end{bmatrix}. \quad (13)$$

Here, $s = [x, y, v, \theta, \phi, \delta]$ is the vector of state variables with acceleration a and steering angular speed ω as control inputs. The variables x and y stand for longitudinal and lateral positions, and v and θ are velocity and the azimuth. The variables δ and ϕ represent the steering angle and speed, respectively. The distance from sprung mass center of gravity to the front and rear axles are denoted as L_a and L_b, respectively, and J is the angular momentum. The variables $F_{y,f}$ and $F_{y,r}$ stand for front and rear tire lateral forces. These forces are computed from the following equations:

$$F_{y,f} = C_y\left(\delta - \frac{L_a\phi}{v}\right) \quad (14a)$$

$$F_{y,r} = C_y\left(\frac{L_b\phi}{v}\right) \quad (14b)$$

where C_y is the lateral tire stiffness. We applied real-world parameters of a 2011 Ford Fusion as $L_a = L_b = 1.5$ m, mass $m = 1700$ kg, and tire stiffness data for our experiments. The MPC formulation to follow the reference path x^r, y^r is the

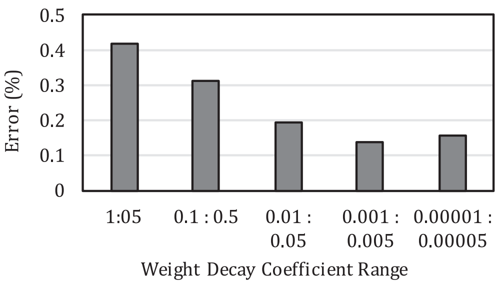

Fig. 7. Performance of PNN for different ranges of weight decay coefficients.

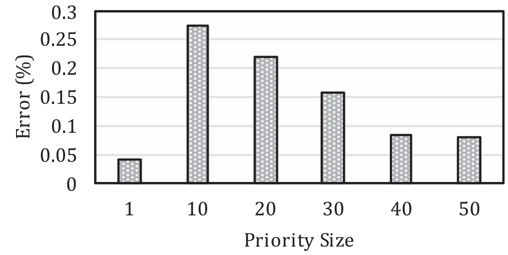

Fig. 8. Performance of PNN for different priority sizes.

solution to the following optimization problem:

$$\min_{x,y} \quad \sum_{t=0}^{T_p} \|\hat{x}(k+1|k) - x^r(k+1|k)\|^2_{Q_c} \tag{15a}$$

$$+ \|\hat{y}(k+1|k) - y^r(k+1|k)\|^2_{Q_c} \tag{15b}$$

$$\text{s.t.} \quad \delta_{\min} \leq \delta \leq \delta_{\max} \tag{15c}$$

$$\omega_{\min} \leq \omega \leq \omega_{\max} \tag{15d}$$

$$a_{\min} \leq a \leq a_{\max}. \tag{15e}$$

We simulate the MPC to predict 101 time steps in the future with time intervals of 5.05 s for a vehicle with an average speed of $v = 10$ (m/s). The appropriate value for the prediction horizon and step size is bounded by some factors such as stability and accuracy requirements and it varies based on plant dynamic characteristics. We implement an FFNN with input size $N_i = 6 + 102$ for six values of current state variables and future control inputs in the next 101 time steps. We select $N_o = 102$ as the output size for our NN to predict the future output of the physical system in the next 101 time steps. The number of hidden neurons in our three-layer FFNN are $N_h = N_o$.

C. PNN Training

In order to fine tune the range of weight decay coefficients $\lambda \in [\lambda_{\min}:\lambda_{\max}]$ and select an appropriate value for the constant factor β in (12), we empirically pick the values that yield the best performance on a held-out dataset. Therefore, we conducted experiments based on five different ranges of coefficients. Fig. 7 shows the error rate of the PNN model with respect to variations in the range of weight decay coefficients. The optimal range of weight decay coefficients for each layer may change with respect to the size of the next layer. In back propagation training, the gradient term in (9) is scaled with the size of the next layer [42]. Therefore, to compensate for the rescaling in the gradient term of the update rule, the optimal range for weight decay coefficients might change. These results are derived for priority size of $p = 10$, which denotes the number of hidden neurons that are removed at each reconfiguration of the model to a smaller subnetwork. Greater values of p restrict the original NN model to be reconfigured to less number of subnetworks. Naturally, there is always a tradeoff between the accuracy of the model and the number of subnetworks as shown in Fig. 8. Considering this tradeoff, the user might select an optimal priority size based on the design requirements for the target application. The error values in this figure are collected while reducing the size of the NN to 50% of its original size. A tradeoff still remains between the number of subnetworks with acceptable error values and the percentage at which the size of the model is reduced. With respect to the application and design requirements, the user may select the appropriate value for the hyper parameter p.

D. Comparison to State-of-the-Art Methodologies

We evaluate the performance of our methodology in training a resource-aware NN model with two state-of-the-art approaches that are proposed as solutions to implement resource efficient NN in embedded system. By using the notation *resource-aware NN model*, we are implying that these NN models are targeted for systems that monitor the resource usage and dynamically manage the allocated resources to the NN model with respect to runtime constraints. The results are collected for a three-layer fully connected NN of 108×102 and 102×102 inputs to its hidden and output layers, respectively. The *Big/Little* approach [16], suggests multiple implementations of an NN model with small to bigger sizes. In the *Incremental* method [20], which is the most similar to ours, the NN is trained based on an iteratively incremental training algorithm where the weights computed in the earlier steps are fixed. The *Big/Little* approach would require separate memory storage to hold model parameters of different sizes. Moreover, a retraining process is mandatory to generate multiple sizes for the NN model. The *Inc* method is more memory efficient such that only one set of model parameters are stored to implement an NN model that can be reconfigured into subnetworks with different sizes. However, this approach suffers from the retraining overhead per increment of size. In today's embedded systems, where runtime continuous learning of NNs is required, retraining process overhead is prohibitive [17]. Our proposed PNN model is memory efficient such that only one set of weights are computed for multiple subnetworks. Furthermore, we compute the model parameters for PNN in a single-training process. Throughout the examples, we use the following abbreviation to indicate the three models: 1) PNN: priority-based; 2) Inc: Incremental; and 3) BL: Big/Little.

Emerging research is based on developing approaches to estimate the number of neurons and hidden layers required for an NN [43]. However, these approximations also depend on the type of the database samples for which the network is designed. Therefore, it is still challenging to determine a good network topology for different applications. Therefore, exhaustive pruning and model reduction methodologies are in demand

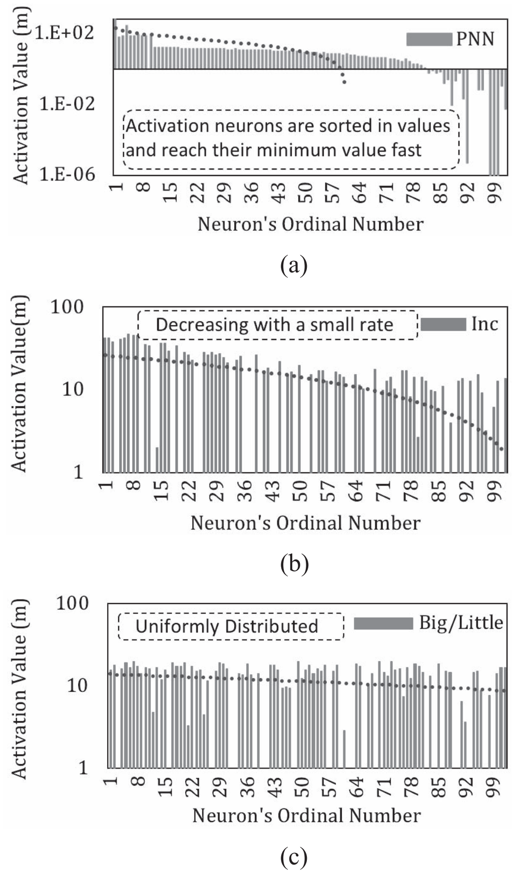

Fig. 9. Comparing activation values of neurons with respect to their ordinal number. Activation values for neurons in (a) PNN, (b) Inc., and (c) Big/Little.

to reduce the over-sized NN models. One advantage of our proposed priority-based training algorithm is that it enforces a relatively sorted distribution to the activation values. We compare the activation value of hidden neurons for our proposed PNN model with respect to the *incrementally* trained model and the *Big/Little* model that is trained with no constraint on its weight parameters in Fig. 9. For fairness of comparison, all experiments are conducted with the same size for all three models. The ordinal number of the neuron denotes the position of the respective neuron in the layer. The dotted red line shows the trend for linear changes in activation values with respect to ordinal number of the neuron. As shown in Fig. 9(a), the activation values for the hidden neurons in PNN with priority size $p = 10$ is following a sorted order. The trend line shows that the density of the model is mostly populated throughout the first neurons and the activation values for the neurons further in the end of the layer are forced to be very small. This is as opposed to the two other methods that show a more uniform distributions of activation values for the neurons. The incremental approach in Fig. 9(b) also shows slight sorted order among activation values. However, as represented by the trend line, the rate of change for neuron's activation value with respect to its ordinal number is very slow compared to PNN method. In other words, in incremental approach, the weight parameters are adjusted more uniformly throughout the network. This decreases the number of subnetworks and the number of hidden neurons that can be pruned from the model without major drop in accuracy.

TABLE I
COMPARING THE TRAINING PROCESS

Model	# of Sub-Networks	Retrain	# of Retrain	Train Time (s)
PNN	6	No	0	2627
Inc	6	Yes	6	21534
Big/Little	6	Yes	6	25020

Table I compares the training process for a three layer fully connected FFNN using the three aforementioned methods. The data is collected to train six separate subnetworks of various sizes using the three methods. As we can see in the table, our proposed method can generate six separate subnetworks in single training process. This is as opposed to the two other methods that require retraining for each of the subnetworks. The performance of these six subnetworks is evaluated in Fig. 10(a) and (b) where the x-axis represents the number of hidden neurons at each subnetwork. The retraining process imposes additional computation complexity to retune the parameters and hyper parameters. We can see that our proposed model reduces the computation overhead for the training process substantially. The training time is a critical matter especially in embedded systems for CPS applications where many NN models are trained on the fly.

In Fig. 10(a), we show the prediction time values over six different subnetwork sizes. The results show similar performance for all three approaches in terms of runtime prediction overhead which increases for larger network size. As shown in the figure, by reducing the number of hidden neurons to half of its original size, we can improve the computation overhead by 30%. However, this saving in computation time comes as a tradeoff for model accuracy. Fig. 10(b) shows the percentage prediction error values for different subnetwork sizes. The results for the BL [16] method that trains the subnetworks separately with no additional constraints show that after a certain point the model error does not change with growth in the NN size. This justifies the over-parameterization phenomena in training the NN that urges pruning and model reduction methodologies. Moreover, the mean of prediction error for six different subnetworks using our proposed PNN method and Inc. [20] are 0.2% and 0.25%, respectively. Therefore, our proposed PNN method outperforms the Inc approach for better prediction performance with no additional retraining process needed.

In order to evaluate the comparability of model accuracy among the three methods, we also show the probability distribution of prediction error values in Fig. 10(c). These results are collected for a full-size NN with no model reduction process performed. We can see in the figure that the low variation in prediction errors using our proposed PNN model, confirms its stable performance in prediction of various test data. Moreover, the average of prediction errors for the PNN model is very close to that of BL method. This experiment ensures that our proposed model is validated as a memory

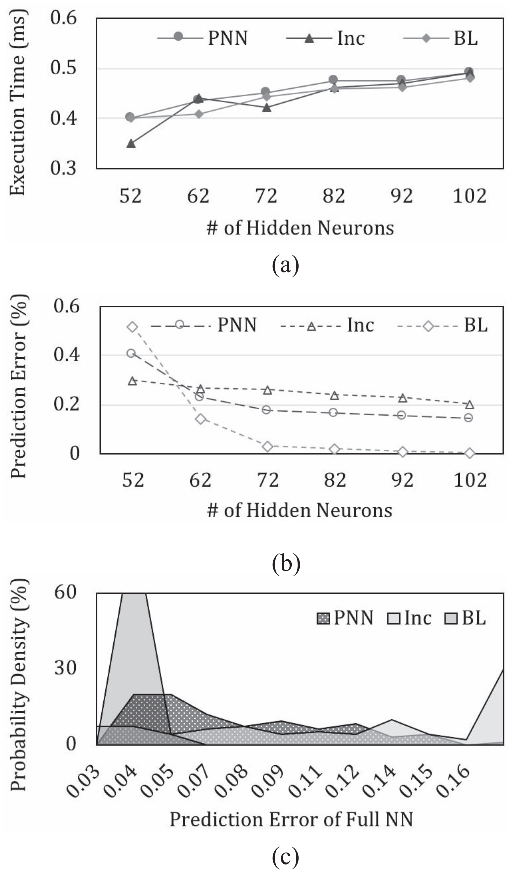

Fig. 10. Performance comparison of three resource-aware approaches. (a) Execution time. (b) Prediction error. (c) Probability distribution of prediction error for full-size NN.

TABLE II
COMPARING MEMORY REDUCTION WITH RESPECT TO ERROR

Model	# of Sub-Networks	# of Parameters	Memory Reduction	Mean Error (%)
PNN	6	21522	78%	0.2
Inc	6	21522	78%	0.25
Big/Little	6	87292	-	0.125

efficient architecture for NN models with small drop in accuracy and comparable performance can be acquired using all three methods.

We compare the efficiency of the three resource-aware methods in terms of memory requirements and model reduction complexity in Table II. The PNN and Inc methods are both memory efficient in that they need one set of weight parameters to store multiple subnetwork sizes. This is as opposed to the BL method that requires separate memory to store each subnetwork. Therefore, we can achieve 78% saving in memory to store six subnetworks with very small loss in accuracy.

To summarize, our proposed PNN model outperforms the BL method with 89% reduction in training time and 78% saving in memory storage. Moreover, the computation complexity of the model reduction process to search for n neurons below the pruning threshold is improved from $O(n)$ to $O(\log n)$. The PNN model shows similar results to Inc method in terms of memory and model reduction complexity. However, we show that PNN follows a single training process to adjust weight parameters as opposed to Inc method that is based on multiple retraining. Therefore, The PNN model can cut down the training time by 86% with respect to Inc method while maintaining a better prediction performance from 0.25% to 0.21%.

V. CONCLUSION

In this paper, we proposed PNN, a resource-aware NN model with a reconfigurable architecture. We proposed a training algorithm to exploit regularization constraints on each neuron based on their ordinal number at a given layer. This enforces a sorted order distribution for the activation value of the neurons. We implemented our model for a three-layer fully connected NN architecture to be employed as the predictive model of a vehicle in MPC for path tracking application. To corroborate the effectiveness of our proposed methodology, we compared it with two state-of-the-art methods for resource-aware NN design. We showed that compared to current state-of-the-art, our approach achieves 75% reduction in memory usage and 87% less training time with no significant drop in accuracy. Moreover, we improve the computational complexity of the model reduction process in order to prune n number of neurons, from $O(n)$ to $O(\log n)$.

REFERENCES

[1] F. Cicirelli, L. Nigro, and P. F. Sciammarella, "Model continuity in cyber-physical systems: A control-centered methodology based on agents," *Simulat. Model. Pract. Theory*, vol. 83, pp. 93–107, Apr. 2017.

[2] K. Vatanparvar and M. A. A. Faruque, "Eco-friendly automotive climate control and navigation system for electric vehicles," in *Proc. ACM/IEEE 7th Int. Conf. Cyber. Phys. Syst. (ICCPS)*, Vienna, Austria, 2016, pp. 1–10.

[3] K. Vatanparvar and M. A. A. Faruque, "Design and analysis of battery-aware automotive climate control for electric vehicles," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 4, p. 74, 2018.

[4] M. Amir and T. Givargis, "Hybrid state machine model for fast model predictive control: Application to path tracking," in *Proc. IEEE 36th Int. Conf. Comput.-Aided Design*, Irvine, CA, USA, 2017, pp. 185–192.

[5] M. Amir and T. Givargis, "HES machine: Harmonic equivalent state machine modeling for cyber-physical systems," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Santa Cruz, CA, USA, 2017, pp. 31–38.

[6] E. F. Camacho and C. Bordons, *Model Predictive Control in the Process Industry*, 2nd ed. London, U.K.: Springer-Verlag, 2007.

[7] A. N. Gorban and D. Roose, *Coping With Complexity: Model Reduction and Data Analysis*, vol. 75. Heidelberg, Germany: Springer, 2011.

[8] J. S. Barlow, "Data-based predictive control with multirate prediction step," in *Proc. IEEE Amer. Control Conf. (ACC)*, Baltimore, MD, USA, 2010, pp. 5513–5519.

[9] A. Jadbabaie and J. Hauser, "On the stability of receding horizon control with a general terminal cost," *IEEE Trans. Autom. Control*, vol. 50, no. 5, pp. 674–678, May 2005.

[10] G. Droge and M. Egerstedt, "Adaptive time horizon optimization in model predictive control," in *Proc. IEEE Amer. Control Conf. (ACC)*, San Francisco, CA, USA, 2011, pp. 1843–1848.

[11] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, *An Overview and Comparative Analysis: Recurrent Neural Networks for Short Term Load Forecasting*. New York, NY, USA: Springer, 2017.

[12] J. Fojdl and R. W. Brause, "The performance of approximating ordinary differential equations by neural nets," in *Proc. 20th IEEE Int. Conf. Tools Artif. Intell. (ICTAI)*, vol. 2. Dayton, OH, USA, 2008, pp. 457–464.

[13] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 6071–6079.

[14] A. Mujika, F. Meier, and A. Steger, "Fast-slow recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5915–5924.

[15] W. Pan, H. Dong, and Y. Guo, "DropNeuron: Simplifying the structure of deep neural networks," *arXiv:1606.07326 [cs, stat]*, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.07326

[16] E. Park *et al.*, "Big/little deep neural network for ultra low power inference," in *Proc. 10th Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Amsterdam, The Netherlands, 2015, pp. 124–132.

[17] K. Vatanparvar and M. A. A. Faruque, "ACQUA: Adaptive and cooperative quality-aware control for automotive cyber-physical systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Irvine, CA, USA, 2017, pp. 193–200.

[18] W. Hu, Y. Qian, F. K. Soong, and Y. Wang, "Improved mispronunciation detection with deep neural network trained acoustic models and transfer learning based logistic regression classifiers," *Speech Commun.*, vol. 67, pp. 154–166, Mar. 2015.

[19] W. He, Y. Chen, and Z. Yin, "Adaptive neural network control of an uncertain robot with full-state constraints," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 620–629, Mar. 2016.

[20] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth. (CODES+ISSS)*, Pittsburgh, PA, USA, 2016, pp. 1–10.

[21] K. Vatanparvar and M. A. A. Faruque, "Battery lifetime-aware automotive climate control for electric vehicles," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2015, pp. 1–6.

[22] C. T. Kelley, *Solving Nonlinear Equations With Newton's Method*. Philadelphia, PA, USA: SIAM, 2003.

[23] P. Krauthausen and U. D. Hanebeck, "A model-predictive switching approach to efficient intention recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Taipei, Taiwan, 2010, pp. 4908–4913.

[24] U. Halldorsson, M. Fikar, and H. Unbehauen, "Nonlinear predictive control with multirate optimisation step lengths," *IEE Proc. Control Theory Appl.*, vol. 152, no. 3, pp. 273–284, May 2005.

[25] N.-B. Hoang and H.-J. Kang, "Neural network-based adaptive tracking control of mobile robots in the presence of wheel slip and external disturbance force," *Neurocomputing*, vol. 188, pp. 12–22, May 2016.

[26] D. Solomatine, L. M. See, and R. J. Abrahart, "Data-driven modelling: Concepts, approaches and experiences," in *Practical Hydroinformatics*. Heidelberg, Germany: Springer, 2009, pp. 17–30.

[27] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[28] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[29] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.

[30] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, 2016, pp. 243–254.

[31] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4860–4874.

[32] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[33] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, vol. 2, 2017, pp. 1389–1397.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[35] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[36] Z. Tang and P. A. Fishwick, "Feedforward neural nets as models for time series forecasting," *ORSA J. Comput.*, vol. 5, no. 4, pp. 374–385, 1993.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[38] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[39] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16. Savannah, GA, USA, 2016, pp. 265–283.

[40] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—An open source framework for automatic control and dynamic optimization," *Opt. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[41] K. Zhang, J. Sprinkle, and R. G. Sanfelice, "Computationally aware control of autonomous vehicles: A hybrid model predictive control approach," *Auton. Robots*, vol. 39, no. 4, pp. 503–517, 2015.

[42] M. Ishii and A. Sato, "Layer-wise weight decay for deep neural networks," in *Proc. Pac.-Rim Symp. Image Video Technol.*, Wuhan, China, 2017, pp. 276–289.

[43] S. Karsoliya, "Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture," *Int. J. Eng. Trends Technol.*, vol. 3, no. 6, pp. 714–717, 2012.

Maral Amir (GS'15) received the first M.S. degree in embedded electrical and computer systems from San Francisco State University, San Francisco, CA, USA, in 2014, and the second M.S. degree in computer science from the University of California at Irvine, Irvine, CA, USA, in 2018, where she is currently pursuing the Ph.D. degree in computer science.

Her current research interests include model reconfiguration/compression methods for resource-aware cyber-physical systems and machine learning applications.

Tony Givargis (SM'98) received the B.S. and Ph.D. degrees in computer science from the University of California at Riverside, Riverside, CA, USA, in 1997 and 2001, respectively.

He is a Professor of Computer Science with the School of Information and Computer Sciences, University of California at Irvine, Irvine, CA, USA. He has authored over 100 peer reviewed papers. He is a named inventor on 11 U.S. patents and has co-authored two popular textbooks on embedded system design. His current research interests include embedded systems with an emphasis on embedded and system software.

Prof. Givargis was a recipient of numerous teaching, service, and research awards, including the Frederick Emmons Terman Award, presented annually to an outstanding young electrical engineering educator by the Electrical and Computer Engineering Division of the American Society for Engineering Education.