Topology Optimization of Neural Networks as an Integrated Process in Training with Control Theory Methods

Petr Dolezel
Faculty of Electrical Engineering and Informatics
University of Pardubice
Pardubice, Czech Republic
petr.dolezel@upce.cz

Dominik Stursa
Faculty of Electrical Engineering and Informatics
University of Pardubice
Pardubice, Czech Republic
dominik.stursa@upce.cz

Daniel Honc
Faculty of Electrical Engineering and Informatics
University of Pardubice
Pardubice, Czech Republic
daniel.honc@upce.cz

Josef Rak
Faculty of Electrical Engineering and Informatics
University of Pardubice
Pardubice, Czech Republic
josef.rak@upce.cz

Simeon Karamazov
Faculty of Electrical Engineering and Informatics
University of Pardubice
Pardubice, Czech Republic
simeon.karamazov@upce.cz

Dusan Kopecky
Faculty of Electrical Engineering and Informatics
University of Pardubice
Pardubice, Czech Republic
dusan.kopecky@upce.cz

***Abstract*—The simultaneous optimization of neural network topology and training remains an underexplored research direction, despite its potential to improve model efficiency and performance dynamically. This paper introduces a control-based framework for jointly adjusting the structure and training process of fully connected neural networks. The methodology formulates the training and pruning process as a multivariable dynamic system with two input variables—training process parameters and network architecture adjustments—and two output variables—model performance and computational complexity. A discrete two-dimensional Proportional-Integral-Derivative (PID) controller is employed to regulate these inputs, ensuring a balanced trade-off between accuracy and computational efficiency. The control system is tested on a function approximation task, where a fully connected network is initially set with redundant capacity and gradually optimized according to predefined reference trajectories of performance and complexity. Experimental results demonstrate the effectiveness of the proposed approach, revealing the dynamic interaction between topology and training in real-time network adaptation. The findings highlight the feasibility of integrating control strategies into neural network optimization and pave the way for future research on more advanced control-based learning architectures.**

***Index Terms*—neural network pruning, topology optimization, control**

I. Introduction

The rapid growth of neural network architectures has brought significant advancements in various domains, yet it has also raised challenges concerning their computational cost and efficiency. This challenge is particularly evident in resource-constrained environments, where large-scale neural networks must operate on end devices such as mobile phones or embedded systems. Network pruning, a popular method to reduce the size of neural networks while maintaining acceptable performance, has emerged as a promising solution [1]–[3].

State-of-the-art pruning techniques can be broadly categorized into unstructured and structured methods. Unstructured pruning focuses on reducing the number of connections in a network, whereas structured pruning removes entire components such as neurons, filters, or layers [4], [5]. Recent advancements include the application of neural topology search [6], evolutionary algorithms [7], and deep reinforcement learning [2], [8] to guide the pruning process.

Unstructured pruning methods, such as weight pruning, aim to identify and remove individual connections in a network based on their magnitude [4]. These approaches, while effective in reducing model size, often result in irregular sparsity patterns, which can be challenging to leverage in practice due to hardware inefficiencies. Structured pruning methods address this limitation by removing entire neurons, channels, or layers, thus maintaining architectural regularity [5]. Examples of structured pruning include channel pruning [9] and filter pruning [3], which rank and remove components based on predefined criteria such as ℓ_1-norms or importance scores. These methods, however, often rely on heuristics and require extensive hyperparameter tuning, limiting their

generalizability.

Reinforcement learning-based pruning methods have emerged as a promising alternative to traditional approaches. For instance, authors in [2] employ a deep reinforcement learning agent to determine the optimal pruning ratios for each layer. The agent is trained on a specific neural network and dataset, allowing it to learn a compression strategy that is tailored to the task. However, reliance on task-specific training limits the scalability of these methods to new architectures or datasets. Similarly, methods such as those proposed by [10] leverage global ranking techniques to improve pruning decisions. Although these approaches address some of the limitations of layer-specific methods, they remain computationally expensive and often require significant manual intervention.

The neural architecture search (NAS) represents another avenue for optimizing neural networks. NAS methods aim to automate the design of network architectures by exploring a predefined search space [11]. Recent works have combined NAS with pruning techniques to identify optimal topologies while simultaneously reducing model complexity [8]. Despite their potential, these methods are often computationally prohibitive, requiring substantial resources to search the architectural space effectively. In addition, the search process is typically decoupled from the training phase, leading to inefficiencies.

The NEON framework introduced by [1] represents a significant step forward in deep reinforcement learning-based pruning. NEON trains a deep reinforcement learning agent on a diverse set of architectures and datasets, enabling it to generalize to previously unseen networks. By incorporating a feature map-based representation of neural architectures, NEON overcomes the limitations of local pruning strategies and provides a global perspective on network optimization. Additionally, NEON introduces a novel reward function that allows users to specify their desired trade-offs between model compression and accuracy preservation. However, NEON's primary focus is on dense networks, leaving its applicability to other architectures, such as recurrent or graph neural networks, an open question.

Despite these advancements, the integration of pruning into the training process remains largely unexplored. Current methods typically treat pruning as a separate post-training step, requiring iterative fine-tuning to recover lost accuracy. This decoupled approach not only increases computational overhead but also limits the ability of the network to dynamically adapt its structure during training. To address this gap, we propose a novel framework that integrates pruning into the training process, guided by principles from multivariable system control. By simultaneously optimizing the network topology and parameters, our method enables more efficient resource utilization and real-time adaptability.

In summary, while existing pruning methods have made significant strides in improving the efficiency of neural networks, they are limited by their reliance on predefined strategies, separate optimization phases, and a lack of adaptability to new tasks. Our work seeks to bridge these gaps by introducing a control-theoretic approach to simultaneous pruning and training, representing a novel contribution to the field of neural network optimization.

The main contributions of this article are summarized as follows.

- We define the process of simultaneous neural network training and pruning as a multivariable dynamic system with two inputs and two outputs. This formulation enables the integration of the process into a control loop.
- We identify two output variables of the system: the performance quality of the neural model and its computational complexity. These variables are treated as dynamic quantities that can be controlled during the process.
- We define two input variables that dynamically affect the outputs and can be used as control signals to control the system.
- We present a simple example demonstrating the use of this methodology for the simultaneous training and pruning of a neural model, showcasing the feasibility and potential of our approach.

The purpose of this contribution is to introduce the core concept of the methodology while leaving a wide range of specific implementations open for further research.

II. Methodology

In this section, our aim is to establish a comprehensive framework that underpins the simultaneous optimization of neural network training and pruning as a dynamic control process. To achieve this, we must define the process that represents the neural network simultaneous training and pruning process as a multivariable dynamic system, including its inputs, outputs, and internal dynamics. This model serves as the foundation for integrating control theory into the training process. Additionally, it is necessary to identify and formalize the output variables that quantify both model performance and computational complexity, as well as the input variables that act as control signals influencing these outputs. By specifying these components, we enable the design of a control loop capable of maintaining an optimal balance between performance and efficiency. The methodology also requires an explanation of the control loop structure, including the choice of control algorithms and the feedback mechanisms employed. These definitions provide the theoretical and practical basis for the proposed approach and pave the way for its implementation and validation in the subsequent sections. For the purposes of this article, we will simplify the problem by considering only fully connected feedforward neural networks. Extending the methodology to general network structures will be the subject of future research.

A. Multivariable Dynamic System

The simultaneous training and pruning of a neural network can be formulated as a multivariable dynamic system suitable for control. This system comprises two input signals and two output variables. The input signals dynamically influence

the output, which represents the key metrics of the neural model. Specifically, the two output variables are defined as model performance, quantifying the accuracy or effectiveness of the network on a given task, and computational complexity, representing the computational resources required for inference and training. By treating these outputs as dynamic quantities, the system can be controlled to achieve a desired trade-off between high performance and low complexity. The input signals, on the other hand, correspond to parameters or actions that directly influence the pruning and training processes, such as the rate of pruning or the adjustment of learning hyperparameters. These inputs serve as control signals within the dynamic framework, enabling affection of the outputs throughout the control process. The diagram of the multivariable dynamic system within the control loop is depicted in Fig 1.

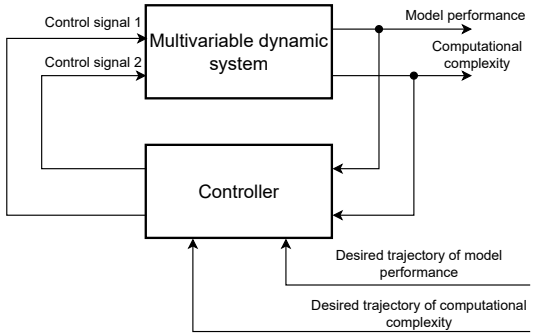

Fig. 1. Multivariable dynamic system representing the process of simultaneous training and pruning of a neural network model. The system is influenced by two input signals provided by a controller, which adjusts these inputs based on the desired trajectories of the system's output variables: model performance and computational complexity.

The neural network model considered in this study is composed of fully connected layers. Each layer processes a set of input features and generates a corresponding set of outputs through weighted connections and nonlinear activation functions. The network receives data samples at its input layer and produces predictions at its output layer, which are then compared to the target values to compute a loss. This loss guides the training process, where the weights are iteratively updated by backpropagation [12].

B. Output Variables

1) Model Performance: Model performance is a critical output variable that quantifies the quality of the neural network during training and evaluation. It is typically measured using a loss function that reflects the network's ability to perform the desired task. The choice of the loss function depends on the specific problem being addressed. For regression tasks, the mean squared error (MSE) is commonly used to capture the deviation between the predicted and the target values. In classification tasks, variants of cross-entropy loss are widely employed, as they effectively penalize incorrect predictions while accounting for class probabilities. Other domain-specific loss functions, such as hinge loss for support vector-based approaches or focal loss for imbalanced datasets, can also be used to define model performance. In this methodology, the selected metric serves not only as a guide for optimization during training, but also as a feedback signal for the control system to ensure that the model achieves and maintains the desired level of performance.

2) Computational Complexity: Computational complexity is another essential output variable, representing the resources required to deploy the neural network. This complexity is influenced by various factors, including the topology of the model, the number of neurons and layers, and the total number of trainable parameters. For instance, deeper networks with more neurons typically demand greater computational power and memory. Additionally, the hardware on which the model is executed plays a significant role in determining computational complexity. Beyond the structure of the network, metrics such as memory footprint, inference latency, and energy consumption can also be considered to quantify computational complexity. By monitoring and controlling this variable, the system can ensure that the network remains feasible for deployment in resource-constrained environments while meeting performance requirements.

C. Input Variables

1) Input Variable 1 - Training process parameters: The first input variable is designed to influence the performance of the model by dynamically adjusting the parameters associated with the training process. Key parameters include the number of training epochs, batch size, learning rate, and regularization terms. For example, increasing the number of epochs allows the model to learn more from the training data, potentially improving performance. Similarly, adjusting the batch size can influence the stability of gradient updates and the convergence behavior of the model. A smaller batch size may yield more frequent updates, enabling finer adjustments to the weights, while a larger batch size can improve computational efficiency by leveraging parallelism. Hyperparameters such as the learning rate and momentum terms are also critical, as they control the speed and direction of weight updates during optimization. Fine-tuning these parameters during the training process enables the system to maintain high model performance, adapting to changes in the data or network structure.

2) Input Variable 2 - Network topology adjustments: The second input variable focuses on computational complexity by dynamically modifying the topology of the neural network. This includes actions such as adding or removing layers, pruning neurons within layers, or adjusting connectivity between layers. For instance, reducing the number of neurons in a layer decreases the total number of trainable parameters and the associated computational load. Similarly, pruning entire layers can significantly reduce the memory footprint and inference time, but with potential impacts on model performance. Although this input variable is primarily aimed

at controlling computational complexity, its effects are not isolated. Modifications to the network topology inherently affect model performance, as the capacity of the network to represent complex functions is directly tied to its structure. Therefore, careful control and balancing of these adjustments are essential to achieve the desired trade-off between performance and efficiency. In other words, the interplay between the two input variables highlights the need for a coordinated control strategy to manage their combined impact on the output variables.

D. Control Strategy

With the process abstractly described as a multivariable dynamic system, complete with defined input and output variables, we can leverage control theory to design a control loop. To implement such a control system, it is essential to specify the desired trajectories for the output variables: model performance and computational complexity. These trajectories serve as benchmarks against which the system's current state is continuously evaluated.

The control loop in this context should incorporate advanced control algorithms, such as model predictive control (MPC) [13] or linear quadratic (LQ) regulation [14]. MPC, for instance, uses a model of the dynamic system to predict future behavior over a finite horizon and optimizes input variables to minimize deviations from the desired trajectories while satisfying constraints. This approach is particularly well-suited for our application, as it accommodates the interplay between the input and output variables and ensures stability and adaptability in dynamic conditions. LQ regulation, on the other hand, minimizes a cost function that balances the trade-offs between deviations from the desired outputs and the control effort required to achieve them. This method is computationally efficient and robust, making it a strong candidate for resource-constrained implementations.

Additionally, traditional multi-input multi-output PID controllers [15] or cascaded control structures [16] can also be considered. These approaches provide well-established and computationally simple solutions that may be advantageous in scenarios where model-based optimization techniques are infeasible or excessive in complexity.

The integration of such control strategies enables the system to dynamically adjust training parameters and architectural modifications in response to real-time feedback, achieving an optimal balance between performance and efficiency. This adaptive approach not only improves the overall effectiveness of the training and pruning process but also paves the way for practical deployment in diverse computational environments.

III. Demonstrative Example

The methodology presented in this work has been described in a highly abstract and concise manner, leaving many concepts undefined or underexplored. These gaps are intentional and will be addressed in future research, allowing for a more comprehensive development of the proposed framework. The purpose of this demonstrative example is to showcase a minimalist implementation of the methodology for designing a computationally efficient topology of a fully connected feedforward neural network for an approximation task. This example serves as a practical illustration of the framework, demonstrating its feasibility and potential to solve real-world problems.

A. Problem Statement

In this demonstrative example, we define the task of approximating a mathematical function using a fully connected neural network while simultaneously pruning the network topology according to the proposed methodology. The target function to be approximated is given by the following expression.

$$z(x_1, x_2) = \sin(\pi x_1) \cdot \cos(\pi x_2) + 0.5x_1^2 - 0.5x_2^2. \quad (1)$$

The function is defined in the domain $x_1, x_2 \in [-1, 1]$. A visualization of the function is shown in Fig. 2. The objective is to train a neural network to approximate this function with high accuracy while dynamically optimizing its topology to achieve computational efficiency. The optimization process follows the methodology described earlier, where model performance and computational complexity are treated as dynamic quantities controlled during training.

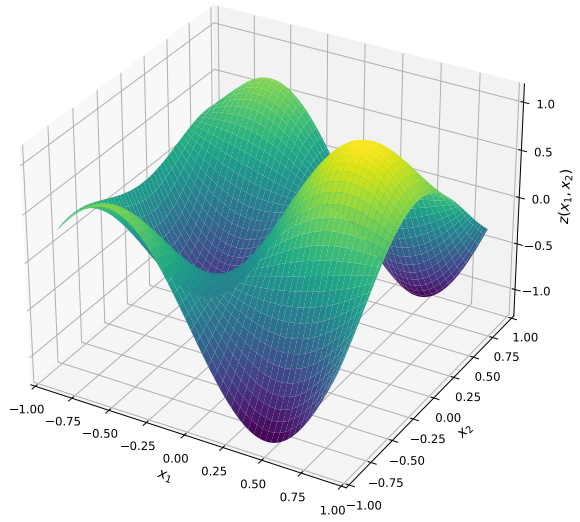

Fig. 2. Visualization of the function to be approximated.

The neural network will consist of fully connected layers, where the number of neurons can be dynamically adjusted. The input layer receives the two-dimensional input (x_1, x_2), and the output layer produces a single scalar value that approximates $z(x_1, x_2)$. The control strategy will regulate both the training parameters and the network structure, ensuring an optimal balance between accuracy and computational cost.

B. Initial Configuration of Neural Network Model

At the beginning of the simultaneous pruning of topology and training, it is necessary to select a sufficiently redundant

neural network configuration. In this study, we start with a fully connected feedforward neural network with two input nodes, a single output neuron, and three hidden layers, each containing ten neurons, i.e., the topology is referred to as [2-10-10-10-1]. The activation function used in all hidden neurons is the hyperbolic tangent (*tanh*), ensuring non-linearity and smooth gradient propagation.

The mathematical formulation of the neural network function can be expressed as follows.

$$\mathbf{h}^{(1)} = \tanh(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \tag{2}$$

$$\mathbf{h}^{(2)} = \tanh(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \tag{3}$$

$$\mathbf{h}^{(3)} = \tanh(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \tag{4}$$

$$\hat{z} = \mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}^{(4)} \tag{5}$$

where $\mathbf{x} \in \mathbb{R}^2$ represents the input vector, $\mathbf{h}^{(i)}$ are the activations of the hidden layers, $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ denote the weight matrices and bias vectors for each layer, and $\hat{z} \in \mathbb{R}$ is the final output of the network.

This initial configuration provides a sufficiently expressive model capable of learning the target function while allowing subsequent structural optimization to improve computational efficiency.

C. Output Variables

For the process of simultaneous pruning of the topology and training, it is essential to define two output variables in accordance with the proposed methodology: model performance and computational complexity. These variables serve as dynamic quantities that guide the regulation process, ensuring an optimal trade-off between the accuracy of the neural model and its computational efficiency throughout the optimization procedure.

1) Model Performance: For approximation tasks, model performance is typically evaluated using a loss function that quantifies the deviation between the predicted outputs and the expected values. A widely used metric in this context is the Mean Squared Error (MSE), which measures the average squared differences between predicted and actual values. Given its effectiveness in capturing the quality of function approximation, we adopt MSE as the performance measure in this study.

Mathematically, the model performance variable, denoted as y_1, is defined as:

$$y_1 = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{z}_i)^2, \tag{6}$$

where N is the number of samples in the dataset, t_i represents the expected output for the i-th sample, and $\hat{z}_i$ is the corresponding predicted output of the neural network.

This metric provides a smooth and differentiable cost function, making it well-suited for gradient-based optimization methods commonly used in neural network training. By minimizing y_1, the network adjusts its parameters to improve its approximation capability.

2) Computational Complexity: The computational complexity of a neural network topology can be evaluated using several approaches. These methods may consider factors such as the number of neurons, the number of layers, and the computational cost of individual operations within the topology. In general, a more complex topology with a larger number of neurons and layers requires greater computational resources, both in terms of training and inference.

For simplicity, in this study, we define computational complexity as the relative number of trainable parameters in the neural network. This metric provides a straightforward yet effective way to quantify the network's structural complexity and serves as a control variable within the proposed optimization framework.

Mathematically, the computational complexity variable, denoted as y_2, is defined as:

$$y_2 = \frac{\sum_{i=1}^{L-1} n_i \cdot n_{i+1}}{\sum_{i=1}^{L_0-1} n_i^0 \cdot n_{i+1}^0}, \tag{7}$$

where:

- L is the number of layers in the current topology,
- n_i represents the number of neurons in the i-th layer of the current topology,
- L_0 is the number of layers in the initial topology,
- n_i^0 represents the number of neurons in the i-th layer of the initial topology.

This formulation ensures that y_2 is normalized within the range $[0, 1]$, where a value of 1 corresponds to the initial topology with maximum complexity, and lower values indicate reduced complexity due to pruning or structural modifications.

By monitoring and controlling y_2, the system can dynamically adjust the network structure to balance computational efficiency with model performance, ensuring optimal trade-offs between accuracy and resource constraints.

D. Input Variables

In accordance with the proposed methodology, it is essential to define two input variables that serve as control actions within the control loop. These variables dynamically influence the structure and training process of the neural network, enabling real-time optimization of both model performance and computational complexity. Each input variable is designed to target a specific aspect of the optimization process.

1) Input Variable 1 - Training process parameters: The training process of a neural network is influenced by various hyperparameters, which govern the optimization procedure and affect the performance of the final model. Key hyperparameters include the learning rate, batch size, regularization coefficients, and the number of training epochs. Each of these parameters plays a crucial role in the convergence and generalization capabilities of the model.

For clarity and simplicity, in this study, we define the first input variable as the number of training epochs per control

step. This choice allows for direct regulation of the extent to which the model is trained before potential structural modifications. The input variable is constrained within a predefined range, where the minimum value corresponds to training for a single epoch, ensuring that the model updates its parameters at least once per step, while the maximum value is set to 50 epochs, allowing for more extensive training when needed. Mathematically, the first input variable u_1 is defined as:

$$u_1 \in [1, 50], \tag{8}$$

where u_1 represents the number of training epochs in a given control step. By dynamically adjusting u_1, the control system can influence model performance while maintaining computational efficiency, forming an essential part of the adaptive optimization process.

2) Input Variable 2 - Network topology adjustments: The topology of a neural network can be adjusted in various ways to optimize computational efficiency while maintaining sufficient representational capacity. Structural modifications may involve altering the number of layers, changing connectivity patterns, or adjusting the number of neurons in individual layers. In this study, we define the second input variable as the total number of neurons across all hidden layers, allowing the system to dynamically add or remove neurons during the optimization process.

The input variable u_2 is constrained within predefined limits, where the minimum configuration consists of two neurons per hidden layer, while the maximum corresponds to the number of neurons in the initial topology. Formally, this variable is defined as:

$$u_2 \in \left[2L, \sum_{i=1}^{L} n_i^0\right], \tag{9}$$

where L is the number of hidden layers, and n_i^0 denotes the initial number of neurons in the i-th hidden layer.

Neuron Addition and Removal Mechanism: In each control step, if u_2 is less than the total number of neurons in the hidden layers, a neuron is removed. In contrast, if u_2 is higher, a neuron is added. To maintain balanced layer-wise adjustments, probabilistic selection mechanisms determine which layer is affected.

a) Layer Selection for Neuron Addition or Removal: The probability of selecting a layer i for neuron modification is determined based on the current distribution of neurons between layers. We define the probability P_i of choosing layer i as:

$$P_i = \frac{n_i}{\sum_{j=1}^{L} n_j}, \tag{10}$$

where n_i is the current number of neurons in layer i. This ensures that modifications are distributed proportionally across layers, preventing excessive reduction in a single layer while preserving overall architectural balance.

b) Neuron Selection for Removal: Once a layer is selected for neuron removal, the specific neuron to be removed is chosen based on the similarity of its incoming weight connections. The probability P_j of removing the neuron j from the layer i is given by:

$$P_j = \frac{1}{Z} \sum_{k \neq j} \exp\left(-\|\mathbf{w}_j - \mathbf{w}_k\|_2\right), \tag{11}$$

where $\mathbf{w}_j$ and $\mathbf{w}_k$ represent the weight vectors of neurons j and k, respectively, and Z is a normalization factor ensuring $\sum P_j = 1$. This approach prioritizes the removal of neurons whose weight configurations are highly redundant with others, thereby minimizing loss of representational diversity.

c) Neuron Addition Mechanism: When a neuron is added, the target layer is selected using the same probability distribution P_i. The newly added neuron inherits random initial weights, following the standard initialization scheme used in the network.

The dynamic adjustment of the network topology directly influences both computational complexity and model performance. By selectively modifying the number of neurons, the system can balance resource constraints while maintaining adequate learning capacity.

E. Neural Network Training Algorithm

For training the neural network, we utilized the Keras deep learning framework, leveraging its built-in optimization functionalities. Specifically, we used Adam optimizer [17], a variant of stochastic gradient descent that adaptively adjusts learning rates for individual parameters. The learning rate was set to 0.001, ensuring a balance between convergence speed and stability. The loss function used for training was the Mean Squared Error, which aligns with the defined model performance metric.

F. Effect of Input Variables on Output Variables in Multivariable Dynamic System

To analyze the impact of the input variables on the output variables, we perform a simulation where step changes in the input variables are introduced in different time instances. By monitoring the resulting trajectories of model performance and computational complexity, we gain insight into the system's response dynamics. The resulting courses are shown in Fig. 3.

Fig. 3 illustrates the impact of the input variables u_1 and u_2 on the output variables y_1 and y_2. It is evident that when u_1 is set to a low value, the decrease in y_1 is gradual, indicating a slow improvement in the model performance. However, when u_1 is increased to a higher value, y_1 exhibits a steeper decline, demonstrating a more rapid convergence of the neural network training process.

Additionally, the results confirm that u_2 does not exclusively influence computational complexity y_2 but also affects the performance of the model y_1. Structural modifications to the network, such as adding or removing neurons, introduce changes in the network's representational capacity, which in turn impacts the accuracy of function approximation.

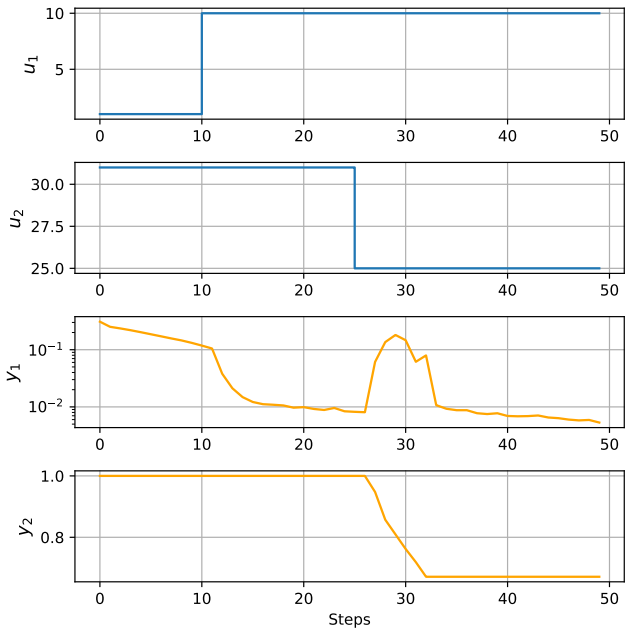

Fig. 3. Step responses of the multivariable dynamic system.

To maintain a consistent positive gain relationship between all input and output variables within the control framework, it is necessary to use y_1 with a negative sign in subsequent analysis. This transformation ensures that an increase in u_1 or u_2 leads to an intuitive and consistent system response.

G. Control Strategy

Various control strategies can be employed to regulate a multivariable dynamic system. However, for simplicity and clarity of this study, we implemented a two-dimensional proportional integral derivative (PID) controller to adjust the input variables u_1 and u_2 based on the deviation of the output variables y_1 and y_2 from their desired values.

The control law is defined as follows.

$$\mathbf{u}(t) = \mathbf{K}_P \mathbf{e}(t) + \mathbf{K}_I \int_0^t \mathbf{e}(\tau) d\tau + \mathbf{K}_D \frac{d\mathbf{e}(t)}{dt}, \qquad (12)$$

where:

- $\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$ is the vector of input variables,
- $\mathbf{e}(t) = \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix} = \begin{bmatrix} y_1^{\text{ref}} - y_1(t) \\ y_2^{\text{ref}} - y_2(t) \end{bmatrix}$ is the vector of control errors, defined as the difference between the reference values and the current system outputs,
- $\mathbf{K}_P, \mathbf{K}_I, \mathbf{K}_D \in \mathbb{R}^{2\times 2}$ are gain matrices for the proportional, integral, and derivative components, respectively.

The PID controller ensures that deviations of y_1 and y_2 from their desired trajectories are dynamically corrected. The proportional term provides immediate corrective action based on the current error, the integral term accumulates past errors to eliminate steady-state offsets, and the derivative term anticipates future errors by considering the rate of change.

Since the process studied in this work operates in discrete steps, the PID controller is discretized using finite difference approximations.

Adjusting the gains in $\mathbf{K}_P$, $\mathbf{K}_I$, and $\mathbf{K}_D$, the controller can be tuned to balance response speed, stability, and steady-state accuracy.

H. Reference values of output variables

During the simultaneous pruning of the topology and the training, both output variables should exhibit a decreasing trend. The final values of these variables are problem-dependent and influenced by the computational capabilities of the hardware on which the resulting neural network will be deployed. An optimal trade-off between model performance and computational complexity must be achieved to ensure both high accuracy and efficiency.

In this study, we define the reference trajectories for the output variables, composed of 50 samples, as follows.

- The model performance metric, y_1, follows a logarithmic decay from an initial value of 0.5 to a final value of 0.001 over 50 steps.
- The computational complexity metric, y_2, follows a linear decrease from an initial value of 1 to a final value of 0.2.

These reference trajectories ensure that the neural network gradually improves its accuracy while simultaneously reducing its structural complexity, allowing for efficient deployment on target hardware. The logarithmic decay of y_1 reflects the typical behavior of loss functions in neural network training, where rapid initial improvements are followed by slower refinements. The linear reduction of y_2 ensures a steady and controlled decrease in computational complexity without abrupt structural changes.

I. Simultaneous Training and Pruning Experiment

The matrices for the PID controller were configured based on preliminary tests, taking into account the expected relationships between the input and output variables. The remaining parameters of the experiment were set according to information reported earlier in the study.

$$\mathbf{K}_P = \begin{bmatrix} 80 & 40 \\ 0 & 80 \end{bmatrix},$$

$$\mathbf{K}_I = \begin{bmatrix} 2 & 0.5 \\ 0 & 2 \end{bmatrix},$$

$$\mathbf{K}_D = \begin{bmatrix} 0.1 & 0.05 \\ 0 & 0.1 \end{bmatrix}.$$

The final courses of all input and output variables are presented in Fig. 4.

The observed courses indicate that the process of simultaneous pruning of the neural network topology during its training yields favorable results in this case. The network topology was

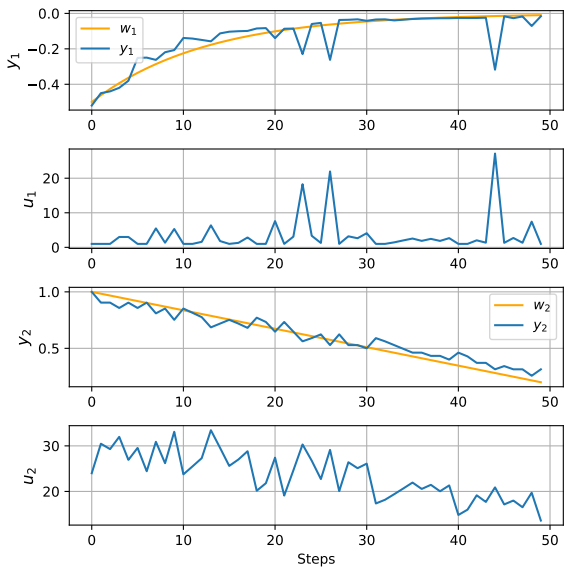

Fig. 4. Simultaneous training and pruning experiment.

streamlined from the initial configuration of [2-10-10-10-1] to [2-6-5-5-1], and a consistent trend of decreasing computational complexity was maintained. Throughout this process, training was conducted, gradually reducing the mean square error y_1 in approximately the desired manner. It is also evident that in the cases where changes in topology caused a sudden fluctuation in the mean square error, the control process eliminated this fluctuation through a faster increase in the input variable u_1.

IV. Conclusions and Future Directions

The aim of this study was to demonstrate the methodology of integrating pruning into the neural network training process, guided by the principles of multivariable system control.

The results of our experiment indicate the feasibility and potential of this methodology, showcasing its effectiveness in achieving a balance between model performance and computational complexity. However, the main purpose of this study was primarily to demonstrate the core concept of the methodology and to present a potential implementation of this methodology through a simple demonstration task. There was no ambition to analytically justify each individual step or to test the validity of the chosen hypotheses. This will be the subject of future research, which opens up several directions for further exploration. These directions include:

1) Advanced control techniques: Future work could explore more sophisticated control strategies, such as model predictive control or linear quadratic control, to further enhance the efficiency and robustness of the training and pruning process.
2) Computational complexity management: Investigating advanced techniques for managing computational complexity, such as hardware-aware pruning or energy-efficient training algorithms, could provide additional benefits to resource-constrained environments.
3) Training process optimization: Further research could focus on optimizing the training process itself, including the development of new hyperparameter tuning methods or adaptive learning rate schedules that dynamically adjust based on the network's performance and complexity.
4) Generalization to other neural network types: Extending the proposed methodology to other types of neural networks, such as recurrent neural networks, convolutional neural networks, or graph neural networks, would broaden its applicability and demonstrate its versatility.

Acknowledgment

The work was supported from ERDF "Multi-sector and Interdisciplinary Cooperation in Research and Development of Communication, Information and Detection Technologies for Control and Signalling Systems (CIDET)" (No. CZ.02.01.01/00/23_021/0008402).

References

[1] L. Hirsch and G. Katz, "Multi-objective pruning of dense neural networks using deep reinforcement learning," *Information Sciences*, vol. 610, p. 381 – 400, 2022.
[2] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11211 LNCS, p. 815 – 832, 2018.
[3] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2017. [Online]. Available: https://arxiv.org/abs/1608.08710
[4] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," vol. 2015-January, 2015, p. 1135 – 1143.
[5] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," 2016, p. 2082 – 2090.
[6] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Communications*, vol. 9, no. 1, 2018.
[7] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2019, p. 4780 – 4789.
[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.
[9] Z. Zhang, C. Cao, F. Zheng, T. Sun, and L. Zhao, "Gcpnet: Gradient-aware channel pruning network with bilateral coupled sampling strategy," *Expert Systems with Applications*, vol. 266, p. 126104, 2025.
[10] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," vol. 5, 2017, p. 3854 – 3863.
[11] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2019.
[12] S. Haykin, *Neural Networks and Learning Machines*, ser. Pearson International Edition. Pearson, 2009. [Online]. Available: https://books.google.cz/books?id=KCwWOAAACAAJ
[13] J. Köhler, M. A. Müller, and F. Allgöwer, "Analysis and design of model predictive control frameworks for dynamic operation—an overview," *Annual Reviews in Control*, vol. 57, 2024.
[14] R. Zhang, R. Lu, and Q. Jin, "Multivariable design of improved linear quadratic regulation control for mimo industrial processes," *ISA Transactions*, vol. 57, p. 276 – 285, 2015.
[15] K. Astrom and T. Hagglund, *Advanced PID Control*. ISA-The Instrumentation, Systems, and Automation Society, 2006.
[16] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. Wiley, 1996.
[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.