A logo with a torch and a cogwheel

Description automatically generated**Neural Networks**

**Prof. Mohammad Farrokhi**

**Homework (1) Spring 2024**

**System Modeling Using MLP**

**Iran University of Science and Technology**

**By Erfan Riazati**

**Abstract**

As we know, neural networks are utilized in a variety of problems, including classification, function approximation, and control tasks. We have become familiar with several of these structures through machine learning courses and neural network studies. For instance, the Perceptron is a binary classification algorithm capable of separating linear patterns. Similarly, algorithms like least squares can determine weights based on which linear separation is achieved. However, each of these algorithms has its limitations, such as dependency on linear separability or computational complexity.

To address these issues, multilayer perceptron (MLP) were introduced. By employing the backpropagation algorithm, this type of neural network becomes a powerful and practical learning structure capable of separating non-linear patterns. The use of this algorithm is justified due to its relatively fast and straightforward computations.

In this report, we aim to estimate a non-linear system using the MLP algorithm. This involves using sample inputs to approximate the output of a non-linear system while learning the parameters of the neural network. Through this process, we attempt to utilize and optimize the backpropagation algorithm for efficient learning and validation.

**Keywords:** Multilayer perceptron, backpropagation algorithm, supervised learning, hidden layer, feature space, validation.

**Introduction**

The purpose of this report is to estimate a non-linear system using a multilayer perceptron (MLP). The project involves determining the architecture of the network, including the number of hidden layers, the number of neurons in each layer, and other parameters such as learning coefficients. Initially, we define the problem and create a dataset. Subsequently, we design the network and its architecture, implementing it step-by-step in MATLAB software. Finally, we test the network’s performance and validate its results.

This report aims to bridge theoretical concepts and practical implementation. The methodology described herein allows for an effective understanding of MLP and their application to non-linear system modeling.

**Problem Description**

The non-linear system in question can be represented by the following equation:

y(k)=αy(k−1)y(k−2)(y(k−2)+β)1+y2(k−1)y2(k−2)+u(k−1)y(k) = \alpha \frac{y(k-1)y(k-2)(y(k-2) + \beta)}{1 + y^2(k-1)y^2(k-2)} + u(k-1)

where α\alpha and β\beta are constants with values α=1.2\alpha = 1.2 and β=1.1\beta = 1.1, and u(k)u(k) represents a control signal defined as:

u(k)=0.5sin⁡(πk11)+0.4cos⁡(πk6.5)+0.2sin⁡(πk45)for 1≤k<400.u(k) = 0.5 \sin \left(\frac{\pi k}{11}\right) + 0.4 \cos \left(\frac{\pi k}{6.5}\right) + 0.2 \sin \left(\frac{\pi k}{45}\right) \quad \text{for } 1 \leq k < 400.

The task is to model the system’s output y(k)y(k) based on the given inputs using an MLP. The dataset for this purpose is generated by simulating the system according to these equations under specified constraints. The control signal u(k)u(k) and past values of y(k)y(k) serve as inputs to the neural network.

**Dataset Construction**

In this stage, MATLAB software is used to compute the desired samples while adhering to the specified constraints. The parameter β=1.1\beta = 1.1 is considered fixed for all calculations. All MATLAB programs utilized in this process are included in the appendix.

The output of the original system, denoted as y(k)y(k), is generated as described earlier. The control signal u(k)u(k) and its relationship with the system's response are shown in the following figures. The dataset consists of 400 samples to ensure adequate training data. Each sample is generated by observing the system’s behavior over time. The dataset is then divided into training, validation, and testing subsets for further processing.

To ensure effective learning, preprocessing is performed on the output data to map it within the range of the activation function. A hyperbolic tangent activation function is employed, defined as:

ϕ(v)=atanh⁡(bv)\phi(v) = a \tanh(bv)

where a=1.7159a = 1.7159 and b=2/3b = 2/3. This function ensures that the system's output remains within the interval [−1,1][-1, 1], preventing saturation of the activation function and improving the efficiency of the learning process. The results of preprocessing the desired output data are depicted in the following figures.

**Preprocessing of Desired Outputs**

In this stage, the desired output data of the system is adjusted using a hyperbolic tangent activation function, ensuring that the outputs fall within the operational range of the activation function. The chosen function, expressed as:

ϕ(v)=atanh⁡(bv)\phi(v) = a \tanh(bv)

with constants a=1.7159a = 1.7159 and b=2/3b = 2/3, scales and centers the output data effectively. This normalization step ensures that the outputs avoid the saturation zones of the activation function, thus accelerating the learning process and improving the convergence of the neural network.

Additionally, to maintain consistency and uniformity, the output data is bounded to the interval [−1,1]. This mapping is crucial to minimize errors during training and ensure compatibility with the network’s architecture. The normalized outputs after preprocessing are presented graphically to illustrate their alignment with the activation function’s range.

**Input Data Normalization**

In addition to preprocessing the outputs, the input data must also be normalized. For effective training and accurate model prediction, the mean of the input data across the entire dataset is subtracted, ensuring that the data is centered around zero. This step minimizes potential drifts during learning and ensures the stability of the weight updates.

After centering the data, it is further scaled to a specific range by applying a factor that adjusts the input values to fall within suitable bounds. The range is chosen to align with the operational constraints of the neural network and prevent any inconsistencies caused by extreme input values.

This normalization process not only enhances the convergence rate of the learning algorithm but also prevents overfitting by maintaining balanced gradients throughout the training. The resulting normalized input data is graphically depicted, showcasing its suitability for use within the MLP framework.

**Multilayer Perceptron**

The architecture of the multilayer perceptron (MLP) used in this study consists of an input layer, one or two hidden layers, and an output layer. The number of input neurons corresponds to the features of the dataset, excluding bias terms, while the output neurons represent the target values. The number of neurons in the hidden layers is flexible and can be adjusted based on the complexity of the problem.

In this implementation, we begin with one hidden layer containing ten neurons and experiment with increasing the complexity by adding a second hidden layer when necessary. Each hidden neuron employs the hyperbolic tangent activation function to introduce non-linearity, enabling the network to model complex patterns.

The MLP training process relies on the backpropagation algorithm to update the weights. This involves two primary phases:

1. **Feedforward Phase**: During this phase, the input data is propagated through the network layer by layer to compute the output predictions. The error between the predicted and actual values is calculated and used in the next phase.
2. **Backpropagation Phase**: In this phase, the error signal is propagated backward through the network to update the weights of each connection based on their contribution to the error. This weight update is governed by the gradient of the loss function and a learning rate parameter.

The weight initialization, network configuration, and training methodology are elaborated in the subsequent sections, highlighting their roles in ensuring effective learning and minimizing computational inefficiencies.

**Initialization of Weights**

To ensure effective training and avoid poor convergence, the weights and biases of the neural network are initialized randomly using a uniform distribution centered around zero. This initialization ensures that no neuron dominates the learning process from the start.

The weights are initialized based on the number of connections associated with each neuron. Specifically, the standard deviation of the weight distribution for a neuron is inversely proportional to the square root of the number of its input connections, as given by:

σw0=m−12\sigma\_{w\_0} = m^{-\frac{1}{2}}

where mm represents the number of incoming connections for a given neuron.

In this study, the initial weights are set to small random values to prevent the activation function from saturating in the early stages of training. Additionally, the biases are initialized to zero to simplify the initial computation of outputs.

Sample weight matrices for the network’s connections are presented in the following sections, providing insights into the distribution of initialized weights.