Short Paper
A Short Subtitle

Mohamed Farid Khalil[a,1,*], Mohammed Twheed Khater[a,2], Seif Ibrahim Hassan[a,3]

[a] *Alexandria University, Mechanical Engineering, Alexandria, Egypt,*

Abstract

This paper presents a genetic algorithm (GA) methodology to optimize neural network hyperparameters in the context of pump impeller trimming. Impeller trimming, a process involving modifications to pump impeller geometry, traditionally requires expert knowledge and empirical methods to achieve the desired performance. The use of neural networks (NN) provides an automated approach to improve the impeller trimming process based on input data and performance outcomes. The proposed method uses a (GA) to identify the optimal NN hyperparameters, such as hidden layer size, training function, activation function, and maximum epochs, aiming to minimize the mean squared error (MSE) between the network's predictions and the actual target outcomes. This paper discusses the implementation details of the optimization process and explains the key components and their significance.

Keywords: Pump impeller trimming, Neural networks, Hyperparameter optimization, Genetic algorithms, Mean squared error

*Corresponding author

Email addresses: mfaridkhalil@yahoo.com (Mohamed Farid Khalil), mohammedtwheed@gmail.com (Mohammed Twheed Khater)

[1] Professor, Mechanical Engineering Department, Faculty of Engineering, Alexandria University, Alexandria, Egypt.

[2] Undergraduate, Mechanical Engineering Department, Faculty of Engineering, Alexandria University, Alexandria, Egypt.

[3] Undergraduate, Mechanical Engineering Department, Faculty of Engineering, Alexandria University, Alexandria, Egypt.

Preprint submitted to Dr. Farid *2024-04-15*

Nomenclature

NN Neural Network

GA Genetic Algorithm

MSE Mean Square Erorr

NNs Neural Networks

CNNs Convolution Neural Networks

RNNs Recurrent Neural Networks

1. introduction

Pump impeller trimming is a critical procedure in optimizing pump performance for specific applications. It involves modifying the impeller's geometry to achieve desired hydraulic characteristics such as head, flow rate, and efficiency. Traditionally, the process has been dependent on empirical methods and engineering expertise. However, the introduction of artificial neural networks (NN) offers a data-driven approach to automate and enhance impeller trimming.

NN excel at modeling complex relationships between input data and desired outputs.

By training an NN on a dataset of impeller designs and performance outcomes, the network can learn to predict new impeller performance based on their geometries.

In the case of pump impeller trimming, the input data in our case is the desired operating point of the pump (flow rate Q, head H), while the target data or output from the network is (the impeller outer diameter D , pump efficiency η).

Achieving optimal NN performance requires selecting appropriate hyperparameters, which influence network architecture and the learning process. Key NN hyperparameters include : 1. the size of the hidden layer. 2. the training function for weight updates. 3. the activation function introducing non-linearity. 4. the maximum number of training epochs.

2. Methodology

This paper outlines a genetic algorithm (GA) methodology for optimizing neural network hyperparameters in pump impeller trimming. (GA) is suitable for searching for optimal solutions in complex, high-dimensional spaces. The (GA) approach used in this study involves a population of candidate hyperparameter sets. Each set is evaluated by training an NN with those hyperparameters and measuring the resulting (MSE) on a validation dataset. The GA iteratively selects promising hyperparameter sets based on (MSE) values of the (NN), performs crossover and mutation to create new candidates, and continues until a stopping criterion (such as maximum generations or elapsed time) is met.

3. Neural Networks

FROM:[https://www.historyofinformation.com/detail.php?entryid=782]

The concept of artificial neural networks (NNs) draws inspiration from the biological structure and function of the human brain. Early work in the 1940s by Warren McCulloch and Walter Pitts established a foundation for artificial neurons as interconnected nodes mimicking biological neurons. In the 1950s, Frank Rosenblatt introduced the perceptron, a simple NN architecture that laid the groundwork for further development. However, limitations of the perceptron led to a period of decline in NN research during the 1970s and 1980s.

The resurgence of NNs can be attributed to advancements in several areas. The development of the back-propagation algorithm in the 1980s provided a more efficient method for training complex NN architectures. Additionally, increased computational power and the introduction of new NN architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), expanded the capabilities of NNs.

Mathematically, NNs operate by transforming input data through a series of interconnected layers. Each layer consists of artificial neurons that perform weighted sums of their inputs and apply an activation function to introduce non-linearity. This allows NNs to learn complex relationships between input and output data, making them powerful tools for tasks like pump impeller trimming performance curve prediction.

FROM:[https://playground.tensorflow.org/]

3.1. Simple numerical example on how neural networks work

In this example we will use a very simple shallow neural network consists of Input layer consists of 2 neurons , Hidden layer consists of 3 neurons and Output layer consists of 2 neurons.

TODO

3.2. Our neural network

our neural network is trained on $(Q\ m^3/h, H\ m)$ to predict $(D\ mm, \eta)$.

the neural network consists of 2-neurons input layer , n-neurons hidden layer and 2-neurons output layer using matlab ga toolbox we will search for optimum hyperparameters (n (number of neurons in hidden layer),training algorithm,activation function and number of epochs) where n varies from 5 to 200 , training algorithms $'trainlm','trainbr','trainrp','traincgb','traincgf','traincgp','traingdx','trainoss'$, number of epochs will vary from 50 to 200 and activations functions will be $'tansig','logsig'$ where Hyperbolic Tangent (tansig) is $f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and Logistic Sigmoid (logsig) is $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$.

we use matlab `mapminmax` to normalize our training dataset to interval $[-1, 1]$ and after the training we recover the scale back using `reverse` option in `mapminmax`. This is because neural network training algorithms typically rely on gradient descent to optimize weights. By normalizing the input data to a specific range (often -1 to 1 or 0 to 1), the gradients calculated during *backpropagation* have a more consistent magnitude. This helps the training process converge faster and avoid getting stuck in local minima.

we set a random seed for reproducibility where Setting a random seed ensures that the neural network training process is reproducible. This allows for consistent results across different runs and helps in comparing models fairly.

4. results

5. Conclusion

This paper presents a GA-based approach for optimizing neural network hyperparameters in the context of pump impeller trimming. The methodology enables a data-driven, automated optimization process, providing potential improvements in efficiency and performance in pump design and operation. Future work may explore refining the genetic algorithm for better convergence or testing the approach with more complex NN architectures.

6. code documentation

Optimizing Neural Network Hyperparameters for Pump Impeller Trimming

This code implements a function called `optimizeNNForTrimmingPumpImpeller` that uses a Genetic Algorithm (GA) to optimize the hyperparameters of a neural network for pump impeller trimming.

1. Starting Timer and User Notification:

```
% Start timer to measure the duration of the optimization process.
tic;
disp("Optimization in progress. This process may take up to 30 seconds...");
```

- The `tic` function starts a timer to measure how long the optimization process takes.
- The `disp` function displays a message to the user indicating that the optimization is underway and might take up to 30 seconds.

2. Defining Training and Activation Function Options:

```
% Define the available options for training functions and activation functions.
trainingFunctionOptions = {'trainlm', 'trainbr', ...
'trainrp', 'traincgb', 'traincgf',...
 'traincgp', 'traingdx', 'trainoss'};
activationFunctionOptions = {'tansig', 'logsig'};
```

- This section defines two cell arrays.
  - `trainingFunctionOptions`: This array stores names of various training functions available in MATLAB's `fitnet` object. Training functions determine how the neural network updates its weights during training, impacting its learning behavior.
  - `activationFunctionOptions`: This array stores names of activation functions that can be applied in the neural network's hidden layer. Activation functions introduce non-linearity, allowing the network to model complex relationships between inputs and outputs.

3. Defining Bounds and Options for Genetic Algorithm (GA):

```
% Define bounds for the genetic algorithm optimization.
lowerBounds = [5, 50, 1, 1];
upperBounds = [200, 200, 8, 2];

% Define options for the genetic algorithm.
gaOptions = optimoptions('ga', 'MaxTime', 20);
```

- **Bounds for GA Search (`lowerBounds` and `upperBounds`)**
  - These vectors define the minimum and maximum allowable values for each hyperparameter during the GA optimization. For instance, `lowerBounds = [5, 50, 1, 1]` specifies a minimum hidden layer size of 5 neurons, a minimum of 50 training epochs, and minimum indices of 1 for the

training function and activation function (since these indices correspond to entries within the options arrays we defined earlier). The upper bounds define the maximum allowed values for each hyperparameter.

- **GA Options (`gaOptions`)**
  - This line uses `optimoptions('ga')` to create a structure containing options for the GA. The GA is a search algorithm inspired by biological evolution. It iteratively tweaks candidate solutions (in our case, sets of hyperparameters) and selects promising ones based on their performance (MSE in this case) to create new generations. You can modify these options to control aspects like population size (number of candidate hyperparameter sets considered simultaneously) and termination criteria (when to stop the search). Here, `'MaxTime', 20` sets a maximum allowed time of 20 seconds for the optimization process per iteration (not total time). The algorithm stops after running for MaxTime seconds, as measured by tic and toc. This limit is enforced after each iteration, so ga can exceed the limit when an iteration takes substantial time.

4. Global Variable to Store Best Network:

```
% Global variable to store the best trained neural network found during optimization.
global bestTrainedNet;
bestTrainedNet = [];
```

- A global variable named `bestTrainedNet` is declared. This variable will be used to store the neural network model that achieves the lowest MSE during the optimization process.

5. Nested Function: evaluateHyperparameters

```
% Nested function to evaluate hyperparameters using the neural network.
    function mse = evaluateHyperparameters(hyperParams, x, t, randomSeed)
        rng(randomSeed); % Set random seed for reproducibility.

        % Extract hyperparameters.
        hiddenLayerSize = round(hyperParams(1)); %Hidden Layer Size
        maxEpochs = round(hyperParams(2));       %Max Epochs
        trainingFunctionIdx = round(hyperParams(3)); %Training Function
        activationFunctionIdx = round(hyperParams(4));%Activation
        %Function or transfere function

        % Define the neural network.
```

```
        net = fitnet(hiddenLayerSize, trainingFunctionOptions{trainingFunctionIdx});
        net.trainParam.showWindow = false; % Suppress training GUI for efficiency.
        net.trainParam.epochs = maxEpochs;
        net.layers{1}.transferFcn = activationFunctionOptions{activationFunctionIdx};

        % Define data split for training, validation, and testing.
        net.divideParam.trainRatio = 0.7;
        net.divideParam.valRatio = 0.15;
        net.divideParam.testRatio = 0.15;

        % Train the neural network.
        [trainedNet, ~] = train(net, x, t);

        % Evaluate the model performance using mean squared error (MSE).
        predictions = trainedNet(x);
        mse = perform(trainedNet, t, predictions);

        % Check if the current MSE is the best MSE so far
        %and update the global variable if necessary.
        if isempty(bestTrainedNet) || mse < perform(bestTrainedNet,...
         t, bestTrainedNet(x))
            bestTrainedNet = trainedNet;
        end
    end
```

The function `evaluateHyperparameters` evaluates a set of hyperparameters for a neural network model. It uses the hyperparameters to define, train, and evaluate a neural network, returning the model's mean squared error (MSE) as a measure of performance.

6.0.1. Function Overview:

1. **Inputs:**
    - `hyperParams`: A vector of hyperparameters including:
        - `hiddenLayerSize`: The size of the hidden layer in the neural network.
        - `maxEpochs`: The maximum number of epochs (training iterations).
        - `trainingFunctionIdx`: The index of the training function to use.
        - `activationFunctionIdx`: The index of the activation function to use.

    - `x`: The input data (features) for the neural network.
    - `t`: The target data (labels) for the neural network.
    - `randomSeed`: The random seed for reproducibility.
2. **Set Random Seed:**
    - The function starts by setting the random seed using `rng(randomSeed)` for reproducibility. This ensures that the neural network training process is consistent and repeatable.
3. **Extract and Apply Hyperparameters:**
    - The function extracts hyperparameters from the input vector `hyperParams`.
    - It uses these hyperparameters to define the neural network architecture, training function, and activation function.
4. **Data Splitting:**
    - The function defines how to split the data into training, validation, and testing sets (70% for training, 15% for validation, and 15% for testing).
5. **Train the Neural Network:**
    - The function trains the neural network using the specified hyperparameters and input data.
6. **Evaluate Performance:**
    - The function evaluates the trained network's performance using mean squared error (MSE), a measure of prediction accuracy.
7. **Track the Best Trained Network:**
    - The function compares the current MSE to the best MSE seen so far. If the current MSE is better, the trained network is stored as the best-trained network.

6.0.2. Importance of Random Seed:

- **Reproducibility:** Setting a random seed ensures that the neural network training process is reproducible. This allows for consistent results across different runs and helps in comparing models fairly.
- **Comparison:** When testing different hyperparameters or models, using the same random seed allows for a direct comparison of their performance.

6.0.3. Concepts Underlying Epochs:

- **Epochs:** An epoch is a complete pass through the entire training dataset. During each epoch, the neural network updates its weights based on the training data.
- **Why Search for Optimal Epochs:** The number of epochs affects how much the network learns from the data:
    - **Too Few Epochs:** The network may not learn enough and can underfit, performing poorly on new data.

  - **Too Many Epochs:** The network may learn too much and can overfit, performing well on the training data but poorly on new data.
- **Optimum Number of Epochs:** An optimal number of epochs strikes a balance between underfitting and overfitting, ensuring the network generalizes well to new data.

References