# 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 (NNs) 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

**Nomenclature**

| | |
|---|---|
| NN | Neural Network |
| GA | Genetic Algorithm |

*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.

## 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 (NNs) offers a data-driven approach to automate and enhance impeller trimming.

NNs excel at modeling complex relationships between input data and desired outputs. In the case of pump impeller trimming, the input data (x) can include geometric parameters of the impeller, while the target data (t) can consist of pump performance metrics. 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.

Achieving optimal NN performance requires selecting appropriate hyperparameters, which influence network architecture and the learning process. Key NN hyperparameters include the size of the hidden layer, the training function for weight updates, the activation function introducing non-linearity, and 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. GAs are 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, performs crossover and mutation to create new candidates, and continues until a stopping criterion (such as maximum generations or elapsed time) is met.

### *2.1. Define Options*

Define the set of available training functions (e.g., Levenberg-Marquardt, Bayesian Regularization) and activation functions (e.g., tansigmoid, logarithmic sigmoid).

### *2.2. Nested Evaluation Function*

Implement a nested function `evaluateHyperparameters(hyperParams, x, t, randomSeed)` to evaluate a candidate hyperparameter set (`hyperParams`), input data (`x`), and target data (`t`). This function: - Extracts individual hyperparameters (hidden layer size, maximum epochs, training and activation function indices). - Defines an NN architecture with the given hyperparameters. - Splits data into training, validation,

and testing sets. - Trains the NN using the training data and specified training function. - Evaluates the NN on validation data and calculates MSE. - Returns the calculated MSE.

### *2.3. Random Seed and Bounds*

set a random seed for reproducibility and define bounds for each hyperparameter based on practical considerations and prior knowledge.

#### *2.3.1. 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.

### *2.4. GA Options*

Configure GA options using `optimoptions('ga')`, specifying parameters such as maximum allowed time.

### *2.5. Perform Optimization*

Execute the GA using `ga` from MATLAB's Optimization Toolbox. This function uses the `evaluateHyperparameters` function as the objective function to minimize, along with the number of hyperparameters, bounds, and GA options.

### *2.6. Extract and Round*

Obtain the final optimized hyperparameter set and round them (if necessary) for practical NN implementation.

### *2.7. Report Results*

Report the optimized hyperparameters, final MSE, random seed used, and total elapsed time of the optimization process.

## 3. 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.

## 4. 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.

#### *4.0.1. 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.

#### *4.0.2. 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.

#### *4.0.3. 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.

## 5. test citation

as dr farid said[1] this is no enough

*References*


[1] P. A. M. Dirac, The Lorentz transformation and absolute time, Physica 19 (1-–12) (1953) 888–896. doi:10.1016/S0031-8914(53)80099-6.

[2] D. E. Knuth, The TEX Book, Addison-Wesley Professional, 1986.

[3] F. Mittelbach, M. Gossens, J. Braams, D. Carlisle, C. Rowley, The LATEX Companion, 2nd Edition, Addison-Wesley Professional, 2004.

[4] L. Lamport, LATEX: a Document Preparation System, 2nd Edition, Addison Wesley, Massachusetts, 1994.

[5] D. E. Knuth, Literate programming, The Computer Journal 27 (2) (1984) 97–111.

[6] M. Lesk, B. Kernighan, Computer typesetting of technical journals on UNIX, in: Proceedings of American Federation of Information Processing Societies: 1977 National Computer Conference, Dallas, Texas, 1977, pp. 879–888.

[7] R. P. Feynman, F. L. Vernon Jr., The theory of a general quantum system interacting with a linear dissipative system, Annals of Physics 24 (1963) 118–173. doi:10.1016/0003-4916(63)90068-X.