



Optimization of Artificial Neural Network Transfer Function for Hydrological Modelling: A Review

¹ORJI, FN; ¹AHANEKU, IE; ¹NDUKWU, MC; ¹UGWU, E; *²AWU, JI; ³JOSEPH, IU; ⁴HELEN, I

¹Michael Okpara University of Agriculture Umudike, Abia State, Nigeria
²National Centre for Agricultural Mechanization, Ilorin, Kwara State, Nigeria
³Nnamdi Azikiwe University Awka, Anambra State, Nigeria
⁴Federal Polytechnic, Oko, Anambra State, Nigeria

*Corresponding Author Email: ecoagricultural@gmail.com

ABSTRACT: Hydrological modeling is crucial for understanding and predicting water-related processes. Artificial Neural Networks (ANN) have emerged as powerful tools for this purpose, utilizing the structure and functions of the biological brain to model complex patterns and forecast hydrological issues. Hence, this study reviews the optimization of artificial neural networks for hydrological modeling. Data obtained reveals that the choice of transfer function significantly impacts the performance of hydrological models, and optimizing it can improve accuracy, precision, and reliability. More so, an optimized transfer function provides interpretability, aligning with the physical understanding of the hydrological system and making the model outputs more meaningful. By optimizing artificial neural network transfer functions and employing other optimization strategies, hydrological models can better simulate and predict water-related processes. This advancement can lead to more effective water resource management and decision-making.

DOI: <https://dx.doi.org/10.4314/jasem.v27i8.2>

Open Access Policy: All articles published by **JASEM** are open-access articles under **PKP** powered by **AJOL**. The articles are made immediately available worldwide after publication. No special permission is required to reuse all or part of the article published by **JASEM**, including plates, figures and tables.

Copyright Policy: © 2023 by the Authors. This article is an open-access article distributed under the terms and conditions of the **Creative Commons Attribution 4.0 International (CC-BY- 4.0)** license. Any part of the article may be reused without permission provided that the original article is cited.

Cite this paper as: ORJI, FN; AHANEKU, IE; NDUKWU, MC; UGWU, E; AWU, JI; JOSEPH, IU; HELEN, I. (2023). Optimization of Artificial Neural Network Transfer Function for Hydrological Modelling: A Review. *J. Appl. Sci. Environ. Manage.* 27 (8) 1617-1626

Dates: Received: 12 June 2023; Revised: 21 July 2023; Accepted: 04 August 2023 Published: 30 August 2023

Keywords: artificial neural network; transfer function; optimization; modelling

Hydrological modeling plays a crucial role in understanding and predicting water-related processes. Artificial Neural Networks (ANN) is a data modelling tool that is based on the structure and functions of biological brain to model complicated patterns and forecast issues. In recent years, the field of system investigation has experienced a resurgence with the adoption of artificial intelligence techniques such as Artificial Neural Networks (ANN) in hydrological modeling (Minns and Hall, 1996). Artificial neural networks possess distinct features that set them apart from other models. They exhibit flexibility in handling data and can effectively solve problems where obtaining primary data is nearly impossible (Ouenes, 2000; Zio, 1997). They can also handle the added

complexity of groundwater chemistry (Gumrah *et al.*, 2000) and address highly nonlinear and spatially/temporally variant processes (Islam and Kothari, 2000). Furthermore, artificial neural networks can handle incomplete, noisy, and ambiguous data (Maier and Dandy, 1998). They are often more cost-effective and simpler to implement in terms of data requirements and model structure compared to physically based models (Campolo *et al.*, 1999). ANNs are well-suited for dynamic problems, efficiently store information within the trained model, and excel at time series pattern matching (Thirumalaiah and Deo, 1998). Hence, artificial neural networks (ANNs) have emerged as powerful tools for hydrological modeling due to their ability to

*Corresponding Author Email: ecoagricultural@gmail.com

capture complex nonlinear relationships. However, the performance of ANNs depends on their optimization, which involves selecting appropriate network architectures and optimizing model parameters. The objective of this study is to review the artificial neural networks optimization for hydrological modelling. Artificial neural network transfer function play a vital role on the model network output. The transfer function is a fundamental component of an ANN that determines the output of each neuron based on the weighted sum of inputs. Choosing an appropriate transfer function is critical for achieving accurate and reliable hydrological modeling results. Therefore, the optimization of the transfer function for hydrological modeling can be justified for several reasons. Hydrological systems often exhibit non-linear behaviors due to the complex interactions between various hydrological processes such as precipitation, evapotranspiration, infiltration, and runoff (Sivakumar *et al.*, 2007). ANNs with appropriate transfer functions can effectively capture and model these non-linear relationships, enabling more accurate simulations and predictions (ASCE, 2000). Likewise, an optimized transfer function enhances the generalization capability of the hydrological model. Generalization refers to the ability of the model to accurately predict outcomes for unseen data. By selecting an appropriate transfer function, overfitting (when a model becomes too specialized for the training data) can be minimized, and the model can better capture the underlying patterns and dynamics of the hydrological system (Zhu *et al.*, 2000). However, ANNs offer a wide range of transfer functions to choose from, including sigmoid, hyperbolic tangent, radial basis function, and rectified linear unit (ReLU) (Haykin, 1999). Each transfer function has its strengths and limitations, and the optimization process helps identify the most suitable transfer function for a specific hydrological modeling task (Moghaddam and Sorooshian, 2001). This flexibility allows for a better representation of the underlying hydrological processes (Duan *et al.*, 1998). Moghaddam and Sorooshian (2001), in their study on a review of artificial neural networks in hydrology, reveal that the choice of the transfer function can significantly impact the performance of the hydrological model. By optimizing the transfer function, the model's accuracy, precision, and reliability can be improved. This optimization process involves fine-tuning the parameters of the transfer function or exploring different transfer functions to find the optimal configuration that minimizes errors and maximizes model performance. Some transfer functions have inherent interpretability, which can provide insights into the hydrological processes being modelled (Moghaddam and Sorooshian (2001). For example, sigmoid functions can represent the

saturation and attenuation behaviour of certain hydrological phenomena. By optimizing the transfer function, it is possible to select a function that aligns with the physical understanding of the hydrological system, making the model outputs more interpretable. Hence, the optimization of the artificial neural network transfer function for hydrological modelling holds significant potential to advance the accuracy of hydrological models, enhance our understanding of hydrological processes, capture non-linear behaviours, improve model flexibility and generalization, enhance model performance, and provide interpretability. By selecting the most suitable transfer function, hydrological models can better simulate and predict various water-related processes, leading to more effective water resource management and decision-making.

ANNs Optimization Strategies: When optimizing artificial neural networks (ANNs) for hydrologic modelling, there are several approaches that are considered. ANNs optimization plays a crucial role in capturing the non-linear relationships between input variables and output variables in ANNs. Here are some optimization strategies that can be employed for ANNs hydrological modelling.

Network Architecture Optimization: Network architecture optimization is a critical process in the field of deep learning, aimed at designing and refining the structure of neural networks to improve their performance and efficiency. The goal is to create models that can effectively learn and extract meaningful information from complex data. The study of Goodfellow *et al.* (2016), reveals that choosing an appropriate model architecture is the first step to having good prediction. This involves selecting from a wide range of neural network types, such as convolutional neural networks (CNNs) for image data, recurrent neural networks (RNNs) for sequential data, or transformer models for natural language processing tasks. Once the type of model is determined, the next step is to design the layers of the network. This includes deciding the number of layers, the type of activation functions, and the connections between layers (e.g., fully connected, convolutional, etc.). Careful consideration is given to strike a balance between model complexity and expressiveness. The choice of network architecture significantly impacts the performance of hydrological models Moghaddam and Sorooshian (2017). Several studies have focused on optimizing the architecture of ANNs for hydrological modelling, including the number of hidden layers, the number of neurons per layer, and the activation functions used. Example of an artificial neural network architecture is shown in Figure 1

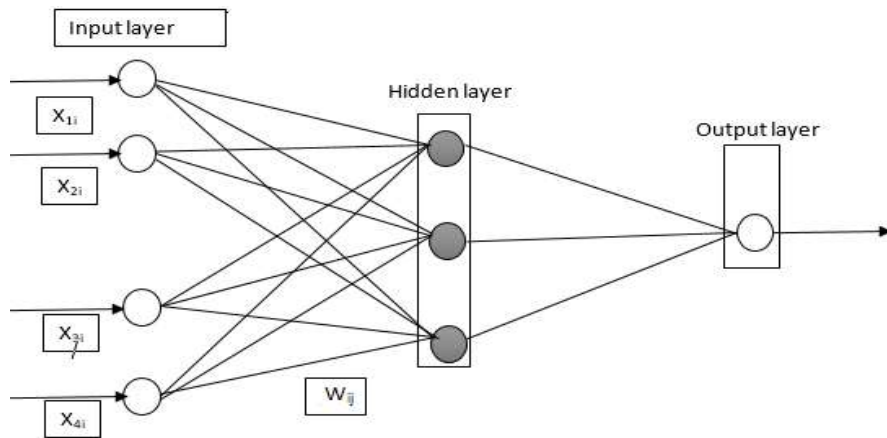


Fig 1: Schematic diagram of Artificial Neural Network Architecture

Transfer Function Optimization: Transfer function selection is another crucial aspect optimizing designing neural networks. Transfer function introduces non-linearity to the network, enabling it to learn complex relationships between inputs and outputs. The choice of Transfer function can have a significant impact on the model's learning capacity, convergence speed, and ability to handle different types of data.

The commonly used Transfer function are:

Sigmoid Function: The sigmoid transfer function, also known as the logistic function, is a mathematical function commonly used in machine learning and neural networks. It maps an input value to a value between 0 and 1, which can be interpreted as a probability.

The sigmoid function has an S-shaped curve and is symmetric around the point (0, 0.5). When the input value x is positive, the output of the sigmoid function approaches 1. As x becomes increasingly negative, the output approaches 0.

The sigmoid function is useful for tasks such as binary classification, where the goal is to predict one of two classes. The output of the sigmoid function can be interpreted as the probability of belonging to one of the classes. For example, if the output is 0.8, it can be interpreted as an 80% probability of belonging to one class and a 20% probability of belonging to the other class. The sigmoidal transfer function is expressed as shown in equation 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

Where: $\sigma(x)$ represents the output value of the sigmoid function for the input value x ; e is the base of the natural logarithm, approximately equal to 2.71828.

The advantage of the sigmoid transfer function is that it is smooth, interpretable as a probability, good for binary classification tasks but the disadvantage is that it saturates at high or low values, prone to vanishing gradients, not suitable for deep networks (Rumelhart *et al.*, 1986).

Rectified Linear Unit (ReLU): The Rectified Linear Unit (ReLU) is a commonly used activation function in deep learning neural networks. It is a piecewise linear function that returns zero for negative input values and the input value itself for non-negative values.

Mathematically, the ReLU function is expressed shown in equation 2:

$$F(x) = \text{Max}(0, \infty) \quad (2)$$

Where x is the input to the function and $f(x)$ is the output. If the input x is negative, the ReLU function returns 0. Otherwise, it returns the input value x .

The ReLU function is preferred over other activation functions like sigmoid and hyperbolic tangent because it helps alleviate the vanishing gradient problem, which can occur during the training of deep neural networks. The vanishing gradient problem refers to the issue where the gradients become extremely small as they propagate backward through the network, leading to slow convergence or even the complete inability of the network to learn.

By setting negative inputs to zero, the ReLU function introduces sparsity in the network, allowing only a subset of neurons to be activated. This sparsity helps in better representation learning and can contribute to improved network performance.

The ReLU function is simple to compute and has a computationally efficient gradient computation. However, one drawback of ReLU is that it can cause dead neurons (also known as "dying ReLU") where neurons become non-responsive to any input and always output zero. This can happen if the weights associated with a neuron are updated in such a way that the neuron never activates during training. Several variations of ReLU, such as Leaky ReLU and Parametric ReLU, have been proposed to address this issue.

The Rectified Linear Unit advantages are that is Simple, computationally efficient, overcomes the vanishing gradient problem, effective for deep networks while its disadvantages are is not centered around zero (can cause dead neurons), and may lead to a "dying ReLU" problem (Nair and Hinton, 2010).

Hyperbolic Tangent (Tanh): The hyperbolic tangent (tanh) transfer function is a commonly used activation function in neural networks. It is a non-linear function that squashes the input values between -1 and 1, making it useful for normalizing and scaling data.

The mathematical expression for the hyperbolic tangent function given in equation 3:

$$\text{Tanh}(x) = (e^x - e^{-x}) / (e^x + e^{-x}) \quad (3)$$

Where e is the base of the natural logarithm (approximately 2.71828) and x is the input value. The tanh function has the range, symmetry and non-linear properties, respectively. The range property ensures that the output of the tanh function is bounded between -1 and 1. As x approaches positive infinity, $\text{tanh}(x)$ approaches 1, and as x approaches negative infinity, $\text{tanh}(x)$ approaches -1. However, the symmetry property shows that the tanh function is an odd function, which means $\text{tanh}(-x) = -\text{tanh}(x)$. This property is useful in certain contexts, such as when dealing with balanced positive and negative inputs. The non-linear property makes the tanh function to be non-linear, enabling neural networks to learn complex relationships between inputs and outputs. The tanh function is commonly used in the hidden layers of neural networks, as it helps introduce non-linearities and capture more complex patterns in the data. It can also be used as an alternative to the sigmoid function when the output range of -1 to 1 is desired.

The advantages of the tanh function are; it is smooth, zero-centered, retains non-linearity, suitable for recurrent neural networks (RNNs), however, the disadvantages is similar to the sigmoid function, suffers from vanishing gradients (Bengio *et al.*, 1994).

Leaky ReLU: The Leaky ReLU (Rectified Linear Unit) is a variant of the ReLU activation function commonly used in neural networks. It addresses one of the drawbacks of the traditional ReLU function, which causes certain neurons to become "dead" and unresponsive if their input falls below zero.

The Leaky ReLU function is expressed using equation 4:

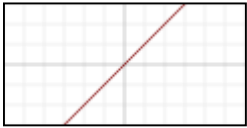
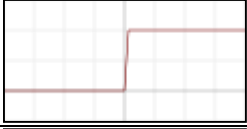
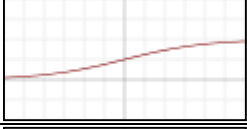
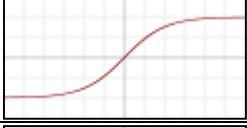
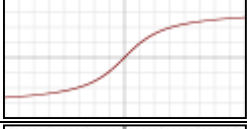
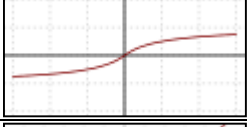


$$f(x) = \max(\alpha x, x) \quad 4$$

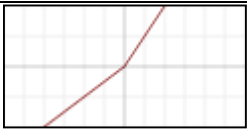
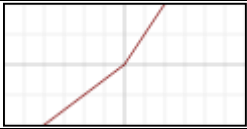
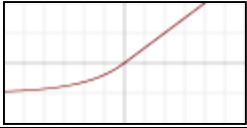
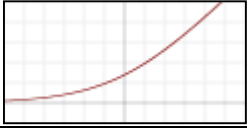

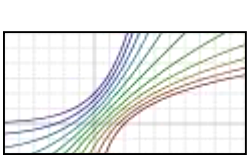
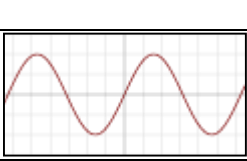
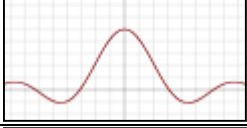

Where x is the input to the function, and a is a small constant typically set to a small positive value (e.g., 0.01 or 0.001), α is a small constant (<1). When x is positive, the function behaves like the standard ReLU and returns x . However, when x is negative, it returns a scaled version of x .

The purpose of introducing the small slope for negative inputs is to allow the flow of a small gradient, preventing neurons from completely "dying" during training. This ensures that even if a neuron's output is negative, there is still a small gradient that can propagate backward and update the weights during backpropagation.

The Leaky ReLU function can be used as an activation function in neural networks, typically in the hidden layers. It has been observed to alleviate the dying ReLU problem and improve the learning capability of deep neural networks, especially in scenarios where there is a high proportion of negative inputs. Its advantages are it overcomes the "dying ReLU" problem, prevents dead neurons, maintains non-linearity while the disadvantages is that it requires tuning the α parameter (Maas *et al.*, 2013). More so, there are many other activation functions available, such as softmax, exponential linear units (ELUs), and scaled exponential linear units (SELUs), each with their own advantages and use cases (Table 1). The choice of activation function depends on the specific task, network architecture, and experimental results. Experiment with different activation functions to identify the one that best represents the underlying hydrologic processes.

Table 1: Comparisons of the properties of some transfer functions (Source: Bergstra *et al.*, 2009).

Name	Plot	Equation	Derivatives with respect to x	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$(0, 1)$
Logistic or sigmoid (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$(-\frac{\pi}{2}, \frac{\pi}{2})$
Softsign		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{1}{(1 + x)^2}$	$(-1, 1)$
Rectified linear unit		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(0, \infty)$
Leaky rectified linear unit		$f(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$

Parametric rectified linear unit		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Randomized leaky rectified linear unit		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Exponential linear unit		$f(\alpha, x) = \begin{cases} \alpha (e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$
SoftPlus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, \infty)$
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$	$(-\infty, \infty)$
Soft Exponential		$f(x) = \begin{cases} -\frac{\ln(1 - \alpha(x + \alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + f & \text{for } \alpha \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(\alpha + x)} & \text{for } x < 0 \\ e^{\alpha x} & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$	$(-1, 1)$
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$	$(\approx -0.217234, \dots)$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	$(0, 1)$

Parameter Initialization / Optimization: Parameter initialization plays a crucial role in training neural networks. Initializing model parameters properly can significantly impact the convergence speed, optimization process, and overall performance of the network. Improper initialization can lead to convergence issues or suboptimal solutions. Techniques like Xavier initialization or He initialization can help in achieving better convergence and performance. Glorot and Bengio (2010) discuss the importance of proper initialization for deep neural networks. Their study revealed that using randomly initialized weights can lead to slow convergence and poor performance. They propose a method for initializing weights that helps to improve convergence and performance. He *et al.*, (2015) introduce the rectifier linear unit (ReLU) activation function. The ReLU function is a simple but effective activation function that can help to improve the performance of deep neural networks. Likewise, Kingma and Ba (2014) introduces the Adam optimizer. The Adam optimizer is a more efficient version of the stochastic gradient descent (SGD) algorithm. The Adam optimizer can help to improve the convergence speed and performance of deep neural networks. The commonly used parameter initialization methods are zero initialization, random initialization, Xavier/Glorot Initialization, He Initialization, and Orthogonal Initialization, respectively. It's important to note that the choice of parameter initialization method depends on the specific network architecture, activation functions used, and the nature of the task. Additionally, modern deep learning frameworks often provide default initialization methods that have been shown to work well in practice (Glorot and Bengio 2010; He *et al.*, 2015 and Saxe *et al.*, 2013).

Hyperparameter Tuning Optimization: Hyperparameter tuning is a crucial step in optimizing the performance of machine learning models. Hyperparameters are configuration settings that determine the behavior and performance of the model, such as learning rate, batch size, regularization strength, and the number of layers or units in a neural network. Effective tuning of these hyperparameters can greatly impact the model's performance and generalization ability. Bergstra and Bengio (2012) introduce the random search algorithm for hyperparameter optimization. Their study showed that random search can be an effective way to find good hyperparameters for a variety of machine learning models. Grids and Bergstra (2017), likewise, establish the HyperOpt library for hyperparameter optimization. The library provides a simple and easy-to-use interface for implementing random search and other hyperparameter optimization algorithms. However,

the study of Shahriari *et al.* (2015) worked with the Bayesian optimization algorithm for hyperparameter optimization and revealed that the Bayesian optimization can be more effective than random search for finding good hyperparameters. Literature has shown that techniques such as grid search or random search are used to explore the hyperparameter space and find the best combination for your specific hydrologic modeling problem. From literature, hyperparameter tuning of artificial neural network for hydrological modelling can be achieved in several ways such as Manual Search, Grid Search, Genetic Algorithms and Automated approaches respectively (Bergstra and Bengio, 2012; Snoek *et al.*, 2012; Real *et al.*, 2019; and Hutter *et al.*, 2019).

Regularization Techniques or Optimization: Regularization techniques or optimization are commonly used in neural network optimization to prevent overfitting, improve generalization, and enhance the model's performance. These regularization techniques can be used individually or in combination to improve the performance and generalization ability of neural networks. The specific choice and combination of regularization techniques depend on the characteristics of the problem, the network architecture, and the available data. Some popular regularization techniques used in neural network optimization include: L1 and L2 Regularization (Weight Decay): L1 and L2 regularization, also known as weight decay, involve adding a penalty term to the loss function during training. This penalty term encourages the neural network to have smaller weights. L1 regularization adds the absolute values of the weights to the loss function, promoting sparsity and feature selection. L2 regularization adds the squared values of the weights, which encourages smaller but non-zero weights. The regularization term controls the trade-off between fitting the training data and keeping the weights small, thus preventing over-fitting.

Dropout: Dropout is a widely used regularization technique that randomly deactivates a fraction of neurons during each training iteration. By doing so, dropout prevents co-adaptation of neurons and encourages the network to learn more robust and generalizable features. Dropout effectively reduces over-fitting by introducing noise and increasing the diversity of the network's activations.

Early Stopping: Early stopping is a simple yet effective regularization technique. It involves monitoring the model's performance on a validation set during training and stopping the training process when the validation loss starts to increase. By stopping

the training before overfitting occurs, early stopping prevents the model from memorizing the training data and improves its generalization ability.

Data Augmentation: Data augmentation is a technique where additional training data is generated by applying various transformations to the existing training samples, such as rotations, translations, flips, or distortions. By increasing the diversity of the training data, data augmentation helps the neural network generalize better and reduces overfitting.

Batch Normalization: Batch normalization is a technique that normalizes the inputs to each layer of the neural network using the mean and variance of the current mini-batch during training. It helps in stabilizing the training process, reducing the internal covariate shift, and accelerating the convergence of the network. Batch normalization acts as a form of regularization by introducing noise in the computation of each layer, thereby reducing overfitting.

Dropout Regularization: In addition to the dropout technique mentioned earlier, dropout regularization can be applied by adding dropout layers to the network architecture. Dropout layers randomly deactivate a fraction of the neurons during training, forcing the network to learn more robust features and reducing overfitting.

Cross-Validation Optimization: When optimizing neural networks, cross-validation methods are commonly used to assess and improve the model's performance. Cross-validation involves dividing the available dataset into multiple subsets and using them for training and evaluation purposes. Some commonly used cross-validation methods for neural network optimization include:

k-Fold Cross-Validation: In k-fold cross-validation, the dataset is divided into k equally-sized subsets or folds. The neural network model is trained and evaluated k times, each time using a different fold as the validation set and the remaining folds as the training set. The results from each iteration are averaged to obtain an overall assessment of the model's performance.

Stratified k-Fold Cross-Validation: Stratified k-fold cross-validation is useful when dealing with imbalanced datasets, where the distribution of classes is uneven. It ensures that each fold maintains the same class distribution as the original dataset, thus preserving the representative nature of the subsets used for training and evaluation. Leave-One-Out Cross-Validation (LOOCV): LOOCV is a special case

of k-fold cross-validation where k is equal to the number of samples in the dataset. In each iteration, the model is trained on all but one sample and evaluated on the left-out sample. This process is repeated for each sample in the dataset. LOOCV provides an unbiased estimate of the model's performance but can be computationally expensive for large datasets.

Holdout Validation: Holdout validation involves splitting the dataset into two sets: a training set and a validation set. The model is trained on the training set and evaluated on the validation set. This method is simple to implement but may lead to high variance in the performance estimation due to the limited amount of data used for validation.

Repeated Random Subsampling Validation: In this method, the dataset is randomly divided into training and validation sets multiple times. The model is trained on the training set and evaluated on the validation set in each iteration. The results are then averaged to obtain an overall performance estimate. Repeated random subsampling validation is useful when computational resources are limited or when the dataset is large. These cross-validation methods help assess the performance of the neural network model and provide insights into its generalization capabilities. They also assist in tuning hyperparameters, such as learning rate, number of hidden layers, and activation functions, to optimize the model's performance. By evaluating the model on different subsets of data, cross-validation helps mitigate overfitting and provides a more reliable estimate of the model's performance on unseen data.

Transfer Learning Optimization: Transfer learning is a powerful technique in neural network optimization that involves leveraging knowledge and learned representations from one task or domain to improve performance on another related task or domain. Some commonly used transfer learning methods in neural network optimization include:

Pre-trained models: Pre-trained models are neural network models that have been trained on a large dataset for a specific task, typically in a different domain. Instead of training a model from scratch, transfer learning involves using the pre-trained model as a starting point and fine-tuning it on a new dataset or task. This approach is particularly useful when the new dataset is small or when the new task is related to the original task the model was trained on. Feature extraction: In this transfer learning approach, the pre-trained model is used as a fixed feature extractor. The early layers of the pre-trained model, which capture low-level and generic features, are retained while the

later layers are replaced with new layers specific to the target task. The pre-trained model's learned representations serve as informative features for the new task, and only the newly added layers are trained on the task-specific data. Fine-tuning: Fine-tuning is a transfer learning technique that involves starting with a pre-trained model and updating the weights of some or all of its layers using the new dataset. Fine-tuning allows the model to adapt to the specific characteristics of the new task while retaining the valuable knowledge learned from the original task. The extent of fine-tuning can vary, from freezing a subset of layers and updating the remaining layers to updating all layers of the pre-trained model.

Domain adaptation: Domain adaptation is a transfer learning method that addresses the problem of differences between the source domain (where the pre-trained model was trained) and the target domain (where the model will be applied). It involves reducing the domain shift by aligning the representations of the source and target domains. Techniques such as adversarial training, which minimizes the discrepancy between domains, or instance reweighting, which assigns higher weights to samples from the target domain, can be used for domain adaptation.

Multi-task learning: Multi-task learning is a transfer learning approach where a single neural network model is trained on multiple related tasks simultaneously. The shared layers of the network learn to extract common representations across tasks, while task-specific layers capture task-specific information. By jointly learning multiple tasks, the model can benefit from the shared knowledge and improve the performance on each individual task. These transfer learning methods help to address challenges such as limited data availability, computational resources, and the need for task-specific optimization. By leveraging knowledge from pre-trained models or related tasks, transfer learning enables more efficient and effective neural network optimization, leading to improved performance on new tasks or domains.

Conclusion: ANNs offer a wide range of transfer functions and each of the transfer function has its strengths and limitations. The contribution of artificial neural network transfer function optimization to knowledge is significant in the field of hydrological modeling and beyond because it improved hydrological modeling accuracy, enhanced generalization capability, improved model flexibility, and interpretability and insights. By optimizing artificial neural network transfer functions, hydrological models can better simulate and predict water-related processes.

REFERENCES

- ASCE Task Committee on Artificial Neural Networks in Water Resources Engineering. (2000). Artificial neural networks in water resources engineering. *J. Hydraulic Engineer.* 126(3), 289-309.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157-166.
- Bergstra, J., and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Machine Learning Res.* 13(Feb), 281-305.
- Campolo, M., Andreussi, P. and A. Soldati (1999). River flood forecasting with a neural network model. *Wat. Resources Res* 35, 1191-97.
- Duan, Q., Sorooshian, S., and Gupta, V. K. (1998). Artificial neural networks for simulating the rainfall-runoff process. *J. Hydrol.* 205(1-4), 407-429.
- Glorot, X., and Bengio, Y. (2010). Understanding the difficulty of training deep feed forward neural networks. *In Proceedings of the 13th international conference on artificial intelligence and statistics* (pp. 249-256).
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning.* MIT Press.
- Grids, D., and Bergstra, J. (2017). HyperOpt: A Python library for hyperparameter optimization. *J. Machine Learning Res.* 18(1), 803-806.
- Gumrah, F., Oz, B., Guler, B. and S. Evin (2000). The application of artificial neural networks for the prediction of water quality of polluted aquifer. *Water Air and Soil Pollution* 119:275-294.
- Haykin, S. (1999). *Neural networks: A comprehensive foundation.* Macmillan.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *In Proceedings of the IEEE international conference on computer vision* (pp. 710-718).
- Hutter, F., Lücke, J., and Schmidt-Thieme, L. (2019). Beyond Manual Tuning of Hyperparameters: Lessons Learned in Algorithm Selection and AutoML. *Data Mining*

- Islam, S. and R. Kothari (2000). Artificial neural networks in remote sensing of hydrologic processes. *Journal of Hydrologic Engineering* 5(2):138–144.
- Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Maier, H.R., Dandy, G.C. and M.D. Burch (1998). Use of artificial neural networks for modelling cyanobacteria *Anabaena* spp. in the River Murray, South Australia. *Ecological Modelling* 105, 257–72.
- Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (pp. 3-11).
- Minns A.W. and Hall M.J. (1996). Artificial Neural Network as Rainfall-Runoff Models. *Hydrological Sciences Journal* 41(3), Pp 399-417.
- Moghaddam, M., and Sorooshian, S. (2001). Artificial neural networks in hydrology: A review. *Journal of Hydrology*, 249(1-4), 3-47.
- Moghaddam, S., and Sorooshian, S. (2017). Artificial neural networks for hydrologic modeling: A review. *Water Resources Research*, 53(1), 1–29. doi:10.1002/wrcr.20538
- Nair, V., and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines.
- Ouenes, A. (2000). Practical application of fuzzy logic and neural networks to fractures reservoir characterization. *Computers and Geosciences* 26(8): 953–962.
- Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized Evolution for Image Classifier Architecture Search. Proceedings of the AAAI Conference on Artificial Intelligence, 33: 4780-4789.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536.
- Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)* (Vol. 32, pp. 1243-1251).
- Shahriari, B.; Kevin, S.; Ziyu W.; Ryan P.A.; Nando F. (2015). Taking the human out of the loop: A scalable bayesian optimization approach to hyperparameter tuning. *arXiv preprint arXiv:1502.02133*.
- Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951-2959).
- Sivakumar, M., Sivapalan, M., and Wood, E. M. (2007). Hydrologic system complexity and nonlinear dynamics. *Hydrological Sciences Journal*, 52(1), 1-18.
- Thirumalaiah, K. and M.C. Deo (1998). Real-time flood forecasting using neural networks. *Computer-Aided Civil and Infrastructure Engineering* 13, 101–11.
- Zhu, J., Sorooshian, S., and Gupta, V. K. (2000). Transfer function selection for artificial neural networks in hydrologic modeling. *Wat. Resources Res.* 36(12), 3575-3586.