Marine Dissolved Oxygen Prediction With Tree Tuned Deep Neural Network

LIHAO WANG, YU JIANG, AND HONG QI

1College of Computer Science and Technology, Jilin University, Changchun 130012, China
2Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China

Corresponding author: Hong Qi (qihong@jlu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 51679105, Grant 51809112, and Grant 51939003.

ABSTRACT Changes in the dissolved oxygen concentration of the ocean have important implications for marine ecosystems and global climate change. However, limited by measurement techniques, the hydrology data is not always complete. Thus, accurate prediction on marine dissolved oxygen concentration (MDOC), is a powerful supplement to the current observation data. Deep neural network is a powerful model to do the prediction, while it is usually difficult and time-consuming to tune its structure. Meanwhile, deep jointly informed neural network (DJINN) provides a user friendly method to tune the structure of neural networks. In this article, a deep learning-based model called marine deep jointly informed neural network (M-DJINN), is proposed to predict MDOC. M-DJINN improves DJINN performance via initializing the weights of neural network using a zero-mean Gaussian distribution with a variance related to the number of neurons in the neighbor layer. In M-DJINN, to get an ideal deep neural network structure, users only need to tune the tree number and max tree depth. While predicting MDOC on World Ocean Database 2013 (WOD13) data with M-DJINN, the novel model proves better than DJINN on both accuracy and convergency. When the max tree depth is set to 10, the mean squared error (MSE) is reduced by 17.6% compared with DJINN. It can be concluded from the experiments that with enough training data, the performance of M-DJINN may keep improving by deepening the neural networks.

INDEX TERMS Marine dissolved oxygen, decision trees, neural networks, DJINN, deep learning.

I. INTRODUCTION

Marine dissolved oxygen concentration (MDOC) is an important parameter in marine chemistry, and one of the most important indicators in seawater quality on the same time. Thus, the oxygen distribution in the world oceans reflects the conditions of marine organisms, such as dead zones [1]. Covering three-quarters of the earth, the ocean plays an important role in the regulation of the global climate. So MDOC is also considered to reflect global warming events [2]. Moreover, oxygen is an important tracer of ocean ventilation and interior ocean circulation. Therefore, the study of MDOC is of great value in the field of oceanography and can help in tracing changes of the geophysical and geochemical systems.

Comprehensive research on MDOC needs substantial data. Many sensors and floats, as the instruments used for making subsurface measurements in the ocean, have been deployed to observe marine dissolved oxygen and other variables in the ocean in some large-scale global observation programs. For example, the World’s Oceans Real-time Network Plan (ARGO) operates and manages more than 3,000 floats distributed in all oceans [3]. Aside from floats, employing sea-going research ships to take bottled water samples is also an alternative way to obtain data. To reorganize the data from different sources, the World Ocean Database (WOD), another ocean-observation program, merges thousands of origins data sets from many different countries and organizations [4]. However, not all the floats in these programs measure MDOC, especially for early floats which only measure temperature and salinity. This leads to fewer samples on the dissolved oxygen concentration, compared to the samples on temperature and salinity. Fig. 1 shows the proportion of total samples on several variables in WOD. New floats with additional items for oxygen observation still have problems such as poor factory calibration [5], and may drift while exposed to light or heat [6]. For bottled water samples, the problems
are required to be tuned well. The performance of DNN is closely related to the tuning of hyperparameters, which is a time-consuming work for those who are not familiar to DNN. Humbird, KD et al. proposed a model referred to as “deep jointly informed neural networks” (DJINN), combining the advantages of both tree-based models and neural networks by initializing the structure of neural network according to decision trees [15]. However, the Xavier initialization used in DJINN drags the prediction accuracy with the increase of max tree depth.

In this article, a novel model called marine deep jointly informed neural network (M-DJINN) is proposed. The model has a well-tuned structure of DNN, and is applied to predict the dissolved oxygen concentration in the ocean. To improve the performance of DJINN in deep learning, we change the method to initialize the weights of neurons in neural networks, so that M-DJINN is expected to get a better accuracy and convergency than DJINN when there are more hidden layers. To accelerate the convergence speed, we also preprocess the data with standardization. Then, we explore the relationship between two hyperparameters of M-DJINN and the performance of the model. Finally, we compare the prediction accuracy between M-DJINN and some other algorithms, and validate M-DJINN on oceanic datasets WOD13. The potential application of the proposed method is promising. For a long time, the tuning of hyperparameters has been an obstacle to researchers on the using of deep learning model. The proposed method would encourage researchers who are inexperienced on tuning hyperparameters to use deep learning model. Especially for oceanic researchers, with the explosive growth of marine data and the improvements on computing in the future, a deeper neural network model is expected to perform well.

The organization of the rest of the research paper is as follows: Section II includes the related work about DJINN and the motivation to improve the algorithm. Section III puts forward a novel model to maximize the advantages of both random forest and neural networks in deep learning. Section IV describes details of the experiment and Section V analyzes the experiment results. In Section VI, conclusions and the vision for further study are given.

II. RELATED WORK

To introduce the proposed method, a discussion on related work is given in this section.

A. A INTRODUCTION TO DJINN

DJINN is a model that combines tree-based models with neural networks. In this way, DJINN is flexible and scalable with few hyperparameters. This model is applicable to both classification tasks and regression tasks. To predict the dissolved oxygen concentration, DJINN for regression tasks is mainly discussed in this article. The core of DJINN is a mapping from decision trees to neural networks. Similar to random forest, users only need to tune two hyperparameters: tree number and the maximum depth of trees. Usually, the tree number and the depth of the trees have positive correlation with the accuracy
of prediction, yet they are not recommended to be excessive large due to limited training time and memory demands. For different dataset, these two hyperparameters are suggested to be tuned specifically. After the mapping, DJINN gets the initialized neural networks, whose structure has been tuned well.

Table 1 gives the description of symbols used in DJINN. For the level \( l \) in the decision tree, DJINN uses arrays \( W^l \) of dimension \( n(l) \times n(l - 1) \) and \( W^{D_l} \) with dimension \( n(D_l) \times N_{out} \) to store initial weights of neural networks. The weights of most neurons are initialized with Xavier initialization, except that for \( i \) from 0 to \( N_{in} - 1 \), \( W^l_{i,i} \) is initialized to 1 for \( l < L_{i}^{max} \) to ensure input values are passed through hidden layers until the decision tree no longer splits on them.

| Symbol | Description |
|--------|-------------|
| \( D_l \) | The maximum decision tree depth |
| \( D_0 \) | \( D_l - 1 \) |
| \( l \) | A layer in neural network or decision tree |
| \( n(l) \) | The number of neurons in each hidden layer |
| \( N_{in} \) | The number of neurons in the input layer |
| \( N_{out} \) | The number of neurons in the output layer |
| \( W^l \) | A \( n(l) \times n(l - 1) \) array to store the weights of neural networks in level \( l \) |
| \( W^l_{i,i} \) | An element of \( W^l \) in row \( i \), column \( i \) |
| \( L_{i}^{max} \) | The record max depth each input occurs as a branch |

When building the neural networks, DJINN uses the rectified linear unit for each hidden layer. During the optimization, the cost function is set to be mean squared error (MSE). The Adam optimizer [26] is used in DJINN for regression to minimize the cost function.

**B. REFLECTION ON DJINN**

DJINN employs the rectified linear unit to build the neural network. Rectified linear unit is a kind of neuron that uses a rectifier function as the activation function [16]:

\[
\text{rectifier}(x) = \max(0, x)
\]

The rectifier function allows a network to obtain sparse representation easily, bringing benefits due to information disentangling, efficient variable-size representation, and linear separability.

On the same time, DJINN employs Xavier initialization to initialize the weights of neural networks. The algorithm chooses zero-mean normal distribution with variance of \( 3/(n_{in}+n_{out}) \) to initialize weights of neurons in level \( i \), where \( n_{in} \) and \( n_{out} \) represents the number of neurons in level \( i \) and the number of neurons in level \( i + 1 \) respectively.

However, to allow gradients to flow backwards, the expected performance of Xavier initialization relies on two assumptions [17]:

- The activation function of the neuron \( f(x) \) is linear and symmetric about the y-axis.
- The variances of input features follow the same distribution.

Obviously, the first assumption fails for the rectifier function, as the function is neither linear nor symmetric about the y-axis. This contradiction may result in the gradient diminishing, and the learning process of the model would probably be stalled. This finding motivates us to improve DJINN, and our improvements is to be presented in Section III.

**III. METHODOLOGY**

With only few easy-to-understand hyperparameters, DJINN with Xavier initialization method can provide a relative high accuracy in prediction and brings much convenience to users. However, the first assumption of Xavier initialization is true only when the activation function is nonlinear, otherwise Xavier initialization may drag the performance on prediction error and convergence [18]. For deep neural networks trained with large ocean datasets, the disadvantage may be even worse. In our improvements, we still keep the second assumption of Xavier initialization.

**A. INITIALIZATION OF THE NEURAL NETWORK**

In neural networks that use \( f(x) \) as the activation function, for the level \( i \), if we write \( W_i \) for the weights vector of neurons, \( Z_i \) for the activation vector, \( b_i \) for the bias vector of neurons, and \( S_i \) for the argument vector, we have

\[
S_i = W_i \cdot Z_i + b_i
\]

\[
Z_{i+1} = f(S_i)
\]

Obviously, with \( \text{Var}(x) \) represents the variance of variable \( x \), and \( b_i \) initialized to zero, there exists:

\[
\text{Var}(S_i) = \text{Var}(W_i \cdot Z_i)
\]

If we write \( w_i, z_i \) and \( s_i \) for random variables of each elements in \( W_i, Z_i \) and \( S_i \) respectively, and \( n_i \) for the number of connected neurons in the parent level, then we get:

\[
\text{Var}(s_i) = n_i \cdot \text{Var}(w_i \cdot z_i)
\]

Note that the following equations exists when \( w_i, z_i \) follow the independent identical distribution:

\[
\text{Var}(w_i \cdot z_i) = (E(w_i^2) - E(w_i)^2)(E(z_i^2) - E(z_i)^2)
\]

\[
E(w_i^2) = (E(w_i)^2) + E(Z_i)^2 + E(Z_i)^2
\]

where \( E(x) \) represents the expectation of variable \( x \).

Taking Equation (7) into Equation (6), we have:

\[
\text{Var}(s_i) = n_i((E(w_i^2) + E(Z_i)^2)(E(z_i^2) + E(Z_i)^2)
\]

\[
- E(Z_i)^2E(Z_i)^2)
\]

Here is the point. With the first assumption, the Xavier initialization simplifies Equation (8) by let \( E(Z_i) \) equal to zero and get its conclusion. To further improve the effect of initialization, we substitute the Xavier initialization in DJINN
with another initialization method which is designed specifically for nonlinear activation functions such as the rectifier function [18].

As we have discussed in Section II-B, the rectifier function is nonlinear and not symmetric about the y-axis. So \( E(z) \) is not zero now. We assume that \( E(w_i) \) is zero, and from Equation(8), we get:

\[
\text{Var}(s_i) = n_i \cdot \text{Var}(w_i) \cdot E(z_i^2)
\]

Note that

\[
\text{Var}(s_{i-1}) = E(s_{i-1}^2) - E^2(s_{i-1})
\]

If we let \( w_{i-1} \) have a symmetric distribution around zero and \( b_{i-1} \) initialized to zero, then \( s_{i-1} \) has zero mean and has a symmetric distribution around zero [17]. This leads to

\[
\text{Var}(s_{i-1}) = E(s_{i-1}^2)
\]

If we write \( g(s_i) \) as the probability density function of \( s_i \), then we have:

\[
E(s_{i-1}^2) = \int_{-\infty}^{\infty} s_{i-1}^2 g(s_{i-1}) ds_{i-1}
\]

Meanwhile, note that \( E(z_i^2) \) can be written in:

\[
E(z_i^2) = \int_{-\infty}^{\infty} z_i^2 g(z_i) dz_i
\]

For rectified linear unit, this leads to:

\[
E(z_i^2) = \int_{0}^{\infty} s_{i-1}^2 g(s_{i-1}) ds_{i-1}
\]

With the distribution of \( s_i \) being symmetric around zero, \( s_{i-1}^2 g(s_{i-1}) \) becomes an even function. Therefore, combining Equation(12) and Equation(14), we have:

\[
E(z_i^2) = E(s_{i-1}^2)/2
\]

Combining Equation(11) and Equation(15), we have:

\[
E(z_i^2) = \text{Var}(s_{i-1})/2
\]

Then Equation (9) transforms to:

\[
\text{Var}(s_i) = (n_i \cdot \text{Var}(w_i) \cdot \text{Var}(s_{i-1}))/2
\]

In DJINN, there are \( D_n \) hidden layers. If we set \( L \) equal to \( D_n \), and put them together, we get:

\[
\text{Var}(s_L) = \text{Var}(s_1) (\prod_{i=2}^{L} n_i \cdot \text{Var}(w_i))/2
\]

To keep the information flow, we expect \( \text{Var}(s_L) \) to be equal to \( \text{Var}(s_1) \) [18]. Thus, we have the conclusion of initialization method:

\[
n_i \cdot \text{Var}(w_i) = 2
\]

From Equation (19) and the assumption that the weights have zero mean, this initialization method lets the weight \( w_i \) follow a zero-mean Gaussian distribution and sets the variance of the distribution to \( n_i/2 \). There are two points to mention. Firstly, for the first layer, \( n_1 \cdot \text{Var}(w_1) \) is supposed to be 1 because there is no rectified linear unit applied on the input signal. However, for simplicity, \( n_1 \cdot \text{Var}(w_1) \) can be just set to 2. Secondly, in this method, Equation(17) is applicable to either forward propagation or backward propagation and can keep information flowing under both circumstances [18].

### B. M-DJINN ALGORITHM AND OPTIMIZATION

The inputs of M-DJINN are the maximum tree depth and the tree number. The output of M-DJINN is an ensemble of finely tuned and initialized neural networks, whose number is decided by the tree number. To begin with, according to the inputs, decision tree-based models would be constructed. Then, to do the mapping from a tree to a neural network, M-DJINN recurses the decision path of tree-based models twice, first to determine the structure and then to initialize the weights.

Algorithm 1 is presented to demonstrate the mapping from a single decision tree to the neural network. Define the primary branch of the tree as the \( l = 0 \) level, and the maximum tree depth as \( D_m \), thereby the maximum branch depth is \( D_m - 1 \), denoted by \( D_n \). After the first recurse, max branch depth (\( D_n \)), max depth where each input occurs as a branch (\( L_{\text{max}}^m \)) and number of branches at each level (\( N_b(l) \)) are determined. The mapped neural network consists of an input layer corresponding to \( l = 0 \) level of the trees, \( D_n \) hidden layers and an output layer. For M-DJINN that has \( N_{\text{in}} \) inputs and \( N_{\text{out}} \) outputs, there would be \( N_{\text{in}} \) neurons in the input layer and \( N_{\text{out}} \) neurons in the output layer. For hidden layers with \( n(l) \) neurons, \( n(l) \) can be defined as follows:

\[
n(l) = n(l - 1) + N_b(l)
\]

For level \( l \) from 1 to \( D_m \), denote the tree node as \( t \), the mapped decisive path of \( t \) as \( c \), and the neuron created by the parent branch as \( p \). If \( t \) is branch, then add a new neuron to this layer according to Equation (18), Connect \( p \) and the new neuron, \( c \) and the new neuron. If \( t \) is leaf, connect \( c \) to output directly. M-DJINN uses the initialization method in Section III-A to initialize the weights of neurons. The biases of neurons are initialized to zero. Repeat above steps for each tree nodes in the level \( l \), and connected neural networks are built.

For M-DJINN with multiple decision trees, the mapping is supposed to be repeated for each tree and an ensemble of neural networks is created in this way.

To sum up, after the pre-training of decision trees, we construct the neural networks according to algorithm 1. Subsequently, we tune the weight of the neural networks by backpropagation. For regression tasks, such predicting MDOC, we choose MSE as the cost function of M-DJINN. To minimize the cost function, Adam optimizer is employed due to its low demands for memory and robustness. The M-DJINN model was implemented in TensorFlow [19].

### C. AN EXAMPLE OF M-DJINN

For the purpose of a clear understanding of M-DJINN, an example is given in the following to illustrate the mapping.
Algorithm 1 Initialization of Neural Network Based on Decision Trees

Input:
Max branch depth: \( D_n \);
Branch number of each level \( N_p(l) \);
Max depth of each branch with respect to each input: \( L_{l}^{\text{max}} \);

Output:
M-DJINN model \( M \);

// create arrays to store weight
1: for \( l = 1, 2, \ldots, D_n \) do
2: \( n(l) = n(l-1) + N_p(l) \);
3: create \( n(l) \times n(l-1) \) array \( W^l \) and initialize \( W^l \) to zero;
4: end for
5: Create \( n(D_n) \times N_{\text{out}} \) array \( W^{D_n+1} \) and initialize \( W^{D_n+1} \) to zero;
6: for \( i = 0, 1, \ldots, N_{\text{in}} - 1 \) do
7: Let \( W_{i,i}^l = 1 \), for \( l < L_i^{\text{max}} \)
8: end for

// recurse decision paths of the tree
9: for levels \( i = 1, 2, \ldots, D_n \) do
10: for each node \( t \) in level \( i \) do
11: Define the neuron created by the parent branch as
\( p \), the mapped decisive path of \( t \) as \( c, \gamma \) as the number of neurons in the layer \( i-1 \) or level \( i \) + 1, and output neurons as \( \text{out} \) for regression tasks.
12: if \( \text{type}(t) = \text{branch} \):
13: Add a new neuron to layer \( i \);
14: Connect \( p \) and new neuron, \( c \) and new neuron;
15: Initialize \( W_{\text{new},p}^l \sim N(0, \frac{2}{\gamma}) \),
\( W_{\text{new},c}^l \sim N(0, \frac{2}{\gamma}) \);
17: else if \( \text{type}(t) = \text{leaf} \):
18: Initialize \( W_{p,p}^l \sim N(0, \frac{2}{\gamma}) \) for \( l \) from \( i + 1 \) to \( D_n - 1 \)
19: Initialize \( W_{D_n,\text{out}}^l \sim N(0, \frac{2}{\gamma}) \)
20: end if
21: end for
22: end for
23: Output the initialized neural network

from decision trees to neural networks. Fig. 2 gives a vivid view of the relationship between a decision tree and a mapped neural network. According to M-DJINN, \( D_n \) of the decision tree in Fig. 2(a) is 3, with \( N_p(l) = (1, 2, 3, 0) \) for \( l = 0, 1, 2, 3 \), and \( L_{l}^{\text{max}} = (2, 2, 2) \) for \( z_1, z_2 \) and \( z_3 \). Weights of the neural network in Fig. 2(b) are initialized to 0 at the beginning, and elements whose levels are less than \( L_{l}^{\text{max}} \) on the diagonal of \( W^l \) are initialized to 1 afterwards. To construct the structure of neural network, we recurse every decision path of the decision tree.

To begin with, we look at the decision path 1 presented by yellow lines. For level 1, add neuron \( n_{14} \) and connect \( z_1 \) to \( n_{14} \), \( z_3 \) to \( n_{14} \). For level 2, add neuron \( n_{26} \) and find the neuron created by the parent branch of \( n_{26} \) is \( n_{14} \), so we connect \( n_{14} \) and \( n_{26} \) firstly. Note that the mapped decisive path \( t \) of node \( z_1 \leq d_4 \) in level 2 is \( z_1 \) to \( n_{11} \) because the path transmits the value of \( z_1 \). Thus, we connect \( n_{11} \) and \( n_{26} \).

Finally, decision path 1 leads to output \( A \), so we connect \( n_{26} \) and \( A \). The corresponding elements of \( W \) are updated according to M-DJINN during the process.

As for decision path 5, in level 1, add neuron \( n_{15} \) and connect \( z_1 \) to \( n_{15} \), \( z_2 \) to \( n_{15} \). In level 2, connect \( n_{15} \) and \( n_{25} \) because node \( z_2 \leq d_2 \) connects to \( A \) directly on decision path 5, and to transmit the value, we also connect \( n_{25} \) to output \( A \) in level 3 directly.

The decision path 2, 3, 4, 6 and 7 can be recursed similarly to the decision path 1. One detail is that some neurons in the neural network are initially unconnected, which are painted in gray. These neurons could not learn, so they are not included in the final architecture of initialized neural networks.

IV. EXPERIMENTS
In this section, we use deep neural networks tuned by M-DJINN to predict the dissolved oxygen in the ocean.
The performance is evaluated on World Ocean Database 2013 (WOD13) provided by National Science & Technology Resource Sharing Service Platform of China [20]. We also validate M-DJINN with several baseline methods. The results of the experiment in discussed in Section V.

The reason why we choose M-DJINN in this problem area is that the proposed method combines the advantages of both decision trees and neural networks. To the best of our knowledge, it is helpful in this problem area, because for researchers who are inexperienced in the tuning of deep learning models, the proposed method can reduce the workload. What’s more, the proposed method is also observed to have a better performance on accuracy and convergence than other similar methods.

### A. DATASETS

WOD13 datasets [21] collects several important oceanographic variables annually, including temperature, salinity, phosphate, and dissolved oxygen. The ocean data in WOD13 are collected globally. The red points in Fig. 3 show the distribution of the ocean stations that provides bottle data for WOD13. We can see that WOD13 datasets comes from many kinds of ocean areas, including but not limited to coastal areas, gulf areas, the arctic ocean, and the equatorial ocean. It is quite difficult to build traditional or mathematical models that hold good in all those ocean areas. However, with plentiful training data, well-tuned deep learning models are expected to be capable to build a universal model for these ocean areas. We randomly choose the observed data from year 2001 to year 2010. The observation points are randomly chosen from the global ocean, and depth ranges from zero to 4200 meters.

![FIGURE 3. The distribution of the ocean stations in WOD13 datasets.](image)

The predictors we choose include temperature and salinity, since warmer water holds less oxygen and salinity also affect oxygen solubility [10]. Other factors, including biological activity [22], longitude and latitude, ocean-atmosphere interactions and ocean circulations are considered to influence the concentration of oxygen too. So, we use phosphate, salinity, temperature, depth, silicate, longitude, latitude, and chlorophyll as the predictors in experiments.

### B. PREPROCESSING DATA

This section presents the preparation before we apply the algorithms to datasets. After these works, we randomly select 80 percent of samples as the training data of each model, and the remaining 20 percent of samples are used as validation data.

In the original WOD13 datasets, due to different sources, some observed data may have quality problems such as excessive gradient and depth inversion. The WOD program has checked the datasets and labeled the error samples with flags. Hence, we filter the data via picking out samples whose flags are set to error in WOD13. Besides, for some reason, some samples lack values at some variables, so we also have to filter these samples out of the datasets.

After choosing 10,000 samples from the cleaned datasets randomly, we find that the ranges of features differ largely. For example, the value of depth ranges from 0 to 4200 meters, while the value of oxygen concentration ranges from 0 to 16 µmol/kg.

In training and fitting, the value of the cost function may be dominated by the large eigenvalue, which leads to the long running time of the program and the slow convergence process of the gradient. Considering that M-DJINN uses MSE as the cost function, it is necessary to scale features before training and fitting.

Here, we choose z-score model to standardize the data. Z-score model can make all the features have zero mean and unit variance with the following formula to transform data:

\[
X' = \frac{X - \mu}{\sigma} \tag{21}
\]

where \(X\) denotes the feature, \(\mu\) denotes the mean value of \(X\), \(\sigma\) denotes the standard deviation of \(X\), and \(X'\) denotes the result. The comparisons between original data and transformed data on depth and temperature are shown in Table 2.

| Depth | Temperature |
|-------|-------------|
| Unused FS | Used FS | Unused FS | Used FS |
| 1.00 | -0.40 | 1.56 | -1.61 |
| 4.00 | -0.37 | 4.43 | -1.13 |
| 50.00 | 0.09 | 8.28 | -0.49 |
| 10.0 | -0.39 | 15.52 | 0.70 |
| 40.11 | -0.01 | 12.79 | 0.25 |
| 0.49 | 0.44 | 10.42 | -0.14 |
| 168.05 | 1.28 | 8.67 | -0.43 |
| 20.00 | -0.21 | 5.20 | -1.01 |
| 0.00 | -0.41 | 2.78 | -1.41 |
| 1.00 | -0.40 | -0.05 | -1.88 |

### C. BASELINE METHODS

We validate the prediction performance of M-DJINN with the models in the following:

1) **DJINN**: DJINN is a neural network model whose structure is tuned by decision trees. This method is set to show the performance of finely tuned neural networks.
with the state-of-art technique. This model initializes the weights of neurons with Xavier initialization.

2) Random Forest: Random forest is an ensemble model based on decision trees. To get a good result, there are two hyperparameters required to be tuned: tree number and the maximum tree depth. To make the comparison fair, we set the tree number and the maximum tree depth in random forest respectively equal to the tree number and maximum tree depth in M-DJINN.

3) Support Vector Regression: Support vector regression (SVR) [23] is a statistical machine learning model for regression tasks. To use it, a kernel function should be selected carefully, and hyperparameters including penalty parameter C and kernel coefficient gamma, are also required to be tuned. To make the comparison fair, we use grid-search technique to tune these hyperparameters. We set C to range in \{0.001, 0.01, 0.1, 1\} and gamma to range in \{0.5, 1, 1.5, 2, 2.5\}. The kernel function of SVR is radial basis function(RBF).

4) Multilayer Perceptron: Multilayer perceptron (MLP) is a neural network model with multiple hidden layers. It is generally considered as a subset of DNN. In deep learning, the structure of MLP is supposed to be designed elaborately, which is difficult for researchers who are not familiar with deep learning. In the experiment, the method is set to show the performance of trivially tuned neural networks. We set the number of layers of MLP equal to the max tree depth in M-DJINN, and optimize MLP via stochastic gradient descent (SGD).

D. EVALUATION METRICS

To quantify the quality of a model’s prediction, we measure different models via two metrics: Mean Squared Error, Accuracy and Explained Variance Score [24].

Mean Squared Error (MSE) is defined in the following:

\[
MSE(y_i, y'_i) = \frac{1}{n} \sum_{i=1}^{n} (y_i - y'_i)^2
\]  

(22)

Here, the number of samples is n, y_i represents the true value of sample i, and y'_i represents the predicted value of sample i. The smaller the MSE, the more accurate the predicted value is. From MSE, we define the accuracy of model as follows:

\[
\text{Accuracy}(y_i, y'_i) = \frac{1}{n} \sum_{i=1}^{n} (y_i - y'_i)^2
\]  

(23)

Explained variance score (Expl_Var) is defined in the following:

\[
\text{Expl_Var}(Y, Y') = 1 - \frac{\text{Var}(Y - Y')}{{\text{Var}(Y)}}
\]  

(24)

Here, Y is the estimated output, Y' is the true output with respect to Y, and Var is variance. As is shown in Equation (22), explained variance score ranges in [0,1]. In fact, the best score is 1, and the lower the score, the worse the prediction explained by the variance of dependent variables is.

V. RESULTS AND DISCUSSION

In this section, we evaluate M-DJINN model under different hyperparameters and compare the results with other common methods. We also analyze the convergence of M-DJINN. For the regression tasks in the following, we choose the predictors discussed in Section IV-A.

A. INFLUENCE OF TREE NUMBER ON M-DJINN

In this section, we discuss the influence of the number of decision trees on M-DJINN. The experiment is conducted on the training set and the validation set. We first set the limit on max tree depth to 5, and the number of decision trees in \{4, 6, 8, 10, 15, 20\}. The evaluation of metrics is shown in Table 3. It can be concluded from the results that in general, the more the decision trees, the more accurate the M-DJINN is. Nevertheless, when tree number increases to a certain level, such as 15 or 20 listed in Table 3, the model may become too complicated and result in the overfitting of model [25]. A suggested method to prevent this phenomenon is to increase the scale of the datasets.

| Tree Number | MSE   | Expl_Var |
|-------------|-------|----------|
| 4           | 0.495 | 0.858    |
| 6           | 0.405 | 0.883    |
| 8           | 0.312 | 0.910    |
| 10          | 0.305 | 0.912    |
| 15          | 0.308 | 0.911    |
| 20          | 0.307 | 0.912    |

B. COST EFFECTIVENESS OF M-DJINN

M-DJINN chooses MSE as the cost function. The experiment in this section is conducted on the training set. Similar to [15], we use training epochs to measure the cost effectiveness of M-DJINN, and compare its performance with DJINN. We can see the performance on cost effectiveness between M-DJINN and DJINN from the Fig 4. There are 5 decision trees in M-DJINN here, and in Figure 5(a) we set the max tree depth to 8 for both M-DJNN and DJINN. For Fig. 4(b), we set the max tree depth to 10. The curves of cost function show that the converge process of M-DJINN is relative fast, and M-DJINN is observed to start at, and often converge to a lower cost than DJINN under the condition.

The results supports our idea in Section III-A. Because of the different initialization methods, M-DJINN has a better performance on convergence than DJINN. As the number of layers grows, M-DJINN allows the forward and backward signal to flow, while DJINN fails to make it.

We also observe that the tuning on the max tree depth influence the cost effectiveness of M-DJINN. In experiment, we set the number of trees to 5, and max tree depth ranges...
in \{6, 8,10,14\}. The comparison is shown in Fig. 5. We can see the performance of M-DJINN improves when the max tree depth increases from 6 to 8, and is the best at 10. When the max tree depth increases to 12 or 14, the convergence of M-DJINN drops. The result reveals that the max tree depth should be carefully optimized for each data set.

Because of the cost on pre-trained decision trees, the computation complexity of M-DJINN is higher than simple tuned MLP. However, to reach a comparable performance with M-DJINN, the hyperparameters of MLP are supposed to be fine-tuned, whose cost is much more than pretraining tree models[15]. Overall, M-DJINN is still a robust model with competitive cost effectiveness.

C. COMPARISON ON ACCURACY

In the section, we apply M-DJINN to predict the marine dissolved oxygen concentration and validate its performance by comparing its performance with DJINN and other baseline methods. We use the training set to train the model, and use the validation set to validate the model. We set the number of decision trees to 5, and the depth ranges in \{9, 10, 11, 12\} for random forest, MLP, DJINN and M-DJINN. We also compare the performance between M-DJINN and Support Vector Regression. The MSE of SVR is 0.332, and explained variance score is 0.904. The final results of M-DJINN, DJINN, random forest, SVR and MLP are evaluated by MSE and explained variance score. The comparisons between baselines and M-DJINN are shown in Fig. 6. Table 4 show the improvement rate of M-DJINN on MSE with different max tree depth.

We also conduct a t-test on MSE between M-DJINN and other baseline methods to verify the significance of the result. We assume that the performance of methods have identical average(expected) values. The result of t-test is shown in table 5, where all the p-values are less than 0.05. From this result, the hypothesis of equal averages is rejected, which supports that M-DJINN performance better than other baseline methods.

From Fig. 6, we can see that compared with roughly tuned deep neural networks, such as MLP in the experiment, deep neural networks that tuned by M-DJINN outperforms visibly. With our adaptation, M-DJINN also outperforms DJINN on different depth. Compared with other baselines, M-DJINN still performs better. Besides, along with max tree depth increase, the MSE of M-DJINN first drops to the smallest, and then increase slightly. On explained variance score, M-DJINN also performs slightly better than the other algorithms. Note that the max tree depth determines the depth of neural network, the results of experiment indicate that for deep learning, M-DJINN shows its advantage over the other baseline methods.

The result support our idea in Section III-A from the aspect of accuracy. For deeper neural networks, the gradients of DJINN would diminish when training the model, which drags its performance on accuracy. Meanwhile, the decision trees in random forest are confined to on-axis split traditionally, which limits its accuracy.
It is also observed that when the max depth of decision tree reaches a certain level, such as 12, the performance of M-DJINN may drop, which is probably caused by overfitting. However, as the scale of oceanic data is increasing nowadays, the influence of overfitting may drop with enough training data.

Considering the time cost and the benefits, we didn’t use the grid-search technique to tune the hyperparameters of the proposed methods. In experiments, although the performance of M-DJINN improves with more decision trees, we find out that when the number of decision tree is more than 6, the performance stalls. Besides, we also observe that under the same number of trees, M-DJINN reaches its best performance when the max depth is 10. So in the trade-off between time cost and accuracy, we set the number of decision trees to be 5, and the max tree depth to be 10. Table 5 shows the comparison between algorithms under these two hyperparameters. Compared with DJINN, random forest, SVR and MLP, the MSE of M-DJINN was reduced by 17.6%, 20.5%, 37.9% and 29.0% respectively.

In Table 4, we also calculate the accuracy of algorithms, and M-DJINN still performs better than other baselines. After training the M-DJINN model with 10,000 processed samples, we randomly choose 1,000 positions around the ocean and predict the MDOC on these positions. The curves of true value and predicted value are drawn in Fig. 7, and it can be observed that the predicted value fits the true value quite well.

### VI. CONCLUSION AND VISION FOR FUTURE WORK

In this article, we put forward an adapted model, M-DJINN, to apply DJINN in deep learning. The novel model keeps the user-friendly features of DJINN, but achieves a reduced prediction error and a better convergence. We evaluate our algorithm on WOD13 datasets and the experiments show that with enough data to train and tune hyperparameters, the performance of M-DJINN is better than DJINN, random forest, SVR and MLP on both MSE and explained variance score. A potential limit of the proposed method is that
M-DJINN is formulated for fully connected feedforward neural networks. However, for more complex neural networks such as ones with convolution layers, M-DJINN is still useful after the features are extracted by convolution layers. Deep learning models with well-tuned structures can help us to learn the potential law of nature more easily. For researchers who are not familiar with tuning deep neural networks, our adapted model provides a user-friendly solution with better accuracy and convergency. We will continue to improve the deep learning models and predict the various hydrological features, and apply the results to a deeper knowledge of the complex dynamics that govern the geophysical and geochemical systems of our planet.

ACKNOWLEDGMENT

The World Ocean Database 2013 data used here were collected and made freely available at https://www.nodc.noaa.gov/OC5/WOD13/ by the National Centers for Environmental Information.

The authors acknowledged the Data Support from the National Marine Science Data Center, National Science and Technology Resource Sharing Service Platform of China.

Constructive comments from the anonymous reviewers were greatly appreciated.

REFERENCES

[1] R. J. Diaz and R. Rosenberg, “Spreading dead zones and consequences for marine ecosystems,” Science, vol. 321, no. 5891, pp. 926–929, Aug. 2008.
[2] H. Song, P. Wignall, H. Song, X. Dai, and D. Chu, “Seawater temperature and dissolved oxygen over the past 500 million years,” J. Earth Sci., vol. 30, no. 2, pp. 236–243, 2019.
[3] Argo Data anage Team. (Nov. 2019). Argo User’s Manual. http://dx.doi.org/10.13155/29825
[4] T. P. Boyer, J. I. Antonov, O. K. Baranova, C. Coleman, H. E. Garcia, A. Grodzky, D. R. Johnson, R. A. Locarnini, A. V. Mishonov, T. D. O’Brien, C. R. Paver, J. R. Reagan, D. Seidov, I. V. Smolyar, M. M. Zweng, World Ocean Database. Sydney, NSW, Australia: NOAA Atlas, 2013.
[5] J. Hervé et al., “The scientific rationale, design and implementation plan for a biogeochemical-argo float array,” 2016, doi: 10.13155/46601.
[6] K. S. Johnson, J. N. Plant, L. J. Coletti, H. W. Jannasch, C. M. Sakamoto, S. C. Riser, D. D. Swift, N. L. Williams, E. Boss, N. Haintjens, L. D. Talley, and J. L. Sarmiento, “Biogeochemical sensor performance in the SOCCOM profiling float array,” J. Geophys. Res., Oceans, vol. 122, no. 8, pp. 6416–6436, Aug. 2017.
[7] R. J. Matear and A. C. Hirst, “Long-term changes in dissolved oxygen concentrations in the ocean caused by protracted global warming,” Global Biogeochem. Cycles, vol. 17, no. 4, pp. 1–5, Dec. 2003.
[8] V. Naik and S. Manjappa, “Mathematical modeling for dissolved oxygen prediction in mixing zone—A case study,” J. Environ. Sci. Eng., vol. 53, pp. 429–436, Oct. 2011.
[9] Y. Jiang, Y. Gou, T. Zhang, K. Wang, and C. Hu, “A machine learning approach to argo data analysis in a thermocline,” Sensors, vol. 17, no. 10, p. 2225, Sep. 2017.
[10] A. C. Ross and C. A. Stock, “An assessment of the predictability of column minimum dissolved oxygen concentrations in chesapeake bay using a machine learning model,” Estuarine, Coastal Shelf Sci., vol. 221, pp. 53–65, May 2019.
[11] D. Giglio, V. Lymbchich, and M. R. Mazloff, “Estimating oxygen in the southern ocean using argo temperature and salinity,” J. Geophys. Res., Oceans, vol. 123, no. 6, pp. 4280–4297, Jun. 2018.
[12] X. Ji, X. Shang, R. A. Dahlgren, and M. Zhang, “Prediction of dissolved oxygen concentration in hypoxic river systems using support vector machine: A case study of wen-rui tang river, China,” Environ. Sci. Pollut. Res., vol. 24, no. 19, pp. 16062–16076, Jul. 2017.
[13] L. Breiman, “Random forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.
[14] Y. Bengio, “Learning deep architectures for AI,” Foundation, vol. 2, pp. 1–55, Feb. 2009.
[15] K. D. Humbird, J. L. Peterson, and R. G. McLaren, “Deep neural network initialization with decision trees,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 5, pp. 1286–1295, May 2019.
[16] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” J. Mach. Learn. Res., vol. 15, no. 4, pp. 315–323, 2011.
[17] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” J. Mach. Learn. Res., vol. 9, pp. 249–256, May 2010.
[18] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in Proc. Int. Conf. Comput. Vis., 2015, pp. 1026–1034.
[19] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous distributed systems,” arXiv:1603.04467, 2016. [Online]. Available: https://arxiv.org/abs/1603.04467
[20] National Marine Science Data Center. National Science & Technology Resource Sharing Service Platform of China. Accessed: Jan. 30, 2020.
[21] D. R. Johnson, T. P. Boyer, H. E. Garcia, R. A. Locarnini, O. K. Baranova, and M. M. Zweng, “World ocean database 2013 user’s manual,” NODC, Sydney, NSW, Australia, Tech. Rep. 22, p. 172, 2013, doi: 10.7209/V5DF6P53.
[22] K. Sutherland, S. Wankel, and C. Hansel, “Dark biological superoxide production as a significant flux and sink of marine dissolved oxygen,” Proc. Nat. Acad. Sci. USA, vol. 117, no. 17, pp. 3433–3439, 2020.
[23] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik, “Linear Support Vector Regression Machines,” in Proc. Adv. Neural Inf. Process. Systems, 1996, pp. 151–161.
[24] S. Raschka and V. Mirjalili, Python Machine Learning, 2nd ed. Birmingham, U.K.: Packt, 2017.
[25] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
[26] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learn. Represent., 2014, pp. 1–5.

LIHAO WANG is currently pursuing the degree in computer science with the College of Computer Science and Technology, Jilin University, Changchun, China, through the Tang Aoqing Honors Program. His research interests include sensor data fusion and mining, wireless sensor networks, and computer vision.

YU JIANG received the Ph.D. degree from the College of Computer Science and Technology, Jilin University, Changchun, China, in 2011. He is currently an Associate Professor with the College of Computer Science and Technology, Jilin University. His research interests include sensor data fusion and mining, visual perception, and decision making.

HONG QI received the Ph.D. degree from the College of Computer Science and Technology, Jilin University, Changchun, China, in 2005. She is currently an Associate Professor with the College of Computer Science and Technology, Jilin University. Her research interests include sensor data fusion and mining, visual perception, and decision making.

* * *

L. Wang et al.: Marine Dissolved Oxygen Prediction With Tree Tuned DNN