Random-Forest-Inspired Neural Networks

SUHANG WANG, Arizona State University
CHARU AGGARWAL, IBM T. J. Watson Research Center
HUAN LIU, Arizona State University

Neural networks have become very popular in recent years, because of the astonishing success of deep learning in various domains such as image and speech recognition. In many of these domains, specific architectures of neural networks, such as convolutional networks, seem to fit the particular structure of the problem domain very well and can therefore perform in an astonishingly effective way. However, the success of neural networks is not universal across all domains. Indeed, for learning problems without any special structure, or in cases where the data are somewhat limited, neural networks are known not to perform well with respect to traditional machine-learning methods such as random forests. In this article, we show that a carefully designed neural network with random forest structure can have better generalization ability. In fact, this architecture is more powerful than random forests, because the back-propagation algorithm reduces to a more powerful and generalized way of constructing a decision tree. Furthermore, the approach is efficient to train and requires a small constant factor of the number of training examples. This efficiency allows the training of multiple neural networks to improve the generalization accuracy. Experimental results on real-world benchmark datasets demonstrate the effectiveness of the proposed enhancements for classification and regression.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Machine learning algorithms**;

Additional Key Words and Phrases: Neural network, random forest, classification, regression

ACM Reference format:
Suhang Wang, Charu Aggarwal, and Huan Liu. 2018. Random-Forest-Inspired Neural Networks. *ACM Trans. Intell. Syst. Technol.* 9, 6, Article 69 (October 2018), 25 pages.
https://doi.org/10.1145/3232230

1 INTRODUCTION

Neural networks have become increasingly popular in recent years because of their tremendous success in computer vision [12, 23, 40, 41], speech recognition [11, 14], and natural language processing tasks [7, 25, 36]. In fact, deep-learning methods have regularly won many recent challenges in these domains [14, 43]. This success is, in part, because the special structure of these domains often allows the use of specialized neural network architectures [39] such as convolutional neural networks [14], which take advantage of the aspects like spatial locality in images. Images, speech, and natural language processing are rather specialized data domains in which the attributes

This material is based on work supported by, or in part by, the National Science Foundation (NSF) grants #1614576 and IIS-1217466 and the Office of Naval Research (ONR) grant N00014-16-1-2257. This study is an extension of Reference [38], which appears in the *Proceedings of the 17th SIAM International Conference on Data Mining*.
Authors' addresses: S. Wang, Arizona State University, Tempe, AZ 85281; email: suhang.wang@asu.edu; C. Aggarwal, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598; email: charu@us.ibm.com; H. Liu, Arizona State University, Tempe, AZ 85281; email: huan.liu@asu.edu.

https://doi.org/10.1145/3232230

exhibit very characteristic spatial/temporal behavior, which can be exploited by carefully designed neural network architectures. Such characteristics are certainly not true for all data domains, and in some cases a dataset may be drawn from an application with unknown characteristic behaviors.

In spite of the successes of neural networks in specific domains, this success has not been replicated across all domains. In fact, methods like random forests [4, 16] regularly outperform neural networks in arbitrary domains [8], especially when the underlying data sizes are small and no domain-specific insight has been used to arrange the architecture of the underlying neural network. This is because neural networks are highly prone to overfitting, and the use of a *generic* layered architecture of the computation units (without domain-specific insights) can lead to poor results. The performance of neural networks is often sensitive to the specific architectures used to arrange the computational units. Although the convolutional neural network architecture is known to work well for the image domain, it is hard to expect an analyst to know which neural network architecture to use for a particular domain or for a specific dataset from a poorly studied application domain.

In contrast, methods like decision forests are considered *generalist* methods in which one can take an off-the-shelf package like caret [24] and often outperform [8] even the best of classifiers. A recent study [8] evaluated 179 classifiers from 17 families on the *entire UCI collection of datasets* and concluded that random forests were the best performing classifier among these families, and in most cases, their performance was better than other classifiers in a statistically significant way. In fact, multiple third-party implementations of random forests were tested by this study and virtually all implementations provided better performance than multiple implementations of other classifiers; these results also suggest that the wins by the random forest method were not a result of the specific implementations of the method but are inherent to the merit of the approach. Furthermore, the datasets in the UCI repository are drawn from a vast variety of domains and are not specific to one narrow class of data such as images or speech. This also suggests that the performance of random forests is quite robust irrespective of the data domain at hand.

Random forests and neural networks share important characteristics in common. Both have the ability to model arbitrary decision boundaries, and it can be argued that in this respect, neural networks are somewhat more powerful when a large amount of data is available. However, neural networks are highly prone to overfitting, whereas random forests are extremely robust to overfitting because of their randomized ensemble approach. The overfitting of neural networks is an artifact of the large number of parameters used to construct the model. Methods like convolutional neural networks drastically reduce the number of parameters to be learned by using specific insights about the data domain (e.g., images) at hand. This strongly suggests that the choice of a neural network architecture that drastically reduces the number of parameters with domain-specific insights can help in improving accuracy.

Domain-specific insights are not the only way in which one can engineer the architecture of a neural network to reduce the parameter footprint. In this article, we show that one can use inspiration from successful classification methods like random forests to engineer the architecture of the neural network. Furthermore, starting with this basic architecture, one can improve on the basic random forest model by leveraging the inherent power in the neural network architecture in a carefully controlled way. The reason is that models like random forests are also capable of approximating arbitrary decision boundaries but with less overfitting on smaller datasets.

It is noteworthy that several methods have been proposed to simulate the output of a decision tree (or random forest) algorithm *on a specific dataset, once it has already been constructed* [2, 33]. In other words, such an approach first constructs the decision tree (or random forests) on the dataset up front and then tries to simulate *this specific instantiation* of the random forest with a neural network. Therefore, the constructed random forest is itself an input to the algorithm. Such

an approach defeats the purpose of a neural network in the first place, because it now has to work with the straitjacket of a specific instantiation of the random forest. In other words, it is hard to learn a model, which is much better than the base random forest model, even with modifications.

In this article, we propose a fundamentally different approach to design a basic architecture of the neural network, so that it constructs a model *with similar properties* as a randomized decision tree, although it does not simulate a specific random forest. A different way of looking at this approach is that it constructs a neural network first, which has the property of being interpreted as a randomized decision tree; therefore, the learning process of the neural network is performed directly with back-propagation, and no specific instantiation of a random forest is used as input. However, a mapping exists from an arbitrary random forest to such a neural network, and a mapping back exists as well. Interestingly, such a mapping has also been shown in the case of convolutional neural networks [22, 32], although the resulting random forests have a specialized structure that is suited to the image domain [32]. This article will focus on designing a neural network architecture that has random forest structure such that it has better classification/regression ability and reduced overfitting. The main contributions of the article are listed as follows:

- We propose a novel architecture of decision-tree-like neural networks, which has similar properties as a randomized decision tree, and an ensemble of such neural networks forms the proposed framework called Neural Network with Random Forest Structure (NNRF);
- We design decision-making functions of the neural networks, which results in forward and backward propagation with low time complexity and with reduced possibility of overfitting for smaller datasets; and
- We conduct extensive experiments to demonstrate the effectiveness of the proposed framework for classification and regression.

The remaining of the article are organized as follows. In Section 2, we introduce the random-forest-inspired architecture. In Section 3, we give the detailed design of the proposed framework NNRF for classification followed by training algorithm and time complexity analysis. In Section 4, we extend NNRF for regression. In Section 5, we conduct experiments to demonstrate the effectiveness of NNRF for classification and regression and analyze the parameter sensitivity on NNRF. In Section 6, we briefly review related works. In Section 7, we conclude with future work.

2 A RANDOM-FOREST-INSPIRED ARCHITECTURE

In this section, we introduce the basic architecture of the neural network used for the learning process. Throughout this article, matrices are written as bold capital letters such as $\mathbf{M}$, $\mathbf{W}_{ij}$, and vectors are denoted as bold lowercase letters such as $\mathbf{p}$ and $\mathbf{p}_{ij}$. $\mathbf{M}(i, j)$ denotes the (i, j)th entry of $\mathbf{M}$ while $\mathbf{M}(i, :)$ and $\mathbf{M}(:, j)$ denotes the ith row and jth column, respectively. Similarly, $\mathbf{p}(i)$ denotes the ith elements of $\mathbf{p}$.

In conventional neural networks, the nodes in the input layer are cleanly separated from the hidden layer. However, in this case, we will propose a neural network in which a clear separation does not exist between the nodes of the input and hidden layers. The internal nodes are not completely hidden, because they are allowed to receive inputs from some of the features. Rather, the neural network is designed with a hierarchical architecture, much like a decision tree. Furthermore, just as a random forest contains multiple independent decision trees, our approach will use multiple independent neural networks, each of which has a randomized architecture based on the randomized choice of the inputs in the "hidden" layers. As we will see later, each neural network can be trained extremely efficiently, which is what makes this approach extremely appealing.

The total number of layers in the neural network is denoted by d, which also represents the height of each decision tree in the random forest that the neural network is simulating. Thus, one

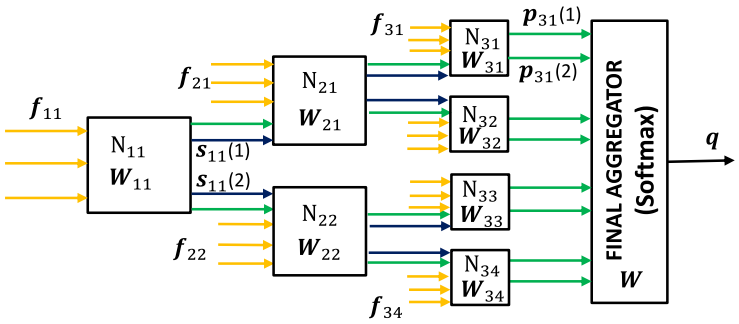


Fig. 1. An illustration of the decision-tree-structured neural network architecture for classification with $d = 3$.

input parameter to the algorithm is the number of layers d of the neural network. The neural network is structured exactly like a binary decision tree, with each node simulating a split and having two outputs out of which exactly one is active. These two outputs feed into a unique node of the next layer of the neural network. Therefore, the number of nodes always doubles from one layer to the next. Thus, the total number of nodes in the neural network is given by $1 + 2^1 + \cdots + 2^{d-1}$, which is equal to $2^d - 1$. In addition, there is a special output node that takes as its input the 2^{d-1} nodes in the final layer and combines them to provide a single prediction of the class label.

Although the number of nodes might seem large, we will see that the required value of d is often quite modest in real settings, because the neural network nodes are able to simulate more powerful splits. Furthermore, because of the tree structure of the neural network, the number of parameters to be learned is quite modest compared to the number of nodes. This is an important factor in avoiding overfitting. Another parameter input to the algorithm is r, which is the number of features that are randomly selected to perform the split at each node. In a traditional random forest, the bag of features to be used for a split at each node is randomly selected up front. Similarly, while setting up the architecture of each neural network, each node in the tree has a bag of features that are randomly selected and *fixed* up front. Thus, the architecture of the neural network is inherently randomized, based on the input features that are selected for each node. As we will see later, this property is particularly useful in an ensemble setting.

The overall architecture of a three-layer neural network for classification is illustrated in Figure 1. For ease of explanation, we name the nodes as N_{ij}, which means the jth node in the ith layer. A parent node N_{ij} is connected to two child nodes $N_{i+1,2^j-1}$ and $N_{i+1,2^j}$. For example, as shown in the figure, N_{22} is the second node in layer 2 and is connected to two child nodes N_{33} and N_{34}. The network contains three types of nodes:

- The nodes in the first layer are input nodes. These nodes only have as input r randomly chosen features from the input data. The node has two outputs, one of which is always 0 (i.e., inactive).
- The nodes in the middle layer are somewhat unconventional from the perspective of most neural networks, in that they are hybrid between the hidden and in the input layer. They receive a single input from an ancestor node (in the treelike neural structure) and also inputs from r randomly chosen features (which were selected up front). Therefore, the node can be viewed as a hybrid node belonging to both the hidden layer and the input layer, since some of its inputs are visible and one input is not. Another important property of this node is that if the hidden input is 0, then all outputs of this node are 0. *This is a crucial property*

to ensure that only one path is activated by a given training instance in the treelike neural network structure.

- The single output node combines the outputs of all the nodes to create a final prediction. However, since only one path in the tree is activated at a given time (i.e., only one of its incoming outputs is nonzero), the output node only uses the one input in practice.

Like a decision tree, only one path is activated in the neural network at a given time. This particularly important, because it means that *the back-propagation algorithm only needs to update the weights of the nodes along this path.* This is a crucial property, because it means that one can efficiently perform the updates for a single path. Therefore, the training phase for a single neural network is expected to work extremely efficiently. However, like any random forest, multiple such "miniature" neural networks are used for prediction.

The overall prediction step uses an ensemble approach like a random forest. Each test instance is predicted with the different neural networks that have been independently trained. These predictions are then averaged to provide the final result.

A number of key properties of this type of neural network can be observed:

- Like a decision tree, only one path in the neural network is activated by a given instance. This makes the back-propagation steps extremely efficient.
- The neural network is potentially more powerful, because the function computed at each internal node can be more powerful than a univariate split.
- The back-propagation algorithm is more powerful than the typical training process of a decision tree, which is very myopic at a given node. The back-propagation effectively adjusts the split criterion all the way from the leaf to the root, which results in a more informed "split" criterion with a deeper understanding of the training data.

3 PROPOSED NNRF FOR CLASSIFICATION

The previous section gave the overall architecture of the neural network; in this section, we give the inner working of the neural network for classification. We will extend the neural network for regression in Section 4. Next, we first introduce the inner working of decision making in each type of node, which guarantees that only one path will be activated. We then introduce how to efficiently perform back-propagation followed by time and space complexity.

3.1 Details of the Proposed Neural Network

Let $\mathbf{f}_{ij} \in \mathbb{R}^{r\times 1}$ be the input features to node N_{ij}, which is fixed up front. Then, given $\mathbf{f}_{11}$ and N_{11}, we can calculate the output $\mathbf{p}_{11}$ as

$$\mathbf{p}_{11} = g(\mathbf{W}_{11}\mathbf{f}_{11} + \mathbf{b}_{11}), \tag{1}$$

where $\mathbf{W}_{11} \in \mathbb{R}^{2\times r}$ and $\mathbf{b}_{11} \in \mathbb{R}^{2\times 1}$ are the weights and the bias of N_{11}. $g(\cdot)$ is the activation function such as *tanh* and Leaky ReLu. Since Leaky ReLu has the advantage of alleviating the gradient-vanishing problem in deep nets and has been proven to outperform tanh [13], in this work, we use Leaky ReLu. Then, $g(x)$ and the derivative of $g(x)$ w.r.t x are given as

$$g(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.2x, & \text{if } x < 0 \end{cases} \qquad g'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0.2, & \text{if } x < 0 \end{cases}. \tag{2}$$

Since only one path will be activated, we need to decide which path to take based on the values of $\mathbf{p}$. Specifically, we define the signal vector $\mathbf{s}_{11} \in \mathbb{R}^{2\times 1}$ as

$$\mathbf{s}_{11}(1) = \mathbb{I}(\mathbf{p}_{11}(1), \mathbf{p}_{11}(2)), \quad \mathbf{s}_{11}(2) = \mathbb{I}(\mathbf{p}_{11}(2), \mathbf{p}_{11}(1)), \tag{3}$$

where $\mathbb{I}(a, b)$ is an indicator function that if $a \geq b$, then $\mathbb{I}(a, b) = 1$; otherwise, $\mathbb{I}(a, b) = 0$. Thus, only one of the elements in $\mathbf{s}_{11}$ will be 1 and $\mathbf{s}_{11}(k) = 1, k = 1, 2$ means that $N_{2,k}$ will be activated. $\mathbf{s}_{11}(1)$ and $\mathbf{p}_{11}(1)$ will go to node N_{21}, i.e., first node in layer 2; and $\mathbf{s}_{11}(2)$ and $\mathbf{s}_{11}(2)$ will go to N_{22}, i.e., second in layer 2, which are shown in Figure 1, i.e., the black arrow with $\mathbf{s}_{11}(k)$ denote the signal $\mathbf{s}_{11}(k)$ and the green line next to it is $\mathbf{p}_{11}(k)$, $k = 1, 2$. Then outputs of N_{21} and N_{22} are calculated as

$$\mathbf{p}_{2j} = \begin{cases} g\left(\mathbf{W}_{2j}\begin{bmatrix}\mathbf{f}_{2j}\\ \mathbf{p}_{11}(j)\end{bmatrix} + \mathbf{b}_{11}\right) & \text{if } \mathbf{s}_{11}(j) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases}, \tag{4}$$

$$\mathbf{s}_{2j} = \begin{cases} \begin{bmatrix}\mathbb{I}(\mathbf{p}_{2j}(1), \mathbf{p}_{2j}(2))\\ \mathbb{I}(\mathbf{p}_{2j}(2), \mathbf{p}_{2j}(1))\end{bmatrix} & \text{if } \mathbf{s}_{11}(j) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases}, \tag{5}$$

where $j = 1, 2$. The idea behind Equation (4) and Equation (5) is that if the input signal $\mathbf{s}_{11}(j), j = 1, 2$ is 0, then the path to node $N_{2,j}$ is inactive. We just simply set the outputs of $N_{2,j}$ as $\mathbf{0} \in \mathbb{R}^{2\times 1}$. However, if $\mathbf{s}_{11}(j)$ is 1, then we use both $\mathbf{p}_{11}(j)$ and the input feature $\mathbf{f}_{2j}$ to calculate $\mathbf{p}_{2j}$ and $\mathbf{s}_{2j}$. This process guarantees that only one path will be activated in next layer. For example, if $\mathbf{s}_{11}(1)$ is 1, then $\mathbf{s}_{22}$ will be $\mathbf{0}$. And $\mathbf{s}_{21}$ will contain only one 1, meaning that only one node will be activated in layer 3. With the same procedure, given $\mathbf{s}_{ij}$ and p_{ij}, the outputs of layer $i + 1$, $i = 2, \ldots, d$ are given as

$$\mathbf{p}_{i+1,t} = \begin{cases} g\left(\mathbf{W}_{i+1,t}\begin{bmatrix}\mathbf{f}_{i+1,t}\\ \mathbf{p}_{ij}(k)\end{bmatrix} + \mathbf{b}_{i+1,t}\right) & \text{if } \mathbf{s}_{ij}(k) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases},$$

$$\mathbf{s}_{i+1,t} = \begin{cases} \begin{bmatrix}\mathbb{I}(\mathbf{p}_{i+1,t}(1), \mathbf{p}_{i+1,t}(2))\\ \mathbb{I}(\mathbf{p}_{i+1,t}(2), \mathbf{p}_{i+1,t}(1))\end{bmatrix} & \text{if } \mathbf{s}_{ij}(k) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases}, \tag{6}$$

where $t = 2(j - 1) + k$ and $k = 1, 2$. It is easy to verify that $N_{i+1,t}$ is connected by $\mathbf{p}_{ij}(k)$. $\mathbf{W}_{i+1,t} \in \mathbb{R}^{2\times(r+1)}$ is the weights of node $N_{i+1,t}$, and $\mathbf{b}_{i+1,t} \in \mathbb{R}^{2\times 1}$ is the corresponding bias. Equation (6) shows that if $\mathbf{s}_{ij} = \mathbf{0}$, then none of its children are activated. Let d be the depth of the neural network. Then the outputs $\mathbf{p}_{d,1}, \ldots, \mathbf{p}_{d,2^{d-1}}$ are used as input to the final aggregator. The final aggregator first aggregate all the inputs as one vector. For simplicity, we use $\mathbf{p} = [\mathbf{p}_{d,1}^T, \ldots, \mathbf{p}_{d,2^{d-1}}^T]^T \in \mathbb{R}^{2^d\times 1}$ to denote the aggregated vector. Then, a softmax function is applied,

$$\mathbf{q}(c) = \frac{\exp(\mathbf{W}(c, :)\mathbf{p} + \mathbf{b}(c))}{\sum_{k=1}^{C}\exp(\mathbf{W}(k, :)\mathbf{p} + \mathbf{b}(k))}, \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{C\times 2^d}$ is the weights of the softmax, $\mathbf{b} \in \mathbb{R}^{C\times 1}$ is the bias terms, and C is number of classes. $\mathbf{q}$ gives the probability distribution that the input data belongs to the C classes. For example, $\mathbf{q}(c)$ means the probability that the input data sample belongs to class c. With the estimated distribution, the cost function is defined using cross-entropy between $\mathbf{q}$ and the ground-truth distribution as follows:

$$L(\mathbf{y}, \mathbf{q}) = -\sum_{c=1}^{C}\mathbf{y}(c)\log\mathbf{q}(c), \tag{8}$$

where $\mathbf{y} \in \mathbb{R}^{C\times 1}$ is the one-hot coding of ground-truth label, i.e., $\mathbf{y}(k)$ is 1 if the label is k; otherwise, it is 0. By minimizing the cross-entropy, we want the estimated distribution $\mathbf{q}$ to be as close

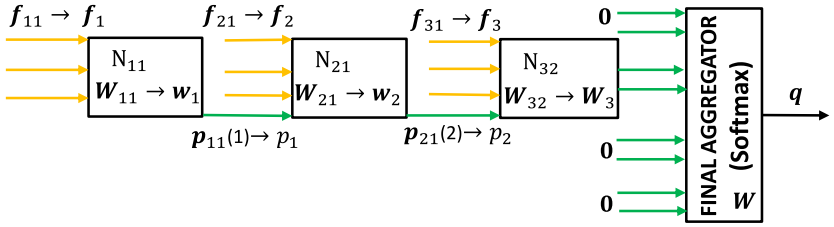


Fig. 2. An example of reduced neural network.

as possible to the ground-truth distribution $\mathbf{y}$ and thus we can train a good neural network for prediction. Given training data matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ and one-hot coding label matrix $\mathbf{Y} \in \mathbb{R}^{n \times C}$, where n is number of data samples and m is number of features, the objective function is written as

$$\min_{\theta} \frac{1}{n} \sum_{l=1}^{n} (L(\mathbf{Y}(l,:), \mathcal{T}(\mathbf{X}(l,:)) + \alpha \mathcal{R}_l(\theta)), \tag{9}$$

where $\theta = \{\mathbf{W}, \mathbf{b}, \mathbf{W}_{ij}, \mathbf{b}_{ij}\}_{i=1,\ldots,n, j=1,\ldots,2^{i-1}}$ is the set of parameters to be learned in the tree-structured neural network $\mathcal{T}$ and $\mathcal{T}(\mathbf{X}(l,:))$ is the estimated class distribution of the input data $\mathbf{X}(l,:)$. $\mathcal{R}_l(\theta)$ is a regularizer associated with $\mathbf{X}(l,:)$ to avoid over-fitting, which is defined as

$$\mathcal{R}_l(\theta) = \|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2 + \sum_{i=1}^{d} \sum_{j=1}^{2^{i-1}} \sum_{k=1}^{2} \mathbf{s}_{ij}(k) \left(\|\mathbf{W}_{ij}(k,:)\|_2^2 + \|\mathbf{b}_{ij}(k)\|^2 \right). \tag{10}$$

The reason that we design the regularizer to be instance specific is that for every forward computation, each instance $\mathbf{X}(l,:)$ will result in one active path, and only those parameters in the active path will be updated using back-propagation (more details in Section 3.2). Therefore, an instance specific regularizer is more appropriate. α is a scalar to control the contribution of the regularizer.

3.2 Efficient Backpropagation

We use back-propagation to train the tree-structured neural network. Let us focus on the cost function in Equation (8) to show how to perform efficient back-propagation as extension to Equation (9) is simple. For simplicity of notation, let us denote $-\sum_{c=1}^{C} \mathbf{y}(c) \log \mathbf{q}(c)$ as $\mathcal{J}$.

Consider that if we take derivative of $\mathcal{J}$ w.r.t to $\mathbf{W}_{ij}$. The term that involves $\mathbf{W}_{ij}$ is through $\mathbf{p}_{ij}$. However, if N_{ij} is inactive, i.e., if the input signal to N_{ij} is 0, then $\mathbf{p}_{ij}$ is set to $\mathbf{0}$ and is independent with $\mathbf{W}_{ij}$. In this case, the derivative of $\mathcal{J}$ w.r.t $\mathbf{W}_{ij}$ is $\mathbf{0}$ and there is no need to update $\mathbf{W}_{ij}$. Thus, we can safely remove inactive nodes from the neural network, which reduces the tree-structured neural network to a small neural network. Figure 2 gives an example of the reduced network of the full network in Figure 1 assuming that the active path is $N_{11} \to N_{21} \to N_{32}$. For simplicity of explanation, we rewrite the weights and bias in the ith layer of the simplified neural network as $\mathbf{w}_i$ and b_i, the features of each node as $\mathbf{f}_i$, and the link connecting the $i-1$th layer to the ith layer as p_i. For example, as shown in Figure 2, $\mathbf{w}_1 = \mathbf{W}_{11}(1,:)$, $\mathbf{w}_2 = \mathbf{W}_{21}(2)$, and $\mathbf{W}_3 = \mathbf{W}_{32}$, $\mathbf{f}_1 = \mathbf{f}_{11}$, $\mathbf{f}_2 = \mathbf{f}_{21}$, $\mathbf{f}_3 = \mathbf{f}_{32}$, $p_1 = \mathbf{p}_{11}(1)$, and $p_2 = \mathbf{p}_{21}(2)$. Then performing back-propagation on the simplified network is very efficient as it only involves a small number of parameters and the network is narrow. The details of the derivative are given as follows. Let $\mathbf{u}(c) = \mathbf{W}(c,:)\mathbf{p} + \mathbf{b}(c)$, we define the "error" $\delta(c)$ as

$$\delta(c) = \frac{\partial \mathcal{J}}{\partial \mathbf{u}(c)} = \mathbf{y}(c)\mathbf{q}(c) - \mathbf{y}(c), \quad c = 1, \ldots, C. \tag{11}$$

Then, we have

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}(c,:)} = \delta_c \mathbf{p}^T, \ \frac{\partial \mathcal{J}}{\partial \mathbf{b}(c)} = \delta_c. \tag{12}$$

Similarly, let $\mathbf{u}_d = \mathbf{W}_d \begin{bmatrix} \mathbf{f}_d \\ \mathbf{p}_{d-1} \end{bmatrix} + \mathbf{b}_d$, then for $k = 1, 2$,

$$\begin{aligned} \delta_d(k) = \frac{\partial \mathcal{J}}{\partial \mathbf{u}_d(k)} &= \sum_c \delta(c) \mathbf{W}(c, ind(\mathbf{p}_d(k))) g'(\mathbf{u}_d(k)) \\ \frac{\partial \mathcal{J}}{\partial \mathbf{W}_d(k)} = \delta_d(k)[\mathbf{f}_d^T, \ \mathbf{p}_{d-1}], &\quad \frac{\partial \mathcal{J}}{\partial \mathbf{b}_d(k)} = \delta_d(k), \end{aligned} \tag{13}$$

where $ind(\mathbf{p}_d(k))$ is the index of the element in $\mathbf{W}(c,:)$ that multiplied with $\mathbf{p}_d(k)$. With the same idea, for $i = d-1, \ldots, 1$, let $u_i = \mathbf{w}_i \begin{bmatrix} \mathbf{f}_i \\ \mathbf{p}_{i-1} \end{bmatrix} + b_i$, then we can get

$$\begin{aligned} \delta_i = \frac{\partial \mathcal{J}}{\partial u_i} &= \begin{cases} \sum_{k=1}^{2} \delta_d(k) \mathbf{W}_d(k, r+1) g'(u_i), & i = d-1 \\ \delta_{i+1} \mathbf{w}_{i+1} (r+1) g'(u_i), & i = d-2, \ldots, 1 \end{cases} \\ \frac{\partial \mathcal{J}}{\partial \mathbf{w}_i} &= \begin{cases} \delta_i [\mathbf{f}_i^T, \ p_{i-1}], & i = d-1, \ldots, 2 \\ \delta_1 \mathbf{f}_1^T, & i = 1 \end{cases} \\ \frac{\partial \mathcal{J}}{\partial b_i} &= \delta_i, \quad i = d-1, \ldots, 1. \end{aligned} \tag{14}$$

3.3 Training Algorithm of NNRF for Classification

The algorithm to train a d-layer tree-structured neural network for classification is shown in Algorithm 1. We first draw a bootstrap sample in Line 1. From Line 2 to Line 5, we draw the input feature for each node of the neural network. From Line 8 to Line 9, we update the parameters using back-propagation.

Following the same idea as random forest, we aggregate N independently trained neural networks for classification, which we name as NNRF. The algorithm of NNRF is shown in Algorithm 2. For an input $\mathbf{x}$, the predicted class distribution is by aggregating the predicted class distribution from the N neural networks,

$$\mathbf{q}_a = \frac{1}{N} \sum_{i=1}^{N} \mathcal{T}_i(\mathbf{x}). \tag{15}$$

Then the label is predicted as the class with the highest probability, i.e, $y = \arg\max_c \mathbf{q}_a$.

3.4 Time Complexity

Since there are only one active path for each input data sample, the cost of performing forward propagation up to depth d using Equation (6) is $O(rd)$. Since $\mathbf{p}$ only has two nonnegative elements, the cost of softmax in Equation (7) to get $\mathbf{q}$ is $O(C)$. Therefore, the cost of forward propagation is $O(rd + C)$ for each data sample in each tree. Similarly, the cost of backward propagation using Equation (13) and Equation (14) is also $O(C + rd)$. Thus, the total cost of training N tree-structured neural networks with depth d is $O(t(C + rd)nN)$, where t is number of epochs for training each neural network. Following the common way of setting r in random forest, r is usually chosen as $\sqrt{m}$ [10], where m is the number of features. Effects of different choices of r can be found in Section 5.5.2. Thus, the total cost is approximately $O(t(C + \sqrt{m}d)nN)$, which is modest, and thus the training is efficient.

ALGORITHM 1: Decision-Tree-Structured Neural Network

Require: $\mathbf{X} \in \mathbb{R}^{n\times m}, \mathbf{y} \in \mathbb{R}^{n\times 1}, d, r, \alpha$
Ensure: d-layer tree-structured neural network
1: Draw a bootstrap sample $\mathbf{X}^*$ of size N from $\mathbf{X}$
2: **for** i = 1:d **do**
3: **for** j = $1{:}2^{i-1}$ **do**
4: Construct $\mathbf{f}_{ij}$ by selecting r features at random from the m features of $\mathbf{X}^*$
5: **end for**
6: **end for**
7: **repeat**
8: Forward propagation to get cost
9: Backward propagation to update parameters
10: **until** convergence
11: **return** Tree Structured Neural Network

ALGORITHM 2: NNRF

Require: $\mathbf{X} \in \mathbb{R}^{n\times m}, \mathbf{y} \in \mathbb{R}^{n\times 1}, d, N, r, \alpha$
Ensure: N d-layer tree-structured neural networks
1: **for** b = 1:N **do**
2: Construct and Learn Tree Structured Neural Network $\mathcal{T}_i$ with Algorithm 1
3: **end for**
4: **return** $\{\mathcal{T}_i\}_{i=1}^{N}$

3.5 Number of Parameters

The main parameters in a decision-tree-like neural network are $\mathbf{W}_{ij} \in \mathbb{R}^{2\times(r+1)}$ and $\mathbf{b}_i \in \mathbb{R}^{2\times 1}$, $i = 2, \ldots, d, j = 1, \ldots, 2^{i-1}$, $\mathbf{W}_1 \in \mathbb{R}^{2\times r}$, and $\mathbf{W} \in \mathbb{R}^{C\times 2^d}$. Thus, the number of parameters is approximately $O(2^d(2r + C))$. We usually set d as $\lfloor \log_2 C \rfloor + 1$, because this ensures that 2^d is no less than C. In other words, $\mathbf{p}$ has dimension of at least C, which is large enough for classification with C classes. Considering the fact that d is usually set as $\lfloor \log_2 C \rfloor + 1$ and r is usually chosen as $\sqrt{m}$, the space complexity is approximately $O((\sqrt{m} + C) \cdot 2^C)$, which is modest and thus can be well trained even when the data size is small.

4 PROPOSED NNRF FOR REGRESSION

In this section, we extend NNRF for regression. We will first introduce how to extend the decision-tree-structured neural network for regression and ensemble of such neural networks result in the proposed NNRF for regression.

4.1 Tree-Structured Neural Network for Regression

Figure 3 gives an illustration of decision-tree-structured neural network for regression. It has the same structure as tree-structured neural network for classification as shown in Figure 1. The feature sampling and decision-making functions used in classification are also adopted for regression. The only *differences are the last layer and the loss function*. In tree-structured neural network for classification, we aggregate all the features to form $\mathbf{p}$, which is used as input to a softmax function to generate empirical class distributions for predicting the class label. For regression, instead of using the softmax to predict the class label, we use linear regression to predict the numerical target

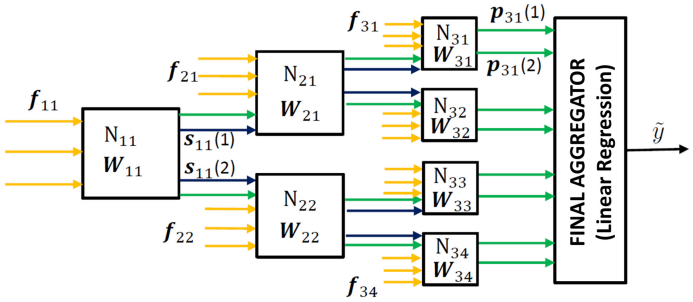


Fig. 3. An Illustration of the decision-tree-structured neural network architecture for regression with $d = 3$.

value $\tilde{y}$ as

$$\tilde{y} = \mathbf{w}^T \mathbf{p} + b, \tag{16}$$

where $\mathbf{w} \in \mathbb{R}^{2^{d_1} \times 1}$ is the weights and b is the bias term. The loss function is defined as the Euclidean distance of the predicted value from the ground-truth value

$$Z(y, \tilde{y}) = (y - \tilde{y})^2, \tag{17}$$

where y is the target value. By minimizing the Euclidean distance, we want the predicted value $\tilde{y}$ to be as close as possible to the ground-truth target value y. Given the training data $\mathbf{X} \in \mathbb{R}^{n \times m}$ and the target value vector $\mathbf{y} \in \mathbb{R}^{n \times 1}$, where n is number of data samples, the objective function for regression is written as

$$\min_{\theta} \frac{1}{n} \sum_{l=1}^{n} (Z(\mathbf{y}(l), \mathcal{T}(\mathbf{X}(l, :))) + \alpha \mathcal{R}_l(\theta)), \tag{18}$$

where $\theta = \{\mathbf{w}, b, \mathbf{W}_{ij}, \mathbf{b}_{ij}\}_{i=1,\ldots,n, j=1,\ldots,2^{i-1}}$ is the set of parameters to be learned in the tree-structured neural network $\mathcal{T}$, and $\mathcal{T}(\mathbf{X}(l, :))$ is the estimated value of the input data $\mathbf{X}(l, :)$. α is a scalar. $\mathcal{R}_l(\theta)$ is the regularizer to avoid over-fitting, which is defined as

$$\mathcal{R}_l(\theta) = \|\mathbf{w}\|_2^2 + b^2 + \sum_{i=1}^{d} \sum_{j=1}^{2^{i-1}} \sum_{k=1}^{2} \mathbf{s}_{ij}(k) \left(\|\mathbf{W}_{ij}(k, :)\|_2^2 + \|\mathbf{b}_{ij}(k)\|^2 \right). \tag{19}$$

Similarly, we use back-propagation to train the decision-tree-structured neural network for regression. As decision-tree-structured neural network for regression and that for classification have similar structure except the last layer and the loss function, the derivation of back-propagation for regression is also very similar to that for classification; the detail of the latter is given in Section 3.2. We thus omit the detailed derivation for regression here. The algorithm to train a d-layer tree-structured neural network for regression is the same as that for training a tree-structured neural network for classification, which is given in Algorithm 1.

4.2 Algorithm of NNRF for Regression

We use Algorithm 2 to train N tree-structured neural networks for regression. Following the same idea as random forest for regression, we aggregate the predicted values of the N independently trained neural networks. Specifically, for an input $\mathbf{x}$, the predicted numeric value is by aggregating the predicted value from the N neural networks as

$$\bar{y} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{T}_i(\mathbf{x}). \tag{20}$$

As there is only one path activated for forward and backward propagation, the time complexity of NNRF for regression is $O(trdnN)$, where t is number of epochs for training each neural network. The number of parameters of NNRF for regression is $O(2^{d+1}r)$, where d is usually chosen within 3 and 6.

5 EXPERIMENTAL RESULTS

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework NNRF for classification and regression. We begin by introducing datasets and the experimental setting, and then we compare NNRF with state-of-the-art classifiers to demonstrate the effectiveness of NNRF for classification. We also compare NNRF with classical and representative regression methods to show the effectiveness of NNRF for regression. Further experiments are conducted to investigate the effects of the hyper-parameters on NNRF for classification and regression, respectively.

5.1 Datasets and Experimental Settings

The classification experiments are conducted on 13 publicly available benchmark datasets, which includes 8 UCI[1] datasets, i.e., forest type mapping (ForestType), speech record of 26 alphabets (Isolet), sonar signals of mines vs. rocks (Sonar), chemical analysis of wines (Wine), Wisconsin diagnostic breast cancer (wdbc), Vehicle Silhouettes (Vehicle), hill valley (Valley) and sensorless drive diagnosis (DriveDiagnosis), three image datasets, i.e., images of 20 objects (COIL-20)[2], images of handwritten digits (USPS),[3] images of faces (MSRA),[4] one bioinfomatics dataset (Bioinformatics) [19], and one hand movement dataset (Movement)[5]. We include datasets of different domains and different format so as to give a comprehensive understanding of how NNRF performs with datasets of various domains and format. The statistics of the datasets used in the experiments are summarized in Table 1. From the table, we can see that these datasets are small datasets with different number of classes. Deep-learning algorithms with large amounts of parameters may not work well on these datasets.

The regression experiments are conducted on seven publicly available benchmark datasets, which includes five UCI[6] datasets, i.e., community crime (Crime), Acetone, blog feedback (Blog), Wisconsin prognostic breast cancer (Wpbc) and relative location of CT slices (CTSlices), one cpu activity dataset (CompActiv),[7] and one structure activity dataset (Triazines).[8] Crime is to predict the community crime ratio. Acetone is a subset of the UCI Gas Sensor Array Drift Dataset. The UCI Gas Sensor Array Drift Dataset is to predict concentration level of six chemical compounds at which the sensors were exposed. We select the subset that predict concentration level of Acetone. Blog is to predict the number of comments the blog will get in the next 24 hours. Wpbc is to predict the time it takes to recur. CTSlices is to estimate the relative location of the CT slice on the axial axis of the human body. ComActiv is to predict cpu activity level from system performance measurements. And Triazines is to predict the inhibition of dihydrofolate reductase by pyrimidines. The statistics of the datasets used in the experiments for regression are summarized in Table 2. The last column of the Table 2 gives the range of the target value, i.e., the minimal and

[1] All eight UCI datasets for classification are available at https://archive.ics.uci.edu/ml/datasets.html.
[2] http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php.
[3] http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/multiclass.html#usps.
[4] http://www.escience.cn/system/file?fileId=82035.
[5] http://sci2s.ugr.es/keel/dataset.php?cod=165#sub2.
[6] All five UCI datasets for regression are available at https://archive.ics.uci.edu/ml/datasets.html.
[7] http://www.cs.toronto.edu/delve/data/datasets.html.
[8] https://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/regression.html.

Table 1. Statistics of the Datasets for Classification

Dataset	Number of Samples	Number of Feature	Number of Class
COIL20	1,440	1,024	20
ForestType	523	26	4
Isolet	7,797	617	26
Bioinformatics	391	20	3
Sonar	208	60	2
USPS	9,298	256	10
Wine	178	13	3
Movement	360	90	15
MSRA	1,799	256	12
wdbc	569	30	2
Vehicle	946	18	4
Valley	1,212	100	2
DriveDiagnosis	58,509	48	11

Table 2. Statistics of the Datasets for Regression

Dataset	Number of Samples	Number of Feature	Target Range
Crime	1,994	127	[0.0, 1.0]
Acetone	1,558	128	[12.0, 500.0]
wpbc	194	32	[1.0, 125.0]
Blog	3,000	281	[0.0, 796.0]
Triazines	186	60	[0.1, 0.9]
CompActiv	8,192	26	[0.0, 99.0]
CTSlices	53,500	385	[1.74, 97.49]

maximal value for the target to be predicted. This is included to give a sense about the scale of RMSE and MAE for each dataset. Similarly, from the table, we can see that these datasets are small datasets from different domains, and deep-learning algorithms usually cannot be well trained for such small datasets.

To evaluate the classification ability of the proposed framework NNRF, two widely used classification evaluation metrics, i.e., Micro-F1 and Macro-F1, are adopted. The larger the Micro-F1 and Macro-F1 scores are, the better the classifier is.

To evaluate the regression performance of the proposed framework NNRF, we use root mean square (RMSE) and mean absolute value (MAE), which are classical evaluation metrics for measuring the quality of regression. The smaller RMSE and MAE are, the closer the predicted numerical value is to the ground-truth value.

5.2 Classification Performance Comparison

We compare the proposed framework NNRF with other classical and state-of-the-art classifiers to evaluate the classification ability of NNRF. The details of these classifiers are listed as follows:

- LR: Logistic regression [18], also known as maximum entropy classifier, is a popular generalized linear regression model used for classification. We use the implementation by scikit-learn [28].

- SVM: Support vector machine [5] is a classical and popular classifier that tries to find the best hyperplane that represents the largest margin between classes. We use the well-known libsvm [5] with rbf kernel for classification.
- NN: This is a one hidden-layer feed forward neural network [3] with softmax as the output layer and cross-entropy as the classifier. We use the implementation of Keras [6], which is a popular deep-learning and neural networks toolbox.
- DBN: Deep belief network [15] is a popular deep generative neural network that is composed of multiple hidden layers with the top layer as restricted Boltzmann machine. It's able to extract hierarchical features. Following the common way of using DBN for classification, a softmax classifier is added on top of DBN and the weights of DBN are fine-tuned together with the softmax classifier.
- DNN: This is a four-layer deep neural network. We use ReLu as activation function, as it can alleviate the gradient vanishing problems in deep neural networks [13]. We adopt ADAM [21] for stochastic optimization. The number of hidden nodes in each layer are tuned via cross validation.
- RF: Random forest [4] is a classical and popular ensemble-based method that aggregate independent decision trees for classification. It is one of the most powerful classifiers [8]. We use the implementation by scikit-learn.
- gcForest: gcForest [45] stacks a set of RFs as stacking hidden layers in NNs. The outputs of several RFs in one layer is concatenated with the original features and then are used as the input to the next level RFs. It is a proposed as an alternative of deep nets for classification. We use the implementation from the author.[9]

For each classifier, there are some parameters to be set. In the experiment, we use 10-fold cross validation on the training data to set the parameters. Note that no test data are involved in the parameter tuning. Specifically, for NNRF, we empirically set $r = \lceil \sqrt{m} \rceil$, $d = \lfloor \log_2 C \rfloor + 1$, $N = 150$, and $\alpha = 0.00005$. The sensitivity of parameters r, d, and N on the classification performance of NNRF will be analyzed in detail in Section 5.5. The experiments are conducted using 5-fold cross validation and the average performance with standard deviation in terms of Macro-F1 and Micro-F1 are reported in Tables 3 and 4, respectively. From the two tables, we make the following observations:

- Generally, the seven compared classifiers, i.e., LR, SVM, NN, DBN, DNN, RF, and gcForest have similar performances in terms of Macro-F1 and Micro-F1 on most of the datasets used. They all performs well on most of the datasets used. Observation in Reference [8] that generally RF achieves the best performance on the majority of the datasets among 179 classifiers.
- The performance of NN and DNN are very close. On datasets such as ForestType, Isolet, Sonar, and Movement, NN is slightly better than DNN, which suggests that going deeper does not necessary gives better results on small datasets.
- NNRF outperforms the compared classifiers on the majority of the datasets. Although NNRF also has a tree structure like RF, it exploits a more powerful neural network in each node to make decisions, i.e., linear combination of features followed by a non-linear function. Furthermore, the back-propagation algorithm receives feedback from the leaf nodes, and therefore, it inherently constructs the "splits" at the internal nodes with a deeper understanding of the training data. Thus, it is able to make better decisions than RF and gives better performance.

[9] https://github.com/kingfengji/gcForest.

Table 3. Classification Results(Macro-F1%±std) of Different Classifiers on Different Datasets

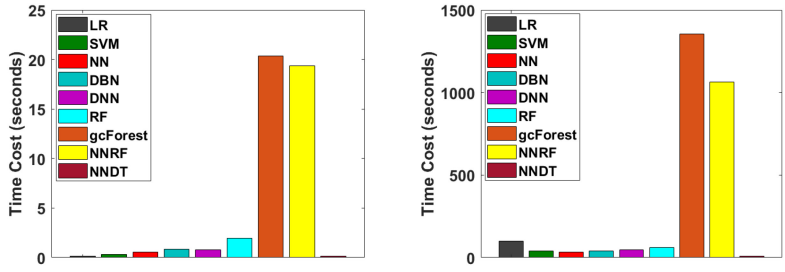
Dataset	LR	SVM	NN	DBN	DNN	RF	gcForest	**NNRF**
COIL20	96.07±1.09	97.97±1.25	98.11±1.07	99.01±0.42	99.39±0.32	**99.93**±0.13	99.66±0.16	**99.93**±0.12
ForestType	88.24±3.11	88.81±2.07	89.51±2.49	89.05±3.12	88.81±3.91	90.95±1.89	87.36±2.79	**91.56**±2.01
Isolet	95.49±0.51	95.68±0.48	95.65±0.72	95.23±0.52	95.37±0.50	94.34±0.39	95.38±0.56	**95.72**±0.51
Bioinfomatics	78.89±6.74	81.32±5.47	73.57±6.94	74.63±4.98	76.30±4.45	70.73±4.47	65.63±4.97	**82.98**±4.94
Sonar	75.96±7.04	80.78±3.24	82.74±4.51	80.35±4.67	82.21±5.25	84.92±6.00	**90.65**±5.42	86.12±4.98
USPS	93.45±0.37	94.86±0.27	94.21±0.43	93.78±0.39	94.90±0.20	95.10±0.34	95.52±0.31	**95.60**±0.36
Wine	59.61±3.96	70.82±3.67	68.11±5.62	67.88±4.99	68.32±4.83	66.87±2.44	70.27±3.69	**71.10**±3.19
Movement	68.94±2.91	76.41±4.16	82.10±3.81	80.64±4.08	81.54±3.96	82.12±2.74	80.68±3.12	**83.02**±2.96
MSRA	99.71±0.23	**100**±0.00	**100**±0.00	99.92±0.05	99.95±0.10	99.47±0.19	**100**±0.00	**100**±0.00
Wdbc	97.68±1.23	97.14±1.81	98.30±1.52	97.64±1.98	97.95±1.36	98.67±2.03	98.12±0.82	**99.9**±0.09
Vehicle	80.27±3.12	76.99±2.78	80.54±2.66	81.01±3.87	81.57±3.60	73.92±2.94	81.39±3.18	**85.12**±2.87
Valley	62.59±2.51	59.24±1.45	61.23±2.25	62.89±2.16	61.43±2.54	64.23±1.94	68.35±2.32	**68.90**±1.98
DriveDiagnosis	75.22±0.54	97.00±0.05	99.07±0.10	99.09±0.14	99.13±0.15	99.65±0.04	99.67±0.04	**99.71**±0.04

Table 4. Classification Results(Micro-F1%±std) of Different Classifiers on Different Datasets

Dataset	LR	SVM	NN	DBN	DNN	RF	gcForest	**NNRF**
COIL20	96.13±1.04	97.95±1.23	98.18±1.10	99.00±0.41	99.39±0.32	99.93±0.13	99.66±0.16	**99.94**±0.14
ForestType	89.62±2.15	89.62±1.94	90.57±1.83	89.99±2.98	89.81±3.39	91.51±1.85	88.68±2.68	**92.45**±1.90
Isolet	95.51±0.51	95.70±0.49	95.66±0.56	95.22±0.61	95.38±0.50	94.35±0.39	95.39±0.56	**95.79**±0.43
Bioinfomatics	82.50±4.11	85.01±2.92	80.09±4.13	80.79±3.27	82.25±2.42	78.75±3.15	77.50±3.39	**86.25**±3.02
Sonar	75.96±7.04	80.78±3.24	82.74±4.51	80.32±4.59	82.32±4.30	84.92±6.00	**90.69**±5.43	86.12±4.98
USPS	94.11±0.33	95.17±0.25	94.89±0.31	93.89±0.40	95.10±0.19	95.62±0.32	96.06±0.30	**96.01**±0.33
Wine	70.35±1.32	70.81±3.15	69.19±5.29	67.92±4.67	69.38±3.67	69.19±1.32	71.12±2.87	**71.38**±2.78
Movement	70.13±1.81	77.33±3.92	82.67±3.61	80.78±3.74	82.13±3.22	82.68±2.43	81.33±2.96	**84.01**±2.68
MSRA	99.73±0.21	**100**±0.00	**100**±0.00	99.93±0.05	99.94±0.10	99.45±0.22	**100**±0.00	**100**±0.00
Wdbc	97.68±1.23	97.14±1.81	98.30±1.52	97.64±1.98	98.09±1.28	98.67±2.03	98.26±0.80	**99.9**±0.09
Vehicle	80.35±3.22	77.78±2.78	80.63±2.85	81.07±4.02	81.52±3.44	73.92±3.00	81.28±3.07	**85.38**±2.91
Valley	66.53±1.97	64.21±0.72	62.79±2.11	64.06±1.97	63.27±2.16	64.31±1.94	70.61±2.08	**71.07**±1.82
DriveDiagnosis	75.40±0.66	97.00±0.06	99.08±0.09	99.09±0.14	99.14±0.14	99.65±0.04	99.67±0.04	**99.71**±0.04

In summary, the proposed framework can achieve better classification result by combining the power from random forest structures and power from activation functions of neural networks.

Running time comparison: We report the running time of NNRF and the compared methods. Since the running time of NNRF and RF are dependent on the number of trees, for fair comparison, we choose the number of trees of both NNRF and RF to be 150. *It is noteworthy that trees of NNRF can be parallelly trained.* Thus, the running time of each decision-tree-structured neural network (NNDT) of NNRF are more important. Therefore, we also report the average running time of NNDT. The results on Valley and Isolet are shown in Figure 4. From the figure, we can observe: Though the running time of NNRF is larger DNN, it is noteworthy that *the average running time of decision-tree-structured neural network in NNRF is much smaller than DNN.* Thus, if we run these 150 trees in parallel on a machine with many CPU cores, *the running time of NNRF can be significantly reduced.* In addition, NNRF belongs to neural networks, which suggests that the training time can be further reduced if we run NNRF on GPUs.



(a) Valley (b) Isolet

Fig. 4. Running time comparison for classification on Valley and Isolet.

5.3 Regression Performance Comparison

To evaluate the regression ability of NNRF, we compare the proposed framework with other classical and state-of-the-art regression models. The details of these regression methods are listed as follows:

- RR: Ridge regression [17] is a linear regression model whose loss function is the linear least squares function, and the regularization is given by the l2-norm. It is a popular method used for regression. We use the implementation by scikit-learn.
- SVR: Support vector regression [34] is an extension of SVM proposed for regression. We use the implementation of libsvm.
- NN: This is a one hidden-layer feedforward neural network [3] with linear regression as the output layer and Euclidean distance as loss function.
- DBN: This is a three layer deep belief network. We add a linear regression on top of DBN for regression during fine-tuning.
- DNN: This is a four layer deep neural network. Instead of using softmax, we use a linear regressor for regression task.
- RF: Random forest [4] is a classical and popular ensemble-based method that aggregate independent decision trees for regression, where the loss function is the Euclidean distance between the predicted values and the target value. We use the implementation by scikit-learn.
- NRF: Neural random forest [2] first constructs random forests on the dataset up front and then try to simulate this specific instantiation of the RF with a neural network. It is proposed for regression task. We use the implementation form the authors.[10]

For each regressor, there are some parameters to be set. In the experiment, we adopt 10-fold cross validation on the training data to tune the parameters. Note that no test data are involved in the parameter tuning. Specifically, for NNRF, we empirically set $r = \lceil \sqrt{m} \rceil$, $d = 3$, $N = 100$, and $\alpha = 0.00005$. The sensitivity of parameters r, d, and N on the regression performance of NNRF will be analyzed in detail in Section 5.5. The experiments are conducted using 5-fold cross validation and the average performance with standard deviation in terms of RMSE and MAE are reported in Tables 5 and 6, respectively. Note that for different datasets, RMSE and MAE have different scale, which is because the scale of the target values for different datasets are different. From the two tables, we make the following observations:

[10] https://github.com/JohannesMaxWel/neural_random_forests.

Table 5. Regression Results (RMSE±std) of Different Regression Methods on Different Datasets

Dataset	RR	SVR	NN	DBN	DNN	RF	NRF	**NNRF**
Crime	0.143±0.014	0.182±0.013	0.145±0.009	0.159±0.011	0.162±0.010	0.139±0.011	0.138±0.013	**0.132**±0.011
Acetone	17.41±1.69	12.24±3.73	15.96±1.85	13.66±2.57	11.78±1.16	11.54±1.99	12.04±2.18	**10.91**±2.02
wpbc	34.34±2.11	33.67±2.96	34.36 ±2.39	35.28±3.08	37.67±4.17	32.92±3.05	33.00±3.14	**31.52**±3.18
Blog	21.70±1.42	23.51±1.77	22.28±3.76	23.49±3.34	23.83±2.03	21.08±3.15	20.09±3.87	**20.51**±3.68
Triazines	0.125±0.020	0.122±0.027	0.134±0.020	0.128±0.022	0.150±0.009	0.114±0.019	0.119±0.023	**0.108**±0.020
CompActiv	9.617±0.415	9.377±0.445	3.475±0.135	3.925±0.146	2.820±0.114	2.508±0.091	2.557±0.109	**2.448**±0.087
CTSlices	9.254±0.162	8.745±0.139	1.734±0.087	1.364±0.079	1.194±0.057	1.258±0.049	1.316±0.059	**1.186**±0.055

Table 6. Regression Results (MAE±std) of Different Classifiers on Different Datasets

Dataset	RR	SVR	NN	DBN	DNN	RF	NRF	**NNRF**
Crime	0.100±0.007	0.135±0.007	0.103±0.004	0.109±0.006	0.111±0.005	0.096±0.004	0.095±0.004	**0.093**±0.005
Acetone	11.13±0.49	4.037±0.591	8.878±0.743	6.826±0.775	5.633±0.559	3.580±0.434	3.932±0.429	**3.275**±0.442
wpbc	28.27±1.44	27.88±1.89	28.98±2.02	29.18±0.775	30.62±2.57	29.21±2.14	27.10±2.28	**25.03**±2.17
Blog	7.362±0.353	5.193±0.403	11.84±0.73	10.23±0.82	9.188±0.856	5.150±0.53	5.985±0.786	**5.120**±0.524
Trianzines	0.096±0.012	0.091±0.017	0.102±0.013	0.099±0.017	0.108±0.007	0.080±0.007	0.087±0.009	**0.080**±0.008
CompActiv	5.933±0.044	3.430±0.125	2.329±0.041	2.496±0.038	1.931±0.093	1.769±0.031	1.897±0.097	**1.674**±0.030
CTSlices	6.650±0.136	4.973±0.107	1.241±0.072	0.974±0.066	0.830±0.051	**0.796**±0.042	0.896±0.048	0.829±0.050

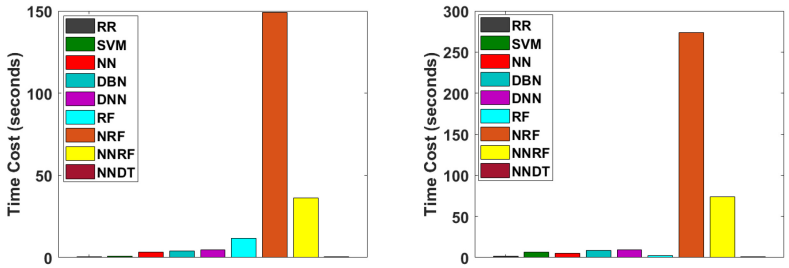
- Compared with RR, SVR, NN, DBN, and DNN, random forest generally has slightly better performance. This observation shows the effectiveness of random forest in regression for small datasets; and
- Though both NRF and NNRF try to take advantage of neural networks and random forest, NNRF outperforms NRF. This is because NRF first learn random forest up front and then train neural networks to simulate the specific instantiation of the RF with a neural network; while NNRF designs a random forest–structured neural network that inherently integrated random forest and neural networks.
- The proposed framework NNRF outperforms the compared regressors on the majority of the datasets used in terms of RMSE and MAE, which implies the effectiveness of NNRF for regression. In particular, though NNRF and RF are both ensemble methods based on tree-structured models, NNRF outperforms RF, because the decision making in each node of NNRF uses more complex nonlinear function, which is able to make better decision than RF.

In summary, by combining the power from random forest and neural networks, the proposed framework NNRF has better regression ability.

Running time comparison: Similarly, we report the running time of NNRF and the compared methods on regression in Figure 5, where the number of trees for RF and NNRF are both 150. NNDT means the average running time of neural network-structured decision tree in NNRF. From the figure, we can also observe that the training of each decision-tree-structured neural network is very efficient. Thus, if we can run these 150 tree-structured neural networks in parallel on a machine of many CPU cores, then *the running time of NNRF can be significantly reduced.*

5.4 Affects of Activation Functions on NNRF

Activation function is very important in NNRF, because it affects which path to go in each node. We adopt Leaky ReLu, because it is effective for alleviating gradient exploding or the vanishing



(a) CompActiv (b) Crime

Fig. 5. Running time comparison for regression.

Table 7. Affects of Different Activation Functions on NNRF for Classification

Dataset	ForestType		Movement	
Metrics	Macro-F1	Micro-F1	Macro-F1	Micro-F1
Sigmoid	0.9048	0.9151	0.8102	0.8193
ReLu	0.9109	0.9212	0.8285	0.8367
Leaky ReLu	0.9156	0.9245	0.8302	0.8401

Table 8. Affects of Different Activation Functions on NNRF for Regression

Dataset	Acetone		CompActiv	
Metrics	RMSE	MAE	RMSE	MAE
Sigmoid	11.38	3.381	2.472	1.702
ReLu	11.25	3.318	2.465	1.680
Leaky ReLu	10.91	3.275	2.448	1.674

problem for deep nets. In this section, we investigate the effects of different activation functions on NNRF for classification and regression. For classification, we empirically set $r = \lceil \sqrt{m} \rceil$, $d = \lfloor \log_2 C \rfloor + 1$, $N = 150$, and $\alpha = 0.00005$. For regression, we keep the other settings the same as that for classification but set $d = 3$. We try three popular activation functions, which includes sigmoid, ReLu, and Leaky ReLu. The results for classification on ForestType and Movement and the results for regression on Acetone and CompActiv are reported in Table 7 and Table 8, respectively. From these two tables, we observe that, (i) generally, Leaky ReLu slightly outperforms ReLu. This is because ReLu is a special case of Leaky ReLu by replacing 0.2 in Equation (2) by 0. In this sense, Leaky ReLu provides more information than ReLu, and (ii) Both Leaky ReLu and ReLu outperform sigmoid. This is because, compared with sigmoid, Leaky ReLu or ReLu can alleviate the gradient vanishing problem and thus can help train the neural networks better.

5.5 Parameter Analysis of NNRF for Classification

The proposed framework for classification has three important parameters, i.e., N, r, and d, where N is the number of neural networks, r is the size of the randomly selected features in each node, and d is the depth of each neural network. In this section, we investigate the impact of the parameters

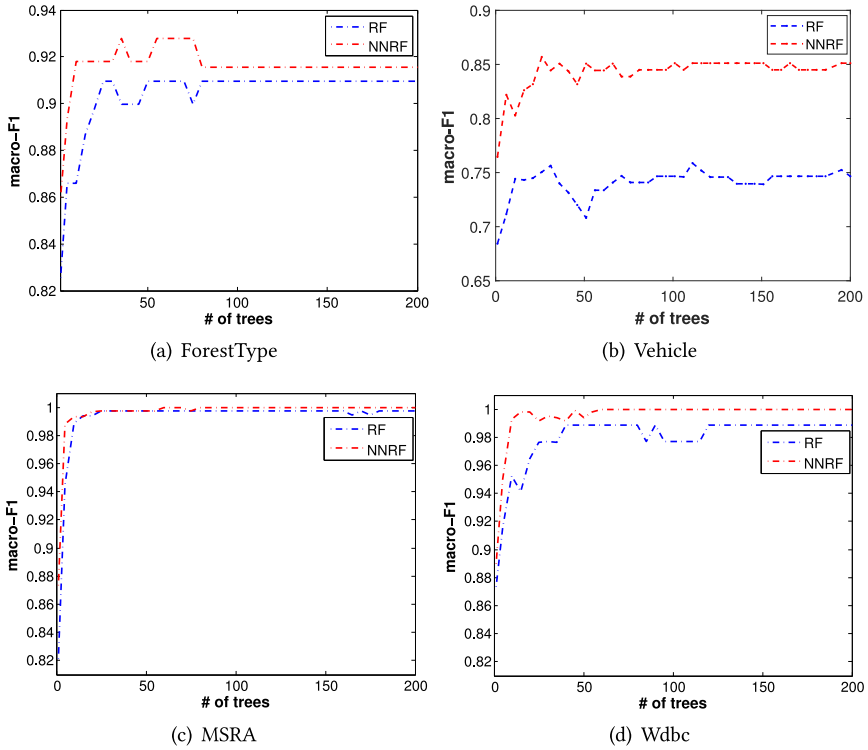


Fig. 6. Macro-F1 of RF and NNRF with different number of trees on ForestType, Vehicle, MSRA, and Wdbc.

N, r, and d on the classification performance of the proposed framework NNRF. Throughout the experiments, α is set to be 0.00005.

5.5.1 Effect of the Number of Trees N on Classification. To investigate the effects of number of trees, N, on classification performance, we first fix r to be $\lceil \sqrt{m} \rceil$ and d to be $\lfloor \log_2 C \rfloor + 1$. We then vary the values of N as $\{1, 5, 10, \ldots, 195, 200\}$. We only show the results in terms of Macro-F1 on ForestType, Vehicle, MSRA, and Wdbc as we have similar observations for the other datasets and the evaluation metric Micro-F1. For comparison, we also conduct experiments with random forest. The experiments are conducted via fivefold cross validation and the average Macro-F1 for the four datasets are shown in Figure 6. From the figure, we make the following observations:

- For both RF and NNRF, the classification performance generally improves with the number of trees N, although increasing beyond a certain point leads to diminishing returns. There are also some random fluctuations in some datasets. From the point of view of tradeoffs between training cost and performance, setting N to be a value within $[50, 150]$ seems like a reasonable point; and
- From the point of view of the comparative performance of RF and NNRF, NNRF tends to consistently outperform RF when N increases. This is because of the more powerful model in NNRF both in terms of the functions at the individual nodes and the less myopic way in which these functions are trained with back-propagation.

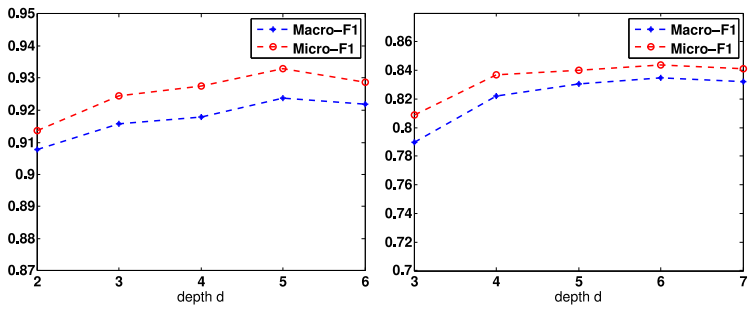
5.5.2 Effect of the Number of Features r on Classification. To investigate the effects of the size of randomly selected features, r, on classification performance, we first set d as $\lfloor \log_2 C \rfloor + 1$ and N to

Table 9. Macro-F1 of RF and NNRF with Different r

Dataset	Algorithm	$\log_2 m$	$\sqrt{m}$	$\frac{m}{4}$	$\frac{m}{2}$
Movement	RF	0.795	0.821	0.764	0.733
	NNRF	0.825	0.830	0.819	0.810
USPS	RF	0.940	0.951	0.932	0.925
	NNRF	0.945	0.956	0.941	0.928

Table 10. Micro-F1 of RF and NNRF with Different r

Dataset	Algorithm	$\log_2 m$	$\sqrt{m}$	$\frac{m}{4}$	$\frac{m}{2}$
Movement	RF	0.797	0.827	0.771	0.743
	NNRF	0.835	0.841	0.825	0.818
USPS	RF	0.943	0.956	0.939	0.929
	NNRF	0.951	0.960	0.945	0.933



(a) ForestType (b) Movement

Fig. 7. Macro-F1 and Micro-F1 of NNRF with different d on ForestType and Movement.

be 150. We then vary r as $\{\log_2 m, \sqrt{m}, \frac{m}{4}, \frac{m}{2}\}$. Note that if any of $\{\log_2 m, \sqrt{m}, \frac{m}{4}, \frac{m}{2}\}$ is non-integer, we round it to the nearest integer. We only show the results in terms of Macro-F1 and Micro-F1 on Movement and USPS as we have similar observations on the other datasets. We conduct fivefold cross validation and the average Macro-F1 and Micro-F1 are reported in Tables 9 and 10. From the two tables, we make the following observations:

- Generally, for both RF and NNRF, the classification performance is better when r is chosen as $\log_2 m$ or $\sqrt{m}$ than when r is chosen as $\frac{m}{4}$ or $\frac{m}{2}$. This is because $\log_2 m$ or $\sqrt{m}$ is smaller than $\frac{m}{4}$ or $\frac{m}{2}$, and thus we introduce more randomness and diversity into the model and can thus learn a model with better generalization [1]; and
- Comparing RF and NNRF, NNRF is more robust and outperforms RF for different r, which shows the effectiveness of NNRF.

5.5.3 Effect of the Neural Network Depth d on Classification. To investigate the effects of the neural network depth d, we first set r as $\lceil \sqrt{m} \rceil$. We then vary d as $\{2, 3, 4, 5, 6\}$ for ForestType and $\{3, 4, 5, 6, 7\}$ for Movement, which is because Forest has 4 classes while Movement has 15 classes. We conduct fivefold cross validation on ForestType and Movement. The average performance in terms of Macro-F1 and Micro-F1 is shown in Figure 7. From the figure, we make the following observations: (i) Generally, as d increases, the performance first increases and then converges or

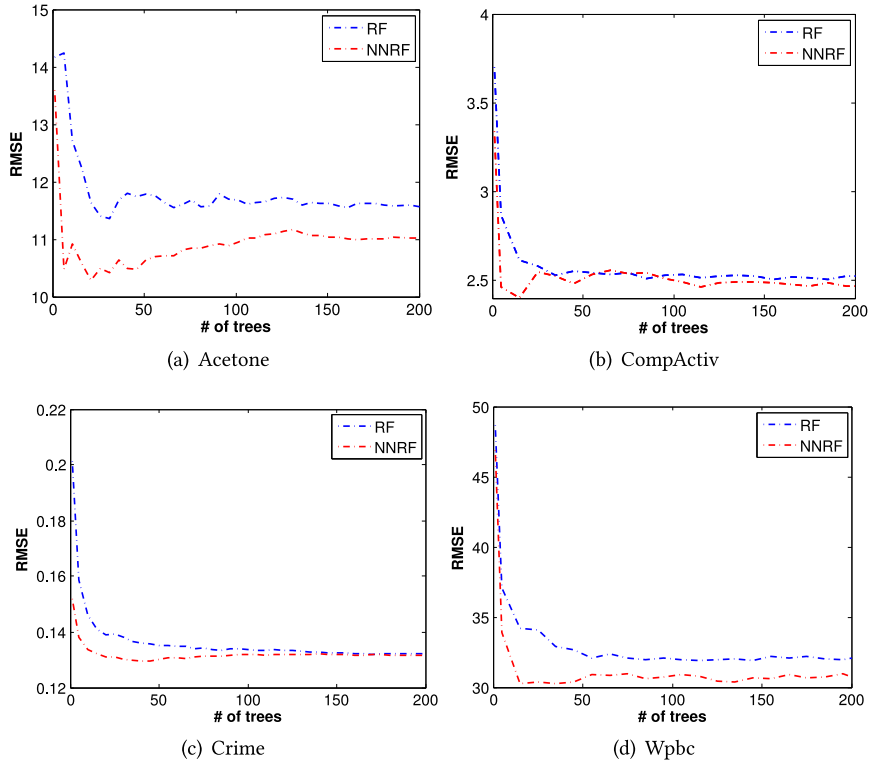


Fig. 8. RMSE of RF and NNRF with different number of trees on datasets.

decreases a little, and (ii) when d is chosen as $\lfloor \log_2 C \rfloor + 1$, the performance is relatively good. For example, for Movement, $\lfloor \log_2 C \rfloor + 1$ is 4 and it already achieves good performance. On the contrary, when d is chosen less than $\lfloor \log_2 C \rfloor$, the performance is not satisfactory. This is because when d is small, the number of leaves in one tree, i.e., dimension of $\mathbf{p}$, is $2^d \leq C$ and does not have enough representation capacity to make good classification.

5.6 Parameter Analysis of NNRF for Regression

Similarly, NNRF for regression also has three important parameters, i.e., N, r, and d. In this section, we investigate the impact of the parameters N, r, and d on the regression performance of the proposed framework NNRF. Throughout the experiments, α is set to be 0.00005.

5.6.1 Effect of the Number of Trees N on Regression. To investigate the effects of number of trees, N, on regression performance, we first fix r to be $\lceil \sqrt{m} \rceil$ and d to be 3. We then vary the values of N as $\{1, 5, 10, \ldots, 195, 200\}$. We only show the results in terms of RMSE on Acetone, CompActiv, Crime, and Wpbc as we have similar observations for the other datasets and the evaluation metric MAE. For comparison, we also conduct experiments with random forest. The experiments are conducted via fivefold cross validation and the average RMSE for the four datasets are shown in Figure 8. From the figure, we make the following observations:

- Similarly to the observation for classification performance, the regression performance of RF and NNRF also improves with the number of trees N and increasing beyond a certain

Table 11. RMSE of RF and NNRF with Different r

Dataset	Algorithm	$\log_2 m$	$\sqrt{m}$	$\frac{m}{4}$	$\frac{m}{2}$
Crime	RF	0.1408	0.1386	0.1299	0.1352
	NNRF	0.1395	0.1320	0.1258	0.1265
Triazines	RF	0.1176	0.1140	0.1105	0.1116
	NNRF	0.1095	0.1079	0.1048	0.10598

Table 12. MAE of RF and NNRF with Different r

Dataset	Algorithm	$\log_2 m$	$\sqrt{m}$	$\frac{m}{4}$	$\frac{m}{2}$
Crime	RF	0.1053	0.0960	0.0917	0.0969
	NNRF	0.1012	0.0935	0.0860	0.0868
Triazines	RF	0.0838	0.0801	0.0793	0.0817
	NNRF	0.0813	0.0801	0.0789	0.0794

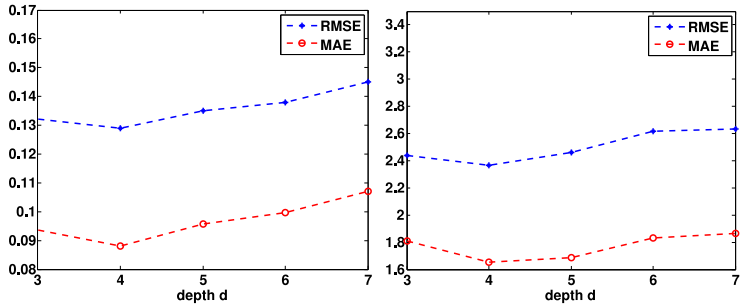
point leads to diminishing returns. For the tradeoffs between training cost and performance, a value of N within $[50, 100]$ seems to be a reasonable choice; and

- Compared with RF, NNRF tends to consistently outperform RF when N increases, which is because of the more powerful model in NNRF both in terms of the functions at the individual nodes and the less myopic way in which these functions are trained with back-propagation.

5.6.2 Effect of the Number of Features r on Regression. To investigate the effects of the size of randomly selected features, r, on regression performance, we first set d as 3 and N to be 100. We then vary r as $\{\log_2 m, \sqrt{m}, \frac{m}{4}, \frac{m}{2}\}$. We round r to the nearest integer if any of $\{\log_2 m, \sqrt{m}, \frac{m}{4}, \frac{m}{2}\}$ is non-integer. We only show the results in terms of RMSE and MAE on Crime and Triazines as we have similar observations on the other datasets. We conduct fivefold cross validation and the average RMSE MAE are reported in Table 11 and 12. From the two tables, we observe that:

- Generally, for both RF and NNRF, the regression performance is better when r is chosen as $\sqrt{m}$ or $\frac{m}{4}$ than when r is chosen as $\log_2 m$ or $\frac{m}{2}$, which is different than classification where or $\log_2 m$ and $\sqrt{m}$ give better classification result. This is for regression; we need more features to predict a numerical value. $\log_2 m$ gives too few features to make good prediction, while $\frac{m}{2}$ introduce too many features and reduces the randomness and diversity in the model. Generally, $\frac{m}{4}$ or $\sqrt{m}$ is between $\log_2 m$ and $\frac{m}{2}$, which can give a good tradeoff between enough prediction capability and randomness and diversity in the model, and
- Comparing RF and NNRF, NNRF is more robust and outperforms RF for different r, which shows the effectiveness of NNRF.

5.6.3 Effect of the Neural Network Depth d on Regression. To investigate the effects of the neural network depth d on the regression performance, we first set r as $\lceil \sqrt{m} \rceil$. We then vary d as $\{3, 4, 5, 6, 7\}$. We conduct fivefold cross validation on Crime and CompActiv. The average performance in terms of RMSE and MAE are shown in Figure 9. From the figure, we observe that, generally, as d increases, the performance first increases then converges or decreases a little. This is because when d is not too large, increasing d increases the learning capacity of NNRF and thus is able to make better prediction. However, when d is too large, the model becomes too complex and may not be well trained on small datasets.



(a) Crime (b) CompActiv

Fig. 9. RMSE and MAE of NNRF with different d on Crime and CompActiv.

6 RELATED WORK

In this section, we will briefly review related work on neural networks, random forest, and effects in combing these two.

6.1 Neural Networks

Neural networks (NN) are very powerful in learning or extracting nonlinear features for many machine-learning and data-mining tasks when abundant training data are given. It has achieved great success in computer vision, speech recognition, and natural language processing. For example, in computervision, convolutional neural networks (CNN) have become the state-of-the-art model for image classification [12, 23], semantic segmentation [26], image representation learning [30], and image generation [31]. Similarly, in natural language processing, LSTM has shown its great ability in sentence classification [44], sentence generation [36], and machine translation [42]. Despite the success of neural networks in specific domains, the success of neural networks are not universe across all domains. First, neural networks, especially deep nets, have massive weights that need a large number of datasets to train and are prone to overfitting on small datasets. Thus, for domains with small datasets, neural networks usually do not work well, while domains with small datasets such as bioinformatics are pervasive. Second, the success of current deep nets such as CNN and LSTM actually rely on the domain insights to design specific structure to reduce the parameters to alleviate overfitting. For example, CNN uses convolution layers and max pooling layers for reducing parameters and learning translation-invariant features from images. Similarly, LSTM shared the weights in each cell to reduce the parameter and designs the forget gate to capture the long-term dependency. However, for general domains whose domain insights are not clear or difficult to be integrated to neural networks, deep nets may not work well. Therefore, designing neural networks that work well on general domains with small datasets is important. Random forest, which is robust to overfitting on small datasets, is a good choice to inspire design, which will be discussed in the next subsection.

6.2 Random Forests

Random forest [4] (RF) is an ensemble-based method that aggregates the results of many decision trees. Generally, each decision tree is a binary tree that decides which path to take based on an input feature at each node. It has been successfully applied in various domains such as bioinformatics [9, 29], remote sensing [27], and compound classification [37]. It is considered a generalist method, which has been demonstrated to beat even the best classifiers on small datasets for general domains [8]. For example, Delgado et al. [8] compared 179 classifiers on the entire UCI

collection of datasets and found that, overall, random forests perform the best. In addition to classification, random forest is also very powerful for regression. Random forest is less likely to overfit than neural network because of the ensemble, bootstrap, and randomized feature sampling used in each decision, which introduces diversity to the model to reduce overfitting. However, unlike neural networks, which learn powerful features by applying nonlinear activation function on input features, RFs only uses one feature to make decision at each node. In this sense, Random forest is not as powerful as neural network. Therefore, it is natural to combine the learning ability of NNs and the ability of reducing overfitting of random forests by designing a random forest–structured neural network.

6.3 Using an NN to Simulate an RF

There are several methods proposed to simulate the output of a decision tree (or random forest) algorithm *on a specific dataset once it has already been constructed* [2, 33]. These methods first construct the decision tree (or random forests) on the dataset up front and then try to simulate *this specific instantiation* of the RF with a NN. Thus, the constructed random forest is itself an input to the algorithm. Such an approach defeats the purpose of a neural network in the first place, because it now has to work with the straitjacket of a specific instantiation of the random forest. That is, it is hard to learn a model that is much better than the base random forest model, even with modifications. Recently, Zhou et al. [45] proposed gcForest by stacking a set of RFs as stacking hidden layers in NNs. The outputs of several RFs in one layer is concatenated with the original features and then are used as the input to the next level RFs. This approach does not take advantage of the feature-learning ability of neural networks, as in each node of the RF, the decision is still made by one feature.

In this article, we propose a fundamentally different approach to design a basic architecture of the neural network, so that it has the property of random forests and has reduced overfitting. We design the decision-making function based on linear combination of input features followed by activation function and thus is more powerful in making decisions.

7 CONCLUSION

In this article, we propose a novel random forest–structured neural network architecture NNRF inspired by random forest. Like random forest, for each input datum, NNRF only has one path activated and thus is efficient to perform forward and backward propagation. In addition, the one-path property also makes the NNRF able to deal with small datasets as the parameters in one path is relatively small. Unlike random forests, NNRF learns complex multivariate functions in each node to choose relevant paths, and is thus able to learn more powerful classifiers. We further extend NNRF for regression by replacing the softmax layer with linear regression layer and using the Euclidean distance as loss function. Extensive experiments on real-world datasets from different domains demonstrate the effectiveness of the proposed framework NNRF for both classification and clustering. Further experiments are conducted to analyze the sensitivity of the hyper-parameters for classification and regression, respectively.

There are several interesting directions that need further investigation. First, in this work, we conduct experiments to demonstrate the effectiveness of NNRF for classification and regression. A detailed theoretical analysis of NNRF, such as its rate of convergence and representation capacity, is worth pursuing. Second, in this article, as an initial attempt of designing random forest–structured neural network, we have not tried the widely adopted tricks such as batch normalization [20] or dropout [35], which has been proven to be very useful for deep nets. Thus, another direction is to introduce these tricks into NNRF for training deeper decision-tree-structured neural networks.

REFERENCES

[1] Charu C. Aggarwal. 2015. *Data Mining—The Textbook*. Springer.

[2] Gérard Biau, Erwan Scornet, and Johannes Welbl. 2016. Neural random forests. *arXiv Preprint arXiv:1604.07143* (2016).

[3] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, New York, NY.

[4] Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[5] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2, 3 (2011), 27:1–27:27.

[6] François Chollet and others. 2015. Keras. Retrieved from https://github.com/fchollet/keras.

[7] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* 12 (Aug. 2011), 2493–2537.

[8] Manuel Fernández Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.* 15, 1 (2014), 3133–3181.

[9] Ramón Díaz-Uriarte and Sara Alvarez De Andres. 2006. Gene selection and classification of microarray data using random forest. *BMC Bioinform.* 7, 1 (2006), 3.

[10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Springer Series in Statistics, Vol. 1. Springer, New York, NY.

[11] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*. IEEE, 6645–6649.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of CVPR*. 770–778.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1026–1034.

[14] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and others. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Sign. Process. Mag.* 29, 6 (2012), 82–97.

[15] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neur. Comput.* 18, 7 (2006), 1527–1554.

[16] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'95)*, Vol. 1. IEEE, 278–282.

[17] Arthur E. Hoerl and Robert W. Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* 12, 1 (1970), 55–67.

[18] David W. Hosmer Jr. and Stanley Lemeshow. 2004. *Applied Logistic Regression*. John Wiley & Sons.

[19] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2010. A practical guide to support vector classification. (2010).

[20] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML'15)*. 448–456.

[21] Diederik P. Kingma and Jimmy Lei Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of 3rd International Conference for Learning Representations*.

[22] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. 2015. Deep neural decision forests. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1467–1475.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

[24] Max Kuhn. 2008. Caret package. *Journal of Statistical Software* 28, 5 (2008).

[25] Yang Li, Quan Pan, Suhang Wang, Tao Yang, and Erik Cambria. 2018. A generative model for category text generation. *Information Sciences* 450 (2018), 301–315.

[26] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 3431–3440.

[27] Mahesh Pal. 2005. Random forest classifier for remote sensing classification. *Int. J. Remote Sens.* 26, 1 (2005), 217–222.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, and others. 2011. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.

[29] Yanjun Qi, Judith Klein-Seetharaman, and Ziv Bar-Joseph. 2005. Random forest similarity for protein-protein interaction prediction from multiple sources. In *Proceedings of the Pacific Symposium on Biocomputing*. 531–542.

[30] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of ICLR*.

[31] Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. 2016. Generative adversarial text to image synthesis. In *Proceedings of the International Conference on Machine Learning (ICML'16)*. 1060–1069.

[32] David L. Richmond, Dagmar Kainmueller, Michael Y. Yang, Eugene W. Myers, and Carsten Rother. 2015. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *arXiv Preprint arXiv:1507.07583* (2015).

[33] Ishwar Krishnan Sethi. 1990. Entropy nets: From decision trees to neural networks. *Proc. IEEE* 78, 10 (1990), 1605–1613.

[34] Alex J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Stat. Comput.* 14, 3 (2004), 199–222.

[35] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*. 3104–3112.

[37] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. 2003. Random forest: A classification and regression tool for compound classification and QSAR modeling. *J. Chem. Inf. Comput. Sci.* 43, 6 (2003), 1947–1958.

[38] Suhang Wang, CHaru Aggarwal, and Huan Liu. 2017. Using a random forest to inspire a neural network and improving on it. In *Proceedings of the SIAM International Conference on Data Mining (SDM'17)*.

[39] Suhang Wang and Huan Liu. 2018. Deep learning for feature representation. *Feature Engineering for Machine Learning and Data Analytics*. 279–307.

[40] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. 2017. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 391–400.

[41] Yilin Wang, Suhang Wang, Guojun Qi, Jiliang Tang, and Baoxin Li. 2018. Weakly supervised facial attribute manipulation via deep adversarial network. In *Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV'18)*. IEEE, 112–121.

[42] Yonghui Wu, Mike Schuster, Zhifeng Chen, and Others. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv Preprint arXiv:1609.08144* (2016).

[43] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning (ICML'15)*. 2048–2057.

[44] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'16)*. 1480–1489.

[45] Zhi-Hua Zhou and Ji Feng. 2017. Deep forest: Towards an alternative to deep neural networks. In *Proceedings of IJCAI (2017)*.

Received November 2017; revised April 2018; accepted June 2018