Using a Random Forest to Inspire a Neural Network and Improving on It

Suhang Wang*† Charu Aggarwal‡ Huan Liu§

Abstract

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 is somewhat limited, neural networks are known not to perform well with respect to traditional machine learning methods such as random forests. In this paper, 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 in order to improve the generalization accuracy. Experimental results on 10 real-world benchmark datasets demonstrate the effectiveness of the proposed enhancements.

1 Introduction

Neural networks have become increasingly popular in recent years because of their tremendous success in image classification [1, 2], speech recognition [3, 4] and natural language processing tasks [5, 6]. In fact, deep learning methods have regularly won many recent challenges in these domains [3, 7]. This success is, in part, because the special structure of these domains often allows the use of specialized neural network architectures such as convolutional neural networks [3], 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 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 data set 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 [8, 9] regularly outperform neural networks in arbitrary domains [10], 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 data set 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 [11], and often outperform [10] even the best of classifiers. A recent study [10] evaluated 179 classifiers from 17 families on the *entire UCI collection of data sets*, 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 data sets 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

*Part of the work is done during first author's internship at IBM T.J Watson

†Arizona State University, suhang.wang@asu.edu

‡IBM T.J. Watson, charu.aggarwal@us.ibm.com

§Arizona State University, huan.liu@asu.edu

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. On the other hand, 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 in order to reduce the parameter footprint. In this paper, 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 data sets.

It is noteworthy that several methods have been proposed to simulate the output of a decision tree (or random forest) algorithm *on a specific data set, once it has already been constructed* [12, 13]. In other words, such an approach first constructs the decision tree (or random forests) on the data set 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 strait jacket 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 paper, 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 backpropagation and no specific instantiation of a random forest is used as input. On the other hand, 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 [14, 15], although the resulting random forests have a specialized structure that is suited to the image domain [15]. This paper will focus on designing a neural network architecture which has random forest structure such that it has better classification ability and reduced overfitting. The main contributions of the paper 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 data sets; and
- We conduct extensive experiments to demonstrate the effectiveness of the proposed framework.

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 paper, matrices are written as boldface capital letters such as $\mathbf{M}$, $\mathbf{W}_{ij}$, and vectors are denoted as boldface 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 i-th row and j-th column, respectively. Similarly, $\mathbf{p}(i)$ denotes the i-th 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

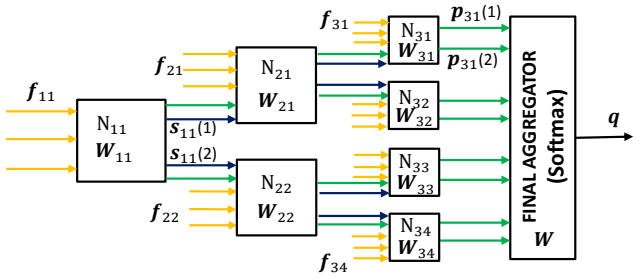


Figure 1: An Illustration of the Decision Tree Structured Neural Network Architecture with $d = 3$

network is simulating. Thus, one 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 $1+2^1+\ldots+2^{d-1}$, 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 in order 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 in order 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 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 *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 3-layer neural network is illustrated in Figure 1. For ease of explanation, we name the nodes as N_{ij}, which means the j-th node in the i-th layer. A parent node N_{ij} are 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 node 2 in layer 2 and are 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 tree-like 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 in order to ensure that only one path is activated by a given training instance in the tree-like 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 backpropagation 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 in order 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 backpropagation 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 backpropagation algorithm is more powerful than the typical training process of a decision tree which is very myopic at a given node. The backpropagation 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 Neural Network Design

Previous section gives the overall architecture of the neural network, in this section, we give the inner working of the neural network. We will 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 $\mathbf{f}_{11}$, N_{11} calculate the output $\mathbf{p}_{11}$ as

$$\mathbf{p}_{11} = g(\mathbf{W}_{11}\mathbf{f}_{11} + \mathbf{b}_{11}) \tag{3.1}$$

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 [16], in this work, we use Leaky ReLu. Then, $g(x)$ and the derivative of $g(x)$ w.r.t x is given as

(3.2)
$$g(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.2x, & \text{if } x < 0 \end{cases} \quad g'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0.2, & \text{if } x < 0 \end{cases}$$

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

$$\mathbf{s}_{11}(1) = \mathbb{I}(\mathbf{a}_{11}(1), \mathbf{a}_{11}(2)) \quad \mathbf{s}_{11}(2) = \mathbb{I}(\mathbf{a}_{11}(2), \mathbf{a}_{11}(1)) \tag{3.3}$$

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., 2-nd 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 the outputs of N_{21} and N_{22} are calculated as

$$\mathbf{p}_{2j} = \begin{cases} g\Big(\mathbf{W}_{2j}\begin{bmatrix}\mathbf{f}_{2j} \\ \mathbf{p}_{11}(j)\end{bmatrix} + \mathbf{b}_{11}\Big) & \text{if } \mathbf{s}_{11}(j) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases} \tag{3.4}$$

$$\mathbf{s}_{2j} = \begin{cases} \begin{bmatrix}\mathbb{I}(\mathbf{p}_{2j}(1), \mathbf{p}_{2j}(2)) \\ \mathbb{I}(\mathbf{p}_{2j}(2), \mathbf{p}_{2j}(1))\end{bmatrix} & \text{if } \mathbf{s}_{11}(j) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases} \tag{3.5}$$

where $j = 1, 2$. The idea behind Eq.(3.4) and Eq.(3.5) 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

$$\mathbf{p}_{i+1,t} = \begin{cases} g\Big(\mathbf{W}_{i+1,t}\begin{bmatrix}\mathbf{f}_{i+1,t} \\ \mathbf{p}_{ij}(k)\end{bmatrix} + \mathbf{b}_{i+1,t}\Big) & \text{if } \mathbf{s}_{ij}(k) == 1 \\ \mathbf{0} \in \mathbb{R}^{2\times 1}, & \text{otherwise} \end{cases} \tag{3.6}$$

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

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. Eq.(3.6) 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

$$\mathbf{q}(c) = \frac{\exp(\mathbf{W}(c, :)\mathbf{p} + \mathbf{b}(c))}{\sum_{k=1}^{C} \exp(\mathbf{W}(k, :)\mathbf{p} + \mathbf{b}(k))} \tag{3.7}$$

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

$$L(\mathbf{y}, \mathbf{q}) = -\sum_{c=1}^{C} \mathbf{y}(c) \log \mathbf{q}(c) \tag{3.8}$$

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 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, the

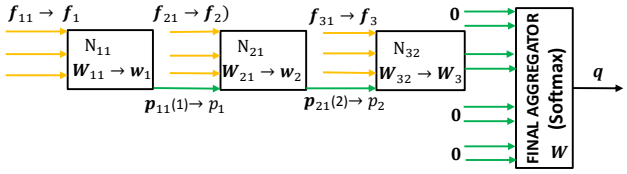


Figure 2: An Example of Reduced Neural Network

objective function is written as

$$\min_{\theta} -\frac{1}{n}\sum_{l=1}^{n} L(\mathbf{Y}(l,:), \mathcal{T}(\mathbf{X}(l,:)) + \alpha\mathcal{R}(\theta) \tag{3.9}$$

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 $\mathbf{X}(l,:)$. $\mathcal{R}(\theta)$ is the regularizer to avoid over-fitting, which is defined as

$$\mathcal{R}(\theta) = \|\mathbf{W}\|_F^2 + \|\mathbf{b}\|^2 + \sum_{i=1}^{d}\sum_{j=1}^{2^{i-1}} \|\mathbf{W}_{ij}\|_F^2 \tag{3.10}$$

3.2 Efficient Backpropagation We use back-propagation to train the tree structured neural network. Let us focus on the cost function in Eq.(3.8) to show how to perform efficient back-propagation as extension to Eq.(3.9) is simple. For simplicity of notation, let's 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's 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 i-th layer of the simplified neural network as $\mathbf{w}_i$ and b_i, the features of each node as $\mathbf{f}_i$, the link connecting $i-1$-th layer to i-th 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 backpropagation 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" as $\delta(c)$ as

$$\delta(c) = \frac{\partial \mathcal{J}}{\partial \mathbf{u}(c)} = \mathbf{q}(c) - \mathbf{y}(c), \quad c = 1, \ldots, C \tag{3.11}$$

Then, we have

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}(c,:)} = \delta_c \mathbf{p}^T, \quad \frac{\partial \mathcal{J}}{\partial \mathbf{b}(c)} = \delta_c \tag{3.12}$$

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

(3.13)

$$\delta_d(k) = \frac{\partial \mathcal{J}}{\partial \mathbf{u}_d(k)} = \sum_c \delta(c)\mathbf{W}(c, ind(\mathbf{p}_d(k)))g'(\mathbf{u}_d(k))$$

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}_d(k)} = \delta_d(k)[\mathbf{f}_d^T,\ \mathbf{p}_{d-1}], \quad \frac{\partial \mathcal{J}}{\partial \mathbf{d}_d(k)} = \delta_d(k)$$

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

$$\delta_i = \frac{\partial \mathcal{J}}{\partial u_i} = \begin{cases} \sum_{k=1}^{2} \delta_d(k)\mathbf{W}_d(k, r+1)g'(u_i), & i = d-1 \\ \delta_{i+1}\mathbf{w}_{i+1}(r+1)g'(u_i), & i = d-2, \ldots, 1 \end{cases} \tag{3.14}$$

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}_i} = \begin{cases} \delta_i[\mathbf{f}_i^T,\ p_{i-1}], & i = d-1, \ldots, 2 \\ \delta_1 \mathbf{f}_1^T, & i = 1 \end{cases}$$

$$\frac{\partial \mathcal{J}}{\partial d_i} = \delta_i, \quad i = d-1, \ldots, 1$$

3.3 Algorithm for NNRF The algorithm to train a d-layer tree structured neural network 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 Neural Network with Random Forest Structure (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

$$\mathbf{q}_a = \frac{1}{N}\sum_{i=1}^{N} \mathcal{T}_i(\mathbf{x}) \tag{3.15}$$

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 Eq.(3.6) is $\mathcal{O}(rd)$. Since $\mathbf{p}$ only has two nonnegative elements, the cost of softmax in Eq.(3.7) to get $\mathbf{q}$ is $\mathcal{O}(C)$. Therefore, the cost of forward propagation is $\mathcal{O}(rd + C)$ for each data sample in each tree. Similarly, the cost of backward propagation using Eq.(3.13) and Eq.(3.14) is also $\mathcal{O}(C + rd)$. Thus, the total cost of training N tree structured neural networks with depth d is $\mathcal{O}(t(C + rd)nN)$, where t is number of epochs for training each neural network. Since r is usually chosen as $\sqrt{m}$, the total cost is approximately $\mathcal{O}(t(C + \sqrt{m}d)nN)$, 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** $\{T_i\}_{i=1}^N$

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 $\mathcal{O}(2^d(2r + C))$. 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 $\mathcal{O}((\sqrt{m}+C)\cdot 2^C)$, which is modest and thus can be well trained even when the data size is small.

4 Experimental Results

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework NNRF. We begin by introducing datasets and experimental setting, then we compare NNRF with state-of-the-art classifiers. Further experiments are conducted to investigate the effects of the hyper-parameters on NNRF.

4.1 Datasets and Experimental Settings The experiments are conducted on 10 publicly available benchmark datasets, which includes 5 UCI[1] datasets, i.e., forest type mapping (ForestType), speech record of 26 alphabets (Isolet), sonar signals of mines v.s. rocks (Sonar), chemical analysis of wines (Wine) and Wisconsin diagnostic breast cancer (wdbc), 3 image datasets, i.e., images of 20 objects (COIL-20)[2], images of handwritten digits (USPS)[3] and images of faces (MSRA)[4], one bioinfomatics dataset (Bioinformatics) [17] 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 amount of parameter may not work well on these datasets.

Table 1: Statistics of the Dataset

Dataset	# Samples	# Feature	# Class
COIL20	1,440	1,024	20
ForestType	523	26	4
Isolet	7,797	617	26
Bioinfomatics	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

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.

4.2 Classification Performance Comparison To evaluate the ability of NNRF on classification, we compare the proposed framework NNRF with other classical and state-of-the-art classifiers. 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 [19].
- SVM: Support vector machine [20] is a classical and popular classifier which tries to find the best hyperplane that represents the largest margin between classes. We use the well-known libsvm [20] with rbf kernel for classification.

[1] All the 5 UCI datasets 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

Table 2: Classification results(Macro-F1%±std) of different classifiers on different datasets.

Dataset	LR	SVM	NN	RF	**NNRF**
COIL20	96.07±1.09	97.97±1.25	98.11±1.07	**99.93**±0.13	**99.93**±0.12
ForestType	88.24±3.11	88.81±2.07	89.51±2.49	90.95±1.89	**91.56**±2.01
Isolet	95.49±0.51	95.68±0.48	95.65±0.72	94.34±0.39	**95.72**±0.51
Bioinfomatics	78.89±6.74	81.32±5.47	73.57±6.94	70.73±4.47	**82.98**±4.94
Sonar	75.96±7.04	80.78±3.24	82.74±4.51	84.92±6.00	**86.12**±4.98
USPS	93.45±0.37	94.86±0.27	94.21±0.43	95.10±0.34	**95.60**±0.36
Wine	59.61±3.96	70.82±3.67	68.11±5.62	66.87±2.44	**71.10**±3.19
Movement	68.94±2.91	76.41±4.16	82.10±3.81	82.12±2.74	**83.02**±2.96
MSRA	99.71±0.23	**100**±0.00	**100**±0.00	99.47±0.19	**100**±0.00
Wdbc	97.68±1.23	97.14±1.81	98.30±1.52	98.67±2.03	**99.9**±0.09

Table 3: Classification results(Micro-F1%±std) of different classifiers on different datasets.

Dataset	LR	SVM	NN	RF	**NNRF**
COIL20	96.13±1.04	97.95±1.23	98.18±1.10	99.93±0.13	**99.94**±0.14
ForestType	89.62±2.15	89.62±1.94	90.57±1.83	91.51±1.85	**92.45**±1.90
Isolet	95.51±0.51	95.70±0.49	95.66±0.56	94.35±0.39	**95.79**±0.43
Bioinfomatics	82.50±4.11	85.01±2.92	80.09±4.13	78.75±3.15	**86.25**±3.02
Sonar	75.96±7.04	80.78±3.24	82.74±4.51	84.92±6.00	**86.12**±4.98
USPS	94.11±0.33	95.17±0.25	94.89±0.31	95.62±0.32	**96.01**±0.33
Wine	70.35±1.32	70.81±3.15	69.19±5.29	69.19±1.32	**71.38**±2.78
Movement	70.13±1.81	77.33±3.92	82.67±3.61	82.68±2.43	**84.01**±2.68
MSRA	99.73±0.21	**100**±0.00	**100**±0.00	99.45±0.22	**100**±0.00
Wdbc	97.68±1.23	97.14±1.81	98.30±1.52	98.67±2.03	**99.9**±0.09

- NN: This is a one hidden-layer feedforward neural network [21] with softmax as the output layer and cross-entropy as the classifier. We use the implementation of Keras, which is a popular deep learning and neural networks toolbox. Note that we also tried neural networks with more hidden-layers. However, generally, neural network with one hidden-layer works best for the data used as the sample sizes are small.

- RF: Random forest [8] is a classical and popular ensemble based method which aggregate independent decision trees for classification. It is one of the most powerful classifiers [10]. We use the implementation by scikit-learn.

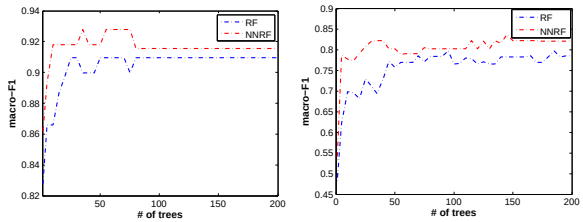
For each classifier, there are some parameters to be set. In the experiment, we use 10-fold cross validation on the training data to set the parameters. Note that no test data are involved in the parameter tuning. Specifically, for NNRF, we empirically set $r = \lceil \sqrt{m} \rceil$, $d = \lfloor \log_2 C \rfloor + 1$, $N = 150$ 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 4.3. 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 2 and 3, respectively. From the two tables, we make the following observations:

- Generally, the four compared classifiers, i.e., LR, SVM, NN and RF 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.

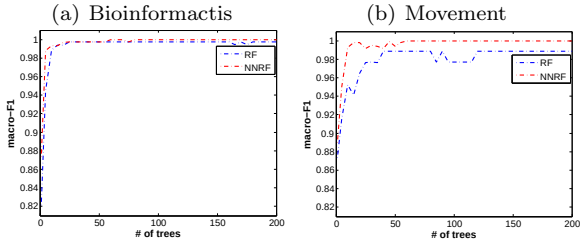
- RF is slightly better than the remaining three classifiers on 6 datasets. This observation is consistent with the observation in [10] that generally RF achieves the best performance on the majority of the datasets among 179 classifiers.

- 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 backpropagation 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.

We conduct the t-test on all performance comparisons and it is evident from t-test that all improvements are significant. 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.



(a) Bioinformactis (b) Movement



(c) MSRA (d) Wdbc

Figure 3: Macro-F1 of RF and NNRF with different number of trees on datasets ForestType, Movement, MSRA and wdbc.

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

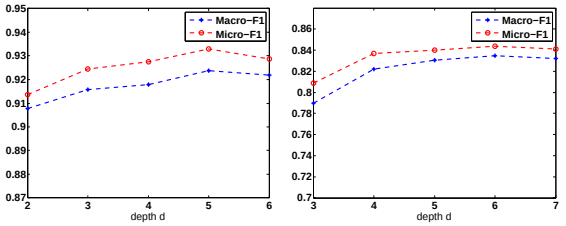
4.3.1 Effect of the Number of Trees N To investigate the effects of number of trees, N, on classification performance, we first fix r to be $\lceil\sqrt{m}\rceil$ and d to be $\lfloor\log_2 C\rfloor + 1$. We then vary the values of N as $\{1, 5, 10, \ldots, 195, 200\}$. We only show the results in terms of Macro-F1 on ForestType, Movement, 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 5-fold cross validation and the average Macro-F1 for the four datasets are shown in Figure 3. From the figure, we make the following observations: (1) 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 data sets. From the point of view of trade-offs between training cost and performance, setting N to be a value within $[50, 100]$ seems like a reasonable point; and (2) 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 backpropagation.

Table 4: Macro-F1 of RF and NNRF with different r

Dataset	Alg	$\log_2 m$	$\sqrt{m}$	$\frac{m}{4}$	$\frac{m}{2}$
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 5: Micro-F1 of RF and NNRF with different r

Dataset	Alg	$\log_2 m$	$\sqrt{m}$	$\frac{m}{4}$	$\frac{m}{2}$
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

Figure 4: Macro-F1 and Micro-F1 of NNRF with different d on ForestType and Movement

4.3.2 Effect of the Number of Features r 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 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 5-fold cross validation and the average Macro-F1 and Micro-F1 are reported in Table 4 and 5. From the two tables, we make the following observations: (i) 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 [22]; and (ii) Comparing RF and NNRF, NNRF is more robust and outperforms RF for different r, which shows the effectiveness of NNRF.

4.3.3 Effect of the Neural Network Depth d 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 5-fold cross validation on

ForestType and Movement. The average performance in terms of Macro-F1 and Micro-F1 are shown in Figure 4. From the figure, we make the following observations: (i) Generally, as d increases, the performance first increase then converges or decrease 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 then $\lfloor \log_2 C \rfloor$, 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 Conclusion

In this paper, we propose a novel neural network architecture NNRF inspired by random forest. Like random forest, for each input data, 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 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. Extensive experiments on real-world datasets from different domains demonstrate the effectiveness of the proposed framework NNRF. Further experiments are conducted to analyze the sensitivity of the hyper-parameters.

There are several interesting directions need further investigation. First, we only consider the classification task in this work. Since random forest is also a very powerful tool for regression, thus improving random forest for regression in a similar way is promising. Therefore, one direction we would like to investigate is to extend NNRF for regression. Second, a detailed theoretical analysis of NNRF, such as rate of convergence, is worth pursuing.

6 Acknowledgements

This material is based upon work supported by, or in part by, the NSF grants #1614576 and IIS-1217466, and the ONR grant N00014-16-1-2257.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*. IEEE, 2013, pp. 6645–6649.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *JMLR*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[7] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *ICML*, vol. 2, no. 3, 2015, p. 5.

[8] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] T. K. Ho, "Random decision forests," in *ICDAR*, vol. 1. IEEE, 1995, pp. 278–282.

[10] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems," *JMLR*, vol. 15, no. 1, pp. 3133–3181, 2014.

[11] M. Kuhn, "Caret package," *Journal of Statistical Software*, vol. 28, no. 5, 2008.

[12] G. Biau, E. Scornet, and J. Welbl, "Neural random forests," *arXiv preprint arXiv:1604.07143*, 2016.

[13] I. K. Sethi, "Entropy nets: from decision trees to neural networks," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1605–1613, 1990.

[14] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo, "Deep neural decision forests," in *CVPR*, 2015, pp. 1467–1475.

[15] D. L. Richmond, D. Kainmueller, M. Y. Yang, E. W. Myers, and C. Rother, "Relating cascaded random forests to deep convolutional neural networks for semantic segmentation," *arXiv preprint arXiv:1507.07583*, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *CVPR*, 2015.

[17] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[18] D. W. Hosmer Jr and S. Lemeshow, *Applied logistic regression*. John Wiley & Sons, 2004.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[20] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *TIST*, 2011.

[21] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[22] C. C. Aggarwal, *Data mining*. Springer, 2015.