Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)

Deep Forest: Towards an Alternative to Deep Neural Networks*

Zhi-Hua Zhou and **Ji Feng**
National Key Lab for Novel Software Technology, Nanjing University, Nanjing 210023, China
{zhouzh, fengj}@lamda.nju.edu.cn

Abstract

In this paper, we propose gcForest, a decision tree ensemble approach with performance highly competitive to deep neural networks in a broad range of tasks. In contrast to deep neural networks which require great effort in hyper-parameter tuning, gcForest is much easier to train; even when it is applied to different data across different domains in our experiments, excellent performance can be achieved by almost same settings of hyper-parameters. The training process of gcForest is efficient, and users can control training cost according to computational resource available. The efficiency may be further enhanced because gcForest is naturally apt to parallel implementation. Furthermore, in contrast to deep neural networks which require large-scale training data, gcForest can work well even when there are only small-scale training data.

1 Introduction

In recent years, deep neural networks have achieved great success in various applications, particularly in tasks involving visual and speech information [Krizhenvsky *et al.*, 2012; Hinton *et al.*, 2012], leading to the hot wave of deep learning [Goodfellow *et al.*, 2016].

Though deep neural networks are powerful, they have apparent deficiencies. First, it is well known that a huge amount of training data are usually required for training, disabling deep neural networks to be directly applied to tasks with small-scale data. Note that even in the big data era, many real tasks still lack sufficient amount of *labeled* data due to high cost of labeling, leading to inferior performance of deep neural networks in those tasks. Second, deep neural networks are very complicated models and powerful computational facilities are usually required for the training process, encumbering individuals outside big companies to fully exploit the learning ability. More importantly, deep neural networks are with too many hyper-parameters, and the learning performance depends seriously on careful tuning of them. For example, even when several authors all use convolutional neural networks [LeCun *et al.*, 1998; Krizhenvsky *et al.*, 2012; Simonyan and Zisserman, 2014], they are actually using different learning models due to the many different options such as the convolutional layer structures. This fact makes not only the training of deep neural networks very tricky, like an art rather than science/engineering, but also theoretical analysis of deep neural networks extremely difficult because of too many interfering factors with almost infinite configurational combinations.

It is widely recognized that the *representation learning* ability is crucial for deep neural networks. It is also noteworthy that, to exploit large training data, the capacity of learning models should be large; this partially explains why the deep neural networks are very complicated, much more complex than ordinary learning models such as support vector machines. We conjecture that if we can endow these properties to some other suitable forms of learning models, we may be able to achieve performance competitive to deep neural networks but with less aforementioned deficiencies.

In this paper, we propose gcForest (multi-Grained Cascade Forest), a novel decision tree ensemble method. This method generates a deep forest ensemble, with a cascade structure which enables gcForest to do representation learning. Its representational learning ability can be further enhanced by multi-grained scanning when the inputs are with high dimensionality, potentially enabling gcForest to be contextual or structural aware. The number of cascade levels can be adaptively determined such that the model complexity can be automatically set, enabling gcForest to perform excellently even on small-scale data. Moreover, users can control training costs according to computational resources available. The gcForest has much fewer hyper-parameters than deep neural networks; even better news is that its performance is quite robust to hyper-parameter settings, such that in most cases, even across different data from different domains, it is able to get excellent performance by using the default setting. This makes not only the training of gcForest convenient, but also theoretical analysis, although beyond the scope of this paper, potentially easier than deep neural networks (needless to say that tree learners are typically easier to analyze than neural networks). In our experiments, gcForest achieves highly competitive performance to deep neural networks, whereas the training time cost of gcForest is smaller than that of deep

*This research was supported by NSFC (61333014), 973 Program (2014CB340501) and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

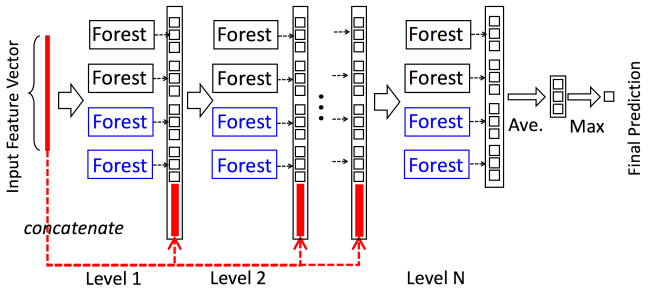

Figure 1: Illustration of the cascade forest structure. Suppose each level of the cascade consists of two random forests (black) and two completely-random tree forests (blue). Suppose there are three classes to predict; thus, each forest will output a three-dimensional class vector, which is then concatenated for re-representation of the original input.

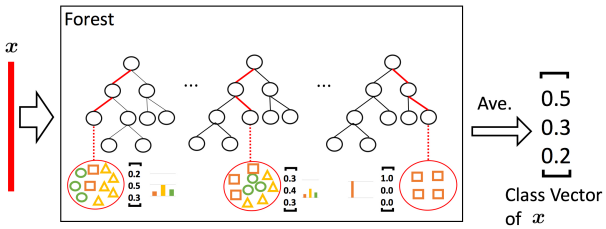

Figure 2: Illustration of class vector generation. Different marks in leaf nodes imply different classes.

neural networks.

We believe that in order to tackle complicated learning tasks, it is likely that learning models have to go deep. Current deep models, however, are always neural networks, multiple layers of parameterized differentiable nonlinear modules that can be trained by backpropagation. It is interesting to consider whether deep learning can be realized with other modules, because they have their own advantages and may exhibit great potentials if being able to go deep. This paper devotes to addressing this fundamental question and illustrates how to construct deep forest; this may open a door towards alternative to deep neural networks for many tasks.

In the next sections we will introduce gcForest and report on experiments, followed by related work and conclusion.

2 The Proposed Approach

In this section we will first introduce the cascade forest structure, and then the multi-grained scanning, followed by the overall architecture and remarks on hyper-parameters.

2.1 Cascade Forest Structure

Representation learning in deep neural networks mostly relies on the layer-by-layer processing of raw features. Inspired by this recognition, gcForest employs a cascade structure, as illustrated in Figure 1, where each level of cascade receives feature information processed by its preceding level, and outputs its processing result to the next level.

Each level is an ensemble of decision tree forests, i.e., an ensemble of ensembles. Here, we include different types of forests to encourage the *diversity*, as it is well known that diversity is crucial for ensemble construction [Zhou, 2012]. For simplicity, suppose that we use two completely-random tree forests and two random forests [Breiman, 2001]. Each completely-random tree forest contains 500 completely-random trees [Liu *et al.*, 2008], generated by randomly selecting a feature for split at each node of the tree, and growing tree until each leaf node contains only the same class of instances. Similarly, each random forest contains 500 trees, by randomly selecting $\sqrt{d}$ number of features as candidate (d is the number of input features) and choosing the one with the best *gini* value for split. The number of trees in each forest is a hyper-parameter, which will be discussed in Section 2.3.

Given an instance, each forest will produce an estimate of class distribution, by counting the percentage of different classes of training examples at the leaf node where the concerned instance falls, and then averaging across all trees in the same forest, as illustrated in Figure 2, where red color highlights paths along which the instance traverses to leaf nodes.

The estimated class distribution forms a class vector, which is then concatenated with the original feature vector to be input to the next level of cascade. For example, suppose there are three classes, then each of the four forests will produce a three-dimensional class vector; thus, the next level of cascade will receive 12 ($= 3 \times 4$) augmented features.

To reduce the risk of overfitting, class vector produced by each forest is generated by k-fold cross validation. In detail, each instance will be used as training data for $k - 1$ times, resulting in $k - 1$ class vectors, which are then averaged to produce the final class vector as augmented features for the next level of cascade. After expanding a new level, the performance of the whole cascade will be estimated on validation set, and the training procedure will terminate if there is no significant performance gain; thus, the number of cascade levels is automatically determined. In contrast to most deep neural networks whose model complexity is fixed, gcForest adaptively decides its model complexity by terminating training when adequate. This enables it to be applicable to different scales of training data, not limited to large-scale ones.

2.2 Multi-Grained Scanning

Deep neural networks are powerful in handling feature relationships, e.g., convolutional neural networks are effective on image data where spatial relationships among the raw pixels are critical [LeCun *et al.*, 1998; Krizhenvsky *et al.*, 2012]; recurrent neural networks are effective on sequence data where sequential relationships are critical [Graves *et al.*, 2013; Cho *et al.*, 2014]. Inspired by this recognition, we enhance cascade forest with a procedure of multi-grained scanning.

As Figure 3 illustrates, sliding windows are used to scan the raw features. Suppose there are 400 raw features and a window size of 100 features is used. For sequence data, a 100-dimensional feature vector will be generated by sliding the window for one feature; in total 301 feature vectors are produced. If the raw features are with spacial relationships, such as a 20×20 panel of 400 image pixels, then a 10×10 window will produce 121 feature vectors (i.e., 121 10×10

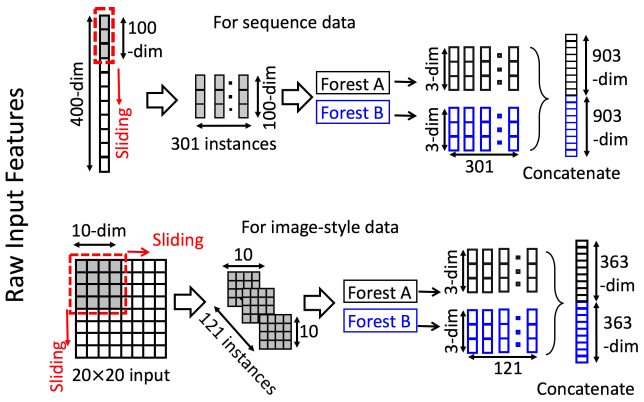

Figure 3: Illustration of feature re-representation using sliding window scanning. Suppose there are three classes, raw features are 400-dim, and sliding window is 100-dim.

panels). All feature vectors extracted from positive/negative training examples are regarded as positive/negative instances, which will then be used to generate class vectors like in Section 2.1: the instances extracted from the same size of windows will be used to train a completely-random tree forest and a random forest, and then the class vectors are generated and concatenated as transformed features. As Figure 3 illustrates, suppose that there are 3 classes and a 100-dimensional window is used; then, 301 three-dimensional class vectors are produced by each forest, leading to a 1,806-dimensional transformed feature vector corresponding to the original 400-dimensional raw feature vector. Note that when transformed feature vectors are too long to be accommodated, feature sampling can be performed, e.g., by subsampling the instances generated by sliding window scanning, since completely-random trees do not rely on feature split selection whereas random forests are quite insensitive to inaccurate feature split selection.

Figure 3 shows only one size of sliding window. By using multiple sizes of sliding windows, differently grained feature vectors will be generated, as shown in Figure 4.

2.3 Overall Procedure and Hyper-Parameters

Figure 4 summarizes the overall procedure of gcForest. Suppose that the original input is of 400 raw features, and three window sizes are used for multi-grained scanning. For m training examples, a window with size of 100 features will generate a data set of $301 \times m$ 100-dimensional training examples. These data will be used to train a completely-random tree forest and a random forest, each containing 500 trees. If there are three classes to be predicted, a 1,806-dimensional feature vector will be obtained as described in Section 2.1. The transformed training set will then be used to train the 1st-grade of cascade forest.

Similarly, sliding windows with sizes of 200 and 300 features will generate 1,206-dimensional and 606-dimensional feature vector, respectively, for each original training example. The transformed feature vectors, augmented with the class vector generated by the previous grade, will then be used to train the 2nd-grade and 3rd-grade of cascade forests, respectively. This procedure will be repeated till convergence of validation performance. In other words, the final model is actually a cascade of cascade forests, where each level in the cascade consists of multiple grades (of cascade forests), each corresponding to a grain of scanning, as shown in Figure 4. Note that for difficult tasks, users can try more grains if computational resource allows.

Given a test instance, it will go through the multi-grained scanning procedure to get its corresponding transformed feature representation, and then go through the cascade till the last level. The final prediction will be obtained by aggregating the four 3-dimensional class vectors at the last level, and taking the class with the maximum aggregated value.

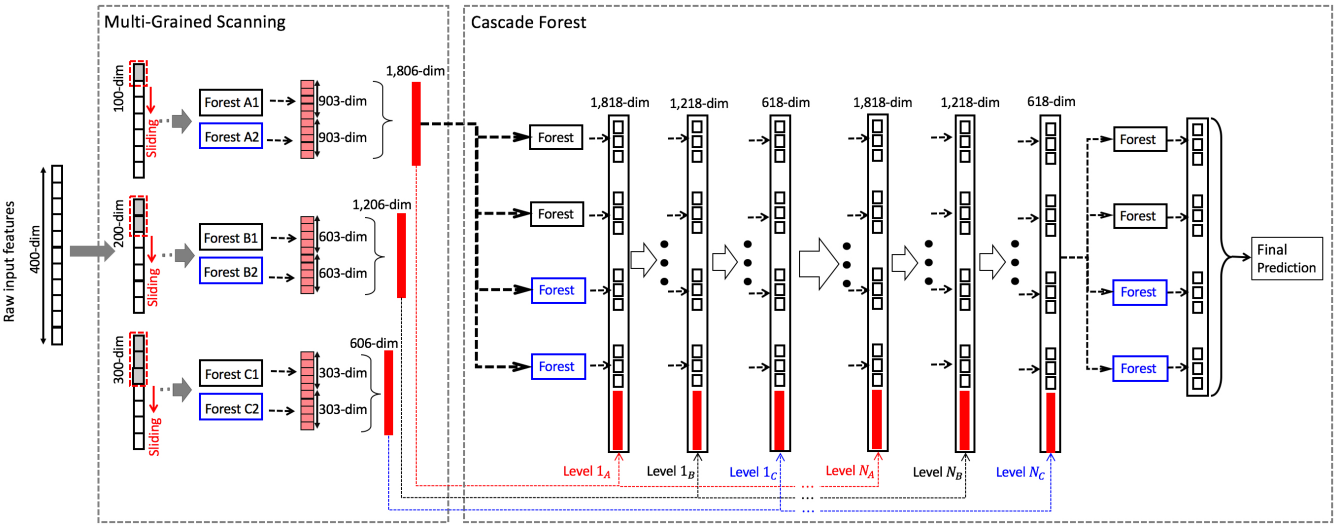

Figure 4: The overall procedure of gcForest. Suppose there are three classes to predict, raw features are 400-dim, and three sizes of sliding windows are used.

Table 1: Summary of hyper-parameters and default settings. Boldfont highlights hyper-parameters with relatively larger influence; "?" indicates default value unknown, or generally requiring different settings for different tasks.

Deep neural networks (e.g., convolutional neural networks)	gcForest
Type of activation functions: Sigmoid, ReLU, tanh, linear, etc. Architecture configurations: **No. Hidden layers**: ? **No. Nodes in hidden layer**: ? **No. Feature maps**: ? **Kernel size**: ? Optimization configurations: **Learning rate**: ? Dropout: {0.25/0.50} **Momentum**: ? **L1/L2 weight regularization penalty**: ? Weight initialization: Uniform, glorot_normal, glorot_uni, etc. Batch size: {32/64/128}	Type of forests: Completely-random tree forest, random forest, etc. Forest in multi-grained scanning: **No. Forests**: {2} **No. Trees in each forest**: {500} Tree growth: till pure leaf, or reach depth 100 **Sliding window size**: $\{\lfloor d/16 \rfloor, \lfloor d/8 \rfloor, \lfloor d/4 \rfloor\}$ Forest in cascade: **No. Forests**: {8} **No. Trees in each forest**: {500} Tree growth: till pure leaf

Table 1 summarizes the hyper-parameters of deep neural networks and gcForest, where the default values used in our experiments are given.

3 Experiments

3.1 Configuration

In this section we compare gcForest with deep neural networks and several other popular learning algorithms. The goal is to validate that gcForest can achieve performance highly competitive to deep neural networks, with easier parameter tuning even across a variety of tasks. Thus, in all experiments gcForest is using the *same* cascade structure: each level consists of 4 completely-random tree forests and 4 random forests, each containing 500 trees, as described in Section 2.1. Three-fold CV is used for class vector generation. The number of cascade levels is automatically determined. In detail, we split the training set into two parts, i.e., growing set and estimating set[1]; then we use the growing set to grow the cascade, and the estimating set to estimate the performance. If growing a new level does not improve the performance, the growth of the cascade terminates and the estimated number of levels is obtained. Then, the cascade is retrained based on merging the growing and estimating sets. For all experiments we take 80% of the training data for growing set and 20% for estimating set. For multi-grained scanning, three window sizes are used. For d raw features, we use feature windows with sizes of $\lfloor d/16 \rfloor$, $\lfloor d/8 \rfloor$, $\lfloor d/4 \rfloor$; if the raw features are with panel structure (such as images), the feature windows are also with panel structure as shown in Figure 3. Note that a careful task-specific tuning may bring better performance; nevertheless, we find that even using the same parameter setting without fine-tuning, gcForest has already been able to achieve excellent performance across a broad range of tasks.

For deep neural network configurations, we use ReLU for activation function, cross-entropy for loss function, adadelta for optimization, dropout rate 0.25 or 0.5 for hidden layers according to the scale of training data. The network structure hyper-parameters, however, could not be fixed across tasks, otherwise the performance will be embarrassingly unsatisfactory. For example, a network attained 80% accuracy on ADULT dataset achieved only 30% accuracy on YEAST with the same architecture (only the number of input/output nodes changed to suit the data). Therefore, for deep neural networks, we examine a variety of architectures on validation set, and pick the one with the best performance, then re-train the whole network on training set and report the test accuracy.

[1]Some experimental datasets are given with training/validation sets. To avoid confusion, here we call the subsets generated from training set as growing/estimating sets.

3.2 Results

Image Categorization

The MNIST dataset [LeCun *et al.*, 1998] contains 60,000 images of size 28 by 28 for training (and validating), and 10,000 images for testing. We compare it with a re-implementation of LeNet-5 (a modern version of LeNet with dropout and ReLUs), SVM with rbf kernel, and a standard Random Forest with 2,000 trees. We also include the result of the Deep Belief Nets reported in [Hinton *et al.*, 2006]. The test results show that gcForest, although simply using default settings in Table 1, achieves highly competitive performance.

Table 2: Comparison of test accuracy on MNIST

gcForest	**99.26%**
LeNet-5	99.05%
Deep Belief Net	98.75% [Hinton *et al.*, 2006]
SVM (rbf kernel)	98.60%
Random Forest	96.80%

Face Recognition

The ORL dataset [Samaria and Harter, 1994] contains 400 gray-scale facial images taken from 40 persons. We compare it with a CNN consisting of 2 conv-layers with 32 feature maps of 3×3 kernel, and each conv-layer has a 2×2 max-pooling layer followed. A dense layer of 128 hidden units is fully connected with the convolutional layers and finally

a fully connected soft-max layer with 40 hidden units is appended at the end. ReLU, cross-entropy loss, dropout rate of 0.25 and adadelta are used for training. The batch size is set to 10, and 50 epochs are used. We have also tried other configurations of CNN, whereas this one gives the best performance. We randomly choose 5/7/9 images per person for training, and report the test performance on the remaining images. Note that a random guess will achieve 2.5% accuracy, since there are 40 possible outcomes. The kNN method here uses $k = 3$ for all cases. The test results show that gcForest runs well across all three cases even by using the same configurations as described in Table 1.

Table 3: Comparison of test accuracy on ORL

	5 image	7 images	9 images
gcForest	**91.00%**	**96.67%**	**97.50%**
Random Forest	91.00%	93.33%	95.00%
CNN	86.50%	91.67%	95.00%
SVM (rbf kernel)	80.50%	82.50%	85.00%
kNN	76.00%	83.33%	92.50%

Music Classification

The GTZAN dataset [Tzanetakis and Cook, 2002] contains 10 genres of music clips, each represented by 100 tracks of 30 seconds long. We split the dataset into 700 clips for training and 300 clips for testing. In addition, we use MFCC feature to represent each 30 seconds music clip, which transforms the original sound wave into a $1,280 \times 13$ feature matrix. Each frame is atomic according to its own nature; thus, CNN uses a 13×8 kernel with 32 feature maps as the conv-layer, each followed by a pooling layer. Two fully connected layers with 1,024 and 512 units, respectively, are appended, and finally a soft-max layer is added in the last. We also compare it with an MLP having two hidden layers, with 1,024 and 512 units, respectively. Both networks use ReLU as activation function and categorical cross-entropy as the loss function. For Random Forest, Logistic Regression and SVM, each input is concatenated into an $1,280 \times 13$ feature vector.

Table 4: Comparison of test accuracy on GTZAN

gcForest	**65.67%**
CNN	59.20%
MLP	58.00%
Random Forest	50.33%
Logistic Regression	50.00%
SVM (rbf kernel)	18.33%

Hand Movement Recognition

The sEMG dataset [Sapsanis *et al.*, 2013] consists of 1,800 records each belonging to one of six hand movements, i.e., spherical, tip, palmar, lateral, cylindrical and hook. This is a time-series dataset, where EMG sensors capture 500 features per second and each record associated with 3,000 features. In addition to an MLP with *input*-1,024-512-*output* structure, we also evaluate a recurrent neural network, LSTM [Gers *et al.*, 2001] with 128 hidden units and sequence length of 6 (500-dim input vector per second).

Table 5: Comparison of test accuracy on sEMG data

gcForest	**71.30%**
LSTM	45.37%
MLP	38.52%
Random Forest	29.62%
SVM (rbf kernel)	29.62%
Logistic Regression	23.33%

Sentiment Classification

The IMDB dataset [Maas *et al.*, 2011] contains 25,000 movie reviews for training and 25,000 for testing. The reviews are represented by *tf-idf* features. This is not image data, and thus CNNs are not directly applicable. So, we compare it with an MLP with structure *input*-1,024-1,024-512-256-*output*. We also include the result reported in [Kim, 2014], which uses CNNs facilitated with word embeding. Considering that *tf-idf* features do not convey spacial or sequential relationships, we skip multi-grained scanning for gcForest.

Table 6: Comparison of test accuracy on IMDB

gcForest	**89.16%**
CNN	89.02% [Kim, 2014]
MLP	88.04%
Logistic Regression	88.62%
SVM (linear kernel)	87.56%
Random Forest	85.32%

Low-Dimensional Data

We also evaluate gcForest on UCI-datasets [Lichman, 2013] with relatively small number of features: LETTER with 16 features and 16,000/4,000 training/test examples, ADULT with 14 features and 32,561/16,281 training/test examples, and YEAST with only 8 features and 1,038/446 training/test examples. Fancy architectures like CNNs could not work on such data as there are too few features without spatial relationship. So, we compare it with MLPs. Unfortunately, although MLPs have less configuration options than CNNs, they are still very tricky to set up. For example, MLP with *input*-16-8-8-*output* structure and ReLU activation achieve 76.37% accuracy on ADULT but just 33% on LETTER. We conclude that there is no way to pick one MLP structure which gives decent performance across all datasets. Therefore, we report different MLP structures with the best performance: for LETTER the structure is *input*-70-50-*output*, for ADULT is *input*-30-20-*output*, and for YEAST is *input*-50-30-*output*. In contrast, gcForest uses the same configuration as before, except that the multi-grained scanning is abandoned considering that the features of these small-scale data do not hold spacial or sequential relationships.

Table 7: Comparison of test accuracy on low-dim data

	LETTER	ADULT	YEAST
gcForest	**97.40%**	**86.40%**	**63.45%**
Random Forest	96.50%	85.49%	61.66%
MLP	95.70%	85.25%	55.60%

3.3 Influence of Multi-Grained Scanning

To study the separate contribution of the cascade forest structure and multi-grained scanning, Table 8 compares gcForest with cascade forest on MNIST, GTZAN and sEMG datasets. It is evident that when there are spacial or sequential feature relationships, the multi-grained scanning process helps improve performance apparently.

Table 8: Results of gcForest w/wo multi-grained scanning

	MNIST	GTZAN	sEMG
gcForest	99.26%	65.67%	71.30%
CascadeForest	98.02%	52.33%	48.15%

3.4 Running time

Our experiments use a PC with 2 Intel E5 2695 v4 CPUs (18 cores), and the running efficiency of gcForest is good. For example, for IMDB dataset (25,000 examples with 5,000 features), it takes 267.1 seconds per cascade level, and automatically terminates with 9 cascade levels, amounting to 2,404 seconds or 40 minutes. In contrast, MLP compared on the same dataset requires 50 epochs for convergence and 93 seconds per epoch, amounting to 4,650 seconds or 77.5 minutes for training; 14 seconds per epoch (with batch size of 32) if using GPU (Nvidia Titan X pascal), amounting to 700 seconds or 11.6 minutes. Multi-grained scanning will increase the cost of gcForest; however, the different grains of scanning are inherently parallel. Also, both completely-random tree forests and random forests are parallel ensemble methods [Zhou, 2012]. Thus, the efficiency of gcForest can be improved further with optimized parallel implementation. Note that the training cost is controllable because users can set the number of grains, forests, trees by considering computational cost available. It is also noteworthy that the above comparison is somewhat unfair to gcForest, because many different architectures have been tried for neural networks to achieve the reported performance but these time costs are not included.

4 Related Work

The gcForest is a decision tree ensemble approach. Ensemble methods [Zhou, 2012] are a kind of powerful machine learning techniques which combine multiple learners for the same task. Actually there are some studies showing that by using ensemble methods such as random forest facilitated with deep neural network features, the performance can be even better than simply using deep neural networks [Kontschieder *et al.*, 2015]. Our purpose of using ensemble, however, is quite different. We are aiming at an alternative to deep neural networks rather than a combination with deep neural networks. In particular, by using the cascade forest structure, we hope not only to do representation learning, but also to decide a suitable model complexity automatically.

The multi-grained scanning procedure uses different sizes of sliding windows to examine the data; this is somewhat related to wavelet and other multi-resolution examination procedures [Mallat, 1999]. For each window size, a set of instances are generated from one training example; this is related to bag generators [Wei and Zhou, 2016] of multi-instance learning [Dietterich *et al.*, 1997]. In particular, the bottom part of Figure 3, if applied to images, can be regarded as the *SB* image bag generator [Maron and Lozano-Pérez, 1998; Wei and Zhou, 2016].

The cascade procedure is related to Boosting [Freund and Schapire, 1997], which is able to automatically decide the number of learners in ensemble, and particularly, a cascade boosting procedure [Viola and Jones, 2001] has achieved great success in object detection tasks. Note that when multiple grains are used, each cascade level of gcForest consists of multiple grades; this is actually a cascade of cascades. Each grade can be regarded as an ensemble of ensembles; in contrast to previous studies such as using Bagging as base learners for Boosting [Webb, 2000], gcForest uses the ensembles in the same grade together for feature re-representation. Passing the output of one grade of learners as input to another grade of learners is related to stacking [Wolpert, 1992; Breiman, 1996]. Based on suggestions from studies about stacking [Ting and Witten, 1999; Zhou, 2012], we use cross-validation procedure to generate inputs from one grade for the next. Note that stacking is easy to overfit with more than two grades, and could not enable a deep model by itself.

To construct a good ensemble, it is well known that individual learners should be accurate and diverse, yet there is no well accepted formal definition of diversity [Kuncheva and Whitaker, 2003; Zhou, 2012]. Thus, researchers usually try to enhance diversity heuristically, such as what we have done by using different types of forests in each grade. Actually, gcForest exploits all the four major categories of diversity enhancement strategies [Zhou, 2012]. In particular, when assigning the label of the original instance to all instances generated by sliding windows, as shown in Figure 3, some label assignments are inherently incorrect; this is related to the Flipping Output method [Breiman, 2000], a representative of output representation manipulation for diversity enhancement.

As a tree-based approach, gcForest could be potentially easier for theoretical analysis than deep neural networks, although this is beyond the scope of this paper. Indeed, some recent theoretical studies about deep learning, e.g., [Mhaskar *et al.*, 2017], seem more intimate with tree-based models.

5 Conclusion

By recognizing that the key of deep learning lies in the representation learning and large model capacity, in this paper we attempt to endow such properties to tree ensembles and propose the gcForest method. Comparing with deep neural networks, gcForest achieves highly competitive performance in experiments. More importantly, gcForest has much fewer hyper-parameters, and in our experiments excellent performance is obtained across various domains by using the same parameter setting. The code of gcForest is available [2].

There are other possibilities to construct deep forest. As a seminal study, we have only explored a little in this direction. In order to tackle complicated tasks, it is likely that learning models have to go deep. Current deep models, however, are always neural networks. This paper illustrates how to construct deep forest, and we believe it may open a door towards alternative to deep neural networks for many tasks.

[2] http://lamda.nju.edu.cn/code_gcForest.ashx

References

[Breiman, 1996] L. Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996.

[Breiman, 2000] L. Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):113–120, 2000.

[Breiman, 2001] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Cho *et al.*, 2014] K. Cho, B. van Meriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[Dietterich *et al.*, 1997] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[Freund and Schapire, 1997] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[Gers *et al.*, 2001] F. A. Gers, D. Eck, and J. Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *ICANN*, pages 669–676, 2001.

[Goodfellow *et al.*, 2016] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[Graves *et al.*, 2013] A. Graves, A. R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649, 2013.

[Hinton *et al.*, 2006] G. E. Hinton, S. Osindero, and Y.-W. Simon. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[Hinton *et al.*, 2012] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[Kim, 2014] Y. Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882*, 2014.

[Kontschieder *et al.*, 2015] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep neural decision forests. In *ICCV*, pages 1467–1475, 2015.

[Krizhenvsky *et al.*, 2012] A. Krizhenvsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012.

[Kuncheva and Whitaker, 2003] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[LeCun *et al.*, 1998] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Lichman, 2013] M. Lichman. UCI machine learning repository, 2013.

[Liu *et al.*, 2008] F. T. Liu, K. M. Ting, Y. Yu, and Z.-H. Zhou. Spectrum of variable-random trees. *Journal of Artificial Intelligence Research*, 32:355–384, 2008.

[Maas *et al.*, 2011] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, 2011.

[Mallat, 1999] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, London, UK, 2nd edition, 1999.

[Maron and Lozano-Pérez, 1998] O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. In *NIPS*, pages 570–576. 1998.

[Mhaskar *et al.*, 2017] H. Mhaskar, Q. Liao, and T. A. Poggio. When and why are deep networks better than shallow ones? In *AAAI*, pages 2343–2349, 2017.

[Samaria and Harter, 1994] F. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *2nd IEEE Workshop on Applications of Computer Vision*, pages 138–142, 1994.

[Sapsanis *et al.*, 2013] C. Sapsanis, G. Georgoulas, A. Tzes, and D. Lymberopoulos. Improving EMG based classification of basic hand movements using EMD. In *35th Annual International Conference on the IEEE Engineering in Medicine and Biology Society*, pages 5754–5757, 2013.

[Simonyan and Zisserman, 2014] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[Ting and Witten, 1999] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[Tzanetakis and Cook, 2002] G. Tzanetakis and P. R. Cook. Musical genre classification of audio signals. *IEEE Trans. Speech and Audio Processing*, 10(5):293–302, 2002.

[Viola and Jones, 2001] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, pages 511–518, 2001.

[Webb, 2000] G. I. Webb. MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

[Wei and Zhou, 2016] X.-S. Wei and Z.-H. Zhou. An empirical study on image bag generators for multi-instance learning. *Machine Learning*, 105(2):155–198, 2016.

[Wolpert, 1992] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

[Zhou, 2012] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC, Boca Raton, FL, 2012.