Deep Forest: Towards An Alternative to Deep Neural Networks

Zhi-Hua Zhou and **Ji Feng**
National Key Laboratory 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 contrast to deep neural networks which require great effort in hyper-parameter tuning, gcForest is much easier to train. Actually, even when gcForest is applied to different data from different domains, excellent performance can be achieved by almost same settings of hyper-parameters. The training process of gcForest is efficient and scalable. In our experiments its training time running on a PC is comparable to that of deep neural networks running with GPU facilities, and the efficiency advantage may be more apparent 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. Moreover, as a tree-based approach, gcForest should be easier for theoretical analysis than deep neural networks.

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 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 expensive labeling cost, leading to inferior performance of deep neural networks on 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 not only makes the training of deep neural networks very tricky, like an art rather than science/engineering, but also makes 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 form 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. It is noteworthy that 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 not only makes the training of gcForest convenient, but also makes theoretical analysis, although beyond the scope of this paper, 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 or even better performance than deep neural networks, whereas the training time cost of gcForest running on a PC is comparable to that of deep neural networks running with GPU facilities. Note that the efficiency advantage can be more apparent be-

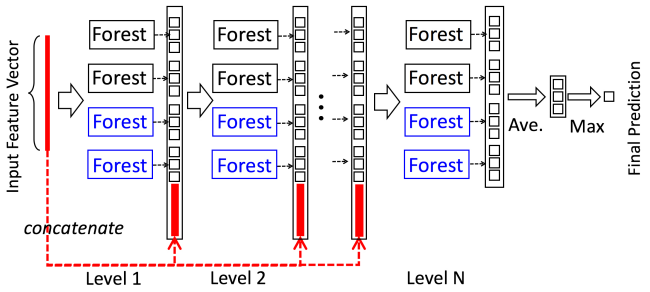

Figure 1: Illustration of the *cascade forest* structure. Each level of the cascade consists of two random forests (blue) and two complete-random tree forests (black). Suppose there are three classes to predict; thus, each forest will output a three-dimensional class vector, which is then concatenated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

cause gcForest is naturally apt to parallel implementation.

We believe that to tackle complicated learning 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 it 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

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, in our implementation we use two complete-random tree forests and two random forests [Breiman, 2001]. Each complete-random tree forest contains 1,000 complete-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 or no more than 10 instances. Similarly, each random forest contains 1,000 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

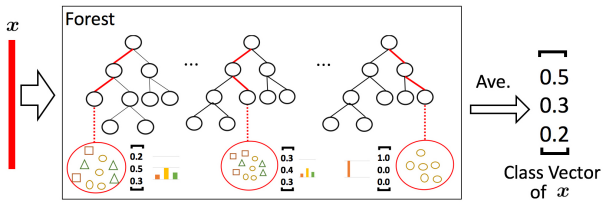

Figure 2: Illustration of class vector generation. Different marks in leaf nodes imply different classes.

class of training examples at the leaf node where the concerned instance falls into, and then averaging across all trees in the same forest, as illustrated in Figure 2, where red 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 next level of cascade. Note that 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 illustrated, 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 panels). All feature vectors extracted from positive/negative training examples are regarded as positive/negative instances; they will be then 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 complete-random tree forest and

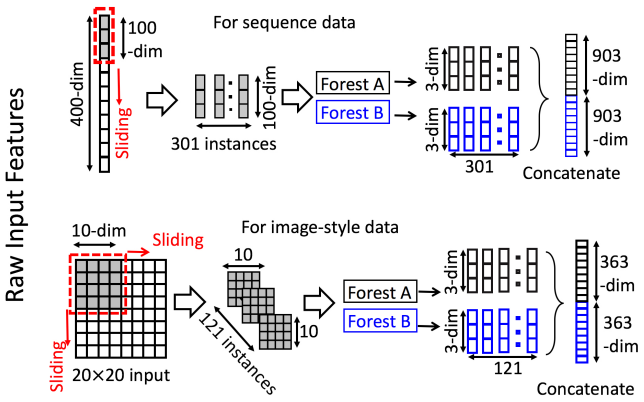

Figure 3: Illustration of feature re-representation using sliding window scanning. Suppose there are three classes, raw features are 400-dim, and sliding window is 100-dim.

a random forest, and then the class vectors are generated and concatenated as transformed features. As Figure 3 illustrated, suppose 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.

Figure 3 shows only one size of sliding window. By using multiple sizes of sliding windows, the final transformed feature vectors will include more features, as shown in Figure 4.

2.3 Overall Procedure and Hyper-Parameters

Figure 4 summarizes the overall procedure of gcForest. Suppose the original input is of 400 raw features, and three window sizes are used for multi-grained scanning. The window with size of 100 features will generate 301 100-dimensional feature vectors from each original training example; if there are m training examples, this sliding window will generate a data set of $301 \times m$ 100-dimensional training examples. These data will be used to train a complete-random tree forest and a random forest, each containing 30 trees. Suppose there are three classes to be predicted; each forest will produce a 3-dimensional class vector for each of the 301 instances. After concatenating all class vectors generated by the two forests, a 1,806-dimensional feature vector is obtained.

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. By concatenating them, a 3,618-dimensional feature vector is obtained, which is the re-representation for the raw feature vector by multi-grained scanning. In other words, each original 400-dimensional feature vector is re-represented by a 3,618-dimensional feature vector.

After that, the m number of 3,618-dimensional feature vectors will be passed to the cascade. As introduced in Section 2.1, if each level consists of four forests, 3,630-dimensional feature vectors will be obtained at the end of the first level of the cascade. These 3,630-dimensional feature vectors will then be input to the next level of the cascade; this process repeats until the validation performance suggests that the expanding of the cascade should be terminated.

Given a test instance, it will go through the multi-grained scanning procedure to get its corresponding 3,618-dimensional 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.

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 purpose 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 are using the *same* cascade structure: each level consists of 2 complete-random tree forests and 2 random forests, each containing 1,000 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 $\lfloor d/16 \rfloor$, $\lfloor d/9 \rfloor$, $\lfloor d/4 \rfloor$; 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 tuning may lead to better performance; however, we have not got sufficient computational resource and time to finely tuned the hyper-parameters. Nevertheless, we find that even using the same parameter setting without fine-tuning, gcForest has already been able to achieve excellent performance.

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 set across tasks, otherwise the performance will be embarrassingly unsatisfying. For example, a network achieved 80% accuracy on ADULT dataset, but the same architecture (only the number of input/output nodes are changed to suit the data) only achieved 30% accuracy on YEAST dataset. Therefore, for deep neural

[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.

Table 1: Summary of hyper-parameters and default settings. Boldfont highlights hyper-parameters with relatively larger influence; "?" indicates default value unknown.

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_uniform, etc. Batch size: {32/64/128}	Type of forests: Complete random tree forest, random forest, etc. Forest in multi-grained scanning: **No. Forests**: {2} **No. trees in each forest**: {30} Tree growth: till pure leaf, or $\leq$ 20 instances **Sliding window size**: $\{\lfloor d/16 \rfloor, \lfloor d/9 \rfloor, \lfloor d/4 \rfloor\}$ Forest in cascade: **No. Forests**: {4} **No. trees in each forest**: {1,000} Tree growth: till pure leaf, or $\leq$ 10 instances

networks, we examine a variety of architectures on validation set, pick the one with the best performance, and then re-train the whole network on training set and report the test accuracy.

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 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 are summarized in Table 2, showing that gcForest, although simply using default settings in Table 1, achieves highly competitive performance.

Table 2: Comparison of test accuracy on MNIST

DNN (LeNet-5)	99.05%
gcForest	**98.96%**
DNN (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 with a CNN consisting of 2 conv-layers with 32 feature maps of 3×3 kernel, each conv-layer has a 2×2 max-pooling layer followed. A dense layer of 128 hidden unit is fully connected with the convolutional layers and finally a fully connected soft-max layer with 40 hidden units is appended at

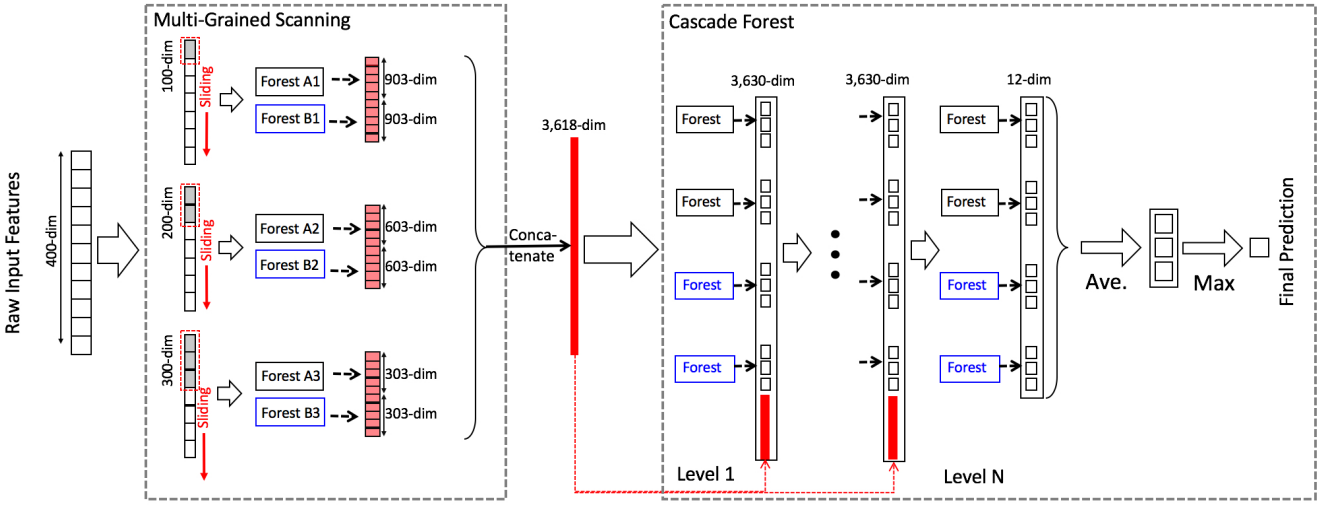

Figure 4: The overall procedure of gcForest. Suppose there are three classes to predict, raw features are 400-dim, and three sizes of sliding windows are used.

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 1/5/9 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 $k = 1$ for one training sample and 3 for the rest cases. The test results are summarized in Table 3.

Table 3: Comparison of test accuracy on ORL

	1 image	5 images	9 images
gcForest	**63.06%**	**94.25%**	**98.30%**
Random Forest	61.70%	91.20%	97.00%
DNN (CNN)	3.30%	86.50%	92.50%
SVM (rbf kernel)	57.90%	78.95%	82.50%
kNN	19.40%	77.60%	90.50%

Given one training sample per person, CNN performs poorly due to the lack of training samples.[2] gcForest runs well across all three cases even by using as same configurations as that used in Table 2.

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 used MFCC feature to represent each 30 seconds music clip, which transforms the original sound wave into a 1280×13 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 1024 and 512 units, respectively, are appended, and finally a soft-max layer is added in the last. We also compare with a MLP having two hidden layers, with 1024 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 a 1280×13 feature vector. The test results are summarized in Table 4.

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 belongs to one of six hand movements (spherical, tip, palmar, lateral, cylindrical and hook). This is a time-series data, where EMG sensors capture 500 features per second and each record is associated with 3,000 features. In addition to an MLP with *input*-1024-512-*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). The test results are summarized in Table 5.

Table 5: Comparison of test accuracy on sEMG data

gcForest	**55.93%**
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 reviews 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 with a MLP with structure *input*-1024-1024-512-256-*output*. We also include the result reported in [Kim, 2014], which uses CNNs facilitated with word embeding. Considering that the *tf-idf* features do not convey spacial or sequential relationships, we do not run multi-grained scanning for gcForest. The test accuracy is summarized in Table 6.

Table 6: Comparison of test accuracy on IMDB

gcForest	**89.32%**
DNN (CNN)	89.02% [Kim, 2014]
DNN (MLP)	88.04%
Logistic Regression	88.62%
SVM (linear kernel)	87.56%
Random Forest	85.32%

Small-Scale 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. It is evident that fancy architectures like CNNs could not work on such data as there are too few features without spatial relationship. So, we compare 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 achieved 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 for all datasets. Therefore, we report different MLP structures with the best performance we can come up with: 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. The test results are summarized in Table 7.

[2]There are studies where CNNs perform excellently for face recognition, by using huge amount of face images to train the model. Here, we simply use the training data.

Table 7: Comparison of test accuracy on small-scale data

	LETTER	ADULT	YEAST
gcForest	**97.25%**	**86.17%**	**63.23%**
Random Forest	96.50%	85.49%	61.66%
MLP	95.70%	85.25%	55.60%

3.3 Influence of Multi-Grained Scanning

The gcForest includes two important components, i.e., cascade forest and multi-grained scanning. To study their separate contributions, Table 8 compares gcForest and 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 significantly.

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

	MNIST	GTZAN	sEMG
gcForest	98.96%	66.00%	55.93%
CascadeForest	97.85%	52.00%	46.11%

3.4 Running time

In our experiments we use a PC with 2 Intel E5 2670 v3 CPUs (24 cores), and the running efficiency of gcForest is quite good. For example, for IMDB dataset (25,000 examples with 5,000 features), it takes 269.5 seconds per cascade level (4 forests with 3-fold growing), and automatically terminates with 4 cascade levels, amounting to 1,078 seconds or 17.9 minutes. In contrast, for MLP described in the previous section on the same dataset, using the most powerful GPU in the deep learning market (Nvidia Titan X pascal) requires 14 seconds per epoch (with batch size of 32), and 50 epochs are required for convergence, amounting to 700 seconds or 11.6 minutes for training; 93 seconds per epoch if using CPUs, amounting to 4,650 seconds or 77.5 minutes. Note that both complete-random tree forests and random forests are parallel ensemble methods [Zhou, 2012]. Thus, the efficiency advantage of gcForest can be more apparent with large-scale parallel implementation.

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 [Peter *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 procedure of passing the output of one level of learners as input to another level 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 level for the next. The cascade procedure is also related to Boosting [Freund and Schapire, 1997], which is able to automatically decide the number of learners in the ensemble, and particularly, a cascade boosting procedure [Viola and Jones, 2001] has achieved great success in object detection tasks. Each level of our cascade 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 level together for feature re-representation. To construct a good ensemble, it is well known that the individual learners should be accurate and diverse. However, until now there is no well accepted formal definition of diversity [Kuncheva and Whitaker, 2003; Zhou, 2012], and thus, researchers usually try to increase diversity heuristically, such as we use different types of forests in each level of our cascade forest structure.

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 will be 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].

5 Conclusion

With the recognition 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 achieved highly competitive or even better performance in our experiments. More importantly, gcForest has much fewer hyper-parameters and is less sensitive to parameter setting; actually in our experiments excellent performance are obtained across various domains by using the same parameter setting, and it can work well no matter on large-scale or small-scale data. Moreover, as a tree-based approach, gcForest should be easier for theoretical analysis than deep neural networks, although this is beyond the scope of this paper. The code of gcForest will be available soon.

There are other possibilities to construct deep forest. As a seminar study, we have only explored a little in this direction. If we had stronger computational facilities, we would like to try big data and deeper forest, which is left for future work. In principle, deep forest should be able to exhibit other powers of deep neural networks, such as serving as feature extractor or pre-trained model. It is worth mentioning that 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.

Acknowledgements: Authors want to thank Songcan Chen, Xin Geng, Sheng-Jun Huang, Chao Qian, Jianxin Wu, Yang

Yu, De-Chuan Zhan, Lijun Zhang, Min-Ling Zhang for reading a draft of the paper.

References

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

[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 *2014 Conference on Empirical Methods in Natural Language Processing*, 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 *International Conference on Artificial Neural Networks*, 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 *38th IEEE International Conference on Acoustics, Speech and Signal Processing*, 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.

[Krizhenvsky *et al.*, 2012] A. Krizhenvsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, 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 *49th Annual Meeting of the Association for Computational Linguistics*, 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 *Advances in Neural Information Processing Systems 10*, pages 570–576. 1998.

[Peter *et al.*, 2015] K. Peter, F. Madalina, C. Antonio, and R. Samuel. Deep neural decision forests. In *IEEE International Conference on Computer Vision*, pages 1467–1475, 2015.

[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 *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 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.