AutoEncoder by Forest[*]

Ji Feng, Zhi-Hua Zhou
National Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210023, China
Collaborative Innovation Center of Novel Software Technology
and Industrialization, Nanjing 210023, China
{fengj, zhouzh}@lamda.nju.edu.cn

Abstract

Auto-encoding is an important task which is typically realized by deep neural networks (DNNs) such as convolutional neural networks (CNN). In this paper, we propose EncoderForest (abbrv. eForest), the first tree ensemble based auto-encoder. We present a procedure for enabling forests to do backward reconstruction by utilizing the Maximal-Compatible Rule (MCR) defined by the decision paths of the trees, and demonstrate its usage in both supervised and unsupervised setting. Experiments show that, compared with DNN based auto-encoders, eForest is able to obtain lower reconstruction error with fast training speed, while the model itself is reusable and damage-tolerable.

Introduction

Auto-encoder (Vincent et al. 2010) is a class of models which aim to map the input to a latent space and map it back to the original space, with low reconstruction error as its objective. Previous approaches for building such device mainly came from the neural network community. For instance, a neural network based auto-encoder usually consists of an encoder and a decoder. The encoder maps the input to a hidden layer and the decoder maps it back to the input space. By concatenating the two parts and setting the reconstruction error as learning objective, back-propagation can be used for training such models. It is widely used for dimensionality reduction (Hinton, Osindero, and Simon 2006), representation learning (Bengio, Courville, and Vincent 2013), as well as some more recent works for generative tasks such as Variational Auto-encoders (Kingma and Welling 2013).

Ensemble learning (Zhou 2012) is a powerful learning paradigm which trains multiple learners and combines to tackle the problem. It is widely used in a broad range of tasks and has demonstrated great performance. Tree ensemble methods, or forests, such as Random Forest (Breiman 2001), for instance, is one of the best off-the-shelf methods for supervised learning (Fernández-Delgado et al. 2014). Other successful tree ensembles such as gradient based decision trees (GBDTs) also proven its ability during the past decade(Friedman 2001; Chen and Guestrin 2016). Besides supervised learning, tree ensembles have also achieved great success in other tasks, such as isolation forest (Liu, Ting, and Zhou 2008) which is an efficient unsupervised method for anomaly detection. Recently, deep model based on forests has also been proposed (Zhou and Feng 2017), and demonstrated competitive performance with DNNs across a broad range of tasks with much fewer hyper-parameters.

In this paper, we present the EncoderForest (abbrv. eForest), the first tree ensemble based auto-encoder model. Concretely, the eForest enables a tree ensemble to perform forward encoding and backward decoding thus making it possible to construct an auto-encoder by forest. In addition, the eForest model can be trained in both supervised or unsupervised fashion and experimental results showed the eForest approach has the following advantages:

- Accurate: Its experimental reconstruction error is lower than a MLP or CNN based auto-encoders.
- Efficient: eForest on a single KNL (many-core CPU) runs even faster than a CNN auto-encoder runs on a Titan-X GPU for training.
- Damage-tolerable: The trained model works well even when it is partially damaged.
- Reusable: A model trained from one dataset can be directly applied on the other datasets in the same domain.

The rest of the paper is organized as follows: first we introduce related works, followed by the proposed eForest model, then experimental results are presented, finally conclusion and future works are discussed.

Related Work

Auto-encoder is an important class of models for representation learning, and is one of the key ingredients of deep learning. (Goodfellow, Bengio, and Courville 2016; Bengio, Courville, and Vincent 2013). The study of auto-encoder dates back to Bourlard and Kamp (1988), of which the goal is to learning association from data. Most of the previous approaches on building such devices are neural network based methods. For instance, the under-complete auto-encoder is designed for dimensionality reduction (Hinton and Salakhutdinov 2006), sparse auto-encoder, on the other hand, gives a sparsity penalty on the activation layer (Hinton and Ranzato 2010), which is related with sparse coding

[*]This research was supported by NSFC (61333014) and 973 Program (2014CB340501).

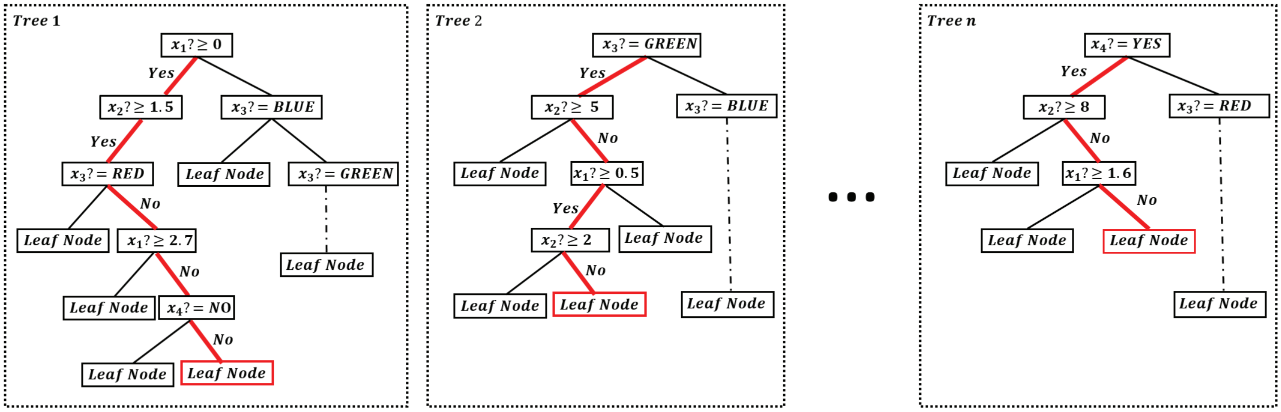


Figure 1: Traversing backward along decision paths

(Willmore and Tolhurst 2001), and de-noising auto-encoders (Bengio et al. 2013) forces the model to learn the mapping from a corrupted input to its noiseless version. Applications of auto-encoders ranging from computer vision (Masci et al. 2011) to natural language processing (Mikolov et al. 2013) and semantic hashing for information retrieval tasks(Ruslan, A. Mnih, and Hinton 2007). In fact, the recent hot wave of deep learning started with training a stack of auto-encoders in a greedy layer-wised fashion. (Hinton, Osindero, and Simon 2006).

Ensembles of decision trees, or called *forest*, are popularly used in ensemble learning (Zhou 2012). For example, Bagging (Breiman 1996) and Boosting (Freund and Schapire 1999) usually take decision trees as component learners. Other famous decision tree ensemble methods include Random Forest (Breiman 2001) and GBDT(Friedman 2001); the former is a variant of Bagging, whereas the latter is a variant of Boosting. Some efficient implementations of GBDT, e.g., XGBoost (Chen and Guestrin 2016), has been widely used in industry and various data science competitions. In addition to the above tree ensembles constructed in supervised setting, there are unsupervised tree ensembles also proven to be useful in various domains. For example, the iForest (Liu, Ting, and Zhou 2008) is an unsupervised forest designed for anomaly detection, and its ingredient, completely-random decision trees, have also been applied to tasks such as streaming new class learning (Mu, Ting, and Zhou in press). Note that both supervised and unsupervised forests, i.e., Random Forest and completely-random tree forest, have been simultaneously exploited in the construction of deep forest (Zhou and Feng 2017).

The Proposed Method

An auto-encoder has two basic functions: encoding and decoding. There is no difficulty for a forest to do encoding tasks, since at least the leaf nodes information can be regarded as one kind of encoding; needless to say, the subsets of nodes or even the branch of paths may be able to offer more information for encoding.

First, we propose the encoding procedure of EncoderForest. Given a trained tree ensemble model of T trees, the forward encoding procedure takes an input data and send this data to each root node of trees in the ensemble, once the data traverse down to the leaf nodes for all trees, the procedure will return a T dimensional vector, where each element t is an integer index of the leaf node in tree i where $i \in \{1, 2, ..T\}$. A more concrete algorithm for forward encoding is shown in Algorithm 1. Notice that this encoding procedure is independent with the particular learning rule on how to split the nodes for trees. For instance, the decision rule can be learned in a supervised setting such as random forest, or can be learned in an unsupervised setting such as completely random tree forest.

Algorithm 1: Forward Encoding

```
Input: A trained forest $F$ with $T$ trees,
       An input data $x$
Output: $x_{enc}$
$x_{enc} \leftarrow zeros[T,1]$ % initialize $x_{enc}$
for $i$ in range($T$) do
    $x_{enc}[i] \leftarrow F.tree[i].get\_leaf\_index(x)$
end
return $x_{enc}$
```

On the other hand, the decoding function is not that obvious. In fact, forests are generally used for forward prediction, by going from the root of each tree to the leaves, whereas it is unknown how to do backward reconstruction, i.e., inducing the original samples from information obtained at the leaves.

For a setup, assume we are handling a binary classification task, with four attributes. The first and second attributes are numerical ones; the third is a triple-valued attribute with values *RED*, *BLUE*, *GREEN*; the fourth is a boolean attribute with values *YES*, *NO*. Given an instance $\boldsymbol{x}$, let x_i denotes the value of $\boldsymbol{x}$ on the i-th attribute. Suppose in the encoding step we have generated a forest as shown in Fig 1. Given the forest, assume we only know the leaf nodes on which the instance $\boldsymbol{x}$ falling into, as shown in Fig 1 as the red nodes, and wish to reconstruct $\boldsymbol{x}$.

Here, we propose an effective yet simple, possibly the simplest, strategy for backward reconstruction in forests to achieve the above goal.

Firstly, each leaf node actually corresponds to a path coming from the root, we can identify the path based on the leaf node without uncertainty. For example, in Fig 1 the identified paths are highlighted in red.

Secondly, each path corresponds to a symbolic rule. For example, the highlighted tree paths correspond to the following rule set, where $RULE_i$ corresponds to the path of the i-th tree in the forest, where $\neg$ denotes the negation of a judgment:

$RULE_1$: $(x_1 \geq 0) \wedge (x_2 \geq 1.5) \wedge \neg(x_3 == RED) \wedge \neg(x_1 \geq 2.7) \wedge \neg(x_4 == NO)$

$RULE_2$: $(x_3 == GREEN) \wedge \neg(x_2 \geq 5) \wedge (x_1 \geq 0.5) \wedge \neg(x_2 \geq 2)$

...

$RULE_n : (x_4 == YES) \wedge \neg(x_2 \geq 8) \wedge \neg(x_1 \geq 1.6)$

This rule set can be further adjusted into a more succinct form:

$RULE'_1 : (2.7 \geq x_1 \geq 0) \wedge (x_2 \geq 1.5) \wedge \neg(x_3 == RED) \wedge (x_4 == YES)$

$RULE'_2 : (x_1 \geq 0.5) \wedge \neg(x_2 \geq 2) \wedge (x_3 == GREEN)$

...

$RULE'_n : \neg(x_1 \geq 1.6) \wedge \neg(x_2 \geq 8) \wedge (x_4 == YES)$

Thirdly, we can derive the Maximal-Compatible Rule (MCR). MCR is such a rule that each of its component coverage cannot be enlarged, otherwise incompatible issue will occur. For example, from the above rule set we can get the following corresponding MCR:

$(1.6 \geq x_1 \geq 0.5) \wedge (2 \geq x_2 \geq 1.5) \wedge (x_3 == GREEN) \wedge (x_4 == YES)$

For each component of this MCR, such as $(2 \geq x_2 \geq 1.5)$, its coverage cannot be enlarged; for example, if it were enlarged to $(3 \geq x_2 \geq 1.5)$, it would have conflict with the condition in $\neg(x_2 \geq 2)$ in $RULE_2$. A more detailed description is shown in Algorithm 2.

It is very easy to prove the following theorem, and thus we omit the proof.

Theorem 1. *The original sample must reside in the input region defined by the MCR.*

Finally, after obtaining the MCR, we can reconstruct the original sample. For categorical attributes such as x_3 and x_4, the original sample must take these values in the MCR; for numerical attributes, such as x_2, we can take a representative value, such as the mean value in the interval $[1.5, 2]$. Thus, the reconstructed sample is $\boldsymbol{x}$ = [0.55, 1.75, GREEN, YES]. Note that for numerical value, we can have many alternative ways for the reconstruction, such as the median, max, min, or even calculate the histograms.

```
Algorithm 2: Calculate MCR
  Input: A path rule list $Rule\_List$ consists of $T$
         rules defined by a forest with $T$ trees
  Output: $MCR$
  $MCR \leftarrow initialize\_list()$
  for $i$ in $range(T)$ do
      $path\_rule \leftarrow rule\_list[i]$
      for $node\_rule$ in path_rule.node_rule_list do
          $j \leftarrow node\_rule.get\_attribute()$
          $bound \leftarrow node\_rule.get\_bound()$
          $MCR[j] \leftarrow intersect(MCR[j], bound)$
      end
  end
  return $MCR$
```

Given the above description, here we give a summary for conducting backward decoding of eForest. Concretely, given a trained forest with T trees along with the forward encoding x_{enc} in R^T for a particular data, the backward decoding will first locate the individual leaf node via each element in x_{enc}, and then obtain T decision rules for the corresponding decision paths. Then, by calculating the MCR and taking a point estimate, we can thus get a reconstruction from x_{enc} back to x_{dec} in the input region. A concrete algorithm is shown in Algorithm 3.

```
Algorithm 3: Backward Decoding
  Input: $x_{enc}$, trained eForest $F$ with $T$ trees
  Output: $x_{dec}$
  rule_list $\leftarrow initialize\_list()$
  for $i$ in range(T) do
      $path \leftarrow$ F.tree[i].$get\_path(x_{enc}[i]$ )
      $path\_rule \leftarrow calculate\_rules(path)$
      $path\_rule \leftarrow simplify(path\_rule)$
      $rule\_list.append(path\_rule)$
  end
  $MCR \leftarrow calculate\_MCR(rule\_list)$
  $x_{dec} \leftarrow sample(MCR$,method='minimum')
  return $x_{dec}$
```

By enabling the eForest to conduct the forward encoding and backward decoding operations, auto-encoding task can thus be realized. In addition, although beyond the scope of this paper, the eForest model might give some insights on a theoretical treatment for the representation learning ability for tree ensemble models, as well as helping to design new models for deep forest.

Experiments

Image Reconstruction

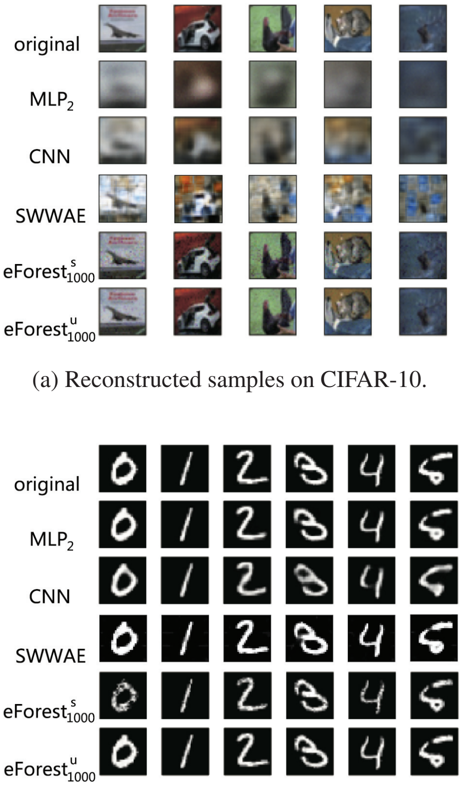
We evaluate the performance of eForest in both supervised and unsupervised setting. In this implementation, we take Random Forest (Breiman 2001) to construct the supervised forest, whereas take the completely-random forest (Zhou and Feng 2017) as the routine for the unsupervised forest. Concretely, for supervised eForest, each non-terminal node randomly select $\sqrt{d}$ attributes in the input space and pick the best possible split for information gain; for unsupervised eForest, each non-terminal node randomly pick one attributes and make a random split. In our experiments we simply grow the trees to pure leaf, or terminate when there are only two instances in a node. We evaluate eForest containing 500 trees or 1,000 trees, denoted by $eForest_{500}$ and $eForest_{1000}$ respectively. Note that $eForest_N$ will re-represent the input instance as a N-dimensional vector.

Since auto-encoders especially DNN-based auto-encoders are mainly designed for image tasks, in this section we run some experiments on image data first. We use the MNIST dataset (LeCun et al. 1998), which consists of 60,000 gray scale 28×28 images for training and 10,000 for testing. We also use CIFAR-10 dataset (Krizhevsky 2009), which is a more complex dataset consists of 50,000 colored 32×32 images for training and 10,000 colored images for testing. For colored images, the eForest process each channel separately for memory saving. During decoding, we just take the min value of the interval defined by the corresponding MCR as indicated in the last sampling step of decoding.

Table 1: Performance comparison (measured by MSE). The subscript s and u denote supervised and unsupervised, respectively.

	MNIST	CIFAR-10
MLP_1	266.85	1284.98
MLP_2	163.97	1226.52
CNN-AE	768.02	865.63
SWW-AE	159.8	590.76
$eForest^s_{500}$	1386.96	1623.93
$eForest^s_{1000}$	701.99	567.64
$eForest^u_{500}$	27.39	579.337
$eForest^u_{1000}$	**6.86**	**153.68**

MLP based AutoEncoders (MLP-AEs) and a convolutional neural network based auto-encoder (CNN-AE) are used for comparison. For MLP-AEs, we follow the suggestions in (Bengio et al. 2007) and use two architectures, with 500-dimensional and 1000-dimensional inner representation, respectively. Concretely, the MLP-AE MLP_1 for MNIST is $(input - 1024 - 500 - 1024 - output)$ and the MLP_2 for MNIST is $(input{-}2048{-}1000{-}2048{-}output)$. Likewise, the MLP-AE MLP_1 for CIFAR-10 is $(input - 4096 - 1024 - 500 - 1024 - 4096 - output)$ and the MLP_2 for CIFAR-10 is $(input - 4096 - 2048 - 1000 - 2048 - 4096 - output)$. For a vanilla CNN-AE, we follow the implementations in the Keras documentation [1] with the following architecture: it consisting of a conv-layers with 16 (3 × 3) kernels followed by 2 conv-layers with 8 (3 × 3) kernels, and each conv-layer has a 2 × 2 max-pooling layer followed. ReLUs are used for activation and logloss is used as training objective. During training, dropout is set to be 0.25 per layer. We also use a more recent CNN based AE, namely SWWAE (Zhao et al. 2015), for comparison. The model use a new form of up-pooling layer instead of the traditional up-sampling layer to explore invariance and equivariance. The SWWAE we use for comparison has the structure (16)5c-(32)3c-8p which is the same structure presented in the paper and we use the same training procedure suggested in the paper.



(a) Reconstructed samples on CIFAR-10.

(b) Reconstructed samples on MNIST.

Figure 2: The original test samples (first rows) and the reconstructed samples

Experimental results are summarized in Table 1 and some reconstructed samples on the test set are shown in Figure 2. The SWWAE indeed has a better performance compared with the vanilla CNN auto-encoder, whereas the eForest auto-encoder achieves the best performance. There might be some more room of improvement for the vanilla CNN-AEs, if some more tuning is performed. Nevertheless, eForest can perform well without such tuning steps.

Table 2: Length of tree depth on MNIST

	Max depth	Ave. depth
$eForest^s_{500}$	48	34.82
$eForest^s_{1000}$	48	34.79
$eForest^u_{500}$	93	70.87
$eForest^u_{1000}$	101	70.07

It is worth noting that the unsupervised eForest had a better performance compared with the supervised eForest, given the same number of trees. Note that each decision tree

[1] https://blog.keras.io/building-autoencoders-in-keras.html

path corresponds to a rule, whereas a longer rule will define a tighter MCR. We conjecture that a tighter MCR might lead to a more accurate reconstruction. Therefore for a forest with longer tree depth may have a better performance. For example, we measured the maximum depth as well as the average depth for all trees on MNIST dataset, as summarized in Table 2. Experimental results give positive supports, as unsupervised eForest indeed has a longer average depth.

Text Reconstruction

In this section, we study the performance of eForest for text reconstruction. Note that the DNN based auto-encoders are mainly designed for images, and if to be applied to texts, some additional mechanisms such as word embedding(Mikolov et al. 2013) are required. Here we want to study the performance of doing auto-encoding directly on text data.

Concretely, we used the IMDB dataset (Maas et al. 2011) which contains 25,000 documents for training and 25,000 documents for testing. Each document was stored as a 5,000-d vector via tf/idf transformation. We used exactly the **same** configuration of eForests for image data. Cosine distance is used for evaluation metric, which is the standard metric for measuring the similarities between documents represented by tf/idf vectors. The lower the cosine distance, the better. The results are summarized in Table 3.

Table 3: Text reconstruction

	Cosine Distance
$eForest^s_{500}$	0.1132
$eForest^s_{1000}$	0.0676
$eForest^u_{500}$	0.0070
$eForest^u_{1000}$	**0.0023**

It should be highlighted that CNN based auto-encoders can not be applied on this kind of input data at all and MLP based auto-encoders is barely useful. After extensive cross-validation for parameter search, the best structure for MLP we could obtained is $(Input - 4096 - 2048 - 1024 - 2048 - 4096 - Output)$, with the performance of 0.512, more than two hundred times worse than eForest.

From the above results, we showed that eForest can also be applied on text data with high performance. Notice that by using only 10% bits of representation (eForest of 500 trees trained unsupervisedly), eForest can already reconstruct the original input with high accuracy. This is a promising result which can be further utilized for data compression or information retrieval.

Computation Efficiency

As a common advantage for tree ensemble models, eForest is also inherently apt for parallel implementation. We implement eForest on a single Intel KNL-7250 (3 TFLOPS peak), and achieved a 67.7 speedup for training 1,000 trees in an unsupervised setting, compared with a serial implementation. For a comparison, we trained the corresponding MLPs and CNN-AEs and SWWAEs with the same configurations as in the previous sections on one Titan-X GPU (6 TFLOPS peak) and the results are summarized in Table 4.

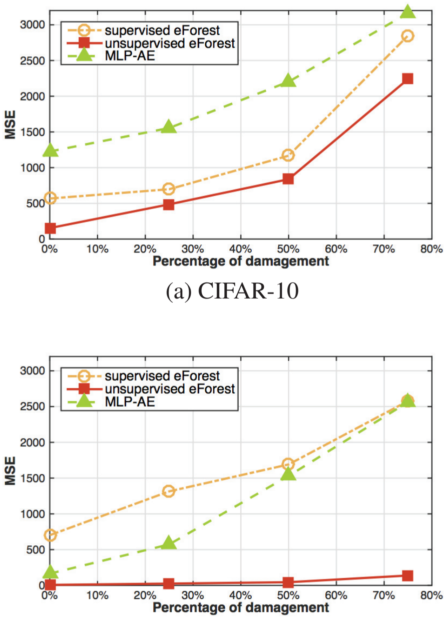
Table 4: Time cost (in seconds). Encoding/decoding is measured in seconds per sample.

Model	MNIST		CIFAR-10	
	Train	Encoding	Train	Encoding
		Decoding		Decoding
$eForest$	19.951	0.225	59.926	0.717
		0.968		2.062
MLP_2	274.471	0.009	1055.958	0.015
		0.006		0.012
CNN	249.958	0.011	280.034	0.01
		0.010		0.01
SWWAE	355.743	0.001	373.080	0.001
		0.001		0.001

From the above results, eForest is much faster than DNN based AE when training, but is slower during encoding/decoding tims. We believe this process can be speedup by some more optimizations for the implementation in the future.

Damage Tolerable

There are cases when the model is partially damaged due to a various reasons such as memory or disk failure. For a partially damaged model is still able to function in such scenarios is one characteristic towards model robustness. The eForest approach for auto-encoding is one such model by its nature since we could still estimate the MCR when facing only a subset of trees in the forest.



(a) CIFAR-10

(b) MNIST

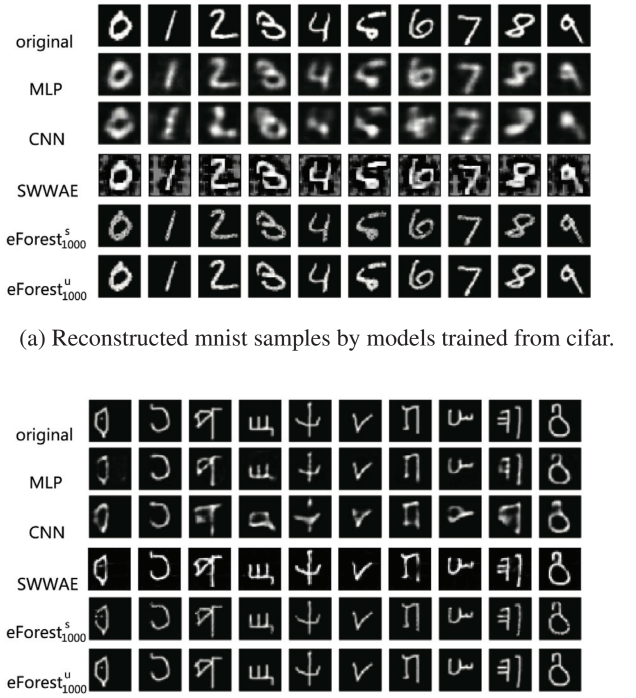
Figure 3: Performance when model is partially damaged

In this section, we test the damage tolerable empirically on CIFAR-10 and MNIST datasets. Concretely, during testing time, we randomly drop 25%, 50% and 75% of the trees and measure the reconstruction error based on the pattern recovered using only the remaining trees. For a comparison, we also randomly turned off 25%, 50% and 75% of the neurons in the MLP_2 with structure exactly the same as in the previous section. The performance results are illustrated in Figure 3.

Form the above result, the eForest approach is more damage tolerable than a MLP-AE, and the unsupervised eForest is the most damage tolerable model among others. Results also showed that eForest is more robust on MNIST than CIFAR-10, we conjecture that this might owe to the fact that MNIST is simpler than CIFAR-10 and thus smaller forest can be good enough, which in turn leading to the fact that more damages are tolerable given the same amount of trees.

Model Reuse for eForest

In an open environment, the test data may belong to a different distribution. In this section, we evaluate the ability for model reuse and the goal here is to train an auto-encoder in one dataset and reuse it in another *without* any modifications or re-training. The ability for model reuse in this context is an important property for future machine learning developments (Zhou 2016).



(a) Reconstructed mnist samples by models trained from cifar.

(b) Reconstructed omniglot samples by models trained from mnist.

Figure 4: The original samples(first rows) and the ones reconstructed by different AEs, where $eForest_{s/u}$ correspond to supervised/unsupervised setting, respectively.

Concretely, we evaluate the ability for model reuse as follows. We trained an unsupervised and an supervised eForest on CIFAR-10 dataset (converted and rescaled to 28×28 gray scale data), each consisting of 1,000 trees , and then use the *exact* trained models to encoding/decoding data from the MNIST test dataset. Likewise, we also trained eForests consists of 1,000 trees on MNIST dataset, and directly test the encoding/decoding performance on the Omniglot datasets (Lake, Salakhutdinov, and Tenenbaum 2015). For a fair comparison, we trained the CNN and MLP based auto-encoders on the same dataset *without* fine-tuning. The architecture for DNNs and the training procedures are the same in the previous sections accordingly.

Table 5: Performance comparison for model reuse (measured by MSE).

Model	cifar train mnist test	Model	mnist train omniglot test
MLP_2	1898.76	MLP_2	596.24
CNN-AE	2657.69	CNN-AE	1280.60
SWW-AE	1612.62	SWW-AE	213.87
$eForest^s$	652.38	$eForest^s$	270.54
$eForest^u$	**90.43**	$eForest^u$	**12.80**

Some randomly picked reconstructed samples are presented in Fig. 4, and the numerical evaluation on the whole test set is presented in Table 5. It can be inferred that eForests has out-performed the DNN approach by a factor more than 100. Notice that the CIFAR-10 dataset is quite different with the MNIST data, and the eForest can still perform well for encoding/decoding even it never encountered MNIST data during training. This showed the generalization ability in terms of model reuse for eForest.

Conclusion

In this paper, we propose the EncoderForest (abbrv. eForest), the first tree ensemble based auto-encoder model, by devising an effective procedure for enabling forests to reconstruct the original pattern by utilizing the Maximal-Compatible Rule (MCR) defined by decision paths of the trees. Experiments demonstrate its good performance in terms of accuracy and speed, as well as the ability of damage tolerance and model re-usability. In particular, on text data, by using only 10% of the input bits, the model is still able to reconstruct the original data with high accuracy. Another advantage of eForest lies in the fact that it can be applied to symbolic attributes or mixed attributes directly, without transforming the symbolic attributes to numerical ones, especially when considering that the transforming procedure generally either lose information or introduce additional bias.

Note that supervised and unsupervised eForest are actually the two ingredients utilized simultaneously in each level of the deep forest constructed by gcForst. This work might offer some additional understandings of gcForst (Zhou and Feng 2017). Constructing a deep eForest model is also an interesting future issue.

References

Bengio, Y.; Lamblin, P.; Popovici, D.; and Larochelle, H. 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems 20*, 153–160.

Bengio, Y.; Yao, L.; Alain, G.; and Vincent, P. 2013. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems 26*, 899–907.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8):1798–1828.

Bourlard, H., and Kamp, Y. 1988. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics* 59(4):291–294.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

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

Chen, T.-Q., and Guestrin, C. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining*, 785–794.

Fernández-Delgado, M.; Cernadas, E.; Barro, S.; and Amorim, D. 2014. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research* 15:3133–3181.

Freund, Y., and Schapire, R. E. 1999. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence* 14(5):771–780.

Friedman, J. H. 2001. Greedy function approximation: A gradient Boosting machine. *The Annals of Statistics* 29(5):1189–1232.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. Cambridge, MA: MIT Press.

Hinton, G., and Ranzato, M. 2010. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Proceedings of the 2010 IEEE Conference on Computer Vision and Pattern Recognition*, 2551–2558.

Hinton, G., and Salakhutdinov, R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

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

Kingma, D.-P., and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science* 350(6266):1332–1338.

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

Liu, F. T.; Ting, K. M.; and Zhou, Z.-H. 2008. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining*, 413–422.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 142–150.

Masci, J.; Meier, U.; Cireşan, D.; and Schmidhuber, J. 2011. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of International Conference on Artificial Neural Networks*, 52–59.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems 26*, 3111–3119.

Mu, X.; Ting, K. M.; and Zhou, Z.-H. in press. Classification under streaming emerging new classes: A solution using completely-random trees. *IEEE Trans. Knowledge and Data Engineering*.

Ruslan, S.; A. Mnih, A.; and Hinton, G. 2007. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine learning*, 791–798.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11:3371–3408.

Willmore, B., and Tolhurst, D. 2001. Characterizing the sparseness of neural codes. *Network: Computation in Neural Systems* 12(3):255–270.

Zhao, J.; Mathieu, M.; Goroshin, R.; and LeCun, Y. 2015. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*.

Zhou, Z.-H., and Feng, J. 2017. Deep forest: Towards an alternative to deep neural networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 3553–3559.

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

Zhou, Z.-H. 2016. Learnware: on the future of machine learning. *Frontiers of Computer Science* 10(4):589–590.