AutoEncoder by Forest

Olga Novitskaia [1] **Maria Begicheva** [1] **Egor Sevriugov** [1] **Kirill Shcherbakov** [1]

Abstract

Autoencoding is an important task which is typically realized by deep neural networks (DNNs) such as convolutional neural networks (CNN). In this paper, we replicate EncoderForest (abbrv. eForest), the tree ensemble based auto-encoder. We present a procedure for enabling forests to do backward reconstruction by utilizing the equivalent classes defined by decision paths of the trees, and demonstrate its usage in both supervised and unsupervised setting. Experiments show that, compared with DNN autoencoders, eForest obtains higher reconstruction error but with fast training speed, while the model itself is reusable and sensitive to partial damage.

1. Introduction

Autoencoder (Vincent et al., 2013) 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. The idea of autoencoders is mostly popular in the field of neural networks. Generally, a neural network based autoencoders learn to copy its input to its output. It has an internal (hidden) layer that describes a code used to represent the input, and it is constituted by two main parts: an encoder that maps the input into the code, and a decoder that maps the code to a reconstruction of the original input. Their most traditional applications were dimensionality reduction (Hinton & Salakhutdinov, 2006) and representation learning (Bengio et al., 2013), but more recently the autoencoder concept has become more widely used for learning generative models of data such as Variational Autoencoders (Kingma & Welling, 2013).

Ensemble learning (Zhou, 2012) is generally considered to be one of the most powerful ways of improving the performance of a model. In short, it is a process by which the models like classifiers, or the experts, are generated and combined together in order to solve a problem. 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 (Fernandez-Delgado et al., 2014). Other successful tree ensembles such as gradient based decision trees (GBDTs) (Chen & Guestrin, 2016) has also proven its ability during the past decade. Recently, deep model based on forests has also been proposed (Zhou & 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) replicated from the original paper (Feng & Zhou, 2017), by enabling a tree ensemble to perform forward encoding and backward decoding operations and can be trained in both supervised or unsupervised fashion.The main contribution of this report, based on the results of the experiment, is the following identified advantages and disadvantages of the eForest:

- Not accurate: the eForest approach experimental reconstruction error is much higher than reasonable CNN based autoencoders, but comparable with a experimental reconstruction error of reasonable MLP based autoencoders both on MNIST and CIFAR10 datasets.
- Efficiency: the eForest on a many-core CPU runs even faster than a CNN autoencoder runs on a Tesla K80 GPU for training.
- Not damage-tolerable: the eForest is less resistant to partial damage than MLP and CNNs with any type of partial model damage simulation (to dropout specific neurons and to dropout channels in the layers).
- Reusable: In addition to replicating the original experiment, reusability of the eForest model were tested and compared additionally on a new experiment (train on CIFAR10 dataset and perform encoding/decoding task on Omniglot dataset), and the model trained from one dataset can be directly applied on the other dataset in the same domain.

The rest of the paper is organized as follows: first we introduce related works, followed by the proposed eForest model

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Kirill Shcherbakov <kirill.shcherbakov@skoltech.ru>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020.

and its algorithms, then experimental results are presented, finally conclusion and future works are discussed.

2. Related Work

Previous approaches for building an autoencoder are neural network based models. For instance, the under-complete autoencoder, which purpose is to compress data for dimensionality reduction (Hinton & Salakhutdinov, 2006) and efficient coding (Liou et al., 2008), sparse auto-encoder gives a sparsity penalty on the on the activation layer (Hinton & Ranzato, 2010), which is related with sparse coding (Willmore & Tolhurst, 2001), and denoising autoencoders (Bengio et al., 2013b) forces the model to learn the mapping from a corrupted input to its noiseless version. Applications ranging from computer vision (Masci et al., 2011) to natural language processing (Mikolov et al., 2013) and semantic hashing (Ruslan et al., 2007) which uses autoencoders in information retrieval tasks. Autoencoding has also been applied in some more recent works such as variational autoencoder for generative models (Kingma & Welling, 2013).

Ensembles of decision trees are popularly used in ensemble learning (Zhou, 2012). For instance, an efficient implementations of GBDT (Friedman, 2001), XGBoost (Chen & Guestrin, 2016), has been widely used in industry and various data analytics competitions. Moreover, Random Forest and completely-random tree forest have been simultaneously exploited in the construction of deep forest (Zhou & Feng, 2017).

3. Algorithms and Models

All autoencoders have two basic functions: encoding and decoding. To make autoencoder via forest we should implement this two functions.

3.1. Encoding via forest

It is easy to do encoding. The forest is set of trees and the prediction of each tree is feature in latent space. But what is actually prediction of tree in terms of encoding? It is the index of leaf corresponding to particular object. Let's summarize it in Algorithm 1.

To illustrate the work of algorithm let's discuss an example in Figure 1. Assume we have two trees in forest. And we try to encode object $x = [1.1, 2, -0.5, 0]$. To do it we should find indexes of leafs to which each tree the object sorted into. As you can see in example the left tree sorted our object into leaf with index 4, and right tree into leaf with index 3. It means that in latent space our object will have representation $x = [4, 3]$.

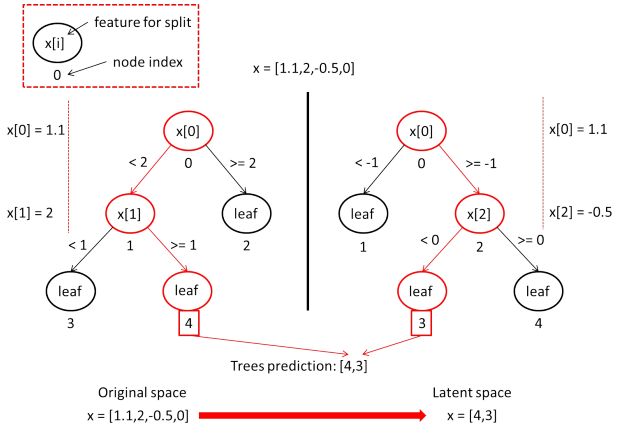

Figure 1. Example of encoding procedure

Algorithm 1 Encoding

Input: trained forest F with T trees, object x
Output: x_{enc}
$x_{enc} = empty(T)$
for i *in* $range(T)$ **do**
  $x_{enc}[i] = F.tree[i].apply(x)$
  % "*apply*" returns the index of leaf in tree i corresponding to object x
end for
return x_{enc}

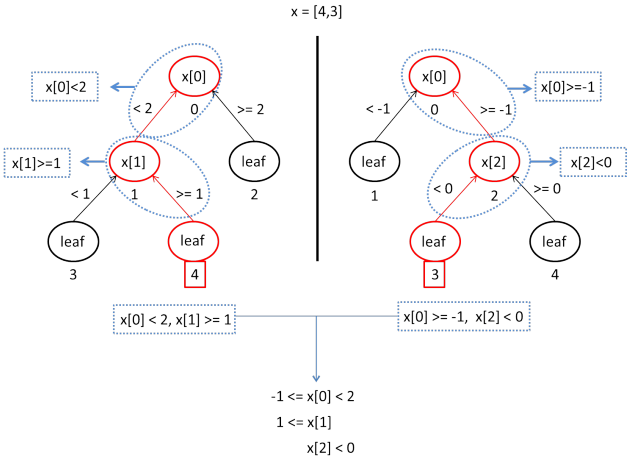

Figure 2. Example of decoding procedure

3.2. Decoding via forest

In previous section we have discussed the encoding procedure and as you could see it can be done without any difficulties. To create the procedure of decoding we should spend more time. Firstly let's try to understand what actu-

ally means decoding - restoring values of original features using the values of latent features. Forest do not provide any formula which can be the link between original feature space and latent feature space. But the forest determines the rules for original feature space. And choosing any object in bounded by rules feature space is result of decoding.

To make it easier let's consider example of decoding described in Figure 2. As you can see in encoding process we start from root of tree and go to the leaf. On the other hand in decoding procedure we go from leaf to root of tree. There are only one way starting from particular leaf to the root. Only object which features satisfies particular conditions, can go throw this path, and our goal is to find such conditions. Assume we have object $x = [4, 3]$ it means that we should start going from leaf 4 in first tree. Moving up from leaf 4 we can find that feature $x[1]$ should be greater or equal 1 to sort object from node 1 into leaf 4. Analogically moving up from node 1 to the root(node 0) we can find that feature $x[0]$ should be less than 2 to sort object from node 0 into node 1. As result from first tree we have set of rules:

$$Rules : x[0] < 2 \cap x[1] \geq 1$$

Analogically we can derive the set of rules from the second tree:

$$Rules : x[0] \geq -1 \cap x[2] < 0$$

To find object which features satisfies all of this rules we should find intersection:

$$-1 \leq x[0] < 2 \cap 1 \leq x[1] \cap x[2] < 0$$

And now one could see the problem: feature $x[0]$ has lower and upper boundary but $x[1]$ has only lower, $x[2]$ has only upper and $x[3]$ has no boundary at all. For feature $x[0]$ we can choose mean value of lower and upper boundary as a result but what about others. One possible way to choose value for feature $x[1]$ and $x[2]$ it's to take the existing boundary as a result - not bad way. But the situation with feature $x[3]$ is not so easy. And proposed solution to take median or mean value for this feature in data set on which the forest trained on. The idea underlying this decision is simple: if haven't found boundary for some feature it means that this feature is not important(no one tree choose this feature for split during training procedure). And it can be caused by the fact that feature value in whole dataset is the same for all objects. One of the examples of such behaviour is MNIST dataset which consist of white numbers on black background. Pixels on the edge of image has the same color - black on all samples in dataset. Sure there are can be other cases like a random values instead of constant, but still not presented in trained forest. In created model because of the reason described above(pixels with the same color in MNIST) we use median as result of decoding for features which are not presented in trained forest. All described above can be divided into three parts:

Algorithm 2 Calculate set of rules for tree

Input: tree t, n - leaf index of object for tree t
Output: rule set(array of size (N,2); for each feature min and max value)
% N - number of features in original space
$set = [[-infty, infty]\ for\ i\ in\ range(N)]$
$path = t.decision\ path(n)$
% path - array of nodes from root to leaf
for $node\ in\ path$ **do**
  $f = node.feature$
  $thres = node.threshold$
  if $next\ node\ in\ decision\ path\ is\ left$ **then**
    % the threshold for the split is upper bound
    $set[f][1] = min(set[f][1], thres)$
  else
    % the threshold for the split is lower bound
    $set[f][0] = max(set[f][0], thres)$
  end if
end for
return set

1. Determine the set of rules for each tree(Algorithm 2)
2. Find intersection between sets of rules for all trees: maximal compatible rule - MCR(Algorithm 3)
3. Create the decoded object, based on MCR(Algorithm 4)

All algorithms created in assumption that all features are numeric(for our project this assumption is valid). To create more general algorithms other types of features should be taken into account.

4. Experiments and Results

4.1. Image Reconstruction

In this section we evaluate and compare the performance of eForest in both supervised and unsupervised setting with DNN-based autoencoders. In this implementation, we take Random Forest (Breiman, 2001) to construct the supervised forest, whereas take RandomTreesEmbedding as the routine for the unsupervised forest. Notice that other decision tree ensemble construction methods can also be used for this purpose. Concretely we set max depth parameter equal 50 in supervised and unsupervised implementation, other parameters were taken as default. We evaluate eForest containing 500 trees or 1,000 trees, denoted by $eForest_{500}$ and

Algorithm 3 MCR

Input: trained forest F with T trees, object x in latent space
Output: final set of rules
$result = [[-infty, infty]\ for\ i\ in\ range(N)]$
for $tree\ in\ F.trees$ **do**
  $set =$ Calculate set of rules for tree$(tree, x[tree])$
  $\%x[tree]$ returns component of x corresponding to $tree$
  $result = intersection(result, set)$
  $\%\ intersection(a, b)$ returns the intersection of rules a and b
end for
return $result$

Algorithm 4 Decoding

Input: trained forest F with T trees, object x in latent space
Output: x_{dec}
$x_{dec} = empty(N)$
% N - number of features in original space
$set = MCR(F, x)$
for $i\ in\ range(N)$ **do**
  if $set[i][0] == -infty\ and\ set[i][1] == infty$ **then**
    $x_{dec}[i] = median[i]$
    $\%\ median[i]$ - median value for feature i
  else if $set[i][0] \neq -infty\ and\ set[i][1] == infty$ **then**
    $x_{dec}[i] = set[i][0]$
  else if $set[i][0] == -infty\ and\ set[i][1] \neq infty$ **then**
    $x_{dec}[i] = set[i][1]$
  else
    $x_{dec}[i] = mean(set[i][0], set[i][1])$
  end if
end for
return x_{dec}

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

MODEL	MNIST	CIFAR10
MLP_1	136.01	518.42
MLP_2	56.49	468.70
CNN	17.59	16.62
$eForest^s_{500}$	1809.92	3108.18
$eForest^s_{1000}$	981.96	1848.89
$eForest^u_{500}$	168.39	1922.49
$eForest^u_{1000}$	38.79	520.97

$eForest_{1000}$ respectively, supervised $eForest^s$ and unsupervised $eForest^u$. Note that $eForest_N$ will represent the input instance as a N-dimensional vector.

We run our experiments on image datasets, because autoencoders, especially DNN-based autoencoders, are mainly designed for image tasks. In particular, we use the MNIST dataset (LeCun et al., 1998), which consists of $60,000$ gray scale 28×28 images (784 dimensional vector per sample) 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 (therefore each image is in R^{1024} per channel) for training and $10,000$ colored images for testing. Since the eForest performance evaluation is quite computationally expensive, we reduce the test size of each dataset to $1,000$ images and for fair comparison, evaluate all models on this test size. Here the MSE is used to compare how far away the target image's pixels from the generated image's pixels.

MLP based autoencoders (MLP-AEs) and a convolutional neural network based autoencoder (CNN-AE) are used for comparison. For CNN-AE, we have tried different architectures with an individual approach to finding training parameters for each dataset due to different dataset's complexity. In particular, the architecture for MNIST dataset has the following encoder: conv-layers with 4 (5×5) kernels, followed by conv-layers with 12 (5×5) kernels, followed by conv-layers with 16 (5×5) kernels, followed by conv-layers with 28 (5×5) kernels, followed by conv-layers with 36 (5×5) kernels, followed by conv-layers with 72 (5×5) kernels, where each is followed by batch-normalization and ReLU activation function. The decoder we used has same structure as encoder except not using batch-normalization after first two conv-layers. The following range of training parameters were tried: for the number of epochs we explored $50, 100, 200$; SGD (Ruder, 2016) and Adam (Kingma & Ba, 2015) optimizers were tried; for the learning rate we tried $0.001, 0.01, 0.1$; the scheduler was the cosine annealing scheduler. The best result for this architecture was shown by the following set of training parameters: the number of epochs is 200 with Adam optimizer (with the default learning rate 0.001) and the cosine annealing scheduler. For CIFAR10 dataset the architecture has the following encoder: conv-layers with 32 (5×5) kernels, followed by conv-layers with 64 (5×5) kernels, followed by conv-layers with 128 (5×5) kernels, followed by conv-layers with 256 (5×5) kernels, followed by conv-layers with 256 (5×5) kernels, followed by conv-layers with 72 (5×5) kernels, where each is followed by batch-normalization and ReLU activation function. The decoder we used has same structure as encoder except not using batch-normalization after first two conv-layers. The same ranges of training parameters were studied and selected as in the first model except the number of epochs that is now 100 because of a time-consuming training process for this specific deep CNN-AE.

For MLP-AE, we also have tried different architectures. In particular, for MNIST dataset we use two architectures, with 500-dimensional and 1000-dimensional inner representation, respectively. Concretely, the MLP-AE MLP_1 for MNIST is $(784 - 1024 - 500 - 1024 - 784)$ and the MLP_2 is $(784 - 2048 - 1000 - 2048 - 784)$. The first layer and the third layers were followed by batch-normalization, ReLU activation function and dropout layer with $p = 0.25$. And the last fourth layer was followed only by batch-normalization and ReLU activation function. The best results (for MLP_1 MSE on the test data was 136.01 and for MLP_2 56.49) were shown with the following set of training parameters: the number of epochs is 1000 with Adam optimizer (learning rate 0.001). No scheduler was applied. MSE is used as training objective for each model. For comparison with the results obtained in the original article, the data is not normalized for all models. For CIFAR10 dataset we use two architectures, with 500-dimensional and 1000-dimensional inner representation, respectively. Concretely, the MLP-AE MLP_1 for CIFAR10 is $(3072 - 4096 - 1024 - 500 - 1024 - 4096 - 3072)$ and the MLP_2 is $(3072-4096-2048-1000-2048-4096-3072)$. The first, second, fourth and fifth layers were followed by batch-normalization, ReLU activation function and dropout layer with $p = 0.25$. And the last sixth layer was followed only by ReLU activation function. The best results (for MLP_1 MSE on the test data was 518.42 and for MLP_2 468.70) were shown with the following set of training parameters: the number of epochs is 500 with Adam optimizer (learning rate 0.001). No scheduler was applied.

MSE is used as training objective for each model. For comparison with the results obtained in the original article, the data is not normalized for all models. All models and experimental results are available in source code[1].

Experimental results are summarized in Table 1. As expected, after carefully tuning all training hyperparameters the second architectures of CNNs models for each dataset outperform others architectures and achieve the best performance. Some reconstructed samples on the test set are shown in Figure 4 and Figure 3. Even though this result looks sad for eForest model on CIFAR-10 dataset comparing to the CNN based autoencoders, the resulted images of eForest are more visually recognizable and clear than blurred images by MLP based models and unsupervised eForest autoencoder with $1,000$ works fine here without careful parameter tuning. 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 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. The model in use is not the best choice, because it is based on default parameters. To obtain a better result the grid search procedure can be done, but it is computationally-intensive.

[1]https://github.com/Olga013/Skoltech-ML-2020-AutoEncoder-by-Forest

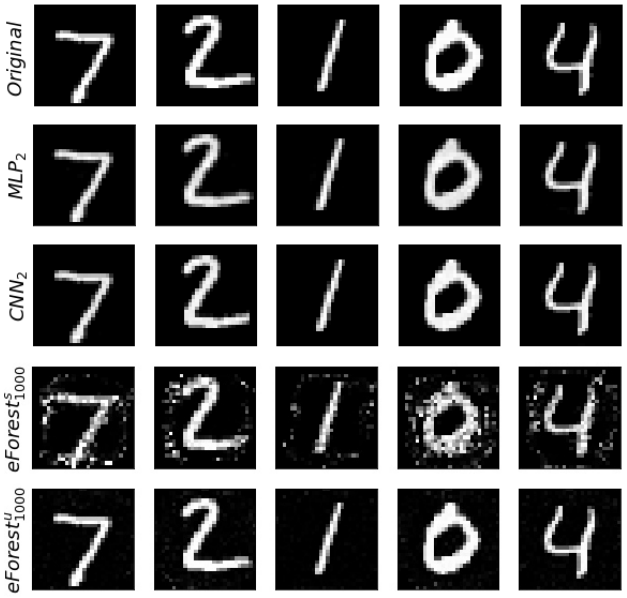

Figure 3. Reconstructed samples on MNIST

4.2. Computation Efficiency

We implement eForest on Intel(R) Core(TM) i7-7700 CPU with 4 cores. For a comparison, we trained the corresponding MLPs and CNN-AEs with the same configurations as in the previous sections on one Tesla K80GPU and the results for training cost as well as testing per sample cost are summarized in the Table 2 and Table 3. From the above results, eForest is more than 10 times fast when training, but is slower during encoding time than DNN-based autoencoders. We hope that the decoding can be speedup by some more optimization and parallelization in the future, since tree ensemble models is inherently apt for parallel implementation.

4.3. 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 cases is one characteristic towards model robustness. 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 mea-

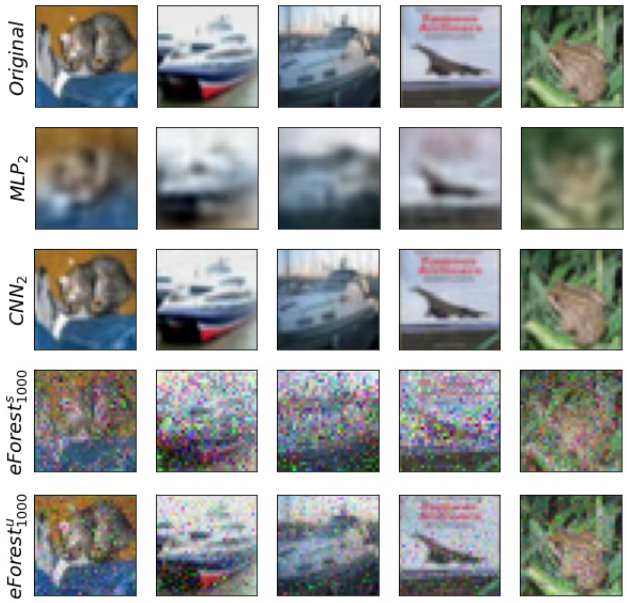

Figure 4. Reconstructed samples on CIFAR-10

Table 2. Time cost (in seconds) on MNIST dataset. Decoding time is measured in sample per seconds.

MODEL	MNIST TRAIN	DECODE
MLP_1	1050	0.0003
MLP_2	1777	0.0003
CNN	2394	0.0006
$eForest^s_{500}$	268	0.9071
$eForest^s_{1000}$	561	1.4393
$eForest^u_{500}$	118	1.0096
$eForest^u_{1000}$	223	1.8121

Table 3. Time cost (in seconds) on CIFAR-10 dataset. Decoding time is measured in sample per seconds.

MODEL	CIFAR-10 TRAIN	DECODE
MLP_1	1237	0.0004
MLP_2	1579	0.0005
CNN	7988	0.0007
$eForest^s_{500}$	1180	1.7208
$eForest^s_{1000}$	2333	3.0264
$eForest^u_{500}$	90	1.7483
$eForest^u_{1000}$	172	3.2822

sure 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-AE and CNN-AE with structure exactly the same as in the previous section. We have also compared eForest damage tolerability with randomly turned off 25%, 50% and 75% channels in the layers of CNN-AE. The performance results are illustrated in Figure 5 and Figure 6.

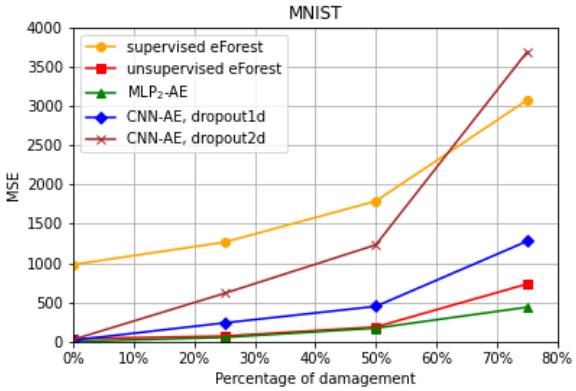

Figure 5. Performance on MNIST when model is partially damaged

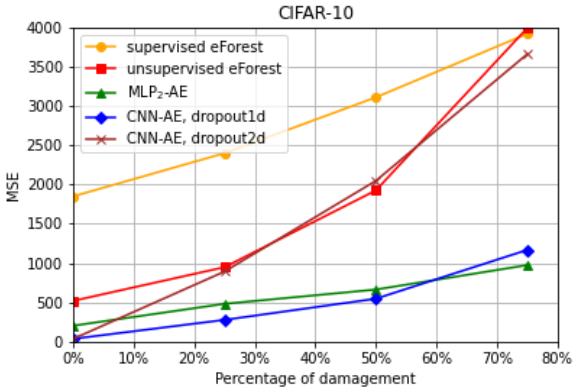

Figure 6. Performance on CIFAR-10 when model is partially damaged

From the obtained results, we see that on MNIST dataset unsupervised eForest is more damage tolerable than CNN-AE and comparable with MLP-AE. But on complex CIFAR-10 dataset eForest looks weak to partial damage together with CNN-AE with randomly turned off channels in the layers. Hence, the more complex the dataset, the more susceptible eForest model is to partial damage.

4.4. Model Reuse

In an open environment, the test data for encoding/decoding may belong to a different distribution with the training data. In this section, we test the ability for model reuse and the goal here is to train a model in one dataset and reuse it

Table 4. Performance comparison for model reuse (measured by MSE) in format: "train dataset" to "test dataset".

MODEL	CIFAR TO MNIST	MNIST TO OMNIGLOT	CIFAR TO OMNIGLOT
MLP_2	931	462	1855
CNN	123	211	269
$eForest^s$	4132	1174	4357
$eForest^u$	1439	104	1671

in another dataset 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).

We test reusability of all presented above models: eForest supervised/unsupervised, CNN and MLP on following tasks:

1. train on CIFAR-10 and test on MNIST, Figure 7
2. train on CIFAR-10 and test on Omniglot (Lake et al., 2015), Figure 8
3. train on MNIST and test on Omniglot, Figure 9

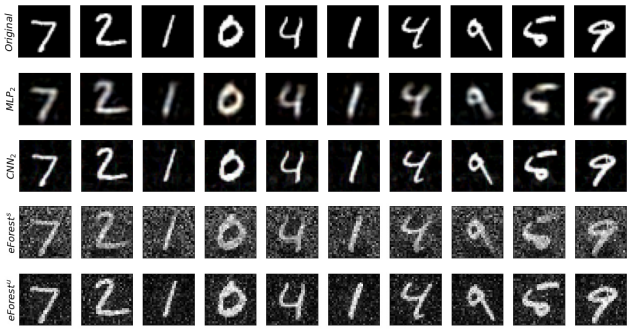

Figure 7. Model reuse with training on CIFAR-10 and testing on MNIST

The numerical evaluation on the whole test set(1000 samples) is presented in Table 4. It can be inferred that eForests has out-performed the DNN approach on simple task: train on MNIST and test on Omniglot. But in spite of good reusability eForest couldn't outperform DNN approach on difficult tasks: train on CIFAR-10 and test on MNIST, train on CIFAR-10 and test on Omniglot. We can say that eForest has a great generalizing ability in comparison with DNN but not applicable to complicated tasks.

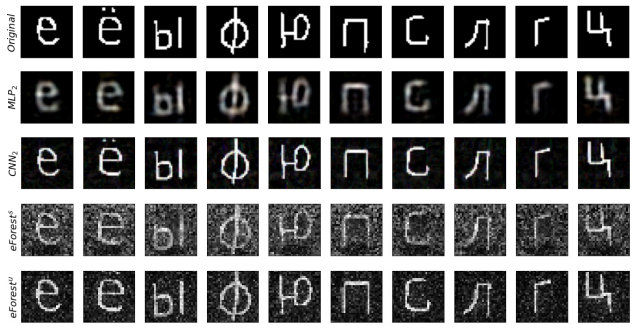

Figure 8. Model reuse with training on CIFAR-10 and testing on Omniglot

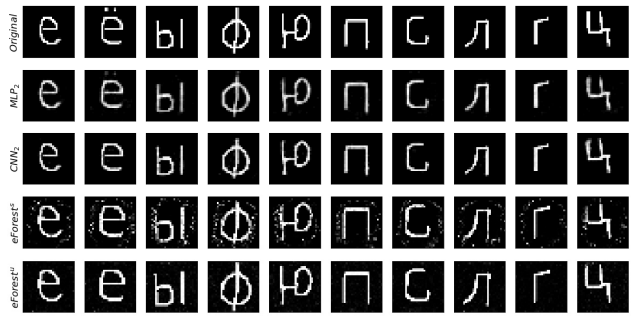

Figure 9. Model reuse with training on MNIST and testing on Omniglot

5. Conclusion

In this paper, replicated the eForest, the first tree ensemble-based autoencoder model, by devising an effective procedure for enabling forests to reconstruct the original pattern by utilizing the MCR defined by decision paths of the trees, and repeated the experiments from the original paper. The results of experiments show that our eForest model is much inferior to DNN-AE on complex tasks, but shows better results in terms of training speed, as well as model reusability. Besides, it was found that on simple datasets eForest demonstrates the ability of damage tolerance, but the more complex the dataset, the more susceptible eForest model is to partial damage. Another advantage of eForest lies in the fact that it can be applied to symbolic attributes or mixed attributes directly (not implemented, but possible), without transforming the symbolic attributes to numerical ones, especially when considering that the transforming procedure generally either lose information or introduce additional bias. An future issue is to find the optimal parameters of eForest to achieve a better MSE, as well as parallelization of the model to speed up training and decode time.

References

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

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

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

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

Feng, J. and Zhou, Z.-H. Autoencoder by forest. *ArXiv*, pp. 1–14, 2017.

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

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

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

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

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. *ICLR*, pp. 2, 2015.

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

Krizhevsky, A. Learning multiple layers of features from tiny images. *Tech. rep., University of Toronto*, 2009.

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

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

Liou, C.-Y., Huang, J.-C., and Yang, W.-C. Modeling word perception using the elman network. *Neurocomputing*, 71 (16):3150–3157, 2008.

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

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

Ruder, S. An overview of gradient descent optimization algorithms. *Cornell University*, pp. 2, 2016.

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

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11: 3371–3408, 2013.

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

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

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

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

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Kirill Shcherbakov

- Coding the CNN-based autoencoders
- Experimenting with CNN AE model parameters on MNIST, CIFAR-10 and Omniglot datasets
- Conducting "Image Reconstruction", "Computation Efficiency", "Damage Tolerable" and "Model Reuse" experiments with CNN-based autoencoders
- Preparing the GitHub Repo (25%)
- Preparing the Section 1, 2 and 5, the Subsection 4.1, 4.2 and 4.3 of this report
- Recording the presentation

Egor Sevriugov

- Coding the eForest auto-encoder
- Experimenting with eForest AE model parameters on MNIST, CIFAR-10 and Omniglot datasets
- Conducting "Image Reconstruction", "Computation Efficiency", "Damage Tolerable" and "Model Reuse" experiments with eForest-based auto-encoder
- Preparing the GitHub Repo (25%)
- Preparing the Section 3, the Subsection 4.4 of this report
- Recording the presentation

Maria Begicheva

- Coding MLP_1/MLP_2 based AE on MNIST training dataset.
- Experimenting with MLP AEs parameters.
- Conducting "Image Reconstruction", "Computation Efficiency", "Damage Tolerable" and "Model Reuse" experiments with MLP AEs on MNIST training dataset and on MNIST/Omniglot testing datasets.
- Preparing the GitHub Repo (25%)
- Preparing the Subsection 4.1 of this report

Olga Novitskaia

- Coding MLP_1/MLP_2 based AE on CIFAR-10 training dataset.
- Experimenting with MLP AEs parameters.
- Conducting "Image Reconstruction", "Computation-Efficiency", "Damage Tolerable" and "Model Reuse" experiments with MLP AEs on CIFAR-10 training dataset and on CIFAR-10/MNIST/Omniglot testing datasets.
- Preparing the GitHub Repo (25%)
- Preparing the presentation

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☐ Yes.
   ☑ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: None

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Full description to eForest algorithm is included in section 3. For CNN-AE and MLP-AE the description of used architectures and training parameters setting is included in subsection 4.1.

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: A link to a downloadable source code is included in subsection 4.1.

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: A link to a downloadable source code is included in subsection 4.1 where all links to downloadable versions of the datasets is located.

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: An explanation of excluded 90% of the test data in each dataset for model evaluations is provided in subsection 4.1.

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: The range of considered hyper-parameters, method to select the best ones is provided in subsection 4.1 for CNN-AE and MLP-AE. The parameters for eForest is explained in section 3.

9. The exact number of evaluation runs is included.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: A description of conducted experiments is included in section 4.

11. A clear definition of the specific measure or statistics used to report results is included in the report.

- [x] Yes.
- [ ] No.
- [ ] Not applicable.

Students' comment: A definition of the specific measure of conducted experiments is included in section 4.

12. Clearly defined error bars are included in the report.

    - [ ] Yes.
    - [ ] No.
    - [x] Not applicable.

    Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - [x] Yes.
    - [ ] No.
    - [ ] Not applicable.

    Students' comment: None