Outlier Detection with Autoencoder Ensembles

Jinghui Chen* Saket Sathe† Charu Aggarwal† Deepak Turaga†

Abstract

In this paper, we introduce autoencoder ensembles for unsupervised outlier detection. One problem with neural networks is that they are sensitive to noise and often require large data sets to work robustly, while increasing data size makes them slow. As a result, there are only a few existing works in the literature on the use of neural networks in outlier detection. This paper shows that neural networks can be a very competitive technique to other existing methods. The basic idea is to randomly vary on the connectivity architecture of the autoencoder to obtain significantly better performance. Furthermore, we combine this technique with an adaptive sampling method to make our approach more efficient and effective. Experimental results comparing the proposed approach with state-of-the-art detectors are presented on several benchmark data sets showing the accuracy of our approach.

1 Introduction

Outliers are data points that differ significantly from the remaining data. In the unsupervised outlier detection setting, prior labels about the anomalousness of data points are not available. In such cases, the most common techniques for scoring data points for outlierness include, distance-based methods [5, 11, 17], density-based methods [6], and linear methods [21]. An overview of different outlier detection algorithms may be found in [1]. The basic approach in neural networks is to use a multi-layer symmetric neural network to reconstruct (i.e, *replicate*) the data. The reconstruction error is used as the outlier score. There are several problems with this approach. First, even though deep neural networks are generally considered a powerful learning tool on large data sets, the effectiveness on smaller data sets remains in doubt because of the overfitting caused by the large number of parameters. Training such neural networks often results in convergence to local optima. Increasing data size reduces overfitting but can cause computational challenges. Furthermore, there is sometimes an inherent bottleneck on data availability. Although neural networks have been explored for outlier detection [10, 13, 22], this class of approaches has not been popular in the outlier detection community because of the aforementioned drawbacks.

In this work, we employ autoencoder ensembles for unsupervised outlier detection. More specifically, instead of fully connected autoencoders, various randomly connected autoencoders with different structures and connection densities, which reduces the computational complexity, are used as base ensemble components. Moreover, we leverage a carefully designed adaptive sample size method within the ensemble framework to achieve the dual goals of improved diversity and training time. Therefore, our approach combines adaptive sampling with randomized model construction in order to achieve high-quality results. We refer to this model as *RandNet*, which stands for *Randomized Neural Network for Outlier Detection*.

Even though the sparse structure of each autoencoder allows overfitting, significant overall gains in efficiency and benefits from diversity of the various components can be obtained if the scores from individual ensemble components are combined. We present experimental results showing the effectiveness of the approach. Although we do not investigate the option of training the base ensemble components in parallel, a salient observation about this approach is that the training process can be easily parallelized.

The remainder of this paper is organized as follows. We will discusses related work in Section 2. Section 3 discusses our proposed autoencoder ensemble method for outlier detection. Section 4 discusses the experimental results, while the conclusions are presented in Section 5.

2 Related Work

The problem of outlier analysis has been studied widely in the community [1]. Numerous methods such as distance-based methods [5, 11, 17], density-based methods [6], linear methods [21], and spectral methods [8, 18] have been proposed. The autoencoder method can be seen as a generalization of the class of linear schemes [10], in which a nonlinear representation of the data is constructed for outlier detection. Comparisons of neural network methods may be found in [22] and a detailed survey on neural networks may be found in [13]. One can view a neural network as a model that scores outliers with the use of nonlinear dimensionality reduction. Recently, ensemble methods have found increas-

*University of Virginia.
†IBM T. J. Watson Research Center.

ing interest in the literature [2, 3, 4]. Several ensemble methods such as feature bagging [12], subspace histograms [19], high-contrast subspaces [9], and locally relevant subspaces [14, 15] have been proposed. The spectral methods in [8, 18] can also be viewed as nonlinear dimensionality reduction methods that reduce the data representation in a nonlinear way in order to score data points as outliers.

We also use adaptive sampling in order to speed up our training process of the neural network. Although the adaptive sampling approach has been used in a different stochastic optimization problem (which is convex) [7], there is no guarantee that it will work in practice in a non-convex optimization problem like neural networks. Correspondingly, the precise approach for adaptive sampling is also somewhat different from the linear adjustment advocated in [7] which seems to be unreasonable with the settings in this paper (a detailed discussion on this topic is provided in section 3.3).

3 The RandNet Model

An autoencoder is a special type of multi-layer neural network that performs hierarchical and nonlinear dimensionality reduction of the data. Typically, the number of nodes in the output layer is the same as the input layer, and the architecture is layered and symmetric. The goal of an autoencoder is to train the output to reconstruct the input as closely as possible. The nodes in the middle layers are smaller in number, and therefore the only way to reconstruct the input is to learn weights so that the intermediate outputs of the nodes in the middle layers represent reduced representations. Figure 1 illustrates a fully connected autoencoder. Note that the outputs of the bottleneck layer represent the reduced representation.

Since the autoencoder creates a reduced representation of the data, it is a natural approach for discovering outliers. The basic idea here is that outliers are much harder to be accurately represented in this form than the inliers (or normal points). Therefore, on reconstructing an outlier, the error will be large. This provides a natural way to score a data point. Nonlinear dimensionality reduction methods such as spectral transformations [18] have recently been explored in the literature with some success. In this light, it is somewhat surprising that the success with neural networks has been limited. An important issue is that the outliers are often themselves included within the training model. As a result, overfitting becomes increasingly likely. This is one of the reasons that neural networks have not achieved much success in spite of the known success of other dimensionality reduction methods in outlier detection. Ensemble learning methods [2] present a natural solution to address this dilemma.

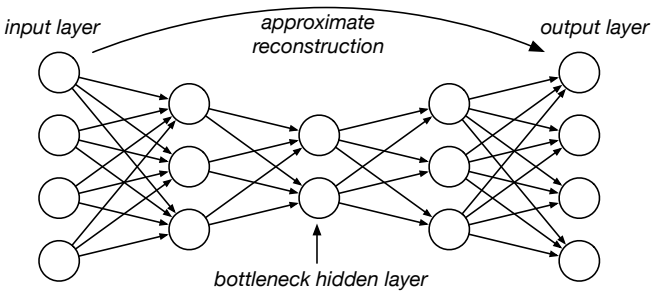


Figure 1: Sketch of the fully connected autoencoder model used by *RandNet*.

Ensemble learning methods are algorithms that combine the predictions from different base detectors in order to create more robust results. The basic idea is that predictive algorithms often have a natural variance in the scores, depending on the choice of the data or the design of the base model. Therefore, by using multiple executions of the model and taking a central estimator of the scores (such as the mean or median), the variance is reduced. However, there is no guarantee that the combination of multiple detectors will always perform better than the best individual detector in the ensemble. In order to make ensemble learning methods work, the individual ensemble components must be adequately diverse. This is achieved by creating predictive models such that each ensemble component is able to capture different parts of the underlying patterns. The combined model is often more powerful than the individual detectors.

The general idea of our method is to use ensemble learning method with autoencoders to obtain higher accuracy. However, directly combining a set of fully connected autoencoders will often not be helpful. Due to the fact that the same network structure is used, the results we obtain from the autoencoders would be somewhat similar. This lack of diversity is unhelpful from the point of view of variance reduction. To address this problem, we use randomly connected autoencoders in which some of the connections are randomly dropped. Figure 2 shows a two-layer example of a randomly connected autoencoder. It is worth noting that this approach is fundamentally different from methods like Dropout [20], in which some of the connections are randomly masked during training and there is only a single model being trained. In *RandNet*, we have a set of completely independent (different) neural networks, and the result are combined. In Dropout [20], the main purpose of masking connections is to avoid co-adaptation of the weights within the same network. While in *RandNet*, we allow overfitting within a single

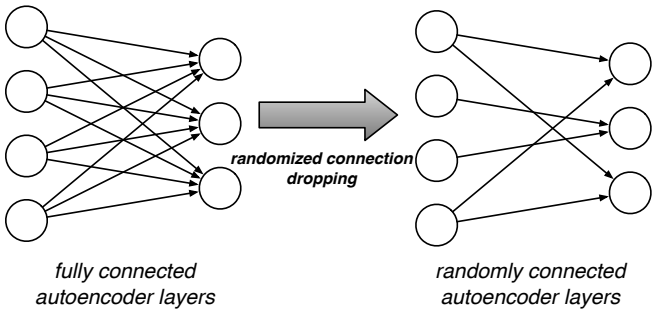


Figure 2: Left: A two-layer fully connected autoencoder. Right: The same autoencoder after permanently dropping connections.

network but reduce the variance only in the combination from multiple networks.

3.1 Neural Network Structure The basic autoencoder structure uses a symmetric design that is similar to that shown in Figure 1. The input layer has exactly the same number of nodes as the dimension of our training data, denoted as d. The output layer will have the same number of nodes due to the symmetric design of autoencoders. For all the following layers, up to the bottleneck hidden layer, the number of nodes is set at a ratio of α from the previous layer. α is referred to as the *structure parameter*. An exactly symmetric structure applies to the layers after the bottleneck layer. Note that for all layers, the minimum limit for number of nodes is 3. Therefore, if the ratio-wise approach indicates fewer than 3 nodes for a layer, then the number of nodes in the layer is set to 3. This is to avoid an excessive level of compression in the middle layer so that the data cannot be properly reconstructed.

Choice of Activation Functions: At the basic level, each node in the neural network computes a linear function of its inputs. The activation function applies a nonlinear function to the linear combination of inputs created by a unit of computation. For the choice of the activation functions we use the Sigmoid and the Rectified Linear (ReLU) [16] units. Specifically, in the first hidden layer (i.e., the one nearest to the input layer) and the output layer, the Sigmoid activation function is used. The Sigmoid activation function is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

For all other layers, we use the ReLU activation function:

$$f(x) = \max(0, x).$$

Using different activation functions in different layers aims at balancing the advantages and disadvantages of the two activation functions. In practice, we found that this combination achieves better performance than a fixed choice. The Sigmoid function is known to cause the "vanishing gradient problem", and its operations are also expensive compared to the ReLU unit. The ReLU unit is simple and computationally costless, and it does not suffer from the vanishing gradient problem. However, it suffers from a different problem known as the "Dying ReLU" problem, recently discovered by researchers in [24]. It means that during training, a weight update triggered by a large gradient flowing through a ReLU neuron could make the neuron inactive for other data points in all future time. Finally, this will cause many of the neurons to be in a dying state where no weight is updated and the network continues to give the same output over iterations. This motivates the use of Sigmoid functions at the two ends of the network, and the ReLU for other layers. On one hand, the use of the Sigmoid functions will ensure that even if all the middle layer ReLU neurons are dead, at least we have two layers properly working at the two ends of the network, which guarantees an improved worst case. On the other hand, the dying ReLU problem usually happens when the gradients are too large, the vanishing gradient caused by the Sigmoid function actually helps in preventing ReLU units from dying during backpropagation.

Random Connection Generation: Here we introduce the randomly connected model for autoencoders. The basic idea of our approach is to randomly drop connections in a neural network, and yet retain a certain level of control on the connection density between various layers and also create models with different types of densities.

The random connection generation proceeds as follows. Suppose, we have two adjacent layers with nodes ℓ_1 and ℓ_2, then there are $\ell_1 \cdot \ell_2$ connections possible between these two layers. From these possible connections we choose $\ell_1 \cdot \ell_2$ connections *with replacement*. Since we sampled with replacement, in the sampled $\ell_1 \cdot \ell_2$ connections we will have some redundant (or repeated) and some missing connections. Then we keep all the connections that are present in the $\ell_1 \cdot \ell_2$-sized sample we choose. The dropped connections are those connections that we did not pick during sampling. Notice that with this approach we vary on the rate of sampling by the uncertain number of missing connections, and thus are able to produce different types of randomly connected autoencoders with varying densities.

3.2 Outlier Scoring Each ensemble component is constructed by randomly sampling connections, inserting the aforementioned activation functions and enforcing the network structure. Also it is trained only on a randomly drawn sample of size s from the full data

set. However, once all the neural networks (or ensemble components) are trained, each data point can be scored by each network by computing autoencoder reconstruction error. The outlier score of each data point in each ensemble component is obtained as the *reconstruction error* of that point, when we run it through the neural network.

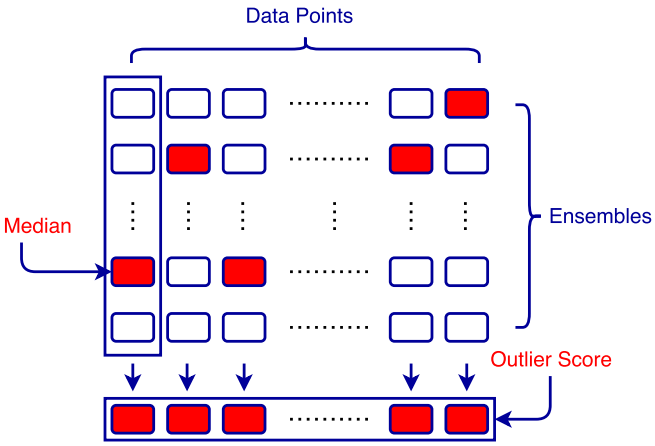


Figure 3: Example of the outlier score vector and final outlier score computation.

Suppose we have m ensemble components, with the training set of n data points and d dimensions. We denote the i-th ensemble component's j-th record input data point as $\mathbf{x}_{ij} \in \mathbb{R}^d$ and the autoencoder's reconstructed output as $\mathbf{o}_{ij} \in \mathbb{R}^d$. Thus, for i-th ensemble component, we create an *Outlier Score Vector*, denoted by OS_i and represented as follows:

$$[OS_i]_j = \sum_{k=1}^{d} ([\mathbf{x}_{ij}]_k - [\mathbf{o}_{ij}]_k)^2. \quad i \in 1 \ldots m, j \in 1 \ldots n$$

Note that the outlier score vector is obtained by the squared sum of the error in reconstruction over the different dimensions of the data point. If this outlier score vector is not normalized, then the scores may vary rather wildly over different components, especially since different components might have different overfitting propensities. To address this problem, we normalize the score vector of each component, so that it has a standard deviation of one unit. The final outlier score of a data point is obtained by computing the median score over different ensemble components. An illustration of the scoring process is shown in Figure 3, where each row represents the outlier score vector of an ensemble component.

3.3 Training and Adaptive Sampling In this section we present *RandNet*'s training and adaptive sampling process. Although the back-propagation algorithm for training a neural network is fairly standard, there are a few major issues related to the learning rate and adaptive sampling that we addressed in our design of *RandNet*. These issues are discussed below:

Adaptive Learning Approach: We use an adaptive learning method, referred to as *RMSprop* [23]. The basic idea is to divide the gradient by a running average of its recent magnitude. It keeps a running average $RA(t)$ as follows:

$$RA(t) = \rho \cdot RA(t-1) + (1-\rho) \cdot g_t^2,$$

where g_t stands for the gradient computed at time t. The parameter ρ regulates the adaptive level of adjustment, and it was chosen to be 0.9. The *RMSprop* approach works by updating the parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{RA(t) + \epsilon}} \cdot g_t,$$

Here, η is the static learning rate and ϵ is a small value that avoids ill-conditioned solutions, such as those caused by division by zero.

Pre-training: It is well-known that with deep neural networks, it is very easy to get trapped in local optima, which prevents further effective backpropagation of errors. Pre-training is a way of initializing the weights, so that the initial starting point is in a good "basin" from where gradient-descent procedure has a better chance of reaching a global optimum. Pre-training is not always needed for accuracy when a very large amount of data is available. However, pre-training speeds up the gradient-descent because the starting point is of a higher quality. Furthermore, pre-training also has accuracy advantages for smaller data sets. We used layer-wise pre-training where the weights of two symmetrically chosen layers were trained at one time. The idea here is to first train the outermost pair of layers with a single hidden layer (thereby creating a shallow, three-layer network) in order to learn the initial set of the weights of the outermost layers and the corresponding reduced representation. This reduced representation is then used to train the next set of layers with another three-layer model and the next level of hierarchical reduction is obtained. For each ensemble component, we used 1/3 of its training data to perform layer-wise pre-training.

Adaptive Sampling: The purpose of adaptive sampling is to make the optimization procedure more efficient by changing (or *adapting*) the training sample size in each iteration. The key insight is that typically a very accurate computation of the gradient is not necessary during the early stages of training. This is because in early stages the solution is relatively crude and all we need is an approximately correct direction. Therefore, small sample sizes are sufficient. As the algorithm progresses and the candidate solution starts to approach

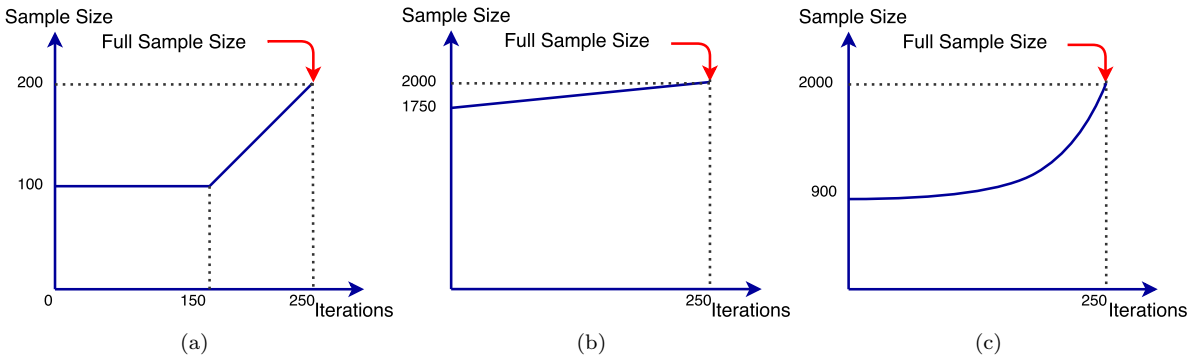


Figure 4: An illustration of the modified adaptive sample size selection method.

the optimal point in the solution space, more accurate gradients are needed to make further progress. Thus, in practice using an adaptively increasing sample size during training would be a beneficial strategy. Although some adaptive sampling methods are occasionally used in convex optimization problems, it is generally harder to use them in neural networks because the optimization problem is highly nonlinear.

In particular, the original algorithm in [7] that is suited to convex optimization problems is adapted to neural networks.. The original method of adaptive sampling is shown in Figure 4(a). This method uses a constant sample size in an initial phase. At some point, the sample size increase linearly until the size increases to that of the full data set. However, a linear increase can be too slow for a neural network in which each iteration is significantly slow and the amount of improvement in later iterations can often be small. A second solution is to forego the initial phase of using a constant sample size and start with a larger sample size. This approach is shown in Figure 4(b). This solution is even worse because the use of an initially large sample size will result in rather slow training. In order to solve these problems, we use a strategy of increasing the sample size by a constant factor in each iteration, which results in an exponentially increasing sample size with the number of iterations. Our strategy is illustrated in Figure 4(c). Given a fixed number of iterations and a factor by which to increase the sample size in each iteration, it is easy to compute the starting sample size and to increase it by the pre-defined factor in each iteration.

4 Experimental Results

In this section, we present our experimental results on several publicly available data sets. We compare our approach against several baseline methods and also present sensitivity analysis results. We set the number of layers to be 7, the structure parameter α as 0.5. The adaptive factor is set as 1.01. We perform the experiments on 100 ensembles and train each of them for 300 iterations. The training size s equals $n/10$.

4.1 Data Sets and Performance Metrics We used several real data sets from the UCI Machine Learning Repository[1]. We summarize the important statistics about these data sets in Table 1. In data sets with unbalanced classes, the instances from the majority class(es) are labeled as inliers, while the instances from the minority class(es) are labeled as outliers. In data sets with reasonably balanced classes, a minority class is created by uniformly down-sampling one of the majority classes. Furthermore, every feature column in each data set has been normalized to be within the range of $[0, 1]$.

Table 1: Summary of the data sets.

Data set	Samples	Features	Percent Outliers(%)
Cardio	1,831	21	9.6
Ecoli	336	7	2.7
Kddcup99	102,563	41	5.2
Lympho	148	18	4.1
Optdigits	5,216	64	2.9
Pendigits	6,870	16	2.2
Seismic	2,584	11	6.5
Thyroid	3,772	6	2.4
Waveform	3,509	21	4.7
Yeast	1,364	8	4.8

The Cardiotocography (Cardio) data set contained measurements of fetal heart rate signals. The classes in the data set were *normal, suspect,* and *pathologic*. The *suspect* class was discarded. The *normal* class was marked as *inliers*, while the *pathologic* class formed the outliers. The Ecoli data set contained 8 classes; here

[1] https://archive.ics.uci.edu/ml/datasets.html

Table 2: Accuracy comparison with state-of-the-art methods. The best AUC is highlighted in boldface.

Data set	*RandNet*	Hawkins [10]	LOF [6]	HiCS [9]	Spectral [18]
Cardio	**92.87**	92.36	50.63	92.37	78.90
Ecoli	85.42	82.87	39.35	53.89	**91.81**
Lympho	**99.06**	98.70	97.77	92.37	78.16
Optdigits	87.11	**87.63**	67.11	43.63	2.66
Pendigits	**93.44**	89.81	54.37	60.61	87.88
Seismic	**71.28**	68.25	55.59	59.90	66.71
Thyroid	**90.42**	87.47	63.04	43.76	72.94
Waveform	**70.05**	61.57	55.48	59.24	62.88
Yeast	**82.95**	82.12	54.30	54.45	77.70

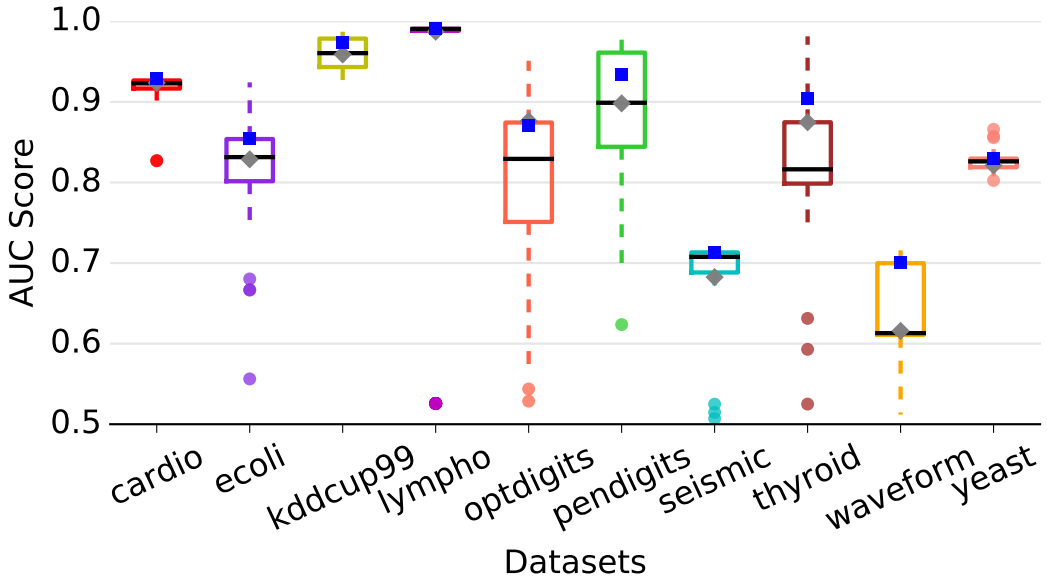


Figure 5: Box-plots showing the ensemble performance of *RandNet*.

we used classes *omL, imL* and *imS* as outliers and the rest were included as inliers. The Kddcup99 data set was a network intrusion data set from the KDD Cup Challenge, 1999. The data points corresponding to intrusion attacks were marked as outliers, while excluding the DDoS (Denial-of-Service) attacks from the data set because of their copious nature. The other packets were marked as inliers. In the Lymphography (Lympho) data set, classes 1 and 4 were outliers while the others were inliers. The Waveform data set contained three classes (namely 0,1, and 2) of waves. We sampled 10% of the points from class 0 and included them as outliers, while all instances of the other classes were included as inliers. The Optdigits was a data set that contained digits in the range of 0-9. The instances of digits 1-9 were inliers and the outliers were 150 instances of the digit 0. The Seismic Bumps (Seismic) contains the data set for forecasting seismic bumps as *hazardous* or *non-hazardous*. Instances of the *hazardous* class were marked as outliers, while the *non-hazardous* was marked as inlier. Non-numerical features were removed from the data set. The ANN-Thyroid (Thyroid) and Pendigits data sets are from [9]. There are 10 classes in the Yeast data set. Out of which, classes *ME3*, *MIT*, *NUC* and *CYT* were included as inliers, while a 5% sample of randomly selected points from other classes were included as outliers.

The Receiver Operating Characteristic (ROC) curves are used to generate the full trade-off between the true positive rate and the false positive rate. The ROC curve can be summarized by the area under the ROC curve. This is known as the AUC. For *RandNet* we report the AUC computed using the final outlier scores.

4.2 Baselines We compare our results with several other baselines reflecting the different aspects of our approach. Specifically, we use another neural network method by Hawkins [10], a conventional distance-based

algorithm like LOF [6], an ensemble-centric algorithm HiCS [9], and another spectral nonlinear dimensionality reduction method [18]. We set the number of nearest neighbors to $k = 5$ in all our experiments. The HiCS method is an ensemble-centric subspace outlier detection technique that averages the scores from multiple subspaces. For HiCS the number of Monte Carlo trials was set to 80, $\alpha = 0.1$, and candidate cutoff was set to 40.

4.3 Accuracy Results In Table 2, we compare our experimental results with the other state-of-the-art outlier detection baselines. In most cases, our approach is able to outperform the state-of-the-art methods significantly. The results are somewhat correlated with the other neural network baseline, which is not particularly surprising. In order to show the specific relationship between the performances of our method and the baseline technique by Hawkins [10], we use box plots to show the performances of the individual base detectors. The box-plot shows the variation in AUC performance of the base components. The median of the box plot is shown by a black line. The grey diamond shows the performance of the baseline neural network by Hawkins [10], and the blue dot represents *RandNet*'s ensemble AUC score. In Figure 5 we show the box plot results of our ensemble method on several public data sets. We can see that in almost all data sets our method achieves major gains over the baseline method. This demonstrates the advantages of using ensembles of neural networks. Note that there is some degradation in the base performance, but we can still achieve high-quality results due to variance reduction.

4.4 Parameter Sensitivity In this part we test the sensitivity of the key parameters in our experiments. Due to space limitations, we only show the results for a subset of the data sets.

Number of Ensemble Components: We analyze the parameter sensitivity to the number of ensemble components m. We vary the number of ensemble components from 25 to 200, to demonstrate the effect of varying this parameter. The result shows that *RandNet* is not very sensitive to the number of ensemble components. As we can see in Figure 6, there is no significant accuracy improvement after the number of ensembles reaches 100. This is typical behavior observed in all ensemble methods where the performance stabilizes after a certain point.

Number of Layers: We analyze the parameter sensitivity to the number of layers that regulate the depth of the network. The number of layers was varied from 3 to 9. It is evident from Figure 7 that our method is quite sensitive to the number of layers in the network, especially with larger data sets. On the one hand, more depth in the network provides more powerful modeling ability for the autoencoder. On the other hand, deeper networks are significantly harder to train, especially with limited data. Observe that in Figure 7 the AUC initially shows improvement but then it starts to degrade. It performs quite poorly when 9 layers are used. It is evident that using 7 layers offers a good trade-off point between modeling capability and training challenges.

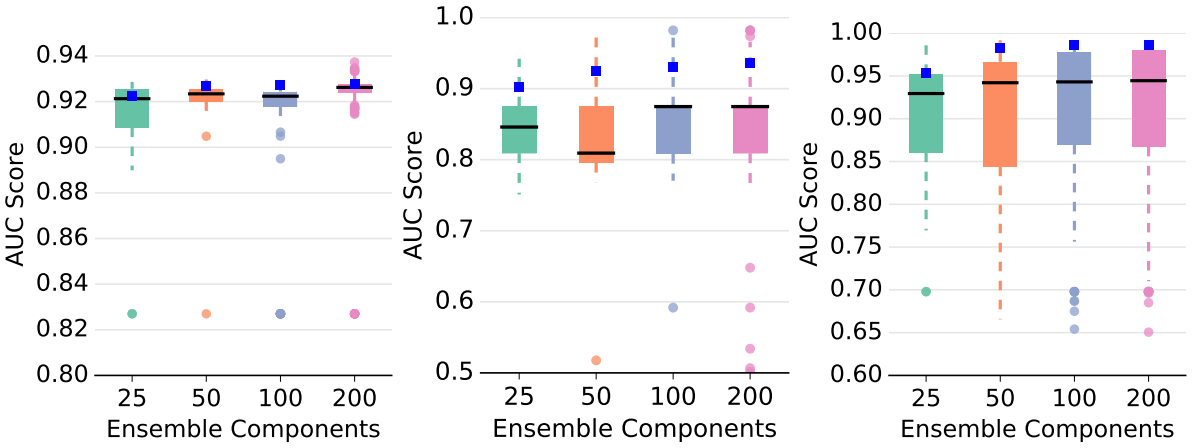
Adaptive Sampling Factor: We analyze the parameter sensitivity to the adaptive sampling factor. The adaptive sampling factor controls the exponential growth of the training sample size in each training iteration, and regulates the effectiveness of the adaptive sampling method. We test it by setting the sample-size to exponential growth ratios of 1.001, 1.005 and 1.01. In addition, we tested the case in which there is no adaptive sample size, which corresponds to the case in which the growth ratio is set to 1.

The results from this experiment are shown in Figure 8. It is evident that the algorithm could indeed achieve good performance without an adaptive sampling strategy. However, the computational cost is much higher if adaptive sampling is not used. In particular, if 300 iterations as shown in Figure 8 are used then the performance is quite similar for all experiments. With the adaptive factor set to 1.001, 1.005, and 1.01 respectively, we need to respectively perform only $86\%, 51\%, 31\%$ of the original number of gradient updates. This result clearly illustrates the significant efficiency improvement brought by adaptive sampling.

Structure Parameter α: The structure parameter α regulates the number of nodes in various layers. We test it under four situations, by setting α to 0.3, 0.4, 0.5 and 0.6, respectively. The results from these experiments are shown in Figure 9. Observe that the accuracy is not very sensitive to the structure factor α, as long as α lies in a reasonable range. Intuitively speaking, larger α's denote more nodes in the network hence it should have more powerful modeling ability. However, higher modeling ability does not always translate to higher accuracy due to the tendency of the neural network to overfit. Within the tested range, the performance improved slightly with α but the differences were not substantial.

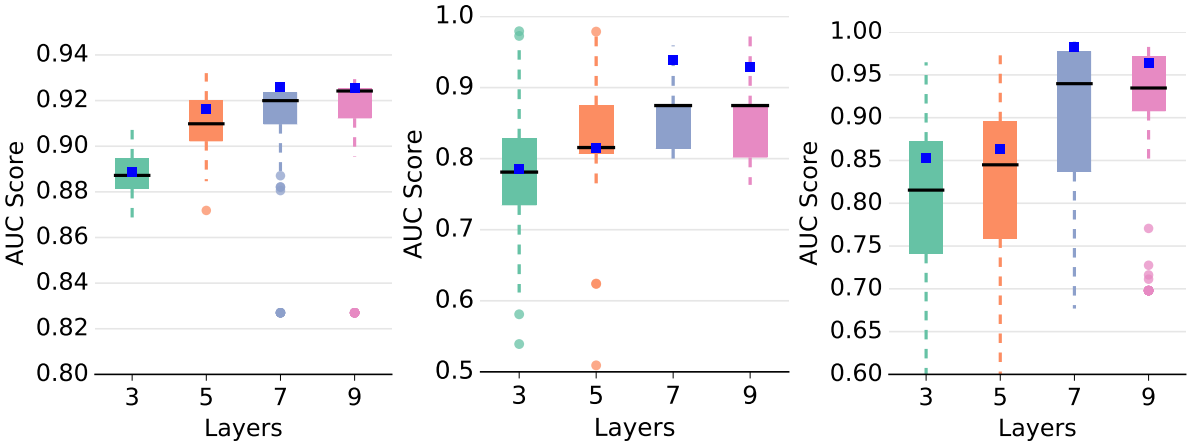
5 Conclusions

This paper shows how to use an ensemble of autoencoders in order to perform anomaly detection. The proposed technique improves significantly over the earlier neural network methods for anomaly detection. It uses random edge sampling in conjunction with adaptive data sampling in order to achieve high-quality re-



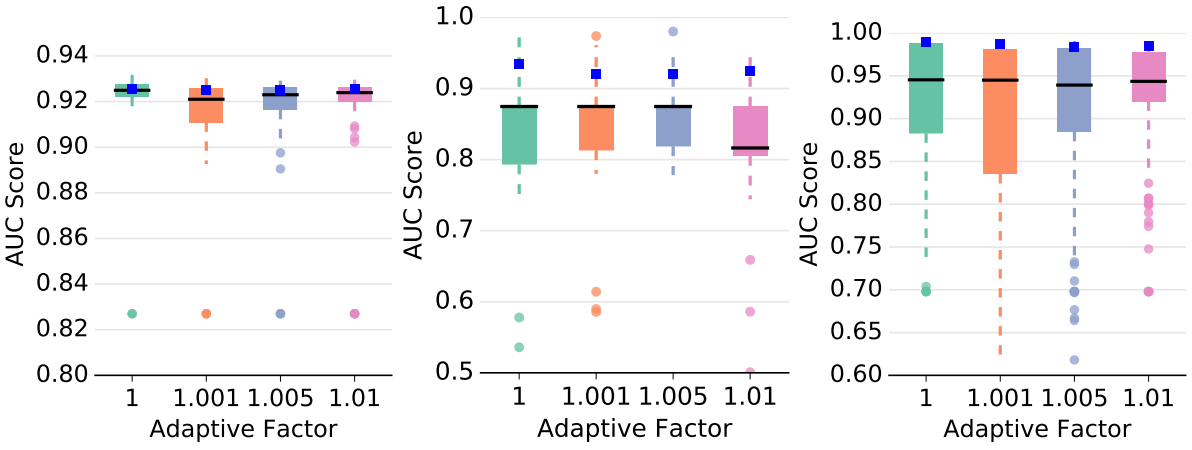
(a) Cardio (b) Thyroid (c) Pendigits

Figure 6: Sensitivity to number of ensemble components.



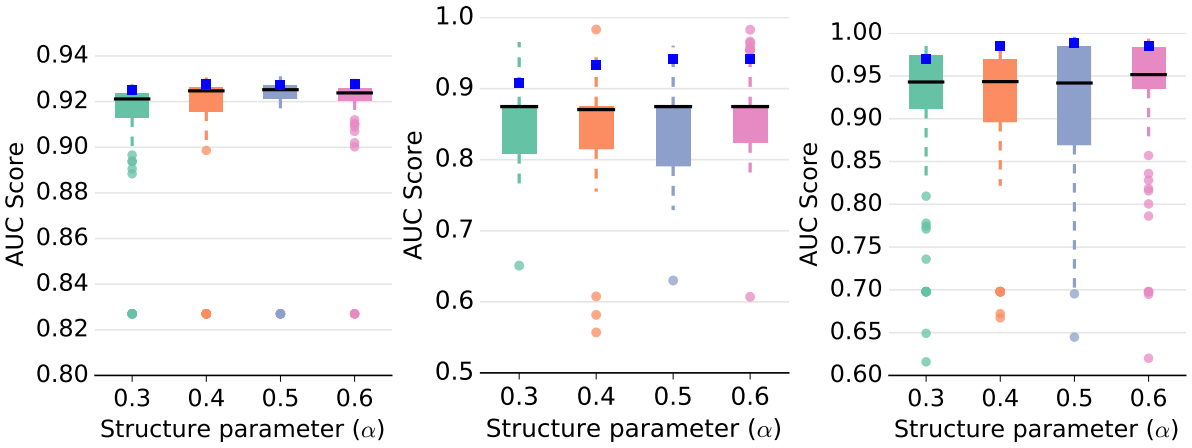
(a) Cardio (b) Thyroid (c) Pendigits

Figure 7: Sensitivity to number of neural network layers.



(a) Cardio (b) Thyroid (c) Pendigits

Figure 8: Sensitivity to adaptive sampling factor.



(a) Cardio (b) Thyroid (c) Pendigits

Figure 9: Sensitivity to the structure parameter α.

sults. Unlike the earlier methods proposed for neural-network based outlier detection, our approach is able to avoid overfitting and achieve robustness because of its ensemble-centric approach. Furthermore, it is also competitive with respect to the state-of-the-art methods.

References

[1] C. Aggarwal. Outlier Analysis. *Springer*, 2017.

[2] C. Aggarwal. Outlier Ensembles: Position Paper, *ACM SIGKDD Explorations*, 2013.

[3] C. Aggarwal, S. Sathe. Theoretical Foundations and Algorithms for Outlier Ensembles. *ACM SIGKDD Explorations*, 2015.

[4] C. Aggarwal, S. Sathe. Outlier Ensembles: An Introduction. *Springer*, 2017.

[5] F. Angiulli, C. Pizzuti. Fast outlier detection in high dimensional spaces, *PKDD*, 2002.

[6] M. Breunig, H.-P. Kriegel, R. Ng, J. Sander. LOF: Identifying Density-based Local Outliers, *SIGMOD*, 2000.

[7] H. Daneshmand, A. Lucchi, and T. Hofmann. Starting small–learning with adaptive sample sizes. *ICML*, 2016.

[8] X. Dang, B. Misenkova, I. Assent, R. Ng. Outlier detection with space transformation and spectral analysis. *SDM*, 2013.

[9] F. Keller, E. Muller, K. Bohm. HiCS: High-Contrast Subspaces for Density-based Outlier Ranking. *ICDE*, 2012.

[10] S. Hawkins, H. He, G. Williams, R. Baxter. Outlier detection using replicator neural networks. In *DaWaK*, pages 170–180. Springer, 2002.

[11] E. Knorr, and R. Ng. Algorithms for Mining Distance-based Outliers in Large Datasets. *VLDB* , 1998.

[12] A. Lazarevic, V. Kumar. Feature Bagging for Outlier Detection, *KDD*, 2005.

[13] M. Markou, S. Singh. Novelty detection: a review: part 2: neural network based approaches. *Signal processing*, 83(12), 2003.

[14] E. Muller, M. Schiffer, T. Seidl. Statistical Selection of Relevant Subspace Projections for Outlier Ranking. *ICDE*, 2011.

[15] E. Muller, I. Assent, P. Iglesias, Y. Mulle, K. Bohm. Outlier Ranking via Subspace Analysis in Multiple Views of the Data, *ICDM*, 2012.

[16] V. Nair, G. Hinton. Rectified linear units improve restricted boltzmann machines. *ICML*, 2010.

[17] S. Ramaswamy, R. Rastogi, K. Shim. Efficient Algorithms for Mining Outliers from Large Data Sets. *SIGMOD*, 2000.

[18] S. Sathe, C. Aggarwal. LODES: Local Density Meets Spectral Outlier Detection. *SDM*, 2013.

[19] S. Sathe, C. Aggarwal. Subspace Outlier Detection in Linear Time with Randomized Hashing. *ICDM*, 2016.

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958, 2014.

[21] M. Shyu, S. Chen, K. Sarinnapakorn, L. Chang. A novel anomaly detection scheme based on principal component classifier. *ICDMW*, 2003.

[22] G. Williams, R. Baxter, H. He, S. Hawkins, L. Gu. A Comparative Study of RNN for Outlier Detection in Data Mining. *ICDM* , 2002.

[23] `http://climin.readthedocs.io/en/latest/rmsprop.html`

[24] `http://cs231n.github.io/neural-networks-1/`