Final Project

Outlier Detection for Time Series with Recurrent Autoencoder Ensembles

Michael Gorjaltsan 321031890 gorjalts@post.bgu.ac.il
Matan Goren 206670549 gorenm@post.bgu.ac.il

In this assignment we chose to review the study: *"Outlier Detection for Time Series with Recurrent Autoencoder Ensembles"* by Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S Jensen, posted in IJCAI 19, Aug. 2019. (link)

Introduction

Given time series data we want to obtain the knowledge of which of the data entries are considered anomalies.

In the paper, the researchers propose two solutions to outlier detection in time series based on recurrent autoencoder ensembles. The solutions exploit autoencoders built using sparsely-connected recurrent neural networks (S-RNNs).

The two solutions are ensemble frameworks, specifically an independent framework and a shared framework, both of which combine multiple S-RNN based autoencoders to enable outlier detection. This ensemble-based approach aims to reduce the effects of some autoencoders being overfitted to outliers, this way improving overall detection quality.

The core idea of autoencoders is to compress original input data into a compact, hidden representation and then to reconstruct the input data from the hidden representation. Since the hidden representation is very compact, it is only possible to reconstruct representative features from the input, not the specifics of the input data, including any outliers. This way, the difference, or reconstruction errors, between the original data and the reconstructed data indicate how likely it is that observations in the data are outliers. Next, autoencoder ensembles are used to further improve the accuracy that can be achieved when using a single autoencoder that may often overfit to the original data

Sparsely-connected (Sequence-to-Sequence) RNNs (S-RNN)

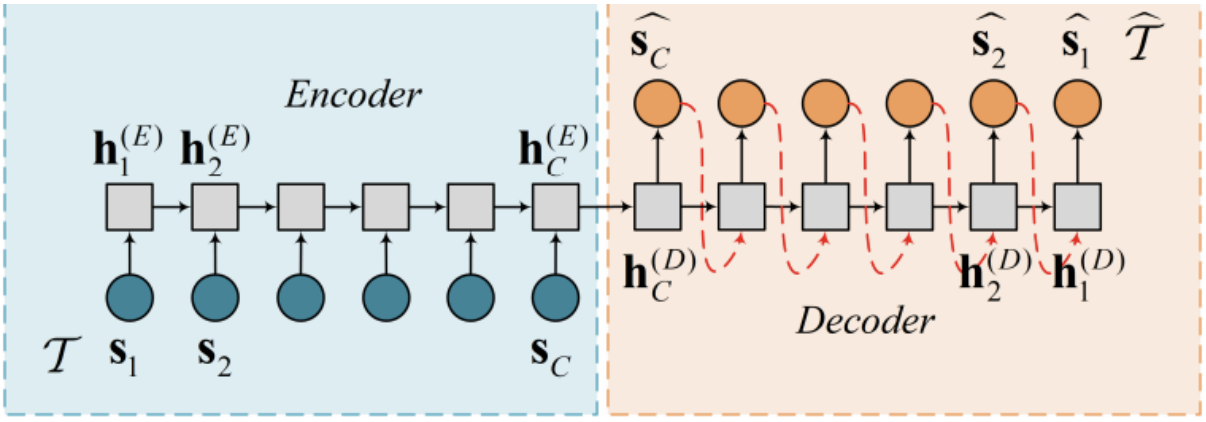


Figure 1:A Sequence-to-Sequence RNN Autoencoder

In the encoder, each vector s_t in time series T is fed into an RNN unit (shown as a square in Figure 1) to perform the following computation.

$$(1)\ h_t^{(E)} = f(s_t, h_{t-1}^{(E)})$$

Here, s_t is the vector at time step t (univariate or multivariate) in the time series and hidden state $h_{t-1}^{(E)}$ is the output of the previous RNN unit at time step $t-1$in the encoder. Next, $f(\cdot)$ is a non-linear function, ranging from the basic tanh or sigmoid to the more sophisticated LSTM or GRU (we used GRU in our implementation).

In the decoder, we reconstruct the time series in reverse order, the last hidden state of the encoder is used as the first hidden state of the decoder. Based on the previous hidden state of the decoder $h_{t+1}^{(D)}$and the previous reconstructed vector $\hat{s}_{t+1}$, we reconstruct the current vector using Equation 2 and also compute the current hidden state using Equation 3, where $f(\cdot)$ and $g(\cdot)$ are again non-linear functions.

$$(2)\ \hat{s}_t = g(\hat{s}_{t+1}, h_{t+1}^{(D)})$$

$$(3)\ h_t^{(D)} = f(\hat{s}_t, h_{t+1}^{(D)})$$

Recurrent Skip Connection Networks (RSCN)

Based on the paper of *Wang and Tian, 2016* (link) in the RSCN model each RNN unit not only consider the previous step but it also consider an additional hidden state L steps before (where L varies from 2 and above)

Formally, we have:

$$\mathbf{h}_t = \frac{f(\mathbf{s}_t, \mathbf{h}_{t-1}) + f'(\mathbf{s}_t, \mathbf{h}_{t-L})}{2}$$

S-RNN With Skip Connection

Now, based on the RSCN model, the researchers in our paper utilized the idea of the skip connection but in more randomized fashion. Meaning, at each step of the autoencoder the connections are selected randomly and some are removed. Specifically, they introduced a sparseness weight vector w_t to control which connection should be removed at each time step t.

Where $w_t = (0, 1)\ or\ (1, 0)\ or\ (1, 1)$ we get the following:

$$\mathbf{h}_t = \frac{f(\mathbf{s}_t, \mathbf{h}_{t-1}) \cdot w_t^{(f)} + f'(\mathbf{s}_t, \mathbf{h}_{t-L}) \cdot w_t^{(f')}}{\|\mathbf{w}_t\|_0}$$

Here, we can see both of the models with L=1 and L=2:

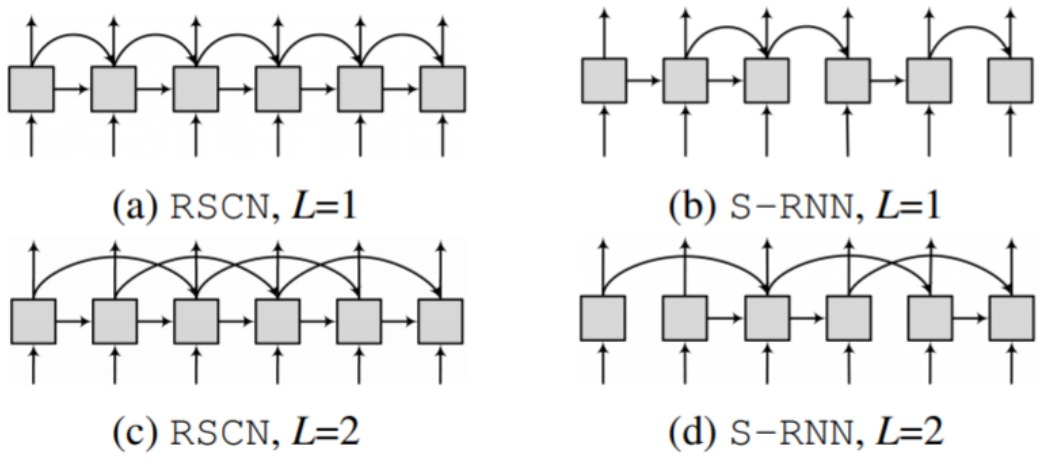


Figure 2: RSCN and S-RNN with skip connection architecture with different values of L (skip size)

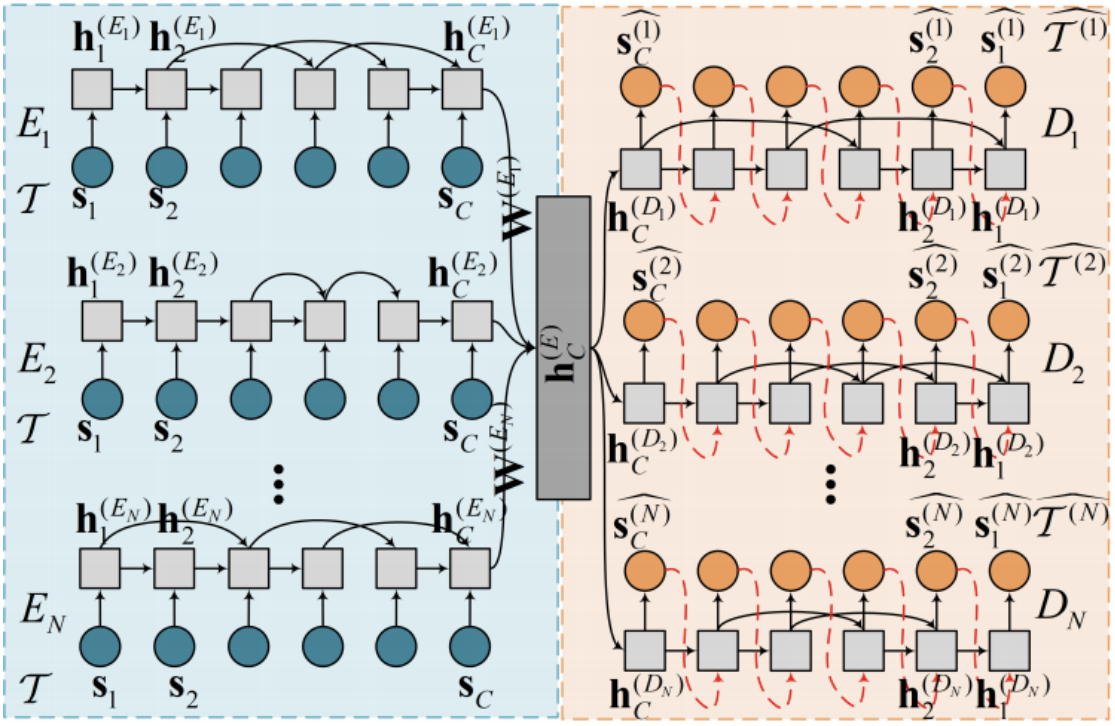
S-RNN Autoencoder Ensembles

To enable an ensemble, the researchers built a set of S-RNN autoencoders, and proposed two different frameworks for integrating the autoencoders into ensembles, Independent Framework and Shared Framework. In both of the frameworks the researchers presented a set of autoencoders where each autoencoder has a uniformly chosen skip size.

In the paper it stated that the Shared Framework model got better performance overall, so we will present only it and build our improved model based on it.

Shared Framework

The basic framework trains different autoencoders independently, meaning that different autoencoders do not interact during the training phase. However, since all autoencoders try to reconstruct the same, original time series, it is relevant to enable interactions among the autoencoders.



The shared framework uses a shared layer, denoted as $h_C^{(E)}$, to concatenate the linear combinations (using linear weight matrices $W^{(E_i)}$) of all the last hidden states of all the encoders. Formally, we have:

$$(4)\ h_{C}^{(E)} = concatenate(h_{C}^{(E_1)} \cdot W^{(E_1)}, \ldots, h_{C}^{(E_N)} \cdot W^{(E_N)})$$

Each decoder D_i employs the concatenated hidden states $h_{C}^{(E)}$ as the initial hidden state when reconstructing time series. In the shared framework, all autoencoders are trained jointly by minimizing the objective function $\boldsymbol{\mathcal{J}}$ that sums up the reconstruction errors of all autoencoders and an L1 regularization term on the shared hidden state:

$$(5)\ \boldsymbol{\mathcal{J}} = \sum_{i=1}^{N} \boldsymbol{\mathcal{J}}_i + \lambda \left\| h_{C}^{(E)} \right\|_1 = \sum_{i=1}^{N} \sum_{t=1}^{C} \left\| s_t - \hat{s}_t^{(D_i)} \right\|_2^2 + \lambda \left\| h_{C}^{(E)} \right\|_1$$

- λ is a weight that controls the importance of the L1 regularization $\left\| h_{C}^{(E)} \right\|_1$.
- $\hat{s}_t^{(D_i)}$ t denotes the reconstructed vector at time step t from decoder D_i, and $||\cdot||_2$ is the L2-norm of a vector.

Ensemble Outlier Scoring

Recall that we have N autoencoders that reconstruct the original time series. Thus, we obtain N reconstructed time series. For each vector we compute N reconstruction errors and use the median of the N errors as the final outlier score of the vector. Then we can define a threshold for an outlier or calculate the PR-AUC or ROC-AUC scores (As they did in the paper).
(Using median instead of mean reduces the influence of the reconstruction errors from the autoencoders that overfit to the original time series).

For accuracy measurement we defined a prior fixed threshold of 0.5 for a vector to be considered as anomaly.

S-RNN Ensemble Model Advantages

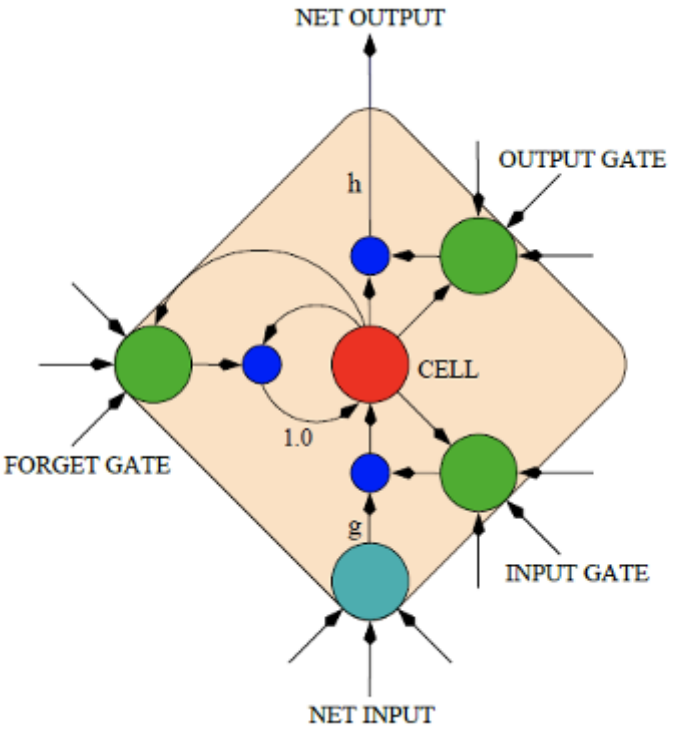
First, the main advantage of this model is that it is an ensemble, which enables more variance in the components in the training process that leads to less overfitting and more accurate results. Moreover, the paper model is generic and does not require any preprocessing stages to prepare the data to a specific form.

S-RNN Ensemble Model Disadvantages

The main disadvantage we could see in the model is regarding the randomness of the skip connection size (and our suggested improvement will mainly focus on that, as you will read). The random skip connection size selection can cause some levels in the ensemble to have the same skip size and in some sort will prevent the model to reach its full capabilities. In addition,

The State-of-the-art Baseline Model: LSTM Autoencoder

An LSTM is a special kind of RNN architecture, capable of learning long-term dependencies. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. An LSTM Autoencoder is an implementation of an autoencoder for sequence data using an Encoder-Decoder LSTM architecture.



The LSTM architecture is considered a state of the art architecture of anomaly detection for sequence data, and the natural choice was to compare our algorithm to another encoder based model.

We build a LSTM-AE with the following sequential structure:

```
model = keras.Sequential(
    [
        layers.InputLayer(input_shape=(X_train.shape[1], X_train.shape[2])),
        LSTM(64, return_sequences=True),
        LSTM(32),
        layers.RepeatVector(X_train.shape[1]),
        LSTM(32, return_sequences=True),
        LSTM(64),
        layers.Dense(X_train.shape[1] *  X_train.shape[2]),
        layers.Reshape([X_train.shape[1], X_train.shape[2]])
    ]
)
```

Our Model: Linear Incremental S-RNN (LIS-RNN)

Our main contribution to the ensemble model is in the case of defining the skip size at each level of the autoencoders ensemble. As aforementioned, in the paper's model, the skip size is being chosen uniformly. We decided to use a linear approach. In our model, ***L**inear **I**ncremental **S-RNN*** (LIS-RNN), we give each encoder a different skip size with incremental order, i.e, the $i'th$ level has skip size of i.

We decided to perform this change for the following reasons:

1. Using higher skip size helps the autoencoder learn the lower resolution features of the training data, and lower skip size helps learn the higher resolution features.
   We wanted to have more control on the training process instead of using the random approach as proposed by the original paper.
2. When using a large ensemble it is likely that due to the random choice of the skip size some different levels in the ensemble will have the same skip size. Therefore, the only difference between them only relies on the randomized connection selection. Which might lead to lack of variance in the ensemble, and can lead to more overfitting and harm the learning process.
3. In the original paper, due to the random selection of the skip size, the encoder part and the decoder part of an autoencoder at a specific level could have a different skip size. Which may lead to inferior performance in the training process. We decided to unify the skip size to both parts of the autoencoder.

In addition, we applied one more change regarding the shared representation. In the original paper, all of the decoders were fed with the same representation denoted as $h_C^{(E)}$ (recall, this was a weighted concatenation of the encoder's output). We decided to apply another concatenation between $h_C^{(E)}$ and the current hidden state at the level. This change was applied under the reasoning that for a specific level at the ensemble we want to connect the global shared representation to the specific representation in that level.

Lastly, we noticed that applying a relu activation function on the concatenation of the hidden states before feeding it to the decoder resulted in better performance.

Datasets

In our project we focused on anomaly detection, therefore, none of the datasets provided fit our purpose. We decided to rely on outside sources and to use well known datasets for anomaly detection.

We took, among others, datasets from the following data repositories:

- **Yahoo S5**: real and synthetic time series with labeled anomaly points. The dataset tests the detection of various anomaly types, including outliers and change-points. The synthetic data contains time series with varying trend, noise, and seasonality while the real data represents the metrics of various Yahoo services.
- **Power Demand:** contains one year of power consumption records measured by a Dutch research facility in 1997.
- **ECG:** Contains anomalous beats from electrocardiograms.
- **2D Gesture:** Contains time series of X-Y coordinates of an actor's right hand. The data is extracted from a video in which the actor grabs a gun from his hip-mounted holster, moves it to the target, and returns it to the holster. The anomalous region is in the area where the actor fails to return his gun to the holster.

Also, we used the following datasets:

- **cardio_blood.csv**
- **cardio_veins.csv**
- **cardio.csv**
- **chess_krkopt_zerovsall.csv**
- **cmc-nominal.csv**

Results

For this section we present the training performance results of each model for each dataset, not including additional hyperparameters selection results and time measurement results, due to lack of space (the full result table is provided in the project repository).

LSTM-AE

dataset	f1	pr_auc	roc_auc	tpr	fpr	precision	accuracy
2d-gesture.csv	0.196246	0.176303	0.240095	0.371649	0.622286	0.166792	0.519248
cardio.csv	0.316434	0.362671	0.816360	0.733992	0.491962	0.356641	0.838762
cardio_blood.csv	0.314148	0.327059	0.833985	0.716468	0.435641	0.354681	0.825762
cardio_veins.csv	0.346360	0.436399	0.905335	0.752048	0.434802	0.384340	0.909524
chess_krkopt_zerovsall.csv	0.005049	0.003864	0.150198	0.224073	0.360208	0.002567	0.876770
chfdb_chf01_275.csv	0.406019	0.250506	0.434127	0.744358	0.773902	0.322298	0.430000
chfdb_chf13_45590.csv	0.491489	0.220139	0.345833	0.896567	0.930722	0.356977	0.407143
chfdbchf15.csv	0.146318	0.116505	0.462010	0.505150	0.534474	0.093141	0.863158
cmc-nominal.csv	0.101916	0.075055	0.170318	0.272784	0.539889	0.072222	0.554990
ltstdb_20221_43.csv	0.462794	0.233194	0.312500	0.733200	0.832475	0.376417	0.433333
ltstdb_20321_240.csv	0.306416	0.244484	0.285000	0.526200	0.680017	0.228001	0.609571
mitdb__100_180.csv	0.246002	0.231937	0.497500	0.425367	0.446873	0.225672	0.686830
power_data.csv	0.164404	0.149990	0.200408	0.364369	0.547415	0.118400	0.569122
power_data_ecc.csv	0.130196	0.136247	0.180540	0.268915	0.497419	0.095021	0.420802
real_17.csv	0.609159	0.903516	0.989831	0.525511	0.016483	0.986580	0.656250
real_18.csv	0.638946	0.917299	1.000000	0.561294	0.011675	0.989761	0.696875
real_19.csv	0.569202	0.913312	0.997468	0.484551	0.029168	0.975711	0.725000
real_32.csv	0.630141	0.757778	0.996429	0.583777	0.008911	0.757342	0.972759
stdb_308_0.csv	0.095825	0.073201	0.256818	0.341433	0.536777	0.056674	0.708718
xmitdb_x108_0.csv	0.374416	0.207088	0.165000	0.622283	0.819970	0.276212	0.442063

Base Model: S-RNN

dataset	f1	pr_auc	roc_auc	tpr	fpr	precision	accuracy
2d-gesture.csv	0.294027	0.176180	0.272411	0.674876	0.784841	0.190772	0.644118
cardio.csv	0.118846	0.086578	0.540020	0.558428	0.553030	0.075986	0.909091
cardio_blood.csv	0.072448	0.072984	0.392165	0.331667	0.422441	0.044375	0.906818
cardio_veins.csv	0.211495	0.219359	0.788935	0.663797	0.459233	0.149696	0.896818
chess_krkopt_zerovsall.csv	0.004588	0.003023	0.350176	0.395997	0.469319	0.002313	0.992286
chfdb_chf01_275.csv	0.514200	0.456448	0.788194	0.943933	0.858450	0.395099	0.301667
chfdb_chf13_45590.csv	0.635420	0.460556	0.900000	0.949167	0.842775	0.515482	0.804762
chfdbchf15.csv	0.401702	0.366895	0.650000	0.774800	0.569937	0.397163	0.886000
cmc-nominal.csv	0.123782	0.040152	0.220609	0.623163	0.845564	0.070862	0.099338
ltstdb_20221_43.csv	0.577944	0.295278	0.322222	0.906433	0.950767	0.432607	0.360000
ltstdb_20321_240.csv	0.287577	0.218730	0.186667	0.524733	0.767922	0.203219	0.517857
mitdb__100_180.csv	0.260794	0.211638	0.485000	0.469367	0.478365	0.200964	0.748297
power_data.csv	0.129700	0.114045	0.204591	0.370892	0.609296	0.082099	0.428157
power_data_ecc.csv	0.125416	0.106562	0.201434	0.356956	0.601649	0.079627	0.390096
real_17.csv	0.738702	0.880115	1.000000	0.679058	0.017532	0.979491	0.817647
real_18.csv	0.704653	0.878024	0.997252	0.630634	0.010468	0.986054	0.802941
real_19.csv	0.679371	0.867094	0.991827	0.584430	0.018261	0.975841	0.862824
real_32.csv	0.854420	0.683542	0.995600	0.918272	0.028937	0.839426	0.973529
stdb_308_0.csv	0.280659	0.090423	0.375758	0.785167	0.828433	0.172191	0.728333
xmitdb_x108_0.csv	0.584489	0.377958	0.485833	0.833128	0.825890	0.495111	0.265675

Our Model: LIS-RNN

dataset	f1	pr_auc	roc_auc	tpr	fpr	precision	accuracy
2d-gesture.csv	0.315370	0.186073	0.316771	0.716444	0.803216	0.207263	0.630882
cardio.csv	0.131228	0.106490	0.550302	0.599705	0.579525	0.078021	0.927091
cardio_blood.csv	0.050953	0.052446	0.241545	0.253862	0.437240	0.030197	0.933545
cardio_veins.csv	0.245432	0.289335	0.809543	0.673357	0.458221	0.207436	0.920545
chess_krkopt_zerovsall.csv	0.003670	0.002022	0.203616	0.262630	0.445886	0.001852	0.958920
chfdb_chf01_275.csv	0.483887	0.323330	0.517460	0.905583	0.883621	0.366261	0.543833
chfdb_chf13_45590.csv	0.608665	0.450000	0.853889	0.950300	0.848583	0.480470	0.833476
chfdbchf15.csv	0.447214	0.258949	0.689792	0.772550	0.508238	0.449263	0.924263
cmc-nominal.csv	0.129058	0.047895	0.144102	0.542910	0.843226	0.076933	0.090486
ltstdb_20221_43.csv	0.555168	0.282500	0.272222	0.864283	0.927083	0.416038	0.525000
ltstdb_20321_240.csv	0.263935	0.207738	0.175833	0.472783	0.785673	0.188153	0.489286
mitdb__100_180.csv	0.295440	0.254399	0.543333	0.525350	0.467237	0.219478	0.776410
power_data.csv	0.147519	0.123346	0.178321	0.392653	0.682258	0.095657	0.299759
power_data_ecc.csv	0.139160	0.107612	0.223426	0.427361	0.647338	0.084902	0.569486
real_17.csv	0.727282	0.880992	1.000000	0.672129	0.019256	0.977701	0.832353
real_18.csv	0.682786	0.879538	0.998494	0.609195	0.010815	0.985618	0.834706
real_19.csv	0.618160	0.870224	0.998071	0.517547	0.014367	0.981164	0.817647
real_32.csv	0.817184	0.625000	0.991882	0.906375	0.031352	0.783593	0.976410
stdb_308_0.csv	0.286978	0.119646	0.374068	0.784733	0.834856	0.178326	0.767917
xmitdb_x108_0.csv	0.581334	0.385933	0.520000	0.848145	0.830722	0.491241	0.322619

Statistical Significance Testing Of The Results

We first performed the Friedman test using `stats.friedmanchisquare` from the `scipy` package.

We tested on all of the metrics and the results are as follows:

Metric	Statistic	P-value
tpr	11.200000000000017	0.0036978637164828986
fpr	11.700000000000017	0.0028798991580882183
precision	0.10000000000002274	0.9512294245007031
accuracy	8.400000000000006	0.01499557682047766
roc-auc	0.700000000000017	0.7046880897187074
pr-auc	3.6000000000000227	0.16529888822158464

As we can see in the above results, for the metrics tpr, fpr and accuracy we can conclude that the three algorithms are statistically significant.

Furthermore, we performed the Nemenyi test on those metrics.

Group 0: Base model LSTM-AE results.

Group 1: Paper's model S-RNN results.

Group 2: Our model LIS-RNN results.

Accuracy

	0	1	2
0	1.000000	0.600820	0.012315
1	0.600820	1.000000	0.139445
2	0.012315	0.139445	1.000000

tpr

	0	1	2
0	1.000000	0.004467	0.030670
1	0.004467	1.000000	0.781714
2	0.030670	0.781714	1.000000

fpr

	0	1	2
0	1.000000	0.046567	0.002589
1	0.046567	1.000000	0.600820
2	0.002589	0.600820	1.000000

We can clearly see in all three metrics that the p-value between the LSTM-AE algorithm and our LIS-RNN algorithm is lower than the p-value between the LSTM-AE algorithm and the paper's S-RNN algorithm.

Conclusions

In this project we have suggested an improved version for a S-RNN anomaly detection model. Our improvement mainly tackled the problem with the randomized approach that was presented in the original paper. By performing extensive experiments on 20 different datasets we could see a major improvement regarding the accuracy of the algorithm.

Our improvement does not require any extra preprocessing stage or extra running time and has no overhead regarding the training phase and still achieves better performance and better accuracy.

Our suggested improvement took advantage of the autoencoder architecture and utilized the idea of skip connection in order to enable the ensemble to learn features in different resolutions by using linear incremental skip size for each level of the ensemble.

By performing the Friedman test and post-hoc evaluation we came to a statistical understanding of our algorithm significance with respect to the others. We could clearly see a major significance in the accuracy metric. But yet, in all of the metrics, our algorithm has achieved more statistical significance than the paper's algorithm.