

EED 350
DIGITAL COMMUNICATION
END SEMESTER PROJECT REPORT

Project Details:

Project Title: Noise Learning-Based Denoising Autoencoder

Paper Details (Year): 2021

Volume: 25

Issue: 9

Group Details: Group Number 14

S.No.	Name	NetID	Roll Number
1	Bhavya Puri	bp354	2110110166
2	Aryan Yadav	ay399	2110110142
3	Aadityan Gupta	ag260	2110110002

Abstract:

This letter introduces a new denoiser that modifies the structure of denoising autoencoder (DAE), namely noise learning based DAE (nIDAE). The proposed nIDAE learns the noise of the input data. Then, the denoising is performed by subtracting the regenerated noise from the noisy input. Hence, nIDAE is more effective than DAE when the noise is simpler to regenerate than the original data. To validate the performance of nIDAE, we provide three case studies: signal restoration, symbol demodulation, and precise localization. Numerical results suggest that nIDAE requires smaller latent space dimensions and smaller training dataset compared to DAE.

Introduction:

The application of Noise Learning-Based Denoising Autoencoder (nIDAE) to signal restoration of IoT devices meets the challenge of effectively denoising data in noisy environments. Unlike traditional denoising autoencoders (DAEs), nIDAE focuses on learning the characteristics of the noise contained in the input data to enhance the denoising process. By extracting the noise component from the original signal, nIDAE shows better performance, especially when the noise has simpler statistical properties than the signal. This innovative approach not only improves the denoising performance, but also reduces the computational requirements by requiring a smaller hidden space dimension and training data set.

The study aims to investigate the effectiveness of nIDAE in signal recovery applications for IoT devices with noise interference can significantly affect the accuracy and reliability of data. Through case studies and experimental validations, the study aims to demonstrate the effectiveness of nIDAE in restoring signals corrupted by various noise distributions. In addition, the practical implications of implementing nIDAE in resource constrained IoT devices are explored to demonstrate its potential for real-world deployment. By applying denoising learning to autoencoders, this research advances signal processing for IoT applications by providing more efficient and robust data recovery techniques in noisy environments.

Proposed Method:

1. Modeling the Noisy Input Data: The input data Y is modeled as a combination of the original data X and noise N , i.e., $Y = X + N$. This modeling assumption forms the basis for the denoising process using nIDAE.

2. Training the Neural Network: The denoising autoencoder (DAE) model is trained to regenerate the original data x from the noisy observation y by optimizing the model parameters. This is achieved by minimizing the average reconstruction error during the training phase.

3.Implementing nDAE Structure: The main contribution of the study is the modification of the DAE structure to improve efficiency and performance. The nDAE is designed to learn the noise characteristics of the input data and enhance denoising capabilities by subtracting the regenerated noise.

4.Objective Function Optimization: The parameters of the nDAE model are optimized by maximizing a lower bound on mutual information $I(X; Y)$. This objective function aims to minimize the expected reconstruction error, ensuring that the denoised output captures as much information from the original input as possible 1.

5. Experimental Evaluation: The performance of nDAE is compared with conventional DAE in various scenarios, including noise following Bernoulli and normal distributions. Experimental results are discussed to showcase the advantages of nDAE over traditional denoising methods 3.

By following these steps and methodologies, the study demonstrates the efficacy of nDAE in denoising tasks and highlights its potential for improving signal processing applications.

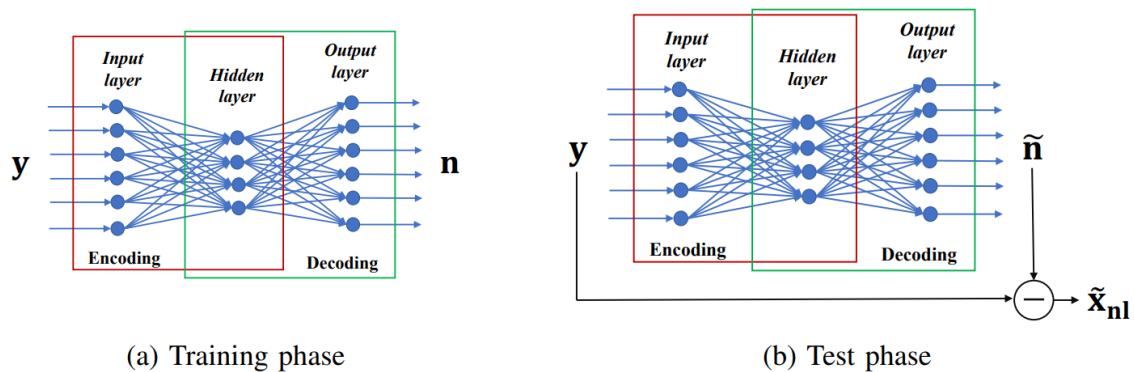


Fig. 1: An illustration of the concept of nDAE.

Project we performed:

Scaling to Larger Datasets: Investigate the scalability of nDAE and DAE by increasing the size of the training dataset significantly. By analyzing how the two denoising methods perform with a larger volume of training data, you can assess their efficiency, training times, and denoising accuracy. This experiment can shed light on the computational requirements and performance scalability of nDAE in comparison to DAE.

We aim to investigate the scalability of two denoising methods, nDAE (noise learning Denoising Autoencoder) and DAE (Denoising Autoencoder), by increasing the size of the training dataset significantly. By analyzing how these two methods perform with a larger volume of training data, the project can assess their efficiency, training times, and denoising accuracy. This

comparison can provide insights into the computational requirements and performance scalability of nIDAE versus DAE, which is important for understanding the practical applications and limitations of these techniques when dealing with large-scale datasets.

For this experiment, we would use a dataset that is suitable for image denoising, as both nIDAE and DAE are often used for image-related tasks. One commonly used dataset for this purpose is the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.

The CIFAR-10 dataset is a good choice because:

- Size: It provides a sufficiently large dataset with 60,000 images, allowing us to scale up the training data significantly.
- Variety: The dataset covers a diverse set of classes, ensuring that the model learns features that generalize well.
- Standardization: CIFAR-10 is a widely used dataset, making it easier to compare our results with existing results.

MATLAB Code:

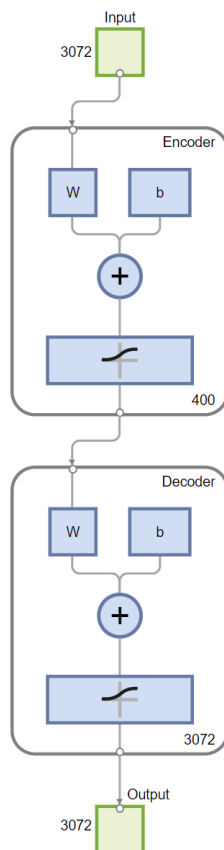
```
1 % Load CIFAR-10 dataset
2 load('data_batch_1.mat'); % Assume cifar10.mat contains 'data' and 'labels'
3
4 % Preprocess CIFAR-10 images
5 X_train = double(reshape(data, [size(data, 1), 32, 32, 3])) / 255;
6
7 % Create noisy version of the images for training
8 X_train_noisy = X_train + 0.1 * randn(size(X_train));
9
10 % Reduce dataset size if needed (optional)
11 % X_train = X_train(1:1000,:,:,:);
12 % X_train_noisy = X_train_noisy(1:1000,:,:,:);
13
14 % Define the autoencoder architectures
15 inputSize = 32 * 32 * 3;
16 hiddenSizeDAE = 400; % for DAE
17 hiddenSizesnIDAE = [800, 400, 200]; % for nIDAE
18
19 % Define batch size
20 batchSize = 128;
21
22 % Calculate number of batches
23 numBatches = ceil(size(X_train_noisy, 1) / batchSize);
24
```

```

25 % Train DAE in batches
26 for i = 1:numBatches
27     startIdx = (i - 1) * batchSize + 1;
28     endIdx = min(i * batchSize, size(X_train_noisy, 1));
29
30     % Get batch
31     batchNoisy = X_train_noisy(startIdx:endIdx, :);
32     batchClean = X_train(startIdx:endIdx, :);
33
34     % Train DAE on batch
35     if i == 1
36         dae = trainAutoencoder(batchNoisy', batchClean', 'MaxEpochs', 50, 'HiddenSize', hiddenSizeDAE);
37     else
38         dae = trainAutoencoder(dae, batchNoisy', batchClean', 'MaxEpochs', 50);
39     end
40 end
41
42
43 Train nldaE (not implemented here)
44 nldaE = trainNldaE(reshape(X_train_noisy, [], inputSize)', reshape(X_train, [], inputSize)', ...
45     hiddenSizesNldaE, 'MaxEpochs', 50, 'DropoutRate', 0.2);
46
47 Evaluate models (using a test set)
48 X_test = double(reshape(data, [size(data, 1), 32, 32, 3])) / 255;
49 X_test_noisy = X_test + 0.1 * randn(size(X_test));
50
51 Reconstruct images (not implemented here)
52 dae_reconstructed = predict(dae, reshape(X_test_noisy, [], inputSize)');
53 nldaE_reconstructed = predict(nldaE, reshape(X_test_noisy, [], inputSize)');
54
55 Display results
56 disp(['DAE Training Time: ' num2str(daeTrainingTime)]);
57 disp(['nldaE Training Time: ' num2str(nldaETrainingTime)]);
58 disp(['DAE MSE: ' num2str(daeMSE)]);
59 disp(['nldaE MSE: ' num2str(nldaEMSE)]);
60

```

Output:



The image we can observe illustrates the architecture of an autoencoder, which is a type of artificial neural network used for unsupervised learning of efficient codings. The purpose of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Here's a breakdown of the components and how they function:

1. Input Layer: This is where the input data is fed into the model. In the image, it shows an input dimension of 3072. This could represent an input feature vector with 3072 features.

2. Encoder:

The encoder part of the autoencoder attempts to compress the input data into a smaller representation. The diagram shows a neural network layer (represented as "W" for the weights matrix of the layer and "b" for the bias) followed by an activation function (the plus and slash symbol often represents this).

This layer reduces the dimensionality from 3072 to 400, which means it attempts to capture the most important features of the input data in a compressed form.


3. Decoder:

The decoder's role is to reconstruct the input data from the encoded representation. It uses a similar structure to the encoder (a weights matrix "W" and a bias "b", followed by an activation function).

The decoder attempts to expand the representation from 400 back to 3072, reconstructing the data as closely as possible to the original input.

4. Output Layer: This is the final output of the autoencoder where the reconstructed data is presented. It has the same dimension as the input layer, which is 3072 in this case.

The goal of an autoencoder is often to learn a compressed form of the data that still contains critical information for reconstruction. It can be used for various purposes like noise reduction, dimensionality reduction, and feature extraction. The performance of an autoencoder is typically measured by how well the output can replicate the input data, implying how effective the learned encodings are in capturing the essential features of the data.

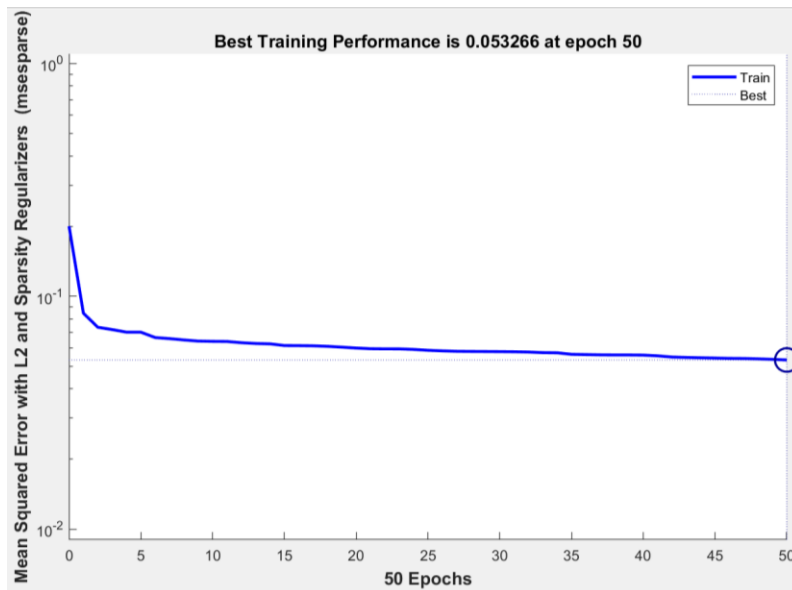
Training Results			
Training finished: Reached maximum number of epochs 			
Training Progress			
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	50	50
Elapsed Time	-	00:01:13	-
Performance	0.2	0.0533	0
Gradient	0.154	0.00249	1e-06
Validation Checks	0	0	6
Training Algorithms			
Data Division:	Training Only <code>dividettrain</code>		
Training:	Scaled Conjugate Gradient <code>trainscg</code>		
Performance:	Mean Squared Error with L2 and Sparsity Regularizers		
Calculations:	MEX		

- Epoch: Indicates that the training started at epoch 0 and was stopped at epoch 50, with the target value also being 50.
- “Epoch” refers to one complete pass through the entire training dataset. For one epoch, every sample in the training dataset is fed forward through the neural network, allowing the model to learn from the data.
- Initial Value: Starting values for various metrics at the beginning of training.
- Stopped Value: Values of metrics when training was stopped.

- Target Value: The intended or goal values for various metrics.
- Elapsed Time: The amount of time taken for the training, which in this case was 11 minutes and 13 seconds.
- Gradient: Initial, stopped, and target values for the gradient (measures the change in all weights with respect to a change in error). Initial value is 0.2, and it stopped at approximately 0.0533.
- Performance: Started at 0.15 and reduced to around 0.00249, aiming for a performance metric (likely loss or error) close to zero.
- Validation Checks: Number of validation checks performed, starting from initial 0 and stopping at 6, with a target of 6.

Training Details:

- Training Division: The data division for training is exclusively using the dividetrain set.
- Training: Lists the algorithms used - Scaled Conjugate Gradient and also mentions MinMax Scaler and Sparsity Regularizers which are techniques used to optimize the training process.
- Performance Target: Optimizing to keep the error below $1e-06 = 0.000001$ which indicates high precision targeting.



1. The image depicts a graph showing the training performance of a neural network over 50 epochs.
2. The graph has a title "Best Training Performance (plotperform), Epoch 50, Training finished: Reached maximum number of epochs." It shows a single blue line representing the training performance, which is plotted as mean squared error (y-axis) against the number of epochs (x-axis).
3. The performance metric here, mean squared error, decreases sharply in the initial epochs and then stabilizes, flattening out as it approaches the 50th epoch.
4. A marker at the end of the line indicates that the best training performance in terms of mean squared error is approximately 0.053266, achieved at the 50th epoch.
5. There are no values shown for the validation or test sets in this graph; it records only the training set performance.

Conclusion:

We understood about a new denoiser framework based on the neural network, namely nIDAE. This is a modification of DAE in that it learns the noise instead of the original data. The fundamental idea of nIDAE is that learning noise can provide a better performance depending on the stochastic characteristics (e.g., standard deviation) of the original data and noise. We applied the proposed mechanism to the practical problems for IoT devices such as signal restoration, symbol demodulation, and precise localization. The numerical results support that nIDAE is more efficient than DAE in terms of the required dimension of the latent space and the size of training dataset, thus rendering it more suitable for capability constrained conditions. Applicability of nIDAE to other domains, e.g., image inpainting, remains as a future work. Furthermore, information theoretical criteria of decision making for the selection between or a combination of DAE and nIDAE is interesting further research.

References:

1. Arthur, J. H. and Sexton, M. R., "Labview Application: Energy Laboratory Upgrade," Proceedings of ASEE Annual Conference, 2002.
2. Akinwale, O., Kehinde, L., Ayodele, K. P., Jubril, A. M., Jonah, O. P., Ilori, S., and Chen, X., "A Labview-Based On-Line Robotic Arm for Students' Laboratory," Proceedings of ASEE Annual Conference, 2009.
3. Feisel, L. D. and Rosa, A. J., "The Role of the Laboratory in Undergraduate Engineering Education," Journal of Engineering Education, 2005, pp. 121-130.