Exploring Data Compression Using Deep Learning for Use in Wireless Sensor Networks and IoT Networks

ECE 659/493: IoT Signal Processing and Intelligent Sensor Networks, Instructor: Dr. Otman A. Basir

Juhi Vasudev Bachani (20979706), Akashdeep Singh Khehra (20988007), Seoyoung Sim (20993717)

Electrical and Computer Engineering

University of Waterloo, Waterloo, Ontario

{jvbachani, as3khehra, y3sim}@uwaterloo.ca

Abstract — Internet of Things (IoT) is a technology that connects devices of different types and characteristics through a network.[1] As it connects a huge number of different devices, it generates massive and varied data that need to be processed and responded to within a very short time. This situation has created an important issue that needs to be addressed in dealing with a massive amount of data from sensors connected to IoT. One of the major challenges of this issue is the high energy consumption due to the transmission of data to the cloud. In this circumstance, compressing data on the base station before transferring data to the cloud could be a solution. It can reduce the usage of storage in the cloud rather than uncompressed data. Also, edge computing allows the workload to be offloaded from the cloud at a location closer to the source of data that need to be processed while saving time, improving privacy, and reducing network traffic.[2]

Index Terms—Internet of Things (IoT), Long Short-Term Memory (LSTM) , Data Compression

I. INTRODUCTION

The Internet of Things (IoT) describes the network of physical objects that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet.[3] It is basically connection between the networks which includes devices that are connected with each other and allows connection between them as well as the cloud. This process can be done without the interference of computers or humans. In IoT, there is a connection between large number of devices from around the world, this leads to an increased consumption of storage or data. As the data is stored in the cloud, so there occurs a storage problem for the excess amount of the data.

Even in case of Wireless Sensor Networks (WSN), as a result of their widespread usage across numerous applications, more bandwidth is needed for the transmission of the sequential data such as text, videos, etc. The sensor nodes which are used to collect data is of very limited capabilities.

In these types of situations, in order to reduce the network traffic, data needs to be compressed on the base station before it gets transferred to the cloud. This will reduce the bandwidth of data to be sent, increase the speed of transmission, and ultimately use less amount of storage on cloud. All the data compression can be done by implementing the deep learning techniques in wireless sensor networks. Techniques such as CNN- based compressive autoencoders have been proposed and are used in the project in order to reduce the dimensions of the data compression. In IOT, majority of data gathered is either sequential, relational or in the form of images. The goal of the project or the research is to determine deep learning -based compression methods with a particular emphasis on high quality or dimensional data. LSTM auto-encoder has been used for this purpose which uses Encoder- Decoder architecture for the sequential data.

II. TYPES OF DATA IN IOT

IoT consumer gadgets including security systems, smart appliances, smart TVs, and wearable health metres are collected via internal sensors. Additionally, information is gathered from commercially available devices including commercial security systems, traffic monitoring equipment, and weather tracking systems. The information is communicated, saved, and accessible at any moment.[4] The types of IoT data are classified as follows [5]-:

2.1 Location Data

Location data conveys the physical location of the system or device. It is often employed in manufacturing, warehousing, and logistics. In case of location data location can be considered as an indoor global positioning system.[4] It makes possible for one to track goods, pallets or equipment in real time rather than sending one to a specific location. It can be helpful in several ways such as- a shop supervisor can find any specific pallet of parts on the shop floor. Even a farmer can track the equipment during the time of harvest.

2.2 Status Data

Status data is fundamental, unprocessed information that conveys a device's or system's status. It is the most basic type of IoT data. The majority of IoT devices produce status data, which is gathered as raw data and used for more intricate analysis.

2.3 Automation Data

Automatic systems and gadgets, like smart thermostats and automated lights, produce this kind of data. The task of someone without automation would be to remember to set the thermostat twice daily, turning out all the lights for the last person to be out of the room.

III. DATA COMPRESSION

Encoding, reorganising, or otherwise altering data to make it smaller is known as data compression. It is basically done by encoding the information again and by using lesser number of bits than the original representation.[6] A program performs compression by using functions or an algorithm to determine how to efficiently reduce the amount of the data. Data compression can be characterised into two types:

3.1 Lossy compression

Lossy compression can produce substantially higher compression ratios, but at the potential expense of file quality degradation. It reduces size by removing extraneous information and simplifying the remaining information. It intentionally discards some data during the compression process.[6] Lossy data compression is advantageous for digital multimedia such as audio, video, photographs, and streaming data. By using lossy compression, the volumes of several types of data can be greatly reduced, allowing for easy Internet dissemination or offline use.

3.2 Lossless compression

This method prevents any information from being really erased. Because no data is lost during lossless compression, it frequently results in smaller compression ratios. As lossy compression loses the data whereas lossless compression is used to maintain the data.[7] It locates and eliminates statistical redundancies to remove bits. When data is compressed without any loss, the original dataset can be completely reconstructed if the compression process is reversed.[8] So, it is different from lossy compression as in that case, there is the reversing process that results in the data loss. It is also named as Lossless audio compression. A set of advanced algorithms is required in order to achieve Lossless compression.

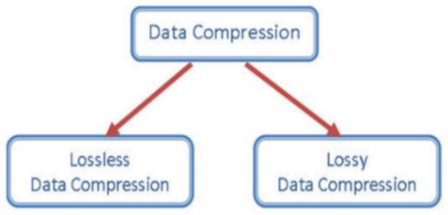


Fig. 1. Data Compression [9]

IV. DIMENSIONALITY REDUCTION

4.1 Dimensionality Reduction (DR)

In machine learning, there are number of features that are considered for classification. As the number of features in the dataset is high, it will be difficult to visualize the dataset. And most of the features are interconnected with each other, so it is baseless to keep all the features. To improve the visualization in ML, we use the dimensionality reduction technique to reduce the number of features. Hence by compressing data, the storage space will also be reduced. Also, DR is a useful technique to remove redundant features.

Dimensionality Reduction is divided into two parts. 1) Feature Selection and, 2) Feature Extraction. In Feature Selection, we select the subset of original data based on features/variables, and in Feature Extraction, we reduce the dimensionality of data in high dimensional space to lower dimension space.

There are Various Methods for Dimensionality Reduction:

A) Principal Component Analysis (PCA)
B) Linear Discriminant Analysis (LDA)
C) Generalized Discriminant Analysis (GDA)
D) Autoencoder.

Depending upon the type of input we will select the DR method.

4.2 Deep Learning Techniques for Dimensionality Reduction

Deep Learning is a machine learning technique that is very useful in today's world. In deep learning, models learn to perform classification.[17] Models are trained by using a large set of labeled data and neural network architectures that contain many layers.[17] It includes the number of neurons in each layer which are connected to each other such that the first layer is the input layer, the last layer is the output layer and in between, there is a hidden layer where most of the preprocessing of data take place. Some of the most popular deep learning techniques are Convolutional neural networks and Recurrent neural networks.

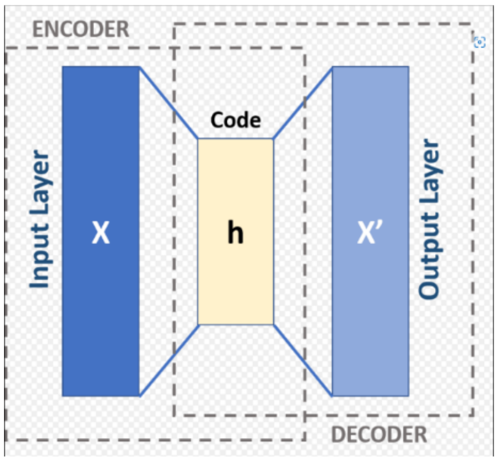


Fig:2 Autoencoder Structure

Autoencoders are a type of neural network which are used for unsupervised learning techniques. As shown in Fig:1, it encodes the input data in a compressed form or in reduced dimensionality and then the decoder decodes the compressed output to the original form. Input and output of Autoencoder have the same dimensionality as there is minimum loss of data. The output will be almost similar to the input form. The section shown as code in Fig:1 is actually the hidden layer which is mainly a part where dimensionality reduction is taking place.

An Autoencoder has the following parts:

A) Encoder – It is a part of the network which takes the input and produces the output in lower dimensionality.
B) Code – It is a part of the network which gives the lower dimensionality output of the encoder.
C) Decoder – It takes the lower dimensionality output of Code and then decodes it to the uncompressed representation form of the input data.

Here choosing the number of nodes in Hidden Layer is important as if the nodes are more in a hidden layer, then the nodes will memorize the input data and the model might show overfitting. But if the number of nodes is very less then the model might lose some information as it will be hard to capture all the inputs.[12] There are seven types of autoencoders: Denoising autoencoder, Sparse Autoencoder, Deep Autoencoder, Contractive Autoencoder, Undercomplete Autoencoder. Convolutional Autoencoder, and Variational Autoencoder.

4.3.1 Denoising Autoencoder

Denoising Autoencoder helps to give uncorrupted data when the input data has some noise in it. The input data which has some noise in it is fed to an autoencoder which will train the corrupted input to provide output data without noise.[26] The input can be an image or text. The image provided as input has noise in it so the encoder apart from reducing dimensionality tries to remove noise as much as possible so that we can have the output image clear. This autoencoder tries hard to provide original output without the noise which was created by the network.

4.3.2 Sparse Autoencoder

Usually, if the number of neurons is less in the hidden layer, the model might underfit and if the number of neurons is more, the model might get overfit as well as increase the time it takes to train the model. In Sparse Autoencoder, the number of nodes in the hidden layer is more than the input layer, still, it can discover important features in the data. The level of activation is represented by the obscurity of a node in a generic sparse autoencoder. A constraint on sparsity is applied to the hidden layer[26]. This will prevent the model from overfitting.

4.3.3 Deep Autoencoder

There are two networks, one is encoding, and the other is decoding. Usually, there are many layers of encoding and decoding in each deep model. For this model, we apply unsupervised layer-by-layer pre-training [26]. But as there are more parameters than input, there are high chances of the model getting overfit.

4.3.4 Contractive Autoencoder

The main objective of the contractive autoencoder is to adopt a representation that is less sensitive to variation in data. It works on the concept of backpropagation. This is another regularization technique. However, this regularize corresponds to the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input.[26] It is usually calculated on input basis and is sum of square of all elements. [26]

4.3.5 Undercomplete Autoencoder

In Undercomplete Autoencoder, the number of neurons in the hidden layer is less than the input layer. It helps to find the most important feature in the data. Also, it helps to minimize the loss of input data. They do not copy the input data to output directly but rather maximize the probability of data as they don't need any regularization.[26]

4.3.6 Convolutional Autoencoder

Convolutional Autoencoder uses the convolution operation which sums all other signals in one signal. Set of signals are encoded first and then it tries to reconstruct the input from them modify the geometry or the reflectance of the image. They are state-of-art tools for unsupervised learning of convolutional filters. [26] Once the model is trained enough, it is given some random input sets to extract the features. Then we can use these features for classification purposes. This is very useful to remove noise and reconstruct the picture but the reconstruction is usually blurred or in lower dimension as some information is lost in the process.

4.3.7 Variational Autoencoder

Variational Autoencoder makes some assumptions for the distribution of latent variables. [26] To do this they use a variational approach, which results in a specific estimator Stochastic Gradient Variational Bayes estimator, and an additional loss component. [26] The advantage of this autoencoder is we can control how to model our latent distribution. But the drawback is we need to calculate the parameter relationship in the network due to backpropagation every time we train the model.

4.4 Principal Component Analysis (PCA)

PCA is the most popular unsupervised dimensionality reduction technique, which reduces higher dimensionality data to lower dimensional data. Principal Components Analysis (PCA) is an algorithm to transform the columns of a dataset into a new set of features called Principal Component.[25] Principal components are useful to make visualization of large information in fewer columns. It basically shows the easy separation of classes or clusters. It is not a feature selection technique. Rather, it is a feature combination technique. 25]

4.4 Why not PCA?

Principle Component Analysis (PCA) is a linear transformation of data while Autoencoder can be linear or nonlinear depending on the type of activation function.[16] So, the major reason for using Autoencoder over PCA is its incapability to deal with non-linear feature representation. If our dataset will have more non-linear data then our PCA would not be helpful for our data in such cases using an autoencoder will provide more appropriate outputs.

4.5 LSTM Autoencoder

In this project, the data we are dealing with is in sequence. When we work with a sequence there are many problems such as we cannot use neural networks as they deal with only fixed input length. Recurrent neural networks, such as the Long Short-Term Memory, or LSTM, network is specifically designed to support sequences of input data.[13] Usually, in a simple neural network, data is processed in a feed-forward manner. But Recurrent Neural networks use sequential data, as input and output both are dependent on each other. That is also a major reason for using the RNN model here. Here, LSTM autoencoder uses the Encoder-Decoder architecture for the sequential data.

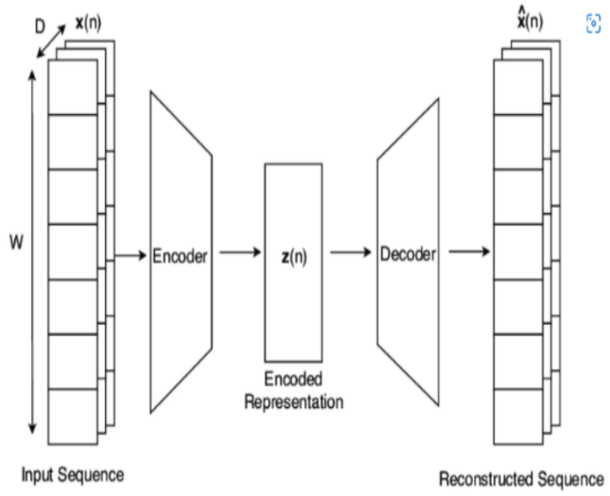


Fig:3 LSTM Autoencoder

V. METHODOLOGY

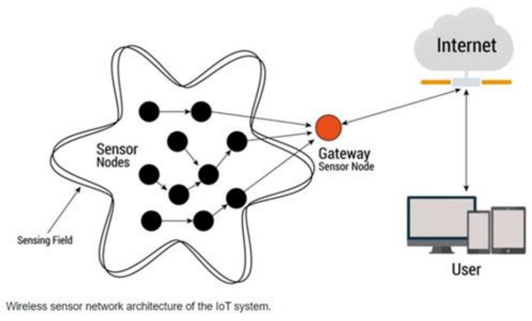


Fig : 4 Current Model

In this project, we are dealing with numerous sensors for collecting data from all over the world. This will increase the consumption of energy as a huge amount of data is transferred to the cloud. To solve this problem, we can use the data compression technique at the base station before transferring data to the cloud. Data compression will not only help in energy consumption but also can reduce the usage of storage in the cloud rather than uncompressed data. To collect IoT Data in a wireless sensor network we will use some deep learning techniques such as RNN-based Autoencoder which will reduce the dimensions of data. Autoencoder also helps by extracting necessary features from raw sensor data which can improve the speed and reduce storage usage.

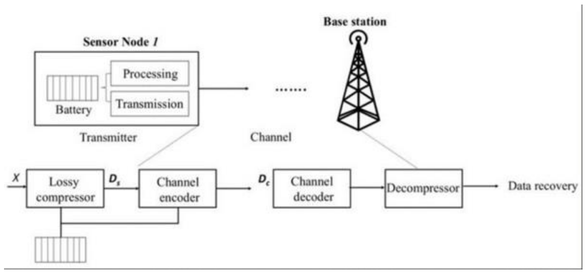


Fig : 5 The adapted Methodology

This is the concept we are using now, the data collected from sensors is compressed using an RNN-based Autoencoder before transferring it to the cloud. We are using Long Short-Term Memory (LSTM) Autoencoder here because it deals with sequential data and data collected from sensors is in sequence. Then this data is transferred to the cloud and when needed we can decode or decompress the data using a decoder to use it again.

VI. MODEL DESCRIPTION

In this section, we will describe the data and algorithm implemented in this project and the methods used for implementing the algorithm.

6.1 Datasets

We use the pump sensor dataset for training the model. The pump sensor dataset (2019) contains time-series data from 52 pump sensors from April 2018 to August 2018 for every minute with the label which shows the status of pump (Normal, Broken, Recovering). The dataset has split twice for train, test, and validation datasets with the training set containing 139160 data, the test set containing 43488 data, and the rest in validation dataset.

6.2 Data preprocessing

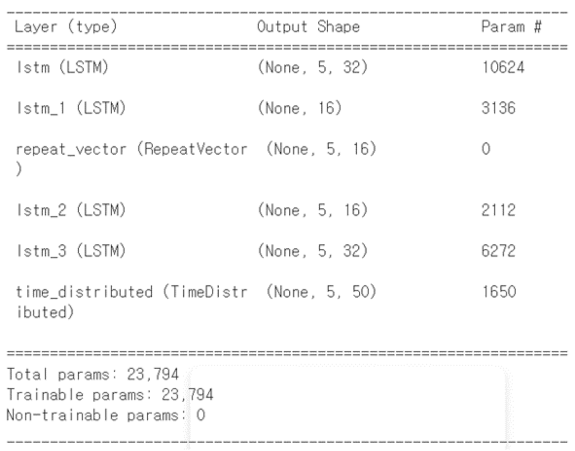
As we used the raw data which contains null values, we went through the data preprocessing process to accurately evaluate the algorithm. Since our goal is to determine whether the reconstruction of data is successfully made after data compression using LSTM autoencoder, we decide to evaluate its accuracy by determining the presence of pump failure with predicting the sensor value at the next minute. In order to do so, we divide the data value of the sensor by time and the label value representing the pump state and set as x value and y value. Also, we transformed the label which is string as a binary value to train the algorithm (Normal : 1, Broken, Recovering : 0).

Although the current data is 2D dimension, the LSTM model has a shape of 3D dimensions corresponding to (samples, timesteps, and features). Thus, we convert the data into a sequence form into 3D by adding timesteps. We set the timesteps for 5 minutes. In the process of LSTM autoencoder learning, we separate the normal (1) and abnormal (0) data because we will only train the model with normal data.

We applied StandardScaler() of scikit-learn which is z-score normalization to standardize different data characteristics. In order to apply the function, it had to be in 2D form, so the scaler is applied after going through the flatten process, and then converted into the 3D form again for LSTM model.

6.3 Define Model

We now explain how we implemented the model of LSTM autoencoder. LSTM autoencoder is constructed in the form of a stacked autoencoder with a symmetrical structure. We used Python and Keras for implementing LSTM autoencoder algorithm in this project. Keras is an API for developing artificial neural networks that runs on top of the Tensorflow engine.[20] [21]



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 5, 32)             10624

 lstm_1 (LSTM)               (None, 16)                3136

 repeat_vector (RepeatVector  (None, 5, 16)            0
 )

 lstm_2 (LSTM)               (None, 5, 16)             2112

 lstm_3 (LSTM)               (None, 5, 32)             6272

 time_distributed (TimeDistr  (None, 5, 50)            1650
 ibuted)

=================================================================
Total params: 23,794
Trainable params: 23,794
Non-trainable params: 0
_________________________________________________________________
```

Fig.6 The architecture of the LSTM implemented on in this project

First, two layers (lstm, lstm_1) are encoder and the following 3 layers after repeat vector (lstm_2, lstm_3, time_distributed) are decoder parts. Layer 1, LSTM (lstm) reads the input data and outputs 32 features with 5 timesteps. After that, layer 2, LSTM (lstm_1) takes the 5*32 input from the previous layer and reduces the feature size to 16. This part is where data compression occurs. We reduced 32 features to 16 features in this part so it outputs a feature vector of size 1*16 which dimension is reduced from 32 to 16. The output of this layer is the encoded feature vector of the input data which is the result of data compression. As the dimension of input data is compressed, it can reduce the usage of storage in the cloud rather than uncompressed data. The repeat vector layer performs as a bridge between the encoder and decoder modules by replicating the feature vector 5 times. It prepares the array input for the first LSTM layer in decoder.

Since we reduced the amount of data, we have to reconstruct them to prove the theory that even if we compressed the data, the function would run properly. This reconstruction process is performed in the decoder part. The decoder layer is implemented in the reverse order of the encoder parts. Layer 4, LSTM (lstm_2), and layer 5, LSTM (lstm_3) are the mirror images of layer2 and layer 1. The output of layer 4 is 5*16 and the output of layer 5 is 5*32 which is exactly the same size as the output of layer 1. After this, the time distributed layer creates a vector of length equal to the number of features from the previous layer and it duplicates it n_feature times. In this model, it makes 32*50 output.

In our implementation, we set the epochs as 200 so when an entire dataset is passed forward and backward through the network for 200 times. And we divide dataset into 128 batches as we set batch size to 128. Also, we set learning rate which is the step size at each iteration as 0.001. We implied Adam optimizer which is an algorithm for first-order gradient-based optimization of stochastic object functions, based on adaptive estimates of lower-order moments. This is because it is computationally efficient, has little memory requirements, and is well suited for problems that are large in terms of data. [22]

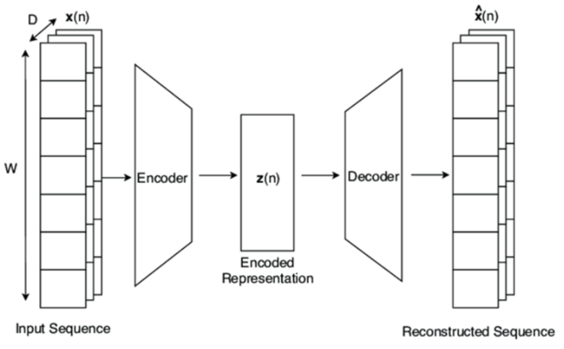
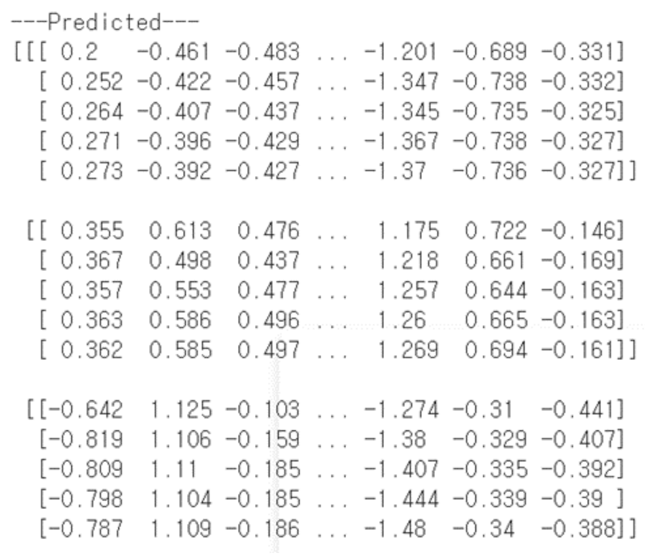


Fig.7 The architecture of the LSTM autoencoder[23]

VII. RESULT

In this section, we describe the results of measuring the performance of using the LSTM autoencoder. It is important to measure the performance of LSTM autoencoder prediction since it is the method to evaluate the reconstruction after data compression is successfully done. If successful prediction is possible with reconstructed data after data compression, we can solve the high energy consumption due to the transmission of huge amount of data to the cloud which is the major challenge in IoT.

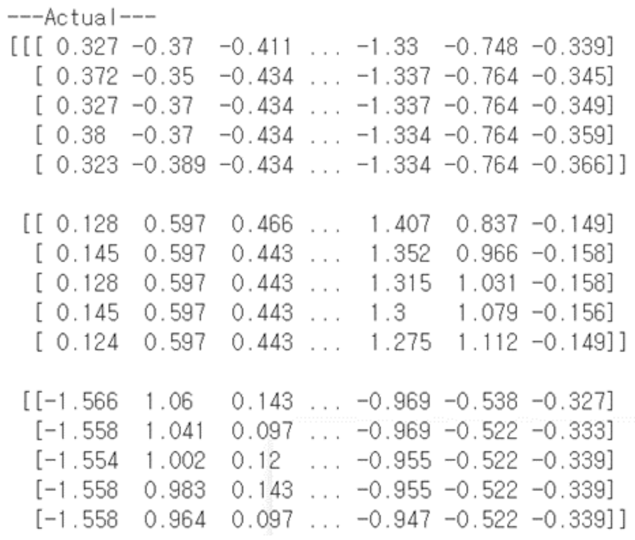


```
---Predicted---
[[[ 0.2   -0.461 -0.483 ... -1.201 -0.689 -0.331]
  [ 0.252 -0.422 -0.457 ... -1.347 -0.738 -0.332]
  [ 0.264 -0.407 -0.437 ... -1.345 -0.735 -0.325]
  [ 0.271 -0.396 -0.429 ... -1.367 -0.738 -0.327]
  [ 0.273 -0.392 -0.427 ... -1.37  -0.736 -0.327]]

 [[ 0.355  0.613  0.476 ...  1.175  0.722 -0.146]
  [ 0.367  0.498  0.437 ...  1.218  0.661 -0.169]
  [ 0.357  0.553  0.477 ...  1.257  0.644 -0.163]
  [ 0.363  0.586  0.496 ...  1.26   0.665 -0.163]
  [ 0.362  0.585  0.497 ...  1.269  0.694 -0.161]]

 [[-0.642  1.125 -0.103 ... -1.274 -0.31  -0.441]
  [-0.819  1.106 -0.159 ... -1.38  -0.329 -0.407]
  [-0.809  1.11  -0.185 ... -1.407 -0.335 -0.392]
  [-0.798  1.104 -0.185 ... -1.444 -0.339 -0.39 ]
  [-0.787  1.109 -0.186 ... -1.48  -0.34  -0.388]]
```

Fig.8 Predicted value by LSTM Autoencoder



```
---Actual---
[[[ 0.327 -0.37  -0.411 ... -1.33  -0.748 -0.339]
  [ 0.372 -0.35  -0.434 ... -1.337 -0.764 -0.345]
  [ 0.327 -0.37  -0.434 ... -1.337 -0.764 -0.349]
  [ 0.38  -0.37  -0.434 ... -1.334 -0.764 -0.359]
  [ 0.323 -0.389 -0.434 ... -1.334 -0.764 -0.366]]

 [[ 0.128  0.597  0.466 ...  1.407  0.837 -0.149]
  [ 0.145  0.597  0.443 ...  1.352  0.966 -0.158]
  [ 0.128  0.597  0.443 ...  1.315  1.031 -0.158]
  [ 0.145  0.597  0.443 ...  1.3    1.079 -0.156]
  [ 0.124  0.597  0.443 ...  1.275  1.112 -0.149]]

 [[-1.566  1.06   0.143 ... -0.969 -0.538 -0.327]
  [-1.558  1.041  0.097 ... -0.969 -0.522 -0.333]
  [-1.554  1.002  0.12  ... -0.955 -0.522 -0.339]
  [-1.558  0.983  0.143 ... -0.955 -0.522 -0.339]
  [-1.558  0.964  0.097 ... -0.947 -0.522 -0.339]]
```

Fig.9 Real value

We trained the model with train dataset of normal status and validate the model with a validation dataset which we split in the stage of data preprocessing. We printed the predicted value as an array to compare with actual data. Although there are some differences, it can be seen that successful predictions have been made since it predicts similar values with actual data as we can see in the figure above.

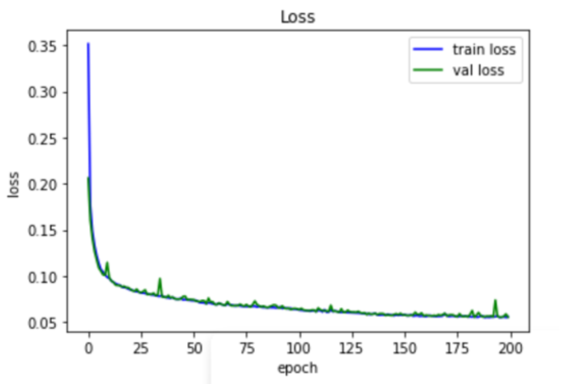


Fig.10 Loss graph of LSTM autoencoder

In this project, Mean Squared Error (MSE) has been used to evaluate the LSTM autoencoder algorithm in predictions in numerical value. The MSE is sum of squares for all prediction error values in each period and divided by the number of prediction periods.[24] We used MSE because it is intuitive and simple indicator to measure the performance of deep learning algorithm. Since MSE is calculated from the difference between the actual value and the predicted value, the smaller the value, the better the performance. As we can see in the plot above, the loss value of both train loss and validation loss converges to zero as the epoch progresses. This means that it can be confirmed that LSTM autoencoder made a successful prediction which means that the decoder part of LSTM autoencoder reconstructed the data well.

VIII. CONCLUSION

In this project, we showed that LSTM autoencoder can compress the sensor data by reducing the data dimension in encoder part. We also showed that reconstruction is successfully done in decoder part by evaluating the prediction function. As we showed the performance of LSTM autoencoder, we can successfully compress and reconstruct the data by using it. This means that we can compress the amount of data from IoT by implementing LSTM autoencoder. Since the dataset we used and most data from real world are time sequence data, we used LSTM autoencoder, however it is also possible to use another sort of autoencoder to reduce the amount of data if it is not sequence data.

By reducing the amount of sensor data connected to IoT which has to be transferred to cloud, we can solve energy consumption problem due to the transmission of huge amount of data. Also, it can reduce the usage of storage for data which leads to cost problem. In conclusion, by implementing LSTM autoencoder with IoT sensor dataset, we can solve major challenges in IoT, which is high energy consumption in data transmission and high cost due to the usage of data storage.

We can also improve the performance of IoT by implementing deep learning algorithm with data by changing the features used in the learning process of algorithm according to the purpose of using sensor data. By extraction only the necessary features that fit the purpose from the IoT sensor data which is complex and large-scale, performance improvement can be achieved while reducing the unnecessary waste of memory and energy in data transmission.

REFERENCES

[1] Laércio Pioli, Carina F. Dorneles, Douglas D. J. de Macedo, Mario A. R. Dantas (2022). “An overview of data reduction solutions at the edge of IoT systems: a systematic mapping of the literature”

[2] Joseph Azara, Abdallah Makhoula, Mahmoud Barhamgib, Rapha¨el Couturier (2019). "An energy efficient IoT data compression approach foredge machine learning"

[3] Oracle. What is IOT. https://www.oracle.com/ca-en/internet-of-things/what-is-iot/

[4] Spiceworks, IOT data https://community.spiceworks.com/topic/2454365-iot-data-how-to-collect-process-and-analyze-them

[5] Snowflake. What-IOT and how does it work. https://www.snowflake.com/guides/what-iot

[6] Barracuda-Data compression, Lossy vs Lossless Data compression. https://www.barracuda.com/glossary/data-compression

[7] Tech Computer Science- Data Compression. https://teachcomputerscience.com/data-compression/

[8] Tutorials Point- Difference between Lossy Compression and Lossless Compression https://www.tutorialspoint.com/difference-between-lossy-compression-and-lossless-compression

[9] Fig:1 Bournetcode-Data compression [accessed Jul 18, 2022] https://bournetocode.com/projects/GCSE_Computing_Fundamentals/pages/3-3-8-compression.html

[10“Introduction to Dimensionality Reduction - GeeksforGeeks,” GeeksforGeeks, Jun. 2017. https://www.geeksforgeeks.org/dimensionality-reduction/

[11] Fig:2 “Autoencoder,” Wikipedia, Aug. 03, 2022. https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_schema.png (accessed Aug. 08, 2022).

[12] A. Roy, “Introduction To Autoencoders - Towards Data Science,” Medium, Dec. 12, 2020. https://towardsdatascience.com/introduction-to-autoencoders-7a47cf4ef14b#:%7E:text=%20An%20Autoencoder%20has%20the%20follo%20Introduction%20To%20Autoencoders.%20A%20Brief%20Overview%20 (accessed Aug. 08, 2022).

[13] https://www.facebook.com/jason.brownlee.39, “A Gentle Introduction to LSTM Autoencoders,” Machine Learning Mastery, Jun. 13, 2019. https://machinelearningmastery.com/lstm-autoencoders/

[14] Fig:3 Detecting Mobile Traffic Anomalies Through Physical Control Channel Fingerprinting: A Deep Semi-Supervised Approach - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/LSTM-Autoencoder-for-Anomaly-Detection_fig2_336594630 [accessed 16 Jul, 2022]

[15] R. Dolphin, “LSTM Networks | A Detailed Explanation,” Medium, Mar. 26, 2021. https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

[16] A. Mungoli, “Dimensionality Reduction: PCA versus Autoencoders,” Medium, Aug. 01, 2020. https://towardsdatascience.com/dimensionality-reduction-pca-versus-autoencoders-338fcaf3297d (accessed Aug. 08, 2022).

[17] “What Is Deep Learning? | How It Works, Techniques & Applications,” Mathworks.com, 2019. https://www.mathworks.com/discovery/deep-learning.html

[18] Fig :4 An Intelligent Failure Detection on a Wireless Sensor Network for Indoor Climate Conditions - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Wireless-sensor-network-architecture-of-the-IoT-system_fig6_331199690 [accessed 18 Jul, 2022]

[19] Fig :5 Analysis of lossy compression and channel coding tradeoff for energy efficient transmission in low power communication systems - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Block-diagram-of-sensor-nodes-sensors-composed-of-a-source-encoder-channel-encoder_fig1_332989863 [accessed 18 Jul, 2022]

[20] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in Proc. OSDI, vol. 16. Savannah, GA, USA, 2016, pp. 265–283.

[21] F. Chollet et al.. (2015). Keras: Deep Learning Library for Theano and Tensorflow. [Online]. Available: https://keras.io/k

[22] Diederik P. Kingma, Jimmy Ba, “Adam : A Method for Stochastic Optimization”, San Diego, 2015

[23] Hoang D. Trinh, Engin Zeydan, Lorenza Giupponi, Paolo Dini, “Detecting Mobile Traffic Anomalies through Physical Control Channel Fingerprinting : a Deep Semi-supervised Approach”, IEEE, 2019

[24] J. Fürnkranz et al., “Mean Squared Error,” in Encyclopedia of Machine Learning, 2010

[25] “Principal Component Analysis (PCA) - Better Explained,” ML+, Mar. 23, 2019. https://www.machinelearningplus.com/machine-learning/principal-components-analysis-pca-better-explained/

[26] “Different types of Autoencoders,” OpenGenus IQ: Computing Expertise & Legacy, Jul. 14, 2019. https://iq.opengenus.org/types-of-autoencoder/#:~:text=Different%20types%20of%20Autoencoders.%201%20Denoising%20autoencoder.%202 (accessed Aug. 08, 2022).