Optimizing data transfer between IoT sensors and server using Neural Networks

Report submitted in fulfillment of the requirements
for the B.Tech Project of

Fourth Year B. Tech.
in
Computer Science and Engineering

Submitted by

Roll No	Names of Students
18074013	Nishant Mittal
18074011	Mudit Bhardwaj

Under the guidance of
Dr. Hariprabhat Gupta

Department of Computer Science and Engineering
Indian Institute of Technology (BHU) Varanasi
Varanasi, Uttar Pradesh, India – 221005

Semester VII

Dedicated to

Our beloved parents, teachers,

Our laptops and Varanasi

Declaration

We certify that

1. The work contained in this report is original and has been done by ourself and the general supervision of our supervisor.
2. The work has not been submitted for any project.
3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.
4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: November 21, 2021

Nishant Mittal (18074013 - IDD)
Mudit Bhardwaj (18074011 - IDD)
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Department of Computer Science and Engineering
Indian Institute of Technology (BHU) Varanasi
Varanasi, INDIA 221005.

Certificate

This is to certify that the work contained in this report entitled **"Optimizing data transfer between IoT sensors and server using Neural Networks"** *being submitted by* **Nishant Mittal (18074013)** *and* **Mudit Bhardwaj (18074011)**, *carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of my supervision.*

Place: IIT (BHU) Varanasi
Date: November 21, 2021

Dr. Hariprabhat Gupta
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

It is a great pleasure for us to express respect and deep sense of gratitude to our supervisor Dr. Hariprabhat Gupta, Assistant Prof., Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, for his wisdom, vision, expertise, guidance, enthusiastic involvement and persistent encouragement during the planning and development of this work. We are indebted and grateful to the Almighty for helping us in this endeavor.

Place: IIT (BHU) Varanasi
Date: November 21, 2021

Nishant Mittal
Mudit Bhardwaj

Contents

List of Figures

1 Introduction

1.1 Abstract

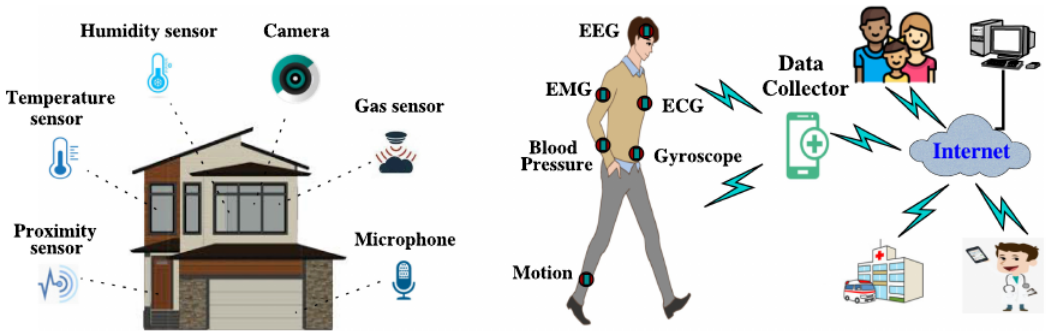
The objective is to design a compression-decompression model using convolutional deep neural networks to optimize the data transmission of time series across IoT sensors and a server and show how it can improve the efficiency of the model.

We further tried to improve our model by breaking it into two arts with different complexity considering that the server has more computational power that can easily run complex models.

1.2 Definitions

1.2.1 IoT Sensor

In an application of Internet of Things(IoT), a system generally consists of various sensors which continuously generate a Multivariate Time Series with different sampling rate. In our project we have assumed that our IoT application receives data from n sensors, which will be constituting our MTS.



(a) Home monitoring (b) Patient monitoring

Figure 1: Illustration of home and patient monitoring using MTS generated by various sensors.

1.2.2 Time Series

A time series is a sequence of data points that occur in consecutive order over a period of time. In other words, it is a sequence taken at successive equally spaced data points in time. Simply, it's a sequence of discrete-time data. Here in this project, we also assume that all the measurements by the sensors are taken at regular time intervals.

1.2.3 Convolutional Neural Networks

A Convolutional Neural Network can be defined as a Deep Learning algorithm which is generally takes an image as a input and then it assigns importance based on various specific features and aspects present in the image and then it is able to differentiate one image from others.In CNN, pre processing required is generally quite low as compared to other classification algorithms present. While in primitive methods filters are hand-engineered, with enough training, CNNs have the ability to learn these characteristics.

1.2.4 Loss

Loss in machine learning can be defined as a metric to judge the performance of the machine learning or deep learning model. To be specific, Loss is the penalty for a bad prediction. That is, loss is a metric indicating how bad the model's prediction was on a single row of dataset. If the model's prediction is accurate, the loss should ideally be zero, otherwise, the loss is greater. There are many types of loss functions we can use, of which we talk about later in this section.

1.2.5 Loss Minimization

As described in the section above our main goal is to set the values of coefficients such that the loss function is minimum. Loss function is nothing but an indicator of how close are the predicted values in comparison to actual values. Some of the most common ways to minimize the loss function are Gradient Descent, Stochastic Gradient Descent, Stochastic Dual Co-ordinate Ascend, etc.

1.2.6 Regularization

While minimizing the loss function, the most important thing we have to take care of is over-fitting. To deal with this we use a technique known as Regularization. We do this by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values. Some of the most commonly used regularization terms are as follows.

1.2.7 Loss Function

The basic function of loss function is that it maps decision taken by our model to their associated cost. Basically in supervised machine learning algorithms, our main goal is to minimize the loss function for each training example during the learning process.

Loss function Vs Cost function Loss function is calculated for every training data and it is also known as an error function whereas cost function is the average loss calculated over the entire training dataset.The optimization strategies aim at minimizing the cost function.

Types of Loss Function: Some of the widely used loss functions are:

1. **Hinge Loss Function:** This type of loss function is mostly used in SVM model.

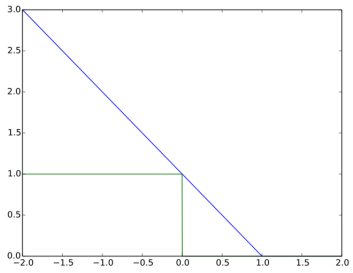


Figure 2: Graphical representation of hinge loss function.

In this graph the x-axis represents the distance from the boundary of any single instance, and the y-axis represents the loss size, or penalty, that the function will incur depending on its distance.

2. **Mean Squared Loss Function:** Mean Squared Error is also known as L2 loss and it is defined as the square of the difference between the predicted and the true values.

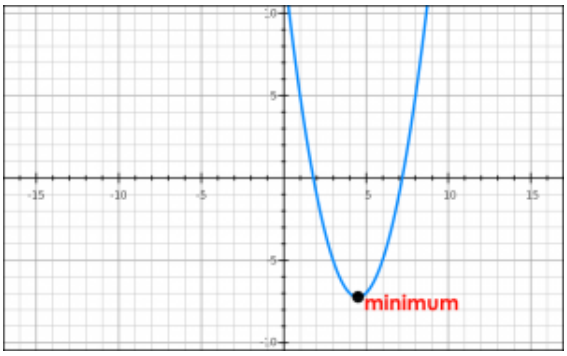


Figure 3: Graphical representation of mean squared loss function.

In Mean Squared Error,loss function penalizes the model for making large errors by squaring them. Squaring a large quantity makes it even larger.

2 Main Approach

2.1 Overview

Our main motive is to design a convolutional deep neural networks to optimize the data transmission of time series across IoT sensors and a server and show how it can improve the efficiency of the model.

First, we developed and trained a deep neural network such that First layer of convolutional neural networks will take multivariate time series with N sensors as input, Middle layer of our model will consists of N/4 neurons which will represent the compressed data points of the time series, and Final layer of our model will again consists of N neurons which will represent the output i.e. final time series. For finalising this model, we set loss functions, neural structure in rest of the layers etc so that there was minimal difference between the original time series and the final time series.

After that, we extracted the weights and biases from our previously trained best performing model and then initialized our new model layers with the extracted values of weights and biases. We further tried to improve our model by breaking it into two parts with different complexity considering that the server has more computational power and we can afford to make a little trade off with loss to make the model on server side more complex than one on the sensor side to minimize the time required by the time series data points to reach the server.

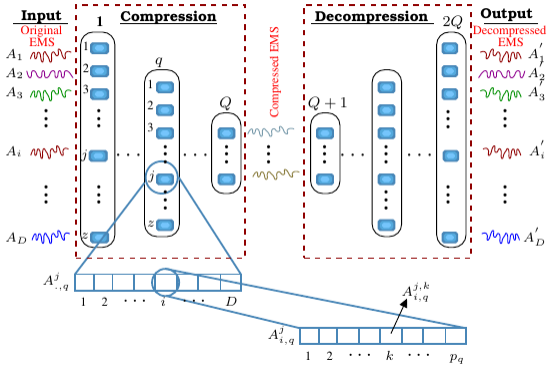


Figure 4: Compression Decompression Model.

2.2 Building the main model

In the very first step, we tried to develop and train a deep neural network which had the following characteristics.

1. First layer of DNN will take multivariate time series with N sensors as input.
2. Middle layer of our model will consists of N/4 neurons which will represent the compressed data points of MTS.
3. Final layer of our model will again consists of N neurons which will represent the output i.e. final MTS.

For finalising this step, we set loss functions, neural structure in rest of the layers etc so that there was minimal difference between the original time series and the final time series.

2.3 Breaking the model into compression decompression models

In this step we broke our initial model into two parts i.e. Compression and Decompression models.

In order to achieve this, we extracted the weights and biases from our previously trained best performing model and then initialized our new model layers with the extracted values of weights and biases.

Main reason to do so is that now our model can be run separately on the server and the sensors. The compression model will run on the sensor-side which will minimize the amount of data to be transferred. And the decompression model can be run on the server-side to know the actual value of the data points sent by the sensors.

2.4 Further optimizing the approach

We further tried to improve our model by breaking it into two parts with different complexity considering that the server has more computational power and we can afford to make a little trade off with loss to make the model on server side more

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 9)                 90
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 80
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 54
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 28
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 15
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 16
_________________________________________________________________
dense_6 (Dense)              (None, 6)                 30
_________________________________________________________________
dense_7 (Dense)              (None, 7)                 49
_________________________________________________________________
dense_8 (Dense)              (None, 8)                 64
_________________________________________________________________
dense_9 (Dense)              (None, 9)                 81
_________________________________________________________________
dense_10 (Dense)             (None, 10)                100
_________________________________________________________________
dense_11 (Dense)             (None, 12)                132
_________________________________________________________________
dense_12 (Dense)             (None, 9)                 117
=================================================================
Total params: 856
Trainable params: 856
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Overview of the convolutional neural network model.

complex than one on the sensor side to minimize the time required by the time series data points to reach the server.

2.5 Implementation

We implemented the above steps using convolutional neural networks. We use python for mainly all of the work including, data pre-processing, building and training models. Following is an screenshot of the pseudo-implementation of the compression model after training and extracting the weights.

We used a dataset, that involved sensors attached to a marker (gyroscopre, accelerometer, etc.) to record the readings while 20 different people, drew different digits on a white-board.

The source code of our project is available on: https://github.com/nishantwrp/time-

```
NN_encoder = Sequential()

# Input Model
NN_encoder.add(Dense(9, input_dim = 9, activation='linear'))

# Hidden Layers
NN_encoder.add(Dense(8, activation='linear'))
NN_encoder.add(Dense(6, activation='linear'))
NN_encoder.add(Dense(4, activation='linear'))

# Output Layer
NN_encoder.add(Dense(3, activation='linear'))

NN_encoder.compile(loss='mae', optimizer='adam', metrics=['mae'])
# The weights extracted from an already trained model.
NN_encoder.load_weights("encoder_weights.hdf5")
NN_encoder.compile(loss='mae', optimizer='adam', metrics=['mae'])

INPUT_CSV = os.path.join(os.getcwd(), "test_input.csv")
OUTPUT_CSV = os.path.join(os.getcwd(), "test_output_mid_layer.csv")

input_data = read_csv(INPUT_CSV, header=None)
predictions = NN_encoder.predict(input_data)
predictions = [['{0:.5f}'.format(x.item()) for x in y] for y in predictions]
with open(OUTPUT_CSV, 'w', newline='') as file:
    csv_writer = csv.writer(file)
    for row in predictions:
        csv_writer.writerow(row)
```

Figure 6: Pseudo Code of Compression Model

series-compression-decompression

3 Results and Analysis

We were successfully able to design a compression-decompression model using convolutional deep neural networks to optimize the data transmission of time series across IoT sensors and a server and we also showed how it can improve the efficiency and runtime of the model.

We further showed how we can improve our model by breaking it into two parts with different complexity considering that the server has more computational power and we can afford to make a little trade off with loss to make the model on server side more complex than one on the sensor side to minimize the time required by the time series data points to reach the server.

On doing so we were able to achieve an average loss of 0.17 between the input time series and the final time series. Finally we can conclude that with the help of deep neural networks we can design such a model that can compress and decompress sensors data quite effectively and efficiently which will in turn help us by saving time during transmission of data and at the same time giving almost correct result.

```
Epoch 00047: val_loss improved from 0.18565 to 0.18138, saving model to Weights-047--0.18138.hdf5
Epoch 48/50
18000/18000 [==============================] - 33s 2ms/step - loss: 0.2167 - mae: 0.2167 - val_loss: 0.2089 - val_mae: 0.2089

Epoch 00048: val_loss did not improve from 0.18138
Epoch 49/50
18000/18000 [==============================] - 35s 2ms/step - loss: 0.2158 - mae: 0.2158 - val_loss: 0.1790 - val_mae: 0.1790

Epoch 00049: val_loss improved from 0.18138 to 0.17903, saving model to Weights-049--0.17903.hdf5
Epoch 50/50
18000/18000 [==============================] - 34s 2ms/step - loss: 0.2151 - mae: 0.2151 - val_loss: 0.1973 - val_mae: 0.1973

Epoch 00050: val_loss did not improve from 0.17903
<keras.callbacks.History at 0x7fb96befca50>
```

Figure 7: Results of the model

Bibliography

1. An Energy Efficient Smart Metering System using Edge Computing in LoRa Network

2. W. Zhang, Y. Wen, Y. J. Zhang, F. Liu, and R. Fan, "Mobile cloud computing with voltage scaling and data compression," in Proc. SPAWC, 2017, pp. 1–5.

3. F. Van den Abeele, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Scalability Analysis of Large-Scale LoRaWAN Networks in NS- 3," IEEE Internet of Things Journal, vol. 4, no. 6, pp. 2186–2198, 2017.

4. J. Kolter and T. Jaakkola, "Approximate inference in additive factorial HMMs with application to energy disaggregation," in Proc. AIS, 2012, pp. 1472–1482.

5. X. Xing, J. Song, L. Lin, M. Tian, and Z. Lei, "Development of Intelligent Information Monitoring System in Greenhouse Based on Wireless Sensor Network," in Proc. ICISCE, 2017, pp. 970–974.