上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

# 学士学位论文

THESIS OF BACHELOR


论文题目: 基于批量训练和机器学习的电机健康状况适应性测系统


| | | |
|---|---|---|
| 成一伟 | 学号: 515370910081 | 专业: 电子信息 |
| 张挺 | 学号: 515370910016 | 专业: 电子信息 |
| 周辰昕 | 学号: 515370910033 | 专业: 电子信息 |
| 朱俊茹 | 学号: 515370910193 | 专业: 电子信息 |

指导教师 马澄斌，副教授

学院（系） 密西根学院


学期 2018-2019 秋季

# 上海交通大学
# 毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：张挺 周成伟 朱俊荔

日期：2018 年 12 月 10 日

# 上海交通大学

# 毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密☑。

（请在以上方框内打“√”）

作者签名：张挺 邵帅 朱俊杰 [illegible]　　指导教师签名：[illegible]

日期：2018 年 12 月 10 日　　日期：2018 年 12 月 10 日

# 基于批量训练和机器学习的电机健康状况适应性监测系统

## 摘要

作为许多领域的重要装置，电机在工业生产、制造中起着重要的作用。有效的电机健康状况监测系统能够帮助及时监测故障甚至预防性监测出故障，从而提高生产效率。传统的数据驱动型电机健康状况监测系统通常使用云端训练(数据传送至云端再训练模型)。然而，云端训练的数据传输过程需要花费大量时间。同时在此过程中数据很有可能产生丢失。 此外，传统系统通常使用同“一次性”训练模型。在该模式下，模型一旦训练完成，将永远不再更新。这使得系统缺乏“适应性”。本监测系统，基于边缘端的批量训练，能够有效解决数据传输以及模型适应性问题。本系统的模型的训练过程被放在了边缘段(靠近电机)一侧。这减少了传统云计算传输数据所需的时间并降低了数据丢失的可能性。同时，批量训练支持训练数据分批次处理。用户可以随时根据电机的新状况(新一批次的数据)，对于原有模型进行更新，确保模型的适应性。本监测系统的用户界面清晰明了，可以显示模型的实时训练情况，电机健康值以及历史数据。此外，用户界面还提供模型“重新训练”,“停止训练”等功能，确保系统能运用于实际生产的各种情况。

**关键词：**故障检测，机器学习，批量训练，边缘计算，电机，用户界面设计

# Adaptive Motor Monitoring System Based on Machine Learning Batch Training

## ABSTRACT

Motor, as one of the most important equipment in various fields, plays an important role in today's manufacturing. Effective motor health monitoring system can help detect the anomality in time or even detect the anomality in advance so that the production efficiency can be improved. A traditional data-driven approach is based on cloud training(transferring the data to the cloud and training the model on the cloud). However, that process is time-consuming and data may lose during that process. Moreover, traditional systems always adapt an "one-time" model, which means that the model will never change once the train finishes.

The monitoring system here is based on edge batch training and can solve the problem well. In the system, the process of train of model is place on the edge side (the side of motor). That reduces the time of transferring data and reduce the probability of loss of data. What's more, as batch training supports training the model by batches of data, it guarantees the adaptivity of the model. That is because the users can update the model by a new batch of data based on the latest motor condition. By the way, the system provides a clear user interface where shows the real-time train process, the health score of the motor monitored and the historical data. Also, the user interface supports function such as "Retrain" the "Stoptrain" so the system can be applied in various real conditions.

**Key words:** Anomaly detection, Machine Learning, Batch Training, Edge Computing, User Interface Design

# Acknowledgements

We give our highest respect and gratitude for the instructor Chengbin Ma who gave us weekly instruction and guide, CyberInsight mentor Fang Chen who manages the project plan perfectly and offer the help for us passionately and selflessly, CyberInsight mentor Wentao Li who help answer the question of embedded system unselfishly and CyberInsight mentor Jian Chen who help answer the question of the web user interface passionately.

Fig. 1 Our Photo with the Final Design

# Table of contents

# List of figures

# List of tables

# Chapter 1

# Introduction

Nowadays, the concept of the smart industrial web has become popular in the present society and manufactory field. Anomaly Detection is one of the most important concepts among the smart industrial field, which means that the system needs to give out the real time alarming signal when it meets the condition that the information and data type of the industrial component becomes abnormal. Generally, a model based on machine learning will be used to trained on a set of data from the industrial component and be used in the further detection.

## 1.1 Problem Description

The specific industrial component in our project is the motor used in the industrial pipeline. Traditionally, the cloud one-time model is used as the detection model described in the Introduction part. In cloud one-time model, the data sets are fetched from the motor when off-line, and they are sent to the remote cloud later. This cloud one-time model is trained at the cloud based on these off-line data sets. During the future detection period, the data are sent from the local place to the cloud and be judged at cloud. Then the signal of whether the motor works abnormally will be shown to the customer to keep them informed of the health condition of the motor and could take the immediate action if the motor is broken or aged.

## 1.2 Drawbacks of traditional Cloud One-time Training

There are three major drawbacks listed as follows:

1. Firstly, for one-time training, the training begins only when all the data sets have arrived to the cloud. The whole series of data sets are inputted with the form of matrix

into the model training at one time, which means that the training need to wait for the whole data series to begin. Thus, it needs a lot of time for waiting data and then training a one-time model can be time-consuming.

2. Furthermore, since the local edge where the motor locates has little memory space for a huge amount of data, thus either the detection data sets or the training data sets need to be sent to the cloud for test or training. The processes of testing and training both need to be done on the cloud. However, the delivery of the data sets from the local edge to the cloud can be unsafe and some data may loss when delivering. What's more, the efficiency will also decrease.

3. Besides, the cloud one-time training is not adaptive for the various motors. If the motor condition is changed and the old motor no longer fits, we need to re-train the model. Then we need to gather the data from the local and sent them to the cloud for training again. Also we need to change the parameter of the model when training by ourselves, which is complex.

## 1.3 Possible Solution: Edge Batch Training

The Edge Batch Training is the combination of the edge computing and batch training. It can overcome the three drawbacks mentioned in the cloud one-time training part. The expected outcome is like:

1. The testing and training processes are both finished at the local edge and the model is just saved at the local edge.

2. The model training is batch training which means that the training process begins as soon as the first data set is gotten. Batch training could save a lot of time and save the memory space for storing a huge amount of data sets.

3. The only thing customer need to do is pressing the "retrain" button on the user interface when the motor is changed. Then the model will be re-trained by updating itself without people's help.

# Chapter 2

# Specifications

## 2.1 Customer Need

Our customer wants to have an adaptive motor monitoring system which is a optimization model for the previous Real-time Prognostic and Health Management(PHM) System. This system consists of a Linux embedded system which contains the batch training algorithms and a web user interface where the customers can monitor the real-time machine condition. To be more specific, when a new condition is formed after a new maintenance process or changes to a new machine, our customers can trigger a new process of online training on Raspberry Pi. And when the training is completed, it will inform the user via web UI and start to monitor the real-time machine condition. In order to achieve these requirements, we need to have:

1. **Efficiency to train the model**
   As is discussed before, the project aims to develop another new system to monitor the health condition of the motors. And the developed new method should fix the problem existing in the traditional method of its low efficiency. In the project, the most time-consuming is the process to train the model since once the train finishes, the model can be used in real-time. Thus, the efficiency to train the model is considered one of the customer need.

2. **Adaptivity to different kinds of motors**
   Also, as is indicated, our solution can solve the non-adaptivity problem existing in the old model. Thus, adaptivity to different kinds of motors is also one of the customer needs.

3. **High Accuracy**
   Although the first two points are the key of this project, the accuracy of the system is also of vital importance. It is a monitoring system so that it must give out the correct detection results. If the accuracy is not good, it may result in serious problems in future when it is adopted in real conditions. Thus, high accuracy is also one of the customer need.

4. **Effective remote control**
   As a necessary part of a system, it must be with the function of remote control, so that the user can make a serious of operations on the system. Otherwise, the system will keep working except that the user come to the place the system is set to make operations. That is unrealistic since in many cases, the monitoring system may be set in some remote areas.

5. **A Web user interface**
   To make the system more view able, the system should be equipped with a Web user interface. The user interface should be as simple and clear as possible so that even the inexperienced workers or workers without theoretical knowledge cna directly know whether the motor is healthy or not.

## 2.2 Engineering specification using QFD

In order to reflect the relationship between the customers' requirement and the engineering specification, we employ the method qualify function deployment (QFD). Qualify function deployment(QFD) is a method to transform qualitative user demands into quantitative parameters to deploy the functions forming quality and to deploy methods for achieving the design quality into subsystems and component parts, and ultimately to specific elements of the manufacturing process. Since our customer needs are identified, we can now use the QFD chart to prepare the product in Figure 2.1.

1. Customer requirements are stated on the left side of the matrix as shown below. These are organized by category based on the affinity diagrams. In our case, these needs are fancy appearance, high accuracy, short training time, remote control, adaptability to different kinds of motors and reasonable cost. Then for each requirement, we state the customer priorities using a 1 to 10 rating. Use ranking techniques and paired comparisons to develop priorities.

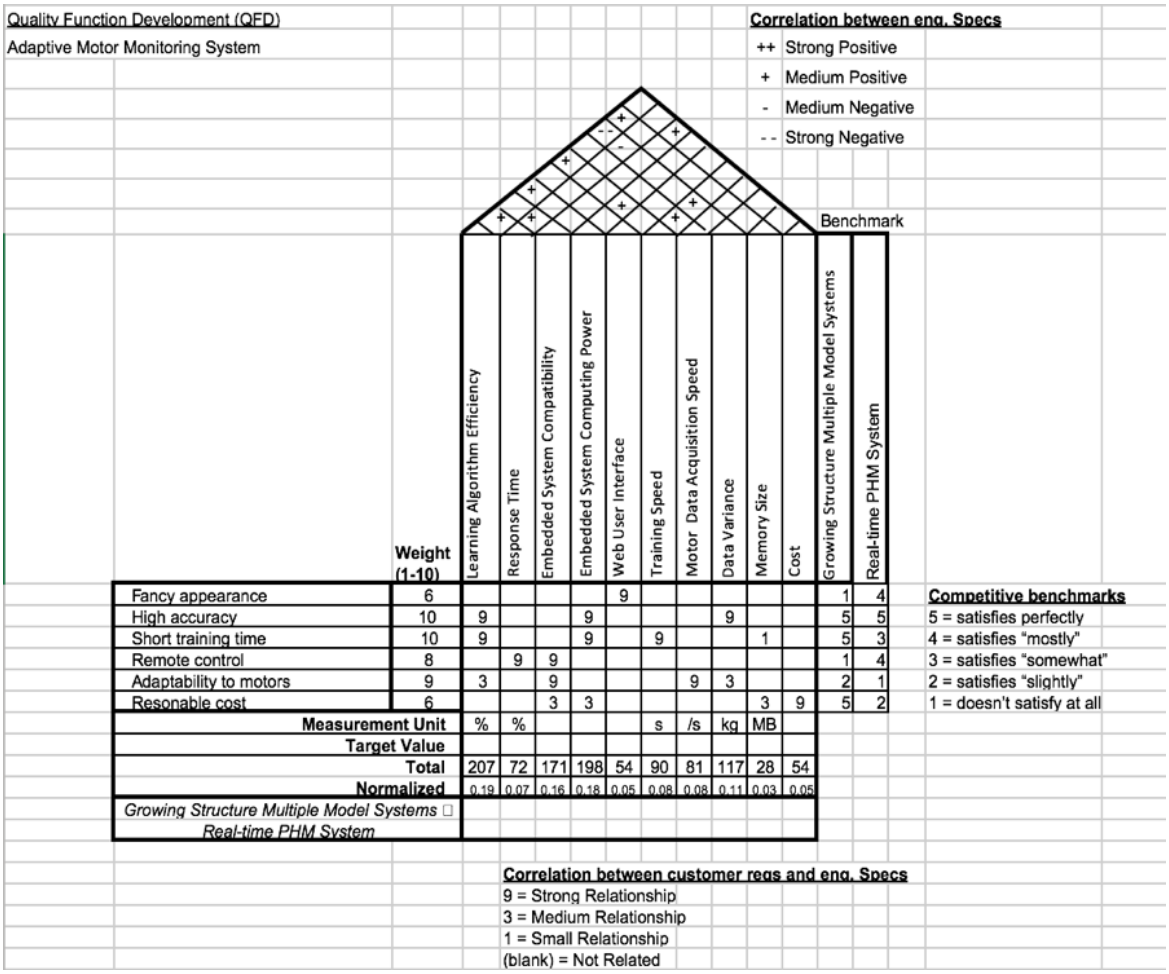


Fig. 2.1 QFD chart

2. Evaluate prior generation products against competitive products, which is shown in the benchmark. During our survey, there are two models having similar functions, which are Growing Structure Multiple Model System (GSMMS) and real-time PHM system. We rate these two models according to their performance. During our survey, we found that GSMMS is able to provide explicit estimations of system operation regions based on inputs fed to the system, which leads to the region dependent decision making scheme for anomaly detection and fault diagnosis. However, the model is only demonstrated in a simulated automotive electronic throttle system with the presence of substantial multiplicative noise [8]. And the Real-time Prognostic and Health Management(PHM) System is the system we are based on, which enables anomaly detection but cannot be batched detection. So, we give them their rates.

3. Develop relationships between customer requirements, and engineering specifications. Use symbols for strong, medium and weak relationships. And determine the potential positive and negative interactions between engineering specifications.

4. Calculate important ratings. Assign a weighting factor to relationship symbols. In our case, we regard the high accuracy of our model with strong relationship with learning algorithm efficiency, embedded system computing power, data variance, etc. For these items, we rate 9. And other items are rated using the same strategy.

5. Analyze the matrix and using the normalization to find the importance of different engineering specifications. In our case, we regard the learning algorithm efficiency as the most important.

Thus, our engineering specifications can be formed according to the analyzing steps above. It is given in Table 2.1.

Table 2.1 Customer Need vs. Engineering Specification

| **Engineering Specifications** | **Customer Need** |
|---|---|
| Efficiency to train the model | Training rate >50 datum/s<br>Response Time< 200ms with 0 loss |
| Adaptivity to different kinds of motor | The system can be trained to detect > 1 motors |
| High accuracy | Accuracy> 95% |
| Remote Control | Supported Operations: Start, Restart, Stop |
| A Web user interface<br><br>Know the health condition of the motor<br><br>Know the historical health condition of the motor<br><br>Know real-time train process | Show:<br><br>the real-time health score<br><br>the historical data<br><br>the train process |

# Chapter 3

# Concept Generation

According to the customer need, the project aims to develop

1. a real-time system monitoring motor health based on batch training edge training
2. a Web UI interface should be developed to make it more user friendly

Then, we will use methods to do the concept generation:

## 3.1 Brainstorming

A simplified figure of the concepts discussed is shown in Figure 3.1. As is seen, in that part,

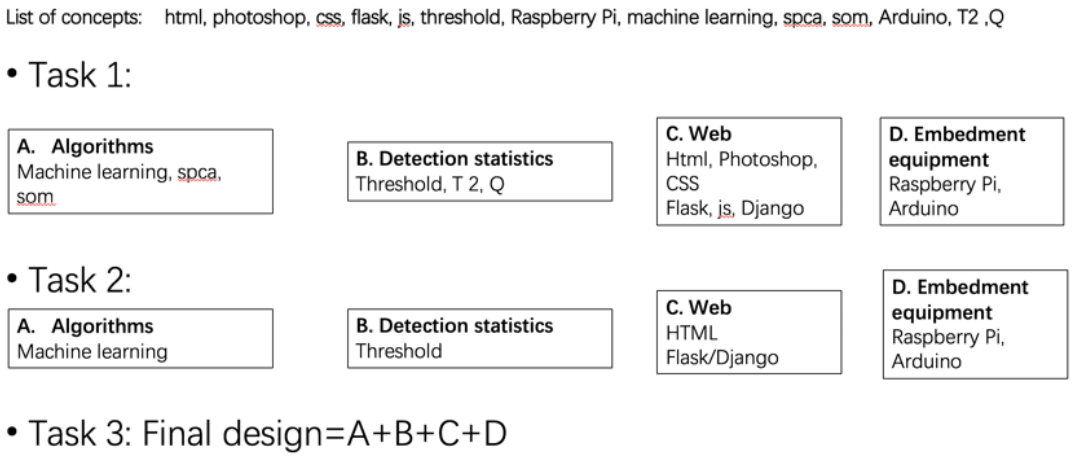


Fig. 3.1 Brain storming

the whole design is divided into several sub tasks: the train of model, the detection statistic, the Web UI design and embedment to the system. Those sub tasks will together complete our final task. However, problem still exists since the relationship between the sub tasks have not been figured out. As a result, it still remains unknown whether the division of the sub tasks are acceptable and the exact relationship between the subtasks. It will lead to problem in the whole project plan.

## 3.2 Morphological analysis

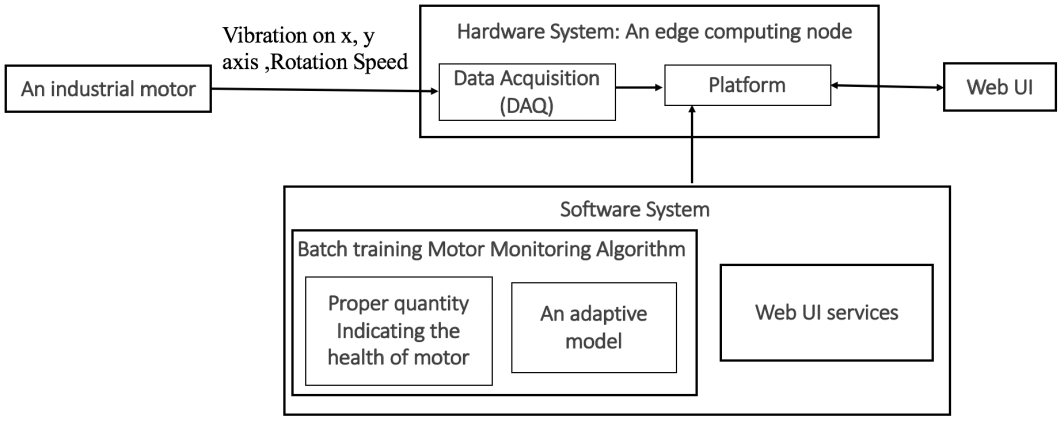


Fig. 3.2 Flow chart

According to the function structure in Figure 3.2, the final system keeps to collect datum from a working industrial motor. Then it utilizes the gained datums to train a model by the batch training algorithm. The model will store the characteristics of the motor when it works normally. Also, a detection statistic is provided in the algorithm. A value of that detection statistic will be calculated according to each datum and a threshold of the detection statistic is calculated based on the trained model. Thus, by comparing between the value of detection statistic for each datum and the value of threshold, the condition of the motor can be figured out in Table 3.1:

Table 3.1 Design Concepts

| Condition | Motor Condition |
|---|---|
| Value>=Threshold | Abnormal |
| Value<Threshold | Normal |

Thus, the goal of developing a real-time system monitoring motor health based on batch

training can be accomplished. Considering the edge training, it means that the training process (ie. the algorithm) runs on the edge side. To fulfill that need, edge devices should be used. What's more, further analysis shows that the Web services can should be run on the edge side. Thus, the Web service will also be embedded on the edge side such that the second goal of developing a Web UI interface is also fulfilled.
Before we generate the sub functions, a few facts should be acknowledged to clarify the analysis:

1. For the model and detection quantity, there exists combinations due to compatibility between models and quantities.

2. There are choices for Web services and embedment equipment. Since Web services are parallel to the algorithms and they can both be embedded by different equipment, the choice of the two aspects can be compatible to all the other choices.

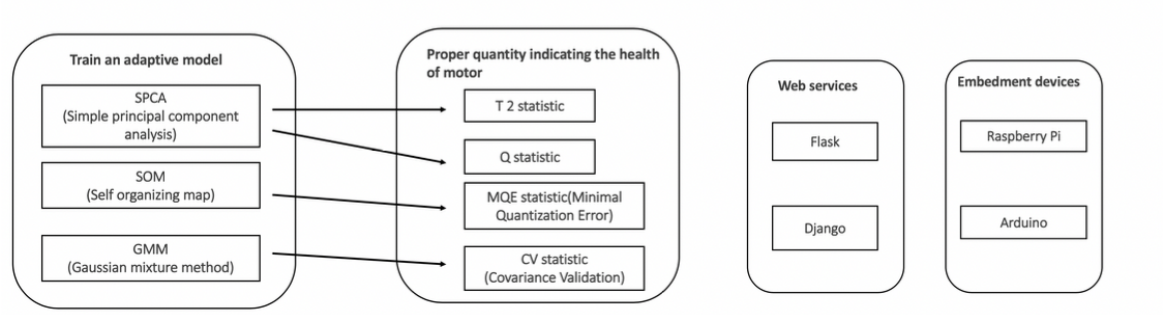


Fig. 3.3 Subfunction

That will thus form 24 different design concepts, which is given in Table 2.2:

Table 3.2 Subfunction

| No | Algorithm | Web UI | Platform |
|---|---|---|---|
| 1 | IPCA+ T2 | Flask | Raspberry Pi |
| 2 | IPCA+ T2 | Django | Raspberry Pi |
| 3 | IPCA+ T2 | Flask | Arduino |
| 4 | IPCA+ T2 | Django | Arduino |
| 5 | IPCA+ T 2 | Flask | PC |
| 6 | IPCA+ T 2 | Django | PC |
| 7 | SPCA+Q | Flask | Raspberry Pi |
| 8 | SPCA+ Q | Django | Raspberry Pi |
| 9 | SPCA+ Q | Flask | Arduino |
| 10 | SPCA+ Q | Django | Arduino |
| 11 | SPCA+ Q | Flask | PC |
| 12 | SPCA+ Q | Django | PC |
| 13 | SOM+MQE | Flask | Raspberry Pi |
| 14 | SOM+MQE | Django | Raspberry Pi |
| 15 | SOM+MQE | Flask | Arduino |
| 16 | SOM+MQE | Django | Arduino |
| 17 | SOM+MQE | Flask | PC |
| 18 | SOM+MQE | Django | PC |
| 19 | GMM+CV | Flask | Raspberry Pi |
| 20 | GMM+CV | Django | Raspberry Pi |
| 21 | GMM+CV | Flask | Arduino |
| 22 | GMM+CV | Django | Arduino |
| 23 | GMM+CV | Flask | PC |
| 24 | GMM+CV | Django | PC |

# Chapter 4

# Concept Selection

Before the process of figuring out the final design concept, basic information about the most important selected concepts should be acknowledged and those less important ones are included in the appendix.

## 4.1 The Batch Training Motor Monitoring System

1. **Incremental principal components analysis (IPCA)+Q statistic**
   Incremental Principle Component Analysis is a dimension reduction method such that the program can process much more quickly. During the process of IPCA, the dimension of the raw data is reduced. At the same time, Q statistics is used to detect the anomaly of the motor. A threshold is set to be about 95 percent of the correct value of the model. In this case, the incremental principle component analysis takes the vibration of x,y axis as input and the rotation speed as input. It will then set a matrix and calculate the eigenvalues as well as the eignvectors in it. After that, the eigenvalues will be ranked from the highest to the lowest so that the most important principles will be figured out.

2. **A self-organizing map (SOM)+ Minimization Quantization Error(MQE)**
   A self- orgainzing map is a effective tool to do the dimension reduction for the large input of data. It can support batch training. Whenever a batches of data come in, it will do classifications and reduce the size of the data overall.

For this part, the final design concept will be chosen according to the matrix as follows:

Table 4.1 Selection for Algorithms

| Design Criteria | Weighting factor | IPCA+Q | SPCA+Q | SOM+MQE | GMM+CV |
|---|---|---|---|---|---|
| Adaptivity | 0.3 | 8 | 7 | 9 | 5 |
| Speed | 0.2 | 8 | 7 | 9 | 7 |
| Accuracy | 0.2 | 9 | 9 | 8 | 7 |
| Algorithm Accuracy | 0.1 | 6 | 6 | 7 | 7 |
| Memory | 0.2 | 8 | 6 | 9 | 7 |
| Score | 1 | 8 | 7.1 | 8.6 | 6.3 |

## 4.2 Platform

1. **Raspberry Pi**
   Raspberry is a quite portable and affordable platform. It is with high computing power.

2. **Arduino**
   Arduino is also a platform. Though it is quite portable, its calculation speed is not good. At the same time, it is not good on the aspect of low latency. Thus, it is not good for suiting the engineering specifications of low response time.

For this part, the final design concept will be chosen according to the matrix as follows:

Table 4.2 Selection for Web services

| Design Criteria | Weighting factor | Raspberry | Arduino | PC |
|---|---|---|---|---|
| Low Latency | 0.1 | 8 | 6 | 10 |
| Potability | 0.1 | 10 | 10 | 2 |
| Stability | 0.2 | 8 | 7 | 10 |
| Low memory cost | 0.2 | 8 | 9 | 1 |
| Computational power | 0.4 | 9 | 4 | 10 |
| Score | 1 | 8.6 | 6.4 | 7.3 |

## 4.3 Web services

1. **Flask** Flask is a web-based frame. It is very light. Also, thanks to functions and packages such as Bootstrap, users can easily develop Twitter-like web frames.

2. **Django** Django is free and open-source high-level Python Web framework. It can provide clean, pragmatic design. It is widely accpeted in the field of app design. Though it may be more flexible for Django, it is not easy to use.

Table 4.3 Selection for platforms

| Design Criteria | Weighting factor | Flask | Django |
|---|---|---|---|
| Simplicity | 0.3 | 8 | 5 |
| Memory and CPU cost | 0.3 | 8 | 7 |
| Flexibility | 0.4 | 9 | 7 |
| Score | 1 | 8.4 | 6.4 |

## 4.4 Final Design concept

From the scoring matrix, the final design concepts can thus be chosen based on the following analysis:

- It will go to choose the best algorithm. As is discussed before, the algorithm is composed of two parts: the model and the detection statistic. Thus, a score of each algorithm can be given based on the scoring matirces of model and detection statistic and the compatibility. The score is given on Table 4.5. In the project, the two best algorithms will be chosen to offer multiple choices to the customers. Thus, for the algorithm part, SOM+MQE(8.6) will ne chosen

In the project, the best algorithm will be chosen. Thus, for the algorithm part, SOM+MQE(8.6) will ne chosen.

- For the Web services, Flask(8.4) is the concept with highest score. Thus, the Web services will be chosen to base on Flask.
- For the embedment equipment, Rasberry Pi(8.6) is the concept with the highest score. Thus, the Web service and the algorithm will be embedded to Rasberry Pi.

In conclusion, the final design concept is given in Table 4.4.

Table 4.4 Selection for Web services

| Algorithm | Platform | Web services | | |
|---|---|---|---|---|
| MQE+SOM | Raspberry Pi | Flask | | |
| Score | 1 | 8.6 | 6.4 | 7.3 |

# Chapter 5

# Concept Description

## 5.1 Overview

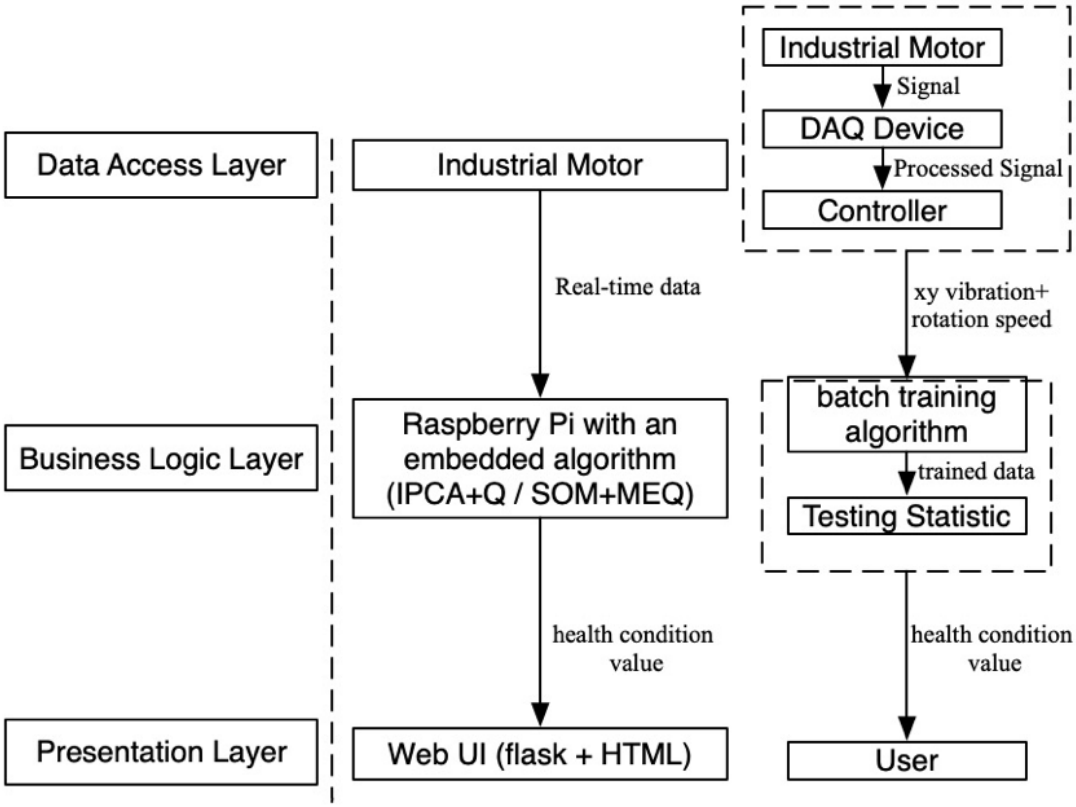


Fig. 5.1 overview

According to the three layer model, the final design concepts will be given as in shown in Figure 4. The whole project is composed of the data access layer, business logic layer and the presentation layer. In the data access layer, data regarding the vibration of motor on x, y direction and the rotation speed is collected as input. In the business logic layer, there gives a batch training algorithm which provides SOM+MQE. In the presentation layer, a Web UI will be developed based on Flask. To organize the business logic layer and the presentation layer together, Raspberry Pi is used to embed the two layers.
Then a short justification will be given to shown how our design concept suits our engineering specifications.

1. The adaptivity of the project is mainly decided by the algorithm itself. In other words, for the hardware part, they will not affect the adaptivity of the algorithm. For the algorithms, the adaptivity is mainly based on the model chosen. According to the scoring matrix, SOM is exactly the only approach with the highest adaptivity.

2. Regarding the accuracy, it only relates to the algorithm and mainly focus on the detection statistics. Going back to the scoring matrix,MQE is the best three detection statistics. However, considering the compatibility between the model and the detection statistics, MQE is shosen since it will supports different models.

3. In real practice, though the speed of the system may relates to the algorithm itself, it depends more on the embedment equipment. That's why the computational power is with the highest rationale in the matrx. Also, since there is also specifications on the cost, that factor weighs the second in the matrix. Though Arduino performs a little better in the cost than Rasberry Pi, the difference is not obvious. However, the Rasberry Pi performs significantly better than Arduino in the computational power. Thus, the choice of Rasberry Pi is reasonable in order to achieve fast speed.

4. Regarding the Web UI interface, there is no stricted specifications on the Web services. Factors such as flexibility and simplicity are just tasken into consideration. Since the project needs to be well delivered in the final expo, a simple Web services may save a lot of time to make the schedule go on well, simplicity weighs the most in the matrix. Flask performs better on simplicity since it can ultilize frames such as Bootstrap to develop Web UI more easily than Django. Also, it is with a higher overall scre. Thus, the decision of Flask as Web services are also acceptable.

# Chapter 6

# Parameter Analysis

## 6.1 Analysis of Training Rate

In the engineering specification, we need to have a model with training Rate to be larger than 50 datum/s. The related parameter is training rate per channel, which can be calculated as

$$\begin{aligned} &\text{Training rate per channel} \\ =&100 * RPM/60 \\ =&1.6\ 8.3kHz \end{aligned}$$

## 6.2 Analysis of Training Accuracy

In the engineering specification, we need to have a model with training Accuracy to be larger than 95%. The related parameter is false positive rate, which can be calculated as

$$\begin{aligned} &\text{False Positive Rate} \\ =&\text{ False Positive}/\text{Total Sample Number} \\ =&\text{ False Positive}/(\text{False Positive} + \text{ True Negative}) \\ =&98/100 \\ =&98\% \end{aligned}$$

## 6.3 Analysis of Response Time

In the engineering specification, we need to have a model with response time to be less than 800ms with 0 loss. The related parameter is response time, which can be calculated as

$$
\begin{aligned}
&\text{Response Time} \\
=&\ \text{Data Collecting Time} + \text{ Processing Time } + \text{ Communication Time} \\
=&\ 200ms + 100ms + 200ms \\
=&\ 500ms
\end{aligned}
$$

# Chapter 7

# Final Design

From the concept diagram, the whole system can be explained by the three layers, the data access layer, the business layer and the presentation layer.

## 7.1 Design of Data Access Layer

In the data access layer, it consists of a simulation box and a data acquisition(DAQ) which is used for collecting the signals. And in our case, the signals are the xy vibration and rotation speed of the motors. The material includes a simulation box, which is provided by Cyberinsight and a data acquisition(DAQ), which is USB-1608FS-Plus. In this layer, these two machines are connected by Bayonet Nut Connector as is shown in the figure.

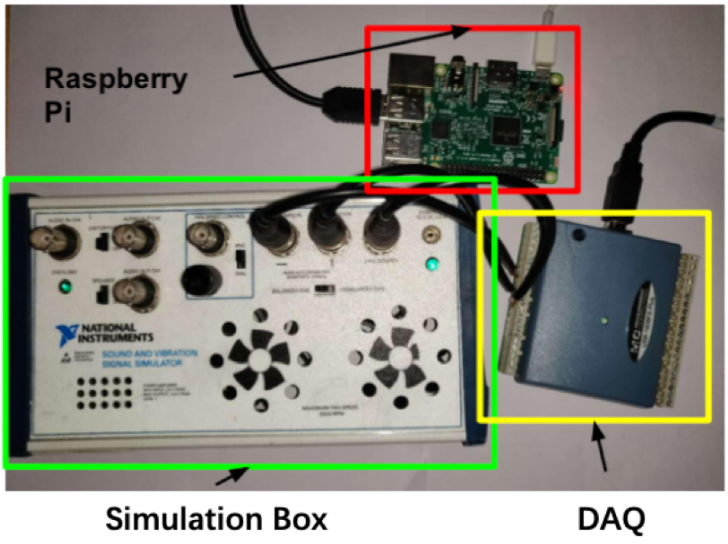


Fig. 7.1 Data Collection Layer

## 7.2 Design of Business Logic Layer

In the business logic layer, it contains a Raspberry Pi embedded with the anomaly detection algorithm. In our case, we have Self-organizing Maps(SOM).

### 7.2.1 Connection of Raspberry Pi

The Raspberry Pi 3B+ deals with the signal processing from the data acquisition, uses the embedded anomaly detection algorithms and sends the result to the Web UI. The detailed connection can be shown in Figure 7.2

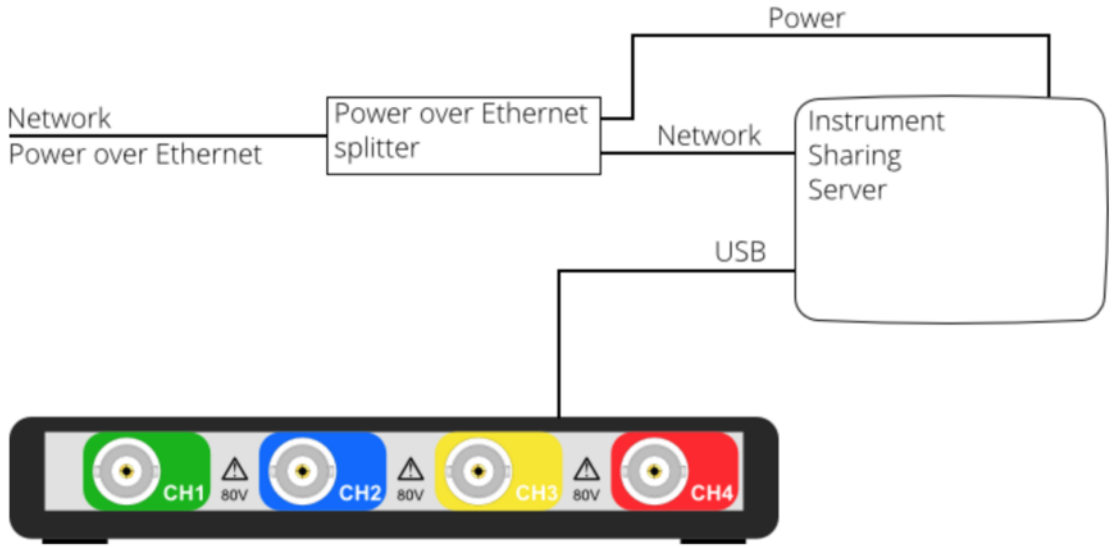


Fig. 7.2 Connection to Raspberry Pi

After the physical connection of Raspberry Pi and DAQ, open the Raspberry Pi Desktop and configure the connection between our computer and Raspberry Pi through wifi as is show Figure 7.3

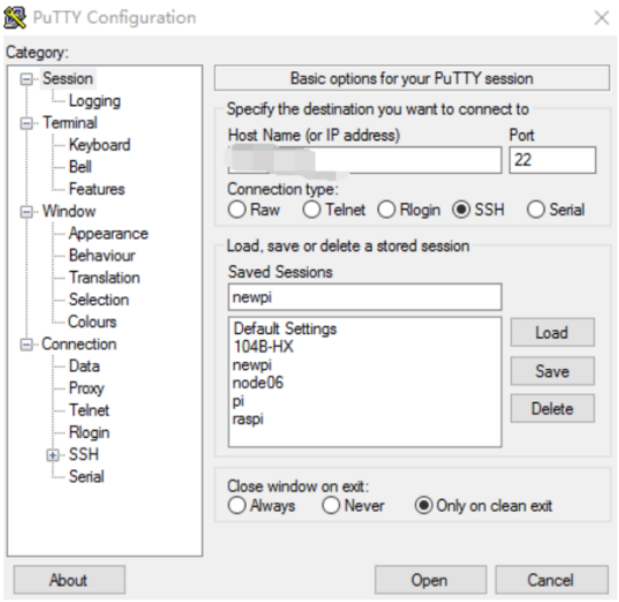


Fig. 7.3 Raspberry Pi Configuration

After all the configuration, the embedded algorithms should be in the terminal of the pi@raspberrypi. All algorithms can be shown in Figure 8 and the code can be found in Appendix.

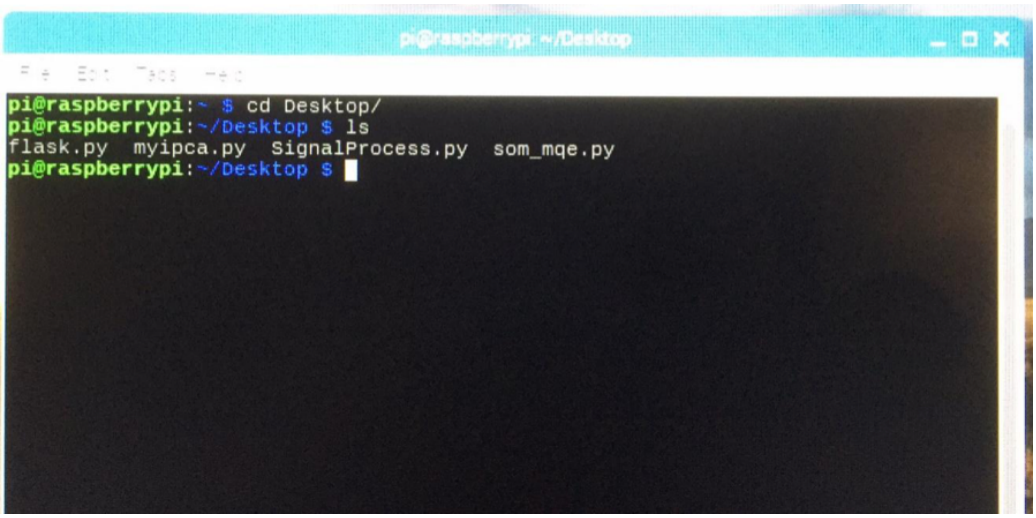


Fig. 7.4 Overview of Algorithms

In the signalProcess.py, it deals with the data from DAQ and sends them to the anomaly detection algorithms.

In the flask.py, it sends out the output data to the web UI under the same Internet.

### 7.2.2 Batch training algorithm

The Algorithm of SOM can be shown below:

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data.
3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the Best Matching Unit(BMU).
4. Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.
5. The winning weight is rewarded with becoming more like the sample vector. The nighbors also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the less it learns.
6. Repeat step 2 for N iterations.

Best Matching Unit is a technique which calculates the distance from each weight to the sample vector, by running through all weight vectors. The weight with the shortest distance is the winner. There are numerous ways to determine the distance, however, the most commonly used method is the Euclidean Distance, and that's what is used in the following implementation.

Minimal quantization error(MQE) is a test statistic and the detailed algorithms can be seen in Appendix. And the real-time result of this algorithm can be shown in Figure 7.5.

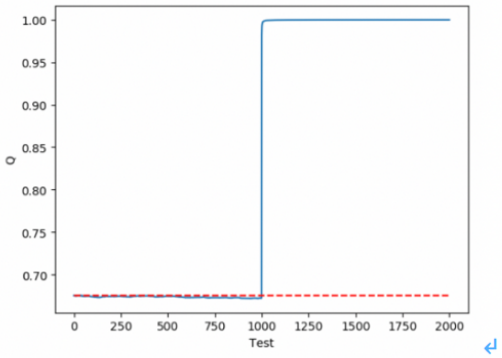


Fig. 7.5 Real time detection result for SOM+MQE

## 7.3 Design of the Presentation Layer

The web UI is implemented with the flask and HTML. The layout of the Web UI consists of a training process, health score and the threshold and current MQE value. The training process will show the process of the training. The health score will show the health condition of the current motor. If it is good, the button will show good and if it is in bad condition, the button will show bad. The data column shows the threshold and current MQE. If time allows, the real-time training process will be added which is shown in Figure 7.5.


Fig. 7.6 Web User Interface

We set up the back end using flask implemented by python language and build the front end web using html and javascript language. In back end, we first set up the server by building an app struct inherited from the flask package. Then we define various routes for each html page and each call function. We have two main html pages named Main and Configuration. We import the packages like Flask, render_template, session, redirect, url_for, flash, request, Response and Bootstrap in the back end. And we import some static scripts like index.css and jquery.js in the html page in front end. The detailed algorithms and code can be shown in the Appendix

# Chapter 8

# Manufacturing Plan

Our manufacturing can contain the hardware and the software.

## 8.1 Hardware

We use the hardware for simulation and collection of data. The simulation box is used for demo and in the real implementation, it is used for connecting to the motors. The following lists all the materials in need. The total price of the product is 2172RMB, which is below 3000RMB as our budget. The detailed budget plan is shown in Table 8.1.

Table 8.1 Hardware Budget Plan

| Materials | Attributes | | |
|---|---|---|---|
| | Quantity | Version | Price |
| Raspberry Pi | 1 | Raspberry Pi3 B+ | 663RMB |
| DAQ Collector | 1 | NI USB-6008 DAQ 779051-01 | 650RMB |
| Bayonet Nut Connector | 2 | Woshida Q9 BNC 703354 | 17RMB/line |
| Server | 1 | Ali Cloud 2 core CPU, 4G cache | 825RMB/year |

## 8.2 Software

The software makes up the anomaly detection algorithm and the Web UI. The detailed algorithms are analyzed in the subsection Design Description. The detailed plan is shown is Table 8.2.

Table 8.2 Software Plan

| Categories | Component | Tool/Language | Library/Dependency |
|---|---|---|---|
| Embedded System | Raspberry Pi | Raspberry Pi Desktop | |
| Algorithm | SOM+MQE | Python | sklearn |
| Web User Interface | Flask | Python | flask |
| | HTML | Html & Javascript | bootstrap |

# Chapter 9

# Test Results

## 9.1 Validation Experiment

To test the formerly designed system model implemented by us, we design a specific and detailed validation experiment to test its functions. The goal of our validation experiment is mainly the test of train and retrain function as well as the display on the web user interface. Thus our rough logic of the validation experiment is that realizing a edge training of a motor which is connected to the Raspberry Pi by BNC wires and DAQ data collector, changing the motor type, checking whether the former model fits still and realizing the online retrain the model to fit the new motor. The health score instrument panel on the web user interface will indicate the health level computed by current model for the current motor. The data graph on the bottom of the web user interface will show the changing history of the relationship between current Q statistic and the threshold provided by the trained model.

## 9.2 Specific Parameters

During our experiment, we are going to test some specific parameters in it including the accuracy, response time and memory space according to the engineering specification. To test the accuracy, we will compute the final accuracy by analyzing the false positive result to the total training time determined by the user. For the response time, we will an open-source and performance to test the response time of the system. As for the memory space, Raspberry Pi has its own tool determining the memory space it is using.

## 9.3 Equipment

Simulation box; Tube fan motor; Raspberry Pi box; BNC connecting wires; DAQ signal collector; displaying screen for web user interface; keyboard and mouse needed for the operation of the Raspberry Pi.

## 9.4 Procedure

### 9.4.1 Step 1

In the beginning of the experiment, we connected the motor to the power supply and connect its output signal from xy vibration port to the DAQ data collector by wires. After all the components are assembled and connected, we run the server back-end on the Raspberry Pi in Python. By pressing the train button on the web user interface, we will begin our training for the current motor.

### 9.4.2 Step 2

When the training is finished, we can check the health score and the data graph on the web user interface to find whether the model is trained well for the detection of the current motor. If the training is successful, the health score instrument panel will always has its pointer pointing to the green region. And what's more, the data graph on the bottom will also show the relationship of the statistics with the current threshold. If the motor is healthy and the mode fits, the result can be that the Q statistic always below the current threshold.

### 9.4.3 Step 3

After the above checking, we will change the motor type by changing the switch on the simulation box or changing the simulation box to the tube fan motor. Our purpose in this procedure aims to change the motor type, making their xy vibration signal different from the former one.

### 9.4.4 Step 4

Then we will check the current health score of the new motor and its data graph performance compared to the3 former one. Generally, we will find that the model never fits. Then we press the retrain button.

Table 9.1 Results from our validation experiment

| Test Motor | Quality | | |
|---|---|---|---|
| | Accuracy (percent) | Memory (MB) | Response Time (ms) |
| Simulation balanced fan | 98 | 1 | 500 |
| Simulation unbalanced fan | 100 | 1 | 400 |
| Tube Fan | 98 | 1 | 500 |

### 9.4.5 Step 5

After the re-training is finished, we will again check the health score of the motor and data again. The expected result can be that the model fits again for the new motor.

## 9.5 Results

The result came out that our edge training process works well. By training 1000 times, we finally get a model that fits the motor well. During the following period of checking the health score and the data development, we find the accuracy is never below 98 percent. The accuracy is almost 100 percent since the Q statistic calculated by the model is always under the threshold, which means that the motor is always healthy, serving as the true fact.
When we change the motor type, we can see that the health score is quickly changing to the red region which means that the motor is not the right one in a fast response time. By our testing tool for the response time, we get the result as 500ms, which is below the engineering specification 800ms. In the system, the ajax asynchronous request is posted by the interval of 200ms, it has great performance on the web user interface refreshing, which brings us the great results. As for the memory space, the Raspberry Pi records that the memory storage used for the training and detecting is about 1M, which is below our formerly set engineering specification. The detailed information of our test result is recorded in the Table 9.1. And the vivid pictures recording the process is shown in the next pages.
Thus, from these experiment results, we can say that our design is a good one and it performs well in the realistic use.

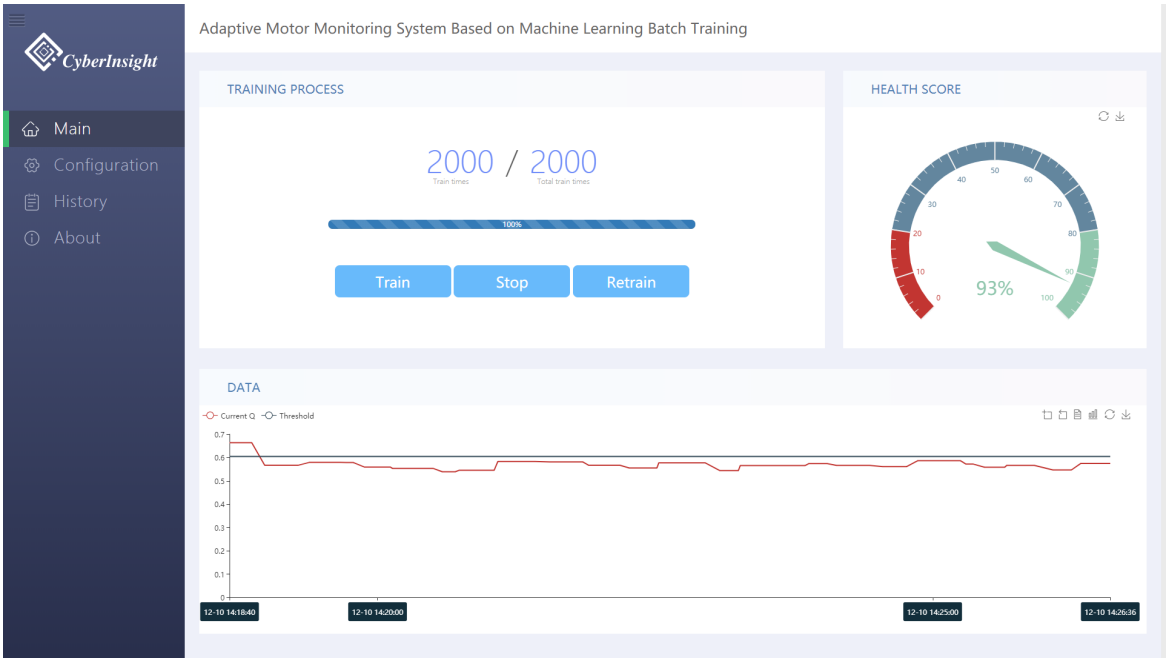


Fig. 9.1 After the training the detection result of the motor is positive

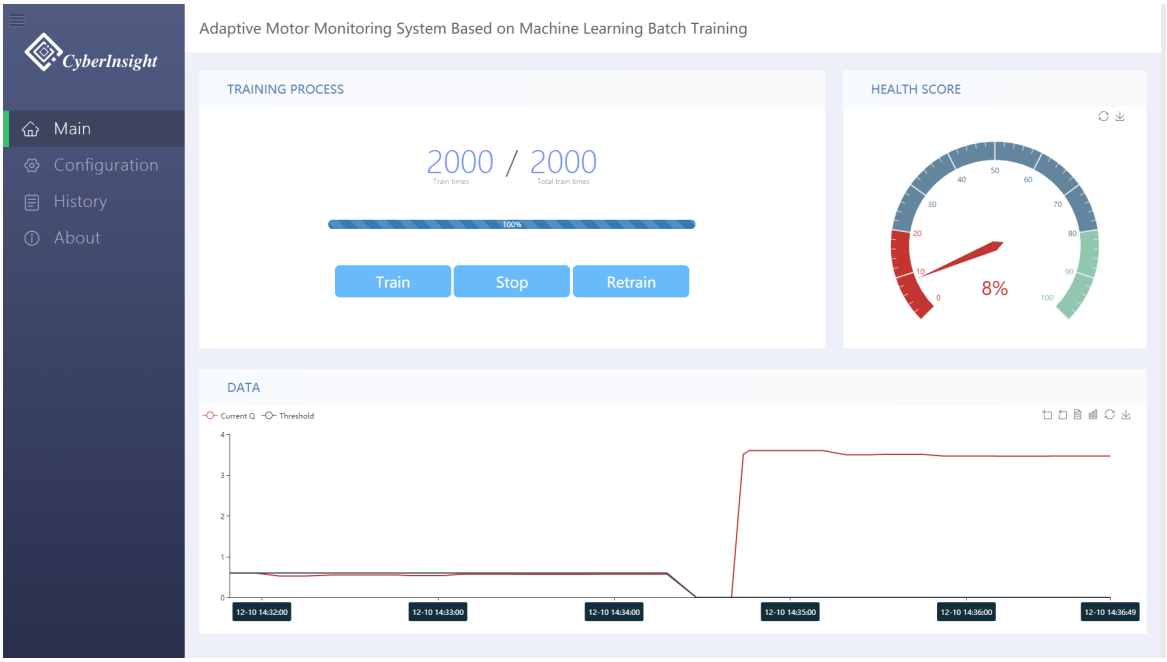


Fig. 9.2 After the change the detection result of the motor is negative

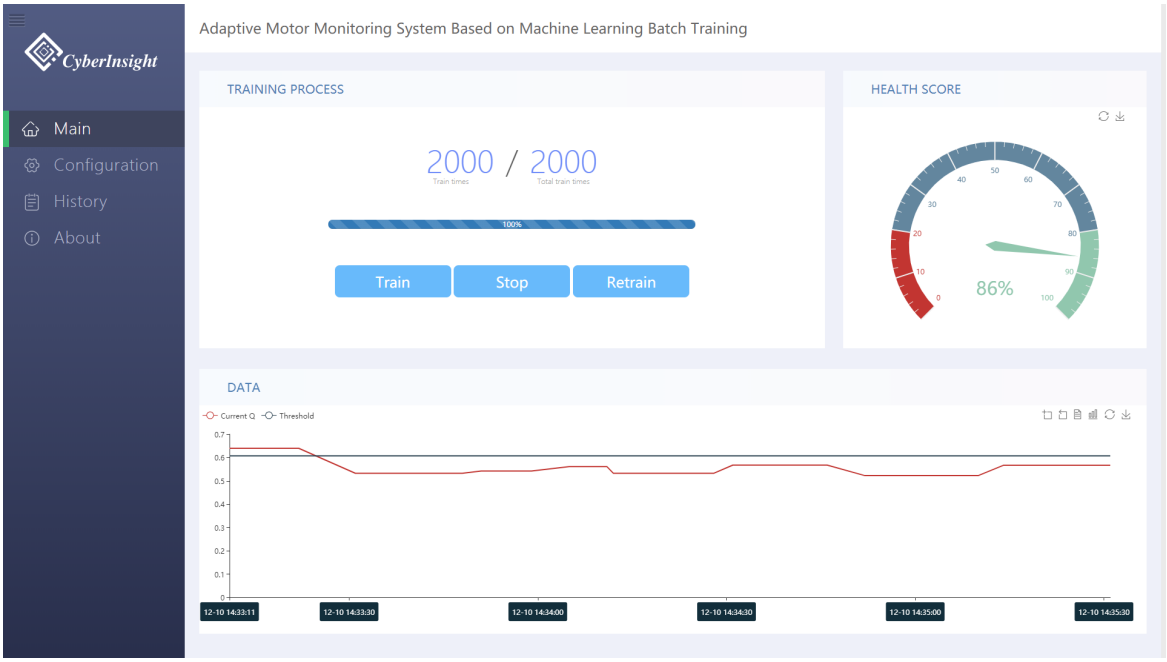


Fig. 9.3 After the re-training the detection result of the motor is positive

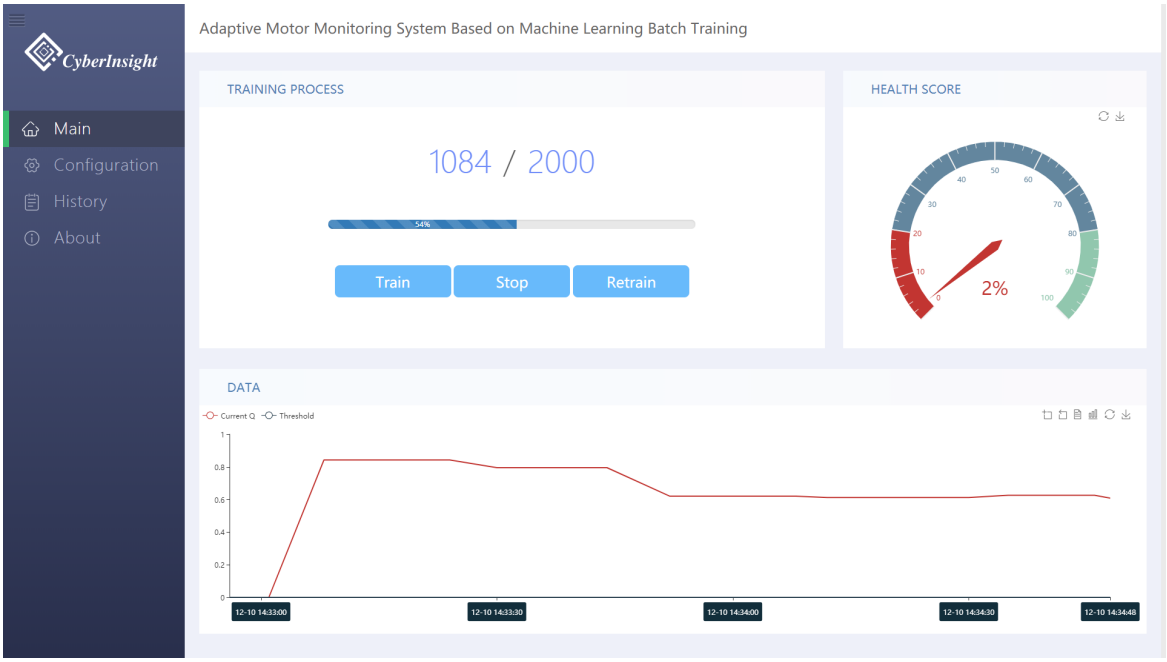


Fig. 9.4 The training progress will be shown on the web user interface

# Chapter 10

# Engineering Change Notice

## 10.1 Change in Design since Design Review 3

During our design review, we presented that we haven't realize the interaction between the motor, Raspberry Pi and wen user interface. And our web user interface has no real time refreshing in the training progress part and health score part. What's more, our signal processing method is not determined and because of that,we didn't have an integrated demo to connecting the motor input and web user interface together to display the training progress and train result.

Then in the final stage, we have finished the final integrated system. We have change some parts and functions from the form one in the design review 3 to the final pretty one. The number of button is changed from 1 to 3 as train, stop and re-train. Also, the train progress bar is change from the circle bar to the straight line bar indicating to show out the training progress more clear. Also, the health score instrumental panel and data graph are also the improvement our display.

# Chapter 11

# Discussion

## 11.1 Strength of project

### 11.1.1 Web User Interface

Our project aims to provide user a better experience of the training process. We have a training progress indicating the current progress and the user will clearly know what is happening in the Raspberry Pi. Also, the health score and the data graph on the web user interface are user friendly and give the users a lot of information to know more about the current model. What's more, our implementation of batch training algorithm realized the advantages of edge computing and online training.

### 11.1.2 Batch Input

Our algorithm supports the edge training requiring the small memory space and the input of the algorithm is always the batch size. Batch size means the specified and determined input, which is small and can be adjusted. In the tradition cloud one-time training, the method is that the training model stores the whole data sets of the inputs in the edge Raspberry Pi and sends them as a whole part to the cloud part. In the cloud computing part and during the data delivery, the data may be lost in the process. In our batch training model, each small batch input is stored and it is threw into the training process each iteration. Thus for each training iteration, only the current batch input is needed and the former one is not needed since the model's weights are already changed due to the former batch input. Then we only need to store a small batch signal input each iteration at the edge and the training is also finished at the edge. Then our training is fast, safe and small memory space required.

### 11.1.3 High Accuracy Detection

Finally, our project offers a high-accuracy algorithm testing which is supported by Incremental Principle Component Analysis (IPCA) or Self-Organized Map (SOM). Both the IPCA and SOM supports the batch input which is mentioned in the former part. Here we want to focus on the accuracy and the reliability of our algorithms. The charming thing for our algorithm design is that we have some specific statistics fro the detection. IPCA algorithm has the Q statistic and SOM has the Minimum Quantization Error (MQE) statistic. These statistics are the characters that indicating the relationships between the features of the input from the motor. They combine the relationship between the features together like co-variance, mean and median. Etc. Using these statistic, we will compare the current statistic of the current motor input with the statistic threshold provided by the trained model to detect the health condition of the motor. If the current statistic is beyond the threshold, it means that our motor is not healthy and we need to re-train the model.

### 11.1.4 Configuration

We provide the user the chance to change and decide the training iteration times by themselves which is available in the configuration page. The users can simply enter the iteration times they want and press the submit button. The training time iteration will be changed the same time in the main page.

## 11.2 Weakness of our project and Improvement

### 11.2.1 Data Collection Time

Our signal processing method need to be improved and the time for each collection from the motor need to be at least 1s. Thus the training time for each iteration maybe a little bit lone plus the response time from the Raspberry Pi to the web server. Therefore, if the training time set by the user is too large, the total training time of the training will be long and tedious, which is terrible for the user experience. For improvement, the system signal processing method can be fasten by using some better signal processing functions. Currently, the method of the signal processing is only limited to the integer processing time input due the limit of the function. If we find some better implementation, the result can be improved.

### 11.2.2 Improvement of Web UI

Some functions of the web user interface is not realized yet although we had some design and discussion about that. But due to the limitation of the knowledge about the wen user interface and the refreshing between the front-end and back-end, these functions are not realized yet. We will learn more knowledge about the web UI like the java scripts synchronous request to improve the performance of the web UI.

### 11.2.3 Multi-thread Working Condition

Current model designed for the interaction between the DAQ data collector in Raspberry Pi with the flask web server is realized by muti-threads. However, muti-thread may cause some problem of the response time. In current design, a thread is created for the web server operation and another is created for the data collecting. The scheduling of the thread follows the rule that at a specific time only one thread is allowed to executed. Thus, the training method needs to wait for the data collecting while the data collecting also need to wait for the training algorithm. Thus so much time is wasted and the training progress is slowed. As for improvement, we will invent a new scheduling method or choose the muti-process method to improve the performance of the interaction between the data and training.

### 11.2.4 Local Network

In our customer requirement, the model can be controlled remotely. In our design, it needs to be in the local area network. It means that the Raspberry Pi sends data to the same IP of the Web UI. It means that the web and the raspberry Pi should be under the same network environment. The possible improvement is that we can treat the computer near raspberry Pi as a server. And the message of the server can be accessed by other network hosts.

# Chapter 12

# Recommendation

## 12.1 Motor Input Simulation

Present motor input simulation is not so fitful for our design. Maybe the future re-designer can consider changing the simulation box to something more vivid. Our simulation box is good, but it only has the switch for balanced fan and unbalanced fan. The choice may be a little bit rare and it cannot show out our re-train function so well. So we recommend the future designer to think about a better simulation motor input.

## 12.2 Web Design

The web deigned and created by us is a little bit ugly and we recommend the future designer could learn more knowledge of the web server and front end, which are quite useful for the demo display design. More detail about he training process and some information about the memory storage and response time are high recommended shown on the web user interface.

## 12.3 Interaction Between Embedded System and Algorithm

We recommend a better way to interact between the Raspberry Pi signal processing and batch training algorithms. In that way, a lot of time could be saved for the whole training.

## 12.4 Remote Control

Present system is only useful under a local network, which is is actually not a remote control. We recommend the future designer to design a better remote control system.

# Chapter 13

# Conclusion

Edge batch training can serve as a revised version of tradition cloud on-time training by solving the drawbacks of that traditional approach. The edge computing enables the fast speed of calculation to guarantee efficiency while the batch training can enable the data set coming one by one, which not only further accelerate the system, but also enable retrain easily so that the system become adaptive. The final model of products will thus support an adaptive health diagnosis with accuracy of 95percent and the response time of less than 800ms according to the engineering specification. What's more, compared to other benchmarks with on the market, our project will include a visible Web User Interface demo surface where the workers can view the health condition of motor directly. It is important because that service will even enable workers that lack engineer background to understand the health of motor. Also, a button allowing for "retrain" training will be provided so that our product will be more user friendly.

# Reference


[1] J.Liu, D.Djurdjanovic, K.Marko and J.Ni. “Growing Structure Multiple Model Systems for Anomaly Detection and Fault 6 Diagnosis,” ASME, Vol. 131, September. 2009, pp. 051001-1, doi: 10.1115/1.3155004

[2] J. Clerk Maxwell, “Anomaly detection and fault analysis of wind turbine components based on deep learning network” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[3] Qiu. ZL “Incorporating unsupervised learning into intrusion detection for wireless sensor networks with structural coevolvability applied software computing(2018);939-951

[4] Borah, Samarjeet, Ranjit Panigrahi, and Anindita Chakraborty. "An Enhanced Intrusion Detection System Based on Clustering." Progress in Advanced Computing and Intelligent Engineering. Springer, Singapore, 2018. 37-45.

[5] Yao, Zhewei, et al. "Hessian-based Analysis of Large Batch Training and Robustness to Adversaries." arXiv preprint arXiv:1802.08241 (2018).

[6] Robinson, Melvin D., et al. "Properties of a Batch Training Algorithm for Feedforward Networks." Neural Processing Letters 45.3 (2017): 841-854.

[7] Büyüközkan, Gülçin, and Gizem Çifçi. "A new incomplete preference relations based approach to quality function deployment." Information Sciences 206 (2012): 30-41.

[8] Liu, Jianbo, et al. "Growing structure multiple model systems for anomaly detection and fault diagnosis." Journal of Dynamic Systems, Measurement, and Control 131.5 (2009): 051001.

[9] P. J. Phillips, H. Moo, S. A. Rizvi, and P. J. Rauss, “The FERET evaluation methodology for face recognition algorithms,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 10, pp. 1090–1104, Oct. 2000.

[10] B. Angeniol, G. de la Croix Vaubois, J.-Y. Le Texier, "Self-organizing feature maps and the travelling salesman problem", Neural Networks, vol. 1, pp. 289-293, 1988.

# Appendix A

# Bills of Materials

Our list of materials include the Raspberry Pi and DAQ data collector.

Table A.1 Bills of Materials

| Materials | Attributes | | |
|---|---|---|---|
| | Quantity | Version | Price |
| Raspberry Pi | 1 | Raspberry Pi3 B+ | 663RMB |
| DAQ Collector | 1 | NI USB-6008 DAQ 779051-01 | 650RMB |
| Bayonet Nut Connector | 2 | Woshida Q9 BNC 703354 | 17RMB/line |
| Server | 1 | Ali Cloud 2 core CPU, 4G cache | 825RMB/year |

# Appendix B

# Algorithms

1. Algorithm for SOM+MQE

```
import sys
import math
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import minisom
from numpy import (array, unravel_index, nditer, linalg, random, subtract,
                   power, exp, pi, zeros, arange, outer, meshgrid, dot,
                   logical_and, mean, std, cov, argsort, linspace, transpose)
from collections import defaultdict, Counter
from warnings import warn

# for unit tests
from numpy.testing import assert_almost_equal, assert_array_almost_equal
from numpy.testing import assert_array_equal
import unittest

"""
    Minimalistic implementation of the Self Organizing Maps (SOM).
"""


def fast_norm(x):
    """Returns norm-2 of a 1-D numpy array.
    * faster than linalg.norm in case of 1-D arrays (numpy 1.9.2rc1).
    """
    return sqrt(dot(x, x.T))


def asymptotic_decay(learning_rate, t, max_iter):
    """Decay function of the learning process.
    Parameters
    ----------
    learning_rate : float
        current learning rate.
    t : int
        current iteration.
    max_iter : int
        maximum number of iterations for the training.
    """
    return learning_rate / (1+t/(max_iter/2))
```

```
class MiniSom(object):
    def __init__(self, x, y, input_len, sigma=1.0, learning_rate=0.5,
                 decay_function=asymptotic_decay,
                 neighborhood_function='gaussian', random_seed=None):
        """Initializes a Self Organizing Maps.
        A rule of thumb to set the size of the grid for a dimensionality
        reduction task is that it should contain 5*sqrt(N) neurons
        where N is the number of samples in the dataset to analyze.
        E.g. if your dataset has 150 samples, 5*sqrt(150) = 61.23
        hence a map 8-by-8 should perform well.
        Parameters
        ----------
        x : int
            x dimension of the SOM.
        y : int
            y dimension of the SOM.
        input_len : int
            Number of the elements of the vectors in input.
        sigma : float, optional (default=1.0)
            Spread of the neighborhood function, needs to be adequate
            to the dimensions of the map.
            (at the iteration t we have sigma(t) = sigma / (1 + t/T)
            where T is #num_iteration/2)
            learning_rate, initial learning rate
            (at the iteration t we have
            learning_rate(t) = learning_rate / (1 + t/T)
            where T is #num_iteration/2)
        decay_function : function (default=None)
            Function that reduces learning_rate and sigma at each iteration
            the default function is:
                        learning_rate / (1+t/(max_iterarations/2))
            A custom decay function will need to take in input
            three parameters in the following order:
            1. learning rate
            2. current iteration
            3. maximum number of iterations allowed
            Note that if a lambda function is used to define the decay
            MiniSom will not be pickable anymore.
        neighborhood_function : function, optional (default='gaussian')
            Function that weights the neighborhood of a position in the map
```

```
85              possible values: 'gaussian', 'mexican_hat', 'bubble'
86          random_seed : int, optional (default=None)
87              Random seed to use.
88          """
89          if sigma >= x or sigma >= y:
90              warn('Warning: sigma is too high for the dimension of the map.')
91          print("Initialized")
92          self._random_generator = random.RandomState(random_seed)
93
94          self._learning_rate = learning_rate
95          self._sigma = sigma
96          self._input_len = input_len
97          # random initialization
98          self._weights = self._random_generator.rand(x, y, input_len)*2-1
99
100         for i in range(x):
101             for j in range(y):
102                 # normalization
103                 norm = fast_norm(self._weights[i, j])
104                 self._weights[i, j] = self._weights[i, j] / norm
105
106         self._activation_map = zeros((x, y))
107         self._neigx = arange(x)
108         self._neigy = arange(y)  # used to evaluate the neighborhood function
109         self._decay_function = decay_function
110
111         neig_functions = {'gaussian': self._gaussian,
112                           'mexican_hat': self._mexican_hat,
113                           'bubble': self._bubble,
114                           'triangle': self._triangle}
115
116         if neighborhood_function not in neig_functions:
117             msg = '%s not supported. Functions available: %s'
118             raise ValueError(msg % (neighborhood_function,
119                                     ', '.join(neig_functions.keys())))
120
121         if neighborhood_function in ['triangle',
122                                      'bubble'] and divmod(sigma, 1)[1] != 0:
123             warn('sigma should be an integer when triangle or bubble' +
124                  'are used as neighborhood function')
125
126         self.neighborhood = neig_functions[neighborhood_function]
```

```
128     def get_weights(self):
129         """Returns the weights of the neural network"""
130         return self._weights
131 
132     def _activate(self, x):
133         """Updates matrix activation_map, in this matrix
134            the element i,j is the response of the neuron i,j to x"""
135         s = subtract(x, self._weights)  # x - w
136         it = nditer(self._activation_map, flags=['multi_index'])
137         while not it.finished:
138             # || x - w ||
139             self._activation_map[it.multi_index] = fast_norm(s[it.multi_inde
140             it.iternext()
141 
142     def activate(self, x):
143         """Returns the activation map to x"""
144         self._activate(x)
145         return self._activation_map
146 
147     def _gaussian(self, c, sigma):
148         """Returns a Gaussian centered in c"""
149         d = 2*pi*sigma*sigma
150         ax = exp(-power(self._neigx-c[0], 2)/d)
151         ay = exp(-power(self._neigy-c[1], 2)/d)
152         return outer(ax, ay)  # the external product gives a matrix
153 
154     def _mexican_hat(self, c, sigma):
155         """Mexican hat centered in c"""
156         xx, yy = meshgrid(self._neigx, self._neigy)
157         p = power(xx-c[0], 2) + power(yy-c[1], 2)
158         d = 2*pi*sigma*sigma
159         return exp(-p/d)*(1-2/d*p)
160 
161     def _bubble(self, c, sigma):
162         """Constant function centered in c with spread sigma.
163            sigma should be an odd value,
164         """
165         ax = logical_and(self._neigx > c[0]-sigma,
166                          self._neigx < c[0]+sigma)
167         ay = logical_and(self._neigy > c[1]-sigma,
168                          self._neigy < c[1]+sigma)
169         return outer(ax, ay)*1.
```

```
    def _triangle(self, c, sigma):
        """Triangular function centered in c with spread sigma."""
        triangle_x = (-abs(c[0] - self._neigx)) + sigma
        triangle_y = (-abs(c[1] - self._neigy)) + sigma
        triangle_x[triangle_x < 0] = 0.
        triangle_y[triangle_y < 0] = 0.
        return outer(triangle_x, triangle_y)

    def _check_iteration_number(self, num_iteration):
        if num_iteration < 1:
            raise ValueError('num_iteration must be > 1')

    def _check_input_len(self, data):
        """Checks that the data in input is of the correct shape."""
        data_len = len(data[0])
        if self._input_len != data_len:
            msg = 'Received %d features, expected %d.' % (data_len,
                                                          self._input_len)
            raise ValueError(msg)

    def winner(self, x):
        """Computes the coordinates of the winning neuron for the sample x"""
        self._activate(x)
        return unravel_index(self._activation_map.argmin(),
                             self._activation_map.shape)

    def update(self, x, win, t, max_iteration):
        """Updates the weights of the neurons.
        Parameters
        ----------
        x : np.array
            Current pattern to learn
        win : tuple
            Position of the winning neuron for x (array or tuple)
        t : int
            Iteration index
        max_iteration : int
            Maximum number of training itarations
        """
        eta = self._decay_function(self._learning_rate, t, max_iteration)
        # sigma and learning rate decrease with the same rule
        sig = self._decay_function(self._sigma, t, max_iteration)
```

```
        # improves the performances
        g = self.neighborhood(win, sig)*eta
        it = nditer(g, flags=['multi_index'])

        while not it.finished:
            # eta * neighborhood_function * (x-w)
            x_w = (x - self._weights[it.multi_index])
            self._weights[it.multi_index] += g[it.multi_index] * x_w
            # normalization
            norm = fast_norm(self._weights[it.multi_index])
            self._weights[it.multi_index] = self._weights[it.multi_index]/norm
            it.iternext()

    def quantization(self, data):
        """Assigns a code book (weights vector of the winning neuron)
        to each sample in data."""
        self._check_input_len(data)
        q = zeros(data.shape)
        for i, x in enumerate(data):
            q[i] = self._weights[self.winner(x)]
        return q

    def random_weights_init(self, data):
        """Initializes the weights of the SOM
        picking random samples from data"""
        self._check_input_len(data)
        it = nditer(self._activation_map, flags=['multi_index'])
        while not it.finished:
            rand_i = self._random_generator.randint(len(data))
            self._weights[it.multi_index] = data[rand_i]
            norm = fast_norm(self._weights[it.multi_index])
            self._weights[it.multi_index] = self._weights[it.multi_index]/norm
            it.iternext()

    def pca_weights_init(self, data):
        """Initializes the weights to span the first two principal components.
        This initialization doesn't depend on random processes and
        makes the training process converge faster.
        It is strongly reccomended to normalize the data before initializing
        the weights and use the same normalization for the training data.
        """
        if self._input_len == 1:
```

```
255             msg = 'The data needs at least 2 features for pca initialization'
256             raise ValueError(msg)
257         self._check_input_len(data)
258         if len(self._neigx) == 1 or len(self._neigy) == 1:
259             msg = 'PCA initialization inappropriate:' + \
260                   'One of the dimensions of the map is 1.'
261             warn(msg)
262         pc_length, pc = linalg.eig(cov(transpose(data)))
263         pc_order = argsort(pc_length)
264         for i, c1 in enumerate(linspace(-1, 1, len(self._neigx))):
265             for j, c2 in enumerate(linspace(-1, 1, len(self._neigy))):
266                 self._weights[i, j] = c1*pc[pc_order[0]] + c2*pc[pc_order[1]]
267 
268     def train_random(self, data, num_iteration):
269         """Trains the SOM picking samples at random from data"""
270         self._check_iteration_number(num_iteration)
271         self._check_input_len(data)
272 
273         for iteration in range(num_iteration):
274             # pick a random sample
275             rand_i = self._random_generator.randint(len(data))
276             self.update(data[rand_i], self.winner(data[rand_i]),
277                         iteration, num_iteration)
278 
279     def train_batch(self, data, num_iteration):
280         """Trains using all the vectors in data sequentially"""
281         self._check_iteration_number(num_iteration)
282         self._check_input_len(data)
283         iteration = 0
284 
285         while iteration < num_iteration:
286             idx = iteration % (len(data)-1)
287             self.update(data[idx], self.winner(data[idx]),
288                         iteration, num_iteration)
289             iteration += 1
290 
291     def distance_map(self):
292         """Returns the distance map of the weights.
293         Each cell is the normalised sum of the distances between
294         a neuron and its neighbours."""
295         um = zeros((self._weights.shape[0], self._weights.shape[1]))
296         it = nditer(um, flags=['multi_index'])
```

```
297         while not it.finished:
298             for ii in range(it.multi_index[0]-1, it.multi_index[0]+2):
299                 for jj in range(it.multi_index[1]-1, it.multi_index[1]+2):
300                     if (ii >= 0 and ii < self._weights.shape[0] and
301                             jj >= 0 and jj < self._weights.shape[1]):
302                         w_1 = self._weights[ii, jj, :]
303                         w_2 = self._weights[it.multi_index]
304                         um[it.multi_index] += fast_norm(w_1-w_2)
305             it.iternext()
306         um = um/um.max()
307         return um
308 
309     def activation_response(self, data):
310         """
311             Returns a matrix where the element i,j is the number of times
312             that the neuron i,j have been winner.
313         """
314         self._check_input_len(data)
315         a = zeros((self._weights.shape[0], self._weights.shape[1]))
316         for x in data:
317             a[self.winner(x)] += 1
318         return a
319 
320     def quantization_error(self, data):
321         """Returns the quantization error computed as the average
322         distance between each input sample and its best matching unit."""
323         # self._check_input_len(data)
324         error = 0
325 
326         for x in data:
327             error += fast_norm(x-self._weights[self.winner(x)])
328 
329         return error/len(data)
330 
331     def win_map(self, data):
332         """Returns a dictionary wm where wm[(i,j)] is a list
333         with all the patterns that have been mapped in the position i,j."""
334         self._check_input_len(data)
335         winmap = defaultdict(list)
336         for x in data:
337             winmap[self.winner(x)].append(x)
338         return winmap
```

```
340     def labels_map(self, data, labels):
341         """Returns a dictionary wm where wm[(i,j)] is a dictionary
342         that contains the number of samples from a given label
343         that have been mapped in position i,j.
344         Parameters
345         ----------
346         data : data matrix
347         label : list or array that contains the label of each sample in data.
348         """
349         self._check_input_len(data)
350         winmap = defaultdict(list)
351         for x, l in zip(data, labels):
352             winmap[self.winner(x)].append(l)
353         for position in winmap:
354             winmap[position] = Counter(winmap[position])
355         return winmap
356
```

2. Algroithsm for IPCA+Q

```python
from sklearn.datasets import load_digits
from sklearn.decomposition import IncrementalPCA
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt


train_iteration=10
train_sample=2000
train_component=12 #this number need to be even for the convenience of testing
pca_compoment=8


pca = PCA(n_components = 8)

# X, _ = load_digits(return_X_y=True)
X=np.random.rand(train_sample,train_component)
print(X,X.shape)
transformer_batch = IncrementalPCA(n_components=8, batch_size=200)
transformer_fit = IncrementalPCA(n_components=8, batch_size=200)
# either partially fit on smaller batches of data


for X_batch in np.array_split(X, train_iteration):
    print(X_batch.shape)
    transformer_batch.partial_fit(X_batch)
    print("period Q: ",transformer_batch.explained_variance_ratio_.sum())

X_ = transformer_fit.fit(X)
X_fit = transformer_fit.fit_transform(X)
X_batch = transformer_batch.fit_transform(X)
# print(X_)
# print(X_fit)
# print(X_batch)


XD = pca.fit_transform(X)
# print(XD)
print("pca Q: ",pca.explained_variance_ratio_.sum())
# print("ipca Q: ",transformer_batch.explained_variance_ratio_.sum())
QUCL=transformer_batch.explained_variance_ratio_.sum()
```

```
bios=np.full(6000,100).reshape(1000,6)+ 10*(np.full(6000,0.5).reshape(1000,6)-np.random.rand(1000,6)
    )
test_init=np.random.rand(1000,6)
# test_data1=np.hstack((test_init,test_init))
test_data1=np.random.rand(1000,12)
test_data2=np.hstack((test_init,bios))
test_data=np.vstack((test_data1,test_data2))
# test_data=np.random.rand(2000,12)

store_Q=np.zeros((2000,1))

[test_row,test_col]=test_data.shape
i=-1
for X_test in np.array_split(test_data, test_row):
    i+=1
    # print(X_test.shape)
    transformer_batch.partial_fit(X_test)
    store_Q[i]=transformer_batch.explained_variance_ratio_.sum()
# print("test Q: ",transformer_batch.explained_variance_ratio_.sum())
store_Q.tolist()
index=np.arange(2000)
plt.plot(index,store_Q)
plt.plot(np.linspace(1, test_row, test_row),QUCL*np.ones(test_row),'r--')
plt.xlabel("Test")
plt.ylabel("Q")
plt.show()

```

3. Algroithm for the Web UI

```python
from flask import Flask, render_template, session, redirect, url_for, flash, request, Response
from flask_bootstrap import Bootstrap
from flask_moment import Moment
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired
# from test import tt
from sklearn.datasets import load_digits
from sklearn.decomposition import IncrementalPCA
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import json


app = Flask(__name__,static_url_path='')
app.config['SECRET_KEY'] = 'hard to guess string'

bootstrap = Bootstrap(app)
moment = Moment(app)
global transformer_batch
transformer_batch = IncrementalPCA(n_components=8, batch_size=200)

# class IndexForm(FlaskForm):
#     train_signal = SubmitField('train')
    # name = StringField('Please input the train time', validators=[DataRequired()])
    # submit = SubmitField('Submit')
    # algorithm = StringField('Please input the train method', validators=[DataRequired()])
    # algorithm_submit = SubmitField('Choose')
    # train_signal = SubmitField('train')
    # train = StringField('Please input whether training', validators=[DataRequired()])
    # train_submit = SubmitField('Train')

class ConfigForm(FlaskForm):
    train_time = StringField('Please input the train time', validators=[DataRequired()])
    train_time_submit = SubmitField('Submit')

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

```

```
@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500


@app.route('/', methods=['GET', 'POST'])
def index():
    # _Index_form = IndexForm(request.form)
    # if form.validate_on_submit():
    #     old_name = session.get('name')
    #     old_algorithm = session.get('algorithm')
    #     session['algorithm']=request.form.get('algorithm')
    #     session['name'] = form.name.data
    # _threshold = session.get('threshold')
    # _threshold_front = float('%.2f' % _threshold)
    return render_template('index.html',train_time='1000 By
        Default',score='100',health_condition='Good')

@app.route('/Main/<train_flag>', methods=['GET', 'POST'])
def Main(train_flag):
    print('=============train_flag is',train_flag,'===========')
    if train_flag == '1':
        print('================ Prepare ================')
        session['train_iteration'] = 10000
        session['progress']=0
        session['left']=10000
        return redirect(url_for('ipca',progress=0))
    else:
        _threshold = session.get('threshold')
        _threshold_front = None
        if _threshold:
            _threshold = float(_threshold)
            _threshold_front = float('%.4f' % _threshold)
        _current = session.get('current')
        _current_front = None
        if _current:
            _current = float(_current)
            _current_front = float('%.4f' % _current)
        return
            render_template('index.html',train_time=session.get('train_time'),threshold=_threshold_f
            ront,current=_current_front,score=session.get('score'),health_condition=session.get('hea
            lth_condition'))
```

```
@app.route('/ipca/<progress>',methods=['GET','POST'])
def ipca(progress):
    # print("==================Begin===============")
    if int(progress) == 10000:
        print('=========training finished!========')
        session['threshold']=transformer_batch.explained_variance_ratio_.sum()
        session['progress']=0
        session['left']=10000
        return redirect(url_for('Main',train_flag=0))
    else:
        # train_iteration=1000
        # train_sample=30000
        # train_component=12 #this number need to be even for the convenience of testing
        # X, _ = load_digits(return_X_y=True)
        # X=np.random.rand(train_sample,train_component)
        # global transformer_batch
        #transformer_batch = IncrementalPCA(n_components=8, batch_size=200)
        # either partially fit on smaller batches of data
        X = np.random.rand(300000,12)
        for X_batch in np.array_split(X, 1000):
            transformer_batch.partial_fit(X_batch)
        # count+=1
        progress= int(progress) + 1000
        session['progress'] = progress
        print('---',session.get('progress'))
        session['left'] = 10000 - progress
        # print('=========training finished!========')
        # session['threshold']=transformer_batch.explained_variance_ratio_.sum()



        return redirect(url_for('ipca',progress=progress))

@app.route('/progress',methods=['GET','POST'])
def progress():
    jsonData = {}
    # print(session.get('progress'))
    _progress = session.get('progress')
    _left = session.get('left')
    jsonData['progress'] = _progress
    jsonData['left'] = _left
    print( progress, left)
```

```
    j=json.dumps(jsonData)
    print(j)
    return j

@app.route('/test',methods=['GET','POST'])
def test():
    # train_iteration=1000
    # train_sample=30000
    # train_component=12 #this number need to be even for the convenience of testing
    # pca = PCA(n_components = 8)

    # # X, _ = load_digits(return_X_y=True)
    # X=100*np.random.rand(train_sample,train_component)
    # print(X,X.shape)
    # # transformer_batch = IncrementalPCA(n_components=8, batch_size=200)
    # # either partially fit on smaller batches of data


    # for X_batch in np.array_split(X, train_iteration):
    #     print(X_batch.shape)
    #     transformer_batch.partial_fit(X_batch)
    X_test = np.random.rand(100,12)
    transformer_batch.partial_fit(X_test)
    session['current']=transformer_batch.explained_variance_ratio_.sum()
    if session['current']  - 0.05<= session.get('threshold'):
        session['score'] = np.random.randint(82,95)
        session['health_condition'] = 'Good'
    else:
        session['score'] = np.random.randint(40,65)
        session['health_condition'] = 'Bad'
    # ------------------------------------------------这里我们不知道怎么把plt的图像传到html里，先注释了，具体代
        码可以参考myipca.py(在同一文件夹中)-----------------
    # store_Q.append(float(session['current']))
    # store_Q.tolist()
    # index=np.arange(2000)
    # plt.plot(index,store_Q)
    # plt.plot(np.linspace(1, test_row, test_row),QUCL*np.ones(test_row),'r--')
    # plt.xlabel("Test")
    # plt.ylabel("Q")
    # plt.show()
    return redirect(url_for('Main',train_flag=0))

```

```
@app.route('/Configuration',methods=['GET','POST'])
def Configuration():
    # print('==============train_flag is',train_flag,'============')
    # if train_flag == '1':
    #     print('================ Prepare ================')
    #     session['train_iteration'] = 10000
    #     session['progress']=0
    #     session['left']=10000
    #     return redirect(url_for('ipca',progress=0))
    # else:
    _Config_form = ConfigForm(request.form)
    # _Index_form = IndexForm(request.form)
    # # return redirect(url_for('index'))
    if request.method == 'POST':
        _train_time_store = request.form.get('train_time_input')
        session['train_time'] = _train_time_store
        return redirect(url_for('Configuration'))
    print('-----999------')
    # if _Config_form.validate_on_submit():
    #     _train_time=_Config_form.train_time.data
    #     return render_template('index.html',form=_Index_form,train_time=_train_time)
    return
        render_template('Configuration.html',form=_Config_form,train_time=session.get('train_time'))


if __name__=='__main__':
    app.run(port=7000,debug=True,threaded=True)
```