UNIVERSITY OF GUELPH

The Analysis of Task Assignment with Unsupervised Self Organizing Map

by

Zimo Zhou (1189792)

Uday Singh (1173955)

Yaowen Mei (1177855)

A thesis submitted in partial fulfillment for the
Summer 2021 ENGG6570 Course

in the
School of Engineering

August 2021

Abstract

Dynamic task scheduling is a very famous open question in computer science, it is consisting in generating a task arrangement for scheduling K robots to accomplish M tasks with the minimum possible total traveling length. The same as the famous traveling sales person problem, this is also a NP-Complete problem. This implies that the time requirement for solving this problem increased rapidly as the number of tasks and number of robots increase. Due to this fact, soft computing, especially self organizing map (SOM), could possibly shining a light on finding the optimal or near optimal solution of this problem. In this paper, different SOM output layer structure are explored (2D output layer for color map, and 1D output layer for robot task assignment), and different quantitative SOM accuracy evaluation methods are also compared.

Key Words: SOM, Algorithms, NP-Complete, Soft Computing

Contents

List of Figures

List of Tables

Chapter 1

Introduction of the Task Scheduling Problem

In modern era, with the development of artificial intelligent and machine learning technology, robots are becoming deeply integrated into people's life. Therefore, robot path planning is a hot topic that attacks a lot of scientists attention. Among all these robot path planning problems, due to the wide potential of application, **task scheduling** is one of the most famous question that that has been under debate for long time. There are many versions of problem statement for this task scheduling problem, but one of the most basic and fundamental format is stated as the following:

1.1 Problem Statement

Assume there are K homogeneous robots and M targets randomly distributed in a bounded 2D space. Each target requires **at least one** robot, and each of the robots need to be assigned to one target (in this basic format, we are not considering the situation where one robot can do multiply tasks). This is to say, the task assignment, f , is a map on $\mathbf{R^2}$ from set **Robots** to set **Tasks**.

$$f : \mathbf{Robots} \rightarrow \mathbf{Tasks} \tag{1.1}$$

The cost function, C, is defined as the summation of robots' travelling path from their origin to their assigned tasks:

$$C = \sum_{\text{robots}} \text{Path} \quad \text{Length} \tag{1.2}$$

The challenge is to develop a good algorithm to find a traveling path for all the robots with the minimum or near minimal cost.

A visual illustration is given as shown in the picture 1.1, all the M green squares are tasks (in this case, $M = 5$, and tasks are labeled as T_0 to T_4) need to be accomplished, and all the K red dots are robots (in this case, $K = 8$, and robots are labeled as R_0 to R_8) that need to be assigned for a task. All the initial position for tasks and robots are randomly generated in the range of $[0, 5]$, and the arrangement with the lowest cost is la-bled with red lines which are generate from a Python brute-force program .

1.2 The Brute-force Method

Theoretically, at small scale, we can solve this problem and find the best solution with the Brute-force method. The time complexity will be bound by $O(M^K)$. Same as the famous traveling sales person (TSP) problem, this is a NP-complete problem as well.

1.2.1 Entropy of this problem

According to Shannon's information theory[6], for a random variable X in a finite set χ with probability distribution $p(x)$, the Shannon's entropy can be written as:

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 (p(x)) \tag{1.3}$$

For each of the robots, it will have at most M choice to choose from as its the target. The total number of possible of arrangement is at most M^K. With considering the boundary condition that each of these M target need to have at least one robot, we can reduce the total number of possible of arrangement, χ, as:

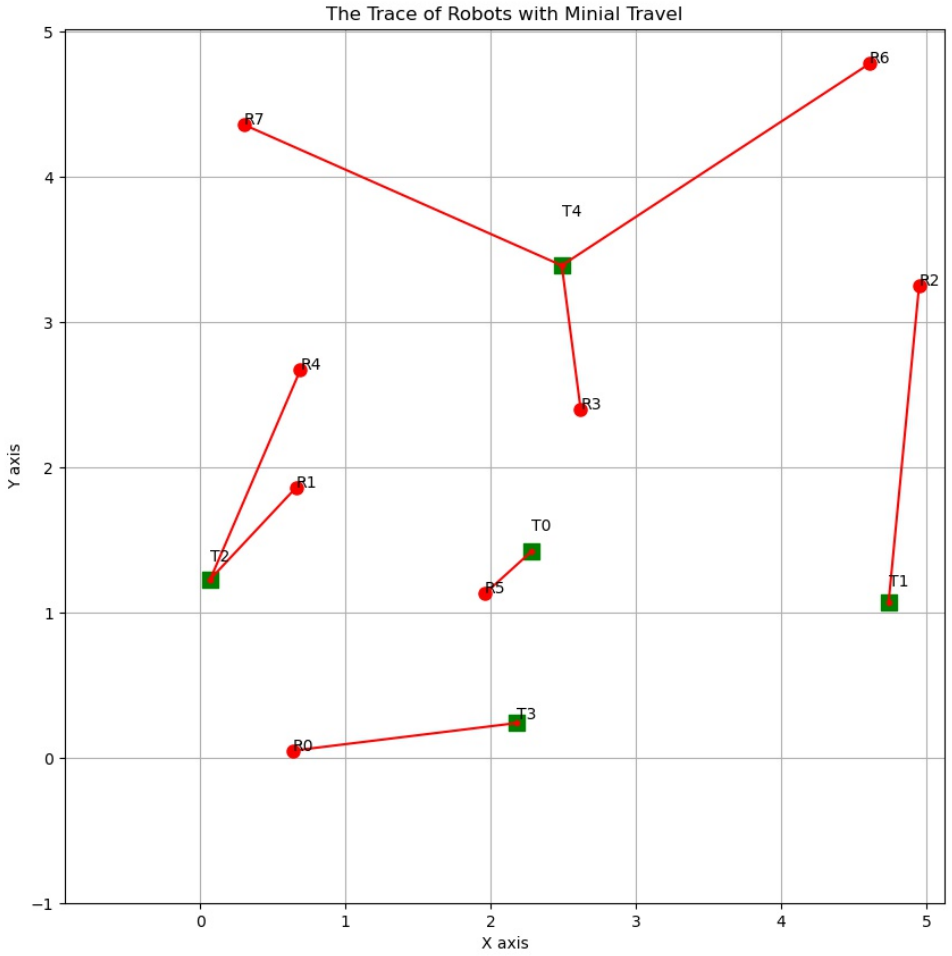

FIGURE 1.1: The illustration of Task Scheduling Problem with 5 tasks and 8 robots.

$$\begin{aligned} \chi &= \underbrace{M\,(M-1)\,(M-2)\ldots 1}_{M!} \times C_{K-M}^{K} M^{K-M} \\ &= M! \frac{K!}{(K-K+M)!(K-M)!} M^{K-M} \\ &= \frac{K!}{(K-M)!} M^{K-M} \end{aligned} \tag{1.4}$$

1.2.2 Time Analysis and Entropy Analysis

With a home used i7-9700K CPU (8 Cores, 3.6GHz), the maximum scale size of this problem that can be solved without memory leaking is 2 tasks and 20 robots (See Figure 1.2), or, 8 robots and 8 tasks. Table 1.1 is a summaries the time complexity of this problem with the Brute-force method:

TABLE 1.1: The time complexity analysis of this problem with 2 tasks, and $K = 2^n$ robots

M	K	Time	χ	$H(\chi)$
2	2	0.0005	2	1.000
2	4	0.0005	14	3.807
2	8	0.001488	254	7.988
2	16	0.65571	65534	15.999
2	20	13.02	1048574	20.000

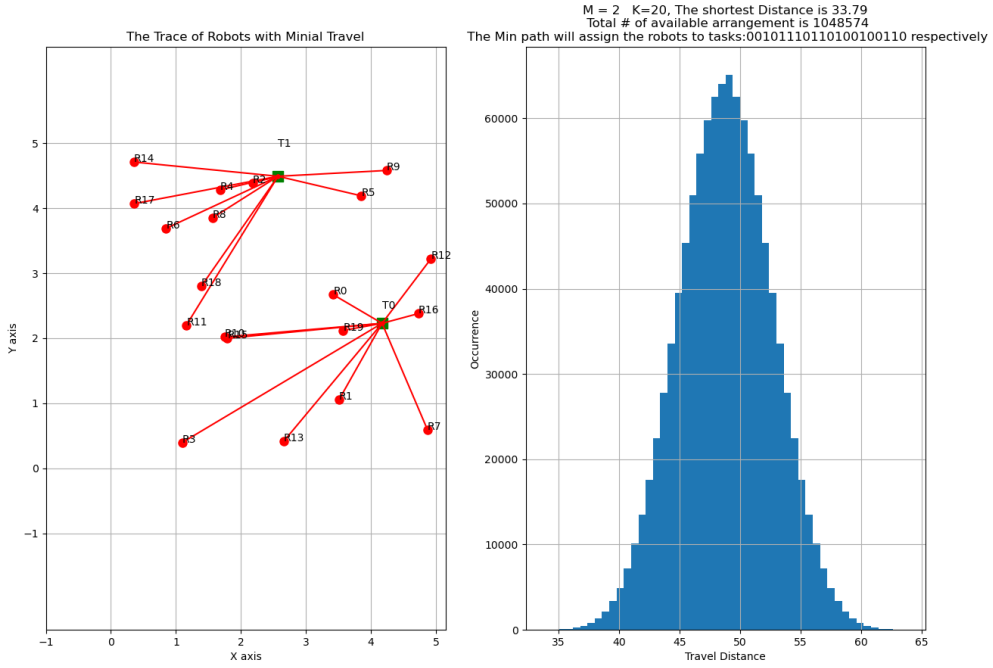

FIGURE 1.2: The maximum scale size of this problem that can be solved on a PC is 2 tasks and 20 robots

Chapter 2

Overview of the Previous Work

2.1 The history of Self Organizing Map (SOM)

Self-Organizing Map (SOM), also know as Kohonen map, is a topological preserving map that can map a higher dimensional space to a lower dimensional space. Along this process, information will be compressed; while, the key parameters in terms of "topological and metric relationships"[7] will be retained.

There are two steps involved in forming a self-organizing map from a raw input data-set[8], respectively to be 1) **competition** and 2) **cooperation**. When a set of data is feed into the system sequentially with random shuffle, for each input data point, **competition** will take place first and, based on a pre-defined cost function, one of the neurons on the output layer with the minimal cost will be selected as a winner; Following the competition, the **cooperation** will then take place. Based on a neighbourhood function, the winner together with it's neighbour neurons will proceed the learning; while, the neurons outside of the winner's neighbour zone will gain no learning. The purpose of the cooperation step is to increase the like-hood that if a similar input pattern present again, the same group of neurons will become the winner with a higher possibility. Iterate with this strategy on the input data-set over a suitable period, without supervising (providing error to the system), the output layer will simultaneously form a map that contains the similar topological structure as the input data.

2.2 Solving the task scheduling problem with SOM

2.2.1 Task scheduling with K output neurons

In 2006, based on the SOM approach, Anmin and Simon proposed a new method to solve the multi-robot system problems in a multi-agent architecture[1]. In this research, robots are scheduled to approach the targets by competition and cooperation.
The robots and targets are assumed to be randomly initialized in a bounded area. Conventional method are limited with the restriction that all the targets are static. However, in this algorithm proposed by Anmin, the path planning and robots' movements start synchronously. The left panel in Figure 2.1 shows the model of this architecture: there are 2 neurons on the input layer and K neurons on the output layer that are representing the coordinate locations of the target and the coordinate locations of the K robots, respectively. The K connecting weight vectors, $\{w_j x, w_j y\}$, where j = 1, 2, ..., K, are computed from the Euclid distance between the input task and the location of each of the robots. In each epoch, the robot with the lowest distance is selected as the winner and the weight vectors within the winner's neighbourhood zone will be updated to approaching the input target. All the targets will be input to the model in a random sequence in each iteration.

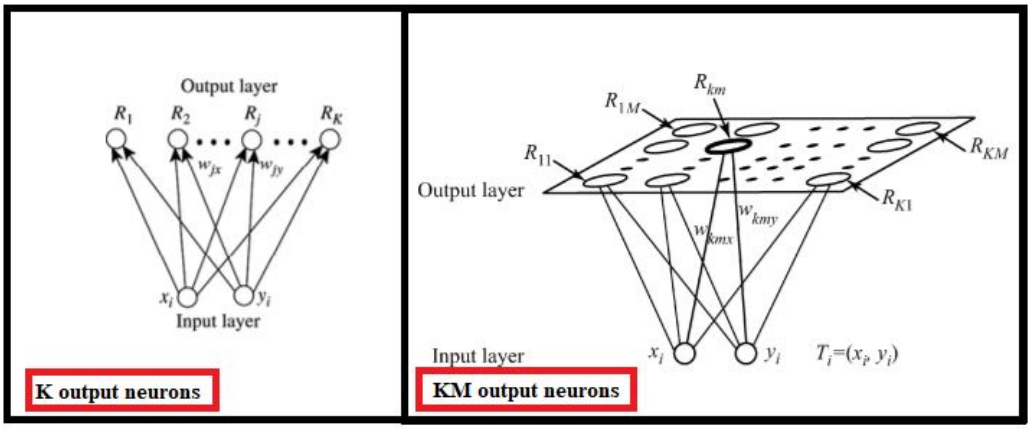

FIGURE 2.1: The illustration of two SOM configuration[1, 4].

Another interesting idea in this paper is the path planning for multi-robot systems. Several depots randomly distribute in the area. The task for robots is to get the goods from depots and reach the destination in the end. Thus, K M nodes are used in this

task. Each robot has a group of M neuron to do the planning of K paths for robots. Here, each neurons used a term P to control the workload for each robot. The distance computation is multiplied by the term P, which is regards as:

$$P = \frac{L + V}{1 + V} \tag{2.1}$$

Where, L represent the path of the robot and V is the average path length.

2.2.2 Task Scheduling with KM output neurons

As illustrating on the right panel of Figure 2.1, Anmin had also proposed a different method to solve this problem. In the second paper[4], the K M neurons are used in the second layer of self-organizing map to do the path planning for dynamic task. For path planning problems, the M neurons are assigned to a group for K robots. Also, the P control is also used to balance the workload for K robots as the previous paper. By doing this, the dynamic trajectories can be recorded with workload balance. In this task, the targets move a step in a random direction in each iteration. And the robot can still get to the target successfully. Although the path is not a near-optimal solution for the task, the task can be completed in computational time.

2.3 Review of previous related paper

2.3.1 A SOM-based multi-agent architecture for multirobot systems[1]

2.3.1.1 Abstract

The approach is capable of handling the "group of mobile robots to complete multiple tasks simultaneously."[1] Furthermore, this approach "uses cooperative and competitive behaviour"; the robots automatically arrange the complete task and dynamically adjust accordingly to the surrounding changes, unlike the conventional methods in static environments. Finally, as an implementation, "it can control a group of mobile robots such that the desired number of robots will arrive at every target location from arbitrary initial locations."[1]

2.3.1.2 Introduction

Multi robots system has a research work since 1970. there are various use cases for multi robots such as exploration, soccer, multitarget observation, path planning, box pushing, and forging[1]. However, it does come with the challenges such as cooperation and coordination. To solve those obstacles, a multi-agent system is the most reliable way to choose"[1]. Moreover, a multi-agent system is a subfield of AI (Artificial Intelligence).

In this paper[1], "a simple SOM-based multi-agent architecture is proposed for multi-robot systems." The multi-robot group can arrange the tasks automatically and dynamically adjust to the changing circumstances.

2.3.1.3 Method

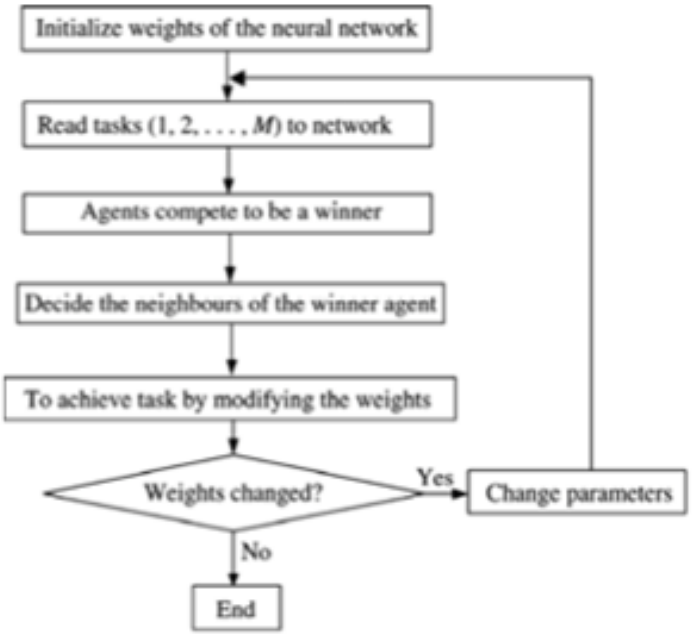

FIGURE 2.2: Block diagram for the approach[1]

The use of SOM, which helps capture the characteristics of the input vector, is capable of binding the outputs of similar inputs close to each other. In this paper, all the multi-agent robot is considered an agent. Thus, the input node is the task, and the output nodes are the robots[1].

Firstly, the SOM-based multi-agent architecture has competitive behaviour where the output nodes(robots) compete to win. The robot is kept aside after winning using an

inhibitory index for each node so that other robots get the chance to win for further competition. The node with the minimum distance towards the input node is considered the winner[1].

Secondly, after the agent becomes a winner in the case of cooperation, the system has to decide the neighbour agents of the winner agent. Then, the neighbourhood function determines the attraction strength of the task data on the winner agent[1].

Finally,[1] the winner and the neighbour agent perform the task by modifying their connection vector, and the other agents remain unchanged. Modification of the input vector depends on the original relationship between the winner agent with its neighbouring agent and input tasks but also the neighbourhood function along with the network learning rate.

2.3.1.4 Conclusion

A single robot that cannot complete multiple tasks became possible using simple SOM Multi-agent architecture for multi-robot systems and the cooperation and competition process[1]. Furthermore, the combination of motion planning and target assignment enabled the robot to move before its final destination, which helps in dealing with the surrounding changes or in cases where a robot does not respond. Adding to this, we can select the number of robots assigned to a particular target, even for the moving target [1].

2.3.2 A Multi-agent Architecture Based Cooperation and Intelligent Decision Making Method for Multirobot [2]

2.3.2.1 Abstract

For dealing with changing environments and controlling multi-robot systems, Yang proposed a multi-agent architecture[6]. "Two associated issues about the architecture are cooperation between robots and intelligent decision making."[2] The parameters responsible for solving and illustrating the role assignment are reward function, ability vector, and cost function. In addition, RBF Neural Networks are used for the decision-making

of the multi-robot system. The results prove that the method can improve the accuracy of the whole decision system[2].

2.3.2.2 Introduction

One of the advantages of multi-robot systems is that the tasks can be divided into many sections so that they can be completed concurrently by a team of robots[6]. Moreover, producing and operating numerous small robots can be smoother, less expensive, more versatile, and more resilient towards fault-tolerant than employing a single powerful robot for each different task[2]. On the other hand, cooperation and coordination are the most difficult tasks in a multi-robot system[6]. "The multi-agent system (MAS) is an emerging subfield of artificial intelligence (AI) and is one of the two sub-disciplines of distributed artificial intelligence (DAI)[2]."

Hierarchical structures and Behavioural structures are the two types proposed by many kinds of researchers when it comes to MAS[2]. Firstly, "In a hierarchical structure, information flows from sensors to high-level decision units in a fixed way; then the decision units send commands to low-level actuator units[2]." Secondly, "In a behavioural structure, the control problem is broken into behaviours without any central intelligent agent present[6]." However, it is not easy to achieve high-level decisions, so to deal with this problem, using a hybrid system with a combination of behavioural structure and hierarchical structure is encouraged[2].

2.3.2.3 Method

The proposed architecture is a heterogeneous approach that consists of several agents[6]. In addition, the agent used in this section is the "master agent and real robot, which is the combination of reasoning agent and actuator agent[2]." The master agent "consists of strategies knowledge in the global knowledge database, static knowledge and rational rules Conclusion" and the reasoning agent "consists of a dynamic knowledge database, reasoning and path planning[2]. " The actuator agent "refers to mechanical and electrical devices of a robot[6]." However, it is not the same as the multi-agent logic structure[2]. It has an "intension framework of hierarchical reasoning and shared plan" rather than the behavioural and hierarchical structure[2].

FIGURE 2.3: A hybrid MAS architecture for Multirobot system[2]

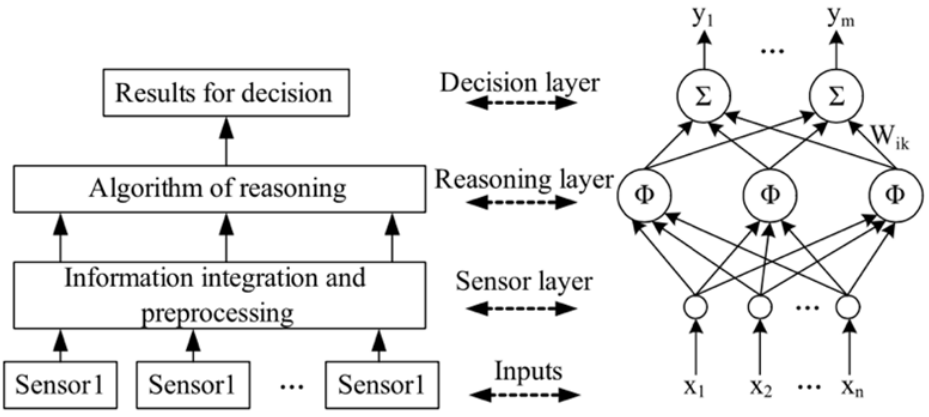

FIGURE 2.4: A layered structure of information fusion and a feedforward NN[2]

2.3.2.4 Conclusion

The architecture comprises an actuator agent, a robot consisting of a reasoning agent and a master agent[2]. Features from the paper are (1) simplification of the reasoning process; (2) real-time reactive ability;(3) robustness and fault tolerance; (4) flexibility[2]. In addition, the ability vector provides abilities required for describing and accomplishing the tasks[6]. RBFNN was robust in enhancing the decision system as contrasted to the standard methods[2].

2.3.3 A fuzzy-logic based chaos GA for cooperative foraging of multi-robots in unknown environments [3]

2.3.3.1 Abstract

In this paper[3], the proposed method was to look into the searching of multi robots in an unknown environment using the robots' sensory information and expert knowledge of finding the target, "foraging problem in this paper is defined as a searching task, where the robots cooperate to find and reach all the targets" efficiently[3]. In order to do so, Ni offers a Fuzzy-logic based Chaos Genetic Algorithm(FCGA)[3] . Moreover, the robots are capable of finding every target, even in any problematic situation, by adjusting themself dynamically. Furthermore, experiments show that the proposed method (FCGA) is more efficient than the Pure Chaos Genetic Algorithm (PCGA) and random-search[3].

2.3.3.2 Introduction

Task allocation is proven to be an NP-complete problem when it comes to task allocation in a multi-robot system[3]. There are three main pointers to look into, starting from hunting, foraging and scheduling[3]. In this paper, a near-to real-world problem is countered as the target is unknown[3]. The robots use their sensors such as the smell of the odorous source, infrared radiation from the origin of heat, the dispersion of the radioactive source[3].Moreover, Ni has an utterly different approach [3] from the "general path planning in multi-robots, where the locations of targets are often known, and there is no cooperation among the robots." Instead, better trajectory and barrier avoidance are automatically achieved in a changing environment by the robots[3]. In addition, this

is possible because of the self-adaptive parameters and the presence of a small number of designer parameters that have to set [3].

2.3.3.3 Method

As the goal is to find the target more optimally, the proposed method uses genetic algorithms (GA)[3]. Furthermore, using fuzzy logic, the directions are given for the movement of the robots[3].

The robots will move randomly if there is no knowledge about the target or if the "information density is the same around the robots[3]." However, using memory and knowledge along with the robots' sensory data could help find a better route. Therefore, fuzzy logic is adopted to "realize the functions of such knowledge[3]."

Part of Fuzzy logic is:

- Dispersion degree: This fuzzy concept checks whether the robots go collectively too much.
- Homodromous degree: This fuzzy concept analyses whether all the robots run towards the same direction.
- District-difference degree: This fuzzy concept investigates whether every robot stays in the same region.

2.3.3.4 Conclusion

The proposed method is FCGA; it deals with various problems independently and is capable of finding all the targets[3]. If the robots are grouped closely or have the exact origin, the proposed approach can scatter them in separate areas[3]. The proposed approach applies in task allocation dynamically in multiple anonymous situations "such as the fire disaster response and the radioactive object searching[3]."

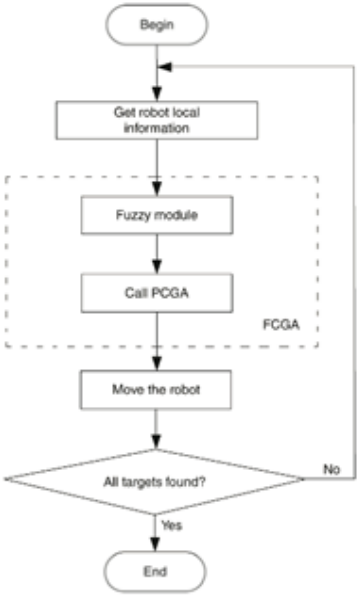

FIGURE 2.5: Block diagram for the approach[3]

Chapter 3

The Proposed Approach to the Problem

In order to achieve the goal of investigating the dynamic robot task scheduling problem with self organizing map with KM neurons on the output layers, there are 4 milestones that we want to achieve.:

- The first milestone is we are going to build a simple python self organizing map module with Google Tensorflow. This module should take a set of RGB colors as input, and then we are expecting this module to be able to classify the input RGB colors by mapping and labeling them on the output layer as a 2D grid of RGB colors;
- The second milestone is we are going to extend the python self organizing map module in milestone-1 and make this module generic. The extended module should take a set of MNIST handwritten digit images as input and be able to classify any user input handwriting digit after training. We are expecting that our self organizing map module to have a robust API that can take more general object (in this case, a set of handwriting images, which is a set of a set of RGB colors) as input and produce more general output (in this case, a 2D grid of a 2D grid of RGB colors).
- The third milestone is to re-produce a task scheduling paper published by Anmin[1]. With the python code and knowledge we have obtained from milestone-1 and

milestone-2, our third milestone is to extend our python self organizing map module so that it can take a set of (x, y) coordinates as input. We are expecting this module to be able to map the coordinates of M task targets to K robots with optimal total robots' traveling path. The challenge/difference are coming from two aspects:

1) In milestone-1 and milestone-2, we have a grid of output neurons; however, we now only have a series of K output neurons on the output layer instead of a grid (see the left panel of Figure 2.1);

2) For milestone-1 and milestone-2, the connecting weights to the output neurons can be initialized as anything; but here, in milestone-3, we have to use the (x, y) coordinates of the robots' initial position as the initial connecting weights.

3.1 SOM Color Map

The structure of a SOM neuron network generally consists only two layers of neurons. The input neurons and the output neurons. These two layers are fully, and directly connected without any hidden layers in between [9]. Here, we are going to illustrate our SOM color map module with a detailed example of mapping 5 input colors: 1) black, 2) dark gray, 3) medium gray, 4) light gray, 5) white on to a 4 rows by 5 columns output grid.

The neurons in the input layers represent the different attributes of the input data set. As shown in Figure 3.1 In this SOM color map example, there are 3 input neurons, they are R, G, B components of each of the input color, respectively; while, in the robot task scheduling example, there will be 2 input neurons, one for the x component and the other one for the y component of each of the task target's location.

The neurons on the output layers (the so-called Kohonen layer) is a reduced dimensional representation of the input data. The output layer can be either one dimensional string[10] or a two dimensional grid[11]. In this SOM color map example, the output layer is a 4 by 5 grid. We have also noticed that the output layer of one of the models Anmin proposed to solve the robot path planing problem is a one-dimensional string[1]; while, the other model's output layer is a K by M two-dimensional grid[4]. However, there is a general belief that 2D output layer can preserve more details of the input data

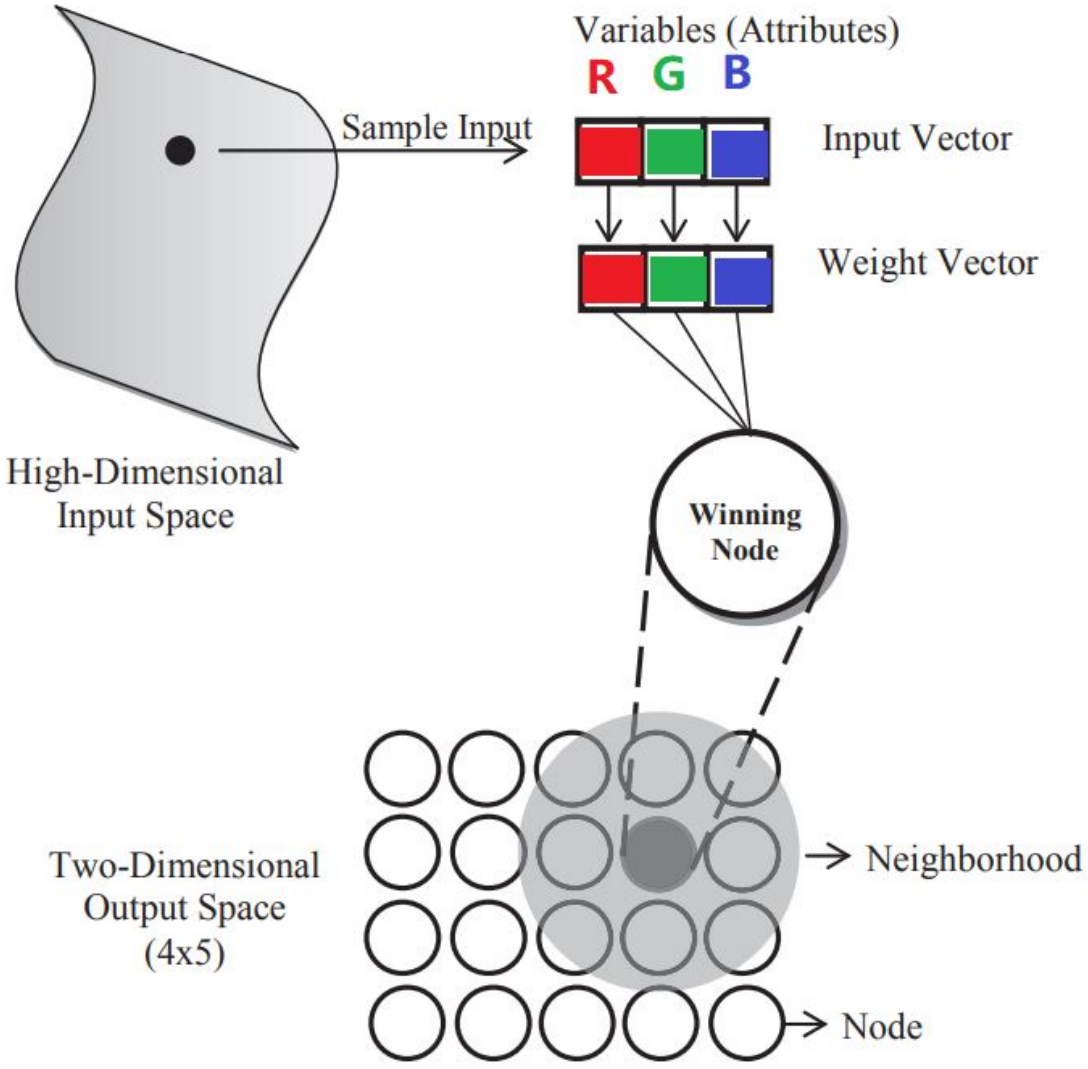

FIGURE 3.1: The illustration of a SOM that map a set of RGB colors onto a 2D Grid. Figure is modified from Asan[5]

set. We will compare the difference of Anmin's two robot path planing models in this project as well.

Back to our Color Map example, Table 3.1 is showing the set of color as input data. Each R, G, B components are normalized to the range of [0,1] from the conventional [0, 255] range. There are totally 5 input samples, and we are going to use a 4 by 5 grid for the output layer for demonstration purpose.

First, we will assign the initial value for the weight vectors for each of 20 neurons on the output layers. The initial value should be assigned as random values, however, some

TABLE 3.1: The SOM color map's input data set of 5 different colors

	Attributes		
Color	**R**	**G**	**B**
Black	0	0	0
Dark Gray	0.33	0.33	0.33
Medium Gray	0.5	0.5	0.5
Light Gray	0.66	0.66	0.66
White	1	1	1

TABLE 3.2: The SOM color map's initial connecting weights

Attributes	w1	w2	...	w20
R	0.5	0.5	...	0.5
G	0.5	0.5	...	0.5
B	0.5	0.5	...	0.5

researchers found that it would be more efficient to unitizing the input data set's largest eigenvalue to assign the initial weight vectors. For demonstration purpose, as shown in Table 3.2, we are going to simply assign all the weight vectors equal to 0.5 (medium gray), and we will see the even with the uniform non-unique weight vectors, the output layer start to show clear color separation right from epoch one. Suppose the 1st input is the black color, then we can find the Euclidean distances for all the 20 output nodes to the 1st input color (black) as the following:

$$d_1(0) = d_2(0) = d_{20}(0) = \sqrt{(0-0.5)^2 + (0-0.5)^2 + (0-0.5)^2} = 0.866$$

The winner neuron is the 1st neuron as all the Euclidean distances are equal, then we are going to update the connecting weight based on the winner. The connecting weight update function can be written as the following:

$$w(t+1) = w(t) + \alpha(t)\, h(t)\, [x(t) - w(t)] \tag{3.1}$$

Where $w(t)$ is the connecting weight at iteration t, α is the learning rate, h is the neighborhood function, and x is the input attributes (either R, G, or B). For demonstration purpose, we used the linear learning rate and Gaussian function as the neighborhood function in this example:

$$\alpha = 0.3\left(1 - \frac{\text{Current Iteration}}{\text{Max Iteration}}\right)$$

$$h = \exp\left(-\frac{d^2}{\sigma^2}\right)$$

where σ is the initial neighborhood range function, it is defined as:

$$\sigma = \frac{\text{Max [Output layer dimensions]}}{2} = 2\left(1 - \frac{\text{Current Iteration}}{\text{Max Iteration}}\right)$$

The updated connection weight for the winner node can be calculated as shown below:

$$\begin{bmatrix} w_R(1) \\ w_G(1) \\ w_B(1) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} + 0.3\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \begin{bmatrix} 0.35 \\ 0.35 \\ 0.35 \end{bmatrix}$$

We can also calculate the winner's direct neighbor's connecting weight as the following:

$$\begin{bmatrix} w_{neighborR}(1) \\ w_{neighborG}(1) \\ w_{neighborB}(1) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} + 0.3\left(\exp\left(-\frac{0^2}{2^2}\right)\right)\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \begin{bmatrix} 0.35 \\ 0.35 \\ 0.35 \end{bmatrix}$$

Following this method for each of these 5 input colors for each epoch, and after 1500 epochs, we can see a clear color separation on the output layer (See Figure 3.2).

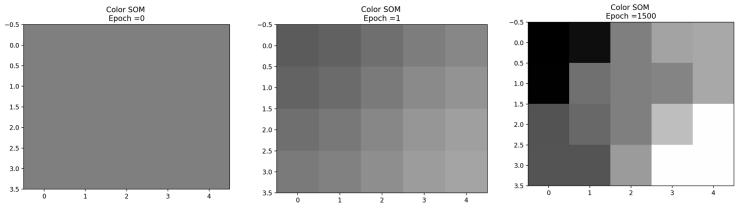

FIGURE 3.2: SOM color map separates black color, gray color and white color on a 4X5 grid. Left panel is at epoch 0, where all the grids are initialized as gray color; middle panel is at epoch 1, where we start see color separation; the right panel is the output layer after 1500 epoch.

3.2 SOM MNIST Handwritten Digit Recognition

Self-organizing feature map (SOFM)[12] was proposed by Kohonen in his paper in 1982. Besides from the SOM algorithm by the same author, the output neuron layer is illustrated by a 2D feature map which explains the topologically information in the training

process. For more research doing on SOFM, Ultsch[13] did a work of visualize the features learnt by the SOFM neuron network model using a U-matrix. In this paper, the U-matrix can store the topological relationship between the nearby output neurons. The learnt weights are able to form a lattice which show the distribution of the output neurons. From the map, people can easily see each part of the clusters. In this project, we would like to implement SOM algorithm and introduce several SOFM applications and show the visualizations of each output neurons.

3.3 SOM Robot task assignment with K output neurons

In the year of 2016, Anmin proposed a SOM-based neuron network that can solve the task assignment problem for a system with K robots and M targets. This SOM based algorithm can solve a very fundamental problem: there are K robots and M targets randomly distributed in a 2D plane, how to assign the robots to the tasks so that the total traveling path for these robots is minimum or near minimum. As shown in Fig.3.3, this neuron network is consistent with 2 layers of neurons, the input layer has 2 neurons representing the x and y coordinates of the location of targets, while the SOM layer consists of a sequence of K neurons representing each of the K robots. The input layer and the SOM layer are fully connected, the connection weights are initialized as the x, y coordinates of each of the robots respectively.

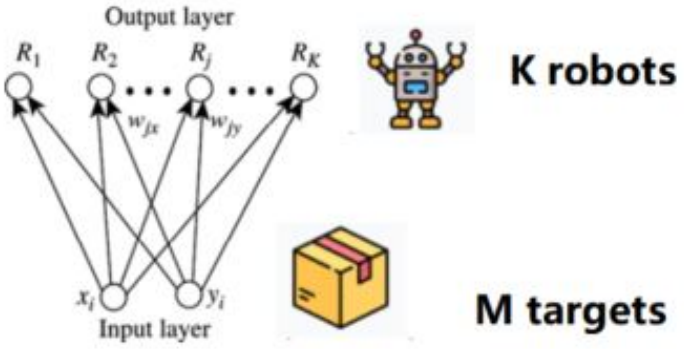

FIGURE 3.3: The Structure of SOM for robot task assignment, the coordinate of M targets are the inputs, and there are K neurons on the SOM output layer representing the real time position of the robots. This figure is modified from Anmin's paper[1]

The training mechanism of this SOM can be described as the following, after the connection weights are initialized, during each epoch, the list of M targets are shuffled and

then feed into the SOM system as input, and for each input, T_i, based on the Euclidean distance between the input and each of the neurons, R_j, a winner with a minimum distance, D_{ij}, is selected for each of the input targets.

$$D_{ij} = |T_i - R_j| = \sqrt{(x_i - w_{jx})^2 + (y_i - w_{jy})^2} \tag{3.2}$$

There is a constraint that none of the neurons can win twice during one epoch so that each of the neurons will have a similar probability to become a winner during a specific epoch. After a winner neuron is selected, the winner will adjust its connecting weight to move toward the associated task together with its neighbor neurons.

$$R_j(t+1) = R_j(t) + \underbrace{\beta}_{\text{learningRate}} \times \underbrace{f(d_j)}_{\text{NeighbourhoodFunc.}} \times \underbrace{D_{ij}(t)}_{\text{EuclideanDist.}} \tag{3.3}$$

Equation 3.3 shows how far the winner and its neighbor neurons (R_j) will move is adjusted by a learning rate, β, and a neighborhood function, $f(d_j)$. The learning rate is a constant over the entire training process. The effective range of the neighborhood function is determined by a cut-off radius, r. The area outside of the radius r circle is not considered as the neighborhood of the winner, while the neurons inside of the radius r circle are the winner's neighbors.

$$f(d_j) = \text{Exp}\left(\frac{-d_j^2}{(1-\alpha)^t G_0}\right) \times \delta_{d_j<r} \tag{3.4}$$

As shown in equation 3.4, how far the neighbors move is adjusted by a Gaussian function of the distance, d_j, between each of the neighbor neurons to the winner. α and G_0 are change rate constant that determine the speed of computation. This Gaussian function is an attenuated function over epoch number, t, which means the width of the Gaussian neighborhood function is getting more and more narrow as epoch increases. The training process stops when no weight vectors change any more.

It is obvious to see that the inputs are in the 2D space (x and y) and after the training, the topological information from the input space is retained and dimensionally reduced to 1D on the SOM map of K neurons. The evolution of the connecting weights for each neuron is the objective of study. They reflect the trajectory path of each of the robots.

At the end of the training, the total path is minimal or near minimal because the winner is determined by the Euclidean distance between input tasks to the robots in each epoch.

As a construct, in the color map case, the inputs are in the 3D space (R,G, and B), and after training, the topological information of the input colors will be displayed on a 2D SOM layer with KM neurons. The fundamental difference between the robot path assignment method and the color map method is how the neighbors are determined in the cooperation process. For the robot path assigning method, neighbors of the winner are determined by the Euclidean distance between the connecting weights of the neurons (the neurons with similar connecting weights are considered as neighbors); while for the color map method, the neighbors of the winner are determined by the location of each of the neurons on the SOM map (the neurons that are physically close to each other on the SOM map are considered as neighbors).

Chapter 4

Results

4.1 Result of SOM Color Map

Using the method introduced in Chapter 3, we have implemented a python tensor flow module that can separate different colors on the output layer. Now we can do something more interesting with the SOM module. In Figure 4.1, we are presenting that our module is able to separate and label 16 different colors and correctly label them on a 50 by 50 2D grid of output layer. It is also interesting to notice that as the number of epoch increase,

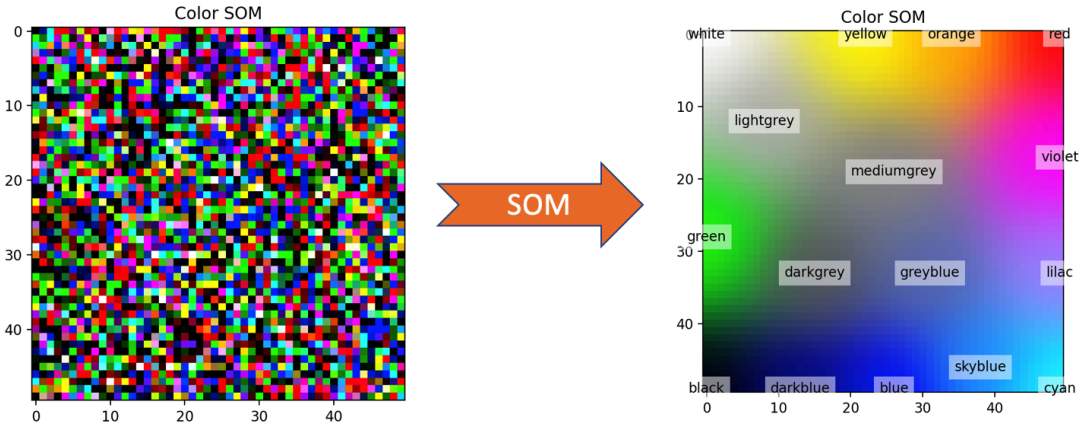

FIGURE 4.1: Separation and labeling of 16 colors with our SOM Module. The left panel is a random initialized 2D output layer, and the left panel is the output layer after 100 times iterations.

the separation is more and more clear. As we can see from the Figure 4.2, comparing the picture with epoch equals to 1200 with the picture with epoch equals to 12, one can see

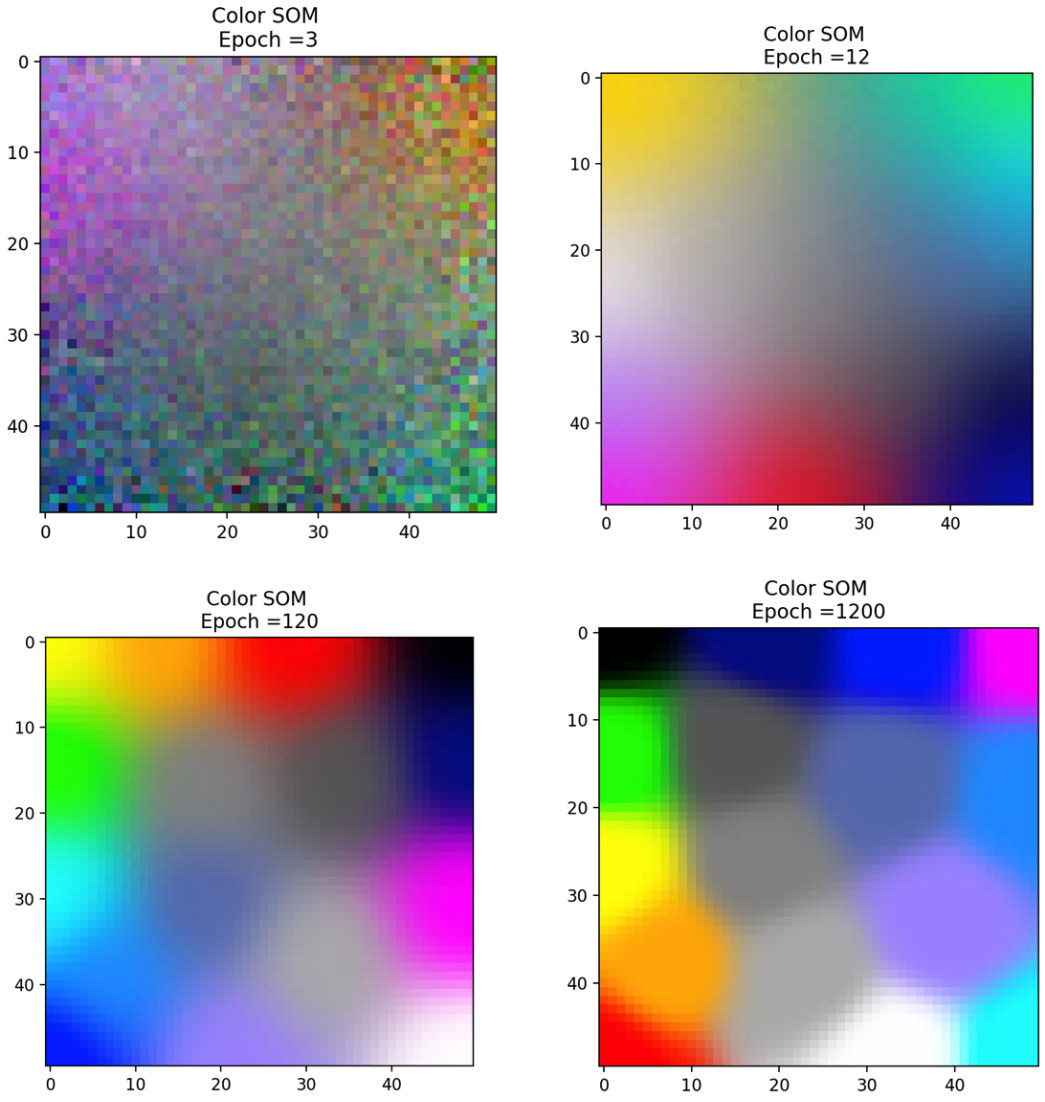

FIGURE 4.2: The snapshot of the color separation process of our SOM module, one can see from the pictures that as the epoch number increase, the separation between different colors becomes more and more obvious

the boundary between different colors are more solid and clear as epoch number increase. It would be nice if we can quantifies the color separation effects. In history, there are a lot of different approaches to evaluate the output layer's topological quality[14], however, in this project, we will use the easiest format: Topographic Error[15]:

$$te \equiv 1 - ta = \frac{1}{M} \sum_{i=1}^{M} err\left(\vec{x}_i\right)$$

Where te is the topographic error, M is the size of the input data set, and ta is the topographic accuracy of the output layer. $err\left(\vec{x}_i\right)$ is the error for each of the input data,

it equals to 1 if the first best matching unit is the same as the second-best matching unit, and equals to 0 otherwise.

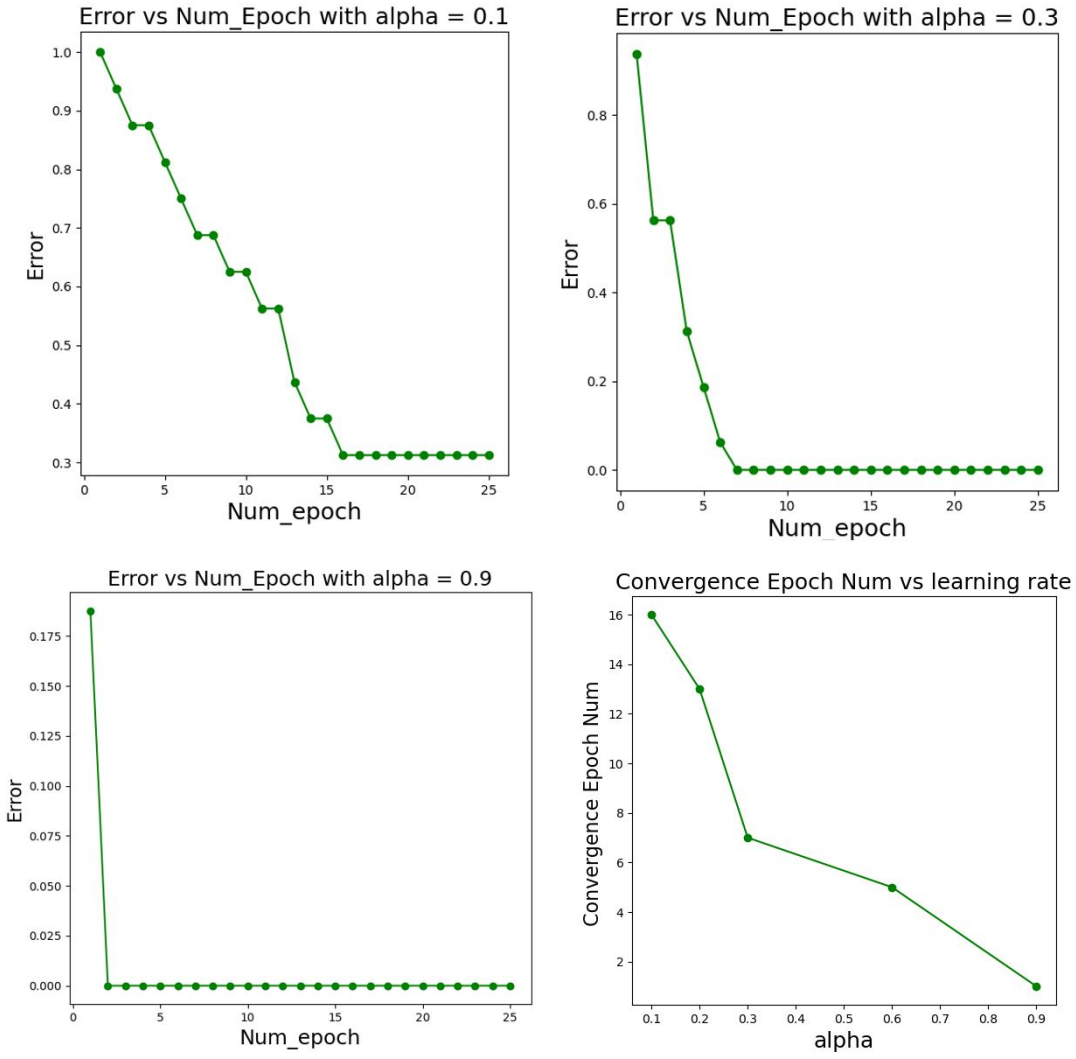

FIGURE 4.3: The comparison of convergence rate vs different learning rate, α (the right bottom panel). The convergence rate here is defined as at which epoch number the topographic error equals zero (as shown in the rest 3 panels).

Let the learning rate $\alpha = 0.1$, 0.2, 0.3, 0.6 and 0.9, the initial neighborhood function $\sigma = 3$, and let's put 15 by 15 neurons on the self organizing map, and select 16 different colors as input (as shown in Table4.1), we can investigate how the convergence of the SOM (at which epoch the topographic error equals zero) is affected by the learning rate α. As shown in Fig.4.3, the topographic error approached zero at epoch 16 when learning rate $\alpha = 0.1$, while it approached zero at epoch 1 when learning rate $\alpha = 0.9$. If we plot the learning rate against the epoch number during which the typographic error

approached zero, one will find that the greater the learning rate α is, the more rapidly the SOM coverage. we almost got a linear relationship between these two terms (as can be seen from the right bottom panel of Fig.4.3).

Another interesting result we got is that we found that the topographic error, even though the definition of it is very simple and easy to understand, but, it might not be the best choice as a error function for image processing. Let the learning rate $\alpha = 0.3$, the initial neighborhood function $\sigma = 3$, and let's put 15 by 15 neurons on the self organizing map, and select 16 different colors as input (as shown in Table4.1). We can plot topographic error over the number of epoch to evaluate the convergence rate of our module. As can be seen from Fig.4.4, even though the topographic error converged to zero at epoch 7, but people can still see that the boundary of these 16 colors are not well separated even at epoch 25. This result indicates that topographic error might not be a good indicator for the color SOM; therefore, more advanced error function should be employed to reflect human eyes' response to color change. Actually, readers can see in the next subsection (SOM MNIST Handwritten Digit Recognition) that the sum of the difference between input and output connecting weights is a much better indicator than topographic error as error function for image processing.

All the code regarding to this sub-section can be found at this Github Link or please go to the next url: https://github.com/ENGG6570/Group-Project/tree/main/PythonColorMap

4.2 Result of SOM MNIST Handwritten Digit Recognition

In the experiments, we implement several applications using SOFM and visualized the feature map. MNIST database is a handwriting data of numbers from 0 to 9, which has 60,000 gray level images of handwritten numbers.

In this project, we use SOM algorithm to train these images as inputs, which are fit in the model with randomly selected sequences. Then, the model updates the weights in each iteration. Figure 4.5 is the Snapshot of the output layer after 100 times iteration.

We also used several update methods for learning rate.

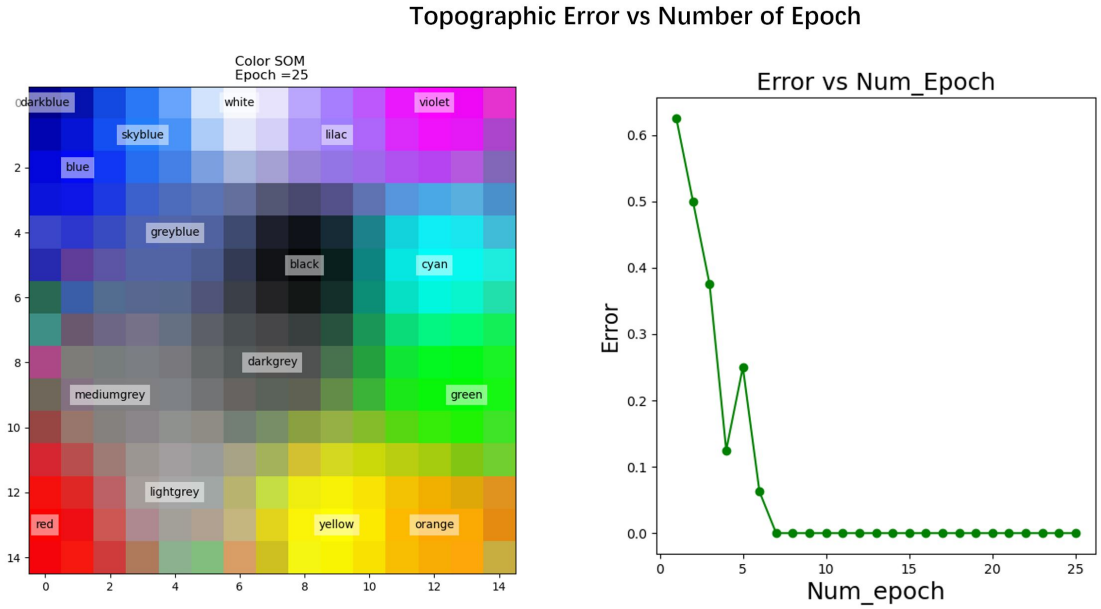

FIGURE 4.4: The left panel is a color SOM at epoch 25 with learning rate $\alpha = 0.5$, initial neighborhood function $\sigma = 3$, and 16 colors as input; The right panel is the change of the Topographic Error of this color SOM over the first 25 epochs

TABLE 4.1: The SOM color map's input data set of more different colors

	Attributes		
Color	**R**	**G**	**B**
Black	0	0	0
Dark Gray	0.33	0.33	0.33
Medium Gray	0.5	0.5	0.5
Light Gray	0.66	0.66	0.66
White	1	1	1
Orange	1.0	0.647	0.0
Yellow	1	1	0
Violet	1	0	1
Cyan	0	1	1
Red	1	0	0
Green	0	1	0
Lilac	0.6	0.5	1.0
Blue	0	0	1
Grey Blue	0.33	0.4	0.67
Sky Blue	0.125	0.529	1.0
Dark Blue	0	0	0.5

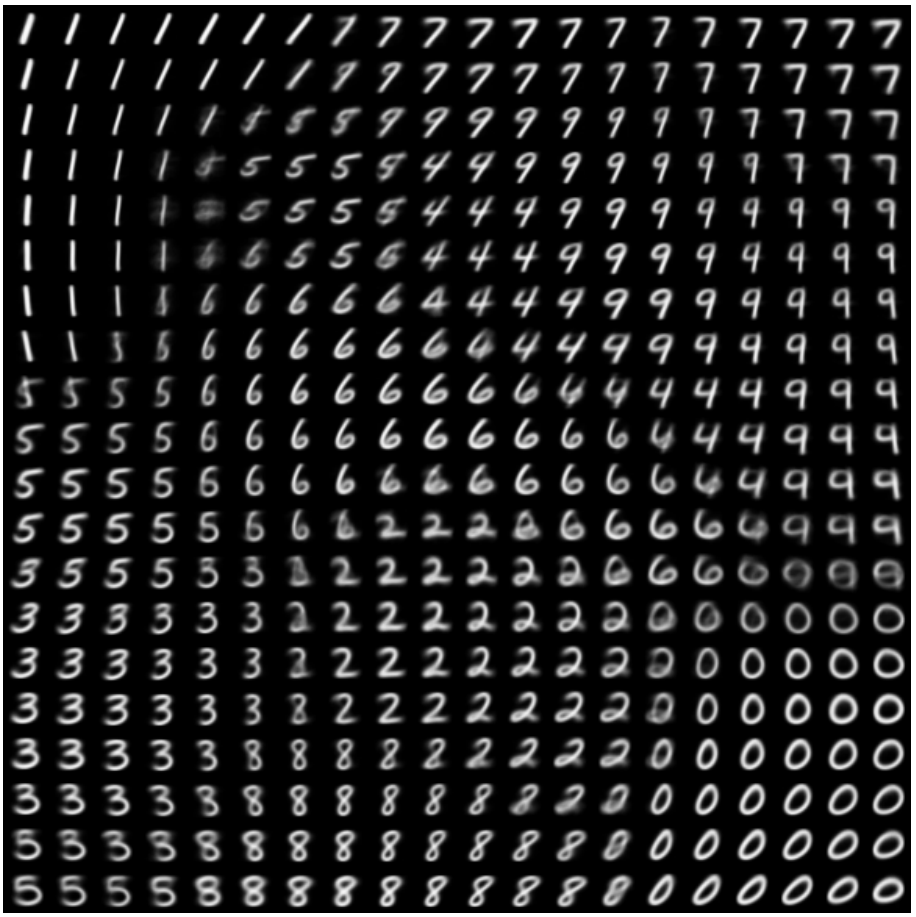

FIGURE 4.5: The feature map of MNIST database after 100 iteration.

1. The first one is fixed learning rate, we choose 0.3 here for our model.

$$\alpha = 0.3$$

2. The second one is a linear relationship with the training iteration, which is regards as:

$$\alpha = 1 - \frac{\text{Current Iteration}}{\text{Max Iteration}}$$

3. Another one people used a common choice of exponential way to do the decay of learning rate, the equation can be written as:

$$\alpha = \alpha \times e^{-t/\tau}$$

where t is the current iteration number and τ is the total iteration number.

The results of these three different methods is showed as in Figure 4.6.

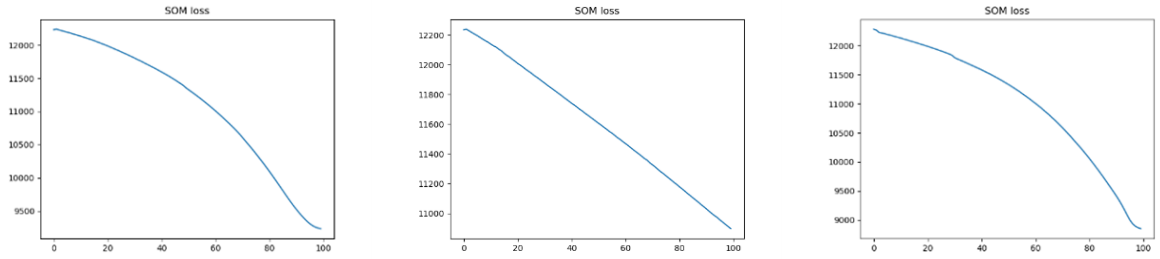

FIGURE 4.6: (a) The SOM loss with constant learning rate. (b) The loss with linear rate. (c) The SOM loss using exponential changing learning rate.

In these experiments, the SOM loss function is calculated as:

$$loss = \sqrt{\sum_{k=0}^{n} \left((W_k - X_k)^2 \right)}$$

where the W_k and X_k denote the kth weight of the input and output neuron. From the figure 4.6, we can see that the exponentially changes learning rates has a better learning speed, which decreases 8551.37 after 100 iteration.

4.3 Result of SOM Robot task assignment with K output neurons

Using the method introduced in Chapter 3, we can compare the task assignment solution provided by the SOM approach with the solution provided by the brute force approach as introduced in Chapter 1. For simplicity, we will focus on the scenarios that the number of targets equals the number of robots. Two arrangements are evaluated in this project, 1) 3 robots and 3 tasks, and 2) 7 tasks and 7 robots.

4.3.1 3 tasks assigned to 3 robots

Suppose on a 8 unit by 8 unit plane, 3 tasks, T0, T1, and T2, and 3 robots, R0, R1, and R2 are randomly distributed. As shown in Fig.4.7, the tasks are shown in green squares and the robots are shown in red solid dots. There are totally 6 possible arrangements as shown in Table 4.2. The brute force solution suggests that Arrangement6, which is to assign task T2, T1, T0 to robots R0, R1, and R2, respectively (the row in red color),

TABLE 4.2: All the possible arrangements to assign 3 tasks to 3 robots on a 2D plane

Possible Arrangement	R0	R1	R2
Arrangement1	T0	T1	T2
Arrangement2	T0	T2	T1
Arrangement3	T1	T0	T2
Arrangement4	T1	T2	T0
Arrangement5	T2	T0	T1
Arrangement6	T2	T1	T0

is the global best solution that provides the minimum over all robot-traveling-path for the specific configuration as shown in Fig.4.7. The left panel in Fig.4.7 shows that the total traveling distance equals to 6.52 unit, the right panel in Fig.4.7 shows that after training SOM Neuron Network (NN) also provides the almost identical traveling path which equals to 6.53 unit.

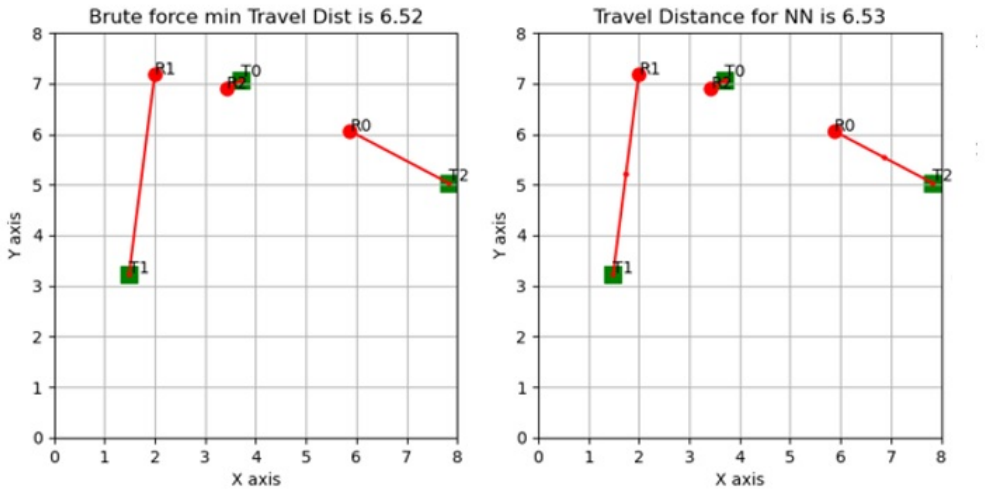

FIGURE 4.7: The global best task assignment solution provide by the brute force method (left panel)is almost identical with the SOM Neural Network (NN) solution (right panel), respectively to be 6.52 unit and 6.53 unit.

For this particular situation as shown in Fig.4.7. If one make a histogram for all the possible arrangements between these 3 robots and 3 tasks provided by the brute force method, with the x axis to be the total-travelling-length for all the 6 possible arrangements, and the y axis to be the how many times a certain total-travelling-length is occurring within in the arrangement-space (6 different arrangements in this particular case), a histogram can be obtained as shown in Fig.4.7. The 5 blue bars in this figure indicates that among all these 6 possible arrangements as shown in Table 4.2, one arrangement leads to the robots' total traveling path equals 6.52, one arrangement

leads to the robots' total traveling path equals 8.1, one arrangement leads to to the robots' total traveling path slightly above 11, one arrangement leads to the robots' total traveling path above 12.5, but there are two arrangements will lead to identical total traveling path. The read dash line in this histogram plot indicates that the SOM NN is indeed providing us the global best solution in this situation, which equals to 6.53 unit in length. The significant of this histogram is that this figure tells people that for this particular situation as shown in Fig.4.7, if one just randomly assign the tasks (without using any algorithms) among these robots, he will get a near uniformly random outcome for the robots-total-traveling-length. Sometimes he will get 6.5, sometimes he will get 8.1, 11.2, 11.5 or 11.6 (each of the blue bars in Fig.4.7), but he almost get equal probability to get each of the numbers between all these 5 numbers. If one repeatedly led the robots to conduct these tasks from the same initial locations, the expectation of the total-traveling-path, as shown in Eq.4.1, will be the average over these bars in the histogram, which is about 10.27 unit length. Comparing to the 6.53 unit length SOM NN solution, we conclude that in the small scale situation (3 tasks and 3 robots), the SOM solution is as good as the brute force solution, and is statistically more than one and a half times (10.27/6.53 = 1.53) better than randomly assign tasks to robots.

$$\langle \mathrm{Total-Travel-Distance} \rangle = \frac{6.5 + 8.1 + 11.2 + 11.5 + 11.5 + 12.6}{6} = 10.27 \qquad (4.1)$$

4.3.2 7 tasks assigned to 7 robots

We also applied the this method to a different situation which is slightly larger in scale (7 tasks and 7 robots), as shown in the Fig.4.9, in this situation, the brute force method yields that the global best solution has the total robots traveling length equals to 11.66 unit length (left panel), while, the SOM NN is generating an arrangement that the total robots traveling length is 14.58 unit length (right panel). The histogram of all the possible total travel distance vs. their occurrence is ploted in Fig.4.10. There are totally 5040 different possible arrangements, and the brute force solution told us that the best arrangement is to assign task 2, 5, 4, 6, 0, 3, 1 to robot 0 to robot 6, respectively. Note that comparing the left panel and the right panel in Fig.4.9, the SOM not only returns a different task assignment arrangement than the global best solution, but also

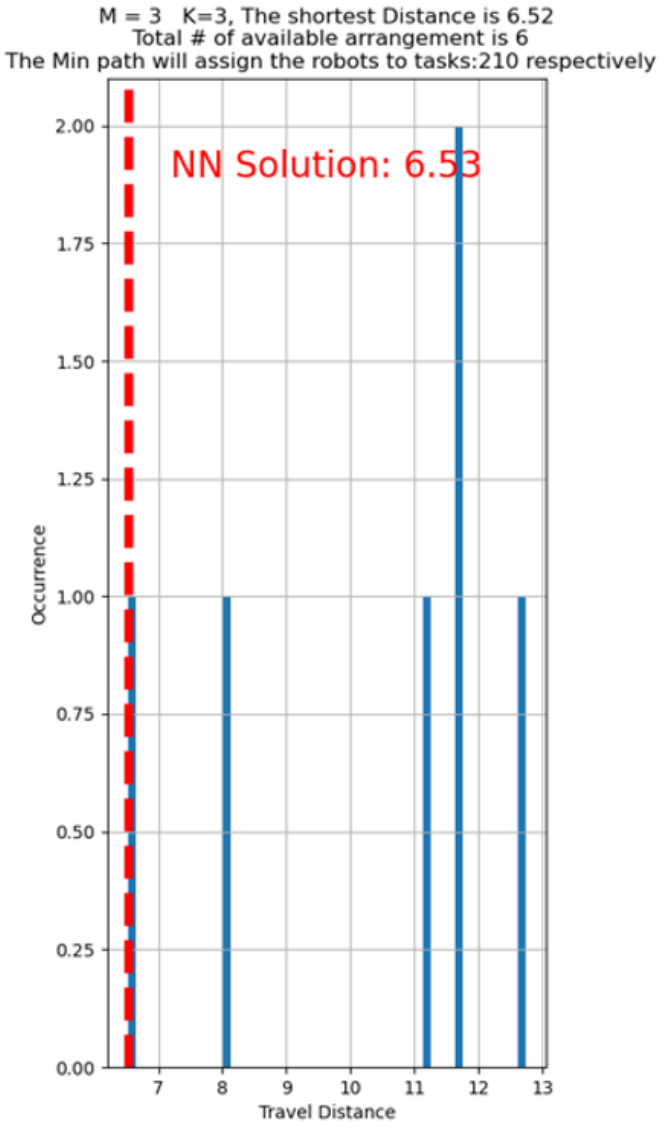

FIGURE 4.8: The histogram of all the possible total-traveling-distance on the x axis, and with their occurrence in the brute force arrangement space

the trajectories are not straight due to the competition and cooperation. However, as it can be seen from Fig.4.10, the red dash line (the SOM NN solution) lies on the left tail of the histogram, which indicates that the SOM NN, even though is not providing the best solution, is still providing a reasonable good solution.

Now, we need to establish a quantitative method to evaluate how good the SOM solution is. The first thing we noticed is the histogram in Fig.4.10 looks like a normal distribution. Gaussian distribution is a very strong condition as well as a very good property. If the arrangement's solution space is Gaussian, then we can use the K-Value method to have

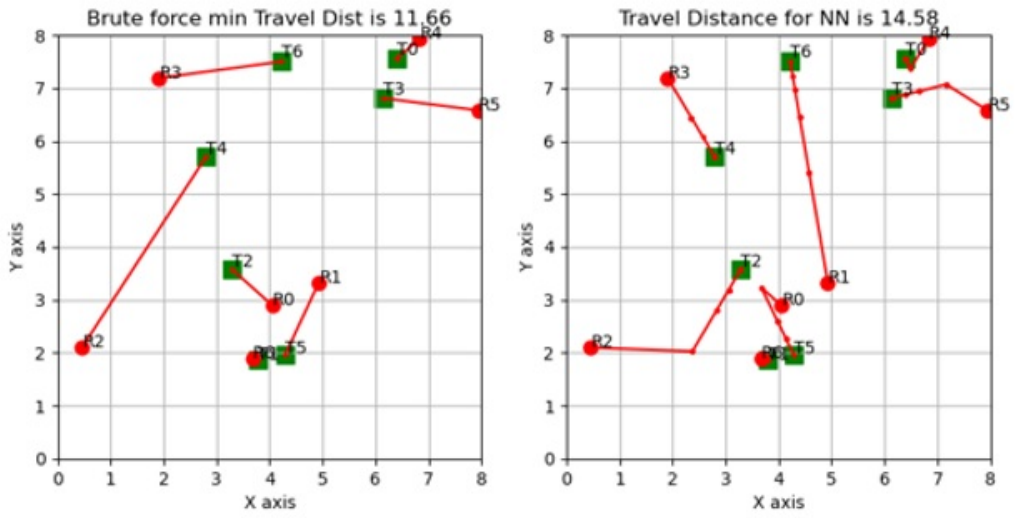

FIGURE 4.9: The global best task assignment solution provide by the brute force method (left panel)is slightly different than the SOM Neural Network (NN) solution (right panel), respectively to be 11.66 unit and 14.58 unit.

an estimate about how good this particular SOM result (as shown in the right panel of Fig.4.9) is.

- **Proof of Gaussian**: First, we feed all the data from the histogram in Fig.4.10 to Minitab, and conducted a Andreson-Darling Normality test. The QQ plot (Fig.4.11) shows that at confidence level 0.005, we reject the null hypothesis and conclude that there is strong statistical evidence that the arrangements' solution space is normal, with mean μ equals 26.84, and σ equals 5.172

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-u)^2}{2\sigma^2}} = \frac{1}{5.17\sqrt{2\pi}} e^{-\frac{(x-26.84)^2}{2\times 5.17^2}}$$

  If somebody just randomly assign a

- **Calculate the Z Score**: the ration of SOM NN path over the normal distributed population's standard deviation is:

$$Z_{\text{score}} = \frac{14.58}{-\sigma} = \frac{14.58}{-5.17} = -2.82 \tag{4.2}$$

  Then, from the "areas under standard normal distribution to the left of Z-Value" table, one can find the Z-value associated with this Z-score is 0.0024 = 0.24%. Therefore, we can concluded that at confidence level 0.005, the SOM method

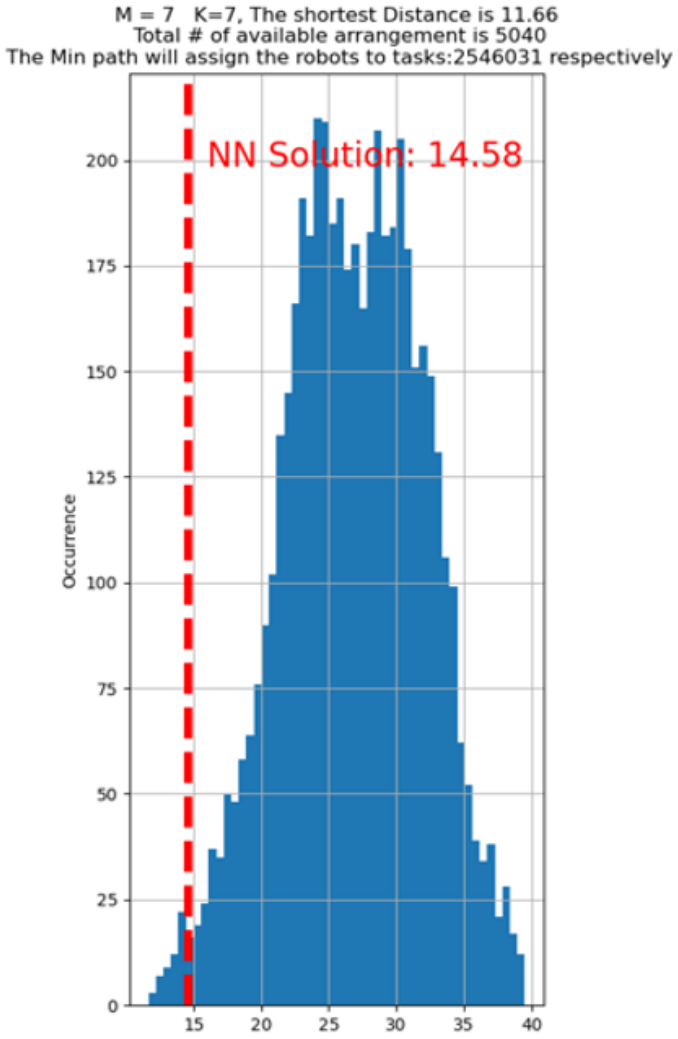

FIGURE 4.10: The histogram of all the possible total-traveling-distance on the x axis, and with their occurrence in the brute force arrangement space

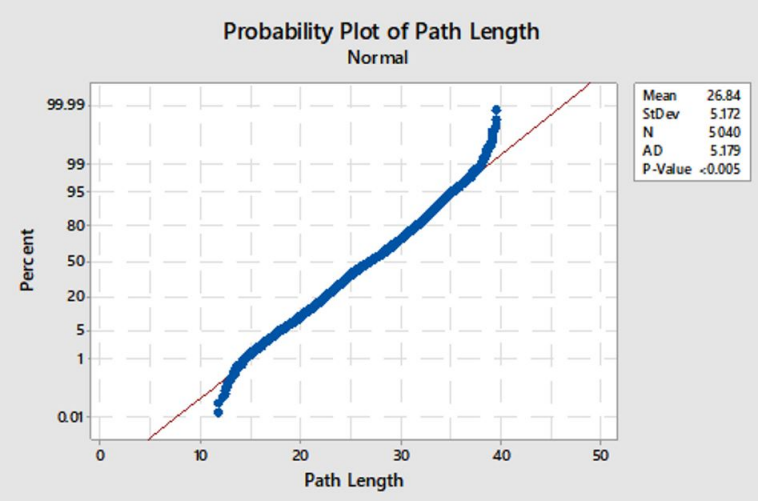

FIGURE 4.11: The normality test for the histogram in Fig.4.10

provided us a solution which is better than 99.76% $(1 - 0.0024 = 99.76\%)$ of the task assignment arraignments which are randomly generated from the solution space. As a comparison, the famous Japanese quality assurance term, six-sigma, is trying to control any process at Z-score -2.576 level, since the SOM is providing us the Z sore at -2.82 level (more tight than six-sigma), we would conclude that SOM NN provide people a very accurate and efficient way to solve this task assignment problem.

All the code regarding to this sub-section can be found at this Github Link or please go to the next url: https://github.com/ENGG6570/Group-Project/tree/main/ENGG6570_Project

Chapter 5

Summary

To summarise, we have reviewed the historical origin of self organizing map as well as the significance of solving (or at least providing a near optimal solution) the NP-complete task assignment problem.

Then, we have explored 3 different structures of SOM, namely to be 1) the color-map SOM, 2) the handwritten digit recognition SOM, 3) the robot task assignment SOM, which is based on Anmin's research paper: SOM-based multi-agent architecture for the multi-robot system. The difference between these three SOM models can be described as the following:

- The color-map SOM and the handwritten digit recognition SOM both have a 2D structure of the output layer, and the topological structure from the input data-set are retained on the output layer based on the fact that the neighbor neurons are selected based on the physical location of the neurons on the output layer;
- While, for the robot task assignment SOM, the output layer is a 1D list of neurons. Due to the fact that the neighbor neurons are selected based on the vector distance between the connecting weights of the output neurons, the topological structure of the input data-set is retained on the connecting weights of neurons on the output layer.

We have also explored different quantitative methods to evaluate the accuracy of a SOM. We found that the topographic error is not a good indicator to evaluate the accuracy of

a SOM for a colored picture; while, the change of output neurons' connecting weights in the vector space is a much better loss function to evaluate the accuracy of a SOM. For the robot task assignment SOM, we compared the SOM result with the brute force result. We have found that SOM provides a very accurate result at small scale (3 robots and 3 tasks), and SOM provides a near optimal result at large scale (7 robots and 7 tasks). By conducting a normality test of the entire arrangement space and calculate the Z sore of the SOM solution, we found that the SOM solution is more accurate than 99.76% of the randomly generated arrangement, which is better than the Six-Sigma level. Therefore, with all these quantitative evidence, we conclude that SOM is a very accurate and efficient way to solve the NP complete classification problems, especially the robot task assignment problem.

Bibliography

[1] A. Zhu and S. X. Yang, "A som-based multi-agent architecture for multirobot systems," *International Journal of Robotics & Automation*, vol. 21, no. 2, p. 91, 2006.

[2] T. Yang, J. Ma, Z.-G. Hou, G. Peng, and M. Tan, "A multi-agent architecture based cooperation and intelligent decision making method for multirobot systems," in *International Conference on Neural Information Processing*, pp. 376–385, Springer, 2007.

[3] J. Ni and S. X. Yang, "A fuzzy-logic based chaos ga for cooperative foraging of multi-robots in unknown environments," *International Journal of Robotics and Automation*, vol. 27, no. 1, p. 15, 2012.

[4] A. Zhu and S. X. Yang, "A neural network approach to dynamic task assignment of multirobots," *IEEE transactions on neural networks*, vol. 17, no. 5, pp. 1278–1287, 2006.

[5] U. Asan and S. Ercan, "An introduction to self-organizing maps," in *Computational Intelligence Systems in Industrial Engineering*, pp. 295–315, Springer, 2012.

[6] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[7] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[8] C. Fyfe, *Hebbian learning and negative feedback networks*. Springer Science & Business Media, 2007.

[9] D. T. Larose and C. Larose, "Discovering knowledge in data, john willey & sons," *Inc., Publication, New Jersey*, 2005.

[10] J. A. Lee and M. Verleysen, "Self-organizing maps with recursive neighborhood adaptation," *Neural Networks*, vol. 15, no. 8-9, pp. 993–1003, 2002.

[11] T. Reutterer, "Competitive market structure and segmentation analysis with self-organizing feature maps," in *Proceedings of the 27th EMAC Conference*, pp. 85–115, Citeseer, 1998.

[12] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[13] A. Ultsch, "Self-organizing neural networks for visualisation and classification," in *Information and classification*, pp. 307–313, Springer, 1993.

[14] E. Merényi, K. Tasdemir, and L. Zhang, "Learning highly structured manifolds: harnessing the power of soms," in *Similarity-based clustering*, pp. 138–168, Springer, 2009.

[15] R. Tatoian and L. Hamel, "Self-organizing map convergence," *International Journal of Service Science, Management, Engineering, and Technology (IJSSMET)*, vol. 9, no. 2, pp. 61–84, 2018.