

**รายงานการทดลอง**

การ Train Multilayer Perceptron โดยใช้ Particle Swarm Optimization (PSO)

**โดย**

นายณปพน วงค์คม

650610834

**เสนอ**

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 261456

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

**บทคัดย่อ**

การทดลองนี้มุ่งศึกษาประสิทธิภาพของอัลกอริธึมการหาค่าที่เหมาะสมที่สุดแบบฝูงอนุภาค (Particle Swarm Optimization, PSO) ในการปรับแต่งพารามิเตอร์และค่าน้ำหนักของโครงข่ายประสาทเทียมแบบเพอร์เซพตรอนหลายชั้น (Multilayer Perceptron, MLP) โดยประยุกต์ใช้เทคนิค PSO เพื่อหาค่าที่เหมาะสมในการ Train โมเดล

ผลการทดลองพบว่า การใช้ PSO สามารถปรับปรุงประสิทธิภาพการเรียนรู้ของ MLP ได้อย่างมีนัยสำคัญ โดยสังเกตจากค่าความคลาดเคลื่อนที่ลดลงและความแม่นยำในการทำนายที่เพิ่มขึ้น ซึ่งแสดงให้เห็นถึงศักยภาพของ PSO ในการเป็นเครื่องมือที่มีประสิทธิภาพสำหรับการ Optimize โมเดลเครือข่ายประสาทเทียม

**บทนำ**

**ที่มาและความสำคัญ**

ในปัจจุบัน เทคโนโลยีปัญญาประดิษฐ์และการเรียนรู้ของเครื่อง (Machine Learning) ได้กลายเป็นเครื่องมือสำคัญในการแก้ปัญหาที่ซับซ้อนในหลากหลายสาขา โดยเฉพาะโครงข่ายประสาทเทียมแบบเพอร์เซพตรอนหลายชั้น (Multilayer Perceptron, MLP) ซึ่งเป็นโมเดลที่มีความสามารถในการเรียนรู้และทำนายผลได้อย่างมีประสิทธิภาพ

การพัฒนาประสิทธิภาพของโครงข่ายประสาทเทียมจำเป็นต้องอาศัยเทคนิคการปรับแต่งพารามิเตอร์ที่เหมาะสม กระบวนการ Train โมเดลด้วยวิธีดั้งเดิมมักประสบปัญหาการติดอยู่ในจุดต่ำสุดเฉพาะที่ (Local Minimum) ซึ่งจำกัดความสามารถในการเรียนรู้และทำนาย จึงจำเป็นต้องค้นหาวิธีการใหม่ๆ เพื่อเพิ่มประสิทธิภาพ

อัลกอริธึมการหาค่าที่เหมาะสมที่สุดแบบฝูงอนุภาค (Particle Swarm Optimization, PSO) เป็นเทคนิคที่น่าสนใจในการแก้ปัญหานี้ โดยได้รับแรงบันดาลใจจากพฤติกรรมการเคลื่อนที่ของฝูงนก PSO สามารถค้นหาพื้นที่โซลูชันได้อย่างครอบคลุมและมีประสิทธิภาพ การประยุกต์ใช้ PSO กับ MLP จึงเป็นแนวทางที่มีศักยภาพในการปรับปรุงประสิทธิภาพการเรียนรู้ของโมเดล

**วัตถุประสงค์**

1. เพื่อพัฒนาโมเดล Multilayer Perceptron (MLP) ที่ได้รับการปรับแต่งพารามิเตอร์ด้วยวิธี Particle Swarm Optimization (PSO)
2. เพื่อเปรียบเทียบประสิทธิภาพการทำนายระหว่างโมเดล MLP ที่ Train ด้วย PSO เมื่อมีจำนวน Hidden layer ต่างๆ
3. เพื่อศึกษาความสามารถในการลดค่าความคลาดเคลื่อน (Error) ของโมเดลโดยใช้เทคนิค PSO

**ขอบเขตการทดลอง**

1. ข้อมูลจาก AirQualityUCI (Air Quality Data Set จาก UCI Machine learning Repository) โดยที่ data set นี้มีทั้งหมด 9358 sample และมี 14 attribute
2. ข้อมูลถูกแบ่งเป็น 90 % สำหรับ Train และ 10 % สำหรับ Test โดยทำ K-fold validation เป็นจำนวน10 folds
3. ใช้จำนวนรอบการค้นหา (Iteration) สูงสุดเป็น 50 รอบ และจำนวนอนุภาค (Particle) เป็น 10 อนุภาค
4. ใช้ ฟังก์ชันกระตุ้น (Activation Function) แบบ Sigmoid
5. ในการหาวัดค่าความคลาดเคลื่อนใช้ Mean Absolute Error

**วิธีดำเนินการ**

1.เขียนโปรแกรมเพื่อฝึกฝน Multilayer Perceptron โดยใช้ Particle Swarm Optimization สำหรับทดสอบ validity, ความเร็วในการ converge และความถูกต้อง accuracy

2.กำหนดค่า hidden layer, iterations, cognitive, social, inertia, particle ที่เหมาสม

3.นำข้อมูลที่มีไปทำการ k-fold validation เพื่อฝึกฝนโมเดล โดยแบ่งเป็น 2 แบบ คือ ทำนาย 5 วันล่วงหน้า กับ 10 วันล่วงหน้า

4.นำข้อมูลที่มีไปทำการ k-fold validation เพื่อนำข้อมูลมาทดสอบกับโมเดล

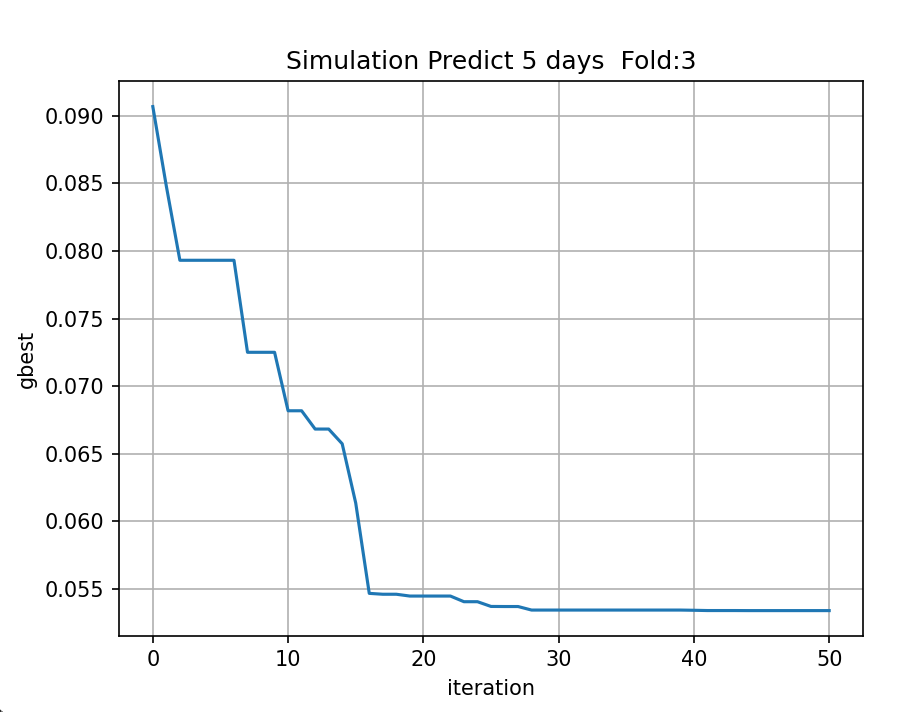
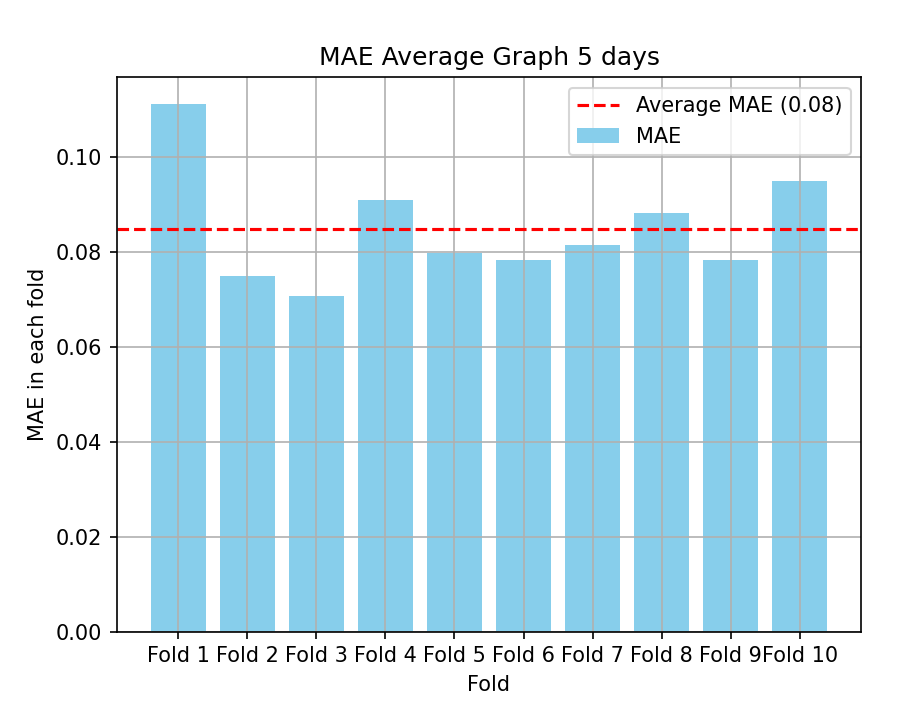
5.บันทึกผลการทดสอบใรรูปของกราฟ

6.ทำตามข้อ 2. ถึง 5. ซ้ำเรื่อยๆ โดยเปลี่ยนแปลงค่า hidden layer และ node จนได้ผลการทดลองที่เพียงพอ

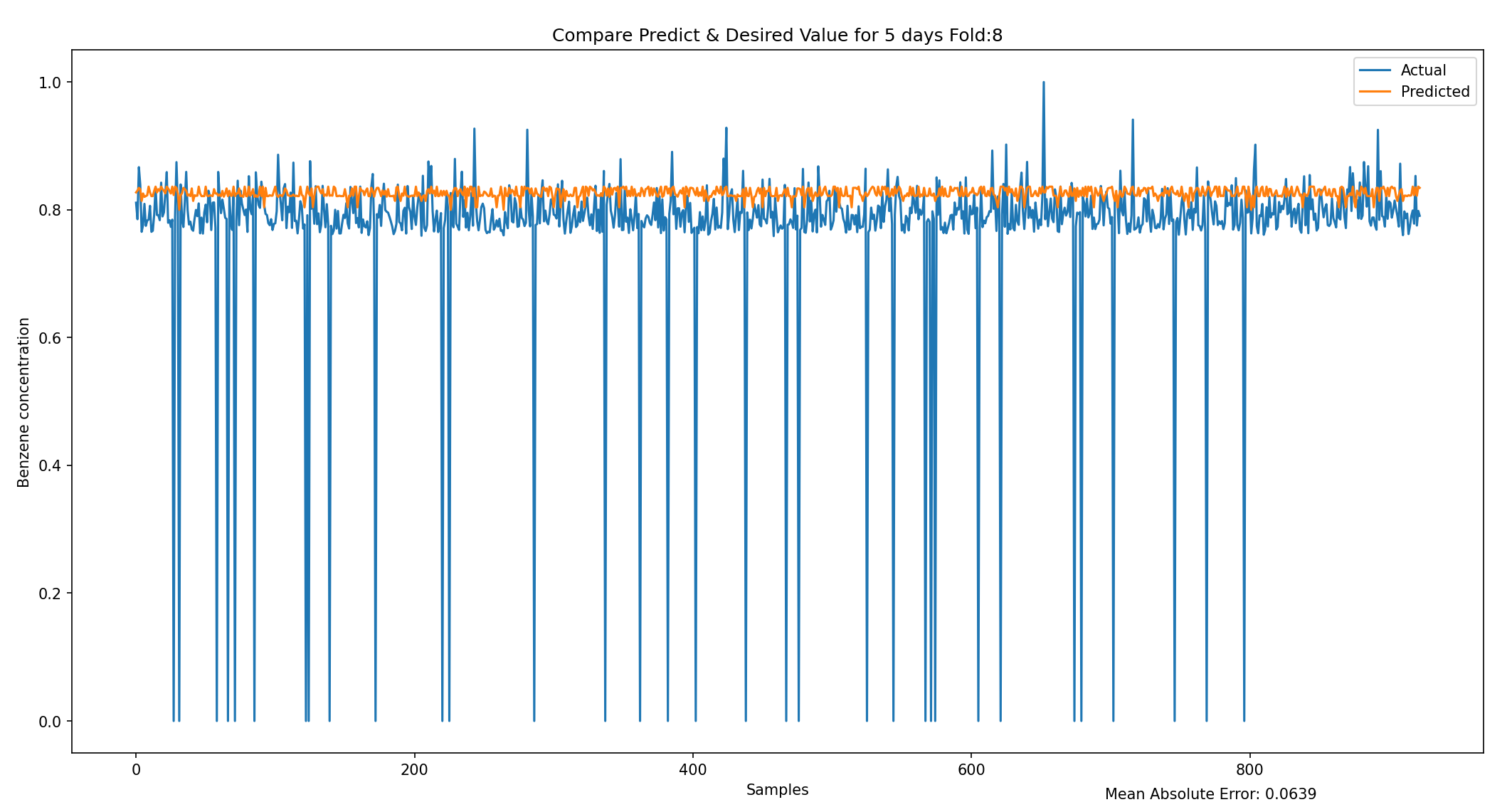
**ผลการทดลอง**

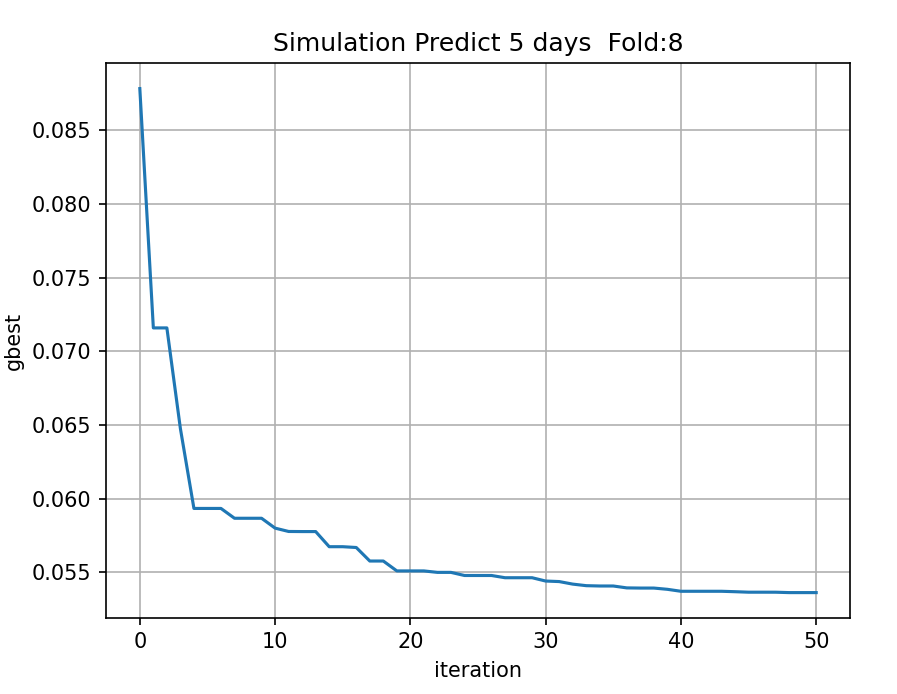
A graph with blue and orange lines

Description automatically generated1. ผลการทดลอง ทำนาย Benzene concentration ล่วงหน้า 5 วัน เมื่อ hidden layer คือ 32,16 ,particles คือ 10 ,cognitive คือ 1.5 ,social คือ 1.5 ,inertia คือ 0.7 และ iterations คือ 50



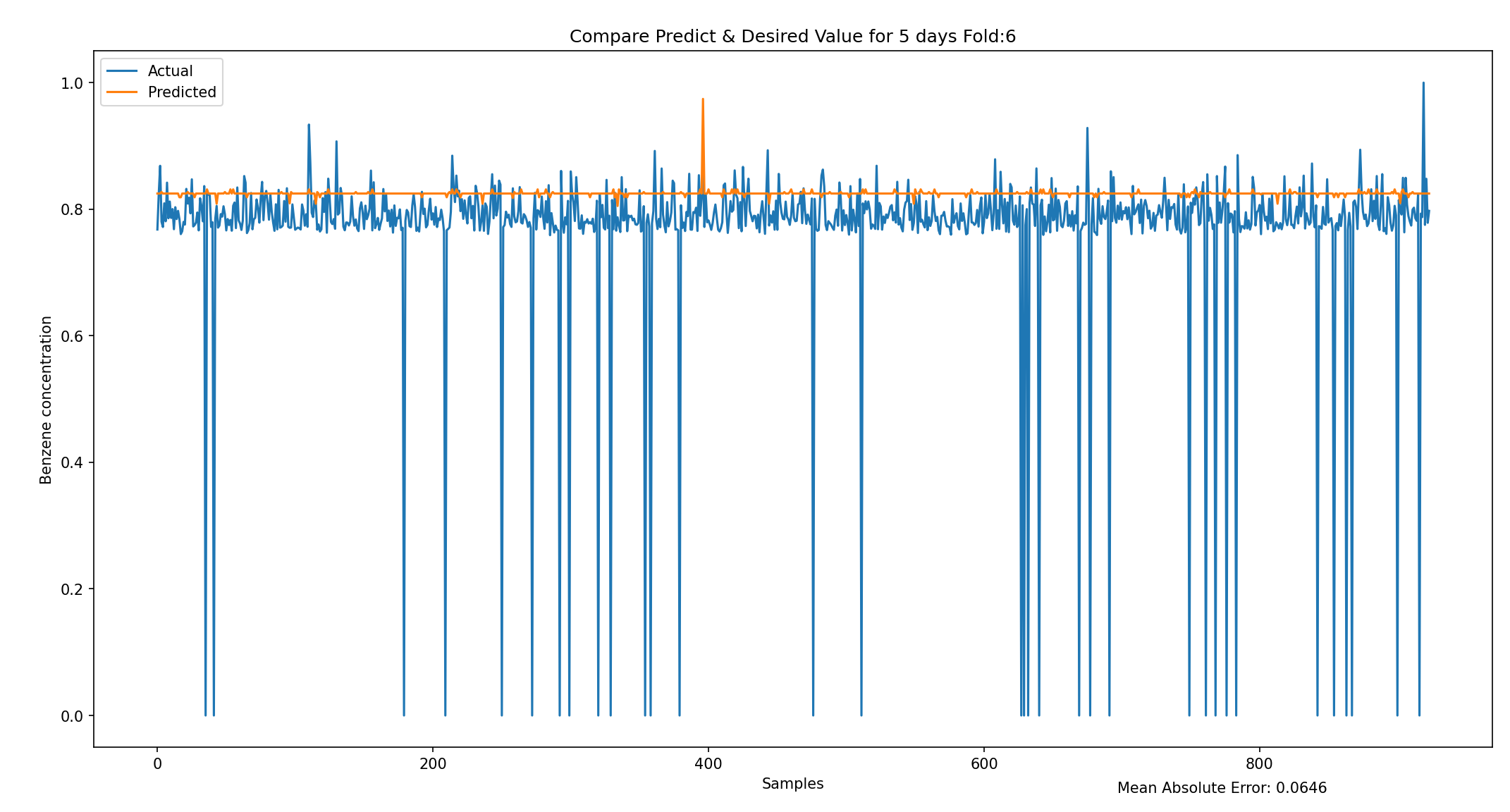
จากการทดลอง เมื่อ hidden layer คือ 32,16 จะได้ Average MAE ที่ 0.0834 หลัง iteration ที่ 16 ค่า MAE เริ่มคงที่

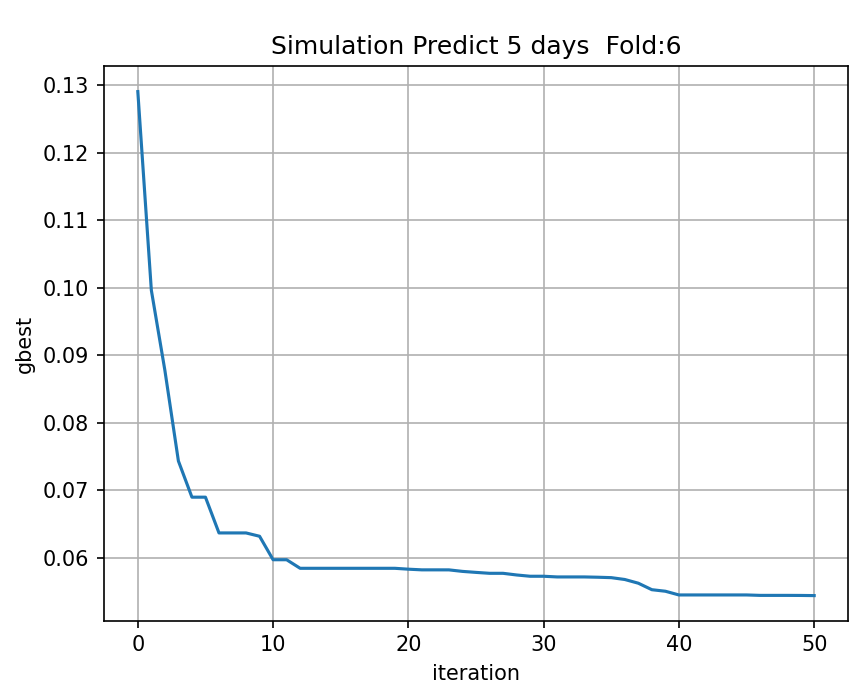
2. ผลการทดลอง ทำนาย Benzene concentration ล่วงหน้า 5 วัน เมื่อ hidden layer คือ 16,4 ,particles คือ 10 ,cognitive คือ 1.5 ,social คือ 1.5 ,inertia คือ 0.7 และ iterations คือ 50

A graph with blue and white lines

Description automatically generated

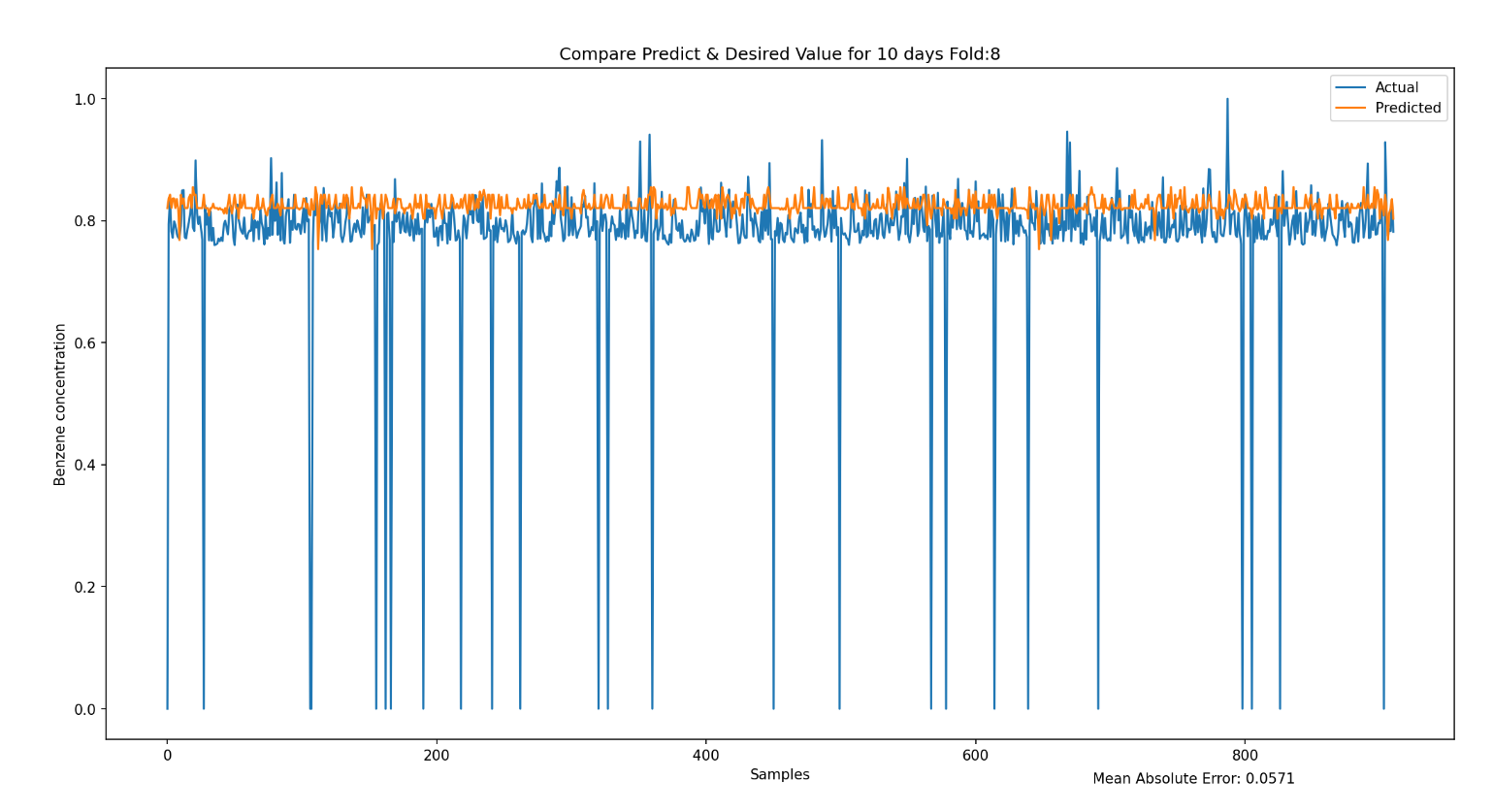
จากการทดลอง เมื่อ hidden layer คือ 16,4 จะได้ Average MAE ที่ 0.0928 หลัง iteration ที่ 20 ค่า MAE เริ่มคงที่

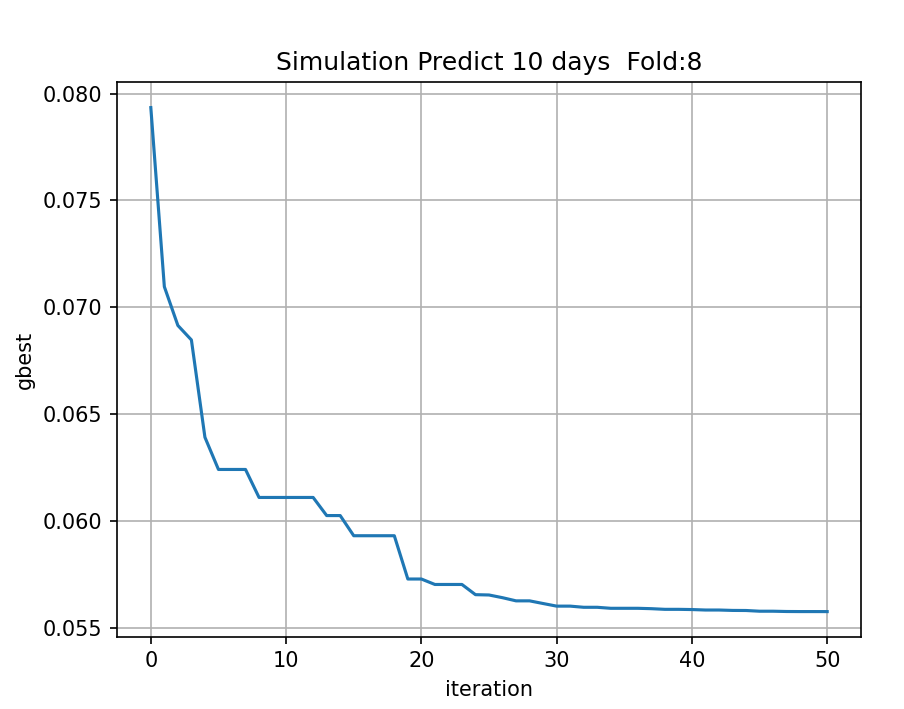
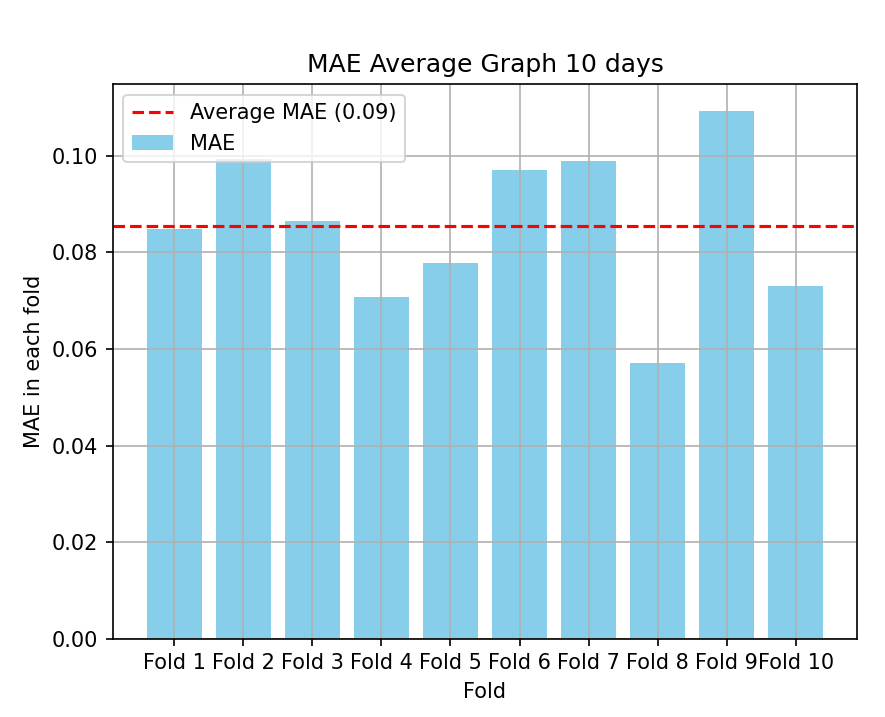
3. ผลการทดลอง ทำนาย Benzene concentration ล่วงหน้า 5 วัน เมื่อ hidden layer คือ 8 ,particles คือ 10 ,cognitive คือ 1.5 ,social คือ 1.5 ,inertia คือ 0.7 และ iterations คือ 50

A graph with blue bars and red lines

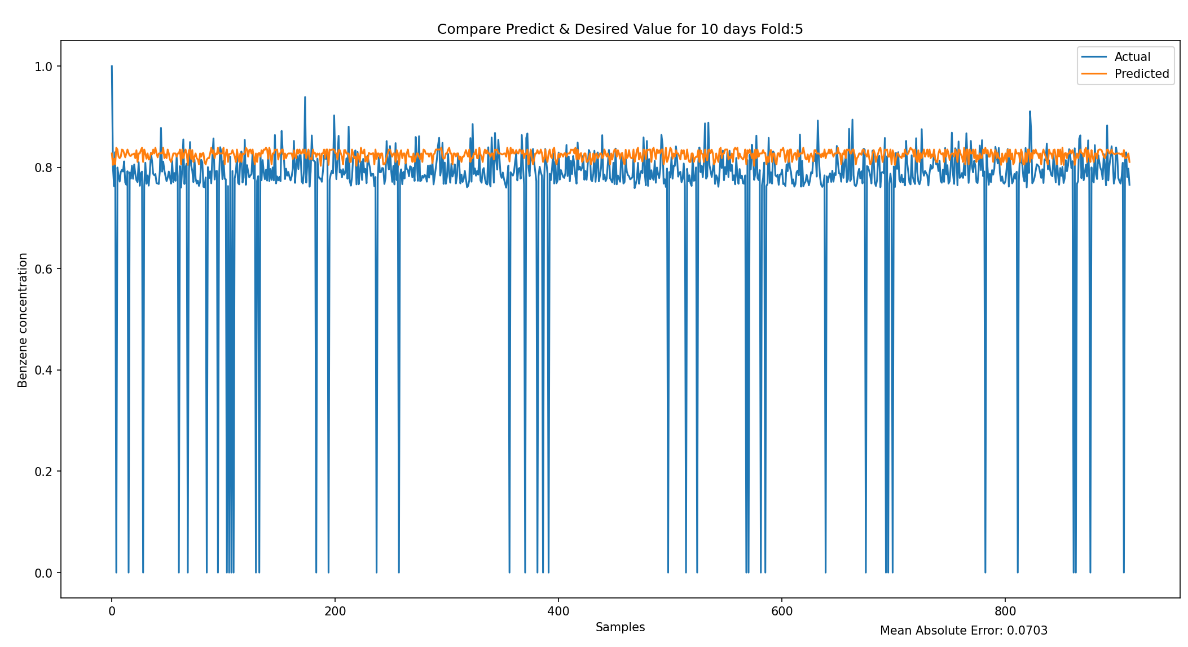
Description automatically generated

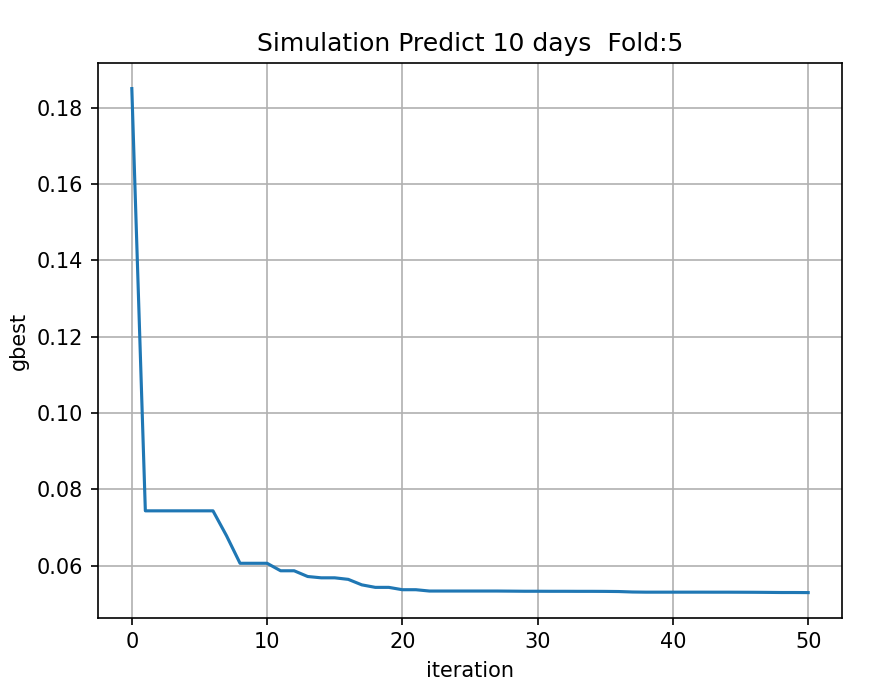
จากการทดลอง เมื่อ hidden layer คือ 8 จะได้ Average MAE ที่ 0.0987 หลัง iteration ที่ 11 ค่า MAE เริ่มคงที่

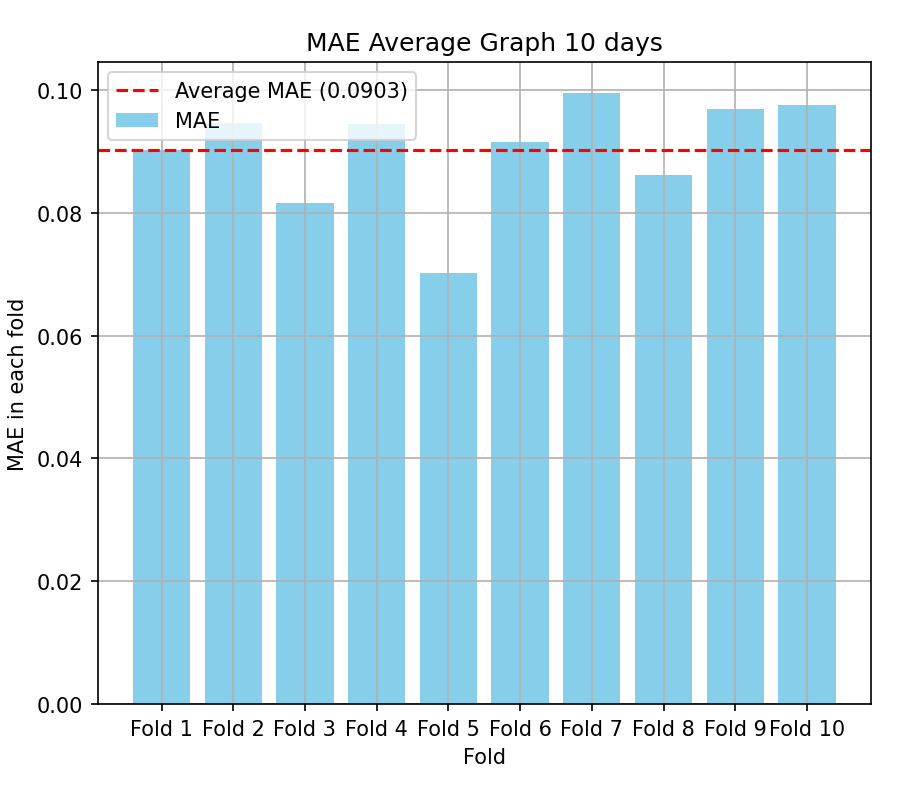
4. ผลการทดลอง ทำนาย Benzene concentration ล่วงหน้า 10 วัน เมื่อ hidden layer คือ 32,16 ,particles คือ 10 ,cognitive คือ 1.5 ,social คือ 1.5 ,inertia คือ 0.7 และ iterations คือ 50



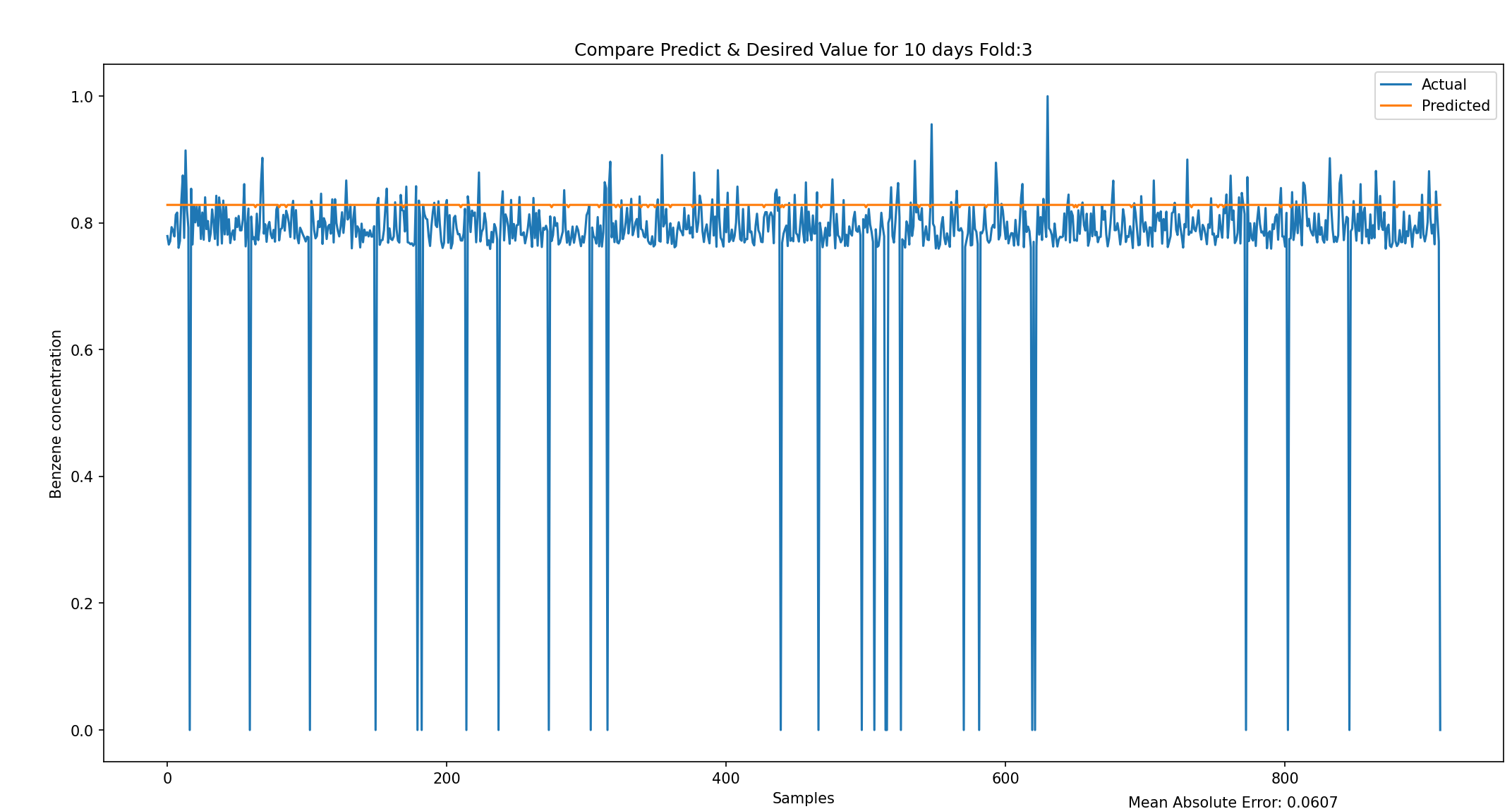
จากการทดลอง เมื่อ hidden layer คือ 32,16 จะได้ Average MAE ที่ 0.0851 หลัง iteration ที่ 30 ค่า MAE เริ่มคงที่

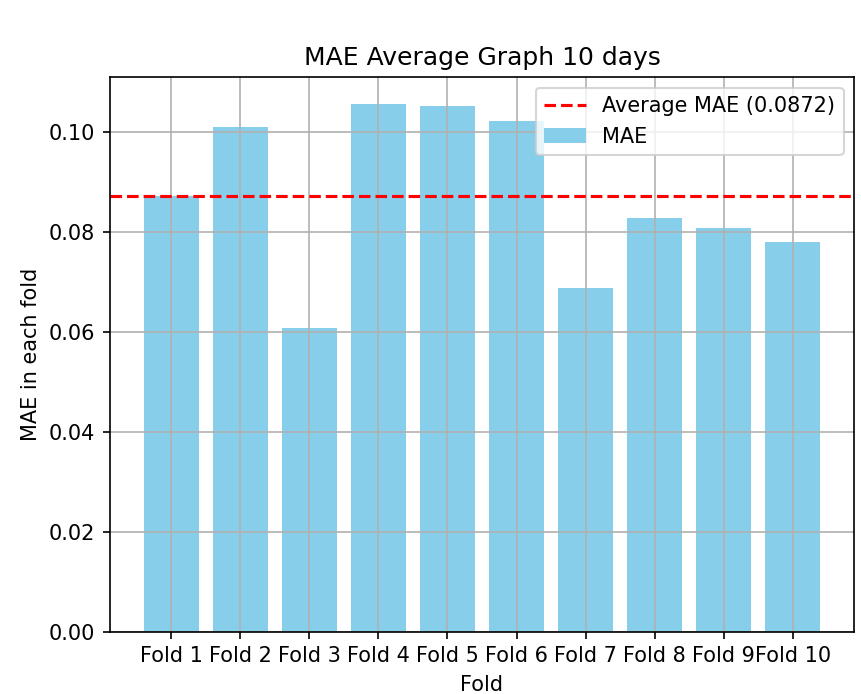
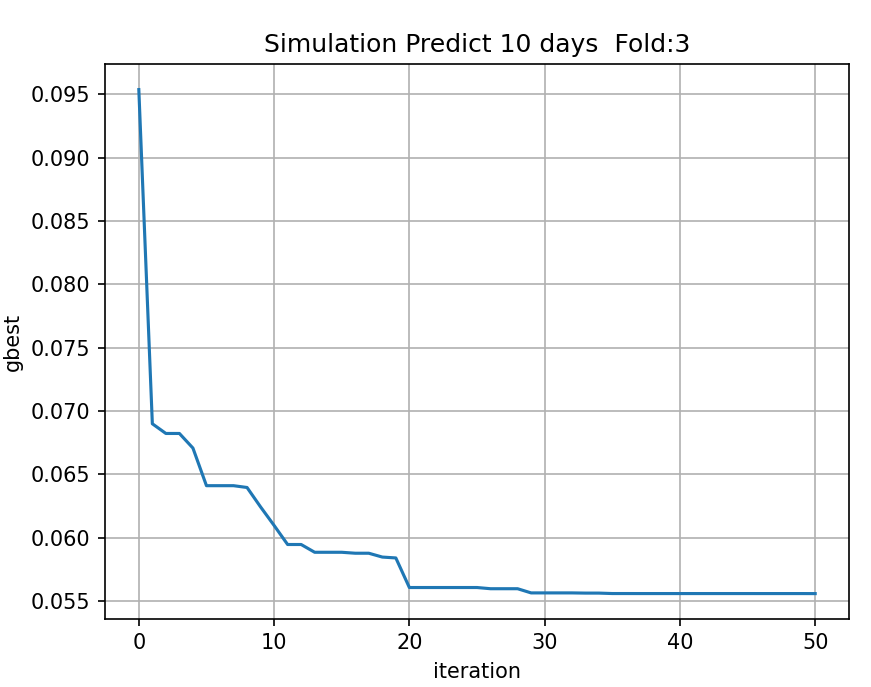
5. ผลการทดลอง ทำนาย Benzene concentration ล่วงหน้า 10 วัน เมื่อ hidden layer คือ 16,4 ,particles คือ 10 ,cognitive คือ 1.5 ,social คือ 1.5 ,inertia คือ 0.7 และ iterations คือ 50





จากการทดลอง เมื่อ hidden layer คือ 16,4 จะได้ Average MAE ที่ 0.0903 หลัง iteration ที่ 20 ค่า MAE เริ่มคงที่

6. ผลการทดลอง ทำนาย Benzene concentration ล่วงหน้า 10 วัน เมื่อ hidden layer คือ 8 ,particles คือ 10 ,cognitive คือ 1.5 ,social คือ 1.5 ,inertia คือ 0.7 และ iterations คือ 50



จากการทดลอง เมื่อ hidden layer คือ 8 จะได้ Average MAE ที่ 0.0872 หลัง iteration ที่ 20 ค่า MAE เริ่มคงที่

**วิเคราะห์ผลการทดลอง**

จากผลการทดลอง เพื่อศึกษา validity, ความเร็วในการ converge และความถูกต้อง Accuracy ของ multilayer perceptron ที่ถูก train ด้วย particle swarm optimization ทำให้ทราบว่า จำนวน hidden layer มีผลต่อ validity, ความเร็วในการ converge และความถูกต้อง Accuracy ของ multilayer perceptron โดยหาก hidden layer มากจะส่งผลให้ multilayer perceptron มีความเร็วในการ converge ช้า แต่ความแม่นยำก็จะมาก ถ้าหาก hidden layer น้อยจะส่งผลให้ความเร็วในการ converge เร็วขึ้น แต่ในทางกลับกันความแม่นยำก็ต่ำลงเช่นกัน ดังนั้นการจะให้ multilayer perceptron มีประสิทธิภาพ มีความแม่นยำสูง มีความเร็วในการ converge ที่เร็ว จะต้องมีค่าของ hidden layer ที่เหมาะสม ไม่มากเกินไป ไม่น้อยเกินไป

**ภาคผนวก**

Github :

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

class MLP:

    def \_\_init\_\_(self, layer\_sizes):

        self.layer\_sizes = layer\_sizes

        self.parameters = self.initialize\_parameters()

    def sigmoid(self, x):

        return 1 / (1 + np.exp(-x))

    def initialize\_parameters(self):

        # Create an object array with the shape (number of layers - 1, 2)

        parameters = np.empty((len(self.layer\_sizes) - 1, 2), dtype=object)

        for i in range(len(self.layer\_sizes) - 1):

            # Initialize weights and biases for each layer

            W = np.random.randn(self.layer\_sizes[i], self.layer\_sizes[i + 1])

            b = np.zeros((1, self.layer\_sizes[i + 1]))  # Biases

            parameters[i, 0] = W  # Place weights in first column

            parameters[i, 1] = b  # Place biases in second column

        return parameters

    def forward\_propagation(self, X):

        A = X

        caches = []

        for i in range(len(self.parameters) - 1):

            W, b = self.parameters[i]

            Z = np.dot(A, W) + b

            A = self.sigmoid(Z)

            caches.append(A)

        W, b = self.parameters[-1]

        Z = np.dot(A, W) + b

        A = self.sigmoid(Z)

        caches.append(A)

        return caches[-1][0][0]

class PSO:

    def \_\_init\_\_(self,mlp,input,Y,particle,c\_cognitive,c\_social,inertia,max\_iteration,ndays,fold):

        self.mlp = mlp

        self.input = input

        self.Y = Y

        self.particle = particle

        self.c\_cognitive = c\_cognitive

        self.c\_social = c\_social

        self.inertia = inertia

        self.max\_iteration = max\_iteration

        self.ndays = ndays

        self.fold = fold

        self.value\_check()

        self.history = []

        self.cache\_iter = []

    def value\_check(self):

        if((self.c\_cognitive + self.c\_social) > 4):

            raise ValueError('sum of c\_cognitive & c\_social should not exceed 4')

    def objective\_func(self,w):

        self.mlp.parameters = w

        Y\_output = []

        for x in self.input:

            Y\_output.append(self.mlp.forward\_propagation(x))

        return MAE(normalize(self.Y,self.Y),Y\_output)

    def rho(self,c):

        r = np.random.rand()

        return r\*c

    def v\_optimize(self,v,w\_pbest,w\_gbest,w):

        v\_new = (self.inertia \* v) + (self.rho(self.c\_cognitive) \* (w\_pbest - w)) + (self.rho(self.c\_social) \* (w\_gbest - w))

        return v\_new

    def train(self):

        v = {}

        w = {}

        pbest = np.zeros(self.particle)

        for k in range(self.particle):

            v[k] = np.random.uniform(-0.1,0.1)

            w[k] = self.mlp.initialize\_parameters()

            pbest[k] = self.objective\_func(w[k])

        w\_pbest = w

        gbest = pbest[np.argmin(pbest)]

        w\_gbest = w\_pbest[np.argmin(pbest)]

        iteration = 0

        while iteration <= self.max\_iteration:

            for p in range(self.particle):

                score = self.objective\_func(w[p])

                if score < pbest[p]:

                    pbest[p] = score

                    w\_pbest[p] = w[p]

                if score < gbest:

                    gbest = score

                    w\_gbest = w[p]

                v[p] = self.v\_optimize(v[p],w\_pbest[p],w\_gbest,w[p])

                w[p] = w[p] + v[p]

            if (iteration > 0):

                print(f"iteration : {iteration}/{self.max\_iteration} gobal best : {gbest}")

            self.cache\_iter.append(iteration)

            self.history.append(gbest)

            iteration += 1

        self.mlp.parameters = w\_gbest

    def test(self, Xtest, Ytest, buffer):

        predicts = []

        for x in Xtest:

            predict = self.mlp.forward\_propagation(x)

            predicts.append(predict)

        predicts = np.array(predicts)

        nor\_Ytest = normalize(Ytest,Ytest)

        mean\_abs\_error = MAE(nor\_Ytest, predicts)

        buffer.append(mean\_abs\_error)

        plt.figure()

        plt.title(f"Compare Predict & Desired Value for {self.ndays} days Fold:{self.fold+1}")

        plt.plot(nor\_Ytest, label="Actual")

        plt.plot(predicts, label="Predicted")

        plt.ylabel("Benzene concentration")

        plt.xlabel("Samples")

        plt.legend()

        plt.figtext(0.75, 0.06, f"Mean Absolute Error: {mean\_abs\_error:.4f}", ha="center", fontsize=10)

        print(f"MAE of output is : {mean\_abs\_error}")

    def plot(self):

        plt.figure()

        plt.title(f'Simulation Predict {self.ndays} days  Fold:{self.fold+1}')

        plt.plot(self.cache\_iter,self.history)

        plt.xlabel('iteration')

        plt.ylabel('gbest')

        plt.grid()

    def bar\_plot(self,MAE\_set):

        x\_set = []

        for i in range(10):

            x\_set.append(f"Fold {i+1}")

        average\_MAE = np.mean(MAE\_set)

        x = x\_set

        y = MAE\_set

        plt.figure()

        plt.title(f'MAE Average Graph {self.ndays} days')

        plt.bar(x,y,color='skyblue', label="MAE")

        plt.axhline(y=average\_MAE, color='red', linestyle='--', label=f"Average MAE ({average\_MAE:.4f})")

        plt.xlabel("Fold")

        plt.ylabel("MAE in each fold")

        plt.legend()

        plt.grid()

def import\_data(file):

    """

    function to import and tranfer data to array

    """

    data = pd.read\_excel(file)

    array\_data = data.to\_numpy()

    new\_data = []

    attribute = [3,6,8,10,11,12,13,14,5]

    for i in range(len(data)):

        data\_buffer = []

        for j in range(len(attribute)):

            data\_buffer.append(array\_data[i][attribute[j]])

        new\_data.append(data\_buffer)

    data\_set = np.array(new\_data)

    return data\_set

def k\_fold\_validation(data\_set, i , k):

    np.random.shuffle(data\_set)

    X = data\_set[ : , : -1]

    Y = data\_set[ : ,-1]

    Xset = np.array\_split(X,k)

    Yset = np.array\_split(Y,k)

    Xtest = Xset[i]

    Ytest = Yset[i]

    Xtrain = np.concatenate([Xset[j] for j in range(k) if j != i])

    Ytrain = np.concatenate([Yset[j] for j in range(k) if j != i])

    return Xtrain, Ytrain, Xtest, Ytest

def MAE(true\_Y,Y):

    true\_Y = np.array(true\_Y)

    Y = np.array(Y)

    MAE = np.mean(np.absolute(true\_Y - Y))

    return MAE

def normalize(data,value):

    """

    function to normalize value to [0,1]

    """

    min\_val = np.min(data)

    max\_val = np.max(data)

    result = (value - min\_val) / (max\_val - min\_val)

    return result

def denormalize(data,value):

    min\_val = np.min(data)

    max\_val = np.max(data)

    result = value \* (max\_val - min\_val) + min\_val

    return result

def create\_dataset\_for\_prediction(data, look\_ahead):

    """

    Function to create dataset for multi-day look-ahead prediction.

    """

    X = data[ : , :-1]

    Y = data[ : ,-1]

    Y = np.roll(Y,-24 \* look\_ahead)

    new\_data = np.concatenate((X,Y.reshape(-1,1)),axis=1)

    new\_data = new\_data[ : -24 \* look\_ahead]

    return new\_data

if \_\_name\_\_ == '\_\_main\_\_':

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    plt.close("all")

    file = 'AirQualityUCI.xlsx'

    k = 10 # number of fold

    data\_set = import\_data(file)

    data\_5day = create\_dataset\_for\_prediction(data\_set,5)

    data\_10day = create\_dataset\_for\_prediction(data\_set,10)

    layer\_size = [8,16,1]

    particle = 10

    c\_cognitive = 1.5

    c\_social = 1.5

    inertia = 0.7

    max\_iteration = 50

    plot\_buffer\_5day = []

    plot\_buffer\_10day = []

    # #=============================================================================

    for f in range(k):

        print("======================================")

        print(f"Fold : {f+1}")

        Xtrain5day,Ytrain5day,Xtest5day,Ytest5day = k\_fold\_validation(data\_5day,f,k)

        Xtrain10day,Ytrain10day,Xtest10day,Ytest10day = k\_fold\_validation(data\_10day,f,k)

        mlp5day = MLP(layer\_size)

        pso\_5day = PSO(mlp5day,Xtrain5day,Ytrain5day,particle,c\_cognitive,c\_cognitive,inertia,max\_iteration,5,f)

        pso\_5day.train()

        pso\_5day.test(Xtest5day,Ytest5day,plot\_buffer\_5day)

        pso\_5day.plot()

        mlp10day = MLP(layer\_size)

        pso\_10day = PSO(mlp10day,Xtrain10day,Ytrain10day,particle,c\_cognitive,c\_cognitive,inertia,max\_iteration,10,f)

        pso\_10day.train()

        pso\_10day.test(Xtest10day,Ytest10day,plot\_buffer\_10day)

        pso\_10day.plot()

    pso\_5day.bar\_plot(plot\_buffer\_5day)

    pso\_10day.bar\_plot(plot\_buffer\_10day)

    plt.show()