Computer Assignment 3

จัดทำโดย

นายรักษ์พงศ์ ทอหุล 600610769

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของวิชา

CPE 261456 (Introduction to Computational Intelligence)

ภาคเรียนที่ 1 ปีการศึกษา 2563

มหาวิทยาลัยเชียงใหม่

สารบัญ

Method

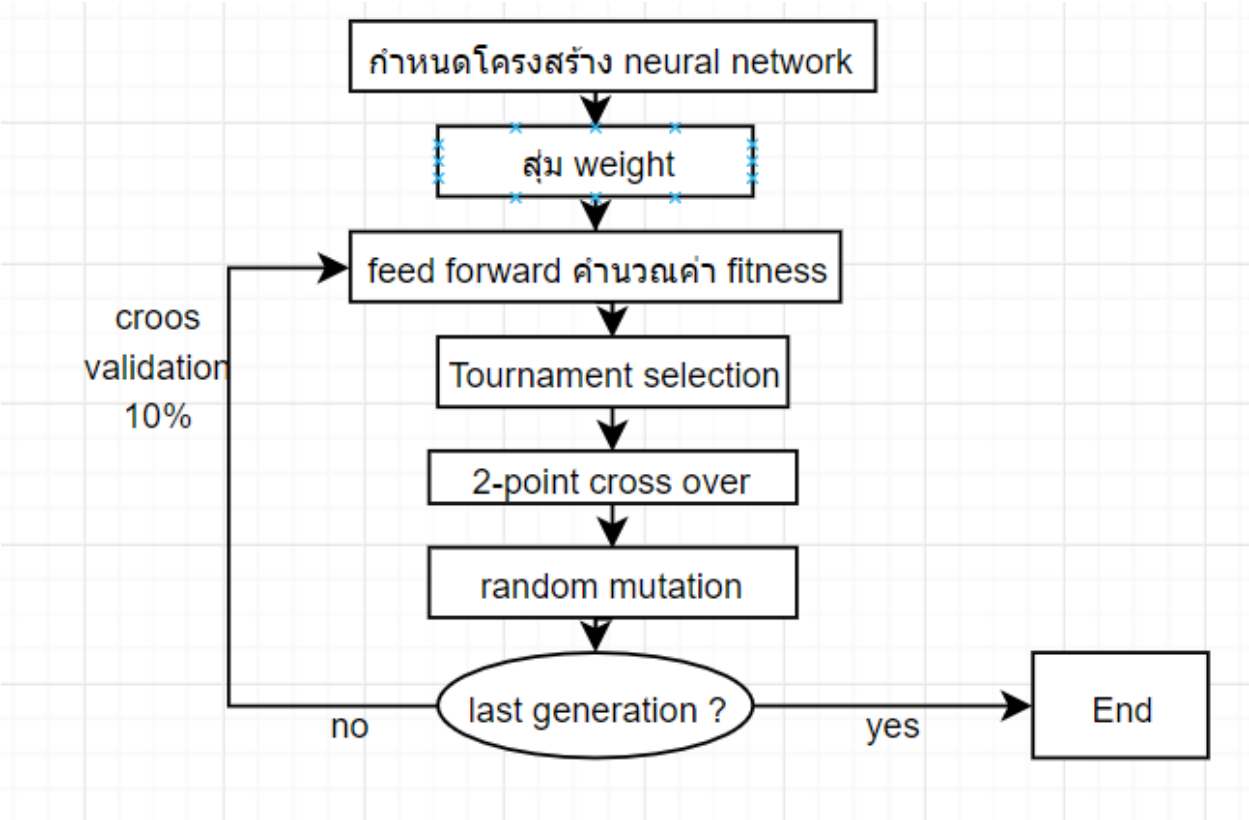
การทำงานของระบบโดยรวม

1. ระบบจะรับอินพุตมาจากไฟล์ wdbc.data โดยเป็นข้อมูล Wisconsin Diagnostic Breast Cancer จาก UCI Machine learning Reposity โดยมีข้อมูลเป็น feature ทั้งหมด 30 feature และมี 1 labels เป็นการระบุว่าเป็นมะเร็งหรือไม่เป็นมะเร็ง ในการทดลองนี้ผมได้ทำการสร้าง neural network ขึ้นมาโดยใช้ input layer เป็น 30 และ output layer เป็น 1 เปลี่ยน weight โดยใช้หลักการของ genetic algorithm โดยข้อมูลก่อนที่ระบบจะรับมานั้นได้ทำการ normalization ค่าให้อยู่ในช่วง 0 ถึง 1 และค่าของ labels มีค่าเป็น 0 และ 1
2. ทำการกำหนดจำนวนประชากรโดยในที่นี้ผมได้กำหนดประชากรเป็น 50 หมายถึงผมจะทำการสร้าง neural network ที่มีโครงสร้างเหมือนกันจำนวน 50 network
3. ทำการสุ่มค่า weight ในแต่ละโหนดในแต่ละ neural network ทั้ง 50 network
4. ค่าอินพุตจากไฟล์ที่กำหนดมาจะทำการ cross validation 10% เป็น train data 90% แล้วนำเข้าสู่ neural network ซึ่งจะทำการ cross validation 10% ใหม่ทุกครั้งที่ทำ neural network ใหม่
5. นำ train data ที่ได้มาผ่านเข้า neural network โดยนำ input mathix ไปคูณกับ weight แต่ละตัว จากนั้นนำไปเข้า activation function แล้วนำไปเข้าเป็น input ของ layer ถัดไปกระบวนการนี้เรียกว่า feed forward
6. จะได้ค่าผลลัพธ์สุดท้ายมาเราก็จะนำมาทำการคำนวณค่า fitness ซึ่งในข้อนี้คือปัญหาการ classification ดังนั้นผมจึงกำหนดการหาค่า fitness เป็น fitness = accuracy/จำนวนpopulation โดยหา accuracy จากความแม่นยำในการ classify ของ model แต่ละตัว
7. ทำกระบวนการคัดเลือก โดยผมเลือกใช้วิธีคัดเลือกแบบแข่งขัน โดยจะสุ่มจับคู่ population ทั้งหมดมา จากนั้นกำหนดค่า prob ไว้ 1 ค่าแล้วสุ่มเลขระหว่าง 0-1 มาในแต่ละรอบหากเลขที่สุ่มมาน้อยกว่าค่า prob จะนำ population ที่ค่า fitness น้อยกว่าไปทำการสืบพันธุ์แต่หากเลขที่สุ่มมามากกว่าค่า prob จะนำ population ที่ค่า fitness มากกว่าไปทำการสืบพันธุ์แทน
8. ก็จะเลือก population อยู่ครึ่งนึงก็จะนำมาทำการ cross over กันเพื่อสร้างประชากรใหม่จนครบ 50 โดยจะทำการ cross over แบบสองจุดโดยสุ่มจุดมาสองซึ่งก็คือตำแหน่งของ weight ใน neural

network จากนั้นทำการสลับค่า weight ของสองตัวตามจุดตัด ก็จะได้ neural network ที่มีค่า weight ใหม่ เป็นประชากรใหม่

9. ทำการสุ่มประชากรมาทำการกลายพันธุ์ โดยจะสุ่มตำแหน่งของ weight แล้วทำการสุ่มค่า weight นั้นใหม่
10. ทำการวนลูปทำงานเรื่อย ๆ ตามค่า generation ที่กำหนดไว้

โดยจะมีแผนผังการทำงานคร่าว ๆ ดังนี้



รูปที่ 1 แสดงแผนผังการทำงานของระบบ

Result

ในการทดลองนี้จะทำการทดลองเปลี่ยนแปลงค่าต่าง ๆ ดังนี้

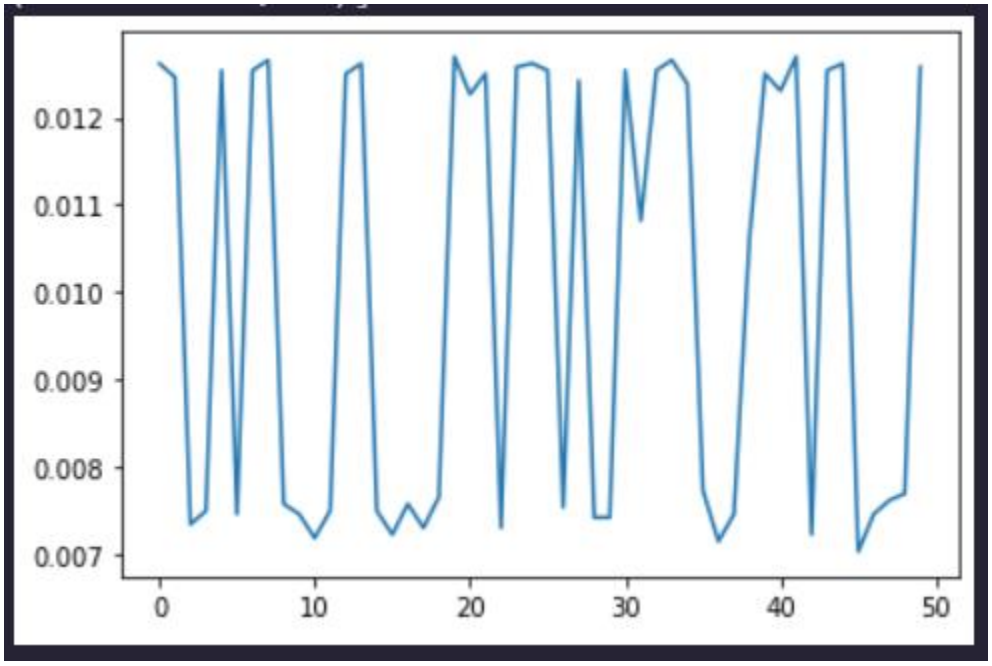
1. ทดลองเปลี่ยนจำนวนโหนดของ hidden layer
   - จำนวน hidden layer จะคงที่ที่ 2 layer
   - จำนวนโหนดจะมี 2, 5, 10 โหนด (ทุก layer จำนวนโหนดเท่ากัน)
2. ทดลองเปลี่ยนจำนวน hidden layer
   - จำนวนโหนดจะคงที่ที่ 2 ทุก layer
   - จำนวน hidden layer จะมี 2, 5, 7 layer (ทุก layer จำนวนโหนดเท่ากัน)

โดยจะมีค่าที่กำหนดไว้ดังนี้

- จำนวนประชากรเท่ากับ 50 ประชากร
- Activation function ใช้ sigmoid ทั้งหมดยกเว้น output ที่จะนำไปแปลงค่าให้เป็น 1 หรือ 0 เท่านั้น
- ค่า prob ในกระบวนการคัดเลือกแบบแข่งขันเป็น 0.4

โดยได้ผลการทดลองดังนี้

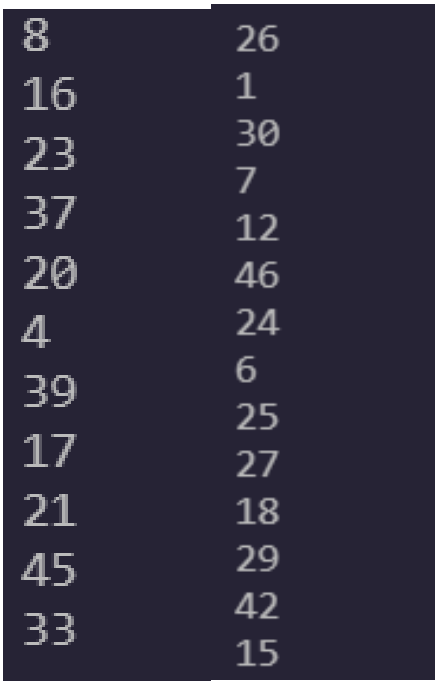
- ทดลองทำการสร้าง neural network 50 network ที่ weight ต่างกันแล้วนำมา feed forward ด้วย train input ที่ cross validation 10% ใหม่ทุกครั้งแล้วทำการคำนวณหาค่า fitness จากค่า accuracy/จำนวนประชากร ในครั้งแรกได้ผลดังนี้



รูปที่ 2 แสดงค่า fitness ที่ได้จากประชากรแต่ละตัว

- และในส่วนของกระบวนการคัดเลือกได้ทำการทดลองได้ประชากรที่ได้รับการคัดเลือกดังนี้

```
def Tselection(prob,fitness):
    fit = fitness.copy()
    random.shuffle(fit)
    half = len(fit)//2
    select = []
    for i in range(half):
        r = random.randrange(0,1)
        if(fit[i][0] < fit[half+i][0]):
            if(r<prob):
                select.append(fit[i])
            else:
                select.append(fit[half+i])
        else:
            if(r<prob):
                select.append(fit[half+i])
            else:
                select.append(fit[i])
    return select
```



โดยจะได้ว่าประชากรที่ได้รับการคัดเลือกจะเป็นประชากรคนที่ 26 ,1 ,30 ,7 ,12 ,46 ,24 ,6 ,25 ,27 ,18 ,29 ,42 ,15 ,8 ,16 ,23 ,37 ,20 ,4 ,39 ,17 ,21 ,45 ,33 ตามลำดับซึ่งการที่ผมกำหนดค่า prob ในการคัดเลือกนี้เป็นการช่วยลด divergent ของ algorithm ที่จะทำให้ลูกหลานออกมาไม่หลากหลาย โดยทำให้ประชากรที่มีค่า fitness น้อยมีโอกาสถูกเลือกนั่นเองซึ่งค่า prob นั้นควรจะค่อย ๆ เพิ่มขึ้นใน generation ที่มากขึ้นต่อไป

ตอนนี้ในส่วนของการทดลองโค้ดยังทำได้ถึงกระบวนการคัดเลือกครับ

ภาคผนวกโปรแกรม

[175]

```
import numpy as np
import random
import pandas as pd
import math
from random import randint
import matplotlib.pyplot as plt
```

[182]

```
def readfile(file):

    # ----- read data -----
    df = pd.read_csv(file)
    df = df.fillna(method = 'ffill')

    # ----- Create Features -----
    X = df.drop(['id','class'], axis=1).copy(deep=False)
    X = (X-X.min())/(X.max()-X.min())

    # ----- Create Desired outputs -----
    Y = df[['class']].copy(deep=False)

    Input = X.to_numpy()
    Output = Y.to_numpy()

    return Input,Output

Input,Output = readfile('data/wdbc.csv')
```

[183]

```
def cross_val(features,labels,val=90):
    train_features = []
    train_labels = []
    all_samples = len(features)
    train_value = int(all_samples * val/100)
    cross_list = list(range(all_samples))
    random.shuffle(cross_list)
    for i in range(train_value):
        train_features.append(features[cross_list[i]])
        train_labels.append(labels[cross_list[i]])
    train_labels_fix = np.asarray(train_labels)
    train_features_fix = np.asarray(train_features)
    return train_features_fix,train_labels_fix

train_features,train_labels = cross_val(Input,Output)
```

```python
class GA :
    def __init__(self,shape,hidden_shape):#shape =[input_layer, output_la
        shape[1:1] = hidden_shape
        self.shape = shape
        # print(self.shape)
        # initiate weight
        weights = []
        for i in range(len(self.shape)-1):
            w = np.random.uniform(-2,2,(self.shape[i], self.shape[i+1]))
            weights.append(w)
        self.weights = weights

        # initiate weights_t-1
        self.flag = False
        self.weights_last  = np.copy(self.weights)
        self.outputs = None
        self.fitness_value = None

        # initiate activations
        activations = []
        for i in range(len(self.shape)):
            a = np.zeros(self.shape[i])
            activations.append(a)
        self.activations = activations
        # print(self.weights)
        # print(np.shape(self.weights))

    def save_weight(seft):
        return seft.weights

    def use_weight(seft,weight_input):
        self.weight = weight_input
```

```
    def Feed_forward(self, X):
        activations = X
        self.activations[0] = X
        for i, w in enumerate(self.weights):
            # calculate NN_input
            v = np.dot(activations, w)
            activations = self.sigmoid(v)
            self.activations[i+1] = activations
        return activations

    def cut_off_func(self,predict):
        pred = predict.copy()
        for i in range(len(pred)):
            if(pred[i] > 0.5):
                pred[i] = 1
            else:
                pred[i] = 0
        return pred

    def fitness(self,pred,labels):
        true_m = labels.copy()
        Temp = 0
        for i in range(len(labels)):
            if((pred[i][0] - true_m[i][0]) == 0) :
                Temp = Temp + 1
        # print(' ------------------------------------
        # print('Accuracy : '+str(Temp/len(labels)))
        self.fitness_value = (Temp/len(labels))
        return (Temp/len(labels))
        # return 0

    def sigmoid(self, s, deriv=False):
        if (deriv == True):
            return s * (1-s)
        return 1/(1 + np.exp(-s))
```

268]

```
def Tselection(prob,fitness):
    fit = fitness.copy()
    random.shuffle(fit)
    half = len(fit)//2
    select = []
    for i in range(half):
        r = random.randrange(0,1)
        if(fit[i][0] < fit[half+i][0]):
            if(r<prob):
                select.append(fit[i])
            else:
                select.append(fit[half+i])
        else:
            if(r<prob):
                select.append(fit[half+i])
            else:
                select.append(fit[i])
    return select
```

[273]

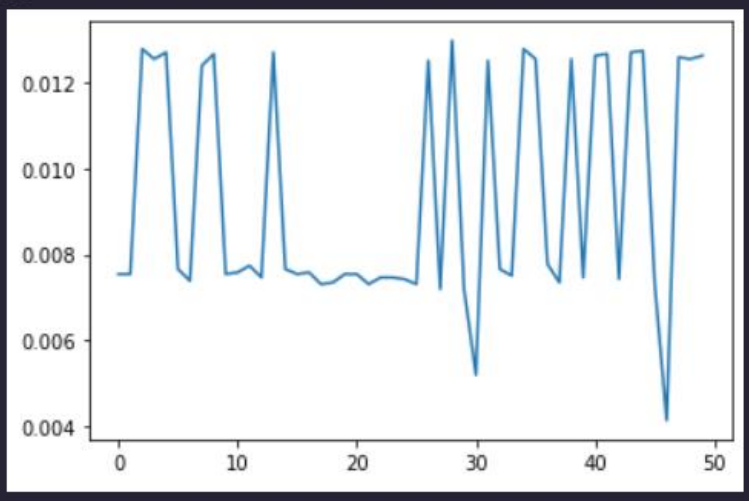
```
fitness = []
population = {}
gen = 50
for i in range(gen):
    ga = GA([30,1],[2,2])
    predict = ga.Feed_forward(train_features)
    pred_fix = ga.cut_off_func(predict)
    fitness.append( ((ga.fitness(pred_fix,train_labels)/gen),i))
    train_features,train_labels = cross_val(Input,Output)
    population["GA"+str(i)] = ga.save_weight()

# print(fitness[0][0])
# print(type(fitness))
select = Tselection(0.4,fitness)
# print(population["GA1"])
for i in range(len(select)):
    print(select[i][1])

fit = []
for i in range(len(fitness)):
    fit.append(fitness[i][0])
# print(fit)
plt.plot(fit, label="fitness")
plt.show()
```

```
plt.show()
```

```
23
37
20
4
39
17
21
45
33
```



[-] M↓