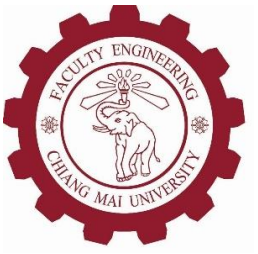


สร้างโปรแกรมสำหรับ Train Multilayer Perceptron โดยใช้ GA

Wisconsin Diagnostic Breast Cancer (WDBC)

โดย

นายศุภกฤต ก๋องคำ

650610858

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 261456

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่1 ปีการศึกษา 2566

มหาวิทยาลัยเชียงใหม่

สร้างโปรแกรมสำหรับ Train Multilayer Perceptron โดยใช้ GA สำหรับ Wisconsin Diagnostic Breast Cancer (WDBC)

ปัญหา :

ให้ทำการทดลองกับ wdbc.data (Wisconsin Diagnostic Breast Cancer (WDBC) จาก UCI Machine learning Repository) โดยที่ data set นี้ มี 2 classes และ 30 features ซึ่งในแต่ละ sample จะมีทั้งหมด 32 ค่าโดยที่

1) ID number

2) Diagnosis (M = malignant, B = benign) à class

3-32) เป็นค่า features ทั้ง 30

ให้ทำการทดลองโดยใช้ 10% cross validation เพื่อทดสอบ validity ของ network ที่ได้ และให้ทำการเปลี่ยนแปลงจำนวน hidden layer และ nodes

หลักการ:

ใช้ GA ในการค้นหาคำตอบที่ดีที่สุดโดยใช้ fitness values จากการคำนวณค่า MSE ที่ได้จาก MLP

โดยจะได้เป็น $1/(0.1+MSE)$

1.ขั้นตอนการสร้างโปรแกรม

1.1) อ่านค่าจากไฟล์ข้อมูล .txt

- แยก 90% ของข้อมูลเพื่อใช้สำหรับ Train
- แยก 10% ของข้อมูลเพื่อใช้สำหรับ Test

```
4
5
6   def data_read(file_path):
7       data=[]
8       with open(file_path,'r') as file:
9           lines = file.readlines()
10          for line in lines:
11              line = line.strip().split(',')
12              data.append(line)
13      return data
14
15
16  source_data ='wdbc.txt'
17  data = data_read(source_data)
18  labels = np.array([1 if entry[1] == 'M' else 0 for entry in data])
19  features = np.array([list(map(float, entry[2:])) for entry in data])
20
21  source_data2 ='wdbc_copy.txt'
22  data2 = data_read(source_data2)
23  labels_test = np.array([1 if entry[1] == 'M' else 0 for entry in data2])
24  features_test = np.array([list(map(float, entry[2:])) for entry in data2])
25
```

1.2) สร้าง โมเดล MLP

- กำหนด weight
- Function forward สำหรับการคำนวณ

```python
class MLP:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.bias_input_hidden = np.zeros(hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_hidden_output = np.zeros(output_size)

    def forward(self, inputs):
        # ส่งออกผลลัพธ์จากชั้นซ่อน
        hidden_input = np.dot(inputs, self.weights_input_hidden) + self.bias_input_hidden
        hidden_output = sigmoid(hidden_input)

        # ส่งออกผลลัพธ์จากชั้นออก
        output_input = np.dot(hidden_output, self.weights_hidden_output) + self.bias_hidden_output
        output = sigmoid(output_input)

        return output

    def set_weights(self, chromosome):
        input_hidden_size = self.input_size * self.hidden_size
        hidden_output_size = self.hidden_size * self.output_size
        self.weights_input_hidden = chromosome[:input_hidden_size].reshape((self.input_size, self.hidden_size))
        self.bias_input_hidden = chromosome[input_hidden_size:input_hidden_size + self.hidden_size]
        self.weights_hidden_output = chromosome[input_hidden_size + self.hidden_size:input_hidden_size + self.hidden_size + hidden_output_size].reshape((s
        self.bias_hidden_output = chromosome[input_hidden_size + self.hidden_size + hidden_output_size:]


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# แปลงคลาส M เป็น 1 และ B เป็น 0
labels = np.array([1 if label == 'M' else 0 for label in labels])
```

1.3) สร้าง function ต่างๆของ Genetic Algorithms

- Fitness function
- Crossover
- Mutation

```python
def mean_squared_error(y_true, y_pred):
    return ((y_true - y_pred) ** 2).mean()



population = [MLP(input_size, hidden_size, output_size) for _ in range(population_size)]

def fitness(mlp,data,target):
    output = mlp.forward(data)
    fitness = 1/(0.01+mean_squared_error(output,target))
    return fitness


def crossover(parent1, parent2):
    crossover_point = np.random.randint(0, parent1.weights_input_hidden.size)  # Use the size of the weight matrix
    child1 = np.concatenate((parent1.weights_input_hidden[:crossover_point], parent2.weights_input_hidden[crossover_point:]))
    # Perform similar concatenation for other attributes
    return child1


def mutate(chromosome, mutation_rate):
    mutation_mask = np.random.rand(len(chromosome)) < mutation_rate
    if mutation_mask.sum() > 0:
        mutation_values = np.random.rand(mutation_mask.sum())
        chromosome[mutation_mask] = mutation_values
    return chromosome
```

2.ทำการทดลอง

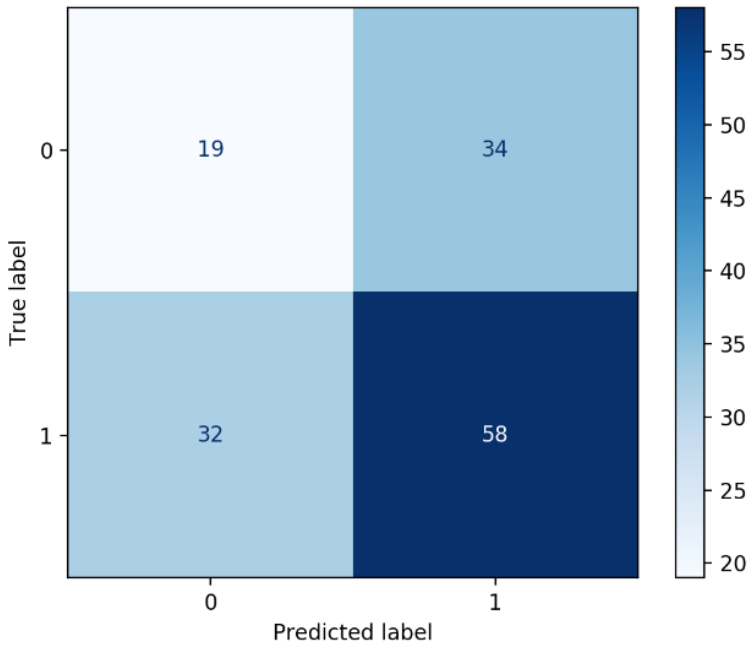
```python
def train(input, target, num_generation):
    global population
    for generation in range(num_generation):
        fitness_values = [fitness(mlp, input, target) for mlp in population]
        parents = np.argsort(fitness_values)[-2:]
        newpop = [population[i] for i in parents]
        while len(newpop) < population_size:
            child = crossover(population[parents[0]], population[parents[1]])
            child = mutate(child, 0.01)
            newpop.append(child)
        population = newpop
        print(f"generation {generation}")
```

```python
def test(input, target):
    mlp = population[np.argmax([fitness(MLP,input,target) for MLP in population])]
    predict = mlp.forward(input)
    predict = np.round(predict)
    pre,tar =[],[]
    for i in predict:
        if i[0] == 1:
            pre.append(1)
        else :
            pre.append(0)
    for i in target:
        if i[0] == 1:
            tar.append(1)
        else :
            tar.append(0)

    pre = np.array(pre)
    pre = np.array(tar)
    tp = np.sum((pre == 1) & (tar == 1))
    tn = np.sum((pre == 0) & (tar == 0))
    fp = np.sum((pre == 1) & (tar == 0))
    fn = np.sum((pre == 0) & (tar == 1))
    confusion_matrix(tp,tn,fp,fn)
    cm = confusion_matrix(tar, pre)
    plt.imshow(cm, cmap=plt.cm.Blues)
    plt.show()

train_ga = train(features,labels,100)
test_ga = test(features_test,labels_test)
```

```python
################################################
# สร้าง MLP
input_size = features.shape[1]
hidden_size = 16
output_size = 1
population_size = 100
#########################################
```



3.วิเคราะห์และสรุปผล

จากการทดลองและสร้างโปรแกรมพบว่า เมื่อใช้หลักการ GA มาช่วยกันกับ MLP สามารถทำให้โมเดลมีประสิทธิภาพและความแม่นยำขึ้นได้ เมื่อทดลองเปลี่ยนค่าต่างๆใน MLP สังเกตได้ว่ามีความเปลี่ยนแปลงน้อยมาก แต่ตัวแปรที่สำคัญที่เปลี่ยนแล้วมีความแตกต่างอย่างเห็นได้ชัดจะอยู่ในส่วนของ GA โดยสรุปแล้วการใช้หลักการทั้ง 2 มาใช้ ทำให้ได้ประสิทธิภาพเพิ่มขึ้น และสามารถปรับปรุงได้ในหลายรูปแบบในโอกาสต่อไป

ภาคผนวก

gann > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix


def data_read(file_path):
    data=[]
    with open(file_path,'r') as file:
        lines = file.readlines()
        for line in lines:
            line = line.strip().split(',')
            data.append(line)
    return data


source_data ='wdbc.txt'
data = data_read(source_data)
labels = np.array([1 if entry[1] == 'M' else 0 for entry in data])
features = np.array([list(map(float, entry[2:])) for entry in data])

source_data2 ='wdbc_copy.txt'
data2 = data_read(source_data2)
labels_test = np.array([1 if entry[1] == 'M' else 0 for entry in data2])
features_test = np.array([list(map(float, entry[2:])) for entry in data2])

##########################################################################
```

```python
######################################################################
# สร้างคลาส MLP
class MLP:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.bias_input_hidden = np.zeros(hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_hidden_output = np.zeros(output_size)

    def forward(self, inputs):
        # ส่งออกผลลัพธ์จากชั้นซ่อน
        hidden_input = np.dot(inputs, self.weights_input_hidden) + self.bias_input_hidden
        hidden_output = sigmoid(hidden_input)

        # ส่งออกผลลัพธ์จากชั้นออก
        output_input = np.dot(hidden_output, self.weights_hidden_output) + self.bias_hidden_output
        output = sigmoid(output_input)

        return output

    def set_weights(self, chromosome):
        input_hidden_size = self.input_size * self.hidden_size
        hidden_output_size = self.hidden_size * self.output_size
        self.weights_input_hidden = chromosome[:input_hidden_size].reshape((self.input_size, self.hidden_size))
        self.bias_input_hidden = chromosome[input_hidden_size:input_hidden_size + self.hidden_size]
        self.weights_hidden_output = chromosome[input_hidden_size + self.hidden_size:input_hidden_size + self.hidden_size + hidden_output_size].reshape((se
        self.bias_hidden_output = chromosome[input_hidden_size + self.hidden_size + hidden_output_size:]


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# แปลงคลาส M เป็น 1 และ B เป็น 0
labels = np.array([1 if label == 'M' else 0 for label in labels])

################################################
```

```python
################################################
# สร้าง MLP
input_size = features.shape[1]
hidden_size = 5
output_size = 1
population_size = 100
##########################################

# ฟังก์ชันคำนวณค่าความผิดพลาด (MSE)
def mean_squared_error(y_true, y_pred):
    return ((y_true - y_pred) ** 2).mean()



population = [MLP(input_size, hidden_size, output_size) for _ in range(population_size)]

def fitness(mlp,data,target):
    output = mlp.forward(data)
    fitness = 1/(0.01+mean_squared_error(output,target))
    return fitness


def crossover(parent1, parent2):
    crossover_point = np.random.randint(0, parent1.weights_input_hidden.size)  # Use the size of the weight matrix
    child1 = np.concatenate((parent1.weights_input_hidden[:crossover_point], parent2.weights_input_hidden[crossover_point:]))
    # Perform similar concatenation for other attributes
    return child1


def mutate(chromosome, mutation_rate):
    mutation_mask = np.random.rand(len(chromosome)) < mutation_rate
    if mutation_mask.sum() > 0:
        mutation_values = np.random.rand(mutation_mask.sum())
        chromosome[mutation_mask] = mutation_values
    return chromosome
```

```python
def train(input, target, num_generation):
    global population
    for generation in range(num_generation):
        fitness_values = [fitness(mlp, input, target) for mlp in population]
        parents = np.argsort(fitness_values)[-2:]
        newpop = [population[i] for i in parents]
        while len(newpop) < population_size:
            child = crossover(population[parents[0]], population[parents[1]])
            child = mutate(child, 0.01)
            newpop.append(child)
        population = newpop
        print(f"generation {generation}")

```

```python
def test(input, target):
    mlp = population[np.argmax([fitness(MLP,input,target) for MLP in population])]
    predict = mlp.forward(input)
    predict = np.round(predict)
    pre,tar =[],[]
    for i in predict:
        if i[0] == 1:
            pre.append(1)
        else :
            pre.append(0)
    for i in target:
        if i[0] == 1:
            tar.append(1)
        else :
            tar.append(0)

    pre = np.array(pre)
    pre = np.array(tar)
    tp = np.sum((pre == 1) & (tar == 1))
    tn = np.sum((pre == 0) & (tar == 0))
    fp = np.sum((pre == 1) & (tar == 0))
    fn = np.sum((pre == 0) & (tar == 1))
    confusion_matrix(tp,tn,fp,fn)
    cm = confusion_matrix(tar, pre)
    plt.imshow(cm, cmap=plt.cm.Blues)
    plt.show()

train_ga = train(features,labels,100)
test_ga = test(features_test,labels_test)
```