باسمه تعالی

دانشگاه صنعتی شریف

دانشکده مهندسی برق

درس هوش محاسباتی

تمرین سری ششم

علی محرابیان96102331

استاد: دکتر رضایی

تابستان 1399

در این تمرین از کتابخانه deap در پایتون برای پیاده سازی خواسته های مسئله استفاده می کنیم.این کتابخانه به منظور سهولت برای genetic programming طراحی شده است.از آن جایی که در الگوریتم های فرا ابتکاری تضمین رسیدن به جواب وجود ندارد،نزدیک ترین جواب های خود را به خواسته سوال گزارش می کنیم.ابتدا به شرح قسمت های مختلف کد می پردازیم.

در ابتدا مجموعه عملگرها و مجموعه ورودی ها را تعریف می کنیم.توجه داریم که ورودی و خروجی های ما از نوع Boolean هستند.در درخت خود دارای 3 ورودی و 1 خروجی هستیم.

```
pset = gp.PrimitiveSetTyped("main",(bool,bool,bool),(bool))
pset.addPrimitive(operator.and_,(bool,bool),bool)
pset.addPrimitive(operator.or_,(bool,bool),bool)
pset.addPrimitive(operator.not_,[bool],bool)
pset.renameArguments(ARG0="x1")
pset.renameArguments(ARG1="x2")
pset.renameArguments(ARG2="x3")
```

در قسمت بعد به ساخت individual های خواسته شده می پردازیم.دو درخت به شیوه grow و دو درخت به شیوه Full که هرکدام طول بیشینه برابر با 2 دارند،تولید می کنیم.

```
toolbox = base.Toolbox()
creator.create("FitnessMin", base.Fitness, weights=(-1.0,-1))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin,pset=pset)
toolbox.register("expr", gp.genFull, pset=pset, min_=1, max_=2)
toolbox.register("individual", tools.initIterate, creator.Individual,toolbox.expr)

t1_ful= toolbox.individual()
t2_ful=toolbox.individual()
```

در قسمت بعد تابع fitness را به این صورت تعریف می کنیم که خروجی متناظر با 8 ورودی ممکن محاسبه کرده و نسبت خروجی های صحیح به کل خروجی ها را در نظر می گیریم.

```
def fitness(a,b,c,d):
    tree=[a,b,c,d]
    tx=[[0,0,0],[1,0,0],[1,1,0],[1,0,1],[1,1,1],[0,1,1],[0,1,0],[0,0,1]]
    ty=[0,1,0,1,1,1,0,0]
    score=[]
    for i in range(4):

        f=gp.compile(tree[i],pset)
        yy=[]

        for j in range(8):
            yy.append(f(tx[j][0],tx[j][1],tx[j][2]))

        score.append((tree[i],len([m for m, n in zip(yy,ty) if m == n])/8))

    return score
```

سپس با چند مرحله تکرار مشخص،دو درختی که بیشترین fitness را داشتند استخراج کرده و crossover آن ها را که دو درخت جدید هستند،حساب می کنیم.در محاسبه crossover هم با احتمال رندوم،از نقاطی که ترمینال و نقاط پایانی ما هستند و یا نقاط میانی هستند،انتخاب کرده و آن ها را جا به جا می کنیم.

```
for u in range(100):

    q=fitness(a1,a2,a3,a4)
    q.sort(key=takeSecond)
    a1=q[3][0]
    a2=q[2][0]
    (a3,a4)=gp.cxOnePointLeafBiased(a1,a2,random.uniform(0,1))
```

در قسمت نهایی،درخت های اولیه و خروجی هایی که تا حد خوبی به جواب نهایی نزدیک بودند را گزارش می کنیم.

```
and_(and_(x3, x2), or_(x1, x2))
```

```
and_(and_(x2, x3), not_(x1))
```

```
and_(not_(x1), or_(x2, x2))
```

```
not_(and_(x2, x1))
```

دو درخت سمت چپ از نوع full و درختان سمت راست از نوع grow هستند.خروجی نهایی درخت زیر است.

```
and_(x1, not_(x2))
```

درختان زیر،حالت اولیه های دیگری هستند که برای رسیدن به جواب استفاده کردیم.

```
or_(or_(x2, x3), not_(x1))
```

```
and_(and_(x2, x2), and_(x1, x2))
```

```
and_(not_(x2), and_(x1, x1))
```

```
and_(or_(x2, x1), and_(x3, x2))
```

خروجی نهایی به صورت زیر است.

```
and_(x2, x3)
```