به نام خدا

پروژه اول هوش مصنوعی و سیستم های خبره

استاد : دکتر آرش عبدی

هانیه اسعدی

99521055

پاییز 1401

تعداد متغیرهای تابع را از کاربر ورودی میگیریم. (تعداد متغیرهای ساپورت شده 3 میباشد.)

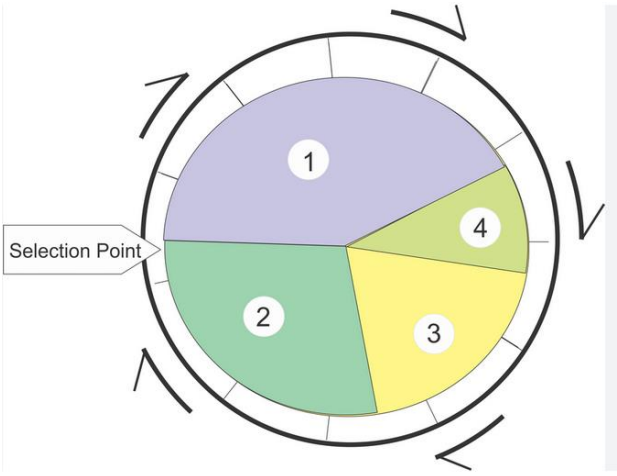
یک تابع را به عنوان تابع black box در نظر میگیریم. سپس مقادیر ورودی و خروجی را برای تابع به دست آورده و در یک دیکشنری ذخیره میکنیم.

نسل صفرم، توابعی رندوم با طول رندوم هستند که با استفاده از آنها به ادامه نسل ها میپردازیم. برای ساخت نسل صفر، جمعیت اولیه را خود تعیین کرده و به اندازه آن تابع میسازیم و در دیکشنری نسل ها ذخیره میکنیم.

پس از به وجود آمدن جمعیت اولیه، مقدار شایستگی هر تابع را بر اساس خطای مقدار واقعی (که همان مقادیر تابع black box است) و مقداری که تابع رندوم بدست می آورد، تخصیص میدهیم. این مقدار شایستگی MSE است که از فرمول زیر بدست می آید:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

برای انتخاب والدین برای تولید فرزند یا همان نسل های بعدی باید از الگوریتم انتخابی استفاده کنیم که شانس انتخاب شدن والد برعکس مقدار شایستگی آن باشد. منظور: هر چه MSE تابعی کمتر باشد، یعنی آن تابع با تابع black box تفاوت کمتری دارد و به تابع اصلی نزدیک تر است پس شانس بیشتری برای تولید فرزند باید داشته باشد.



الگوریتم Roulette wheel را انتخاب میکنیم. این الگوریتم مانند گردانه ها عمل میکنند و هر چه احتمال تابعی بیشتر باشد، قطاع بیشتری به آن اختصاص میدهد. برای بدست آوردن مقدار مساحت اختصاص یافته شده به هر تابع، ابتدا جمع معکوس همه شایستگی ها را بدست می آوریم. سپس حاصل تقسیم معکوس هر شایستگی بر جمع شایستگی ها، آن مساحت را نشان میدهد.

برای انتخاب والد باید دو والد از دو تابع مختلف انتخاب کنیم. پس دوبار عددی رندوم بین صفر و یک انتخاب میکنیم و با توجه به آن تابع های والد را انتخاب میکنیم. برای تولید فرزند، از هر والد، یک نقطه انتخاب کرده و از آن نقطه برای شکستن تابع والد استفاده میکنیم. تکه اول تابع والد اول را به تکه دوم تابع والد دوم میچسبانیم. همینطور تکه دوم تابع والد دوم را به تکه دوم تابع اول وصل میکنیم. ضمن اینکه هر نسل، به تعداد دلخواهی که تعیین میکنیم از نسل قبل بیشتر است. یک جهش هم داریم که بصورت رندوم اتفاق میفتد. اگر مقدار عدد رندومی که برای جهش انتخاب کرده ایم کمتر از 0.0005 باشد، جهش رخ میدهد و تیکه دوم تابع والد اول به تکه دوم تابع والد دوم نمیچسبد که در واقع جهش در نسل رخ داده است.

همه این کارها را برای 20 نسل انجام میدهیم تا تابع با بهترین شایستگی خروجی دهد.

اگر تابعی با شایستگی کمتر از 20 هنگام ساختن نسل ها پیدا کرد، همان تابع را برگردانده و ادامه نمیدهد. اما اگر تا 20 نسل، تابعی با شایستگی کمتر از 10 پیدا نکرد، بهترین تابع با بهترین شایستگی در نسل های قبل را به عنوان خروجی میدهد.

توجه شود که هنگام اختصاص مساحت در تابع roulette چون معکوس اعداد را در نظر گرفته ایم، و چون اعداد را گرد کرده ایم، ممکن است به خطای تقسیم بر صفر بخوریم و چون توابع و شایستگی ها کاملا رندوم هستند، نمیتوان از این دسته خطاها جلوگیری کرد. پس در صورتی که حین اجرای برنامه به این ارور برخوردید، چندین بار دیگر برنامه را اجرا کنید تا به جواب برسید. همچنین ممکن است تا 20 نسل نتوانیم تابع مناسبی پیدا کنیم و مجبور به خروج از برنامه شویم. در این صورت نیز چند بار دیگر اجرا کنید تا جواب بدهد.

برای جمعیت اولیه 100 :

(1

```
Enter number of variables you want: 1
Function is : 12*x + 5*x - 2*x + 10
In Generation 4, I found the equation : 15*x+8 and the Best fitness over Generati
ons equals to : 4.0
Total time : 2.077 seconds
```

(2

```
Enter number of variables you want: 2
Function is : x + 10*y - 4*x
In Generation 6, I found the equation : 15+x/9-x-2+x/19-x-19+y+9*y+18/x/10 and th
e Best fitness over Generations equals to : 7.00813
Total time : 2.534 seconds
```

مقدار خروجی دو تابع برای دو مقدار x=1 و x=2 و y=1 و y=2:

```
>>> eval("x + 10*y - 4*x",{"x":1, "y":1})
7
>>> eval("11*y-y-x*3/14-x*3",{"x":1, "y":1})
6.7857142857142865
>>> eval("x + 10*y - 4*x",{"x":2, "y":2})
14
>>> eval("11*y-y-x*3/14-x*3",{"x":2, "y":2})
13.571428571428573
>>>
```

(3

```
Enter number of variables you want: 3
Function is : x -y + 2*z - 4
In Generation 2, I found the equation : 5+x+6-y-12+z and the Best fitness over Generations equals
to : 1.5
Total time : 2.303 seconds
```

مقدار خروجی دو تابع برای دو مقدار x=1 و x=2 و y=1 و y=2 و z=1 و z=2:

```
>>> eval("x -y + 2*z - 4",{"x":1, "y":1, "z":1})
-2
>>> eval("5+x+6-y-12+z",{"x":1, "y":1, "z":1})
0
>>> eval("x -y + 2*z - 4",{"x":2, "y":2, "z":2})
0
>>> eval("5+x+6-y-12+z",{"x":2, "y":2, "z":2})
1
>>>
```

(4

```
Function is : 10*x+3+2*x+8
In Generation 8, I found the equation : 15/4+4+x*6+x*6+4 and the Best fitness ove
r Generations equals to : 0.5625

Total time : 1.096 seconds
```

ساده شده تابع : 12x +11.75

در اجرای دیگری برای همین تابع، با تعداد نسل های بیشتر، جواب دیگری میگیریم که درست است که شبیه نیست اما به ازای ورودی یکسان با تابع اصلی، مقدار خروجی به مقدار خروجی تابع اصلی ما خیلی نزدیک است.

```
Function is : 10*x+3+2*x+8
In Generation 13, I found the equation : 29-x+13*x-13*10/31-10/31-14 and the Best
 fitness over Generations equals to : 0.2704

Total time : 8.976 seconds
```

مقدار خروجی دو تابع برای دو مقدار x=1 و x=2 :

```
>>> eval(" 10*x+3+2*x+8",{"x":1})
23
>>> eval("29-x+13*x-13*10/31-10/31-14",{"x":1})
22.483870967741936
>>> eval(" 10*x+3+2*x+8",{"x":2})
35
>>> eval("29-x+13*x-13*10/31-10/31-14",{"x":2})
34.483870967741936
>>>
```

(5

```
Function is : 12*x-2*x+5*x
In Generation 5, I found the equation : 2/x-x-4/83+51*x/3-x and the Best fitness
over Generations equals to : 0.05822

Total time : 1.529 seconds
```

مقدار خروجی دو تابع برای دو مقدار x=1 و x=2 :

```
>>> eval("12*x-2*x+5*x",{"x":1})
15
>>> eval("12*x-2*x+5*x",{"x":2})
30
>>> eval("2/x-x-4/83+51*x/3-x",{"x":1})
16.951807228915662
>>> eval("2/x-x-4/83+51*x/3-x",{"x":2})
30.951807228915662
>>>
```

(6

```
Enter number of variables you want: 1
Function is : x**2 + 15
In Generation 4, I found the equation : 8+x*x+8 and the Best fitness over Generations equals to :
1.0
Total time : 1.345 seconds
```

اجرای دیگر برای همین تابع

```
Enter number of variables you want: 1
Function is : x**2 + 15
In Generation 8, I found the equation : 11/4+4+x/x*4+x/1*x+4 and the Best fitness over Generations
 equals to : 0.0625
Total time : 2.764 seconds
```

مقدار خروجی دو تابع برای دو مقدار x=1 و x=2 :

```
>>> eval("x**2 + 15",{"x":1})
16
>>> eval("11/4+4+x/x*4+x/1*x+4",{"x":1})
15.75
>>> eval("x**2 + 15",{"x":2})
19
>>> eval("11/4+4+x/x*4+x/1*x+4",{"x":2})
18.75
>>>
```

اختلاف دو تابع در آزمایش های انجام شده قابل چشم پوشی است. پس نتیجه میگیریم برنامه با دقت تقریبا خوبی به تولید تابع بر اساس ورودی ها و خروجی ها می پردازد.