بسمه تعالی

دانشکده مهندسی کامپیوتر

مبانی هوش محاسباتی

نام استاد: دکتر مزینی

تمرین پنجم

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

آذر ۱۴۰۰

پاسخ سوال اول

(۱.۱)

f به معنی fitness است:

$f(x_1) = 6+5-4-1+3+5-3-2 = 9$

$f(x_2) = 8+7-1-2+6+6-0-1 = 23$

$f(x_3) = 2+3-9-2+1+2-8-5 = -16$

$f(x_4) = 4+1-8-5+2+0-9-4 = -19$

پس به ترتیب fit بودن از راست به چپ، x_2 ، x_1 ، x_3 و x_4 هستند. این مقادیر نشان می دهند که مطلوب ترین ژن جمعیت برای مثلا، x_2 است و بعد x_1 و ...

(۲.۱)

- دو تای fit ترین، x_1 و x_2 هستند. با middle-point، روش one-point

x_1: 6541 | 3532 ⟹ 65416601

x_2: 8712 | 6601 ⟹ 87123532

} اعضای جدید

- با x_1 و x_3، با نقطه های داده شده و به روش two-point:

x_1: 65 | 4135 | 32 ⟹ 65 9212 32

x_3: 23 | 9212 | 85 ⟹ 23413585

} اعضای جدید

- یعنی با x_2 و x_3 به روش uniform: $(p(c) = 50\%)$

x_2: 8712 6601 (نصف بیشتا)

x_3: 23921285 (عوض می شوند)

⟹ 83921681

⟹ 27126205

} اعضای جدید

(۳.۱

دیدیم که ۶ عضو جدید داریم. با تابع f = fitness ، برای همه آن‌ها

محاسبه می‌کنیم:

$f(65416601) = 17$

$f(87123532) = 15$

$f(65921232) = -2$

$f(23413585) = -5$

$f(83921681) = -2$

$f(27126205) = 9$

پس مجموع fitness نسل جدید برابر 32 است. مجموع fitness نسل قبلی هم با توجه به قسمت 1.1، برابر 7 است. می‌بینیم که به طور میانگین $\frac{32}{6} > \frac{7}{4}$، یعنی نسل جدید به طور میانگین به وضوح fitness بیشتری دارند. این تکامل و پیشرفت جمعیت را نشان می‌دهد.

پاسخ سوال دوم

این سوال را برای x و y های بین ۰ و ۶۴ حل میکنیم. چون می دانیم که $y = 59-2x^2$ همواره عددی کمتر یا مساوی ۵۹ به دست می آید. البته ممکن است معادله جواب هایی به ازای y منفی و x های بزرگتر از ۶۴ داشته باشد اما فعلا در این سوال جواب های صحیح این بازه را در نظر گرفته ایم.

پس هر کدام از کروموزوم های موجود در جمعیت را به صورت یک tuple از (x,y) در نظر گرفته ایم. که x و y هردو اعدادی باینری ۶ بیتی (از ۰ تا ۶۳) هستند.

هایپرپارامترهای حل مسئله را اینطور در نظر گرفته ایم که اعداد مناسبی باشند و زود به جواب برسیم:

```
number_of_initialize_population=10,
binary_length=6,
max_population=20,
mutation_probability=0.1
```

تابع fitness: قدر مطلق $2x^2 + y - 59$، پس هرچقدر مقدار کمتری داشته باشد زوج (x,y) آن کروموزوم بهتر به جواب نزدیک بوده اند.

شرط پایان: پیدا کردن کروموزومی با fitness=0. (جوابی برای معادله)

در ابتدا با ۱۰ موجود در جمعیت شروع میکنیم. و آن ها به همراه fitness هایشان را ذخیره میکنیم.

کمترین fitness که نشاندهنده fit ترین عضو است را پیدا میکنیم. سپس دومین عضو fit را پیدا میکنیم.

X های مربوط به این دو عضو را به عنوان والد به two_point_crossover میدهیم که به صورت رندوم بخشی از آن ها را با هم جابجا می کند و ۲ فرزند جدید را برمیگرداند. همین کار را با y ها هم انجام می دهیم. در نتیجه از ترکیب x و y های جدید، ۴ فرزند به جمعیت اضافه می شود. البته پس از اضافه کردن این ۴ فرزند، fitness های آن ها را نیز اضافه میکنیم.

جهش هم با تابع mutation مدل کرده ایم و در اینجا با احتمال ۰.۱، یکی از ژن ها را به طور رندوم برعکس کرده ایم. (اگر صفر بود آن را یک میکنیم و بالعکس)

این کار را نسل به نسل انجام میدهیم. البته ماکسیمم جمعیت را برابر ۲۰ گرفته ایم و هرگاه از آن بیشتر می شود اعضای اولیه را از بین میبریم.

این سوال چندین جواب در این بازه دارد که با هر بار اجرا میبینیم یکی از این جواب ها را میابد. چون در ابتدا رندوم است نمیدانیم به کدام می رسد، اما میدانیم که در تعداد نسل نه چندان زیاد همواره به یک جواب میرسد.

```
result found after 57 generations:
x , y = 000101 , 001001
which means x= 5 , y= 9
```

پیاده سازی و توضیحات ریز توابع در نوتبوک، قسمت Q2 پیوست شده است.

پاسخ سوال سوم

پیاده سازی و توضیحات ریز توابع در نوتبوک، قسمت Q3 پیوست شده است.

این سوال راا شبیه به سوال قبلی پیاده سازی میکنیم. تفاوت ها در کروموزوم ها و تابع fitness است. به جای زوج x,y، فقط هر کروموزوم تکی یک جواب مساله است. هر کروموزوم را عددی باینری ۸ بیتی در نظر میگیریم که اگر بیت i ام ۱ باشد، به معنی انتخاب شدن آیتم شماره i ام توسط دزد است.

تابع fitness: ۱ تقسیم بر مجموع ارزش های برداشته شده توسط دزد، به شرط آن که مجموع وزن ها از ظرفیت یعنی ۲۵ کمتر یا مساوی باشد. و خود ۱ هم برای حالتی که بیش از ظرفیت برداشته باشیم، یا این که مجموع ارزش برداشته شده برابر ۰ باشد!

در نتیجه هر چقدر این عدد کمتر باشد، یعنی مجموع ارزش بیشتر بوده و جواب fit تری داریم.

سوال را با این داده ها:

```
VALUES = [30, 10, 20, 50, 70, 15, 40, 25]
WEIGHTS = [2, 4, 1, 3, 5, 1, 7, 4]
NAMES = ["zomorod", "noqre", "yaqut", "almas", "berellian", "firuze", "aqiq", "ka
hroba"]
```

و این هایپرپارامتر ها:

```
number_of_initialize_population=5,
binary_length=8,
max_population=10,
mutation_probability=0.1,
generations=1000
```

two_point_crossover و mutation و مقداردهی رندوم اولیه مشابه سوال قبل هستند. جمعیت اولیه و حداکثر جمعیت را کمی کمتر گرفته ایم. چون تفاوتی که وجود دارد این است که در هر مرحله که دو تای fit تر را پیدا میکنیم و از آن ها ۲ فرزند جدید متولد می شوند (نه ۴ تا مانند سوال قبل).

در این مساله شرط پایان را تعداد نسل ها (iterationها) قرار دادیم که با ۱۰۰۰ نسل تقریبا مطمئنیم به بهترین جواب میرسیم. پاسخی که با اجرا به آن میرسیم به این صورت بود:

```
best chromosome: 10111111
 -----------------
zomorod
yaqut
almas
berellian
firuze
aqiq
kahroba
```

که اگر خودمان هم مساله را حل کنیم میبینیم بهترین جواب همین است.

پاسخ سوال چهارم

۱.٤)

میدانیم که در الگوریتم ACO، مورچه ها از خود فرمون بر جای میگذارند و فرمون تازه تر احتمال دنباله رویی بیشتری دارد. در این مثال فرض کرده ایم که مورچه ها از خانه به سمت غذا از دو مسیر متفاوت حرکت میکنند. سپس با چند iteration همزمان حرکت میکنند و مورچه ای که مسیر کوتاه تر دارد به مقصد می رسد.

و مورچه با مسیر کوتاه تر، با منبع غذایی به لانه برمی گردد. به یاد داشته باشید که هر مورچه ردی از فرمون ها را روی زمین بر جای می گذارد. بنابراین وقتی اولین مورچه با غذا به لانه می‌رسد، مورچه دیگر مسیر فرمونی آن را دنبال می‌کنند تا به منبع غذا برسد. همچنین فرمون مسیر طولانی تر بیشتر از مسیر کوتاه تبخیر شده است. چون مسیر بلندتر زمان بیشتری را میگیرد و فرومون آن بیشتر تبخیر میشود لذا مسیر برگشت او هم از مسیر کوتاه خواهد بود. اگر مورچه های بیشتری داشته باشیم هم این فرمون مسیر کوتاه، بیشتر تقویت می شود و عملا مسیر بلند از الگوریتم کنار می رود.

۲.٤)

ابتدا باید هر individual را شامل ژن هایی برابر وزن های شبکه بگیریم. یعنی individualهای ما باید به طول ۱۸ (6*2+6) باشند. پس بردار مکان و سرعت هر کدام ۱۸ رقمی هستند.

ابتدا جمعیتی را مثلا ۲۰۰ جواب مختلف را به صورت رندوم مقداردهی میکنیم. منظور از جواب individual هایی با بردار مکان ۱۸ تایی و بردار سرعت ۱۸ تایی رندوم است.

حالا به ازای هرکدام از جواب ها، طبق فرمول های MLP، مرحله forward tracking را انجام می دهیم. یعنی وزن ها را از بردار مکان هر individual برمیداریم، بعد با رابطه relu(w*x) برای لایه میانی و تایع فعالسازی sigmoid برای لایه نهایی، این مسئله regression را حل میکنیم. در نهایت مقدار mean squared error را به ازای این مقادیر از وزن محاسبه میکنیم. در واقع میزان خطای ما، همان تابع fitness است. که اگر خطا کمتر باشد درواقع fitتر بوده است.

حالا به جای این که از روش back propagation برای آپدیت وزن ها استفاده کنیم، با استفاده از PSO این کار را انجام می دهیم. باید سرعت را با این فرمول آپدیت کنیم:

$$v(k+1) = [w * v(k)] + [c1 * random * (Pbest – x(k))] + [c2 * random * (Gbest – x(k))]$$

pbest = بهترین نتیجه ای که این ذره به آن رسیده است. (بهترین وزن های آن در طول اجرای الگوریتم)

Gbest = بهترین نتیجه ای که کل ذرات تا این لحظه به آن رسیده اند.

w = ضریبی که تعیین میکند چه قدر از سرعت قبلی حفظ شود. (هایپرپارامتر)

c1 و c2 = ثابت هایی که هایپرپارامتر های مسئله هستند و میزان تاثیر بردار gbest و pbest را تعیین میکنند.

و پس از آپدیت کردن سرعت در هر مرحله، باید خود x ذرات را هم آپدیت کنیم. (x ذرات = مقادیر وزن ها برای این ذره). که فرمول آن واضح است:

$$x(k+1) = x(k) + v(k+1)$$

این روند را تا جایی ادامه میدهیم تا به یک Gbest خوب برسیم که خطای آن (مقدار fitness) از مقداری که تعیین میکنیم کمتر باشد.

۳.٤)

پیاده سازی این بخش به همراه توضیحات کد داخل نوتبوک موجود است.

همان الگوریتم توضیح داده شده در بخش قبل را پیاده سازی کرده ام. شبکه mlp همواره با ۶ نورون در تنها لایه میانی و یک نورون در لایه خروجی است. و برای هرکدام از ۴ تابع زیر داده های آموزشی و تست را با اعدادی صحیح بین −۱۰۰۰۰ و ۱۰۰۰۰ به تعداد ۱۰۰۰۰ عدد برای آموزش و ۱۰۰ عدد برای تست ایجاد کردم. سپس با هایپرپارامترهای نسبتا مناسب w=0.5 و c1=3 و c2=1، چون میخواهیم تاثیر pbest بیشتر از gbest باشد و در نتیجه تمام ذرات زود به بهترین نقطه پیدا شده همگرا نمی شوند و بهتر جستجو میکنند، برنامه اجرا کردم. بستگی به پیچیدگی تابع، تعداد ذرات اولیه و تعداد iteration های اجراشدن الگوریتم را تغییر می دهیم. همچنین در تابع اول فقط یک ورودی داریم و لذا تعداد وزن ها ۱۲ تاست، در سه مورد بعدی ۲ ورودی داریم و تعداد وزن ها ۱۸ است. در نهایت برای هر کدام از بخش ها ابتدا شبکه را آموزش میدهیم و سپس روی دیتای train و test نتایج را به دست آورده ام.

- Y=x:

```
accuracy on train: 100.0 %
accuracy on test: 100.0 %
```

- Y = x1+x2:

```
accuracy on train: 100.0 %
accuracy on test: 100.0 %
```

- Y = sin(x1) + cos(x2): در اینجا چون اعداد اعشاری با دق ۸ رقم بعد از اعشار هستند، تقریبا هیچ جا خروجی شبکه برابر مقدار واقعی به دست نمی آید. پس کم بودن تابع mean squared error را به عنوان دقت لحاظ میکنیم.

```
the numbers are float here so we can't take accuracy as a good method because the network can't predict the exact value!
mean square error on train: 0.9956675467425754
mean square error on test: 1.027357302671627
```

- Y = x1 * x2: این شبکه برای فیت شدن روی این تابع به اندازه کافی قدرمتند نیست.

```
accuracy on train: 1.0 %
accuracy on test: 10.0 %
```