تمرین سری 6 درس هوش محاسباتی

بکتاش انصاری

99521082

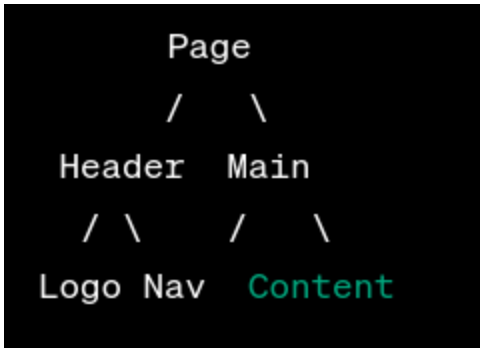
سوال ۱)

در این سوال فرآیند کلی به این شکل است که ابتدا جمعیت رندوم اولیه را بوجود بیارویم و چون در حال استفاده از GP هستیم. کروموزوم های ما شامل یک سری درخت میباشند. برای جمعیت اولیه نیز میتوانیم به صورت رندوم تا یک حد مشخصی از عمق درخت (Dmax) درخت‌هایی را درست کنیم. که در عمق آخر مشخص شده، برگ های درخت قرار می‌گیرند. که ما دو ست مختلف داریم به نام های function set و terminal set که نودهایی که در عمق آخر قرار دارند (برگ‌ها) از مجموعه terminal set و نودهای میانی از مجموعه function set انتخاب میشوند.

حال مجموعه‌های F و T چه میتوانند باشند؟

ساختار درخت میتواند به این شکل باشد که چون میخواهیم ساختار یک webpage را بوجود آوریم. Page ما شامل بخش‌های کلی مانند Page میباشد که این بخش را نودهای بالایی درخت تشکیل میدهند. نودهای میانی درخت میتوانند بخش های مربوط به html سایت یعنی مثلا html tag ها باشند مانند: div header body و ....

بخش‌های انتهایی و terminal ها هم میتوانند عکس‌ها و متون یا content هر tag باشند. مانند شکل کلی زیر:



حال بقیه فرآیند الگوریتم با استفاده از fitness function و ساخت جمعیت جدید انجام میشود. که ساخت درخت‌های جدید میتواند از روش‌های crossover یا mutation باشد.

ب)

بخش اصلی و مهم و سخت این مسئله، تعریف کردن fitness function اصولی و کامل است به طوری که بتواند دو یا چند دیزاین(درخت) مختلف از یک صفحه را با هم مقایسه کند و به هر کدام یک امتیازی بین 0 و 1 به آن‌ها بدهد به طوری که درختی که امتیاز بالاتری دارد از نظر اصول دیزاین و html css ، جلوه‌ی بهتری داشته باشد.

برای تعریف این fitness function میتوانیم ابتدا از یک سری اصول اولیه شروع کنیم که شرح زیر است:

- تعداد تگ‌های تو در تو چقدر است
- متناسب با میزان تو رفتگی تگ ، متون تگ نیز size متناسبی داشته باشند
- رنگ‌های هر تگ و همینطور رنگ background با هم خوانایی داشته باشند.
- برای خوانایی رنگ میتوانیم یک سری ترکیب خوب و بد را از قبل برای تابع مورد نظر تعریف کنیم.

- اندازه و فرمت container ها و همینطور button ها متناسب با هم باشند تا از گوناگونی زیاد جلوگیری شود
- باید چک شود که آیا دیزاین انجام شده قابلیت responsive بودن دارد یا خیر
- از انیمیشن‌ها در سایت استفاده شده است یا همچی بصورت ایستا است.
- آیا دیزاین dynamic است؟
- آیا اصول فرم‌نویسی رعایت شده است؟
- آیا فرم دارای دکمه submit در انتهای تمام فیلدها است یا خیر
- آیا هر input field دارای constraint هایی برای ورودی است یا خیر
- آیا متون تگ‌های داخلی از تگ‌های خارجی بیرون زده اند؟ یا درون آنها جای گرفته‌اند.
- آیا header قبل از تمام تگ‌ها و footer بعد از تمام تگ‌ها قرار دارد یا خیر.

حال میتوانیم در هر کدام از این قوانین که در تابع fitness چک میشود قوانین ریز تری نیز قرار دهیم.
در نهایت به هر کدام از این قوانین یک وزنی مثبت یا منفی‌ای قرار میدهیم و با هم جمع میکنیم و نرمالایز میکنیم که یک عدد بین صفر و یک به بدهد.
و از این معیار برای ساخت جمعیت بعدی استفاده میکنیم.

پ)
برای این مثال داده شده موارد بالا را چک میکنیم که مهم‌ترین آن‌ها عبارت‌اند از:

- آیا اصول فرم‌نویسی رعایت شده است؟
- آیا فرم دارای دکمه submit در انتهای تمام فیلدها است یا خیر

- آیا هر input field دارای constraint هایی برای ورودی است یا خیر
- آیا متون تگ‌های داخلی از تگ‌های خارجی بیرون زده اند؟ یا درون آنها جای گرفته‌اند.
- آیا هر input field دارای constraint هایی برای ورودی است یا خیر

ابتدا یک سری جامعه اولیه (درخت های اولیه) تولید میکنیم و مقدار fitness فانکشن را برای آن‌ها چک میکنیم. سپس از روی کروموزوم هایی که مقدار fitness آنها بالاتر شده است جامعه جدید را میسازیم و این کار را تکرار میکنیم.

سوال ۲)

در ابتدا توضیح میدهم که چگونه پیاده سازی را انجام داده‌ام:

در این نوع سوال تابع fitness ما دقیقا خود چندجمله‌ای مورد نظر است و هر چه مقدار این fitness کمتر باشد یعنی ورودی تابع عدد ایده‌آل تری بوده. برای آنکه مقادیر منفی رانیز در نظر بگیریم **تابع fitness را مقدار قدر مطلق خروجی تابع مورد نظر تعریف میکنیم.**

```
# Define the fitness function
CodiumAI: Options | Test this function
def fitness(root):
    # print("root: ",root)
    # Compute the value of the polynomial for this root
    y = sum(c * root**i for i, c in enumerate(coefficients))
    # print("value: ", y)

    # The fitness is the absolute value of y (since we want to minimize y)
    return abs(y)
```

برای ورودی گرفتن تابع تنها ضرایب را به ترتیب از چپ به راست ورودی میگیریم و آن‌را در لیستی به نام coefficients ذخیره میکنیم.

برای مثال برای این معادله داریم:

$4x^3 + 5x^2 - 6x + 1 \rightarrow$ coefficients = [4, 5, -6, 1]

حال هر ژنوم یا کروموزوم اعدادی را قرار میدهیم که به عنوان ورودی به تابع میدهیم.

که در اینجا من فرض کردم این اعداد در بازه -10 تا 10 هستند و چون معادلات داده شده ریشه‌هایی در این بازه دارند. (کد زده شده بر روی تمامی بازه ها جوابگو است).

و جامعه اولیه را به صورت رندوم از بین این دو عدد تولید میکنیم. که من population size را 200 در نظر گرفتم.

```
# Define the population size and the range of the roots
pop_size = 200
root_range = (-10, 10)

# Initialize the population with random values
population = [random.uniform(*root_range) for _ in range(pop_size)]
```

حال الگوریتم ژنتیک را تا 300 تکرار اجرا میکنم.

در ابتدا fitness را برای تمام جامعه محاسبه میکنم و آن‌ها را بر اساس این مقدار سورت میکنم و 30 درصد بالایی جامعه (جامعه‌ای که fitness کمتری دارند) را جدا میکنم.

```
# Evaluate the fitness of the population
fitnessValues = []
for root in population:
    fitValue = fitness(root)
    fitnessValues.append((root, fitValue))

# Create a new population
new_population = []

# sort the chromozones based on their fitness values
sorted_tuples = sorted(fitnessValues, key=lambda x: x[1])

# select the 30 percent of the top chromozones
threshold_index = int(len(sorted_tuples) * 0.3)
children = [t[0] for t in sorted_tuples[:threshold_index]]
```

حال جامعه جدید ما این 30 درصد جامعه قبلی خواهد شد.

حال بر روی این جامعه فرآیند crossover و mutation را اجرا میکنم.

برای crossover برای 10 بار دو مقدار از ژنوم های این جامعه را به صورت تصادفی انتخاب میکنم و **میانگین آن‌ها را محاسبه میکنم**. دلیل آنکه 10 بار اینکار را کردم و با احتمال اینکار را نکردم این بود که چون هر بار دارم 30 درصد بالایی جامعه را انتخاب میکنم اگر با احتمال جلو بروم بعد از چند نسل دیگر ژنومی باقی نمیماند.

```
def crossOver(population):
    new_population = []
    for i in range(10):
        samples = random.sample(population, 2)
        new_population.append((samples[0] + samples[1]) / 2)
    return population + new_population
```

برای mutation نیز 10 بار یک ژنوم را به طور تصادفی انتخاب میکنم و آن را با یک عدد رندوم و توزیع شده بین -1 و +1 جمع میکنم و دوباره به جامعه آن را اضافه میکنم.

```python
def mutation(population):
    new_population = []
    for i in range(10):
        samples = random.sample(population, 1)
        new_population.append(samples[0] + random.uniform(-1, 1))
    return population + new_population
```

این فرآیند را 300 بار تکرار میکنم و در نهایت از جامعه نهایی ژنومی که بهترین خروجی را دارد خروجی میدهم.

خروجی کد برای مثال های زده شده:

```
baktash@baktash:~/term7/Computational Intelligence/HW6$ /bin/py
Enter the Coefficients of your polynomial equation: 2 -4
Best solution: x = 1.9999999124169285
baktash@baktash:~/term7/Computational Intelligence/HW6$ /bin/py
Enter the Coefficients of your polynomial equation: 1 -8 4
Best solution: x = 7.46405694043592
baktash@baktash:~/term7/Computational Intelligence/HW6$ /bin/py
Enter the Coefficients of your polynomial equation: 4 -5 1 -1
Best solution: x = 1.2136962215156302
```

```
Best solution: x = 1.2136962215156302
baktash@baktash:~/term7/Computational Intelligence/HW6$ /bin/python3 "/hom
Enter the Coefficients of your polynomial equation: 186 -7.22 15.5 -13.2
Best solution: x = 0.3585110308831946
baktash@baktash:~/term7/Computational Intelligence/HW6$
```

سوال ۳)

میتوانیم این سوال را اینگونه در نظر بگیریم که چون در ابتدا تنها برای ما این مهم است که تعداد اعداد زوج و فرد در هر سطر و ستون برابر باشد. میتوانیم تمام اعداد زوج را بربر با عدد +1 و تمام اعداد فرد را برابر با -1 در نظر بگیریم. اینگونه اگر مقدار اعداد فرد و زوج داخل یک ستون را با هم جمع کنیم اگر مقدار بدست آمده 0 شود یعنی تعداد فردها با زوج‌ها برابر بوده است. اگر مثبت شود تعداد زوج‌ها بیشتر و اگر منفی شود تعداد فردها بیشتر بوده است. یعنی در کل ما 18 عدد -1 و 18 عدد +1 خواهیم داشت و مسئله ژنتیک را ابتدا اینگونه حل میکنیم.

در نتیجه ساختار ژنوم ما یک ماتریس 6*6 خواهد بود که مقادیر آن -1 و +1 هستند.

برای شروع گام‌ها داریم:

- ابتدا به صورت رندوم این 36 عدد -1 و +1 را درون این 36 خانه قرار میدهیم.
- برای تابع fitness دو شرط را باید رعایت کنیم و متناسب به آن‌ها یک وزن مناسبی قرار دهیم که عبارت‌اند از:
    - 1. جمع تمام سطرها باید برابر 0 و جمع تمام ستون‌ها باید برابر 0 شود
    - 2. حال جمع تمام سطر و ستون‌ها را با هم جمع میکنیم. هر چه این عدد نزدیک به صفر باشد بهتر است.
    - 3. حال باید چک کنیم که آیا تعداد اعداد فرد و زوج در کل برابر است یا نه (یعنی در هر کروموزوم ما 18 تا +1 و 18 تا -1 داریم یا خیر) که یک عدد نیز به این مقدار بین صفر و یک میدهیم
    - 4. حال یک برآورد کلی از این دو قانون در میاوریم و یک عدد بین صفر و یک میسازیم به طوری که هر چه به یک نزدیک تر باشد ایده‌آل تر است.

- حال با استفاده از fitness ساخته شده الگوریتم را شروع میکنیم و crossover و mutation را روی آن پیاده‌سازی میکنیم. میتوانیم احتمال crossover را 0.3 و احتمال mutation را 0.05 درنظر بگیریم.
- شرط پایان را نیز مقدار fitness یک در نظر میگیریم.
- حال با اتمام الگوریتم یک ماتریس 6*6 داریم که مقدار اعداد زوج و فرد در هر سطر و ستون برابر است و در کل نیز تعداد آنها برابر است.
- حال اعداد 1 تا 36 را طبق علاقه خودمان در خانه‌های مربوطه قرار میدهیم به طوری که اعداد فرد در خانه‌هایی با مقدار -1 و برعکس قرار گیرند.

سوال۴)

S = 2+1 = 3

برای ساده‌سازی فرض میکنم که بعد دوم هر تصویر وجود ندارد. و فرض میکنیم 5 تصویر موجود در یک خط قرار دارند. میتوانیم یک رابطه ریاضی تعریف کنیم که این ۵ عکس را به یک بعد کاهش دهیم.

حال با توجه به فرضی که کردیم مقدار X و V برابر میشود با:

$X = [x_1, x_2, x_3, x_4, x_p]$

$V = [v_1, v_2, v_3, v_4, v_p]$

حال برای X مقادیر رندومی بین 1 تا 5 ایجاد میکنیم و برای v نیز مقادیری بین -5 تا 5 در نظر میگیریم. نکته مهم اینجا است که نباید اجازه دهیم particle مورد نظر از این مقادیر X خارج شود.

حال برای تابع fitness باید طوری تعریف کنیم که اگر به خانه‌ای که شبیه s=3 است نزدیک شد. مقدار آن بیشینه شود. برای این کار کافی است ماترین خانه‌ای که رو آن است را با این ماتریس زیر خانه به خانه ضرب کند و جمع کند: (با توجه به شماره دانشجویی باید حالت بهینه حالت سوم شود)

1	1	1	1
1	0	0	0
1	0	0	0
1	0	0	0

با اینکار ماتریسی که کاملا به این عدد شبیه باشد مقدار فیتنس آن 7 میشود.

حال الگوریتم را اجرا میکنیم. به این شکل که ابتدا یک تعداد particle درون خط تعریف میکنیم و مقادیر آن‌ها را بین 1 تا 5 مقداردهی میکنیم. (مثلا 20 تا particle) حال برای سرعت آن‌ها نیز مقادیری بین -5 تا +5 در نظر میگیریم.

حال برای چندین دور (مثلا 100 دور) موقعیت هر particle را آپدیت میکنیم. داریم:

v(k+1) = w * v(k)+ c1 * random1() * (PBest - x(k)) + c2 * random2() * (GBest - x(k))

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

که مقادیر c1 و c2 و w را میتوانیم برای مثال 1 , 2 , 0.7 تعریف کنیم.

و مقدار GBest را هم باید طبق PBest آپدیت کنیم.

$PBest > GBest \rightarrow GBest = PBest$