تمرین ششم درس هوش محاسباتی

98522328

سوال 1)

الف) توضیحاتی که از کاربر گرفته می شود، می تواند شامل موارد مختلف مانند فونت، رنگ بندی ، محتوای سایت ، نحوه چیدمان سایت باشد.

نحوه ی چیدمان درخت می تواند به این شکل باشد که هر نود یا هر گره ، هر عنصر مجزا در صفحه وب است. هر نود از این درخت، خود شامل ویژگی های مختلف است. می تواند به دو دسته نوع و ویژگی تبدیل شود. (درواقع خود این ویژگی ها می تواند یک subtreeبرای هر عنصر باشد. برای مثال این ویژگی ها خود می تواند شامل رنگ، فونت و غبره باشد.

برای مثال :

Each node or subtree : a specific element 1- type (depends on the usage) , 2- properties (color, font, name , content , ..)

در واقع درخت ساخته شده می تواند برگرفته از نحوه جیدمان در html و cssباشد.

ب) برای تعریف fitness function ، باید معیار های مورد نظر را برای ژنوم در نظر داشته باشیم. مواردی که می تواند حائز اهمیت باشد ، می تواند شامل فونت، نحوه ی کنار هم چیدن اجزا (مثل بررسی نوع تشکیل درخت) ، انتخاب رنگ و فونت برای هر عنصر بسته به type مورد نظر باشد. برای مثال می توان تمام propery های هر عنصر درخت را (color , font , name) یک عدد خاص نسبت داد و به نسبت هر تایپ، (برای مثال تقسیم) یک عدد نسبی برای هر نود از درخت به دست اورد. و سپس با بررسی کردن نحوه ی اتصال هر کدام از element ها و عدد هر کدام(در ساده ترین حالت جمع همه ی المنت ها) . و(یا جمع کردن هر کدام از شاخه ها تا زمانی که به برگ برسد)، می توان یک شاخص برای تمیزی یا مورد قبول بودن هر درخت مورد نظر در نظر گرفت.

پ)

برای مثال :

Node 1 :

: (type = “Form”, Properties: “name : register_form”, method: “post”)

Childerns of Node 1 :

Node 2: (type : “input”, properties: “ name= Name, font:=Area, color=black,align=rtl”)

Node 3: (type : “input”, properties: “ name= Last name, font:=Area, color=black, align = rtl ,fontsize = 14”)

Node 4: (type : “button”, properties: “ name= submit, font:=Area, color=yellow, fontsize = 14 , ”)

اختصاص دارد عدد 2 inputدر حالت ساده برای مثال فرم عدد 1 و

Node 1 : (method : 1) => 1/1 = 1

Node 2 : (color rate : 5 , rtl : 2 ,font : 1 , = > (8/2) = 4

Node 3 : 4

Node 4 : bottun : 2 , color : 3 , font : 1 => 4/2 : 2

The simplest way is to sum up each nodes and devide them by each node type value) :

(1 + 2 + 4 + 4) / 1+2+2+1 = 9/6 = 3/2 = 1.5

می توان این عدد را به یک نسبت ماکس حالت خواسته شده مقایسه کرد

سوال 2)

ایده ی اصلی این سوال با کمک الگوریتم ژنتیک حل می شود. به این صورت که هر چند جمله ای polynomial به عنوان فیتنس فانکشن در نظر گرفته می شود. در صورتی که قدر مطلق جواب به صفر نزدیک تر باشد آن جواب جواب بهتری است. کروموروم ما در این سوال هر کدام از نقطه ها در یک بازه ی مشخص است. برای هر چند جمله ای یک جمعیت انتخاب می شود. در هر مرحله از انتخاب کروموزوم های جدید، 20 درصد از کروموزوم های بهتر انتخاب و باقی به صورت رندوم انتخاب می شوند. حداکثر تعداد دور طی شده 1000 تا است و زمانی که فیتنس بهترین نتیجه از 0.01 کم تر باشد، به عنوان جواب اصلی انتخاب می شود.

توضیح کد :

1- Fitness function : همان طور که در توضیح گفته شد، فیتنس فانکشن برای هر معادله خود معادله ی اصلی است.

```
def fitness(equation, variables):
    x = variables[0]
    result = eval(equation)
    return abs(result)
```

2- initial population : جمعیت اولیه را میسازد که هر کدام شامل ژنوم های مختلف است. در اینجا بین بازه ی عدد -20 تا +20 این بازه بررسی می شودبه صورتی که popsize سایز هر population و gen_numتعداد ژنوم ها است.

```
def generate_population(pop_size, num_genes):
    population = np.random.uniform(-20, 20, size=(pop_size, num_genes))
    return population
```

3- Select parents: تابع select_parents هر individual را ارزیابی می کند، آنها را بر اساس fitnessمرتب می کند_ 20٪ برتر را انتخاب می کند و سپس 80٪ باقی مانده از والدین را به طور تصادفی برای نسل بعدی انتخاب می کند. این حرکت درواقع نوع selection را توضیح میدهد.

```
Def select_parents(population, equation):
    fitness_values = np.array([fitness(equation, individual) for individual in
population])
    sorted_indices = np.argsort(fitness_values)
    sorted_population = population[sorted_indices]
    # Select top 20%
    top_20_percent = int(0.3 * len(sorted_population))
    selected_parents = sorted_population[:top_20_percent]
    # Randomly select the remaining 80%
    remaining_parents_indices = np.random.choice(np.arange(top_20_percent,
len(sorted_population)), size=len(sorted_population) - top_20_percent,
replace=False)
    remaining_parents = sorted_population[remaining_parents_indices]
    return np.concatenate((selected_parents, remaining_parents))
```

4- Cross over and mutation : برای این روش، ابتدا در هنگام cross overیک نقطه را انتخاب می کنیم و دو وارد را ترکیب می کنیم.

برای جهش، یک mutation_mask با تولید مقادیر تصادفی برای هر ژن ایجاد می شود. اگر مقدار تصادفی کمتر از mutation_rate باشد، ژن مربوطه دچار جهش می شود.

برای ژن هایی که نیاز به جهش دارند (جایی که mutation_mask True است)، مقادیر تصادفی جدیدی تخصیص داده می شود.

```
def crossover_mutation(parents, mutation_rate):
    crossover_point = np.random.randint(0, len(parents[0]))
    child = np.concatenate((parents[0][:crossover_point],
parents[1][crossover_point:]))


    mutation_mask = np.random.rand(len(child)) < mutation_rate
    child[mutation_mask] = np.random.uniform(-20, 20, size=np.sum(mutation_mask))

    return child
```

5- اجرای مراحل الگوریتم ژنتیک :

```
6- def genetic_algorithm(equation, x, pop_size=100, num_genes=1,
   max_generations=1000, mutation_rate=0.1, elitism_rate=0.1):
7-     population = generate_population(pop_size, num_genes)
8-     for generation in range(max_generations):
9-         parents = select_parents(population, equation)
10-        child = crossover_mutation(parents, mutation_rate)
11-        population[-1] = child
12-        num_elites = int(elitism_rate*pop_size)#preserve top individuals
13-        elites = population[:num_elites].copy()
14-        # Generate new individuals to fill the rest of the population
15-       new_individuals =generate_population(pop_size-
   num_elites,num_genes)
16-        population[:num_elites] = elites
17-        population[num_elites:] = new_individuals
18-        sorted_indices = np.argsort([fitness(equation, ind) for ind in
   population]) #sort
19-        population = population[sorted_indices]
20-        if generation % 50 == 0:
21-            best_fitness = fitness(equation, population[0])
22-            print(f"Generation {generation}: Best fitness =
   {best_fitness}")
23-            if best_fitness < 0.001:
24-                print("Solution found!")
25-                break
26-    return population[0] # Return the best individual
```

این تابع بخش اصلی الگوریتم است. به طور کلی، این تابع ابتدا جمعیت اصلی را تشکیل میدهد. مرحله ی selection ، mutation cross over را بر روی داده ها عملی می کند، جمعیت را بر اساس بهترین individualها مرتب می کند و این عمل را تا زمانی که 1 – تعداد دور ها کم تر از 1000 باشد و یا در نسلی که فیتنس آن محاسبه می شود، بهترین جواب ها وجود داشته باشد. در نتیجه بهتربن جواب را برمبگرداند. یعنی جوابی که در بین جمعیت بهتر از همه فیت شده باشد، یکی از ریشه های چند جمله ای ما است.
پاسخ ها به ازای مثال های زیر:

```
equations = [
    "2*x - 4",
    "x**2 - 8*x + 4",
    "4*x**3 - 5*x**2 + x - 1",
    "186*x**3 - 7.22*x**2 + 15.5*x - 13.2"
]

for equation in equations:
    x_result = genetic_algorithm(equation, x=0)
    print(f"\nEquation: {equation}")
    print(f"Roots: x = {x_result[0]}")
    print(f"Fitness: {fitness(equation, x_result)}")
    print("="*30)
```

معادله اول و دوم:

```
Generation 0: Best fitness = 0.7847804977355111
Generation 50: Best fitness = 0.005419581475763 38
Generation 100: Best fitness = 0.00013836227250862976
Solution found!

Equation: 2*x - 4
Roots: x = 1.9999308188637457
Fitness: 0.00013836227250862976
==============================
Generation 0: Best fitness = 2.004054678607517
Generation 50: Best fitness = 0.006930622084586169
Generation 100: Best fitness = 0.005086340120278976
Generation 150: Best fitness = 0.005086340120278976
Generation 200: Best fitness = 0.005086340120278976
Generation 250: Best fitness = 0.0005742306872065228
Solution found!

Equation: x**2 - 8*x + 4
Roots: x = 7.46401873108573
Fitness: 0.0005742306872065228
```

معادله ی سوم:

```
Generation 0: Best fitness = 0.4369069680923534
Generation 50: Best fitness = 0.01784733613856826
Generation 100: Best fitness = 0.010587685168339078
Generation 150: Best fitness = 0.001266267308719371
Generation 200: Best fitness = 0.001266267308719371
Generation 250: Best fitness = 0.001266267308719371
Generation 300: Best fitness = 0.001266267308719371
Generation 350: Best fitness = 0.001266267308719371
Generation 400: Best fitness = 0.001266267308719371
Generation 450: Best fitness = 0.001266267308719371
Generation 500: Best fitness = 0.001266267308719371
Generation 550: Best fitness = 0.001266267308719371
Generation 600: Best fitness = 0.001266267308719371
Generation 650: Best fitness = 0.001266267308719371
Generation 700: Best fitness = 0.001266267308719371
Generation 750: Best fitness = 0.001266267308719371
Generation 800: Best fitness = 0.001266267308719371
Generation 850: Best fitness = 0.001266267308719371
Generation 900: Best fitness = 0.001266267308719371
Generation 950: Best fitness = 4.1066306838644095e-05
Solution found!

Equation: 4*x**3 - 5*x**2 + x - 1
Roots: x = 1.213735243062814
Fitness: 4.1066306838644095e-05
```

معادله آخر

```
Generation 0: Best fitness = 11.78722450729035
Generation 50: Best fitness = 0.7151128110517178
Generation 100: Best fitness = 0.7151128110517178
Generation 150: Best fitness = 0.1628122509633858
Generation 200: Best fitness = 0.1628122509633858
Generation 250: Best fitness = 0.1628122509633858
Generation 300: Best fitness = 0.11478844804088695
Generation 350: Best fitness = 0.11478844804088695
Generation 400: Best fitness = 0.10225346165341875
Generation 450: Best fitness = 0.10225346165341875
Generation 500: Best fitness = 0.04581646986763843
Generation 550: Best fitness = 0.04581646986763843
Generation 600: Best fitness = 0.04581646986763843
Generation 650: Best fitness = 0.04581646986763843
Generation 700: Best fitness = 0.04581646986763843
Generation 750: Best fitness = 0.04581646986763843
Generation 800: Best fitness = 0.03486716959874414
Generation 850: Best fitness = 0.03486716959874414
Generation 900: Best fitness = 0.03486716959874414
Generation 950: Best fitness = 0.03486716959874414

Equation: 186*x**3 - 7.22*x**2 + 15.5*x - 13.2
Roots: x = 0.3589390426603565
Fitness: 0.03486716959874414
==============================
```

همان طور که مشخص است، الگوریتم توانسته است یکی از ریشه های معادله را برای هر معادله با تقریب خوبی بیابد.

سوال سوم)

1- انتخاب کروموزوم: انتخاب کروموزوم در این سوال، یک رشته ی 36تایی است که هر کدام از اعداد ها بیانگر یکی از خانه های مربع به صورت (flatten) هستند. برای مثال عدد های اول تا ششم ردیف اول و عدد های هفتم تا 12 هم بیانگر اعداد ردیف دوم هستند و هر کدام از اعداد بین 1-36 هستند.

2- انتخاب فیتنس فانکشن : در این سوال فیتنس فانکشن، تعداد برابر بودن اعداد زوج و فرد هر سطر و هر ستون را بررسی می کند. به این صورت که برابر بودن اعداد زوج و فرد در تمام سطر ها و تمام ستون ها را بررسی می کند. و اختلاف اعداد زوج و فرد را در تمام ردیف ها و در تمام ستون ها را محاسبه می کند. در حالتی که عدد به 0 زدیک تر باشد، مربع ما به حالت خواسته شده نزدیک تر است. یعنی فیتنس فانکشن برای هر کدام از ردیف ها (ایندکس خانه های 1-6 و 7-12 و 13-18 19-24و 25-30و 31-36 اختلاف اعدادزوج و فرد را بررسی و با هم جمع می کند و. همچنین هر ستون را جدا بررسی می کند (,(..,2,8), (...,1,7,13) (...و همانند ردیف ها اختلاف زوج و فرد بودن را در هر ستون بررسی می کند و برای هر ستون را جمع می کند. تمامی مقادیر ستونی و ردیفی با یک دیگر نیزز جمع می شوند. هر چه مقدار این فانکشن کم تر باشد، تابع ما فیت تر است و حالت صفر حالت خواسته شده برای ماست.. در واقع هر چقدر مقدار تابع ما به صفر نزدیک تر باشد به جواب نزدیک تر است.
(یا در حالت حساب کردن ماکسیمم می توان حالت ماکس اختلاف را از صفر حساب کرد تا حالت مکمل بیشترین حالت را در نظر بگیریم.)

3- روش cross-over: برای کراس اور باید به صورتی باشد که مربع هایی با ترتیب اعداد جدید تولید بشوند.
یکی از این روش می تواند به این صورت باشد که با تقسیم مربع به 4 مربع با 9 خانه، جای مربع ها را به صورت قطری با یک دیگر جا به جا کرد. یعنی مربع اول پرنت اول با مربع پرنت دوم جایگزین شود و اعداد به همان ترتیبی که عوض شده اند، به صورت دیگری با اعدادی که جایگرین شود، جای آن ها عوض شود. (درواقع جایگشت متفاوتی از اعداد در خانه های مختلف داریم))
در میوتیشن نیز جای مربع ها در هر کروموزوم می تواند با خودش عوض یشود.
نوع دیگر می تواند به این صورت باشد که هر ردیف با ردیف دیگر بین هر والد با والد دیگر جابه جا شود. برای مثال سه ردیف اول از اعداد والد اول باشد و سه ردیف دوم از اعدادی که از مربع اول است حایگزین شود و در صورتی که اعداد از سه ردیف اول تکرارب بود آن ها را طبق حایگشت پیش آمده بررسی کند. در این جا یکی از پارامتر ها mutation rate است که باید نیز یک عدد مناسب انتخاب شود تا هر از چند گاهی حهش ایجاد شود. (یا این جهش نیز می تواند برا یمثال جا به جا شدن خانه اول و اخر یک قطر از این مستطیل باشد.)

4- Selection: می توان برای هر دور و در هر سلکشن ، 20 درصد حالت برتر را انتخاب و 80 درصد کروموزوم های دیگر رندوم بااشد.

5- روش الگوریتم کلی: ابتدا جمعیت اولیه را تعیین می کنیم. باید چند کروموزوم 36 تایی از اعداد 1-36 ایجاد کنیم. یعنی یک ارایه 36 تایی از اعداد 1-36. (برای تبدیل به حالت باینری کردن می توان گفت 36 عدد باینری با طول 6 و حداکثر مقدار 100100) . برای مثال، به طور کلی ما !36 است. جمعیت اولیه ما شامل به طور مثال 36 حالت از جمعیت اولیه است.
سپس به ترتیب عملیات کراس اور، میوتیشن را روی آن ها انجام میدهیم و فیتنس را براای آن ها بررسی می کنیم.
در هر حالت و برای هر نسل 20 درصد برتر را انتخاب و 80 درصد را از جمعیت قبلی و همچنین جمعیت جدید تشکیل میدهیم.
این عملیات را با تعداد دور مشخص انجام میدهیم و در صورتی که در یکی از جایگشت ها fitness صفر شود، مربع ما پیدا می شود و در غیر این صورت باید به اندازه تعداد ایپاک مشخص ادامه دهد.

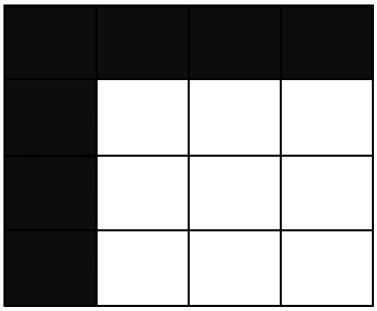
سوال 4)

حالت تشخیص : عکس سوم :

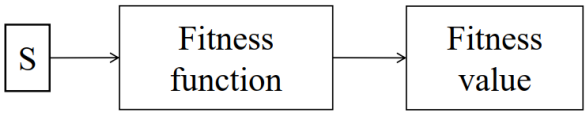
می دانیم 5 عکس داریم و هر عکس شامل یک پترن خاص است. فیتنس فانکشن را می توانیم پترن عکس خاص بررسی کنیم. از 5 مورچه داریم و هر کدام از مورچه ها برای هر تصویر، فیتنس را بر اساس نزدیک بودن به فیتنس فانکشن بررسی می کنند.

به عبارتی :

Fitness_function = x1+x2+x3+x4+x5+x9+x13



- Swarm: a set of particles (S)
- Particle: a potential solution
  - Position: $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,n}) \in \Re^n$
  - Velocity: $\mathbf{v}_i = (v_{i,1}, v_{i,2}, ..., v_{i,n}) \in \Re^n$
- Each particle maintains
  - Individual best position (PBest)
- Swarm maintains its global best (GBest)



حالت اول) مشخص کردن سرعت اولیه: بیت (-vmax – vmax) و برای هر عکس متفاوت یک مورچه.

$$\mathbf{v}_i(k+1) = \text{Inertia} + \text{cognitive} + \text{social}$$

$$\mathbf{v}_i(k+1) = \omega \times \mathbf{v}_i(k) + c_1 \times random_1() \times (PBest_i - \mathbf{x}_i(k)) + c_2 \times random_2() \times (GBest - \mathbf{x}_i(k))$$

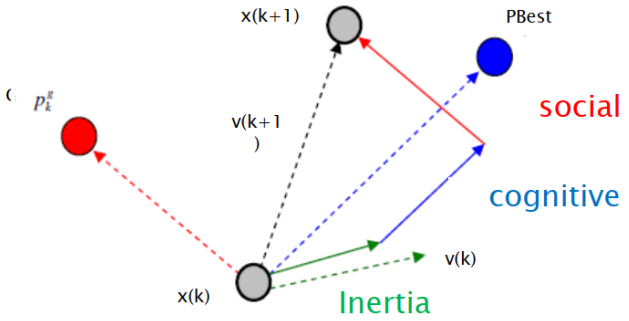
- w, c_1, c_2: Constant
- $random_1()$, $random_2()$: random variable

- Position update

$$\mathbf{x}_i(k+1) = \mathbf{x}_i(k) + \mathbf{v}_i(k+1)$$

- Particle's velocity

$$\mathbf{v}_i(k+1) = \text{Inertia} + \text{cognitive} + \text{social}$$



در هر مرحله ، پوزیشن و سرعت را update می کنیم. هر موچه یک نقطه ی اپتیمم در بررسی کردن هر خانه دارد و همچنین یک global optimum وجود دارد. حال برای این که هر مورچه را در هر مرحله اپدیت کنیم، باید موقعیت بعدی مورچه و سرعت آن را با کمک فرمول بالا تعریف کنیم. همچنین مقدار w, c1,c2 مقادیری هستند که به دست ما تعیین می شوند و بسته به انتخاب ما سرعت کم یا زیاد می شود. در صورت کم بودن سرعت نقطه ماکسیمم با سرعت کم ترو با دقت بیشتری پیدا می شود و انتخاب مقادیر مناسب بسته به حالت های مناسب دارد.

این عملیات را انقدر این مورچه ها ادامه میدهند تا بهترین فیتنس ممکن در بهترین نقظه موجود (که عکس ها یما هستند) پیدا می شود. در صورت وجود نویز، نزدیک ترین خانه ی ممکن انتخاب می شود.

این حرکت در چند راوند مختلف انجام می شود و در نقطه ای که بیشتر مورچه ها در آن نقظه جمع می شوند. نزدیک ترین عکس ممکن به نتیجه حاصل است. ماکس فیتنس فانکشن ما نیز 5 است.