

به نام خدا

ایمان علیپور

۹۸۱۰۲۰۲۴

پروژه ۱ هوش مصنوعی

پیاده سازی الگوریتم ژنتیک

استاد: دکتر آرش عبدی هجراندوست

مفروضات من:

- اولاً فرض کردم \sin و \cos فقط برگ های درخت میتوانند باشند. (چرا که اگر نود میانی باشند مشکلاتی در اضافه کردن نود جدید و خالی بودن یکی از نود های راست و چپ به وجود می آید)
- فرض کردم تابع تک متغیره است. (در واقع n متغیره بودن کمی مشکل ایجاد میکرد که تلاش کردم اول مسئله را ساده کنم و بعد اگر شد پیچیدگی ها را بیشتر کنم.)
- متأسفانه دیر متوجه شدم که وقتی عملگر توان چندین بار پشت سر هم می آید یا توان یک عدد مختلط می شود الگوریتم من دچار مشکل میشود (برای محاسبات) و بنابراین این فرض را کردم که چنین اتفاقاتی نباید بیفتد و برای بسیاری از رخداد ها مقدار بینهایت برگرداندم.

مشکلات راه:

- بسیاری از وقت ها والدین تکراری برای تولید نسل بعدی انتخاب می شوند که من این شرط را گذاشتم که این اتفاق نیفتد اما بخش انتخاب والدین بسیار بسیار زمان بر شد.
- داشتن چندین متغیر مسئله را سخت میکند و من ایده ای برای حل این مشکل نداشتم.
- انتخاب والدین به صورتی که مورد های تکراری زیاد نباشند بسیار زمان بر بود و در بسیاری از موارد در این مرحله کد من زمان بسیاری میبرد و هر فکری کردم زمان را نتوانستم بهبود ببخشم.
- در بسیاری از مواقع بخاطر تولید تصادفی جمعیت اولیه الگوریتم به زمان زیاد و iteration های زیاد نیاز دارد تا بتواند تابعی نزدیک به تابع خوب را پیدا کند، برای همین من یک سقف برای تعداد iteration ها گذاشتم که بتوانم گزارشم را بنویسم.
- در مواقعی که بوسیله زاد و ولد یا crossover والدین، فرزند خوبی تولید نشود و با جهش هم فایده ای حاصل نشود، الگوریتم جواب بهینه را پیدا نمیکند اما اگر زمان کافی بدهیم ممکن است این اتفاق بیفتد.
- یکی از اشتباهات من این بود که بجای اینکه والدین بهینه را نگه دارم، آن را احتمالاتی کردم تا بقیه والدین هم امکان ماندن را داشته باشند اما حس میکنم شاید یکی از دلایل نتایج نه چندان خوب همین بوده.
- در کل زمان اجرای این پیاده سازی بسیار زیاد است و به کمی hyper parameter tuning نیاز دارد که با اینکه من آزمایش های بسیار زیادی کردم، نتوانستم حالت بهینه ای بیابم.
- یکی دیگر از مشکلات دخیل بودن زیاد شانس برای الگوریتم برای یافتن جواب است که شاید اگر کمی ذکاوت به خرج میدادم نتایج بهتری حاصل میشد، اما تعداد پارامتر های مجهول و متغیر مسئله به حدی زیاد بود که مجبور شدم به نتیجه فعلی بسنده کنم.

پیاده سازی:

نگاشت مسئله به درخت:

برای این کار فرض کردم هر عبارت به صورت یک درخت ذخیره شده است، حال این درخت را با استفاده از تابعی مقدارش را در نقطه x محاسبه میتوان کرد، همچنین پیمایش درخت به صورت inorder به ما عبارت ریاضی را میدهد. حال کافی است تعدادی درخت اولیه ایجاد کنیم و هر بار آنها را با هم ترکیب کنیم و بعضی وقت ها یک جهش در یکی از فرزندان ایجاد کنیم.

```
45 def inorder(node):
46     if node:
47         print('(', end = '')
48         inorder(node.left)
49         print(node.data, end = '')
50         inorder(node.right)
51         print(')', end = '')
```

```
34 class Node:
35     def __init__(self, data):
36         self.left = None
37         self.right = None
38         self.data = data
39
```

```

75 def generate_random_leaf_node():
76     probability = generate_random_number(1, 100)
77     if probability <= 70:
78         return generate_random_number(0, 10)
79     elif probability <= 90:
80         return 'x'
81     elif probability <= 95:
82         return 'sin(x)'
83     else:
84         return 'cos(x)'

```

```

62 def generate_random_inner_node():
63     probability = generate_random_number(1, 100)
64     if probability <= 20:
65         return '+'
66     elif probability <= 40:
67         return '-'
68     elif probability <= 60:
69         return '*'
70     elif probability <= 80:
71         return '/'
72     else:
73         return '^'
74

```

ساخت جمعیت اولیه:

برای این کار، از یک تابع استفاده کردم که ابتدا بصورت رندوم ارتفاع درخت را بدست می‌آورد، سپس با اساین کردن احتمال برابر برای اپراتور های مختلف، داده نودهای میانی را یک اپراتور، و داده نود های برگ را یک عدد ثابت، x یا \sin یا \cos قرار دادم که بیشترین احتمال برای عدد ثابت رندوم است و پس از آن x و احتمال اضافه شدن \sin و \cos را هم ۵ درصد قرار دادم.

حال به تعداد دلخواه با این تابع می‌توان جمعیت دلخواه ساخت و در ادامه هم همین کار را کردم، توجه کنید شاید ساخت هوشمندانه تر جمعیت اولیه باعث بهبود عملکرد الگوریتم میشد که حس میکنم شاید من تا انتها به آن زیاد فکر نکردم که میتوان این جمعیت اولیه را هوشمندانه تر تولید کرد، مثلا ترکیبات توان های مختلف x و ضرب ضرایب مختلف در \sin و \cos و ...

```

86 def create_initial_generation(size_of_generation, max_tree_length = 5):
87     initial_generation = []
88
89     for i in range(size_of_generation):
90         length_of_tree = generate_random_number(1, max_tree_length)
91         tree_leaves = []
92         root = Node(generate_random_inner_node())
93         tree_leaves.append(root)
94         initial_generation.append(root)
95
96         for _ in range(length_of_tree - 1):
97             tmp_tree_leaves_copy = tree_leaves
98             for i in range(len(tmp_tree_leaves_copy)):
99                 left_child = Node(generate_random_inner_node())
100                 right_child = Node(generate_random_inner_node())
101                 parent = tree_leaves.pop(0)
102                 parent.right = right_child
103                 parent.left = left_child
104                 tree_leaves.append(parent.right)
105                 tree_leaves.append(parent.left)
106
107             tmp_tree_leaves_copy = tree_leaves
108             for _ in range(len(tree_leaves)):
109                 left_child = Node(generate_random_leaf_node())
110                 right_child = Node(generate_random_leaf_node())
111                 parent = tree_leaves.pop(0)
112                 parent.right = right_child
113                 parent.left = left_child
114     return initial_generation

```

تابع شایستگی:

برای تابع شایستگی این کار را کردم که در هر نقطه x مقدار درخت را حساب میکنم و قدر مطلق فاصله را از مقدار اصلی می‌گیرم و جمع میکنم، حال مقدار $fitness$ را برابر با

$1000/distance$

قرار میدهم، دلیل آن هم این است که اگر این مقدار برابر 1000 باشد یعنی فاصله بسیار کم است و هرچه بیشتر باشد بهتر است، هر چقدر هم فاصله بیشتر باید مقدار شایستگی کمتر میشود.

```

160 def fitness_score(generation, x_values, expectation):
161     fitness_values = []
162
163     for i in range(len(generation)):
164         difference = 0
165         for j in x_values:
166             try:
167                 difference += abs(evaluate_expression_tree(generation[i], j))
168             except:
169                 difference = float('inf')
170         try:
171             distance = abs(expectation[i] - difference)
172         except:
173             distance = float('inf')
174         try:
175             if(not math.isnan(distance)):
176                 fitness_values.append(1000/distance)
177         except:
178             fitness_values.append(0)
179     return fitness_values
180

```

انتخاب والدین:

برای انتخاب والدین هر بار دو عدد رندوم انتخاب میکنم و چک میکنم کدامین درخت ها شایستگی نزدیک تری به آن دارند و اینگونه دوتایی هایی انتخاب می شوند و اگر دوتایی انتخاب شده جدید نباشد، دوباره گردش انجام میشود تا والدین جدیدی انتخاب شوند. کدی که در اینجا زدم خیلی جالب نبود، چالش های زیادی داشتم و بارها به باگ برخورد کردم و دلیل آن هم این بود که میخواستم بر اساس شایستگی بیشتر نود ها را انتخاب کنم و اینکه عدم انتخاب جفت والدین تکراری زمان این بخش از کد را بسیار بسیار زیاد کرد.

قطعا کمی refactoring کد این بخش به بهبود سرعت الگوریتم کمک خواهد کرد که من بخاطر باگ های پیاپی زیاد موفق به بهبود آن نشدم.

```

def parent_selector(generation, fitness_values, next_generation_size = 100): # This ugly code tr
    next_generation_parents = []
    next_gen_size = next_generation_size//2
    for i in range(next_gen_size):
        chosen_fitness = random_number_between_0_and_1()*1000
        parent_1_index = return_closest_item_in_list(fitness_values, chosen_fitness)
        parent_1 = generation[parent_1_index]
        chosen_fitness = random_number_between_0_and_1()*1000
        parent_2_index = return_closest_item_in_list(fitness_values, chosen_fitness)
        while parent_2_index == generation[parent_1_index]:
            chosen_fitness = random_number_between_0_and_1()*1000
            parent_2_index = return_closest_item_in_list(fitness_values, chosen_fitness)
        parent_2 = generation[parent_2_index]
        while (parent_1, parent_2) in next_generation_parents:
            chosen_fitness = random_number_between_0_and_1()*1000
            parent_1_index = return_closest_item_in_list(fitness_values, chosen_fitness)
            parent_1 = generation[parent_1_index]
            chosen_fitness = random_number_between_0_and_1()*1000
            parent_2_index = return_closest_item_in_list(fitness_values, chosen_fitness)
            while parent_2_index == generation[parent_1_index]:
                chosen_fitness = random_number_between_0_and_1()*1000
                parent_2_index = return_closest_item_in_list(fitness_values, chosen_fitness)
            parent_2 = generation[parent_2_index]
        next_generation_parents.append((parent_1, parent_2))
    return next_generation_parents

```

تولید نسل بعد:

برای تولید نسل بعد، با توجه به مقداری که ورودی داده میشود، عددی رندوم تولید می شود تا ببینیم ایا والدین را نگه داریم یا فرزندان جدید را جایگزین کنیم(در واقع زمانی که این کار را کردم هدفم انجام آزمایش بود تا ببینم آیا با زیاد نکردن تاثیر درخت های خوب آیا نتیجه بهبود می یابد یا خیر که بنظر جواب بله است اما با افزایش توان محاسباتی بسیار بسیار زیاد).

حال اگر بر اساس عدد رندوم تولید شده تصمیم به ساخت نود های فرزند گرفته شود، دو فرزند جدید طی crossover میشوند و با تولید یک عدد تصادفی دیگر با احتمال ۵ درصد جهش می یابند.

برای جهش به احتمال ۵۰ درصد(با توجه به عدد تصادفی تولید شده) این کار را میکنیم:

یک مقدار رندوم بین ۱ تا ۵ برای عمق جهش انتخاب می شود و به صورت رندوم با احتمال برابر به راست یا چپ می رویم، حال اگر در نود های میانی باشیم operand را عوض میکنیم(باز هم بصورت رندوم) و اگر در برگ ها باشیم یا یک مقدار ثابت رندوم قرار میدهیم یا x یا \sin و \cos .

همچنین با احتمال ۲۵ درصد جهش بزرگ انجام میدهیم که این یا موجب از بین رفتن تمام فرزندان و کات شدن این نود میشود و یا شامل تغییر کلی و در مواردی extend شدن آن.

همچنین برای crossover فقط فرزند راست و چپ دو والد را جابجا میکنم و بیشتر اتفاقات مهم دیگر و حالت های دیگر را ب رخ دادن جهش واگذار میکنم.

یکی از کارهایی که انجامش خوب بود این بود که بهترین والدین را نیز نگه داریم، در ابتدا این کار را کرده بودم، اما چون تعداد فرزندان تولید شده کم میشد آنرا تغییر دادم و یک دلیل دیگر هم این بود که صفات والدین را فرزندانیشان دارند و نگه داشتن دوباره آنها امکان ایجاد جهش و تولید چیزهایی که در رابطه درست وجود دارند را کم می کرد، اما این نیز باز یک آزمایش بود و من بخاطر توان محاسباتی کم، با وجود آزمایش های مکرر موفق به گرفتن نتیجه گیری خوبی نشدم.

```
207 def make_next_generation(selected_parents, new_generation_rate = 0.7):
208     new_generation = []
209     for i in selected_parents:
210         parent_1 = i[0]
211         parent_2 = i[1]
212         probability = random_number_between_0_and_1()
213         if probability <= new_generation_rate:
214             child_1, child_2 = cross_over(parent_1, parent_2)
215             mutation_probability = random_number_between_0_and_1()
216             if mutation_probability > 5:
217                 new_generation.append(child_1)
218             else:
219                 mutated_child_1 = mutation(child_1)
220                 new_generation.append(mutated_child_1)
221             mutation_probability = random_number_between_0_and_1()
222             if mutation_probability > 5:
223                 new_generation.append(child_2)
224             else:
225                 mutated_child_2 = mutation(child_2)
226                 new_generation.append(mutated_child_2)
227         else:
228             new_generation.append(parent_1)
229             new_generation.append(parent_2)
230     return new_generation
```



```

def cross_over(parent_1, parent_2):
    probability = generate_random_number(1, 100) # Better crossover?
    copy_of_parent_1 = copy.deepcopy(parent_1)
    copy_of_parent_2 = copy.deepcopy(parent_2)
    try:
        right_node_of_parent_1 = parent_1.right
        copy_of_parent_1.right = parent_2.right
        copy_of_parent_2.right = right_node_of_parent_1
    except:
        copy_of_parent_1 = copy_of_parent_1
        copy_of_parent_2 = copy_of_parent_2
    return copy_of_parent_1, copy_of_parent_2

def mutation(child_node):
    probability = generate_random_number(1, 100) # Better mutation?
    node = child_node
    if probability <= 25: # Cut the node
        node = Node(generate_random_leaf_node())
    elif probability <= 50: # Extend the node
        new_node = create_initial_generation(1, 4)
        node = new_node[0]
    else: # Change nodes data
        how_far_to_go = generate_random_number(1, 5)
        while node.right != None and how_far_to_go != 0:
            if random_number_between_0_and_1() < 0.5:
                node = node.right
            elif node.left != None:
                node = node.left
        if node.right == None:
            node.data = generate_random_leaf_node()
        else:
            node.data = generate_random_inner_node()
    return node

```

شرط های خاتمه الگوریتم:

در ابتدا تنها شرطی که گذاشته بودم این بود که شایستگی به بینهایت میل کند این یعنی تابع خروجی الگوریتم رفتاری بسیار شبیه به نقاط داده شده و تابع احتمالی داده شده داشته باشد، اما متأسفانه هیچ وقت با این استراتژی الگوریتم من کامل نشد و خروجی ای نداشتم، برای همین در روز آخر سقف تعداد iteration را به کدم اضافه کردم تا حداقل نتایجی تولید و گزارش کنم و آزمایش هایم را انجام بدهم و

نمیدانم آیا این کار درستی بود یا نه. اما در بعضی از مواقع میتوانستیم نتیجه ای بگیریم که بعضا توجیه پذیر باشد.

آزمایشات:

آزمایش اول:

در اولین آزمایش، تابع $2x$ را در بازه ۱ تا ۱۰۰ به الگوریتم دادم و تعداد گردش ها را به ۵ محدود کردم تا بتوانم خروجی داشته باشم. متاسفانه خروجی برای این ورودی زیاد جالب نبود و فکر میکنم دلیل آن تولید درخت های اولیه نه چندان خوب بود که من چند بار این را آزمایش کردم اما یا تابع خروجی بسیار طولانی و عجیب بود یا مثل این خروجی یک عدد ثابت که البته شایستگی این تابع ثابت بالاست و این نشان از این است که بقیه توابع اولیه ساخته شده طی جهش ضابطه خوبی تولید نکردند و باید تعداد گردش ها بیشتر باشد تا نتیجه بگیریم.

```
Terminal Window Help
genetic_programming.py — P1_98102024

Get Started genetic_programming.py x
genetic_programming.py > genetic_algorithm_driver

297
298
299 start = time.time()
300
301 x_values = []
302 expectation = []
303 for i in range(1, 1000):
304     x_values.append(i)
305     #expectation.append(math.cos(i) + 2*math.sin(i) + i**3 + 2*i + 12)
306     #expectation.append(2*i)
307     #expectation.append(math.sin(i))
308     #expectation.append(math.log(i))
309 formula = genetic_algorithm_driver(x_values, expectation, 0.7, 500, 10000, 5)
310 inorder(formula)
311 print()
312
313 end = time.time()
314 print(f"Runtime of the program is {end - start}")
315
316

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 % /Users/imanalipour/opt/anaconda3/bin/
mming.py
#### iteration = 1
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(((4)-(5))/(x)+9)))
Fitness of this tree: 999.0956554142539
#### iteration = 2
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(0)
Fitness of this tree: 455.11961331341865
#### iteration = 3
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(((4)-(5))7)
Fitness of this tree: 4614.694561840138
#### iteration = 4
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(0)
Fitness of this tree: 1442.6950408889634
#### iteration = 5
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(0)
Fitness of this tree: 910.2392266268373
Algorithm execution finished, total # of iterations: 5

(0)
Runtime of the program is 545.9659080505371
(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 %
```

```
Terminal Window Help
genetic_programming.py — P1_98102024

Get Started genetic_programming.py x
genetic_programming.py > ...

297
298
299 start = time.time()
300
301 x_values = []
302 expectation = []
303 for i in range(1, 100):
304     x_values.append(i)
305     #expectation.append(math.cos(i) + 2*math.sin(i) + i**3 + 2*i + 12)
306     #expectation.append(2*i)
307     #expectation.append(math.sin(i))
308     #expectation.append(math.log(i))
309 formula = genetic_algorithm_driver(x_values, expectation, 0.7, 500, 10000, 5)
310 inorder(formula)
311 print()
312
313 end = time.time()
314 print(f"Runtime of the program is {end - start}")
315
316

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 % /Users/imanalipour/opt/anaconda3/bin/
mming.py
#### iteration = 1
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(1)
Fitness of this tree: 333.3333333333333
#### iteration = 2
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(cos(x))
Fitness of this tree: 288.67400959165917
#### iteration = 3
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(cos(x))
Fitness of this tree: 228.4641487611986
#### iteration = 4
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(0)
Fitness of this tree: 500.0
#### iteration = 5
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(1)
Fitness of this tree: 1000.0
Algorithm execution finished, total # of iterations: 5

(1)
Runtime of the program is 51.01364183425903
(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 %
```

آزمایش دوم:

در این آزمایش تابع $\log x$ را در بازه ۱ تا ۱۰۰ به الگوریتم دادم و خروجی آن مقدار ۰ شد که باز هم بخاطر تولید درخت های اولیه نه چندان خوب بود، اما این نتیجه بسیار گویاست چرا که مقدار $\log x$ در این بازه به ۰ بسیار نزدیک است.

آزمایش سوم:

در این آزمایش تابع نسبتاً پیچیده ای که ترکیب یک سینوس و یک کسینوس و X^2 بود را به الگوریتم دادم که خروجی آن $\cos x$ شد اما روند تغییر تابع خروجی در این ورودی جالب است که ابتدا یک تابع پیچیده است و کم کم به \cos تبدیل میشود و شایستگی آن زیاد میشود اما باز هم شایستگی مقدار کمی دارد و این نتیجه بخاطر تعداد گردش های کم بوده، زمان اجرای الگوریتم روی این ورودی بسیار زیاد بود. (۵ دقیقه، در شکل هم آمده)

```
Terminal Window Help
genetic_programming.py — P1_98102024

Get Started genetic_programming.py X
genetic_programming.py > ...

297
298
299 start = time.time()
300
301 x_values = []
302 expectation = []
303 for i in range(1, 100):
304     x_values.append(i)
305     expectation.append(math.cos(i) + 2*math.sin(i) + i**3 + 2*i + 12)
306     #expectation.append(2*i)
307     #expectation.append(math.sin(i))
308     #expectation.append(math.log(i))
309 formula = genetic_algorithm_driver(x_values, expectation, 0.7, 200, 10000, 5)
310 inorder(formula)
311 print()
312
313 end = time.time()
314 print(f"Runtime of the program is {end - start}")
315
> 16

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 % /Users/imanalipour/opt/anaconda3/bin/p
mming.py
#### iteration = 1
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
((((8)/(6))/((4)+(5)))*((x)^(sin(x)))/((1)/(1)))
Fitness of this tree: 14.756990689012783
#### iteration = 2
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
((((8)/(6))/((4)+(5)))*((cos(x))^(3))/((6)/(6))))
Fitness of this tree: 26.21583146630827
#### iteration = 3
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(2)
Fitness of this tree: 23.584160706685186
#### iteration = 4
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(((3)-(3))-((9)-(x)))
Fitness of this tree: 133.93095313322902
#### iteration = 5
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(cos(x))
Fitness of this tree: 22.068897035213332
Algorithm execution finished, total # of iterations: 5

(cos(x))
Runtime of the program is 330.4430730342865
(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 %
```

آزمایش چهارم:

تابع خط خطی را این بار به عنوان ورودی به الگوریتم دادم که مقدار آن را در بالا سمت راست میبینید. باز هم خروجی مقداری ثابت است و دلیل آن این است که ثلورانس شایستگی بقیه توابع را بسیار کم کرده است، یک نکته جالب انتخاب $\sin x$ در یکی از گردش ها است که بسیار جالب بنظر میرسد، یعنی الگوریتم این نوسانی بودن ورودی را فهمیده اما بعداً متوجه شده شاید یک تابع ثابت از یک تابع نوسانی بهتر باشد که شایستگی بیشتر گواه این نکته است.

```
(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 % /Users/imanalipour/opt/anaconda3/bin/p
mming.py
(0, 455)(1, 459)(2, 632)(3, 627)(4, 976)(5, 504)(6, 294)(7, 924)(8, 850)(9, 311)
(10, 986)(11, 123)(12, 696)(13, 346)(14, 121)(15, 880)(16, 174)(17, 334)(18, 546)(19, 212)
(20, 58)(21, 386)(22, 67)(23, 837)(24, 342)(25, 545)(26, 216)(27, 20)(28, 28)(29, 46)
(30, 601)(31, 772)(32, 305)(33, 678)(34, 919)(35, 855)(36, 173)(37, 164)(38, 281)(39, 501)
(40, 614)(41, 73)(42, 197)(43, 730)(44, 248)(45, 168)(46, 403)(47, 834)(48, 479)(49, 703)
(50, 977)(51, 822)(52, 447)(53, 357)(54, 634)(55, 826)(56, 883)(57, 967)(58, 671)(59, 694)
(60, 102)(61, 555)(62, 398)(63, 655)(64, 251)(65, 978)(66, 497)(67, 509)(68, 583)(69, 525)
(70, 146)(71, 282)(72, 581)(73, 167)(74, 669)(75, 653)(76, 510)(77, 880)(78, 804)(79, 902)
(80, 439)(81, 521)(82, 988)(83, 899)(84, 272)(85, 182)(86, 139)(87, 793)(88, 359)(89, 851)
(90, 164)(91, 470)(92, 164)(93, 213)(94, 6)(95, 364)(96, 5)(97, 18)(98, 16)(99, 680)
#### iteration = 1
Selection of parents completed.
```

```
Terminal Window Help
genetic_programming.py — P1_98102024

Get Started genetic_programming.py X
genetic_programming.py > ...

298
299 start = time.time()
300
301 x_values = []
302 expectation = []
303 #for i in range(1, 100):
304     #x_values.append(i)
305     #expectation.append(math.cos(i) + 2*math.sin(i) + i**3 + 2*i + 12)
306     #expectation.append(2*i)
307     #expectation.append(math.sin(i))
308     #expectation.append(math.log(i))
309 for i in range(100):
310     x_values.append(i)
311     y_value = generate_random_number(0, 1000)
312     expectation.append(y_value)
313     print((i, y_value), end = '')
314     if i % 10 == 9:
315         print()
316 formula = genetic_algorithm_driver(x_values, expectation, 0.7, 500, 10000, 5)
317 inorder(formula)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

(80, 439)(81, 521)(82, 988)(83, 899)(84, 272)(85, 182)(86, 139)(87, 793)(88, 359)(89, 851)
(90, 164)(91, 470)(92, 164)(93, 213)(94, 6)(95, 364)(96, 5)(97, 18)(98, 16)(99, 680)
#### iteration = 1
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(2)
Fitness of this tree: 333.3333333333333
#### iteration = 2
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(sin(x))
Fitness of this tree: 16.800305001137193
#### iteration = 3
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(2)
Fitness of this tree: 50.0
#### iteration = 4
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(sin(x))
Fitness of this tree: 283.8708367078697
#### iteration = 5
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(3)
Fitness of this tree: 1000.0
Algorithm execution finished, total # of iterations: 5

(3)
Runtime of the program is 240.1326600657837
(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 %
```

آزمایش پنجم:

در این آزمایش من ورودی الگوریتم را $\sin x$ دادم و آن را اجرا کردم و محدودیت ها را نیز کمتر کردم تا کمی نتیجه با قبلی ها متفاوت تر باشد، در خروجی های قبلی تابع خروجی کوچک بود و خیلی اشتباه انداز، همانطور که میبینید برای این ورودی تابع خروجی پیچیده تر است و شکل آن بنظر می رسد با $\sin x$ مشابهت هایی داشته باشد.

```
Terminal Window Help
genetic_programming.py — P1_98102024

Get Started genetic_programming.py x
genetic_programming.py > ...

297
298
299 start = time.time()
300
301 x_values = []
302 expectation = []
303 for i in range(1, 100):
304     x_values.append(i)
305     #expectation.append(math.cos(i) + 2*math.sin(i) + i*3 + 2*i + 12)
306     #expectation.append(2*i)
307     expectation.append(math.sin(i))
308     #expectation.append(math.log(i))
309 formula = genetic_algorithm_driver(x_values, expectation, 0.7, 500, 10000, 5)
310 inorder(formula)
311 print()
312
313 end = time.time()
314 print(f"Runtime of the program is {end - start}")
315
316

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 % /Users/imanalipour/opt/anaconda3/bin/python /Users/Ima
mming.py
#### iteration = 1
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
((((x)-(3))*((7)*(x)))/((x)^(6))/((x)*(2))))/(((8)+(3))*((7)^(6)))-((sin(x))/(3))-((cos(x))/(5))))
Fitness of this tree: 7087.297479015186
#### iteration = 2
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
((((x)-(3))*((7)*(x)))/((x)^(6))/((x)*(2))))/(((x)+(8))*((3)^(9)))-(((3)/(10))-((5)/(5))))
Fitness of this tree: 4074.7252758971326
#### iteration = 3
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(0)
Fitness of this tree: 1463.56802734982
#### iteration = 4
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(0)
Fitness of this tree: 1209.440389291636
#### iteration = 5
Selection of parents completed.
Creation of new generation completed.
Fittest member of new generation selected and evaluated.
Best tree after this iteration:
(((9)^(cos(x)))^((2)/(5)))/((x)*(x))*((cos(x))/(x)))
Fitness of this tree: 28.342246803198197
Algorithm execution finished, total # of iterations: 5

(((9)^(cos(x)))^((2)/(5)))/((x)*(x))*((cos(x))/(x)))
Runtime of the program is 138.35111689567566
(base) ImanAlipour@Imans-MacBook-Pro P1_98102024 %
```

جمع بندی و نتایج

اولا می‌خواهم بگویم بنظر من دلیل اصلی نتایج نه چندان جالب بخاطر این بود که کاملاً خودم کد را پیاده سازی کردم، و خب بسیاری از جاها پیاده سازی من بهینه نبوده است و بعضی جاها کارهایی کرده ام که شاید منطقی نبوده باشند و به همین دلیل که کاملاً خودم کد را نوشتم نتایج زیاد جالب نبودند. مجدداً با اینکه به شخصه از نتایج پیاده سازی خودم خرسند نیستم، اما حس میکنم بیشتر دلیل این نتایج نه چندان مطلوب بخاطر عوامل تصادفی زیاد در پیاده سازی من بود، دلیل دیگر نیز طول کشیدن اجرا شدن شدن الگوریتم بود. زمانی که کمی محدودیت ها را کمتر میکردم، بعضاً به جواب نمی رسیدم، اما فکر میکنم با کم شدن محدودیت ها این کد من هم نتایج خوبی داشته باشد.

در کل نتیجه گیری من از الگوریتم های ژنتیک، نیاز به میزان محاسبه و سعی و خطای بسیار بالاست که حس میکنم با واقعیت هم سازگار است چرا که موجودات در طی سالیان بسیار طولانی این تغییرات ژنتیکی در آنها رخ داده است و نمونه های بسیار زیادی هم از هر کدام بوجود آمده و از بین رفته است، یکی از مسائلی که در این پیاده سازی به آنها دقت نشد این بود که در طبیعت اعضای ضعیف زنده نمی‌مانند اما در پیاده سازی من به آنها این امکان داده می‌شود، حس میکنم باید احتمال اجازه به آن اعضا را بسیار بسیار کمتر کنم، مسئله دیگر هم محدودیت محاسبات بود که به نظر من برای پیاده سازی من مشکل ایجاد کرد.

به نظر خودم نتیجه گیری من با واقعیت هم سازگار است و بنظر می‌آید برای نتیجه گرفتن از این الگوریتم ها زمان و محاسبات زیادی نیاز است و همچنین هوشمندانگی بیشتر، قطعاً با تغییر در پیاده سازی بعضی از قسمت های کد من، نتایج بسیار بهتر خواهند شد.

در کل الگوریتم های ژنتیکی بخاطر سادگی نسبی بنظر می‌آید بتوانند سوالات را حل کنند یا برای آنها جواب تقریبی خوبی بیابند اما این کار به زمان و توان محاسباتی بالایی نیاز دارد.