پروژه اول

تقریب تابع با کمک برنامه نویسی ژنتیک(GP)

درس هوش مصنوعی وسیستم های خبره

نگین حقیقی - 99521226

استاد درس: دکتر آرش عبدی

نیم سال اول 1402-1401

سخن آغازین:

در این پروژه قصد داریم با استفاده از برنامه نویسی ژنتیک (Genetic Programming-GP) تابع ها را تقریب بزنیم.

نحوه نگاشت مساله به درخت: هر عضو از جمعیت ما در این پروژه، یک درخت است. که نمایشگر یک عبارت ریاضی میباشد. راس های میانی این درختان، عملگرهای ریاضی هستند که از آرایه All_opertors انتخاب میشوند و برگ های آن، مقادیر عددی ثابت و متغیر های ورودی هستند. و بعدا قرار است ورودیها از طریق آن به درخت تزریق شوند.

با اجرای این برنامه، بهترین درخت عبارت ریاضی تقریب زده شده، میزان شایستگی این بهترین درخت، زمان اجرای کل برنامه، تعداد محاسبه شایستگی، تعداد نسلها و.. قابل مشاهده است. و همچنین نمودار هردوتابع نشان داده میشود(تابع اصلی و تابع تخمین زده شده توسط کد ما)

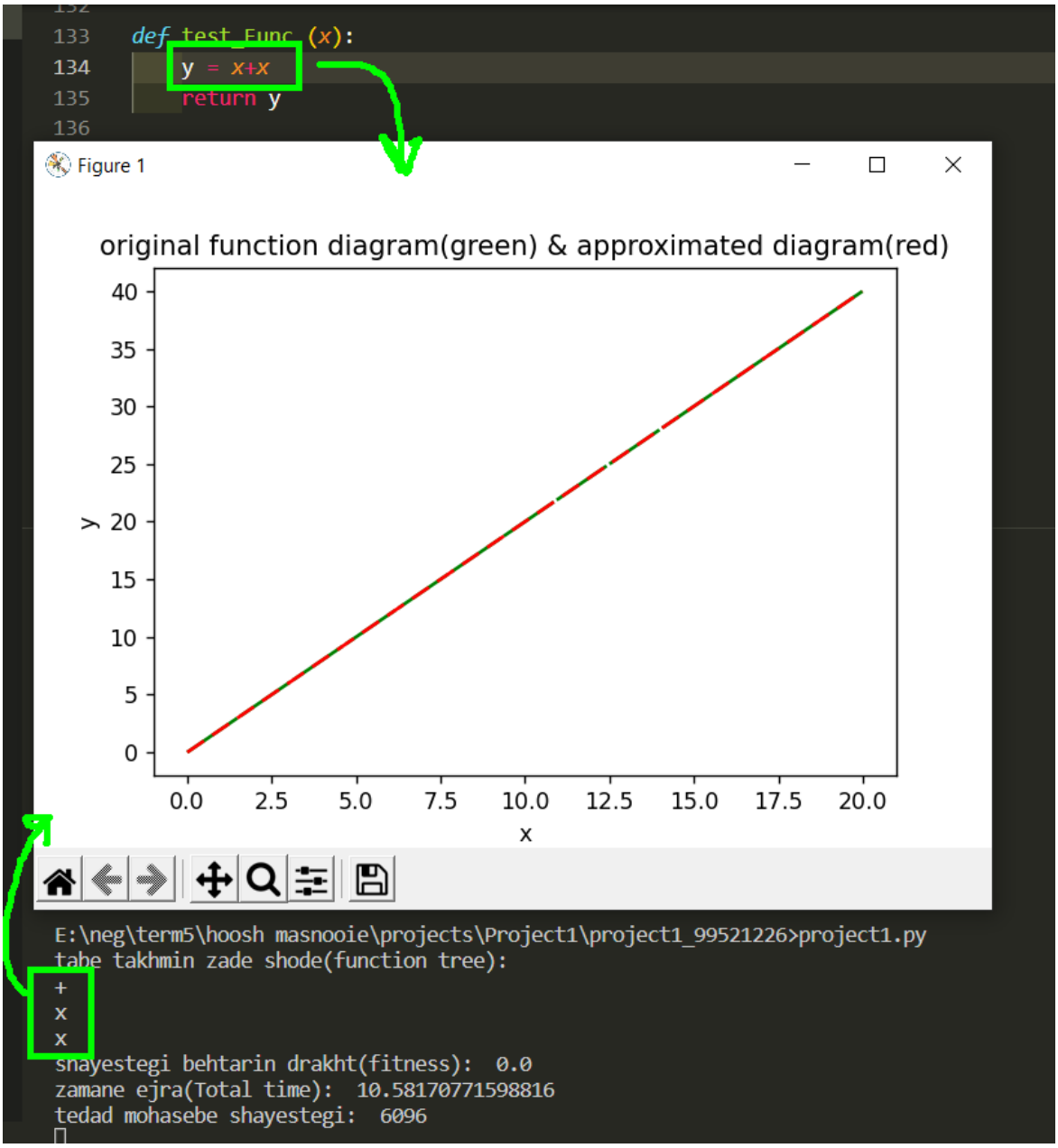
تابع اصلی با رنگ سبز و نمودار تابع تقریب زده ما، با رنگ قرمز مشخص شده. اگر این دو نمودار کاملا برهم منطبق باشند، یعنی تقریب ما بسیار دقیق بوده و دقیقا خود تابع را پیدا کرده ایم. و fitness آن صفر است(درادامه راجع به نحوه اندازه گیری fitness یا همان تابع شایستگی توضیح میدهم اما در یک جمله میتوان گفت، شایستگی هر درخت، انحراف از معیار آن است و طبیعتا هرچه کمتر باشد، بهتر است)

چند آزمایش مختلف و ارزیابی و تحلیل عملکرد الگوریتم در آنها:

ابتدا از مثالی ساده مانند y=x+x=2x شروع میکنیم تا از عملکرد کلی کد مطمئن شویم. انتظار داریم تابع این مثال به طور دقیق و با شایستگی 100درصد پیدا شود.

همانطور که در شکل نیز میبینیم، دقیقا تابع به درستی پیدا شده و اگر درخت آن را رسم کنید، یک درخت با 3 راس داریم که root آن عملگر + است و برگ های آن متغیر x هستند که دقیقا نشانگر همان تابع x+x است. و همچنین اگر نمودار را نیز نگاه کنیم، دو نمودار سبز رنگ(تابع اصلی) و قرمز رنگ(تابع تخمینی) کاملا بر هم منطبق هستند و روی هم افتاده اند و هر دو 2x را نشان میدهند. و همینطور طبق انتظار ما، شایستگی این تابع 100 درصد است و یعنی fitness این

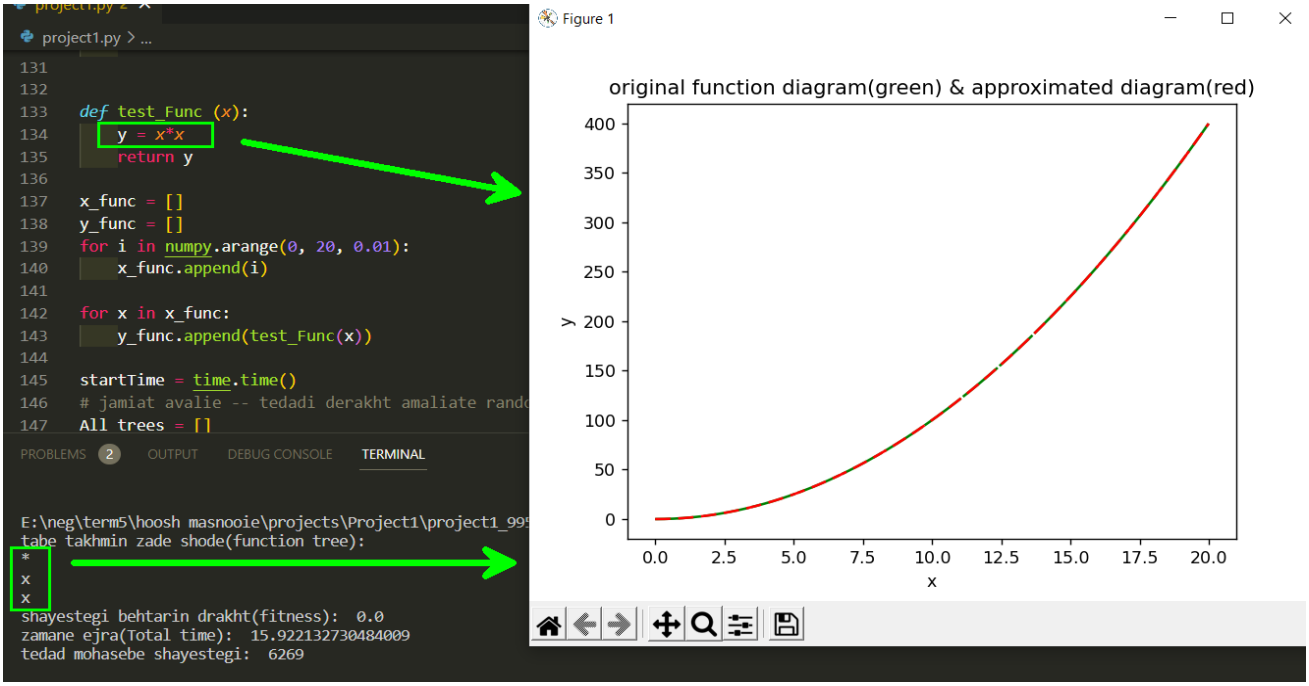
درخت، صفر است که به معنی آن است که انحراف از معیار تمام xهای روی نمودار، صفر شده و تابع تقریب زده ما، دقیقا خود تابع ورودی است.



مثال دوم: باز هم بد نیست یک مثال نسبتا ساده دیگر را بررسی کنیم تا با نوع تحلیل خروجی و درختی که در کنسول چاپ شده و طریقه رسم و تحلیل آن و ارزیابی جواب الگوریتم و میزان دقت و تقریب آن بیشتر آشنا شویم و تسلط پیدا کنیم.

تابع اصلی ما x*x یا همان x^2 است. همانطور که در شکل نیز میبینیم، دقیقا خود تابع پیدا شده و اگر درخت آن را نیز رسم کنید، یک درخت با 3 راس داریم که root آن عملگر * (ضرب) است

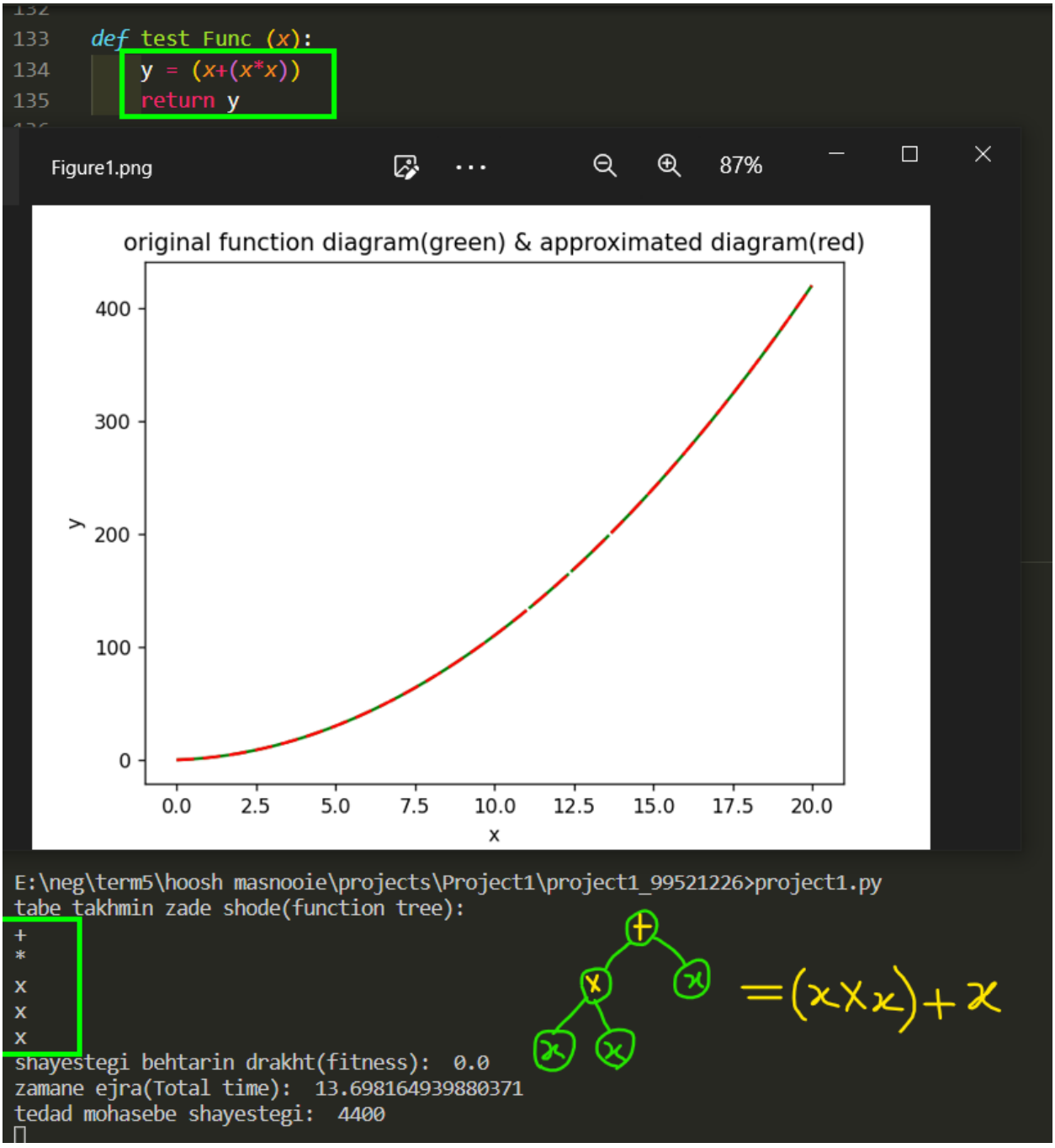
و برگ های آن متغیر x هستند که دقیقا نشانگر همان تابع x*x است. و همچنین اگر نمودار را نیز نگاه کنیم، دو نمودار سبز رنگ(تابع اصلی) و قرمز رنگ(تابع تخمینی) کاملا بر هم منطبق هستند و روی هم افتاده اند و هر دو x^2 یا x*x را نشان میدهند. و همانطور که انتظار داشتیم، شایستگی این تابع 100 درصد است و یعنی fitness این درخت، صفر است که به معنی آن است که انحراف از معیار مقادیر xهای روی نمودار، صفر شده و تابع تقریب زده ما، دقیقا خود تابع ورودی است.



مثال بعدی که قصد داریم بررسی و تحلیل کنیم، تابع y=(x+(x*x)) است. ابتدا درختی که در خروجی به ما داده شده و توسط الگوریتم ما بدست آمده را بررسی میکنیم تا مطمئن شویم همان تابع اصلی را به درستی تقریب زده.

درخت ما 5 گره دارد و اگر از ابتدا شروع کنیم، بالاترین گره یعنی root، عملگر + (جمع) است که یعنی قرار است دو عبارت جبری فرزندانش را باهم جمع کند. حال یکی از فرزندان آن، عملگر * (ضرب) است که دو فرزندش را ضرب میکند. و درمرحله بعد میبینیم که فرزندان این عملگر، هردو x هستند و همان x*x ما را میسازد. و فرزند سمت راست عملگر + که در root دیده بودیم نیز x است و یعنی قرار است x را با نتیجه ی فرزندان چپی جمع کنیم.

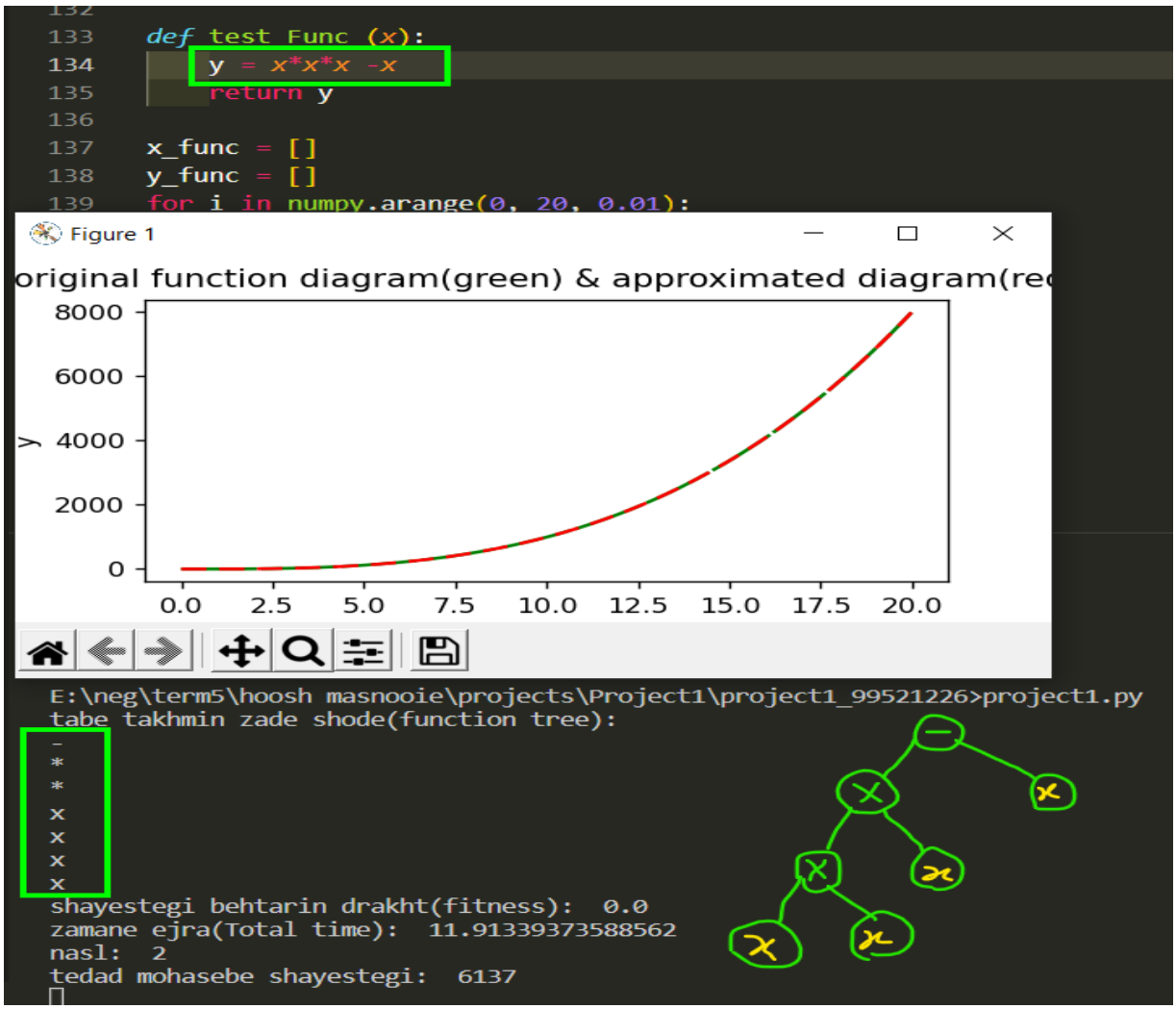
همانطور که دیدید، درختی که در خروجی برگردانده شده، دقیقا درخت همان تابع ((x+(x*x)) است. و فیتنس آن طبق انتظار صفر شده و همینطور دو نمودار سبز رنگ(نمودار تابع اصلی) و قرمز رنگ(نمودار تابع تخمینی) کاملا بر هم منطبق هستند و روی هم افتاده اند. (برای درک بهتر، شکل درخت را نیز در تصویر زیر رسم کرده ام)



تابع دیگری که میخواهیم باهم بررسی کنیم، تابع y=x*x*x-x است. ابتدا درختی که در خروجی به ما داده شده و توسط الگوریتم ما بدست آمده را بررسی میکنیم تا مطمئن شویم همان تابع اصلی

را به درستی تقریب زده. درخت ما 7 گره دارد و اگر از ابتدا شروع کنیم، بالاترین گره یعنی root، عملگر - (تفریق) است که یعنی قرار است دو عبارت جبری فرزندانش را از هم کم کند. حال یکی از فرزندان آن، عملگر * (ضرب) است که دو فرزندش را ضرب میکند. و درمرحله بعد میبینیم که فرزندان این عملگر، یکی x و دیگری مجددا * ضرب است و که دو تا x را درهم ضرب میکندو x*x را میسازد. حال اگر یک مرحله به عقب برگردیم و اولین عملگر ضرب را اعمال کنیم، x*x را در x ضرب میکند و x*x*x ساخته میشود. و فرزند سمت راست عملگر - که در root دیده بودیم نیز x است و یعنی قرار است x را از نتیجه ی فرزندان چپی کم کند.

همانطور که دیدید، درختی که در خروجی برگردانده شده، دقیقا درخت همان تابع x*x*x-x است. و فیتنس آن طبق انتظار صفر شده و همینطور دو نمودار سبز رنگ(نمودار تابع اصلی) و قرمز رنگ(نمودار تابع تخمینی) کاملا بر هم منطبق هستند و روی هم افتاده اند. این درخت نیز در نسل دوم پیدا شده(برای درک بهتر، شکل درخت را نیز در تصویر زیر رسم کرده ام)



تابع شایستگی:

هر یک اعضای جمعیت مان را باید با یک روش سنجید و ارزیابی کرد و درختای بهتر و بدتر را بتوان مشخص کرد. این تابع و نسبت دادن یک عدد به عنوان fitness به هر درخت، میتواند به حالات مختلفی انجام شود. مثلا با نشان دادن یک عدد به درصد، بگوییم هر درختی که عددش به 100 نزدیکتر باشد، بهتر است. یا اگر خود 100 باشد یعنی درخت بهترین است و خود تابع اصلی را نشان میدهد. (به ازای تمام مقادیر ورودی، خروجی اش همان خروجی تابع اصلی است)

روش دیگر برای تابع fitness استفاده از انحراف معیار است. که من در کد پروژه از همین روش استفاده کرده ام. برای هر درخت، یک سری ورودی به آن میدهم. (چه ورودی ای؟ تمام اعداد بین 0 تا 20 که به فاصله 0.01 از هم هستند. یعنی 0.01 و 0.02 و 0.03 و....19.99 و 20.00)

حال به ازای این ورودی ها، ما دو خروجی داریم. یکی خروجی تابع اصلی. مثلا تابع اصلی ما 2x است و یعنی خروجی، تمام ورودی ها را دوبرابر میکند. و یکی خروجی درخت ما. یعنی هردفعه میاییم درخت را به ازای یک ورودی حساب میکنیم(عدد را از برگ ها به درخت تزریق میکنیم و اینقدر محاسبه میکنیم تا به root برسیم).

حال برای محاسبه انحراف از معیار، از فرمولش استفاده میکنیم و هردفعه دو خروجی متناظر با هم را از هم کم میکنیم و به توان دو میرسانیم و باهم جمع میکنیم و حاصل نهایی را تقسیم بر تعداد میکنیم. این عدد هرچه به صفر نزدیک تر باشد بهتر است. برای درک مفهومی و دقیقتر، مثلا فرض کنیم دو خروجی عیییناً شبیه هم هستند. پس حاصل تفریق دو به دو آنها صفر شده و وقتی به توان دو میرسند نیز همچنان صفر هستند و درنهایت عدد صفر تقسیم بر تعداد ورودیها میشود و درنتیجه fitness نهایی صفر است.

کد تابع شایستگی:

```
37  #harchi fitness kamter, behtar
38  def Calc_fitness(y_funcs, tree_Eval):
39      Sum = 0
40      for i in range(len(y_funcs)):
41          Sum += ( (y_funcs[i] - tree_Eval[i]) * (y_funcs[i] - tree_Eval[i]) )
42      enheraf_meyar = Sum/(len(y_funcs))
43      return enheraf_meyar
```

تولید جمعیت اولیه:

در ابتدا باید یک جمعیت اولیه بسازیم. که هر فرد از این جمعیت، یک درخت برای یک عبارت ریاضی رندوم است. مثلا یک درخت x+x را نشان میدهد، یکی دیگر درخت مربوط به عبارت (x+(x*x)-(x/(x+4)+x*x*x) است و به همین ترتیب.

من دراین پروژه، تعداد اعضای جمعیت اولیه ام را 400 در نظر گرفتم اما هرعدد دیگری میتوانید بسته به زمان و میزان دقت مد نظرتان، انتخاب کنید. سپس یک level_count رندوم برای درختان در نظر گرفتم که هردرخت چند ردیف داشته باشد. و درخت را با استفاده از تابع Build_Tree ساختم که همانطورکه در کد زیر میبینید، درخت اینگونه ساخته میشود که به طور بازگشتی، برای تمام نودها (به جز برگها) یک عملگر رندوم، از لیست عملگرها انتخاب میشود. و سپس برگ ها هم مقدارx را میگیرند که متغیر ماست و بعدا قرار است ورودیها از طریق آن به درخت تزریق شوند.(x یکی از متغیرهاست و میتوان از توابعی با دو متغیر مانندx1 وx2 و یا بیشتر نیز استفاده کرد) و درآخر چه زمانی این تابع بازگشتی تمام میشود؟ زمانی که تعداد ردیف های درخت به مقدار مشخص شده که بالاتر گفتم، برسد. اگر level_count 1 باشد یعنی درخت 1 ردیف دارد و یعنی 1 گره دارد.. اگر 2 باشد یعنی درخت 2 ردیف دارد یعنی1+2= 3 گره. و به همین ترتیب... با محاسبه ریاضی به این نتیجه میرسیم که میتوان تعداد گره های درخت را از تعداد ردیف ها بدست آورد1-(2^level_count) = node_count

```
# jamiat avalie -- tedadi derakht amaliate random
All_trees = []
for i in range(400):
    tr = tree()
    level_count = random.randint(1,5)
    # level_count = 1
    rand_op = random.choice(All_operators)
    root = node(rand_op)
    tr.Init_Root(root)
    node_count = (2**level_count)-1
    Build_Tree(tr,root,node_count,0)
    All_trees.append(tr)
```

```
All_operators = ['-', '+','/','*']

def Build_Tree(tr, parent, node_count, count):
    if count == node_count:
        tr.Add_newChild(node('x'), parent)
        tr.Add_newChild(node('x'), parent)
        return
    node1 = node(random.choice(All_operators))
    tr.Add_newChild(node1, parent)
    node2 = node(random.choice(All_operators))
    tr.Add_newChild(node2, parent)
    Build_Tree(tr, node1, node_count, count+1)
    Build_Tree(tr, node2, node_count, count+1)
```

نحوه انتخاب والدین:

برای انتخاب والدین از بین نسل فعلی، از تابع select_LocalBest که در زیر نیز آن را آوردم، استفاده میکنیم. هربار داخل این تابع، به تعداد size (مثلا 200تا) از درختان جمعیت را به صورت رندوم جدا کرده و بهترین آنها را پیدا میکنیم. (یعنی میگردیم بین این 200 درخت، کدامشان fitness کمتر دارد و شایسته تر است) و آن را ریترن میکنیم. این درخت میشود یک مادر یا یک پدر. حال چند بار اینکار را تکرار میکنیم تا نسل جدید ساخته شود. (هر دو بار که از این تابع استفاده کرده و یک مادر و یک پدر انتخاب کردیم، از انها یک بچه (child) تولید میکنیم. که در ادامه به طور دقیق توضیح خواهم داد).

```
def select_LocalBest(trees,size):
    random_pop = []
    random_pop = random.sample(trees, size)
    bestTree = random_pop[0]
    for i in range(size):
        if random_pop[i].fitness < bestTree.fitness:
            bestTree = random_pop[i]
    return bestTree
```

نحوه تولید نسل بعد:

همانطور بالاتر گفته شد، هر بار که و یک مادر و یک پدر انتخاب کردیم، از انها یک بچه (child) تولید میشود که در زیر کد مربوط به آن را میبینید. در زندگی واقعی، مادر و پدر هرکدام یک ژن دارند و بچه تکه هایی از ژن آنها را گرفته و باکنار هم قرار گرفتنش، ژن کودک تولید میشود. در این پروژه نیز، میدانیم مادر و پدر، کدام یک گراف(درخت) هستند که یک آرایه از نودها دارند. (که اولین عضو آن root است و سپس راس های میانی هستند که مانند root عملگر هستند و سپس برگ ها که متغییر های ورودی اند، در انتهای آرایه قرار گرفتند) حال آرایه گره های فرزند را از ترکیب گره های مادر و پدر میسازیم. به اینصورت که یک بازه رندوم از مادر(دو عدد رندوم (دو عدد رندوم برای start وend ش انتخاب میشود) را برمیداریم. و با کنار هم گذاشتن تکه ها ژن فرزند را میسازیم. تکه ها به این صورت انتخاب میشوند که از ابتدا تا Start مادر را اول میگذاریم، سپس از Start تا End پدر را میگذاریم و سپس از End تا ته نودهای میانی (عملگرها)ی مادر را میگذاریم.

و ته این لیست هم به تعداد برگها، متغیر ها برای مثالx را اضافه میکنیم. این میشود ژن(گره های درخت) فرزند. این ژن را به روش های دیگر هم میتوان ساخت. برای مثال مانند خطی که در عکس زیر کامنت شده. به جای سه تکه کردن ژن و اینکه ابتدا و انتها ژن مادر باشد و وسط ژن پدر باشد، میتوان مثلا 4 تکه کرد که اول و سوم از پدر، و تکه دوم و چهارم از مادر باشد. و روشهای دیگر...

از هر یک مادر و پدر، یک فرزند به این روش تولید میشود و این فرزندان روی هم نسل جدید را میسازند.

کد مربوط به تولید فرزندان، را در پایین گذاشتم:

```
def make_Child(mother,father):
    child = ['x','x']
    m_Operatorscount = (len(mother.nodes)-1)//2
    f_Operatorscount = (len(father.nodes)-1)//2
    # num_radif = int(math.log((len(child)+1),2))
    # while num_radif != int(num_radif):
    while math.log((len(child)+1),2) != int(math.log((len(child)+1),2)):
        child = []
        mother_first = numpy.random.randint(0,m_Operatorscount)
        father_first = numpy.random.randint(0,f_Operatorscount)
        mother_last = numpy.random.randint(mother_first,m_Operatorscount)
        father_last = numpy.random.randint(father_first,f_Operatorscount)
        # child = father.nodes[:father_first] + mother.nodes[mother_first : mother_last] + father.nodes[father_last :f_Operatorscount] + mother.nodes[mother_last :m_Operatorscount]
        child = mother.nodes[:mother_first] + father.nodes[father_first : father_last] + mother.nodes[mother_last :m_Operatorscount]

    for i in range ((len(child)+1)):
        n_ch = node('x')
        child.append(n_ch)
    return child
```

(برای جهش نیز میتوان یک ژن را به نحوی تغییر داد تا جهش یابد، مثلا یک عملگر را رندوم انتخاب کند، و به جای آن یک عملگر رندوم دیگر بگذارد. فقط باید دقت شود که اگر عملگر اولیه دوتایی بود، یعنی * یا + یا – یا / بود، باید یک عملگر دوتایی جایگزین شود ولی اگر تکی بود مانند سینوس، باید یک عملگر تکی جایگزین شود. و به همین نحو برای متغیر ها)

شرط خاتمه الگوریتم:

این الگوریتم و مراحل ارزیابی جمعیت، انتخاب والدین، اعمال عملگرهای ژنتیکی و تولید فرزندان، انتخاب بازماندگان تا زمانی ادامه دارد که به یک شرط خاتمه ای برسیم. این شرط خاتمه بسته به خود ماست که چگونه انتخاب کنیم، براساس تعداد مشخصی مثلا nبار باشد تا زمان اجرای کد از یه حدی بیشتر نشود، یا شرط دیگری مانند اینکه درختی پیدا شود که خیلی خوب باشد مثلا Fitnessش کمتر از یک مقدار ثابتی مانند 0.05 باشد.

شرط خاتمه من در این پروژه، روش اول است. یعنی یک تعداد باری این روند انجام شود و هروقت به عددی مانند 2000 رسید الگوریتم تمام شود.

چالش های مواجه شده و روش حل آنها:

در طی این حدود چهارهفته با چالش های زیادی سر پروژه برخورد کردم که بعضی ها مربوط به خود الگوریتم ژنتیک و بعضی مربوط به کد و نحوه پیاده سازی آن بود. در زیر چند تا از این چالش ها را نام میبرم:

- نحوه ساخت درخت های رندوم-> تابع Build_Tree به صورت بازگشتی تعریف شد که هر دفعه فرزندان یک گره به صورت رندوم از عملگرها انتخاب شده و در لیست فرزندان گره parent قرار میگیرند و درخت فعلی به عنوان ورودی به تابع بازگشتی داده میشود. و درانتها در برگها متغیرهای ورودی اضافه میشوند.
- زمان بسیاد زیاد برای اجرای کد-> گاهی برای اجرای کد زمان زیادی حدود چند دقیقه طول میکشید تا تابع را توسط الگوریتم تقریب بزند. اما با کم کردن جمعیت اولیه و انتخاب بهینه شرط خاتمه، این زمان تا حد خوبی کاهش یافت و به چند ثانیه رسید و میتوان گفت از دقت برنامه و نزدیکی جواب به تابع اصلی، چیز زیادی کم نشد و همچنان اغلب توابع تا حد خوبی به جواب نزدیک بوده و یا یکی هستند و fitness نهایی بسیار کم است.
- انتخاب تابع مناسب برای ارزیابی درختان و تابع شایستگی -> همانطور که قبلا گفتیم، روشهای زیادی برای تابع شایستگی وجود دارد مثلا میتوان با نشان دادن یک عدد به درصد، بگوییم هر درختی که عددش به 100 نزدیکتر باشد، بهتر است. یا اگر خود 100 باشد یعنی درخت بهترین است و خود تابع اصلی را نشان میدهد. (به ازای تمام مقادیر ورودی، خروجی

اش همان خروجی تابع اصلی است) یا مثلا برای تابع fitness استفاده از انحراف معیار است. که هرچه مقدارش کمتر باشد و به صفر نزدیکتر باشد، درخت ما بهتر است. انتخاب روش و تابع شایستگی از جمله چالش هایی بود که با آن مواجه شدم.

- تولید فرزندان -> یکی دیگر از اجزای اصلی الگوریتم، تولید فرزندان بود که باز روشهای متفاوتی میتوان برای آن در نظر گرفت. من ابتدا با ترکیب بسیار ساده مادر و پدر(گذاشتن ژن پدر درانتهای ژن مادر) یک فرزند تولید میکردم اما کمی جلوتر پی بردم که این روش خوبی نیس و دقت بالایی ندارد و توابع به خوبی بدست نمی ایند. برای همین از تابعی که بالاتر توضیح دادم استفاده کردم. (که می آمد یک بازه رندوم از مادر و پدر(دو عدد رندوم (دو عدد رندوم برای start وend ش انتخاب میشود) را برمیداشت. و با کنار هم گذاشتن تکه ها ژن فرزند را میساخت. تکه ها به این صورت انتخاب میشوند که از ابتدا تا Start مادر را اول میگذارد، سپس از Start تا End پدر را میگذارد و سپس از End تا ته نودهای میانی(عملگرها)ی مادر را میگذارد. و ته این لیست هم به تعداد برگها، متغیر ها برای مثالx را اضافه میکند.)
- نحوه خروجی دادن درخت -> یک کلاس به اسم tree دارم که درختان از آن جنس هستند. دراین کلاس، یک پراپرتی آرایه به اسم nodes قرار دارد. موقع چاپ خروجی این آرایه پرینت میشود. اما چالش اینجاست که از بین یکسری نود پرینت شده چگونه درخت را رسم و تحلیل کنیم. شیوه کلی به این صورت است که اولین گره، همان root درخت ماست و سپس ورودی های هر عملگر، درست بلافاصله بعد از آن می آیند. یعنی ورودی عملگر + بلافاصله بعد آن خواهند آمد. این روش به رسم سریع تر درخت و درک آن کمک شایانی میکند.

جمع بندی:

در این پروژه یک تابع ورودی را با استفاده از الگوریتم ژنتیک(GP)، تقریب زدیم. روش کلی الگوریتم ژنتیک به صورت زیر است:

1) تعیین روش بازنمایی و تابع برازش(fitness)
2) ایجاد جمعیت اولیه

3) تکرار تا برقراری شرط خاتمه

3-1) ارزیابی جمعیت

3-2) انتخاب والدین

3-3) اعمال عملگرهای ژنتیکی و تولید فرزندان

3-4) انتخاب بازماندگان

که هرکدام از مراحل بالا را در این داکیومنت به طور کامل توضیح دادیم. در این الگوریتم، با انتخاب روش های مناسب برای یکسری توابع مانند تابع شایستگی یا نحوه تولید فرزندان و با انتخاب مقادیر درست مانند تعداد جمعیت اولیه و تعداد اعضای نسلها، میتوان میزان دقت و صحت الگوریتم را بالا برد تا تابع تقریب زده شده حتی المقدور به تابع اصلی نزدیک و نزدیک تر باشد.

در ابتدای این گزارش، تعدادی تابع آزمایش شد تا از صحت کد و عملکرد آن مطمئن شویم و خروجی آنها نیز تحلیل شد و همچنین تمامی کدها و مراحل الگوریتم ژنتیک نیز توضیح داده شد.

پایان

نگین حقیقی-پاییز1401