تمرین سری دوم – بخش دوم (جستجو آگاهانه) به نام خدا محمدرضا باطنی ۹۸۱۲۰۰۶

توضیح روش حل مسئله: الگوریتم ژنتیک را برای حل این مسئله انتخاب میکنیم.

به طوری که هر ژن را معادل یک ترتیب قرارگیری حیوانات (یک جایگشت از اعداد ۱ تا n) قرار میدهیم و کافی است اعمال crossover و mutation و تابع هزینه را روی این ژن ها تعریف کنیم.

Crossover : برای تعریف این عمل باید توجه شود که باید crossover دو ژن معتبر، دو ژن معتبر دیگر شود و از والد ها ساخته شود. برای این کار دو جایگشت p1 و p2 را در نظر میگیریم. به ازای هر i از ۱ تا n ، تعریف میکنیم c1[i] = p2[p1[i]] و همچنین c2[i] = p1[p2[i]] و به این ترتیب دو جایگشت معتبر دیگر ساخته میشود که فرزند این دو هستند.

Mutation : کافی است هر بار دو اندیس رندوم i و j را انتخاب کنیم و جای p[i] و p[j] را عوض کنیم.

تابع هزینه : تعداد جا هایی در ژن که دو حیوانی که نباید، در کنار هم قرار گرفتند را میشماریم و برابر misses قرار میدهیم. این مقدار باید مینیمم شود برای همین مقدار 1/(misses+1) را ماکسیمم میکنیم.

توضیح کد: کلاس GeneticAlgorithmModel را تعریف میکنیم (این کلاس خیلی شبیه کلاس های tensorflow.keras تعریف شده و با الگوگیری از آن ساخته شده.)

کار این کلاس این است. در initializer این کلاس تمام attribute های کلاس تعریف شده (هر کدام جلویش راهنمایی کرده که جنس آن متغیر چیست (همچنین متغیر هایی که مربوط به تعریف مدل اند گرفته میشود.)

سپس تابع compile صدا زده میشود. در این تابع متغیر هایی که مربوط به روند اجرای الگوریتم اند داده میشوند. یعنی توابع crossover و mutation و همچنین تابعی که جمعیت اولیه را ایجاد میکند. همچنین درصد mutation و crossover در این تابع داده میشود و در داخل تابع، تعداد کلcrossover ها و mutation ها در هر مرحله محاسبه میشود. (crossover_num = population_size * crossover_coeff)

سپس تعدادی تابع کمکی (برای استفاده در تابع fit) تعریف میشوند:

choose_weighted(self, k) : این تابع k تا ژن از کل جمعیت را خروجی میدهد. همچنین این کار را به شکلی انجام میدهد که هر ژنی که در جمعیت، تابع هزینه بهتری داشت، با احتمال بیشتری انتخاب شود.

choose_best(self, k) : این تابع k تا بهترین اعضای جمعیت را انتخاب میکند و خروجی میدهد.

تابع extend لیستی از چند ژن را میگیرد و به جمعیت اضافه میکند و تابع remove یک ژن را میگیرد و آن را از جمعیت حذف میکند. (این دو تابع از بیرون کلاس قابل دسترسی اند تا در مواقع دلخواه بتوانیم یک ژن دلخواه را به مدل اضافه کنیم یا از آن حذف کنیم.

تابع print_on_epoch(self, epoch, metrics) در هر epoch الگوریتم صدا زده میشود و اطلاعات مورد نیاز را پرینت میکند (در لیست metrices مقادیری که میخواهیم خروجی داده شوند را به صورت یک رشته وارد میکنیم.)

در نهایت با تعریف توابع بالا به تابع اصلی میرسیم که پس از compile شدن مدل اجرا میشود و روند اصلی الگوریتم ژنتیک است، نام این تابع fit است.

توضیح : در ابتدا با استفاده از random_population_fun که در کامپایل به مدل داده شد، جمعیت اولیه تولید میشود و در population ریخته میشود. سپس مقادیر تابع هدف به ازای تک تک آنها حساب میشود و در objectives ریخته میشود. و در اینجا epoch صفرم (یعنی مقادیر به ازای جمعیت رندوم اولیه) چاپ میشود.

سپس یک حلقه داریم که به اندازه تعداد epoch ها که ورودی داده شده بود تکرار میشود.

در داخل، یک حلقه جدید وجود دارد که ابتدا به تعداد crossover_num/2 بار عمل کراس اوور انجام میشود تا crossover_num تا فرزند از جمعیت اولیه ایجاد شود.

- دو والد رندوم با احتمال وزن دار از جمعیت انتخاب میشود، عمل کراس اوور روی کپی این دو والد انجام میشود تا خود والد ها تغییر نکنند، سپس این دو فرزند به جمعیت اضافه میشوند.

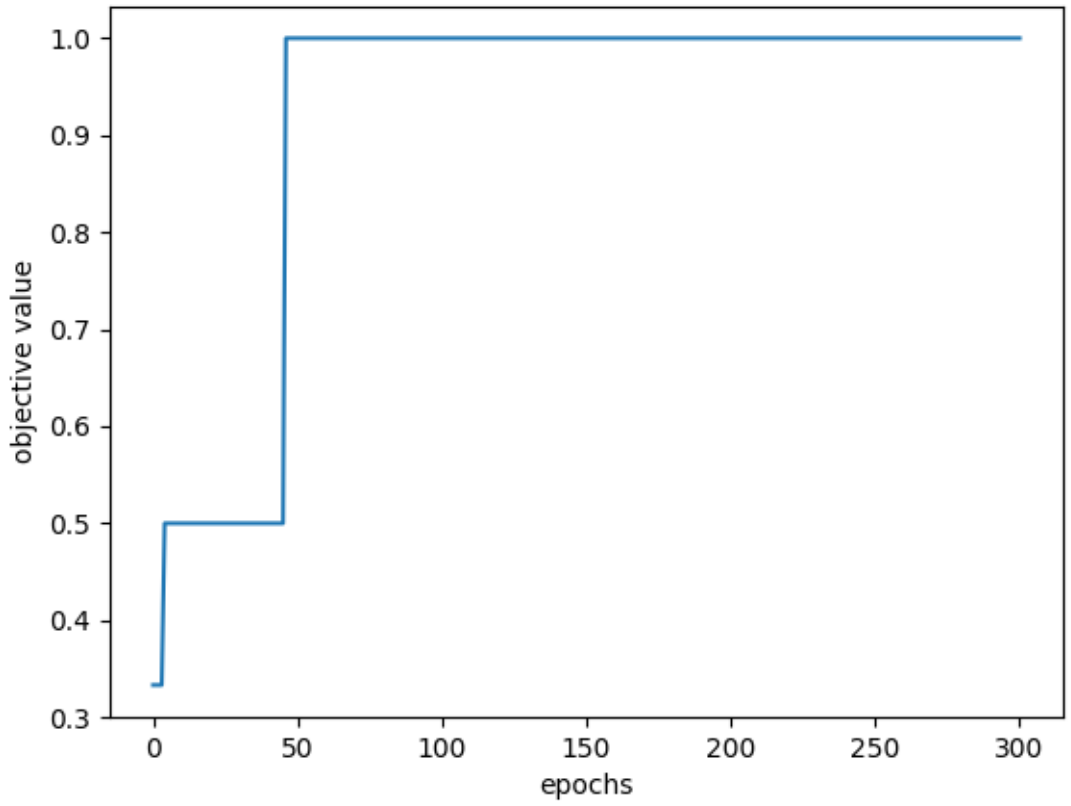
سپس یک حلقه دیگر وجود دارد که ابتدا به تعداد mutation_num بار عمل میوتیشن انجام میشود تا mutation_num تا ژن جهش یافته از جمعیت اولیه ایجاد شود.

- یک ژن رندوم با احتمال یکنواخت از جمعیت انتخاب میشود، عمل کراس اوور روی کپی این ژن انجام میشود تا خود ژن تغییر نکند، سپس این ژن جهش یافته به جمعیت اضافه میشود.

اکنون که جمعیت زیاد شده است، population_size تا بهترین آنها را در population نگه میداریم و لیست objectives را هم با توجه به ژن های جدید آپدیت میکنیم.

کد الگوریتم پیوست شده است.

همانطور که مشاهده میکنید تابع هدف از ۰.۳ شروع کرده، بعد از چند مرحله اجرای الگوریتم به ۰.۵ رسیده و در epoch حدودا ۵۰ ام به ۱ رسیده و از آنجایی که تابع هدف برابر 1/(misses+1) این یعنی مقدار miss صفر شده و به جواب مطلوب رسیده ایم.



الگوریتم ژنتیک یک جمعیت اولیه را میگیرد و تلاش میکند آنها را بهینه بکند اما الگوریتم sa یک نقطه را میگیرد و سعی میکند همان نقطه را بهتر کند و فقط بعضی وقت ها با احتمال کمی به سمت نقطه بدتر میرود.

به طور کلی نگه داشتن تعدادی جمعیت که بهینه نیستند و همچنین عمل mutation در الگوریتم ژنتیک باعث میشود سخت تر در لوکال ماکسیمم گیر کنیم و این عمل که نقاط الگوریتم sa با احتمال کمی به سمت نقطه بدتر میرود باعث میشود سخت تر در لوکال ماکسیمم گیر کنیم.

در کل الگوریتم ژنتیک با جمعیت ۱ شباهت زیادی به sa دارد. (البته اگر در هر نسل دقیقا k تا بهترین را انتخاب نکنیم و احتمالی پیش برویم تا بعضی وقت ها نقطه بدتر شود.)