۱۴۰۱/۱۲/۸

مبانی و کاربردهای هوش مصنوعی

پروژه شماره ۱

بررسی وجود رابطه نمایی در رشد و جهش زمان اجرا نسبت به توابع همسان اما با تعداد حلقه های تودرتو متفاوت

استاد: جناب آقای دکتر فیضي درخشي

نویسنده: **پارسا یوسفی نژاد محمدی**

شماره دانشجویی: ۱۴۰۰۵۳۶۱۱۰۴۸

مقدمه

می‌دانیم که در حالت تئوری در صورت اضافه نمودن هر حلقه for اضافه، مرتبه اجرایی به صورت نمایی رشد خواهد کرد. اما آیا در دنیای حقیقی و با آزمایش‌های عملی هم میتوان به چنین نتیجه‌ای رسید یا نه؟

هدف پروژه

در این پروژه میخواهیم تحقیق کنیم که اضــافه کردن حلقه‌های والد مشــابه بیشــتر(به تعداد k تا) به صــورت تودرتو آیا زمان اجرای برنامه را به شـکل نمایی، یعنی به شـکل n^k برابر که در آن n تعداد دفعات اجرای یک تابع دلخواه چون f(X) است را افزایش می‌دهد یا نه؟ یعنی به عنوان نمونه در صورتی که **یک** حلقه بیشـتر به برنامه به صورت nested و وابسته اضافه کنیم، آیا زمان اجرای برنامه **n برابر** (رشد نمایی : $O(n^{d+1}) \mid d = \text{number of fors}$) می‌شود؟

تشریح کد پروژه

ابتدا ۵ تابع بعد D تعریف می‌کنیم (D۱ تا D۵) در بعد D۱ یک حلقه for داریم و در هر بعد یک for دیگر اضافه میشود.

بلوک اصلی کد که اجرا میشود: ۱. *محاسبه تابع f*   ۲. *پیدا کردن ماکسیمم مقدار تابع و انتساب آن به عنوان y_max در صورت نیاز و پیدا کردن ماکسیسم رادیان‌هایی است که موجب این بهینگی شده‌اند.*

البته که همواره درایه‌های بردارماکسیمم با هم برابراند، چرا که اگر سینوس xای موجب ماکسیمم شدن نتیجه بشود، تمام xهای بردار X هم به ازای همان رادیان، ماکسیمم سینوس را نتیجه میدهند که در نهایت منجر به ماکسیمم جمع(sum) می‌شود.

همچنین برای بررسی زمان اجرا صرفن خود بلاک اصلی کد که در داخل ترین for قرار دارد، قبل

و بعد forها تایمر میگذاریم تا زمان محاسبه تکرار تابع اصلی f بدست بیاید و تا حد امکان زمان سایر عملیات‌ها چون تعریف متغیرهای اولیه و دیگر پارامترها در زمان تمام شده اثر نداشته باشند. البته گذاشتن تایمر در ابتدا و یا از زمان شروع حلقه، هیچ تاثیر ملموسی در زمان‌های اجرا بدست آمده نمی‌گذارند، اما حتمن باید تایمر در درون تابع بعد باشد و نه بیرون از تابع اصلی.

بعد از اتمام اجرای حلقه ها، تابع printRes را فراخوانی میکنیم تا نتایج محاسبه شده در تابع f(x) را چاپ کند. سپس از تابع بدون برگرداندن هیچ مقداری خارج میشویم و برنامه خاتمه می‌یابد.

تابع f را به‌نحوی تعریف میکنیم که یک ورودی بردار X که همان آرایه تک بعدی است را به همراه اندازه بردار(تعداد درایه‌ها) دریافت بکند، و سپس سینوس تک‌تک درایه‌ها را بر حسب رادیان محاسبه و با هم جمع میکند و در آخر سر نتیجه را return میکند. این تابع، همان $\sum_{i=1}^{n}\sin(x_i)$ را محاسبه می‌کند.

در تابع main هم تنها به صورت تکی به فراخوانی تابع‌ها بعد با step (Δx)های مختلف از ۰.۹ تا ۰.۰۰۱ می‌پردازیم، علت فراخوانی توابع به صورت تکی و نه با اجراهای پشت سر هم بدلیل خنثی کردن سایر پارامترهایی هستند که در زمان نهایی اثر میگذارند است.

جمع بندی:

در مجموع تمام کاری که برنامه برای ما قرار است انجام دهد، محاسبه زمان اجرای ۵ تابع بعد با ۵ مقدار step مختلف است ، یعنی در کل ۲۵ مرتبه فراخوانی و چاپ کردن زمان اجرای تابع اصلی f(X) که در صفحه بعدی، جدول زمانی ۲۵ اجرا (در صورت ممکن بودن محاسبه) را رسم می‌کنیم.

شرح پروژه

پس از تکمیل کد مربوطه، توابع هر بعد را با سه نوع Δx مختلف اجرا میکنیم و زمان اجرای هر کد را بدست میاوریم و آن را در داخل سلول جدول زمان اجرا (بر حسب ***نانوثانیه***) یادداشت میکنیم، با توجه به اینکه در هر بار اجرای برنامه برای هر بعد با استپ های مشخص، زمان‌های متفاوتی نسبت به رابطه نمایی بنا به ویژگی‌های سخت‌افزاری و سایر پارامترهای دخیل بدست میاد. هر سلول را از میانگین ۵ مرتبه اجرا بدست آورده‌ایم تا به زمانی دقیق‌تر برسیم.

مشاهده میکنیم که برخی از سلول‌های پایانی، خالی گذاشته شده‌اند و این بدین دلیل است که محاسبه زمان اجرای تابع در یک زمان معقول امکان پذیر نبوده، بدین جهت در ادامه پس از تحلیل جدول زمانی، با تخمین زدن زمان تقریبی اجرای بلوک‌های خالی را محاسبه میکنیم و جدول را تکمیل می‌کنیم.

راهنمای جدول:

- ستون‌های جدول، نشان دهنده شماره بعد (تعداد for) توابع D می‌باشند.
- سطر های جدول، نشان دهنده گام‌های حلقه for می‌باشند.
- تمامی زمان‌های اجرا بدست آمده به صورت دقیق و برحسب نانوثانیه (ns) می‌باشند.
- بلوک‌های خالی به منزله ممکن نبودن محاسبات در زمان معقول هستند.

✦جدول زمانی اجرای تابع f(X) در بعدها و قدم های مختلف

قدم \ بعد	D1	D2	D3	D4	D5
$\Delta x = 0.9$	774ns	6,334ns	75,958ns	908,376ns	11,872,458ns
$\Delta x = 0.3$	1,437ns	34,125ns	1,328,750ns	40,088,635ns	1,041,358,098ns
$\Delta x = 0.1$	2,858ns	271,175ns	25,920,543ns	1,970,328,042ns	237,861,642,841ns
$\Delta x = 0.01$	22,209ns	22,456,750ns	20,148,465,291ns	----	----
$\Delta x = 0.001$	137,750ns	1,151,899,292ns	----	----	----

If $\Delta x = 0.9 \Rightarrow b = n = (10-0)/\mathbf{0.9}) = 11.1$

If $\Delta x = 0.3 \Rightarrow b = n = (10-0)/\mathbf{0.3}) = 33.3$

If $\Delta x = 0.1 \Rightarrow b = n = (10-0)/\mathbf{0.1}) = 100$

If $\Delta x = 0.01 \Rightarrow b = n = (10-0)/\mathbf{00.1}) = 1000$

If $\Delta x = 0.001 \Rightarrow b = n = (10-0)/\mathbf{0.001}) = 10000$

تحلیل جدول زمانی + نتیجه‌گیری

تعداد دفعات تکرار فراخوانی تابع f(x) را فارغ از وابستگی حلقه درونی(Σ) به بعد (اندازه بردارX) آن درنظر میگیریم. حال برای بررسی وجود رابطه نمایی در افزایش زمان از یک بعد به بعد دیگر با Δxهای یکسان، چندین نمونه را بررسی میکنیم و نشان میدهیم که فرض زمان اجرا فورهای متوالی با Δx=0.3 برای تابع f در بعد mام 33^{m-n} برابر بعد nام(m>n) میباشد.

بخش اول: اثبات عملي وجود رابطه رشد نمايي در بعدها با استپ های مختلف

بررسی جهش زمانی Δx=0.1 D1 → D2

در تابع بعد D۱ در صورتی که Δx (step) برابر ۰.۱ باشد، تعداد تکرار تابع f(x)، ۱۰۰=۰.۱/(۰-۱۰) بار خواهد بود از طرفی مرتبه اجرایی برنامه برای تابعمان، برابر O(n) خواهد بود چرا که تنها n بار، که در اینجا n برابر ۱۰۰ است، تابع f اجرا خواهد شد. در این حالت میانگین زمان اجرای D۱ برابر ۲۸۵۸ ns است.

حال با همین Δx سراغ تابع بعد دوم D۲ که یک حلقه تودرتو (nested) اضافه دارد میرویم، در این مورد تابع f ۱۰۰ برابر تعداد بعد اول، یعنی ۱۰۰×۱۰۰ بار اجرا خواهد شد. مرتبه اجرایی کد فوق واضح است که O(n×n) = O(n۲) است چون یک for اضافی‌تر داریم.

زمان انجام این دستور برابر با ۲۷۱۱۷۵ns میباشد. در شرایط تئوری (آزمایشگاهی) انتظار داریم که زمان اجرا هم ۱۰۰برابر شود، در عمل هم مشاهده می‌شود که زمان اجرا D2 نسبت به D1 ۹۴ = ۲۷۱۱۷۵/۲۸۵۸ برابر میشود که بسیار نزدیک به حالت تئوری بوده و علت عدم تطابق دقیق بدلیل وجود برخی از پیچیدگی‌های سخت‌افزاری-نرم‌افزاری است.

پس در این مورد با ۱۰۰ برابر کردن تعداد دفعات اجرا، زمان اجرا **۹۴** برابر میشود و این بدین معناست که با افزودن یک حلقه for تودرتو اضافه‌تر توانستیم به صورت عملی نمایی بودن جهش زمان اجرا (n برابر شدن= افزایش درجه تابع چندجمله‌ای) را ببینیم.

بررسی جهش زمانی $\Delta x=0.1$ $D4 \rightarrow D5$

جهش زمانی از D۴ به D۵ با ۰.۱ = Δx را مقایسه می‌کنیم، از آن جهت که در D۴ حدودن ۱.۹۷میلیارد نانوثانیه اجرا طول میکشد، با اضافه کردن یک حلقه تودرتو، تعداد دفعات اجرای تابع f مجددن ۱۰۰ برابر میشود، اما زمان اجرا تقریبن۱۲۰ برابر میشود که یکی از دلایل بشدت تاثیرگذار، اضافه شدن یک مرحله جمع دیگر بخاطر افزایش بعد (یک درایه اضافه‌تر بردار) در تابع f می‌باشد. در کنار همه این عوامل، پارامترهای بسیاری در زمان اجرای واقعی دخیل‌اند که نمیشود از آن‌ها چشم پوشی کرد. پس با اغماض از چنین پارامترهایی میتوان گفت که رشد پیچیدگی زمانی از بعد چهارم($O(n^۴)$) به بعد پنجم($O(n^۵)$) به شکل نمایی بوده چرا که با افزایش هر بعد، مرتبه اجرایی، n برابر شده و در بعد Mام به بعد Kام مرتبه اجرایی $n^{(K-M)}$ برابر شده است که به صورت جهش نمایی است.

بررسی جهش زمانی $\Delta x=0.3$ $D3 \rightarrow D4$

برای بررسی بیشتر به جهش از D۳ به بعد D۴ با ۰.۳ = Δx میپدازیم. با step=۰.۳، تعداد اجرای هر حلقه برابر است با: ۳۳.۳=(۰.۳/(۰-۱۰)) ، این بدین معنیست که در بعد سوم که ۳ حلقه تودرتو داریم، دستور f به تعداد $(۳۳.۳)^۳$ بار اجرا خواهد شد، و در بعد D۴ هم $(۳۳.۳)^۴$ بار.
پس ما انتظار داریم که زمان اجرا هم در بعد چهارم ، ۳۳.۳ برابر بعد سوم باشد. در عمل با مقایسه دو زمان اجرا بدست آمده داریم: ۳۰=۱۳۲۸۷۵۰/ ۴۰۰۸۸۶۳۵ ، یعنی زمان اجرا ۳۰ برابر شده است که به نتیجه آزمایشگاهی که جهش ۳۳.۳ برابری است، بسیار نزدیک بوده، از این رو میتوان گفت که در جهش از هر بعد به هر بعد بالاتر به صورت نمایی افزایش پیدا میکند.

بررسی جهش زمانی $\Delta x=0.9$ $D2 \rightarrow D5$

میتوانیم با کمی اغماض از سایر پارامترهای سخت‌افزاری-نرم‌افزاری و همچنین متغیر بودن عملکرد تابع f در هر بعد، بگوییم که با استپ ۰.۹ از بعد D۲ که مرتبه اجرایی آن $O(n^۲)$ | n==۱۱.۱ است اگر به بعد D۵ که زمان اجرای دستور f آن $O(n^۵)$ است برویم، جهش زمانی **۱۳۳۱**=$۱۱^۳$ برابری خواهیم داشت که نتایج جدول هم این مورد را تصدیق میکنند، چرا که: ۱۸۷۴=۱۱۸۷۲۴۵۸/۶۳۳۴
این یعنی اینکه زمان اجرا از D۳ به D۵، **۱۸۷۴** برابر شده بسیار به رشد n^3 برابری حالت تئوری نزدیک است.

بررسی جهش زمانی Δx=0.001 D1 → D2

حال به بررسی جهش از بعد D۱ به بعد D۲ اما با step = ۰.۰۰۱ پرداخت میکنیم، تعداد اجرای هر حلقه به صورت دقیق برابر است با :۱۰۰۰۰=(۰.۰۰۱/(۰-۱۰)) یعنی هر حلقه **ده هزار بار** تابع f را فراخوانی می‌کند، از طرفی در بعد D۱ چون یک حلقه داریم، پس به همین تعداد هم تابع اصلی اجرا میشود، در بعد دوم، چون دو حلقه for تودرتو وجود دارد، پس ۱۰۴×۱۰۴ شاهد اجرای تابع f خواهیم بود، برای بررسی این جهش n برابری از یک بعد به بعد بالاتر در عمل، زمان‌های اجرای دو تابع را با کمک جدول زمانی مقایسه میکنیم: **۱۱۲۱۰** =۱,۱۵۱,۸۹۹,۲۹۲/۱۳۷,۷۵۰ همانطور که میبینیم با یک زمان اجرا در واقعیت رشد **یازده هزار برابری** را تجربه کرده است. و یکی از علت‌های اصلی دقیق نبودن کامل با نتیجه تئوری، اضافه شدن یک مرحله محاسبه جمع بیشتر در تابع f که منجر به طول کشیدن بیشتر زمان اجرای D۲ شده است.

نتیجه‌گیری:

در این بخش توانستیم با بررسی کامل **پنج** پرش از بعدهای گوناگون با استپ‌های مختلف در عمل نیز رابطه نمایی در جهش مرتبه زمان اجرایی تابع f را نتیجه بگیریم، به همین منوال میتوان برای تک‌تک سلول‌های جدول زمانی با سطرهای یکسان وجود این رابطه را تنها با تناسب بستن زمان اجرای یک تابع بعد به تابع بعد دیگر در همان سطر به آسانی در عمل نشان داد.

بخش دوم: تخمین زمان اجرای *سلول‌های خالی* جدول با کمک وجود رابطه رشد نمایی طور مرتبه‌های اجرایی تابع f

حال در این بخش پس از اینکه توانستیم وجود چنین رابطه‌ای را به صورت عملی و با آزمایش‌ها و نمونه‌های واقعی نشان بدهیم، اکنون میتوانیم با تقریب بسیار مناسبی سلول‌های خالی‌ای را که محاسبه زمان اجرای آن‌ها با یک کامپیوتر شخصی در یک زمان معقول امکان پذیر نبود را محاسبه کنیم و همچنین به علت عدم ممکن بودن این امر هم بپردازیم و بزرگی چنین جهشی را در دنیای واقعی در غالب این کدها به شکل ملموسی ببینیم

* محاسبه زمان اجرای [Δx=0.01] [$\mathcal{D}$4]

در سلول D۳ در ردیف Δx = ۰.۰۱ مشاهده میشود که زمان اجرا بر حسب ثانیه تقریبن برابر ۲۰ ثانیه می‌باشد، میدانیم که افزودن هر حلقه for (رفتن به بعد بالاتر) منجر به n برابر شدن تعداد اجرای تابع f می‌شود، از طرفی n (b) در اینجا با توجه به استپمان برابر است با: = ۰.۰۱/(۰-۱۰) ۱۰۰۰ که این بدین معناست که در صورت اضافه کردن یک حلقه والد دیگر از D۳ برای رفتن به بعد چهارم، زمان اجرا تقریبن ۱۰۰۰ برابر خواهد شد، یعنی زمان اجرای تابع در بعد جدید برابر است با:

20s × ۱۰۰۰ = ۲۰۰۰۰ ثانیه = 5:30 ساعت

پس با تقریب خوبی میتوانیم بگوییم که با اضافه کردن تنها یک حلقه، برنامه به عبارتی تقریبن **۵ونیم ساعت** زمان برای اجرا نیاز دارد.

* محاسبه زمان اجرای [Δx=0.01] [$\mathcal{D}$5]

برای محاسبه این سلول میتوانیم از D4 یا D3 و یا ابعاد کمتر کمک بگیریم، به دلخواه از زمان D4 کمک میگیریم، زمان اجرای D4 با استپ 0.01 را ۵:۳۰ ساعت بدست آوردیم، با رفتن به بعد بالاتر(اضافه کردن یک حلقه بیشتر) زمان مجددن ۱۰۰۰ برابر خواهد شد، پس زمان اجرای تقریبی برابر خواهد بود با:

5:30h * 1000 = 229 Days (روز)

پس برای بدست آوردن زمان اجرای این درایه از جدول، با یک کامپیوتر معمولی به ***حداقل*** **۲۲۹**

روز زمان احتیاج خواهیم داشت، می‌گوییم حداقل چرا که در بعد بالاتر یک مرحله جمع بیشتر در تابع f داریم که آن را در محاسباتمان لحاظ نکردیم.

---

*محاسبه زمان اجرای [Δx=0.001] [D3]

زمان D2 (۱.۱ ثانیه) در همان سطر را در تعداد دفعات تکرار(۱۰۰۰۰بار) یک حلقه ضرب می‌کنیم:

1.151899s×10000 = ۱۱۵۱۸s = 3:12 hours

*محاسبه زمان اجرای [Δx=0.001] [D4]

از D3 برای محاسبه مدت زمان اجرای D4 استفاده می‌کنیم:

3:12h * 10000 = 1329 Days = 3.64155 Years

اجرای این تابع ۳.۶ سال نیاز به زمان دارد.

*محاسبه زمان اجرای [Δx=0.001] [D5]

چون نمونه‌های قبل این مورد را محاسبه می‌کنیم:

3.64155 Years * 10000 = 36415.5Y = 36.4 Centuries

اجرای این کد به حداقل ۳۶.۴ قرن زمان نیاز دارد یعنی **حداقل ۳۶.۴ هزارسال**.

می‌خواهیم D۵ را با استپ‌های ۰.۰۰۱ اجرا بنماییم، تعداد اجرای هر حلقه برابر است با ۱۰۰۰۰=((۱۰-۰)/(۰.۰۰۱) دفعه، که با ۵ حلقه تودرتو، به‌عبارتی میشود: ۱۰۰۰۰^{۵} = ۱۰^{۲۰} مرتبه که تعداد بسیار زیادی بوده، و انجام هیچ دستوری برای کامپیوتر عادی با این تعداد عملیات در یک زمان معقول یعنی ۳۶.۴ قرن امکان پذیر نبوده، لذا امکان محاسبه زمان اجرای برنامه با چنین استپ‌هایی ممکن نیست، که این خود مهر تاییدی بر رشد نمایی زمان اجرایی بوده، چرا که آنقدر جهش بزرگی از D۴ با Δx=۰.۰۰۱ به D۵ یعنی ***ده هزار برابر انجام بیشتر*** داشتیم، که اصلن کامپیوتر قادر به محاسبه‌اش نیست.

رسم مجدد جدول زمانی با کمک اطلاعات حاصل شده برای مقایسه و دید بهتر زمان‌های اجرایی برنامه با یکاهای متناسب با بزرگی هر زمان

در نهایت پس از تخمین و محاسبه کردن تمامی سلول‌های خالی، جدول تکمیل شده زمانی را متناسب با **بهترین یکا** برای آن زمان رسم می‌کنیم:

راهنمای جدول:
- ستون‌های جدول، نشان دهنده شماره بعد (تعداد for) توابع D می‌باشند.
- سطرهای جدول، نشان دهنده گام‌های حلقه for می‌باشند.
- تمامی زمان‌های اجرا بدست آمده برحسب بهترین یکا ممکن (**نانو: ns** – **میکرو: us** – **میلی: ms** – **ثانیه: s** – **دقیقه: min** - **ساعت: h** – **روز: d** – **سال: y** – **قرن: Cen**) برای آن داده می‌باشند.

✦**جدول *کامل* زمان اجرای تابع f(X) در بعدها و قدم های مختلف**

قدم \ بعد	$\mathcal{D}1$	$\mathcal{D}2$	$\mathcal{D}3$	$\mathcal{D}4$	$\mathcal{D}5$
$\Delta x = 0.9$	774ns	6.3us	75.9us	908.3us	11.8ms
$\Delta x = 0.3$	1.4us	34.1us	1.3ms	40ms	1s
$\Delta x = 0.1$	2.8us	271.1us	25.9ms	1.9s	237.8s
$\Delta x = 0.01$	22.2us	22.4ms	20.1s	5:30h	299d
$\Delta x = 0.001$	137.7us	1,1s	3:12h	3.6y	36.4Cen

Parsa Yousefi Nejad

جمع‌بندی پروژه اول

در این پروژه در ابتدا به بررسی کد برنامه پرداختیم و قسمت‌های مختلف آن را شرح دادیم، سپس جدولی ۲۵ خانه‌ای برای تمام حالت‌های بعد و Δx ساختیم و در بخش اول به شکل عملی به بررسی وجود رابطه نمایی بین پرش‌های مختلف پرداختیم و درستی رابطه را به شکل تئوری و آزمایشگاهی هم نتیجه گرفتیم. در بخش دوم خانه‌های خالی جدول را که توسط کامپیوتر شخصی قابل محاسبه نبودند را با استفاده از وجود رابطه نمایی، تخمین و تقریب زدیم و مشاهده کردیم، بدلیل خاصیت جهش نمایی، چه رشد سرسام آوری در مرتبه زمان اجرای حداقل توابع داریم. سپس جدول زمانی را با بهترین یکاها برای آن زمان ها تکمیل کردیم.