حلقه
نشریهٔ دانشجویی دانشکدهٔ ریاضی و علوم کامپیوتر
شماره ۴ - تابستان ۱۴۰۰

امنیت اطلاعات در عصر ارتباطات: مقدمه‌ای بر رمزنگاری

به نام او

اکنون که چهارمین شماره از مجلهٔ علمی-دانشجویی حلقه منتشر شده‌است برخود می‌بالیم که یک سال در کنار شما بودیم و قدم به قدم کامل‌تر شده‌ایم. این موفقیت حاصل نمی‌شد مگر با تلاش‌های جمعی دانشجویان علاقه‌مند به دنیاهای بی‌پایان ریاضیات و علوم کامپیوتر اما همچنان در آغاز راه هستیم و امیدواریم بتوانیم با کمک پیشنهادات شما شماره به شماره پیشرفت کنیم. در این شماره از حلقه، در بخش ریاضی به موضوعات مختلفی می‌پردازیم. ابتدا در خصوص اثبات‌های هیچ‌آگاهی سخن می‌گوییم و گریزی به مطالب نظریهٔ کدگذاری می‌زنیم آنگاه به چگونگی ساخت مجموعهٔ کانتور و برخی ویژگی‌های آن می‌پردازیم سپس در ادامه موضوع آشنایی با خمینه‌ها، از شمارهٔ قبلی، آن‌ها را بیشتر بررسی می‌کنیم. گرایشی از ریاضی کاربردی را معرفی کرده و دربارهٔ تابع پایه ۱۳ کانوی می‌خوانیم اما سوال آخر این است: چند عدد وجود دارد؟ در بخش علوم کامپیوتر نیز گستره‌ای از موضوعات مختلف را پوشش می‌دهیم. در مطلب رمزنگاری با روش‌های مختلف رمزنگاری از گذشته تا به امروز آشنا می‌شویم، با شبکه‌های گرافی و برخی کاربردهای آن‌ها در مطلب شبکه‌های گرافی آشنا می‌شویم، سپس با مروری بر روش‌های حل معادله در کامپیوتر، روشی جدید و سریع‌تر را معرفی می‌کنیم، با زبان برنامه‌نویسی ارلنگ و ویژگی‌های خاص آن برای طراحی سیستم‌های موازی آشنا می‌شویم، سیستم کمک‌خلبان گیت‌هاب برای پیشنهاد و تکمیل کد را بررسی می‌کنیم، مقدمه‌ای بر الگوریتم‌های برنامه‌ریزی پویا را می‌خوانیم، با پدیدهٔ رویای عمیق در شبکه‌های عصبی پیچشی آشنا می‌شویم و در پایان مروری گذرا بر مسئله‌ی پیش‌بینی ساختار سوم پروتئین در بیوانفورماتیک خواهیم داشت. علاوه بر این مطالب، چکیده‌ای از مصاحبه با دکتر مصطفی محسن‌وند، محقق یادگیری ماشین در شرکت اپل، در مورد هوش مصنوعی و آینده‌ی آن را خواهیم داشت. امیدواریم از مطالعه این شماره لذت ببرید.

ارادتمند شما؛ فاطمه پیمانی، هستی برقراریان، امیرمهدی عسلی و نیما حسینی دشت بیاض

انتقادات و پیشنهادات

https://forms.gle/HeG8ypneJCu82Pbp7

گروه تعیین محتوا و ویرایش علمی

فاطمه پیمانی، هستی برقراریان، نیما حسینی، دانیال کاظم‌لو، حسین رودبارانی، غزاله خرادپور، یحیی پورسلطانی، شروین شفیعی، متین معزی، امیرمهدی عسلی

هیئت تحریریه ریاضی

مسیح مزکی، آرمین احمدخان‌بیگی، هستی برقراریان، شروین شفیعی، محمد سروش غفاری، زهرا کشاورز، فریماه شاه‌محمدیان، یلدا افتخاری

هیئت تحریریه علوم کامپیوتر

محمدرضا باطنی، سبحان مقیمی، امیرحسین رجبی، زهرا سراج، غزاله خرادپور، امیر اسکندری، امیرحسین غزنوی، امیرحسین رجبی، محمدرضا باطنی، فاطمه عبدالحی

گروه مصاحبه

نیما حسینی، مهلا کریمیان، امیرحسین فروغی، یلدا افتخاری، احمدرضا بهرامی، ویدا زمانی

گروه ویراستاری نگارشی

پارسا پهلوان، نیما حسینی، زهرا پورشیخ‌علی، طیبه محبی، شروین شفیعی، محمدرضا امانی، یلدا افتخاری، فاطمه پیمانی، آرمین خان‌بیگی، امیرمهدی عسلی

صفحه آرایی

متین معزی، شروین شفیعی، نفیسه آغوئی

طراحی جلد

امیرمهدی عسلی

زمینه‌ی نشر: علمی

صاحب امتیاز: انجمن علمی دانشکده ریاضی و علوم‌کامپیوتر دانشگاه صنعتی امیرکبیر

مدیرمسئول: فاطمه پیمانی

شورای سردبیری: امیرمهدی عسلی، نیما حسینی

دانشگاه صنعتی امیرکبیر(پلی‌تکنیک تهران)

شماره چهارم - ۱۴۰۰/۰۶/۳۱

دوره نشر: فصلنامه

فهرست مطالب

Mathematics

ریـــــــاضی

تابع پایه ۱۳ کانوی

Conway Base 13 Function

یلدا افتخاری

تابع پایه ۱۳ کانوی، تابعی است که توسط ریاضی‌دان انگلیسی جان کانوی [1] به عنوان مثال نقضی برای عکس قضیه مقدار میانی ارائه شد. به بیان دیگر، تابعی است که ویژگی مشخصی از مقدار میانی را دارد (در هر بازه (a,b) تابع f تمام مقادیر بین $f(a)$ و $f(b)$ را می‌گیرد.) ولی پیوسته نیست.

اما دلیل به وجود آمدن این تابع چیست؟ مثال نقض‌های دیگری نیز برای عکس قضیه مقدار میانی وجود دارد؛ مثلاً تابع $f:\mathbb{R}\to\mathbb{R}$ یکی از این مثال‌ها است:

$$f(x):=\begin{cases}\sin(\frac{1}{x}) & x\neq 0\\ 0 & x=0\end{cases}$$

اگرچه تابع ارائه‌شده مثال نقض است ولی می‌توان دید که مثال ضعیفی است؛ زیرا تابع f فقط در $x=0$ ناپیوسته است.

تابع پایه ۱۳ کانوی به همین منظور ارائه شد. در این روند، چالش این بود که تابعی ساده و قابل‌درک ساخته شود که هر مقدار حقیقی در هر بازه دل‌خواه را اخذ کرده و در کل بازه پوشا و در تمام نقاط نیز ناپیوسته نیز باشد.

هر عدد حقیقی مانند x را می‌توان به صورت یکتا در پایه ۱۳ نشان داد. در این نمایش از ارقام ۰ تا ۹ و سه نماد A,B,C استفاده می‌کنند. برای مثال، عدد ۵۴۳۴۹۵۸۹ در پایه ۱۳ به صورت B۳۴C۱۲۸ نمایش داده می‌شود.

اگر به‌جای A، B، C به ترتیب از علائم +، −، . استفاده کنیم اتفاق جالبی می‌افتد؛ بعضی از اعداد در پایه ۱۳ نمایشی مانند اعداد اعشاری در پایه ۱۰ خواهند داشت. برای مثال، عدد ۵۴۳۴۹۵۸۹ در پایه ۱۳ نمایشی به صورت ۱۲۸.۳۴− دارد. البته بسیاری از اعداد در این پایه نمایش معقولی نخواهند داشت؛ برای مثال، عدد ۳۶۲۹۲۶۵ در پایه ۱۳ به صورت $9+0--7$ خواهد بود.

تابع پایه ۱۳ کانوی عدد حقیقی x را دریافت می‌کند و نمایش آن در پایه ۱۳ را به شکل دنباله‌ای از نمادهای ۰، ۱، ...، ۹، +، −، . در نظر می‌گیرد. اگر از یک موقعیت به بعد نمایش به شکل خوش‌فرم اعشاری r بود، $f(x)=r$ و در غیر این صورت $f(x)=0$ می‌باشد. (منظور از خوش‌فرم بودن عدد اعشاری r این است که از جایی به بعد، عدد با علامت + یا − آغاز شود و حداکثر یک نقطه اعشار داشته باشد و در ادامه فقط ارقام ۰ تا ۹ را داشته باشد.) برای مثال، عدد x نمایشی به صورت $8++2.19+07+3.141592653...$ دارد و در نتیجه $f(x)=+3.141592653....$

تابع پایه ۱۳ کانوی تابعی است $f:\mathbb{R}\to\mathbb{R}$ که به شکل زیر تعریف می‌شود:

x را در پایه ۱۳ بنویسید. اگر از جایی به بعد نمایش x در پایه ۱۳ به فرم $Ax_1x_2...x_nCy_1y_2...$ بود و تمام x_i ها و y_i ها ارقام ۰ تا ۹ باشند، آن‌گاه $f(x)=x_1...x_n.y_1...y_n$ در پایه ۱۰ خواهد بود. به‌طور مشابه اگر نمایش پایه ۱۳ عدد x به شکل $Bx_1x_2...x_nCy_1y_2...$ باشد، آن‌گاه $f(x)=-x_1...x_n.y_1...y_n$ و در غیر این‌صورت $f(x)=0$ خواهد بود. (دقت کنید که مقدار $f(x)$ در پایه ۱۰ است نه، ۱۳.) برای مثال:

$$f(12345A3C14159...{}_{13})=$$
$$f(A3C14159...{}_{13})=3.14159...,$$
$$f(B1C234_{13})=-1.234,$$
$$f(1C234A567_{13})=0.$$

براساس قضیه مقدار میانی، هر تابع حقیقی پیوسته f دارای ویژگی مقدار میانی است؛ در هر بازه (a,b) ، تابع f از هر نقطه بین $f(a)$ و $f(b)$ می‌گذرد. تابع پایه ۱۳ کانوی نشان می‌دهد که عکس قضیه غلط است؛ عکس این قضیه ویژگی مقدار میانی را برآورده می‌کند ولی پیوسته نیست.

همانطور که گفته شد، تابع پایه ۱۳ کانوی ویژگی بسیار قویتری از مقدار میانی را در هر بازه (a,b) دارا است. تابع f از هر عدد حقیقی می‌گذرد. این تابع ناپیوستگی قوی‌تری دارد و در همه‌جا ناپیوسته است.

برای اثبات اینکه تابع پایه ۱۳ کانوی این ویژگی قوی‌تر مقدار میانی را برآورده می‌کند، بازه (a,b)، نقطه‌ی c در این بازه و عدد دل‌خواه و حقیقی r را در نظر بگیرید. نمایش r در پایه ۱۳ را به این شکل بسازید: با نمایش پایه ۱۰ r شروع کنید، نقطه اعشار را با C جایگزین کرده و سپس اگر r مثبت بود A و اگر r منفی بود B را به ابتدای آن اضافه کنید. از تعریف تابع پایه ۱۳ کانوی، رشته حاصل شده $\hat{r}$ این ویژگی را

[1] John H. Conway

دارد که $f(\hat{r}) = r$. علاوه بر این، هر رشته در پایه ۱۳ که به $\hat{r}$ ختم شود این ویژگی را خواهد داشت؛ بنابراین اگر ما $\hat{r}$ را به انتهای c اضافه کنیم برای عدد حاصل شده $f(\hat{c}) = r$.

با معرفی این تغییر برای نمایش پایه ۱۳ عدد c، می‌توان اطمینان حاصل کرد که عدد جدید $\hat{c}$ همچنان در بازه (a, b) قرار خواهد داشت. این اثبات می‌کند که برای هر عدد r در هر بازه‌ای می‌توان نقطه c' را طوری پیدا کرد که $f(c') = r$ باشد.

برای نشان دادن ناپیوستگی تابع پایه ۱۳ کانوی به صورت زیر عمل می‌کنیم:

فرض کنید $f : \mathbb{R} \to \mathbb{R}$ تابعی با ویژگی $(\star)$ باشد (براساس این ویژگی برای هر بازه باز $I \subseteq \mathbb{R}$ داریم $f[I] = \mathbb{R}$. می‌توان ثابت کرد که چنین تابعی وجود دارد). با برهان خلف ادعا می‌کنیم $x \in \mathbb{R}$ وجود دارد که f در x پیوسته‌است. $V = B_1(f(x))$ گوی بازی به شعاع یک حول $f(x)$ در نظر بگیرید. در این‌صورت همسایگی باز U حول x وجود دارد به طوری که $f[U] \subseteq V$. چون f تابعی با ویژگی $(\star)$ بود، واضح است که $f[U] = \mathbb{R}$ که این یعنی $V \subseteq \mathbb{R}$ و به تناقض می‌رسیم. بنابراین، تابع پایه ۱۳ کانوی در تک‌تک نقاط ناپیوسته است.

مقدمه‌ای بر خمینه‌ها - بخش دوم

Introduction to Manifolds(2)

شروین شفیعی

پیش از شروع بحث، لازم است اشاره شود بخش نخست این نوشتار در یادداشتی با عنوان «مقدمه‌ای بر خمینه‌ها - بخش اول» در شماره‌ی قبل نشریه‌ی حلقه (بهار ۱۴۰۰) چاپ شده‌است. توصیه می‌شود پیش از مطالعه‌ی این یادداشت، یادداشت پیشین را به خوبی مطالعه فرمایید. بدیهی‌ست فهم مطالب ذکرشده در این نوشتار نیاز به یادگیری مفاهیم تعریف‌شده در نوشتار قبلی دارد.

یادآوری

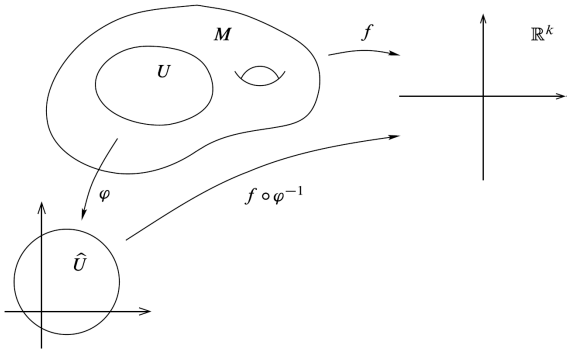
در بخش قبل به تعاریف اولیه و اصلی یک خمینه اشاره کرده و مثال‌ها و نامثال‌های گوناگونی نیز از خمینه‌های توپولوژیک و هموار آورده شد. یادآوری می‌کنیم یک خمینه‌ی هموار در حقیقت یک ساختار دیفئومورفیسم روی یک شیء هندسی بود که این امکان را برای ما فراهم می‌کرد که بتوانیم آن شیء ناشناخته را با ابزار فضای کاملاً شناس اقلیدسی بررسی کنیم. اشاره کردیم اگر به عنوان یک ناظر روی نقاط یک خمینه‌ی هموار رفته و اطرافمان را نگاه کنیم، فضایی هم‌سان‌ریخت با $\mathbb{R}^n$ مشاهده خواهیم کرد. همین ویژگی مهم و اساسی به ما کمک می‌کرد تا سایر رفتارهای یک خمینه‌ی هموار را نیز از طریق همین ارتباطش با فضای اقلیدسی شناسایی کنیم. برای روشن‌تر شدن بحث، به سراغ تعاریف اولیه‌ی موجود در ریاضی عمومی روی خمینه‌ها رفته و مشخص می‌کنیم مفاهیمی مانند حد و پیوستگی یا مشتق برای خمینه‌ها چگونه تعریف می‌شوند.

ریاضی عمومی روی خمینه‌ها

توابع هموار[1]

جالب است بدانیم علت اصلی معرفی خمینه‌های هموار ایجاد توانایی برای تعریف نگاشت‌ها و توابع هموار روی خمینه‌ها بود. در این بخش می‌خواهیم به سراغ این دو تعریف مهم برویم.

تعریف ۱.۲. فرض کنیم M یک خمینه هموار n-بعدی، k یک عدد صحیح نامنفی و $f : M \to \mathbb{R}^k$ یک تابع دلخواه است. در این صورت f را یک **تابع هموار** گوییم اگر $\forall p \in M$، یک کارت هموار (U, φ) برای M موجود باشد به طوری که $p \in U$ و تابع ترکیب $f \circ \varphi^{-1}$ روی باز $\tilde{U} = \varphi(U) \subseteq \mathbb{R}^n$ یک تابع هموار باشد. در یادداشت قبل اشاره کردیم منظور ما از تابع هموار، تابعی‌ست که هرکدام از توابع جزئی‌اش دارای مشتق جزئی موجود و پیوسته از مرتبه‌ی بی‌نهایت باشد. نگاره‌ی زیر تعریف فوق را به خوبی مشخص می‌کند:



شکل ۱: تعریف توابع هموار - [۱]

توجه کنید تابع ترکیب $f \circ \varphi^{-1}$ یک تابع مابین دو فضای اقلیدسی با ابعاد مختلف (یا برابر) بوده و نحوه‌ی کارکرد و ویژگی‌های آن برای ما کاملاً مشخص است. در حقیقت، قدرت تعریف فوق در انتقال تابع ناشناخته‌ی f به تابع کاملا شناخته‌شده‌ی ترکیب با دامنه و بُردی از فضای اقلیدسی‌ست. به اضافه، توجه کنید چون طبق تعریف خمینه‌های هموار انتخاب کارت برای ما اهمیت نداشت، تعریف توابع هموار نیز مستقل از انتخاب کارت است؛ یعنی اگر $g : M \to \mathbb{R}^k$ یک تابع هموار باشد، تابع $g \circ \varphi^{-1}$ با هر کارت (U, φ) از M هموار است. اثبات این حقیقت به عنوان یک تمرین مفید به خواننده واگذار می‌شود.

تعریف ۲.۲. فرض کنید تابع $f : M \to \mathbb{R}^k$ و کارت (U, φ) برای M داده شده‌است. تابع $\tilde{f} : \varphi(U) \to \mathbb{R}^k$ تعریف‌شده به صورت $\tilde{f}(x) = f \circ \varphi^{-1}(x)$ را **نمایش مختصاتی** f[2] می‌نامند. طبق تعریف، تابع f هموار است اگر و تنها اگر نمایش مختصاتی آن حول هر نقطه یا یک کارت هموار، هموار باشد؛ یعنی می‌توان نشان داد توابع هموار حول هر کارت هموار دارای نمایش مختصاتی هموار نیز هستند.

مثال ۳.۲. یادآوری می‌شود هر تابع $f : A \to \mathbb{R}$ را یک **تابع**

[1] Smooth Function

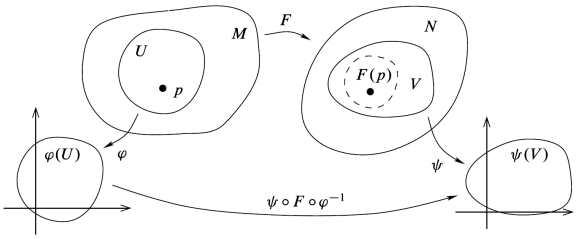
[2] Coordinate Representation of f

حقیقی‌مقدار[3] گوییم زیرا دارای بُردی در اعداد حقیقی است. مهم‌ترین مثال خاص یک تابع هموار نیز توابع هموار حقیقی‌مقدار به شکل $g : M \to \mathbb{R}$ بوده که M یک خمینه است. مجموعه‌ی چنین توابعی با $C^\infty(M)$ نمایش داده می‌شود. چون جمع دو تابع هموار و ضرب اعداد ثابت در آن‌ها باز هم یک تابع هموار می‌شود، پس مجموعه‌ی $C^\infty(M)$ یک فضای برداری روی $\mathbb{R}$ است.

نگاشت‌های هموار بین خمینه‌ها[4]

واضح است تعریف توابع هموار روی خمینه‌ها به راحتی می‌تواند به مفهوم نگاشت‌های بین خمینه‌ها تعمیم یابد؛ یعنی تکنیک و روشی که برای تعریف نگاشت‌ها استفاده می‌کنیم، کاملاً مشابه تعریف قبل است.

تعریف ۴.۲. فرض کنید M, N دو خمینه‌ی هموار مفروض و $F : M \to N$ نگاشت دلخواهی بین این دو خمینه است. گوییم F یک **نگاشت هموار** است اگر برای هر $p \in M$ کارت هموار (U, φ) شامل p و (V, ψ) شامل $F(p)$ موجود باشد به طوری که $F(U) \subseteq V$ و تابع ترکیب $\psi \circ F \circ \varphi^{-1}$ یک تابع هموار از $\varphi(U)$ به $\psi(V)$ باشد. نگاره‌ی زیر این تعریف را به خوبی مشخص می‌کند:



شکل ۲: تعریف نگاشت‌های هموار - [۱]

مشابه تعریف قبل، تابع ترکیب $\psi \circ F \circ \varphi^{-1}$ یک تابع مابین دو فضای اقلیدسی بوده و نحوه‌ی کار کردن با آن برای ما کاملاً مشخص است. به عنوان چند مثال مهم از نگاشت‌های هموار، می‌توان به نگاشت ثابت $c : M \to N$ یا نگاشت همانی اشاره کرد. هموار بودن هر دوی آن‌ها طبق تعریف فوق و با یک محسابات ساده به دست می‌آید. به اضافه، مستقیماً از تعریف، چون این تابع ترکیب هموار است، می‌توان قضیه‌ی زیر را نتیجه گرفت:

گزاره ۵.۲. *هر نگاشت هموار پیوسته است.*

اثبات. فرض کنید M, N دو خمینه‌ی هموار و $F : M \to N$ نگاشت هموار بین این دو خمینه است. نگاشت هموار است؛ پس برای هر $p \in M$ کارت هموار (U, φ) شامل p و (V, ψ) شامل $F(p)$ موجود است، به طوری که $F(U) \subseteq V$ و تابع ترکیب $\psi \circ F \circ \varphi^{-1}$ یک تابع هموار از $\varphi(U)$ به $\psi(V)$ است. می‌دانیم طبق تعریف خمینه‌های هموار نگاشت‌های $\varphi : U \to \varphi(U)$ و $\psi : V \to \psi(V)$ هم‌سان‌ریختی هستند. پس داریم:

$$F|_U : \psi^{-1} \circ (\psi \circ F \circ \varphi^{-1}) \circ \varphi : U \to V$$

ترکیب پیوسته‌ی چند نگاشت پیوسته است؛ پس خودش پیوسته است. چون F حول هر نقطه پیوسته است، روی M پیوسته است. □

تعریف ۶.۲. فرض کنید نگاشت $f : M \to N$ و کارت‌های (U, φ) و (V, ψ) به ترتیب برای M و N داده شده‌است. نگاشت $\tilde{F} : \psi \circ F \circ \varphi^{-1}$ را **نمایش مختصاتی** F[5] می‌نامند. طبق تعریف، نمایش مختصاتی F حول هر کارت هموار M و N هموار است.

گزاره ۷.۲. *فرض کنید نگاشت $F : M \to N$ و نگاشت $G : N \to P$ هموار باشند. در این صورت نگاشت ترکیب $G \circ F : M \to P$ هموار است.*

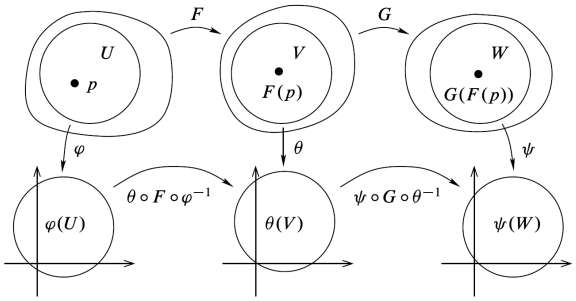
اثبات. فرض کنید $p \in M$ موجود باشد. طبق تعریف هموار بودن نگاشت G، کارت‌های (V, θ) شامل $F(p)$ و (W, ψ) شامل $G(F(p))$ موجود هستند، به طوری که $G(V) \subseteq W$ و نگاشت $\psi \circ G \circ \theta^{-1} : \theta(V) \to \psi(W)$ هموار است. نگاشت F هموار است؛ پس طبق گزاره‌ی قبل پیوسته هم است؛ یعنی $F^{-1}(V)$ یک باز (همسایگی) از p در M است. در این صورت، کارت (U, φ) برای خمینه‌ی M چنان موجود است که $p \in U \subseteq F^{-1}(V)$. طبق تعریف قبل (۶.۲)، نگاشت $\theta \circ F \circ \varphi^{-1} : \varphi(U) \to \theta(V)$ نیز هموار است. در این صورت داریم $G \circ F(U) \subseteq G(V) \subseteq W$ و نگاشت

$$\psi \circ (G \circ F) \circ \varphi^{-1} =$$
$$(\psi \circ G \circ \theta^{-1}) \circ (\theta \circ F \circ \varphi^{-1}) : \varphi(U) \to \psi(W)$$

هموار است؛ زیرا ترکیب چند نگاشت هموار مابین زیرمجموعه‌هایی از فضاهای اقلیدسی‌ست. برای درک بهتر اثبات نگاره‌ی ۳ را مشاهده کنید. □

بلافاصله پس از تعریف توابع و نگاشت‌های هموار، ویژگی‌های موضعی این توابع از قبیل حد و پیوستگی نیز برای ما جالب می‌شوند. اکنون می‌خواهیم به سراغ تعریف این دو مفهوم روی خمینه‌ها برویم.

[3] Real-Valued Function
[4] Smooth Maps Between Manifolds
[5] Coordinate Representation of F



شکل ۳: ترکیب چند نگاشت هموار، هموار است - [۱]

حد و پیوستگی

در این بخش به طور طبیعی تعریفی از حد و پیوستگی توابع روی خمینه‌ها را بیان می‌کنیم. در حقیقت تعریف حد و پیوستگی روی خمینه را با استفاده از کارت‌ها به روی $\mathbb{R}^n$ منتقل نموده و از تعریف حد و پیوستگی در این فضا بهره می‌بریم. با استفاده از خواص همئومورفیسم بودن کارت‌ها و تعاریف حد و پیوستگی توابع حقیقی چندمتغییره روی $\mathbb{R}^n$ این مفاهیم به راحتی قابل بیان است.

تعریف ۸.۲. فرض کنیم M یک خمینه هموار n-بعدی، k یک عدد صحیح نامنفی و $f : M \to \mathbb{R}^k$ یک تابع دلخواه است. در این صورت می‌گوییم تابع f در نقطه‌ی p دارای حد $L \in \mathbb{R}^k$ است اگر کارت هموار (U, φ) برای M موجود باشد به طوری که $p \in U$ و تابع ترکیب $f \circ \varphi^{-1} : \mathbb{R}^n \to \mathbb{R}^k$ با تعاریف شناخته‌شده‌ی ما در ریاضیات عمومی در نقطه‌ی $\varphi(p)$ دارای حد باشد؛ به عبارت دیگر

$$\lim_{\varphi(x)\to\varphi(p)} f \circ \varphi^1(\varphi(x)) = L$$

که یعنی:

$$\forall \epsilon > 0, \exists \delta > 0, \ ||\varphi(x) - \varphi(p)|| < \delta \Rightarrow$$
$$||f \circ \varphi^{-1}(\varphi(x)) - L|| < \epsilon$$

توجه داریم برای تعریف حد نیز از تابع ترکیب و فضای شناخته‌شده‌ی اقلیدسی بهره گرفتیم. در حقیقت، این یک راهکار رایج و ساده برای بیان مفاهیم روی خمینه‌هاست. به اضافه، انتظار داریم این تعریف نیز همانند تعاریف قبلی مستقل از انتخاب کارت باشد. این نیز یک گزاره‌ی درست است. به عنوان دو تمرین مفید دیگر اثبات کنید وجود حد در یک نقطه بستگی به انتخاب کارت ندارد و همچنین حد در صورت وجود یکتاست.

تعریف ۹.۲. فرض کنید M, N دو خمینه‌ی هموار مفروض و $F : M \to N$ نگاشت دلخواهی بین این دو خمینه است. در این صورت می‌گوییم تابع F در نقطه‌ی $p \in M$ دارای حد $L \in N$ است اگر کارت هموار (U, φ) شامل p و (V, ψ) شامل $F(p)$ موجود باشد به طوری که $F(U) \subseteq V$ و تابع ترکیب $\psi \circ F \circ \varphi^{-1}$ در نقطه‌ی $\varphi(p)$ دارای حد با تعریف شناخته‌شده‌ی ما در ریاضی عمومی باشد.

به وضوح، این تعریف نیز مستقل از انتخاب کارت است. همچنین، پیوستگی توابع و نگاشت‌ها روی خیمنه‌ها نیز دقیقاً مشابه حالات قبل تعریف شده و ویژگی‌های موجود در تعریف حد، در تعریف پیوستگی نیز صادق است.

بردار و فضای مماس بر خمینه‌ها[6]

پس از تعریف مفاهیم حد و پیوستگی، اکنون به سراغ یکی از مهم‌ترین و اساسی‌ترین مفاهیم موجود در خمینه‌ها می‌رویم. می‌دانیم پس از تعریف حد و پیوستگی، مفهوم مشتق و کاربردش روی خمینه‌ها برای ما حائز اهمیت می‌شود. نحوه‌ی تعریف این مفهوم کلیدی کمی با تعاریف قبلی تفاوت دارد. وقتی در رویه‌های تعریف‌شده روی فضای اقلیدسی بحث مشتق را مطرح می‌کنیم، در حقیقت منظور ما بررسی بردار مماسی‌ست که در آن نقطه بر روی رویه تشکیل می‌شود. با اضافه، این بردارها تشکیل صفحات مماسی بر روی رویه نیز می‌دهند. هدف این بخش تعریف این مفاهیم کلیدی روی خمینه‌هاست. برای تعریف، ما از دو راه مستقیم و غیرمستقیم استفاده می‌کنیم که در تعریف غیرمستقیم مشتق روی خمینه‌ها، از کاربرد خم‌ها بهره می‌بریم.

تعریف ۱۰.۲. فرض کنید M یک خمینه‌ی هموار و $p \in M$ نقطه‌ای روی این خمینه باشد. در این صورت نگاشت خطی $v : C^\infty(M) \to \mathbb{R}$ را یک **مشتق‌گیری**[7] روی خمینه در نقطه‌ی p می‌نامیم اگر شرط زیر برقرار باشد: [8]

$$v(fg) = f(p)vg + g(p)vf \quad \forall f, g \in C^\infty(M)$$

ممکن است در نگاه اول این تعریف کمی گنگ به نظر برسد. البته برای کسانی که درس ریاضی ۳ را گذرانده‌اند، این تعریف کاملاً آشنا است. در حقیقت جنس v یک عملگر است و روی توابع در هر نقطه عمل می‌کند. خروجی این عملگر در هر نقطه نیز عضوی از اعداد حقیقی‌ست. ساده‌ترین نوع این عملگر همان مشتق عادی در ریاضی ۱ است که روی توابع حقیقی عمل کرده و در هرنقطه یک مقدار را به

[6] Tangent Vector and Tangent Space
[7] Derivation at p
[8] طبق تعریف آورده‌شده در مثال ۳.۲ این نمادِ مجموعه‌ی تمام توابع حقیقی‌مقدار روی خیمنه است. پس ورودی نگاشت خطی فوق یک تابع است.

عنوان خروجی تحویل می‌دهد. در درس ریاضی ۲ با نوع دیگری از این عملگرها به نام مشتق سویی آشنا شده‌اید. مشتق سویی نیز نوعی عملگر مشتق است که در خواص بالا صدق می‌کند. پس در نتیجه منظور ما از عبارات vg و vf در حقیقت همان خروجی عملگر v روی توابع g و f است. در این صورت، مجموعه‌ی تمام مشتق‌گیری‌های $C^\infty(M)$ در نقطه‌ی p که با T_pM نشان داده می‌شود یک فضای برداری‌ست که آن را **فضای مماس بر** M **در نقطه‌ی** p[9] می‌نامند. هر عضو v از این مجموعه را نیز یک **بردار مماس در** p[10] می‌گویند. گزاره‌ی زیر نحوه‌ی کارکرد این بردارها را بهتر مشخص می‌کند:

گزاره ۱۱.۲. *فرض کنید M یک خمینه‌ی هموار، $p \in M$، $v \in T_pM$ و $f, g \in C^\infty(M)$ باشد. در این‌صورت داریم:*

- *اگر f تابع ثابت باشد، $vf = 0$.*
- *اگر $f(p) = g(p) = 0$، $v(fg) = 0$.*

همانطورکه اشاره شد و برای درک بهتر، بردارهای مماس را به شکل فلش‌هایی روی نقاط خمینه ببینید که روی فضای مماس نشسته‌اند. تصور این مسئله و داشتن درک هندسی از آن به فهم مبحث بعد کمک می‌کند.

دیفرانسیل یک نگاشت هموار

باید بدانیم علت اصلی معرفی مفهوم بردار و صفحات مماس روی خمینه‌ها کمک به تعریف یکی از پایه‌ای‌ترین مفاهیم موجود روی خمینه‌ها یعنی مشتق نگاشت‌های هموار[11] است. از ریاضیات عمومی به یاد داریم مفهوم مشتق به نوعی ارائه‌ی بهترین تقریب خطی برای توابع بود. در ریاضی ۲ گفتیم برای توابع چندمتغیره، نقش این تقریب خطی را ماتریس ژاکوبین بازی می‌کند که اعضایش همان مشتقات جزئی بودند. در حقیقت، ماتریس ژاکوبین یک نگاشت خطی بود که به یک تابع چندمتغیره و نقطه‌ای مفروض، بهترین تقریب خطی را حول آن نقطه نسبت می‌داد. دروضعیت خمینه‌ها نیز بحث همین نگاشت خطی مطرح است، اما طبیعتاً سخن گفتن از نگاشت خطی بین دو خمینه بی‌مفهوم است. اما در عوض، می‌توانیم این تعریف را روی صفحات مماس داشته باشیم.

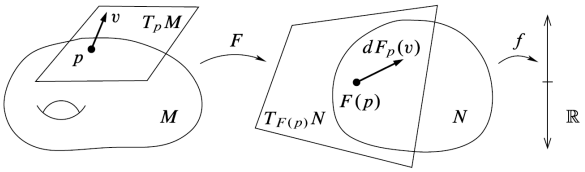
تعریف ۱۲.۲. فرض کنید M, N دو خمینه‌ی هموار و نگاشت $F : M \to N$ هموار باشد. در این صورت برای هر $p \in M$ نگاشتی که به صورت

$$dF_p : T_pM \to T_{F(p)}N$$

تعریف می‌شود را **دیفرانسیل** F **در نقطه‌ی** p[12] می‌نامیم. فرض کنید $v \in T_pM$، در این صورت $dF_p(v)$ یک مشتق‌گیری در $F(p)$ بوده و روی $f \in C^\infty(N)$ به صورت زیر عمل می‌کند:

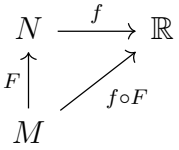
$$dF_p(v)(f) = v(f \circ F).$$

برای درک بهتر تعریف فوق به نگاری زیر توجه کنید:



شکل ۴: دیفرانسیل یک نگاشت هموار - [۱]

در حقیقت چون $f \in C^\infty(N)$ داریم:



چون نگاشت $f \circ F$ یک نگاشت ازخمینه‌ی M به $\mathbb{R}$ است یعنی $f \circ F \in C^\infty(M)$ پس v به عنوان عضوی از فضای مماس M می‌تواند روی آن عمل کند و تعریف فوق در حقیقت قابل تعریف است. چون v یک عملگر خطی‌ست، پس عملگر $dF_p(v) : C^\infty(N) \to \mathbb{R}$ نیز خطی‌ست. به علاوه این عملگر واقعاً یک مشتق‌گیری در $F(p)$ است؛ زیرا برای هر نگاشت $f, g \in C^\infty(N)$ داریم:

$$dF_p(v)(fg) = v\Big((fg) \circ F\Big) =$$
$$v\Big((f \circ F)(g \circ F)\Big) =$$
$$f \circ F(p)v(g \circ F) + g \circ F(p)v(f \circ F) =$$
$$f\Big((F(p)\Big)dF_p(v)(g) + g\Big(F(p)\Big)dF_p(v)(f).$$

گزاره ۱۳.۲. *فرض کنید M, N, P سه خمینه‌ی هموار، نگاشت‌های $F : M \to N$ و $G : N \to P$ هموار و $p \in M$. در این صورت داریم:*

(i) $dF_p : T_pM \to T_{F(p)}N$ *خطی‌ست.*

[9] Tangent Space to M at p.
[10] Tangent Vector at p
[11] The Differential of Smooth Maps
[12] Differential of F at p

(ii)

$$d(G\circ F)_p = dG_{F(p)}\circ dF_p : T_pM \to T_{G\circ F(p)}P.$$

(iii) $d(Id_M)_p = Id_{T_pM} : T_pM \to T_pM.$

(iv) اگر F دیفئومورفیسم باشد (دوسویی و دارای وارون هموار)، نگاشت $dF_p : T_pM \to T_{F(p)}N$ همسان‌ریختی‌ست.

گزاره ۱۴.۲. *فرض کنید M یک خمینه‌ی هموار، $U \subseteq M$ یک زیرمجموعه‌ی باز این فضا و $\iota : U \hookrightarrow M$ نگاشت شمول باشد؛ یعنی $\iota(x) = x$. در این صورت برای هر $p \in M$ نگاشت دیفرانسیل $d\iota_p : T_pU \to T_pM$ یک همسان‌ریختی‌ست.*

همانطور در گزاره‌های قبل مشخص است، ویژگی‌های فضای مماس همانند تعریف نگاشت مشتق در فضای اقلیدسی است. تا اینجا، ما با خود نگاشت و نحوه‌ی کارکردش آشنا شدیم. حال، می‌خواهیم به سراغ نحوه‌ی تعریف و کارکرد این نگاشت روی کارت‌های مختصاتی برویم. توجه داریم فضای مماس برای هرنقطه روی خمینه تعریف می‌شود در صورتی که کارت‌های مختصاتی روی بازهای حول نقطه عمل می‌کنند. این مسئله ممکن است تعریف ما را در دو باز حول نقطه که دارای اشتراکند دچار مشکل کند. برای رفع این مشکل، گزاره‌ی زیر را مطرح می‌کنیم:

گزاره ۱۵.۲. *فرض کنید M یک خمینه‌ی هموار، $p \in M$، $v \in T_pM$ و $f, g \in C^\infty(M)$ به صورتی باشد که حول یک همسایگی نقطه p هم‌ارز باشند؛ یعنی مثلاً باز U موجود است به طوری که $f|_U = g|_U$. در این صورت داریم $vf = vg$.*

گزاره‌ی قبل نشان می‌دهد بردارهای مماس به صورت موضعی عمل می‌کنند و خیالمان بابت مشکل مطرح‌شده راحت است. گزاره‌ی آخر این بخش مربوط به بعد این فضای مماس است.

گزاره ۱۶.۲. *فرض کنید M یک خمینه‌ی هموار با بعد n است. در این صورت برای هر $p \in M$، بعد فضای مماس T_pM نیز برابر با n است.*

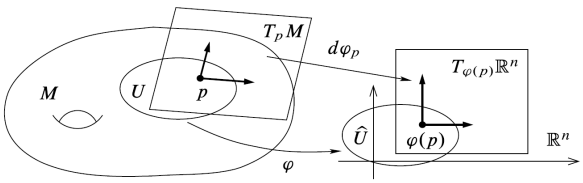
محاسبات روی کارت‌های مختصاتی

مفهوم دیفرانسیل و فضای مماسی که برای خمینه‌ها تعریف کردیم بسیار مجرد و انتزاعی و احتمالاً خسته‌کننده بود. در این بخش می‌خواهیم سراغ محاسبات رفته و کمی روی زمین بیاییم!

پس فرض کنیم M یک خمینه‌ی هموار و (U, φ) یک کارت هموار مختصاتی آن باشد. طبق تعریف، φ یک دیفئومورفیسم از باز $U \subseteq M$ به باز $\tilde{U} \subseteq \mathbb{R}^n$ است. طبق مورد ۴ گزاره‌ی ۲.۴، $d\varphi_p : T_pU \to T_{\varphi(p)}\mathbb{R}^n$ یک هم‌سان‌ریختی‌ست. در نهایت با استفاده از گزاره‌ی ۳.۴ نتیجه می‌گیریم نگاشت دیفرانسیل $d\varphi_p : T_pM \to T_{\varphi(p)}\mathbb{R}^n$ یک هم‌سان‌ریختی‌ست. این یک نتیجه‌ی بسیار مهم است؛ زیرا نشان می‌دهد یک تناظر ۱ به ۱ مابین اجزای فضای مماس خمینه و فضای مماس اقلیدسی برقرار است. از ریاضی ۳ به خاطر داشتیم که مجموعه‌ی

$$\mathcal{B} = \{\partial/\partial x^1|_{\varphi(p)}, ..., \partial/\partial x^n|_{\varphi(p)}\}$$

یک پایه برای فضای $T_{\varphi(p)}\mathbb{R}^n$ است. پس می‌توان نتیجه گرفت تصویروارون هر کدام از این بردارها تحت هم‌سان‌ریختی $d\varphi_p$ تشکیل یک پایه برای فضای مماس T_pM می‌دهند. نگاره‌ی زیر این مسئله را بهتر مشخص می‌کند:



شکل ۵: بردارهای مماس در مختصات - [۱]

منظور ما از عملگر فوق روی خمینه به این صورت است:

$$\left.\frac{\partial}{\partial x^i}\right|_p = (d\varphi_p)^{-1}\left(\left.\frac{\partial}{\partial x^i}\right|_{\varphi(p)}\right) = d\left(\varphi^{-1}\right)_{\varphi(p)}\left(\left.\frac{\partial}{\partial x^i}\right|_{\varphi(p)}\right)$$

نحوه‌ی عمل کردن آن روی یک تابع $f \in C^\infty(U)$ نیز به صورت مشابه تعاریف قبل است:

$$\left.\frac{\partial}{\partial x^i}\right|_p f = \left.\frac{\partial}{\partial x^i}\right|_{\varphi(p)}\left(f\circ\varphi^{-1}\right) = \frac{\partial \tilde{f}}{\partial x^i}(\tilde{p})$$

که در آن $\tilde{f} = f\circ\varphi^{-1}$ نمایش مختصاتی تابع f و $\tilde{p} = (p^1, ..., p^n) = \varphi(p)$ نمایش مختصاتی نقطه‌ی p است. در حقیقت عملگر $\partial/\partial x^i|_p$ یک مشتق‌گیری‌ست که i-امین مشتق جزئی نمایش مختصاتی f را در نمایش مختصاتی p محاسبه می‌کند. بردارهای $\partial/\partial x^i|_p$ را **بردارهای مختصاتی در** p[۱۳] وابسته به سیستم مختصات

[13] Coordinate Vectors at p

داده‌شده می‌نامند. در حالت خاصی که خمینه‌ی ما $\mathbb{R}^n$ باشد، این بردارها همان عملگرهای شناخته‌شده‌ی مشتقات جزئی در ریاضی ۲ هستند. گزاره‌ی زیر نتیجه تمامی استدلال‌های فوق است:

گزاره ۱۷.۲. *فرض کنید M یک خمینه با بعد n و $p \in M$ باشد. در این صورت، T_pM نیز یک فضای برداری با بعد n بوده و برای هر کارت هموار $(U, (x^i))$ شامل p، بردارهای مختصاتی $\partial/\partial x^1|_p, ..., \partial/\partial x^n|_p$ تشکیل یک پایه برای T_pM می‌دهند؛ یعنی هر بردار مماس $v \in T_pM$ به صورت یکتا توسط ترکیب خطی زیر نوشته می‌شود:*

$$v = v^i \left.\frac{\partial}{\partial x^i}\right|_p .$$

*پایه‌ی مرتب $(\partial/\partial x^i|_p)$ را یک **پایه‌ی مختصاتی برای** T_pM[14] و اعداد $(v^1, ..., v^n)$ را **اجزای** v[15] نسبت به پایه‌ی مختصاتی می‌نامند. اگر بردار v شناخته‌شده باشد، محاسبه‌ی اجزای آن از طریق عملش روی توابع مختصاتی بسیار راحت است. در حقیقت به ازای هر j، اجزای v از طریق رابطه‌ی $v^j = v(x^j)$ قابل محاسبه است. (در این‌جا x^j به عنوان یک یک تابع هموار حقیقی‌مقدار روی U فرض شده‌است.) رابطه‌ی فوق درست است زیرا:*

$$v(x^j) = \left(v^i \left.\frac{\partial}{\partial x^i}\right|_p\right)(x^j) = v^i \frac{\partial x^j}{\partial x^i}(p) = v^j .$$

دیفرانسیل روی کارت‌های مختصاتی

در بخش آخر این قسمت به سراغ بازتعریف مفهوم نگاشت دیفرانسیل روی کارت‌های مختصاتی می‌رویم. پس فرض کنید $F : U \to V$ یک نگاشت هموار باشد. برای شروع تعریف و داشتن یک ایده، ابتدا حالت خاصی را در نظر می‌گیریم که $U \subseteq \mathbb{R}^n$ و $V \subseteq \mathbb{R}^m$ باشد. در این صورت برای هر $p \in U$، ماتریس $dF_p : T_p\mathbb{R}^n \to T_p\mathbb{R}^m$ را باتوجه به پایه‌های استاندارد این فضاها در نظر می‌گیریم. فرض کنیم $(x^1, ..., x^n)$ مختصاتی برای دامنه و $(y^1, ..., y^n)$ مختصاتی برای هم‌دامنه باشد. با استفاده از قانون زنجیره‌ای نحوه‌ی عمل dF_p روی پایه‌های معمول مختصاتی را محاسبه می‌کنیم:

$$\begin{aligned} dF_p\left(\left.\frac{\partial}{\partial x^i}\right|_p\right) f &= \left.\frac{\partial}{\partial x^i}\right|_p (foF) \\ &= \frac{\partial f}{\partial y^i}\Big(F(p)\Big)\frac{\partial F^j}{\partial x^i}(p) \\ &= \left(\frac{\partial F^j}{\partial x^i}(p)\left.\frac{\partial}{\partial y^j}\right|_{F(p)}\right) f. \end{aligned}$$

یعنی داریم:

$$dF_p\left(\left.\frac{\partial}{\partial x^i}\right|_p\right) = \frac{\partial F^j}{\partial x^i}(p)\left.\frac{\partial}{\partial y^j}\right|_{F(p)} .$$

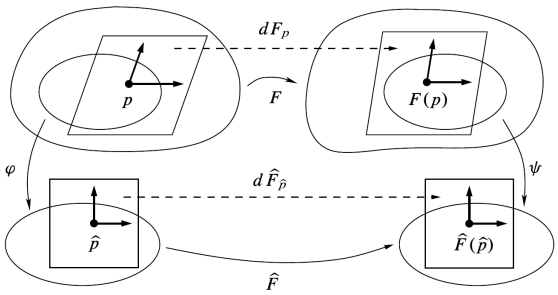
به بیان دیگر، ماتریس dF_p نسبت به پایه‌های مختصاتی عبارت است از:

$$\begin{pmatrix} \frac{\partial F^1}{\partial x^1}(p) & \cdots & \frac{\partial F^1}{\partial x^n}(p) \\ \vdots & \ddots & \vdots \\ \frac{\partial F^m}{\partial x^1}(p) & \cdots & \frac{\partial F^m}{\partial x^n}(p) \end{pmatrix} .$$

این ماتریس همان ماتریس ژاکوبین F در نقطه‌ی p است که ماتریس نمایش مشتق کلی $DF(p) : \mathbb{R}^n \to \mathbb{R}^m$ نیز می‌باشد. پس در این حالت خاص، نگاشت دیفرانسیل $dF_p : T_p\mathbb{R}^n \to \mathbb{R}^m$ همان مشتق کلی را مشخص می‌کند. حال با این مشاهده‌ی مفید، به سراغ حالت عمومی برویم. پس فرض کنید $F : M \to N$ یک نگاشت هموار بین دو خمینه‌ی هموار باشد. کارت هموار (U, φ) شامل p از M و (V, ψ) شامل $F(p)$ از N را انتخاب می‌کنیم. در این صورت، نمایش مختصاتی عبارت است از

$$\hat{F} = \psi \circ F \circ \varphi^{-1} : \varphi(U \cap F^{-1}(V)) \to \psi(V).$$

برای درک بهتر به نگاره‌ی زیر توجه کنید:



شکل ۶: دیفرانسیل در مختصات - [۱]

فرض کنید $\hat{p} = \varphi(p)$ نمایش مختصاتی نقطه‌ی p باشد. با توجه

[14] Coordinate Basis For T_pM
[15] Components Of v

به محاسبات قبل، $d\hat{F}_{\hat{p}}$ نسبت به پایه‌های استاندارد مختصاتی توسط ماتریس ژاکوبین $\hat{F}$ در نقطه‌ی $\hat{p}$ نمایش داده می‌شود. با استفاده از رابطه‌ی $F \circ \varphi^{-1} = \psi^{-1} \circ \hat{F}$ خواهیم داشت:

$$\begin{aligned} dF_p\left(\frac{\partial}{\partial x^i}\Big|_p\right) &= dF_p\left(d(\varphi^{-1})_{\hat{p}}\left(\frac{\partial}{\partial x^i}\Big|_{\hat{p}}\right)\right) \\ &= d(\psi^{-1})_{\hat{F}(\hat{p})}\left(d\hat{F}_{\hat{p}}\left(\frac{\partial}{\partial x^i}\Big|_{\hat{p}}\right)\right) \\ &= d(\psi^{-1})_{\hat{F}(\hat{p})}\left(\frac{\partial \hat{F}^j}{\partial x^i}(\hat{p})\frac{\partial}{\partial y^j}\Big|_{\hat{F}(\hat{p})}\right) \\ &= \frac{\partial \hat{F}^j}{\partial x^i}(\hat{p})\frac{\partial}{\partial y^j}\Big|_{F(p)}. \end{aligned}$$

به این ترتیب، ثابت کردیم که dF_p با استفاده از پایه‌های مختصاتی توسط ماتریس ژاکوبینِ نمایش مختصاتی F نشان داده می‌شود. در حقیقت، مفهوم دیفرانسیل طوری تعریف شده بود که به طور مستقل از کارت‌های مختصاتی، به ماتریس ژاکوبین مربوط شود. به زبان هندسه‌ی دیفرانسیل، این دیفرانسیل گاهی نگاشت مماس یه به طور ساده مشتق F نیز نامیده می‌شود. دلیل این نام‌گذاری نیز واضح است؛ زیرا این نگاشت در حقیقت بردارهای مماس موجود در خمینه‌ی دامنه را به هم‌دامنه منتقل می‌کند.

مثال ۱۸.۲. خمینه‌ی هموار $\mathbb{R}^2$ را در نظر بگیرید. نگاشت تغییر کارت مابین مختصات قطبی و مختصات اقلیدسی روی بازهای مناسب فضا به صورت $(x, y) = (r\cos\theta, r\sin\theta)$ تعریف می‌شود. فرض کنید $p \in \mathbb{R}^2$ نقطه‌ای با مختصات قطبی $(r, \theta) = (2, \pi/2)$ و $v \in T_p\mathbb{R}^2$ یک بردار مماس به طوری باشد که نمایش مختصات قطبی آن به صورت زیر است:

$$v = 3\frac{\partial}{\partial r}\Big|_p - \frac{\partial}{\partial \theta}\Big|_p.$$

می‌خواهیم با استفاده از تعاریف قبل، ضابطه‌ی v در مختصات اقلیدسی را بیابیم. با توجه به $(x, y) = (r\cos\theta, r\sin\theta)$ داریم:

$$\begin{aligned} \frac{\partial}{\partial r}\Big|_p &= \cos(\frac{\pi}{2})\frac{\partial}{\partial x}\Big|_p + \sin(\frac{\pi}{2})\frac{\partial}{\partial y}\Big|_p = \frac{\partial}{\partial y}\Big|_p \\ \frac{\partial}{\partial \theta}\Big|_p &= -2\sin(\frac{\pi}{2})\frac{\partial}{\partial x}\Big|_p + 2\cos(\frac{\pi}{2})\frac{\partial}{\partial y}\Big|_p \\ &= -2\frac{\partial}{\partial x}\Big|_p \end{aligned}$$

پس با جایگذاری داریم:

$$v = 3\frac{\partial}{\partial r}\Big|_p - \frac{\partial}{\partial \theta}\Big|_p = 3\frac{\partial}{\partial y}\Big|_p + 2\frac{\partial}{\partial x}\Big|_p$$

بردار سرعت روی خمینه‌ها

بخش آخر این نوشتار را به معرفی منحنی‌ها (خم‌ها) و بردار سرعت روی خمینه‌ها اختصاص داده و با ارائه‌ی تعریف جایگزین دیفرانسیل با کمک منحنی‌ها بحث را به پایان می‌رسانیم.

تعریف ۱۹.۲. فرض کنید M یک خمینه است. نگاشت پیوسته‌ی $\gamma : J \to M$ را یک **منحنی روی** M[16] تعریف می‌کنیم اگر $J \subseteq \mathbb{R}$.

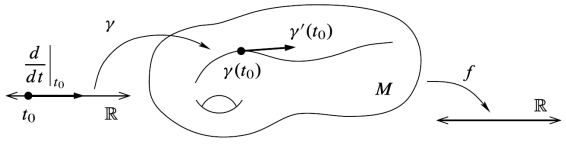
حال فرض کنید M یک خمینه‌ی هموار باشد. نحوه‌ی تعریف ما از فضاهای مماس کار را بری تفسیر مفهوم بردار روی خمینه بسیار آسان می‌کند. فرض کنید $\gamma : J \to M$ یک منحنی هموار روی این خمینه و $t_0 \in J$ باشد. در این حالت، **بردار سرعت** γ **در** t_0[17] که با نماد $\gamma'(t_0)$ نشان داده می‌شود به صورت بردار زیر تعریف می‌شود:

$$\gamma'(t_0) = d\gamma\left(\frac{d}{dt}\Big|_{t_0}\right) \in T_{\gamma(t_0)}M,$$

که $d/dt|_{t_0}$ پایه آشنای مختصاتی استاندارد $T_{t_0}\mathbb{R}$ است. نحوه‌ی عمل این بردار مماس روی توابع $f \in C^\infty(M)$ به صورت زیر است:

$$\begin{aligned} \gamma'(t_0)f = d\gamma\left(\frac{d}{dt}\Big|_{t_0}\right)f &= \frac{d}{dt}\Big|_{t_0}(f\circ\gamma) \\ &= (f\circ\gamma)'(t_0) \end{aligned}$$

نگاره‌ی زیر مفهوم فوق را مشخص می‌کند:



شکل ۷: بردار سرعت منحنی - [۱]

به بیان دیگر، $\gamma'(t_0)$ یک مشتق‌گیری در $\gamma(t_0)$ است که با محاسبه‌ی مشتق تابع مربوط به γ به دست می‌آید.

[16] Curve in M
[17] Velocity of γ at t_0

حال فرض کنید (U, φ) یک کارت مختصاتی هموار با توابع مختصاتی (x^i) باشد. اگر $\gamma(t_0) \in U$ باشد، می‌توان نمایش مختصاتی γ را به صورت $\gamma(t) = (\gamma^1(t), ..., \gamma^n(t))$ نوشت در شرایطی که t به اندازه‌ی کافی به t_0 نزدیک است. در این صورت، فرمول مختصاتی دیفرانسیل به صورت زیر نوشته می‌شود:

$$\gamma'(t_0) = \frac{d\gamma^i}{dt}(t_0) \left.\frac{\partial}{\partial x^i}\right|_{\gamma(t_0)}.$$

فرمول فوق به این مفهوم است که برای محاسبه‌ی $\gamma'(t_0)$ فقط به همان فرمول شناخته‌شده در فضای اقلیدسی نیاز داریم. گزاره‌ی بعد نشان می‌دهد که یک تناظر ما بین بردار سرعت و بردارهای مماس وجود دارد؛ به بیان دیگر هر بردار مماس یک خمینه‌ی مفروض، در حقیقت بردار سرعتی برای یک منحنی دلخواه روی خمینه است. اثبات این گزاره به عنوان یک تمرین با درجه‌ی سختی بالا به خواننده واگذار می‌شود!

گزاره ۲۰.۲. *فرض کنید M یک خمینه و $p \in M$ باشد. هر بردار $v \in T_pM$، بردار سرعت یک منحنی دلخواه روی خمینه است؛ یعنی منحنی $\gamma : J \to M$ موجود است به طوری که $\gamma'(0) = v$.*

توجه کنید به طور قرارداد فرض می‌کنیم بردار مماس، بردار سرعت منحنی دلخواهی در نقطه‌ی صفر است. گزاره‌ی بعد نشان می‌دهد بردارهای سرعت قابل جابه‌جایی ما بین نگاشت‌های هموار هستند:

گزاره ۲۱.۲. *فرض کنید $F : M \to N$ یک نگاشت هموار بین خمینه‌های هموار و $\gamma : J \to M$ یک منحنی هموار روی خمینه‌ی M باشد. به ازای هر $t_0 \in J$، بردار سرعت منحنی ترکیب $F \circ \gamma : J \to N$ در $t = t_0$ به صورت زیر است:*

$$(F \circ \gamma)'(t_0) = d(F(\gamma'(t_0))$$

اثبات.

$$\begin{aligned}(F \circ \gamma)'(t_0) &= d(F \circ \gamma)\left(\left.\frac{d}{dt}\right|_{t_0}\right) \\ &= dF \circ d\gamma \left(\left.\frac{d}{dt}\right|_{t_0}\right) = dF\Big(\gamma'(t_0)\Big)\end{aligned}$$

□

گزاره‌ی قبل یک ابزار مناسب برای محاسبه‌ی بردار سرعت منحنی ترکیب است. اما در حقیقت کاربرد اصلی این گزاره، در جهت عکس و برای محاسبه‌ی دیفرانسیل‌ها است؛ یعنی فرض کنید $F : M \to N$ یک نگاشت هموار باشد و می‌خواهیم نگاشت dF_p را در یک نقطه‌ی دلخواه p محاسبه کنیم. برای این کار کافی‌ست برای $v \in T_pM$ منحنی γ را طوری انتخاب کنیم که بردار سرعتش در نقطه‌ی صفر برابر با v باشد. در این صورت به راحتی می‌توان $dF_p(v)$ را با اعمال گزاره‌ی ۲.۵ روی منحنی ترکیب $F \circ \gamma$ محاسبه کرد. گزاره‌ی آخر این یادداشت، جمع‌بندی همین مطالب است:

گزاره ۲۲.۲. *فرض کنید $F : M \to N$ یک نگاشت هموار بین خمینه‌های هموار، $p \in M$ و $v \in T_pM$ باشد. در این صورت*

$$dF_p(v) = (F \circ \gamma)'(0)$$

برای هر منحنی هموار $\gamma : J \to M$ به طوری که $0 \in J$، $\gamma(0) = p$ و $\gamma'(0) = v$.

گزاره‌ی قبل یک تعریف مناسب و جایگزین برای محاسبه‌ی مشتق F است که با یافتن منحنی با شرایط مورد نظر انجام می‌پذیرد. با نشان‌دادن این راه جیگزین، بحث را به پایان می‌رسانیم.

مراجع

[1] Lee, John M. *Smooth manifolds*. Springer, 2013.

اثبات‌هایی متقاعدکننده و بدون راه حل: مقدمه‌ای بر اثبات‌های هیچ‌آگاهی

مسیح مزکی

حتماً تاکنون توسط استادی نسبت به کامل بودن راه حل اثبات خود گوشزد شده‌اید. در ریاضیات، دقیق بودن اثبات‌ها اهمیت بسیاری دارد و این تأکید بر دقت بی‌دلیل هم نیست؛ به عنوان مثال، اثبات معروف اندرو وایلز[1] برای قضیه آخر فرما برای بار اول در سال ۱۹۹۳ منتشر شد ولی بعد از بررسی در آن خطایی کشف شد که درست کردن آن ۲ سال به طول انجامید. قضیه چهار رنگ هم اثبات‌های اشتباه بسیاری داشته‌است؛ یکی از این اثبات‌ها تا ۲۱ سال به عنوان اثباتی صحیح به شمار می‌رفت!

در تمامی مثال‌های ما و بقیه مثال‌های موجود، این بقیه ریاضی‌دان‌ها هستند که با بررسی ریزبه‌ریز جزئیات یک اثبات سعی بر راستی‌آزمایی آن می‌کنند. به نظر می‌رسد که برای این کار نیاز است تا اول اثبات نوشته شده و به بقیه نشان داده شود؛ ولی آیا راهی هست که بتوان بدون منتشر کردن جزئیات اثبات، گروهی از ریاضی‌دانان شکاک را از درستی اثبات خود مطمئن کرد؟ جواب این مسئله در اثبات‌های هیچ‌آگاهی[2] نهفته است.

اثبات‌های هیچ‌آگاهی برای اولین بار در سال ۱۹۸۵ به وجود آورده شدند. در این تکنیک سعی بر این است تا بدون افشای هرگونه اطلاعاتی در مورد *نحوه به دست آوردن داده*، داشتن آن را اثبات کرد. اثبات‌های هیچ‌آگاهی نمی‌توانند اطمینان کامل به تصدیق‌کننده بدهند؛ به این معنا که یک فرآیند هیچ‌آگاهی تنها می‌تواند به طور *احتمالی* نتایجی را ارائه دهد. همانطور که خواهیم دید، اثبات‌های هیچ‌آگاهی متکی بر آزمون‌هایی هستند که تکرار آن‌ها به طور متوالی باعث افزایش اطمینان به فرد اثبات‌کننده خواهد شد.

کار بر روی اثبات‌های هیچ‌آگاهی یکی از دلایل اهدای جایزه آبل سال ۲۰۲۱ به آوی ویگدرسون [3] است. پاراگراف زیر از بیانیه اعطای جایزه‌ی آکادمی علوم نروژ برداشته شده‌است:

> “Wigderson’s other contributions include zero-knowledge proofs that provide proofs for claims without revealing any extra information besides the claims’ validity, and lower bounds on the efficiency of communication protocols, circuits, and formal proof systems.”

دکتر ویگدرسون به همراهی دو همکار دیگر یکی از پراهمیت‌ترین نتایج مربوط به اثبات‌های هیچ‌آگاهی را به دست آورده‌است که در ادامه به آن خواهیم پرداخت. قبل از آن، بهتر است تا کمی بیشتر با ریشه‌های اثبات هیچ‌آگاهی آشنا شویم.

اثبات چیست؟

اثبات به طور کلی گرفتن نتایج از جملاتی به نام **اصول** است که درستی آن‌ها به عنوان فرض قبول می‌شود. از این اصول می‌توان با استفاده از قواعد منطقی و زنجیره‌ای از **نتیجه‌گیری‌ها**، عبارات درست جدیدی را به دست آورد؛ به عنوان مثال، در هندسه اقلیدسی ما با پنج اصل شروع می‌کنیم:

۱. از هر دو نقطه متمایز، یک و تنها یک خط راست می‌گذرد.

۲. هر پاره‌خط را می‌توان تا بی‌نهایت روی خط راست امتداد داد.

۳. با یک نقطه به عنوان مرکز و یک پاره‌خط به عنوان شعاع، می‌توان یک دایره رسم نمود.

۴. همه‌ی زوایای قائمه با هم برابرند.

۵. اگر یک خط، دو خط دیگر را قطع کند، آن دو خط در طرفی که جمع زوایای داخلی تولید شده توسط خط مورب کمتر از دو قائمه است به هم می‌رسند.

سپس این اصول را با استفاده از قوانین استنتاجی با هم ترکیب می‌کنیم تا نتایجی مانند قضیه فیثاغورس را به دست آوریم. به طور کلی، هر اثبات از سه بخش تشکیل شده‌است: بستری از اصول، زنجیره‌ای از نتیجه‌گیری‌ها با استفاده از قواعد استنتاجی و عبارات نتیجه‌گیری‌شده. در ادامه بیشتر با مفهوم **تعاملی** بودن یک اثبات آشنا می‌شویم. برای انتقال درستی یک اثبات معمولاً نیاز داریم تا به نحوی، اطلاعاتی در مورد آن را با یک فرد دیگر به اشتراک بگذاریم.

[1] Andrew Wiles
[2] Zero-knowledge proofs
[3] Avi Wigderson

اثبات‌های تعاملی

سیستم‌های اثبات هیچ‌آگاهی یک زیرمجموعه از *اثبات‌های تعاملی*[4] هستند. یک سیستم اثبات تعاملی هدفش اثبات کردن درستی یک عبارت منطقی توسط یک اثبات‌کننده[5] به یک تصدیق‌کننده[6] است. این سیستم به صورت یک دستگاه انتزاعی[7] است که عمل محاسبه کردن را به صورت ردوبدل پیام مدل می‌کند. اثبات‌کننده محدودیت محاسباتی‌ای ندارد اما نمی‌توان به او اطمینان کرد. تصدیق‌کننده اما، قابل اعتماد است ولی قدرت محاسباتی محدودی در اختیار دارد. پیام‌ها بین این دو مبادله می‌شوند تا وقتی که تصدیق‌کننده از داشتن جواب درست مسئله اطمینان پیدا کند.

تصدیق‌کننده مانند ریاضی‌دانان شکاک ما است که سعی بر تعیین راستی ادعای یک اثبات‌کننده دارند. در طی فرایند توضیح اثبات، ریاضی‌دانان با هوش محدودی که دارند سعی می‌کنند تا با بالاترین احتمال ممکن از درست بودن یک قضیه مطمئن شوند و در این راه با اثبات‌کننده مکالماتی هم خواهند داشت. تمامی سیستم‌های اثبات تعاملی باید دو ویژگی مشترک را داشته باشند:

۱. **تمامیت (Completeness)**: اگر عبارتی درست باشد و تصدیق‌کننده مراحل بررسی را درست انجام دهد، یک اثبات‌کننده غیرمعتمد می‌تواند درستی آن را به تصدیق‌کننده اثبات کند.

۲. **بی‌نقصی (Soundness)**: اگر عبارتی غلط باشد، یک اثبات‌کننده نمی‌تواند تصدیق‌کننده را متقاعد به درستی آن کند مگر با احتمال بسیار کم.

با اضافه کردن یک شرط سوم به اثبات‌های هیچ‌آگاهی می‌رسیم:

۳. **هیچ‌آگاهی (Zero-knowledge)**: اگر عبارتی درست باشد، تصدیق‌کننده جز درست بودن عبارت هیچ اطلاعات دیگری را به دست نخواهد آورد. تصدیق‌کننده نه اطلاعاتی در مورد نحوه به دست آوردن اثبات و نه اطلاعاتی در مورد محتوای آن خواهد دانست.

در ادامه یک مثال از فرایند اثبات هیچ‌آگاهی خواهیم دید.

[4] Interactive proofs
[5] Prover
[6] Verifier
[7] Abstract machine

توپ‌های رنگی

با ارائه یک مثال می‌توانیم بهتر با کارکرد یک سیستم هیچ‌آگاهی آشنا شویم.

فرض کنید که شما دو توپ قرمز و سبز دارید. این دو توپ جز در رنگ هیچ تفاوت دیگری با هم ندارند. حال فرض کنید فردی کوررنگ می‌خواهد ادعای شما مبنی بر تفاوت رنگی این دو توپ را بررسی کند. شما نیز نمی‌خواهید او هیچ‌گونه اطلاعاتی در مورد رنگ این دو توپ به دست بیاورد. به عبارتی، شما می‌خواهید متفاوت بودن این توپ را اثبات کنید بدون این که به این فرد (تصدیق‌کننده) اعلام کنید که کدام توپ قرمز و کدام توپ سبز است.

این مسئله را می‌توان به این شکل حل کرد: شما دو توپ را به تصدیق‌کننده می‌دهید و او این دو توپ را در پشت خود و خارج از دید شما قرار می‌دهد. سپس یک توپ را در دست راستش به شما نشان می‌دهد و دوباره توپ‌ها را در پشت خود پنهان می‌کند. او سپس توپی را دوباره با دست راست به شما نشان داده و از شما می‌پرسد که: «آیا من دو توپ را عوض کرده‌ام یا خیر؟»

اگر شما (اثبات‌کننده) واقعاً در مورد اطلاع از رنگ توپ‌ها درست گفته باشید، می‌توانید در هر صورت به راحتی پاسخ این سوال را بدهید. برای درک بهتر، سلسله مراتب زیر را در نظر بگیرید:

۱. تصدیق‌کننده دو توپ را پنهان می‌کند. نیازی نیست که فرد کوررنگ قدرت تشخیص دو توپ را داشته باشد و صرفاً می‌تواند موقع دریافت اولیه‌ی توپ‌ها موقعیت آن‌ها را به صورت توپ دست چپ یا توپ دست راست حفظ کند.

۲. تصدیق‌کننده در دست راست خود توپ قرمز را نشان می‌دهد و دوباره توپ‌ها را پنهان می‌کند.

۳. تصدیق‌کننده توپ‌ها را عوض کرده و دوباره که دست راست خود را نشان می‌دهد، توپ سبز در دستش خواهد بود.

۴. حال اگر ادعای اثبات‌کننده درست باشد و دو توپ واقعا فرق کنند، او به راحتی پاسخ مثبت به سوال تصدیق‌کننده خواهد داد؛ اما اگر اثبات‌کننده دروغ گفته باشد و دو توپ هر دو یک رنگ باشند، اثبات‌کننده (با فرض تصادفی بودن فرایندها) تنها با احتمال ۵۰ درصد می‌تواند حدس درستی در مورد تعویض توپ‌ها بزند.

۵. این فرایند تا زمانی که تصدیق‌کننده نیاز داشته باشد ادامه خواهد داشت. با هر تکرار، احتمال اینکه اثبات‌کننده بتواند با حدس زدن، تصدیق‌کننده را گمراه کند کمتر خواهد شد؛ به عنوان مثال،

بعد از ۱۰ بار تکرار این فرآیند احتمال این که یک اثبات‌کننده متقلب بتواند نتیجه تمامی آزمایش‌ها را حدس بزند برابر با $\frac{1}{2^{10}} < 0.1\%$ خواهد بود.

در طی این فرایند، فرد کوررنگ نه اطلاعاتی در مورد رنگ هر توپ به دست می‌آورد و نه می‌تواند متوجه شود که شما از کجا می‌دانید رنگ توپ‌ها چه هستند. او صرفاً می‌تواند با تکرار این آزمایش از درستی ادعای متفاوت بودن توپ‌ها اطمینان پیدا کند.

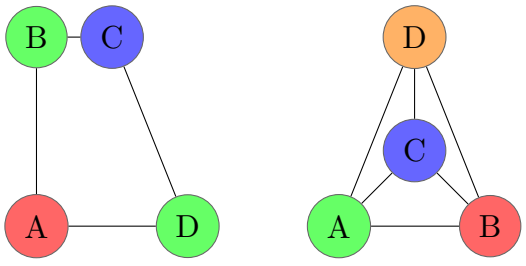
حال که یک مثال از اثبات‌های هیچ‌آگاهی دیدیم، خوب است تا سری هم به کارهای دکتر ویگدرسون بزنیم.

اثبات‌های هیچ‌آگاهی و مسائل NP

مسائل NP[8] مسائلی هستند که با در دست داشتن جوابشان، می‌توان با پیچیدگی چندجمله‌ای (یا به عبارتی، بسیار سریع) درست بودن جواب را چک کرد. اما با داشتن صورت مسئله، به دست آوردن جواب می‌تواند کاری بسیار سنگین و زمان‌بر باشد. یک مثال از چنین مسائلی بازی سودوکو است که به راحتی می‌توان درستی یک جدول را بررسی کرد اما حل کردن یک جدول بسیار سخت است.

اثبات‌های هیچ‌آگاهی از طریق یک مسئله به مسائل NP متصل می‌شوند: مسئله ۳-رنگ‌پذیری گراف.[9]

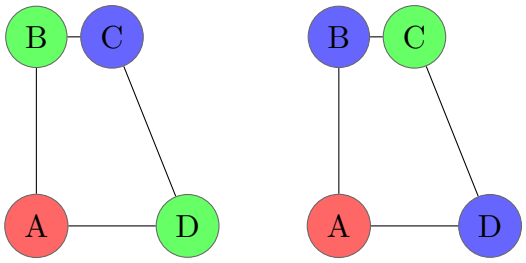
این مسئله به عدد رنگی گراف‌ها مربوط می‌شود. در واقع، مسئله ۳-رنگ‌پذیری می‌پرسد که آیا می‌شود رأس‌های یک گراف را تنها با ۳ رنگ یا کمتر به گونه‌ای رنگ کرد که هیچ دو رأس مجاور هم‌رنگ نباشند؟ در شکل ۱ می‌توانید مثال‌هایی از گراف‌های مختلف ببینید. مسئله ۳-



شکل ۱: گراف سمت چپ را می‌توان با ۳ رنگ، رنگ کرد؛ اما گراف سمت راست به حداقل ۴ رنگ نیاز دارد، پس ۳-رنگ‌پذیر نیست.

رنگ‌پذیری از لحاظ پیچیدگی محاسباتی در رسته NP-Complete قرار دارد. به این معنا که می‌توان تمامی مسائل NP دیگر را با این مسئله شبیه‌سازی کرد. حال، اگر برای این مسئله یک فرایند هیچ‌آگاه

[8] **N**ondeterministic **P**olynomial time
[9] Graph 3-Coloring problem



شکل ۲: با تعویض رنگ سبز و آبی، به یک رنگ‌آمیزی جدید رسیدیم که ساختارش تغییری نکرده و صرفاً از لحاظ ظاهری تغییر کرده‌است.

پیدا کنیم، یعنی تمامی مسائل NP دارای یک اثبات هیچ‌آگاه هستند. این نکته، در قلب کار دکتر ویگدرسون قرار دارد [۲].

فرض کنید یک گراف دارید و می‌خواهید به یک ریاضی‌دان شکاک ثابت کنید که این گراف ۳-رنگ‌پذیر است. آیا امکان دارد تا بدون نشان دادن گراف رنگ شده، ریاضی‌دان تصدیق‌کننده را متقاعد کرد؟ پاسخ مثبت است. قبل از ارائه روش، دقت کنید که اگر گرافی را با ۳ رنگ بزنیم، در واقع ۶ رنگ‌آمیزی مختلف داریم؛ با تعویض رنگ‌ها می‌توانیم به یک رنگ‌آمیزی جدید برسیم. به عنوان مثال شکل ۲ را در نظر بگیرید. حال آماده‌ایم تا یک فرایند هیچ‌آگاه را معرفی کنیم.

۱. اثبات‌کننده ادعا می‌کند که گراف G یک گراف ۳-رنگ‌پذیر است. او رنگ هر رأس را بر روی کاغذی نوشته، در یک پاکت امن قرار داده و بر روی رأس متناظرش قرار می‌دهد.

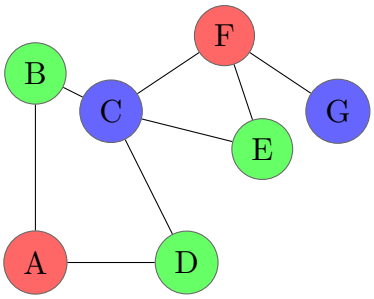
۲. تصدیق‌کننده به طور تصادفی از اثبات‌کننده می‌خواهد که رنگ دو رأس مجاور را نمایان کند. اگر این دو رأس رنگهایشان متفاوت بود که اثبات‌کننده آزمایش را با موفقیت پشت سر می‌گذارد. در غیر این صورت شکست می‌خورد.

اما تصدیق‌کننده می‌تواند با تکرار این فرایند رنگ تمامی رأس‌ها را استخراج کند؛ پس این فرآیند به خودی خود هیچ‌آگاه نیست و باید یک مرحله دیگر را هم اضافه کرد:

۳. پس از هر آزمون، اثبات‌کننده به طور تصادفی یکی از ۶ جایگشت سه رنگ گراف را انتخاب کرده و اعمال می‌کند.

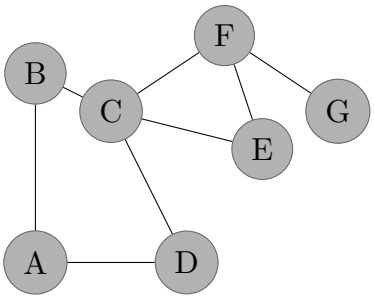
در این صورت، در هر آزمون تصدیق‌کننده صرفاً دو رنگ متفاوت بی‌ربط به بقیه آزمایش‌ها خواهد دید و نمی‌تواند از کنار هم قرار دادن داده‌های آزمایشات به رنگ‌آمیزی گراف پی‌ببرد. حال، یک مثال مرحله به مرحله از این اثبات را با هم بررسی می‌کنیم. فرض کنید که شما می‌خواهید ۳-رنگ‌پذیری گراف زیر را به یک تصدیق‌کننده اثبات کنید:

در مرحله اول، رنگ‌ها را می‌پوشانیم. این پوشاندن به نحوی است که بعد از پوشانده شدن، نه ما می‌توانیم به رنگ‌ها دسترسی پیدا کنیم و



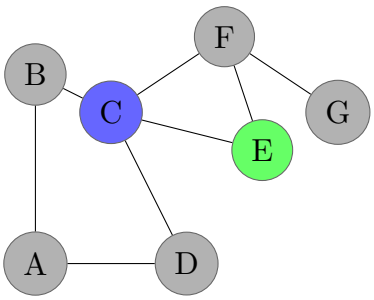
شکل ۳: این گراف ۳-رنگ‌پذیر است و می‌خواهیم بدون فاش کردن رنگ رأس‌ها، این ویژگی این گراف را به یک تصدیق‌کننده اثبات کنیم.

آن‌ها را تغییر دهیم، و نه تصدیق‌کننده می‌تواند بدون اجازه ما رنگ‌ها را ببیند. می‌توانید فرض کنید که اگر گراف را روی یک کاغذ رسم کرده‌ایم، رنگ هر رأس را داخل یک پاکت‌نامه نوشته و بر روی آن رأس قرار می‌دهیم. اگر ما به پاکت‌ها دست بزنیم و آن‌ها را بدون اجازه باز کنیم تصدیق‌کننده متوجه می‌شود و برعکس. حال، تصدیق‌کننده گراف زیر را مشاهده می‌کند:



شکل ۴: رنگ‌ها از تصدیق‌کننده پنهان هستند.

سپس تصدیق‌کننده به طوری تصادفی دو رأس مجاور را انتخاب کرده و از ما می‌خواهد تا رنگ آن‌ها را نمایان کنیم. فرض کنید که چالش او، نمایان کردن رنگ دو رأس C, E است: ما در آزمایش اول موفق

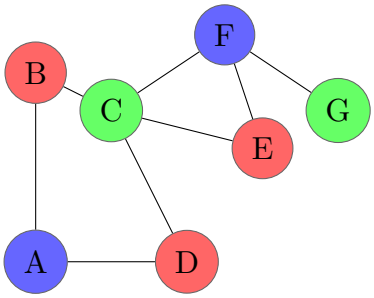


شکل ۵: اثبات‌کننده در این آزمایش موفق می‌شود چون رنگ دو رأس مجاور مطابق ادعایش متفاوت بود.

شدیم چون رنگ دو رأس مطابق ادعای ما درست بود. حال، می‌خواهیم برای آزمایش دوم آماده شویم. در ابتدا به طور تصادفی یک جایگشت از سه رنگ قرمز، آبی و سبز را انتخاب می‌کنیم.

$$\text{Green} \mapsto \text{Red}$$
$$\text{Blue} \mapsto \text{Green}$$
$$\text{Red} \mapsto \text{Blue}$$

بعد از اعمال این جایگشت به گراف اولیه ما (شکل ۳) خواهیم داشت: برای درک بهتر این پروسه دقت کنید که در شکل ۳ سه رأس

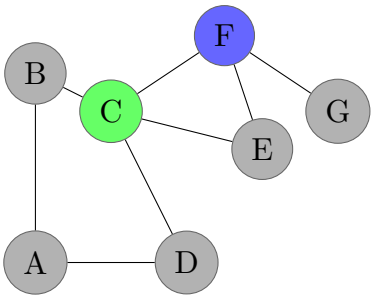


شکل ۶: ماهیت این رنگ‌آمیزی با شکل ۳ هیچ فرقی ندارد و صرفاً رنگ‌ها با هم تعویض شده‌اند.

B, D, E هم‌رنگ و تنها رأس‌های سبز هستند، در شکل ۶ هم این سه رأس هم‌رنگ و تنها رأس‌های قرمز ما می‌باشند. منظور ما از تغییر نکردن ماهیت رنگ‌آمیزی گراف، ثابت ماندن مجموعه رئوس هم‌رنگ است. بعد از اعمال این جایگشت دوباره همانند شکل ۴ رنگ‌ها را پنهان می‌کنیم.

حال اگر تصدیق‌کننده دوباره رنگ C, E را از ما بخواهد، دو رنگ سبز و قرمز را خواهد دید که ربطی به آزمایش قبلی ندارد. همچنین اگر از ما رنگ C, F را بخواهد نمی‌تواند با این اطلاعات جدید نتیجه‌ای در مورد رنگ سه رأس C, F, E در رنگ‌آمیزی به دست آورده شده توسط ما بگیرد و صرفاً خواهد دید که رنگ رئوس مجاور انتخاب شده در هر مرحله (همانند ادعای ما) متفاوت است.

به این ترتیب و با استفاده از جایگشت‌ها می‌توانیم بدون افشای رنگ‌آمیزی، درست بودن آن را به یک تصدیق‌کننده اثبات کنیم. نکته مهم این است که برای تقلب در این آزمون، ما یا باید به طور تصادفی به رأس‌ها رنگ می‌دادیم که احتمال موفقیت آن در آزمایش‌های متوالی و متعدد کم است؛ یا باید سعی می‌کردیم نحوه انتخاب رئوس توسط تصدیق‌کننده را متوجه شویم که اگر او هوشمندانه عمل کرده و کاملاً تصادفی این کار را انجام دهد برای ما امکان‌پذیر نخواهد بود. تصدیق‌کننده هم اگر می‌خواست رنگ‌آمیزی ما را متوجه شود یا باید



شکل ۷: آزمایش شماره دوم که یک موفقیت برای اثبات‌کننده است.

محاسبات سنگینی انجام می‌داد (که معادل وقت گذاشتن برای حل خود مسئله است و برای تصدیق‌کننده جذاب نیست) یا باید سعی می‌کرد تا محتوای پاکت‌ها را بخواند که آن وقت ما متوجه تقلب او می‌شدیم.

حال که توانستیم یک روش هیچ‌آگاه برای نشان دادن ۳-رنگ‌پذیری پیدا کنیم، در واقع یعنی هر مسئله NP یک اثبات هیچ‌آگاه دارد؛ می‌توانیم با تبدیل مسئله به یک مسئله رنگ‌آمیزی گراف، آن را به صورت هیچ‌آگاه به یک تصدیق‌کننده اثبات کنیم.

این قضیه شاید در نگاه اول جذابیتی نداشته باشد، اما در حوزه‌هایی مانند رمزنگاری و رمزارزها بسیار کاربردی است. هر جایی که نیاز به امنیت باشد، از اثبات‌های هیچ‌آگاه می‌توان استفاده کرد؛ یک مثال از این مسئله‌ها را بررسی می‌کنیم:

لگاریتم گسسته

مسئله پیدا کردن لگاریتم گسسته به شرح زیر است: با داشتن یک عدد اول p، یک مولد g و یک متغیر y، یک x پیدا کنید به صورتی که

$$g^x \bmod p = y.$$

این مسئله در رمزنگاری کاربرد بسیاری دارد. می‌توان با استفاده از مقادیر x و y یک سیستم احراز هویت ایجاد کرد که در آن مقدار y عمومی است و شما هر جا که نیاز به اثبات هویت خود دارید، با ارائه مقدار x هویت خود را اثبات می‌کنید. چون پیدا کردن یک مقدار برای اعداد اول بزرگ بسیار سخت است، به راحتی نمی‌توان مقدار x را پیدا و جعل هویت کرد. فرض کنید شما یک جواب برای این مسئله دارید. بسیار راحت می‌توان با به اشتراک گذاشتن مقدار x به هر کسی ثابت کرد که شما مقدار درست را می‌دانید؛ اما این نگرش دارای مشکلی است. اگر کسی بتواند موقع برقراری ارتباط مقدار x را رصد کند، یا بتواند از تصدیق‌کننده آن را بگیرد، دیگر استفاده از x بی‌معنا خواهد شد.

اینجا است که اثبات‌های هیچ‌آگاه مفید واقع می‌شوند. یک فرایند هیچ‌آگاه برای این مسئله به شرح زیر می‌باشد [۱]:

۱. در شروع هر آزمون، اثبات‌کننده یک مقدار تصادفی $r \in \{0, \ldots, p-1\}$ انتخاب کرده و مقدار $C = g^r$ را اعلام می‌کند.

۲. بعد از دریافت این مقدار، تصدیق‌کننده یک $b \in \{0, 1\}$ را به صورت تصادفی انتخاب کرده و اعلام می‌کند. حال، اثبات‌کننده باید مقدار $C_b = (bx + r) \bmod (p-1)$ را اعلام کند.

۳. حال تصدیق‌کننده بررسی می‌کند که آیا این مقدار اعلام شده توسط اثبات‌کننده در تساوی

$$g^{C_b} = C \cdot y^b \qquad (۱)$$

صدق می‌کند یا خیر.

در فرایند بالا، اگر مقدار b به طور کاملاً تصادفی انتخاب شود، یک اثبات‌کننده متقلب فقط می‌تواند در ۵۰ درصد چالش‌ها موفق به فریب تصدیق‌کننده شود. همانند قبل، می‌توان با تکرار آزمایش احتمال موفقیت او در تقلب کردن را به سرعت کاهش داد. همچنین، هیچ اطلاعاتی راجع به مقدار x به تصدیق‌کننده داده نمی‌شود؛ یا مقدار r اعلام می‌شود و یا $(x+r) \bmod (p-1)$ که هر دو با انتخاب درست، تصادفی خواهند بود و اطلاعاتی را به تصدیق‌کننده منتقل نخواهند کرد؛ به عنوان مثال، با همچین پروسه‌ای یک بانک می‌تواند بدون نگه‌داشتن رمز کارت (x)، حتی به صورت رمزی با استفاده از تکنیکی مانند هشینگ[10]، صرفاً با داشتن یک مقدار عمومی (y) صحت تراکنش‌ها را (با مقدار کمی خطا) تأیید کند.

در این مقاله سعی شد تا اثبات‌های هیچ‌آگاه معرفی شوند. این حوزه جدا از کاربردهای روزافزونی که در عصر کامپیوترمحور امروز دارد، از لحاظ ریاضیاتی هم بسیار جالب است. این دسته از الگوریتم‌ها تمامی تصورات ما از مفاهیمی مانند «اثبات» را دچار تغییر می‌کنند. همچنین، این حوزه محل اتصال بسیاری از قسمت‌های ریاضیات است؛ از تکنیک‌های آماری برای تعیین معیار اعتماد به آزمون گرفته تا ریاضیات مربوط به رنگ‌آمیزی گراف‌ها و نظریه اعداد و علوم کامپیوتر.

مراجع

[1] Chaum, David, Evertse, Jan-Hendrik, and van de Graaf, Jeroen. An improved protocol

[10] Hashing

for demonstrating possession of discrete logarithms and some generalizations. pages 127–141, 1988.

[2] Goldreich, Oded, Micali, Silvio, and Wigderson, Avi. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991.

مجموعه‌ی کانتور

Cantor set

هستی برقراریان

جرج کانتور [1] (۱۹۱۸- ۱۸۴۵) که بیش‌تر به دلیل فعالیتش در زمینه‌ی نظریه‌ی مجموعه‌ها [2] شهرت دارد، برای اولین‌بار مجموعه‌ای را تعریف کرد که به «مجموعه‌ی سه‌تایی کانتور» معروف است. این تعریف از این نقطه‌نظر اهمیت دارد که، مجموعه‌های کامل چنین نیستند که باید همه‌جا متراکم باشند. (می‌توانند در هیچ‌جا متراکم باشند.)

در این‌جا سعی داریم که مجموعه‌ی سه‌گانه‌ی کانتور را مختصری از دیدگاه فراکتال‌ها و بُعد هاسدورف مطالعه و درباره‌ی ویژگی‌های این مجموعه‌ی زیبا صحبت کنیم.

مجموعه‌های کانتور از نظر هندسی با تصور حذف پیوسته‌ی یک قسمت تعیین‌شده از یک شکل برابرند؛ به طوری که در هر مرحله از حذف، هر کدام از قسمت‌های باقی‌مانده از شکل درصد یکسانی از حذف را در خود دارند. اگر فرآیند حذف، بی‌نهایت بار تکرار شود، آن‌گاه، قسمت‌های کوچک باقی‌مانده از شکل، یک مجموعه‌ی کانتور را تشکیل می‌دهند.

بعد چنین مجموعه‌ای یک عدد صحیح نیست؛ بلکه یک بعد فراکتالی دارد که با استفاده از تعریف فراکتال به دست می‌آید.

می‌توان درک کرد که نقطه‌ها از بعد 0 هستند و خطوط دارای بعد 1 می‌باشند ولی بعد یک مجموعه‌ی فراکتال می‌تواند هر مقداری بین اعداد صحیح را، بسته به نوع تشکیل مجموعه، اتخاذ کند.

اصطلاح مجموعه‌ی کانتور، اغلب برای اشاره به مجموعه‌ی سه‌گانه‌ی کانتور به کار می‌رود که به صورت زیر ساخته می‌شود:

فرض کنید I بازه‌ی $[0,1]$ باشد. I را به سه زیربازه تقسیم کنید. زیربازه‌ی باز را که در میانه قرار دارد حذف کنید (زیربازه‌ی $(\frac{1}{3},\frac{2}{3})$). نام مجموعه‌ی باقی‌مانده را A_1 بگذارید. پس داریم:

$$A_1 = [0,\tfrac{1}{3}] \cup [\tfrac{2}{3},1].$$

فرآیند حذف کردن را به همین شکل برای دو زیربازه‌ی باقی‌مانده در A_1 انجام دهید. مجموعه‌ای که باقی مانده‌است را A_2 بنامید. پس داریم:

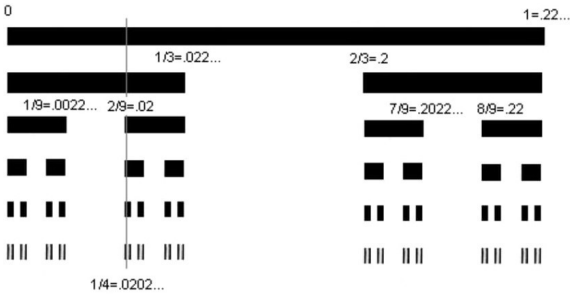
$$A_2 = [0,\tfrac{1}{9}] \cup [\tfrac{2}{9},\tfrac{1}{3}] \cup [\tfrac{2}{3},\tfrac{7}{9}] \cup [\tfrac{8}{9},1].$$

به همین ترتیب کار را برای هر $k \in \mathbb{N}$ انجام دهید. یعنی زیربازه‌ی میانی را حذف کنید و مجموعه‌ی باقی‌مانده را A_{k+1} بنامید. برای هر $k \in \mathbb{N}$، A_k اجتماع 2^k بازه‌ی بسته است که طول هر کدام 3^{-k} می‌باشد.

تعریف:

مجموعه‌ی سه‌گانه‌ی کانتور، که آن را با C_3 نمایش می‌دهیم، مجموعه‌ی حدی این فرآیند است:

$$C_3 = \bigcap_{k=1}^{\infty} A_k$$



شکل ۱: پنج مرحله از فرآیند حذف کردن در ساخت مجموعه‌ی کانتور

هر عنصر از C_3 می‌تواند در یک بسط سه‌سه‌ای از فقط 0 ها و 2 ها نوشته شود. در هر مرحله از فرآیند حذف کردن، هر عدد با یک بسط سه‌سه‌ای که شامل رقم 1 در جایگاه مشخصی باشد، حذف می‌شود.

برای مثال، در پله‌ی اول فرآیند حذف، هر عددی که باقی می‌ماند، باید رقم $c_1 = 0$ یا $c_1 = 2$ را در $x = 0.c_1c_2c_3...$ داشته‌باشد؛ اگر $x \in [0,\frac{1}{3}]$، آن‌گاه $c_1 = 0$ و اگر $x \in [\frac{2}{3},1]$، آن‌گاه $c_1 = 2$.

با تکرار همین استدلال برای هر مرحله از فرآیند حذف، می‌توان نشان داد که پس از n مرحله، اگر عدد x در مجموعه باقی بماند، آن‌گاه $c_1 = 0$ یا $c_1 = 2$.

مجموعه‌ی سه‌گانه‌ی کانتور، به خاطر پارادوکس آشکارش، در ریاضیات بسیار جالب توجه است؛ اما به هر حال، ساختگی است. توجه کنید که تعداد نامتناهی بازه که مجموع طول آن‌ها 1 است، از یک بازه به طول 1 حذف می‌شوند. پس مجموعه نمی‌تواند شامل هیچ بازه‌ای به طول غیرصفر باشد. در حالی که مجموعه شامل نامتناهی نقطه است و در حقیقت کاردینالی برابر با کاردینال بازه‌ی $[0,1]$ دارد.

[1] Georg Cantor
[2] set theory

پس مجموعه‌ی کانتور به تعداد نقاط مجموعه‌ای که از آن حذف شده‌است، نقطه دارد؛ اما دارای هیچ بازه‌ای نیست و در هیچ‌جا متراکم است. این بدان معناست، که مجموعه هیچ نقطه‌ای ندارد که به طور کامل توسط نقاط دیگر این مجموعه احاطه شده باشد. می‌دانیم که مجموعه دارای نامتناهی نقطه است؛ زیرا نقاط پایانی هر بازه‌ی بسته، پیوسته در مجموعه باقی می‌مانند. اما باید توجه کرد که مجموعه‌ی کانتور تعداد بیش‌تری از تعداد نقاط پایان بازه‌های بسته را شامل می‌شود.

برای مثال، $\frac{1}{4} \in C_3$؛ اما $\frac{1}{4}$ نقطه‌ی پایانی هیچ بازه‌ای در هیچ یک از مجموعه‌های A_k نیست.

می‌توانیم $\frac{1}{4}$ را در بسط سه‌سه‌ای به صورت $0.020202...$ بنویسیم. در مرحله‌ی k ام از فرآیند حذف کردن، هر نقطه‌ی پایانی جدید یا دارای رقم 2 در جایگاه $3^{-(k-1)}$ در بسط سه‌سه‌ای است، که به صورت نامتناهی ادامه می‌یابد، یا در جایگاه 3^{-k} ام در بسط سه‌سه‌ای خاتمه می‌یابد.

از آن‌جا که $\frac{1}{4} = 0.020202...$ ، پس الگوی نقطه‌های پایانی را دنبال نمی‌کند. بنابراین $\frac{1}{4}$ در مجموعه‌ی کانتور، یک نقطه‌ی پایانی نیست و طبیعتاً بی‌نهایت نقطه مانند آن در این مجموعه وجود دارد.

ویژگی‌های مجموعه‌ی کانتور:

ویژگی‌های بدیهی مجموعه‌ی کانتور در کنار هم، که به راحتی نیز اثبات می‌شوند، نشان‌دهنده‌ی طبیعت خاص مجموعه‌ی کانتور می‌باشند.

۱. C_3 دارای تعداد عناصر نامتناهی است.

فرض کنید :

$$C_3 = \{x \in [0,1) | x = 0.c_1c_2c_3... \wedge (c_n = 0 \vee c_n = 2)\}$$

شمارا است.

پس تابعی مثل $f : \mathbb{N} \to C_3$ وجود دارد که پوشا و یک‌به‌یک است.

در نظر بگیرید برای هر $n \in \mathbb{N}$ داریم: $x_n = f(n)$.

پس داریم:

$$C_3 = \{x_1, x_2, x_3, ..., x_n, ...\}$$

که در آن:

$$x_1 = 0.c_{11}c_{12}c_{13}...$$
$$x_2 = 0.c_{21}c_{22}c_{23}...$$
$$x_3 = 0.c_{31}c_{32}c_{33}...$$

که x_{nm} برای هر n, m، 0 یا 2 است.

تعریف کنید:

$$c = 0.c_1c_2c_3...$$

که:

$$c_1 = \begin{cases} 2 & c_{11} = 0 \\ 0 & c_{11} = 2 \end{cases}$$

$$c_2 = \begin{cases} 2 & c_{22} = 0 \\ 0 & c_{22} = 2 \end{cases}$$

. . .

$$c_n = \begin{cases} 2 & c_{nn} = 0 \\ 0 & c_{nn} = 2 \end{cases}$$

به وضوح $c \in C_3$؛ ولی از آن‌جا که c و x_n در جایگاه 3^{-n} ام متفاوتند، به ازای هر n ، $c \neq x_n$؛ و این تناقض است؛ بنابراین C_3 ناشمارا است.

۲. C_3 شامل هیچ بازه‌ای نیست.

نشان خواهیم داد که طول متمم مجموعه‌ی کانتور، برابر با 1 است و در نتیجه شامل هیچ بازه‌ای نمی‌باشد.

در مرحله‌ی k ام، 2^{k-1} بازه از مجموعه‌ی بازه‌ها را حذف می‌کنیم. توجه داریم که طول هر بازه $\frac{1}{3^k}$ است.

پس از تعداد نامتناهی از حذف‌ها، طول متمم داخل $[0,1]$ برابر است با:

$$\sum_{k=1}^{\infty} 2^{k-1}(\tfrac{1}{3^k}) = \tfrac{1}{3}\sum_{k=1}^{\infty}(\tfrac{2}{3})^{k-1} = \tfrac{1}{3}\sum_{k=0}^{\infty}(\tfrac{2}{3})^{k} = 1$$

بنابراین یک بازه با طول 1 را از بازه‌ی یکه‌ی $[0,1]$ حذف می‌کنیم.

در نتیجه مجموعه‌ی کانتور باید طولی برابر با 0 داشته باشد. این بدان معنی است که هیچ بازه‌ای ندارد.

۳. C_3 فشرده است.

با استفاده از تئوری هاینه-برل [3] ، که بیان می‌کند که یک زیرمجموعه از $\mathbb{R}$ فشرده است اگر و تنها اگر بسته و کراندار باشد، می‌توان نشان داد که C_3 نیز فشرده است.

C_3 اشتراک یک کلکسیون از مجموعه‌های بسته است. پس خود، مجموعه‌ای بسته است. از آن‌جا که هر A_k در بازه‌ی $[0,1]$ است، C_3 به عنوان اشتراک مجموعه‌های کراندار است. بنابراین از آن‌جا که C_3 کراندار و بسته است، پس C_3 فشرده است.

تا به حال بیان کردیم که مجموعه‌ی کانتور، زیرمجموعه‌ای از بازه‌ی $[0,1]$ است که تعداد نامتناهی عنصر دارد ولی شامل هیچ بازه‌ای

[3] Heine-Borel Theorem

نیست. کاردینال آن با کاردینال اعداد حقیقی برابر است ولی طول آن 0 است.

۴. C_3 در هیچ‌جا متراکم است.

یک مجموعه در هیچ‌جا متراکم در نظر گرفته می‌شود، اگر درون بستار مجموعه تهی باشد. این بدان معنا است که وقتی نقاط حدی مجموعه را اضافه می‌کنید، هم‌چنان هیچ بازه‌ای وجود نداشته باشد. بستار یک مجموعه اجتماع نقاط مجموعه با نقاط حدی آن است. پس، از آن‌جا که هر نقطه در مجموعه‌ی C_3 یک نقطه‌ی حدی است، C_3 یک مجموعه‌ی در هیچ‌جا متراکم در نظر گرفته می‌شود، اگر درون بستار مجموعه تهی باشد. این بدان معنا است، که وقتی نقاط حدی مجموعه را اضافه می‌کنید، هم‌چنان هیچ بازه‌ای وجود نداشته باشد. پس از آن‌جا که هر نقطه در مجموعه، یک نقطه‌ی حدی است، بستار مجموعه، خود مجموعه می‌باشد. در تعداد نامتناهی مرحله از حذف کردن، اگر بازه‌ای از اعداد وجود داشته باشد، قسمت میانی آن بازه حذف می‌شود و فرآیند حذف به طور نامتناهی در مقیاس کوچک‌تری ادامه خواهد یافت تا در نهایت همه‌ی نقاط بین دو نقطه‌ی مورد نظر حذف شوند.

بنابراین مجموعه‌ی کانتور در هیچ‌جا متراکم است.

بعد مجموعه‌ی کانتور:

به نظر می‌رسد که مجموعه‌ی کانتور صرفاً مجموعه‌ای از نقاط مجرد است؛ پس همان‌طور که هر مجموعه‌ی تصادفی از نقاط، بعد 0 دارد، این مجموعه هم باید بعد 0 داشته باشد.

یادآور می‌شویم که از نظر توپولوژیکی یک نقطه بعد 0 دارد، یک خط بعد 1 دارد، یک صفحه بعد 2 دارد، یک مکعب بعد 3 دارد و فضای اقلیدسی $\mathbb{R}^n$ بعد n دارد.

بعد توپولوژیکی یک فضا برابر است با تعداد پارامترهای حقیقی لازم برای توصیف و توضیح نقاط مختلف یک فضا. با توجه به این تعریف، بعد توپولوژیکی مجموعه‌ی کانتور 0 می‌باشد. اما با استفاده از تعریفی متفاوت، مثل بعد هاسدورف [۴] ، می‌توان بعد فراکتالی مجموعه کانتور را به دست آورد.

بعد هاسدورف:

بعد هر زیرمجموعه‌ی E از مجموعه‌ی $\mathbb{R}^n$ می‌تواند با استفاده از بعد هاسدورف اندازه‌گیری شود. به عبارت ساده‌تر، بعد هاسدورف شامل پوشاندن مجموعه‌ی E با دلتا-پوشش‌ها است. اگر مجموع قطر همه‌ی دلتا-پوشش‌های E را به توان s برسانیم، اینفیمم این مجموع با $H^s_\delta(E)$ نمایش داده می‌شود. اگر سوپریمم همه‌ی مقادیر ممکن برای $H^s_\delta(E)$ را در نظر بگیریم، اندازه‌ی s-بعدی هاسدورف فضای E را به دست می‌آوریم. پس بعد هاسدورف E با گرفتن کوچک‌ترین مقدار عدد حقیقی s برای مجموعه‌ای که اندازه‌ی s-بعدی هاسدورف 0 است و کوچک‌ترین مقدار s برای مجموعه‌ای است که هنوز می‌توانیم آن را پوشش دهیم، مشخص می‌شود.

بعد هاسدورف، مفهوم بعد یک فضای برداری را به گونه‌ای تعمیم می‌دهد که بعد هاسدورف نقاط 0 باشد، خط ها بعد 1 داشته‌باشند و... . اما باید توجه داشت که بعد هاسدورف یک مجموعه لزوما یک مقدار صحیح نیست.

فراکتال‌ها به عنوان مجموعه‌هایی تعریف می‌شوند که بعد هاسدورف آن‌ها از بعد توپولوژیکی‌شان بیش‌تر است؛ که البته بعد هاسدورف فراکتال‌ها به طور مشخص غیرصحیح است.

یک راه میان‌بر برای محاسبه‌ی بعد هاسدورف مجموعه‌هایی که در طبیعت خودهمانند هستند وجود دارد. خودهمانندی یک ویژگی است که بسیاری از فراکتال‌ها آن را دارند و زمانی آشکار می‌شود که یک مجموعه کاملا یا تقریبا شبیه قسمتی از خودش باشد. به بیانی؛ با بزرگ‌نمایی قسمت‌های کوچک و کوچک‌تر یک مجموعه‌ی خودهمانند یک کپی از مجموعه را خواهیم‌داشت.

از نظر ریاضی، برای خودهمانند بودن، نگاشت‌هایی که مجموعه‌ی A را تا بی‌نهایت می‌سازند، باید یک مجموعه‌ی متناهی از شبیه‌سازی‌ها باشند که نگاشت‌های Q_i هستند که با حفظ هندسه‌ی مجموعه، آن را تولید می‌کنند. همان‌طور که مجموعه‌ی A با توجه به نگاشت‌های Q_i ثابت است و باید عدد حقیقی مثبتی مثل s وجود داشته‌باشد که H^s برای A مثبت باشد اما برای اشتراک دو نگاشت مختلف A مانند Q_1, Q_2 صفر باشد.

پس از آن‌جا که نگاشت‌های مجموعه‌ی کانتور، تمام تغییرات روی x، هندسه‌ی مجموعه را حفظ می‌کند و یک اندازه‌ی s-بعدی هاسدورف مثبت از مجموعه وجود دارد (و نه یک اندازه‌ی s-بعدی مثبت از اشتراک دو نگاشت مختلف)، مجموعه‌ی کانتور یک خودهمانند است.

تعمیم دادن:

با این‌که در ساخت مجموعه‌ی کانتور از قانون سه‌تایی استفاده کردیم، به راحتی می‌توانیم این ایده‌ی یک‌بعدی را به هر طولی به غیر از $\frac{1}{3}$ تعمیم بدهیم؛ البته قطعاً به استثنای در نظر گرفتن موارد.

تعمیم یک‌بعدی مجموعه‌ی کانتور:

برای ساخت، ابتدا یک طول مثل $\delta \in (0, \frac{1}{2})$ را در نظر می‌گیریم.

از بازه‌ی $[0,1]$، بازه‌ی باز وسطی I_0 به طول $(1-2\delta)$ را حذف کنیم؛ به طوری که:

$$[0,1] - I_0 = [0,\delta] \cup [1-\delta, 1]$$

که $I_0 = (\delta, 1-\delta)$.

[4] Hausdorff Dimension

سپس، از $[0,\delta], [1-\delta,1]$، بازه‌های باز وسطی را که I_{11}, I_{12} هستند و طول هر کدام $(\delta - 2\delta^2)$ می‌باشد، حذف می‌کنیم. اجتماع آن‌ها را I_1 بنامید:

$$I_1 = (\delta^2, \delta - \delta^2) \cup (1-(\delta-\delta^2), 1-\delta^2)$$

که طول آن برابر با $(2\delta - 4\delta^2)$ است.

آن‌چه که باقی می‌ماند، $[0,1] - (I_0 \cup I_1)$ است.

با پیش رفتن این روش، دنباله‌هایی از مجموعه‌های باز I_n را می‌سازیم که هر کدام، اجتماع متناهی تا از بازه‌های مجزا از هم‌اند.

تعریف:

مجموعه کانتور تعمیم‌یافته‌ی C_δ تعریف می‌شود:

$$C_\delta = [0,1] - \bigcup_{i=0}^{\infty} I_i$$

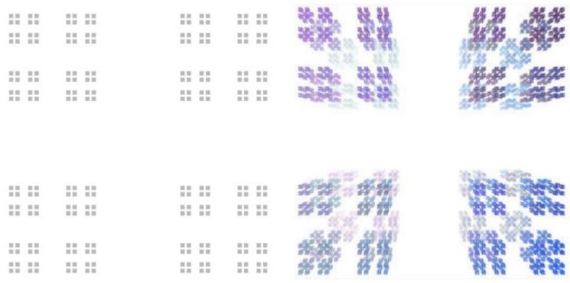
می‌توان دید که عبارت $\delta = \frac{1}{3}$، مجموعه‌ی کانتور سه‌گانه‌ی استاندارد را به دست می‌دهد.

این صورت از مجموعه‌ی کانتور برای هر $\delta \in (0, \frac{1}{2})$، ویژگی‌های مشابهی با مجموعه‌ی کانتور سه‌تایی را آشکار می‌کند که می‌دانیم، فشرده، ناتهی، کامل و... است.

تعمیم در بعد بالاتر:

می‌توان ایده‌ی یک‌بعدی را به دوبعدی هم گسترش داد:

با شروع از یک مربع واحد $[0,1] \times [0,1]$ به جای بازه‌ی یکه و به طور مشابه می‌توان با شروع از مکعب یکه، به سه‌بعدی هم گسترش داد. نمونه‌های دوبعدی و سه‌بعدی در زیر نشان داده شده‌اند:



شکل ۲: نمونه‌های دوبعدی و سه‌بعدی مجموعه‌ی کانتور

مراجع

[1] Nelson, Dylan R. The cantor set-a brief introduction. *University of California & Berkeley, Berkeley, CA*, 94704, 2019.

[2] Shaver, Christopher. An exploration of the cantor set. *Rose-Hulman Undergraduate Mathematics Journal*, 11(1):1, 2010.

مقدمه‌ای بر نظریه کدگذاری و کاربرد آن در ارسال تصاویر از مریخ

آرمین احمدخان‌بیگی

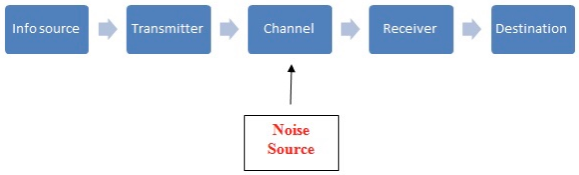
دهه‌هاست که کاوشگرهای فضایی داده‌ها را از دورترین سیاره‌ها به زمین ارسال می‌کنند. این در حالی است که توان فرستنده‌های رادیویی این دستگاه‌ها، تنها چند وات است. حال سوالی که پیش می‌آید این است که چطور این حجم از داده، بدون گرفتار شدن در بحبوحه‌ی نویزها، از میلیون‌ها مایل دورتر به ما می‌رسد؟

نظریه کدگذاری شاخه‌ای از مخابرات است که مربوط به انتقال داده‌ها از طریق کانال‌های نویزدار و بازیابی پیام است. این نظریه مربوط به آسان‌تر کردن خواندن پیام است و ارتباطی با نظریه رمزگذاری که مربوط به سخت‌تر کردن خواندن پیام است ندارد.

این شاخه در مهندسی برق و مخابرات، قدمتی طولانی دارد ولی در سال‌های اخیر به دلیل توجه خاص به تحلیل مباحث تئوری در ساخت و کدگشایی کدها، علاقه ریاضی‌دانان به این زمینه بیشتر شده است و با نگاهی اجمالی می‌توان متوجه شد که بسیاری از شاخه‌های ریاضیات محض و کاربردی مانند جبر، هندسه، گراف و ترکیبیات، نظریه احتمالات و ... هرکدام به گونه‌ای در این زمینه نقش موثر داشته‌اند. در واقع، نظریه کدگذاری مثال خوبی از کاربرد ریاضیات محض در حل مسائل عملی است.

فرض کنیم پیام ما در قالب اعداد یا بیت‌های باینری است؛ یعنی رشته‌هایی از 0 و 1. ما باید این بیت‌ها را در یک کانال منتقل کنیم (مثلا یک خط تلفن) که به‌صورت تصادفی، خطاهایی در آن رخ می‌دهد. اما این خطاها یک نرخ قابل حدس دارند. برای جبران این خطاها، باید بیت‌های بیشتری را در پیام اصلی ارسال کنیم.

یک کد دوتایی، گردایه‌ای از دنباله‌های دوتایی n رقمی به نام کدواژه



شکل ۱: کانال مخابراتی و تاثیر نویز بر کدواژه‌ها

است. به عنوان مثال، ۱۱۰۱۰۰ یک کدواژه به طول ۶ است. اگر کدواژه‌ها ارسال شوند، امکان بروز خطا در اثر پارازیت وجود دارد و بنابراین کلمات دریافتی ممکن است متفاوت از کدکلمات ارسال شده باشند. ایده اصلی این است که با اضافه کردن چند بیت، کدواژه‌ها را به قدر کافی متفاوت از یکدیگر انتخاب کنیم به قسمی که اگر در هنگام ارسال چندخطا صورت بگیرد، واژه دریافتی هنوز به کدواژه ارسال شده نسبت به کدواژه‌های دیگر نزدیک‌تر باشد. این ایده(اصل افزونگی) در واقعیت زندگی انسان نیز حضور دارد. کلمات زبان ما، قسمت کوچکی از تمامی رشته‌های ممکن از حروف را تشکیل می‌دهند. در نتیجه یک اشتباه چاپی در یک کلمه طولانی قابل تشخیص است؛ چرا که این کلمه به چیزی تبدیل می‌شود که کلمه صحیح را بیشتر از هر کلمه مشابه دیگر، به ذهن ما متبادر می‌سازد.

در اینجا مفهوم فاصله بین دو کدواژه را داریم، که عبارت است از تعداد مکان‌های نظیر به نظیری که این دو کدواژه با هم متفاوتند. اگر تمامی کدواژه‌ها به گونه‌ای انتخاب شوند که فاصله هر دو 2 کدواژه حداقل $2t+1$ باشد، آنگاه اگر در ارسال کدواژه‌ها حداکثر t خطا رخ دهد (درواقع احتمال خطا در $t+1$ مکان کمتر از 10^{-5} باشد)، هنوز به کدواژه ارسالی در مقایسه با سایر کدواژه‌ها، نزدیک‌تر بوده و بنابراین می‌توان آن را به درستی به نزدیک‌ترین کدواژه، کدگشایی (decode) کرد.

بنابراین تعریف می‌کنیم: فاصله همینگ $d(x,y)$ بین هر دوکدواژه x و y برابر است با تعداد مکان‌های نظیر به نظیری که با هم تفاوت دارند. اگر برای هر دو کدواژه x و y از مجموعه کد C داشته باشیم

$$2t+1 \leq d(x,y)$$

آنگاه کد C یک کد t تصحیح کننده‌ی خطا نامیده می‌شود.

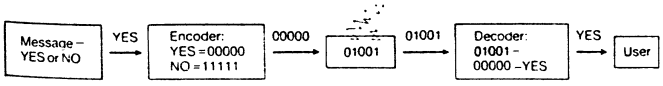
با یک مثال ساده این موضوع را توضیح می‌دهیم. فرض کنید می‌خواهیم در یک کانال مخابراتی خبری را به صورت رشته باینری بدهیم که این خبر "yes" یا "no" است. در حالت عادی با ارسال 0 و 1 به ترتیب، متوجه "yes" و "no" می‌شویم؛ ولی با ارسال 1 به دلیل وجود نویز در کانال مخابراتی، ممکن است 1 به 0 یا بالعکس تبدیل شود. حال، به راه حل مطرح شده می‌پردازیم: ارسال بیت‌های بیشتر در کدواژه‌ها و انتخاب کدواژه‌های به اندازه کافی متفاوت از هم.

همان‌طور که پیش‌تر گفتیم، یکی از مباحثی که در نظریه کدگذاری پرکاربرد است، نظریه احتمالات است. برای همین مثال، کاربرد کوچکی از آن را مطرح می‌کنیم. فرض کنیم در کانال مخابراتی ما احتمال خطا 0.01 است؛ یعنی اگر یک بیت ارسال کنیم، به احتمال 1 درصد احتمال خطا دارد (دقت کنید در اینجا و در نظریه کدگذاری، احتمال خطای هر بیت را مستقل از سایر بیت‌ها در نظر می‌گیریم). چون اینجا تنها به دوکدواژه نیاز داریم (یکی برای "yes" و دیگری برای "no")، کافیست tای انتخاب کنیم که طول دو کدواژه ما $2t+1$ باشد و احتمال خطا در

$t+1$ مکان کمتر از 10^{-5} باشد. پس t باید در رابطه زیر صدق کند: (این رابطه احتمال وجود $t+1$ خطا هنگام انتقال یک کدواژه به طول $2t+1$ بافرض 1 درصد خطا برای هر بیت است.)

$$\binom{2t+1}{t+1}(0.01)^{t+1}(0.99)^{t} < 10^{-5}$$

t=2 در این رابطه برقرار است؛ پس اگر دو کدواژه ما 11111 و 00000 به نمایندگی از "no" و "yes" باشند، در هنگام انتقال دو کدواژه در کانال، حداکثر احتمال دو خطا وجود دارد و چون کدواژه ما به طول 5 است، پس می‌توانیم کدواژه را decode کنیم. به عنوان مثال تصویر



فوق را در نظر بگیرید. در اثر نویز، کدواژه 01001 به کاربر رسیده است که چون به 00000 نزدیک‌تر است، آن را "yes" در نظر می‌گیرد و به این ترتیب کدگشایی می‌شود. گرچه این احتمال وجود دارد که همچنان سایر بیت‌ها دچار خطا شده باشند و راهی برای اطمینان از صحت آن وجود ندارد، ولی احتمال پایین آن قابل صرف نظر کردن است و ما فقط سعی می‌کنیم احتمال دقت را تا آنجا که ممکن است پایین بیاوریم.
در اینجا تنها دو کدواژه نیاز داشتیم اما اگر تعداد کدواژه‌های مورد نیاز بیشتر بود، باید چه راه حلی در پیش بگیریم؟
در ادامه و برای ارائه تعداد کدواژه‌های بیشتر، به ماتریس‌های هادامارد نیاز خواهیم داشت که در اینجا به توضیح مختصری از آن می‌پردازیم.

تعریف ۱.۵ (ماتریس هادامارد). یک $(+1,-1)$ ماتریس، ماتریسی است که همه درایه‌های آن 1 و -1 هستند. یک $(+1,-1)$ ماتریس $H_{n\times n}$، هادامارد از مرتبه n است اگر $H^TH = HH^T = nI$

ماتریس‌های هادامارد در بسیاری از زمینه‌های ترکیبیات مطرح شده و در ارسال عکس از مریخ به زمین از آن‌ها استفاده شده است که جلوتر به آن خواهیم پرداخت.
روش سرراستی برای ساخت ماتریس‌های هادامارد از مرتبه 2^m وجود دارد. فرض کنید

$$H_0 = [1], H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

و برای هر $m \geq 1$، H_m را با استقرا به شکل زیر تعریف کنید (به این روش ساخت، روش دوبرابرسازی می‌گویند):

$$H_m = \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix}$$

در این صورت به سادگی می‌توان دید که H_m یک ماتریس هادامارد از مرتبه 2^m است.
برای ماتریس‌های هادامارد قضیه مهمی داریم که برای اثبات آن و اطلاعات بیشتر درباره آن‌ها می‌توانید به منبع یک مراجعه کنید.

قضیه ۲.۵. *اگر یک ماتریس هادامارد از مرتبه $n > 2$ وجود داشته باشد، آنگاه باید n مضربی از 4 باشد.*

حال به بررسی کدهای با تعداد کدواژه‌های بالاتر می‌پردازیم. دو سیمای متضاد در یک کد وجود دارد. برای یک n مفروض، مطلوب است مینیمم فاصله بین کدواژه‌ها تا حد امکان بزرگ باشد (تا تصحیح خطا را امکان‌پذیر سازد)، از طرف دیگر می‌خواهیم تعداد کدواژه‌ها تا حد امکان زیاد باشد. ولی این دو مطلوب در تضاد با همند. شما نمی‌توانید کدواژه‌های زیادی داشته باشید که دو به دو از هم فاصله زیادی دارند. بنابراین، یک مسئله بنیادی داریم: برای دوعدد مفروض n و k، چند دنباله دوتایی به طول n می‌توان یافت به قسمی که فاصله همینگ هر دو دنباله حداقل k باشد؟
به حالت خاصی از این مسئله، وقتی $k = \lceil \frac{n}{2} \rceil$ می پردازیم. ابتدا حالتی را در نظر می‌گیریم که n فرد است.

قضیه ۳.۵. *فرض کنید N دنباله دوتایی به طول $n = 2m-1$ وجود دارد به طوری که هر دو کدواژه در حداقل m مکان با هم تفاوت دارند. در این صورت $N \leq n+1$.*

اثبات : N دنباله را به عنوان سطرهای یک ماتریس دوتایی $N \times n$، ماتریس با درایه های 0 و 1، در نظر بگیرید.
فرض کنید S معرف مجموع تمامی فاصله‌های همینگ $d(x,y)$ بین واژه‌ها باشد:

$$\Sigma d(x,y)$$

در اینجا مجموع روی تمامی $\binom{N}{2}$ دوتایی متمایز از دنباله های x و y است.
S را به دو روش مختلف می شماریم. برای هر جفت x و y داریم

$$m \leq d(x,y)$$

بنابراین:

$$\binom{N}{2} m \leq S$$

حال ستون زام ماتریس را در نظر بگیرید. اگر این ستون شامل a درایه 0 و b درایه 1 باشد، آنگاه $a+b=N$ و چون هریک از a درایه 0، یک تفاوت با هر یک از b درایه 1 ایجاد می‌کند، سهم این ستون در

مقدار S برابر ab است. بنابراین $S = \sum_{j=1}^{n} ab$. اکنون به آسانی (مثلا با حساب انتگرال یا نابرابری حسابی هندسی) ملاحظه می‌شود که اگر $x + y = N$، بیشترین مقدار xy برابر $\frac{N^2}{4}$ خواهد بود. بنابراین هر ab حداکثر $\frac{N^2}{4}$ است، و داریم:

$$S \leq n.\frac{N^2}{4}$$

از دو رابطه بالا نتیجه می‌گیریم

$$m\frac{N(N-1)}{2} \leq n.\frac{N^2}{4}$$

و از آنجا

$$(2m - n)N \leq 2m$$

یعنی (چون $n = 2m - 1$)

$$N \leq n + 1$$

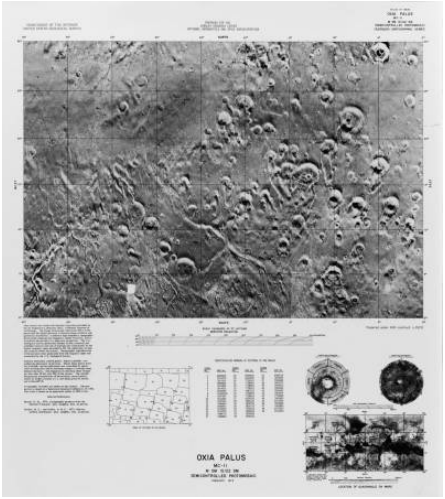
قضیه ۴.۵. *فرض کنید C یک کد به طول $n = 2m$ حاوی N کدواژه بوده که فاصله همینگ هر دو کدواژه حداقل m باشد. آنگاه $N \leq 2n$. علاوه بر این، اگر یک ماتریس هادامارد از مرتبه n وجود داشته باشد، یک کد این چنینی با $2n$ کدواژه موجود است.*

اثبات: کدواژه‌هایی از C را در نظر بگیرید که با 0 شروع می‌شوند. با حذف این 0 از واژه‌های مربوطه، کدواژه‌های به طول $2m - 1$ به دست می‌آوریم که حداقل در m مکان متفاوت هستند. بنابر قضیه ۲.۰، حداکثر $2m$ کدواژه این چنینی وجود دارد. مشابها، حداکثر $2m$ کدواژه از C با 1 شروع می‌شوند. بنابراین C حداکثر $4m = 2n$ کدواژه دارد.

اگر $n = 2$، آنگاه $C = \{11, 10, 01, 00\}$ شرط مطلوب را تامین می‌کند. حال فرض کنید که یک ماتریس هادامارد H از مرتبه $n > 2$ وجود داشته باشد. در این صورت، به ازای عددی چون u خواهیم داشت: $n = 4u$. هردو سطری از H دقیقا در $2u = \frac{n}{2}$ مکان متفاوت و در $\frac{n}{2}$ مکان برابرند.

فرض کنید A ماتریس حاصل از H با تبدیل $1-$ها به 0 باشد و فرض کنید $\overline{A}$ معرف ماتریس حاصل از A با جابه‌جا کردن 0 و 1ها باشد. در این صورت، هر دو سطری از A در حداقل $\frac{n}{2}$ مکان متفاوتند، همچنان که هر دو سطری از $\overline{A}$ چنین هستند و یک سطر دلخواه از A با هر سطر از $\overline{A}$ یا در $\frac{n}{2}$ یا تمامی n مکان تفاوت دارند. از این رو، سطرهای A و $\overline{A}$ کدواژه‌های مطلوب هستند.

حال به مسئله‌ای که فضانوردان در سال 1972 برخورد کردند توجه می‌کنیم. بسیاری از خوانندگان، عکس‌هایی با کیفیت عالی از مریخ،



شکل ۲: عکس ارسالی از مریخ توسط جست‌وجوگر شماره ۹ در سال ۱۹۷۲

زحل و سایر سیاره‌هایی که با ماهواره‌هایی همچون مارینر، وُیاگر و غیره گرفته شده را دیده اند. به منظور انتقال این عکس‌ها به زمین، شبکه‌ای نازک بر روی عکس قرارگرفته و برای هر مربع شبکه، درجه‌ای از تیرگی اندازه گیری می‌شود که از درجه 0 تا 63 است. این اعداد برحسب دستگاه‌های دوتایی بیان می‌شوند؛ یعنی هر مربع، رشته‌ای 6 تایی از 1 و 0 تولید می‌نماید. همان‌طور که پیش‌تر مطرح کردیم، این رشته‌ها براثر نویز (حرارتی) دچار تغییر می‌شوند و تغییرات زیادی بر روی عکس اعمال می‌کنند. برای همین این کدها را با استفاده از اصل افزونگی به کدواژه‌های با طول بیشتر تبدیل کردند. برای ساخت این کدواژه‌ها از ماتریس‌های هادامارد به شکل زیر استفاده کردند:

ماتریس هادامارد H_5 حاصل از روش دوبرابرسازی را در نظر بگیرید. این ماتریس 32 سطر و ستون دارد. فرض کنید A معرف ماتریس حاصل از H_5 با تبدیل $1-$ به 0 بوده، و $\overline{A}$ ماتریس حاصل از جابه‌جا کردن 0ها و 1ها در ماتریس A باشد. در این صورت، سطرهای A و $\overline{A}$ یک کد 64 واژه‌ای به طول 32 می‌سازند که هردو کدواژه، در حداقل 16 مکان تفاوت دارند؛ بنابراین، این کد 7 تصحیح کننده‌ی خطاست.

دنباله درجه‌های کد شده با استفاده از این کد 7 تصحیح کننده‌ی خطا، کدگذاری شدند و عکس‌های دریافت شده فوق‌العاده خوب بودند! در واقع برای هر کدام از اعداد 0 تا 63 آن را به کدواژه‌ای به طول 32 تناظر دادیم که هر دو کدواژه در حداقل 16 مکان تفاوت دارند و در هنگام انتقال اگر 7 مکان یک کدواژه تغییر کنند، باز هم قابل کدگشایی (decode) است.

فضاپیماهای جدیدتر از کدهای پیچیده‌تری استفاده می‌کنند، همچنان که دیسک‌های فشرده و دیگر وسایل مدرن از این کدهای پیچیده استفاده کرده‌اند.

مراجع

[1] Anderson, Ian. *A first course in combinatorial mathematics*. Clarendon Press, 1989.

[2] Hill, Raymond. *A first course in coding theory*. Oxford University Press, 1986.

آشنایی با گرایش آنالیز عددی

محمدسروش غفاری

هدف از این متن آشنایی مقدماتی با گرایش آنالیز عددی در رشته‌ی ریاضی کاربردی است.

درباره‌ی گرایش

گرایش آنالیز عددی[1] زیرمجموعه‌ای از ریاضی کاربردی به حساب می‌آید. ریاضی کاربردی سعی دارد تا ما را به کاربرد ریاضی در رشته‌های دیگر آشنا کند. در واقع ریاضی کاربردی پُلی بین ریاضیات و رشته‌های دیگر اعم از مکانیک، برق، اقتصاد، زیست‌شناسی[2]، کامپیوتر و... است.

تقریباً بدون اغراق می‌توان گفت در این رشته، روش‌ها، تحقیقات، الگوریتم‌ها و بررسی نتایجی که انجام می‌شود به هر رشته و هر پدیده‌ای می‌تواند ارتباط پیدا کند و آن را پوشش ریاضیاتی دهد! شما می‌توانید کاربرد ریاضی را در اکثریت علوم، از پزشکی گرفته تا کامپیوتر، ببینید، معادلات و نتایج امواج را بررسی کنید، آب و هوا را مورد تحلیل قرار دهید، خطاها را بررسی کنید همچنین علل بوجود آمدن خطا در معادلات و بسیاری کار دیگر که همه‌ی آن‌ها توسط ریاضی اتفاق می‌افتد!

درواقع مثال معروفی که ریاضی **مادر** علوم است در اینجا تحقق پیدا می‌کند؛ البته نباید از سختی‌های راه بگذریم!

پیش‌نیازها

برای تحصیل در این گرایش، بعد از علاقه داشتن باید پیش‌نیازهایی را بدانید. (بهتر است در دوره کارشناسی آن‌ها را گذرانده باشید.)

این پیش‌نیازها شامل دروسی مثل معادلات دیفرانسیل معمولی (که تقریباً در کارشناسی مهندسی و علوم پایه جزو دروس ضروری‌ست)، معادلات دیفرانسیل با مشتقات جزئی، جبرخطی عددی، مبانی آنالیز عددی و آنالیز عددی است! بازهم لازم به ذکر است که ممکن است توانایی آن را داشته باشید که بدون گذراندن یکی از این دروس اصلی به مقطع ارشد بیایید و با مطالعه از قبل یا در طول ترم به دانش کافی برسید و به مطالب مسلط شوید که همه به خود شما بستگی دارد، چون مقطع ارشد نسبتاً کوتاه است و دروس سنگین هستند، امّا هیچ چیز غیرممکن نیست!

البته که بسته به دانشگاه مورد نظر این استانداردها و دروس ممکن است متفاوت باشند و حتی سطح علمی و تصمیمات هیئت علمی دانشگاه در رشته‌ی ریاضی نیز از عوامل تأثیرگذار است! در اینجا مورد بحث ما به صورت کلی درباره‌ی دانشگاه صنعتی امیرکبیر است.

یکی از مهمترین چیزهایی که باید در گرایش آنالیز عددی به آن تسلط کافی داشته باشید برنامه‌نویسی است؛ به خصوص برنامه‌نویسی به زبان متلب [3] که باید بتوانید توانایی تحلیل روش‌ها و الگوریتم‌های حل مسائل و توانایی پیاده‌سازی آن‌ها در متلب را داشته باشید!

درست است که متلب به‌عنوان بهترین زبان برنامه‌نویسی در این حوزه مورد استفاده قرار می‌گیرد، امّا زبان‌های مورد استفاده‌ی مهم و کاربردی دیگری نیز وجود دارد. مانند:

1.Maple

2.Mathematica

3.Fortran

5.Python

6.Julia

با چه دروسی مواجه خواهیم شد؟!

دروسی که در کارشناسی ارشد یا تحصیلات تکمیلی باید بگذرانید بسته به اساتید راهنما، استاد مشاور و موضوعی که قرار است برای پایان‌نامه و تحقیقات آتی برحسب علاقه یا توانایی بر روی آن کار کنید، متفاوت است. از آن‌جا که ریاضی کاربردی بین رشته‌ای می‌باشد، پس ممکن است دانشجویی مشتاق به دنبال کردن مباحث و جزئیات ریاضی در مکانیک باشد، دانشجوی دیگری کاربرد ریاضی در هوش مصنوعی، یا حتی ریاضی در زیست‌شناسی و خیلی موضوعات دیگر که به قدری گسترده هستند که می‌توان در چندین کتاب درباره آن‌ها توضیح داد و مطلب نوشت! حتی شما می‌توانید در زمینه‌ی موضوعاتی که ریاضیات کاربردی را به ریاضی محض مرتبط می‌کنند، کار کنید و مطالعه داشته باشید.

دروس اصلی که در ابتدای راه تحصیلات تکمیلی باید آن‌ها را گذراند شامل آنالیز عددی پیشرفته، روش‌های عددی در جبرخطی و درس آنالیز

[1] Numerical Analysis
[2] Biology
[3] MATLAB

حقیقی می‌باشد.

درس آنالیز حقیقی یک درس اجباری (تقریباً در همه‌ی دانشگاه‌ها) و با فضای محض می‌باشد. درس آنالیز عددی پیشرفته ادامه و تکمیل کننده دروس آنالیز عددی در کارشناسی است.

در ادامه دروسی مانند روش‌های عددی در جبرخطی، حل عددی معادلات دیفرانسیل با مشتقات جزئی، روش‌های بدون شبکه، عناصر متناهی و دروسی مانند مباحثی در جبرخطی، مباحثی در آنالیز عددی و دروس دیگری که در هر دانشکده متفاوت است، وجود دارند که همانطور که گفتیم بنا به تشخیص استاد راهنما و استاد مشاور طبق پایان‌نامه‌ی شما این دروس باید گذرانده شود.

در مقطع کارشناسی ارشد شما باید ۳۲ واحد شامل ۲۴ واحد درسی، ۲ واحد سمینار و ۶ واحد پایان‌نامه را به طور معمول در ۲ سال تحصیلی بگذرانید پس وقت برای گذراندن تمام واحدهای درسی ارائه شده نیست! از این جهت تحت نظر اساتید مناسب‌ترین دروس انتخاب واحد می‌شوند و طبق آن‌ها مسیر تحقیقات خود را ادامه خواهید داد.

دروس ارائه شده شامل الگوریتم‌ها، روش‌ها و حل مسائل و معادلات و... هستند. در هر درس یاد می‌گیرید متفاوت‌تر بیندیشید و عمل کنید. گاهاً روش‌ها در دو درس یکسان هستند، اما فضای معادلات و بررسی پدیده‌ها متفاوتند.

گرایش آنالیز عددی مخصوصاً هنگامی که مطالعات به سمت و سوی معادلات دیفرانسیل جزئی و روش‌های آن‌ها کشیده می‌شوند دریچه‌ی تازه‌ای از فضای ریاضیاتی که در تفکر عام به‌عنوان تنها اعداد و فرمول گماشته شده به‌روی دانشجویان باز می‌کند. این گرایش سالانه خروجی‌های بسیاری در زمینه مقالات جدید دارد و یکی از محبوب‌ترین شاخه‌ها برای کار کردن می‌باشد به طوری که از باقی رشته‌های مهندسی به دروس این گرایش رجوع می‌کنند تا بتوانند تحقیقات و نتایج بهتر و دقیق‌تری ارائه دهند!

وقتی کاربرد هر روش و هر معادله را در مقاطع دیگر می‌بینیم، ذهن ما تفکری جدید را تجربه می‌کند و می‌اندیشیم هر چیز برحسب همین اعداد و فرمول‌ها می‌تواند نشان داده شود. گاهی پِی می‌بریم زندگی و هر چیز در اطراف ما برحسب ریاضیات پدید آمده و باقی شاخه‌ها را شکل داده است، پس خیلی چیزها نیازمند ریاضیات و اعدادش است!

سخن آخر

در این چند خط که گفتیم، سعی در این داشتیم تا کمی با این گرایش و نحوه ورود به آن، پیش‌نیازها و اطلاعات مقدماتی آشنا شوید.

امید است تا در آینده به کاربردهای این گرایش و معادلات جالب و خیلی از کاربردهایی که تنها به گفتن آن‌ها بسنده کردیم، بپردازیم. کم‌کم به سراغ مطالب جذّاب که ریاضی را از حالت خشک (!) درمی‌آورد، می‌رویم و می‌بینیم در هر پدیده‌ای می‌توان تحلیل و تفکر ریاضی انجام داد.

چند عدد قابل شناسایی در جهان ریاضیات وجود دارد؟
بی‌نهایت اثبات مختلف، ریاضی را به یافتن پاسخی به این پرسش نزدیک‌تر کرده‌است.

فریماه شاه‌محمدیان

برای ۵۰ سال ریاضی‌دانان بر این باور بودند که دانستن تعداد دقیق اعداد حقیقی دست‌نیافتنی است؛ اما اثباتی جدید خلاف این را می‌گوید.

در اکتبر ۲۰۱۸، دیوید آسپرو[1] در تعطیلات خود در ایتالیا و درحالی‌که از پنجره اتومبیل به بیرون نگاه می‌کرد، این به ذهنش رسید: پیوند گم‌شده چیزی که که اکنون اثباتی جدید و برجسته درباره اندازه بی‌نهایت است. او می‌گوید: «این یک تجربه روشن و آنی بود.»

آسپرو، ریاضیدانی در دانشگاه شرق انگلستان(UEA)، با رالف اشنایدر[2] در دانشگاه مونستر آلمان، همکاری که او هم به دنبال این اثبات بود تماس گرفت و دیدگاه خود را برای او توضیح داد. اشنایدر می‌گوید: «این کاملاً برای من نامفهوم بود»، ولی در نهایت آنها، دو نفره این فانتزی را به منطقی محکم تبدیل کردند.

اثبات آن‌ها، که در ماه می ۲۰۲۱ در سالنامه ریاضیات[3]، منتشر شد، دو اصل رقیب که به عنوان مبانی رقابت‌کننده ریاضیات نامتناهی مطرح شده بودند را متحد می‌کند. آسپرو و اشنایدر نشان دادند که یکی از این اصول بر دیگری دلالت دارد و همچنین احتمال درست بودن این اصول (و هرچه آن‌دو درباره بی‌نهایت مطرح می‌کنند) را بیشتر کردند.

مناخم مگیدور[4]، منطق‌ریاضی‌دانی از دانشگاه عبری اورشلیم در این‌باره می‌گوید: «این اثبات فوق‌العاده است. صادقانه، من در تلاش بودم که خود به آن برسم.»

مهم‌تر از همه این نتیجه بر علیه فرضیه پیوستار[5] بود؛ حدسی بسیار تأثیرگذار در سال ۱۸۷۸ که درباره اندازه مجموعه‌های نامتناهی است. هردوی این اصول که در اثبات جدید همگرا شده‌اند، نادرستی فرض پیوستار و وجود مقدار دیگری بین این دو مقدار که ۱۴۳ سال پیش به عنوان اولین و دومین اعداد بی‌نهایت بزرگ حدس زده شده بودند را نشان می‌دهد.

الیخاس فراه[6]، ریاضیدانی از دانشگاه یورک تورنتو می‌گوید: «ما هم‌اکنون جایگزین منسجمی برای فرضیه پیوستار داریم.»

این نتیجه، پیروزی بزرگی برای گروهی از ریاضیدانان که عمیقاً حس می‌کردند فرض پیوستار غلط است، محسوب می‌شود. ژولیت کندی[7]، منطق‌ریاضی‌دان و فیلسوف از دانشگاه هلسینکی می‌گوید: «این نتیجه به طرز چشمگیری تصویر را روشن می‌کند.»

اما گروه دیگر، طرفدار چشم‌انداز دیگری از ریاضیات هستند که فرض پیوستار در آن قرار دارد و درست می‌باشد. ولی نبرد بین این دو گروه فاصله زیادی با پیروزی دارد.

کندی می‌گوید: «زمان فوق‌العاده‌ای است. این یکی از هیجان‌انگیزترین و دراماتیک‌ترین اتفاقاتی است که در تاریخ ریاضیات افتاده است».

بی‌نهایتی از بی‌نهایت‌ها

بله، بی‌نهایت مقادیر مختلفی دارد. در سال ۱۸۷۳، ریاضیدان آلمانی، جورج کانتور[8] ، با کشف خود ریاضیات را شدیداً تحت تاثیر قرار داد. او کشف کرد تعداد اعداد حقیقی که محور اعداد را (بیشتر با اعدادی که قسمت اعشاری آنها پایانی ندارد مانند ...۳/۱۴۱۵۹۲) بیشتر از اعداد طبیعی مانند ۱، ۲و۳ است علیرغم اینکه از هردوی آنها بی‌نهایت وجود دارد.

مجموعه‌های نامتناهی اعداد، شهود ما از اندازه را آشفته می‌کنند، برای شروع، دو مجموعه اعداد طبیعی $\{1, 2, 3, ...\}$ و اعداد فرد $\{1, 3, 5, ..\}$ را مقایسه کنید. ممکن است فکر کنید مجموعه اول از دومی بزرگ‌تر است؛ زیرا مجموعه دوم فقط نصف اعضای مجموعه اول را داراست. کانتور متوجه شد علی‌رغم این موضوع می‌توان اعضای این دو مجموعه را در تناظر یک‌به‌یک با یکدیگر قرار داد. شما می‌توانید عضو اول از هر مجموعه را با هم متناظر کنید (۱ و ۱) و سپس اعضای دوم را با هم (۲و۳) و اعضای سوم را با هم (۳و ۵) و تا ابد پیش بروید و تمام اعضای هردو مجموعه را پوشش دهید. با این استدلال، دو مجموعه نامتناهی، اندازه یا آن‌طور که کانتور نام‌گذاری کرد، «کاردینالیتی» برابر دارند. او اندازه این مجموعه‌ها را با عدد کاردینال $\aleph_0$ (الف صفر) نشان داد.

اما کانتور کشف کرد که اعداد طبیعی را نمی‌توان در تناظر یک‌به‌یک با اعداد حقیقی پیوستار قرار داد. برای مثال بیاید سعی کنیم ۱ را با

[1] David Asperó
[2] Ralf Schindler
[3] سالنامه ریاضیات نشریه ریاضیاتی است که هر دو ماه یک بار توسط دانشگاه پرینستون و مؤسسه مطالعات پیشرفته منتشر می شود.
[4] Menachem Magidor
[5] فرض پیوستار می‌گوید: هیچ مجموعه‌ای وجود ندارد که اندازه‌اش بین اندازه مجموعه اعداد طبیعی و اندازه مجموعه اعداد حقیقی باشد.
[6] Ilijas Farah
[7] Juliette Kennedy
[8] Georg Cantor

...۱/۰۰۰۰۰۰ و ۲ را با ...۱/۰۰۰۰۱ متناظر قرار دهیم. ما در اینجا بی‌نهایت عدد حقیقی (مانند...۱/۰۰۰۰۰۰۰۰۰۰۱) را جا انداخته‌ایم. پس امکان ندارد که شما بتوانید تمام اعداد حقیقی را بشمارید؛ در واقع کاردینالیتی اعداد حقیقی از اعداد طبیعی بزرگ‌تر است.

مقادیر بی‌نهایت در اینجا متوقف نمی‌شوند. کانتور کشف کرد که مجموعه توانی هر مجموعه نامتناهی(مجموعه تمام زیرمجموعه های یک مجموعه)، کاردینالیتی بزرگ‌تری از خود آن مجموعه دارد. همچنین هرمجموعه توانی خود دارای مجموعه توانی می‌باشد. پس می‌توان گفت این اعداد کاردینال، برجی بی‌نهایت بلند از بی‌نهایت‌ها می‌سازند.

کانتور در حالی که در پایین این بنای ممنوعه ایستاده بود، روی طبقات ابتدایی آن تمرکز کرد. او موفق شد ثابت کند مجموعه‌ای که از همه‌ی روش‌های مختلف چیدن اعداد طبیعی (مثلاً از کوچک به بزرگ، یا ابتدا اعداد فرد، سپس زوج) تشکیل شده است، دارای کاردینالیتی $\aleph_1$ است و یک طبقه بالاتر از اعداد طبیعی قرار دارد. علاوه بر این، هرکدام از این «انواع ترتیب» عددی حقیقی را رمزنگاری می‌کنند.

فرضیه پیوستار او ادعا می‌کند این دقیقاً اندازه پیوستار است به این معنا که دقیقا $\aleph_1$ عدد حقیقی وجود دارد. به زبانی دیگر کاردینالیتی پیوستار بلافاصله بعد از $\aleph_0$ می‌نشیند و هیچ مقدار بی‌نهایت دیگری بین این‌دو وجود ندارد.

اما کانتور به دلیل مشکلات و گرفتاری‌های فراوان نتوانست این را اثبات کند.

در سال ۱۹۰۰، دیوید هیلبرت فرضیه پیوستار را در اول لیست معروفش از ۲۳ مسئله‌ای که باید در قرن بیستم حل می‌شدند قرار داد. هیلبرت شیفته ریاضیات نامتناهی جدید بود و آن را «بهشت کانتور» نامید و فرضیه پیوستار مانند میوه‌ای در این بهشت به نظر می‌رسید که به راحتی می‌شد به آن دست یافت.

اما خلاف این اتفاق افتاد و انقلاب‌های شوکه‌کننده‌ای که در این قرن اتفاق افتادند، سؤال کانتور را به معمایی عمیق تبدیل کردند.

مشکل در سال ۱۹۳۱ آغاز شد، هنگامی که کورت گودل[9]، ریاضیدان اتریشی کشف کرد که هر مجموعه‌ای از اصول موضوع که به عنوان پایه‌ای برای ریاضیات مطرح شوند به طور گریزناپذیری ناقص خواهد بود و همیشه سؤالاتی وجود خواهند داشت که این اصول نمی‌توانند حلشان کنند؛ مانند قضیه‌های درست ریاضی که نمی‌توانند اثبات کنند.

گودل بلافاصله مشکوک شد که فرضیه پیوستار مسئله‌ای مستقل از اصول موضوع ریاضیات است.

این اصول موضوع که ده عدد هستند، با نام ZFC شناخته می‌شوند و زیربنای تقریبا تمام ریاضیات مدرن هستند. این اصول، ویژگی‌های بنیادی گردایه‌ای از اشیا یا مجموعه‌ها را شرح می‌دهند. از آنجایی که می‌توان تقریباً هرچیزی در ریاضیات را به‌وسیله مجموعه‌ها ساخت (به طور مثال، مجموعه تهی $\{\}$ ، صفر، مجموعه $\{\{\}\}$ ، یک، مجموعه $\{\{\},\{\{\}\}\}$ ،دو را نشان می‌دهند و همین‌طور به ترتیب می‌توان پیش رفت.) قوانین مجموعه‌ها برای ساخت اثبات‌های ریاضیاتی تقریباً کافی هستند.

در سال ۱۹۴۰، گودل نشان داد نمی‌توان با استفاده از ZFC، فرضیه پیوستار را رد کرد و در سال ۱۹۶۳ ریاضیدان آمریکایی، پاول کوهن[10] عکس این قضیه را نشان داد؛ یعنی فرضیه پیوستار را با ZFC اثبات نیز نمی‌توان کرد. اثبات کوهن و گودل همراه هم نشان می‌دهند که فرضیه پیوستار و اصول موضوع ZFC مستقل از یکدیگر است.

علاوه بر فرضیه پیوستار، مشخص شد بیشتر دیگر سؤالات درباره مجموعه‌های نامتناهی مستقل از ZFC هستند. این عدم وابستگی گاهی اوقات این‌طور تفسیر می‌شود که این سؤالات پاسخی ندارند؛ اما بیشتر نظریه‌پردازان نظریه مجموعه‌ها این را یک سوءتفاهم عمیق می‌دانند.

آنها معتقدند که پیوستار اندازه دقیقی دارد و ما تنها به ابزارهای جدیدی از منطق نیاز داریم تا آن را بیابیم. این ابزارها به شکل اصول موضوع جدید به دست خواهند آمد. مگیدور می‌گوید: «اصول موضوع، این سوالات را حل نمی‌کنند؛ پس باید آن‌ها را به سیستم اصول موضوع غنی‌تری گسترش دهیم.»

از زمان کوهن، نظریه‌پردازان نظریه مجموعه‌ها تلاش کرده‌اند با اضافه‌کردن حداقل یک اصل جدید به ZFC پایه‌های ریاضیات نامتناهی را تقویت کنند. این اصل (یا اصول) باید ساختار مجموعه‌های نامتناهی را روشن کند، قضیه‌های طبیعی و زیبایی بسازد، باعث به وجود آمدن تناقضات ویرانگر نشود و البته سؤال کانتور را حل کند.

گودل به نوبه خود معتقد بود فرضیه پیوستار نادرست است و تعداد اعداد حقیقی از آنچه کانتور حدس زده است، بیشتر است. او مشکوک بود که $\aleph_2$ از آنها وجود دارد. او همان‌طور که در سال ۱۹۴۷ نوشت، پیش‌بینی کرده‌است: نقش مسئله پیوستار در نظریه مجموعه‌ها این خواهد بود: بالاخره این مسئله ما را به سمت کشف اصول موضوع جدید راهنمایی می‌کند که رد حدس کانتور را ممکن می‌سازند.

منبع نور

دو اصل رقیب فقط برای انجام همین کار ساخته شدند. برای دهه‌ها، آن‌دو مشکوک به این بودند که از نظر منطقی کامل نیستند. اشنایدر می‌گوید: «این تنش همیشه وجود داشته است.»

[9] Kurt Gödel

[10] Paul Cohen

برای فهمیدن آن‌ها، باید به کار کوهن در سال ۱۹۶۳ برگردیم، هنگامی که او تکنیکی به نام فورسینگ را ایجاد کرد و توسعه داد. کوهن با مدلی از جهان ریاضی شروع کرد که در آن $\aleph_0$ عدد حقیقی وجود دارد تا پیوستار را توسعه دهد تا شامل اعداد حقیقی جدیدی فراتر از آن مدل شود. کوهن و معاصرانش به زودی متوجه شدند بسته به ویژگی‌های روش، فورسینگ این اجازه را به شما می‌دهد تا هرچقدر دوست دارید ($\aleph_2$ یا $\aleph_{35}$) عدد حقیقی اضافه کنید. علاوه بر اعداد حقیقی جدید، ریاضیدانان روش کوهن را برای حدس رفتار دیگر چیزهای ممکن که از لحاظ منطقی کامل نیستند تعمیم دادند. این، جهان‌های چندگانه‌ای از جهان‌های ممکن ریاضیاتی ساخت.

هیو وودین[11] نظریه‌پرداز نظریه مجموعه‌ها از دانشگاه هاروارد می‌گوید: «روش او ابهامی در جهان مجموعه‌های ما به وجود می‌آورد. این ابری از جهان‌های مجازی می‌سازد و من چطور می‌توانم بفهمم در کدام جهانم؟»

کدام یک واقعی و کدام یک مجازی بود؟ کدام یک از چیزهای متناقض که با روش فورسینگ تولید شده بودند باید قابل قبول واقع می‌شدند؟ این واضح نبود که هرچیزی فقط به این خاطر که با روش کوهن قابل تصور است واقعاً وجود دارد یا خیر.

برای حل این مشکل، ریاضیدانان «اصول موضوع فورسینگ» مختلفی مطرح کردند. قوانینی که وجود چیزهایی که با روش کوهن ارائه می‌شوند را تعیین می‌کردند. مگیدور توضیح می‌دهد: «اگر بتوانید تصور کنید چیزی وجود دارد، پس وجود دارد.»این یک اصل شهودی راهنماست که به اصول موضوع فورسینگ منجر می‌شود. در سال ۱۹۸۸ مگیدور، متیو فورمن و سحرون شیلاه با مطرح کردن «ماکزیمم مارتین» این ایده را به نتیجه‌ای منطقی تبدیل کردند که می‌گوید هرچیزی که بتوان آن را هر روشی از فورسینگ تولید کرد به شرط اینکه این روش یک وضعیت مشخص توافق شده‌ای شده داشته باشد، یک نتیجه درست ریاضیاتی خواهد بود.

در چهارچوب گستردگی ماکزیمم مارتین، برای اینکه به طور همزمان تمام محصولات فورسینگ (هنگامی‌که شرایط مشخصی دارند) تحقق پیدا کنند، اندازه پیوستار در خوش‌بینانه‌ترین حالت به $\aleph_2$ که یک عدد کاردینال بیشتر از کمترین مقدار ممکن است جهش پیدا می‌کند.

علاوه بر حل مسئله پیوستار، ماکزیمم مارتین ثابت کرد که می‌تواند ابزاری قدرتمند برای کشف ویژگی‌های مجموعه‌های نامتناهی می‌باشد. طرفداران می‌گویند این، گزاره‌های گسترده و قضایای کلی بسیاری را ایجاد می‌کند. در مقابل اگر فرض کنیم پیوستار دارای کاردینالیتی $\aleph_1$ است با استثنائات و موانع زیادی برای اثبات مواجه می‌شویم، چیزی که مگیدور به آن «بهشت مثال‌های نقض» می‌گوید.

ماکزیمم مارتین، به عنوان بسطی از ZFC بسیار محبوب شد ولی در دهه ۹۰، وودین اصل موضوع قانع‌کننده دیگری مطرح کرد که همزمان فرضیه پیوستار را نابود می‌کند و اندازه پیوستار را $\aleph_2$ تعیین می‌کند؛ اما به روشی کاملاً متفاوت. وودین این اصل را «ستاره (*)» نامید. او به من گفت: «این مانند منبعی روشن است، منبعی از ساختارها، منبعی از نور.»

(*) مدلی از جهان ریاضیات را در نظر می‌گیرد که از ۹ اصل ZFC را به علاوه اصل موضوع مشخصه‌سازی به جای اصل موضوع انتخاب پیروی می‌کند. مشخصه‌سازی و انتخاب از لحاظ منطقی با یکدیگر در تناقض است و به همین دلیل است که (*) و ماکزیمم مارتین ناسازگار به نظر می‌رسند؛ اما وودین روشی فورسینگ را اختراع کرد و به وسیله آن مدل جهان ریاضیاتی خود را به مدلی بزرگ‌تر گسترش داد که با ZFC در تناقض نباشد. در این جهان بود که (*) درست بود.

چیزی که (*) را بسیار روشنگر می‌کرد این بود که به ریاضیدانان اجازه می‌داد با مراجعه به ویژگی‌های مجموعه‌ها در دامنه‌شان، گزاره‌هایی به شکل «برای همه X ها، Y وجود دارد به طوری‌که Z» بسازند. چنین گزاره‌هایی روش قدرتمندی از استدلال ریاضی هستند. یکی از این گزاره‌ها این است: برای هر مجموعه از $\aleph_1$ عدد حقیقی، عدد حقیقی‌ای وجود دارد که در آن مجموعه‌ها نیست. این برخلاف فرضیه پیوستار است؛ بنابراین طبق (*) حدس کانتور نادرست است. این واقعیت که (*) به ریاضیدانان اجازه می‌دهد نادرستی فرضیه پیوستار و همچنین بسیاری از ویژگی‌های دیگر مجموعه‌های اعداد حقیقی را نتیجه بگیرند، آن را به گفته اشنایدر به «فرضیه‌ای جذاب» تبدیل کرده‌است.

با این دو اصل شدیداً مولد، طرفداران فورسینگ با نتایج زیاد و آشفته‌ای روبه‌رو شدند. شنیدلر می‌گوید: «هردوی این اصول (ماکزیمم مارتین و *) زیبا هستند و درست و طبیعی به نظر می‌رسند. شما کدام یک را انتخاب می‌کنید؟»

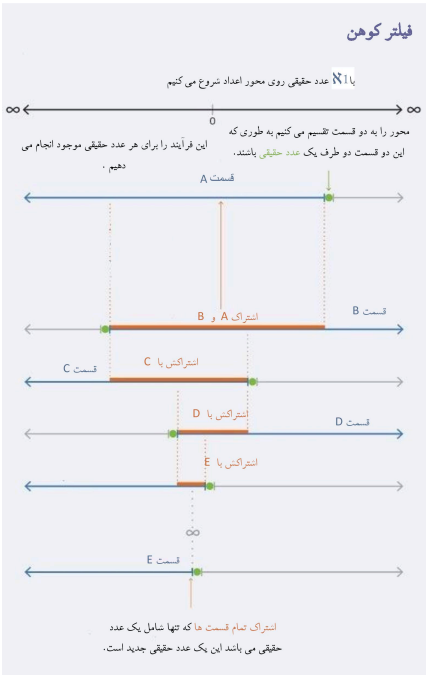
اگر این اصول با یکدیگر در تناقض باشند، قبول کردن یکی از آن‌ها به معنای قربانی کردن دیگری و نتایج خوبش است و داوری بین آن‌ها ممکن است مستبدانه به نظر برسد. اشنایدر می‌گوید: «شما باید دلایلی بیاورید که یکی از آن‌ها درست و دیگری نادرست است یا شاید هردوی آن‌ها نادرست هستند».

در عوض، کار جدید او و آسپرو نشان می‌دهد ماکزیمم مارتین++ (یک تکنیک متفاوت از ماکزیمم مارتین) بر (*) دلالت دارد. اشنایدر می‌گوید: «اگر شما این نظریه‌ها را همان طور که ما انجام دادیم متحد کنید، حدس من این است که شما می‌توانید بگویید ممکن است مردم یک چیز را درست فهمیده باشند.»

[11] Hugh Woodin

فورسینگ چیست؟

برای آنکه شیوه کار فورسینگ را متوجه شوید، تصور کنید $\aleph_1$ عدد حقیقی روی محور اعداد وجود دارند. شما می‌توانید محور را به بی‌نهایت روش برش بزنید. برای مثال می‌توانید آن را به دو قسمت تقسیم کنید، به طوری‌که این دو قسمت دو طرف یک عدد حقیقی باشند. همین کار را برای هر عدد حقیقی دیگر انجام دهید. کوهن فیلتری را در نظر گرفت که یک قسمت از دو قسمت هر تقسیم‌بندی ممکن را انتخاب می‌کند و هربار اشتراک قسمت‌های انتخاب شده را به‌دست می‌آورد. هنگامی که این فرآیند کامل شود، اشتراک تمام قسمت‌ها بسیار بسیار کوچک خواهد شد (مجموعه‌ای شامل یک عدد حقیقی). اما به این خاطر که هر عدد حقیقی که ما با آن شروع کردیم، یکی از نقاطی بود که به‌وسیله آن محور را به دو قسمت تقسیم کردیم. (به همین خاطر این عدد در قسمت هایی که انتخاب کردیم وجود ندارد.) اشتراک تمام قسمت‌ها نمی‌تواند یکی از اعداد حقیقی که ما با آن شروع کردیم باشد. این یک عدد حقیقی جدید است. با تعمیم دادن این روش شما می‌توانید پیوستار را گسترش دهید تا هر تعداد عدد حقیقی که دوست دارید را شامل شود.



پیوند گم‌شده

آسپرو و اشنایدر، ۲۰ سال پیش محققان جوانی در موسسه‌ای در وین بودند. اثبات آن‌ها چند سال بعد جوانه زد هنگامی که اشنایدر دست‌نوشته‌ای از نظریه‌پزدار نظریه مجموعه‌ها، رونالد جنسن، خواند. در آن، جنسن تکنیکی به نام«L فورسینگ» اختراع کرده بود. اشنایدر تحت تاثیر آن قرار گرفت و از دانشجویانش خواست تا سعی کنند آن را بیشتر گسترش دهند. پنج سال بعد در سال ۲۰۱۱، او روش L فورسینگ را برای آسپرو که او را در دانشگاه مونستر می‌دید، توضیح داد. آسپرو بی‌درنگ پیشنهاد کرد که شاید بتوانند با استفاده از این تکنیک، (*) را از ماکزیمم مارتین نتیجه بگیرند.

دو سال بعد در سال ۲۰۱۲ آنها اعلام کردند که اثباتی دارند. بلافاصله وودین اشتباهی در آن یافت و آنها با خجالت آن را پس گرفتند. آن‌ها در سال‌های بعد به بررسی مجدد اثبات پرداختند؛ اما همواره دریافتند یک ایده کلیدی را کم دارند طبق گفته آسپرو آن «پیوند گم‌شده» در زنجیره منطقی‌ای بود که ماکزیمم مارتین را به (*) می‌رساند.

نقشه آن‌ها برای نتیجه‌گیری اصل دوم از اصل قبلی این بود که روش فورسینگی مانند L فورسینگ ایجاد کنند تا با آن چیزی را تولید کنند که آن را مدرک نامیدند. این مدرک تمام گزاره‌هایی که فرم (*) دارند را تأیید می‌کند. تا زمانی که روش فورسینگ از شرایط لازم و توافق شده پیروی کند، ماکزیمم مارتین ثابت می‌کند که وقتی می‌توان با فورسینگ مدرک را به وجود آورد، پس آن وجود دارد و به دنبال آن (*) نتیجه می‌شود.

آسپرو می‌گوید: «ما می‌دانستیم چگونه همچین فورسینگی بسازیم.» اما آنها نمی‌توانستند بفهمند چگونه تضمین کنند که روش فورسینگ آن‌ها نیاز کلیدی ماکزیمم مارتین را برآورده می‌کند. در سال ۲۰۱۸، «تجربه آنی و روشن» آسپرو در ماشین بالاخره راه را نشان داد: آنها می‌توانستند فورسینگ را به دنباله‌ای بازگشتی از فورسینگ‌ها تجزیه کنند که هرکدام شرایط ضروری را مهیا می‌کند. او می گوید: «به یاد دارم که بسیار مطمئن بودم که این کار جزئی باعث می‌شود اثبات کار کند.» اگرچه بعد از آن آسپرو و اشنایدر هردوی آن‌ها بسیار تلاش کردند تا اثبات به ثمر برسد.

ستاره‌های دیگر

همگرایی ماکزیمم مارتین و (*) پایه‌ی محکمی برای برجی از بی‌نهایت‌ها می‌سازد که در آن کاردینالیتی پیوستار $\aleph_2$ است. پیتر کولنر[12]، نظریه‌پرداز نظریه مجموعه‌ها در دانشگاه هاروارد می‌پرسد: «سؤال اینجاست که آیا این درست است یا خیر؟»

طبق گفته کولنر، دانستن این‌که قوی‌ترین اصل فورسینگ در (*) دلالت دارد می‌تواند هم سندی برای اثبات آن و هم برعلیه آن باشد. او می‌گوید: «این بستگی به آن دارد که شما (*) را چه چیزی در نظر بگیرید.»

نتیجه همگرایی، ویژگی های معقول (*) را مورد بررسی دقیق قرار می‌دهد. زیرا (*) به ریاضیدانان اجازه می‌دهد گزاره‌های قدرتمند «برای تمام X ها Y وجود دارد.» را بسازند که نتایجی از ویژگی‌های اعداد حقیقی را به همراه دارند.

علیرغم سودمندی بسیار (*) در ساخت آن گزاره‌ها، که به نظر

[12] Peter Koellner

تناقضی در آن‌ها وجود ندارد، کولنر از کسانی است که به این اصل مشکوک است. یکی از پیامدهایش (شبیه‌سازی ساختار یک کلاس بزرگ مشخص از مجموعه‌ها با تعداد زیادی مجموعه کوچک‌تر) است که به نظر او عجیب است.

به طور قابل ملاحظه‌ای، کسی که باید بیشترین اشتیاق را برای درست بودن (*) داشته باشد هم بر علیه آن شد. وودین در جلسه‌ای که این تابستان در زوم داشتیم گفت: «من یک خائن محسوب می‌شوم.»

۲۵ سال پیش هنگامی که او (*) را مطرح کرد فکر می‌کرد که فرضیه پیوستار نادرست است و بنابراین (*) مانند منبعی از نور است اما در دهه اخیر نظر او تغییر کرد. او حالا معتقد است پیوستار دارای کاردینالیتی $\aleph_1$ و (*) و فورسینگ «محکوم» هستند.

وودین اثبات آسپرو اشنایدر را «یک نتیجه خارق‌العاده» می‌نامد که شایستگی اینکه در سالنامه ریاضیات منتشر شود را دارد. (سالنامه ریاضیات به طور گسترده‌ای به عنوان برترین مجله ریاضیات در نظر گرفته می‌شود.) همچنین او اعتراف می‌کند که این نوع نتیجه همگرایی «به عنوان گواهی بر نوعی از حقیقت تلقی می‌شود»، اما طرفدار آن نیست. در کار آنها مشکلی وجود داشت که توسط کولنر مطرح شد و مشکل دیگر بزرگ‌تری را هم وودین خیلی زود بعد از خواندن پیش‌نویس آسپرو و شنیدلر تشخیص داد. وودین می‌گوید: «این یک پیچش غیرمنتظره در این داستان است.»

هنگامی‌که او (*) را مطرح کرد، انواعی قوی‌تر از آن که (*)+ و (*)++ نامیده می‌شوند را نیز مطرح کرد که برای تمام مجموعه‌های توانی اعداد حقیقی به کار می‌روند. مشخص است که در مدل‌های مختلف جهان ریاضیاتی (اگر در حالت کلی این‌طور نباشد)، (*)+ با ماکزیمم مارتین در تناقض است. در اثباتی جدید که او در ماه می با ریاضیدانان مطرح کرد، او نشان می‌دهد (*)++ با (*)+ هم‌ارز است که نشان می‌دهد (*)++ در مدل‌های مختلف با ماکزیمم مارتین در تناقض است.

(*)+ و (*)++ بسیار بیشتر از (*) درخشیدند؛ زیرا آن‌ها به ریاضیدانان اجازه می‌دادند گزاره‌های به شکل «مجموعه‌هایی از اعداد حقیقی وجود دارد که...» بسازند و بنابراین ویژگی‌های هر مجموعه‌ای از اعداد حقیقی را آنالیز و توصیف کنند.

به این دلیل که به نظر می‌رسد ماکزیمم مارتین و (*)+ و (*)++ در تناقضند، به نظر می‌رسد وجود گزاره‌های مربوط به مجموعه‌های اعداد حقیقی در چهارچوب ماکزیمم مارتین ممکن نیست. برای وودین این یک عهدشکنی محسوب می‌شود.

بقیه بازیکنان اصلی در حال هضم اثبات وودین هستند؛ اما تعداد کمی از آن‌ها تأکید دارند که این یک فرضیه است. حتی خود وودین اعتراف می‌کند که یک کشف سوپرازکننده می‌تواند تصویر (و همچنین نظر) او را همانطور که پیش‌تر اتفاق افتاده بود، تغییر دهد.

افراد بسیاری منتظر نتیجه تلاش وودین برای اثبات حدس «Ul-timate L» هستند؛ یعنی وجود یک تعمیم همه‌جانبه از مدل جهان مجموعه‌های گودل. اگر این حدس وجود داشته باشد (وودین دلیل خوبی برای این‌که فکر کنیم چنین است، دارد و هم اکنون در حال تلاش برای تمام کردن اثبات ۴۰۰ صفحه‌ای خود است.) برای او روشن خواهد شد که آن «اصل موضوع رویایی» که باید به ZFC اضافه شود، همین حدس او می‌باشد. و در اینجا کانتور درست می‌گوید: پیوستار دارای کاردینالیتی $\aleph_1$ است. اگر اثبات کار کند، این حدس حتی اگر انتخابی برای گسترش ZFC نباشد، حداقل رقیبی نیرومند برای ماکزیمم مارتین خواهد بود.

از زمانی‌که گودل و کوهن استقلال فرضیه پیوستار از ZFC را ثابت کردند، ریاضیات نامتناهی تبدیل به یک داستان ماجراجویانه شده است که نظریه‌پردازان مجموعه می‌توانند تعداد اعداد حقیقی را هر مقداری ($\aleph_{35}$ یا $\aleph_{1000}$) گسترش دهند و نتایج آن را کشف کنند. اما با نتایج اشنایدر و اسپرو که به $\aleph_2$ اشاره می‌کند و وودین که در تلاش است ثابت کند اندازه پیوستار $\aleph_1$ است، دوگانگی آشکاری ایجاد شده‌است که اخیرا به نظر می‌رسد ممکن است برنده‌ای داشته باشد. بیشتر نظریه‌پردازان نظریه مجموعه‌ها چیزی را بیشتر از خروج از چندگانگی ریاضیاتی و یکی‌شدن پشت تصویر بهشت کانتور، تصوری که به اندازه کافی زیباست تا درست تلقی شود، دوست ندارند.

کندی بر طبق آنچه که خودش درست می‌داند، فکر می‌کند که ما به عنوان نسل بشر ممکن است که به زودی به دوران پیش از سقوط خود به روی زمین (یعنی دوران زندگی آدم و حوا) برگردیم. او همچنان می‌گوید: «هیلبرت در سخنرانی خود گفت که منزلت انسان بر عهده‌ی ماست که بتوانیم هرچیزی در ریاضیات را به صورت بله یا خیر معین کنیم. این، مسئله‌ی نجات دادن بشریت بود از این که آیا ریاضیات همان چیزی است که همیشه فکر می‌کردیم؟ ابزاری برای اثبات حقیقت و نه فقط این حقیقت و آن حقیقت؛ نه بر اساس احتمالات. قابلیت پیوستار این است، تمام.»

Computer Science

علوم کامپیوتر

رمزنگاری

Cryptography

سید سبحان مقیمی

مقدمه

از زمان پیدایش زبان تلاش‌ها برای مخفی نگه داشتن پیام در جریان بوده است. در ابتدا تلاش بر آن بود که این پیام‌ها به دست شخص دیگری نرسد. ساده‌ترین مثال آن استفاده از صندوقچهٔ دارای قفل می‌باشد. اما در کنار روش‌های فیزیکی، انسان‌ها به دنبال روشی بودند که اگر پیام توسط شخص سوم کشف شد، نتواند به محتوای آن پی ببرد. این‌گونه بود که رمزنگاری به میان آمد.

رمزنگاری (کریپتوگرافی[1]) به چه معناست؟

کریپتوگرافی از دو کلمهٔ یونانی kryptos، به معنای پنهان و graphein، به معنای نوشتن نشأت می‌گیرد. رمزنگاری شیوه‌ای برای ویرایش اطلاعات به شکلی است که تنها اشخاص مجاز بتوانند آن‌ها را درک کنند. به بیان دیگر، رمزنگاری، داده‌های قابل خواندن را به گونه‌ای تغییر می‌دهد تا از لحاظ ظاهری به صورت تصادفی و غیرقابل فهم به نظر برسند. فرد برای انجام این کار نیاز به استفاده از کلید رمزنگاری دارد. این کلید شامل مجموعه‌ای از مقادیر ریاضی بوده که فقط فرستنده و گیرنده مفهوم پیام رمزگذاری شده را می‌فهمند.

فرآیند رمزنگاری به طور کلی شامل دو مرحله می‌باشد:

رمزگذاری[2]: فرآیند کدگذاری متن و تبدیل آن به ساختار غیرقابل تشخیص برای خواننده می‌باشد.

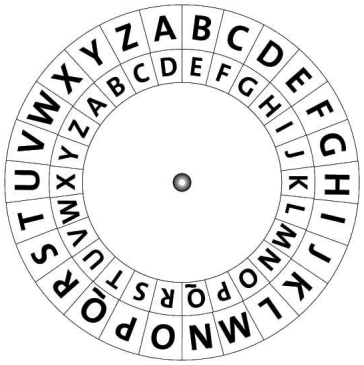
رمزگشایی[3]: عمل معکوس کدگذاری است که طی آن، پیام‌های ناخوانا به شکل اصلی خود تبدیل می‌شوند.

رمزنگاری متقارن[4]

رمزنگاری متقارن فرآیندی از رمزنگاری است که در آن رمزگشایی فقط عمل معکوس رمزگذاری می‌باشد. برای مثال، اگر پیام نوشته شده در فارسی با جایگزینی هر حرف به ۳ حرف بعد خود رمزگذاری شود (الف با ت، ب با ث و...)، برای رمزگشایی کافی است هر حرف را به ۳ جایگاه قبل خود ببریم. بنابراین در سیستم رمزنگاری متقارن هم فرستنده و هم گیرنده از یک کلید (رمز) مشابه استفاده می‌کنند. از این رمزها به عنوان cipher یاد می‌شود.

به طور کلی در رمزنگاری متقارن دو نوع رمز وجود دارد: رمز جایگزینی[5] و رمز جابه‌جایی[6]. رمز جایگزینی متن اولیه را با نمادهای دیگر جایگزین می‌کند تا پیام رمزگذاری شده را تولید کند؛ مانند مثالی که در ابتدا زده شد. اما در رمز جایگزینی ما جایگشتی از متن را به عنوان متن رمزگذاری شده تولید می‌کنیم، بدون آن که حرفی حذف یا تولید کنیم.

یکی از مهم‌ترین و ابتدایی‌ترین نمونه از رمز جایگزینی رمز سزار[7] می‌باشد. این کلید همان مثالی است که در ابتدا زده شد! کافی است هر حرف را ۳ واحد به بعد از خود منتقل کنیم! (شکل-۱)



شکل ۱: نمودار جایگزینی در رمز سزار

نقطه ضعف این رمز حدود ۸۰۰ سال بعد توسط ابویوسف الکندی، فیلسوف و ریاضی‌دان عرب، منتشر شد. او رمز سزار را بر اساس یکی از مهم‌ترین ویژگی‌های زبان، یعنی فراوانی حروف شکست!

هنگامی که از زبان فارسی یا هر زبان دیگر برای انتقال پیام استفاده می‌کنیم، به طور ناخودآگاه سرنخی را برای شنودکننده به‌جای می‌گذاریم. وقتی فرد الف و ب بارها و بارها از کد سزار برای انتقال پیام استفاده می‌کنند یا یک متن نسبتاً طولانی را ردوبدل می‌کنند، سرنخ شروع به ظاهر شدن می‌کند. تحلیل فراوانی حروف به‌کاررفته

[1] Cryptography
[2] Encryption
[3] Decryption
[4] Symmetric Cryptography
[5] Substitution Cipher
[6] Transposition Cipher
[7] Caesar's Cipher

در متن رمزنگاری‌شده و مقایسه آن با فراوانی حروف در زبان فارسی میزان جابه‌جایی کلمات در کد سزار را برملا می‌کند. برای مثال اگر حرف «ت» در پیام رمزنگاری‌شده بیشترین فراوانی را دارد، از آنجا که بیشترین فراوانی در زبان فارسی به حرف «الف» تعلق دارد، با احتمال زیاد میزان جابه‌جایی ۳ است. به این روش تحلیل فراوانی[8] در شکستن رمز گفته می‌شود.

فرآیندهای تصادفی در رمزنگاری

برای سال‌ها پرسش این بود که چگونه می‌توان به طور کامل سرنخ‌ها را در پیام‌های رمزنگاری‌شده از بین برد. پاسخ در فرآیندهای تصادفی است. فرض کنید فرد الف برای تولید میزان جابه‌جایی حروف، یک تاس ۳۲ وجهی می‌اندازد و میزان جابه‌جایی هر حرف را بر اساس آن تعیین می‌کند. مهم است که این کار را به تعداد حروف پیام انجام دهد. به این ترتیب یک کلید رمز بلند که کاملاً تصادفی تولید شده، ایجاد می‌گردد. در این حالت شنودکننده پیام دچار مشکل می‌شود؛ چراکه روش رمزنگاری دو ویژگی مهم دارد. اول، چون برای هر حرف میزان جابه‌جایی به شکل تصادفی (با توزیع احتمال یکسان) به‌دست‌آمده، هیچ الگوی تکرارشونده‌ای در متن رمزگذاری شده یافت نخواهد شد. دوم، وقتی فرد ج پیام‌های منتقل شده را تحلیل فراوانی می‌کند با یک نمودار فراوانی کاملاً یکنواخت مواجه می‌شود. به این ترتیب برای فرد ج ناممکن می‌شود که آن را با توزیع فراوانی واقعی حروف مقایسه کند. چرا که فقط برای یک کلمه‌ی شش‌حرفی حدود یک میلیارد حالت مختلف با احتمال یکسان وجود دارد!

اما تولید عدد تصادفی برای انسان با استفاده از ماشین ساده نیست. ماشین‌ها قابل پیش‌بینی هستند و الگوهای تکرارشونده در رفتار آنان وجود دارد. همین عدم قابلیت تولید اعداد کاملاً تصادفی موجب شد که دستگاه رمزنگاری انیگما[9] در جنگ جهانی دوم شکسته شود. انیگما یک دستگاه الکترومکانیکی بود که آلمانی‌ها در جنگ جهانی دوم از آن برای مبادله پیام بین واحدهای نظامی خود استفاده می‌کردند. انیگما در ظاهر مانند یک دستگاه تایپ معمولی بود (شکل۲). برای تبادل پیام، ارسال‌کننده و گیرنده هردو باید تنظیمات یکسانی از دستگاه را می‌داشتند. انیگما با این هدف طراحی شده بود که حتی اگر شنودکننده به آن دسترسی پیدا کند و از ساختار داخلی آن مطلع شود، نتواند کد پیام‌هایی را که با آن رمزگذاری می‌شود بشکند (نمونه‌ای از محرمانگی ایده‌آل).

[8] Frequency Analysis
[9] Enigma Machine

شکل ۲: دستگاه رمزنگاری انیگما

انیگما دارای چرخ‌های گرداننده‌ای بود که موقعیت آن‌ها نسبت به هم کلید رمز را مشخص می‌کرد. این رمزها هر ۲۴ ساعت تغییر می‌یافتند. بر اساس کلید رمز، انیگما حروف ورودی را به حروف دیگری تبدیل و پیام را رمزگذاری می‌کرد. گرچه آلمانی‌ها تمامی سعی خود را کرده بودند تا حروف به شکل تصادفی جابه‌جا شود، خطاهایی کوچک در طراحی موجب شد تا این فرآیند آن‌چنان هم که آنان تصور می‌کردند، تصادفی نباشد. آلن تورینگ[10]، دانشمند انگلیسی و از بنیان‌گذاران علم محاسبهٔ نوین و هوش مصنوعی، نقش مهمی در شکستن کدهای انیگما داشت. فعالیت‌های تورینگ در این پروژه کمک‌های زیادی به توسعه کامپیوترهای امروزی و حوزه‌ی هوش‌مصنوعی کرد.

رمزهای DES و AES

هورست فیستل گویا اولین فرد ارائه دهندهٔ ایدهٔ استفاده از کلیدی است که ورودی را به دو بخش مساوی تقسیم کرده و وارد چند مرحله می‌کند که در آن تابع‌ها و رمزها اعمال می‌شوند و همچنین نیمهٔ چپ و راست جابه‌جا می‌شوند. این روش پایه‌ای برای بسیاری از رمزگذاری‌های متقارن از جمله استاندارد رمزگذاری داده[11] (DES) می‌باشد.

DES (شکل۳) اولین محصول رمزنگاری تجاری در تاریخ بوده است. در حدود قرن ۱۹، نیاز شرکت‌ها برای برقراری ارتباط خصوصی موجب شد تا دولت ایالات متحده با شرکت IBM همکاری داشته باشد تا اولین سیستم رمزنگاری کامل را به بازار عرضه کند. در سال ۱۹۷۶ دفتر ملی استاندارد آمریکا DES را به عنوان استاندارد پردازش اطلاعات فدرال اعلام کرد. ورودی رمزنگار در آن یک رشته ۶۴ بیتی

[10] Alan Turing
[11] Data Encryption Standard

است.

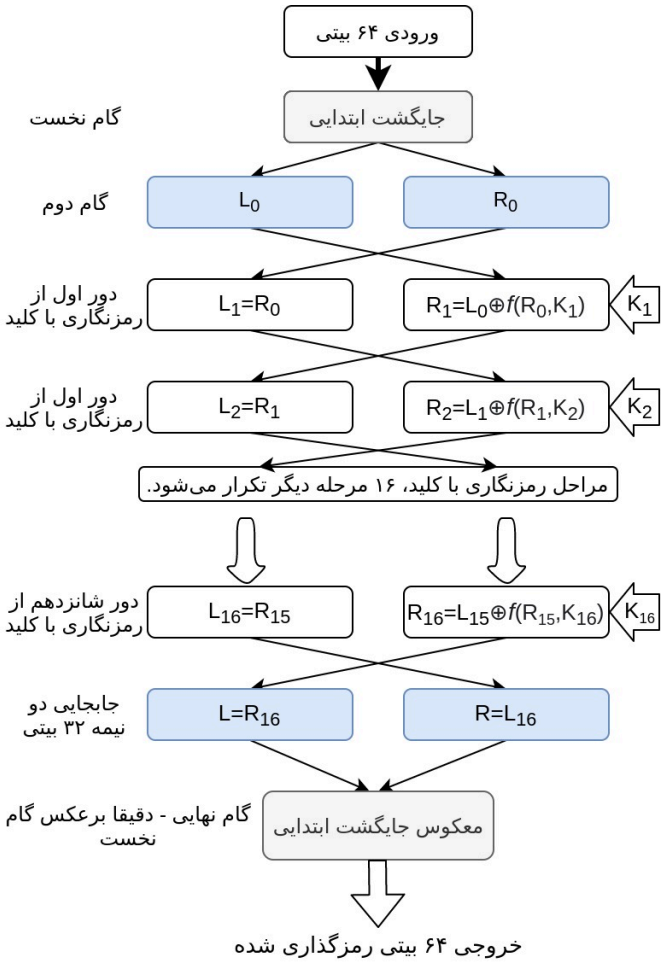
- اولین عملی که بر روی رشته ورودی انجام می‌شود، جابه‌جا کردن محل بیت‌های رشته ۶۴ بیتی است. به این عمل جایگشت مقدماتی گفته می‌شود که کلید رمز هیچ دخالت و تأثیری در این جابه‌جایی ندارد.
- در گام دوم ورودی ۶۴ بیتی به دو نیمه ۳۲ بیتی چپ و راست تبدیل خواهد شد. (این مرحله بیشتر به یک قرارداد شبیه است تا یک عمل مؤثر.)
- در گام سوم فرآیند رمزنگاری مبتنی بر کلید آغاز می‌شود و تا شانزده دور ادامه می‌یابد. الگوریتم رمزنگاری در تمام دورها یکسان است. این شانزده دور به شانزده کلید ۴۸ بیتی متفاوت نیاز دارند که همگی آن‌ها به روشی غیرخطی و نسبتاً پیچیده از کلید ۵۶ بیتی اصلی استخراج می‌شوند.
- پس از دور شانزدهم، جای نیمهٔ ۳۲ بیتی چپ و راست عوض می‌شود. سپس عکس عمل جایگشتی که در ابتدا انجام شده بود صورت می‌گیرد تا بیت‌ها سر جای اصلی خود برگردند.

در سال ۱۹۹۷ تصمیم گرفته شد که پارامترهای DES برای ارائهٔ امنیت مورد نیاز بسیار کم است. درنتیجه، نیازی جهانی برای یک سیستم رمزنگاری جدید به‌وجود آمد. پس از تلاش‌های جهانی، در سال ۲۰۰۲ استانداردی جدید به عنوان استاندارد رمزگذاری پیشرفته[12] (AES) توسط موسسهٔ استانداردها و تکنولوژی[13] انتخاب شد. AES مانند DES شامل ساختار فیستل نمی‌شود. این روش از جایگزینی و جایگشت‌ها در طی چند مرحله صورت می‌گیرد.

رمزنگاری کلید عمومی (نامتقارن)[14]

یکی از مشکلات بزرگ و عامل عدم پیشرفت رمزنگاری در صنعت، مبادلهٔ کلیدها (رمزها) بود. در صورتی که دولت روش‌های مختلفی برای مدیریت کلیدها ارائه داده بود، صنعت محتاط بوده و زیر بار این روش‌های مبادله‌ی کلید نمی‌رفت. در حدود دههٔ ۱۹۶۰ از این به عنوان «مشکل کنترل کلید»[15] یاد شده‌است و یک دهه بعد پاسخ آن یافت شد. این پاسخ مقدمه‌ای بر رمزنگاری کلید عمومی بوده است.

[12] Advanced Encryption Standard
[13] National Institute of Standards and Technology (NIST)
[14] Public Key (Asymmertric) Cryptography
[15] Key Management Problem



شکل ۳: الگوریتم رمزنگاری DES

در سال ۱۹۷۶، ویتفیلد دیفی[16] و مارتین هلمن[17] در مقاله‌ای، الگوریتمی ارائه کردند که مبنای رویکرد رمزنگاری نامتقارن است. رمزنگاری نامتقارن به طرفین اجازه می‌دهد که یک کلید عمومی را با یکدیگر به اشتراک بگذارند تا پیام تنها توسط کسی که کلید خصوصی[18] معادل آن کلید عمومی را دارد رمزگشایی شود.

تبادل کلید دیفی-هلمن

نحوهٔ عملکرد این الگوریتم، به همراه مثالی برای درک بهتر، در مراحل زیر آمده است:

مرحلهٔ اول: ابتدا فرد اول و دوم بر روی مقدار g و p توافق می‌کنند، به طوری که g ریشه اول به پیمانه p است. برای مثال فرض می‌کنیم $p = 13$ و $g = 5$. این همان کلید عمومی است.

مرحلهٔ دوم: فرد اول عددی تصادفی a (برای مثال ۱۱) را انتخاب

[16] Whitfield Diffie
[17] Martin Hellman
[18] Private Key

می‌کند. این همان کلید خصوصی فرد اول است. سپس معادلهٔ زیر را حل کرده و x یا همان پاسخ را به طور عمومی به فرد دوم می‌فرستد.

$$g^a \overset{p}{\equiv} x \quad , \quad 5^{11} \overset{13}{\equiv} 8$$

مرحلهٔ سوم: مانند مرحله دوم، این‌بار فرد دوم عدد تصادفی b (برای مثال ۱۷) را که همان کلید خصوصی وی می‌باشد انتخاب می‌کند. وی نیز معادلهٔ زیر را حل کرده و y یا همان پاسخ معادله را به صورت عمومی به فرد اول می‌فرستد.

$$g^b \overset{p}{\equiv} y \quad , \quad 5^{17} \overset{13}{\equiv} 5$$

مرحلهٔ چهارم: اکنون فرد اول پاسخ معادلهٔ نفر دوم ($y = 5$) را به صورت عمومی از فرد دوم گرفته و آن را به توان کلید خصوصی خود می‌رساند و حاصل را در پیمانه $p = 13$ محاسبه می‌کند تا به کلید پنهان مشترک برسند.

$$y^a \overset{p}{\equiv} (g^b)^a \overset{p}{\equiv} g^{ab} \quad , \quad 5^{11} \overset{13}{\equiv} 8$$

مرحلهٔ پنجم: به طور مشابه فرد دوم پاسخ معادلهٔ نفر اول ($x = 8$) را به صورت عمومی از فرد اول گرفته و آن را به توان کلید خصوصی خود می‌رساند و حاصل را در پیمانه $p = 13$ محاسبه می‌کند تا به کلید پنهان مشترک برسند.

$$x^b \overset{p}{\equiv} (g^a)^b \overset{p}{\equiv} g^{ab} \quad , \quad 8^{17} \overset{13}{\equiv} 8$$

بنابراین هر دو فرد به یک کلید مشترک، یعنی عدد ۸ رسیدند، بدون آن‌که فرد سوم بتواند به آن پی ببرد. این الگوریتم مسئلهٔ تبادل کلید را به طور کامل حل کرده، اما هنوز کسی طرح کاملی برای آن نیافته بود.

الگوریتم‌های تکمیل کنندهٔ کلید دیفی-هلمن

در سال ۱۹۷۸، اولین طرح رمزنگاری به کمک تبادل کلید دیفی-هلمن توسط سه فرد به نام‌های Rivest، Shamir و Adelman منتشر شد که به نام RSA شناخته می‌شود. سیستم امنیتی RSA براساس سختی تجزیه اعداد بزرگ بوده است. درسال ۱۹۹۷ یافت شد که دولت انگلیس نیز طرحی مشابه را در اوایل دههٔ ۱۹۷۰ ابداع کرده بود.

چند سال پس از طرح RSA الگوریتم رمزنگاری الجمل[19] به عنوان رساله دکتری طاهر الجمل ارائه شد. امنیت در این الگوریتم با RSA تقریباً متفاوت است. در الجمل هدف مشخص کردن توان در معادله‌ای به شکل $a = b^x$ می‌باشد که در آن a و b مشخص هستند. روش‌های دیگری از رمزنگاری نامتقارن وجود دارند که این دو مهم‌ترین آن‌ها بودند.

[19] ElGamal

امضای دیجیتال

ترکیب الگوریتم دیفی-هلمن و یکی از الگوریتم‌های نام‌بردهٔ فوق، کار را برای شنودکننده بسیار دشوار می‌کند تا بتواند به کلید رمز مشترک دست پیدا کند. ولی ازآن‌جایی که دو طرف تابه‌حال هیچ ارتباطی با یک‌دیگر نداشتند، این امکان وجود دارد که شنودکننده در ارتباط با فرد اول وانمود کند که فرد دوم است و کلید عمومی فرد دوم را در اختیار دارد. پس فرد اول باید قبل از اجرای این الگوریتم، اطمینان حاصل کند که کلید عمومی واقعاً متعلق به فرد دوم است.

یک روش برای ایجاد اطمینان این است که از پروتکل‌های مبتنی بر ایجاد اعتماد از طریق شخص سوم[20] استفاده شود. در این حالت یک شخص سوم که مورد اعتماد همه است اعتبارنامه‌هایی را برای افراد صادر و کلیدهای عمومی آنان را نیز امضا می‌کند. بنابراین به‌جای این که فرد اول بخواهد مستقیماً به فرد دوم اعتماد کند، اعتبارنامه او را معیار قرار می‌دهد. وب‌سایت‌هایی که با پروتکل HTTPS[21] کار می‌کنند، این‌گونه هستند.

فرض کنید شما می‌خواهید به درگاه یک وب‌سایت به نشانی www.site.ir متصل شوید. وقتی به درگاه سایت وصل می‌شوید، یک کلید عمومی از طریق نرم‌افزار مرورگر خود دریافت می‌کنید. این کلید عمومی برای ساخت یک کلید رمز مشترک با سرور درگاه سایت استفاده می‌شود تا بعداً پیام‌ها از طریق این کلید رمز مشترک به شکل متقارن رمزگذاری شوند. وقتی شما نام کاربری و گذرواژه را در فرم درگاه بانگ وارد می‌کنید، این اطلاعات بر اساس کلید رمز مشترک رمزگذاری می‌شوند. تنها سرور سایت که کلید خصوصی متناظر را دارد می‌تواند این اطلاعات را رمزگشایی کند.

این در حالتی است که شما مطمئن باشید آدرس بالا متعلق به همان وب‌سایت است. اگر شنودکننده یک وب‌سایت جعلی درست کرده باشد، یک کلید عمومی جعلی برای شما ارسال می‌کند که کلید خصوصی معادل آن را نیز دارد. به این ترتیب با واردکردن اطلاعات در وب‌سایت به اطلاعات شما دست پیدا می‌کند. اینجا جایی است که نقش شخص سوم پررنگ می‌شود. شخص سوم سازمان معتبری است که کلید عمومی و اطلاعات مالکان همه وب‌سایت‌ها را دارد. همه نیز این سازمان را می‌شناسند و کلید عمومی آن سازمان را در اختیار دارند.

وقتی وب‌سایت موردنظر از پروتکل HTTPS استفاده می‌کند، مرورگر کلید عمومی آن وب‌سایت را که توسط سازمان معتبر امضای دیجیتالی شده دریافت می‌کند. مرورگر کلید عمومی سازمان معتبر را نیز

[20] Trusted Third Party
[21] Hypertext Transfer Protocol Secure

دارد به همین دلیل می‌تواند امضای دیجیتال او را تائید کند. درنتیجه مرورگر مطمئن است کلید عمومی به سایت واقعی تعلق دارد. در این حالت سایت جعلی تنها می‌تواند از کلید عمومی بانک واقعی و نه کلید عمومی دیگری استفاده کند. اما اگر سایت جعلی بخواهد از کلید عمومی واقعی برای رمزنگاری استفاده کند، باید کلید خصوصی متناظر با آن را داشته باشد که امکان‌پذیر نیست.

پس از امضای دیجیتال فناوری‌های جدیدتری مانند بلاکچین از رویکرد کاملاً متفاوتی برای ایجاد اعتماد استفاده می‌کنند، به طوری که نهاد واسطه را کاملاً حذف کرده‌اند!

توابع هش[22]

فرآیند رمزنگاری نامتقارن مشکلاتی دارد؛ کند است و حجم زیادی از اطلاعات (حداقل دوبرابر اندازه پیام اصلی) تولید می‌کند. نوعی پیشرفت در این زمینه استفاده از توابع هش یک طرفه در مسیر رمزنگاری نامتقارن می‌باشد. یک تابع هش زنجیره‌ای از اعداد و حروف را می‌گیرد و یک خروجی یکتا با طول ثابت می‌دهد. این خروجی عددی با طول ثابت، می‌تواند در سایر الگوریتم‌های رمزنگاری استفاده شود. یکی از توابع پرکاربرد درهم‌سازی، الگوریتم SHA-256 نام دارد که در رمزگذاری پیام‌ها در بیت‌کوین نیز استفاده می‌شود. به خروجی این توابع، هش گفته می‌شود. برای دریافت هش پیام دلخواه خود می‌توانید به لینک زیر مراجعه کنید.

https://tools.superdatascience.com/blockchain/hash

توابع هش دارای چند ویژگی مهم می‌باشند:

- خروجی آن برای یک زنجیره ورودی مشخص هیچ‌گاه تغییر نمی‌کند. اگر یک پیام را بارها و بارها به تابع درهم‌سازی بدهید، همواره خروجی یکسانی دریافت می‌کنید.
- محاسبه مسیر معکوس بسیار دشوار است. اگر خروجی یک تابع درهم‌سازی را داشته باشید، تقریباً ناممکن است تا ورودی متناظر را پیدا کنید.
- کوچک‌ترین تغییری در زنجیره ورودی، خروجی را به طور کلی تغییر می‌دهد. این ویژگی موجب می‌شود حدس زدن خروجی یک زنجیره ورودی از روی زنجیره ورودی مشابه امکان‌پذیر نباشد.

[22] Hash Functions

نام SHA از عبارت Secure Hash Algorithm گرفته شده است. عدد ۲۵۶ بیانگر این است که خروجی این تابع همواره ۲۵۶ بیت از حافظه اشغال می‌کند. علت این است که خروجی تابع SHA-۲۵۶ عددی ۶۴ رقمی در مبنای ۱۶ است و چون هر رقم ۴ بیت را اشغال می‌کند در مجموع خروجی تابع ۲۵۶ بیت از حافظه را به خود اختصاص می‌دهد. با روش جستجو نمی‌توان به راحتی SHA-۲۵۶ را شکست. گرچه نسل‌های قدیمی‌تر توابع درهم‌سازی شکسته شده‌اند، تا امروز گزارشی از شکسته شدن SHA-۲۵۶ منتشر نشده است. ضمن این‌که نسل‌های جدید توابع درهم‌سازی نیز درحال توسعه یا به‌کارگیری هستند.

رمزنگاری، فراتر از پنهان‌سازی داده‌ها

با ورود به عصر مدرن و سادگی استفاده از کامپیوترها، رمزنگاری به ضرورتی برای همگان تبدیل شده است و کاربردهای آن نیز در کنار پنهان‌سازی داده‌ها توسعه یافت. برخی از آن‌ها شامل شناسایی فرستنده‌ها، احراز هویت فرستنده و گیرنده علاوه‌بر پیام‌ها و مهم‌تر از همه، امضای دیجیتال هستند. در سال‌های اخیر نیز با ورود رمزارزها به دنیای تکنولوژی، تحولی عظیمی در آن رخ داده‌است.

روش‌های نفوذ به الگوریتم‌های رمزنگاری، همگام با توسعه‌ی این الگوریتم‌ها، تکامل یافته‌اند. به همین جهت، نمی‌توان هیچ الگوریتمی را برای همیشه امن دانست! از طرفی روش‌های رمزنگاری کنونی با بیشتر شدن احتمال استفاده و عرضه‌ی کامپیوترهای کوانتومی، بیش از هر زمانی در معرض خطر قرار دارند. بنابراین این احتمال وجود دارد که با حضور کامپیوترهای کوانتومی در آینده، الگوریتم‌های رمزنگاری کوانتومی هم تکامل پیدا کنند.

مراجع

[1] Batten, Lynn Margaret. *Public key cryptography: applications and attacks*, volume 16. John Wiley & Sons, 2013.

[2] Diffie, Whitfield and Hellman, Martin. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[3] Mollin, Richard A. *An introduction to cryptography*. Chapman and Hall/CRC, 2006.

شبکه‌های عصبی گرافی: از ایده تا کاربرد

Graph Neural Networks: From Idea to Application

امیر اسکندری

رشد سریع تکنولوژی هوش مصنوعی به ویژه یادگیری ماشین و یادگیری عمیق، این امید را ایجاد کرده‌است که بتوان از این تکنولوژی برای حل مسائل پیچیده‌ای که انسان‌ها قادر به‌حل آن نیستند، استفاده نمود. اما الگوریتم‌های موجود برای طیف خاصی از مسائل قابل استفاده هستند که دارای داده‌ها با ساختار ویژه‌ای هستند؛ مانند دنباله‌های زمانی، تصاویر و... . اما همه‌ی مسائل پیچیده‌ای که انسان‌ها برای حل آن‌ها توسط هوش مصنوعی هیجان‌زده هستند، از این نوع نیستند و باید الگوریتم‌های هوشمندانه‌تری که بر روی داده‌های بدون ساختار ویژه هم عملکرد خوبی داشته باشند، ارائه شود.

گراف‌ها

گراف‌ها ساختارهایی هستند که از دو مجموعه تشکیل شده‌اند؛ مجموعه‌ی رأس‌ها و یال‌ها. گرافی از یال‌ها و رأس‌ها می‌تواند تقریباً هر چیزی در جهان واقعی را نمایش دهد. گره‌ها می‌توانند هر سوژه یا موجودیتی مثل مردم، شرکت‌ها، کامپیوترها یا هر موجودیت دیگری مانند پیکسل‌های یک تصویر باشند. یال‌ها می‌توانند روابط دوستانه‌ی بین مردم، دادوستد بین مردم در سیستم اقتصادی یا هر ویژگی و تجربه‌ی مشترکی بین رأس‌ها باشند. همه‌ی موجودهای جهان پیرامون عضوی از نوعی شبکه هستند؛ به بیان دیگر هر موجودی دارای شباهت‌ها و وابستگی‌هایی با دیگر موجودها است. مثل مردم که به انواع شبکه‌ها، مثل شبکه‌ی خانواده، شبکه‌های دوستان و شبکه‌ی همکاران تعلق دارند. شبکه‌هایی از مردم وجود دارد که در آن افراد می‌توانند سرمایه، دیدگاه‌های سیاسی یا سایر چیزها را به اشتراک بگذارند. خارج از علوم اجتماعی و مسائل مربوط به تعاملات مردم با هم، انواعی از مسائل پیچیده در حوزه‌های مختلف وجود دارد که ساختارهای غیر مشخصی دارند که می‌توان آن‌ها را با گراف توصیف کرد. مثل ساختارهای مولکول‌های شیمیایی، شبکه‌های اجتماعی، شبکه‌ی استناد علمی و... . ریاضیات سنتی گراف و ابزار تحلیلی آن برای توصیف چنین شبکه‌های پیچیده‌ای چندان مناسب نیستند.

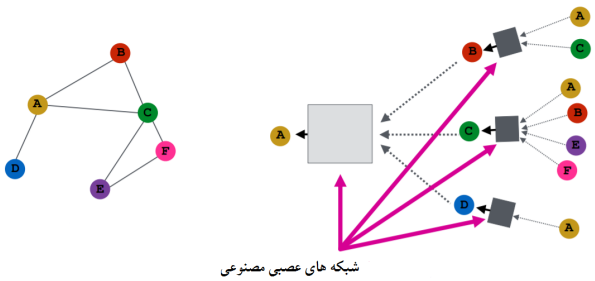
شبکه‌های عصبی گراف

در بخش قبل بیان شد که گراف‌ها توانایی توصیف ساختارهای داده‌ای مختلفی را دارند؛ پس نتیجه می‌شود ارائه‌ی الگوریتم یا راه‌حلی برای داده‌های گرافی توسط متخصصان هوش مصنوعی می‌تواند مسئله‌ی عدم توانایی الگوریتم‌های هوش مصنوعی برای داده‌های بدون ساختار را برطرف کند. در واقع هدف اصلی در این شبکه‌های عصبی آن است که یک نمایش برداری یکتا برای رأس‌ها یا حتی یال‌ها بر اساس ساختار گراف به دست آید. از نمایش برداری رأس‌ها می‌توان برای مسائل مختلف مانند دسته‌بندی رأس‌ها، پیش‌بینی یال‌های غایب یا حتی دسته‌بندی ساختار کلی گراف استفاده نمود. طرز کار و هسته‌ی اصلی این شبکه‌ها بر اساس مکانیزم انتقال پیام بین رأس‌ها است. این مکانیزم می‌تواند بر اساس ساختار گراف، یعنی اتصالات بین رأس‌ها، برای هر رأس در گراف یک شبکه‌ی عصبی یکتا را تعریف کند[۱].

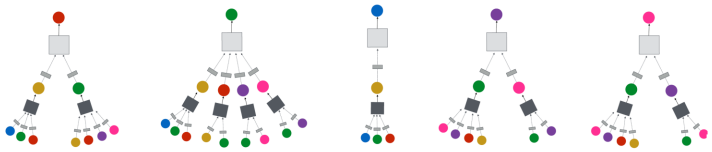
$$\begin{aligned}&\text{بیان برداری جدید رأس مشخص} = \\ &\mathbb{W}\left(\text{بیان برداری قبلی رأس مشخص}\right) \\ &+ \mathbb{B}\left(\text{پیام‌های رسیده از رأس‌های همسایه}\right)\end{aligned}$$

این مکانیزم به بیانی ساده در بالا آورده شده است؛ پیام‌ها همان بیان برداری همسایه‌های هر رأس به‌خصوص هستند که با استراتژی‌های مختلف (جمع، میانگین و بیشترین) گردآوری می‌شوند و با بیان برداری حاضر رأس به‌خصوص، بیان جدید را برای آن به‌وجود می‌آورند. پارامترهای $\mathbb{W}$و $\mathbb{B}$ پارامترهای یادگیرنده‌ی شبکه‌ی عصبی هستند. با همین مکانیزم ساده برای هر رأس به‌خصوص در گراف می‌توان شبکه‌ی عصبی یکتایی را به دست آورد؛ چراکه هر رأس جایگاه به‌خصوصی در گراف دارد.

در شکل ۱ برای رأس به‌خصوص A یک شبکه‌ی یکتا به دست آمده است. این شبکه پیام‌های همسایه‌های درجه ۱ و ۲ را جمع‌آوری و بیان‌برداری یکتایی برای رأس A ارائه می‌کند. مربع‌های اشاره شده در این شکل، شبکه‌های عصبی مصنوعی هستند. این مربع‌ها طرز جمع‌آوری پیام برای ارائه‌ی بیان‌برداری بهتر برای هر رأس بخصوص را



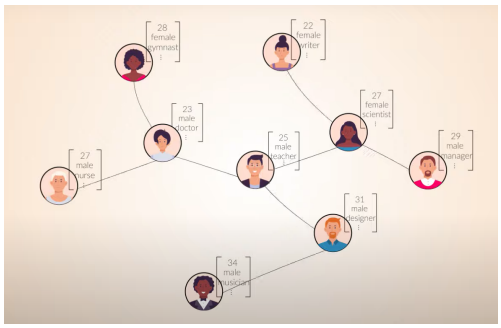
شکل ۱: یک گراف و طرز تعریف شبکه‌های عصبی برای رأس A



شکل ۲: شبکه‌ی عصبی برای ۵ رأس دیگر گراف

در طول آموزش مدل، یاد می‌گیرند. در شکل ۲ نیز شبکه‌های عصبی برای دیگر رأس‌ها نشان داده شده است؛ از این شکل مشخص است که برای هر هر رأس گراف، یک شبکه‌ی عصبی یکتا ارائه شده است.

برای رسیدن به شهودی در مورد این نوع شبکه‌ها مثالی را در ادامه بیان می‌کنیم. یک شبکه از دوستان را به عنوان گرافی در نظر بگیرید. هر کدام از افراد به عنوان رئوس گراف و یال‌های گراف روابط دوستانه را مشخص می‌کنند (شکل ۳).



شکل ۳: شبکه‌ای از دوستان

در این شبکه هر کدام از افراد دارای روابطی دوستانه با دیگران هستند، از طرفی هر فرد دارای ویژگی‌هایی منحصربه‌فرد است که با برداری نمایش داده می‌شود. المان‌های غیر عددی این بردار را می‌توان با استفاده از کدگذاری مناسب به صورت عددی تبدیل کرد که امکان انجام محاسبات فراهم باشد.

$$\text{علی} = \begin{bmatrix} ۲۳ \\ \text{لیسانس} \\ \text{معلم} \\ \vdots \end{bmatrix} \quad \text{محدثه} = \begin{bmatrix} ۳۰ \\ \text{دکتری} \\ \text{پزشک} \\ \vdots \end{bmatrix} \quad \text{و} \quad \dots$$

سناریویی را در نظر بگیرید که انتخاباتی پیش رو است و دو نامزد انتخاباتی آبی و قرمز در انتخابات حضور دارند. تعدادی از این دوستان تصمیم خود را برای نامزد مورد نظر تعیین کرده‌اند و تعدادی هم خیر. هدف آن است که رأی همه‌ی افراد پیش‌بینی شود. در این صورت با یک مسئله دسته‌بندی نیمه‌نظارتی[1] مواجه هستیم. باید بیان برداری‌ای برای هر فرد به دست آوریم یا به بیان دیگر بردار ویژگی هر فرد را به فضایی ببریم که افرادی که رای مشابه‌ای دارند، با افرازی خطی و غیرخطی دسته‌بندی شوند.

شکل ۴: نمایی از شبکه‌ی عصبی چند لایه

در شکل ۴ طرز کار شبکه برای همچین کاربردی نمایش داده شده است. این شبکه بردار ویژگی هر فرد را به یک فضای دوبعدی برده است که افراد همنظر به یکدیگر نزدیک‌تر هستند. این یک مدل ساده شده است. تعداد افراد شبکه و فضای نهایی معمولاً ابعاد بالاتری دارد. با پیش‌انتشار[2] ورودی، در لایه‌ی آخر نمایش برداری نهایی به دست می‌آید. با محاسبه‌ی خطا و پس‌انتشار[3] آن به ورودی، پارامترهای شبکه یاد گرفته می‌شود.

کاربرد شبکه‌های عصبی گراف

به تعداد محدودی از کاربرد شبکه‌های عصبی گرافی در زیر اشاره می‌کنیم. ارائه همه‌ی کابردهای این شبکه‌های عصبی در حجم محدود این مقاله امکان پذیر نیست[۲].

گراف دانش[4]

گراف دانش مجموعه‌ای از توصیف‌های به هم پیوسته یا مرتبط از موجودات، اشیا و رویدادهای دنیای واقعی یا مفاهیم انتزاعی را نشان می‌دهد. این گراف‌ها مبنای بسیاری از روش‌های تحلیلی در حوزه‌های مختلف کسب‌وکار، پزشکی و... هستند. شبکه‌های عصبی

[1] Semi-Supervised Classification
[2] Forward Propagate
[3] Back Propagate
[4] Knowledge Graph

گرافی می‌توانند رأس‌ها یا یال‌های این نوع گراف‌ها را با بیان برداری نمایش دهند. از این بیان‌های برداری می‌توان برای کارها و تحلیل‌ها برای نتایج مطلوب استفاده کرد و حتی یال‌های جدیدی بین مفاهیم مختلف را پیش‌بینی نمود. به طور مثال یکی از مسائل بیچیده‌ی حوزه‌ی پزشکی، بررسی عوارض داروهایی که با یکدیگر استفاده می‌شوند یا عوارض ناشناخته‌ی داروها به صورت تک است. در ترکیب دوتایی از مصرف داروها، با توجه به تعداد موجود داروهای مورد تایید FDA، ۱۳ میلیارد ترکیب وجود دارد. از گرافی می‌توان استفاده کرد که رأس‌های آن داروها و عوارض‌ها، یال‌های بین داروها بیانگر شباهت ساختاری آن‌ها و یال‌های بین داروها و عوارض بیانگر عوارض آن داروها باشد. در این‌صورت می‌توان عوارض کشف نشده برای داروهای مختلف را از طریق پیش‌بینی یال‌ها به دست آورد.

شیمی

مولکول‌ها در حد اندازه‌ی نانو یک ساختار گرافی دارند؛ در این گراف یون‌ها یا اتم‌ها رأس‌های گراف و پیوندهای مرتبط بین آن‌ها با یال‌ها مدل می‌شوند. اتم‌ها، یون‌ها و پیوندهای مختلف را می‌توان با استفاده از نمایش برداری اولیه تعبیه نمود، سپس با ارائه به شبکه‌ی عصبی گرافی به نمایش برداری مطلوب نزدیک شد. شبکه‌های عصبی گرافی را می‌توان در هر دو سناریو به کار برد؛ یادگیری درباره‌ی ساختارهای مولکولی موجود، تشخصی سمی یا غیر سمی بودن مولکول‌ها و همچنین کشف ساختارهای شیمیایی جدید برای کشف دارویی جدید. این امر تأثیر قابل توجهی در طراحی داروها به کمک رایانه داشته است.

اثر انگشت مولکولی یک مشخصه برای شناخت و تفکیک مولکول‌ها از یکدیگر است. در یادگیری ماشین این اثر انگشت مولکولی به صورت برداری نمایش داده می‌شود. مدل‌های یادگیری ماشین با یادگیری از مولکول‌های نمونه که از اثر انگشت با طول ثابت به عنوان ورودی استفاده می کنند، خواص یک ساختار جدید را پیش‌بینی می‌کنند. شبکه‌های عصبی گرافی می‌توانند جایگزین وسایل سنتی شوند که رمزگذاری و اثرانگشت ثابتی از مولکول را ارائه می‌دهند تا امکان ایجاد اثر انگشت مولکولی متمایز متناسب با نوع کاری که مدل قرار است انجام دهد را فراهم کند.

پردازش زبان طبیعی

یک کاربرد کلاسیک شبکه‌های عصبی گرافی در پردازش زبان طبیعی، طبقه‌بندی و دسته‌بندی متون است. این نوع شبکه‌ها از روابط متقابل اسناد یا کلمات برای استنباط برچسب اسناد استفاده می‌کنند. شبکه‌های پیچشی گرافی[5] برای حل این مسئله استفاده می‌شود. در این روش متن به گراف کلمات تبدیل می‌شود. نتایج نشان داده‌است که استفاده از گراف کلمات برای دسته‌بندی متون نسبت به روش‌های دنباله‌ای عملکرد بهتری را ارائه می‌کند. حوزه‌های مختلف پردازش زبان طبیعی مانند ترجمه‌ی ماشینی و... نیز از شبکه‌های عصبی تأثیر پذیرفته‌اند.

فیزیک

مدل‌سازی سیستم‌های فیزیکی در دنیای واقعی یکی از اساسی‌ترین جنبه‌های هوش در انسان است. با مدل‌سازی اشیا به عنوان گره و روابط و تأثیر آن‌ها به عنوان یال، می‌توان درمورد اشیا و سیستم‌های فیزیکی و نیز، ارتباط بین آن‌ها، استدلال‌هایی را برمبنای هوش مصنوعی انجام داد. شبکه‌های تعاملی بین اجزا را می‌توان در مورد تعامل اجسام در یک سیستم فیزیکی پیچیده استدلال کرد.

نتیجه‌گیری

طی چند سال اخیر، شبکه‌های عصبی گراف به ابزاری قدرتمند و کاربردی برای هر مشکلی که توانایی مدل شدن با گراف‌ها را دارد، تبدیل شده‌اند. هدف از این مقاله نیز، یک بیان واضح از ایده اصلی این نوع شبکه‌ها و تفاوتشان با بقیه شبکه‌های عصبی بوده است. انتظار می‌رود خواننده با مطالعه‌ی این مقاله به یک تعبیر درست از این نوع شبکه‌ها و درک قدرتمندی آن‌ها برسد.

مراجع

[1] Wu, Zonghan, Pan, Shirui, Chen, Fengwen, Long, Guodong, Zhang, Chengqi, and Philip, S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[2] Zhou, Jie, Cui, Ganqu, Hu, Shengding, Zhang, Zhengyan, Yang, Cheng, Liu, Zhiyuan, Wang, Lifeng, Li, Changcheng, and Sun, Maosong.

[5] Graph Convolutional Networks

Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

الگوریتمی جدید برای سریع‌تر حل کردن معادلات!

نویسنده: کوین هارتنت[1]
مترجم: امیرحسین غزنوی

مقدمه

اغلب اوقات دبیران ریاضی مدارس به این معروف هستند که به دانش‌آموزان خود توصیه می‌کنند که جواب مسئله را صرفاً حدس نزنند؛ بلکه برای پاسخ خود اثبات مشخصی داشت باشند. در واقع داشتن یک روش درست برای حدس زدن، گاهی بهترین روش برای حل یک معادله‌ی خطی است که یکی از پایه‌های اساسی محاسبات در ریاضی است.

در نتیجه اثبات، اولین روش را برای بالا بردن سرعت حل برخی از مسائل، که قبلاً زمان زیادی مصرف می‌کرده است، فراهم می‌کند.

مارک گیزبرشت[2] از دانشگاه واترلو[3] می‌گوید: «این یکی از اساسی‌ترین مسائل در محاسبات است. حال ما برای آنکه می‌توانیم سریع‌تر کار کنیم، اثبات داریم.»

روش اولیه، توسط ریچارد پنگ[4] و سانتوش ومپالا[5] از موسسه فناوری گرجستان[6]، در ماه ژولای به صورت برخط[7] گزارش شد و در ماه ژانویه در نشست سالانه‌ی ACM-SIAM برای الگوریتم‌های گسسته[8] جایزه‌ی بهترین مقاله را برنده شد.

دستگاه‌های خطی شامل دو یا تعداد بیشتری معادله با متغیرهای مختلف می‌باشند که در آن‌ها اشیای مختلف، به نحوی با یکدیگر ارتباط برقرار می‌کنند. دلیل اینکه اصطلاح «خطی» را برای آن‌ها به کار می‌بریم این است که تنها توان قابل قبول در آن‌ها یک است و نمودارهای جواب‌های آن‌ها درون صفحه قرار دارد.

یک مثال معروف از دستگاه‌های خطی (که بین دانش‌آموزان ریاضی شناخته شده است) درباره‌ی مزرعه‌ای از مرغ‌ها و خوک‌ها است. اگر شما تنها بدانید که در این مزرعه ۱۰ سر و ۳۰ پا موجود است، تعداد مرغ‌ها و خوک‌ها چند تا است؟ با توجه به آموخته‌ی دانشجویان جبر، یک روش دقیق برای حل این مسئله وجود دارد: نوشتن دو عبارت جبری و حل آن‌ها با هم.

ومپالا می‌گوید: «دستگاه‌های خطی، مانند اسبی بارکش برای محاسبات مدرن است.»

اثبات جدید با کنار زدن یکی از روش‌های اصلی‌ای که معمولاً به کار می‌رود، راهی سریع‌تر برای حل یک دسته‌ی بزرگ از دستگاه‌های خطی پیدا کرده است. این تکنیک، که به آن ضرب ماتریس می‌گویند، درگذشته سرعت خوبی برای حل دستگاه‌های خطی نداشت که البته هنوز در کارها به عنوان مکمل استفاده می‌شود. نویسندگان این را با یک رویکرد جدید که ذاتاً نوعی پیش‌گویی توسعه‌یافته است، اتصال دادند.

پنگ می‌گوید: «می‌توانید راه خود را برای جواب‌ها (حل سوال) حدس بزنید.» و هیچ معلمی از این کارتان عصبانی نخواهد شد!

مزرعه‌ی ریاضی!

برای اینکه شهود بهتری به دستگاه‌های خطی و اینکه چگونه آن‌ها را حل کنید، پیدا کنید به مسئله‌ی مزرعه برمی‌گردیم، ولی اکنون آن را کمی تغییر می‌دهیم. مرغ‌ها، کرگدن‌های تک‌شاخ، بزهای دوشاخ. شما یک محاسبه‌ی سریع انجام می‌دهید و متوجه می‌شوید که در مزرعه، ۱۲ سر، ۳۸ پا و ۱۰ شاخ وجود دارد. آیا می‌توانید بگویید از هرکدام از حیوانات چه تعداد در مزرعه وجود دارد؟

برای ادامه دادن کار، یک متغیر به هر یک از حیوانات نسبت می‌دهیم (c برای تعداد مرغ‌ها، r برای تعداد کرگدن‌ها و g برای تعداد بزها) و برای هر کدام از داده‌های مسئله یک عبارت جبری می‌نویسیم. ضرایب برای هر متغیر، نشان دهنده‌ی تعداد آن عضو برای حیوان مورد نظر است.

$$\begin{cases} c+r+g=12 & \text{سرها} \\ 2c+4r+4g=38 & \text{پاها} \\ 0c+1r+2g=10 & \text{شاخ‌ها} \end{cases}$$

حال شما سه معادله و سه مجهول دارید.

یک راه برای حل این دستگاه، کار کردن با یکی از معادله‌ها و به دست آوردن یکی از متغیرها برحسب دو متغیر دیگر است. برای مثال، از $0c+1r+2g=10$ می‌توان نتیجه گرفت: $r=10-2g$. این مقدار r را در دو معادله‌ی دیگر جایگزین می‌کنیم و ادامه دهیم تا موقعی

[1] Kevin Hartnett
[2] Mark Giesbrecht
[3] University of Waterloo
[4] Richard Peng
[5] Santosh Vempala
[6] Georgia Institute of Technology
[7] online
[8] ACM-SIAM Symposium on Discrete Algorithms

که بتوانید همه‌ی متغیرها را برحسب یک متغیر به دست آورید. سپس متغیر به دست آمده را در معادلات جایگزین کنید و مقدار بعدی را به دست آورید و همینطور برای به دست آوردن بقیه‌ی متغیرها ادامه دهید.

اما راه‌حل دیگر که پیچیده‌تر هم نیز هست، ساختن ماتریسی است که شامل ضرایب معادله است. از سه معادله‌ی داده شده ماتریس زیر به دست می‌آید:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 4 \\ 0 & 1 & 2 \end{bmatrix}$$

حال ما ماتریس دیگری برای مجهولات خودمان می‌سازیم:

$$\begin{bmatrix} c \\ r \\ g \end{bmatrix}$$

در آخر تعداد کل سرها، پاها و شاخ‌ها را در ماتریس دیگری قرار می‌دهیم:

$$\begin{bmatrix} 12 \\ 38 \\ 10 \end{bmatrix}$$

ما می‌توانیم این سه ماتریس را در یک دستگاه خطی با یک معادله ترکیب کنیم؛ بدین صورت که ماتریس ضرایب در ماتریس مجهولات ضرب شده و برابر با سومین ماتریس شده است. حال اینجا می‌توانیم به وسیله‌ی جبر خطی ماتریس مجهولات را به دست آوریم:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 4 \\ 0 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ g \end{bmatrix} = \begin{bmatrix} 12 \\ 38 \\ 10 \end{bmatrix}$$

چه از روش اول و چه از روش دوم، شما مجبور هستید تعداد یکسانی محاسبات برای پیدا کردن جواب انجام دهید. تعداد محاسبات برابر مکعب تعداد متغیرها در دستگاه (n^3) است. در این سؤال ما ۳ متغیر داریم. در نتیجه به ۲۷ مرحله محاسبه نیاز داریم. اگر ۴ حیوان و ۴ معادله داشتیم، ۶۴ مرحله محاسبه نیاز داشتیم.

در طی ۵۰ سال گذشته محققان روش‌هایی پیدا کرده‌اند که این روند را به طور مؤثرتر انجام دهند. اغلب این روش‌ها میانبرهایی (راه‌هایی مانند نادیده گرفتن یا ترکیب کردن چند معادله) بودند که باعث می‌شدند آن‌ها قادر باشند این دستگاه‌ها را در مراحل کم‌تری انجام دهند.

در سال ۱۹۶۹، فولکر استراسن[9] یک الگوریتم کشف کرد که روش ضرب ماتریس‌ها را در تنها $n^{2.81}$ عملیات انجام می‌داد. از آن‌موقع ریاضی‌دان‌ها و متخصصان علوم کامپیوتر، برای کم کردن تعداد عملیات‌ها تلاش کردند. آخرین پیشرفت انجام شده در این امر، در ماه اکتبر توسط ویرجینیا واسیلوسکا ویلیامز[10] از مؤسسهٔ تکنولوژی ماساچوست[11] و همچنین جاش المان[12]، یک محقق فوق دکتری در دانشگاه هاروارد[13]، انجام شد که در آن اثبات شد این کار در تعداد $n^{2.37286}$ عملیات نیز ممکن است. یک بهینه‌سازی در توان نسبت به رویکرد قبلی.

نتیجه این شد که که همه‌ی دستگاه‌های خطی که می‌خواهیم آن‌ها را حل کنیم، می‌توانند به یک سوال از ضرب ماتریس‌ها تبدیل شوند و امروزه این مسائل می‌توانند در $n^{2.37286}$ عملیات انجام شوند.

اما شواهد زیادی نشان می‌دهد که حل این معادلات می‌تواند حتی سریع‌تر از این انجام شود (به طور بالقوه در n^2 عملیات). ما از ضرب ماتریس‌ها استفاده می‌کنیم چون بهترین ابزار در دسترس است، اما این بدین معنی نیست که ابزار بهتری برای کشف وجود ندارد.

ومپالا می‌گوید: «دلیلی وجود ندارد که این مسئله برای حل دستگاه‌های خطی، به پیشرفت در ضرب ماتریس‌ها وابسته باشد.»

راه‌حل‌های حدسی

برای فهمیدن ابزار پیشرفته‌ی جدید، شما نیاز به یک روش مقرر دیگر برای حل دستگاه‌های خطی دارید. این یکی از شهودهای آن است؛ راهی که شاید وقتی از آن استفاده می‌کنید که ابتدا یک نگاه کلی به گله مرغ‌ها، کرگدن‌ها و بزها بیاندازید؛ تعداد هر کدام را حدس بزنید، اعدادتان را درون معادلات قرار دهید و ببینید چقدر با جواب فاصله دارید؛ سپس دوباره حدس می‌زنید.

این «رویکرد تکراری» همان چیزی است که مهندسان و دانشمندان اغلب استفاده می‌کنند. این کار برای بسیاری از مسائل کاربردی جواب می‌دهد؛ زیرا متخصصان کورکورانه حدس نمی‌زنند که این‌کار باعث کاهش یافتن تعداد حدس‌ها قبل از پیدا کردن جواب می‌شود.

پنگ می‌گوید: «برای مسائل علمی محاسباتی دنیای واقعی، انسان شهود خوبی از اینکه جواب نهایی چیست دارد.»

این روش‌ها در مثال‌های خاص مفید هستند که شهود به بهتر شدن آن کمک می‌کند. همچنین در حل دستگاه‌های خطی‌ای که شامل تعداد

[9] Volker Strassen
[10] Virginia Vassilevska Williams
[11] Massachusetts Institute of Technology
[12] Josh Alman
[13] Harvard University

زیادی از متغیرها با ضریب صفر هستند، مفیداند.

این ویژگی در مثال مزرعه وجود دارد (و مفید است). یکی از ویژگی‌هایی که کار را آسان می‌کند شاخ است! چرا؟ چون مرغ‌ها شاخ ندارد! که این باعث صفر شدن ضریب می‌شود. در نتیجه حیوانات مسئله را از سه حیوان به دو حیوان کاهش می‌دهد و هنگامی که اطلاعاتی درباره‌ی متغیرها از رابطه‌ی شاخ‌ها به دست آوردید، می‌توانید آن را در معادلات پاها و سرها استفاده کنید و حل مسئله را تسریع ببخشید.

در دستگاه‌های خطی پیچیده‌تر، این اتفاق که در آن همه صفات مربوط به همه متغیرها نیست، می‌تواند فراگیر باشد. ممکن است میلیون‌ها متغیر و میلیون‌ها معادله داشته باشید، اما هر معادله ممکن است فقط تعداد کمی از متغیرها را شامل شود. این نوع دستگاه‌های خطی، «پراکنده» نامیده می‌شوند که نشان‌دهنده‌ی این است که اکثر متغیرها در اکثر معادلات مقدار صفر می‌گیرند. این وضعیتی است که اغلب در دستگاه‌های خطی دنیای واقعی ظاهر می‌شود و همچنین یکی از شرایطی است که روش‌های حدسی می‌توانند ضرب ماتریسی را شکست دهند!

ویلیامز می‌گوید: «این‌کار فقط زمانی به درد می‌خورد که ماتریس شما به اندازه کافی پراکنده باشد.»

اما قبل از این اتفاق جدید، هیچ کس نتوانسته بود ثابت کند که روش‌های حدس و خطا برای همه‌ی دستگاه‌های خطی پراکنده، همیشه سریع‌تر از ضرب ماتریس هستند.

تصادفی هماهنگ شده!

تکنیک جدید پنگ و ومپالا از نسخه پیشرفته‌ای از استراتژی حدس زدن استفاده می‌کند. الگوریتم آن‌ها به جای یک حدس واحد، حدس‌های زیادی را به طور موازی انجام می‌دهد. این رویکرد، جستجو را سرعت می‌بخشد؛ همان‌طور که اگر افراد بیشتری را برای پیدا کردن یک جواهر در جنگل روانه کنید، زودتر آن را پیدا خواهید کرد!

گیزبرشت می‌گوید: «[حل] موازی، جایی است که جادو اتفاق می‌افتد.»

ممکن است بدیهی به نظر برسد که ارائه چندین حدس همزمان مفید است، اما عملی کردن این استراتژی چندان ساده نیست. اثربخشی الگوریتم جدید تا حد زیادی متکی است به هوشمند بودن نحوه ایجاد حدس‌های اولیه که روند حدس و خطا را ایجاد می‌کند و یافتن راه‌های هوشمندانه برای ترکیب نتایج حدس‌های هم‌زمان، در یک پاسخ نهایی.

به مسئله‌ی مزرعه بازمی‌گردیم؛ الگوریتم ممکن است سه حدس اولیه بزند که در آن هر حدس یک ماتریس ۳ در ۱ است که تعداد مرغ‌ها، کرگدن‌ها و بزها را مشخص می‌کند. این الگوریتم میزان فاصله و خطای هر حدس را بررسی می‌کند، سپس حدس های بیشتری می‌زند و با توجه به سرنخ‌های به دست آمده، حدس زدن را به صورت موازی پیش می‌برد.

کلید موفقیت نهایی الگوریتم این است که سه حدس اولیه را به طور تصادفی ایجاد می‌کند. این روش ممکن است مبنای خوبی برای حدس زدن به نظر نرسد، اما به عنوان یک روش همه منظوره مزایای خود را دارد. به ویژه هنگامی که با مسائل پیچیده سر و کار دارید. در واقع تصادفی بودن این اطمینان را به شما می‌دهد که به سمت خاصی از مسئله سوگیری ندارید و توجهی به اینکه جواب دقیقا در کجا قرار دارد نمی‌کنید.

پنگ می‌گوید: «من باید مطمئن شوم که همه‌ی حدس‌های من به اندازه‌ی کافی تصادفی هستند تا همه‌ی ترکیبات احتمالی را پوشش دهند. این روش بسیار وحشتناکی برای حدس زدن است که با بزرگ شدن مسئله، روش ترجیحی است.»

بسیاری از تکنیک‌های پیچیده در مقاله‌ی پنگ و ومپالا شامل اثبات این است که لایه‌های مختلف حدس‌های تصادفی باهم کار می‌کنند و در ارتباطند؛ از جمله هر حدس خاصی که درواقع پاسخ مسئله نیز هست.

ومپالا می‌گوید: «یک کار تصادفی هماهنگ‌شده در آن وجود دارد.»

این بدان معناست که حدس‌های تصادفی نه تنها مقدار خود حدس را شامل می‌شوند، بلکه تمام حدس‌های ممکن بین آنها را نیز پوشش می‌دهند. در مثال جنگل مانند این است که افرادی که در جنگل به دنبال جواهر می‌گردند، علاوه بر مسیری که روی آن راه می‌روند، فاصله‌ی بین خودشان هم مورد جست‌وجو قرار دهند.

ومپالا می‌گوید: «هر چیزی بین دو [حدس] نیز بررسی می‌شود.»

این ویژگی جستجو اطمینان می‌دهد که الگوریتم در جایی با پاسخ مسئله روبرو می‌شود. اما به خودی خود مشخص نمی‌کند که جواب واقعاً چیست! برای انجام این کار (فهمیدن دقیق پاسخ مسئله) پنگ و ومپالا باید چیز دیگری را ثابت کنند.

این الگوریتم حدس‌های تصادفی خود را درون ماتریس قرار می‌دهد و نتایج را بررسی می‌کند. در اینجا دوباره به ضرب ماتریس می‌رسیم. البته ضرب ماتریس در ابتدا مانعی بود که قصد کنار گذشتن آن و استفاده از ایده‌های جدید را داشتیم، اما در اینجا آن‌ها را مجدداً، صرفاً برای بررسی حدس‌ها استفاده می‌کنیم.

از آن‌جا که ورودی‌های ماتریس تصادفی هستند و هماهنگی‌ای بین آنها ایجاد می‌شود، درنهایت ماتریس تقارن خاصی پیدا می‌کند. این تقارن‌ها میانبرهای محاسباتی را در دسترس قرار می‌دهند. مانند هر شیء متقارنی که شما می‌توانید با دیدن تنها قسمتی از آن، کل آن را تصور کنید.

در نتیجه، الگوریتم پنگ و ومپالا می‌تواند پاسخ را درون ماتریس، سریع‌تر از ماتریسی به همان اندازه ولی بدون تقارن پیدا کند. متقارن بودن ماتریس مزیت دیگری نیز دارد؛ به آنها این اطمینان را می‌دهد که حدس‌ها آنقدر زیاد نمی‌شوند که کارایی الگوریتم را کاهش دهد.

پنگ می‌گوید: «ما باید کنترل کنیم که چه تعداد عدد در مراحل حدس زدن و هماهنگی بررسی می‌شوند.»

پنگ و ومپالا ثابت می‌کنند الگوریتم آن‌ها می‌تواند هر دستگاه خطی پراکنده را در $n^{2.332}$ عملیات حل کند؛ که این یک پیشرفت نسبت به تعداد عملیات‌های قبلی ($n^{2.37286}$) است. این قضیه تأثیری در حذف ضرب ماتریس‌ها به این زودی در برنامه‌های کاربردی نخواهد نداشت. اما این پیشرفت جزئی یک چیز را ثابت می‌کند؛ روش کاملاً بهتری برای حل دستگاه‌های خطی وجود دارد.

ومپالا می‌گوید: «از نظر فلسفی ما قبلاً نمی‌دانستیم که آیا می‌توانیم سریع‌تر از ضرب ماتریس پیش برویم یا خیر.»

حال این کار را انجام داده‌ایم!

Erlang

امیرحسین رجبی

در این نوشته قصد داریم زبان ارلنگ (Erlang) و مهم‌تر از آن سیستم (محیط) اجرای [۱] آن را معرفی کنیم. ارلنگ یک زبان برنامه‌نویسی فانکشنال است و ابتدا به بایت‌کد ماشین مجازی BEAM کامپایل شده و سپس توسط آن اجرا می‌شود. درباره تفاوت‌های مهم این ماشین مجازی با ماشین‌های مجازی مرسوم مانند JVM [۲] برای جاوا و CLR [۳] برای زبان های .NET. صحبت می‌کنیم.

ارلنگ در سال ۱۹۸۶ توسط جو آرمسترانگ و همکارانش [۴] به عنوان یک نرم‌افزار انحصاری در شرکت مخابراتی Ericsson توسعه پیدا کرده بود؛ اما در سال ۱۹۹۸ به عنوان نرم‌افزار آزاد منتشر شد. ارلنگ در ابتدا به هدف بهبود سوییچ‌های مخابراتی، درون شرکت اریکسون نوشته شد. مهم‌ترین ویژگی‌هایی که ارلنگ را از سایر زبان‌های فانکشنال متمایز می‌کنند عبارتند از:

۱. قابلیت استفاده در سیستم‌های توزیع‌شده یا Distributed

۲. سیستم‌های Fault-tolerant: یعنی سیستم‌هایی که در آنها خطا به دلایل گوناگون رخ می‌دهد و سیستم باید بتواند برای مدت طولانی (سال‌ها) بدون توقف اجرا شود؛ همچنین این خطاها باید ثبت شده تا امکان رفع آنها باشد.

۳. قابلیت Hot Swapping یا Hot Plugging: یعنی امکان تغییر کد سیستم در حال اجرا بدون توقف آن برای رفع باگ‌ها و یا افزودن ویژگی‌های جدید.

۴. سیستم‌هایی با نیاز High Availability: برای مثال سیستم‌هایی که Down time آنها حدود ۳ دقیقه در سال باشد.

اما همه این موارد مدیون ماشین مجازی BEAM است. در واقع تفاوت اصلی آن با CLR یا JVM این است که سیستم اجرای ارلنگ شبیه‌سازی یک سیستم‌عامل است، به‌طوری که هنگام اجرای آن، فرایندهایی [۵] ایجاد می‌شوند و با یکدیگر پیام ردوبدل می‌کنند.

در حالی که در یک نمونه از ماشین مجازی جاوا، فقط یک فرایند در حال اجراست. اما فرایندهایی که در BEAM اجرا می‌شوند با فرایندهایی که در سطح سیستم‌عامل هستند تفاوت دارند. به اصطلاح آنها را lightweight می‌خوانند. یعنی سربار ساخت و تعویض متن [۶] آنها بسیار کمتر از فرایندهای سیستم‌عامل است. در این حد تصور کنید که هنگام اجرای یک نمونه از BEAM روی یک کامپیوتر معمولی، [۷] امکان اجرای میلیون‌ها فرایند به طور هم‌زمان [۸] وجود دارد. این تعداد فرایند برای یک سیستم‌عامل قابل تصور نیست. در بعضی کتاب‌های سیستم‌عامل نخ‌ها به دسته‌های Kernel Threads، User Threads و Green Threads تقسیم می‌شوند. در توصیف نخ‌های سبز گفته می‌شود که تعدادی از آنها به یک نخ سطح هسته نگاشته می‌شوند. از طرفی نخ‌های سبز توسط سیستم‌عامل زمان‌بندی [۹] نمی‌شوند بلکه این کار توسط کتابخانه سمت کاربر [۱۰] انجام می‌شود. با این توصیفات، می‌توان گفت فرایندهای موجود در BEAM بسیار به نوع سوم یعنی Green Threads شبیه هستند.

نکته مهم بعدی ارتباط میان‌فرایندی [۱۱] در BEAM است. تنها راه ارتباط بین فرایندهای ارلنگ تبادل پیام [۱۲] است. اساساً در دیدگاه آرمسترانگ حافظه اشتراکی معنا ندارد. او مانند Alan Kay [۱۳] [۱۴] معتقد بود که ایده مهم برنامه‌نویسی شیگرا، تبادل پیام است؛ نه برنامه‌نویسی Class Oriented که امروزه مرسوم به Object Oriented است. خواننده علاقه‌مند، به مقاله‌ها و بحث‌های حول این موضوع در تئوری زبان‌های برنامه‌نویسی ارجاع داده می‌شود. به‌طور خلاصه آرمسترانگ سیستم اجرای ارلنگ را این‌گونه توصیف می‌کند:

۱. همه چیز فرایند است و به‌طور هم‌زمان در حال اجرا هستند.

۲. فرایندها اطلاعاتی را با یکدیگر به اشتراک نمی‌گذارند.

۳. فرایندها با یکدیگر صحبت می‌کنند. (پیام تبادل می‌کنند.)

[1] Erlang Runtime System (ERTS)
[2] Java Virtual Machine
[3] Common Language Runtime
[4] Joe Armstrong, Robert Virding, and Mike Williams
[5] Process
[6] Context Switch
[۷] لپ‌تاپ‌های من و شما
[۸] Concurrent نه Parallel
[9] CPU Scheduling
[10] User Library
[11] Inter Process Communication (IPC)
[12] Message Passing
[۱۳] از بنیان‌گذاران تفکر شیءگرا
[14] Alan Kay once said, “I’m sorry that I long ago coined the term “objects” for this topic because it gets many people to focus on the lesser idea. The big idea is “messaging”.”

۴. فرایندها ممکن است با خطا روبه‌رو شوند و یک فرایند دیگر باید این خطا را مدیریت کند. (یک فرایند که اشراف بیشتری به موضوع دارد.)

اینها اصول Actor Model هستند. برای فهم این اصول آرمسترانگ، نگاهمان به سیستم اجرای ارلنگ باید تغییر کند. باید آن را شبیه یک جامعه ببینیم. به هر عضو این جامعه یک Actor می‌گوییم. پس تعبیر این اصول این‌گونه خواهد بود:

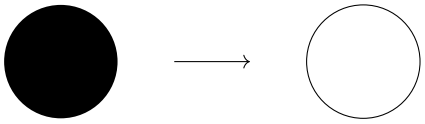
۱. اعضای جامعه کارهای خود را به‌صورت موازی و همزمان پیش می‌برند.

۲. افراد جامعه با هم صحبت می‌کنند. مغزهای خود را به اشتراک نمی‌گذارند!

۳. اگر یک فرد جامعه سکته قلبی بکند، خودش ابداً نمی‌تواند خودش را احیا کند بلکه فرد دیگری باید به او کمک کند. این همان اصل چهارم است. در همین مورد آرمسترانگ می‌گوید که خطاها نباید به‌صورت محلی مدیریت شوند، بلکه یک فرایند دیگر باید این وظیفه را به عهده بگیرد. [۱۵]

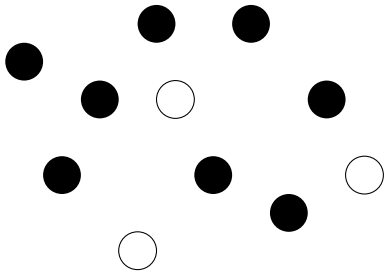
پس با این توصیف، فرایندها (Actors) کاملاً از هم ایزوله هستند؛ به‌طوری که زباله‌روب [۱۶] ارلنگ Per Process اجرا می‌شود (مانند فرایندهای یک سیستم‌عامل). اما این پیام‌رسانی به چه صورت است؟ طبیعی‌ترین روش ممکن! اگر من بخواهم برای شما پیامی ارسال کنم، باید ایمیل یا نامه ارسال کنم و شما آن را در صندوق پستی خود دریافت می‌کنید. به همین صورت، در ارلنگ هر فرایند یک Mailbox دارد و پیام‌های فرایندهای دیگر در آن به ترتیب زمانی قرار می‌گیرند.

با این توصیفات به یکی از اسلایدهای معروف آرمسترانگ در سخنرانی‌هایش اشاره می‌کنیم. در زبان‌های برنامه نویسی معمولی، سیستم‌ها به صورت تک‌فرایندی هستند:



دایره توپر یک فرایند در حال اجرا را نشان می‌دهد. دایره توخالی یک فرایند متوقف‌شده [۱۷] را نشان می‌دهد.

به همین دلیل، تمام تلاش برنامه این است که دچار خطا نشود و باید همه خطاها کنترل شوند، و اگر نه، کل برنامه متوقف می‌شود. اما در یک نود [۱۸] ارلنگ فرایندها به‌صورت زیر هستند:



اینجا ذکر دو نمونه کاربرد ارلنگ ضروری است. اول اینکه پیام‌رسان WhatsApp با این زبان نوشته شده است. نویسنده از این پیام‌رسان خوشش نمی‌آید ولی چرا اشاره به این پیام‌رسان مهم است؟ پاسخ به تعداد کاربران آن در دنیا مربوط است. این پیام‌رسان از فوریه ۲۰۲۰ بیش از ۲ میلیارد کاربر در دنیا دارد و محبوب‌ترین پیام‌رسان دنیا در سال ۲۰۱۵ شناخته شده است. این در حالی است که Telegram در حدود ۵۰۰ میلیون کاربر دارد. دومین کاربرد در شبکه‌های LTE است. یعنی به احتمال ۵۰ درصد، دیتایی که به کمک اینترنت موبایلتان دریافت کرده‌اید توسط تعدادی فرایند ارلنگ به دستتان رسیده است. یعنی نیمی از ترافیک شبکه موبایل دنیا!

در ادامه به پیاده‌سازی یک Server-Client در ارلنگ می‌پردازیم؛ اما قبل از آن کمی درباره انواع داده ابتدایی، سینتکس و نحوه اجرای BEAM صحبت می‌کنیم. به دلیل ذات متفاوت ماشین مجازی ارلنگ با دیگر ماشین‌های مجازی ذکر شده، طبیعتاً آن را باید به‌صورت مستقیم اجرا کرد. پس از نصب ارلنگ [۱۹] در سیستم‌عامل خود، می‌توانید با دستور `erl` در شل [۲۰] سیستم‌عامل، یک نمونه ماشین مجازی را اجرا کنید. (ماشین مجازی BEAM به همراه دیگر ابزار مثل کامپایلر و دیباگر نصب می‌شود.) چیزی شبیه زیر را خواهید دید:

```
Erlang/OTP 23 [erts-11.1.7] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1]

Eshell V11.1.7  (abort with ^G)
1>
```

در این شل ارلنگ می‌توان دستورات مختلفی اجرا کرد؛ از جمله کامپایل کد، لود کردن ماژول‌ها، بررسی وضعیت ماشین و فرایندهای در

[15] Error handling is non-local.
[16] Garbage Collector
[17] Crashed Process
[18] node
[19] توضیح در انتها
[20] shell

حال اجرای آن و غیره. مرحله بعد نوشتن یک ماژول است. در زبان‌های فانکشنال، یک برنامه به چند ماژول تقسیم می‌شود که هر کدام شامل تعدادی تابع، ماکرو و رکورد است. یک فایل با نام factorial.erl و محتویات زیر در همان مسیری که ماشین مجازی را اجرا کردیم ایجاد می‌کنیم:

```
% factorial.erl
-module(factorial).
-export([f/1]).

f(0) -> 1; % If 0 then return 1, (first
  clause ends with semicolon)
f(N) -> N * f(N - 1). % Otherwise compute
   the answer recursively. (second clause
   ends the function definition with a
  dot)
```

و سپس در شل ارلنگ خواهیم داشت:

```
1> c(factorial).
{ok,factorial}
2> factorial:f(0).
1
3> factorial:f(5).
120
```

اولین دستور، فایل factorial.erl را کامپایل و در ماشین مجازی لود می‌کند. از این پس می‌توان به کمک نام ماژول، توابع export شده را فراخوانی کرد. دستورات بعدی، این فراخوانی که به‌صورت Module:Function(Arguments) هست را نشان می‌دهند. امضا f/1 [۲۱] نشان می‌دهد که تابعی با یک ورودی به نام f در ماژول وجود دارد که از خارج از ماژول قابل دسترسی است. همچنین پاسخ هر فراخوانی در شل معلوم است. در ارلنگ هر چیزی که با حرف بزرگ شروع شود یک متغیر است؛ یک متغیر یک‌بارمصرف. یعنی:

```
4> A = 1.
1
5> A = 2.
** exception error: no match of right
  hand side value 2
6> A = 1.
1
7> 1 = A.
1
8> 1 = B.
* 1: variable 'B' is unbound
9> a = 1.
** exception error: no match of right
  hand side value 1
10> a = A.
** exception error: no match of right
  hand side value 1
11> A = a.
** exception error: no match of right
  hand side value a
12> a = a.
a
```

وقتی مقدار ۱ به متغیر A نسبت داده می‌شود دیگر قابل تغییر نخواهد بود. اما اولین خطا در کوئری پنجم کمی عجیب است. no match دیگر چیست؟ مگر متغیرها باید با مقدار سمت راست تطابق داشته باشند؟ بله! مفهومی به نام Pattern Matching در ارلنگ نهفته است. یعنی مقادیر دو طرف تساوی باید یک چیز باشند. اگر سمت چپ یک متغیر جدید باشد، آن مقدار به آن متغیر نسبت داده می‌شود و اگر نه، در صورت عدم برابری طرفین، ارور نوشته شده رخ خواهد داد. موضوع عجیب بعدی کوئری نهم است. گفتیم متغیرها با حرف بزرگ شروع می‌شوند. پس آنها که با حرف کوچک شروع می‌شوند چه هستند؟ atom. اتم‌ها فقط با خودشان برابرند. در کوئری اول دو اتم دیدیم. نام ماژول فاکتوریل و ok که از پرکاربردترین اتم‌های ارلنگ است. این اتم عموماً مقدار برگشتی توابع است و به معنای انجام موفقیت‌آمیز دستور می‌باشد. یک نکته ریز: هر خط ارلنگ به یک نقطه ختم می‌شود. آنچه در کد ماژول فاکتوریل می‌بینیم ممکن است کمی عجیب باشد ولی این قانون آنجا نیز برقرار است. در واقع تعریف تابع در خط اول تمام نشده است؛ بلکه یک حالت (بخوانید Clause) از تابع تعریف شده است.

یکی از فواید Pattern Matching هنگام Unpack کردن چندتایی‌ها [۲۲] مشخص می‌شود. مشابه آن را ممکن است در زبان پایتون دیده باشید: x, y = (1, 2) که متغیر های x و y به ترتیب مقادیر ۱ و ۲ می‌گیرند. با دیدگاه Pattern Matching این به این معناست که سمت راست و چپ باید برابر باشند و در نتیجه، منطقی است که ۱ در x جای بگیرد. البته عبارت بالا در پایتون به‌صورت زیر در ارلنگ نوشته می‌شود:

[۲۱] به ۱ در این امضا arity گفته می‌شود.

[22] tuple

```
13> {X, Y} = {1, 2}.
{1,2}
14> X.
1
15> Y.
2
```

با این دانسته‌ها دوباره به برنامه فاکتوریل نگاه کنید. یک تابع با دو Clause تعریف شده‌است. اولی حالت پایه را تعیین می‌کند. یعنی اگر ۰ به تابع داده شود این خط اجرا شده و مقدار ۱ از تابع بازگردانده می‌شود. اگر هر مقدار دیگری به ورودی داده شود با N (ورودی Clause دوم) Pattern Match می‌شود و این Clause اجرا می‌شود. تکه کد زیر تعریف یک تابع دیگر را نشان می‌دهد:

```
g({X, Y}) ->
  Z = X + Y,
  T = X * Y,
  {Z, T}.
```

تابع بالا یک ورودی به فرم دوتایی {X, Y} را می‌گیرد و شامل سه خط [۲۳] است. در خط اول جمع ورودی‌ها را در متغیر Z می‌ریزد؛ سپس ضرب ورودی‌ها را در T ریخته و در نهایت یک دوتایی که مؤلفه اول Z و مؤلفه دوم T است را برمی‌گرداند. در ارلنگ مقدار آخرین statement به عنوان خروجی تابع در نظر گرفته می‌شود و کلیدواژه‌ای مانند return وجود ندارد. پس یک تابع به‌طور کلی به‌صورت زیر است:

```
FunctionName(ArgumentPattern1) ->
  Statement11,
  Statement12,
  ...
  Statement1X_1;
FunctionName(ArgumentPattern2) ->
  Statement21,
  Statement22,
  ...
  Statement2X_2;
...
FunctionName(ArgumentPatternN) ->
  StatementN1,
  StatementN2,
  ...
  StatementNX_N.
```

دقت کنید ArgumentPatternها باید تعداد مساوی آرگومان داشته باشند. تابع‌های هم‌نام ولی با تعداد ورودی‌های متفاوت (Arity متفاوت) به طور مجزا از هم نوشته می‌شوند. همچنین برای اجرای تابع با ورودی Arg از بالا به پایین، اولین تابع با ArgumentPattern که با Arg تطابق داشته باشد اجرا می‌شود. برای مطالعه بیشتر به منابع مراجعه کنید.

اکنون ابزار لازم برای کار با فرایندها را داریم. برای ایجاد یک فرایند در ارلنگ از بیف [۲۴] spawn استفاده می‌شود. کار این تابع ساخت یک فرایند و اجرای یک تابع در آن است. (مانند توابعی که نخ می‌سازند.) در نتیجه از سه ورودی این تابع، اولی اسم ماژولی که تابع مدنظر در آن قرار دارد می‌باشد که یک اتم است، دومی نام تابع مدنظر که باز هم یک اتم است و سومی ورودی‌های آن تابع به‌صورت یک لیست. کد ماژول server.erl را ببینید:

```
% server.erl
-module(server).
-export([start/0, loop/0]).

start() ->
  spawn(server, loop, []).

loop() ->
  receive
    {Pid, N} -> Pid ! N + 1
  end,
  server:loop().
```

همان‌طور که می‌بینید با فراخوانی تابع start یک فرایند که کد آن را در تابع loop مشاهده می‌کنید، ایجاد شده و شروع به اجرا می‌کند. در صورتی که این تابع با موفقیت اجرا شود، شماره منحصربه‌فرد فرایند [۲۵] در ماشین مجازی برگردانده می‌شود. به کمک دستور i() نام و Pid همه فرایندهای در حال اجرای ماشین مجازی را می‌توان مشاهده کرد:

```
16> c(server).
{ok,server}
17> ServerPid = server:start().
<0.116.0>
18> i().
```

[23] statement

[24] Built-in Function (BIF)

[25] Process Identifier (PID)

```
    ...
    PatternN -> StatementN1, StatementN2,
   ... StatementNX_N
end
```

برای ساختارهای پیشرفته‌تر یا استفاده از timeout به کتاب مرجع اول مراجعه شود.

اکنون کد سرور را این‌گونه تغییر می‌دهیم:

```
loop() ->
  receive
    {Pid, hi} -> Pid ! hi_there, server:loop();
    {Pid, N} -> Pid ! N + 1, server:loop();
    stop -> ok
  end.
```

و داریم:

```
21> c(server).
{ok,server}
22> ServerPid ! {self(), 1}.
{<0.82.0>,1}
23> receive X -> X end.
2
24> ServerPid ! {self(), hi}.
{<0.82.0>,hi}
25> flush().
Shell got hi_there
ok
26> ServerPid ! stop.
stop
27> process_info(ServerPid).
undefined
```

در بالا چندین اتفاق افتاد. در خط اول، ماژول دوباره کامپایل شد. دقت کنید ماشین مجازی خاموش نشده و فرایند شماره <0.82.0> همچنان در حال اجرا است. با ارسال پیام به فرایند، این موضوع نشان داده شده است. سپس یک پیام با توجه به پیاده‌سازی جدید به سرور ارسال می‌شود. در اینجا از تابع مخصوص شل به نام flush استفاده شده است. این تابع محتویات صندوق پستی شل را نشان داده و همه پیام‌ها را از Mailbox خارج می‌کند. (مقدار برگشتی فلاش است.) با ارسال پیام stop، سرور دیگر فراخوانی بازگشتی را

```
Pid        Initial Call   Heap Reds Msgs
 Registered Current Function Stack
...
<0.116.0> server:loop/0 233  6    0
          server:loop/0    2
...
```

گفتیم که هر فرایند در سیستم، یک صندوق پستی دارد. اما چگونه می‌توان به این صندوق پستی پیام ارسال کرد؟ به کمک عملگر !. در واقع با داشتن شماره Pid یک فرایند، می‌توان به صندوق پستی آن پیام ارسال کرد (پیام می‌تواند چندتایی، لیست، رکورد یا هر داده ساختار ارلنگ باشد.):

```
19> ServerPid ! {self(), 5}.
{<0.82.0>,5}
20> receive X -> X end.
6
```

یک بیف دیگر به اسم self شماره فرایند فعلی را می‌دهد. در کوئری ۱۹ به صندوق پستی فرایندی که پیشتر ساختیم پیام {self() , 5} را می‌فرستیم؛ Pid خودمان و پیام مدنظر. اما Pid خودمان یعنی چه؟ مگر این دستورات در شل اجرا نمی‌شوند؟ پاسخ ساده است. خود شل نیز یک فرایند در BEAM است و دارای یک Pid و یک صندوق پستی می‌باشد. این فرایند هیچ تفاوتی با سایر فرایندها ندارد. صرفاً با stdout و stdin در ارتباط است. البته ارسال Pid خودمان به یک فرایند دیگر اجباری نیست ولی چون می‌خواهیم سرور نیز جواب ما را بدهد نیاز است که Pid خودمان را برایش ارسال کنیم. سپس به کمک کلیدواژه receive می‌توان از صندوق پستی یک پیام دریافت کرد. اما کدام پیام‌ها را؟ آنهایی که با الگوهای معرفی‌شده در بدنه این دستور تطابق داشته باشند، به ترتیب ورودشان به صندوق پستی، دریافت می‌شوند. در کوئری ۲۰ تنها یک الگو معرفی شده است و آن هم با هر چیزی تطابق داشته باشد همان را برمی‌گرداند. با بررسی کد سرور، می‌بینیم که وقتی پیام به فرم {Pid, N} در صندوق پستی دریافت می‌شود؛ پیام N + 1 به فرایند با شماره Pid ارسال می‌شود و حلقه سرور ادامه پیدا می‌کند (با فراخوانی دوباره loop).

به طور کلی ساختار دریافت پیام از صندوق پستی این‌گونه است:

```
receive
  Pattern1 -> Statement11, Statement12,
  ... Statement1X_1;
  Pattern2 -> Statement21, Statement22,
  ... Statement2X_2;
```

انجام نداده و فرایند خاتمه می‌یابد. این موضوع با تابع شل به نام process_info قابل مشاهده است. در خلال این کار ویژگی Hot Swapping را مشاهده کردیم. کد فرایند در حال اجرا تغییر کرد و در پیام‌های بعدی که به سرور ارسال می‌شود، این رویه ادامه خواهد داشت.
یک نکته مهم در این زمینه وجود دارد: تلاش می‌شود فراخوانی‌های بازگشتی در زبان‌های فانکشنال به‌صورت Tail Recursive باشند. یعنی آخرین کاری که تابع انجام می‌دهد، فراخوانی بازگشتی باشد. چرا؟ چون این فراخوانی‌ها توسط کامپایلر بهینه‌سازی می‌شوند و حافظه استک مصرف نمی‌کنند. این پیاده‌سازی تابع فاکتوریل را با پیاده‌سازی‌ای که بالاتر اشاره کردیم مقایسه کنید:

```
-module(factorial_tail_recursive).
-export([f/1]).
f(N) -> f(N, 1).
f(0, X) -> X;
f(N, X) -> f(N - 1, X * N).
```

توجه کنید در صورتی که تابع loop یک سرور بصورت Tail Recursive نباشد و ۱۰۰۰ پیام به سرور ارسال شود، ۱۰۰۰ نود استک حافظه اشغال می‌شود. یعنی تا زمانی که فرایند خاتمه نیابد برای هر پیام یک نود استک وجود دارد.
بحث اینجا تمام می‌شود و مطالعه درباره اکوسیستم خطا در ارلنگ که در طراحی سیستم‌های Fault-tolerant نقشی پررنگ دارد را به خواننده علاقه‌مند واگذار می‌کنیم. همچنین چگونگی ارتباط نودهای ارلنگ در شبکه نیز از نقاط قوت ارلنگ است.

چگونه ارلنگ را نصب کنیم؟

دو راه وجود دارد. مراجعه به erlang.org، دانلود کد منبع و کامپایل آن. راه دوم، استفاده از باینری‌های آماده است که در مخازن شرکت Erlang Solutions قرار دارند. مراجعه شود به اینجا.

منابع

آمار و ارقام یادشده همگی از ویکی‌پدیای مطلب مربوطه استخراج شده‌اند. (مانند تعداد کاربران WhatsApp و Telegram)

۱. کتاب Erlang Programming نوشته Francesco Cesarini و Simon Thompson منتشر شده توسط O'Reilly
۲. کتاب Programming Erlang (Software for a concurrent world) نوشته جو آرمسترانگ
۳. سخنرانی جو آرمسترانگ با موضوع Systems that run forever self-heal and scale در سال ۲۰۱۳. لینک
۴. سخنرانی جو آرمسترانگ با موضوع How we program multicores. لینک
۵. سخنرانی جو آرمسترانگ با موضوع The Do's and Don'ts of Error Handling در کنفرانس‌های GOTO سال ۲۰۱۸. لینک

کمک‌خلبان گیت‌هاب، هوش مصنوعی شما در برنامه‌نویسی

GitHub Copilot, Your AI Pair Programmer

غزاله خرادپور

مقدمه

کمک‌خلبان، که گیت‌هاب آن را «هوش مصنوعی جفت-برنامه‌نویس شما[۱]» می‌خواند، نتیجهٔ همکاری با OpenAI، آزمایشگاه غیرانتفاعی سابقی که با هوش مصنوعی قدرتمند مولّد زبان شناخته می‌شود، است. در قلب آن یک شبکهٔ عصبی است که توسّط مقادیر زیادی دیتا آموزش داده شده است. با این حال، به جای متن، منبع اصلی اجزای کوپایلت کدها هستند؛ میلیون‌ها خط کد که توسّط ۶۵ میلیون کاربر گیت‌هاب، بزرگ‌ترین پلتفرم برای توسعه‌دهنده‌ها به منظور همکاری و به اشتراک‌گذاری کارهایشان، بارگذاری شده‌اند. هدف اصلی کوپایلت این است که با الگوهای کد آشنا شود.

گیت‌هاب کوپایلت چیست؟

گیت‌هاب کوپایلت یک جفت-برنامه‌نویس است که کمک می‌کند تا کدها سریع‌تر و با کار کم‌تری نوشته شوند. گیت‌هاب کوپایلت زمینه[۲] را از کد و کامنت‌ها می‌گیرد و خطوط منحصربه‌فرد و کل تابع را فوراً پیشنهاد می‌دهد. کوپایلت مجهّز به OpenAI Codex است؛ یک سیستم هوش مصنوعی جدید که توسط OpenAI ایجاد شده است. پیش‌نمایش فنّی گیت‌هاب کوپایلت به عنوان افزونهٔ Visual Studio Code در دسترس است.

کوپایلت کد-بریده‌ها را به طور اتوماتیک کامل می‌کند، خط‌های جدید کد را پیشنهاد می‌دهد و حتّی می‌تواند توابع را به صورت کامل بر اساس توضیحاتی که دریافت می‌کند، بنویسد.

- نظرات[۳] را به کد تبدیل می‌کند.
  این ابزار نظرات را به کد تبدیل می‌کند و کافیست کاربر در نظر خود منطق برنامه‌ای که می‌خواهد بنویسد را توضیح دهد و به کوپایلت اجازه دهد تا کمکش کند.
- کدهای تکراری را به طور اتوماتیک پر می‌کند.
  کوپایلت برای تولید کدهای تکراری بسیار خوب کار می‌کند و فقط کافیست کاربر مثال‌های کوچکی از کار خود را بنویسد تا بقیّهٔ کد را کوپایلت بنویسد.
- به راحتی تست می‌نویسد.
  تست نوشتن ستون فقرات هر مهندسی نرم‌افزاری قدرتمند است. کافیست کاربر یک بستهٔ[۴] مربوط به تست وارد کد کند تا کوپایلت تست‌هایی را که با دقّت خوبی مرتبط به کد پیاده‌سازی‌شده هستند را پیشنهاد دهد.

طبق بلاگ گیت‌هاب، این ابزار فقط یک الگوریتم مولّد زبان بر اساس ورودی کاربر نیست بلکه یک جفت-برنامه‌نویس مجازی است. این ابزار یاد می‌گیرد خود را به عادت‌های کد زدن کاربر تطبیق دهد، پایگاه‌کدهای در دسترس را تحلیل کند و پیشنهاداتی بر اساس میلیاردها خط کدهای عمومی‌ای که روی آن آموزش دیده است، ارائه دهد. این بریده‌پیشنهادات توسط دکمه‌ای به کد کاربر اضافه می‌شود. در این صورت، نیازی نیست که زمان زیادی صرف جست‌وجو در مستندات API یا در بریده‌کدهایی در سایت‌هایی هم‌چون StackOverFlow شود و فقط می‌تواند به ابزار ویرایش کد خود تمرکز کند.

چگونه کار می‌کند؟

در هستهٔ آن، گیت‌هاب کوپایلت از الگوریتم جدید مولّد زبان توسط OpenAI به نام Codex استفاده می‌کند. مدیر اجرایی OpenAI، آن را نسل GPT-3 ولی متمرکز روی تولید کد توصیف می‌کند. الگوریتم Codex با استفاده از چندین ترابایت کد عمومی موجود در گیت‌هاب و گزیده‌ای از جملات انگلیسی آموزش دیده است. این الگوریتم این قابلیّت را به کوپایلت می‌دهد که کدهای مبتنی بر زمینه با دقّتی باورنکردنی بنویسد.

هم‌اکنون، این ابزار یک افزونه برای Visual Studio Code است و موقعیّت برای پیش‌نمایش فنّی محدود است. به طور عمیق‌تر، این افزونه کد و کامنت‌های شما را به سرویس گیت‌هاب کوپایلت که از الگوریتم Codex برای ترکیب و ایجاد پیشنهاد استفاده می‌کند می‌فرستد.

---

[۱] programmer pair
[۲] context
[۳] comments
[۴] package

شکل ۱: خلاصه‌ای از طرز کار گیت‌هاب کوپایلت

این ابزار با هر زبان برنامه‌نویسی‌ای به طور مجازی صحبت می‌کند ولی با زبان Python، JavaScript، TypeScript، Ruby و Go بهتر کار می‌کند.

بر اساس انتخاب کاربرها، این ابزار می‌تواند تا ۱۰ جایگزین برای یک پیشنهاد تولید کند. این الگوریتم به طور مداوم با استفاده از ثبت کردن این که کدام پیشنهاد از سوی کاربر پذیرفته شده یا خیر، بهتر می‌شود.

چرا استفاده از آن مفید است؟

توسعه‌دهندگان در گیت‌هاب یک آزمایشی انجام داده‌اند که دقّت این ابزار را اندازه‌گیری می‌کند؛ روی مجموعه‌ای از توابع زبان Python تست شده است که همگرایی خوبی نسبت به مخازن متن-باز[۵] داشته است. تمامی بدنهٔ تابع حذف شده است و تنها مواردی که ارائه می‌شد، اسم تابع و docstring مربوط به آن بود. کوپایلت آن را در ۴۳ درصد از مواقع به درستی پر کرد و دقّت آن پس از ۱۰ بار تلاش به ۵۷ درصد افزایش یافت. از زمانی که ابزارها قابلیّت تولید کدهای قابل اجرایی که بتوان آن را در پروژه‌ها به کار برد، پیدا کرده‌اند، این تکنولوژی یک شاهکار قابل‌توجّهی به حساب می‌آید. این تکنولوژی به محض آن که در دسترس همگان قرار گیرد، به طور قابل‌ملاحظه‌ای به صنعت توسعه سرعت می‌بخشد.

برای گرفتن بیش‌ترین بهره از آن، پیشنهاد می‌شود تا کدها را به قطعات کوچک‌تری تقسیم کنید و اسامی توابع، متغیّرها و docstringهای معناداری به سیستم ارائه دهید. نمونه‌ای را که خود سایت گیت‌هاب برای زبان پایتون ارائه داده است را می‌توانید در شکل ۲ مشاهده کنید.

به عبارت دیگر، این تکنولوژی جفت-برنامه‌نویس شماست؛ شما را وادار به دنبال کردن بهترین شیوهٔ[۶] مهندسی نرم‌افزار می‌کند و در مقابل از کد شما برای بهتر کردن پیشنهاداتش آموزش می‌بیند.

[۵] open-source repositories
[۶] best practice

کیفیت کد چگونه است؟

با این که در نیمی از اوقات کوپایلت درست کار می‌کند امّا پدیدآورندگان می‌گویند که خروجی آن باید تحت نظر قرار بگیرد: «گیت‌هاب کوپایلت سعی می‌کند قصد شما را بفهمد و بهترین کدی که می‌تواند را تولید کند امّا ممکن است کدی که پیشنهاد می‌کند، همیشه کار نکند یا حتّی معنی ندهد.» کیفیّت پیشنهادهای کوپایلت هم‌چنین به کدهای موجود بستگی دارد. کوپایلت می‌تواند فقط از فایل جاری به عنوان زمینه استفاده کند، نمی‌تواند معنای کد خود را تست کند یا حتّی اجرا یا کامپایل کند. علاوه بر آن، سؤالات متداول[۷] می‌گوید مسئولیت استفاده از این ابزار با خود کاربران است چون این ابزار ممکن است نسخهٔ قدیمی یا منسوخ فریم‌ورک‌ها[۸] و کتاب‌خانه‌ها[۹] را پیشنهاد دهد. نگرانی‌ای نیز در مورد مجموعه داده‌های آموزشی کوپایلت وجود دارد که این کدها توسّط میلیون‌ها نفر نوشته شده‌اند.

سوال بدیهی این است که «آیا تا به حال کدی را از مجموعه داده‌های آموزشی تکرار کرده است؟» که جواب آن مثبت است. مشاهده شده است که حدود ۰/۱ درصد احتمال دارد که این ابزار کدی را از مجموعه داده‌های آموزشی پیشنهاد بدهد.

با این حال، مطالعات گسترده‌تر روی این مشکل نشان می‌دهند که این اتّفاق فقط زمانی می‌افتد که زمینه‌های کافی برای یادگیری از آن‌ها برای کوپایلت وجود نداشته باشد. مخصوصاً زمانی که فایل جاری کوتاه یا خالی باشد، این الگوریتم با احتمال بیش‌تری اشتباه می‌کند.

آیا دانشمندان داده باید نگران باشند؟

با در نظر گرفتن خوبی‌ها و بدی‌های این تکنولوژی، باید پرسید که آیا گیت‌هاب کوپایلت شغل توسعه‌دهندگان را در آینده تحت تأثیر قرار می‌دهد یا خیر؟

زمانی که GPT-3 منتشر شد، جواب این سؤال به طور آزمایشی و ضعیفی «بله» بود. با این حال، اکنون که کوپایلت بیرون آمده است و به صورت تجاری در دسترس است که با یکی از بزرگ‌ترین محیط توسعه یک‌پارچه[۱۰] ادغام شده است؛ باید در مورد جواب سؤالمان تجدید نظر کنیم.

پدیدآورندگان ادّعا می‌کنند که این ابزار باعث بیش‌تر شدن بهره‌وری و آزاد شدن توسعه‌دهندگان از انجام کارهای دستی می‌شود و به آن‌ها کمک می‌کند تا روی کارهای جالب‌تری تمرکز کنند. هم‌چنین ممکن است

[۷] FAQs
[۸] frameworks
[۹] libraries
[۱۰] IDEs

sentiment.ts write_sql.go parse_expenses.py addresses.rb

```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                         float(value),
                         currency))
    return expenses
```

Copilot

شکل ۲: نمونه‌ای ورودی و خروجی به زبان پایتون

سختی‌ها را برای افرادی که به تازگی می‌خواهند وارد صنعت نرم‌افزار شوند، کم کند. این باور وجود دارد که هوش مصنوعی هر چه قدر هم پیشرفت کند، به این زودی جایگزین انسان‌ها نخواهد شد. در واقع هوش مصنوعی در آینده در تمامی زمینه‌های برنامه‌نویسی بهتر می‌شود و می‌تواند هر مسئلهٔ پیچیده و منحصربه‌فرد را به تنهایی حل کند.

این مطلب به خصوص در قلمروی علوم داده صدق می‌کند. هر قدر که گیت‌هاب کوپایلت خوب باشد، به نظر نمی‌رسد بتواند حتّی با ساده‌ترین کتابخانهٔ یادگیری ماشین اتوماتیک رقابت کند. علوم داده فقط نوشتن کد خوب نیست؛ مجموعه‌ای از دانش حوزهٔ مربوطه، فهم دادهٔ مورد نظر و ریاضیات است.

با این حال نمی‌توان گفت که یک ابزار مبتنی بر هوش مصنوعی مانند این، تکنولوژی‌ای به پشتیبانی شرکتی چندمیلیاردی، در جریان کار روزانهٔ برنامه‌نویس‌ها در آینده نقش مهمّی ایفا نکند. یکی از مدافعان این ابزار از شرکت Google Cloud می‌گوید:

> «توسعه‌دهندگان همان قدر باید از این ابزار بترسند که ریاضی‌دانان از ماشین حساب.»

یک نگرانی دیگری که وجود دارد این است که اگر گیت‌هاب کوپایلت قطعات بزرگی از کدهای موجود را بازتولید می‌کند، ممکن است کپی‌رایت را زیر پا بگذارد یا از کدهای متن-باز بدون مجوّز مناسب در مصارف تجاری استفاده کند. حتّی یکی از فعّالان حوزهٔ وب احتمال این را داده است که افرادی که از مخزن‌های معروف نگهداری می‌کنند، کدهای غلطی قرار دهند تا این ابزار کارایی خود را از دست بدهد.

به هر حال وجود این تکنولوژی خالی از لطف نیست و قطعاً تغییر بزرگی در صنعت نرم‌افزار به وجود می‌آورد.

مراجع

[1] Inc., GitHub. OpenAI Launches GitHub Copilot: AI Focused On Code Generation. Should We Be Worried Now?, 2021.

[2] Newman, Jared. GitHub's new tool uses AI to craft code. Some developers are furious, 2021.

[3] T., Bex. OpenAI Launches GitHub Copilot: AI Focused On Code Generation. Should We Be Worried Now?, 2021.

برنامه‌ریزی پویا

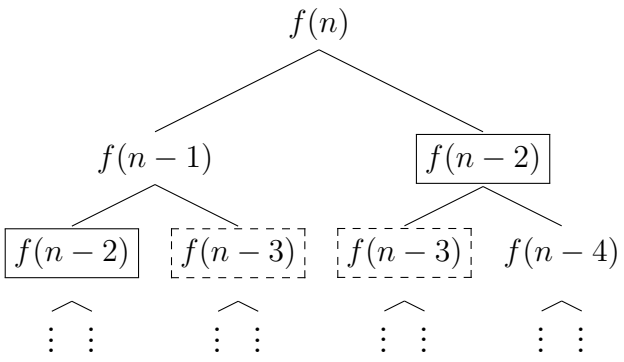
Dynamic Programming

امیرحسین رجبی و محمدرضا باطنی

در این نوشته قصد داریم درباره برنامه‌ریزی پویا صحبت کنیم. برنامه‌ریزی پویا یا Dynamic Programming یا اصطلاحاً DP یک روش حل مسئله است که با مصرف حافظه اضافی موجب افزایش سرعت حل مسئله می‌شود. با یک مسئله شروع می‌کنیم. می‌خواهیم جمله nام دنباله فیبوناچی را محاسبه کنیم. اولین ایده‌ای که به ذهن می‌رسد رابطه بازگشتی آن است و بنابراین یک تابع بازگشتی!

```
int fibo(int n) {
  if (n == 1 || n == 2) return 1;
  return fibo(n - 1) + fibo(n - 2);
}
```

اما با انجام تحلیلی زمانی، مشاهده می‌شود که این پیاده‌سازی از مرتبه نمایی است؛ یعنی با بزرگ شدن n، زمانی که لازم است تا جمله nام دنباله در خروجی چاپ شود به صورت نمایی رشد می‌کند.[1] این موضوع را به طور شهودی نشان می‌دهیم. درخت محاسبه زیر را در نظر بگیرید. این درخت نشان می‌دهد که به ازای فراخوانی `fibo(n)` چه فراخوانی‌های دیگری نیز انجام می‌شود. به‌جای `fibo(n)` از $f(n)$ استفاده شده‌است.



مشاهده می‌شود $f(n-2)$ و $f(n-3)$ دو بار فراخوانی شده‌است و همینطور با بررسی سطوح مختلف این درخت، فراخوانی‌های تکراری دیگری نیز دیده می‌شوند؛ اما اگر بتوانیم این مقادیر را ذخیره کنیم[2] و در فراخوانی‌های بعدی از آن‌ها استفاده کنیم، در زمان صرفه‌جویی کرده‌ایم. یعنی اولین بار که $f(n-2)$ محاسبه شد، مقدار آن در یک آرایه ذخیره شود. این همان مصرف اضافی حافظه است که در ابتدا به آن اشاره شد. در مسئله فیبوناچی، زیرمسئله‌ها مقادیر دنباله برای nهای کوچک‌تر است. در مسائل برنامه‌ریزی پویا نیز همین ماجرا برقرار است. ابتدا زیرمسئله‌ها مشخص می‌شوند و سپس پاسخ زیرمسئله‌های پرکاربرد در آرایه، ماتریس و... ذخیره می‌شوند. پیاده‌سازی زیر را برای مسئله فیبوناچی در نظر بگیرید:

```
int answer[N] = {0};
int fibo_top_down(int n) {
  if (answer[n])
    return answer[n]; // use the cached
   answer
  if (n == 1 || n == 2)
    return 1;
  answer[n] = fibo_top_down(n - 1) +
    fibo_top_down(n - 2);
  return answer[n];
}
```

در این پیاده‌سازی مقادیر محاسبه شده درون یک آرایه به نام `answer` ذخیره می‌شوند و در هر فراخوانی تابع، ابتدا بررسی می‌شود

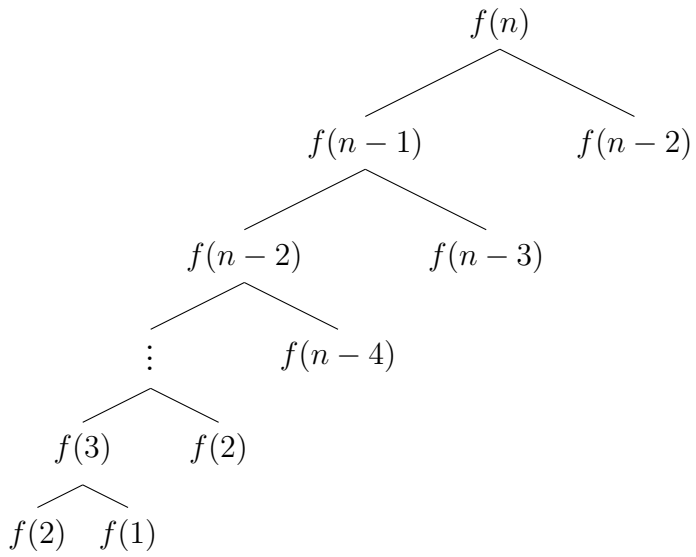
[1] اثبات برای علاقه‌مندان: اگر مقدار زمانی که طول می‌کشد تا تابع `fibo` به ازای ورودی n خاتمه یابد را $T(n)$ بنامیم و فرض کنیم $T(1)$ و $T(2)$ هر دو در ۱ ثانیه انجام شوند (به شرط درون تابع دقت کنید.) و همچنین فرض کنیم عملیات جمع زمان‌بر نباشد، به این رابطه بازگشتی برای تابع T می‌رسیم: $T(n) = T(n-1) + T(n-2)$؛ زیرا زمان اجرای تابع با ورودی n با مفروضات بالا، مجموع زمان اجرای فراخوانی‌های داخل تابع است. از طرفی

$$T(n) = T(n-1) + T(n-2) < 2T(n-1)$$

پس با تعریف $S(n) = \frac{T(n)}{T(n-1)}$ خواهیم داشت $S(n) < 2$ از طرفی $T(n) = S(3) \times S(4) \times \cdots \times$ اما $S(n) > 1 + \frac{1}{2} > \frac{3}{2}$ پس $1 + \frac{1}{S(n-1)}$ $S(n) > \left(\frac{3}{2}\right)^{n-2}$ که یعنی تابع $T(n)$ رشد نمایی دارد.

[2] به این کار Memoization گفته می‌شود. ذخیره جواب برای زیرمسئله‌های کوچک‌تر جهت عدم محاسبه دوباره آنها.

که آیا این مقدار قبلاً محاسبه شده‌است یا خیر. (در زبان سی، کد داخل بلوک if زمانی اجرا می‌شود که مقدار شرط ناصفر باشد. از آنجایی که همه خانه‌های آرایه در ابتدا با صفر پر شده‌اند و همچنین هیچ مقدار دنباله صفر نیست، این شرط همان کاری را می‌کند که می‌خواهیم.) اگر محاسبه شده بود مقدار آن که درون آرایه است برگردانده می‌شود. در غیر این صورت، فراخوانی‌های بازگشتی برای محاسبه آن انجام می‌شود. برای مثال با فراخوانی تابع با ورودی n، درخت محاسبه زیر به وجود می‌آید. این درخت $2n-3$ رأس دارد که نشان می‌دهد تنها به این تعداد فراخوانی انجام می‌شود. پس زمان خاتمه یافتن این تابع رابطهٔ خطی با n دارد که به مراتب از راه‌حل نمایی کمتر است.



به پیاده‌سازی بالا Top Down می‌گویند؛ زیرا از بالا پیمایش درخت محاسبه را شروع می‌کنیم. این پیاده‌سازی‌ها به صورت بازگشتی انجام می‌شوند؛ یعنی با فراخوانی $f(n)$ و سپس فراخوانی $f(n-1)$ و همین‌طور تا پایین درخت، محاسبه انجام می‌شود. اگر پیمایش این درخت از پایین به بالا باشد به آن پیاده‌سازی Bottom Up می‌گویند. یعنی با جمع کردن $f(1)$ و $f(2)$ مقدار $f(3)$ را به‌دست بیاوریم و سپس با جمع $f(3)$ و $f(2)$ مقدار $f(4)$ را به‌دست بیاوریم و همین‌طور تا $f(n)$ ادامه دهیم. پیاده‌سازی زیر این گونه است.

```
int fibonacci_bottom_up(int n) {
  int answer[n] = {0};
  answer[0] = answer[1] = 1;
  for (int i = 2; i < n; i++)
    answer[i] = answer[i - 1] + answer[i
  - 2];
  return answer[n];
}
```

البته دربارهٔ فیبوناچی، فقط ذخیره دو جمله قبلی دنباله کافی است.[۳]

```
int fibonacci(int n) {
  return fibonacci_internal(n, 1, 1);
}
int fibonacci_internal(int n, int a, int
 b) {
  if (n == 1) return a;
  return fibonacci_internal(n - 1, b, a +
   b);
}
```

بنابراین در مسائلی که به کمک این تکنیک حل می‌شوند دو اصل وجود دارد؛ وجود زیرمسئله‌های **هم‌پوشان**[۴] و **بهینه**[۵]. یعنی اولاً امکان خرد کردن مسئله اصلی به مسئله‌های کوچک‌تر وجود داشته باشد و سپس بتوان با تجمیع پاسخ زیرمسئله‌ها، پاسخ مسئله اصلی را محاسبه کرد؛ (هر مسئله‌ای که یک رابطه بازگشتی دارد در این دسته است زیرا حل مسئله اصلی وابسته به حل مسئله‌های کوچک‌تر است.) ثانیاً زیرمسئله‌ها کاملاً مجزا و بی‌ربط نباشند. یعنی حل یک زیرمسئله موجب تسریع محاسبه دیگر زیرمسئله‌ها شود. (بسیار مسئله‌هایی وجود دارند که زیرمسئله‌ها از هم مستقلند. به عنوان مثال الگوریتم مرتب‌سازی ادغامی[۶].)

اکنون سراغ مسئله‌ای دشوارتر می‌رویم. هدف در این مسئله خرد کردن مقداری پول به کمک تعدادی نوع سکه است. فرض کنید مقداری پول M داریم و همچنین n نوع سکه با ارزش‌های $C_1, C_2, \ldots, C_n$ داریم. به دنبال تعداد روش‌های خرد کردن M با این سکه‌ها هستیم. مثلاً اگر ۸ ریال داشته باشیم و سکه‌ها ۱ ریالی، ۲ ریالی و ۵ ریالی باشند یک روش $8=5+2+1$ و دیگری $8=5+1+1+1$ است. قبل از ادامه دادن، خودتان روی این مسئله فکر کنید. مسئله قابل شکستن به تعدادی زیرمسئله است؛ اما این زیرمسئله‌ها چه هستند؟ یک روش این است که مسئله را برای هر مقدار پول $N \leq M$ حل کنیم. یعنی تعداد روش‌های خرد کردن N به سکه‌های مذکور را بیابیم. اما به سادگی می‌توان بررسی کرد که رابطه‌ای میان این مسائل قابل تعریف نیست. پس باید زیرمسئله‌ها دقیق‌تر تعریف شوند. هر روش خرد کردن N به سکه‌ها بصورت $N = N_1 \times C_1 + N_2 \times C_2 + \cdots + N_n \times C_n$ است که $N_i \in \{0, 1, 2, 3, \ldots\}$. پس با انتخاب تعدادی سکه

[۳] این یک پیاده‌سازی Tail Recursive است. به مقاله مربوط به زبان Erlang مراجعه کنید.
[۴] Overlapping Subproblems
[۵] Optimal Substructure
[۶] Merge Sort

با ارزش C_n مانند c می‌توانیم به دنبال تعداد روش‌های خرد کردن $N - c \times C_n$ با سکه‌های $C_1, C_3, \ldots, C_{n-1}$ باشیم. یعنی همزمان تعداد سکه‌ها نیز کاهش یابند. پس تابع $f(N,k)$ را تعریف می‌کنیم که تعداد روش‌های خرد کردن N ریال با k سکه نخست باشد. اکنون می‌توانیم یک رابطه بازگشتی تعریف کنیم:

$$f(N,k) = f(N,k-1) + f(N-C_k,k) \quad (k \geq 1, N > 0)$$
$$f(N,k) = 0 \quad (N < 0 \text{ یا } k \leq 0)$$
$$f(0,k) = 1 \quad (1 \leq k \leq n)$$

در واقع، یا از سکه kام استفاده نمی‌کنیم که مسئله به $k-1$ سکه کاهش می‌یابد؛ یا مقدار C_k را از مقدار پول کم کرده و مسئله به استفاده از k سکه نخست و مقدار پول $N - C_k$ کاهش می‌یابد. همچنین خرد کردن صفر ریال با هر تعداد سکه‌ای تنها به یک روش امکان‌پذیر است و به این صورت است که از هیچ سکه‌ای استفاده نکنیم. پس مانند مسئلهٔ فیبوناچی، مقادیر تابع f را به محض محاسبه در ماتریسی به نام dp ذخیره می‌کنیم.[۷]

```
int solution(int N, int k, int coins[],
 int dp[][n])
{
  if (N < 0 || k <= 0)
    return 0;
  if (dp[N][k] != -1)
    return dp[N][k]; // use the cached
 answer
  if (N == 0)
    return 1;
  dp[N][k] = solution(N, k - 1, coins, dp
 ) +
    solution(N - coins[k], k, coins, dp);
  return dp[N][k];
}
int main() {
  int dp[M + 1][n];
  for (int i = 0; i <= M; i++)
    for (int j = 0; j < n; j++)
      dp[i][j] = -1;
  solution(M, n, coins, dp);
}
```

در تکه کد بالا، ماتریس dp تعریف شده‌است که $M+1$ سطر دارد و n ستون دارد و درایه dp[N][k] مقدار $f(N,k)$ را ذخیره می‌کند. یعنی تعداد روش‌های خرد کردن N ریال به کمک k سکه نخست. برای این منظور ابتدا همه درایه‌ها برابر -1 قرار داده شده‌است و این به معنی مقدار محاسبه‌نشده است. سپس درون تابع اگر dp[N][k] قبلاً محاسبه شده باشد (منفی یک نباشد)، مقدارش برگردانده می‌شود، در غیر این صورت محاسبه می‌شود و مقدار آن ذخیره می‌شود. می‌توان راه‌حلی Bottom Up نیز ارائه کرد که حتی حافظه‌ی مصرفی آن به جای ماتریسی $M \times n$، آرایه‌ای تک بعدی به طول M باشد. در راه حل Bottom Up مسئله فیبوناچی نحوه پر کردن آرایه روشن بود. اما در مسئله سکه‌ها، ماتریس مربوطه را به کمک رابطه بازگشتی به چندین روش می‌توان پر کرد. پس تمرکز این گونه راه‌حل‌ها روی این است که ماتریس یا آرایه dp چگونه به کمک رابطه بازگشتی پر شود. با توجه به رابطه بازگشتی که پیش‌تر معرفی کردیم، اگر بخواهیم ماتریس dp در پیاده‌سازی بالا را با فرض ثابت بودن k پر کنیم، پیش‌روی به صورت ستون به ستون خواهد بود. یعنی برای محاسبه $f(N,k)$ که مقادیر ستون kام هستند، نیاز به مقادیر $f(N,k-1)$ یعنی مقادیر ستون قبل و مقادیر $f(N-C_k,k)$ یعنی مقادیر در حال پر شدن همان ستون است. اینجا این ایده به ذهن می‌رسد که اگر برای پر کردن درایه‌های ماتریس فقط به مقادیر ستون فعلی و ستون قبلی نیاز است، چرا باید ماتریس را در حافظه ذخیره کنیم. حتی می‌توان دید که نیازی به ذخیره مقادیر دو ستون در حافظه نیست. تصور کنید مقادیر ستون kام ماتریس را در آرایه‌ای ذخیره کرده‌ایم و دنبال محاسبه مقادیر ستون $k+1$ام هستیم. می‌خواهیم مقادیر ستون $k+1$ام را در همین آرایه ذخیره کنیم. به وضوح C_{k+1} درایه اول ستون $k+1$ام برابر درایه‌های متناظر ستون kام است. (این درایه‌ها در آرایه بدون تغییر می‌مانند.) برای درایه C_{k+1}ام (خرد کردن C_{k+1} ریال با $k+1$ سکه نخست) دو انتخاب داریم؛ یا از سکه C_{k+1} استفاده نمی‌کنیم (مقدار متناظر با ستون kام) یا این‌که از این سکه استفاده کنیم که مسئله به خرد کردن صفر ریال با $k+1$ سکه نخست تبدیل می‌شود (یعنی درایه اول)؛ پس این دو مقدار را جمع کرده و در درایه C_{k+1}ام آرایه ذخیره می‌کنیم. به طور کلی مقدار جدید درایه Nام آرایه، مجموع مقدار پیشین این درایه در آرایه و درایه $N - C_{k+1}$ است. پیاده‌سازی زیر همین کار را انجام می‌دهد:

```
int solution_bottom_up(int M, int n, int
 coins[])
{
  int dp[M + 1] = {0};
```

[۷] این نام برای چنین ماتریس‌هایی (آرایه) در روش برنامه‌ریزی پویا مرسوم است زیرا برگرفته از نام تکنیک یعنی Dynamic Programming است.

```
    dp[0] = 1;
    for (int k = 0; k < n; k++)
      for (int N = coins[k]; N <= M; N++)
        dp[N] += dp[N - coins[k]];
    return dp[M];
  }
```

بنابراین در مسائلی که با این تکنیک حل می‌شوند چند گام طی می‌شود:

۱. پیدا کردن زیرمسئله‌های هم‌پوشان و بهینه که در صورت حل آن‌ها بتوانیم مسئله اصلی را حل کنیم. (این مرحله مهمترین گام حل مسئله است؛ زیرا مراحل بعدی روی این مرحله سوار می شوند.)

۲. پیدا کردن ارتباط میان این زیرمسئله‌ها. این ارتباط عموماً به صورت یک رابطه بازگشتی تعریف می‌شود.

۳. مشخص کردن آرایه یا ماتریس dp که قرار است پاسخ زیرمسئله‌ها در آن ذخیره شود و پر کردن درایه‌های آن به کمک رابطه بازگشتی. ممکن است این فرآیند به صورت سطر به سطر، ستون به ستون، قطری یا... باشد.

۴. حل زیرمسئله‌های بدیهی. این کار در مسائل استقرایی بیشتر شبیه اثبات مسئله برای پایه استقراست. در این مرحله درایه‌های خاصی از dp پر می‌شود.

۵. مشخص کردن پاسخ مسئله؛ یعنی پاسخ در کدام درایه یا درایه‌ها قرار دارد. ممکن است پاسخ جمع، ضرب یا تابعی از تعدادی درایه مشخص باشد.

مسئله‌ای برای حل

آریو برزن، سردار نام‌دار ایرانی، آخرین حلقه مقاومت ایرانیان در برابر اسکندر مقدونی بود. او پیشنهاد اسکندر مقدونی که گفت مقاومت را ترک کن تا فرمانده ایران شوی را رد کرد و تا آخرین قطره خون، برای دفاع از وطن خویش جنگید. آریو برزن با چهل سواره و پنج هزار سرباز پیاده و وارد کردن تلفات سنگین به دشمن، خط محاصره اسکندر را شکست و برای یاری به پایتخت به سوی پارسه شتافت؛ ولی سپاهیانی که اسکندر دستور داده بود از راه جلگه به طرف پارسه بروند، پیش از رسیدن او به شهر دست یافته بودند. او حاضر به تسلیم نشد و آن‌قدر در پیکار با دشمن پافشاری کرد که همه یارانش از پای افتادند و جنگ وقتی به پایان رسید که آخرین سرباز پارسی تحت فرمان آریوبرزن به خاک افتاده بود.

این سردار غیور در جنگ آخر خود دوست داشت سربازان را با ترتیبی در یک خط راست بچیند. فرض کنید در میدان جنگ، او ۱۰۰ سرباز پیاده و ۳۵ سواره در اختیار دارد. همچنین او یک ترتیب را «قدرتمند» می‌نامد اگر در هیچ جای این صف که شامل ۱۳۵ سرباز (پیاده و سواره) است اکیداً بیش از ۱۰ سرباز پیاده و همچنین اکیداً بیش از ۹ سرباز سواره به صورت متوالی دیده نشود. او در محاسبات خود نیاز دارد بداند چندتا از $\frac{135!}{100! \times 35!}$ جایگشت این سربازان قدرتمند هستند. [۸] (جواب را به پیمانه یک میلیون خروجی دهید. همچنین گروه سربازان سواره و پیاده‌ها را یکسان در نظر بگیرید.) مثال: اگر تعداد سربازان پیاده و سواره به ترتیب ۲ و ۳ باشد و همچنین محدودیت ما روی اکیداً بیشتر از ۱ سرباز پیاده متوالی و اکیداً بیشتر از ۲ سواره متوالی باشد، جواب ۵ است: (منظور از ۱، سرباز پیاده و منظور از ۲، سواره است.)

12122, 12212, 21212, 21221, 22121

[۸] پاسخ: ۸۸۸۸۵۸

رویای عمیق

Deep Dream

محمدرضا باطنی

شبکه‌های عصبی مصنوعی در سال‌های اخیر تأثیر غیر قابل انکاری روی دنیا گذاشته‌اند و روزبه‌روز کابردهای جدید آنها بیشتر دیده می‌شوند، کاربردهایی که نظر ما نسبت به هنر و مفهوم خلاقیت را عوض می‌کنند، یکی از این تکنولوژی‌ها که تصاویری خارق‌العاده و گاهی ترسناک خلق می‌کند رویای عمیق یا Deep Dream است.

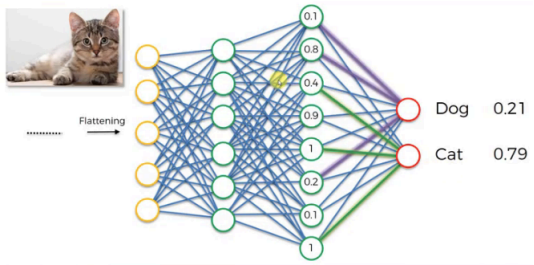
شکل ۱: شاید باورش سخت باشد اما تصویر بالا توسط این پدیده و بدون دخالت مستقیم انسان ساخته شده‌است.

در این مقاله سعی می‌کنیم به توضیح نرم‌افزار Deep Dream شرکت گوگل بپردازیم؛سپس بعد از توضیح مقدمات شبکه‌های عصبی که برای فهم Deep Dream لازم است به توضیح کامل نحوه کارکرد و ایده طلایی آن خواهیم پرداخت.

نرم‌افزار Deep Dream در سال ۲۰۱۵ توسط یکی از محققان گوگل، Alexander Mordvintsev ساخته شد. گوگل سیاستی دارد که به محققان خود اجازه می‌دهد تا ۲۰ درصد از زمان خود را روی تحقیقات حاشیه کار بگذارند و علاقه خود را دنبال کنند؛ زیرا مدیران گوگل معتقدند که یک کارمند نمی‌تواند در لحظه ذهن خود را خاموش و روشن کند تا روی کار تمرکز کند و اگر روی آن زمان جداگانه گذاشته نشود اشتیاق حل یک مسئله در ناخودآگاه او می‌ماند و تمرکز او را می‌گیرد.

الکساندر هم مشتاق بود بداند در داخل شبکه‌های عصبی چه می‌گذرد، تا قبل آن تصور می‌شد لایه‌های مخفی شبکه‌های عصبی صرفاً مانند «جعبه سیاه» هستند و کسی نمی‌داند در آن چه‌جور اعدادی جابه‌جا می‌شوند.

کمی به توضیح شبکه عصبی می‌پردازیم تا بتوانیم ایده الکساندر را بررسی کنیم. در شبکه عصبی برای تشخیص اینکه چه عکسی داده شده،



شکل ۲: نمونه‌ای از یک شبکه عصبی چند لایه برای تشخیص سگ و گربه.

تمام پیکسل‌های تصویر به عدد تبدیل می‌شوند و عددها در لایه ورودی (چپ‌ترین لایه) جمع‌آوری می‌شوند و با توجه به اعدادی که به هرکدام از یال‌های بین نورون‌های لایه اول و دوم نسبت داده شده‌اند، اعداد لایه دوم محاسبه می‌شوند و به همین ترتیب تا اعداد لایه خروجی محاسبه می‌شوند، در نهایت هر نورون لایه آخر متناظر با یک شی می‌شود (سگ، گربه، اتومبیل و...) و عدد نشان داده شده در آن لایه احتمال این را نشان می‌دهد که تصویر دیده شده، برابر شیء متناظر باشد؛ برای مثال تصویر بالا به احتمال ۷۹ درصد گربه است و به احتمال ۲۱ درصد سگ است.

اینکه اعداد هر لایه دقیقاً چگونه محاسبه می‌شود و اعداد یال‌ها از کجا داده می‌شوند به آموزش شبکه عصبی برمی‌گردد که از حوصله این مقاله خارج است اما در همین حد کافی است بدانید که اعداد روی لایه‌ها متغیر هستند و باید به دست بیایند، کار اینگونه انجام می‌شود که با نشان دادن تصویری که از قبل می‌دانیم چه چیزی است (مثلاً نشان دادن تصویر یک گربه) سعی می‌کنیم اعداد روی یال‌ها که متغیر بودند را به نحوی تغییر دهیم که به ازای تصویر داده شده، شبکه عصبی ما همان جوابی را بدهد که می‌دانستیم درست بود.

برای مثال اگر لایه خروجی شامل سه نورون مربوط به احتمال سگ، گربه و اتومبیل بودن بود و می‌دانستیم تصویر داده شده گربه است، هدف ما پیدا کردن اعداد روی یال‌هاست به نحوی که عدد لایه نهایی نزدیک به (۰، ۱، ۰) شود؛ یعنی شبکه عصبی هم فکر کند به احتمال نزدیک به ۱، تصویر داده شده گربه است.

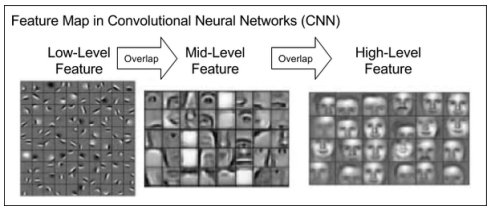
ایده الکساندر موردوینتسو این بود که این کار را برعکس انجام دهیم، به شبکه عصبی اختیار عمل بیشتری بدهیم و به جای تغییر دادن وزن‌های روی یال‌ها برای منطبق شدن با جواب، تصویر ابتدایی را تغییر دهیم تا به جواب خواسته شده برسیم. یعنی عدد پیکسل‌ها را متغیر و اعداد

روی یال‌ها را ثابت بگیریم.

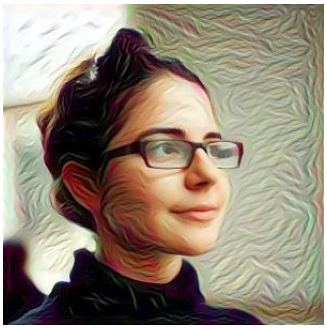
کار اول (نزدیک کردن وزن یال‌ها برای منطبق شدن تصویر با جواب) که برای آموزش شبکه عصبی از قبل استفاده می‌شد، با استفاده از الگوریتم معرفی شده توسط هینتون یعنی Backpropagation انجام می‌شود و الگوریتم الکساندر موردوینتسو برعکس این کار را انجام می‌دهد. یعنی اگر به عنوان مثال تصویر نقاشی «شب پرستاره» را به شبکه عصبی‌ای که از قبل آموزش داده شده بدهیم، (همین شبکه با سه نورون را در نظر بگیرید.) واضح است جواب لایه خروجی عددی نزدیک به (۰، ۰، ۰) است. (چون در این تصویر سگ یا گربه یا اتومبیل وجود ندارد.) اگر این بار به جای اینکه اعداد روی یال‌ها را تغییر دهیم، تصویر را تغییر دهیم، یعنی سعی کنیم پیکسل‌های تصویر ورودی را به نحوی تغییر دهیم که خروجی به جای (۰، ۰، ۰) نزدیک به (۱، ۰، ۰) شود، با این کار تصویر ورودی به نحوی تغییر می‌کند تا شبیه به یک سگ شود.

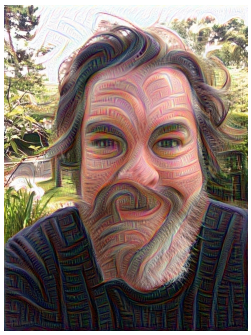
کارهای بی‌نظیری با این ایده می‌توان انجام داد؛ برای مثال به جای اینکه به شبکه عصبی بگوییم: «پیکسل‌ها را به نحوی تغییر بده تا به جواب (۰، ۰، ...۱، ۰، ...۰) برسی.» می‌توانیم یک تصویر دیگر به شبکه عصبی بدهیم تا جواب داده شده شبکه عصبی به ازای آن به دست آید؛ سپس با استفاده از الگوریتم موردوینتسو سعی کنیم تصویر دوم را به نحوی تغییر دهیم تا به جواب اول در لایه نهایی برسد. بدین نحو تصویر اول شمایل تصویر دوم را می‌گیرد؛ یعنی رنگ بندی تصویر اول را دارد اما مفهوم و اشیا تصویر دوم را می‌گیرد. می‌توانید نمونه بسیار زیبای این پدیده را در تصویر زیر ببینید. از دیگر کارهایی که می‌توان با

این پدیده کرد این است که به نحوی تصویر را تغییر دهیم تا به طور کلی در لایه‌های مخفی ابتدایی، اعداد بزرگ‌تری نمایش داده شوند. همانطور که در تصویر می‌بینید و در ابتدا هم الکساندر به همین موضوع فکر می‌کرد، لایه‌های مخفی شبکه عصبی کاملاً مرموز نیستند؛ به عبارتی به غیر از اینکه می‌دانیم لایه اول، پیکسل‌ها و لایه آخر احتمال بودن اشیا مختلف را نشان می‌دهد، می‌دانیم لایه‌های میانی اولیه، احتمال بودن خصوصیات ابتدایی مختلف و لایه‌های بعدی احتمال بودن خصوصیات پیشرفته‌تر را نشان می‌دهند. برای مثال ممکن است یک نورون در لایه دوم نشان دهنده وجود داشتن زاویه خاصی باشد و یک نورون در لایه دهم نشان دهنده وجود دایره باشد. اکنون می‌توان حدس زد که در



Deep Dream اگر به شبکه گفته شود که سعی کند به طور کلی اعداد لایه دوم را زیاد کند، چه اتفاقی می‌افتد. بله، تصویر به نحوی تغییر می‌کند که تعداد خطوط و زوایا (خصوصیات ابتدایی) افزایش یابد و تصویری خلق می‌شود که احتمالاً تا کنون در خیلی از نرم‌افزارهای موبایل شبیه آن را دیده‌اید و اکنون می‌دانید این تصاویر چگونه خلق می‌شوند. همچنین اگر به شبکه گفته شود که اعداد پیکسل‌ها را به نحوی تغییر





دهد که به طور کلی اعداد آخرین لایه یعنی اعداد نشان دهنده خود اشیا، زیاد شوند، تصاویر به نحوی تغییر می‌کنند که هر شکلی که ممکن است در تصویر دیده شود؛ به طور مثال اگر گوشه‌ای از ابرها در تصویر آسمان شبیه یک حیوان هست، آن را پررنگ‌تر کن و پیکسل‌ها را به نحوی تغییر بده که آن حیوان بهتر دیده شود، به قول خود الکساندر موردوینتسو داریم به کامپیوتر در این حالت می‌گوییم: «هر آنچه می‌بینی، از آن بیشتر به من نشان بده.» و در این حالت تصاویر بی‌نظیری شبیه رویا خلق می‌شود. تصویر ابتدای مقاله و تصاویر بی نظیر زیر همه با این روش تولید شده‌اند، لذت ببرید.

لازم به ذکر است این تصاویر از اجرای متعدد الگوریتم موردوینتسو به دست می‌آیند؛ یعنی یک بار الگوریتم اجرا و تعدادی از اشیا خیالی داخل تصویر اولیه بیرون می‌آیند. بار دوم تصاویر بیشتری خود را نشان می‌دهند و مثلاً بعد از ۵۰ بار تکرار الگوریتم این تصاویر خارق‌العاده

خلق می‌شوند. همچنین الگوریتم می‌تواند از تصویر ابتدایی واقعی یا حتی تصویر نویز کار خود را شروع کند و تصاویر مختلفی را بسازد.

مراجع

[1] https://news.artnet.com/market/google-inceptionism-art-sells-big-439352.

[2] https://www.youtube.com/watch?v=BsSmBPmPeYQ.

[3] https://slate.com/technology/2015/07/google-deepdream-its-dazzling-creepy-and-tells-us-a-lot-about-the-future-of-a-i.html.

[4] https://thereader.mitpress.mit.edu/deepdream-how-alexander-mordvintsev-excavated-the-computers-hidden-layers/.

پیش‌گویی ساختار سوم پروتئین‌ها

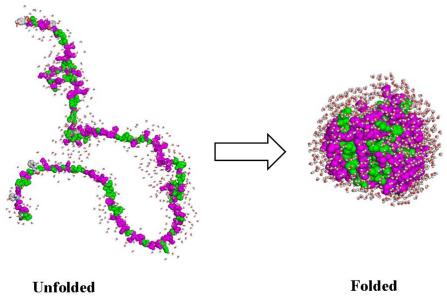
Protein Tertiary Structure Prediction

زهرا سراج

پروتئین چیست؟

پروتئین‌ها، کارگرهای سلول هستند! تقریباً تمام فرایندهای سلولی، به کمک پروتئین‌ها انجام می‌شوند؛ کاتالیز واکنش‌های سلول، کنترل عبورومرور مولکول‌ها به سلول، انتقال پیام بین سلول‌ها، حمل اتم‌ها و مولکول‌های مهم (مانند حمل مولکول اکسیژن به کمک هموگلوبین) و... .

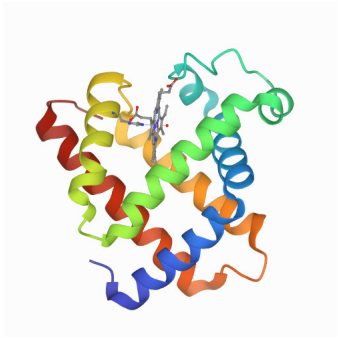
از پروتئین با عنوان یک درشت‌مولکول[1] یاد می‌شود؛ زیرا اجزای سازنده آن مولکول‌های کوچک‌تری به نام آمینواسید هستند. ۲۰ نوع آمینواسید، با ویژگی‌های شیمیایی متفاوت، وجود دارند. از اتصال آمینواسیدها به یکدیگر به واسطه‌ی پیوند کووالانسی، رشته‌ای از آمینواسیدها به وجود می‌آید که به آن پروتئین می‌گوییم. هر نوع پروتئین، یک توالی آمینواسیدی یکتا دارد. تا کنون، هزاران پروتئین با توالی آمینواسیدی منحصربه‌فرد، شناخته شده‌اند.



شکل ۱: توالی (سمت چپ) و ساختار سوم (سمت راست) یک پروتئین

ساختار سوم پروتئین

پروتئین‌ها با ساختار زنجیره‌ای از آمینواسیدها، که به آن ساختار اول یا توالی نیز گفته می‌شود، در بدن قادر به انجام کاری نیستند(!)؛ بنابراین، یک ساختار سه‌بعدی به خود می‌گیرند و به اصطلاح، تا می‌شوند[2]. به این ساختار سه‌بعدی، ساختار سوم پروتئین گفته می‌شود. ساختار سوم دو پروتئین با توالی متفاوت، ممکن است متفاوت باشد و بستگی به عملکردشان دارد، اما ساختار سوم دو پروتئین با توالی یکسان، یکسان است.



شکل ۲: ساختار سوم پروتئین Myoglobin

چرا شناسایی ساختار سوم پروتئین مهم است؟

همان‌طور که گفته شد، پروتئین‌ها، کارگرهای سلول هستند و هر پروتئین، مختص عملی در سلول است. شناسایی عملکرد [3] پروتئین‌ها برای پژوهشگران، اهمیت زیادی دارد و به فهم چگونگی عملکرد سلول کمک می‌کند. به دلیل اهمیت این مسئله، پیش‌گویی عملکرد پروتئین، یکی از مسائل مهم در بیوانفورماتیک است. عملکرد یک پروتئین، به طور مستقیم به ساختار سوم آن بستگی دارد؛ بنابراین، با شناسایی و مطالعه ساختار سوم پروتئین‌ها، می‌توان اطلاعات سودمندی در مورد عملکرد آن‌ها به دست آورد.

روش‌های آزمایشگاهی شناسایی ساختار سوم پروتئین

رایج‌ترین روش‌ها برای تعیین ساختار سوم پروتئین، X-ray crystallography و Nuclear magnetic resonance هستند. ساختار سوم بیش از ۱۸۰ هزار پروتئین در پایگاه داده Protein Data Bank[4] وجود دارد و به صورت رایگان، قابل دسترسی است.

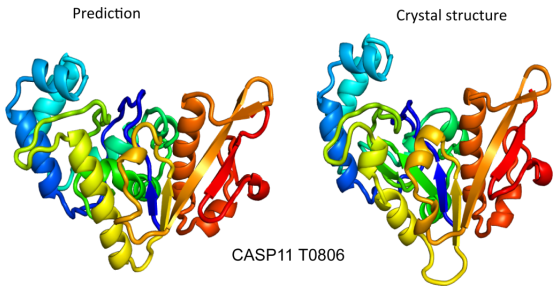
[1] Macromolecule
[2] Folding
[3] Function
[4] https://www.rcsb.org/

پیش‌گویی ساختار سوم پروتئین

روش‌های آزمایشگاهی پیش‌گویی ساختار سوم پروتئین، علی‌رغم دقت بالا، از لحاظ زمانی و اقتصادی بسیار هزینه‌بر هستند؛ بنابراین این سوال مطرح می‌شود: آیا می‌توان الگوریتمی یافت که با داشتن توالی یک پروتئین، ساختار سوم آن را با دقت قابل قبولی پیش‌گویی کند؟ به این مسئله در بیوانفورماتیک، پیش‌گویی ساختار سوم پروتئین می‌گویند.

مسابقات CASP[5]

از سال ۱۹۹۴، برای پیش‌گویی ساختار پروتئین، مسابقات جهانی دوسالانه به نام CASP برگزار شده‌اند. هر فرد یا گروه، با یک الگوریتم پیش‌گویی ساختار پروتئین، در این مسابقات شرکت می‌کند و این الگوریتم‌ها، مورد ارزیابی قرار می‌گیرند. در نهایت الگوریتمی که بیشترین صحت[6] را داشته باشد، به عنوان برنده اعلام می‌شود.
در این مسابقه، توالی چند پروتئین، در اختیار شرکت‌کنندگان قرار می‌گیرد. شرکت‌کنندگان باید در زمانی محدود و مشخص، ساختار سوم پروتئین‌ها را پیش‌گویی و ارائه کنند. سپس این ساختارها با ساختار آزمایشگاهی مقایسه می‌شوند.



شکل ۳: ساختار آزمایشگاهی (سمت راست) و ساختار پیش‌گویی شده (سمت چپ)

در شکل ۳ ساختار پیش‌گویی‌شده برای پروتئین هدف T0806 (سمت چپ) و ساختار به دست آمده از آزمایشگاه (سمت راست) در دوره یازدهم CASP را ملاحظه می‌کنید.
در آخرین دوره CASP (دوره‌ی چهاردهم) که در سال ۲۰۲۰ برگزار شد، برنامه‌ی هوش مصنوعی AlphaFold2 از Google DeepMind برنده شد؛ یک الگوریتم که از یادگیری عمیق برای پیش‌گویی ساختار پروتئین کمک می‌گیرد. AlphaFold2 در بخش اهداف پروتئینی نسبتاً سخت، امتیاز ۹۰ از ۱۰۰ را به دست آورد؛ در حالی که سایر تیم‌ها حداکثر امتیاز ۷۵ را به دست آوردند.
میزان بالای صحت پیش‌گویی این الگوریتم، به قدری بود که دانشمندان و مهندسان را در حیرت فرو برد و سروصدای زیادی به پا کرد. در بسیاری از موارد، میزان صحت پیش‌گویی، قابل رقابت با ساختار آزمایشگاهی بوده است. بسیاری از متخصصان معتقدند، AlphaFold2 پیشرفت شگرفی در زمینه پیش‌گویی ساختار پروتئین‌ها محسوب می‌شود.

[5] Critical Assessment of Protein Structure Prediction
[6] Accuracy

مراجع

[1] Alberts, Bruce, Johnson, Alexander, Lewis, Julian, Morgan, David, Raff, Martin, Roberts, Keith, Walter, Peter, Wilson, John, and Hunt, Tim. *Molecular biology of the cell*. WW Norton & Company, 2017.

[2] Costanzo, Linda S. This will never do, 2018.

[3] Jumper, John, Evans, Richard, Pritzel, Alexander, Green, Tim, Figurnov, Michael, Ronneberger, Olaf, Tunyasuvunakool, Kathryn, Bates, Russ, Žídek, Augustin, Potapenko, Anna, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[4] Moult, John, Fidelis, Krzysztof, Kryshtafovych, Andriy, Schwede, Torsten, and Tramontano, Anna. Critical assessment of methods of protein structure prediction (casp)—round x. *Proteins: Structure, Function, and Bioinformatics*, 82:1–6, 2014.

[5] Williamson, Michael. *How proteins work*. Garland Science, 2012.

Interview
مصاحبه

هوش مصنوعی: چشم‌انداز و موقعیت‌ها
مصاحبه با دکتر مصطفی محسن‌وند

مهمان این شماره از فصل‌نامهٔ حلقه، دکتر مصطفی محسن‌وند[1]، محقق دانشگاه MIT و محقق یادگیری ماشین در شرکت Apple هستند. ایشان تحصیلات آکادمیک خود را در رشته‌های برق و مهندسی پزشکی در دانشگاه صنعتی امیرکبیر شروع کردند، در مقطع ارشد در دانشگاه آکسفورد به مطالعهٔ مدل‌سازی ریاضی پرداختند و هم‌اکنون در مقطع دکترا در زمینهٔ هوش مصنوعی و علوم شناختی در دانشگاه MIT مشغول هستند. دکتر محسن‌وند، رسیدن به شناخت و درک از خود و استفاده از آن برای ساخت ابزار کارآمد را دلیل جذابیت هوش مصنوعی و نوروساینس[2] برای خود معرفی می‌کنند. در ادامه، بخش‌هایی از مصاحبه با ایشان در تاریخ ۱۸ مرداد ۱۴۰۰ آمده است که در آن به هوش مصنوعی، نوروساینس و فرصت‌های پیرامون آن‌ها پرداخته می‌شود.

شکل ۱: دکتر مصطفی محسن‌وند

سوال اول: پیشنهاد شما برای کسانی که در ابتدای راه هوش مصنوعی هستند و قرار است تازه این مسیر را شروع کنند، چه خواهد بود؟ ابتدا مفاهیم ریاضیاتی در هوش مصنوعی را تقویت کنند یا مستقیما وارد فاز برنامه نویسی و اجرایی کار شوند؟

اینکه مستقیما وارد فاز برنامه نویسی شوند را توصیه نمی‌کنم. به این دلیل که بعد ها دچار این تصور می‌شوند که تمام اقدامات هوش مصنوعی با سعی و خطا و برنامه نویسی انجام خواهد شد. این تصوری است که حتی خود من هم در آن گیر افتاده بودم و فکر می‌کردم اگر کتابخانه‌های تخصصی این حوزه به عنوان مثال کتابخانه‌های برنامه نویسی مرتبط با یادگیری عمیق را یاد بگیرم و شروع به برنامه نویسی کنم، می‌توانم مشارکت خوبی در این زمینه از خود به نمایش بگذارم و موارد جدیدی را کشف کنم، در صورتی که چنین چیزی نبود. تمامی کارهای قابل توجه در هوش مصنوعی، به پایه تئوریک قوی نیاز دارند؛ من‌جمله ریاضیات، جبر خطی و آمار و احتمالات. بخش دیگر شامل نظریه اطلاعات و یادگیری ماشین خواهد بود. پیشنهاد می‌کنم که دوره‌های مربوطه را حتما بگذرانید. در کل، این حوزه شاخه‌های مختلفی دارد که می‌توان به طرق مختلف به آن ورود کرد. به عنوان مثال می‌توانید وارد فیلد علوم اعصاب بشوید و نحوه عملکرد مغز را مطالعه کنید و از این طریق به هوش مصنوعی ورود کنید. من افراد زیادی را می‌شناسم که در این زمینه فعالیت می‌کنند و ایده‌های جالبی دارند؛ یا افرادی هستند که پیش زمینه برق و مخابرات و کنترل دارند و ایده‌های خود را در این زمینه پرورش داده‌اند. بنابراین تمامی مواردی که گفته شد، لازم هستند و نمی‌توانیم این مفاهیم را نادیده بگیریم و به سراغ بخش برنامه نویسی برویم. لازم به ذکر است که نباید حتما مفاهیم تئوریک را قوی کنید و سال‌ها روی آن مطالعه داشته باشید و بعد به دنبال مهارت‌های برنامه نویسی بروید؛ سعی کنید همزمان با هم این موارد را جلو ببرید.

به عنوان مثال، من به دانشجویان دکتری که در آزمایشگاه ما برای کار در هوش مصنوعی مشغول می‌شوند، توصیه می‌کنم که همزمان کتاب All of Statistics به عنوان یک کتاب در حوزه‌ی آمار و یک کتاب خوب دیگر در حوزه‌ی یادگیری ماشین مانند Information The-ory,Inference and Learning Algorithms نوشته‌ی دیوید مک‌کای یا Pattern Recognition and Machine Learning نوشته‌ی کریستوفر بی‌شاپ را شروع به خواندن کنند و در کنار این‌ها، یادگیری دو کتابخانه برنامه نویسی مانند scikit-learn و pytorch یا TensorFlow را در برنامه خود قرار دهند تا بتوانند برخی از الگوریتم‌ها را دستی پیاده‌سازی کرده و نسبت به نحوه عملکرد آن‌ها دید پیدا کنند. بعد از ممارست و تمرین در این راه که یک ماهه بدست نمی‌آید، خواندن مقالات شروع می‌شود تا بتوانید بفهمید که ایده‌های جدید کدام هستند. به عنوان مثال، اگر بخواهید در زمینه بینایی ماشین ورود کنید، در ابتدا با خواندن کتاب‌ها و مقالات ابتدایی این حوزه، به این بر می‌خورید که از CNN استفاده می‌کنند در حالی که اگر مقالات جدید را بررسی کنید، متوجه خواهید شد که در حال حاضر از CNN عبور کرده‌اند و به سمت transformerها حرکت کرده‌اند یا از معماری Dense

[1] https://www.media.mit.edu/people/mmv/
[2] Neuroscience

استفاده می‌کنند که درک این‌ها کمی سخت‌تر است و آموزش این چنین سیستم‌هایی پیچیده‌تر است؛ به همین دلیل، شما هر کاری کنید ممکن است احساس کنید یک مقدار عقب هستید و بهتر هست هر چه زودتر پایه‌ها را قوی کنید تا بتوانید خواندن مقالات را شروع کنید. البته، این توصیه برای افرادی است که قصد دارند کار تحقیقاتی کنند؛ اگر شما قرار است که به عنوان مثال نرم‌افزاری کاربردی بسازید که از هوش مصنوعی استفاده می‌کند، راه‌های سریع‌تری برای رسیدن به این نرم‌افزار وجود دارد.

سوال دوم: یکی از مواردی که خیلی به چشم می‌خورد، این است که وقتی صحبت از هوش مصنوعی و یادگیری ماشین می‌شود، ذهن‌ها بیشتر به سمت یادگیری عمیق، NLP یا مواردی مانند بینایی ماشین معطوف می‌شود. به نظر شما در این فیلد، کدام حوزه‌ها هستند که خیلی مورد توجه قرار نگرفته‌اند ولی در آینده جای رشد خواهند داشت؟

سوال خیلی خوبی را مطرح کردید. نکته‌ای که وجود دارد این است که NLP و بینایی ماشین، هر دو حوزه‌های خیلی خاصی هستند و ویژگی مشترک که این دو فیلد این است که شما می‌توانید با چشم و گوش معمولی یک انسان هم بررسی کنید که آیا نتایج بدست آمده خوب هستند یا خیر. در واقع می‌توانید الگوریتم‌هایی طراحی کنید که در آن از شهودی که راجع به دیدن یا فهم زبان از آن دارید، استفاده کنید؛ اما این موضوع الزاما در شاخه‌های دیگر هوش مصنوعی درست نیست و نمی‌توانید به راحتی نتایج را تست کنید و به صورت شهودی طراحی کنید. بنابراین، دلیل پیشرفت بسیار این دو فیلد نیز همین بوده است که خیلی راحت‌تر می‌توان ایده‌ها را پیاده سازی و راستی آزمایی کرد. ولی اگر یک قدم عقب‌تر برویم، می‌بینیم که اگر شهود را از این دو حوزه برداریم، موارد پایه‌ای‌تر می‌ماند که یادگیری آن‌ها بسیار اهمیت دارد و اصلا یکی از دلایل رشد سریع یادگیری عمیق، پیدا کردن اصولی است که مستقل از حل مسئله‌ی موردنظرمان باشد؛ اصول پایه‌ای مثل آن که پی بردیم چگونه می‌توان با Gradient Descent، شبکه‌های دلخواه را بهینه کرد یا چگونه این شبکه‌ها را تعدیل کرد تا بیش برازش رخ ندهد. بنابراین، این درک عمیقی که رخ داده الزاما به بینایی ماشین یا پردازش زبان طبیعی ربطی ندارد و بیشتر به درک فرد از نظریه اطلاعات و آمار و احتمالی که پشت این تحلیل‌هاست، برمی‌گردد. من احساس می‌کنم حوزه‌های بسیار دیگری مانند یادگیری تقویتی و یادگیری نیمه نظارتی (Semi-Supervised Learning) وجود دارد که دید خیلی عمیق‌تری به فرد می‌دهد و پیشرفت‌هایی که اخیرا چه در حوزه بینایی ماشین چه NLP صورت گرفته، به این سمت حرکت می‌کند که با سیستم‌های کلی‌تری کار کنیم. کلی‌تر به این معنی است که به عنوان مثال، اگر یک CNN که قبلا در بینایی ماشین استفاده می‌شد را نگاه کنید، برای این طراحی شده که بتواند یک عکس را درک کند و به اصطلاح یک Inductive Bias دارد؛ یعنی مستقیما از این حقیقتی که عکس‌ها از پیکسل درست شده‌اند و این پیکسل‌ها در همسایگی هم قرار دارند، برای طراحی هوش مصنوعی استفاده می‌کنیم. حال، اگر به مقالات جدیدتر (از سال ۲۰۲۰ به بعد) نگاه کنیم، می‌بینیم که درحال فاصله گرفتن از این Inductive biasهاییم و به سمت سیستم‌هایی می‌رویم که دیگر هیچ فرضی راجع به این که داده‌ها چگونه باشند، ندارند. به عنوان مثال همین، چند روز قبل مقاله‌ای آمده است از DeepMind به نام perceiver IOکه سیستمی است که می‌توان به آن عکس، ویدیو، زبان یا صدا دهید و برای آن فرقی نمی‌کند که ورودی چیست. بنابراین، به نحوی طراحی شده‌است که با هر نوع ورودی می‌تواند کار کند و من فکر می‌کنم این مسیری است که خیلی آینده‌گرایانه‌تر و جذاب‌تر است. به عنوان مثال، در حال حاضر مدل‌هایی مثل transformerها تقریبا در همه‌ی مدل‌های یادگیری ماشین جای خود را باز کرده‌اند و این مدل‌ها inductive bias خیلی کمی دارند. به اصطلاح می‌گوییم که مکانیزم داخلش Self-Attention است و می‌تواند convolution و recurrence مثل RNN و LSTM و رفتار مواردی که قبلا طراحی کرده‌ایم را یاد بگیرد و خیلی کلی‌تر است. اگر می‌خواهید وارد یادگیری ماشین شوید و در دراز مدت موفق باشید، پیشنهاد می‌کنم روی این موارد تمرکز کنید. مواردی که عمیق و ساده هستند، inductive bias ندارند و روی همه جور مسئله‌ای می‌توان از آن استفاده کرد. البته این دیدی است که در حال حاضر دارم و شاید اگر پنج سال دیگر از من بپرسید، نظرم عوض شده باشد ولی در حال حاضر یادگیری مواردی مثل transformerها، contrastive learning و پیدا کردن یک دهک از روی یادگیری تقویتی به طراحی شبکه و الگوریتم‌هایی که روی همه نوع داده‌ای کار می‌کنند، خیلی می‌تواند کمک کند.

سوال سوم: معمولا اینگونه مطرح می‌شود که مادر تمامی مواردی که یادگیری ماشین جدید را به وجود آورده است، می‌تواند شبکه‌های عصبی باشد که الهام گرفته از مغز انسان و به قصد شبیه‌سازی آن بوده است. با توجه به این که شبکه‌های عصبی تفاوت زیادی با مغز انسان دارند، به نظر شما چه تسک‌هایی را بهتر و چه تسک‌هایی را بدتر از مغز انسان انجام می‌دهند؟

تفاوت بین شبکه‌های عصبی و مغز خیلی زیاد است و این که به شبکه‌های عصبی چنین نامی داده‌اند، به دلیل تاریخ پشت آن است: در آن اوایل که افراد در پی اختراع آن بودند، فرض می‌کردند که واقعا یک سری نورون داریم و سیگنال‌ها وارد نورون‌ها شده و تجمیع می‌شود؛ یعنی مدلی به شدت ساده شده از سلول‌های مغز بود ولی چیزی که

در حال حاضر به آن یادگیری عمیق می‌گوییم، دیگر هیچ ربطی به نورون‌ها ندارد و عملا داریم یک سری تبدیل خطی را با یک سری تبدیل غیرخطی ترکیب می‌کنیم که توسط Gradient Descent بهینه می‌شود. بنابراین، حتی دیگر شبیه آن نورون‌ها هم نیست. در واقع یادگیری عمیق، هنر بهینه کردن توابع غیرخطی بزرگ است. یعنی یک تابع غیرخطی با پارامترهای زیاد داریم که می‌خواهیم آن را بهینه کنیم و اینکه به چه نحوی این امر را انجام دهیم، مربوط به یادگیری عمیق است و ارتباطی با مغز ندارد. حال برای مقایسه اینکه چه تفاوت‌ها و شباهت‌هایی با مغز دارند، مثال ساده‌ای را مطرح می‌کنم. در مغز انسان عدد نداریم؛ اما در شبکه عصبی، یک سری عدد را پردازش می‌کنیم در صورتی که در مغز انسان عصب‌ها جرقه می‌زنند و در فاصله بین این جرقه‌هاست که اطلاعات در حال پردازش است. این تنها قسمتی از کار مغز است و این عصب‌ها از طریق مواد شیمیایی نیز باهم ارتباط برقرار می‌کنند؛ انواع neurotransmitterها هستند که بین این عصب‌ها رد و بدل می‌شوند و از طرف دیگر، تنها سلول‌های مغز فقط از نوع عصبی نیستند و سلول‌های دیگری مانند Glial Cells، کارهای فوق‌العاده پیچیده‌ای داخل مغز انجام می‌دهند. به عنوان مثال، از نورون‌ها محافظت می‌کنند، آن‌ها را رشد می‌دهند، عذاب می‌دهند. این سلول‌ها می‌توانند طی یک سری فرایندهای شیمیایی و الکتریکی با عصب‌ها برهمکنش داشته باشند که هیچ کدام از این موارد در ساختار یادگیری عمیق لحاظ نشده‌است؛ یعنی یادگیری عمیق، فوق‌العاده از مغز ساده‌تر است. با این مقدمه، عده‌ای هستند که سعی می‌کنند مدارهایی شبیه مدارهای مغز درست کنند تا بتوانند یک سری از فرضیه‌ها را تست کنند. مثلا یک فرضیه داریم که برای تصمیم گیری در مغز، سلسله مراتبی وجود دارد (درخت تصمیم). حال، با تعدادی اطلاعات آزمایشگاهی جمع‌آوری شده از مغز، با یادگیری عمیق یک شبکه عصبی شبیه به آن چیزی که تصور می‌کنند در مغز درحال رخ دادن است، درست می‌کنند و بعد، می‌توانند با آموزش آن، فرضیه مد نظر را تایید یا نقض کنند. ولی این به این معنا نیست که فکر کنیم چیزی دقیقا مشابه یادگیری عمیق در مغز انسان در حال رخ دادن است. در یادگیری عمیق از gradient descent و back propagation استفاده می‌کنیم، در صورتی که به نظر نمی‌رسد در مغز چنین اتفاقی در حال رخ دادن باشد. محققان می‌گویند که منطقی نیست back propagation اصطلاحاً bio-logically plausible باشد، یعنی سیستم‌های زیستی نیز همین‌گونه کار کنند. بنابراین، تفاوت‌هایشان با هم خیلی زیاد است. شبکه‌های عصبی که ما می‌سازیم، عموما خیلی ساده هستند. یک سری لایه داریم و نورون‌هایی که در هر لایه هستند، با هم ارتباطی ندارند؛ زیرا ارتباط‌ها بین لایه‌هاست. ساده‌سازی‌های بسیاری صورت گرفته است که بتوانیم آن را برنامه نویسی کنیم و طوری بسازیم که بتوان آن را به صورت موازی روی یک GPU آموزش داد. در صورتی که در مغز، فرایند خیلی پیچیده‌تری در حال رخ دادن است و تعداد زیادی اتصال و ارتباطات به اصطلاح کهن وجود دارد. مواردی که به مدت میلیون‌ها سال در حال تکامل بوده است. از حدود ۵۴۰ میلیون سال پیش که به تدریج اولین مغزها به وجود آمدند، یک سری ساختارها نهادینه شده که مثلا بصل‌النخاع، تالاموس، هیپوکمپوس و سپس قشر مغز (cortex) به وجود آمده است؛ یعنی یک فرایند تکاملی میلیون‌ها ساله وجود داشته که بعضی از این ساختارها را به وجود آورده است و ما هنوز نمی‌دانیم برخی از این ساختارها دقیقا چه کار می‌کنند ولی می‌دانیم که بین همه‌ی جانوران مشترک هستند. هنوز به چنین درکی در شبکه‌های عصبی نرسیده‌ایم و نمی‌دانیم که به عنوان مثال، ما هم باید هیپوکمپوس بسازیم یا خیر. می‌توانیم یک چیز کلی درست کنیم و صرفاً خودش با ترینینگ یاد بگیرد که همان کارها را انجام دهد. در واقع، پیچیدگی در مطالعه مغز خیلی زیاد است و هنوز ۹۹ درصد آن به صورت معما باقی مانده و کسی نمی‌داند که دقیقا چه کاری می‌کند.

سوال ۴: در دهه‌های مختلف عواملی بودند که باعث می‌شدند رشد و پیشرفت این حوزه متوقف شود؛ مثلا تا سال ۲۰۱۰، این مانع کم بودن توان پردازشی بود. به نظر شما، درحال حاضر این مانع چه چیزی است و اصلا چنین مانعی وجود دارد یا خیر؟

هنوز موانع زیادی وجود دارد. یکی از آن‌ها این است که شبکه‌های عصبی که در یادگیری عمیق می‌سازیم، پارامترهای زیادی دارند و این ما را محبور می‌کند که data خیلی زیادی به این شبکه‌ها بدهیم تا چیزی را یاد بگیرند. مثلا شما می‌خواهید شبکه‌ای درست کنید که فرق سگ و گربه را تشخیص بدهد، یعنی عکس سگ و گربه را به آن بدهیم و بتواند آن‌ها را تمییز دهد. شما مجبور هستید که صدها و حتی هزاران عکس بدهید تا بتواند به شکل قابل قبولی این دو را تفکیک کند. در صورتی که اگر به یک بچه چند عکس از سگ و گربه نشان بدهید، دیگر به راحتی می‌تواند آن را تعمیم دهد و مثال‌های جدید را نیز تفکیک کند و این کار را با داده‌های بسیار کمی انجام می‌دهد.

این مسئله که در یادگیری عمیق sample complexity یا sam-ple efficiency نام دارد، هنوز حل نشده‌است. این بدین معنا نیست که کسی ایده‌ای ندارد؛ بلکه ایده‌های متنوع و جالبی مانند Self-supervised learning وجود دارد که من هم روی آن کار می‌کنم؛ ایده این است که اگر تعدادی وسیله کنار بچه‌ای بگذارید، بدون اینکه به او بگویید با وسیله‌ها بازی کند، خودش شروع به بازی می‌کند و به واکنش بزرگترها توجه می‌کند و این کار را به صورت اتوماتیک انجام می‌دهد. حال، ما سعی کرده‌ایم این کار را با شبکه‌های عصبی شبیه‌سازی

کنیم؛ یعنی شبکه عصبی‌ای بسازیم که اول با دیتا بازی کند، بدون اینکه کسی بگوید که چه کاری باید انجام دهد (supervision) و از طریق این بازی کردن، به یک هوش اولیه دست پیدا می‌کند که بعد با چند مثال کوچک (مثلا چند عکس سگ و گربه) خودش بتواند آن را تعمیم بدهد. این یکی از مسائل بسیار مهم است که هنوز حل نشده و شاید سال‌ها طول بکشد تا حل شود.

مسئله دیگری که وجود دارد، این است که شبکه‌های عصبی پارامترهای زیادی دارند که باعث می‌شود نتوانیم دقیق بگوییم آیا سیستم یاد گرفته است یا صرفاً درک ناقصی بدست آورده که فقط روی test data set جواب می‌دهد و اگر دیتای متفاوتی به آن بدهیم، نمی‌تواند خوب کار کند.

برای همین، در زمینه finance خیلی کم از شبکه عصبی استفاده می‌شود یا اصلا استفاده نمی‌شود، چون اگر یک بار اشتباه کند، منجر به ضرر بزرگی خواهد شد.

ما توانایی تست کامل را نداریم، یک data set برای یادگیری داریم و یک data set برای تست کردن. data set تست ما محدود است و نمی‌توان گفت شبکه می‌تواند به هر data set تستی generalize کند. بخاطر پارامترهای زیادی که دارد، از لحاظ ریاضی می‌توان نشان داد که همه شبکه‌های عصبی درحال overfit کردنند و چیز دیگری را یاد می‌گیرند که الزاما درک مدنظر ما نیست. همچنین چون در هنگام test دیتاهای جدید استفاده می‌شود، بازدهی به شدت افت می‌کند. برای همین استفاده از این شبکه‌ها در دنیای واقعی همچنان با شک و تردید همراه است و باعث می‌شود که در حوزه‌های مهمی مثل پزشکی و مالی استفاده چندانی نشود.

مشکل دیگری که با شبکه‌های عصبی داریم این است که ما هنوز نمی‌دانیم که این شبکه‌ها چگونه یاد می‌گیرند و تئوری ریاضی و مدونی از اینکه فرآیند یادگیری در این شبکه‌ها چگونه رخ می‌دهد نداریم. البته ایده‌های خوبی مثل information bottleneck و renormalization group مطرح شده‌است ولی همچنان تئوری مدونی نداریم و تا وقتی تئوری خوبی نداشته باشیم که همه پدیده‌ها را توصیف کند، نمی‌توانیم با اطمینان از آن در کاربرد استفاده کنیم.

سوال ۵: لطفا درباره حوزه‌های مختلف cognitive science و neuroscience توضیح دهید. به نظر شما برای ورود به این حوزه‌ها چه چیزهایی باید یاد گرفت؟

neuroscience حوزه‌های بسیار گسترده‌ای دارد. ولی اگر بخواهیم با دید مقیاس به آن نگاه کنیم، از اندازه مولکولی و سلولی (molecular neuroscience and cellular neuroscience) شروع می‌شود. سپس mesoscale neuroscience را داریم که در آن هزاران نورون مورد مطالعه قرار می‌گیرند و به آن systems neuroscience می‌گویند که در آن سیستم‌های کوچک در مغز را بررسی می‌کنیم. بعد از آن به مقیاس‌های بزرگ‌تر مانند cognitive neuroscience و behavioral neuroscience می‌رسیم که در آن رفتارهای یک موجود را بررسی می‌کنیم؛ مواردی مثل حافظه، توجه، تصمیم‌گیری و . . . . هرکدام از این حوزه‌ها بسیار گسترده‌اند و نمی‌توان همه را عمیقا یاد گرفت. عموما شما در یکی از این حوزه‌ها به فعالیت می‌پردازید و این به خودتان و دانش قبلیتان بستگی دارد؛ مثلا اگر شما دانش زیست شناسی داشته باشید بهتر است که به molecular neuroscience و cellular neuroscience بپردازید و برای مثال، به این فکر کنید که بیان شدن ژن‌های مختلف چه تاثیری در رفتار نورون می‌گذارد یا sectorهایی که روی نورون وجود دارند، چه رفتارهایی دارند و چگونه کار می‌کنند، یا نورون‌ها چگونه با هم شبکه تشکیل می‌دهند، چگونه رشد می‌کنند و چطور بعد از learning and training عوض می‌شوند.

اگر شما دانش قبلی برق و کامپیوتر و دیگر رشته‌های مرتبط داشته باشید، می‌توانید وارد حوزه systems neuroscience و cognitive neuroscience شوید. یا مثلا با دانش روانشناسی cognitive neuroscience حوزه مناسب‌تری برای شما خواهد بود.

حتی از فلسفه، مسیر متفاوتی به سمت neurosience وجود دارد. من افرادی را می‌شناسم که در این زمینه کار می‌کنند و فکرهای عمیق و اصول خاصی در ذهنشان دارند؛ مثلا یک موجود هوشمند چگونه باید باشد و چه کارهایی انجام دهد. بعد آن را به ساختار مغز و فرایندهایی که در مغز اتفاق می‌افتد، ربط می‌دهند.

همچنین، افرادی که اقتصاد خوانده‌اند نیز در این علم فعالیت می‌کنند و طرز فکر اقتصاد را وارد neurosience می‌کنند. به فرض، چگونه یک سیستم با داشتن منابع محدود می‌تواند کارهای درست انجام دهد؛ مثلا شما میخواهید bounded rationality را مطالعه کنید؛ مسئله‌ای که به چگونگی عملکرد بهینه‌ی مغز با وجود محدودیت‌هایی مثل انرژی یا مواد اولیه می‌پردازد.

حتی عده‌ای از طریق زبان شناسی وارد neurosience می‌شوند. یکی از زیبایی‌های این رشته این است که شما با هر رشته‌ای می‌توانید وارد آن شوید و مسیرهای بسیار متفاوتی به آن وجود دارد.

مثلا در زمانی که من neurosience را شروع کردم، در آزمایشگاهی که بودم سه نفر از برق و کامپیوتر، دو نفر از زیست شناسی و رشته‌های تجربی مثل پزشکی بودند، فردی از زبان شناسی داشتیم و در کل رشته‌ای بسیار جالب و تطبیق‌پذیر است که از تمام رشته‌ها امکان فعالیت در آن وجود دارد.

سوال ۶: به نظر شما کارهایی که در سطح جهان انجام می‌شود

(مانند Neuralink)، چه تاثیری در حوزه یادگیری ماشین و هوش مصنوعی می‌گذارد؟

تاثیر این موضوع در طولانی مدت نمایان می‌شود و تاثیر مستقیم آن را نمی‌توان دید. کارهایی که Neuralink انجام داده و تکنولوژی‌ای که ارائه کرده و مقاله‌هایی در این مورد چاپ شده، قبلا وجود داشته است؛ کاری که Neuralink کرد، این بود که افراد را دور هم جمع کرده است. دستاورد بزرگ Neuralink، صرفاً درست کردن این تکنولوژی برای ارتباط مغز و کامپیوتر نیست؛ بلکه درست کردن آن به صورت امن است. اینکه چطور آن‌ها را درست کنند که قابل هک نباشد، با بدن واکنش نامناسب ندهد و بعدا بتوان آن را آپدیت کرد. این دست مسائل، مهندسی و امنیتی است که احتیاج به کمپانی بزرگ و سرمایه زیاد برای حل شدن دارد؛ ایده‌ها را به راه حل‌های کاربردی و قابل استفاده برای درمان بیماری‌های مختلف یا بهبود دادن افرادی که مشکلات Neurodegeneration یا روانی دارند یا قطع نخاع هستند، تبدیل می‌کند. (انواع مشکلاتی که می‌تواند برای بافت عصبی پیش بیاید.)

اینکه چه تاثیری در هوش مصنوعی می‌گذارد، برنامه ای طولانی مدت و برای بیش از ده، پانزده سال آینده است. ابتدا باید برای افرادی که نیاز بیشتری به این تکنولوژی دارند (برای ادامه زندگی یا داشتن زندگی نرمال‌تر) استفاده شود؛ بعد از اینکه اطمینان حاصل کردیم که عوارض جانبی زیادی ندارد و امن است، افراد معمولی هم از آن استفاده کنند. از این طریق می‌توانیم بهتر درک کنیم که سیستم عصبی چگونه کار می‌کند و با این دید برای گسترش دادن هوش مصنوعی استفاده کنیم. در مجموع، الهام گرفتن از مغز برای ساختن هوش مصنوعی الزاما تنها راه درست فکر کردن به هوش مصنوعی نیست بلکه فقط یکی از راه‌هاست. از seymour papert می‌پرسند که به‌نظرتان کامپیوترها می‌توانند فکر کنند؟ و او پاسخ می‌دهد:«این سوال که کامپیوترها می‌توانند فکر کنند مثل این سوال است که آیا زیردریایی‌ها می‌توانند شنا کنند یا نه.»

وقتی یک ماشین درست می‌کنید، الزامی ندارد که دقیقا از همان فعالیت‌های زیستی استفاده کنید و آن‌ها را عینا کپی کنید. اگر نگاه کنید می‌بینید وقتی ماهی شنا می‌کند، صرفاً از قوانین فیزیک و مکانیک سیالات برای حرکت خودش در آب استفاده می‌کند. اگر ما مکانیک سیالات را خوب درک کرده باشیم، می‌توانیم چیزی بسازیم که حتی از آن ماهی هم بهینه‌تر باشد، بهتر حرکت کند، به انسان‌ها کمک کند و قابلیت‌های بیشتری داشته باشد.

همین دید را درباره هوش مصنوعی هم دارم، یعنی چیزی که ما باید درک کنیم، بیشتر از درک اینکه مغز چیست، راه‌های کلی مسئله حل کردن است. چگونه می‌توان یک مسئله کلی را حل، از داده‌ها استفاده و آن را به یک اپلیکیشن کاربردی تبدیل کنیم. این پایه هوش مصنوعی و learning theory است که ما در آن ضعیف هستیم. مغز می‌تواند برای ما الهام بخش باشد ولی هوش مصنوعی نهایی قرار نیست عینا شبیه مغز باشد. بسیاری از ساختارهای مغز به این موضوع وابسته است که از انرژی به نحو احسن استفاده شود، از فیزیک خاصی مانند کانال‌های یونی، Action Potential و Neuro Transmitter استفاده شود که برای آن بهینه شده‌است. ولی وقتی می‌خواهیم یک مغز با استفاده از ترانزیستور بسازیم، دیگر آنچنان این محدودیت‌ها را نداریم. انرژی برای مغز فوق‌العاده مهم است ولی در کار صنعتی، ما توانیم انرژی بخریم.

برای همین، نباید همه تمرکزمان را روی کپی کردن مغز بگذاریم؛ این که چگونه از نگاه کردن به دنیا دانش کسب کنیم و آن دانش را به کاربرد تبدیل کنیم، مسئله اصلی هوش مصنوعی است که لزوماً ربطی به neurosience ندارد و موضوعی ریاضی‌تر و عمیق‌تر است.

سوال ۷: به عنوان یک محقق که تجربه کاری در شرکت‌های بزرگ را دارید می‌توانید وظایف و کارهایی که باید به عنوان یک محقق یادگیری ماشین انجام دهید را بفرمایید؟

شباهت زیادی به دانشجو دکترا بودن دارد. برای مثال، یک دانشجوی دکترا با خواندن مقاله پروژه را تعریف می‌کند تا چیزی را بسازد یا یک مسئله را بهتر کند. وظیفه اول یک محقق این است که بررسی کند چه کارهایی انجام شده‌است که دوباره انجام نشود و دلیل و منطقی هم که در آن بوده است را متوجه شود. کاری که من در اپل انجام می‌دهم به همین صورت است. تفاوت زیادی بین یک محقق و دانشجوی دکتری وجود ندارد. تفاوت در راحتی کار در اپل است. به این صورت که گروهی از افراد هستند که وظیفه دارند به من کمک کنند؛ ولی در دکترا این طور نیست و باید از صفر تا صد کارها را دانشجو به تنهایی انجام دهد. در اپل یک نفر هست که Data Pipeline من را درست می‌کند و داده را پردازش کرده و آماده می‌کند و تحویل من می‌دهد. شخص دیگری نتایج را نمایش می‌دهد و من راحت‌تر می‌توانم روی هوش مصنوعی تمرکز کنم.

در کل کاری که من در اپل انجام می‌دهم فقط هوش مصنوعی نیست. ترکیب هوش مصنوعی و HCI است. به این صورت که من فکر می‌کنم که هوش مصنوعی چگونه می تواند با انسان رابطه برقرار کند یا این که چگونه ارتباط انسان و کامپیوتر را بهبود ببخشم. در کل شباهت زیادی با کار دانشگاهی دارد با این تفاوت که راحت‌تر است، حمایت خیلی بیشتر و پول بیشتری هم دارد. به گونه‌ای مثل پست دکترا می باشد یا به تعبیری راحت‌تر است. لازم نیست grant بنویسیم و به دلیل بودجه بی‌نهایتی که کمپانی‌ها در اختیار دارند، مشکلاتی که در دانشگاه وجود دارد را نخواهیم داشت. بنابراین می‌توان پروژه‌های

بیشتری را انجام داد.

سوال ۸: برای استخدام در این پوزیشن چه مهارت‌هایی لازم است و شرکت‌های بزرگ چه مهارت‌هایی را بررسی می‌کنند؟

چند نوع مهارت لازم است. مهم‌ترین آن‌ها Soft Skill یا مهارت‌های ارتباطی است.به این ترتیب که شما برای کار کردن با یک تیم چقدر مناسب هستید، چگونه با بقیه افراد بحث می‌کنید و منظور خود را چگونه می‌رسانید، آیا توانایی نوشتن شما خوب هست یا خیر.

شما شخصی را در نظر بگیرید که از نظر ریاضی نابغه باشد ولی مهارت‌های ذکر شده را نداشته باشد؛ در این صورت شرایط حضور در یک شرکت بزرگ را نخواهد داشت. پس مهارت‌های اجتماعی از موارد بسیار مهم است.

مورد بعدی که اهمیت دارد این است که شخص بتواند مسئله را به صورت مستقل طراحی کند و فعال باشد و لزومی نداشته باشد که دائم به او بگویند که چه کاری انجام بدهد. به عنوان یک محقق مهم‌ترین ویژگی فعال بودن‌است؛ زیرا در این زمینه کاری، مهارت او از همه بیشتر است و شخص دیگری نمی تواند بهتر از او مسئله درست برای حل کردن را طراحی کند، پیشرفت‌های خوب در این زمینه را تشخیص دهد و با استفاده از آن‌ها محصول تولید نماید.

مورد دیگر این که باید قابلیت تصمیم‌گیری داشته باشد و تعیین کند چه مواردی مهم هستند و چه مواردی مهم نیستند. اولویت‌بندی مناسبی داشته باشد و همه این مسائل بستگی به مقدار دانش شخص دارد. معمولاً این شرکت‌ها سعی می‌کند شخصی را استخدام کنند که قبلاً مقاله چاپ کرده‌است، در کنفرانس ها شرکت کرده باشد و خودش پروژه‌های جدید انجام داده باشد. یکی از مواردی که خیلی به استخدام من کمک کرد انجام چند پروژه بود که قبلاً کسی انجام نداده بود و این برای شرکت‌ها خیلی جالب بود که شخص این کارها را انجام داده است؛ حتی از تعداد مقالات چاپ شده نیز برای آن‌ها مهمتر بود. این که خود من یک مسئله جدید طرح کرده‌ام و برای آن وقت گذاشته‌ام برای آن‌ها مهمتر از این بود که آیا در کنفرانس‌ها مقاله ارائه کردم یا خیر.

در کنار این موارد، یک ویژگی مورد توجه شرکت‌ها، مهربان بودن و خوش برخورد بودن شخص می باشد. من جدا از اپل، وقتی در MIT دانشجو می‌گرفتند، چند بار عضو کمیته‌ای بودم که رزومه‌ها را می‌خواندیم و دانشجوها را گزینش می کردیم و به استاد معرفی می‌شدند. این نکته برای ما مهم بود که شخص مهربان باشد و بتوان با او دوست شد، نزدیک شد و در نهایت با او کار کرد. این موارد را در مصاحبه با شخص و معاشرت با او بررسی می‌کردند و در نهایت شرایطی را ایجاد می‌کردند که مشخص شود آیا فردی هست که بتوان با او کار کرد یا ادامه همکاری با او سخت است.

سوال ۹: به عنوان کسی که در دانشگاه امیرکبیر، در کانادا و آمریکا تحصیل و کار کرده‌اید، موقعیت‌های کاری و تحصیلی این کشورها را چگونه با هم مقایسه می‌نمایید و از نظر کیفی و کمی چگونه ارزیابی می‌نمایید؟

به دلیل اقامت یک‌ساله‌ی من در کانادا و شناخت کمتر از کانادا مقایسه بین کانادا و آمریکا خیلی مناسب نیست و در مورد انگلیس که دو سال در آن زندگی کردم اطلاعات بیشتری دارم. ولی در کل منابع مالی و فرصت‌های کاری در آمریکا بیشتر از جاهای دیگر دنیا است. مقدار پولی که در هر زمینه‌ای در آمریکا صرف می‌شود به مراتب بیشتر از کانادا است.

البته استثناهایی هم وجود دارد. در حال حاضر دولت کانادا به شدت در زمینه هوش مصنوعی سرمایه‌گذاری می‌کند. اگر شما یک محقق در زمینه هوش مصنوعی باشید آمریکا و کانادا خیلی تفاوتی ندارند ولی تعداد مشاغل در آمریکا بیشتر است. به عنوان مثال، شهر کمبریج حدود ۵۰ هزار استارتاپ دارد. عده بسیاری از آن‌ها در زمینه هوش مصنوعی فعالیت می‌کنند و در ایالت‌های دیگر آمریکا باز هم بیشتر است. درواقع آمریکا به خاطر جمعیت ۱۰ برابری نسبت به کانادا و شرایط رقابتی و همچنین آدم های عجیب و جالبی که از فعالیت‌های آن‌ها تعجب می‌کنید، این شرایط را دارد و فرصت‌های بیشتری نسبت به کانادا در اختیار شما می‌گذارد.

ولی در کنار تمام این خوبی‌ها، بدی‌هایی نیز دارد. در کل آرامش آمریکا کمتر از کانادا است و استرس کار در آمریکا بیشتر است. در آمریکا خدمات پزشکی برای کسی که تازه‌وارد باشد و پول‌دار نباشد خیلی خوب نیست؛ ولی برای متمولین مناسب می باشد. از نظر اجتماعی مردم کانادا منعطف‌تر هستند، به ملیت ایرانی واکنش نشان نمی‌دهند و شما یک فرد معمولی هستید و کاری به سیاست ندارند.

ولی در آمریکا به دلیل روابط ایران و آمریکا، افراد غیرتحصیل‌کرده به نحوه‌ی دیگری با شما رفتار می‌کنند؛ در صورتی که در کانادا این موارد خیلی کمتر است. این هم یک مقیاس برای شما می تواند باشد.

مثلاً آمریکا کشوری است که ۷۰ میلیون نفر در آن به دونالد ترامپ رای دادند؛ یعنی شما باید این را در نظر داشته باشید که نصف جمعیت این کشور عقاید تندتری نسبت به ایران و برخی کشورها دارند. در صورتی که در کانادا این شرایط وجود ندارد. کلاً کانادا این دیدگاه را دارد که فردی که وارد این کشور شد فرهنگ خود را هم می‌تواند بیاورد و همه فرهنگ‌ها را می‌پذیرند. مثلاً جشن سال نوی چینی را در کانادا می‌بینید و مهاجرانی که با همان فرهنگ خودشان لباس می‌پوشند و مردم نیز به آن‌ها احترام می‌گذارند. در آمریکا این‌گونه نیست. شخصی که وارد آمریکا می‌شود باید فرهنگ خود را کنار گذاشته و برای این که

کاملاً مورد قبول جامعه باشد باید فرهنگ آمریکایی را یاد بگیرد.

در کانادا ایرانی‌ها در مجلس هستند و به مرور وارد دولت می‌شوند و خیلی رشد کرده‌اند. ایرانیانی را می‌بینید که کارآفرین های موفقی هستند و وام‌های خیلی بزرگی از دولت کانادا گرفته‌اند.

در آمریکا، قدرت دست سفید پوست‌های آمریکایی است و شما برای رشد کردن باید به این افراد نزدیک شوید.به عنوان یک ایرانی به راحتی و سریع نمی‌توانید قدرت پیدا کنید؛ البته در جاهای مختلف می‌توانید استخدام شوید و کار پیدا کنید و به عنوان یک نیروی فنی،یک مهندس خوب یا یک ریاضی‌دان خوب پذیرفته می‌شوید و حقوق خوبی دریافت می‌کنید؛ ولی این که بتوانید رئیس و CEO شوید، برای ایرانی‌ها خیلی سخت‌تر است.

این‌ها شاید از دید من این‌گونه باشد و بر اساس مواردی که خودم دیده‌ام، می‌گویم. هیچ‌گاه آمار نگرفته‌ام و شاید اگر آمار دقیقی بگیرید این‌گونه نباشد؛ ولی این چیزی است که من برداشت کرده‌ام.

Books Recommendations

معرفی کتاب

عصر در هم تنیدگی، وقتی فیزیک کوانتومی دوباره متولد شد

فاطمه عبدالحی

دنیای کوانتوم، سرشار از هیجان، شگفتی و البته چالش است و فهم ما از این نظریه، کامل نمی‌شود. از مبتدی‌ترین انسان‌های این حوزه تا باهوش‌ترین دانشمندانی که بنیان‌گذاران آن بوده‌اند، هیچ‌گاه از مطالعه‌ی بیشتر در مورد آن بی‌نیاز نشده‌اند. درست همانطور که نیلز بور، یکی از نوابغ کوانتومی می‌گوید: «اگر کسی گفت فیزیک کوانتوم را فهمیده، پس چیزی نفهمیده است». یکی از کتاب‌های این حوزه که به یکی از موضوعات مهم در کوانتوم اشاره می‌کند «عصر درهم‌تنیدگی، وقتی فیزیک کوانتومی دوباره متولد شد» نوشته لوئیس گیلدر است. این کتاب از توضیح مفهوم درهم‌تنیدگی آغاز شده و به شما می‌گوید چه‌طور دو ذره که میلیون‌ها سال یا حتی میلیاردها سال از هم فاصله دارند، به طرز اسرارآمیزی با هم ارتباط می‌گیرند و بر هم تاثیر می‌گذارند. در اصل، هر چیزی یا هر اتفاقی که برای یک ذره پیش می‌آید بی‌واسطه و بدون درنگ سبب تغییر و تحول در ذره دیگر می‌شود. در رابطه میان آدمیان و در ضمیر ناخودآگاه آدمی نیز چنین درهم‌تنیدگی‌ای به کرات دیده می‌شود. درهم‌تنیدگی میان دوتن، میان دو انسان و میان دو ذهن که یک‌باره همه‌ی میل و نیاز آدمی را به میل دیگری بدل می‌کند. شناخت و بینشی که از درهم‌تنیدگی میان ذرات حاصل می‌کنیم را می‌توان راه‌گشا و نشانه‌ای بر راهبردهای عجیب و عاشقانه‌ای که در جهان هستی به عنوان یک کلیت غیرقابل‌شناخت وجود دارد دانست. جهانی که دیگر ناظران و مشاهده‌گرانش بیرون از ماجرا نیستند؛ هر مشاهده‌اش بر نتیجه‌ی جهان اثربخش است و هر نگاهی بخشی از این قلمرو بزرگ به حساب می‌آید. نگاه‌هایی که فهمش ممکن است سنگین باشد اما گیلدر با زیرکی تمام، ازمایش‌ها و روند توسعه‌ی این شاخه را در دیالوگ‌هایی که بین شخصیت‌های اصلی رد و بدل می‌شود بیان می‌کند و برای فهم بهتر، از نامه و دست نوشته‌هایشان نیز کمک می‌گیرد. این کتاب یک فانوس کوچک برای ورودتان به دنیای غریب و تاریکِ کوانتومِ درهم‌تنیدگی است. دنیایی که شاید در انتهایش با یک کرم‌چاله، به نقطه‌ی آغازین خود برگردید.

مغازه‌ی خودکشی

فاطمه عبدالحی

« اگر در زندگی‌تان شکست خورده‌اید، در مرگ موفق باشید» این جمله، شعار مغازه‌ای است در ناکجاآباد! جایی در آینده دور، که زندگی چشمانش را بسته و نومیدی در پوست و گوشت مردم نفوذ کرده. جایی‌که رهبرانش دیوانه و سیاه‌اند. اخبار، لبریز از خبرهای فاجعه‌آمیز و تلخ است و دیگر گلی نمی‌روید. در چنین جایی که مردم یا خودکشی می‌کنند یا حداقل به آن فکر می‌کنند، خانواده تواچ مغازه‌ای را می‌گردانند که به افراد کمک می‌کند تا صحیح و سالم به مقصد خودشان برسند؛ خودکشی! مغازه‌ای است که دست خالی از آن برنخواهید گشت. انواع سلاح‌ها، سم‌های لمس‌کردنی و بوئیدنی، طناب‌های محکم و ابزارهای خلاق دست‌ساخته، همان تردید کوچک‌تان را از بین می‌برد و شما را به وجد مرگ می‌آورد تا این بند نازک سیاه زندگی را ببرید. و به قول مرلین «مرگ پشت و پناهتون باشه». اما با آمدن فرزند سوم خانواده، پسرکی خنده‌رو و امیدوار، همه چیز دست‌خوش تغییر می‌شود. خانواده تواچ فکر می‌کنند آلن هم مثل پسر و دختر دیگرشان، ونسان و مرلین ، قرار است افسرده باشد. تمام فکر و ذکرش به دنبال خودکشی یا حداقل به دنبال ابداع روش‌های جدید و تضمینی برای مرگ باشد. ولی پسرک داستان ما کسی است که همیشه نیمه‌ی پر لیوان را می‌بیند. در بدترین اتفاقات، نکته‌های مثبت به چشمش می‌آید. نگاهی پر شور و شوق به اطرافش دارد و وقتی در مدرسه از او می‌پرسند که چه کسانی خودکشی می‌کنند؟ می گوید: « آدم‌های شاکی»، آدم‌های شاکی که تنهایند و یا شاید کسی را ندارند تا علت بودن‌شان را مرور کند. کسانی که فکر می‌کردند زندگی محل نبرد است و حالا آن‌ها بدون هیچ سلاحی باخته‌اند. تاب و تحمل این فضای مضطرب‌گونه را ندارند یا شاید شجاعت دیدن نیمه پر لیوان را. به قول آقای تواچ: «ما همیشه با محصولات طبیعی و حیات وحش مشکل داشتیم. از قورباغه‌های طلایی بگیر تا افعی و عنکبوت سیاه! می‌دونید چیه؟ مشکل اینه که مردم به قدری تنهان که حتی وقتی این موجودات زهردار رو هم به اون‌ها می‌فروشیم، باز جذبشون می‌شن. عجیب اینکه این موجودات هم همین حس رو دارند و نیششون نمی‌زنند».

مردم در این شهر آن‌قدر تنها و افسرده‌اند که خودشان هم از خودشان فراری‌اند و به مغازه خودکشی پناه می‌آورند تا اگر زندگی حقیرانه‌ای داشتند، مرگ باشکوهی را برای خود رقم بزنند. ولی آلن، پسر کوچک خانواده، انگار قرار است که معنای تازه‌ای به مغازه خودکشی ببخشد و هیچ جوره کوتاه‌بیا نیست. «آلن، آخه چند بار باید بهت بگم؟ وقتی مشتری‌هامون از مغازه خرید می‌کنن، بهشون نمی‌گیم به‌زودی می‌بینمت. ما باهاشون وداع می‌کنیم. چون هیچ وقت برنمی‌گردن. آخه کِی این رو تو کله‌ت فرو می‌کنی؟»

ژان تولی، نویسنده کتاب، به زیبایی هر چه تمام‌تر توانسته که کمدی سیاهی را برای‌مان بنویسید. از تاریک‌ترین قسمت وجودمان حرف بزند که گم شده است. از تلخ‌ترین وجه پنهان جهان اطراف‌مان بگوید و آینده‌ای از این حجم افسردگی که شاهدش هستیم به تصویر بکشد. ژان تولی، با طنز جذابی که در قلمش دارد به شما می‌گوید اگر در سیاهی افسردگی غرق بشوید، چه می‌شود. اهمییت زندگی کردن و زنده ماندن را بهتان نشان می‌دهد، خنده‌ها را برای‌تان پر شورتر و غم را برای‌تان کمرنگ‌تر می‌کند. به قول آلن: «زندگی همینه. اگر سخت بگیری، اونم بهت سخت می‌گذرونه. این ماییم که بهش ارزش می‌دیم. با همه کمبودهایی که این دنیا داره، زیبایی‌های خودش رو هم داره، نباید از زندگی انتظار زیادی داشته باشیم. نمی‌شه باهاش جنگید! بهتر اینه که نیمه پر لیوان رو ببینیم.» در اصل، دیدگاه ما می‌تواند یک زندگی دل‌فریب هیجان‌انگیز بسازد، یا در تاریکی نومیدی غرق‌مان کند. می‌توانیم در حال تماشای اخبار منفی باشیم، اما ظاهر زیبای گوینده خبر را ببینیم. به‌هرحال فراموش نکنیم که زنده بودن زمان می‌برد. از همه‌چیز بریدن هم زمان می‌برد.

داستان فضاسازی پیچیده و چندلایه‌ای دارد؛ گفت‌وگوها، رفتار و عادت‌های شخصیت‌ها و برخی ماجراهای داستان ماهیتی طنزآلود دارند. وقتی بعضی از جملات را می‌خوانید، نمی‌دانید از تلخی آن گریه کنید یا از روی ناباوری، قهقهه بزنید. نام اعضای خانواده یاد آور افراد سرشناسی در تاریخ است که خودکشی کرده‌اند. نام میشیمیا، پدر خانواده، یادآور یوکیو میشیماست. نویسنده ژاپنی که سه‌بار جایزه نوبل را برد و در آخر به روش هاراکیری خودکشی کرد. ونسان، پسر بزرگ خانواده، از ون‌گوگ الهام گرفته است. نقاش پر آوازه هلندی که به قلب خودش شلیک کرد. مرلین، تک دختر تواچ، برگرفته از مرلین مونرو است. بازیگری آمریکایی که به علت مصرف بیش از اندازه آرام‌بخش و خواب‌آور به خواب ابدی رفت. آلن، پسر کوچک میشیما و لوکریس، پژواکی از آلن تورینگ است. ریاضی دان نابغه انگلیسی که با گاز زدن سیب آغشته به سیانور به دیار باقی شتافت.

ترجمه روان احسان کرم‌ویسی، کار را برای‌تان دل‌انگیز می‌کند. انتخاب درست کلمات و ترکیب آن‌ها، عجیب به دل‌تان می‌نشیند. این کتاب فرانسوی، سی‌وچهار فصل دارد و تا به حال به بیست زبان دنیا ترجمه شده است. درضمن اولین کتابی است که از این نویسنده فرانسوی،

به فارسی ترجمه شده. مغازه خودکشی که در نشر چشمه به چاپ رسیده و کتابی صوتی با خوانش هوتن شکیبا هم دارد، یک کمدی سیاه تکان‌دهنده را برای‌تان روایت می‌کند. فکر می‌کنم با توجه به کم‌حجم بودنش، برای کسانی که دل‌شان می‌خواهند خودشان را به چالش بکشند و ذهن‌شان را تا مدت‌ها درگیر کنند، خواندنی باشد.

Entertainment
سرگرمی

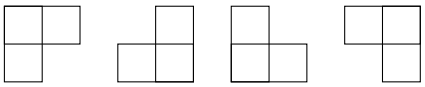
سرگرمی

امیرحسین رجبی و محمدرضا باطنی

پاسخ چالش سری گذشته (امیرحسین رجبی)

صورت مسئله چنین بود:

جدولی $m \times n$ (m سطر و n ستون) را با کاشی‌های زیر به گونه‌ای فرش کرده‌ایم که هر خانه جدول توسط دقیقاً یک کاشی پوشانده شود. ثابت کنید اگر تعداد کاشی‌های به کار رفته از نوع ۱ برابر a و همچنین تعداد کاشی‌های به کار رفته از نوع ۲ برابر b باشد، آنگاه تفاضل a و b بر ۳ بخش‌پذیر است.



۱ ۲ ۳ ۴

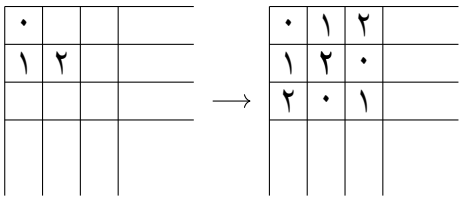
یک تکنیک برای حل چنین مسائلی که می‌خواهیم حکمی درباره تعداد کاشی‌ها بدهیم، وزن دهی است. یعنی می‌خواهیم کاشی‌ها علاوه بر تعداد خانه‌هایی که می‌پوشانند، نوعی ارزش دیگری نیز داشته باشند. چون می‌خواهیم تعداد کاشی های نوع اول و دوم را به پیمانه ۳ بررسی کنیم، وزن ها را ۰، ۱ و ۲ انتخاب می‌کنیم. جدول را به صورت زیر وزن دهی می‌کنیم: (به خانه‌های جدول، عدد نوشته شده روی آن را نسبت می‌دهیم.)

۰	۱	۲	...	۰	۱	۲
۰	۱	۲	...	۰	۱	۲
	⋮				⋮	
۰	۱	۲	...	۰	۱	۲

در یک کاشی کاری معتبر، mn بر ۳ بخشپذیر است (هر کاشی سه خانه را می‌پوشاند و کل خانه‌های جدول پوشانده می‌شوند.)؛ پس بدون کاسته شدن از کلیت مسئله، می‌توانیم فرض کنیم n بر ۳ بخش‌پذیر است. با این روش مشاهده می‌شود که مجموع وزن خانه‌های کاشی نوع اول به پیمانه ۳ برابر ۱، کاشی نوع دوم ۲، کاشی نوع سوم ۱ و کاشی نوع چهارم ۲ است. همچنین مجموع وزن خانه‌های جدول به پیمانه ۳، صفر است (تعداد خانه‌ها با وزن ۱ و ۲ برابرند.)؛ در نتیجه اگر تعداد کاشی‌های نوع سوم و چهارم به ترتیب c و d باشند، خواهیم داشت:

$$0 \overset{3}{\equiv} a + 2b + c + 2d \overset{3}{\equiv} a - b + c - d.$$

اما این نتیجه دل‌خواه ما را نمی‌دهد. وزن‌دهی را تغییر می‌دهیم. دوست داریم ضریب c و d در عبارت بالا برابر صفر شود (یعنی مجموع وزن خانه‌های کاشی نوع سوم و چهارم بر ۳ بخش‌پذیر باشد.)، همچنین ضریب a و b متمایز و ناصفر باشد؛ پس وزن دهی را این گونه شروع می‌کنیم:



به سادگی دیده می‌شود که این الگو باید ادامه پیدا کند؛ یعنی:

۰	۱	۲	۰	۱	۲	
۱	۲	۰	۱	۲	۰	
۲	۰	۱	۲	۰	۱	
۰	۱	۲	۰	۱	۲	
۱	۲	۰	۱	۲	۰	
۲	۰	۱	۲	۰	۱	

به وضوح دیده می‌شود که مجموع وزن خانه‌های کاشی نوع اول به پیمانه ۳، برابر ۲، کاشی نوع دوم ۱، کاشی نوع سوم صفر و کاشی نوع چهارم نیز صفر است و مجموع وزن خانه‌های جدول به پیمانه ۳، صفر است. پس:

$$0 \overset{3}{\equiv} 2a + b \overset{3}{\equiv} b - a.$$

چالش این سری: سودوکو کشنده (محمد رضا باطنی)

سودوکو کشنده (Killer sudoku) یک بازی معروف ژاپنی است که در سال ۱۹۹۰ به عنوان یکی از انواع دیگر سودوکو عادی معرفی شد. قوانین این بازی شامل قوانین سودوکوی عادی است؛ یعنی باید خانه‌های جدول به نحوی پر شوند که در هیچ سطر و ستونی عدد تکراری وجود نداشته باشد و همچنین در هیچ یک از ۹ مربع 3×3 مشخص‌شده نباید عدد تکراری ظاهر شود. اما در سودوکوی کشنده قانون جدیدی وجود دارد که آن را به شدت چالشی‌تر و جذاب‌تر می‌کند؛ اینکه جمع خانه‌هایی که دور آن‌ها با خطچین مشخص شده‌است باید برابر عددی شود که در گوشه بالا-چپ آن‌ها نوشته شده‌است. اکنون باید خانه‌های جدول به نحوی پر شوند که همه قوانین گفته شده در آن رعایت شده باشد.

راهنمایی: از آنجایی که در هر سطر، ستون و مربع 3×3 عدد تکراری وجود ندارد و فقط از اعداد ۱ تا ۹ استفاده می‌شود، پس جمع اعداد هر کدام برابر $9 + \cdots + 2 + 1$ می‌شود که برابر ۴۵ است. از این تساوی‌ها استفاده کنید.

19			3		16	17		6
11		12	20			9		
7	4		7				13	
		19		24	22			
						23		
25					6		9	12
12		10	20	18		5		
14	9						10	
				5		18		

سطح متوسط

16	4	5		18			22	
		14	19		11	15	10	
	27		15					
11		24					21	9
				14				
			10	24		19		
18		10			10		16	
	14							12
		6		11				

سطح دشوار

"علم با حقایق ساخته شده است، همانطور
که یک خانه از سنگ اما مجموعه‌ای از حقایق
همانقدر علم است که توده‌ای از سنگ، خانه!"

هنری پوانکاره

دانشکده
ریاضی و علوم کامپیوتر

انجمن علمی دانشجویی دانشکده ریاضی و علوم کامپیوتر دانشگاه صنعتی امیرکبیر

دانشگاه صنعتی امیرکبیر
اداره انجمن های علمی دانشجویی