مقدمه‌ای بر شبکه‌های عصبی(قسمت اول)

سید سروش هاشمی

چکیده

در این مقاله ابتدا مفاهیم اولیه یادگیری ماشین به ساده‌ترین شکل ممکن مطرح می‌شوند. سپس به بررسی شبکه‌های عصبی واقعی پرداخته می‌شود و تلاش می‌شود با الهام گرفتن از آن‌ها شبکه‌های عصبی مصنوعی طراحی کنیم. در نهایت نیز برخی انواع پیچیده شبکه‌های عصبی معرفی می‌شوند و تعدادی از کاربردهای شگفت‌انگیز شبکه‌های عصبی بیان می‌شوند.

۱ مقدمه

در طول تاریخ ریاضی‌دانان و دانشمندان بسیاری سعی کردند کارهایی که مختص انسان‌ها تلقی می‌شد را در ماشین‌ها شبیه سازی کنند یا به صورت قابل دفاعی به آن‌ها نسبت دهند. به طور مثال Alan Turing در سال ۱۹۳۷ ماشینی ساخت که توانایی استنتاج و تصمیم گیری بنابر استنتاج‌های خود را داشت(به تعبیری می‌توانست فکر کند و تصمیم بگیرد). همین طور John von Neumann در سال ۱۹۴۹ برنامه‌ای نوشت که توانایی بازنویسی کد خود از نو را داشت(به تعبیری می‌توانست مشابه ویروس‌ها تولید مثل کند). همچنین موفقیت‌های چشمگیری در افزایش درک ماشین‌ها از دنیای اطراف و ایجاد حواس پنجگانه در آن‌ها به دست آمده است. علاوه بر این رویکردهای عملی، ایده‌پردازان و فیلسوفان بسیاری درباره شباهت ماشین‌ها و انسان‌ها سخن گفته‌اند که یکی از جالب‌ترین آن‌ها ایده Samuel Butler درباره اعمال شدن نظریه داروین روی ماشین‌هاست. او معتقد بود همان طور که نظریه داروین برای موجودات زنده درست است، برای ماشین‌ها نیز صدق می‌کند و روزی می‌رسد که ماشین‌ها هویت(cognition) پیدا می‌کنند و کم کم جایگزین انسان‌ها می‌شوند.

در بین تمام این تلاش‌ها و ایده‌ها، یکی از پرچالش‌ترین و پرکاربردترین موضوعات، «یادگیری ماشین» بود. آیا یک ماشین می‌تواند چیزی بیاموزد؟ آیا می‌تواند همانند انسان تجربه کسب کند و اشتباهاتش را اصلاح کند؟ آیا یک ماشین می‌تواند ماشین‌های دیگر را آموزش دهد؟ آیا می‌توان برای یک ماموریت خطرناک(مانند نجات انسان‌ها از حریق، یا از کار انداختن یک بمب) تیمی از ماشین‌ها تربیت کرد که هر یک در زمینه‌ای متخصص باشند و بتوانند ماموریت را بدون به خطر افتادن جان انسان‌ها انجام دهند؟ «یادگیری» می‌تواند ماشین‌ها را به موجوداتی به مراتب کاربردی‌تر تبدیل کند.

۲ مفاهیم اولیه یادگیری ماشین

در مرحله اول باید به یک مدل ریاضی برای «یادگیری» دست یابیم. یکی از مدل‌های بسیار ساده و البته کاربردی به این صورت است:

«تابعی مانند $f : X \mapsto Y$ داریم که لزوما محاسبه پذیر نیست یا فرمولی برای محاسبه آن نداریم. مثلا یافتن نام یک

انسان با دیدن چهره او. تعدادی ورودی و خروجی نمونه مانند $(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\dots,(x^{(n)},y^{(n)})$ داریم. می‌خواهیم با کمک این نمونه‌ها تابعی مانند $\hat{f}:X\mapsto Y$ بیابیم که علاوه بر نمونه‌های موجود بتواند برای نمونه‌های جدید نیز خوب کار کند.»

علت این نوع از اندیس گذاری برای نمونه‌ها این است که خود x,y ممکن است ساختاری پیچیده‌تر از یک عدد داشته باشند. مثلا یک بردار یا ماتریسی از اعداد باشند. در این صورت نیاز داریم برای اشاره به درایه‌های ماتریس از اندیس‌های بیشتری استفاده کنیم. برای جلوگیری از ابهام، در این مقاله همواره اندیس‌های بالای متغیر نشانه اندیس نمونه است و اندیس پایین متغیر نشانه اندیس اجزاء (مثلا درایه‌ها) آن نمونه از آن متغیر. مثلا اگر x ساختار ماتریسی داشته باشد، $x^{(17)}_{3,5}$ نشان دهنده درایه سطر ۳ و ستون ۵ از نمونه شماره ۱۷ است.

احتمالا با دیدن این مدل تکنیک‌هایی مانند regression در آمار و علوم دیگر به ذهنتان خطور کرده است. با وجود سادگی، این تکنیک‌ها نیز جزو راه حل‌های این سوال محسوب می‌شوند. بنابراین شما می‌توانید از یک رگرسیون ساده به عنوان یک ابزار «یادگیری ماشین» استفاده کنید.

حال چگونه میزان شباهت یک تابع $\hat{f}$ با تابع f را بسنجیم؟

۱.۲ تابع loss

یک تعریف ساده و البته بسیار الهام بخش برای «میزان تفاوت تابع $\hat{f}$ و f » به این صورت است:

$$loss(f,\hat{f}):=\int_{x\in D_f} d(f(x),\hat{f}(x))$$

که در آن

$$d(f(x),\hat{f}(x))$$

تفاوت خروجی تابع f و $\hat{f}$ به ازای ورودی x است که بسته به نوع خروجی تابع f می‌توان آن را به صورت‌های مختلف تعریف کرد. به طور مثال اگر خروجی تابع f یک ماتریس ستونی باشد، می‌توان این تابع را به صورت

$$d(f(x),\hat{f}(x)):=(\hat{f}(x)-f(x))^T(\hat{f}(x)-f(x))$$

تعریف کرد.

حال اگر بخواهیم به ازای یک $\hat{f}$ داده شده، مقدار $loss(f,\hat{f})$ را محاسبه کنیم، چه کنیم؟ برای محاسبه این مقدار یک مشکل اساسی داریم. مشکل این است که تابع f را نداریم و نمی‌توانیم به ازای هر ورودی دلخواه $x\in D_f$ مقدار $f(x)$ را محاسبه کنیم و در فرمول تابع loss از آن استفاده کنیم. راه حل این مشکل، تخمین زدن تابع loss با مقدار آن در نقاط نمونه $(x_1,y_1),\dots,(x_n,y_n)$ است. بنابراین فرض می‌کنیم

$$loss(f,\hat{f})\approx\sum_{(x,y)\in\text{samples}} d(y,\hat{f}(x)) \qquad (۱)$$

تقریب خوبی برای تابع loss است. (البته برای این که این تقریب قابل قبول باشد باید نمونه‌ها و روش نمونه گیری چند خاصیت خوب، مثل عادلانه بودن نمونه گیری، را داشته باشند زیرا در غیاب این شروط بسیاری از نتایج آتی به خطر می‌افتند)

بنابراین سوال اولیه تبدیل به این سوال شد: «تابع $\hat{f}$ را بیابید که مقدار (تخمین) تابع loss را کمینه کند». این یک سوال بهینه سازی جالب و بسیار شایع در بسیاری از شاخه‌های علمی است.

۲.۲ مدل

چون در این سوال محدودیتی برای $\hat{f}$ قائل نشدیم، می‌توانیم به هر شکلی که بخواهیم این تابع را تعریف کنیم. بنابراین به جای این که به دنبال یک ضابطه برای این تابع بگردیم، این تابع را به صورت جفت‌های مرتب تعریف می‌کنیم به طوری که

$$(x^{(1)},y^{(1)}),\dots,(x^{(n)},y^{(n)})\in\hat{f}$$

باشد و مقدار $\hat{f}$ برای سایر مقادیر دامنه را به دلخواه مقداردهی می‌کنیم. این تابع مقدار تخمین loss را صفر می‌کند، بنابراین بهترین تابع است. اما در تعریف این تابع می‌توانیم به ورودی‌هایی که در نمونه‌ها نیستند هر مقدار خروجی دلخواه را نسبت دهیم، پس لزوما خروجی تابع $\hat{f}$ و f در سایر نقاط دامنه مشابه نیستند حتی اگر تخمین loss صفر باشد. این مشکل به این خاطر پیش آمد که ما به دنبال هیچ الگو و رابطه‌ای بین ورودی‌ها و خروجی‌های نمونه نگشتیم و فقط سعی کردیم تخمین تابع loss را کمینه کنیم. برای

حل این مشکل، تابع $\hat{f}$ را به دسته‌ای از توابع محدود می‌کنیم که حدس می‌زنیم می‌توانند الگو طبیعی بین ورودی و خروجی را بیابند و از آن برای تخمین مقدار f در سایر اعضای دامنه نیز استفاده کنند. این دسته از توابع را «مدل» می‌گویند. مثلا در رگرسیون خطی، فرض می‌کنیم تابع $\hat{f}$ به صورت

$$\hat{f}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b = b + \sum_i w_i x_i$$

است پس مدل، تمام توابع خطی است. مدل می‌تواند خیلی ساده (مانند مثال قبل) یا بسیار پیچیده باشد. به عنوان یک مثال از یک مدل پیچیده می‌توان به مدلی که شامل توابعی به شکل زیر است اشاره کرد ($\mathbf{x}$ یک بردار ستونی با k درایه است) :

$$\begin{aligned}\hat{f}(\mathbf{x}) = {} & w_1 x_1 + \cdots + w_k x_k \\ & + w_{k+1} x_1 x_2 + w_{k+2} x_1 x_3 + \cdots + w_{\frac{k(k+1)}{2}} x_{n-1} x_n \\ & \vdots \\ & + w_{2^k-1} x_1 x_2 \dots x_k\end{aligned} \qquad (۲)$$

این تابع در واقع تمامی 2^k ترکیب k درایه ورودی خود را به صورت وزن‌دار با هم جمع می‌کند.

باید مدل را طوری بسازیم که شامل خود تابع f یا یک تخمین خوب از آن باشد. برای این کار از دانش اولیه خود درباره مسئله الهام می‌گیریم. به طور مثال اگر باور داشته باشیم که یک ترکیب خطی از ورودی، خروجی را تولید می‌کند، کافی است مدل را به گونه‌ای تعریف کنیم که فقط توابع خطی را شامل شود. البته تلاش‌های جالبی برای یافتن خانواده‌ای از توابع که می‌توانند هر تابعی را تخمین بزنند انجام شده که یکی جالب‌ترین آن‌ها قضیه universal approximation [۲] است که ثابت می‌کند یک شبکه عصبی با تنها یک لایه مخفی [1] می‌تواند دسته بسیار بسیار بزرگی از توابع شایع در مسائل بسیاری از حوزه‌ها را تخمین بزند.

حال که تابع loss را تعریف کردیم و تابع $\hat{f}$ را به مدل محدود کردیم، باید بهترین تابع موجود در مدل را بیابیم که تخمین loss را کمینه کند. به الگوریتمی که بهترین تابع (یا یک تابع خوب) از بین توابع موجود در مدل را برای تخمین f می‌یابد، «بهینه‌ساز» [2] گویند.

۳.۲ الگوریتم‌های بهینه‌ساز

درباره این الگوریتم‌ها ذکر چند مسئله خالی از لطف نیست. هر الگوریتم بهینه‌ساز برای دسته خاصی از مدل‌ها طراحی شده است. بعضی اوقات مدل طوری تعریف شده که یافتن بهترین تابع برای تخمین f ممکن است. در این حالت الگوریتم‌های مختلف بر سر زمان یافتن آن با هم رقابت می‌کنند. اما در بسیاری مواقع، گستره توابع مدل به اندازه‌ای بزرگ و ناهموار می‌شود که الگوریتمی برای یافتن بهترین تابع $\hat{f}$ در آن وجود ندارد. در این شرایط علاوه بر زمان اجرا، کیفیت توابع پیدا شده نیز در مقایسه دو الگوریتم مورد نظر قرار می‌گیرند.

به طور خلاصه هر الگوریتم یادگیری ماشین از ۴ بخش تشکیل شده:

۱. داده‌های نمونه

۲. تعریف تابع loss

۳. مدل

۴. الگوریتم بهینه‌ساز

قبل از ورود به شبکه‌های عصبی، ذکر نکته‌ای دیگر درباره الگوریتم‌های یادگیری ماشین خالی از لطف نیست. فرض کنید ما ۱۰۰ عکس از چهره یک شخص در حالت‌های مختلف داریم و می‌خواهیم به یک ماشین آموزش دهیم که بتواند وجود یا عدم وجود چهره این شخص در عکس را تشخیص دهد. وقتی این ماشین آموزش داده شد، می‌خواهیم دقت آن را اندازه گیری کنیم. مثلا بگوییم این ماشین در ٪۹۰ مواقع درست کار می‌کند. برای اندازه گیری این دقت چه راه حلی دارید؟

[1] hidden
[2] optimizer

۴.۲ محاسبه دقت

یک راه ساده و البته الهام بخش برای اندازه گیری دقت، اینگونه است: «روی تمام اعضای دامنه، تابع $\hat{f}$ چند درصد جواب درست تولید می‌کند؟» اما برای محاسبه دقت با این تعریف، به خود تابع f نیاز داریم. مثلا در مثال عکس از چهره، باید بتوانیم تمام عکس‌های موجود را تولید کنیم و به ازای هر عکس تشخیص دهیم آیا چهره آن شخص در عکس وجود دارد یا نه و سپس نتیجه تشخیص خود را با خروجی تابع $\hat{f}$ مقایسه کنیم. این کار برای مسائلی که دامنه f در آن‌ها بزرگ است اصلا امکان پذیر نیست. حتی برای حالاتی که دامنه f خیلی بزرگ نیست، اصلا صرفه زمانی و اقتصادی ندارد. برای حل این مشکل مجددا سعی می‌کنیم این مقدار را تخمین بزنیم. برای تخمین زدن دقت، کافی است چند نمونه‌ی دیگر از دامنه به همراه خروجی تابع f در آن نمونه‌ها را داشته باشیم و فقط خروجی f و $\hat{f}$ را در آن نمونه‌ها مقایسه کرده و دقت را تخمین بزنیم. اگر برای محاسبه دقت از همان نمونه‌هایی که با آن‌ها $\hat{f}$ را یافته‌ایم استفاده کنیم قطعا تعداد پاسخ‌های غلط بسیار کمی خواهیم داشت (زیرا $\hat{f}$ طوری طراحی شده که تخمین تابع loss را کمینه کند) و در نتیجه دقت محاسبه شده بالا خواهد بود. اما اگر برای محاسبه دقت از داده‌های جدیدی استفاده کنیم، تخمین ما از دقت، مقدار واقعی‌تری را نشان می‌دهد. به همین خاطر قبل از شروع فاز یادگیری، معمولا داده‌های موجود را به نسبت ٪۸۰ به ٪۲۰ به دو دسته «داده‌های یادگیری» [3] و «داده‌های تست» [4] تقسیم می‌کنند و در فاز یادگیری فقط از داده‌های یادگیری و در فاز تست (محاسبه دقت) فقط از داده‌های تست استفاده می‌کنند. البته هیچ نکته خاصی در اعداد ۸۰ و ۲۰ وجود ندارد و شما می‌توانید به هر نسبتی که خودتان می‌پسندید داده‌هایتان را تقسیم کنید. البته در مواقعی که داده‌های کمی موجود است از ایده‌های دیگری استفاده می‌کنند.

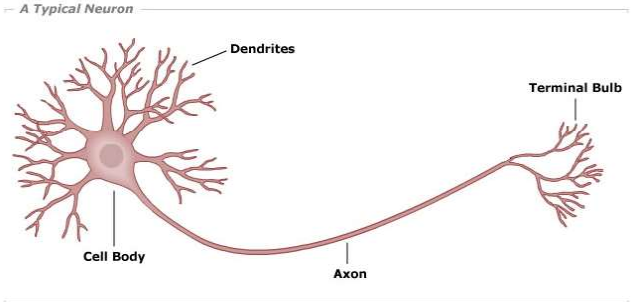
حال که جنبه‌های مختلف یک الگوریتم یادگیری ماشین را دیدیم به بررسی شبکه‌های عصبی می‌پردازیم.

۳ شبکه‌های عصبی

ابتدا کمی شبکه‌های عصبی بدن موجودات زنده را بررسی می‌کنیم. سپس تلاش می‌کنیم با الهام گرفتن از آن‌ها مدل ریاضی تولید کنیم که یک تابع را محاسبه می‌کند. نهایتا این مدل را به شبکه‌های عصبی واقعی پیچیده‌تر نزدیک می‌کنیم و مدل ریاضی تولید شده را ارتقا می‌دهیم.

۱.۳ شبکه‌های عصبی در علوم زیستی

شبکه عصبی در بدن یک موجود زنده از تعداد زیادی نورون تشکیل شده است که هر یک تعدادی سیگنال ورودی دریافت کرده و یک سیگنال خروجی تولید می‌کند. به تعبیری یک نورون تابعی از سیگنال‌های ورودی خود را به عنوان خروجی تولید می‌کند. خروجی هر نورون می‌تواند به تعداد دلخواهی نورون منتقل شود. (ورودی و خروجی نورون‌ها یک پالس الکتریکی است. در محاسبه خروجی نورون، تنها ولتاژ این پالس‌های الکتریکی تاثیرگذار است. می‌توان نورون را به صورت یک تابع از ولتاژ پالس‌های الکتریکی ورودی به ولتاژ پالس الکتریکی خروجی دید. پس اگر خروجی یک نورون را به صورت موازی به چند نورون ارسال کنیم ولتاژ ثابت می‌ماند و فقط جریان تغییر می‌کند) یک شبکه عصبی متشکل از تعداد زیادی نورون است که ورودی هر یک، خروجی تعداد دلخواهی نورون دیگر است. بنابراین یک شبکه عصبی معادل یک گراف جهت‌دار بزرگ از نورون‌هاست. ذکر این نکته جالب است که شبکه‌های عصبی با دور نیز مشاهده شده‌اند.

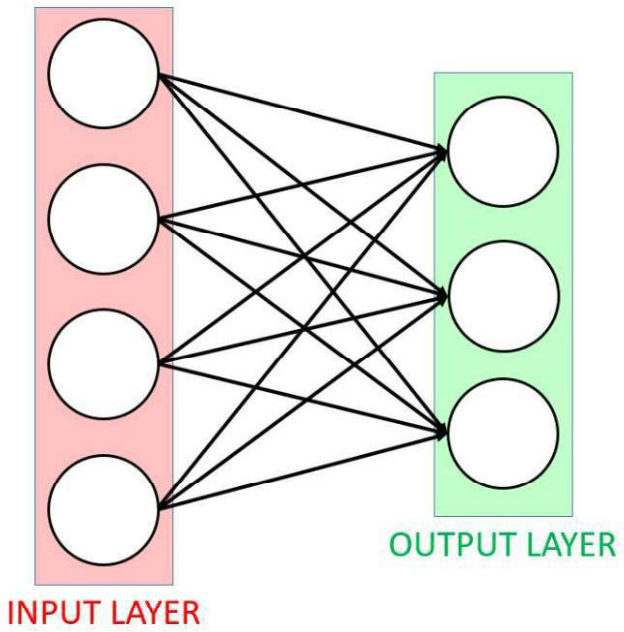


شکل ۱.۳

[3] train set
[4] test set

۲.۳ شبکه‌های عصبی بدون لایه مخفی

فرض کنید تابع f یک ماتریس ستونی با ۴ درایه را می‌گیرد و یک ماتریس ستونی با ۳ درایه تولید می‌کند. برای شبیه‌سازی این تابع با نورون‌ها می‌توانیم به ازای هر درایه خروجی تابع یک نورون در نظر بگیریم که تمام ۴ درایه ورودی به آن وصل هستند. هر نورون باید یک تابع از ۴ ورودی خود را حساب کند و خروجی دهد. برای ساده شدن مدل ریاضی فرض می‌کنیم هر نورون یک ترکیب خطی از ورودی‌های خود را محاسبه می‌کند. در این صورت مدل تبدیل به شکل زیر می‌شود. در این شکل برای هر یال یک «وزن» تعریف می‌کنیم که در واقع همان ضریب درایه متناظر آن در ورودی برای محاسبه درایه مورد نظر خروجی است. مثلا یالی که درایه اول ورودی را به درایه دوم خروجی وصل می‌کند وزن درایه اول در ترکیب خطی متناظر با درایه دوم خروجی است.



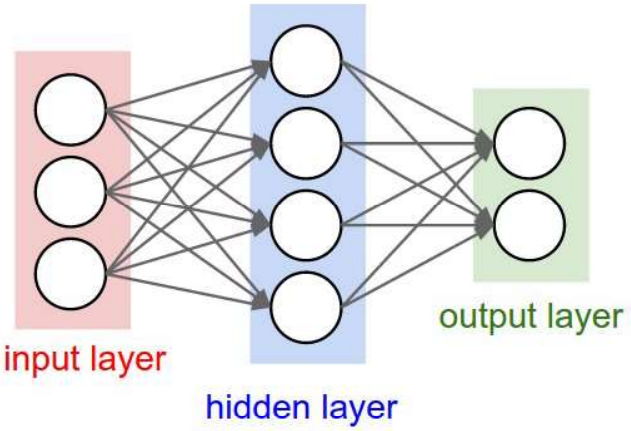
شکل ۲.۳

این مدل ریاضی به صورت جالب و ساده‌ای تبدیل به یک ضرب ماتریسی می‌شود. اگر ماتریس ورودی را $\mathbf{x}$ و ماتریس خروجی را $\mathbf{y}$ بگیریم می‌توان تعریف کرد:

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

که $\mathbf{W}$ یک ماتریس با ۳ سطر و ۴ ستون است که درایه‌های سطر i ام آن ضرایب ترکیب خطی متناظر با درایه i ام خروجی است.

در شبکه‌های عصبی واقعی، نورون‌ها ورودی خود را از تعدادی نورون می‌گیرند و خروجی خود را به تعدادی نورون دیگر می‌دهند. همچنین تابعی که نورون‌ها محاسبه می‌کنند که ترکیب خطی ساده نیست. در ادامه تلاش می‌کنیم مدل خود را کمی پیچیده‌تر کنیم تا این ساختار را شبیه سازی کنیم. برای این کار ابتدا فرض می‌کنیم یک لایه از نورون‌ها داریم که با محاسبه ترکیب خطی ورودی‌ها لایه‌ای میانی از اعداد تولید می‌کنند. سپس لایه‌ای دیگر از نورون‌ها، خروجی این لایه میانی را گرفته و با محاسبه تابعی از این اعداد، اعداد درایه‌های خروجی را تولید می‌کنند.



شکل ۲.۳

اگر تابعی که نورون‌های لایه دوم محاسبه می‌کنند نیز یک ترکیب خطی از ورودی‌های خود باشد، درایه‌های ماتریس خروجی کل این شبکه عصبی، ترکیب خطی ورودی‌های لایه اول آن می‌شود. پس اگر قصد پیچیده‌تر شدن مدل را داریم باید تابع دیگری برای این لایه از نورون‌ها انتخاب کنیم. فرض می‌کنیم نورون‌های موجود در لایه دوم این شبکه عصبی، همگی تابع غیر خطی و مشابه g را محاسبه می‌کنند. بنابراین مدل جبری این شبکه عصبی به صورت زیر است (این مدل برای شکل ۲.۳ است. در این مثال، $\mathbf{W}$ یک ماتریس با ۴ سطر و ۳ ستون است) :

$$\hat{f}(\mathbf{x}) = \begin{bmatrix} g(\mathbf{W}\mathbf{x}) \\ g(\mathbf{W}\mathbf{x}) \end{bmatrix} \qquad (۳)$$

ممکن است سوال کنید چون تابع g و ورودی $\mathbf{x}$ برای هر ۲ درایه ماتریس خروجی یکی است، آیا این ۳ مساوی نمی‌شوند؟ این بستگی به تعریف تابع g دارد. ممکن است تابع g طوری تعریف شود که وابسته به اندیس درایه خروجی خود باشد. مثلا نورونی که درایه دوم ماتریس خروجی را تولید می‌کند تابعی از درایه سوم و چهارم لایه میانی باشد و نورونی که درایه اول ماتریس خروجی را تولید می‌کند تابعی از دو درایه اول لایه میانی باشد. همچنین ممکن است تابع g پارامتر دیگری علاوه بر x داشته باشد که این پارامتر برای دو نورون ماتریس خروجی متفاوت باشد. همچنین روش‌هاش دیگری برای تمایز دادن بین توابع این دو درایه وجود دارد.

می‌توان ایده اضافه کردن لایه میانی را بیش از یکبار استفاده کرد و شبکه‌های عصبی با لایه‌های میانی بیشتر ساخت.

۳.۳ شبکه‌های عصبی عمیق

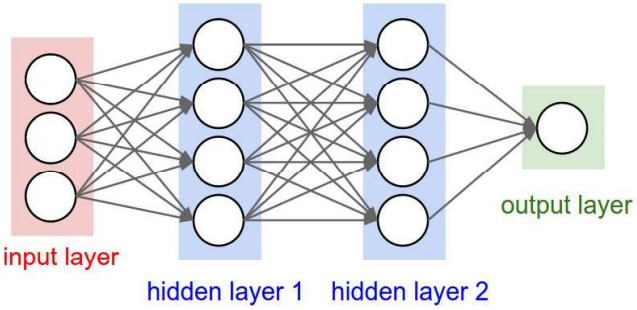
یک شبکه عصبی عمیق از تعدادی لایه میانی، تشکیل شده که خروجی هر لایه تابعی از خروجی لایه قبلی و پارامترهای آن لایه است. مثلا در مثال قبل، لایه اول تابعی از ورودی شبکه عصبی و ماتریس $\mathbf{W}$ بود. همچنین لایه دوم تابعی از خروجی لایه اول و پارامترهای تابع g . یک شبکه عصبی عمیق به صورت

$$\hat{f}(x) = f_n(f_{n-1}(\dots f_2(f_1(x,\theta_1),\theta_2)\dots),\theta_{n-1}),\theta_n)$$

تعریف می‌شود که برای وضوح بیشتر می‌توان آن را به صورت زیر نوشت:

$$\begin{aligned} y_1 &= f_1(x,\theta_1) \\ y_2 &= f_2(y_1,\theta_2) \\ &\vdots \\ y_{n-1} &= f_{n-1}(y_{n-2},\theta_{n-1}) \\ y_n &= f_n(y_{n-1},\theta_n) \end{aligned} \qquad (۴)$$

لایه اول (تابع f_1) ورودی x را گرفته و تابعی بر حسب آن و پارامترهای خود (θ_1) را محاسبه می‌کند. لایه دوم، خروجی لایه اول (y_1) را گرفته و تابعی بر حسب آن و پارامترهای خود (θ_2) محاسبه می‌کند. همین طور تا لایه آخر که تابعی بر حسب خروجی لایه قبلی (y_{n-1}) و پارامترهای خود (θ_n) را محاسبه می‌کند و به عنوان خروجی کل شبکه عصبی می‌دهد.



شکل ۳.۳

به تعداد توابع این شبکه عصبی، عمق آن می‌گویند. برد هر تابع میانی به دلخواه طراح شبکه عصبی است ولی برد تابع آخر باید مشابه برد تابع f باشد. به تعداد درایه‌های ماتریس خروجی هر تابع میانی، عرض آن لایه می‌گویند. تحقیقات جالبی درباره ارتباط عمق و عرض شبکه‌های عصبی با پیچیدگی توابع تولید شده توسط آن‌ها انجام شده است که یکی از جذاب‌ترین آن‌ها [۳] است. برای طرح یکی از نتایج جالب این مقاله لازم است یک اصطلاح پر کاربرد در تئوری شبکه‌های عصبی را معرفی کنم.

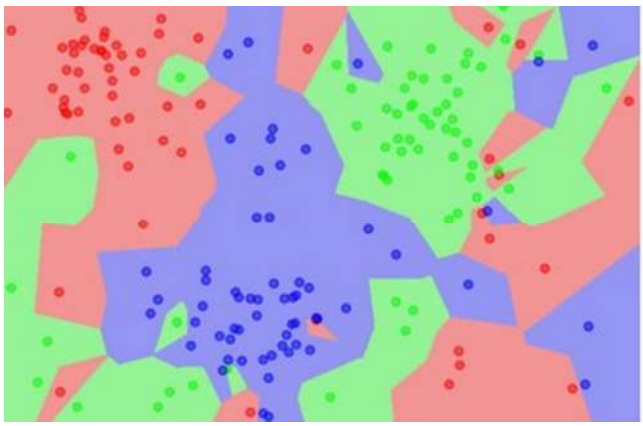
۱.۳.۳ ظرفیت یک شبکه عصبی

برای یک مدل، «ظرفیت [۵] » مدل، به طور شهودی، گستره توابع موجود در مدل و پیچیدگی آن‌ها تعریف می‌شود. هر چه تعداد و پیچیدگی توابع موجود در یک مدل بیشتر باشد، ظرفیت مدل بیشتر است. مثلا مدلی که فقط شامل توابع خطی است نمی‌تواند یک تابع درجه دو را نمایش دهد بنابراین ظرفیت آن کم است. اما مدلی که علاوه بر توابع خطی، توابع درجه دو را نیز شامل می‌شود دارای ظرفیت بیشتری است. برای محاسبه کمّی ظرفیت مدل، روش‌های زیادی وجود دارد که یکی از آن‌ها «بعد VC [۶] » است. این معیار، ظرفیت یک مدل را این گونه تعریف می‌کند:

[5] capacity
[6] VC dimension

«فرض می‌کنیم برد تابع f فقط مجموعه $\{0,1\}$ باشد. یک مجموعه دلخواه $x_1, x_2, \ldots, x_n$ در نظر می‌گیریم. اگر بتوانیم به ازای هر 2^n حالت $y_1, y_2, \ldots, y_n$ یک تابع مانند g در مدل بیابیم که $\forall i \in \{1, \ldots, n\} : \ g(x_i) = y_i$ می‌گوییم این مدل می‌تواند مجموعه $x_1, x_2, \ldots, x_n$ را به صورت دلخواه پخش کند. حال اگر مدل توانایی پخش کردن دلخواه هر مجموعه n تایی از ورودی‌ها را داشته باشد، می‌گوییم مدل ظرفیت پخش دلخواه n ورودی دلخواه را دارد. بزرگترین عدد n که مدل توانایی پخش دلخواه n ورودی دلخواه را داشته باشد، ظرفیت مدل در بعد VC است.»

می‌توان این تعریف را به حالات پیچیده‌تری نیز تعمیم داد. مثلا فرض کنید سبز,آبی,قرمز $\in y$ و $\mathbf{x}$ یک نقطه در صفحه $\mathbf{R}^2$ باشد و نمونه‌ها در شکل ۱.۳.۳ با نقطه مشخص شده باشند. در این صورت تابعی که در شکل ۱.۳.۳ ناحیه‌های صفحه را رنگ آمیزی کرده، نمونه‌ها را به درستی در مجموعه‌های «قرمز»، «آبی» و «سبز» پخش کرده است.



شکل ۱.۳.۳

در مقاله [۳] ، فرمولی برای تعداد ناحیه‌هایی که یک خانواده شایع از شبکه‌های عصبی می‌تواند آن‌ها را از هم تمیز دهد (مشابه شکل ۱.۳.۳) ارائه شده است. در این فرمول فرض شده عرض تمام لایه‌های میانی یکسان و برابر w است. همچنین عمق شبکه عصبی (تعداد لایه‌های میانی آن) برابر d و تعداد درایه‌های ماتریس ورودی کل شبکه عصبی n در نظر گرفته شده. تابع غیر خطی لایه‌های میانی نیز تابعی در نظر گرفته شده که مجموعه شایعی از شبکه‌های عصبی را شامل شود. در این مقاله ثابت شده تعداد ناحیه‌های خطی قابل تمیز این خانواده از شبکه‌های عصبی، از فرمول زیر به دست می‌آید.

$$O\left(\binom{w}{n}^{n(d-1)} - w^n\right) \qquad (۵)$$

این فرمول نشان می‌دهد اضافه کردن d بسیار بیشتر از اضافه کردن سایر مشخصه‌های شبکه عصبی می‌تواند ظرفیت شبکه عصبی را افزایش دهد. به همین خاطر شبکه‌های عصبی عمیق بیشتر از شبکه‌های عصبی عریض مورد توجه و استفاده قرار گرفته‌اند.

۴.۳ شبکه‌های عصبی پیچیده‌تر

یکی از مشاهدات جالب دانشمندان علوم شناختی، شبکه‌های عصبی‌ای بود که دور داشتند، یعنی خروجی یک نورون به نورون دیگری فرستاده می‌شد و آن نورون خروجی‌اش را به نورون دیگری می‌فرستاد و در نهایت این سلسله مجددا به نورون اولیه برمی‌گشت. محققان شبکه‌های عصبی مصنوعی تلاش‌هایی برای شبیه‌سازی این نوع شبکه‌های عصبی واقعی کردند [۷] . همچنین شبکه‌های عصبی مصنوعی طراحی شد که دارای حافظه بودند (مانند ماشین تورینگ اتفاقاتی که تا به این لحظه برای آن‌ها رخ داده است را به شکلی نگهداری می‌کردند) [۸] . همچنین شبکه‌های عصبی متنوعی برای کاربردهای خاص با معماری‌ها و توابع میانی خاص طراحی شدند و نتایج شگفت‌آوری تولید کردند. به طور مثال شبکه‌های عصبی پیچشی [۹] که برای تحلیل عکس‌ها مورد استفاده قرار می‌گیرند. یادگیری عمیق در حال حاضر یکی از فعال‌ترین و بزرگترین جامعه محققان را دارد و هر سال تعداد بسیار زیادی مقاله منتشر می‌شوند که هر کدام مسیر جدید برای استفاده از شبکه‌های عصبی در مسائل واقعی گشوده‌اند.

[7] Recursive Neural Networks
[8] Recurrent Neural Networks, Neural Turing Machines
[9] Convolutional Neural Networks

۴ کاربردهای شگفت‌انگیز شبکه‌های عصبی

همان طور که در این مقاله خواندید شبکه‌های عصبی از کنار هم قرار گرفتن تعداد زیادی ایده ساده تشکیل شده‌اند و هر زمان که سادگی یکی از این ایده‌ها، دقت پاسخ یا سرعت اجرای کد را به اندازه غیر قابل چشم‌پوشی‌ای تحت تاثیر قرار می‌داد، ایده‌هایی به عنوان جایگزین مطرح می‌شدند. این روند همچنان ادامه دارد و هر ساله، تغییرات شگرفی در ایده‌های اولیه شبکه‌های عصبی ایجاد می‌شود و همچنین ایده‌های جدید نیز دستخوش تغییراتی می‌شوند. جامعه بزرگ و توانمندی از دانشمندان و مهندسان حوزه علوم کامپیوتر و علوم دیگر از جمله علوم زیستی، هر سال با کمک شبکه‌های عصبی به پیشرفت‌های شگفت‌آوری در حوزه‌های مختلف دست می‌یابند. تشخیص بیماری‌هایی از جمله سرطان، تشخیص تخلف رانندگی، تشخیص چهره(با دقتی فراتر از انسان)، تبدیل گفتار به نوشتار و نوشتار به گفتار، نقاشی کردن، خلق یک اثر موسیقی، امنیت، تشخیص الگوهای بازار بورس، رنگی کردن عکس‌های سیاه و سفید، استخراج صدا از فیلم‌های صامت فقط با استفاده از لرزش حاصل از موج‌های مکانیکی، تشخیص سن و جنسیت از روی عکس، تولید اتوماتیک caption برای عکس‌ها و ... تعداد کمی از کاربردهای شگفت‌انگیز شبکه‌های عصبی هستند.

یکی از استفاده‌های بسیار شگفت‌انگیز شبکه‌های عصبی توانایی آن‌ها در یادگیری بازی‌های ساده کامپیوتری از جمله قارچ‌خور است. می‌توان شبکه عصبی طراحی کرد که بتواند بدون راهنمایی و فقط با بازی کردن بازی قارچ‌خور یاد بگیرد چطور بازی کند که امتیاز بیشتری کسب کند (دقیقا مشابه انسان). در واقع شما بازی را اجرا می‌کنید و شبکه عصبی شروع به بازی کردن می‌کند. در ابتدا حتی نمی‌داند باید به سمت راست حرکت کند که برنده شود. اما بعد از مدتی و با آزمون و خطا متوجه می‌شود (مانند انسان). بعد از آن شروع به مشاهده و یادگیری اجزا مختلف بازی می‌کند. مثلا ابتدا مستقیما به دل یک دشمن می‌زند و می‌بازد اما بعد از مدتی متوجه می‌شود که این کار موجب باخت می‌شود. همین طور در اواسط بازی خود را درون چاه‌ها می‌اندازد ولی به مرور متوجه می‌شود چطور از روی چاه‌ها بپرد. جالب این جاست که بعد از حدود ۳۰۰ دقیقه بازی کردن، سطح بازی این شبکه عصبی از انسان نیز فراتر می‌رود و حرکات ترکیبی شگرفی انجام می‌دهد که از توان محاسباتی انسان خارج است.

۵ ادامه دارد ...

در قسمت بعدی این مقاله تعدادی از الگوریتم‌های بهینه‌ساز شبکه‌های عصبی را بررسی می‌کنیم و برخی از توابع استفاده شده در شبکه‌های عصبی را معرفی و تجزیه و تحلیل خواهیم کرد.

مراجع

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. deep learning. deeplearningbook.org.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

[3] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Advances in neural information processing systems, pages 2924–2932, 2014.