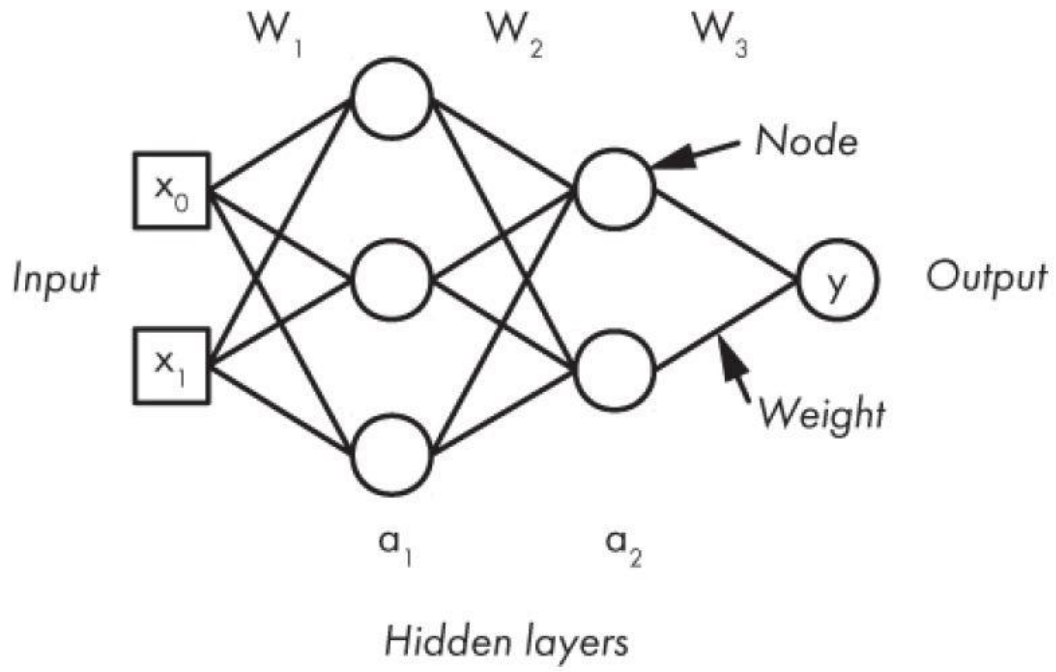
فصل ۸

معرفی شبکه‌های عصبی

شبکه‌های عصبی قلب یادگیری عمیق در فصل ۹ هستند. ما یک بررسی عمیق درمورد چیزی که به‌عنوان "شبکه‌های عصبی سنتی" شناخته می‌شود، انجام خواهیم داد. قبل از این کار آناتومی شبکه عصبی به‌همراه یک مثال کوچک معرفی خواهد شد.

به‌طور مشخص، اجزای شبکه عصبی پیشخور کاملاً متصل[۱] را ارائه می‌دهیم. از لحاظ نمایشی، می‌توانید شبکه را به‌صورت شکل ۱_۸ تصور کنید. ما در این فصل و فصل بعدی به این شکل ارجاع خواهیم داد. وظیفه شما این است که این تصویر را به خاطر بسپارید تا با ورق زدن کتاب و بازگشت مداوم به این شکل، کتاب را خراب نکنید!



شکل ۱_۸: یک نمونه از شبکه عصبی

بعد از بحث درمورد ساختار و اجزای شبکه عصبی، طی یک مثال، از شبکه عصبی استفاده خواهیم کرد

[۱] fully connected feed- forward neural network

تا گل‌های زنبق[۱] را دسته‌بندی[۲] کنیم. فصل ۹ درمورد الگوریتم گرادیان کاهشی[۳] و الگوریتم انتشار به عقب یا پس‌انتشار[٤] است. این الگوریتم‌ها روش‌هایی استاندارد برای آموزش شبکه‌های عصبی، از جمله شبکه‌های عصبی عمیق، هستند. این فصل به‌عنوان دست گرمی است، کار سنگین از فصل ۹ شروع خواهد شد!

آناتومی شبکه عصبی

شبکه عصبی یک گراف[٥] است. در علوم کامپیوتر، یک گراف از گره‌ها[٦] و یال‌ها[۷] تشکیل شده‌است. گره‌ها با دایره نمایش داده می‌شوند و به‌وسیله یال‌هایی که با خط نمایش داده می‌شوند، به‌هم وصل شده‌اند. این روش برای نشان دادن انواع مختلف روابط مفید است. به‌عنوان مثال: راه‌های بین شهری، افرادی که در شبکه‌های اجتماعی هم‌دیگر را می‌شناسند، ساختار اینترنت یا یک سری از واحدهای محاسباتی که می‌توانند برای تخمین هر تابع ریاضیاتی استفاده شوند.

مثال آخر، عمداً به‌کار برده شد. شبکه‌های عصبی تخمین زننده‌های جامع و فراگیر برای توابع هستند. شبکه‌های عصبی از ساختار گراف استفاده می‌کنند تا یک سری از مراحل محاسباتی را نشان دهند برای این‌که بتوانند یک بردارد ورودی از ویژگی‌ها[۸] را به خروجی تبدیل کنند. معمولاً مقدار خروجی به‌صورت احتمال تفسیر می‌شود. شبکه‌های عصبی به‌صورت لایه‌ای ایجاد می‌شوند. مفهوماً شبکه‌های عصبی از سمت چپ به راست عمل می‌کنند و یک بردار از ویژگی‌ها را به مقدار (یا مقادیر) خروجی تبدیل می‌کنند. این کار با عبور دادن مقادیر (اعداد) توسط یال‌ها به گره‌ها انجام می‌شود. توجه شود که گره‌های یک شبکه عصبی گاهاً سلول عصبی یا نورون[۹] هم نامیده می‌شود. علت این امر را به زودی خواهیم دید. گره‌ها، مقادیر جدیدی را براساس ورودی‌های خود محاسبه می‌کنند. سپس مقادیر جدید به لایه بعدی از گره‌ها می‌روند و همین کار ادامه پیدا می‌کند تا این‌که به گره‌های خروجی برسیم. در شکل ۱_۸، یک لایه ورودی

---

۱ irises

۲ classify

۳ Gradient descent

٤ backpropagation

٥ graph

٦ nodes

۷ edges

۸ features

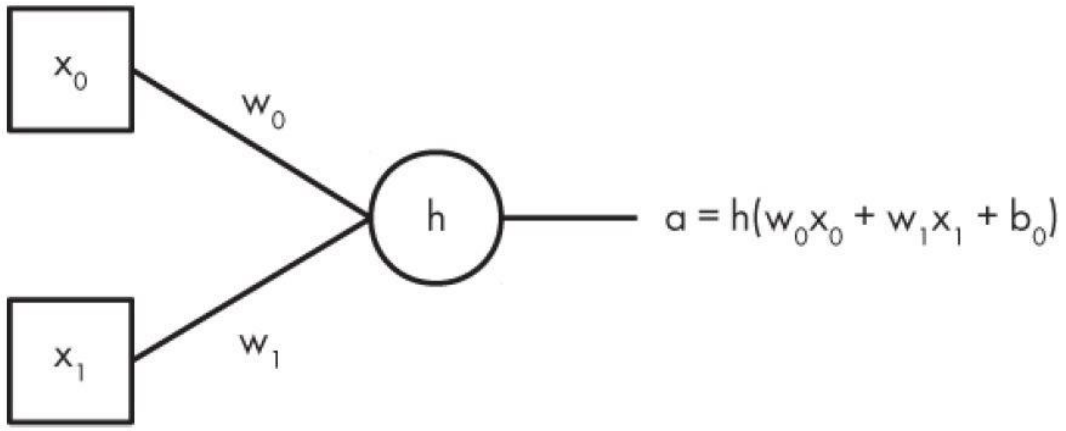
۹ neuron

در سمت چپ، در سمت راست آن یک لایه مخفی[1]، یک لایه مخفی دیگر در سمت راست آن و یک گره تکی در لایه خروجی وجود دارد.

در بخش قبل عبارت " شبکه عصبی پیشخور کاملاً متصل" را بدون توضیحات کافی آوردیم. در اینجا نگاه دقیق‌تری به آن می‌کنیم. "کاملاً متصل" بدین معناست که هر گره از یک لایه، خروجی خود را به تمامی گره‌ها از لایه بعدی می‌فرستد یا به عبارتی به تمامی گره‌های لایه بعدی متصل است. "پیشخور" بدین معناست که اطلاعات از سمت چپ به راست درون شبکه منتقل می‌شوند و به لایه‌های قبلی فرستاده نمی‌شوند. درواقع هیچ پسخور۲ (فیدبک) یا حلقه‌ای در ساختار شبکه وجود ندارد. تنها "شبکه عصبی" باقی می‌ماند که در ادامه بررسی می‌شود.

نورون

شخصاً هم احساس تنفر و هم عشق نسبت به عبارت "شبکه عصبی" دارم. خود عبارت از این حقیقت ناشی می‌شود که در یک تخمین نه چندان دقیق، واحد پایه‌ای شبکه، به نورون در مغز شباهت دارد. شکل ۲_۸ را درنظر بگیرید. به جزئیات این شکل به زودی پرداخته خواهد شد.



شکل ۲_۸: یک گره تکی از شبکه عصبی

به یاد دارید که تصویر ما از شبکه همواره به این صورت است که اطلاعات از سمت چپ به راست حرکت می‌کنند. به‌همین دلیل در این شکل، گره (دایره) ورودی را از چپ دریافت می‌کند و یک خروجی

[1] Hidden layer

[2] feedback

تکی در سمت راست خود دارد. در اینجا دو ورودی وجود دارد اما می‌توان صدها ورودی داشت.

این موضوع که تعداد زیادی ورودی به یک خروجی تبدیل شده‌اند نشان دهنده نحوه عملکرد نورون در مغز است. بخشی که دندریت[1] نام دارد، ورودی را از تعداد زیادی نورون دیگر دریافت می‌کند و یک بخش تکی به نام آکسون[2]، خروجی آن است. من این مقایسه را بسیار دوست دارم زیرا منجر به یک راه جالب برای فکر کردن و صحبت کردن درمورد شبکه‌ها می‌شود. اما من از این مقایسه متنفرم زیرا نورون‌های مصنوعی از لحاض عملیاتی کاملاً با نورون‌های واقعی متفاوت اند. با وجود این‌که از لحاض آناتومی یک شباهتی بین نورون‌های واقعی و مصنوعی وجود دارد؛ اما آن‌ها یکسان نیستند و این باعث سردرگمی برای افرادی می‌شود که با یادگیری ماشین آشنا نیستند. ممکن است افراد این عقیده را داشته باشند که دانشمندان کامپیوتر واقعاً درحال ساختن مغزهای مصنوعی هستند و یا شبکه‌ها می‌توانند فکر کنند. معنی کلمه "فکر کردن" دشوار است اما از نظر من فکر کردن کاری نیست که شبکه‌های عصبی می‌کنند.

در شکل ۲_۸، دو عدد مربع در سمت چپ، دو عدد خط، یک دایره و یک خط در سمت راست دیده می‌شود. همچنین نوشته‌هایی در آن تصویر وجود دارد. اگر ما شکل ۲_۸ را درک کنیم، به‌خوبی می‌توانیم شبکه‌های عصبی را درک کنیم. بعداً، مدل بصری خود را به‌صورت کد پیاده‌سازی می‌کنیم و متوجه خواهیم شد که چقدر ساده است.

همه‌چیز در شکل ۲_۸ روی دایره متمرکز شده‌است. این دایره یک گره است. این دایره درواقعیت یک تابع ریاضی را پیاده‌سازی می‌کند که تابع فعال‌سازی نام دارد. این تابع خروجی گره را به‌صورت یک عدد محاسبه می‌کند. دو مربع، ورودی‌های گره هستند. این گره ویژگی‌ها را از یک بردار ویژگی دریافت می‌کند. از مربع استفاده شده‌است تا نسبت به دایره متمایز شود اما ورودی می‌تواند از یک مجموعه از گره‌های دایره‌ای دیگر که مربوط به لایه قبلی است، بیاید.

هر ورودی یک عدد است، یک مقدار اسکالرِ تکی، که ما آن‌ها را به‌صورت x_0 و x_1 می‌شناسیم. این دو ورودی از طریق دو خط با عنوان w_0 و w_1 به سمت گره حرکت می‌کنند. این خط‌ها، وزن‌ها را نشان می‌دهند که درواقع نشان دهنده قدرت ارتباط است. از لحاض محاسباتی ورودی‌های (x_0, x_1) در ضرایب (w_0, w_1) ضرب می‌شوند، سپس با هم جمع می‌شوند و به تابع فعال‌سازی گره داده می‌شوند. در اینجا تابع فعال‌سازی را با h نمایش می‌دهیم.

مقدار تابع فعال‌سازی خروجی گره است. ما در اینجا این خروجی را a می‌نامیم. ورودی‌ها بعد از این‌که در وزن‌ها ضرب شدند به‌هم اضافه می‌شوند و به تابع فعال‌سازی داده می‌شوند تا مقدار خروجی را تولید

---

[1] dendrite

[2] axon

کند. لازم به ذکر است که یک مقدار b_0 نیز اضافه می‌شود و پس از آن به تابع فعال‌سازی ارسال می‌شود. این مقدار، بایاس یا اریبی[۱] نام دارد. این مقدار برای تطبیق دادن محدوده ورودی‌ها استفاده می‌شود تا برای تابع فعال‌سازی مناسب باشند. یک مقدار بایاس برای هر گره در هر لایه وجود دارد. در شکل ۲_۸ از آن‌جایی که b_0 اندیس 0 دارد، نشان‌دهنده این است که این گره اولین گره شبکه عصبی است. (به یاد بیاورید که در کامپیوتر همیشه شمارش از صفر شروع می‌شود و نه از یک.)

تمام کاری که یک گره از شبکه عصبی انجام می‌دهد این است: یک گره از شبکه عصبی چندین ورودی $x_0, x_1, \ldots$ را دریافت می‌کند، هرکدام از آن‌ها را در وزن‌ها $w_0, w_1, \ldots$ ضرب می‌کند، حاصل را با هم جمع می‌کند و مقدار بایاس b را اضافه می‌کند و حاصل مجموع را به تابع فعال‌سازی h می‌دهد تا یک مقدار اسکالر به‌عنوان خروجی a ایجاد کند:

$$a = h(w_0 x_0 + w_1 x_1 + \cdots + b)$$

به‌همین سادگی. تعدادی از گره‌ها را کنار هم قرار دهید، به‌طور مناسب آن‌ها را به‌هم متصل کنید، روشی برای آموزش آن‌ها و تعیین وزن‌ها و بایاس پیدا کنید و سپس یک شبکه عصبی مناسب خواهید داشت. همانطور که در فصل آینده خواهید دید، آموزش شبکه عصبی کار ساده‌ای نیست اما پس از آموزش استفاده از آن ساده است. بدین صورت که یک بردار از ویژگی‌ها را دریافت می‌کند و دسته‌بندی را انجام می‌دهد.

ما این گراف‌ها را شبکه عصبی نام نهادیم و با همین نام ادامه خواهیم داد. شاید گاهاً از مخفف NN استفاده کنیم. اگر مقالات یا کتاب‌های دیگر را خوانده باشید ممکن است که این گراف‌ها را شبکه‌های عصبی مصنوعی[۲] یا پرسپترون‌های چندلایه[۳] نامیده باشند. من استفاده از شبکه عصبی را پیشنهاد می‌کنم، اما خب این فقط نظر من است.

توابع فعال‌سازی

بیایید درمورد توابع فعال‌سازی صحبت کنیم. تابع فعال‌سازی برای یک گره، یک مقدار اسکالر را دریافت می‌کند که مجموع حاصل ضرب ورودی‌ها در وزن‌هایشان، به‌علاوه بایاس است. تابع فعال‌سازی سپس روی این مقدار کارهایی انجام می‌دهد. به‌طور ویژه، ما نیاز داریم که تابع فعال‌سازی غیرخطی باشد تا مدل بتواند توابع پیچیده را یاد بگیرد. از لحاض ریاضی، برای این‌که بدانیم تابع غیرخطی چیست بهتر است ابتدا بگوییم تابع خطی چه چیزی است و سپس هر تابعی که خطی نباشید، غیرخطی است.

---

[۱] bias

[۲] artificial neural networks (ANNs)

[۳] multi-layer perceptrons (MLPs)

خروجی یک تابع خطی، مثلاً g، به‌طور مستقیم متناسب با ورودی‌اش است. این موضوع را به‌صورت $g(x) \propto x$ نشان می‌دهیم که در آن $\propto$ به‌معنای متناسب بودن است. می‌توان گفت که نمودار یک تابع خطی یک خط راست است و درنتیجه هر تابعی که نمودارش خط راست نباشد، غیرخطی است.

به‌عنوان مثال تابع زیر

$$g(x) = 3x + 2$$

یک تابع خطی است زیرا نمودار آن یک خط راست است. یک تابع ثابت مانند $g(x) = 1$ نیز یک تابع خطی است. اما تابع زیر

$$g(x) = x^2 + 2$$

یک تابع غیرخطی است زیرا توان x، ۲ است. توابع غیرجبری[۱] نیز غیرخطی هستند. توابع غیرجبری توابعی هستند مانند $g(x) = logx$ یا $g(x) = e^x$ که در آن $e = 2.718 \dots$ پایه لگاریتم طبیعی است. توابع دیگری مانند سینوس و کسینوس، معکوس آن‌ها و توابعی مانند تانژانت که که از سینوس و کسینوس ساخته می‌شوند نیز توابع غیرجبری هستند. این توابع، غیرجبری هستند زیرا نمی‌توانید آن‌ها را از ترکیب متناهی از عملیات جبری اصلی ایجاد کنید. این توابع غیرخطی هستند زیرا نمودار آن‌ها خط راست نیست.

شبکه به تابع فعال‌سازی غیرخطی نیاز دارد؛ درغیر این صورت تنها می‌تواند نگاشت‌های خطی را یاد بگیرد و برای این‌که شبکه‌ها به‌صورت کلی کاربردی باشند، نگاشت‌های خطی کافی نیست.

یک شبکه آزمایشی متشکل شده از ۲ گره را تصور کنید که هر گره یک ورودی دارد. این بدین معناست که یک وزن و یک مقدار بایاس برای هر گره وجود دارد. فرض کنید خروجی گره اول، ورودی گره دوم است. اگر ما تابع خطی $g(x) = 5x - 3$ را درنظر بگیریم، شبکه برای ورودی x خروجی a_1 را به‌صورت زیر محاسبه می‌کند.

$$\begin{aligned}
a_1 &= h(w_1 a_0 + b_1) \\
&= h(w_1 h(w_0 x + b_0) + b_1) \\
&= h(w_1(5(w_0 x + b_0) - 3) + b_1) \\
&= h(w_1(5w_0 x + 5b_0 - 3) + b_1) \\
&= h(5w_1 w_0 x + 5w_1 b_0 - 3w_1 + b_1) \\
&= 5(5w_1 w_0 x + 5w_1 b_0 - 3w_1 + b_1) - 3 \\
&= (25w_1 w_0)x + (25w_1 b_0 - 15w_1 + 5b_1 - 3) \\
&= Wx + B
\end{aligned}$$

که در آن $W = 25w_1 w_0$ و $B = 25w_1 b_0 - 15w_1 + 5b_1 - 3$ است که خود یک تابع خطی است. درواقع این تابع یک خط دیگر با شیب W و عرض از مبدأ B است و هیچ کدام از W و B به x وابسته نیستند. بنابراین یک شبکه عصبی با تابع فعال‌سازی خطی تنها مدل‌های خطی را می‌تواند یاد بگیرد زیرا ترکیب توابع خطی دوباره یک تابع خطی می‌دهد. دقیقاً همین محدودیت خطی بودن تابع فعال‌سازی بود

[۱] Transcendental functions

که باعث شد اولین شبکه عصبی توسط وینتر[1] در دهه ۱۹۷۰ ایجاد شود. تحقیق درمورد شبکه‌های عصبی به‌طور کامل کنار گذاشته شده بود زیرا تصور بر این بود که آن‌ها برای یادگیری توابع پیچیده پیش پا افتاده هستند.

باشه، پس ما توابع فعال‌سازی غیرخطی می‌خواهیم. اما کدامشان؟ تعداد بی‌نهایت از این توابع وجود دارند. در عمل تعداد کمی از آن‌ها، به‌دلیل مفید بودن یا خواص خوبشان مورد استفاده قرار می‌گیرند. شبکه‌های عصبی سنتی از تابع فعال‌سازی سیگموئید[2] یا تانژانت هیپربولیک[3] استفاده می‌کردند. تابع سیگموئید به‌صورت زیر است.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

و تابع تانژانت هیپربولیک به‌صورت زیر است.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

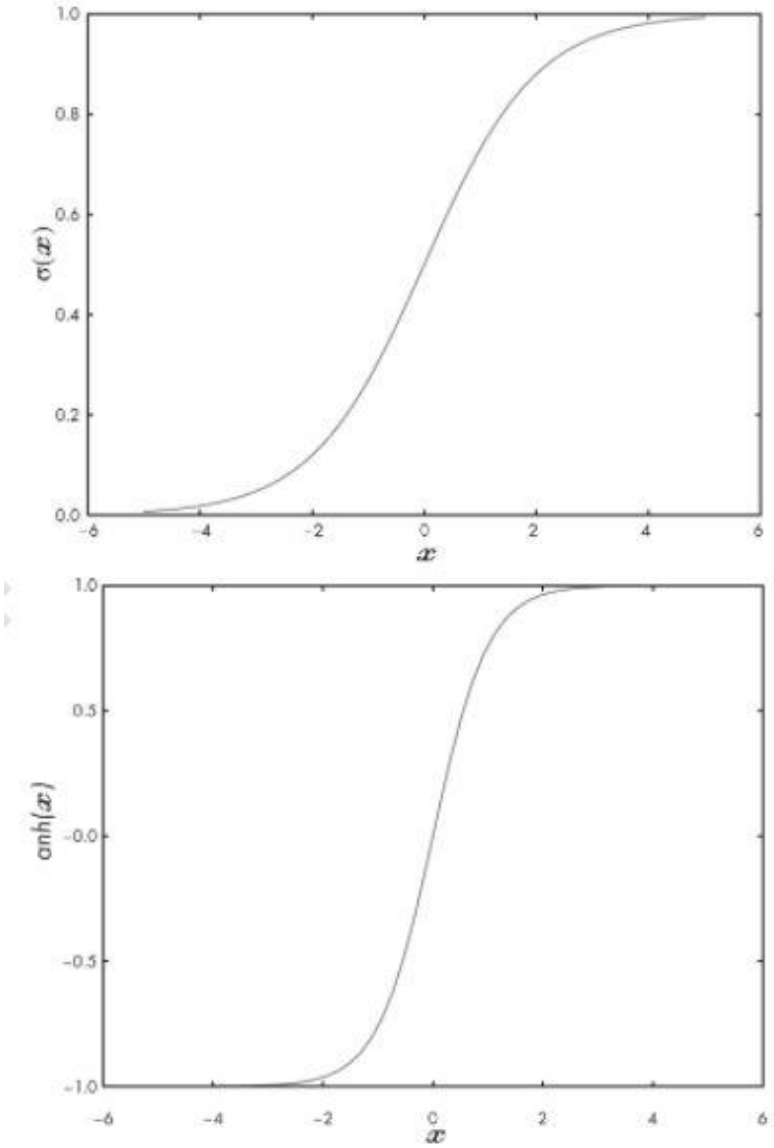
نمودار این دو تابع در شکل ۳_۸ آورده شده‌است. در این شکل نمودار سیگموئید در بالا و نمودار تانژانت هیپربولیک در زیر آن قرار داده شده‌است.

اولین موردی که توجه را جلب می‌کند این است که هر دو تابع شکلی شبیه "s" دارند. در تابع سیگموئید هرچه شما به سمت چپ محور x ها بروید، مقدار تابع به سمت صفر میل می‌کند و هرچه به سمت راست محور x ها بروید مقدار تابع به سمت ۱ میل می‌کند. در x=0 تابع مقدار 0.5 را دارد. تابع تانژانت هیپربولیک همین رفتار را دارد اما از -1 تا 1 مقدار می‌گیرد و در x=0 مقدار صفر را دارد.

[1] winter

[2] sigmoid

[3] hyperbolic tangent



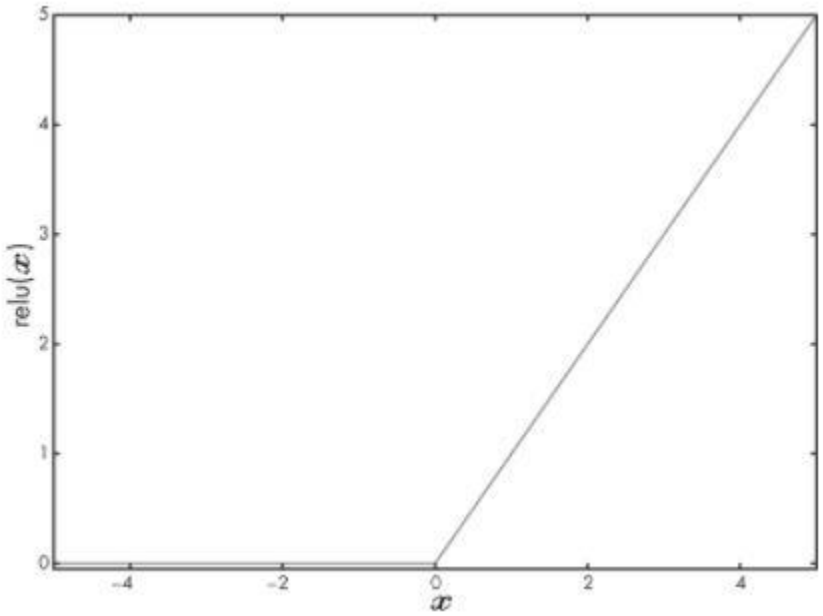
شکل ۳_۸: یک تابع سیگموئید (بالا) و یک تابع تانژانت هیپربولیک (پایین). توجه شود که مقیاس محور y یکسان نیست.

اخیراً به جای توابع سیگموئید و تانژانت هیپربولیک از واحد یکسوساز خطی[۱] استفاده می‌شود. این تابع را به‌طور مختصر با ReLU نشان می‌دهند. ReLU ساده است و خواص مناسبی برای شبکه‌های عصبی دارد. اگرچه لفظ خطی در اسم آن آمده است، اما ReLU تابعی غیرخطی است (نمودار آن یک خط مستقیم نیست). زمانی که آموزش شبکه‌های عصبی با استفاده از پس‌انتشار را در فصل ۹ بررسی کنیم، متوجه خواهیم شد که چرا از ReLU بیشتر استفاده می‌شود.

[۱] rectified linear unit

تابع ReLU به‌صورت زیر است و نمودار آن در شکل ٤_٨ آمده است.

$$ReLU(x) = \max(0, x) = \begin{cases} 0 & ,if\ x < 0 \\ x & ,otherwise \end{cases}$$



شکل ٤_٨: تابع فعال‌سازی یکسوساز خطی، $ReLU(x) = \max(0, x)$

ReLU، یکسوساز است زیرا مقادیر منفی را حذف می‌کند و به‌جای آن‌ها صفر قرار می‌دهد. در حقیقت افرادی که در حوزه یادگیری ماشین فعالیت دارند نسخه‌های متفاوتی از این تابع را استفاده می‌کنند، اما همه‌ی این افراد مقادیر منفی را حذف می‌کنند و به‌جای آن‌ها یک مقدار ثابت یا مقادیر دیگر را قرار می‌دهند. خاصیت قطعه قطعه بودن ReLU باعث می‌شود که این تابع، غیرخطی، و برای استفاده به‌عنوان تابع فعال‌سازی در یک شبکه عصبی مناسب باشد. همچنین این تابع از لحاض محاسباتی ساده، و بسیار سریع‌تر از توابع سیگموئید یا تانژانت هیپربولیک است. دلیل این امر این است که توابع سیگموئید و تانژانت هیپربولیک از e^x استفاده می‌کنند که به زبان کامپیوتر به‌معنای فراخوانی تابع exp است. این تابع به‌صورت مجموع یک دنباله پیاده‌سازی می‌شود و عملیات زیادی را می‌طلبد. می‌توان این تعداد از عملیات را با یک if ساده در تابع ReLU مقایسه کرد. صرفه‌جویی‌های کوچکی مانند این مورد وقتی در یک شبکه هزاران گره وجود دارد خود را بیشتر نشان می‌دهد.

معماری یک شبکه

ما تا اینجا درمورد گره‌ها و نحوه عملکرد آن‌ها بحث کردیم و اشاره کردیم که گره‌ها به‌هم متصل می‌شوند تا شبکه‌ها را ایجاد کنند. معماری شبکه درواقع نحوه اتصال گره‌ها است. شبکه‌های عصبی استاندارد مانند مواردی که ما در این فصل با آن‌ها کار می‌کنیم، همانطور که در شکل ۱_۸ نشان داده شد، به‌صورت لایه‌ای ایجاد می‌شوند. نیازی نیست که نحوه اتصال به این صورت باشد اما همانطور که خواهیم دید، اینکار باعث سادگی محاسباتی و ساده‌سازی آموزش می‌شود. یک شبکه پیشخور دارای یک لایه ورودی، یک یا چند لایه مخفی و یک لایه خروجی است. لایه ورودی همان بردار ویژگی‌ها، و لایه خروجی همان پیش‌بینی (احتمال) است. اگر شبکه برای یک مسئله چندکلاسی[۱] باشد، ممکن است لایه خروجی بیش از یک گره داشته باشد که در آن هر گره نشان دهنده پیش‌بینی مدل از احتمال تعلق ورودی به هرکدام از کلاس‌ها است.

لایه‌های مخفی از گره‌ها تشکیل شده‌اند. گره‌های لایه i ام، خروجی گره‌های لایه i-1 ام را به‌عنوان ورودی دریافت می‌کنند و خروجی خود را به‌عنوان ورودی گره‌های لایه i+1 می‌دهند. ارتباط بین لایه‌ها معمولاً به‌صورت کاملاً متصل است؛ بدین معنی که هر خروجی از هر گره در لایه i-1 ام به‌عنوان یک ورودی به هر گره از لایه i ام استفاده می‌شود. در اینجا هم نیازی نیست که حتماً نحوه اتصال اینگونه باشد اما این کار پیاده‌سازی را ساده می‌کند.

تعداد لایه‌های مخفی و تعداد گره‌ها در هر لایه، معماری شبکه را مشخص می‌کند. ثابت شده‌است که یک لایه مخفی با تعداد کافی از گره‌ها می‌تواند هر نوع تابع یا نگاشتی را یاد بگیرد. این موضوع خوب است زیرا نشان می‌دهد که شبکه‌های عصبی می‌توانند برای مسائل یادگیری ماشین به‌کار بروند. در انتها، مدل به‌عنوان یک نگاشت پیچیده عمل می‌کند که ورودی‌ها را به خروجی تبدیل می‌کند. به هر حال این بدین معنا نیست که یک شبکه تک لایه می‌تواند برای همه شرایط به‌کار برده شود. وقتی تعداد گره‌ها در هر لایه‌ای زیاد شود، تعداد پارامترهای یادگیری (وزن‌ها و بایاس‌ها) نیز زیاد می‌شود و درنتیجه مقدار داده مورد نیاز برای آموزش افزایش می‌یابد. این دقیقاً همان مشقت بُعد–چندی[۲] است.

مشکلاتی از این قبیل، باعث توقف شبکه‌های عصبی برای دومین بار در دهه ۱۹۸۰ میلادی شد. کامپیوترها برای یادگیری شبکه‌های بزرگ خیلی کند بودند و همچنین در کل داده‌ی بسیار کمی برای آموزش شبکه در دست بود. متخصصین می‌دانستند که اگر شرایط برای هر دو مورد تغییر کند، این امکان وجود خواهد داشت که شبکه‌های بزرگی آموزش داده شوند، که نسبت به شبکه‌های کوچک موجود در

---

[۱] multiclass

[۲] Curse of dimensionality (مترجم: این یک مفهوم ریاضیاتی است که به مشقت هایی که افزایش ابعاد به مسئله اضافه می کند می پردازد)

آن زمان عملکرد بسیار بهتری داشته باشند. خوشبختانه این شرایط در اوایل دهه ۲۰۰۰ میلادی تغییر کرد.

انتخاب یک معماری مناسب برای شبکه تأثیر به‌سزایی روی این‌که آیا مدل چیزی را یاد خواهد گرفت یا نه، خواهد داشت. در اینجا تجربه و بینش نقش بازی می‌کنند. انتخاب معماری درست، هنر سیاه استفاده از شبکه‌های عصبی است. در ادامه تعدادی قوانین سر انگشتی در این مورد آورده شده‌است.

- اگر ورودی شما روابط فضایی مشخصی دارد، مانند قسمت‌های یک تصویر، ممکن است بهتر باشد از شبکه‌های عصبی پیچشی استفاده شود (فصل ۱۲)
- بیش از ۳ لایه مخفی استفاده نکنید. از تئوری به یاد بیاورید که یک لایه مخفی که به اندازه کافی بزرگ باشد، کافی است. پس تا جای ممکن از لایه‌های مخفی کم، به میزان نیاز استفاده کنید. اگر مدل با یک لایه مخفی یاد می‌گیرد، لایه دوم را اضافه کنید و ببینید که چیزی بهبود می‌یابد یا خیر.
- تعداد گره‌ها در اولین لایه مخفی باید یا برابر و یا (در حالت ایده‌آل) بیشتر از تعداد گره‌های ورودی باشد.
- به‌جز اولین لایه مخفی (که در قانون قبلی به آن اشاره شد)، تعداد گره‌ها در هر لایه باید یا برابر و یا بین تعداد گره‌ها در لایه قبلی و بعدی باشد. اگر لایه i-1 تعداد N گره، و لایه i+1 تعداد M گره دارد، خوب است که لایه i تعداد $N \leq x \leq M$ گره داشته باشد.

قانون اول می‌گوید که شبکه‌های عصبی سنتی برای موقعیت‌هایی که ورودی، روابط فضایی نداشته باشد، یعنی تصویر نباشد، بهتر است. همچنین وقتی که ابعاد ورودی‌ها کوچک است یا زمانی که تعداد زیادی داده برای آموزش در دست ندارید (که آموزش شبکه‌های عصبی پیچشی را بسیار دشوار می‌کند) باید شبکه‌های عصبی سنتی را امتحان کنید. اگر فکر می‌کنید که در موقعیتی هستید که شبکه عصبی سنتی مورد نیاز است، از کوچک شروع کنید و تا زمانی که عملکرد بهبود می‌یابد آن را بزرگ کنید.

لایه‌های خروجی

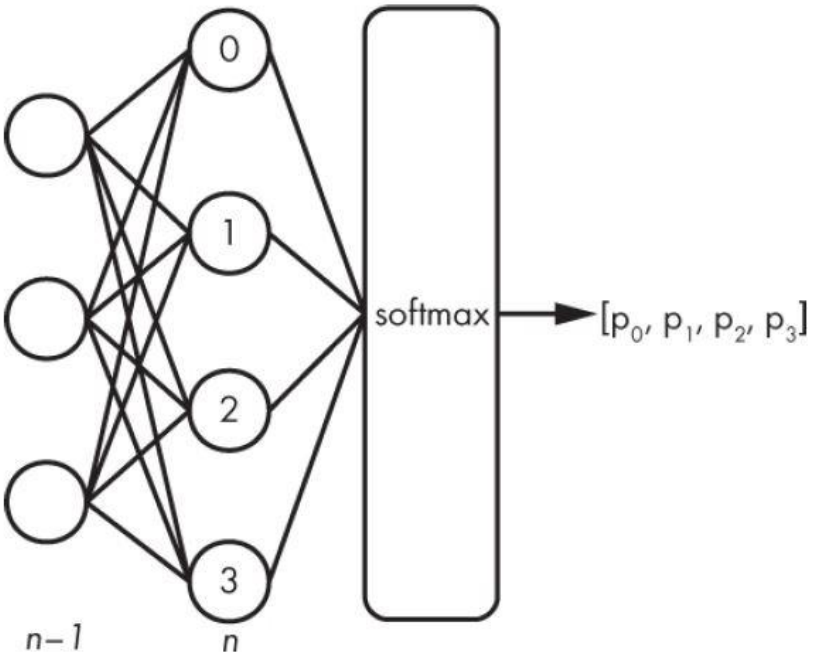
آخرین لایه در شبکه عصبی لایه خروجی است. اگر شبکه در حال مدل کردن یک مقدار پیوسته است (که به آن رگرسیون می‌گویند و ما در این کتاب آن را بررسی نمی‌کنیم) لایه خروجی یک گره است که از تابع فعال‌سازی استفاده نمی‌کند و صرفاً ورودی تابع h در شکل ۲_۸ را به‌عنوان خروجی می‌دهد. توجه کنید که این کار دقیقاً مانند این است که بگوییم تابع فعال سازی، همان تابع همانی $h(x) = x$ است.

شبکه‌های عصبی ما در این کتاب، برای دسته‌بندی به کار می‌روند و از آن‌ها انتظار داریم که یک مقدار تصمیم را به‌عنوان خروجی بدهند. اگر ما دو کلاس با برچسب‌های صفر و یک داشته باشیم، تابع فعال‌سازی

آخرین گره را سیگموئید درنظر می‌گیریم. این تابع یک مقدار بین صفر و یک می‌دهد که ما می‌توانیم آن را به‌عنوان احتمال تعلق ورودی به کلاس صفر یا یک، تعبیر کنیم. ما تصمیم‌گیری درمورد دسته‌بندی را براساس مقدار خروجی با یک قانون ساده انجام می‌دهیم. اگر مقدار تابع فعال‌سازی کمتر از 0.5 باشد، ورودی را متعلق به کلاس صفر، و در غیر این صورت متعلق به کلاس ۱ می‌دانیم. در فصل ۱۱ خواهیم دید که چگونه تغییر دادن این آستانه 0.5، می‌تواند باعث بهبود عملکرد مدل شود.

اگر بیش از دو کلاس وجود داشته باشد، باید روش دیگری انتخاب کنیم. به‌جای یک گره در لایه خروجی N گره خواهیم داشت و به هر کلاس یک گره اختصاص می‌دهیم. همه گره‌ها تابع همانی $h(x) = x$ را برای h خواهند داشت. پس از آن عملیات softmax را روی این N گره اجرا می‌کنیم و خروجی که بیشترین مقدار softmax را داشته باشد انتخاب می‌کنیم.

فرض کنید یک مجموعه داده با چهار کلاس در اختیار دارید. چیزی که این کلاس‌ها نشان می‌دهند مهم نیست، شبکه این را نمی‌فهمد. کلاس‌ها با اعداد 2،1،0 و3 شماره‌گذاری شده‌اند. بنابراین N=4 نشان می‌دهد که شبکه‌ی ما چهار خروجی خواهد داشت و هرکدام، از تابع همانی برای h استفاده می‌کنند. این موضوع در شکل ۵_۸ نشان داده شده‌است. در این شکل ما عملیات softmax و بردار خروجی حاصل از آن را نیز نشان داده ایم.



شکل ۵_۸: آخرین لایه مخفی n-1 و لایه خروجی n (که گره‌ها شماره گذاری شده‌اند) برای یک شبکه عصبی با چهار کلاس. عملیات softmax اجرا شده و یک بردار خروجی با چهار عنصر $[p_0, p_1, p_2, p_3]$ ایجاد شده‌است.

ما اندیس بزرگترین مقدار در بردار خروجی را به‌عنوان برچسب منتخب برای ورودی درنظر می‌گیریم. عملیات softmax اطمینان حاصل می‌کند که مجموع عناصر بردار خروجی برابر ۱ شود. بنابراین با اغماض می‌توانیم مقادیر بردار خروجی را احتمال تعلق ورودی به هر یک از این چهار کلاس بدانیم. به‌همین دلیل است که بیشترین مقدار را برای تعیین کلاس درنظر می‌گیریم.

عملیات softmax سرراست است. احتمال هر یک از خروجی‌ها به‌صورت زیر است:

$$p_i = \frac{e^{a_i}}{\sum_j e^{a_j}}$$

که در آن a_i، i امین خروجی است و در مخرج، جمع روی تمامی مقادیر خروجی انجام شده‌است. به‌عنوان مثال i=0,1,2,3 و اندیس بیشترین مقدار به‌عنوان کلاس ورودی درنظر گرفته می‌شود.

به‌عنوان مثال فرض کنید خروجی چهار گره موجود در آخرین لایه به‌صورت زیر باشد:

$$a_0 = 0.2$$
$$a_1 = 1.3$$
$$a_2 = 0.8$$
$$a_3 = 2.1$$

درنتیجه softmax به‌صورت زیر محاسبه می‌شود:

$$p_0 = e^{0.2}/(e^{0.2} + e^{1.3} + e^{0.8} + e^{2.1}) = 0.080$$
$$p_1 = e^{1.3}/(e^{0.2} + e^{1.3} + e^{0.8} + e^{2.1}) = 0.240$$
$$p_2 = e^{0.8}/(e^{0.2} + e^{1.3} + e^{0.8} + e^{2.1}) = 0.146$$
$$p_3 = e^{2.1}/(e^{0.2} + e^{1.3} + e^{0.8} + e^{2.1}) = 0.534$$

کلاس ۳ انتخاب می‌شود زیرا p_3 بیشترین مقدار را دارد. توجه کنید همانطور که انتظار داشتیم جمع p_iها برابر ۱ است.

دو نکته باید ذکر شود. در فرمول بالا، از تابع سیگموئید برای محاسبه خروجی شبکه استفاده کردیم. اگر تعداد کلاس‌ها را ۲ درنظر بگیریم و softmax را محاسبه کنیم، دو مقدار برای خروجی خواهیم داشت، به‌عنوان مثال یکی p و دیگری 1-p است. این امر دقیقاً مشابه استفاده از تابع سیگموئید به تنهایی است که احتمال تعلق داشتن ورودی به کلاس ۱ را نشان می‌داد.

نکته دوم درمورد پیاده‌سازی softmax است. اگر خروجی شبکه یا مقادیر a خیلی بزرگ باشند، مقدار e^a نیز بزرگ می‌شود. این مورد چیزی نیست که کامپیوتر دوست داشته باشد. حداقل این است که دقت از بین می‌رود، یا ممکن است سرریز[۱] کند و درنتیجه خروجی بی‌معنا شود.

اگر قبل از محاسبه softmax مقدار بزرگترین a را از تمامی مقادیر دیگر a کم کنیم، توان تابع نمایی کوچکتر خواهد شد و احتمال سرزیر کاهش خواهد یافت. انجام این کار برای مثال قبلی باعث می‌شود

[۱] overflow

مقادیر جدید برای a داشته باشیم.

$$a'_0 = 0.2 - 2.1 = -1.9$$
$$a'_1 = 1.3 - 2.1 = -0.8$$
$$a'_2 = 0.8 - 2.1 = -1.3$$
$$a'_3 = 2.1 - 2.1 = 0$$

ما مقدار 2.1 را کم کردیم زیرا بیشترین مقدار a است. با انجام این کار دقیقاً همان مقادیر p قبلی بدست خواهند آمد. تفاوت این است که اگر هرکدام از مقادیر a خیلی بزرگ باشد، این بار در برابر سرریز محافظت شده‌ایم.

نمایش وزن‌ها و بایاس‌ها

قبل از بررسی یک مثال از شبکه عصبی، بهتر است وزن‌ها و بایاس‌ها را بازبینی کنیم. متوجه خواهیم شد که اگر شبکه را به‌صورت ماتریسی یا برداری ببینیم، پیاده‌سازی آن به‌شدت آسان می‌شود.

یک نگاشت از یک بردار ویژگی ورودی با ۲ عنصر به اولین لایه مخفی با ۳ گره را درنظر بگیرید (a_1 در شکل ۱_۸). یال‌های بین این دو لایه (وزن‌ها) را به‌صورت w_{ij} اسم گذاری می‌کنیم که در این مثال برای ورودی‌های x_0 و x_1 مقادیر i=0,1 را داریم و برای سه گره در لایه مخفی مقادیر j=0,1,2 را داریم که در شکل از بالا به پایین شماره گذاری شده‌اند. به‌علاوه برای هر گره در لایه مخفی یک مقدار بایاس (درمجموع سه مقدار) نیاز داریم که در شکل نشان داده نشده‌است. این مقادیر را b_0، b_1 و b_2 می‌نامیم و در اینجا نیز این نام گذاری از بالا به پایین است. برای محاسبه خروجی توابع فعال‌سازی h، برای این سه گره باید مقادیر زیر محاسبه شوند.

$$a_0 = h(w_{00}x_0 + w_{10}x_1 + b_0)$$
$$a_1 = h(w_{01}x_0 + w_{11}x_1 + b_1)$$
$$a_2 = h(w_{02}x_0 + w_{12}x_1 + b_2)$$

با توجه به نحوه عملکرد ضرب ماتریسی و جمع برداری می‌توان این عملیات را به‌صورت زیر انجام داد.

$$\vec{a} = h\left(\begin{bmatrix} w_{00} & w_{10} \\ w_{01} & w_{11} \\ w_{02} & w_{12} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}\right) = h(W\vec{x} + \vec{b})$$

که در آن

که در آن $\vec{a} = (a_0, a_1, a_2)$، $\vec{x} = (x_0, x_1)$، $\vec{b} = (b_0, b_1, b_2)$ و W یک ماتریس 2×3 از مقادیر وزن‌ها است. در این مورد، یک بردار از ورودی‌ها به تابع فعال‌سازی h داده شده‌است تا مقادیر خروجی را تولید کند. این کار با اعمال تابع h روی هرکدام از عناصر $W\vec{x} + \vec{b}$ انجام می‌شود. به‌عنوان مثال اعمال h روی یک بردار $\vec{x}$ با سه عنصر به‌صورت زیر است.

$$h(\vec{x}) = h\big((x_0, x_1, x_2)\big) = \big(h(x_0), h(x_1), h(x_2)\big)$$

درواقع تابع h به‌طور مجزا روی هرکدام از عناصر $\vec{x}$ اعمال شده‌است.

از آن جایی که ماژول *NumPy* در پایتون برای کار با آرایه‌ها ایجاد شده‌است، و بردارها و ماتریس‌ها آرایه هستند، به این نتیجه می‌رسیم که وزن‌ها و بایاس‌های یک شبکه عصبی می‌توانند در آرایه‌های *NumPy* ذخیره شوند. درنتیجه ما تنها به عملیات ساده ماتریسی (مثلاً *np.dot*) و عملیات جمع برای کار کردن با یک شبکه عصبی کاملاً متصل نیاز داریم. توجه کنید که علت این‌که علاقه داریم از شبکه‌های کاملاً متصل استفاده کنیم این است که پیاده‌سازی آن سرراست است.

برای ذخیره سازی شبکه موجود در شکل ۱_۸، یک ماتریس برای وزن‌ها و یک بردار برای بایاس بین هرکدام از لایه‌ها نیاز داریم. با این کار سه ماتریس و سه بردار خواهیم داشت: یک ماتریس و یک بردار برای ورودی به اولین لایه مخفی، اولین لایه مخفی به دومین لایه مخفی و دومین لایه مخفی به خروجی. ماتریس‌های وزن به‌ترتیب دارای ابعاد 2×3، 3×2 و 2×1 هستند. بردارهای بایاس نیز به‌ترتیب دارای طولی به اندازه *3*،*2* و *1* هستند.

پیاده‌سازی یک شبکه عصبی ساده

در این بخش ما شبکه عصبی ساده موجود در شکل ۱_۸ را پیاده‌سازی می‌کنیم. پس از آن، این شبکه را با دو ویژگی از مجموعه داده گل‌های زنبق آموزش می‌دهیم. ما پیاده‌سازی شبکه را از ابتدا شروع می‌کنیم اما برای آموزش آن از *sklearn* استفاده می‌کنیم. هدف این بخش این است که نشان دهد چقدر پیاده‌سازی یک شبکه عصبی ساده، سرراست است. امیدواریم این کار ابهاماتی که از مباحث قبلی بوجود آمده است را نیز برطرف نماید.

شبکه موجود در شکل ۱_۸، یک بردار ویژگی ورودی با دو ویژگی (مترجم: یک بردار ورودی با دو عنصر) را دریافت می‌کند. این شبکه دارای دو لایه مخفی است، یکی با سه گره و دیگری با دو گره و یک خروجی سیگموئید نیز دارد (مترجم: منظور این است که تابع فعال‌سازی لایه خروجی سیگموئید است). توابع فعال‌سازی لایه مخفی نیز سیگموئید هستند.

ساختن مجموعه داده

قبل از بررسی کد شبکه عصبی، ابتدا مجموعه داده‌ای که با آن مدل را آموزش می‌دهیم بررسی می‌کنیم.

از قبل با مجموعه داده گل زنبق آشنایی داریم، اما برای این مثال تنها از دو کلاس و دو ویژگی از چهار ویژگی استفاده می‌کنیم. کد مربوط به ساختن مجموعه داده آموزش و تست در لیست ۱_۸ قرار دارد.

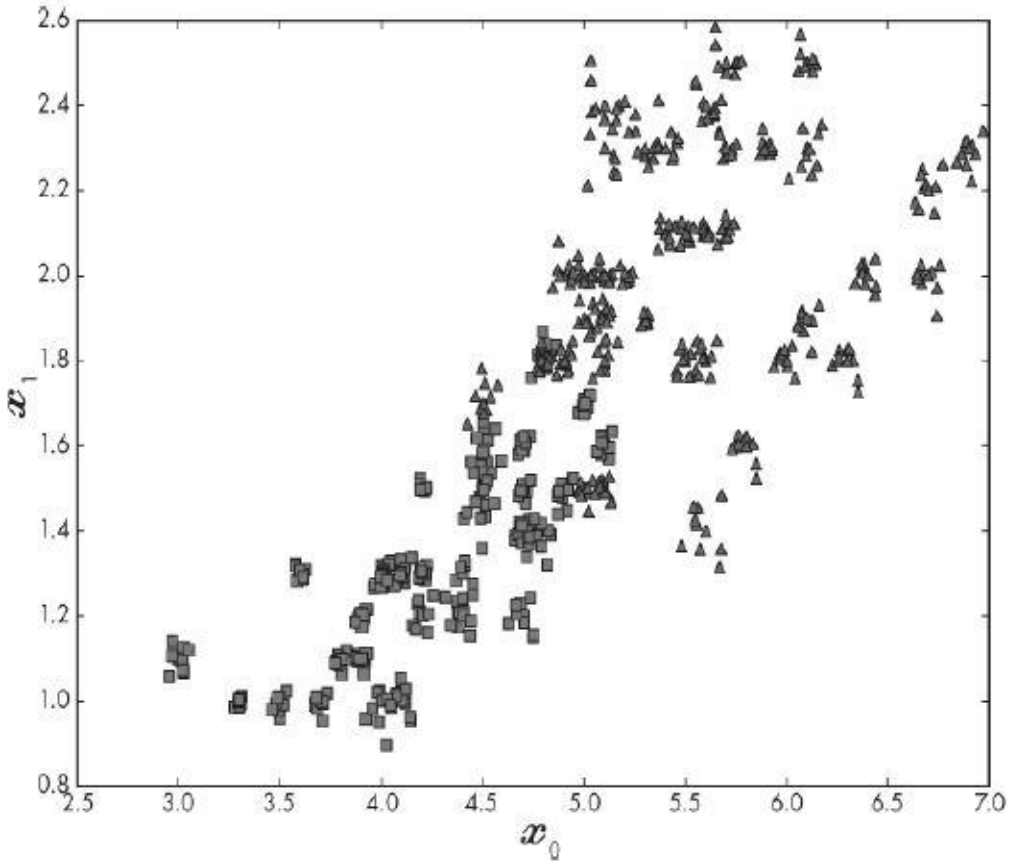
```
import numpy as np
❶ d = np.load("iris_train_features_augmented.npy")
l = np.load("iris_train_labels_augmented.npy")
d1 = d[np.where(l==1)]
d2 = d[np.where(l==2)]
❷ a=len(d1)
b=len(d2)
x = np.zeros((a+b,2))
x[:a,:] = d1[:,2:]
x[a:,:] = d2[:,2:]
❸ y = np.array([0]*a+[1]*b)
i = np.argsort(np.random.random(a+b))
x = x[i]
y = y[i]
❹ np.save("iris2_train.npy", x)
np.save("iris2_train_labels.npy", y)
❺ d = np.load("iris_test_features_augmented.npy")
l = np.load("iris_test_labels_augmented.npy")
d1 = d[np.where(l==1)]
d2 = d[np.where(l==2)]
a=len(d1)
b=len(d2)
x = np.zeros((a+b,2))
x[:a,:] = d1[:,2:]
x[a:,:] = d2[:,2:]
y = np.array([0]*a+[1]*b)
i = np.argsort(np.random.random(a+b))
x = x[i]
y = y[i]
np.save("iris2_test.npy", x)
np.save("iris2_test_labels.npy", y)
```

لیست ۱_۸: ساختن مجموعه داده ساده برای مثال مربوط به شبکه عصبی. nn_iris_dataset.py را ببینید.

در این کد مقداری داده‌ها را دستکاری کردیم. با مجموعه داده تقویت شده (augmented) شروع کردیم و نمونه‌ها و برچسب‌ها را بارگذاری کردیم ❶. ما تنها کلاس‌های ۱ و ۲ را می‌خواهیم. بنابراین اندیس‌های این نمونه‌ها را پیدا کردیم و فقط آن‌ها را در مجموعه داده درنظر گرفتیم. ما تنها ویژگی‌های ۲ و ۳ را نیاز داریم. بنابراین این ویژگی‌ها را در x قرار دادیم ❷. پس از آن برچسب‌ها (y) را ساختیم ❸. توجه کنید که برچسب‌ها را به‌صورت ۰ و ۱ کدگذاری کردیم. درنهایت ترتیب داده‌ها را به‌هم ریختیم و مجموعه داده

جدید را روی دیسک ذخیره کردیم ❹.در آخرین مرحله همین فرآیند را برای ساختن نمونه‌های تست تکرار کردیم ❺.

شکل ٦_٨ مجموعه آموزش را نشان می‌دهد. ما می‌توانیم این شکل را بکشیم زیرا تنها دو ویژگی داریم.



شکل ٦_٨: داده‌های آموزش با دو کلاس و دو ویژگی که از مجموعه داده گل زنبق بدست آمده است

در نگاه اول متوجه می‌شویم که مجموعه داده به سادگی قابل مجزا سازی نیست. هیچ خط ساده‌ای وجود ندارد که ما بتوانیم بکشیم به‌طوری که مجموعه آموزش را به‌درستی به دو گروه به‌صورتی تقسیم کند، که یک گروه شامل تمامی کلاس صفر و دیگری شامل تمامی کلاس یک شود.

پیاده‌سازی شبکه عصبی

در اینجا پیاده‌سازی شبکه عصبی شکل ١_٨ در پایتون با استفاده از Numpy را خواهیم دید. فرض می‌کنیم که این شبکه از قبل آموزش دیده است، یعنی از قبل تمام وزن‌ها و بایاس‌ها را می‌داند. کد مربوطه در لیست ٢_٨ آمده است.

```
import numpy as np
import pickle
import sys
def sigmoid(x):
     return 1.0 / (1.0 + np.exp(-x))
def evaluate(x, y, w):
❶ w12,b1,w23,b2,w34,b3 = w
  nc = nw = 0
  prob = np.zeros(len(y))
  for i in range(len(y)):
     a1 = sigmoid(np.dot(x[i], w12) + b1)
     a2 = sigmoid(np.dot(a1, w23) + b2)
     prob[i] = sigmoid(np.dot(a2, w34) + b3)
     z = 0 if prob[i] < 0.5 else 1
     ❷ if (z == y[i]):
          nc += 1
     else:
          nw += 1
  return [float(nc) / float(nc + nw), prob]
❸ xtest = np.load("iris2_test.npy")
ytest = np.load("iris2_test_labels.npy")
❹ weights = pickle.load(open("iris2_weights.pkl","rb"))
score, prob = evaluate(xtest, ytest, weights)
print()
for i in range(len(prob)):
     print("%3d: actual: %d predict: %d prob: %0.7f" %
     (i, ytest[i], 0 if (prob[i] < 0.5) else 1, prob[i]))
print("Score = %0.4f" % score)
```

لیست ۲_۸: استفاده از وزن‌ها و بایاس‌های بدست آمده از آموزش برای دسته‌بندی نمونه‌های تست.

nn_iris_evaluate.py را ببینید.

شاید اولین چیزی که متوجه بشویم این باشد که کد چقد کوتاه است. تابع evaluate شبکه را پیاده‌سازی می‌کند. همچنن باید تابع sigmoid را پیاده‌سازی کنیم زیرا NumPy این تابع را درون خود ندارد. قسمت main کد ، نمونه‌های تست (xtest) و برچسب‌های مربوط به آن‌ها (ytest) را بارگذاری می‌کند ❸. این‌ها فایل‌هایی هستند که در کد قبلی تولید شده بودند. بنابراین می‌دانیم که xtest دارای ابعاد 2×23 است زیرا ما 23 نمونه برای تست داریم و هرکدام دو ویژگی دارند. به‌همین ترتیب ytest یک بردار است که شامل 23 برچسب می‌باشد.

پس از آموزش این شبکه، وزن‌ها و بایاس‌ها را به‌صورت یک آرایه NumPy ذخیره می‌کنیم. روش

پایتون برای ذخیره‌سازی یک لیست در دیسک، استفاده از ماژول pickle است. بنابراین از pickle برای بارگذاری وزن‌ها از دیسک استفاده می‌کنیم ❹. لیستی که weights نام دارد، شامل شش عنصر است که سه عنصر آن ماتریس‌های وزن‌ها، و سه عنصر دیگر آن بردارهای بایاس شبکه هستند. این‌ها اعداد "جادویی[۱]" هستند از طریق آموزش شبکه به‌دست می‌آیند و مشروط به مجموعه داده هستند. درنهایت evaluate را فراخوانی می‌کنیم تا هرکدام از نمونه‌های تست را به شبکه بدهیم. این تابع، امتیاز (دقت[۲]) و احتمالات خروجی (prob) را برای هر نمونه بر می‌گرداند. مابقی کد شماره نمونه، برچسب واقعی، برچسب پیش‌بینی شده و احتمال محاسبه شده برای تعلق ورودی به کلاس ۱ را نشان می‌دهد. درنهایت امتیاز (دقت) نمایش داده می‌شود.

شبکه توسط evaluate پیاده‌سازی شده‌است. در اینجا درمورد چگونگی آن صحبت می‌کنیم. در ابتدا ماتریس‌های مربوط به وزن و بردارهای مربوط به بایاس را از لیست وزن‌ها، که ورودی تابع است، بدست می‌آوریم ❶. این موارد آرایه‌های NumPy هستند. `w12` یک ماتریس 3×2 است که نگاشت دو عنصر ورودی به سه گره موجود در لایه مخفی اول را انجام می‌دهد. `w23` یک ماتریس 2×3 است که نگاشت اولین لایه مخفی به دومین لایه مخفی را انجام می‌دهد. `w34` یک ماتریس 1×2 است که نگاشت دومین لایه مخفی به خروجی را انجام می‌دهد. بردارهای بایاس شامل `b1` با سه عنصر، `b2` با دو عنصر و `b3` با یک عنصر (اسکالر) می‌باشد.

توجه کنید که ابعاد ماتریس‌های وزن، مانند ابعادی که قبلاً گفته بودیم نیست بلکه ترآنهاده[۳] آن‌ها استفاده شده‌است. علت این امر این است که ما در این کد بردارها را در ماتریس‌ها ضرب می‌کنیم. استفاده از ترآنهاده ماتریس در اینجا باعث می‌شود که همان نتایج قبلی را بگیریم.

در مرحله بعدی evaluate تعداد صحیح (nc) و تعداد غلط (nw) را صفر قرار می‌دهد. این مقادیر برای محاسبه امتیاز کلی روی تمام مجموعه تست استفاده می‌شوند. به‌همین منوال، prob را تعریف کردیم که برداری برای نگه‌داشتن مقادیر خروجی احتمال برای هر نمونه از مجموعه تست است.

حلقه موجود، روی هر نمونه تست، شبکه را اعمال می‌کند. در ابتدا بردار ورودی را روی اولین لایه مخفی نگاشت می‌دهیم و `a1` را محاسبه می‌کنیم که شامل ۳ عدد است. این اعداد نشان دهنده خروجی تابع فعال‌سازی در هرکدام از گره‌های لایه مخفی اول هستند. سپس مقادیر فعال‌سازی اولین لایه مخفی را می‌گیریم و مقادیر فعال‌سازی دومین لایه مخفی `a2` را محاسبه می‌کنیم. `a2` یک بردار با دو عنصر است

---

[۱] magic

[۲] accuracy

[۳] transpose

زیرا در لایه مخفی دوم، دو گره داریم. سپس برای ورودی جاری مقدار خروجی را محاسبه می‌کنیم و در آرایه prob ذخیره می‌نماییم. برچسب کلاس یا z با بررسی این‌که آیا خروجی شبکه کمتر از 0.5 است یا خیر، به ورودی تخصیص داده می‌شود. درنهایت شمارنده‌های صحیح (nc) یا غلط (nw) را براساس برچسب واقعی نمونه (y[i]) افزایش می‌دهیم ❷. زمانی که تمامی نمونه‌ها وارد شبکه شدند، مقدار دقت کلی محاسبه می‌شود. این مقدار با تقسیم تعداد نمونه‌هایی که درست دسته‌بندی شده‌اند بر تعداد کل نمونه‌ها به‌دست می‌آید.

تا اینجای کار همه چیز خوب بوده است. ما می‌توانیم یک شبکه را پیاده‌سازی کنیم و بردارهای ورودی را به آن بدهیم و ببینیم که شبکه چقدر خوب کار می‌کند. اگر شبکه یک لایه مخفی سوم هم داشت، باید خروجی لایه مخفی دوم (a2) را به آن بدهیم و سپس مقدار خروجی را محاسبه کنیم.

آموزش و تست شبکه عصبی

کد داده شده در لیست ۲_۸، مدلی که از قبل آموزش داده شده بود را روی داده‌های تست پیاده‌سازی کرد. قبل از اینکار، برای آموزش مدل از sklearn استفاده می‌کنیم. کد برای آموزش مدل در لیست ۳_۸ آورده شده‌است.

```
import numpy as np
import pickle
from sklearn.neural_network import MLPClassifier
xtrain= np.load("iris2_train.npy")
ytrain= np.load("iris2_train_labels.npy")
xtest = np.load("iris2_test.npy")
ytest = np.load("iris2_test_labels.npy")
❶ clf = MLPClassifier(
      ❷ hidden_layer_sizes=(3,2),
      ❸ activation="logistic",
      solver="adam", tol=1e-9,
      max_iter=5000,
      verbose=True)
clf.fit(xtrain, ytrain)
prob = clf.predict_proba(xtest)
score = clf.score(xtest, ytest)
❹ w12 = clf.coefs_[0]
w23 = clf.coefs_[1]
w34 = clf.coefs_[2]
b1 = clf.intercepts_[0]
b2 = clf.intercepts_[1]
b3 = clf.intercepts_[2]
```

```
weights = [w12,b1,w23,b2,w34,b3]
pickle.dump(weights, open("iris2_weights.pkl","wb"))
print()
print("Test results:")
print(" Overall score: %0.7f" % score)
print()
for i in range(len(ytest)):
     p = 0 if (prob[i,1] < 0.5) else 1
     print("%03d: %d - %d, %0.7f" % (i, ytest[i], p, prob[i,1]))
print()
```

لیست ۳_۸: استفاده از sklearn برای آموزش شبکه عصبی با داده‌های گل زنبق. nn_iris_mlpclassifier.py را ببینید.

ابتدا داده‌های آموزش و تست را از دیسک بارگذاری می‌کنیم. این‌ها همان فایل‌هایی هستند که قبلاً ایجاد کرده بودیم. سپس یک شیء از کلاس شبکه عصبی MLPClassifie ایجاد می‌کنیم ❶. شبکه دارای دو لایه مخفی است. لایه اول سه گره و لایه دوم دو گره دارد ❷. این دقیقاً مشابه معماری شکل ۱_۸ است. همچنین این شبکه از لایه‌های logistic استفاده می‌کند ❸. لاجستیک نام دیگری برای لایه سیگموئید است. مانند بقیه مدل‌های sklearn، در اینجا نیز با استفاده از fit، مدل را آموزش می‌دهیم. از آنجایی که مقدار verbose را True قرار داده‌ایم، در خروجی مقدار loss را برای هر تکرار نمایش می‌دهد.

predict_proba احتمالات خروجی را روی داده‌های تست به ما می‌دهد. این تابع در تعداد زیادی از مدل‌های دیگر sklearn نیز پشتیبانی می‌شود. این مقدار نشان دهنده اطمینان مدل از برچسبی است که تخصیص داده. سپس از score استفاده می‌کنیم تا امتیاز را روی داده‌های تست محاسبه کند. این کار را قبلاً هم انجام داده بودیم.

ما می‌خواهیم تمامی وزن‌ها و بایاس‌ها را ذخیره کنیم تا بتوانیم برای کد تست استفاده کنیم. این مقادیر را می‌توانیم مستقیماً از مدل آموزش دیده شده استخراج کنیم ❹. همه‌ی این مقادیر در یک لیست (weights) تجمیع شدند و در یک فایل pickle پایتون ریخته شدند.

مابقی کد، نتایج اجرای مدل آموزش دیده شده روی داده‌های تست را نمایش می‌دهد. یک مثال از اجرای این کد نتایج زیر را داده است.

```
Test results:
Overall score: 1.0000000
000: 0 - 0, 0.0705069
001: 1 - 1, 0.8066224
002: 0 - 0, 0.0308244
003: 0 - 0, 0.0205917
```

```
004: 1 - 1, 0.9502825
005: 0 - 0, 0.0527558
006: 1 - 1, 0.9455174
007: 0 - 0, 0.0365360
008: 1 - 1, 0.9471218
009: 0 - 0, 0.0304762
010: 0 - 0, 0.0304762
011: 0 - 0, 0.0165365
012: 1 - 1, 0.9453844
013: 0 - 0, 0.0527558
014: 1 - 1, 0.9495079
015: 1 - 1, 0.9129983
016: 1 - 1, 0.8931552
017: 0 - 0, 0.1197567
018: 0 - 0, 0.0406094
019: 0 - 0, 0.0282220
020: 1 - 1, 0.9526721
021: 0 - 0, 0.1436263
022: 1 - 1, 0.9446458
```

این نتایج نشان می‌دهد که مدل برای مجموعه داده تست عالی عمل کرده است. در خروجی شماره نمونه، برچسب واقعی، برچسب تخصیص داده شده و خروجیِ احتمال تعلق داشتن به کلاس ۱ نشان داده شده‌است. اگر فایل pickle، که دربردارنده وزن‌ها و بایاس‌های مدل sklearn است را در کدِ ارزیابی که ایجاد کرده بودیم (لیست ۲_۸) قرار دهیم، خواهیم دید که احتمالات خروجی دقیقاً برابر مقادیر کد بالا خواهد شد. این موضوع نشان می‌دهد که شبکه عصبی که با دست خود از اول ساختیم درست کار می‌کند.

خلاصه

در این فصل درمورد آناتومی شبکه عصبی بحث کردیم. معماری، چینش گره‌ها و ارتباط بین آن‌ها را توصیف کردیم. گره‌های لایه خروجی و توابعی که محاسبه می‌کنند را مورد بحث قرار دادیم. سپس مشاهده کردیم که تمامی وزن‌ها و بایاس‌ها به راحتی می‌توانند توسط ماتریس‌ها و بردارها نمایش داده شوند. در آخر یک شبکه ساده برای دسته‌بندی یک زیرمجموعه از داده‌های گل زنبق ایجاد کردیم و نحوه آموزش و ارزیابی شبکه ایجاد شده را دیدم.

حال که راه را شروع کرده‌ایم، بیایید یک نگاه عمیق به تئوری موجود در پشت شبکه‌های عصبی بیاندازیم.