فصل ۹
آموزش یک شبکه عصبی

در این فصل، نحوه آموزش یک شبکه عصبی را مورد بحث قرار می‌دهیم. ما روش‌ها و تکنیک‌های استانداردی که امروزه مورد استفاده قرار می‌گیرند را بررسی می‌کنیم. مقداری ریاضیات و تعدادی مفهوم و اصطلاحات جدید در انتظار ما است. اما نیازی ندارید که ریاضیات را به‌طور عمیق دنبال کنید: ما مباحث را به‌اندازه‌ای بیان می‌کنیم که مورد نیاز است.

این فصل شاید، حداقل از لحاض مفاهیم، چالش برانگیزترین فصل در این کتاب باشد. از لحاض ریاضیاتی قطعاً چالش برانگیزترین است. در این فصل با وجود این که بدست آوردن بصیرت و درک کردن مفاهیم بسیار مهم است، گاهی وقت‌ها بی‌صبر می‌شویم و زودتر دست به کار می‌شویم تا مواردی را امتحان کنیم. این امر به‌دلیل وجود کتابخانه‌هایی که از قبل ایجاد شده‌اند، امکان‌پذیر است. اگر علاقه دارید که با شبکه‌های عصبی بازی کنید و قبل از یادگیری نحوه عملکردشان دستتان را گرم کنید، به فصل ۱۰ بروید، اما اگر این کار را کردید حتماً به این فصل بازگردید تا تئوری را یاد بگیرید.

این امکان وجود دارد که جعبه‌ابزارهایی[۱] مانند sklearn و Keras را بدون فهمیدن نحوه عملکردشان، یاد بگیرید. اگرچه این کار وسوسه کننده است اما کسی را راضی نمی‌کند.

فهمیدن نحوه عملکرد این الگوریتم‌ها کاملاً ارزش این را دارد که روی آن وقت بگذارید.

یک بررسی سطح بالا

این فصل را با یک بررسی اجمالی از مفاهیمی که بحث خواهیم کرد شروع می‌کنیم. این بررسی را مطالعه کنید اما اگر واضح نبود، نگران نباشید. سعی کنید حسی نسبت به فرآیند کلی بدست آورید.

اولین مرحله برای آموزش شبکه عصبی، انتخاب هوشمندانه‌ی مقادیر وزن‌ها و بایاس‌ها است. سپس از گرادیان کاهشی[۲] استفاده می‌کنیم تا این وزن‌ها و بایاس‌ها را به‌گونه‌ای اصلاح کنیم که خطا روی مجموعه

---

[۱] toolkit

[۲] gradient descent

آموزش[1] کاهش یابد. ما از میانگین تابع زیان استفاده می‌کنیم تا خطا را اندازه‌گیری کنیم. این خطا نشان می‌دهد که شبکه جاری چه مقدار اشتباه می‌کند. ما می‌توانیم بفهمیم که شبکه درست کار می‌کند یا اشتباه زیرا مقدار صحیح خروجی مورد انتظار از نمونه‌های تست را در دست داریم (برچسب کلاس داده‌های تست).

گرادیان کاهشی الگوریتمی است که نیازمند گرادیان است. می‌توانید به گرادیان به‌عنوان شیب نگاه کنید. هرچه گرادیان بیشتر باشد، تابع در آن نقطه پرشیب‌تر است. ما از گرادیان کاهشی استفاده می‌کنیم تا کمترین مقدار تابع زیان را بدست بیاوریم. برای این کار نیاز داریم که امکان پیدا کردن گرادیان‌ها را داشته باشیم. برای بدست آوردن گرادیان‌ها از پس‌انتشار[2] استفاده می‌کنیم. این همان الگوریتم پایه‌ای شبکه‌های عصبی است که به آن‌ها این امکان را می‌دهد که با موفقیت آموزش ببینند. این روش برای بدست آوردن گرادیان‌هایی که نیاز داریم، از خروجی شبکه شروع می‌کند و با طی کردن شبکه رو به عقب، به سمت ورودی حرکت می‌کند. در طی مسیر، گرادیان‌های مربوط به هر وزن و بایاس را محاسبه می‌کند.

با داشتن مقادیر گرادیان، می‌توانیم با استفاده از الگوریتم گرادیان کاهشی، به‌گونه‌ای وزن‌ها و بایاس‌ها را تغییر دهیم که دفعه بعدی، وقتی داده‌های آموزش را به شبکه می‌دهیم، تابع زیان مقدار کمتری از قبل داشته باشد. به عبارت دیگر شبکه عصبی ما کمتر اشتباه خواهد کرد. این دقیقاً معنای آموزش است و امیدواریم که این کار باعث شود که یک شبکه داشته باشیم که ویژگی‌های قابل تعمیم داده را یاد گرفته باشد.

یاد گرفتن ویژگی‌های قابل تعمیم در مجموعه داده[3] نیازمند قاعده‌سازی[4] است. روش‌های زیادی برای قاعده‌سازی وجودی دارد و ما موارد اصلی را بحث خواهیم کرد. بدون قاعده‌سازی، فرآیند آموزش در خطر بیش‌برازش[5] است. در این صورت ممکن است با شبکه‌ای روبرو شویم که توان تعمیم نداشته باشد. با وجود قاعده‌سازی می‌توانیم یک مدل کاربردی بدست بیاوریم.

بنابراین، بخش‌های بعدی درمورد گرادیان کاهشی، پس‌انتشار، تابع زیان[6]، مقداردهی اولیه وزن‌ها[7] و

---

[1] Training set

[2] backpropagation.

[3] Data set

[4] Regularization (مترجم: این کلمه منظم سازی یا باقاعده‌سازی نیز ترجمه شده‌است اما ترجیح این است که از کلمه انگلیسی استفاده شود)

[5] overfitting

[6] Loss function

[7] Weight initialization

درنهایت قاعده‌سازی است. این‌ها اجزای اصلی یک آموزش موفق شبکه عصبی هستند. ما به تمامی جزئیات ریاضیاتی دشوار این مباحث نیاز نداریم و درعوض نیاز داریم که مفاهیم آن‌ها را متوجه شویم تا بتوانیم یک بینش نسبت به آموزش شبکه عصبی بدست بیاوریم. با بینش بدست آمده می‌توانیم استفاده بهتری از پارامترهایی که sklearn و Keras در اختیار ما قرار می‌دهند، بکنیم.

گرادیان کاهشی

راه استاندارد برای آموزش یک شبکه عصبی استفاده از گرادیان کاهشی است.

بیایید اجزای عبارت "گرادیان کاهشی" را بررسی کنیم. ما می‌دانیم که کلمه "کاهشی" چه معنایی دارد. کاهشی (مترجم: یا همان نزولی) به‌معنای پایین رفتن از یک جای بلندتر است. پاسخ کوتاه به معنی "گرادیان" این است: گرادیان نشان می‌دهد که یک چیز، نسبت به سرعت تغییر یک چیز دیگر، چقدر سریع تغییر می‌کند. اندازه‌گیری میزان تغییر یک چیز نسبت به تغییرات یک چیز دیگر برای همه ما آشنا است. همه ما می‌دانیم سرعت چیست. سرعت میزان تغییر موقعیت نسبت به تغییر زمان است. ما حتی این موضوع را در واحد آن انعکاس داده‌ایم: مایل بر ساعت یا کیلومتر بر ساعت.

احتمالاً شما با گرادیان در یک جای دیگر آشنا هستید. معادله یک خط را درنظر بگیرید.

$$y = mx + b$$

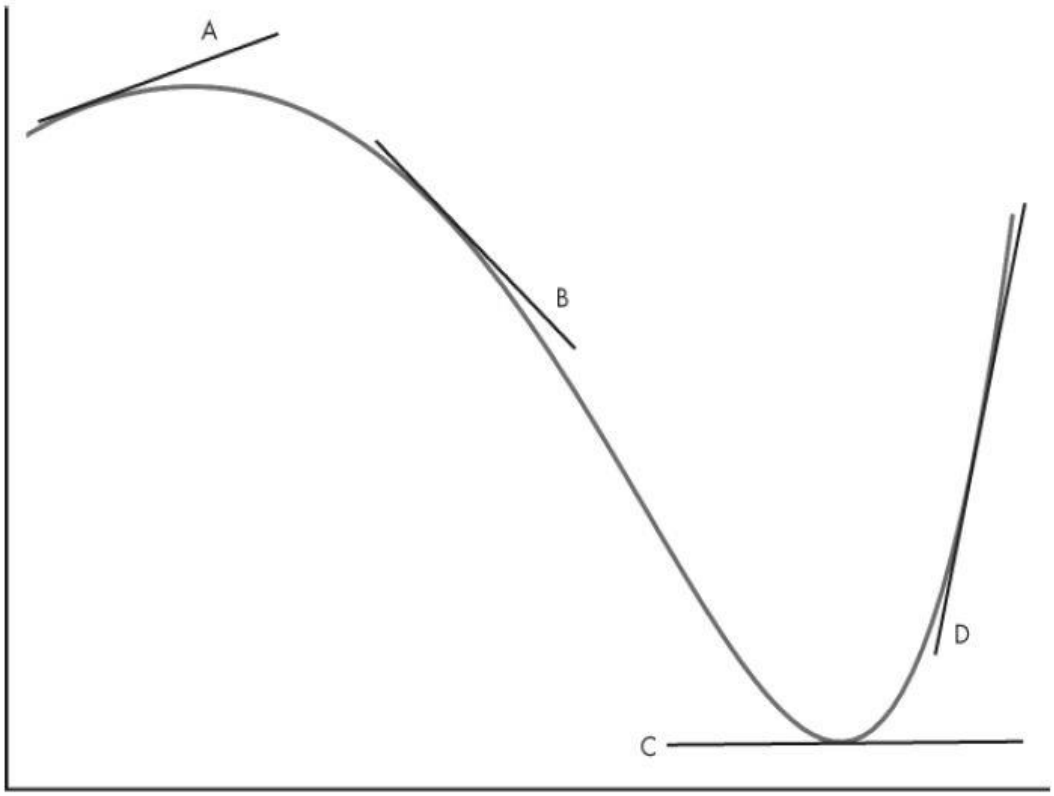
که در آن m شیب و b محل تقاطع خط با محور y (عرض از مبدأ خط) است. شیب نشان می‌دهد که تغییرات موقعیت y، نسبت به هر واحد تغییرات در موقعیت x چقدر سریع است. اگر ما دو نقطه از خط (x_0, y_0) و (x_1, y_1) را داشته باشیم، می‌توانیم شیب را به‌صورت زیر محاسبه کنیم:

$$m = \frac{y_0 - y_1}{x_0 - x_1}$$

می‌توان گفت واحد آن y بر x است. این یک سنجه برای میزان تند بودن شیب یک خط است: این یک گرادیان است. در ریاضیات گاهی وقت‌ها از تغییرات در یک متغیر صحبت می‌کنیم و نحوه نمایش آن با قرار دادن Δ (دلتا) پشت متغیر است. درنتیجه می‌توانیم شیب یک خط را به‌صورت زیر نشان دهیم.

$$m = \frac{\Delta y}{\Delta \mathrm{x}}$$

درواقع شیب، میزان تغییرات در y به‌ازای هر واحد تغییر در x است. خوشبختانه مشخص شده‌است که نه تنها خط‌ها، بلکه اکثر توابع در هر نقطه یک شیب دارند. غیر از خطوط مستقیم، این شیب از نقطه‌ای به نقطه دیگر تغییر می‌کند. یک تصویر می‌تواند کمک کننده باشد. شکل ۱_۹ را درنظر بگیرید.



شکل ۱_۹: یک تابع که چندین خط مماس روی آن مشخص شده

نمودار موجود در شکل ۱_۹ مربوط به یک تابع چندجمله‌ای است. توجه کنید که خطوطی که در شکل کشیده شده‌اند تنها تابع را لمس کرده‌اند. این‌ها خطوط مماس[۱] هستند. همانطور که در شکل دیده می‌شود، چون این‌ها خط هستند پس شیب هم دارند. تصور کنید که یک خط را روی تابع به‌گونه‌ای حرکت می‌دهید که همواره فقط در یک نقطه تابع را لمس می‌کند، تصور کنید وقتی خط روی نقاط مختلف تابع حرکت می‌کند، شیب این خط چگونه تغییر می‌کند.

مشخص شده‌است که نحوه تغییر شیب در طول یک تابع، خود یک تابع است که مشتق[۲] نام دارد. اگر یک تابع و مقدار x را داشته باشیم، شیب تابع در آن نقطه از x برابر با مشتق در آن نقطه است. این که توابع مشتق دارند یکی از بینش‌های مهم ریاضیات است و برای ما هم بسیار مهم است.

وجود ایده مشتق ضروری است زیرا برای یک تابع تک متغیره، مشتق در نقطه x، همان گرادیان در نقطه

---

۱ tangent

۲ derivative

x است. درواقع جهتی است که تابع در آن جهت در حال تغییر است. اگر حداقل مقدار یک تابع را بخواهیم، یا درواقع مقداری از x که کمترین y را بدهد، باید در خلاف جهت گرادیان حرکت کنیم. زیرا این جهت ما را به سمت حداقل می‌برد.

مشتق به طرق مختلفی نوشته می‌شود. اما اگر بخواهیم آن را به‌صورتی بنویسیم که نشان‌دهنده ایده شیب (نحوه تغییر y به‌ازای تغییر در x) باشد به‌صورت زیر خواهد بود.

$$\frac{dy}{dx}$$

زمانی که الگوریتم پس‌انتشار را مورد بحث قرار دهیم، به این مورد بازخواهیم گشت. همین مقدار برای "گرادیان" کافی است. بیاید یک نگاه دقیق‌تر به "کاهشی" بیاندازیم.

پیدا کردن حداقل‌ها

از آنجایی که مدلی می‌خواهیم که اشتباهات کمی مرتکب شود، نیاز داریم یک مجموعه از پارامترها را به‌گونه‌ای پیدا کنیم که منجر به مقادیر کمی از تابع زیان می‌شوند. به‌عبارت دیگر، نیاز داریم که حداقل مقدار را برای تابع زیان پیدا کنیم.

دوباره به شکل ۱_۹ نگاه کنید. حداقل مقدار در سمت راست است، همان جایی که خط مماس C وجود دارد. ما می‌توانیم ببینیم که این مقدار، حداقل است و متوجه می‌شویم که گرادیان در آن‌جا صفر است. مقدار صفر برای گرادیان نشان‌دهنده این است که ما در حداقل (یا حداکثر) هستیم. اگر از B شروع کنیم می‌بینیم که شیب خط مماس منفی است (رو به پایین و به سمت راست). بنابراین باید در جهت مثبت x (مترجم: جهت مثبت محور xها) حرکت کنیم زیرا مخالف علامت گرادیان است. با این کار به مقدار حداقل در C نزدیک‌تر می‌شویم.

به‌طور مشابه اگر از D شروع کنیم، شیب خط مماس مثبت است (رو به بالا و به سمت راست). این بدین معناست که باید در جهت منفی x حرکت کنیم تا به مقدار حداقل در C نزدیک‌تر شویم.

اگر بخواهیم تمامی این نکات را به‌صورت یک الگوریتم برای پیدا کردن حداقل یک تابع به‌کار ببریم باید بگوییم: یک نقطه شروع انتخاب کنید (یک مقدار برای x) و از گرادیان استفاده کنید تا به طرف نقطه‌ی پایین‌تر حرکت کنید.

برای توابعی که فقط تابع یک متغیر x هستند، مانند شکل ۱_۹، با فرض انتخاب نقطه شروع خوبی مانند B یا D این روش خیلی خوب کار می‌کند. زمانی که بیش از یک بعد داشته باشیم، مشخص شده‌است که این روش با فرض شروع از یک محل مناسب باز هم خوب کار می‌کند.

اگر دوباره به شکل ۱_۹ نگاه کنید، با فرض این که از نقطه B شروع کنیم، می‌بینیم که گرادیان به ما می‌گوید که به سمت راست حرکت کنیم که دقیقاً به طرف C است. اما چگونه باید مقدار x بعدی را به‌گونه‌ای مشخص کنیم که به C نزدیکتر باشد؟ این موضوع مربوط به اندازه گام[۱] است. اندازه گام به ما می‌گوید که پرش ما از یک مقدار x به مقدار بعدی x باید چقدر بزرگ باشد. اندازه گام پارامتری است که ما باید انتخاب کنیم. در عمل این پارامتر که نرخ یادگیری[۲] نیز نامیده می‌شود، ثابت نیست و هرچه بیشتر حرکت می‌کنیم کوچک و کوچکتر می‌شود. این کار با این فرض انجام می‌شود که هرچه ما بیشتر حرکت می‌کنیم، به مقدار حداقل نزدیک و نزدیک تر می‌شویم؛ بنابراین گام‌های کوچک و کوچکتری نیاز داریم.

تا اینجای کار همه چیز خوب بوده است. اما یک مشکل کوچک وجود دارد. اگر به جای شروع از B یا D، از A شروع کرده بودیم چه اتفاقی می‌افتاد؟ گرادیان در A ما را به سمت چپ هدایت می‌کند و نه راست. در این حالت، الگوریتم ساده ما شکست خواهد خورد. این الگوریتم ما را به سمت چپ هدایت می‌کند و ما هیچ وقت به C نخواهیم رسید. در شکل، تنها یک مقدار حداقل نشان داده شده‌است؛ اما ما می‌توانیم به‌سادگی یک مقدار حداقل دیگر در سمت چپ A متصور شویم که پایین‌تر از C نمی‌رود (مقدار y آن به کوچکی مقدار y در C نیست). اگر از A شروع کنیم به سمت این مقدار حداقل حرکت می‌کنیم و نه C. الگوریتم ما در یک حداقل محلی[۳] گیر می‌افتد. وقتی الگوریتم در این حداقل محلی وارد شود، نمی‌تواند از آن خارج شود و ما قادر نخواهیم بود که حداقل سراسری[۴] در C را ببینیم. خواهیم دید که این موضوع یک مشکل واقعی برای شبکه‌های عصبی است.

درنهایت چگونه تمامی این موارد به ما کمک می‌کنند که شبکه عصبی را آموزش دهیم؟ گرادیان به ما می‌گوید که یک تغییر کوچک در x، چگونه y را تغییر می‌دهد. اگر x یکی از پارامترهای شبکه باشد و y خطایی باشد که توسط تابع زیان تعیین می‌شود، آنگاه گرادیان به ما می‌گوید که یک تغییر در این پارامتر چقدر خطای کلی شبکه را تحت تأثیر قرار می‌دهد. حال که این را فهمیدیم، در موقعیتی هستیم که براساس مقدار گرادیان به‌گونه‌ای پارامتر را اصلاح کنیم که ما را به سمت یک مقدار حداقل از خطا هدایت کند. وقتی که خطا روی مجموعه آموزش حداقل شود می‌توانیم بگوییم که شبکه آموزش دیده است.

بیایید بیشتر درمورد گرادیان و پارامترها صحبت کنیم. تمامی بحث‌های ما تا اینجای کار، براساس شکل ۱_۹، یک بعدی بوده است. درواقع تابع، تنها تابعی از یک x بوده است. ما درمورد تغییر تنها یک چیز

---

[۱] step size

[۲] learning rate

[۳] local minimum

[۴] global minimum

صحبت کردیم و آن هم تغییر مقدار x بود تا ببینیم مقدار y چگونه تغییر می‌کند. در واقعیت تنها با یک بعد کار نخواهیم کرد. هر وزن و بایاس در شبکه ما یک پارامتر است و مقدار تابع زیان به همه‌ی آن‌ها وابسته است. برای شبکه ساده شکل ۱_۸، بیست پارامتر وجود دارد و این یعنی تابع زیان یک تابع بیست بعدی است. با این وجود الگوریتم ما یکسان باقی خواهد ماند. اگر ما گرادیان برای هر پارامتر را بدانیم، باز هم می‌توانیم الگوریتم مان را اعمال کنیم تا یک مجموعه از پارامترها را به‌گونه‌ای بیابیم که مقدار زیان کمینه شود.

به‌روزرسانی وزن‌ها

در آینده‌ای نزدیک خواهیم دید که چگونه مقادیر گرادیان را بدست بیاوریم. تا آن زمان فرض کنید این مقادیر را داریم. با وجود پیکربندی جاری شبکه، مقادیر گرادیان اعدادی هستند که نشان می‌دهند که تغییر در هر کدام از وزن‌ها یا بایاس‌ها چگونه مقدار تابع زیان را تغییر می‌دهد. با این دانش می‌توانیم گرادیان کاهشی را اعمال کنیم: ما وزن‌ها و بایاس‌ها را، به‌صورت یکجا و همزمان، با کسری از مقدار گرادیان به‌گونه‌ای اصلاح می‌کنیم که ما را به سمت حداقل مقدار کل تابع زیان هدایت کند.

از لحاض ریاضی هر وزن و بایاس را با استفاده از یک قانون ساده به‌روزرسانی می‌کنیم:

$$w \leftarrow w - \eta \Delta w$$[1]

در اینجا w یکی از وزن‌ها (یا بایاس‌ها) است، η (اتا) نرخ یادگیری (یا اندازه گام) و Δw مقدار گرادیان است.

لیست ۹ یک الگوریتم برای آموزش یک شبکه عصبی براساس گرادیان کاهشی را ارائه داده است.

---

۱. مقادیر هوشمندانه اولیه‌ای برای وزن‌ها و بایاس‌ها انتخاب کنید.
۲. مجموعه آموزش را روی شبکه براساس وزن‌ها و بایاس‌های جاری اجرا کنید و مقدار متوسط زیان را محاسبه نمایید.
۳. از این مقدار زیان برای بدست آوردن گرادیان برای هر وزن و بایاس استفاده کنید.
٤. مقدار وزن یا بایاس را با ضرب کردن اندازه گام در مقدار گرادیان، به‌روزرسانی کنید.
۵. از مرحله ۲ تکرار کنید تا مقدار زیان به اندازه کافی کم باشد.

---

[1] مترجم: در کتاب این فرمول به‌صورت $w \leftarrow w - \Delta w$ بیان شده‌است که پارامتر η را ندارد اما به‌نظر من اشتباه است. شما می توانید فرمولی که در کتاب است را خودتان مشاهده کنید و تصمیم بگیرید که کدام فرمول درست است.

لیست ۱_۹: گرادیان کاهشی (نه چندان دقیق) در ۵ مرحله ساده

الگوریتم به‌نظر ساده می‌آید اما مشکلات در جزئیات است. ما باید در هر مرحله انتخاب‌هایی بکنیم و هر انتخابی که می‌کنیم سؤالات دیگری را ایجاد می‌کنند. به‌عنوان مثال، مرحله ۱ می‌گوید "مقادیر هوشمندانه اولیه‌ای انتخاب کنید". این مقادیر چقدر باید باشند؟ مشخص شده‌است که آموزش موفق یک شبکه عصبی، کاملاً وابسته به انتخاب مقادیر اولیه خوب است. خودمان هم این موضوع را طی شکل ۱_۹ دیدیم. در این شکل با شروع از A به حداقل مقدار در C نخواهیم رسید. در طی سالیان تحقیقات زیادی انجام شده‌است که مربوط به مرحله ۱ می‌شود.

مرحله ۲ سرراست است. در این مرحله رو به جلو، شبکه را طی می‌کنیم. ما درمورد جزئیات تابع زیان صحبت نکرده‌ایم. برای این مرحله فرض کنید تابع زیان تابعی است که میزان مؤثر بودن شبکه را روی مجموعه آموزش اندازه‌گیری می‌کند.

مرحله ۳ در اینجای کار ناشناخته است. به‌زودی درمورد آن صحبت خواهیم کرد. در اینجا فرض کنید که می‌توانیم مقادیر گرادیان را برای هر پارامتر بدست بیاوریم.

مرحله ٤ کار فرمول قبلی را انجام می‌دهد. در این مرحله به‌گونه‌ای مقدار جدید پارامتر براساس مقدار فعلی آن بدست می‌آید که زیان کلی کاهش پیدا کند. در عمل این فرمول ساده کافی نیست. موارد دیگری مانند تکانه[۱] نیز وجود خواهند داشت. تکانه کسری از تغییر وزن قبلی را برای تکرار بعدی (دفعه بعدی که داده‌های آموزش از شبکه عبور داده می‌شوند) نگه می‌دارد تا پارامترها با شدت زیادی تغییر نکنند. بعداً تکانه را دوباره خواهیم دید. اکنون بیاید به یکی از انواع گرادیان کاهشی که برای آموزش شبکه‌های عمیق استفاده می‌شود، نگاهی بیاندازیم.

گرادیان کاهشی تصادفی[۲]

مراحل قبلی، آموزش یک شبکه عصبی از طریق گرادیان کاهشی را توصیف می‌کنند. همانطور که انتظارش را داریم، در عمل روش‌های بسیار زیادی از این ایده پایه‌ای نشأت گرفته‌اند. یکی از این روش‌ها که مرسوم شده‌است و به‌صورت تجربی خوب کار می‌کند گرادیان کاهشی تصادفی است که به‌صورت مخفف با SGD نشان داده می‌شود. کلمه تصادفی به یک فرآیند تصادفی اشاره دارد. در بخش بعدی

[۱] momentum

[۲] Stochastic Gradient Descent

خواهیم دید که چرا کلمه تصادفی بعد از عبارت گرادیان کاهشی آمده است.

دسته‌ها[1] و ریزدسته‌ها[2]

مرحله دوم لیست ۱_۹ می‌گوید که تمامی مجموعه آموزش را با استفاده از مقادیر فعلی وزن‌ها و بایاس‌ها از شبکه عبور دهید. این روش آموزش دسته‌ای[3] نام دارد. این نام به این دلیل گذاشته شده‌است که ما از تمامی داده‌های آموزش استفاده می‌کنیم تا گرادیان‌ها را تخمین بزنیم. این کار منطقی است. ما داده‌های آموزش را به‌گونه‌ای جمع آوری کرده‌ایم که نشان‌دهنده فرآیند ناشناخته والدی باشد که داده‌ها را تولید می‌کند. ما از شبکه انتظار داریم که این فرآیند والد را با موفقیت مدل کند.

اگر مجموعه داده ما، مانند مجموعه داده گل‌های زنبق در فصل ۵، کوچک باشد استفاده از آموزش دسته‌ای منطقی است. اما اگر مجموعه داده ما کوچک نباشد چه؟ اگر صدها هزار یا حتی میلیون‌ها نمونه داشته باشیم چه؟ ما با زمان آموزش طولانی و طولانی‌تری مواجه خواهیم شد.

ما به یک مشکل خوردیم. ما یک مجموعه آموزش بزرگ می‌خواهیم به این دلیل که فرآیند ناشناخته والدی که می‌خواهیم مدل کنیم را بهتر نشان دهد. اما هرچه مجموعه آموزش بزرگتر شود، عبور دادن تمامی نمونه‌ها از شبکه، گرفتن میانگین روی مقدار زیان و به‌روزرسانی وزن‌ها و بایاس‌ها بیشتر طول خواهد کشید. ما عبور دادن تمامی مجموعه آموزش از شبکه را یک دوره[4] (اپوک) می‌نامیم. ما تعداد زیادی از این اپوک‌ها را برای آموزش شبکه نیاز داریم. داشتن یک نمایش بهتر از چیزی که می‌خواهیم مدل کنیم (مترجم: مجموعه داده آموزش بزرگتر)، به معنی زمان محاسباتی طولانی و طولانی‌تر است زیرا تمامی نمونه‌ها باید از شبکه عبور داده شوند.

اینجا، جایی است که SGD نقش بازی می‌کند. به جای عبور تمامی داده‌های آموزش، یک زیرمجموعه کوچک از داده‌های آموزش را جدا کنید و از میانگین زیان محاسبه شده از روی آن‌ها، پارامترها را به‌روزرسانی کنید. با این کار گرادیان را "اشتباه" بدست خواهیم آورد زیرا با این کار ما مقدار زیان روی تمامی داده‌های آموزش را براساس تنها یک نمونه کوچک از داده‌های آموزش تخمین خواهیم زد؛ اما زمان زیادی را صرفه‌جویی خواهیم کرد.

بیایید با یک مثال ساده ببینیم که این نمونه‌گیری چگونه کار می‌کند. ما با استفاده از NumPy یک بردار

---

[1] batch

[2] Minibatches

[3] Batch training

[4] Epoch (مترجم: ترجمه این کلمه دوره است، اما در متن از همان اپوک استفاده خواهد شد)

از ۱۰۰ بایت تصادفی ایجاد می‌کنیم.

```
>>> d = np.random.normal(128,20,size=100).astype("uint8")
>>> d
130, 141, 99, 106, 135, 119, 98, 147, 152, 163, 118, 149, 122,
133, 115, 128, 176, 132, 173, 145, 152, 79, 124, 133, 158, 111,
139, 140, 126, 117, 175, 123, 154, 115, 130, 108, 139, 129, 113,
129, 123, 135, 112, 146, 125, 134, 141, 136, 155, 152, 101, 149,
137, 119, 143, 136, 118, 161, 138, 112, 124, 86, 135, 161, 112,
117, 145, 140, 123, 110, 163, 122, 105, 135, 132, 145, 121, 92,
118, 125, 154, 148, 92, 142, 118, 128, 128, 129, 125, 121, 139,
152, 122, 128, 126, 126, 157, 124, 120, 152
```

در این جا مقادیر بایت‌ها از یک توزیع نرمال با میانگین ۱۲۸ پیروی می‌کنند. میانگین واقعی این نمونه ۱۰۰ تایی، 130.9 است. انتخاب زیرمجموعه‌های ۱۰ تایی از این مقادیر به ما یک تخمین از میانگین واقعی این نمونه (130.9) می‌دهد.

```
>>> i = np.argsort(np.random.random(100))
>>> d[i[:10]].mean()
138.9
```

با تکرار این زیرمجموعه‌ها، تخمین میانگین به‌صورت 135.7، 131.7، 134.2، 128.1 و الی آخر بدست آمد. هیچ کدام از میانگین‌های تخمین زده شده برابر مقدار واقعی نیستند اما همه آن‌ها به مقدار واقعی نزدیک هستند. حال که می‌توانیم میانگین را از روی یک زیرمجموعه تصادفی از کل داده‌ها بدست بیاوریم، به‌طور مشابه می‌توانیم گرادیان‌های تابع زیان را از زیرمجموعه‌ای از تمامی داده‌های آموزش بدست بیاوریم. از آن‌جایی که نمونه به‌صورت تصادفی انتخاب شده‌است، گرادیان‌های بدست آمده نیز، تخمین‌هایی هستند که ماهیت تصادفی دارند. به همین دلیل است که کلمه تصادفی را بعد از عبارت گرادیان کاهشی قرار دادیم.

وقتی برای آموزش و در هر بار به‌روزرسانی وزن‌ها و بایاس‌ها، تمامی داده‌های آموزش را به شبکه بدهیم، آموزش دسته‌ای نام دارد و وقتی یک زیرمجموعه از داده‌ها را به شبکه بدهیم آموزش با دسته‌های کوچک یا آموزش ریزدسته‌ای[۱] (minibatch) نام دارد. شما مشاهده می‌کنید که افراد کلمه minibatch را زیاد استفاده می‌کنند. یک minibatch، یک زیرمجموعه از داده‌های آموزش است که برای هر مرحله از گرادیان کاهشی تصادفی استفاده می‌شود. آموزش معمولاً در تعدادی اپوک انجام می‌شود به‌طوری که رابطه بین اپوک‌ها و minibatchها به‌صورت زیر است.

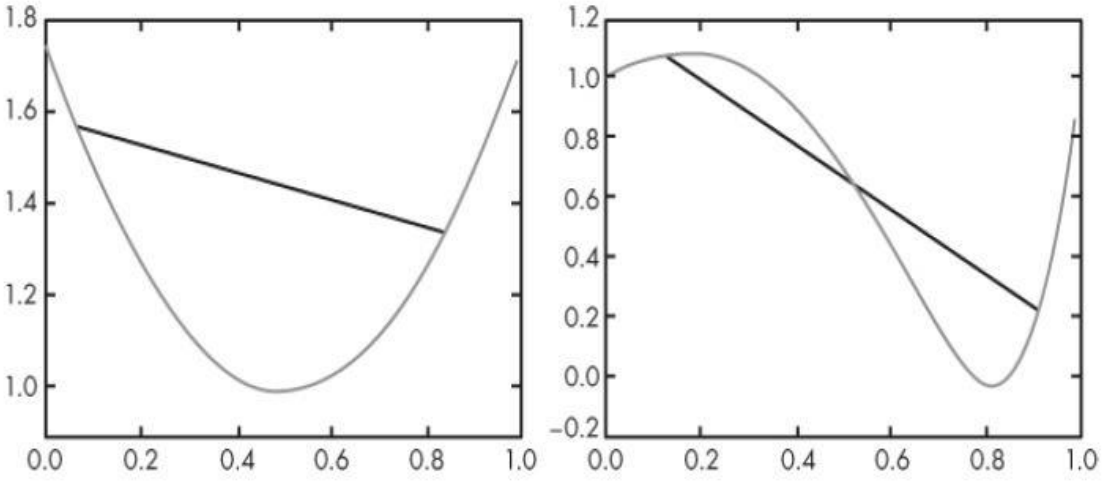
[۱] minibatch training (مترجم: برای این کلمه ترجمه مرسومی وجود نداشت ولی نزدیک ترین ترجمه همان آموزش ریزدسته‌ای است. ترجیح بر این است که از همان کلمه انگلیسی استفاده شود.)

$$1epoch = \left(\frac{\text{تعداد نمونه‌های مجموعه آموزش}}{\text{اندازه minibatch}} \right) \text{minibatches}$$

در عمل نمی‌خواهیم که minibatchها را به‌صورت تصادفی از کل مجموعه آموزش انتخاب کنیم. اگر این کار را بکنیم این ریسک وجود دارد که از همه نمونه‌ها استفاده نشود. یعنی تعدادی از نمونه‌ها ممکن است هیچ‌گاه انتخاب نشوند درحالی که تعدادی دیگر، بارها انتخاب شوند. معمولاً در ابتدا ترتیب نمونه‌های آموزش را به‌صورت تصادفی تغییر می‌دهیم و بلوک‌هایی از نمونه را با اندازه ثابت و به ترتیب، به‌عنوان minibatch انتخاب می‌کنیم. وقتی تمامی نمونه‌های آموزش استفاده شدند، می‌توانیم ترتیب کل داده‌های آموزش را به هم بریزیم و دوباره همین فرآیند را تکرار کنیم. برخی از جعبه‌ابزارها (کتابخانه‌ها) یادگیری عمیق، حتی این کار را نمی‌کنند، درعوض همان مجموعه از minibatchها را به‌صورت دایره‌وار استفاده می‌کنند (مترجم: منظور این است که در برخی از کتابخانه‌ها، الگوریتم به‌گونه‌ای نوشته شده‌است که پس از انتخاب آخرین minibatch ترتیب داده‌ها عوض نمی‌شود و minibatchهای جدید تعریف نمی‌شوند، بلکه همان minibatchهای سابق دوباره به‌کار می‌روند).

توابع محدب و غیرمحدب

SGD را می‌توان به‌عنوان یک کمک حال برای استفاده عملی از شبکه‌های عصبی درنظر گرفت. در تئوری دوست نداریم که از SGD استفاده کنیم و انتظار داریم که نتایج آموزش، از آن متضرر شوند. به هر حال، این مخالفت در کل درست است. می‌توان گفت که آموزش شبکه عصبی با گرادیان کاهشی نباید اصلاً کار کند زیرا این الگوریتم در اصل برای توابع محدب طراحی شده‌است اما ما در حال پیاده‌سازی آن روی تابعی غیرمحدب هستیم. شکل ۲_۹ تفاوت یک تابع محدب و غیرمحدب را نشان می‌دهد.



شکل ۲_۹: یک تابع محدب از x (چپ). یک تابع غیرمحدب از x (راست)

یک تابع محدب به‌گونه‌ای است که اگر یک وتر بین هر دو نقطه‌ای از تابع رسم کنیم، هیچ نقطه دیگری از تابع را قطع نکند. خط سیاه در نمودار سمت چپ موجود در شکل ۲_۹ یک مثال برای همین مورد است. در این نمودار هر وتر مشابهی رسم شود، در هیچ نقطه دیگری تابع را قطع نخواهد کرد، و این بدین معناست که این تابع محدب است. نمی‌توان صحت این موضوع را برای شکل سمت راست تأیید کرد. این همان نموداری است که در شکل ۱_۹ وجود داشت. در اینجا خط سیاه تابع را قطع کرده است.

گرادیان کاهشی برای پیدا کردن مقدار حداقل در توابعی طراحی شده‌است که محدب هستند. از آنجایی که این روش تنها به گرادیان یا همان مشتق اول متکی است، گاهاً به‌عنوان روش بهینه‌سازی مرتبه اول نیز شناخته می‌شود. در کل گرادیان کاهشی نباید برای توابع غیرمحدب کار کند زیرا خطر این وجود دارد که به‌جای پیدا کردن حداقل سراسری، در حداقل محلی به دام بیوفتد. ما این موضوع را در مثال مربوط به شکل ۱_۹ دیدیم.

در این جا گرادیان کاهشی تصادفی کمک می‌کند. در حالتی که چندین بعد داریم، گرادیان به سمتی اشاره می‌کند که لزوماً نزدیک‌ترین حداقل در تابع زیان نیست. به این معنی که در این مرحله، ممکن است جهت محاسبه شده مقدار کمی اشتباه باشد، اما همین که مقداری اشتباه داریم ممکن است کمک کند که در جایی که دوست نداریم (مترجم: همان حداقل محلی) به دام نیوفتیم.

موقعیت بیشتر از این‌ها پیچیده و اسرارآمیز است. از یک طرف در عمل موفقیت کامل بهینه‌سازی مرتبه اول روی توابع غیرمحدب مشاهده می‌شود و از طرف دیگر این واقعیت وجود دارد که این روش نباید اصلاً کار کند. افرادی که در حوزه یادگیری ماشین فعالیت می‌کنند با این تناقض در حال دست و پنجه نرم کردن هستند. دو ایده مطرح می‌شود. اولین ایده چیزی است که بیان کردیم: گرادیان کاهشی تصادفی با هدایت کردن ما در جهتی که مقداری اشتباه است، به ما کمک می‌کند. ایده دوم امروزه بیشتر اثبات

شده‌است. این ایده می‌گوید که مشخص شده‌است توابع زیانی که در یادگیری عمیق استفاده می‌شوند تعداد بسیار بسیار زیادی حداقل محلی دارند و اساساً همه آن‌ها یکسان هستند. بنابراین اگر در هرکدام از آن‌ها قرار بگیریم شبکه‌ای خواهیم داشت که عالی عمل می‌کند.

بعضی از محققان اشاره کرده‌اند که اکثر یادگیری‌های گرادیان کاهشی به یک نقطه زینی[۱] منتهی می‌شود. نقطه زینی نقطه‌ای است که شبیه به حداقل است اما حداقل نیست. یک زین اسب را تصور کنید و یک مهره را در وسط آن قرار دهید. مهره در یک نقطه خواهد ایستاد، اما شما می‌توانید مهره را در یک جهت خاص فشار دهید تا از روی زین بیوفتد (مترجم: در اینجا منظور نویسنده این است که نقطه‌ی زینی شبیه حداقل است زیرا مهره در آن‌جا ساکن می‌شود اما مقادیر کمتری از آن هم وجود دارد). ادعای محققان این است که اکثر آموزش‌ها به یک نقطه زینی منتهی می‌شوند، و نتیجه بهتری هم وجود دارد که می‌توان با یک الگوریتم بهتر به آن‌ها رسید. به هرحال، اگر هم یک نقطه زینی وجود داشته باشد باز هم برای اهداف عملی نقطه خوبی است و مدل به خوبی کار می‌کند.

بنابراین در عمل باید از گرادیان کاهشی تصادفی استفاده کنیم. زیرا استفاده نکردن از دسته‌های کامل، منجر به آموزش بهتر می‌شود و زمان آموزش را کاهش می‌دهد. این روش یک پارامتر جدید را وارد معادلات می‌کند و آن هم اندازه minibatch است. این مقدار باید قبل از آموزش انتخاب شود.

اتمام آموزش

ما هنوز درمورد یک پرسش اساسی بحث نکرده‌ایم: چه زمانی باید آموزش را متوقف کنیم؟ به‌یاد بیاورید که در فصل ۵ تلاش کردیم که مجموعه آموزش، مجموعه ارزیابی[۲] و مجموعه تست را ایجاد کنیم. در این قسمت از مجموعه‌های ارزیابی استفاده می‌کنیم. در هنگام آموزش می‌توانیم از دقت[۳] یا سنجه‌های دیگر روی مجموعه تست استفاده کنیم و براساس آن تصمیم بگیریم که کجا متوقف شویم. اگر از SGD استفاده کنیم، معمولاً برای هر minibatch یا مجموعه‌ای از minibatch ها، مجموعه ارزیابی را از شبکه عبور می‌دهیم و دقت را محاسبه می‌کنیم. با دنبال کردن دقت روی مجموعه ارزیابی، می‌توانیم تصمیم بگیریم که چه زمانی آموزش را متوقف کنیم. اگر برای مدت زیادی آموزش را ادامه دهیم، به‌تدریج دو اتفاق خواهد افتاد. اولین اتفاق این است که خطا روی مجموعه آموزش به سمت صفر حرکت می‌کند و ما روی

---

[۱] saddle point (مترجم: توضیح دادن نقطه زینی در نوشته یک مقدار تصور فضایی نیاز دارد اما با سرچ کردن همین کلید واژه در گوگل و رفتن به تصاویر می توانید منظور از نقطه زینی را به راحتی متوجه شوید)

[۲] Validation set

[۳] Accuracy

مجموعه آموزش بهتر و بهتر می‌شویم. دومین اتفاق این است که خطا روی مجموعه ارزیابی به سمت صفر می‌رود و پس از مدتی دوباره زیاد می‌شود.

این اثرات به‌دلیل بیش‌برازش اتفاق می‌افتند. خطای آموزش کمتر و کمتر می‌شود زیرا مدل بیشتر و بیشتر توزیع والدی که مجموعه داده را تولید کرده، یاد گرفته است. اما بعد از مدتی، مدل چیزهای قابل تعمیم درمورد مجموعه آموزش را دیگر یاد نمی‌گیرد. در این نقطه ما در حال بیش‌برازش هستیم و علاقه‌مندیم که آموزش را متوقف کنیم زیرا مدل دیگر ویژگی‌های قابل تعمیم را یاد نمی‌گیرد. درعوض، مدل جزئیاتی درمورد همان مجموعه آموزش خاصی که درحال استفاده از آن هستیم را یاد می‌گیرد. ما می‌توانیم این موضوع را با استفاده از مجموعه ارزیابی در هنگام آموزش مشاهده کنیم. از آن‌جایی که از نمونه‌های موجود در مجموعه ارزیابی برای به‌روزرسانی وزن‌ها و بایاس‌ها استفاده نمی‌کنیم، یک تست منصفانه از وضعیت جاری شبکه به ما می‌دهد. زمانی که بیش‌برازش شروع می‌شود، خطا در مجموعه ارزیابی، پس از عبور از حداقل مقدارش، شروع به افزایش می‌کند. کاری که می‌توانیم بکنیم این است که وزن‌ها و بایاس‌های بدست آمده در حداقل مقدار خطا روی مجموعه ارزیابی را نگه داریم و ادعا کنیم که این بهترین مدلی است که داریم.

ما نمی‌خواهیم هیچ داده‌ای که در آموزش استفاده شده‌است را برای اندازه‌گیری میزان خوب بودن شبکه استفاده کنیم. ما از مجموعه ارزیابی استفاده می‌کنیم تا تصمیم بگیریم که چه زمانی آموزش را متوقف کنیم. بنابراین مشخصات نمونه‌های موجود در مجموعه ارزیابی روی مدل نهایی تأثیر دارد. این بدین معناست که ما نمی‌توانیم کاملاً مطمئن باشیم که مجموعه ارزیابی نشان‌دهنده چگونگی عملکرد مدل روی داده‌های جدید است. برای پیدا کردن شهودی از چگونگی عملکرد مدل در مواجهه با داده‌های جدید تنها به مجموعه تست رجوع می‌کنیم زیرا تا زمانی که آموزش به اتمام نرسیده باشد مورد استفاده قرار نگرفته‌اند. بنابراین همانطور که استفاده از دقت در داده‌های تست برای نشان دادن میزان خوب بودن مدل مردود است، استفاده از دقت در داده‌های ارزیابی نیز برای این کار مردود است.

به‌روزرسانی نرخ آموزش

در فرمول عمومی معرفی شده برای به‌روزرسانی وزن‌ها و بایاس‌ها براساس گرادیان، پارامتر η (اتا) نرخ یادگیری یا اندازه گام، معرفی شد. این پارامتر یک عامل مقیاس است که نشان‌دهنده این است که به چه اندازه باید وزن‌ها و بایاس‌ها را براساس مقدار گرادیان به‌روزرسانی کنیم.

قبلاً ذکر شد که نیازی نیست که نرخ یادگیری ثابت باشد، بلکه می‌تواند (و حتی باید) کوچکتر و کوچکتر شود. این حرف با فرض این است که ما برای بدست آوردن مقدار دقیق حداقل تابع زیان به

گام‌های کوچکتر و کوچکتری نیاز داریم. ما بیان نکردیم که چطور باید نرخ یادگیری را به‌روزرسانی کنیم.

بیش از یک راه برای به‌روزرسانی اندازه گام وجود دارد؛ اما بعضی از آن‌ها مفیدتر از بقیه هستند. کلاس MLPClassifier از sklearn از SGD استفاده می‌کند و سه گزینه دارد. اولی این است که هیچ وقت نرخ یادگیری را تغییر نمی‌دهد و مقدار η را همان مقدار اولیه η_0 درنظر می‌گیرد. دومی این است که η را با اپوک‌ها (minibatchها) براساس فرمول زیر کاهش می‌دهد.

$$\eta = \frac{\eta_0}{t^p}$$

که در آن η_0 توسط کاربر تعیین می‌شود، t شماره تکرار (اپوک یا minibatch) است و p توان t است و توسط کاربر تعیین می‌شود. مقدار پیش‌فرض sklearn برای p عدد 0.5 است. بدین معنا که مقیاس توسط $\sqrt{t}$ انجام می‌شود و به‌نظر منطقی می‌آید.

گزینه سوم، به‌روزرسانی نرخ آموزش براساس مشاهده مقادیر تابع زیان است. تا زمانی که زیان درحال کاهش باشد، نرخ آموزش را همان چیزی که است قرار می‌دهد. وقتی که زیان دیگر کاهشی نداشته باشد، نرخ آموزش را بر یک عدد مانند ۵ تقسیم می‌کند که مقدار پیش‌فرض sklearn است. اگر مقدار نرخ آموزش را هیچ وقت تغییر ندهیم و مقدار آن زیاد باشد، ممکن است هیچ وقت مقدار حداقل زیان را نبینیم زیرا همواره در حال حرکت در اطراف آن خواهیم بود و همواره از روی آن می‌پریم. بنابراین کاهش نرخ یادگیری در هنگام استفاده از SGD ایده خوبی است. در ادامه‌ی این کتاب، روش‌های دیگر بهینه‌سازی برای تنظیم خودکار نرخ یادگیری را خواهیم دید.

تکانه

تکانه آخرین موردی است که درمورد SGD باید بررسی کنیم. همانطور که قبلاً دیدیم، معادله به‌روزرسانی وزن‌ها و بایاس‌ها هم برای گرادیان کاهشی و هم برای SGD به‌صورت زیر است.

$$w \leftarrow w - \eta \Delta w$$

ما وزن‌ها را با ضرب کردن نرخ آموزش (η) در گرادیان (Δw) به‌روزرسانی می‌کنیم.

یک تکنیک قدرتمند این است که یک جمله برای تکانه اضافه کنیم به‌طوری که کسری از Δw قبلی (یا همان مقدار به‌روزرسانی در minibatch قبلی) را به معادله اضافه کند. جمله تکانه از تغییرات خیلی سریع w در مواجهه با یک minibatch خاص، جلوگیری می‌کند. با این کار به معادله زیر خواهیم رسید.

$$w_{i+1} \leftarrow w_i - \eta \Delta w_i + \mu \Delta w_{i-1}$$

ما از اندیس استفاده کردیم تا مرحله بعدی به‌روزرسانی $(i+1)$، مرحله کنونی (i) و مرحله قبلی

$(i-1)$ را نشان دهیم. در اینجا به مقدار قبلی Δw نیاز داریم. یک مقدار معمولی برای μ (میو) حدود 0.9 است. در حقیقت همه ی جعبه‌ابزارها از جمله sklearn از تکانه استفاده می‌کنند.

پس‌انتشار

تا اینجا با این فرض کار می‌کردیم که مقدار گرادیان برای هر پارامتر را می‌دانیم. بیایید درمورد چگونگی بدست آوردن این اعداد جادویی توسط پس‌انتشار بحث کنیم. الگوریتم پس‌انتشار شاید مهمترین توسعه در تاریخ شبکه‌های عصبی است. این الگوریتم این امکان را می‌دهد که شبکه‌هایی با صدها، هزاران، میلیون‌ها و حتی میلیاردها پارامتر آموزش ببینند. این مورد مخصوصاً درمورد شبکه‌های پیچشی که در فصل ۱۲ خواهیم دید صدق می‌کند.

الگوریتم پس‌انتشار توسط Rumelhart، Hinton و Williams در مقاله پانوشت[۱] سال ۱۹۸٦ معرفی شد. این روش استفاده با احتیاط از قانون زنجیره‌ای در مشتقات است. الگوریتم پس‌انتشار نام دارد زیرا از خروجی شروع می‌کند و به سمت عقب حرکت می‌کند تا به لایه ورودی برسد. در حین حرکت، خطای تابع زیان را روی هر پارامتر شبکه انتشار می‌دهد. در زبان محاوره به این الگوریتم backprop گفته می‌شود. ما از همین عبارت استفاده می‌کنیم تا بیشتر شبیه متخصصان یادگیری ماشین باشیم.

اضافه کردن backprop به گرادیان کاهشی و سپس تغییر آن به SGD الگوریتم موجود در لیست ۲_۹ را نتیجه می‌دهد.

۱. مقادیر هوشمندانه اولیه‌ای برای وزن‌ها و بایاس‌ها انتخاب کنید.
۲. minibatch را روی شبکه براساس وزن‌ها و بایاس‌های جاری اجرا کنید و مقدار متوسط زیان را محاسبه نمایید.
۳. از این مقدار زیان و backprop برای بدست آوردن گرادیان برای هر وزن و بایاس استفاده کنید.
٤. مقدار وزن یا بایاس را با ضرب کردن اندازه گام در مقدار گرادیان، به‌روزرسانی کنید.
۵. از مرحله ۲ تکرار کنید تا مقدار زیان به اندازه کافی کم باشد.

لیست ۲_۹: گرادیان کاهشی تصادفی با backprop

[۱] Learning Representations by Back-propagating Errors.

مرحه دوم لیست ۲_۹ مسیر رو به جلو (forward pass) و مرحله سوم آن مسیر رو به عقب (backward pass) است. مسیر رو به جلو نحوه استفاده ما پس از این است که شبکه به‌طور کامل آموزش دیده است. مسیر رو به عقب backprop است. در این مسیر گرادیان‌ها محاسبه می‌شوند تا بتوانیم پارامترها را در مرحله ٤ به‌روزرسانی کنیم.

ما backprop را دوبار توصیف می‌کنیم. یک بار با یک مثال ساده کار می‌کنیم و از مشتقات واقعی استفاده می‌کنیم. بار دوم از علائم اختصاری استفاده می‌کنیم تا ببینیم به‌طور کلی backprop چگونه روی شبکه‌های عصبی واقعی اعمال می‌شود. هیچ راهی برای ساده کردن این بخش وجود ندارد. این بخش شامل مشتقات است، اما از مباحث قبلی درمورد گرادیان کاهشی، یک شهود خوب از این موضوع بدست آورده‌ایم.

Backprop، برداشت ۱

فرض کنید دو تابع $z = f(y)$ و $y = g(x)$ را داریم که نتیجه می‌دهد $z = f(g(x))$. می‌دانیم که مشتق تابع g به ما dy/dx را می‌دهد. این مقدار به ما چگونگی تغییر y در هنگام تغییر x را می‌گوید. به همین ترتیب می‌دانیم که مشتق تابع f به ما dz/dy را می‌دهد. مقدار z به ترکیب f و g بستگی دارد، بدین معنا که خروجی g ورودی f است. بنابراین اگر چگونگی تغییر z درهنگام تغییر x را بخواهیم درواقع به‌دنبال dz/dx هستیم و به یک راه برای ارتباط توابع ترکیب شده نیاز داریم. این ارتباط را قانون زنجیره‌ای در مشتق به ما می‌دهد:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

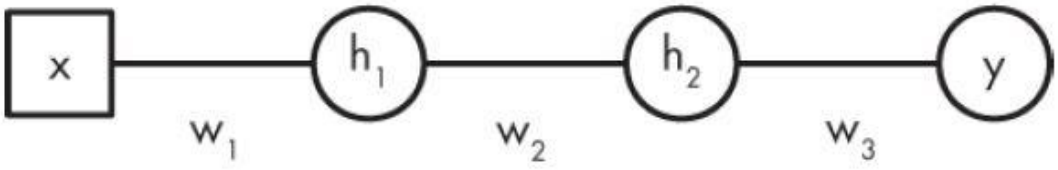
این نحوه نمایش خوب است. زیرا درصورت واقعی بودن کسرها، می‌توانیم dy را به‌صورت عبارتی ببینیم که از صورت و مخرج خط می‌خورد.

این چگونه به ما کمک می‌کند؟ در یک شبکه عصبی خروجی یک لایه، ورودی لایه بعدی است. این همان ترکیب است. بنابراین به‌طور شهودی می‌توانیم ببینیم که قانون زنجیره‌ای می‌تواند اجرا شود. به‌یاد بیاورید که ما مقادیری را می‌خواهیم که به ما چگونگی تغییر تابع زیان نسبت به تغییر وزن‌ها و بایاس‌ها را بگوید. بیایید تابع زیان را L و هر وزن یا بایاس داده شده‌ای را w فرض کنیم. ما می‌خواهیم $\frac{\partial \mathcal{L}}{\partial w}$ را برای همه وزن‌ها و بایاس‌ها محاسبه کنیم.

پاراگراف قبلی یک نماد جدید معرفی کرد. تاکنون مشتقات را به‌صورت dy/dx نمایش می‌دادیم. اما مشتق مربوط به تابع زیان به‌صورت $\frac{\partial \mathcal{L}}{\partial w}$ نشان داده شد. ∂ چیست؟

وقتی تابعی داشتیم که تنها یک متغیر داشت، در هر نقطه‌ای تنها یک شیب وجود داشت. به محض این که تابعی با بیش از یک متغیر داشته باشیم، ایده شیب در یک نقطه کمی با ابهام همراه می‌شود. در هر نقطه‌ای بی نهایت خط مماس بر تابع وجود دارد. پس ما به ایده مشتق جزئی[1] نیاز داریم. مشتق جزئی شیب خط در راستای متغیری است که درنظر داریم، درحالی که تمامی متغیرهای دیگر را ثابت درنظر گرفته‌ایم. این به ما چگونگی تغییر خروجی درهنگام تغییر یک متغیر را می‌گوید. برای نشان دادن این که در حال گرفتن مشتق جزئی هستیم، به جای d از ∂ استفاده می‌کنیم.

بیایید یک شبکه مستقیم تنظیم کنیم تا بتوانیم ببینیم که چگونه قانون زنجیره‌ای ما را به عبارتی که می‌خواهیم می‌رساند. ما به شبکه موجود در شکل ۳_۹ نگاه می‌کنیم. این شبکه شامل یک ورودی، دو لایه مخفی و یک لایه خروجی، هر کدام با یک گره است.



شکل ۳_۹: یک شبکه ساده برای نشان دادن قانون زنجیره‌ای

برای سادگی از مقادیر بایاس صرف‌نظر می‌کنیم. همچنین فرض کنید تابع فعال‌سازی تابع همانی $h(x) = x$ است. این ساده‌سازی مشتق تابع فعال‌سازی را حذف می‌کند تا همه چیز شفاف‌تر شود.

در این شبکه، مسیر رو به جلو (forward pass) موارد زیر را محاسبه می‌کند.

$$h_1 = w_1 x$$
$$h_2 = w_2 h_1$$
$$y = w_3 h_2$$

این همان شمایلی است که قبلاً استفاده کرده بودیم. در اینجا با قرار دادن خروجی یک لایه به‌عنوان ورودی لایه بعدی، آن‌ها به هم زنجیر شده‌اند. این محاسبات برای ورودی x به ما خروجی شبکه y را می‌دهد. برای آموزش شبکه به یک مجموعه آموزش نیاز داریم. یعنی یک مجموعه از زوج‌های $(x_i, \hat{y}), i = 0,1, \ldots$ که مثال‌هایی از خروجی‌هایی است که به‌ازای ورودی باید داشته باشیم. توجه کنید که مسیر رو به جلو از ورودی x به خروجی y حرکت کرد. در ادامه خواهیم دید که چرا مسیر رو به عقب (*backward pass*) از خروجی به سمت ورودی حرکت می‌کند.

اگر تابع زیان $\mathcal{L}$ را به‌صورت مربع خطا بین y و $\hat{y}$ تعریف کنیم به شکل زیر خواهد بود

[1] partial derivative

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

y همان خروجی شبکه به‌ازای یک ورودی داده شده x است و $\hat{y}$ خروجی‌ای است که باید دریافت کنیم (مترجم: مقدار درست خروجی به‌ازای ورودی x).

زیان درواقع یک میانگین روی مجموعه آموزش یا تعدادی minibatch است. برای سادگی این حقیقت را کنار می‌گذاریم. ضریب ½ ضروری نیست اما برای این که مشتق مقداری بهتر عمل کند، استفاده می‌شود. از آنجایی که ما به‌دنبال حداقل مقدار تابع زیان برای یک مجموعه از وزن‌ها هستیم، اهمیتی ندارد که همیشه مقدار تابع را در یک ضریب ثابت ½ ضرب کرده‌ایم (مقدار حداقل همواره مقدار حداقل باقی خواهد ماند مستقل از اینکه مقدار عددی آن چند باشد).

برای استفاده از گرادیان کاهشی، به چگونگی تغییر زیان با تغییر وزن‌ها نیاز داریم. در شبکه‌ای که مثال زدیم ما به سه مقدار گرادیان برای w_1، w_2 و w_3 نیاز داریم. در اینجا قانون زنجیره‌ای وارد معادلات می‌شود. در ابتدا معادلات را می‌نویسیم و سپس درمورد آن‌ها صحبت می‌کنیم:

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

ترتیب این معادلات نشان می‌دهد که چرا اسم این الگوریتم "پس‌انتشار" است. برای بدست آوردن مشتق جزئی برای پارامتر لایه خروجی، تنها به مقدار خروجی y و مقدار زیان $\mathcal{L}$ نیاز داریم. برای بدست آوردن مشتق جزئی برای وزن لایه میانی به این دو مشتق جزئی از لایه خروجی نیاز داریم:

$$\frac{\partial \mathcal{L}}{\partial y}$$
$$\frac{\partial y}{\partial h_2}$$

درنهایت برای بدست آوردن مشتق جزئی برای وزن لایه ورودی، ما به مشتقات جزئی از لایه خروجی و لایه میانی نیاز داریم. در عمل، ما از طریق شبکه به سمت عقب حرکت کردیم و مقادیر را از لایه‌های بعدی انتشار دادیم.

اگر هرکدام از این معادلات را به‌صورت کسرهای واقعی درنظر بگیریم، عبارت‌های صورت و مخرج ساده می‌شوند و سمت راست و چپ معادله با هم برابر خواهد شد. از آن‌جایی که یک شمایل ساده برای

شبکه درنظر گرفتیم می‌توانیم گرادیان‌ها را با دست حساب کنیم. ما به گرادیان‌های زیر که در سمت راست معادلات بالا هستند، نیاز داریم:

$$\frac{\partial \mathcal{L}}{\partial y} = (y - \hat{y})$$
$$\frac{\partial y}{\partial w_3} = h_2 = w_2 h_1 = w_2 w_1 x$$
$$\frac{\partial y}{\partial h_2} = w_3$$
$$\frac{\partial h_2}{\partial w_2} = h_1 = w_1 x$$
$$\frac{\partial h_2}{\partial h_1} = w_2$$
$$\frac{\partial h_1}{\partial w_1} = x$$

$\frac{\partial \mathcal{L}}{\partial y}$ براساس تابعی که برای زیان درنظر گرفتیم و قوانین مشتق در ریاضیات، بدست آمد.
قرار دادن این مقادیر در معادلات مربوط به گرادیان‌های وزن‌ها معادلات زیر را می‌دهد.

$$\frac{\partial \mathcal{L}}{\partial w_3} = (y - \hat{y}) w_2 w_1 x$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = (y - \hat{y}) w_3 w_1 x$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = (y - \hat{y}) w_3 w_2 x$$

بعد از مسیر رو به جلو (forward pass)، مقادیر عددی سمت راست این معادلات را خواهیم داشت. درنتیجه مقادیر عددی گرادیان‌ها را خواهیم داشت. قانون به‌روزرسانی از گرادیان کاهشی به ما پیشنهاد می‌دهد که وزن‌ها را به صورت زیر تغییر دهیم:

$$w_3 \leftarrow w_3 - \eta \frac{\partial \mathcal{L}}{\partial w_3} = w_3 - \eta (y - \hat{y}) w_2 w_1 x$$
$$w_2 \leftarrow w_2 - \eta \frac{\partial \mathcal{L}}{\partial w_2} = w_2 - \eta (y - \hat{y}) w_3 w_1 x$$
$$w_1 \leftarrow w_1 - \eta \frac{\partial \mathcal{L}}{\partial w_1} = w_1 - \eta (y - \hat{y}) w_3 w_2 x$$

که در آن η نرخ یادگیری است و نشان می‌دهد که هنگام به‌روزرسانی، گام چقدر بزرگ باشد.

به‌طور خلاصه، ما از قاعده زنجیره‌ای به‌عنوان قلب الگوریتم backprop استفاده می‌کنیم تا گرادیان‌هایی که برای به‌روزرسانی وزن‌ها در هنگام آموزش نیاز داریم را، بدست بیاوریم. برای شبکه ساده ما، توانستیم مقادیر این گرادیان‌ها را به‌وضوح با طی کردن شبکه به سمت عقب، از خروجی به ورودی، بدست بیاوریم. البته که این یک شبکه بسیار ساده بود. بیایید یک نگاه دیگری به چگونگی استفاده از backprop بیاندازیم تا یک شهود از محاسبه گرادیان‌ها برای هر شبکه‌ای بدست بیاوریم.

Backprop، برداشت ۲

با بازدید تابع زیان و معرفی تعدادی علائم جدید شروع می‌کنیم. تابع زیان، تابعی از تمامی پارامترهای شبکه است. بدین معنی که همه وزن‌ها و بایاس‌ها در آن نقش بازی می‌کنند. به‌عنوان مثال تابع زیان شبکه موجود در شکل ۱_۸ که ۲۰ عدد وزن و بایاس دارد به‌صورت زیر نوشته می‌شود.

$$loss = \mathcal{L}\begin{pmatrix} w_{00}^{(1)}, w_{01}^{(1)}, w_{02}^{(1)}, w_{10}^{(1)}, w_{11}^{(1)}, w_{12}^{(1)}, b_0^{(1)}, b_1^{(1)}, b_2^{(1)}, \\ w_{00}^{(2)}, w_{01}^{(2)}, w_{10}^{(2)}, w_{11}^{(2)}, w_{20}^{(2)}, w_{21}^{(2)}, b_0^{(1)}, b_1^{(1)}, \\ w_{00}^{(3)}, w_{10}^{(3)}, b_0^{(3)} \end{pmatrix}$$

در اینجا یک علامت جدید برای پارامترها به‌صورت زیر معرفی کردیم:

$$w_{jk}^{(i)}$$

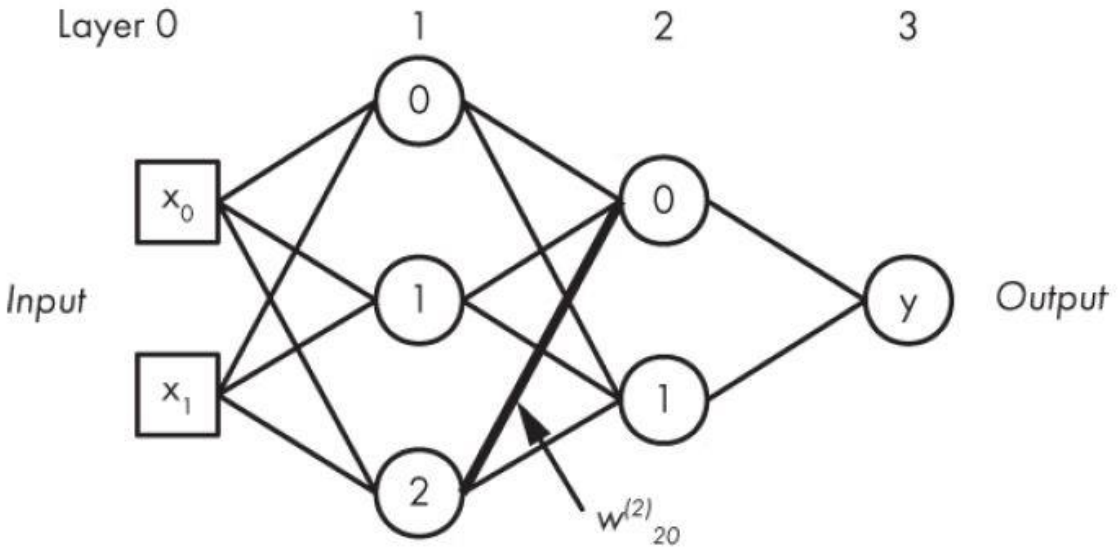
این نشان‌دهنده وزنی است که خروجی j ام لایه i-1 ام را به گره k ام از لایه i ام وصل می‌کند. همچنین پارامتر زیر را داریم:

$$b_k^{(i)}$$

که نشان‌دهنده مقدار بایاس برای k امین گره از لایه i ام است. در اینجا لایه صفر، لایه ورودی است. اعدادی که در پرانتز موجود در توان نوشته شده‌اند نشان‌دهنده شماره لایه هستند و نباید به‌عنوان توان واقعی تفسیر شوند. بنابراین

$$w_{20}^{(2)}$$

نشان‌دهنده وزنی است که سومین خروجی لایه اولی را به اولین گره در لایه دوم متصل می‌کند. این وزن در شکل ٤_۹ نشان داده شده‌است. به‌یاد بیاورید که ما شماره گره‌ها را از بالا به پایین می‌نوشتیم و از صفر شروع می‌کردیم.



شکل ٤_٩: شبکه موجود در شکل ١_٨ درحالی که w^2_{20} با خط تیره مشخص شده‌است.

این نحوه نمایش مقداری دلهره آور است اما این اجازه را به ما می‌دهد که دقیقاً به هر وزن یا بایاسی در شبکه اشاره کنیم. اعدادی که برای استفاده از backprop نیاز داریم، مشتقات جزئی زیان نسبت به هر وزن یا بایاس است. بنابراین چیزی که درنهایت می‌خواهیم پیدا کنیم، به‌صورت زیر نوشته می‌شود:

$$\frac{\partial \mathcal{L}}{\partial w^{(i)}_{jk}}$$

این به ما شیب را می‌دهد. مقداری که تابع زیان تغییر می‌کند نسبت به تغییر در وزنی که k امین گره از لایه i ام را به j امین خروجی از لایه i-1 ام وصل کرده است. یک معادله مشابه به ما مشتقات جزئی بایاس‌ها را می‌دهد.

ما می‌توانیم تنها با کار کردن با شماره لایه، این نمایش سخت را ساده کنیم. با دانستن این که چیزی که در علائم زیر وجود دارد بردار (بایاس‌ها یا مقادیر فعال‌سازی) یا ماتریس (وزن‌ها) است، می‌توانیم فرمول را به این صورت بنویسیم:

$$\frac{\partial \mathcal{L}}{\partial w^{(i)}} \; and \; \frac{\partial \mathcal{L}}{\partial b^{(i)}}$$

$w^{(i)}$ نشان‌دهنده یک ماتریس برای تمامی وزن‌هایی است که لایه *i-1* ام را به لایه i ام متصل می‌کند و $b^{(i)}$ یک بردار برای تمامی مقادیر بایاس لایه i ام است.

با دیدن عبارت‌ها به‌صورت بردارها و ماتریس‌ها، از همین نحوه نمایش استفاده خواهیم کرد. بیایید به

لایه خروجی یا همان لایه L نگاه کنیم. ما می‌دانیم که مقدار فعال‌سازی لایه خروجی L از طریق زیر بدست می‌آید.

$$a^{(L)} = h(W^{(L)}a^{(L-1)} + b^{(L)})$$

که در آن a مقادیر فعال‌سازی از لایه *L-1*، b بردار بایاس برای لایه L و W ماتریس وزن‌ها بین لایه *L-1* و L است.h تابع فعال‌سازی است.

علاوه بر این، ما ورودی h را $z^{(L)}$ می‌نامیم که به‌صورت زیر تعریف می‌شود.

$$z^{(L)} = W^{(L)}a^{(L-1)} + b^{(L)}$$

به $\frac{\partial \mathcal{L}}{\partial z^{(l)}}$ خطا می‌گوییم که درواقع سهم زیان از ورودی تا لایه l ام است. حال عبارت زیر را تعریف می‌کنیم.

$$\delta^{(l)} \equiv \frac{\partial \mathcal{L}}{\partial z^{(l)}}$$

حال از این به بعد می‌توانیم با δ (دلتا) کار کنیم.

برای لایه خروجی می‌توانیم δ را به‌صورت زیر بنویسیم.

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial z^{(L)}} = \frac{\partial \mathcal{L}}{\partial a^{(L)}}.h'\left(z^{(L)}\right)$$

عبارت $h'\left(z^{(L)}\right)$ روش دیگری برای نشان دادن مشتق h نسبت به z در نقطه $z^{(L)}$ است. "." نشان‌دهنده ضرب درایه به درایه[1] است. این روشی است که NumPy هنگام ضرب کردن دو آرایه با اندازه یکسان انجام می‌دهد. اگر C=A.B آنگاه $C_{ij} = A_{ij}.B_{ij}$. از لحاض فنی این نوع ضرب، ضرب Hadamard نام دارد که به افتخار یک ریاضی دان فرانسوی به اسم Jacques Hadamard نام‌گذاری شده‌است.

صحبت‌هایی که شد این نتیجه را می‌دهد که برای استفاده از پس‌انتشار، به تابع زیانی نیاز داریم که مشتق‌پذیر باشد (یک تابع زیان که در تمامی نقاط آن مشتق وجود داشته باشد). برآوردن این خواسته زیاد سخت نیست. توابع زیانی که در بخش بعد امتحان می‌کنیم، این معیار را دارند. همچنین برای پیدا کردن h(z) به یک تابع فعال‌سازی نیاز داریم که مشتق‌پذیر باشد. دوباره تمامی توابع فعال‌سازی که تاکنون مورد بررسی قرار داده‌ایم، اساساً مشتق‌پذیر هستند.

توجه: من از کلمه "اساساً" استفاده کردم زیرا مشتق ReLU در x=0 تعریف نشده‌است. مشتق چپ برابر صفر و مشتق راست برابر ۱ است. در عمل، پیاده‌سازی ReLU به‌گونه‌ای اتفاق می‌افتد که مشتق ReLU در این نقطه را دقیقاً برابر صفر درنظر بگیرد. به‌عنوان مثال TensorFlow نگاه می‌کند که آیا مقدار

[1] Elementwise (مترجم: در این نوع ضرب، هر درایه از یک ماتریس در درایه متناظرش در ماتریس بعدی ضرب می‌شود.)

x کمتر یا مساوی صفر است یا خیر؛ اگر بود به‌سادگی مقدار مشتق را صفر برمی‌گرداند و در غیر این صورت مقدار یک را برمی‌گرداند. این کار درست است زیرا در هنگام محاسبات تعداد زیادی گرد کردن مقادیر اعشاری اتفاق می‌افتد و خیلی غیر محتمل است که نقطه‌ای که می‌خواهیم در آن مشتق را حساب کنیم دقیقاً برابر صفر مطلق باشد.

معادله δ نشان‌دهنده خطا از لایه ورودی تا یک لایه مشخص است. در آینده خواهیم دید که چگونه از این استفاده کنیم تا خطا را از هر وزن در یک لایه بدست بیاوریم.

با استفاده از فرمول زیر و داشتن $\delta^{(L)}$ می‌توانیم خطا را به لایه قبلی انتشار دهیم[1].

$$\delta^{(l)} = \left((W^{(l+1)})^T \delta^{l+1}\right).h'(z^{(l)})$$

که در آن برای لایه یکی مانده به آخر $l+1=L$ خواهد شد . T نشان‌دهنده ترآنهاده ماتریس است. ترآنهاده یک عملیات استاندارد روی ماتریس‌هاست که ماتریس را حول قطر آن قرینه می‌کند. بنابراین اگر ماتریس A به شکل زیر باشد:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

آنگاه ترآنهاده آن به‌صورت زیر است.

$$A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

ما ترآنهاده ماتریس وزن را نیاز داریم زیر در جهت مخالف مسیر رو به جلو حرکت می‌کنیم. اگر ۳ گره در لایه l و دو گره در لایه $l+1$ وجود داشته باشد، آنگاه ماتریس وزن بین آن‌ها یا همان W، یک ماتریس 2×3 خواهد بود. بنابراین Wx یک بردار با دو عنصر خواهد بود. در backprop از لایه $l+1$ به لایه l می‌رویم؛ به همین دلیل از ترآنهاده ماتریس وزن استفاده می‌کنیم تا بردار دو عنصری δ را به یک بردار سه عنصری در لایه l، نگاشت دهد.

معادله $\delta^{(l)}$ برای تمامی لایه‌ها استفاده می‌شود و براساس آن شبکه را رو به عقب طی می‌کنیم. شروع فرآیند با مقادیر لایه خروجی است که با $\delta^{(L)}$ نشان داده می‌شود.

---

[1] مترجم: فکر کنم این فرمول باید اینطوری باشه:

$$\delta^{(l)} = \left((W^{(l+1)})^T \delta^{l+1}\right).h'(z^{(l)})$$

میتونید خودتون اخر این سایت رو چک کنید: https://cnl.salk.edu/~schraudo/teach/NNcourse/backprop.html

یا حتی این سایت: https://stats.stackexchange.com/questions/294873/what-is-the-significance-of-the-delta-matrix-in-neural-network-backpropagation

یا سایتایی که خودتون می دونید ولی تو کتاب فرمولی که تو متن هست رو نوشته. تصمیم گیری با خودتونه.

پس از این که مقادیر خطا در هر لایه را داشته باشیم، می‌توانیم مقادیر گرادیان مورد نیاز را بدست بیاوریم. برای بایاس‌ها در یک لایه، مقادیرگرادیان همان عناصر δ است.

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

که درواقع j امین عنصر بایاس برای لایه l ام است. برای وزن‌ها به عبارت زیر نیاز داریم.

$$\frac{\partial \mathcal{L}}{\partial w_{kj}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}$$

که خروجی k ام لایه قبلی را به j امین خطای لایه کنونی (لایه l ام) مرتبط می‌کند.

با استفاده از این دو فرمول، مقادیر گرادیان وزن‌ها و بایاس‌ها بدست می‌آید و می‌توان از آن‌ها برای اجرای گرادیان کاهشی استفاده کرد.

امیدوارم شما از این بخش به این دیدگاه رسیده باشید که ما می‌توانیم از یک تعریف ریاضیاتی سر راست درمورد خطا استفاده کنیم تا یک فرآیند تکرار شونده را ایجاد کنیم که خطا را از لایه خروجی شبکه به سمت لایه ورودی حرکت می‌دهد. ما نمی‌توانیم خطا در یک لایه را بدون در دست داشتن خطای لایه‌های بعد از آن محاسبه کنیم. بنابراین کاری که می‌کنیم این است که خطا را رو به عقب انتشار می‌دهیم و به همین دلیل اسم این کار پس‌انتشار است.

توابع زیان

تابع زیان در حین آموزش استفاده می‌شود تا نشان دهد که شبکه چقدر بد عمل می‌کند. هدف آموزش این است که همزمان با تعمیم خصیصه‌های واقعی داده‌ها، این مقدار را تا جای ممکن کوچک کند. در تئوری اگر احساس کنیم هر تابع زیانی برای مسئله مناسب است، می‌توانیم از آن استفاده کنیم و امکان تعریف هر نوع تابع زیانی را داریم. اگر ادبیات مربوط به یادگیری عمیق را مطالعه کنید می‌بینید که در مقالات این کار همواره انجام می‌شود. با این حال، در اکثر تحقیقات تنها تعداد محدودی از توابع زیان مورد توجه قرار گرفته‌اند. این‌ها توابع استانداردی هستند که به‌طور تجربی در اکثر مواقع عملکرد خوبی داشته‌اند. ما در اینجا سه مورد از این‌ها را بررسی می‌کنیم: زیان قدر مطلق[1] (گاهی وقت‌ها زیان L_1 نامیده

[1] absolute loss

می‌شود)، میانگین مربعات خطا[1] (گاهی وقت‌ها زیان L_2 نامیده می‌شود) و زیان آنتروپی متقابل[2].

زیان قدر مطلق خطا و میانگین مربعات خطا

بیایید با توابع زیان قدر مطلق خطا و میانگین مربعات خطا شروع کنیم. ما درمورد آن‌ها در یک جا بحث می‌کنیم زیرا از لحاض ریاضی بسیار شبیه هم هستند.

قبلاً در مبحث backprop، میانگین مربعات خطا را دیدیم. قدر مطلق خطا جدید است. از لحاض ریاضی معادلات مربوط به این دو، به‌صورت زیر است.

$$\mathcal{L}_{abs} = |y - \hat{y}|$$
$$\mathcal{L}_{MSE} = \frac{1}{2}(y - \hat{y})^2$$

اندیس abs را برای قدر مطلق و MSE را برای میانگین مربعات خطا قرار دادیم. توجه کنید که همیشه خروجی شبکه در طی مسیر رو به جلو به‌ازای یک ورودی x را با y نشان می‌دهیم. همچنین همیشه برای برچسب صحیح کلاس از $\hat{y}$ استفاده می‌کنیم. این برچسب یک عدد صحیح است که از صفر شروع می‌شود.

اگرچه ما تابع زیان را به شکل ساده ای نوشتیم، باید به‌یاد بیاوریم که در عمل، مقدار زیان، میانگین تابع زیان روی تمامی مجموعه آموزش یا minibatch است. اصل کلمه میانگین در عبارت "میانگین مربعات خطا" از همین موضوع نشأت گرفته است. بنابراین در اصل باید فرمول زیر را بنویسیم.

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y_i - \hat{y}_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

در اینجا ما میانگین N مقدار از مربعات خطا را روی مجموعه آموزش (یا minibatch) بدست می‌آوریم.

اگر معنای این دو تابع را درنظر بگیریم، هر دو مورد منطقی به‌نظر می‌آیند. ما می‌خواهیم که شبکه مقداری را به‌عنوان خروجی بدهد که با مقدار واقعی آن (برچسب واقعی نمونه) مطابقت داشته باشد. اختلاف بین این دو، نشانه‌ای از میزان خطای خروجی شبکه است. در زیان قدرمطلق، این اختلاف را پیدا می‌کنیم و علامت آن را درنظر نمی‌گیریم. درنظر نگرفتن علامت کاری است که قدر مطلق انجام می‌دهد. در زیان MSE، اختلاف را پیدا می‌کنیم و سپس آن را به توان ۲ می‌رسانیم. این کار هم مقدار اختلاف را مثبت می‌کند؛ زیرا ضرب یک مقدار منفی در خودش همواره یک مقدار مثبت است. همانطور که در بخش "پس‌انتشار" در صفحه ۲۰۰ اشاره شد، ضریب ½ در MSE، مشتق را ساده می‌کند و در نحوه عملکرد

---

[1] mean squared error

[2] cross-entropy loss

کلی آن اثری ندارد.

زیان قدر مطلق و MSE با هم متفاوت اند. MSE به داده‌های پرت بیشتر حساس است. دلیل این امر این است که ما اختلافات را به توان ۲ می‌رسانیم. نمودار $y=x^2$ وقتی که x (یا همان اختلاف) بزرگتر شود، سریع‌تر افزایش می‌یابد. در زیان قدر مطلق، این اثر کمینه شده‌است زیرا هیچ توان دویی وجود ندارد و مقدار اختلاف به همان صورت باقی می‌ماند.

درحقیقت، معمولاً هیچ کدام از این دو تابع زیان وقتی هدف شبکه‌های عصبی دسته‌بندی است، استفاده نمی‌شوند. در این کتاب نیز شبکه‌های عصبی برای دسته‌بندی مطالعه می‌شوند. در این نوع از شبکه‌های عصبی به‌طور معمول از زیان آنتروپی متقابل استفاده می‌شود که در بخش بعدی مورد بررسی قرار می‌گیرد. ما می‌خواهیم که خروجی شبکه عصبی، برچسب درست کلاس ورودی باشد. اما کاملاً امکان‌پذیر است که یک شبکه عصبی با خروجی اعداد پیوسته و حقیقی آموزش دهیم. این کار رگرسیون[۱] نام دارد. برای رگرسیون هر دو مورد از توابع زیانی که در بالا معرفی شد، مفید هستند.

زیان آنتروپی متقابل

اگرچه می‌توانیم توابع زیان قدر مطلق و MSE را برای شبکه‌های عصبی که کارشان دسته‌بندی است، استفاده کنیم؛ اما تابعی که برای این کار بیشتر استفاده می‌شود زیان آنتروپی متقابل است (ارتباط نزدیکی با log-loss دارد). این تابع فرض می‌کند که خروجی شبکه برای مواردی که چند کلاس[۲] (بیش از دو کلاس) داشته باشیم یک (بردار) softmax است و برای مواردی که دو کلاس داشته باشیم سیگموئید (لاجستیک، اسکالر) است. از لحاظ ریاضی این تابع برای وقتی که M کلاس داشته باشیم به شکل زیر است.

بیش از دو کلاس: $\mathcal{L}_{ent} = -\sum_i^M \hat{y}_i \log(y_i)$

دو کلاس: $\mathcal{L}_{ent} = -\hat{y}\log(y) + (1-\hat{y})\log(1-y)$

تابع زیان آنتروپی متقابل چه کاری انجام می‌دهد که آن را برای کاربرد دسته‌بندی در شبکه‌های عصبی مناسب‌تر کرده است؟ بیایید موردی که بیش از دو کلاس با خروجی softmax داریم را درنظر بگیریم. براساس تعریف softmax، خروجی‌های شبکه می‌توانند به‌عنوان تخمینی از احتمال تعلق ورودی به هر کدام از کلاس‌ها، تعبیر شود. اگر سه تا کلاس داشته باشیم ممکن است یک خروجی softmax به شکل زیر دریافت کنیم.

---

[۱] regression

[۲] multiclass

$$y = (0.03,0.87,0.10)$$

این خروجی تقریباً می‌گوید که شبکه فکر می‌کند که 3 درصد، 87 درصد و 10 درصد شانس وجود دارد که، به ترتیب، ورودی متعلق به کلاس 0، 1، و 2 باشد. این بردار خروجی y است. برای محاسبه تابع زیان به برچسب واقعی نیاز است. برچسب واقعی توسط یک بردار نشان داده می‌شود که در عنصری که ورودی متعلق به آن کلاس باشد، عدد 1 و در عناصری که ورودی متعلق به کلاس آن‌ها نیست عدد 0 قرار داده شده‌است. بنابراین بردار $\hat{y}$ مربوط به ورودی‌ای که y قبلی را نتیجه داده بود، به شرح زیر است.

$$\hat{y} = (0,1,0)$$

میزان زیان کلی به‌صورت زیر محاسبه می‌شود.

$$\mathcal{L}_{ent} = -\big(0(log0.03) + 1(log0.87) + 1(log0.10)\big) = 0.139262$$

سه پیش‌بینی شبکه در کنار هم می‌تواند به‌عنوان یک توزیع احتمال درنظر گرفته شود؛ دقیقاً مانند احتمالات مختلف برای حالات مختلف پرتاب دو تاس. ما همچنین توزیع احتمالی برچسب‌های کلاس‌ها را نمی‌دانیم. برای مثال گفته شده، کلاس واقعی، کلاس 1 است. بنابراین ما یک توزیع احتمالی ایجاد کرده‌ایم که به کلاس‌های 0 و 2 هیچ شانسی نمی‌دهد و به کلاس 1، 100 درصد شانس می‌دهد. وقتی شبکه آموزش ببیند، ما انتظار داریم که توزیع خروجی هرچه بیشتر به (0,1,0) نزدیک شود. (0,1,0) همان توزیع برچسب مربوطه است.

کمینه کردن آنتروپی متقابل باعث می‌شود که شبکه پیش‌بینی‌های بهتر و بهتری از توزیع احتمالی کلاس‌هایی که می‌خواهیم شبکه یاد بگیرد، داشته باشد. درحالت ایده آل این توزیع‌های خروجی مشابه برچسب‌های آموش خواهند شد. یعنی برای هر کلاسی غیر از کلاس واقعی مقدار 0 و برای کلاس واقعی مقدار 1 را به‌عنوان خروجی خواهند داد.

برای دسته‌بندی معمولاً از زیان آنتروپی متقابل استفاده می‌کنیم. کلاس MLPClassifier در sklearn از آنتروپی متقابل استفاده می‌کند. Keras نیز آنتروپی متقابل را دارد، اما تعداد زیادی از توابع زیان دیگری را نیز شامل می‌شود، از جمله زیان قدرمطلق و میانگین مربعات خطا.

مقداردهی اولیه[1] وزن‌ها

قبل از آموزش یک شبکه عصبی به مقادیر اولیه‌ای برای وزن‌ها و بایاس‌ها نیاز داریم. مرحله اول در لیست ۱_۹ برای گرادیان کاهشی می‌گوید: " مقادیر هوشمندانه اولیه‌ای برای وزن‌ها و بایاس‌ها انتخاب

[1] initialization

کنید."

تکنیک مقداردهی اولیه‌ای که در این جا مورد بحث قرار می‌گیرد به‌صورت انتخاب اعداد تصادفی در یک بازه است. علاوه بر این، اعداد تصادفی یا باید توزیع یکنواخت یا نرمال داشته باشند. توزیع یکنواخت یعنی تمامی‌اعداد موجود در آن بازه شانس یکسانی برای انتخاب شدن داشته باشند. درواقع اعداد صحیح بین 1 و 6 که براساس پرتاب یک تاس سالم بدست می‌آید توزیع یکنواخت دارند. توزیع نرمال در فصل 4 معرفی شد. این توزیع یک میانگین و انحراف استاندارد مشخص دارد. اعداد نزدیک میانگین بیشترین احتمال انتخاب شدن را دارند و هرچه از میانگین فاصله بگیریم این احتمال به صفر نزدیک می‌شود. این توزیع همان نمودار زنگوله‌ای معروف را دارد. هر دو توزیع می‌توانند انتخاب شوند. نکته اصلی این است که تمامی مقادیر اولیه وزن‌ها یک عدد یکسان (مثل صفر) نیستند. اگر یکسان باشند، تمامی گرادیان‌ها یکسان می‌شوند و تمامی وزن‌ها به یک اندازه تغییر می‌کنند. وزن‌ها اولیه باید متفاوت باشند تا این تقارن را بشکنند و این اجازه را داشته باشند که به‌طور جداگانه خوشان را با داده‌ها آموزش تطبیق دهند.

در اوایل دوران شبکه‌های عصبی، افراد مقادیر اولیه وزن‌ها و بایاس‌هایشان را با انتخاب یکنواخت مقادیر در بازه [0,1) یا همان توزیع یکنواخت U(0,1) بدست می‌آوردند. راه دوم انتخاب مقادیر از توزیع نرمال N(0,1) با میانگین 0 و انحراف استاندارد 1 بود. این مقادیر معمولاً در یک مقدار ثابت و کوچک مثل 0.01 ضرب می‌شدند. با این حال، زمانی که شبکه‌ها پیچیده‌تر شدند، این روش‌های ساده منسوخ شدند. شبکه‌هایی که با این روش مقداردهی اولیه می‌شدند، در آموزش به مشکل می‌خوردند و تعداد زیادی از آن‌ها اصلاً آموزش نمی‌دیدند.

تحقیقات زیادی در طی چند دهه روی این موضوع انجام شد. محققان متوجه شدند که دقیقاً وزن‌های هر لایه چگونه باید مقداردهی شوند. در درجه اول این مقداردهی به چند چیز بستگی داشت: نوع تابع فعال‌سازی و تعداد وزن‌هایی که به لایه وارد می‌شوند (f_{in}) و احتمالاً تعداد وزن‌هایی که از لایه خارج می‌شوند (f_{out}). این ادراک باعث شد که به روش‌های مقداردهی اولیه مهمی که امروزه استفاده می‌شوند برسیم.

کلاس MLPClassifier در sklearn از مقداردهی اولیه Glorot استفاده می‌کند که گاهاً مقداردهی اولیه Xavier نیز نامیده شده‌است. اما منظور بعضی از جعبه‌ابزارها از این دو اسم، دو چیز متفاوت است[1] (توجه کنید که Xavier و Glorot به فرد یکسانی اشاره دارند). بیایید ببینیم که sklearn چگونه از Glorot برای

[1] برای اطلاعات بیشتر به مقاله زیر مراجعه کنید:
Glorot, Xavier, and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks."

مقداردهی اولیه استفاده می‌کند. متد[1] اصلی در MLPClassifier برای مقداردهی اولیه وزن‌ها _init_coef است. این متد از توزیع یکنواخت استفاده می‌کند و دامنه آن را به‌گونه‌ای تنظیم می‌کند که وزن‌ها در بازه زیر قرار بگیرند.

$$\left[-\sqrt{\frac{A}{\mathrm{f}_{in}+\mathrm{f}_{out}}}, \sqrt{\frac{A}{\mathrm{f}_{in}+\mathrm{f}_{out}}}\right]$$

براکت نشان‌دهنده کوچکترین مقدار ممکن (سمت چپ) و بزرگترین مقدار ممکن (سمت راست) برای انتخاب است. چون توزیع یکنواخت است، تمامی مقادیر موجود دراین بازه شانس یکسانی برای انتخاب شدن دارند.

ما هنوز مشخص نکرده‌ایم که A چیست. این مقدار به تابع فعال‌سازی استفاده شده بستگی دارد. براساس ادبیات اگر تابع فعال‌سازی سیگموئید (لاجستیک) باشد، آنگاه A=2 پیشنهاد شده‌است. در غیر این صورت توصیه شده که از A=6 استفاده شود.

بعضی از جعبه‌ابزارها مانند Caffe، از فرم جایگزین مقداردهی اولیه Xavier استفاده می‌کند. در این فرم آن‌ها توزیع نرمال استاندارد را در یک عدد ضرب می‌کنند. درواقع مقادیر اولیه وزن‌ها از توزیع زیر بدست می‌آید.

فرم جایگزین مقداردهی اولیه Xavier [2]: $N(0,1)\sqrt{\frac{1}{\mathrm{f}_{in}}}$

معرفی واحد یکسوساز خطی (ReLU) باعث پیچیدگی بیشتر شد و یک تغییر دیگر را پیشنهاد داد که در مقداردهی اولیه He [3] مشاهده می‌شود. در اینجا عدد 1 موجود در مقداردهی اولیه Xavier، به عدد 2 تغییر می‌کند. این روش مقداردهی اولیه به‌صورت زیر است.

مقداردهی اولیه He، فقط برای ReLU: $N(0,1)\sqrt{\frac{2}{\mathrm{f}_{in}}}$

برای اطلاعات بیشتر در این مورد به مقاله Kaiming He و همکاران با عنوان " Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification" مراجعه نمایید. نکته اصلی در این روش‌های مقداردهی اولیه این است که پیشنهادهای قدیمی "مقادیر کوچک تصادفی" به روش‌های نظام‌مندتری تبدیل شده‌اند. در روش‌های جدید، معماری شبکه نیز با استفاده از f_{in} و f_{out} درنظر گرفته شده‌است.

در مبحثی که داشتیم از مقادیر بایاس صحبت نشد. این کار عمدی بود. اگرچه ممکن است به‌نظر برسد که

---

[1] method

[2] alternate Xavier initialization

[3] He initialization

مقداردهی اولیه مقادیر بایاس نسب به اینکه همه آن‌ها را 0 قرار دهیم ارجحیت دارد؛ اما دانش کنونی می‌گوید که بهترین مقدار اولیه آن‌ها 0 است. با این وجود MLPClassifier در sklearn مقادیر اولیه بایاس را مانند وزن‌ها تعیین می‌کند.

بیش‌برازش و تنظیم گری

هدف از آموزش مدل این است که مدل ویژگی‌های اساسی و قابل تعمیم توزیع والدی که مجموعه داده، نمونه‌ای از آن است را یاد بگیرد. بدین صورت اگر مدل با ورودی‌های جدیدی مواجه شود، آمده است که آن‌ها را به‌درستی درک کند. همانطور که در این فصل دیدیم، روش اولیه آموزش یک شبکه عصبی شامل بهینه‌سازی است. درواقع به‌دنبال بهترین مجموعه از پارامترها بودیم، به‌طوری که شبکه کمترین خطای ممکن را روی مجموعه آموزش داشته باشد.

با این حال، پیدا کردن بهترین مجموعه از پارامترهایی که خطای آموزش را کمینه می‌کند، کافی نیست. اگر در هنگام دسته‌بندی داده‌های آموزش هیچ خطایی نداشته باشیم، معمولاً علت آن بیش‌برازش و عدم یادگیری ویژگی‌های قابل تعمیم داده‌ها است. این حالت بیشتر در مدل‌های سنتی و کلاسیک شبکه‌های عصبی و کمتر در مدل‌های عمیق، مانند شبکه‌های پیچشی در فصل ۱۲ اتفاق می‌افتد.

درک بیش‌برازش

تاکنون هر از چند گاهی درمورد بیش‌برازش صحبت کرده بودیم اما هیچ وقت یک بینش خوب از چیستی آن بدست نیاوردیم. یک روش برای فکر کردن به بیش‌برازش این است که یک مسئله جدید را درنظر بگیریم: مسئله برازش یک تابع برای مجموعه‌ای از نقاط. این موضوع برازش منحنی[1] نیز نام دارد. یکی از روش‌های برازش منحنی این است که یک سنجه از خطا روی نقاط داشته باشیم و سپس با تعیین پارامترهای تابع این سنجه را بهینه کنیم. این کار آشنا به‌نظر می‌رسد. این دقیقاً کاری است که هنگام آموزش یک شبکه عصبی انجام می‌دهیم.

به‌عنوان یک نمونه از از برازش منحنی، نقاط زیر را درنظر بگیرید.

[1] curve fitting

x	y
0.00	50.0
0.61	–17.8
1.22	74.1
1.83	29.9
2.44	114.8
3.06	55.3
3.67	66.0
4.28	89.1
4.89	128.3
5.51	180.8
6.12	229.7
6.73	229.3
7.34	227.7
7.95	354.9
8.57	477.1
9.18	435.4
9.79	470.1

ما می‌خواهیم یک تابع y=f(x) را به‌گونه‌ای پیدا کنیم که این نقاط را شرح دهد. یک تابع که ممکن است تابع والدی باشد که این نقاط ،هرچند همراه با نویز، از آن تولید شده‌اند.

معمولاً هنگام برازش منحنی فرم تابع را می‌دانیم و تنها به‌دنبال مقدار پارامترها هستیم. اما اگر فقط بدانیم که تابع یک نوع چندجمله‌ای است و فرم دقیق تابع را ندانیم چه؟ در کل، یک چندجمله‌ای با حداکثر توان n شبیه تابع زیر است.

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n$$

هدف برازش یک چندجمله‌ای به مجموعه داده، پیدا کردن پارامترهای $a_0, a_1, a_2, \ldots, a_n$ است. y مقدار خروجی برای یک ورودی داده شده‌ی x است و f(x) خروجی تابع برازش شده با مجموعه پارامترهای فعلی برای همان ورودی x است. معمولاً پارامترها با کمینه کردن مربعات اختلافات بین این دو مقدار به‌ازای تمامی ورودی‌ها بدست می‌آیند. این کار باید خلی آشنا به‌نظر بیاید زیرا دقیقاً استفاده از این نوع تابع زیان برای آموزش یک شبکه عصبی را قبلاً مورد بحث قرار دادیم.

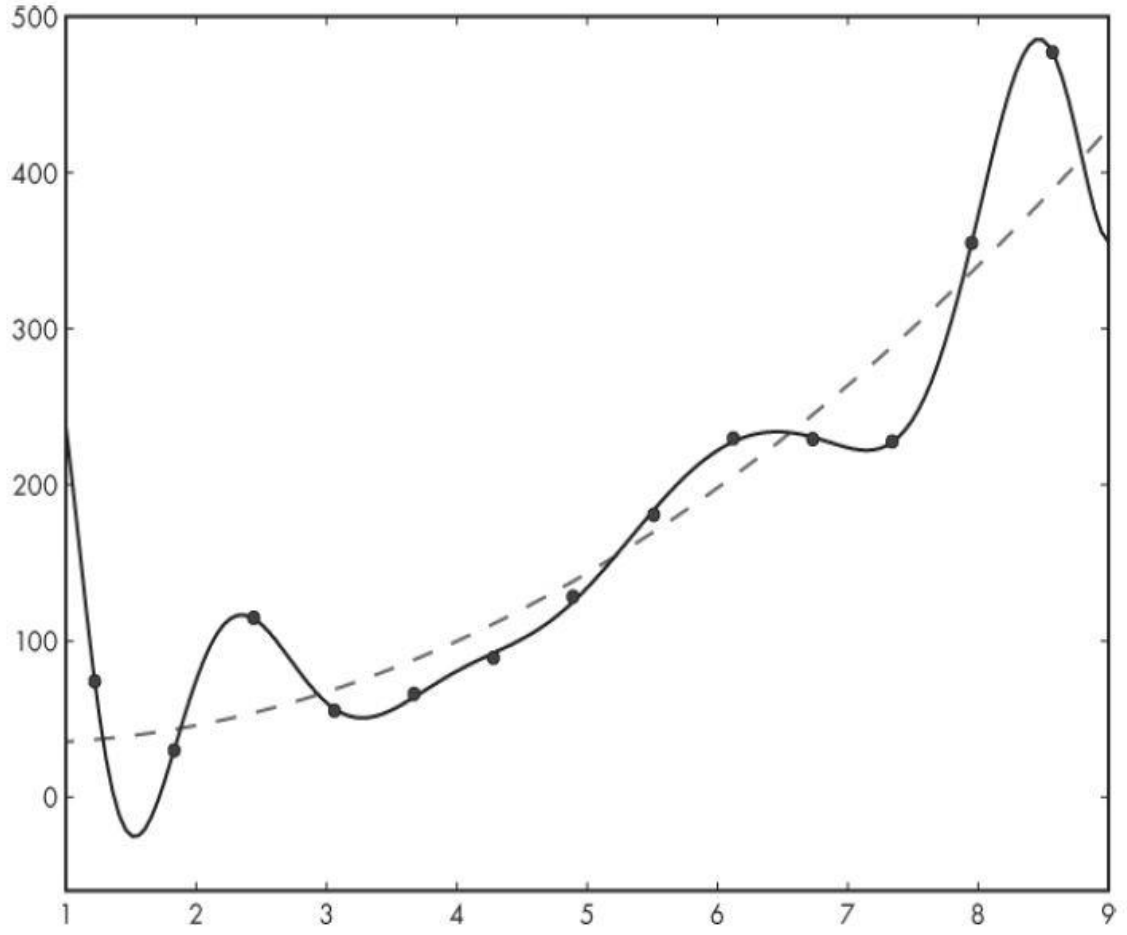
این‌ها چگونه به بیش‌برازش مربوط می‌شوند؟ بیایید نتایج برازش دو تابع متفاوت برای مجموعه داده قبلی را رسم کنیم. اولین تابع به‌صورت زیر است.

$$y = a_0 + a_1x + a_2x^2$$

این یک تابع درجه ۲ است. همان تابعی که وقتی تازه با جبر آشنا شده بودید از آن متنفر بودید. تابع دوم به‌صورت زیر است.

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{14}x^{14} + a_{15}x^{15}$$

این یک تابع درجه ۱۵ است. نتایج در شکل ۵_۹ نشان داده شده‌است.



شکل ۵_۹: یک مجموعه داده و دو تابع برازش شده برای آن: یک تابع درجه۲ (خط چین) و یک تابع درجه۱۵ (خط تو پر)

کدام تابع روند کلی مجموعه داده را بهتر توصیف می‌کند؟ به‌وضوح تابع درجه ۲ از روند کلی داده‌ها تبعیت می‌کند؛ این درحالی است که تابع درجه ۱۵ همه جا هست! یک بار دیگر به شکل ۵_۹ نگاه کنید. اگر تنها معیار خوب بودن ما برای برازش داده‌ها فاصله بین مقدار واقعی y و مقدار برازش شده باشد، خواهیم گفت که تابع درجه ۱۵ برازش بهتری است. این تابع تقریباً از روی تمامی نقاط می‌گذرد. این شبیه آموزش یک شبکه عصبی و بدست آوردن کمال در یک مجموعه آموزش است. هزینه کمال می‌تواند توانایی کم در تعمیم دادن به‌ازای یک ورودی جدید باشد. تابع درجه ۲ در شکل ۵_۹ از روی نقاط عبور نکرد؛ اما روند کلی داده‌ها را توصیف کرد. همین موضوع باعث می‌شود که این تابع سودمندتر باشد و زمانی که بخواهیم پیش‌بینی y مورد انتظار به‌ازای یک مقدار جدید از x را بدست بیاوریم، از این تابع

استفاده می‌کنیم.

وقتی یک انسان می‌خواهد به چیزی شبیه مجموعه داده مثال ما، یک منحنی برازش کند، معمولاً به داده‌ها نگاه می‌کند، روند کلی آن را مورد توجه قرار می‌دهد و براساس این روند یک تابع برای برازش انتخاب می‌کند. همچنین ممکن است که فرم تابع از قبل توسط تئوری ارائه شده باشد. با این وجود، اگر بخواهیم که شبیه به شبکه‌های عصبی باشیم، در موقعیتی هستیم که تابع مناسب برای برازش را نمی‌دانیم و باید از فضای توابع x، بهترین تابع را به همراه پارامترهایش پیدا کنیم.

امیدوارم این مثال یک ایده درمورد این داده باشد که آموزش شبکه‌های عصبی یک مسئله بهینه‌سازی مانند سایر مسائل بهینه‌سازی نیست. ما به "چیزی" نیاز داریم که تابعی که شبکه در حال آموزش آن است را به جهتی هدایت کند که ماهیت داده‌ها توصیف شود، بدون این که بخواهد توجه بیش از حد به برخی از ویژگی‌های داده‌های آموزش بکند. این "چیزی" قاعده‌سازی است و شما به آن نیاز دارید؛ مخصوصاً برای شبکه‌های بزرگ با ظرفیت عظیم.

درک قاعده‌سازی

قاعده‌سازی هرچیزی است که شبکه را مجبور کند که ویژگی‌های مناسب توزیع والد را یاد بگیرد و جزئیات مجموعه آموزش را یاد نگیرد. بهترین نوع قاعده‌سازی افزایش اندازه و طبیعت نمایشی مجموعه داده‌های آموزش است. هرچه مجموعه داده بزرگتر باشد و همه‌ی انواع نمونه‌هایی که شبکه درواقعیت خواهد دید را بهتر نمایش دهد، شبکه بهتر خواهد آموخت. البته که معمولاً ما مجبوریم که با یک مجموعه آموزش محدود کار کنیم. افرادی که در حوزه یادگیری ماشین کار می‌کنند، زمان و انرژی‌های بی‌پایانی را برای این صرف کرده‌اند که یاد بگیرند که چگونه بینش بیشتری از مجموعه‌های داده کوچک بدست بیاورند.

در فصل ۵، شاید دومین راه خوب برای قاعده‌سازی یک مدل را یاد گرفتیم که همان داده‌افزایی[۱] است. این روش یک نماینده از داشتن یک مجموعه داده بزرگتر است. در این روش از داده‌هایی که داریم برای تولید نمونه‌های جدیدی برای مجموعه آموزش استفاده می‌کنیم به‌طوری که بتوان قبول کرد که از توزیع والد هستند. به‌عنوان مثال برای یک مجموعه محدود از تصاویر، عملیات ساده چرخش، معکوس کردن، و تغییر مکان تصاویری که از قبل در مجموعه آموزش بوده‌اند، مد نظر بود. داده‌افزایی قدرتمند است و اگر امکانش وجود داشت باید از آن استفاده کنید. مخصوصاً وقتی تصاویر ورودی‌های شبکه باشند، انجام این کار ساده است. در فصل ۵ یک روش برای تقویت یک مجموعه داده که شامل بردارهایی با اعداد پیوسته

[۱] Data augmentation (تقویت داده هم ترجمه شده‌است)

بود، دیدیم.

حال دو تکنیک در جعبه‌ابزار قاعده‌سازی خود داریم: داده‌های بیشتر و داده‌افزایی. این‌ها بهترین روش‌ها هستند، اما روش‌های دیگری نیز وجود دارد که در صورت امکان باید استفاده کنید. در اینجا دو مورد از آن‌ها بررسی می‌شود: قاعده‌سازی L2 و dropout. امروزه مورد اول استاندارد است و توسط جعبه‌ابزارهایی از جمله sklearn و Keras پشتیبانی می‌شود. مورد دوم قدرتمند است و زمانی که در سال ۲۰۱۲ بوجود آمد بازی را عوض کرد.

قاعده‌سازی L2

مدلی که تعداد کمی وزن با مقادیر بزرگ دارد؛ به‌گونه‌ای پیچیده‌تر از مدلی است که وزن‌های کوچکتری دارد. بنابراین با کوچک نگه داشتن وزن‌ها می‌توانیم امیدوار باشیم که شبکه بتواند یک تابع ساده‌تر را پیاده‌سازی کند، تابعی که برای کاری که ما می‌خواهیم شبکه یاد بگیرد مناسب‌تر باشد.

ما می‌توانیم با استفاده از قاعده‌سازی L2 وزن‌ها را به کوچک بودن تشویق کنیم. قاعده‌سازی L2 به تابع زیان یک جمله اضافه می‌کند و تابع زیان به‌صورت زیر بدست می‌آید.

$$\mathcal{L} = \mathcal{L}(x, y, w, b) + \frac{\lambda}{2}\sum_i w_i^2$$

که در آن اولین جمله همان تابعی که است که تاکنون استفاده می‌کردیم و دومین جمله، جمله جدید قاعده‌سازی L2 است. توجه کنید که زیان تابعی است از ورودی (x)، برچسب (y)، وزن‌ها (w) و بایاس‌ها (b). جمله قاعده‌سازی یک جمع روی تمامی وزن‌های شبکه است و فقط شامل وزن‌ها می‌شود. اسم"L2" باعث شده‌است که ما تمامی وزن‌ها را به توان ۲ برسانیم.

در اینجا L2 به یک نوع از نُرم[۱] یا فاصله اشاره دارد. ممکن است با معادله مربوط به فاصله بین دو نقطه در صفحه آشنایی داشته باشید: $d^2 = (x_2 + x_1)^2 + (y_2 + y_1)^2$. این فاصله اقلیدسی[۲] است، که به آن فاصله L2 نیز گفته می‌شود. علت این اسم این است که مقادیر به توان ۲ رسیده‌اند. همچنین این امکان وجود دارد که یک جمله زیان L1 استفاده شود که در آن به‌جای مربع کردن وزن‌ها، مقادیر قدر مطلق آن‌ها قرار می‌گیرد. در عمل قاعده‌ساز L2 معمول‌تر است و حداقل از لحاظ تجربی، برای شبکه‌های عصبی دسته‌بند، بهتر عمل می‌کند.

ضریب λ اهمیت این جمله را تنظیم می‌کند. هرچه این ضریب بیشتر شود، این جمله بر زیان کلی

---

[۱] norm

[۲] Euclidean distance

استفاده شده در آموزش شبکه، غالب‌تر می‌شود. معمولاً مقادیر λ حدود 0.0005 هستند. در آینده‌ای نزدیک خواهیم دید که چرا به‌جای λ از $\frac{\lambda}{2}$ استفاده شده‌است.

جمله L2 چه کاری انجام می‌دهد؟ به‌یاد بیاورید که زیان چیزی است که در هنگام آموزش می‌خواهیم کمینه کنیم. جمله جدید L2 مربعات وزن‌های شبکه را به توان ۲ می‌رساند. اگر وزن‌ها زیاد باشد، زیان زیاد خواهد بود، و این چیزی است که ما نمی‌خواهیم. وزن‌های کوچکتر، جمله L2 را کوچکتر می‌کند و در نتیجه گرادیان کاهشی طرفدار وزن‌های کوچک می‌شود. علامت وزن‌ها مهم نیست زیرا ما آن‌ها را به توان ۲ رسانده‌ایم. اگر تمامی وزن‌های شبکه نسبتاً کوچک باشند و هیچ کدام از آن‌ها خیلی غالب نباشد، شبکه از تمامی وزن‌ها برای نشان دادن داده‌ها استفاده می‌کند. این کار برای جلوگیری از بیش‌برازش خوب است.

قاعده‌سازی L2 همچنین زوال وزن[۱] هم نامیده می‌شود. این نام گذاری به‌دلیل کاری است که جمله L2 در حین Backprop می‌کند. Backprop به ما مشتق جزئی تابع زیان را نسبت به w_iها می‌دهد. اضافه کردن قاعده‌سازی L2 به این معناست که مشتق جزئی جمله L2 نسبت به هر وزن مشخص، به مشتق جزئی زیان کلی اضافه می‌شود. مشتق $\frac{\lambda}{2}w^2$ به‌صورت λw است. ½ به وسیله توان دوم w حذف می‌شود. از آنجایی که مشتق جزئی را نسبت به یک وزن مشخص مانند w_i می‌گیریم، همه‌ی قسمت‌های دیگر L2 صفر می‌شوند. به‌روزرسانی برای وزن w_i در حین گرادیان کاهشی به‌صورت زیر خواهد شد.

$$w_i \leftarrow w_i - \eta \frac{\partial \mathcal{L}}{\partial w_i} - \eta\lambda w_i$$

که در آن η (اتا) نرخ آموزش است و جملات مازاد مربوط به تکانه را درنظر نگرفته‌ایم. جمله $\eta\lambda w_i$ جدید است. این جمله به خاطر قاعده‌سازی L2 است. می‌توانیم ببینیم که این جمله در فرآیند آموزش در حال هدایت کردن وزن‌ها به سمت صفر است، زیرا η و λ کمتر از ۱ هستند و در هر minibach در حال کم کردن کسر کوچکی از مقادیر وزن‌ها هستیم. وزن‌ها می‌توانند زیاد شوند اما برای اینکار باید گرادیان زیان اصلی بزرگ باشد.

قبلاً گفتیم که فرم تابع زیان به خودمان، به‌عنوان توسعه دهنده شبکه، بستگی دارد. یک قاعده‌ساز تنها نوع از جملاتی نیست که ما می‌توانیم به تابع زیان اضافه کنیم. ما می‌توانیم جملاتی را بسازیم و اضافه کنیم که رفتار شبکه را حین آموزش تغییر دهد و به آن کمک کند که چیزی که ما می‌خواهیم را یاد بگیرد. این یک تکنیک قدرتمند است که می‌تواند برای شخصی سازی جنبه‌های مختلف چیزهایی که شبکه عصبی یاد می‌گیرد استفاده شود

[۱] weight decay (مترجم: تنزل وزن، کاهش وزن هم از لحاظ معنی درست هستند)

Dropout

Dropout زمانی که در سال ۲۰۱۲ آمد، مانند یک طوفان برای یادگیری ماشین بود. مقاله Alex Krizhevsky و همکاران با عنوان زیر را ببینید:

" Imagenet Classification with Deep Convolutional Neural Networks"

تا پاییز سال ۲۰۲۰ به این مقاله بیش از ۷۰۰۰۰ مرتبه ارجاع داده شده‌است. یکی از محققان نام آشنای یادگیری ماشین به من گفت "اگر Dropout را دهه ۱۹۸۰ داشتیم، اکنون دنیای متفاوتی داشتیم". خب Dropout چیست و چرا همه درباره آن اینقدر هیجان زده‌اند؟ برای پاسخ داد به این پرسش باید مفهوم مدل‌های ترکیبی[۱] (ensemble) را بررسی کنیم. ما مقداری درمورد آن‌ها در فصل ٦ صحبت کردیم. یک ensemble یک گروه از مدل‌هاست که به مقدار کمی متفاوت هستند و همگی روی یک مجموعه داده یا نسخه‌های متفاوتی از مجموعه داده که به مقدار کمی تفاوت دارند، آموزش داده شده‌اند. ایده سر راست است: از آن‌جایی که اکثر مدل‌ها ماهیت تصادفی دارند، آموزش چندین مدل مشابه، یک مجموعه را نتیجه می‌دهد که متقابلاً تقویت کننده است. یک مجموعه از خروجی‌ها را می‌توان ترکیب کرد تا نتیجه‌ای تولید شود که بهتر از هر کدام از مدل‌ها به تنهایی، باشد. Ensemble ها مفید هستند و ما بارها از آن‌ها استفاده می‌کنیم. اما مشکل آن‌ها از جنس زمان اجرا[۲] است. اگر x میلی ثانیه طول بکشد که یک نمونه را در یک شبکه عصبی اجرا کنیم و یک ensemble از 20 شبکه داشته باشیم، اگر امکان اجرای موازی را کنار بگذاریم، زمان استنتاج ما به 20x میلی ثانیه تبدیل می‌شود. در بعضی از موقعیت‌ها این کار غیرقابل قبول است (که درمورد ذخیره‌سازی و توان مورد نیاز برای ۲۰ شبکه بزرگ در مقابل یک شبکه چیزی نگوییم). از آنجایی که نتیجه یک ensemble از مدل‌ها از لحاض عملکرد کلی بهتر است، می‌توانیم بگوییم یک ensemble، یک قاعده‌سازی نیز است، زیرا از "خرد جمعی"[۳] استفاده می‌کند.

Dropout ایده ensemble را به یک نقطه حداکثری برد، اما تنها در هنگام آموزش و بدون ساختن هیچ شبکه دیگری. بنابراین در آخر، تنها یک شبکه خواهیم داشت. مانند همه‌ی ایده‌های خوب در آمار، این هم نیاز به تصادفی بودن دارد. هم اکنون، وقتی ما یک شبکه را آموزش می‌دهیم، مسیر رو به جلو (مترجم: همان forward pass) را با استفاده از وزن‌ها و بایاس‌های فعلی طی می‌کنیم. اگر در حین این مسیر رو به جلو، ما به‌طور تصادفی اعداد 0 و 1 را به همه گره‌های شبکه تخصیص دهیم به‌طوری که گره‌های 1 در

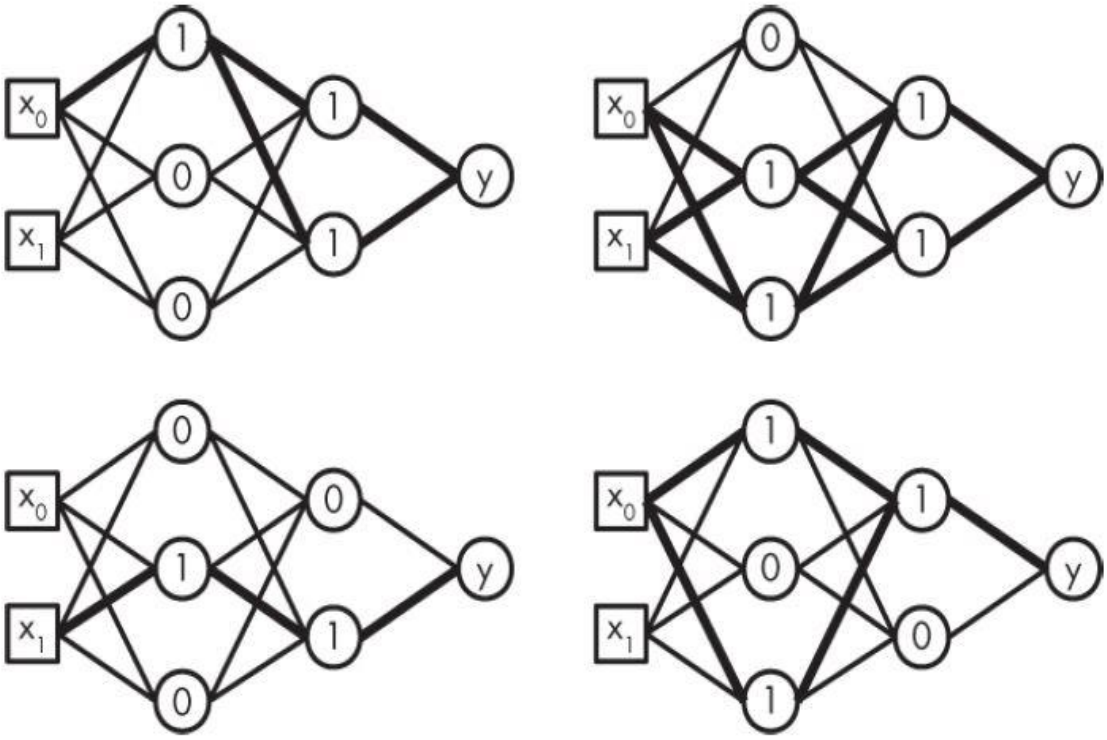
---

[۱] ensembles of models (مدل‌های جمعی نیز ترجمه می‌شود)

[۲] runtime

[۳] wisdom of the crowd

لایه بعدی استفاده شوند[1] و سایر گره‌ها بیرون بیوفتند[2] چه؟ ما به‌طور کارایی، هر بار نمونه‌های آموزش را روی یک پیکربندی متفاوت از شبکه عصبی اجرا می‌کنیم. به‌عنوان مثال شکل ٦_٩ را ببینید.



شکل ٦_٩: شبکه‌های ممکن در هنگام اعمال Dropout در حین آموزش

در اینجا شبکه موجود در شکل ۱_۸ را آوردهایم، اما به هر گره مخفی 0 یا 1 داده‌ایم. این 0 یا 1ها نشان می‌دهند که آیا خروجی استفاده شود یا خیر. خط‌های پر رنگ اتصالاتی که معتبر هستند را نشان می‌دهد. به عبارت دیگر، خطوط تیره شبکه‌ای را نشان می‌دهد که واقعاً استفاده شده‌است. خروجی این شبکه برای Backprop استفاده می‌شود. اگر این کار را روی هر کدام از نمونه‌های آموزش انجام دهیم، به راحتی می‌بینیم که تعداد بسیار زیادی شبکه را آموزش داده‌ایم و هر کدام روی یک نمونه آموزش دیده اند. علاوه بر این، از آنجایی که وزن‌ها و بایاس‌ها بین مسیرهای رو به جلو باقی می‌مانند، این وزن‌ها بین همه شبکه‌ها مشترک است، به این امید که فرآیند مقادیر وزنی که نشان‌دهنده ماهیت مجموعه داده است را تقویت کند. همانطور که بارها در این فصل گفتیم، یادگیری ماهیت مجموعه داده هدف آموزش است. ما می‌خواهیم توان تعمیم دادن به یک داده جدید از همان توزیع والدی که مجموعه آموزش را تولید کرده است، داشته

[1] مترجم: واقعا نویسنده جمله ی به این مهمی را بد نوشته. چرا باید جمله رو به‌صورت پرسشی بنویسه؟ بگذریم. یه توضیح کوچیک: هر گره یه خروجی داره دیگه درسته؟ که این خروجی، به‌عنوان ورودی لایه بعدی استفاده میشه. داره میگه گره‌هایی که ۱ دارند خروجیشون به لایه بعدی میره درحالی که گره‌هایی که صفر دارند نمیره.

[2] مترجم: کلمه Dropout به معنای بیرون انداختن است و در اینجا می توان دید که چرا این اسم را دارد. از بیرون انداختن در متن خود داری کردم تا وقتی که اسمی از Dropout می آید متوجه شوید که منظور همان روش Dropout است و نه فعل بیرون انداختن.

باشیم. Backprop یک قاعده‌سازی جدی است.

قبلاً گفتم که ما به‌طور تصادفی اعداد 0 و 1 را به گره‌ها تخصیص می‌دهیم. آیا به یک اندازه آن‌ها را تخصیص می‌دهیم؟ احتمالی که با آن گره‌ها را بیرون می‌اندازیم، چیزی است که ما باید مشخص کنیم. بیایید این احتمال را p نام گذاری کنیم. معمولاً p=0.5 است؛ بدین معنا که ۵۰ درصد از گره‌ها در یک لایه برای هر نمونه آموزش بیرون انداخته می‌شود. p=0.8، هشتاد درصد از گره‌ها و p=0.1، ده درصد از گره‌ها را بیرون می‌اندازد. گاهی وقت‌ها احتمالات متفاوتی برای لایه‌های متفاوت شبکه استفاده می‌شود. مخصوصاً لایه اول که باید احتمال کمتری نسبت به گره‌های مخفی داشته باشد. اگر تعداد زیادی از گره‌های ورودی را بیرون بیاندازیم، منبع سیگنالی را از دست می‌دهیم که در تلاشیم شبکه آن را تشخیص دهد. اجرای Dropout روی لایه ورودی را می‌توان به‌صورت نوعی داده‌افزایی نگاه کرد.

مفهوماً، Dropout آموزش یک مجموعه بزرگ از شبکه‌هایی است که وزن‌های مشترکی دارند. خروجی هر یک از این شبکه‌ها می‌تواند توسط یک میانگین هندسی ترکیب شود. این کار با فرض استفاده از خروجی softmax امکان‌پذیر است. میانگین هندسی دو عدد، ریشه دوم حاصل ضرب آن‌هاست. میانگین هندسی n عدد، ریشه n ام حاصل ضرب آن‌ها است. درمورد Dropout مشخص شده‌است که این را می‌توان با استفاده از کل شبکه و ضرب کردن همه وزن‌ها در احتمال اینکه بیرون نمی‌افتند، تخمین زد. گفتیم p احتمال این است که یک گره بیرون بیوفتند. در نتیجه 1-p احتمال این است که گره باقی بماند. بنابراین اگر p=0.5 برای همه گره‌ها درنظر گرفته شود، آنگاه شبکه نهایی شبکه‌ای خواهد بود که تمامی وزن‌هایش تقسیم بر ۲ شده‌است.

در موقع نوشتن این متن، MLPClassifier موجود در sklearn از Dropout پشتیبانی نمی‌کند اما Keras می‌کند. بنابراین ما دوباره Dropout را در فصل ۱۲ خواهیم دید.

خلاصه

چون این فصل مهم است، بیایید یک مروری بکنیم روی چیزهایی که یاد گرفتیم. در این فصل یاد گرفتیم که چطور یک شبکه را با استفاده از گرادیان کاهشی و پس‌انتشار آموزش دهیم. ترتیب کلی مراحل به شرح زیر است:

۱. معماری شبکه را انتخاب کنید. این یعنی تعداد لایه‌ها، اندازه آن‌ها و نوع تابع فعال‌سازی

۲. وزن‌ها و بایاس‌ها را با استفاده از مقادیر اولیه‌ای که هوشمندانه انتخاب شده‌اند، مقداردهی کنید.
۳. یک minibatch از نمونه‌های آموزش را از طریق شبکه اجرا کنید و میانگین زیان را روی minibatch محاسبه کنید. ما توابع زیان مرسوم را بحث کردیم.
٤. با استفاده از پس‌انتشار، نقش هر یک از وزن‌ها و بایاس‌ها را در زیان کلی در minibatch محاسبه کنید
۵. با استفاده از گرادیان کاهشی و براساس سهمی که از پس‌انتشار بدست آمده، مقادیر وزن‌ها و بایاس‌ها را به‌روزرسانی کنید.
٦. از مرحله ۳ تکرار کنید تا زمانی که به تعداد اپوک یا minibatchهای مورد نظر برسید، یا اینکه مقدار زیان کمتر از یک آستانه شود، یا زیان خیلی تغییر نکند، یا امتیاز (مترجم: همان دقت) در یک مجموعه از داده‌های ارزیابی به حداقل مقدار خود برسد.
۷. اگر شبکه به خوبی آموزش نمی‌بیند، قاعده‌سازی را روی آن اعمال کنید و دوباره امتحان کنید. ما قاعده‌سازی L2 و dropout را در این فصل پوشش دادیم. داده‌افزایی، افزایش اندازه یا افزایش نمایش طبیعت داده‌های آموزش نیز می‌توانند به‌عنوان قاعده‌سازی دیده شوند.

هدف آموزش یک شبکه عصبی یادگیری پارامترهایی از مدل است که به خوبی روی ورودی‌های دیده نشده تعمیم می‌شوند. این هدفِ تمامیِ یادگیریِ ماشینِ با نظارت[۱] است. برای یک شبکه عصبی می‌دانیم که با ظرفیت و مجموعه داده کافی می‌تواند هر تابعی را تخمین بزند. اگر ساده نگاه کنیم، ما کاری بیش از بهینه‌سازی معمولی انجام نمی‌دهیم، اما از جهات مهمی اینطور نیست. کمال روی مجموعه داده معمولاً چیز خوبی نیست. معمولاً نشانه‌ای از بیش‌برازش است. به جای آن، ما می‌خواهیم که مدل تابعی را یاد بگیرد که طبیعت واقعی مجموعه آموزش را بازتاب کند. ما از داده تست استفاده می‌کنیم تا به ما اطمینان دهد که یک تابع مفید را آموزش داده ایم.

در فصل بعدی در واقعیت از شبکه‌های عصبی سنتی استفاده می‌کنیم و با استفاده از sklearn به تجربه خود می‌افزاییم.

---

[۱] supervised machine learning