فصل چهارم: شبکه‌های عصبی مصنوعی

شبکه‌های عصبی مصنوعی[1] یا همان ANNها متدی کلی و کاربردی برای یادگیری توابع حقیقی مقدار، گسسته مقدار و برداری از روی نمونه‌هاست. الگوریتم‌هایی یادگیری شبکه‌ای چون Backpropagation از روش‌هایی چون شیب نزول[2] استفاده کرده تا پارامترهای شبکه را طوری تنظیم کنند تا با دسته نمونه‌های آموزشی مطابقت داشته باشند. یادگیری شبکه‌های عصبی در مقابل خطاها در داده‌های آموزشی مقاوم[3] است و در تفسیر صحنه‌های تصویری، تشخیص صحبت و یادگیری استراتژی‌های کنترل ربات به کاربرد دارد.

٤,١ معرفی

متد یادگیری شبکه‌های عصبی روش‌هایی مقاوم به نویز برای تخمین توابع هدف حقیقی مقدار، گسسته مقدار و برداری ارائه می‌کند. در انواع خاصی از مسائل مثل یادگیری تفسیر ورودی‌های پیچیده‌ی حسگرها، شبکه‌های عصبی بهترین روش شناخته شده هستند. برای مثال، الگوریتم Backpropagation، که در این فصل کاملاً آن را توضیح خواهیم داد، موفقیت‌های چشم‌گیری در حل مسائل کاربردی‌ای نظیر تشخیص کاراکترهای دست‌نویس (LeCun et al. 1989) ، تشخیص صحبت (Lang et al. 1990) و تشخیص چهره (Cottrell 1990) از خود نشان داده است. بررسی‌ای از کاربردهای واقعی شبکه‌های عصبی توسط (Rumelhart et al. 1994) گردآوری شده است.

٤,١,١ انگیزه‌ی زیستی

مطالعه‌ی شبکه‌های عصبی مصنوعی از سیستم‌های یادگیر زیستی که از شبکه‌های خیلی پیچیده‌ی اعصاب ساخته شده‌اند الهام گرفته شده است. در نگاه سطحی، این سیستم‌ها از انبوهی از دسته واحدهای متصل به هم ساده ساخته شده‌اند که هر واحد ورودی‌های حقیقی مقداری

[1] Artificial neural networks
[2] Gradient decend
[3] robust

دریافت کرده (بیشتر این ورودی‌ها خروجی‌های واحدهای دیگر هستند) و مقدار حقیقی‌ای را محاسبه می‌کند (که ممکن است ورودی واحدهای دیگری باشد).

برای درک بهتر، چند حقیقت از عصب‌شناسی را با هم ملاحظه می‌کنیم. برای مثال، تخمین زده می‌شود که مغز انسان 10^{11} عصب دارد که هر کدام به طور متوسط به 10^{4} عصب دیگر متصل‌اند. فعالیت عصب به طور عادی در یکی از دو حالت برانگیخته[۱] و غیر برانگیخته[۲] است. سریع‌ترین اعصاب در مرتبه‌ی 10^{-3} ثانیه بین این دو حالت سوییچ می‌کنند (این مقدار در مقابل کامپیوترها 10^{-10} مرتبه‌ی کندتر است). با این حال انسان می‌تواند به سرعت تصمیمات بسیار پیچیده‌ای بگیرد. برای مثال، شما تصویر مادرتان را در مدت حدوداً 10^{-1} ثانیه تشخیص می‌دهید. توجه دارید که در مدت این 10^{-1} ثانیه، با توجه به سرعت عملکرد اعصاب، اعصاب حداکثر چند صد بار برانگیخته شده‌اند. مشاهدات نشان داده که قدرت پردازش اطلاعات در سیستم‌های عصبی زیستی ناشی از عملیات‌های موازی بسیاری است که بر روی تعداد زیادی از اعصاب اجرا می‌شوند. یکی از انگیزه‌های به‌کارگیری شبکه عصبی رسیدن به چنین محاسبات موازی‌ای که توسط تعدادی زیادی واحد انجام می‌شود است. با وجود اینکه الگوریتم‌های سریع‌تری بر روی ماشین‌های محاسبه‌ی موازی استفاده شده و همچنین سخت‌افزارهای خاصی برای برنامه‌های شبکه عصبی طراحی شده، اما اکثر برنامه‌های شبکه عصبی بر روی ماشین‌هایی ترتیبی[۳] اجرا می‌شوند که عمل محاسبه‌ی غیرمتمرکز را شبیه‌سازی می‌کنند.

با وجود اینکه شبکه‌های عصبی برداشتی از شبکه‌های عصبی زیستی است، اما بسیاری از پیچیدگی‌های شبکه‌های عصبی زیستی در شبکه‌های عصبی مدل‌سازی نمی‌شود، همان‌طور که بسیاری از خواص شبکه‌های عصبی (که درباره‌ی آن‌ها بحث خواهیم کرد) با سیستم‌های زیستی مطابقت ندارد. برای مثال، ما فرض می‌کنیم که واحدهای شبکه عصبی یک سیگنال خروجی دارند، درحالی‌که در اعصاب زیستی خروجی سری‌ای ترکیبی از ضربه‌ها در طول زمان است.

بر اساس تاریخچه، دو دسته از محققان بر روی شبکه‌های عصبی مصنوعی کار می‌کردند. گروه اول که سعی داشتند با تقلید شبکه‌های عصبی فرایندهای یادگیری زیستی را مطالعه و مدل‌سازی کنند و گروه دوم کسانی که سعی داشتند به الگوریتم‌های یادگیری ماشین مؤثری دست یابند، جدا از اینکه این الگوریتم‌ها از شبکه‌های عصبی به دست آمده است. در طول این کتاب، ما نیز جزو گروه دوم محسوب می‌شویم و بنابراین بر مدل‌سازی زیستی تأکید نمی‌کنیم. برای اطلاعات بیشتر در مورد مدل‌سازی سیستم‌های زیستی توسط شبکه‌های عصبی می‌توانید به کتب زیر مراجعه کنید:

Churchland and Sejnowski (1992);

Zornetzer et al. (1994);

Gabriel and Moore (1990).

[۱] excited
[۲] inhibited
[۳] sequential

٤,٢ معرفی شبکه‌های عصبی

یک نمونه‌ی تمام عیار از یادگیری شبکه عصبی توسط سیستم Pomerleau (1993)، به نام ALVINN شبیه‌سازی شده است. این سیستم برای کنترل فرمان اتومبیل با سرعت متوسط در بزرگراه‌ها طراحی شده است. ورودی این شبکه‌ی عصبی یک تصویر 30x32 نقطه‌ای است که از دوربین رو به جلویی که در داخل اتومبیل کار گذاشته شده گرفته می‌شود. خروجی شبکه‌ی عصبی جهتی است که فرمان بـه آن سـمت بایـد بچرخد. شبکه برای تقلید فرمان دهی انسان در طول حدود ۵ دقیقاً آموزش داده می‌شود. ALVINN موفق شده تا با شبکه‌ی آمـوزش دیـده‌ی خود، خودرو را تا سرعت ۷۰ مایل در ساعت و برای مسافت ۹۰ مایل در بزرگراه کنترل کند (راننـدگی‌ای کـه در خـط سـرعت بزرگـراه بـوده و بزرگراه نیز خط‌کشی شده بوده و دیگر وسایل نقلیه هم در بزرگراه حضور داشته‌اند).

شکل ۴,۱ شبکه‌ی عصبی استفاده شده در یکی از نسخه‌های ALVINN و نحوه‌ی نمایش بسیاری از شبکه‌های عصبی را نشان می‌دهد. شبکه با نمونه‌ای از تصویرهای دوربین در چپ تصویر نشان داده شده است. هر واحد[۱] (در شکل دایره) در شبکه نشان‌دهنده‌ی خروجی یک واحد است و خطوط متصل به زیر هر واحد نیز ورودی‌های آن هستند. همان‌طور که در شکل نیز نشان داده شده ۴ واحد مستقیماً به تمـامی نقـاط تصـویر متصل‌اند. این چهار واحد پنهان[۲] نامیده می‌شوند زیرا که بر خروجی به طور غیرمستقیم اثر می‌کنند و هیچ گاه به صورت مستقیم تأثیری ندارنـد. هر یک از این چهار واحد خروجی‌ای بر اساس ۹۶۰ ورودی وزن‌دار خود ایجاد می‌کنند. خروجی این ۴ واحد پنهان به عنوان ورودی به ۳۰ واحـد خروجی[۳] داده می‌شود. هر واحد خروجی متناسب با یکی از فرمان‌های جهتی اتومبیل است (مثل کمی به راست، کمی به چپ، کاملـاً بـه چـپ ، کاملاً به راست یا مستقیم) و این خروجی‌ها نشان می‌دهد که کدام جهت برای فرمان ارجحیت دارد.

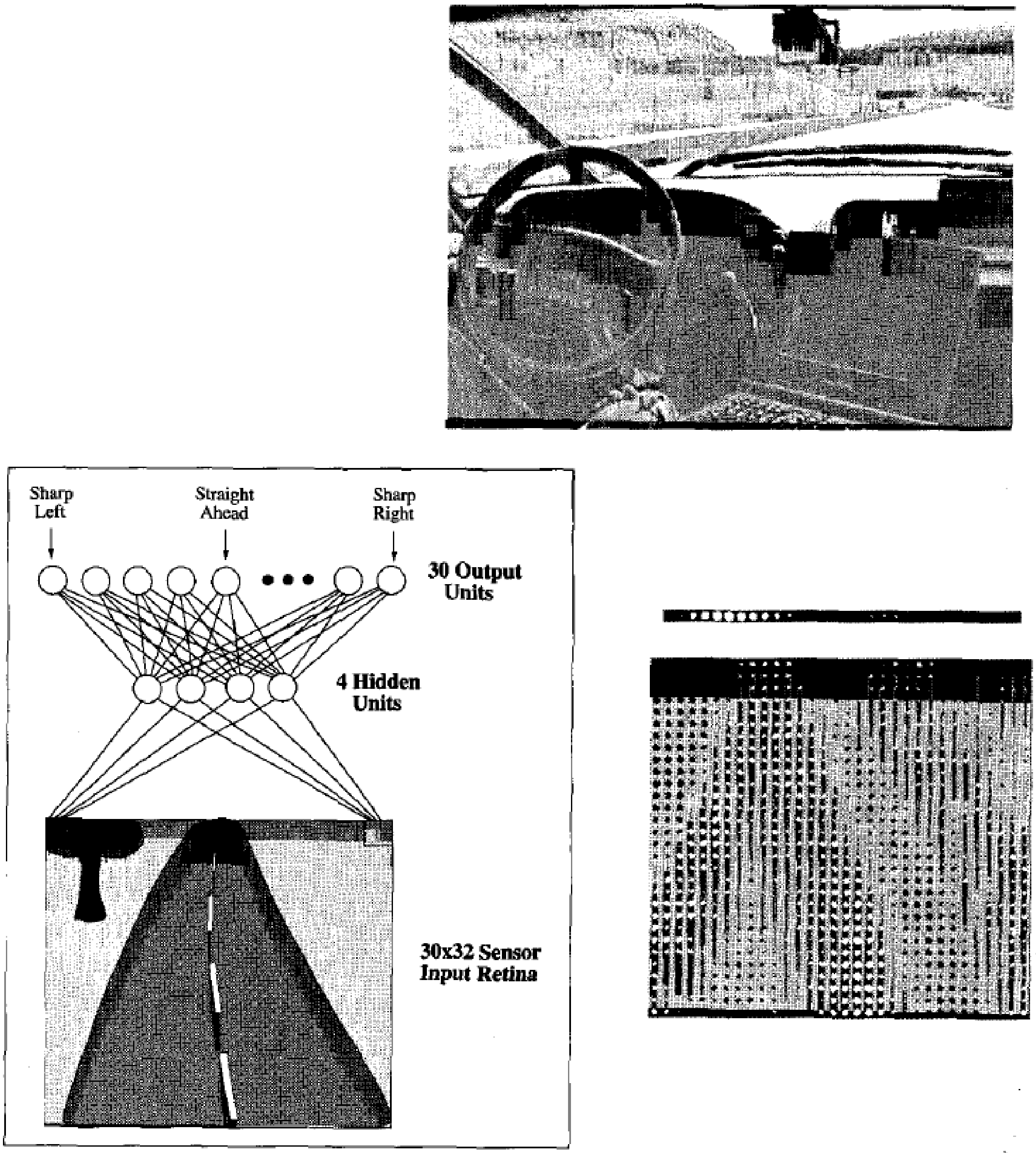
سمت راست شکل وزن‌های[۴] یاد گرفته شده‌ی متناسب با یکی از این چهار واحد را نمایش می‌دهد. ماتریکس سیاه و سفیدی که در سمت راست و پایین شکل نشان داده شده وزن‌های متناسب با نقطه‌ها در یکی از ۴ واحد پنهان است. در این شکل مربع‌های سیاه نشان‌دهنده‌ی وزن منفـی و مربع‌های سفید نشان‌دهنده‌ی وزن مثبت‌اند و اندازه‌ی مربع نیز بزرگی وزن را نشان می‌دهد. مستطیل بالـای مربـع وزن‌هـای ورودی از هـر ۴ واحد پنهان به ۳۰ خروجی را نشان می‌دهد.

ساختار شبکه‌ی ALVINN در بسیاری از شبکه‌های عصبی به کار برده می‌شود. در این چنین شبکه‌هایی فقط ارتباط‌های بین لایه‌ای وجود دارد و گراف جهت‌دار نظیر دور ندارد[۵]. در کل، ساختار شبکه‌ها ممکن است هر نوع گرافی باشند (اعم از دور دار[۶] و بدون دور[۷]، جهت‌دار[۸] و یـا بـدون جهت[۹]) . در این فصل به بررسی پرکاربردترین و عمومی‌ترین ویژگی‌های شبکه‌های عصبی که بـر پایـه‌ی الگـوریتم Backpropagation است می‌پردازیم. الگوریتم Backpropagation فرض می‌کند که شبکه ساختاری ثابت و متناسب با یک گـراف جهـت‌دار دارد کـه ممکـن

---

[۱] node
[۲] hidden
[۳] output
[۴] Weight value
[۵] Directed acyclic graph
[۶] cyclic
[۷] acyclic
[۸] directed
[۹] adirected

است دور نیز داشته باشد. و سعی می‌کند تا مقادیر متناسب با هر یال[1] در این گراف را یاد بگیرد. با وجود اینکه حلقه در شبکه مجاز است اما اکثر شبکه‌های کاربردی بدون حلقه و به فرم تک‌سویه[2] هستند، درست مثل ساختار شبکه‌ی ALVINN.



شکل ۴,۱ شبکه‌ی عصبی‌ای که برای کنترل فرمان خودرو طراحی شده.

[1] edge
[2] feed-forward

سیستم ALVINN از Backpropagation با هدف یادگیری کنترل خودرو استفاده می‌کند (شکل بالایی). شکل سمت چپ نشان می‌دهد که چگونه ۹۶۰ نقطه‌ی تصویر دوربین به ۴ واحد پنهان متصل به ۳۰ واحد خروجی متصل شده‌اند. خروجی شبکه فرمان‌های کنترل فرمان خواهد بود. شکل سمت راست وزن‌های نظیر یکی از واحدهای پنهان را نشان می‌دهد. وزن‌ها در یک ماتریس 30x32 نشان داده شده‌اند. در این ماتریس وزن‌های مثبت سفید و وزن‌های منفی سیاه رنگ‌اند، اندازه‌ی هر وزن متناسب با اندازه‌ی هر مربع است. مستطیل کوچک بالای این ماتریس وزن‌های ۳۰ واحد خروجی متصل به این واحد را مشخص می‌کند. همان‌طور که در مستطیل نیز معلوم است تهییج این واحد پنهان باعث گردش به چپ می‌گردد.

٤,٣ مسائل متناسب با یادگیری شبکه‌های عصبی

یادگیری شبکه‌های عصبی مناسب مسائلی با داده‌های ورودی نویز دار یا ترکیبی از چندین حسگر[1] مثل دوربین و میکروفن است. همچنین در مسائلی که کاملاً به صورت نمادی[2] بیان می‌شود، مثل مسائلی که در فصل ۳ بررسی شد، کاربرد دارند. در چنین مسائلی شبکه‌های عصبی و درخت تصمیم‌گیری دقت قابل‌مقایسه‌ای دارند. در (Shvlik et al. (1991 و (Weiss and Kapouleas (1989 مقایسه‌های تجربی این دو راهبرد برای مسائل مختلف بررسی شده است. الگوریتم Backpropagation پرکاربردترین متد یادگیری شبکه‌های عصبی محسوب می‌شود. این الگوریتم برای مسائلی با ویژگی‌های زیر متناسب است:

- نمونه‌ها به صورت nتایی‌های مرتب‌اند. تابع هدف بر روی نمونه‌هایی تعریف شده که توسط بردارهایی از ویژگی‌ها بیان می‌شوند، مثل مقدار نقطه‌های تصویر در مثال ALVINN. این مقادیر ممکن است کاملاً وابسته[3] و یا کاملاً مجزا[4] باشند. مقادیر ورودی می‌توانند هر مقدار حقیقی‌ای باشند.
- تابع هدف ممکن است گسسته مقدار، حقیقی مقدار، یا برداری ترکیبی از حقیقی مقدار و گسسته مقدار باشد. برای مثال در ALVINN خروجی برداری از ۳۰ ویژگی است. خروجی هر یک از مقادیر حقیقی بین ۰ تا ۱ را می‌تواند داشته باشد، در این مثال این مقدار اطمینان شبکه به پیچیدن به آن جهت را مشخص می‌کند. همچنین می‌توان شبکه‌ای آموزش داد که علاوه بر کنترل فرمان کنترل سرعت اتومبیل را نیز در اختیار داشته باشد. کافی است مقداری برای کنترل شتاب به خروجی‌ها اضافه کنیم.
- نمونه‌ها ممکن است خطا داشته باشند. متدهای یادگیری شبکه عصبی در مقابل داده‌های آموزشی نویز دار مقاوم است.
- زمان آموزش زیاد قابل‌قبول است. الگوریتم‌های آموزش شبکه زمانی بیشتر از آموزش‌های دیگر الگوریتم‌ها (مثلاً درخت تصمیم‌گیری) لازم دارند. زمان آموزش‌ها بسته به تعداد وزن‌های در شبکه ، تعداد نمونه‌های آموزشی، و تنظیمات پارامترهای الگوریتم یادگیری ممکن است از چند ثانیه تا چندین ساعت تغییر کند.
- ارزیابی سریع تابع هدف لازم باشد. با وجود اینکه شبکه‌های عصبی به نسبت کند آموزش داده می‌شوند، اما ارزیابی نمونه‌های جدید توسط شبکه‌ی آموزش دیده بسیار سریع انجام می‌گردد. برای مثال در ALVINN هر ثانیه چندین بار شبکه‌ی عصبی خود را ارزیابی می‌کند و دستورات فرمان دهی را تغییر می‌دهد.
- قدرت انسان برای درک تابع هدف یاد گرفته شده مهم نیست! گاهی تفسیر مقادیر یاد گرفته شده برای وزن‌ها ممکن نیست و قوانین شبکه‌های عصبی آموزش دیده برای انسان به سادگی قابل‌درک نیستند.

---

[1] sensor
[2] Symbolic representation
[3] correlated
[4] independent

در ادامه‌ی این فصل: ابتدا انواع طراحی واحد را در شبکه‌های عصبی بررسی خواهیم کرد (واحدهای پرسپترون[۱] ، واحدهای خطی و واحدهای سیگموید[۲])، سپس به الگوریتم‌های آموزش تک واحدها می‌پردازیم. پس از آن، الگوریتم Backpropagation را برای آموزش شبکه‌های چندلایه ساخته شده از چنین واحدهایی بیان خواهیم کرد و به مطالب کلی‌تری نظیر قابلیت‌های شبکه‌ها، مسئله‌ی overfit و جایگزین‌های Backpropagation می‌پردازیم. و در آخر نیز یک مثال توضیحی از استفاده الگوریتم Backpropagation برای آموزش شبکه با هدف تشخیص چهره آورده‌ایم تا خواننده بتواند از این الگوریتم برای آموزش شبکه استفاده‌ی کاربردی کند.

٤,٤ پرسپترون‌ها

نوعی از سیستم‌های شبکه عصبی بر پایه‌ی نوعی واحد به نام پرسپترون ساخته می‌شود (شکل ۴,۲). پرسپترون برداری حقیقی مقدار دریافت کرده و ترکیبی خطی از آن را محاسبه می‌کند، اگر این مقدار از مقدار خاصی (مقدار آستانه)[۳] بیشتر بود خروجی را ۱ و در غیر این صورت خروجی را 1- می‌دهد. به صورت دقیق‌تر، اگر $x_1, \dots, x_n$ ورودی‌ها باشند و $o(x_1, \dots, x_n)$ خروجی واحد باشد داریم:

$$o(x_1, \dots, x_n) = \begin{cases} 1\,, & w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n > 0 \\ -1\,, & \text{در غیر این صورت} \end{cases}$$

در این تابع هر w_i مقدار حقیقی ثابتی یا همان وزن است که میزان تأثیر x_i را در خروجی پرسپترون تعیین می‌کند. توجه دارید که مقدار w_0 نیز یک مقدار آستانه است و ترکیب $w_1x_1 + w_2x_2 + \cdots + w_nx_n$ باید حداقل مقدار $(-w_0)$ را داشته باشد تا خروجی ۱ شود.

برای ساده‌تر شدن نمایش فرض می‌کنیم که $x_0 = 1$، و به جای عبارت شرط می‌نویسیم $\sum_{i=0}^{n} w_ix_i > 0$ و یا به صورت برداری داریم که $\vec{w}.\vec{x} > 0$. و برای اختصار می‌نویسیم

$$o(\vec{x}) = sgn(\vec{w}.\vec{x})$$

در این نمایش تابع sgn همان تابع علامت است:

$$sgn(y) = \begin{cases} 1\,, & y > 0 \\ -1\,, & \text{در غیر این صورت} \end{cases}$$

یادگیری برای پرسپترون به معنای پیدا کردن مقدار مناسب برای $w_0, \dots, w_n$ است. پس فضای فرضیه‌ای متناسب با این یادگیری تمام بردارهای حقیقی مقدار خواهند بود:

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

---

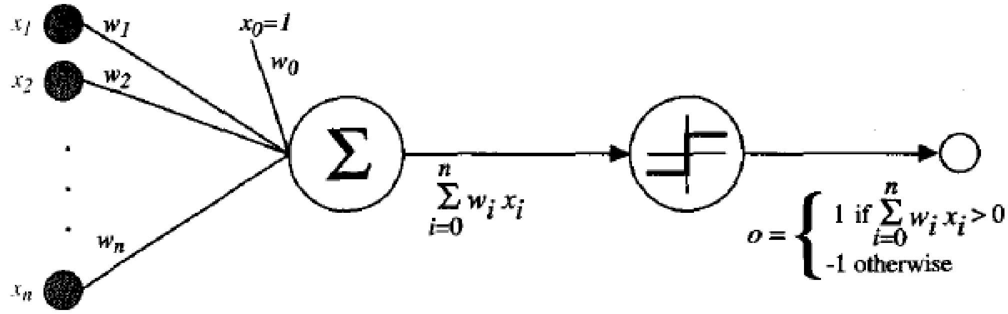
[۱] perceptron

[۲] sigmoid

[۳] threshold

٤,٤,١ قدرت پرسپترون‌ها

ما می‌توانیم پرسپترون را ابر صفحه‌ای[۱] سطح تصمیم در فضای n بعدی نمونه‌ها بدانیم. پرسپترون برای نمونه‌هایی که در یک‌طرف این ابر صفحه هستند ۱ و برای نمونه‌هایی که در طرف دیگر این ابر صفحه هستند مقدار 1- را بر می‌گرداند (شکل ۴,۳). معادله‌ی این ابر صفحه‌ی تصمیم‌گیری به فرم $\vec{w}.\vec{x} = 0$ نوشته می‌شود. البته تمامی دسته نمونه‌های آموزشی را نمی‌توان بدین شکل دسته‌بندی کرد. دسته مثال‌هایی را که این گونه دسته‌بندی می‌شوند دسته‌بندی پذیر خطی[۲] می‌نامند.



شکل ۴,۲ یک پرسپترون.

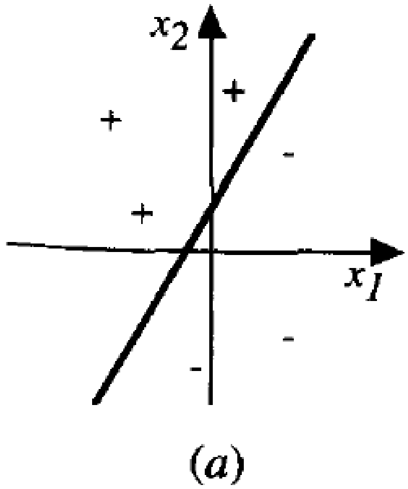
پرسپترون‌ها به تنهایی می‌توانند بسیاری از توابع منطقی مقدار را یاد بگیرند. برای مثال اگر مقدار ۱ را درست (True) و مقدار 1- را غلت (False) در نظر بگیریم برای شبیه‌سازی تابع AND می‌توان وزن‌ها را به صورت $w_1 = w_2 = .5$ و $w_0 = -.8$ را در نظر گرفت. همین پرسپترون را می‌توان با عوض کردن مقدار w_0 به 3.- به تابع OR تبدیل کرد. در واقع توابع AND و OR را می‌توان به صورت توابع خاص m از n دانست: توابعی که زمانی مقدار درست را بر می‌گردانند که حداقل m تا از n ورودی‌شان درست باشد. در تابع OR، m=1 و در تابع AND، m=n است. چنین توابعی را به سادگی می‌توان با یکی کردن وزن‌ها و تعیین مقدار متناسب w_0 توسط پرسپترون‌ها تقلید کرد.

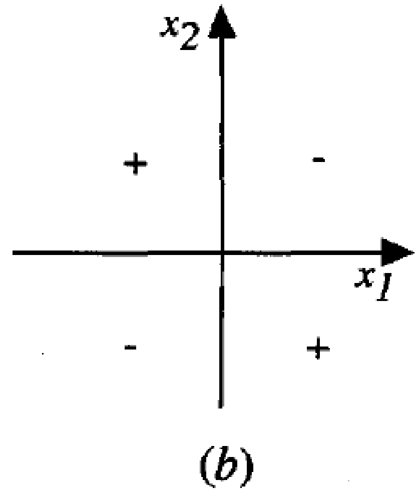
پرسپترون‌ها تمامی توابع ساده‌ی منطقی اعم از AND، OR، NAND (¬AND) و NOR (¬OR) را تقلید می‌کنند. اما متأسفانه پرسپترون‌ها نمی‌توانند توابعی همچون XOR را تقلید کنند. تابع XOR زمانی درست است که $x_1 \neq x_2$. در شکل ۴,۳ (b) تمام نمونه‌های آموزشی مربوط به XOR نشان داده شده است.

---

[۱] hyperplane

[۲] linearly separable





شکل ۴٫۳ فضای مثال‌ها برای پرسپترون‌هایی که دو ورودی دارند.
(a) نمونه‌هایی آموزشی و فضای آن‌ها که پرسپترون آن‌ها را درست دسته‌بندی می‌کند. (b) دسته مثال‌هایی آموزشی که دسته‌بندی پذیر خطی نیستند (با هیچ خطی نمی‌توان نمونه‌های مثبت را از نمونه‌های منفی جدا کرد). نمونه‌های مثبت با "+" و نمونه‌های منفی با "−" در شکل نشان داده شده است.

قدرت پرسپترون‌ها برای یادگیری توابع AND، OR، NAND و NOR از این رو اهمیت دارد که تمامی توابع منطقی توسط این ترکیب توابع شبیه‌سازی هستند. در واقع تمامی توابع منطقی توسط دو سری از پرسپترون‌های متصل به هم (خروجی سری اول به ورودی سری دوم وصل باشد) قابل شبیه‌سازی‌اند. یک راه معمول بیان توابع به صورت فصلی از توابع پایه است (برای مثال، فصلی (OR) از عطف‌های (AND) بین ورودی‌ها و نقیضشان). توجه داشته باشید که می‌توان به سادگی تمام ورودی‌ها را با تغییر علامتشان وزنشان نقیض کرد.

چون که شبکه‌ی واحدهای آستانه‌ای می‌توانند دسته‌ی وسیعی از توابع را یاد بگیرند (در مقابل تک واحدهای آستانه‌ای که فقط تعداد کمی از توابع را یاد می‌گیرند)، علاقه‌ی ما بیشتر به شبکه‌های چندلایه‌ی این نوع واحدهاست.

٤,٤,٢ قانون آموزش پرسپترون‌ها

با وجود اینکه علاقه‌ی ما بیشتر به شبکه‌هایی با تعداد زیاد از واحدهاست، اما بیایید از نحوه‌ی یادگیری وزن‌های یک تک پرسپترون شروع کنیم. اینجا مسئله این است که برداری از وزن‌ها را بیابیم که پرسپترون با آن بتواند تمامی برای نمونه‌های آموزشی خروجی درست را تعیین کند.

برای حل چنین مسائلی الگوریتم‌های بسیاری وجود دارد. در اینجا ما به بررسی دو تا از این الگوریتم‌ها می‌پردازیم: قانون پرسپترون[۱] و قانون دلتا[۲] (نسخه‌ای از قانون LMS که در فصل ۱ برای یادگیری تابع ارزیابی استفاده شد). این دو الگوریتم تضمین می‌کنند که در شرایط خاص مختلف به فرضیه‌های مختلف قابل‌قبولی میل کنند. اهمیت چنین الگوریتم‌هایی از آن جهت است که پایه‌ی یادگیری برای شبکه‌ی با تعداد بالای واحد هستند.

[۱] perceptron rule
[۲] delta rule

یکی از راه‌های یادگیری بردار وزن‌ها ایجاد برداری تصادفی و امتحان کردن آن با تک‌تک نمونه‌های آموزشی است، اگر با خروجی یکی از نمونه سازگار نبود، وزن‌ها را عوض می‌کنیم، این فرایند آن‌قدر ادامه می‌یابد تا برداری پیدا شود که با تمامی نمونه‌های آموزشی سازگار باشد. بر اساس قانون آموزشی پرسپترون در هر مرحله وزن‌ها به صورت زیر تغییر می‌کنند:

$$w_i \leftarrow w_i + \Delta w_i$$

که در آن

$$\Delta w_i = \eta(t - o)x_i$$

در این رابطه t خروجی تابع هدف برای نمونه فعلی، o خروجی پرسپترون و η ثابتی به نام ضریب یادگیری[1] است. نقش ضریب یادگیری کنترل میزان تغییر وزن‌ها در هر مرحله است. ضریب یادگیری معمولاً عددی کوچک (مثلاً 0.1) است که با زیاد شدن تعداد تکرارها کم‌کم کمرنگ می‌شود.

چرا باید چنین فرایندی به سمت مقادیر درست برای وزن‌ها میل کند؟ برای درک بهتر، حالتی خاص را بررسی می‌کنیم. فرض کنید که نمونه‌های آموزشی همگی توسط پرسپترون درست دسته‌بندی می‌شوند در چنین شرایطی همیشه مقدار عبارت (t-o) صفر و متعاقباً Δw_i خواهند بود پس وزن‌ها تغییر نخواهند کرد. حال فرض کنید که پرسپترون برای یک مثال که خروجی 1+ است اشتباهاً خروجی 1- می‌دهد. برای اینکه این اشتباه تصحیح شود وزن‌ها باید طوری تغییر کنند که مقدار $\vec{w}.\vec{x}$ بیشتر شود. مثلاً اگر $x_i > 0$، با افزایش w_i می‌توان مقدار پرسپترون را درست کرد. توجه داشته باشید که چون (t-o)، η، و x_i در این مثال همگی مثبت‌اند w_i افزایش می‌یابد. برای مثال اگر

$$x_i = .8 \text{ و } \eta = .1 \text{ و } t = 1 \text{ و } o = -1$$

خواهیم داشت که

$$\Delta w_i = \eta(t - o)x_i = 0.1\big(1 - (-1)\big)0.8 = 0.16$$

از طرف دیگر اگر t=-1 و o=1 مقدار تغییر وزن به صورت عکس در می‌آمد و x_i کاهش می‌یافت.

در واقع، ثابت می‌شود که فرایند بالا به در طی تعداد محدودی تکرار به برداری از وزن‌ها خواهد رسید که تمامی نمونه‌های آموزشی را درست دسته‌بندی می‌کند (به شرط آنکه نمونه‌های آموزشی دسته‌بندی پذیر خطی و η نیز به اندازه‌ی کافی کوچک باشد (Papert 1969)). اگر داده‌ها دسته‌بندی پذیر خطی نباشند اطمینانی نیست که داده‌ها به مقدار خاصی میل کنند.

۴,۴,۳ شیب نزول[2] و قانون دلتا

با وجود اینکه قانون پرسپترون زمانی که داده‌ها دسته‌بندی پذیر خطی باشند به درستی برداری برای وزن‌ها پیدا می‌کند، اما در زمانی که داده‌ها دسته‌بندی پذیر خطی نیستند در این کار شکست می‌خورد. قانون آموزش دومی، به نام قانون دلتا، طراحی شده که حتی با وجود چنین مشکلی به مقدار خاصی میل کند. اگر داده‌ها دسته‌بندی پذیر خطی نباشند، قانون دلتا به سمتی میل می‌کند تا بهترین تقریب را از تابع هدف داشته باشد.

---

[1] learning rate

[2] Gradient Descent

نکته‌ی کلیدی‌ای که در قانون دلتا به کار رفته این است که این قانون از شیب نزول برای جستجوی فضای فرضیه‌ای بردارهای وزن را برای پیدا کردن متناسب‌ترین بردار استفاده می‌کند. اهمیت قانون دلتا از این رو است که پایه‌ای برای الگوریتم Backpropagation است. الگوریتم Backpropagation برای آموزش شبکه‌هایی با تعداد زیادی واحد به کار می‌رود. از سوی دیگر، شیب نزول پایه‌ای برای الگوریتم‌هایی که جستجو در فضای پیوسته‌ی فرضیه‌ای انجام می‌دهند است.

قانون دلتا، در پرسپترون‌های بدون مقدار آستانه قابل‌درک‌تر است. در چنین پرسپترون‌هایی داریم :

$$o(\vec{\mathrm{x}}) = \vec{w}.\vec{\mathrm{x}}$$

بنابراین، واحدی خطی (بدون مقدار آستانه) متناسب با هر پرسپترون مشخص می‌شود. برای اشتقاق یک وزن برای واحد خطی، از تعریف میزان خطای فرضیه (بردارهای وزن) شروع می‌کنیم. با وجود اینکه توابع بسیاری برای به دست آوردن خطا وجود دارد اما تعریف می‌کنیم که:

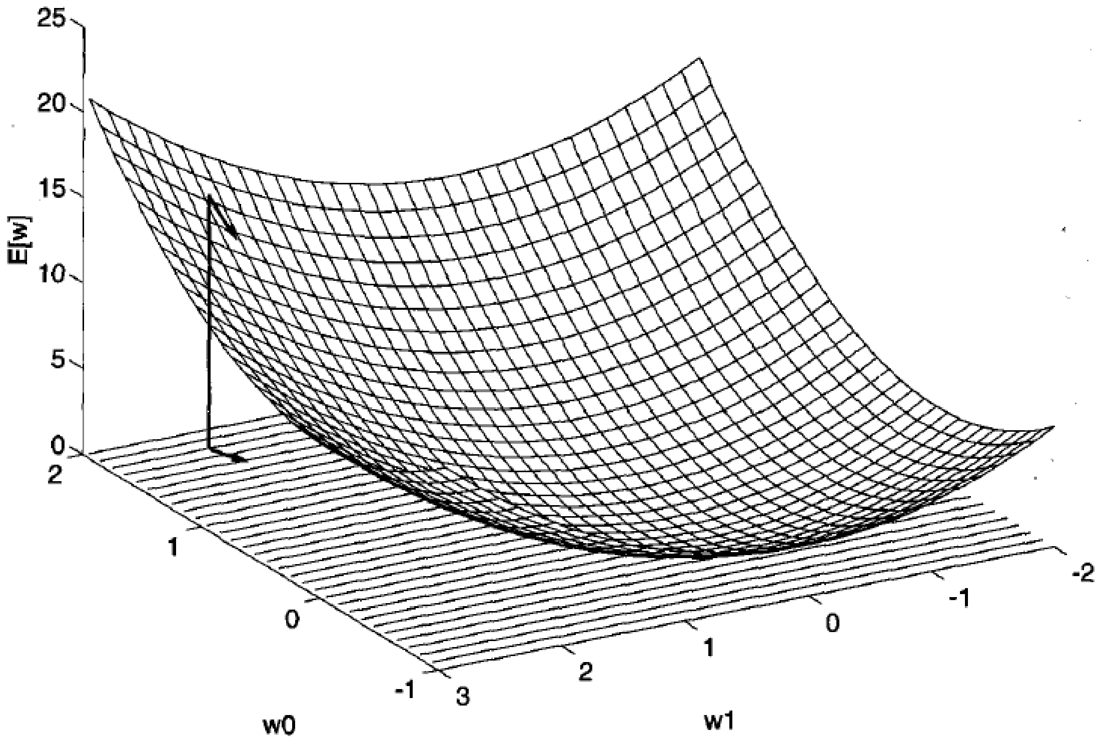
$$E(\vec{w}) \equiv \frac{1}{2}\sum\nolimits_{\mathrm{d}\in\mathrm{D}}(\mathrm{t_d} - \mathrm{o_d})^2$$

در این رابطه D دسته نمونه‌ها، $\mathrm{t_d}$ مقدار تابع هدف برای نمونه d، و $\mathrm{o_d}$ مقدار خروجی پرسپترون برای نمونه‌ی d است. طبق این تعریف، $E(\vec{w})$ نصف مجموع مجذور اختلاف‌های بین تابع هدف $\mathrm{t_d}$ و خروجی پرسپترون خطی $\mathrm{o_d}$ در تمامی نمونه‌های آموزشی است. در اینجا ما E را به عنوان تابعی از $\vec{w}$ تعریف کرده‌ایم زیرا که خروجی o به $\vec{w}$ وابسته است. البته E علاوه بر $\vec{w}$ به نمونه‌های آموزشی نیز وابسته است اما این نمونه ثابت فرض شده‌اند. در فصل ۶ توجیهی بیزی[1] برای نحوه‌ی تعریف E می‌آوریم. در کل، نشان خواهیم داد که در تحت شرایطی فرضیه‌ای که E را مینیمم کند متناسب‌ترین فرضیه‌ی درون H با داده‌های آموزشی است.

٤,٤,٣,١ تصور فضای فرضیه‌ها

برای درک الگوریتم شیب نزول، بد نیست فضای فرضیه‌ای و رابطه‌ی آن را با مقادیر E تصور کنیم (شکل ۴,۴). در شکل دو محور w_0 و w_1 دو مقدار ممکن برای بردار وزن واحد خطی هستند. محور سوم E میزان خطای مربوط به دسته‌ای از نمونه‌های آموزشی خاص را نشان می‌دهد. سطح خطای نشان داده شده در شکل ارجحیت هر بردار وزن را در فضای فرضیه‌ها نشان می‌دهد (بردارهایی ارجحیت دارند که خطای کمتری داشته باشند). با توجه به نحوه‌ی تعریف E، برای واحدهای خطی، سطح خطا همیشه سهمی‌وار است و یک نقطه‌ی مینیمم مطلق خواهد داشت. این نقطه‌ی مینیمم مطلق، همان‌طور که واضح است، به دسته نمونه‌های آموزشی وابسته است.

[1] Bayesian



شکل ۴٫۴ خطای فرضیه‌های مختلف.

برای واحدی خطی با دو وزن، فضای فرضیه‌ای H صفحه‌ی w_1 و w_0 خواهد بود. محور عمودی میزان خطای فرضیه‌ها را برای دسته نمونه ثـابتی نشـان می‌دهد. فلش‌های شکل شیب منفی را در نقطه‌ای خاص نشان می‌دهند، این فلش‌ها به سمتی اشاره می‌کنند که میزان خطا در آنجا به حداقل می‌رسد.

جستجوی شیب نزول بردار وزنی را مشخص می‌کند که در آن E کمینه است. در این الگوریتم ابتدا از برداری دلخواه شروع کـرده و مرحلـه بـه مرحله آن با تغییرهای کوچک به بردار وزن مطلوب میل می‌کند. در هر مرحله، بردار وزن به طرف بیشترین کاهش خطا حرکـت داده مـی‌شـود. این فرایند آن‌قدر ادامه پیدا خواهد کرد تا به مینیمم مطلق تابع خطا برسیم.

٤٫٤٫٣٫٢ اشتقاق قانون شیب نزول

چگونه می‌توان بیشترین کاهش خطا را پیدا کرد؟ این جهت با مشتق گرفتن ضمنی از میزان خطای E بر حسب تمامی مؤلفه‌های بـردار $\vec{w}$ بـه دست می‌آید. این بردار گرادیان[۱] E نامیده می‌شود و به صورت $\nabla E(\vec{w})$ نشان داده می‌شود.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial \mathrm{w}_0}, \frac{\partial E}{\partial \mathrm{w}_1}, \dots, \frac{\partial E}{\partial \mathrm{w}_\mathrm{n}}\right] \qquad (4.3)$$

توجه داشته باشید که خود $\nabla E(\vec{w})$ نیز یک بردار است که مؤلفه‌هایش مشتقات E بر حسب w_i هاست. زمانی کـه بـه گرادیـان بـه صـورت برداری در فضای وزن‌ها نگاه کنیم، سمت بیشترین افزایش E را مشخص خواهد کرد. در نقطه‌ی مقابل خلاف این سمت بیشـترین کـاهش E را

[۱] gradient

به دنبال خواهد داشت. برای مثال، در شکل ۴,۴ عکس گرادیان $(-\nabla E(\vec{w}))$ برای نقطه‌ای دلخواه در صفحه‌ی w_1 و w_0 نشان داده شـده است.

از آنجایی که گرادیان سمت بیشترین کاهش E را مشخص می‌کند، قانون یادگیری برای شیب نزول به شکل زیر خواهد بود:

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

که در آن

$$\Delta\vec{w} = -\eta\nabla E(\vec{w}) \tag{4.4}$$

در اینجا نیز η مقداری مثبت است که ضریب یادگیری نامیده می‌شود. این مقدار اندازه‌ی قدم‌های را در الگوریتم شیب نزول مشخص مـی‌کنـد. علامت منفی به خاطر این است که می‌خواهیم بردار وزن‌ها را به سمت کاهش میزان E حرکت دهیم. می‌توان به صورت ساده‌تر این قانون را بر روی مؤلفه‌های بردار وزن‌ها نیز نوشت:

$$w_i \leftarrow w_i + \Delta w_i$$

که در آن

$$\Delta w_i = -\eta\left(\frac{\partial E}{\partial \mathrm{w}_i}\right) \tag{4.5}$$

این نشان می‌دهد که برای رسیدن به بیشترین کاهش باید هر مؤلفه را متناسب با مقدار $\frac{\partial E}{\partial \mathrm{w}_i}$ تغییر داد.

برای تبدیل این فرایند به الگوریتم و تکرار مراحل توسط رابطه‌ی (۴,۵) لازم است که راهی مؤثر برای محاسبه‌ی گرادیان در هـر مرحلـه داشـته باشیم. خوشبختانه این کار چندان هم مشکل نیست. مشتقات سازنده‌ی بردار گرادیان $\frac{\partial E}{\partial \mathrm{w}_i}$ به سـادگی بـا اسـتفاده از رابطـه‌ی (۴,۲) محاسـبه می‌شود:

$$\frac{\partial E}{\partial \mathrm{w}_i} = \frac{\partial}{\partial \mathrm{w}_i}\frac{1}{2}\sum\nolimits_{d\in D}(\mathrm{t_d} - \mathrm{o_d})^2$$

$$= \frac{1}{2}\sum\nolimits_{d\in D}\frac{\partial}{\partial \mathrm{w}_i}(\mathrm{t_d} - \mathrm{o_d})^2$$

$$= \frac{1}{2}\sum\nolimits_{d\in D}2(\mathrm{t_d} - \mathrm{o_d})\frac{\partial}{\partial \mathrm{w}_i}(\mathrm{t_d} - \mathrm{o_d})$$

$$= \frac{1}{2}\sum\nolimits_{d\in D}2(\mathrm{t_d} - \mathrm{o_d})\frac{\partial}{\partial \mathrm{w}_i}(\mathrm{t_d} - \vec{\mathrm{w}}.\vec{\mathrm{x}}_\mathrm{d})$$

$$\frac{\partial E}{\partial \mathrm{w}_i} = \sum\nolimits_{d\in D}(\mathrm{t_d} - \mathrm{o_d})(-\mathrm{x_{id}}) \tag{4.6}$$

در این رابطه x_{id} نشان‌دهنده‌ی مؤلفه‌ی iام در نمونه‌ی d است. حالا ما معادلی برای $\frac{\partial E}{\partial w_i}$ داریم که به مقادیر x_{id}، o_d و t_d (مقدار تابع هدف نمونه آموزشی) وابسته است. با جایگزینی مقادیر رابطه‌ی (4.6) در رابطه‌ی (4.5) رابطه‌ی تغییر مقادیر وزن‌ها برای شیب نزول به دست می‌آید:

$$\Delta w_i = \eta \sum_{d\in D} (t_d - o_d)(x_{id}) \tag{4.7}$$

به طور خلاصه، الگوریتم شیب نزول برای آموزش واحدهای خطی به صورت زیر است: ابتدا برداری دلخواه برای وزن‌ها انتخاب کن. سپس برای هر نمونه آموزشی مقدار واحد خطی را محاسبه کن، و Δw_iها را برای هر وزن حساب کن (رابطه‌ی ۴,۷). هر وزن را با اضافه کردن Δw_i تغییر بده و این فرایند را تا اتمام نمونه‌های آموزشی تکرار کن. در جدول ۴,۱ این الگوریتم آورده شده است. چون سطح خطا فقط یک مینیمم مطلق دارد، این الگوریتم به برداری با کمترین خطا میل می‌کند، بدون توجه به اینکه داده‌ها دسته پذیر خطی هستند یا نه. فقط کافی است که η به اندازه‌ی کافی کوچک باشد. اگر η خیلی بزرگ باشد، احتمال دارد الگوریتم شیب نزول به کمترین مقدار خطا میل نکند. یکی از روش‌های حل این مشکل کم کردن تدریجی η در طول مراحل الگوریتم است.

٤,٤,٣,٣ تقریب اتفاقی شیب نزول

شیب نزول نمونه‌ی کلی مهمی از یادگیری است. این الگوریتم استراتژی‌ای برای جستجوی فضاهای بزرگ و نامتناهی فرضیه‌ای است. از این الگوریتم به شرطی می‌توان استفاده کرد که (1) فضای فرضیه‌ای پیوسته (برای مثال، فضای وزن‌ها در واحد خطی)، و (2) خطاها بر حسب پارامترهای این فرضیه صریح باشد. مشکلات استفاده از شیب نزول این است که (1) همگرایی به یک مقدار مینیمم موضعی بعضی مواقع زیادی طول می‌کشد (مثلاً، صدها گام لازم است تا به مقدار خاصی همگرا شویم) و (2) اگر چند مینیمم موضعی وجود داشته باشد تضمینی نیست که الگوریتم به مینیمم مطلق میل کند.

الگوریتم Gradient-Descent(training_examples,η)

هر نمونه آموزشی به صورت $<\vec{x}, t>$ مشخص می‌شود، $\vec{x}$ نمونه و t مقدار نمونه است. η نیز نرخ یادگیری را تعیین می‌کند.

- w_i ها را با مقادیر دلخواه کوچکی مقداردهی اولیه کن.
- تا زمانی که به شرط پایانی نرسیده‌ای حلقه‌ی زیر را اجرا کن
  - هر Δw_i را صفر مقداردهی اولیه کن.
  - برای هر مثال $<\vec{x}, t>$ حلقه‌ی زیر را اجرا کن
    - $\vec{x}$ را به واحد خطی بده و خروجی o را دریافت کن
    - برای هر وزن w_i دستور زیر را انجام بده

$$\Delta w_i \leftarrow \Delta w_i + \eta(t-o)x_i \tag{T4.1}$$

  - برای هر وزن واحد خطی دستور زیر را انجام بده

$$w_i \leftarrow w_i + \Delta w_i \tag{T4.2}$$

جدول ۴,۱ الگوریتم Gradient-Descent برای آموزش یک واحد خطی.

برای تبدیل به تقریب اتفاقی برای شیب نزول رابطه‌ی (T4.2) حذف می‌شود و رابطه‌ی (T4.1) نیز با رابطه‌ی $\boldsymbol{w_i \leftarrow w_i + \eta(t-o)x_i}$ *جایگزین می‌شود.*

یکی از راه‌های حل این مشکلات ، استفاده از متد شیب نزولی افزایشی[۱] یا متد شیب نزولی تصادفی[۲] است. قانون شیب نزول تغییر وزن‌ها را بعد از جمع بستن همه‌ی نمونه‌ها انجام می‌دهد (رابطه‌ی 4.7). اما متد شیب نزول تصادفی سعی می‌کند تا با افزایش ذره‌ذره‌ی وزن‌ها روش جستجوی شیب نزول را تخمین بزند و سپس خطا را برای هر نمونه محاسبه کند. این قانون آموزش نظیر قانون آموزش بیان شده در معادله‌ی 4.7 است با این فرق که بعد از هر تکرار طبق رابطه‌ی زیر وزن‌ها را تغییر می‌دهیم

$$\Delta w_i = \eta(t-o)x_i \tag{4.10}$$

در این رابطه t، o و x_i به ترتیب مقدار تابع هدف، خروجی واحد و i امین ویژگی نمونه آموزشی مورد بحث هستند. برای تبدیل الگوریتم شیب نزول (جدول ۴,۱) به شیب نزول تصادفی، رابطه‌ی (T4.2) حذف و به جای رابطه‌ی (T4.1) رابطه‌ی $w_i \leftarrow w_i + \eta(t-o)x_i$ جایگزین می‌شود. برای بیان الگوریتم شیب نزول تصادفی به بیانی دیگر کافی است تابع خطایی به نام $E_d(\vec{w})$ را برای هر نمونه آموزشی d به شکل زیر تعریف کنیم:

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2 \tag{4.11}$$

که در آن t_d و o_d به ترتیب مقدار تابع هدف و خروجی واحد برای نمونه d هستند. الگوریتم شیب نزول تصادفی برای تمامی نمونه‌های d در D تکرار خواهد شد و در هر تکرار وزن‌ها را بر اساس گرادیان و با توجه به $E_d(\vec{w})$ تغییر می‌دهد. سری‌ای از این تغییر وزن‌ها معیار خوبی برای تخمین کاهش گرادیان با توجه به تابع خطای اصلی $E(\vec{w})$ است. با کم کردن مقدار η (اندازه‌ی قدم‌ها در شیب نزول) به اندازه‌ی کافی، شیب نزول تصادفی می‌تواند به اندازه‌ی دلخواه به خود شیب نزول نزدیک شود. تفاوت‌های اساسی بین شیب نزول و شیب نزول تصادفی در زیر آورده شده است:

- در شیب نزول، خطا برای تمامی نمونه‌ها قبل از تغییر در وزن‌ها جمع زده می‌شد اما در شیب نزول تصادفی محاسبه‌ی خطاها و تغییر وزن‌ها همزمان انجام می‌شود.
- جمع خطا برای چندین نمونه در شیب نزول نیاز به محاسبات بیشتری در هر تکرار حلقه دارد. در مقابل چون از گرادیان اصلی برای تغییرات استفاده می‌شود، در هر قدم (به نسبت شیب نزول تصادفی) بیشتر به مینیمم E نزدیک می‌شود.
- در مواقعی که $E(\vec{w})$ چندین مینیمم‌های موضعی دارد گاهی شیب نزول تصادفی می‌تواند از افتادن در چنین مینیمم‌هایی پرهیز کند زیرا که شیب نزول تصادفی برای کنترل سمت جستجو بجای $\nabla E(\vec{w})$ از $\nabla E_d(\vec{w})$ استفاده می کند.

هر دو الگوریتم شیب نزول و شیب نزول تصادفی به یک اندازه در کاربرد استفاده می‌شوند.

به قانون آموزش رابطه‌ی (4.10) را قانون دلتا، LMS[۳]، قانون Adaline، یا قانون Window-Hoff (همنام ارائه کننده) نیز می‌نامند. در فصل ۱ از LMS برای توصیف کاربردش برای یادگیری ارزیابی‌ای از بازی استفاده کردیم. توجه داشته باشید که قانون دلتا در رابطه‌ی 4.10

---

[۱] incremental gradient descent

[۲] stochastic gradient descent

[۳] least-mean-square

مشابه قانون آموزش پرسپترون‌ها در قسمت ۴,۴,۲ است. در واقع از نظر ظاهری این دو رابطه با هم یکی هستند، با این وجود قانون دلتا o مربوط به رابطه‌ی خروجی واحد خطی یعنی $o(\vec{x}) = \vec{w}.\vec{x}$ و قانون پرسپترون o مربوط به رابطه‌ی خروجی واحد آستانه یعنی $o(\vec{x}) = sgn(\vec{w}.\vec{x})$ است.

با وجود اینکه قانون دلتا برای واحدهای خطی بدون مقدار آستانه بررسی شد اما می‌توان این قانون را برای آموزش پرسپترون‌ها نیز استفاده کرد. فرض کنید که $o = \vec{w}.\vec{x}$ خروجی بدون مقدار آستانه واحد خطی باشد و $o' = sgn(\vec{w}.\vec{x})$ خروجی واحد آستانه دار یا همان پرسپترون باشد. حال اگر می‌خواهیم که پرسپترون را با توجه به مقادیر تابع هدف که ± 1 هستند آموزش دهیم، می‌توانیم همان مقادیر را با استفاده از قانون برای آموزش o به کار ببریم. واضح است که اگر واحد خطی بتواند تمامی نمونه‌ها را یاد بگیرد پرسپترون نیز به حالت نظیر می‌تواند تمامی نمونه‌ها را یاد بگیرد (زیرا که sgn(1)=1 و sgn(-1)=-1). حتی زمانی که نمی‌توان با استفاده از واحد خطی همه‌ی نمونه‌ها را به دقت یاد گرفت مقدار آستانه این کمبود دقت را جبران می‌کند و مقادیر خروجی را به ± 1 می‌رساند (فقط کافی است واحد خطی علامت خروجی را درست تعیین کرده باشد). توجه داشته باشید که تلاش فرایند برای کم کردن خطای واحد خطی o است و ممکن است الزاماً دسته وزن‌هایی را مشخص نمی‌کند که کمترین خطای دسته‌بندی o′ را داشته باشد.

٤,٤,٤ ملاحظات

در قسمت قبلی دو الگوریتم تکراری[۴] برای پیدا کردن وزن‌های پرسپترون ارائه کردیم. تفاوت این دو الگوریتم در اینجا است که قانون پرسپترون وزن‌ها را برای پرسپترون‌هایی با مقدار آستانه پیدا می‌کند اما قانون دلتا وزن‌ها را برای پرسپترون‌هایی بدون مقدار آستانه پیدا می‌کند.

تفاوت این دو الگوریتم بر ویژگی‌های همگرایی آن‌ها نیز تأثیر گذاشته است. قانون پرسپترون، با فرض اینکه داده‌ها دسته‌بندی پذیر خطی باشند، پس از تعداد محدودی تکرار به فرضیه‌ی درست می‌رسد. در حالی که قانون دلتا به طور مجانبی به فرضیه‌ی درست میل می‌کند، و ممکن است برای همگرایی تا بی‌نهایت طول بکشد. در عوض قانون دلتا بدون توجه نیاز به دسته‌بندی پذیر خطی بودن داده‌ها همگرا می‌شود. برای اطلاعات بیشتر در مورد همگرایی این دو روش به Hertz et al. (1991) مراجعه کنید.

الگوریتم سوم برای یادگیری بردار وزن‌ها برنامه‌نویسی خطی[۵] است. برنامه‌نویسی خطی متدی کارآمد و کلی برای حل نامساوی‌های خطی است. توجه دارید که هر نمونه آموزشی متناسب با یک نامساوی به فرم $\vec{w}.\vec{x} > 0$ یا $\vec{w}.\vec{x} \leq 0$ است و جواب نامعادله نیز همان بردار وزن‌هاست. متأسفانه این روش نیز فقط زمانی به جواب می‌رسد که داده‌ها دسته‌بندی پذیر خطی باشند، با این وجود (Duda and Hart 1973, p. 168) فرمولی زیرکانه برای مواقعی که داده‌ها دسته‌بندی پذیر خطی نیز نیستند پیشنهاد داده است. به هر حال روش برنامه‌نویسی خطی برای شبکه‌های چندلایه تعمیم ندارد. در مقابل، روش شیب نزول که قانون دلتا نیز با کمک آن ساخته شده، به راحتی برای شبکه‌های چندلایه تعمیم می‌یابد. در قسمت آینده این تعمیم را بررسی خواهیم کرد.

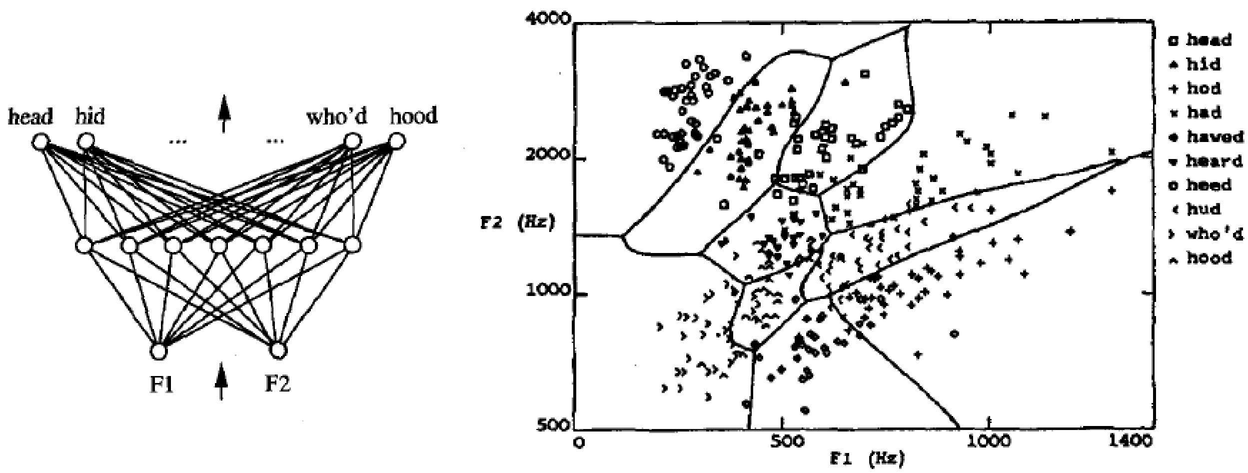
---

[۴] iterative

[۵] linear programming

٤,٥ شبکه‌های چندلایه و الگوریتم Backpropagation

همان‌طور که در قسمت ۴,۴,۱ نیز گفته شد، تک پرسپترون‌ها فقط سطوح خطی تصمیم‌گیری را می‌توانند یاد بگیرند. در مقابل، شبکه‌های چندلایه که توسط الگوریتم Backpropagation آموزش داده می‌شوند می‌توانند انواع مختلفی از سطوح تصمیم‌گیری غیرخطی را نیز یاد بگیرند. برای مثال، یک شبکه‌ی چندلایه و سطح تصمیم‌گیری آن در شکل ۴,۵ نشان داده شده است. در این مثال کار تشخیص گفتار برای تشخیص حرف صدادار بین دو حرف بی‌صدای h و d (در ده حالت مختلف) آورده شده. ورودی سیگنال صحبت به صورت دو پارامتر عددی که از آنالیز صدا به دست آمده است. این اعداد سطح ۲ بعدی تصمیم‌گیری را تشکیل می‌دهند. همان‌طور که در شکل نیز نشان داده شده شبکه‌های چندلایه می‌توانند سطوح تصمیم‌گیری خیلی پیچیده‌تری را نسبت به سطوح خطی (شکل ۴,۳) یاد بگیرند. در این بخش به نحوه‌ی آموزش شبکه‌های چندلایه توسط الگوریتم شیب نزول می‌پردازیم.



شکل ۴,۵ فضای تصمیم‌گیری یک شبکه‌ی چندلایه‌ی تک‌سویه.
شبکه‌ی نشان داده شده برای تشخیص یکی از ده صدای بین حروف h و d آموزش داده شده است. ورودی شبکه دو پارامتر F1 و F2 هستند که از آنالیز صدا به دست می‌آیند. ده خروجی شبکه متناسب با ده صدای مختلف هستند. پیش‌بینی شبکه صدایی است که بیشترین مقدار خروجی شبکه را داشته باشد. سمت راست سطح تصمیم‌گیری غیرخطی این شبکه را نشان می‌دهد. نقطه‌های نشان داده شده در شکل نمونه‌های آموزشی هستند. (گرفته شده از (Haung and Lippmann 1988)).

٤,٥,١ واحد آستانه‌ای مشتق‌پذیر

چه نوع واحدهایی برای تشکیل پایه‌های شبکه‌های چندلایه به کار می‌روند؟ در ابتدا ممکن است فکر کنیم که واحدهای خطی‌ای که پیش‌تر قانون یادگیری‌شان را پیدا کردیم مناسب‌اند. با این وجود، درحالی‌که شبکه‌هایی که ترکیب واحدهای خطی‌اند فقط توابع خطی را ایجاد می‌کنند، در حالی که هدف ما از شبکه‌های چندلایه پیدا کردن شبکه‌هایی است که توابع غیرخطی را بیان کنند. گزینه‌ی دیگر واحد پرسپترون است، اما ناپیوستگی مقدار آستانه‌ی این واحد آن را مشتق ناپذیر می‌کند. و واحدهایی که مشتق ناپذیرند، گرادیان ندارند و متعاقباً برای شیب نزول مناسب نیستند. در اینجا به واحدی با خروجی مشتق‌پذیر و غیرخطی نیاز داریم. واحد سیگموید یکی از راه‌حل‌های ممکن است. واحدی که خیلی مشابه پرسپترون و تابع مقدار آستانه‌اش پیوسته و مشتق‌پذیر است.

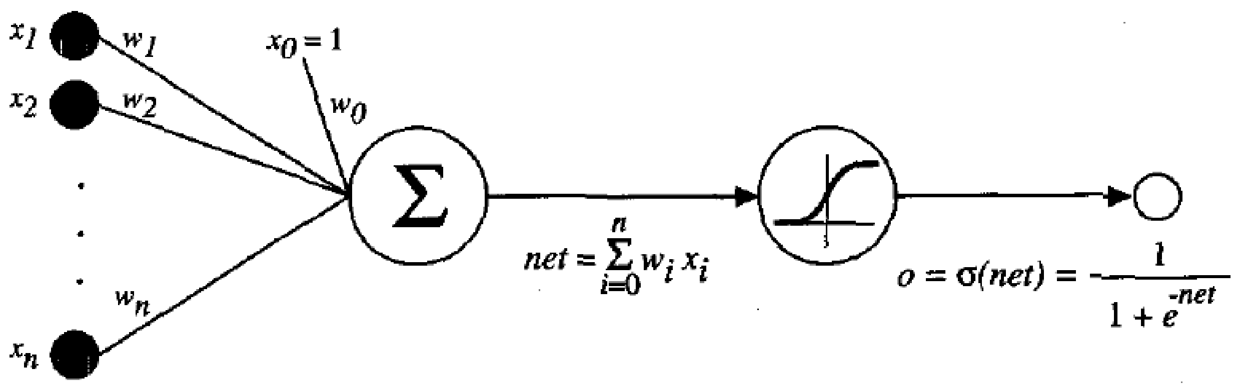
تابع سیگموید در شکل ۴٫۶ نشان داده شده. مثل واحد پرسپترون، سیگموید نیز ابتدا ترکیبی خطی از ورودی‌ها را محاسبه کرده و سپس آن را بعد از تأثیر تابع آستانه‌اش خروجی می‌دهد. در واحد سیگموید خروجی تابعی پیوسته از ورودی‌هاست. به عبارت دقیق‌تر خروجی واحد سیگموید از فرمول زیر محاسبه می‌شود:

$$o = \sigma(\vec{w}.\vec{\mathrm{x}})$$

که در آن

$$\sigma(y) = \frac{1}{1+e^{-y}} \qquad (4.12)$$

به تابع σ تابع سیگموید[6] یا تابع منطق[7] نیز می‌گویند. توجه دارید که خروجی این تابع عددی بین صفر تا یک است، که متناسب با ورودی‌هاست. (تابع سیگموید در شکل ۴٫۶ نشان داده شده). چون تابع سیگموید پهنای بزرگی از خروجی‌ها را به پهنای کوچکی می‌برد گاهی به آن تابع فشرده‌ساز[8] نیز می‌گویند. یکی دیگر از خواص بسیار مفید تابع سیگموید بیان مشتق آن بر حسب خودش است. [به عبارت دیگر، $\frac{\partial \sigma(\mathrm{y})}{\partial \mathrm{y}} = \sigma(\mathrm{y}).(1-\sigma(\mathrm{y}))$]. همان‌طور که بعداً نیز خواهیم دید، در استفاده از گرادیان این رابطه محاسبات را بسیار ساده‌تر می‌کند. توابع مشتق‌پذیر دیگر در بعضی موارد به جای σ به کار می‌روند. برای مثال گاهی به جای e^{-y} در تابع سیگموید $e^{-k.y}$ قرار می‌دهند که در آن k عددی ثابت و مثبت است که تندی مقدار آستانه را مشخص می‌کند. در بعضی موارد نیز از تابع tanh به جای سیگموید استفاده می‌شود (تمرین ۴٫۸).



شکل ۴٫۶ واحد آستانه‌ای سیگموید.

---

[6] sigmoid

[7] logistic function

[8] squashing function

٤,٥,٢ الگوریتم Backpropagation

الگوریتم Backpropagation وزن‌های لازم برای یک شبکه‌ی چندلایه با ساختار شبکه‌ی ثابت را پیدا می‌کند. این الگوریتم از شیب نزول برای مینیمم کردن میزان خطا، مربع اختلاف بین خروجی شبکه و تابع هدف، استفاده می‌کند. در این بخش الگوریتم Backpropagation و در بخش بعدی مشتقات لازم برای قانون شیب نزول وزن‌ها در این الگوریتم را ارائه می‌کنیم.

چون در این شبکه‌ها خروجی یک عدد نیست، پس کار را با تعریف دوباره‌ی E آغاز می‌کنیم و E را جمع خطای تمامی خروجی‌ها تعریف می‌کنیم:

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{\mathrm{k}\in\mathrm{outputs}}(\mathrm{t_{kd}}-\mathrm{o_{kd}})^2 \qquad (4.13)$$

که در آن outputs مجموعه‌ی تمامی خروجی‌های شبکه، $\mathrm{t_{kd}}$ مقدار تابع هدف و $\mathrm{o_{kd}}$ خروجی شبکه برای k امین خروجی و d امین نمونه است.

مسئله فعلی پیدا کردن وزن‌های متناسب با نمونه‌های آموزشی در میان فضای فرضیه‌ای تمامی وزن‌های ممکن است. می‌توان وضعیت را دوباره مثل شکل ۴,۴ تصور کرد. در وضعیت محور عمودی تعریف جدید E، و بقیه‌ی محورها، وزن‌های تمامی واحدهای شبکه هستند. درست مشابه زمانی که تنها یک واحد داشتیم در این وضعیت نیز می‌توان از شیب نزول برای یافتن فرضیه‌ای با کمترین میزان خطا کمک گرفت.

تنها فرق این است که برعکس حالت قبلی که فقط یک مینیمم داشت (شکل ۴,۴) در این حالت ممکن است چندین مینیمم موضعی وجود داشته باشد. متأسفانه شیب نزول تنها تضمین می‌کند که به سوی مینیممی موضعی میل کند، و این مقدار همیشه با مینیمم مطلق یکی نیست. با وجود این مانع، در عمل Backpropagation ثابت کرده که می‌تواند جواب‌های بسیار خوبی در کاربردهای واقعی پیدا کند.

الگوریتم Backpropagation در جدول ۴,۲ آورده شده است. این الگوریتم برای شبکه‌های یک‌سویه‌ای است که از دولایه واحد سیگموید تشکیل شده‌اند که هر واحد در هر لایه به تمامی واحدهای لایه‌ی قبلی متصل است. این الگوریتم یکی از دو نسخه‌ی شیب نزول تصادفی یا افزایشی الگوریتم Backpropagation است. نماد گزاری استفاده شده جز در موارد زیر مشابه قسمت‌های قبلی است:

- به هر گره[۱] یک اندیس[۲] نسبت داده شده است که در آن گره یک ورودی به شبکه یا خروجی واحدی است.
- x_{ij} نماینده‌ی ورودی گره‌ی i به واحد j است و w_{ij} وزن این متناظر است.
- δ_n نماد خطای مربوط به واحد n است. و نقش مقدار (t-o) را که قبلاً در قانون دلتا درباره‌ی آن بحث کردیم ایفا می‌کند. همان‌طور که بعداً نیز خواهیم دید: $\delta_n = \frac{\partial E}{\partial net_n}$.

Backpropagation ($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

هر نمونه آموزشی به صورت زوج مرتب $< \vec{x}, \vec{t} >$ مشخص می‌شود که در آن $\vec{x}$ بردار مقدارهای ورودی شبکه و $\vec{t}$ مقادیر تابع هدف است.

---

[۱] node
[۲] index

η ضریب یادگیری است، n_{in} تعداد ورودی‌های شبکه n_{out} تعداد خروجی‌های شبکه و n_{hidden} تعداد واحدها پنهان شبکه هستند.

ارتباط بین واحد i ام و واحد j ام به صورت x_{ij} نشان داده شده است و وزن متناسب با این ارتباط نیز با نماد w_{ij} نشان داده شده.

به هر گره[1] یک اندیس[2] نسبت داده شده است که در آن گره یک ورودی به شبکه یا خروجی واحدی است.

- شبکه‌ای یک‌طرفه با n_{in} واحد ورودی n_{hidden} واحد پنهان و n_{out} واحد خروجی بساز
- تمامی وزن‌های شبکه را با اعداد کوچک تصادفی مقداردهی اولیه کن (مثلاً بین ۰٫۵ و 0.5-)
- تا رسیدن به شرط پایانی حلقه‌ی زیر را اجرا کن
  - برای هر مثال $< \vec{x}, \vec{t} >$ در training_examples حلقه‌ی زیر را اجرا کن

ورودی را در جهت شبکه میان شبکه پخش کن:

۱. $\vec{x}$ را به ورودی بده و خروجی o_u را برای هر خروجی u دریافت کن

خطاها را خلاف جهت شبکه در میان شبکه پخش کن:

۲. برای هر خروجی k مقدار δ_k را از رابطه‌ی زیر به دست آور

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

۳. برای هر واحد پنهان h مقدار زیر را حساب کن

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in output} w_{kh}\delta_k \tag{T4.4}$$

۴. هر وزن w_{ji} را از رابطه‌ی زیر تغییر بده

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

که در آن

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

جدول ۴٫۲ نسخه‌ی شیب نزول اتفاقی الگوریتم Backpropagation برای شبکه‌های تک‌سویه که دو لایه واحد سیگموید دارند.

توجه دارید که الگوریتم جدول ۴٫۲ با ساخت یک شبکه‌ی جدید با همان تعداد واحد پنهان و همان تعداد واحد خروجی و مقداردهی اولیه‌ی وزن‌های آن با اعداد تصادفی کوچک آغاز می‌گردد. با توجه به اینکه ساختار شبکه ثابت و معلوم است، حلقه‌ی اصلی الگوریتم فقط برای نمونه‌های آموزشی مختلف تکرار می‌شود و بقیه‌ی موارد تغییری نمی‌کنند. برای هر نمونه‌ی آموزشی، نمونه به شبکه داده شده و خروجی را دریافت می‌شود، سپس خطای خروجی را برای نمونه مذکور محاسبه می‌کند. در ادامه، گرادیان را با توجه به خطای محاسبه شده محاسبه و در آخر نیز مقدار وزن‌ها را تغییر می‌دهد. این مرحله‌ی شیب نزول تا زمانی که خطای شبکه به حد مطلوب برسد تکرار می‌شود (گاهی این تکرارها تا صدها بار ادامه می‌یابد و همان نمونه‌ها چندین دفعه تکرار می‌شوند).

[1] node
[2] index

قانون تغییر وزن‌های شیب نزول (رابطه‌ی [T4.5] در جدول ۴,۲) مشابه رابطه‌ی قانون دلتا (رابطه‌ی [4.10]) است. مثل قانون دلتا، این رابطه مقدار هر وزن را به نسبت ضریب یادگیری η و مقدار ورودی x_{ji} که وزن به آن اعمال شده و مقدار خطای خروجی تغییر می‌دهد. تنها تفاوت بین این دو رابطه این است که خطا در قانون دلتا (t-o) بوده و در رابطه‌ی جدید با مقداری پیچیده‌تر δ_j جایگزین شده است. صورت دقیق δ_j از اشتقاق رابطه‌ی تغییر وزن‌ها در قسمت ۴,۵,۳ ناشی شده است. برای درک بهتر، ابتدا به فرمول محاسبه‌ی δ_k برای خروجی k ام شبکه دقت کنید (رابطه‌ی [T4.4]). با این وجود، از آنجایی که نمونه‌های آموزشی مقدار t_k را برای خروجی‌های شبکه دارند، پس مقدار تابع هدف برای واحدهای پنهان معلوم نیست و نمی‌توان خطا را به صورت مستقیم محاسبه کرد. پس به جای آن برای محاسبه‌ی خطای واحد پنهان h، از جمع خطاهای δ_k برای هر خروجی که h بر آن تأثیر دارد می‌شود. برای محاسبه‌ی درست‌تر لازم است که هر میزان خطا در میزان تأثیر h بر k w_{kh} ضرب شود. این کار باعث می‌شود که هر واحد پنهان به اندازه‌ای که در هر خطا "مسئول[1]" است در خطا سهم داشته باشد.

الگوریتم جدول ۴,۲ وزن‌ها را به صورت افزایشی و با برخورد به نمونه‌های مختلف تغییر می‌دهد. این تقریب تصادفی از شیب نزول است. برای رسیدن به خود گرادیان E باید مقادیر $\delta_j x_{ji}$ را قبل از تغییر وزن‌ها برای تمامی نمونه‌های آموزشی جمع زد.

حلقه‌ی تغییر وزن‌ها در Backpropagation در کاربردهای واقعی ممکن است صدها بار تکرار شود. با داشتن تنوع شروط خروج[2] می‌توان تکرار این فرایند را به متوقف کند. مثلاً ممکن است می‌توان تعیین کرد که شرط پس از تعداد خاصی تکرار متوقف شود، یا زمانی که مقدار خطا به کمتر از مقدار آستانه‌ی خاصی رسید، یا میزان خطا برای دسته نمونه‌های جداگانه به کمتر از مقدار خاصی برسد. انتخاب شرط پایانی از اهمیت خاص برخوردار است زیرا که تعداد کم تکرار ممکن است به مینیمم نشدن خطا بینجامد و تکرار زیاد نیز باعث می‌شود که شبکه فقط نمونه‌های آموزشی را تشخیص دهد (مشکل overfit). درباره‌ی این مسئله بعداً در قسمت ۴,۶,۵ مفصلاً بحث خواهد شد.

٤,٥,٢,١ اضافه کردن تکانه

چون Backpropagation الگوریتم پرکاربردی است، نسخه‌های بسیاری از این الگوریتم پدید آمده است. شاید معروف‌ترین این نسخه‌ها، نسخه‌ای است که به جای رابطه‌ی تغییر وزن‌ها (رابطه‌ی (T4.5)) از رابطه‌ی بازگشتی استفاده می‌کند:

$$\Delta w_{ji}(n) = \eta\delta_j x_{ji} + \alpha w_{ji}(n-1)$$

در اینجا $\Delta w_{ji}(n)$ تغییر وزنی است که در حلقه‌ی n ام حلقه‌ی اصلی انجام می‌شود. و به α که 0≤α<1 تکانه[3] می‌گویند. توجه دارید که جمله‌ی اول این رابطه همان تغییر وزن در رابطه‌ی (T4.5) است. جمله اضافی، جمله‌ی دوم، جمله‌ی تکانه نامیده می‌شود. برای درک بهتر، فرض کنید که در الگوریتم شیب نزول مسیر طی شده توسط یک گوی طی می‌شد، در آنجا این گوی هیچ تکانه‌ای نداشت. اثر α اضافه کردن تکانه به گوی مورد بحث است، و باعث می‌شود در هر حلقه ما تمایل داشته باشیم به سمتی حرکت کنیم که در حلقه‌ی قبلی به آن سمت حرکت کرده‌ایم. این اثر باعث به دام نیفتادن در مینیمم‌های نسبی‌ای که خطا خیلی در آن کم نمی‌شود و حرکت به سمت مینیمم مطلق خواهد شد. همچنین در جایی که سطح افقی می‌شود گوی بدون تکانه از حرکت باز می‌ایستد در حالی که گویی که تکانه دارد چنین مشکلی ندارد. همچنین در جایی که شیب تغییر نمی‌کند، اندازه‌ی قدم‌ها را بیشتر می‌کند تا در حلقه‌های کمتری به مینیمم برسیم.

[1] responsible
[2] termination condition
[3] momentum

۴,۵,۲,۲ یادگیری در شبکه‌های بدون دور با ساختار دلخواه

تعریفی که در جدول ۴,۲ از Backpropagation آورده شد فقط برای شبکه‌های دو لایه بود، با این وجود به راحتی می‌توان ایـن تعریـف را برای تمامی شبکه‌های تک‌سویه تعمیم داد. در این تعمیم، تغییری در رابطه‌ی تغییر وزن‌ها (رابطه‌ی (T4.5)) به وجود نمی‌آید و فقط رابطـه‌ی محاسبه‌ی δ عوض می‌شود. در کل برای محاسبه‌ی δ_r برای واحد r از لایه‌ی m از مقـدار δ_s هـای لایـه‌ی بعـدی از فرمـول زیـر محاسـبه می‌شود:

$$\delta_r = o_r(1-o_r)\sum_{s\in layer\ m+1} w_{sr}\delta_s \tag{4.19}$$

توجه دارید که این رابطه معادل پله‌ی سوم در الگوریتم جدول ۴,۲ است، و این پله در الگوریتم باید برای هر لایه‌ی پنهان در شبکه تکرار شود.

در واقع چنین استراتژی‌ای را می‌توان برای تمام شبکه‌هایی که ساختاری مشابه گراف‌های بدون دور دارند به کار گرفت، و نیازی به لایه‌ای بودن ساختار گراف نیست. برای شبکه‌هایی که لایه‌ای نیستند رابطه‌ی محاسبه‌ی δ برای تمامی واحدهای میانی به فرم زیر خواهد بود:

$$\delta_r = o_r(1-o_r)\sum_{s\in Downstream(r)} w_{sr}\delta_s \tag{4.20}$$

در رابطه‌ی فوق Downstream(r) مجموعه‌ی تمامی واحدهایی است که به طور مستقیم از واحد r ورودی دریافت می‌کنند یـا بـه عبـارت دیگر تمامی واحدهایی که مستقیماً پایین r هستند. در قسمت بعدی برای محاسبات از این فرم استفاده می‌کنیم.

۴,۵,۳ اشتقاق قانون Backpropagation

در این بخش به مشتق رابطه‌ی تغییر وزن در قانون Backpropagation می‌پردازیم. می‌توانید در اولین خواندن این کتـاب ایـن قسـمت را نخوانید!

در اینجا ما به اشتقاق رابطه‌ی شیب نزول تصادفی استفاده شده در جدول ۴,۲ می‌پردازیم. با توجـه بـه رابطـه‌ی ۴,۱۱ داریـم کـه شـیب نـزول تصادفی به هر نمونه به طور جداگانه نگاه می‌کند و برای هر نمونه d خطای E_d را به طور مجزا کم می‌کند. به عبارت دیگر، بـرای هـر نمونـه آموزشی d هر وزن w_{ji} با اضافه کردن Δw_{ji} تغییر می کند:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \mathrm{w_{ji}}} \tag{4.21}$$

که در آن E_d طبق تعریف خطای نمونه d است که برای تمام خروجی‌های شبکه محاسبه و جمع زده شده است:

$$E_d(\vec{w}) \equiv \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

در این رابطه outputs مجموعه‌ی واحدهای خروجی در شبکه، t_k مقدار تابع هـدف بـرای خروجـی k ام و نمونـه آموزشـی d و o_k مقـدار خروجی واحد k ام در شبکه برای نمونه d است.

اشتقاق قانون شیب نزول تصادفی از نظر مفهومی آسان است اما نیاز به توجه به اندیس‌ها و متغیرها دارد. در اینجا از همـان نمـایش شـکل ۴٫۶ استفاده می‌کنیم با این تفاوت که اندیس **j** را برای نمایش **j** امین واحد شبکه اضافه می‌کنیم:

- x_{ji} = **i** امین ورودی به واحد **j**
- w_{ji} = وزن مربوطه‌ی **i** امین ورودی به واحد **j**
- $net_j = \sum_i w_{ji}x_{ji}$ (مجموع وزن‌دار ورودی های واحد **j**)
- o_j = خروجی محاسبه شده برای واحد **j**
- σ = تابع سیگموید
- outputs = مجموعه‌ی واحدهای خروجی در لایه‌ی آخر شبکه
- Downstream(j) = مجموعه‌ی تمامی واحدهایی که از خروجی واحد **j** (در ورودی) استفاده می‌کنند

حال مقدار عبارت $\frac{\partial E_d}{\partial w_{ji}}$ را برای قانون شیب نزول تصادفی که در رابطه‌ی ۴٫۲۱ آمده محاسبه می‌کنیم. برای شروع، توجه داشته باشید کـه وزن w_{ji} فقط از طریق net_j بر شبکه اثر بگذارد. پس با توجه به قاعده‌ی زنجیره‌ای مشتق داریم:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j}\ x_{ji} \qquad (4.22)$$

با توجه به رابطه‌ی ۴٫۲۲، فقط کافی است که $\frac{\partial E_d}{\partial net_j}$ را به طرز قابل‌قبولی بیان کنیم. دو حالت را در نظر می‌گیریم: حالتی کـه واحـد **j** واحـدی خروجی است و حالتی که واحد **j** واحدی داخلی است.

حالت اول: قانون آموزش برای واحدهای خروجی. همان‌طور که گفته شد w_{ji} فقط از طریق net_j می‌تواند بر بقیـه‌ی شـبکه تـأثیر بگذارد و net_j نیز فقط از طریق o_j می‌تواند بر شبکه تأثیر بگذارد. بنابراین با توجه به قاعده‌ی زنجیره ای در مشتق:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \qquad (4.23)$$

برای شروع، فقط جمله‌ی اول رابطه‌ی ۴٫۲۳ را محاسبه می‌کنیم:

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

مقدار مشتق $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ برای تمامی خروجی‌های k به جز **j** صفر است. پس خواهیم داشت:

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\left(t_j - o_j\right)^2$$

$$= \frac{1}{2}\ 2\ \left(t_j - o_j\right)\frac{\partial\left(t_j - o_j\right)}{\partial o_j}$$

$$= -\left(t_j - o_j\right) \tag{4.24}$$

حالا جمله‌ی دوم رابطه‌ی ۴,۲۳ را محاسبه می‌کنیم. از آنجایی که $o_j = \sigma(net_j)$، مشتق $\frac{\partial o_j}{\partial net_j}$ فقط مشتق تابع سیگموید بـه ازای net_j خواهد بود ($\sigma\left(net_j\right)\left(1 - \sigma\left(net_j\right)\right)$). بنابراین:

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma\left(net_j\right)}{\partial net_j}$$

$$= o_j\left(1 - o_j\right) \tag{4.25}$$

با توجه به روابط ۴,۲۴ ، ۴,۲۵ و ۴,۲۳ داریم:

$$\frac{\partial E_d}{\partial net_j} = -\left(t_j - o_j\right)o_j\left(1 - o_j\right) \tag{4.26}$$

با ترکیب این رابطه با روابط ۴,۲۱ و ۴,۲۲ قانون شیب نزول تصادفی برای واحدهای خروجی به دست می‌آید.

$$\Delta w_{ji} = -\eta\frac{\partial E_d}{\partial \mathrm{w_{ji}}} = \eta\left(t_j - o_j\right)o_j\left(1 - o_j\right)x_{ji} \tag{4.27}$$

توجه داشته باشید که این قانون تغییر وزن‌ها معادل رابطه‌های (T4.3) و (T4.5) در جدول ۴,۲ است. علاوه بر این حال معلوم شد کـه مقـدار δ_k در رابطه‌ی (T4.3) مساوی کمیت $-\frac{\partial E_d}{\partial net_k}$ است. در ادامه‌ی این قسمت از δ_i به جای $-\frac{\partial E_d}{\partial net_i}$ استفاده می کنیم.

حالت دوم: قانون آموزش برای واحدهای پنهان. در این حالت واحد j واحدی پنهان یا داخلی است، در مشـتق‌گیری از قـانون آمـوزش برای w_{ji} باید توجه داشت که به طور غیرمستقیم w_{ji} بر خروجی شبکه و متعاقباً خطای E_d تأثیر خواهد داشت. به همین دلیل، بد نیست کـه به تمامی واحدهایی که مستقیماً از j ورودی دریافت می‌کنند اسمی اطلاق کنیم. ایـن دسـته از واحـدها را بـا Downstream(j) نمادگـذاری می‌کنیم. توجه داشته باشید که net_j فقط از طریق Downstream(j) می‌تواند بر روی خروجی‌ها و متعاقباً E_d تأثیر بگذارد. بنابراین داریم:

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k\frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j \left(1 - o_j\right) \tag{4.28}$$

با بازنویسی‌ای رابطه و استفاده از δ_j به جای $-\frac{\partial E_d}{\partial net_j}$ داریم:

$$\delta_j = o_j\left(1 - o_j\right) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

و

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

که دقیقاً همان قانون کلی‌ای است که در رابطه‌ی (4.20) آمده. از این رابطه می‌توان برای آموزش تمامی واحدهای پنهان در شبکه‌های بدون دور دلخواه استفاده کرد. توجه داشته باشید که رابطه‌ی (T4.4) در جدول 4.2 حالت خاصی از همین قانون است که Downsteam(j)=outputs.

٤,٦ نکاتی در مورد الگوریتم Backpropagation

٤,٦,١ همگرایی و مینیمم نسبی

همان‌طور که در بالا نیز گفته شد، الگوریتم Backpropagation از شیب نزول برای جستجوی فضای وزن‌های ممکن استفاده می‌کند و در هر بار اجرای حلقه مقدار خطای E (اختلاف بین تابع هدف و خروجی) را کمتر می‌کند. چون سطح خطا برای شبکه‌های چندلایه ممکن است چندین مینیمم نسبی داشته باشد، این امکان وجود دارد که شیب نزول در یکی از این مینیمم‌های نسبی به دام بیفتد و هیچ تضمینی نیست که این مینیمم نسبی همان مینیمم مطلق برای E باشد.

بر خلاف این ضعف الگوریتم Backpropagation، این الگوریتم در عمل متد تقریبی بسیار مفیدی است. در بسیار از کاربردهای واقعی مشکل مینیمم‌های نصبی به اندازه‌ای که گفته شد شدید نیست. برای درک مستقیم، شبکه‌هایی با تعداد زیادی از وزن‌ها در فضای خطایی با بعد زیاد را در نظر بگیرید (به ازای هر وزن یک بعد اضافه می‌شود). زمانی که شیب نزول داخل یکی از این مینیمم‌های نسبی می‌افتد، مینیمم یکی از وزن‌هاست، برای دیگر وزن‌ها داخل مینیمم نسبی نخواهد بود. در واقع، زمانی که تعداد وزن‌ها در شبکه افزایش می‌یابد متناسباً تعداد وزن‌ها و بعدها نیز افزایش یافته و امکان وجود راه فرار از مینیمم نسبی نیز افزایش می‌یابد.

مینیمم‌های نسبی جنبه‌ی دیگری نیز دارند و این تأثیر هنگامی که تعداد تکرارهای حلقه‌ی الگوریتم افزایش می‌یابد ظاهر می‌شود. توجه دارید که اگر برای مقداردهی اولیه وزن‌ها مقدار صفر را انتخاب کنیم، شیب نزول در قدم‌های اول اجرایش به تابعی بسیار هموار همگرا می‌شود که تقریباً خطی است. دلیل این امر این است که تابع سیگموید نیز در زمانی که وزن‌ها نزدیک به صفرند تقریبی خطی است (شکل تابع سیگموید در شکل ۴٫۶ آمده است). فقط هنگامی که وزن‌ها زمان کافی برای به اندازه‌ای کافی بزرگ شدن را داشته باشند می‌توانند به نقطه‌ای برسند که توابع شبکه غیرخطی را نیز تقلید کنند. می‌توانیم تصور کنیم که زمانی که تعداد مینیمم‌های نسبی زیادی در فضای وزن‌ها زیاد است شبکه توابع پیچیده‌تری را می‌تواند تقلید کند. و می‌توان امید داشت که زمانی که وزن‌ها به چنین نقاطی می‌رسند، به اندازه‌ی کافی به مینیمم مطلق نزدیک شده‌ایم.

بر خلاف آنچه در بالا گفته شد، شیب نزول برای سطوح خطای پیچیده‌تر قابل‌درک نیست و هیچ متدی وجود ندارد که با اطمینان مواردی که مینیمم مطلق مشکل‌ساز می‌شود را مشخص کند. ایده‌هایی که برای حل مشکل مینیمم نسبی ارائه شده به شرح زیر است:

- اضافه کردن جمله‌ی تکانه به رابطه‌ی تغییر وزن‌ها (استفاده از معادله‌ی 4.18). در بعضی موارد استفاده از این روش می‌تواند شیب نزول را از یک مینیمم موضعی به مینیمم مطلق ببرد (و در بعضی موارد نیز برعکس می‌تواند ما را از مینیمم مطلق به مینیمم موضعی بکشاند)
- استفاده از شیب نزول تصادفی به جای خود شیب نزول. همان‌طور که در بخش ۴٫۴٫۳٫۳ نیز گفته شده شیب نزول تصادفی تخمینی مفید از شیب نزول دارد که خطای دیگری را برای هر نمونه کم می‌کند، و با توجه به میانگین این خطاها و کل نمونه‌ها گرادیان را تخمین می‌زند. این سطوح مختلف خطا معمولاً مینیمم‌های نسبی مختلفی دارند و معمولاً الگوریتم در آن‌ها به دام نمی‌افتد.
- آموزش چندین شبکه با نمونه‌های آموزشی یکسان، مقادیر مختلف تصادفی وزن‌ها در ابتدای هر آموزش. اگر این چند آموزش مختلف به چند مینیمم موضعی مختلف در خطاها برسد، آنگاه می‌توان شبکه‌ای را انتخاب کرد که مینیمم موضعی کمتری دارد. متناوباً، می‌توان تمامی شبکه‌ها را دوباره آموزش داد و به عنوان "کمیته[1]" یا مجموعه‌ای از شبکه‌ها که خروجی آن‌ها متوسط خروجی شبکه‌های نظیر است استفاده کرد.

٤٫٦٫٢ معرفی قدرت شبکه‌های تک‌سویه

چه توابعی را می‌توان به شبکه‌های تک‌سویه آموزش داد؟ البته جواب این سؤال به عمق[2] و پهنای[3] شبکه وابسته است. با وجود اینکه هنوز هیچ اطلاعاتی کامل در مورد اینکه چه دسته توابعی را می‌توان به چه شبکه‌هایی آموزش داد در دسترس نیست، اما سه دسته تابع بخصوص را می‌توان به این نوع شبکه‌ها آموزش داد:

- توابع منطقی. هر تابع منطقی را می‌توان با شبکه‌های دو لایه یاد گرفت، اما با در بدترین حالت[4] این وجود تعداد گره‌های پنهان با افزایش ورودی‌های به صورت نمایی بالا می‌رود. برای معلوم شدن این توانایی، تابع منطقی دلخواهی را در نظر بگیرید، به ازای هر بردار بخصوص از ورودی‌ها واحد پنهانی را در نظر بگیرید که وزن‌هایش به شکلی هستند که تنها با آن ورودی بخصوص تهییج

---

[1] comettee

[2] depth

[3] width

[4] worse case

می‌شود. با چنین ساختار و آموزشی، شبکه‌ای به وجود می‌آید که در لایه‌ی پنهان آن همیشه یک واحد فعال است. حال با استفاده از واحدهای OR برای خروجی شبکه‌ای به سازید که به ازای مقادیر مختلف لایه‌ی پنهان خروجی متناسب را بدهد.

- توابع پیوسته. هر تابع کران‌دار پیوسته را می‌توان با مقداری خطای دلخواه (کمتر از حد دلخواه خاصی) با شبکه‌ی دو لایه یاد گرفت (Cybenko 1989;Hornic et al 1989). چنین شبکه‌هایی در لایه‌ی پنهان واحد سیگموید و در لایه‌ی خروجی واحد خطی (بدون مقدار آستانه) دارند. تعداد واحدهای پنهان لازم به تابع بستگی دارد.
- توابع دلخواه. هر تابع دلخواهی را می‌توان با دقت دلخواه توسط شبکه‌ای با ۳ لایه یاد گرفت (Cybenko 1988). باز هم واحدهای خروجی واحد خطی و واحدهای لایه‌های پنهان واحد سیگموید هستند. باز هم در حالت کلی معلوم نیست که هر لایه چند واحد نیاز دارد. اثبات این قضیه با استفاده از این است که نشان می‌دهند هر تابعی را می‌توان با ترکیب توابع خطی‌ای که فقط در محدوده‌ای غیر صفرند نشان داد. در ادامه‌ی اثبات ثابت می‌کنند که دو لایه واحد سیگموید کافی است تا تقریب خطی را برای هر محدوده‌ی کوچکی نشان دهند.

این نتایج به دست آمده نشان می‌دهد که شبکه‌های تک‌سویه با عمق محدود فضای فرضیه‌ای شاملی برای الگوریتم Backpropagation ایجاد می‌کنند. با این وجود بد نیست همیشه در نظر داشته باشیم که بردار وزن‌هایی که از طریق الگوریتم شیب نزول از مقداردهی اولیه به دست می‌آیند همیشه تمامی بردار وزن‌های ممکن را در بر ندارند. در کتاب (Hertz et al. 1991) درباره‌ی موارد بالا بیشتر بحث شده است.

٤,٦,٣ جستجو در فضای فرضیه‌ها و بایاس استقرایی

بد نیست که فضای فرضیه‌ای حاصل از جستجوی الگوریتم Backpropagation را با جستجوی دیگر الگوریتم‌ها مقایسه کنیم. در Backpropagation هر بردار وزن‌ها یک فرضیه را تشکیل می‌دهد که ممکن است توسط یادگیر یاد گرفته شود. به عبارت دیگر، فضای فرضیه‌ای فضایی n بعدی اقلیدسی است که بر پایه‌ی n بردار پایه ایجاد می‌شود. توجه دارید که این فضای فرضیه‌ای پیوسته است اما در مقابل فضای فرضیه‌ای درخت یادگیری و بقیه‌ی متدها گسسته هستند. با توجه به پیوستگی فضای فرضیه‌ای و اینکه E مشتق‌پذیر است (زیرا که متغیرهایش پیوسته‌اند)، پس خطای تعریف شده گرادیان دارد که همین گرادیان کمک بسیار بزرگی در سازمان‌دهی جستجو می‌کند. این ساختار سازمان‌دهی با ترتیب کلی‌تری (که در یادگیری مفهوم نمادین بود) و ترتیب ساده به پیچیده (که در درخت تصمیم‌گیری الگوریتم‌های ID3 و C4.5 بود) بسیار متفاوت است.

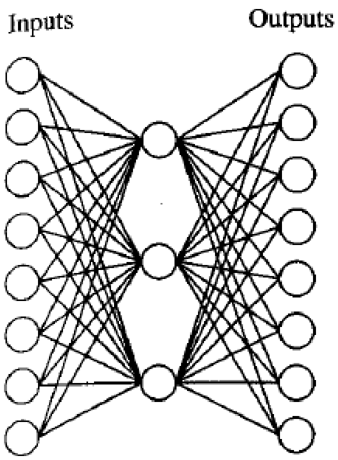
بایاس استقرایی‌ای که Backpropagation برای استقرا روی نمونه‌های آموزشی فرض می‌کند چیست؟ دقیقاً مشخص کردن بایاس استقرایی Backpropagation مشکل است زیرا که به اثر متقابل بین جستجوی شیب نزول و نحوه‌ای که فضای وزن‌ها فضای توابع قابل‌نمایش را پوشش می‌دهد بستگی دارد. با این وجود می‌توان این بایاس استقرایی را درون‌یابی بین نقاط داده‌ها دانست. مثلاً با داشتن دو نمونه مثبتی که هیچ نمونه منفی‌ای بین آن‌ها نیست، Backpropagation تمایل دارد که نقاط میانی این دو نقطه را نیز مثبت دسته‌بندی کند. چنین رفتاری را می‌توان در سطح تصمیم‌گیری‌ای که در شکل ۴,۵ آمده دید، در این شکل نمونه‌های آموزشی منطقه‌های تصمیم‌گیری را معلوم می‌کنند.

٤,٦,٤ معرفی لایه‌ی پنهان

یکی از ویژگی‌های خاص Backpropagation این است که در لایه‌ی پنهان در داخل شبکه مقادیر مفیدی را نمایش می‌دهد. زیرا که نمونه‌های آموزشی فقط مقادیر ورودی و خروجی را در خود دارند و فرایند تغییر وزن‌ها آزاد است که مقادیر لایه‌ی پنهان را به دلخواه تغییر دهد

تا خطا را مینیمم کند. همین آزادی باعث می‌شود در لایه‌های پنهان مقادیری را پیدا کند که صریحاً در نمونه‌ها بیان نشده اما ویژگی‌هایی را بیان می‌کنند که بیشترین تأثیر را در یادگیری تابع هدف دارند.

برای مثال شبکه‌ی شکل ۴,۷ را در نظر بگیرید، در این شبکه ۸ ورودی به ۳ واحد لایه‌ی پنهان وصل شده‌اند و این ۳ واحد نیز به ۸ واحد خروجی متصل هستند. به خاطر این ساختار، سه واحد لایه‌ی پنهان لازم است به صورتی مقادیر ۸ ورودی را با ویژگی‌های مرتبطی بیان کنند تا در انتها بتوانند همان مقادیر را به عنوان خروجی بدهند.



Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

شکل ۴,۷ مقادیر لایه‌ی پنهان برای نمونه‌های آموزشی.

این شبکه‌ی 8x3x8 با ۸ نمونه که در شکل است برای یادگیری تابع همانی آموزش داده شده است. بعد از ۵۰۰۰ بار اجرای حلقه مقادیر ۳ واحد پنهان مقادیر ورودی را به درستی کد می‌کنند. توجه داشته باشید که مقادیر کد شده را به صفر و یک گرد کنیم نتیجه کد باینری برای هشت ورودی خواهد بود.

شبکه‌ی شکل ۴,۷ برای یادگیری تابع هدف بسیار ساده‌ی $f(\vec{x}) = \vec{x}$ که در آن $\vec{x}$ برداری با هشت صفر و یک است در نظر بگیرید. شبکه باید یاد بگیرد تا ۸ ورودی را دوباره ایجاد کند. با اینکه این تابع هدف بسیار ساده است اما ۳ واحد پنهان برای یادگیری این تابع بسیار کم است. در این مثال شبکه مجبور است مهم‌ترین اطلاعات لازم را از طریق این سه واحد به سمت خروجی‌های انتقال دهد.

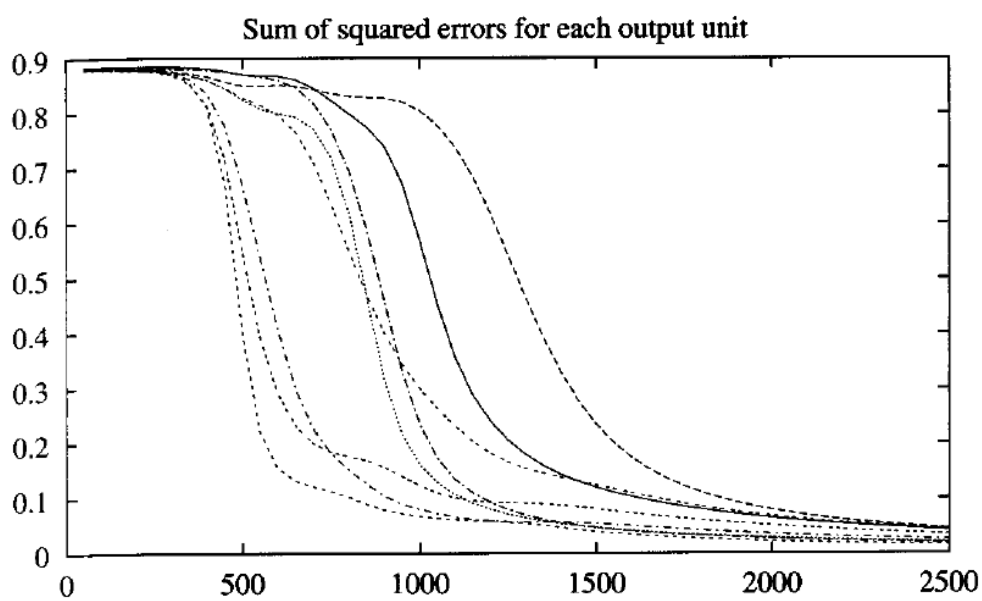
با استفاده از الگوریتم Backpropagation و نمونه‌های آموزشی مشخص شده در شکل این تابع هدف به شبکه یاد داده شده است. چه نمایشی از ورودی‌ها توسط شیب نزول در لایه‌ی پنهان نمایش داده می‌شود؟ با بررسی بیشتر مشخص می‌شود که این مقادیر که در لایه‌ی پنهان ظاهر می‌شود همان کد آشنای باینری برای هشت عدد است که با ۳ بیت نمایش داده می‌شود (۰۰۰، ۰۰۱، ۰۱۰، ...، ۱۱۱). مقدار دقیق این مقادیر در شکل ۴,۷ آورده شده است.

این قابلیت شبکه‌های عصبی در پیدا کردن نمایش‌های خاص در لایه‌های پنهان منحصر به فرد است. بر خلاف دیگر متدهای یادگیری که فقط مواردی را که طراح انسانی در نظر گرفته را در نظر می‌گیرند، این خاصیت به شبکه‌های عصبی این قابلیت را می‌دهد کاملاً انعطاف‌پذیر باشند و ویژگی‌هایی را استخراج کنند که طراح انسانی در نظر نگرفته است. البته بدیهی است که تمامی این ویژگی‌های استخراج شده باید از ورودی‌ها

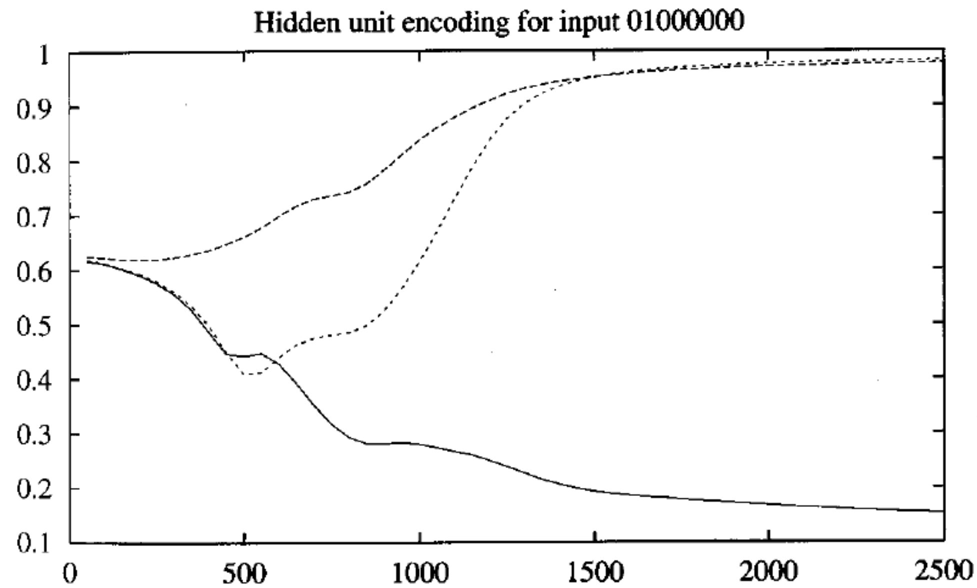
توسط واحدهای سیگموید قابل استخراج باشند. توجه دارید که زمانی که لایه‌های بیشتری در شبکه وجود دارند خواص پیچیده‌تری قابل‌استخراج‌اند. مثال دیگری از خواص لایه‌ی پنهان در قسمت ۴،۷، کاربرد در تشخیص چهره[۵]، آورده شده است.

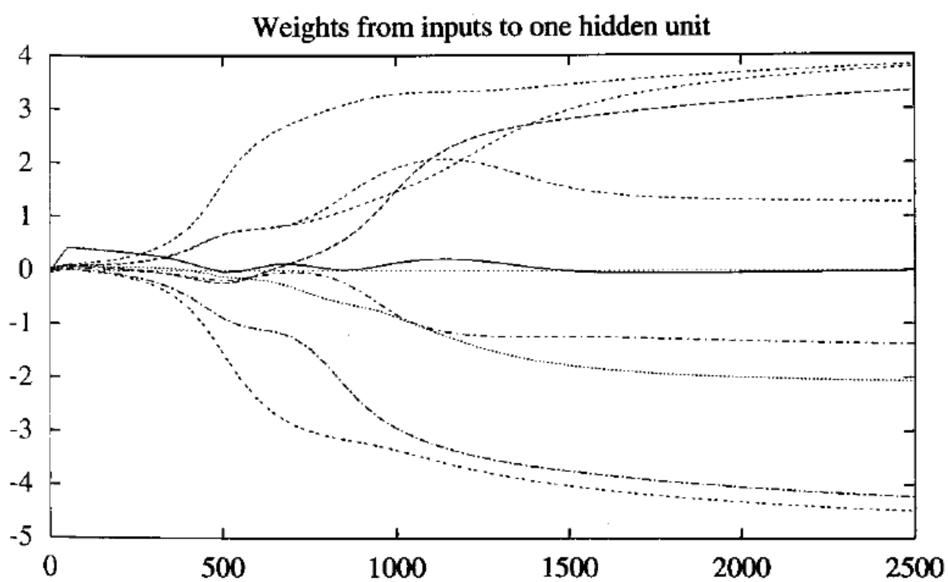
برای درک بهتر از عملیات Backpropagation بیایید در این مثال عملیات فرایند شیب نزول را دقیق‌تر بررسی کنیم (کد استفاده شده در این مثال در آدرس http://www.cs.cmu.edu/~tom/mlbook.html آمده است). شبکه‌ی شکل ۴،۷ با الگوریتم جدول ۴،۲ آموزش داده شده است، در مقداردهی اولیه‌ی وزن‌ها از اعداد بازه‌ی (-0.1,0.1) استفاده شده است و ضریب آموزشی نیز η=0.3 بوده و تکانه هم استفاده نشده است (α=0). یادگیری با استفاده از ضرایب آموزشی دیگر و تکانه نیز به همین نتایج رسیده است. مقادیر لایه‌ی پنهان (که در شکل ۴،۷) آمده بعد از ۵۰۰۰ بار اجرای حلقه اصلی الگوریتم (یعنی ۵۰۰۰ بار تکرار هر نمونه آموزشی) به دست آمده است. البته اکثر وزن‌ها در ۲۵۰۰ اجرای اول مشخص شده بودند.

با کشیدن نمودار خطا بر حسب تعداد گامی که شیب نزول برداشته، می‌توان تلاش شیب نزول را برای کاهش خطا دید. این نمودار در شکل ۴،۸ کشیده شده است. هر خط در این نمودار نشان‌دهنده‌ی مجموع خطاها برای تمامی نمونه‌های آموزشی در یکی از خروجی‌هاست. محور افقی تعداد تکرارهای حلقه‌ی اصلی الگوریتم را نشان می‌دهد. همان‌طور که نمودار نیز گویای مطلب است با ادامه‌ی کار شیب نزول مجموع خطای خروجی برای خروجی‌ها کاهش پیدا می‌کند، ممکن است این کاهش در بعضی خروجی‌ها شدیدتر و در بعضی دیگر ملایم‌تر باشد.



[۵] face recognition





شکل ۴٫۸ یادگیری شبکه‌ی 8x3x8.

نمودار اول مجموع خطاها را برای هر یک از ۸ خروجی را بر حسب تعداد تکرار حلقه‌ی اصلی الگوریتم نشان می‌دهد. نمودار دوم مقادیر لایه‌ی پنهان را برای ورودی ی "01000000" نشان می‌دهد. و نمودار آخر وزن‌ها را برای یکی از سه واحد پنهان نشان می‌دهد.

سیر تکامل لایه‌ی پنهان در نمودار دوم شکل ۴٫۸ دیده می‌شود. این نمودار مقدار سه واحد لایه‌ی پنهان را که در هر مرحله محاسبه می‌شود برای یکی از ورودی‌ها ("01000000") نشان می‌دهد. مثل نمودار اول محور افقی تعداد تکرارهای حلقه‌ی اصلی الگوریتم را نشان می‌دهد. همان‌طور که شکل نیز گویاست قبل از اینکه شبکه به نحوه‌ی کد سازی آخری برسد تعدادی از کد سازی‌های ممکن را برای ورودی امتحان کرده است.

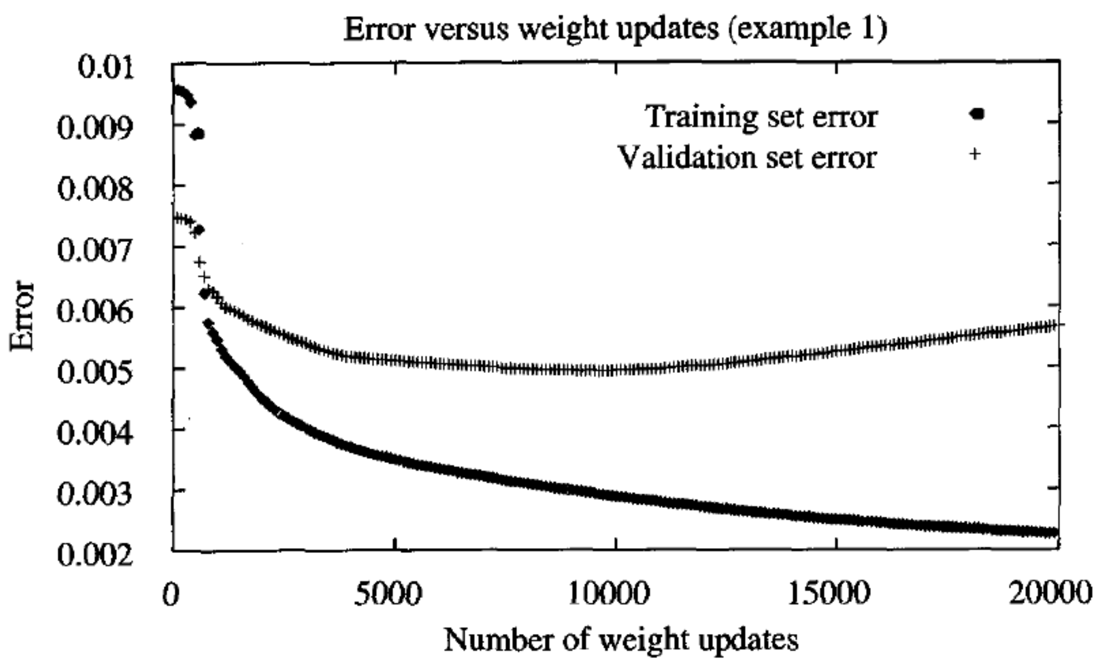
و بالاخره نمودار آخر شکل ۴٫۸ سیر تکامل وزن‌های شبکه را نشان می‌دهد. این نمودار تکامل وزن‌های ارتباط‌دهنده‌ی بین هشت واحد ورودی (و مقدار ثابت ۱ برای مقدار آستانه) و یکی از واحدهای لایه‌ی پنهان نشان می‌دهد. توجه داشته باشید که تغییرات قابل‌توجه در مقدار وزن‌ها

همزمان با تغییرات قابل‌توجه در میزان خطا و مقدار لایه‌ی پنهان است. وزنی که به مقداری نزدیک صفر میل می‌کند همان وزنی است که برای مقدار آستانه در نظر گرفته شده (w_0).

٤,٦,٥ معیارهای تعمیم، **overfit** و توقف

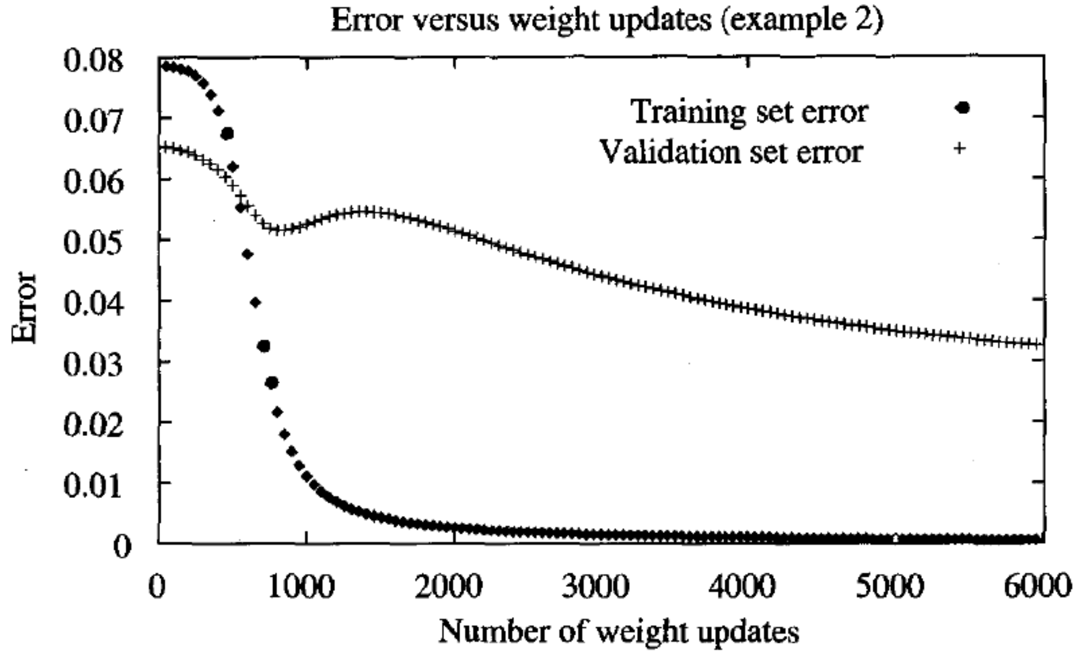
در جدول ۴,۲ از شرط پایانی[۶] صحبت شد اما معلوم نشد که این شرط دقیقاً چیست. شرط مناسب برای پایان حلقه‌ی تغییر وزن‌ها چیست؟ یک شرط بسیار ساده ادامه دادن مراحل تا زمانی که خطا (E) برای نمونه‌های آموزشی کمتر از مقدار خاصی بشود است. در واقع این استراتژی برای پایان بسیار ضعیف است زیرا که Backpropagation مستعد است تا برای کم کردن خطا قدرت تعمیم شبکه را برای نمونه‌های جدید کم کند.

برای روشن شدن خطر کم کردن افراطی خطا برای نمونه‌های آموزشی، توجه کنید که با افزایش تعداد تکرارهای حلقه‌ی اصلی الگوریتم خطای E چگونه کاهش می‌یابد. شکل ۴,۹ این تفاوت را برای استفاده از Backpropagation را برای دو مثال کاربردی نشان می‌دهد. نمودار اول را در نظر بگیرید. منحنی پایین‌تر در نمودار خطای E را برای نمونه‌های آموزشی نشان می‌دهد که با افزایش تعداد تکرارهای الگوریتم کاهش می‌یابد. منحنی بالایی خطای E را برای مجموعه‌ی تأیید نشان می‌دهد که از نمونه‌های آموزشی مجزاست. این منحنی قدرت تعمیم شبکه[۷] را برای نمونه‌های جدید نشان می‌دهد.



---

[۶] termination condition
[۷] generalization accuracy



شکل ۴،۹ نمودار خطای E به عنوان تابعی از تعداد تکرار تغییر وزن‌ها برای دو درک متفاوت یک ربات.
در هر دو حالت خطای E در طی تکرارهای بیشتر کمتر می‌شود. خطای مربوط به مجموعه‌ی تأیید[۸] معمولاً در ابتدا کاهش پیدا می‌کند و سپس بعد از افزایش تعدادهای تکرار بر اثر overfit احتمال دارد افزایش یابد. شبکه‌ای که کمترین خطا برای مجموعه‌ی تأیید را داشته باشد مناسب‌ترین شبکه برای تعمیم روی نمونه‌های جدید است. توجه داشته باشید که در نمودار دوم به محض افزایش جزئی مقدار خطای مجموعه‌ی تأیید نباید اجرای حلقه متوقف شود.

توجه داشته باشید که قدرت تعمیم شبکه که از مجموعه‌ی تأیید محاسبه می‌شود ابتدا کاهش می‌یابد و سپس افزایش می‌یابد، حتی اگر خطـای نمونه‌های آموزشی همچنان کاهش بیابد. دلیل این اتفاق چیست؟ دلیل این اتفاق این است که وزن‌هـا بعـد از آن طـوری تغییـر مـی‌کننـد کـه منحصراً با نمونه‌های آموزشی مطابق باشند و خاصیت تعمیمی خود را از دست می‌دهند. تعداد زیاد پارامترهای شبکه عصـبی درجـه آزادی‌هـای زیادی برای الگوریتم باقی می‌گذارد تا بتواند شبکه را منحصراً با نمونه‌های آموزشی مطابقت دهد (overfit).

چرا همیشه overfit در تکرارهای آخر الگوریتم اتفاق می‌افتد و در تکرارهای ابتدایی هیچ اثری از overfit نیست؟ فرض کنید که وزن‌هـا را با مقادیر کوچک تصادفی مقداردهی اولیه کرده‌ایم. چون وزن‌ها تقریباً یکی هستند، سطح تصمیم‌گیری بسیار هموار خواهد بود. در ادامه برای کم کردن میزان خطای نمونه‌های آموزشی بعضی از وزن‌ها افزایش می‌یابد و پیچیدگی بیشتری به سطح تصمیم‌گیری می‌دهند. در ادامه با افـزایش تعداد تکرارها بر پیچیدگی فرضیه‌هایی که به آن می‌رسیم بالا می‌رود، (پیچیدگی در این مراحل مفید است). با ادامه‌ی این تکرارها بـه انـدازه‌ی کافی می‌توان به سطوح تصمیم‌گیری پیچیده‌تری رسید که هم نویز نمونه‌های آموزشی را بر طرف می‌کند و هم ویژگی‌های غیر مربوطه به تابع هدف نمونه‌های آموزشی را بیان می‌کند. مشکل overfit در شبکه‌های عصبی درست مشابه همین مشکل در درخت تصمیم‌گیری است (فصل ۳).

تکنیک‌های بسیاری برای حل مشکل overfit در Backpropagation وجود دارد. یکی از این تکنیک‌ها weight decay نام دارد. در این متد در هر حلقه مقداری از هر وزن کم می‌شود. این درست مشابه این است که در تعریف E جمله‌ای را اضافه کنیم تا برخلاف آن عمل کند

[۸] validation set

و مانع overfit شود. به این جمله، جمله‌ی جریمه[۹] می‌گویند. هدف از این متد این است که مقدار وزن‌ها را کوچک نگه داریم تا بایاسی ایجاد کرده باشیم تا سطح تصمیم‌گیری بسیار پیچیده نشود.

یکی از موفق‌ترین متدها برای حل مشکل overfit، استفاده از دسته‌ی تأیید به همراه نمونه‌های آموزشی برای کنترل جستجوی شیب نزول است. الگوریتم می‌توان از خطای این مجموعه‌ی تأیید برای کنترل شیب نزول استفاده کند. از این نظر، این روش به الگوریتم اجازه می‌دهد که دو منحنی نشان داده شده در شکل ۴,۹ را در دسترس داشته باشد. اما حلقه‌ی تغییر وزن‌های الگوریتم چند بار باید اجرا شود؟ واضح است که حلقه باید به تعدادی اجرا شود که خطا روی دسته‌ی تأیید مینیمم شود. در کاربردهای معمول این روش دو نسخه از وزن‌های شبکه ذخیره می‌شود: یک کپی برای آموزش و کپی‌ای از بهترین وزن‌ها به دست آمده تا این مرحله بر اساس خطا بر روی مجموعه‌ی تأیید. زمانی که وزن‌های به دست آمده به مقدار خطای قابل‌توجهی بیشتر بر روی مجموعه‌ی تأیید می‌رسد، آموزش پایان می‌یابد و وزن‌های ذخیره شده به عنوان فرضیه‌ی نهایی خروجی داده می‌شود. زمانی که از این فرایند برای حالت شکل ۴,۹ استفاده شد بعد از ۹۴۰۰ بار تکرار الگوریتم به وزن‌های خروجی رسید. شکل دوم در شکل ۴,۹ نشان می‌دهد که همیشه رسیدن به کمترین مقدار خطای دسته‌ی تأیید را نمی‌توان به راحتی تعیین کرد. در این شکل ابتدا خطا بر روی دسته‌ی تأیید کاهش، سپس افزایش و دوباره کاهش می‌یابد. باید دقت کافی را در نتیجه‌گیری رسیدن خطای تأیید به مینیمم خود مبذول داشت، این شبکه در تکرار ۸۵۰ به مینیمم خطای تأیید می‌رسد.

در کل، مشکل overfit و چگونگی حل آن نیاز به دقت زیادی دارد. روش cross-validation بالا زمانی کارایی دارد که مقدار زیادی داده در دسترس باشد تا بتوان دسته‌ی تأیید تشکیل داد. با این وجود، مشکل overfit در مجموعه‌های آموزشی کوچک‌تر شدیدتر است. در چنین شرایطی، گاهی از روش k-fold cross-validation استفاده می‌شود، در این روش cross-validation، k بار متفاوت و هر بار با قسمت متفاوتی از داده‌ها به عنوان مجموعه‌ی آموزشی و دسته‌ی تأیید انجام می‌شود و نتیجه میانگین نتایج خواهد بود. در یکی از نسخه‌های این روش، m نمونه‌ی موجود به k زیرمجموعه‌ی مجزا با اندازه‌های m/k تقسیم می‌شوند. فرایند cross-validation، k بار و با استفاده از یکی از این k زیرمجموعه به عنوان دسته‌ی تأیید و بقیه‌ی نمونه‌ها به عنوان مجموعه‌ی آموزشی اجرا می‌شود. بنابراین، هر نمونه در یک آزمایش در دسته‌ی تأیید و در k-1 آزمایش در دسته‌ی آموزشی خواهد بود. در هر یک از آزمایش‌های روش cross-validation بالا برای تعیین تعداد تکرارها، i، مورد استفاده قرار می‌گیرد، i شماره‌ی تکراری است که در آن دسته‌ی تأیید بهترین خطا را داشته است. میانگین i برای این مقادیر محاسبه می‌شود، در انتها نیز از backpropagation برای آموزش شبکه بر روی تمامی n نمونه بدون دسته‌ی تأیید با تعداد تکرار میانگین i ها استفاده می‌شود. این فرایند بسیار مشابه فرایند مقایسه‌ی دو متد یادگیری با داده‌های محدود در فصل ۵ است.

٤,٧ یک مثال: تشخیص چهره[۱۰]

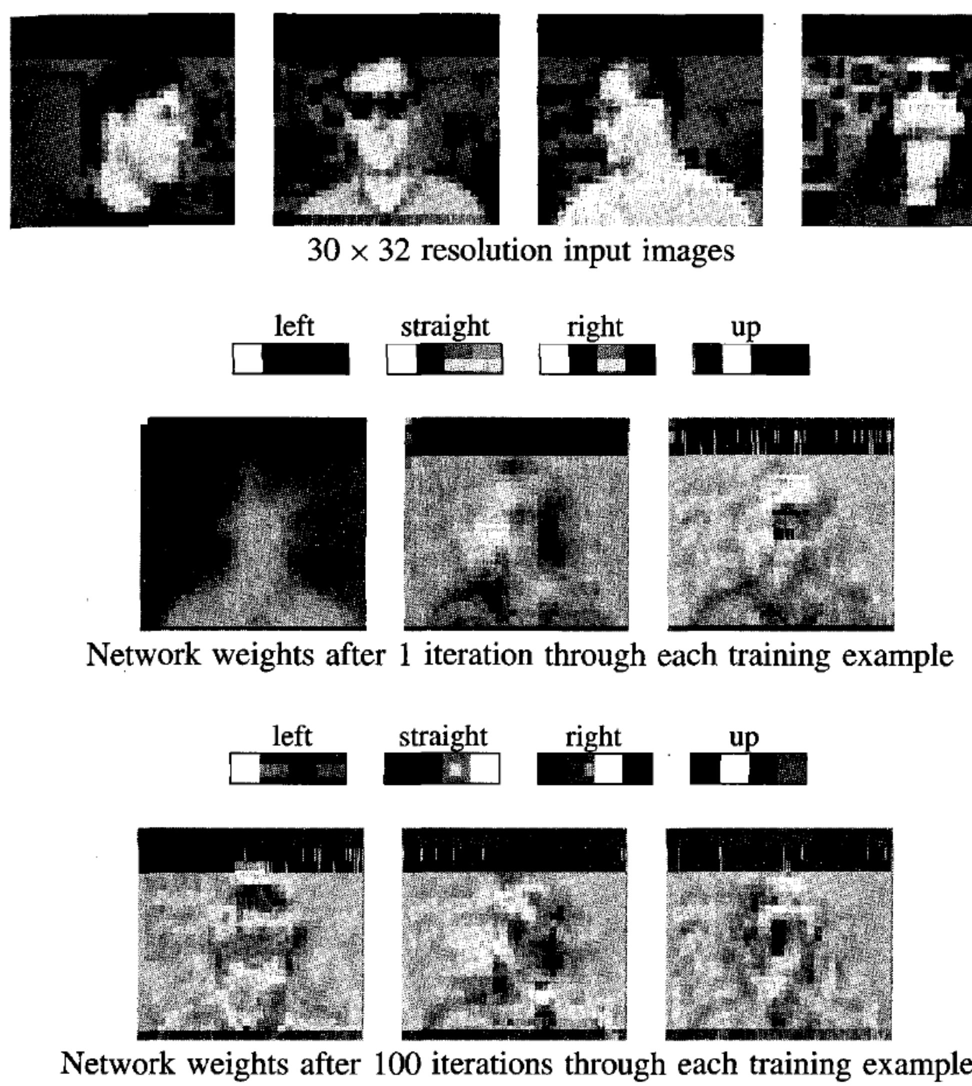
برای درک انتخاب‌های طراحی در نظر گرفته شده در اعمال الگوریتم Backpropagation، در این بخش استفاده از آن را در عمل یادگیری تشخیص چهره بررسی می‌کنیم. تمامی داده‌ها و اطلاعات مربوطه‌ی این قسمت در http://www.cs.cmu.edu/~tom/mlbook.html، به همراه نکات مربوطه‌ی استفاده از این کدها موجود است.

[۹] penalty term
[۱۰] face recognition

٤,٧,١ تعریف مسئله

مسئله‌ی یادگیری در اینجا شامل دسته‌بندی تصاویر دوربین از چهره‌ی افراد مختلف در زاویه‌های مختلف است. تصاویر ۲۰ نفر، به طور تقریبی ۳۲ تصویر از هر نفر با حالات مختلف چهره (شاد، غمگین، خشمگین، معمولی) و با زاویه‌های مختلف (چپ، راست، تمام‌رخ، بالا)، جمع‌آوری شده است. در شکل ۴,۱۰ نمونه‌ای از این تصاویر را می‌بینید، همچنین در این تصاویر مکان چهره‌ی شخص، پس‌زمینه، لباس افراد ثابت نیستند. در مجموع ۶۲۴ تصویر سیاه و سفید با دقت ۱۲۰×۱۲۸ با دقت رنگ ۰ (سیاه) تا ۲۵۵ (سفید) جمع‌آوری شد.

توابع هدف مختلفی را می‌توان از این مجموعه از داده‌های تصویری یاد گرفت. برای مثال، قرار دادن مجموعه تصاویر در ورودی شبکه می‌توان هدف را تشخیص هویت شخص، جهت صورت وی، جنسیت شخص، عینک دودی زدن وی و ... قرار داد. تمامی این توابع هدف را می‌توان با دقت بالا از این مجموعه داده یاد گرفت و شدیداً توصیه می‌شود که یادگیری این توابع هدف را خود خواننده امتحان کند. در ادامه‌ی این بخش ما به یک هدف یادگیری بخصوص می‌پردازیم: سمت صورت شخص (شامل چپ، راست، تمام‌رخ و بالا).



شکل ۴,۱۰ یادگیری یک شبکه‌ی عصبی مصنوعی برای تشخیص سمت صورت.

در اینجا شبکه‌ای با ابعاد ۴×۳×۹۶۰ بر روی لایه‌ی خاکستری تصویر چهره‌ی افراد آموزش داده می‌شود تا سمت چهره‌ی شخص را تشخیص دهد (چپ، راست، تمام‌رخ، بالا). بعد از یادگیری روی ۲۶۰ تصویر چهره، شبکه به دقت ۹۰٪ بر روی دسته‌ی تست مجزایی رسید. وزن‌های شبکه بعد از یک و صد بار تکرار حلقه‌ی یادگیری در بالا نشان داده شده‌اند. هر واحد خروجی (چپ، راست، تمام‌رخ، بالا) چهار وزن‌دارد که وزن‌هایشان با مربع‌های تیره (منفی) و روشن (مثبت) نشان داده شده است. چپ‌ترین مربع مربوط به وزن w_0 است که مقدار آستانه‌ی واحد را نشان داده و سه مربع دیگر وزن‌های ورودی واحد را از واحدهای پنهان نشان می‌دهد. وزن‌های مربوطه‌ی ورودی هر یک از سه واحد پنهان از نقاط نیز به طور نقطه‌ای در مکان خودشان نشان داده شده‌اند.

۴,۷,۲ انتخاب‌های طراحی

در اعمال الگوریتم Backpropagation به هر مسئله تعدادی انتخاب طراحی انجام می‌شود. این انتخاب‌های طراحی را برای مسئله‌ی مطرح یادگیری سمت چهره به صورت زیر خلاصه می‌کنیم. با وجود اینکه تلاشی برای طراحی بهینه در این مسئله انجام نمی‌شود، طراحی مطرح شده با دقت قابل‌توجهی تابع هدف را یاد می‌گیرد. بعد از یادگیری بر روی دسته‌ای ۲۶۰ عضوی از تصاویر، دقت دسته‌بندی بر روی دسته‌ی تست مجزا ۹۰٪ است. در حالی که، معمولاً دقت به دست آمده در حدس یکی از این چهار حالت ۲۵٪ است.

کدگذاری ورودی. با معلوم بودن اینکه ورودی شبکه باید نمایشی از تصویر باشد یکی از انتخاب‌های کلیدی تصمیم‌گیری نوع کدگذاری ورودی شبکه است. برای مثال می‌توانستیم تصویر را برای تشخیص لبه‌های رنگ‌ها پیش‌پردازش کرده یا دیگر خواص ناحیه‌ای تصویر را استخراج کرده و سپس به عنوان ورودی به شبکه بدهیم. یکی از مشکلات چنین ورودی‌هایی ایجاد تعداد متغیری از ویژگی‌ها (لبه‌ها) است در حالی که شبکه عصبی تعداد معلومی واحد ورودی دارد. انتخاب طراحی این قسمت استفاده از مجموعه نقاط ثابت ۳۰×۳۲ عکس و قرار دادن یک واحد ورودی برای هر نقطه است. بازه‌ی ۰ تا ۲۵۵ شدت رنگ نیز به طور خطی به بازه‌ی ۰ تا ۱ نگاشت شد تا مشابه لایه‌ی پنهان بازه‌ی خروجی واحد بین صفر تا یک باشد. تصویر ۳۰×۳۲ نقطه‌ای، در حقیقت، یک خلاصه‌ای از تصویر اصلی ۱۲۰×۱۲۸ نقطه‌ای موجود است که هر یک از نقاط با میانگین‌گیری چهار نقطه‌ی متناظر در تصویر با کیفیت بالاتر است. با استفاده از این کاهش کیفیت تعداد ورودی‌های شبکه به تعداد قابل‌کنترل‌تری تبدیل می‌شود، و بنابراین پیچیدگی محاسباتی نیز در عین حفظ دقت لازم برای دسته‌بندی درست تصاویر کاهش می‌یابد. با توجه به شکل ۴,۱ سیستم ALVINN نیز از کاهش دقت مشابهی برای ورودی شبکه استفاده می‌کند. یکی از نکات جالب این است که ALVINN بجای استفاده از میانگین نقاط محدوده‌ی مربوطه در تصویر با کیفیت بالاتر فقط نقطه‌ای تصادفی را در آن محدوده انتخاب کرده و شدت رنگ آن را به عنوان شدت رنگ مربوطه در نظر می‌گیرد. انگیزه‌ی این کار در ALVINN این است که محاسبات لازم برای ایجاد تصاویر کم‌دقت‌تر به طور قابل‌توجهی با این روش کاهش می‌یابد. این ویژگی هنگامی که شبکه لازم است تعداد زیادی از تصاویر را در ثانیه پردازش کرده و خودرو را کنترل کند بیشتر قابل‌توجه می‌شود.

کدگذاری خروجی. شبکه‌ی عصبی می‌بایست یکی از چهار ویژگی مربوطه‌ی جهت صورت شخص (چپ، راست، تمام‌رخ، بالا) را خروجی دهد. توجه داشته باشید که ما می‌توانستیم این چهار جهت را با یک خروجی و نسبت دادن ۰,۲، ۰,۴، ۰,۶، ۰,۸ برای چهار مقدار مربوطه این مقادیر خروجی را کد سازی کنیم. در مقابل، ما چهار واحد خروجی مجزا برای نمایش هر یک از جهات صورت در نظر گرفته‌ایم و واحدی که بیشترین مقدار را داشته باشد خروجی شبکه در نظر گرفته خواهد شد. این نوع کدگذاری گاهی کدگذاری ۱ از n [۱۱] نیز نامیده می‌شود. دو انگیزه برای انتخاب کدگذاری ۱ از n بجای در نظر گرفتن یک خروجی وجود دارد. ابتدا اینکه درجه‌ی آزادی بیشتری برای شبکه برای نمایش شبکه‌ی هدف باقی خواهد گذاشت (در لایه‌ی خروجی n برابر وزن موجود است). دوم اینکه در کدگذاری ۱ از n تفاوت بزرگ‌ترین مقدار خروجی و دومین (بزرگ‌ترین) مقدار خروجی را می‌توان به عنوان درجه‌ی اطمینان پیش‌بینی شبکه دانست (دسته‌بندی‌های مبهم ممکن است به

---

[۱۱] 1-of-n output encoding

خروجی‌های نزدیک و حتی مساوی بیانجامد). یکی دیگر از انتخاب‌های طراحی این است که مقدار خروجی این چهار واحد چه باید باشند؟ یکی از انتخاب‌های واضح استفاده از چهار مقدار <1,0,0,0> برای کد سازی جهت چپ، <0,1,0,0> برای جهت راست و ... است. در مقابل می‌توان بجای مقادیر ۰ و ۱ از مقادیر ۰,۱ و ۰,۹ استفاده کرد، <0.9,0.1,0.1,0.1> بردار مربوطه‌ی جهت چپ خواهد بود. یکی از دلایل پرهیز از ۰ و ۱ این است که واحدهای سیگموید نمی‌توانند این خروجی‌ها را با مقادیر محدود وزن‌ها ایجاد کنند. اگر سعی کنیم شبکه را برای مقادیر ۰ و ۱ آموزش دهیم شیب نزول مجبور به رشد بدون مرز وزن‌ها خواهد بود، در مقابل می‌توان با وزن‌های محدود به مقادیر ۰,۹ و ۰,۱ رسید.

ساختار گراف شبکه. همان‌طور که قبلاً هم توصیف شد، Backpropagation را می‌توان به هر گراف بدون دور واحدهای سیگموید اعمال کرد. بنابراین، گزینه‌ی طراحی دیگری نیز که با آن مواجهیم انتخاب تعداد واحدهای شبکه و چگونگی ارتباط بین آن‌هاست. متداول‌ترین ساختار شبکه ساختار لایه‌ای است که تمامی واحدهای یک لایه به تمامی واحدهای لایه‌ی بعد متصل‌اند. در طراحی فعلی از این ساختار متداول با دو لایه واحد سیگموید (یک لایه‌ی پنهان و یک لایه‌ی خروجی) استفاده کرده‌ایم. استفاده از یک یا دو لایه سیگموید متداول است و در مواقع خاص از سه لایه نیز استفاده می‌کنند. استفاده از تعداد لایه‌ی بیشتر متداول نیست زیرا که آموزش شبکه را بسیار کند می‌کند، همچنین شبکه‌ای با سه لایه سیگموید می‌تواند انواع بسیار زیادی از توابع را نمایش دهد (به قسمت ۴,۶,۲ مراجعه کنید). اگر بخواهیم از بین شبکه‌های تک‌سویه با یک لایه‌ی پنهان استفاده کنیم، مسئله اصلی تعین تعداد واحدهای پنهان شبکه خواهد بود. در مثال آورده شده در شکل ۴,۱۰ تنها از سه واحد پنهان استفاده شده است، و مجموعه خروجی دقت ۹۰٪ دارد. در آزمایش‌های دیگر، که از ۳۰ واحد پنهان استفاده شد دقت یک تا دو درصد بالاتر رفت. با وجود اینکه دقت کلی سازی برای این دو آزمایش تنها میزان کمی اختلاف دارد، اما آزمایش دوم به مقدار زمان آموزش قابل‌توجه بیشتری نیاز داشت. با مجموعه ۲۶۰ عکس آموزشی، بر روی یک سیستم خاص (sun sparc5) شبکه‌ی ۳۰ واحده زمان تقریبی یک ساعت را برای آموزش نیاز داشت در حالی که شبکه ۳ واحده تنها در حدود پنج دقیقه آموزش یافت. در بسیاری از کاربردها، تعداد واحدهای پنهان لازم برای یادگیری تابع هدف با دقت خاص ثابت است و واحدهای پنهان بیشتر بر روی دقت کلی سازی تأثیری ندارند، از روش‌های (cross-validation) برای تعیین تعداد تکرار لازم برای الگوریتم شیب نزول استفاده می‌شود. اگر از چنین روش‌های استفاده نگردد، در بعضی موارد استفاده از تعداد واحد پنهان بیشتر تمایل شبکه به (overfit) را افزایش داده و دقت کلی سازی را کم می‌کند.

دیگر پارامترهای الگوریتم یادگیری. در این آزمایش‌های یادگیری ضریب یادگیری η مقدار 0.3 و تکانه مقدار 0.3 را داشته است. مقادیر اولیه‌ی کم برای هر دو پارامتر دقت تعمیم نسبتاً مساوی‌ای را نتیجه خواهند داد اما زمان آموزش را افزایش خواهند داد. در مقابل اگر این پارامترها را بزرگ در نظر بگیریم، شبکه به شبکه‌ای با مقدار خطای قابل‌قبول روی دسته‌ی آموزشی میل نخواهد کرد. در تمامی این آزمایشات از شیب نزول تنها[۱۲] استفاده شده است (در مقابل تخمین احتمالی شیب نزولی که در جدول ۴,۲ معرفی شد). مقادیر وزن‌های خروجی به مقادیر کوچک نزدیک به صفر مقداردهی اولیه شده‌اند، اما مقادیر وزن‌های ورودی به صفر مقداردهی اولیه شده‌اند زیرا که با این روش تصور هوشمندانه‌تری از وزن‌های آموزشی می‌توان داشت (به شکل ۴,۱۰ رجوع کنید)، این انتخاب در تعمیم تأثیر نخواهد گذاشت. تعداد تکرار آموزش با تقسیم داده‌های موجود به دسته‌ی آموزشی و دسته‌ی تأیید مجزا تعیین شده است. از شیب نزول برای مینیمم کردن خطای مجموعه‌ی آموزشی استفاده شده و بعد از هر ۵۰ تکرار یک بار کارایی شبکه بر روی دسته‌ی تأیید بررسی شده است. شبکه‌ی انتخاب شده‌ی نهایی یکی از شبکه‌هایی است که بیشترین دقت را روی دسته‌ی تأیید داشته است. برای توجیه و توضیح این روش به قسمت ۴,۶,۵ رجوع کنید. دقت نهایی گزارش شده (۹۰٪ برای شبکه‌ی شکل ۴,۱۰) بر روی دسته‌ی سومی از نمونه‌های تست انجام شده است که هیچ دخالتی در آموزش نداشته‌اند.

[۱۲] full gradient decend

٤,٧,٣ نمایش پنهان یاد گرفته شده

بررسی مقادیر وزن‌های ۲۸۹۹ ارتباط شبکه از نظر تحلیلی جالب است. شکل ۴,۱۰ مقادیر وزن‌های نظیر این ارتباط‌ها را بعد از یـک و صـد بـار تکرار حلقه‌ی آموزش برای کل تصاویر آموزشی را نشان می‌دهد.

برای درک این نمودارها، ابتدا به مستطیل اول زیر تصویر توجه کنید. هر یک از مستطیل‌ها وزن‌های یکی از چهار واحد خروجی شبکه را نشـان می‌دهد (چپ، راست، تمام‌رخ، بالا). چهار مربع هر یک از این مستطیل‌ها چهار وزن هر واحـد خروجـی را نشـان مـی‌دهـد، وزن w_0 کـه مقـدار آستانه‌ی واحد را مشخص می‌کند در سمت چپ قرار دارد و سه وزن دیگر به ترتیب وزن‌های مربوطه‌ی واحدهای پنهان را نشان می‌دهند. شدت رنگ مربع‌ها نشان‌دهنده‌ی مقدار وزن نظیر است، سفید روشن وزن مثبت بزرگ و سیاه تیره نشان‌دهنده‌ی وزن منفی بزرگ است، خاکستری نیز مقادیر میانی وزن را نشان می‌دهد. برای مثال، خروجی واحد بالا مقدار وزن آستانه‌ی w_0 نزدیک به صفر، وزن مثبتی برای واحـد پنهـان اول و وزن بزرگ منفی‌ای برای واحد پنهان دوم دارد.

مقادیر وزن‌های شبکه بعد از ۱۰۰ بار تکرار شیب نزول برای تمامی نمونه‌های آموزشی در بالای شـکل نمـایش داده شـده‌اند. توجـه داریـد کـه چپ‌ترین واحد پنهان وزن‌های بسیار متفاوتی نسبت به وزن‌های پس از یک تکرار دارد، البته دو واحد دیگر نیز تغییر وزن داشته‌اند. درک نسـبی این کدگذاری در این مجموعه‌ی محدود از وزن‌ها خیلی هم سخت نیست. برای مثال، فرض کنید واحد خروجی‌ای که جهـت صـورت راسـت را مشخص می‌کند را در نظر بگیرید. این واحد وزن مثبت بزرگی از واحد پنهان دوم و وزن منفـی بزرگـی از واحـد پنهـان سـوم دارد. بـا بررسـی وزن‌های این دو واحد، می‌توان به آسانی فهمید که با چرخش صورت فرد به سمت راست (چپ مـا)، صـورت روشـن فـرد بـه طـور تقریبـی بـا وزن‌های مثبت قوی این واحد پنهان هم‌مکان خواهد شد و موی تیره‌ی فرد به طور تقریبی با وزن‌های منفی هم‌مکـان خواهـد شـد کـه باعـث بزرگ بودن خروجی این واحد خواهد شد. تصویر مشابهی می‌تواند باعث خروجی دادن واحد پنهان سوم نزدیک به صفر شود، زیـرا کـه صـورت روشن شخص با وزن‌های منفی بزرگ هم‌مکان خواهد شد.

٤,٨ مباحث پیشرفته‌ی شبکه‌های عصبی مصنوعی

٤,٨,١ گزینه‌های مختلف برای تابع خطا

همـان‌طور کـه قبلـاً نیـز گفتـه شـد، شـیب نـزول را مـی‌تـوان بـرای مینـیمم کـردن هـر تـابع مشـتق‌پذیر E بـه کـار بـرد. الگـوریتم اصـلی Backpropagation خطا را به فرم مجموع مربعات اختلافات با تابع هدف تعریف می‌کند، با این وجـود تعریف‌هـای دیگـری بـرای اعمـال شروط دیگر برای این خطا پیشنهاد می‌شود. برای هر تعریف E که به کار می‌بریم برای به دست آوردن گرادیان مشـتق بگیـریم. در زیـر توابـع خطای مرسوم آورده شده است:

- اضافه کردن جمله‌ی خطا برای اندازه‌ی وزن‌ها. همان‌طور که قبلاً نیز توضیح داده شد می‌توان با اضافه کردن جمله‌ی جدیـدی بـه E اضافه کرد که با افزایش بردار وزن‌ها افزایش یابد . این عمل باعث می‌شود تا جستجوی شیب نزول در بین بردارهـا، بردارهـایی کـه اندازه‌ی کوچک‌تری دارند را ارجح بداند و متناسباً (با در ناحیـه‌ی خطـی بـودن واحـدهای سـیگموید و کـم بـودن پیچیـدگی) خطـر overfit کم خواهد شد. پس یکی از روش‌های تعریف E به صورت زیر خواهد بود:

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2+\gamma\sum_{i,j}w_{ji}^2$$

پس در رابطه تغییر وزن‌ها در Backpropagation نیز باید همین تغییر را اضافه کرد، تمامی رابطه دست‌نخورده باقی خواهد ماند و فقط در هر تکرار در عدد ثابت (1-2γη) ضرب می‌شود. پس استفاده از این تعریف E معادل استفاده از استراتژی weight decay است (تمرین ۴,۱۰).

- اضافه کردن جمله‌ای برای خطاها در شیب یا مشتق تابع هدف. گاهی علاوه بر مقادیر تابع هدف، مقادیر مشتق تابع هدف نیز در نمونه‌های آموزشی وجود دارد. برای مثال، (simard et al. 1992) کاربردی را برای تشخیص کاراکتر معرفی می‌کند که در آن از مشتقات نمونه‌های آموزشی نیز استفاده شده است، در این کاربرد شبکه را مجبور می‌کند که نسبت به جابجایی حروف در صفحه بی‌تفاوت باشد. (Mitchell and Thrun 1993) نیز متدهایی را برای محاسبه‌ی مشتقات آموزشی بر اساس دانش قبلی ارائه می‌کنند. در هر دوی این سیستم‌ها (که در فصل ۱۲ آمده‌اند)، تابع خطا با اضافه شدن جمله‌ای که اختلاف مشتقات آموزشی و مشتقات واقعی شبکه است تغییر می‌یابد. مثالی از چنین تابع خطایی در زیر آمده،

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}\left[(t_{kd}-o_{kd})^2+\mu\sum_{j\in inputs}\left(\frac{\partial t_{kd}}{\partial x_d^j}-\frac{\partial o_{kd}}{\partial x_d^j}\right)^2\right]$$

در اینجا x_d^j نشان‌دهنده‌ی واحد ورودی j ام برای نمونه‌ی آموزشی d است. بنابراین، $\frac{\partial t_{kd}}{\partial x_d^j}$ مشتق آموزشی که چگونگی تغییر خروجی هدف t_{kd} را با تغییر x_d^j نشان می‌دهد. به صورت مشابه $\frac{\partial o_{kd}}{\partial x_d^j}$ نیز نشان‌دهنده‌ی مشتق واقعی شبکه یاد گرفته شده است. ثابت μ نیز وزن نسبی مقادیر نمونه‌های آموزشی و مشتقات آموزشی را مشخص می‌کند.

- مینیمم کردن cross entropy شبکه برای مقادیر تابع هدف. یادگیری یک تابع احتمالی، مثل پیش‌بینی اینکه درخواست‌کننده وامی، وام را کامل برگرداند را بر اساس ویژگی‌هایی چون سن درخواست‌کننده و موجودی حسابش را در نظر بگیرید. با وجود اینکه نمونه‌های آموزشی فقط مقادیر منطقی تابع هدف را در بر دارد. (۱ یا صفر، بسته به اینکه درخواست‌کننده وام را برگردانده یا خیر). بهترین نمایش تابع هدف، مدل کردن خروجی برای احتمال بازگرداندن وام است، به جای اینکه یاد بگیریم که خود مقادیر ۰ یا ۱ را برای هر نمونه واقعی یاد بگیریم. در چنین شرایطی که در آن هدف یادگیری تخمین احتمالات است، می‌توان نشان داد که بهترین تخمین زننده‌ی احتمال شبکه‌هایی هستند که مقدارcross entropy را با تعریف زیر مینیمم کنند،

$$-\sum_{d\in D}t_d log o_d+(1-t_d)\log(1-o_d)$$

در اینجا o_d تخمین احتمال خروجی شبکه برای نمونه‌ی آموزشی d است و t_d نیز مقدار هدف برای نمونه‌ی آموزشی d است. فصل ۶ شرایط و دلیل اینکه محتمل‌ترین فرضیه‌ی شبکه فرضیه‌ای است که cross entropy را مینیمم می‌کند را بررسی کرده و قانون شیب نزول را برای این معیار و واحدهای سیگموید پیدا خواهد کرد. همچنین در آنجا شرایط اینکه محتمل‌ترین فرضیه همان فرضیه‌ی است که مجموع خطاهای مربعی را مینیمم می‌کند را بررسی خواهیم کرد.

- با تغییر مؤثر تابع خطا می‌توان اشتراک وزن‌ها[۱۳] یا بستن به هم[۱۴] را برای واحدهای مختلف ورودی یا خروجی ایجاد کرد. ایده‌ی اصلی اجبار وزن‌های مختلف شبکه به داشتن مقادیر یکی است، معمولاً این کار توسط کاربرد به خاطر دانش قبلی در مورد مسئله صورت می‌گیرد. برای مثال (Waibel et al. 1989) و (Lang et al. 1990) کاربردی از شبکه‌های عصبی برای تشخیص صحبت را مطرح می‌کنند که در آن ورودی‌های شبکه عناصر فرکانسی صحبت در زمان‌های مختلف در پنجره‌ی زمانی ۱۴۴ میلی‌ثانیه است. یکی از فرض‌هایی که می‌توان انجام داد این است که عناصر فرکانسی که صدای مشخصی هستند (برای مثال صدای "eee") را باید مستقل از زمان دقیق آن در طول ۱۴۴ میلی‌ثانیه تشخیص داد. برای اعمال این قید، واحدهای مختلفی که ورودی از قسمت‌های مختلف پنجره‌ی زمانی دریافت می‌کنند باید وزن‌های مشترکی داشته باشند. اثر کلی این قید در فضای فرضیه‌های ممکن، کم کردن ریسک overfit و بهبود احتمال تعمیم به وضعیت‌های مشاهده نشده است. چنین اشتراک وزن‌هایی معمولاً با آموزش جداگانه‌ی وزن‌های مشترک و جایگزینی میانگینشان به جای آن‌ها صورت می‌گیرد. حاصل فرایند این است که اشتراک وزن‌ها به سمت مینیمم کردن تابع خطای دیگری میل می‌کنند که با تابع خطای اصلی یکسان نیست.

۴,۸,۲ دیگر متدهای مینیمم کردن خطا

شیب نزول یکی از اصلی‌ترین متدهای پیدا کردن فرضیه‌ای با مینیمم کردن تابع خطاست، اما این متد همیشه مؤثرترین متد نیست. بعضی مواقع در آموزش شبکه‌های پیچیده، backpropagation برای همگرا شدن به ده‌ها هزار تکرار حلقه‌ی تغییر وزن‌ها دارد. به همین دلیل، تعدادی الگوریتم بهینه‌سازی وزن‌ها ارائه شده و مورد مطالعه قرار گرفته است. برای مشاهده‌ی دیگر روش‌ها، بهتر است متد تغییر وزنی را با دو انتخاب در نظر بگیریم: انتخاب یک جهت برای تغییر بردار وزن‌ها و انتخاب طولی برای حرکت به آن سمت. در backpropagation این جهت با عکس گرادیان انتخاب می‌شود و طولی که حرکت می‌کنیم با ثابت ضریب یادگیری η معلوم می‌گردد.

یکی از متدهای بهینه‌سازی، که جستجوی خطی[۱۵] نامیده می‌شود روشی متفاوت برای انتخاب طول تغییر وزن ارائه می‌کند. در کل، زمانی که خطی برای جهت تغییر وزن‌ها انتخاب می‌شود، طول تغییر با پیدا کردن مینیمم تابع خطا بر روی این خط انتخاب می‌شود. توجه دارید که این کار ممکن است باعث تغییر بسیار بزرگ یا بسیار کوچکی، متناسب با فاصله‌ی مینیمم تابع خطا بر روی این خط، در وزن‌ها شود. روش دیگری که با ایده‌ی جستجوی خطی ایجاد شده، روش مکمل گرادیان[۱۶] نام دارد. در این جا، سری‌ای از جستجوهای خطی برای جستجوی مینیممی در سطح خطا انجام می‌شود. در مرحله‌ی اول این سری جستجو جهت عکس گرادیان انتخاب می‌شود. در هر مرحله جهتی جدید انتخاب شده که تغییر در آن جهت باعث تغییر در جهت‌های قبلی نگردد و عنصر خطای گرادیان که صفر شده صفر باقی خواهد ماند.

با وجود اینکه استفاده از متدهای دیگر گاهی در سرعت آموزش شبکه تأثیر دارند، اما متدهایی چون مکمل گرادیان تأثیر خاصی بر روی خطای تعمیم شبکه‌ی خروجی ندارند. تنها تأثیر بر روی خطای نهایی تفاوت بین فرایندهای مینیمم سازی در افتادن در مینیمم‌های نسبی متفاوت است. (Bishop 1996) بحث کاملی درباره‌ی متدهای بهینه‌سازی برای آموزش شبکه‌های عصبی انجام می‌دهد.

---

[۱۳] weight sharing
[۱۴] tying together
[۱۵] line search
[۱۶] conjugate gradient

۴,۸,۳ شبکه‌های دور دار

تا به الآن فقط به شبکه‌هایی پرداختیم که متناسب با گراف‌های بدون دور بودند. شبکه‌های دور دار[۱۷] شبکه‌های عصبی‌ای هستند که برای داده‌های سری‌های زمانی استفاده می‌شوند و خروجی شبکه در زمان t ورودی زمان t+1 خواهد بود. در چنین شرایطی، حلقه‌ای در شبکه موجود است. برای درک بهتر، فرض کنید که می‌خواهیم با استفاده از شاخص‌های سهام در هر روز x(t) متوسط قیمت سهام را برای روز بعدی y(t+1) پیش‌بینی کنیم. با داشتن سری‌ای از این اطلاعات، یکی از راه‌حل‌های بسیار ساده استفاده از شبکه‌ی تک‌سویه و استفاده از x(t) ها برای پیش‌بینی y(t+1)هاست. شبکه‌ای مثل آنچه در شکل ۴,۱۱ قسمت a آمده است.

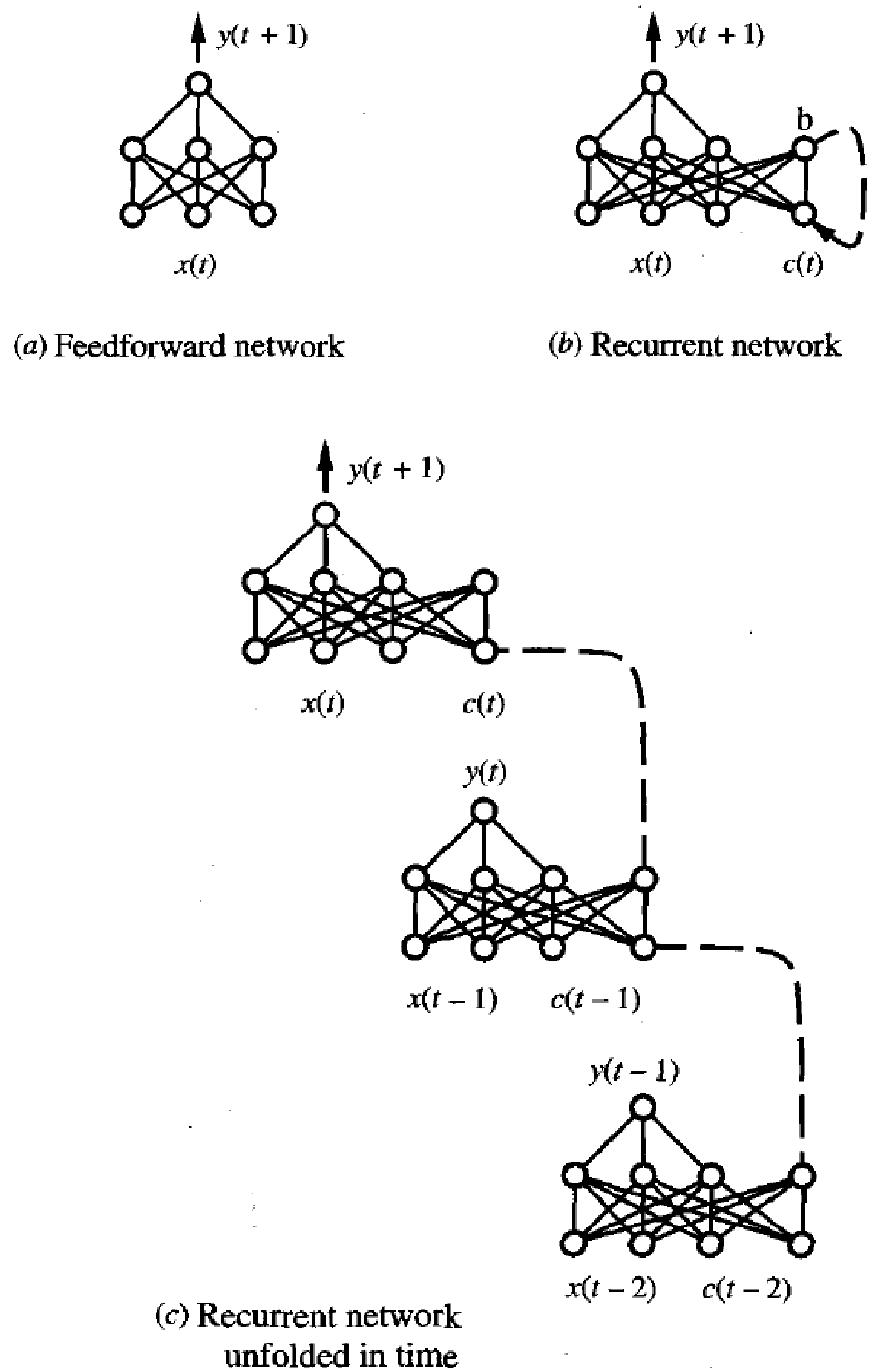
یکی از مشکلات این راه این است که مقدار y(t+1) فقط با توجه به x(t) پیش‌بینی می‌شود و هیچ تأثیری از مقادیر قبلی x نخواهد پذیرفت. در حالی که این تأثیر بسیار حیاتی است، برای مثال، فرض کنید که متوسط قیمت سهام روز بعد به میزان تغییر شاخصی بین امروز و دیروز وابسته است. با وجود اینکه این مشکل با اضافه کردن مقادیر x(t-1) حل می‌شود اما نمی‌توان از این راه‌حل برای اضافه کردن تمامی مقادیر گذشته‌ی x به سیستم استفاده کرد. شبکه‌ی دور داری که در شکل (b) 4.11 نشان داده شده راه‌حلی برای اساسی این مشکل است. در این شبکه ما واحد پنهان اضافی b و واحد ورودی جدید c(t) را به شبکه اضافه کرده‌ایم. مقدار c(t) تعریف شده تا همیشه مقدار واحد b را در زمان t-1 داشته باشد. پس مقدار ورودی c(t) در هر پله‌ی زمان دقیقاً همان مقدار واحد پنهان b در پله‌ی قبلی زمان است. چنین ساختاری باعث می‌شود تا شبکه رفتاری با توجه به گذشته انجام دهد، واحد b اطلاعات لازم را برای آینده ذخیره می‌کند و واحد c نیز از گذشته خبر می‌دهد. چون هر بار که b محاسبه می‌شود به عنوان ترکیبی از x(t) و c محاسبه می‌شود پس اطلاعات مربوط به داده‌های قدیمی‌تر را نیز در خود ذخیره کرده است. از ساختار شبکه‌های دور دار دیگری نیز می‌شد برای این مثال استفاده کرد. برای مثال، می‌توان چندین لایه بین ورودی و واحد b قرار داد و یا اینکه می‌توان از چندین حلقه به جای یک حلقه استفاده کرد.

چگونه می‌توان شبکه‌های دور دار را آموزش داد؟ انواع مختلفی از شبکه‌های دور دار وجود دارد و متدهای بسیاری برای آموزش آن‌ها پیشنهاد شده است. (برای اطلاعات بیشتر به Jordan 1986; Elman 1990; Mozer 1995; Williams and Zipser 1995 رجوع کنید). بسیار جالب است که بدانیم می‌توان با یک تغییر کوچک در Backpropagation شبکه‌هایی نظیر شکل (b) 4.11 را آموزش داد. برای درک این تغییر، شکل (c) 4.11 را در نظر بگیرید که ساختار تا نشده‌ی[۱۸] شبکه را نشان می‌دهد. در اینجا به جای حلقه‌هایی در طول زمان از کپی‌های بسیاری از همان شبکه استفاده کرده‌ایم. توجه دارید که این ساختار شبکه‌ی جدید هیچ دوری ندارد. بنابراین می‌توان شبکه‌ی تا نشده را با استفاده از Backpropagation آموزش داد. در واقع در عمل فقط یک کپی از شبکه آموزش داده می‌شود و یک دسته وزن خواهیم داشت. بنابراین بعد از آموزش شبکه‌ی تا نشده می‌توان مقدار هر وزن را میانگین وزن‌های نظیر در تمامی کپی‌ها دانست. (Mozer 1995) این فرایند را با جزئیات توضیح داده است. در کل، شبکه‌های دور دار سخت‌تر از شبکه‌های ساده آموزش داده می‌شوند و تعمیم‌های قابل‌اطمینانی نیز نمی‌دهند. با این وجود چون قابلیت تعمیم را افزایش می‌دهند همچنان جزو شبکه‌های مهم محسوب می‌شوند.

---

[۱۷] recurrent networks

[۱۸] unfolded



شکل ۴,۱۱ شبکه‌های دور دار.

۴,۸,۴ ساختار شبکه‌ی پویا

تا این لحظه به آموزش شبکه‌های عصبی به عنوان پیدا کردن وزن‌ها برای شبکه‌ای با ساختار گرافی ثابت پرداخته‌ایم. متدهای بسیاری درباره‌ی پویا[۱۹] بودن ساختار شبکه ارائه شده که در صورت نیاز، شبکه‌ها می‌توانند افزایش یا کاهش واحد و یا ارتباط (گره) داشته باشد تا قدرت تعمیم و مؤثر بودن آموزش را افزایش دهد.

یکی از این ایده‌ها شروع از شبکه بدون هیچ واحد پنهان، و افزایش واحدهای پنهان مطابق با نیاز، تا میزان خطا را تا حد قابل‌قبولی کاهش دهد. الگوریتم Cascade-Correlation (Fahlman, Lebiere 1990) چنین الگوریتمی است. این الگوریتم در ابتدا شبکه‌ای می‌سازد که هیچ واحد پنهانی ندارد. برای مثال، در مثال تشخیص چهره فقط ۴ واحد خروجی که هر کدام مستقیماً با تمامی گره‌های ورودی صفحه‌ی 30x32 در ارتباط‌اند ایجاد می‌کند. بعد از آموزش این شبکه، معلوم می‌شود که مقدار خطای قابل‌توجهی از باقی می‌ماند، زیرا که تابع هدف را نمی‌توان با شبکه‌ای تک لایه یاد گرفت. پس بنابراین الگوریتم یک واحد پنهان به شبکه اضافه می‌کند و وزن‌های مربوطه را چنان تعیین می‌کند که رابطه‌ی بین مقدار واحد پنهان و خطای باقی‌مانده‌ی شبکه حداکثر شوند. سپس این واحد جدید نصب می‌شود، تمامی وزن‌هایش ثابت نگه داشته می‌شود و ارتباطی بین آن و تمامی خروجی‌ها ایجاد می‌گردد. دوباره فرایند به حرکت می‌افتد، وزن‌های قدیمی دوباره آموزش داده می‌شوند (وزن‌های واحد پنهان همچنان ثابت می‌ماند). مقدار خطای باقیمانده دوباره چک می‌شود و اگر به اندازه‌ی قابل‌توجهی بزرگ بود واحدی دیگر به لایه‌ی پنهان اضافه خواهد شد. هر گاه که واحدی جدید به لایه‌ی پنهان اضافه می‌شود، از تمامی ورودی‌های اصلی و تمامی واحدهای پنهان قبلی ورودی دریافت می‌کند. به همین منوال شبکه گسترش پیدا خواهد کرد تا خطا از حد آستانه‌ای کمتر شود و به مقدار قابل‌قبولی برسد. (Fahlman and Lebiere 1990) با توجه به اینکه همیشه فقط واحد جدید آموزش داده می‌شود نشان دادند که در Cascade-Correlation می‌توان به طور قابل‌توجهی تعداد آموزش‌ها را کم کرد. یکی از مشکلات کاربردی‌ای که هنگام به کار بردن این الگوریتم پیش می‌آید این است که چون تعداد واحدهای اضافه شده نامحدود است پس خیلی ساده مشکل overfit رخ می‌دهد، پس همیشه باید اقدامات لازم را برای جلوگیری از overfit در استفاده از این الگوریتم انجام داد.

راه دیگر برای استفاده از ساختار شبکه‌ی پویا، دقیقاً عکس این Cascade-Correlation است. بجای شروع از ساده‌ترین شبکه‌ها و پیچیده‌تر کردن آن طی مراحل، با شبکه‌ای پیچیده شروع می‌کنیم و با معلوم شدن اینکه بعضی ارتباطاها مهم نیستند آن‌ها را هرس می‌کنیم. یکی از راه‌های تشخیص اینکه یک ارتباط مهم نیست این است که وزن ارتباطهای غیر مهم معمولاً نزدیک به صفرند. راه دوم، که در کاربرد موفق‌تر به نظر می‌رسد، این است که ببینیم تغییر کوچکی در مقدار وزن چه تأثیری بر خطای E می‌گذارد. اثر تغییر w بر E $(\frac{\partial E}{\partial w})$ را می‌توان معیاری برای اهمیت ارتباط نظیر دانست. (LeCun et al. 1990) فرایندی را معرفی می‌کند که طی آن شبکه آموزش داده می‌شود و کم‌اهمیت‌ترین ارتباطاها نیز حذف می‌شوند، این فرایند آن‌قدر تکرار می‌شود که به شرط پایانی خاصی برسیم. به این فرایند، روش بهینه‌سازی ی صدمات مغز[۲۰] نیز می‌گویند، زیرا که در هر مرحله، الگوریتم سعی می‌کند تا کم‌اهمیت‌ترین ارتباطاها را حذف کند. گفته می‌شود در شبکه‌های بسیار بزرگ برای تشخیص کاراکتر چنین روشی می‌تواند تعداد ارتباطاها را تا ۱/۴ کاهش دهد، و قدرت تعمیم شبکه را کمی بهبود می‌بخشد و بازده آموزش را به طور قابل‌توجهی بالا می‌برد.

---

[۱۹] dynamic

[۲۰] optimal brain damage

در کل، تکنیک‌های ساختار شبکه‌های پویا موفق آمیز بوده است. فقط باید دید تا آن‌ها به اندازه‌ی Backpropagation در افزایش قدرت تعمیم قوی هستند یا نه. با این وجود در مواردی نشان داده شده که به میزان قابل‌توجهی در زمان آموزش تأثیر می‌گذارند.

٤,٩ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر می‌شود:

- شبکه‌های عصبی مصنوعی متدی کاربردی برای یادگیری توابع حقیقی مقدار و برداری را بر روی ویژگی‌های گسسته و پیوسته ارائه می‌کنند، این متد در مقابل خطای داده‌های آموزشی مقاوم است. الگوریتم Backpropagation متداول‌ترین متد یادگیری شبکه است و در بسیاری از کارهای یادگیری نظیر تشخیص دستخط و کنترل ربات با موفقیت به کار رفته است.
- فضای فرضیه‌ای در نظر گرفته شده برای الگوریتم Backpropagation فضای تمامی توابع ممکنی که با تغییر وزن‌های شبکه ثابتی از واحدها و ارتباطاها بیان می‌شوند است. شبکه‌های تک‌سویه با تعداد کافی واحد در هر لایه که شامل سه لایه واحد می‌شوند قابلیت تخمین هر تابعی را با دقت دلخواه را دارند. حتی شبکه‌هایی با اندازه‌ی واقعی می‌توانند فضای غنی‌ای از توابع غیرخطی را نمایش دهند، به همین دلیل شبکه‌های تک‌سویه گزینه‌ی خوبی برای یادگیری توابع گسسته و پیوسته که در حالت کلی فرم کلی مجهولی دارند است.
- Backpropagation فضای تمامی فرضیه‌های ممکن را با استفاده از شیب نزول و کاهش متناوب خطای بین شبکه و نمونه‌های آموزشی جستجو می‌کند. شیب نزول به سمت مینیمم نسبی خطای بین شبکه و نمونه‌های آموزشی برای وزن‌های شبکه میل خواهد کرد. در حالت کلی‌تر، شیب نزول متدی بالقوه مفید برای جستجوی فضاهای فرضیه‌ای پیوسته‌ی چند متغیره که در آن خطای آموزشی تابعی مشتق‌پذیر از پارامترهای فرضیه است.
- یکی از فریبنده‌ترین ویژگی‌های Backpropagation قابلیت آن در ایجاد ویژگی‌های جدیدی است که به طور صریح در ورودی شبکه در نظر گرفته نشده است. در کل، لایه‌های داخلی (پنهان) در یک شبکه‌ی چندلایه یاد می‌گیرند تا ویژگی‌های میانی‌ای را نمایش دهند که برای یادگیری تابع هدف مفید بوده و به طور ضمنی در ورودی‌های شبکه بیان شده‌اند. این قابلیت، برای مثال، در شبکه‌ای ۸×۳×۸ در قسمت ۴,۶,۴ در کدگذاری منطقی انجام شده برای اعداد ۱ تا ۸ و در مثال تشخیص چهره در قسمت ۴,۷ با ویژگی‌های عکس در لایه‌ی پنهان بیان شده‌اند.
- overfit بر روی داده‌های آموزشی مشکلی مهم در یادگیری شبکه‌های عصبی است. overfit به شبکه‌ای ختم می‌شود که تعمیم ضعیفی روی داده‌های جدید دارد اما کارایی عالی‌ای روی داده‌های آموزشی دارد. متدهای Cross-Validation را می‌توان برای تخمین نقطه‌ی ایست مناسب برای جستجو و مینیمم کردن ریسک overfit به کاربرد.
- با وجود اینکه Backpropagation متداول‌ترین الگوریتم برای یادگیری شبکه‌های عصبی است، اما الگوریتم‌های بسیار دیگری ارائه شده‌اند، این الگوریتم‌ها شامل الگوریتم‌هایی برای اهداف خاص می‌شوند. برای مثال، متدهای شبکه‌های عصبی حلقه‌دار، شبکه‌هایی را آموزش می‌دهند که حلقه‌های مستقیم دارند و الگوریتم‌هایی چون Cascade Correlation ساختار شبکه را علاوه بر وزن‌های شبکه تغییر می‌دهند.

اطلاعات بیشتر در مورد شبکه‌های عصبی را می‌توانید در فصول دیگر این کتاب بیابید. توجیهی بیزی برای انتخاب معیار خطای مربعی در فصل ۶ ارائه می‌شود، در این فصل همچنین توجیهی برای مینیمم کردن cross-entropy به جای خطای مربعی در شرایط خاص آورده شده

است. نتایج تئوری تعداد نمونه‌های آموزشی لازم برای رسیدن به شبکه‌ای قابل‌اطمینان برای توابع حقیقی و بعد Vapnik-Chervonenkis برای شبکه‌های خاص در فصل ۷ بحث شده‌اند. در فصل ۵ نیز بحثی در مورد overfit و چگونگی دوری از آن آورده شده. همچنین در فصل ۱۲ متدهایی برای استفاده از دانش قبلی برای بهبود تعمیم دقت شبکه‌های عصبی مورد بحث قرار گرفته است.

کار بر روی شبکه‌های عصبی به زمان‌های اولیه‌ی علوم کامپیوتر بر می‌گردد. (McCulloch and Pitts 1943) مدلی برای یک نورون مشابه پرسپترون ارائه کردند، در طی دهه‌ی ۱۹۶۰ کارهای بسیاری بر روی قابل‌قبول بودن این مدل انجام گرفت. در اوایل دهه‌ی ۶۰ (Windrow and Hoff 1960) شبکه‌های پرسپترون (که آن را adeline می‌نامیدند) و قانون دلتا را مورد بررسی قرار دادند، (Rosenblatt 1962) همگرایی قانون آموزش پرسپترون را ثابت کرد. با این وجود، در اواخر دهه‌ی ۶۰ مشخص شد که پرسپترون تک لایه محدودیت نمایش دارد و الگوریتم مؤثری نیز برای آموزش شبکه‌های چندلایه معرفی نشده بود. (Minsky and Papert 1969) نشان دادند که حتی تابع ساده‌ای چون XOR را نمی‌توان با شبکه‌های پرسپترون تک لایه نمایش داد و کار بر روی شبکه‌های عصبی در دهه‌ی ۷۰ متوقف شد.

در اواسط دهه‌ی ۸۰ با اختراع Backpropagation و الگوریتم‌های مربوطه‌ی آموزش شبکه‌های چندلایه (Rumelhart and McClelland 1986; Parker 1985)، کار بر روی شبکه‌های عصبی دوباره از سر گرفته شد. اساس این ایده‌ها را می‌توان در کارهای قبلی جست (Werbos 1975). از دهه‌ی ۸۰ Backpropagation به یکی از متداول‌ترین متدهای یادگیری تبدیل شد و بسیاری از روش‌های دیگر مربوطه‌ی شبکه‌های عصبی مورد مطالعه قرار گرفت. با ظهور رایانه‌های کم قیمت در همان دوره تحقیقات بر روی الگوریتم‌های محاسباتی‌تر که در دهه‌ی ۶۰ ممکن نبود شروع شد.

کتاب‌های زیادی به مبحث شبکه‌های عصبی اختصاص یافته است. یکی از کتاب‌های قدیمی اما مفید درباره‌ی متدهای یادگیری پارامتری برای تشخیص الگو توسط (Duda and Hart 1973) نوشته شده است. کتاب (Widrow and Stearns 1985) به پرسپترون‌ها و شبکه‌های تک لایه‌ی مربوطه و کاربردشان می‌پردازد. (Rumelhart and McClelland 1986) مجموعه‌ی منتخبی از مقالات که علاقه‌ی کار بر روی این متدها را افزایش داد را از اواسط دهه‌ی ۸۰ به بعد جمع‌آوری کرده‌اند. کتاب‌های اخیر در مبحث شبکه‌های عصبی شامل (Bishop 1996)، (Chauvin and Rumelhart 1995)، (Freeman and Skapina 1991)، (Fu 1994)، (Hecht-Nielsen 1990) و (Hertz et al. 1991) می‌شود.

تمرینات

۴٫۱ مقادیر وزن‌های w_0، w_1 و w_2 را برای پرسپترونی که سطح تصمیمش در شکل ۴٫۳ آورده شده تعیین کنید. فرض کنید که این سطح محور x_1 را در ۱− و محور x_2 را در ۲ قطع می‌کند.

۴٫۲ پرسپترونی با دو ورودی طراحی کنید که تابع منطقی $A \wedge \neg B$ را نشان دهد. از شبکه‌ای دو لایه از پرسپترون‌ها برای نشان دادن A XOR B استفاده کنید.

۴٫۳ دو پرسپترون با رابطه‌ی مقدار حدی $w_0 + w_1x_1 + w_2x_2 > 0$ را در نظر بگیرید. پرسپترون A وزن‌های زیر را دارد

$$w_0 = 1, \qquad w_1 = 2, \qquad w_2 = 1$$

و پرسپترون B وزن‌های زیر را داراست

$$w_0 = 0, \qquad w_1 = 2, \qquad w_2 = 1$$

تعیین کنید که آیا پرسپترون A از پرسپترون B کلی‌تر است؟ (تعریف کلی‌تر بودن در فصل ۲ آمده است).

۴,۴ از قانون آموزش دلتا برای یک واحد خطی با دو ورودی استفاده کنید و آن را برای تناسب با مفهوم هدف $-2 + x_1 + 2x_2 > 0$ آموزش دهید. خطای E را بر حسب تعداد تکرارهای آموزش رسم کنید. سطح تصمیم را بعد از ۵، ۱۰، ۵۰، ۱۰۰، ... بار اجرا رسم کنید.

(a) از مقادیر مختلف ثابت برای η استفاده کرده و همچنین از مقدار متغیر η_0/i برای آموزش استفاده کنید. عملکرد کدام حالت بهتر است؟

(b) از افزایش و آموزش دسته‌ای[۲۱] استفاده کنید. کدام یک زودتر همگرا می‌شود؟ هر دو معیار تعداد تغییر وزن‌ها و کل زمان اجرا را در نظر بگیرید.

۴,۵ قانون شیب نزول را برای تک خروجی‌ای به فرم o به شکل زیر استخراج کنید

$$o = w_0 + w_1x_1 + w_1x_1^2 + w_2x_2 + w_2x_2^2 + \cdots + w_nx_n + w_nx_n^2$$

۴,۶ به طور غیررسمی توضیح دهید که چرا قانون آموزش دلتا در رابطه‌ی ۴,۱۰ فقط تخمینی از قانون شیب نزول رابطه‌ی ۴,۷ است.

۴,۷ شبکه‌ی عصبی تک‌سویه‌ای را در نظر بگیرید که دو ورودی a و b و یک واحد پنهان c و یک واحد خروجی d دارد. این شبکه پنج وزن $(w_{ca}, w_{cb}, w_{c0}, w_{dc}, w_{d0})$ دارد، در این نمایش w_{x0} نشان‌دهنده‌ی مقدار آستانه‌ی برای واحد x است. این وزن‌ها را با مقادیر اولیه‌ی (.1,.1,.1,.1,.1) مقداردهی اولیه کنید سپس الگوریتم Backpropagation را به آن‌ها اعمال کنید. فرض کنید که ضریب یادگیری η=.3، تکانه α=.9، وزن‌ها را برای هر نمونه جداگانه تغییر دهید، و نمونه‌های آموزشی به صورت زیرند:

a	B	D
۱	۰	۱
۰	۱	۰

۴,۸ الگوریتم Backpropagation در جدول ۴,۲ را طوری تغییر دهید که بر روی واحدهایی که از تابع tanh به جای تابع سیگموید استفاده می‌کنند عمل کند. بدین معنا که $o = \tanh(\vec{w}.\vec{x})$ قانون تغییر وزن را برای لایه خروجی و لایه پنهان ارائه کنید.

(راهنمایی $\tanh'(x) = 1 - \tanh^2(x)$).

۴,۹ شبکه‌ی ۸×۳×۸ نشان داده شده در شکل ۴,۷ را در نظر بگیرید. فرض کنید که برای چنین کار مشابهی می‌خواهیم از شبکه‌ای ۸×۱×۸ کمک بگیریم؛ شبکه‌ای که فقط یک واحد پنهان دارد. توجه دارید که هشت نمونه‌ی آموزشی شکل ۴,۷ را می‌توان با هشت مقدار برای گره پنهان نظیر کرد (برای مثال ۰,۱، ۰,۲، ... ۰,۸). آیا بنابراین شبکه‌ای با فقط یک واحد پنهان می‌تواند تابع همانی را بر روی این نمونه‌های آموزشی یاد بگیرد. راهنمایی: این سؤال را در نظر بگیرید که "آیا مقادیری برای وزن‌های لایه‌ی پنهان وجود دارد که بتواند کدگذاری بالا را در

[۲۱] batch learning

لایه‌ی پنهان ایجاد کند؟" "آیا مقادیری برای وزن‌های خروجی وجود دارد که بتوان ورودی را از این کدگذاری به سادگی استخراج کرد؟" و "آیا شیب نزول می‌تواند چنین وزن‌هایی را پیدا کند؟"

۴,۱۰ تابع خطای جایگزین زیر را به جای رابطه‌ی قسمت ۴,۸,۱ در نظر بگیرید.

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_k - o_{kd})^2 + \gamma\sum_{i,j} w_{ji}^2$$

قانون تغییر شیب نزول را برای این تعریف E محاسبه کنید. نشان دهید که این طرح را می‌توان با ضرب هر وزن در یک ثابت قبل از اعمال قانون شیب نزول جدول ۴,۲ پیاده‌سازی کرد.

۴,۱۱ از Backpropagation برای کار تشخیص چهره استفاده کنید. برای جزئیات کار از جمله داده‌های تصویری صورت، کد Backpropagation و کارهای خاص به http://www.cs.cmu.edu/~tom/book.html مراجعه کنید.

۴,۱۲ الگوریتم شیب نزول را در نظر بگیرید که برای یادگیری مفاهیم هدف متناسب با مستطیل‌های موجود در صفحه‌ی x,y به کار می‌رود. هر فرضیه را با گوشه سمت چپ پایین و راست بالا متناسب با llx,lly,urx,ury توصیف می‌شود. نقطه‌ی <x,y> تنها زمانی توسط فرضیه‌ی <llx,lly,urx,ury> مثبت دسته‌بندی می‌شود که نقطه‌ی <x,y> درون مستطیل قرار داشته باشد. از تعریف خطای E آمده در فصل استفاده کنید. آیا می‌توانید نسخه‌ای بازبینی شده از شیب نزول را ارائه دهید که چنین فرضیه‌های مستطیلی‌ای را یاد بگیرد. توجه دارید که E بر حسب llx,lly,urx,ury پیوسته نیست، مثل حالت پرسپترون. (راهنمایی: دو راه استفاده شده برای پرسپترون را در نظر بگیرید: (۱) تغییر قانون دسته‌بندی به صورتی که تابع پیش‌بینی تابعی پیوسته بر حسب ورودی‌ها باشد و (۲) تعریف تابع خطای دیگری، مثل فاصله تا مرکز مستطیل، برای استفاده در قانون دلتا برای آموزش پرسپترون). آیا الگوریتمتان به فرضیه‌ای با خطای مینیمم که در آن نمونه‌های مثبت و منفی با مستطیلی قابل‌تقسیم باشند میل می‌کند؟ چه زمانی نمی‌توان با مستطیل چنین کاری کرد؟ آیا مشکلات مینیمم‌های نسبی رخ می‌دهند؟ الگوریتم شما چه رابطه‌ای با متدهای نمادین ای که برای عطف ویژگی‌ها استفاده می‌شوند دارد؟

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

ابر صفحه‌ای	Hyperplane
اندیس	Index
بدترین حالت	worse case
برانگیخته	Excited
برنامه‌نویسی خطی	linear programming
بیزی	Bayesian
پهنای شبکه	Network width
تابع فشرده‌ساز	squashing function
تابع منطق	logistic function
تا نشده	Unfolded
ترتیب ساده به پیچیده	simple-to-complex
تک‌سویه	feed-forward

penalty term	جمله‌ی خطا
Interpolation	درون‌یابی
True	درست
linearly separable	دسته‌بندی پذیر خطی
Artificial neural networks	شبکه‌های عصبی مصنوعی
termination condition	شرط پایانی
Gradient Descent	شیب نزول
Incremental gradient descend	شیب نزول افزایشی
learning rate	ضریب یادگیری
False	غلت
Inhibited	غیر برانگیخته
perceptron rule	قانون پرسپترون
delta rule	قانون دلتا
generalization accuracy	قدرت تعمیم شبکه
Gradient	گرادیان
Node	گره
Sequential	ماشین‌های ترتیبی
validation set	مجموعه‌ی تأیید
Threshold	مقدار آستانه
linear unit	واحد خطی
Edge	یال