۱.

در آغاز فصل ۴ از کتاب، برای تعریف یک شبکه عصبی از ابزار گراف استفاده شده و شبکه عصبی به عنوان یک گراف معرفی می‌شود:

یک شبکه عصبی مصنوعی را می‌توان به صورت یک گراف جهت‌دار $G = (U, C)$ با مجموعه رئوس U و مجموعه یال‌های C تعریف کرد که در این صورت هر راس گراف $u \in U$ یک نورون نامیده می‌شود. در یک تقسیم‌بندی می‌توان مجموعه نورون‌ها را به سه دسته ورودی، مخفی و خروجی تقسیم نمود. مجموعه نورون‌های ورودی و خروجی می‌توانند با هم اشتراکاتی داشته باشند (یعنی یک نورون می‌تواند همزمان در دسته ورودی و خروجی قرار بگیرد) اما مجموعه نورون‌های مخفی هیچ اشتراکی با دو دسته دیگر ندارد. همچنین مجموعه نورون‌های ورودی و خروجی دارای حداقل یک عضو هستند در صورتی که نورون‌های مخفی می‌تواند تهی باشد:

$$U = U_{in} \cup U_{out} \cup U_{hidden}$$

$$U_{in} \neq \emptyset\,, \quad U_{out} \neq \emptyset$$

$$U_{hidden} \cap (U_{in} \cup U_{out}) = \emptyset$$

هر یال از این گراف به صورت $c = (v, u) \in C$ نشان‌دهنده یک اتصال جهت‌دار از نورون v به نورون u می‌باشد. همچنین در یک شبکه عصبی، برای هر یال یک مقدار وزن در نظر گرفته می‌شود که آن را با w_{uv} نشان می‌دهند. مجدداً به طور ویژه تاکید می‌شود که وزن w_{uv} اتصال $u \leftarrow v$ را نشان می‌دهد.

همچنین به منظور توصیف عملکرد هر نورون، چهار مقدار حقیقی به این نورون نسبت داده می‌شود که در پرسش ۳ به طور دقیق هرکدام را توضیح می‌دهیم. این چهار مقدار عبارتند از:

- ورودی شبکه : net_u
- مقدار فعالیت: act_u
- مقدار خروجی: out_u
- ورودی بیرونی: ext_u

سه مقدار اول در بین ۴ مقدار فوق، به ترتیب توسط سه تابع زیر در هر نورون تولید می‌شوند:

- تابع ورودی شبکه: که با دریافت مقادیر از پدران نود و وزن‌های ارتباطی یال‌های متناظر، یک مقدار فعالیت برای نورن تولید می‌کند.

$$f_{net}^{(u)}: \mathbb{R}^{2|pred\ (u)|+\kappa_1(u)} \to \mathbb{R}$$

- تابع فعالسازی: با گرفتن خروجی تابع قبل و یک فیدبک از خروجی خود، مقدار فعالیت جدیدی را تولید می‌کند:

$$f_{act}^{(u)}: \mathbb{R}^{\kappa_2(u)} \to \mathbb{R}$$

- تابع خروجی: با اعمال یک تابع نهایی روی مقدار فعالیت، خروجی نورون را تولید می‌کند.

$$f_{out}^{(u)}: \mathbb{R} \to \mathbb{R}$$

توجه کنید که در روابط فوق، منظور از $|pred\ (u)|$، تعداد پدران نورون مورد نظر در شبکه گرافی است. همچنین $\kappa_1(u)$ و $\kappa_2(u)$ بسته به نوع فعالیت نورون و انواع پارامترهایی که لازم دارد، تعیین می‌شوند. توابع فوق به طور مفصل‌تر در سوال سوم توضیح داده می‌شوند.

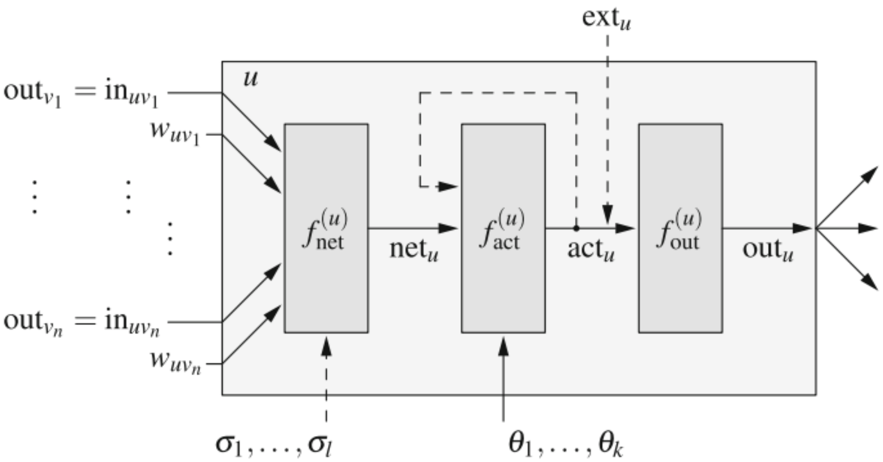
.۲

برای پاسخ به این سوال، بار دیگر یادآوری می‌شود که وزن w_{uv} اتصال $u \leftarrow v$ را نشان می‌دهد. لذا همه مقادیری که در یک سطر ماتریس زیر قرار گرفته‌اند، مقادیر ورودی به یک نورون خاص هستند:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0.4 & 0 & 0 & 0 & 0 \\ 0.6 & 0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0.6 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & 0 & 1 \end{bmatrix}$$

.۳

برای توضیح ساختار درونی هر نورون در کلی‌ترین حالت، می‌توان به شکل زیر از کتاب مرجع رجوع کرد:



همانطور که در شکل فوق دیده می‌شود، اولین تابع در بلوک فوق $f_{net}^{(u)}$ نام دارد. این تابع به تعداد $2|pred(u)|$ مقدار را در ورودی خود از سمت نودهای پدر دریافت می‌کند. در واقع $|pred(u)|$ تا از این مقادیر حاصل خروجی نورون‌های پدر بوده و به همین تعداد مقدار وزن‌های اتصال از نودهای پدر به نود مورد نظر وجود دارد. همچنین این تابع ممکن است برای محاسبات خود به پارامترهای $\sigma_1, \ldots, \sigma_l$ نیاز داشته باشد که در شکل فوق در اختیار آن گذاشته شده است. تابع مورد نظر با گرفتن ورودی‌های توضیح داده شده، در خروجی خود یک مقدار به نام net_u تولید می‌کند. بنابراین تعداد ورودی‌های این تابع می‌تواند بسته به تعداد پارامترها متفاوت باشد، لذا این تابع را با کمک تابع κ_1 و به صورت $f_{net}^{(u)}: \mathbb{R}^{2|pred\,(u)|+\kappa_1(u)} \to \mathbb{R}$ تعریف می‌کنند.

تابع بعدی تابع $f_{act}^{(u)}$ نام دارد که یک ورودی را از خروجی تابع قبل گرفته (net_u) و یک ورودی احتمالی هم به عنوان رابطه فیدبکی از خروجی خودش دریافت می‌کند (act_u). در نهایت این تابع با استفاده از پارامترهای $\theta_1, \ldots, \theta_k$ و دو ورودی ذکر شده، مقدار فعالیت جدید نورون (act_u) را محاسبه می‌نماید. تابع مذکور بسته به پارامترهای مد نظر و با توجه به وجود یا عدم وجود فیدبک، می‌تواند تعداد ورودی‌های متفاوتی دریافت کند لذا این تابع به کمک κ_2 و به صورت $f_{act}^{(u)}: \mathbb{R}^{\kappa_2(u)} \to \mathbb{R}$ تعریف می‌شود. ضمناً باید اشاره شود که اگر نورون ما از جنس یک نورون ورودی باشد، مقدار فعالیت آن (act_u) توسط ورودی بیرونی (ext_u) مقدار دهی اولیه می‌شود.

در جایگاه آخرین تابع، $f_{out}^{(u)}$ قرار دارد که از روی مقدار فعالیت نورون (act_u) مقدار خروجی نورون را تولید می‌نماید. این تابع معمولاً به منظور $scale$ کردن مقدار فعالیت نورون در بازه‌ای خاص با کمک تبدیل‌های خطی به کار می‌رود.

۴.

محاسبات صورت گرفته در نورون را می‌توان به دو فاز ورودی و فاز کاری تقسیم بندی نمود:

فاز ورودی به طور خلاصه به دریافت ورودی‌های خارجی و مقداردهی اولیه نورون‌ها اختصاص دارد. در این فاز مقدار فعالیت نورون‌های ورودی (act_u)، برابر با مقدار ورودی‌های خارجی (ext_u) قرار می‌گیرد. برای نورون‌های دیگر (یعنی نورون‌های مخفی و خروجی)، مقدار اولیه فعالیت (act_u) به طور دلخواه (و معمولاً صفر) قرار داده می‌شود. در انتهای این گام، تمامی مقادیر $f_{out}^{(u)}$ روی مقادیر فعالیت‌ها (act_u) اثر کرده و خروجی‌های اولیه (out_u) را تولید می‌نمایند. لذا در این مقطع همه نورون‌ها یک مقدار فعالیت اولیه و یک خروجی از روی آن دارند که در فاز بعد مورد استفاده قرار می‌گیرد.

در **فاز کاری**، مقدار ورودی‌های خارجی قطع می‌شود و حالا نوبت به دینامیک شبکه می‌رسد تا مقادیر فعالیت‌های هر نورون و در نتیجه مقدار خروجی هر نورون را محاسبه کند و برای این کار احتمالا لازم است تا محاسبات چندین بار تکرار شود. برای این که این اتفاق صورت بگیرد، همانطور که بالاتر نیز توضیح داده شد، سه تابع $f_{net}^{(u)}$، $f_{act}^{(u)}$ و $f_{out}^{(u)}$ بر روی ورودی‌های خود اثر کرده و مقادیر موجود در هر نورون را به روزرسانی می‌کنند. البته اگر نورون مد نظر یکی از نورون‌های ورودی باشد و هیچ پدری نداشته باشد که از آن مقداری را دریافت نماید، ما در آن نورون فرض می‌کنیم مقدار فعالیت ثابت مانده و از انتهای فاز ورودی به بعد تغییری در این مقدار اعمال نمی‌کنیم. این مسئله کمک می‌کند در شبکه‌های feedforward، علی‌رغم قطع شدن ورودی خارجی، باز هم نورون‌های ورودی مقادیر درستی را در خودشان خفظ نمایند.

تعداد گام‌هایی که برای انتهای فاز کاری و متوقف کردن محاسبات لازم است را می‌توان به صورت یک مقدار مشخص از پیش تعیین شده در نظر گرفت و یا می‌توان آن قدر محاسبات را انجام داد و جلوبرد تا بالاخره دینامیک شبکه به یک نقطه ثابت برسد. منظور از نقطه ثابت، چینشی از مقدارگیری نورون‌هاست که بعد از آن با انجام محاسبات دیگر تغییری در مقادیر به وجود نیاید.

همچنین ترتیب آپدیت شدن نورون‌ها در فاز کاری باید از قبل مشخص شود. مثلاً می‌توان فرض کرد که همگی نورون‌ها در فاز کاری به صورت همزمان آپدیت می‌شوند و از مقادیر گذشته پدران خود در زمان آپدیت استفاده کنند، یا با یک ترتیب از قبل مشخص شده و به طور غیر سنکرون آپدیت‌ها صورت بگیرد.

۵.

ترتیب توپولوژیک یا مرتب‌سازی توپولوژیکی، از جمله مفاهیمی است که برای گراف‌های جهت‌دار بدون دور تعریف می‌شود (DAG). در واقع در چنین گراف‌هایی می‌توان یک یال جهت دار را به چشم یک رابطه مقایسه‌ای بزرگتر کوچکتری نگاه کرد. در این صورت می‌توان یک ترتیب مثل $u_1, u_2, \dots, u_n$ روی راس‌های گراف معرفی نمود که در این صورت می‌توان به u_1 به عنوان کوچکترین مقدار دنباله و به u_n به چشم بیشترین مقدار دنباله نگاه کرد. به عبارت دیگر، در یک دنباله مرتب شده به صورت فوق، تنها یال‌های از چپ به راست بین نودها مجاز است و یال‌های برگشتی وجود ندارد. وجود چنین ترتیبی در گراف‌های DAG تضمین شده است.

وجود یک ترتیب توپولوژیک در یک شبکه عصبی، می‌تواند این اجازه را به ما بدهد که برای محاسبه تک تک مقادیر نودهای شبکه، از اولین نود در دنباله مرتب شده شروع کرده، محاسبات را به ترتیب انجام داده و یکی یکی روی همان لیست جلو رویم تا نهایتاً به انتهای دنباله برسیم. در این صورت با توجه به این که در این نوع ترتیب همه یال‌ها حتماً از چپ به راست است، در محاسبات مربوطه به هر نود، حتماً پدران آن نود پیش از خود نود آپدیت شده و مقادیر جدید را به دست آورده‌اند. این نحوه آپدیت کردن نودها همان روشی است که در شبکه‌های feedforward مد نظر قرار دارد.

.۶

در جدول رو به رو مقادیر هر نورون در هر گام آورده شده است. ترتیب آپدیت متغیرها به صورت u_3، u_2 و در نهایت u_1 می‌باشد. بنابراین مقادیر قرمز آن هایی هستند که در هر گام مقدار جدیدی گرفته‌اند. همچنین همانطور که ملاحظه می‌شود، گام ۷ دقیقاً شبیه گام اول است لذا مقادیر از این گام به بعد دقیقاً تکرار می‌شوند.

در گام اول لازم است تا مقدار u_3 آپدیت شود. این نود یک ورودی -2 از u_1 دارد و یک ورودی 0 از u_2. لذا مقدار ورودی این نورون از مقدار آستانه 1 کمتر خواهد بود و نورون فعال نمی‌شود. در گام بعدی نورون u_2 تنها یک ورودی $+1$ از سمت u_1 دارد لذا مقدار آستانه 1 را فعال می‌کند و مقدار یک به خود می‌گیرد. نهایتاً در گام سوم، نورون u_1 که فقط ورودی صفر را از سمت u_3 دریافت می‌کند، به آستانه نمی‌تواند برسد و مقدار بعدی آن برابر صفر خواهد بود. با همین روش می‌توان گام‌های بعدی محاسبات را نیز دنبال کرد و جدول رو به رو به دست می‌آید.

	u_1	u_2	u_3	
Init value	1	0	0	
1	1	0	0	$net\ u_3 = -2$
2	1	1	0	$net\ u_2 = 1$
3	0	1	0	$net\ u_1 = 0$
4	0	1	1	$net\ u_3 = 3$
5	0	0	1	$net\ u_2 = 0$
6	1	0	1	$net\ u_1 = 4$
7	1	0	0	$net\ u_3 = -2$
8	1	1	0	$net\ u_2 = 1$
9	0	1	0	$net\ u_1 = 0$
10	0	1	1	$net\ u_3 = 3$

.۷

یک شبکه نورونی با مجموعه نورون‌های ورودی $U_{in} = \{u_1, \dots, u_n\}$ و مجموعه نورون‌های خروجی $U_{out} = \{v_1, \dots, v_m\}$ را در نظر بگیرید. در این صورت، منظور از یک وظیفه یادگیری معین که آن را با L_{fixed} نشان می‌دهند، عبارت است از مجموعه یک سری نمونه‌های آموزشی مانند $l = (i^{(l)}, o^{(l)})$. به عبارت دقیق‌تر، هر الگو یا نمونه آموزشی را می‌توان تشکیل شده از دو بخش دانست. بخش اول بردار ورودی ($i^{(l)} = (ext_{u_1}, \dots, ext_{u_n})$) و بخش دوم یک بردار خروجی ($o^{(l)} = (o^l_{v_1}, \dots, o^l_{v_2})$)

با تفاسیر فوق، منظور از آموزش شبکه برای انجام یک وظیفه یادگیری معین مثل L_{fixed} آن خواهد بود که اگر هر پترن آموزشی مثل l از داخل L_{fixed} برداشته شود و قسمت ورودی آن ($i^{(l)}$) به به عنوان ورودی به بخش U_{in} داده شود، در این صورت شبکه بتواند خروجی‌های مطلوب ($o^{(l)}$) را در نودهای U_{out} تولید نماید.

در این صورت، خطا را می‌توان به صورت زیر تعریف کرد:

$$e = \sum_{l \in L_{fixed}} e^{(l)} = \sum_{v \in U_{out}} e_v = \sum_{l \in L_{fixed}} \sum_{v \in U_{out}} e_v^{(l)} = \sum_{l \in L_{fixed}} \sum_{v \in U_{out}} \left(o_v^{(l)} - out_v^{(l)}\right)^2$$

همانطور که در تعریف فوق دیده می‌شود، خطا را می‌توان به صورت حاصل جمع خطای هر یک از نمونه‌های آموزشی موجود در L_{fixed} به دست آورد. به عبارت دیگر، به ازای هر $l \in L_{fixed}$، قسمت ورودی داده به ورودی شبکه داده می‌شود و خروجی شبکه با خروجی مطلوب مقایسه می‌شود. توان دوی مجموع مقدار تفاوت ایجاد شده بین این دو مقدار، به عنوان خطای شبکه گزارش می‌شود. استفاده از توان دو به این علت است که مقدارهای خطای مثبت و منفی ایجاد شده همدیگر را خنثی نکنند. همچنین خطای MSE این ویژگی را دارد که در سرتاسر دامنه خود مشتق‌پذیر است و همچنین مقادیر خطای زیاد را به مقدار خیلی بیشتری جریمه می‌کند.

.۸

در یک امر یادگیری آزاد، برخلاف یادگیری معین، نمونه‌های آموزشی تنها از یک بخش تشکیل شده‌اند و امر آموزش تمرکز در یافتن الگوهای ورودی به صورت unsupervised دارد. در واقع برای یک شبکه با مجموعه نورون‌های ورودی $U_{in} = \{u_1, \dots, u_n\}$، یک وظیفه یادگیری آزاد که آن را با L_{free} نمایش می‌دهند، تشکیل شده است از یک سری داده‌های آموزشی مثل $l = \left(i^{(l)}\right)$ که هر $i^{(l)}$ خود به صورت یک چینش مقداردهی به نورون‌های ورودی تعریف می‌شود. ($i^{(l)} = \left(ext_{u_1}, \dots, ext_{u_n}\right)$)

بر خلاف قسمت قبل که خروجی خاصی از هر داده ورودی مد نظر بود، در این جا هدف یافتن مقادیر خروجی خاصی نیست و لذا نمی‌توان رابطه خطا را مشابه قبل پیدا کرد. در واقع تمرکز یادگیری در این قسمت بر آن است که با دیدن هر داده ورودی، شبکه بتواند یک مجموعه ویژگی از دل این داده‌ها استخراج کند که این داده‌ها نماینده و خلاصه کننده خوبی از داده‌های ورودی باشند. به عبارت دیگر، ورودی‌هایی که به هم شباهت دارند، خروجی مشابهی تولید کنند در حالی که خروجی‌های تولید شده برای ورودی‌های متفاوت، از هم فاصله زیادی داشته باشند. خروجی چنین شبکه‌ای می‌تواند به عنوان مثال برای خوشه بندی داده‌های ورودی استفاده شود یا به عبارت دیگر، ورودی‌هایی که شباهت زیادی به یکدیگر دارند در یک دسته کنار هم قرار بگیرند.

لذا برای تعریف مقدار خطا و تابع هدف در چنین شبکه‌هایی، ما به یک تابع تعریف فاصله احتیاج داریم. در نتیجه هدف ما در چنین شبکه‌ای آن خواهد بود که به ازای ورودی‌های با فاصله کم از یکدیگر، خروجی‌های مشابه در این شبکه تولید شود و به ازای ورودی‌های با فاصله، خروجی‌ها نیز از هم فاصله‌دار باشند.

.۹

یک پرسپترون r-لایه‌ای، یک شبکه عصبی با یک گراف $G = (U, C)$ است که شروط زیر را برآورده می‌سازد:

- $U_{in} \cap U_{out} = \emptyset$

یا به عبارت دیگر، نورون‌های ورودی و خروجی هیچ اشتراکی با یکدیگر ندارند.

- $U_{hidden} = U_{hidden}^{(1)} \cup U_{hidden}^{(2)} \cup \dots \cup U_{hidden}^{(r-2)} = \emptyset$
- $\forall 1 \leq i < j \leq r-2 \colon U_{hidden}^{(i)} \cap U_{hidden}^{(j)} = \emptyset$

یا به عبارت دیگر، نورون‌های لایه مخفی به $r-2$ دسته افراز می‌شوند که این دسته‌ها با هم هیچ اشتراکی ندارند. هر پرسپترون حتماً یک لایه ورودی و یک لایه خروجی دارد اما می‌تواند لایه‌های مخفی داشته باشد یا نداشته باشد.

- $C \subseteq \left(U_{in} \times U_{hidden}^{(1)}\right) \cup \left(U_{hidden}^{(1)} \times U_{hidden}^{(2)}\right) \cup \dots \cup \left(U_{hidden}^{(r-2)} \times U_{out}\right)$

یا به عبارت دیگر، تنها یال‌های از ورودی به لایه اول مخفی، از لایه iام مخفی به لایه $i+1$ ام مخفی و از لایه آخر مخفی به خروجی مجاز هستند. البته اگر $r = 2$ باشد، در اینصورت لایه ورودی مستقیماً به لایه خروجی متصل است و خواهیم داشت: $C \subseteq U_{in} \times U_{out}$.

علاوه بر موارد فوق که ساختار گرافی و نحوه اتصالات درون شبکه‌ای را نشان می‌دهد، یک شبکه پرسپترون دارای ویژگی‌های زیر در توابع ورودی و تابع فعالسازی خود می‌باشد:

- مقدار net_u برای نورون‌های لایه مخفی و خروجی، به صورت جمع وزن‌دار ورودی‌های نورون تعیین می‌شود. به عبارت دیگر، برای هر نورون $u \in U_{out} \cup U_{in}$ داریم:

$$f_{net}^{(u)} = \sum_{v \in pred\,(u)} out_v \times w_{uv}$$

- مقدار فعالیت هر نورون، با اعمال یک تابع غیرنزولی غیرخطی به نام sigmoid یا یک تابع خطی بر روی ورودی‌های آن به دست می‌آید. در مورد ویژگی‌های تابع sigmoid در سوال بعدی صحبت خواهد شد. اما در مورد تابع فعالسازی خطی با پارامترهای α و θ، می‌توان رفتار کلی زیر را معرفی نمود:

$$f_{act}^{(u)}(net_u, \theta, \alpha) = \alpha \times net_u - \theta$$

.۱۰

به طور کلی، یک تابع فعالسازی sigmoid، تابعی غیرنزولی و محدود است که ویژگی‌های زیر را داشته باشد:

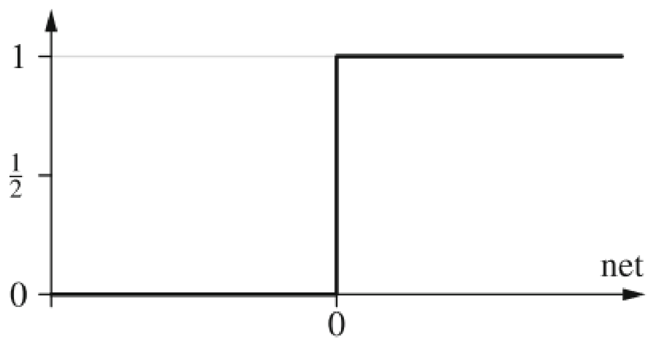
- $f: \mathbb{R} \rightarrow [0,1]$
- $\lim\limits_{x\to\infty} f(x) = 1$
- $\lim\limits_{x\to-\infty} f(x) = 0$

البته مورد آخر برای توابع sigmoid تک قطبی صادق است. در واقع برای تابعی مثل $tanh$ که یک تابع $sigmoid$ دوقطبی محسوب می‌شود، ویژگی آخر به صورت $\lim\limits_{x\to-\infty} f(x) = -1$ برقرار می‌باشد. البته تبدیل یک تابع سیگموید دو قطبی به تک قطبی، با یک تبدیل ساده قابل انجام است لذا کتاب تمرکز خود را روی توابع تک قطبی قرار می‌دهد. همچنین لازم است اشاره شود که توابع sigmoid غالباً یک پارامتر قابل تنظیم مثل θ دارند که ویژگی‌های آن را مشخص می‌کند.

به عنوان یک تابع سیگموید ناپیوسته، می‌توان تابع پله را معرفی نمود:

(Heaviside or unit) step function:

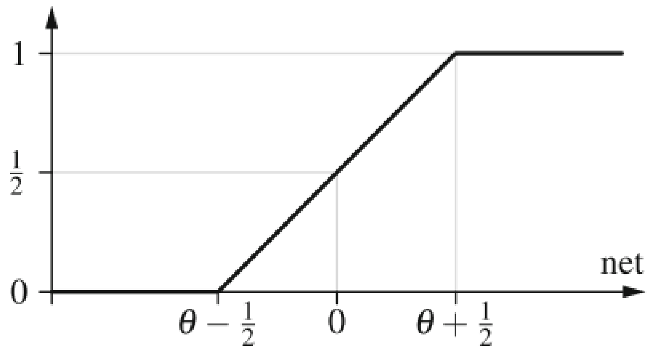
$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1 & \text{if net} \geq \theta, \\ 0 & \text{otherwise.} \end{cases}$$



همچنین تابع شبه خطی، یک تابع پیوسته و مشتق ناپذیر است:

semi-linear function:

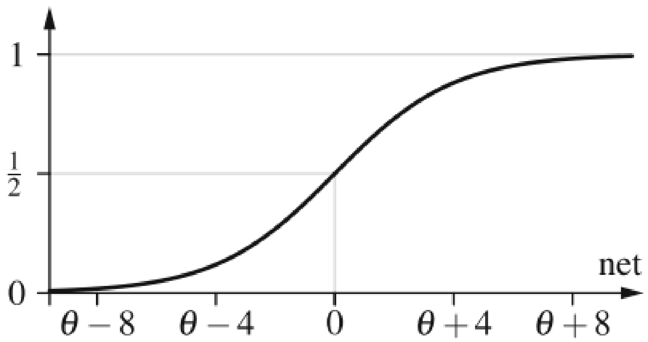
$$f_{\text{act}}(\text{net},\theta) = \begin{cases} 1 & \text{if net} > \theta + \frac{1}{2}, \\ 0 & \text{if net} < \theta - \frac{1}{2}, \\ (\text{net} - \theta) + \frac{1}{2} & \text{otherwise.} \end{cases}$$



و به عنوان مثال آخر از یک تابع پیوسته و مشتق پذیر، می‌توان تابع logistic را معرفی نمود:

logistic function:

$$f_{\text{act}}(\text{net},\theta) = \frac{1}{1 + e^{-(\text{net} - \theta)}}$$

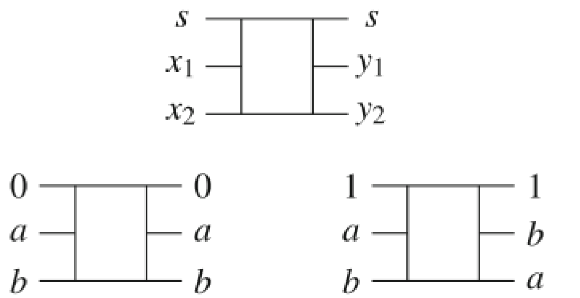


.۱۱

گیت Fredkin که یک مدارمحاسباتی از خانواده مدارهای منطقی می‌باشد، در حوزه reversible computing استفاده زیادی دارد. این گیت منطقی یک گیت universal می‌باشد که این مسئله به آن معنیست که هر گیت منطقی دیگری را می‌توان توسط چنین گیتی پیاده‌سازی نمود.

نحوه عملکرد این گیت به این صورت است که سه ورودی a،b و s دریافت کرده و سه خروجی نیز تولید می‌کند. ورودی s مستقیماً و بدون تغییر به خروجی منتقل می‌شود. اما ورودی s در مورد a و b مانند یک سوییچ عمل می‌کند. یعنی اگر s صفر باشد، a و b بدون تغییر به خروجی منتقل می‌شوند اما اگر s یک باشد، جای دو متغیر a و b عوض می‌شود.

عملکرد این گیت را می‌توان در شکل‌ها و جدول صحت زیر به طور خلاصه مشاهده کرد:



s	0 0 0 0 1 1 1 1
x_1	0 0 1 1 0 0 1 1
x_2	0 1 0 1 0 1 0 1
y_1	0 0 1 1 0 1 0 1
y_2	0 1 0 1 0 0 1 1

در کتاب مورد بررسی، یک شبکه پرسترون سه لایه معرفی می‌شود که می‌تواند عملکردی مشابه گیت فوق از خود نشان دهد.

.۱۲

علت آن است که با وجود صرفاً فعال‌سازی‌های خطی، عمیق کردن شبکه تاثیری در تغییر قدرت آن نخواهد داشت. به عبارت دیگر، اگر یک پرسپترون چند لایه فقط از واحدهای خطی ساخته شده باشد، هر چقدر هم لایه‌های میانی به آن افزوده شود، می‌توان کل شبکه را با یک مدل دو لایه‌ای خطی با وزن‌های معادل مدل کرد.

برای درک بهتر این مسئله و به عنوان یک مثال ساده، فرض کنید شبکه از سه لایه ورودی (U_{in})، مخفی (U_{hidden}) و خروجی (U_{out}) با وزن‌های ماتریسی مشخص و توابع فعالسازی خطی تشکیل شده باشد. در این صورت:

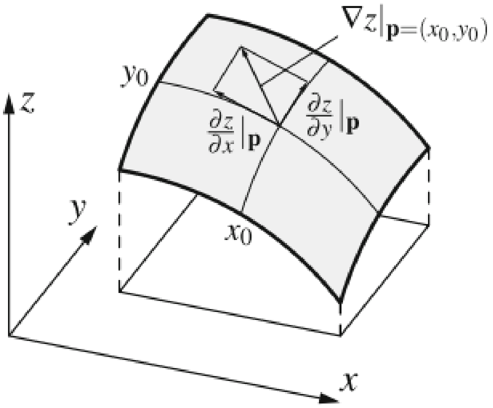
$$net_{U_{hidden}} = \boldsymbol{W_1}.\boldsymbol{in}_{U_{hidden}} = \boldsymbol{W_1}.\boldsymbol{out}_{U_{in}} \Rightarrow out_{U_{hidden}} = act_{U_{hidden}} = \alpha_1 \times net_{U_{hidden}} - \theta_1$$

$$\begin{aligned} net_{U_{out}} &= \boldsymbol{W_2}.\boldsymbol{in}_{U_{out}} = \boldsymbol{W_2}.\boldsymbol{out}_{U_{hidden}} \Rightarrow out_{U_{out}} = act_{U_{out}} = \alpha_2 \times net_{U_{out}} - \theta_2 \\ &= \alpha_2 \boldsymbol{W_2}.\left(\alpha_1 \times net_{U_{hidden}} - \theta_1\right) - \theta_2 = \alpha_2 \boldsymbol{W_2}.\left(\alpha_1 \boldsymbol{W_1}.\boldsymbol{out}_{U_{in}} - \theta_1\right) - \theta_2 \\ &= \underbrace{\alpha_2 \boldsymbol{W_2}.\alpha_1 \boldsymbol{W_1}}_{\boldsymbol{W}}.\boldsymbol{out}_{U_{in}} \underbrace{-\theta_1 - \theta_2}_{-\theta} \end{aligned}$$

همانطور که دیده می‌شود، می‌توان لایه میانی را کاملاً حذف کرد و نورون‌های ورودی را مستقیماً با یک ماتریس وزن معادل به نورون‌های لایه خروجی متصل نمود.

فرض کنید که مقادیر پارامترهای قابل یادگیری شبکه، با یک سری مقادیر دلخواه مقداردهی اولیه شده‌اند. اگر در یک همسایگی کوچک از این مقادیر بنگریم، می‌بینیم که به ازای یک سری نقاط در این همسایگی، تابع خطا مقدار کمتری خواهد داشت. در واقع اگر از نقطه فعلی یک گام هرچند کوچک به سمت برخی از راستاهای خاص برداریم، می‌بینیم که مقدار تابع ضرر کاهش می‌یابد. ایده استفاده از gradient descent همین است که در هر گام، پارامترهای مدل را به اندازه کمی اما در جهتی آپدیت کنیم که نسبت به نقطه فعلی، خطای کمتری داشته باشد. اگر چندین مرتبه این گام‌های کوچک را پشت سر بگذاریم، در نهایت خواهیم دید که در نقطه‌ای قرار می‌گیریم که مقدار خطا خیلی کمتر از مقدار اولیه شده است. بنابراین با کمک این روش می‌توان پارامترهای شبکه را طوری آموزش داد که خطای کلی کم شود.

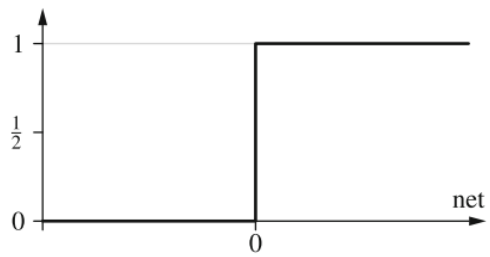
اما برای پیدا کردن جهت کاهش خطا، می‌توان از یک شیوه تحلیلی استفاده کرد و از مشتقات تابع ضرر نسبت به پارامترها استفاده نمود. در واقع از بحث‌های ریاضی ۲ به خاطر داریم که در هر نقطه از یک رویه در فضای پارامترها، راستایی که بیشترین کاهش را در مقدار رویه ایجاد می‌کند راستای خلاف گرادیان است. (یک نمونه در شکل نشان داده شده است) لذا به نظر می‌رسد برای یافتن راستایی که بیشترین کاهش را در مقدار خطا نتیجه می‌دهد، کافی است تا گرادیان را به صورت تحلیلی محاسبه کرده و در خلاف آن حرکت نماییم.



به طور کلی روش مطرح شده، یک روش بسیار کارا و سریع برای حل بسیاری از مسائل بهینه سازی می‌باشد. البته یکی از شروط کارایی این الگوریتم آن است که تابع خطا نسبت به پارامترها مشتق پذیر بوده و همچنین تابع ثابت نباشد. مثلاً اگر تابع ثابت پله را در نظر بگیریم، در این صورت در نقطه آستانه، تابع مشتق پذیر نیست. همچنین برای مقادیر دیگر ورودی، مشتق خروجی نسبت به ورودی صفر است چرا که با یک خط ثابت صفر یا یک با شیب صفر سر و کار داریم. این مسئله باعث می‌شود که امکان محاسبه مشتق خروجی نسبت به پارامترها وجود نداشته باشد (چون مقدار مشتق صفر می‌شود) و لذا یافتن جهت حرکت بهینه نامکن خواهد بود.

(Heaviside or unit) step function:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1 & \text{if net} \geq \theta, \\ 0 & \text{otherwise.} \end{cases}$$



.۱۴

در رویکرد یادگیری برخط، هر بار که یکی از داده‌ها به مدل داده می‌شود، همه وزن‌ها در راستای کم کردن خطای مربوط به آن داده خاص آپدیت می‌شوند. سپس در گام بعد نمونه آموزشی بعدی به مدل داده می‌شود، مجدداً آپدیت‌ها متناسب با آن انجام می‌شوند و همینطور برای همه داده‌های بعدی این اتفاق می‌افتد.

اما در رویکرد یادگیری دسته‌ای، همه داده‌های دیتاست یکجا در اختیار مدل قرار می‌گیرند. سپس مقدار خطا برای همه این داده‌ها محاسبه شده و همه این مقادیر خطا با هم جمع زده می‌شوند. در گام آپدیت کردن وزن‌ها، مقدار مشتق با توجه به این مقدار خطای کلی محاسبه می‌شود و لذا آپدیت کردن همه وزن‌ها در یک گام و با توجه به تمامی دادگان صورت می‌گیرد. در این روش به هر یک از گام‌های آپدیت پارامترها یک epoch گفته می‌شود.

.۱۵

قاعده به روزرسانی برای نودهای لایه آخر، به صورت زیر است:

$$\forall u \in U_{out}:\ \boldsymbol{\Delta w_u^{(l)}} = -\frac{\eta}{2}\boldsymbol{\nabla}_{\boldsymbol{W_u}} e_u^{(l)} = \eta\left(o_u^{(l)} - out_u^{(l)}\right)\frac{\partial out_u^{(l)}}{\partial net_u^{(l)}}\boldsymbol{in_u^{(l)}}$$

که در رابطه فوق، η نرخ یادگیری، $o_u^{(l)}$ مقدار خروجی مطلوب و $out_u^{(l)}$ مقدار خروجی به دست آمده از شبه می‌باشد. همچنین $\frac{\partial out_u^{(l)}}{\partial net_u^{(l)}}$ مشتق خروجی نسبت به ورودی نورون و $\boldsymbol{in_u^{(l)}}$ بردار ورودی به نورون لایه آخر را نشان می‌دهد. همانطور که در رابطه نیز دیده می‌شود، مقدار $\boldsymbol{\Delta w_u^{(l)}}$ تنها به ازای نمونه lام از دیتاست محاسبه شده است و برای در نظر گرفتن رویکرد batch update باید روی تمامی دادگان جمع زده شود. توجه شود که وجود علامت منفی، تضمین می‌کند که آپدیت وزن‌ها در جهت کم کردن مقدار خطا صورت پذیرد.

مشکلی که در استفاده از قاعده دلتا به منظور به‌روزرسانی وزن‌های لایه‌های میانی وجود دارد، آن است که برای لایه‌های میانی، دسترسی به مقدار هدف بهینه وجود ندارد و لذا نمی‌توان تابع خطا را تشکیل داد و متناسب با آن مشتق گرفت.

.۱۶

قاعده backpropagation، پیشنهاد می‌کند که به منظور محاسبه مقدار آپدیت یک پارامتر، از مشتق‌پذیری تابع خطا نسبت به آن پارامتر استفاده نماییم. به عبارت دیگر، تابع خطا که به دنبال کمینه کردن آن هستیم، نسبت به هریک از وزن‌های نورون‌های میانی شبکه دارای مشتق بوده و می‌توان از مشتق‌گیری خطا نسبت به آن پارامتر برای پیدا کردن جهت گرادیان استفاده کرد.

ابزاری از ریاضیات که در اینجا به کمک ما برای محاسبه گرادیان نسبت به نورون‌های لایه‌های میانی می‌آید، قاعده زنجیره‌ای در بحث مشتق‌هاست. قاعده زنجیره‌ای این اجازه را می‌دهد که برای محاسبه مشتق نسبت به پارامترهای لایه $\boldsymbol{i}$ام، از مقدار مشتق محاسبه شده نسبت به لایه $(i+1)$ام استفاده شود. لذا در الگوریتم backpropagation، محاسبه مشتق خطا نسبت به پارامترها از لایه آخر شروع شده و به سمت لایه‌های ابتدایی پیش می‌رود و در بین راه محاسبات لازم برای هر لایه را انجام می‌دهد. در نهای قاعده آپدیت برای وزن‌های یک لایه میانی دلخواه به صورت زیر به دست می‌آید:

$$\boldsymbol{\Delta w_u^{(l)}} = -\frac{\eta}{2}\boldsymbol{\nabla}_{\boldsymbol{W_u}} e^{(l)} = \eta\delta_u^{(l)}\boldsymbol{in_u^{(l)}} = \eta\left(\sum_{s\in succ\,(u)} \delta_s^{(l)} w_{su}\right)\frac{\partial out_u^{(l)}}{\partial net_u^{(l)}}\boldsymbol{in_u^{(l)}}$$

در رابطه فوق، منظور از $succ\ (u)$ تمامی نودهایی هستند که در گراف جهت‌دار، فرزند نود u محسوب می‌شوند. همچنین $\delta_u^{(l)}$ برای تک تک نورون‌های میانی به صورت بازگشتی و بر حسب $\delta_s^{(l)}$ نورون‌های لایه‌های بالاتر حساب می‌شود. و برای نورون‌های لایه آخر، $\delta_u^{(l)}$ به صورت زیر محاسبه می‌شود:

$$\delta_u^{(l)} = \left(o_u^{(l)} - out_u^{(l)}\right)\frac{\partial out_u^{(l)}}{\partial net_u^{(l)}}$$