
بسم الله الرحمن الرحيم

**دانشگاه صنعتی اصفهان**
دانشکده علوم ریاضی

# نگاهی به یادگیری عمیق از دیدگاه ریاضیات کاربردی

**پروژه کارشناسی**

**پریماه الصفی**

شماره دانشجویی :۹۹۱۸۷۴۳

استادراهنما

**دکتر رضا مختاری**

**۱۴۰۳**

# فهرست مطالب

# فهرست تصاویر

## چکیده

این گزارش به بررسی جامع روش‌ها و تکنیک‌های مورد استفاده در شبکه‌های عصبی مصنوعی و یادگیری عمیق می‌پردازد. در بخش ابتدایی، به ساختار شبکه‌های عصبی مصنوعی (ANN) و مفاهیم پایه‌ای همچون توابع فعال‌سازی، شبکه‌های عصبی چندلایه و شبکه‌های عصبی پیچشی (CNN) پرداخته شده است. همچنین، روش‌های بهینه‌سازی شامل پس‌انتشار خطا، گرادیان نزولی و روش‌های بهینه‌سازی تصادفی مانند «گرادیان نزول تصادفی» (SGD) و «حذف تصادفی» (Dropout) به منظور بهبود عملکرد و کاهش بیش‌برازش معرفی می‌شوند.در ادامه، کاربردهای عملی این تکنیک‌ها در زمینه‌های مختلف از جمله پردازش تصویر، دسته‌بندی داده‌های پیچیده، و یادگیری ویژگی‌های غیرخطی در داده‌های حجیم بررسی می‌شود. ابزارهای نرم‌افزاری نظیر MatConvNet که برای پیاده‌سازی شبکه‌های عمیق و تسریع فرایند یادگیری مورد استفاده قرار می‌گیرند، معرفی شده‌اند. گزارش همچنین به تحلیل دقیق روش‌های بهینه‌سازی پارامترها با استفاده از توابع هزینه مختلف مانند «تابع هزینه نرم بیشینه» (Softmax Loss) و «تابع هزینه درجه دوم» می‌پردازد.نتایج به‌دست‌آمده نشان می‌دهند که استفاده از شبکه‌های عصبی پیچشی برای مسائل پردازش تصویر، همراه با تکنیک‌های بهینه‌سازی مناسب، می‌تواند دقت و کارایی مدل‌ها را به‌طور قابل‌توجهی بهبود بخشد.

# فصل ۱
# مقدمه‌ای بر یادگیری عمیق و شبکه‌های عصبی مصنوعی

یادگیری عمیق [۱] شاخه‌ای از یادگیری ماشین[۲] است که با بهره‌گیری از شبکه‌های عصبی عمیق[۳] و تعداد زیادی لایه‌های مخفی، قادر به استخراج ویژگی‌ها و الگوهای پیچیده از داده‌های حجیم و غیرساخت‌یافته هست. این تکنیک به شبکه‌های عصبی امکان می‌دهد تا با تحلیل لایه به لایه داده‌ها، روابط پنهان و غیرمشخص را کشف کنند.

## ۱ - ۱ معماری و اصول پایه‌ای شبکه‌های عصبی مصنوعی

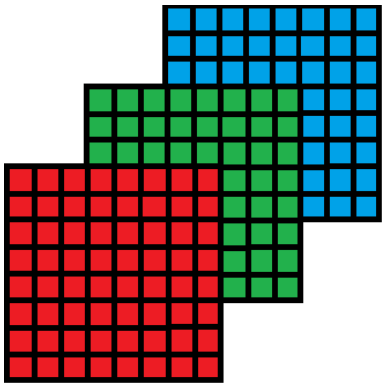
شبکه‌های عصبی مصنوعی [۴] ساختارهای محاسباتی الهام گرفته از شبکه عصبی مغز انسان هستند. این شبکه‌ها شامل مجموعه‌ای از نودها یا نورون‌ها هستند که به‌صورت لایه‌ای سازماندهی شده‌اند. هر نورون ورودی‌هایی را دریافت و با وزن‌های مشخص ترکیب می‌کند، سپس یک تابع فعال‌سازی را اعمال کرده و خروجی را به نورون‌های بعدی ارسال می‌کند.[۱]

### ۱ - ۱ - ۱ روش‌های پیشرفته دسته‌بندی تصاویر با استفاده از شبکه‌های عصبی

فرض کنید می‌خواهیم یک شبکه عصبی طراحی کنیم که تصاویر گربه را از تصاویر غیر گربه تشخیص دهد، به‌طوری‌که اگر تصویر ورودی یک گربه باشد، خروجی ۱ و در غیر این صورت ۰ باشد. تصاویر معمولاً به‌صورت

---
[1] Deep Learning
[2] Machine Learning
[3] Neural Networks
[4] Artificial Neural Networks

RGB [1] هستند۱ - ۱ و به همین دلیل به شکل بردار یا ماتریس به شبکه وارد می‌شوند.[۲]



شکل ۱ - ۱: ماتریس RGB

برای آموزش چنین شبکه‌ای، مهم‌ترین مسئله، دقت شبکه در فاز آموزشی و قابلیت تعمیم آن به داده‌های جدید است. در این فرایند، ورودی‌ها که به‌صورت ماتریس و بردار هستند، باید به اعدادی باینری (۰،۱) تبدیل شوند. برای این کار از توابع فعال‌سازی مختلفی استفاده می‌شود.

## ۱ - ۲ بررسی جامع توابع فعال‌سازی در یادگیری عمیق

یکی از مهم‌ترین توابع فعال‌سازی[2]، تابع سیگموید [3] است که به‌صورت

$$\sigma(x) = \frac{1}{1+e^{-x}}. \qquad (۱-۱)$$

این تابع را می‌توان به‌عنوان نسخه‌ای از یک تابع پله‌ای [4] در نظر گرفت. به این معنا که رفتار یک نورون در مغز را تقلید می‌کند؛ به این صورت که وقتی ورودی به‌اندازه کافی بزرگ باشد، فعال می‌شود (خروجی ۱) و در غیر این صورت غیرفعال می‌ماند (خروجی ۰).

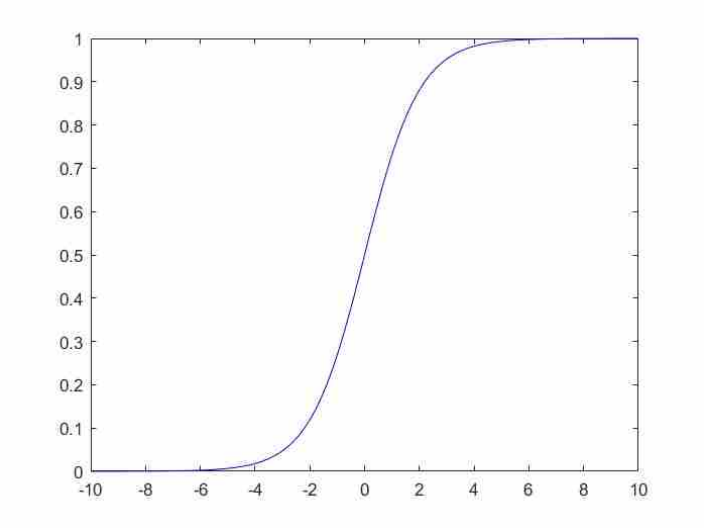
تابع سیگموید خاصیت مفیدی دارد که مشتق آن به شکل ساده‌ای به‌صورت

$$\sigma'(x) = \sigma(x)\left(1-\sigma(x)\right). \qquad (۱-۲)$$

برای ساده‌سازی بیشتر، تابع سیگموید را به‌صورت برداری تعریف می‌کنیم. اگر $\mathbf{z} \in \mathbb{R}^m$ باشد، و $\sigma : \mathbb{R}^m \to$ $\mathbb{R}^m$تابع سیگموید با اعمال به اجزای هر مؤلفه تعریف می‌شود، به‌طوری‌که

[1] Red, Green and Blue
[2] Activation Functions
[3] Sigmoid Function
[4] Step Function

$$(\sigma(\mathbf{z}))_i = \sigma(z_i). \qquad (۳-۱)$$



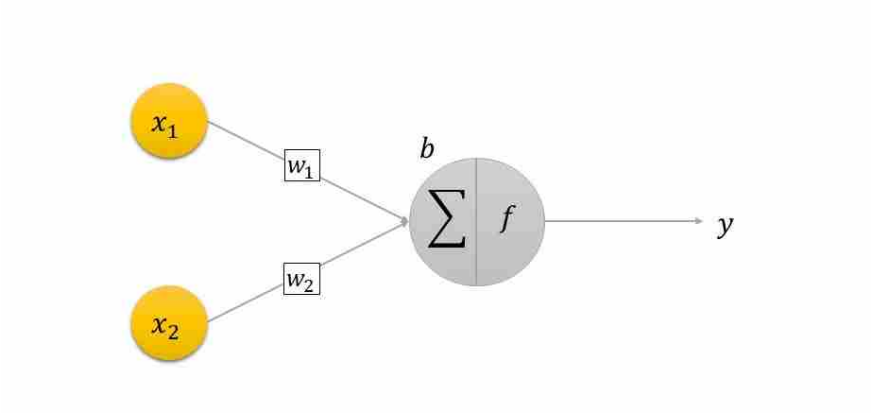

شکل ۱-۲: تابع سیگموید

## ۱-۳ تحلیل ساختار شبکه‌های عصبی و کاربردهای آن

با نمادگذاری گفته شده در قسمت قبل، می‌توانیم لایه‌هایی از نورون‌ها را ایجاد کنیم. در هر لایه، هر نورون یک عدد حقیقی تولید می‌کند که به نورون‌های لایه بعدی ارسال می‌شود. در لایه بعدی، هر نورون ترکیب وزنی مختص به خود از این مقادیر را تشکیل داده، بایاس خود را اضافه کرده و تابع سیگموید را اعمال می‌کند. اگر اعداد حقیقی تولید شده توسط نورون‌ها در یک‌لایه در یک بردار $\mathbf{a}$ جمع‌آوری شوند، بردار خروجی‌ها از لایه بعدی به‌صورت (مانند شکل ۱-۳)

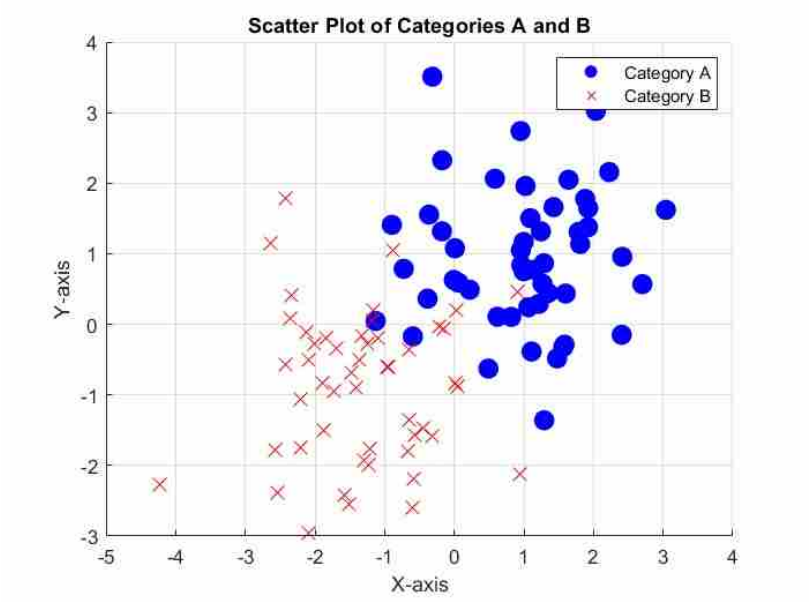
$$\sigma(\mathbf{W}\mathbf{a} + \mathbf{b}). \qquad (۴-۱)$$

در اینجا، $\mathbf{W}$ یک ماتریس و $\mathbf{b}$ یک بردار است. $\mathbf{W}$ حاوی وزن‌ها و $\mathbf{b}$ حاوی بایاس‌ها است. تعداد ستون‌های $\mathbf{W}$ با تعداد نورون‌هایی که بردار $\mathbf{a}$ را در لایه قبلی تولید کرده‌اند، مطابقت دارد. تعداد ردیف‌های $\mathbf{W}$ و مؤلفه‌های $\mathbf{b}$ با تعداد نورون‌های لایه فعلی مطابقت دارد. فرض کنید می‌خواهیم دسته‌بندی روی یک مجموعه‌داده انجام دهیم. به‌عنوان‌مثال، مجموعه‌داده درباره دو منطقه حفاری است که به دودسته مناطق $\mathrm{A}$ و



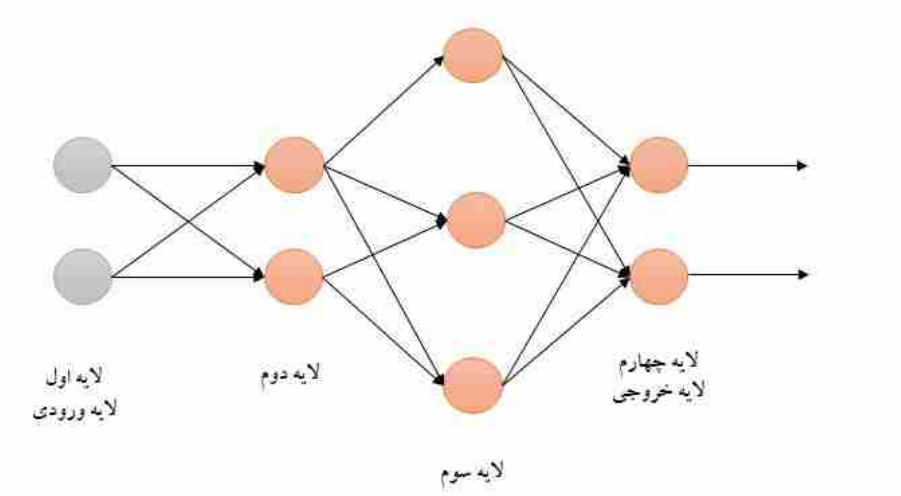

شکل ۱-۳: معماری شبکه‌ـساختار نورون

مناطق B دسته‌بندی شده‌اند. پس از برازش، خروجی به‌صورت نمودار نشان داده می‌شود، به‌طوری‌که مناطق A با دایره و مناطق B با ضربدر مشخص شده‌اند.




شکل ۱-۴: نقاط در فضای $\mathbb{R}^2$. دایره‌ها نشان‌دهنده نقاط دسته A و ضربدرها نشان‌دهنده نقاط دسته B هستند.

شکل ۱-۵ یک شبکه عصبی مصنوعی با چهارلایه را نشان می‌دهد. این شکل از شبکه را برای مسئله‌ای که در شکل ۱-۴ تعریف شده است، به کار خواهیم گرفت. در شبکه نشان‌داده‌شده در شکل ۱-۵، لایه اول (ورودی) با دو دایره نشان‌داده‌شده است، زیرا نقاط داده ورودی ما دارای دو مؤلفه هستند. لایه دوم دارای دو دایره توپر است که نشان می‌دهد دو نورون در حال استفاده هستند. پیکان‌های بین لایه ۱ و لایه ۲ نشان می‌دهند که هر دو مؤلفه داده‌های ورودی به دو نورون در لایه ۲ منتقل می‌شوند. ازآنجاکه داده‌های ورودی به‌صورت $x \in \mathbb{R}^2$ است، وزن‌ها و بایاس‌های لایه ۲ به ترتیب با ماتریس $\mathbf{W}^{[2]} \in \mathbb{R}^{2\times 2}$ و بردار $\mathbf{b}^{[2]} \in \mathbb{R}^2$ نشان داده




شکل ۱-۵: نمایی از یک شبکه عصبی با چهار لایه

می‌شوند. خروجی از لایه ۲ به صورت

$$\sigma(\mathbf{W}^{[2]}\mathbf{x}+\mathbf{b}^{[2]}) \in \mathbb{R}^2, \tag{۱-۵}$$

لایه ۳ دارای سه نورون است که هر کدام ورودی در $\mathbb{R}^2$ دریافت می‌کنند، بنابراین وزن‌ها و بایاس‌های لایه ۳ به ترتیب با ماتریس $\mathbf{W}^{[3]} \in \mathbb{R}^{3\times 2}$ و بردار $\mathbf{b}^{[3]} \in \mathbb{R}^3$ نشان داده می‌شوند. خروجی از لایه ۳ به صورت

$$\sigma(\mathbf{W}^{[3]}\sigma(\mathbf{W}^{[2]}\mathbf{x}+\mathbf{b}^{[2]})+\mathbf{b}^{[3]}) \in \mathbb{R}^3, \tag{۱-۶}$$

لایه چهارم (خروجی) دارای دو نورون است که هر کدام ورودی در $\mathbb{R}^3$ دریافت می‌کنند، بنابراین وزن‌ها و بایاس‌های این لایه به ترتیب با ماتریس $\mathbf{W}^{[4]} \in \mathbb{R}^{2\times 3}$ و بردار $\mathbf{b}^{[4]} \in \mathbb{R}^2$ نشان داده می‌شوند. خروجی از لایه ۴، و در نتیجه از کل شبکه، به صورت

$$F(\mathbf{x}) = \sigma(\mathbf{W}^{[4]}\sigma(\mathbf{W}^{[3]}\sigma(\mathbf{W}^{[2]}\mathbf{x}+\mathbf{b}^{[2]})+\mathbf{b}^{[3]})+\mathbf{b}^{[4]}) \in \mathbb{R}^2. \tag{۱-۷}$$

عبارت ۱-۷ تابع $F : \mathbb{R}^2 \to \mathbb{R}^2$ را بر اساس ۲۳ پارامترش تعریف می‌کند. ما نیاز داریم که $F(x)$ برای نقاط داده در دسته A به $[\mathbf{1},\mathbf{0}]^T$ و برای نقاط داده در دسته B به $[\mathbf{0},\mathbf{1}]^T$ نزدیک باشد. سپس، با داشتن یک نقطه جدید $x \in \mathbb{R}^2$، منطقی است که آن را بر اساس بزرگ‌ترین مؤلفه $F(x)$ دسته‌بندی کنیم؛ به این صورت که اگر $F_1(x) > F_2(x)$ باشد، دسته A و اگر $F_1(x) < F_2(x)$ باشد، دسته تعلق،B دارد.

### ۱ - ۴ تابع هزینه

نیازمندی بیان شده در $F$ می‌تواند از طریق یک تابع هزینه مشخص شود. با نشان‌دادن ده نقطه داده در شکل ۱ - ۴ با $\{x^{(i)}\}_{i=1}^{10}$، از $y(x^{(i)})$ برای خروجی هدف استفاده می‌کنیم؛ به این صورت که:

$$y(x^{(i)}) = \begin{cases} [\mathbf{1},\mathbf{0}]^T & \text{اگر } x^{(i)} \text{ متعلق به دسته A باشد} \\ [\mathbf{0},\mathbf{1}]^T & \text{اگر } x^{(i)} \text{ متعلق به دسته B باشد} \end{cases}, \tag{۱ - ۸}$$

بنابراین تابع هزینه به صورت

$$\text{cost}(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}) = \frac{1}{10}\sum_{i=1}^{10}\frac{1}{2}\|y(x^{(i)}) - F(x^{(i)})\|_2^2. \tag{۱ - ۹}$$

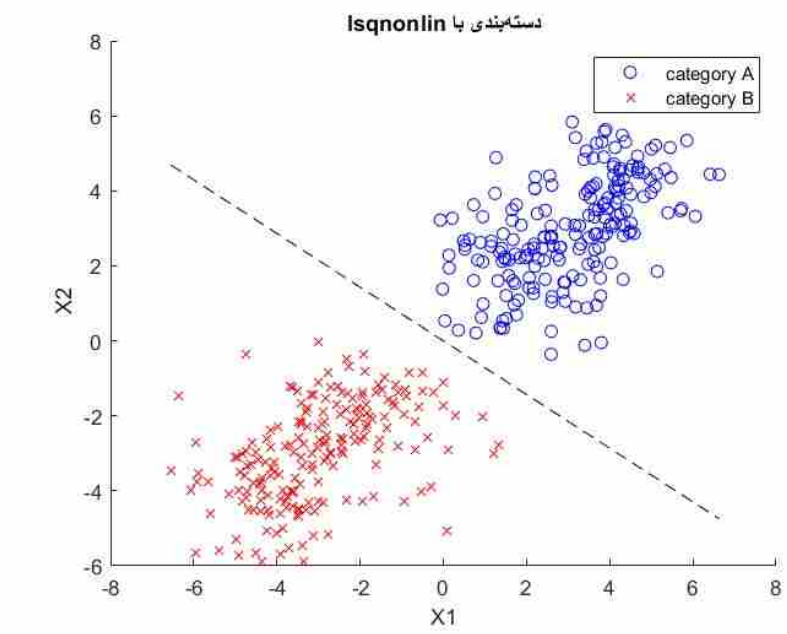
در اینجا، ضریب $\frac{1}{2}$ برای راحتی لحاظ شده است؛ زمانی که شروع به مشتق‌گیری می‌کنیم، این امر مسائل را ساده‌تر می‌کند. تأکید می‌کنیم که تابع هزینه، تابعی از وزن‌ها و بایاس‌ها است .

#### آموزش شبکه

انتخاب وزن‌ها و بایاس‌ها به‌گونه‌ای که تابع هزینه را به حداقل برساند، به‌عنوان آموزش شبکه شناخته می‌شود. توجه داریم که در اصل، تغییر مقیاس یک تابع هدف، مسئله بهینه‌سازی را تغییر نمی‌دهد، به این معنی که اگر تابع هزینه را به‌عنوان‌مثال به $100 \cdot \text{cost}$ یا $\text{cost}/30$ تغییر دهیم، باید به همان نقطه کمینه برسیم، بنابراین ضرایب $\frac{1}{10}$ و $\frac{1}{2}$ در ۱ - ۹ نباید بر نتیجه تأثیری داشته باشند.

#### بهینه‌سازی پارامترها

برای تحلیل داده‌های موجود در شکل ۱ - ۴، ما از جعبه‌ابزار بهینه‌سازی MATLAB برای بهینه‌سازی تابع هزینه ۱ - ۹ استفاده کردیم که این تابع به ۲۳ پارامتر مختلف شامل ماتریس‌های وزن $W^{[2]}, W^{[3]}, W^{[4]}$ و بردارهای بایاس $b^{[2]}, b^{[3]}, b^{[4]}$ وابسته است. به‌طور خاص، از الگوریتم «`lsqnonlin`» که یک روش برای حل مسائل بهینه‌سازی با کمینه کردن خطاهای مربعی به‌طور غیرخطی است، بهره بردیم. در نتیجه، شبکه آموزش‌دیده، که در شکل ۱ - ۶ نشان داده شده است، مرز تفکیکی را ایجاد می‌کند به طوری که $F_1(x) > F_2(x)$ برای دسته A و $F_1(x) < F_2(x)$ برای دسته B.




شکل ۱ - ۶: خروجی شبکه عصبی بر روی داده های شکل۱ - ۴

# فصل ۲
# روش‌های بهینه‌سازی و بهبود عملکرد شبکه‌های عصبی

## ۲-۱ بهبود عملکرد شبکه‌های عصبی چند لایه با تکنیک‌های بهینه‌سازی

مثال چهارلایه‌ای در فصل ۱ مفهوم نورون‌ها را معرفی کرد، که با استفاده از تابع سیگموید نشان داده می‌شوند و در لایه‌های مختلف عمل می‌کنند. در یک لایه عمومی، هر نورون ورودی یکسانی دریافت می‌کند و یک مقدار حقیقی تولید می‌کند که به هر نورون در لایه بعدی ارسال می‌شود. دو لایه استثنایی وجود دارد. در لایه ورودی، «لایه قبلی» وجود ندارد و هر نورون بردار ورودی را دریافت می‌کند. در لایه خروجی، «لایه بعدی» وجود ندارد و این نورون‌ها خروجی کلی را فراهم می‌کنند. لایه‌های بین این دو لایه، «لایه‌های مخفی» نامیده می‌شوند که محاسبات میانی را انجام می‌دهند. یادگیری عمیق به معنای استفاده از تعداد زیادی لایه‌های مخفی است.

### ۲-۱-۱ نمادگذاری

در این بخش، فرم کلی نمادگذاری را به‌تفصیل بیان می‌کنیم. فرض می‌کنیم شبکه دارای $L$ لایه است، به‌طوری‌که لایه‌های ۱ و $L$ به ترتیب لایه‌های ورودی و خروجی هستند. فرض کنید لایه $l$، برای $l = 1, 2, 3, \ldots, L$، شامل $n_l$ نورون است. بنابراین $n_1$ بُعد داده‌های ورودی است و به‌طور کلی، شبکه از $R^{n_1}$ به $R^{n_L}$ نگاشت می‌کند.

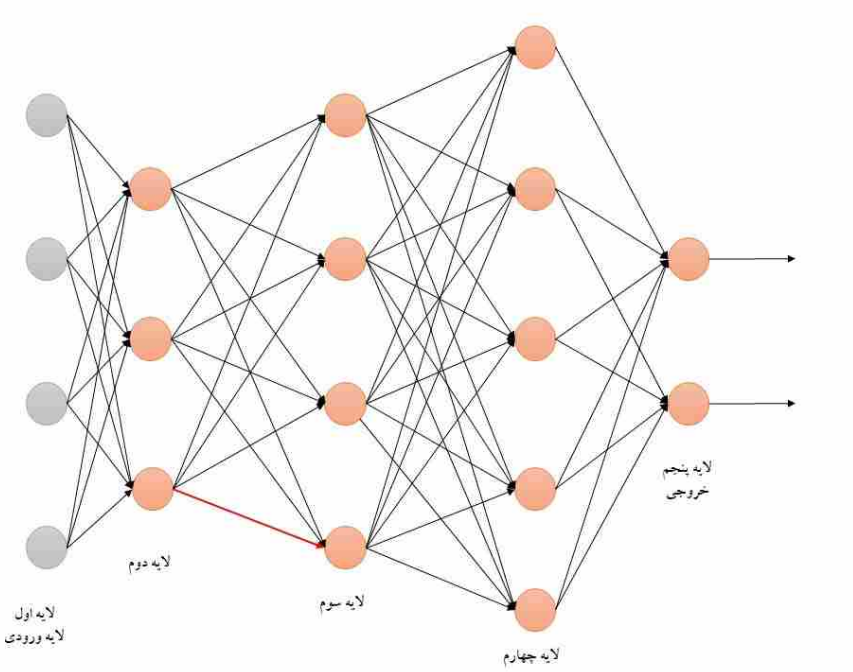
ما از $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ برای نشان‌دادن ماتریس وزن‌ها در لایه $l$ استفاده می‌کنیم. به طور دقیق‌تر، $w_{jk}^{[l]}$ وزنی است که نورون $j$ در لایه $l$ به خروجی نورون $k$ در لایه $l-1$ اعمال می‌کند. به همین ترتیب، $b^{[l]} \in \mathbb{R}^{n_l}$ بردار بایاس‌ها برای لایه $l$ است، به‌طوری‌که نورون $j$ در لایه $l$ از بایاس $b_j^{[l]}$ استفاده می‌کند.

در شکل ۲-۱، مثالی با $L = 5$ لایه ارائه می‌دهیم. در این مثال، $n_1 = 4, n_2 = 3, n_3 = 4, n_4 = 5, n_5 = 2$

هستند و

$$W^{[2]} \in \mathbb{R}^{3\times 4}, \quad W^{[3]} \in \mathbb{R}^{4\times 3}, W^{[4]} \quad \in \mathbb{R}^{5\times 4}, \quad W^{[5]} \in \mathbb{R}^{2\times 5}, b^{[2]} \in \mathbb{R}^{3}, \quad b^{[3]} \in \mathbb{R}^{4},$$

$$b^{[4]} \in \mathbb{R}^{5}, \quad b^{[5]} \in \mathbb{R}^{2}.$$




شکل ۲-۱: نمایی از یک شبکه عصبی با پنج لایه

با داشتن یک ورودی $\mathbf{x} \in \mathbb{R}^{n_1}$، می‌توانیم به طور مرتب عملکرد شبکه را باتوجه‌به اینکه $a_j^{[l]}$ خروجی یا فعال‌سازی نورون $j$ در لایه $l$ را نشان می‌دهد، خلاصه کنیم؛ بنابراین

$$a^{[1]} = \mathbf{x} \in \mathbb{R}^{n_1}, \tag{۲-۱}$$

$$a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l} \quad \text{for} \quad l = 2, 3, \ldots, L. \tag{۲-۲}$$

روابط فوق، نحوه عملکرد الگوریتم را در پردازش ورودی از طریق شبکه به تولید یک خروجی $a^{[L]} \in \mathbb{R}^{n_L}$ نشان می‌دهند.

### ۲-۱-۲ استراتژی‌های بهینه‌سازی و کاهش خطا در شبکه‌های عصبی

فرض کنید که $N$ داده یا نقاط آموزشی در $R^{n_1}$ داریم، یعنی $\{x^{(i)}\}_{i=1}^{N}$ که برای آن‌ها خروجی‌های هدف $\{y(x^{(i)})\}_{i=1}^{N}$ در $R^{n_L}$ داده شده‌اند. با تعمیم رابطه ۱-۹، تابع هزینه به شکل زیر است:

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|y(x^{(i)}) - a^{[L]}(x^{(i)})\|_2^2. \qquad (۲\text{-}۳)$$

### ۲-۱-۳ تکنیک‌های بهینه‌سازی با استفاده از روش گرادیان تصادفی

به طور کلی وزن‌ها و بایاس‌ها به‌صورت ماتریس‌ها و بردارها هستند، اما در این مرحله راحت‌تر است که تصور کنیم آن‌ها به‌عنوان یک بردار واحد ذخیره می‌شوند که آن را $\mathbf{p}$ می‌نامیم. مثال شکل ۱-۵ شامل مجموعاً ۲۳ وزن و بایاس است. بنابراین، در آن حالت، $\mathbf{p} \in \mathbb{R}^{23}$ است. به‌طور کلی، فرض خواهیم کرد که $\mathbf{p} \in \mathbb{R}^s$ و تابع هزینه در ۲-۳ را به صورت $\text{Cost}(\mathbf{p})$ می‌نویسیم بنابراین $\text{Cost} : \mathbb{R}^s \to \mathbb{R}$.

اکنون یک روش کلاسیک در بهینه‌سازی معرفی می‌کنیم که اغلب به‌عنوان نزول گرادیان [1] شناخته می‌شود. این روش به‌صورت تکراری عمل می‌کند و دنباله‌ای از بردارها در $\mathbb{R}^s$ را محاسبه می‌کند، باهدف همگرا شدن به یک بردار که تابع هزینه را به حداقل می‌رساند. فرض کنید که بردار فعلی ما $\mathbf{p}$ است؛ چگونه باید یک تغییر، $\Delta\mathbf{p}$، انتخاب کنیم به‌طوری‌که بردار بعدی، $\mathbf{p}+\Delta\mathbf{p}$، بهبود را نشان دهد؟ اگر $\Delta\mathbf{p}$ کوچک باشد، با نادیده‌گیری جملات مرتبه $\|\Delta\mathbf{p}\|^2$، بسط سری تیلور به‌صورت

$$\text{Cost}(\mathbf{p}+\Delta\mathbf{p}) \approx \text{Cost}(\mathbf{p}) + \sum_{r=1}^{s}\frac{\partial \text{Cost}(\mathbf{p})}{\partial p_r}\Delta p_r. \qquad (۲\text{-}۴)$$

در اینجا $\frac{\partial \text{Cost}(\mathbf{p})}{\partial p_r}$ مشتق جزئی تابع هزینه نسبت به $r$امین پارامتر را نشان می‌دهد. برای راحتی، اجازه می‌دهیم که $\nabla\text{Cost}(\mathbf{p}) \in \mathbb{R}^s$ بردار مشتقات جزئی را نشان دهد که به‌عنوان گرادیان شناخته می‌شود، به‌طوری‌که

$$(\nabla\text{Cost}(\mathbf{p}))_r = \frac{\partial \text{Cost}(\mathbf{p})}{\partial p_r},$$

سپس معادله ۲-۴ به شکل زیر تبدیل می‌شود

$$\text{Cost}(\mathbf{p}+\Delta\mathbf{p}) \approx \text{Cost}(\mathbf{p}) + \nabla\text{Cost}(\mathbf{p})^T\Delta\mathbf{p}. \qquad (۲\text{-}۵)$$

هدف ما کاهش مقدار تابع هزینه است. رابطه ۲-۵ ایده انتخاب $\Delta\mathbf{p}$ را برای منفی کردن $\nabla\text{Cost}(\mathbf{p})^T\Delta\mathbf{p}$ تا حد ممکن انگیزه می‌دهد. ما می‌توانیم این مسئله را با استفاده از نابرابری کوشی - شوارتز حل کنیم که

[1] Gradient descent

بیان می‌کند برای هر $\mathbf{f}, \mathbf{g} \in \mathbb{R}^s$، داریم $|\mathbf{f}^T\mathbf{g}| \leq \|\mathbf{f}\|_2\|\mathbf{g}\|_2$؛ بنابراین منفی‌ترین مقداری که $\mathbf{f}^T\mathbf{g}$ می‌تواند باشد، $-\|\mathbf{f}\|_2\|\mathbf{g}\|_2$ است که زمانی رخ می‌دهد که $\mathbf{f} = -\mathbf{g}$؛ بنابراین، بر اساس ۲-۵ ، باید $\Delta\mathbf{p}$ را در جهت $-\nabla\text{Cost}(\mathbf{p})$ انتخاب کنیم. با درنظرداشتن اینکه ۲-۵ یک تقریب است که تنها برای $\Delta\mathbf{p}$ کوچک مرتبط است، ما خود را به یک‌قدم کوچک در آن جهت محدود خواهیم کرد. این منجر به به‌روزرسانی زیر می‌شود

$$\mathbf{p} \to \mathbf{p} - \eta\nabla\text{Cost}(\mathbf{p}). \tag{۲-۶}$$

اینجا $\eta$ یک اندازه گام کوچک است که در این زمینه به‌عنوان نرخ یادگیری [1] شناخته می‌شود. این معادله روش نزول تندترین [2] را تعریف می‌کند. ما یک بردار اولیه انتخاب می‌کنیم و با استفاده از ۲-۶ تکرار می‌کنیم تا زمانی که یک معیار توقف برآورده شود یا تعداد تکرارها از بودجه محاسباتی فراتر رود.

تابع هزینه ما ۲-۳ شامل مجموعی از جملات فردی است که بر روی داده‌های آموزشی اجرا می‌شود؛ بنابراین مشتق جزئی $\nabla\text{Cost}(\mathbf{p})$ جمعی از مشتقات جزئی فردی بر روی داده‌های آموزشی است.

$$C_{x^{(i)}} = \frac{1}{2}\|y(x^{(i)}) - a^{[L]}(x^{(i)})\|_2^2, \tag{۲-۷}$$

سپس، از ۲-۳،

$$\nabla\text{Cost}(\mathbf{p}) = \frac{1}{N}\sum_{i=1}^{N}\nabla C_{x^{(i)}}(\mathbf{p}). \tag{۲-۸}$$

هنگامی که تعداد زیادی پارامتر و نقاط آموزشی داریم، محاسبه بردار گرادیان ۲-۸ در هر تکرار از روش نزول گرادیان ۲-۶ می‌تواند به‌شدت هزینه‌بر باشد. جایگزین بسیار ارزان‌تری این است که میانگین گرادیان‌های فردی را بر روی تمام نقاط آموزشی با گرادیان در یک نقطه آموزشی تصادفی انتخاب‌شده جایگزین کنیم. این منجر به ساده‌ترین شکل از روشی می‌شود که به آن روش گرادیان تصادفی [3] می‌گویند که یک گام از آن را به‌صورت زیر خلاصه می‌شود:

۱. یک عدد صحیح $i$ را به صورت تصادفی و یکنواخت از مجموعه $\{1, 2, 3, \ldots, N\}$ انتخاب کنید.

۲. به‌روزرسانی کنید

$$\mathbf{p} \to \mathbf{p} - \eta\nabla C_{x^{(i)}}(\mathbf{p}). \tag{۲-۹}$$

[1] Learning rate
[2] Steepest descent
[3] Stochastic gradient

به زبان ساده، در هر مرحله از روش گرادیان تصادفی، یک نقطه آموزشی از میان تمامی داده‌ها به‌صورت تصادفی انتخاب می‌شود و از آن به‌عنوان نمایندهٔ کل مجموعه‌داده‌ها استفاده می‌شود. با پیشرفت هر مرحله، مدل داده‌های بیشتری را مشاهده می‌کند، به همین دلیل انتظار می‌رود که این روش، حتی اگر در هر مرحله کاهش هزینه زیادی به همراه نداشته باشد، در طول زمان به‌طورکلی مؤثر باشد.

نکته مهم این است که حتی با انتخاب یک مقدار بسیار کوچک برای نرخ یادگیری ($\eta$)، تضمینی وجود ندارد که تابع هزینهٔ کلی کاهش یابد زیرا به‌جای میانگین‌گیری از تمام داده‌ها، از یک نمونه استفاده شده است؛ بنابراین، باوجوداینکه عبارت «نزول گرادیان تصادفی» به طور گسترده‌ای استفاده می‌شود، ما ترجیح می‌دهیم از عبارت «گرادیان تصادفی» استفاده کنیم تا دقت بیشتری در بیان داشته باشیم.

نسخه‌ای از روش گرادیان تصادفی که در رابطه ۲ - ۹ معرفی کردیم، ساده‌ترین شکل از یک طیف گسترده از روش‌های موجود است. در این روش، شاخص $i$ با استفاده از نمونه‌گیری با جایگزینی انتخاب می‌شود—به این معنا که پس از استفاده از یک نقطه آموزشی، این نقطه به مجموعه آموزشی بازگردانده می‌شود و در گام بعدی به‌اندازه هر نقطه دیگر احتمال انتخاب‌شدن دارد.

یکی از جایگزین‌های موجود، نمونه‌گیری بدون جایگزینی است که در آن، نقاط آموزشی به ترتیب تصادفی و بدون تکرار انتخاب می‌شوند. انجام $N$ گام به این شیوه، به‌عنوان تکمیل یک "دوره" [1] شناخته می‌شود و می‌تواند به‌صورت زیر خلاصه شود:

۱. اعداد $\{1, 2, 3, \ldots, N\}$ را به ترتیب جدیدی $\{k_1, k_2, k_3, \ldots, k_N\}$ مرتب کنید.

۲. برای $i = 1$ تا $N$، به‌روزرسانی کنید

$$p \to p - \eta \nabla C_{x_{\{k_i\}}}(p). \qquad (۱۰\text{-}۲)$$

اگر روش گرادیان تصادفی را به‌عنوان تخمینی از میانگین روی تمام نقاط آموزشی در رابطه ۲ - ۸ با یک نمونه تکی در نظر بگیریم، منطقی است که یک راه میانه را نیز در نظر بگیریم که در آن از میانگین یک نمونه کوچک استفاده شود. برای برخی $m \ll N$ می‌توانیم گام‌هایی از فرم زیر برداریم:

۱. $m$ عدد صحیح $\{k_1, k_2, k_3, \ldots, k_m\}$ را به‌صورت تصادفی و یکنواخت از $\{1, 2, 3, \ldots, N\}$ انتخاب کنید.

۲. به‌روزرسانی کنید

$$p \to p - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla C_{x_{\{k_i\}}}(p). \qquad (۱۱\text{-}۲)$$

[1] epoch

در این روش، مجموعه‌ای از داده‌ها به نام $\{x^{(k_i)}\}_{i=1}^{m}$ به‌عنوان یک minibatch یا دسته کوچک شناخته می‌شود. یک روش دیگر، بدون جایگزینی داده‌ها، این است که فرض کنیم تعداد کل داده‌ها $N$ برابر با $Km$ است که در آن $k$ تعداد دسته‌های کوچک [۱] است. در این روش، مجموعه‌داده‌ها به‌صورت تصادفی به $k$ دسته کوچک مجزا تقسیم می‌شود و سپس به ترتیب بین این دسته‌ها حرکت می‌شود.

ازآنجاکه روش گرادیان تصادفی معمولاً برای محاسبات بسیار بزرگ استفاده می‌شود، انتخاب‌های الگوریتمی مانند اندازه دسته‌های کوچک و نحوه تصادفی‌سازی داده‌ها، بر اساس نیازهای سیستم‌های محاسباتی با عملکرد بالا تعیین می‌شود. همچنین، این امکان وجود دارد که در طول آموزش، اندازه دسته‌ها و دیگر پارامترها مانند نرخ یادگیری را به‌صورت دینامیکی تغییر دهیم تا فرایند همگرایی سریع‌تر انجام شود. در واقع، داشتن یک الگوریتم مناسب برای تعیین نرخ یادگیری، یکی از بخش‌های کلیدی برای اجرای مؤثر محاسبات است.

## ۲-۲ روش پس‌انتشار

اکنون در موقعیتی هستیم که می‌توانیم روش گرادیان تصادفی را برای آموزش یک شبکه عصبی مصنوعی به کار بگیریم. بردار عمومی پارامترها، $\mathbf{p}$ که در بخش قبل استفاده شد، اکنون به ورودی‌های ماتریس‌های وزن و بردارهای بایاس تغییر می‌کند. وظیفه ما محاسبه مشتقات جزئی تابع هزینه نسبت به هر $w_{jk}^{[l]}$ و $b_j^{[l]}$ است.

روش گرادیان تصادفی از ساختار تابع هزینه بهره می‌برد: زیرا ۲-۳ ترکیبی خطی از عبارات فردی است که بر روی داده‌های آموزشی اجرا می‌شود، همین امر در مورد مشتقات جزئی آن نیز صدق می‌کند؛ بنابراین توجه خود را بر محاسبه آن مشتقات جزئی فردی متمرکز می‌کنیم.

برای یک نقطه آموزشی ثابت، $C_{x^{(i)}}$ در ۲-۷ به‌عنوان تابعی از وزن‌ها و بایاس‌ها در نظر گرفته می‌شود؛ بنابراین می‌توانیم وابستگی به $x^{(i)}$ را حذف کرده و به‌سادگی بنویسیم

$$C = \frac{1}{2}\|\mathbf{y} - \mathbf{a}^{[L]}\|_2^2. \qquad (۱۲\text{-}۲)$$

به یاد می‌آوریم از ۲-۲ که $\mathbf{a}^{[L]}$ خروجی شبکه عصبی مصنوعی است. وابستگی $C$ به وزن‌ها و بایاس‌ها تنها از طریق $\mathbf{a}^{[L]}$ حاصل می‌شود. برای استخراج عبارات مفید برای مشتقات جزئی، معرفی دو مجموعه متغیر دیگر سودمند است. ابتدا داریم

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \in \mathbb{R}^{n_1} \quad \text{for} \quad l = 2, 3, \ldots, L, \qquad (۱۳\text{-}۲)$$

[1] Minibatch

$z_j^{[l]}$ به عنوان ورودی وزن‌دار برای نورون $j$ در لایه $l$ اشاره می‌کند. رابطه بنیادی ۲ - ۲ که اطلاعات را در شبکه منتشر می‌کند، سپس

$$\mathbf{a}^{[l]} = \sigma(\mathbf{z}^{[l]}) \quad \text{for} \quad l = 2, 3, \ldots, L, \tag{۱۴-۲}$$

آنگاه، $\delta_j^{[l]} \in \mathbb{R}^{n_1}$ را به‌صورت

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} \quad \text{for} \quad 1 \leq j \leq n_l \quad \text{and} \quad 2 \leq l \leq L. \tag{۱۵-۲}$$

عبارت بالا که اغلب به‌عنوان «خطای نورون $j$» در لایه $l$ نامیده می‌شود، یک کمیت میانی است که برای تحلیل و محاسبات به‌کار می‌رود. با این حال، باید توجه داشت که استفاده از اصطلاح «خطا» در اینجا ممکن است کمی گمراه‌کننده باشد. در یک لایه میانی، مشخص نیست که تا چه حد می‌توان هر نورون را برای اختلافات در خروجی نهایی مسئول دانست. همچنین، در لایه خروجی $L$، عبارت $\delta_j^{[l]}$ مستقیماً این اختلافات را اندازه‌گیری نمی‌کند. دلیل اینکه به $\delta_j^{[l]}$ در اینجا به‌عنوان «خطا» اشاره می‌شود، این است که تابع هزینه فقط زمانی می‌تواند به حداقل برسد که تمام مشتقات جزئی آن صفر باشند؛ بنابراین، داشتن $\delta_j^{[l]} = 0$ می‌تواند هدف مناسبی باشد. به طور کلی می‌توان گفت که $\delta_j^{[l]}$ در واقع حساسیت تابع هزینه نسبت به ورودی وزن‌دار نورون $j$ در لایه $l$ را اندازه‌گیری می‌کند.

در این مرحله، ما همچنین نیاز داریم که ضرب هادامار [1] یا ضرب مؤلفه به مؤلفه دو بردار [2] را تعریف کنیم. اگر $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$، آنگاه $\mathbf{x} \circ \mathbf{y} \in \mathbb{R}^n$ به‌صورت $(\mathbf{x} \circ \mathbf{y})_i = x_i y_i$ تعریف می‌شود. به زبان ساده، ضرب هادامار با ضرب جفت به جفت مؤلفه‌های متناظر تشکیل می‌شود.

**لم ۱.۲**: داریم

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (\mathbf{a}^{[L]} - \mathbf{y}), \tag{۱۶-۲}$$

$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]} \quad \text{for} \quad 2 \leq l \leq L-1, \tag{۱۷-۲}$$

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]} \quad \text{for} \quad 2 \leq l \leq L, \tag{۱۸-۲}$$

[1] Hadamard
[2] componentwise

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]} \quad \text{for} \quad 2 \le l \le L. \tag{۱۹-۲}$$

**اثبات**: ابتدا با اثبات رابطهٔ ۲-۱۶ شروع می‌کنیم. رابطهٔ ۲-۱۴ با $l = L$ نشان می‌دهد که $z_j^{[L]}$ و $a_j^{[L]}$ توسط $a^{[L]} = \sigma(z^{[L]})$ به هم متصل هستند و از این رو

$$\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \sigma'(z_j^{[L]}),$$

همچنین، از رابطهٔ ۲-۱۲ داریم

$$\frac{\partial C}{\partial a_j^{[L]}} = \frac{\partial}{\partial a_j^{[L]}} \frac{1}{2} \sum_{k=1}^{n_L} \left(y_k - a_k^{[L]}\right)^2 = -(y_j - a_j^{[L]}),$$

بنابراین، با استفاده از قاعدهٔ زنجیره‌ای

$$\delta_j^{[L]} = \frac{\partial C}{\partial z_j^{[L]}} = \frac{\partial C}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = (a_j^{[L]} - y_j)\sigma'(z_j^{[L]}),$$

که این شکل جزئی از رابطهٔ ۲-۱۶ است. برای نشان‌دادن رابطهٔ ۲-۱۷، از قاعدهٔ زنجیره‌ای استفاده می‌کنیم تا از $z_j^{[l]}$ به $\{z_k^{[l+1]}\}_{k=1}^{n_{l+1}}$ تبدیل کنیم. با اعمال قاعدهٔ زنجیره‌ای و استفاده از تعریف ۲-۱۵ داریم

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}}. \tag{۲۰-۲}$$

حال، از رابطهٔ ۲-۱۳ می‌دانیم که $z_k^{[l+1]}$ و $z_j^{[l]}$ به‌صورت زیر به هم متصل هستند

$$z_k^{[l+1]} = \sum_{s=1}^{n_l} w_{ks}^{[l+1]} \sigma(z_s^{[l]}) + b_k^{[l+1]},$$

بنابراین،

$$\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = w_{kj}^{[l+1]} \sigma'(z_j^{[l]}),$$

در رابطه ۲-۲۰ داریم

$$\delta_j^{[l]} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} w_{kj}^{[l+1]} \sigma'(z_j^{[l]}),$$

که می‌توان آن را به‌صورت زیر بازنویسی کرد

$$\delta_j^{[l]} = \sigma'(z_j^{[l]}) \left((\mathbf{W}^{[l+1]})^T \delta^{[l+1]}\right)_j.$$

این شکل جزئی از رابطهٔ ۲-۱۷ است. برای نشان دادن رابطهٔ ۲-۱۸، توجه داریم که از روابط ۲-۱۳ و ۲-۱۴ می‌توان فهمید که $z_j^{[l]}$ به $b_j^{[l]}$ به‌صورت زیر متصل است

$$z_j^{[l]} = (\mathbf{W}^{[l]}\sigma(\mathbf{z}^{[l-1]}))_j + b_j^{[l]},$$

چون $\mathbf{z}^{[l-1]}$ به $b_j^{[l]}$ وابسته نیست،

$$\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = 1,$$

سپس با استفاده از قاعده‌ی زنجیره‌ای،

$$\frac{\partial C}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}}\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} = \delta_j^{[l]},$$

با استفاده از تعریف ۲-۱۵. این نتیجه رابطهٔ ۲-۱۸ را به دست می‌دهد. در نهایت، برای به‌دست‌آوردن رابطهٔ ۲-۱۹، با نسخهٔ جزءبه‌جزء رابطهٔ ۲-۱۵ شروع می‌کنیم:

$$z_j^{[l]} = \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]},$$

که نتیجه می‌دهد

$$\frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = a_k^{[l-1]}, \tag{۲-۲۱}$$

که به صورت مستقل از $j$ است. و

$$\frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = 0 \quad s \neq j. \tag{۲-۲۲}$$

به زبان ساده، روابط ۲-۲۱ و ۲-۲۲ به این دلیل حاصل می‌شوند که نورون $j$ـام در لایه $l$ فقط از وزن‌های ردیف $j$ـام ماتریس $\mathbf{W}^{[l]}$ استفاده می‌کند و این وزن‌ها را به‌صورت خطی اعمال می‌کند. سپس، با استفاده از قاعدهٔ زنجیره‌ای، روابط ۲-۲۱ و ۲-۲۲ به‌صورت

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \sum_{s=1}^{n_l} \frac{\partial C}{\partial z_s^{[l]}}\frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}}\frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} a_k^{[l-1]} = \delta_j^{[l]} a_k^{[l-1]},$$

که در مرحلهٔ آخر از تعریف $\delta_j^{[l]}$ در رابطهٔ ۲-۱۵ استفاده شده است. □

بسیاری از جنبه‌های لم ۱.۲ نیاز به توجه دارند. از روابط ۲-۱، ۲-۱۳ و ۲-۱۴ می‌توان یادآوری کرد که خروجی $\mathbf{a}^{[L]}$ را می‌توان از طریق یک عبور رو به جلو از شبکه محاسبه کرد، به ترتیب محاسبه‌ی

$$\mathbf{a}^{[1]}, \mathbf{z}^{[2]}, \mathbf{a}^{[2]}, \mathbf{z}^{[3]}, \ldots, \mathbf{a}^{[L]}.$$

پس از انجام این کار، از رابطه‌ی ۲-۱۶ می‌بینیم که $\delta^{[L]}$ به طور مستقیم در دسترس است. سپس، با استفاده از رابطه‌ی ۲-۱۷، $\delta^{[l-1]}, \delta^{[l-2]}, \ldots, \delta^{[2]}$ می‌تواند در عبور رو به عقب محاسبه شود. از روابط ۲-۱۸ و ۲-۱۹ می‌توانیم به مشتقات جزئی دسترسی پیدا کنیم. محاسبه‌ی گرادیان‌ها به این روش به نام **پس‌انتشار** [1] شناخته می‌شود.

**درک عمیق‌تر روابط پیش‌انتشار و مشتقات جزئی**

برای درک بیشتر فرمول‌های پیش‌انتشار ۲-۱۸ و ۲-۱۹ در لم ۱.۲، مفید است که تعریف بنیادی مشتق جزئی را به یاد بیاوریم. مشتق جزئی $\frac{\partial C}{\partial w_{jk}^{[l]}}$ نشان می‌دهد که چگونه تابع هزینه $C$ با تغییرات کوچک در وزن $w_{jk}^{[l]}$ تغییر می‌کند. به عنوان مثال، شکل۲-۱ وزن $w_{43}^{[3]}$ را برجسته می‌کند. تغییر در این وزن تأثیری بر خروجی لایه‌های قبلی ندارد، بنابراین برای محاسبهٔ $\frac{\partial C}{\partial w_{43}^{[3]}}$ نیازی به دانستن مشتقات جزئی در لایه‌های قبلی نداریم. اما باید بتوان $\frac{\partial C}{\partial w_{43}^{[3]}}$ را به کمک مشتقات جزئی در لایه‌های بعدی بیان کرد به طور دقیق‌تر، فعال‌سازی که به نورون چهارم در لایه ۳ تغذیه می‌شود $z_4^{[3]}$ است و طبق تعریف، $\delta_4^{[3]}$ حساسیت $C$ نسبت به این ورودی را اندازه‌گیری می‌کند. ورودی به این نورون به صورت ثابت $+ w_{43}^{[3]} a_3^{[2]}$ است، بنابراین منطقی است که

$$\frac{\partial C}{\partial w_{43}^{[3]}} = \delta_4^{[3]} a_3^{[2]}, \tag{۲۳-۲}$$

به طور مشابه، از نظر بایاس، ثابت $+ b_4^{[3]}$ به نورون تغذیه می‌شود، که توضیح می‌دهد چرا

$$\frac{\partial C}{\partial b_4^{[3]}} = \delta_4^{[3]} \times 1. \tag{۲۴-۲}$$

برای اجتناب از استفاده از ماتریس‌های قطری در روابط ۲-۱۶ و ۲-۱۷، می‌توانیم از نماد ضرب هادامارد استفاده کنیم. اجازه دهید $D^{[l]} \in \mathbb{R}^{n_l \times n_l}$ ماتریس قطری باشد که عنصر $(i,i)$ آن به صورت $\sigma'(z_i^{[l]})$ تعریف می‌شود. بنابراین داریم

$$\delta^{[L]} = D^{[L]}(a^{[L]} - y), \tag{۲۵-۲}$$

و

$$\delta^{[l]} = D^{[l]}(W^{[l+1]})^T \delta^{[l+1]}, \tag{۲۶-۲}$$

که می‌توان این را به صورت زیر گسترش داد

$$\delta^{[l]} = D^{[l]}(W^{[l+1]})^T D^{[l+1]}(W^{[l+2]})^T \cdots D^{[L-1]}(W^{[L]})^T D^{[L]}(a^{[L]} - y). \tag{۲۷-۲}$$

[1] Backward pass

همچنین از رابطهٔ ۱ - ۲ به یاد داریم که محاسبهٔ $\sigma'(z)$ ساده است.

رابطهٔ ۲ - ۱۸ نشان می‌دهد که $\delta^{[l]}$ دقیقاً برابر با گرادیان تابع هزینه نسبت به بایاس‌ها در لایه $l$ است. اگر $\frac{\partial C}{\partial w_{jk}^{[l]}}$ به‌عنوان مؤلفه $(j,k)$ در یک ماتریس از مشتقات جزئی در لایه $l$ در نظر گرفته شود، آنگاه رابطه ۲ - ۱۹ نشان می‌دهد که این ماتریس حاصل‌ضرب خارجی $\delta^{[l]}(a^{[l-1]})^T \in \mathbb{R}^{n_l \times n_{l-1}}$ است. با ترکیب این موارد، می‌توانیم شبه کد زیر را برای الگوریتمی که شبکه‌ای را با استفاده از تعداد ثابتی از تکرارهای گرادیان نزولی تصادفی $N_{\text{iter}}$ آموزش می‌دهد، بنویسیم. برای سادگی، نسخه ساده ۲ - ۹را در نظر می‌گیریم که در آن نمونه‌ها به‌صورت با جایگزینی انتخاب می‌شوند. برای هر نقطه آموزشی، یک عبور روبه‌جلو از شبکه انجام می‌دهیم تا فعال‌سازی‌ها، ورودی‌های وزنی و خروجی کلی $a^{[L]}$ را ارزیابی کنیم. سپس یک عبور روبه‌عقب انجام می‌دهیم تا خطاها و به‌روزرسانی‌ها را محاسبه کنیم.

با این روند، برای هر نقطه آموزشی، ابتدا شبکه را در جهت پیشرو [1]ارزیابی می‌کنیم و سپس خطاها را با استفاده از پیش انتشار [2] محاسبه می‌کنیم. این روش به‌روزرسانی وزن‌ها و بایاس‌ها را برای بهبود عملکرد شبکه در پیش‌بینی داده‌های آموزشی ممکن می‌سازد.

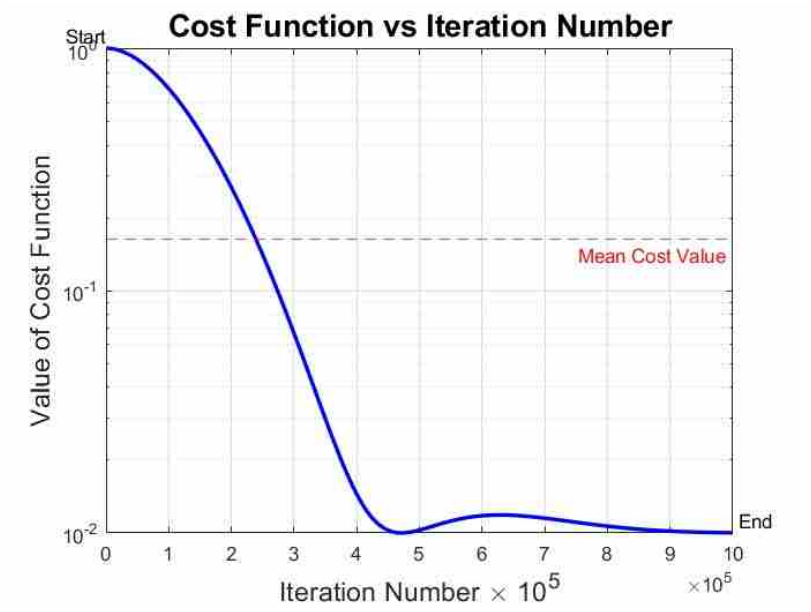
**مثال عملی**

در این بخش، یک مثال عملی از پیش انتشار و روش گرادیان نزولی تصادفی را ارائه می‌دهیم. <<Listing1>> نشان می‌دهد که چگونه یک شبکه عصبی با ساختار نشان‌داده‌شده در شکل ۱ - ۵ می‌تواند بر روی داده‌های موجود در شکل ۱ - ۴ مورداستفاده قرار گیرد. تعداد لایه‌ها به طور ثابت تعیین شده و مراحل عبور پیشرو و پس رو خط‌به‌خط اجرا شده‌اند. ازآنجاکه وزن‌ها و بایاس‌ها در هر لایه ابعاد یکسانی ندارند، ذخیره آن‌ها در یک آرایه سه‌بعدی مناسب نیست. می‌توانیم از یک آرایه سلولی یا ساختاری استفاده کنیم و سپس عملیات پیشرو و پس رو را در حلقه‌های `for` پیاده‌سازی کنیم. تابع `netbp` در<<Listing1>> شامل یک تابع تودرتو به نام `cost` است که نسخه‌ای مقیاس شده از هزینه در رابطه ۱ - ۹ را محاسبه می‌کند. ازآنجاکه این تابع تودرتو است، به متغیرهای موجود در تابع اصلی، به‌ویژه داده‌های آموزشی، دسترسی دارد. لازم به ذکر است که تابع تو در توی `cost` به طور مستقیم در مراحل پیشرو و پس رو استفاده نمی‌شود. این تابع در هر تکرار از روش گرادیان نزولی تصادفی فراخوانی می‌شود تا پیشرفت آموزش را پایش کنیم.

کد <<Listing2>> تابع `activate` را نشان می‌دهد که توسط `netbp` استفاده می‌شود و تابع سیگموید را به‌صورت برداری اعمال می‌کند. در ابتدای `netbp`، داده‌های آموزشی و مقادیر هدف $y$ را همان‌طور که در رابطه

[1] Forward pass
[2] Backward pass

۱-۸ تعریف شده‌اند، تنظیم می‌کنیم. سپس تمام وزن‌ها و بایاس‌ها را با استفاده از مولد اعداد تصادفی نرمال `randn` مقداردهی اولیه می‌کنیم. برای سادگی، نرخ یادگیری ثابت $\eta = 0.05$ را تنظیم کرده و تعداد ثابتی از تکرارها را با مقدار $N_{\text{iter}} = 1e6$ انجام می‌دهیم. ما از روش تکرار گرادیان تصادفی ساده که در پایان‌بخش ۵ خلاصه شده است، استفاده می‌کنیم. در اینجا، دستور `randi(10)` یک عدد صحیح را به طور یکنواخت و مستقل از بین ۱ تا ۱۰ انتخاب می‌کند. پس از ذخیره‌سازی مقدار تابع هزینه در هر تکرار، از دستور `semilogy` برای نمایش پیشرفت تکرار استفاده می‌کنیم. در این آزمایش، حدس اولیه ما برای وزن‌ها و بایاس‌ها مقدار ۳.۵ را برای تابع هزینه به دست آورد. پس از $10^6$ مرحله گرادیان تصادفی، این مقدار به $7.3 \times 10^{-4}$ کاهش یافت. شکل ۲-۲ نمودار `semilogy` را نشان می‌دهد و می‌بینیم که کاهش هزینه یک‌نواخت نیست.




شکل ۲-۲: تابع هزینه در مقابل تعداد تکرار

این مثال MATLAB نشان می‌دهد که چگونه می‌توان یک شبکه عصبی ساده را با استفاده از پیش انتشار و پس انتشار آموزش داد. نتایج نشان می‌دهند که این روش گرادیان نزولی تصادفی می‌تواند به‌تدریج خطای شبکه را کاهش داده و به بهبود عملکرد پیش‌بینی کمک کند. این روند کاهش هزینه در شکل ۲-۲ نمایش‌داده‌شده است و نشان می‌دهد که اگرچه فرایند کاهش هزینه یک‌نواخت نیست، اما با گذر زمان و تعداد تکرارهای زیاد، بهبود قابل‌توجهی حاصل می‌شود.

```
function netbp
% NETBP - Neural network using backpropagation for training
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%% Initial Setup %%%%%%%%%%%
% Input data
x1 = [0.1, 0.3, 0.1, 0.6, 0.4, 0.6, 0.5, 0.9, 0.4, 0.7];
x2 = [0.1, 0.4, 0.5, 0.9, 0.2, 0.3, 0.6, 0.2, 0.4, 0.6];

% Output labels
y = [ones(1, 5), zeros(1, 5); zeros(1, 5), ones(1, 5)];

% Initialize weights and biases
rng(5000);
W2 = initializeWeights(2, 2);
W3 = initializeWeights(3, 2);
W4 = initializeWeights(2, 3);
b2 = initializeWeights(2, 1);
b3 = initializeWeights(3, 1);
b4 = initializeWeights(2, 1);

% Training settings
eta = 0.05;           % Learning rate
Niter = 1e6;          % Number of iterations
savecost = zeros(Niter, 1); % Cost function value at each iteration

% Training loop
for counter = 1:Niter
k = randi(10); % Choose a random training sample
x = [x1(k); x2(k)];

% Forward pass
[a2, a3, a4] = forwardPass(x, W2, W3, W4, b2, b3, b4);

% Backward pass and weight update
[W2, W3, W4, b2, b3, b4] = backPropagation(x, y(:, k), a2, a3, a4,
    W2, W3, W4, b2, b3, b4, eta);

% Compute and save cost function value
savecost(counter) = computeCost(x1, x2, y, W2, W3, W4, b2, b3, b4);
end

% Plot cost function decay
semilogy(1:1e4:Niter, savecost(1:1e4:Niter));
xlabel('Iteration');
ylabel('Cost');
title('Cost Function Decay');
grid on;

%%%%%%%%%%%%% Helper Functions %%%%%%%%%%%%%

% Function to initialize weights
function W = initializeWeights(rows, cols)
W = 0.5 * randn(rows, cols);
end

```

```
    % Function for forward pass
    function [a2, a3, a4] = forwardPass(x, W2, W3, W4, b2, b3, b4)
    a2 = activate(x, W2, b2);
    a3 = activate(a2, W3, b3);
    a4 = activate(a3, W4, b4);
    end

    % Function for backpropagation and updating weights
    function [W2, W3, W4, b2, b3, b4] = backPropagation(x, y, a2, a3,
        a4, W2, W3, W4, b2, b3, b4, eta)
    delta4 = a4 .* (1 - a4) .* (a4 - y);
    delta3 = a3 .* (1 - a3) .* (W4' * delta4);
    delta2 = a2 .* (1 - a2) .* (W3' * delta3);

    W4 = W4 - eta * delta4 * a3';
    W3 = W3 - eta * delta3 * a2';
    W2 = W2 - eta * delta2 * x';
    b4 = b4 - eta * delta4;
    b3 = b3 - eta * delta3;
    b2 = b2 - eta * delta2;
    end

    % Function to compute cost
    function costval = computeCost(x1, x2, y, W2, W3, W4, b2, b3, b4)
    costvec = zeros(10, 1);
    for i = 1:10
    x = [x1(i); x2(i)];
    [a2, a3, a4] = forwardPass(x, W2, W3, W4, b2, b3, b4);
    costvec(i) = norm(y(:, i) - a4, 2);
    end
    costval = norm(costvec, 2)^2;
    end

    % Activation function (sigmoid)
    function a = activate(x, W, b)
    a = 1 ./ (1 + exp(-(W * x + b)));
    end
    end

    % Second MATLAB Code Block
```

Listing 1: Neural Network with Backpropagation

```
1
2     function y = activate(x, W, b)
3     % ACTIVATE - Computes the sigmoid function output
4     %
5     % This function takes the input vector x and computes the sigmoid
6     % output by applying weights and biases.
7     %
8     % Inputs:
9     %   x - Input vector (n x 1)
10    %   W - Weight matrix (m x n)
11    %   b - Bias vector (m x 1)
12    %
13    % Output:
14    %   y - Sigmoid output vector (m x 1)
15    %
16    % Computation:
17    %   y = 1 ./ (1 + exp(-(W * x + b)));
18
19    % Compute sigmoid output
20    z = W * x + b;   % Linear combination of inputs
21    y = 1 ./ (1 + exp(-z));   % Apply sigmoid function to the linear
          combination
22    end
```

Listing 2: Activation Function

# فصل ۳
## کاربردهای عملی شبکه‌های عصبی و ابزارهای پیاده‌سازی

در این بخش به یک مسئله کاربردی‌تر می‌پردازیم که به ما امکان می‌دهد قابلیت‌های رویکرد یادگیری عمیق را به‌صورت مؤثرتری به نمایش بگذاریم. برای این منظور، از جعبه‌ابزار MatConvNet استفاده می‌شود که برای ارائه بلوک‌های اصلی یادگیری عمیق به‌صورت دستورات ساده MATLAB طراحی شده است؛ بنابراین این ابزار، محیطی ایده‌آل برای نمونه‌سازی سریع و کاربردهای آموزشی به شمار می‌آید. علاوه بر این، پشتیبانی ازGPU ها همچنین MatConvNet را برای محاسبات در مقیاس بزرگ کارآمد می‌سازد و شبکه‌های از پیش آموزش‌دیده نیز به‌سادگی قابل‌استفاده هستند. استفاده از MatConvNet در یک مسئله در مقیاس بزرگ همچنین این فرصت را فراهم می‌کند تا مفاهیم بیشتری را که به محاسبات عملی را مورد بررسی قرار دهیم.

### ۳-۱ شبکه‌های عصبی پیچشی: اصول، ساختار و کاربردها

جعبه ابزار MatConvNet از یک دسته ویژه از شبکه‌های عصبی مصنوعی به نام شبکه‌های عصبی پیچشی[1] (CNNs) استفاده می‌کند که به یک ابزار استاندارد در کاربردهای بینایی کامپیوتر تبدیل شده‌اند. به عنوان مثال، یک تصویر رنگی که از $200 \times 200$ پیکسل تشکیل شده و هر پیکسل دارای اجزای قرمز، سبز و آبی است را در نظر بگیرید. این به یک بردار ورودی با ابعاد

$$n_1 = 200 \times 200 \times 3 = 120{,}000$$

[1] Convolutional Neural Network

و در نتیجه یک ماتریس وزن $W^{[2]}$ در سطح ۲ با ۱۲۰،۰۰۰ ستون منجر می‌شود. اگر وزن‌ها و بایاس‌ها را به‌طورکلی در نظر بگیریم، این رویکرد به‌وضوح غیرممکن است. CNNها با محدودکردن مقادیر مجاز در ماتریس‌های وزن و بردارهای بایاس، این مسئله را به طور مؤثری مدیریت می‌کنند. به‌جای یک تبدیل خطی کامل، CNNها یک هسته یا فیلتر خطی کوچک را به طور مکرر در بخش‌های مختلف داده‌های ورودی خود اعمال می‌کنند. در واقع، ماتریس‌های وزن مورداستفاده در CNNها بسیار پراکنده و با ساختار خاص هستند.

برای درک اینکه چرا این رویکرد ممکن است مفید باشد، در نظر بگیرید که یک بردار ورودی در $\mathbb{R}^6$ را با ماتریس

$$\begin{bmatrix} 1 & -1 & & & & \\ & 1 & -1 & & & \\ & & 1 & -1 & & \\ & & & 1 & -1 & \\ & & & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{5\times 6} \qquad (۱\text{-}۳)$$

ضرب کنیم. این کار یک بردار در $\mathbb{R}^5$ تولید می‌کند که از تفاوت‌های بین مقادیر مجاور تشکیل شده است. در این مورد، از یک فیلتر $[1,-1]$ و گام طول یک استفاده می‌شود(فیلتر بعد از هر بار استفاده به یک مکان جلو می‌رود). تعمیم‌های مناسب این ماتریس به حالت بردارهای ورودی ناشی از تصاویر دو بعدی می‌تواند برای شناسایی لبه‌ها در تصویر استفاده شود(که در صورت وجود تغییر ناگهانی در مقادیر پیکسل‌های مجاور، یک مقدار بزرگ را تولید می‌کند). حرکت‌دادن یک فیلتر در سراسر یک تصویر می‌تواند ویژگی‌های دیگری مانند انواع خاصی از منحنی‌ها یا لکه‌های هم‌رنگ را نیز آشکار کند؛ بنابراین، با مشخص‌کردن اندازه فیلتر و طول گام، می‌توانیم اجازه دهیم که فرایند آموزش وزن‌های فیلتر را به‌عنوان ابزاری برای استخراج ساختارهای مفید یاد بگیرد.

اصطلاح «پیچشی»[1] به این دلیل به کار می‌رود که تبدیل‌های خطی مرتبط با این شبکه‌ها را می‌توان به‌صورت پیچشی نوشت. در حالت یک‌بعدی، کانولوشن بردار $x \in \mathbb{R}^p$ با فیلتر $g_{1-p}, g_{2-p}, \ldots, g_{p-2}, g_{p-1}$ دارای مؤلفهٔ $k$ـام به‌صورت زیر

$$y_k = \sum_{n=1}^{p} x_n g_{k-n}. \qquad (۲\text{-}۳)$$

مثال موجود در۳-۱ مربوط به فیلتری با $g_0 = 1$، $g_{-1} = -1$، و سایر $g_k = 0$ است. در حالتی که

[1] Convolutiona

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} a & b & c & d & & & & & & & & \\ & & a & b & c & d & & & & & & \\ & & & & a & b & c & d & & & & \\ & & & & & & a & b & c & d & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ 0 \end{bmatrix} \quad (۳-۳)$$

ما از فیلتری با چهار وزن $a$، $b$، $c$ و $d$ استفاده می‌کنیم که دارای طول گام دو است. به دلیل عدم تطابق طول بردار ورودی $x$ با طول فیلتر، یک مقدار صفر به آن اضافه کرده‌ایم.

## ۳-۲ پیشگیری از بیش‌برازش در پردازش تصاویر

در عمل، داده‌های تصویری به طور معمول به‌عنوان یک تانسور[1] سه‌بعدی در نظر گرفته می‌شوند به‌طوری‌که هر پیکسل دارای دو مختصات فضایی و یک مقدار قرمز/سبز/آبی است. از این دیدگاه، فیلتر به شکل یک تانسور کوچک است که به طور متوالی به بخش‌های مختلف تانسور ورودی اعمال می‌شود و عملیات پیچشی مربوطه به‌صورت چندبعدی انجام می‌گیرد. از دید محاسباتی، یکی از مزایای کلیدی شبکه‌های عصبی پیچشی این است که محاسبهٔ حاصل‌ضرب‌های ماتریس - بردار که در مراحل پیشرو و پس‌رو در شبکه دخیل هستند، می‌تواند به طور بسیار کارآمدی با استفاده از تکنیک‌های تبدیل سریع انجام شود.

یک‌لایه پیچشی معمولاً با یک‌لایه ادغام [2] دنبال می‌شود که بعد را با نگاشت مناطق کوچک از پیکسل‌ها به اعداد منفرد کاهش می‌دهد. به‌عنوان‌مثال، وقتی این مناطق کوچک به‌صورت مربع‌هایی از چهار پیکسل همسایه در یک تصویر دوبعدی انتخاب می‌شوند، یک‌لایه بیشینه ادغام [3]هر مجموعه از چهار پیکسل را به ترتیب با بیشترین یا میانگین مقدار آن‌ها جایگزین می‌کند.

### ۳-۲-۱ بیش‌برازش

بیش برازش [4]زمانی رخ می‌دهد که یک شبکهٔ آموزش‌دیده روی داده‌های موجود عملکرد بسیار دقیقی دارد، اما نمی‌تواند به‌خوبی به داده‌های جدید تعمیم یابد. به‌طورکلی، این بدان معنی است که فرایند برازش بیش از حد روی »نویز»های غیرمهم و غیر نمایشی در داده‌های موجود متمرکز شده است. روش‌های بسیاری برای مقابله با بیش برازش پیشنهاد شده است که برخی از آن‌ها می‌توانند به طور هم‌زمان استفاده شوند. یک تکنیک مفید این است که داده‌های موجود را به دو گروه مجزا تقسیم کنید.

- **داده‌های آموزشی:**[5] داده‌های آموزشی برای تعریف تابع هزینه‌ای که مسئله بهینه‌سازی را مشخص می‌کند،

[1] Tensor
[2] pooling
[3] Max Pooling
[4] Over-fitting
[5] Training Data

استفاده می‌شوند. این داده‌ها فرآیند یادگیری را هدایت می‌کنند، که در طی آن وزن‌ها به‌طور تکراری به‌روزرسانی می‌شوند.

- **داده‌های اعتبارسنجی:**[1] این داده‌ها در فرایند بهینه‌سازی استفاده نمی‌شوند(آن‌ها هیچ تأثیری در نحوه به‌روزرسانی وزن‌ها در هر مرحله ندارند). ما از داده‌های اعتبارسنجی فقط برای ارزیابی عملکرد شبکه فعلی استفاده می‌کنیم. در هر مرحله از بهینه‌سازی، می‌توانیم تابع هزینه مربوط به داده‌های اعتبارسنجی را ارزیابی کنیم. این عملکرد مجموعه فعلی از وزن‌های آموزش‌دیده روی داده‌های نادیده را اندازه‌گیری می‌کند.

به طور شهودی، بیش برازش به وضعیتی اشاره دارد که فرایند بهینه‌سازی در حال کاهش تابع هزینه خود (ارائهٔ برازش بهتر به داده‌های آموزشی) است، اما تابع هزینه برای خطای اعتبارسنجی دیگر در حال کاهش نیست (بنابراین عملکرد در داده‌های نادیده بهبود نمی‌یابد)؛ بنابراین، منطقی است که آموزش را در مرحله‌ای متوقف کنیم که هیچ بهبودی در داده‌های اعتبارسنجی مشاهده نمی‌شود.

یک روش محبوب دیگر برای مقابله با بیش برازش، حذف تصادفی و مستقل نورون‌ها در طول فاز آموزش است. به‌عنوان‌مثال، در هر مرحله از روش گرادیان تصادفی، می‌توانیم هر نورون را با احتمال $p$ حذف کنیم و شبکه باقی‌مانده را آموزش دهیم. در پایان فرایند، به دلیل اینکه وزن‌ها و بایاس‌ها بر روی این شبکه‌های کوچک‌تر آموزش‌دیده‌اند، می‌توانیم هر یک را برای استفاده در شبکه با اندازه کامل، در عامل $p$ ضرب کنیم. این تکنیک که به نام حذف تصادفی [2] شناخته می‌شود، این روش به‌نوعی تفسیر شهودی اشاره دارد«ما با ایجاد میانگینی از بسیاری از شبکه‌های آموزش‌دیده، به اجماعی دست می‌یابیم که از هر یک از شبکه‌های فردی به‌تنهایی قابل‌اعتمادتر است».

## ۳-۳ تحلیل عمیق توابع هزینه و فعال‌سازی در شبکه‌های عصبی

در فصل‌های قبل، ما از توابع فعال‌سازی به شکل سیگموئید ۱-۱و تابع هزینهٔ درجه دوم۲-۳ استفاده کردیم. انتخاب‌های متداول دیگری نیز وجود دارد و عملکرد نسبی آن‌ها بستگی به کاربرد خاص دارد. در زمینهٔ طبقه‌بندی تصویر، استفاده از یک واحد خطی یک‌سو شده[3] معمول است که به‌صورت زیر تعریف می‌شود:

$$\sigma(x) = \begin{cases} 0 & \text{for} x \leq 0, \\ x & \text{for} x > 0. \end{cases} \qquad (۴-۳)$$

[1] Validation Data
[2] Dropout
[3] Rectified Linear Unit (ReLU)

## ۳-۴ بهینه‌سازی دسته‌بندی تصاویر

اگر داده‌های آموزشی ما $\{x^{(i)}\}_{i=1}^{N}$ شامل $K$ دسته‌بندی مختلف با برچسب‌های مشخص باشد، برچسب مربوط به داده $x^{(i)}$ را با $l_i \in \{1, 2, \ldots, K\}$ نمایش می‌دهیم. به‌جای استفاده از تابع هزینه درجه دوم ۲-۳، می‌توانیم از روش بیشینه هموار [1] به شکل زیر استفاده کنیم. خروجی $v^{(i)} := a^{[L]}(x^{(i)})$ از شبکه به‌صورت یک بردار در $\mathbb{R}^K$ است، به‌طوری‌که مؤلفهٔ $j$ـام از این بردار بزرگ‌تر است زمانی که تصویر متعلق به دستهٔ $j$ باشد

$$(v^{(i)})_s \mapsto \frac{e^{v_s^{(i)}}}{\sum_{j=1}^{K} e^{v_j^{(i)}}}.$$

این کار، مؤلفه‌های بزرگ را تقویت کرده و یک بردار از وزن‌های مثبت تولید می‌کند که مجموع آن‌ها برابر با یک است و می‌توان آن را به‌عنوان احتمال تفسیر کرد. اکنون هدف ما این است که مقدار بیشینه هموار برای نقطه آموزشی $x^{(i)}$ در مؤلفه $l_i$ (که همان برچسب صحیح است) تا حد ممکن به عدد یک نزدیک باشد. به‌جای استفاده از تابع هزینه درجه دوم، از یک معیار لگاریتمی استفاده می‌کنیم و به تابع هزینه زیر می‌رسیم

$$-\sum_{i=1}^{N} \log\left(\frac{e^{v_{l_i}^{(i)}}}{\sum_{j=1}^{K} e^{v_j^{(i)}}}\right). \qquad (۵-۳)$$

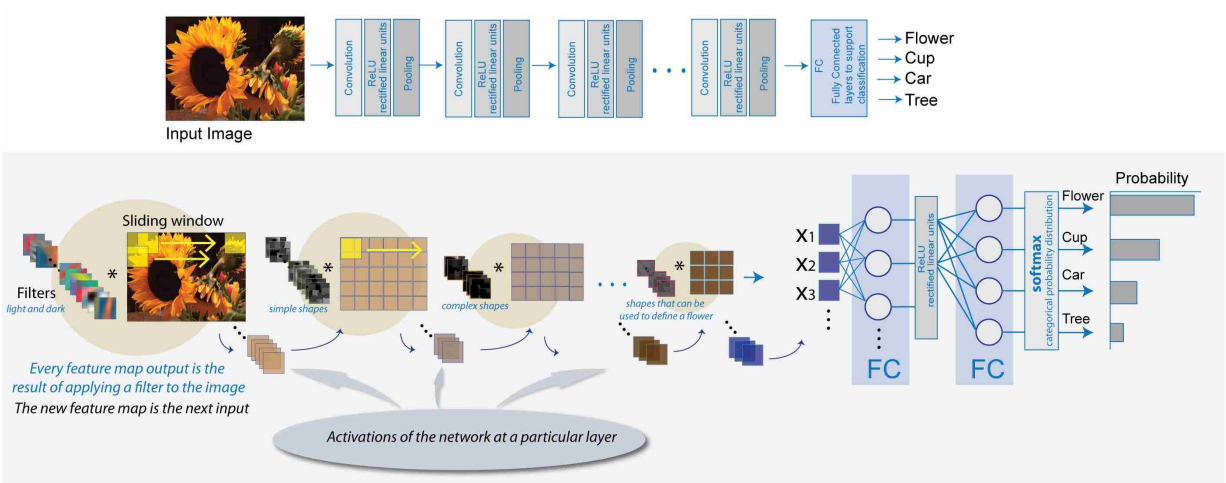
### ۱-۴-۳ پیاده‌سازی و ارزیابی مدل‌های دسته‌بندی تصویر

اکنون نتایج یک کار یادگیری نظارت شده در دسته‌بندی تصویر را نشان می‌دهیم. ما مجموعه‌ای از تصاویر را در نظر می‌گیریم که هر کدام دقیقاً در یکی از ده دستی: هواپیما، اتومبیل، پرنده، گربه، گوزن، سگ، قورباغه، اسب، کشتی، کامیون قرار می‌گیرند. ما از داده‌های برچسب‌دار مجموعه CIFAR-۱۰ استفاده می‌کنیم. تصاویر کوچک هستند و دارای ۳۲ در ۳۲ پیکسل هستند که هر کدام دارای یک مؤلفه قرمز، سبز و آبی هستند؛ بنابراین یک قطعه داده آموزشی شامل $32 \times 32 \times 3 = 3072$ مقدار است. ما از یک مجموعه آموزشی شامل ۵۰,۰۰۰ تصویر و ۱۰,۰۰۰ تصویر دیگر به‌عنوان مجموعه اعتبارسنجی استفاده می‌کنیم. پس از اتمام بهینه‌سازی و آموزش شبکه، عملکرد آن را روی مجموعه جدیدی از ۱۰,۰۰۰ تصویر تست، با ۱,۰۰۰ تصویر از هر دسته، ارزیابی می‌کنیم. توجه داشته باشید که استراتژی‌های اعتبارسنجی متقاطع پیشرفته‌تری می‌توانند مورداستفاده قرار گیرند، جایی که همان مجموعه‌دادهٔ کلی به طور مکرر به مجموعه‌های مجزا برای آموزش، اعتبارسنجی و تست تقسیم می‌شود. بر اساس معماری مورداستفاده در کدهای مربوط به جعبه‌ابزار MatConvNet، شبکه‌ای را راه‌اندازی

[1] Softmax

می‌کنیم که لایه‌های آن به پنج بلوک تقسیم می‌شوند. در اینجا ابعاد ورودی‌ها/خروجی‌ها و وزن‌ها را به‌صورت فشرده در قالب تانسور توصیف می‌کنیم.

**بلوک ۱** شامل یک‌لایه پیچشی است که با یک‌لایه ادغام و فعال‌سازی دنبال می‌شود. این تبدیل ورودی اصلی $3 \times 32 \times 32$ را به بعد $32 \times 16 \times 16$ تبدیل می‌کند. به طور دقیق‌تر، لایه پیچشی از فیلترهای $5 \times 5$ استفاده می‌کند که همچنین در سه کانال رنگی اسکن می‌کنند. ۳۲ فیلتر مختلف وجود دارد، بنابراین به‌طورکلی وزن‌ها می‌توانند در یک آرایه $32 \times 3 \times 5 \times 5$ نمایش داده شوند. خروجی از هر فیلتر ممکن است به‌عنوان یک نقشه ویژگی توصیف شود. فیلترها با یک طول گام واحد [1] اعمال می‌شوند. به‌این‌ترتیب، هر یک از ۳۲ نقشه ویژگی دارای ابعاد $32 \times 32$ است. سپس بیشینه ادغام با استفاده از طول گام دو به هر نقشه ویژگی اعمال می‌شود. این ابعاد نقشه‌های ویژگی را به $16 \times 16$ کاهش می‌دهد. سپس از فعال‌سازی $ReLU$ استفاده می‌شود.[۳]




شکل ۳-۱: نمای کلی معماری شبکه

**بلوک ۲**: در این بلوک ابتدا عملیات پیچشی اعمال می‌شود، سپس تابع فعال‌سازی و در نهایت یک‌لایه ادغام‌کننده. این کار ابعاد را به $32 \times 8 \times 8$ کاهش می‌دهد. به طور دقیق‌تر، ما از ۳۲ فیلتر استفاده می‌کنیم که هر کدام $5 \times 5$ هستند و در تمام ۳۲ نقشه ویژگی اسکن می‌کنند؛ بنابراین وزن‌ها را می‌توان به‌صورت یک تانسور $32 \times 32 \times 5 \times 5$ در نظر گرفت. طول گام برابر یک است، بنابراین نقشه‌های ویژگی ۳۲تایی بعد از کانولوشن همچنان ابعاد $16 \times 16$ دارند. پس از اعمال تابع فعال‌سازی $ReLU$، یک‌لایه میانگین ادغام با گام دو اعمال می‌شود که ابعاد هر کدام از نقشه‌های ویژگی را به $8 \times 8$ کاهش می‌دهد.

**بلوک ۳**: این بلوک ابتدا یک‌لایه پیچشی و سپس تابع فعال‌سازی و عملیات ادغام را اعمال می‌کند که ابعاد

[1] Stride Length

را به $4 \times 4 \times 64$ کاهش می‌دهد. به طور دقیق‌تر، ۶۴ فیلتر به کار می‌روند. هر فیلتر $5 \times 5$ است و در تمام ۳۲ نقشه ویژگی اسکن می‌کند؛ بنابراین وزن‌ها را می‌توان به‌صورت یک تانسور $5 \times 5 \times 32 \times 64$ در نظر گرفت. طول گام برابر یک است و نقشه‌های ویژگی حاصل ابعاد $8 \times 8$ دارند. پس از فعال‌سازی *ReLU*، یک‌لایه میانگین ادغام با گام دو اعمال می‌شود که ابعاد هر کدام از ۶۴ نقشه ویژگی را به $4 \times 4$ کاهش می‌دهد.

**بلوک ۴**: این بلوک فقط از پیچشی و سپس فعال‌سازی استفاده می‌کند، و ابعاد را به $1 \times 1 \times 64$ کاهش می‌دهد. به طور دقیق‌تر، از ۶۴ فیلتر استفاده می‌شود. هر فیلتر $4 \times 4$ است و در ۶۴ نقشه ویژگی اعمال می‌شود؛ بنابراین وزن‌ها را می‌توان به‌صورت یک تانسور $4 \times 4 \times 64 \times 64$ در نظر گرفت و هر فیلتر یک عدد واحد تولید می‌کند.

**بلوک ۵**: این بلوک شامل عملی‌ات پیچشی نیست. از یک ماتریس وزن عمومی (به طور کامل متصل) از نوعی که در بخش‌های ۲ تا ۶ بررسی شد، استفاده می‌کند تا خروجی با ابعاد $1 \times 1 \times 10$ را بدهد. این معادل یک ماتریس وزن با ابعاد $64 \times 10$ است. در نهایت، تابع فعال‌سازی بیشینه هموار هر کدام از ده مؤلفه خروجی را به بازه $[0, 1]$ تبدیل می‌کند.شکل ۱.۷ نمای کلی معماری شبکه را به صورت بصری نشان می‌دهد.

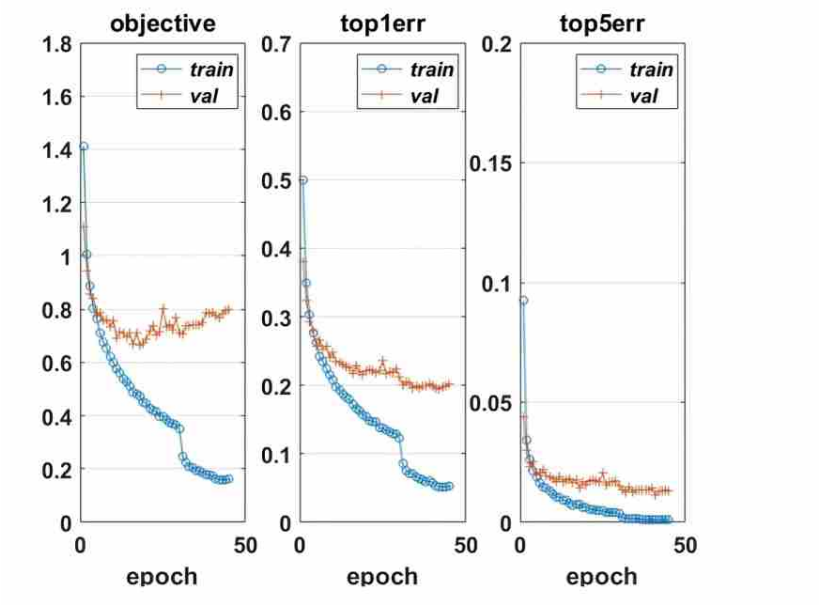
خروجی شبکه یک بردار شامل ده عدد حقیقی است. تابع هزینه برای مسئله بهینه‌سازی به‌صورت لاگ‌لاس با *سافت مکس* و $K = 10$ در نظر گرفته شده است. برای بهینه‌سازی از روش گرادیان تصادفی با momentum استفاده می‌کنیم که در آن «میانگین متحرک» جهت‌های گرادیان فعلی و گذشته محاسبه می‌شود. داده‌ها به‌صورت دسته‌های کوچک [1] با اندازه ۱۰۰ مورداستفاده قرار می‌گیرند و تعداد ثابتی از ۴۵ دوره ($epoch$) برای آموزش تعیین شده است. نرخ یادگیری به‌صورت پیش‌فرض برای هر دوره به این شکل تعریف می‌شود: $\eta = 0.05$ برای ۳۰ دوره اول، $\eta = 0005$ برای ۱۰ دوره بعدی، و $\eta = 0.0005$ برای ۵ دوره نهایی. اجرای این شبکه روی یک پردازنده گرافیکی Tesla C۲۰۷۵ در حالت دقت تک، ۴۵ دوره [2] را در کمتر از چهار ساعت تکمیل می‌کند. به‌عنوان آزمایشی اضافی، ما همچنین شبکه را با استفاده از تکنیک حذف تصادفی [3] آموزش می‌دهیم. در اینجا، در هر مرحله گرادیان تصادفی، خروجی هر نورون با یک احتمال مستقل به صفر بازنشانی می‌شود.

### ۳-۴-۲ تحلیل نتایج و تاثیر حذف تصادفی در بهبود دقت مدل

در شکل ۳-۲، فرایند آموزش را در حالت بدون حذف تصادفی نشان می‌دهیم. در نمودار سمت چپ، دایره‌ها نشان‌دهنده کاهش تابع هدف ۳-۵ پس از هر یک از ۴۵ دوره آموزشی هستند. همچنین از ضربدرها برای نمایش مقدار تابع هدف بر روی داده‌های اعتبارسنجی استفاده می‌شود. (این اندازه‌گیری‌ها به صورت میانگین

[1] Minibatches
[2] Epoch
[3] Dropout

بر روی بچ‌هایی که دوره آموزش را تشکیل می‌دهند، محاسبه شده‌اند. توجه داشته باشید که وزن‌ها پس از هر بچ به‌روزرسانی می‌شوند.) هدف کلی ما طبقه‌بندی تصاویر به یکی از ده دسته است، و نمودار میانی در شکل ۳-۲ درصد خطاهای طبقه‌بندی را با بالاترین احتمال نشان می‌دهد. به‌طور مشابه، نمودار سمت راست درصد مواردی را نشان می‌دهد که دسته صحیح در میان پنج دسته برتر نیست.

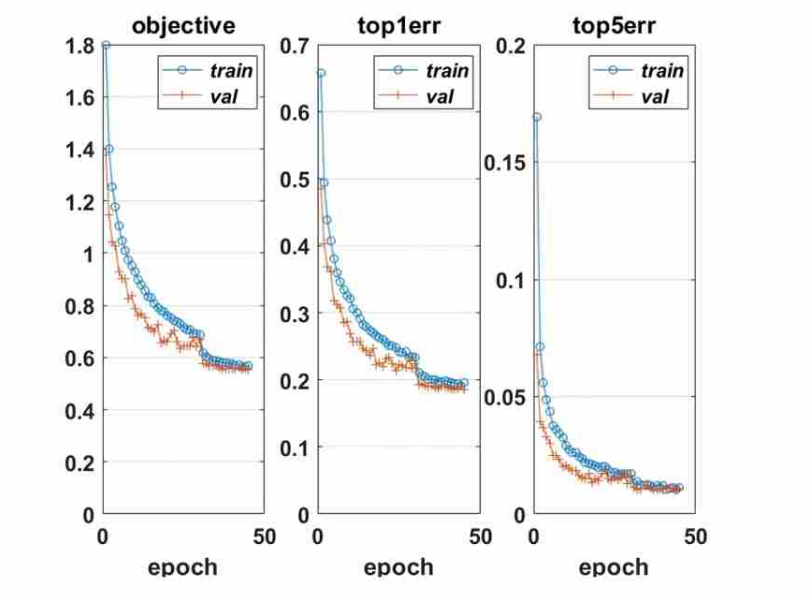



شکل ۳-۲: آموزش شبکه بدون حذف تصادفی

از شکل ۳-۲ مشاهده می‌کنیم که خطای اعتبارسنجی در مرحله‌ای به حالت ثابت می‌رسد، درحالی‌که روش گرادیان تصادفی همچنان کاهش‌های قابل‌توجهی در خطای آموزش ایجاد می‌کند. این نشان‌دهنده بیش برازش است یعنی شبکه در حال یادگیری جزئیات دقیق داده‌های آموزشی است که به بهبود عملکرد در مواجهه با داده‌های دیده نشده کمک نخواهد کرد.

شکل ۳-۳ نتایج مشابه را در حالتی که از حذف تصادفی استفاده شده است، نشان می‌دهد. در این حالت، خطاهای آموزشی به طور قابل‌توجهی بزرگ‌تر از شکل ۳-۲ هستند، و خطاهای اعتبارسنجی دارای بزرگی مشابهی هستند. بااین‌حال، دو ویژگی کلیدی در حالت حذف تصادفی عبارت‌اند از: (الف) خطای اعتبارسنجی کمتر از خطای آموزشی است و (ب) خطای اعتبارسنجی به همراه خطای آموزشی به کاهش ادامه می‌دهد. این موارد نشان می‌دهند که فرایند بهینه‌سازی در حال استخراج اطلاعات مفید در طول تمام دوره‌های آموزش است.

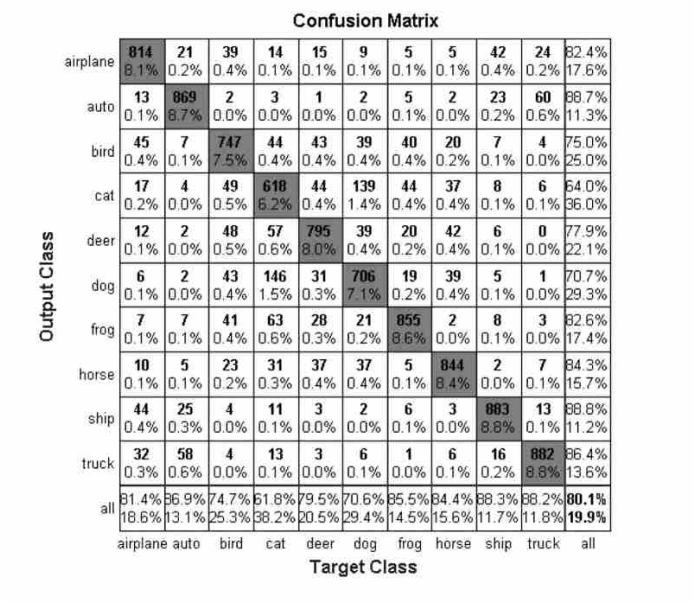
شکل ۳-۴عملکرد شبکه آموزش‌دیده بدون حذف تصادفی (پس از ۴۵ دوره) را به‌صورت یک ماتریس سردرگمی خلاصه می‌کند. مقدار صحیح در ورودی عمومی $(i, j)$ تعداد مواردی را نشان می‌دهد که شبکه دسته $i$ را برای تصویری از دسته $j$ پیش‌بینی کرده است. عناصر خارج از قطر اصلی نشان‌دهنده خطاهای طبقه‌بندی هستند.



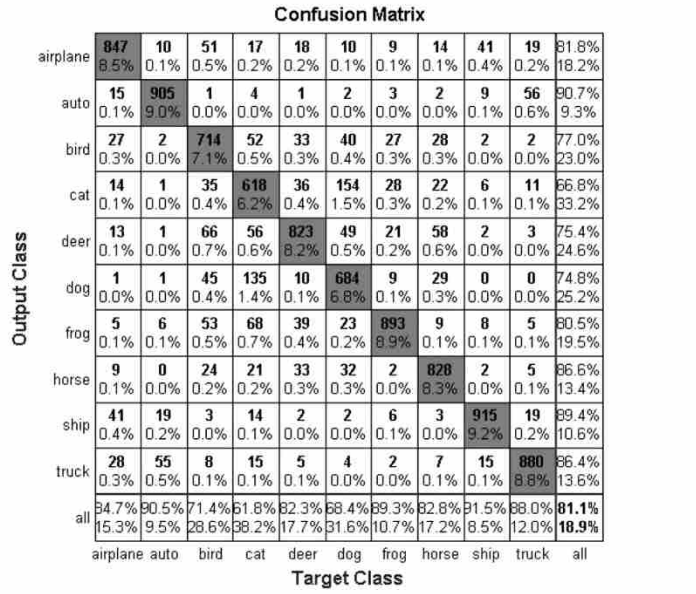

شکل ۳-۳: آموزش شبکه با حذف تصادفی

برای مثال، عنصر $(1,1)$ برابر با ۸۱۴ تعداد تصاویر هواپیما را که به‌درستی به‌عنوان هواپیما طبقه‌بندی شده‌اند، نشان می‌دهد، و عنصر $(1,2)$ برابر با ۲۱ تعداد تصاویر خودرو را که به‌اشتباه به‌عنوان هواپیما طبقه‌بندی شده‌اند، نشان می‌دهد. در زیر هر عدد صحیح درصد مربوطه، به یک رقم اعشار گرد شده، داده شده است، باتوجه‌به اینکه داده‌های آزمایشی از هر دسته شامل ۱۰۰۰ تصویر است. سطر اضافی با برچسب «همه» ورودی‌های هر ستون را خلاصه می‌کند. به‌عنوان‌مثال، مقدار ۸۱/۴٪ در ستون اول سطر نهایی به این دلیل است که ۸۱۴ از ۱۰۰۰ تصویر هواپیما به‌درستی طبقه‌بندی شده‌اند. در زیر این، مقدار ۱۸/۶٪ به این دلیل است که ۱۸۶ از این تصاویر هواپیما به‌اشتباه طبقه‌بندی شده‌اند. ستون نهایی ماتریس، با برچسب «همه»، هر سطر را خلاصه می‌کند. به‌عنوان‌مثال، مقدار ۸۲/۴٪ در ستون نهایی سطر اول به این دلیل است که ۹۸۸ تصویر توسط شبکه به‌عنوان هواپیما طبقه‌بندی شده‌اند که از این تعداد ۸۱۴ طبقه‌بندی صحیح بوده‌اند. در زیر این، مقدار ۱۷/۶٪ به این دلیل است که باقی‌مانده ۱۷۴ از این ۹۸۸ طبقه‌بندی هواپیما نادرست بوده‌اند. در نهایت، ورودی‌های گوشه پایین سمت راست همه دسته‌ها را خلاصه می‌کنند. می‌بینیم که ۸۰/۱٪ از تمام تصاویر به‌درستی طبقه‌بندی شده‌اند (و بنابراین ۱۹/۹٪ به‌اشتباه طبقه‌بندی شده‌اند).

شکل ۳-۵ نتایج مربوط به حالتی که در آن از حذف تصادفی استفاده شده است را نشان می‌دهد. مشاهده می‌کنیم که استفاده از حذف تصادفی به‌طورکلی عملکرد را بهبود بخشیده است و به‌ویژه نرخ موفقیت کلی را از ۸۰/۱٪ به ۸۱/۱٪ افزایش داده است. حذف تصادفی در نه دسته از ده دسته مقادیر بیشتری در عناصر قطری ماتریس سردرگمی به وجود آورده است.




شکل ۳-۴: ماتریس در هم‌ریختگی بدون حذف تصادفی



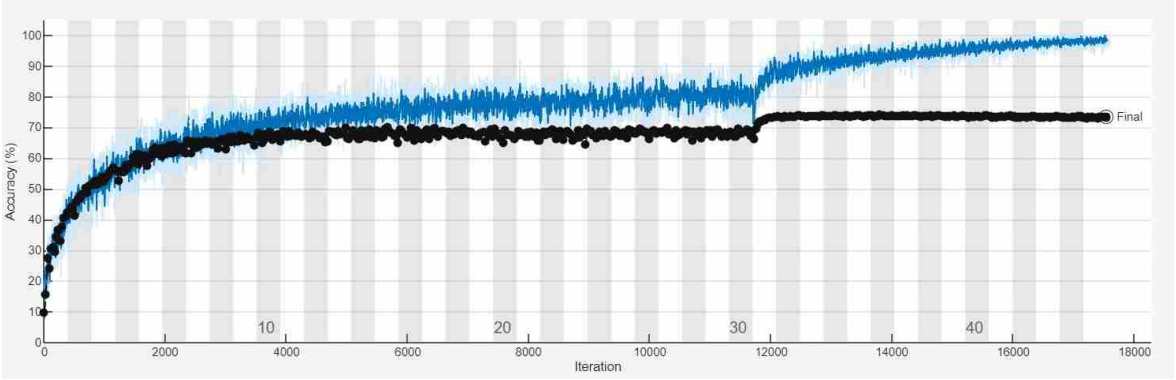

شکل ۳-۵: ماتریس در هم‌ریختگی با حذف تصادفی

## ۳-۵ شبیه‌سازی و ارزیابی مدل‌های شبکه عصبی

در ادامه به شبیه‌سازی شبکه مورد نظر با استفاده از کد های <<Listing3>> پرداخته و نتایج حاصل از آن را ارائه می‌دهیم.نمودار۳-۶ دقت[1] کد شبیه‌سازی‌شده را نشان می‌دهد؛ خط آبی نشان‌دهنده دقت مدل در طول آموزش

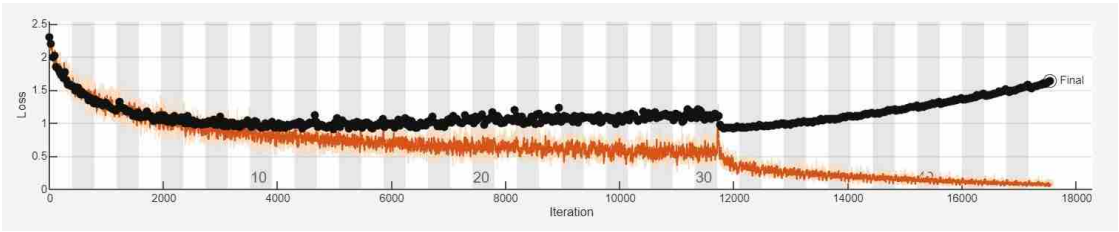
[1] Accuracy

است و خط مشکی نشان‌دهنده دقت مدل در مجموعه‌داده‌های اعتبارسنجی است. دقت مدل در ابتدا افزایشی بوده و در نهایت به دقتی حدود ۷۳ درصد میرسد همچنین دقت اعتبارسنجی نیز به تریج روند صعودی داشته و در نهایت به مقدار ثابتی نزدیک شده است. نمودار ۳-۷ میزان خطای شبکه [۱] را نشان می‌دهد. خط نارنجی




شکل ۳-۶: دقت آموزش شبکه شبیه سازی شده بدون حذف تصادفی

نشان‌دهنده خطای مدل در طول آموزش و خط مشکی نشان‌دهنده خطای مدل در مجموعه‌داده‌های اعتباری است. خطای مدل به‌تدریج کاهش‌یافته و به سطح پایداری نزدیک می‌شود؛ اما در مدل اعتبارسنجی به طور ناگهانی افزایش می‌یابد و می‌تواند نشانه‌ای از بیش برازش باشد. به همین ترتیب، دو نمودار ۳-۸ ، ۳-۹ نشان‌دهنده

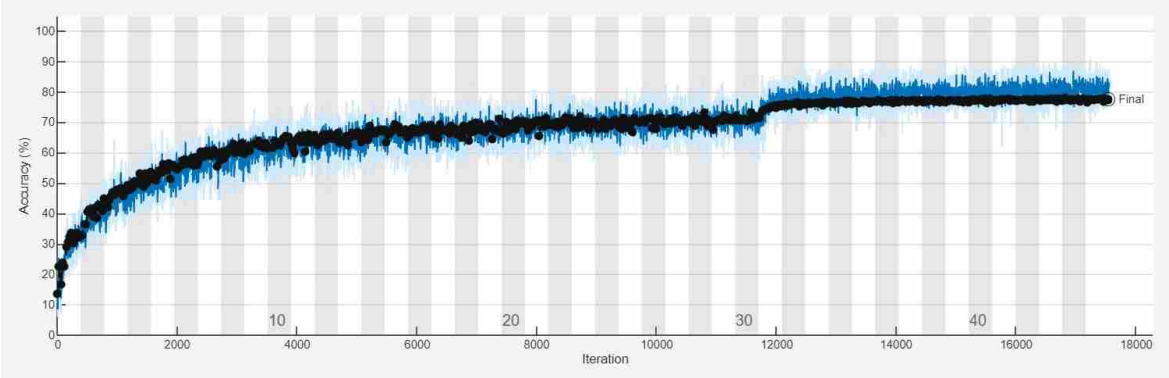



شکل ۳-۷: خطای شبکه شبیه سازی شده با حذف تصادفی

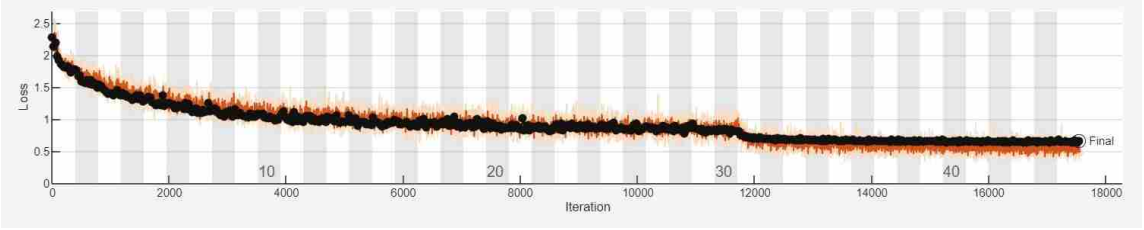
دقت و خطای مدل با اعمال حذف تصادفی در شبکه هستند. در این حالت، دقت اعتبارسنجی به بیش از ۷۷ درصد رسیده است که نشان‌دهنده بهبود عملکرد مدل در مقایسه با مدل اولیه است.

دو نمودار ۳-۱۰ ، ۳-۱۱ ماتریس درهم‌ریختگی در دو مدل ذکر شده را نشان می‌دهد.

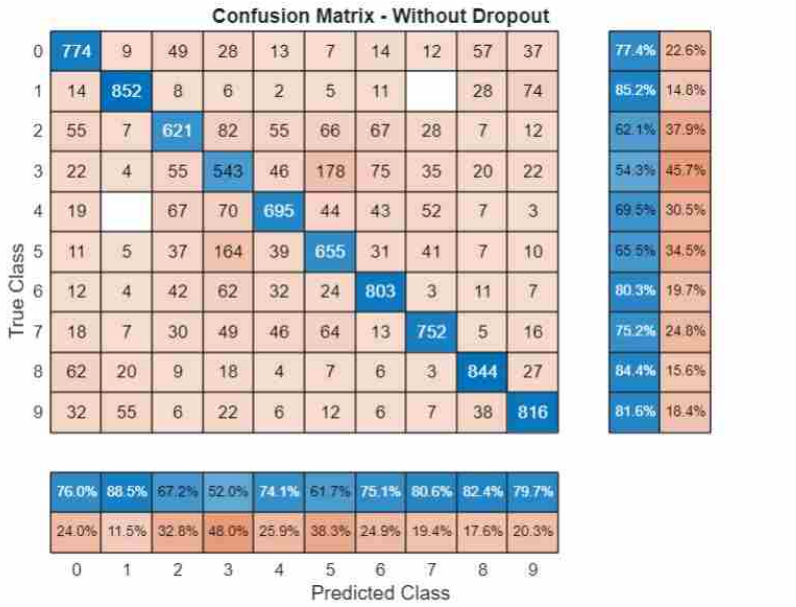
[1] Loss



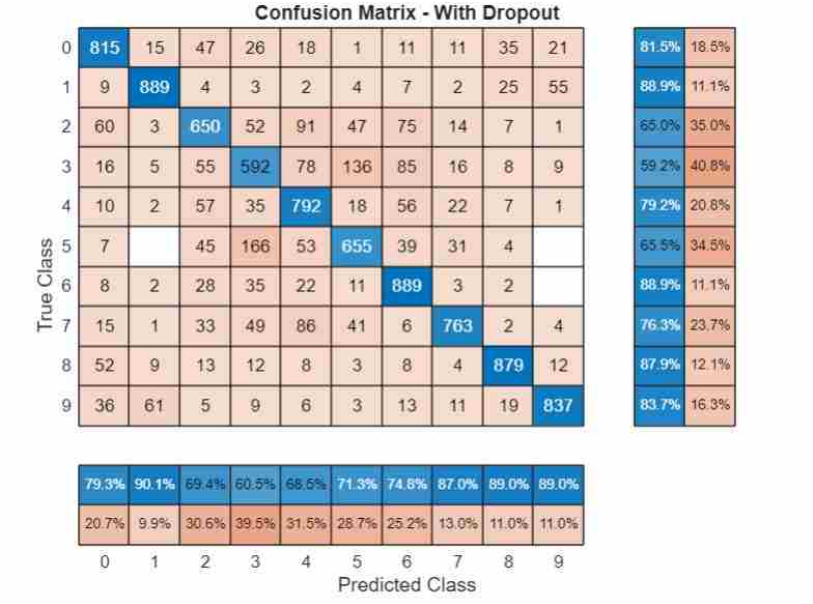

شکل ۳-۸: دقت آموزش شبکه شبیه سازی شده با حذف تصادفی




شکل ۳-۹: خطای شبکه شبیه سازی شده با حذف تصادفی




شکل ۳-۱۰: ماتریس در هم‌ریختگی مدل شبیه سازی شده بدون حذف تصادفی




شکل ۳-۱۱: ماتریس در هم‌ریختگی مدل شبیه سازی شده با حذف تصادفی

## ۳-۶ جمع‌بندی

در این گزارش، به بررسی مفاهیم پایه‌ای و تکنیک‌های پیشرفته در حوزه شبکه‌های عصبی مصنوعی و یادگیری عمیق پرداخته شد. روش‌های مختلف بهینه‌سازی و ساختارهای متنوع شبکه‌های عصبی از جمله شبکه‌های چندلایه (MLP) و شبکه‌های عصبی پیچشی (CNN) به‌تفصیل موردبحث قرار گرفتند. همچنین، تکنیک‌های بهبود عملکرد مانند پس انتشار خطا، گرادیان نزولی و حذف تصادفی که به کاهش بیش برازش و افزایش دقت مدل‌ها کمک می‌کنند، معرفی شدند.

در بخش‌های مختلف گزارش، کاربردهای عملی این تکنیک‌ها در مسائل پیچیده‌ای نظیر پردازش تصویر و یادگیری ویژگی‌های غیرخطی مورد بررسی قرار گرفت. استفاده از ابزارهایی مانند MatConvNet نشان داد که می‌توان با بهره‌گیری از کتابخانه‌های مناسب، فرایند آموزش مدل‌ها را تسریع و بهینه کرد.

نتایج نشان می‌دهد که استفاده هوشمندانه از تکنیک‌های بهینه‌سازی و تنظیمات مختلف شبکه‌های عصبی می‌تواند تأثیر چشمگیری در افزایش دقت و کارایی مدل‌ها داشته باشد. بااین‌حال، برای دستیابی به بهترین عملکرد، نیازمند تنظیم دقیق پارامترها و روش‌های بهینه‌سازی هستیم که این امر بستگی به نوع داده‌ها و مسئله موردنظر دارد.

به‌طورکلی، این گزارش نشان می‌دهد که یادگیری عمیق و شبکه‌های عصبی با وجود چالش‌های موجود، ابزارهای قدرتمندی برای حل مسائل پیچیده در حوزه‌های مختلف هستند و با پیشرفت‌های مداوم در الگوریتم‌ها و سخت‌افزارها، انتظار می‌رود که کاربردهای آن‌ها به طور قابل‌توجهی گسترش یابد.

```
% Load CIFAR-10 dataset
cifar10Data = fullfile(tempdir, 'cifar10');
cifar10URL = 'https://www.cs.toronto.edu/~kriz/cifar-10-matlab.tar.gz';

% Download and extract CIFAR-10 dataset
if ~exist(cifar10Data, 'dir')
disp('Downloading CIFAR-10 dataset...');
downloadFolder = tempdir;
cifar10TarFile = fullfile(downloadFolder, 'cifar-10-matlab.tar.gz');
websave(cifar10TarFile, cifar10URL);
untar(cifar10TarFile, cifar10Data);
end

% Load CIFAR-10 data
cifar10Data = fullfile(cifar10Data, 'cifar-10-batches-mat');
[trainImages, trainLabels, valImages, valLabels] = loadCIFAR10(cifar10Data);

% Define neural network layers without Dropout
layers_no_dropout = [
imageInputLayer([32 32 3], 'Name', 'input')
convolution2dLayer(5, 32, 'Padding', 2, 'Stride', 1, 'Name', 'conv1')
batchNormalizationLayer('Name', 'bn1')
reluLayer('Name', 'relu1')
maxPooling2dLayer(3, 'Stride', 2, 'Padding', [0 1 0 1], 'Name', 'pool1')
convolution2dLayer(5, 32, 'Padding', 2, 'Stride', 1, 'Name', 'conv2')
reluLayer('Name', 'relu2')
averagePooling2dLayer(3, 'Stride', 2, 'Padding', [0 1 0 1], 'Name', 'pool2')
convolution2dLayer(5, 64, 'Padding', 2, 'Stride', 1, 'Name', 'conv3')
reluLayer('Name', 'relu3')
averagePooling2dLayer(3, 'Stride', 2, 'Padding', [0 1 0 1], 'Name', 'pool3')
convolution2dLayer(4, 64, 'Padding', 0, 'Stride', 1, 'Name', 'conv4')
reluLayer('Name', 'relu4')
convolution2dLayer(1, 10, 'Padding', 0, 'Stride', 1, 'Name', 'conv5')
softmaxLayer('Name', 'softmax')
classificationLayer('Name', 'classification')];

% Set training options
options = trainingOptions('sgdm', ...
'InitialLearnRate', 0.1, ...
'LearnRateSchedule', 'piecewise', ...
'LearnRateDropPeriod', 30, ...
'LearnRateDropFactor', 0.1, ...
'MaxEpochs', 45, ...
'ValidationData', {valImages, valLabels}, ...
'ValidationFrequency', 30, ...
'Verbose', false, ...
```

```
    'Plots', 'training-progress');

    % Train network without Dropout
    net_no_dropout = trainNetwork(trainImages, trainLabels,
        layers_no_dropout, options);

    % Function to load CIFAR-10 data
    function [trainImages, trainLabels, valImages, valLabels] =
        loadCIFAR10(cifar10Data)
    [trainImages, trainLabels] = loadBatch(fullfile(cifar10Data, '
        data_batch_1.mat'));
    for i = 2:5
    [images, labels] = loadBatch(fullfile(cifar10Data, ['data_batch_'
        num2str(i) '.mat']));
    trainImages = cat(4, trainImages, images);
    trainLabels = [trainLabels; labels];
    end
    [valImages, valLabels] = loadBatch(fullfile(cifar10Data, '
        test_batch.mat'));
    end

    function [images, labels] = loadBatch(filename)
    load(filename, 'data', 'labels');
    images = reshape(data', 32, 32, 3, []);
    images = permute(images, [2 1 3 4]);
    labels = categorical(labels);
    end
```

Listing 3: MATLAB code for loading and training neural networks

# مراجع


[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. https://www.deeplearningbook.org

[2] MathWorks. *8x8 RGB LED Matrix*. 2024. https://de.mathworks.com/help/simulink/supportpkg/raspberrypi_ref/8x8rgbledmatrix.html Accessed: 2024-09-04

[3] MathWorks. *Introduction to Convolutional Neural Networks*. 2024. https://de.mathworks.com/help/deeplearning/ug/introduction-to-convolutional-neural-networks.html Accessed: 2024-09-04

[4] MathWorks. *MATLAB Code for CNN on CIFAR-10*. 2024. https://viewer.mathworks.com/?viewer=plain_code&url=https%3A%2F%2Ffr.mathworks.com%2Fmatlabcentral%2Fmlc-downloads%2Fdownloads%2Fe5aef38d-4a80-11e4-9553-005056977bd0%2Fc36e6b81-a2f6-b65f-4ecb-8fa3c6764b50%2Ffiles%2Fexamples%2Fcifar%2Fcnn_cifar_init.m&embed=web Accessed: 2024-09-04

[5] Catherine F. Higham and Desmond J. Higham. *Deep Learning: An Introduction for Applied Mathematicians*. SIAM Review, 61(4):860-891, 2019.