دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

تمرین سوم هوش محاسباتی: شبکه های عصبی و کاربردها

Neural Networks & Applications

نگارش

دانیال شفیعی

مهدی مهدیه

امیررضا نجفی

استاد راهنما

دکتر کارشناس

خرداد ۱۴۰۴

فهرست مطالب

• مقدمه

هدف از این تمرین آشنایی بیشتر با شبکه های عصبی و استفاده‌ی بیشتر از آن‌ها در کاربردهای عملی است.

۱ مفاهیم و حل مسئله

۱. بله، هر نورون در یک شبکهٔ عصبی حامل نوعی اطلاعات است؛ اما ماهیت و میزان «وضوح» این اطلاعات بسته به عمق لایه و ویژگی‌های بنیادین شبکه متفاوت است.

چهار ویژگی بنیادی و سلسله‌مراتبی بودن نمایش:

(آ) **توابع غیرخطی (Nonlinearity)**

- هر نورون پس از ترکیب خطی ورودی‌ها (ضرب وزن‌ها + بایاس) خروجی را از طریق تابعی مانند ReLU، sigmoid یا tanh عبور می‌دهد.
- بدون غیرخطی‌سازی، شبکه عملاً یک عملگر خطی بزرگ خواهد بود و قادر به تشخیص زیرویژگی‌های پیچیده نیست.
- تابع فعال‌سازی باعث می‌شود هر نورون تنها در صورت وقوع یک الگوی خاص «فعال» شود و در نتیجه به‌عنوان یک تشخیص‌دهندهٔ ساده عمل کند.

(ب) **نمایش توزیع‌شده (Distributed Representation)**

- برخلاف سیستم‌های سمبلیک که هر مفهوم را با یک واحد منفرد نمایش می‌دهند، شبکه‌های عصبی مفاهیم را به‌صورت همزمان در بردار فعال‌سازی تعداد زیادی نورون کدگذاری می‌کنند.
- این پراکندگی اطلاعات باعث افزایش مقاومت شبکه در برابر نویز و آسیب به نورون‌های منفرد می‌شود.
- هر نورون سهم جزئی اما معنادار در تشخیص زیرویژگی‌های ساده یا انتزاعی دارد.

(ج) **یادگیری گرادیان‌محور (Gradient-based Learning)**

- با استفاده از الگوریتم پس‌انتشار (Backpropagation)، وزن‌ها و بایاس هر نورون به‌روزرسانی می‌شود تا خطای خروجی به کمترین مقدار برسد.
- در طی آموزش، هر نورون به زیرویژگی‌هایی پاسخ می‌دهد که برای کاهش خطا در مسئلهٔ مشخص مفیدند.
- در پایان آموزش، وزن‌های ورودی هر نورون تعیین می‌کنند که آن نورون به چه الگو یا ویژگی حساس باشد.

(د) **سلسله‌مراتب ویژگی‌ها (Hierarchical Feature Learning)**

- لایه‌های ابتدایی شبکه‌های عمیق معمولاً به زیرویژگی‌های ساده مانند لبه‌های عمودی/افقی یا بافت‌ها حساس‌اند.
- لایه‌های میانی ترکیب این زیرویژگی‌ها را انجام داده و الگوهای پیچیده‌تر را می‌آموزند.
- در لایهٔ خروجی (مثلاً نورون‌های softmax) احتمال تعلق هر ورودی به یک کلاس نهایی (مثلاً «گربه» یا «سگ») کدگذاری می‌شود.

۲. در شبکه‌های عصبی، «دانش» در قالب پارامترها (وزن‌ها و بایاس‌ها) ذخیره می‌شود و از طریق فرآیند آموزش شکل می‌گیرد؛ در ادامه، یک پاسخ یکپارچه و مرتب‌شده ارائه شده است:

(آ) **شکل‌گیری دانش در شبکه‌های عصبی**

i. **تعریف ساختار شبکه (Architecture):** انتخاب تعداد لایه‌ها (Input, Hidden, Output)، نوع آن‌ها (fully-connected، کانولوشن، بازگشتی و ...) و تعداد نورون در هر لایه.

ii. **مقداردهی اولیه پارامترها (Initialization):** وزن‌ها و بایاس‌ها معمولاً با توزیع‌های تصادفی (مثل Xavier یا He) مقداردهی می‌شوند.

iii. **انتشار رو به جلو (Forward Propagation):** برای هر ورودی x، در هر لایه:

$$z^{(\ell)} = W^{(\ell)}\, a^{(\ell-1)} + b^{(\ell)}, \quad a^{(\ell)} = \sigma\big(z^{(\ell)}\big)$$

در نهایت $a^{(L)}$ خروجی نهایی شبکه است.

iv. **محاسبه خطا (Loss Calculation):** با تابع هزینه $L(y_{\text{pred}}, y_{\text{true}})$، مانند MSE برای رگرسیون یا Cross-Entropy برای طبقه‌بندی.

v. **پس انتشار خطا (Backpropagation):** مشتق تابع هزینه را نسبت به پارامترها محاسبه می‌کنیم:

$$\frac{\partial L}{\partial W^{(\ell)}}, \quad \frac{\partial L}{\partial b^{(\ell)}}$$

vi. **به‌روزرسانی پارامترها (Optimization):** با الگوریتم‌هایی مثل Gradient Descent یا Adam:

$$W^{(\ell)} \leftarrow W^{(\ell)} - \eta \frac{\partial L}{\partial W^{(\ell)}}, \quad b^{(\ell)} \leftarrow b^{(\ell)} - \eta \frac{\partial L}{\partial b^{(\ell)}}$$

این چرخه تا رسیدن به همگرایی تکرار می‌شود.

(ب) فرمول‌بندی «معادل بودن» دو شبکه عصبی

i. **معادل تابعی (Exact Functional Equivalence)** دو شبکه $N(x) = f_{\theta_N}(x)$ و $M(x) = f_{\theta_M}(x)$ دقیقاً معادل‌اند اگر:

$$\forall x \in X, \quad N(x) = M(x).$$

ii. *معادل ساختاری تحت تبدیلات* (*Structural Equivalence*) در لایه‌های Dense، جابجایی نورون‌ها (پرموتیشن π) همراه با جابجایی سطرها/ستون‌های متناظر در W, b، خروجی را تغییر نمی‌دهد.

iii. *تقریب معادل* (*Approximate Equivalence*) با فاصلهٔ خروجی $d(x) = \|N(x) - M(x)\|_p$:

$$\forall x \in X,\ d(x) < \epsilon \quad \text{یا} \quad \mathrm{KL}\big(N(x) \| M(x)\big) < \delta.$$

(ج) مثال ریاضی

i. **حالت سادهٔ خطی:** دو شبکه خطی با یک لایه پنهان و $\sigma(z) = z$:

$$N(x) = W_2\,(W_1\,x + b_1) + b_2, \quad M(x) = W_2'\,(W_1'\,x + b_1') + b_2'.$$

آن‌ها معادل‌اند اگر:

$$W_2 W_1 = W_2' W_1', \quad W_2 b_1 + b_2 = W_2' b_1' + b_2'.$$

ii. *اشاره‌ای به حالت غیرخطی:* در شبکه‌های غیرخطی (مثلاً ReLU)، تبدیلات پیچیده‌ترند؛ اما با ادغام BatchNorm یا تبدیلات جبری می‌توان مشابهت رفتار را نشان داد.

۳. در شبکه‌های عصبی، توانایی یادگیری، به‌خاطر سپاری (Memorization) و تعمیم (Generalization) بر پایه‌ی ساختار معماری، الگوریتم‌های آموزش و ویژگی‌های داده‌ها شکل می‌گیرد. در ادامه، این سه قابلیت را همراه با مبانی ریاضی و مثال‌های عینی بررسی می‌کنیم.

(آ) یادگیری (Learning)

شبکه با کمینه‌سازی تابع هزینه

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\big(f_\theta(x_i), y_i\big)$$

و به‌کارگیری انتشار رو به جلو و پسانتشار خطا (Backpropagation)، پارامترهای θ (وزن‌ها و بایاس‌ها) را با الگوریتم‌های گرادیان‌محور (SGD، Adam و...) به‌روزرسانی می‌کند:

$$\theta \;\leftarrow\; \theta - \eta\, \nabla_\theta L(\theta).$$

- **قضیه تقریب جهانی (Universal Approximation Theorem)** هر شبکه‌ی با حداقل یک لایه پنهان و تابع فعال‌سازی غیرفابی (مثلاً ReLU یا Sigmoid) می‌تواند هر تابع پیوسته روی یک مجموعه‌ی کامپکت را تقریب بزند.
- **ویژگی توزیعی (Distributed Representation)** هر نورون یا زیراسختار فقط بخشی از ویژگی‌های داده را مدل می‌کند و با ترکیب میلیون‌ها پارامتر، شبکه قادر به نمایش الگوهای غیرخطی و سلسله‌مراتبی است.

(ب) **به‌خاطر سپاری (Memorization)**

شبکه‌های overparameterized (پارامترها خیلی بزرگتر از نمونه‌ها) می‌توانند جزئیات حتی نویزی داده‌های آموزشی را حفظ کنند:

VC-Dimension Capacity: High و Complexity Rademacher بزرگ.

- **مثال Bias–Variance Tradeoff:**

$$\text{Complexity}\uparrow \;\rightarrow\; \text{Bias}\downarrow,\ \text{Variance}\uparrow,\ \text{Memorization}\uparrow$$

اگر هیچ ضابطه‌ای (Regularization) وجود نداشته باشد، شبکه می‌تواند اطلاعات آموزشی را تقریباً کامل بازتولید کند.

(ج) **تعمیم (Generalization)**

تعمیم یعنی عملکرد خوب روی داده‌های ندیده. این امر با ترکیب مکانیزم‌های implicit و explicit regularization و Inductive Bias حاصل می‌شود:

$$L_{\text{reg}}(\theta) = L(\theta) + \lambda\|\theta\|_2^2$$

- **Implicit Regularization:** رفتار SGD شبکه را به سمت مینیمم‌های تخت (flat minima) هدایت می‌کند.
- **Regularization صریح:**
  - Weight Decay (L2): افزودن $\lambda\|\theta\|_2^2$ به تابع هزینه.
  - Dropout: غیرفعال‌سازی تصادفی نورون‌ها.
  - Early Stopping: پایان آموزش پیش از شروع شدید Overfitting.
- **Inductive Bias معماری:**

- CNN: اشتراک وزن‌ها و حساسیت به ویژگی‌های مکانی.
- RNN/Transformer: نگاشت توالی‌های زمانی و وابستگی‌های ترتیبی.

- **نرمال‌سازی (Batch Normalization)** کاهش حساسیت به مقیاس وزن‌ها و تثبیت جریان گرادیان.
- **داده‌های متنوع و کافی** کمیت و کیفیت داده‌های آموزشی پایه‌ی استخراج قاعده‌های عمومی و کاهش Overfitting است.

(د) **پیوند با «معادل بودن دو شبکه» و «شکل‌گیری دانش»**

شکل‌گیری دانش = پارامترهای بهینه θ که از فرآیند یادگیری به‌دست می‌آیند. *معادل بودن دو شبکه* وقتی است که:

$$\forall x,\ N(x) = M(x) \quad \text{Functional) (Exact}$$

یا با پرموتیشن π روی نورون‌ها (Structural Symmetry)، یا تقریباً:

$$\|N(x) - M(x)\|_p < \varepsilon \quad \text{یا} \quad \mathrm{KL}\big(N(x) \,\|\, M(x)\big) < \delta.$$

۴. در زیر سه نوع رایج از "توابع تبدیل نورونی" (یا به‌عبارت دیگر انواع نورون‌های مرتبه‌بالا و RBF) را به‌صورت ریاضی می‌بینید:

(آ) **نورون درجه دوم (Quadratic Neuron):**

$$\text{net}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\, x_i\, x_j + \sum_{i=1}^{n} v_i\, x_i + b,$$
$$y = f\big(\text{net}(\mathbf{x})\big),$$

که در آن

- $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$،
- w_{ij} ضرایب ضرب‌های درجه دوم،
- v_i ضرایب ترکیب خطی،
- b بایاس،
- $f(\cdot\cdot\cdot)$ تابع فعال‌سازی.

(ب) **نورون کروی (Spherical / RBF Neuron):**

$$\text{net}(\mathbf{x}) = \big\|\mathbf{x} - \boldsymbol{\mu}\big\| = \sqrt{\sum_{i=1}^{n}(x_i - \mu_i)^2}, \quad y = f\big(\text{net}(\mathbf{x})\big),$$

یا گونه‌ی مربعی بدون ریشه:

$$\text{net}(\mathbf{x}) = \sum_{i=1}^{n}(x_i - \mu_i)^2, \quad y = \exp\big(-\gamma\, \text{net}(\mathbf{x})\big),$$

که در آن

- $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$ مرکز نورون،
- $\gamma > 0$ ضریب پهنای باند،
- $f(\cdot)$ می‌تواند تابع خطی یا نمایی باشد.

(ج) **نورون چندجمله‌ای (Polynomial Neuron):** ابتدا ترکیب خطی و توان:

$$u = \sum_{i=1}^{n} w_i\, x_i + b, \quad y = u^d = \Big(\sum_{i=1}^{n} w_i\, x_i + b\Big)^d.$$

به‌صورت کلی برای ورودی چندبعدی:

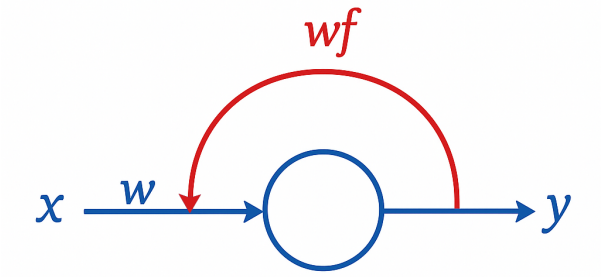
$$y = \sum_{\alpha_1+\cdots+\alpha_n \le d} w_{\alpha_1,\ldots,\alpha_n}\; x_1^{\alpha_1} \cdots x_n^{\alpha_n},$$

که در آن

- d درجه چندجمله‌ای،
- $\alpha_i \in \mathbb{N}_0$ اندیس‌های چندجمله‌ای،
- $w_{\alpha_1,\ldots,\alpha_n}$ ضرایب متناظر.

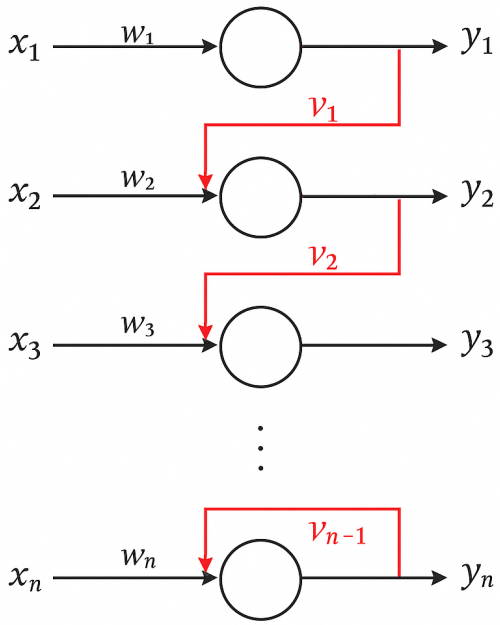
۵. در این تمرین ساختار دو نوع شبکه عصبی با فیدبک مورد بررسی قرار گرفته است:

- **شبکه عصبی تک‌نورونی با فیدبک به خود:** این نوع شبکه فقط شامل یک نورون است که خروجی آن دوباره به ورودی خودش بازمی‌گردد. این فیدبک باعث می‌شود که رفتار شبکه وابسته به حالت قبلی خروجی باشد. ساختار کلی آن در تصویر زیر نمایش داده شده است:



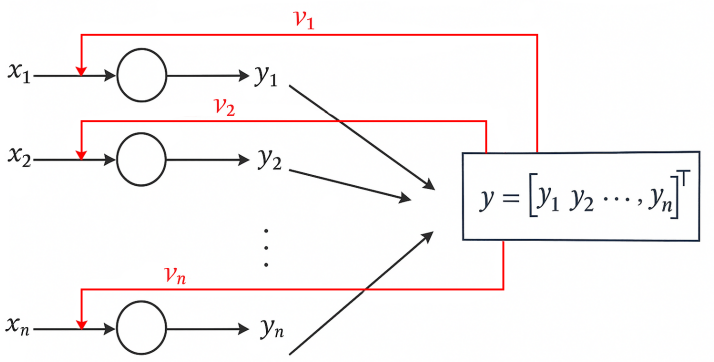
شکل ۱: شبکه عصبی تک‌نورونی با فیدبک به خود

- **شبکه عصبی تک‌لایه با فیدبک (ساختار اول):** در این ساختار چندین نورون در یک لایه وجود دارند و خروجی یکی از نورون‌ها به‌صورت فیدبک به ورودی خود یا سایر نورون‌ها داده می‌شود. تصویر زیر یکی از حالت‌های ممکن را نشان می‌دهد:



شکل ۲: شبکه عصبی تک‌لایه با فیدبک - حالت اول

- **شبکه عصبی تک‌لایه با فیدبک (ساختار دوم)**: در این حالت ممکن است فیدبک از خروجی کل شبکه به همه نورون‌ها اعمال شود. تصویر زیر نوعی دیگر از این ساختار را نمایش می‌دهد:



شکل ۳: شبکه عصبی تک‌لایه با فیدبک - حالت دوم

۶. (آ) **طراحی پرسپترون تک‌لایه**

فرض کنیم می‌خواهیم الگوها را به صورت برچسب $t_1 = -1$ برای P_1 و $t_2 = +1$ برای P_2 دسته‌بندی کنیم. باید $w \in \mathbb{R}^3$ و بایاس b را طوری بیابیم که

$$\begin{cases} \text{sign}(w^\top P_1 + b) = -1, \\ \text{sign}(w^\top P_2 + b) = +1. \end{cases}$$

این معادلات به صورت نابرابری‌های زیر نوشته می‌شوند:

$$w^\top(-1,-1,1) + b < 0, \quad w^\top(+1,-1,1) + b > 0.$$

به سادگی می‌توانیم مثلاً وزن‌ها را به صورت $w = (1,0,0)$ ، و بایاس $b = 0$ انتخاب کنیم:

$$w^\top P_1 + b = -1 < 0, \quad w^\top P_2 + b = +1 > 0.$$

لذا تابع تصمیم $y = \text{sign}(x_1)$ دو الگو را به درستی تفکیک می‌کند.

(ب) **طراحی شبکه Hamming**

شبکه همینگ برای N الگو P_k به صورت زیر است:

$$\mathbf{W} = \begin{bmatrix} P_1^\top \\ P_2^\top \end{bmatrix}, \quad y = \arg\max_k \big(\mathbf{W}\, x\big)_k.$$

برای P_1, P_2 داریم:

$$\mathbf{W} = \begin{pmatrix} -1 & -1 & 1 \\ +1 & -1 & 1 \end{pmatrix}, \quad \text{انتخاب } k \text{ با بیشینه‌ی } \sum_i W_{k,i} x_i .$$

(ج) طراحی شبکه Hopfield

شبکه هاپفیلد با الگوهای باینری ± 1 به کمک قاعده $T = \sum_k P_k P_k^\top$ ساخته می‌شود. اینجا داریم:

$$T = P_1 P_1^\top + P_2 P_2^\top = \begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & -2 \\ 0 & -2 & 2 \end{pmatrix}.$$

سپس حالت نرونی‌ها با قاعده $x_i \leftarrow \operatorname{sign}\left(\sum_j T_{ij} x_j\right)$ به سمت نزدیک‌ترین الگو جذب می‌شود.

۷. **(آ) طراحی مرز تصمیم و شبکه پرسپترون تک‌لایه**

با انتخاب وزن‌ها و بایاس زیر:

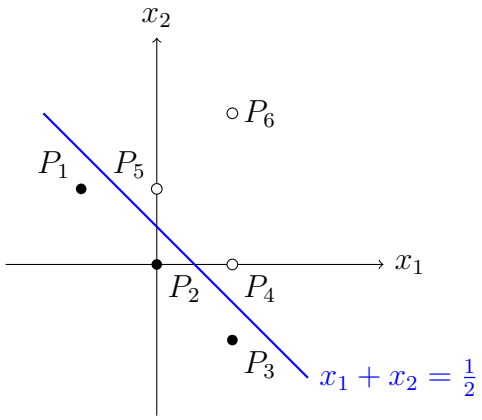
$$\mathbf{w} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad b = \frac{1}{2}$$

تابع فعال‌سازی گام به این صورت خواهد بود:

$$y = \begin{cases} 1, & \mathbf{w}^\top \mathbf{x} + b > 0, \\ 0, & \text{وگرنه}. \end{cases}$$

معادله مرز تصمیم:

$$-x_1 - x_2 + \frac{1}{2} = 0 \quad \Longleftrightarrow \quad x_1 + x_2 = \frac{1}{2}.$$



(ب) تشخیص قابلیت جداسازی و تعیین بازه ε

از نامعادلات زیر برای کلاس بندی استفاده می‌کنیم:

$$\begin{cases} -x_1 - x_2 + b > 0 & 1 \\ -x_1 - x_2 + b < 0 & 0 \end{cases}$$

نتیجه می‌شود که برای هر $\varepsilon \geq 0$ می‌توان $w_1 = w_2 = -1$ و $b \in (0,1)$ (مثلاً $b = 1$) را انتخاب کرد و جداسازی خطی امکان‌پذیر است.

(ج) اجرای الگوریتم پرسپترون و نتایج نهایی

برای سه مقدار ε اجرای الگوریتم با نرخ یادگیری $\eta = 1$، وزن و بایاس را از صفر مقداردهی کرده و تا خطای صفر تکرار می‌کنیم.

برنامهٔ ۱: پیاده‌سازی الگوریتم پرسپترون

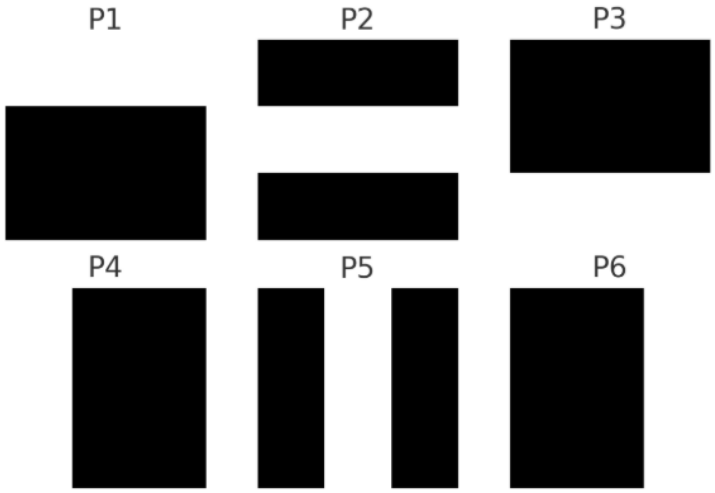
```python
import numpy as np

def perceptron_train(P, t, lr=1, max_epochs=1000):
  w = np.zeros(2)
  b = 0.0
  epc = 0
  for epoch in range(max_epochs):
    errors = 0
    for x, target in zip(P, t):
      y = 1 if np.dot(w, x) + b > 0 else 0
      if y != target:
        errors += 1
        update = lr * (target - y)
        w += update * x
        b += update
    if errors == 0:
      break
    epc = epoch
  return w, b, epc+1

epsilons = [1, 2, 6]
for eps in epsilons:
  P = [
    np.array([0,1]), np.array([1,-1]), np.array([-1,1]),
    np.array([1,eps]), np.array([1,0]), np.array([0,0])
  ]
  t = [0,1,1,0,0,1]
  w, b, epochs = perceptron_train(P, t)
  print(f" ={eps}: w={w}, b={b}, epochs={epochs}")
```

$$\varepsilon = 1: \quad \mathbf{w} = (-1, -1),\ b = 1,\ \text{epochs} = 2,$$
$$\varepsilon = 2: \quad \mathbf{w} = (-2, -2),\ b = 1,\ \text{epochs} = 4,$$
$$\varepsilon = 6: \quad \mathbf{w} = (-3, -4),\ b = 3,\ \text{epochs} = 4.$$

(د) خلاصه نتایج

- مرز تصمیم: $x_1 + x_2 = \frac{1}{2}$.
- بازه ε: $\varepsilon \geq 0$ (تمام مقادیر غیرمنفی).
- وزن‌ها و بایاس نهایی برای $\varepsilon = 1, 2, 6$ مطابق جدول فوق.
- الگوریتم پرسپترون حداکثر تا ۴ دور همگرا شده و خطای صفر حاصل شد.

۸. (آ) نمایش تصویری هر الگو به صورت یک تصویر 3×3 دودویی:



شکل ۴: الگوهای P_1 تا P_6 به صورت پیکسل‌های سفید و سیاه

(ب)
- برای تشخیص الگوهای P_1 تا P_6 از یکدیگر با استفاده از یک پرسپترون تک‌لایه با یک نرون، نیاز داریم که هر الگو را به صورت یک بردار ورودی مناسب برای نرون تبدیل کنیم. از آنجا که هر الگو یک ماتریس 3×3 است، می‌توانیم آن را به یک بردار 9×1 تبدیل کنیم. برای سادگی، فرض می‌کنیم که ماتریس را به صورت سطر به سطر به یک بردار ۹ عنصری تبدیل می‌کنیم.

- تبدیل الگوها به بردارهای ورودی:

$$
\begin{aligned}
P_1 &= [1,1,1,-1,-1,-1,-1,-1,-1],\\
P_2 &= [-1,-1,-1,1,1,1,-1,-1,-1],\\
P_3 &= [-1,-1,-1,-1,-1,-1,-1,1,1],\\
P_4 &= [1,-1,-1,1,-1,-1,1,-1,-1],\\
P_5 &= [-1,1,-1,1,-1,1,-1,1,-1],\\
P_6 &= [-1,-1,1,-1,-1,1,-1,-1,1].
\end{aligned}
$$

- هدف: جداسازی دو کلاس با پرسپترون تک‌نرونی:
  - کلاس ۱: P_1, P_2, P_3 (تارگت $t=1$)
  - کلاس ۰: P_4, P_5, P_6 (تارگت $t=0$)
- مشخصات پرسپترون:
  - تعداد ورودی: ۹
  - تعداد نرون: ۱
  - تابع فعال‌سازی: تابع پله با آستانه صفر
  - خروجی $y = \begin{cases} 1, & \text{اگر } \text{net} \geq 0, \\ 0, & \text{اگر } \text{net} < 0. \end{cases}$
- مقادیر اولیه:
  - وزن‌ها: $w^{(0)} = [0,0,0,0,0,0,0,0,0]$
  - بایاس: $b^{(0)} = 0$
  - نرخ یادگیری: $\eta = 1$
- قانون به‌روزرسانی:

$$
\begin{aligned}
w^{(\text{new})} &= w^{(\text{old})} + \eta\,(t-y)\,x,\\
b^{(\text{new})} &= b^{(\text{old})} + \eta\,(t-y).
\end{aligned}
$$

- مراحل یادگیری (اپوک‌ها):

i. اپوک ۱:

  - الگو P_1: محاسبه $\text{net}=0$، $y=1$، خطا $e=0$ (بدون تغییر)
  - الگو P_2 و P_3: مشابه، خطا صفر (بدون تغییر)
  - الگو P_4: $\text{net}=0$، $y=1$، $e=-1$، به‌روزرسانی:

$$
\begin{aligned}
w &= w + (-1)\,P_4 = [-1,1,1,-1,1,1,-1,1,1],\\
b &= -1.
\end{aligned}
$$

– الگو P_5: محاسبه $\text{net}=0$، $y=1$، $e=-1$، به‌روزرسانی:

$$w=[0,0,2,-2,2,0,0,0,2],$$
$$b=-2.$$

– الگو P_6: $\text{net}=2$، $y=1$، $e=-1$، به‌روزرسانی تا پایان اپوک ۱:

$$w=[1,1,1,-1,3,-1,1,1,1],$$
$$b=-3.$$

ii. اپوک ۲:

– ... (ادامه تا همگرایی)

iii. ...

iv. اپوک ۶ (نهایی): وزن‌ها و بایاس نهایی:

$$w=[0,0,-2,-2,6,-4,-4,0,-2],\quad b=0.$$

- نحوه تشخیص الگوها: با وزن‌ها و بایاس نهایی، خروجی ۱ برای P_1,P_2,P_3 و خروجی ۰ برای P_4,P_5,P_6 خواهد بود.

طراحی یک پرسپترون تک‌لایه با یک نورون برای جدا کردن این الگوها:

- وزن‌ها و بایاس اولیه:

$$w_1=w_2=\cdots=w_9=0,\quad b=0.$$

- تابع فعال‌سازی: $y=\text{sign}\big(\mathbf{w}^\top\mathbf{x}+b\big)\in\{+1,-1\}$.
- قانون یادگیری پرسپترون:

$$w_i\leftarrow w_i+\eta\,(d-y)\,x_i,\quad b\leftarrow b+\eta\,(d-y),$$

که η نرخ یادگیری، d برچسب هدف و x_i مؤلفهٔ iام ورودی است.

برنامهٔ ۲: پیاده‌سازی پرسپترون برای سوال ۸

```
# perceptron8.py
import numpy as np

patterns = np.array([
    [ 1,  1,  1, -1, -1, -1, -1, -1, -1],  # P1
    [-1, -1, -1,  1,  1,  1, -1, -1, -1],  # P2
    [-1, -1, -1, -1, -1, -1,  1,  1,  1],  # P3
    [ 1, -1, -1,  1, -1, -1,  1, -1, -1],  # P4
```

```
    [-1,  1, -1,  1, -1,  1, -1,  1, -1],  # P5
    [-1, -1,  1, -1, -1,  1, -1, -1,  1],  # P6
])

num_classes = patterns.shape[0]
num_features = patterns.shape[1]

targets = -np.ones((num_classes, num_classes))
for i in range(num_classes):
    targets[i, i] = 1

W = np.zeros((num_classes, num_features))
b = np.zeros(num_classes)
eta = 0.1

for i in range(num_classes):
    converged = False
    while not converged:
        converged = True
        for k, x in enumerate(patterns):
            d = targets[i, k]
            z = np.dot(W[i], x) + b[i]
            y = 1 if z >= 0 else -1
            if y != d:
                W[i] += eta * (d - y) * x
                b[i] += eta * (d - y)
                converged = False

for i in range(num_classes):
    print(f"Perceptron {i+1}: w = {W[i]}, b = {b[i]}")
```

در پایان آموزش تا همگرایی، وزن‌ها و بایاس نهایی به صورت زیر به دست می‌آیند:

$$\mathbf{w}_{\text{final}} = [\, w_1^*, w_2^*, \ldots, w_9^* \,], \quad b_{\text{final}} = b^*.$$

۹. برای تقسیم $\mathbb{R}^2$ به m ناحیهٔ مجزا با یک شبکهٔ دو لایه، باید k نورون در لایهٔ پنهان داشته باشیم به طوري که حداکثر تعداد ناحیه‌هاي ایجادشده توسط k خط برابر باشد با:

$$N(k) = \frac{k(k+1)}{2} + 1.$$

پس باید عدد k کمترین عدد صحیح باشد که

$$N(k) \geq m \quad \Longrightarrow \quad \frac{k(k+1)}{2} + 1 \geq m.$$

با حل این نامعادله، خواهیم داشت:

$$k^2 + k - 2(m-1) \geq 0 \quad \Longrightarrow \quad k \geq \frac{-1 + \sqrt{1 + 8(m-1)}}{2}.$$

بنابراین حداقل تعداد نورون‌هاي لایهٔ پنهان:

$$k = \left\lceil \frac{-1 + \sqrt{8m-7}}{2} \right\rceil.$$

اثبات فرمول حداکثر تعداد ناحیه‌ها با استفاده از استقرا:

قضیه: با k خط در صفحه حداکثر $N(k) = k(k+1)/2 + 1$ ناحیه متناظر ایجاد مي‌شود.

پایهٔ استقرا ($k = 0$): با صفر خط در صفحه، تمام صفحه یک ناحیهٔ یکپارچه است. بنابراین

$$N(0) = \frac{0 \cdot 1}{2} + 1 = 1,$$

که صحیح است.

فرض استقرا: فرض کنیم براي k خط، فرمول

$$N(k) = \frac{k(k+1)}{2} + 1$$

برقرار باشد.

گام استقرا ($k \to k+1$): اگر یک خط تازه به مجموعهٔ k خط اضافه کنیم، این خط جدید با هر یک از k خط قبلي در یک نقطه تلاقی مي‌کند، بنابراین در مجموع k نقطهٔ تلاقی ایجاد مي‌شود. این k نقطه، خط جدید را به $k+1$ قطعه تقسیم مي‌کند. هر قطعه یک ناحیهٔ جدید ایجاد مي‌کند. بنابراین:

$$N(k+1) = N(k) + (k+1).$$

با جاگذاري فرض استقرایي داریم:

$$N(k+1) = \frac{k(k+1)}{2} + 1 + (k+1) = \frac{k(k+1) + 2(k+1)}{2} + 1 = \frac{(k+1)(k+2)}{2} + 1.$$

بنابراین فرمول براي $k+1$ نیز برقرار است.

این اثبات کامل است و نتیجه مي‌دهد که براي تقسیم $\mathbb{R}^2$ به m ناحیه، حداقل

$$\left\lceil \frac{-1+\sqrt{8m-7}}{2} \right\rceil$$

نورون در لایهٔ پنهان نیاز داریم.

۲ کدزنی و پیاده سازی

۱.۲ بخش‌های ۱ تا ۴

۱.۱.۲ مقدمه

هدف این پروژه، پیاده‌سازی کامل یک شبکه عصبی از پایه و بدون استفاده از کتابخانه‌های آماده مانند TensorFlow یا PyTorch برای اهداف آموزشی است. این پروژه شامل مراحل مختلفی از جمله رگرسیون لجستیک، شبکه با لایه پنهان و در نهایت طبقه‌بندی چندکلاسه می‌باشد.

۲.۱.۲ پیش‌پردازش داده‌ها

برای انجام این پروژه، از مجموعه داده CIFAR-10 استفاده شده است. تصاویر با استفاده از توابع موجود در torchvision بارگذاری شده و به آرایه‌های CuPy تبدیل گردیده‌اند تا از قدرت محاسباتی GPU بهره‌برداری شود.
دیتاست به دو صورت مختلف برای وظایف دودویی و چندکلاسه پردازش شد:

- طبقه‌بندی دودویی: تصاویر هواپیما (airplane) با برچسب صفر و سایر کلاس‌ها با برچسب یک.
- طبقه‌بندی چندکلاسه: برچسب‌ها به صورت one-hot برای ده کلاس مختلف نگاشته شدند.

۳.۱.۲ بخش اول: رگرسیون لجستیک

در این مرحله، هدف تشخیص اینکه آیا یک تصویر متعلق به کلاس هواپیما است یا خیر می‌باشد. مدل تنها شامل یک لایه است با تابع فعال‌سازی سیگموئید:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
$$\sigma' = \sigma \times (1 - \sigma)$$

$$\hat{y} = \sigma(Wx + b)$$

تابع خطای مورد استفاده، Binary Cross-Entropy است:

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

۴.۱.۲ آموزش مدل با گرادیان کاهشی

مدل `LogisticRegression` برای یادگیری رگرسیون لجستیک با استفاده از گرادیان کاهشی پیاده‌سازی شده است. مراحل کلیدی شامل موارد زیر است:

- **مقداردهی اولیه**: وزن‌ها با مقادیر کوچک تصادفی و بایاس با صفر مقداردهی می‌شوند.
- **پیش‌بینی**: محاسبه $z = XW + b$ و سپس $a = \sigma(z)$.

- **محاسبه گرادیان:**

$$dz = \hat{y} - y, \quad dW = \frac{1}{m} X^T dz, \quad db = \frac{1}{m} \sum dz$$

- **به‌روزرسانی پارامترها:**

$$W \leftarrow W - lr \cdot dW, \quad b \leftarrow b - lr \cdot db$$

- **آموزش:** اجرای مراحل فوق به‌صورت mini-batch در طول اپوک‌ها و ثبت دقت و زیان.
- **پیش‌بینی نهایی:** با آستانه ۰.۵، خروجی دودویی تولید می‌شود.

کلاس `SimpleLoader` برای تقسیم داده‌ها به دسته‌های تصادفی طراحی شده است.
پارامترها با گرادیان نزولی به‌روزرسانی شدند.
مدل با معیارهای مختلف بر روی داده آزمون ارزیابی شده است. برای تعیین بهترین آستانه طبقه‌بندی، جستجویی میان ۰.۱ تا ۰.۹ انجام شد.

نتایج ارزیابی پس از ۲۰۰ اپوک با lr=0.0001 و batch size=512:

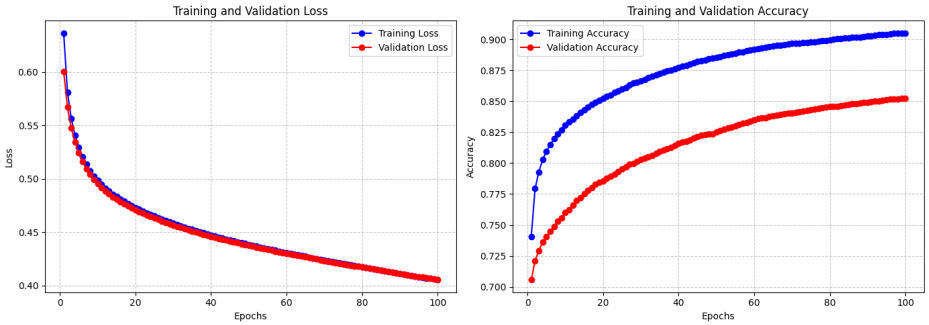
- **بهترین آستانه:** 0.50
- **بهترین F۱-score:** 0.4460
- **دقت آزمون:** 0.8872

ماتریس درهم‌ریختگی:

Airplane Pred	Other Pred	
۵۸۲	۸۴۱۸	Other True
۴۵۴	۵۴۶	Airplane True

گزارش طبقه‌بندی:

support	f۱-score	recall	precision	
9000	0.9372	0.9353	0.9391	Other
1000	0.4460	0.4540	0.4382	Airplane
10000	**0.8872**			**accuracy**
10000	0.6916	0.6947	0.6887	avg macro
10000	0.8881	0.8872	0.8890	avg weighted



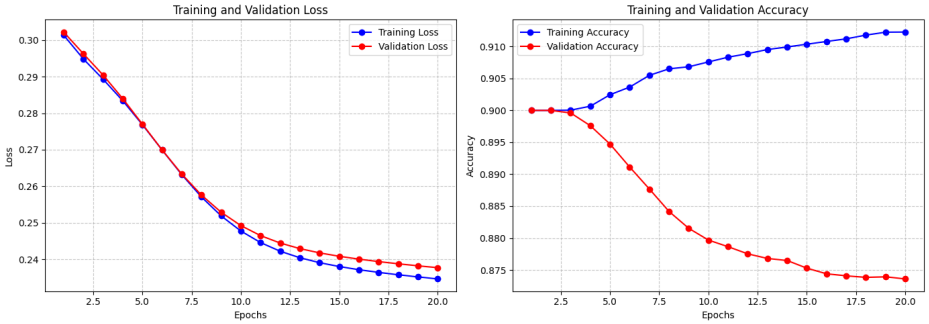
شکل ۵: تغییرات دقت و هزینه در طی آموزش با مدل رگرسیون لجستیک

۵.۱.۲ بخش دوم: شبکه با یک لایه پنهان

در این بخش، مدل با افزودن یک لایه پنهان با ۶۴ نورون توسعه یافت. معماری شبکه به صورت زیر است:

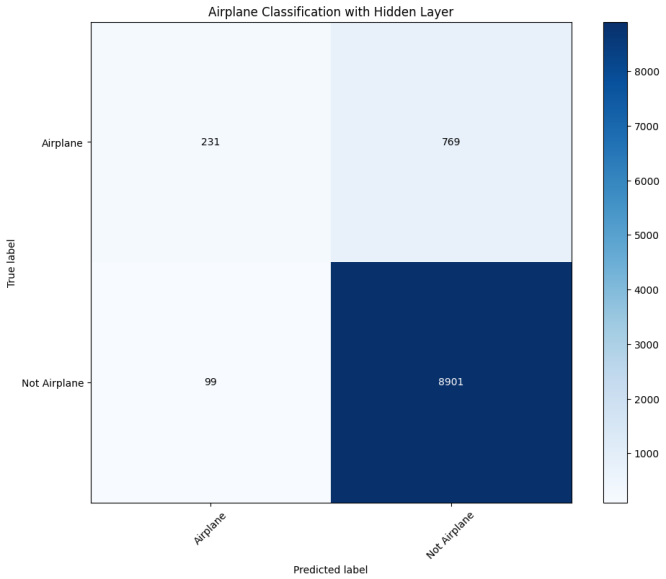
$$Z_1 = W_1 X + b_1$$
$$A_1 = \sigma(Z_1)$$
$$Z_2 = W_2 A_1 + b_2$$
$$A_2 = \sigma(Z_2)$$

در اینجا نیز از سیگموئید به عنوان تابع فعال‌سازی استفاده شده و آموزش با الگوریتم گرادیان نزولی انجام شده است.



شکل ۶: تغییرات دقت و هزینه در طی آموزش با مدل رگرسیون لجستیک چند لایه

```
Classification Report:
              precision    recall  f1-score   support

Airplane         0.7000    0.2310    0.3474      1000
Not Airplane     0.9205    0.9890    0.9535      9000

accuracy                             0.9132     10000
macro avg        0.8102    0.6100    0.6504     10000
weighted avg     0.8984    0.9132    0.8929     10000
```



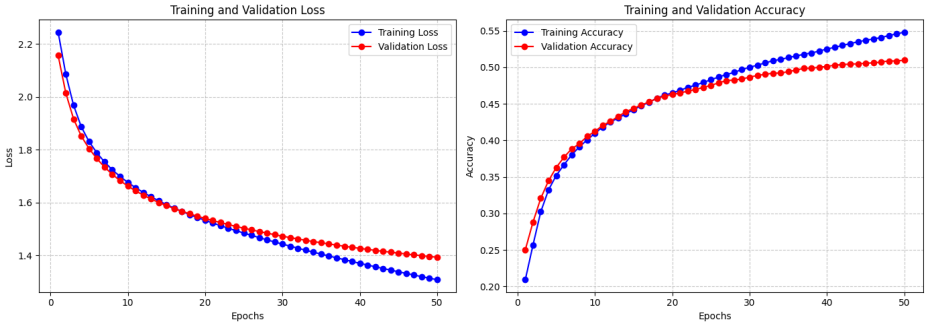
شکل ۷: ماتریس آشفتگی برای مدل رگرسیون لاجستیک چند لایه بعد ۲۰ ایپاک

۶.۱.۲ بخش سوم: طبقه‌بندی چندکلاسه

در این مرحله، شبکه برای شناسایی تمامی ۱۰ کلاس CIFAR-10 طراحی شد. خروجی شبکه دارای ۱۰ نورون با تابع فعال‌سازی Softmax است:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}$$

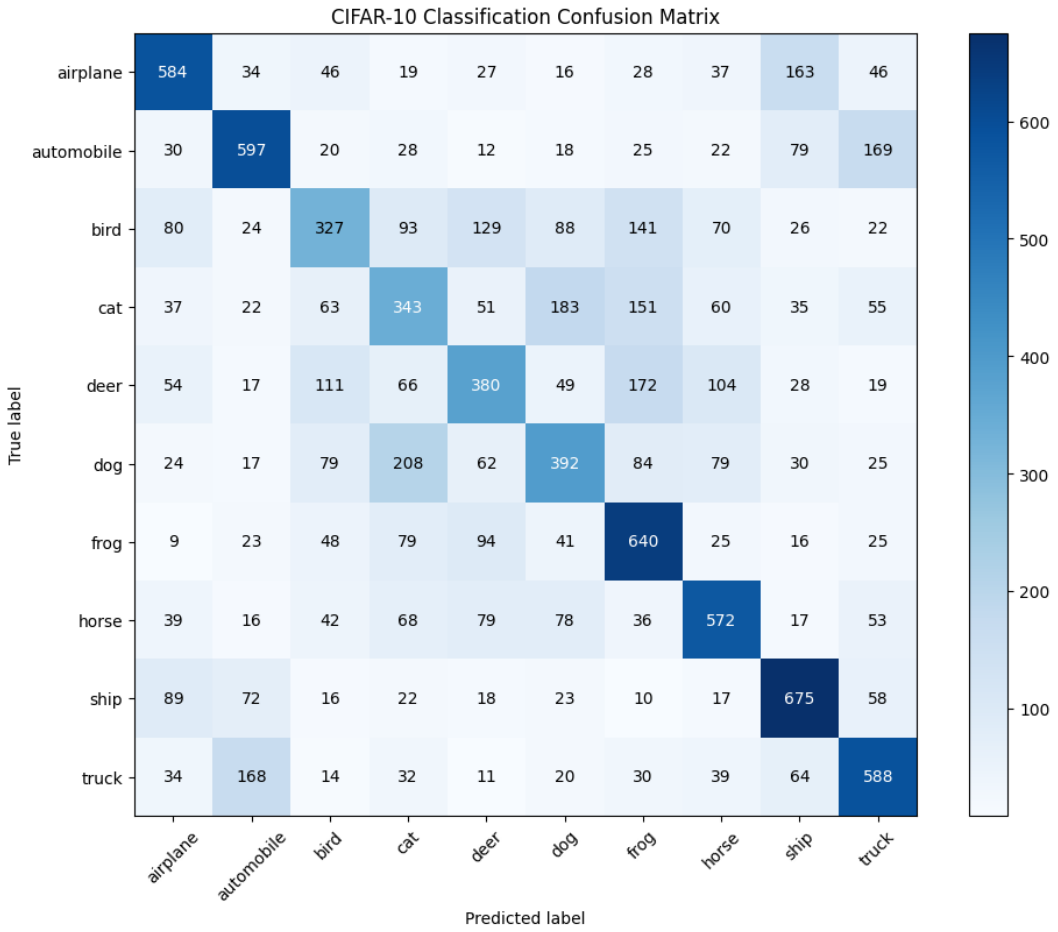
تابع هزینه مورد استفاده، Categorical Cross-Entropy است.



شکل ۸: تغییرات دقت و هزینه در طی آموزش با مدل رگرسیون وانهات چند لایه

```
    Classification Report:
    precision                    recall  f1-score   support

    airplane          0.5959    0.5840     0.5899      1000
    automobile        0.6030    0.5970     0.6000      1000
    bird              0.4269    0.3270     0.3703      1000
    cat               0.3580    0.3430     0.3504      1000
```

```
    deer          0.4403    0.3800    0.4079      1000
    dog           0.4317    0.3920    0.4109      1000
    frog          0.4860    0.6400    0.5524      1000
    horse         0.5580    0.5720    0.5649      1000
    ship          0.5958    0.6750    0.6329      1000
    truck         0.5547    0.5880    0.5709      1000

    accuracy                          0.5098     10000
    macro avg     0.5050    0.5098    0.5051     10000
    weighted avg  0.5050    0.5098    0.5051     10000
```



شکل ۹: ماتریس آشفتگی برای مدل رگرسیون وان هات چند لایه بعد ۵۰ ایپاک

۷.۱.۲ بخش چهارم: مقایسه‌ی ویژگی‌های شبکه عصبی نیمه پیشرفته

در این بخش، یک شبکه عصبی نیمه پیشرفته با قابلیت استفاده از بهینه‌سازهای مختلف و توابع فعال‌سازی گوناگون پیاده‌سازی شده است. ساختار شبکه به‌گونه‌ای طراحی شده که از وزن‌دهی اولیه مناسب، ذخیره‌سازی گرادیان‌ها، و بهینه‌سازهای مدرن مانند

Adam پشتیبانی می‌کند.

۸.۱.۲ توابع فعال‌سازی

توابع فعال‌سازی نقش مهمی در آموزش شبکه‌های عصبی ایفا می‌کنند. در این پروژه از سه تابع فعال‌سازی پرکاربرد شامل ReLU، Sigmoid، و Tanh استفاده شده است.

- **تابع Sigmoid:**

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

- **تابع Tanh:**

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

- **تابع ReLU:**

$$f(x) = \max(0,x), \quad f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

در تحلیل تجربی، تابع ReLU به‌دلیل سادگی محاسباتی و اجتناب از مشکل ناپدید شدن گرادیان‌ها عملکرد برتری نسبت به سایر توابع نشان داد.

۹.۱.۲ الگوریتم‌های بهینه‌سازی

در این پروژه سه الگوریتم زیر پیاده‌سازی و بررسی شده‌اند:

۱. SGD: به‌روزرسانی وزن‌ها به‌صورت مستقیم و بر پایه گرادیان فعلی صورت می‌گیرد:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

که در آن η نرخ یادگیری و $J(\theta)$ تابع هزینه است.

۲. Momentum: این روش برای بهبود همگرایی، از سرعت قبلی استفاده می‌کند:

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta J(\theta) \quad , \quad \theta = \theta + v_t$$

که در آن γ ضریب مومنتوم است.

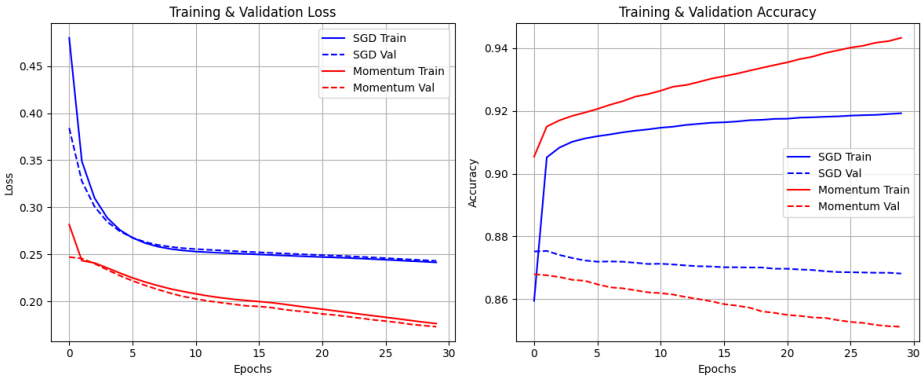
۳. . Adam: این روش تخمین‌هایی از میانگین و واریانس لحظه‌ای گرادیان‌ها ذخیره می‌کند:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$



شکل ۱۰: مقایسه‌ی همگرایی SGD و تکانه با فعال‌سازهای یکسان در دو لایه در داده‌های تست و آموزش

۱۰.۱.۲ مقداردهی اولیه وزن‌ها

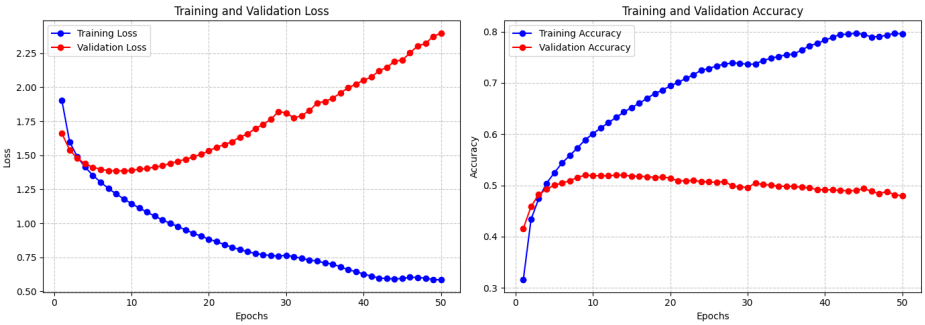
برای بهبود عملکرد یادگیری، از دو روش مقداردهی اولیه استفاده شده است:

- **Initialization He** برای توابع ReLU:

$$\mathrm{Var}(w) = \frac{2}{n_{\text{in}}}$$

- **Initialization Xavier** برای توابع Sigmoid و Tanh:

$$\mathrm{Var}(w) = \frac{1}{n_{\text{in}}}$$

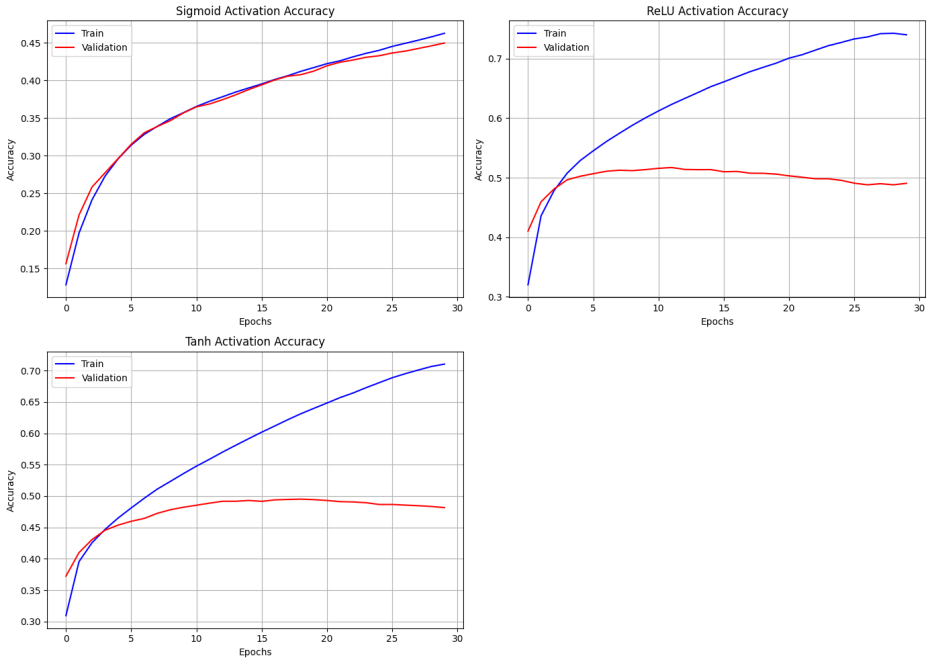


شکل ۱۱: اثر مقداردهی اولیه He در یک شبکه دو لایه با بهینه‌ساز تکانه در ۳۰ ایپاک

۱۱.۱.۲ مقایسه تجربی توابع فعال‌سازی

در بخش سوم، برای مقایسه عملکرد توابع فعال‌سازی، سه شبکه با ساختار یکسان ولی با توابع Sigmoid، ReLU و Tanh آموزش داده شد. نتایج نشان دادند که:

- ReLU منجر به همگرایی سریع‌تر و دقت بالاتر در داده‌های تست شد.
- Sigmoid به‌دلیل ناپدید شدن گرادیان در لایه‌های عمیق ضعیف‌ترین عملکرد را داشت.
- Tanh عملکردی بین دو تابع دیگر ارائه داد.



شکل ۱۲: مقایسه‌ی همگرایی توابع فعالسازی متفاوت با ۳۰ ایپاک

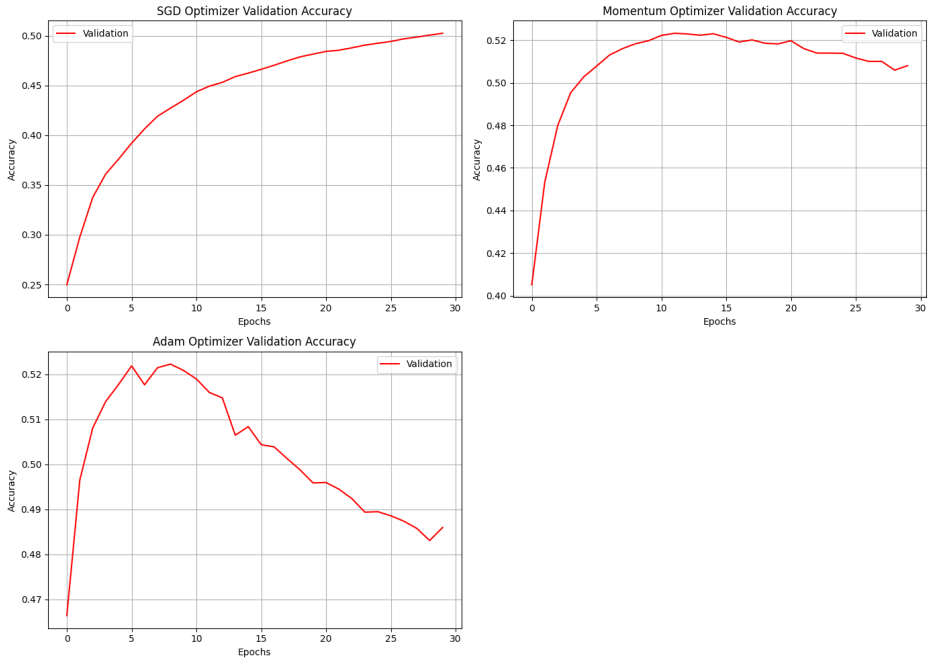
نتایج در قالب نمودارهای دقت در طول اپوک‌ها رسم گردیدند.

۱۲.۱.۲ مقایسه تجربی بهینه‌سازها

در بخش پایانی، عملکرد سه الگوریتم SGD، Momentum و Adam بر روی یک شبکه سه‌لایه آزمایش شد. نتایج حاکی از آن بود که:

- Adam در اکثر موارد سریع‌تر همگرا شد و به دقت بالاتری رسید.
- Momentum پایداری بهتری نسبت به SGD داشت.
- استفاده از بهینه‌ساز مناسب نقش مهمی در کیفیت آموزش دارد.

مدل طراحی‌شده در این پروژه به‌دلیل ساختار ماژولار و قابل گسترش، قابلیت استفاده در کاربردهای مختلف را دارد. نتایج تجربی نشان دادند که انتخاب دقیق تابع فعال‌سازی، مقداردهی اولیه مناسب، و الگوریتم بهینه‌سازی مؤثر نقش کلیدی در موفقیت مدل ایفا می‌کند.



شکل ۱۳: مقایسه‌ی همگرایی بین بهینه‌سازهای مختلف در ۳۰ ایپاک

۲.۲ بخش ۵: آموزش مدل از طریق شبکه عصبی CNN

۱.۲.۲ تنظیم داده‌ها و بارگذاری CIFAR-۱۰

مجموعه‌داده CIFAR-۱۰ را با نرمال‌سازی مناسب بارگذاری کنید و هایDataLoader آموزش و تست را با اندازه بچ و شافل دلخواه ایجاد نمایید.

برنامهٔ ۳: تعریف transforms و DataLoader

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyper-parameters
num_epochs = 20
batch_size = 128
learning_rate = 0.01

# CIFAR-10 dataset transforms
transform = transforms.Compose([
    transforms.ToTensor(),
```

```
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))
])

# CIFAR-10 dataset
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
    transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
    transform=transform)
print(train_dataset.classes)
# Data loaders
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
    shuffle=True, num_workers=4)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
    shuffle=False, num_workers=4)
```

۲.۲.۲ تعریف معماری ساده CNN

یک کلاس PyTorch از نوع nn.Module با ساختار زیر پیاده‌سازی کنید:

- Conv2d با ۳۲ فیلتر، کرنل 3×3، فعال‌سازی ReLU
- MaxPool2d با کرنل 2×2
- Conv2d با ۶۴ فیلتر، کرنل 3×3، فعال‌سازی ReLU
- MaxPool2d با کرنل 2×2
- Flatten() و دو لایه تمام‌متصل ۱۲۸ و ۱۰ نورونی Softmax) خروجی)

برنامهٔ ۴: کلاس SimpleCNN

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
```

```
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

۳.۲.۲ حلقه آموزش مدل

حلقه آموزشی را با تابع هزینه CrossEntropyLoss و بهینه‌ساز SGD با Momentum پیاده کنید. در هر صد گام آموزشی، میانگین loss را چاپ نمایید.

برنامهٔ ۵: حلقه آموزش train loop

```
if __name__ == "__main__":
    model = SimpleCNN().to(device)

    # Loss and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)

    # Training loop
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for i, (images, labels) in enumerate(train_loader):
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            if (i + 1) % 100 == 0:
                print(f'Epoch [{epoch+1}/{num_epochs}], Step
   [{i+1}/{len(train_loader)}], Loss: {running_loss / 100:.4f}')
                running_loss = 0.0
```

۴.۲.۲ ارزیابی مدل و گزارش دقت

مدل آموزش‌دیده را روی مجموعه تست ارزیابی کنید و دقت نهایی را محاسبه و چاپ کنید. سپس وزن‌های مدل را ذخیره نمایید.

برنامهٔ ۶: ارزیابی و ذخیره مدل

```
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Test Accuracy of the model on the 10000 test images: {100 * correct / total:.2f}%')

    # Save the model checkpoint
    torch.save(model.state_dict(), 'simple_cnn_cifar10.pth')
```

۵.۲.۲ گزارش معماری و تحلیل نتایج

یک گزارش مختصر بنویسید که شامل معماری نهایی، جزییات پیاده‌سازی، روند آموزش و نتایج ارزیابی باشد. همچنین مزایا و معایب استفاده از CNN را در مقایسه با پرسپترون چندلایه مورد بحث قرار دهید.

۶.۲.۲ مزایا و معایب استفاده از CNN در مقابل یک پرسپترون چندلایه

مزایا:

- **استخراج ویژگی‌های مکانی:** لایه‌های کانولوشنال با حفظ ساختار دوبعدی تصویر، الگوهای محلی مانند لبه‌ها و بافت‌ها را بهتر استخراج می‌کنند، در حالی که MLP تصویر را صاف‌شده دریافت می‌کند.
- **کاهش پارامترها:** با اشتراک وزن در فیلترها، تعداد پارامترها به‌طور قابل‌توجهی کمتر از MLP بوده و خطر بیش‌برازش کاهش می‌یابد.
- **مقیاس‌پذیری:** استفاده از pooling و فیلترهای محلی باعث می‌شود CNN بتواند روی تصاویر بزرگ‌تر یا پیچیده‌تر نیز کارایی مناسبی داشته باشد.
- **تعمیم بهتر:** CNN در برابر تغییرات کوچک در تصویر (جابجایی، چرخش، تغییر نور) مقاوم‌تر است و به همین دلیل در داده‌های بصری عملکرد بهتری دارد.

معایب:

- **پیچیدگی پیاده‌سازی:** طراحی معماری CNN اعم از انتخاب تعداد لایه‌ها، فیلترها و سایر ابرپارامترها نیازمند تجربه و آزمون‌وخطای بیشتری است.

- **زمان آموزش:** محاسبات کانولوشن هزینه‌برتر از ضرب‌های برداری ساده در MLP بوده و ممکن است زمان آموزش طولانی‌تری داشته باشد.
- **وابستگی به GPU:** برای آموزش سریع و مؤثر CNN معمولاً نیاز به شتاب‌دهنده‌هایی مانند GPU است، در حالی که MLP را می‌توان حتی روی CPU اجرا کرد.