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

۰ 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 $L\big(y_{\text{pred}},\, y_{\text{true}}\big)$، 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. **(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

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

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 $y = \mathrm{sign}(x_1)$ 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.

۴. (آ) طراحی مرز تصمیم و شبکه پرسپترون تک‌لایه

با انتخاب وزن‌ها و بایاس زیر:

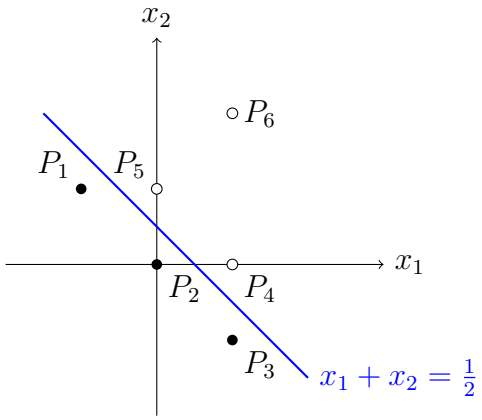
$$\mathbf{w} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad b = \frac{1}{2}$$

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.

۲ کدزنی و پیاده سازی