

شبکه‌های عصبی و یادگیری عمیق
دکتر صفابخش



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین اول

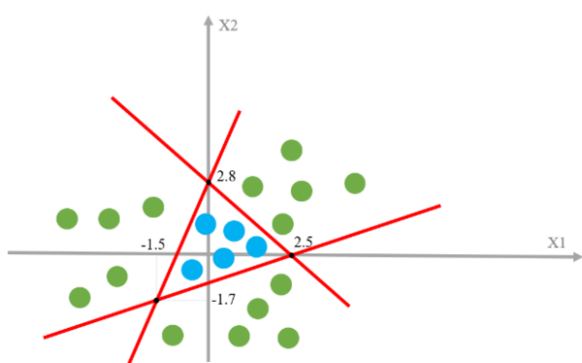
شبکه Perceptron و Adalin

۴ فروردین ۱۴۰۳

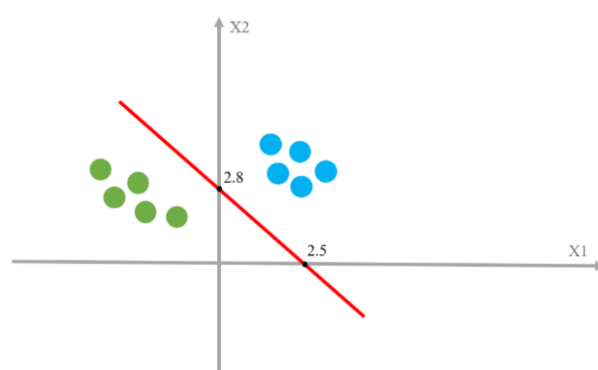


سوال اول - نظری

همانگونه که در کلاس درس آشنا شده‌اید، واحد پردازشی پرسپترون و آدالین امکان دریافت ورودی، توان‌های متعدد آن و حاصل ضرب ورودی‌ها را داشته و می‌تواند مسئله دسته‌بندی خطی را حل نمایند. در این سوال، قصد بدست آوردن وزن‌های یک نرون پردازشی پرسپترون را به صورت نظری و با محاسبات دستی داریم.



شکل ۱-ب



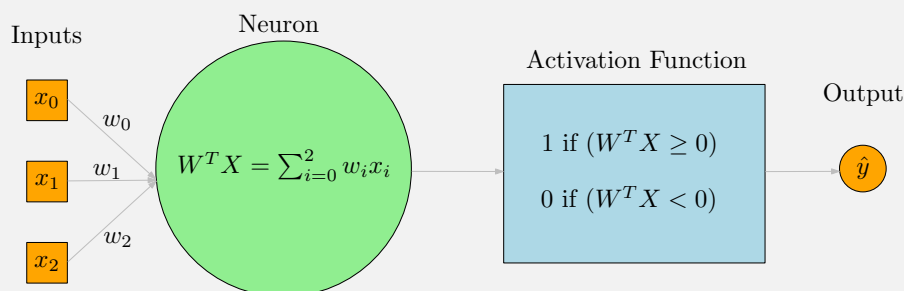
شکل ۱-ا

شکل ۱: مسئله مورد بحث

۱. شکل ۱-ا را برای دسته بندی مسئله دودویی در نظر بگیرید. معماری نرون مورد نظر را توضیح داده و وزن‌های آن را بدست آورید.

پاسخ

به دلیل آنکه داده های این قسمت جداپذیر خطی هستند، می‌توان برای طبقه بندی آنها، از شبکه پرسپترون تک لایه استفاده کرد. معماری این شبکه به صورت «شکل ۲» است.



شکل ۲: معماری شبکه پرسپترون تک لایه

پاسخ

در این شبکه، ورودی/خروجی‌ها با مربع‌های نارنجی، نورون‌ها با دایره سبز و تابع فعال ساز با مربع آبی نشان داده شده است. تعداد دیتا ورودی شبکه ۲ است. x_1 و x_2 . بایاس این شبکه با x_0 نشان داده شده است. وزن‌های شبکه نیز با w نشان داده شده است. بنابراین بردار ورودی و وزن‌های شبکه به صورت زیر است:

$$X = \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \end{bmatrix} \quad W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

طبق تئوری شبکه‌های عصبی می‌دانیم خروجی نرون به صورت زیر محاسبه می‌شود: (در اینجا برای انجام محاسبات ساده، تابع فعال‌ساز در نظر گرفته نشده است)

$$\hat{y} = W^T X = \sum_{i=0}^2 w_i x_i = w_0 x_0 + w_1 x_1 + w_2 x_2 \xrightarrow{x_0=1} w_0 + w_1 x_1 + w_2 x_2$$

طبق شکل a-1 دو نقطه از خط جداکننده دو کلاس را داریم. بنابراین می‌توان معادله خط را به صورت زیر نوشت. می‌دانیم معادله خط به صورت زیر تعریف می‌شود:

$$y - y_0 = m(x - x_0)$$

که در آن m شیب خط است و به صورت زیر $\frac{\Delta y}{\Delta x}$ تعریف می‌شود. با جاگذاری یک از نقاط در معادله خط، می‌توان معادله خط را بدست آورد.

$$P_1 = \begin{bmatrix} 2.5 \\ 0 \end{bmatrix} \quad P_2 = \begin{bmatrix} 0 \\ 2.8 \end{bmatrix}$$

$$m = \frac{2.8 - 0}{0 - 2.5} = -1.12 \rightarrow y - 0 = -1.12(x - 2.5) \rightarrow \boxed{y = -1.12x + 2.8}$$

حالا اگر معادله خروجی نورون را به صورت زیر مرتب کنیم، می‌توان از مقایسه با معادله خط بدست آمده وزن‌های شبکه را تعیین کرد.

$$x_1 = \frac{-w_2}{w_1} x_2 - \frac{w_0}{w_1}, \quad x_2 = \frac{-w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

در اینجا به دلیل آنکه دو معادله و ۳ مجهول (w_0, w_1, w_2) داریم، نیاز است که یکی از وزن‌ها را فرض کرده و دو وزن دیگر را بدست آورد.

$$\begin{aligned} \frac{-w_2}{w_1} &= -1.12 \rightarrow w_2 = 1.12w_1 \\ \frac{-w_0}{w_1} &= 2.8 \rightarrow w_0 = -2.8w_1 \\ \rightarrow \begin{cases} w_2 - 1.12w_1 = 0 \\ -2.8w_1 - w_0 = 0 \end{cases} & \text{assume } w_0 = 2.8 \rightarrow \boxed{w_1 = -1, w_2 = -1.12} \end{aligned}$$

ذکر این نکته الزامیست که این جواب، یکتا نمی‌باشد و برحسب اینکه مقدار w_0 را چه انتخاب کنیم، مقدار ۲ وزن دیگر متفاوت می‌شود.

۲. حال شکل b-1 را در نظر بگیرید. چرا مسئله جداپذیر خطی نیست؟ چگونه می‌توان آن را در قالب حل چند مسئله خطی حل نمود؟ معماری پیشنهادی خودتان را رسم و وزن‌های موجود در آن را با انجام محاسبات بدست آورید. معماری شما می‌تواند حاصل از کنار هم چیدن و پشت هم چیدن یک یا چند نورون پرسپترونی باشد.

پاسخ

به مسائلی جداپذیر خطی گفته می‌شود که بتوان داده‌ها (کلاس‌ها) را با استفاده از فقط یک خط از هم جدا کرد. در این مثال، دو کلاس آبی و سبز را نمی‌توان فقط با یک خط از هم جدا کرد. بنابراین این مسئله جداپذیر خطی نمی‌باشد. برای حل این مسئله، چندین روش وجود دارد که در ادامه آنها را توضیح خواهیم داد.

(آ) افزایش ابعاد ویژگی‌های مسئله:

یعنی در این مثال که مسئله مورد بحث ما دو بعدی است، یک بعد به آن اضافه کنیم و آن را به عنوان یک مسئله ۳ بعدی حل کنیم و تلاش کنیم که یک صفحه برای جدا کردن دو کلاس پیدا کنیم.

(ب) دادن ورودی‌هایی با توان بالا به عنوان ورودی شبکه:

در صورتی می‌توان از این روش استفاده کرد که مقدار دقیق تمام نمونه‌ها را داشته باشیم. در این صورت می‌توان ورودی‌ها را به توان‌های بالا (۲ و ۳ و...) رساند و امیدوار باشیم فضای قرارگیری ویژگی‌های جدید در صفحه به صورت جداپذیر خطی باشد. «این کار در سوال دوم همین سری تمرین انجام شده است و با به توان ۲ رساندن ویژگی‌های ورودی شبکه، مسئله‌ای که جداپذیر خطی نبود به جداپذیر خطی تبدیل می‌شود.» در این مسئله به دلیل نداشتن موقعیت دقیق نمونه‌های هرکلاس نمی‌توان از این روش استفاده کرد.

(ج) اضافه کردن لایه مخفی:

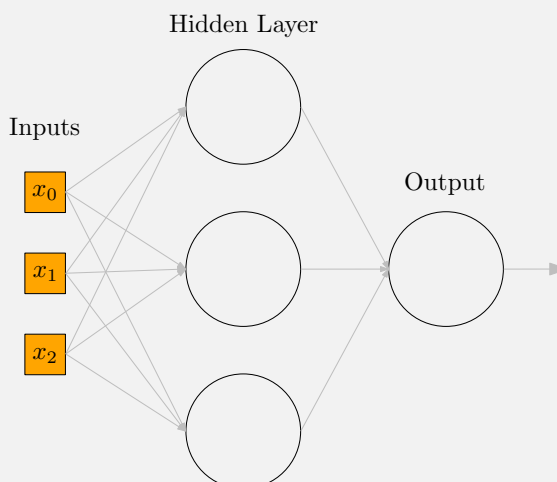
در اضافه کردن تعداد لایه‌های مخفی شبکه، آزاد هستیم اما باید به این نکته توجه داشت که اگر بیشتر از یک لایه مخفی به شبکه اضافه کنیم، حل مسئله از نظر خطی بودن خارج شده و نواحی تصمیم‌گیری شامل خط راست نمی‌شود و نواحی پیچیده‌تری مانند منحنی‌ها را در بر می‌گیرد. اما اگر فقط یک لایه مخفی ۳ نورونی به شبکه اضافه کنیم، می‌توان مسئله‌ای که ذاتاً جداپذیر خطی نیست را به وسیله ۳ خط جدا کننده که تعداد این خط‌ها برابر است با تعداد نورون‌های لایه مخفی، حل نمود. ساختار این مدل در «شکل ۳» آورده شده است.

اما مشکلی که وجود دارد آن است که در صورت سوال از ما خواسته شده وزن‌های شبکه را بدست آوریم، اگر از این ساختار برای حل استفاده کنیم، به دلیل نداشتن مقدار ورودی‌ها نمی‌توان ۳ وزن لایه متصل به خروجی را بدست آورد.

(د) شکستن مسئله به ۳ زیرمسئله خطی:

برای حل این سوال از این روش استفاده شده است. در قسمت قبل دیدیم که یک نورون پرسپترونی می‌تواند یک خط جدا کننده در صفحه رسم کند. در این مثال، برای جدا کردن این دو کلاس به ۳ خط نیاز داریم. پس داده‌های ورودی مسئله را به ۳ بخش جداپذیر خطی تقسیم می‌کنیم. «شکل ۴» اکنون می‌توان همانند قسمت قبل، وزن‌ها را برای هر سه زیرمسئله بدست آورد. در این قسمت ساختار و معماری شبکه همان ساختار قسمت a-1 است. «شکل ۲»

پاسخ



شکل ۳: معماری شبکه ۳ لایه پیشنهادی



شکل ۴: داده‌های شکسته شده به سه بخش

زیر مسئله شماره ۱ در قسمت قبل حل شده است و وزن‌های آن به صورت زیر بدست آورده شد:

$$w_0 = 2.8, \quad w_1 = -1, \quad w_2 = -1.12$$

برای دو زیر مسئله دیگر هم، همانند قسمت قبل وزن‌ها را بدست می‌آوریم.

$$P_1 = \begin{bmatrix} 2.5 \\ 0 \end{bmatrix} \quad P_2 = \begin{bmatrix} -1.5 \\ 1.7 \end{bmatrix} \rightarrow m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-1.5 - 2.5}{-1.7 - 0} = \frac{-4}{-1.7} = \boxed{2.35}$$

$$y - 0 = 2.35(x - 2.5) \rightarrow \boxed{y = 2.35x - 5.88}$$

$$\rightarrow x_1 = \frac{-w_2}{w_1}x_2 - \frac{w_0}{w_1} \rightarrow \begin{cases} \frac{-w_2}{w_1} = 2.35 \rightarrow w_2 = -2.35w_1 \\ \frac{-w_0}{w_1} = -5.88 \rightarrow w_0 = 5.88w_1 \end{cases}$$

Assume $w_0 = 5.88 \rightarrow w_1 = 1, w_2 = -2.35$

پاسخ

$$P_1 = \begin{bmatrix} -1.5 \\ -1.7 \end{bmatrix} P_2 = \begin{bmatrix} 0 \\ 2.8 \end{bmatrix} \rightarrow m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0 + 1.5}{2.8 + 1.7} = \frac{1.5}{4.5} = 0.33$$

$$y - 2.8 = 0.33(x - 0) \rightarrow y = 0.33x + 2.8$$

$$\rightarrow x_1 = \frac{-w_2}{w_1}x_2 - \frac{w_0}{w_1} \rightarrow \begin{cases} \frac{-w_2}{w_1} = 0.33 \rightarrow w_2 = -0.33w_1 \\ \frac{-w_0}{w_1} = 2.8 \rightarrow w_0 = -2.8w_1 \end{cases}$$

Assume $w_0 = 2.8 \rightarrow w_1 = -1, w_2 = 0.33$

۳. شگرد هسته^۱ چیست و چگونه می‌توان قسمت قبل را با آن حل نمود؟ توضیح دهید.

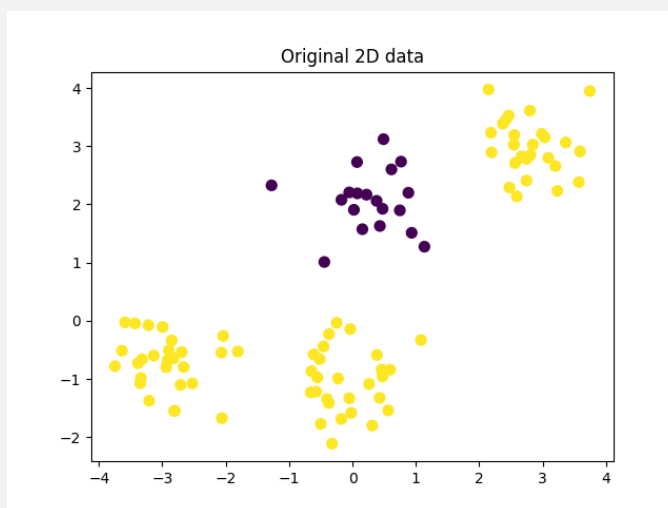
پاسخ

این تکنیک به عنوان یکی از روش‌های حل مسئله قبل معرفی شد و توضیح مختصری در مورد آن داده شد. در این قسمت توضیحات کامل آن را بیان خواهیم کرد.

روش شگرد هسته، یکی از چندین روش موجود برای حل مسائل جداناپذیر خطی با روش‌های خطی است. بدین صورت که اگر ابعاد ویژگی‌های مسئله را R در نظر بگیریم و داده‌ها در R بعد جداناپذیر خطی باشند، ممکن است با افزایش بعد ویژگی‌ها ($R+n$) و اضافه کردن ویژگی‌ای جدید، داده‌ها جداناپذیر خطی شوند و بتوان مسئله را با یک شبکه خطی حل نمود.

در این سوال، ویژگی‌های ورودی ۲ بعدی هستند و داده‌ها در ۲ بعد جداناپذیر خطی هستند. بنابراین می‌توان یک ویژگی جدید در بعد سوم به ورودی‌ها اضافه کرد و خروجی را نمایش داد تا شاید داده‌ها جداناپذیر خطی شوند.

این کار را انجام دادیم و اثبات کردیم که با افزایش بعد این مسئله، داده‌ها جداناپذیر خطی می‌شوند. شکل زیر را به عنوان داده‌های ورودی مسئله در نظر بگیریم:



شکل ۵: داده‌های ورودی مسئله در ۲ بعد

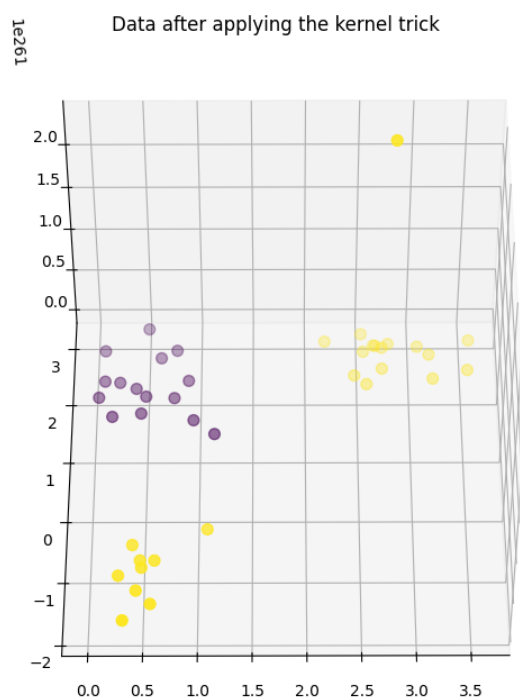
برای افزودن بعد سوم به این داده‌های ۲ بعدی، تابع زیر را نوشته‌ایم.

^۱Kernel trick

پاسخ

```
def kernel_trick(x):
    return np.append(x, np.expand_dims(x[:, 0]**2 ** x[:, 1]**2,
    axis=1), axis=1)
```

این تابع ویژگی جدید x_1^2 را به داده های ورودی به عنوان بعد سوم اضافه می‌کند. بنابر در فضای ویژگی جدید، هر نقطه از بردار ویژگی شامل ۳ عضو است. (این نکته لازم به ذکر است که ویژگی جدید تولید شده، با سعی و خطا بدست آمده است و می‌توان به جای آن هر ویژگی جدید دیگری را قرار داد) پس از رسم فضای ویژگی جدید، مشاهده می‌شود که داده ها جداپذیر خطی شده اند و می‌توان آنها را با یک صفحه از هم جدا نمود.



شکل ۶: فضای ویژگی های جدید در ۳ بعد

سوال دوم - عملی

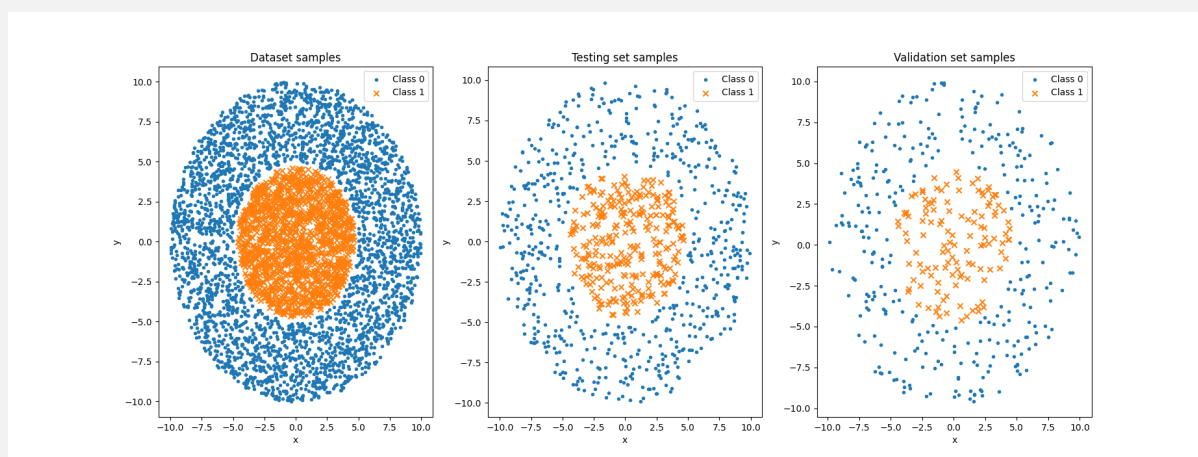
مجموعه داده ضمیمه شده را بارگزاری کرده و آن را نمایش دهید. تفکیک مجموعه داده را با نسبت ۱:۲:۷ به ترتیب برای آموزش، آزمون و اعتبارسنجی در نظر بگیرید. تمامی کدهای سوالات عملی پیوست شده است. همچنین می‌توانید کدها را از گیت‌هاب بنده به لینک زیر مشاهده و بررسی کنید:

github.com/rezaAdinepour/Deep-Learning-Homework

همچنین به دلیل آنکه تمامی کدها در فایل‌های notebook به طور کامل توضیح داده شده است، در این گزارش به دلیل جلوگیری از طولانی شدن آن، از آوردن کد پرهیز کرده ایم.

پاسخ

پس از load کردن دیتاست و تقسیم کردن آن به ۳ دسته آموزش، تست و اعتبار سنجی، آن را رسم کردیم. خروجی به صورت زیر است:



شکل ۷: دیتاست مسئله

همانطور که مشاهده می‌شود، داده‌ها جدایی‌پذیر خطی نیستند پس نمی‌توان به صورت معمولی آن را با شبکه پرسپترون حل نمود. در این سوال تلاش خواهیم کرد با روش‌های معرفی شده در سوال ۱، به صورت عملی با شبکه پرسپترون، یک مسئله غیر خطی را حل نماییم.

۱. با یک نورون پرسپترون و صرفاً بر اساس ویژگی‌های ورودی، وزن‌های نورون خود را با آموزش بدست آورید و دسته‌بندی را انجام دهید. و معیارهای صحت^۲ و امتیاز $F1$ را به ازای هر دسته گزارش نمایید. همچنین در نهایت وزن‌های معماری‌تان را به همراه طرحواره آن گزارش کنید.

Accuracy^۲

پاسخ

شبکه پرسپترون تک لایه با استفاده از کتابخانه PyTorch تعریف شده است. از تابع Sigmoid به عنوان تابع فعال‌ساز استفاده شده است. در این شبکه از تابع بهینه‌ساز stochastic gradient descent استفاده شده است و برای محاسبه خطا، از mean squared error به عنوان تابع هزینه استفاده شده است.

پس از ۵۰۰ اپیاک آموزش شبکه، خروجی شبکه به صورت زیر شده است:

Epoch 500/500, Loss: 0.3373, Accuracy: 0.5263, F1 Score: 0.5115

Validation Loss: 0.3900, Validation Accuracy: 0.5173, Validation F1 Score: 0.4689

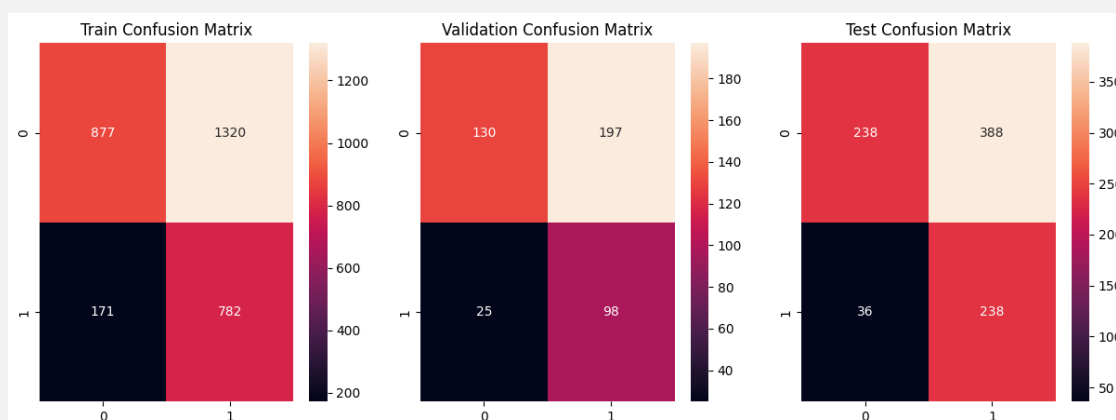
همچنین نمودارهای دقت و خطا بر حسب تعداد Epoch به صورت زیر شده است:



شکل ۸: نمودارهای خطا و دقت

همانطور که از نمودارها و خروجی شبکه مشاهده می‌شود، نشان از نوسان بالا در شبکه را دارد که کاملاً طبیعیست چرا که داده‌ها جدایپذیر خطی نیستند و شبکه هرچه تلاش می‌کند تا دو کلاس را از هم تفکیک کند نمی‌تواند و دچار نوسان می‌شود. مقدار Loss از 0.3 پایین تر نمی‌آید همچنین مشاهده می‌شود دقت روی مجموعه داده اعتبارسنجی بعد از اپیاک ۳۰۰ به شدت افت شده است.

ماتریس پراکندگی^a برای مجموعه داده‌های آموزش، تست و اعتبارسنجی به صورت زیر شده است:



شکل ۹: ماتریس پراکندگی

Confusion matrix^a

پاسخ

از خروجی ماتریس‌های پراکندگی نیز مشاهده می‌شود که تعداد نمونه‌های که اشتباه شناسایی شده‌اند (نمونه‌های روی قطر فرعی) زیاد هستند که نشان از عدم آموزش شبکه دارد.

پس از فاز آموزش شبکه، نوبت به تست شبکه آموزش داده شده به وسیله مجموعه داده‌های تست می‌رسد. خروجی شبکه بر روی داده‌های تست به صورت زیر شده است:

Test loss: 0.3493, Test Accuracy: 0.5289, Test F1 Score: 0.5289

وزن‌های نهایی شبکه پس از آموزش به صورت زیر به دست آمده است:

Final weights: tensor([[0.1273, -0.1831]], device='cuda:0')

۲. به ورودی قسمت قبل، توان بالای ورودی‌ها تا توان سوم را افزوده و نتیجه حاصل را ضمن گزارش تحلیل نموده و توجیه کنید.

پاسخ

در این قسمت، ابتدا توان دوم و سوم ورودی را محاسبه می‌کنیم و هر دو آن را به شبکه می‌دهیم. بر اساس آنچه که در سوال اول توضیح دادیم، ممکن است با به توان رساندن ویژگی‌های ورودی داده‌هایی که جدایند خطی نیستند، جدایند خطی شوند. در این مثال داده‌ها را پس از به توان رساندن رسم کردیم:



شکل ۱۰: ویژگی‌های به توان رسانده شده

همانطور که از «شکل ۱۰» مشاهده می‌شود، توان دوم داده‌های ورودی جدایند خطی شده است و امکان حل آن با یک نرون پرسپترون وجود دارد اما توان سوم داده‌های ورودی همچنان جدایند خطی است. توان چهارم نیز تست شد، آن هم جدایند خطی بود اما به دلیل آنکه در صورت سوال خواسته نشده است. گزارش آن آورده نشده است.

انتظار داریم اگر به ورودی شبکه، توان دوم داده‌های دیتاست را بدهیم، شبکه بتواند مسئله را حل نماید و مقدار Loss آن مینیمم و Accuracy آن ماکزیمم شود. این بار داده‌های جدید را با همان شبکه قبلی مجدد Train می‌کنیم و خروجی‌های آن را گزارش می‌دهیم.

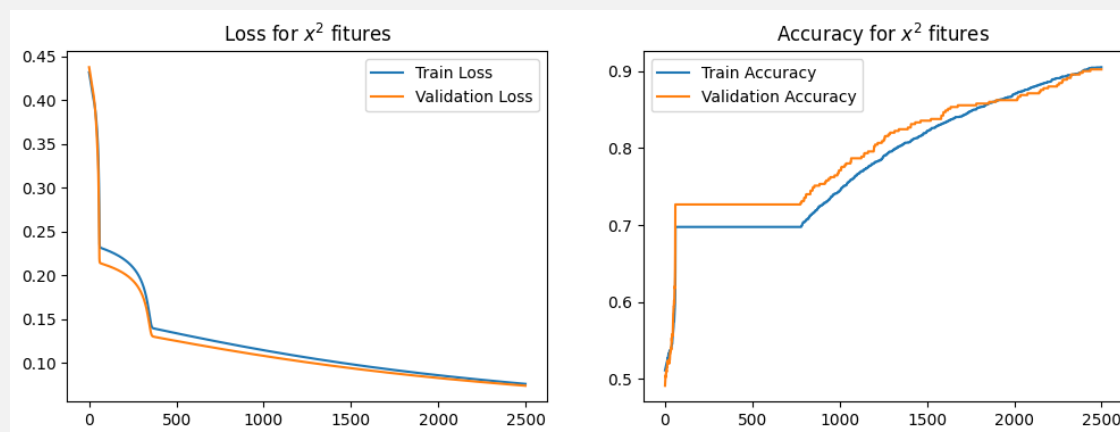
پس از آموزش شبکه با ویژگی‌های جدید، خروجی شبکه به صورت زیر به دست آمده است:

Epoch 500/500, Loss: 0.0760, Accuracy: 0.9051, F1 Score: 0.8139

پاسخ

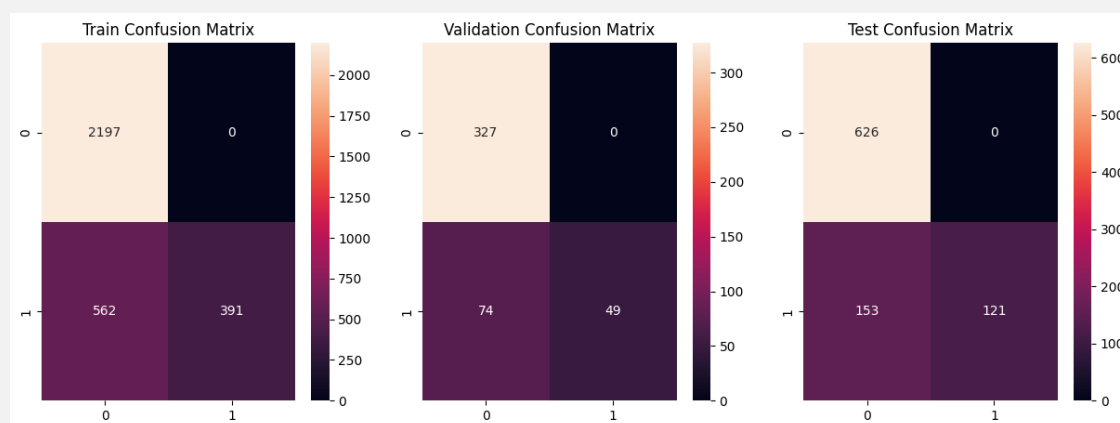
Validation Loss: 0.1162, Validation Accuracy: 0.7977, Validation F1 Score: 0.7822

مشاهده می‌شود که مقدار Loos گزارش شده بر روی داده‌های آموزش بسیار کوچکتر از حالت قبل است و تقریباً نزدیک به صفر. همچنین Accuracy شبکه نیز نسبت به حال قبل مقدار ۴۰ درصد افزایش یافته است. مقادیر گزارش شده بر روی داده‌های اعتبارسنجی نیز نسبت به حال قبل بهبود یافته‌اند. نمودار Loss و Accuracy خروجی شبکه به صورت زیر گزارش می‌شود:



شکل ۱۱: نمودارهای خطا و دقت

نوسانات کمتری نسبت به قسمت قبل در نمودار Accuracy مشاهده می‌شود و دیگر دقت برای مجموعه داده‌های اعتبارسنجی افت نکرده است. همچنین مقدار خطا نیز با شیب بیشتری نسبت به حالت قبل کاهش پیدا کرده است. خروجی ماتریس پراکندگی شبکه نیز به صورت زیر است:



شکل ۱۲: ماتریس پراکندگی

مشاهده می‌شود که نمونه‌های درست طبقه‌بندی شده (نمونه‌های روی قطر اصلی) نسبت به حال قبل بیشتر است و همچنین نمونه‌های اشتباه نیز به مراتب کمتر از حالت قبل است.

پاسخ

خروجی شبکه بر روی مجموعه داده‌های Test به صورت زیر شده است:

Test loss: 0.0965, Test Accuracy: 0.8300, Test F1 Score: 0.6127

خروجی شبکه بر روی داده‌های تست نیز بهبود زیادی نسبت به حالت قبل داشته است. همچنین وزن‌های نهایی شبکه به صورت زیر گزارش می‌شود:

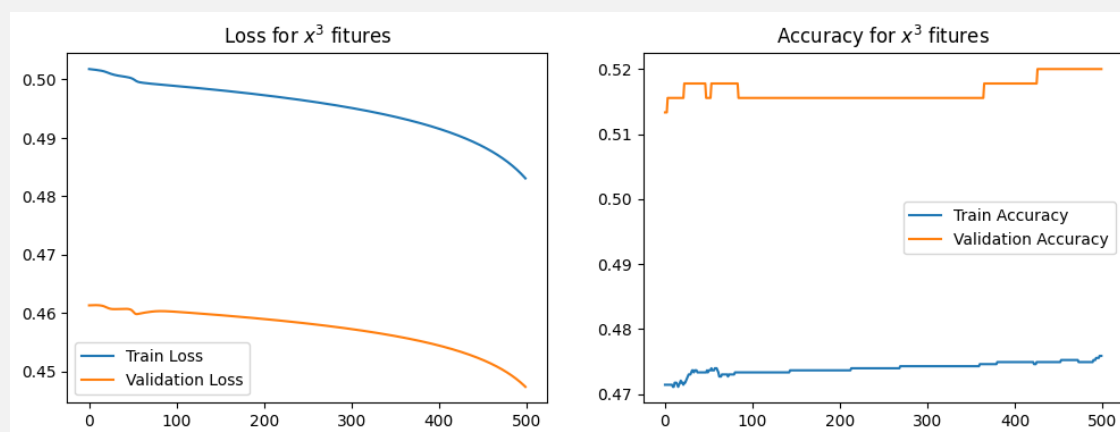
Final weights: tensor([[-0.0438, -0.0453]], device='cuda:0')

مجدداً تمام مراحل را برای توان سوم داده‌های ورودی تکرار می‌کنیم. مطابق با «شکل ۱۰» انتظار داریم خروجی شبکه پس از آموزش تقریباً همانند توان اول داده‌ها باشد، چرا که در هر دو حالت داده‌ها جداناپذیر خطی هستند. خروجی شبکه آموزش دیده بر روی توان سوم داده‌های ورودی به صورت زیر گزارش می‌شود:

Epoch 500/500, Loss: 0.4831, Accuracy: 0.4759, F1 Score: 0.3548

Validation Loss: 0.4572, Validation Accuracy: 0.5167, Validation F1 Score: 0.3793

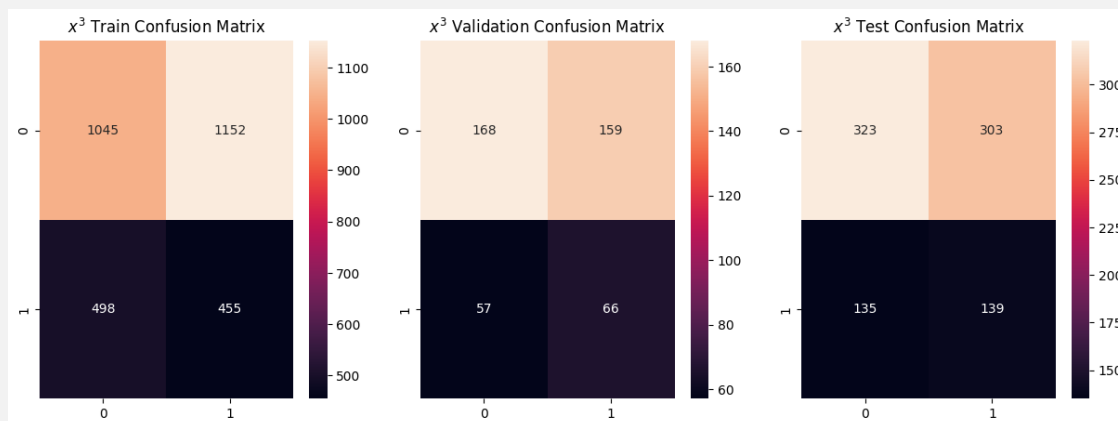
با توجه به نتایج بدست آمده، پیش‌بینی ما درست است. چرا که مقدار خطا نسبت به حالت دوم، مجدداً افزایش یافته و دقت نیز کاهش یافته است. نمودارهای خروجی نیز مطابق با شکل زیر گزارش می‌شود:



شکل ۱۳: نمودارهای خطا و دقت

از نمودارها نیز می‌توان نتیجه گرفت که شبکه به درستی آموزش ندیده است چرا که در ۵۰۰ تکرار مقدار خطا خیلی نرم و با شیب کمی کاهش یافته است همچنین مقدار دقت نیز در طول فرایند آموزش تقریباً ثابت بوده است. خروجی ماتریس‌های پراکندگی نیز به صورت زیر گزارش می‌شود:

پاسخ



شکل ۱۴: ماتریس پراکندگی

در این حالت نیز مجدداً نمونه‌های روی قطر غیر اصلی ماتریس افزایش یافته است که نشان از عدم طبقه‌بندی درست مسئله دارد.

خروجی شبکه بر روی مجموعه داده‌گان تست به صورت زیر گزارش می‌شود:

Test loss: 0.4470, Test Accuracy: 0.5133, Test F1 Score: 0.3883

شبکه بر روی داده‌های تست نیز خروجی مطلوبی ندارد و مقدار Loss و Accuracy آن نسبت به حالت قبل افت کرده است.

وزن‌های نهایی شبکه به صورت زیر گزارش می‌شود:

Final weights: tensor([[0.0587, 0.1524]], device='cuda:0')

۳. حال حاصل ضرب ورودی‌های قسمت قبل برای مثال $(x_1x_2, x_1x_2^2, x_1^3x_2)$ را به ورودی پرسپترون افزوده و مجدداً نتیجه حاصل را ضمن گزارش تحلیل نموده و توجیه کنید. به جهت ریاضیاتی و هندسی، این جمله‌ها بیانگر افزودن چه ویژگی‌هایی هستند؟

پاسخ

برای بدست ویژگی‌های جدید این قسمت، از کلاس PolynomialFeatures در کتابخانه sklearn استفاده کرده ایم. این کلاس ویژگی‌های ورودی را می‌گیرد و بر اساس درجه‌ای که برای آن مشخص می‌کنیم، تمام ترکیبات چند جمله‌ای تا آن درجه را تولید می‌کند. برای مثال اگر ماتریس ویژگی‌های ما به صورت زیر باشد:

$$X = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}$$

با فراخوانی کلاس PolynomialFeatures و فیت کردن داده‌ها با تابع fit_transform به صورت زیر برای ساختن ویژگی‌های چند جمله‌ای حداکثر تا توان ۲:

```
poly = PolynomialFeatures(degree=2, include_bias=True)
X_poly = poly.fit_transform(X)
```

خروجی (ماتریس ویژگی‌های جدید) به صورت زیر خواهد بود:

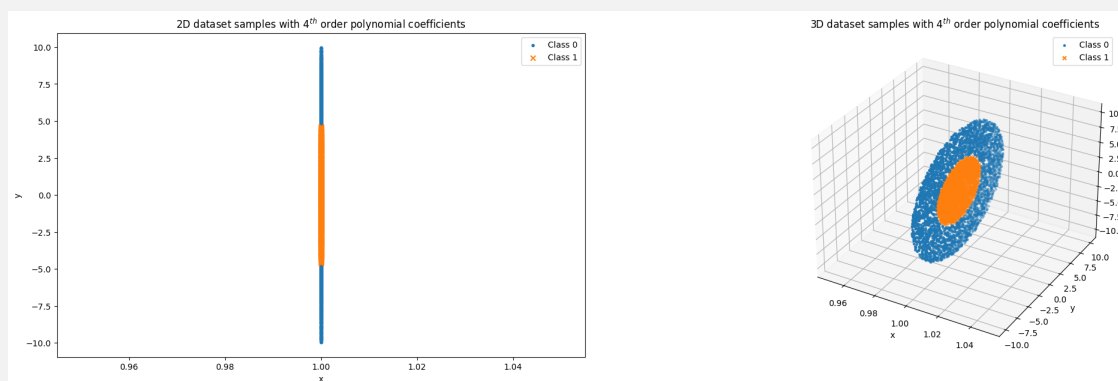
پاسخ

$$X_{new} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 2 & 3 & 4 & 6 & 9 \\ 1 & 4 & 5 & 16 & 20 & 25 \end{bmatrix}$$

در ویژگی جدید، علاوه بر دو مقدار قبل، در درایه اول هر سطر، بایاس با مقدار ۱ اضاف می‌شود، همچنین در درایه های ۳ تا ۶ در هر سطر، به ترتیب ویژگی های a^2 , ab , b^2 حساب شده است. تعریف رسمی این کلاس در داکيومنت sklearn به صورت زیر است:

Generate a new feature matrix consisting of all polynomial combinations of the feaxtures with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are $[1, a, b, a^2, ab, b^2]$.

در این مثال، degree را برابر با ۴ قرار دادیم. یعنی چندجمله‌ای هایی حداکثر تا درجه ۴ را به عنوان ویژگی تولید می‌کنیم. ابعاد دیتاست ورودی مسئله (4500, 3) است. پس از تولید ویژگی های جدید ابعاد مسئله به (4500, 15) افزایش می‌یابد. بنابر این تعداد نوروں های ورودی مسئله نیز از ۳ عدد به ۱۵ افزایش پیدا می‌کند و وزن های نهایی باید شامل ۱۵ عدد وزن باشد. فضای جدید ویژگی ها را به صورت ۲ بعدی و ۳ بعدی رسم کرده‌ایم و خروجی آن به صورت زیر شده است:



شکل ۱۵: فضای جدید ویژگی‌ها

همانطور که مشاهده می‌شود، ویژگی های جدید بدست آمده جداناپذیر خطی هستند و انتظار داریم با آموزش شبکه بر روی این ویژگی ها، خروجی ای همانند حالت توان سوم ویژگی‌ها حاصل شود و شبکه به درستی آموزش نبیند. به این دلیل داده ها در توان‌های بالا جداناپذیر خطی می‌شوند که مقدار اغلب داده‌های خام کوچکتر از ۱ هستند. به همین دلیل توان‌های بالای آنها مقدار بسیار کوچکی می‌شود و باعث فشرده شدن و روی هم افتادن داده‌ها می‌شود. اگر بخواهیم از داده‌های خطی برای حل مسائل غیرخطی استفاده کنیم، طبیعتاً نباید انتظار داشت خروجی مطلوبی حاصل شود. ورودی‌های توان بالا و چندجمله‌ای ها این امکان را برای شبکه فراهم می‌کنند تا تلاش کند با داده‌های غیر خطی (توان‌های دوم و سوم ورودی و حاصل ضرب آنها) آموزش ببیند تا شاید بتواند مسئله غیرخطی‌مان را حل نماید.

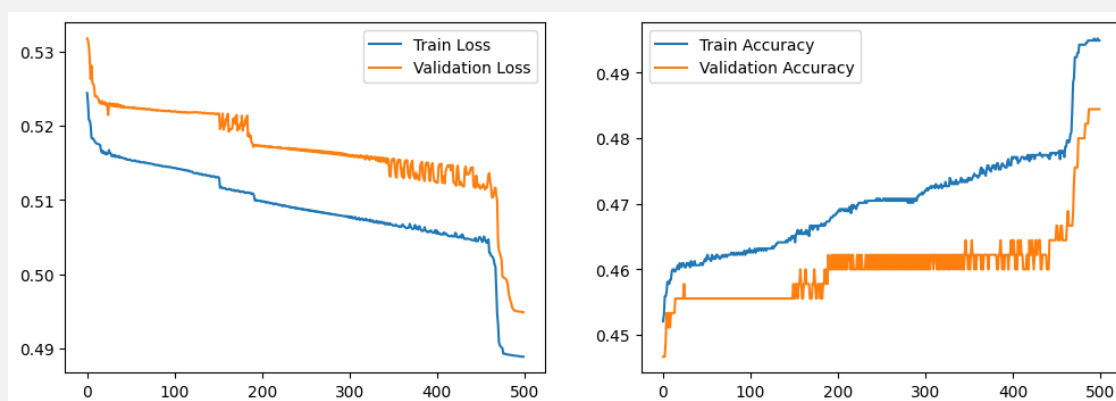
برای آنکه مطمئن شویم، شبکه را با داده‌های جدید Train می‌کنیم و خروجی را گزارش می‌کنیم. خروجی شبکه پس از آموزش با داده‌های جدید به صورت زیر است:

پاسخ

Epoch 500/500, Loss: 0.4888, Accuracy: 0.4949, F1 Score: 0.3850

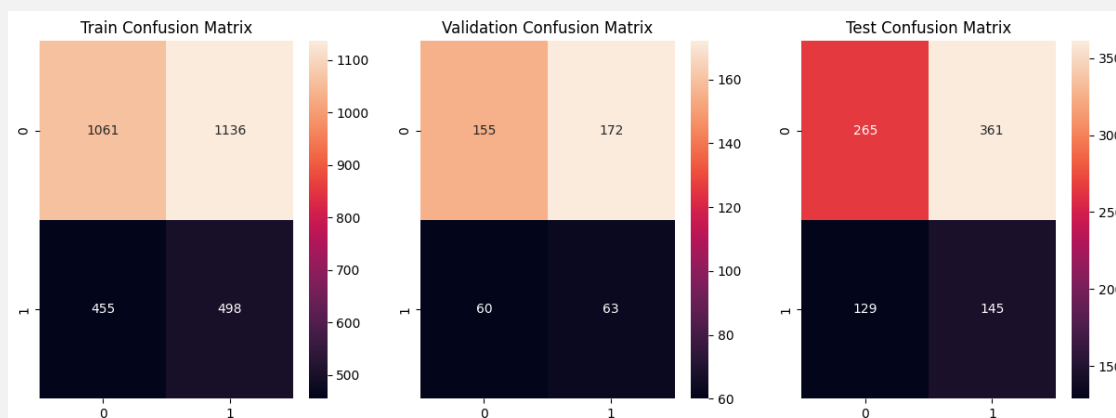
Validation Loss: 0.5167, Validation Accuracy: 0.4607, Validation F1 Score: 0.3520

از نتایج مشخص است که شبکه نتوانسته به درستی آموزش ببیند و مقدار Loss و Accuracy بدست آمده تقریباً همانند قسمت اول (آموزش با داده‌های خام) است. نمودارهای Loss و Accuracy نیز این حرف را توجیه می‌کنند/ چرا که بسیار نوسانی بوده و مقادیر آنها از یک حد کمتر یا بیشتر نشده است و این امر نشان از عدم آموزش صحیح شبکه دارد. خروجی نمودارهای به صورت زیر گزارش می‌شود:



شکل ۱۶: نمودارهای خطا و دقت

ماتریس پراکندگی داده‌ها به صورت زیر گزارش می‌شود:



شکل ۱۷: ماتریس پراکندگی

همچنین تست شبکه بر روی مجموعه داده‌های تست و وزن‌های نهایی شبکه نیز به صورت زیر گزارش می‌شود:
 Test loss: 0.5271, Test Accuracy: 0.4556, Test F1 Score: 0.3718
 Final weights: tensor([[-0.0076, 0.1363, 0.1950, -0.0310, 0.1818, -0.0550, -0.1191, 0.0098, -0.0858, 0.2467, 0.2740, 0.1114, -0.0028, 0.1657, -0.1999]], device='cuda:0')

سوال سوم - عملی

فرض کنید می‌خواهیم تابع $f(x) = \sin(x) + 3x^{17} - 5x^2$ را با یک نورون پرسپترونی تخمین و در صورت امکان محاسبه نماییم.

۱. با تحقیق و مطالعه کافی توضیح دهید چگونه می‌توان از سری تیلور برای تخمین توابع استفاده نمود؟

پاسخ

سری تیلور بیان می‌کند که ما می‌توانیم هر تابع n بار مشتق‌پذیری را بر اساس یک مجموع بی‌نهایت از مشتق‌های آن حول نقطه صفر بنویسیم. فرمول بسط تیلور به صورت زیر بیان می‌شود:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} (x)^n = f(0) + \frac{f'(0)}{1!} (x) + \frac{f''(0)}{2!} (x)^2 + \dots + \frac{f^{(n)}(0)}{n!} (x)^n$$

آقای مک‌لورن این تقریب را به صورت عمومی گسترش داد و بیان کرد که می‌توان توابع مختلف n بار مشتق‌پذیر را حول هر نقطه مشخصی بسط داد و فرمول تیلور را به صورت زیر بازنویسی کرد:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n = f(a) + \frac{f'(a)}{1!} (x-a) + \frac{f''(a)}{2!} (x-a)^2 + \dots + \frac{f^{(n)}(a)}{n!} (x-a)^n$$

بنابر این هر تابع n بار مشتق‌پذیری را با داشتن مشتق‌های آن و مقدار آن تابع در نقطه مورد نظر، تقریب زد. برای مثال، تابع e^x ، n بار مشتق‌پذیر است پس می‌توان نوشت:

$$\begin{aligned} f(x) &= e^x \rightarrow f'(x) = f''(x) = f'''(x) = f^{(4)}(x) = \dots = f^{(n)}(x) = e^x \\ \text{if } a &= 0 \rightarrow e^x \approx e^0 + \frac{e^0}{1!} (x) + \frac{e^0}{2!} (x)^2 + \frac{e^0}{3!} (x)^3 + \dots + \frac{e^0}{n!} (x)^n \\ &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots + \frac{1}{n!}x^n = \sum_{n=0}^{\infty} \frac{1}{n!}x^n \end{aligned}$$

طبیعتاً هرچقدر تعداد جملات بیشتری را در مجموع باهم جمع کنیم، تقریب دقیق تری از تابع حاصل می‌شود اما بار محاسباتی سنگین تر می‌شود و نیازمند محاسبه مشتق‌هایی با مرتبه‌های بالا هستیم.

۲. با استفاده از سری تیلور، تخمین تابع یاد شده را تا ۱۰ جمله محاسبه و بدست آورید. فرایند محاسبه را در گزارش بنویسید.

پاسخ

تابع داده شده از ۲ ترم تشکیل شده است. ترمی سینوسی و ترم چندجمله‌ای. به دلیل آنکه از چندجمله‌ای تیلور استفاده می‌کنیم برای تقریب و $a = 0$ است، مقادیر این ترم و مشتقات آن در نقطه صفر، صفر خواهد بود. بنابر این این دو ترم چند جمله‌ای را که در پایین با آبی مشخص شده است را با تقریب تیلور محاسبه نمی‌کنیم و فقط ترم سینوسی را با استفاده از سری تیلور حساب می‌کنیم. مشتقات تابع سینوسی (تا مرتبه ۹) را محاسبه می‌کنیم و مقادیر آنها در نقطه صفر محاسبه کرده و آن‌ها را در فرمول سری تیلور قرار می‌دهیم. در نهایت ترم چند جمله‌ای را عیناً به تابع تقریب زده شده اضافه می‌کنیم. محاسبات را به صورت زیر انجام داده ایم:

پاسخ

$$\begin{aligned}
 f(x) &= \sin(x) + 3x^{17} - 5x^2 \\
 f_1(x) &= \sin(x) : a = 0 \rightarrow \sum_0^9 \frac{f_1^{(n)}(0)}{n!} x^n = f_1(0) + \frac{f_1'(0)}{1!} x^1 + \dots + \frac{f_1^{(9)}(0)}{9!} x^9 \\
 f_1(x) &= \sin(x) \rightarrow f_1(0) = 0 \\
 f_1'(x) &= \cos(x) \rightarrow f_1'(0) = 1 \\
 f_1''(x) &= -\sin(x) \rightarrow f_1''(0) = 0 \\
 f_1'''(x) &= -\cos(x) \rightarrow f_1'''(0) = -1 \\
 f_1^{(4)}(x) &= \sin(x) \rightarrow f_1^{(4)}(0) = 0 \\
 &\vdots \\
 &\vdots \\
 &\vdots \\
 f_1^{(9)}(x) &= \cos(x) \rightarrow f_1^{(9)}(0) = 1 \\
 \rightarrow &= 0 + \frac{1}{1!} x^1 + \frac{0}{2!} x^2 + \frac{-1}{3!} x^3 + \frac{0}{4!} x^4 + \frac{1}{5!} x^5 + \frac{0}{6!} x^6 + \frac{-1}{7!} x^7 + \frac{0}{8!} x^8 + \frac{1}{9!} x^9 \\
 &= x - \frac{1}{6} x^3 + \frac{1}{120} x^5 - \frac{1}{5040} x^7 + \frac{1}{362880} x^9 = \hat{f}(x)
 \end{aligned}$$

۳. حال یک نرون پرسپترونی طراحی نمایید که بتواند تابع فوق را با جملات سری تیلور محاسبه کند (وزن‌های نرون را با محاسبات بدست‌آورده و در گزارش خود بیان کنید). در یک نمودار، تابع و تخمین‌های آن را به ازای استفاده از جملات ۱ تا ۱۰ رسم کنید (خروجی ۱۱ منهنی خواهد بود). در یک جدول خطای حاصل از تقریب را به ازای استفاده از جملات مختلف با تابع MSE گزارش کنید.

توجه مهم: ورودی نرون‌های طراحی شده‌تان صرفاً بایستی توانی از ویژگی‌های اصلی یا حاصل ضرب توانی از ویژگی‌ها باشد و فرم دیگری قابل قبول نیست. برای مثال اگر یک ویژگی x باشد، $\sin(x)$ یا e^x نمی‌تواند ورودی یک نرون باشد.

پاسخ