

### تمرین شماره ۳

علیرضا حسینی

شماره دانشجویی : ۸۱۰۱۰۱۱۴۲

یادگیری ماشین

دکتر ابوالقاسمی و دکتر توسلی پور

۱۴۰۲ بهار

## فهرست مطالب

7	1-سوال 1 : مفاهیم اساسی شبکه عصبی
7	1-1-1- توابع هزینه و توابع فعالساز
11	1-1-2- تاثیر مقدار دهی اولیه وزن ها و بایاس ها در یادگیری
14	1-1-3- تفاوت Forward و Backward propagation
16	1-1-4- تاثیر کوچک یا بزرگ بودن مقدار learning rate
19	1-1-5- تاثیر تعداد لایه های شبکه عصبی
21	1-1-6- بزرگترین مشکل شبکه عصبی در مرحله آموزش
24	1-سوال 2 : پیشنهاد معماری MLP
24	1-2-1- الف
25	1-2-2- ب
25	1-2-3- ج
25	1-سوال 3 : کانولوشن
27	1-4- سوال 4 : سوالاتی در مورد توابع فعالساز
27	1-4-1- بار محاسباتی کمتر تابع relu
28	1-4-2- مشکل vanishing gradient را ندارد
30	1-4-3- استفاده از sigmoid در لایه آخر برای مساله طبقه بندی
31	1-4-4- فرق بین Relu و Leaku Relu
33	1-سوال 5 : محاسبات دستی درخت تصمیم
36	1-سوال 6 : طبقه بندی داده ای ECG
36	1-6-1- لود داده ها و بررسی توزیع داده ها
39	1-6-2- آموزش یک شبکه MLP با چهار لایه مخفی بر روی دیتای خام
45	1-6-3- نرمال سازی داده ها
47	1-6-4- تبدیل مساله به طبقه بند باینری abnormal و normal
49	1-6-5- نتیجه گیری
51	1-سوال 7 : پیاده سازی Madeline
51	1-7-1- تولید دیتا ها
53	1-7-2- پیاده سازی کلاس Madeline
61	1-7-3- نتایج Madeline به ازای لایه های مخفی و نورون های لایه اخر مختلف
66	1-سوال 8 : طبقه بندی تصاویر 10 Cifar
66	1-8-1- لود کردن دیتاست CIFAR و پیش پردازش
68	1-8-2- آموزش شبکه به optimizer های مختلف
71	1-سوال 9 : Decision Tree بر روی داده بیماران دیابتی
71	1-9-1- لود کردن دیتاست و پیش پردازش
72	1-9-2- طراحی decision Tree از scratch
79	1-9-3- درخت تصمیم با استفاده از کتابخانه آماده
80	1-9-4- درخت تصمیم با حداقل طول 2
81	1-9-5- مقایسه تفسیر پذیری درخت تصمیم با طول 2 با حداقل طول های بالاتر



## فهرست اشکال

24	..... شکل (۱-۱) معماری پیشنهادی سوال ۲ - الف
26	..... شکل (۲-۱) محاسبات کانولوشن مربوط به سوال ۳
33	..... شکل (۳-۱) محاسبات بخش الف سوال ۵
34	..... شکل (۴-۱) محاسبات بخش ب سوال ۵
35	..... شکل (۵-۱) جواب بخش ج سوال ۵
36	..... شکل (۶-۱) کد پایتون لود فایل CSV داده ها
36	..... شکل (۷-۱) کد پایتون بررسی تعداد داده های هر کلاس
37	..... شکل (۸-۱) کد پایتون و منحنی توزیع داده ها بر حسب هر کلاس
39	..... شکل (۹-۱) دیتافریم اولیه و دلیل حذف ستون اول
39	..... شکل (۱۰-۱) کد پایتون مپ کردن لیل ها و حذف ستون اول
40	..... شکل (۱۱-۱) خروجی دیتافریم پس از تغییرات اعمال شده
40	..... شکل (۱۲-۱) کد پایتون تقسیم داده های فیچر و لیل ECG
40	..... شکل (۱۳-۱) کد پایتون تبدیل داده های $X$ به one-hot numpy array و کردن لیل ها
41	..... شکل (۱۴-۱) کد پایتون پیاده سازی مدل و کامپایل و آموزش مدل
41	..... شکل (۱۵-۱) منحنی loss مدل تحت آموزش در حالت بدون هیچ پردازشی
42	..... شکل (۱۶-۱) منحنی دقت شبکه تحت آموزش بر روی داده های val در حالت بدون هیچ پیش پردازشی
43	..... شکل (۱۷-۱) کد پایتون گرفتن خروجی بر روی داده های تست و بررسی f1score و precision و recall
43	..... شکل (۱۸-۱) مقادیر Precison و recall و F1 و ماتریس آشتفتگی در حالت بدون هیچ پیش پردازشی
44	..... شکل (۱۹-۱) کد پایتون بررسی میزان دقت در هر کلاس
44	..... شکل (۲۰-۱) خروجی کد بررسی دقت روی هر کلاس
45	..... شکل (۲۱-۱) کد پایتون نرمال سازی داده ها و تقسیم داده ها
46	..... شکل (۲۲-۱) منحنی loss - در حالت نرمال سازی داده ها
46	..... شکل (۲۳-۱) منحنی دقت شبکه روی داده های val - در حالت نرمال سازی داده ها
47	..... شکل (۲۴-۱) مقادیر recall و f1 و precision و f1 و ماتریس آشتفتگی روی داده های تست - در حالت نرمال سازی داده ها
47	..... شکل (۲۵-۱) میزان دقت شبکه بر روی داده های تست بر حسب هر کلاس
47	..... شکل (۲۶-۱) حذف کلاس چهارم و تجمعی کلاس A و O
48	..... شکل (۲۷-۱) منحنی loss در حالت طبقه باینری abnormal و normal و abnormal و normal
48	..... شکل (۲۸-۱) منحنی دقت val در طبقه بند
49	..... شکل (۲۹-۱) مقادیر recall و precision و F1 و ماتریس آشتفتگی روی داده های تست در حالت طبقه بند باینری
51	..... شکل (۳۰-۱) کد پایتون تولید داده های سوال ۷ و پلات آن
52	..... شکل (۳۱-۱) داده های تولید شده برای سوال ۷
52	..... شکل (۳۲-۱) کانکت داده های سوال ۷ و نمونه ای data ساخته شده

54	..... شکل (۱-۳۳) بخش 1 کد Madeline
55	..... شکل (۱-۳۴) بخش 2 کد Madeline
55	..... شکل (۱-۳۵) بخش 3 کد Madeline
55	..... شکل (۱-۳۶) بخش 4 کد Madeline
56	..... شکل (۱-۳۷) بخش 5 کد Madeline
56	..... شکل (۱-۳۸) بخش 6 کد Madeline
57	..... شکل (۱-۳۹) بخش 7 کد Madeline
58	..... شکل (۱-۴۰) بخش 8 کد Madeline
59	..... شکل (۱-۴۱) بخش 9 کد Madeline
60	..... شکل (۱-۴۲) بخش 10 کد Madeline
61	..... شکل (۱-۴۳) اعمال Madeline بر روی داده ها با تعداد نورون های متفاوت
62	..... شکل (۱-۴۴) منحنی loss در madeline برای جداسازی با 3 خط
62	..... شکل (۱-۴۵) مقادیر precision و recall و F1Score برای Madeline و جداسازی با 3 خط
62	..... شکل (۱-۴۶) مرز های تصمیم در Madeline در حالت جداسازی با 3 خط
63	..... شکل (۱-۴۷) منحنی loss در Madeline در حالت جداسازی با 4 خط و 6 لایه پنهان
63	..... شکل (۱-۴۸) مقادیر precision و recall و F1Score برای madeline و جداسازی با 4 خط و 6 لایه پنهان
64	..... شکل (۱-۴۹) مرز تصمیم در حالت جداسازی با 4 خط و 6 لایه پنهان
64	..... شکل (۱-۵۰) منحنی loss در حالت جداسازی با 6 خط و 9 لایه پنهان
65	..... شکل (۱-۵۱) مقادیر precision و recall در حالت جداسازی با 6 خط و 9 لایه پنهان
65	..... شکل (۱-۵۲) جداسازی با 6 خط و 9 لایه پنهان (Madeline)
66	..... شکل (۱-۵۳) کد پایتون لود دیتاست
66	..... شکل (۱-۵۴) کد پایتون نمایش رندوم 5 نمونه از داده های CIFAE10
67	..... شکل (۱-۵۵) 5 نمونه داده از داده های CIFAR10
67	..... شکل (۱-۵۶) کد پایتون پیش پردازش داده های CIFAR و تقسیم داده ها
67	..... شکل (۱-۵۷) داده های تقسیم شده Shape
68	..... شکل (۱-۵۸) Optimizer های مطرح شده در صورت مساله
68	..... شکل (۱-۵۹) پیاده سازی شبکه داده شده با فریم وورک تنسورفلو
69	..... شکل (۱-۶۰) کد پایتون آموزش شبکه به ازای optimizer های متفاوت
70	..... شکل (۱-۶۱) منحنی Loss و دقت برای adam Opt
70	..... شکل (۱-۶۲) منحنی loss و دقت برای SGD Opt.
70	..... شکل (۱-۶۳) منحنی loss و دقت بر روی RMSprop opt
71	..... شکل (۱-۶۴) مقادیر precision و recall و F1score و Recall به ازای optimizer های مختلف
72	..... شکل (۱-۶۵) لود کردن دیتاست دیابت
72	..... شکل (۱-۶۶) کد پایتون پیش پردازش داده های دیابت و تبدیل به فضای ورودی و خروجی

73	..... شکل (۱-۱) بخش 1 کد decision Tree
73	..... شکل (۱-۲) بخش 2 کد decision Tree
74	..... شکل (۱-۳) بخش 3 کد decision Tree
74	..... شکل (۱-۴) بخش 4 کد decision Tree
75	..... شکل (۱-۵) بخش 5 کد decision Tree
75	..... شکل (۱-۶) بخش 6 کد decision Tree
76	..... شکل (۱-۷) بخش 7 کد decision Tree
76	..... شکل (۱-۸) بخش 8 کد decision Tree
77	..... شکل (۱-۹) کد پایتون اعمال داده ها به scratch نوشته شده و مقادیر دقت
78	..... شکل (۱-۱۰) تابع رسم درخت تصمیم از scratch
78	..... شکل (۱-۱۱) درخت تصمیم رسم شده از scratch
79	..... شکل (۱-۱۲) کد پایتون اعمال درخت تصمیم بر داده ها و رسم آن و مقادیر دقت ها
79	..... شکل (۱-۱۳) درخت تصمیم با حداکثر عمق ۵
80	..... شکل (۱-۱۴) درخت تصمیم با حداکثر طول ۲ - روی کلاس آماده و کلاس نوشته شده از scratch
81	..... شکل (۱-۱۵) درخت تصمیم با حد کثر طول برابر ۲

## 1-1- سوال 1: مفاهیم اساسی شبکه عصبی

### 1-1-1- توابع هزینه و توابع فعالساز

شبکه های عصبی مدل های قدرتمند یادگیری ماشینی هستند که به طور گسترده در حوزه های مختلف مانند بینایی کامپیوتر، پردازش زبان طبیعی و تجزیه و تحلیل داده ها استفاده می شوند. دو جزء حیاتی شبکه های عصبی، توابع هزینه و توابع فعالسازی هستند که نقش مهمی در آموزش و بهینه سازی شبکه دارند. این ادامه به بررسی اهمیت توابع هزینه و توابع فعالسازی، انواع آنها و تأثیر آنها بر عملکرد شبکه های عصبی می پردازد.

شبکه های عصبی شامل گره ها یا نورون های به هم پیوسته ای هستند که ساختار و عملکرد مغز انسان را تقلید می کنند. این شبکه ها از داده های می گیرند و بر اساس الگوهای روابطی که کشف می کنند، پیش بینی یا تصمیم می گیرند. توابع هزینه و توابع فعال سازی عناصر ضروری هستند که شبکه های عصبی را قادر می سازند تا یادگیرند و با وظایف پیچیده سازگار شوند.

#### توابع هزینه:

توابع هزینه که به عنوان توابع ضرر یا توابع هدف نیز شناخته می شوند، اختلاف بین خروجی های پیش بینی شده و خروجی های واقعی را کمیت می دهند. هدف اصلی تابع هزینه، ارائه معیاری از عملکرد شبکه است. در طول مرحله آموزش، شبکه های عصبی پارامترهای خود را برای به حداقل رساندن تابع هزینه تنظیم می کنند که منجر به بهبود عملکرد می شود.

#### انواع توابع هزینه:

انواع مختلفی از توابع هزینه وجود دارد، و انتخاب بستگی به ماهیت مشکل حل شده دارد. برخی از توابع هزینه رایج عبارتند از:

- میانگین مربعات خطأ (MSE): MSE میانگین اختلاف مجدد بین مقادیر پیش بینی شده و واقعی را محاسبه می کند. معمولاً در مشکلات رگرسیون استفاده می شود.
- آنتروپی متقاطع باینری (binary cross entropy) : آنتروپی متقاطع باینری زمانی که با وظایف طبقه بندی باینری سروکار داریم استفاده می شود. عدم تشابه بین احتمالات پیش بینی شده و برچسب های باینری واقعی را اندازه گیری می کند.
- آنتروپی متقاطع طبقه ای: آنتروپی متقاطع طبقه ای برای مسائل طبقه بندی چند طبقه استفاده می شود. تفاوت بین احتمالات کلاس های پیش بینی شده و برچسب های کلاس واقعی را کمیت می کند.
- سایر توابع هزینه: توابع هزینه تخصصی برای وظایف خاص وجود دارد، مانند تابع ضرر هوبر برای رگرسیون قوی و واگرایی Kullback-Leibler برای اندازه گیری عدم تشابه بین توزیع های احتمال.

### توابع فعال سازی:

توابع فعال سازی غیرخطی بودن شبکه های عصبی را معرفی می کنند و آنها را قادر می سازد تا روابط پیچیده بین ورودی ها و خروجی ها را مدل سازی کنند. توابع فعال سازی بر روی خروجی های نورون های منفرد اعمال می شود و تعیین می کند که آیا آنها باید فعال شوند یا نه و بر جریان اطلاعات از طریق شبکه تأثیر می گذارد.

### توابع فعال سازی رایج:

- چندین توابع فعال سازی معمولاً در شبکه های عصبی مورد استفاده قرار می گیرند، از جمله:
- سیگموئید: تابع سیگموئید ورودی ها را در محدوده ای بین 0 و 1 ترسیم می کند که نشان دهنده احتمال فعال شدن نورون است. به طور گسترده ای در لایه های پنهان شبکه ها استفاده می شود.
- واحد خطی اصلاح شده (ReLU): ReLU تمام ورودی های منفی را صفر می کند و ورودی های مثبت را بدون تغییر ارسال می کند. این به حل مشکل گرادیان ناپدید شدن کمک می کند و به طور گسترده در شبکه های عصبی عمیق استفاده می شود.

- تائزانت هایپربولیک (Tanh): تابع  $\text{Tanh}$  ورودی ها را به محدوده ای بین  $-1$  و  $1$  نگاشت می کند. این تابع مشابه تابع سیگموئید است اما در حدود صفر متقارن است.
- سایر توابع فعال سازی: توابع فعال سازی دیگری مانند softmax (مورد استفاده در مسائل طبقه‌بندی چند کلاسه)، Leaky ReLU (به مشکل در حال مرگ ReLU می‌پردازد) و ELU (واحد خطی نمایی) وجود دارد.

### تأثیر بر عملکرد شبکه عصبی:

انتخاب تابع هزینه و تابع فعال سازی می تواند به طور قابل توجهی بر عملکرد شبکه های عصبی تأثیر بگذارد.

#### تأثیر تابع هزینه:

یک تابع هزینه مناسب تضمین می کند که شبکه به راه حل مورد نظر همگرا می شود. انتخاب تابع هزینه اشتباه ممکن است منجر به نتایج غیربهینه یا مسائل همگرایی شود. به عنوان مثال، استفاده از MSE برای وظایف طبقه‌بندی می تواند به دلیل عدم تطابق بین الزامات وظیفه و ویژگی های تابع هزینه، عملکرد ضعیفی داشته باشد.

#### تأثیر عملکرد فعال سازی:

انتخاب تابع فعال سازی شبکه را از نظر ظرفیت یادگیری، سرعت همگرایی و توانایی مدل سازی روابط پیچیده تحت تأثیر قرار می دهد.

- ظرفیت یادگیری: توابع فعال سازی مختلف ظرفیت یادگیری متفاوتی دارند. توابع فعال سازی غیرخطی مانند سیگموئید، ReLU و  $\text{tanh}$ ، شبکه های عصبی را قادر می سازد تا روابط پیچیده و غیرخطی را در داده ها یاد بگیرند. آنها غیرخطی بودن را به شبکه وارد می کنند و به آن اجازه می دهند هر تابع دلخواه را تقریبی کنند. از سوی دیگر، توابع فعال سازی خطی، ظرفیت یادگیری شبکه را محدود می کنند، زیرا آنها فقط می توانند روابط خطی را مدل کنند.

• سرعت همگرایی: انتخاب تابع فعالسازی می‌تواند بر سرعت همگرایی شبکه عصبی در طول

آموزش تأثیر بگذارد. توابع فعالسازی که مشتقات کاملاً مشخصی دارند و از اشباع یا مشکلات

گرادیان محو می‌شوند، هم‌گرایی سریع‌تر را تسهیل می‌کنند. ReLU و انواع آن، مانند Leaky

ReLU و ELU، به دلیل ماهیت خطی تکه‌ای و رفتار غیراشباع‌کننده‌شان، به سرعت بخشیدن به

تمرینات معروف هستند.

• مدیریت روابط پیچیده: برخی از عملکردهای فعالسازی برای مدیریت انواع خاصی از داده‌ها یا

وظایف مناسب‌تر هستند. به عنوان مثال، توابع سیگموئید و  $\tanh$  معمولاً در لایه‌های پنهان شبکه‌های

عصبی استفاده می‌شوند، زیرا می‌توانند ورودی‌ها را در یک محدوده فشرده قرار دهند و به جریان

گرادیان و یادگیری بهتر کمک کنند. تابع فعالسازی Softmax به طور گسترده در لایه خروجی

برای کارهای طبقه‌بندی چند کلاسه استفاده می‌شود، زیرا خروجی‌ها را در یک توزیع احتمال عادی

می‌کند.

• تأثیر بر نزول گرادیان: توابع فعالسازی بر محاسبه و انتشار گرادیان‌ها در طول انتشار پس‌پشتی تأثیر

می‌گذارند، که برای بهروزرسانی پارامترهای شبکه بسیار مهم است. انتخاب تابع فعالسازی شکل

مشتق مورد استفاده در الگوریتم گرادیان نزول را تعیین می‌کند. تابع فعالسازی با مشتقات به

راحتی قابل محاسبه، مانند ReLU، محاسبات گرادیان را ساده می‌کند و روند آموزش را سرعت

می‌بخشد.

## ملاحظات برای انتخاب تابع هزینه و توابع فعالسازی:

هنگام انتخاب تابع هزینه و تابع فعالسازی برای یک شبکه عصبی، چندین فاکتور باید در نظر گرفته شود:

• الزامات Task: ماهیت کار در دست، مانند رگرسیون، طبقه‌بندی باینری، یا طبقه‌بندی چند طبقه،

باید انتخاب تابع هزینه را هدایت کند. باید با معیارهای هدف و ارزیابی کار همخوانی داشته باشد.

- معماری شبکه و دینامیک آموزشی: انتخاب تابع فعال سازی باید معماری شبکه و دینامیک آموزش را تکمیل کند. به عنوان مثال،  $ReLU$  معمولاً در شبکه‌های عصبی عمیق به دلیل توانایی آن در کاهش مشکل گرادیان ناپدید شدن استفاده می‌شود. با این حال، اگر به دقت مدیریت نشود، ممکن است منجر به سلول‌های عصبی مرده شود.
- غیر خطی بودن و پیچیدگی: اگر مشکل روابط غیر خطی را نشان می‌دهد یا نیاز به مدل‌سازی الگوهای پیچیده دارد، توابع فعال‌سازی غیرخطی مانند  $ReLU$  یا sigmoid ممکن است مناسب‌تر باشند. توابع فعال‌سازی خطی برای کارهایی که ذاتاً روابط خطی دارند مناسب هستند.
- ارزیابی عملکرد: عملکرد ترکیب‌های مختلف توابع هزینه و توابع فعال‌سازی باید با استفاده از تکنیک‌های اعتبارسنجی مناسب، مانند اعتبارسنجی متقاطع یا اعتبار سنجی موقت، ارزیابی شود. این امکان انتخاب موثرترین ترکیب را بر اساس معیارهایی مانند دقت، دقت، یادآوری یا میانگین مربعات خطای فراهم می‌کند.

## 2-1-1- تاثیر مقداردهی اولیه وزن‌ها و بایاس‌ها در یادگیری

بله، مقداردهی اولیه وزن‌ها و بایاس‌ها در شبکه‌های عصبی نقش مهمی در فرآیند یادگیری ایفا می‌کند و می‌تواند به طور قابل توجهی بر عملکرد شبکه تأثیر بگذارد. مقداردهی اولیه مناسب به دستیابی به همگرایی سریعتر، اجتناب از ناپدید شدن یا exploding gradients و توانمندسازی شبکه برای یادگیری موثر کمک می‌کند.

گرادیان های در حال ناپدید شدن و یا exploding در طول آموزش، گرادیان ها در طول فرآیند انتشار پس از بین، از طریق لایه های یک شبکه عصبی به سمت عقب منتشر می شوند. اگر وزن ها به طور نامناسب مقداردهی اولیه شوند، گرادیان ها می توانند بسیار کوچک (شیب های در حال ناپدید شدن) یا بسیار بزرگ (شیب های زیاد) شوند. این می تواند روند یادگیری را مختل کند زیرا گرادیان های کوچک ممکن است باعث همگرایی کند شود یا شبکه ممکن است الگوهای پیچیده را یاد نگیرد، در حالی که گرادیان های بزرگ می توانند منجر به یادگیری ناپایدار یا واگرایی شود.

تکنیک های اولیه سازی:

تکنیک های اولیه سازی مختلفی برای کاهش مسائل مربوط به ناپدید شدن و انفجار گرادیان ها ایجاد شده اند.

برخی از تکنیک های رایج مورد استفاده عبارتند از:

- مقدار دهی اولیه تصادفی: راه اندازی تصادفی وزن ها و بایاس ها از یک محدوده کوچک، مانند توزیع یکنواخت یا گاووسی، یک رویکرد رایج است. این به شکستن تقارن و اطمینان از شروع وزنه ها با مقادیر مختلف کمک می کند. با این حال، عملکرد مطلوب را تضمین نمی کند و ممکن است همچنان از شیب های ناپدید یا انفجار رنج ببرد.

Xavier/Glorot Initialization • هدف اولیه سازی Xavier: هدف اولیه سازی Xavier/Glorot Initialization که واریانس فعال سازی ها و گرادیان ها در سراسر شبکه نسبتاً ثابت بماند. این تکنیک تعداد ورودی ها و خروجی های هر لایه را در نظر می گیرد و از مقیاس مناسب برای وزن ها اطمینان می دهد. نشان داده است که در کاهش مشکل گرادیان ناپدید شدن/انفجار در شبکه های عمیق با توابع فعال سازی متقارن موثر است.

• اولیه سازی HE: مقداردهی اولیه HE یک توسعه اولیه از Xavier است و به طور خاص برای شبکه هایی طراحی شده است که از توابع فعال سازی با واحدهای خطی اصلاح شده (ReLU) و انواع آن

استفاده می کنند. تعداد ورودی های هر نورون را در نظر می گیرد و وزن ها را بر این اساس مقایس

می کند. این مقدار دهی اولیه به جلوگیری از مشکل ناپدید شدن گرادیان کمک می کند و در عین

حال مزایای توابع فعال سازی ReLU مانند را حفظ می کند.

تأثیر مقدار دهی اولیه بر یادگیری:

مقدار دهی اولیه صحیح وزن ها و سوگیری ها می تواند تأثیرات مثبت زیر را بر روند یادگیری داشته باشد:

- همگرایی سریعتر: وزنه هایی که به خوبی مقدار دهی شده اند می توانند به همگرایی سریعتر شبکه

در طول تمرین کمک کنند. هنگامی که وزن های اولیه به یک راه حل بهینه نزدیک می شوند، شبکه

برای رسیدن به عملکرد خوب به تکرارهای کمتری نیاز دارد و زمان تمرین را کاهش می دهد.

- جریان گرادیان بهبود یافته: مقدار دهی اولیه مناسب تضمین می کند که گرادیان ها نه خیلی بزرگ

هستند و نه خیلی کوچک، جریان گرادیان صاف تر و پایدار تر را در طول انتشار پس پشتی ترویج

می کند. این امکان به روز رسانی موثر تر پارامترها را فراهم می کند و منجر به یادگیری بهتر می

شود.

- تعمیم پیشرفته: مقدار دهی اولیه مناسب می تواند به جلوگیری از برازش بیش از حد کمک کند،

جایی که شبکه بیش از حد به داده های آموزشی تخصصی می شود و روی داده های دیده نشده

ضعیف عمل می کند. با ارائه یک نقطه شروع خوب، مقدار دهی اولیه مناسب می تواند شبکه را به

سمت تعمیم بهتر و عملکرد بهتر در نمونه های دیده نشده هدایت کند.

در نتیجه راه اندازی وزن ها و سوگیری ها در شبکه های عصبی برای یادگیری مؤثر ضروری است.

مقدار دهی اولیه نامناسب می تواند منجر به ناپدید شدن یا انفجار شبکه ها شود که منجر به همگرایی کند یا

یادگیری ناپایدار می شود. تکنیک های اولیه سازی مناسب، مانند مقدار دهی اولیه تصادفی، مقدار دهی اولیه

Xavier، یا مقدار دهی اولیه He، به رفع این مسائل و ترویج همگرایی سریعتر، بهبود جریان گرادیان و تعمیم

پیشرفتی کمک می کند. انتخاب روش اولیه سازی مناسب به معماری شبکه، توابع فعال سازی و الزامات خاص کار در دست بستگی دارد.

### 1-1-3- تقاضا و Forward propagation

انتشار به جلو و انتشار به عقب فرآیندهای اساسی در آموزش یک شبکه عصبی هستند. آنها مسئول جریان اطلاعات و محاسبه گرادیان های لازم برای به روز رسانی پارامترهای شبکه در طول فرآیند یادگیری هستند. در حالی که انتشار رو به جلو، خروجی پیش‌بینی شده شبکه را با یک ورودی محاسبه می‌کند، انتشار به عقب، گرادیان‌های پارامترهای شبکه را با توجه به یک تابع هزینه معین محاسبه می‌کند و امکان به روز رسانی پارامترها را از طریق گرادیان نزولی (gradient decent) فراهم می‌کند.

انتشار به جلو، همچنین به عنوان عبور به جلو، شامل انتقال داده های ورودی از طریق لایه های شبکه برای تولید یک خروجی پیش‌بینی شده است. این یک جریان متوالی را دنبال می‌کند و داده ها را از لایه ورودی از طریق لایه های پنهان به لایه خروجی پردازش می‌کند.

فرآیند انتشار رو به جلو را می‌توان به صورت زیر خلاصه کرد:

- 1- داده های ورودی به لایه ورودی شبکه عصبی وارد می‌شود.
- 2- داده های ورودی در وزن های مربوط به اتصالات بین لایه ورودی و اولین لایه پنهان ضرب می‌شود.
- 3- سپس مجموع وزنی ورودی ها از طریق یک تابع فعال سازی عبور داده می‌شود که غیرخطی بودن را معرفی می‌کند و خروجی اولین لایه پنهان را تولید می‌کند.
- 4- این فرآیند برای لایه های مخفی بعدی تا رسیدن به لایه خروجی تکرار می‌شود.
- 5- لایه خروجی تابع فعال سازی خود را برای تولید خروجی پیش‌بینی شده نهایی شبکه اعمال می‌کند.

هدف از انتشار رو به جلو، تولید پیش‌بینی‌ها بر اساس وزن‌ها و سوگیری‌های جاری شبکه است. سپس این پیش‌بینی‌ها با خروجی‌های واقعی برای محاسبه تابع هزینه یا ضرر مقایسه می‌شوند، که اختلاف بین مقادیر پیش‌بینی‌شده و واقعی را کمیت می‌دهد.

انتشار به عقب، همچنین به عنوان گذر به عقب یا پس انتشار نیز شناخته می‌شود، فرآیند محاسبه گرادیان پارامترهای شبکه (وزن‌ها و بایاس‌ها) با توجه به تابع هزینه است. این گرادیان‌ها برای به روز رسانی پارامترها و بهبود عملکرد شبکه در طول آموزش استفاده می‌شوند.

فرآیند انتشار به عقب را می‌توان به صورت زیر خلاصه کرد:

- 1- مشتق تابع هزینه با توجه به خروجی پیش‌بینی شده محاسبه می‌شود.
- 2- سپس مشتق از طریق شبکه، لایه به لایه، با استفاده از قانون زنجیره ای حساب دیفرانسیل و انگرال منتشر می‌شود.
- 3- در هر لایه، مشتق در مشتق تابع فعال‌سازی لایه ضرب می‌شود تا گرادیان ورودی‌های لایه محاسبه شود.
- 4- گرادیان وزن‌ها و بایاس‌ها با ضرب گرادیان ورودی‌های لایه در خود ورودی‌ها محاسبه می‌شود.
- 4- گرادیان‌ها انباسته می‌شوند و برای به روز رسانی پارامترهای شبکه با استفاده از یک الگوریتم بهینه سازی، معمولاً گرادیان نزول، استفاده می‌شوند.

هدف از انتشار به عقب تعیین جهت و مقدار به روز رسانی پارامترها است که تابع هزینه را به حداقل می‌رساند. با محاسبه گرادیان پارامترها، شبکه یاد می‌گیرد که چگونه وزن‌ها و بایاس‌های خود را تنظیم کند تا اختلاف بین خروجی‌های پیش‌بینی شده و واقعی را کاهش دهد.

رابطه بین انتشار رو به جلو و عقب:

انتشار رو به جلو و انتشار به عقب فرآیندهای وابسته به هم در آموزش شبکه عصبی هستند. انتشار رو به جلو خروجی پیش‌بینی شده را محاسبه می‌کند که برای ارزیابی عملکرد شبکه و محاسبه تابع هزینه ضروری است. از

طرف دیگر، انتشار به عقب از هزینه محاسبه شده برای محاسبه گرادیان پارامترها استفاده می کند و به روز رسانی آنها را از طریق گرادیان نزول امکان پذیر می کند.

در طول آموزش، شبکه به طور مکرر انتشار را به جلو و عقب را انجام می دهد. پاس رو به جلو پیش بینی ها را تولید می کند و پاس به عقب، گرادیان ها را محاسبه می کند که برای به روز رسانی پارامترها استفاده می شود. این فرآیند تکراری تا زمانی ادامه می یابد که شبکه به سطح رضایت بخشی از عملکرد یا همگرایی برسد.

#### 1-1-4- تاثیر کوچک یا بزرگ بودن مقدار learning rate

نرخ یادگیری یک فرآپارامتر است که اندازه مرحله ای را تعیین می کند که در آن وزن ها و بایاس های یک شبکه عصبی در طول فرآیند آموزش به روز می شوند. نقش مهمی در همگرایی و عملکرد شبکه ایفا می کند. انتخاب نرخ یادگیری، چه بزرگ یا کوچک باشد، می تواند به طور قابل توجهی بر آموزش شبکه عصبی تأثیر بگذارد.

##### تأثیر نرخ یادگیری زیاد:

- یادگیری اولیه سریع: نرخ یادگیری زیاد به شبکه اجازه می دهد تا وزن ها و سوگیری های خود را به سرعت به روز رسانی کند و در نتیجه یادگیری اولیه سریع تر انجام شود. این می تواند هنگام شروع از وزن های تصادفی یا بد مقدار دهی مفید باشد، زیرا به شبکه اجازه می دهد تا فضای پارامتر را به سرعت کشف کند.
- غلبه بر راه حل های بهینه: با این حال، نرخ یادگیری زیاد همچنین می تواند باعث شود که شبکه

بیش از حد یا در حول راه حل بهینه نوسان کند. در چنین مواردی، وزن‌ها و سوگیری‌ها ممکن است

از مقادیر بهینه عبور کنند و منجر به بی‌ثباتی و همگرایی کندر شوند. این می‌تواند از همگرایی

شبکه به یک راه حل بهینه یا نزدیک به بهینه جلوگیری کند.

• مسائل همگرایی: در موارد شدید، نرخ یادگیری بسیار زیاد می‌تواند منجر به واگرایی وزن‌ها و

سوگیری‌ها شود و باعث بدتر شدن عملکرد شبکه شود. به روزرسانی‌ها می‌توانند بیش از حد بزرگ

شوند و یافتن راه حل پایدار را برای شبکه دشوار می‌کند.

### تأثیر نرخ یادگیری کوچک:

بر عکس، زمانی که نرخ یادگیری روی مقدار کمی تنظیم شود، می‌تواند اثرات زیر را داشته باشد:

• همگرایی آهسته: یک نرخ یادگیری کوچک منجر به روزرسانی‌های کوچک در وزن‌ها و

سوگیری‌ها می‌شود که می‌تواند منجر به همگرایی کند شود. شبکه ممکن است به تکرارها یا دوره

های بیشتری نیاز داشته باشد تا به سطح عملکرد قابل قبولی برسد.

• گیر کردن در Local Minima: یک نرخ یادگیری کوچک ممکن است باعث شود شبکه در

حداقل‌های محلی به دام بیفتد و از کاوش موثر فضای پارامتر جلوگیری کند. ممکن است برای

فرار از راه حل‌های غیربهینه تلاش کند و نتواند به حداقل‌های جهانی همگرا شود.

• یادگیری پایدارتر: از جنبه مثبت، یک نرخ یادگیری کوچک می‌تواند پویایی یادگیری پایدارتری

را فراهم کند، و به شبکه اجازه می‌دهد تا به تدریج به راه حل بهینه بدون بیش از حد یا نوسان نزدیک

شود.

### یافتن نرخ یادگیری بهینه:

انتخاب یک نرخ یادگیری مناسب برای آموزش موفق بسیار مهم است. اغلب به آزمایش و تنظیم دقیق

تکراری نیاز دارد. راهبردهای یافتن نرخ یادگیری مطلوب عبارتند از:

- زمان بندی میزان نرخ یادگیری (scheduling) : کاهش تدریجی میزان یادگیری در طول زمان (کاهش نرخ یادگیری) می تواند مفید باشد. با سرعت یادگیری نسبتاً زیادی برای پیشرفت سریع اولیه شروع می شود و با پیشرفت آموزش آن را کاهش می دهد و امکان تنظیمات دقیق تر را فراهم می کند.
  - نرخ های یادگیری تطبیقی: با استفاده از الگوریتم های نرخ یادگیری تطبیقی، مانند Adam، RMSprop یا AdaGrad، می توان به طور خودکار نرخ یادگیری را بر اساس گرادیان های مشاهده شده در طول آموزش تنظیم کرد. این روش ها می توانند نیاز به تنظیم دستی را کاهش دهند و به تعادل بین سرعت همگرایی و ثبات کمک کنند.
  - تست محدوده نرخ یادگیری: انجام تست محدوده نرخ یادگیری شامل آموزش شبکه با نرخ های یادگیری متفاوت در یک محدوده مشخص است. با مشاهده منحنی های از دست دادن یا دقت و انتخاب محدوده ای که شبکه به سرعت بدون نوسان همگرا می شود، به شناسایی نرخ یادگیری مناسب کمک می کند.
- در نتیجه نرخ یادگیری یک فرآپارامتر حیاتی است که بر آموزش شبکه عصبی تأثیر می گذارد. انتخاب نرخ یادگیری مناسب یک عمل متعادل کننده بین سرعت همگرایی و ثبات است. نرخ های زیاد یادگیری ممکن است باعث بی ثباتی شود و از همگرایی جلوگیری کند، در حالی که نرخ های کوچک یادگیری می تواند منجر به همگرایی کند یا گیر کردن در حداقل های محلی شود. تکنیک هایی مانند زمان بندی نرخ یادگیری، نرخ های یادگیری تطبیقی، و آزمون های محدوده نرخ یادگیری می توانند به یافتن نرخ یادگیری بهینه برای آموزش موثر شبکه عصبی کمک کنند.

## ۱-۱-۵- تاثیر تعداد لایه های شبکه عصبی

شبکه های عصبی با لایه های بیشتر، که اغلب به عنوان شبکه های عصبی عمیق یا مدل های یادگیری عمیق شناخته می شوند، نسبت به شبکه های با لایه های کمتر، مزایای متعددی دارند. این مزایا به بهبود عملکرد آنها در کارهای مختلف، از جمله تشخیص الگو، طبقه بندی تصویر، پردازش زبان طبیعی و موارد دیگر کمک می کند. در اینجا دلایل وجود دارد که نشان می دهد شبکه های عصبی با لایه های بیشتر بهتر هستند:

- افزایش قدرت representation: شبکه های عصبی عمیق ظرفیت بالاتری برای نمایش الگوهای روابط پیچیده در داده ها دارند. هر لایه در شبکه ویژگی های انتزاع های متفاوتی را از داده های ورودی می آموزد و استخراج می کند. همانطور که اطلاعات در چندین لایه جریان می یابد، شبکه می تواند نمایش های سلسله مراتبی داده ها را بیاموزد و ویژگی های سطح پایین و سطح بالا را ثبت کند. این قدرت نمایش سلسله مراتبی، شبکه های عمیق را قادر می سازد تا روابط پیچیده و ظریف را مدل سازی کنند، و آنها را برای کارهایی با داده های پیچیده مناسب تر می سازد.

- استخراج و انتزاع ویژگی: با هر لایه اضافی، شبکه های عمیق می توانند ویژگی های انتزاعی و سطح بالا را بیاموزند. لایه های اولیه در شبکه معمولاً ویژگی های ساده مانند لبه ها یا بافتها را یاد می گیرند، در حالی که لایه های عمیق تر ویژگی های پیچیده تری مانند اشکال یا قطعات شی را می آموزند. با ترکیب این ویژگی های آموخته شده در چندین لایه، شبکه های عمیق می توانند ساختارها و معنایی پیچیده را به تصویر بکشند و آنها را قادر می سازد به خوبی به داده های دیده نشده

تعیین دهند.

- یادگیری بهبود یافته بازنمودهای ثابت: شبکه های عمیق می توانند نمایش های ثابت داده ها را که برای مدیریت تغییرات و تبدیل ها ضروری هستند، بیاموزند. نمایش های ثابت نسبت به تغییرات در

ترجمه، چرخش، مقیاس یا سایر تبدیل‌ها در داده‌های ورودی قوی هستند. لایه‌های عمیق‌تر در شبکه

می‌توانند یاد بگیرند که چنین تغییراتی را انتزاع کنند و به شبکه اجازه دهنند الگوهای اشیاء را بدون

توجه به موقعیت، جهت یا مقیاس آنها تشخیص دهد. این ویژگی باعث می‌شود شبکه‌های عمیق

در برابر نویز و تغییرات داده‌های دنیای واقعی سازگارتر و انعطاف‌پذیرتر شوند.

• جریان گرادیان و دینامیک آموزش: شبکه‌های عمیق از جریان گرادیان بهبود یافته در طول انتشار

پس‌پشتی بهره می‌برند، که فرآیند محاسبه گرادیان برای بهروزرسانی وزن است. همانطور که

گرادیان‌ها در لایه‌ها منتشر می‌شوند، می‌توانند سیگنال ارزشمندی را برای به روز رسانی پارامترها

ارائه کنند و به شبکه اجازه می‌دهد تا به طور موثرتری یاد بگیرد. شبکه‌های عمیق با توابع اولیه و

فعال‌سازی مناسب می‌توانند مشکل گرادیان ناپدید شدن را کاهش دهنند و انتشار و بهروزرسانی

مناسب وزن‌ها را برای گرادیان‌ها آسان‌تر کنند. این منجر به همگرایی بهتر و پویایی یادگیری

پایدارتر در شبکه‌های عمیق می‌شود.

• آموزش انتقالی و پیش آموزش: شبکه‌های عمیق اغلب بر روی مجموعه داده‌های مقیاس بزرگ یا

مدل‌های از پیش آموزش دیده شده، که ویژگی‌های کلی را از مقادیر گسترده داده دریافت می‌کنند،

از قبل آموزش داده می‌شوند. این پیش آموزش شبکه را قادر می‌سازد تا نمایش‌های اولیه مفیدی را

یاموزد، که سپس می‌تواند روی وظایف خاص یا مجموعه داده‌های کوچک‌تر تنظیم شود. عمق

شبکه به آن اجازه می‌دهد تا از دانش از پیش آموزش دیده به طور موثر استفاده کند و آن را با کار

خاص تطبیق دهد و در نتیجه عملکرد بهتری با تکرارهای آموزشی کمتر ایجاد کند.

• اجرای مدرن: شبکه‌های عمیق عملکرد پیشرفته‌ای را در حوزه‌های مختلف از جمله تشخیص تصویر،

تشخیص گفتار، پردازش زبان طبیعی و بسیاری موارد دیگر نشان داده‌اند. توانایی آنها در ثبت الگوهای

و روابط پیچیده در داده‌ها، همراه با پیشرفت در الگوریتم‌های بهینه سازی، منابع محاسباتی و

محاسبات موازی، منجر به پیشرفت های قابل توجهی در تحقیقات و کاربردهای هوش مصنوعی شده است.

## محدودیت ها و چالش ها

در حالی که شبکه های عمیق مزایای قابل توجهی را ارائه می دهند، با چالش هایی نیز همراه هستند. شبکه های عمیق از نظر محاسباتی نیاز بیشتری دارند و برای آموزش به داده های برقیسپ دار بیشتری نیاز دارند. تطبیق بیش از حد، ناپدید شدن گرادیان ها، و فعال سازی های ناپدید/انفعار همچنان می توانند چالش هایی در شبکه های عمیق باشند. با این حال، تکنیک های مختلفی مانند منظم سازی، نرمال سازی دسته ای، اتصالات پرش و اتصالات باقیمانده برای کاهش این مسائل و بهبود عملکرد شبکه های عمیق توسعه یافته اند.

در نتیجه، شبکه های عصبی با لایه های بیشتر، که به عنوان شبکه های عصبی عمیق شناخته می شوند، افزایش قدرت بازنمایی، استخراج و انتزاع ویژگی، یادگیری نمایش های ثابت، بهبود جریان گرادیان، قابلیت های یادگیری انتقال، و عملکرد پیشرفتی را ارائه می دهند. این مزایا شبکه های عمیق را قادر می سازد تا وظایف پیچیده را انجام دهند و از مجموعه داده های بزرگ و متنوع بیاموزند، و آنها را به انتخابی ارجح در بسیاری از حوزه ها تبدیل می کند.

### 1-1-1- بزرگترین مشکل شبکه عصبی در مرحله آموزش

یکی از بزرگترین مشکلاتی که در مرحله آموزش شبکه های عصبی با آن مواجه می شود، بحث بیش بر از این است با توجه به اینکه مشکل overfit بیشتر رخ میدهد در اینجا این مورد بررسی می شود. overfit ( ) و underfit ( ) تطبیق بیش از حد زمانی اتفاق می افتد که یک شبکه عصبی یاد می گیرد که روی داده های آموزشی عملکرد خوبی داشته باشد، اما نتواند به داده های جدید و نادیده تعمیم دهد. شبکه بیش از حد به مجموعه آموزشی

تخصصی می شود و الگوها و روابط زیربنایی را که آن را قادر می سازد تا در ورودی های جدید پیش بینی دقیقی انجام دهد، ثبت نمی کند.

### دلایل رخداد مشکل : overfit

تطابق بیش از حد به دلیل توانایی شبکه برای به خاطر سپردن یا تطبیق بیش از حد با داده های آموزشی، به جای یادگیری الگوهای اساسی و تعمیم از آنها، ایجاد می شود. برخی از علل رایج بیش از حد برآذش عبارتند از:

- داده های آموزشی ناکافی: اگر اندازه مجموعه داده آموزشی کوچک باشد، شبکه ممکن است در

عرض تغییرات و سناریوهای مختلف محدود باشد و آن را مستعد بیش از حد برآذش کند. شبکه

ممکن است نمونه های آموزشی را به جای یادگیری بازنمایی های معنادار «به خاطر بسپارد».

- ظرفیت مدل پیچیده: هنگامی که ظرفیت مدل (تعداد پارامترها یا لایه ها) نسبت به پیچیدگی کار یا

داده های آموزشی موجود بیش از حد بزرگ باشد، شبکه به راحتی می تواند بیش از حد مناسب

باشد. مدل ممکن است ظرفیت کافی برای جا دادن نویز یا ویژگی های خاص در داده های آموزشی

داشته باشد که در نتیجه تعمیم ضعیفی دارد.

- عدم regularization: عدم وجود تکنیک های منظم سازی نیز می تواند به بیش از حد برآذش کمک

کند. روش های منظم سازی، مانند منظم سازی L1 یا L2، توقف زودهنگام، به جلوگیری از حساس

شدن بیش از حد شبکه به داده های آموزشی و بهبود توانایی آن در تعمیم کمک می کنند.

### راه حل:

برای رفع مشکل بیش از حد برآذش، می توان از تکنیک های منظم سازی مختلف در طول مرحله آموزش استفاده کرد. این تکنیک ها به جلوگیری از تطبیق بیش از حد شبکه با داده های آموزشی کمک می کنند و به تعمیم بهتر کمک می کنند. برخی از تکنیک های متداول منظم سازی عبارتند از:

- regularization: منظم سازی L1 و L2 یک اصطلاح منظم سازی به تابع ضرر در طول

آموزش اضافه می کند. این عبارات جریمه ای را برای وزن های بزرگ در نظر می گیرند و شبکه را تشویق می کنند تا وزن های کوچک تر و توزیع یکنواخت تری را در اولویت قرار دهند. این مانع از تأکید بیش از حد شبکه به ویژگی های فردی یا تطبیق بیش از حد به اطلاعات پر سر و صدا یا نامربوط می شود.

Drop Out : یک تکنیک منظم سازی است که به طور تصادفی در صد معینی از واحدها

(نرون ها) را در یک لایه در طول هر تکرار آموزشی "قطع" می کند. با انجام این کار، شبکه مجبور به یادگیری نمایش های اضافی می شود و از انطباق مشترک واحدهای خاص جلوگیری می کند.

Drop out به عنوان شکلی از یادگیری گروهی عمل می کند و شبکه را قوی تر می کند و کمتر مستعد بیش از حد برآش است.

توقف زودهنگام ( early stopping ) : توقف زودهنگام شامل نظارت بر عملکرد شبکه بر روی یک مجموعه اعتبارسنجی در طول آموزش است. هنگامی که عملکرد مجموعه اعتبارسنجی شروع به بدتر شدن می کند، آموزش زودتر متوقف می شود و از تطبیق بیشتر شبکه جلوگیری می کند. توقف زودهنگام به یافتن تعادل بین تمرین برای عملکرد بهینه در مجموعه آموزشی و تعمیم به داده های دیده نشده کمک می کند.

افزایش داده ها: تکنیک های افزایش داده ها شامل تولید نمونه های آموزشی جدید با اعمال تبدیل هایی مانند چرخش، مقیاس گذاری یا چرخش به داده های موجود است. این به طور مصنوعی مجموعه آموزشی را گسترش می دهد و نمونه های متنوع تری را برای یادگیری در اختیار شبکه قرار می دهد و اضافه برآش را کاهش می دهد.

نرمال سازی دسته ای ( batch normalization ) : نرمال سازی دسته ای تکنیکی است که فعال سازی های درون هر مینی بچ را در طول آموزش عادی می کند. این به ثبیت فرآیند آموزش با

کاهش تغییر متغیر داخلی کمک می کند، شبکه را نسبت به تغییرات کوچک در ورودی کمتر

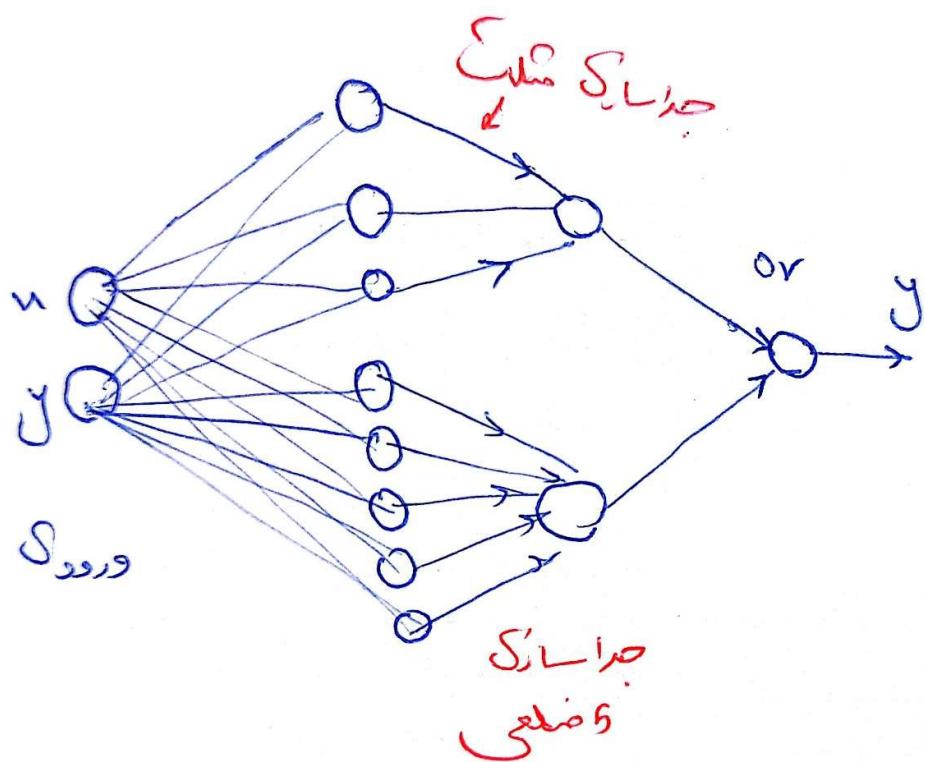
حساس می کند و تعمیم را بهبود می بخشد.

## 1-2- سوال 2 : پیشنهاد معماری MLP

### 1-2-1- الف

در این حالت معماری پیشنهادی استفاده از 2 تا Madeline میباشد که اولی 5 ضلعی را جدا میکند و دومی

نیز مثلث را و در نهایت خروجی ها با هم or میشود.



شکل (1-1) معماری پیشنهادی سوال 2-الف

مطابق شکل فوق ، MLP با 2 لایه مخفی میباشد که در لایه دوم نیازی به اتصال تمام کانکشن ها نمیباشد (

وروودی مختصات میباشد )

## ب-1-2-2

این مساله نیز همانند قبلی میباشد . که تمام آن را با 6 نورون میانی میتوان جدا کرد.

بنابراین معماری پیشنهادی MLP با یک لایه میانی که در لایه میانی 6 نورون وجود دارد میباشد.

## ج-1-2-3

هدف با توجه به شکل پیاده سازی XOR میباشد.

معماری پیشنهادی یک MLP با یک لایه پنهان ( لایه پنهان با 2 نورون ) میباشد. س

## ۳-۱-سوال 3 : کانولوشن

الف ) مشخصه بارز موجود ، وجود یک لوپ مستطیلی یا مربعی تمام 1 در کلاس 1 میباشد در حالیکه در کلاس 2 همچنین چیزی اصلا وجود ندارد.

ب) شکل زیر محاسبات کانولوشن و pooling را نشان میدهد:

با توجه به شکل زیر و محاسبات انجام شده پر واضح است که فیلتر داده شده به دنبال یک لوپ مربعی میگردد و قطعا عدد خروجی کلاس 1 بیشتر میشود و میتوان با یک threshold مثلا در این مساله threshold برابر 3 ، میتوان عمل طبقه بندی را انجام داد.

$$\begin{array}{|c|c|c|} \hline
 \cdot & \cdot & \times \\ \hline
 \times & \times & \times \\ \hline
 \cdot & \times & \times \\ \hline
 \times & \times & \times \\ \hline
 \cdot & \times & \times \\ \hline
 \end{array} \times
 \begin{array}{|c|c|c|} \hline
 \times & \times & \times \\ \hline
 \times & \cdot & \times \\ \hline
 \times & \times & \times \\ \hline
 \times & \times & \times \\ \hline
 \end{array} = \begin{array}{|c|c|} \hline
 1 & 3 \\ \hline
 2 & 5 \\ \hline
 \end{array} \xrightarrow{\text{man pool}} \begin{array}{|c|} \hline
 6 \\ \hline
 \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \end{array} \quad * \quad
 \begin{array}{|c|c|c|c|} \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \backslash & \backslash & \backslash & \backslash \\ \hline
 \end{array} \quad = \quad
 \begin{array}{|c|c|} \hline
 5 & 1 \\ \hline
 5 & 3 \\ \hline
 \end{array} \quad \xrightarrow{\text{man pool}} \quad \begin{array}{|c|} \hline
 15 \\ \hline
 \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline
 \text{A} & \times & \begin{array}{|c|c|c|c|} \hline
 \text{B} & = & \begin{array}{|c|c|} \hline
 0 & 2 \\ \hline
 1 & 1 \\ \hline
 \end{array} & \xrightarrow{\substack{\text{mean} \\ P=0.1}} & \begin{array}{|c|} \hline
 2 \\ \hline
 \end{array} \\ \hline
 \end{array}$$

### شکل (۱-۲) محاسبات کانولوشن مربوط به سوال ۳

## ۱-۴-سوال ۴ : سوالاتی در مورد توابع فعالساز

### ۱-۴-۱-بار محاسباتی کمتر تابع $relu$

بار محاسباتی کمتر تابع فعالسازی  $ReLU$  (واحد خطی اصلاح شده) در مقایسه با سیگموئید و  $tanh$  را می توان به فرمول ساده ریاضی و ماهیت رفتار فعالسازی آن نسبت داد.

#### فرمول ساده ریاضی

تابع  $ReLU$  به صورت زیر تعریف می شود:

$$Relu(X) = \max(0, X)$$

یک مقدار ورودی  $X$  می گیرد و حداکثر آن مقدار و صفر را برمی گرداند. محاسبات مربوط به ارزیابی  $ReLU$  ساده و از نظر محاسباتی کارآمد است. این فقط به یک عملیات مقایسه و یک عملیات حداکثر نیاز دارد که محاسبه هر دو نسبتاً سریع است.

#### فعال سازی پراکنده ( Sparse Activation )

تابع فعال سازی  $ReLU$  دارای خاصیت فعال سازی پراکنده است، به این معنی که تمایل دارد تنها زیر مجموعه ای از نورون ها در یک شبکه عصبی فعال کند. هنگامی که ورودی مثبت است، تابع  $ReLU$  مقدار ورودی را همانطور که هست خروجی می دهد. با این حال، اگر ورودی منفی باشد، خروجی تابع  $ReLU$  صفر است. این پراکنده‌گی در فعال سازی می تواند از نظر بازده محاسباتی سودمند باشد.

از آنجایی که  $ReLU$  همه مقادیر منفی را صفر می کند، به طور موثر تعداد محاسبات مورد نیاز برای انتشار به جلو و عقب را کاهش می دهد. نورون های با فعال سازی صفر در محاسبات کلی و شبکه ها در طول انتشار پس پشتی کمکی نمی کنند، که منجر به کاهش بار محاسباتی می شود.

حذف محاسبات نمایی:

در مقابل، توابع فعال سازی سیگموئید و  $\tanh$  شامل محاسبات نمایی است. سیگموئید به صورت  $\tanh(x) = (\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x))$  و  $\text{sigmoid}(x) = 1 / (1 + \exp(-x))$  به صورت سیگموئید تعريف می شود. این محاسبات نمایی در مقایسه با مقایسه های ساده و حداکثر عملیات  $\text{ReLU}$  از نظر محاسباتی گران تر هستند.

عدم وجود محاسبات نمایی در  $\text{ReLU}$  باعث می شود که از نظر محاسباتی سریع تر شود، به ویژه زمانی که با شبکه های عصبی در مقیاس بزرگ یا معماری های عمیق با پارامتر های متعدد سروکار داریم.

## مشکل $\text{ReLU}$ را vanishing gradient ندارد

مشکل ناپدید شدن گرادیان به پدیده ای اشاره دارد که در آن گرادیان های محاسبه شده در طول فرآیند آموزش در شبکه های عصبی عمیق، با انتشار به عقب از لایه خروجی به لایه های اولیه، بسیار کوچک می شوند. این مسئله توانایی شبکه را برای یادگیری موثر، به ویژه در معماری های عمیق با لایه های زیاد، مختل می کند.

مشکل گرادیان ناپدید شدن:

مشکل ناپدید شدن گرادیان به دلیل ویژگی های توابع فعال سازی مانند سیگموئید و  $\tanh$  رخ می دهد که دارای مناطق اشباع شده است. در این مناطق اشباع، مشتق (گرادیان) تابع فعال سازی بسیار کوچک می شود.

به عنوان مثال، در تابع فعال سازی سیگموئید:

$$\text{sigmoid}(x) = 1 / (1 + \exp(-x))$$

با دور شدن مقادیر ورودی از صفر در جهت مثبت یا منفی، تابع سیگموئید اشباع می شود، به این معنی که

مقادیر خروجی به 0 یا 1 نزدیک می شوند.

به طور مشابه، تابع  $\tanh$  به صورت:

$$\tanh(x) = (\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x))$$

همچنین برای ورودی های مثبت یا منفی بزرگ اشباع می شود که منجر به شیب های کوچک در آن مناطق می شود.

هنگام انتشار مجدد گرادیان ها از لایه خروجی به لایه های اولیه، از قانون زنجیره ای برای محاسبه گرادیان در هر لایه با ضرب گرادیان لایه های بعدی استفاده می شود. اگر این گرادیان ها کوچک باشند (نرديک به صفر)، ضرب آنها منجر به شیب های کوچک تری می شود که در شبکه به عقب منتشر می شوند. در نتیجه، لایه های اولیه سیگنال های گرادیان بسیار ضعیفی دریافت می کنند که مانع از توانایی آنها در به روز رسانی وزن هایشان به طور موثر و مانع از فرآیند یادگیری می شود.

ReLU و راه حل:

ReLU با داشتن یک رفتار فعال سازی غیر اشباع، بر مشکل گرادیان ناپدید غلبه می کند. تابع ReLU مقدار ورودی را همانطور که مثبت است برمی گرداند و اگر ورودی منفی باشد خروجی صفر می دهد. این رفتار خطی تکه ای از اشباع جلوگیری می کند و به گرادیان ها اجازه می دهد در ناحیه فعال سازی مثبت بزرگ و ثابت بمانند. در نتیجه، نورون های ReLU می توانند سریع تر و مؤثر تر یاد بگیرند، زیرا از ناپدید شدن گرادیان رنج نمی برنند. شیب ها در ناحیه مثبت محدود یا کاهش نمی یابند و سیگنال های پس انتشار قوی تر را به لایه های قبلی تسهیل می کنند. این ویژگی ReLU آن را برای شبکه های عصبی عمیق مناسب می کند و مشکل ناپدید شدن گرادیان را کاهش می دهد.

با این حال، مهم است که توجه داشته باشید که ReLU می تواند با یک مشکل مرتبط به نام مشکل "ReLU در حال مرگ" مواجه شود، که در آن برخی از نورون ها می توانند به طور دائم در طول آموزش غیرفعال شوند (خروجی صفر). این مشکل را می توان با استفاده از انواع ReLU، مانند Leaky ReLU یا Parametric ReLU که شیب های کوچکی را برای ورودی های منفی برای جلوگیری از مرگ نورون معرفی می کند، کاهش داد.

### 3-4-1- استفاده از sigmoid در لایه آخر برای مساله طبقه بندی

در یک مشکل طبقه بندی، استفاده از تابع فعال سازی سیگموئید در آخرین لایه برای وظایف طبقه بندی بایزی در فعال سازی softmax برای وظایف طبقه بندی چند کلاسه معمول است. انتخاب تابع فعال سازی به الزامات و ویژگی های خاص مشکل بستگی دارد.

تابع فعال سازی سیگموئید که به عنوان تابع لجستیک نیز شناخته می شود، ورودی را به مقداری بین 0 و 1 ترسیم می کند. معمولاً در مسائل طبقه بندی بایزی استفاده می شود، جایی که هدف طبقه بندی ورودی ها به یکی از دو کلاس است (به عنوان مثال بله / خیر) تابع سیگموئید برای این کار مناسب است زیرا می تواند یک خروجی شبیه به احتمال تولید کند که نشان دهنده احتمال تعلق ورودی به یک کلاس خاص است. در حالیکه  $relu$  با توجه به ماهیت تابع آن این خاصیت را ندارد.

در تابع سیگموئید با آستانه گذاری خروجی سیگموئید در 0.5، می توانیم ورودی را به کلاس مثبت اگر خروجی بزرگتر یا مساوی 0.5 باشد و در غیر این صورت به کلاس منفی اختصاص دهیم.

فعال سازی سافت مکس:

تابع فعال سازی softmax در مسائل طبقه بندی چند کلاسه استفاده می شود، جایی که ورودی می تواند به یکی از چندین کلاس تعلق داشته باشد. Softmax خروجی ها را به یک توزیع احتمال بر روی کلاس ها عادی می کند و اطمینان حاصل می کند که مجموع احتمالات به 1 می رسد.

تابع softmax به صورت زیر تعریف می شود:

$$\text{softmax}(x_i) = \exp(x_i) / \sum(\exp(x_j)) \text{ for all } j$$

در اینجا  $x_i$  ورودی تابع softmax را نشان می دهد و  $\sum x_j$  ورودی های تابع را برای همه کلاس ها نشان می دهد.

فعال سازی softmax احتمالات بالاتری را به کلاس هایی با ورودی های بزرگتر اختصاص می دهد و آن را

برای انتخاب محتمل ترین کلاس از بین گزینه‌های متعدد مناسب می‌سازد.

انتخاب بین فعال سازی سیگموئید و سافت مکس در آخرین لایه به مشکل طبقه بندی خاص بستگی دارد.

اگر مشکل باینری باشد، معمولاً از فعال سازی سیگموئید استفاده می‌شود. با این حال، اگر مشکل شامل چندین کلاس باشد، فعال سازی softmax ترجیح داده می‌شود زیرا یک توزیع احتمال نرمال شده را در تمام کلاس‌ها ارائه می‌دهد.

#### 1-4-4- فرق بین ReLU و Leaky ReLU

ReLU و Leaky ReLU هر دو توابع فعال‌سازی هستند که در شبکه‌های عصبی استفاده می‌شوند، اما در نحوه مدیریت مقادیر ورودی منفی با هم تفاوت دارند.

ReLU در صورت مثبت بودن مقدار ورودی  $x$  و اگر ورودی منفی باشد صفر را برمی‌گرداند. به عبارت دیگر، ReLU تمام مقادیر منفی را صفر می‌کند، در حالی که مقادیر مثبت را بدون تغییر می‌گذراند.

: Leaky ReLU

یک نسخه اصلاح شده از ReLU است که شبکه کوچکی را برای مقادیر ورودی منفی Leaky ReLU معرفی می‌کند. فرمول ریاضی Leaky ReLU به صورت زیر است:

$$\text{Leaky ReLU}(x) = \max(a^*x, x)$$

در اینجا  $a$  یک ثابت مثبت کوچک است (معمولاً کسری کوچک مانند 0.01).

به جای تنظیم ورودی‌های منفی دقیقاً روی صفر، Leaky ReLU یک خروجی کوچک و غیر صفر را برای ورودی‌های منفی اجازه می‌دهد. این شبکه کوچک به کاهش مشکل "ReLU در حال مرگ" کمک می‌کند، جایی که نورون‌های ReLU در صورت دریافت گرادیان‌های منفی بزرگ می‌توانند در طول آموزش به طور دائم غیرفعال شوند. Leaky ReLU تضمین می‌کند که حتی برای ورودی‌های منفی، یک گرادیان کوچک وجود

دارد که می‌تواند در حین انتشار به عقب منتشر شود.

### زمان استفاده از Leaky ReLU به جای ReLU

معمولًاً در سناریوهای خاصی بر ReLU ترجیح داده می‌شود:

- اجتناب از نورون‌های مرده: نورون‌های ReLU می‌توانند «مرده» شوند، اگر به‌طور مداوم در حین

آموزش گرادیان‌های منفی دریافت کنند، که باعث می‌شود همیشه خروجی صفر داشته باشند.

Leaky ReLU با ارائه یک گرادیان کوچک برای ورودی‌های منفی و جلوگیری از مرگ نورون

به کاهش این مشکل کمک می‌کند. از این‌رو، Leaky ReLU هنگامی که با شبکه‌های عصبی

عمیق یا معماری‌هایی که احتمال وجود نورون‌های مرده بیشتر است، مورد استفاده قرار می‌گیرد.

- مدیریت ورودی‌های منفی: ReLU ورودی‌های منفی را صفر می‌کند و به‌طور موثر هرگونه

اطلاعات منفی را دور می‌زند. در برخی موارد، حفظ مقادیر منفی کوچک ممکن است سودمند

باشد، به خصوص زمانی که با داده‌هایی سروکار داریم که محدوده معنی‌داری در حدود صفر

دارند. Leaky ReLU امکان حفظ مقادیر منفی کوچک را فراهم می‌کند، که می‌تواند برای

گرفتن الگوها یا اطلاعات ظریف در داده‌ها مفید باشد.

- انتشار گرادیان: Leaky ReLU جریان گرادیان‌ها را حتی برای ورودی‌های منفی تضمین می‌کند،

که می‌تواند به انتشار گرادیان بهتر در حین انتشار backward کمک کند. این جریان گرادیان

بهبود یافته می‌تواند به همگرایی سریعتر و پایدارتر در طول تمرین کمک کند.

با این حال، مهم است که توجه داشته باشید که انتخاب بین ReLU و Leaky ReLU وابسته به task است

و عملکرد می‌تواند بر اساس وظیفه و مجموعه داده خاص متفاوت باشد. در برخی موارد، ReLU ممکن است

همچنان انتخاب ارجح باشد، به‌ویژه زمانی که با مجموعه‌های داده سروکار داریم که مقادیر منفی معنی‌دار نیستند

یا زمانی که مسئله نورون‌های مرده حداقل است.

به طور کلی، Leaky ReLU یک جایگزین مفید برای ReLU در شرایطی است که حفظ مقادیر منفی کوچک و اجتناب از نورون‌های مرده مورد نظر است.

### ۱-۵- سوال ۵: محاسبات دستی درخت تصمیم

محاسبات مربوط به بخش الف و ب در عکس های زیر آمده است:

الف) ایک جانب آئندگی استعمال پر احتال ہر رخدا صول یا رد نہیں اور

### شكل (٣-١) محاسبات بخش الف سوال ٥

ب) برای گام بعد احتمال کرجی حرف (استم) را بازیابی کنیم

$$\begin{array}{c}
 \text{تحصیلات} \rightarrow \text{لیسان} \rightarrow \begin{cases} P(\text{reject} \mid \text{bachelor}) = \frac{1}{4} \\ P(\text{accept} \mid \text{bachelor}) = \frac{3}{4} \end{cases} \text{بررسی ساده} \\
 \text{دانشگاهی} \\
 \text{ارزش} \rightarrow \begin{cases} P(\text{reject} \mid \text{master}) = \frac{3}{4} \\ P(\text{accept} \mid \text{master}) = \frac{1}{4} \end{cases} \\
 \text{دکترا} \rightarrow \begin{cases} P(\text{reject} \mid \text{Drs.}) = 1 \\ P(\text{accept} \mid \text{Drs.}) = 0 \end{cases} \text{صریح}
 \end{array}$$

$$\begin{aligned}
 H(\text{degree, employment}) &= P(\text{bachelor}) H\left(\frac{1}{4}, \frac{3}{4}\right) + P(\text{master}) H\left(\frac{3}{4}, \frac{1}{4}\right) + P(\text{Drs.}) H(1) \\
 &= \frac{4}{10} \times 0.81 + \frac{4}{10} \times 0.81 = 0.648
 \end{aligned}$$

$$\begin{aligned}
 \text{Information gain (degree)} &= H(\text{employment}) - H(\text{degree, employment}) \\
 &= 0.97 - 0.648 = \underline{\underline{0.322}} \text{ gain}
 \end{aligned}$$

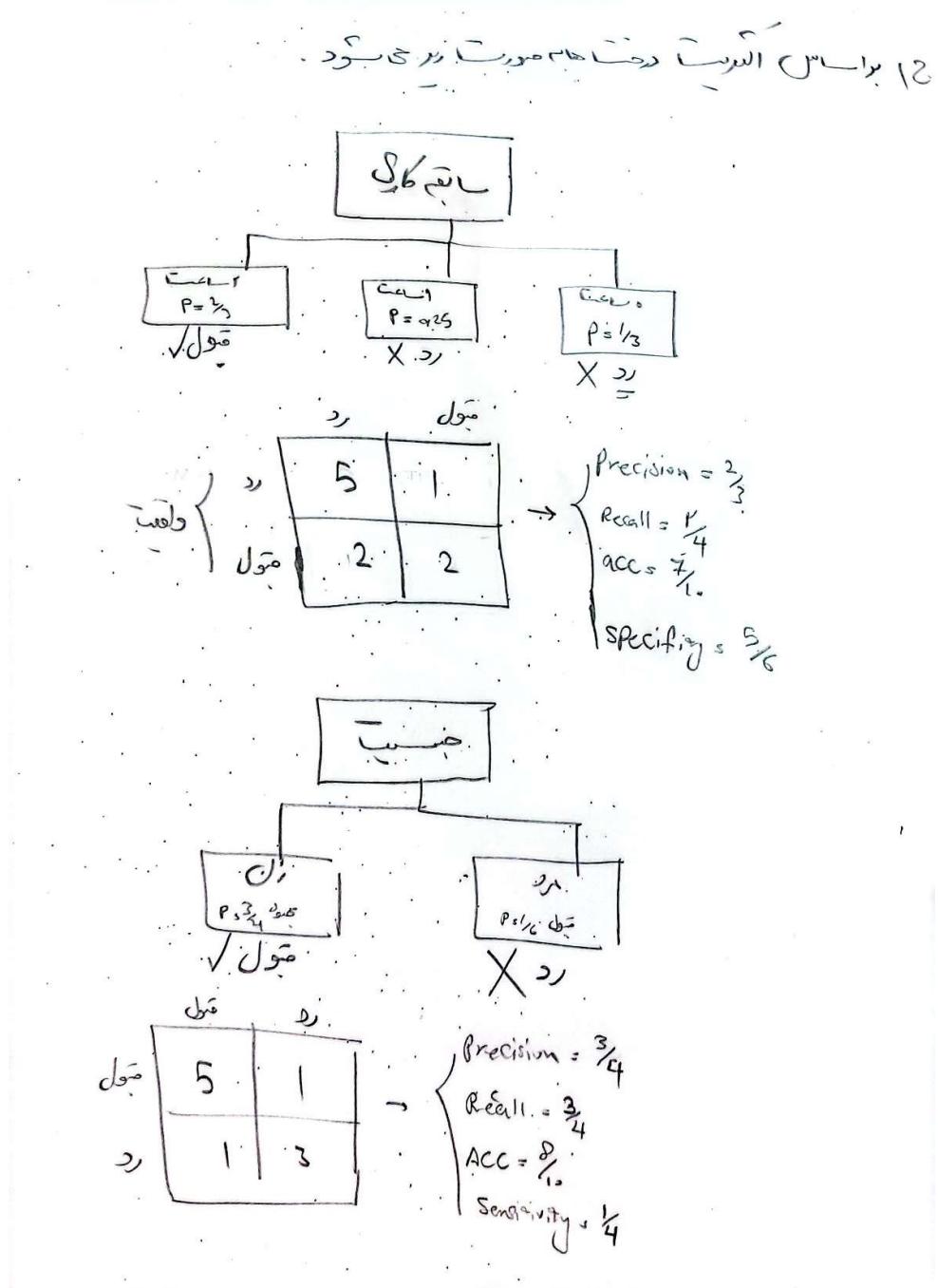
بررسی اور در برای  $\text{inf. gain}$  ساقه طرح:

ساقه طرح	میانگین	ردی
۰	۱	۲
۱	۱	۳
۲	۲	۱

$$\begin{aligned}
 H(\text{experience, employment}) &= P(y=0) H\left(\frac{1}{3}, \frac{2}{3}\right) + P(y=1) H\left(\frac{1}{4}, \frac{3}{4}\right) \\
 &\quad + P(y=2) H\left(\frac{1}{3}, \frac{2}{3}\right) \\
 &= \frac{3}{10} (0.92) + \frac{4}{10} (0.81) + \frac{3}{10} (0.92) = 0.876
 \end{aligned}$$

$$\begin{aligned}
 \text{Information gain (experience, employment)} &= \underline{\underline{0.97 - 0.876}} \\
 &= \underline{\underline{0.1094}} \text{ gain}
 \end{aligned}$$

ج) تصویر زیر موارد مورد سوال بررسی شده است.



### شكل (٥-١) جواب بخش ج سوال ٥

## ۱-۶-سوال ۶: طبقه بندی داده ای ECG

### ۱-۶-۱-لود داده ها و بررسی توزیع داده ها:

ابتدا به کمک کتابخانه pandas داده ها را لود کرده و تعداد داده های هر کلاس و هیستوگرام آن را به

کمک دستور های زیر بررسی و رسم میکنیم.

```
df = pd.read_csv('ECG.csv')
df.head()
```

شکل (۱-۶) کد پایتون لود فایل csv داده ها

```
class_counts = df['label'].value_counts()
print(class_counts)
```

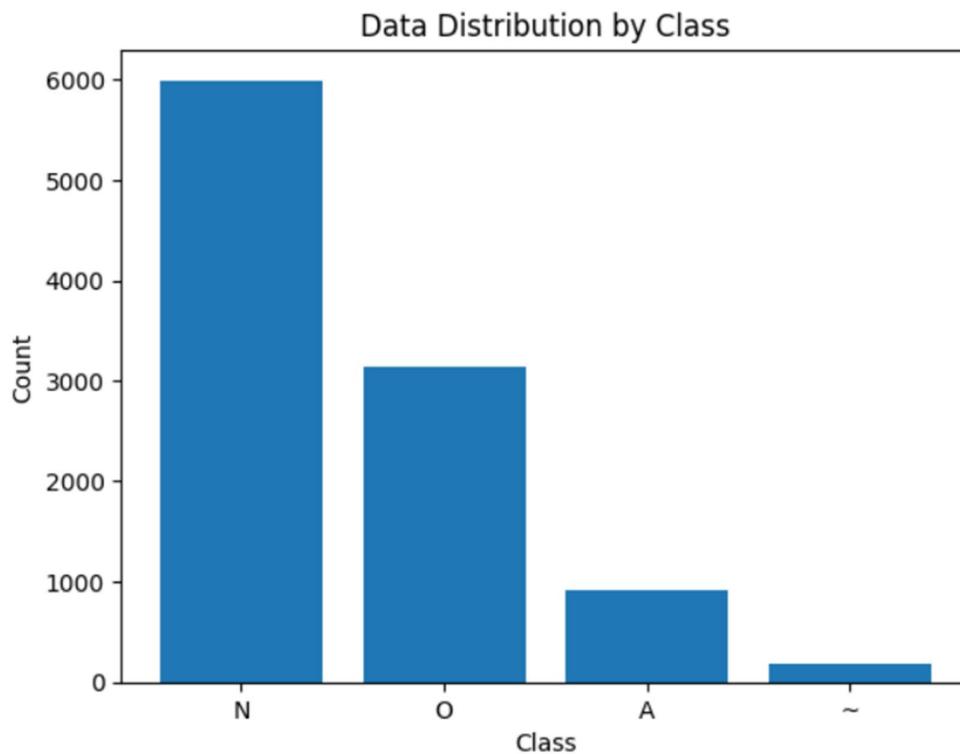
```
N      5992
O      3151
A      923
~      187
Name: label, dtype: int64
```

شکل (۱-۷) کد پایتون بررسی تعداد داده های هر کلاس

```

plt.bar(class_counts.index, class_counts.values)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Data Distribution by Class')
plt.show()

```



شکل (۱-۸) کد پایتون و منحنی توزیع داده ها بر حسب هر کلاس

با توجه به داده های فوق بیشترین توزیع برای داده های کلاس N و کمترین توزیع برای داده های کلاس ~ میباشد.

توزیع نابرابر داده ها در یک مشکل طبقه بندی می تواند چندین مشکل ایجاد کند:

- تعصب نسبت به کلاس اکثریت: وقتی یک کلاس بر مجموعه داده تسلط دارد، طبقه بندی کننده

ممکن است به سمت پیش بینی کلاس اکثریت سوگیری پیدا کند. این سوگیری می تواند منجر به

عملکرد ضعیف در کلاس اقلیت شود که منجر به کاهش دقت و قدرت پیش بینی آن کلاس می

شود.

• تعمیم ضعیف: داده های نامتعادل می تواند منجر به تعمیم ضعیف مدل شود. این مدل ممکن است

بیش از حد با طبقه اکثریت مطابقت داشته باشد و نتواند الگوها و ویژگی های طریف طبقه اقلیت را

به تصویر بکشد. در نتیجه، مدل ممکن است برای طبقه بندی دقیق نمونه های طبقه اقلیت در داده های

دیده نشده مشکل داشته باشد.

• معیارهای ارزیابی گمراه کننده: معیارهای ارزیابی مانند دقت در حضور داده های نامتعادل می تواند

گمراه کننده باشد. دقت بالایی که با پیش بینی کلاس اکثریت به دست می آید ممکن است عملکرد

ضعیف را در کلاس اقلیت پنهان کند. در نظر گرفتن معیارهای ارزیابی اضافی مانند دقت، یادآوری،

و امتیاز F1 که به صراحة عملکرد هر کلاس را محاسبه می کند، ضروری است.

• مشکل در یادگیری کلاس های نادر: نمایش محدود کلاس اقلیت در مجموعه داده ها، یادگیری

ویژگی های متمایز آن را برای مدل چالش برانگیز می کند. مدل ممکن است نمونه های کافی برای

یادگیری کافی ویژگی های طبقه اقلیت نداشته باشد، که منجر به کاهش عملکرد طبقه بندی در

موارد نادر می شود.

• سوگیری مبتنی بر داده: مجموعه داده های نامتعادل می توانند سوگیری را در فرآیند یادگیری ایجاد

کنند، جایی که مدل سوگیری های موجود در داده های آموزشی را یاد می گیرد و تداوم می بخشد.

این می تواند پیامدهای اخلاقی و اجتماعی داشته باشد، به ویژه اگر طبقه اقلیت نماینده گروه های

محروم یا به حاشیه رانده باشد.

## آموزش یک شبکه MLP با چهار لایه مخفی بر روی دیتای خام

در ابتدا هیچ پردازش یا تغییری روی داده ها انجام نمی‌دهیم.

تنها کاری که باید بکنیم این است که ابتدا ستون اول را حذف کنیم چرا که تنها شماره داده ها می‌باشد به

علاوه کلاس هایی که با A و N و ~ نامگذاری شده است به اعداد map کنیم.

	Unnamed: 0	label	feat1	feat2	feat3	feat4	feat5	feat6	feat7	feat
0	0	O	5	661.190476	658.333333	55.278736	106.962730	111.113889	13	0.92857
1	1	O	5	661.666667	658.333333	56.579873	108.407273	112.510209	13	0.92857
2	2	O	8	661.904762	660.000000	58.758004	112.796617	117.091412	13	0.92857
3	3	O	7	660.952381	663.333333	59.029597	113.649634	117.981601	13	0.92857
4	4	O	5	660.476191	663.333333	55.759820	107.623910	111.732026	13	0.92857

5 rows × 171 columns

شکل (۹) دیتافریم اولیه و دلیل حذف ستون اول

```
# Map class labels to integers
label_mapping = {'A': 0, 'N': 1, 'O': 2, '~': 3}
df['label'] = df['label'].map(label_mapping)

df = df.iloc[:, 1:]
df.head()
```

شکل (۱۰) کد پایتون مپ کردن لیل ها و حذف ستون اول

	label	feat1	feat2	feat3	feat4	feat5	feat6	feat7	f
0	2	5	661.190476	658.333333	55.278736	106.962730	111.113889	13	0.92
1	2	5	661.666667	658.333333	56.579873	108.407273	112.510209	13	0.92
2	2	8	661.904762	660.000000	58.758004	112.796617	117.091412	13	0.92
3	2	7	660.952381	663.333333	59.029597	113.649634	117.981601	13	0.92
4	2	5	660.476191	663.333333	55.759820	107.623910	111.732026	13	0.92

5 rows × 170 columns

شکل (۱-۱۱) خروجی دیتا فریم پس از تغییرات اعمال شده

در ادامه ستون لیبل را در y و مابقی ستون ها در x قرار داده و داده ها را به صور ترندوم ۹۰ به ۱۰ برای

آموزش و ولیدیشن تقسیم میکنیم.

```
# Split the data into features (X) and labels (y)
X = df.iloc[:, 1:] # Extract all columns except the 'label'
y = df['label']

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

شکل (۱-۱۲) کد پایتون تقسیم داده های فیچر و لیبل ECG

در ادامه جهت ورودی دادن به شبکه نیاز است تا داده های label را one hot کرده و داده های X را هم به

numpy array تبدیل کنیم.

```
# Convert data to numpy arrays
X_train = X_train.values
X_test = X_test.values
y_train = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=4)
```

شکل (۱-۱۳) کد پایتون تبدیل داده های X به one-hot numpy array و کردن لیبل ها

در نهایت مدل را آماده کرده و آموزش را انجام میدهیم.

```

# Create a sequential model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(100, activation='relu', input_shape=(169,)),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
])

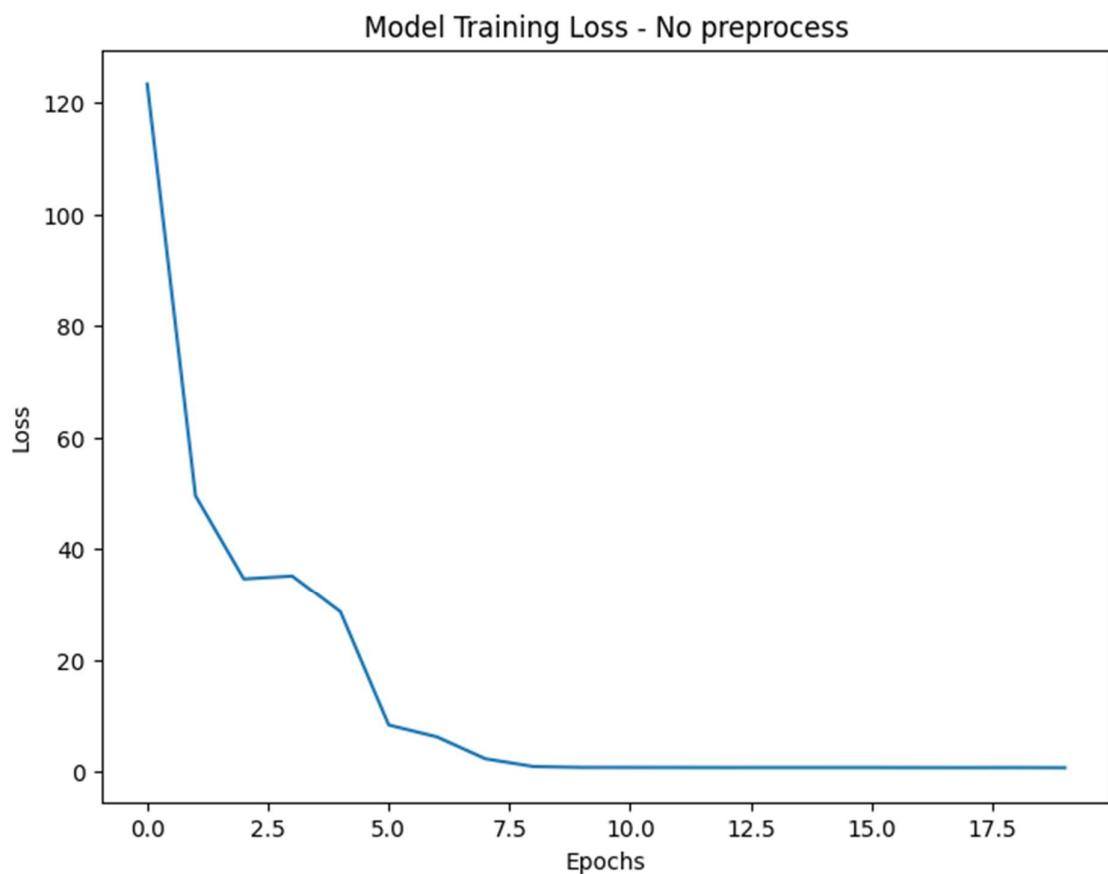
# Compile
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
History = model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=1)

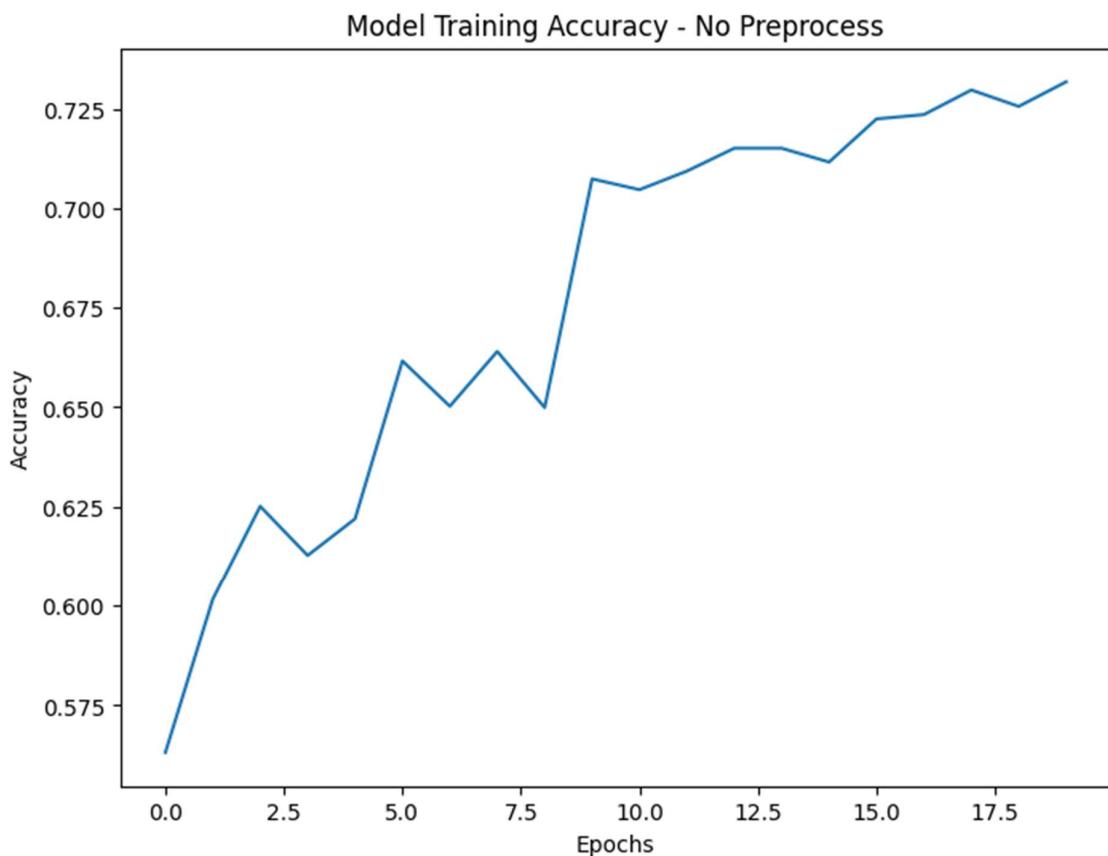
```

شکل (۱-۱۴) کد پایتون پیاده سازی مدل و کامپایل و آموزش مدل

شکل های زیر منحنی های loss و accuracy آموزش مدل را نشان میدهد.



شکل (۱-۱۵) منحنی loss مدل تحت آموزش در حالت بدون هیچ پردازشی



شکل (۱-۱۶) منحنی دقت شبکه تحت آموزش بر روی داده های val در حالت بدون هیچ پیش پردازشی

لازم به ذکر است با توجه به  $\log$  آموزش شبکه و تصاویر فوق در نهایت به مقادیر زیر در داده های

validation رسیده است :

loss: 0.6582 - accuracy: 0.7318

در ادامه به کمک کد زیر بر روی داده های تست که اصلا شبکه آن ها را ندیده است نتایج را گرفته و

precison و recall و F1score و ماتریس آشتفتگی را بررسی میکنیم.

```

# Make predictions on the test data
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Convert one-hot encoded labels to integers
y_test_classes = np.argmax(y_test, axis=1)

33/33 [=====] - 0s 2ms/step

```

```

# Calculate evaluation metrics
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1score = f1_score(y_test_classes, y_pred_classes, average='weighted')
confusion = confusion_matrix(y_test_classes, y_pred_classes)

```

شکل (۱-۱۷) کد پایتون گرفتن خروجی بر روی داده های تست و بررسی recall و precision و f1score

```

Precision: 0.7019
Recall: 0.7203
F1-Score: 0.7063
Confusion Matrix:
[[ 34  11  53  0]
 [  7 513  69  1]
 [ 12 117 192  0]
 [  1    3  13  0]]

```

شکل (۱-۱۸) مقادیر Precision و recall و F1 و ماتریس آشفتگی در حالت بدون هیچ پیش پردازشی

برای بررسی آنکه دقت پیشینی در کدام کلاس بیشتر بوده از کد زیر استفاده میکنیم و آن را بر روی ماتریس آشفتگی به دست آمده اعمال میکنیم.

لازم به ذکر است با توجه به شکل زیر که mapping را نشان میدهد کلاس ۱ همان کلاس N که بیشترین توزیع را داشت میباشد.

```

# Map class labels to integers
label_mapping = {'A': 0, 'N': 1, 'O': 2, '~":" 3}
df['label'] = df['label'].map(label_mapping)

```

```

# Get the diagonal elements (true positive predictions)
TP_0 = confusion[0][0]
TP_1 = confusion[1][1]
TP_2 = confusion[2][2]
TP_3 = confusion[3][3]

# Calculate the total number of instances for each class
total_0 = sum(confusion[0])
total_1 = sum(confusion[1])
total_2 = sum(confusion[2])
total_3 = sum(confusion[3])

# Calculate the accuracy for each class
accuracy_0 = TP_0 / total_0
accuracy_1 = TP_1 / total_1
accuracy_2 = TP_2 / total_2
accuracy_3 = TP_3 / total_3

# Print the accuracy for each class
print(f"Accuracy for class 0: {accuracy_0:.4f}")
print(f"Accuracy for class 1: {accuracy_1:.4f}")
print(f"Accuracy for class 2: {accuracy_2:.4f}")
print(f"Accuracy for class 3: {accuracy_3:.4f}")

```

شکل (۱-۱۹) کد پایتون بررسی میزان دقت در هر کلاس

```

Accuracy for class 0: 0.3469
Accuracy for class 1: 0.8695
Accuracy for class 2: 0.5981
Accuracy for class 3: 0.0000

```

شکل (۱-۲۰) خروجی کد بررسی دقت روی هر کلاس

مشاهده میشود کلاس N که بیشترین توزیع را داشته بیشترین دقت و کلاس ~ که کمترین توزیع را داشته

است کمترین دقت (دقت ۰) را دارد. و همان مشکلات ذکر شده در بخش قبل در مورد توزیع نامتوازن در این

حالت مشاهده شد.

### ۱-۶-۳- نرمال سازی داده ها

در این حالت ابتدا داده های  $X$  را نرمال کرده و سپس همان مراحل قبل را تکرار میکنیم.

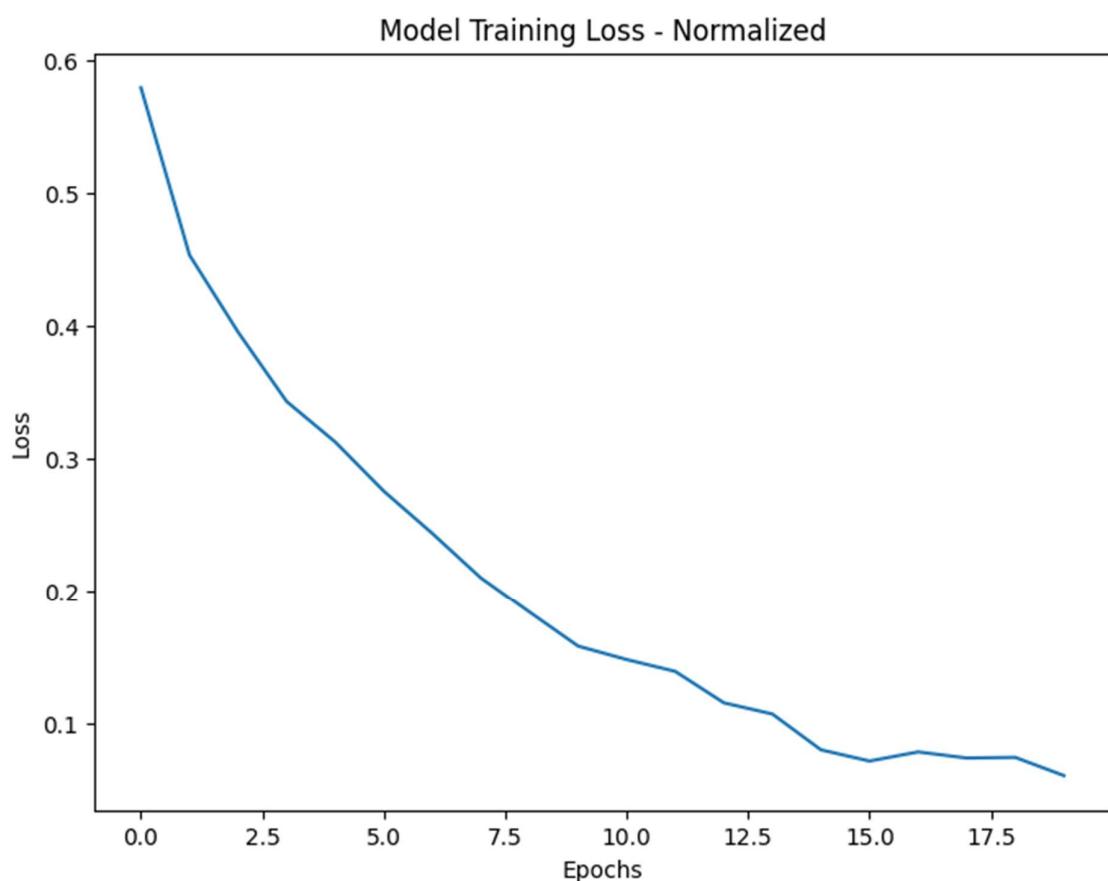
```
# Normalize the feature data using standard normalization
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

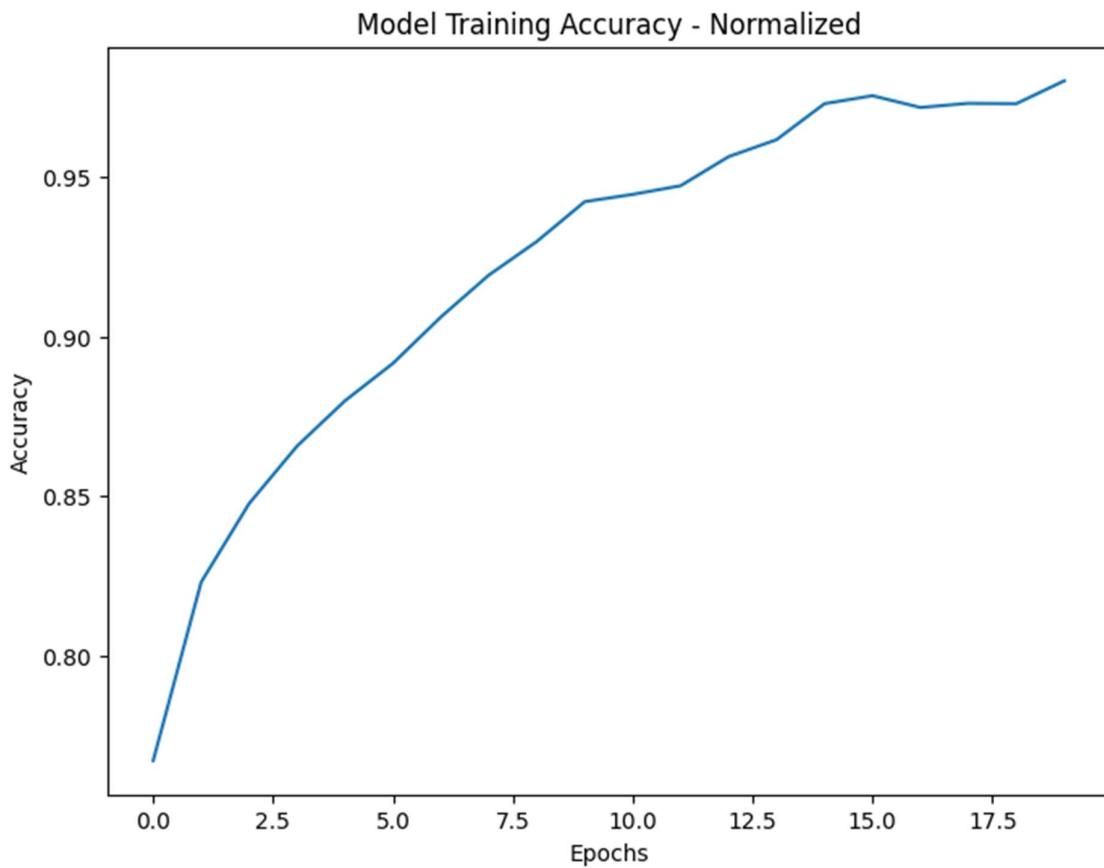
# Convert data to numpy arrays
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=4)
```

شکل (۱-۲۱) کد پایتون نرمال سازی داده ها و تقسیم داده ها

در ادامه نیز مانند قبل مراحل آموزش را تکرار میکنیم که خروجی ها به صورت زیر میباشد.



شکل (۱-۲۲) منحنی loss آموزش – در حالت نرمالسازی داده ها



شکل (۱-۲۳) منحنی دقت شبکه روی داده های val – در حالت نرمالسازی داده ها

مشاهده میشود که loss: 0.0604 - accuracy: 0.9801 که نشان از این مورد دارد که شبکه وقتی پیش

پردازش نرمالسازی را کردیم خیلی خیلی بهتر شد.

اما دقت را بروی داده های تست باید بررسی کنیم.

```
Precision: 0.8454
Recall: 0.8333
F1-Score: 0.8368
Confusion Matrix:
[[ 79  3 13  3]
 [ 4 497 81  8]
 [ 2  41 273  5]
 [ 0   6   5  6]]
```

شکل (۱-۲۴) مقادیر recall و precision و f1 و ماتریس آشفتگی روی داده تست – در حالت نرمالسازی داده ها

```
Accuracy for class 0: 0.8061
Accuracy for class 1: 0.8424
Accuracy for class 2: 0.8505
Accuracy for class 3: 0.3529
```

شکل (۱-۲۵) میزان دقت شبکه بر روی داده های تست بر حسب هر کلاس

مشاهده میشود که دقت ها خیلی بهتر شده اما همچنان دقت خوبی بر روی داده های کلاس چهارم وجود ندارد.

## ۱-۶-۴- تبدیل مساله به طبقه بند باینری abnormal و normal

با توجه به صورت مساله ، mapping و داده ها را به صورت زیر تعریف میکیم .

```
df = pd.read_csv('ECG.csv')

df_filtered = df[df['label'].isin(['N', 'O', 'A'])]

# Map class labels to integers
label_mapping = {'N': 0, 'O': 1, 'A': 1}
df_filtered['label'] = df_filtered['label'].map(label_mapping)

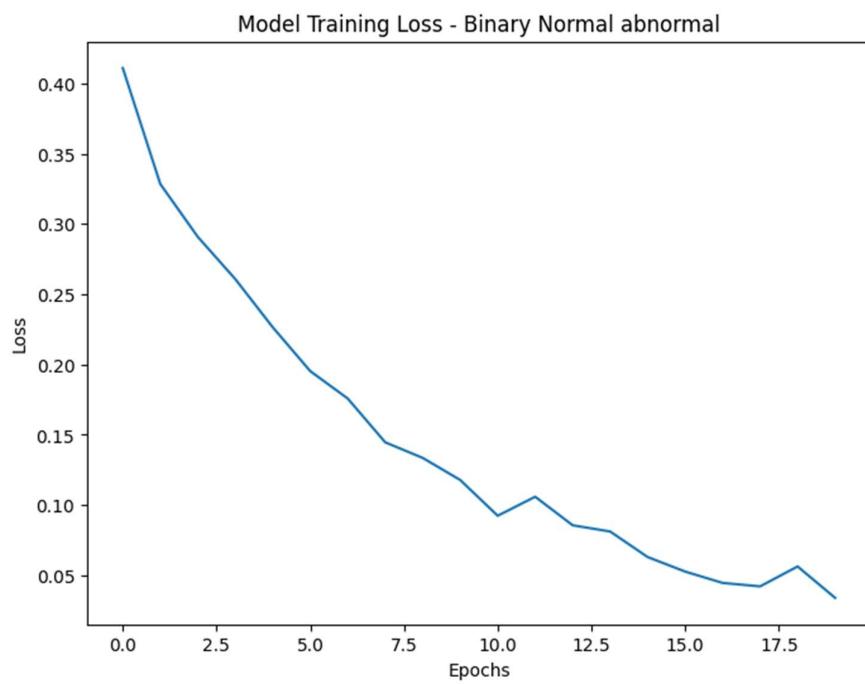
df_filtered = df_filtered.iloc[:, 1:]
```

شکل (۱-۲۶) حذف کلاس چهارم و تجمع کلاس A و O

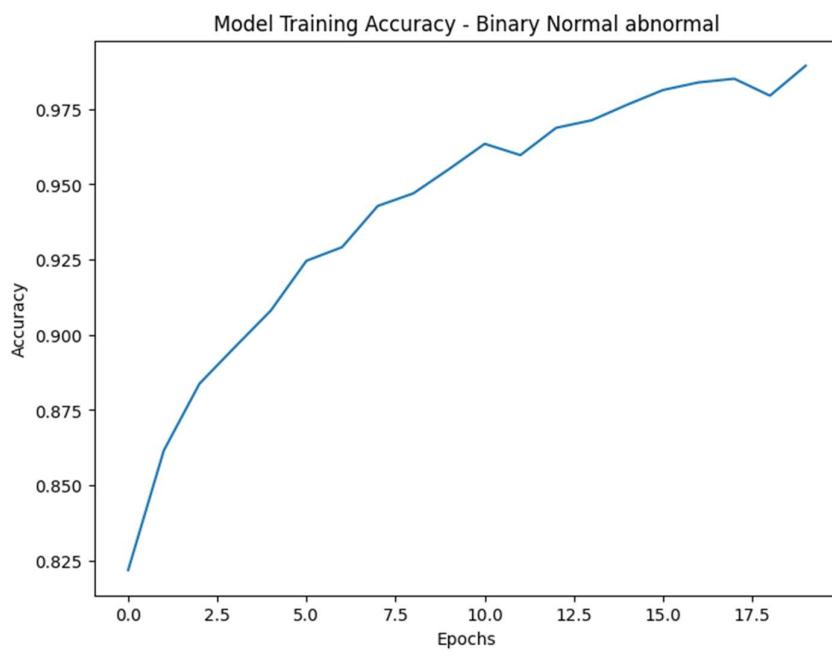
در این حالت همان کد های مرحله قبل است فقط تعداد کلاس ها و خروجی لایه آخر به ۲ تغییر میکند در

لایه آخر خروجی برای تابع فعال ساز میتوان از softmax و یا sigmoid استفاده کرد ( sigmoid توصیه میشود

(



شکل (۱-۲۷) منحنی loss در حالت طبقه باينری normal و abnormal



شکل (۱-۲۸) منحنی دقت val در طبقه بند normal و abnormal

```

Precision: 0.9292
Recall: 0.8221
F1-Score: 0.8723
Confusion Matrix:
[[583 25]
 [ 71 328]]

```

شکل (۱-۲۹) مقادیر recall و precision و F1 و ماتریس آشونگی روی داده های تست در حالت طبقه بندی باینری

## ۱-۶-۵- نتیجه گیری

باید این سه سناریو را با هم مقایسه کنیم:

**داده های غیر عادی با کلاس های نامتعادل:**

دقت: ۷۳٪ (اعتبار سنجی) و ۷۰٪ (تست)

کلاس ۴ دارای دقت ۰٪ است، در حالی که کلاس N دارای دقت ۸۴٪ است.

در این سناریو، داده های غیر عادی و کلاس های نامتعادل منجر به دقت کلی نسبتاً پایین تر شد. این مدل به

دلیل نمایش محدود آن در مجموعه داده، در طبقه بندی صحیح کلاس ۴ تلاش کرد. عدم تعادل باعث شد که

مدل به سمت کلاس اکثریت (N) سوگیری کند و در نتیجه دقت بالاتری برای آن کلاس ایجاد شود.

**داده های نرمال شده با کلاس های نامتعادل:**

دقت: ۹۸٪ (اعتبار) و ۸۴٪ (تست)

کلاس ۴ دارای دقت ۳۴٪ است، در حالی که کلاس های دیگر (N و غیر طبیعی) دارای دقت بین ۸۰-۸۵٪ هستند.

عادی سازی داده ها عملکرد طبقه بندی را به طور قابل توجهی بهبود بخشید. با مقیاس گذاری ویژگی ها،

مدل توانست از همه کلاس ها به طور مؤثر تری یاد بگیرد و پیش بینی های دقیق تری انجام دهد. با این حال، عدم

تعادل کلاس همچنان بر توزیع دقت تأثیر می گذارد، به طوری که کلاس ۴ در مقایسه با کلاس های دیگر دقت

کمتری دارد.

### داده های نرمال شده با کلاس های متعادل (N در مقابل غیر عادی):

دقت: 92٪ (اعتبار) و 98٪ (تست)

در این سناریو، با ترکیب کلاس های O و A در یک کلاس «غیر طبیعی» و حذف کلاس  $\sim$ ، داده ها بین کلاس های عادی (N) و غیر عادی متعادل شدند. این عملکرد کلی طبقه بندی را حتی بیشتر بهبود بخشد. با یک مجموعه داده متعادل، مدل می تواند هر دو کلاس را به طور موثر یاد بگیرد و طبقه بندی کند و در نتیجه دقت بالاتری برای هر دو کلاس ایجاد کند.

به طور کلی، عادی سازی داده ها با فعال کردن نمایش و یادگیری بهتر ویژگی به بهبود عملکرد طبقه بندی کمک کرد. علاوه بر این، متعادل کردن مجموعه داده با ترکیب کلاس ها و حذف کمترین کلاس، منجر به بهبود بیشتر در دقت شد. یک مجموعه داده متعادل تضمین می کند که مدل می تواند از همه کلاس ها یاد بگیرد و پیش بینی های دقیقی انجام دهد، و از سوگیری نسبت به هر کلاس خاصی اجتناب کند.

## ۱-۷-۷- سوال ۷ : پیاده سازی Madeline

### ۱-۷-۱- تولید دیتا ها

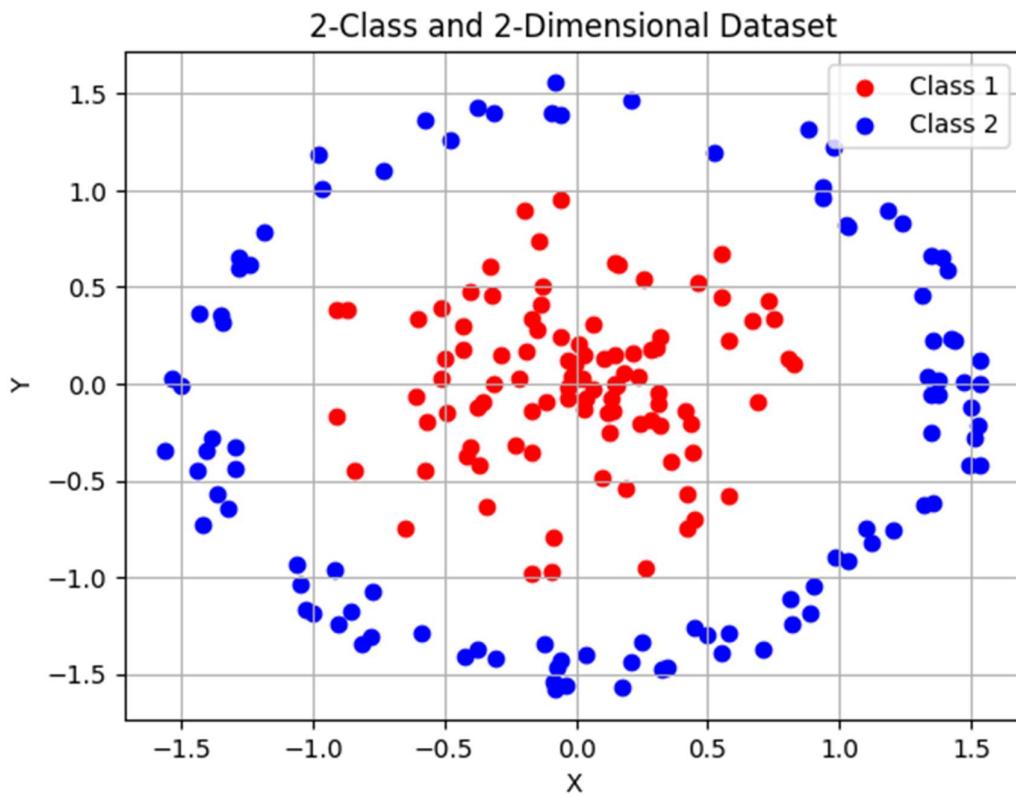
ابتدا به کمک کد زیر داده ها را تولید کرده و آن را رسم میکنیم.

```
# class 1
radius_1 = np.random.uniform(low=0, high=1, size=100)
angle_1 = np.random.uniform(low=0, high=2*np.pi, size=100)
x_1 = radius_1 * np.cos(angle_1)
y_1 = radius_1 * np.sin(angle_1)

# Class 2
radius_2 = np.random.uniform(low=1.3, high=1.6, size=100)
angle_2 = np.random.uniform(low=0, high=2*np.pi, size=100)
x_2 = radius_2 * np.cos(angle_2)
y_2 = radius_2 * np.sin(angle_2)

# Plot
plt.scatter(x_1, y_1, c='red', label='Class 1')
plt.scatter(x_2, y_2, c='blue', label='Class 2')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2-Class and 2-Dimensional Dataset')
plt.legend()
plt.grid(True)
plt.show()
```

شکل (۱-۳۰) کد پایتون تولید داده های سوال ۷ و پلات آن



شکل (۱-۳۱) داده های تولید شده برای سوال ۷

در ادامه نقاط x و y هر داده و کلاس آن ها را concat میکنیم.

```
# Create the target labels for the classes
labels_1 = np.zeros_like(x_1)
labels_2 = np.ones_like(x_2)

# Combine the data and labels
data = np.concatenate([np.column_stack((x_1, y_1, labels_1)), np.column_stack((x_2, y_2, labels_2))], axis=0)
np.random.shuffle(data) # Shuffling the data
```

data

```
array([[ 4.97270864e-01, -1.29654073e+00,  1.00000000e+00],
       [-9.13484820e-01,  3.78766560e-01,  0.00000000e+00],
       [-1.02428066e+00, -1.16989728e+00,  1.00000000e+00],
       [ 5.24031547e-01,  1.19831820e+00,  1.00000000e+00],
```

شکل (۱-۳۲) کانکت داده های سوال ۷ و نمونه ای data ساخته شده

## 1-7-2- پیاده سازی کلاس Madeline

### مقدمه

همچنین به عنوان "Perceptron" با خطوط تصمیمی تطبیقی چندگانه "شناخته می شود، یک مدل شبکه عصبی ساده شده است که می تواند برای وظایف طبقه بندی باینری استفاده شود. نام آن برگرفته از مخترعش، برنارد ویدرو، است که آن را در سال 1960 معرفی کرد.

شبکه Madeline از دو لایه تشکیل شده است: یک لایه پنهان و یک لایه خروجی. لایه پنهان حاوی چندین خط تصمیمی تطبیقی است که به عنوان "نرون ها" یا "واحدها" نیز شناخته می شوند، که مجموعاً جداسازی خطی داده های ورودی را انجام می دهند. لایه خروجی خروجی های واحدهای لایه پنهان را ترکیب می کند تا تصمیم نهایی طبقه بندی را بگیرد.

معماری شبکه Madeline با اجزای زیر مشخص می شود:

- لایه ورودی: لایه ورودی ویژگی های نمونه داده ها را نشان می دهد. هر ویژگی با یک وزن مرتبط است و ورودی ها در وزن های مربوط به آن ها ضرب می شوند.
- لایه پنهان: لایه پنهان وظیفه جداسازی خطی داده های ورودی را بر عهده دارد. از چندین واحد تشکیل شده است که هر کدام با وزن و تعصب خود مرتبط هستند. هر واحد یک مجموع وزنی از ورودی ها را محاسبه می کند و یک تابع فعال سازی را برای تعیین خروجی خود اعمال می کند. تابع فعال سازی معمولاً یک تابع آستانه است که بر اساس اینکه مجموع وزنی از آستانه از پیش تعریف شده فراتر رود، ۱- یا ۱ را اختصاص می دهد.
- لایه خروجی: لایه خروجی خروجی های واحدهای لایه پنهان را ترکیب می کند تا تصمیم نهایی طبقه بندی را بگیرد. در مورد طبقه بندی باینری، لایه خروجی اغلب از یک رای اکثریت ساده یا

یک تابع آستانه برای اختصاص بر چسب کلاس استفاده می کند.

شبکه Madeline از طریق فرآیندی به نام "یادگیری مبتنی بر خطای" یا "یادگیری نظارت شده" یاد می گیرد. وزن و بایاس واحدها در لایه پنهان را بر اساس خطای بین خروجی پیش بینی شده و مقدار هدف واقعی تنظیم می کند. الگوریتم یادگیری به طور مکرر وزن‌ها و سوگیری‌ها را به روزرسانی می کند تا خطای طبقه‌بندی را تا زمانی که یک معیار توقف برآورده شود، مانند رسیدن به یک آستانه از پیش تعریف شده یا هم گرایی، به حداقل برساند. یکی از مزایای شبکه Madeline سادگی و قابلیت تفسیر آن است. خطوط تصمیمی که توسط واحدهای لایه پنهان آموخته می شود را می توان تجسم کرد و بینش هایی را در مورد نحوه جداسازی شبکه داده های ورودی ارائه کرد. با این حال، شبکه Madeline دارای محدودیت هایی در مدیریت وظایف طبقه‌بندی پیچیده است که به مرزهای تصمیم غیرخطی نیاز دارند. در درجه اول برای مجموعه داده های قابل جداسازی خطی موثر است. است.

**پیاده سازی :**

یک کلاس Madeline مینویسیم.

```
class MadaLine():
    def __init__(self, features_dimension = 2, hidden_units = 2, learning_rate = 0.01, threshold = 0.05, stop_threshold = 1e-10):
```

شکل (1-۳۳) بخش 1 کد Madeline

بخش 1 کلاس MadaLine را با یک روش اولیه تعریف می کند. مقداردهی اولیه مقادیر پیش‌فرض پارامترهایی مانند تعداد ویژگی‌ها، تعداد واحدهای پنهان، نرخ یادگیری آستانه معیارهای توقف و آستانه توقف را تعیین می کند.

```

self.hidden_units = hidden_units
self.features_dimension = features_dimension

self.weights_1 = np.random.randn(hidden_units, features_dimension) * 0.001
self.bias_1 = np.random.randn(hidden_units, 1) * 0.001
self.weights_2 = np.array ([1 / hidden_units] * hidden_units).T
self.bias_2 = (hidden_units - 1) / hidden_units
self.learning_rate = learning_rate
self.threshold = threshold
self.stop_threshold = stop_threshold
self.loss_function = []
self.cost_function = []

```

شکل (۱-۳۴) بخش ۲ کد Madeline

بخش ۲ متغیرهای نمونه کلاس MadaLine را مقداردهی اولیه می کنند. تعداد واحدهای پنهان و ابعاد ویژگی ها را تنظیم می کند، وزن ها و بایاس ها را برای لایه پنهان و لایه خروجی به طور تصادفی مقداردهی اولیه می کند، نرخ یادگیری، آستانه، آستانه توقف را تنظیم می کند و لیست های خالی را برای تابع ضرر و هزینه مقداردهی اولیه می کند.

```

def activation_function(self, g):
    return ((g >= 0) - 0.5) * 2

```

شکل (۱-۳۵) بخش ۳ کد Madeline

این روش تابع فعال سازی مورد استفاده در شبکه را تعریف می کند. اگر ورودی  $g$  بزرگتر یا مساوی ۰ باشد ۱ و در غیر این صورت -۱ را برمی گرداند.

```

def loss_calculation(self, target, net):
    return 0.5 * np.power (target - net, 2)

```

شکل (۱-۳۶) بخش ۴ کد Madeline

بخش 4 مقدار loss بین هدف و خروجی شبکه را محاسبه می کند. اختلاف مجدد بین هدف و خروجی

شبکه را در 0.5 ضرب می کند.

```
def cost_calculation(self, loss):  
    return np.mean(loss)
```

شکل (1-۳۷) بخش 5 کد Madeline

این بخش تابع هزینه را با گرفتن میانگین مقادیر زیان محاسبه می کند.

```
def forward_propagation(self, Xi):  
  
    Xi = Xi.reshape(Xi.shape[0], 1)  
    Z_in = np.dot(self.weights_1, Xi) + self.bias_1  
    Z = self.activation_function(Z_in).reshape(Z_in.shape[0], 1)  
    Y_in = np.dot(self.weights_2, Z) + self.bias_2  
    y = self.activation_function(Y_in)  
  
    return Z_in, Y_in, y
```

شکل (1-۳۸) بخش 6 کد Madeline

این بخش انتشار رو به جلو را از طریق شبکه انجام می دهد. با توجه به ورودی  $X_i$ ، ورودی را تغییر شکل می

دهد، ورودی لایه پنهان ( $Z_{in}$ ) را محاسبه می کند، تابع فعال سازی را برای به دست آوردن خروجی لایه پنهان

( $Z$ ) اعمال می کند، ورودی لایه خروجی را محاسبه می کند ( $Y_{in}$ )، تابع فعال سازی را اعمال می کند.

```

def backward_propagation(self, xi, target, z_in, y):
    if y != target:
        z_in = z_in.flatten()
        if target == 1:
            close_to_zero_index = np.argmin(abs(z_in))

            dW = xi * (1 - z_in[close_to_zero_index])
            db = 1 - z_in[close_to_zero_index]
            self.weights_1[close_to_zero_index, :] += self.learning_rate * dW
            self.bias_1[close_to_zero_index] += self.learning_rate * db

        elif target == -1:
            for index in range(self.hidden_units):
                if z_in[index] > 0:
                    dW = xi * (-1 - z_in[index])
                    db = -1 - z_in[index]

                    self.weights_1[index, :] += self.learning_rate * dW
                    self.bias_1[index] += self.learning_rate * db
    return True
return False

```

شکل (1-۳۹) بخش 7 کد Madeline

این روش وظیفه به روزرسانی وزن‌ها و بایاس‌های شبکه Madeline را در حین برگشت به عقب بر عهده

دارد.

- ورودی  $X_i$ ، مقدار هدف، مجموع وزن لایه پنهان  $Z_{in}$  و خروجی پیش‌بینی شده  $y$  را به عنوان

ورودی می‌گیرد.

- اگر خروجی پیش‌بینی شده  $y$  با مقدار هدف مطابقت نداشته باشد، به روزرسانی‌ها را برای وزن‌ها و

بایاس‌ها در لایه پنهان بر اساس مقدار هدف محاسبه می‌کند.

- اگر مقدار هدف 1 باشد، شاخص واحد را در لایه پنهان که نزدیکترین به صفر است پیدا می‌کند

و وزن و بایاس آن را بر این اساس به روز می‌کند.

- اگر مقدار هدف -1 باشد، روی تمام واحدهای لایه پنهان تکرار می‌شود و وزن‌ها و بایاس‌های

واحدهایی را که دارای مجموع وزنی مثبت هستند، به روزرسانی می‌کند.

- در نهایت، در صورت بهروزرسانی وزن، True و در غیر این صورت False را برمی‌گرداند.

```
def train(self, X, Y, n_iterations = 1000):

    from sklearn.utils import shuffle

    X, Y = shuffle(X, Y, random_state = 69)

    for _ in tqdm(range(n_iterations)):
        self.loss_function = []
        updating_changes = []
        for x, target in zip(X, Y):
            Z_in, Y_in, y = self.forward_propagation(x)
            updating_changes.append(self.backward_propagation(x, target, Z_in, y))
            self.loss_function.append(self.loss_calculation(target, y))

        self.cost_function.append(self.cost_calculation(self.loss_function))
        if not any(updating_changes):
            print ("No more weight updating ...")
            return
        elif self.threshold > self.cost_calculation(self.loss_function):
            print ("Cost reached the minimum threshold ...")
            return
        print ("Maximum Number of Iterations has been completed ...")
```

شکل (۱-۴۰) بخش 8 کد Madeline

این بخش وظیفه آموزش شبکه Madeline با استفاده از داده‌های ورودی X و برچسب‌های Y را بر عهده

دارد.

- داده‌های ورودی و برچسب‌های هدف را با استفاده از تابع `shuffle` از ماثول `sklearn.utils` به هم

می‌زنند.

- این آموزش را برای تعداد مشخصی از تکرارها (n\_iterations) انجام می‌دهد.

- در هر تکرار، روی هر نقطه داده x و هدف برچسب هدف مربوطه با استفاده از تابع `zip` تکرار می

شود.

- برای هر نقطه داده، انتشار رو به جلو را انجام می‌دهد تا مجموع وزنی لایه پنهان `Z_in`، مجموع

وزنی لایه خروجی  $Y_{in}$ ، و خروجی پیش بینی شده  $y$  را به دست آورد.

- زیان محاسبه شده با استفاده از برچسب هدف و خروجی پیش بینی شده را به لیست loss\_function اضافه می کند.

- همچنین هزینه محاسبه شده با استفاده از loss\_function را به لیست cost\_function اضافه می کند.

- اگر هیچ بهروزرسانی وزنی در طول تکرار انجام نشده باشد (یعنی هیچ تغییری در لیست

وجود نداشته باشد)، "No more weight updatedation" را چاپ می کند

و برمی گردد.

- اگر هزینه به حداقل مقدار آستانه تعیین شده توسط self.threshold برسد، "هزینه به حداقل آستانه

رسید" چاپ می شود و برمی گردد.

- پس از تکمیل تمام تکرارها، "حداکثر تعداد تکرارها تکمیل شده است" را چاپ می کند.

```
def evaluation(self, X, Y):  
    predicts = []  
    for x in X:  
        y = self.forward_propagation(x)[2]  
        predicts.append(y)  
    from sklearn.metrics import classification_report  
    print (classification_report(Y, predicts))  
  
def prediction(self, X):  
    Y = []  
    for x in X:  
        z_in, Y_in, y = self.forward_propagation(x)  
        Y.append (y)  
    return np.array (Y)
```

شکل (۱-۴۱) بخش ۹ کد Madeline

روش ارزیابی:

این روش شبکه Madeline را بر روی داده های ورودی ارائه شده  $X$  و برچسب های  $Y$  هدف ارزیابی می کند.

روی هر نقطه داده  $x$  در  $X$  تکرار می شود و از روش forward\_propagation برای به دست آوردن خروجی پیش بینی شده  $y$  استفاده می کند.

خروچی های پیش بینی شده به لیست پیش بینی ها اضافه می شوند.  
از تابع classification\_report از ماثول sklearn.metrics برای تولید گزارش طبقه بندی استفاده می کند که پیش بینی های خروجی های پیش بینی شده و برچسب های هدف واقعی  $Y$  را مقایسه می کند.

روش پیش بینی:

این بخش پیش بینی هایی را برای داده های ورودی داده شده  $X$  ایجاد می کند.  
روی هر نقطه داده  $x$  در  $X$  تکرار می شود و از روش forward\_propagation برای به دست آوردن خروجی پیش بینی شده  $y$  استفاده می کند.

خروچی های پیش بینی شده به لیست  $Y$  اضافه شده و به صورت یک آرایه numpy برگردانده می شوند.

```
# Visualization

def plot_loss(self):
    plt.plot(self.cost_function, c = 'red')
    plt.title("MadaLine Cost Function History")
    plt.xlabel("N. Epoch")
    plt.ylabel("Loss Function")
    plt.show()

def plot_scatter(self, X, Y):
    plt.scatter(X[0, :], X[1, :], c = Y, alpha = 0.7, edgecolor = 'k')

    xx, yy = np.meshgrid(np.arange(min(X[0, :]) - 2, max(X[0, :]), 0.01) + 1,
                         np.arange(min(X[1, :]) - 2, max(X[1, :]), 0.01) + 1)
    Z = self.prediction(np.c_[xx.reshape(-1), yy.reshape(-1)])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha = 0.4)
    plt.title("MadaLine Desicion Boundary")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

شکل (۱-۴۲) کد 10 بخش Madeline

در نهایت نیز از این بخش برای نمایش منحنی های آموزش و خروجی ها استفاده میشود.

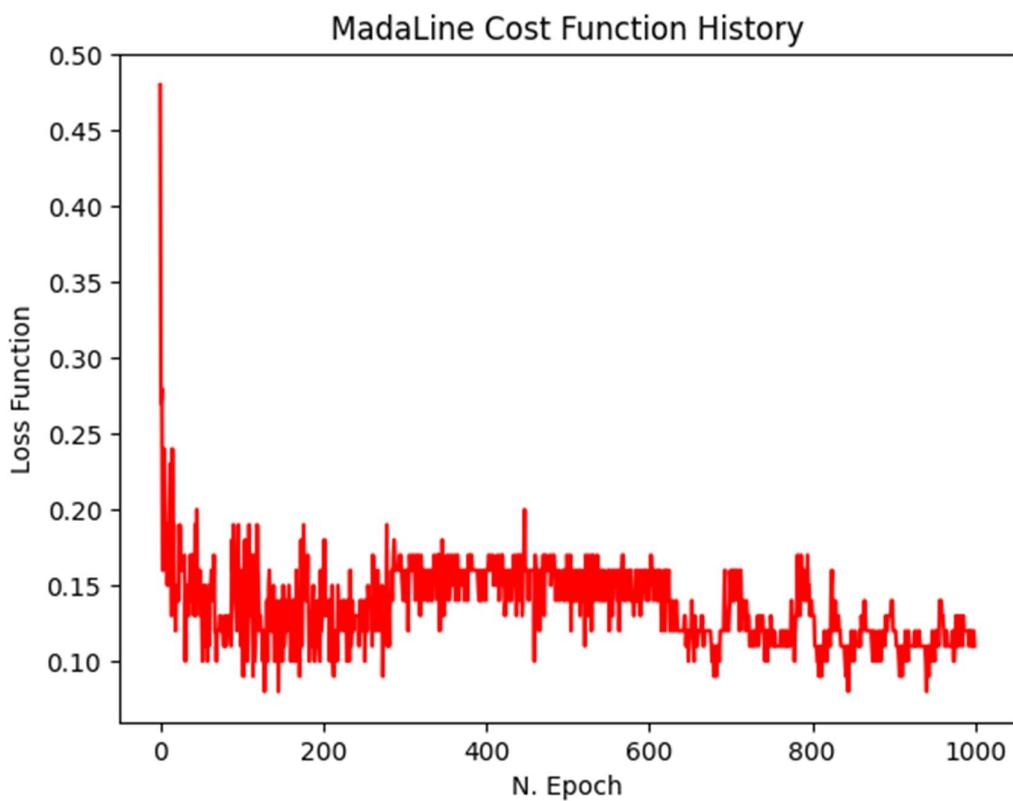
### 1-7-3- نتایج Madeline به ازای لایه های مخفی و نورون های لایه اخر مختلف

```
for N in [3, 6, 9]:
    print ("with %d units in hidden layer" % N)
    madaline = MadaLine(features_dimension = 2, hidden_units = N, learning_rate = 0.01, threshold = 0.01)
    # Split into features (X) and labels (Y)
    X = np.array(data[:, :2]).T
    Y = np.where(data[:, 2] == 1, 1, -1).reshape(1, X.shape[1])

    madaline.train(X.T, Y.T, n_iterations = 1000)
    madaline.plot_loss()
    madaline.evaluation(X.T, Y.T)
    madaline.plot_scatter(X, Y)
```

شکل (۱-۴۳) اعمال Madeline بر روی داده ها با تعداد نورون های متفاوت

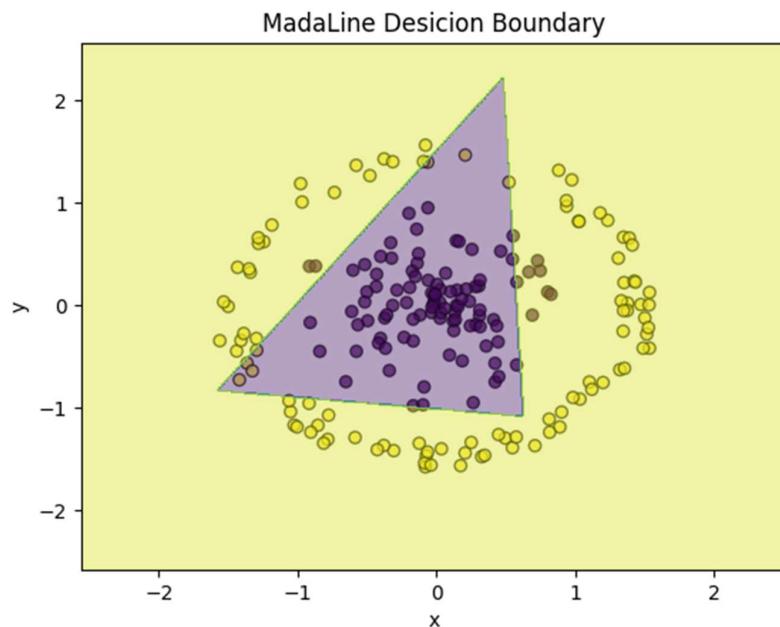
جدا سازی با ۳ خط و ۳ لایه پنهان :



شکل (۱-۴۴) منحنی loss در madeline برای جداسازی با ۳ خط

	precision	recall	f1-score	support
-1	0.93	0.90	0.91	100
1	0.90	0.93	0.92	100
accuracy			0.92	200
macro avg	0.92	0.92	0.91	200
weighted avg	0.92	0.92	0.91	200

شکل (۱-۴۵) مقادیر F1Score و recall و precision برای Madeline و جداسازی با ۳ خط

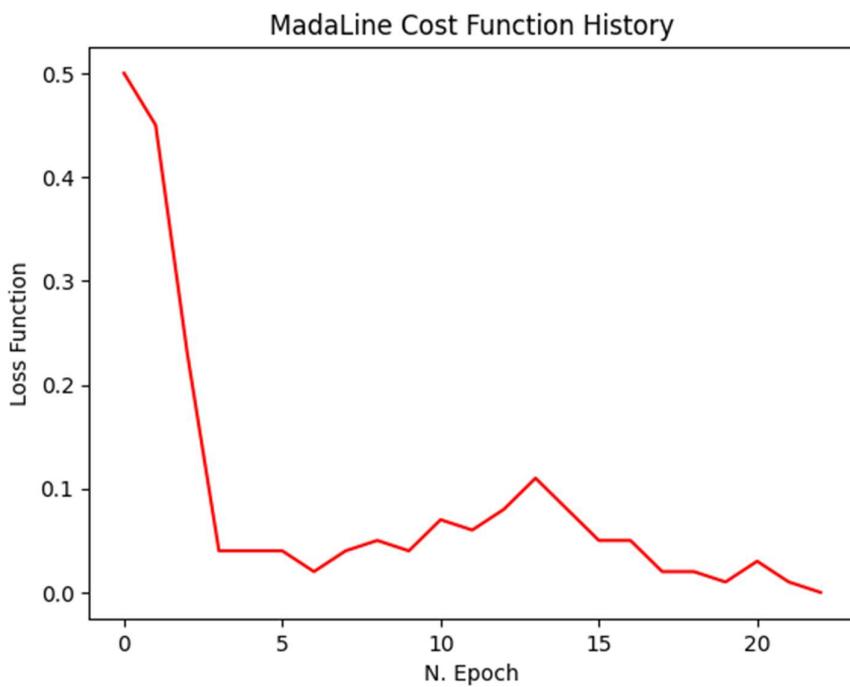


شکل (۱-۴۶) مرز های تصمیم در Madeline در حالت جداسازی با ۳ خط

مشاهده میشود که دقت در این حالت ۹۲ درصد میباشد.

**جداسازی با ۴ خط و ۶ لایه پنهان :**

در این حالت مقادیر به صورت زیر میباشد.



شکل (۱-۴۷) منحنی loss در Madeline در حالت جداسازی با ۴ خط و ۶ لایه پنهان

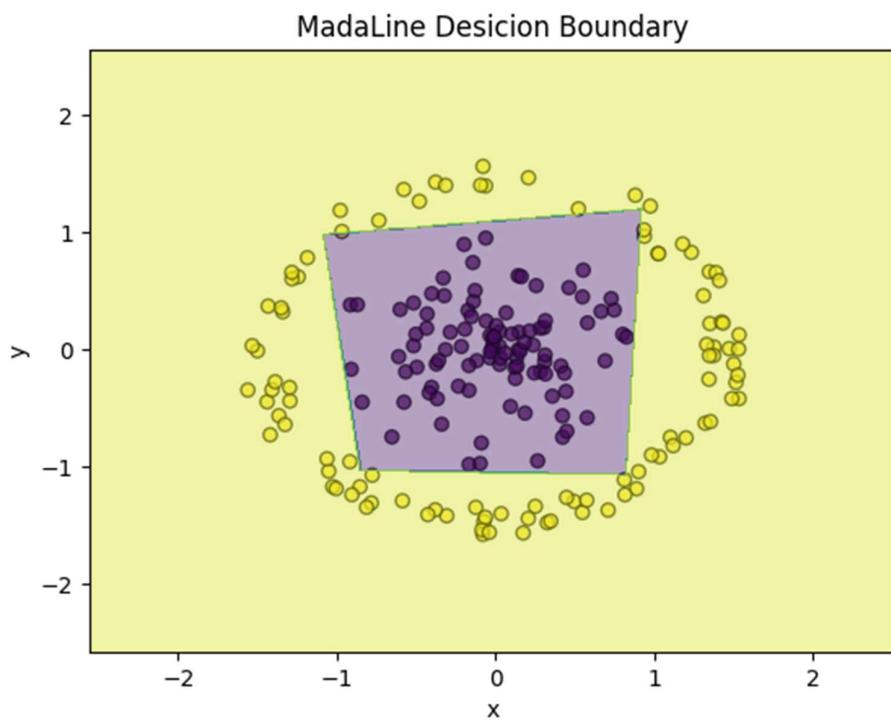
مشاهده میشود در این حالت خیلی زودتر از حالت قبل به شرط توقف رسیده ایم ( البته در حالت قبل در

یک loop گیر کرده بود و گرنه زودتر به نتایج مشابه رسیده بودیم)

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	100
1	1.00	1.00	1.00	100
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

شکل (۱-۴۸) مقادیر precision و recall و .. در حالت madeline و جداسازی با ۴ خط و ۶ لایه پنهان

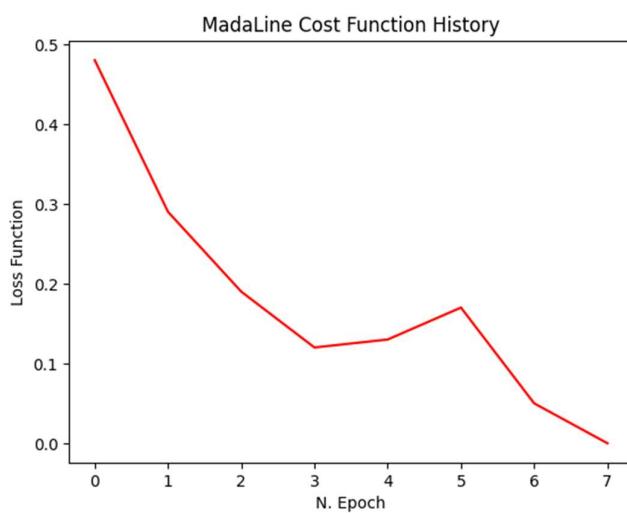
مشاهده میشود که با ۴ خط به راحتی میشود داده ها را با دقت 100 درصد جدا کرد



شکل (۱-۴۹) مرز تصمیم در حالت جداسازی با ۴ خط و ۶ لایه پنهان :

**جداسازی با ۶ خط و ۹ لایه پنهان :**

در حالت نتایج به شرح زیر است :



شکل (۱-۵۰) منحنی loss در حالت جداسازی با ۶ خط و ۹ لایه پنهان

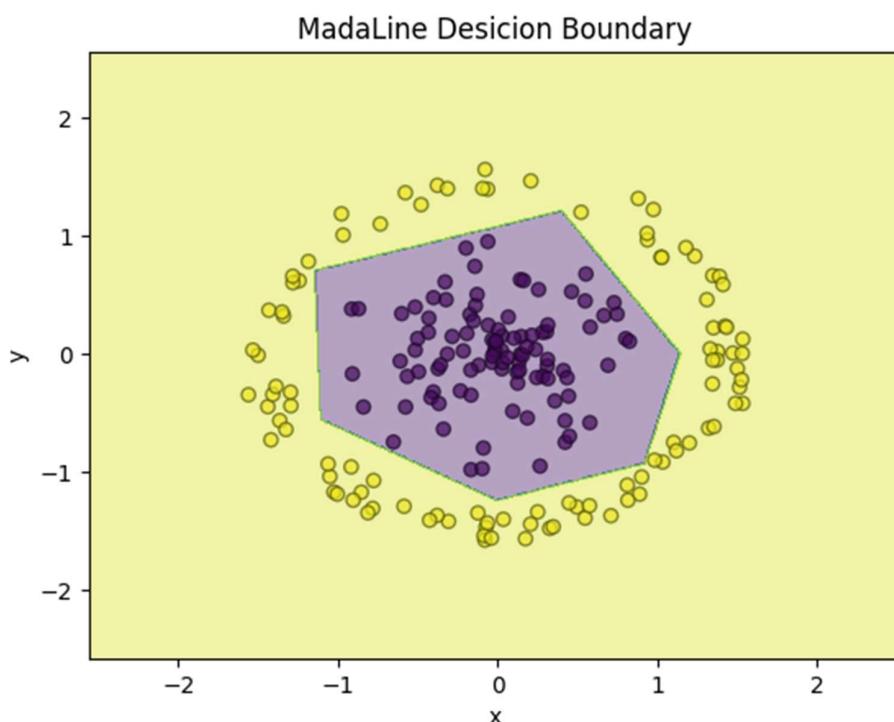
مشاهده میشود با زیاد کردن نورون لایه ها زودتر به شرط توقف میرسیم.

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	100
1	1.00	1.00	1.00	100
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

شکل (1-۵۱) مقادیر precision و recall و .. در حالت جداسازی با 6 خط و 9 لایه پنهان

مشاهده میشود همچون قبل دقت 100 درصد است و به راحتی میتوان با madeline طبقه بندی را انجام داد

البته با 4 خط هم به همین دقت میرسیدیم و زیاد کردن نورون ها صرفا پیچیدگی شبکه را بیشتر کرده است.



شکل (1-۵۲) جداسازی با 6 خط و 9 لایه پنهان ( Madeline )

## ۱-۸-سوال ۸: طبقه بندی تصاویر Cifar 10

### ۱-۸-۱-لود کردن دیتاست CIFAR و پیش پردازش

به کمک دستور زیر دیتاست مورد نظر را لود کرده و در ادامه ۵ نمونه از آن را به صورت تصادفی نمایش

میدهیم.

```
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Define class labels
class_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [=====] - 6s 0us/step
```

شکل (۱-۵۳) کد پایتون لود دیتاست

```
# Display five random images
num_images = 5
random_indices = np.random.randint(0, x_train.shape[0], num_images)

plt.figure(figsize=(10, 2))
for i, index in enumerate(random_indices):
    plt.subplot(1, num_images, i+1)
    plt.imshow(x_train[index])
    plt.title(class_labels[y_train[index][0]])
    plt.axis('off')

plt.show()
```

شکل (۱-۵۴) کد پایتون نمایش رندوم ۵ نمونه از داده های CIFAR10



شکل (1-۵۵) ۵ نمونه داده از داده های CIFAR10

در ادامه داده های X را بر 255 تقسیم کرده تا نرمال سازی را انجام دهیم و طبق گفته صورت مساله تقسیم

بندی را به صورت 40K برای آموزش و 10K برای ارزیابی و 10K برای تست در نظر میگیریم.

```

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# Split the data into training, validation, and test sets
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=10000, random_state=42)

```

شکل (1-۵۶) کد پایتون پیش‌پردازش داده های CIFAR و تقسیم داده ها

```

print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_val shape:", x_val.shape)
print("y_val shape:", y_val.shape)
print("x_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)

```

```

x_train shape: (40000, 32, 32, 3)
y_train shape: (40000, 10)
x_val shape: (10000, 32, 32, 3)
y_val shape: (10000, 10)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 10)

```

شکل (1-۵۷) داده های Shape تقسیم شده

## ۱-۸-۲- آموزش شبکه به optimizer های مختلف

برای آنکه کد مرتب تر باشد optimizer های مختلف را جداگانه تعریف کرده و در یک لوب آموزش را

بر روی شبکه داده شده انجام میدهیم.

```
# Define the optimizers
optimizers = {
    'Adam': Adam(),
    'SGD': SGD(),
    'RMSprop': RMSprop()
}
```

شکل (۱-۵۸) Optimizer های مطرح شده در صورت مساله

```
# Reset the model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

شکل (۱-۵۹) پیاده سازی شبکه داده شده با فریم وورک تنسورفلو

به کمک کد زیر شبکه داده شده را در یک لوب آموزش داده (دلیل آنکه خود مدل در لوب مجدداً تکرار

میشود reset مدل میباشد.)

به علاوه مقادیر خواسته شده برای گزارش در مساله را هم در لیست هایی به ازای نام optimizer جهت

بررسی و مقایسه ذخیره میشود.

```

history = {}
precision_scores = {}
recall_scores = {}
f1_scores = {}

for optimizer_name, optimizer in optimizers.items():
    # Reset the model
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

    print(f'Training model with {optimizer_name} optimizer')
    h = model.fit(x_train, y_train, batch_size=64, epochs=20, validation_data=(x_val, y_val), verbose=1)
    history[optimizer_name] = h.history

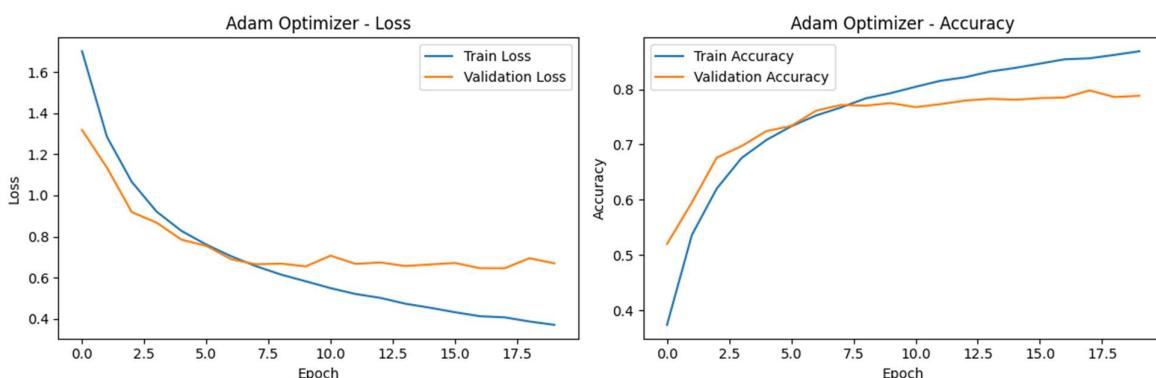
    # Evaluate the model
    y_pred = model.predict(x_test)
    y_pred = np.argmax(y_pred, axis=1)
    y_true = np.argmax(y_test, axis=1)

    precision_scores[optimizer_name] = precision_score(y_true, y_pred, average='macro')
    recall_scores[optimizer_name] = recall_score(y_true, y_pred, average='macro')
    f1_scores[optimizer_name] = f1_score(y_true, y_pred, average='macro')

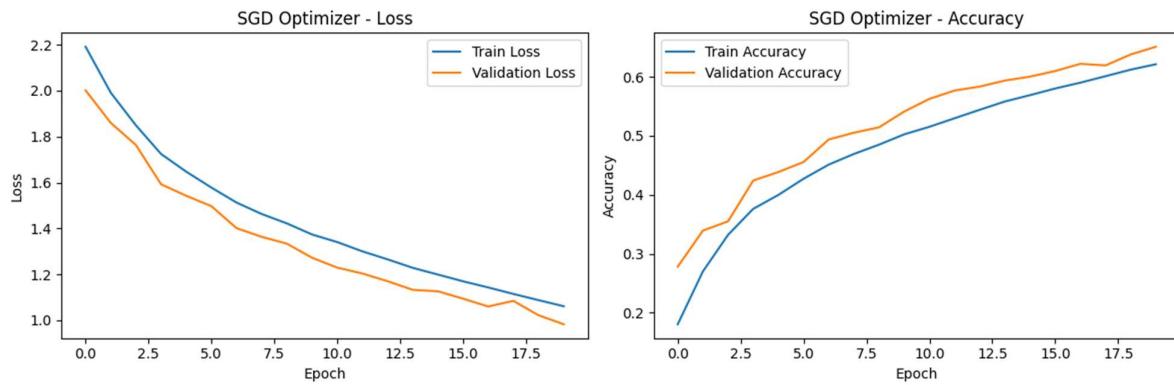
```

شکل (۱-۶۰) کد پایتون آموزش شبکه به ازای optimizer های مختلف

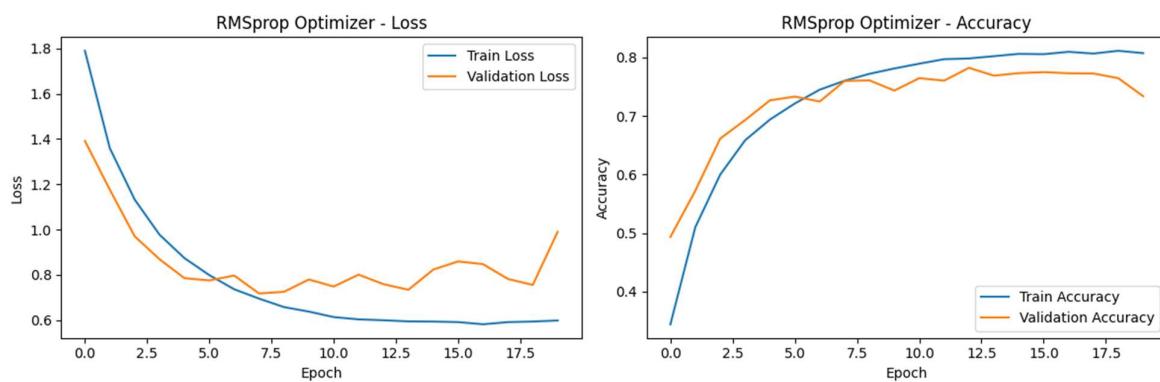
شکل های زیر منحنی های loss اترین و دقت آموزش و Val را به ازای optimizer های مختلف نشان میدهد.



شکل (۱-۶۱) منحنی Loss و دقت برای adam Opt



شکل (۱-۶۲) منحنی loss و دقت برای SGD Opt.



شکل (۱-۶۳) منحنی loss و دقت بر روی RMSprop opt

شکل زیر نیز مقادیر precision و recall و F1Score را بر روی داده های تست به ازای Optimizer های

مختلف نشان میدهد

```
Optimizer: Adam
Precision: 0.7863752434214992
Recall: 0.7865
F1 Score: 0.7853666543504559
```

```
Optimizer: SGD
Precision: 0.6501601751325873
Recall: 0.6547
F1 Score: 0.648690924950683
```

```
Optimizer: RMSprop
Precision: 0.7520145980048323
Recall: 0.7307
F1 Score: 0.7296098265142222
```

شکل (۱-۶۴) مقادیر precision و Recall و F1score به ازای optimizer های مختلف

با توجه به موارد فوق میتوان گفت بر روی val و ترین به Adam و RMSprop عملکرد بهتری داشتند که

قطعاً به این دلیل است که این ۲ تا optimizer SGD از الگوریتم schaduling استفاده میکنند و مرتب

lr را کم میکنند ولی SGD این قابلیت را ندارد.

بر روی داده های تست نیز مشخص است این ۲ تا oprimizer عملکرد بهتری داشتن و precision بالاتری

دارند و نسبت به هم نیز عملکرد تقریباً مشابه ای داشته ولی با توجه به نتایج میتوان گفت در این مساله adam از

باقي optimizer ها بهتر است.

## ۱-۹-سوال ۹ : Decision Tree بر روی داده بیماران دیابتی

### ۱-۹-۱- لود کردن دیتاست و پیش پردازش

به کمک کتابخانه پانداز دیتاست را لود میکنیم.

```
data = pd.read_csv('Diabetes.csv')
data.head()
```

	pregnant	glucose	BP	skin	insulin	BMI	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

شکل (۱-۶۵) لود کردن دیتاست دیابت

با توجه به ستون های بالا تقسیم داده ها را به صورت زیر انجام میدهیم :

```
x = data.drop('label', axis=1)
y = data['label']
X_train, X_val, y_train, y_val = train_test_split(x, y, test_size=0.3, random_state=42)
```

شکل (۱-۶۶) کد پایتون پیش پردازش داده های دیابت و تبدیل به فضای ورودی و خروجی

## ۱-۹-۲- طراحی scratch از decision Tree

با توجه به مساله که از کتابخانه آماده فقط برای ویژوال کردن میشود استفاده کرد مساله را از scratch پیاده

میکنیم.

در ادامه بخش های مختلف کد و توضیحات آن امده است.

```

def gini_index(groups, classes):
    n_instances = float(sum(len(group) for group in groups))
    gini = 0.0
    for group in groups:
        size = float(len(group))
        if size == 0:
            continue
        score = 0.0
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        gini += (1.0 - score) * (size / n_instances)
    return gini

```

شکل (1-۶۷) بخش 1 کد decision Tree

پیاده سازی شاخص جینی: تابع `gini_index` شاخص جینی را برای یک تقسیم معین از داده ها محاسبه می کند. گروه ها را می گیرد که تقسیم داده ها را به دو گروه نشان می دهد، و کلاس هایی که حاوی مقادیر کلاس منحصر به فرد هستند. این تابع در هر گروه تکرار می شود، امتیاز جینی را برای آن گروه محاسبه می کند، و امتیاز جینی را برای همه گروه ها جمع می کند تا شاخص جینی را محاسبه کند. شاخص جینی معیاری برای سنجش ناچالصی در یک گره است.

```

def get_best_split(data):
    class_values = list(set(row[-1] for row in data))
    best_index, best_value, best_score, best_groups = float('inf'), float('inf'), float('inf'), None
    for index in range(len(data[0]) - 1):
        for row in data:
            groups = split_data(index, row[index], data)
            gini = gini_index(groups, class_values)
            if gini < best_score:
                best_index, best_value, best_score, best_groups = index, row[index], gini, groups
    return {'index': best_index, 'value': best_value, 'groups': best_groups}

```

شکل (1-۶۸) بخش 2 کد decision Tree

یافتن بهترین تقسیم: تابع `get_best_split` بهترین نقطه تقسیم را در داده ها بر اساس شاخص جینی پیدا می کند. بر روی هر ویژگی و هر مقدار در آن ویژگی تکرار می شود، داده ها را بر اساس مقدار ویژگی تقسیم می کند، شاخص جینی را برای تقسیم محاسبه می کند، و بهترین تقسیم را با پایین ترین شاخص جینی پیگیری می کند. یک

فرهنگ لغت حاوی شاخص، مقدار و گروه های حاصل از بهترین تقسیم را برمی گرداند.

```
def split_data(index, value, data):
    left, right = [], []
    for row in data:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right
```

شکل (1-۶۹) بخش 3 کد decision Tree

تقسیم داده ها: تابع `split_data` داده ها را بر اساس شاخص و مقدار مشخصه به دو گروه تقسیم می کند. در هر ردیف از داده ها تکرار می شود و اگر مقدار ویژگی کمتر از مقدار داده شده باشد، آن را به گروه چپ و در غیر این صورت به گروه سمت راست اضافه می کند. گروه های چپ و راست را برمی گرداند.

```
class DecisionTree:
    def __init__(self, max_depth=5):
        self.max_depth = max_depth
        self.tree = None
```

شکل (1-۷۰) بخش 4 کد decision Tree

کلاس `DecisionTree`: کلاس `DecisionTree` یک درخت تصمیم را نشان می دهد. دارای یک متد `__init__` است که حداقل عمق درخت و ویژگی درخت را مقداردهی می کند، که درخت تصمیم گیری آموخته شده را ذخیره می کند.

```

def fit(self, X, y, depth=0):
    data = np.column_stack((X, y))
    if len(set(y)) == 1:
        return {'label': y[0]}

    if depth >= self.max_depth:
        return {'label': self.get_majority_class(y)}

    if len(data) == 0:
        return {'label': self.get_majority_class(y)}

    if len(set(y)) == 1:
        return {'label': y[0]}

    split = get_best_split(data)
    left_data, right_data = split['groups']

    if len(left_data) == 0 or len(right_data) == 0:
        return {'label': self.get_majority_class(y)}

    left = self.fit(np.array(left_data)[:, :-1], np.array(left_data)[:, -1], depth + 1)
    right = self.fit(np.array(right_data)[:, :-1], np.array(right_data)[:, -1], depth + 1)

    self.tree = {'index': split['index'], 'value': split['value'], 'left': left, 'right': right}
    return self.tree

```

شکل (1-۷۱) بخش ۵ کد decision Tree

روش برازش برای ساخت درخت تصمیم به صورت بازگشتی استفاده می شود. ویژگی های ورودی  $X$ ، برچسب های هدف  $y$  و پارامتر عمق را می گیرد. ابتدا ویژگی ها و برچسب ها را در یک آرایه داده واحد ترکیب می کند. سپس شرایط مختلف پایان را بررسی می کند، مانند اینکه آیا همه برچسب ها یکسان هستند یا به حد اکثر عمق رسیده اند. اگر شرایط پایان برآورده شود، یک گره برگ با برچسب کلاس اکثریت بر می گردد. در غیر این صورت، بهترین تقسیم را با استفاده از تابع `get_best_split` پیدا می کند، داده ها را تقسیم می کند و به صورت بازگشتی متد `fit` را در زیر مجموعه های چپ و راست داده ها فراخوانی می کند. ویژگی درخت را با اطلاعات تقسیم به روز می کند و درخت را بر می گرداند.

```

def get_majority_class(self, labels):
    unique_classes, class_counts = np.unique(labels, return_counts=True)
    majority_class = unique_classes[np.argmax(class_counts)]
    return majority_class

```

شکل (1-۷۲) بخش ۶ کد decision Tree

برچسب ها بر می گرداند. کلاس های منحصر به فرد و تعداد آنها را محاسبه می کند و برچسب کلاس را با بیشترین تعداد بر می گرداند.

```
def predict(self, X):
    predictions = []
    for sample in X:
        predictions.append(self.predict_single(sample, self.tree))
    return predictions
```

شکل (1-۷۳) بخش 7 کد decision Tree

پیش بینی: روش پیش بینی برچسب های کلاس را برای مجموعه ای از ویژگی های ورودی  $X$  پیش بینی می کند. روی هر نمونه در  $X$  تکرار می شود و متد `predict_single` را فراخوانی می کند تا درخت تصمیم را طی کند و پیش بینی کند.

```
def predict_single(self, sample, node):
    if 'label' in node:
        return node['label']

    if sample[node['index']] < node['value']:
        return self.predict_single(sample, node['left'])
    else:
        return self.predict_single(sample, node['right'])
```

شکل (1-۷۴) بخش 8 کد decision Tree

روش `predict_single` به صورت بازگشتی درخت تصمیم را که از یک گره معین شروع می شود طی می کند و بر اساس مقادیر ویژگی نمونه ورودی پیش بینی می کند. اگر گره فعلی یک گره برگ

باشد، برچسب کلاس را برمی گرداند. در غیر این صورت، مقدار ویژگی نمونه را با مقدار تقسیم ذخیره شده در

گره مقایسه می کند و بر این اساس به گره فرزند چپ یا راست می رود تا زمانی که به یک گره برگ برسد.

پس از پیاده سازی دیتا ها را اعکال کرده و حد اکثر عمق را برابر 5 در نظر میگیریم.

```
decision_tree = DecisionTree(max_depth=5)
decision_tree.fit(X_train.values, y_train.values)

train_predictions = decision_tree.predict(X_train.values)
val_predictions = decision_tree.predict(X_val.values)

train_accuracy = accuracy_score(y_train, train_predictions)
val_accuracy = accuracy_score(y_val, val_predictions)

print("Train Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)
```

Train Accuracy: 0.8305400372439479  
Validation Accuracy: 0.7532467532467533

شکل (۱-۷۵) کد پایتون اعمال داده ها به decision Tree که از scratch نوشته شده و مقادیر دقت

با توجه به تصویر فوق دقت شبکه روی داده آموزشی و val به ترتیب برابر 83 و 75 میباشد.

برای ویژوال کردن نیز فانکشن زیر را نوشته تا از scratch نوشتن را تکمیل کنیم (در بخش بعدی گرافیکی

تر و بهتر رسم شده است.

```

def build_graph(node, dot, parent=None, edge_label=None):
    if "label" in node:
        dot.node(str(id(node)), str(node["label"]), shape="box")
        if parent is not None:
            dot.edge(str(id(parent)), str(id(node)), label=edge_label)
    else:
        dot.node(str(id(node)), "Feature " + str(node["index"]))
        if parent is not None:
            dot.edge(str(id(parent)), str(id(node)), label=edge_label)

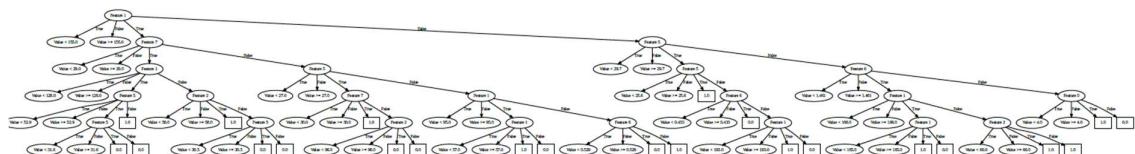
    dot.node(str(id(node)) + "L", "Value < " + str(node["value"]))
    dot.node(str(id(node)) + "R", "Value >= " + str(node["value"]))
    dot.edge(str(id(node)), str(id(node)) + "L", label="True")
    dot.edge(str(id(node)), str(id(node)) + "R", label="False")

    build_graph(node["left"], dot, parent=node, edge_label="True")
    build_graph(node["right"], dot, parent=node, edge_label="False")

dot = graphviz.Digraph()
build_graph(decision_tree.tree, dot)
dot.render("decision_tree_scratch", format="pdf", cleanup=True)

```

شکل (۷۶-۱) تابع رسم درخت تصمیم از scratch



شکل (۷۷-۱) درخت تصمیم رسم شده از scratch

لازم به ذکر است فایل pdf با کیفیت درخت تصمیم در گزارش آمده است.

### 3-9-1- درخت تصمیم با استفاده از کتابخانه آماده

```

decision_tree = DecisionTreeClassifier(max_depth=5)

decision_tree.fit(X_train, y_train)

train_predictions = decision_tree.predict(X_train)
val_predictions = decision_tree.predict(X_val)

train_accuracy = accuracy_score(y_train, train_predictions)
val_accuracy = accuracy_score(y_val, val_predictions)

print("Train Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)

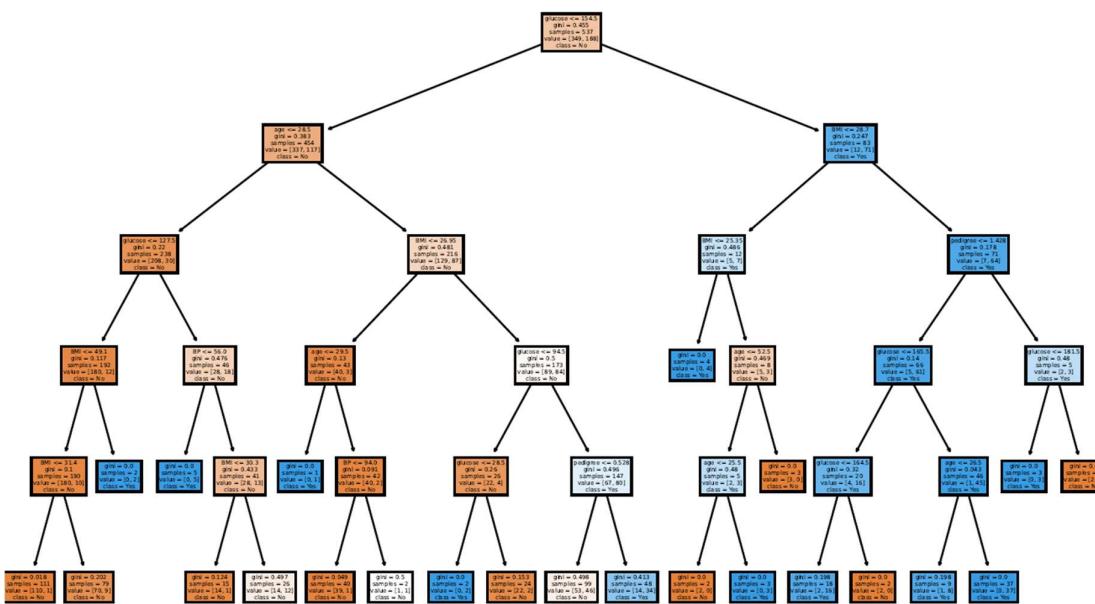
plt.figure(figsize=(10, 6))
plot_tree(decision_tree, feature_names=X_train.columns, class_names=['No', 'Yes'], filled=True)
plt.savefig('decision_tree_library.pdf')
plt.show()

```

Train Accuracy: 0.8324022346368715  
 Validation Accuracy: 0.7532467532467533

شکل (1-78) کد پایتون اعمال درخت تصمیم بر داده ها و رسم آن و مقادیر دقت ها

مشاهده میشود دقت ها همان دقت های قبلی میباشد.



شکل (1-79) درخت تصمیم با حداقل عمق 5

فایل با کیفیت تصویر فوق با فرمت PDF در فایل ارسالی آماده است.

## 2-1-9-4- درخت تصمیم با حداقل طول 2

کافیست در همان کدهای قبلی حداقل طول را به عنوان ارگمن روى 2 قرار بدهیم.

```
##### scratch :  
decision_tree = DecisionTree(max_depth=2)  
decision_tree.fit(X_train.values, y_train.values)  
train_predictions = decision_tree.predict(X_train.values)  
val_predictions = decision_tree.predict(X_val.values)  
train_accuracy = accuracy_score(y_train, train_predictions)  
val_accuracy = accuracy_score(y_val, val_predictions)  
  
print("Train Accuracy:", train_accuracy)  
print("Validation Accuracy:", val_accuracy)
```

```
Train Accuracy: 0.7597765363128491  
Validation Accuracy: 0.7186147186147186
```

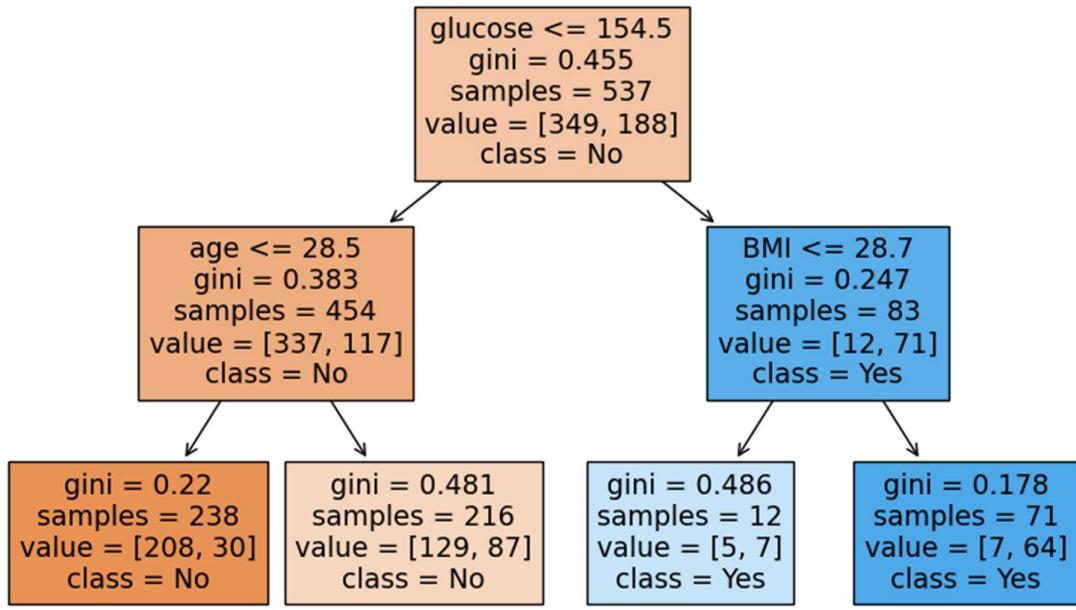
```
decision_tree = DecisionTreeClassifier(max_depth=2)  
decision_tree.fit(X_train, y_train)  
train_predictions = decision_tree.predict(X_train)  
val_predictions = decision_tree.predict(X_val)  
train_accuracy = accuracy_score(y_train, train_predictions)  
val_accuracy = accuracy_score(y_val, val_predictions)  
print("Train Accuracy:", train_accuracy)  
print("Validation Accuracy:", val_accuracy)  
plt.figure(figsize=(10, 6))  
plot_tree(decision_tree, feature_names=X_train.columns, class_names=['No', 'Yes'], filled=True)  
plt.savefig('decision_tree_library_depth2.pdf')  
plt.show()
```

```
Train Accuracy: 0.7597765363128491  
Validation Accuracy: 0.7186147186147186
```

شکل (1-۸۰) درخت تصمیم با حداقل طول 2 - روی کلاس آمده و کلاس نوشته شده از scratch

مشاهده میشود که در این حالت دقت روی ترین 76 درصد و روی val 72 درصد است.

شکل زیر نیز درخت تصمیم با حداقل طول برابر 2 را نشان میدهد.



شکل (۱-۸۱) درخت تصمیم با حد کثر طول برابر ۲

### ۱-۹-۵- مقایسه تفسیر پذیری درخت تصمیم با طول ۲ با حد اکثر طول های بالاتر

هنگام مقایسه درختان تصمیم با اعمق مختلف از نظر تفسیرپذیری، ملاحظات زیر را می توان در نظر گرفت:

**عمق ۲:** درخت تصمیم با حد اکثر عمق ۲ ساختاری ساده و کم عمق خواهد داشت. این فقط از چند گره

و شاخه تشکیل شده است که در ک و تفسیر آن را آسان می کند. درخت قوانین اساسی تصمیم گیری و روابط

بین تعداد کمی از ویژگی ها را به تصویر می کشد. با این حال، ممکن است الگوهای پیچیده یا تعاملات بین

ویژگی ها را ثبت نکند.

**عمق بالاتر:** افزایش حد اکثر عمق درخت تصمیم به آن اجازه می دهد تا عمیق تر رشد کند و الگوها و

تعاملات پیچیده تری را در داده ها ثبت کند. درخت گره ها و شاخه های بیشتری خواهد داشت که مرزهای

تصمیم گیری دقیق تری را نشان می دهد. این می تواند منجر به دقت بالاتر در داده های آموزشی شود، اما درخت پیچیده تر و تفسیر آن سخت تر می شود.

از نظر **تفسیرپذیری**، درخت تصمیم با عمق 2 به طور کلی قابل تفسیرتر از درخت عمیق تر در نظر گرفته می شود. درخت کم عمق یک نمای کلی در سطح بالا از مهم ترین ویژگی ها و آستانه تصمیم گیری آنها ارائه می دهد. می توان آن را به راحتی برای ذینفعان غیر فنی توضیح داد و در ک خوبی از عوامل کلیدی موثر بر فرآیند تصمیم گیری ارائه می دهد.

از سوی دیگر، درختان عمیق تر مستعد بیش از حد برآش هستند، زیرا می توانند نویز یا الگوهای نامربوط را در داده ها ثبت کنند. علاوه بر این، تفسیر قوانین تصمیم گیری پیچیده و تعاملات در یک درخت عمیق چالش برانگیزتر می شود و به در ک عمق تری از روابط بین ویژگی ها نیاز دارد.

بنابراین، اگر تفسیرپذیری در اولویت باشد، درخت تصمیم با عمق 2 ممکن است ارجح باشد. تعادلی بین سادگی و گرفتن قوانین تصمیم گیری ضروری ایجاد می کند. با این حال، ارزیابی تعادل بین تفسیرپذیری و دقت بسیار مهم است. گاهی اوقات، فدا کردن مقداری از تفسیرپذیری برای دقت بالاتر (به عنوان مثال، با عمق 5 یا بالاتر) ممکن است قابل قبول باشد، به خصوص زمانی که تمرکز در درجه اول بر عملکرد پیش بینی است.

## 1-9-6- روش Pre-pruning

هرس ( pre- pruning ) تکنیکی است که با کاهش پیچیدگی درخت و بهبود قابلیت تعمیم آن، برای جلوگیری از برازش بیش از حد در درختان تصمیم استفاده می شود. پیش هرس که به نام توقف زودهنگام یا هرس مبتنی بر محدودیت نیز شناخته می شود، یک رویکرد خاص برای هرس است که در آن درخت در طول فرآیند ساخت و ساز محدود می شود تا از پیچیده شدن بیش از حد آن جلوگیری شود.

هدف از هرس متوقف کردن زودتر رشد درخت با اعمال معیارهای توقف خاص بر اساس شرایط از پیش تعریف شده است. با محدود کردن رشد درخت، پیش هرس به جلوگیری از گرفتن نویز یا الگوهای نامربوط درخت در داده های آموزشی کمک می کند، که منجر به بهبود تعمیم و عملکرد بهتر در داده های دیده نشده می شود.

در اینجا چند تکنیک رایج پیش از هرس مورد استفاده در درختان تصمیم وجود دارد:

حداکثر عمق: یکی از ساده ترین تکنیک های پیش از هرس، تعیین حداکثر عمق برای درخت است. فرآیند ساخت درخت با رسیدن به حداکثر عمق متوقف می شود. این تضمین می کند که درخت فراتر از سطح معینی از پیچیدگی رشد نمی کند.

مشخص می کند. اگر تعداد نمونه ها در یک گره کمتر از این آستانه باشد، گره شکافته نمی شود و تبدیل به یک گره برگ می شود. این از پارسیشن بندی بیشتر گره ها با داده های ناکافی جلوگیری می کند که می تواند منجر به بیش از حد برازش شود.

مشابه حداقل تقسیم نمونه، این معیار حداقل تعداد نمونه مورد نیاز برای تشکیل یک گره برگ را مشخص می کند. اگر تعداد نمونه ها در یک گره کمتر از این آستانه باشد پس از تقسیم، تقسیم

انجام نمی شود و گرہ تبدیل به یک گرہ برگ می شود. این از ایجاد گرہ های برگ با نمونه های بسیار کم

جلوگیری می کند، که ممکن است منجر به برازش بیش از حد شود.

حداکثر ویژگی ها: در درخت های تصمیم با انتخاب ویژگی یا نمونه برداری تصادفی ویژگی، معیار حداکثر

ویژگی ها، تعداد ویژگی های در نظر گرفته شده برای هر تقسیم را محدود می کند. از استفاده درخت از همه

ویژگی های موجود جلوگیری می کند و روی یک زیر مجموعه تمرکز می کند، پیچیدگی درخت را کاهش

می دهد و به طور بالقوه تعمیم آن را بهبود می بخشد.

آستانه ناخالصی: از معیارهای ناخالصی، مانند ناخالصی جینی یا آنتروپی، می توان برای جلوگیری از شکافتن

گرہ ها زمانی که کاهش ناخالصی به زیر یک آستانه خاص می رسد، استفاده کرد. اگر کاهش ناخالصی به اندازه

کافی قابل توجه نباشد، گرہ شکافته نمی شود و تبدیل به گرہ برگ می شود. این به جلوگیری از شکافهای غیر

ضروری که کمک زیادی به بهبود خلوص کلاس ها نمی کند، کمک می کند.

برای کنترل رشد درخت تصمیم می توان از تکنیک های پیش هرس به صورت جداگانه یا ترکیبی استفاده

کرد. انتخاب پارامترهای پیش از هرس بستگی به مجموعه داده، پیچیدگی مشکل و مبادله بین پیچیدگی و عملکرد

مدل دارد.

با بکارگیری تکنیک های پیش هرس، درخت های تصمیم را می توان ساده سازی کرد، که منجر به مدل های

قابل تفسیر تر، کاهش بیش از حد برازش و بهبود عملکرد تعمیم در داده های دیده نشده می شود.