

به نام خدا



دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

فاطمه رشیدی شهری	نام عضو اول
۶۱۰۳۹۹۱۳۱	شماره دانشجویی
آراد وزیرپناه	نام عضو دوم
۶۱۰۳۹۹۱۸۲	شماره دانشجویی

فهرست

1.....	قوانين
1.....	پرسش ۱ - McCulloch Pitts
1.....	۱-۱. شبکه محاسبه مکمل
1.....	۱-۲. پیاده‌سازی تئوری شبکه
4.....	۱-۳. پیاده‌سازی کد شبکه
9.....	پرسش ۲ - حملات خصمانه در شبکه‌های عصبی
9.....	۲-۱. آشنایی با مجموعه دادگان
11.....	۲-۲. ایجاد و آموزش مدل
13.....	۲-۳. پیاده‌سازی حمله FGSM
16.....	۲-۴. پیاده‌سازی حمله PGD
21.....	پرسش ۳ - Madaline و Adaline
21.....	۳-۱. Adaline
21.....	۳-۲. Wine
21.....	۳-۳. نرمال کردن داده
22.....	۳-۴. نمودار پراکندگی داده‌ها
23.....	۴-۱. پیاده‌سازی شبکه Adaline
29.....	۴-۲. Madaline
29.....	الف) الگوریتم‌های MRI و MII
31.....	ب) آموزش مدل
50.....	پرسش ۴ - شبکه عصبی بهینه
50.....	۴-۱. رگرسن
50.....	الف) برازش بیش از حد (overfitting)
51.....	ب) راه حل رفع overfitting
51.....	پ) هایپرپارامتر
53.....	ت) پیاده سازیتابع سینوسی و رگرسن
59.....	ث) GridSearchCV

60.....	۲-۴ طبقه بندی
60.....	۱-۲-۴ افزایش داده‌ها
63.....	۲-۴ افزایش لایه‌ها

شکل‌ها

- 1 شکل 1. تصویر نمونه از ساختار شبکه مدل نظر
- 1 شکل 2. شبیه‌سازی عملیات OR منطقی با استفاده از نورون‌های M&P
- 2 شکل 3. شبیه‌سازی عملیات XOR منطقی با استفاده از نورون‌های M&P
- 3 شکل 4. شبکه محاسبه مکمل 2 با استفاده از نورون McCulloch Pitts
- 4 شکل 5. شبکه محاسبه مکمل 2 با استفاده از نورون McCulloch Pitts-ادغام شده
- 8 شکل 6. نتایج حاصل از شبکه‌ی پیاده‌سازی شده برای محاسبه‌ی مکمل 2 یک عدد 4 بیتی
- 9 شکل 7. نمایش یک تصویر تصادفی از هر کلاس در مجموعه دادگان
- 10 شکل 8. هیستوگرام مربوط به توزیع کلاس‌ها در داده‌های train و test
- 12 شکل 9. نمودارهای متناظر با Accuracy و loss برای داده‌های train و validation
- 13 شکل 10. تغییرات دقت مدل و اثرگذاری حمله‌ی FGSM بر داده‌های تست برای مقادیر مختلف epsilon
- 14 شکل 11. پیش‌بینی FC_model از اولین عضو از کلاس در مجموعه دادگان تست
- 15 شکل 12. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله FGSM با epsilon=0.025
- 15 شکل 13. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله FGSM با epsilon=0.1
- 16 شکل 14. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله FGSM با epsilon=0.5
- 17 شکل 15. تغییرات دقت مدل و اثرگذاری حمله‌ی PGD بر داده‌های تست برای مقادیر مختلف alpha و epsilon
- 18 شکل 16. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله PGD با alpha=0.01
- 18 شکل 17. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله PGD با alpha=0.05
- 19 شکل 18. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله PGD با alpha=0.1
- 22 شکل 19. نمودار پراکندگی داده Wine بعد از نرمالایز شدن برای دو ویژگی اول
- 24 شکل 20. نمودار میانگین خطای epoch برای آموزش شبکه Adaline برای جداسازی کلاس
- 25 شکل 21. نمودار میانگین خطای epoch در آموزش شبکه Adaline برای جداسازی کلاس 1
- 26 شکل 22. نمودار پراکندگی داده‌های wine به همراه خط‌های جداکننده آموزش دیده برای دو ویژگی اول
- 27 شکل 23. نمودار میانگین خطای epoch برای هر epoch در آموزش شبکه با همه ویژگی‌ها
- 29 شکل 24. نمونه‌ای شبکه Madaline (شبکه Adaline درونی)
- 31 شکل 25. توزیع داده‌های مصنوعی تولید شده
- 32 شکل 26. نتایج برای learning rate: 0.001, epochs = 200
- 32 شکل 27. نمودار پراکندگی و confusion matrix برای داده‌های تست با epoch=200, lr=0.001
- 33 شکل 28. نمودار پراکندگی و خطوط جدا کننده
- 33 شکل 29. نمودار پراکندگی و confusion matrix برای داده‌های تست با epoch=200, lr=0.01
- 34 شکل 30. نمودار پراکندگی داده‌ها به همراه سه خط جدا کننده
- 34 شکل 31. نمودار پراکندگی و confusion matrix برای داده‌های تست با epoch=200, lr=0.0001
- 35 شکل 32. نمودار پراکندگی داده‌ها و خطوط جدا کننده
- 35 شکل 33. نمودار پراکندگی داده‌های تست و confusion matrix برای epoch=400, lr=0.001
- 36 شکل 34. نمودار پراکندگی داده‌ها و خطوط جدا کننده
- 36 شکل 35. نمودار پراکندگی و confusion matrix برای داده‌های تست با epoch=1000, lr=0.001

37 شکل 36. نمودار پراکندگی دادهها و خطوط جدا کننده
37 شکل 37. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=2000, lr=0.001
38 شکل 38. نمودار پراکندگی دادهها و خطوط جدا کننده
38 شکل 39. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=3000, lr=0.001
39 شکل 40. نمودار پراکندگی دادهها و خطوط جدا کننده
39 شکل 41. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=5000, lr=0.001
40 شکل 42. نمودار پراکندگی دادهها و خطوط جدا کننده
40 شکل 43. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=200, lr=0.01
41 شکل 44. نمودار پراکندگی دادهها و خطوط جدا کننده
41 شکل 45. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=100, lr=0.001
42 شکل 46. نمودار پراکندگی دادهها و خطوط جدا کننده
42 شکل 47. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=300, lr=0.001
43 شکل 48. نمودار پراکندگی دادهها و خطوط جدا کننده
43 شکل 49. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=200, lr=0.01
44 شکل 50. نمودار پراکندگی دادهها و خطوط جدا کننده
44 شکل 51. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=200, lr=0.0001
45 شکل 52. نمودار پراکندگی دادهها و خطوط جدا کننده
45 شکل 53. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=200, lr=0.001
46 شکل 54. نمودار پراکندگی دادهها و خطوط جدا کننده
46 شکل 55. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=300, lr=0.001
47 شکل 56. نمودار پراکندگی دادهها و خطوط جدا کننده
47 شکل 57. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=100, lr=0.001
48 شکل 58. نمودار پراکندگی دادهها و خطوط جدا کننده
48 شکل 59. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=200, lr=0.01
49 شکل 60. نمودار پراکندگی دادهها و خطوط جدا کننده
49 شکل 61. نمودار پراکندگی و confusion matrix برای دادههای تست epoch=200, lr=0.0001
53 شکل 62.تابع سینوسی تولید شده
54 شکل 63. نمودار دادههای آموزش به همراه پیش‌بینی مدل از آن‌ها
54 شکل 64. نمودار دادههای تست به همراه پیش‌بینی مدل از آن‌ها
55 شکل 65. تغییرات loss، با تغییر test size
56 شکل 66. تغییرات loss با تابع فعال ساز ReLU
56 شکل 67. پیش‌بینی دادههای آموزش برای بهترین مدل
56 شکل 68. پیش‌بینی دادههای تست برای بهترین مدل
57 شکل 69. نمودار تغییرات loss با تغییرات test size
58 شکل 70. پیش‌بینی دادههای آموزش برای مدل ۶ لایه
58 شکل 71. پیش‌بینی دادههای تست برای مدل ۶ لایه
61 شکل 72. نتایج شبکه عصبی سه لایه برای استفاده از میزان متفاوت داده آموزش

63 شکل 73. پارامترهای آموزش شبکه‌های عصبی با تعداد لایه‌های متفاوت
64 شکل 74. نتایج حاصل از آموزش شبکه‌های مختلف از ۳ تا ۲۰ لایه
65 شکل 75. نتایج مدل‌ها از ۳ تا ۸ لایه
66 شکل 76. نتایج مدل‌ها از ۹ تا ۱۴ لایه
66 شکل 77. نتایج مدل‌ها از ۱۵ تا ۲۰ لایه

جدول‌ها

21.....	اطلاعات دیتاست Wine
25.....	جدول 2. پارامترهای آموزش شبکه Adaline
27.....	جدول 3. پارامترهای آموزش شبکه Adaline برای همه ویژگی‌ها
60.....	جدول 4. مشخصات شبکه عصبی مد نظر در هر مرحله
61.....	جدول 5. پارامترهای آموزش شبکه عصبی پیاده‌سازی شده

قبل از پاسخ دادن به پرسش‌ها، موارد زیر را با دقت مطالعه نمایید:

- از پاسخ‌های خود یک گزارش در قالبی که در صفحه‌ی درس در سامانه‌ی Elearn با نام **REPORTS TEMPLATE.docx** قرار داده شده تهیه نمایید.
- پیشنهاد می‌شود تمرین‌ها را در قالب گروه‌های دو نفره انجام دهید. (بیش از دو نفر مجاز نیست و تحويل تک نفره نیز نمره‌ی اضافی ندارد) توجه نمایید الزامی در یکسان ماندن اعضای گروه تا انتهای ترم وجود ندارد. (یعنی، می‌توانید تمرین اول را با شخص A و تمرین دوم را با شخص B و ... انجام دهید)
- **کیفیت گزارش شما در فرآیند تصحیح از اهمیت ویژه‌ای برخوردار است؛** بنابراین، لطفاً تمامی نکات و فرض‌هایی را که در پیاده‌سازی‌ها و محاسبات خود در نظر می‌گیرید در گزارش ذکر کنید.
- در گزارش خود مطابق با آنچه در قالب نمونه قرار داده شده، برای شکل‌ها زیرنویس و برای جدول‌ها بالانویس در نظر بگیرید.
- الزامی به ارائه توضیح جزئیات کد در گزارش نیست، اما باید نتایج بدست آمده از آن را گزارش و تحلیل کنید.
- **تحلیل نتایج الزامی می‌باشد، حتی اگر در صورت پرسش اشاره‌ای به آن نشده باشد.**
- **دستیاران آموزشی ملزم به اجرا کردن کدهای شما نیستند؛** بنابراین، هرگونه نتیجه و یا تحلیلی که در صورت پرسش از شما خواسته شده را به طور واضح و کامل در گزارش بیاورید. در صورت عدم رعایت این مورد، بدیهی است که از نمره تمرین کسر می‌شود.
- **کدها حتماً باید در قالب نوت‌بوک با پسوند ipynb.** تهیه شوند، در پایان کار، تمامی کد اجرا شود و خروجی هر سلول حتماً در این فایل ارسالی شما ذخیره شده باشد. بنابراین برای مثال اگر خروجی سلولی یک نمودار است که در گزارش آورده‌اید، این نمودار باید هم در گزارش هم در نوت‌بوک کدها وجود داشته باشد.
- **در صورت مشاهده تقلب امتیاز تمامی افراد شرکت‌کننده در آن، ۱۰۰ - لحظ می‌شود.**
- تنها زبان برنامه نویسی مجاز Python است.
- استفاده از کدهای آماده برای تمرین‌ها به هیچ وجه مجاز نیست. در صورتی که دو گروه از یک منبع مشترک استفاده کنند و کدهای مشابه تحويل دهند، تقلب محسوب می‌شود.
- نحوه محاسبه تاخیر به این شکل است: پس از پایان رسیدن مهلت ارسال گزارش، حداکثر تا یک هفته امکان ارسال با تاخیر وجود دارد، پس از این یک هفته نمره آن تکلیف برای شما صفر خواهد شد.

- سه روز اول: بدون جریمه
- روز چهارم: ۵ درصد
- روز پنجم: ۱۰ درصد
- روز ششم: ۱۵ درصد
- روز هفتم: ۲۰ درصد

- حداکثر نمره‌ای که برای هر سوال می‌توان اخذ کرد ۱۰۰ بوده و اگر مجموع بارم یک سوال بیشتر از ۱۰۰ باشد، در صورت اخذ نمره بیشتر از ۱۰۰، اعمال نخواهد شد.
- برای مثال: اگر نمره اخذ شده از سوال ۱ برابر ۱۰۵ و نمره سوال ۲ برابر ۹۵ باشد، نمره نهایی تمرین ۹۷.۵ خواهد بود و نه ۱۰۰.
- لطفاً گزارش، کدها و سایر ضمایم را به در یک پوشه با نام زیر قرار داده و آن را فشرده سازید، سپس در سامانه Elearn بارگذاری نمایید:

HW[Number]_[Lastname]_[StudentNumber]_[Lastname]_[StudentNumber].zip

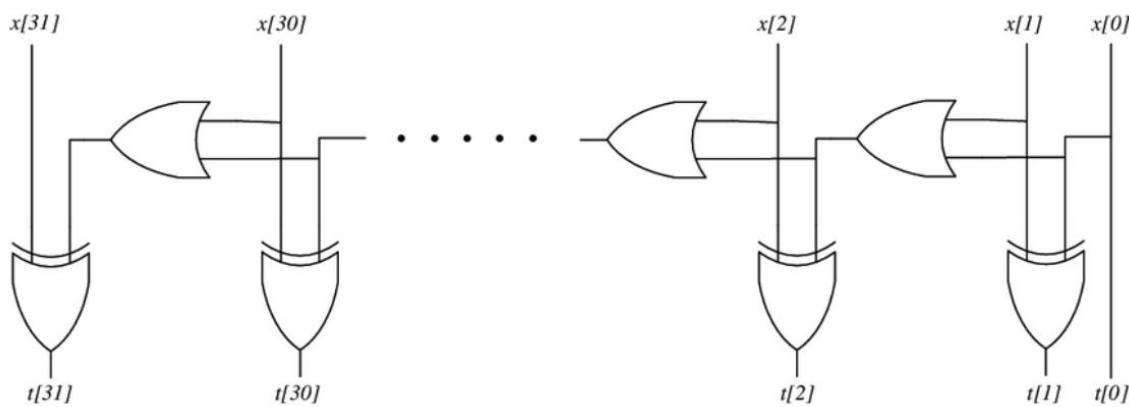
(HW1_Ahmadi_810199101_Bagheri_810199102.zip) (مثال:

- برای گروههای دو نفره، بارگذاری تمرین از جانب یکی از اعضا کافی است ولی پیشنهاد می‌شود هر دو نفر بارگذاری نمایند.

پرسش ۱ McCulloch Pitts - ۱

۱-۱. شبکه محاسبه مکمل ۲

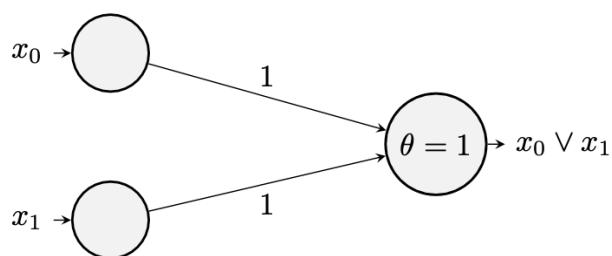
در این بخش، ابتدا سعی داریم با استفاده از نورون‌های McCulloch Pitts، شبکه‌ی محاسبه کننده‌ی مکمل ۲ یک عدد ۴ بیتی باینری را با استفاده از شبیه‌سازی مدار منطقی زیر برای ۴ بیت، انجام دهیم.



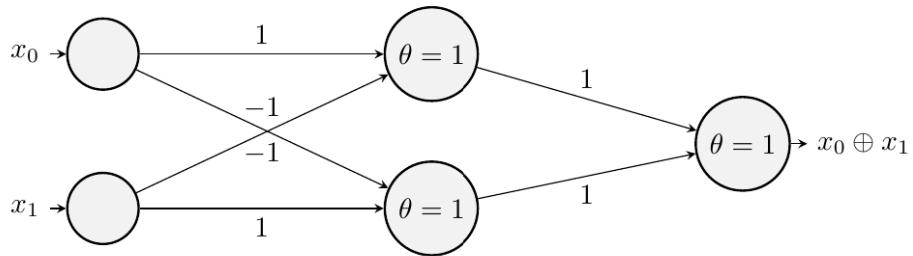
شکل ۱. تصویر نمونه از ساختار شبکه مدنظر

۱-۲. پیاده‌سازی تئوری شبکه

برای پیاده‌سازی، کافیست به جای هر یک از گیت‌های منطقی در مدار، ترکیبی از نورون‌ها که همان کاربرد را دارند، قرار دهیم. نورون‌های متناظر با اعمال OR و XOR و با آستانه‌ی $\theta = 1$ به ترتیب در زیر آمده‌اند:



شکل ۲. شبیه‌سازی عملیات OR منطقی با استفاده از نورون‌های M&P



شکل 3. شبیه‌سازی عملیات **XOR** منطقی با استفاده از نورون‌های M&P

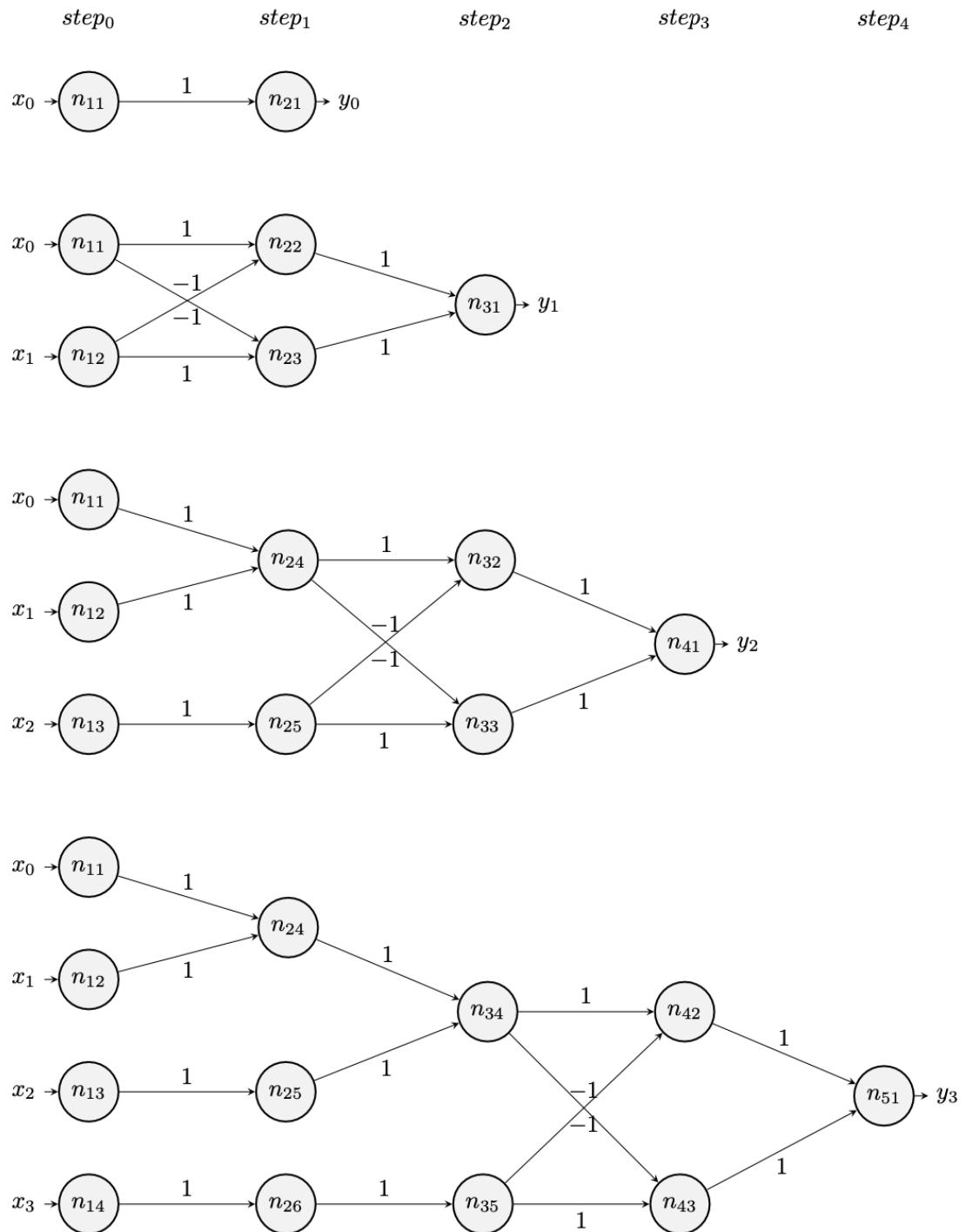
از آنجا که در صورت سوال، پیاده‌سازی شبکه برای یک عدد باینری ۴ بیتی خواسته شده، عدد ورودی را x و حاصل خروجی را y می‌نامیم و هر یک را به صورت زیر در نظر میگیریم طوری که اندیس کوچکتر، ارزش کمتری در عدد دارد:

$$x = x_3x_2x_1x_0, y = y_3y_2y_1y_0$$

طبق مداری که در شکل ۱ آمده، برای شبیه‌سازی شبکه‌ی مورد نظر به صورت زیر عمل می‌کنیم:

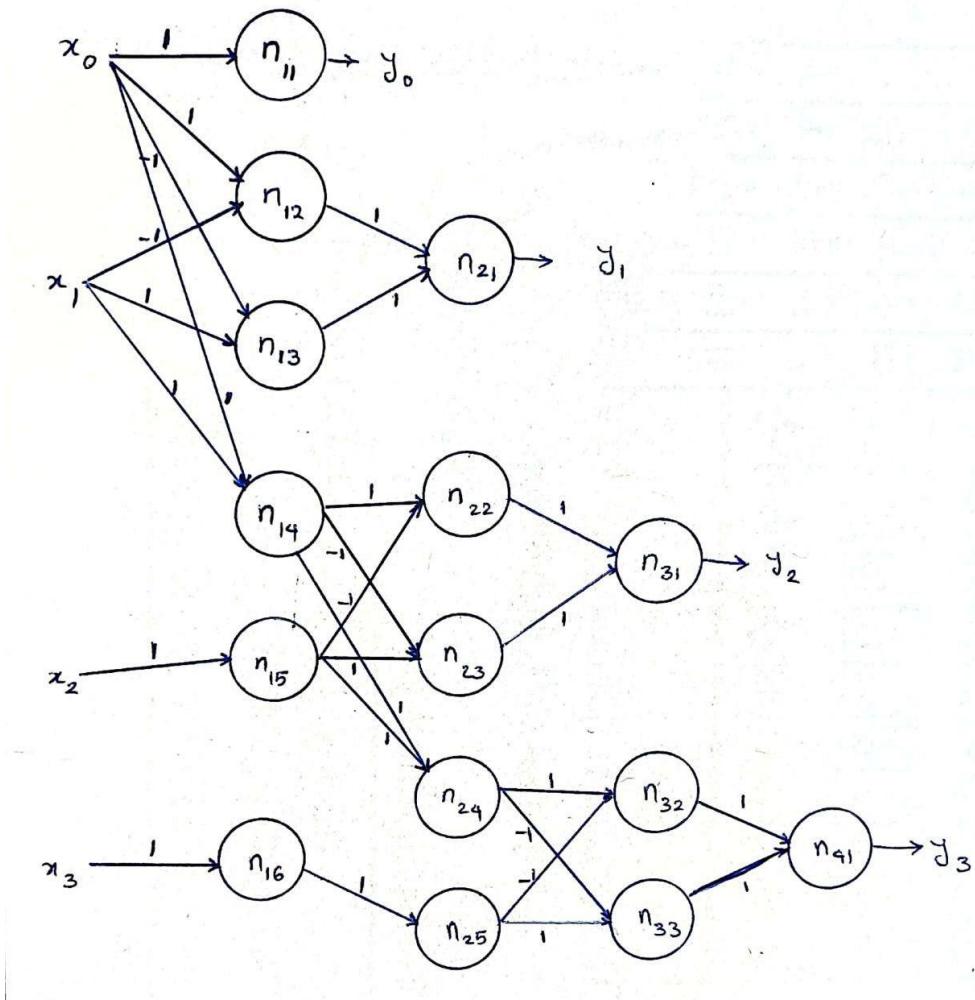
- برای تولید کم‌ارزش‌ترین رقم (y_0) در خروجی، کافیست کم‌ارزش‌ترین رقم ورودی (x_0) را قرار دهیم.
- برای تولید دومین رقم کم ارزش در خروجی (y_1), باید $x_0 \oplus x_1$ را قرار دهیم.
- برای تولید سومین رقم کم ارزش در خروجی (y_2) نیز باید $x_2 \oplus (x_1 \vee x_0)$ را قرار دهیم.
- برای تولید بالارزش‌ترین رقم نیز اینگونه عمل می‌کنیم: $y_3 = x_3 \oplus (x_2 \vee (x_1 \vee x_0))$.

ایده طراحی این شبکه براین اساس است که در مدار داده شده، گیتهای OR و XOR را با زیر شبکه‌های مرتبط ساخته شده با استفاده از نورون‌های McCulloch Pitts جایگزین کنیم. در این شبکه آستانه تمام نورون‌ها ۱ در نظر گرفته شده و بدین ترتیب وزن اتصالات بین نورون‌های شبکه، محاسبه شده است. همانگونه که مشاهده می‌کنید، در هر step به صورت موازی، اطلاعات انتقال پیدا می‌کنند. بنابراین در مرحله اول، کم‌ارزش‌ترین بیت مکمل ۲ عدد ورودی تولید می‌شود. سپس در مرحله دوم، دومین بیت کم‌ارزش، و به همین ترتیب عدد مکمل ۲ توسط شبکه تولید می‌شود. در این شکل برخی از نورون‌ها در واقع یک نورون هستند (با یک نام) اما به دلیل تمیزی و مشخص بودن شبکه و بهتر دیده شدن اتصالات بین نورون‌ها، تکرار شده‌اند.



شکل 4. شبکه محاسبه مکمل ۲ با استفاده از نورون McCulloch Pitts

مرج شده‌ی شبکه‌های فوق نیز به صورت زیر است:



شکل ۵. شبکه محاسبه مکمل ۲ با استفاده از نورون McCulloch Pitts-ادغام شده

۱-۳. پیاده‌سازی کد شبکه

در فایل ژوپیتر پیوست شده، سه کلاس وجود دارد که این سه کلاس، در واقع ساختار یک شبکه که از نورون‌های McCulloch Pitts استفاده می‌کند را می‌سازند.

۱. در هر نورون صرفا آستانه آن نورون (در اینجا همه ۱) نگهداری می‌شود. همچنین این کلاس دارای تابع محاسبه کننده‌ی خروجی بر حسب ورودی با توجه به آستانه‌ی تعیین شده است که در اینجا از step function استفاده می‌کنیم؛ به این صورت که مقادیر بزرگتر و مساوی از آستانه، مقدار ۱ و مابقی مقادیر، مقدار ۰ می‌گیرند.

2. در هر لایه از این شبکه، تعداد نورون‌ها و همچنین خود آنها ذخیره شده‌اند. از طرفی، تابع activation function نیز وجود دارد که از تابع مرتبط با آن از کلاس نورون، برای محاسبه خروجی‌های هر لایه استفاده می‌کند.

3. در هر شبکه، تعداد لایه‌ها، تعداد نورون در هر لایه و همچنین اطلاعات مربوط به آن‌ها ذخیره می‌شود. همین‌طور، وزن‌های یال‌های بین نورون‌ها در لایه‌های متوالی به صورت ماتریس در آن ذخیره می‌شوند. علاوه بر این‌ها، یک تابع تست برای بررسی کارکرد شبکه براساس وزن‌ها و یال‌های شبکه وجود دارد.

نحوه کارکرد کد و اعمال یک ورودی به شبکه تا رفتن به خروجی و محاسبه مکمل ۲: ابتدا یک نمونه از این شبکه با توجه به پارامترهای موجود در شکل قسمت قبل (طراحی شبکه) ایجاد می‌کنیم. حال با استفاده از تابع تست موجود در کلاس شبکه، ورودی را اعمال می‌کنیم. در ابتدا وزن‌های لایه‌ی اول به دوم در بردار ستونی ورودی از چپ ضرب می‌شوند و بردار ورودی برای لایه دوم تولید می‌شود. این وزن‌ها با توجه به ساختار شبکه در مرحله طراحی به صورت ماتریس درآمده‌اند؛ به این صورت که هر سطر بیانگر نورونی در لایه بعدی (در اینجا لایه دوم) است که هر ستون آن بیان می‌کند که چه وزنی به کدام یک از نورون‌های لایه فعلی (در اینجا، لایه اول) متصل هستند، که با توجه به نحوه کارکرد نورون M&P، مقادیر نورون‌های لایه فعلی در وزن‌ها ضرب شده و ورودی‌های نورون‌های لایه بعد را تولید می‌کنند. حال، با استفاده از توابع فعال‌ساز نورون‌ها در لایه‌ی دوم، خروجی لایه‌ی دوم تولید می‌شود. در این مرحله، اولین عضو بردار ستونی خروجی از لایه‌ی دوم که بیانگر کمارزش‌ترین بیت مکمل ۲ عدد ورودی در مرحله‌ی اول است را طبق شکل جدا و ذخیره کرده و با استفاده از مابقی ورودی، به کار خود ادامه می‌دهیم (چون دیگر نیازی به این مقدار نداریم، آن را حذف کردیم. همچنین می‌توانستیم آن را تا مرحله‌ی آخر نگه داشته و یکباره همه‌ی بیت‌ها را خروجی دهیم). حال همین کار را برای رفتن از لایه‌ی دوم به لایه‌ی سوم و سپس به لایه‌ی چهارم انجام می‌دهیم که در هر مرحله مانند مرحله اول، بیت‌های مطلوب یک به یک تولید شده و آنها را از مابقی جدا و ذخیره می‌کنیم. بنابراین، به صورت موازی مرحله‌ها جلو رفته و بیت‌ها تک به تک تولید می‌شوند. در انتهای نیز، نتیجه نهایی بازگردانده می‌شود. با توجه به تست کیس‌های موجود در فایل ژوپیتر، مشاهده می‌شود که شبکه به خوبی و طبق انتظار ما عمل کرده و مکمل ۲ اعداد ورودی را تولید کرده و خروجی می‌دهد.

• توضیحات تکمیلی کد:

از آنجا که برای نگهداری وزن‌ها از ماتریس استفاده می‌کنیم و قرار است روی این ماتریس‌ها اعمالی مثل ضرب را انجام دهیم، در ابتدای امر، کتابخانه‌ی NumPy را که برای کار با آرایه‌ها و ماتریس‌ها است را import می‌کنیم. سپس به پیاده‌سازی عمومی شبکه می‌پردازیم؛ همانطور که گفته شد، شبکه‌ی ما شامل سه کلاس است: کلاس اول با نام Neuron که برای هر نورون در شبکه یک آستانه و همچنین یکتابع فعال‌ساز در نظر می‌گیرد. کلاس دوم با نام Layer که برای هر لایه به طور خاص در نظر گرفته می‌شود و طبق تعریف، هر لایه در شبکه‌ی عصبی، دارای تعدادی نورون از کلاس اول است که برای هر نورون نیز یک آستانه در نظر گرفته می‌شود و ما در این شبیه‌سازی، مشابه قسمت طراحی شبکه، تمام آستانه‌ها را ۱ در نظر گرفته‌ایم. همچنین، یک تابع با نام activation function نیز برای هر لایه در نظر گرفته شده که با توجه به شکل لایه‌ی ورودی، تابع فعال‌ساز برای نورون‌ها در لایه‌ی فعلی را تنظیم می‌کند.

کلاس سوم نیز با نام M_P_network تعریف شده که مربوط به کلیت شبکه‌ی عصبی است که طبق تعریف، شامل تعدادی لایه است، در هر لایه، اعدادی نورون دارد، هر نورون دارای یک آستانه است و هر یال که بین نورون‌ها در شبکه متصل است، دارای وزن می‌باشد. پس برای ساخت چنین شبکه‌ای، باید در ابتدا به اندازه‌ی تعداد لایه‌های شبکه، از کلاس Layer به شبکه‌مان لایه اضافه کنیم و با استفاده از تعداد نورون‌ها و آستانه‌هایی که ورودی داده شده‌اند، نورون‌ها را متناظر با هر لایه، به آن بیفزاییم. در این کلاس، یک تابع construct input نیز وجود دارد که ...

تا اینجا به ساخت کلیت یک شبکه‌ی عصبی پرداختیم. حال برای پیاده‌سازی شبکه‌ای که مکمل دو یک عدد باینتری 4 بیتی را محاسبه کند، باید پارامترهای شبکه را مشخص کنیم؛ این کار را بر اساس شکل 5 انجام می‌دهیم:

1. تعداد لایه‌ها: همانطور که در شکل مشخص است، شبکه‌ی ما، بدون در نظر گرفتن لایه‌ی ورودی، دارای 4 لایه است.

2. تعداد نورون‌ها: در لایه‌ی اول 6 نورون، در لایه‌ی دوم 5 نورون و در لایه‌های سوم و چهارم به ترتیب 3 و 1 نورون داریم.

3. آستانه‌ها: تمام آستانه‌ها در شبکه‌ی ما 1 در نظر گرفته شده‌اند (می‌توان مقادیر دیگری را نیز در نظر گرفت اما ما برای سادگی، 1 در نظر گرفته‌ایم). لازم به ذکر است که هر درایه در لیست دو بعدی thresholds، به تعداد نورون‌های لایه‌ی متناظر با اندیس آن درایه، حاوی 1 است.

4. وزن‌ها: با توجه به نورون‌های M&P متناظر با هر یک از عملیات مثل انتقال، OR و XOR، وزن یال‌ها در شکل ۵ مشخص شده و این مقادیر را به صورت ماتریسی ذخیره می‌کنیم و به شبکه می‌دهیم تا خواسته مسئله ارضاء شود. از آنجا که شبکه دارای ۴ لایه است، پس لیست حاوی وزن‌ها، دارای 4×4 ماتریس است که هر یک متناظر با یال‌های خروجی از یک لایه است. ماتریس اول 4×6 است چرا که عدد ورودی ۴ بیتی و لایه‌ی اول دارای ۶ نورون است و هر سطر از ماتریس، متناظر با یک نورون در آن لایه است طوری که هر سطر، به تعداد یال‌های ورودی به آن نورون، درایه‌ی غیر صفر دارد. مثلا سطر دوم در ماتریس اول به صورت $[1, -1, 0, 0]$ است که یعنی نورون دوم در لایه‌ی اول شبکه، یک یال ورودی از x_0 با وزن ۱ و یک یال ورودی از x_1 با وزن -۱ دارد. ماتریس دوم 5×5 است زیرا لایه‌ی دوم دارای ۵ نورون است و همچنین تعداد نورون‌هایی از لایه‌ی اول که یال متصل به نورون‌های لایه‌ی دوم دارند، برابر ۵ است (اولین نورون در لایه‌ی اول، پس از تولید خروجی، غیر فعال می‌شود). ماتریس سوم 4×3 است زیرا این لایه دارای ۳ نورون است و تعداد نورون‌هایی از لایه‌ی دوم که به این لایه یال دارند، برابر ۴ است. و نهایتاً، ماتریس چهارم 2×1 است چرا که این لایه یک نورون دارد و تعداد نورون‌هایی از لایه‌ی سوم که به این لایه متصل‌اند، برابر ۲ است. نکته‌ی قابل توجه در ساخت این ماتریس این است که در صورتی که درایه‌ی متناظر با یک خانه در یک ماتریس، یالی متناظر در شکل نداشته باشد، مقدار آن را برابر صفر قرار داده‌ایم.

در نهایت، یک شبکه‌ی عصبی با پارامترهایی که خواسته مسئله را برآورده کنند، می‌سازیم و به تست کردن آن می‌پردازیم. در ادامه، پارامترهای شبکه و همینطور نتایج تست‌کیس‌های داده شده به شبکه آورده شده‌اند:

```
n_layers = 4

n_neurons = [6, 5, 3, 1]

thresholds = [[1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1], [1]]

weights = [[[1, 0, 0, 0], [1, -1, 0, 0], [-1, 1, 0, 0], [1, 1, 0, 0],
[0, 0, 1, 0], [0, 0, 0, 1]],

[[1, 1, 0, 0], [0, 0, 1, -1, 0], [0, 0, -1, 1, 0], [0, 0, 1, 1,
0], [0, 0, 0, 0, 1]],

[[1, 1, 0, 0], [0, 0, 1, -1], [0, 0, -1, 1]],

[1, 1]]
```

0001 is the binary representation of 1. We can test that the 2's complement of this number which is -1, is calculated correctly.

```
[ ] 1 print(comp_model.Test("0001"))  
1111
```

0000 is the binary representation of 0. We can test that the 2's complement of this number which is 0, is calculated correctly.

```
[ ] 1 print(comp_model.Test("0000"))  
0000
```

1111 is the binary representation of -1. We can test that the 2's complement of this number which is 1, is calculated correctly.

```
[ ] 1 print(comp_model.Test("1111"))  
0001
```

1011 is the binary representation of 11. We can test that the 2's complement of this number which is -11, is calculated correctly.

```
[ ] 1 print(comp_model.Test("1011"))  
0101
```

0110 is the binary representation of 6. We can test that the 2's complement of this number which is -6, is calculated correctly.

```
[ ] 1 print(comp_model.Test("0110"))  
1010
```

شکل 6. نتایج حاصل از شبکه‌ی پیاده‌سازی شده برای محاسبه‌ی مکمل ۲ یک عدد ۴ بیتی

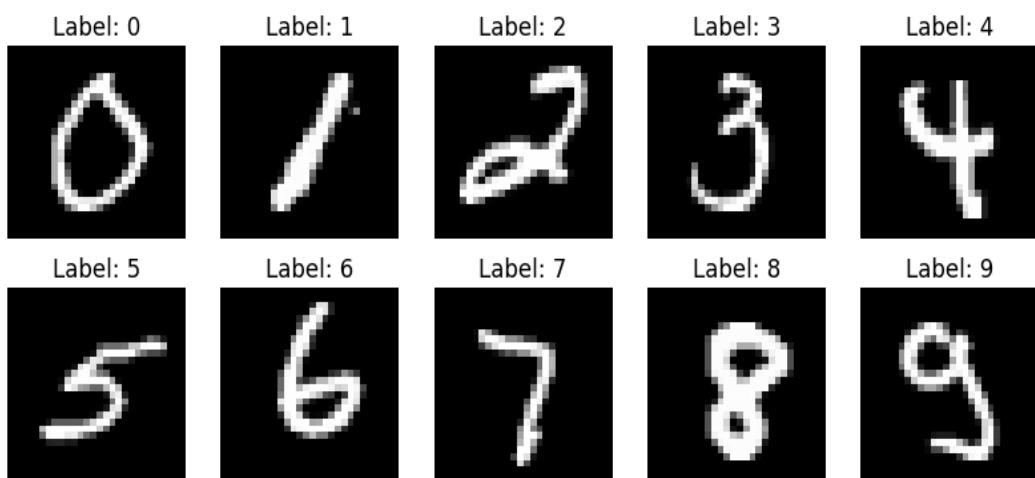
همانطور که در شکل فوق پیداست، شبکه برای تست‌کیس‌های مختلف به درستی عمل می‌کند.

پرسش ۲ - حملات خصمانه در شبکه‌های عصبی

۲-۲. آشنایی با مجموعه دادگان

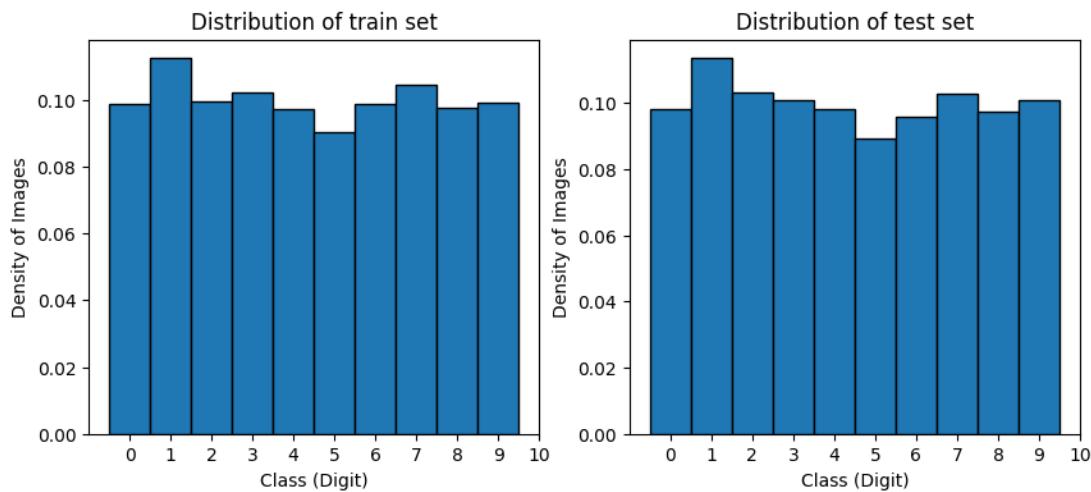
در این بخش، ابتدا داده‌های مورد نظر را از کلاس tesnorflow.keras.datasets.mnist فراخوانی می‌کنیم. از آنجا که برای آموزش شبکه، به دو دسته نمونه نیاز داریم، داده‌ها را در دو بخش train و test می‌خوانیم. برای این کار، با استفاده از mnist.load_data داده‌ها را تقسیم می‌کنیم. با بررسی مجموعه‌ها در می‌باییم که مجموعه‌ی آموزش دارای 60000 نمونه و مجموعه‌ی تست دارای 10000 نمونه است. همچنین، ابعاد تصاویر در هر دو مجموعه (28, 28) است.

برای نمایش یک نمونه از تصاویر هر یک از ۱۰ کلاس دیتاست، با استفاده از کتابخانه‌ی random که آن را در ابتدا فراخوانده بودیم، از هر کلاس یک نمونه را به تصادف انتخاب کرده و نمایش می‌دهیم. نکته‌ی قابل توجه این است که از (42) استفاده کرده‌ایم که خروجی تصاویر در اجراء‌ای متفاوت، یکسان باشد و خروجی حاصل در نوت‌بوک و گزارش تفاوتی نداشته باشند. در شکل زیر نتایج این بخش قابل مشاهده است:



شکل ۷. نمایش یک تصویر تصادفی از هر کلاس در مجموعه دادگان

برای رسم هیستوگرام مربوط به داده‌های آموزش و تست، ابتدا دو لیست از برچسب تصاویر در هر یک از مجموعه‌های train و test تهیه می‌کنیم و آنها را train_labels و test_labels می‌نامیم. نهایتاً با استفاده‌ها از کتابخانه‌ی matplotlib و نمودار hist از بخش pyplot این پکیج، به رسم هیستوگرام متناظر با هر یک از مجموعه دادگان train و test می‌پردازیم. هیستوگرام مربوط به توزیع کلاس‌ها برای هر یک از مجموعه‌های train و test به صورت زیر است:



شکل 8. هیستوگرام مربوط به توزیع کلاس‌ها در داده‌های test و train

با توجه به نمودارهای شکل ۸، توزیع کلاس‌ها در مجموعه‌های train و test تقریباً یکسان است و اصطلاحاً گفته می‌شود مجموعه‌ی دادگان در حالت تعادل (بالанс) است. بنابراین نیازی به پردازشی جهت بالانس کردن آن نداریم.

برای انجام عمل min-max normalization از کتابخانه‌ی sklearn که در ابتدا فراخوانده بودیم، استفاده می‌کنیم. ابتدا یک scaler تعریف می‌کنیم و بازه‌ای که می‌خواهیم به آن نرمال کنیم را مشخص می‌کنیم: (0, 1). سپس، از آنجا کهMinMaxScaler فقط داده‌ها با بعد ۲ را پذیرا است و داده‌ها در دیتاست MNIST سه بعدی هستند، باید داده‌ها را reshape کنیم. درواقع بعد train_images و test_images برابر ۳ است و به این صورت است:

$$(n_samples(train/test), 28, 28)$$

مولفه‌ی اول، تعداد داده‌های آموزش، و مولفه دوم و سوم به ترتیب طول و عرض تصاویر را مشخص می‌کنند. پس به راحتی می‌توان بعد مجموعه را به ۲ کاهش داد طوری که ماتریس دو بعدی حاصل 6000×784 باشد. در واقع در این کاهش بعد، هر سطر ماتریس جدید متناظر با یک نمونه در دیتاست است و از آنجا که تصاویر دارای بعد $(28, 28)$ هستند و $28 \times 28 = 784$ هستون نماینده‌ی یک پیکسل در تصویر است.

نرمال سازی داده‌ها به چند دلیل انجام می‌شود: از آنجا که دیتاست شامل تعداد زیادی تصویر است، بدیهی است که محدوده‌ی مقادیر پیکسل‌ها در تصاویر مختلف، متفاوت باشد. اما با نرمال کردن به بازه‌ی $(0,1)$ ، تمام تصاویر یکسان می‌شود و پردازش دقیق‌تر و آسان‌تر صورت می‌گیرد. همچنین بعضی الگوریتم‌های یادگیری ماشین با دیتای نرمال شده خیلی سریع‌تر همگرا می‌شوند. همچنین، اگر ویژگی‌ها مقیاس‌های مختلفی داشته باشند (بعضی بزرگ و بعضی کوچک)، ویژگی‌هایی با مقادیر بزرگ‌تر ممکن است بر ویژگی‌های دیگر غلبه کنند و منجر به یادگیری biased شوند. دلیل دیگر این است که نرمال سازی داده‌ها منجر به کاهش خطای محاسباتی در الگوریتم‌هایی می‌شود که محاسبات دقیق و حساسی دارند. در واقع این کار باعث می‌شود محاسبات پایداری بیشتری داشته باشند و نتایج دقیق‌تر باشند. بنابراین، با مقیاس‌بندی ویژگی‌ها به محدوده‌های یکسان، هر ویژگی به‌طور متناسبی به فرآیند یادگیری کمک می‌کند و از این‌که هر ویژگی به تنها‌ی فرآیند یادگیری را تعیین کند، جلوگیری می‌کند و به یادگیری بهتر و عادلانه‌تری را منجر می‌شود.

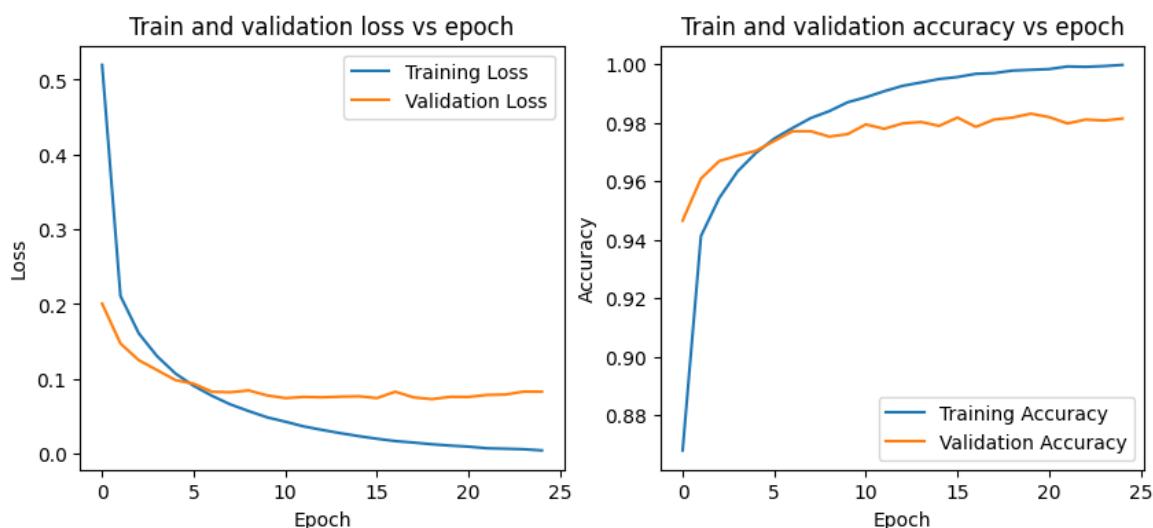
۳-۲. ایجاد و آموزش مدل

همانطور که گفته شده، تصاویر در مجموعه دادگان 28×28 هستند. برای دادن تصاویر به یک شبکه، نیاز داریم که بعد آن را از ۲ به ۱ کاهش دهیم. بنابراین بجای اینکه هر تصویر را به صورت 784 پیکسل که در یک مربع 28×28 جای گرفته‌اند در نظر بگیریم، آن‌ها را به صورت یک آرایه با 784 درایه و در یک بعد در نظر می‌گیریم.

در لایه‌های پنهان از تابع فعال‌ساز ReLU استفاده شده که دلیل استفاده از آن، سادگی‌اش و همچنین کارایی این تابع در تبدیل ترکیبات خطی به خروجی‌های غیر خطی است (ویژگی غیر خطی را افزایش می‌دهد). علاوه براین، ReLU مشکل vanishing gradient را نیز کاهش می‌دهد. از طرفی، معمولاً در مسائل طبقه‌بندی چند کلاسه softmax در آخرین لایه به عنوان فعال‌ساز استفاده می‌شود. این تابع خروجی‌های خام شبکه یا همان logits را به صورت احتمالاتی بین کلاس‌های مختلف موجود، توزیع می‌کند طوری که هر تصویر ورودی، با یک مقدار احتمال به یکی از کلاس‌ها خروجی تخصیص داده شود و جمع این احتمالات برای هر ورودی، برابر یک شود. بنابراین، در آخرین لایه از softmax استفاده می‌کنیم تا کلاسی که با بیشترین احتمال ورودی به آن تعلق دارد را پیش‌بینی کنیم.

پس از ساخت مدل، نوبت به بررسی عملکرد آن روی داده‌های تست می‌رسد. با انجام این کار، به نتایج زیر دست یافته‌ایم که مشخص می‌کند برای داده‌های validation، مقدار accuracy مدل در ایپاک‌ها به طور پیوسته در حال افزایش است. همچنین loss برای این داده‌ها پیوسته کم می‌شود. در این تغییرات، بیشترین شیب (به ترتیب افزایش و کاهش) تا قبل از ایپاک پنجم اتفاق می‌افتد و پس از آن، تغییرات به سرعت کمتری رقم می‌خورند (طبق این مسئله، حدود ۵ ایپاک نیز برای حصول نتایج مشابه، کافی بود). همچنین روی داده‌های تست، علی‌رغم یک دست و صاف نبودن نمودار، به طور کلی accuracy در حال افزایش و loss در حال کاهش است و بدین معناست که با پیشروی در ایپاک‌ها، عملکرد مدل هم روی داده‌های validation و هم روی داده‌های train در حال بهبود است. نهایتاً با evaluate کردن مدل روی داده‌های تست به حدود ۹۸٪ درصد و loss می‌رسیم که مقادیر قابل قبولی هستند.

در شکل ۹، روند افزایشی دقت مدل روی داده‌های train و همچنین داده‌های validation و از طرفی روند کاهشی loss برای این داده‌ها را می‌توان دید.



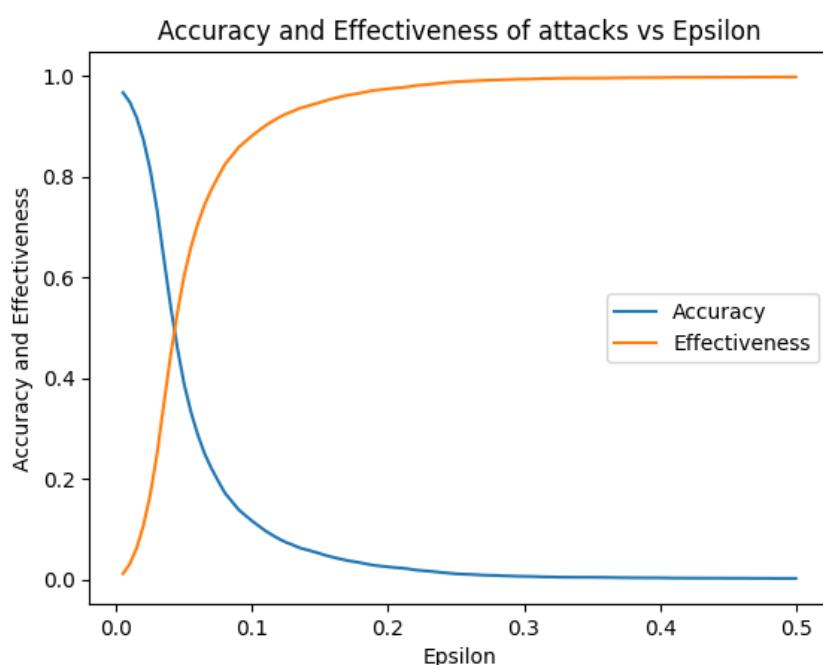
شکل ۹. نمودارهای متناظر با **train** و **validation** روی داده‌های **loss** و **Accuracy**

۴-۲. پیاده‌سازی حمله FGSM

حمله‌ی FGSM یک روش برای کاهش کارایی مدل‌های یادگیری ماشین است که با فرض آگاهی کامل از مدل، مقداری نویز براساس گرادیان loss به تصویر ورودی اعمال می‌کند و شبکه را در پیش‌بینی برچسب خروجی، گمراه می‌کند طوری که ممکن است حتی نویز توسط چشم انسان قابل شناسایی نباشد اما شبکه adversarial خروجی نادرستی ارائه دهد. در واقع این روش از گرادیان استفاده می‌کند تا یک نمونه‌ی loss بسازد. عملکرد این حمله این‌گونه است که برای هر تصویر ورودی، گرادیان loss بر حسب تصویر ورودی را محاسبه می‌کند تا یک تصویر جدید (adversarial image) تولید کند که loss را maximize کند. پیاده‌سازی این حمله، در بخش متناظر با FGSM در نوت‌بوک انجام شده‌است.

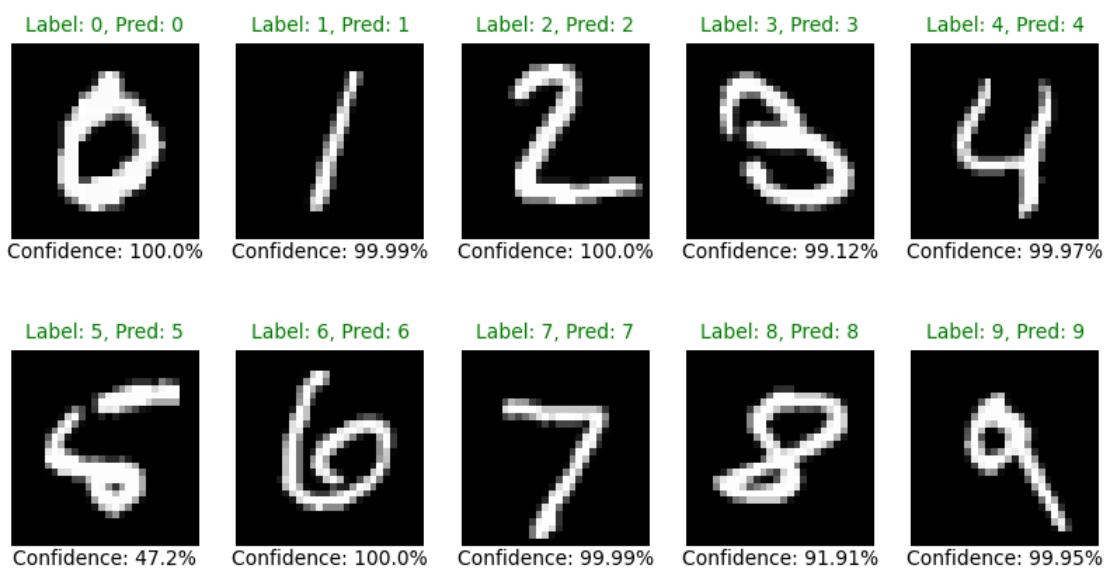
در بخش بعدی ما دقت و میزان اثرگذاری حمله را برای مقادیر اپسیلون از 0.005 تا 0.5 محاسبه کرده‌ایم و با افزایش مقدار اپسیلون، طبق انتظار، دقت پیش‌بینی مدل و میزان اثرگذاری حمله به طور پیوسته به ترتیب کاهش و افزایش یافتند.

می‌توانیم این افزایش میزان اثرگذاری و همچنین کاهش دقت مدل روی داده‌های حمله شده را در شکل ۱۰ مشاهده کنیم.



شکل ۱۰. تغییرات دقت مدل و اثرگذاری حمله‌ی FGSM بر داده‌های تست برای مقادیر مختلف epsilon

در این بخش سعی داریم نتایج تعدادی از حملات با مقادیر مختلف اپسیلوون را بررسی کنیم. برای این کار ابتدا اولین تصویر از هر کلاس در مجموعه تست را استخراج می‌کنیم و آنها را به همراه پیش‌بینی مدل نمایش می‌دهیم. همانطور که در شکل زیر قابل مشاهده است، تمام این مقادیر توسط مدل هنگامی که حمله‌آی صورت نگرفته باشد، به درستی پیش‌بینی می‌شوند.

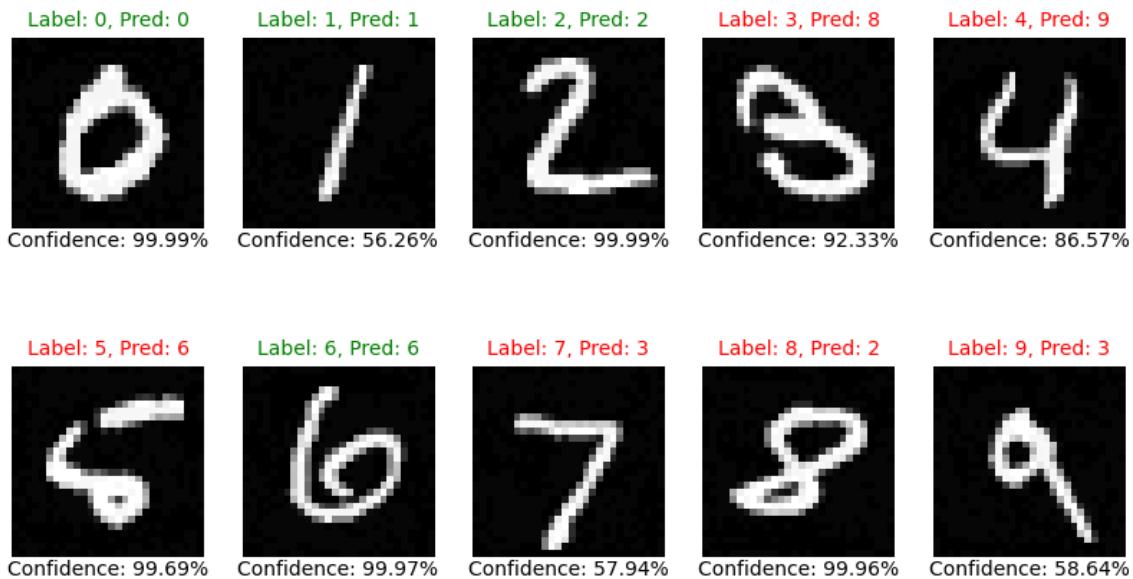


شکل 11. پیش‌بینی **FC_model** از اولین عضو ار کلاس در مجموعه دادگان تست

همان‌طور که می‌توان دید، مدل برای داده‌های قبل از حمله به خوبی و با درصد اطمینان بالایی برچسب‌های داده‌های تست را پیش‌بینی می‌کند. حال برای داده‌های حمله شده بررسی می‌کنیم:

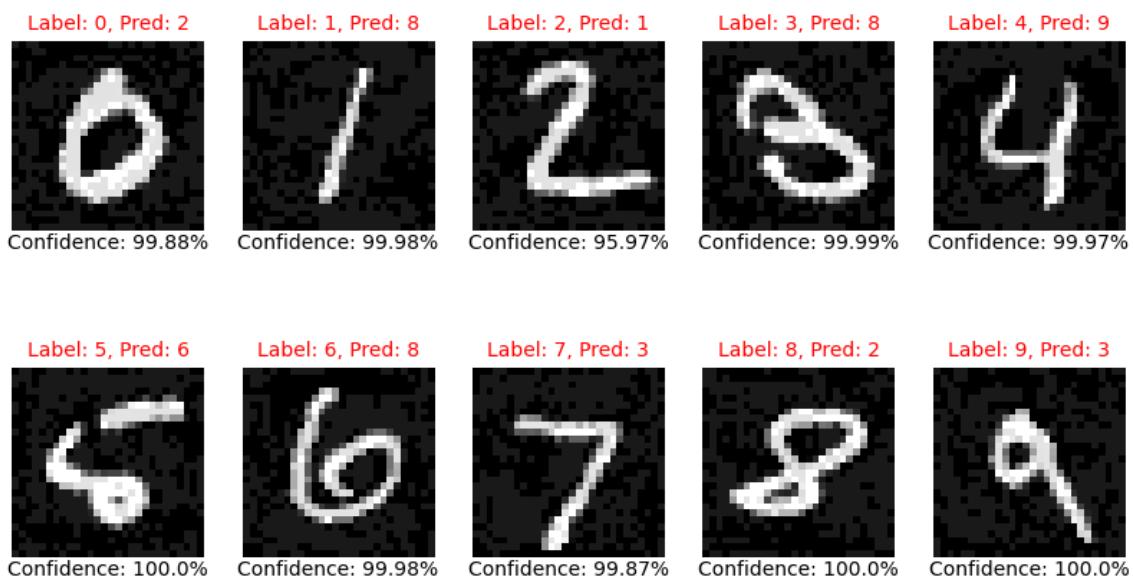
مقادیر 0.1 ، 0.05 و 0.025 را به دلخواه برای اپسیلون انتخاب می‌کنیم. در ادامه، نتایج ارزیابی تصاویر حاصل از حملات متناظر با هر یک از این مقادیر به ترتیب آورده می‌شود.

$\text{epsilon} = 0.025$

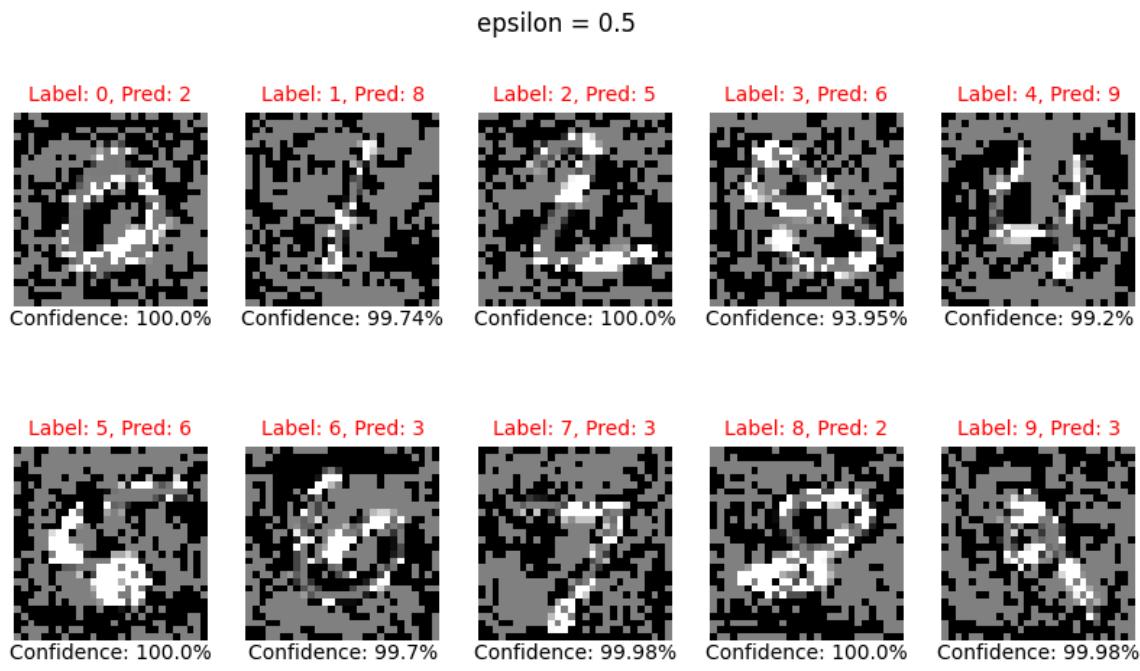


شکل 12. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله **FGSM** با $\text{epsilon}=0.025$

$\text{epsilon} = 0.1$



شکل 13. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله **FGSM** با $\text{epsilon}=0.1$



شکل 14. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله FGSM با $\epsilon=0.5$. همانطور که از نمودار شکل 10 نتیجه گرفته بودیم، با افزایش مقدار اپسیلون، دقت مدل و اثر گذاری حمله به ترتیب کاهش و افزایش می‌یابند. بنابراین، اینکه به طور پیوسته در شکل‌های 12 تا 14 تعداد پیش‌بینی‌های درست مدل و همچنین وضوح تصاویر کاهش یافته، منطقی است.

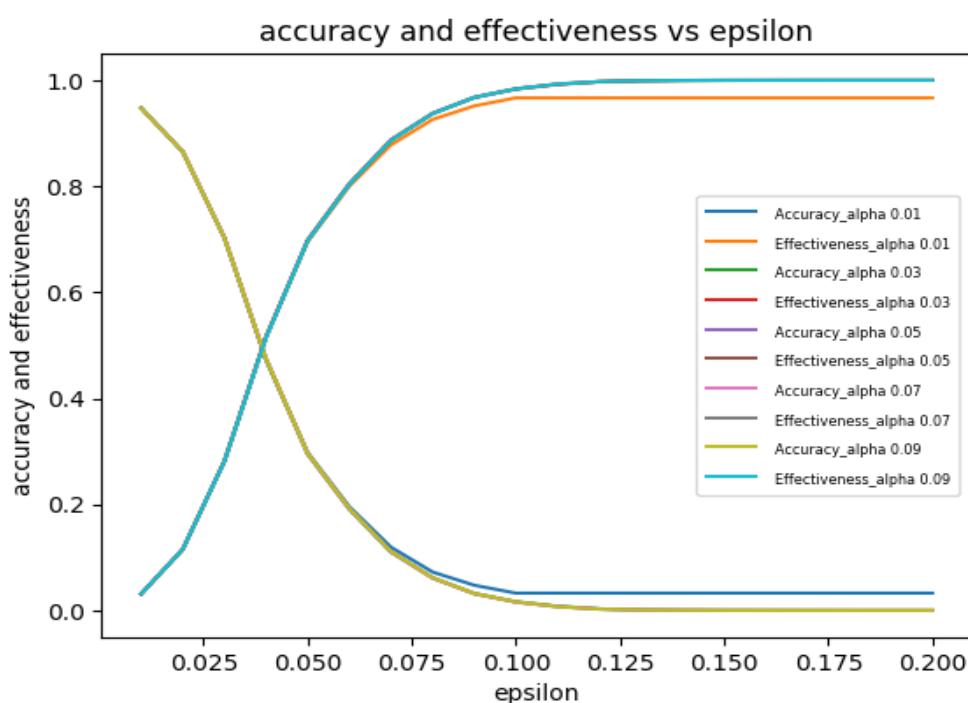
۲-۵. پیاده‌سازی حمله PGD

این حمله نیز مشابه حمله FGSM است با این تفاوت که در چند گام متوالی نویز را ایجاد می‌کند و در هر مرحله، از خروجی ایجاد شده در گام قبلی استفاده می‌کند. پیاده‌سازی این حمله در نوتبوک آورده شده است. در این حمله، در هر گام، برای هر تصویر ورودی آن مرحله گرادیان loss بر حسب تصویر ورودی محاسبه می‌شود و یک تصویر جدید (adversarial image) تولید شود که loss را maximize کند. همچنین تصویر خروجی هر مرحله، ورودی گام بعدی است.

برای بررسی حمله‌ی پیاده‌سازی شده مقادیر مختلفی برای اپسیلون و آلفا را در نظر گرفته‌ایم. برای هر یک از این دو پارامتر به ترتیب بازه‌های $(0.01, 0.2)$ و $(0.01, 0.1)$ در نظر گرفته شده‌اند. همچنین تعداد iteration‌ها روی ۵۰ تنظیم شده است. با بررسی کارکرد این حمله درمی‌یابیم که برای تمام مقادیر اپسیلون، به طور یکنواخت دقت مدل روی داده‌های تست کاهش یافته و اثرگذاری حمله افزایش یافته است که انتظار ما از ایجاد حمله را برآورده می‌کند. همچنین همانطور که در نمودار زیر قابل مشاهده است،

با افزایش مقدار آلفا، شیب کاهش دقت مدل روی داده‌های تست افزایش یافته و اثر گذاری حمله نیز افزایش یافته است. در واقع پس از حمله در آلفاهای بزرگتر، عملکرد مدل با سرعت بیشتری کاهش می‌یابد.

همانطور که در نمودار زیر مشخص است، در این حمله، برای آلفاهای متفاوت تا زمانی که مقدار اپسیلون به 0.05 می‌رسد، تقریباً هیچ تفاوتی ندارد و نمودارها بر هم منطبقند. اما از 0.05 به بعد، با افزایش آلفا، سرعت کاهش دقت مدل، کم شده و سرعت افزایش اثرگذاری حمله زیاد می‌شود. البته، این تغییر سرعت برای یک بازه‌ی محدود است و پس از آن که به یک نزدیک شدیم، سرعت‌ها یکسان می‌شوند. (شیب نمودار تقریباً صفر می‌شود).

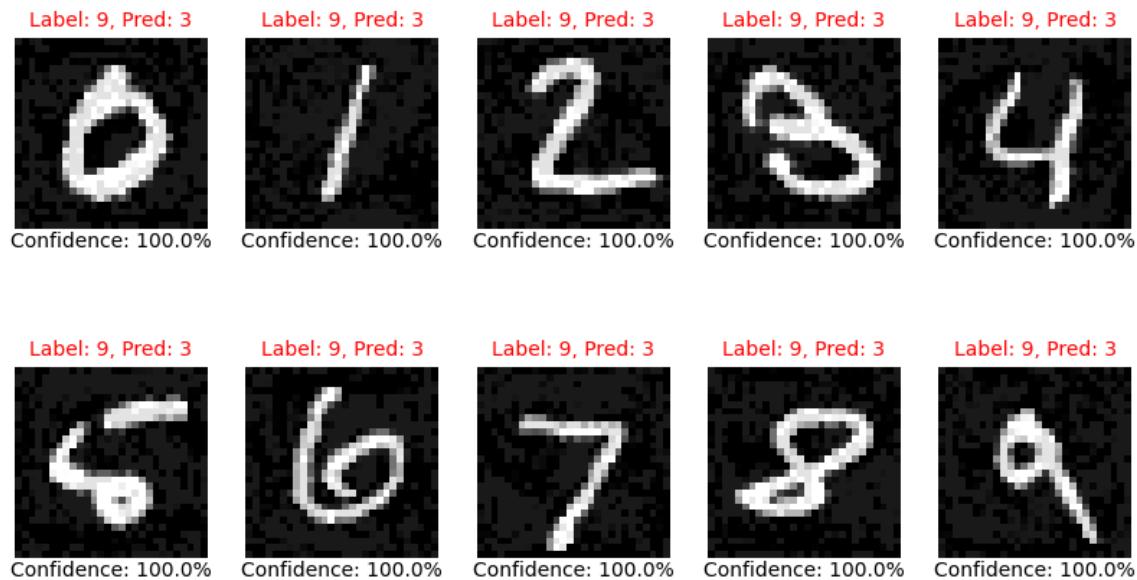


شکل 15. تغییرات دقت مدل و اثرگذاری حمله‌ی PGD بر داده‌های تست برای مقادیر مختلف **alpha** و **epsilon**

نتایج مطرح شده در بالا برای تصاویر 16 تا 18 (نمونه حمله و تاثیر گذاری بر داده‌های ورودی شبکه برای حمله PGD) نیز قابل تعمیم است. همانطور که در این شکل‌ها قابل مشاهده است، با افزایش آلفا، دقت مدل کاهش یافته و اثرگذاری حمله زیاد می‌شود. این در حالی است که با ضریب اطمینان نسبتاً بالایی مدل، برچسب‌ها را پیش‌بینی می‌کند!

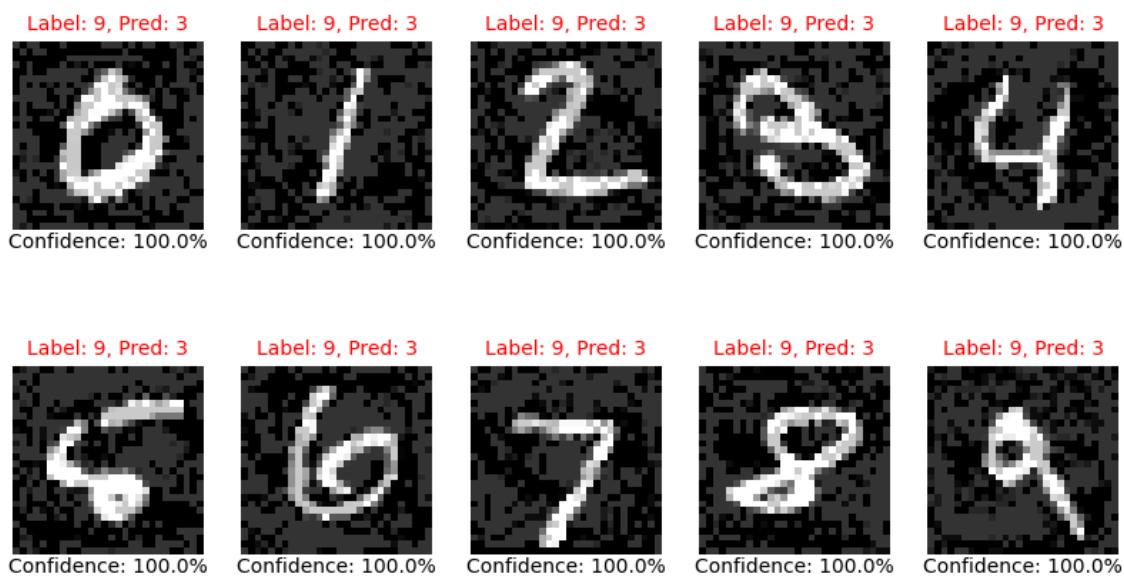
حال داده‌های perturbed شده با استفاده از الگوریتم PGD را برای آلفا برابر 1×10^{-5} و 1×10^{-4} در حالی که پارامتر اپسیلون در این ورودی برابر ۰.۱ است، بررسی می‌کنیم:

$$\text{Alpha} = 0.01$$

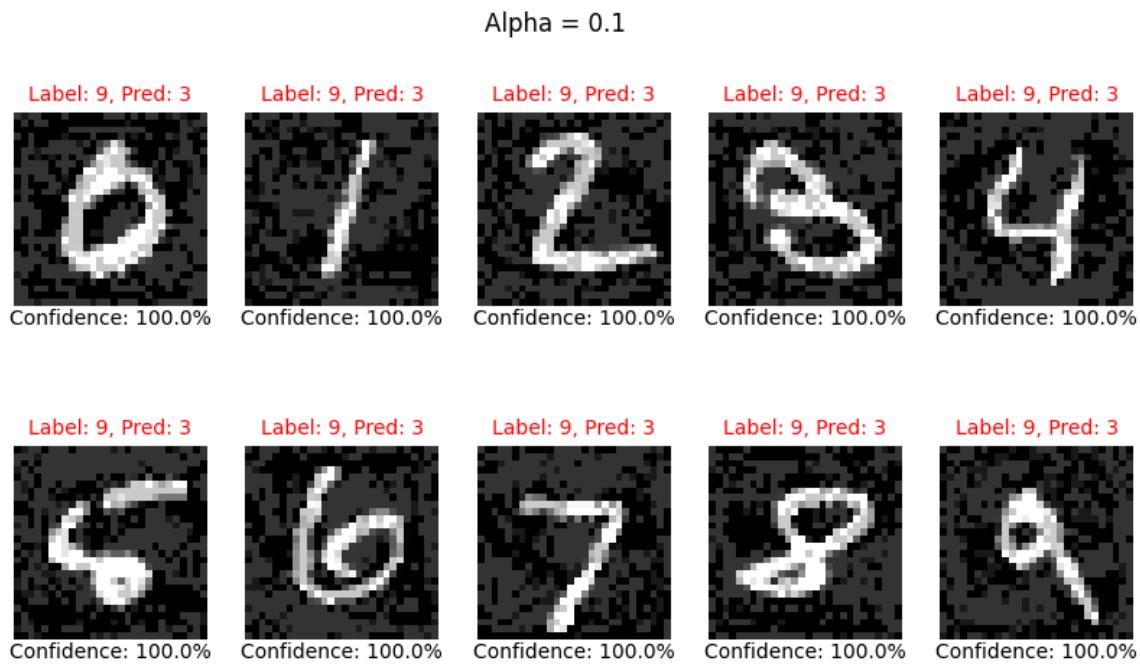


شکل ۱۶. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله PGD با $\alpha=0.01$

$$\text{Alpha} = 0.05$$



شکل ۱۷. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله PGD با $\alpha=0.05$



شکل 18. عملکرد مدل روی اولین تصویر از هر کلاس در مجموعه دادگان تست پس از حمله PGD با $\alpha=0.1$

• تفاوت‌های FGSM و PGD

هر دو این روش‌ها، تکنیک‌هایی برای سردرگم کردن مدل‌های یادگیری ماشین هستند همانطور که قبلاً توضیح داده شد، کار کرد مشابهی نیز دارند. اما ماهیت iterative بودن PGD منجر به ایجاد تفاوت‌هایی بین این دو روش می‌شود که در ادامه به آنها می‌پردازیم:

1. روش FGSM یک روش تک مرحله‌ای است که از علامت گرادیان محاسبه شده استفاده می‌کند؛ در حالیکه PGD یک روش iterative است که در گام‌های متوالی الگوریتم Gradient descent را اجرا می‌کند.

2. روش FGSM تغییرات جزئی در تصاویر ایجاد می‌کند که ممکن است تاثیر زیادی در برابر اقدامات دفاعی نداشته باشند، در حالیکه PGD محدوده‌ی بزرگتری از perturbation ها را در نظر می‌گیرد و نمونه‌های adversarial بهتری را ایجاد می‌کند.

3. روش PGD با توجه به ماهیت iterative بودنش، robustness بیشتری در مقایسه با FGSM دارد.

4. روش FGSM از لحاظ محاسباتی از PGD بهینه‌تر است، چرا که PGD به منابع بیشتری احتیاج دارد و گران‌تر تمام می‌شود. اما نتایج PGD در تولید تصاویر adversarial بهینه‌تر هستند و در واقع در این زمینه ما با یک trade-off مواجه هستیم!

• FGSM بر PGD برتری

1. با اینکه PGD منابع محاسباتی بیشتری نیاز دارد، اما انعطاف‌پذیری و پایداری بیشتر آن نسبت به FGSM باعث می‌شود که این روش برتری داشته باشد و مدل‌های یادگیری ماشین را بیشتر دچار سردرگمی کند و دقیق‌تر از FGSM کاهش دهد. (همانطور که در نتایج حاصل از هر دو حمله در نمودارهای ... و ... مشخص است، در FGSM، اثرگذاری حمله وقتی به یک نزدیک می‌شود که اپسیلون از 0.1 گذر کند؛ در حالی که در PGD ماهیت آن منجر می‌شود که برای تمام مقادیر آلفا، پیش از اینکه اپسیلون به 0.1 برسد، اثرگذاری حمله، به یک نزدیک شود).
2. تصاویر adversarial ساخته شده توسط PGD معمولاً قابلیت transferability بیشتری دارند، به این معنا که می‌توانند محدوده‌ی گسترده‌تری از مدل‌های یادگیری ماشین که برای انجام وظایف یکسانی آموزش دیده‌اند را دچار سردرگمی کنند.

پرسش ۳ – Madaline و Adaline

Adaline .۱-۳

Wine .۱-۱-۳

این داده شامل ۱۷۸ نمونه است. هر کدام از این نمونه‌ها با استفاده از ۱۳ ویژگی در دیتابست ذخیره شده‌اند. این ویژگی‌ها عبارت‌اند از:

Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavonoid, Nonflavonoid phenols, Proanthocyanins, Color intensity, Hue, od280/od315 of diluted wines, Proline.

بنابراین برای داده Wine داریم:

Wine Dataset	
Number of samples	178
Number of features	13

جدول ۱. اطلاعات دیتابست Wine

شراب‌های متفاوت با نام‌های class_0، class_1 و class_2 نامگذاری شده‌اند (متفاوت با صورت سوال).

۲-۱-۳. نرمال کردن داده

شبکه Adaline فرض می‌کند که داده‌ای که برای ورودی به آن استفاده می‌کنیم، داده‌ای با مرکز صفر و واریانس ۱ در هر ویژگی است. این کار را به وسیله StandardScaler از کتابخونه sklearn بخش preprocessing انجام می‌دهیم.

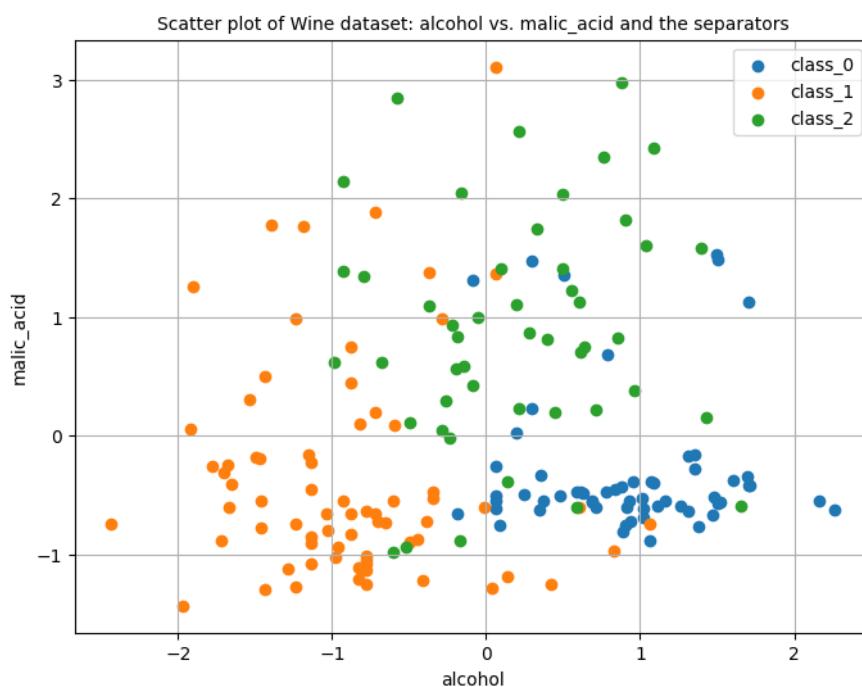
بنابراین داده بروز شده ما در هر یک از ۱۳ ویژگی خود میانگین صفر و واریانس ۱ خواهد داشت که کمک می‌کند تا شبکه Adaline ما بهتر یاد گرفته شود و برخی از وزن‌های آن زیادی بزرگ و یا زیادی کوچک نشوند.

همچنین لازم به ذکر است که دلیل استفاده از StandardScaler به این علت بود که این نرمال کنند، داده‌ها به صورت مورد نظر (میانگین صفر و واریانس یک) نرمال می‌کند. می‌توانستیم از مابقی نرمال‌سازها نیز استفاده کنیم، اما به دلیل استفاده از شبکه Adaline، ممکن است نتیجه به خوبی نتیجه استفاده از StandardScaler نباشد.

۳-۱-۳. نمودار پراکندگی داده‌ها

حال می‌توانیم نمودار پراکندگی داده‌ها را که با استفاده از کتابخونه matplotlib کشیده شده، بررسی

نماییم:



شکل ۱۹. نمودار پراکندگی داده Wine بعد از نرمالایز شدن برای دو ویژگی اول

همانطور کن مشاهده می‌کنیم، این داده‌ها برای دو ویژگی alcohol و malic_acid، به خوبی از یکدیگر جدا پذیر نیستند و داده‌هایی وجود دارد که ممکن است به اشتباه predict شود، بنابراین احتمالاً در مدل کردن خطی جدا پذیر برای این داده‌ها، به دقت خیلی خوبی نخواهیم رسید.

کلاس ۲ پخش‌ترین (عدم تمرکز در یک محدوده) کلاس‌ها است، بنابراین جداسازی این کلاس از کلاس‌های دیگر سخت‌تر از جداسازی کلاس‌های دیگر از مابقی کلاس‌ها است. (در واقع برای تشکیل یک مسئله ۲ کلاسه، برای مثال تشخیص کلاس صفر از مابقی کلاس‌ها، کلاس ۲ کاندیدا خوبی نمی‌باشد).

از طرفی با توجه به نمودار پراکندگی، می‌توان گفت که کلاس ۱ را، راحت‌تر می‌توان از مابقی کلاس‌ها جدا کرد، چرا که داده‌های کمتری از این کلاس با کلاس‌های دیگر تداخل دارند.

همانطور که مشخص است داده‌های کلاس صفر با رنگ آبی، کلاس یک با رنگ نارنجی و کلاس دو با رنگ سبز مشخص شده‌اند.

۴-۱-۳. پیاده‌سازی شبکه Adaline

حال برای ایجاد یک شبکه Adaline و آموزش آن، از کلاس **Adaline** استفاده می‌کنیم. این کلاس برای شبیه‌سازی شبکه Adaline با n ورودی و ۱ نورون خروجی استفاده می‌شود.

۱. در ابتدا با ساخت یک نمونه (instance) از این کلاس، تعداد نورون‌های ورودی را مشخص می‌کنیم. (طبق گفته صورت سوال، در این مرحله تنها از دو ویژگی اول یعنی alcohol و malic_acid استفاده می‌کنیم).

۲. سپس با استفاده از تابع `fit()` موجود در این کلاس و با استفاده از داده‌های آموزش مد نظر که قبلا target آن‌ها را به شیوه دلخواه (کلاس ۰ دارای target برابر ۱ و کلاس ۱ و ۲ دارای target ۱- باشند) درآورده. همچنین با استفاده از $random\ seed = 42$ ، داده‌ها را به داده‌های آموزش و تست تقسیم می‌کنیم.

نحوه آموزش شبکه با استفاده از تابع `fit()` موجود در کلاس **Adaline**:

- در این تابع، داده‌های آموزش و برچسب‌های متناظر با آن‌ها به همراه تعداد epoch لازم برای آموزش شبکه، learning rate و همچنین یک مقدار مینیمم خطأ برای هر داده دریافت می‌شود که اگر خطأ به ازای همه داده‌های آموزش کمتر از این مقدار باشد، به معنای آن است که شبکه به اندازه کافی آموزش داده شده و می‌توان این فرآیند را زودتر از تعداد epoch مدنظر خاتمه داد. (البته این شرایط در زمان آموزش برای ما پیش نیامد و همواره از شرط تعداد epoch برای خاتمه آموزش استفاده شد)

- حال تا زمانی که شبکه آموزش داده نشده (با توجه به شرایط ورودی):
 - ۱. دونه به دونه داده‌های آموزش را به شبکه داده و با استفاده از فرمول‌های موجود برای آموزش شبکه Adaline، وزن‌های شبکه و همچنین bias را بروز می‌کنیم. فرمول‌ها به صورت زیر هستند:

$$w_i^+ = w_i^- + \alpha(t - net)x_i$$

$$b^+ = b^- + \alpha(t - net)$$

که برای فرمول net داریم:

$$net = \sum_{i=1} x_i w_i + b$$

2. حال زمانی که همه داده‌های آموزش وارد شده و وزن‌های شبکه به ازای همه آن‌ها یادگرفته شدند، برای همه داده‌ها چک می‌کنیم که آیا خطا آن‌ها از مقدار مشخص ورودی کم‌تر است یا خیر، اگر بود آموزش را خاتمه میدهیم، در غیر اینصورت تا زمانی که تعداد epoch‌های مورد نظر را بگذرانیم، ادامه می‌دهیم.

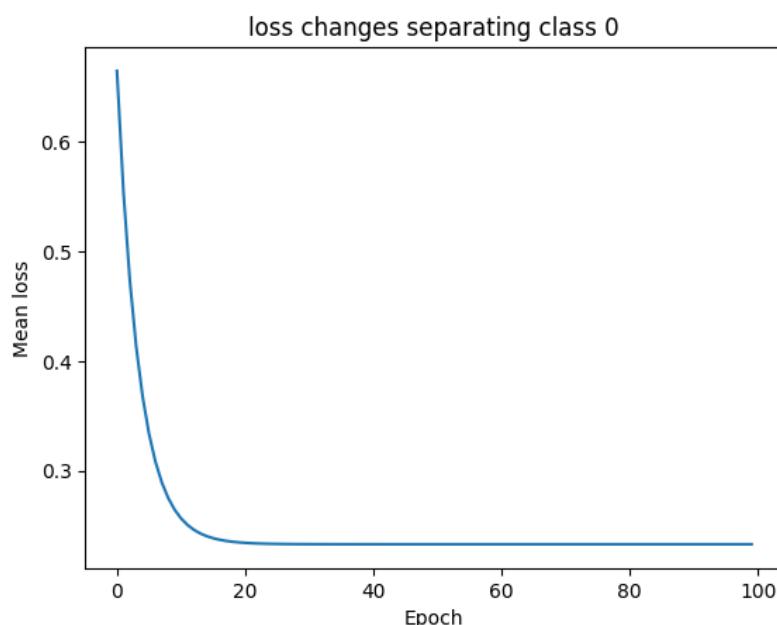
- همچنین خطا محاسبه شده میانگین گرفته شده و نمودار آن در انتهای آن داده می‌شود.
- همچنین لازم به ذکر است که خطا به ازای هر داده ورودی به این صورت محاسبه می‌شود:

$$error = \frac{1}{2} (t - net)^2$$

از طرفی دوتابع activation و predict نیز در شبکه وجود دارد که به ترتیب تابع فعال‌ساز برای نورون خروجی و پیش‌بینی داده ورودی استفاده می‌شوند. برای تابع فعال‌ساز داریم:

$$f(net) = \begin{cases} +1 & net \geq 0 \\ -1 & net < 0 \end{cases}$$

حال اگر این شبکه را برای داده‌هایی که کلاس صفر را از مابقی کلاس‌ها جدا می‌کند، آموزش دهیم، برای نمودار کاهش خطا خواهیم داشت:

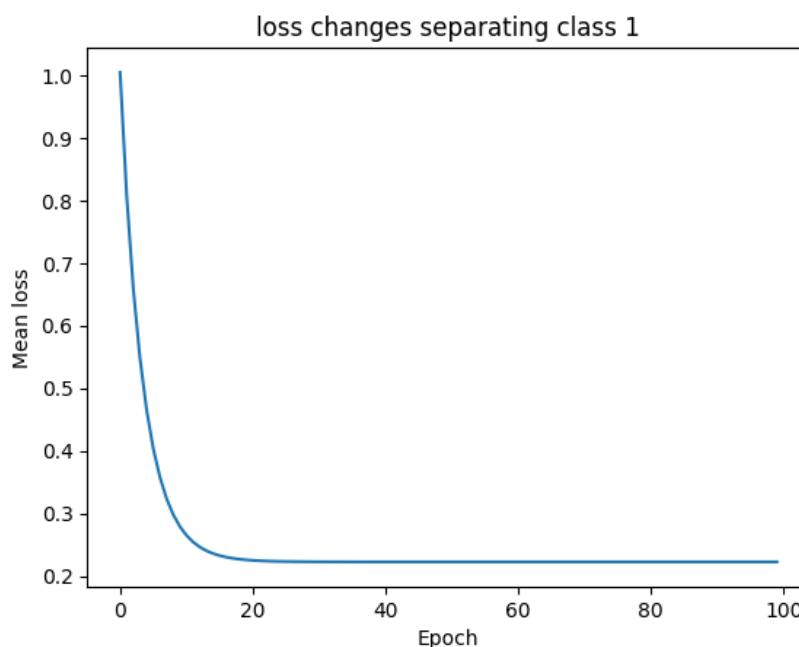


شکل 20. نمودار میانگین خطا برای هر epoch در آموزش شبکه Adaline برای جداسازی کلاس ۰.

همانطور که می‌بینیم میانگین خطا تا ۲۰ epoch کاهش خوبی داشته و بعد از آن تقریباً در نزدیکی ۰.۲۳ ثابت مانده. که این نشان‌دهنده آموزش کافی شبکه بر روی داده‌های آموزش است.

همچنین اگر داده‌های تست را بر روی شبکه امتحان کنیم، دقت ۹۷.۷۵ درصد خواهیم گرفت. اگر اینکار را برای داده‌های آموزش نیز انجام دهیم، دقت ۸۷.۶۴ خواهیم داشت که نشان‌دهنده میزان خوب یادگیری شبکه است.

حال اگر اینکار را برای داده‌هایی که به هدف جداسازی کلاس ۱ از مابقی کلاس‌ها تنظیم شده‌اند انجام دهیم، برای نمودار میانگین خطای خواهیم داشت:



شکل 21. نمودار میانگین خطای آموزش شبکه **Adaline** برای جداسازی کلاس ۱

حال اگر دقت کنیم، خطای در این حالت به نسبت به حالت قبل (جداسازی کلاس ۰ از دیگر کلاس‌ها) بیشتر کاهش یافته و میانگین خطای برابر ۰.۲۲ دارد. بنابراین می‌توان نتیجه گرفت که احتمالاً دقت شبکه در این حالت بیشتر از حالت قبل باشد. که اگر بررسی کنیم، این چنین نیز خواهد بود.

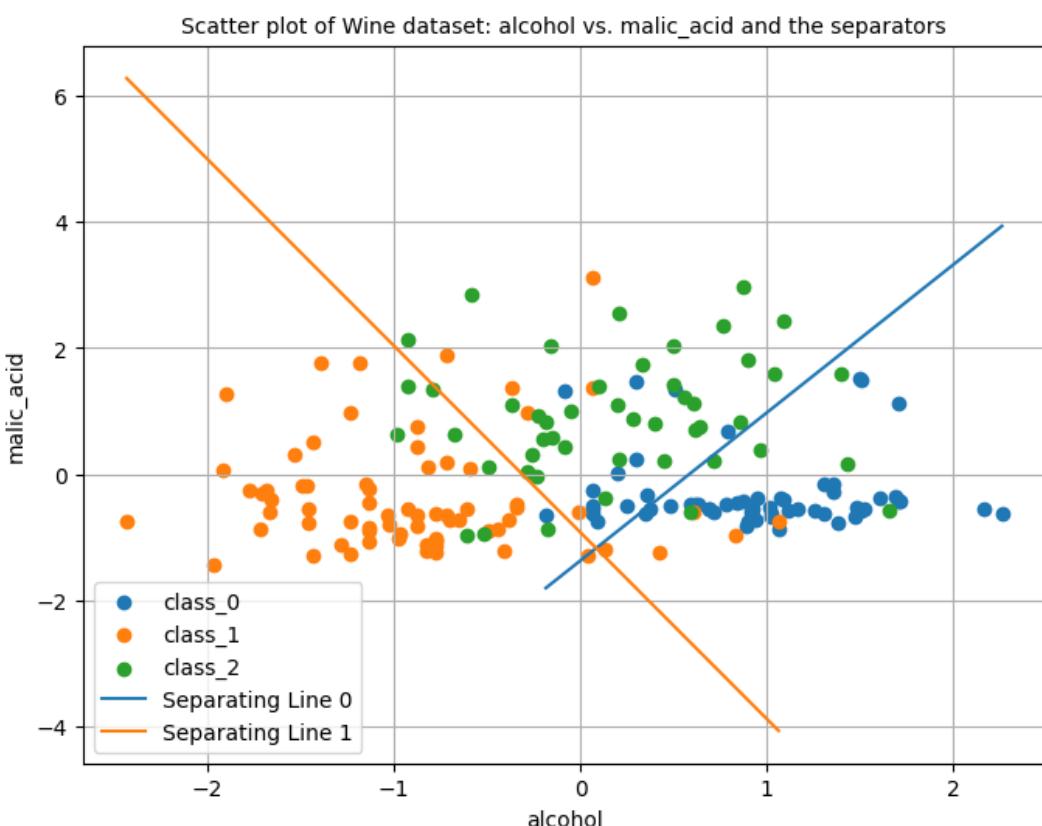
برای آموزش هر ۲ شبکه از $\text{random seed} = 42$ استفاده شد و پارامترهای ورودی برای آموزش هر دو شبکه عبارت‌اند از:

Learning rate	1e-3
Criterion	1e-2
Number on epochs	100
Number of input features	2 (alcohol, malic_acid)
Number of train data	142

جدول 2. پارامترهای آموزش شبکه **Adaline**

حال اگر دقت این مدل را برای داده‌های تست بررسی کنیم، دقت ۹۸.۸۸ درصد می‌گیریم، اما برای داده‌های آموزش دقت ۸۹.۸۹ درصد خواهیم گرفت. این اتفاق که دقت شبکه روی داده‌های آموزش کمتر از داده‌های تست است، احتمالاً دلیل در عدم تقسیم درست داده‌های اصلی به داده‌های آموزش و تست است (پراکندگی نادرست و غیر تصادفی داده‌ها در دیتاست (همه داده‌های یک کلاس پشت هم بودند)). اما در کل مشخص است که با توجه به میانگین خطای کمتر نسبت به شبکه قبل، دقت آن نیز بیشتر خواهد بود. در واقع به این معنی است که داده‌های کلاس ۱ نبست به کلاس ۰، بهتر از مابقی کلاس‌ها جدا می‌شوند.

حال اگر این داده‌ها را به همراه خط‌های جدا کننده آن در دو بعد، برای ویژگی‌های alcohol و malic_acid نمایش دهیم، خواهیم داشت:



شکل 22. نمودار پراکندگی داده‌های wine به همراه خط‌های جدا کننده آموزش دیده برای دو ویژگی اول

اگر دقت کنیم، خط‌ها به خوبی کلاس‌های مختلف را از هم جدا نمی‌کنند، اما خوب می‌توان دید که این خط بهترین خط و با کمترین خطای جداسازی یک کلاس، از مابقی کلاس‌های است. این اتفاق به دلیل این است داده‌ها به صورت ذاتی به خوبی جدایزیر نیستند و جدا کردن آنها با یک خط نسبتاً زیاد (۰.۲٪) انجام می‌شود.

از طرفی با توجه به دقت شبکه‌های آموزش داده شده برای جداسازی کلاس‌های ۰ و ۱ از مابقی کلاس‌ها (به صورت جداگانه)، دیدم که کلاس ۱ نسبت به کلاس ۰، بهتر از مابقی کلاس‌ها جدا می‌شود.

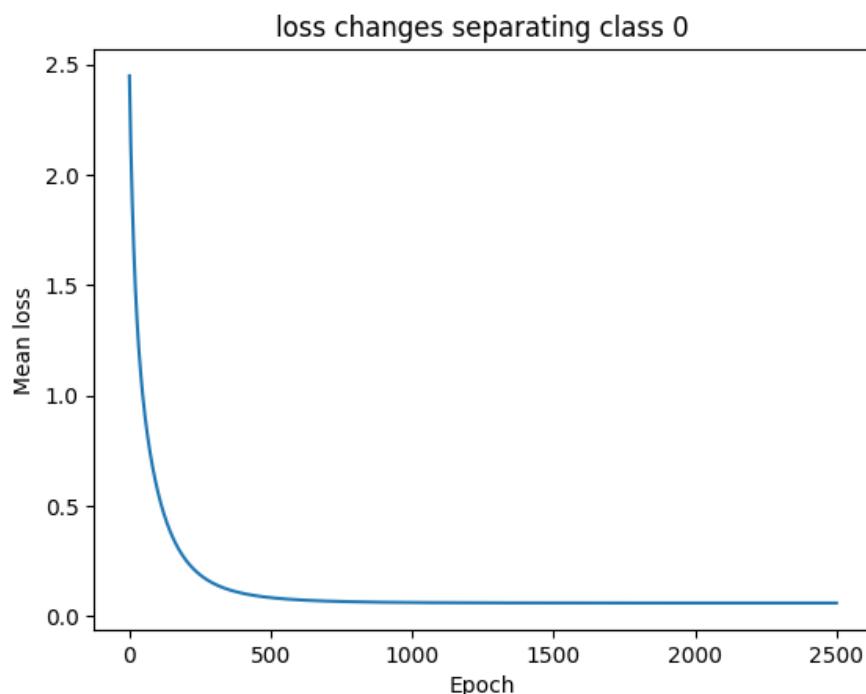
این اتفاق دلیل در خود داده‌ها به صورت ذاتی دارد. در واقع این داده‌ها برای دو ویژگی، آموزش دیده (malic_acid و alcohol)، به خوبی جدا پذیر نیستند. از طرفی با توجه به نمودار پراکندگی داده‌ها در این دو ویژگی، می‌توانیم ببینیم که کلاس ۰ و کلاس ۲، را نمی‌توان به خوبی از هم جدا کرد، در صورتی که این اتفاق (جداسازی کلاس ۱ از کلاس ۲) راحت‌تر برای کلاس ۱ می‌افتد.

حال اگر شبکه‌ای با همه ویژگی‌ها آموزش دهیم، احتمالاً نتیجه بهتری خواهیم گرفت. شبکه جدید را با این پارامترها آموزش می‌دهیم:

Learning rate	5e-5
Criterion	1e-2
Number on epochs	2500
Number of input features	13 (all features)
Number of train data	142

جدول 3. پارامترهای آموزش شبکه **Adaline** برای همه ویژگی‌ها

اگر نمودار کاهش میانگین خطای آن را بررسی کنیم، داریم:



شکل 23. نمودار میانگین خطای برای هر epoch در آموزش شبکه با همه ویژگی‌ها

مشاهده می کنیم که میانگین خطابه خوبی کاهش یافته (تا حدود ۰.۰۶) که بیانگر نتیجه بهتر در استفاده از همه ویژگی‌ها برای جداسازی کلاس · از مابقی کلاس‌ها است.

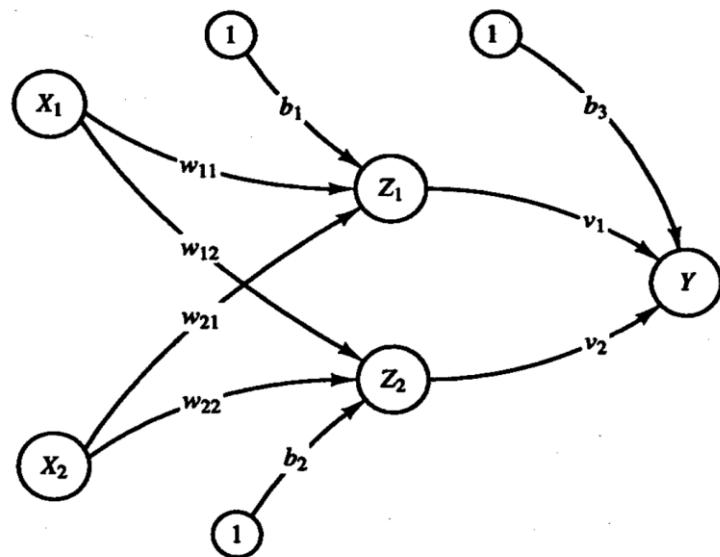
همچنین اگر دقت آن را هم برای داده‌های آموزش و تست بررسی کنیم، خواهیم داشت:

- دقت ۱۰۰ درصد برای داده‌های آموزش
- دقت ۱۰۰ درصد برای داده‌های تست

بنابراین استفاده از همه ویژگی‌ها نتیجه خیلی بهتر (۱۰۰ درصد برای همه داده‌ها) نسبت به استفاده از تنها دو ویژگی اول برای جداسازی کلاس‌ها در این دیتاست دارد.

الف) الگوریتم‌های MRI و MII

یک شبکه است که از کنار هم قرار گرفتن چند Adaline در لایه‌ی پنهان خود تشکیل شده و امکان حل مسائل پیچیده‌تری نسبت به Adaline را دارد. این شبکه دو الگوریتم متفاوت دارد؛ نوع اول و اساسی آن الگوریتمی است که در آن فقط وزن‌ها و bias برای Adaline‌های پنهان تنظیم می‌شود و وزن‌های لایه‌ی خروجی ثابت است. در حالیکه در الگوریتم نوع دوم، تمام وزن‌ها و bias ها در شبکه، از جمله وزن‌ها و bias لایه‌ی خروجی نیز توسط شبکه تنظیم می‌شود.



شکل 24. نمونه‌ای شبکه Adaline (شبکه ۲ درونی)

اگر شکل بالا را برای یک Madaline در نظر بگیریم، در MRI وزن‌های v_1 ، v_2 و b_3 ثابت‌اند اما سایر وزن‌ها در شبکه آپدیت می‌شوند تا زمانی که شرط توقف برآورده شود. اما در MII این سه مقدار نیز علاوه برای وزن‌های Adaline‌های پنهان نیز در هر مرحله آپدیت می‌شوند. طراحی هر دوی این شبکه‌ها طوریست که اگر حداقل یکی از Z_1 یا Z_2 یک باشد، خروجی 1 می‌شود و در غیر این صورت، خروجی -1 خواهد بود. در واقع Y همان نقش عملگر منطقی OR را در این شبکه ایفا می‌کند. برای ارضا نوش Y در این شبکه نیاز است که v_1 ، v_2 و b_3 همگی $\frac{1}{2}$ تنظیم شوند. همچنین، تابع فعال‌ساز برای این الگوریتم به صورت زیر است:

$$f(\text{net}) = \begin{cases} +1 & \text{net} \geq 0 \\ -1 & \text{net} < 0 \end{cases}$$

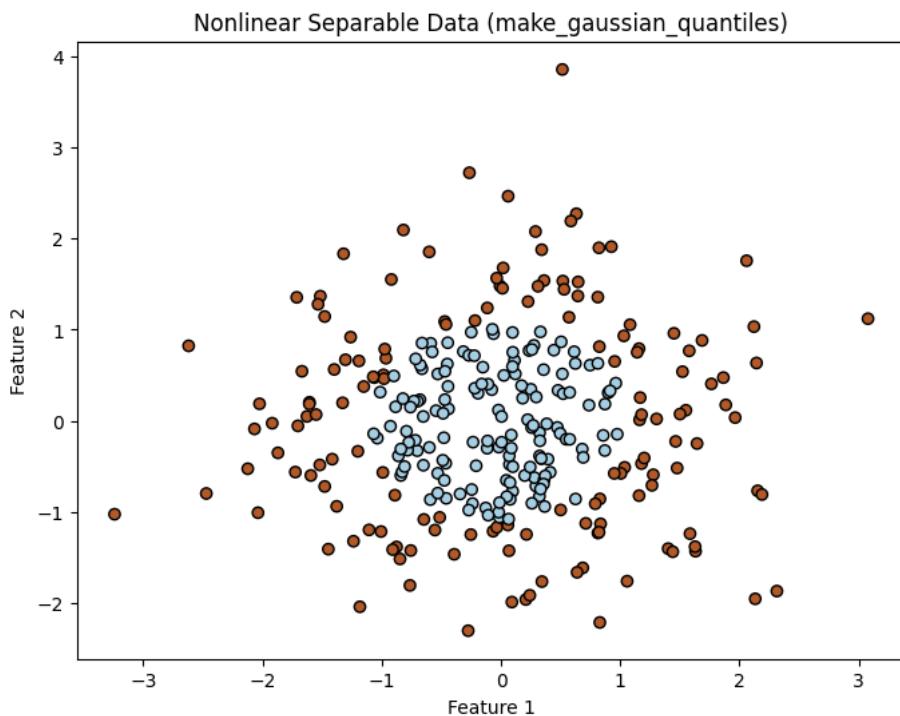
در الگوریتم MRI ابتدا وزن‌ها را با مقادیر کوچک رندوم مقداردهی اولیه می‌کنیم. همچنین وزن‌های مربوط به لایه‌ی خروجی را طوری تنظیم و ثابت می‌کنیم که γ مانند عملگر منطقی OR عمل کند. ضریب یادگیری را نیز روی یک مقدار کوچک تنظیم می‌کنیم. سپس، تا زمانی که شرط توقف برقرار شود، net input را برای هر Adaline پنهان در شبکه محاسبه کرده و خروجی هر Adaline را با اعمال تابع فعال‌ساز روی مقادیر آن، بدست می‌آوریم. نهایتاً، با استفاده از مقادیر به دست آمده خروجی شبکه را محاسبه کرده و تابع فعال‌ساز را روی آن اعمال می‌کنیم. همچنین، خطای را محاسبه می‌کنیم و به به روز رسانی وزن‌ها می‌پردازیم طوری که اگر خروجی شبکه برابر با target بود، هیچ وزنی آپدیت نشود. در غیر این صورت، اگر $target = 1$ باشد، فقط وزن‌های بخشی که net input آن بیشترین نزدیکی به صفر را دارد به روز می‌کنیم و اگر $target = -1$ باشد، وزن تمام بخش‌هایی که net input آن‌ها مثبت است را به روز می‌کنیم.

الگوریتم MRII مشابه MRI است با این تفاوت که تمام وزن‌ها به صورت تصادفی انتخاب شده و ضریب یادگیری نیز تنظیم می‌شود. تفاوت دیگر نیز هنگام به روز رسانی وزن‌هاست طوری که اگر خروجی شبکه برابر با target بود، هیچ وزنی آپدیت نشود. در غیر این صورت، برای هر بخش پنهانی که net input آن به اندازه‌ی کافی به صفر نزدیک است، با شروع از بخشی که بیشترین نزدیکی به صفر را دارد، علامت خروجی آن بخش (Adaline) را تغییر می‌دهیم و پاسخ شبکه را مجدداً حساب می‌کنیم. اگر پاسخ جدید خطای کمتری داشت، خروجی جدید آن بخش را (خروجی تغییر علامت داده شده) به عنوان target در نظر می‌گیریم و delta rule را مجدداً اعمال می‌کنیم.

ب) آموزش مدل

«برای پیاده‌سازی، الگوریتم MRI انتخاب شده است.»

از آنجا که داده‌ها طوری ساخته می‌شوند که دارای توزیع گاوی باشند، لذا نیاز به نرمال کردن داده‌ها نداریم. اما چون برچسب‌ها به صورت 0 و 1 هستند، باید آنها را به فرم bipolar تبدیل کنیم. لذا، لازم است تمام target‌های 0 را با -1 جایگزین کنیم. سپس به ساخت و آموزش شبکه می‌پردازیم.

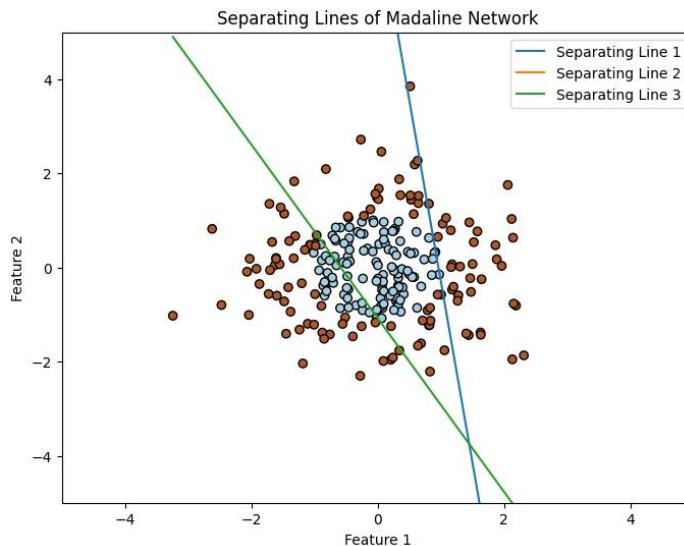


شکل 25. توزیع داده‌های مصنوعی تولید شده

همانطور که در بخش قبل توضیح داده شد، هر Adaline شامل تعدادی Madaline در شبکه‌ی ورودی خودش است که در کلاس متعلق به Madaline این تعداد به عنوان پارامتر ورودی دریافت می‌شود. همچنین، learning rate و تعداد epoch‌ها نیز از ورودی گرفته می‌شوند. برای تنظیم وزن‌های اولیه در این الگوریتم نیاز است که وزن‌ها و bias مربوط به لایه‌ی پنهان را با مقادیر تصادفی و کوچک تنظیم کنیم تا نورون‌ها در حالت فعال قرار بگیرند. اما وزن‌ها و bias مربوط به لایه‌ی خروجی ثابت هستند و مقدار هر یک از آنها را برابر 1 تقسیم بر تعداد Adaline‌های شبکه در مظر می‌گیریم تا خروجی شبکه به ازای هر ورودی مانند OR منطقی رفتار کند.تابع فعال‌ساز نیز همانطور که در قسمت قبل توضیح داده شد، در نظر گرفته می‌شود و در تابع fit در کلاس مذکور، الگوریتم MRI که در بالا توضیح دادیم، پیاده‌سازی شده است، طوری که در هر ایپاک، برای تمام داده‌ها خروجی شبکه به دست می‌آید و در صورت نیاز، وزن‌ها و bias‌ها آپدیت می‌شوند و خطای نیز محاسبه می‌گردد.

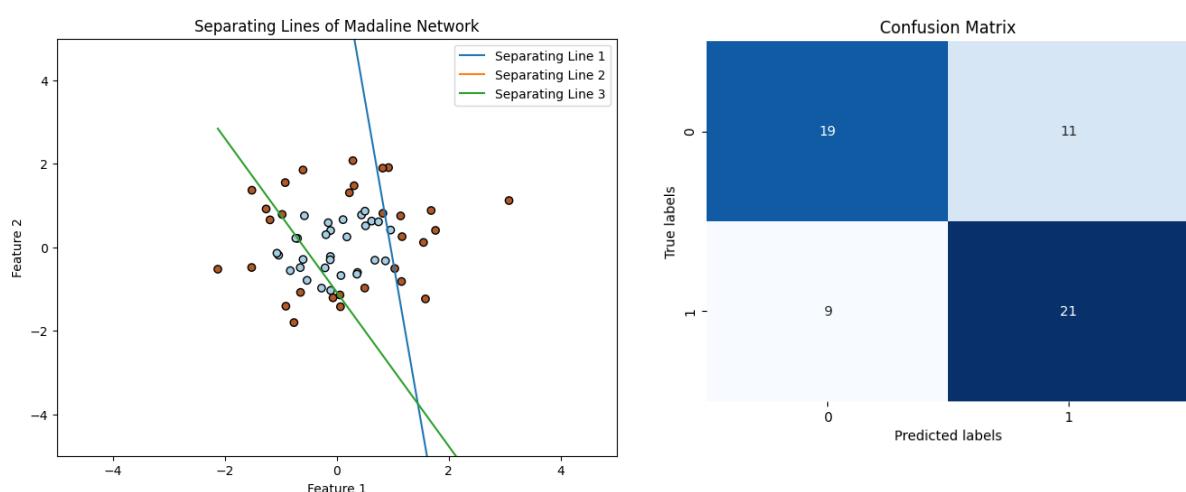
در ابتدا طبق خواسته‌ی سوال تعداد Adaline‌ها را برابر با ۳ در نظر می‌گیریم و حالات مختلف را بررسی می‌کنیم تا به بهترین مقادیر بررسیم:

learning rate: 0.001, epochs = 200 •



شکل 26. نتایج برای *learning rate: 0.001, epochs = 200*

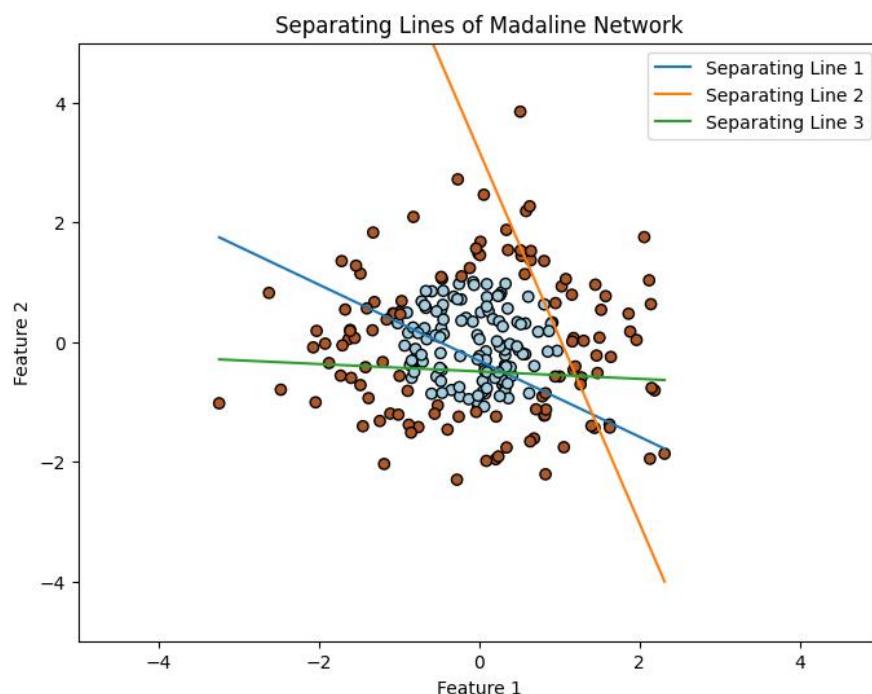
همانطور که مشخص است، با این مقادیر، یکی از خطهای جدا کننده از داده‌های تست خیلی دورتر است. این بدین معناست که پارامترهای تنظیم شده به خوبی داده‌ها را طبقه‌بندی نمی‌کنند و با پتانسیلی که از Madaline در ذهن داریم، این شبکه می‌تواند عملکرد بهتری داشته باشد. اما نتیجه‌ی اعمال این مدل با این مقادیر روی داده‌های تست حدود 67 درصد دقت است. نتایج روی مجموعه تست و ماتریس confusion به ترتیب در شکل‌های ۲۷ و ۲۸ آمده است.



شکل 27. نمودار پراکندگی و **confusion matrix** برای داده‌های تست با epoch=200, lr=0.001

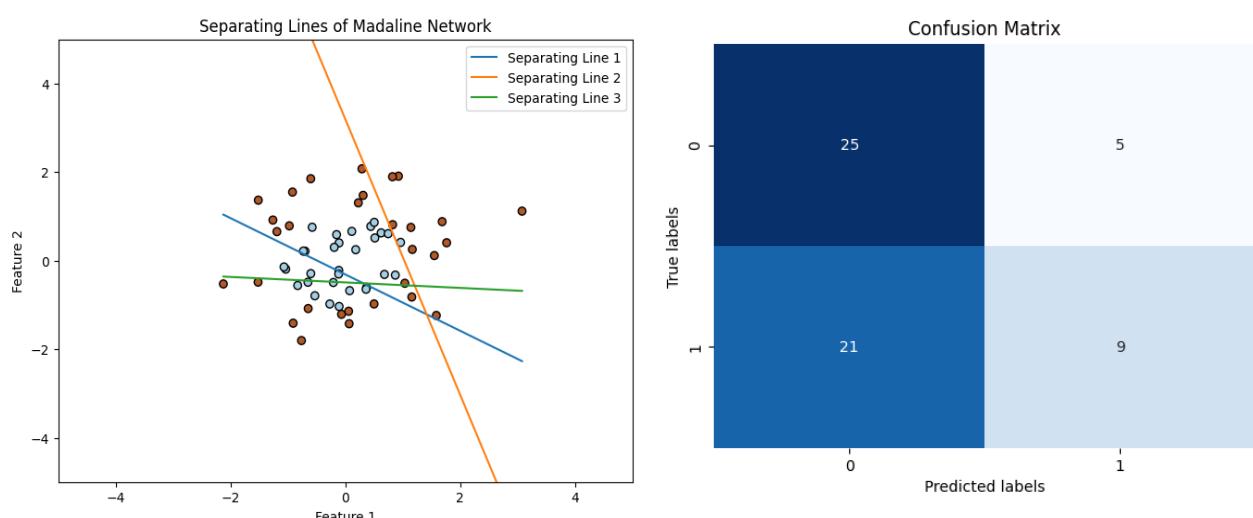
learning rate: 0.01, epochs = 200 •

خطوط جدا کننده بدست آمده توسط شبکه با این مقادیر به صورت زیر است. همانطور که نسبت به حالت قبل قابل مشاهده است، با افزایش learning rate خط سوم نیز در نزدیکی دادهها است اما دقیق آن روی دادههای تست باید بررسی شود.



شکل 28. نمودار پراکندگی و خطوط جدا کننده

شکل های زیر نتیجه هی اعمال شبکه روی این مقادیر پارامترها است.

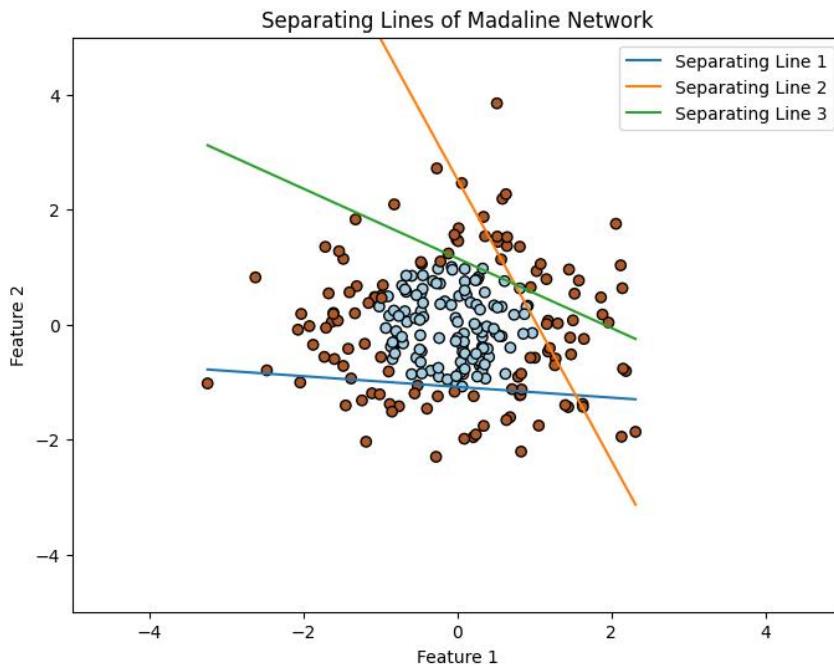


شکل 29. نمودار پراکندگی و confusion matrix برای دادههای تست با epoch=200, lr=0.01

همانطور که در نوٹ بوک نیز مشخص است، با این مقادیر، دقت مدل روی داده‌های تست 57 درصد است و افزایش learning rate منجر به کاهش accuracy شده است.

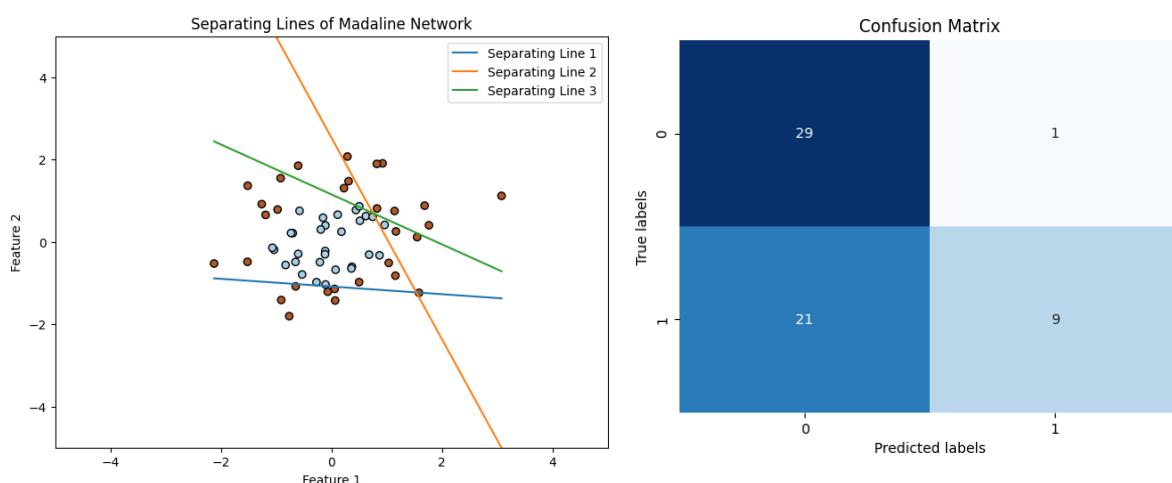
learning rate: 0.0001, epochs = 200 •

در این قسمت، مدل را برای learning rate کوچکتر بررسی می‌کنیم.



شکل 30. نمودار پراکندگی داده‌ها به همراه سه خط جدا کننده

فرم کلی شکل بالا به نسبت دو حالت قبل بیانگر افزایش دقت مدل با این پارامتر هاست. نتایج پیش‌بینی مدل روی داده‌های تست به صورت زیر است:

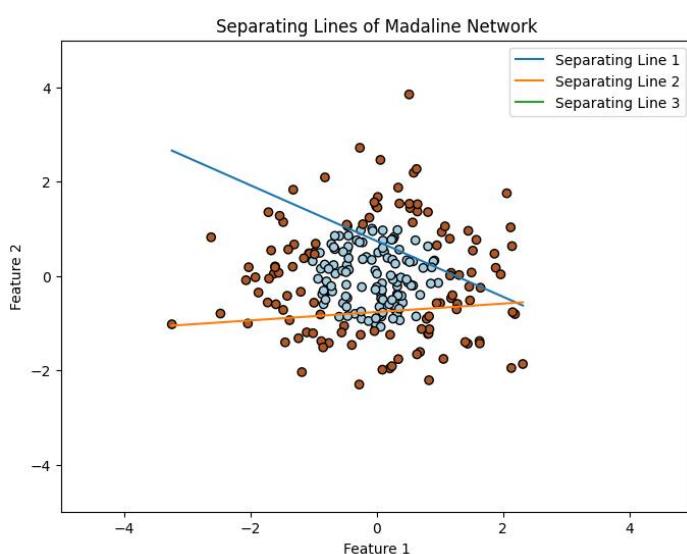


شکل 31. نمودار پراکندگی و confusion matrix برای داده‌های تست با epoch=200, lr=0.0001

دقت مدل روی داده‌های تست با این مقادیر برای پارامترها، حدود 63.3 درصد است. یعنی کاهش learning rate نسبت به قبل، تاثیر خوبی در افزایش دقیقت مدل دارد. بنابراین، مقدار 0.0001 را برای learning rate در نظر می‌گیریم و از این پس برای سه نورون، تعداد ایپاک‌ها را تغییر می‌دهیم تا ببینیم آیا عملکرد مدل بهتر می‌شود یا خیر.

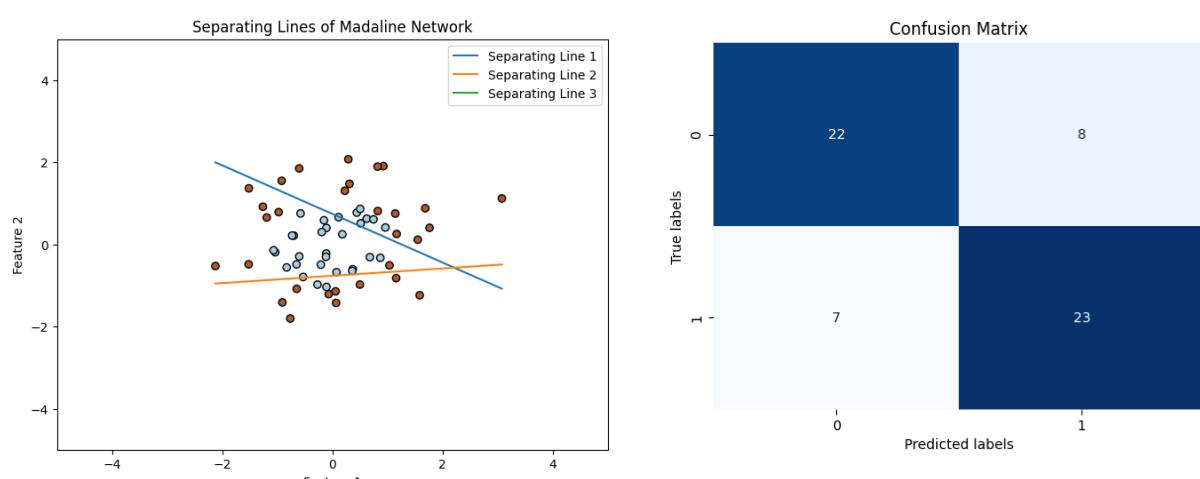
learning rate: 0.001, epochs = 400 •

عملکرد شبکه روی داده‌های آموزش به شکل زیر است:



شکل 32. نمودار پراکندگی داده‌ها و خطوط جدا کننده

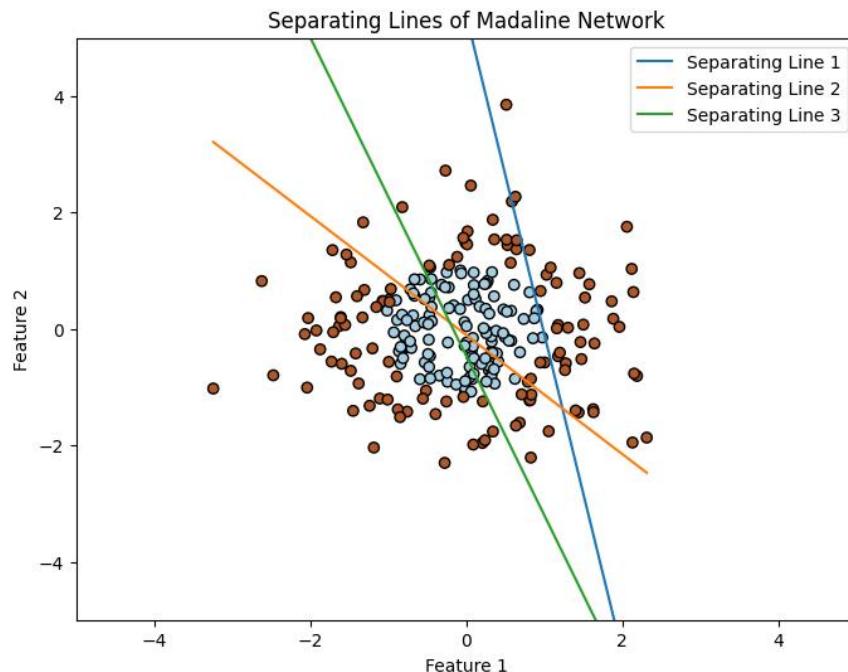
با بررسی شبکه روی داده‌های تست، شکل زیر بدست می‌آید و مشخص می‌شود که دقیقت مدل به حدود 75 درصد می‌رسد.



شکل 33. نمودار پراکندگی داده‌های تست و confusion matrix برای epoch=400, lr=0.001

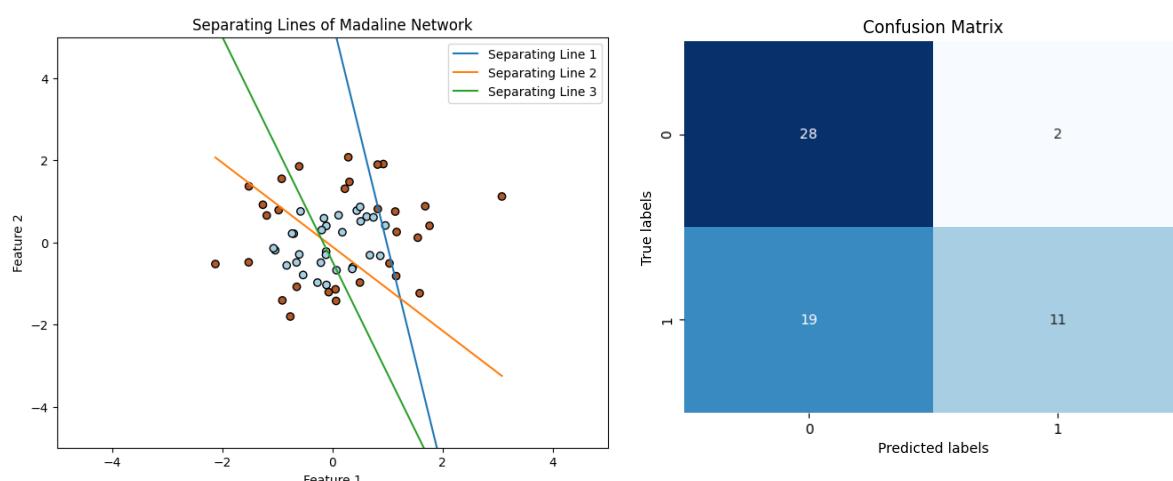
learning rate: 0.001, epochs = 1000 •

شکل زیر نمایانگر آموزش شبکه با این مقادیر پارامترها است. همانطور که در شکل زیر پیداست، نمی‌توان انتظار عملکرد بهتری از شبکه داشت.



شکل 34. نمودار پراکندگی داده‌ها و خطوط جدا کننده

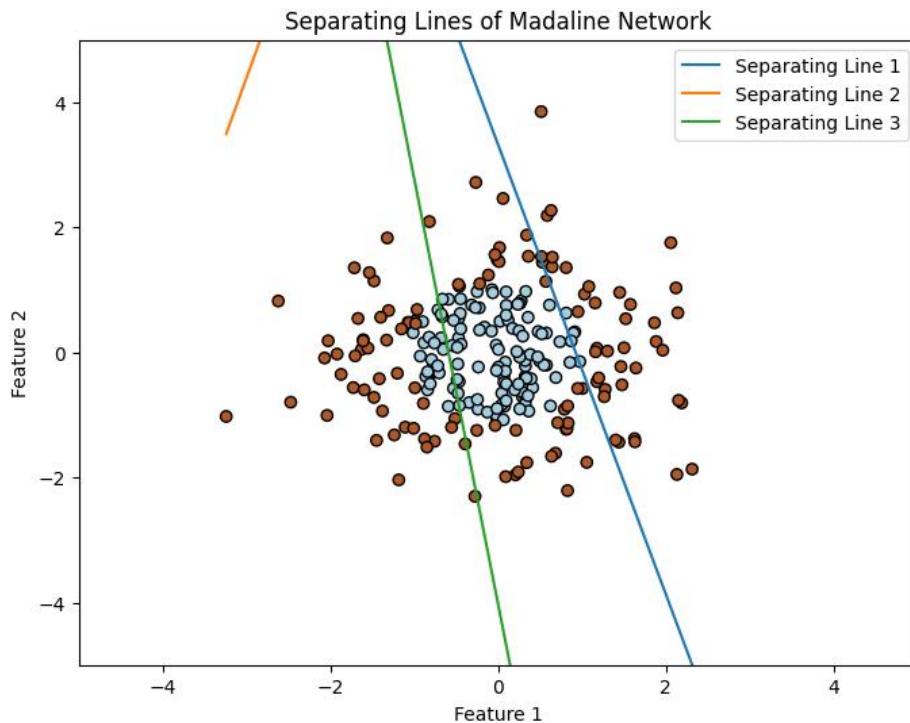
شکل زیر نتیجه‌ی اعمال شبکه‌ی آموزش دیده با این مقادیر پارامترها را دارد. دقیق برای داده‌های تست در این حالت برابر 65 درصد است و همانطور که انتظار می‌رفت، دقیق مدل نه تنها زیاد نشده، که کاهش نیز یافته است.



شکل 35. نمودار پراکندگی و confusion matrix برای داده‌های تست با epoch=1000, lr=0.001

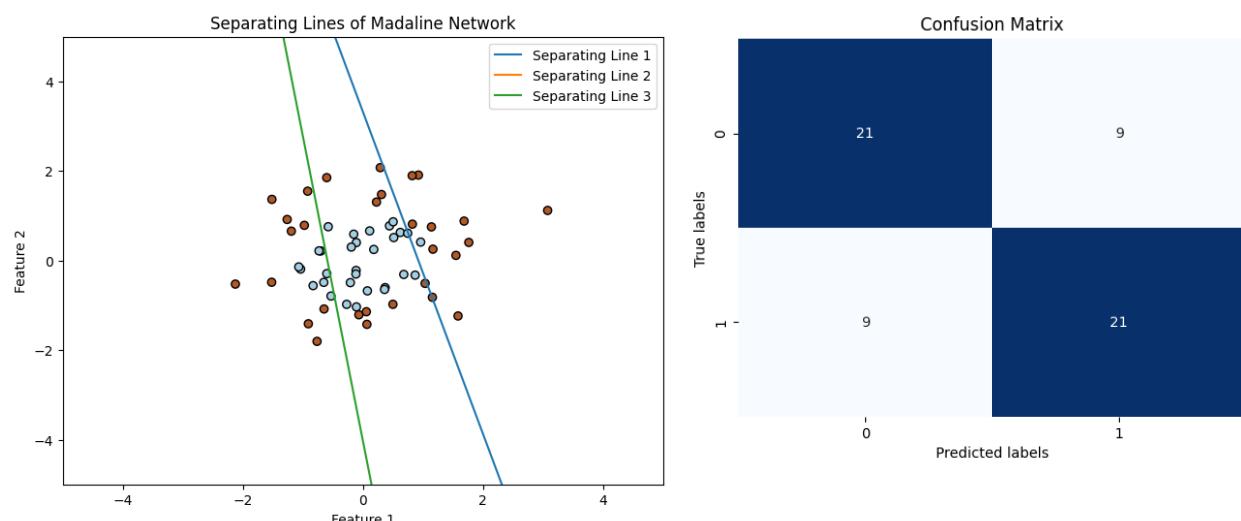
learning rate: 0.001, epochs = 2000 •

در این حالت نیز دقت برای شبکه با ۳ جدا کننده برای داده های تست برابر ۷۰ درصد است که نمودارهای آن را برای کل داده ها و داده های تست می بینیم.



شکل 36. نمودار پراکندگی داده ها و خطوط جدا کننده

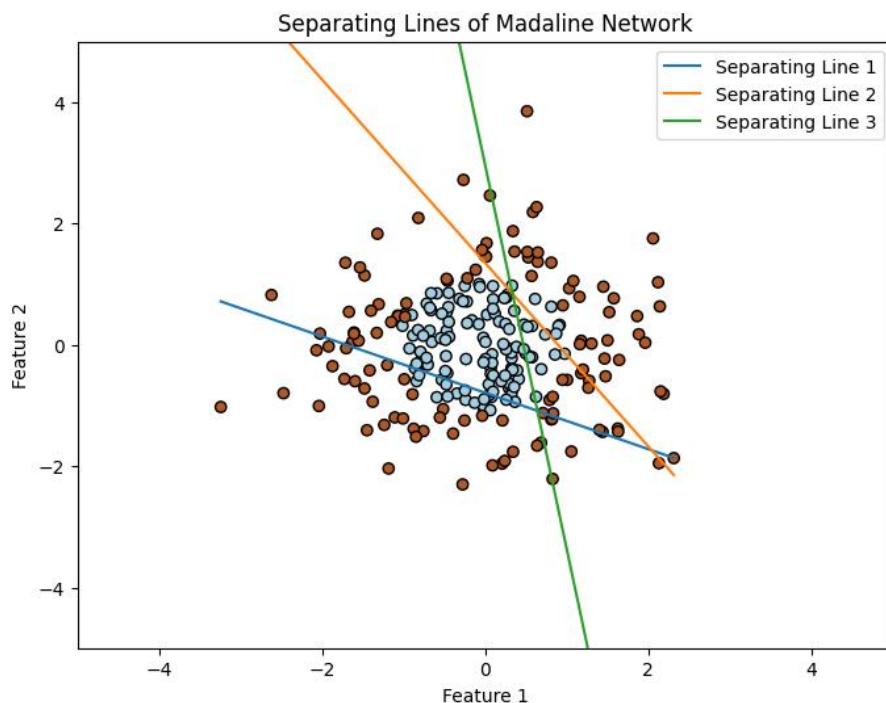
همچنین نتایج آن برای داده های تست به این صورت است.



شکل 37. نمودار پراکندگی و confusion matrix برای داده های تست epoch=2000, lr=0.001

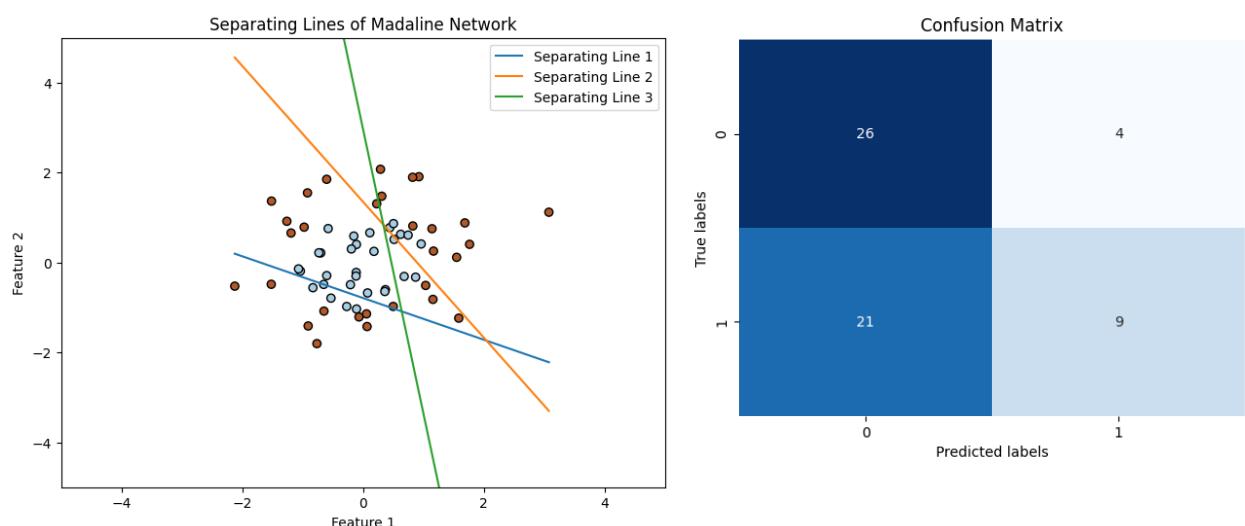
learning rate: 0.001, epochs = 3000 •

در این حالت نیز دقت شبکه با ۳ جدا کننده برای داده های تست برابر ۵۸.۳ درصد است که نمودارهای آن را برای کل داده ها و داده های تست می بینیم.



شکل 38. نمودار پراکندگی داده ها و خطوط جدا کننده

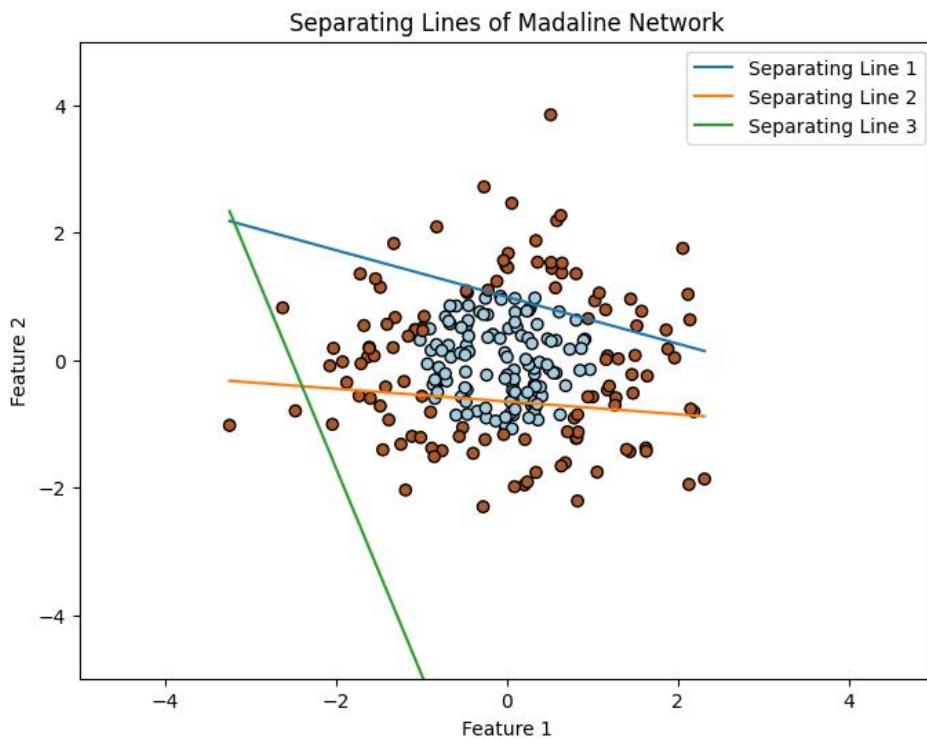
همچنین نتایج آن برای داده های تست به این صورت است.



شکل 39. نمودار پراکندگی و confusion matrix برای داده های تست epoch=3000, lr=0.001

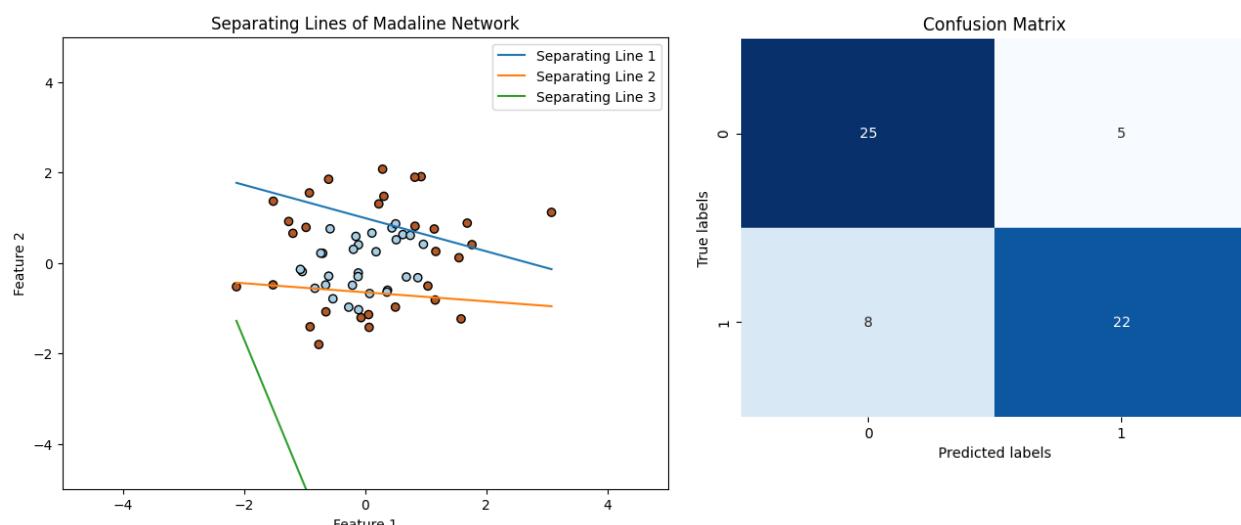
learning rate: 0.001, epochs = 5000 •

در این حالت نیز دقت برای شبکه با ۳ جدا کننده برای داده های تست برابر ۷۸.۳ درصد است که نمودارهای آن را برای کل داده ها و داده های تست می بینیم.



شکل 40. نمودار پراکندگی داده ها و خطوط جدا کننده

همچنین نتایج آن برای داده های تست به این صورت است.



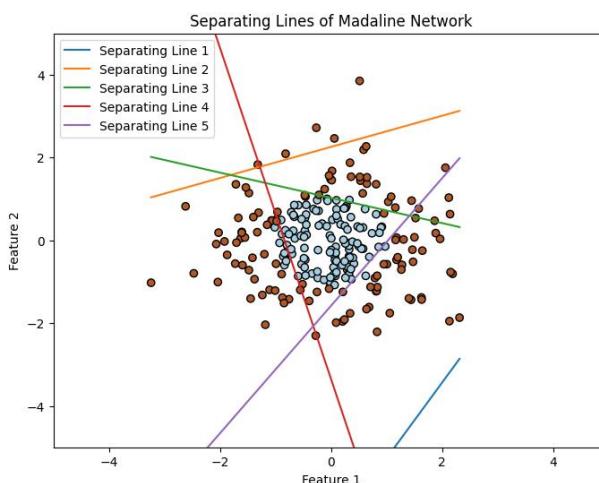
شکل 41. نمودار پراکندگی و confusion matrix برای داده های تست epoch=5000, lr=0.001

بنابراین، می‌توان نتیجه گرفت که با ثابت بودن learning rate و افزایش تعداد ایپاکها، دقت مدل به صورت یکنواخت، تغییر نمی‌کند و بهترین نتایج متعلق به حالتی است که learning rate و تعداد ایپاکها را به ترتیب 0.001 و 5000 تنظیم می‌کنیم چرا که بهترین نتایج را برای این داده‌ها به ما داده‌اند.

در مرحله‌ی بعدی، تعداد نورون‌ها را 5 ست می‌کنیم و مجدداً به تحلیل می‌پردازیم.

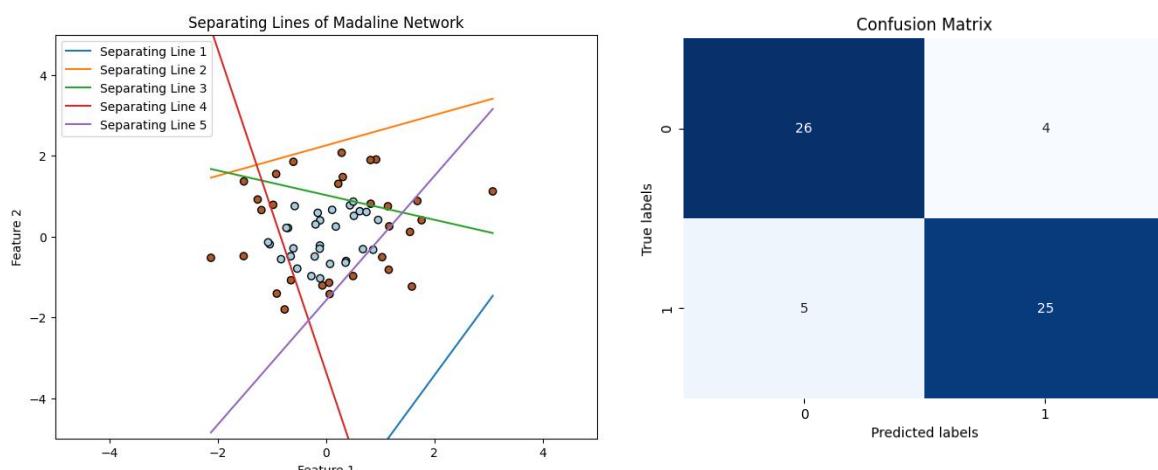
learning rate: 0.001, epochs: 200 •

همانطور که مشخص است، همه‌ی خطوط به جز خط آبی رنگ عملکرد نسبتاً خوبی روی جداسازی داده‌ها دارند.



شکل 42. نمودار پراکندگی داده‌ها و خطوط جدا کننده

با بررسی مدل با این پارامترها روی داده‌های تست نتایج زیر به دست می‌آید و مشخص می‌شود که دقت در این حالت برابر با 85 درصد است.

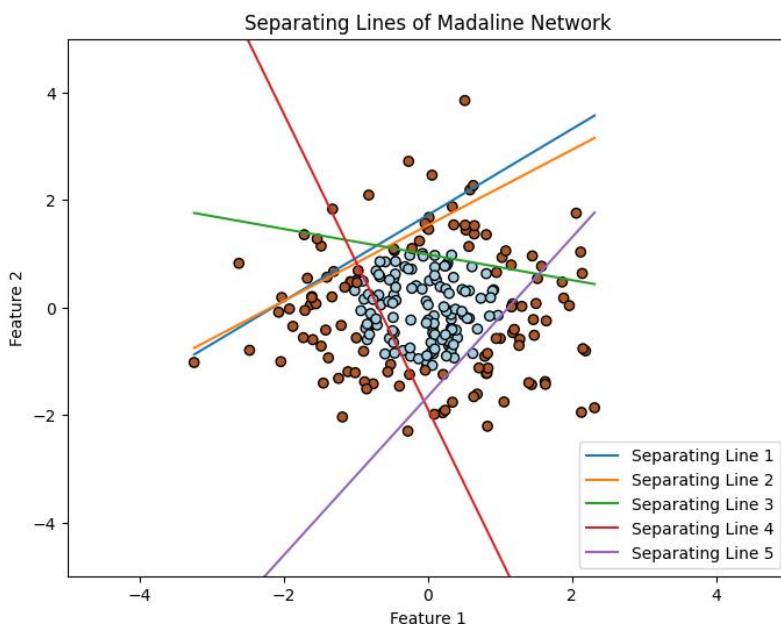


شکل 43. نمودار پراکندگی و confusion matrix برای داده‌های تست epoch=200, lr=0.01

learning rate: 0.001, epochs: 100 •

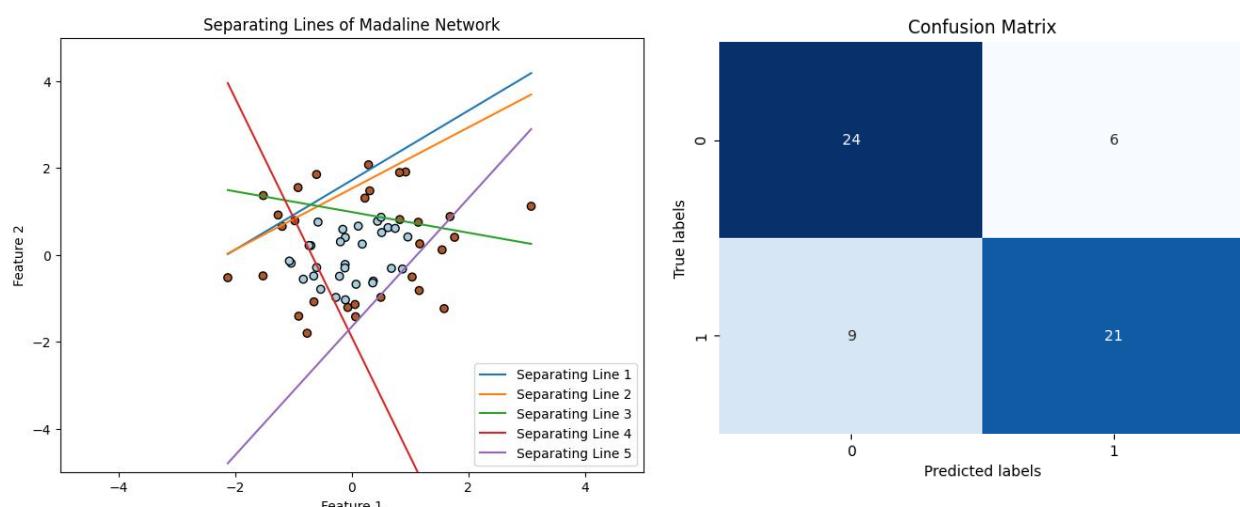
در این حالت تعداد ایپاک ها را کاهش داده ایم و عملکرد مدل روی داده های آموزش به

شکل زیر است:



شکل 44. نمودار پراکندگی داده ها و خطوط جدا کننده

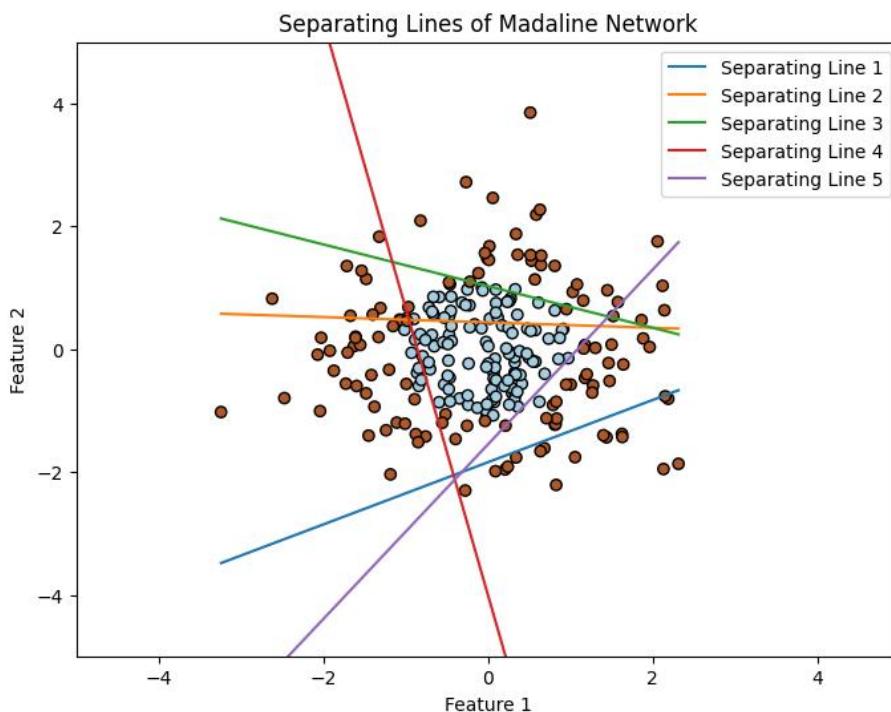
با بررسی مدل آموزش دیده با این پارامتر ها روی داده های تست به نتایج زیر دست پیدا می کنیم و دقت برابر 75 درصد خواهد بود. بنابراین کاهش تعداد ایپاک ها منجر به بهبود عملکرد شبکه نمی شود.



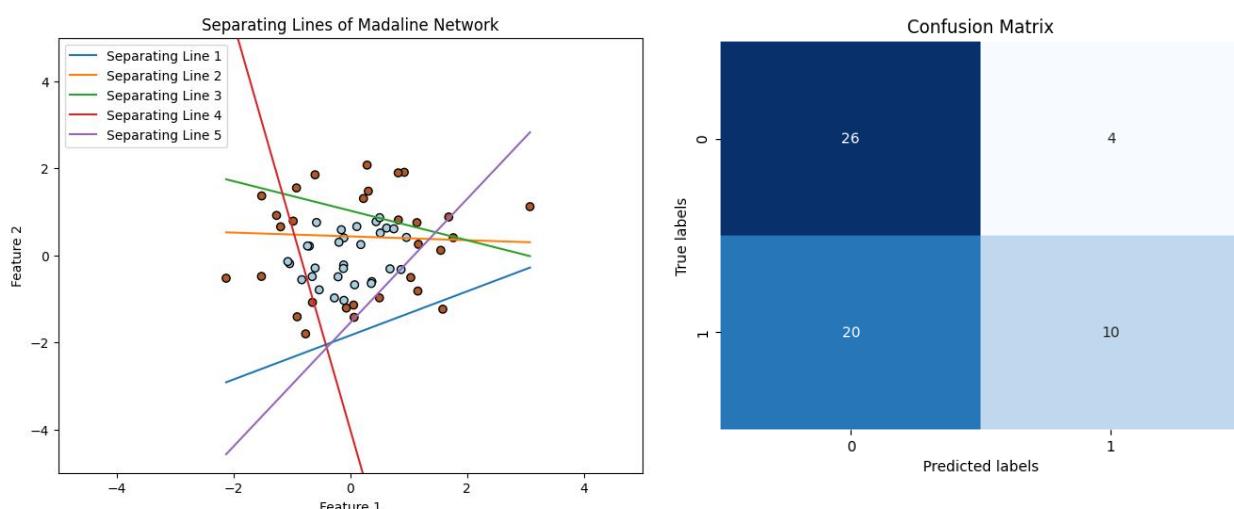
شکل 45. نمودار پراکندگی و confusion matrix برای داده های تست epoch=100, lr=0.001

learning rate: 0.001, epochs: 300 •

در این حالت تعداد ایپاک ها را افزایش داده ایم . نتایج روی داده های آموزش و تست به شکل زیر است. دقیق شبکه روی داده های تست در این حالت برابر 60 درصد است و همانطور که مشخص است، افزایش تعداد ایپاک ها نیز منجر به کاهش دقیق شبکه روی داده ها میشود.



شکل 46. نمودار پراکندگی داده ها و خطوط جدا کننده

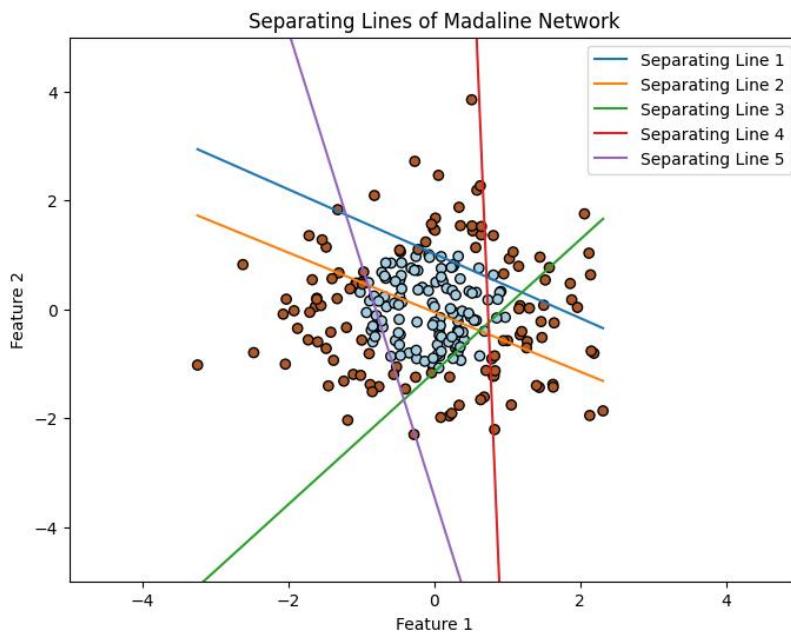


شکل 47. نمودار پراکندگی و confusion matrix برای داده های تست epoch=300, lr=0.001

بنابراین، تا اینجای کار بهترین نتایج زمانی حاصل می‌شود که تعداد ایپاک‌ها که متغیر بود، برابر 200 باشد.
حال به تغییر learning rate می‌پردازیم.

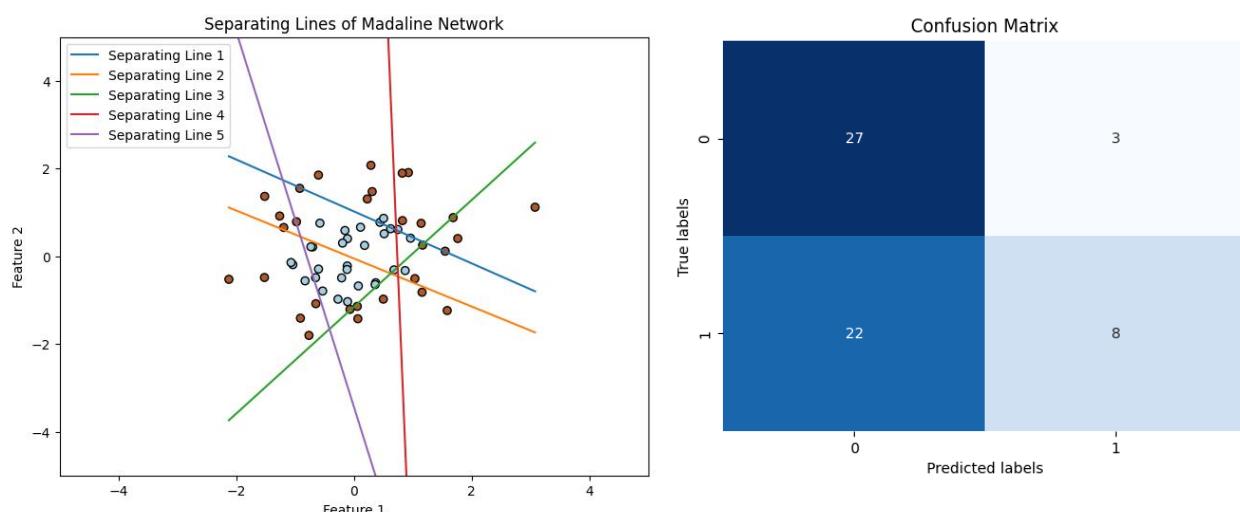
learning rate: 0.01, epochs: 200 •

با افزایش ضریب یادگیری، عملکرد شبکه روی داده‌های تست به شکل زیر است.



شکل 48. نمودار پراکندگی داده‌ها و خطوط جدا کننده

همانطور که از نتایج اعمال شبکه روی داده‌های تست در زیر مشخص است، در این حالت دقیق
مدل کاهش یافته و برابر 58 می‌باشد.

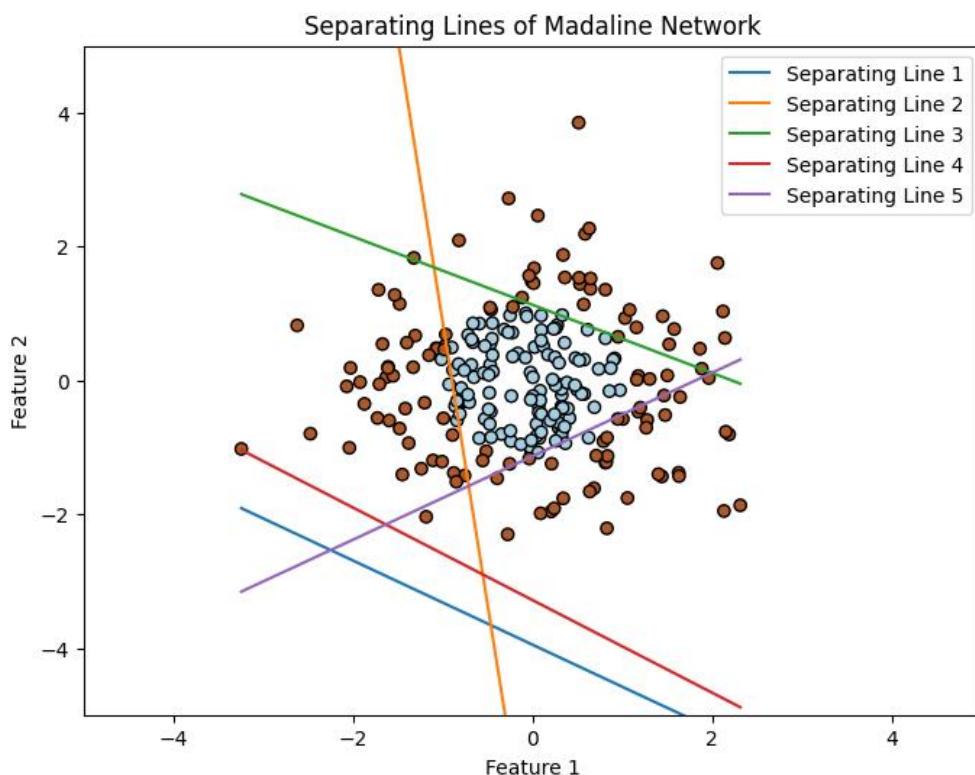


شکل 49. نمودار پراکندگی و confusion matrix برای داده‌های تست epoch=200, lr=0.01

learning rate: 0.0001, epochs: 200 •

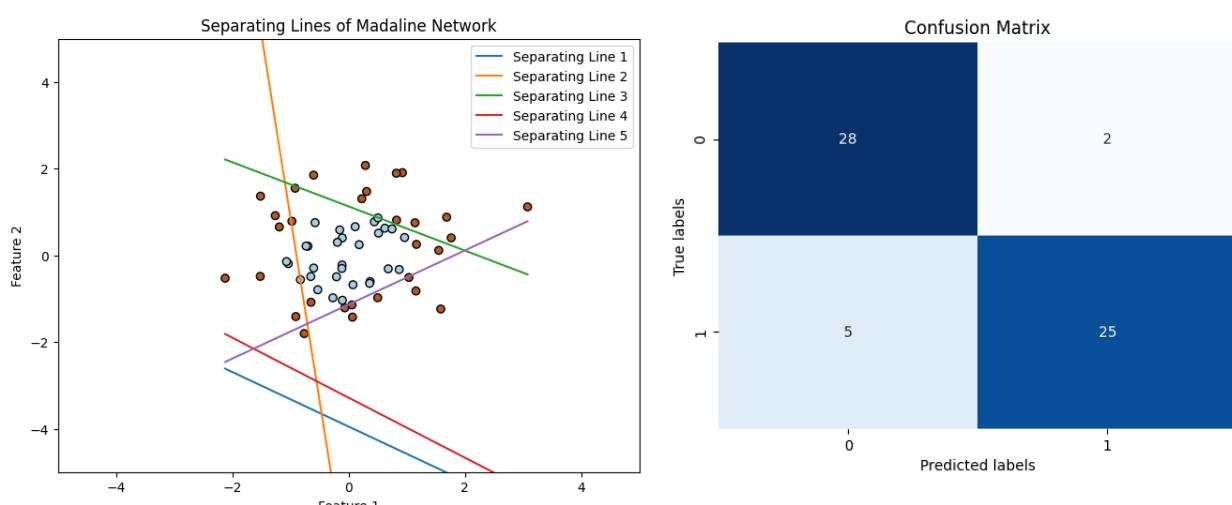
حال به کاهش ضریب یادگیری می‌پردازیم . مدل را با پارامترهای جدید آموزش می‌دهیم. نتایج

آن به صورت زیر است:



شکل 50. نمودار پراکندگی داده‌ها و خطوط جدا کننده

اکنون، این مدل را روی داده‌ای تست بررسی می کنیم:



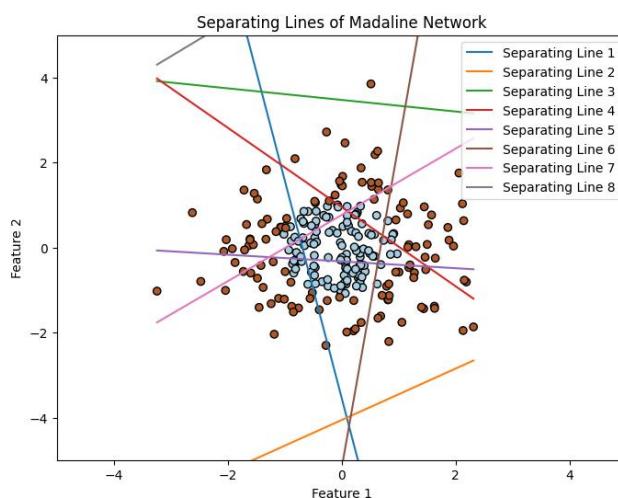
شکل 51. نمودار پراکندگی و confusion matrix برای داده‌ای تست epoch=200, lr=0.0001

از شکل های فوق پیداست که کاهش ضریب یادگیری منجر به افزایش دقت مدل شده و دقت آن روی داده های تست حدود 88.3 درصد است که نسبت به قبل رشد داشته است. بنابراین، با 5 نورون، کاهش و افزایش تعداد ایپاک ها و همچنین افزایش ضریب یادگیری منجر به کاهش دقت مدل شد، در حالی که کاهش ضریب یادگیری باعث افزایش دقت شده است. پس بهترین مقادیر برای پارامتر ها با 5 نورون به صورت ضریب یادگیری 0.0001 و learning rate 0.0001 است.

اکنون حالاتی را بررسی می کنیم که تعداد نورون ها برابر 8 باشد.

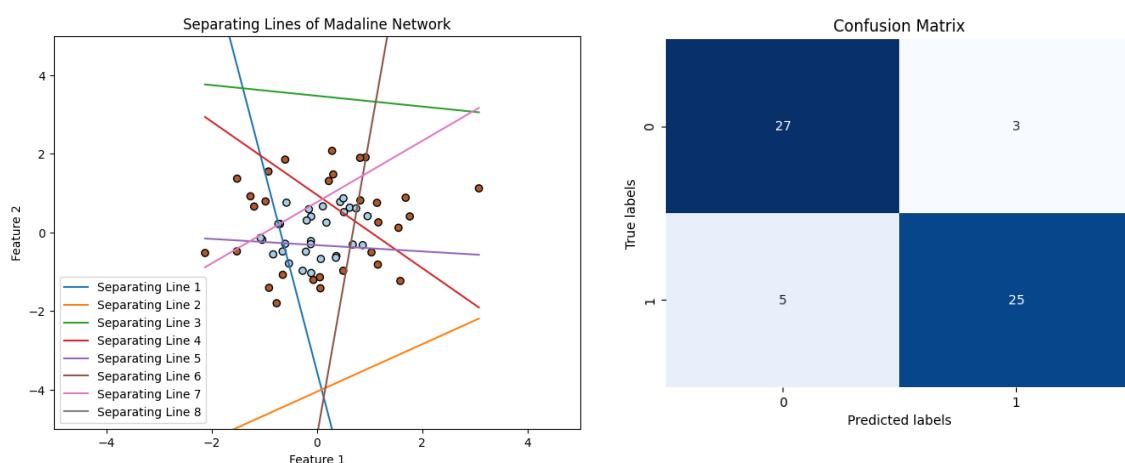
learning rate: 0.001, epochs: 200 •

در این حالت، عملکرد مدل روی داده های آموزش به شکل زیر است:



شکل 52. نمودار پراکندگی داده ها و خطوط جدا کننده

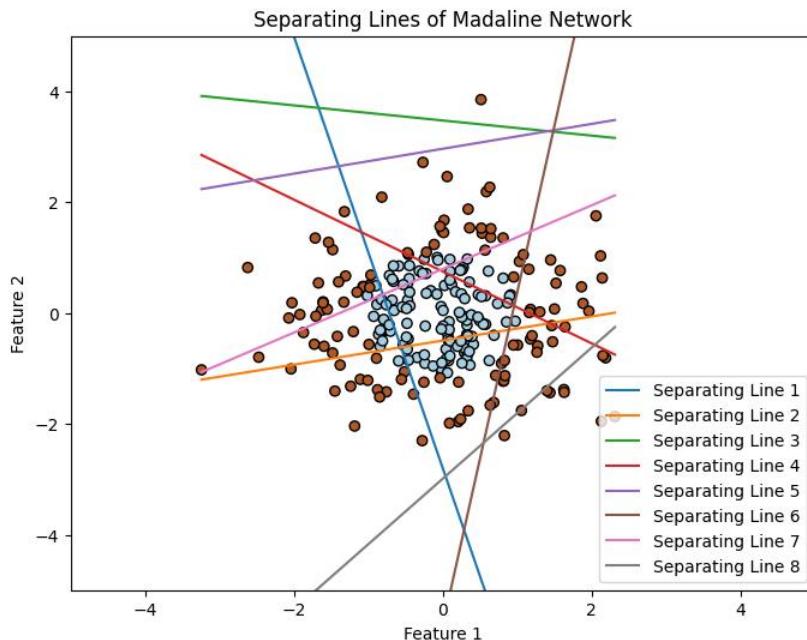
اعمال شبکه‌ی آموزش دیده روی داده های تست مشخص میشود که دقت آن حدود ۸۷ درصد است.



شکل 53. نمودار پراکندگی و confusion matrix برای داده های تست epoch=200, lr=0.001

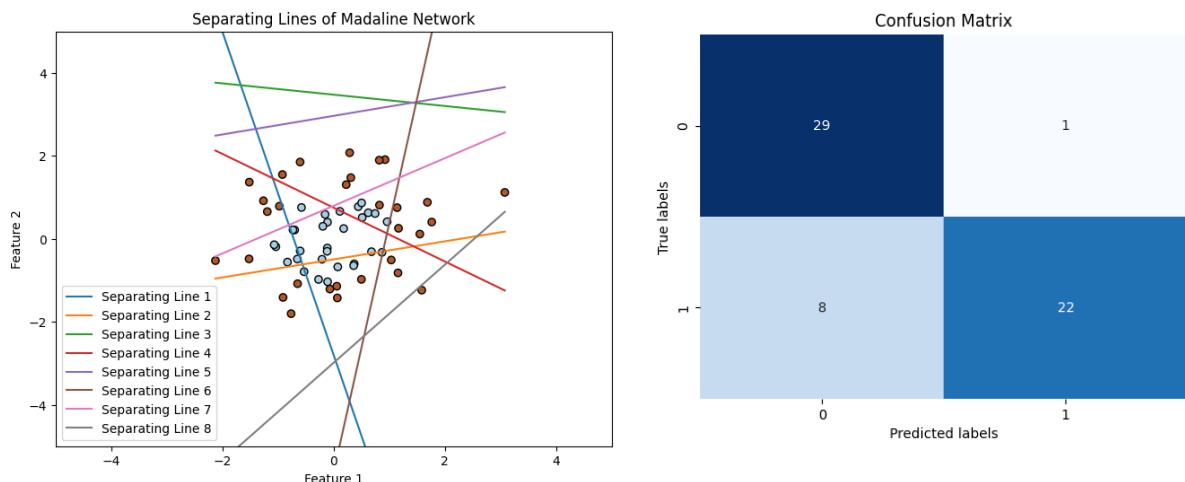
learning rate: 0.001, epochs: 300 •

در این حالت تعداد ایپاک ها را افزایش داده ایم تا تاثیر آن را بررسی کنیم. شکل زیر نتیجه‌ی آموزش شبکه روی داده های آموزشی است:



شکل 54. نمودار پراکندگی داده‌ها و خطوط جدا کننده

با اعمال شبکه روی داده های تست به نتایج زیر میرسیم.

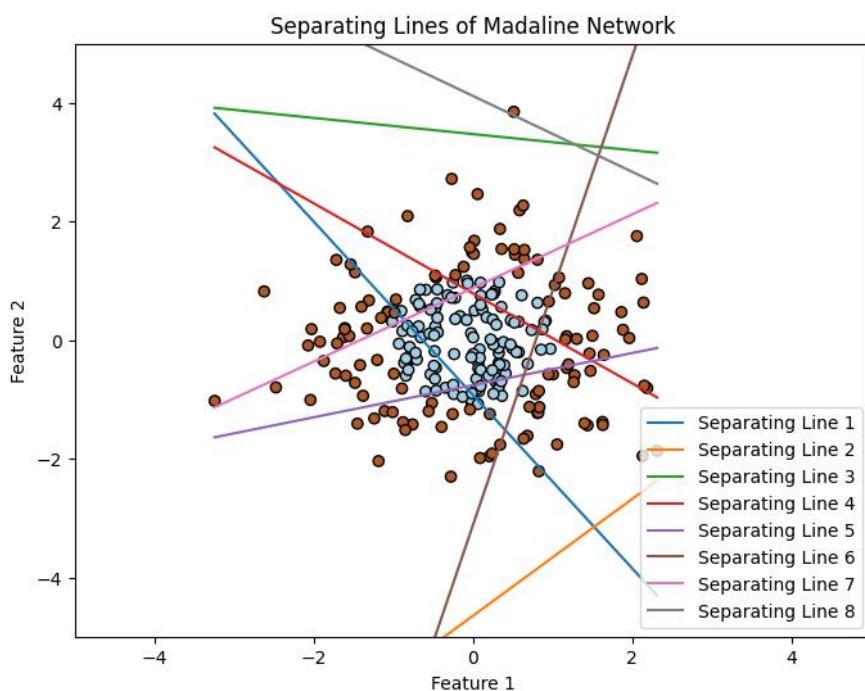


شکل 55. نمودار پراکندگی و confusion matrix برای داده‌های تست epoch=300, lr=0.001

دقت شبکه روی داده های تست برابر ۸۵ درصد است و نسب به حالت قبل تنها ۳ درصد کاهش داشته است.

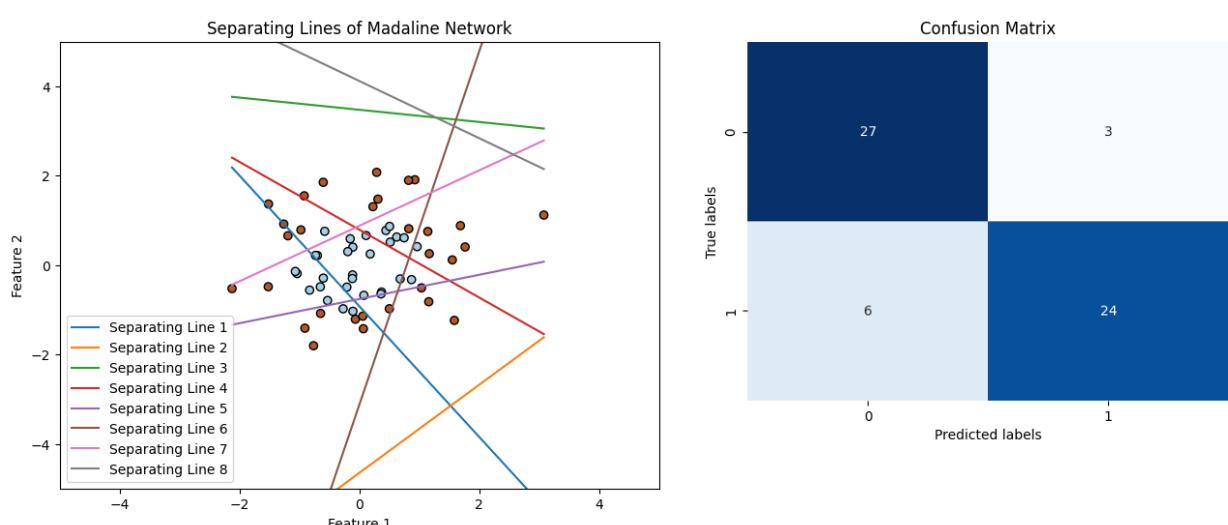
learning rate: 0.001, epochs: 100 •

با کاهش تعداد ایپاک ها و آموزش مدل، خطوط جدا کننده به صورت زیر در می آیند.



شکل 56. نمودار پراکندگی داده ها و خطوط جدا کننده

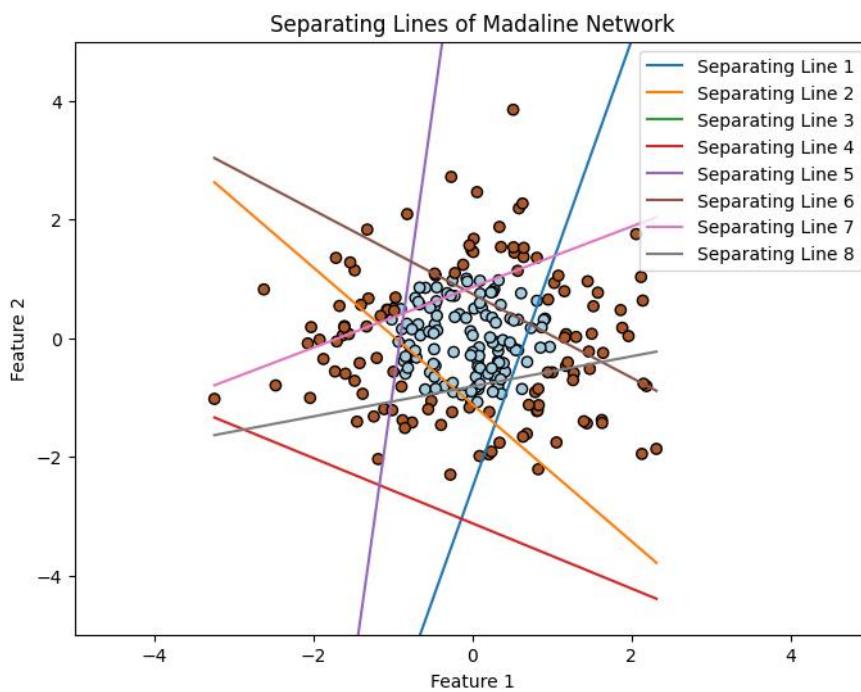
نتایج روی داده های تست تیز به صورت زیر است و دقت شبکه برابر ۸۵ درصد است. بنابراین با کاهش تعداد ایپاک ها نیز حدود ۳ درصد دقت کاهش داشته است. پس تا اینجا تعداد ایپاک ها را همان ۲۰۰ در نظر میگیریم . به تغییر learning rate می پردازیم.



شکل 57. نمودار پراکندگی و confusion matrix برای داده های تست epoch=100, lr=0.001

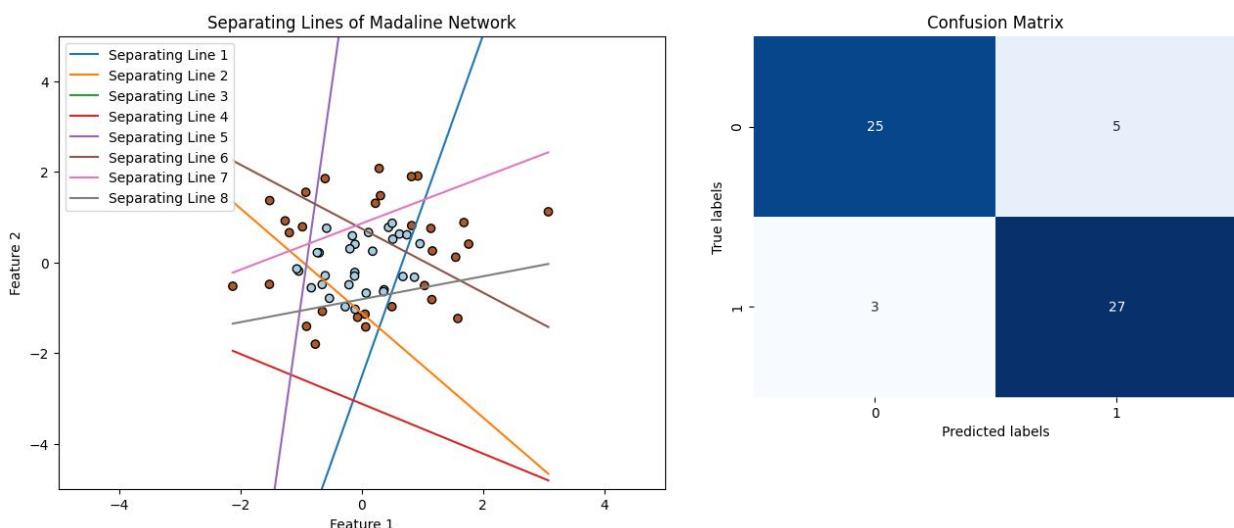
learning rate: 0.01, epochs: 200 •

با کاهش ضریب یادگیری و آموزش مدل خطوط زیر حاصل می‌شوند:



شکل 58. نمودار پراکندگی داده‌ها و خطوط جدا کننده

نتایج اعمال شبکه روی داده‌های تست نیز به صورت زیر است و دقت برابر 87 درصد است.

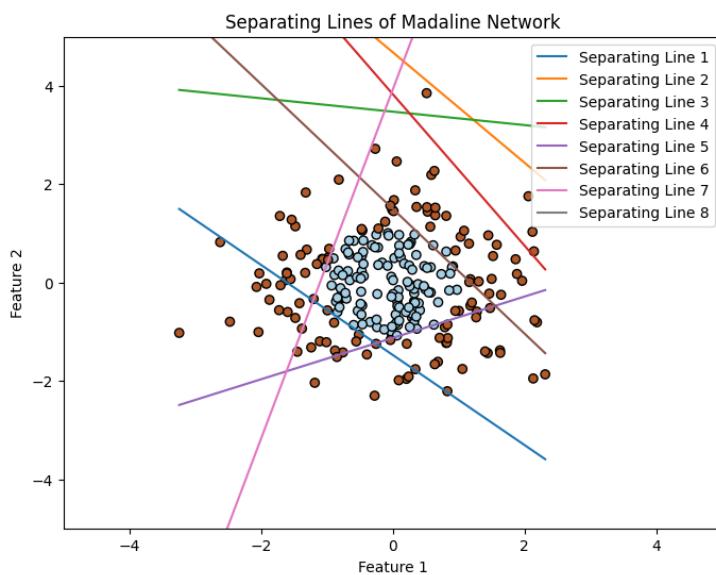


شکل 59. نمودار پراکندگی و confusion matrix برای داده‌های تست epoch=200, lr=0.01

همانطور که واضح است، افزایش ضریب یادگیری تاثیری در دقت مدل نداشت.

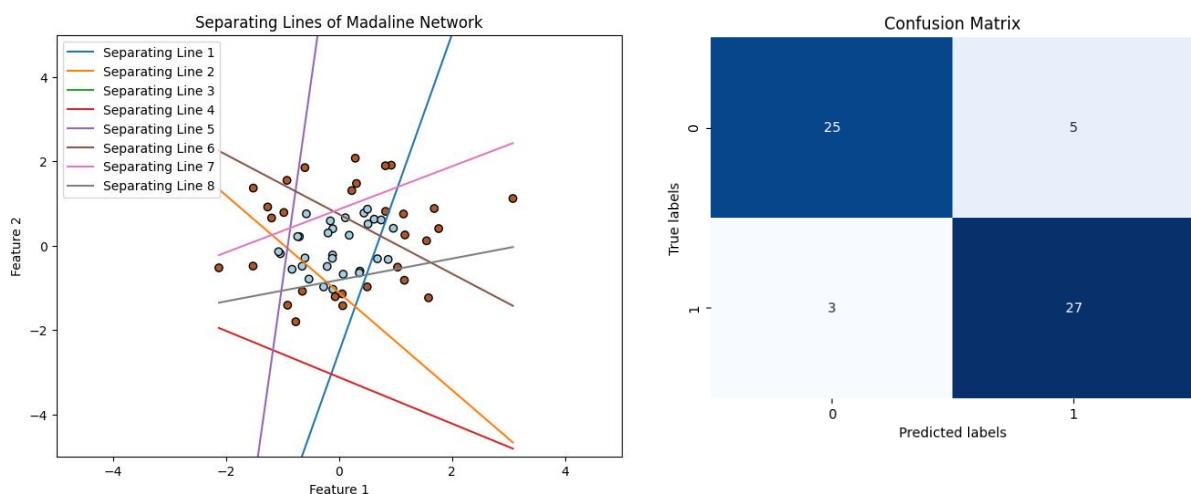
learning rate: 0.0001, epochs: 200 •

با کاهش ضریب یادگیری و آموزش مدل، نتایج زیر روی داده های آموزشی بدست می آید:



شکل 60. نمودار پراکندگی داده ها و خطوط جدا کننده

با اعمال شبکه روی داده های تست نیز نتایج زیر حاصل میشود:



شکل 61. نمودار پراکندگی و confusion matrix برای داده های تست epoch=200, lr=0.0001

از نتایج بالا می توان نتیجه گرفت که کاهش نرخ یادگیری باعث افزایش دقت مدل شده و آن را به درصد می رساند. بنابراین با استفاده از 8 نورون میتوان در ۹۲ درصد موقع داده ها را به درستی طبقه بندی کرد.

همانطور که از بررسی های فوق واضح است و بنا به انتظار، با افزایش تعداد نورون ها دقت مدل افزایش می یابد و داده ها بهتر طبقه بندی می شوند.

۱-۱. رگرسن

الف) برازش بیش از حد (overfitting)

برازش بیش از حد یا overfitting زمانی رخ می‌دهد که مدل و شبکه ما داده‌های آموزش را خیلی خوب یاد می‌گیرد. در واقع اگر دقیق‌تر بخواهیم بگوییم، شبکه ما، داده‌های آموزش را به جای یاد گرفتن، حفظ کرده و خاصیت تعمیم پذیری یا generalization خود را از دست می‌دهد. در این صورت، شبکه ما، داده‌های آموزش را همراه با noise را یاد می‌کند و ممکن است ویژگی‌های اصلی داده‌ها برای آموزش، اهمیت کمتری در فرآیند یادگیری پیدا کنند. برخی از دلایل برازش بیش از حد، به صورت زیر است.

- پیچیدگی زیاد مدل:

برازش بیش از حد برخی اوقات به دلیل پیچیدگی زیاد مدل نسبت به حجم داده‌های آموزش است. در واقع اگر تعداد پارامترها و پیچیدگی مدل (رابطه مستقیم پیچیدگی و تعداد پارامترها) بسیار زیاد بوده و داده‌های آموزش موجود نسبت به آن کم باشد، در این صورت پارامترهای ما فرصت بروز رسانی کافی را پیدا نخواهند کرد و در واقع مدل تنها داده‌ها را حفظ کرده و نویزها و اطلاعات کم اهمیت را نیز استفاده می‌کند. بنابراین در این حالت تعداد لایه‌ها و پارامترهای شبکه احتمالاً زیاد است. لازم به ذکر است که در این حالت به دلیل عدم کفايت حجم داده در دسترس برای آموزش کافی مدل، ممکن است علاوه بر نویز، مقادیر اولیه داده شده به وزن‌ها نیز به روز نشده و در پیش‌بینی نهایی مدل تاثیر داشته باشند و ویژگی‌های اصلی استخراج نشوند.

- نامناسب بودن داده آموزش:

اگر حجم داده‌های آموزش ما نسبت به پیچیدگی مدل کم بوده و همچنین توزیع داده‌های آموزش و تست یکی نباشند و یا کلاس‌های موجود در دیتاست، balance نباشند، در این صورت مدل تنها داده‌های آموزش را حفظ کرده خاصیت یادگیری نخواهد داشت.

- ویژگی‌های نامرتب در دیتاست:

اگر ویژگی‌های موجود در دیتاست، correlation ای با یکدیگر نداشته باشند، در این صورت مدل داده‌های آموزش را حفظ کرده و خاصیت تعمیم پذیری را از دست می‌دهد.

تعداد زیاد ایپاک:

اگر آموزش با تعداد بالای ایپاک اجرا شود (دیده شدن بیش از حد داده آموزش)، در این صورت مدل شروع به حفظ کردن داده‌ها شده و از تعمیم پذیری برای داده‌های خارج از آن، دور می‌شود.

ب) راه حل رفع overfitting

برای جلوگیری از overfitting، روش‌های مختلفی وجود دارد. در ادامه به بررسی برخی از آن‌ها می‌پردازیم.

Data augmentation •

می‌توانیم داده‌های آموزش را به طرق مختلف افزایش دهیم. برای مثال با اضافه کردن نویز، چرخاندن یا rescale کردن داده‌ها و این کار باعث افزایش خاصیت تعمیم پذیری مدل ما می‌شود.

انتخاب ویژگی:

همانگونه که گفتیم، وجود ویژگی‌های نامربوط در دیتاست ممکن است باعث برآش بیش از حد بشود. بنابراین با کاهش ویژگی‌ها با استفاده از روش‌های مختلف، احتمالاً بتوانیم مشکل برآش بیش از حد شبکه را برطرف کنیم.

مدل ساده‌تر:

می‌توانیم از مدلی که پیچیدگی و تعداد پارامترهای کمتری دارد استفاده کنیم.

استفاده از Validation set:

می‌توانیم با استفاده از validation set، شروع overfitting را تشخیص داده و دیگر فرآیند آموزش را ادامه ندهیم.

پیش پردازش داده‌ها:

با انجام preprocessing بر روی داده‌ها، و نرمالایز کردن و balance کردن دیتا، وزن‌های مدل بهتر بروز شده و فقط به یک سری از وزن‌ها توجه نمی‌شود.

پ) هایپرپارامتر

در شبکه‌های عصبی، هایپرپارامترها، پارامترهایی هستند که مستقل از داده‌های آموزش هستند و نیاز به تنظیم دستی دارند تا عملکرد بهتری برای مدل به دست آید. برخی از مهم‌ترین هایپرپارامترها عبارتند از:

Learning rate . 1

این پارامتر مشخص می‌کند که در هر مرحله از فرآیند یادگیری، چه مقداری از گرادیان را برای بروزرسانی وزن‌ها استفاده کنیم. مقدار این هایپرپارامتر می‌تواند بین 0 و 1 باشد.

:Batch size . 2

این هایپرپارامتر مشخص می‌کند که در هر مرحله از فرآیند یادگیری، چه تعداد نمونه از داده‌ها برای محاسبه گرادیان و به روزرسانی وزن‌ها استفاده شود.

3 . تعداد لایه‌ها و نورون‌ها:

تعیین تعداد لایه‌ها و نورون‌ها در هر لایه از شبکه نیز یک هایپرپارامتر است که باید با دقت تنظیم شود.

Activation function . 4

انتخاب تابع فعال‌سازی برای هر لایه نیز یک هایپرپارامتر است که می‌تواند تأثیر زیادی بر عملکرد شبکه داشته باشد.

می‌توان مقادیر برای هر کدام از این هایپرپارامترها را با استفاده از آزمون و خطا و دانش مربوط به مدل‌های شبکه عصبی و خود دیتابست، تنظیم کرد. به این عمل parameter tuning گفته می‌شود.

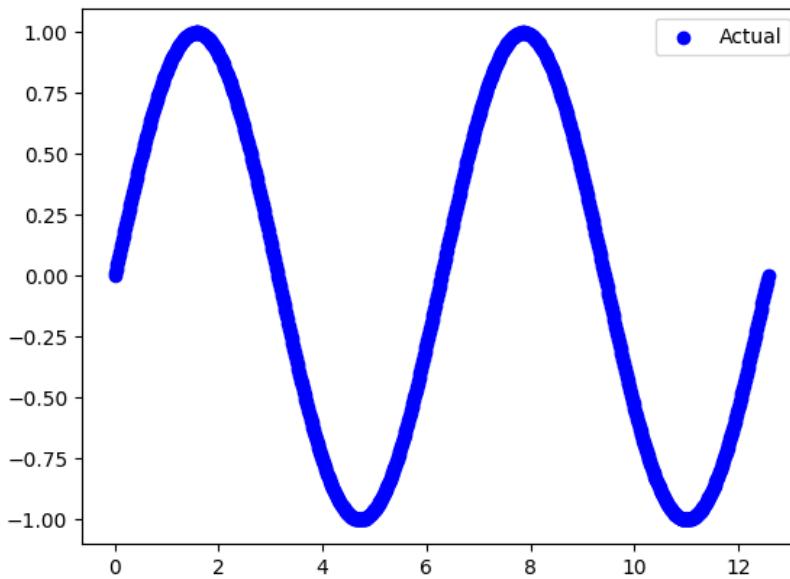
تفاوت اصلی بین هایپرپارامترها و پارامترها در شبکه‌های عصبی به این صورت است که پارامترهای شبکه، همان وزن‌های یادگرفته شده در شبکه ما هستند، در حالی که هایپرپارامترها، تعیین می‌شوند تا شبکه آموزش ببیند. به صورت تفاوت آن‌ها به صورت زیر است.

- پارامترها:
 - پارامترها مقادیری هستند که مستقیماً توسط مدل در فرآیند آموزش تغییر می‌کنند تا مدل بتواند داده‌ها را بهتر توصیف کند.
 - این مقادیر به طور اتوماتیک توسط الگوریتم‌های بهینه‌سازی (مانند روش‌های gradient descent) تنظیم می‌شوند.
 - مثال‌هایی از پارامترها شامل وزن‌ها و بایاس‌های هر لایه در شبکه‌های عصبی هستند.

- هایپرپارامترها:
 - هایپرپارامترها مقادیری هستند که خارج از مدل قرار دارند و نیاز به تنظیم دستی دارند تا عملکرد بهتری برای مدل به دست آید.
 - این مقادیر معمولاً توسط خود ما تعیین می‌شوند.
 - مثال‌هایی از هایپرپارامترها شامل learning rate، batch size، تعداد لایه‌ها و نورون‌ها، نوع تابع فعال‌سازی و ... می‌باشد.

ت) پیاده سازی تابع سینوسی و رگرسن

با استفاده از قطعه کد داده شده ۱۰۰۰ نقطه را به عنوان دیتا تولید میکنیم که به صورت زیر هستند.

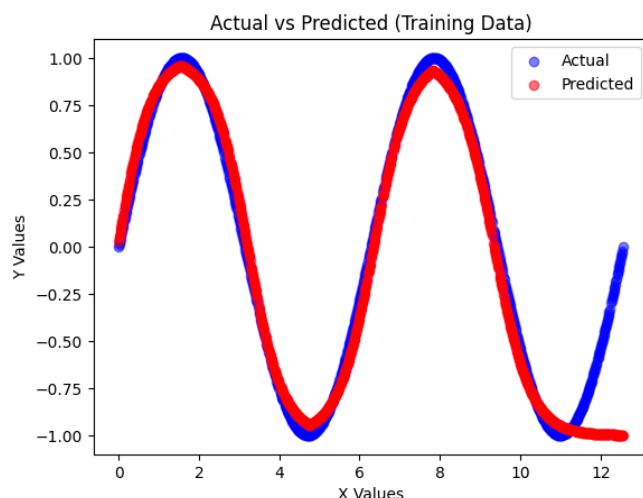


شکل 62. تابع سینوسی تولید شده

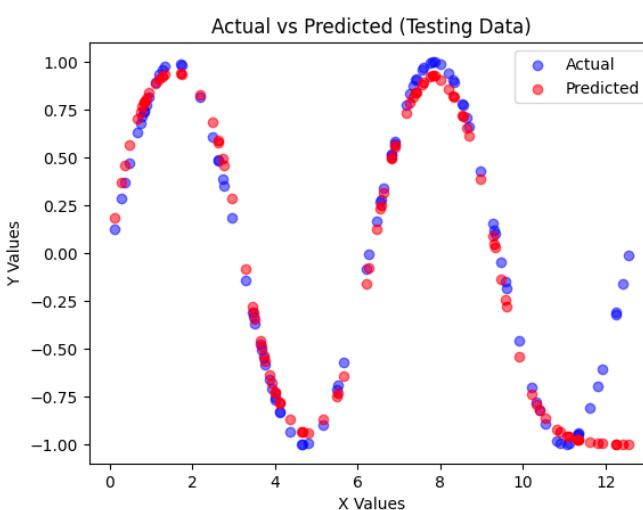
افزایش داده‌ها

برای اینکه به راحتی بتوانیم مدل را با دیتا در اندازه‌های متفاوت ساخته و تست کنیم، یک تابع مینویسیم. تابع `create_model` برای هر اندازه‌ای از `test size` که از ۰.۱ تا ۰.۹ متغیر است، یک مدل می‌سازد و روی دیتای متناظر با آن مدل را آموزش داده و تست می‌کند. در ادامه عملکرد این تابع را برای مقادیر متفاوت می‌بینیم. اما پیش از آن نیاز است درباره‌ی ساختار مدل کمی بیشتر توضیح بدهیم. قبل از ساخت مدل و استفاده از `tf.keras.Sequential` لازم است دیتا را `reshape` کنیم تا بتوان آن را برای آموزش مدل ساخته شده با این مازول استفاده کرد. حال مدل را می‌سازیم؛ مدلی که در هر بار صدا زدن این تابع ساخته می‌شود، علاوه بر لایه‌ی ورودی، یک لایه‌ی پنهان و یک لایه‌ی خروجی نیز دارد. در لایه‌ی پنهان از تابع فعال‌ساز **ReLU** استفاده کردیم که ویژگی غیرخطی شبکه را افزایش می‌دهد. همچنین، برای لایه‌ی خروجی از فعال‌ساز `tanh` استفاده کردیم چرا که در این حالت خروجی `smoothness` بیشتری دارد. همچنین، `y` در داده‌ها عددی بین -۱ و ۱ است و برای اینکه خروجی مدل نیز در این بازه قرار بگیرد، بهتر است از این تابع کمک بگیریم.

حال به بررسی عملکرد مدل روی مقادیر مختلف برای test size می‌پزداییم که نتایج حاصل، در ۱-Gif به پیوست آمده است. همانطور که مشخص است، تا زمانی که اندازه‌ی داده‌های آموزشی به اندازه‌ی کافی باشد، loss زیر ۰.۱ است و طبق نمودارهای رسم شده، مدل روی داده‌های تست به خوبی عمل کرده است. البته لازم به ذکر است که ماهیت regression که overfitting برای آن رخ می‌هد نیز قابل چشم پوشی نیست. اما با افزایش غیر منطقی سایز داده‌های تست نسبت به داده‌های آموزشی (از $test\ size = 0.7$ تا $test\ size = 0.9$ ، از آنجا که مدل به خوبی یاد نمی‌گیرد و همچنین داهدی کافی برای به خاطر سپردن ندارد، loss افزایش یافته و طبق تصاویر تولید شده، prediction روی داده‌های تست به خوبی صورت نگرفته است. شکل زیر، نمایش عملکرد مدل روی داده‌های آموزشی و تست در بهترین حالت که در آن نسبت داده‌های آموزشی و تست برابر ۰.۹ و ۰.۱ بود را نمایش می‌دهد.

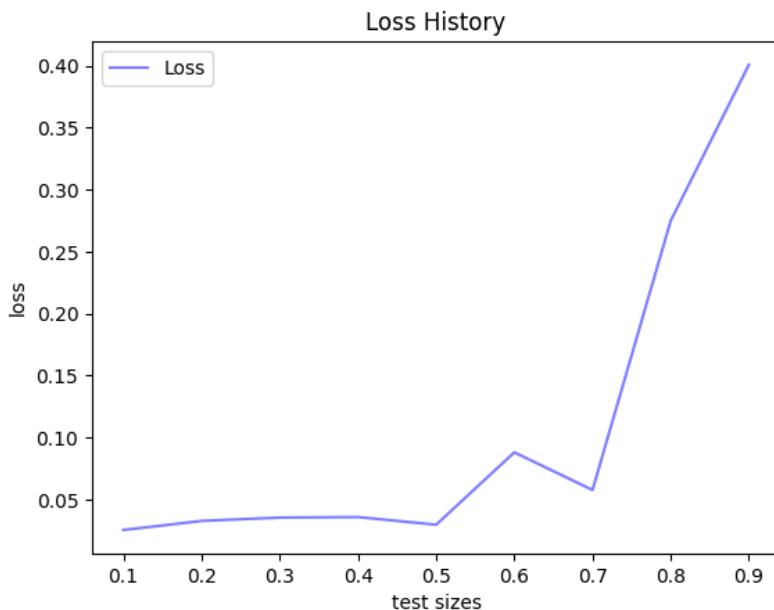


شکل ۶۳. نمودار داده‌های آموزش به همراه پیش‌بینی مدل از آن‌ها



شکل ۶۴. نمودار داده‌های تست به همراه پیش‌بینی مدل از آن‌ها

در زیر نمودار تغییرات loss روی داده‌های تست برای test size های مختلف را می‌بینیم:



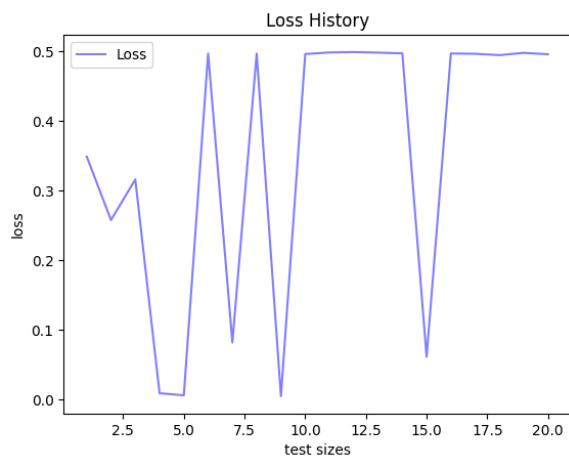
شکل 65. تغییرات loss، با تغییر test size

افزایش لایه‌ها

برای این بخش، تعداد حالات پنهان را در بازه‌ی ۱ تا ۲۰ و در دو حالت را بررسی کرده‌ایم که در هر دوی آنها برای هر لایه‌ی پنهان ۵ نورون در نظر گرفته شده و آنها را در پایان با هم مقایسه خواهیم کرد.

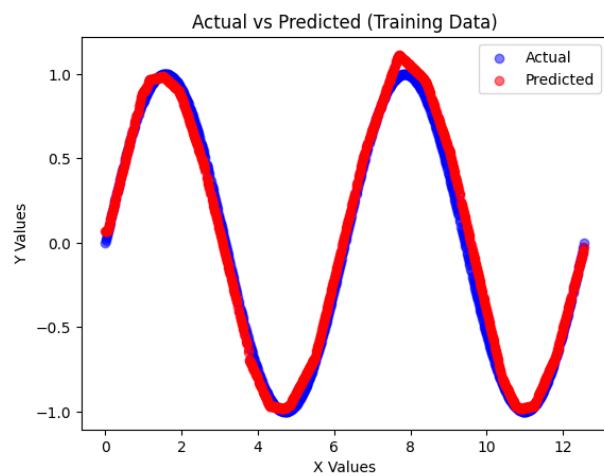
حالت اول: تابع فعال‌ساز **ReLU** برای تمام لایه‌های پنهان-لایه‌ی خروجی، قادر تابع فعال‌ساز.

تصاویر حاصل از عملکرد مدل‌هایی که بر این اساس ساخته می‌شود، در Gif-2-ReLU در پیوست آمده است. همانطور که در نمودار زیر مشخص است، loss تا جایی به طور کلی کاهش می‌یابد و از آنجا به بعد، روند افزایشی دارد. دلیل این اتفاق این است که از جایی به بعد در روند محاسبات وزن‌های لایه‌ها، مقادیر منفی می‌شوند و با اعمال **ReLU**، صفر پیش‌بینی می‌شوند. بنابراین، افزایش لایه‌ها همیشه اتفاق خوبی نیست و باید تعداد لایه‌های بهینه را متناسب با مسئله پیدا کنیم. برای این حالت، بهترین تعداد لایه با کمترین loss برابر ۹ است و بیشتر کردن تعداد لایه‌ها از ۹، تاثیر منفی روی آموزش مدل دارد.

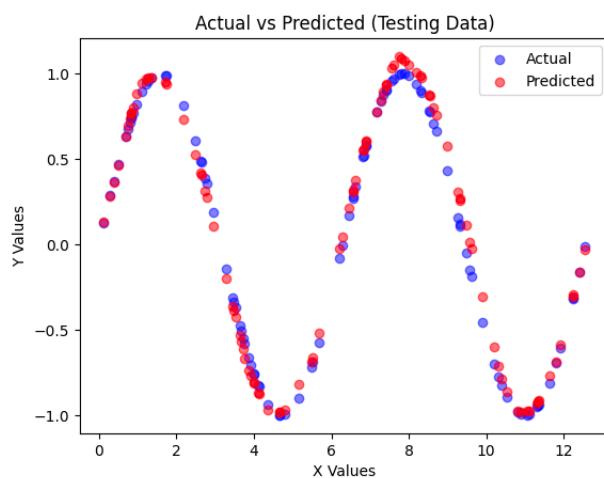


شکل 66. تغییرات loss با تابع فعال ساز ReLU

در زیر عملکرد مدل روی داده های آموزشی و تست را در بهترین حالت که ۹ لایه داشتیم می بینیم:

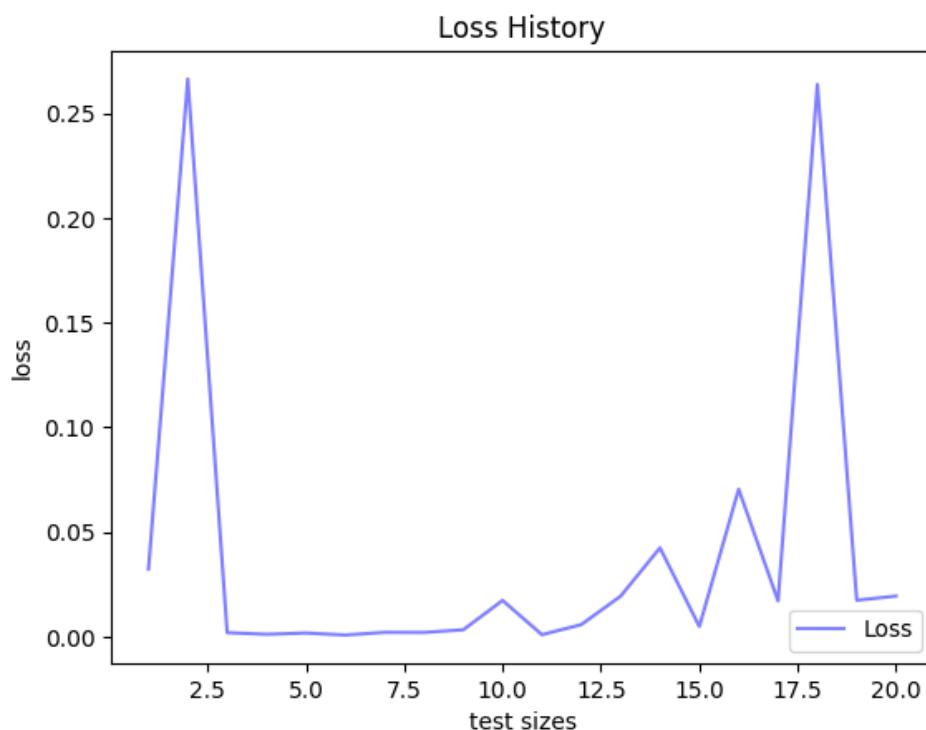


شکل 67. پیش بینی داده های آموزش برای بهترین مدل



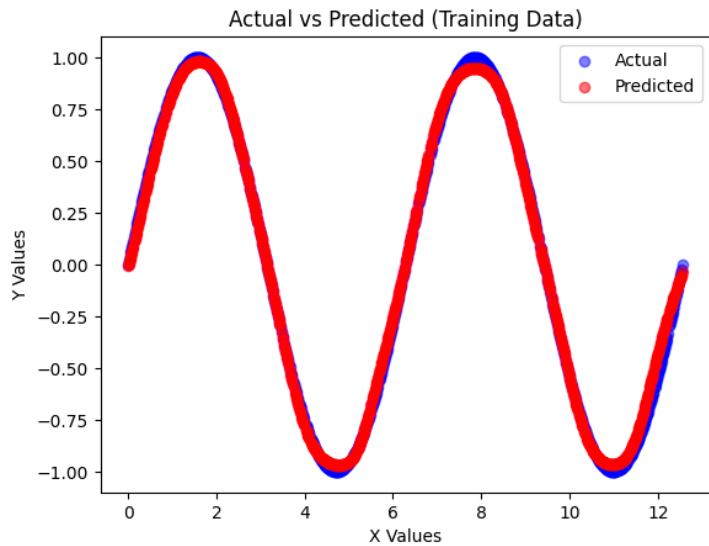
شکل 68. پیش بینی داده های تست برای بهترین مدل

حال، به بررسی حالت دوم می‌پردازیم. در این مدل از \tanh به عنوان تابع فعال‌ساز استفاده کردۀ‌ایم و نمایش کامل گام‌های آن در گیف Gif-3-tanh قابل مشاهده است. طبق نمودار زیر که loss را برای تعداد لایه‌های مختلف نمایش می‌دهد میتوان نتیجه گرفت که تا جایی، افزایش تعداد لایه‌ها به طور کلی منجر به کاهش loss می‌شود و از آنجا به بعد، مجدداً دچار فراز و نشیب‌های پی در پی می‌شود. این تغییر رفتار منطقی است چرا که انتظار داشتیم با افزایش تعداد لایه‌ها، مدل یادگیری بهتری داشته باشد و این اتفاق افتاد. اما اگر تعداد لایه‌ها از حدی بیشتر شود، پیچیدگی محاسبات بیشتر شده و منجر به رویارویی با مشکل vanishing gradient می‌شود که در شکل‌های نهایی این مشکل به صورت خطی افقی قابل مشاهده است. در حقیقت در این موقع، یادگیری برای مدل اتفاق نمی‌افتد که این مشکل در شکل‌های متناظر با تعداد لایه‌های بالا مثل ۱۷ و ۱۸ به صورت یک خط افقی قابل مشاهده است. بنابراین دقیق مدل کمتر شده و loss افزایش می‌یابد.

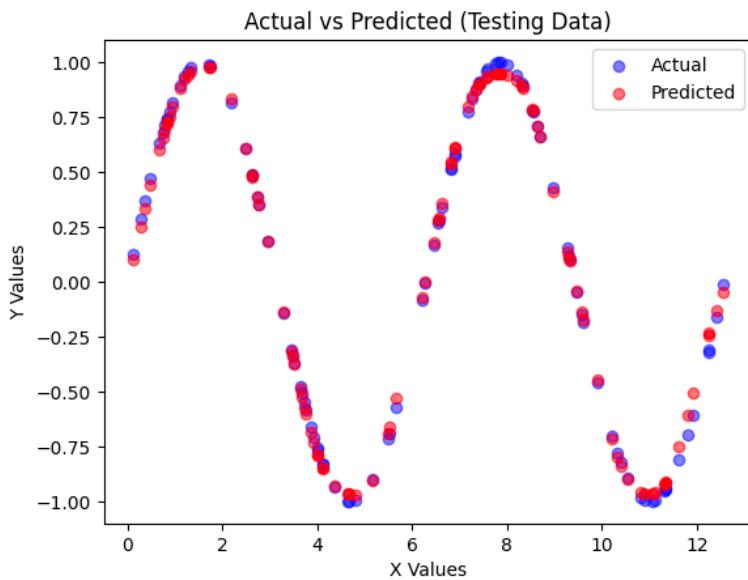


شکل 69. نمودار تغییرات loss با تغییرات test size

در ادامه شکل متناظر با مدل آموزش دیده شامل ۶ لایه را مشاهده می‌کنیم.



شکل 70. پیش‌بینی داده‌های آموزش برای مدل ۶ لایه



شکل 71. پیش‌بینی داده‌های تست برای مدل ۶ لایه

با مقایسهٔ دو مدل فوق که توابع فعال‌ساز آنها متفاوت بود در می‌یابیم که افزایش تعداد لایه‌ها، در هر دو مدل، تا حدی منجر به بهبود عملکرد شبکه می‌شود. اما از جایی به بعد، تاثیر مثبتی ندارد ضمن اینکه بار محاسباتی را افزایش می‌دهد.

همچنین نتیجهٔ می‌گیریم که فعال ساز \tanh برای این مدل عملکرد بهتری دارد چرا که در کنار اینکه کمترین loss آن، ناچیز و کمتر از loss مینیمم loss آن نیز از ماکزیمم loss قبلی که با ReLU بود، کمتر است.

ث) GridSearchCV

برای کار با GridSearchCV یک کلاس ساختیم. سپس، برای استفاده از این تکنیک یک instance از این کلاس می‌سازیم. نهایتاً، با استفاده از GridSearchCV، برای مدل ساخته شده با تعداد لایه‌های پنهان متفاوت (از ۱ تا ۲۰ لایه)، آن را اجرا می‌کنیم.

در نتیجه بهترین تعداد لایه‌ها در بازه‌ی ۱ تا ۲۰ را برای ما ۶ اعلام کرد که neg_mean_squared_error آن نیز ۰.۰۰۴۰ گزارش شده است. همانطور که قبلاً و با رسم نمودار loss ها نتیجه گرفته بودیم، این تکنیک نیز بهترین تعداد لایه را برای ما ۶ گزارش کرده است.

۴-۲. طبقه بندی

در این بخش به بررسی overfitting در مسئله طبقه بندی و مدل‌های classifier می‌پردازیم.

در این بخش طبق خواسته سوال با مجموعه دادگان MNIST، مسئله classification را انجام می‌دهیم. همانگونه که در سوال دو نیز دیدیم، این مجموعه دادگان شامل عکس‌های (28, 28) پیکسل از اعداد ۰ تا ۹ که به صورت دستنویس نوشته شده‌اند، است. برای استفاده از شبکه fully connected، نیاز داریم که این داده‌ها را از (28, 28) به یک آرایه به طول 784 درآوریم. در این صورت در شبکه‌هایی که در آینده آموزش می‌دهیم، لایه ورودی به تعداد ۷۸۴ نورون (تعداد پیکسل‌های هر عکس) خواهد داشت.

همچنین می‌دانیم که داده‌های آموزش و تست برای این مجموعه جدا بوده و ۶۰۰۰۰ عکس برای آموزش و ۱۰۰۰۰ عکس برای تست موجود است.

قبل از آموزش شبکه، بهتر است که این دادگان را نرمالایز کنیم. این کار را با استفاده از MinMaxScaler از کتابخونه sklearn بخش preprocessing انجام می‌دهیم. و همچنین نیاز است که label یا در واقع خروجی مد نظر برای هر داده را به صورت One Hot در آوریم که این کار با استفاده از get_dummies کتابخونه pandas انجام می‌شود. در این صورت ما یک آرایه خروجی به طول ۱۰ خواهیم داشت. بنابراین شبکه‌هایی که در آینده آموزش می‌دهیم، تعداد ۱۰ نورون در لایه خروجی خواهند داشت.

۴-۲-۱. افزایش داده‌ها

در هر حالت برای استفاده از بخشی از داده‌ها، یک شبکه fully connected با سه لایه با مشخصات زیر تعریف می‌کنیم.

Model Structure

Layer	Input → output	Activation
FC_1	784 → 256	ReLU
FC_2	256 → 64	ReLU
FC_3	64 → 10 (#classes)	SoftMax

جدول 4. مشخصات شبکه عصبی مد نظر در هر مرحله

همچنین در هر مرحله برای آموزش این شبکه، از پارامترهای جدول شماره ۵ استفاده می‌شود تا با بخشی از داده آموزش، شبکه سه لایه را آموزش دهیم.

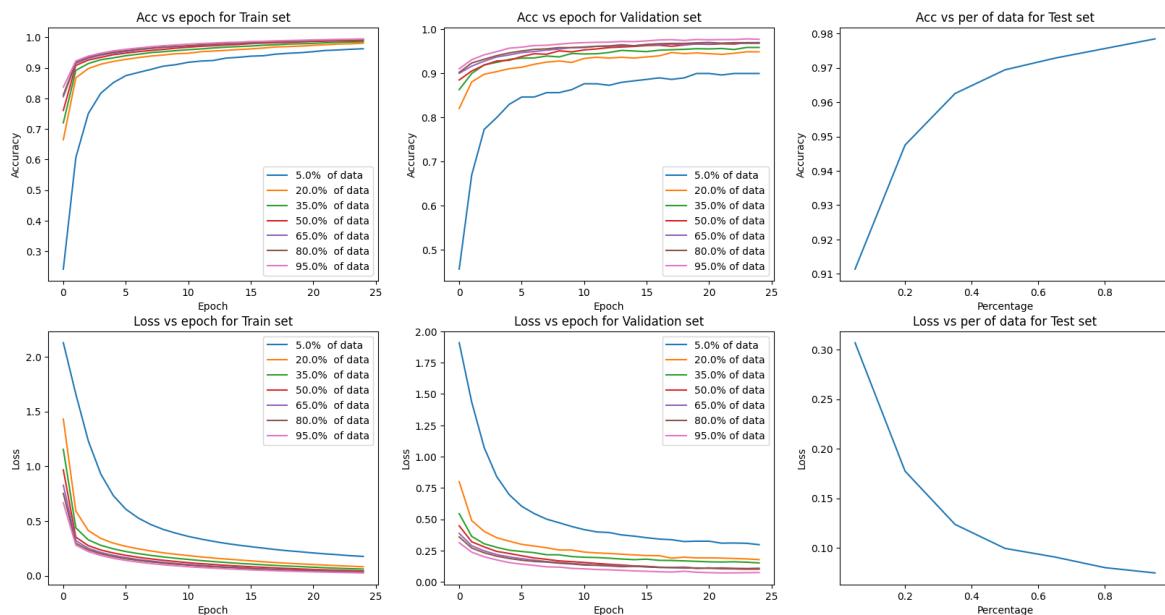
Model's training parameters

OPTIMIZER	Adam
LEARNING RATE	6e-5
EPOCHS	25
CRITERION	CrossEntropyLoss

جدول 5. پارامترهای آموزش شبکه عصبی پیاده‌سازی شده

حال اگر به ترتیب برای ۵، ۲۰، ۳۵، ۵۰، ۶۵، ۸۰ و ۹۵ درصد داده‌های آموزش، شبکه را آموزش دهیم.
نمودار زیر را برای نتایج آن خواهیم داشت.

Results of multiple modeling the MNIST data for different percentage of train data



شکل 72. نتایج شبکه عصبی سه لایه برای استفاده از میزان متفاوت داده آموزش

با توجه به نمودارهای بالا، مشخص است که همواره با افزایش میزان استفاده از داده‌های آموزش، شبکه ما آموزش بهتری برای این داده‌ها خواهد داشت. همچنین می‌توان گفت که با افزایش داده آموزش، شبکه ما خاصیت generalization را بدست خواهد آورد که همواره هدف از آموزش یک شبکه عصبی، همین است.

چند مورد وجود دارد که می‌توان از آن‌ها متوجه آموزش بهتر شبکه با افزایش میزان داده‌های آموزش است:

1. در مورد نمودارهای مربوط به داده‌های آموزش:

a. در نمودار دقت، با افزایش داده‌ها، در ایپاک آخر (۲۵ ام)، دقت بالاتری روی این مجموعه مشاهده می‌شود، که نشان‌دهنده یادگیری بهتر داده است. اما ممکن است که این دقت، به خاطر overfit شدن شبکه بر روی داده‌های آموزش باشد. درستی یا نادرستی این اتفاق را می‌توان با بررسی نمودار مربوط به داده‌های تست، بررسی کرد.

b. در نمودار دقت، با افزایش داده‌ها، در ایپاک اول، دقت شبکه از مقدار بزرگ‌تری شروع می‌شود، که نشان‌دهنده یک شروع خوب و یادگیری شبکه است (ممکن است باعث ایجاد خاصیت generalization برای شبکه ما شود)

c. در نمودار خط، با افزایش داده‌ها، خطا نهایی همواره کاهش می‌ابد.

d. در نمودار خط، با افزایش داده‌ها، مقدار خطای ایپاک اول از مقدار کمتری شروع می‌کند.

2. در نمودار مربوط به داده‌های validation:

در این داده‌ها نیز نتایج مشابه به آنچه در مورد داده‌های تست اتفاق میافتد داریم که تاکیدی دو چندان بر درستی نتیجه‌گیری ما مربوط به بخش آموزش است.

3. در نمودار مربوط به داده‌های آموزش:

a. در این نمودار نیز با افزایش داده‌ها، دقت افزایش پیدا می‌کند که نشان‌دهنده خاصیت generalization بوده و همچنین overfit شدن را رد می‌کند.

b. در این نمودار نیز با افزایش داده‌ها، خطای برای داده‌های تست کاهش پیدا می‌کند که نشان‌دهنده کیفیت خوب آموزش شبکه است.

بنابراین نتیجه می‌گیریم که در این مجموعه دادگان، همواره با افزایش میزان داده‌های آموزش استفاده شده، نتیجه بهتری خواهیم گرفت و شبکه بهتری برای استفاده خواهیم داشت.

این اتفاق در بیشتر حالات اتفاق میافتد، مگر اینکه دادگان آموزش شرایط خاصی داشته باشند. و یا برای مثال فراوانی داده‌های آموزش و تست ما balance نباشد و با افزایش داده‌های آموزش، ممکن است در این شرایط دقت کاهش پیدا کند. اما در حالت کلی و دادگان منطقی، افزایش داده‌های آموزش، باعث بهتر شدن شبکه آموزش دیده خواهد شد.

۴-۲-۲. افزایش لایه‌ها

در این حالت، شبکه‌هایی با تعداد لایه‌های مختلف را برای داده‌های آموزش MNIST، آموزش می‌دهیم. همه‌ی این شبکه‌هایی که تعریف می‌کنیم، با پارامترهای زیر آموزش می‌بینند.

Model's training parameters

OPTIMIZER	Adam
LEARNING RATE	1e-5
EPOCHS	10
CRITERION	CrossEntropyLoss

شکل 73. پارامترهای آموزش شبکه‌های عصبی با تعداد لایه‌های متفاوت

حال در هر مرحله، شبکه‌ای می‌سازیم که لایه ورودی آن ۷۸۴ نورون دارد. سپس با ترکیب‌های مختلفی که بعداً توضیح خواهیم داد، لایه‌های مخفی (از ۲ تا ۱۹ لایه چرا که وقتی می‌گوییم شبکه‌ای مثل ۳ لایه دارد، بدین معنا است که یک لایه ورودی، ۲ لایه مخفی و یک لایه خروجی دارد) را باتابع فعال ساز ReLU به مدل اضافه می‌کنیم. سپس در نهایت لایه خروجی را که به تعداد کلاس‌های مسئله classification نورون دارد (۱۰ نورون) را به شبکه باتابع فعال ساز SoftMax اضافه می‌کنیم. دلیل استفاده ازتابع SoftMax به این خاطر است که این تابع فعال ساز خروجی‌های نورون‌ها را به صورت احتمالی (مجموع ۱ و مقادیر همه بین ۰ و ۱) تولید می‌کند، بنابراین انتخاب کلاس مناسب برای داده ورودی در این مجموعه دادگان، ساده‌تر خواهد بود.

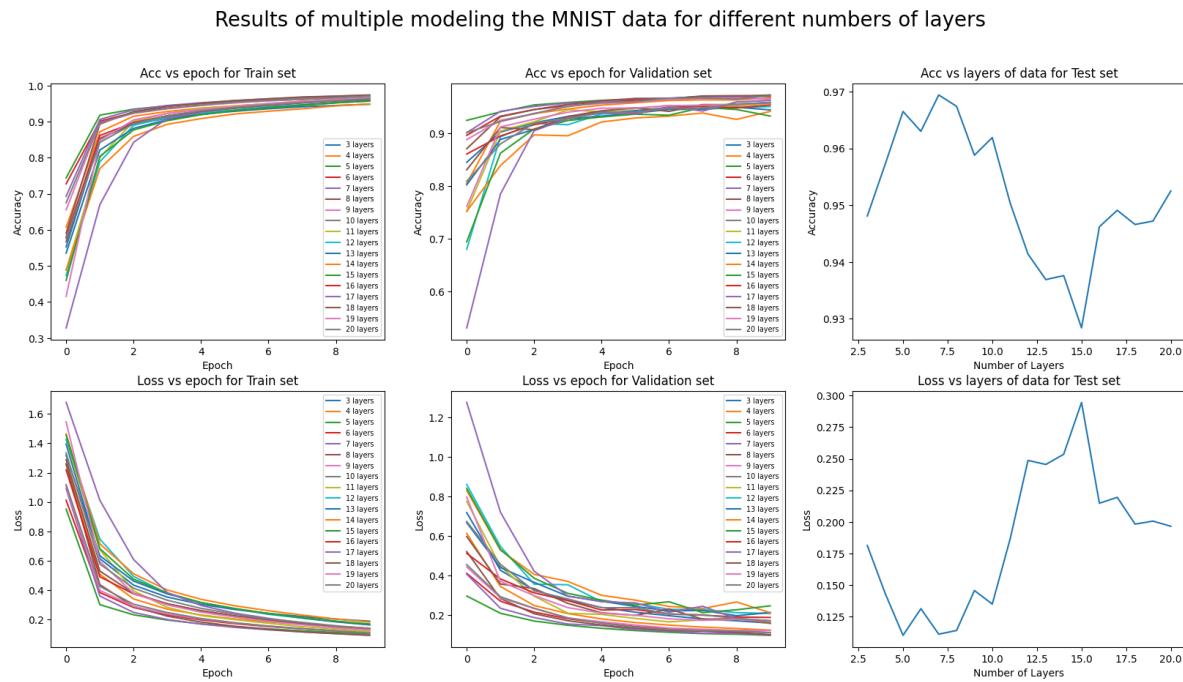
حال به نحوه اضافه کردن شبکه‌های عصبی می‌پردازیم. این ترتیب از لایه‌ها و نورون‌ها، در حالتی است که ما ۱۹ لایه مخفی داشته باشیم:

750, 700, 650, 600, 550, 512, 500, 450, 400, 350, 300, 256, 250, 200, 150, 128, 64, 32, 16

اگر تعداد لایه‌های مخفی ما کمتر باشد (برای مثال ۳ لایه)، ترکیبی از این لایه‌ها و تعداد نورون‌ها را انتخاب می‌کنیم، به گونه‌ای که ابتدا در لایه اول ۷۵۰ نورون (ایندکس صفر) و مابقی لایه‌ها با فواصل مساوی به صورتی در کل به تعداد دلخواه لایه انتخاب کنیم، لایه‌ها را انتخاب می‌کنیم. برای مثال اگر بخواهیم سه لایه انتخاب کنیم، ابتدا ۷۵۰ را برداشته و سپس با فاصله ۹ تایی از آن، ۳۵۰ را انتخاب کرده و سپس ۱۶ را انتخاب می‌کنیم.

در این صورت شبکه‌هایی با ترکیبی مختلف از لایه‌ها و تعداد نورون‌های آن‌ها تا ۱۹ لایه مخفی را آموزش خواهیم داد.

حال اگر به بررسی نتایج بپردازیم، خواهیم داشت.



شکل 74. نتایج حاصل از آموزش شبکه‌های مختلف از ۳ تا ۲۰ لایه

با بررسی نمودارهای مختلف، به چند مورد می‌توان پی برد.

۱. با افزایش لایه‌ها لزوماً دقت مدل افزایش پیدا نمی‌کند، چرا که با دقت به نمودار دقت و خطای برای داده‌های تست، دقت تا یک جایی افزایش یافته، ولی بعد از آن دقت کاهش یافته و خطای افزایش میابد. این اتفاق دلیل در overfit کردن مدل دارد. در واقع با افزایش لایه‌ها و پارامترهایی که لازم به آموزش دارند، نیاز ما به داده‌های مورد نیاز برای آموزش نیز به شدت زیاد شده و اگر این میل به داده‌های زیاد برطرف نشود، مدل توابع پیچیده‌ای که هدف در یادگیری آن‌ها را داریم، یاد نگرفته و بلکه تنها آن‌ها را حفظ می‌کند. یا به عبارت دیگر خاصیت generalization خود را از دست داده و نمی‌تواند برای داده‌های خارج از مجموعه آموزش، به خوبی پیش‌بینی کند (البته که میزان خوب بد بودن به مسئله و داده در دسترس بستگی دارد. برای مثال در اینجا و در حالت overfit در بدترین حالت ما دقت ۹۳ درصد روی داده‌های تست را داریم که ممکن است برای داده دیگری، آنچنان هم بد نبوده و بلکه بسیار مناسب باشد. بنابراین وقتی می‌گوییم به خوبی پیش‌بینی نمی‌کند، به صورت نسبی و در مقایسه با شبکه‌ای که به اندازه کافی روی داده‌های در دسترس آموزش دیده، می‌باشد. که در اینجا برای مثال دقت ۹۷ را داریم که لزوماً هم بهترین شبکه ممکن نیست).

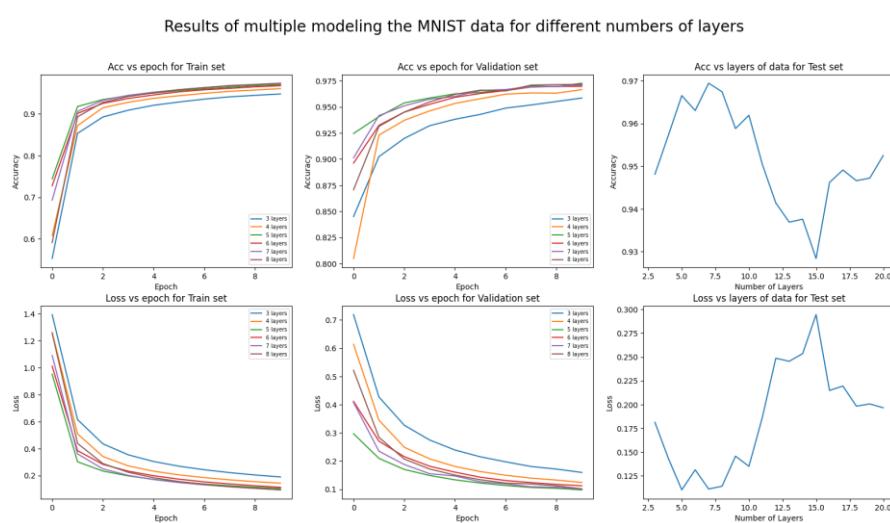
۲. این افزایش و کاهش دقت و خطأ، لزوماً دارای روند خاصی نمی‌باشد. برای مثال می‌توانیم بینیم با اینکه از جایی به بعد دقت مدل برای داده‌های تست کاهش یافته، اما بعد از کاهش شدید، دوباره شروع به افزایش می‌کند. این اتفاق ممکن است دلیل در ترکیب خوبی از لایه‌ها و ... باشد.

۳. مثلاً شبکه با ۱۷ لایه، از خطأ ۱.۲ (نسبتاً بالا) و همچنین دقت نسبت به پایینی شروع کرده، اما در انتهای به دقت نسبتاً خوبی رسیده، این اتفاق احتمالاً دلیل در مقادیر تصادفی ابتدایی برای وزن‌های بین لایه‌ها و همچنین تعداد زیاد آن‌ها در لایه‌های زیاد بوده.

۴. اگر بیشتر دقت کنیم، متوجه می‌شویم که زمانی که دقت کاهش و خطأ افزایش یافته و این تغییرات به خاطر افزایش لایه‌ها بوده، ترکیبی که از لایه‌ها داشتیم، ترکیب مناسبی نبوده، در واقع به خوبی تعداد نورون‌ها در هر لایه کم نشده و برای مثال یک لایه تعداد زیادی نورون داشته و با از دست دادن اطلاعات نسبتاً ارزشمندی، به لایه‌ای با تعداد کمی از نورون‌ها منتقل شده. در این صورت دقت کاهش خواهد یافت. اما اگر با افزایش لایه‌ها، از ترکیب مناسبی از آن‌ها استفاده کنیم که آرام آرام اطلاعات را منتقل کند، احتمالاً به شبکه خوبی دست خواهیم یافت. در واقع با پیش بردن این عمل، به سمت شبکه‌های deep خواهیم رفت. بنابراین می‌توانیم با اطمینان خوبی دلیل افزایش دوباره دقت را همین بدانیم.

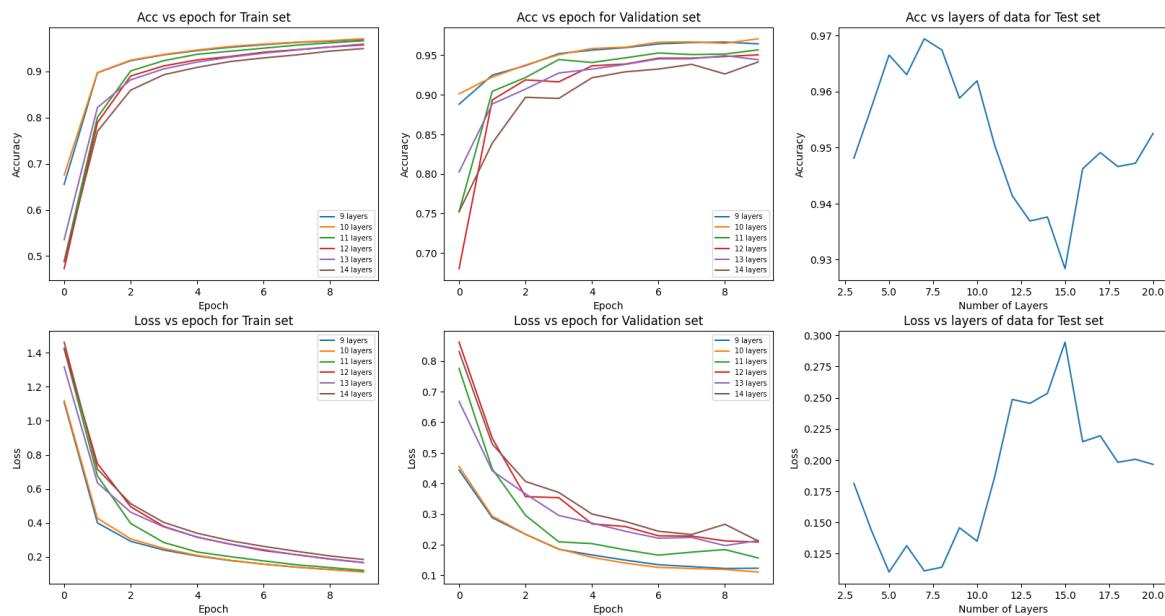
۵. جالب است که بینیم شبکه‌ای که با ۱۵ لایه آموزش دیده، در ابتدا و در اولین ایپاک با بهترین دقت و کمترین خطأ نسبت به سایر شبکه‌ها شروع کرده. اما در نهایت با کمترین دقت و بیشترین خطأ روی داده‌های تست آموزش خود را به پایان رسانده.

می‌توانیم نمودارهای جداگانه (۶ مدل ۶ مدل) را در ۳ شکل بعدی ببینیم.



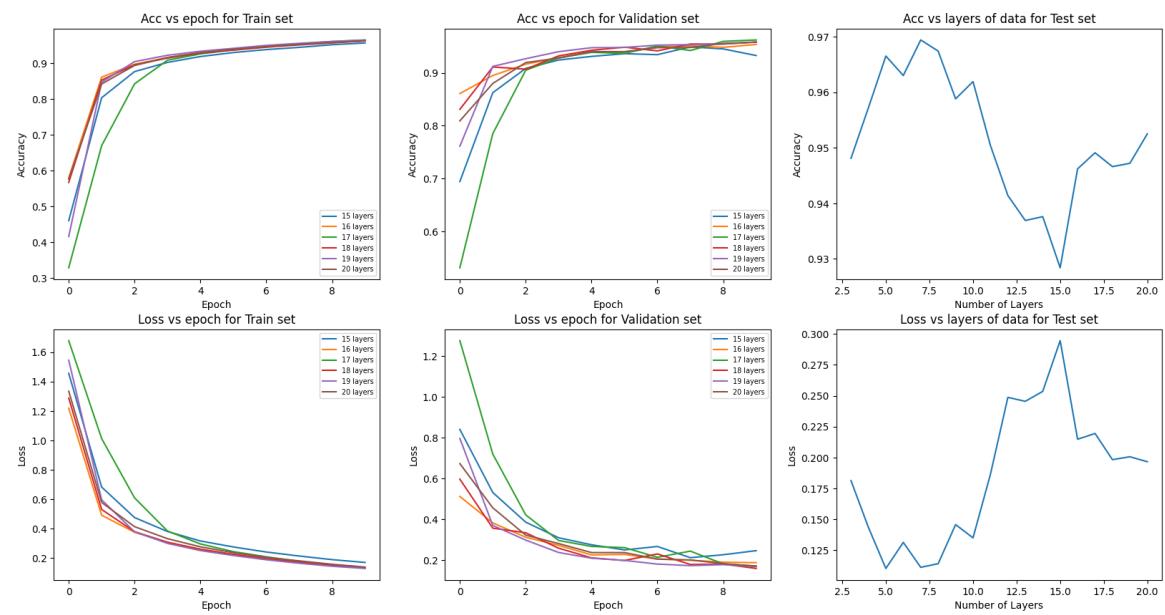
شکل ۷۵. نتایج مدل‌ها از ۳ تا ۸ لایه

Results of multiple modeling the MNIST data for different numbers of layers



شكل 76. نتایج مدل‌ها از ۹ تا ۱۴ لایه

Results of multiple modeling the MNIST data for different numbers of layers



شكل 77. نتایج مدل‌ها از ۱۵ تا ۲۰ لایه