به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

یادگیری ماشین

تمرین سوم

نام و نام خانوادگی : حسین سیفی

شماره دانشجویی : ۸۱۰۱۰۰۳۸۶

اردی‌بهشت ۱۴۰۲

سوال ۱

الف

تابع هزینه: یکی از عناصر حیاتی شبکه‌های عصبی هستند و نقش آن‌ها در سنجش عملکرد شبکه بر روی یک وظیفه خاص، بسیار مهم است. نحوه کار یک تابع هزینه در شبکه عصبی، اندازه‌گیری فاصله بین خروجی پیش‌بینی شده توسط شبکه و خروجی واقعی برای مجموعه‌ی داده‌های آموزشی است. در حالت یادگیری با نظارت[1]، هدف یک شبکه عصبی یادگیری نگاشتی از داده‌های ورودی به برچسب‌های خروجی است. در طول آموزش، شبکه به صورت مکرر وزن‌های خود را تنظیم می‌کند تا تابع خطا را کمینه کند.

توابع فعالساز: یک تابع فعال سازی تصمیم می گیرد که آیا نورون باید فعال شود و مقدار در خروجی قرار بگیرد یا خیر و در انواع دیگری از شبکه‌های عصبی نورون به کمک این تابع تصمیم خواهد گرفت که آیا خروجی نورون مذکور به شبکه در فرآیند پیش‌بینی برچسب داده‌ها می‌تواند کمک کند؟ و در صورت مثبت بودن پاسخ این پرسش، با توجه به ورودی‌های نورون، چه مقداری را بر روی خروجی خود قرار دهد.

ب

بله، وزن‌ها و بایاس‌های اولیه یک شبکه عصبی می‌تواند تأثیر قابل‌توجهی بر سرعت یادگیری شبکه و تعمیم آن به داده های جدید آن داشته باشد. در هنگام انتساب مقادیر اولیه به یک شبکه عصبی، معمولا وزن‌ها و بایاس‌ها مقادیر تصادفی کوچک انتخاب می‌شوند اما انتخاب مقادیر خیلی بزرگ یا خیلی کوچک برای وزن‌ها و بایاس‌های اولیه می‌تواند باعث شود که شبکه در طول تمرین با مشکلاتی مانند هم‌گرایی آهسته یا ناپدید شدن گرادیان‌ها[2] مواجه شود.

ج

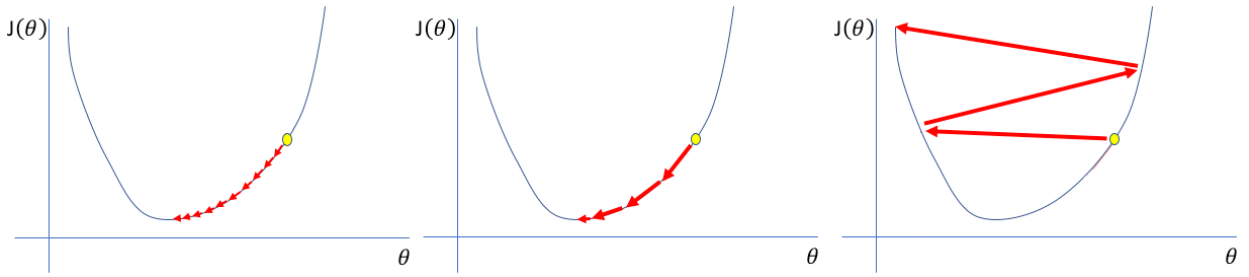
دو مفهوم Forward propagation و Backward propagation دو فرآیند مهم در آموزش شبکه‌های عصبی هستند. با توجه به تعاریف موجود در شبکه‌های عصبی، در Forward propagation به ترتیب از لایه اول تا آخر، ورودی‌های هر نورون به صورت مشخص وارد می‌شود و پس از جمع وزن‌دار ورودی هر نورون و اعمال تابع فعالساز، خروجی آن مشخص می‌شود و به لایه بعدی می‌رود. همچنین در این مرحله مشتق خروجی هر نورون را نسبت به ورودی(یا ورودی‌ها) آن محاسبه و ذخیره می‌شود. این روند تا زمانی ادامه می‌یابد که خروجی نهایی شبکه و پیش‌بینی آن برای هر نمونه مشخص شود. پس از به دست آمدن مقدار پیش‌بینی شده شبکه برای هر نمونه، به کمک تابع هزینه مقدار هزینه‌ای محاسبه می‌شود و سپس مرحله Backward propagation آغاز می‌شود. در این مرحله مشتق تابع هزینه نسبت به ورودی‌های هر نورون به کمک مشتق‌های محاسبه شده در مرحله Forward propagation، محاسبه می‌شود و با توجه به تاثیر مقدار نرخ یادگیری، بر روی وزن‌های موجود در شبکه تاثیر می‌گذارند تا در تکرارهای بعدی فرآیند فوق، مقدار هزینه را به حداقل مقدار خود برسانند. فرآیند Backward propagation که توضیح داده شد از لایه آخر شبکه آغاز می‌شود و در نهایت به لایه اول می‌رسد و به همین دلیل این نام را به خود گرفته است. بنابراین مهمترین تفاوت Backward propagation و Forward Propagation جهت انجام محاسبات مربوط به آن‌ها است که در هر کدام مطابق با نام آن انجام می‌شود.

د

یکی از مهم‌ترین نکات شبکه‌های عصبی انتخاب مقدار مناسب برای هایپرپارامتر نرخ یادگیری است. انتخاب نرخ یادگیری به صورت مناسب می‌تواند منجر به همگرایی[3] و یافتن بهینه سراسری[4] در زمانی معقول شود اما در صورتی که این مقدار بسیار کوچک انتخاب شود ممکن است

[1] Supervised Learning
[2] Vanishing Gradient
[3] Convergence
[4] Global Optimum

شبکه نتواند به مقداری بهتر از بهینه محلی[5] دست یابد و همچنین زمان همگرایی نیز ممکن است بسیار طولانی شود. در صورتی که مقدار نرخ یادگیری بسیار بزرگ انتخاب شود می‌تواند مزایایی به همراه داشته باشد که شامل فرار از بهینه‌های محلی است اما در نقطه مقابل، ممکن است که باعث شود به نقطه مناسب همگرا نشود یا حتی می‌تواند باعث واگرایی شود و هرگز به نقطه‌ای نزدیک به بهینه هم نرسد.



شکل ۱

برای مثال در اشکال فوق، شکل سمت چپ نشان دهنده حالتی است که نرخ یادگیری مقدار کوچکی انتخاب شده است و تعداد گام‌های زیادی تا رسیدن به بهینه نیاز دارد در صورتی که در شکل میانی که نشان دهنده مقدار مناسب برای نرخ یادگیری است، با تعداد گام‌های کمتری به بهینه دست یافته است و شکل سمت راست که مقدار بزرگی برای نرخ یادگیری را انتخاب کرده است منجر به واگرایی از نقطه بهینه شده است.

۵

به طور کلی هر چه شبکه عصبی دارای تعداد لایه‌های بیشتری باشد، قادر است تا ویژگی‌های بیشتر و پیچیده‌تری را از داده‌های ورودی یاد بگیرد و توابع بهینه‌تری را برای دادگان ورودی تخمین بزند. هر لایه در یک شبکه عصبی مجموعه‌ای از ویژگی‌ها را از داده‌ها می‌آموزد. افزودن لایه‌های بیشتر، شبکه می‌تواند سطوح بیشتری از انتزاع را بیاموزد و الگوهای ظریف‌تری را در داده‌ها شناسایی کند. برای مثال در شبکه‌ی BERT که برای پردازش متون مورد استفاده قرار می‌گیرد، شبکه در لایه‌های ابتدایی ویژگی‌های سطح کاراکتر را از جملات یاد می‌گیرد، در لایه‌های میانی به تدریج به یادگیری قواعد زبان و گروه‌های اسمی و در لایه‌های پایانی روابط بین گروه‌های اسمی و کلمات متفاوت را می‌شناسد. با این حال، افزودن لایه‌های بیشتر و در نتیجه پیچیده‌تر کردن شبکه، طبق قواعد تعادل بایاس و واریانس می‌تواند باعث کاهش خطای بایاس و افزایش خطای واریانس شود و به Overfit شدن شبکه بیانجامد.

۶

شبکه‌های عصبی در مرحله‌ی آموزش دارای مشکلات و سختی‌های فراوانی هستند:

۱. **انتخاب مقدار مناسب نرخ یادگیری**: همانطور که در بخش‌های قبلی همین سوال توضیح داده شد، انتخاب نرخ یادگیری دارای اهمیت زیادی است و عدم انتخاب مقدار مناسب می‌تواند منجر به عملکری غیربهینه از شبکه شود.
۲. **انتخاب معماری شبکه**: این مورد نیز در بخش قبل تشریح شد که عدم انتخاب معماری مناسب می‌تواند به Overfit شدن شبکه منجر شود و انتخاب تعداد لایه‌ها و اندازه هر لایه دارای اهمیت بسیار زیادی است.
۳. **نیازمند حجم زیادی از دادگان آموزشی**: به دلیل استخراج خودکار ویژگی‌ها توسط شبکه‌های عصبی، این مدل‌ها نیاز به تعداد زیادی داده دارند که در برخی کاربردها جمع آوری این حجم داده کاری طاقت فرسا و حتی در بعضی موارد غیرممکن است. از طرفی دیگر پردازش این حجم از دادگان نیازمند منابع پردازشی زیاد است.

[5] Local Optimum

۴. **Overfit شدن شبکه**: تعداد لایه‌های کم برای یک شبکه عصبی قدرت عمومی‌سازی[۶] آن را کاهش می‌دهد و زیاد کردن لایه‌ها نیز می‌تواند منجر به Overfit شدن شبکه شود.

هر کدام از مشکلات و چالش‌های فوق، راه‌حل‌هایی به شکل زیر به همراه دارند که با هزینه‌هایی اضافی، مشکلات ایجاد شده را رفع می‌کنند:

۱. استفاده از تکنیک‌های Grid search و تست کردن مقادیر مختلف نرخ یادگیری و سایر هایپرپارامترهای موجود در شبکه عصبی راه‌حلی مناسب برای یافتن بهترین مقدار است که می‌تواند زمان‌بر باشد و در ازای صرف کردن زمانی قابل توجه، بهترین عملکرد شبکه را به ما نشان دهد.
۲. برای تعیین بهترین معماری متناسب با کاربرد مورد نظر ما، راه‌های متفاوتی مانند تحلیل کاربرد و مشخص کردن بهترین معماری و تکنیک‌های متفاوت مانند Grid search و به طور خلاصه آزمون و خطا می‌تواند به ما کمک کند.
۳. در صورت جمع‌آوری دادگان مورد نیاز، امکان رفع مشکلات منابع پردازشی به کمک قطعه قطعه کردن دادگان و ذخیره Checkpoint می‌تواند به پشت سر گذاشتن این چالش توسط ما کمک کند.

می‌توان برای سه مشکل مطرح شده اول راه‌حل‌هایی(اگرچه زمانبر و پرهزینه) متصور شد اما جدی‌ترین مشکل شبکه‌های عصبی که Overfit شدن آن‌هاست راه حل معین و مشخصی ندارد و به فاکتورهای بسیار زیادی بستگی دارد.

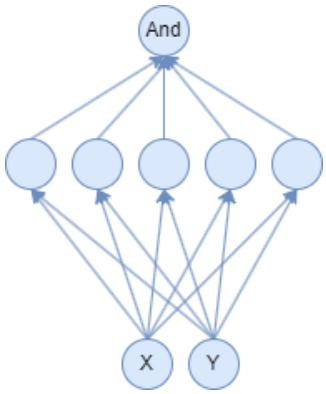
[6] Generalization

سوال ۲

در این سوال معماری شبکه عصبی که بتواند عملیات خواسته شده را در هر بخش انجام دهد، پیشنهاد می‌شود:

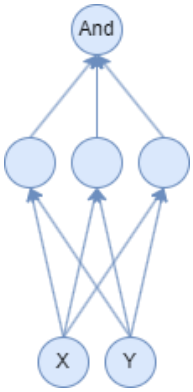
الف

در این سوال ناحیه قرمز رنگ مشخص شده را می‌توان به دو بخش مثلث و پنج ضلعی تقسیم کرد. برای هر کدام از این اشکال نیاز به یک لایه ورودی با دو نورون، یک لایه خروجی با یک نورون نیاز دارند. شکل پنج ضلعی نیاز به ۵ نورون در لایه میانی خود دارد تا مشخص کند نقطه مورد نظر در کدام سمت هر کدام از اضلاع قرار گرفته است که با تنظیم وزن‌های ورودی هر نورون می‌توان به طوری آن را تغییر داد تا در صورتی که نقطه در سمتی قرار گرفت که ناحیه قرمز در آن سمت قرار دارد، مقدار ۱ را در خروجی خود قرار دهد و لایه خروجی، عملیات منطقی And را بر روی مقدار خروجی نورون‌های لایه قبل اعمال می‌کند و در صورتی که این نورون مقدار ۱ را بر روی خروجی خود قرار دهد، به معنی این است که نقطه ورودی در ناحیه قرمز قرار گرفته است. شکل شبکه‌ی مورد نظر برای تشخیص قرار گرفتن نقطه در داخل پنج ضلعی به شکل زیر است:



شکل ۲

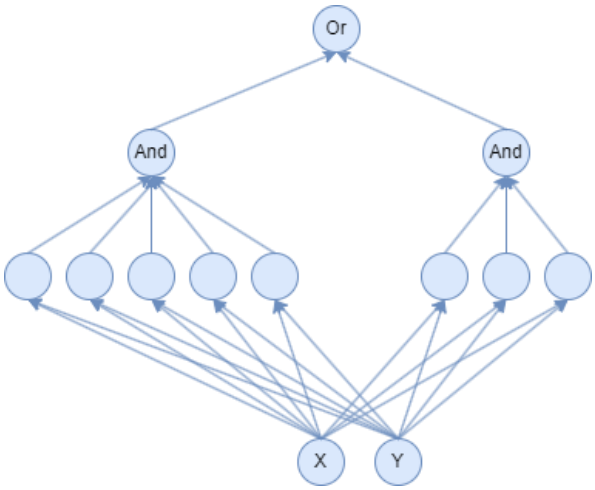
با استدلالی مشابه، شبکه‌ای به شکل زیر برای تشخیص مثلث قرمز رنگ ایجاد می‌شود با این تفاوت که لایه میانی این شبکه تنها ۳ نورون دارد:



شکل ۳

و در نهایت برای تجمیع دو شبکه معرفی شده در مراحل قبل، نیاز به یک لایه جدید با تنها یک نورون وجود دارد که عملیات منطقی Or را بر روی خروجی نهایی دو شبکه فوق اعمال می‌کند و به عنوان نورون خروجی کل شبکه عمل می‌کند. در صورتی که مقدار ۱ را در خروجی قرار

دهد به معنی این است که نقطه ورودی در ناحیه قرمز رنگ قرار دارد و در صورتی که مقدار ۰ را در خروجی قرار دهد به معنی این است که نقطه ورودی در ناحیه قرمز قرار ندارد. شبکه نهایی به شکل زیر می‌باشد:



شکل ۴

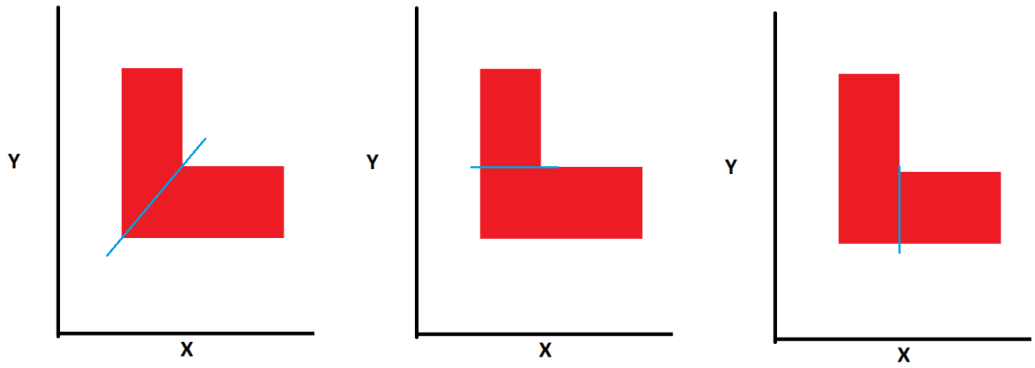
جزییات شبکه مورد نظر در جدول زیر نیز نمایش داده شده است:

جدول ۱

شماره لایه	نوع لایه	تعداد نورون
۰	ورودی	۲
۱	نهان(میانی)	۸
۲	نهان(میانی)	۲
۳	خروجی	۱

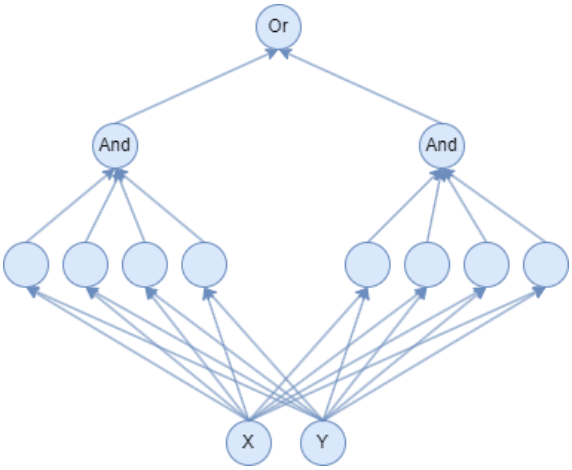
ب

در این بخش سوال می‌توان شکل داده شده را به یکی از روش‌های زیر به دو قسمت تقسیم کرد و سپس با استفاده از رویکردی مشابه با بخش (الف) همین سوال به طراحی شبکه عصبی مورد نیاز پرداخت. هیچکدام از نحوه‌ی تقسیم‌های زیر در توپولوژی شبکه تاثیرگذار نخواهند بود و تنها بر روی وزن‌های تخصیص یافته به ورودی‌های هر نورون تاثیر می‌گذارند.



شکل ۵

با استدلال مشابهی با بخش (الف) می‌توان برای این شکل نیز شبکه عصبی طراحی کرد. شبکه مورد نیاز دارای ۲ نورون در لایه ورودی، ۸ نورون برای تشخیص جهت نقطه نسبت به اضلاع دو چهارضلعی در لایه بعدی، ۲ نورون برای مشخص کردن حضور یا عدم حضور نقطه داخل هر کدام از چهارضلعی‌ها و یک نورون خروجی است. توپولوژی شبکه به شکل زیر است:



شکل ۶

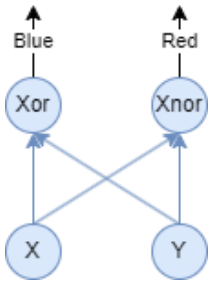
در تصویر فوق هر کدام از شبکه‌های سمت چپ و راست می‌توانند حضور نقطه در یکی از چهارضلعی‌ها را تشخیص دهند و در نهایت لایه آخر نتایج دو شبکه قبل را تجمیع می‌کند. جزئیات لایه‌های شبکه فوق در جدول زیر نیز نمایش داده شده است:

جدول ۲

شماره لایه	نوع لایه	تعداد نورون
۰	ورودی	۲
۱	نهان(میانی)	۸
۲	نهان(میانی)	۲
۳	خروجی	۱

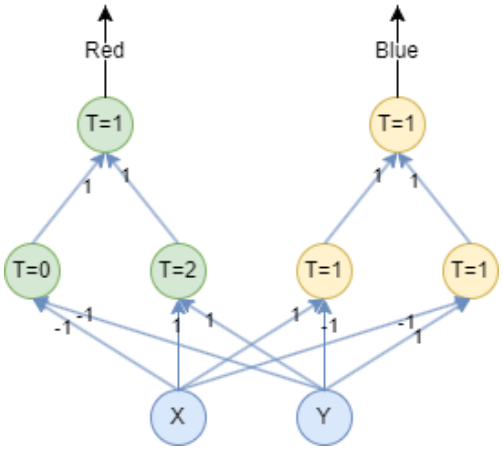
ج

در این بخش سوال، در صورتی که فضای ویژگی‌ها را پیوسته در نظر بگیریم، برچسب داده‌ها می‌تواند شامل سه نوع آبی، قرمز و هیچکدام باشد. در معماری این شبکه نیز همانند شبکه‌های بخش (الف) و (ب) نیاز به دو نورون در لایه ورودی داریم. در لایه بعدی نیاز به عملیات Xor برای تشخیص کلاس آبی و عملیات Xnor برای تشخیص کلاس قرمز نیاز داریم. شبکه توصیف شده به صورت نمادین به شکل زیر رسم شده است:



شکل ۷

هر کدام از عملیات Xor و Xnor توسط تنها یک نورون قابل پیاده‌سازی نیست و بنابراین شبکه فوق در شکل زیر به صورت باز شده به همراه وزن‌های مورد نیاز و آستانه فعالسازی نورون‌ها مشخص شده است:



شکل ۸

در معماری فوق، نورون‌های زرد رنگ عملیات Xor و نورون‌های سبز رنگ عملیات Xnor را پیاده‌سازی کرده‌اند. با توجه به توضیحات داده شده، دو نورون به عنوان خروجی در نظر گرفته شده است تا در صورتی که نقطه انتخاب شده عضو هیچکدام از کلاس‌های آبی و قرمز نبود هر دو نورون خروجی مقدار صفر به خود بگیرند. همچنین با توجه به تعریف و جدول مقادیر ورودی و خروجی عملگرهای Xor و Xnor، دو نورون لایه خروجی هیچگاه به طور همزمان مقداری برابر با یک نخواهند داشت. جزئیات معماری معرفی شده در جدول زیر ذکر شده است:

جدول ۳

شماره لایه	نوع لایه	تعداد نورون
۰	ورودی	۲
۱	نهان(میانی)	۴
۲	خروجی	۲

سوال ۳

الف

مشخصاً تصاویر کلاس اول شامل یک محیط بسته و به صورت دقیق‌تر شامل یک چهارضلعی محدب هستند در حالی که تصویر کلاس دوم شامل این ویژگی نیستند.

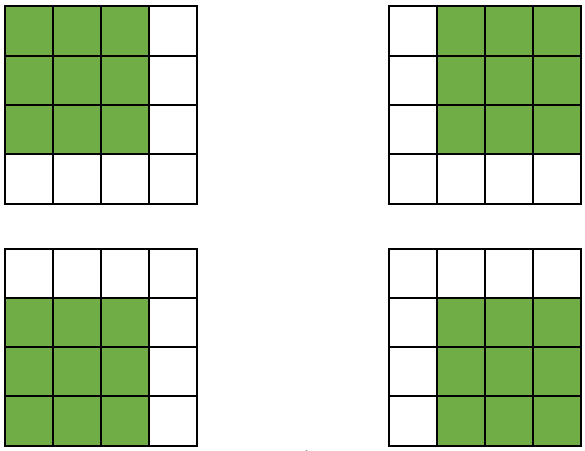
ب

برای اعمال فیلتر بر روی تصاویر داده شده به جای نقاط سفید فیلتر مقدار ۰ قرار داده می‌شود و به جای خانه‌های قرمز مقدار ۱ قرار می‌گیرد.(می‌توان به جای خانه‌های سفید مقدار ۱- قرار داد تا خانه‌های قرمز اضافی در تصویر با جریمه روبرو شوند) فیلتر تعریف شده به شکل زیر مقداردهی می‌شود:

۱	۱	۱
۱	-۱	۱
۱	۱	۱

شکل ۹

فیلتر فوق با میزان لغزش ۱ روی هر کدام از ۴ تصویر اعمال می‌شود و ماتریسی ۲ در ۲ به دست می‌آید که مقدار هر کدام از خانه‌های ماتریس ایجاد شده با اعمال فیلتر ۳ در ۳ فوق برای روی تصاویر ۴ در ۴ به شکلی که در تصویر زیر نشان داده شده است، محاسبه می‌شود:



شکل ۱۰

با انجام فرآیند توصیف شده و جمع کردن مقدار بایاس داده شده(۲-) با هر کدام از مقادیر به دست آمده، برای هر کدام از تصاویر داده شده ماتریس‌های زیر به دست می‌آید:(از انجام محاسبات صرف نظر شده است)

۱	۳
۲	۶

۵	۱
۵	۱

۱	۲
۲	۲

۱	۲
۱	۱

شکل ۱۱

با اعمال تابع فعالساز ReLU بر روی ماتریس‌های فوق، مقادیر کمتر از ۰ با ۰ جایگزین می‌شود و از آنجایی که مقادیر کمتر از صفر در این ماتریس‌ها قرار ندارد، پس از عبور از تابع فعالساز به شکل فوق باقی می‌مانند.

سپس بر روی هر ماتریس به دست آمده از اعمال فیلتر روی هر کدام از تصاویر، Maxpooling با اندازه ۲ در ۲ را اعمال می‌کنیم تا بیشترین مقدار هر ماتریس به دست آید. نتیجه به شکل زیر خواهد بود:

۶	۵
۲	۲

شکل ۱۲

اعداد فوق به همان ترتیب تصاویر، معرف تصاویر داده شده پس از اعمال عملیات خواسته شده می‌باشند.

ج

پس از اعمال فیلتر و Maxpooling، یک ماتریس(در مسئله فوق یک ماتریس ۱ در ۱ به عنوان نتیجه به دست آمد اما با استفاده از ابعاد دیگر Maxpooling به خصوص برای تصاویر بزرگتر، نتیجه می‌تواند یک ماتریس باشد.) به دست می‌آید که می‌تواند به عنوان ویژگی‌های یک تصویر تحت اثر یک فیلتر در نظر گرفته شود. در واقع این ماتریس به نوعی شامل ویژگی‌های استخراج شده از تصویر به کمک فیلتر و Maxpooling است. پس از به دست آمدن این ویژگی‌ها، می‌توان با پیاده‌سازی و آموزش یک شبکه Fully connected با معماری مناسب، روی مدل CNN این تصاویر را دسته‌بندی کرد و پس از آموزش، برچسب مناسب را برای داده‌های دیده نشده(تست) انتخاب کرد.

همچنین در حالت فعلی که تنها یک عدد به ازای هر تصویر به دست آمده است، با استفاده از یک آستانه که با توجه به دادگان موجود انتخاب می‌شود(مانند عدد ۴) می‌توان تصاویر را طبقه‌بندی کرد.

سوال ۴

الف

تابع فعالساز ReLU بار محاسباتی کمتری نسبت به توابع فعالساز Tanh و Sigmoid دارد چرا که ضابطه تعریف آن بسیار ساده‌تر از دو تابع دیگر است. ضابطه تابع Relu به شکل زیر است:

$$ReLU(x) = \begin{cases} 0 & if\ x < 0 \\ x & if\ x > 0 \end{cases}$$

در حالی که ضابطه دو تابع دیگر نسبت به این تابع بسیار پیچیده‌تر است. همچنین ReLU دارای مشتقی بسیار ساده است که در ادامه تعریف شده است. ساده‌تر بودن مشتق این تابع باعث می‌شود مراحل Backward Propagation و Forward Propagation بسیار سریع‌تر و با صرف منابع و زمان کمتری انجام گیرد.

$$\frac{dReLU(x)}{dx} = \begin{cases} 0 & if\ x < 0 \\ 1 & if\ x > 0 \end{cases}$$

با توجه به توضیحات فوق، تابع فعالساز ReLU بار محاسباتی کمتری نسبت به توابع فعالساز Tanh و Sigmoid دارد.

ب

مشکل Vanishing gradient زمانی رخ می‌دهد که گرادیان‌های محاسبه شده که برای به‌روزرسانی وزن‌های شبکه استفاده می‌شوند، بسیار کوچک یا نزدیک به صفر شوند. دلیل این اتفاق این است که در طول فرآیند Backward Propagation در شبکه‌های عصبی عمیق، زمانی که محاسبه مقدار گرادیان از آخرین لایه شروع می‌شود و تا لایه ابتدایی ادامه می‌یابد، به دلیل تاثیر تابع فعالساز مقدار گرادیان کوچک و کوچک‌تر می‌شود تا زمانی که به صفر میل کند و می‌تواند باعث شود که شبکه کندتر آموزش ببیند یا اصلاً یاد نگیرد.

همانطور که در بخش قبل گفته شد، تابع فعالساز ReLU دارای مشتق ساده‌تری نسبت به دو تابع دیگر است. مشتق این تابع نمی‌تواند دارای مقادیر بسیار بزرگ یا بسیار کوچک باشد. این تابع در هنگام Backward Propagation نیز باعث نمی‌شود که مقادیر بسیار کوچک گرادیان از لایه انتهایی تا لایه ابتدایی به صورت مقادیر بسیار کوچک منتشر شوند. از طرفی دیگر، دو تابع فعالساز دیگر(Sigmoid و Tanh) دو تابع غیرخطی هستند که با دریافت مقادیر بسیار بزرگ به عنوان ورودی، مقدار بسیار کوچکی را در خروجی خود قرار می‌دهند و پس از انجام Backward Propagation برای تعداد لایه‌های زیاد، ممکن است که باعث شود مقدار گرادیان بسیار کوچک شود و باعث Vanishing gradient شود. برای مثال مقدار مشتق تابع Sigmoid همیشه بین ۰ تا ۰.۲۵ است(برای Tanh بین ۰ و ۱ است) و پتانسیل اینکه مقداری بسیار کوچک و نزدیک به صفر را به عنوان خروجی ارائه دهد را داراست.

ج

در لایه آخر یک مسئله طبقه‌بندی باید از تابع فعالساز Sigmoid استفاده کرد چرا که این تابع تمامی مقادیر را به بازه ۰ تا ۱ نگاشت می‌کند و به دلیل اینکه خروجی در یک بازه‌ی بسته قرار دارد، می‌توان با تعریف آستانه[7](مقدار ۰.۵ در ساده‌ترین حالت) این بازه را به نواحی مشخصی تقسیم کرد که به هر کدام از کلاس‌ها تعلق دارد در حالی که بازه خروجی تابع فعالساز ReLU از یک سمت بسته و از سمت دیگر باز است($[0, +\infty)$) و نمی‌توان این بازه را به قسمت‌هایی مشخص شکست.

[7] Threshold

د

دو تابع ReLU و Leaky ReLU به ازای مقادیر ورودی بزرگتر از ۰ به صورت یکسانی عمل می‌کنند. اما تفاوت در رفتار تابع به ازای مقادیر ورودی کوچکتر از صفر است که تابع ReLU همیشه مقدار صفر و Leaky ReLU مقدار αx را بازمی‌گرداند و α مقداری بسیار کوچک و مثبت است(معمولا مقدار ۰.۰۱ استفاده می‌شود) در نتیجه مقدار مشتق دو تابع نیز برای مقادیر ورودی کوچکتر از صفر متفاوت است. مقدار مشتق هر دو تابع برای مقادیر ورودی بزرگتر از صفر برابر با ۱ است و برای مقادیر ورودی کوچکتر از صفر، تابع ReLU مقدار ۰ و تابع Leaky ReLU مقدار α را داراست. این تفاوت در مشتق به ازای ورودی‌های منفی باعث می‌شود نورون‌هایی که برای همه ورودی‌ها صفر خروجی می‌دهند و به اصطلاح نورون مرده[8] شناخته می‌شوند، کمک کند و این نورون‌ها را به جریان شبکه بازگرداند.

به طور کلی تابع ReLU به دلیل ساده‌تر و بهینه‌تر بودن محاسبات، انتخاب امن‌تری است اما می‌توان باعث به وجود آمدن تعداد زیادی نورون با خروجی صفر شود(نورون مرده) بنابراین در مواقعی که شبکه عصبی پیاده‌سازی شده شامل مجموعه داده‌ای عظیم و شامل نویز زیادی است استفاده از Leaky ReLU می‌تواند بهتر باشد و از Overfit شدن شبکه جلوگیری کند و به Generalization کمک کند. همچنین Leaky ReLU به دلیل جلوگیری از به وجود آمدن نورون‌های مرده برای شبکه‌های عمیق‌تر، مناسب‌تر از ReLU است. از طرفی دیگر Leaky ReLU سختی دیگری دارد که شامل تنظیم هایپرپارامتر اضافه شده به شبکه است.

[8] Dead Neurons

سوال ۵

الف

عبارت ریاضی محاسبه آنتروپی به شکل زیر تعریف می‌شود:

$$H(x) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

تعداد کل داده‌های استخدام برابر با ۱۰ می‌باشد و جزئیات ستون هدف(استخدام یا عدم استخدام) به صورت زیر می‌باشد:

جدول ۴

برچسب	تعداد	احتمال
قبول	۴	۰.۴
رد	۶	۰.۶

بنابراین محاسبات بی نظمی به شکل زیر انجام می‌شود:

$$H\left(\text{استخدام}\right) = -(0.4 * \log_2(0.4)) - (0.6 * \log_2(0.6)\ )$$

$$H\left(\text{استخدام}\right) = -\big(0.4 * (-1.32)\big) - (0.6 * (-0.73) = 0.528 + 0.442$$

$$H\left(\text{استخدام}\right) = 0.97$$

آنتروپی استخدام برابر با ۰.۹۷ به دست می‌آید.

ب

برای محاسبه Information gain صفت تحصیلات دانشگاهی ابتدا آنتروپی شرطی ستون استخدام نسبت به مقادیر مختلف این صفت را محاسبه می‌کنیم. جزئیات مورد نیاز جهت محاسبه این آنتروپی‌ها در جدول زیر نمایش داده شده است:

جدول ۵

مقدار صفت تحصیلات دانشگاهی	تعداد کل	تعداد با برچسب قبول	تعداد با برچسب رد
کارشناسی	۴	۳	۱
کارشناسی ارشد	۴	۱	۳
دکتری	۲	۰	۲

سپس مقادیر آنتروپی شرطی به شکل زیر محاسبه می‌شود:

$$H\left(\text{استخدام} \mid \text{تحصیلات} = \text{کارشناسی}\right) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.31 + 0.5 = 0.81$$

$$H\left(\text{استخدام} \mid \text{تحصیلات} = \text{کارشناسی ارشد}\right) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.5 + 0.31 = 0.81$$

$$H\left(\text{استخدام} \mid \text{تحصیلات} = \text{دکتری}\right) = -\frac{0}{2}\log_2\frac{0}{2} - \frac{2}{2}\log_2\frac{2}{2} = undefined + 0 = 0$$

سپس آنتروپی شرطی وزن‌دار بر اساس مقادیر فوق محاسبه می‌شود:

$$H\left(\text{استخدام} \mid \text{تحصیلات}\right) = \frac{4}{10} * 0.81 + \frac{4}{10} * 0.81 + \frac{2}{10} * 0 = 0.324 + 0.324 + 0 = 0.648$$

و در نهایت Information gain این صفت به شکل زیر به دست می‌آید:

$$IG\left(\text{تحصیلات}\right) = H\left(\text{استخدام}\right) - H\left(\text{استخدام} \mid \text{تحصیلات}\right) = 0.97 - 0.648 = 0.322$$

مقدار Information gain برای صفت سابقه کاری به روش مشابهی محاسبه می‌شود. جزئیات مورد نیاز جهت محاسبه این آنتروپی‌ها در جدول زیر نمایش داده شده است:

جدول ۶

مقدار صفت سابقه کاری	تعداد کل	تعداد با برچسب قبول	تعداد با برچسب رد
۰	۳	۱	۲
۱	۴	۱	۳
۲	۳	۲	۱

سپس مقادیر آنتروپی شرطی به شکل زیر محاسبه می‌شود:

$$H\left(\text{استخدام} \mid \text{سابقه کاری} = 0\right) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.528 + 0.389 = 0.817$$

$$H\left(\text{استخدام} \mid \text{سابقه کاری} = 1\right) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.5 + 0.31 = 0.81$$

$$H\left(\text{استخدام} \mid \text{سابقه کاری} = 2\right) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.389 + 0.528 = 0.817$$

سپس آنتروپی شرطی وزن‌دار بر اساس مقادیر فوق محاسبه می‌شود:

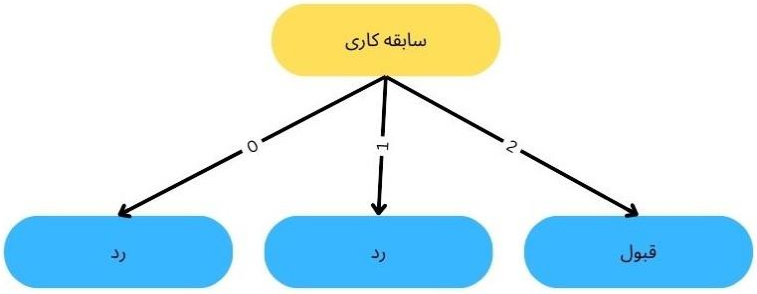
$$H\left(\text{استخدام} \mid \text{سابقه کاری}\right) = \frac{3}{10} * 0.817 + \frac{4}{10} * 0.81 + \frac{3}{10} * 0.817 = 0.2451 + 0.324 + 0.2451 = 0.8142$$

و در نهایت Information gain این صفت به شکل زیر به دست می‌آید:

$$IG\left(\text{سابقه کاری}\right) = H\left(\text{استخدام}\right) - H\left(\text{استخدام} \mid \text{سابقه کاری}\right) = 0.97 - 0.8142 = 0.1558$$

ج

با توجه به اینکه هیچکدام از مقادیر صفت سابقه کاری خلوصی کامل(۱۰۰ درصد) را نمی‌توانند ایجاد کنند، در شاخه‌ای از درخت تصمیم که هر کدام از مقادیر این صفت انتخاب می‌شوند، کلاسی که دارای سهم بیشتری از داده‌های آن شاخه است به عنوان برچسب دادگان انتخاب می‌شود. برای مثال در بین دادگان آموزشی که مقدار ۱ را برای سابقه کاری دارا هستند، ۳ نمونه داده دارای برچسب رد و ۱ نمونه داده دارای برچسب قبول است، بنابراین در شاخه‌ای که مقدار ۱ برای صفت سابقه کاری انتخاب می‌شود، دادگان برچسب رد را می‌گیرند. با توجه به توضیحات فوق درخت تصمیم به شکل زیر رسم شده است:



شکل ۱۳

در درخت فوق، مقادیر روی یال‌ها نشان دهنده صحت گزاره ذکر شده در گره قبلی درخت است. برای درخت تصمیم فوق و دادگان آموزشی، ماتریس آشفتگی به شکل زیر به دست آمده است:

جدول ۷

		واقعی	
		قبول	رد
پیش‌بینی شده	قبول	۲	۱
	رد	۲	۵

با کمک ماتریس آشفتگی فوق، مقدار معیارهای ارزیابی خواسته شده به شکل زیر محاسبه می‌شود:

$$Precision\left(\text{قبول}\right) = \frac{TP}{TP + FP} = \frac{2}{2+1} = 0.66$$

$$Precision\left(\text{رد}\right) = \frac{TP}{TP + FP} = \frac{5}{5+2} \approx 0.714$$

$$Precision_{macro} = \frac{0.66 + 0.714}{2} = 0.687$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2+5}{2+5+1+2} = \frac{7}{10} = 0.7$$

$$Recall\left(\text{قبول}\right) = \frac{TP}{TP + FN} = \frac{2}{2+2} = 0.5$$

$$Recall\left(\text{رد}\right) = \frac{TP}{TP + FN} = \frac{5}{5+1} \approx 0.83$$

$$Recall_{macro} = \frac{0.5 + 0.83}{2} = 0.665$$

$$Precision_{micro} = Recall_{micro} = Accuracy = 0.7$$

$$Specificity\left(\text{قبول}\right) = \frac{TN}{TN + FP} = \frac{5}{5+1} \approx 0.83$$

$$Specificity\left(\text{رد}\right) = \frac{TN}{TN + FP} = \frac{2}{2+2} = 0.5$$

مقدار Precision و Recall برای هر کلاس به صورت جداگانه و مقدار کلی برای ارزیابی طبقه‌بند به دو صورت Micro_averaged که میانگین مقادیر را به صورت وزن‌دار محاسبه می‌کند و Macro_averaged که میانگین بدون‌وزن محاسبه می‌کند، محاسبه شده است که هر مقدار Micro_averaged برابر با مقدار Accuracy می‌باشند. همچنین برای مقدار Specificity نیز می‌توان این دو میانگین را محاسبه کرد که برابر با مقدار Recall خواهند بود.

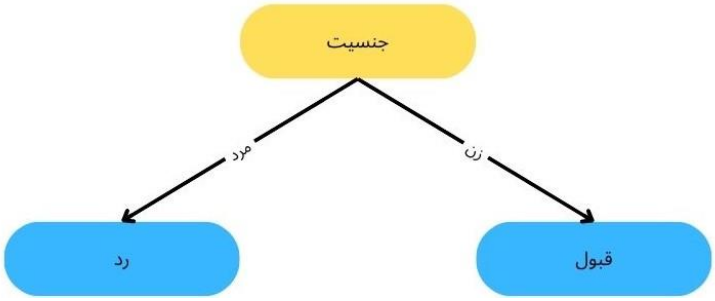
د

در جدول زیر تعداد برچسب‌های قبول و رد برای هر کدام از مقادیر ستون جنسیت مشخص شده است:

جدول ۸

مقدار صفت جنسیت	تعداد کل	تعداد با برچسب قبول	تعداد با برچسب رد
مرد	۶	۱	۵
زن	۴	۳	۱

همانند درخت رسم شده در بخش (ج)، با توجه به اینکه هیچکدام از مقادیر صفت جنسیت خلوصی کامل(۱۰۰ درصد) را نمی‌توانند ایجاد کنند، در شاخه‌ای از درخت تصمیم که هر کدام از مقادیر این صفت انتخاب می‌شوند، کلاسی که دارای سهم بیشتری از داده‌های آن شاخه است به عنوان برچسب دادگان انتخاب می‌شود.



شکل ۱۴

برای درخت تصمیم فوق ماتریس آشفتگی به شکل زیر رسم می‌شود:

		واقعی	
		قبول	رد
پیش‌بینی شده	قبول	۳	۱
	رد	۱	۵

معیارهای خواسته شده به شکل زیر محاسبه شده‌اند:

$$Precision\left(\text{قبول}\right) = \frac{TP}{TP + FP} = \frac{3}{3+1} = 0.75$$

$$Precision\left(\text{رد}\right) = \frac{TP}{TP + FP} = \frac{5}{5 + 1} \approx 0.83$$

$$Precision_{macro} = \frac{0.75 + 0.83}{2} = 0.791$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3 + 5}{3 + 5 + 1 + 1} = \frac{8}{10} = 0.8$$

$$Recall\left(\text{قبول}\right) = \frac{TP}{TP + FN} = \frac{3}{3 + 1} = 0.75$$

$$Recall\left(\text{رد}\right) = \frac{TP}{TP + FN} = \frac{5}{5 + 1} \approx 0.83$$

$$Recall_{macro} = \frac{0.75 + 0.83}{2} = 0.791$$

$$Precision_{micro} = Recall_{micro} = Accuracy = 0.8$$

$$Specificity\left(\text{قبول}\right) = \frac{TN}{TN + FP} = \frac{5}{5 + 1} \approx 0.83$$

$$Specificity\left(\text{رد}\right) = \frac{TN}{TN + FP} = \frac{3}{3 + 1} = 0.75$$

مقدار Precision و Recall برای هر کلاس به صورت جداگانه و مقدار کلی برای ارزیابی طبقه‌بند به دو صورت Micro_averaged که میانگین مقادیر را به صورت وزن‌دار محاسبه می‌کند و Macro_averaged که میانگین بدون‌وزن محاسبه می‌کند، محاسبه شده است که هر مقدار Micro_averaged برابر با مقدار Accuracy می‌باشند. همچنین برای مقدار Specificity نیز می‌توان این دو میانگین را محاسبه کرد که برابر با مقدار Recall خواهند بود.

سوال ۶

الف

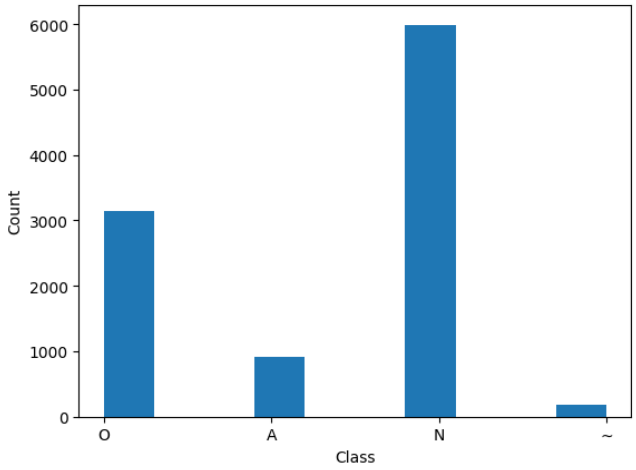
تعداد داده‌های هر کلاس به شرح زیر می‌باشد:

جدول ۹

کلاس	تعداد
N	۵۹۹۲
O	۳۱۵۱
A	۹۲۳
~	۱۸۷

ب

توزیع کلاس داده‌ها به شکل زیر می‌باشد:



شکل ۱۵

همانطور که در شکل فوق قابل مشاهده است، داده‌ها نسبت به کلاس خود بسیار نامتوازن هستند به طوری که تعداد نمونه‌های کلاس N حدود ۳۰ برابر تعداد نمونه‌های کلاس ~ است. این اتفاق می‌تواند باعث شود که طبقه‌بند، کلاس‌هایی با داده‌های بیشتر را بهتر یاد بگیرد و به سمت آن‌ها بایاس شود. همچنین باعث می‌شود در مرحله تست از برچسبی که در مجموعه داده بیشتر وجود دارد، بیشتر استفاده کند و این بایاس بودن به سمت کلاسی که داده‌های بیشتری دارد، در معیارهای ارزیابی مانند دقت[9] منعکس نمی‌شود. به عبارتی دیگر، مدل آموزش دیده با دادگان نامتوازن توانایی عمومی‌سازی پایینی را داراست.

ج

در این بخش سوال، در ابتدا دادگان را به کمک تابع train_test_split با نسبت ۹ به ۱ به دو بخش آموزش و تست تقسیم می‌کنیم:

[9] Accuracy

```
X_train, X_test, y_train, y_test = train_test_split(ecg[['feat'+str(i) for i
in range(1,170)]], ecg['label'], train_size=0.9)
```

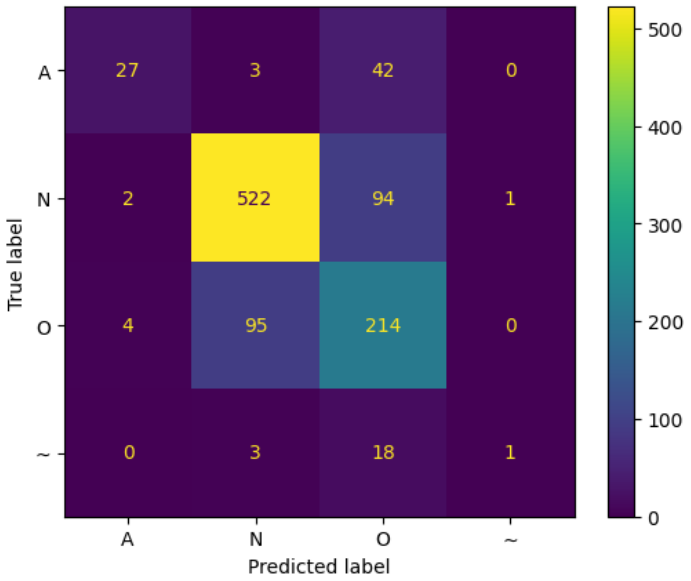
سپس با استفاده از کتابخانه scikit-learn یک مدل MLP با ۴ لایه نهان ایجاد می‌شود که اندازه لایه‌های نهان آن به ترتیب برای با ۱۲۸، ۳۲، ۳۲ و ۱۶ است. این اندازه‌ها به صورت تجربی انتخاب شدند و بین تعداد ویژگی‌های مجموعه داده(۱۶۹) و تعداد کلاس‌ها(۴) تغییر کردند تا بهترین مقدار برای اندازه هر لایه انتخاب شود. سایر خواص مدل تعریف شده در جدول زیر قابل مشاهده است:

جدول ۱۰

ویژگی	مقدار
آلفا	$۳*۱۰^{-۵}$
نرخ یادگیری	Adaptive
Early stopping	True
Shuffle	True
Random state	True
Max iter	200

ویژگی Early stopping کمک می‌کند تا از ادامه بدون بهره آموزش خودداری کنیم و هر زمان که دقت مدل از حدی بالاتر نرفت و Loss از حدی پایین‌تر نیامد، آموزش را متوقف می‌کند. ویژگی Shuffle ترتیب تاثیرگذاری دادگان بر وزن‌های شبکه در هر Epoch را تغییر می‌دهد و ویژگی Random state نیز به جای قرار دادن مقادیر صفر، مقادیر تصادفی برای وزن‌ها انتخاب می‌کند.

پس از تعریف شبکه به شکل فوق، این شبکه را با داده‌های آموزشی افراز شده از کل دادگان آموزش دادیم و مطابق انتظار، به کمک ویژگی تعریف شده Early stopping نیاز به گذراندن تمامی تکرارها وجود نداشت. پس از اتمام مرحله آموزش، مدل برچسب مناسب را برای هر کدام از داده‌های تست پیش‌بینی می‌کند. ماتریس آشفتگی برای داده‌های تست به شکل زیر خواهد بود:



شکل ۱۶

مقدار **دقت کل** برای این شبکه برابر با ۷۴ درصد است و مقدار سایر معیارهای ارزیابی خواسته شده در جدول زیر ذکر شده است:

Class	Precision	Recall	F1-Measure	Support
A	0.82	0.38	0.51	72
N	**0.84**	**0.84**	**0.84**	619
O	0.58	0.68	0.63	313
~	0.50	0.05	0.08	22
Macro Avg	0.68	0.49	0.52	1026
Weighted Avg	0.75	0.74	0.74	1026

همانطور که در ماتریس آشفتگی در شکل ۱۶ قابل مشاهده است، مدل آموزش دیده بر روی دادگان کلاس ~ به هیچ عنوان عملکرد جالب توجهی ندارد و مشخص است که هیچگونه یادگیری از الگوهای موجود در این کلاس صورت نگرفته است که به طور قطع به دلیل کمبود داده‌های آموزشی در این کلاس است. همچنین کلاس A دارای صحت[10] نسبتا مناسبی است اما مقدار فراخوانی[11] آن قابل قبول نیست. عملکرد نه چندان قابل قبول این کلاس نیز به دلیل کمبود نسبی داده‌های آن است. از طرفی دیگر کلاس N که تعداد داده‌ی زیادی داشت، عملکرد خوبی در تمامی معیارها به نمایش گذاشته است. با توجه به ماتریس آشفتگی مشخص است که تعداد زیادی داده از کلاس‌های دیگر برچسب کلاس N را دریافت کرده‌اند اما به دلیل زیاد بودن داده‌های این کلاس، داده‌های کلاس‌های دیگر که توسط مدل به این کلاس نسبت داده شده‌اند تاثیر چندان زیادی در معیارهای ارزیابی این کلاس نگذاشته‌اند. به طور کلی با بررسی معیارهای ارزیابی، مدل تنها بر روی یکی از کلاس‌ها(کلاس N) عملکرد خوبی از خود نشان داده است و یادگیری و عمومی‌سازی خوبی از سایر کلاس‌ها در این مدل وجود ندارد که به دلیل نامتوازن بودن کلاس‌ها است. اما با توجه به مقدار دقت به دست آمده، همانطور که پیش‌بینی شد کلاسی با داده‌های بیشتر که یادگیری بهتری روی آن‌ها صورت گرفته است و در نتیجه در بین دادگان تست نیز تعداد داده‌های بیشتری دارد، باعث بالا بودن غیر واقعی مقدار دقت شده است.

د

در این بخش ابتدا هر ویژگی را به صورت مستقل نرمال می‌کنیم. این عمل نیز به شکل زیر با استفاده از کتابخانه scikit-learn انجام گرفته است:

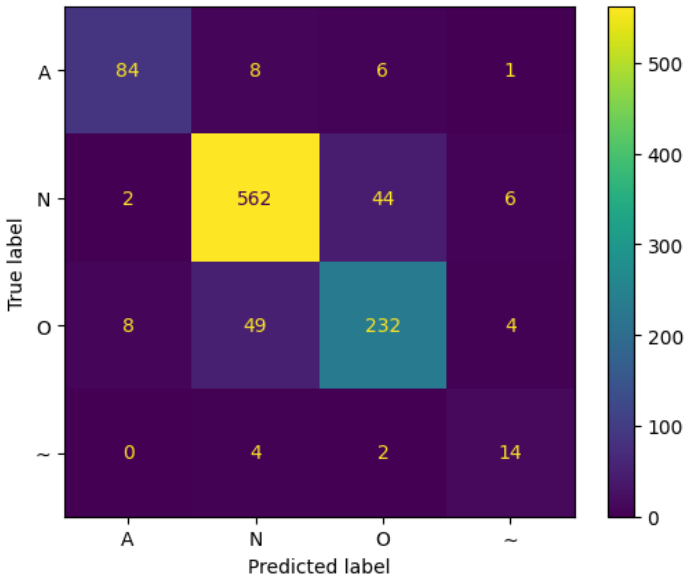
```
scaler = StandardScaler()

necg = scaler.fit(ecg[['feat'+str(i) for i in
range(1,170)]]).transform(ecg[['feat'+str(i) for i in range(1,170)]])
```

سپس شبکه‌ای مشابه با بخش ج آموزش می‌بیند و ارزیابی به شکل مشابهی انجام می‌گیرد. ماتریس آشفتگی و مقدار معیارهای ارزیابی به شرح زیر می‌باشد(مقدار **دقت کل** نیز برابر با ۸۷ درصد به دست آمده است):

جدول ۱۱

Class	Precision	Recall	F1-Measure	Support
A	0.89	0.85	0.87	99
N	**0.90**	**0.92**	**0.91**	614
O	0.82	0.79	0.80	293
~	0.56	0.70	0.62	20
Macro Avg	0.79	0.81	0.80	1026
Weighted Avg	0.87	0.87	0.87	1026

[10] Precision
[11] Recall



شکل ۱۷

عملکرد مدل روی تمامی کلاس‌ها بهبود یافته است اما همچنان روی کلاس N عملکرد بهتری نسبت به سایر کلاس‌ها از خود نشان می‌دهد که دلایل آن در بخش قبل به طور مفصل توضیح داده شد و به دلیل زیاد بودن تعداد نمونه‌های این کلاس صورت گرفته است. نکته جالب توجه عملکرد بهتر مدل بر روی کلاس A نسبت به کلاس O است. با وجود اینکه کلاس A دادگان کمتری از کلاس O در اختیار دارد اما مدل یادگیری بهتری از این کلاس داشته است. همچنین در کلاس ~ بهبود قابل توجهی مشاهده می‌شود و با توجه به تعداد دادگان این کلاس یادگیری خوبی صورت گرفته است.

۵

عملکرد طبقه‌بند بخش د نسبت به بخش ج بهبود یافته است. با استفاده از نرمال‌سازی دادگان نتایج بهبود یافته است که می‌تواند به چند دلیل باشد:

۱. کاهش تاثیر Outlierها: با توجه به اینکه با نرمال‌سازی مقادیر تمامی دادگان به یک بازه مشخص نگاشته می‌شوند، حتی فاصله Outlierها هم از سایر مقادیر کاهش می‌یابد و باعث تاثیر بیشتر از اندازه عادی این مقادیر می‌کاهد.
۲. قیاس ویژگی‌ها: در یک مجموعه داده هر ویژگی می‌تواند بازه منحصر به خود را برای مقادیر مجاز داشته باشد. برای مثال ممکن است برای یک ویژگی بازه ۱ تا ۲ مجاز باشد در حالی که برای ویژگی دیگری بازه مجاز ۰ تا ۱۰۰۰ باشد. در این شرایط ممکن است که طبقه‌بند تاثیر بیشتری را برای ویژگی دوم در نظر بگیرد در حالی که تاثیر هر کدام از ویژگی‌ها مستقل از بازه مجاز برای مقادیر آن‌ها است و نباید صرفا بر این اساس برای تاثیرگذاری ویژگی‌ها تصمیم‌گیری انجام گیرد. با نرمال‌سازی دادگان به یک بازه مشخص، از تاثیر ذکر شده مقادیر بر طبقه‌بند جلوگیری می‌شود و مقایسه ویژگی‌ها ممکن می‌شود.
۳. ثبات عددی مدل: بسیار کوچک یا بسیار بزرگ بودن مقادیر ویژگی‌ها می‌تواند منجر به سرریز یا زیرریز شود و نرمال‌سازی از این اتفاق جلوگیری می‌کند.

و

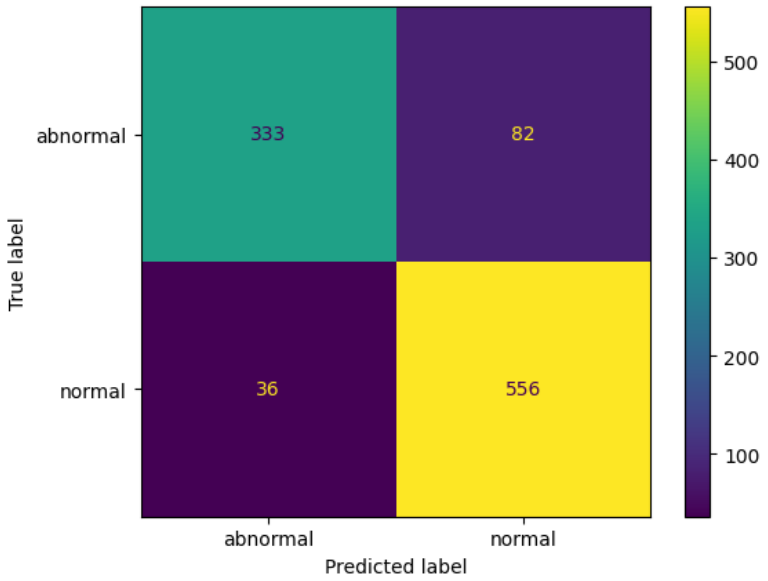
در این بخش یک پیش‌پردازش به داده‌ها اضافه شده است. داده‌های کلاس ~ از مجموعه داده حذف می‌شود و داده‌های کلاس‌های A و O تحت عنوان abnormal با یکدیگر ترکیب می‌شوند. این پیش‌پردازش به شکل زیر انجام گرفته است:

```
aecg = ecg.drop(ecg[ecg['label']=='~'].index)

aecg['label'] = aecg['label'].replace('N', 'normal').replace('O',
'abnormal').replace('A', 'abnormal')
```

سپس باقی مراحل مانند بخش د انجام می‌شود. . ماتریس آشفتگی و مقدار معیارهای ارزیابی به شرح زیر می‌باشد(مقدار **دقت کل** نیز برابر با ۸۸ درصد به دست آمده است):

جدول ۱۲

Class	Precision	Recall	F1-Measure	Support
Abnormal	**0.90**	0.80	0.85	412
normal	0.87	**0.94**	**0.90**	595
Macro Avg	0.89	0.87	0.88	1007
Weighted Avg	0.88	0.88	0.88	1007



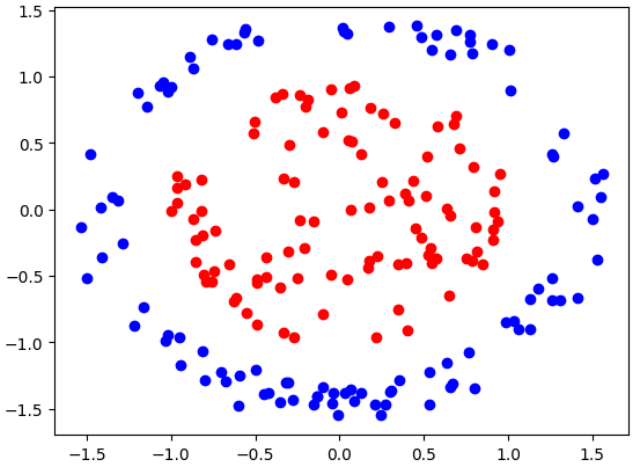
شکل ۱۸

نکته جالب توجه پس از ترکیب دو کلاس A و O در مقابل کلاس N این است که کلاس N همچنان کلاس غالب است و بنابراین عملکرد بهتر طبقه‌بند(در دو معیار Recall و F1-Measure) روی این کلاس آنچنان دور از انتظار نبود. با توجه به ماتریس آشفتگی و گزاره ذکر شده، تعدادی زیادی (۸۲ عدد) از نمونه‌های کلاس abnormal برچسب کلاس normal را دریافت کرده‌اند که به دلیل بایاس بودن مدل به سمت کلاس بزرگتر است. اما به طور کلی با ترکیب دو کلاس کوچکتر و حذف کلاس نویزی، عملکرد مدل بسیار بهتر از پیش شد. در چنین شرایطی که تعداد نمونه‌های دو کلاس نیز تا حد قابل قبولی به یکدیگر نزدیک شده‌اند، می‌توان مقدار دقت کلی این مدل که برابر با ۸۴ درصد است را نیز معتبر دانست و عملکرد مدل را قابل قبول توصیف کرد.

سوال ۷

الف

همانند یکی از سوالات تمرین دوم یادگیری ماشین، تابعی برای ایجاد دادگان تصادفی به صورت دایره و حلقه تعریف شده است.دادگان ایجاد شده توسط این تابع با توجه به نیاز توصیف شده در این سوال به شکل زیر هستند:



شکل ۱۹

ب

برای تعریف مدل Madaline یک کلاس با سه تابع تعریف شده است. تابع اول Constructor است که متغیرهای مورد نیاز در آن تعریف شده است و مقادیر مناسب را به آن‌ها اختصاص می‌دهد. این تابع به شکل زیر می‌باشد:

```
def __init__(self, num_l, epochs=500, learning_rate=0.1):
        self.epochs = epochs
        self.learning_rate = learning_rate
        self.w = np.random.rand(num_l, 2)
        self.v = [1] * num_l
        self.b = np.random.rand(num_l, 1)
        self.b2 = num_l - 1
```

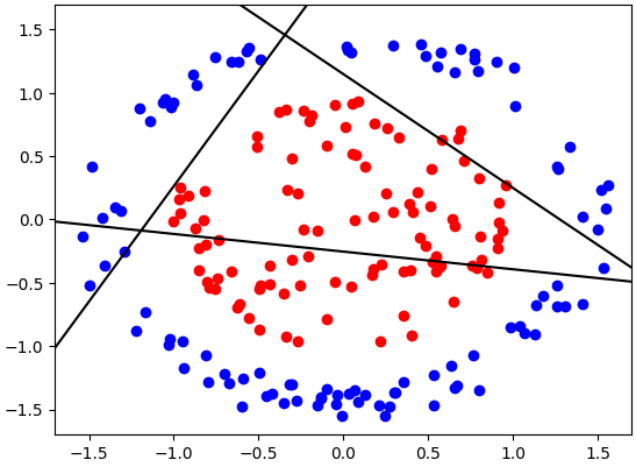
سپس کلاس آموزش مدل بر اساس تعاریف مطرح شده Madaline به شکل زیر تعریف شده است:

```
def fit(self, X, Y):
        count = 0
        for iter in range(self.epochs):
            for x, label in zip(X, Y):
                z_in = np.array([np.matmul(x, self.w.T)]).T + self.b
                z = np.heaviside(z_in, 1) * 2 - 1
                y_in = np.dot(np.squeeze(z), np.squeeze(self.v)) + self.b2
                y = np.heaviside(y_in, 1) * 2 - 1
                if y != label:
                    if label == 1:
                        z_j = max(z_in)
                        indices = np.where(z_in == z_j)
                        self.w[indices, :] = self.w[indices, :] + self.learning_rate * (1 -
z_in[indices]) * np.array(x)
                        self.b[indices] = self.b[indices] + self.learning_rate * (1 -
z_in[indices])
                    else:
                        indices = [i for i, x in enumerate(z_in) if x > 0]
                        for ind in indices:
                            self.w[ind, :] = self.w[ind, :] + self.learning_rate * (-1 -
z_in[ind]) * np.array(x)
                            self.b[ind] = self.b[ind] + self.learning_rate * (-1 -
z_in[ind])
```

و در نهایت کلاس زیر برای پیش‌بینی کلاس داده‌ها بر اساس پارامترهای آموزش دیده مدل تعریف شده است:

```
def predict(self, X):
        y = []
        for item in X:
            z_in = np.array([np.matmul(item, self.w.T)]).T + self.b
            z = np.heaviside(z_in, 1) * 2 - 1
            y_in = np.dot(np.squeeze(z), np.squeeze(self.v)) + self.b2
            y.append(np.heaviside(y_in, 1) * 2 - 1)
        return y
```

پس از تعریف کلاس Madaline یک نمونه از این مدل ایجاد می‌شود و با داده‌های به دست آمده آموزش می‌بیند. تعداد نورون‌های لایه آخر مورد استفاده در این مدل برابر ۳ خواهد بود. سپس با استفاده از وزن‌های به دست آمده از مرحله آموزش، خطوط جداساز رسم شده‌اند. نمودار داده‌ها به همراه خطوط جداساز به شکل زیر می‌باشند:



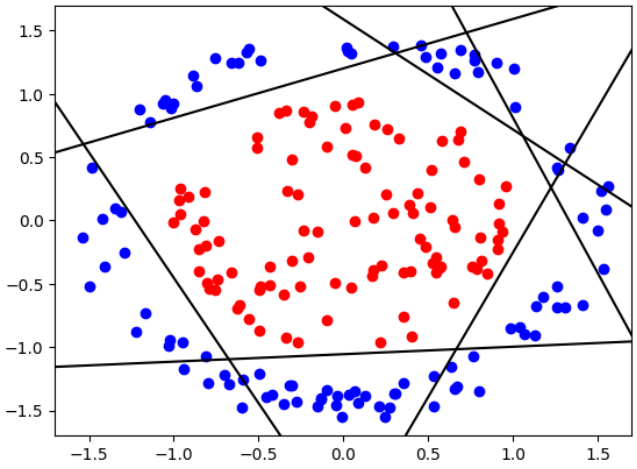
شکل ۲۰

دقت این طبقه‌بند به کمک تابع accuracy موجود در فایل کد برابر با **۰.۷۷** به دست آمده است.

$$Accuracy = 0.77$$

ج

در این بخش عملیات صورت گرفته در بخش ب با ۶ نورون در لایه آخر انجام می‌گیرد. شکل نقاط به همراه مرزهای جداساز به شکل زیر است:



شکل ۲۱

دقت این طبقه‌بند نیز برابر با **۱** به دست آمده است.

$$Accuracy = 1.0$$

د

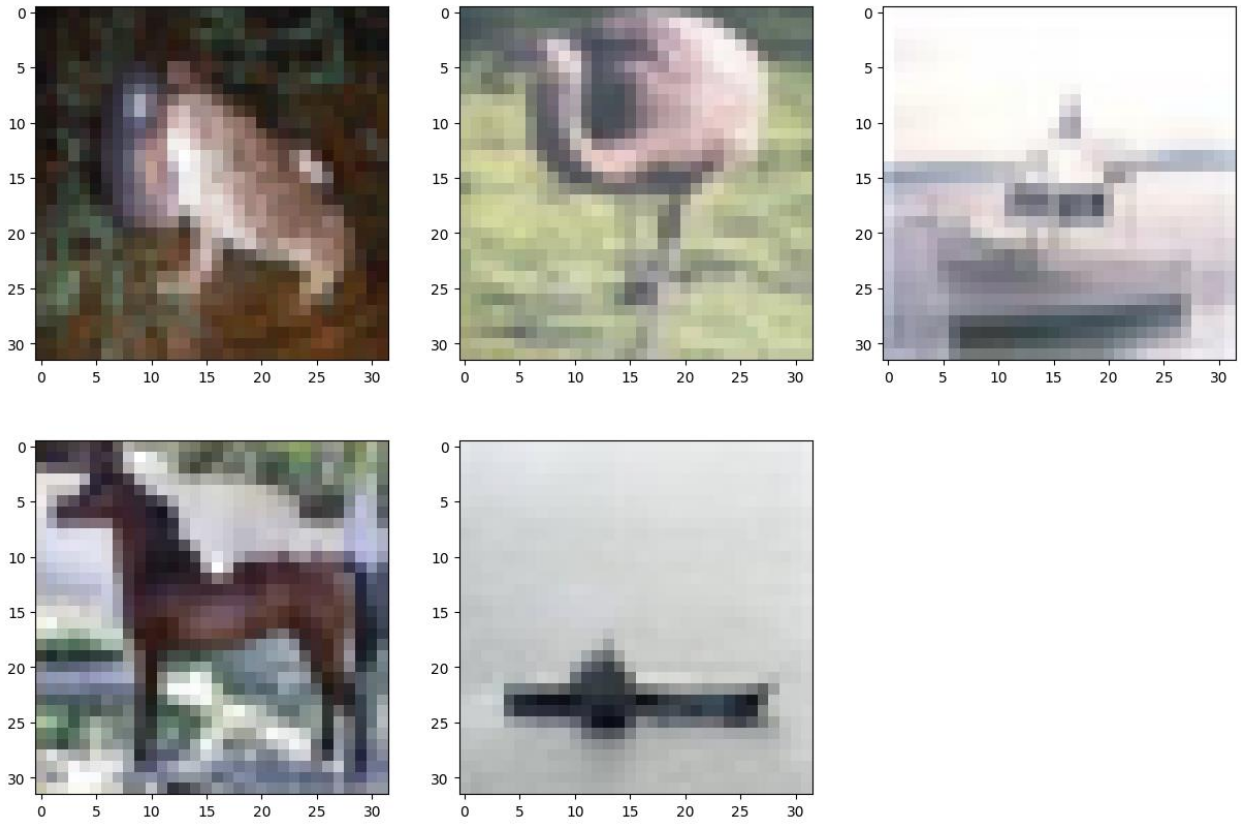
همانطور که تصاویر شامل خطوط جداساز دو طبقه‌بند نشان می‌دهند، طبقه‌بند با ۳ نورون در لایه آخر نتوانسته است نقطه‌های دو کلاس را به خوبی از یکدیگر جدا کند و دقتی برابر با ۰.۷۷ را ارائه کرده است در حالی که طبقه‌بند با ۶ نورون در لایه آخر به خوبی و با دقت ۱ این دادگان را جدا کرده است. در برخی اجراها با ۶ نورون در لایه آخر پیش می‌آمد که یک یا دو خط جداساز هیچ مشارکت قابل توجهی در طبقه‌بندی دادگان انجام نمی‌دادند اما در نمونه ذکر شده در شکل ۲۱ هر کدام از خطوط، با مشارکتی متفاوت در نتیجه طبقه‌بند تاثیر گذار بوده‌اند. اما در طبقه‌بند با ۳ نورون در لایه آخر(شکل ۲۰) یکی از خطوط جداساز عملکرد خوبی از خود نشان نداده است و تعداد زیادی از دادگان کلاس قرمز را به کلاس آبی اختصاص داده است.

به دلیل وجود فاصله نسبتا زیاد بین ویژگی‌های دو کلاس فوق، و کم بودن تعداد داده‌های آموزشی، شاید با ۴ یا نهایتا ۵ خط بتوان به خوبی داده‌ها را طبقه‌بندی کرد و بهترین مقدار دقت برای دادگان آموزشی را به دست آورد اما در صورتی که این فاصله بین ویژگی‌ها کمتر شود و تعداد داده‌های آموزشی به سمت بی‌نهایت میل کند، تعداد خطوط جداساز مورد نیاز نیز به صورت قابل توجهی افزایش می‌یابد. این اتفاق به این دلیل است که شکل داده‌های میانی به دایره میل می‌کند و برای دستیابی به طبقه‌بندی با دقت ۱، نیاز به بی‌نهایت خط مماس بر دایره میانی وجود دارد و در صورت کم بودن فاصله بین ویژگی‌های دو کلاس، این تعداد می‌تواند طبقه‌بندی قابل قبولی انجام دهد.

سوال ۸

الف

با استفاده از قطعه کد داده شده مجموعه داده cifar10 در دو بخش آموزش و تست بارگیری شد و ۵ تصویر زیر به صورت تصادفی به نمایش درآمد:



شکل ۲۲

ب

داده‌های مورد استفاده نیاز به چند پیش‌پردازش جزئی دارند که به شرح زیر می‌باشد:

- **تبدیل برچسب‌ها به فرمت One-hot**: تنها پیش‌پردازش مورد نیاز برچسب‌های دادگان، تبدیل آن‌ها از فرمت عددی به فرمت One-hot است که با توجه به معماری شبکه عصبی مورد استفاده و اختصاص یک نورون خروجی به هر یک از کلاس‌ها، این پیش‌پردازش مورد نیاز است.
- **تبدیل نوع داده‌ای پیکسل‌ها به float**: مقدار هر پیکسل در ابتدا از نوع numpy.uint8 است که باید به نوع float تبدیل شود.
- **نرمال‌سازی ویژگی‌ها**: بهتر است در ابتدا ویژگی‌ها(مقادیر پیکسل‌ها) نرمال شوند. با توجه به اینکه هر ویژگی مقداری بین ۰ تا ۲۵۵ دارد، با تقسیم ویژگی‌ها بر عدد ۲۵۵ آن‌ها نرمال می‌شوند.

سه پیش‌پردازش فوق با استفاده از قطعه کد زیر انجام می‌گیرد:

```
x_train = trainx.astype('float32')/255.0
x_test = testx.astype('float32')/255.0
y_train = to_categorical(trainy)
y_test = to_categorical(testy)
```

سپس ۱۰۰۰۰ نمونه از بخش آموزش مجوعه داده را به کمک تابع train_test_split به عنوان دادگان validation جدا می‌کنیم.

ج

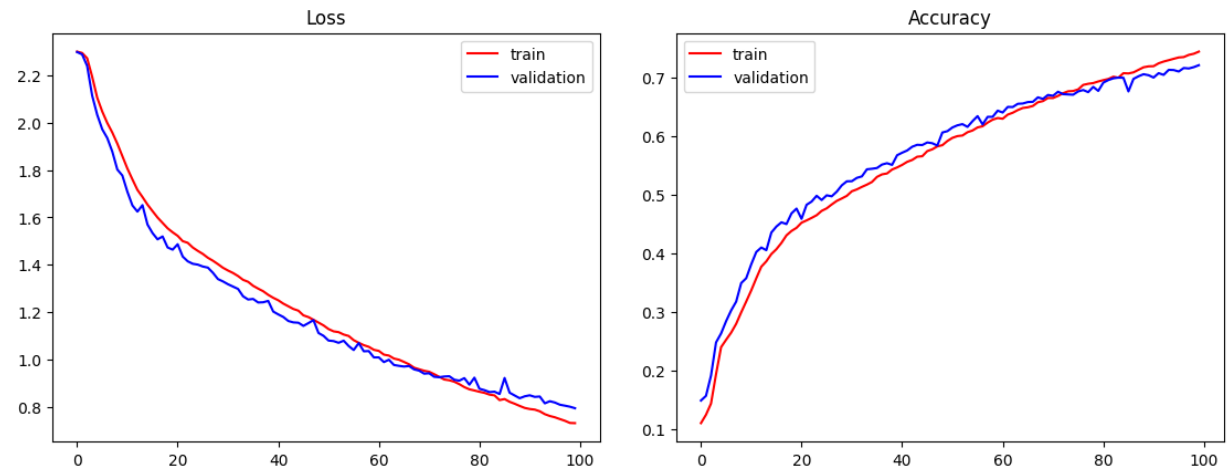
با استفاده از کتابخانه keras شبکه توصیف شده در قالب یک تابع با نام create_model پیاده‌سازی شد. تنها تفاوت مدل پیاده‌سازی شده با مدل توصیف شده در انجام عمل Flatten پیش از لایه‌های Dense است که با توجه به دو بعدی بودن لایه‌های Maxpooling و Conv2d و یک بعدی بودن لایه‌های Dense، نیاز به تبدیل خروجی لایه‌های قبلی به فرمتی مناسب برای ورود به لایه Dense وجود داشت. هایپرپارامترهای هر کدام از شبکه‌ها با Optimizer متفاوت که به صورت تجربی و با آموزش و خطا به دست آمده‌اند، به شکل زیر مقداردهی شده‌اند:

جدول ۱۳

Optimizer	SGD	Adam	RMSProp
Learning rate	$3*10^{-3}$	$3*10^{-5}$	$3*10^{-5}$
Epochs	100	100	100
Batch size	64	128	128

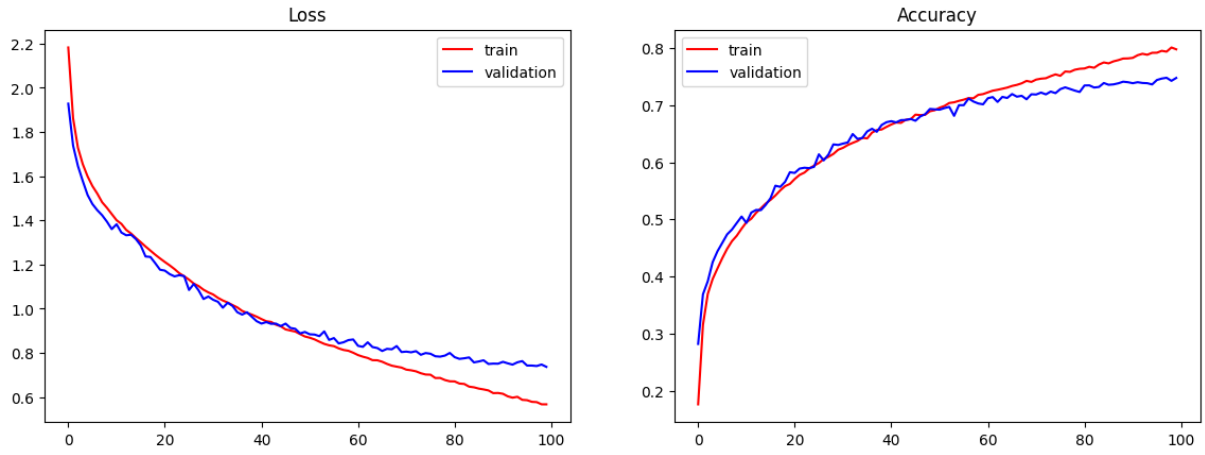
در ادامه نمودار خطا و دقت آموزش و اعتبارسنجی به دست آمده از هر کدام از مدل‌ها در طول آموزش با استفاده از Checkpointهای ذخیره شده‌ی مدل نمایش داده شده است.

SGD Optimizer:



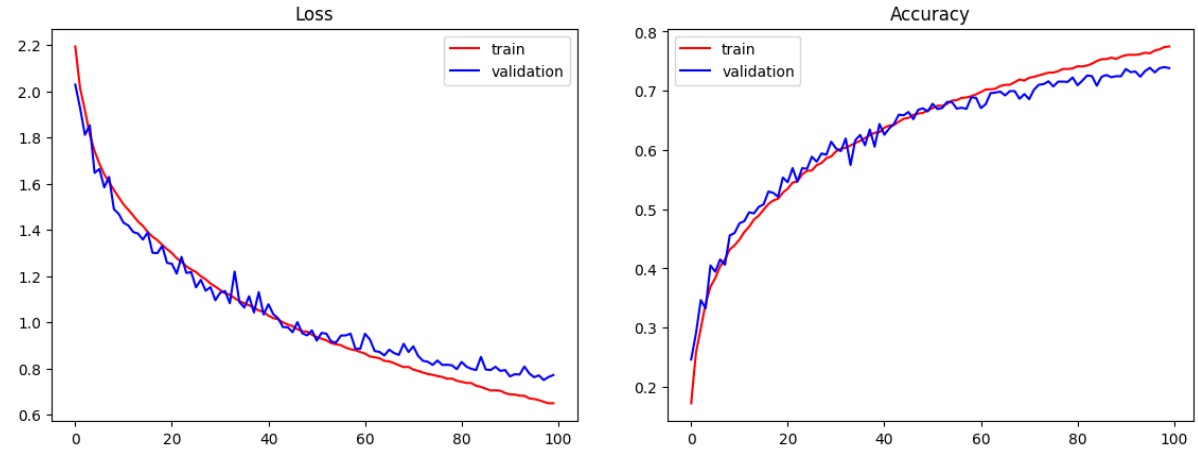
شکل ۲۳ SGD Optimizer

Adam Optimizer:



شکل ۲۴ *Adam Optimize*

RMSProp Optimizer:



شکل ۲۵ *RMSProp Optimizer*

در هر سه نمودار فوق مشخص است که از یک تکرار مشخص به بعد، رشد دقت و نزول Loss برای دادگان آموزشی از دادگان اعتبارسنجی پیش گرفته است. در صورت ادامه دادن این روند و استفاده از تعداد Epochهای بیشتر ممکن بود شاهد نزول دقت و افزایش Loss برای دادگان اعتبارسنجی شویم که به معنی Overfit شدن مدل است. نکته قابل توجه دیگر کاهش مداوم قدر مطلق مقدار شیب در هر دو نمودار برای دادگان آموزش و نویزی بودن نمودار برای دادگان اعتبارسنجی است و در نمودارهای دادگام آموزشی شاهد نویز بسیار کمی هستیم.

د

مقدار معیارهای ارزیابی خواسته شده برای هر یک از مدل‌ها بر روی دادگان تست به شرح زیر است:

جدول ۱۴

Optimizer	SGD	Adam	RMSProp
Accuracy	0.723	0.746	0.735
Precision	0.721	0.744	0.738
Recall	0.723	0.746	0.735
F1-Score	0.721	0.744	0.734

سوال ۹

الف

در ابتدا دادگان را به کمک کتابخانه Pandas در محیط پایتون بارگیری می‌کنیم و سپس با استفاده از تابع train_test_split از کتابخانه scikit-learn آن‌ها را با نسبت ۳۰/۷۰ به دو بخش آموزش و تست تقسیم می‌کنیم. پس از تقسیم داده‌ها با استفاده از کتابخانه scikit-learn یک درخت تصمیم آموزش می‌بینند. کد این بخش به شرح زیر است:

```
DT = DecisionTreeClassifier().fit(X_train, y_train)
```

سپس برچسب هر کدام از مجموعه داده‌های آموزش و تست بر اساس مدل آموزش دیده به دست می‌آید و مقدار دقت برای هر کدام از این مجموعه دادگان محاسبه شده است. دقت به دست آمده برای مجموعه دادگان آموزش و تست به شرح زیر است:

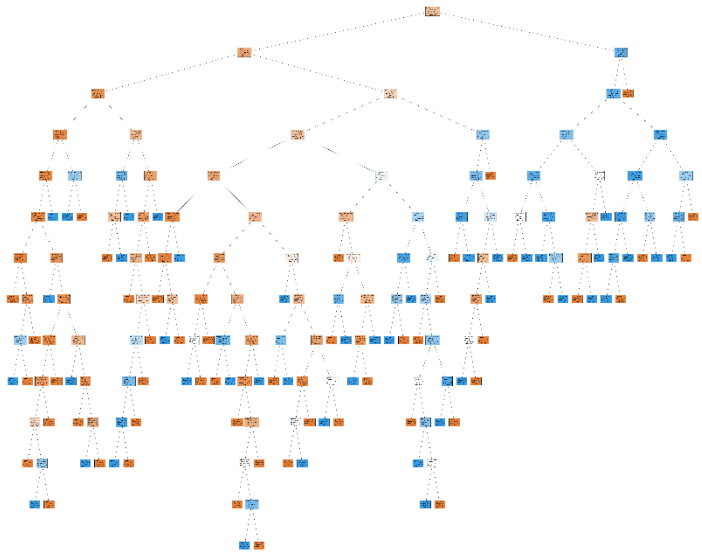
جدول ۱۵

دادگان	دقت
آموزش	۱.۰
تست	۰.۶۶۲

همانطور که مشاهده می‌شود دقت درخت تصمیم ایجاد شده بر روی مجموعه دادگان آموزش دارای بیشترین مقدار ممکن است اما بر روی دادگان تست دقت نسبتا پایینی برای یک طبقه‌بند دو کلاسه دارد. این اتفاق نشان دهنده رخ دادن Overfitting بر روی مدل استفاده شده است. به عبارتی دیگر درخت تصمیم ایجاد شده دادگان آموزشی را حفظ کرده است و توانایی Generalization ندارد و در نتیجه بر روی دادگان تست عملکرد خوبی از خود نشان نمی‌دهد.

ب

با استفاده از تابع plot_tree از کتابخانه scikit-learn درخت تصمیم ایجاد شده به شکل زیر رسم شده است:



شکل ۲۶

همانگونه که مشاهده می‌شود درخت تصمیم رسم شده بسیار پیچیده است و دارای شاخه‌های بسیاری است که این موضوع نیز تاییدی بر Overfit شدن این مدل است. لازم به ذکر است که محتوای گره‌های این درخت در هیچ ابعادی قابل مشاهده نیست و این شکل تنها به منظور مشاهده ساختار درخت ضمیمه شده است.

ج

Pre-pruning تکنیکی است که برای جلوگیری از رشد بیش از اندازه درختان تصمیم پیش از اینکه بیش از حد پیچیده شود و منجر به Overfit شدن شود، استفاده می شود. در Pre-pruning، الگوریتم درخت تصمیم به گونه‌ای بهینه‌سازی می‌شود که در صورت تحقق یک شرط خاص، الگوریتم متوقف شود. شروط متفاوتی را برای این روش می‌تواند متصور شد که چند مورد آن در ادامه معرفی می‌شوند:

۱. حداکثر عمق: وقتی درخت به عمق معینی رسید.
۲. حداقل نمونه: وقتی تعداد نمونه‌های گره به کمتر از یک آستانه معین رسید.
۳. حداکثر ویژگی: تعداد ویژگی های در نظر گرفته شده برای ایجاد شاخه‌های جدید در هر گره محدود شوند.
۴. آستانه سود: وقتی اطلاعات حاصل از تقسیم یک گره، به زیر یک آستانه معین رسید، ادامه دادن آن شاخه متوقف شود.

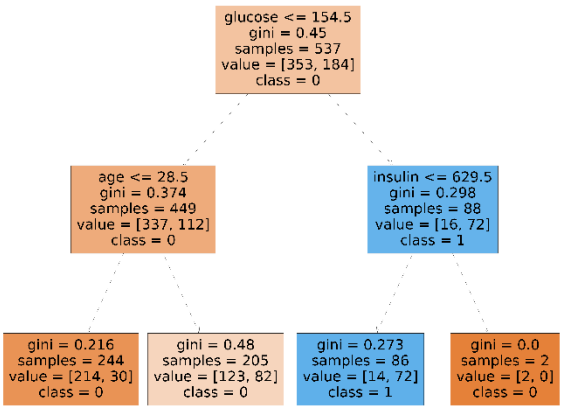
د

به صورت مشابهی با بخش الف و تنها با یک پارامتر بیشتر که مقدار حداکثر عمق مجاز درخت را نشان می‌دهد، درختی با عمق ۲ طراحی می‌شود و مقادیر دقت بر روی دادگان آموزش و تست به شکل زیر به دست آمده است:

جدول ۱۶

دادگان	دقت
آموزش	۰.۷۶۵
تست	۰.۷۰۹

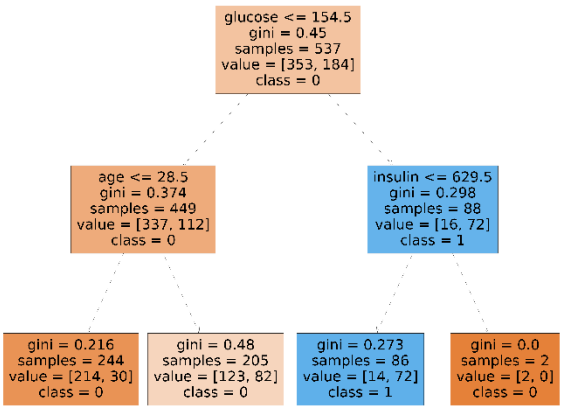
همانطور که مشاهده می‌شود مقدار دقت بر روی دادگان آموزشی کاهش یافت اما دقت دادگان تست بهتر از پیش شد. در حالت کلی استفاده از مدل ساده‌تری که دقت تست بهتری دارد را به مدلی که بر روی دادگان آموزشی دقت خوبی به دست می‌آورد اما عملکرد مناسبی بر روی دادگان تست ندارد ارجحیت می‌دهیم. همچنین درخت تصمیم ایجاد شده به شکل زیر است:



شکل ۲۷

۵

درخت بخش د به شکل زیر می‌باشد(در شکل ۲۷ نیز ذکر شده است):



شکل ۲۸

درخت بخش ب تا حدی زیادی توسعه یافته است و دارای شاخه‌های بسیار زیادی است که درک آن را بسیار مشکل می‌کند. همچنین هنگامی که بخواهیم کلاس یک نمونه جدید را تشخیص دهیم زمان زیادی را برای مقایسه با شروط هر شاخه نیاز دارد در صورتی که درخت بخش د در بیشترین حالت تنها به دو مقایسه نیاز دارد. به صورت کلی هر چه درخت ساده‌تر باشد، احتمال Overfit شدن نیز کمتر می‌شود(کاهش خطای واریانس و افزایش خطای بایاس).