به نام خدا

دانشگاه شهید بهشتی

دانشکده علوم ریاضی

گزارش تمرین یک شبکه عصبی

زهرا دهقانی تفتی (۹۶۲۲۲۰۳۷)

آبان ۹۹

فهرست مطالب

راه حل و ایده‌های کلی

در این مسئله که داده‌های ما مربوط به اطلاعات مشتریان یک بانک است، هدف ما این است که پیش‌بینی کنیم مشتریان در بانک می‌مانند یا بانک را ترک می‌کنند؛ پس ما با یک مسئله binary classification سر و کار داریم و جواب یا صفر است که یعنی مشتری بانک را ترک نمی‌کند و یا جواب یک است که یعنی بانک را ترک می‌کند.

داده‌ی ما شامل ۱۰۰۰۰ سطر و ۱٤ ستون است. هر سطر مربوط به یک مشتری است و هر ستون نشان دهنده‌ی ویژگی‌های مشتریان است مثلا سن مشتری، جنسیت مشتری و ...

در ابتدا نیاز است کتابخانه‌های مورد نیاز مانند numpy برای نمایش بردارها، کتابخانه‌ی pandas برای دریافت ورودی‌ها، کتابخانه‌ی matplotlib برای رسم نمودارهای دقت و تابع هزینه، کتابخانه seaborn که از کتابخانه matplotlib برای نمایش گراف‌ها و نمودارها استفاده می‌کند و همچنین کتابخانه‌های tensorflow و keras که برای ساخت شبکه عصبی استفاده می‌شود را فراخوانی کنیم.

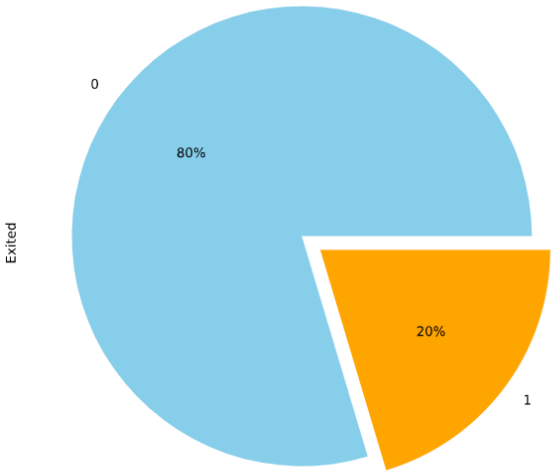
در ابتدا نیاز است که داده‌ها را که در فایل csv هستند به کمک pandas فراخوانی کنیم.بعد از گرفتن داده‌ها نیاز است preprocessing انجام شود و روی داده‌ها تغییراتی انجام شود تا تمیزتر و قابل استفاده شوند.

در ابتدا نیاز است که بدانیم آیا ستونی وجود دارد که مقدار خالی یا null داشته باشد، اینکار را با دستور dataset.isnull().sum() انجام می‌دهیم که تعداد مقادیر خالی برای هر ستون را نمایش می‌دهد. مشاهده می‌شود هیچ ستونی خالی نیست. اگر مقدار خالی وجود داشت چند راهکار وجود دارد که یا می‌توان آن سطر شامل مقدار خالی را کامل از داده‌ها پاک کرد که این در صورتی است که آن ویژگی اهمیت زیادی در جواب خروجی نداشته باشد و یا می‌توان آن مقدار خالی را با مقادیری مثل میانگین آن ویژگی پر کرد.

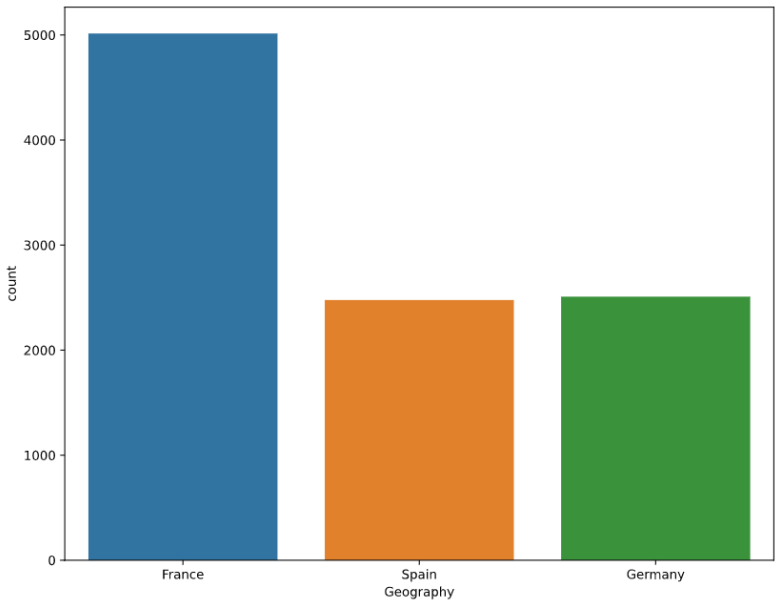
یک چیز دیگری که باید برای تمیز کردن داده‌ها چک شود؛ وجود مقادیر تکراری در داده‌ها است که در صورت وجود باید حذف شود. بعد از این کار تعداد مقادیر منحصر به فرد برای هر ویژگی شمرده می‌شود که با توجه به آن می‌شود فهمید که مثلا ویژگی سن و جنسیت تنها دو مقدار می‌گیرندو احتمالا از نوع ویژگی‌های دسته بندی می‌باشند. بعد از اینکار باید داده‌های ورودی (x) و داده‌های خروجی (y) مشخص شوند و در همین حین داده‌هایی که در خروجی تاثیری نمی‌گذارند نیز حذف شود مثلا در اینجا ۳ ستون اول داده‌ها (RowNumber, CustomerId, Surname) را حذف می‌کنیم زیرا مقادیر این ستون‌ها طبق شمارش مقادیر منحصر به فرد برای هر ستون، تقریبا متفاوت است و هیچ تاثیری در خروجی نمی‌گذارند. (اسم یک فرد، شماره مشتری و شماره سطر تاثیری در ماندن یا ترک کردن بانک ندارد.)

سپس با رسم یکسری نمودارها می‌خواهیم درک خود را از داده‌ها بیشتر کنیم. در صفحه بعد تاثیر یک‌سری از ویژگی‌هایی که جز ویژگی‌های دسته بندی هستند را روی خروجی به نمایش می‌گذاریم.

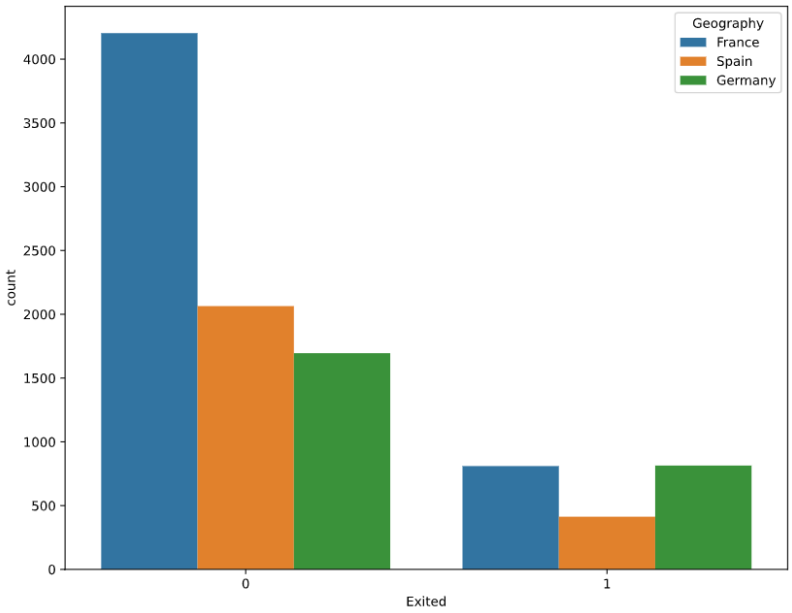
نمودار زیر نسبت تعداد مشتریان ترک کننده‌ی بانک و مشتریانی که هنوز بانک را ترک نکرده‌اند را نشان می‌دهد. همانطور که می‌بینیم، ۲۰ درصد مشتریان بانک را ترک می‌کنند.



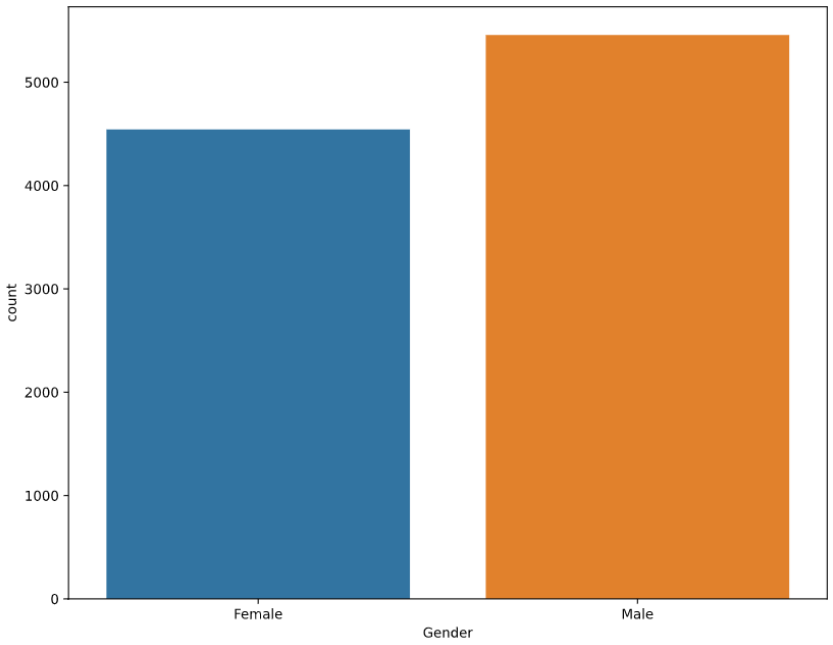
نمودار زیر تعداد مشتریان بانک از کشورهای مختلف را نشان می‌دهد. همانطور که مشخص است تعداد مشتریان فرانسوی دو برابر تعداد هریک از مشتریان آلمانی و اسپانیایی هستند و تعداد مشتریان آلمانی و اسپانیایی با هم برابر است.



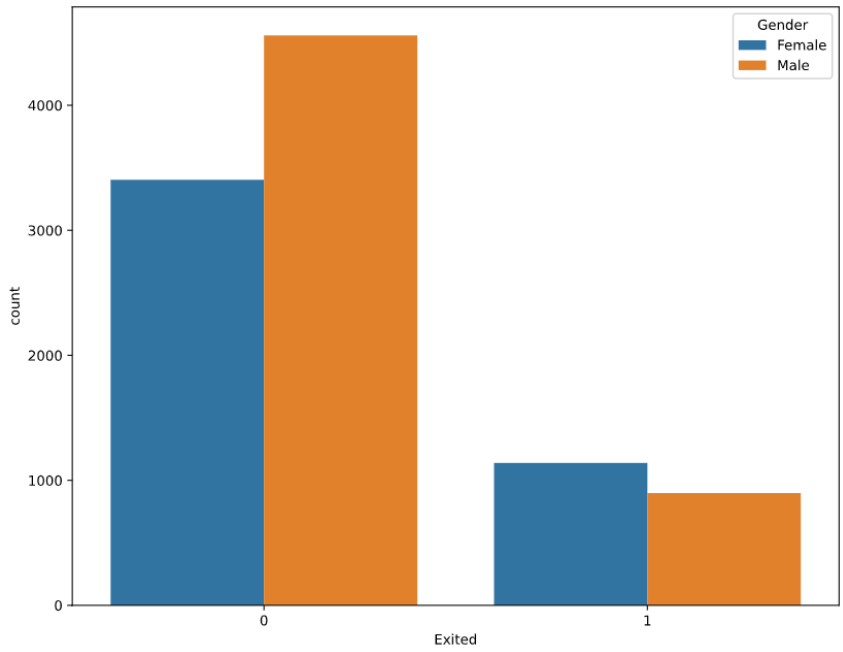
نمودار زیر نسبت خروج مشتریان را با توجه به کشوری که در آن هستند نشان می‌دهد. همانطور که مشخص است تعداد مشتریان فرانسوی و آلمانی که بانک را ترک می‌کنند با هم برابر است و بیشتر از تعداد مشتریان اسپانیایی است که بانک را ترک می‌کنند.



نمودار زیر تعداد مشتریان مرد و مشتریان زن بانک را نشان می‌دهد و همانطور که مشخص است بیشتر مشتریان مرد هستند.



در نمودار زیر نسبت خروج مشتریان بانک را با توجه به جنسیتی که دارند نشان می‌دهد.همانطور که می‌بینیم مشتریان زن بیشتر از مشتریان مرد بانک را ترک می‌کنند.



با استفاده از دستور describe می‌توان جزییات داده‌ها مانند میانگین هر ویژگی، مینیمم، چارک اول، میانه و ... را به نمایش گذاشت.

حال می‌خواهیم با استفاده از کتابخانه‌ی sklearn داده‌ها را به نوع مورد قبول برای شبکه‌های عصبی درآوریم به این صورت که ویژگی‌هایی را که از نوع object هستند را به شکل عدد درآوریم و آن‌ها را کد کنیم مثلا اگر ویژگی دو مقدرا زن و مرد می‌گیرد به آن عدد ۰ و ۱ نسبت دهیم.. دلیل این کار این است که داده‌های عددی به صورت مستقیم به شبکه عصبی داده می‌شوند ولی داده‌هایی که از نوع object باشند را باید کد کنیم تا بتواند توسط شبکه عصبی قابل دریافت باشد.

بعد از این کار باید داده‌ها را با استفاده از sklearn به دو بخش داده‌های آموزشی و داده‌های تست تقسیم کنیم به اینصورت که ۸۰ درصد داده‌ها را آموزشی در نظر می‌گیریم و ۲۰ درصد داده‌ها را تست در نظر می‌گیریم. در هنگام جداسازی داده‌ها باید random state را تنظیم کنیم تا داده‌ها تکرار پذیر باشند و هربار جواب متفاوت نگیریم. بعد از جدا کردن داده ها به دو بخش تست و آموزشی حال باید داده‌ها را rescale کنیم. برای اینکار طبق زیر عمل می کنیم:

$$X\_train = (x - \min(X\_train)) / (\max(X\_train) - \min(X\_test))$$

$$X\_test = (X\_test - \min(X\_train)) / (\max(X\_train) - \min(X\_test))$$

*اصلاحیه : اما در کدی که من زدم اینکار را انجام ندادم و ابتدا داده‌ها را استاندار کردم و سپس داده‌ها را جدا کردم در صورتی که اینکار غلط است چون در واقع ما به داده‌های تست دسترسی نداریم. و باید ابتدا داده‌ها به دوبخش تست و آموزشی جدا شوند و سپس استاندارد شوند.

استاندارد سازی یکی از کارهای مهمی است که در یادگیری ماشین باید انجام شود و دلیل آن این است که می‌خواهیم متغیرهای مختلف را در یک مقیاس قرار دهیم و بتوانیم آن‌ها را با هم مقایسه کنیم.

بعد از preprocessing وارد طراحی شبکه عصبی برای این مسئله می‌شویم. شبکه عصبی که برای اینکار استفاده می‌کنیم از نوع multi layer perceptron است که یک سری ورودی می‌گیرد و با ترکیب کردن این وِرودی‌ها و کانکشن ها در لایه‌های مختلف به خروجی می‌رسیم.

شبکه عصبی ما سه نوع لایه دارد که عبارتند از لایه ی ورودی، لایه مخفی و لایه خروجی

برای ساخت شبکه عصبی و اضافه کردن لایه‌های ورودی،خروجی و مخفی باید کتابخانه‌های مورد نیاز مانند tensorflow و keras را فراخوانی کنیم. در این شبکه عصبی من به عنوان optimizer، Adam را در نظر گرفته ام زیرا در بین بقیه‌ی optimizer ها در اکثر موارد بهترین است. سپس با اضافه کردن لایه های مخفی و نورون‌های لایه‌ی مخفی، تغییر تابع فعالسازی برای هر لایه، تغییر وزن ورودی، تغییر نرخ یادگیری، تغییر سایز batch و تغییر epoch ها سعی در آزمون و خطا برای رسیدن به مدل داریم.

در اینجا ما در تابع کامپایل، binary crossentropy را به عنوان تابع هزینه در نظر گرفته‌ایم زیرا خروجی من به صورت ۰ و ۱ است. در تابع کامپایل متریک را دقت قرار می‌دهیم تا بتوانیم نحوه‌ی عملکرد شبکه را ببینیم.

در تابع fit باید اطلاعاتی مانند سایز دسته،تعداد epoch و داده‌های validation داده شود.

*اصلاحیه : من در تابع fit برای شبکه‌های عصبی چون نمی‌دانستم داده‌های تست را به عنوان validation در نظر گرفتم در صورتی که می دانیم در هنگام ساخت مدل ما به داده‌های تست دسترسی نداریم و باید یک بخشی از داده‌های آموزشی را به عنوان داده validation جدا کنیم تا تقریبی از خطا برای داده‌های تست بدست آوریم. در واقع داده‌های اولیه ما به سه قسمت تقسیم می‌شوند: ٦٠ درصد داده‌های آموزشی، ٢٠ درصد داده‌های validation، ٢٠ درصد داده‌های تست

برای هر مدل وزن‌ها چاپ می‌شوند که نحوه‌ی چاپ شدن آن‌ها به صورت است که ابتدا وزن لایه‌ی اول چاپ می‌شود سپس bias اول چاپ می‌شود و به همین ترتیب برای لایه‌های بعد

همچنین برای هر مدل پیش‌بینی انجام می‌شود و برحسب آن ماتریس کانفیژن مربوط به آن چاپ می‌شود که درک بهتری از مدل به ما می‌دهد. بعد از آن نمودارهای مقایسه تابع هزینه و دقت برای داده‌های اموزشی و داده‌های validation را داریم.

وزن هر شبکه را با استفاده از تابعی که خود کراس دارد در یک فایل h5. ذخیره می‌کنیم که بعدا قابل load کردن باشد.

ارزیابی نتایج

در ابتدا از مدل ضعیف تر با تعداد لایه و نورون کم شروع می‌کنیم و تاثیر پارامترهای مختلف را روی نتایج بررسی میکنیم. در تمامی مدل‌ها از بهینه‌گر Adam استفاده شده زیرا این بهینه‌گر در اکثر مدل‌ها بهتر از بقیه‌ی بهینه‌گرها عمل می‌کند و ترکیبی از RMSprop و momentum می‌باشد.

آزمایش ۱:

شبکه عصبی ما ۱۰ نورون ورودی دارد زیرا ۱۰ ویژگی قابل استفاده برای کلاس بندی در داده های ما موجود است و در خروجی یک نورون داریم زیرا هدف ما کلاس بندی است و نتیجه ۰ یا ۱ می‌باشد.

در آزمایش اول با یک لایه ی مخفی و تعداد یک نورون در لایه‌ی مخفی شروع می‌کنیم.

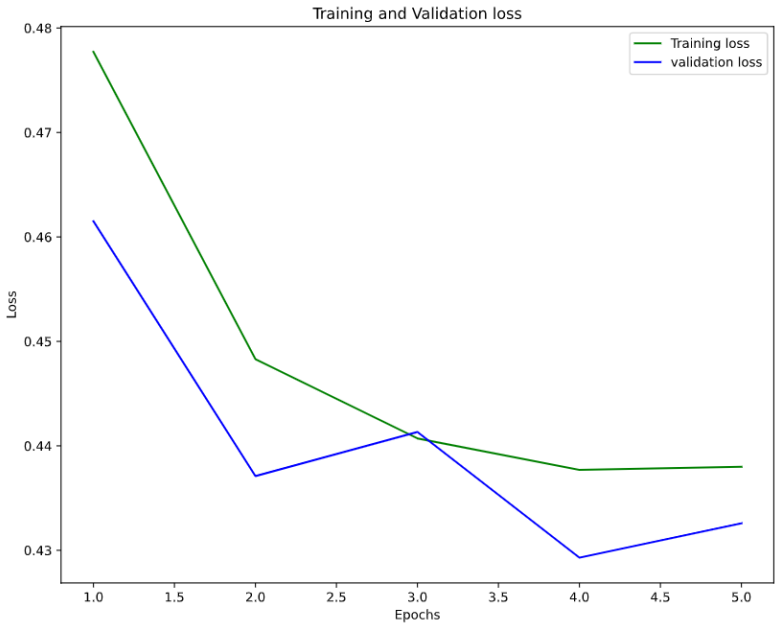
پارامترهای انتخاب شده:

epoch = 5, lr (learning rate) = 0.01, batch = 5, weight initializer = RandomUniform ,hidden_activation_function = Relu

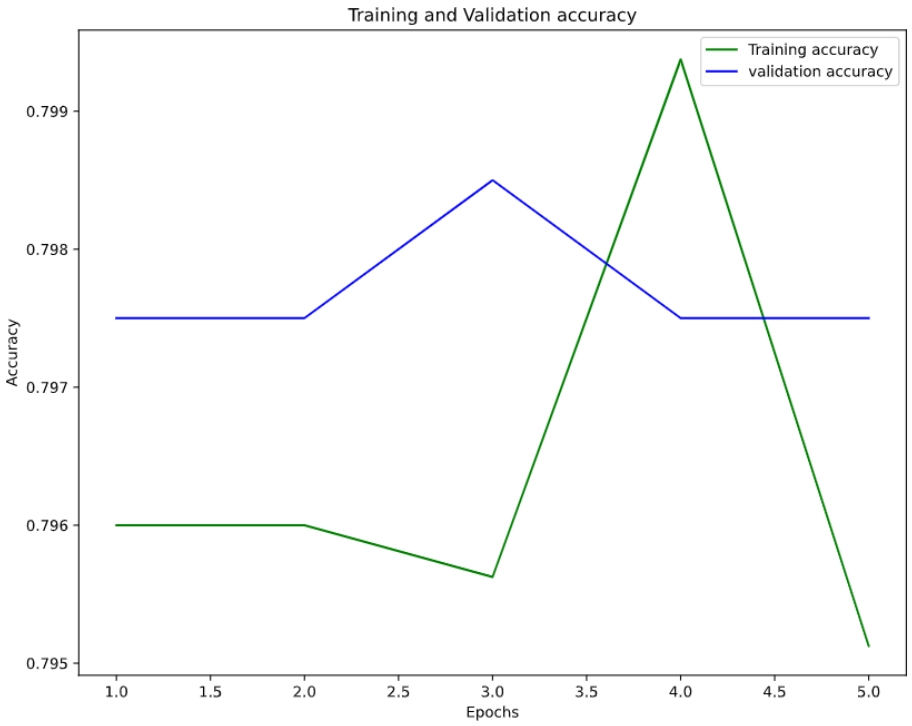
با پارامترهای انتخاب شده به نتایج زیر می‌رسیم.

loss: 0.4380 - accuracy: 0.7951 - val_loss: 0.4326 - val_accuracy: 0.7975

شکل زیر مقایسه‌ی تابع هزینه برای داده‌های آموزشی و داده‌های تست را نشان می‌دهد. شیب کم شدن تابع هزینه برای داده‌های اموزشی کم است و مقدار زیادی کم نمی‌شود که احتمالا نشان دهنده‌ی این است که lr کم است.



در شکل زیر مقایسه‌ی دقت داده‌های اموزشی و داده تست را می‌بینیم.



آزمایش ۲:

در این آزمایش lr را افزایش می‌دهیم تا روند کاهش تابع هزینه سرعت بیشتری داشته باشد و همچنین برای اینکه سرعت آموزش مدل افزایش پیدا کند سایز batch را نیز افزایش می‌دهیم.

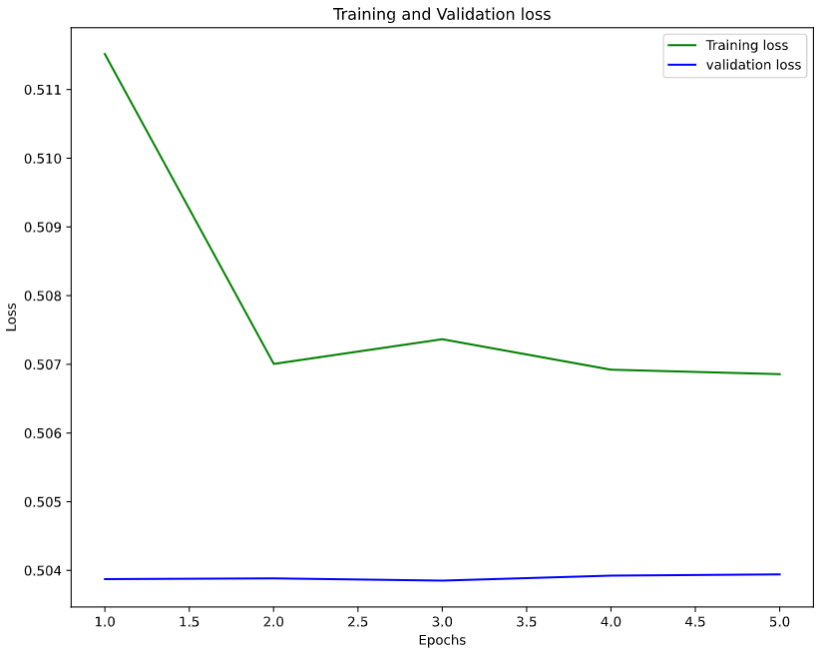
پارامترهای انتخاب شده:

epoch = 5, lr (learning rate) = 0.03, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

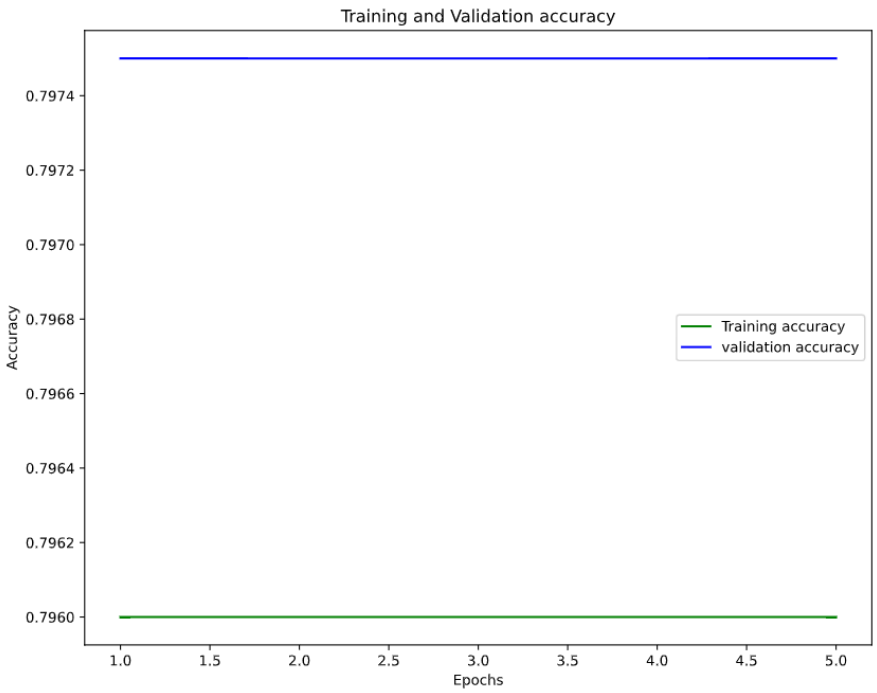
با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

loss: 0.5069 - accuracy: 0.7960 - val_loss: 0.5039 - val_accuracy: 0.7975

مشاهده می‌شود تابع هزینه هم برای داده‌های آموزشی و هم برای داده های تست، نسبت به آزمایش ۱ بدتر شده.شکل زیر که مقایسه‌ی تابع هزینه‌ی داده های آموزشی و داده‌های تست می‌باشد نشان می‌دهد که lr زیاد است چون با شیب خیلی تیزی کم شده است. پس در مرحله‌ی بعد باید این نرخ یادگیری کمتر شود بین نرخ یادگیری آزمایش ۱ و ۲ قرار بگیرد. ولی در این مرحله چون سایز batch بزرگتر شده بود سرعت یادگیری افزایش پیدا کرده است.



شکل زیر مربوط به دقت داده های آموزشی و داده‌های تست می‌باشد.



آزمایش ۳:

در این مرحله با توجه به نتیجه مرحله‌ی قبل lr را کاهش می‌دهیم و برای افزایش دقت تعداد epoch ها را افزایش می‌دهیم.

پارامترهای انتخاب شده:

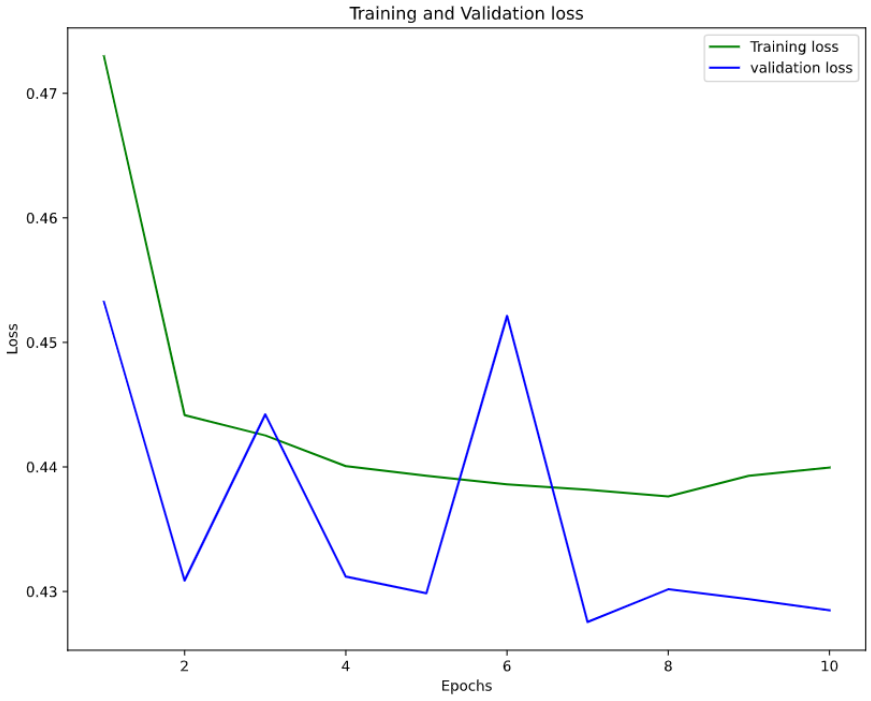
epoch = 10, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

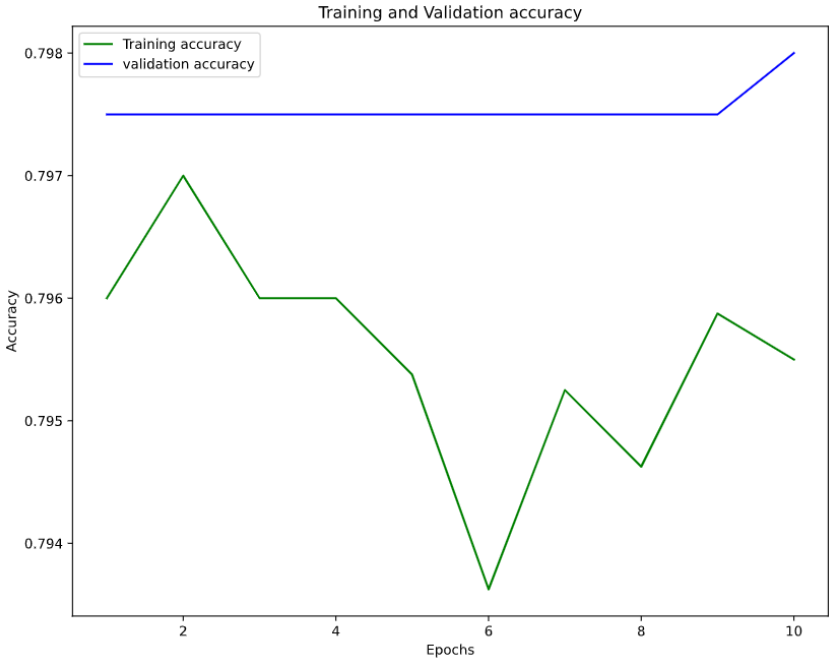
–loss: 0.4400 - accuracy: 0.7955 - val_loss: 0.4285 - val_accuracy: 0.7980

در این مرحله به نظر می‌رسد که lr بهتر شده اما هنوز باید کمتر شود. دقت نسبت به قبل تغییر محسوسی نداشته است ولی تابع هزینه کمتر شده است. طبق نمودار دقت که برای این مدل بدست آوردیم، دقت داده‌های تست بیشتر از داده‌های آموزشی است و در نمودار هزینه، هزینه‌ی داده‌های تست نوسان زیادی دارد که به نظر می‌رسد مدل underfit شده است و پیچیدگی مدل ما کمتر از مسئله است. یک راه حل ممکن افزایش تعداد نورون‌ها در تنها لایه ی مخفی‌ای است که داریم و باعث پیچیده‌تر شدن مسئله می‌شود.

شکل زیر مقایسه‌ی تابع هزینه برای داده‌های آموزشی و داده‌های تست می‌باشد.



شکل زیر مقایسه‌ی دقت داده‌های آموزشی و داده‌های تست می‌باشد



آزمایش ٤:

در این آزمایش با توجه به آزمایش قبل برای اینکه underfit نشویم پیچیدگی مدل را بیشتر می‌کنیم و تعداد نورون‌های لایه‌ی مخفی را بیشتر می‌کنیم.

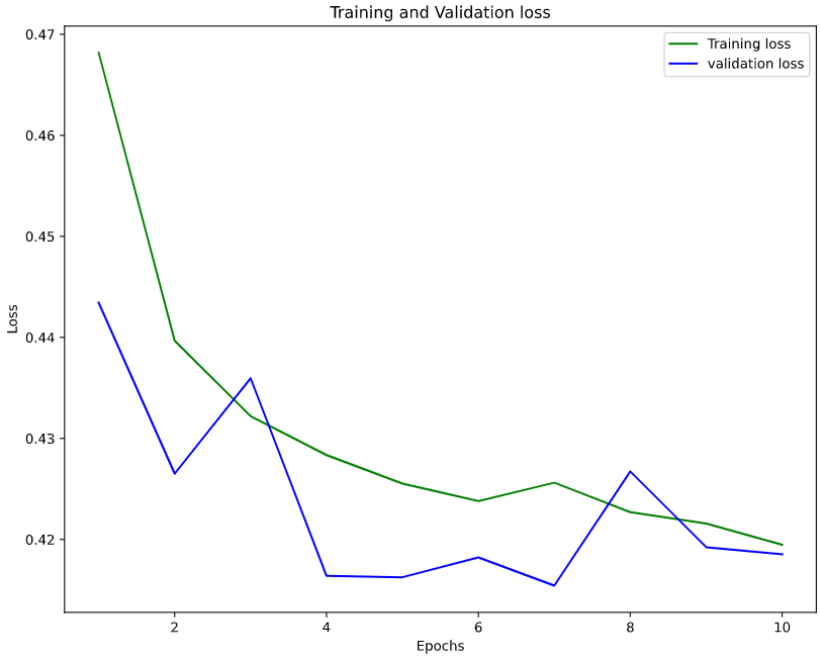
پارامترهای انتخاب شده:

epoch = 10, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

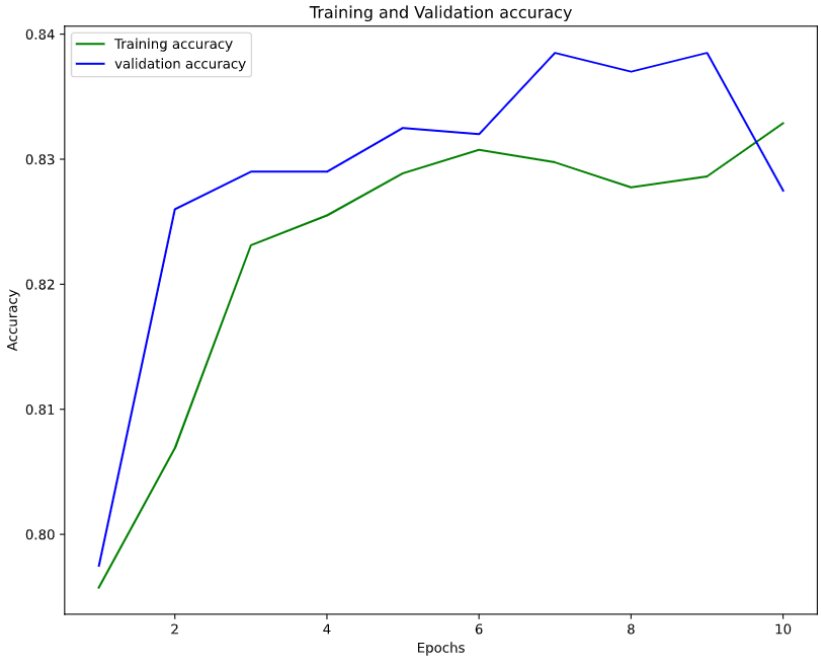
با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

loss: 0.4195 - accuracy: 0.8329 - val_loss: 0.4185 - val_accuracy: 0.8275

شکل زیر مقایسه‌ی تابع هزینه برای داده‌های آموزشی و داده‌های تست را نشان می‌دهد.و همانطور که می‌بینیم نوسان داده‌های تست کمتر شده ولی همچنان نوسان دارد و باید مدل پیچیده‌تر شود. و lr برای این آزمایش مناسب است.



شکل زیر مقایسه‌ی دقت داده‌های اموزشی و داده‌های تست می‌باشد و همانطور که مشخص است همچنان دقت داده‌های آموزشی کمتر از داده‌های تست است پس در مرحله بعد مدل باید پیچیده تر شود تا بهتر عمل کند. ولی نسبت به آزمایش قبل دراین آزمایش دقت بیشتر شده است که بنظر می‌رسد به خاطر پیچیده تر شدن مدل و فیت شدن بهتر آن بر روی داده‌ها است.



آزمایش ۵:

در این آزمایش با توجه به آزمایش قبل برای اینکه مدل پیچیده‌تر شود، تعداد نورون‌های لایه‌ی مخفی را بیشتر و از ۵ نورون به ۱۰ نورون افرایش می‌دهیم.

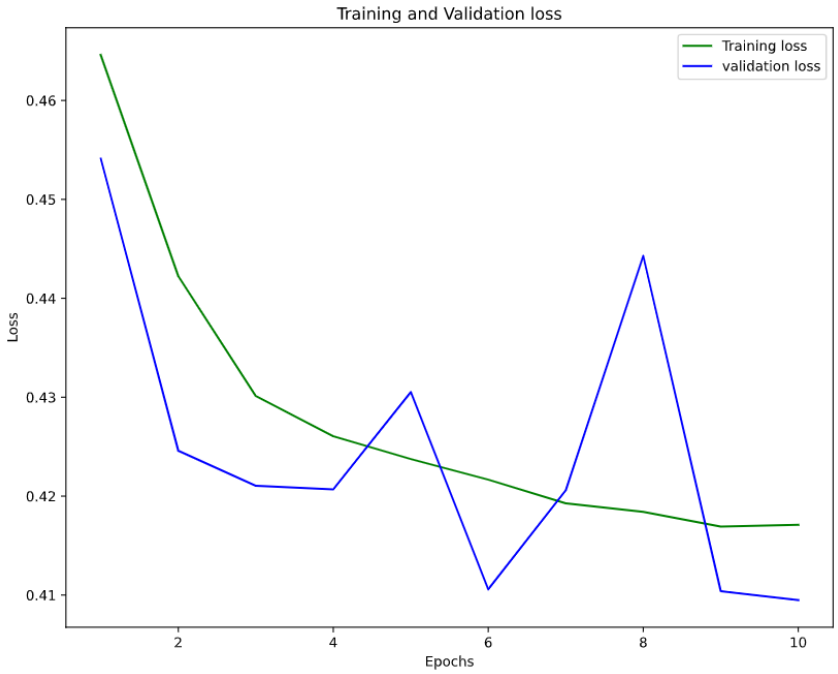
پارامترهای انتخاب شده:

epoch = 10, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

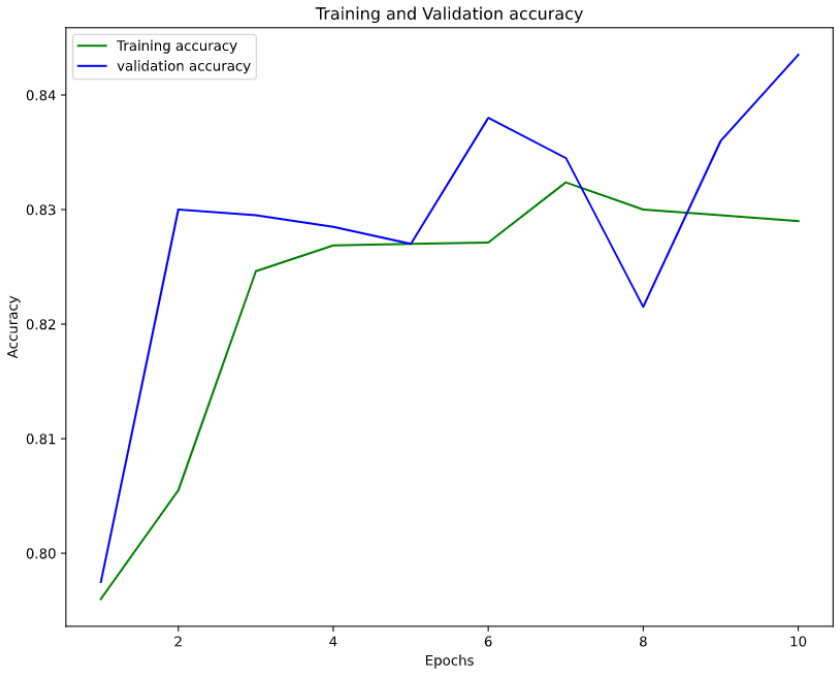
با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

loss: 0.4171 - accuracy: 0.8290 - val_loss: 0.4095 - val_accuracy: 0.8435

شکل زیر مقایسه‌ی تابع هزینه برای داده‌های آموزشی و داده‌های تست است، همچنان تابع هزینه برای داده‌های تست نوسان زیادی دارد و هنوز از underfit خارج نشده‌ایم. بنظر می‌رسد اگر یک لایه‌ی مخفی دیگر اضافه شود بهتر باشد و مدل پیچیده‌تر شود.اینگونه به نظر می‌رسد که چون مدا پیچیدگی کمی دارد نمی‌تواند داده‌ها را درک کند به همین دلیل روی داده‌های تست نوسان دارد.



شکل زیر مربوط به دقت داده‌های آموزشی و داده‌های تست است.



آزمایش ٦:

در این آزمایش با توجه به آزمایش قبل برای اینکه مدل پیچیده‌تر شود، یک لایه‌ی مخفی دیگر به این مدل اضافه می‌کنیم و در آن لایه یک نورون قرار می‌دهیم.

پارامترهای انتخاب شده:

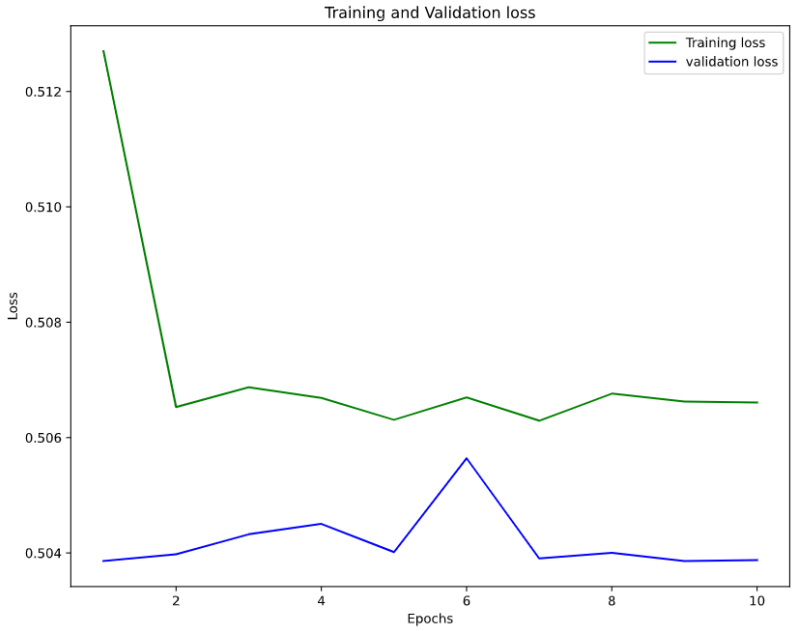
epoch = 10, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

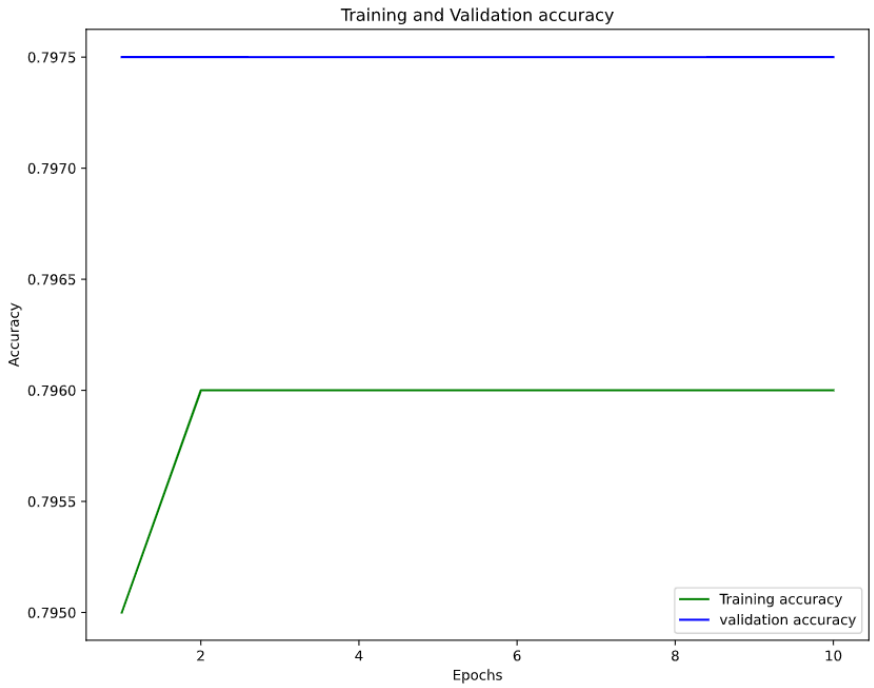
loss: 0.5066 - accuracy: 0.7960 - val_loss: 0.5039 - val_accuracy: 0.7975

در این آزمایش نسبت به آزمایش قبل دقت بدتر شد همچنین تابع هزینه نیز بیشتر و بدتر شد. طبق شکل زیر نوسان تابع هزینه داده‌های آموزشی کمتر شد ولی طبق شکل مقایسه‌ی دقت‌ها همچنان دقت روی داده‌های آموزشی نسبت به داده‌های تست کمتر است و مدل برای فهم داده‌های آموزشی باید پیچیده‌تر شود. همچنین تابع هزینه داده‌های آموزشی با شیب خیلی تیزی کم شده و نوسان کمی دارد که نشان از زیاد بودن lr دارد.

مقایسه‌ی تابع هزینه داده‌های آموزشی و داده‌های تست.



مقایسه‌ی دقت داده‌های آموزشی و داده‌های تست.



آزمایش ۷:

با توجه به نتیجه ای که از آزمایش قبل گرفتیم، اینجا تعداد نورون‌های لایه‌ی مخفی دوم را به ۵ افزایش میدهیم و برای افزایش دقت تعداد epoch ها را نیز افزایش می‌دهیم؛ همچنین lr را برای کم کردن نوسان تابع هزینه داده‌های آموزشی کم می‌کنیم.

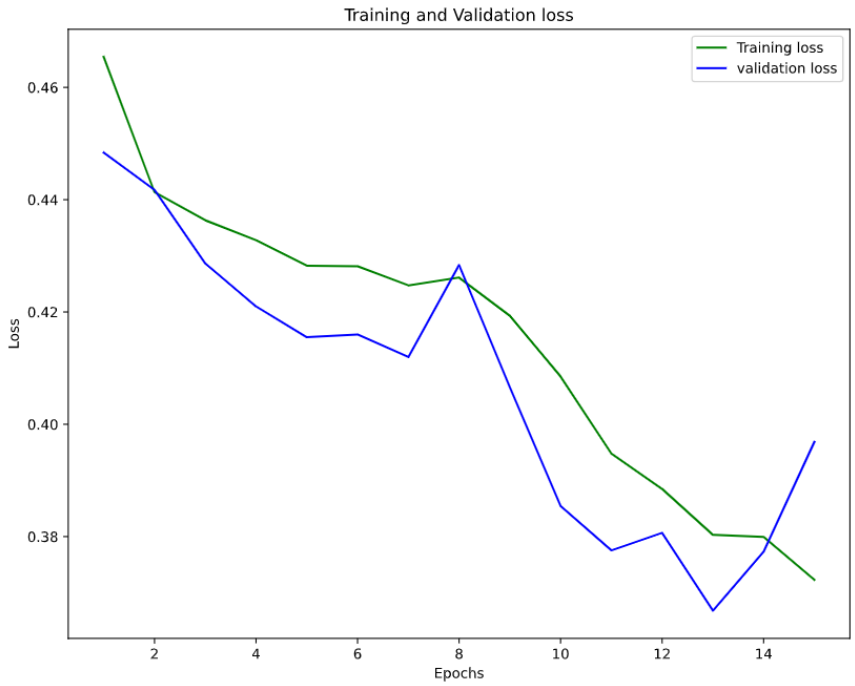
پارامترهای انتخاب شده:

epoch = 15, lr (learning rate) = 0.015, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

loss: 0.3723 - accuracy: 0.8484 - val_loss: 0.3969 - val_accuracy: 0.8320

در این آزمایش دقت داده‌های آموزشی و داده‌های تست نسبت به آزمایش قبل افزایش پیدا کرد و همچنین تابع هزینه کاهش پیدا کرد و بهتر شد. با توجه به نمودار زیر که مقایسه‌ی تابع هزینه‌ی داده‌های آموزشی و داده‌های تست است، متوجه می‌شویم نوسان داده‌های تست کمی کمتر شده اما همچنان نوسان دارد. در این نمودار تابع هزینه‌ی داده‌های آموزشی با سرعت کمی کم شده، پس lr برای این مدل کم است زیرا باید با پیچیده‌تر شدن مدل lr هم افزایش یابد.



با توجه به نمودار زیر باز هم مدل نیاز به پیچیده‌تر شدن دارد تا دقت روی داده‌های آموزشی افزایش پیدا کند. پس در مرحله‌ی بعد تعداد نورون‌های لایه‌ی مخفی دوم را افزایش می‌دهیم.

آزمایش ۸:

با توجه به نتیجه آزمایش قبل در این آزمایش نورون‌های لایه مخفی دوم را به ۱۰ افزایش می‌دهیم و lr را افزایش می‌دهیم.

پارامترهای انتخاب شده:

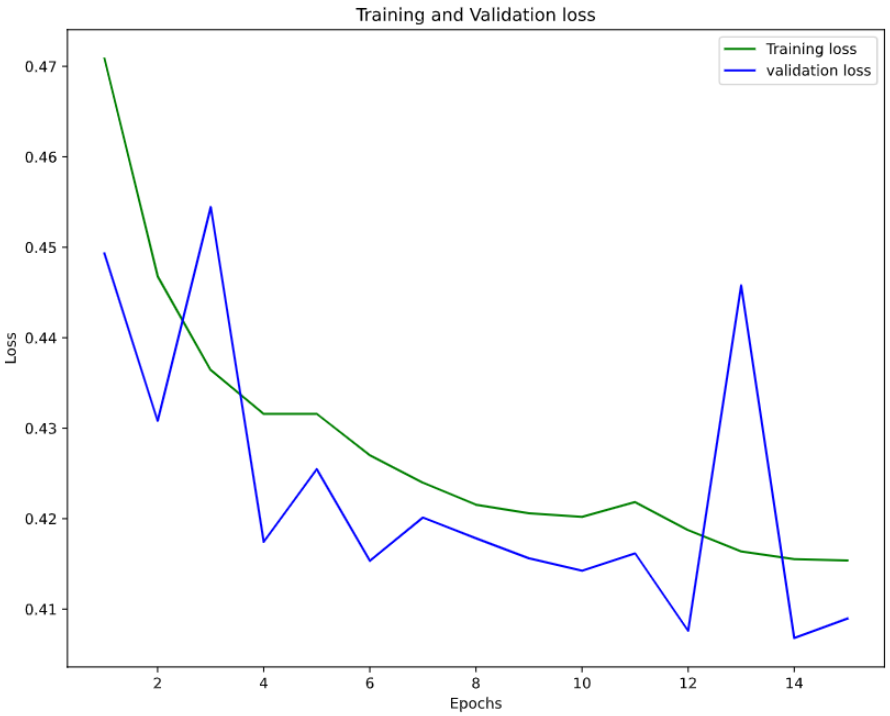
epoch = 15, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

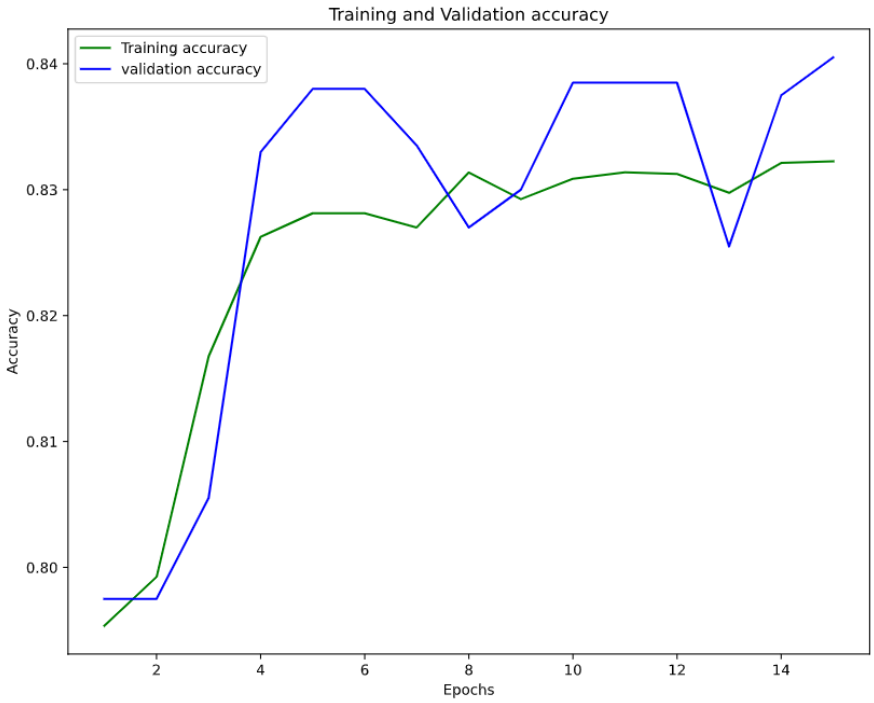
loss: 0.4154 - accuracy: 0.8322 - val_loss: 0.4090 - val_accuracy: 0.8405

با توجه به نمودارهای بدست آمده باز هم مدل باید پیچیده‌تر شود. در اینجا lr مناسب است و با شیب خوبی هزینه کم شده.و با توجه به نوسان های زیاد به نظر می‌رسد مدل روی داده‌های آموزشی overfit شده است.

نمودار مقایسه تابع هزینه برای داده‌های آموزشی و داده‌های تست:



نمودار مقایسه دقت برای داده‌های تست و آموزشی:



آزمایش۹:

با توجه به نتیجه آزمایش قبلی اینبار ما تعداد epoch ها را بیشتر کرده و تعداد نورون‌های دو لایه‌ی مخفی را هرکدام به ۲۰ می‌رسانیم.

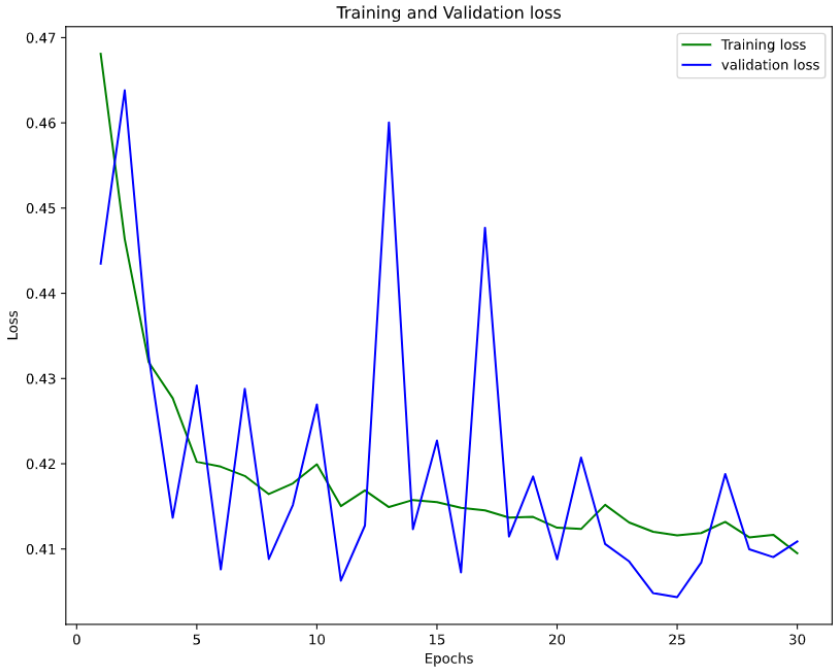
پارامترهای انتخاب شده:

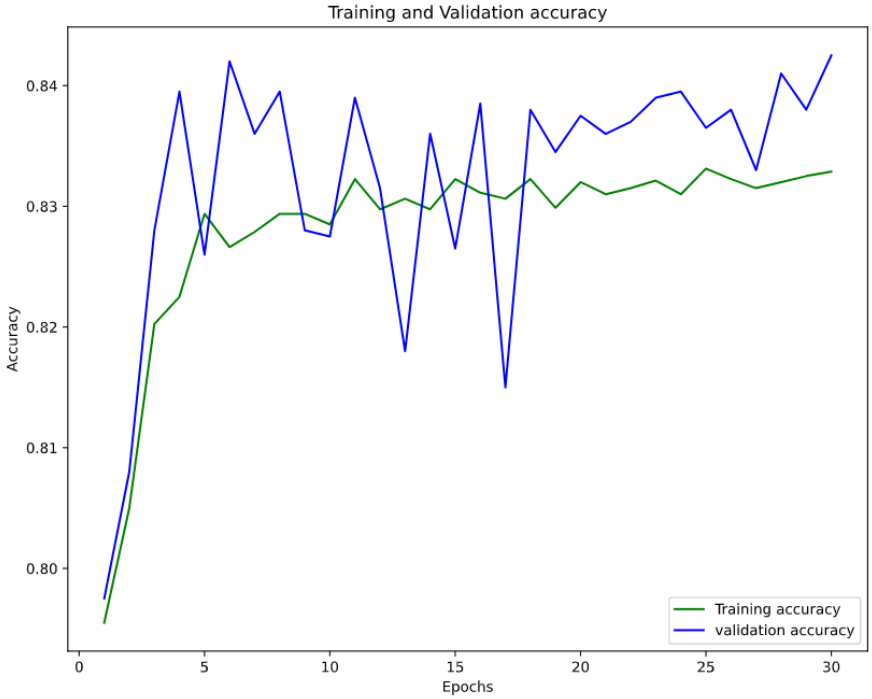
epoch = 15, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomUniform ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

loss: 0.4095 - accuracy: 0.8329 - val_loss: 0.4109 - val_accuracy: 0.842

طبق نمودارهای بدست آمده از این آزمایش هنوز هم تابع هزینه‌ی داده‌های تست نوسان خیلی زیادی دارد و احتمالا overfit شده که نمی‌تواند روی داده‌های تست دقت خوبی داشته باشد و هزینه اش را کم کند.





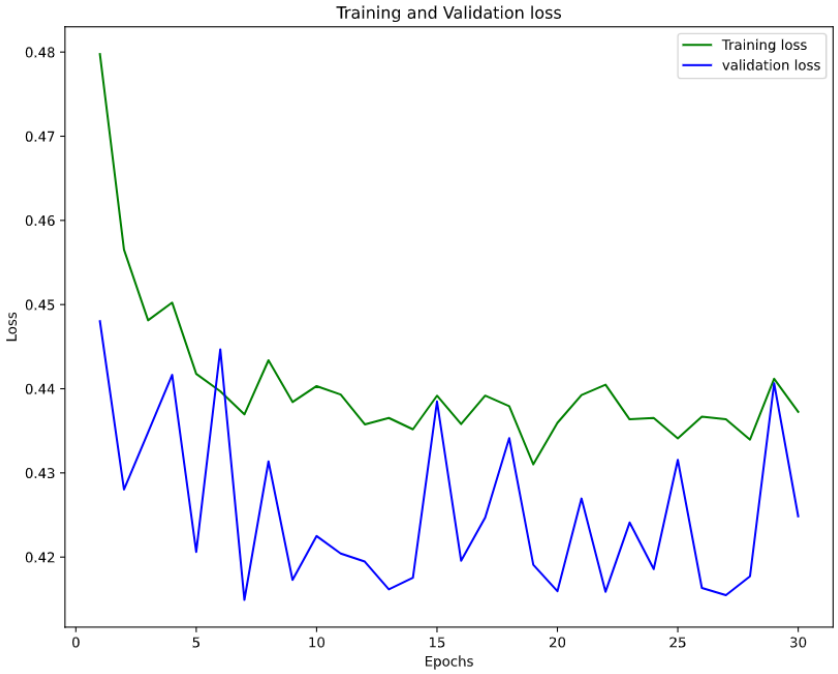
آزمایش ۱۰:

با توجه به نمودارهای قبلی، احتمال اینکه مدل روی داده‌های آموزشی overfit شده باشد وجود دارد، زیرا بر روی داده‌های تست خیلی نوسان دارد و در یک سری از epoch ها تابع هزینه برای داده‌های تست خیلی زیاد شده، بنابراین در اینجا از droupout استفاده می‌کنیم تا یک سری از نورون‌ها off شوند.

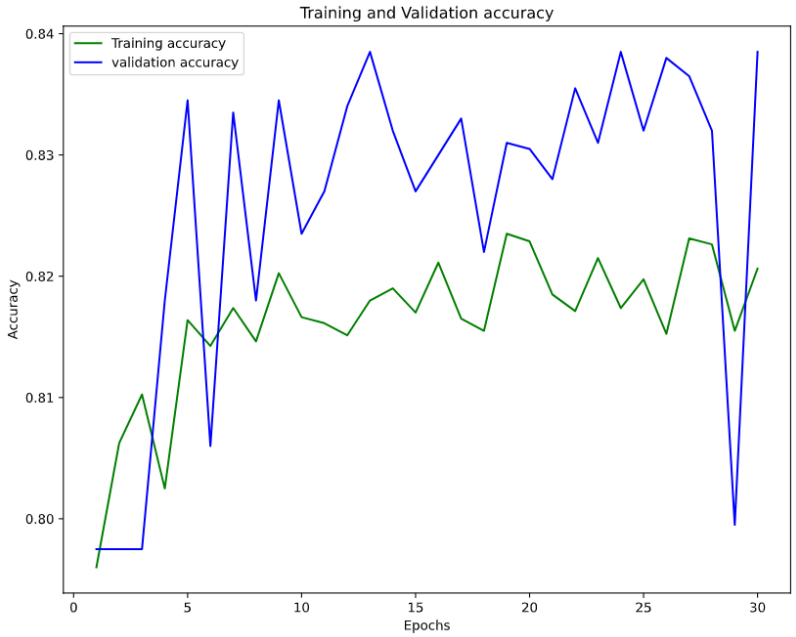
در این مرحله فقط از droupout با نرخ 0.1 استفاده می‌کنیمو در پارامترهای دیگر تغییر نمی‌دهیم. به نتیجه‌ی زیر می‌رسیم.

loss: 0.4373 - accuracy: 0.8206 - val_loss: 0.4248 - val_accuracy: 0.8385

با توجه به نمودار بدست آمده از تابع هزینه، نوسان داده‌های تست کم‌تر شده ولی داده‌های آموزشی نوسان کمی پیدا کردند که این نشان از lr بالا است. در مرحله بعد از regularization به جای droupout استفاده می‌کنیم تا ببینیم چه تغییری می‌کند.



نمودار دقت برای داده‌های آموزشی و داده‌های تست:

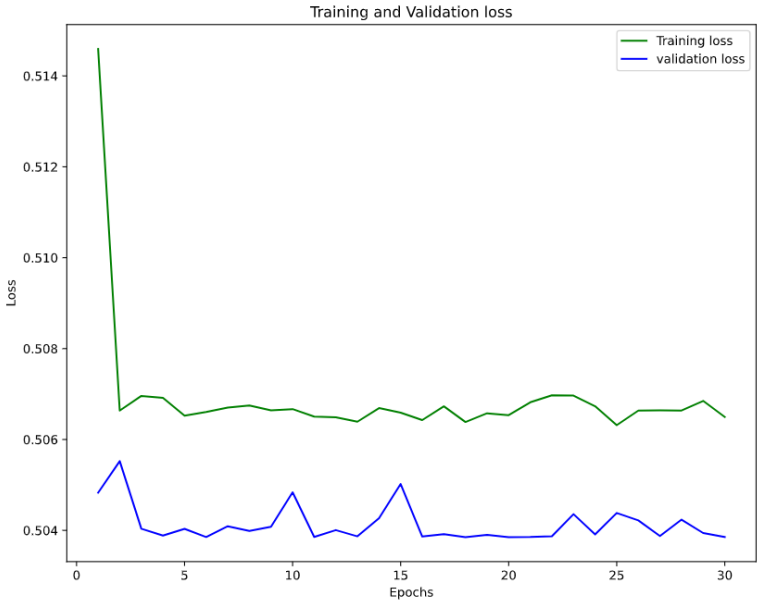


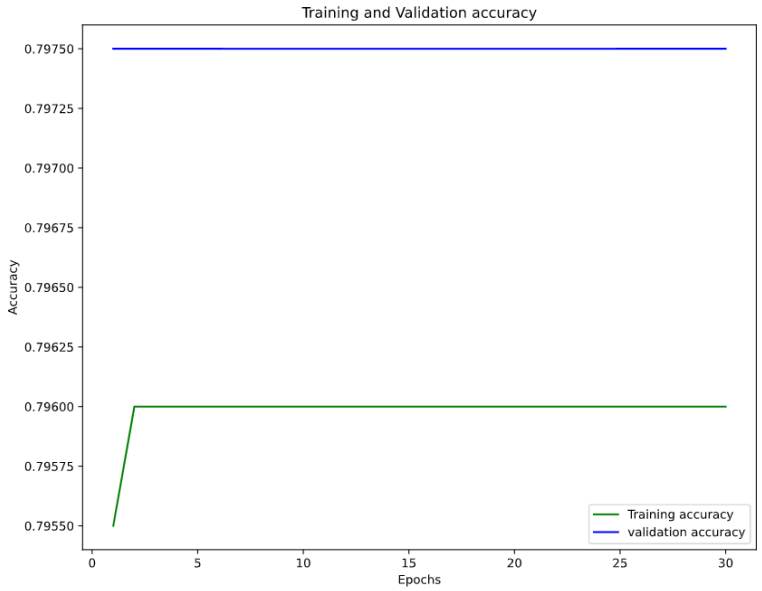
آزمایش ۱۱:

در این مرحله regularization را برای تمام لایه‌ها با نرخ ۰,۰۱ به کار میبریم و به نتیجه زیر می‌رسیم.

loss: 0.5065 - accuracy: 0.7960 - val_loss: 0.5039 - val_accuracy: 0.7975

با توجه به نمودار زیر نوسان تابع هزینه برای داده‌های تست کم ترشده است.



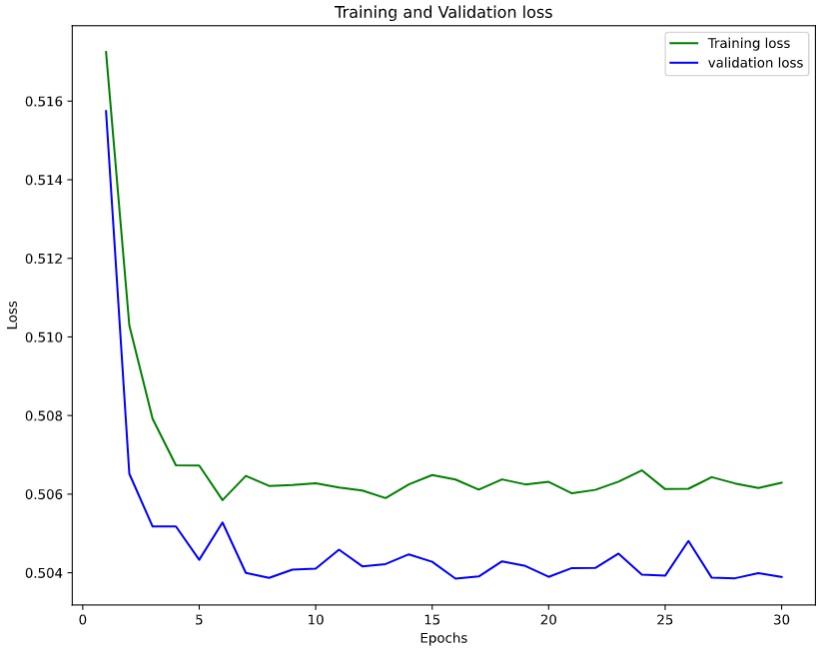


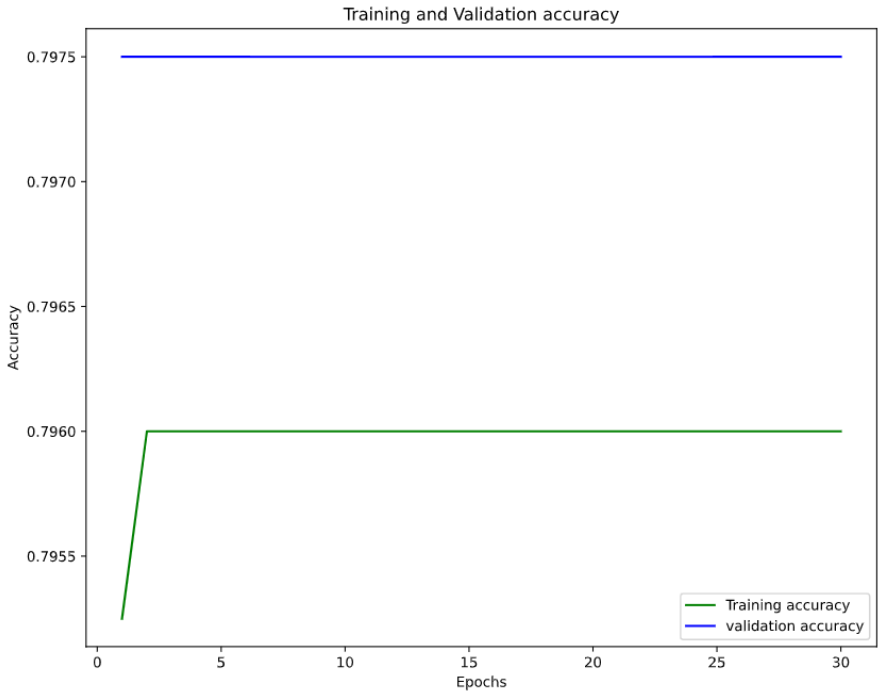
آزمایش ۱۲:

نرخ lr را به ۰,۰۱ کاهش می‌دهیم تا شاید نوسان تابع هزینه کم‌تر شود.

loss: 0.5063 - accuracy: 0.7960 - val_loss: 0.5039 - val_accuracy: 0.7975

طبق نمودار زیر، چیزی که به نظر می‌رسد این است که شاید مدل در یک مینیمم محلی گیر کرده است که تابع هزینه‌ی آن از یک مقداری پایین‌تر نمی‌آید و احتمالا مشکل از وزن وردی است که به یک مینیمم محلی نزدیک بوده؛ پس کاری که در آزمایش بعد می‌کنیم این است که نرخ یادگیری را به همان ۰,۰۲ برسانیم که توانایی اکتشاف آن بالاتر رود و وزن ورودی را نیر تغییر می‌دهیم. همچنین برای دقت بالاتر تعداد epoch ها را نیز بیشتر می‌کنیم.





آزمایش ۱۳:

در اینجا با توجه به نتیجه قبل، تعداد epoch ها و نرخ یادگیری افزایش پیدا کرده و همچنین وزن ورودی تغییر داده شده است.

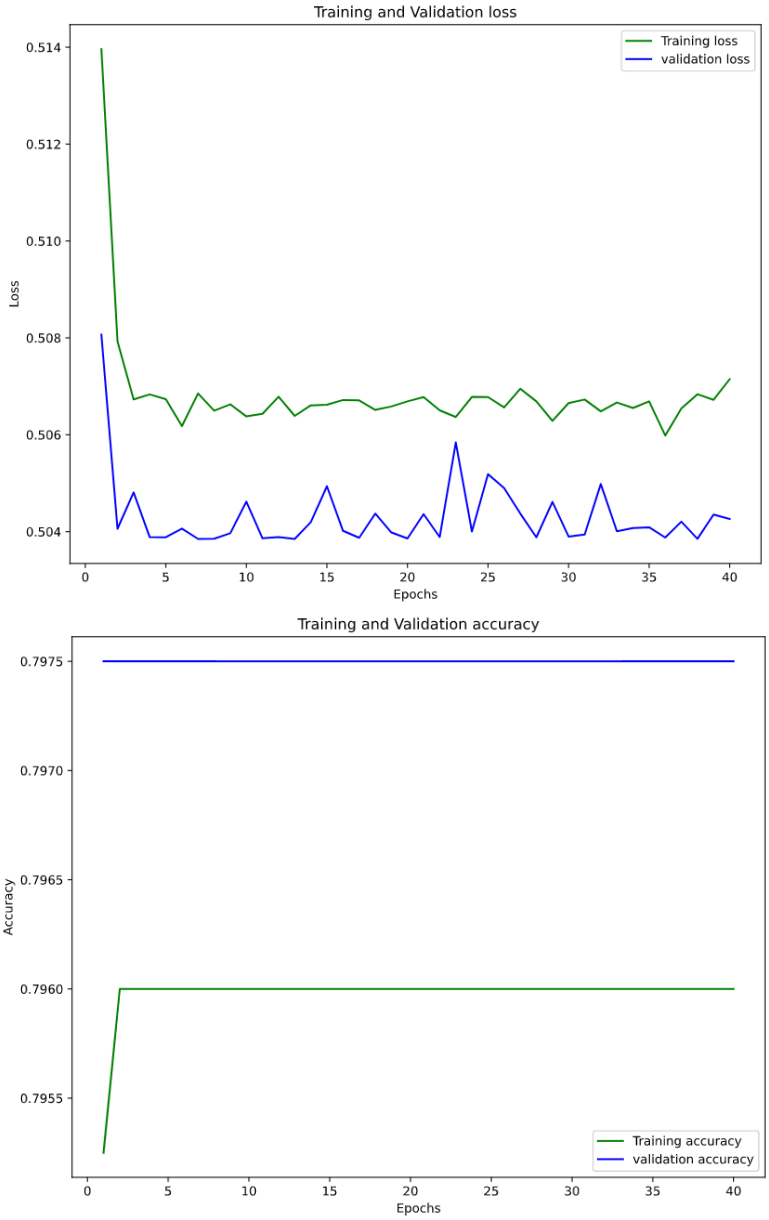
پارامترهای انتخاب شده:

epoch = 40, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomNormal ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم

loss: 0.5071 - accuracy: 0.7960 - val_loss: 0.5043 - val_accuracy: 0.7975

با توجه به شکل زیر نوسان تابع هزینه بیشتر می‌شود که این بخاطر بالا رفتن نرخ یادگیری است و دقت و تابع هزینه نسبت به حالت قبل تغییر چندانی نمی‌کند و وزن ورودی تاثیری نداشت. در مرحله بعد مدل را پیچیده‌تر می‌کنیم تا شاید دقت و تابع هزینه بهتر شوند.



آزمایش ۱۴:

در این مرحله یک لایه‌ی دیگر به مدل اضافه می‌شود تا مدل پیچیده‌تر شود و یک شبکه عصبی چند لایه با ۳ لایه‌ی مخفی می‌سازیم. در لایه سوم ۵ نورون قرار می دهیم.

پارامترهای انتخاب شده:

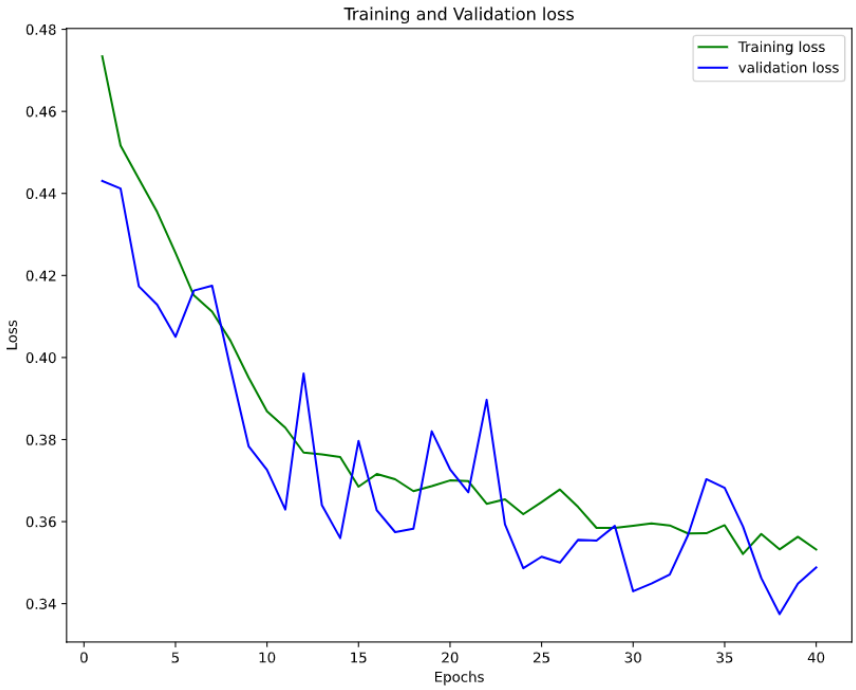
epoch = 40, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomNormal ,hidden_activation_function = Relu

با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم.

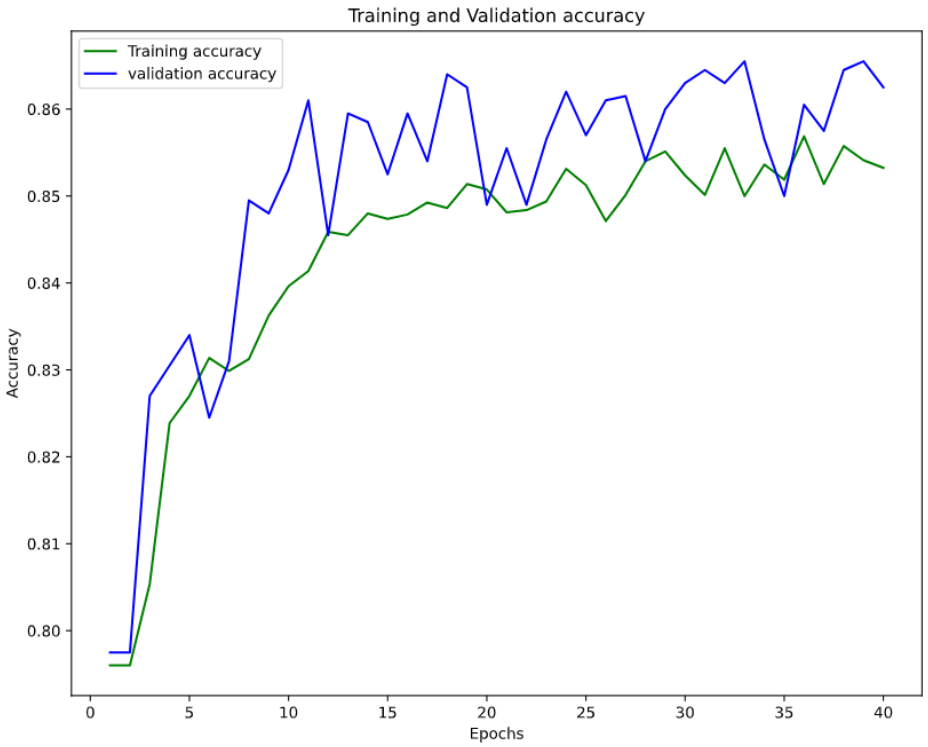
loss: 0.3532 - accuracy: 0.8533 - val_loss: 0.3489 - val_accuracy: 0.8625

در این آزمایش تابع هزینه نسبت به تمام مراحل دیگر بهتر شده و دقت آن نیز به نسبت مراحل دیگر بهتر است و بنظر می‌رسد تا اینجا مدل بهینه باشد.

نمودار تابع هزینه برای داده‌های آموزشی و دادهای تست به صورت زیر است:



نمودار دقت برای داده‌های آموزشی و داده‌ها تست:



آزمایش ۱۵:

در این مرحله میخواهیم بدانیم که آیا مدل بهتر از این می‌شود یا نه. برای افزایش دقت ابتدا تعداد epoch ها را بیشتر می‌کنیم تا تاثیر آن را ببینیم.

پارامترهای انتخاب شده:

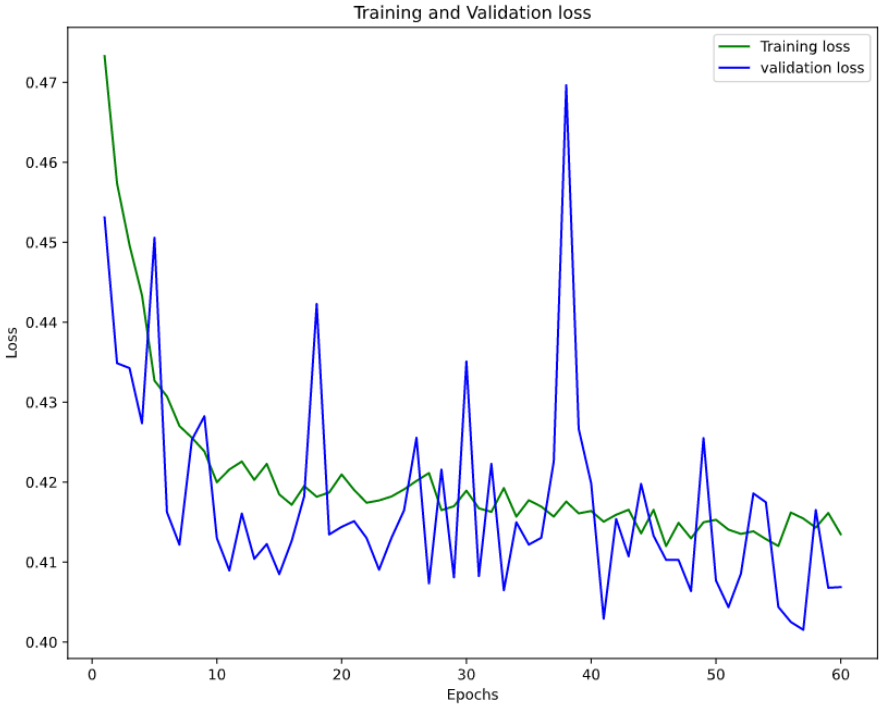
epoch = 60, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomNormal ,hidden_activation_function = Relu

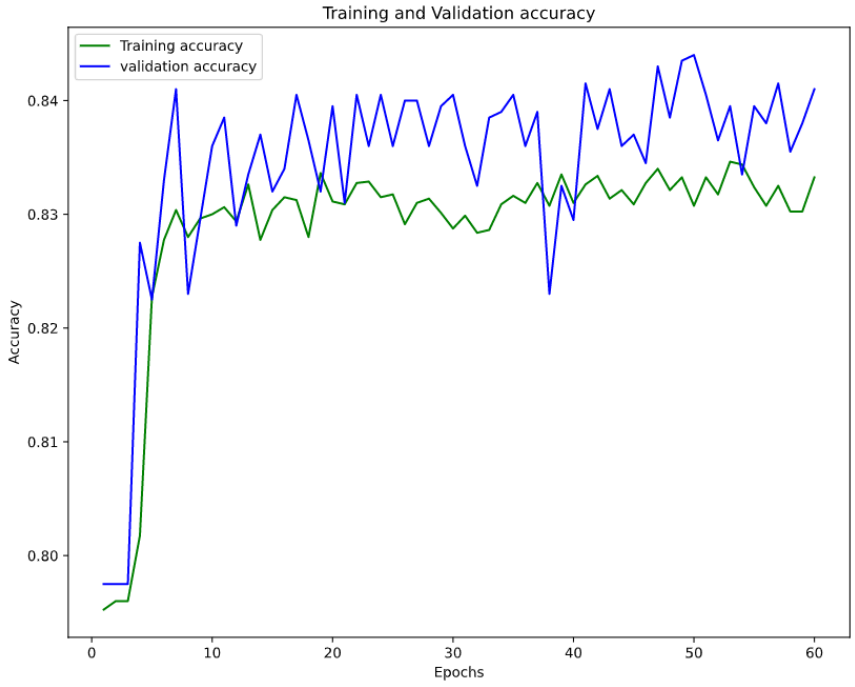
با پارامترهای انتخاب شده به نتیجه زیر می‌رسیم.

loss: 0.4135 - accuracy: 0.8332 - val_loss: 0.4069 - val_accuracy: 0.8410

نتیجه نسبت به دفعه‌ی پیش بدتر شده، هم دقت و هم تابع هزینه نسبت به آزمایش ۱٤ بدتر شده است.

نمودارهای مقایسه‌ی تابع هزینه و دقت برای داده‌های آموزشی و داده‌های تست:





آزمایش ۱۶:

در این مرحله روی آزمایش شماره ۱۴ کار می‌کنیم که تا حالا بهینه ترین مدل ما بوده، برای افزایش سرعت یادگیری سایز batch را افزایش می‌دهیم و احتمالا با این کار نوسان داده‌ها باید کمتر شود.

پارامترهای انتخاب شده:

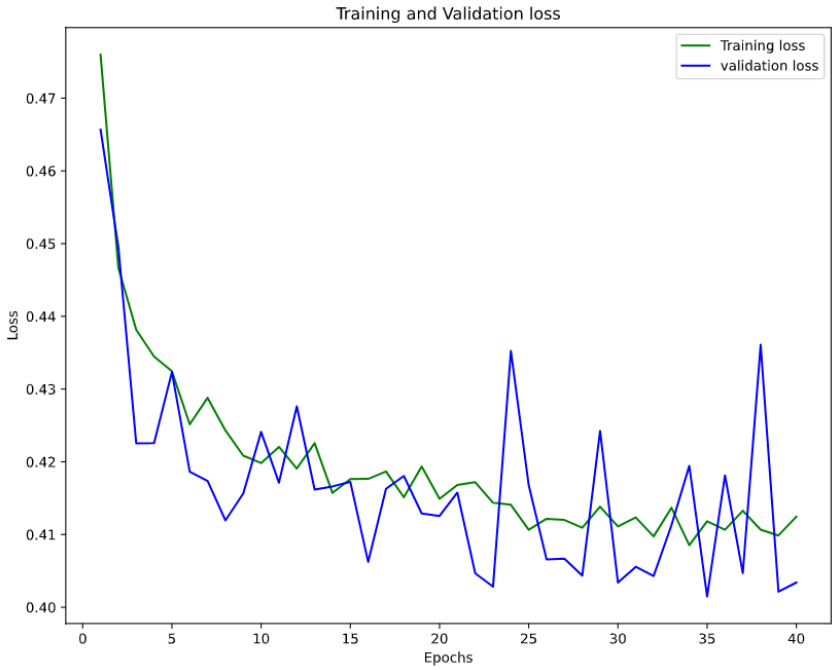
epoch = 40, lr (learning rate) = 0.02, batch = 20, weight initializer = RandomNormal ,hidden_activation_function = Relu

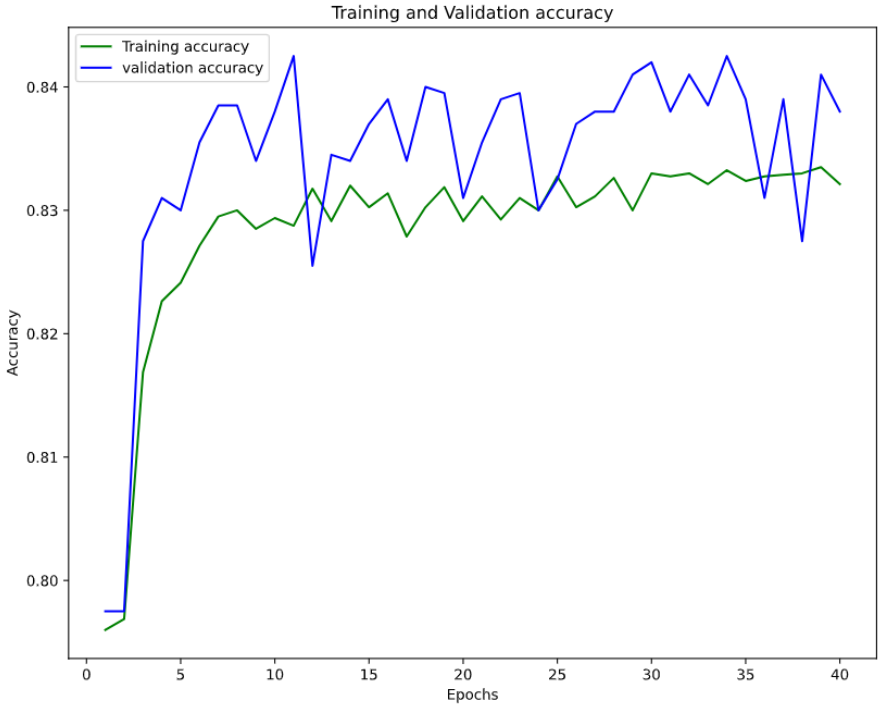
با پارامترهای انتخاب شده به نتیجه زیر می‌رسیم:

loss: 0.4125 - accuracy: 0.8321 - val_loss: 0.4034 - val_accuracy: 0.8380

مشاهده می‌شود نتیجه نسبت به آزمایش ۱۴ بدتر شده ولی سرعت یادگیری بیشتر شد.

نمودار های تابع هزینه و دقت برای داده‌های آموزشی و داده‌های تست به صورت زیر است:





آزمایش ۱۷:

حالا ترکیبی از آزمایش ۱۵ و ۱۶ را امتحان می‌کنیم که هم تعداد epoch ها بالاتر برود و هم سایز batch افزایش پیدا کند با اینکار شاید نوسان نمودار کمتر شود.

پارامترهای انتخاب شده:

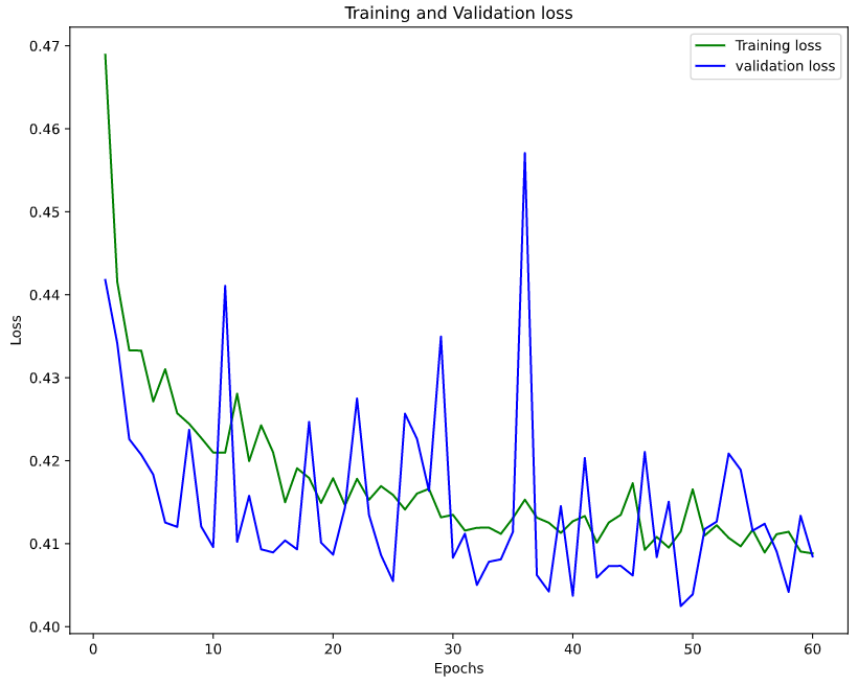
epoch = 60, lr (learning rate) = 0.02, batch = 20, weight initializer = RandomNormal ,hidden_activation_function = Relu

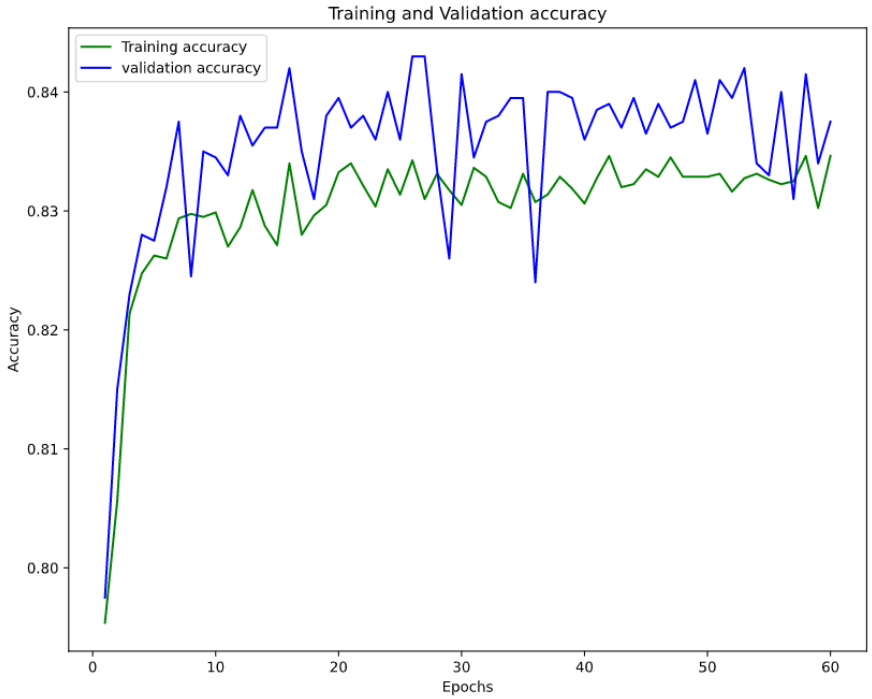
با پارامترهای انتخاب شده به نتیجه‌ی زیر می‌رسیم:

loss: 0.4088 - accuracy: 0.8346 - val_loss: 0.4085 - val_accuracy: 0.8375

در این روش نیز نتیجه ی بهتری نسبت به آزمایش ۱۴ بدست نمی‌آید.

شکل های صفحه بعد به ترتیب نمودارهای تابع هزینه و دقت برای داده‌های آموزشی و داده‌های تست می‌باشند:





آزمایش ۱۸:

در اینجا روی آزمایش ۱٤ امتحان می‌کنیم و برای اینکه نوسان داده‌های آموزشی و تست کم شود یه راه حل این است که نرخ یادگیری را کمتر کنیم.

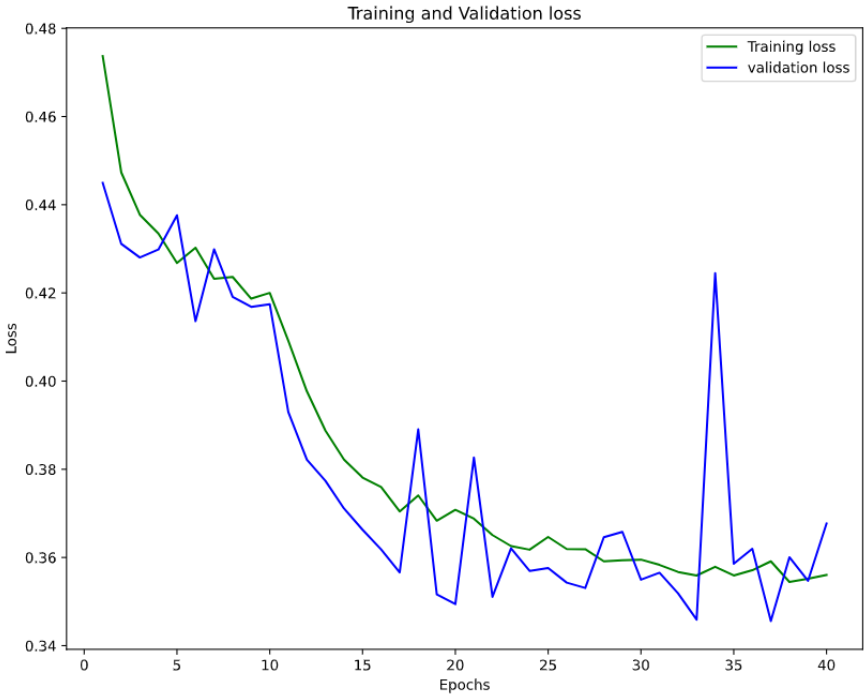
پارامترهای انتخاب شده:

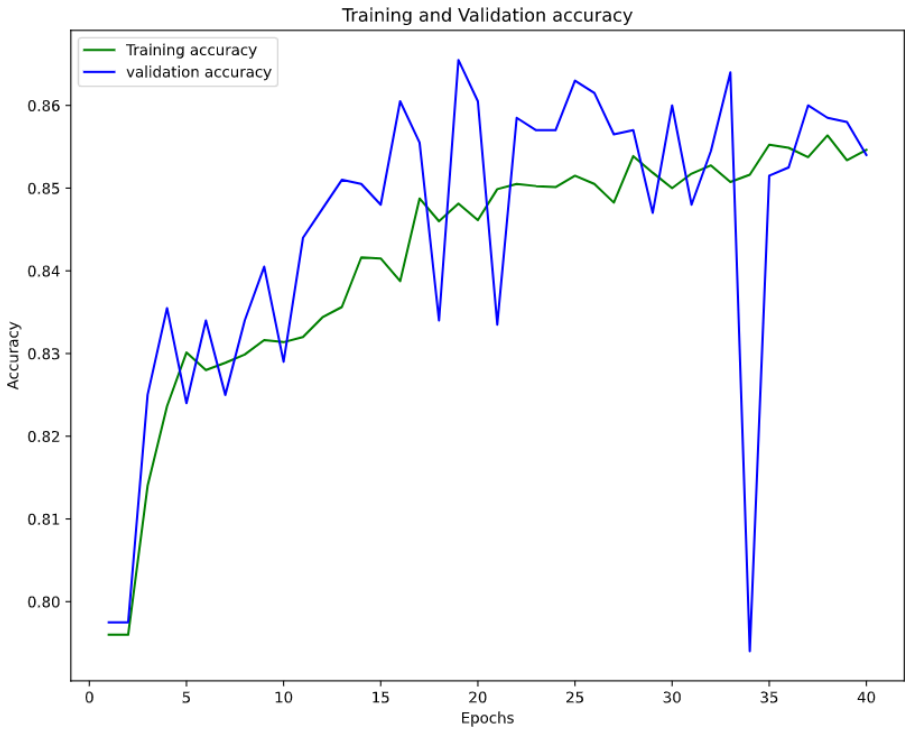
epoch = 40, lr (learning rate) = 0.01, batch = 10, weight initializer = RandomNormal ,hidden_activation_function = Relu

با پارامترای انتخاب شده به نتیجه زیر می‌رسیم:

loss: 0.3560 - accuracy: 0.8546 - val_loss: 0.3677 - val_accuracy: 0.8540

به نتیجه بهتری نرسیدیم و ولی فقط دقت روی داده‌های آموزشی افزایش یافت. و تابع هزینه با سرعت کمتری کاهش می‌یابد پس این مقدار lr کم می‌باشد.نمودارهای تابع هزینه و دقت برای داده‌های آموزشی به صورت زیر است:





آزمایش ۱۹:

در این آزمایش نیز روی آزمایش ۱٤ تغییراتی انجام می‌دهیم و می‌خواهیم تاثیر توابع فعالیت در لایه‌های مخفی را بینیم. در این آزمایش از تابع tanh برای لایه‌های مخفی استفاده می‌کنیم.

پارامترهای انتخاب شده:

epoch = 40, lr (learning rate) = 0.02, batch = 10, weight initializer = RandomNormal ,hidden_activation_function = tanh

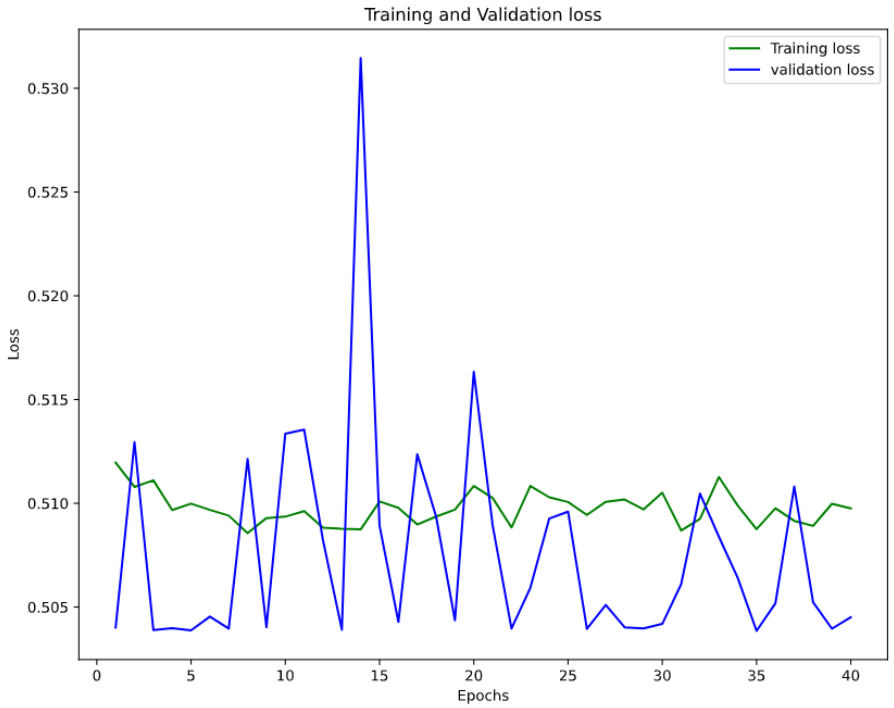
با پارامترهای اننتخاب شده به نتیجه زیر می‌رسیم:

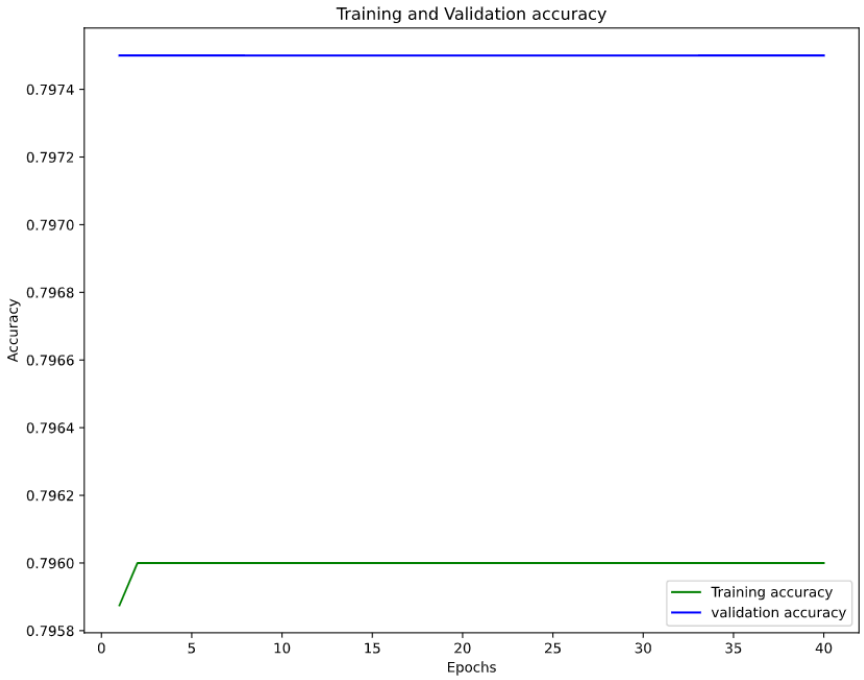
loss: 0.5098 - accuracy: 0.7960 - val_loss: 0.5045 - val_accuracy: 0.7975

در این حالت انتخاب تابع tanh باعث شده که تابع هزینه بدتر شود و کاهش پیدا نکند؛ زیرا مشتق این تابع بسیار کم است و در نقاط دورتر از صفر، مشتق به صفر می‌رسد و همین باعث می‌شود تابع هزینه کاهش پیدا نکند.

همانطور که در نمودار تابع هزینه دیده می‌شود تابع هزینه برای داده‌های آموزشی تغییری نمی‌کند و تقریبا ثبات است.

نمودارهای تابع هزینه و قیمت برای داده‌های آموزشی و داده‌های تست:





آزمایش ۲۰:

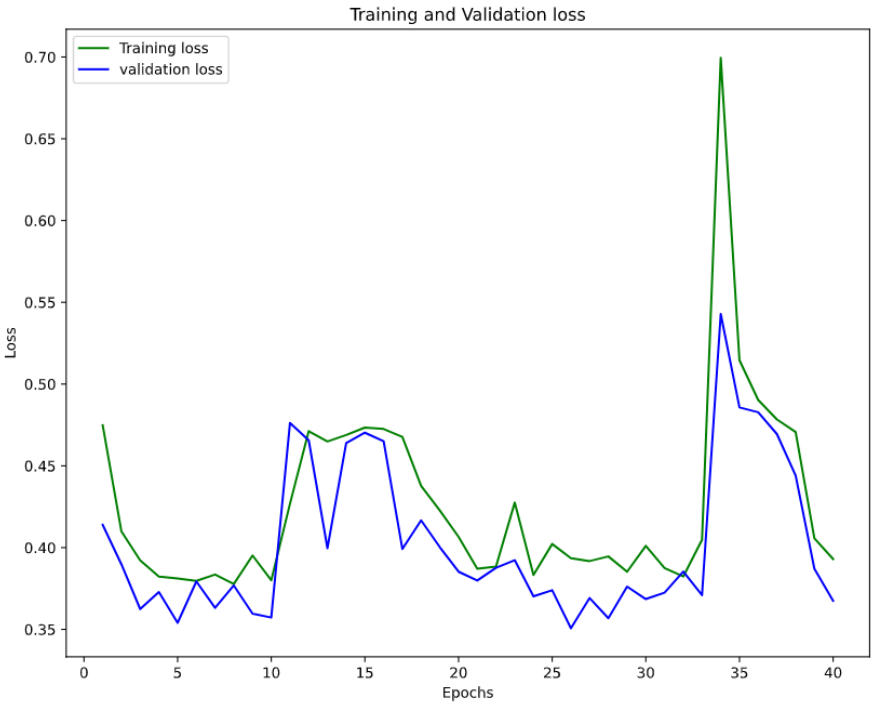
در این آزمایش روی آزمایش ۱٤ تغییر انجام می‌دهیم و تابع فعالسازی لایه خروجی را tanh می‌گذاریم ولی بقیه پارامترها را تغییر نمی‌دهیم.

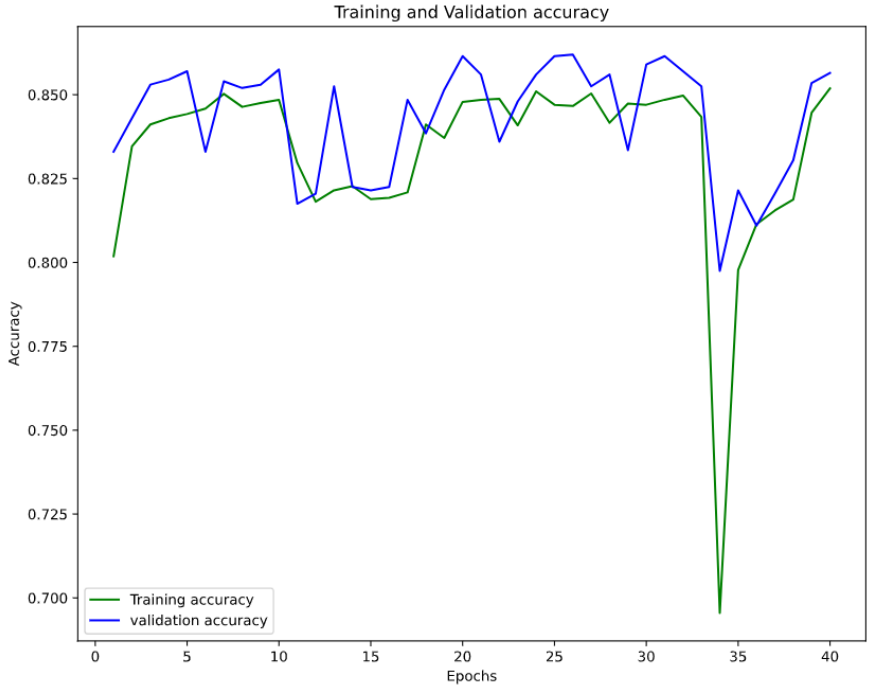
به نتیجه زیر می‌رسیم:

loss: 0.3930 - accuracy: 0.8519 - val_loss: 0.3674 - val_accuracy: 0.8565

که نسبت به مدل ۱٤ نتیجه بدتری دارد و تابع هزینه برای آن بیشتر می‌شود به نظر می‌رسد دلیل آن این است که چون خروجی tanh بازه‌ی بزرگ تری را درنظر می‌گیرد و بازه (۱و۰) را یک و بازه (۰و۱–) را صفر در نظر می‌گیرد تعداد زیادی از جواب های پیش بینی را یک درنظر می‌گیرد در صورتی که باید صفر باشند و با بالعکس. در صورتی که در sigmoid این بازه کوچکتر است و (۱و۰,۵) را یک و بازه (۰,۵و۰) را صفر در نظر می‌گیرد.

از این آزمایش نمودارهای زیر را داریم:


Training and Validation loss
Training loss
validation loss
Loss
Epochs
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
5
10
15
20
25
30
35
40


Training and Validation accuracy
Training accuracy
validation accuracy
Accuracy
Epochs
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0
5
10
15
20
25
30
35
40

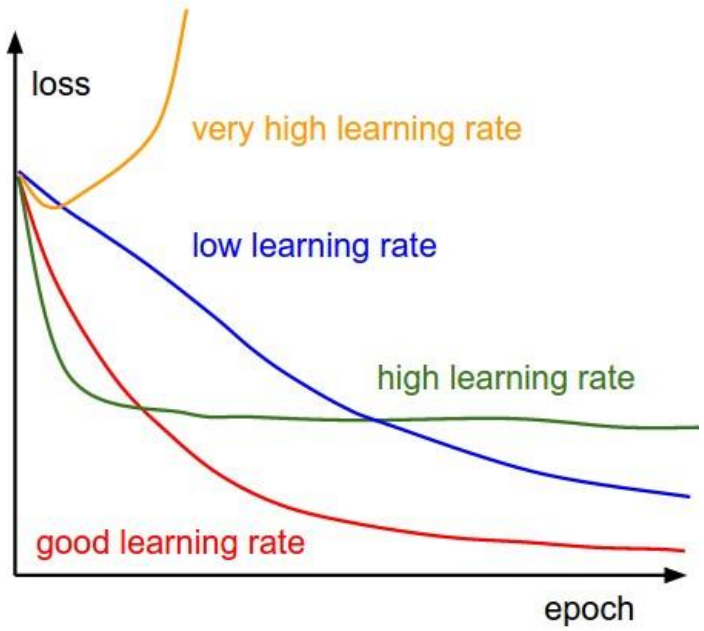
جمع بندی و نتیجه‌گیری

طبق آزمایش‌های آنجام شده آزمایش ۱۴ بهترین آزمایش است که شبکه‌ی عصبی آن، ۳ لایه مخفی دارد و تعداد نورن های لایه مخفی آن به ترتیب ۲۰،۲۰ و ۵ است؛ زیرا هم دقت خوبی روی داده‌های آموزشی و تست دارد و هم تابع هزینه برای داده‌های آموزشی به نسبت بقیه‌ی آزمایش‌ها تا حد خوبی کاهش یافته است.

طبق آزمایش‌های انجام شده متوجه می‌شویم که هر چقدر سایز batch افزایش پیدا کند سرعت learn کردن مدل افزایش پیدا می‌کند و نویز کم‌تر می‌شود زیرا در اینصورت برآیند داده‌ها را در نظر می‌گیریم و با اینکار اثر نویز داده ها کم‌تر می‌شود و نوسان کم‌تری در نمودارها می‌بینیم. ولی هر چقدر سایز batch کم‌تر باشد مانند این است که داده‌ها را تک تک به مدل بدهیم و خب در این حالت نویز خیلی زیادی خواهیم داشت. هر چقدر سایز batch افزایش پیدا کند learning rate بالاتری می‌خواهد برای اینکه خوب عمل کند.

با افزایش تعداد epoch یعنی با افزایش تعداد بارهایی که کل داده‌ها را به مدل می‌دهیم، دقت مدل افزایش پیدا می‌کند. البته این افزایش تعداد epoch تا زمانی خوب است که دقت برای داده‌های validation کم نشود اگر با افزایش تعداد epoch دقت برای داده‌های آموزشی زیاد شود ولی برای داده‌های validation کم شود (و یا اینکه تابع هزینه برای داده‌های آموزشی کم شود ولی از یک جایی تابع هزینه برای داده‌های validation زیاد شود) به این معنی است که مدل ما overfit شده است و باید با استفاده از تکنیک droupout تعدادی از نورون‌ها یا تعدادی از کانکشن‌ها را off کنیم و یا با استفاده از regularization تعداد وزن‌های غیرصفر آن را کاهش دهیم.

هر چقدر مدل پیچیده‌تر شود (افزایش تعداد نورون ها و تعداد لایه‌ها)، برای اینکه مدل خوب عمل کند learning rate هم باید بالاتر رود که تابع هزینه با شیب مناسب و خوبی مانند شکل زیر کاهش پیدا کند.



در مورد activation function ها بهترین تابع برای لایه‌های میانی Relu هستند زیرا مشتق آن‌ها در نقاط غیر صفر یک است و مشتق آن نسبت به تابع هایی مثل sigmoid و tanh بیشتر است؛ زیرا این دو تابع در اکثر نقاط مشتق صفر دارند و در نقاط نزدیک صفر بیشترین مشتق را دارند که برای Sigmoid حداکثر ۰,۲ و برای tanh حداکثر ۱ است.

تابع هایی مانند tanh و sigmoid به دلیل مشتق کم،در عمق باعث vanishingو یا exploding gradient می‌شود به همین دلیل از یکجایی به بعد باعث بهبود وزن نمی‌شوند زیرا مشتقشان صفر می‌شود. ولی در relu چنین مشکلی نداریم.

از این دو تابع بیشتر برای لایه‌های خروجی استفاده می‌شود.

در مورد تابع فعالسازی برای لایه‌ی خروجی طبق آزمایشی که من انجام دادم تابع sigmoid بهتر عمل می‌کند و به نظر می‌رسد دلیل آن این است که چون خروجی tanh بازه‌ی بزرگ تری را درنظر می‌گیرد و بازه (۱و۰) را یک و بازه (۰و۱-) را صفر در نظر می‌گیرد تعداد زیادی از جواب های پیش بینی را یک درنظر می‌گیرد در صورتی که باید صفر باشند و با بالعکس. در صورتی که در sigmoid این بازه کوچکتر است و (۱و۰,۵) را یک و بازه (۰,۵و۰) را صفر در نظر می‌گیرد.

برای مقادیر اولیه‌ی وزن‌ها توزیع ها و کلاس‌های مختلفی در کراس وجود دارد که ساخته می‌شود و برای هر لابه یک weight initializer وجود دارد و باید امتحان شوند زیرا یکسری از وزن‌های اولیه به دلیل نزدیک بودن به یک مینیم محلی باعث می‌شوند ما در مینیم گیر کنیم و تابع هزینه کاهش پیدا نکند.