امیر سالاری تلمادره
۶۱۰۳۹۷۱۱۹

تمرین سری هشتم حسابگری زیستی

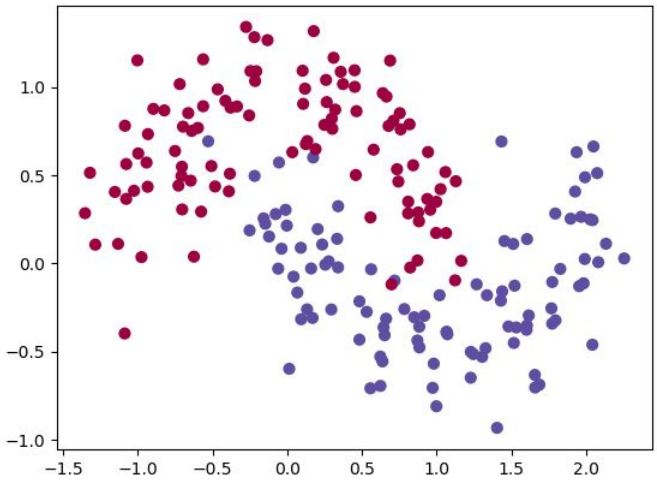
Final Project

فایل بخش های مختلف پروژه با اسم آن به صورت py. ۱ در فایل قرار دارد.

من برای اکثر موارد از کد اصلی که در فایل فرستاده شده در hw8 وجود داشت استفاده کردم.

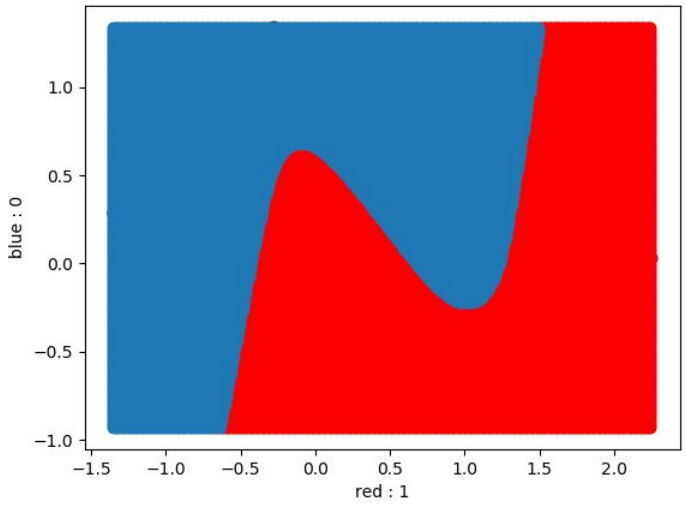
برای کشیدن decision boundary با فاصله ی هر ۰.۰۱ تمام نقاط صفحه را به مدل ساخته شده دادم و شکل نهایی را کشیدم.

طبق دیتاست موجود در فایل نقاط به صورت زیر هستند.



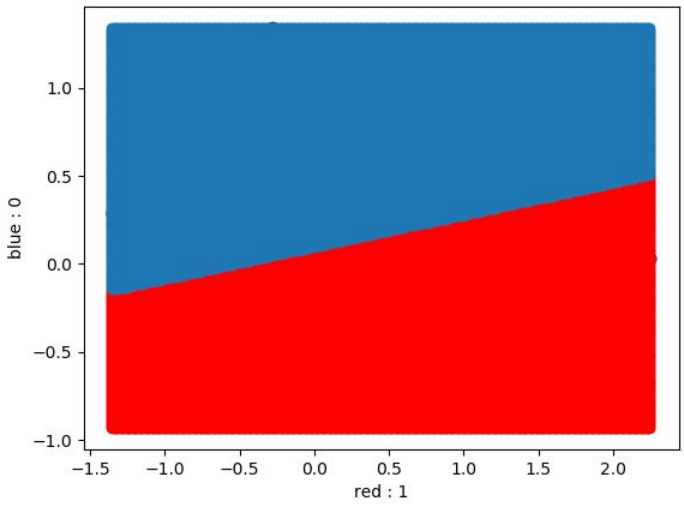
سوال ۱

طبق کد موجود در فایل و همینطور دیتاست موجود در فایل به همراه یک لایه ی مخفی با ۳ نرون برنامه را برای ۲۰,۰۰۰ بار اجرا کردم و decision boundary بدست آمده به صورت زیر است.

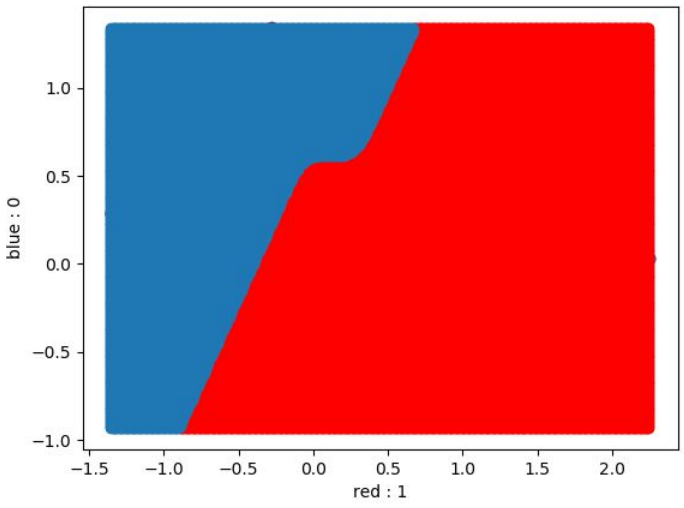


سوال ۲

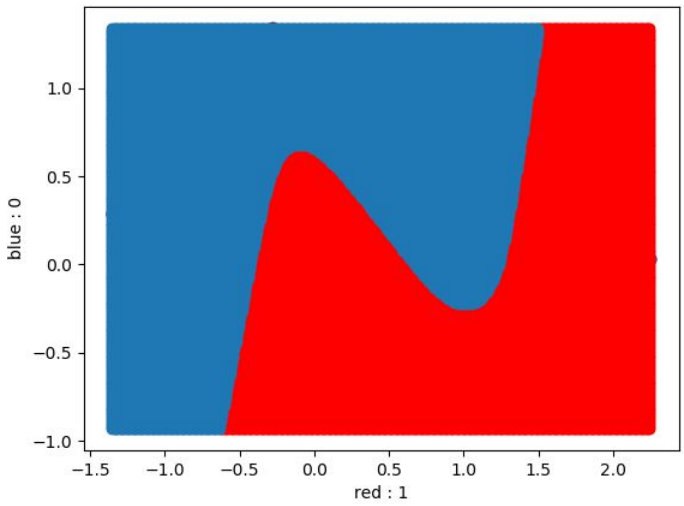
مانند بخش قبل با تکرار ۲۰,۰۰۰ بار و همینطور با تعداد نرون های مختلف در لایه مخفی decision boundary ها به صورت های زیر شدند.



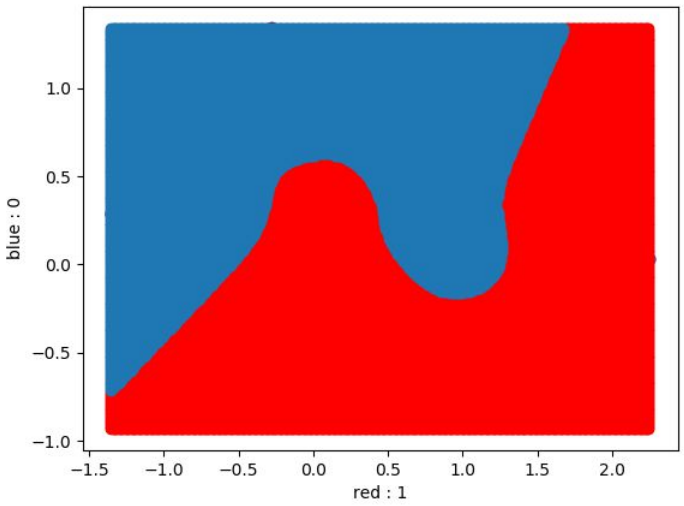
nn_hdim = 1 , loss = 0.33



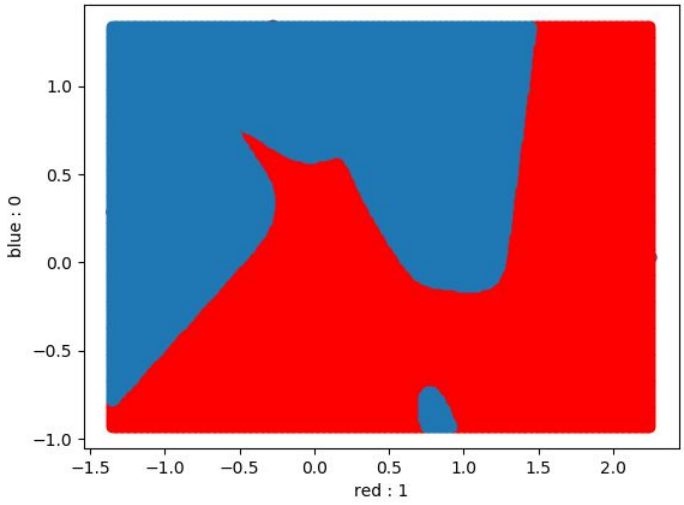
nn_hdim = 2 , loss = 0.32



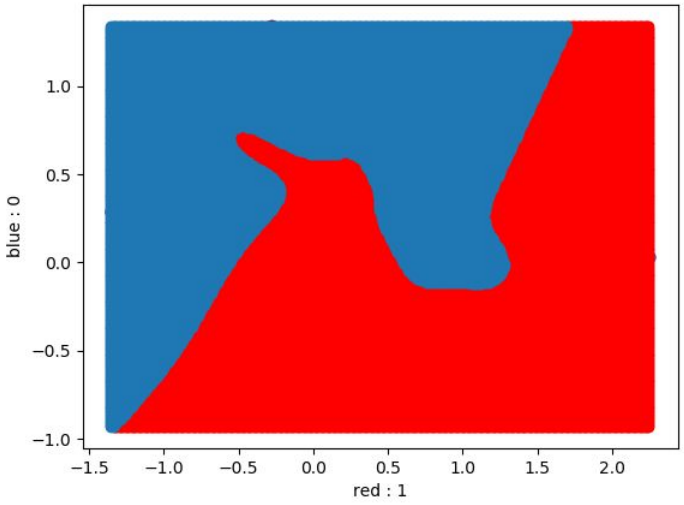
nn_hdim = 3, loss = 0.070



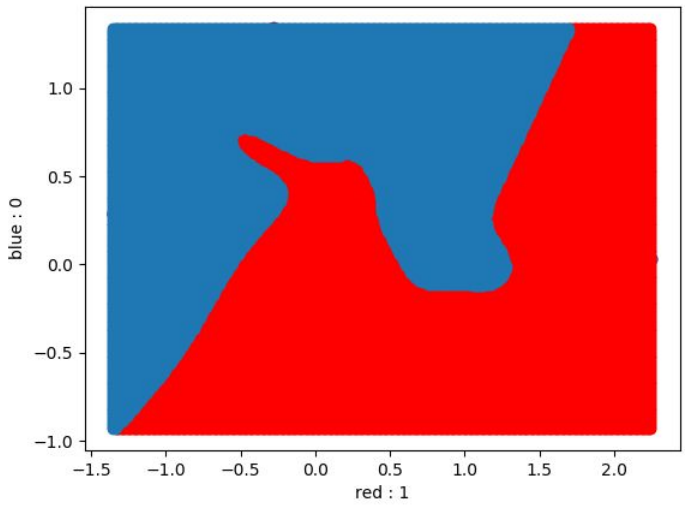
nn_hdim = 4 , loss = 0.052



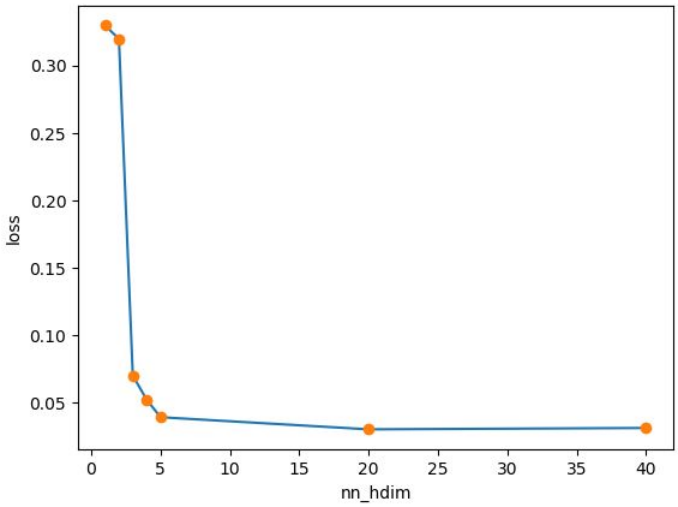
nn_hdim = 5 , loss = 0.039



nn_hdim = 20 , loss = 0.030



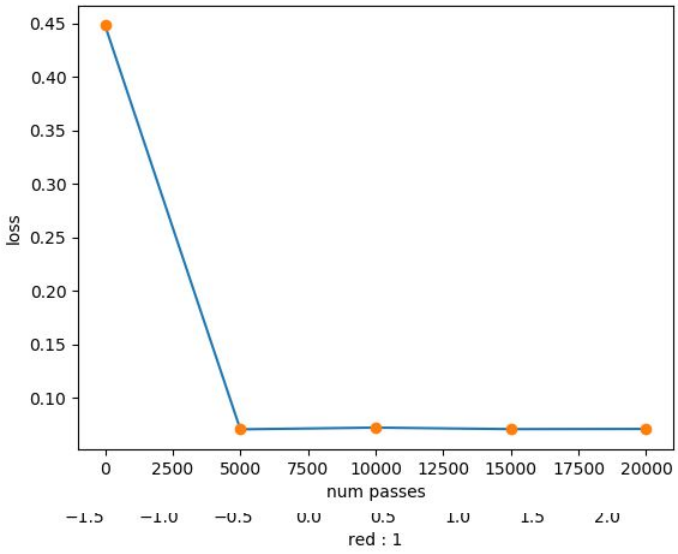
Nn_hdim = 40, loss = 0.031



همانطور که در نمودار ها مشخص است از ۵ به بعد تفاوت چندانی ندارند و بنظر من تعداد ۳ نرون بهترین است چون هم خطای نسبتا کمی دارد و هم زمان کمی صرف ساختنش میشود. ولی خب طبق درصد خطا بهترین تعداد نرون ۲۰ است.

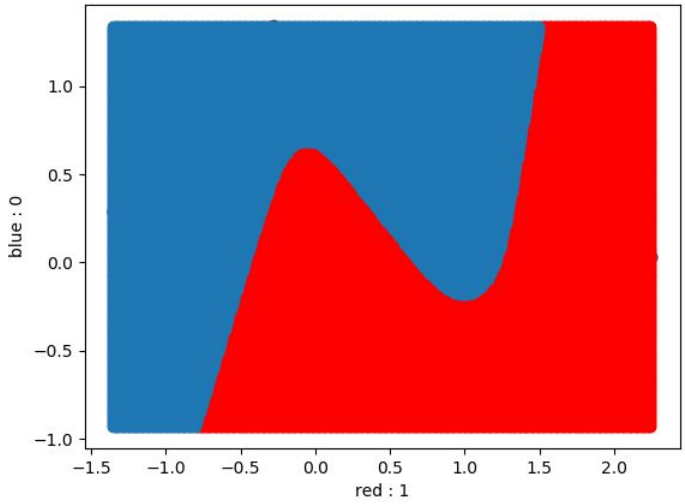
سوال ۳

در این بخش برای اینکه از دیتاست اولیه به صورت قسمت شده و همینطور تصادفی در هر مرحله استفاده شود در ابتدا یک متغیر جهانی با اسم num_batch اضافه کردم کهمشخص میکند در هر مرحله چند داده باید آموزش یابند. بعد از آموزش کامل کل داده برای یک بار دوباره دیتاست های اولیه به صورت تصادفی تغییر مکان میدهند تا این بار به صورت دیگری به نورال نتورک ما داده شود. این تغییر مکان تصادفی اجباری است چون در غیر این صورت داده ها مانند حالت batch gradient descent عمل میکند. شکل زیر decision boundary برای num_batch = 50 است. بعد از اولین آموزش خطا بسیار پایین میاید و مقدار خطای آن از حال batch gradient descent نیز کمتر است.



سوال ۴

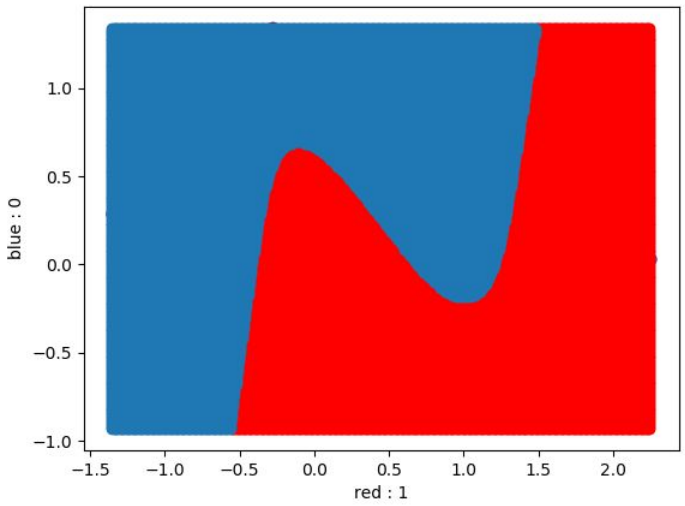
برای اینکه یک کاهش خوب برای learning rate داشته باشیم بهترین کار این است هر ۱۰۰۰ بار یا هر چند باری که دلمان میخواهد آن را کوچک کنیم. برای مثال من با epsilon = 0.05 شروع کردم و هر ۱۰۰۰ بار آن را نصف میکنم. درصد خطا از ۰.۷ به ۰.۰۶۹ کاهش یافت. شکل زیر decision boundary این روش است.



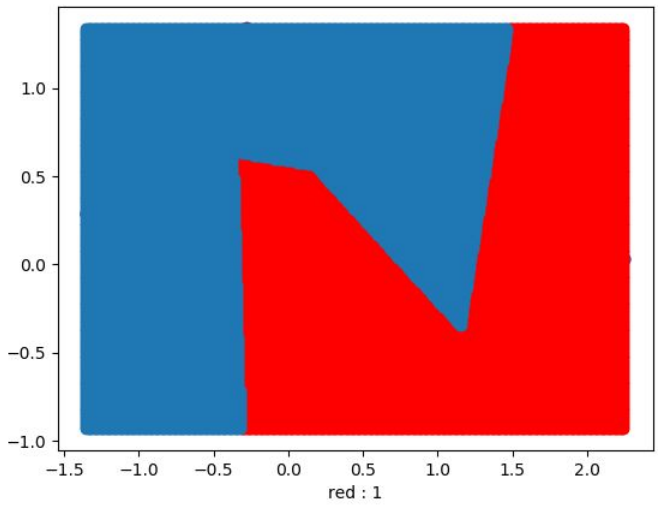
سوال ۵

من با تابع های sigmoid و ReLU اکتیویشن فانکشن ها را تغییر دادم و آن ها را برای ۲۰,۰۰۰ تکرار و تعداد ۳ نورون در لایه مخفی در شکل زیر رسم کردم. در اینجا ReLU مقدار خطای کمتری داشت ولی شکل decision boundary در ReLU بسیار عجیب و متفاوت از بقیه است. بهرحال هر دوی این توابع جواب های خوبی دارند و اگر بخواهیم خیلی دقیق باشیم ترتیب از بهترین این گونه است :

tanh > ReLU > sigmoid (دلیل اینکه ۳ نورون در لایه مخفی قرار دادم این بود که بهترین هرکدام همین ۳ نورون بود وگرنه بسیار بد عمل میکردند.)



sigmoid, loss = 0.78



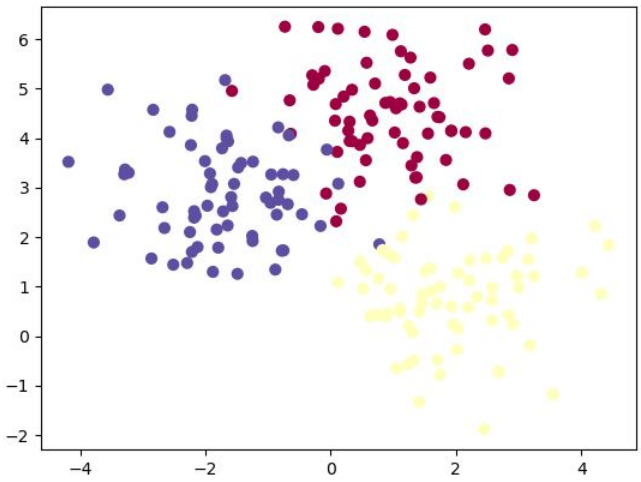
ReLU, loss = 0.071

سوال ۶

برای ساختن دیتا ست از تابع make_blobs که از کتابخانه sklearn گرفته میشود استفاده کردم.

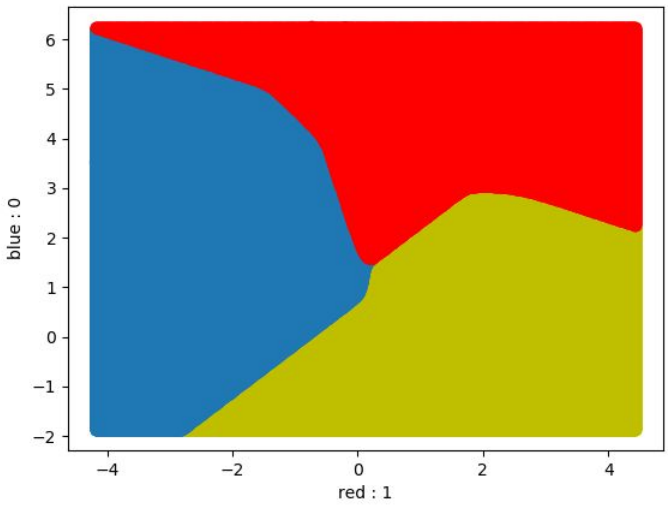
```
X, y = datasets.make_blobs(n_samples=200)
```

تعدادی نقطه با سه رنگ مختلف که شکل آن به صورت زیر است.

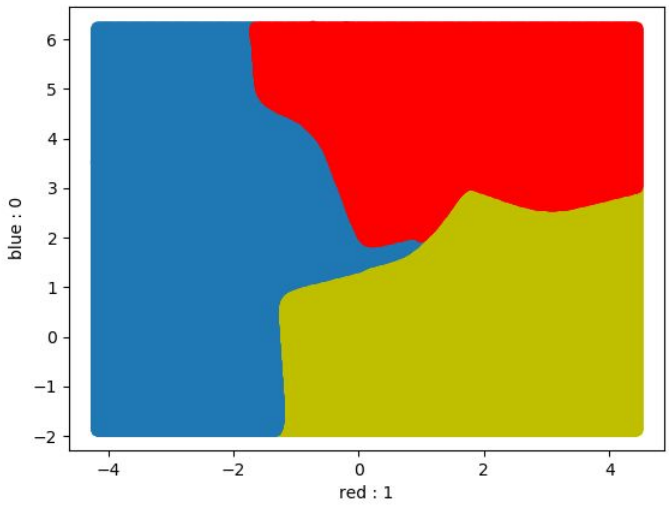


حال به کد قبلی فقط تعداد نرونهای لایه خروجی را ۳ قرار میدهیم و کار تمام است.

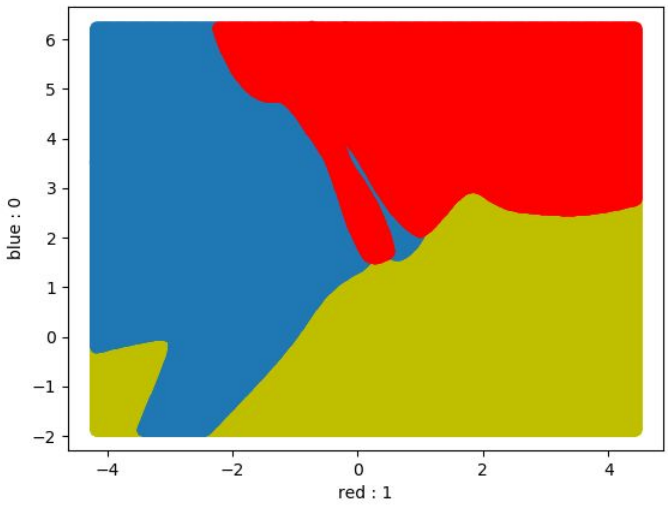
شکل های زیر decision boundary و خطای آن با تعداد نرون های لایه مخفی مختلف است.



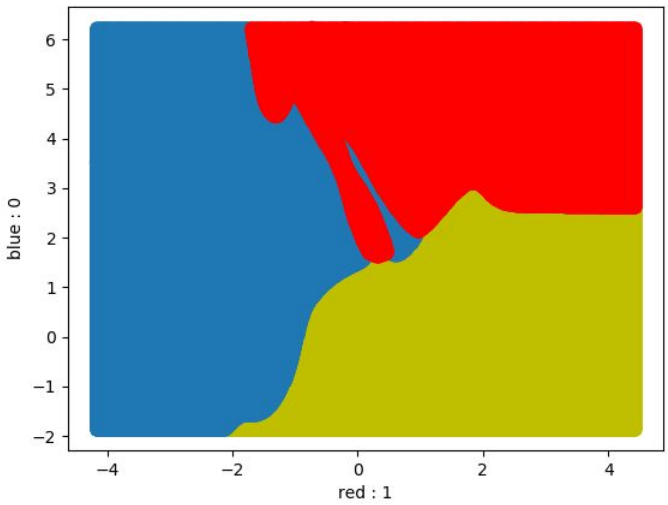
nn_hdim = 3 , loss = 0.074



nn_hdim = 5 , loss = 0.047



nn_hdim = 20 , loss = 0.028



nn_hdim = 40 , loss = 0.034

سوال ۷

فایل سوال ۷ در زیپ قرار دارد. به ۴ لایه با تابع فعال سازی tanh در لایه های وسط و softmax در لایه آخر.