

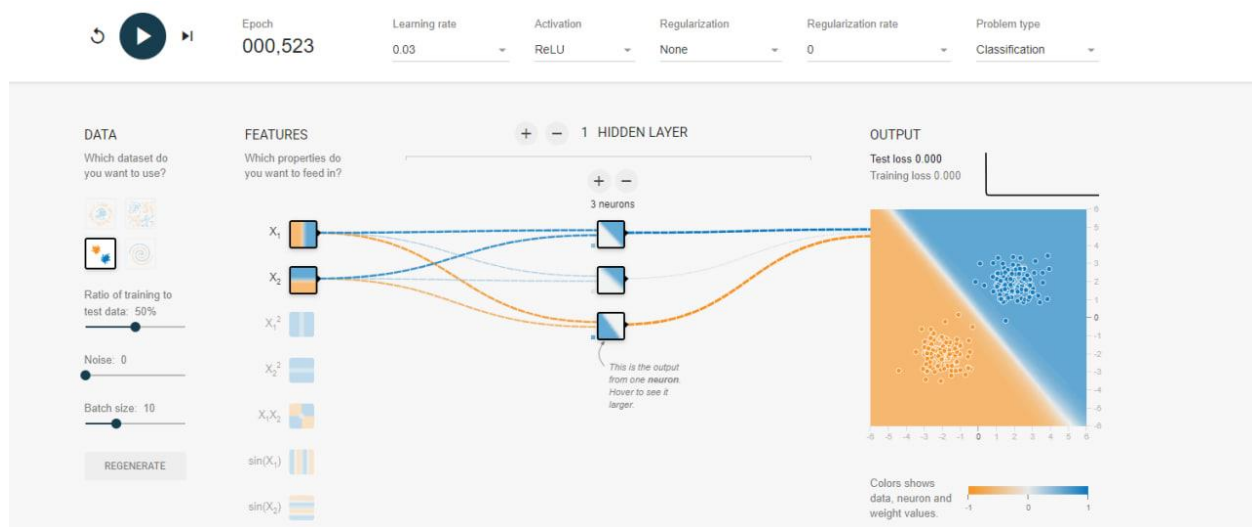
پاسخ سوالات 3 تا 6 سری تمرین 2

امیرمحمد کمیجانی 99522032

(3) در این سوال انواع دیتا با انواع توابع فعالسازی مختلف را بررسی میکنیم:

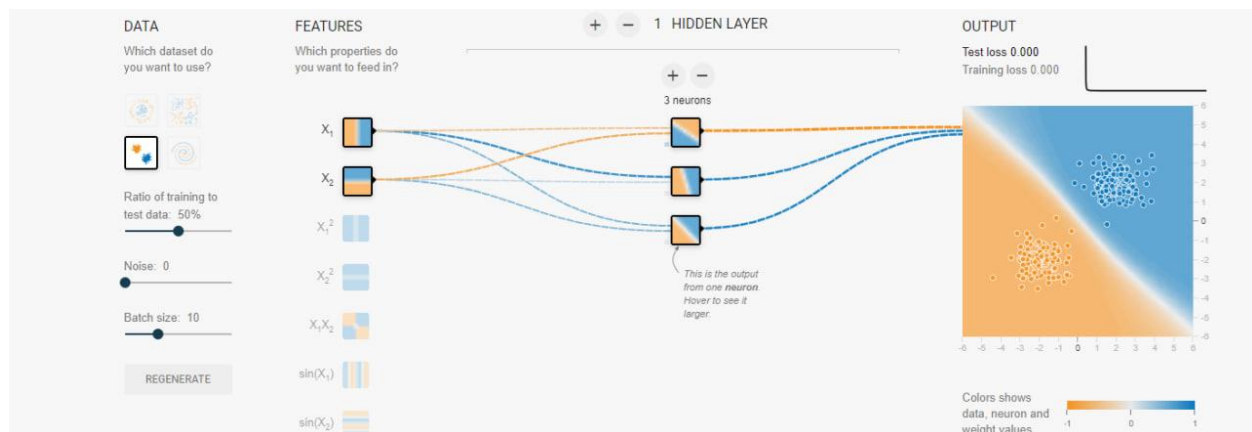
Data = Gaussian

(a) تابع فعالسازی ReLU



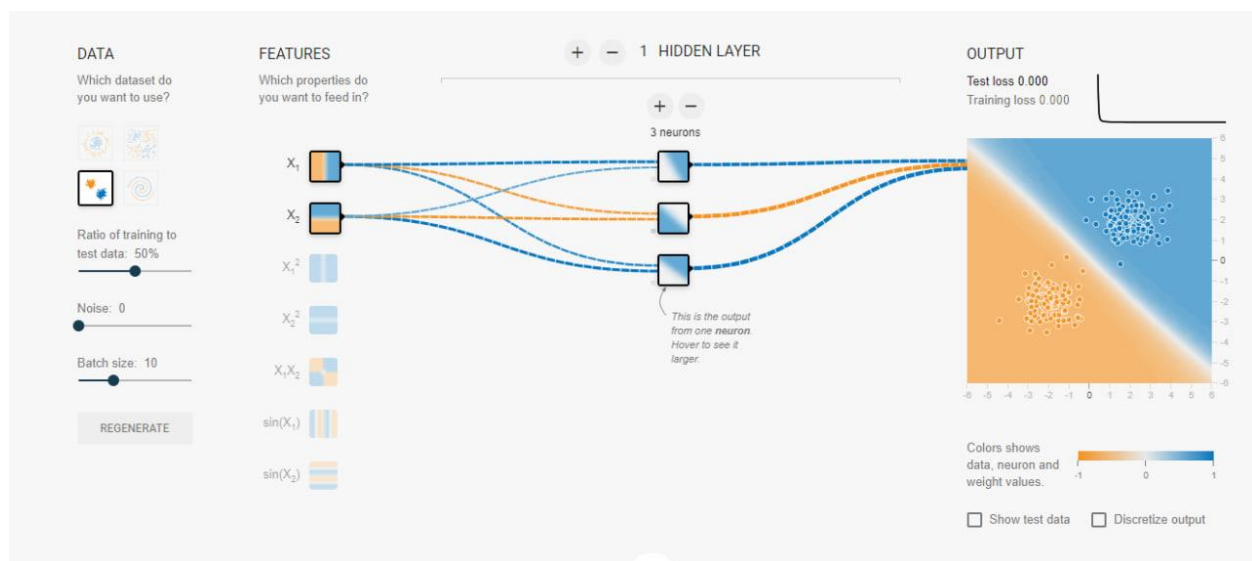
که مشخص است به خوبی مسئله classification را توانستیم با این تابع فعالسازی حل کنیم.

(b) تابع فعالسازی Tanh



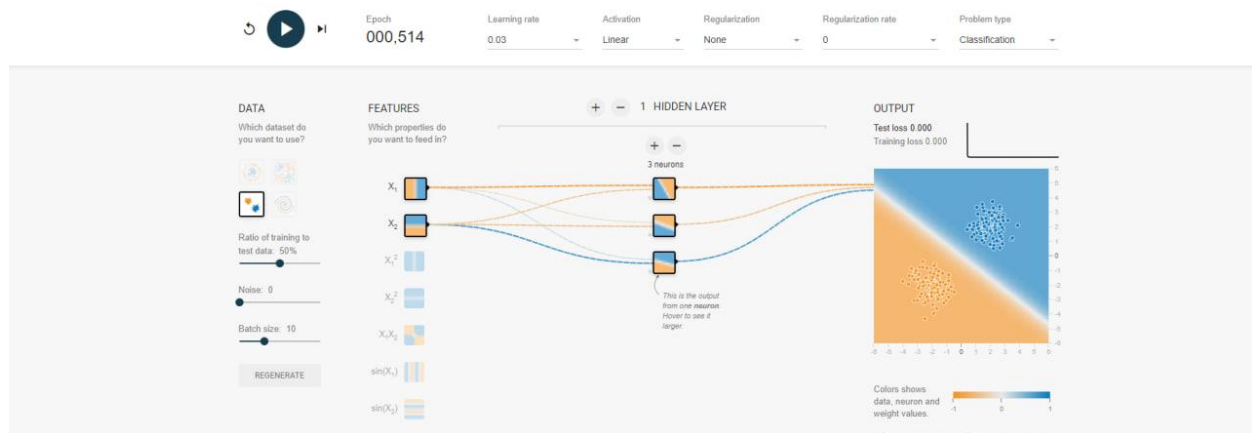
این تابع فعالسازی هم از پس این نوع دیتا بر آمد.

sigmoid activation function (c)



باز هم به خوبی عمل میکند

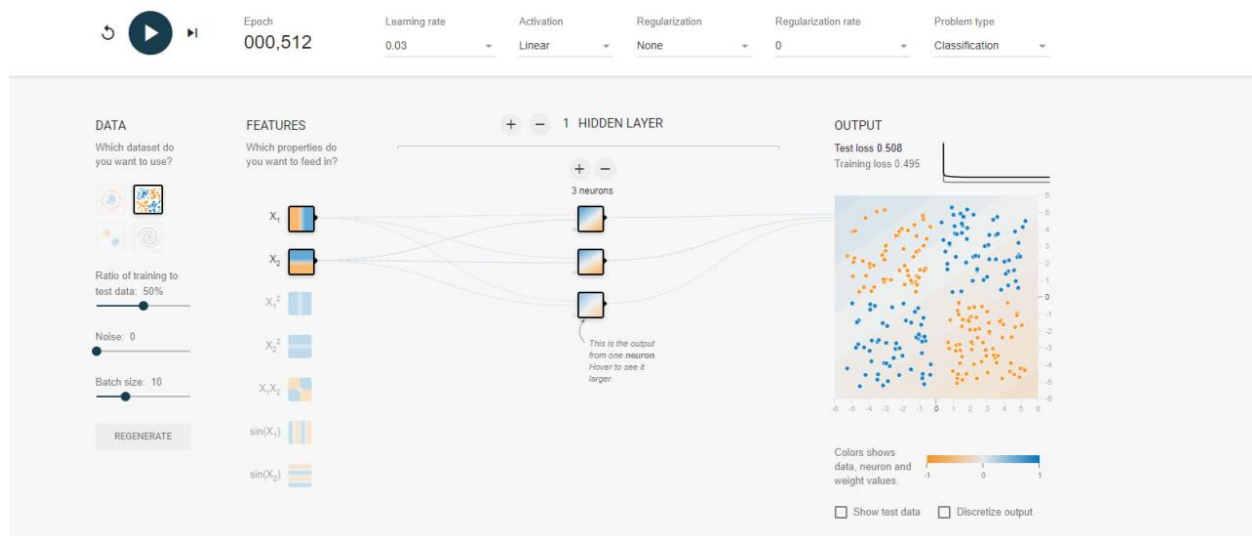
linear (d)



در این نوع دیتا هر 4 تابع فعالسازی به خوبی و حتی با 100 بار تمرین کردن ما را به مطلوب مسئله می‌رسانند

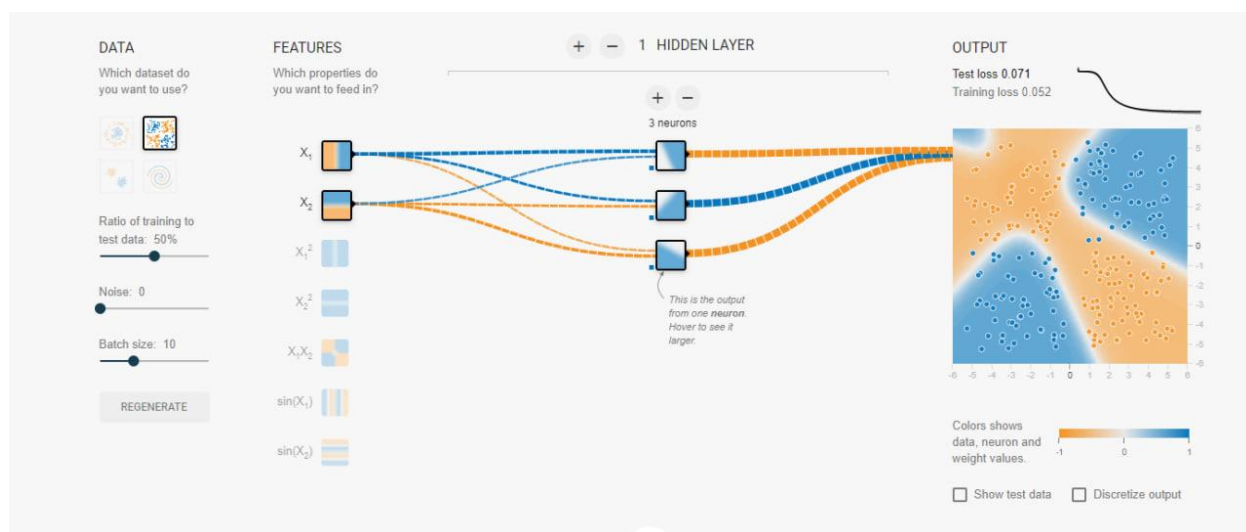
Data = XOR

linear (a



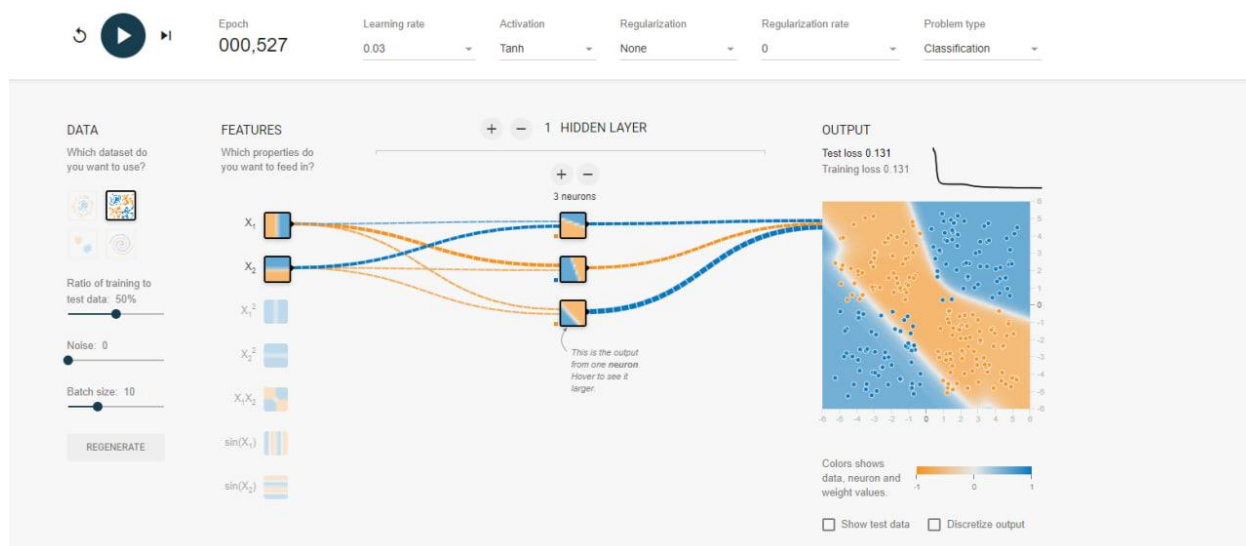
اصلا نتوانسته مطلوب ما را فراهم کنند و برای این نوع دیتا اصلا مناسب نیستند

sigmoid (b)



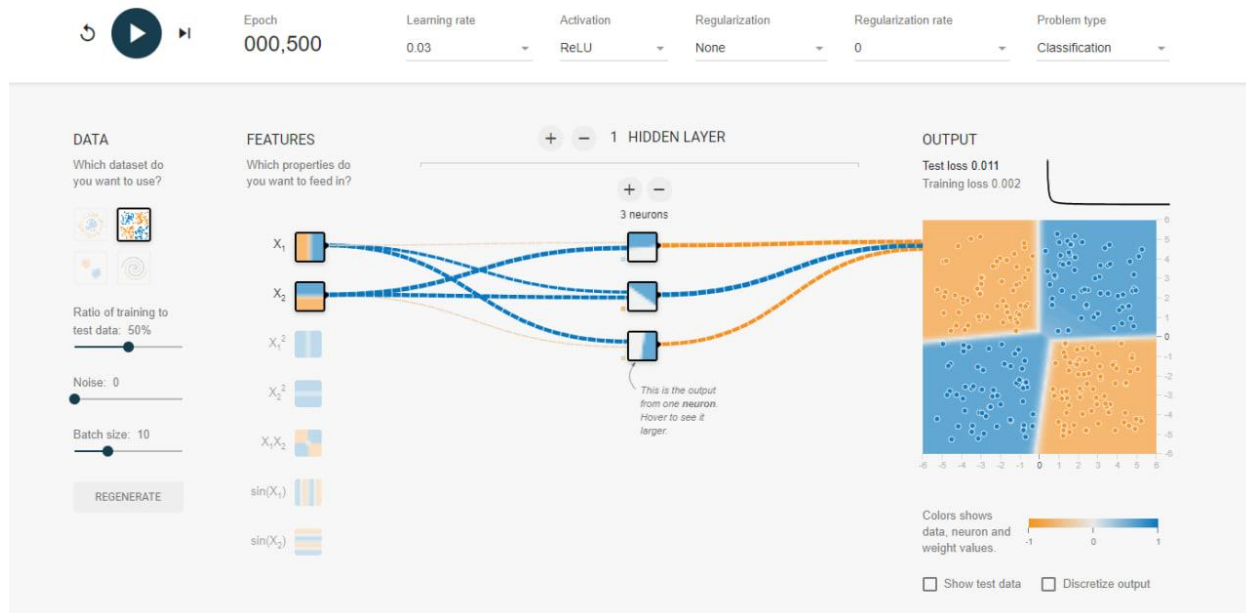
تابع sigmoid تا حد خوبی توانسته پاسخ ما را بدهد اما دیتاهای مرزی قابل توجه هستند و تعدادی دیتا هستند که وارد کلاس دیگری شده اند اما تعدادشان کم است و میشود چشم پوشی کرد و این تابع تا حدودی قابل اتکا هنگام مواجهه با این نوع دیتا هست

tanh (c)



این تابع هم دارای وضعی همانند تابع sigmoid میباشد

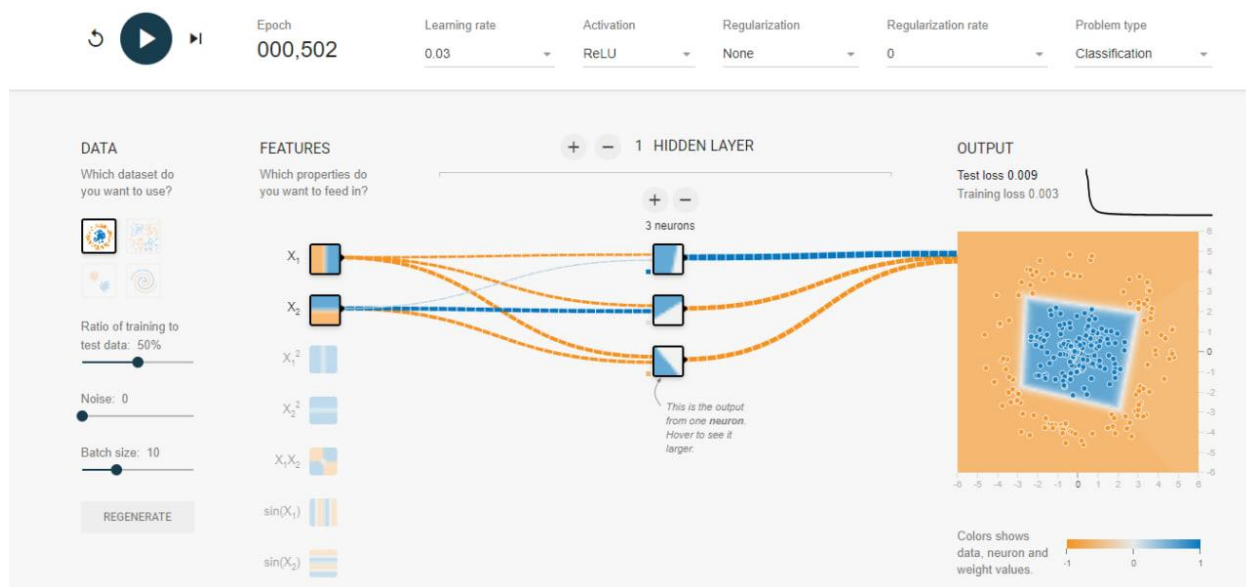
relu (d)



تابع relu برای این نوع دیتا مناسب تر می باشد و طبقه بندی بهتری نسبت به باقی توابع فعال سازی دارد

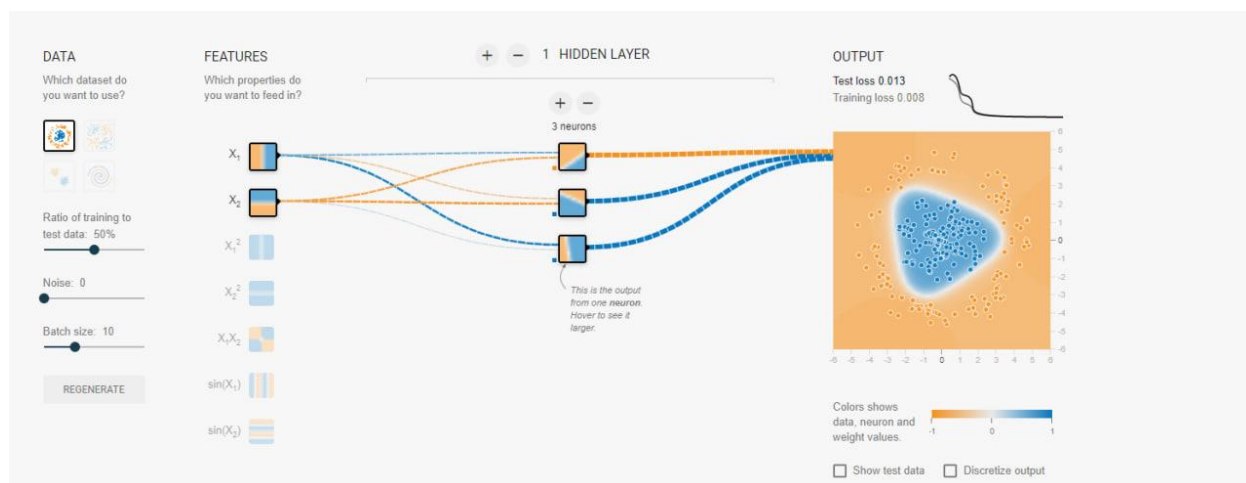
Data = Circle

relu (a)



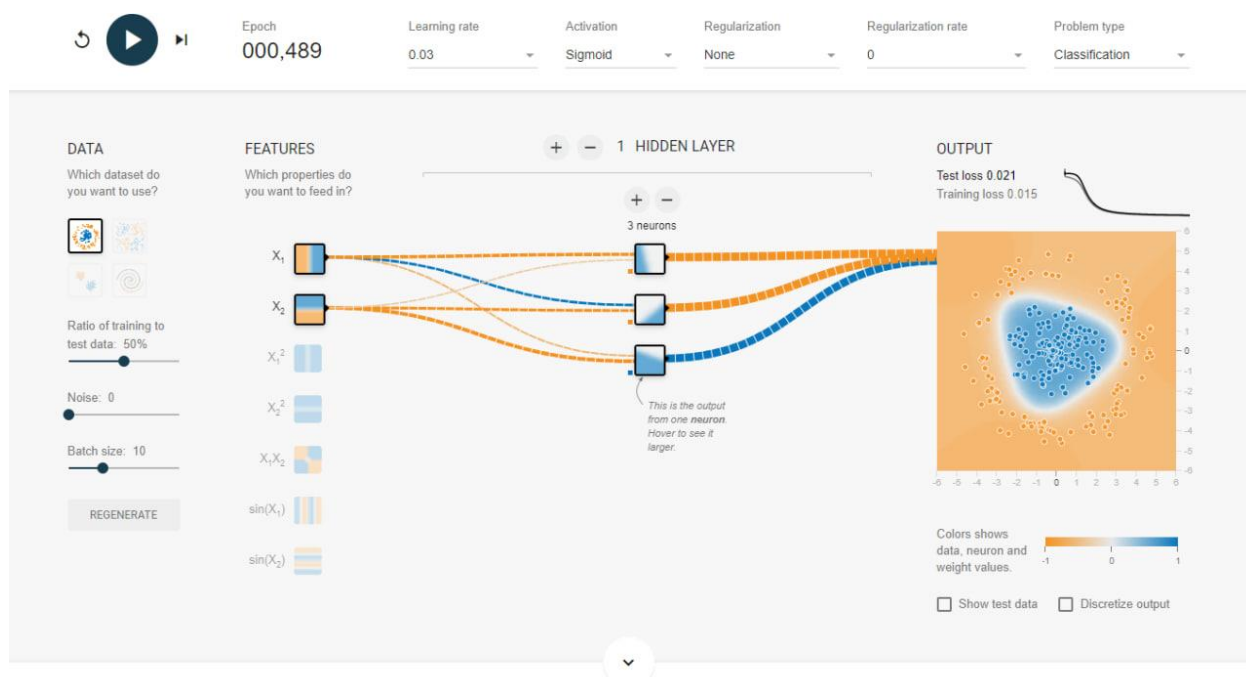
این تابع به خوبی در این نوع دیتا کار میکند و حتی با 100 تمرین هم توانست classification انجام دهد.

\tanh (b)



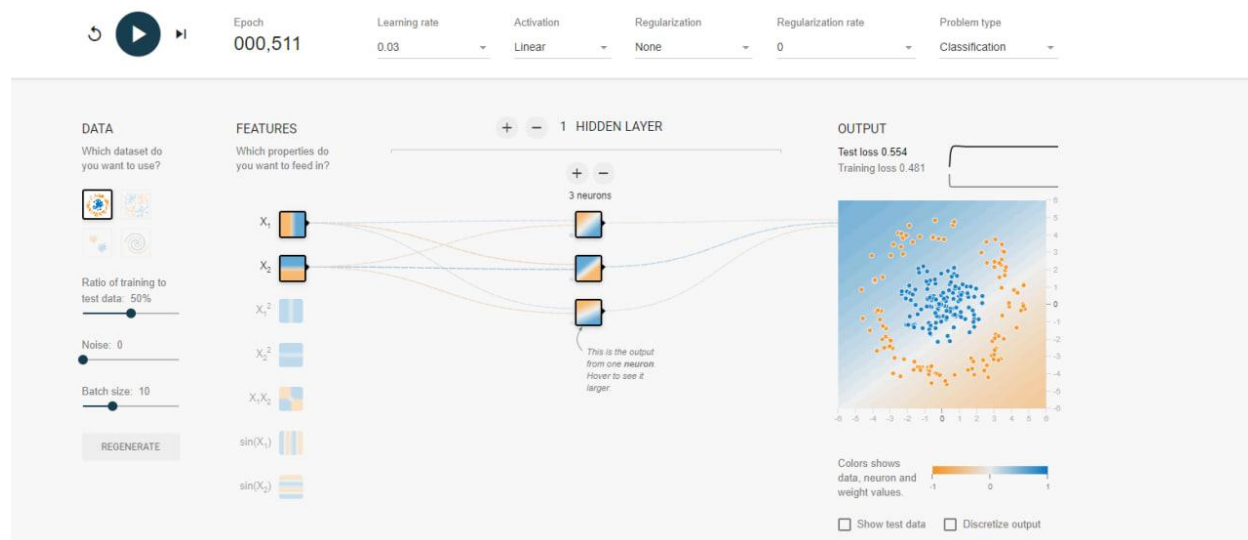
این تابع هم به خوبی پاسخگو بود و سرعتش هم بسیار بالا بود اما سرعت آن کمی کمتر از sigmoid بود

sigmoid (c)



این تابع هم سرعت خوبی داشت و سرعت آن نزدیک به \tanh بود. بعد از 150 تکرار توانست به طبقه بندی برسد

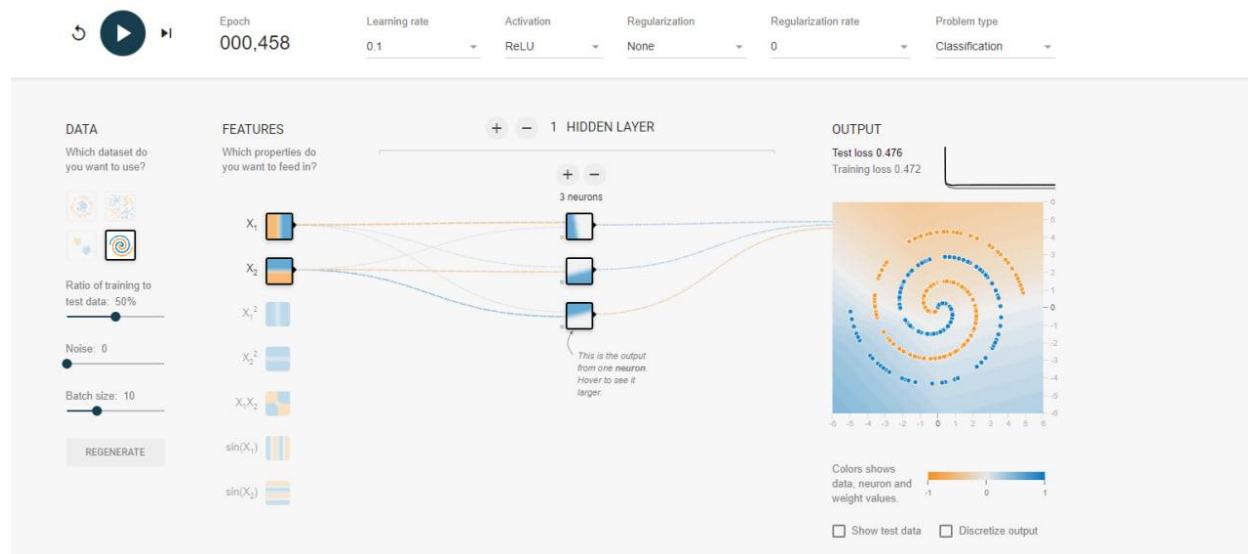
linear (d)



باز هم تابع خطی در برابر این نوع دیتا جوابگو نبود

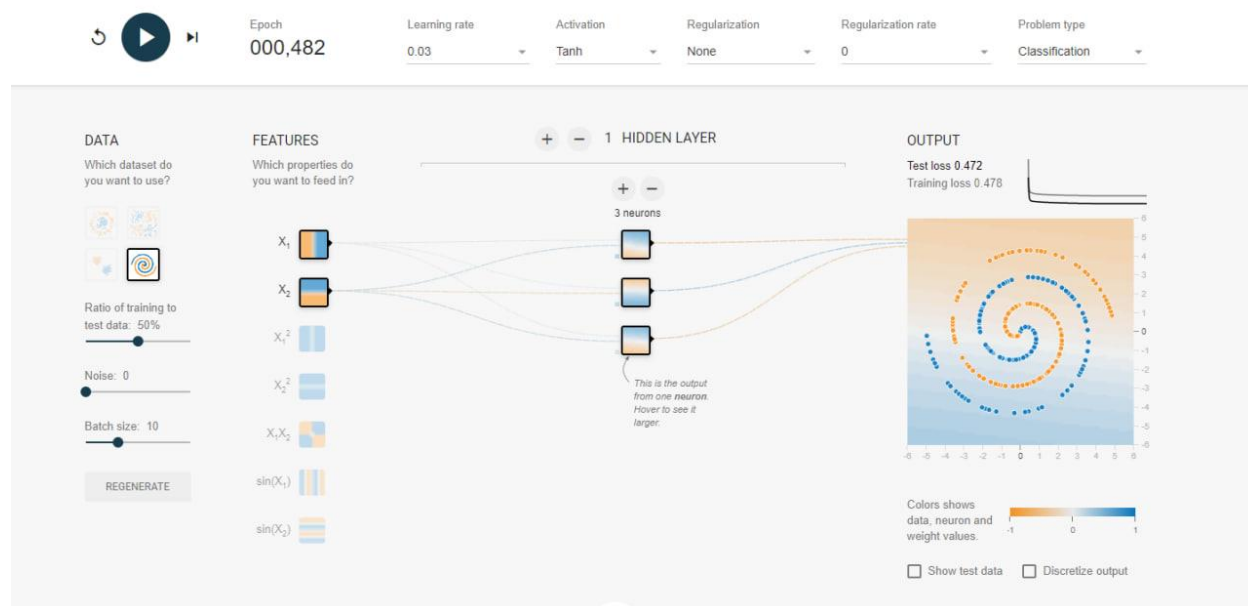
Data = Spiral

relu (a)



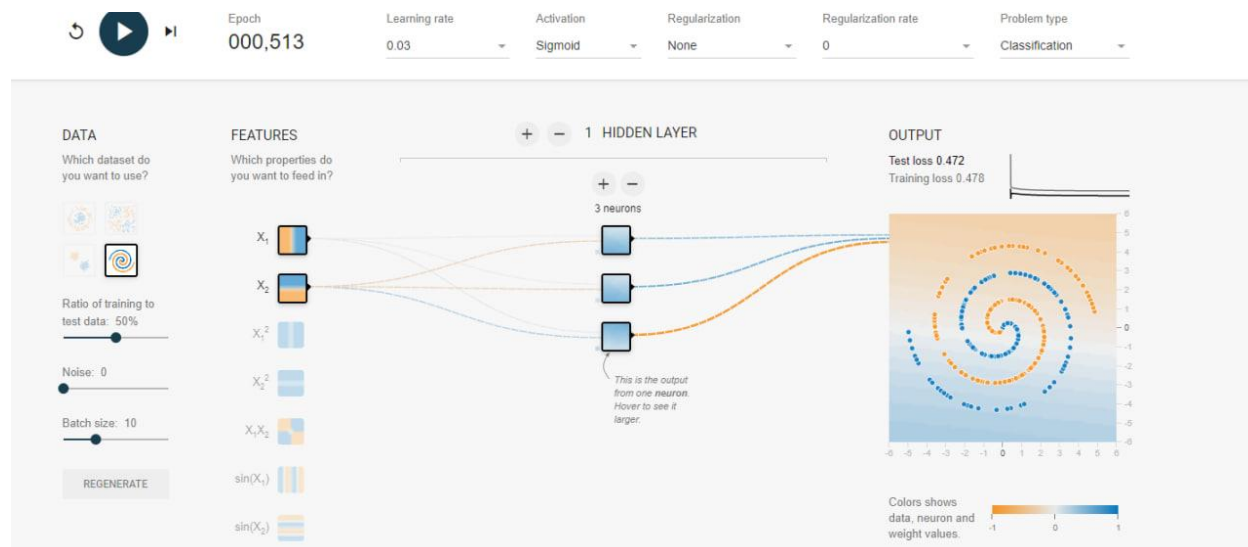
این تابع جوابگو نیست حتی با افزایش نرخ یادگیری

tanh (b)

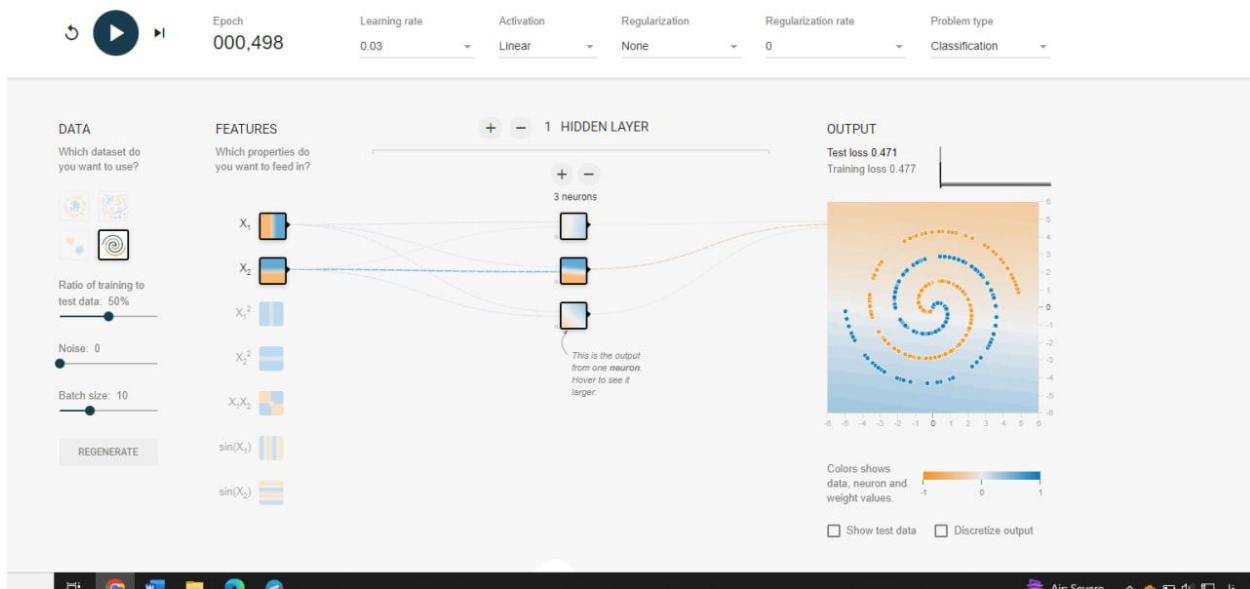


این تابع نیز جوابگو نیست

sigmoid (c)



linear (d)



این تابع نیز مطابق همیشه ناتوان است

(4)

به طور کلی برای ساخت یک شبکه عصبی از `model = Sequential()` استفاده میکنیم
برای ساخت لایه های MLP مدنظر از `model.add(dense())` استفاده میکنیم.
برای تشریح دقیق تر تابع `dense` میتوان گفت که در پارامتر اول ما ابتدا تعداد نوروں ها در آن لایه را مشخص میکنیم سپس ابعاد ورودی را طبق گفته سوال برابر 25 قرار داده ایم.
و سپس در لایه آخر 10 نوروں داریم.

تابع فعال سازی Relu که آنرا در لایه های ورودی قرار دادیم رویکرد غیر خطی دارد و برای مسائل پیچیده مناسب است و از لحاظ محاسباتی دارای `efficiency` مناسبی میباشد و میتواند ما را از مسئله `dying Relu` نجات دهد که در باقی توابع فعالسازی این امکان نیست

تابع فعالسازی softmax که در لایه `output` از آن استفاده میکنیم. مزیت این تابع تبدیل اعداد به احتمالات میباشد که نشان دهنده احتمال وقوع تمام خروجی های ممکن را به ما میدهد.

برای قسمت دوم کد که مدل را کامپایل کردیم در پارامتر `loss` تابع `loss`ی که مدل از آن در مدت `train` استفاده میکند را قرار میدهم که این تابع برای مسائل `multi-class classification` مناسب میباشد.

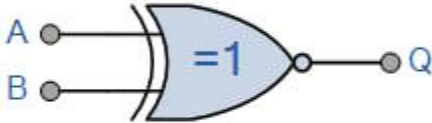
پارامتر `optimizer` الگوریتمی که ما با آن وزن را آپدیت میکنیم را تعیین میکند که از `adam` استفاده میکنیم که با سرعت نسبتا خوبی همگرا میشود. همچنین نرخ یادگیری را تغییر میدهد تا سریعتر به همگرایی برسیم.

منابع:

<https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78>

<https://builtin.com/machine-learning/relu-activation-function>
<https://keras.io/api/optimizers/adam>

(5)

Symbol	Truth Table		
 <p>2-input Ex-NOR Gate</p>	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = \overline{A \oplus B}$	Read if A AND B the SAME gives Q		

دیتا برای training set طبق جدول درستی xnor در نظر میگیریم.
ابتدا مقادیر ثابت را مقدار دهی کردیم مثلا مقادیری برای تعداد iteration ها و نرخ یادگیری تعیین کردیم و تعداد نورون های میانی و ورودی و خروجی را مشخص کردیم.

سپس وزن ها را مقدار دهی اولیه کردیم.

تابع sigmoid/logistic و مشتق آنرا تعریف کردیم.

سپس تمامی خطوط را مطابق با جزوه پیش بردیم و پیمایش را شروع کردیم؛
ابتدا مسیر forward pass چک میشود تا مقدار خروجی ای که پرسپترون چندلایه ما نمایش میدهد با مقدار خروجی حقیقی مقایسه شود و مقدار ارور را محاسبه کنیم.
سپس در مسیر backward pass مطابق با ارور و فرمول هایی که داریم اروری که مربوط به هر نورون میتواند باشد را محاسبه کردیم و وزن را مطابق با آن آپدیت کردیم.

و در نهایی بعد از مرحله train مرحله test را انجام دادیم و به موفقیت رسیدیم.

*تعداد iteration هایی که داشتیم را با اعداد 100 و 1000 امتحان کردیم و به موفقیت نرسیدیم یعنی خروجی ها درست نمیشدند اما با 5000 بار موفق شدیم.

(6)

کد سوال 6 تا حد زیادی مشابه با کد سوال 4 میباشد فلذا مواردی که در این سوال اضافه شده اند را توضیح میدهیم.

دیتاست mnist عکس هایی از اعداد دارد و با خواندن دیتا از این دیتابیس ورودی ما تعداد سمپل های ما و در اندازه $28 * 28$ میباشد.

ابتدا عملیاتی تحت عنوان flattening انجام میدهیم که عمق عکس را بگیریم مثلا اگر فرمت rgb داشته باشیم باید آنرا به grayscale تبدیل کنیم.

در رابطه با خروجی و تابع to categorical این تابع با توجه به این که multi class classification داریم یعنی خروجی به چند کلاس تبدیل میشود استفاده میشود. در رابطه با تابع fit که استفاده کردیم برای train کردن مدل استفاده میشود. سپس در انتها نمودار accuracy , loss را رسم کردیم.

عدد 784 که ابعاد ورودی را نشان میدهد از حاصلضرب $28 * 28$ نشأت میگیرد. برای انتخاب تعداد لایه ها و نوروں ها میتوان گفت بر اساس چندین فاکتور میباشد ولی تعداد دقیق آنرا نمیتوان کامل مشخص کرد اما با در نظر گرفتن مسائلی مانند complexity , overfitting میتوان این موارد را انتخاب کرد.

منابع:

<https://medium.com/@afzbek /how-to-train-a-model-with-mnist-dataset-d79f8123ba84>