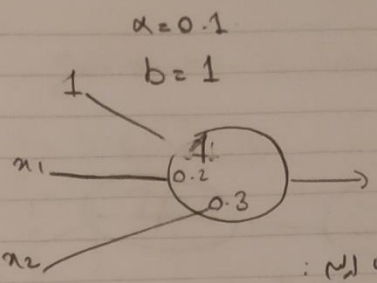
سوال یک

سوال ②:

x_1	x_2	bias	target
1	1	1	-1
1	-1	1	1
-1	1	1	1
-1	-1	1	1



وزن‌های اولیه:

$w_1=0.1 \quad w_2=0.2 \quad b=1$

ورودی اول: (1, 1)

مرحله ①:

$w_1x_1 + w_2x_2 + b$

$0.1 \cdot 1 + 0.2 \cdot 1 + 1 = 1.3$

Delta rule: $b(new)=b(old)+\eta(d-y)$

$w_i(new)=w_i(old)+\eta(d-y)x_i$

new:

$b = 1 + 0.1(-1-1.3) = 1+0.1\cdot(-2.3) = 0.77$

$w_1 = 0.1 + 0.1(-1-1.3)\times 1 = 0.1+0.1(-2.3) = -0.13$

$w_2 = 0.2 + 0.1(-1-1.3)\times 1 = 0.2+0.1(-2.3) = -0.03$

ورودی دوم: (1, -1) مرحله ②

$-0.13 \cdot 1 + -0.03 \cdot -1 + 0.77 = 0.67$

$b = 0.77 + 0.1(1-0.67) = 0.803$

$w_1 = -0.13 + 0.1(1-0.67)\times 1 = -0.097$

$w_2 = -0.03 + 0.1(1-0.67)\times -1 = -0.063$

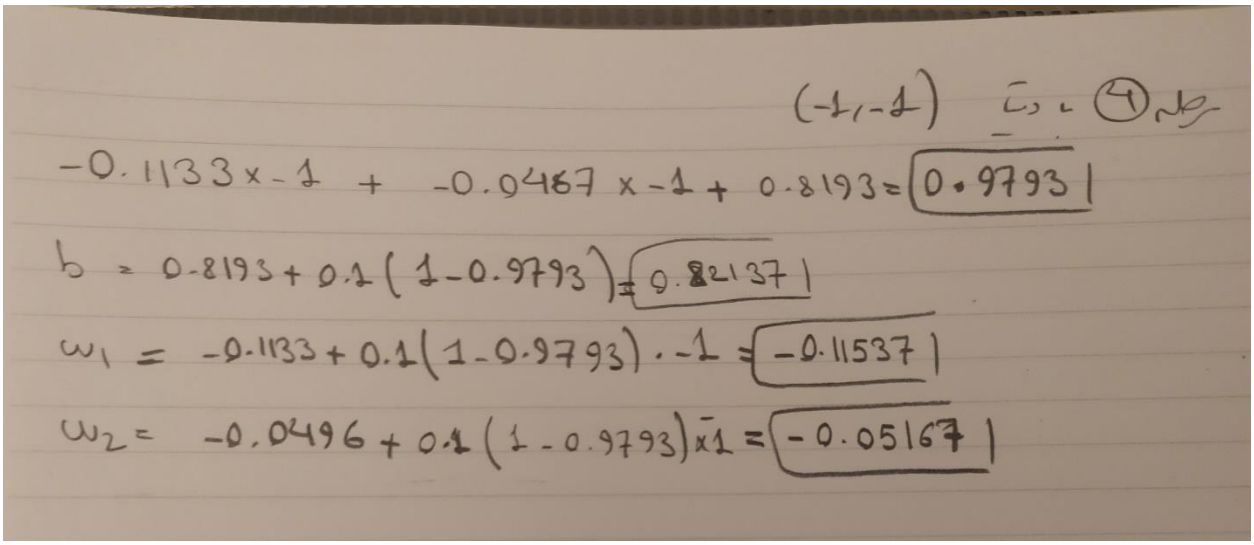
ورودی سوم: (-1, 1) مرحله ③

$-0.097 \times -1 + -0.063 \times 1 + 0.803 = 0.837$

$b = 0.803 + 0.1(1-0.837) = 0.8193$

$w_1 = -0.097 + 0.1(1-0.837)\times -1 = -0.1133$

$w_2 = -0.063 + 0.1(1-0.837)\times 1 = -0.0467$



سوال دو

الف)

تابع فعال‌سازی خطی (Linear Activation Function)

تابع فعال‌سازی خطی یک تابع خطی ساده است که خروجی آن به طور مستقیم و خطی با ورودی مرتبط است. به عبارت دیگر، خروجی تابع فعال‌سازی خطی برابر با یک ضرب ثابت از مقدار ورودی است. معمولاً تابع هویت (Identity Function) به عنوان یک تابع فعال‌سازی خطی استفاده می‌شود. استفاده از تابع فعال‌سازی خطی محدودیت‌هایی را در مدل ایجاد می‌کند و توانایی مدل در یادگیری الگوهای پیچیده را محدود می‌کند. به همین دلیل، توابع غیرخطی معمولاً در شبکه‌های عصبی بیشتر استفاده می‌شوند.

تابع فعال‌سازی غیرخطی (Nonlinear Activation Function)

تابع فعال‌سازی غیرخطی توانایی مدل را در تعبیه الگوهای پیچیده افزایش می‌دهد. این توابع تغییرات غیرخطی را در خروجی شبکه ایجاد می‌کنند و به مدل امکان یادگیری روابط پیچیده‌تری را می‌دهند. توابع فعال‌سازی غیرخطی معمولاً توابعی هستند که خروجی آنها بر اساس توابع غیرخطی و ناخطی از ورودی محاسبه می‌شود، مانند تابع سیگموئید(Sigmoid) ، تابع تانژانت هیپربولیک (Hyperbolic Tangent) و ReLU (Rectified Linear Unit)این توابع به شبکه عصبی اجازه می‌دهند الگوهای پیچیده‌تر را یاد بگیرد و در بسیاری از مسائل عملکرد بهتری نسبت به توابع خطی دارند.

با استفاده از توابع فعال‌سازی غیرخطی، شبکه عصبی قادر است تعبیه الگوهای پیچیده‌تری را انجام دهد و در مسائلی مانند تشخیص تصاویر، ترجمه ماشینی و تشخیص گفتار، بهبود عملکرد مدر مکالمه قبلی توضیح داده شد که تابع فعال‌سازی خطی توانایی مدل را در تعبیه الگوهای پیچیده محدود می‌کند و توابع غیرخطی به مدل

امکان یادگیری روابط پیچیده‌تری می‌دهند. البته توجه داشته باشید که هنوز هم تعدادی مدل و معماری شبکه عصبی وجود دارند که از توابع فعال‌سازی خطی استفاده می‌کنند، اما در بسیاری از موارد توابع غیرخطی مورد استفاده قرار می‌گیرند زیرا عملکرد بهتری ارائه می‌دهند.

ب)

بایاس رندوم و وزن‌ها صفر: در این حالت، بایاس‌ها با مقادیر تصادفی اولیه تنظیم می‌شوند و وزن‌ها با مقدار صفر شروع می‌کنند. این به این معنی است که هر نورون در لایه‌های مختلف شبکه، از یک بایاس تصادفی مستقل از دیگر نورون‌ها استفاده می‌کند و تمام وزن‌های ورودی به آن نورون صفر هستند.

بایاس صفر و وزن‌ها رندوم: در این حالت، بایاس‌ها با مقدار صفر شروع می‌شوند و وزن‌ها با مقادیر تصادفی اولیه تنظیم می‌شوند. این بدین معنی است که هر نورون در لایه‌های مختلف شبکه، از یک بایاس ثابت برابر با صفر استفاده می‌کند و تمام وزن‌های ورودی به آن نورون با مقادیر تصادفی مختلف تنظیم می‌شوند.

در هر دو حالت فوق، پس از تنظیم مقادیر اولیه وزن‌ها و بایاس‌ها، مدل MLP شروع به آموزش می‌کند. روند آموزش شامل مراحل زیر است:

۱. پیش‌بینی: در این مرحله، مدل با استفاده از وزن‌ها و بایاس‌های اولیه، ورودی‌ها را به صورت پیش‌بینی می‌کند. در هر لایه، محاسباتی انجام می‌شود و خروجی‌های لایه‌ها به عنوان ورودی‌های لایه‌های بعدی استفاده می‌شوند.

۲. محاسبه خطا: مقدار خطا بین خروجی مدل و خروجی مورد انتظار محاسبه می‌شود. این خطا معمولاً با استفاده از تابع هدف (مانند خطا میانگین مربعات) محاسبه می‌شود.

۳. به‌روزرسانی وزن‌ها و بایاس‌ها: با استفاده از الگوریتم بهینه‌سازی (مانند روش پس‌انتشار خطا)، وزن‌ها و بایاس‌ها به گونه‌ای تغییر می‌کنند که خطا کاهش یابد. این به‌روزرسانی‌ها براساس gradient descentنسبت به وزن‌ها و بایاس‌ها صورت می‌گیرد و با هدف به‌روزرسانی‌ها انجام می‌شود.

۴. تکرار مراحل ۱ تا ۳: مراحل پیش‌بینی، محاسبه خطا و به‌روزرسانی وزن‌ها و بایاس‌ها تا زمانی که مدل به میزان قابل قبولی دقت برسد یا تا زمانی که شرایط مشخصی برای پایان آموزش تعیین شود.

در بعضی موارد، مقدار دادن بایاس‌ها به صفر می‌تواند منجر به vanishingمیشود. این مشکل زمانی رخ می‌دهد که خروجی لایه‌های پایین‌تر شبکه به دلیل بایاس صفر، به صفر نزدیک باشد و در نتیجه تابع فعال‌سازی (مانند

تابع سیگموئید) نزدیک به زمینه خطی قرار بگیرد. این موضوع می‌تواند باعث از بین رفتن اطلاعات و عدم توانایی شبکه در یادگیری الگوهای پیچیده شود.

از طرف دیگر، اگر تمام وزن‌ها را صفر بدهیم، **همه نورون‌ها در لایه‌ها متصل به هم خروجی یکسانی خواهند داشت و نتیجه طیف محدودی از خروجی‌ها خواهد بود**. این موضوع می‌تواند باعث عدم قدرت شبکه در تقسیم بندی داده‌ها و یادگیری ویژگی‌های مهم شود.

به طور کلی، معمولاً بهتر است مقادیر اولیه وزن‌ها و بایاس‌ها را به صورت تصادفی و با توزیع‌های کوچک و متفاوت انتخاب کنیم. برای مثال، مقادیر اولیه معمولاً از توزیع نرمال با میانگین صفر و انحراف معیار کوچک استفاده بکنیم.

ج)

قابلیت تعمیم در شبکه‌های عصبی بیشتر به عوامل مختلفی بستگی دارد، از جمله معماری شبکه، تعداد لایه‌ها و نورون‌ها، توابع فعال‌سازی، الگوریتم آموزش و حجم و تنوع داده‌های آموزش.

بین شبکه هایی که تا الان خوندیم (MLP) می‌تواند به خوبی قابلیت تعمیم را داشته باشد MLP با داشتن لایه‌های مخفی و تعداد زیادی نورون، قادر است الگوهای پیچیده را یاد بگیرد و در کاربردهای مختلف موفقیت‌آمیز باشد. همچنین، MLPمی‌تواند با استفاده از توابع فعال‌سازی متنوع مانند سیگموئید، تانژانت هیپربولیک و رلو،تطبیق خوبی با داده ها داشته باشد.

اما برای بقیه شبکه هایی که خوندیم پرسپترون، AdalineوMadaline ، قابلیت تعمیم آنها به طور کلی کمتر است. این مدل‌ها به عنوان شبکه‌های تک‌لایه و بدون لایه‌های مخفی، قابلیت یادگیری الگوهای پیچیده را ندارند و در مسال خطی و ساده عملکرد خوبی دارند.

د)

در ابتدا توضیحی در رابطه با خود رابطه میدهم که دقیقا چیکار میکند:

محاسبه Matrix hessian :

در این روش، ابتدا ماتریس هسین (Hessian matrix) محاسبه می‌شود. ماتریس هسین شامل مشتق‌های جزئی دوم تابع هدف نسبت به وزن‌ها است. به عبارت دیگر، این ماتریس نشان می‌دهد که تغییرات وزن‌ها چگونه تاثیر می‌گذارند روی گرادیان تابع هدف.

محاسبه معکوس ماتریس هسین:(Inverse Hessian Matrix)
بعد از محاسبه ماتریس هسین، معکوس آن محاسبه می‌شود. این مرحله نیازمند محاسبه معکوس یک ماتریس است. معکوس ماتریس هسین (Hessian inverse) با نماد H^−1 نمایش داده می‌شود.

محاسبه تغییرات وزن‌ها:
با محاسبه گرادیان تابع هدف نسبت به وزن‌ها$(\partial E/\partial w)$ ، تغییرات وزن‌ها $(\Delta\omega)$ با استفاده از رابطه $\Delta\omega$ $-H\text{^}-1\ \partial E/\partial w\ H$ =محاسبه می‌شود. این رابطه نشان می‌دهد که تغییرات وزن‌ها برابر است با ضرب معکوس ماتریس هسین در گرادیان تابع هدف.

به‌روزرسانی وزن‌ها:
با داشتن تغییرات وزن‌ها$(\Delta\omega)$ ، می‌توان وزن‌ها را به‌روزرسانی کرد. این به‌روزرسانی معمولاً با استفاده از یک الگوریتم بهینه‌سازی مانند گرادیان کاهشی (gradient descent) انجام می‌شود. با اعمال تغییرات وزن‌ها، مدل شبکه MLP بهبود می‌یابد و تطبیق بهتری با داده‌های آموزش می‌کند.

مزایا:

۱. همگرایی سریع: از مزیت‌های اصلی این روش، همگرایی سریع مدل است. با استفاده از اطلاعات ماتریس هسین، تغییرات وزن‌ها به صورت دقیق و هدفمند محاسبه می‌شود که می‌تواند به سرعت بیشتری در رسیدن به نقطه‌ای که تابع هدف به حداقل می‌رسد، کمک کند.

۲. مقاومت در برابر لوکال مینیمم: این روش بهترین راه‌حل ممکن را در نزدیکی نقطه‌ای که تابع هدف به حداقل می‌رسد، پیدا می‌کند. این مزیت به معنای مقاومت در برابر گیر کردن در مینیمم‌های محلی است و می‌تواند به دستیابی به نقاط بهینه برتر در فضای پارامترها کمک کند.

معایب:

۱. پیچیدگی محاسباتی: محاسبه معکوس ماتریس هسین و ضرب آن در گرادیان تابع هدف، محاسبات پیچیده‌ای هستند و نیازمند منابع محاسباتی قابل توجهی هستند. این موضوع می‌تواند زمان زیادی را برای آموزش مدل در برخی موارد مصرف کند.

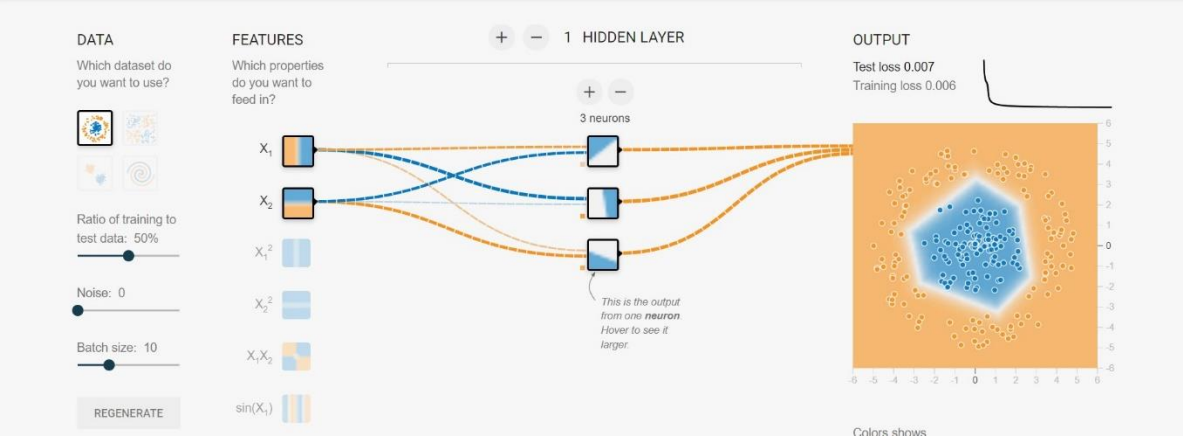
۲. حساسیت به داده‌های نویزی: این روش ممکن است حساسیت بیشتری به داده‌های نویزی داشته باشد. به عبارت دیگر، اگر داده‌های ورودی حاوی نویز زیادی باشند، این روش ممکن است دچار مشکلات همگرایی شود و نتایج آموزشی ضعیفی داشته باشد.

۳. نیاز به اطلاعات دقیق ماتریس هسین: برای استفاده از معکوس ماتریس هسین، نیاز به داشتن اطلاعات دقیق و کامل از ماتریس هسین وجود دارد. این اطلاعات ممکن است در برخی موارد محدود و یا سخت قابل دستیابی باشند.

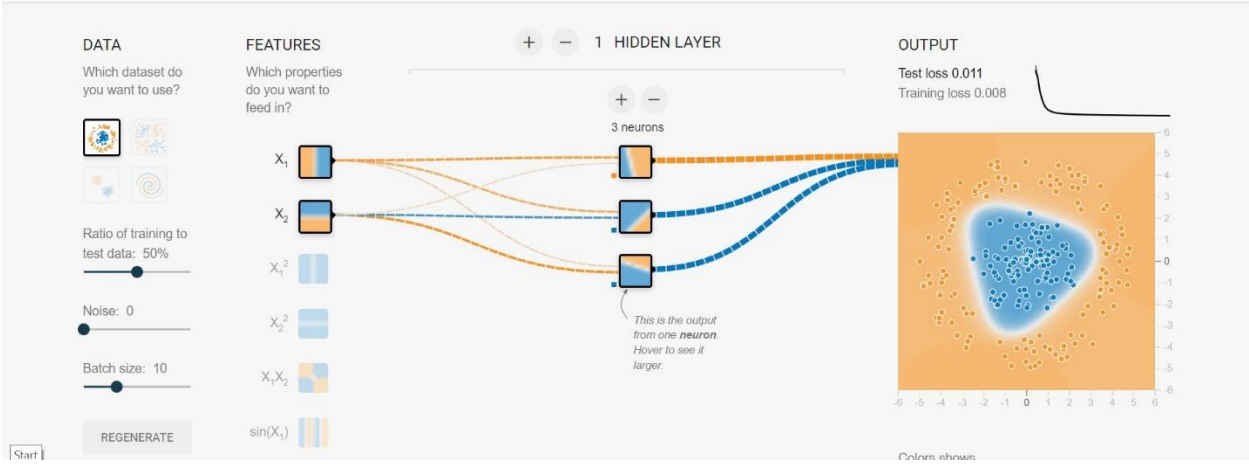
سوال سه

دیتا اول:

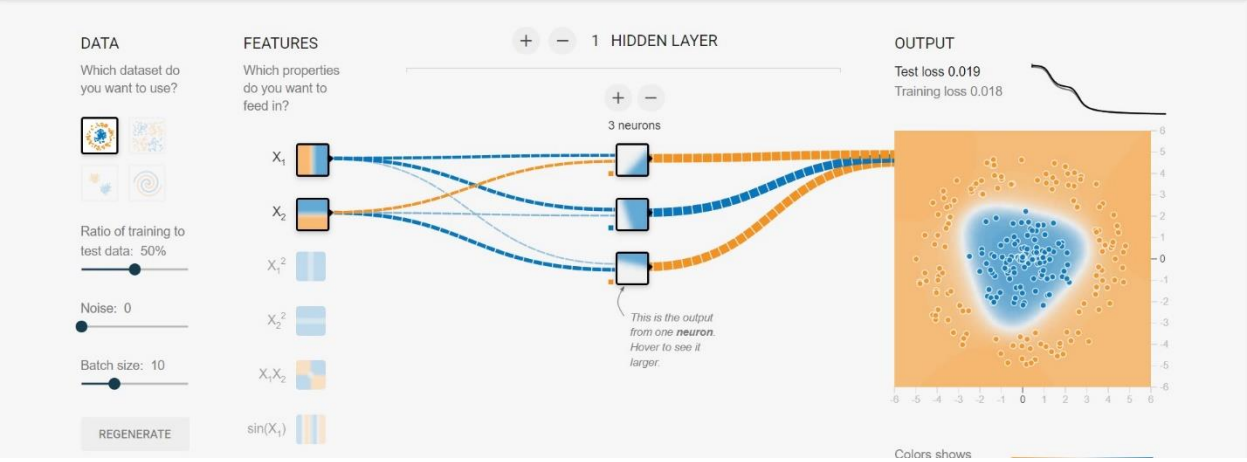
برای دیتا اول با تابع فعال ساز RElu :



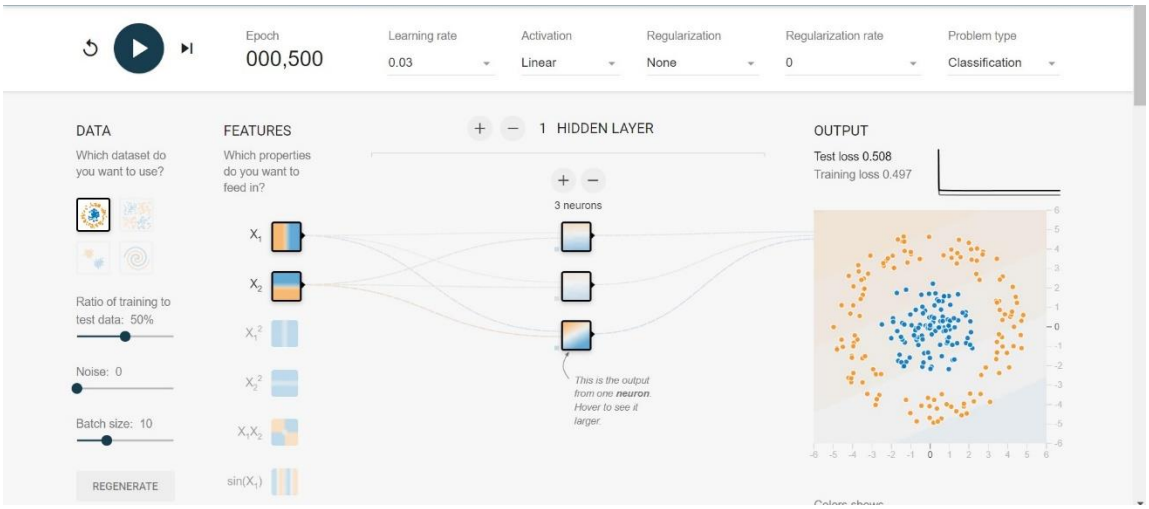
دیتا اول تابع فعال ساز tanh :



دیتا اول تابع فعال ساز sigmoid:



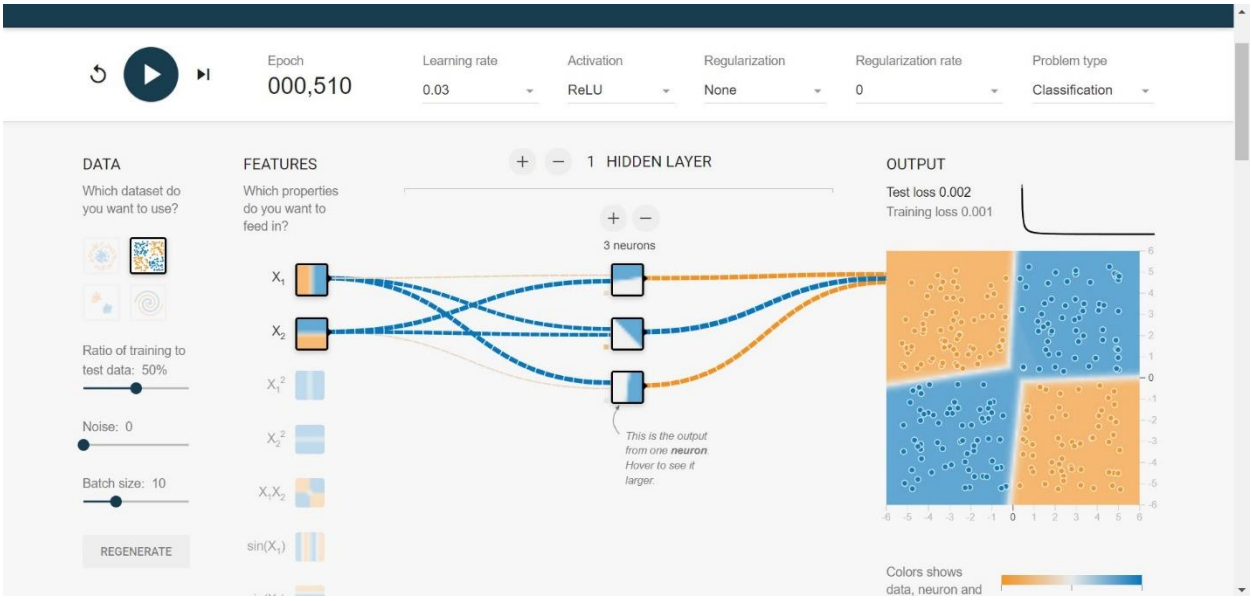
دیتا اول با تابع فعال ساز linear :



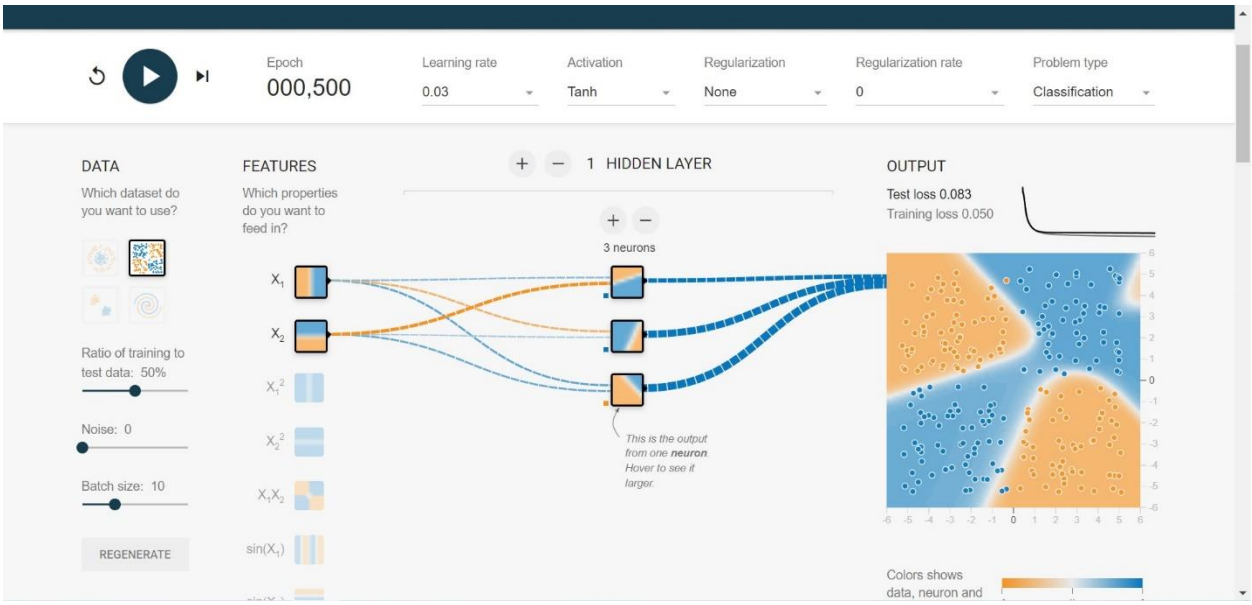
برای دیتاست اول بهترین نتیجه را تابع relu داشت و بدترین نتیجه را تابع linear که طبیعی هست چون دیتاها را با یک خط نمیتوان جدا کرد و هم چنین

دیتا دوم:

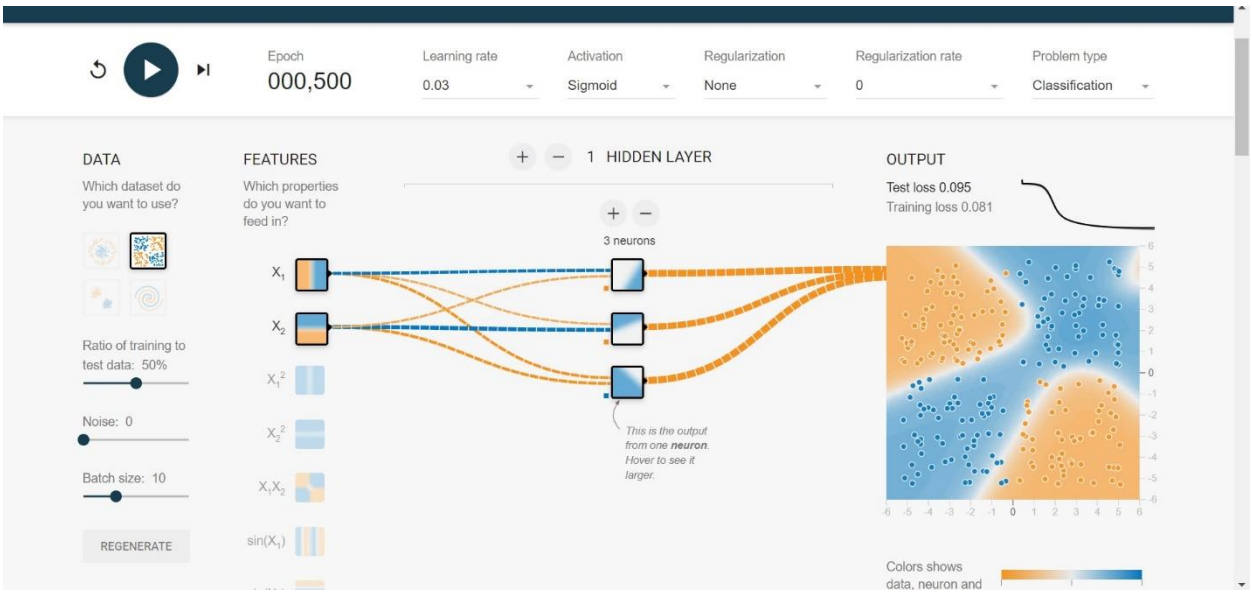
Relu :



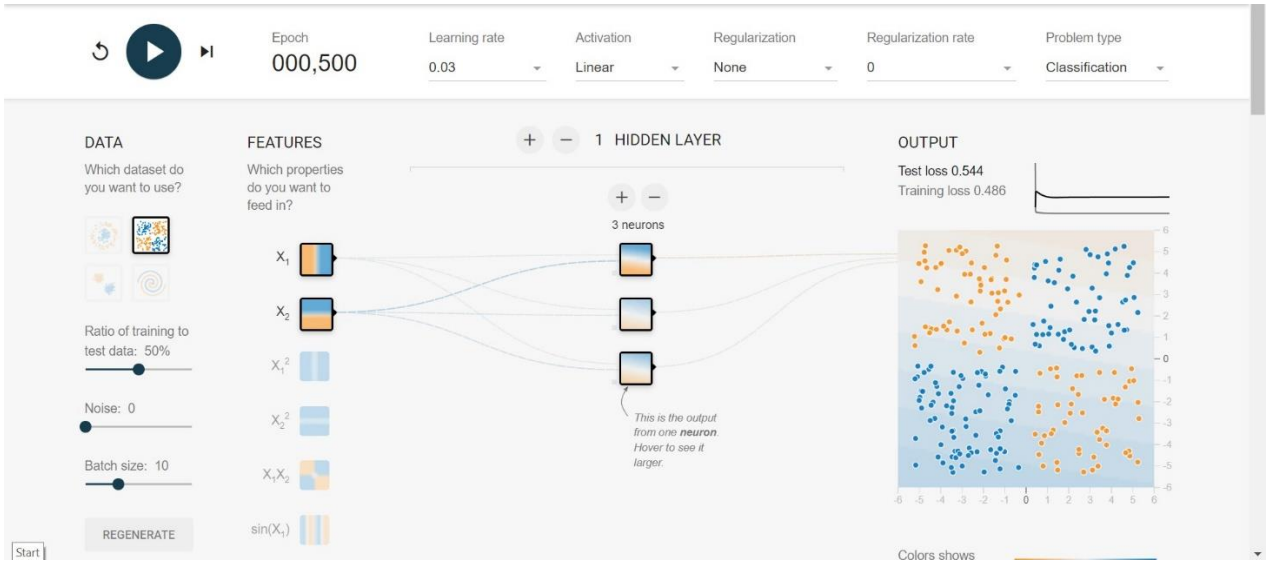
: Tanh



: Sigmoid

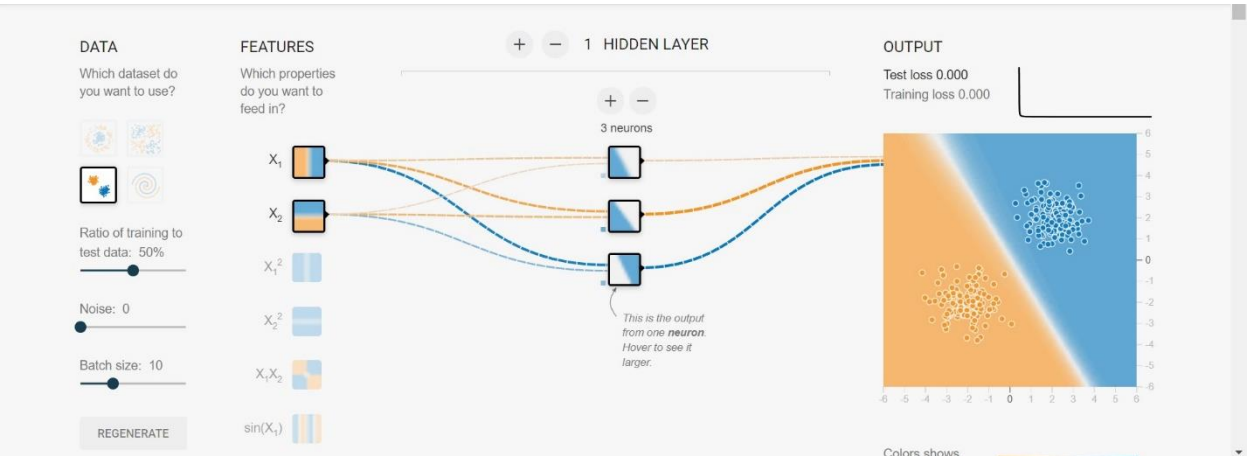


: Linear

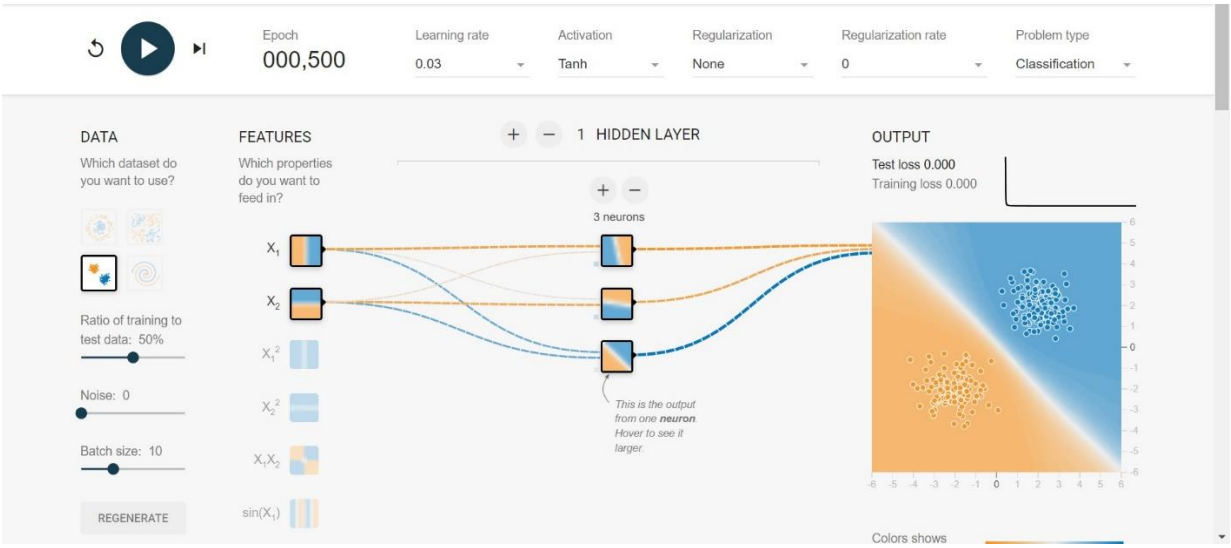


دیتا سوم:

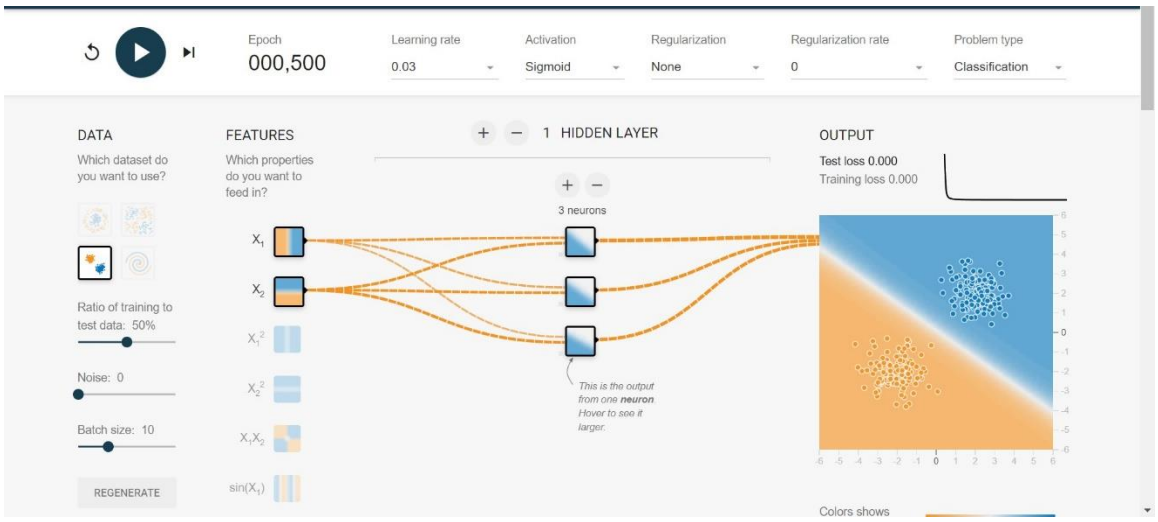
: Relu



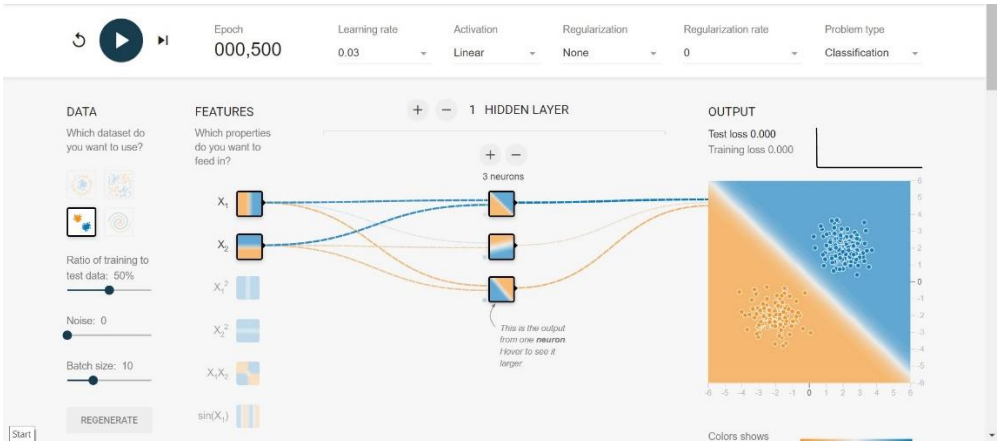
: Tanh



: Sigmoid

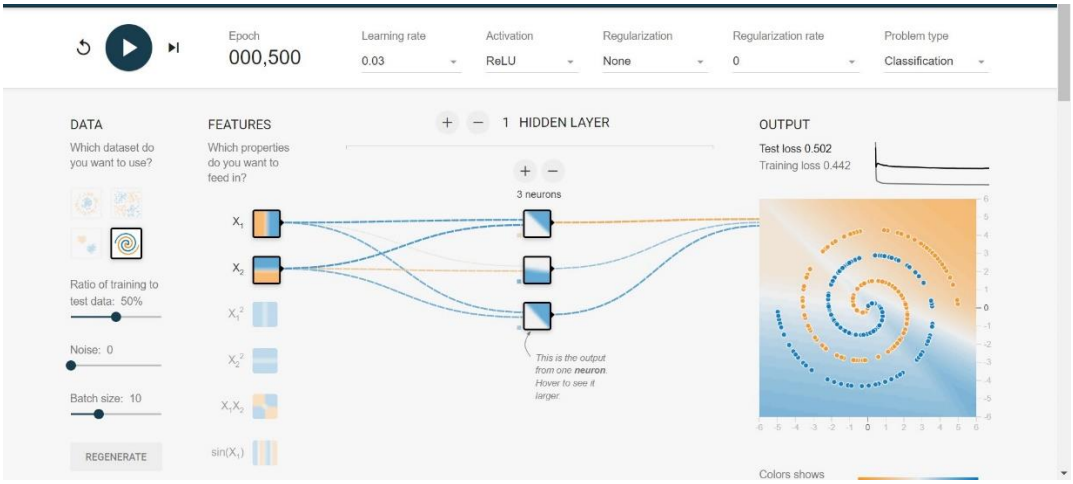


: Linear

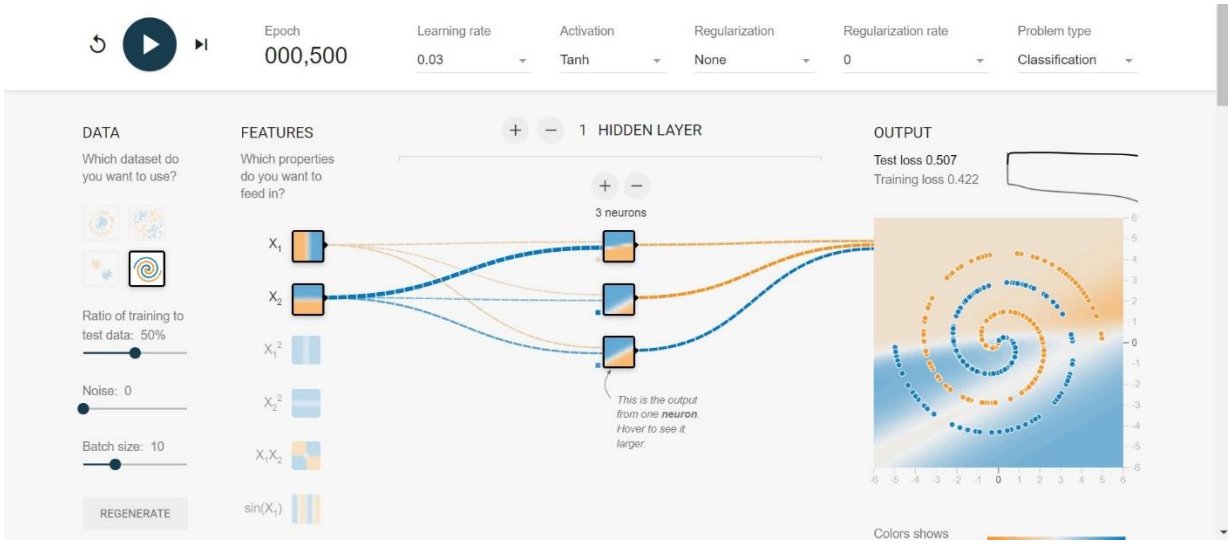


دیتا چهارم:

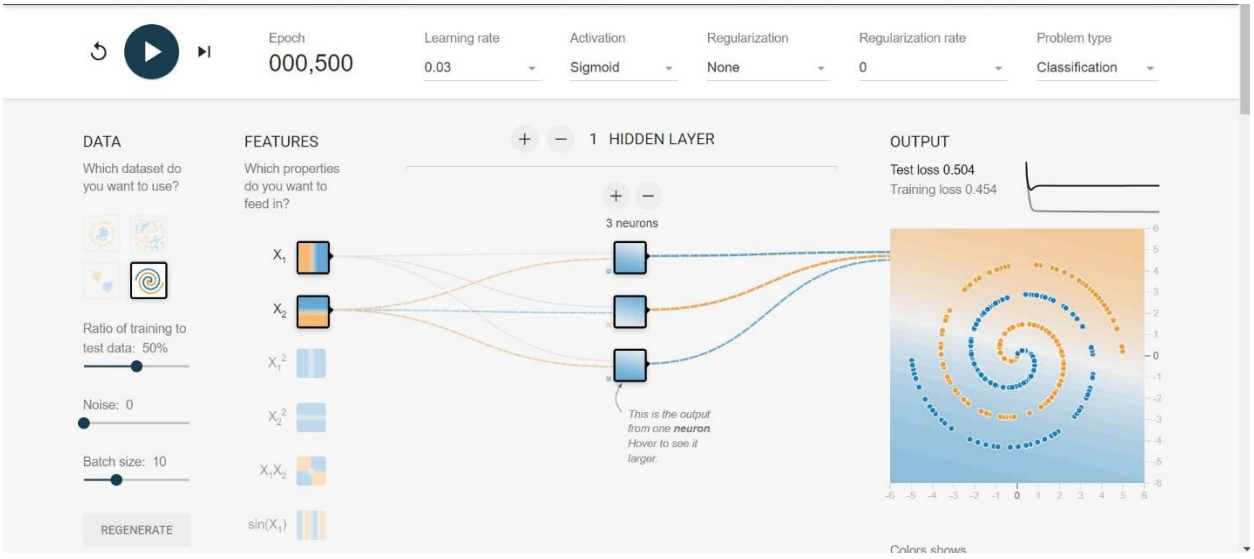
: Relu



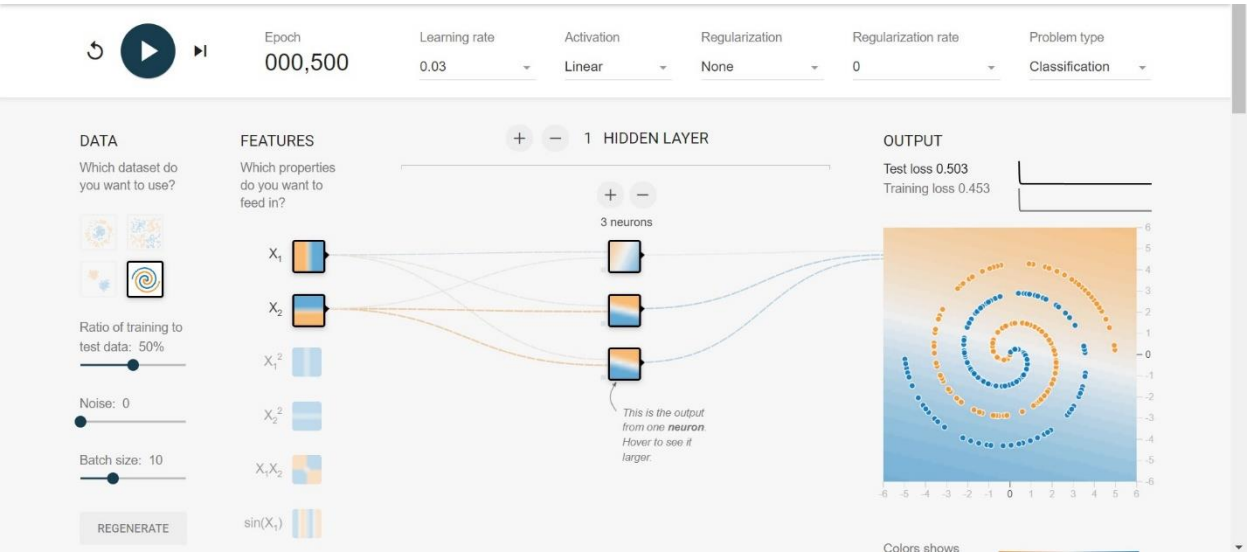
: Tanh



: Sigmoid



: Linear



تحلیل:

برای دیتاست اخر چون جداسازی دو کلاس کمی پیچیده است مشکل از شبکه ساده ای هست که در نظر گرفتیم به همین علت هیچ کدام از توابع فعال سازی نمی تواند خوب جداسازی کند.

برای دیتاست سوم به علت اینکه با یک خط جدا میشود روی تمامی توابع فعالسازی ضرر کمی داریم.

برای تابع linear به این دلیل که جز دیتاست سوم بقیه با خط جدا نمی شوند نمی تواند به خوی تفکیک کند.

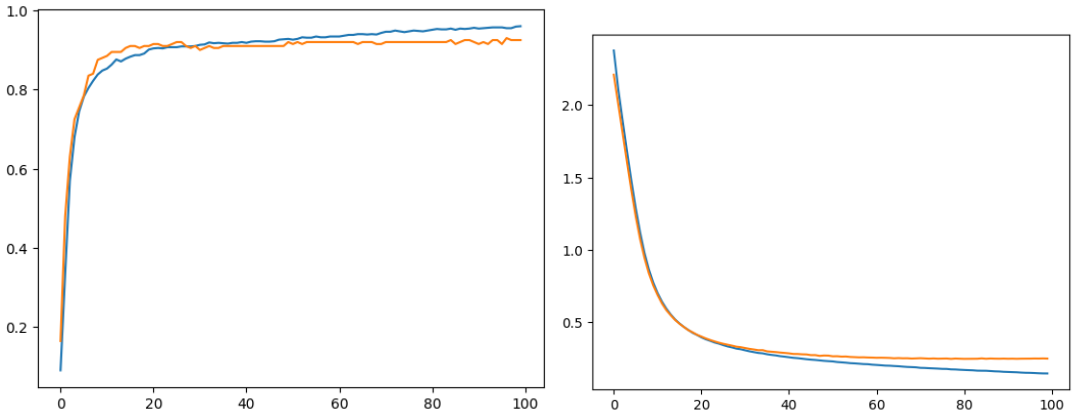
برای دو دیتاست اول relu از بقیه عملکرد بهتری دارد چون مشتق ان به ازای مقادیر زیاد به صفر میل نمی کند و میتواند با خطوط نقاط را تفکیک کند. و بعد از relu عملکرد بهتر را tanh نسبت به sigmoid دارد چون مقادیر خروجی در بازه منفی نیز تولید می کند بر عکس sigmoid ک خروجی ان بین ۰ و۱ هست.

سوال چهار

دو لایه dense که یکی لایه hidden هست را به صورت زیر تعریف می کنیم:

```
model=Sequential()
model.add(Dense(64, activation='relu', input_dim=25))
model.add(Dense(10, activation='softmax'))
```

و در ادامه نمودار های loss و accuracy به صورت زیر میشود:



به میزان بسیار کمی مدل overfit شده است ولی دقت بسیار خوبی دست یافتیم.

سوال پنج

کد من شامل تعدادی لایه ورودی، لایه‌های مخفی و یک لایه خروجی است. در این کد، از تابع فعال‌سازی sigmoidبرای فعال‌سازی خروجی لایه‌های مخفی و خروجی نهایی استفاده شده است. تابع sigmoid برای تبدیل خروجی شبکه به یک مقدار بین ۰ و ۱ استفاده می‌شود.

داده‌های ورودی (X) شامل چهار سطر و دو ستون است که هر سطر یک نمونه داده ورودی را نشان می‌دهد. داده‌های خروجی (y) نیز شامل چهار سطر و یک ستون است که هر سطر مقدار خروجی مورد انتظار برای هر نمونه داده ورودی را نشان می‌دهد.

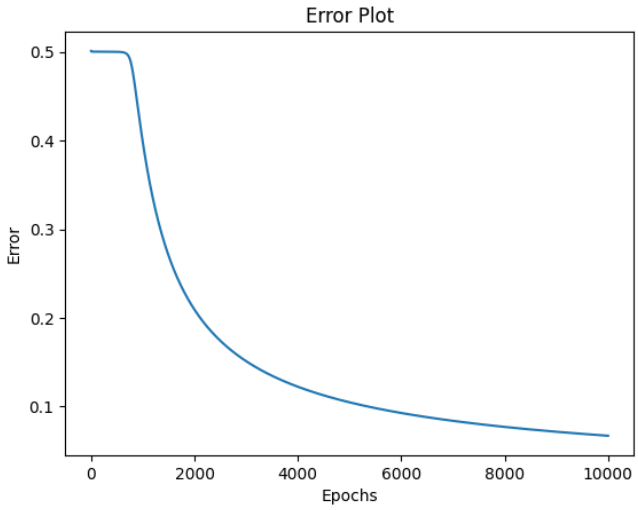
ماتریس وزنیز شامل دو سطر و دو ستون است. هر سطر از ماتریس w1 نشان‌دهنده وزن‌های لایه مخفی است که به داده‌های ورودی وارد می‌شوند. ماتریس w2 نیز شامل دو سطر و یک ستون است و نشان‌دهنده وزن‌های لایه خروجی است.

در هر مرحله از حلقه تکرار(epochs) ، ابتدا با استفاده از فیدفوروارد، خروجی شبکه محاسبه می‌شود. سپس خطا بین خروجی محاسبه شده و خروجی مورد انتظار محاسبه می‌شود. سپس با استفاده از الگوریتم پس‌انتشار خطا، گرادیان خطا نسبت به وزن‌ها محاسبه می‌شود و وزن‌ها به‌روزرسانی می‌شوند. این فرآیند تا به تعداد تکرارها (epochs) ادامه پیدا می‌کند.

در نهایت، با استفاده از وزن‌های به‌روزرسانی شده، خروجی نهایی شبکه محاسبه می‌شود و چاپ می‌شود.

با اجرای این کد، خروجی نهایی شبکه برای داده‌های ورودی X چاپ خواهد شد که خروجی همانطور که در نوتبوک Q5 هست درست هست.

نمودار ارور نیز به صورت زیر است که کاهشی است:



سوال شش

لایه ورودی: لایه ورودی با ۵۱۲ نرون و تابع فعال‌سازی ReLU استفاده شده است.

لایه مخفی: یک لایه مخفی با ۲۵۶ نرون و تابع فعال‌سازی ReLU استفاده شده است. این تعداد نرون به عنوان یک تعداد وسطی برای مدل انتخاب شده است.

لایه خروجی: لایه خروجی با تعداد نرون‌های برابر با تعداد کلاس‌ها (در اینجا ۱۰) و تابع فعال‌سازی softmax استفاده شده است. تابع softmax برای تولید احتمالات کلاس‌ها استفاده می‌شود.

در اینجا، تعداد لایه‌ها و نرون‌ها به صورت تجربی تعیین شده است. معمولاً در مدل‌های MLP برای مسائل پیچیده‌تر، از تعداد لایه‌ها و نرون‌های بیشتر استفاده می‌شود تا مدل بتواند ویژگی‌های پیچیده‌تر را یاد بگیرد. اما برای مسئله MNIST که تشخیص اعداد در تصاویر ساده است، تعداد لایه‌ها و نرون‌های کمتری نیز کافی است.

در این حالت خاص، لایه ورودی شامل ۵۱۲ نرون است تا بتواند ویژگی‌های اولیه تصاویر را استخراج کند. سپس لایه مخفی با ۲۵۶ نرون استفاده شده است تا به عنوان یک نقطه وسطی برای استخراج ویژگی‌های بیشتر از تصاویر عمل کند. در نهایت، لایه خروجی با تعداد نرون‌های برابر با تعداد کلاس‌ها (۱۰) استفاده شده است تا احتمالات مربوط به هر کلاس را تولید کند.

به دقت 0.9933 رو داده های train و به دقت ۰.۹۸ در داده های تست رسیدیم.

```
model = keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')
])
```

و نمودار loss و accuracy ان به صورت زیر است: