به نام خدا

گزارش تمرین دوم هوش محاسباتی

علیرضا اسلامی خواه

99521064

منابع استفاده شده برای سوالات : , chatgpt و playground.tensorflow

گزاش سوال 1 در pdf جداگانه ای شرح داده شده است.

سوال 2 :

الف)

مهم نیست که چند لایه پنهان را در شبکه عصبی متصل می کنیم، همه لایه ها به یک شکل رفتار می کنند زیرا ترکیب دو تابع خطی خود یک تابع خطی است. نورون فقط با یک تابع خطی متصل به آن نمی تواند یاد بگیرد. یک تابع فعال‌سازی غیر خطی به آن اجازه می‌دهد طبق خطای تفاوت w.r.t یاد بگیرد. از این رو ما به یک تابع فعال سازی نیاز داریم.

توابع فعال سازی با وارد کردن غیر خطی بودن به مدل، نقش مهمی در شبکه های عصبی مصنوعی بازی می کنند. تفاوت اصلی بین توابع فعال سازی خطی و غیر خطی در نحوه تبدیل آنها به داده های ورودی نهفته است.

1. تابع فعال سازی خطی:

- یک تابع فعال سازی خطی مجموع وزنی ورودی های خود را محاسبه می کند و هیچ تبدیل غیرخطی اعمال نمی کند. خروجی نسبت مستقیمی با ورودی دارد.

- از نظر ریاضی، یک تابع فعال سازی خطی را می توان به صورت f(x) = ax نشان داد که در آن 'a' یک ضریب ثابت است. هیچ گونه انحنا یا غیرخطی به مدل وارد نمی کند.

- وقتی چندین لایه در یک شبکه عصبی از توابع فعال سازی خطی استفاده می کنند، کل شبکه مانند یک مدل خطی رفتار می کند و روی هم قرار گرفتن لایه های خطی ظرفیت یا قدرت بیان مدل را افزایش نمی دهد.

2. تابع فعال سازی غیر خطی:

- توابع فعال‌سازی غیرخطی، غیرخطی بودن را به شبکه عصبی وارد می‌کند و به آن اجازه می‌دهد تا روابط پیچیده و غیرخطی را در داده‌ها یاد بگیرد.

- برخی از توابع فعال سازی غیر خطی رایج عبارتند از:

- Sigmoid: f(x) = 1 / (1 + e^(-x))، که ورودی را در محدوده (0، 1) له می کند.

- مماس هایپربولیک (Tanh): f(x) = (e^(x) - e^(-x)) / (e^(x) + e^(-x))، که ورودی را در محدوده (-1، 1).

- واحد خطی اصلاح شده (ReLU): f(x) = max(0, x) که در صورت مثبت بودن ورودی و در غیر این صورت صفر خروجی می دهد.

- ReLU نشتی: f(x) = x اگر x > 0 باشد، در غیر این صورت f(x) = ax، که در آن 'a' یک ثابت مثبت کوچک است.

- پارامتریک ReLU (PReLU): گسترشی از Leaky ReLU که در آن 'a' در طول آموزش یاد می‌شود.

- واحد خطی نمایی (ELU): یک تقریب صاف از ReLU که قابل تمایز است و مقادیر منفی را مجاز می کند.

توابع فعال‌سازی غیرخطی حیاتی هستند زیرا شبکه‌های عصبی را قادر می‌سازند تا روابط پیچیده در داده‌ها را مدل‌سازی کنند، و آنها را قادر به یادگیری و نمایش طیف وسیعی از توابع می‌کند. بدون توابع فعال‌سازی غیرخطی، یک شبکه عصبی به تبدیل‌های خطی محدود می‌شود و ظرفیت آن برای حل وظایف پیچیده به شدت محدود می‌شود.

در عمل، ReLU و انواع آن مانند Leaky ReLU و ELU اغلب برای لایه‌های پنهان در شبکه‌های عصبی عمیق به دلیل کارایی محاسباتی و توانایی آنها در کاهش مشکل گرادیان ناپدید شدن ترجیح داده می‌شوند. با این حال، انتخاب تابع فعال‌سازی می‌تواند به مشکل و معماری خاص بستگی داشته باشد و اغلب برای تعیین مناسب‌ترین تابع فعال‌سازی برای یک کار مشخص، آزمایش لازم است.

ب)

در یک مدل MLP (Multi-Layer Perceptron)، مقادیر اولیه وزن‌ها و بایاس‌ها می‌توانند تأثیر زیادی بر فرآیند آموزش داشته باشند. در اینجا، دو حالت مختلف برای مقادیر اولیه وزن‌ها و بایاس‌ها را بررسی خواهیم کرد:

1. بایاس رندوم و وزن‌ها صفر:

- اگر بایاس‌ها به صورت رندوم انتخاب شوند و وزن‌های تمام لایه‌ها صفر باشند، مدل ابتدا به مشکلاتی برخورده و فرآیند آموزش به دشواری پیش خواهد رفت.

- زمانی که وزن‌ها صفر هستند، تمام نورون‌ها در یک لایه خروجی صفر می‌دهند. این به معنای این است که مدل بسیار محدود و خطی است و توانایی یادگیری نمی‌تواند داشته باشد.

- با این تنظیمات، شبکه به راحتی قادر به یادگیری هر نوع الگو یا اطلاعات پیچیده‌تر نخواهد بود و آموزش مدل به احتمال زیاد سریعاً متوقف می‌شود.

2. بایاس صفر و وزن‌ها رندوم:

- در این حالت، بایاس‌ها برابر با صفر تعیین شده‌اند که در اصل به تحریک مقدماتی نورون‌ها اضافه می‌شود و به معنای این است که تمام نورون‌ها از ابتدا فعالیت خواهند کرد.

- وزن‌های تصادفی به هر نورون نوعی تنوع در یادگیری می‌دهند. این تنوع اجازه می‌دهد که مدل از ابتدا انواع الگوها را تجسم کند.

در این سناریو، مدل با مقادیر سوگیری صفر و مقادیر وزن تصادفی برای همه اتصالات بین نورون ها شروع می شود.

سوگیری های صفر نشان می دهد که پیش بینی های اولیه متقارن (برابر) هستند و ممکن است الگوهای اساسی در داده ها را ثبت نکنند.

از سوی دیگر، وزن‌های تصادفی، مقداری عدم تقارن اولیه را در پیش‌بینی‌های مدل ایجاد می‌کنند که می‌تواند برای یادگیری مفید باشد.

در طول آموزش، الگوریتم پس انتشار وزن‌ها را تنظیم می‌کند و عدم تقارن را در مدل برای برازش بهتر داده‌ها معرفی می‌کند.

آموزش در این سناریو ممکن است سریع‌تر از سناریوی قبلی همگرا شود زیرا وزن‌های تصادفی اولیه می‌تواند به مدل کمک کند تا الگوها را بلافاصله از داده‌ها شروع کند.

در عمل، استفاده از مصالحه بین این سناریوهای افراطی معمول است. تکنیک‌های اولیه‌سازی وزن مانند مقداردهی اولیه Xavier/Glorot یا مقداردهی اولیه He اغلب برای تنظیم وزن‌ها و بایاس‌های اولیه به‌گونه‌ای استفاده می‌شوند که عدم تقارن و پایداری گرادیان را برای تمرین سریع‌تر و پایدارتر متعادل کند. این تکنیک ها برای ارائه ویژگی های همگرایی بهتر و اجتناب از برخی مسائل مرتبط با مقداردهی اولیه کاملا تصادفی یا صفر طراحی شده اند. انتخاب وزن اولیه و بایاس می تواند به طور قابل توجهی بر عملکرد و سرعت همگرایی شبکه عصبی شما تأثیر بگذارد.

- این تنظیمات از نقطه شروع خوبی برای آموزش مدل هستند. از اینجا، مدل می‌تواند وزن‌ها را به تدریج بهینه‌سازی کند و الگوهای پیچیده‌تر را یاد بگیرد.

در کل، مقادیر اولیه وزن‌ها و بایاس‌ها برای آموزش مدل MLP بسیار مهم هستند. حالت دوم (بایاس صفر و وزن‌ها رندوم) معمولاً برای شروع آموزش مدل‌های عمیق توصیه می‌شود، زیرا از نقطه شروع مناسبی برای یادگیری الگوها و ویژگی‌های داده‌ها شروع می‌کند.

ج)

توانایی یک شبکه عصبی برای تعمیم به عوامل مختلفی از جمله معماری آن، داده های آموزشی و مشکل خاصی که روی آن اعمال می شود بستگی دارد. تعمیم به ظرفیت شبکه برای پیش‌بینی دقیق داده‌های دیده نشده اشاره دارد، که یک جنبه حیاتی از یادگیری ماشین است. در اینجا ما هر یک از شبکه های عصبی را که ذکر کردید از نظر قابلیت تعمیم آنها به اختصار مورد بحث قرار دهیم:

Artificial Neuron:

نورون های مصنوعی یا تک نورون ها ساده ترین شکل شبکه های عصبی هستند.

آنها عموماً در توانایی خود برای تعمیم محدود هستند، به ویژه هنگامی که با الگوهای پیچیده و غیر خطی در داده ها سروکار دارند.

تک نورون ها معمولاً برای کارهای خطی یا به عنوان بلوک های ساختمانی در معماری های پیچیده تر شبکه عصبی استفاده می شوند.

Perceptron:

پرسپترون ها شبکه های عصبی تک لایه با خروجی دودویی (0 یا 1) هستند. آنها می توانند به خوبی برای مسائل قابل جداسازی خطی تعمیم دهند، اما در مورد داده های غیرخطی قابل جداسازی محدودیت هایی دارند.

پرسپترون ها عمدتاً برای وظایف طبقه بندی باینری با مرزهای تصمیم خطی مناسب هستند.

Adaline :

آدالین نسخه پیشرفته تری از پرسپترون است و می تواند برای مسائل قابل جداسازی خطی بهتر تعمیم دهد.

با این حال، مانند پرسپترون، آدالین برای مدیریت داده های غیرخطی قابل تفکیک مناسب نیست.

: Madaline

مدالین شبکه‌ای از آدالین است و می‌تواند برای مسائل پیچیده‌تر قابل تفکیک خطی تعمیم دهد.با این وجود، هنوز با داده های غیرخطی قابل تفکیک مبارزه می کند.

: Kohonen

نقشه های خودسازماندهی Kohonen شبکه های عصبی بدون نظارت هستند که برای خوشه بندی و تجسم استفاده می شوند.آنها می توانند به خوبی برای کارهای خوشه بندی و تجسم داده ها تعمیم دهند.قابلیت تعمیم آنها به توزیع داده ها و مشکل موجود بستگی دارد.

: MLP

MLPها شبکه های عصبی قدرتمندتری با لایه های پنهان متعدد و توابع فعال سازی غیر خطی هستند.

MLPها قادر به تعمیم برای طیف وسیعی از وظایف، از جمله مسائل خطی و غیر خطی هستند.عملکرد تعمیم آنها اغلب به عواملی مانند معماری شبکه، کیفیت داده های آموزشی و تنظیم هایپرپارامتر بستگی دارد.

به طور کلی، پرسپترون‌های چندلایه (MLP) یکی از متنوع‌ترین و قدرتمندترین معماری‌های شبکه عصبی هستند که قادر به تعمیم خوبی برای طیف وسیعی از وظایف هستند. آنها می توانند توابع پیچیده را تقریب بزنند و به طور گسترده در برنامه های مختلف، از جمله تشخیص تصویر، پردازش زبان طبیعی، و بسیاری دیگر از وظایف یادگیری ماشین استفاده می شوند. با این حال، قابلیت تعمیم هر شبکه عصبی به ویژگی های مشکل، کیفیت و کمیت داده های آموزشی و تنظیم مناسب فراپارامترها بستگی دارد.

بعد از آن kohonen و سپس سایر شبکه های عصبی در این رده بندی قرار میگیرند.

د)

فرمولی که در صورت سوال نوشته شده مربوط به محاسبه به روز رسانی وزن در یک شبکه عصبی با استفاده از روش مرتبه دوم، به ویژه ماتریس Hessian، برای بهینه سازی است. این روش به "روش نیوتن" یا "بهینه سازی مرتبه دوم" معروف است. هم مزایا و هم معایبی دارد:

مزایا:

همگرایی سریعتر: روشهای مرتبه دوم، از جمله روش نیوتن، در صورت استفاده مناسب می توانند سریعتر از روشهای مرتبه اول (مانند نزول گرادیان) همگرا شوند. آنها هم شیب و هم انحنای تابع از دست دادن را در نظر می گیرند، که می تواند به مسیرهای مستقیم تری به راه حل بهینه منجر شود.

نرخ یادگیری بهبود یافته: با در نظر گرفتن ماتریس Hessian، این روش‌ها می‌توانند به طور تطبیقی نرخ یادگیری را برای هر وزن تنظیم کنند، و امکان به‌روزرسانی‌های بزرگ‌تر در مناطق مسطح و به‌روزرسانی‌های کوچک‌تر در مناطق شیب‌دار چشم‌انداز از دست دادن را فراهم کنند.

همگرایی جهانی: روش های مرتبه دوم تمایل به تضمین نظری بهتری برای رسیدن به حداقل جهانی تابع ضرر تحت شرایط خاص دارند.

معایب:

پیچیدگی محاسباتی: محاسبه و معکوس کردن ماتریس هسین یا حل سیستم خطی نشان داده شده توسط آن می تواند از نظر محاسباتی گران باشد، به خصوص برای شبکه های عصبی بزرگ با پارامترهای متعدد. این پیچیدگی می تواند روش های درجه دوم را برای یادگیری عمیق کمتر کاربردی کند.

نیازهای حافظه: ذخیره و دستکاری ماتریس Hessian یا تقریب آن به حافظه قابل توجهی نیاز دارد که ممکن است برای مدل های بزرگ امکان پذیر نباشد.

حساس به شرایط اولیه: روش های مرتبه دوم می توانند به انتخاب شرایط اولیه حساس باشند و زمانی که شرایط اولیه از راه حل بهینه فاصله دارند، ممکن است به خوبی کار نکنند.

مستعد بیش از حد برازش: در برخی موارد، نرخ یادگیری بسیار تطبیقی روش‌های مرتبه دوم می‌تواند منجر به تطبیق بیش از حد شود، به خصوص زمانی که مجموعه داده آموزشی کوچک یا پر سر و صدا باشد.

پایداری عددی: مسائل ناپایداری عددی می‌تواند هنگام محاسبه یا معکوس کردن ماتریس هسین، به‌ویژه زمانی که شرایط نامناسبی داشته باشد، ایجاد شود.

در عمل، روش‌های مرتبه دوم مانند آنچه ذکر شده، به دلیل هزینه محاسباتی و نیاز به حافظه، کمتر برای آموزش شبکه‌های عصبی، به ویژه شبکه‌های عصبی عمیق استفاده می‌شوند. روش‌های مرتبه اول مانند نزول گرادیان تصادفی (SGD) و انواع آن (به عنوان مثال، Adam، RMSprop) گزینه‌های اصلی برای اکثر وظایف یادگیری عمیق هستند، زیرا از نظر محاسباتی کارآمد هستند و اغلب نتایج رضایت‌بخشی را ارائه می‌دهند.

با این حال، روش‌های مرتبه دوم می‌توانند در موقعیت‌های خاصی مانند مسائل در مقیاس کوچک با ماتریس‌های Hessian به خوبی شرطی‌شده یا برای تنظیم دقیق یک مدل از پیش آموزش‌دیده، ارزشمند باشند. محققان به کاوش راه‌هایی برای عملی‌تر کردن و کارآمدتر کردن روش‌های مرتبه دوم برای یادگیری عمیق ادامه می‌دهند، و انواع مختلفی مانند L-BFGS وجود دارد که بین روش‌های مرتبه اول و دوم تعادل برقرار می‌کند.

سوال 3 :

در ابتدا 4 تا تابع فعال ساز در سایت را بررسی میکنیم :

سیگموئید (لجستیک):

تابع سیگموئید مقادیر ورودی را در محدوده ای بین 0 و 1 ترسیم می کند.

این یک منحنی صاف و S شکل است و در لایه های پنهان شبکه های کم عمق و لایه های خروجی وظایف طبقه بندی باینری استفاده می شود.

مقادیر ورودی را در مقیاسی شبیه به احتمال له می کند و برای کارهایی که خروجی نیاز به نمایش احتمالات یا مقادیر بین 0 و 1 دارد، مناسب است.

از مشکل ناپدید شدن گرادیان برای ورودی های بسیار بزرگ یا کوچک رنج می برد، که می تواند سرعت تمرین در شبکه های عمیق را کاهش دهد.

مماس هایپربولیک (tanh):

تابع مماس هذلولی شبیه به سیگموئید است اما مقادیر ورودی را در محدوده ای بین -1 و 1 ترسیم می کند.

مانند سیگموئید، منحنی صاف است و در لایه های پنهان شبکه های عصبی استفاده می شود.

این مزیت این است که مرکز صفر است که به کاهش مشکل گرادیان ناپدید شدن در مقایسه با سیگموئید کمک می کند.

برای کارهایی مناسب است که خروجی باید مقادیری را در مقیاسی نشان دهد که می تواند مثبت یا منفی باشد.

:ReLU

Rectified Linear Unit یک تابع خطی تکه تکه است که ورودی را برای مقادیر مثبت و صفر را برای مقادیر منفی برمی گرداند.

ReLU به دلیل سادگی و کارایی به یکی از پرکاربردترین توابع فعال سازی در یادگیری عمیق تبدیل شده است.

این به کاهش مشکل گرادیان ناپدید شدن کمک می کند و همگرایی را در طول تمرین تسریع می کند.

با این حال، زمانی که نورون‌ها در حین آموزش در حالت غیرفعال گیر می‌کنند، می‌تواند از مشکل "ReLU در حال مرگ" رنج ببرد، که می‌توان با استفاده از انواعی مانند Leaky ReLU یا Parametric ReLU به آن پرداخت.

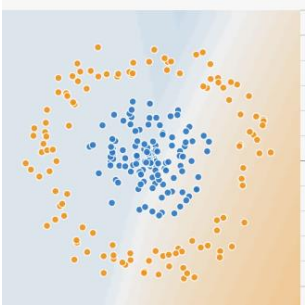
خطی :

تابع فعال سازی خطی به سادگی ورودی را همانطور که هست برمی گرداند و آن را به یک خط مستقیم با شیب 1 تبدیل می کند.

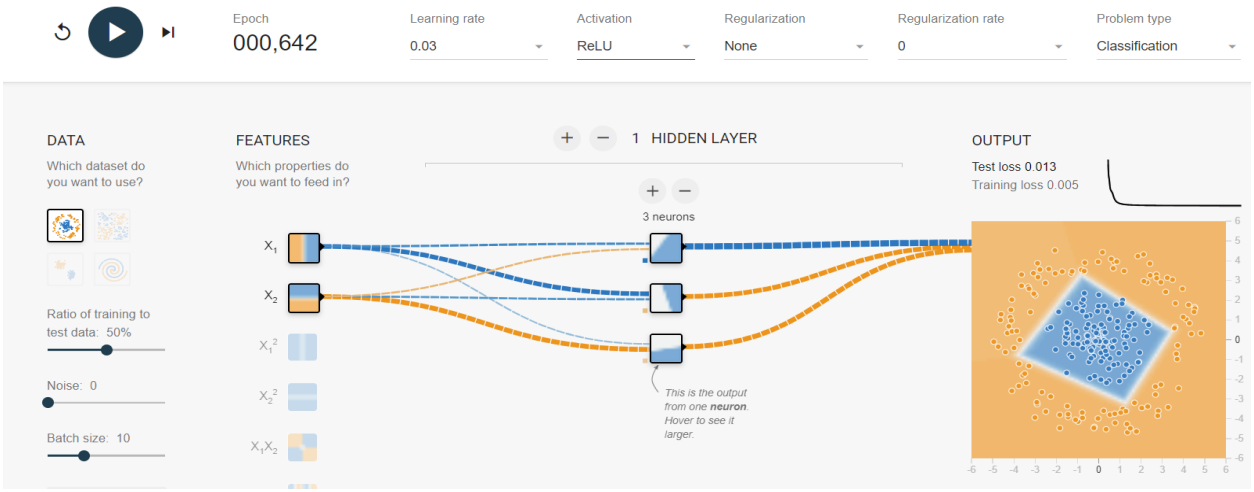
معمولاً در لایه خروجی وظایف رگرسیون استفاده می شود که در آن شبکه نیاز به پیش بینی مقادیر پیوسته دارد.

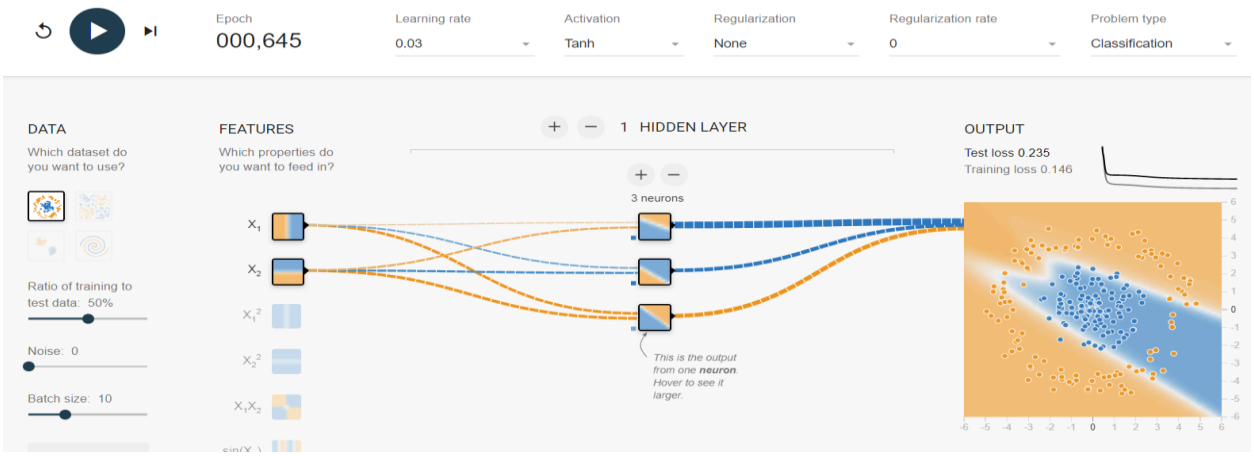
برخلاف سایر توابع فعال‌سازی، غیرخطی بودن را معرفی نمی‌کند، بنابراین برای لایه‌های مخفی زمانی که به نگاشت‌های پیچیده نیاز است، مناسب نیست.

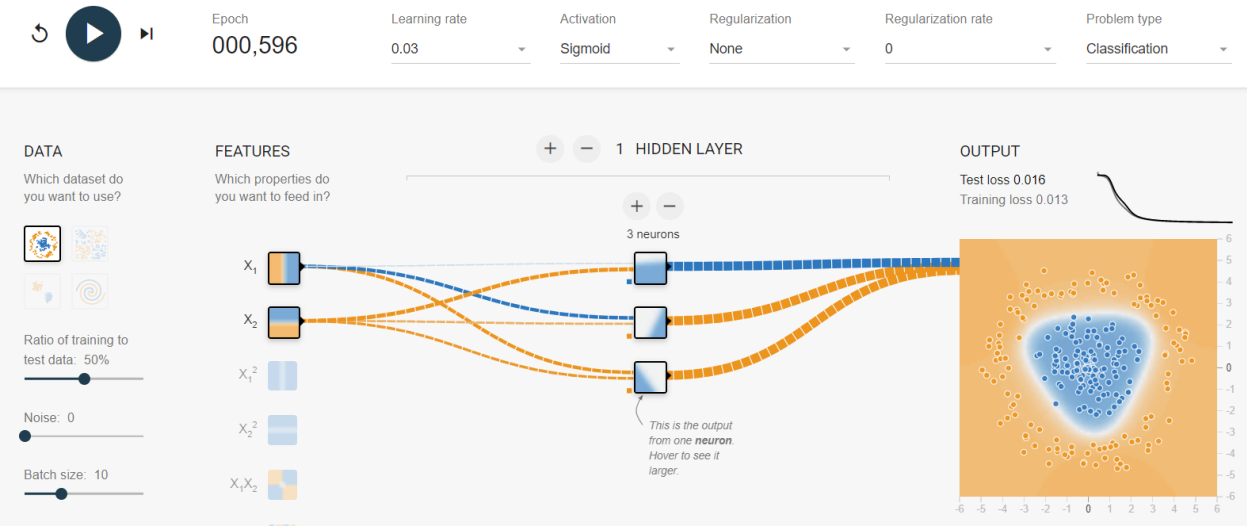
بررسی دیتاهای مختلف ، دیتای دایره ای :

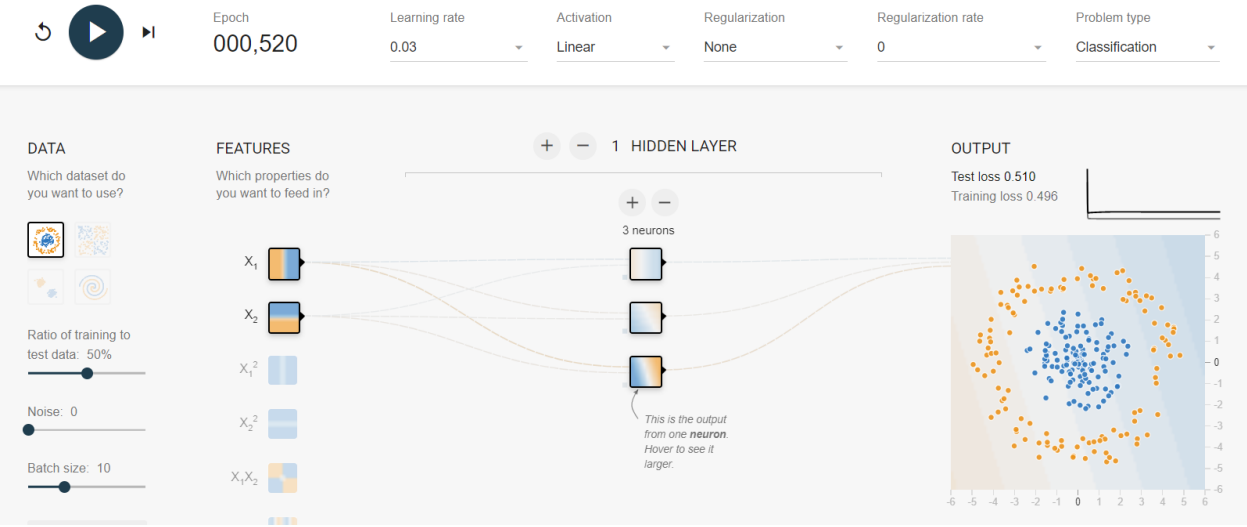


با توجه به نتایج بدست آمده از آموزش روی این صورت از داده به این نتیجه رسیدیم که تابع خطی توانایی جداسازی اینها را ندارد. چون وقتی داده‌هایی داریم که یک الگوی دایره‌ای را تشکیل می‌دهند، یک تابع فعال‌سازی خطی نمی‌تواند رابطه دایره‌ای زیربنایی بین ورودی و خروجی را ثبت کند. در بقیه حالت ها تابع relu به صورت چند ضلعی و صفر و یکی و توابع sigmoid و tanh به نتایج جداسازی خوبی رسیده اند.

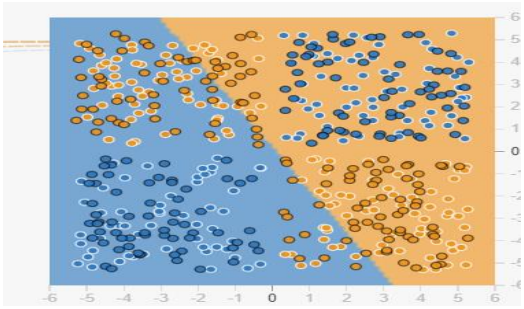




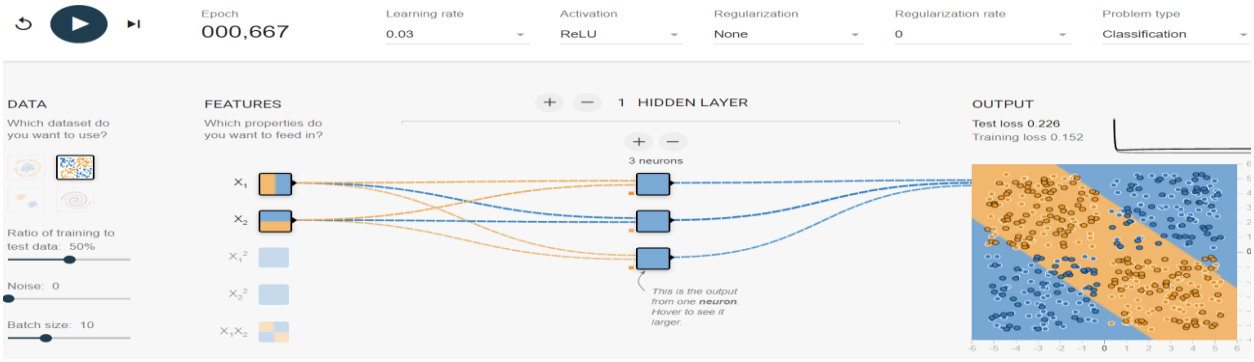


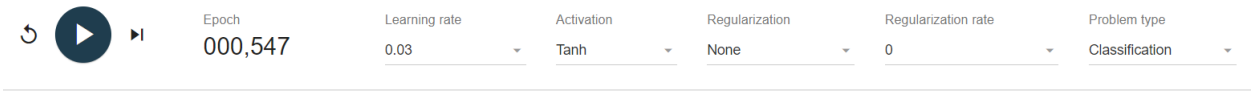


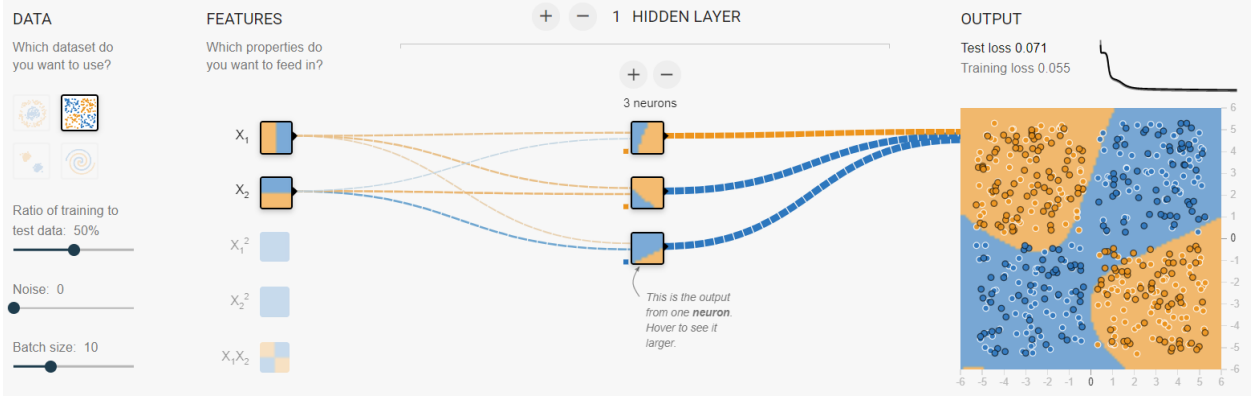
دیتای دوم ، xor :

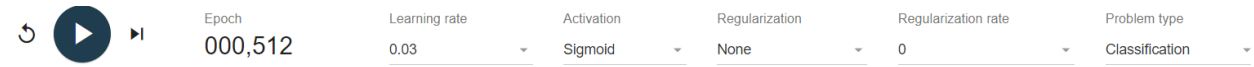


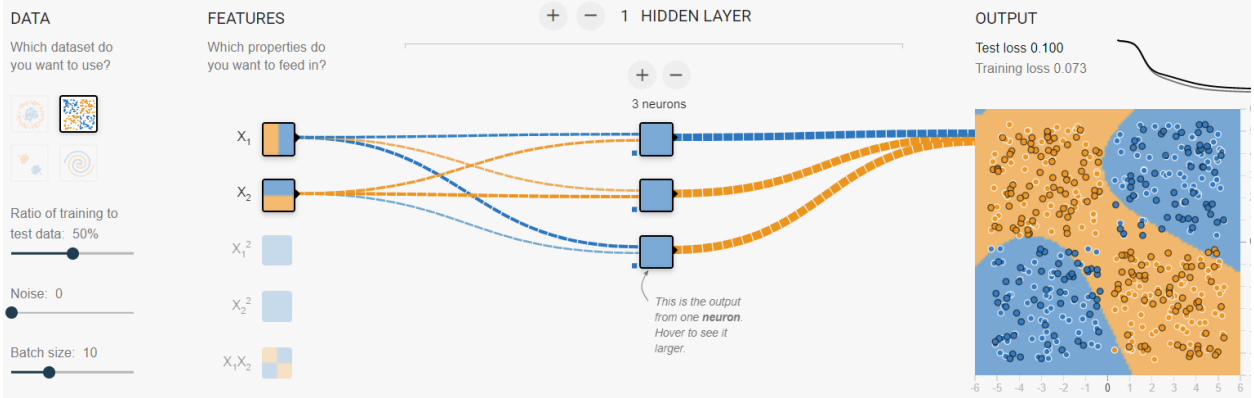
در اینجا باز هم فعالساز خطی نمیتواند به درستی جداسازی را انجام دهد. از طرفی تابع relu بهترین جداسازی را داشته و بعد از آن tanh و sigmoid توانسته اند جداسازی را انجام دهند.

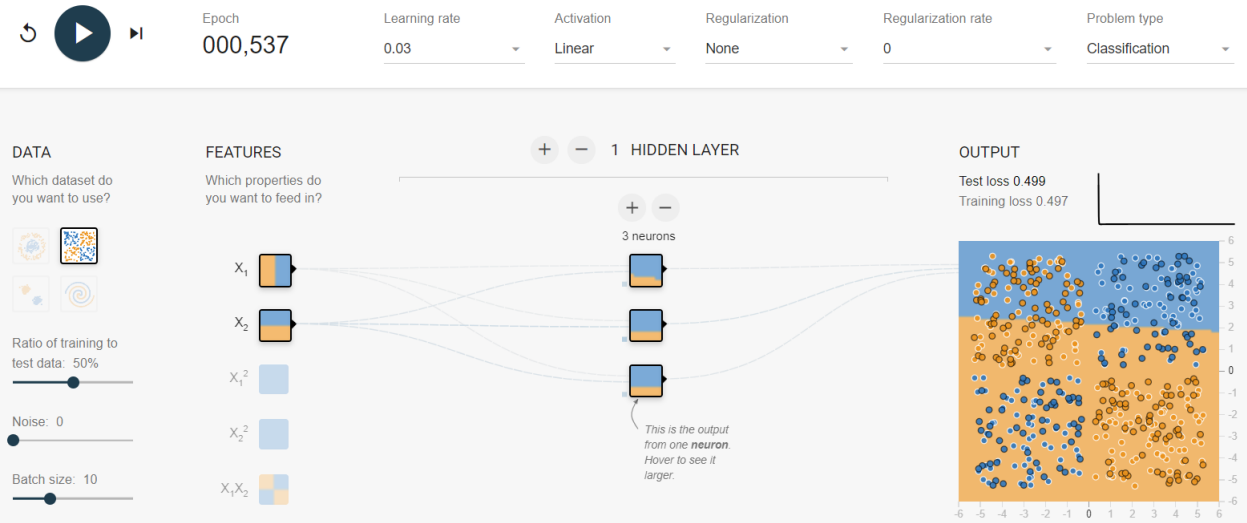

Epoch
000,667
Learning rate
0.03
Activation
ReLU
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.226
Training loss 0.152


Epoch
000,547
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification

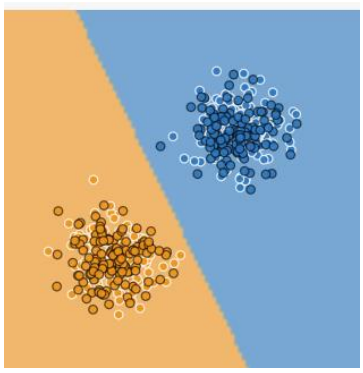

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.071
Training loss 0.055


Epoch
000,512
Learning rate
0.03
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification

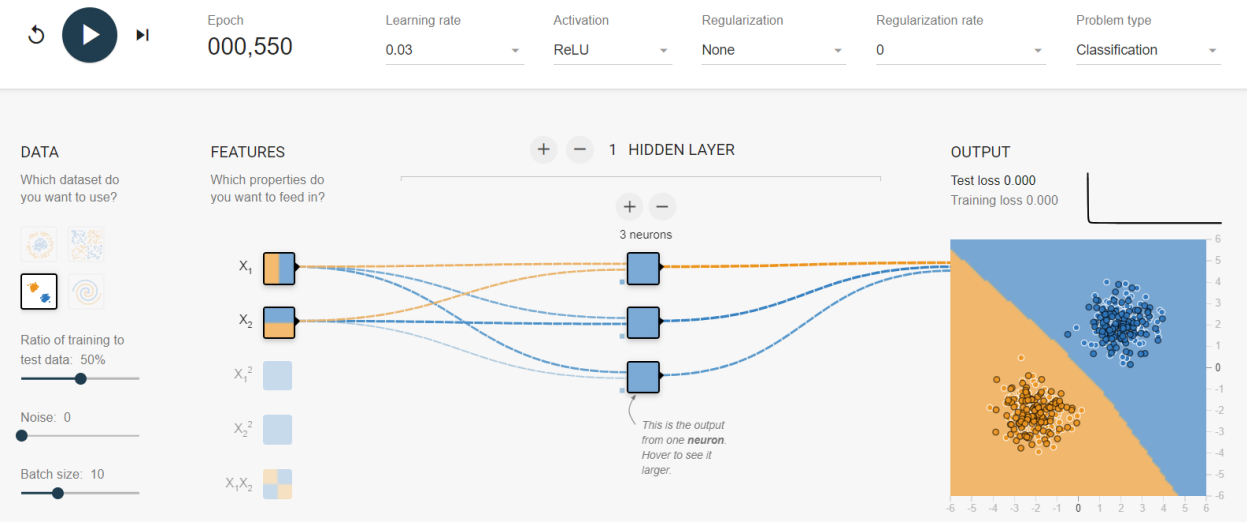

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.100
Training loss 0.073

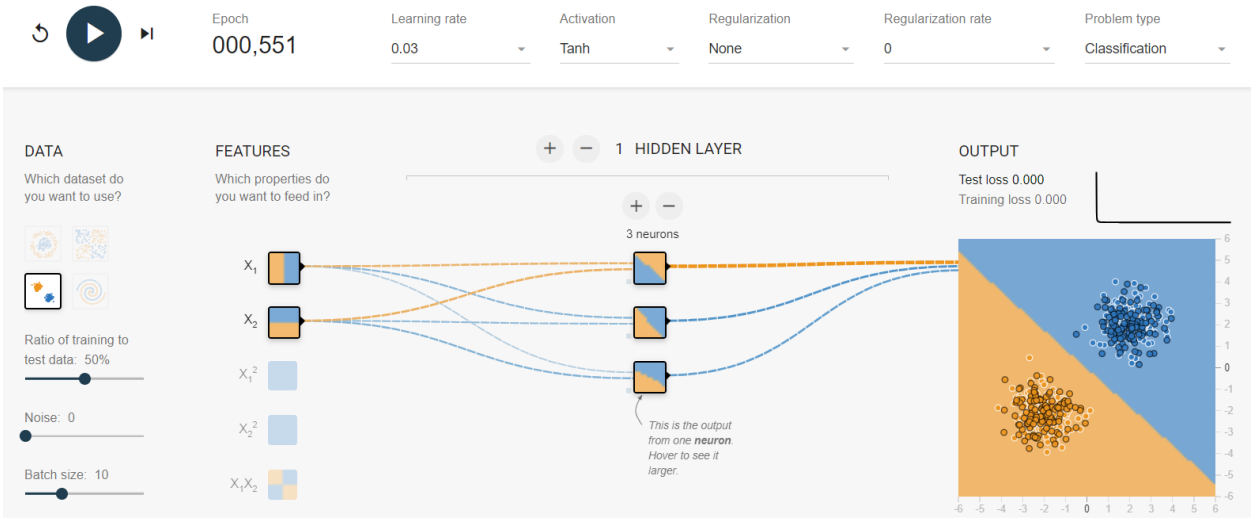


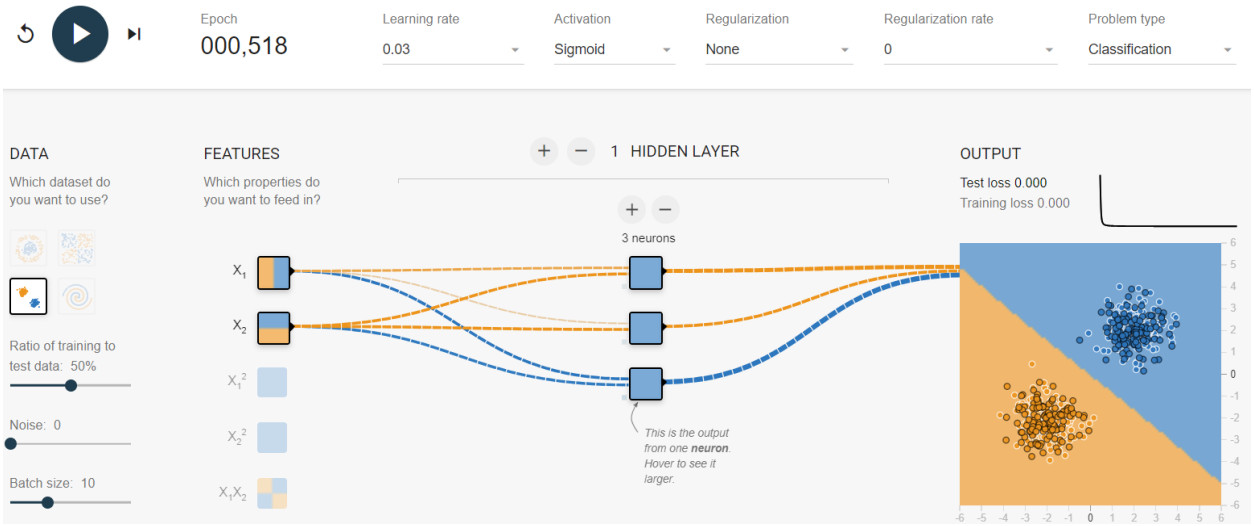
دیتای سوم ، گاوسی :

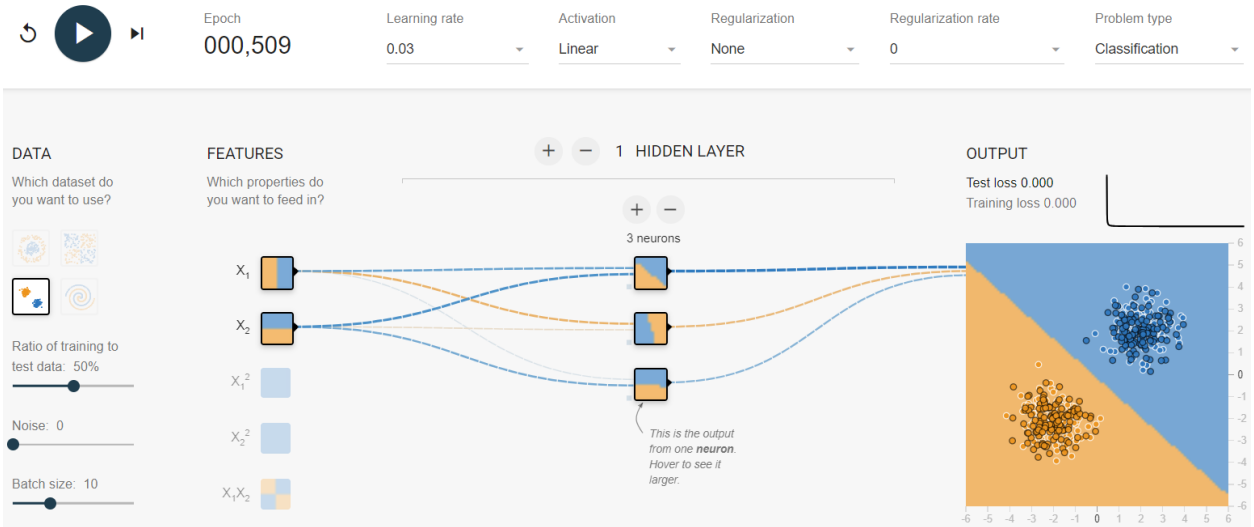


این داده به راحتی قابل جداسازی میباشد و با هر 4 تابع فعال ساز قابل جداسازی میباشد.

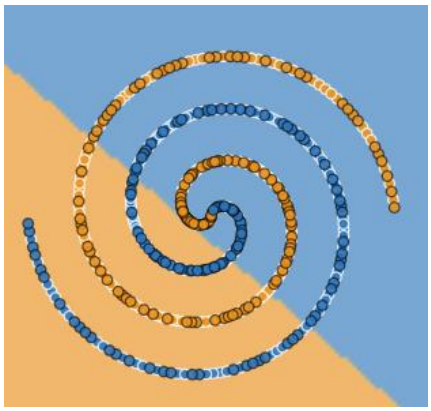



Epoch
000,551
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
X1
X2
X1²
X2²
X1X2
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.000
Training loss 0.000

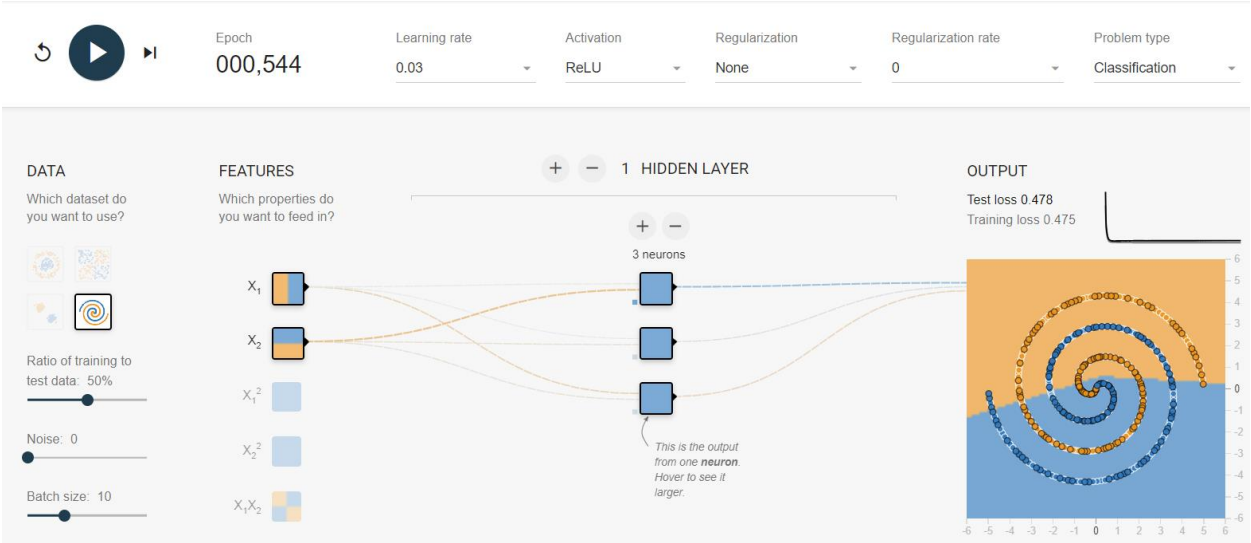

Epoch
000,518
Learning rate
0.03
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
X1
X2
X1²
X2²
X1X2
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.000
Training loss 0.000

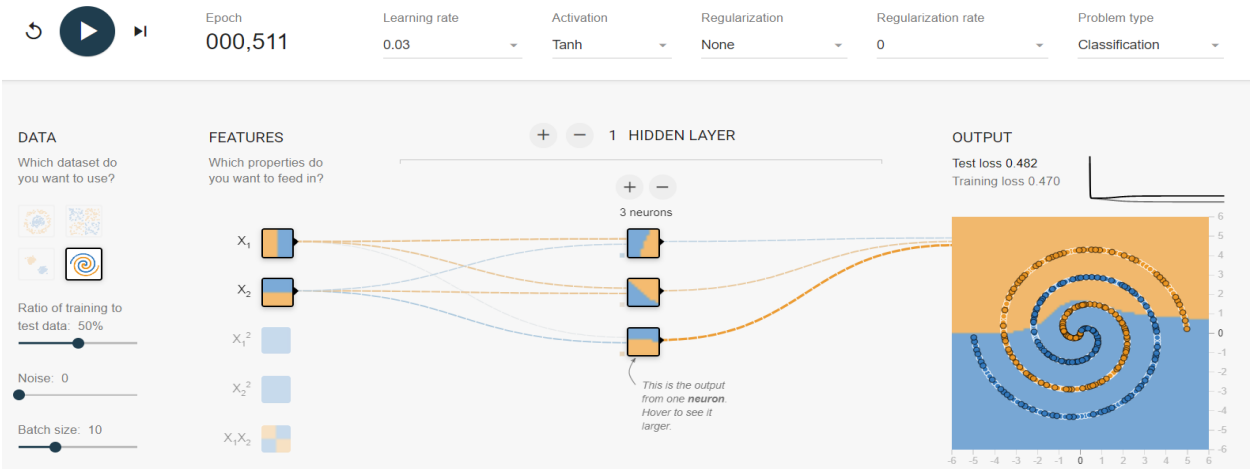

Epoch
000,509
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
X1
X2
X1²
X2²
X1X2
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.000
Training loss 0.000

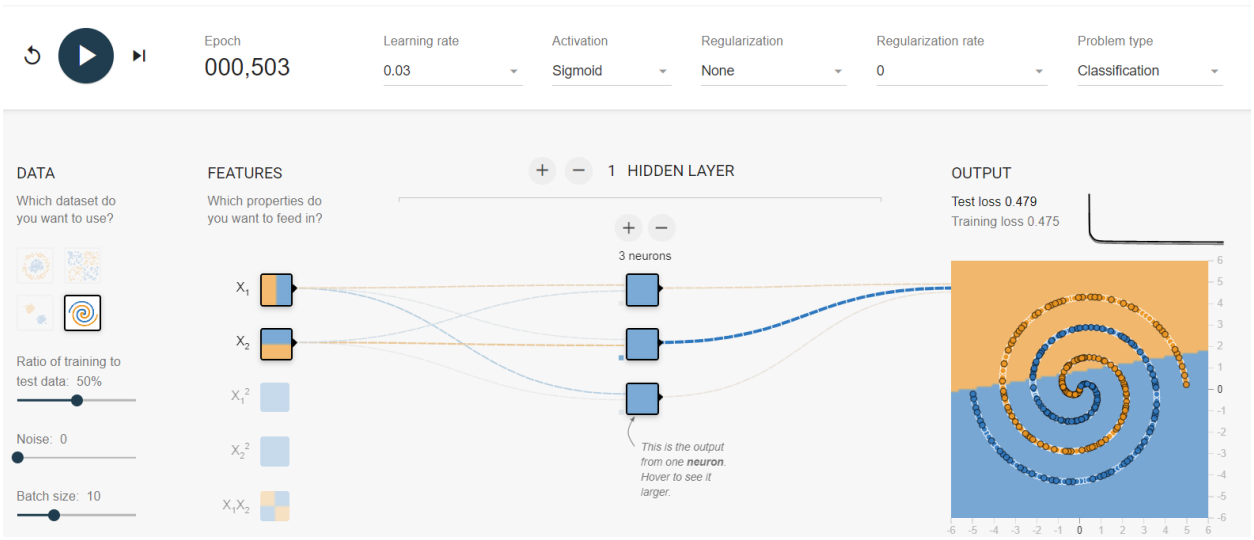
دیتای چهارم ، مارپیچی :

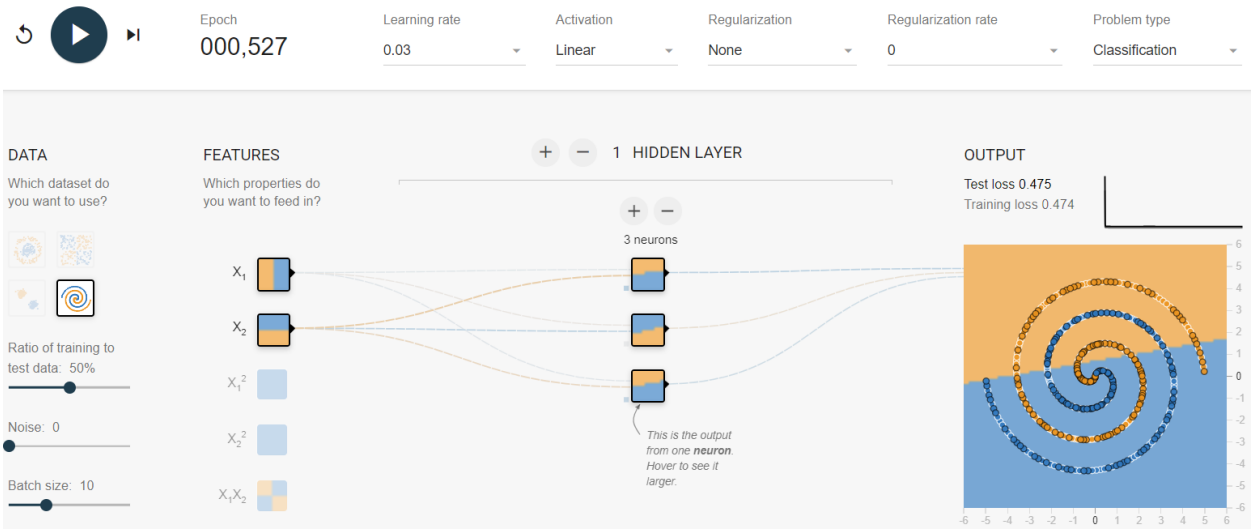


این دیتا از لحاظ جداسازی بسیار سخت میباشد و هیچکدام از 4 تابع فعالساز قادر به جداسازی نبودند و عملا با یک لایه پنهان و 3 نورون میانی این کار بسیار سخت میباشد و تمام اینها بخاطر این میباشد که مرز بسیار باریکی بین داده های ما وجود دارد.






Epoch
000,503
Learning rate
0.03
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.479
Training loss 0.475


Epoch
000,527
Learning rate
0.03
Activation
Linear
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.475
Training loss 0.474

سوال 4 :

ابتدا کدهای اضافه شده به این فایل را بررسی میکنیم :

```
model = Sequential([Dense(10, input_dim=25, name='layer'),Activation('softmax',
name='softmax_active')], name='hoda_model')
```

در اصل، این کد یک شبکه عصبی feed forward ساده را با یک لایه پنهان از 10 نورون و تابع فعال‌سازی softmax در لایه خروجی تعریف می‌کند. این 'hoda_model' نام دارد و برای کارهای طبقه بندی چند کلاسه با 25 ویژگی ورودی مناسب است، جایی که هدف آن پیش بینی یکی از چندین کلاس است. فعال‌سازی softmax تضمین می‌کند که خروجی مدل احتمالات کلاس را نشان می‌دهد و آن را برای کارهایی مانند طبقه‌بندی تصویر، طبقه‌بندی متن و موارد دیگر مناسب می‌سازد.

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

هنگامی که مدل را با این تنظیمات کامپایل کردید، با استفاده از بهینه ساز و عملکرد ضرر مشخص شده برای آموزش آماده است. در طول تمرین، مدل وزن‌ها و سوگیری‌های خود را به‌روزرسانی می‌کند تا از دست دادن آنتروپی متقاطع را به حداقل برساند و هدف آن بهبود دقت آن در داده‌های آموزشی ارائه‌شده است.

پس از کامپایل، از روش model.fit برای آموزش مدل بر روی داده های آموزشی خود استفاده کردیم و از تابع بهینه ساز و ضرر مشخص شده برای به روز رسانی پارامترهای آن استفاده می کند.

حال به بررسی نتایج بدست از کد و بررسی نمودارها میپردازیم :

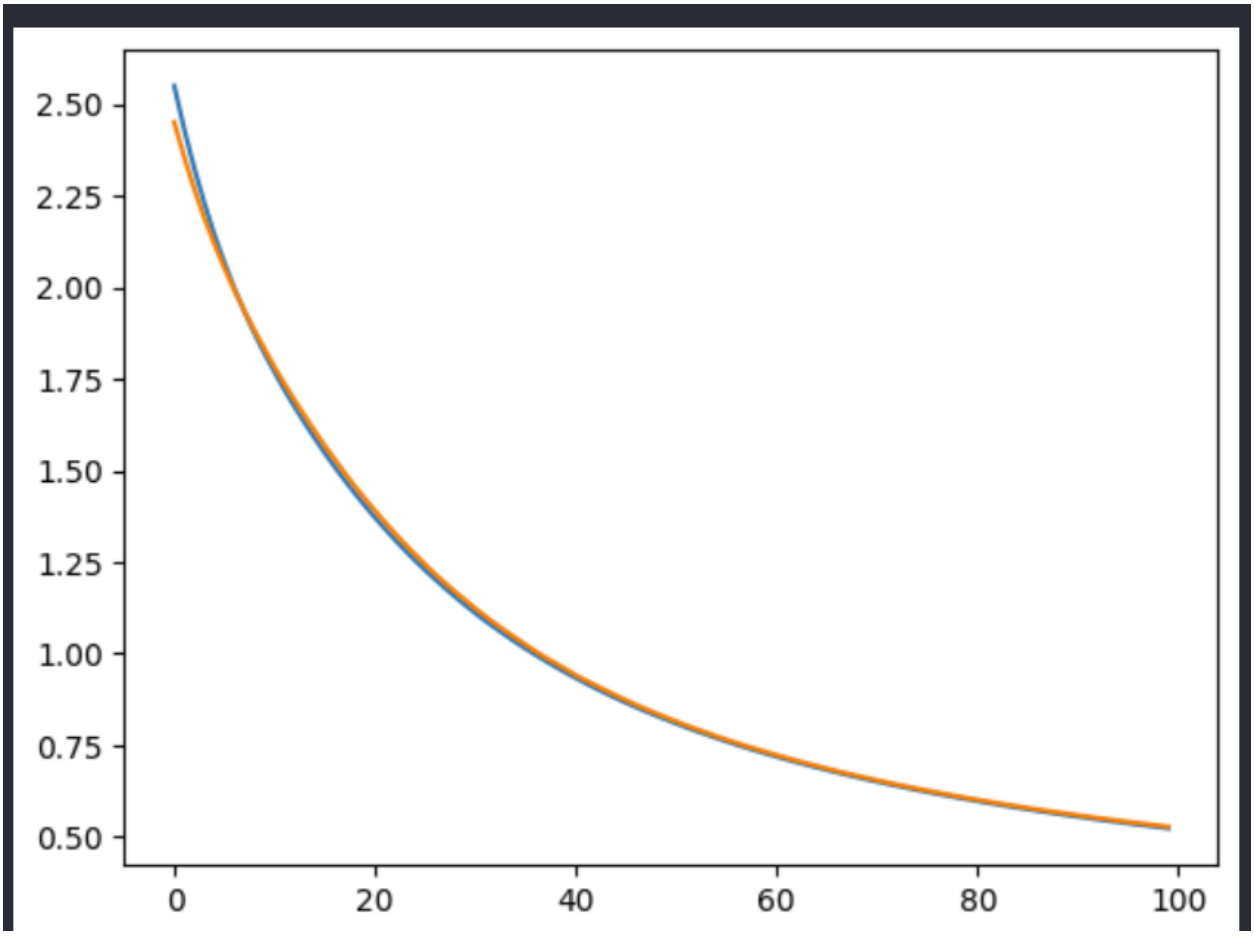
```
Model: "hoda_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer (Dense)               (None, 10)                260

 softmax_active (Activation  (None, 10)                0
 )

=================================================================
Total params: 260 (1.02 KB)
Trainable params: 260 (1.02 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

این خلاصه مدل یک نمای کلی مختصر از معماری مدل ارائه می‌کند و بررسی تعداد پارامترها و اشکال لایه‌ها را آسان می‌کند، که می‌تواند برای اشکال‌زدایی و بهینه‌سازی مفید باشد.

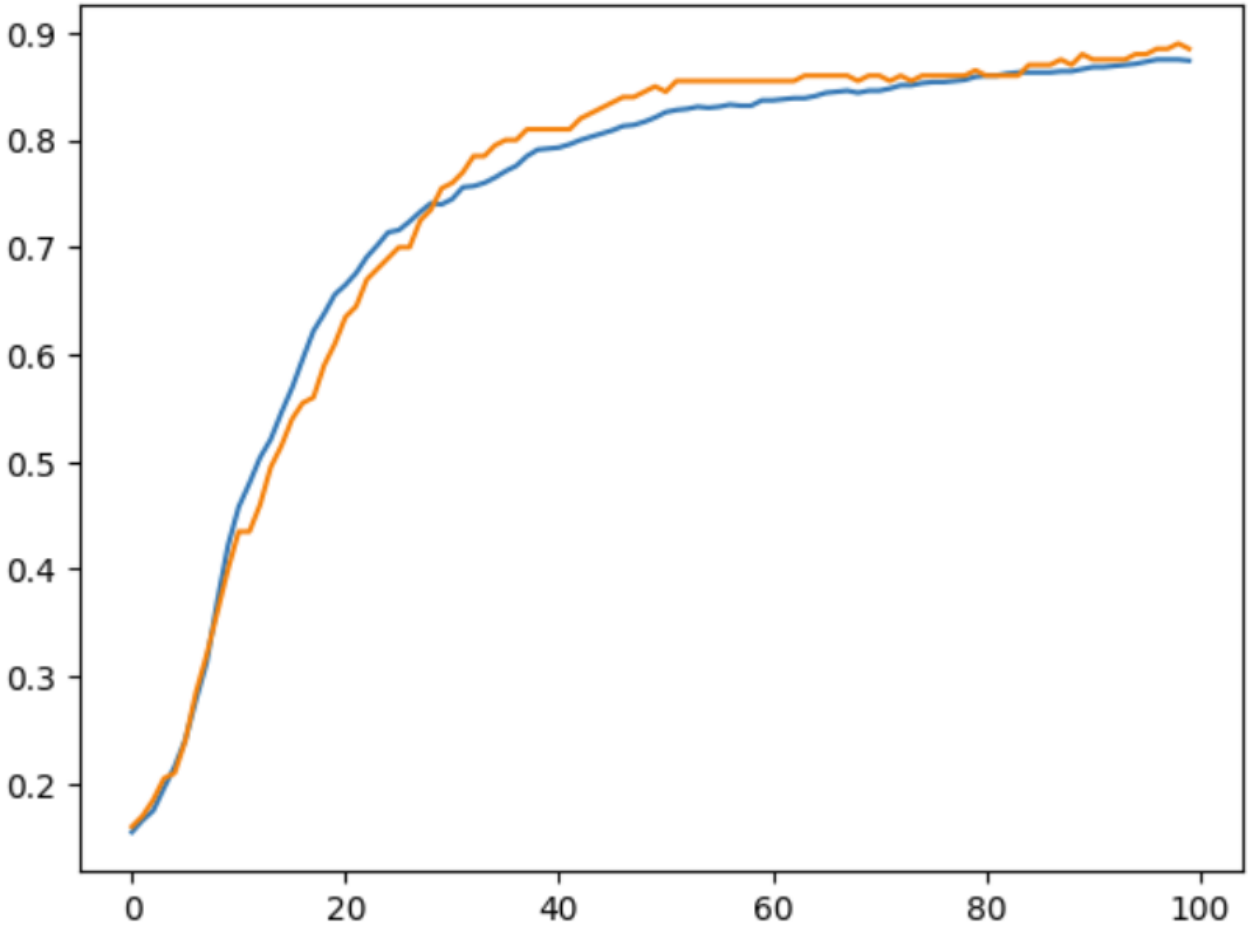
```
16/16 [==============================] - 0s 4ms/step - loss: 1.8187 - accuracy: 0.4220 - val_loss: 1.8310 - val_accuracy: 0.4000
Epoch 11/100
16/16 [==============================] - 0s 5ms/step - loss: 1.7678 - accuracy: 0.4580 - val_loss: 1.7813 - val_accuracy: 0.4350
Epoch 12/100
16/16 [==============================] - 0s 5ms/step - loss: 1.7188 - accuracy: 0.4800 - val_loss: 1.7352 - val_accuracy: 0.4350
Epoch 13/100
...
Epoch 99/100
16/16 [==============================] - 0s 4ms/step - loss: 0.5269 - accuracy: 0.8750 - val_loss: 0.5314 - val_accuracy: 0.8900
Epoch 100/100
16/16 [==============================] - 0s 4ms/step - loss: 0.5239 - accuracy: 0.8740 - val_loss: 0.5278 - val_accuracy: 0.8850
```

این نتایج به ازای هر epoch صحت نتایج بدست آمده را با جواب واقعی مقایسه کرده و accuracy را بدست میاوریم. این epoch ها در واقع نتایج تست کردن به صورت مستقیم شبکه عصبی موجود میباشد.

نمودار زیر درباره مقدار loss میباشد که همانطور که مشاهده میشود به تدریج کمتر شده است.



و نمودار بعدی درباره صحت یا همان accuracy میباشد که به تدریج افزایش یافته است.



به طور خلاصه در این کد ویژگی ها یا همان x هم در بخش train و هم در بخش test با تقسیم شدن به 255 بخش نرمالایز شده است.

و بعد از اینها همانطور که بالاتر توضیح داده شد مدل با اطلاعات خواسته شده طراحی میشود.

در مرحله بعدی با train کردن مدل به accuracy مد نظر رسیدیم.

سوال 5 :

توضیحات این فایل از کد با کامنت مشخص شده است و نتایج آن قابل دستیابی میباشد.

سوال 6 :

ما از دو لایه پنهان با 128 و 64 نورون استفاده کردیم. انتخاب تعداد نورون‌ها معمولاً بر اساس تجربه و آزمایش صورت می‌گیرد. در اینجا، از تعداد نورون‌های نسبتاً کم شروع کردیم و سپس می‌توانید آن‌ها را تنظیم کنید. از تابع فعال‌سازی ReLU برای لایه‌های پنهان استفاده کردیم، و از تابع فعال‌سازی softmax برای لایه خروجی. ما از بهینه‌ساز Adam با نرخ یادگیری 0.001 برای آموزش استفاده کردیم. مدل برای 20 دوره آموزش شده و نمودارهای Loss و Accuracy رسم شده‌اند. این یک معماری ابتدایی است و می‌توان با آزمایش معماری‌ها و پارامترها، به دقت حداقل 95 درصد در دیتاست MNIST دست پیدا کرد.