به نام خدا



درس هوش محاسباتی

تمرین دوم

مدرس : دکتر مزینی

دستياران : بیان دیوان آذر - محمود کلانتری

دانشجو : سارا سادات یونسی / 98533053

فهرست

سوال 1................................................................................................................................................... صفحه 3

سوال 2.................................................................................................................................................... صفحه7

سوال 3.................................................................................................................................................. صفحه19

سوال 4.................................................................................................................................................. صفحه29

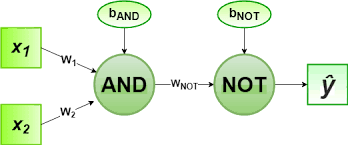
سوال 5.................................................................................................................................................. صفحه34

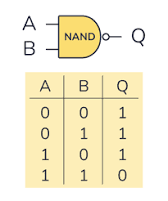
سوال 6.................................................................................................................................................. صفحه45

منابع استفاده شده...............................................................................................................................صفحه57

1- با استفاده از Neuron Adaline تابع گیت NAND را مدلسازی کنید. برای حل مسئله مقدار نرخ یادگیری(α ) را برابر با ،0.1 بایاس ها 1 و وزنهای اولیه را به صورت رندوم در نظر بگیرد و مدل را برای 4 مرحله آموزش دهید و تمام مراحل آموزش را یاداشت کنید

پاسخ 1)





Neuron Adaline

ابتدا توضیحی درباره ی این روش می پردازم :

از عملکرد فعال سازی دوقطبی استفاده می کند. نورون آدالین را می توان با استفاده از قانون دلتا یا قانون حداقل میانگین مربع (LMS) یا قانون widrow-hoff آموزش داد. ورودی خالص با مقدار هدف مقایسه می شود تا سیگنال خطا محاسبه شود. بر اساس الگوریتم تمرین تطبیقی وزن ها تنظیم می شوند ساختار اصلی آدالین شبیه پرسپترون است که دارای یک حلقه بازخورد اضافی است که با کمک آن خروجی واقعی با خروجی مورد نظر/هدف مقایسه می شود. پس از مقایسه بر اساس الگوریتم تمرین، وزن ها و سوگیری ها به روز می شوند

مرحله 0: وزن‌ها را مقداردهی کنید و بایاس‌ها روی مقادیر تصادفی تنظیم می‌شوند اما روی صفر نیستند، همچنین نرخ یادگیری α را مقداردهی کنید

مرحله 1 : مراحل 2-7 را زمانی که شرط توقف نادرست است انجام دهید

. مرحله 2 : مراحل 3-5 را برای هر جفت تمرین دوقطبی s:t انجام دهید.

مرحله 3 : هر واحد ورودی را به صورت زیر فعال کنید .

xi=si(i=1ton)

مرحله 4 : وروردی خالص را بدست آورید

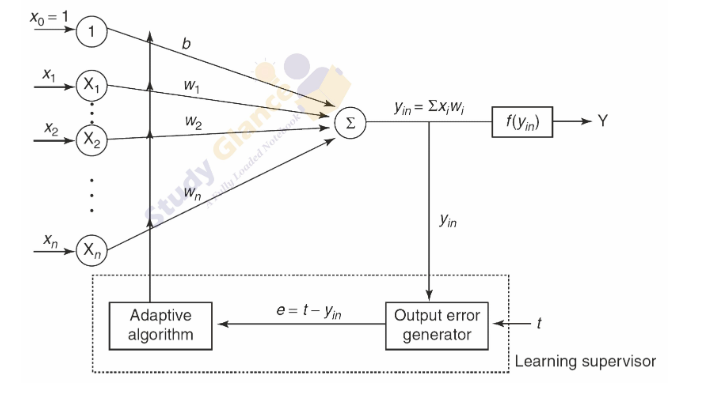
yin=∑inxi.wi+b

مرحله 5 : تا زمانی که حداقل میانگین مربع (t - yin) به دست آید، وزن و بایاس را به صورت زیر تنظیم کنید.

wi(new) = wi(old) + α(t - yin)xi  
      b(new) = b(old) + α(t - yin)

**E = (t - yin)2**

مرحله 7 - شرایط توقف را آزمایش کنید، اگر خطای ایجاد شده کمتر یا برابر با تلورانس مشخص شده باشد، متوقف شوید.



کد اجرا شده و توضیح آن :

از مشتق دوم تابع خطا نسبت به وزن، نرخ یادگیری را برای هر وزن به طور جداگانه تنظیم می کند. این کد شامل مراحل زیر است:

• وارد کردن کتابخانه های numpy و matplotlib.pyplot برای کار با آرایه ها و نمایش نمودارها

• تعریف جفت های ورودی-خروجی دروازه NAND در دو آرایه x و y

• تعریف معماری شبکه عصبی Adaline که شامل اندازه لایه ورودی و خروجی، وزن ها و بایاس است

• تعریف تابع فعال سازی که در این مثال تابع هویساید است که 1 را برای ورودی های مثبت و 0 را برای ورودی های منفی برمی گرداند

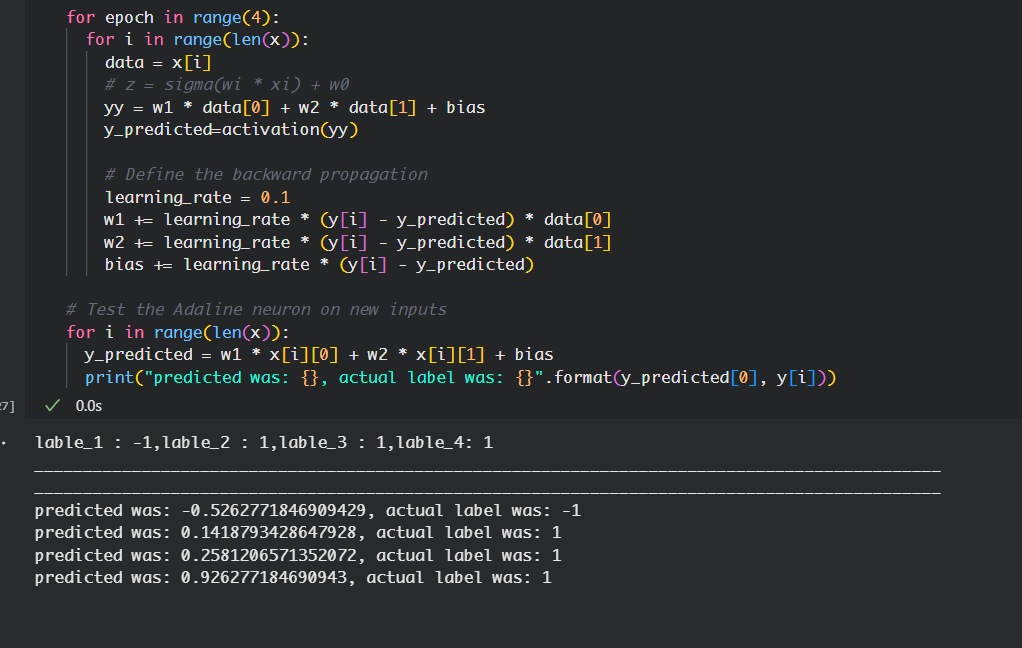
• تعریف پیش خور (forward propagation) که شامل محاسبه خروجی شبکه برای هر ورودی با استفاده از تابع جمع وزن دار و اعمال تابع فعال سازی است

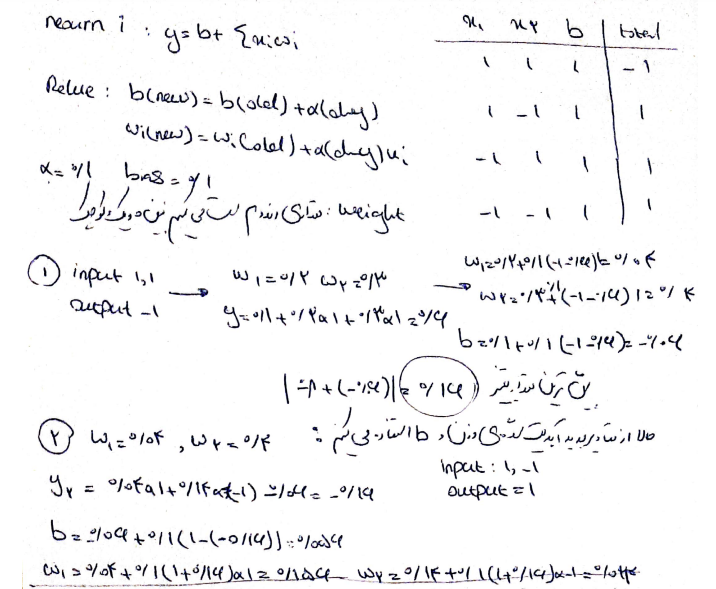
• تعریف پس خور (backward propagation) که شامل به روز رسانی وزن ها و بایاس با استفاده از قاعده یادگیری Adaline است

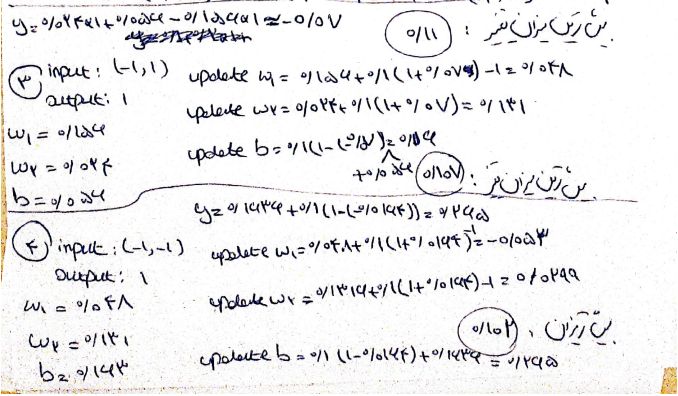
• آموزش شبکه عصبی Adaline در 4 دوره (epoch) با استفاده از حلقه های for

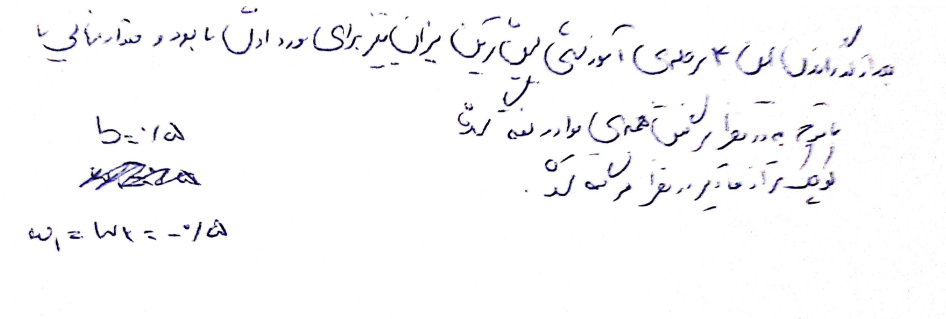


چون تابع ما nand است ابتدا یک دیتاست مطابق با لاجیک مان درست می کنیم سپس فقط 1و 1 است که به ما صفر می دهد و با بقیه متفاوت است حالا به ان ها لیبل می دهیم و وزن های رندوم و بایاس را اضافه می کنیم و نرخ یادگیری را نیز .1 می کذاریم همچنین یک تابع فعالسازی هم پیاده سازی کردیم بعد از طی کردن مراحل پس خور و پیش خور و 4 مرحله یادگیری مشاهده می کنیم که اولین لیبل موزد نظر ما مقداری منفی و بقیه آن ها مقداری مثبت است.









2-به سوالات زیر پاسخ دهید .

الف) تفاوت تابع فعال سازی خطی و غیر خطی چیست؟

پاسخ الف) تابع فعال سازی یک تابع ریاضی است که در هر نورون یک شبکه عصبی اعمال می شود و خروجی آن را به لایه بعدی منتقل می کند. تابع فعال سازی می تواند خطی یا غیرخطی باشد.

تابع فعال سازی خطی یک تابع است که خروجی آن به صورت مستقیم به ورودی آن بستگی دارد. به عبارت دیگر، تابع فعال سازی خطی یک تابع است که نمودار آن یک خط راست است. مثلاً، تابع هویت (identity function) که f(x) = x را برمی گرداند، یک تابع فعال سازی خطی است.

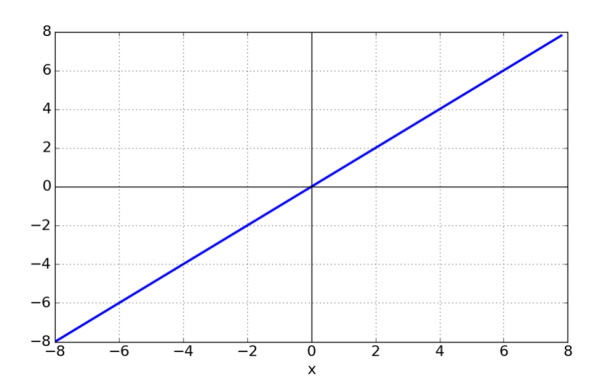
تابع فعال سازی غیرخطی یک تابع است که خروجی آن به صورت غیرمستقیم و پیچیدهتر به ورودی آن بستگی دارد. به عبارت دیگر، تابع فعال سازی غیرخطی یک تابع است که نمودار آن چندین خم و پلک دارد. مثلاً، تابع سیگموید (sigmoid function) که f(x) = 1 / (1 + e^-x) را برمی گرداند، یک تابع فعال سازی غیرخطی است.

تفاوت اصلی بین تابع فعال سازی خطی و غیرخطی در این است که تابع فعال سازی خطی قادر به انجام عملهای پایهای و ساده است، اما تابع فعال سازی غیرخطی قادر به انجام عملهای پژوهشگرایانه و پوششدهندهتر است. به عبارت ديگر، شبکه های عصبي با استفاده از توابع فعالسازي غيرخطي ميتوانند الگوهای پيچيدهتري را ياد بگيرند و پيشبيني قابلقبولي را در خروجي ارائه كنند.

تابع فعالسازی خطی

به تابع فعالسازی خطی، «تابع همانی» (Identify Function) نیز گفته می‌شود. این تابع، بر روی مجموع وز‌ن‌دار ورودی، محاسبات انجام نمی‌دهد و این مقدار را بدون هیچ تغییری به لایه بعد منتقل می‌کند.

خروجی این نوع تابع فعالسازی در شبکه های عصبی، به صورت خطی است و نمی‌توان مقدار آن را در بازه‌ای مشخص تعیین کرد. بنابراین از این نوع تابع فقط در مدل‌های رگرسیون خطی استفاده می‌شود.



می‌توان تابع ریاضی این فعالساز را به صورت زیر نوشت:

f(x)=x

تابع فعالسازی خطی، محدودیت‌ها و مشکلات اساسی دارد که در زیر فهرست شده‌اند:

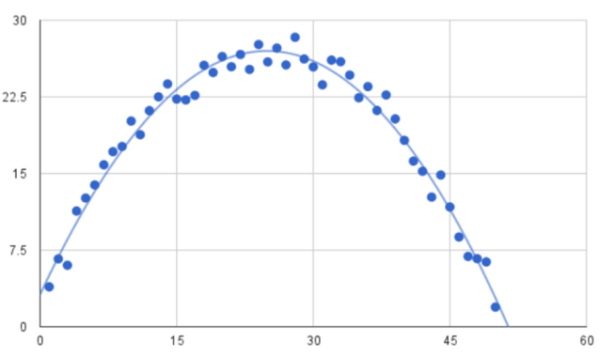
از این تابع نمی‌توان در مرحله انتشار به عقب استفاده کرد زیرا مشتق این تابع برابر با یک عدد ثابت است و هیچ رابطه‌ای با مقدار ورودی x ندارد.

نتیجه چندین تابع خطی برروی یک مقدار ثابت ورودی، یکسان است. بنابراین، اهمیتی ندارد شبکه عصبی از چندین لایه پنهان ساخته شده باشد؛ زیرا خروجی تابع فعالسازی در لایه آخر شبکه عصبی برابر با خروجی تابع فعالسازی در لایه اول است.

زمانی که از پارامترهای زیاد و پیچیده در شبکه عصبی استفاده شده باشد، این نوع تابع فعالسازی عملکرد خوبی نخواهد داشت.

تابع فعالسازی غیرخطی

این نوع توابع بیشترین استفاده را در شبکه های عصبی دارند. تعمیم‌پذیری و تطبیق‌پذیری مدل با انواع مختلف داده با استفاده از توابع فعالسازی غیرخطی آسان می‌شود.



استفاده از توابع فعالسازی غیرخطی در شبکه های عصبی گام اضافه‌تری را به محاسبات هر لایه در مرحله انتشار پیش‌خور اضافه می‌کند؛ اما این گام اضافه‌تر مزیت مهمی دارد. اگر یک شبکه عصبی بدون توابع فعالسازی وجود داشته باشد، هر گره صرفاً برروی ورودی‌های خود با استفاده از مقادیر وزن‌ها و بایاس محاسبات خطی انجام می‌دهد. در این حالت، تعداد لایه‌های پنهان در شبکه عصبی اهمیت ندارند و تمامی لایه‌ها عملکردی مشابه دارند زیرا حاصل ترکیب دو تابع خطی، یک تابع خطی خواهد بود.

با این که در این حالت، شبکه عصبی ساده‌تر است، اما شبکه قادر به انجام وظیفه‌های پیچیده‌تر نخواهد بود و کاربرد شبکه عصبی فقط به مسائل رگرسیون خطی محدود می‌شود.

توابع فعالسازی غیرخطی محدودیت‌های تابع‌های فعالسازی خطی را رفع کرده‌اند. ویژگی‌های اصلی این توابع فعالسازی عبارت‌اند از:

این توابع مشکل مربوط به مرحله انتشار به عقب را حل کردند. به عبارتی، توابع غیرخطی در مرحله انتشار رو به عقب می‌توانند مشخص کنند کدام وزن‌ گره ورودی به تشخیص نهایی مدل می‌تواند کمک بهتری کند.

با استفاده از این توابع می‌توان مسائل مربوط به خروجی‌های چندگانه را حل کرد. به عبارتی، خروجی توابع غیرخطی، ترکیب غیرخطی از ورودی‌ها است که از لایه‌های متعدد عبور کرده است.

وابع فعالسازی غیرخطی در شبکه های عصبی را می‌توان به چندین نوع تقسیم کرد که در ادامه فهرست شده‌اند:

تابع فعالسازی سیگموئید (Sigmoid | Logistic)

تابع فعالسازی تابع تانژانت هذلولوی (Tanh | Hyperbolic Tangent)

تابع فعالسازی یکسوساز (RELUs)

تابع فعالسازی یکسوساز رخنه‌دار (Leaky RELUs)

تابع فعالسازی یکسوساز پارامتریک (Parametric RELUs)

تابع فعالسازی واحدهای خطی نمایی (Exponential Linear Units | ELUss)

تابع فعالسازی Softmax

تابع فعالسازی Swish

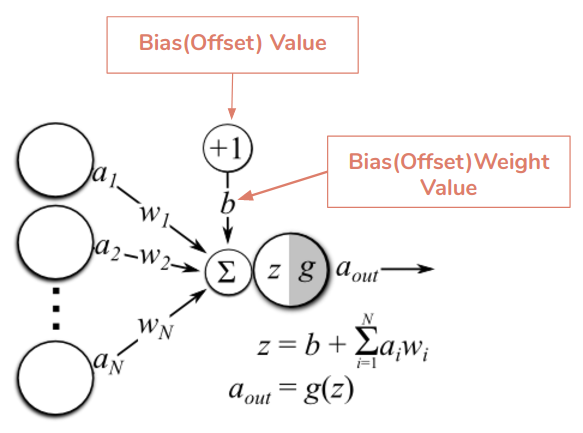
تابع فعالسازی واحد خطی خطای گاوسی (Gaussian Error Linear Unit | GELUs)

تابع فعالسازی واحد خطی مقیاس‌بندی‌شده نمایی (Scaled Exponential Linear Unit | SELUs)

ب) اگر در مدل MLP مقادیر اولیه وزن ها و بایوس به حاالت زیر در نظر بگیریم ، توضیح دهید آموزش مدل به چه صورت پیش می رود؟

• بایاس رندوم و وزن ها صفر

پاسخ )



باید توجه داشت که این حالت چندین مشکل را به همراه دارد. اول اینکه، اگر وزن ها صفر باشند، خروجی هر نورون صفر خواهد بود و تابع فعال سازی هم تاثیر نخواهد داشت. دوم اینکه، اگر وزن ها صفر باشند، گرادیان خطا نسبت به وزن ها صفر خواهد بود و در نتیجه بهینه سازی نخواهد کار کرد. سوم اینکه، اگر بایاس ها random باشند، خطای پیش بینی شده با خطای واقعی تفاوت زیاد دارد و در نتیجه آموزش طولانی تر خواهد شد.

بنابراین، آموزش مدل MLP با مقادیر اولیه داده شده به صورت زیر پیش می رود:

• در forward propagation، خروجی هر نورون صفر است و فقط بایاس ها تاثیر دارند.

• در backward propagation، گرادیان خطا نسبت به وزن ها صفر است و فقط بایاس ها به روز می شوند.

• در نتیجه، آموزش نامناسب است.

در نهایت مدل نمی تواند آموزش ببیندو هر آنچه در اینپوت های ما بوده است جمع می شود در نتیجه خروجی هر نورون ما صفر می شود و با ابایاس ها جمع می شود.

اگر در مدل MLP مقدار اولیه وزن ها صفر باشد و مقدار بایاس رندوم باشد، در ابتدا خروجی هر لایه صفر خواهد بود و همین باعث می شود که خطای بالایی برای تمام داده های آموزشی محاسبه شود. در نتیجه، در هر دور از آموزش، تمام وزن ها به یکسان به روز رسانی می شوند و مدل به یک حالت خاص محدود می شود که نمی تواند به صورت درستی آموزش داده شود. بنابراین، بهتر است وزن ها و بایاس با مقادیر کوچک و رندوم شروع شوند تا مدل بتواند به صورت مناسب آموزش داده شود.

*Initializing all the weights with zeros leads the neurons to learn the same features during training.*

In fact, any constant initialization scheme will perform very poorly. Consider a *neural network* with two hidden units, and assume we initialize all the biases to 0 and the weights with some constant *α*. If we forward propagate an input in this network, the output of both hidden units will be *relu*(*αx*1​+*αx*2​). Thus, both hidden units will have identical influence on the cost, which will lead to identical gradients. Thus, both neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things.

گر همه وزن ها را با صفر مقداردهی کنید، هر hidden neurons مستقل از ورودی صفر می شود. بنابراین، وقتی همه نورون‌های پنهان با وزن‌های صفر شروع می‌شوند، همه آنها از شیب یکسانی پیروی می‌کنند و به همین دلیل «فقط بر مقیاس بردار وزن تأثیر می‌گذارد، نه جهت.

پاسخ)

• بایاس صفر و وزن ها رندوم

باید توجه داشت که این حالت چندین نکته را در نظر بگیرد. اول اینکه، اگر بایاس ها صفر باشند، خروجی هر نورون فقط به وزن ها بستگی دارد و تابع فعال سازی کمتر تاثیر دارد. دوم اینکه، اگر وزن ها random باشند، خطای پیش بینی شده با خطای واقعی تفاوت زیاد دارد و در نتیجه آموزش طولانی تر خواهد شد. سوم اینکه، اگر وزن ها random باشند، ممکن است به نقطه چالش برانگیر (saddle point) یا حداقل محلی (local minimum) منجر شوند که باعث کاهش کارایی الگوریتم بهینه سازی می شود

بنابراین، آموزش مدل MLP با مقادیر اولیه داده شده به صورت زیر پیش می رود:

• در forward propagation، خروجی هر نورون به صورت random است و فقط وزن ها تاثیر دارند.

• در backward propagation، گرادیان خطا نسبت به وزن ها random است و فقط وزن ها به روز می شوند.

• در نتیجه، آموزش مدل کند است

مدل می تواند آموزش ببیند اما آموزش دادن ان با مشکلاتی رو به رو می شود و کند خواهد بودچون ما وزن هایی بین صفر تا یک به هر نورون اختصاص می دهیم و با وجود نداشتن بایاس مدت زیادی طول خواهد کشید که خروجی های مورد نظر ما تولید شود و دیر همگرا می شود چون مقدار بایاس جمع نمی شود از این رو یا باید یک بایاش مقدار مناسب با ان جمع کنیم یا وزنی بیش از حد معمول به آن اختصاص بدهیم

اگر مقدار اولیه بایاس ها صفر باشد و مقدار وزن رندوم باشد، در ابتدا خروجی هر لایه صفر نخواهد بود ولی به دلیل اینکه وزن ها رندوم هستند، ممکن است خروجی هر لایه به صورت تصادفی تغییر کند. این باعث می شود که مدل به یک حالت خاص محدود نشود و بتواند به صورت مناسب آموزش داده شود. با این حال، بهتر است مقادیر اولیه وزن ها و بایاس ها با مقادیر کوچک و رندوم شروع شوند تا مدل بتواند به صورت بهینه آموزش داده شود.

f a neural network does not have a bias node in a given layer, it will not be able to produce output in the next layer that differs from 0 (on the linear scale, or the value that corresponds to the transformation of 0 when passed through the activation function) when the feature values are 0.

ج) به نظر شما قابلیت تعمیم در کدام یک از شبکه های عصبی که تاکنون شناخته اید بیشتر و در کدام یک کمتر است؟

پاسخ ج )

تفاوت اصلی بین Perceptrone و Adaline، این است که Perceptrone، response binary گرفته و سپس error را محاسبه کرده و از آن برای آپدیت کردن وزن ها، استفاده می کند؛ اما Adaline، از response continous استفاده کرده و وزن ها را آپدیت می کند. این ویژگͬ Adaline، باعث می شود آپدیت های آن قبل از تعیین شدن threshold، بیشتر شبیه ارور واقعی باشد؛ در نتیجه به مدل ما اجازه می دهد سریع تر همگرا شود و در نتیجه قابلیت generalization آن بیشتر باشد. همچنین Madaline، از AND چند Adaline به دست می آید و قابلیت حل مسائل غیر خطی را نیز دارد، بنابراین قابلیت تعمیم Madaline هم از Adaline بیشتر است؛ زیرا Adaline فقط برای مسائل تفکیک پذیر خطی قابل کاربرد هست، به علاوه تعداد لایه ها در Madaline از Adaline بیشتر می باشد. در MLP، به دلیل توانایی نگاشت غیر خطی و در نتیجه قابل استفاده بودن آن در مسائل classification پیچیده تر، قابلیت تعمیم آن، از همه بیشتر است. بنابراین اگر بخواهیم این مدل ها را براساس قابلیت تعمیم پذیری شان طبقه بندی کنیم، به شرح زیر می باشد

MLP> Madaline > Adaline > Perceptrone

تفاوت اصلی بین این دو، این است که یک Perceptron آن پاسخ دودویی (مانند یک نتیجه طبقه بندی) را میگیرد و یک خطا را محاسبه میکند که برای آپدیت وزن ها استفاده می شود، در حالی که یک Adaline از یک مقدار پاسخ پیوسته برای آپدیت وزنها استفاده می کند. پرسپترون وزن ها را با محاسبه تفاوت بین مقادیر کالس مورد انتظار و پیش بینی شده به روز می کند. به عبارت دیگر، پرسپترون هم یشه +1 یا -1 )مقادیر پی ش بینی شده( را با +1 یا -1 )مقادیر مورد انتظار( مقایسه میکند. پرسپترون تنها زمانی یاد م یگیرد که خطا ایجاد شود. در مقابل، Adaline تفاوت بین مقدار کالس مورد انتظار y ( +1 یا -1( و مقدار خروج ی پیوسته y را از تابع خط ی محاسبه میکند که میتواند هر عدد واقعی باشد. ا ین بسیار مهم است زیرا به این معنی است که Adaline میتواند حتی زمانی که ه ی چ اشتباه ی در طبقه بندی انجام نشده است یاد بگیرد. از آنجایی که Adaline هم یشه و پرسپترون فقط پس از خطاها یاد می گی رد، Adaline راه حلی سریعتر از پرسپترون برا ی همان مشکل پیدا م ی کند. این واقعیت که Adaline این کار را انجام میدهد، اجازه میدهد آپدیتهای آن، قبل از اینکه آستانه تعیین شود، بی شتر شبیه خطای واقعی باشند، که به نوبهی خود به مدل اجازه م یدهد تا سریعتر همگرا شود.پس قابلیت تعمیم پذیری Adaline از Perceptron بیشتر است. همچنین Madaline ترکیب AND چند Adaline است که بدیهیست نسبت به دو مورد قبلی تعمیمپذیری بیشتری دارد. مزیت MLP نیز نسبت به Perceptron و Adaline کالسیک این است که با اتصال نورونهای مصنوعی در این شبکه از طریق توابع فعالسازی غیرخطی ، میتوانیم مرزهای تصم یمگیری پیچیده و غیرخطی ایجاد کنیم که به ما امکان میدهد با مشکالتی که در آن کالسهای مختلف قابل جداسازی خطی نیستند، مقابله کنیم. پس قابلیت تعمیمپذیری آن از تمام موارد دیگر ذکرشده بیشتر است. )لینک ( پس به ترتیب ابتدا MLP سپس Madaline، Adaline و در آخر Perceptron است.

شبکه کوهونن شامل یک لایه ورودی و یک لایه خروجی است. لایه خروجی به صورت یک شبکه دوبعدی از نورونها است که هر کدام یک بردار وزن دارند. در طول یادگیری، شبکه کوهونن سعی میکند تا نورونهای خروجی را به نحوی مرتب کند که نمایندهی الگوهای موجود در دادههای ورودی باشند. برای این کار، شبکه کوهونن از دو مرحله رقابت و همکاری استفاده میکند. در مرحله رقابت، شبکه کوهونن برای هر داده ورودی، نورونی را که بیشترین شباهت را به آن دارد، به عنوان نورون برنده انتخاب میکند. در مرحله همکاری، شبکه کوهونن وزنهای نورون برنده و همسایههای آن را به سمت داده ورودی تغییر میدهد تا شبکه بهتر بتواند دادهها را نمایش دهد.

تعمیم پذیری Kohonen بیشتر از MLP است. الگوریتم Kohonen برای شبکه های خودسازماندهی و تحلیل مؤلفه های اصلی (PCA) استفاده می شود و معمولاً برای تحلیل داده های بدون نظارت و کاهش بعد استفاده می شود. در حالی که MLP برای یادگیری ماشین، شناسایی الگو و پیش بینی استفاده می شود. بنابراین، Kohonen دارای تعمیم پذیری بالاتری است زیرا می تواند در مسائل گوناگون از جمله تحلیل تصویر، پردازش زبان طبیعی و شناسایی الگو استفاده شود. در حالی که MLP معمولاً در مسائل پیچیده تر و با داده های بزرگتر استفاده می شود.

There is no definitive answer to whether Kohonen is more generalizable than MLP, as it depends on various factors such as the type and complexity of the problem, the size and quality of the data, the architecture and parameters of the network, and the evaluation criteria. However, some general observations can be made based on the literature:

• Kohonen has the advantage of being able to discover the underlying structure and patterns of the data without requiring any labels or prior knowledge. This can be useful for exploratory data analysis, data visualization, feature extraction, and anomaly detection. Kohonen can also adapt to changing data and learn incrementally. However, Kohonen has the disadvantage of being sensitive to the order and distribution of the data, the choice of the network size and topology, and the initialization and update of the weights. Kohonen may also suffer from overfitting or underfitting if the network is too large or too small for the

data.

<https://arxiv.org/abs/2203.12893>

• MLP has the advantage of being able to learn complex nonlinear relationships between the input and output variables and perform various tasks such as classification, regression, prediction, and approximation. MLP can also benefit from various techniques such as regularization, dropout, batch normalization, and optimization algorithms to improve its generalization performance. However, MLP has the disadvantage of requiring a large amount of labeled data and computational resources to train effectively. MLP may also suffer from overfitting or underfitting if the network is too deep or too shallow, or if the learning rate is too high or too low.

<https://link.springer.com/article/10.1007/s11277-022-10079-4>

Therefore, the generalization performance of Kohonen and MLP depends on the specific problem and data at hand, and the optimal network design and configuration. Some studies have compared the generalization ability of Kohonen and MLP on different tasks and domains, and have reported mixed results. For example:

• A study on different deep learning algorithms used in deep neural nets: MLP SOM and DBN

<https://link.springer.com/article/10.1007/s11277-022-10079-4>

found that MLP outperformed SOM and DBN on wireless network optimization problems, but SOM performed better on image classification problems.

• A study on generalization aspect of accurate machine learning models for wireless networks

<https://link.springer.com/article/10.1007/s12243-021-00853-z>

found that KNN outperformed MLP on wireless network parameter estimation problems, but MLP performed better on wireless network classification problems. Combining the predictions of KNN and MLP NN ensemble did not improve accuracy.

• A study on FAMLP: A Frequency-Aware MLP-Like Architecture For Domain Generalization

<https://arxiv.org/abs/2203.12893>

found that FAMLP, a novel frequency-aware MLP architecture, outperformed the state-of-the-art methods on domain generalization problems, which aim to learn a model that can generalize well across different domains or distributions of data.

Top of Form

قابلیت تعمیم یک شبکه عصبی به معنای توانایی آن است که بتواند بر روی داده های جدید و ناشناخته که در زمان آموزش دیده نشده اند، عملکرد خوبی داشته باشد. قابلیت تعمیم یک شبکه عصبی به چندین عامل بستگی دارد

• تعداد و کیفیت داده های آموزش

• معماری و پارامترهای شبکه عصبی

• روش آموزش و بهینه سازی شبکه عصبی

• مقدار منظم سازی (regularization) و جلوگیری از بیش برازش (overfitting)

به طور کلی، نمی توان گفت که کدام نوع شبکه عصبی قابلیت تعمیم بالاتر یا پایین تری دارد، زیرا این مسئله به مسئله خاص، داده های موجود و روش آموزش بستگی دارد. اما می توان چند نکته را در نظر گرفت:

• شبکه های عصبی با لایه های بسیار زیاد (deep neural networks) معمولاً قابلیت ژنرال سازی بالاتری نسبت به شبکه های عصبی با لایه های کمتر (shallow neural networks) دارند، زیرا می توانند بازنمایی های پیچیده تر و عمیق تر از داده ها را یاد بگیرند. البته این شرط لازم است که داده های آموزش کافی و مناسب باشند و روش منظم سازی مناسب اعمال شود.

• شبکه های عصبی با لایه های بازگشتی (recurrent neural networks) قابلیت تعمیم خوبی در پردازش داده های دارای توالی (sequence) مانند متن، صوت و ویدئو دارند، زیرا می توانند اطلاعات گذشته را حفظ کنند و به خروجی خود اضافه کنند. البته این نوع شبکه ها نسبت به مشکل ناپدید شدن گرادین (vanishing gradient) حساس هستند و نسخه های بهبود یافته آ ، مانند LSTM (Long Short-Term Memory) و GRU (Gated Recurrent Unit) پیدا شده اند.

• شبکه های عصبی با لایه های کانولوشنی (convolutional neural networks) قابلیت تعمیم خوبی در پردازش داده های دارای ساختار شبکه ای (grid) مانند تصویر و صوت دارند، زیرا می توانند ویژگی های محلی و سلسله مراتبی را از داده ها استخراج کنند. البته این نوع شبکه ها نسبت به مشکل بیش برازش (overfitting) حساس هستند و نیاز به روش های منظم سازی مانند dropout و batch normalization دارند

قابلیت تعمیم در شبکه های عصبی پرسپترون چند لایه (MLP) بیشتر است. این شبکه ها قابلیت یادگیری الگوهای پیچیده را دارند و قادر به تعمیم دادن دانش به داده های جدید هستند. همچنین، شبکه های MLP قابلیت تطبیق با مجموعه داده های بزرگ را دارند و به طور کلی به عنوان یکی از قوی ترین شبکه های عصبی برای کاربردهای گوناگون شناخته می شوند.

شبکه حافظه کوتاه مدت بلند مدت یا LSTM شاید موفق ترین شبکه RNN باشد زیرا بر مشکلات آموزش یک شبکه تکراری غلبه می کند و به نوبه خود در طیف گسترده ای از برنامه ها استفاده شده است. به طور کلی، مدل‌های ساده‌تر کمتر مستعد برازش بیش‌ازحد هستند، زیرا به دلیل سادگی، قوانین تصمیم‌گیری درشت‌تر هستند و به تعمیم‌های بیشتری نیاز دارند. البته، اگر یک مدل برای یک کار خیلی ساده باشد، ممکن است نتواند قوانین تصمیم گیری خوبی را که ظرافت های کار را در بر می گیرد، یاد بگیرد. محققان مدل‌ها را از طریق تکنیک‌های مختلف ساده‌تر می‌کنند، از جمله داشتن مدل‌هایی با پارامترهای کمتر یا تشویق پارامترهایی که مدل باید در اندازه کوچک با کاهش وزن باشد.

د) برای تعیین مقدار تغییر وزن در هر مرحله از آموزش شبکه MLP میتوان از رابطه روبرو استفاده نمود. به نظر شما مزایا و معایب این روش چیست؟

∆𝜔 = −𝐻 −1 𝜕𝐸/ 𝜕𝑤

𝐻 = 𝜕2 𝐸/ 𝜕𝑤2

پاسخ د )

روشی که شما اشاره کردید، یک نوع از روش های بهینه سازی مبتنی بر گرادیان می باشد که به آن روش نیوتن (Newton's method) یا روش نیوتن-رافسون (Newton-Raphson method) می گویند. این روش از ماتریس هسین (Hessian matrix) که دومین مشتق تابع هدف (مثلاً تابع خطا) را نسبت به پارامترها (مثلاً وزن ها) نشان می دهد، استفاده می کند. این روش مزایا و معایب زیر را دارد:

مزایا:

• این روش می تواند به سرعت به حداقل محلی (local minimum) یا حداقل جهانی (global minimum) تابع هدف برسد، زیرا می تواند از اطلاعات دومین مشتق برای تنظیم مقدار و جهت گام بهینه سازی استفاده کند. -update-formula

• این روش می تواند در مواردی که تابع هدف خمیده (convex) یا نزدیک به خمیده باشد، عملکرد خوبی داشته باشد، زیرا در این صورت حداقل جهانی وجود دارد و ماتریس هسین مثبت معین (positive definite) است. in-multilayer-perceptron

معایب:

• این روش نیاز به محاسبه ماتریس هسین و وارون آن دارد که هزینه محاسباتی زیادی دارد، به خصوص اگر تعداد پارامترها زیاد باشد.

• این روش ممکن است در مواردی که تابع هدف غیرخمیده (non-convex) باشد، عملکرد خوبی نداشته باشد، زیرا در این صورت ممکن است به نقطه چالش برانگیر (saddle point) یا حداکثر محلی (local maximum) منجر شود که ماتریس هسین منفی معین (negative definite) یا نامعین (indefinite)

روش نیوتن رافسون یا روش نیوتن روشی قدرتمند برای حل عددی معادلات است. معمولاً برای تقریب ریشه های توابع با ارزش واقعی استفاده می شود. روش نیوتن رپسون توسط آیزاک نیوتن و جوزف رافسون توسعه داده شد، از این رو روش نیوتن راپسون نامگذاری شد. روش نیوتن رافسون شامل اصلاح مکرر حدس اولیه برای همگرایی آن به سمت ریشه مورد نظر است. با این حال، این روش برای محاسبه ریشه های چند جمله ای ها یا معادلات با درجات بالاتر کارآمد نیست، اما در مورد معادلات درجه کوچک، این روش نتایج بسیار سریعی را به همراه دارد.

یکی از سریع ترین همگرایی ها به ریشه. به صورت درجه دوم روی ریشه همگرا می شود. تبدیل آسان به ابعاد مختلف می توان از آن برای "جلی بخشیدن" ریشه ای که با روش های دیگر یافت می شود استفاده کرد.

این روش میتواند به سرعت به بهینهی محلی یا جهانی همگرا شود، زیرا نرخ یادگیری را بر اساس شیب تابع خطا تغییر میدهد.

• این روش میتواند از گیر کردن در نقاط ثابت یا saddle point جلوگیری کند، زیرا نرخ یادگیری را در صورت کاهش شیب تابع خطا افزایش میدهد.

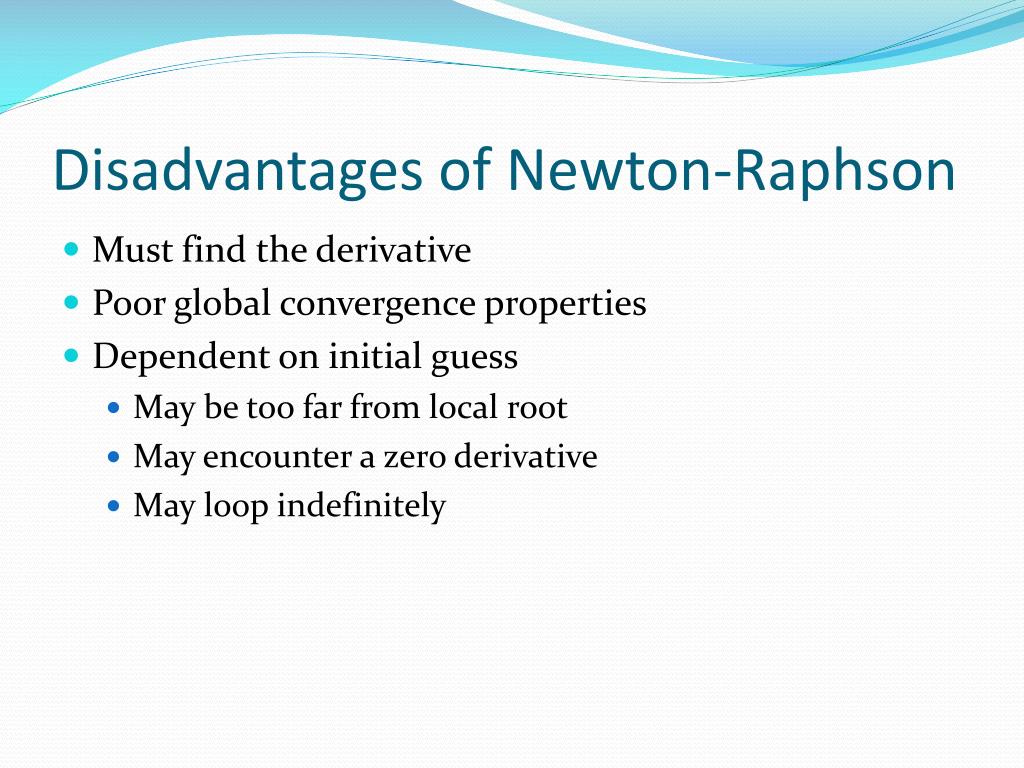
• این روش میتواند به کاهش اثر نویز یا تغییرات تصادفی در دادهها کمک کند، زیرا نرخ یادگیری را در صورت افزایش شیب تابع خطا کاهش میدهد.

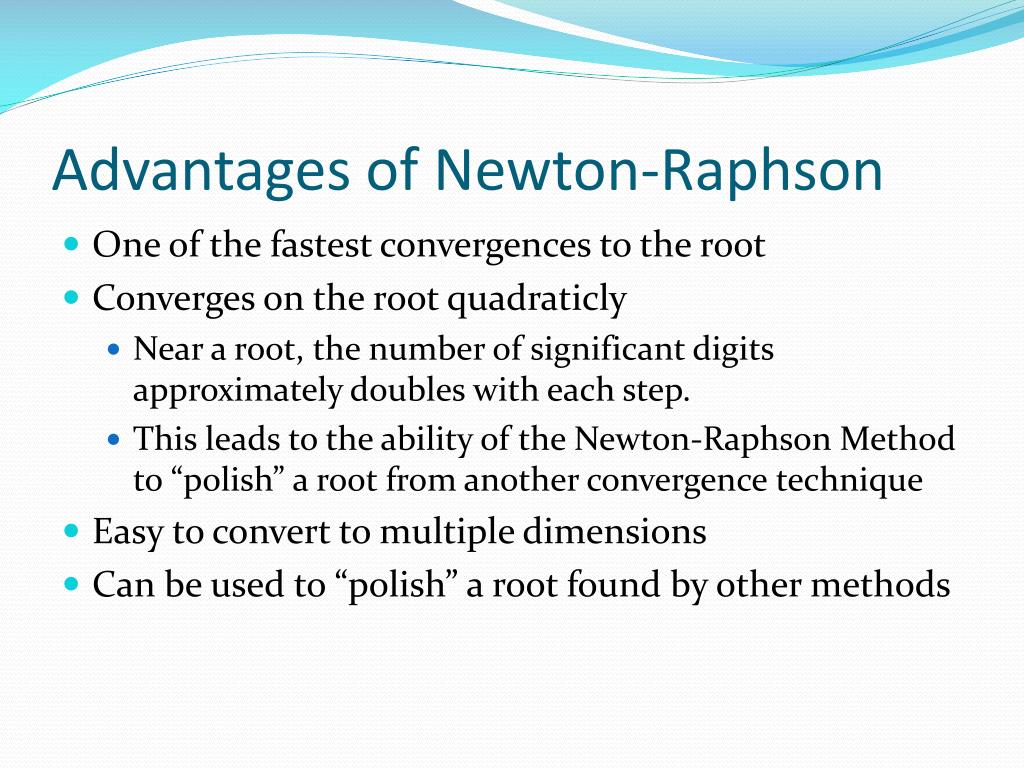
• معایب:

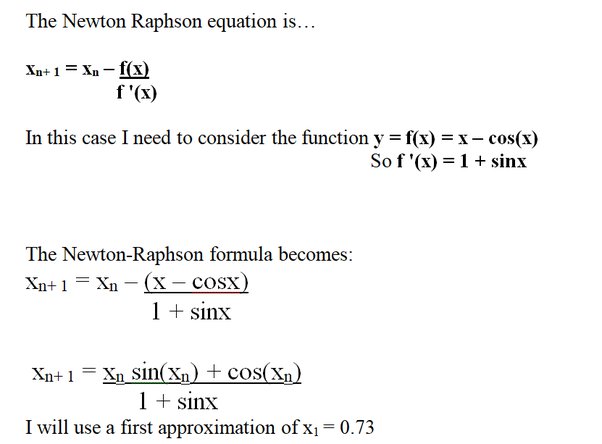
• این روش ممکن است به حداقل محلی نزدیک شود و از حداقل جهانی دور شود، زیرا نرخ یادگیری را در صورت کاهش شیب تابع خطا افزایش میدهد.

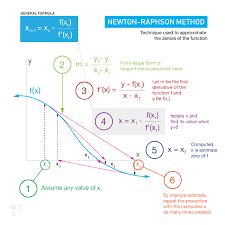
• این روش ممکن است به حالت نوسان یا oscillation برود و از همگرایی دور شود، زیرا نرخ یادگیری را در صورت افزایش شیب تابع خطا کاهش میدهد.

• این روش نیاز به محاسبهی مشتق دوم تابع خطا نسبت به وزن دارد که ممکن است پیچیده و زمانبر باشد









چرا رویکرد مبتنی بر Hessian از لحاظ نظری بهتر از SGD است؟ اکنون، بهینه‌سازی مرتبه دوم با استفاده از روش نیوتن برای یافتن تکراری «x» بهینه، یک هک هوشمندانه برای بهینه‌سازی سطح خطا است، زیرا برخلاف SGD که در آن صفحه را در نقطه x\_0 قرار می‌دهید و سپس پرش گام به گام را تعیین می‌کنید. در بهینه‌سازی مرتبه دوم، منحنی درجه دوم را در x\_0 پیدا می‌کنیم و مستقیماً حداقل انحنا را پیدا می‌کنیم. این فوق العاده کارآمد و سریع است. ولی !!! با این حال، از نظر تجربی، آیا اکنون می توانید تصور کنید که یک Hessian برای شبکه ای با میلیون ها پارامتر محاسبه کنید؟ البته بسیار ناکارآمد می شود زیرا مقدار ذخیره سازی و محاسبات مورد نیاز برای محاسبه Hessian نیز مرتبه درجه دوم است. بنابراین، اگرچه از نظر تئوری، این عالی است، اما در عمل بسیار بد است. ما به یک هک برای هک نیاز داریم! و به نظر می رسد که پاسخ در Gradient های مزدوج نهفته است. ماتریس Hessian ماتریس مربعی از مشتقات جزئی مرتبه دوم یک اسکالر است که انحنای محلی یک تابع چند متغیره را توصیف می کند. به طور خاص در مورد یک شبکه عصبی، Hessian یک ماتریس مربع است که تعداد سطرها و ستون‌ها برابر با تعداد کل پارامترهای شبکه عصبی است.

ویکرد مبتنی بر Hessian در بهینه سازی شبکه عصبی یک روشی است که از ماتریس دومین مشتق تابع هزینه یا ماتریس هسین Hessian استفاده میکند. این روش میتواند سرعت و دقت همگرایی شبکه عصبی را بهبود بخشد و از گیر کردن در کمینههای محلی جلوگیری کند. اما این روش هم دارای معایبی است که عبارتند از:

• محاسبه ماتریس هسین برای شبکههای عصبی با ابعاد بالا ممکن است بسیار پیچیده و زمانبر باشد.

• ماتریس هسین ممکن است نا مثبت تعریف شده یا نا معین باشد که باعث میشود روش بهینهسازی ناپایدار شود.

• ماتریس هسین ممکن است تغییرات بزرگی در مقادیر خود داشته باشد که باعث میشود روش بهینهسازی حساس به انتخاب پارامترهای

این روش میتواند جهت بهینه برای به روز رسانی وزنها و بایاسها را تعیین کند که منجر به سرعت و دقت بالاتر همگرایی شبکه عصبی میشود.

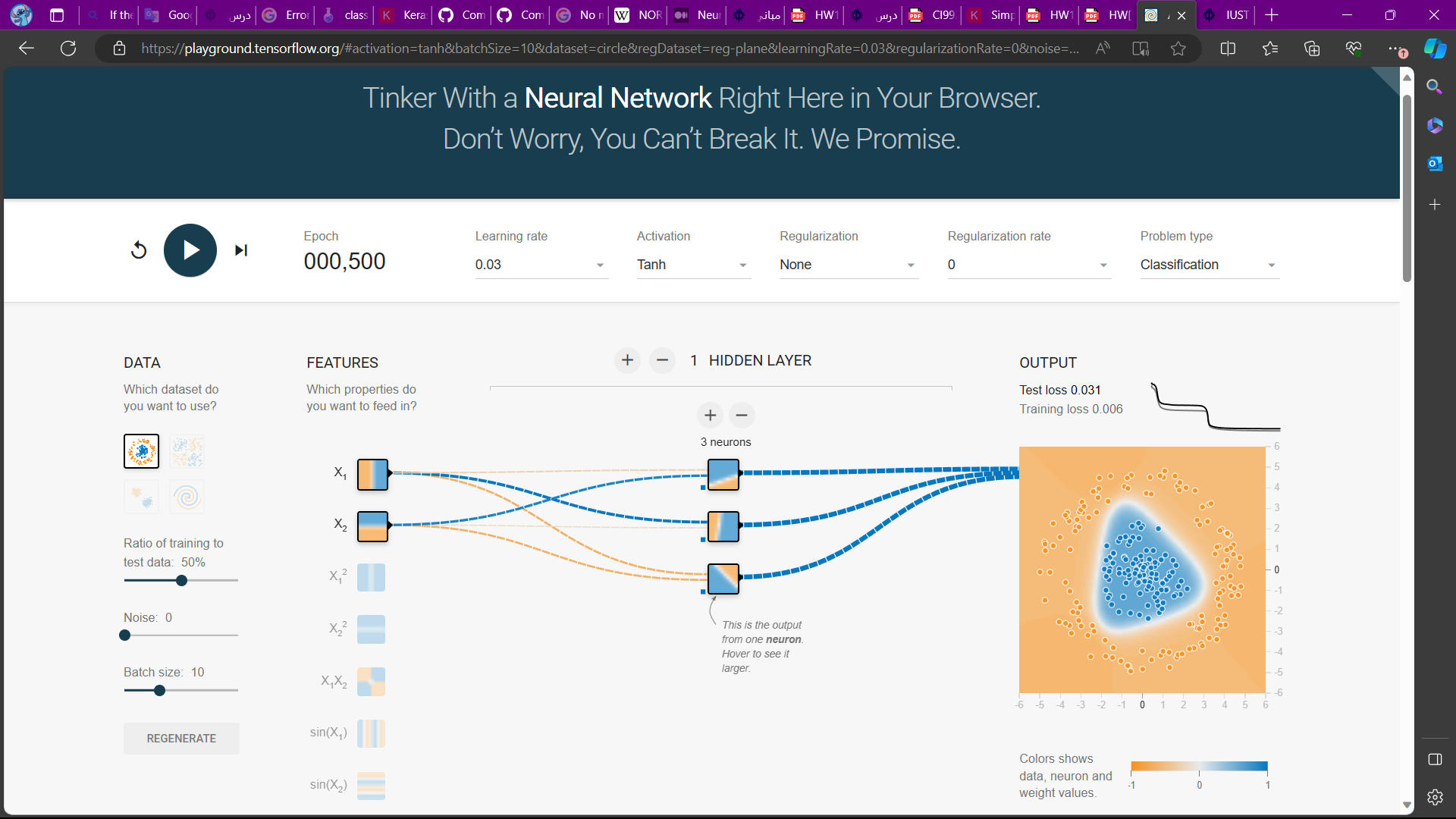
• این روش میتواند از گیر کردن در کمینههای محلی جلوگیری کند و به دنبال کمینههای سراسری باشد که معمولاً مقدار تابع هزینه را کمتر میکنند.

• این روش میتواند نرخ یادگیری را برای هر وزن به طور جداگانه تنظیم کند و از انتخاب یک نرخ یادگیری ثابت یا کاهشی برای تمام وزنها بپرهیزد که ممکن است باعث کند شدن یادگیری یا پرش از کمینهها شود

3- به این لینک مراجعه کنید و مدل MLP با 1 لایه پنهان و 3 نورون میانی را روی هر چهار دیتا موجود در وبسایت ایجاد کنید. سپس با توابع فعالساز مختلف چندبار مدل را نهایتا تا 500 مرحله آموزش دهید و تاثیر این توابع فعالساز را روی هر دیتا تحلیل کنید.

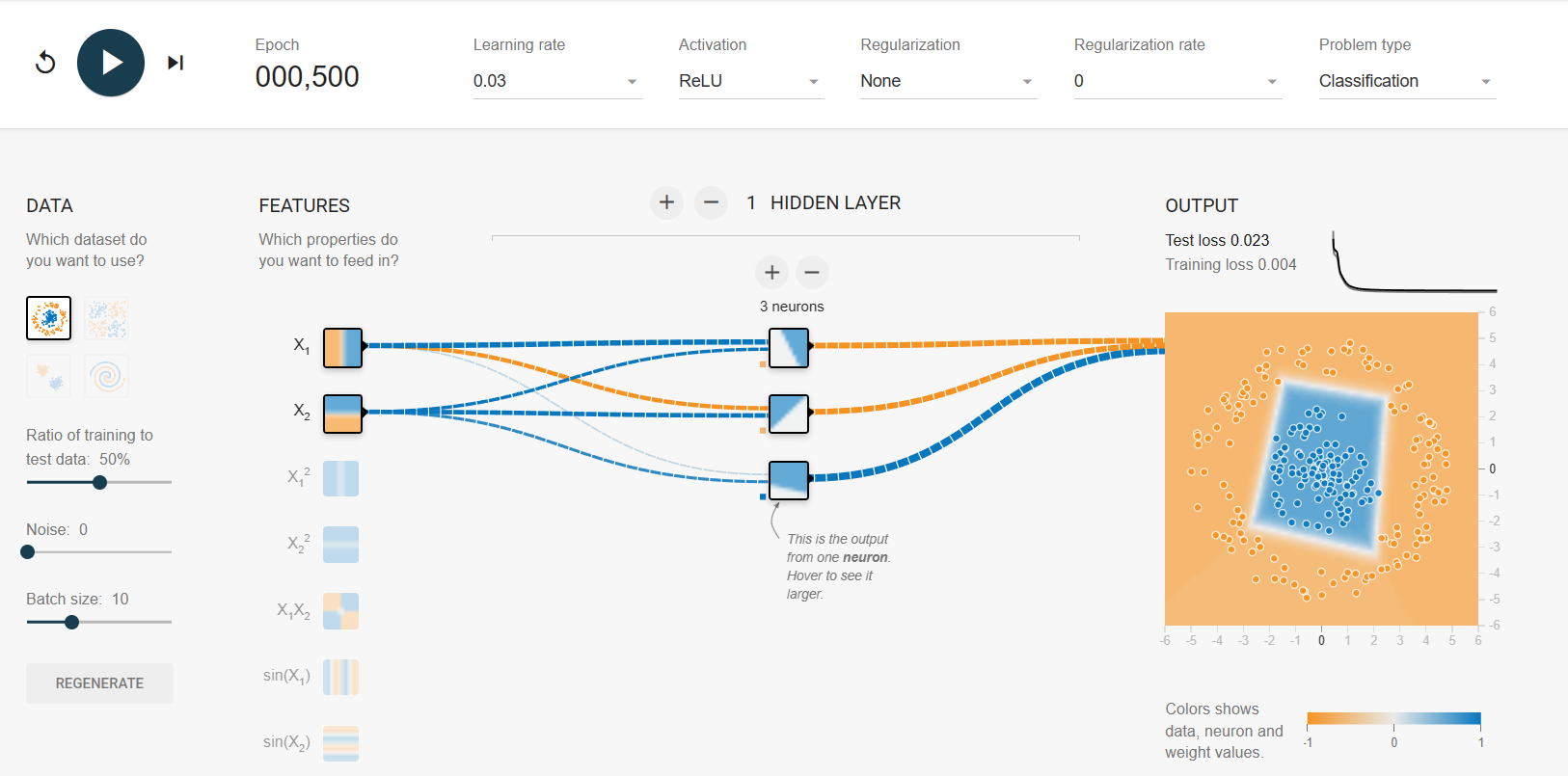
پاسخ 3 )

1. دیتاست circle



مقدار لاس = 0.031

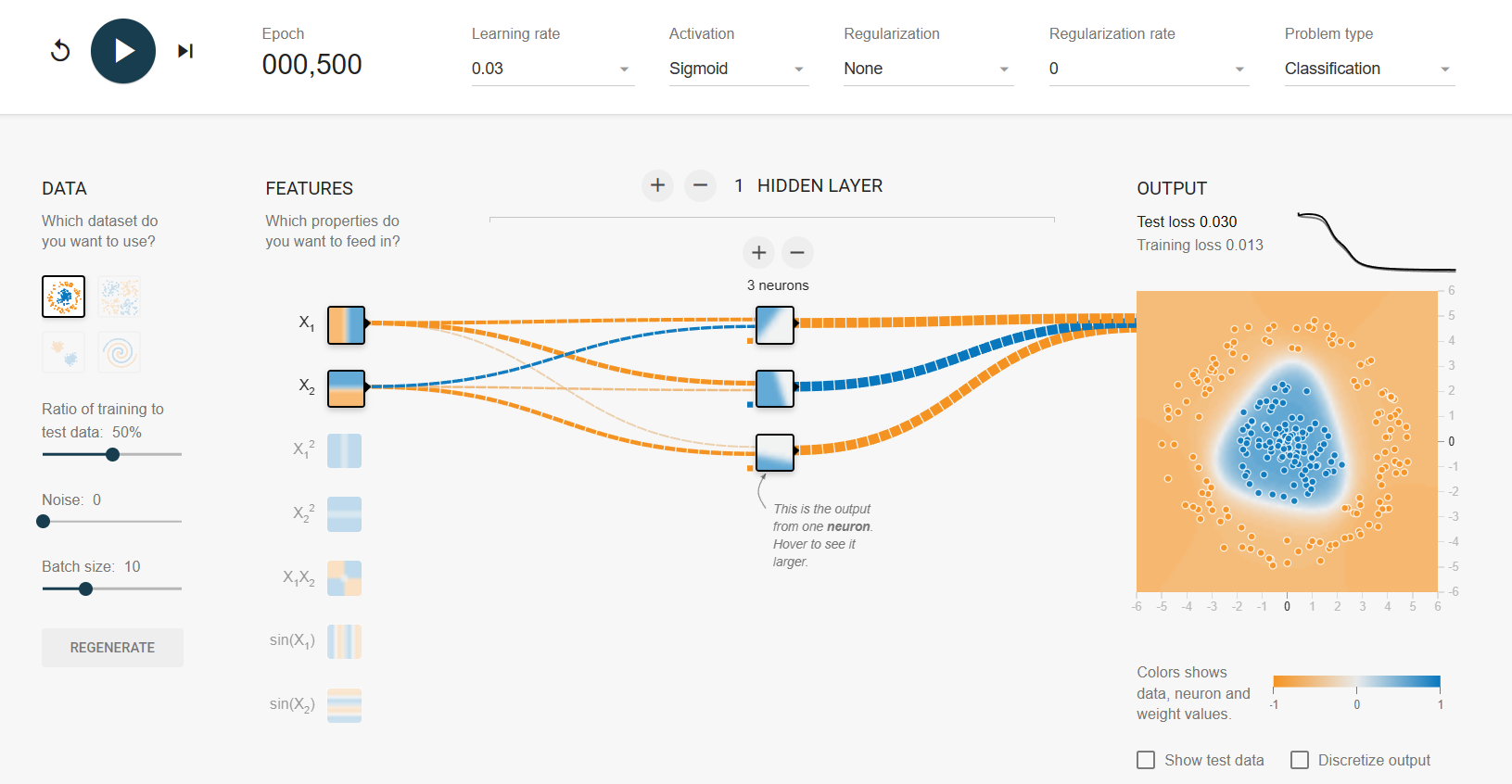
مقدار خوبی است و خوب تفکیک شده است و مرز تقریبا مثلثی تشکیل داده و محدود بین بازه 1 و -1 است باعث سرعت همگرایی می شود



کم ترین مقدار لاس بهترین تابع فعال سازی برای مجموعه داده دایره ای به نوع مدل شبکه عصبی و لایه خروجی بستگی دارد. برای لایه‌های پنهان، یک انتخاب رایج واحد خطی اصلاح‌شده یا ReLU است که رایج‌ترین تابع فعال‌سازی امروزه است. ReLU یک تابع ساده است که در صورت مثبت بودن ورودی و در غیر این صورت صفر را برمی گرداند. ReLU مزیت غلبه بر مشکل گرادیان ناپدید شدن را دارد و به مدل‌ها امکان می‌دهد سریع‌تر یاد بگیرند و عملکرد بهتری داشته باشند.

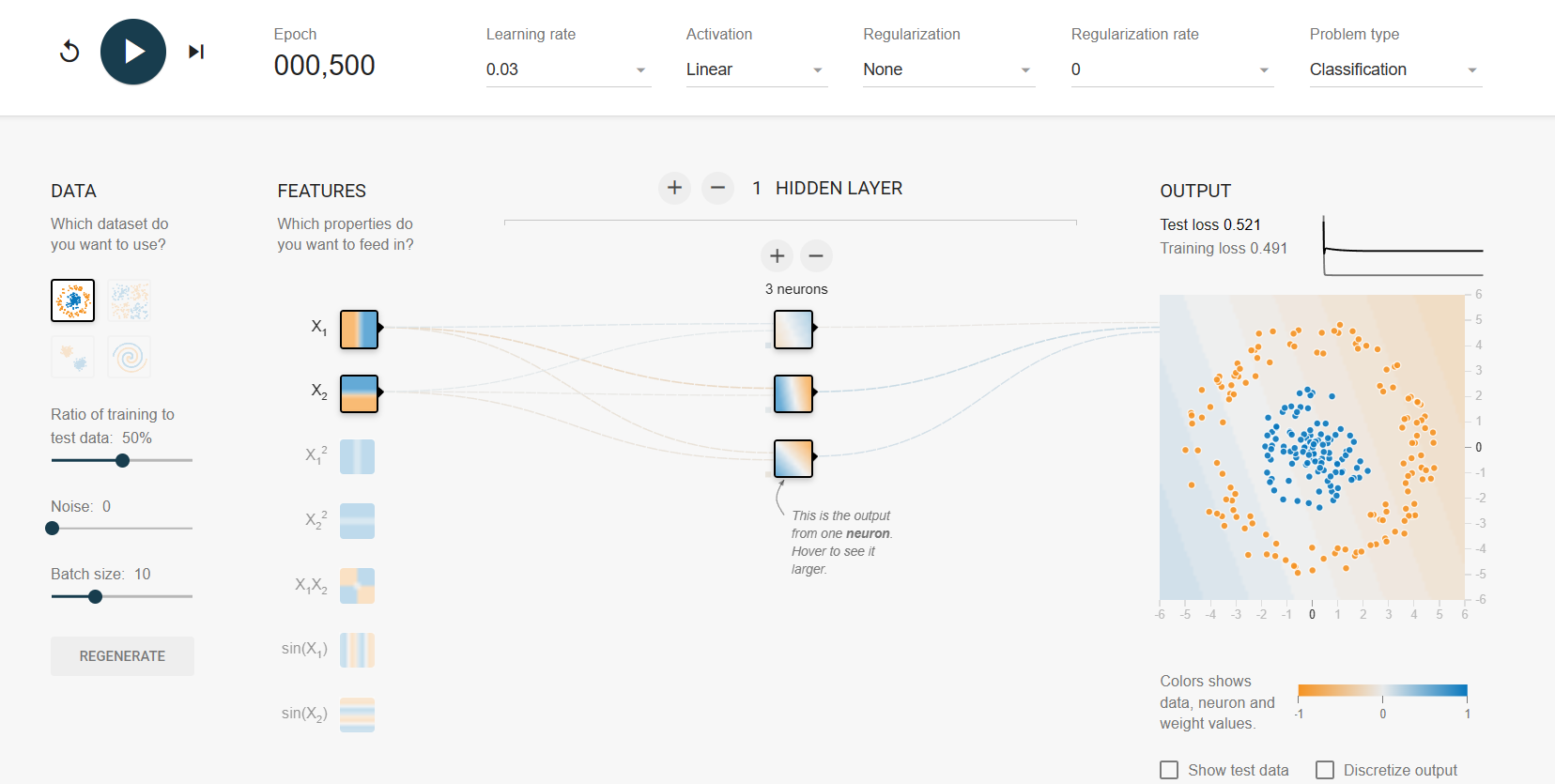
مقدار لاس=0.023

به صورت 4 ضلعی مرز مشخص شده است. همگرایی و شیب مناسب تری دارد



مقدار لاس = 0.030

مقدار خوبی است و خوب تفکیک شده است و مرز تقریبا مثلثی تشکیل داده



مقدار لاس از همه موارد دیگر بیش تر است و تفکیک پذیری صورت نگرفته چون این تابع فعالسازی خطی است و نمی تواند ویژگی های غیر خطی را پوشش دهد و تفکیک پذیر و تعمیم پذیری درستی داشته باشد همانگونه که در سوال 2 اشاره کردیم به تفاوت توابع فعال سازی خطی و غیرخطی.

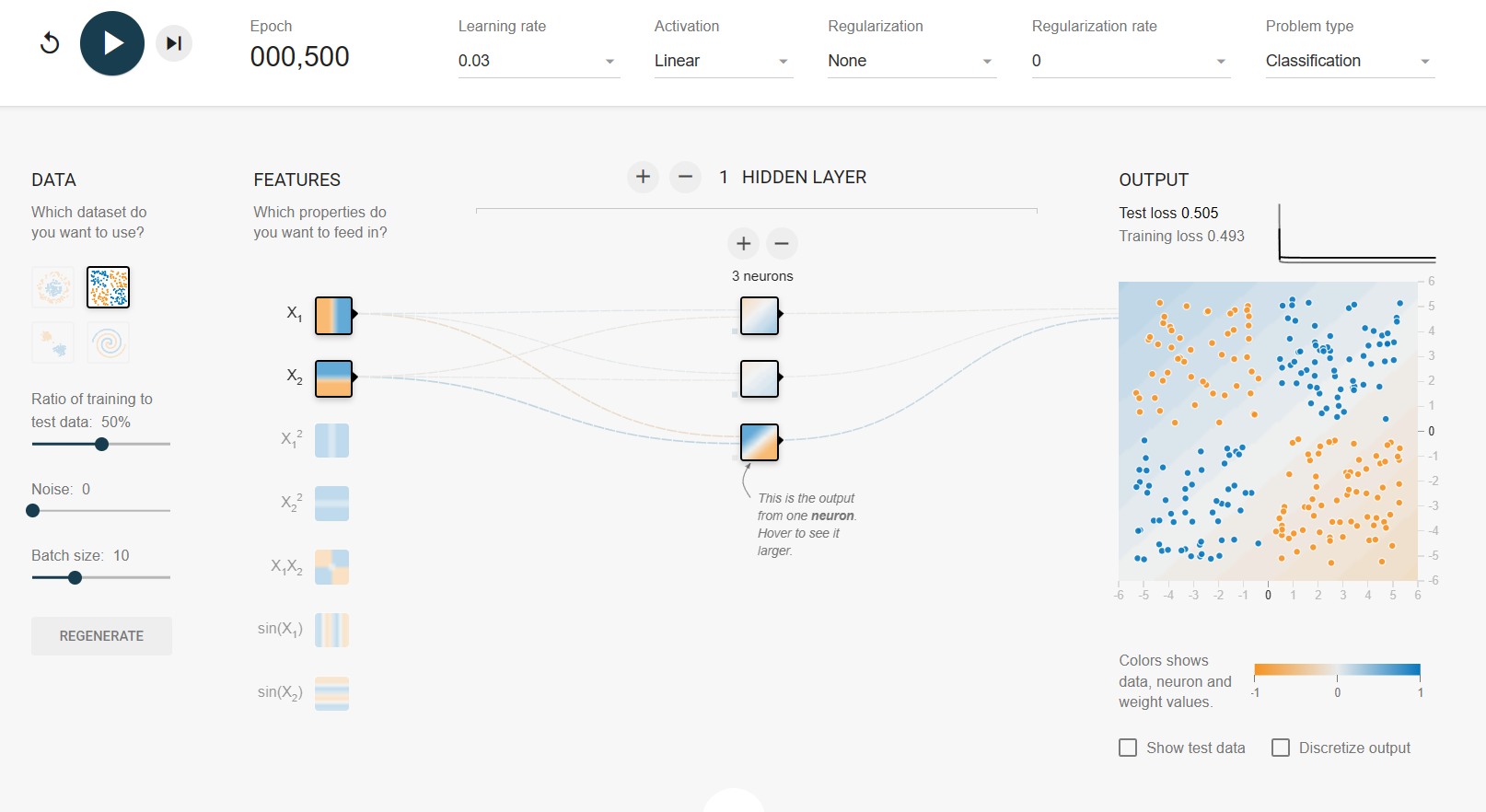
اینجا 50 درصد لاس داریم یعنی به صورت کاملا رندم هم و بدون اعمال شبکه هم در حالت عدی با امید ریاضی رندوم رو به رو خواهیم شد.

تابع خطی نمی تواند مرز دایره ای را تشخیص دهد باید از ویژگی های درجه دو استفاده کند و غیر از این این تابع خطی بقیه توابع تا حد خوبی مرز را شناسایی کردند و مرز را به صورت چند ضلعی و یا دایروی شناسایی کردند.

بهترین تابع فعال سازی برای این دیتا ست به ترتیب

Relu>Tan h>sigmoid>linear

1. Exclusive or

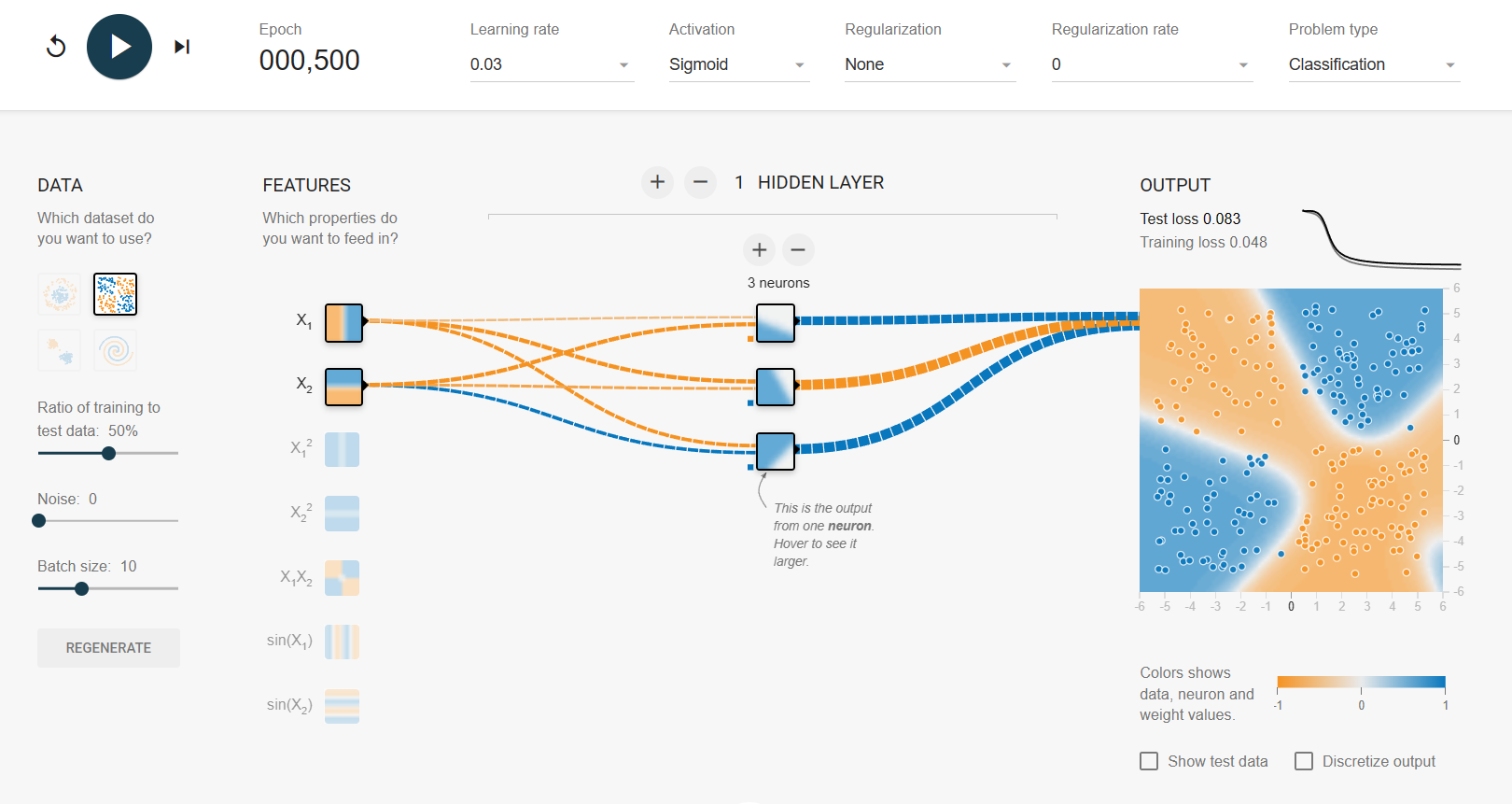
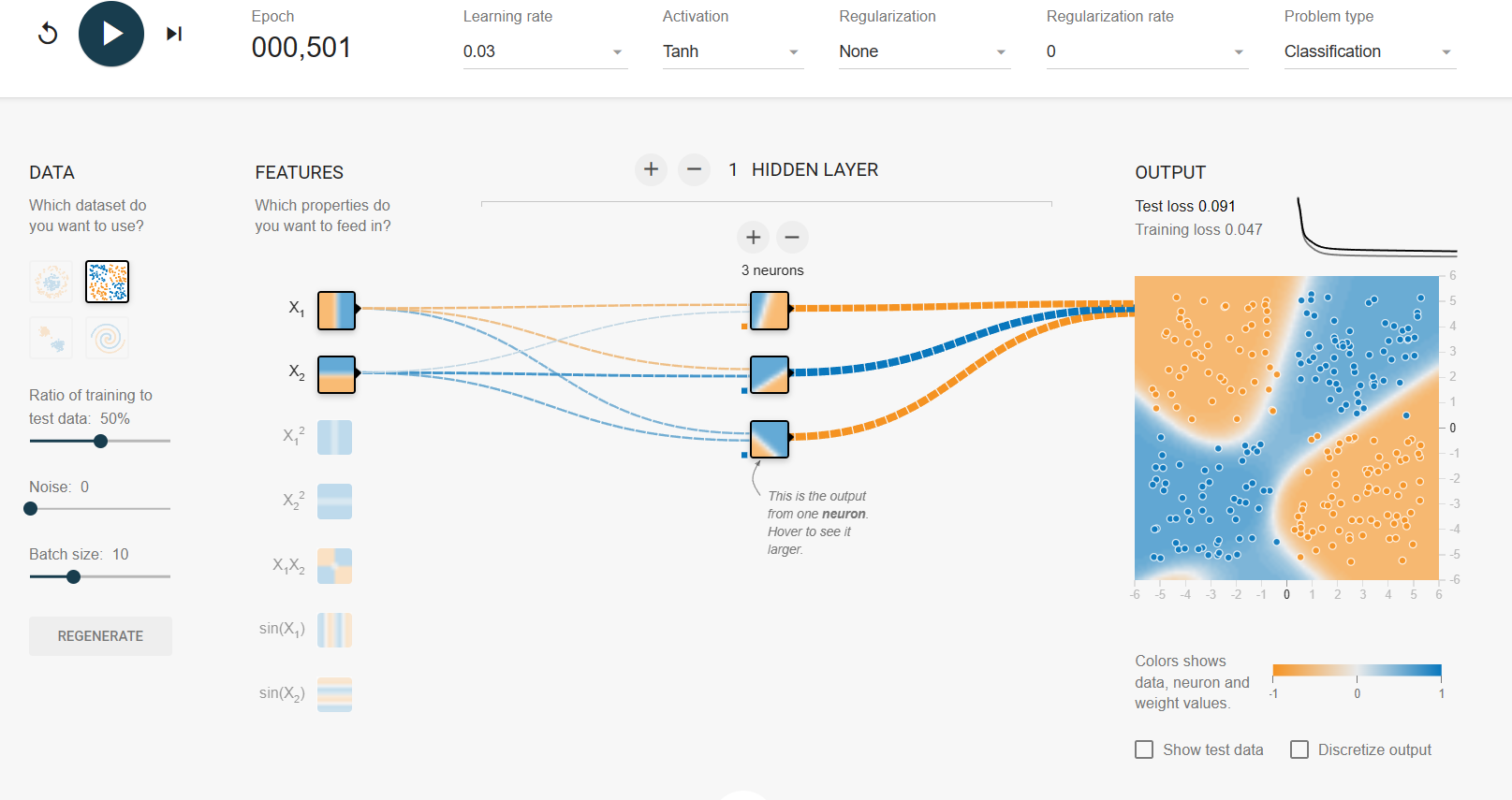


مقدار لاس =0.0505

مقدار لاس از همه موارد دیگر بیش تر است و تفکیک پذیری صورت نگرفته چون این تابع فعالسازی خطی است و نمی تواند ویژگی های غیر خطی را پوشش دهد و تفکیک پذیر و تعمیم پذیری درستی داشته باشد همانگونه که در سوال 2 اشاره کردیم به تفاوت توابع فعال سازی خطی و غیرخطی. و گویا رندوم و بدون داشتن شبکه هم به همین نتیجه برسیم

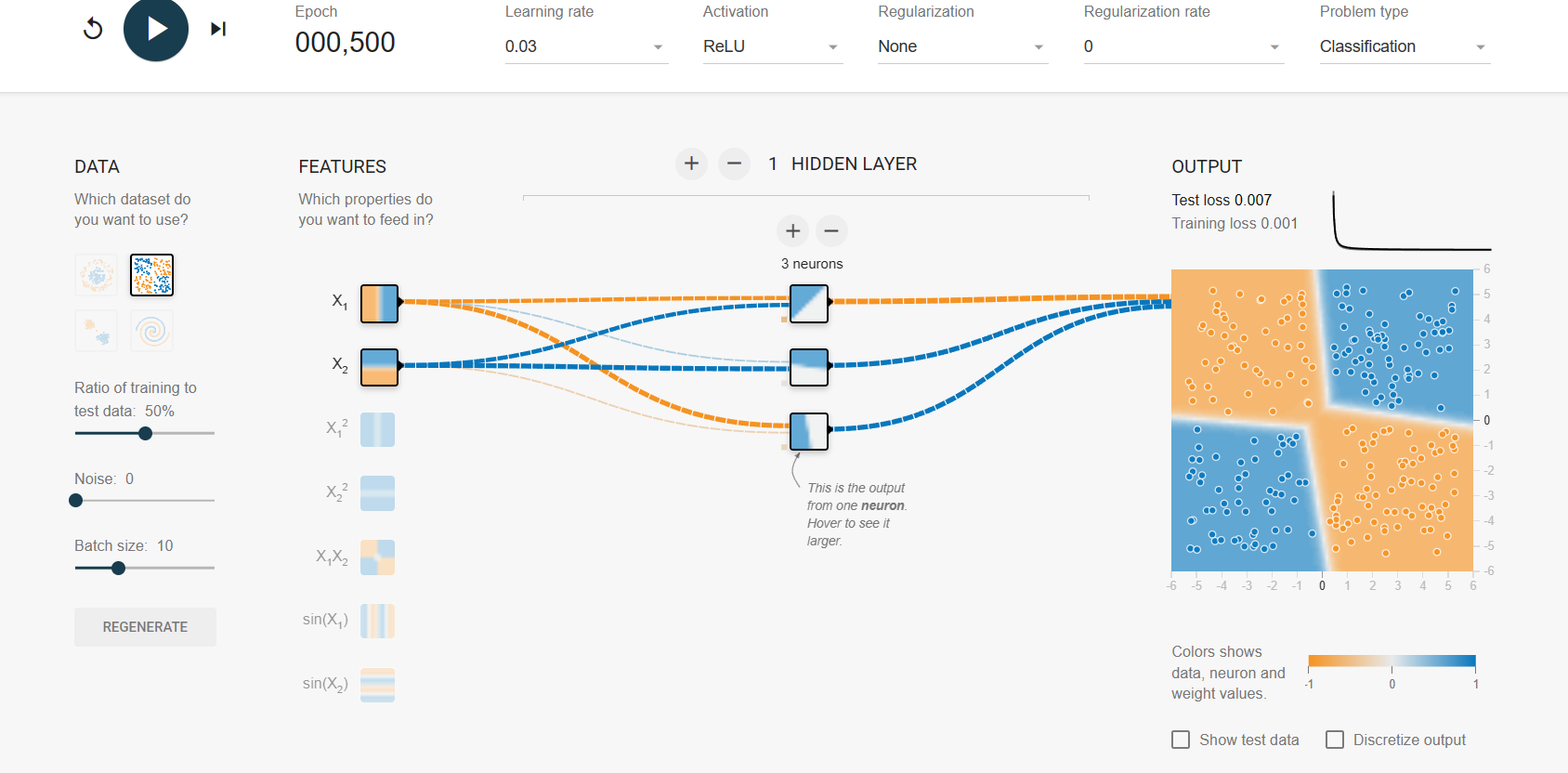
مقدار لاس =0.0505

مقدار لاس =0.0505

مقدار لاس =0.083 هنوز خوب همگرا نشده در ایپوک های بعد به مرز بهتری خواهد رسید یا با نورون های بیش تراما تا حدود خوبی تنوانسته شناسایی کند  

مقدار لاس =0.091

خوب همگرا نشده در ایپوک های بعد به مرز بهتری خواهد رسید یا با نورون های بیش تراما تا حدود خوبی تنوانسته شناسایی کند



مقدار لاس =0.007

تقریبا لاس به صفر رسیده و مرز ها کاملا شناسایی شده و بهترین تابع بوده

کم ترین مقدار لاس در بین توابع فعال سازی

Relu >sigmoid> Tan h >linear

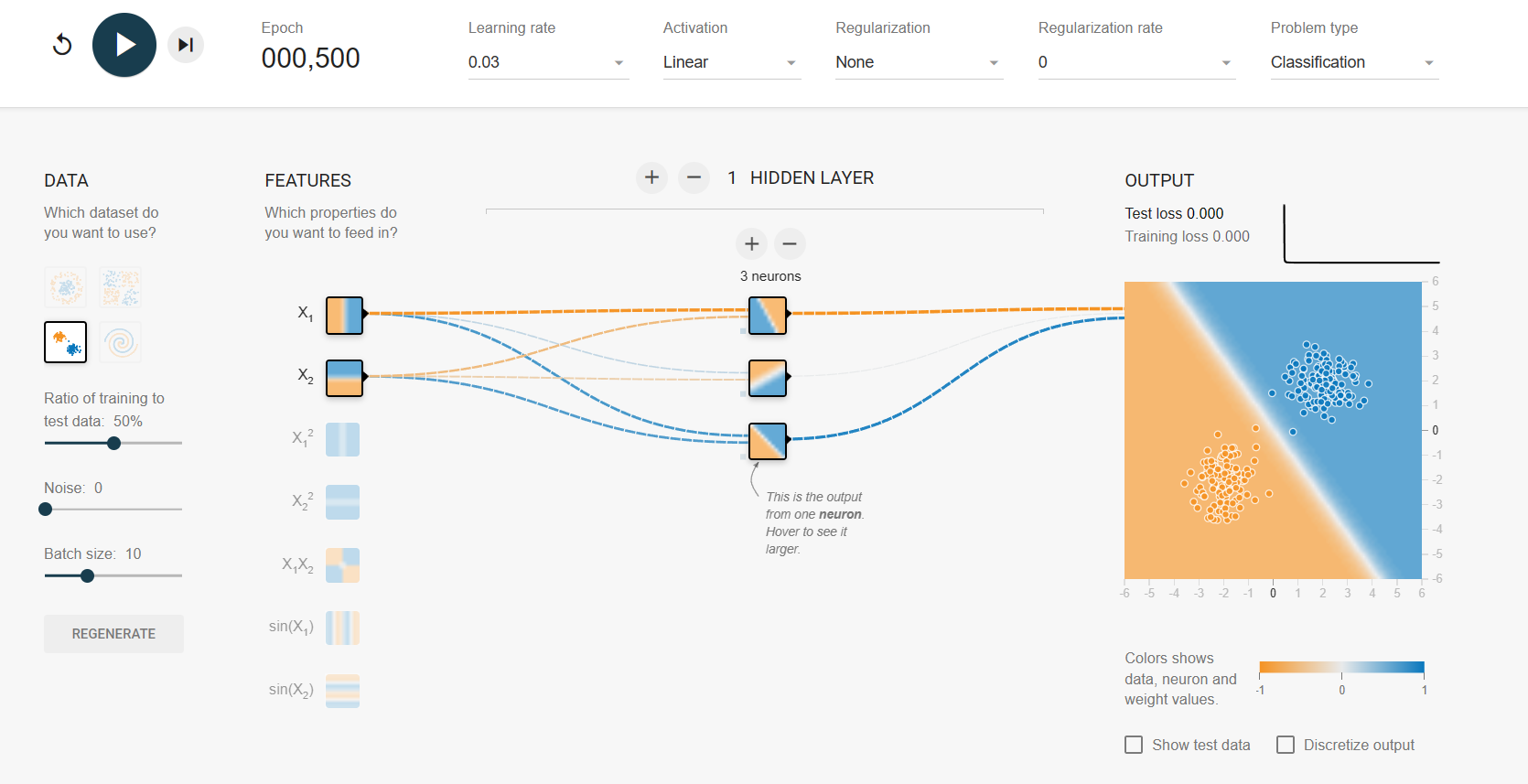
بهترین تابع فعال سازی برای یک مجموعه داده انحصاری یا (XOR) یک تابع غیرخطی است که می تواند داده ها را به دو کلاس تقسیم کند. مجموعه داده XOR نوعی داده است که دارای دو ویژگی (مختصات x و y) و دو کلاس (0 یا 1) است، که در آن کلاس 1 است اگر x و y مقادیر متفاوتی داشته باشند و در غیر این صورت 0 است. نمونه ای از مجموعه داده های XOR در زیر نشان داده شده است: ![داده داده XOR]

یک تابع فعال سازی خطی، مانند تابع هویت یا تابع سیگموئید، نمی تواند داده ها را به دو کلاس تقسیم کند، زیرا داده ها به صورت خطی قابل تفکیک نیستند. یک تابع فعال سازی خطی یک خط مستقیم را به عنوان مرز تصمیم تولید می کند که برخی از نقاط داده را به اشتباه طبقه بندی می کند. به عنوان مثال، تابع سیگموئید یک مرز تصمیم مانند زیر تولید می کند:

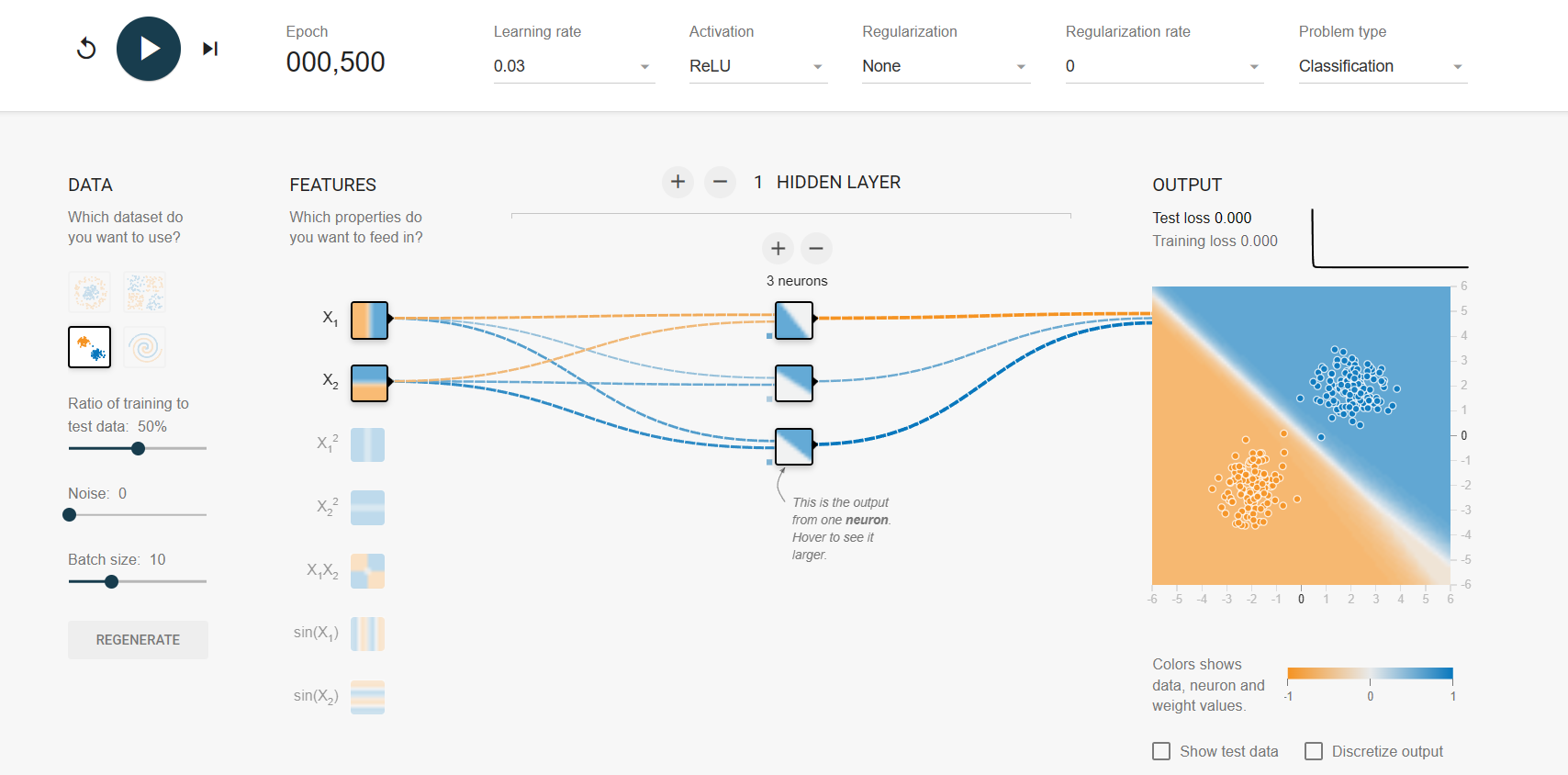
![عملکرد سیگموئید]

یک تابع فعال‌سازی غیرخطی، مانند تابع tanh یا تابع ReLU، می‌تواند داده‌ها را به دو کلاس تقسیم کند، زیرا داده‌ها به صورت غیرخطی قابل تفکیک هستند. یک تابع فعال سازی غیرخطی یک خط منحنی را به عنوان مرز تصمیم تولید می کند که به درستی تمام نقاط داده را طبقه بندی می کند. به عنوان مثال، تابع tanh یک مرز تصمیم مانند زیر تولید می کند: ![تابع Tanh] بنابراین، بهترین تابع فعال سازی برای یک مجموعه داده XOR یک تابع غیرخطی است، مانند تابع tanh یا تابع ReLU. این توابع می توانند الگوهای پیچیده در داده ها را بیاموزند و در وظایف طبقه بندی به خوبی عمل کنند. برای اطلاعات بیشتر در مورد عملکردهای فعال سازی و نحوه انتخاب آنها،

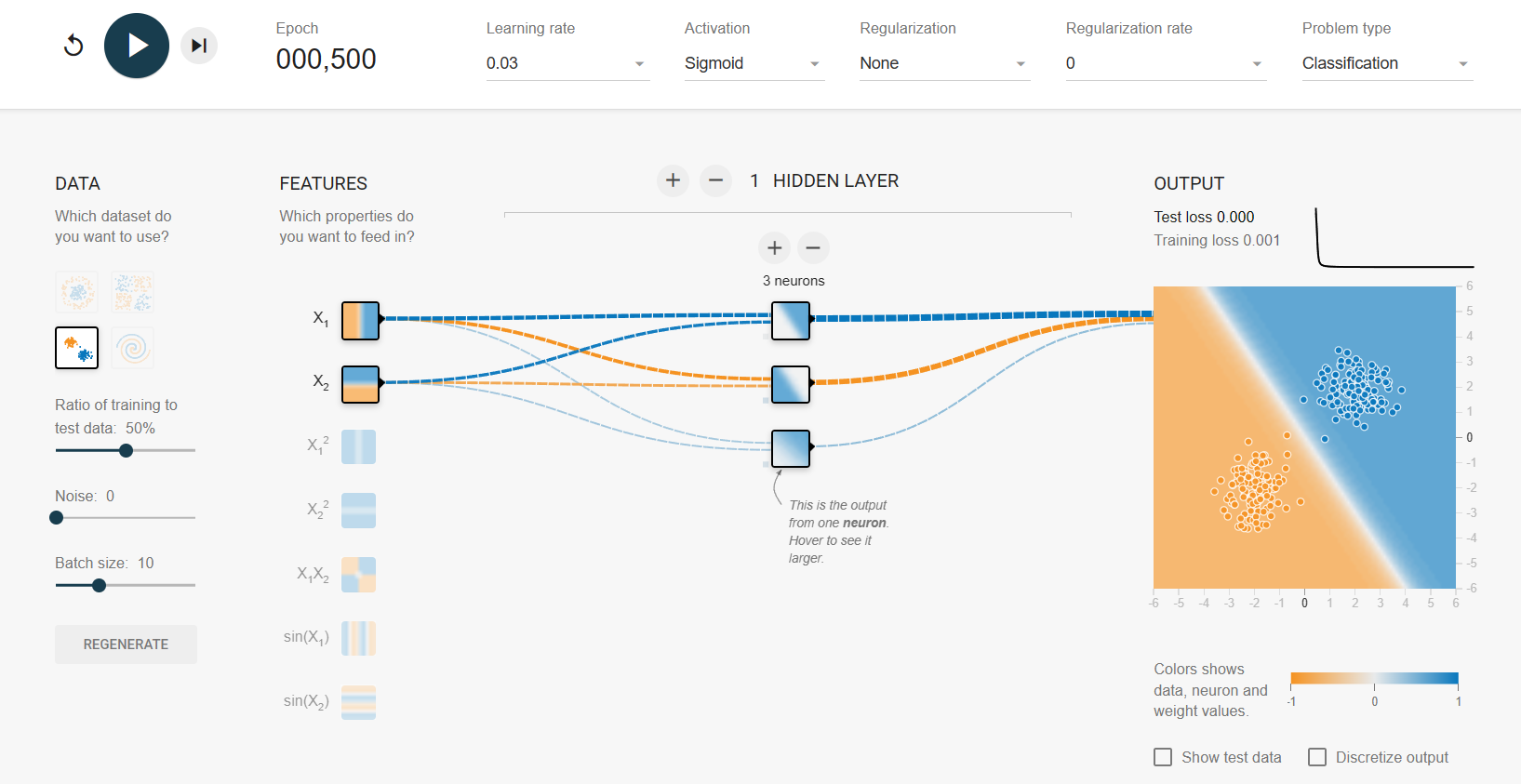
1. Gaussian



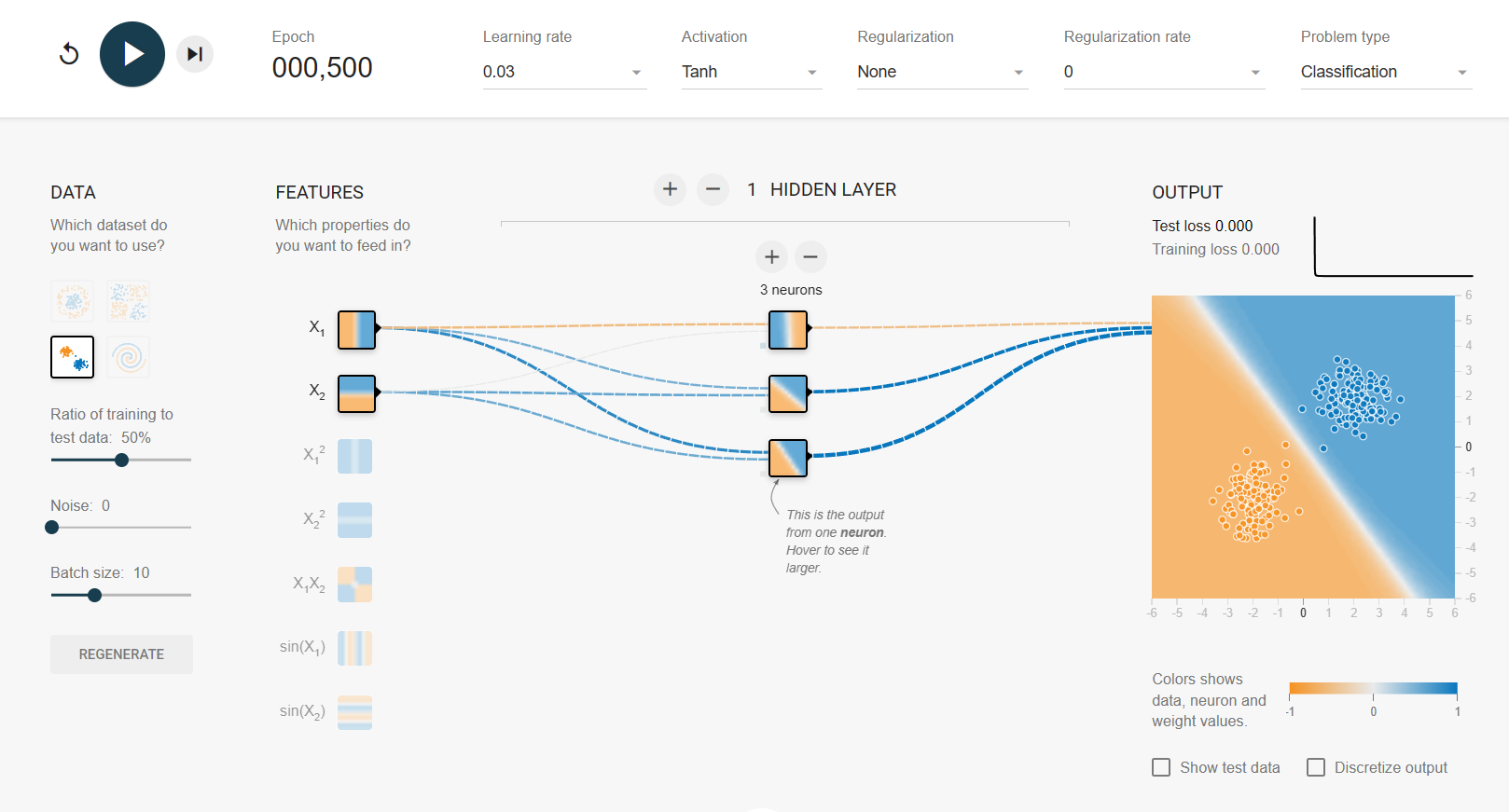
کم ترین مقدار لاس=0



کم ترین مقدار لاس=0



کم ترین مقدار لاس=0



کم ترین مقدار لاس=0

Relu=Tan h=sigmoid=linear

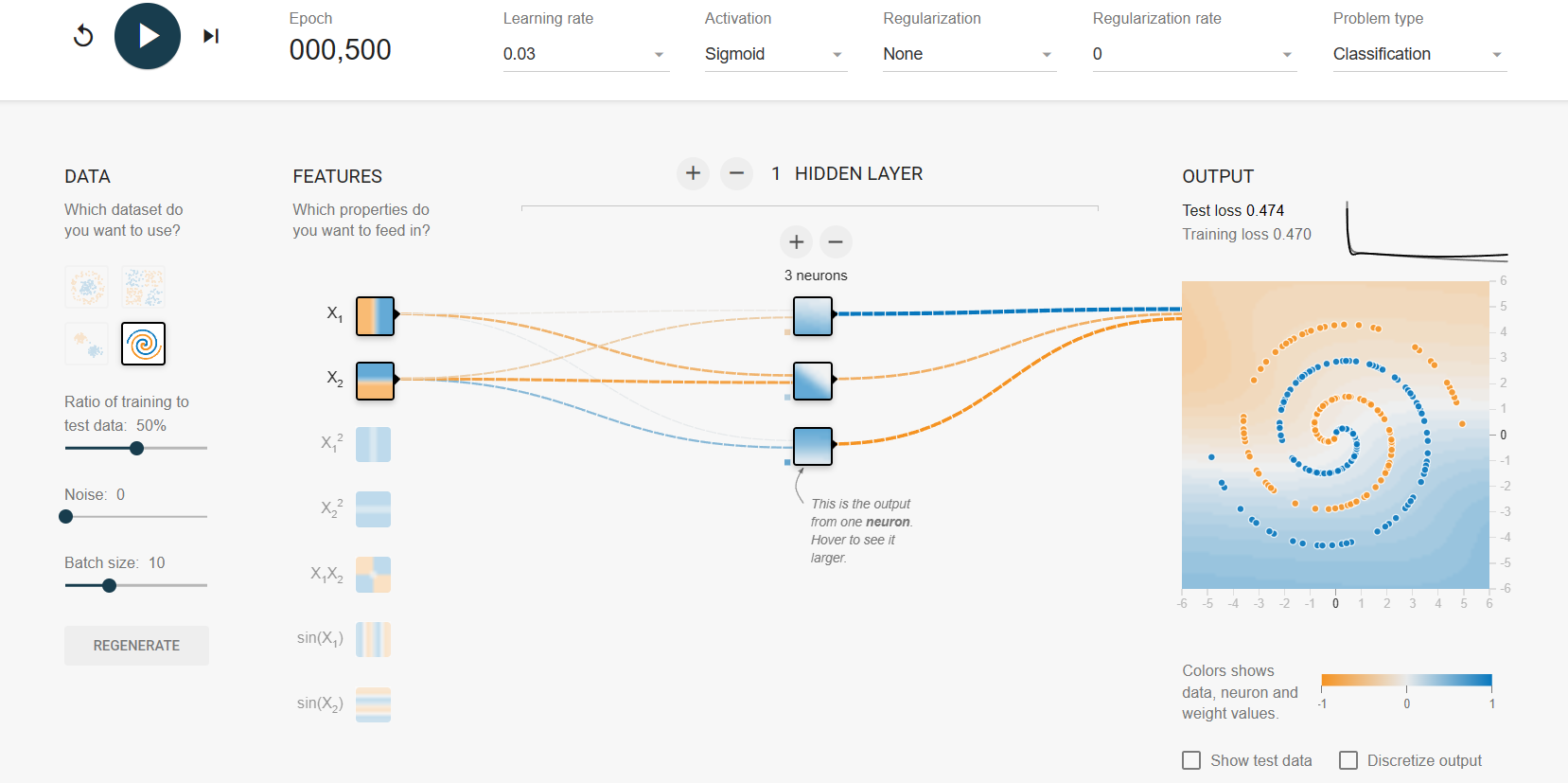
چون داده و دیتاست به دو مجموعه تقسینم می سود با یک خط ساده قابلیت جداسازی دارد و بعد از 500 مرحله هر چهار تابع حتی تابع خطی ساده بعد از چند مرحله اموزش توانسته مرز را بسیار خوب جداسازی کند.با این که قدر برای ویژگی استخراج ویژگی های پیچیده را ندارد

داده گاوسی نوعی از داده است که دارای دو ویژگی (مختصات x و y) و یک یا چند کلاس است که در آن نقاط داده بر اساس توزیع نرمال یا گاوسی توزیع می شوند. نمونه ای از داده های گاوسی در زیر نشان داده شده است: !داده های گاوسی بهترین تابع فعال سازی برای داده های گاوسی به نوع مدل شبکه عصبی و لایه خروجی بستگی دارد. برای لایه‌های پنهان، یک انتخاب رایج واحد خطی خطای گاوسی یا GELU است که یک تابع فعال‌سازی است که به صورت x Φ(x) تعریف می‌شود، که در آن Φ(x) تابع توزیع تجمعی استاندارد گاوسی است. عملکرد فعال سازی GELU برخلاف تابع ReLU که در صفر ناپیوستگی دارد، مزیت صاف و پیوسته بودن را دارد. تابع GELU همچنین این مزیت را دارد که می‌تواند الگوهای خطی و غیرخطی را در داده‌ها ثبت کند، برخلاف توابع سیگموئید یا tanh که در نهایت اشباع می‌شوند. نشان داده شده است که عملکرد GELU از سایر عملکردهای فعال سازی در برنامه های مختلف مانند پردازش زبان طبیعی و دید کامپیوتری بهتر عمل می کند. برای لایه خروجی، انتخاب تابع فعال سازی به نوع مشکل پیش بینی بستگی دارد. اگر مشکل یک مشکل رگرسیونی باشد، جایی که خروجی یک مقدار پیوسته است، یک انتخاب رایج تابع هویت است که به سادگی ورودی را به عنوان خروجی برمی گرداند. تابع هویت این مزیت را دارد که ساده و خطی است و هیچ محدودیتی در محدوده خروجی اعمال نمی کند. با این حال، تابع هویت همچنین دارای مضرات حساس بودن به نقاط پرت و نویز در داده ها است. اگر مشکل یک مشکل طبقه بندی باشد، که در آن خروجی یکی از چندین کلاس ممکن است، یک انتخاب رایج تابع softmax است. تابع softmax تابعی است که یک بردار از مقادیر بین 0 و 1 را برمی گرداند که نشان دهنده احتمالات ورودی متعلق به هر کلاس است. تابع softmax این مزیت را دارد که قابل تفکیک و تفسیر واضح است. با این حال، تابع softmax این مضرات را نیز دارد که به نقاط پرت حساس است و در مرکز صفر قرار ندارد. بنابراین، بهترین تابع فعال‌سازی برای داده‌های گاوسی به نوع مدل شبکه عصبی و لایه خروجی بستگی دارد. بسته به نوع مشکل پیش‌بینی، یک ترکیب ممکن برای لایه‌های پنهان و هویت یا softmax برای لایه خروجی است.

تفکیک پذیری این دیتاست اسان بوده و هر چهار تابع به خوبی توانسته اند لاس را به صفر برسانند.

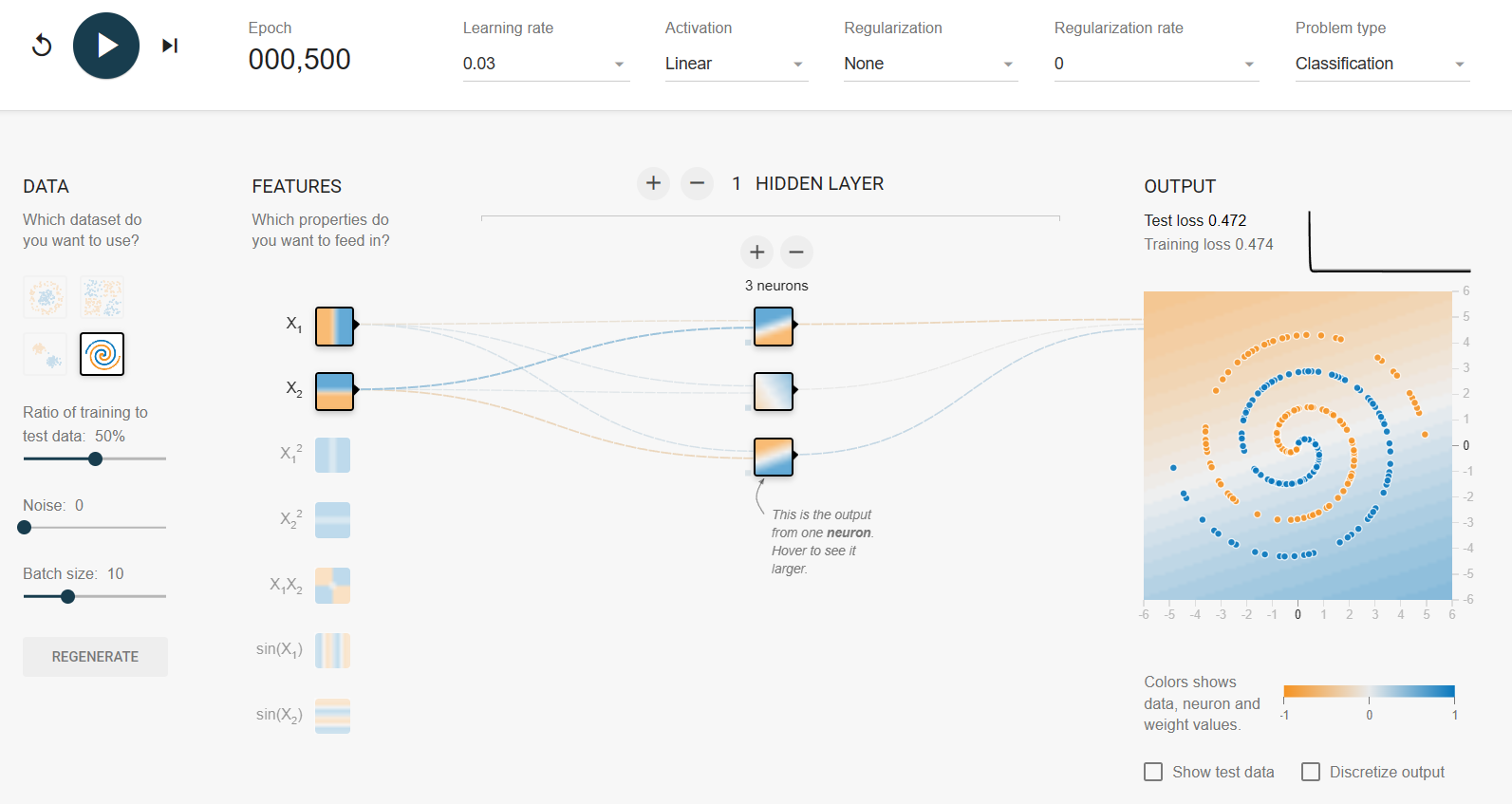
1. Spiral

این دیتاست بسیار پیچیده است و چند لایه ی تو در تو را شامل می شود و نمی توان با داشتن یک لایه ی نهان که تنها سه نورون دارد به نتیجه ی مطلوب رسد و مرز را پیدا کرد بلکه نیاز به لایه های پنهان بیش تر نورون های بیش تر و ویژگی های غیر خطی و استخراج انواع ویژگی ها نیاز داریم و الان پیش بینی ها به صورت رندوم هم به همین نتیجه می رسیدیم و توابع به نتیجه ای نرسیدند



مقدار لاس = 0.474

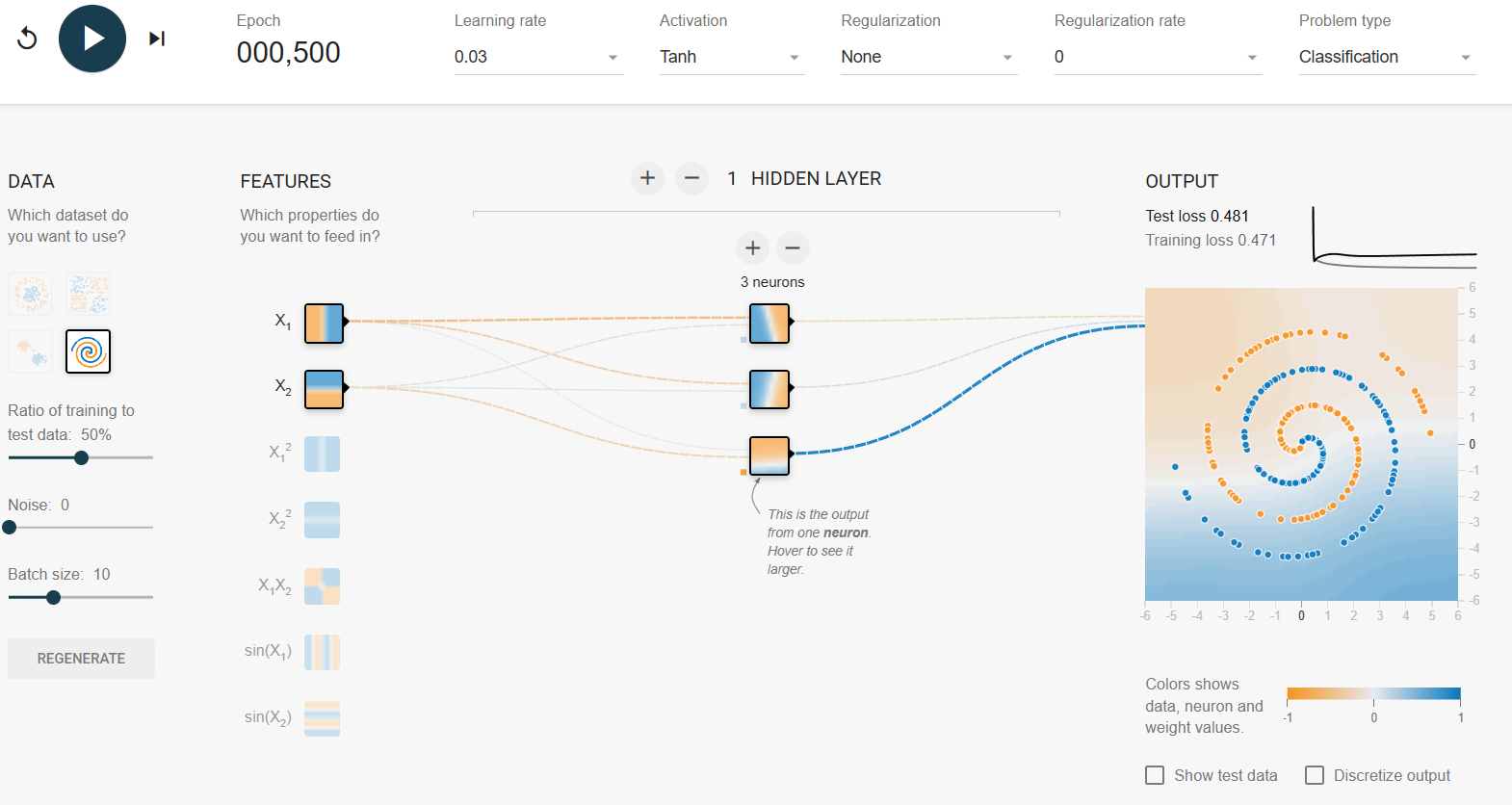
نزدیک به 50 درصد لاس داریم که حالت رندوم میشود و نامناسب است



مقدار لاس = 0.472

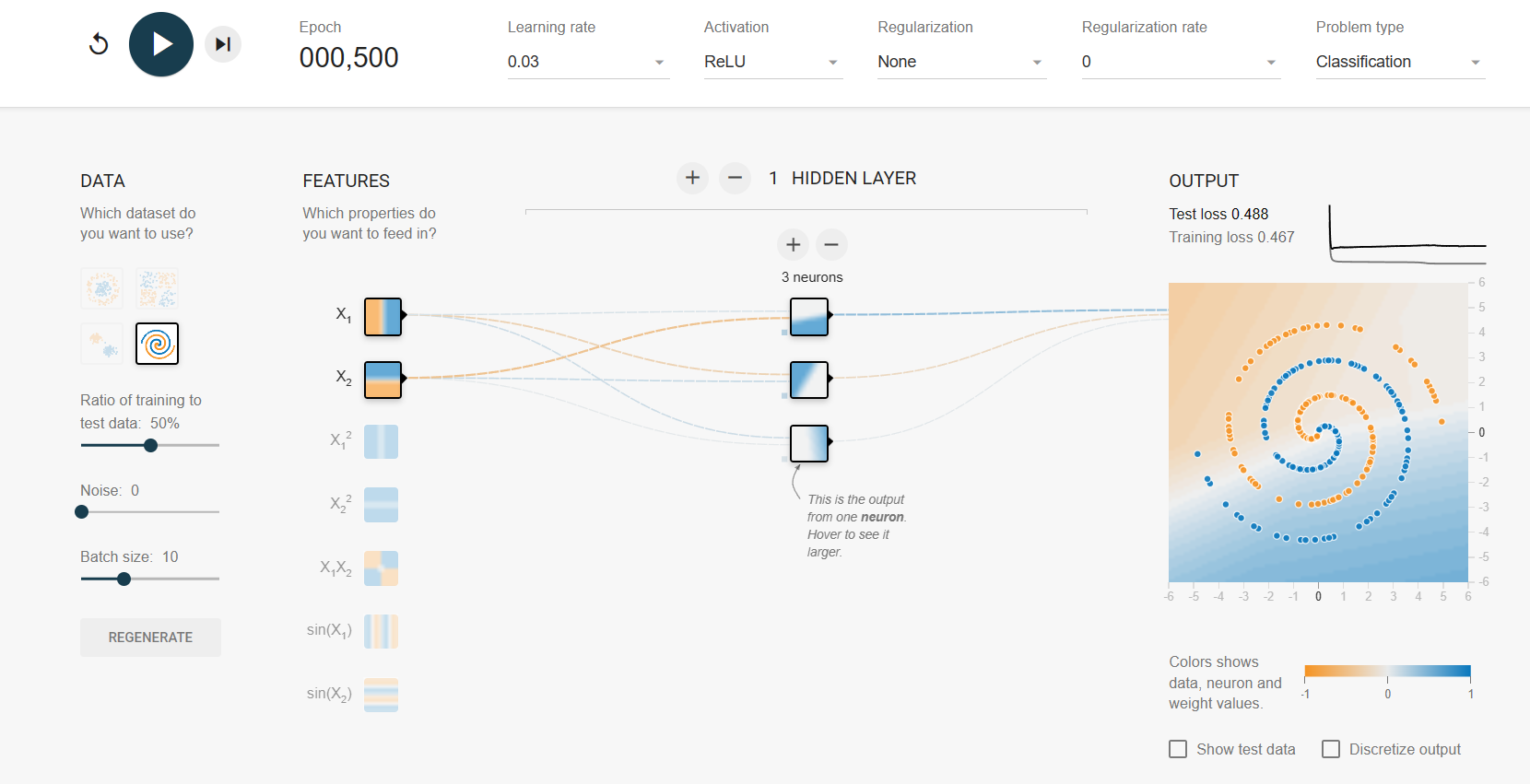
نزدیک به 50 درصد لاس داریم که حالت رندوم میشود و نامناسب است

کم ترین مقدار لاس



مقدار لاس = 0.481

نزدیک به 50 درصد لاس داریم که حالت رندوم میشود و نامناسب است



مقدار لاس = 0.488

نزدیک به 50 درصد لاس داریم که حالت رندوم میشود و نامناسب است

بدترین مقدار و بیشترین لاس

تقریبا لاس دیتا ست ها با یکدیگر برابر است و نتوانتند تفکیک پذیری به خوبی صورت بگیرد حتی بعد از 500 مرحله داده ها بسیار در یکدیگر پراکنده هستند و نتوانستند جداسازی به خوبی صورت بگیرد

4-با توجه به کامنت ها نوتبوک 2-4HW را کامل کنید. از سلول ها ران بگیرید سپس کد و نمودار های رسم شده را تحلیل کنید

پاسخ 4 )

توضیح سلول اول :

در قسمت اول من مدل را به دو روش پیاده سازی کردم

روش اول :

model = Sequential(

    [

        Dense(10, input\_dim=25 , name="L1"),

        Activation( "softmax"),

    ]

)

از دنس تنسورفلو برای پیاده سازی شبکه فولی کانکتد استفاده می کنیم .

• یک لایه خروجی با 10 نورون و تابع فعالسازی softmax که احتمالات تعلق به هر یک از 10 کلاس را محاسبه میکند. هر کلاس متناظر با یک عدد از 0 تا 9 است

خروجی ما 10 نورونه می باشد یا 10 کلاسه و تابع فعال ساز اخر را یک سافت مکس استفاده می کنم که کلاس ها را بر اساس احتمالاتشان پیش بینی کند و یک توزیع احتمالاتی داشته باشیم که کلاس با احتمال بیش تر انتخاب می شود.

تعداد نورون های ورودی یا اینچوت ها 25 تاست

حالا روشی که در آن بیش ترین تابع فعال ساز را استفاده کردم توضیح می دهم:

ین کد یک مدل شبکه عصبی چند لایه پیشرو یا MLP را با استفاده از API سلسله مراتبی یا Sequential در Keras ایجاد میکند. این مدل برای دستهبندی تصاویر دست نوشتهی اعداد 0 تا 9 با استفاده از مجموعه داده hoda طراحی شده است

که اعداد دست نوشته ایرانی را طبقه بندی می کند ب یکی از 10 کلاس عددی ما

یک لایه ورودی شکل یا shape ورودی را برابر با تعداد ویژگیهای دادههای آموزشی تعیین میکند

25 نورون و تابع فعالسازی relu که یک تابع غیرخطی است و میتواند روابط پیچیدهتری را بین ورودی و خروجی بیاموزد.

• یک لایه حذف یا Dropout با نرخ 0.5 که به منظور جلوگیری از بیشبرازش یا overfitting به کار میرود. این لایه به طور تصادفی برخی از نورونهای لایه قبلی را حذف میکند و باعث میشود مدل به ویژگیهای مهمتر توجه کند.

• یک لایه خروجی با 10 نورون و تابع فعالسازی softmax که احتمالات تعلق به هر یک از 10 کلاس را محاسبه میکند. هر کلاس متناظر با یک عدد از 0 تا 9 است

خروجی ما 10 نورونه می باشد یا 10 کلاسه و تابع فعال ساز اخر را یک سافت مکس استفاده می کنم که کلاس ها را بر اساس احتمالاتشان پیش بینی کند و یک توزیع احتمالاتی داشته باشیم که کلاس با احتمال بیش تر انتخاب می شود.



توضیح سلول دوم :

پارامتر دقت را دریافت کرده ایم متریک = دقت

از تابع بهینه ساز آدام استفاده کردیم = یک تابع خوب با همگرایی بالا

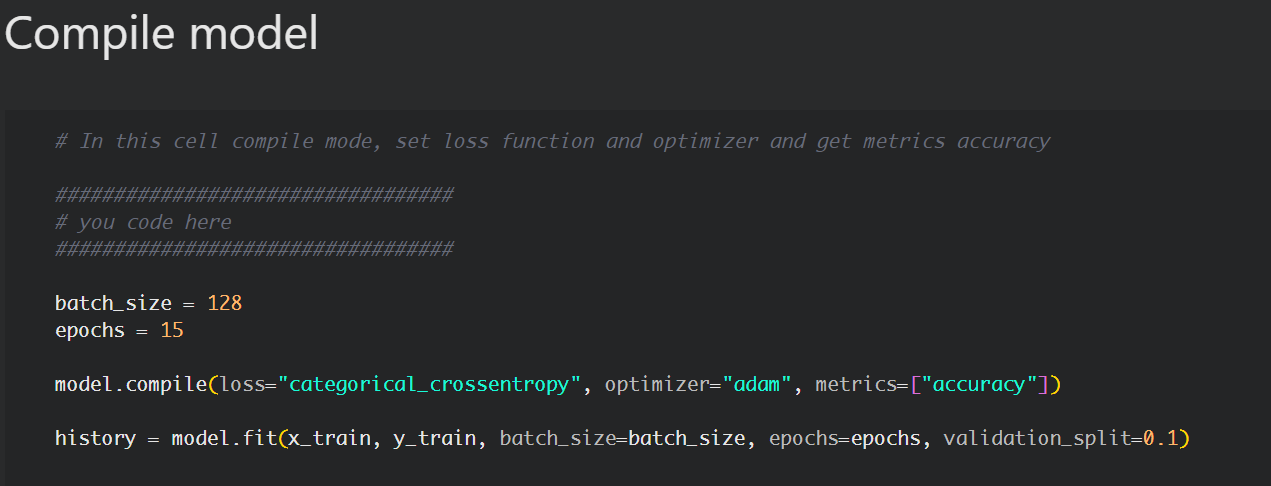
بهینه ساز Adam، مخفف Adaptive Moment Estimation optimizer، یک الگوریتم بهینه سازی است که معمولا در یادگیری عمیق استفاده می شود. این الگوریتم توسعه‌ای از الگوریتم شیب تصادفی (SGD) است و برای به‌روزرسانی وزن‌های یک شبکه عصبی در طول تمرین طراحی شده است.

categorical\_crossentropy: به عنوان یک تابع ضرر برای مدل طبقه بندی چند کلاسه که در آن دو یا چند برچسب خروجی وجود دارد استفاده می شود. برچسب خروجی یک مقدار رمزگذاری یک دسته داغ به شکل 0 و 1 اختصاص داده می شود.

و از لاس فانکشن categorical\_crossentropy = مناسب برای طبقه بندی چند کلاسه

• تابع compile که شبکه عصبی را برای آموزش آماده میکند. این تابع سه پارامتر مهم را مشخص میکند: تابع هزینه، الگوریتم بهینهسازی و معیار ارزیابی. در این مثال، تابع هزینه categorical\_crossentropy است که میزان اختلاف بین خروجی مورد انتظار و خروجی واقعی را محاسبه میکند. الگوریتم بهینهسازی adam است که یک روش محبوب و کارآمد برای به روز رسانی وزنها و بایاسها است. معیار ارزیابی accuracy است که میزان تطابق بین برچسبهای واقعی و پیشبینی شده را نشان میدهد.

• تابع fit که شبکه عصبی را بر روی دادههای آموزشی آموزش میدهد. این تابع چندین پارامتر را دریافت میکند: دادههای ورودی و خروجی، اندازه دسته، تعداد دورهها و نسبت اعتبارسنجی. در این مثال، دادههای ورودی و خروجی x\_train و y\_train هستند که شامل تصاویر و برچسبهای دست نوشتهی اعداد فارسی هستند. اندازه دسته 128 است که نشان میدهد که شبکه عصبی 128 نمونه را در هر مرحله آموزش میبیند. تعداد دورهها 15 است که نشان میدهد که شبکه عصبی کل دادههای آموزشی را 15 بار میبیند. نسبت اعتبارسنجی 0.1 است که نشان میدهد که شبکه عصبی 10 درصد از دادههای آموزشی را برای ارزیابی عملکرد خود در طول آموزش استفاده میکند



توضیح کلی کد :

ین کد یک شبکه عصبی چند لایه پیشرو یا MLP را با استفاده از کتابخانه کراس در پایتون ایجاد میکند. این شبکه برای دستهبندی تصاویر دست نوشتهی اعداد فارسی با استفاده از مجموعه داده Hoda طراحی شده است. این کد شامل مراحل زیر است:

• دانلود مجموعه داده Hoda از گوگل درایو با استفاده از ابزار gdown و فازی مچینگ

• وارد کردن کتابخانههای keras، numpy و matplotlib.pyplot برای کار با شبکه عصبی، آرایهها و نمایش نمودارها

• وارد کردن ماژول dataset که توابعی برای بارگذاری و پیشپردازش دادههای Hoda را فراهم میکند

• تقسیم دادهها به دو بخش آموزشی و آزمونی و تبدیل برچسبها به بردارهای دودویی با استفاده از تابع to\_categorical

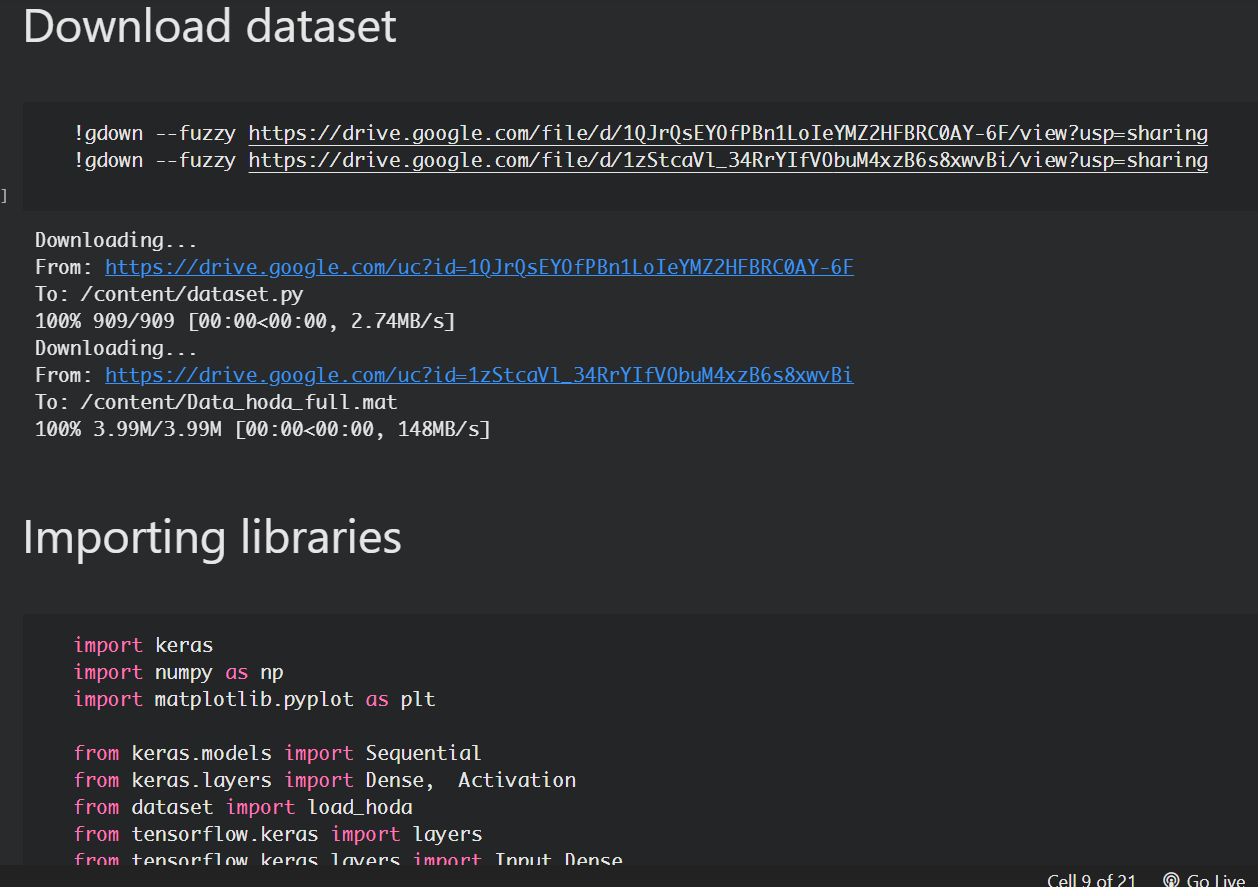
• چاپ اطلاعات مربوط به نوع، شکل و مقدار دادهها قبل و بعد از پیشپردازش با استفاده از تابع print\_data\_info

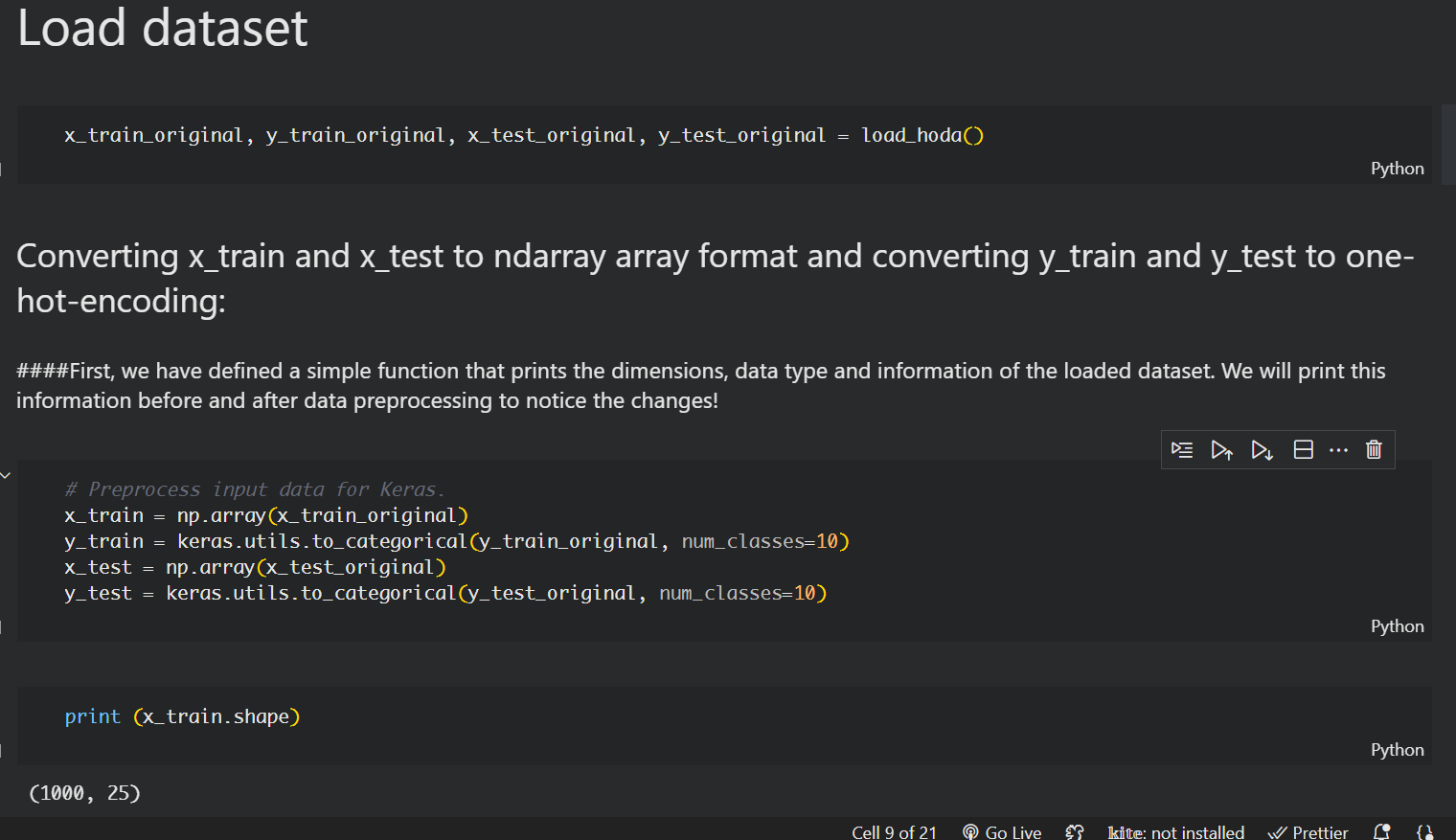
• تبدیل دادهها به نوع float32 و نرمال سازی آنها با تقسیم بر 255

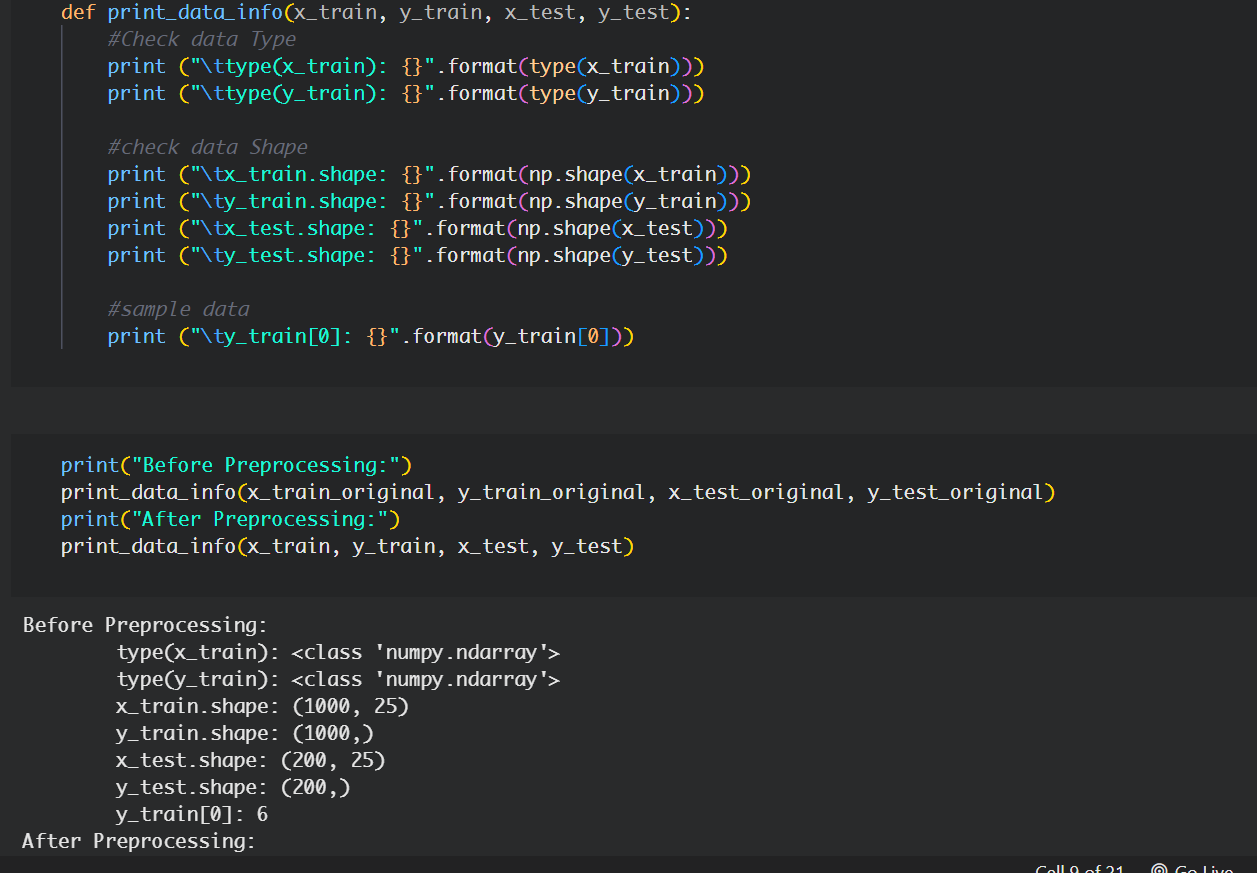
• تعریف معماری شبکه عصبی MLP که شامل یک لایه ورودی، یک لایه پنهان با 25 نورون و تابع فعالسازی relu، یک لایه حذف با نرخ 0.5 برای جلوگیری از بیشبرازش، و یک لایه خروجی با 10 نورون و تابع فعالسازی softmax است

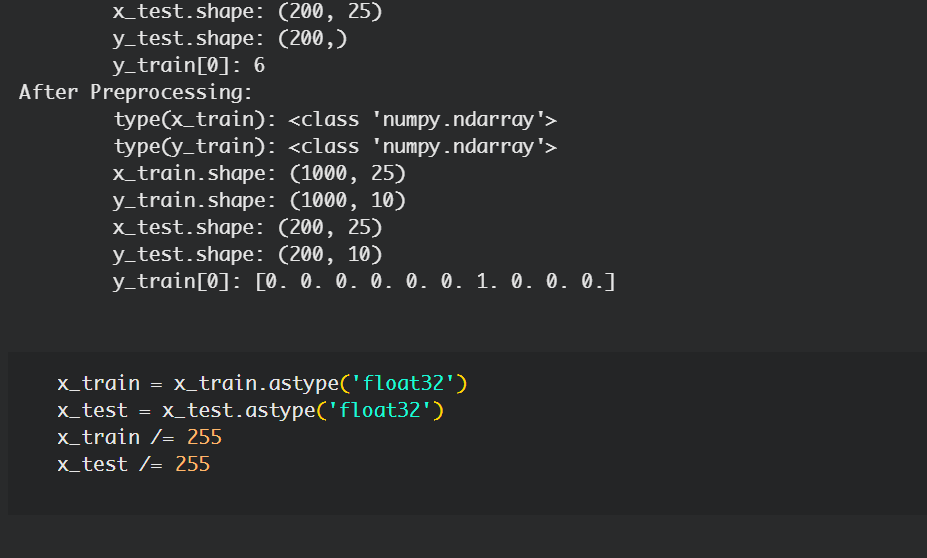
• تعیین اندازه دسته، تعداد دورهها، تابع هزینه، الگوریتم بهینهسازی و معیار ارزیابی برای شبکه عصبی

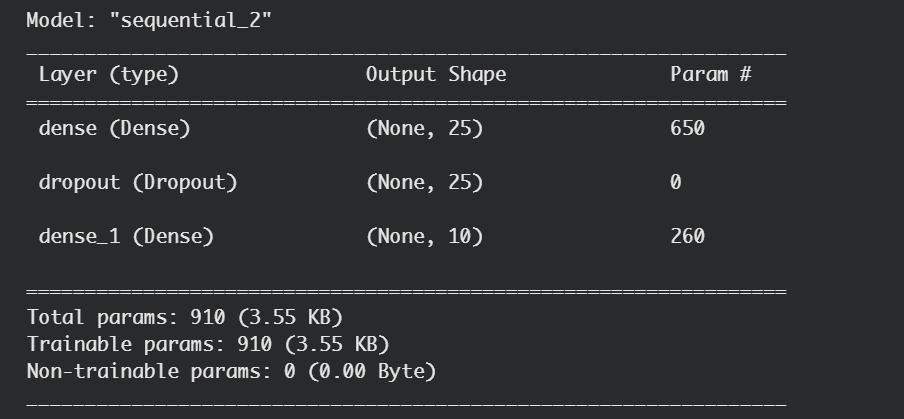
• آموزش شبکه عصبی بر روی دادههای آموزشی با استفاده از تابع fit و ارزیابی شبکه عصبی بر روی دادههای آزمونی با استفاده از تابع evaluate

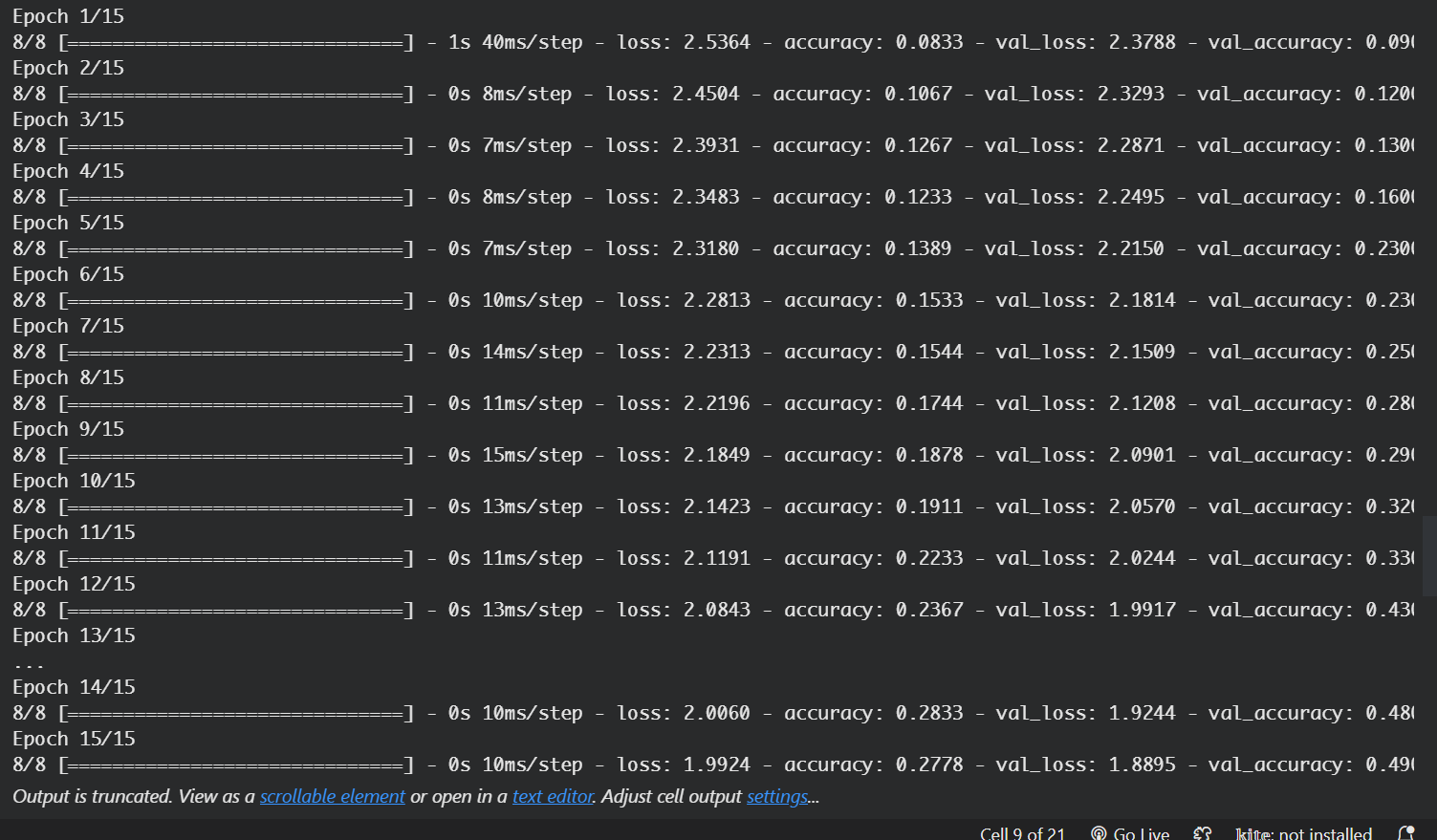


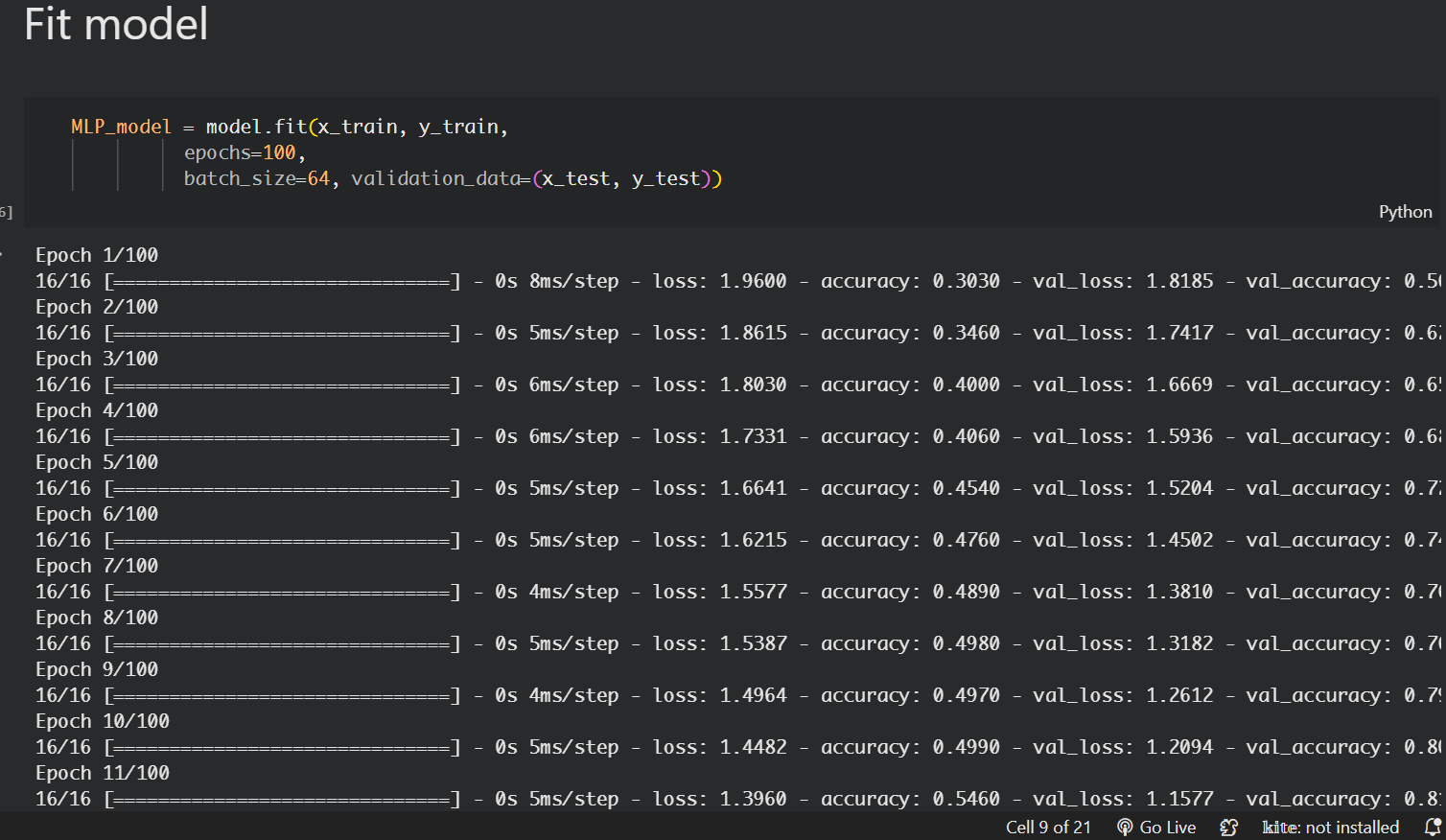


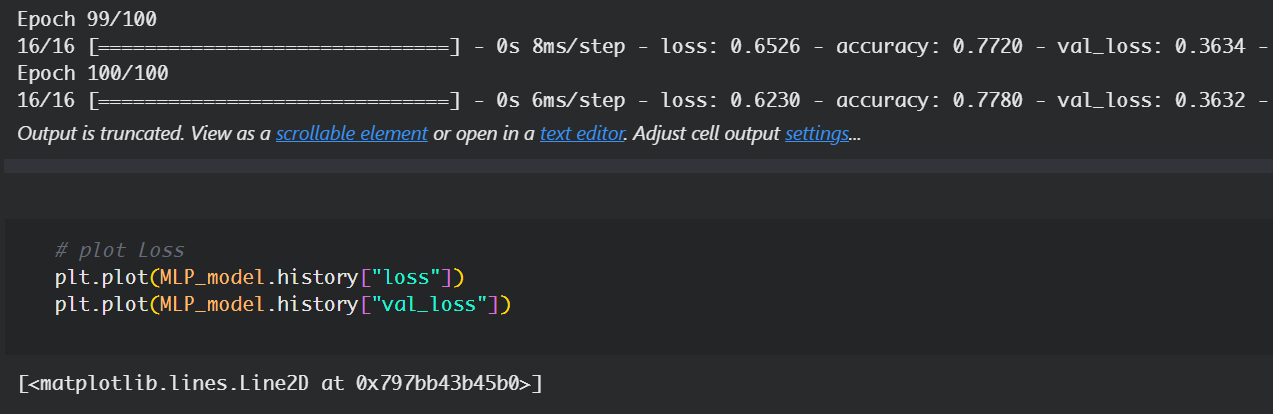












توضیح نمودار ها و تحلیل نتایج :

همانگونه که میبینیم با جلورفتن مراحل لاس کاهش یافته و دقت افزایش پیدا کرده است .

نمودار لاس نزولی بوده و نمودار دقت صعودی بوده و همانگونه که مشاهد می کنیم با جلو رفتن ایپوک ها و از ایپوک یک تا 100 :

نمودار loss

از 1.96 به 0.623

نمودار val\_loss

از 0.5 به 0.36

نمودار accuracy

از 0.3030به 07780

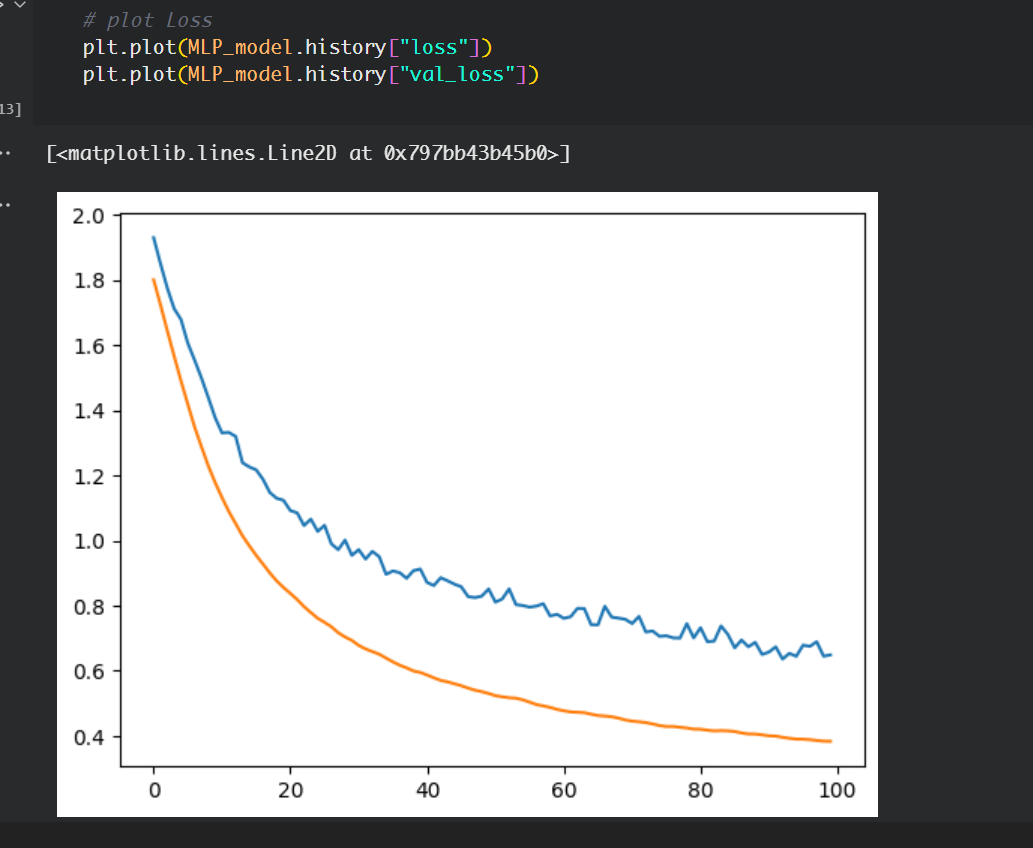
نمودار val\_accuac

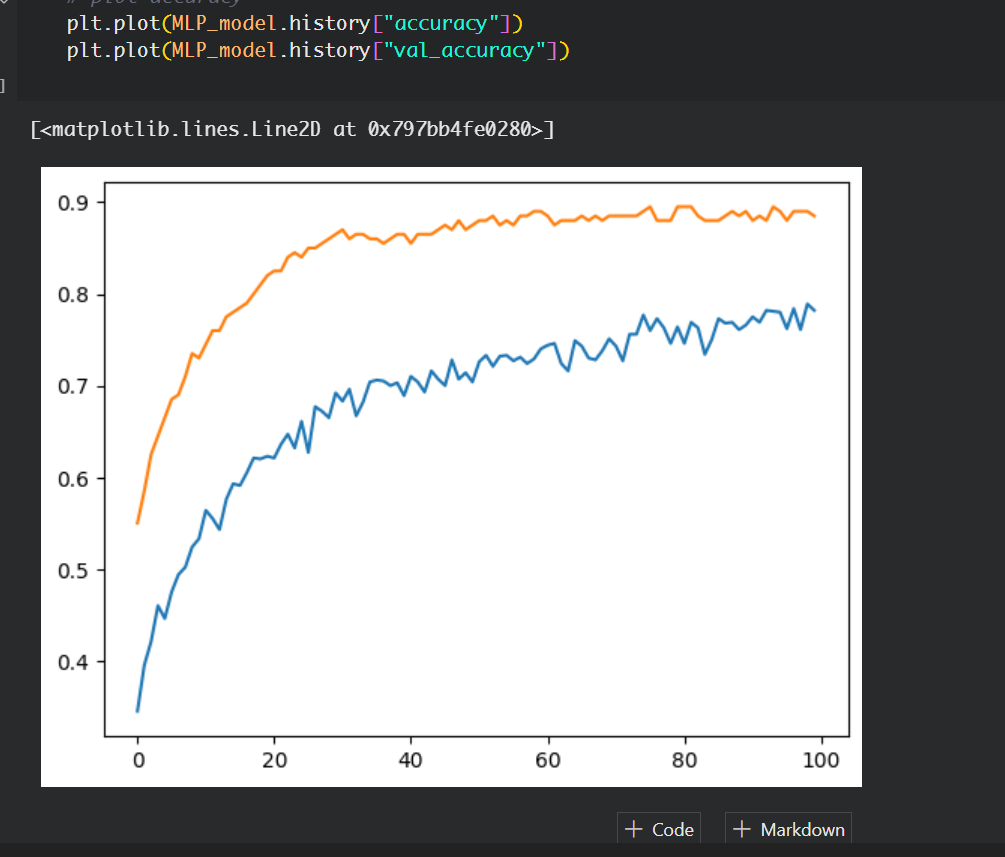
از 0.5650 به 0.9100 رسیدیم

دقت داده ی اموزشی و تست هر دو زیاد شده و لاس داده ی اموزشی و تست کم شده است.

چون از قسمت تست به خوبی نتیجه داده و درصد بالایی داریم 90 درصد مشخص است که under فیت نشده است و توانسته شبکه لرن داشته باشد هرچند داشتن لرنینگ ریت و تنظیم ابر پارامتر ها و یا استفاده از شبکه های عمیق تر و پیجیده تر میتواند این مسئله طبقه بندی را نیز حل کند و دقت ما را به یک و لاس را به صفر برساند اما اختلاف کم نمودار ها نشان از عملکرد درست این شبکه نشان می دهد و قسمت فیت شدن شبکه به خوبی فیت شده است.

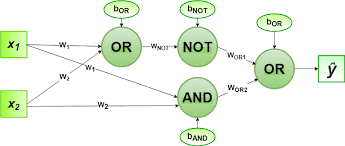
لاس ماهم به حدود 40 درصد رسیده و به طور اکید نزولی بوده است و اورفیت هم نشده و تا حدود خوبی توانسته که تعمیمم بدهد و generalization صورت بگیرد(با افزایش داده ها در دیتاست و یا دراپ اوت این مقدار بهبود نیز میابد) چون مدل علاوه بر روی داده های اموزشی روی داده های تست هم خوب عمل کرده است .البته که نمودار ها کمی باهم اختلاف دارند ولی قابل چشم پوشی است و یک فلوی یکسان را پیش گرفته اند.

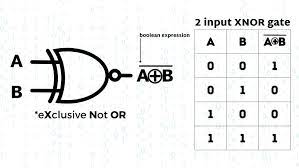




5-با کمک کتابخانه NumPy یک پرسپترون چند لایه آموزش دهید که تابع XNOR را یاد بگیرد.

پاسخ 5 )

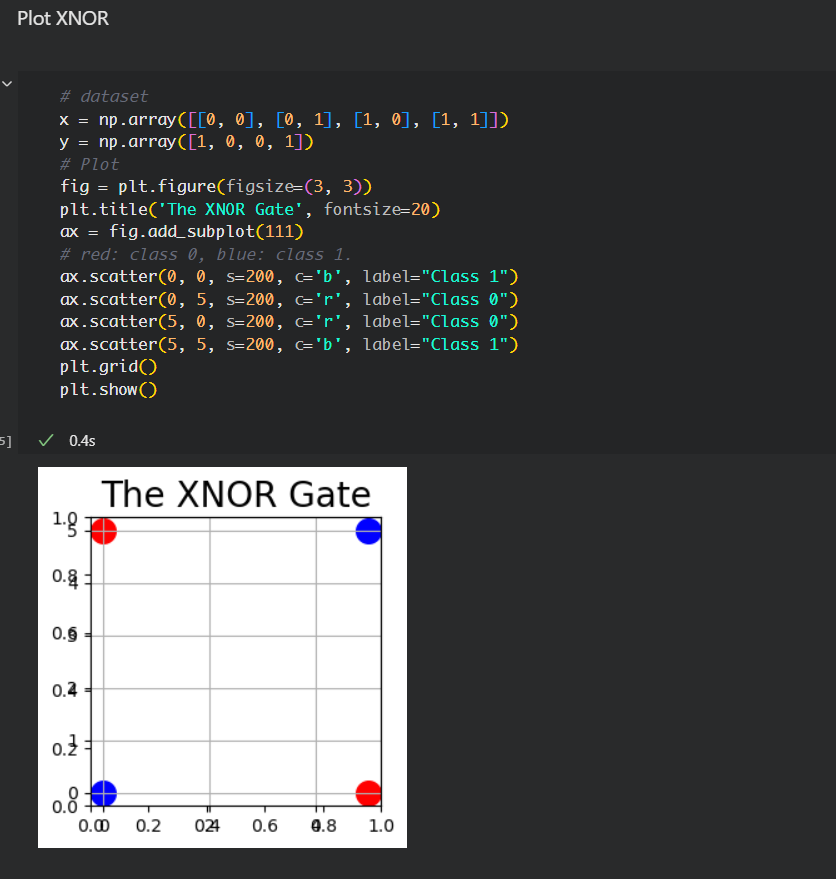




در ابتدا توابعی پایه ای که برای حل این سوال نیاز داریم را تعریف می کنیم. تابع فعال سازی استفاده شده در این مثال، sigmoid بوده که یک تابع با همان نام در کدمان تعریف می کنیم. همچنین تابع محاسبه ضررمان، error entropy cross می باشد که در کدمان با نام CEE تعریف شده است. سپس چون برای فاز propagation-back به مشتق تابع ارور و سیگموید نیاز داریم، تابعی برای محاسبه این دو نیز می نویسیم که با نام dCEE و dsigmoid در کدمان وجود دارند. توابعیͬ که تا اینجا توضیح داده شد، در قطعه کد زیر آورده شدە اند.

سپس در این بخش، یک دیتاست تعریف کرده و آن را plot می کنیم. می دانیم که گیت XNOR در صورت غیر شبیه بودن بیت همه ورودی ها، ۰ و در غیر این صورت ۱ می شود. کد زده شده در این بخش به همراه نتیجه حاصله در شکل زیر آورده شده است.

اول در اینجا یک دیتاست مطابق با لاجیک XNor درست کردم و 0و0 و 1و1 را لیبل 1 می گیرند و بقیه صفر حالا پلات ان را میبینم در مختصات ها .



توضیحات لازم در کد اورده شده است اما ابتدا کتابخانه های مورد نیاز را ایمچورت می کنیم و دو وزن رندوم را اساین می کنیم

سپس مقدار سیگموید و مشتق ان را محاسبه می کنیم که برای اصلاح وزن ها و بک پروپگیشن استفاده می شود.

  Creating function for running and testing NN after training

       Feed forward through three layers in NN

         Results are returned in normalized form in appropriate dimensions

    Feeding our NN with training set and calculating output

             Feed forward through three layers in NN

             With 'numpy' library and function 'dot' we multiply matrices with values in layers to appropriate weights

             Results are returned in normalized form in appropriate dimensions

             Creating function for training the NN

             Using Backpropagation for calculating values to correct weights

             Calculating an error for output layer (Layer 2) which is the difference between desired output and obtained output

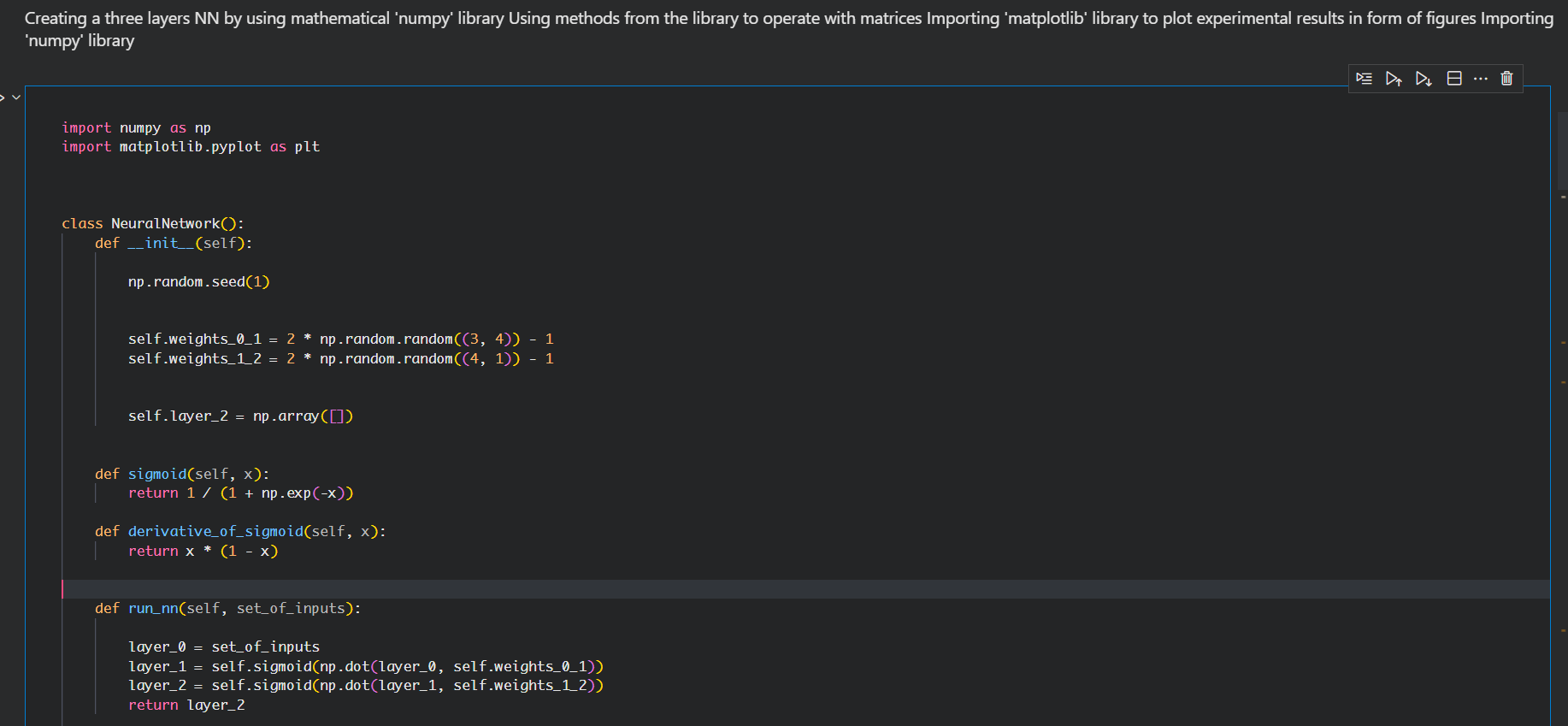
             We subtract matrix 4x1 of received outputs from matrix 4x1 of desired outputs

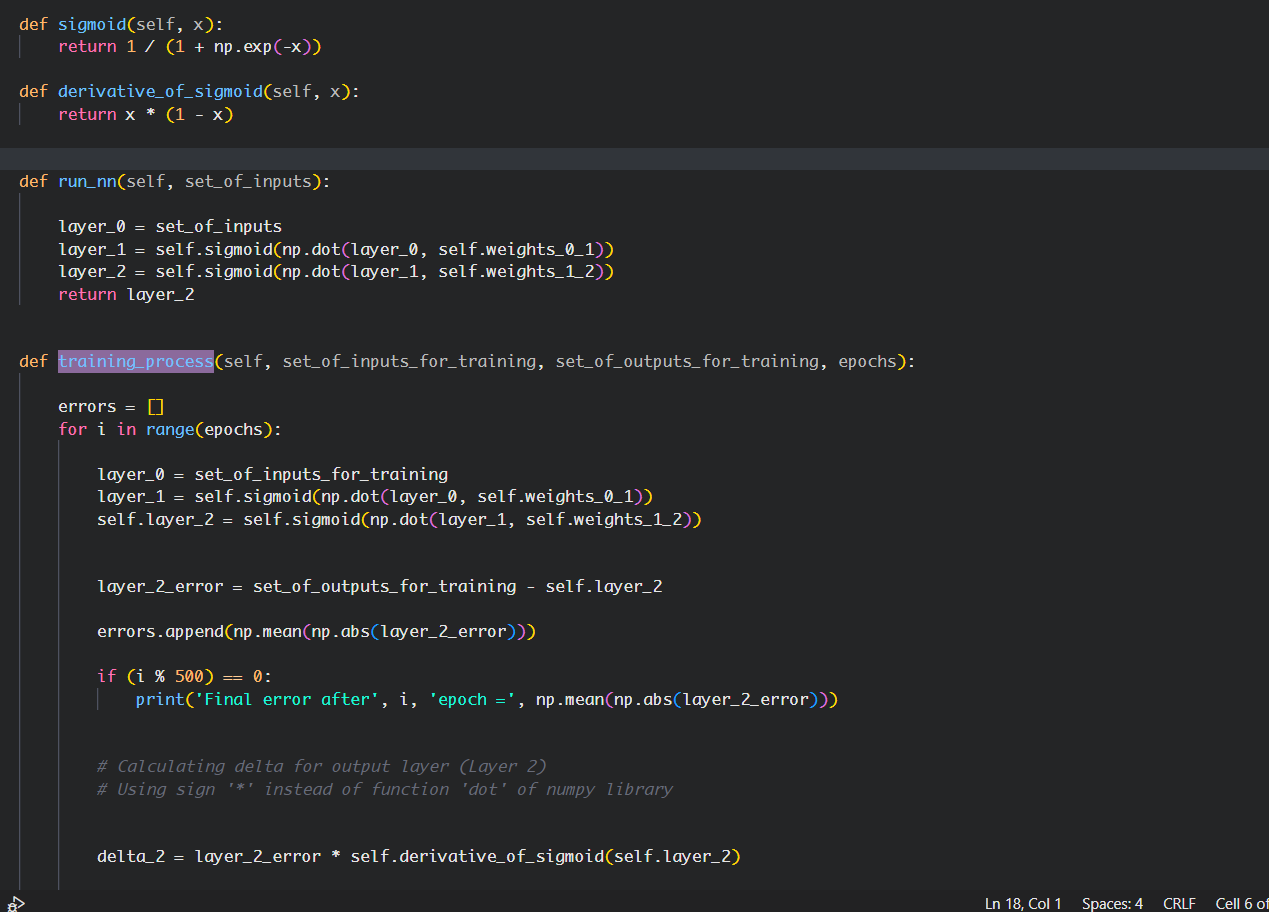
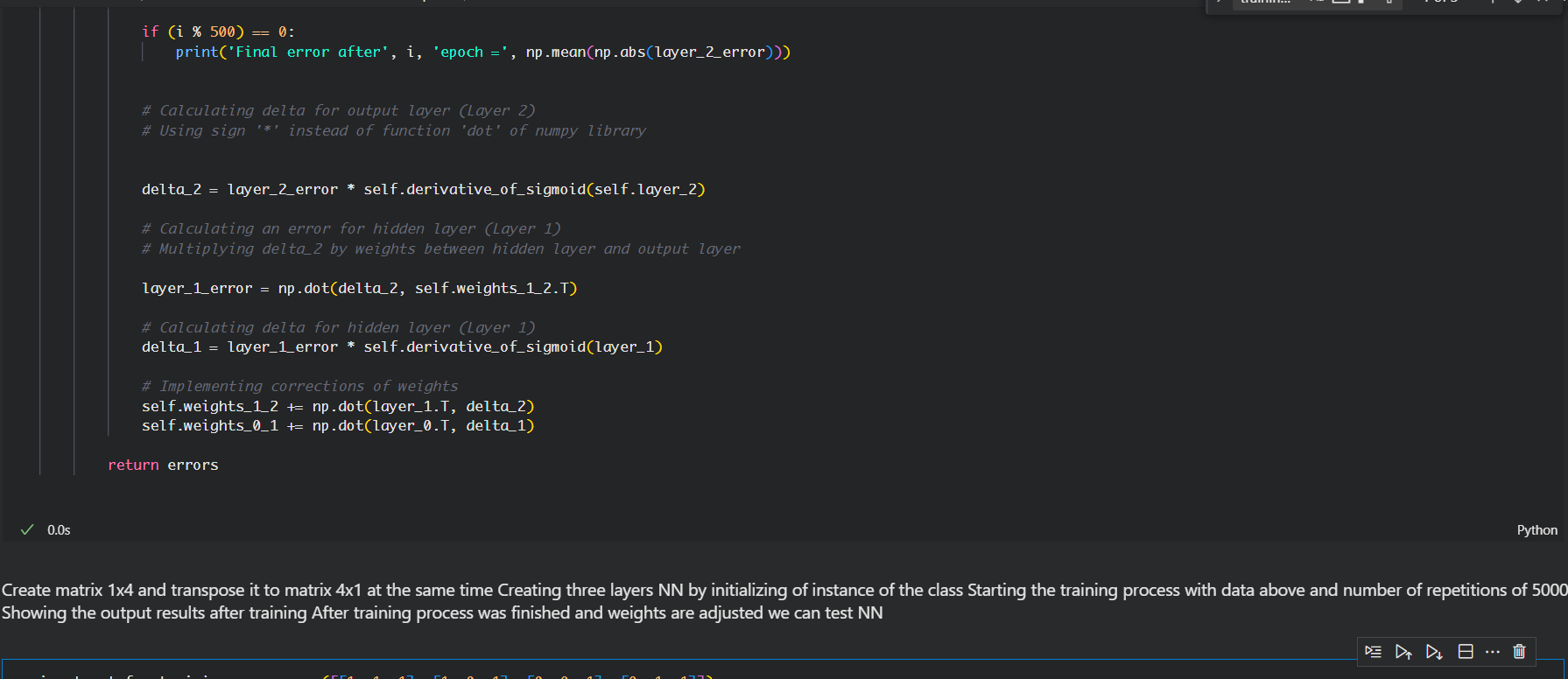
      Showing the error each 500 iterations to track the improvements

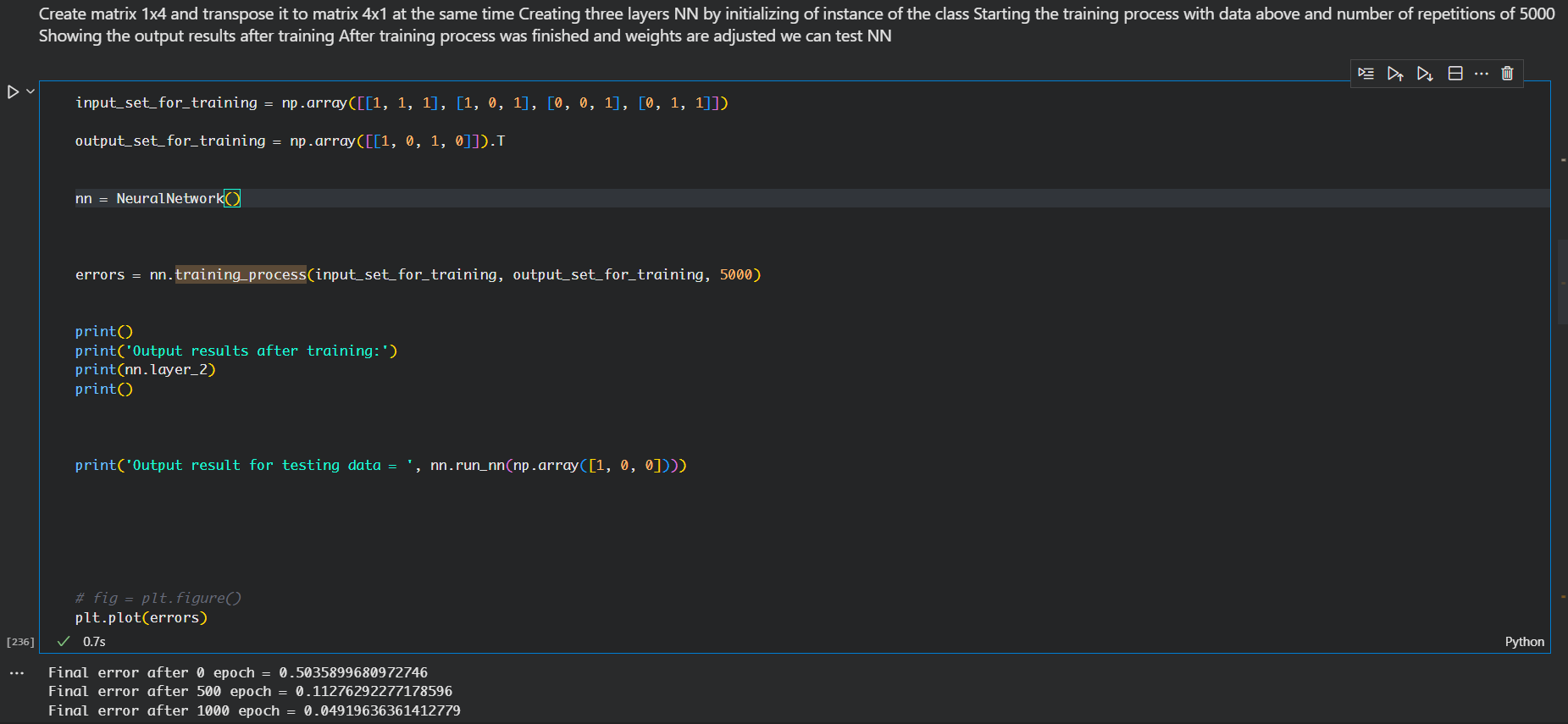
و بعد دلتا را برای لایه ی خروجی و اررو را محاسبه می کنیم و وزن ها را اصلاح می کنیم

سپس مدل با مجموعه ی اموزسی ما که لیبل های مشخص دارد تا 5000 مرحله اموزش می بیند و بعد روی داده ی تست ان را تست می کنیم .

نتیجه :



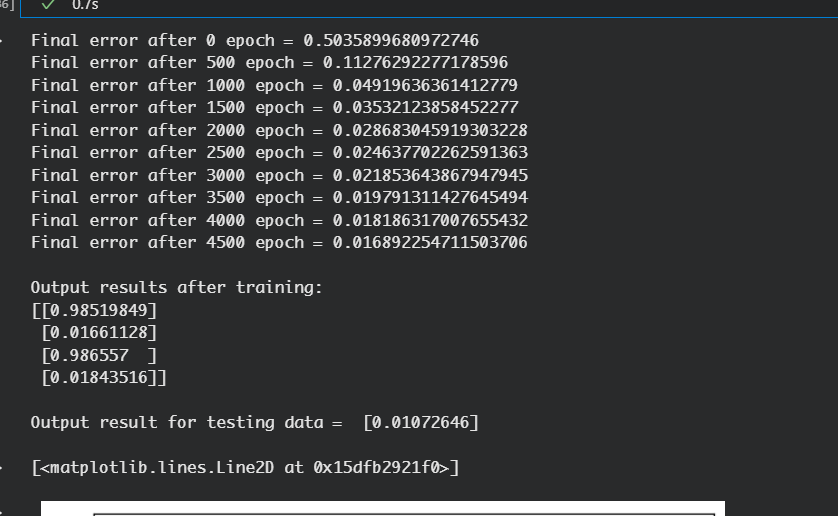
 

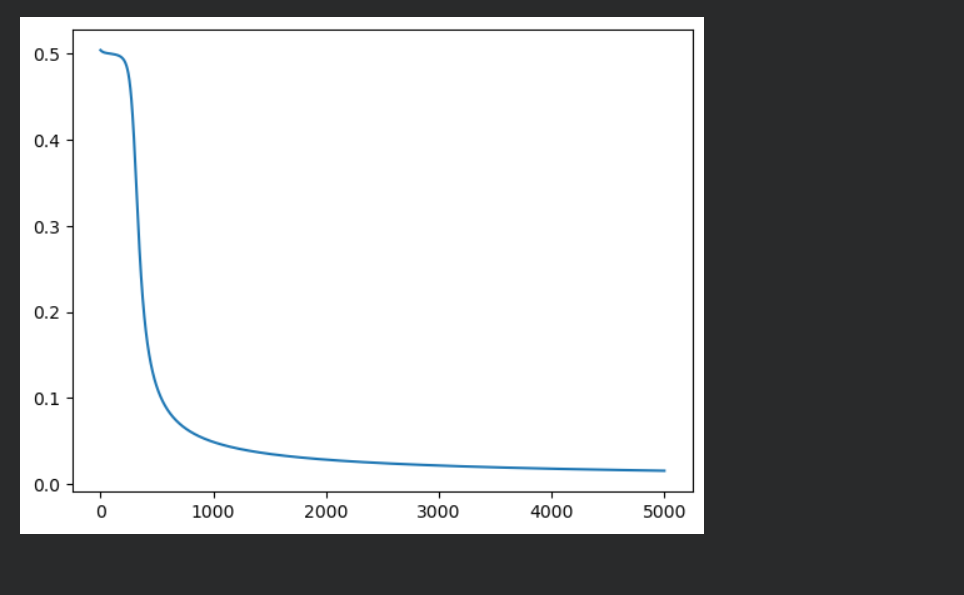


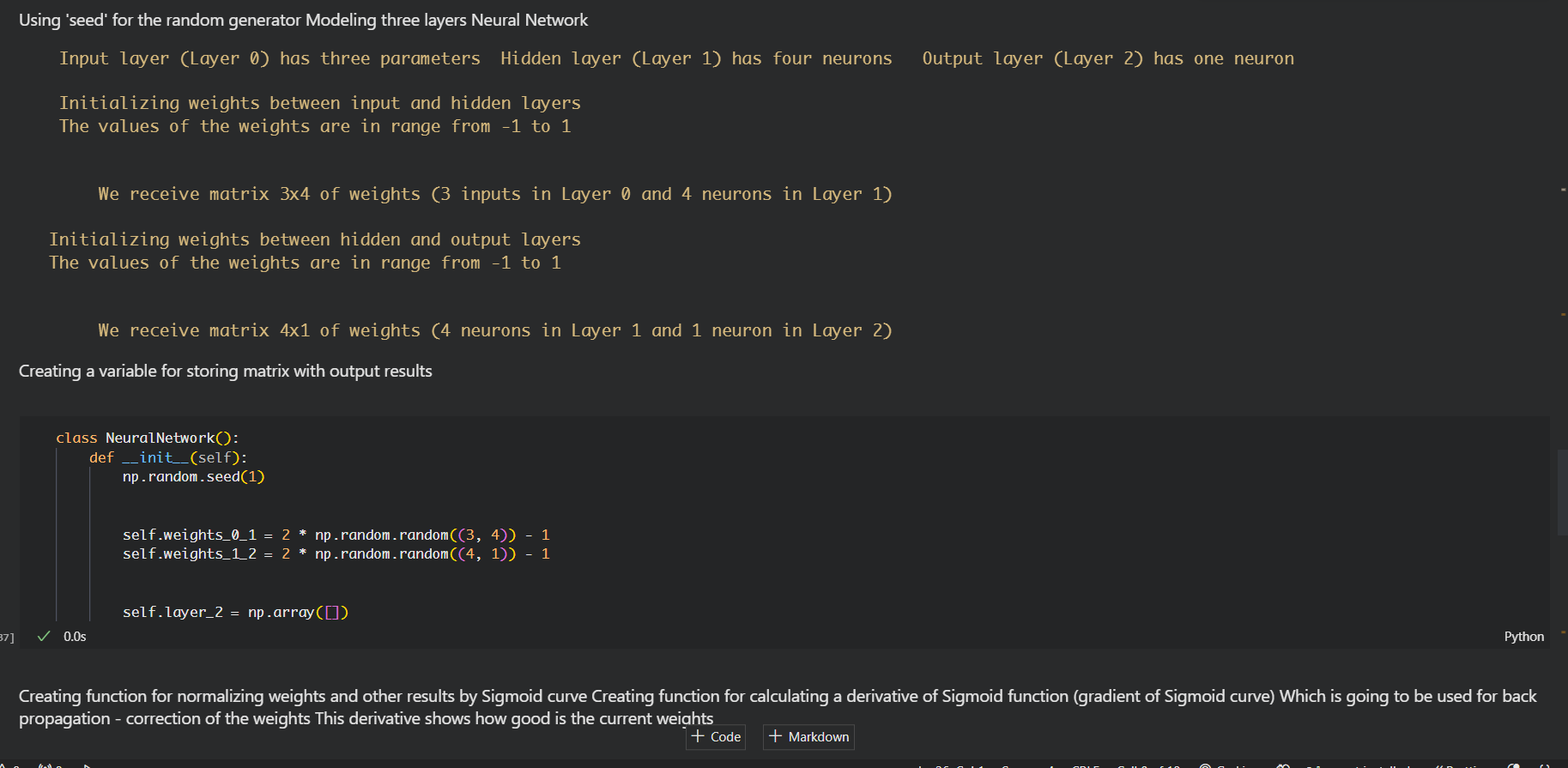
نتیجه

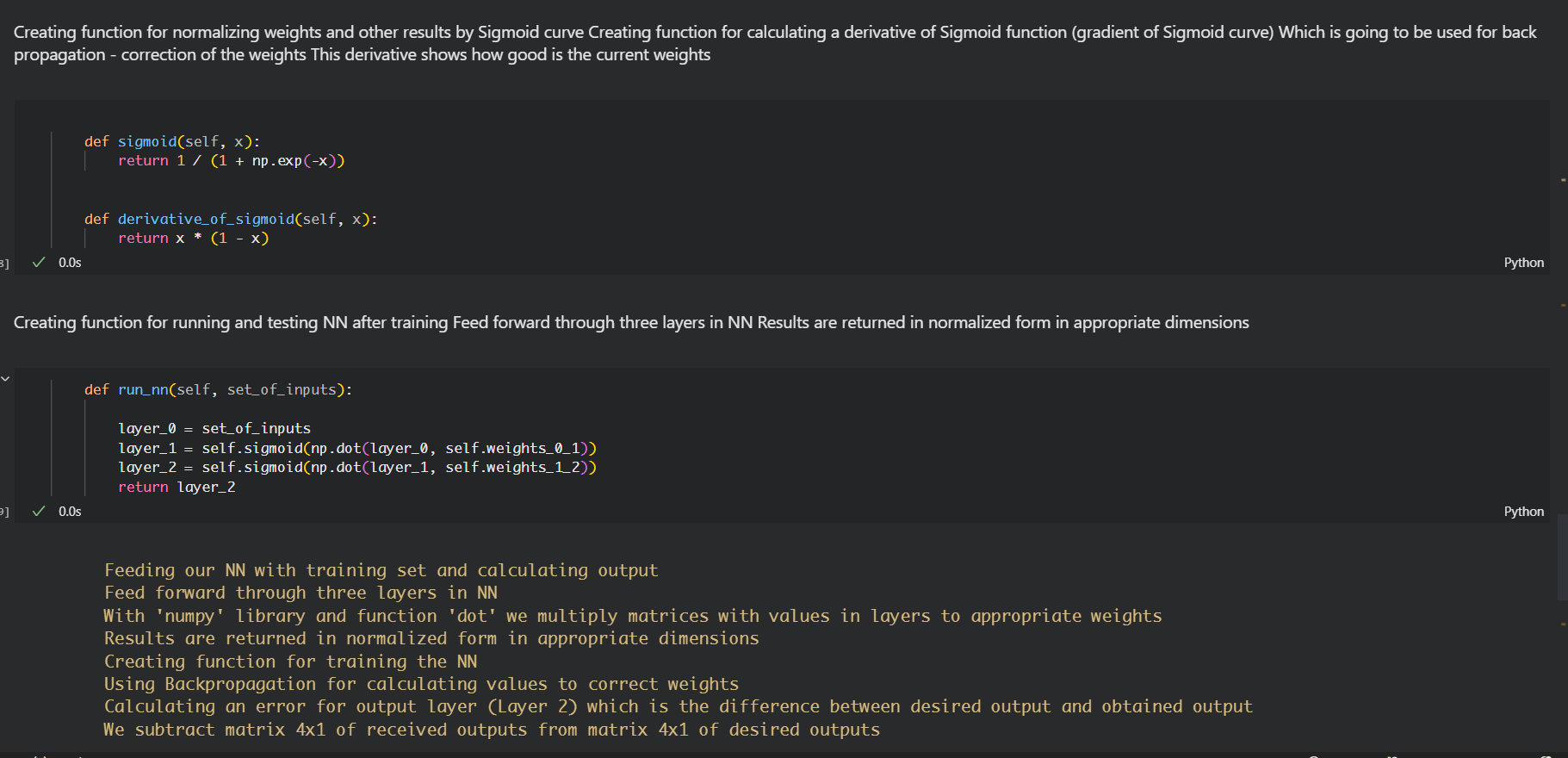
مشاهده می کنیم مدل شبکه ی عصبی ما به خوبی ترین شده و فیت شده در طول مرحله ها و توانسته داده های اموزشی را به خوبی یاد بگیردو اررو ها از 0.5 شروع شده و به 0.01 رسیده است. و تعمیم دهی شبکه نیز خوب بوده و اورفیت نشده و به مشکلی بر نخوردیم و می توان حتی با دراپ اوت که در سوال های دیگر استفاده کردم یا جمع کردن دیتاهای بیش تر یا منظم سازی ها عملکرد شبکه را بهتر نیز کرد .

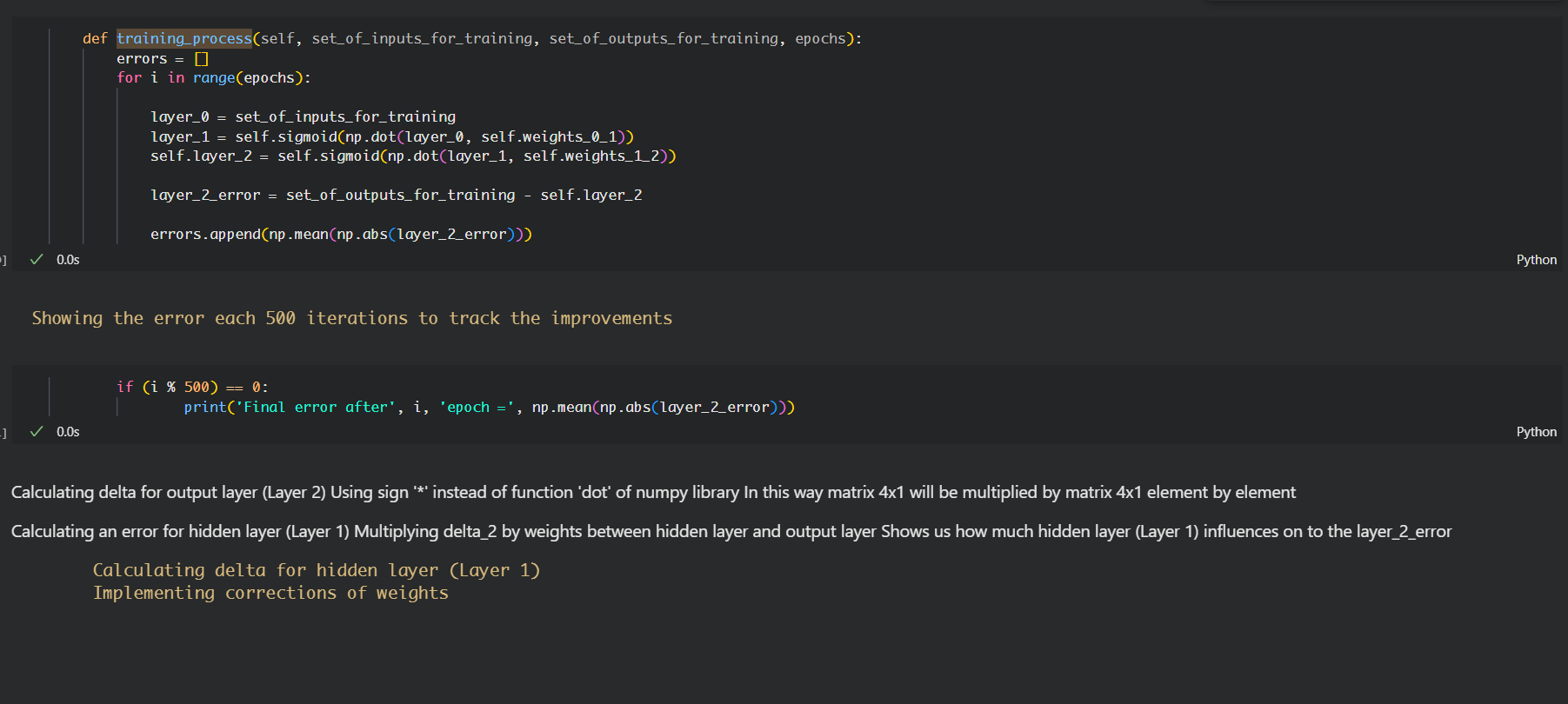
و دقت هم در طی مرحله ها بالاتر رفته . همانگونه که میبینم دو دیتایی که مطابق گیت نند جواب یک را به ما می دهند مقدار بیش تر و دو مقدار دیگر درصد پایین دارند همچنین روی دیتای تست مان نیز جواب درست را مشاهده می کنیم











6-یک مدل MLP روی دیتاست MNIST طراحی کنید به طوری که به دقت حداقل 95 درصد دست یابد. در گزارش خود توضیح دهید تعداد لایه ها و نورون ها را بر چه اساس انتخاب کردید. در پایان کار با استفاده از matplotlib نمودار توابع Loss و Accuracy را رسم کنید و در گزارش خود بیاورید.

پاسخ 6 )

تعداد لایه ها :

در ابتدا تعداد کلاس ها و سایز عکسهای ورودی را تعریف میکنیم. سپس دیتای آموزشی و تست را لود میکنیم و تقسیم بر 255 میکنیم تا به رنج 0 تا یک اسکیل شوند.

سپس لیبل داده های تست و ترین را categorical\_to میکنیم تا به صورت یک بردار به تعداد کلاس های ما در بیایند و مقدار هر خانه از بردار احتمال مربوط بودن نمونه به آن کلاس میشود که بین صفر تا یک است.

در مرحله ی بعد مدل خود را به صورت sequential تعریف میکنیم. در لایه های اولیه و میانی از تابع فعالسازی relu و در لایه ی آخر از تابع فعالسازی softmax استفاده میکنیم. زیرا مسئله ی ما کلاس بندی چند کلاسه است. لایه ها نیز به صورت کامل متصل هستند. برای بهبود مدل خود از dropout نیز استفاده کردهایم.

• یک لایه حذف یا Dropout با نرخ 0.5 که به منظور جلوگیری از بیشبرازش یا overfitting به کار میرود. این لایه به طور تصادفی برخی از نورونهای لایه قبلی را حذف میکند و باعث میشود مدل به ویژگیهای مهمتر توجه کند.

مدل خود را کامپایل و روی دادههای آموزشی اجرا میکنیم. بدلیل نوع مسئله از تابع ضرر crossentropy\_categorical و از تابع بهینه ساز adam استفاده می کنیم. سایز batchها را 128 و تعداد ایپاک ها را 15 تعریف میکنیم. سپس نمودار دقت و خطا را رسم میکنیم.

همانطور که میبینیم به دقت بالای %98 برای دادههای آموزشی و validation رسیده و همچنین خطای کمتر از %0.3 را داریم

دو مدل پیاده سازی مدل :

روش اول پیاده سازی مدل

-همچنبن قسمت سکوانشال مدل را بر اساس دانش درس بینایی ماشین کد زدم و از لایه های Conv2D و maxpool استفاده کردم اما می توانستم از لایه ای فولی کانکتد و دنس مانند سوال دیتاست هدا استفاده کرد ولی برای دقت بالا تر از چند لایه متناوب استفاده کردم که اگر پیچیده تر کرد شبکه را و هزینه بالا می رود ولی دقت بسیار بالا و لاس بسیار کم در ایپوک های نهایی خود مشاهده می کنیم .98 درصد دقت در نهایت خواهیم داشت .

7 لایه :

یک لایه ورودی : تک کاناله شکل داده 28\*28=784

یک لایه خروجی : یکی از 10 کلاس 10 نورون

5 لایه نهان

• یک لایه ورودی که شکل داده های ورودی را به عنوان پارامتر می گیرد. این لایه هیچ محاسباتی را انجام نمی دهد، اما شکل ورودی را برای بقیه شبکه تعریف می کند.

• یک لایه کانولوشن که 32 فیلتر به اندازه 3 در 3 را با عملکرد فعال سازی relu روی ورودی اعمال می کند. این لایه با کشیدن فیلترها روی ورودی و اعمال تابع فعال سازی روی نتیجه، ویژگی ها را از ورودی استخراج می کند.

• یک لایه ادغام حداکثر که با گرفتن حداکثر مقدار در هر پنجره 2 در 2، اندازه ورودی را کاهش می دهد. این لایه به کاهش هزینه های محاسباتی و جلوگیری از برازش بیش از حد با کاهش نمونه ورودی کمک می کند.

• یک لایه کانولوشنال دیگر که 64 فیلتر در اندازه 3 در 3 را با عملکرد فعال سازی relu روی ورودی اعمال می کند. این لایه با اعمال فیلترهای بیشتر و افزایش عمق شبکه، ویژگی های بیشتری را از ورودی استخراج می کند.

• یک لایه max pooling دیگر که با گرفتن حداکثر مقدار در هر پنجره 2 در 2، اندازه ورودی را کاهش می دهد. این لایه همان عملکرد لایه max pooling قبلی را انجام می دهد.

• یک لایه مسطح که ورودی را به یک بردار یک بعدی تغییر شکل می دهد. این لایه با صاف کردن ورودی چند بعدی در یک بردار، ورودی را برای لایه متراکم نهایی آماده می کند.

• یک لایه حذفی که به طور تصادفی 50 درصد از واحدهای ورودی را حذف می کند. این لایه با افزودن مقداری نویز و تنظیم به شبکه به جلوگیری از برازش بیش از حد کمک می کند.

• یک لایه متراکم که 10 واحد با عملکرد فعال سازی softmax خروجی می دهد. این لایه وظیفه طبقه بندی نهایی را با نگاشت بردار ورودی به 10 واحد خروجی انجام می دهد که هر یک نشان دهنده یک کلاس است. تابع فعال سازی softmax تضمین می کند که واحدهای خروجی تا 1 جمع می شوند و می توان آنها را به عنوان احتمال تفسیر کرد.

• یک لایه ورودی که تعداد نورونهای آن برابر با شکل دادههای ورودی است. در این مثال، شکل دادههای ورودی 28 در 28 در 1 است که نشان میدهد که تصاویر سیاه و سفید با اندازه 28 در 28 پیکسل هستند. بنابراین، تعداد نورونهای لایه ورودی 784 است.

• یک لایه پیچشی که تعداد نورونهای آن برابر با تعداد فیلترهای آن است. در این مثال، تعداد فیلترها 32 است که هر کدام از اندازه 3 در 3 هستند. بنابراین، تعداد نورونهای لایه پیچشی اول 32 است.

• یک لایه ادغام حداکثر که تعداد نورونهای آن برابر با تعداد نورونهای لایه قبلی است. در این مثال، تعداد نورونهای لایه قبلی 32 است. بنابراین، تعداد نورونهای لایه ادغام حداکثر اول 32 است.

• یک لایه پیچشی دیگر که تعداد نورونهای آن برابر با تعداد فیلترهای آن است. در این مثال، تعداد فیلترها 64 است که هر کدام از اندازه 3 در 3 هستند. بنابراین، تعداد نورونهای لایه پیچشی دوم 64 است.

• یک لایه ادغام حداکثر دیگر که تعداد نورونهای آن برابر با تعداد نورونهای لایه قبلی است. در این مثال، تعداد نورونهای لایه قبلی 64 است. بنابراین، تعداد نورونهای لایه ادغام حداکثر دوم 64 است.

• یک لایه صافی که تعداد نورونهای آن برابر با تعداد عناصر لایه قبلی است. در این مثال، تعداد عناصر لایه قبلی 64 در 4 در 4 است که برابر با 1024 است. بنابراین، تعداد نورونهای لایه صافی 1024 است.

• یک لایه خروجی که تعداد نورونهای آن برابر با تعداد کلاسهای مورد نظر است. در این مثال، تعداد کلاسها 10 است که نشان میدهد که ده عدد انگلیسی هستند. بنابراین، تعداد نورونهای لایه خروجی 10 است.

دلیل استفاده از رلو

ایه های کانولوشن و یادگیری عمیق: محبوب ترین تابع فعال سازی برای آموزش لایه های کانولوشن و مدل های یادگیری عمیق است. سادگی محاسباتی: تابع یکسو کننده برای پیاده سازی بی اهمیت است و فقط به تابع max() نیاز دارد. پراکندگی نمایشی: یک مزیت مهم تابع یکسو کننده این است که قادر به خروجی یک مقدار صفر واقعی است. رفتار خطی: بهینه سازی یک شبکه عصبی زمانی که رفتار آن خطی یا نزدیک به خطی باشد آسان تر است

دلیل استفاده از softmax

تابع فعال‌سازی softmax با آسان‌تر کردن تفسیر خروجی‌های شبکه عصبی، این کار را برای شما ساده می‌کند! تابع فعال‌سازی softmax خروجی‌های خام شبکه عصبی را به بردار احتمالات تبدیل می‌کند، که اساساً یک توزیع احتمال روی کلاس‌های ورودی است.

دلیل استفاده از آدام

تایج بهینه‌ساز Adam عموماً بهتر از هر الگوریتم بهینه‌سازی دیگری است، زمان محاسبات سریع‌تری دارد و به پارامترهای کمتری برای تنظیم نیاز دارد. به همین دلیل، Adam به عنوان بهینه ساز پیش فرض برای اکثر برنامه ها توصیه می شود.

دلیل استفاده از categorical\_crossentropy

نتروپی متقاطع طبقه ای برای طبقه بندی چند طبقه استفاده می شود. آنتروپی متقاطع با واگرایی KL متفاوت است اما می توان آن را با استفاده از واگرایی KL محاسبه کرد. همچنین با از دست دادن گزارش متفاوت است، اما زمانی که به عنوان یک تابع از دست دادن یادگیری ماشین استفاده می شود، همان مقدار را محاسبه می کند.

روش دوم:

: 7 لایه دارد با استفاده از mlp ساده و بدون کانولوشنی ها ساخته شده

تعداد لایه ها و نورون ها با توجه به عوامل مختلف تایین می شود :

• نوع مسئله و دادههایی که قرار است حل شوند. برای مثال، برای دستهبندی تصاویر، معمولا از شبکههای عصبی پیچشی استفاده میشود که دارای چندین لایه پیچشی و ادغام حداکثر هستند. برای پیشبینی دنبالههای زمانی، معمولا از شبکههای عصبی بازگشتی استفاده میشود که دارای چندین لایه بازگشتی و حافظه کوتاه مدت بلند هستند.

• پیچیدگی و تنوع الگوهای موجود در دادهها. برای مثال، برای شناسایی چهره، معمولا از شبکههای عصبی عمیق استفاده میشود که دارای چندین لایه پنهان هستند. برای شناسایی دستنوشتهها، معمولا از شبکههای عصبی سادهتر استفاده میشود که دارای یک یا دو لایه پنهان هستند.

• منابع محاسباتی و حافظهای که در دسترس هستند. برای مثال، برای آموزش شبکههای عصبی بسیار عمیق، معمولا از کارتهای گرافیکی قوی و حافظههای بزرگ استفاده میشود. برای آموزش شبکههای عصبی کمعمق، معمولا از پردازندههای مرکزی و حافظههای کمتر استفاده میشود.

• تجربه و دانش فردی که شبکه عصبی را طراحی میکند. برای مثال، برای انتخاب بهترین تابع فعالسازی، تعداد نورونها، نرخ یادگیری و دیگر پارامترهای شبکه عصبی، معمولا از دانش نظری و تجربی در زمینهی یادگیری ماشین و شبکههای عصبی استفاده میشود.

اینپوت ورودی شکل 28\*28

خروجی یکی از ده نورون کلاس

لایه های پنهان :5 لایه و تعداد نورون دو لایه اخر 10 و بقیه لایه ها 128

dense\_18 (Dense) (None, 128) 100480

flatten\_6 (Flatten) (None, 128) 0

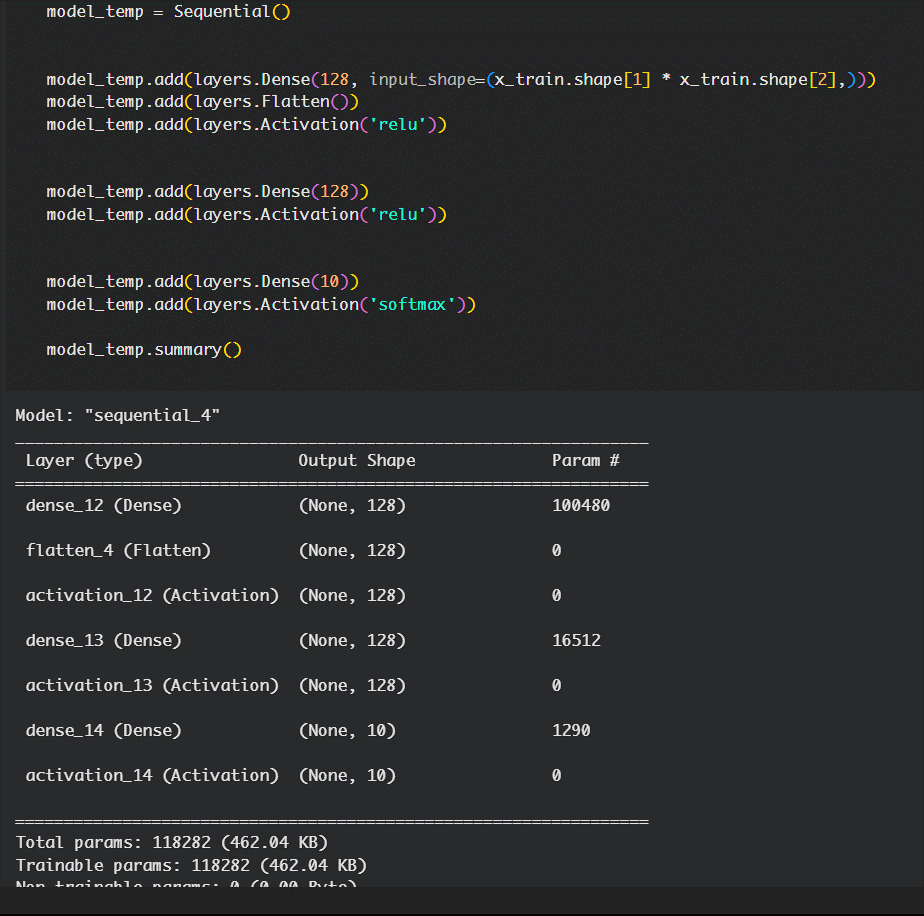
activation\_18 (Activation) (None, 128) 0

dense\_19 (Dense) (None, 128) 16512

activation\_19 (Activation) (None, 128) 0

dense\_20 (Dense) (None, 10) 1290

activation\_20 (Activation) (None, 10) 0



*# Input Layer*

model\_temp.add(layers.Dense(128, input\_shape=(x\_train.shape[1] \* x\_train.shape[2],)))

model\_temp.add(layers.Activation('relu'))

*# Hidden Layer*

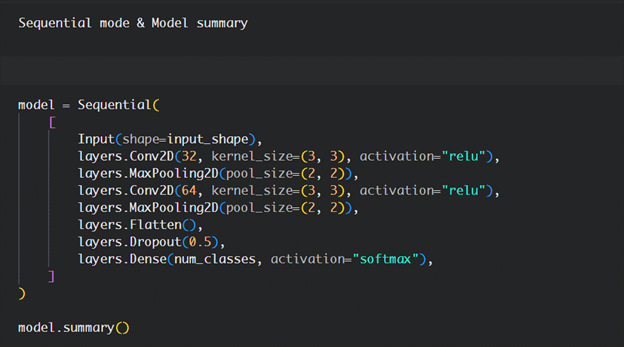
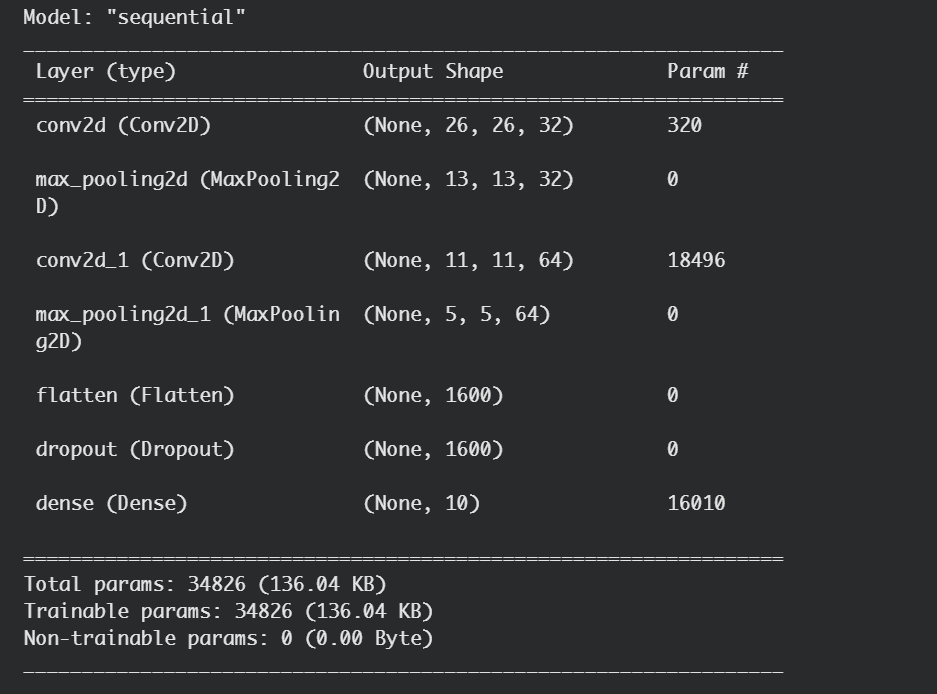
model\_temp.add(layers.Dense(128))

model\_temp.add(layers.Activation('relu'))

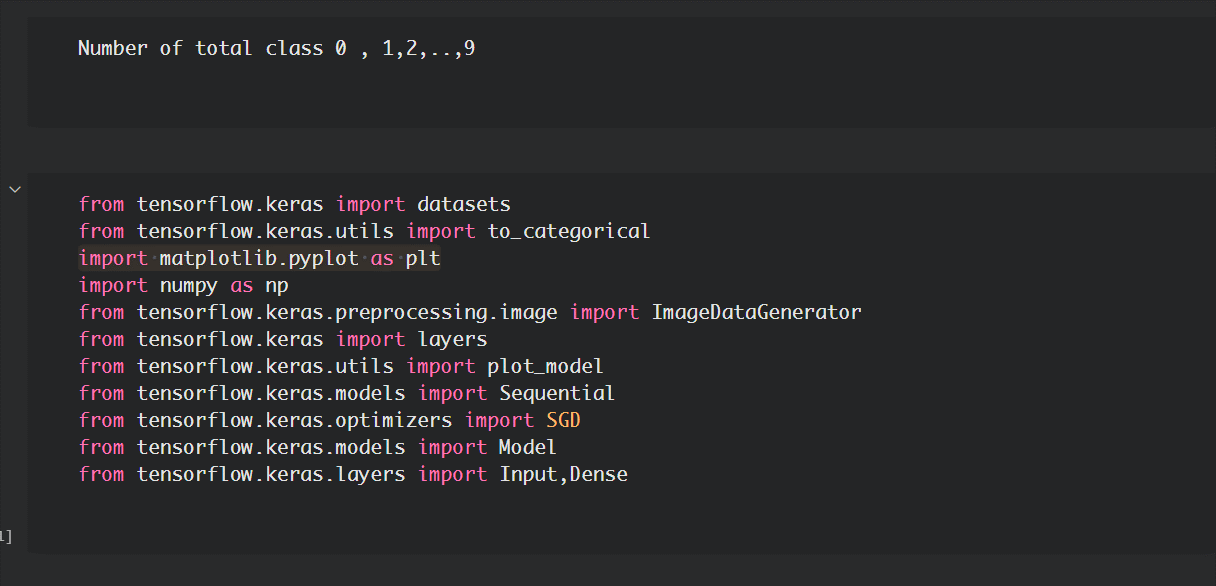
*# Output Layer*

model\_temp.add(layers.Dense(10))

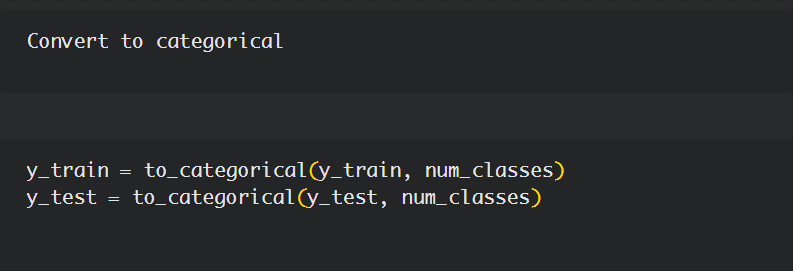
model\_temp.add(layers.Activation('softmax'))

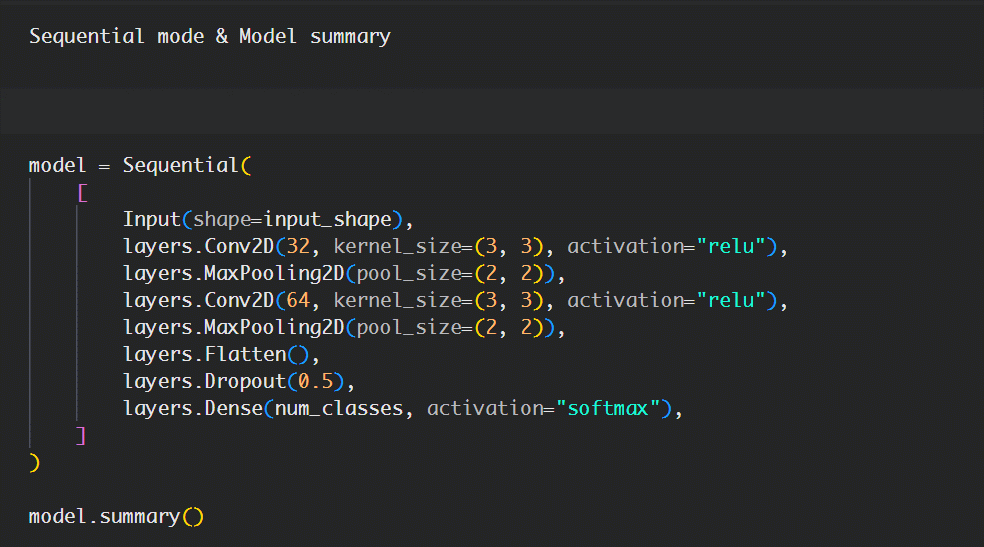


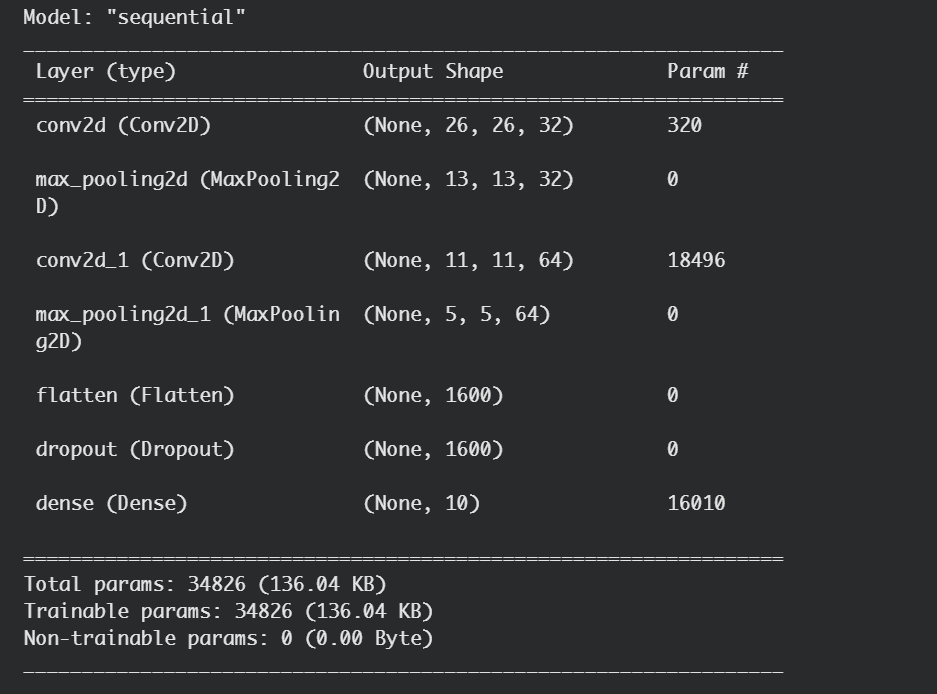
.

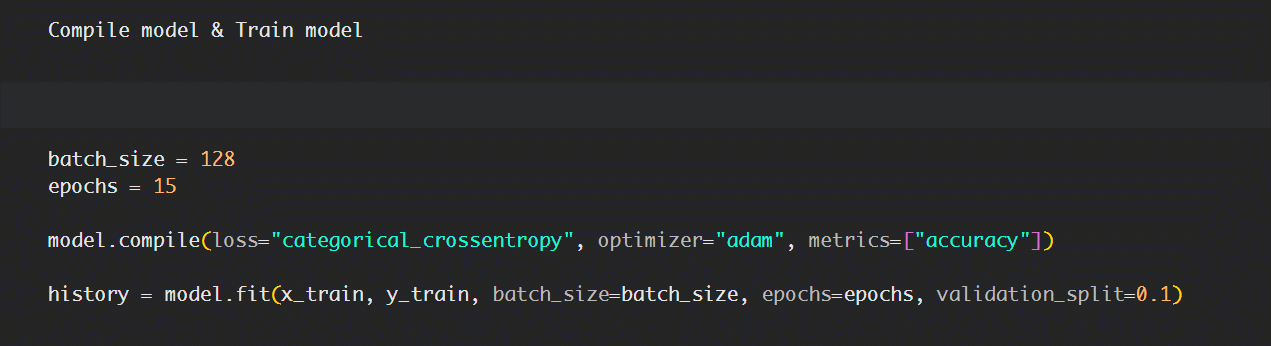


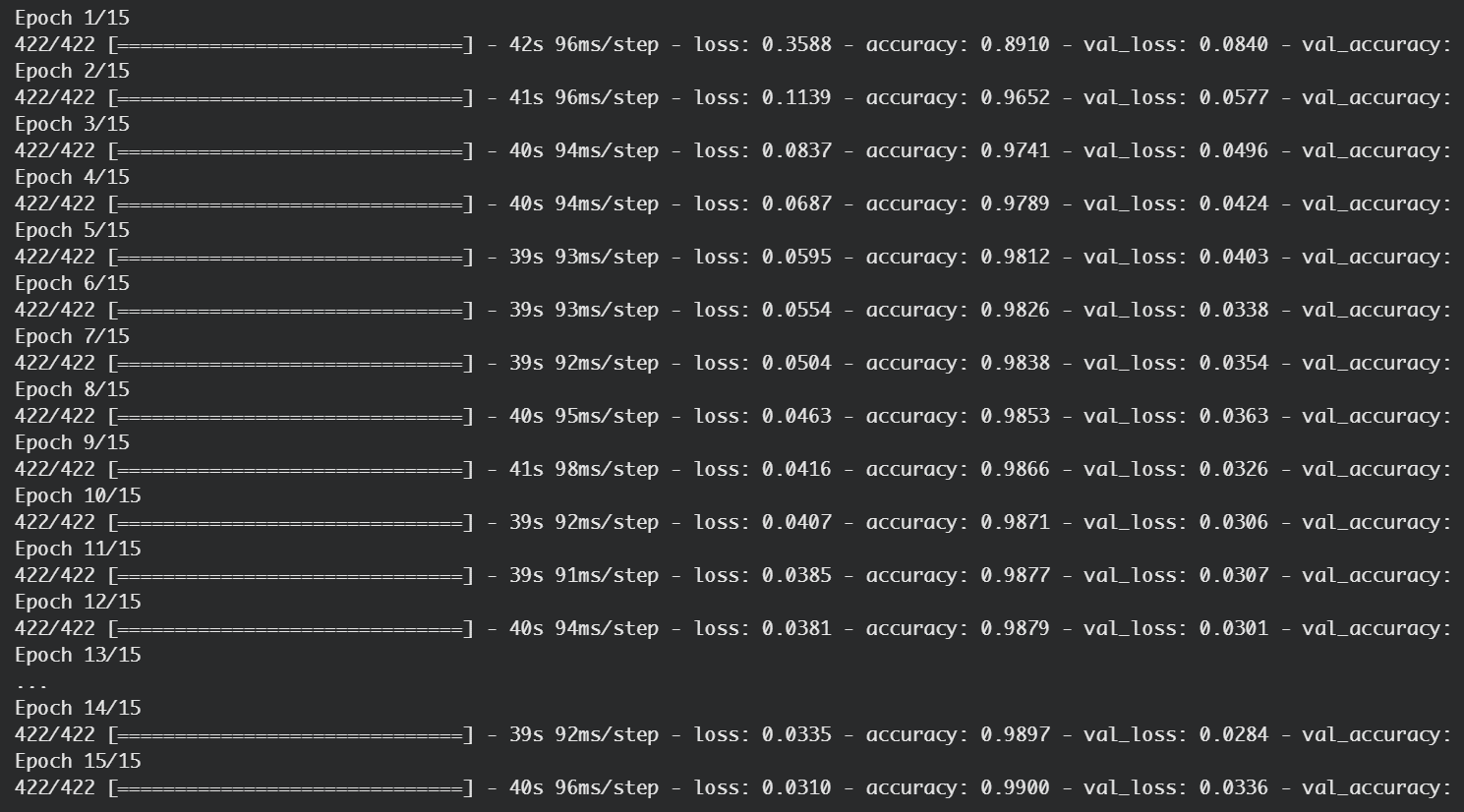


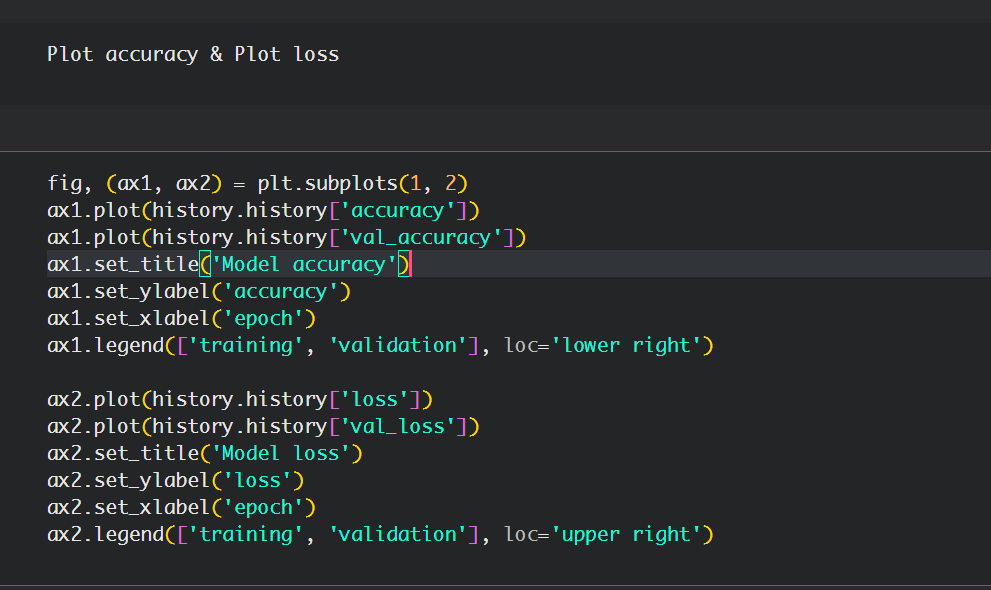




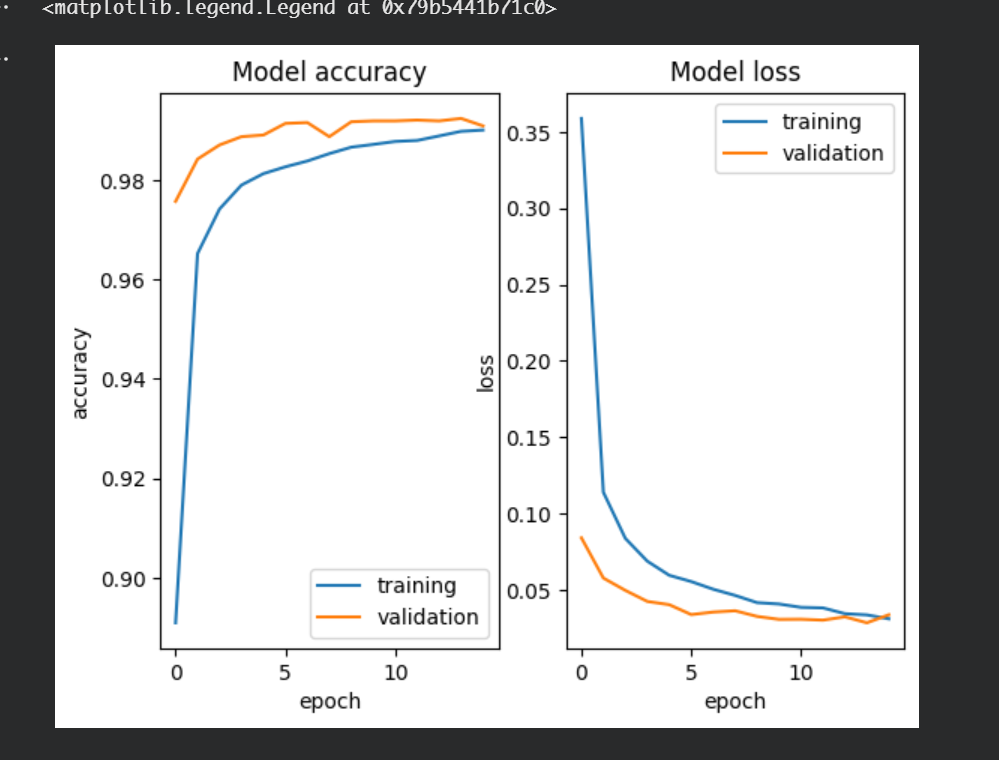








درستی و صحت مدل ها را از نمودار ها می توان متوجه شد مقدار لاس و دقت در داده های تست و اموزش یک فلوی یکسان را طی می کند و دقت زیادو لاس کم شده



مراجع

سوال 1 :

<https://studyglance.in/nn/display.php?tno=5&topic=Adaptive-Linear-Neuron>

سوال 2:

[neural network - What is the difference between Perceptron and ADALINE? - Data Science Stack Exchange](https://datascience.stackexchange.com/questions/36368/what-is-the-difference-between-perceptron-and-adaline)

<https://www.slideserve.com/walda/newton-raphson-method>

<https://www.slideserve.com/walda/newton-raphson-method>

وبسایت فرادرس

Bing

سوال 5 :

[Backpropagation in Neural Network (NN) with Python - Valentyn Sichkar (sichkar-valentyn.github.io)](https://sichkar-valentyn.github.io/Backpropagation/)

سوال 6 :

[Simple MNIST convnet (keras.io)](https://keras.io/examples/vision/mnist_convnet/)