

به نام خدا

تمرين سري دوم
درس مبانى هوش محاسباتى
دكتر ناصر مزيني

فرزان رحماني
۹۹۵۲۱۲۷۱

سوال اول

جواب اين سوال مشابه اسلайд ۳۶ تا ۴۲ NM CI-perceptron.ppt می باشد. لذا از آن کمک گرفتيم و آن را عوض کردیم. بجای and nand استفاده کردیم و دوباره اعداد را محاسبه نمودیم. همچنان با توجه به بحث مطرح شده در گروه بایاس را /۱ گذاشتیم. جواب این سوال در فایل CI_HW2_Q1.ppt پیوست شده است و میتوانید آن را مشاهده کنید.

سوال دوم

(الف)

تابع فعال سازی خطی تابعی است که ورودی های خود را به صورت خطی به خروجی های خود نگاشت می کند. این بدان معنی است که خروجی یک تابع فعال سازی خطی به سادگی مجموع وزنی ورودی های آن است.

در مقابل، تابع فعال سازی غیرخطی تابعی است که ورودی های خود را به صورت غیرخطی به خروجی های خود نگاشت می کند. این بدان معناست که خروجی یک تابع فعال سازی غیرخطی را نمی توان به صورت مجموع وزنی ساده ورودی های آن بیان کرد.

در اینجا چند نمونه از توابع فعال سازی خطی و غیرخطی آورده شده است:

تابع فعال سازی خطی:

- تابع هويت(Identity function) : $f(x) = x$
- تابع خطی(Linear function) : $f(x) = mx + b$

تابع فعال سازی غیرخطی:

- تابع پله واحد(Step function) : $f(x) = 1 \text{ if } x > 0, 0 \text{ if } x \leq 0$
- تابع سیگموئید(Sigmoid function) : $f(x) = 1 / (1 + \exp(-x))$
- تابع تانژانت هیپربولیک(Tanh function) : $f(x) = \tanh(x)$
- تابع واحد خطی اصلاح شده(ReLU) : $f(x) = \max(0, x)$

تابع فعال سازی غیرخطی در شبکه های عصبی مهم هستند زیرا به شبکه های عصبی اجازه می دهند تا روابط پیچیده بین ورودی و خروجی خود را بیاموزند. بدون تابع فعال سازی غیرخطی، شبکه های عصبی فقط می توانند روابط خطی را بیاموزند که توانایی های آنها را محدود می کند.

همچنین طبق اصل تابع خطی هر تابع خطی، تابعی خطی است در جبر خطی اگر در تمامی لایه های شبکه عصبی و نورون ها از تابع فعالسازی خطی استفاده کنیم در نهایت تمامی لایه ها و نورون ها تابعی خطی ($y = ax + b$) تشکیل می دهند و تنها محاسبات را زیاد میکنیم و قدرت شبکه تنها به اندازه یک تابع خطی ساده است که فقط میتوانند مسائل بسیار ساده مانند linear regression را حل کنند. علاوه بر این با استفاده از تنها یک نورون ساده ندارد. پس استفاده از توابع فعال ساز غیرخطی یک امر ضروری در شبکه های عصبی است.

در اینجا یک مثال ساده از نحوه استفاده از توابع فعال سازی غیرخطی برای یادگیری روابط پیچیده آورده شده است:

تصویر کنید در حال آموزش یک شبکه عصبی برای طبقه بندی تصاویر گربه ها و سگ ها هستیم. شبکه عصبی یک تصویر در مقیاس خاکستری را به عنوان ورودی می گیرد و احتمال اینکه تصویر یک گربه یا یک سگ باشد را خروجی می دهد.

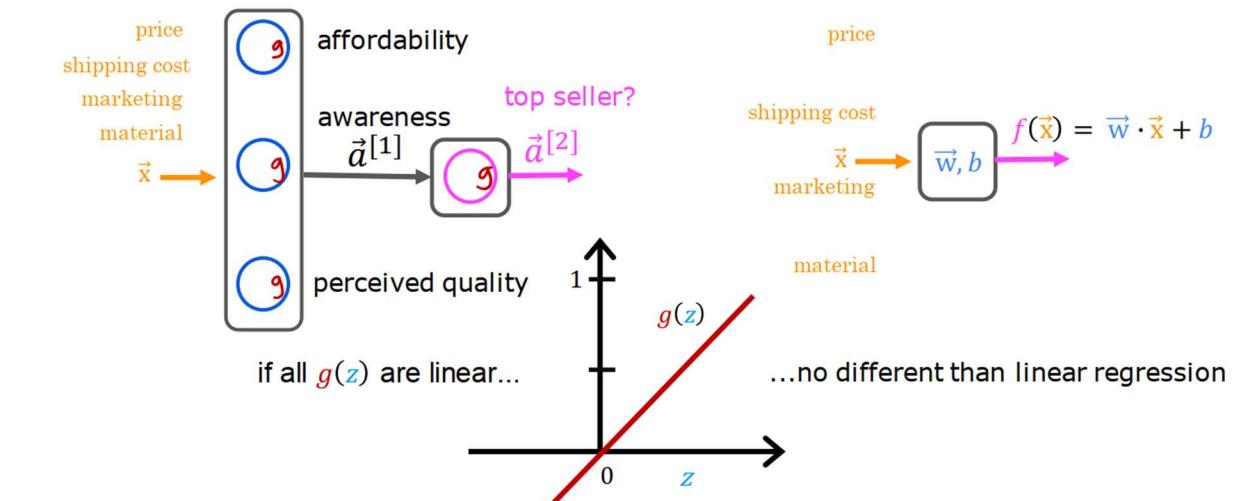
اگر از توابع فعال سازی خطی در شبکه عصبی استفاده کنیم، شبکه عصبی فقط می تواند روابط خطی بین پیکسل های تصویر ورودی و احتمال گربه یا سگ بودن تصویر را بیاموزد. این برای یادگیری طبقه بندی دقیق تصاویر گربه ها و سگ ها کافی نیست.

با این حال، اگر از توابع فعال سازی غیرخطی در شبکه عصبی استفاده کنیم، شبکه عصبی می تواند روابط پیچیده بین پیکسل های تصویر ورودی و احتمال گربه یا سگ بودن تصویر را بیاموزد. این به شبکه عصبی اجازه می دهد تا تصاویر گربه ها و سگ ها را به طور دقیق طبقه بندی کند.

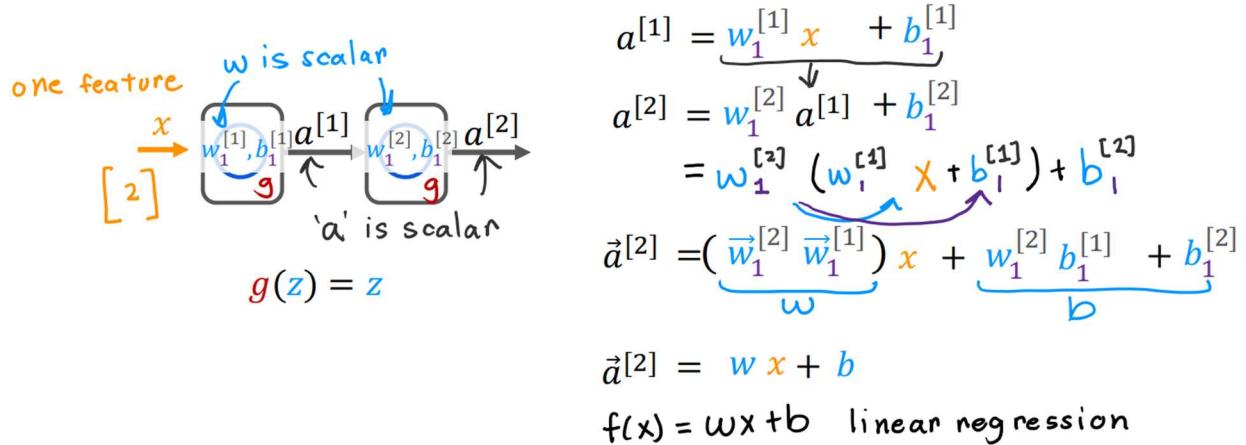
به طور کلی، توابع فعال سازی غیرخطی برای آموزش شبکه های عصبی برای انجام وظایف پیچیده ضروری هستند.

در اسلاید های زیر(ML Stanford course) دلیل اینکه چرا توابع فعال ساز غیرخطی نیاز داریم آمده است:

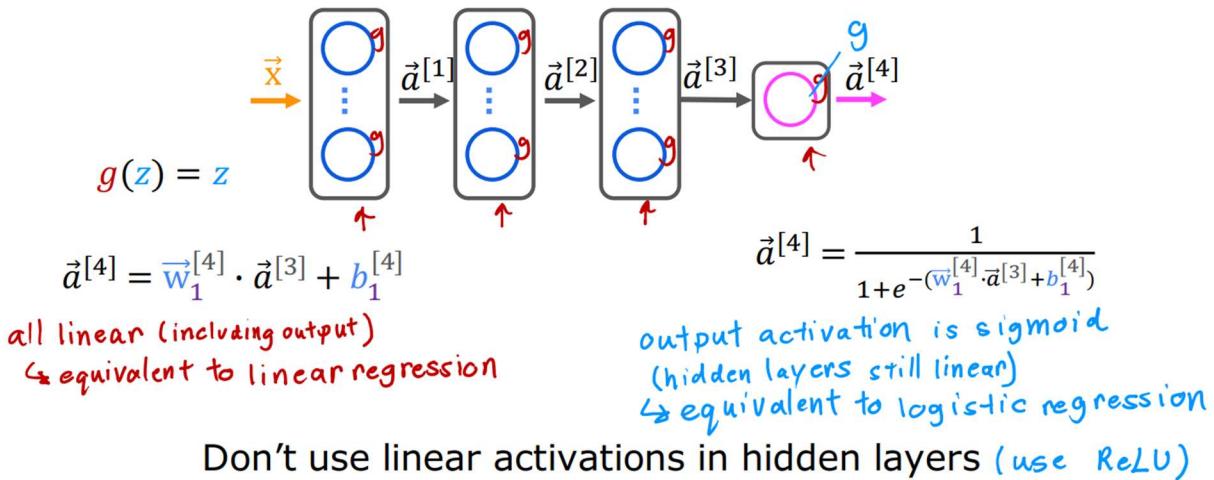
Why do we need activation functions?



Linear Example



Example



جواب سوال به بیان دیگر به زبان chat gpt

The main difference between linear and non-linear activation functions lies in their ability to introduce complexity and non-linearity into a neural network, allowing it to learn and represent complex relationships in data.

1. Linear Activation Function:

- The linear activation function produces an output that is directly proportional to the input.
- The output is a linear combination of the input and the weights. This means the function is a straight line, and the output can be any real number.
- For example, in the case of a simple linear activation function $f(x)=cx$, where c is a constant, the output increases linearly with the input.
- However, the use of linear activation functions in deep neural networks is limited because the whole network becomes equivalent to a single-layer network. This restricts the complexity and representation capabilities of the network.

2. Non-Linear Activation Function:

- Non-linear activation functions introduce non-linearity to the network, enabling it to learn and represent complex relationships in the data.
- They are essential for capturing intricate patterns in the data, which would not be possible with linear activation functions.
- Common examples of non-linear activation functions include the Sigmoid, Tanh, ReLU (Rectified Linear Unit), Leaky ReLU, and Softmax functions.
- Non-linear activation functions allow neural networks to model and learn complex data, making them highly effective for tasks like image recognition, natural language processing, and various other complex pattern recognition problems.

In summary, non-linear activation functions are crucial for enabling deep neural networks to learn complex patterns and relationships within data, which is essential for solving sophisticated real-world problems.

Initialization of weights

- In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.
- If some inputs are much larger than others, random initialization may bias the network to give much more importance to larger inputs. In such a case, weights can be initialized as follows:

– input (first) layer:

$$W_{ji} = \pm \frac{1}{2N} \sum_{i=1, \dots, N} \frac{1}{|x_i|}$$

– Other layers:

$$W_{kj} = \pm \frac{1}{2N} \sum_{i=1, \dots, N} \frac{1}{\varphi(\sum w_{ji} x_i)}$$

30

بایاس تصادفی و وزن ها صفر

اگر بایاس ها را در یک مدل پرسپترون چندلایه (MLP) به طور تصادفی و وزن ها را صفر کنیم، مدل قادر به یادگیری چیزی نخواهد بود. این به این دلیل است که خروجی هر نورون در مدل به سادگی مجموع وزنی ورودی های آن خواهد بود و از آنجایی که همه وزن ها صفر هستند، خروجی هر نورون صفر و بعد از جمع شدن با بایاس برابر بایاس خواهد بود.

حتی اگر آموزش مدل را شروع کنیم، وزن ها هرگز تغییر نمی کنند زیرا گرادیان تابع ضرر نسبت به وزنه ها صفر خواهد بود. این به این دلیل است که خروجی هر نورون بدون توجه به ورودی صفر است.

بایاس صفر و وزن های تصادفی

اگر بایاس ها را در یک مدل MLP به صفر و وزن ها را به صورت تصادفی مقدار دهی کنیم، مدل قادر به یادگیری خواهد بود، اما آموزش آن دشوار و کند خواهد بود. این به این دلیل است که خروجی هر نورون در مدل در ابتدا بسیار کوچک خواهد بود و زمان زیادی طول می کشد تا وزن ها به اندازه کافی به روز شوند تا خروجی های معنی دار تولید شود.

یک از راههای سرعت بخشیدن به فرآیند آموزش، استفاده از یک مقدار مثبت کوچک برای بایاس های اولیه، مانند ۰.۱٪ است. این تضمین می کند که خروجی هر نورون غیر صفر است، حتی اگر وزن ها کوچک باشند.

راه دیگر برای سرعت بخشیدن به روند آموزش، استفاده از weight initialization scheme پیچیده تر، مانند مقداردهی اولیه Xavier یا مقداردهی اولیه He است. این طرحها وزنهای را به گونه‌ای مقدار دهی اولیه می‌کنند که همگرایی سریع‌تر را در طول تمرین افزایش می‌دهند.

به طور کلی، بهتر است از صفر کردن تمام بایاس‌ها و وزن‌ها خودداری کنید، زیرا این کار می‌تواند آموزش مدل را دشوار یا غیرممکن کند.

در اینجا خلاصه‌ای از نحوه انجام آموزش یک مدل MLP، بسته به مقدارهای اولیه وزن‌ها و سوگیری‌ها آورده شده است:

وزن‌ها و بایاس‌های اولیه	روند آموزش
بایاس تصادفی و وزن‌ها صفر	مدل قادر به یادگیری چیزی نخواهد بود.
بایاس صفر و وزن‌های تصادفی	مدل قادر به یادگیری خواهد بود، اما آموزش آن دشوار و کند خواهد بود.

:chat gpt به زبان دیگر به بیان دیگر به زبان

The Multi-Layer Perceptron (MLP) is a type of feedforward artificial neural network that consists of multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer. During training, the network adjusts its weights and biases to minimize the difference between the predicted output and the actual output, using an optimization algorithm such as gradient descent.

Let's consider the two scenarios you mentioned:

1. Random bias and zero weights:

- In this scenario, the initial bias is chosen randomly, and all the initial weights are set to zero.
- Since the initial weights are zero, the network will initially produce the same output for all inputs, regardless of the input values. This makes it impossible for the network to learn from the data during the initial stages of training.
- As a result, the gradients will be zero, and the weights will not be updated during the backpropagation process. This will likely lead to the network being unable to learn the patterns in the data and failing to converge to an optimal solution.

2. Zero bias and random weights:

- In this scenario, the initial bias is set to zero, and all the initial weights are chosen randomly.
- With zero bias, the network may initially struggle to fit the data properly, especially if the data is not centered around zero.
- However, the random weights can introduce some level of diversity in the initial predictions, allowing the network to start learning from the data.
- During the training process, the weights will be updated based on the gradients computed during backpropagation, and the network will gradually learn to map the inputs to the correct outputs, aiming to minimize the loss function.

In both cases, it's essential to have a suitable initialization strategy to help the network converge to an optimal solution during training. Initializing biases and weights appropriately, often using techniques such as Xavier or He initialization, can facilitate faster convergence and better performance of the MLP model.

ج) شبکه های که تاکنون شناخته ایم:

- (Artificial Neuron) Perceptron .۱
- Adaline .۲
- Madaline .۳
- Kohonen .۴
- MLP .۵

تعمیم سازی توانایی یک شبکه عصبی برای عملکرد خوب روی داده های جدید است که قبلاً در طول آموزش ندیده است.

همگی شبکه های عصبی نسبتاً ساده ای هستند که فقط می توانند روابط خطی بین ورودی و خروجی خود را یاد بگیرند. در نتیجه، آنها در تعمیم به داده های جدید خیلی خوب نیستند.

پرسپترون های چندلایه (MLP) شبکه های عصبی پیچیده تری هستند که می توانند روابط غیر خطی بین ورودی و خروجی خود را بیاموزند. این باعث می شود که آنها در تعمیم به داده های جدید بهتر از شبکه های عصبی ساده تر باشند. در واقع مزیت MLP این است که با اتصال نورونهای مصنوعی در این شبکه از طریق توابع فعالسازی غیرخطی، میتوانیم مرازهای تصمیمگیری پیچیده و غیرخطی ایجاد کنیم که به ما امکان میدهد با مشکلاتی که در آن کلاسهای مختلف قابل جداسازی خطی نیستند، مقابله کنیم. پس قابلیت تعمیمپذیری آن از تمام موارد دیگر ذکرشده بیشتر است.

نقشه های خودسازمانده (SOM) نوعی شبکه عصبی هستند که به طور خاص برای یادگیری روابط توپولوژیکی بین ورودی های خود طراحی شده اند. این باعث می شود آنها را در تعمیم به داده های جدیدی که ساختار توپولوژیک مشابهی با داده هایی که روی آنها آموزش دیده اند، خوب تعمیم دهند. معمولاً برای کارهایی مانند خوش بندی (clustering) و کاهش ابعاد (dimensionality reduction) استفاده می شود و به آن اجازه می دهد به خوبی به الگوهای ورودی مشابه تعمیم یابد.

در اینجا رتبه بندی شبکه های عصبی از نظر قابلیت تعمیم (از بیشتر به کمتر) آنها آورده ایم:

- MLP .۱
- Kohonen (SOM) .۲
- Madaline .۳
- Adaline .۴
- (Artificial Neuron) Perceptron .۵

در اینجا توضیح مختصری در مورد اینکه چرا هر شبکه عصبی توانایی تعمیم خاص خود را دارد آورده شده است:

- MLP: MLP ها می توانند روابط غیر خطی بین ورودی و خروجی خود را بیاموزند. این باعث می شود که آنها بتوانند الگوهای پیچیده ای را در داده ها ثبت کنند که به آنها امکان تعمیم به داده های جدید را می دهد. (البته ممکن است overfit هم بشنود ولی با تکینگ های میتوانیم جلوگیری کنیم.)
- Kohonen (SOM): Kohonen (SOM) ها به طور خاص برای یادگیری روابط توپولوژیک بین ورودی های خود طراحی شده اند. این باعث می شود که آنها بتوانند به داده های جدیدی تعمیم دهنده که ساختار توپولوژیک مشابهی با داده هایی دارند که روی آنها آموزش دیده اند.
- Perceptron (Artificial Neuron), Adaline, Madaline: این شبکه های عصبی فقط می توانند روابط خطی بین ورودی و خروجی خود را یاد بگیرند. این توانایی آنها را برای ثبت الگوهای پیچیده در داده ها محدود می کند.

کند، که باعث می شود آنها کمتر بتوانند به داده های جدید تعمیم دهنند. همچنین (Several) Madaline in parallel چون از چندین نورون استفاده میکند قدرت بیشتری نسبت به دو شبکه دیگر داراست و از چندین خط میتوانند استفاده کنند. در واقع ترکیب AND چند Adaline است که بدینهیست نسبت به بقیه موارد تعمیم پذیری و قدرت بیشتری دارد. Adaline هم به نسبت perceptron بیشتر عمل میکند. تفاوت اصلی بین این دو، این است که یک Perceptron، پاسخ دودوی (مانند یک نتیجه طبقه بندی) را میگیرد و یک خطا را محاسبه میکند که برای آپدیت وزن ها استفاده می شود، در حالی که از یک مقدار پاسخ پیوسته برای آپدیت وزنها استفاده می کند. وزن ها را با محاسبه تفاوت بین مقادیر کلاس مورد انتظار و پیش بینی شده به روز می کند. به عبارت دیگر، پرسپترون همیشه $+1$ یا -1 مقادیر پیش بینی شده را با $+1$ یا -1 مقادیر مورد انتظار مقایسه میکند. تفاوت تنها زمانی یاد میگیرد که خطا ایجاد شود. در مقابل، Adaline تفاوت بین مقدار کلاس مورد انتظار $+1$ یا -1 و مقدار خروجی پیوسته y را از تابع خطی محاسبه میکند که میتواند هر عدد واقعی باشد. این به این معنی است که Adaline میتواند حتی زمانی که هیچ اشتباہی در طبقه بندی انجام نشده است یاد بگیرد. از آنجایی که Adaline همیشه و پرسپترون فقط پس از خطاهای یاد میگیرد، راه حلی سریعتر از پرسپترون برای همان مشکل پیدا میکند. این واقعیت که Adaline این کار را انجام میدهد، اجازه میدهد آپدیتهای آن، قبل از اینکه آستانه تعیین شود، ببشر شبهه خطای واقعی باشند و مزایای بیشتری پیدا کند، که به نوبه خود به مدل اجازه میدهد تا سریعتر همگرا شود. ولی در کل همه آنها قابلیت تعمیم کمی برای مسائل پیچیده دارند. در این بین اگر بخواهیم مقایسه ای انجام دهیم به نحوه زیر است:

Madaline > Adaline > perceptron(Artificial Neuron)

Neural network	Generalization ability
Multilayer perceptron (MLP)	High
Kohonen (SOM)	Medium
Perceptron (Artificial neuron)	Low
Adaline	Low
Madaline	Low

:chat gpt به بیان دیگر به زبان

1. Artificial Neuron: An artificial neuron is the simplest unit in a neural network that takes inputs, applies weights and biases, and produces an output using an activation function. Due to its basic structure, it has limited capabilities and may not be able to generalize complex patterns effectively.
2. Perceptron: A perceptron is a simple neural network used for binary classification tasks. It can only handle linearly separable data and has a binary step activation function. While effective for simple tasks, its limited structure may affect its ability to generalize to more complex data.
3. Adaline (Adaptive Linear Neuron): Adaline is an early neural network model that uses a linear activation function and adjusts its weights based on the input data. While suitable for linear regression tasks, its linear nature restricts its ability to generalize to complex patterns.

4. Madaline (Multiple Adaptive Linear Neurons): Madaline extends Adaline to handle multiple inputs and outputs. It is suitable for linear classification tasks but may struggle with non-linear patterns, limiting its generalization capabilities.
5. MLP (Multi-Layer Perceptron): MLP is a versatile neural network model with multiple hidden layers, non-linear activation functions, and the ability to learn complex patterns. It can generalize well to various types of data, making it suitable for a wide range of tasks including classification, regression, and more complex pattern recognition.
6. Kohonen Network (Self-Organizing Map): A Kohonen network, or self-organizing map, is an unsupervised learning neural network that uses competitive learning to create a low-dimensional representation of the input space. It is commonly used for tasks such as clustering and dimensionality reduction, allowing it to generalize well to similar input patterns.

In general, the MLP and the Kohonen network tend to generalize more effectively compared to simpler networks like the artificial neuron, perceptron, Adaline, and Madaline due to their ability to handle more complex patterns and non-linear relationships in data. The Kohonen network, specifically designed for unsupervised learning tasks such as clustering and dimensionality reduction, can generalize well to similar input patterns.



(د)

روش‌های وفقی

- انتخاب مقدار درست برای نرخ آموزش η بسیار مهم و البته مشکل است
 - آیا می‌توان η را به صورت خودکار انتخاب کرد یا اصلاً به آن نیاز نداشت؟
- می‌توان از انحنای منحنی (curvature) هم استفاده کرد
- روش نیوتن: در بسط تیلور از مرتبه دوم هم استفاده کنیم

$$f(\mathbf{x} + \boldsymbol{\epsilon}) = f(\mathbf{x}) + \boldsymbol{\epsilon}^T \nabla f(\mathbf{x}) + \frac{1}{2} \boldsymbol{\epsilon}^T \nabla^2 f(\mathbf{x}) \boldsymbol{\epsilon} + \mathcal{O}(\|\boldsymbol{\epsilon}\|^3)$$

- ماتریس Hessianتابع f با ابعاد $d \times d$ به صورت $H \triangleq \nabla^2 f(\mathbf{x})$ نمایش داده می‌شود
 - برای d ‌های بزرگ از لحاظ مصرف حافظه و محاسبات سنگین خواهد بود
 - تقریب مرتبه ۲ دارای یک بهینه سراسری و قابل محاسبه است

روش نیوتن

$$f(\mathbf{x} + \boldsymbol{\epsilon}) = f(\mathbf{x}) + \boldsymbol{\epsilon}^T \nabla f(\mathbf{x}) + \frac{1}{2} \boldsymbol{\epsilon}^T \mathbf{H} \boldsymbol{\epsilon} + \mathcal{O}(\|\boldsymbol{\epsilon}\|^3)$$

- برای یافتن مقدار بهینه این تابع تقریبی، می‌توان مشتق آن نسبت به $\boldsymbol{\epsilon}$ را برابر با صفر قرار داد
- $$\nabla f(\mathbf{x}) + \mathbf{H} \boldsymbol{\epsilon} = 0 \Rightarrow \boldsymbol{\epsilon} = -\mathbf{H}^{-1} \nabla f(\mathbf{x})$$

- به جای نرخ آموزش از معکوس ماتریس Hessian استفاده شده است

مثال: $f(x) = x^2 - 2x + 3 \Rightarrow f'(x) = 2x - 2 \Rightarrow f''(x) = 2$

$$x_0 = 12 \Rightarrow x \leftarrow x + \epsilon = 12 - \frac{1}{2} \times 22 = 1$$

- در این مثال خاص در یک تکرار از هر نقطه شروعی به جواب بهینه می‌رسد

معادله ای که ارائه شده است مربوط به محاسبه تغییر وزن با استفاده از معکوس ماتریس هسین (Hessian matrix) \mathbf{H} مشتق دوم خطای E نسبت به وزن‌های W در یک آموزش شبکه MLP است. این رویکرد اغلب با روش‌های بهینه سازی مرتبه دوم همراه است. بیایید در مورد مزایا و معایب احتمالی این روش بحث کنیم:

مزایا:

۱. همگرایی کارآمد (Efficient Convergence): روش‌های بهینه سازی مرتبه دوم اغلب سریعتر از روش‌های مرتبه اول همگرایی شوند زیرا از اطلاعات انحنای سطح خطای استفاده می‌کنند. این می‌تواند منجر به همگرایی سریعتر به حداقل شود.

۲. مدیریت موثر خمیدگی (Effective Handling of Curvature): ماتریس هسین به ثبت انحنای سطح خطای کمک می‌کند و اطلاعات ارزشمندی در مورد شکل چشم انداز تابع ضرر در اطراف نقطه فعلی ارائه می‌دهد.

۳. استحکام (Robustness): در مواردی که سطح خطای بسیار غیر خطی است یا دارای انحنای پیچیده است، روش‌های مرتبه دوم می‌توانند قوی تر و پایدارتر باشند و حقی با سطوح خطای نامنظم یا نامنظم به همگرایی بهتر منجر شوند.

معایب:

۱. پیچیدگی محاسباتی (Computational Complexity): محاسبه و معکوس کردن ماتریس Hessian می‌تواند از نظر محاسباتی گران باشد، به خصوص برای مجموعه داده‌های در مقیاس بزرگ و شبکه‌های پیچیده. این می‌تواند روش‌های مرتبه دوم را برابر مدل‌های بزرگ کمتر امکان پذیر کند.

۲. نیازهای حافظه (Memory Requirements): ذخیره و دستکاری ماتریس Hessian می‌تواند به مقدار قابل توجهی از حافظه نیاز داشته باشد، به خصوص برای مسائل با ابعاد بالا. این ممکن است کاربرد روش‌های مرتبه دوم را به مشکلاتی با ابعاد متوسط محدود کند.

۳. حساسیت به نویز (Sensitivity to Noise): روش‌های مرتبه دوم ممکن است به داده‌های پرس و صدا یا پراکنده حساس باشند که منجر به تخمین‌های نادرست از انحنای سطح خطای شود که می‌تواند بر کیفیت به روز رسانی وزن تأثیر بگذارد.

۴. امکان Overfitting : استفاده از Hessian دقیق می تواند در برخی موارد به اضافه برآش منجر شود، به خصوص زمانی که داده های آموزشی محدود است. این روش ممکن است بسیار نزدیک به داده های آموزشی خاص تطبیق داده شود، و این مدل را کمتر قابل تعمیم می کند.

به دلیل پیچیدگی محاسباتی و نیازهای حافظه، استفاده از روش های بهینه سازی مرتبه دوم اغلب در عمل محدود است. اما در مواردی که منابع محاسباتی کافی است و انحنای سطح خطاب برای بهینه سازی حیاتی است، این روش ها می توانند در دستیابی به همگرایی سریع تر و بهبود عملکرد مؤثر باشند.

جواب سوال به بیان دیگر به زبان bard:

The equation you provided is for the Hessian update, which is a second-order optimization algorithm used to train MLP networks. The Hessian is a matrix that contains the second-order partial derivatives of the loss function with respect to the weights.

The Hessian update has a number of advantages over other optimization algorithms, such as gradient descent:

- It can converge to the optimal solution more quickly, especially when the loss function is highly non-convex.
- It is less likely to get stuck in local minima.
- It can provide more information about the curvature of the loss function, which can be used to improve the performance of the network.

However, the Hessian update also has some disadvantages:

- It is computationally expensive to calculate the Hessian matrix.
- It can be sensitive to noise in the data.
- It can be difficult to regularize the Hessian update to prevent overfitting.

Overall, the Hessian update is a powerful optimization algorithm that can be used to train MLP networks more quickly and effectively than other algorithms. However, it is important to be aware of the computational cost and sensitivity to noise before using the Hessian update.

Here are some additional thoughts on the advantages and disadvantages of the Hessian update:

Advantages:

- The Hessian update can be used to train MLP networks with a wide variety of loss functions, including non-convex loss functions.
- The Hessian update can be used to regularize the network weights, which can help to prevent overfitting.
- The Hessian update can be used to accelerate the training process, especially for large networks.

Disadvantages:

- The Hessian update is computationally expensive to calculate, especially for large networks.

- The Hessian update can be sensitive to noise in the data.
- The Hessian update can be difficult to tune, and the optimal hyperparameters can vary depending on the dataset and the network architecture.

Overall, the Hessian update is a powerful optimization algorithm that can be used to train MLP networks more quickly and effectively than other algorithms. However, it is important to be aware of the computational cost and sensitivity to noise before using the Hessian update.

مراجع:

<https://chat.openai.com/>
<https://bard.google.com/>
<https://claude.ai/chats>
<https://DeepLearning.ai>

سوال سوم

- تابع فعال سازی سیگموئید: تابع سیگموئید ورودی را در محدوده $(0, 1)$ نگاشت می‌کند. این تابع در وظایف طبقه بندی بازیزی مفید است، اما ممکن است از شیب‌های ناپدید شونده (vanishing gradients) رنج ببرد، که باعث می‌شود برای شبکه‌های عمیق موثر نباشد.
- تابع فعال سازی Tanh: مشابه تابع سیگموئید است، اما خروجی در محدوده $(-1, 1)$ مقیاس بندی شده است. Tanh با نزدیک کردن میانگین داده‌ها به صفر به مرکز داده‌ها کمک می‌کند. همچنین شیب نرم تری نسبت به سیگموئید دارد.
- تابع فعال سازی ReLU (واحد خطی اصلاح شده): ReLU برای هر ورودی منفی 0 و برای هر ورودی مثبت مقدار ورودی را برمی‌گرداند. بر مشکل گرادیان ناپدید غلبه می‌کند و معمولاً در مدل‌های یادگیری عمیق استفاده می‌شود.
- تابع فعالساز خطی (linear): بود و نبود آن فرق ندارد و وقتی از آن استفاده می‌کنیم انگا که هیچ فعال سازی ای انجام نمی‌دهیم. چراکه طبق اصل تابع خطی هر تابع خطی، تابعی خطی است در جبر خطی اگر در تمامی لایه‌های شبکه عصبی و نورون‌ها از تابع فعالسازی خطی استفاده کنیم در نهایت تمامی لایه‌ها و نورون‌ها تابع خطی ($y = ax + b$) تشکیل می‌دهند و تنها محاسبات را زیاد می‌کنیم و قدرت شبکه تنها به اندازه یک تابع خطی ساده است که فقط میتوانند مسائل بسیار ساده مانند linear regression را حل کنند. عملاً فرق با استفاده از تنها یک نورون ساده ندارد. پس استفاده از توابع فعال ساز غیر خطی یک امر ضروری در شبکه‌های عصبی است. تابع فعال سازی خطی اغلب در آخرین لایه یک شبکه عصبی استفاده می‌شود، جایی که هدف خروجی یک پیش‌بینی خطی است، مانند یک مقدار رگرسیون یا یک احتمال.

دیتا اول



مرز بین دو کلاس یک دایره است که خیلی تابع پیچیده‌ای نیست و انتظار داریم همه توابع به جز خطی بتوانند مرز‌های خوبی (شبیه دایره) پیدا کنند. با سه نورون و یک لایه (سه مرز خطی) مدل‌هایی که نتیجه خوبی داده اند مانند sigmoid و tanh کلاس آبی را با یک مرز تقریباً مثلث شکل (متشكل از سه خط) جدا کرده اند و ReLU تیز با یک مرز چند ضلعی (شش ضلعی) کلاس‌ها را جدا کرده اند. بهترین نتایج را ReLU و Tanh بدست آورده‌اند، sigmoid نتیجه خوبی داشته است و Linear بدترین عملکرد را داشته است. برای دستیابی به نتیجه بهتره با توجه به معادله دایره و مهندسی ویژگی می‌توانیم علاوه بر ویژگی‌های درجه یک از ویژگی‌های درجه دو نیز

استفاده کنیم مثلا علاوه بر x توان دوم آن(x^2) را هم به عنوان ورودی بدهیم.

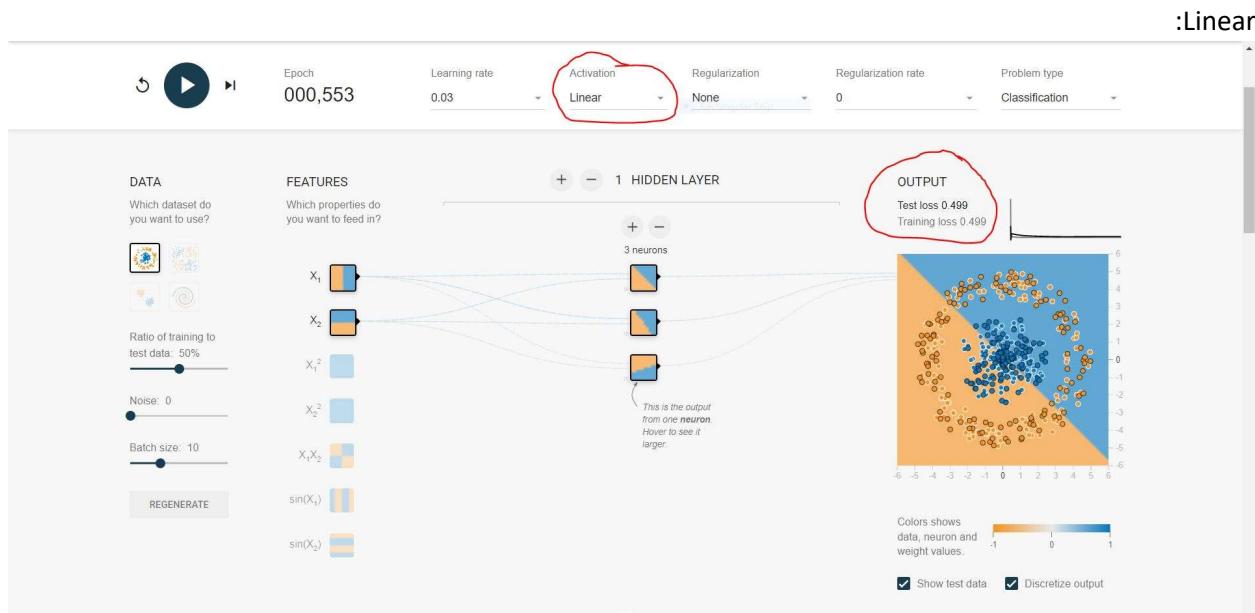
ترتیب نتایج از خوب به بعد:

ReLU .۱

Tanh .۲

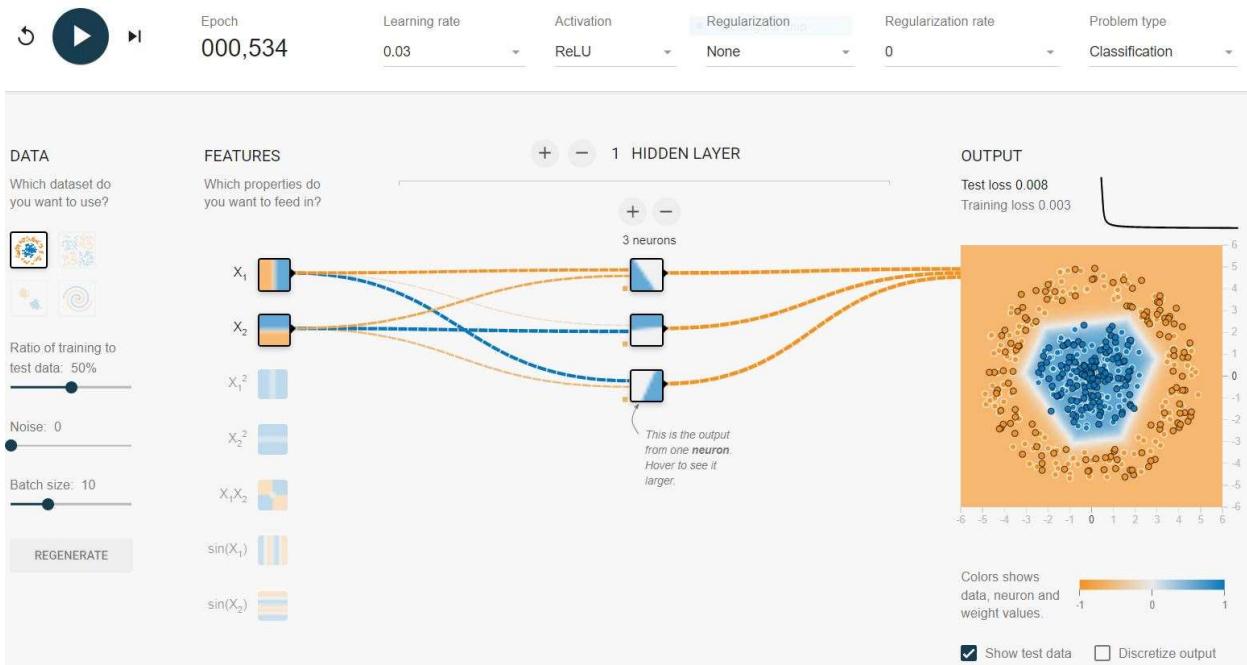
Sigmoid .۳

Linear .۴



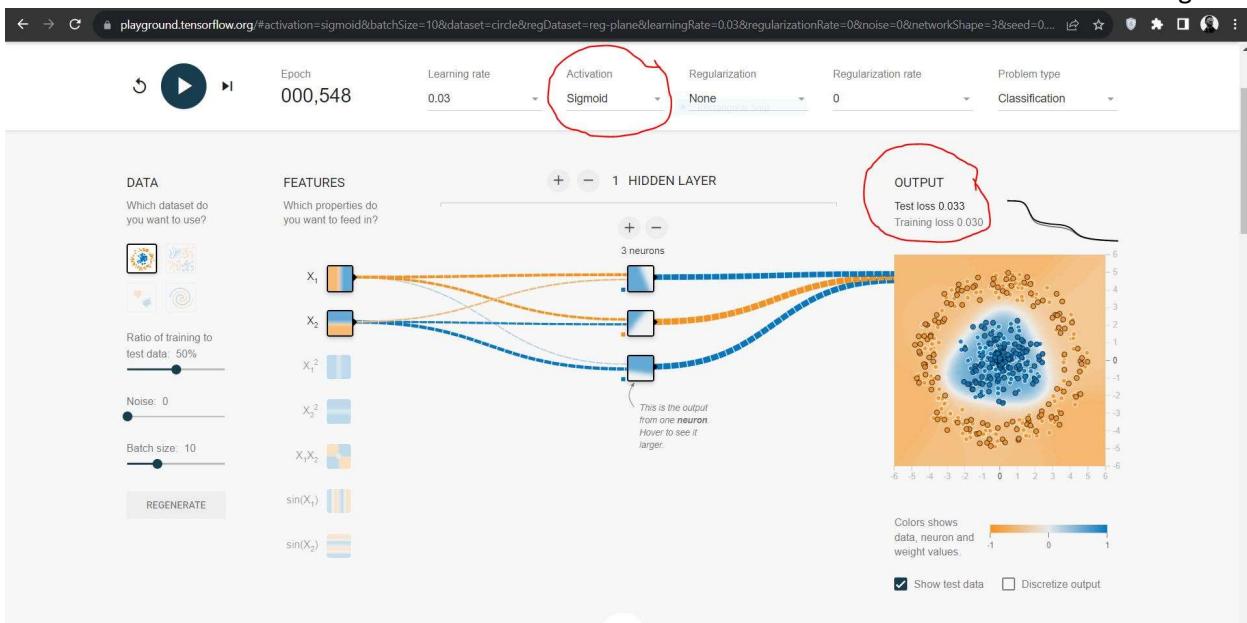
ضرر تست به ۰/۴۹۹ رسیده است که بسیار بد است (امید ریاضی ضرر تست تابع رندم (بدون هیچ شبکه عصبی ای) برای این مسئله ۰/۵ است). البته انتظار داشتیم که نتیجه خوبی نداشته باشیم چرا که با تابع فعال ساز خطی کل شبکه عملاً یک خط است و قدرت کافی برای یافتن مرز دایره ای را ندارد.

:ReLU

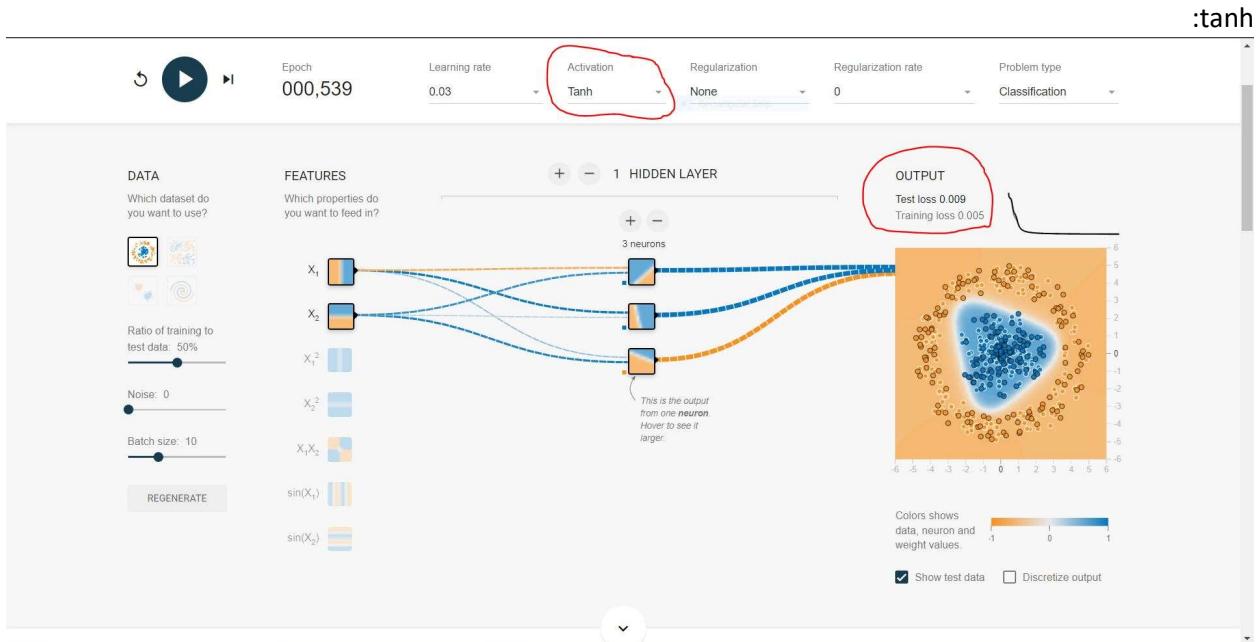


ضرر تست به ۰.۰۰۸ رسیده است که نتیجه بسیار خوبی است و مدل با سه خط(نورون میانی در لایه پنهان) مرز چند ضلعی خوبی(محیط بر نقاط آبی) پیدا کرده است. این تابع بهترین نتیجه را داشته که ۰.۰۰۱ کمتر نسبت به \tanh است.

:sigmoid



ضرر تست به ۰.۰۳۳ رسیده است که نتیجه خوبی است و مدل با سه خط(نورون میانی در لایه پنهان) تقریباً مرز مثلثی(محیط بر نقاط آبی) خوبی پیدا کرده است.



ضرر تست به ۰.۰۰۹ رسیده است که نتیجه بسیار خوبی است و مدل با سه خط(نورون میانی در لایه پنهان) تقریباً مرز مثلثی خوبی(محیط بر نقاط آبی) پیدا کرده است. دلیل اینکه عملکرد tanh از sigmoid بوده است میتواند همگرایی سریع تر نسبت به sigmoid باشد. چرا که خروجی tanh در محدوده (-۱، ۱) مقیاس بندی شده است. Tanh با نزدیک کردن میانگین داده ها به صفر به مرکز داده ها کمک می کند. همچنین شبی نرم تری نسبت به سیگموئید دارد.

دیتا دوم



دیتابست این سوال فضای را به چهار قسمت تقسیم میکند و دو ناحیه برای کلاس نارنجی(منفی) که با دو خط قابل جدا سازی است. همان طور که انتظار داریم فعالساز خطی(linear) نمی تواند آن را جدا کند و بدترین عملکرد را دارد. انتظار داریم همه توابع به جز خطی بتوانند مرز های خوبی پیدا کنند. درین آنها، ReLU بهترین عملکرد را دارد و کمترین ضرر را دارد که با توجه به تعریف آن و advantage های دور از انتظار نبود. بعد از آن، tanh عملکرد خوبی دارد. و بعد از tanh، sigmoid عملکرد متوسط دارد.

ترتیب نتایج از خوب به بعد:

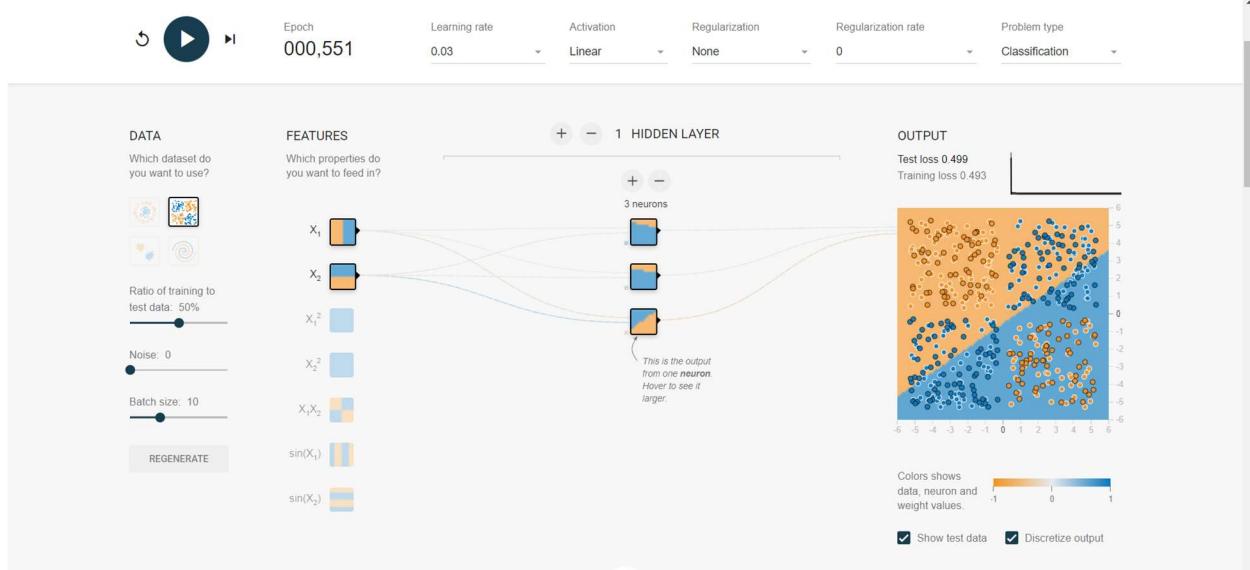
ReLU .۱

Tanh .۲

Sigmoid .۳

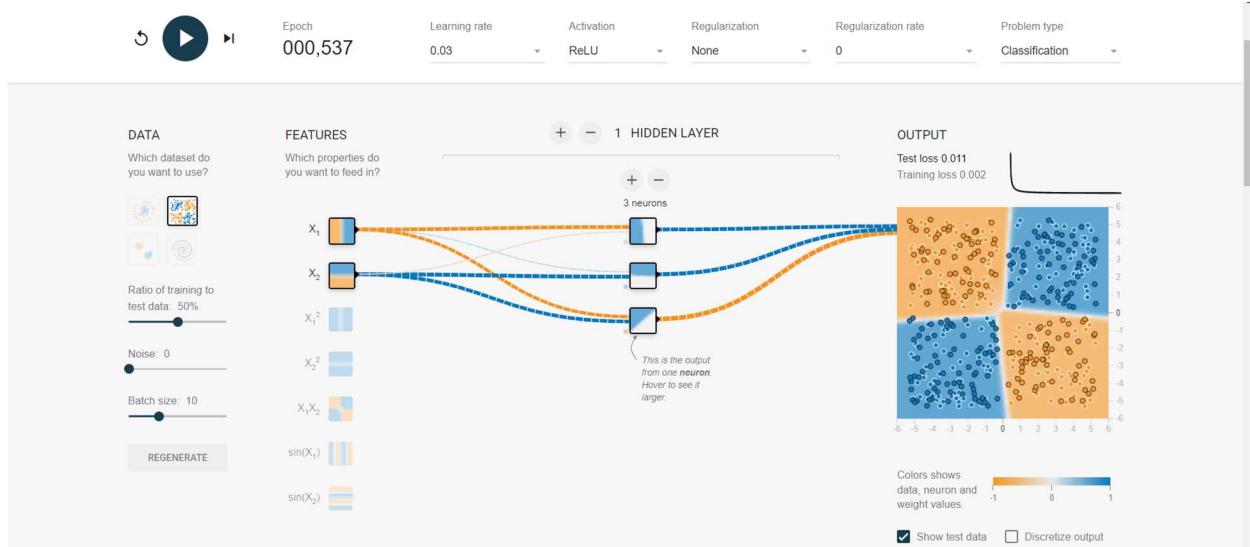
Linear .۴

:linear



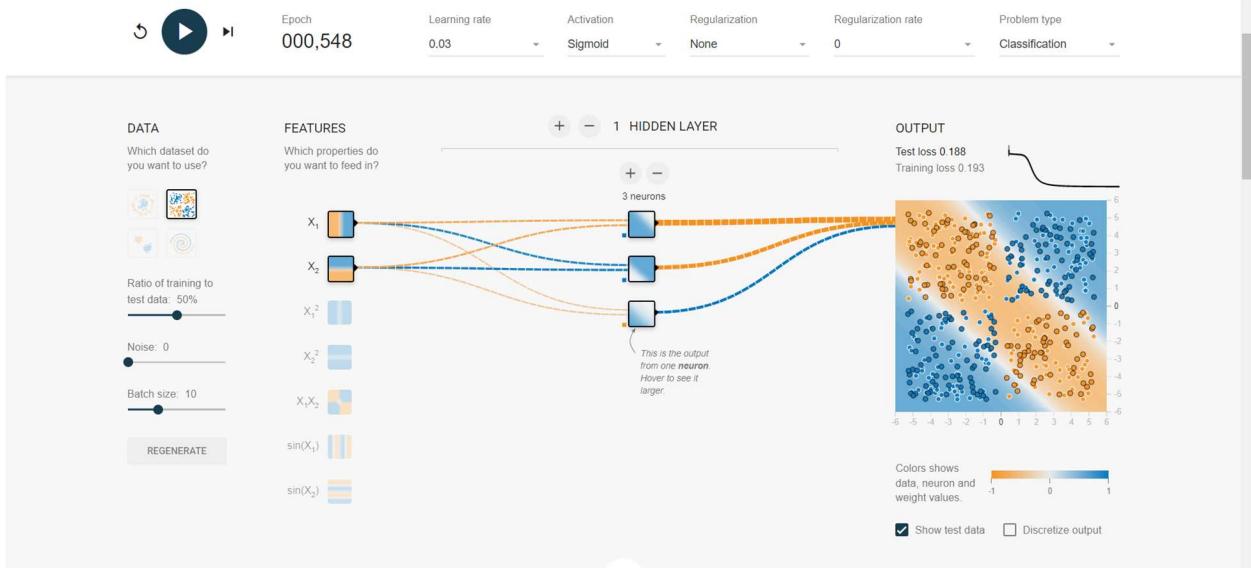
ضرر تست به ۰.۴۹۹ رسانیده است که بسیار بد است (امید ریاضی ضرر تست تابع رندم (بدون هیچ شبکه عصبی ای) برای این مسئله ۰.۵ است). البته انتظار داشتیم که نتیجه خوبی نداشته باشیم چرا که با تابع فعال ساز خطی کل شبکه عملاً یک خط است و قدرت کافی برای یافتن دو مرز خطی و قضاوت بین آنها را ندارد.

:Relu



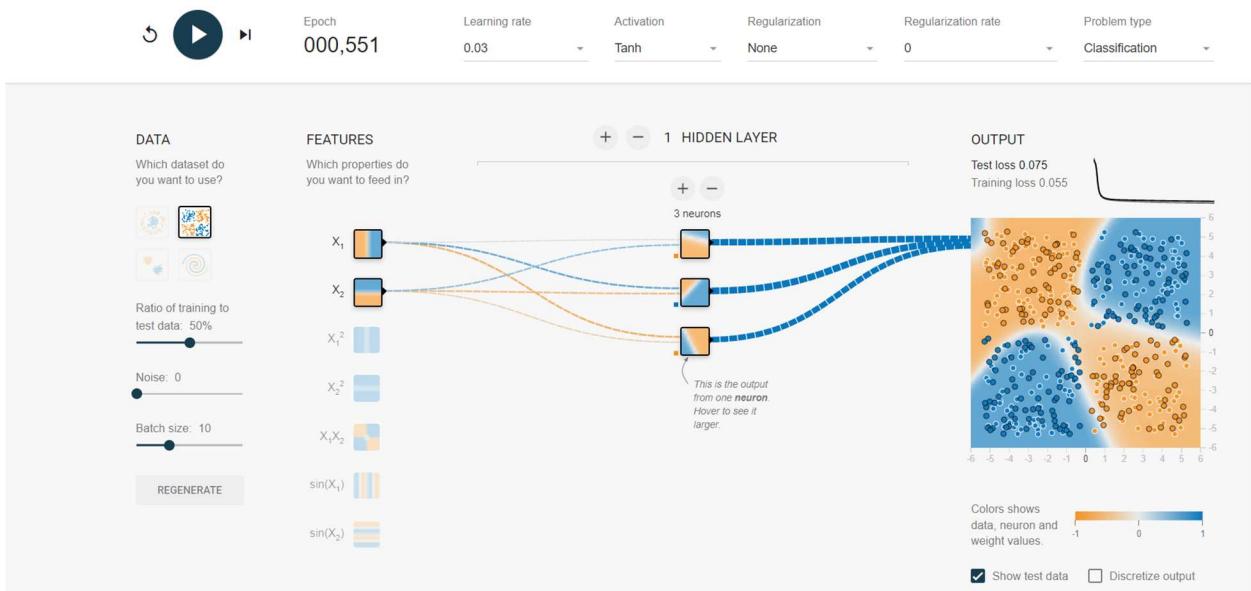
ضرر تست به ۰.۱۱ رسانیده است که نتیجه بسیار خوبی است و مرز خوبی پیدا کرده است. این تابع بهترین نتیجه را در بین توابع دیگر داشته است.

:sigmoid



ضرر تست به ۰/۱۸۸ رسیده است که نتیجه متوسطی است و مدل مرز های خوبی پیدا نکرده است و میتوانست مرز های بهتری را پیدا کند مانند ReLU. (شاید به زمان نیاز دارد تا همگرا بشود)

:Tanh

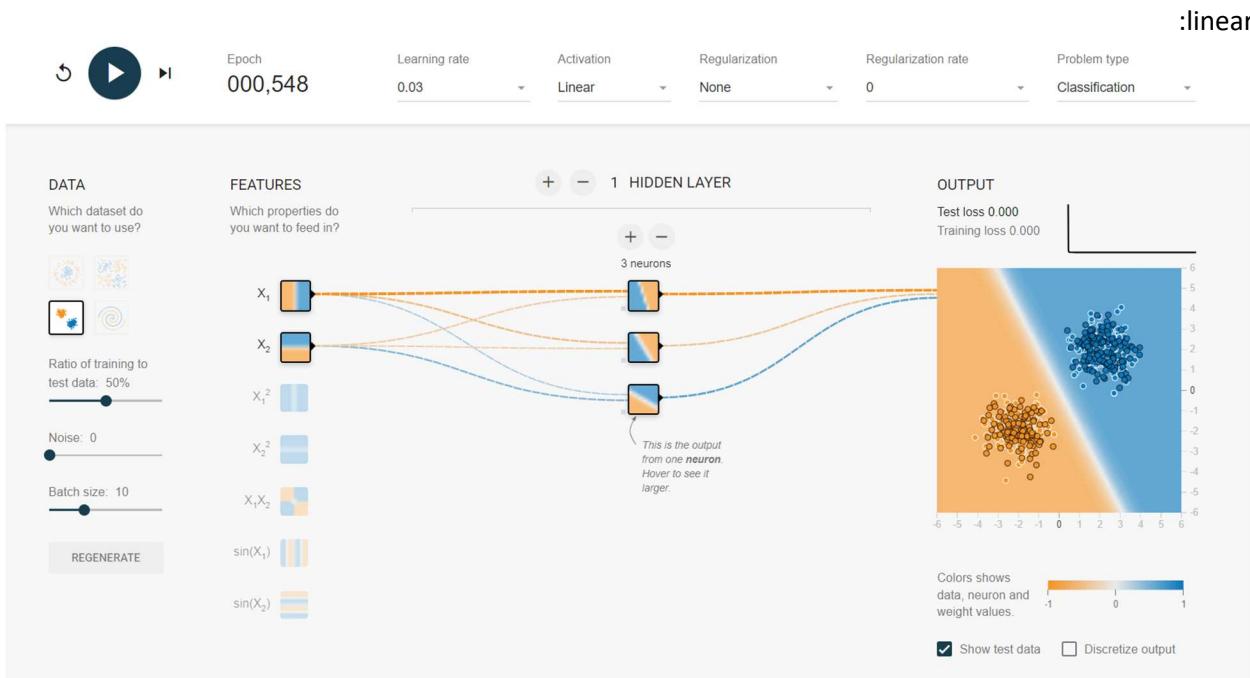


ضرر تست به ۰/۰۷۵ رسیده است که نتیجه خوبی است و مدل تقریباً مرز خوبی پیدا کرده است ولی میتوانست مرز بهتری را پیدا کند (شاید به زمان نیاز دارد تا همگرا بشود). دلیل اینکه عملکرد tanh از sigmoid بهتر بوده است میتواند همگرای سریع تر نسبت به sigmoid باشد. چراکه خروجی tanh در محدوده (-1, 1) مقایس بندی شده است. Tanh با نزدیک کردن میانگین داده ها به صفر به مرکز داده ها کمک می کند. همچنین شبی نرم تری نسبت به سیگموئید دارد.

دیتا سووم

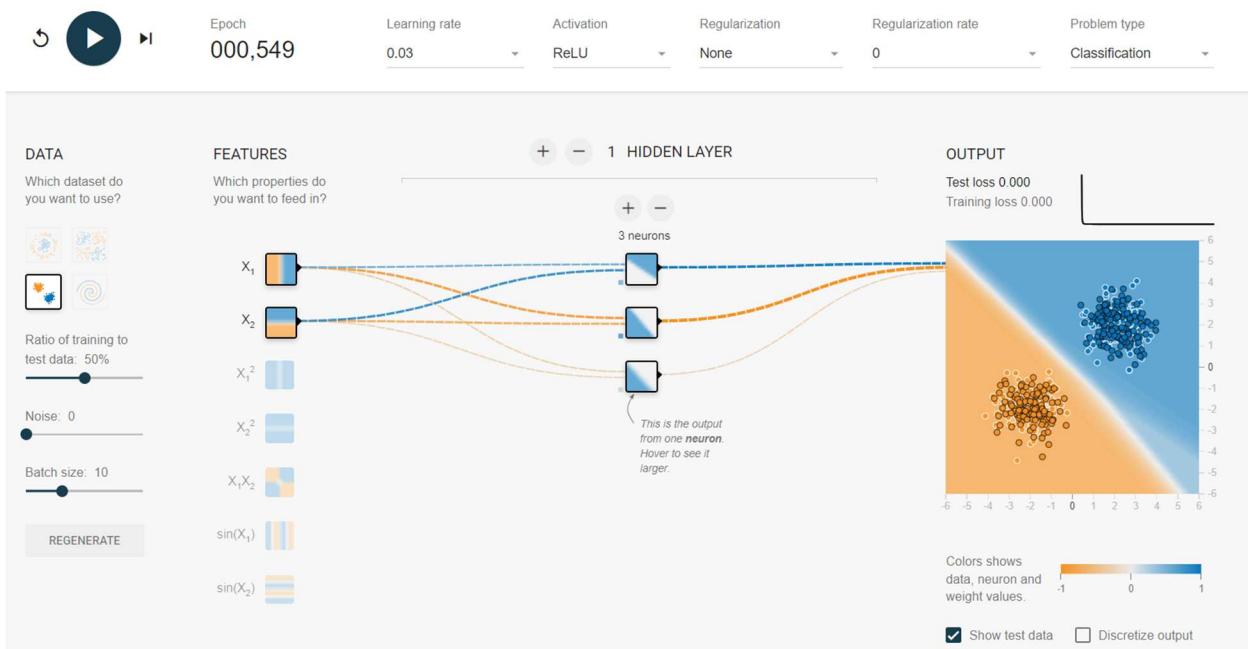


این دیتاست بسیار ساده است و فقط با یک خط قابل جدا سازی (linearly-separable) است. همان طور که مشاهده می کنید بعد از حدود ۵۰۰ مرحله آموزش، ضرر آموزش و تست با هر ۴ تابع فعال سازی به صفر رسیده است حتی برای تابع فعال ساز خطی (linear) که کمترین پیچیدگی و قدرت را دارد و عملا فقط یک نورون است. چراکه مرز جدا سازی دو کلاس فقط یک خط است.



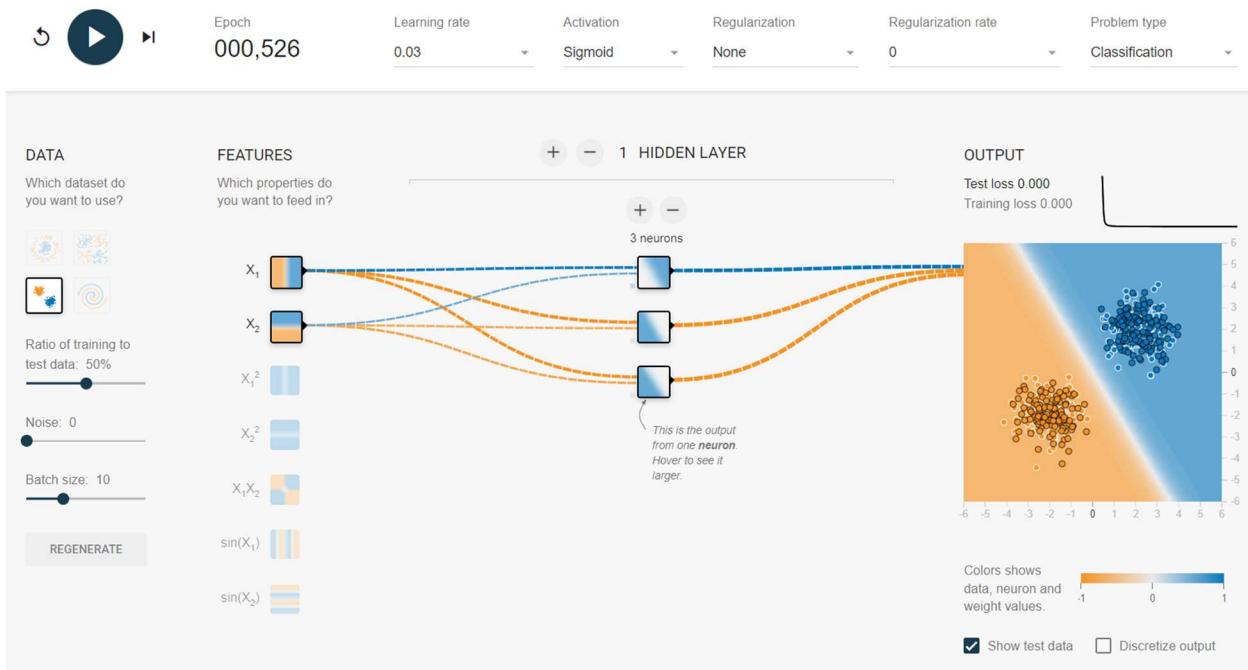
به ضرر . رسیده ایم. (بهترین نتیجه ممکن)

:Relu

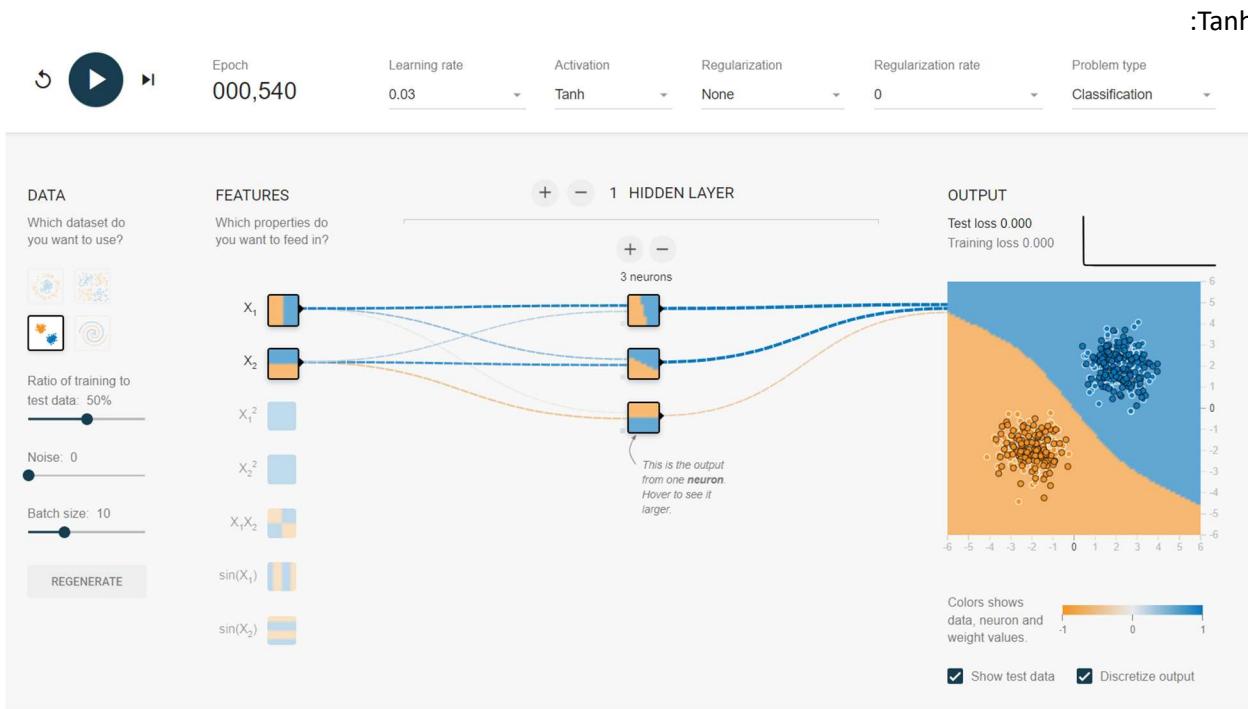


به ضرر . رسیده ایم. (بهترین نتیجه ممکن)

:sigmoid



به ضرر . رسیده ایم. (بهترین نتیجه ممکن)



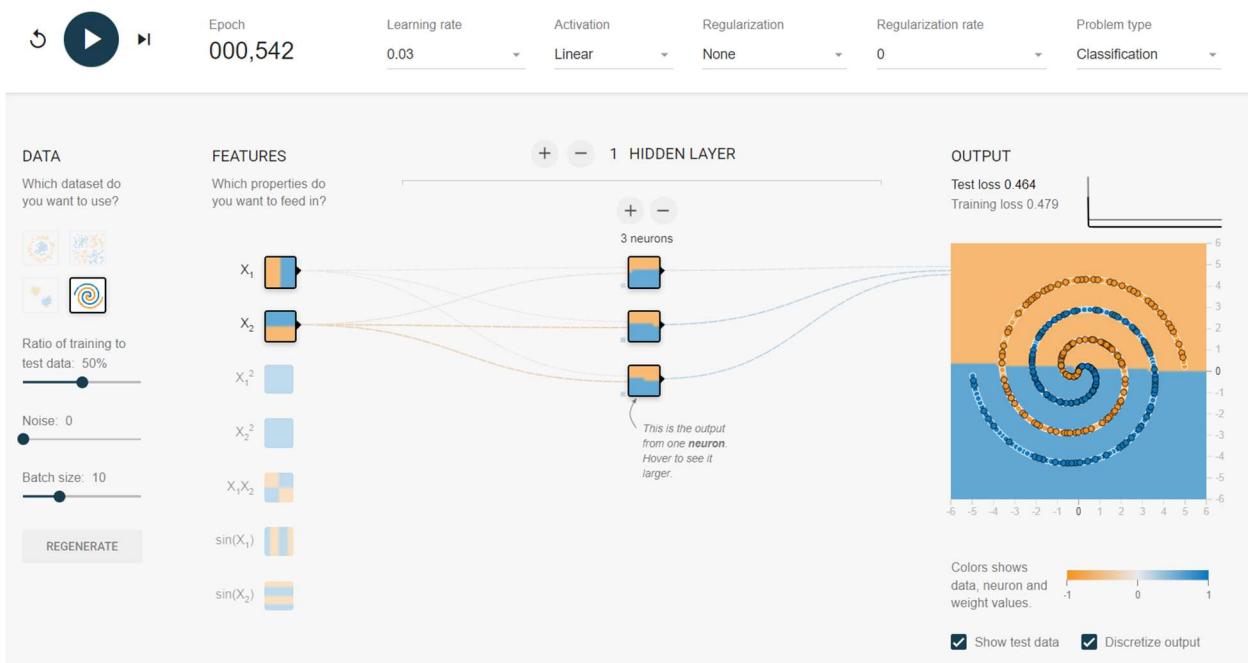
به ضرر . رسیده ایم. (بهترین نتیجه ممکن)

دیتا چهارم



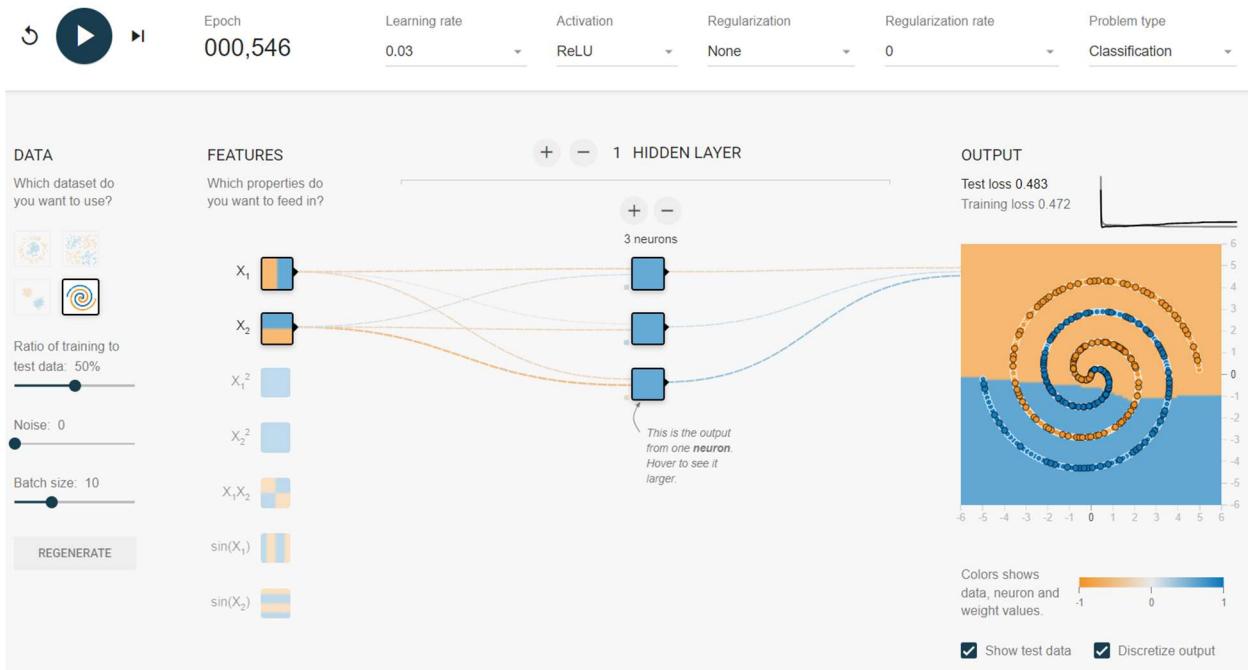
این دیتاست بسیار پیچیده است و هیچ یک از توابع فعال سازی نتوانسته اند که عملکرد خوبی داشته باشند و تنها چند درصد از ۰/۵ فاصله داشته اند. دقیق کنید که دقت ۰/۵ بدون هیچ شبکه عصبی و تنها با یک خط ساده مانند $y=0$ در دسته بندی دو کلاسه قبل حصول است. هیچ کدام از مدل ها نتوانسته اند الگوی مناسبی را پیدا کنند البته این نتیجه مورد انتظار است چرا که با ۱ لایه پنهان و ۳ نورون نمیتوان چنین مرز پیچیده ای را تخمین زد. مرز بین دو کلاس خیلی پیچیده است و برای اینکه به نتیجه خوبی برسیم مهندسی ویژگی و بردن فیچرهای وردی به فضای دیگری می تواند راه گشا باشد.

:linear



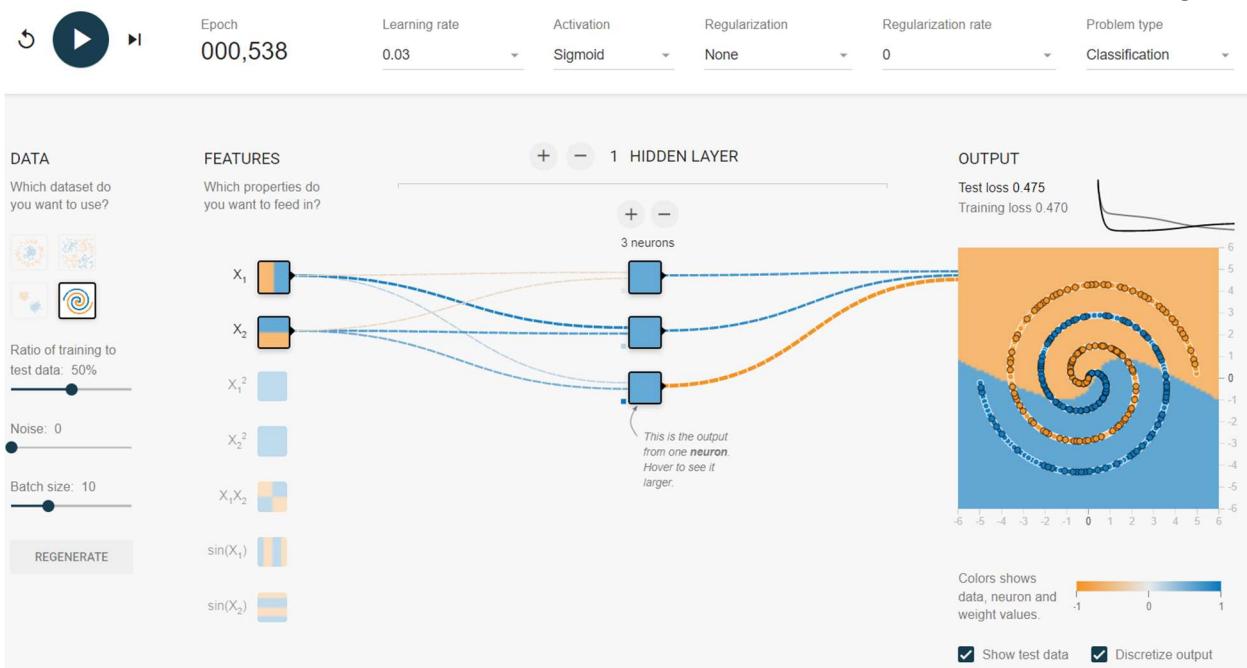
ضرر تست به ۰/۴۷۹ رسیده است که نتیجه خوبی نیست.

:ReLU



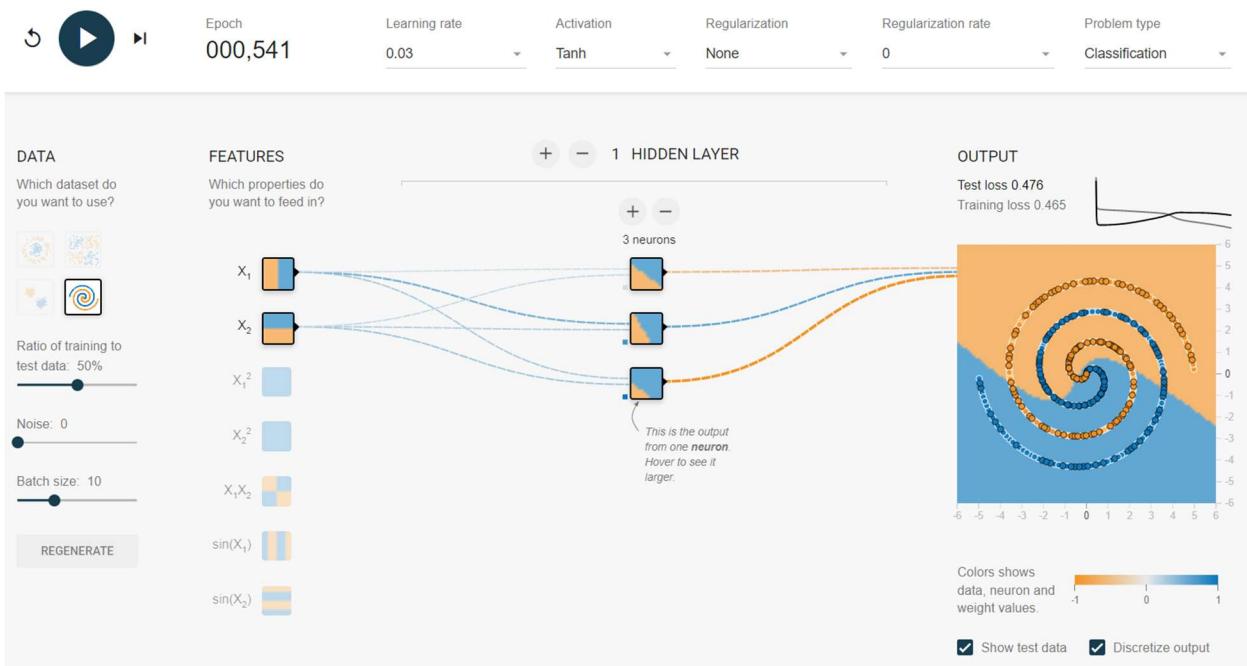
ضرر تست به ۰/۴۷۲ رسیده است که نتیجه خوبی نیست.

:sigmoid



ضرر تست به ۰/۴۷۵ رسیده است که نتیجه خوبی نیست.

:Tanh



ضرر تست به ۰/۴۷۶ رسیده است که نتیجه خوبی نیست.

مقایسه توابع فعالسازی در مجموع(از بهترین به بدترین):

ReLU .۱

Tanh	.۲
Sigmoid	.۳
Linear	.۴

مراجع:

<https://chat.openai.com/>

سوال چهارم

ابتدا کد هایی که خودم پیاده سازی کردم را توضیح می دهم:

```
# In this Create the model, input dim=25 and output dim = 10

#####
# you code here
# model = Sequential()
# model.add(Dense(10, input_dim=25))
# model.add(Activation('softmax'))
#####

model = Sequential([
    Dense(10, input_dim=25, name='layer1'),
    Activation('softmax', name='softmax_activation')
], name='hoda_model')

## we used a very simple model. Use a deeper MLP or a different type of
neural network, such as a CNN may get better result
```

در کد بالا با توجه به کامنت داده شده مدل دو لایه ای را پیاده سازی کرده ام. به دو روش پیاده سازی کرده ام که روش اول کامنت شده است. برای پیاده سازی هم از تابع Sequential تنسورفلو استفاده کردم. برای اضافه کردن لایه fully connected یا همان linear از تابع Dense نظر گرفته بپرسید که همان طور مبینید ابعاد ورودی (تعداد نورون ها) را ۲۵ و بعدا خروجی را (تعداد نورون ها) ۱ نظر گرفته ایم. چرا که ابعاد داده های ورودی ۲۵ است و میخواهیم طبقه بندی ۱ کلاسه انجام دهیم. و همچنین چون طیقه بندی چند کلاسه داریم از تابع فعال ساز softmax استفاده کردم که خروجی ها به توضیح احتمالات تبدیل شوند. برای دست یافته نتیجه بهتر میتوانیم از یک MLP عمیق تر یا نوع دیگری از شبکه عصبی مانند CNN (چون داده های ورودی تصویر رقم هستند) استفاده کنیم.

```
# In this cell compile model, set loss function and optimizer and get
metrics accuracy

#####
# you code here
# loss='categorical_crossentropy' -> multiclass classification
```

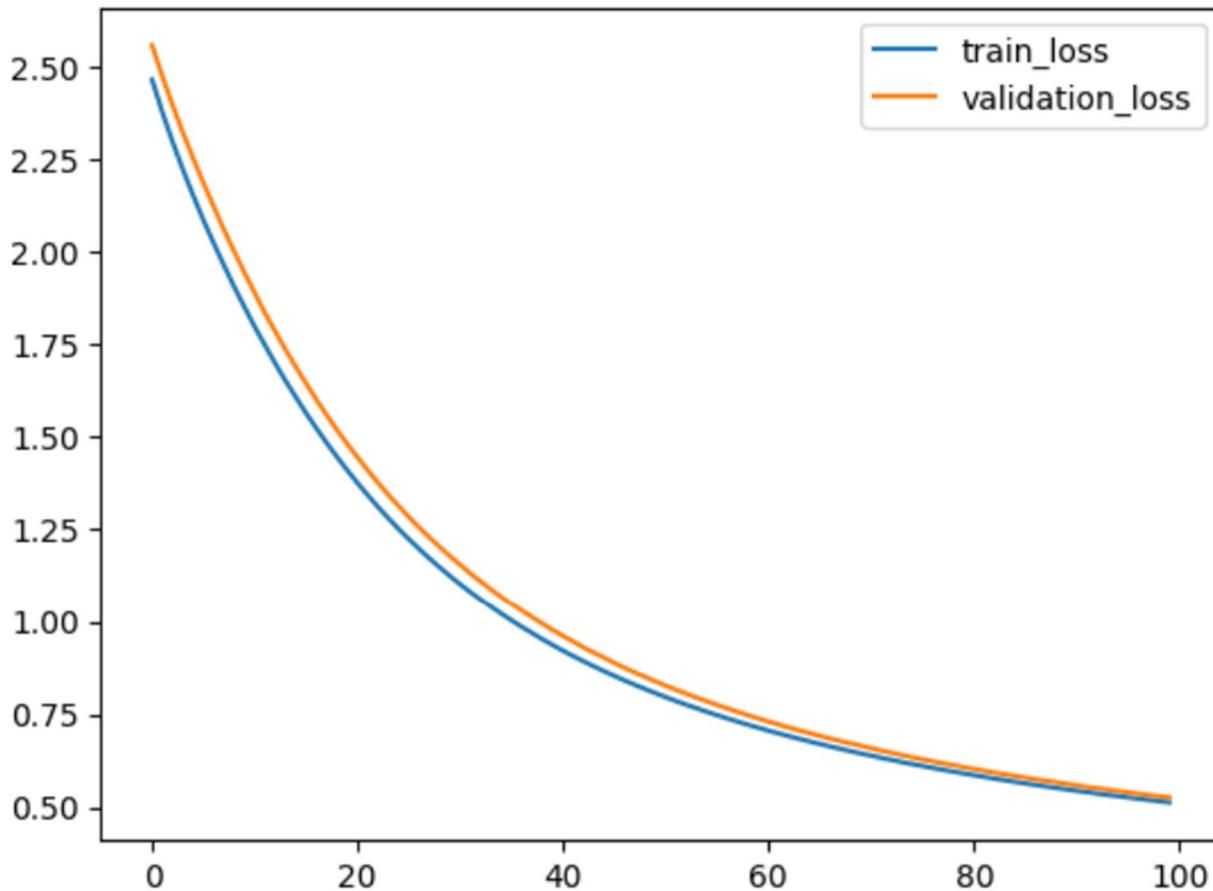
```

# optimizer='adam' -> adaptive moment estimation (adam) because it's
faster than sgd
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])
#####

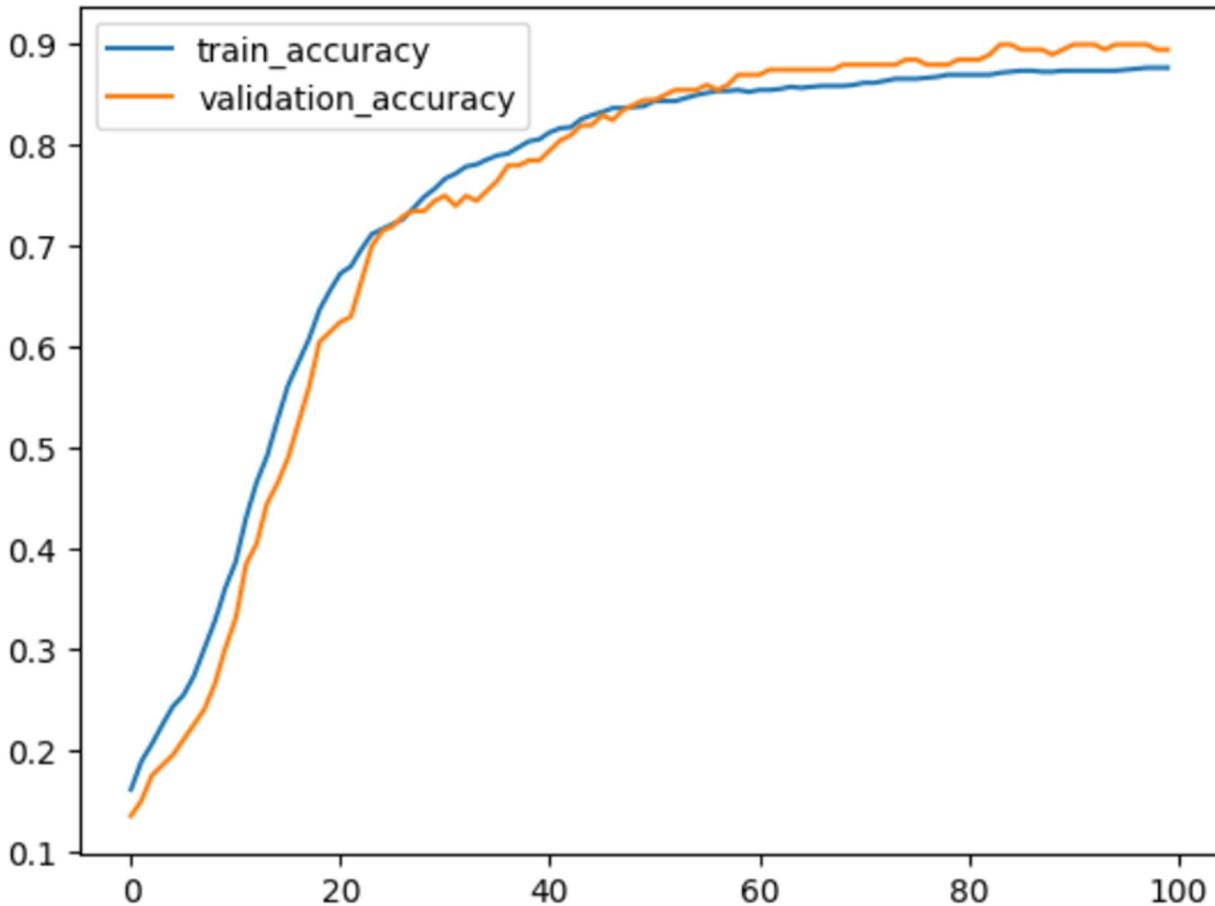
```

همچنین بعد از تعریف مدل باید با مشخص کردن تابع ضرر و بهینه سازی آن را `compile` کنیم. همان طور که میبینید از تابع ضرر استفاده کردیم چرا که مسئله طبقه بندی چند کلاسه هست. همچنین برای بهینه از `categorical_crossentropy` استفاده کرده ایم چرا که نسبت به `gradient decent` زود تر همگرا می شود.

تحلیل نمودارها:



ضرر (Loss) ها به حدود ۰/۵ رسیده اند. همان طور که می بینید ضرر ها به طور پیوسته کاهش یافته اند و این نتیجه خوبی است. همچنین ضرر آموزش با ضرر تست قابل مقایسه است و اختلاف کمی با هم دارند که نشان می دهد مدل `overfit` نشده است. همچنین ضرر ها هم اعداد به نسبت کوچک هستند که نشان می دهد مدل `underfit` نشده است. پس مدل به خوبی روی داده ها fit شده است. ولی برای دست یابی به نتیجه بهتر میتوانستیم از یک MLP عمیق تر یا نوع دیگری از شبکه عصبی مانند CNN (چون داده های ورودی تصویر رقم هستند) استفاده کنیم تا ضرر های کوچکتر داشته باشیم.



دقت(accuracy) ها به حدود ۹۰ درصد رسیده اند که نتیجه خوبی است و نشان می دهد مدل underfit نشده است. همان طور که می بینید دقتهای با گذر زمان افزایش یافته اند و این نتیجه خوبی است. همچنین دقتهای آموزش با دقتهای تست قابل مقایسه است و اختلاف کمی با هم دارند که نشان می دهد مدل overfit نشده است. همچنین دقتهای هم اعداد به نسبت خوبی(۰/۹) هستند که نشان می دهد مدل underfit نشده است. پس مدل به خوبی روی داده های fit شده است. ولی برای دست یابی به نتیجه بهتر میتوانیم از یک MLP عمیق تر یا نوع دیگری از شبکه عصبی مانند CNN(چون داده های ورودی تصویر رقم هستند) استفاده کنیم تا ضررهای کوچکتر داشته باشیم.

در نهایت:

نمودارهای ضرر(Loss) و دقتهای ضرر(accuracy) نشان می دهد که مدل قادر است داده های آموزشی را به خوبی یاد بگیرد و به خوبی به داده های آزمون تعیین دهد. loss آموزش با گذشت زمان به طور پیوسته کاهش می یابد و دقتهای آموزش به طور پیوسته در طول زمان افزایش می یابد. loss و دقتهای تست نیز خوب است و با loss و دقتهای آموزش قابل مقایسه است. پس مدل به خوبی روی دیتابست شده است. همچنین برای دست یابی به نتیجه بهتر میتوانیم از یک MLP عمیق تر یا نوع دیگری از شبکه عصبی مانند CNN(چون داده های ورودی تصویر رقم هستند) استفاده کنیم.

به طور کلی، کد به خوبی نوشته شده است و مدل به خوبی در مجموعه داده HODA عمل می کند.

توضیح کد به طور مختصر(به طور کامل در کامنت هایی که گذاشت قابل فهم است):

دستورات gdown --fuzzy! اینها دستورات پوسته‌ای در نوت بوک Jupyter هستند که برای دانلود فایل‌ها از Google Drive استفاده می‌شود. پرچم --fuzzy برای نشان دادن اینکه شناسه ارائه شده ممکن است دقیق نباشد استفاده می‌شود و حتی اگر شناسه دقیق نباشد به بازیابی فایل از Google Drive کمک می‌کند.

خطوط بعدی، کتابخانه‌های لازم شامل numpy، keras و matplotlib.pyplot را وارد می‌کنند که معمولاً برای یادگیری عمیق، محاسبات عددی و تجسم داده‌ها استفاده می‌شوند.

از keras.layers import Dense، Activation و از keras.models import Sequential این خطوط مازول‌های خاصی را از Keras برای ساخت مدل شبکه عصبی وارد می‌کنند.

from data import import load_hoda: این خط تابع load_hoda را از مازول مجموعه داده وارد می‌کند. این تابع برای بارگذاری مجموعه داده دست نویس فارسی به نام «هودا» استفاده می‌شود.

سپس کد با استفاده از تابع load_hoda بارگذاری داده‌ها را ادامه می‌دهد، که داده‌های آموزشی و آزمایشی اصلی را بازیابی می‌کند.

پس از این، کد داده‌های ورودی Keras را با تبدیل داده‌ها به آرایه‌های NumPy و رمزگذاری یکباره بر جسب‌ها، از قبل پردازش می‌کند.

تابع print_data_info برای چاپ اطلاعات مربوط به نوع و شکل داده‌ها قبل و بعد از پیش‌پردازش تعریف شده است و به کاربر امکان می‌دهد ابعاد و نوع داده‌ها را بررسی کند.

سپس کد اطلاعات داده را قبل و بعد از پیش‌پردازش چاپ می‌کند تا تغییرات را که در طول مراحل پیش‌پردازش رخ می‌دهد نشان دهد.

کد، داده‌ها را با تبدیل مقادیر به محدوده [۰، ۱] عادی می‌کند تا همگرایی بهتر در طول آموزش را تسهیل کند.

سپس، کد مدل شبکه عصبی را با استفاده از Sequential API Keras ایجاد می‌کند. مدلی را با یک لایه ورودی بعد ۲۵ و یک لایه خروجی با بعد ۱۰ با یک تابع فعال سازی softmax تعریف می‌کند.

تابع model.summary() خلاصه‌ای از مدل شامل لایه‌ها و تعداد پارامترها را چاپ می‌کند و یک نمای کلی از معماری مدل ارائه می‌دهد.

کد مدل را با استفاده از روش کامپایل کامپایل می‌کند و بهینه ساز را به عنوان "adam" و تابع ضرر را به عنوان "categorical_crossentropy" مشخص می‌کند که برای کارهای طبقه‌بندی چند کلاسه مناسب است. معیارها برای ردیابی "دقت" در طول آموزش تنظیم شده‌اند.

اسکریپت مدل را با استفاده از روش تناسب آموزش می‌دهد و تعداد دوره‌ها، اندازه دسته و داده‌های اعتبارسنجی را مشخص می‌کند. فرآیند آموزش در متغیر MLP_model ذخیره می‌شود.

در نهایت، اسکریپت با استفاده از matplotlib.pyplot، از دست دادن آموزش و اعتبارسنجی و همچنین دقت آموزش و اعتبارسنجی را ترسیم می‌کند. از تاریخچه ذخیره شده در MLP_model برای دسترسی به مقادیر از دست دادن و دقت در طول آموزش استفاده می‌کند.

هدف کل این کد، پیش‌پردازش و آموزش یک مدل شبکه عصبی بر روی مجموعه داده «hoda» و تجسم عملکرد آموزش و اعتبارسنجی با استفاده از نمودارهای ضرر و دقت است.

در اینجا چند پیشنهاد خاص برای بهبود مدل و فرآیند آموزش وجود دارد:

- از یک MLP عمیق تر یا نوع دیگری از شبکه عصبی مانند CNN استفاده کنید.
- برای تنظیم میزان یادگیری (learning rate scheduler) در طول آموزش از یک زمانبندی نرخ یادگیری (learning rate scheduler) استفاده کنید.
- برای جلوگیری از برازش بیش از حد (overfitting)، از تکنیک‌های منظم‌سازی، مانند منظم‌سازی L1 و L2 استفاده کنید.

- از تکنیک های تقویت داده ها(data augmentation) برای افزایش اندازه و تنوع مجموعه داده آموزشی استفاده کنید.

• مراجع:

<https://chat.openai.com/>

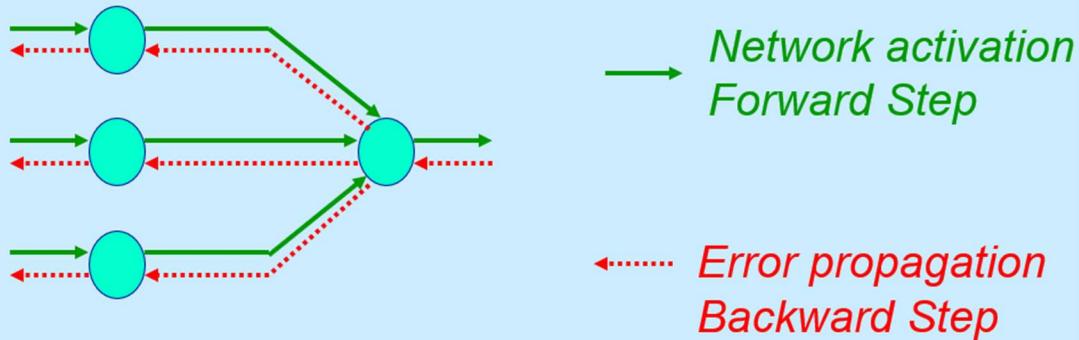
<https://bard.google.com/>

<https://claude.ai/chats>

سوال پنجم

BP

- Back-propagation training algorithm



- BP adjusts the weights of the network in order to minimize the total mean squared error.

Total Mean Squared Error

- The error of output neuron j after the activation of the network on the n^{th} training example ($X_{(n)}$, $d_{(n)}$) is:

$$e_j(n) = d_j(n) - y_j(n)$$

- The network error is the sum of the squared errors of the output neurons:

$$E(n) = \frac{1}{2} \sum_{j \text{ output node}} e_j^2(n)$$

- The total mean squared error is the average of the network errors of the training examples.*

$$E_{AV} = \frac{1}{N} \sum_{n=1}^N E(n)$$

Weight Update Rule

The BP weight update rule is based on the gradient descent method:

Take a step in the direction yielding the maximum decrease of the network error E .
This direction is the opposite of the gradient of E .

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

Summary: Delta Rule

- **Delta rule** $\Delta w_{ji} = \eta \delta_j y_i$

$$\delta_j = \begin{cases} \varphi'(v_j)(d_j - y_j) & \text{IF } j \text{ output node} \\ \varphi'(v_j) \sum_{k \text{ of next layer}} \delta_k w_{kj} & \text{IF } j \text{ hidden node} \end{cases}$$

Where $\varphi'(v_j)$ is the derivative of activation function

در نوت بوک پیوست شده (CI_HW2_Q5.ipynb) با کامنت ها و markdown توضیح مفصل و کامل آمده است. با این حال در اینجا به طور مختصر به توضیح می پردازیم.

این اسکریپت پایتون یک شبکه عصبی مصنوعی (ANN) با معماری پرسپترون چند لایه (MLP) را برای اجرای عملیات XNOR پیاده سازی می کند.تابع XNOR یک عملیات منطقی است که اگر هر دو ورودی درست یا نادرست باشند، مقدار true را برعی گرداند. این اسکریپت یک MLP را برای تقلید از عملیات XNOR آموزش می دهد. در زیر توضیح دقیقی از کد ارائه شده است:

اسکریپت با وارد کردن کتابخانه های لازم و تعریف تابع فعال سازی سیگموئید و مشتق آن آغاز می شود. این توابع برای مراحل انتشار به جلو و عقب در شبکه عصبی ضروری هستند.

تابع XNOR برای برگرداندن ۱ در صورت مساوی بودن هر دو ورودی و . در غیر این صورت تعریف شده است.

داده های آموزشی به شکل یک ماتریس X حاوی تمام ترکیب های ورودی ممکن (به همراه بایاس) تابع XNOR ایجاد می شود و برچسب های مربوطه y با استفاده از تابع XNOR تولید می شوند.

فرابارامترهای (hyperparameters) شبکه عصبی شامل اندازه لایه ورودی، اندازه لایه پنهان، اندازه لایه خروجی، نرخ یادگیری و تعداد دوره های آموزشی تعریف شده اند.

وزن های تصادفی برای دو لایه شبکه عصبی مقداردهی اولیه می شوند.

سپس اسکریپت وارد حلقه ای می شود که مدل را برای تعداد معین دوره آموزش می دهد. حلقه شامل مراحل زیر است:
آ. Forward pass: خروجی های لایه پنهان و لایه خروجی را با استفاده از تابع فعال سازی سیگموئید و حاصل ضرب نقطه ای ماتریس های ورودی و وزن محاسبه می شود.

ب. Backward pass: خطاهای و مشتق‌ها را برای لایه‌های خروجی و پنهان با استفاده از خروجی‌های محاسبه شده و مشتق تابع سیگموید محاسبه می‌شود.

ج. وزن‌ها را با استفاده از گرادیان‌های محاسبه شده و نخ یادگیری از طریق الگوریتم کاهش گرادیان به روز می‌شوند.

در طول آموزش، اسکریپت دوره فعلی و میانگین مربعات خط (MSE) را در فواصل زمانی منظم برای نظارت بر پیشرفت آموزش چاپ می‌کند.

پس از آموزش، اسکریپت با چاپ ورودی، خروجی پیش‌بینی شده و خروجی مدل برای هر ورودی، مدل را روی داده‌های آزمایشی آزمایش می‌کند.

این اسکریپت از تابع فعال سازی سیگموئید، پس انتشار و کاهش گرادیان برای آموزش شبکه عصبی برای انجام عملیات XNOR استفاده می‌کند. خروجی نهایی عملکرد مدل آموزش دیده را بر روی داده‌های آزمون نشان می‌دهد.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

سوال ششم

معماری شبکه MLP برای جداسازی اعداد دستنویس انگلیسی ۰ تا ۹ در دیتاست :mnist

```
# define model
model = keras.Sequential(name="mnist_mlp")
model.add(keras.layers.Input(shape=x_train[0].shape, name="input_layer")) # (28, 28)
model.add(keras.layers.Flatten(name="flatten_layer")) # (28, 28) -> (28*28,) = (784, )
model.add(keras.layers.Dense(units=512, activation='relu', name="layer_1"))
model.add(keras.layers.Dense(units=256, activation='relu', name="layer_2"))
model.add(keras.layers.Dense(units=num_classes, activation='softmax', name="layer_3")) # num_classes=10

model.summary()

Model: "mnist_mlp"



| Layer (type)            | Output Shape | Param # |
|-------------------------|--------------|---------|
| flatten_layer (Flatten) | (None, 784)  | 0       |
| layer_1 (Dense)         | (None, 512)  | 401920  |
| layer_2 (Dense)         | (None, 256)  | 131328  |
| layer_3 (Dense)         | (None, 10)   | 2570    |


Total params: 535818 (2.04 MB)
Trainable params: 535818 (2.04 MB)
Non-trainable params: 0 (0.00 Byte)
```

تعداد لایه‌ها: ۴

دلیل: این یک مجموعه داده نسبتاً ساده با تنها ۳ کلاس است، بنابراین یک شبکه ۴ لایه باید کافی باشد و قدرت لازم را داشته باشد تا

pattern های موجود را بیاموزد. لذا از یک لایه ورودی(input layer)، دو لایه پنهان(hidden layer) و یک لایه خروجی(output layer) استفاده میکنیم. افزودن لایه های بیشتر می تواند منجر به پیچیدگی محاسباتی، پیچیدگی بیش از حد مدل و بیش از حد برآنش(overfitting) شود. همچین بعد از لایه input از لایه Flatten استفاده کردیم که پارامتری ندارد و صرفا عکس های دو بعدی را به ارایه های یک بعدی تبدیل می کند.

تعداد نورون ها در هر لایه:

لایه ورودی: 28×28 نورون یعنی یک نورون برای هر پیکسل در تصویر (چون تصاویر ورودی را graysacle(تک کاناله) هستند و سایز ۲۸ در ۲۸ دارند). در مجموع ۷۸۴ نورون می شوند.

لایه پنهان اول: ۵۱۲ نورون

لایه پنهان دوم: ۲۵۶ نورون

لایه خروجی: ۱۰ نورون (یک نورون برای هر کلاس(0, 1, 2, 3, 4, 5, 6, 7, 8, 9))

دلیل: لایه ورودی باید نورون های کافی برای نمایش تصویر ورودی(به تعداد پیکسل ها) داشته باشد. لایه های پنهان می توانند تعداد نورون های کمتری نسبت به لایه ورودی داشته باشد، اما همچنان باید نورون های کافی برای یادگیری الگوهای پیچیده در داده ها داشته باشد. لایه خروجی باید برای هر کلاس یک نورون داشته باشد تا احتمال هر کلاس را پیش بینی کند.

تابع فعال سازی لایه های پنهان: ReLU

دلیل: تابع فعال سازی ReLU از نظر محاسباتی کارآمد است و نسبت به تابع فعال سازی سیگموئید کمتر مستعد ناپدید شدن گرادیانها است. و به طور معمول انتخاب مناسبی برای لایه های میانی شبکه های عصبی است.

تابع فعال سازی لایه خروجی: Softmax

برای لایه خروجی از softmax استفاده کردیم تا اعداد خروجی(logits) تبدیل به توزیع احتمالاتی شوند ولی اجباری نیست (هزینه محاسباتی اندکی دارد) چرا که با گرفتن ماکریم logit ها می توانیم کلاس پیش بینی شده با بیشترین احتمال را پیدا کنیم به عبارتی بزرگترین logit بعد از عبور از تابع softmax باز هم بزرگترین مقدار را دارد چون تابع softmax اکیدا صعودی است. ولی با استفاده از Softmax در لایه آخر میتوانیم احتمال تعلق به هر یک از ارقام را بینیم و دید بهتری داشته باشیم.

تابع ضرر: Categorical Cross-entropy loss

دلیل: تابع Categorical Cross-entropy loss معمولاً برای کارهای طبقه بندی چند کلاسه(multi-class classification) استفاده می شود. این یک انتخاب خوب برای این کار است زیرا اندازه گیری می کند که چگونه پیش بینی های مدل با بر جسب های واقعی مطابقت دارند.

بهینه ساز: Adam

بهینه ساز Adam یک بهینه ساز قدرتمند و همه کاره است که می تواند برای آموزش طیف گسترده ای از شبکه های عصبی استفاده شود. لذا بهینه ساز Adam یک انتخاب محبوب برای آموزش شبکه های عصبی است زیرا دارای چندین مزیت نسبت به بهینه سازهای دیگر است و زود تر همگرا می شود:

کارآمد(Efficient): بهینه ساز Adam از نظر محاسباتی کارآمد است، به خصوص برای مجموعه داده های بزرگ.

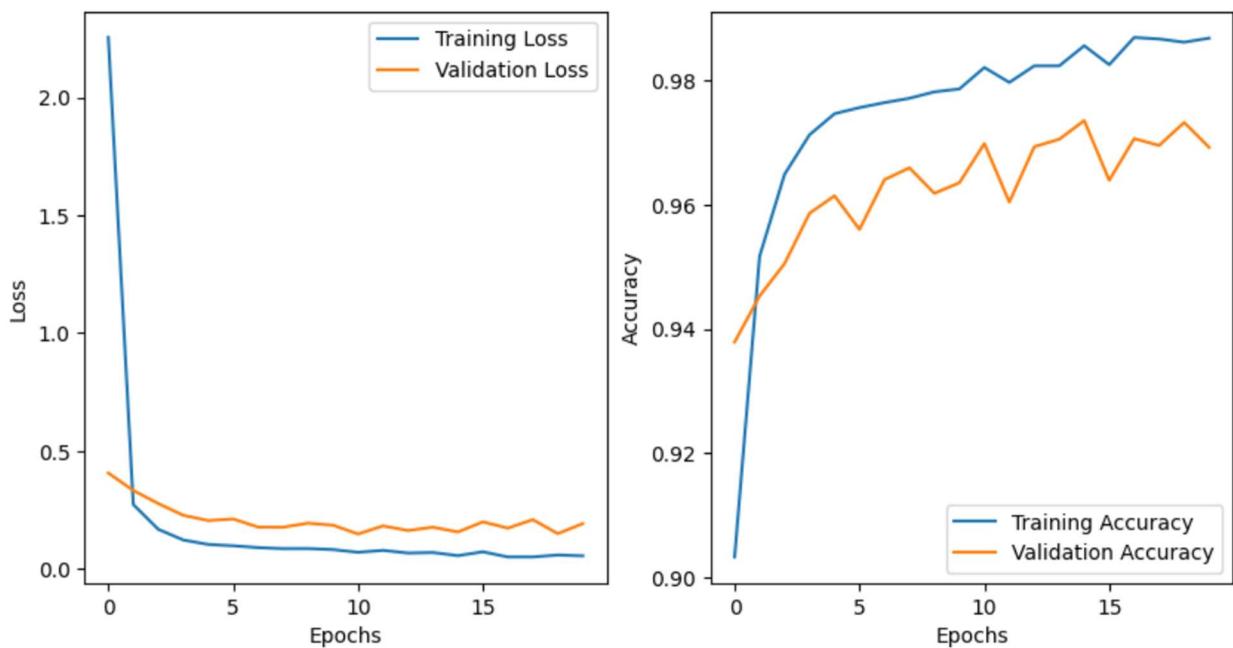
موثر(Effective): بهینه ساز Adam قادر است راه حل های خوبی برای مسائل بهینه سازی پیدا کند، حتی برای مسائل پیچیده و غیر محدب.

قوی(Robust): بهینه ساز Adam نسبت به تنظیم های پارامتر قوی است، به این معنی که نسبت به مقادیر فرابارامترها مانند نرخ یادگیری حساسیت کمتری دارد.

نتیجه:

این شبکه MLP قادر است به طور موثر اعداد دستنويis انگلیسي ۰ تا ۹ در دیتاست mnist را از هم جدا کند. تابع فعال سازی ReLU در لایه های میانی، تابع Softmax در لایه آخر، بهینه ساز Adam و تابع Categorical Cross-entropy loss برای این کار مناسب هستند.

نمودار توابع Loss و Accuracy



همان طور که میبینید ضرر آموزش به 0.05 ، ضرر تست به 0.19 ، دقت آموزش به 0.98 و دقت تست به 0.96 رسیده است. همچنین با گذشت زمان ضرر های کاهش یافته اند و دقت های افزایش یافته اند. این ها نتایج خوبی هستند که نشان می دهد مدل به خوبی رو داده ها fit شده است و با توجه به دقت های بالای 0.90 درصد underfit نشده است. همچنین دقت و ضرر های آموزش و تست قابل ملاحظه اند و اختلاف زیادی با هم ندارند که نشان می دهد overfit هم وجود ندارد. لذا این مدل یک مدل مناسب برای این مسئله است و به دقت بالای 0.95 رسیده است.

توضیح مختصر کد (توضیح بیشتر در کامنت های کد):

from tensorflow import keras: این API را از TensorFlow وارد می کند که برای ساخت و آموزش مدل های یادگیری عمیق استفاده می شود.

import numpy as np: این کتابخانه NumPy را برای محاسبات عددی، به خصوص عملیات آرایه وارد می کند.
from matplotlib import pyplot as plt: این ماژول pyplot را از کتابخانه matplotlib وارد می کند که برای تجسم و رسم داده ها استفاده می شود.

keras.datasets.mnist.load_data(): این خط مجموعه داده MNIST را بارگیری می کند و آن را به مجموعه های آموزشی و آزمایشی تقسیم می کند.

خطوط بعدی کد نمونه ای از مجموعه داده را با استفاده از matplotlib نمایش می دهند. تابع imshow تصویر را نمایش می دهد و plt.title عنوان هر تصویر را بر روی برجسب مربوطه تنظیم می کند.

سپس کد اشکال داده های آموزشی و آزمایشی را برای درک ابعاد آنها چاپ می کند.

keras.utils.to_categorical کدگذاری می کند تا برجسب های عدد صحیح را به ماتریس های کلاس باینری تبدیل کند.

کد داده های ورودی را با تقسیم مقادیر بر $255/0$ عادی می کند تا آنها را در محدوده $[0, 1]$ مقیاس کند. این مرحله کمک می کند تا مدل در طول آموزش سریعتر همگرا شود.

کد یک مدل متوالی را با استفاده از API Keras Sequential تعريف می کند. لایه های مانند لایه ورودی(input)، لایه مسطح(flatten) و چندین لایه متراکم(Dense) با عملکردهای فعال سازی ReLU اضافه می کند. در آخر هم لایه خروجی با فعال ساز softmax وجود دارد. دلیل این انتخاب ها بالاتوضیح داده شد.

() خلاصه ای از مدل را چاپ می کند و جزئیاتی در مورد لایه ها، شکل خروجی و تعداد پارامترها ارائه می دهد.

کد مدل را با استفاده از compile method کامپایل می کند وتابع ضرر را به عنوان 'categorical_crossentropy'، بهینه ساز را به عنوان 'adam' و متريک را به عنوان 'دقت' مشخص می کند.

این مدل با استفاده از fit method با اندازه دسته مشخص، تعداد دوره ها و داده های اعتبار سنجی آموزش داده می شود. Shuffle=Argument True داده های آموزشی را قبل از هر دوره به هم می زند.

اسکریپت برچسب های خروجی (y_pred) را برای داده های تست پیش بینی می کند و چند تصویر آزمایشی را همراه با برچسب های واقعی و پیش بینی شده شان نمایش می دهد.

در نهایت، اسکریپت با استفاده از matplotlib.pyplot، ضرر آموزش و اعتبار سنجی و همچنین دقت آموزش و اعتبار سنجی را ترسیم می کند. این به تجسم عملکرد مدل در طول فرآیند آموزش کمک می کند.

مراجع:

<https://chat.openai.com/>

<https://bard.google.com/>

<https://claude.ai/chats>

<https://colab.research.google.com/drive/1Bk6vn1UddLwIAVlzqJzNrFGcXTTf3xx4>

پایان