به نام خدا

پروژه دوم

شبکه عصبی چند لایه

درس هوش مصنوعی

احمد زعفرانی97105985

تابستان 99

راهنمای اجرا

پیشنیازها: از کتابخانه های زیر در این کد استفاده شده است:

sklearn – numpy – matplotlib – PIL (pillow)

بخش اول : در پوشه بخش اول، 5 فایل وجود دارند که هر کدام نشانگر یکی از مثال های استفاده شده برای این بخش هستند. برای مشاهده توابع استفاده شده در هر مثال، به بخش توضیحات برنامه مراجعه فرمایید.

بخش دوم : در پوشه مربوط به این بخش نیز 5 فایل، متناظر با فایل های بخش قبل، وجود دارند که هر کدام نشانگر اضافه شدن نویز به آن مثال هستند.

بخش سوم : در این پوشه ... فایل وجود دارند که هر کدام مربوط به یکی از مثال های توضیح داده شده در این بخش هستند. عدد چاپ شده بیانگر خروجی lose function در نظر گرفته شده برای هر بخش می باشد.

بخش چهارم :

بخش پنجم : در این پوشه نیز فایل مربوط به این بخش قرار گرفته است. برای اجرای درست برنامه، خطوط 22 و 23 این فایل باید تغییر کنند؛ به این صورت که آدرس پوشه های مربوط به داده های آموزشی و آزمایشی پایگاه داده USPS، به ترتیب، باید در این خطوط جایگزین شوند.

بخش ششم : در این پوشه نیز فایل مربوط به این بخش قرار گرفته است. برای اجرای درست برنامه، خطوط 39 و 40 این فایل باید تغییر کنند؛ به این صورت که آدرس پوشه های مربوط به داده های آموزشی و آزمایشی پایگاه داده USPS، به ترتیب، باید در این خطوط جایگزین شوند.

توضیحات برنامه

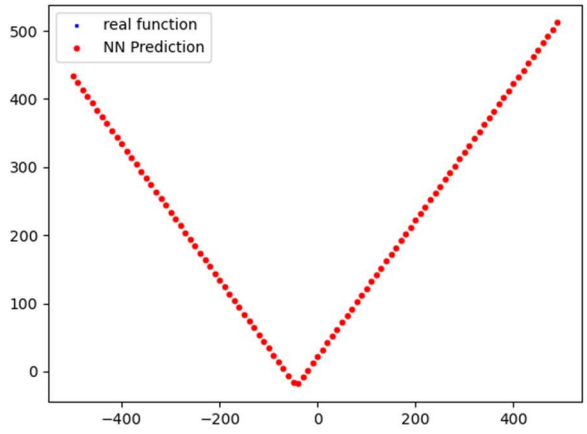
بخش اول

در بخش اول، اولین چالش نرمال سازی داده های آموزشی و آزمایشی است. این پارامتر تاثیر به سزایی در دقت الگوریتم ایفا می کند. اما با توجه به صورت سوال، دامنه داده های آزمایشی را 5 برابر بزرگتر از داده های آموزشی در نظر گرفتیم (جالب آنکه اگر بازه تغییرات مختصات داده های آزمایشی کوچکتر از دامنه داده های آموزشی باشد، دقت شبکه به طرز چشمگیری افزایش پیدا می کند). همچنین بنابر مشاهدات، افزایش تعداد داده های آموزشی، تاثیر چندانی بر دقت یادگیری شبکه نمی گذارد. همچنین در برخی مثال ها عنصر تصادف در یادگیری شبکه تاثیر گذار است؛ مثلا ممکن است اجرای چندین باره یک کد، خروجی متفاوتی داشته باشد.

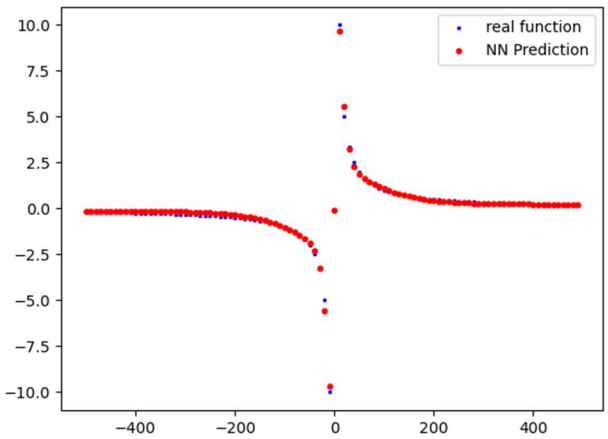
در تمامی مثال های زیر، ابتدا 20 نقطه آموزشی بوسیله تابع مشخص شده بصورت تصادفی تولید می کنیم، سپس 100 نقطه آزمایشی دیگر (باز هم بصورت تصادفی) تولید کرده، آنها را بوسیله شبکه عصبی دسته بندی می کنیم. دقت کنید که نقاط آبی رنگ مربوط به تابع اصلی و نقاط قرمز رنگ مربوط به تابع یادگرفته شده توسط شبکه عصبی می باشد:

1. تابع : $|x + 44| - 22$

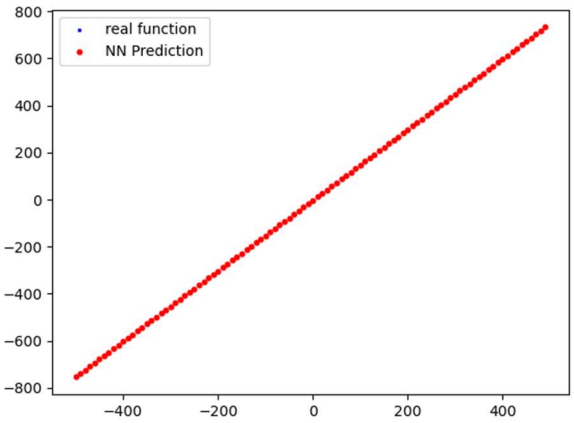
   در این مثال نقاط آبی رنگ به علت پنهان شدن زیر نقاط قرمز مشخص نیستند؛ این به این معناست که دقت پیشبینی شبکه 100% بوده است!



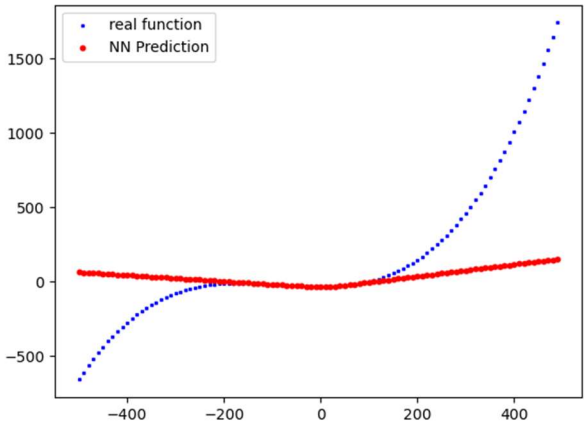
2. تابع : $100/x$



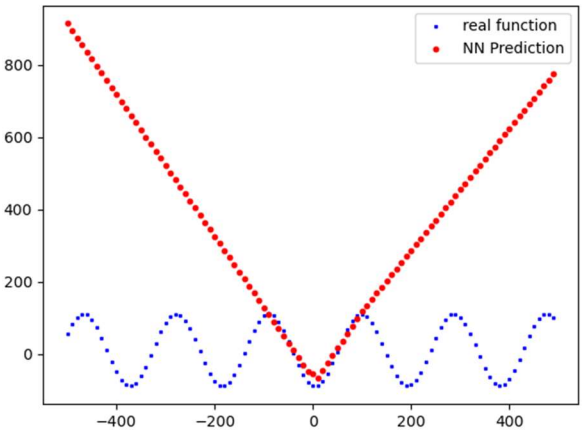
3. تابع : 1.5 * x – 3.5



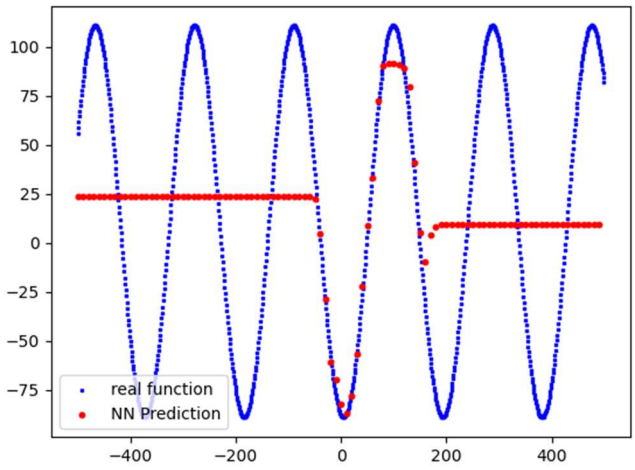
4. تابع : 0.00001 * x**3 + 0.0025 * x**2 – 33



5. تابع $100 * sin\left(\frac{1}{30*x} - 8\right) + 11$



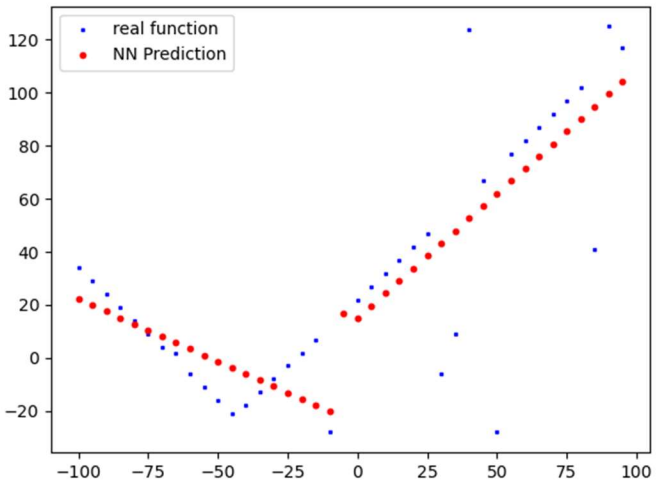
همانطور که در این مثال مشاهده می کنیم چون دامنه داده های آموزشی بین -100 تا 100 بوده، شبکه در این دامنه عملکرد نسبتا خوبی دارد اما خارج از این شبکه کاملا غلط عمل کرده است (شبکه را فریب دادیم). برای همین این مثال را با داده های آموزشی به اندازه تمام دامنه نقاط آزمایشی تکرار کردیم:



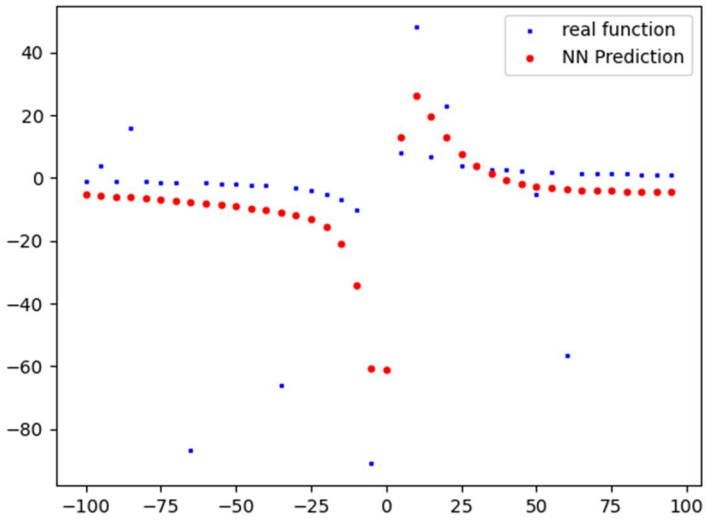
بخش دوم

در این بخش بدون تغییر ساختار شبکه عصبی، به مثال های بالا مقداری نویز اضافه کردیم؛ اما برای سهولت کار شبکه عصبی، دامنه داده های آموزشی و آزمایشی را برابر درنظر گرفتیم:

1. درصد نویز : 10%

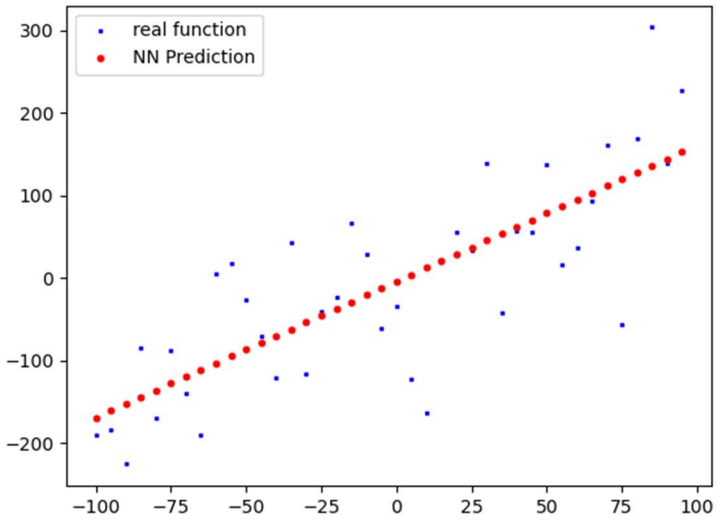


2. درصد نویز : 15%



3. درصد نویز : 90%

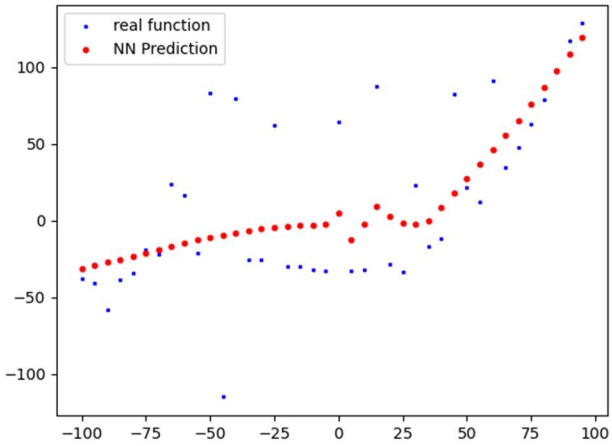
همانطور که در این مثال مشاهده می کنیم، به دلیل ساختار ویژه شبکه (استفاده از ماتریس همانی به عنوان تابع فعالساز نورون ها)، با اینکه داده های آموزشی تنها نقاط آبی رنگ هستند، و شبکه هیچ درکی درباره اینکه ممکن است نقاط اصلی به صورت دیگری باشند، اما تابع تخمین زده شده اصلا تغییر نمی کند. این موضوع قدرت بی نظیر شبکه عصبی در کاربردهایی مانند رگرسیون خطی را نشان می دهد.



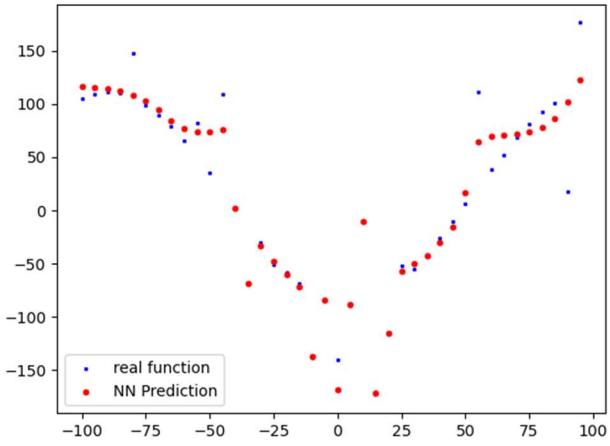
4. درصد نویز : 40%

در این مثال برای بهتر نمایش داده شدن نمودار، ضرایب تابع اصلی را کمی تغییر دادیم:

0.0001 * x**3 + 0.0095 * x**2 – 33



5. درصد نویز : 25%



بخش سوم

در این بخش تنها به توضیح مثال های طراحی شده می پردازیم و یک نمونه از خروجی برنامه ها را می آوریم. در تمامی مثال های این بخش تعداد نقاط آموزشی برابر 30 و تعداد نقاط آزمایشی برابر 60 عدد می باشد. بازه تغییرات داده های آزمایشی نیز 2 برابر دامنه داده های آموزشی می باشد. عدد چاپ شده در خروجی هر برنامه، نشانگر lose function ای است که برای آن مثال در نظر گرفته شده است. lose function ای که از آن استفاده کردیم، برای تمام مثال ها «مجموع اختلاف نرم بردار خروجی شبکه از بردار اصلی» می باشد. (نرم فروبنیوس):

1. تابع مورد استفاده در این مثال، از فضای 10 بعدی به 20 بعدی است؛ به این صورت که یک بردار 10 تایی را ورودی گرفته، درایه های آن را بر 10 تقسیم می کند، و در نهایت بردار بدست آمده را به انتهای بردار اولیه append می کند تا برداری 20 تایی بوجود آید. دقت شبکه در این مثال، خروجی lose function را به اردر 0.05 می رساند.

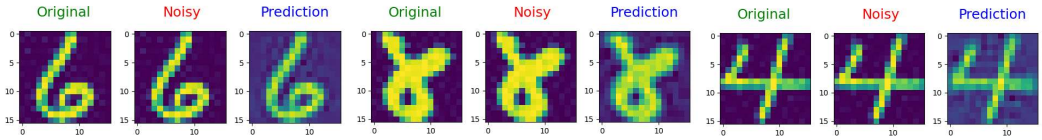
2. تابع مورد استفاده در این مثال، از فضای 10 بعدی به 5 بعدی است؛ به این صورت که یک بردار 10 تایی را ورودی گرفته، 5 درایه آخر آن را حذف می کند تا یک بردار 5 تایی باقی بماند دقت شبکه در این مثال، خروجی lose function را به اردر 0.06 می رساند.
3. تابع مورد استفاده در این مثال، از R**10 به R**10 است؛ به اینصورت که در ابتدا یک ماتریس A= 10*10 با درایه های تصادفی و بردار تصادفی b را تولید می کند، سپس به ازای هر بردار مانند x، حاصل را Ax+b خروجی می دهد. بنابراین شبکه عصبی باید بردارهای 10 تایی را خروجی بدهد. دقت شبکه در این مثال از اردر 0.1 می باشد.
4. تابعی که در این مثال از آن استفاده کردیم، در واقع تابع sort است؛ یعنی با بردار ورودی مانند یک آرایه 10 تایی برخورد می کند و مرتب شده این آرایه را خروجی می دهد. lose function در این برنامه از اردر 100 می باشد که عدد نسبتا بزرگی می باشد.
5. در این مثال، تابع مورد استفاده یک بردار 10 تایی را می گیرد و جمع درایه های آن را خروجی می دهد. بنابراین lose function را در این مثال برابر با «مجموع مربعات اختلاف خروجی شبکه با خروجی واقعی» تعریف کرده ایم. دقت برنامه از اردر 10**-5 است.

بخش پنجم

در این بخش از همان پایگاه داده USPS استفاده کردیم و تصاویر را بصورت grayscale لود کردیم. با این کار هر تصویر متناظر با یک ماتریس 16*16 می باشد که هر درایه این ماتریس عددی بین 0 تا 256 است. سپس با کنار هم قرار دادن ستون های این ماتریس، آن را به یک بردار 256 تایی تبدیل کردیم. نکته مهم اینکه برخلاف بقیه بخش های این پروژه، در این بخش از ماژول MLPClassifier استفاده کردیم؛ نه MLPRegressor. برای تعیین برچسب و دسته هر تصویر نیز، از اولین کاراکتر موجود در نام هر فایل استفاده می کنیم. طبق تجربه، استفاده از تابع logistic به عنوان تابع فعال ساز نورون ها دقت را افزایش می دهد.
همچنین افزایش تعداد لایه های نهان شبکه و تعداد نورون های هر کدام از این لایه ها ابتدائا بر افزایش دقت شبکه تاثیر گذارند اما این تاثیر پس از رسیدن این تعداد به حد معینی، کمتر می شود. برای مثال استفاده از یک لایه پنهان با 1000 نورون، دقت را به 94% افزایش می دهد، اما استفاده از دولایه 1000 نورونی، دقت را حتی به 92.5٪ کاهش می دهد. البته لازم به ذکر است که افزایش تعداد لایه ها و نورون ها، تاثیر شگرفی بر افزایش زمان اجرای برنامه می گذارد.

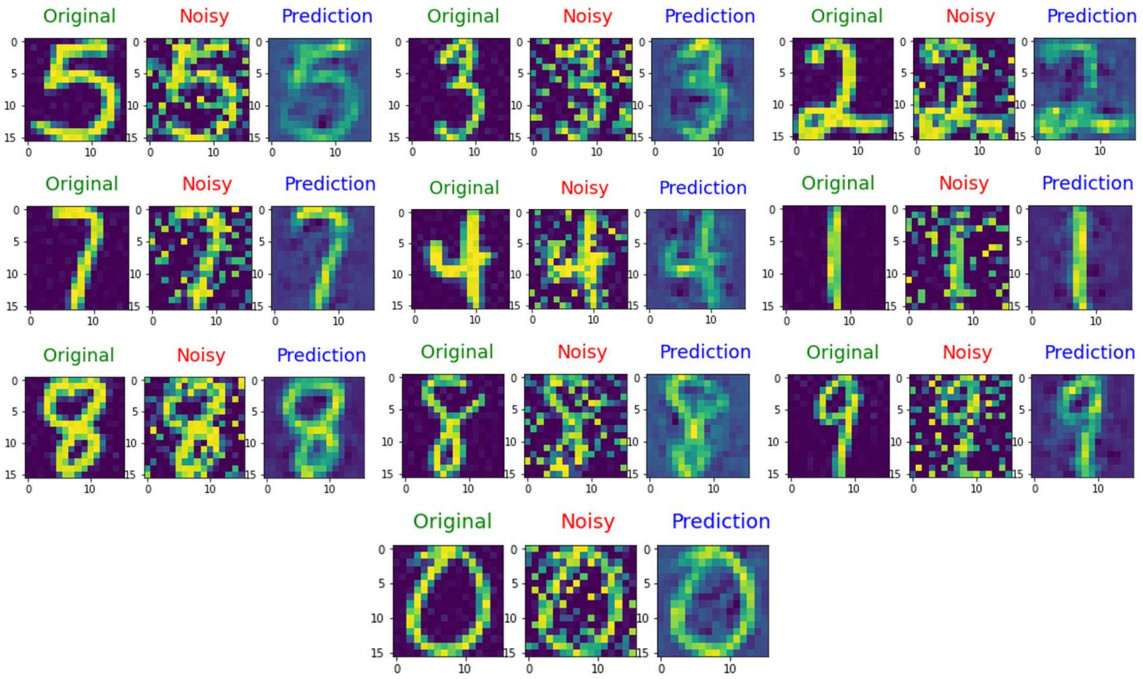
بخش ششم

در این بخش تمام تصاویر پایگاه داده USPS را، چه در پوشه test و چه در پوشه train ، لود می کنیم و درون آرایه ای می ریزیم. سپس از تصاویرمان یک کپی تهیه می کنیم، به آن اندکی نویز اضافه می کنیم و در نهایت همه آنها را نیز درو آرایه دیگری می ریزیم. حال با کمک تابع train_test_split که یکی از توابع موجود در کتابخانه sklearn.model_selection می باشد؛ این دو آرایه را به دو بخش آموزشی و آزمایشی تقسیم می کنیم. اینکه نسبت این دو دسته چگونه باشد، توسط train_test_split و بصورت تصادفی تعیین می شود (این تابع داده ها را shuffle نیز می کند). پس از آموزش شبکه، 5 عدد از این تصاویر را نمایش می دهیم، به این صورت که خروجی شامل 3 عکس است که از چپ به راست به این شرح می باشند: تصویر اصلی – تصویر نویز دار – تصویر حاصل از خروجی شبکه
اینکه آیا شبکه می تواند نویز تصویر را کاهش دهد یا خیر، سوال دشواری است. برای مثال خروجی شبکه، هنگامی که هیچ نویزی در تصویر ایجاد نمی کنیم (یعنی تصویر چپ و وسط یکسان هستند)، در شکل زیر نشان داده شده است:



اگر بخواهیم با دقت صحبت کنیم، باید بگوییم شبکه عصبی خودش نویز ایجاد کرده است و عامل فساد شده است!

اما ارزش واقعی شبکه عصبی وقتی مشخص می شود که فقط اندکی نویز به تصاویرمان اضافه شود. تصاویر زیر حاصل اضافه شدن تنها 30٪ نویز به تصاویر واقعی هستند:



برای بهتر مشخص شدن قدرت شبکه، سعی کنید فقط به تصویر میانی نگاه کنید. در بعضی موارد تشخیص اینکه این تصویر مربوط به چه رقمی است، برای چشم انسان نیز سخت و بعضا غیرممکن است. اما شبکه خروجی قابل تشخیص و بعضا بسیار دقیقی به ما می دهد.

نحوه تولید این نویز به این صورت بوده است که به ازای هر درایه بردار 256 تایی مربوط به یک تصویر، یک عدد تصادفی بین 0 و 1 تولید می کنیم، اگر این عدد بزرگتر از کوچکتر از 0.3 باشد، مقدار آن درایه را به عددی تصادفی بین 1 تا 256 تغییر می دهیم.

خروجی شبکه نیز به این صورت است که 20 تصویر متفاوت، مشابه تصاویر بالا به عنوان مشتی نمونه خروار از خروجی های شبکه را نمایش می دهیم.

جالب است که زمان اجرای برنامه را با ویژگی های زیر بررسی کنیم:

شبکه عصبی با یک لایه پنهان 1000 نورونی : 12 دقیقه - شبکه عصبی با یک لایه پنهان 1000 نورونی : 41 دقیقه - شبکه عصبی با یک لایه پنهان 1000 نورونی : 70 دقیقه

پایان