بسمه تعالی

دانشکده مهندسی مکانیک

پردیس دانشکده های فنی

دانشگاه تهران

هوش مصنوعی

پروژه 5

دانشجو:

شهریار نامداری

810098043

استاد: دکتر فدایی

خرداد 1401

بخش اول:

- توضیحات در فایل notebook آمده است.

بخش دوم:

- بخش های ناقص کد تکمیل شد.

بخش سوم:

قسمت اول:

شبکه به کمک مقدار دهی اولیه نرمال برای وزن‌ها آموزش داده شد و نتیجه آخرین ایپاک چنین شد:

```
Epoch 15:
        Train: Average Accuracy: 0.09835189527159047    Average Loss: 0.03125341930441578
        Test: Average Accuracy: 0.09854364809782609     Average Loss: 1.0
```

قسمت دوم:

اگر تمامی وزن های اولیه مقدار صفر داشته باشند، چون مقدار همگی این وزن ها یکسان است، در نتیجه مقدار وزن های درون یک لایه در به روز رسانی هر مرحله یکسان خواهد بود و یک وزن های یک لایه همیشه یکسان میماند که این مطلوب نیست و از قابلیت شبکه در حل مسائل پیچیده میکاهد. اما هنگام وزن دهی اولیه رندوم، این اتفاق نمی‌افتد و شبکه میتواند در یک لایه وزن های متفاوتی داشته باشد و به همین دلیل توانایی حل مسائل پیچیده و غیر خطی را پیدا میکند. در نهایت قابل ذکر است که این مشکل وزن دهی اولیه صفر فقط برای مقدار صفر نیست بلکه برای هر وزن دهی اولیه یکسان به همه لایه ها پیش می‌آید.

قسمت سوم:

پس از مقداری آزمون و خطا به نظر مقدار بهینه برای نرخ یادگیری مقدار 0.001 میباشد. نتیجه با این نرخ:

```
Epoch 15:
        Train: Average Accuracy: 0.09835189527159047    Average Loss: 0.03125341930441578
        Test: Average Accuracy: 0.09854364809782609     Average Loss: 1.0
```

با نرخ یادگیری 0.001

نتیجه با مقادیر بیشتر و کمتر نیز چنین است:

```
Epoch 15:
        Train: Average Accuracy: 0.09835189527159047    Average Loss: nan
        Test: Average Accuracy: 0.09854364809782609     Average Loss: nan
```

با نرخ یادگیری 0.1

```
Epoch 15:
        Train: Average Accuracy: 0.09835189527159047    Average Loss: nan
        Test: Average Accuracy: 0.09854364809782609     Average Loss: nan
```

با نرخ یادگیری 0.01

```
Epoch 15:
        Train: Average Accuracy: 0.964741109808519      Average Loss: 0.001522580029313832
        Test: Average Accuracy: 0.9622664741847826      Average Loss: 0.05949449749171656
```

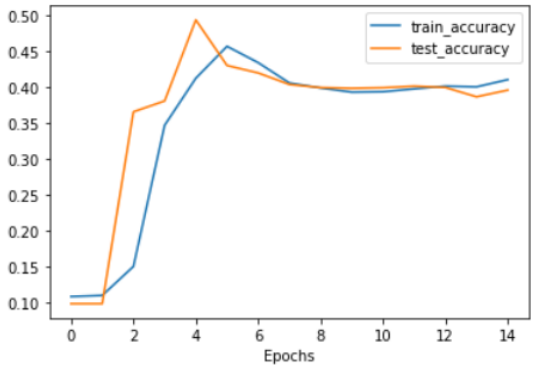
با نرخ یادگیری 0.0001

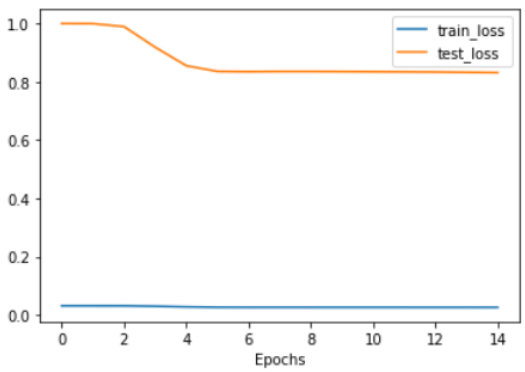
با نرخ یادگیری بیشتر نسبت به حالت بهینه، نتایج تغییری نکرد و مقدار loss برای حالت بیشتر به مقدار nan تغییر کرد. با نرخ یادگیری کمتر نتایج بهتری نسبت به حالت بیشتر به دست می‌آوریم ولی حالت بهینه همان مقدار 0.001 میباشد. در کل نرخ بالا سرعت یادگیری را افزایش میدهد ولی دقت را کاهش داده و گاهی مشکل ساز میشود. قابل ذکر است مواردی که نمودارشان رسم نشده به این دلیل بوده که در تمام ایپاک ها تغییری در دقت و خطا حاصل نشده و نمودار ثابت میبود.

قسمت چهارم:

تابع فعالساز sigmoid :

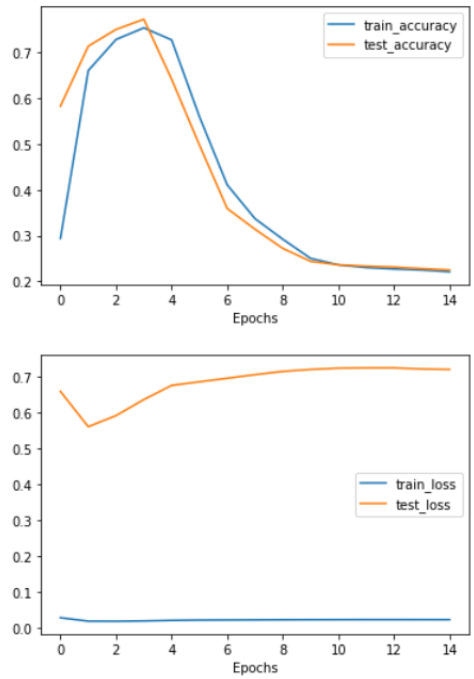
Sigmoid:





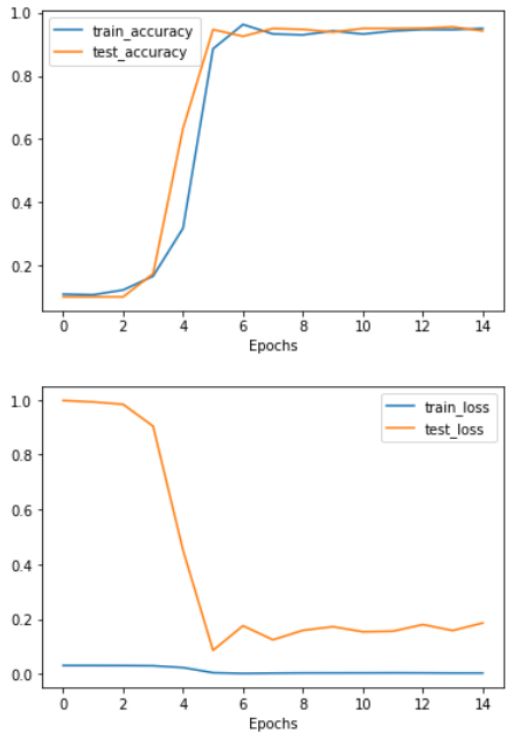
تابع فعالساز Tanh :

Tanh:



تابع فعالساز LeakyRelu :

Leaky Relu:



همانطور که از نتایج مشخص است توابع tanh و sigmoid خوب کار نکرده‌اند ولی در مقابل آن ها تابع leakyrelu به خوبی کار کرده و توانسته دقت خوبی را حاصل کند.

دلیل کار نکردن توابع sigmoid و tanh در این مساله:

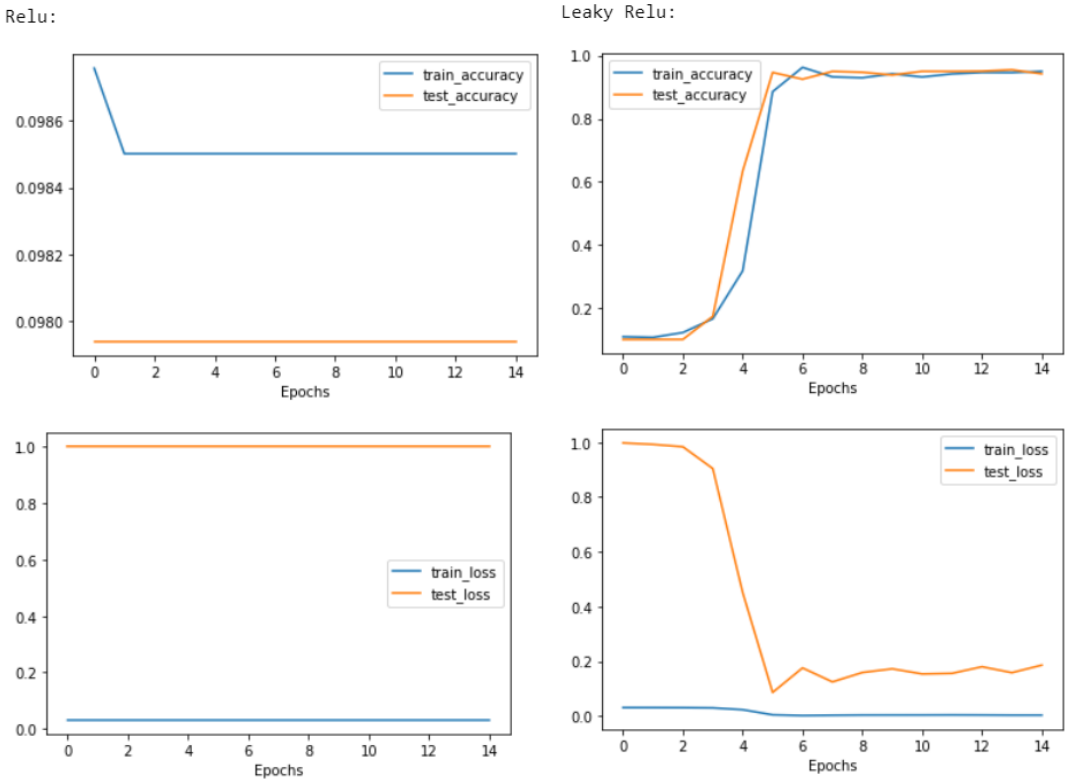
در تابع sigmoid به دلیل شکل ظاهری تابع و اینکه وقتی به بی نهایت میل میکنیم شیب تقریبا ثابت میشود در نتیجه تغییرات اندکی در شیب و مقدار تابع خواهیم داشت که این مورد باعث میشود تا یادگیری ما و مشتقات متوالی در لایه ها با مشکل مواجه شود. همچنین در sigmoid مقدار mean activation نزدیک به صفر نیست در نتیجه آموزش کندتر میشود. همچنین مشکل پیچیدگی مدل و هزینه زمانی بالا را برای تابع فعالساز Sigmoid داریم. برای تابع فعالساز tanh فقط مشکل mean zero را نداریم و باقی مشکلات ذکر شده برای sigmoid وجود دارد.

برتری LeakyRelu نسبت به Relu :

یکی از مشکلات Relu مشکل Dying Relu است. یعنی زمانی که در قسمت منفی گیر میکنیم و شبکه مدام مقدار صفر را باز میگرداند. استفاده از LeakyRelu دو مزیت دارد:

1. مشکل dying relu را حل میکند.
2. تابع LeakyRelu کمک کننده به آموزش سریع تر است. میدانیم اگر مقدار mean activation نزدیک به صفر باشد آموزش سریع تری خواهیم داشت. همچنین تابع LeakyRelu نسبت به Relu بالانس تر است که در نتیجه به یادگیری کمک میکند.

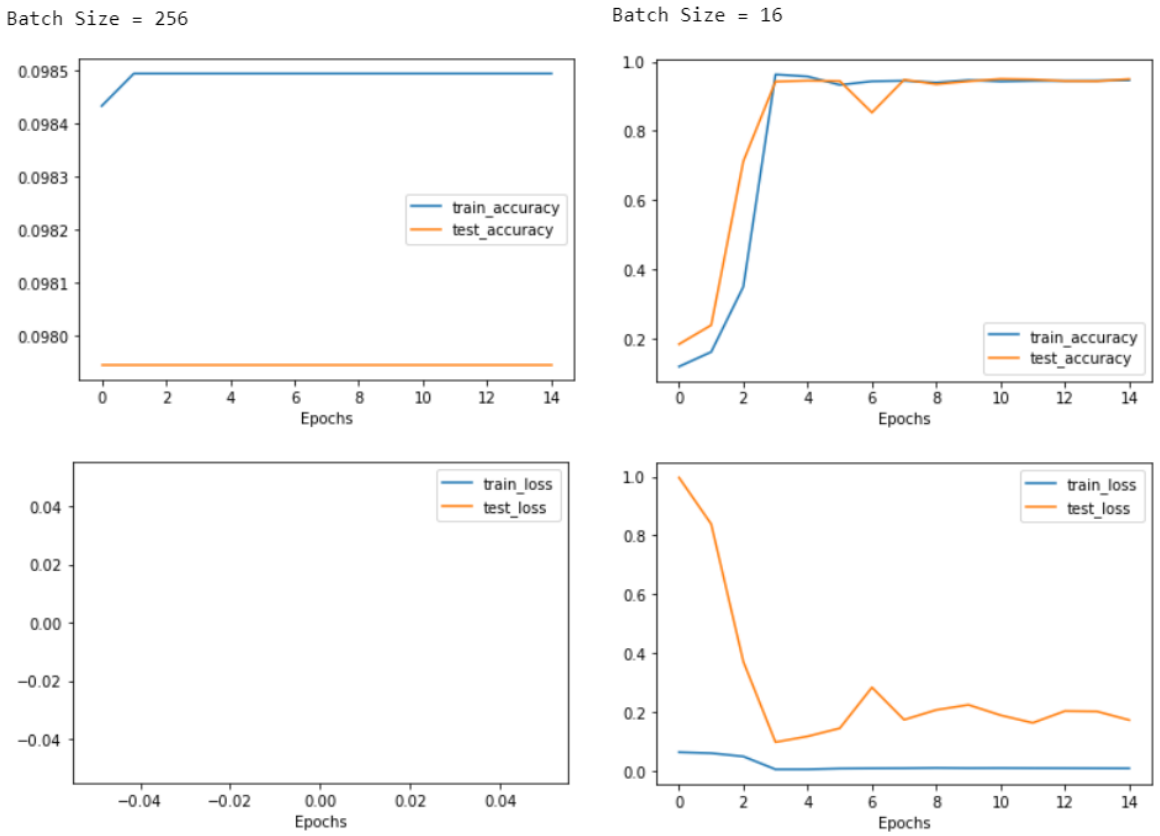
مقایسه نمودار نتایج Relu و LeakyRelu در کنار یکدیگر:



پس با توجه به نتایج در ادامه از تابع فعالساز Leaky Relu استفاده میکنیم.

قسمت پنجم:

نتیجه استفاده از batch size با مقدار 16 و 256 در زیر آمده است:



همانطور که مشاهده میشود برای سایز 16 نسبت به حالت 32 نتیجه تقریبا تغییری نکرده اما برای سایز 256 نتیجه جالبی حاصل نشده و دقت به شدت کاهش یافته است.

علت استفاده از batch در فرایند آموزش:

با استفاده از batch میتوانیم مقداری سرعت آموزش را افزایش دهیم زیرا دیگر شبکه هر داده را جداگانه منجر به آپدیت وزن ها نمیکند(استفاده نکردن از batch مانند استفاده از batch با سایز 1 است). همچنین از دچار شدن به خطای داده های نویز دار جلوگیری میکنیم و مقداری از این خطا دوری میکنیم.

معایب استفاده از batch size خیلی کوچک:

1. باعث میشود داده ها دچار نویز بالا باشد و در نتیجه به نقطه optimal نرسیم.
2. سرعت آموزش پایین می‌آید.

3. دچار sample bias میشویم و جهت گیری خاصی نسبت به کلاس خاصی ایجاد میشود. زیرا در تعداد داده کم پخش کردن یکسان داده ها از هر کلاس دشوار تر است و باید مدام با این مشکل سر و کله زد.

معایب استفاده از batch size خیلی بزرگ:

1. برعکس حالت قبل با افزایش زیاد سایز batch تعداد به روز رسانی های وزن ها کاهش میباد. و تعداد ایپاک بیشتری نیاز است تا شبکه به آن مقدار مطلوب خود برسد اما با batch size با مقدار مشخص و متوسط میتوان با تعداد ایپاک معقول به نتیجه رسید.