پروژه 5 هوش مصنوعی فاز1

طنین زراعتی 810197627

نام پروژه :پیاده سازی و آزمون شبکه های عصبی Feed Forward.

هدف :حدس زدن کلاس تصاویری که اطلاعات پیکسل هایشان موجود است به وسیله شبکه عصبی پس از یادگیری بر روی تصاویر با کلاس مشخص.

شرح کلی :هر تصویر ابتدا مسطح شده و به صورت بردار به عنوان ورودی به شبکه عصبی داده می شود. هر درایه این بردار (معادل با یک پیکسل تصویر) یک ویژگی برای آن تصویر محسوب می شود. شبکه قرار است بر اساس این ویژگی ها و با ساختن ترکیبات غیرخطی از آن ها، وزن اتصالات بین لایه هایش را طوری تنظیم کند که خروجی آن ضمن داشتن کمترین خطا کلاس تصویر ورودی متناظر را به درستی تشخیص دهد.

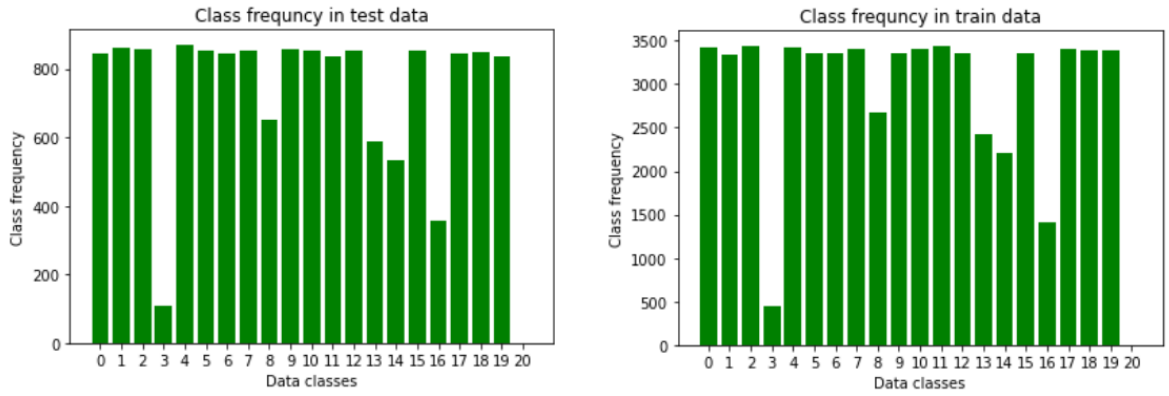
بخش های پروژه:

قسمت اول: بررسی و پیش پردازش داده ها

1.به طور رندوم از بین کلاس ها یک کلاس را انتخاب کرده و یکی از داده های آن را پرینت میکنیم.مقادیر پیکسل ها بین 0 تا 255 است.

2.از هرکلاس اولین مقدار مشاهده شده به همراه کلاس آن نمایش داده شده است.

3.



4. نرمالایز کردن داده ها سرعت یادگیری را افزایش می دهد و باعث می شود بردار وزن ها سریع تر همگرا شود. علت دیگر این است که اگر این کار را نکنیم در هنگام مشتق گرفتن ها ممکن است به علت استفاده از توابع نمایی سرریزOverflow رخ دهد.

در نرمال کردن میانگین را 0 (به صفر نزدیک میکنیم) میکنیم و یکی از دلایل استفاده از آن این است که اگر تمام داده ها مثبت و یا منفی باشند یادگیری را تا حدی سخت میکند(مخصوصا در روش هایی مثل relu که داده های منفی را صفر در نظر میگیرد) پس وقتی میانگین را صفر میکنیم یعنی بخشی از داده مثبت و بخشی منفی است.

همچنین این اعدادی که بین صفر و یک هستند می توانند به عنوان احتمال در نظر گرفته شوند.

در این جا از روش ماکسیموم (255) استفاده شده است که با توجه به سرریز هایی که در طول پروژه مشاهده شد به جای 255اعداد بر 2550تقسیم شده اند.

قسمت دوم: تکمیل بخش های ناقص شبکه عصبی

شبکه ای با دو لایه مخفی که به ترتیب 40و 20نورون دارند با شیوه وزن دهی uniformطراحی شده است. با توجه به سرریز هایی که در طول پروژه مشاهده شد وزن دهی uniformبه صورت اعداد تصادفی در بازه صفر تا 0.05تعریف شده است.

قسمت سوم: طبقه بندی داده ها:

Batch Size : تعداد سمپل هایی که در شبکه propagate میشود را تعیین میکند.

بهتر است batch size از تعداد کل سمپل ها کمتر باشد چراکه به حافظه کمتری نیاز دارد. از آنجا که شبکه را با استفاده از نمونه های کمتری آموزش می دهید ، روش کلی آموزش به حافظه کمتری نیاز دارد. این امر به ویژه اگر نتوانید کل مجموعه داده را در حافظه دستگاه خود جا دهید بسیار مهم است.

همچنین سرعت کار نیز بالا میرود.

از آنجایی که مقدار کم batch size بسیار noise دارد درنتیجه خطای تعمیم کمتری نیز دارد و راحت تر هم fit میشود.

البته از یک حدی هم کمتر مناسب نیست چون تعداد آپدیت ها کافی نیست.(جلوتر در مقایسه batch size های مختلف توضیحات بیشتر ارائه شده است)

Epoch : یک hyper parameter است که تعداد دفعاتی را که الگوریتم یادگیری در کل مجموعه داده های آموزشی کار می کند ، تعریف می کند.به طور معمول این مقدار زیاد است در اوردر 100 و 1000 و .... که این امکان را برای الگوریتم یادگیری ایجاد میکند که error خود را مینیموم کند با این حال اگر از یک حدی بیشتر شود امکان overfittingداریم. همجنین با افزایش تعداد epoch ها زمان اجرا نیز زیاد میشود(خیلی زیاد)

گام یک :در صورتی که شبکه ای با معماری گفته شده طراحی کنیم، با نرخ یادگیری 0.001به دقت زیر می رسیم:

```
Epoch 15:
        Train: Average Accuracy: 0.6453558102345416     Average Loss: 1.2063224945866264
        Test: Average Accuracy: 0.6074024822695034      Average Loss: 1.3650927201856198
Epoch 16:
        Train: Average Accuracy: 0.6602978411513859     Average Loss: 1.1517505233880572
        Test: Average Accuracy: 0.6129654255319149      Average Loss: 1.3444472520782604
Epoch 17:
        Train: Average Accuracy: 0.6647954424307037     Average Loss: 1.1329416505989447
        Test: Average Accuracy: 0.6237367021276595      Average Loss: 1.320304184422743
Epoch 18:
        Train: Average Accuracy: 0.6774720149253731     Average Loss: 1.1022117264934401
        Test: Average Accuracy: 0.625   Average Loss: 1.3042179255299002
Epoch 19:
        Train: Average Accuracy: 0.6829690831556503     Average Loss: 1.0695450626457765
        Test: Average Accuracy: 0.6343085106382979      Average Loss: 1.275973133781085
Epoch 20:
        Train: Average Accuracy: 0.6844183102345416     Average Loss: 1.0768507631587194
        Test: Average Accuracy: 0.6324468085106383      Average Loss: 1.2832798778628325
```

گام دو: اگر وزن 0 بود :

```
Epoch 15:
        Train: Average Accuracy: 0.05660314498933902    Average Loss: 2.954509790272198
        Test: Average Accuracy: 0.05791223404255319     Average Loss: 2.9531551968935066
Epoch 16:
        Train: Average Accuracy: 0.055903518123667374   Average Loss: 2.954382295760801
        Test: Average Accuracy: 0.05702570921985816     Average Loss: 2.9529110219759307
Epoch 17:
        Train: Average Accuracy: 0.056519856076759065   Average Loss: 2.9543177660530215
        Test: Average Accuracy: 0.05702570921985816     Average Loss: 2.952570825800064
Epoch 18:
        Train: Average Accuracy: 0.05438765991471215    Average Loss: 2.954143969625913
        Test: Average Accuracy: 0.05702570921985816     Average Loss: 2.9527513579487668
Epoch 19:
        Train: Average Accuracy: 0.05628664712153518    Average Loss: 2.9540370312099573
        Test: Average Accuracy: 0.05713652482269503     Average Loss: 2.9527901979034517
Epoch 20:
        Train: Average Accuracy: 0.05608675373134328    Average Loss: 2.954012877955187
        Test: Average Accuracy: 0.055873226950354606    Average Loss: 2.9524686067637678
```

با قرار دادن صفر به صورت default از انجا به بعد تمامی neuron ها از ان پیروی میکنند و gradient مشابه خواهند داشت و همان کار را مجددا انجام میدهند و پیشرفتی حاصل نمیشود در نتیجه نسبت به حالت قبل بدتر عمل میکند.

(با توجه به صورت سوال که گفته شده پیاده سازی لازم نیست در کد این بخش نیست و پس از گرفتن نتایج به حالت اول برگرداندم)

گام سه :نتایج حاصل از نرخ های یادگیری 0.001 ,0.01.0.0001 به صورت زیر است:

Learning Rate	Train	Test
0.01	47.7	44.9
0.001	68.4	63.2
0.0001	37.1	29.0

مشاهده می شود که وقتی نرخ یادگیری بالاتر می رود در این تابع خاص به طور کلی عملکرد تابع مختل می شود.

همچنین وقتی این نرخ را کمتر بکنیم نرخ افزایش دقت کمتر می شود به این علت که بردارهای وزن را کم تغییر می دهیم. در حالتی که نرخ یادگیری را یک صدم بکنیم این امر مشهود تر است.

با توجه به نتایج نرخ پرش در 0.01 تا حدود 10 درصد است و برای 0.0001 زیر 10 درصد و برای 0.001 بین 10 تا 20 درصد است.

در حین انجام پروژه، گاها مشاهده شد با زیاد کردن نرخ یادگیری می توان افزایش دقت را تسریع کرد. ولی بعد از همگرا شدن شبکه این زیاد بودن نرخ باعث به هم ریختن مجدد دقت می شد. به این شکل که شبکه را از آن اکسترمم محلی که داخلش بود بیرون می انداخت. البته این مساله

ممکن است برای حالاتی که احتمال همگرا شدن زودهنگام روی یک اکسترمم محلی برای شبکه وجود دارد مفید باشد

گام چهار:

Activation Func	Train	Test
sigmoid	7.8	7.2
Tanh	2.0	1.6
Leaky Relu	68.7	63.4

مشاهده می شود که تابع فعال سازی Reluبهترین دقت را روی داده های آزمون به ما می دهد.
در روند یادگیری تابع Sigmoidنوساناتی مشاهده می شود. توابعی مثل Sigmoidو tanhچون در ازای ورودی های خیلی کم و خیلی زیاد به خطوطی افقی میل می کنند و مشتقشان صفر می شود، از جایی به بعد روند یادگیری را مختل می کنند. بنابراین به جای آنها از توابعی مثل Reluو LeakyReluاستفاده می شود.

: Relu VS Leaky Relu

در Relu مشکلی به اسم "dying relu" داریم که به ازای تمامی مقادیر منفی صفر برمیگرداند در نتیجه شیب relu در محدود منفی نیز صفر در نظر گرفته میشود و به محض اینکه neuron منفی شود احتمال recoveryان کم است. چنین neuron هایی عملا بی فایده هستند. به این ترتیب با گذشت زمان بخش بزرگی از شبکه را سلول هی بی فایده تشکیل میدهند.(اگر learning rate بسیار بالا باشد این مشکل خیلی به چشم میخورد.)

درحالیکه در leaky relu این مشکل را نداریم. همچنین متعادل تر و سرعت بیشتری نیز دارد.

: Sigmoid وTanh

با افزایش تعداد لایه ها و ضرب gradient ها مقادیر کوچکتر از 1 خیلی سریع به 0 میرسند.(به صفر میل میکنند و مقادیر بسیار کوچکی حاصل میشود.) و از انجایی که در deep learning تعداد بیشتر لایه ها (تعداد مشخص و محدود) بسیار مفید است این مشکل مهمی تلقی میشود.

مشکل بالا را در هردو روش داریم.

گام پنج: برای Batch Sizeهای 32, 16 , 256 نتایج تست به صورت زیر است:

Batch Size	Train	test
16	71.9	65.1
32	71	65
64	68.7	63.4
256	6.9	7.6

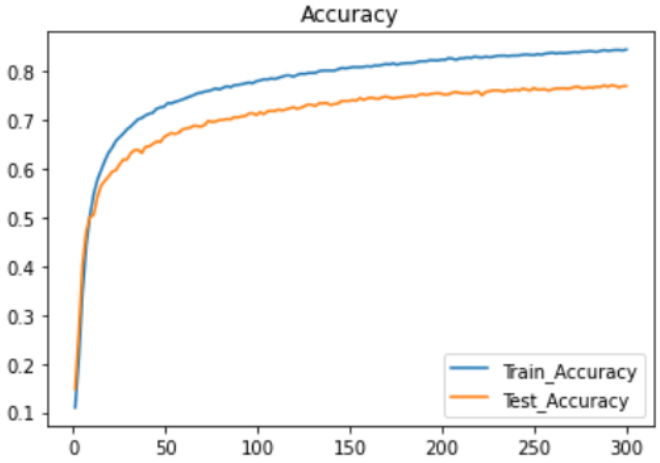
اگر Batch Sizeخیلی کوچک باشد اثر داده های پرت (نویزها) در آن بیشتر به چشم می آید، همچنین ممکن است یک Batchایجاد اکسترمم محلی بکند و بعد از آن روند یادگیری مختل شود.

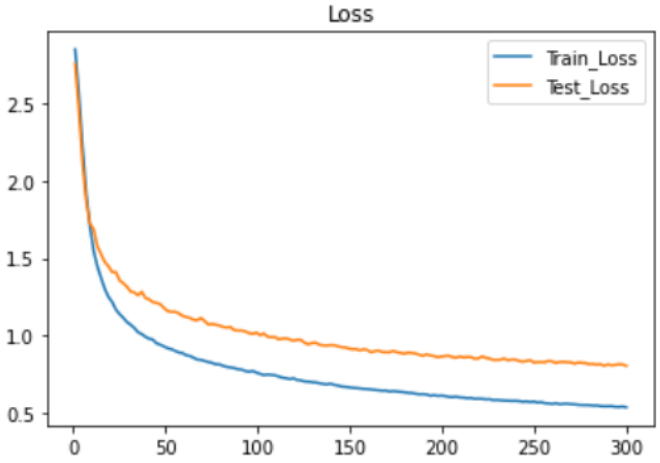
Batch size 16 و 32 بهتر عمل میکنند و سرعت بیشتری نیز دارند.

گام شش: با صرفا یک مرتبه یادگیری وزن های شبکه مقدار زیادی تغییر نمی کنند و در نتیجه انتظار دقت بالایی نمی توان داشت. باید آموزش تا جایی که دقت در حال افزایش است ادامه داده شود. از طرفی اگر این زیاد همِلییات یادگیری و به روز رسانی وزن ها را انجام دهیم تمام نویزهای موجود در داده های آموزش هم در وزن دهی دخیل می شوند و باعث می شود دقت شبکه روی داده های آزمون به خاطر درست فرض کردن این داده های غلط کم شود. همچنین مشاهده می شود که مقدار تابع Lossرفته رفته روی داده های آزمون زیاد می شود که مورد نامطلوبیست.

تعداد epoch هارا تا حدی باید زیاد کنیم که شبکه با داده های train سازگار نشود و overfitting رخ ندهد. با توجه به نتایج بدست آمده از 150 epoch تایی زمانی که overfitting رخ میدهد مشاهده میکنیم فاصله دقت افزایش میابد و روی داده train بهتر و بهتر عمل میکند درحالی که روی test به خوبی train عمل نمیکند. در نتیجه نباید epoch را بیشتر افزایش دهیم و تا جایی افزایش میدهیم که overfitting نداشته باشین.

روند تغییر دقت روی داده های آموزش و آزمون و همچنین هزینه شبکه روی این دو مجموعه داده به صورت زیر است.


Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
Train_Accuracy
Test_Accuracy


Loss
2.5
2.0
1.5
1.0
0.5
0
50
100
150
200
250
300
Train_Loss
Test_Loss