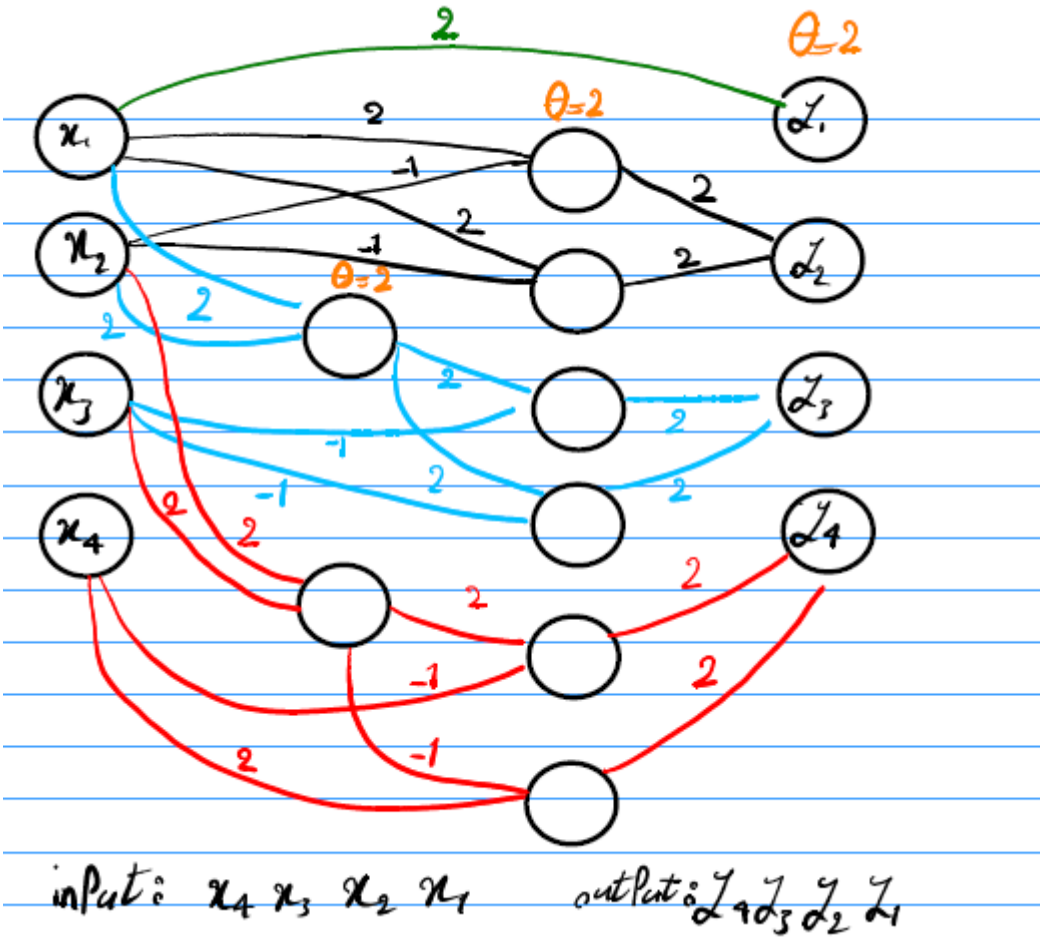
به نام خدا
دانشگاه تهران
دانشکده مهندسی برق و
کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین اول

امیرعباس رضا سلطانی	**610399205**
نیما نیرومند	610399199

پرسش 1. Mcculloch Pitts

۱-۲. همانطور که در شکل صورت سوال دیده می شود، خروجی اول به طور مستقیم از ورودی اول حاصل می شود، خروجی دوم، xor ورودی اول و ورودی دوم است، و به همین ترتیب خروجی های بعدی، xor بیت متناظر در ورودی و or بیت های قبلی ورودی است. بنابراین شبکه سه لایه به صورت زیر داریم:



۱-۳.

برای پیاده سازی آن ابتدا یک کلاس MccullochPittsNeuron را تعریف می‌کنیم که دارای تابع‌های زیر می‌باشد.

تابع __init__:

در آن وزن‌ها و ترشولد نورون را ورودی می‌گیریم و برای نورون آها در کلاس نگه می‌داریم

تابع outpt: به ازای یک ورودی با توجه به وزن‌های نورون ورودی نورون را پیدا و سپس با توجه به ترشولد داده شده خروجی را حساب می‌کنیم و برمی‌گدانیم

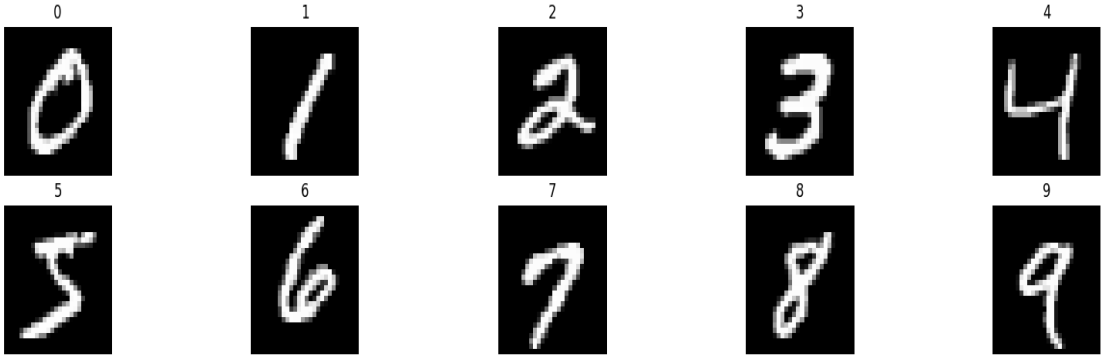
حال پس از تعریف کلاس تابع complement2 را تعریف می‌کنیم که مطابق ساختاری که در بخش قبل سوال به دست آوردیم و با استفاده از کلاسی که تعریف کردیم دونه دونه نورون‌ها به وجود می‌اوریم و خروجی آن ها را محسابه میکنیم تا y1,y2,y3,y4 محاسبه شود و آن را باز گردانیم

در انتهای برای تمام اعداد بارنتی ۴ رقمی این تابع را فراخوانده و مکمل دوم آن‌ها را خروجی می‌دهیم.

```
The twos complement of 0000 is: 0000
The twos complement of 0001 is: 0000
The twos complement of 0010 is: 1000
The twos complement of 0011 is: 0000
The twos complement of 0100 is: 1100
The twos complement of 0101 is: 0100
The twos complement of 0110 is: 1000
The twos complement of 0111 is: 0000
The twos complement of 1000 is: 0111
The twos complement of 1001 is: 0111
The twos complement of 1010 is: 1011
The twos complement of 1011 is: 0011
The twos complement of 1100 is: 1101
The twos complement of 1101 is: 0101
The twos complement of 1110 is: 1001
The twos complement of 1111 is: 0001
```

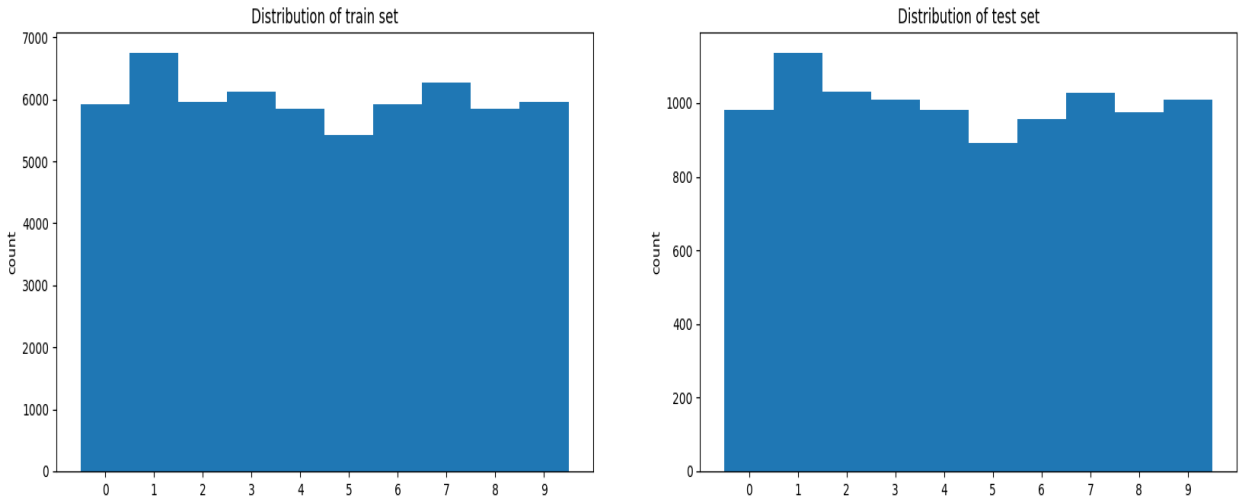
.

2-2. داده‌های تست متشکل از **60000** تصویر train در مقیاس gray scale با ابعاد 28 در 28 و داده‌های train متشکل از **10000** تصویر test با ابعاد 28 در 28 در مقیاس gray scale می‌باشد.



شکل . تصویر هر برچسب از دیتا

طبق شکل زیر، توزیع برچسب‌ها یکنواخت است. درصورت عدم یکنواختی توزیع، عمل دسته بندی تصاویر ممکن است با overfitness همراه شود. زیرا خطوط جداگر برچسب‌ها به سمت کلاس‌ها با تعداد داده‌های کمتر متمایل می‌شوند. اما دراینجا داده‌ها یکنواخت هستند و نیازی به بالانس کردن توزیع داده‌ها نیست.



شکل . توزیع برچسب‌های دیتاست

داده‌های در بازه 0 تا 255 هستند. به دو علت مقدار داده‌های تصویر را به مقادیر بین صفر و یک نگاشت می‌کنیم. اولا اینکه میان تصویرهای ارائه شده، پیکسل‌هایی وجود دارند که در اغلب تصاویر مقدار صفر دارند اما بقیه پیکسل‌ها مقادیرشان اغلب به 255 نزدیک‌تر است. بنابراین این اختلاف در

بازه باعث می‌شود که بعضی پیکسل‌ها در فرایند یادگیری غالب باشند. همچنین مقادیر بزرگ ورودی در آستانه یک نورون در صورت استفاده از توابع فعالساز tanh یا sigmoid باعث اشباع شدن آن توابع و در نتیجه آپدیت نشدن مقادیر وزن‌های متناظر می‌شود. تفاوت در مقادیر پیکسل‌ها باعث حساس شدن شبکه نسبت به وزن‌دهی اولیه نیز می‌شود.

2-3. هر عکس ورودی از ابعاد 28 در 28 می‌باشد. بنابراین در لایه ورودی نیاز به استفاده از 28×28 نورون داریم.

خروجی شبکه عصبی را logit می‌گوییم که میتواند هر عدد حقیقی را اختیار کند. اما در این سوال، که به ازای هر برچسب، یک نورون در لایه خروجی داریم، می‌خواهیم خروجی هر نورون، مقادیر احتمالی بین صفر و یک باشند که نشان‌دهنده احتمال تعلق ورودی به کلاس مربوطه است. بنابراین مقادیر خروجی هر نورون لایه خروجی را از activation function زیر گذر می‌دهیم.

$$sigmoid(out_j) = \frac{exp(out_j)}{\sum_{i=0}^{9} exp(out_i)}$$

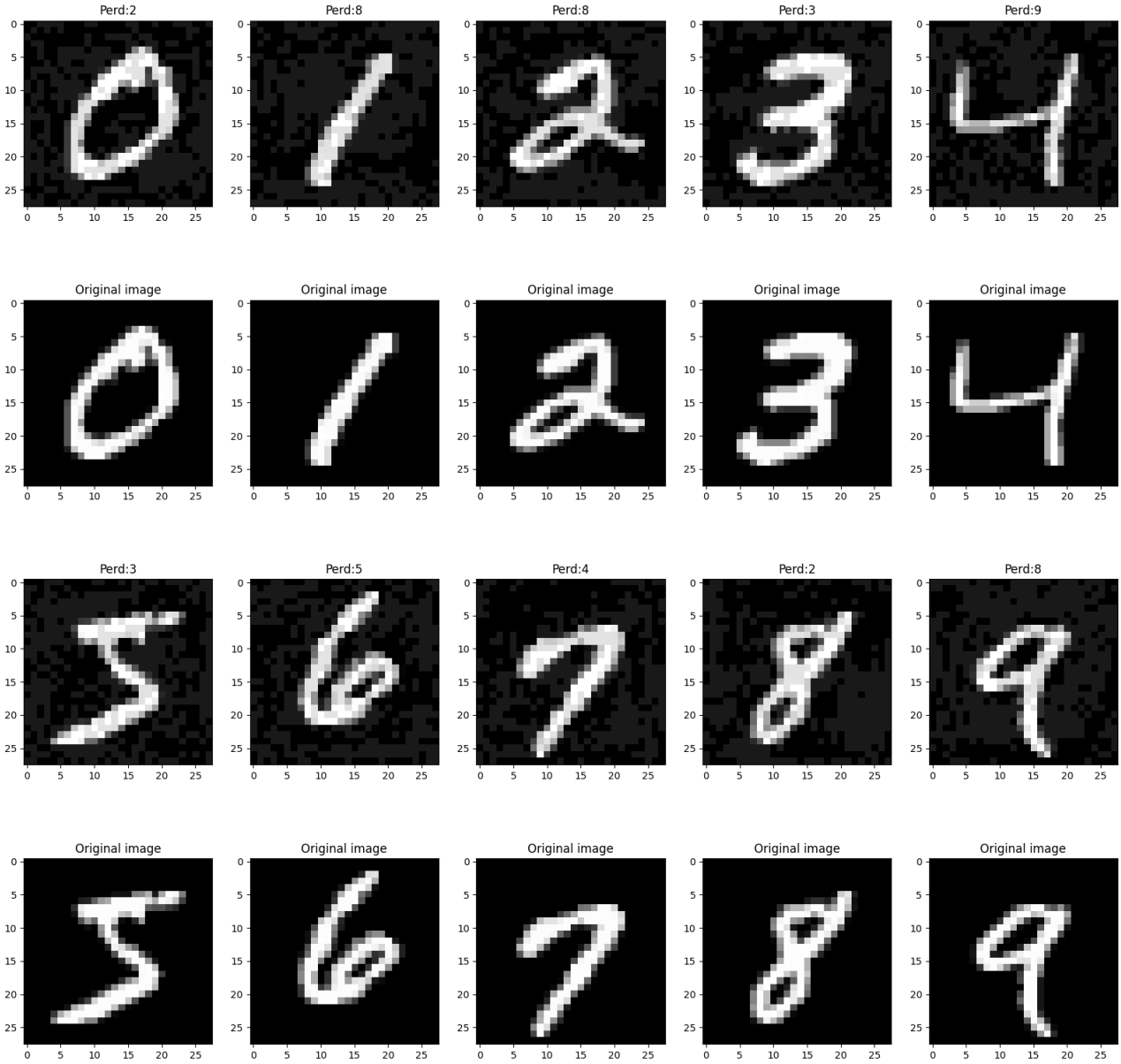
که در آن out_i خروجی نرون i در لایه خروجی است.

2-4.

جدول . پارامترهای حمله FGSM

epsilon	0.1

پس از اعمال حمله FGSM و محاسبه دقت پیش‌بینی مدل، حدود 10 درصد، مدل تشخیص صحیح داده است. دقت مدل قبل از حمله 91 درصد بود. پس حمله 81 درصد موثر واقع شده است.



شکل . نمونه تصاویر pertubed شده با حمله FGSM در کنار تصاویر اصلی آن‌ها

2-5.

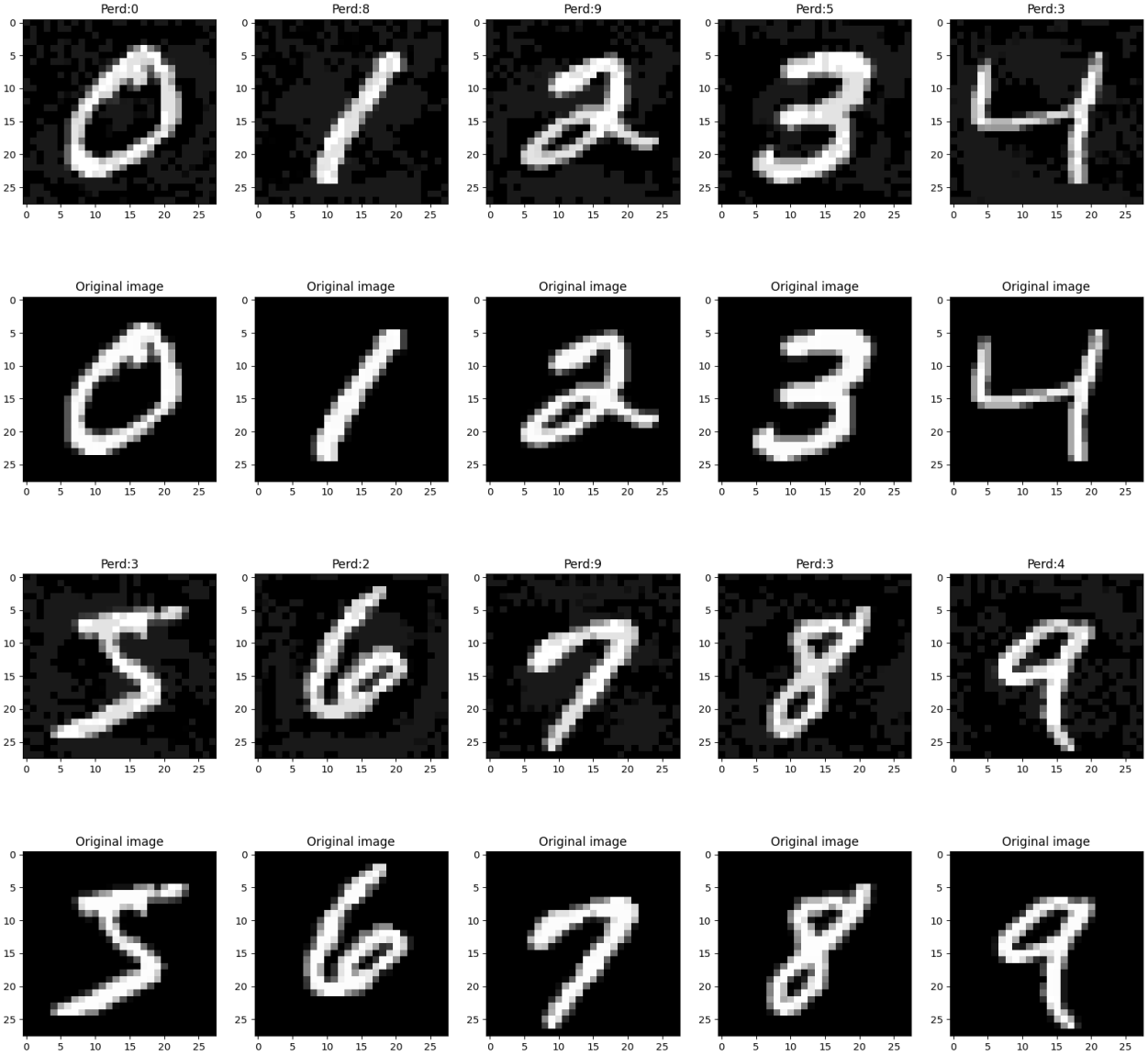
جدول . پارامترهای حمله PGD

epsilon	0.1
alpha	0.005
iterations	20

پس از اعمال حمله PGD و محاسبه دقت پیش‌بینی مدل، حدود 6 درصد، مدل تشخیص صحیح داده است و 85 درصد مواقع، حمله، موثر واقع شده است.

در این روش برخلاف روش FGSM، بهینه‌سازی از روش **greedy** صورت می‌گیرد. در روش FGSM، صرفا در یک گام، در جهت گرادیان حرکت می‌کنیم تا مقدار loss در بیشترین جهت افزایش یابد. اما در روش PGD در چند iteration و به صورت greedy در جهت گرادیان برای افزایش loss حرکت می‌کنیم. بنابراین از پارامتر آلفا برای تنظیم میزان حرکت در خلاف جهت گرادیان استفاده می‌کنیم که در FDSM موجود نیست.

استفاده از روش greedy در PGD، مزیتی نسبت به FGSM محسوب می‌شود. زیرا با این روش بهینه سازی در گام های متوالی به دنبال بهترین جهت تغییر تصویر برای افزایش **loss** حرکت می‌کنیم، اما در FGSM در یک گام، و به دلخواه در جهت افزایش loss حرکت می‌کنیم و تلاشی برای حرکت در بهترین مسیر برای افزایش loss نمی‌شود. به عنوان شاهد، همان‌طور که از نتایج مشاهده شد، به ازای اپسیلون (میزان اثر حمله) یکسان، اثر حمله حدود 4 درصد افزایش پیدا کرد.



شکل . نمونه تصاویر pertubed شده با حمله PGD در کنار تصاویر اصلی آن‌ها

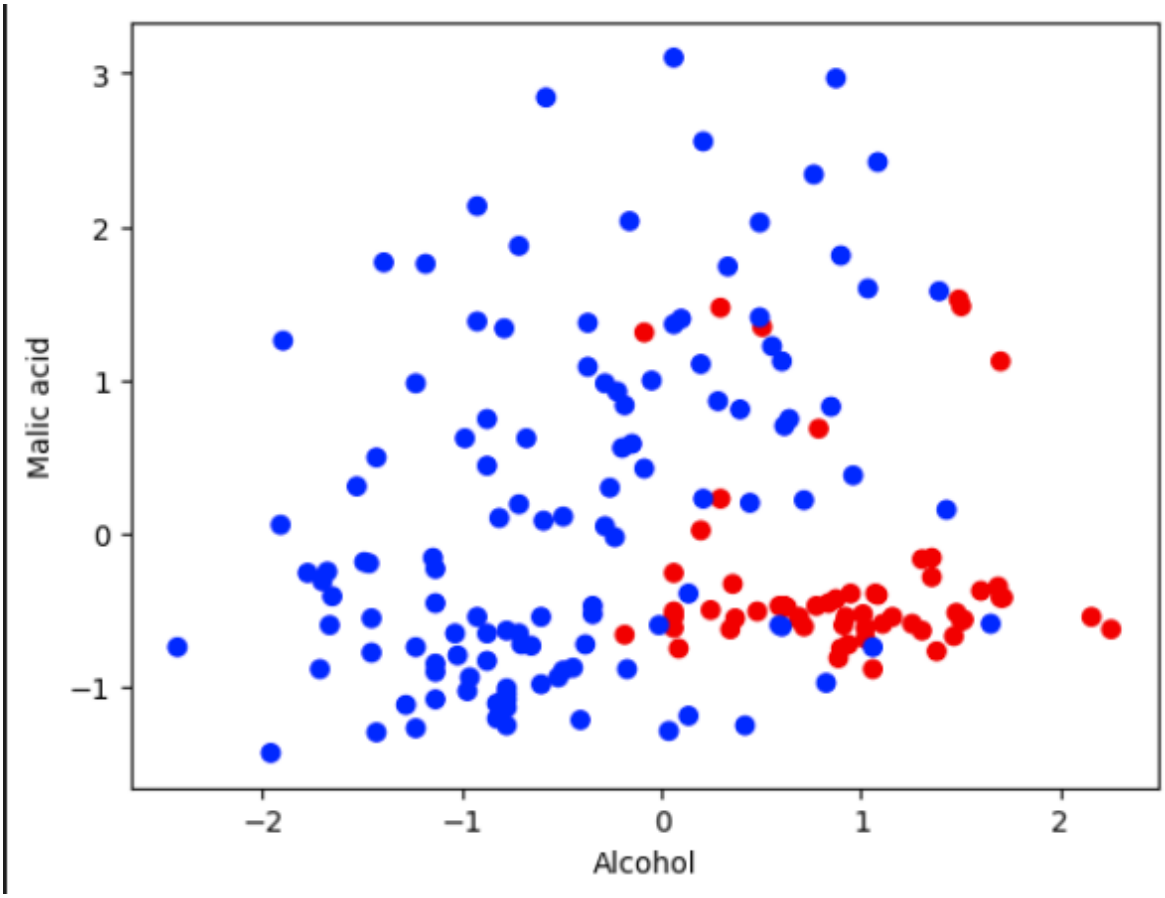
۱-۳. Adaline

ابتدا مجموعه داده Wine را می‌خوانیم و مربوط به ۱۷۸ شراب است که هر نمونه دارای ۱۳ ویژگی زیر میباشد و همچنین کلاس که مقادیر ۱ و ۲ و۳ را دارا می‌باشد نشان دهنده این است که شراب ما از کدام نوع می‌باشد و در واقع لیبل‌های ما برای این مسئله می‌باشند.

#	Column	Non-Null Count	Dtype
0	Class	178 non-null	int64
1	Alcohol	178 non-null	float64
2	Malic acid	178 non-null	float64
3	Ash	178 non-null	float64
4	Alcalinity of ash	178 non-null	float64
5	Magnesium	178 non-null	int64
6	Total phenols	178 non-null	float64
7	Flavanoids	178 non-null	float64
8	Nonflavanoid phenols	178 non-null	float64
9	Proanthocyanins	178 non-null	float64
10	Color intensity	178 non-null	float64
11	Hue	178 non-null	float64
12	OD280/OD315 of diluted wines	178 non-null	float64
13	Proline	178 non-null	int64

الف) در این قسمت لیبل‌های ما بر اساس این می‌باشد که شراب از نوع کلاس یک می‌باشد یا خیر به همین منظور قسمت class از مجموعه دادگان را جدا می‌کنیم و اگر مقدار یک داشت آن یک می‌گذاریم غیر این صورت منفی یک می‌گذاریم و به این شکل لیبل‌های این مسئله دوکلاسه ما را به دست می‌آوریم.

همانطور که در شکل بالا مشاهده کردیم در داده‌هامان مقادیر گم شده نداریم و همچنین همه داده‌ها به طور عددی هستند حال به عنوان پیش پردازی بر روی داده ها می‌خواهیم مقیاس ویژگی‌ها را به صورت توزیع نرمال تغییر دهیم و نرمال سازی کنیم پس برای هر ویژگی نمونه‌های آن را منهای میانگینشان و سپس تقسیم بر انحراف معیارشان می‌کنیم تا توبعشان نرمال شود.



حال نمودار پراکندگی داده‌ها بر اساس دو ویژگی Alchol و Milad Acid رسم می‌کنیم کلاس قرمز نشان دهنده شراب‌هایی است از نوع یک هستند و آبی نشان دهنده شراب‌هایی که از نوع یک نیستند. و مشاهده میشود که عموم شراب‌های نوع یک در گوشه سمت راست پایین هستند و شراب‌های نوع‌های دیگر هم در آنجا وجود دارند پس احتمالا از آنجایی که شبکه آدالاین به صورت خطی جدا می‌کند پس به دقت زیاد مطلوبی نخواهیم رسید.

حال به پیاده سازی شبکه آدالاین می‌پردازیم که و یک کلاس Adaline تعریف میکنیم که شامل بخش‌های زیر می‌باشد.

تابع __init__:

که وزن‌های اولیه شبکه را به صورت تصادفی و با مانگین صفر و انحراف معیار 0.01 مقداردهی می‌کنیم.(از اونجایی که فقط بر دو ویژگی Alchol و Milad Acid گفته شده است استفاده کنیم و همچنین دارای بایاس هم هستیم و شبکه آدالاین تک لایه هست پس ۳ تا وزن داریم.)

همچنین لیست cost را به وجود می‌اوریم که نگه دارنده cost ما برای هر ایپاک است.

تابع predict:

با گرفتن ورودی‌ها ، ورودی نورون خروجی و همچنین خروجی آن(پس از عبور از تابع فعال ساز) را محاسبه و هردو را باز می‌گرداند.

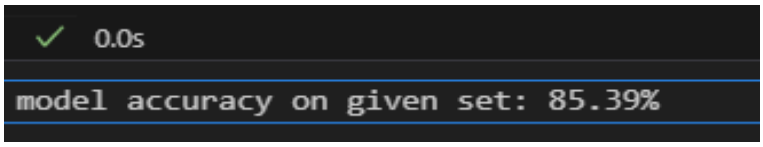
تابعupdate_weights_for_one_epoch:

که در آن ورودی‌های و خروجی‌های واقعی را می‌گیریم و براساس اینکه برای هر ورودی مدل چی پیش‌بینی کرده و تفاوت آن با لیبل وافعی برای هر ورودی و بر اساس learning_rate وزن‌ها را آپدیت می‌کنیم.

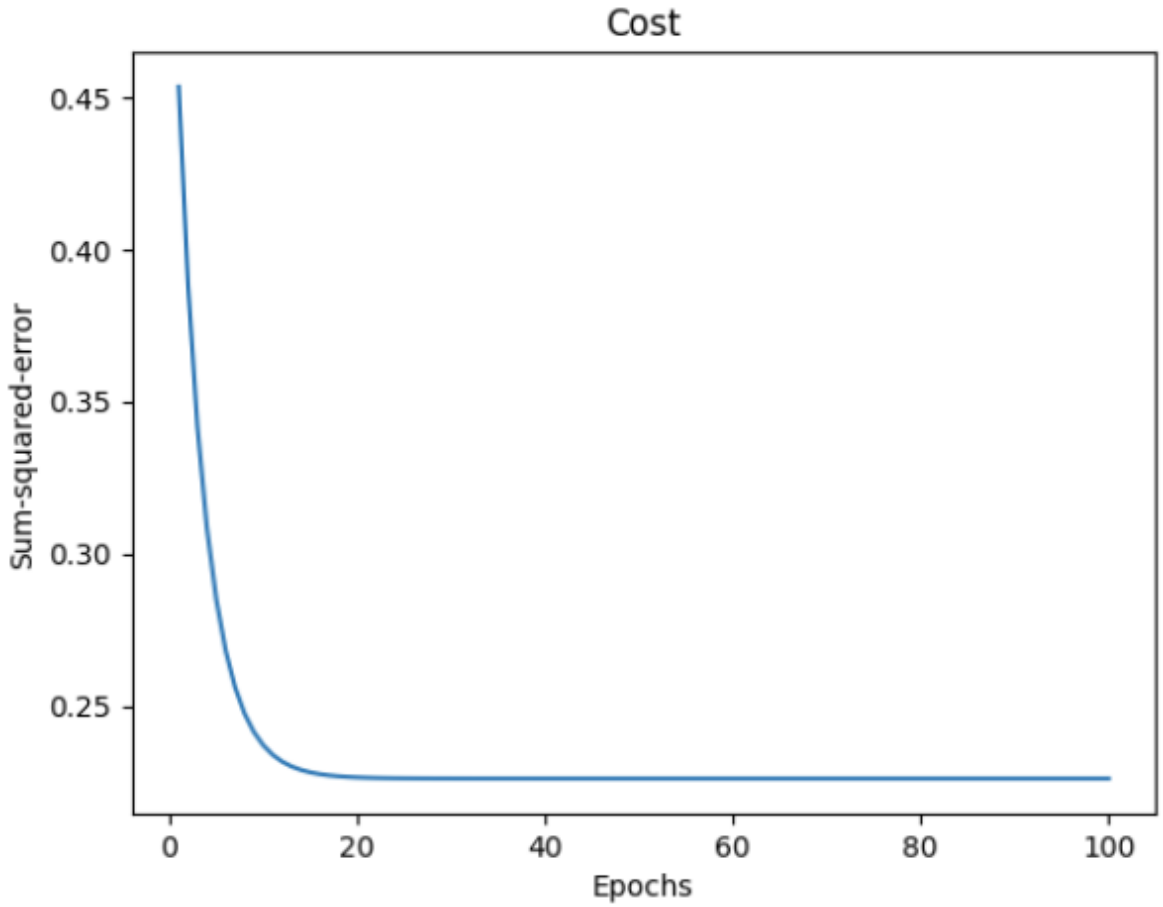
تابع fit:

به تعداد epoch های گفته شده وزن‌های شبکه را به روز رسانی می‌کنیم.

حال یک مدل از شبکه آدلاین را با تسفاده از دیتاهای خودمان و با learning_rate = 0.001 و تعداد ۱۰۰ ایپاک آموزش می‌دهیم و سپس آن را ارزیابی می‌کنیم.

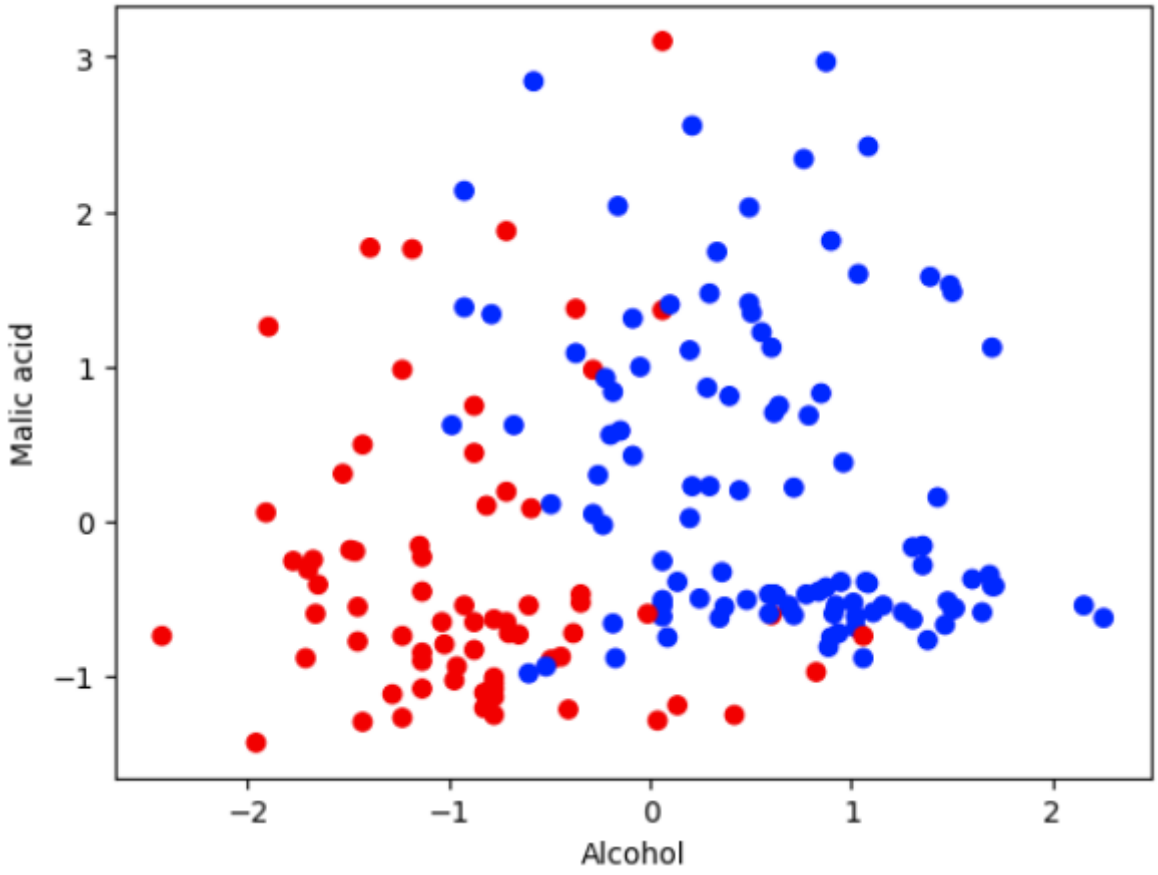


که دقت مدل به ۸۵درصد می‌رسد.



و نمودار تغییرات خطا رسم می‌کنیم و مشاهده می‌شود که از ایپاک ۱۸ تقریبا در 0.22 مانده است و تغیرات کمی داشته است.

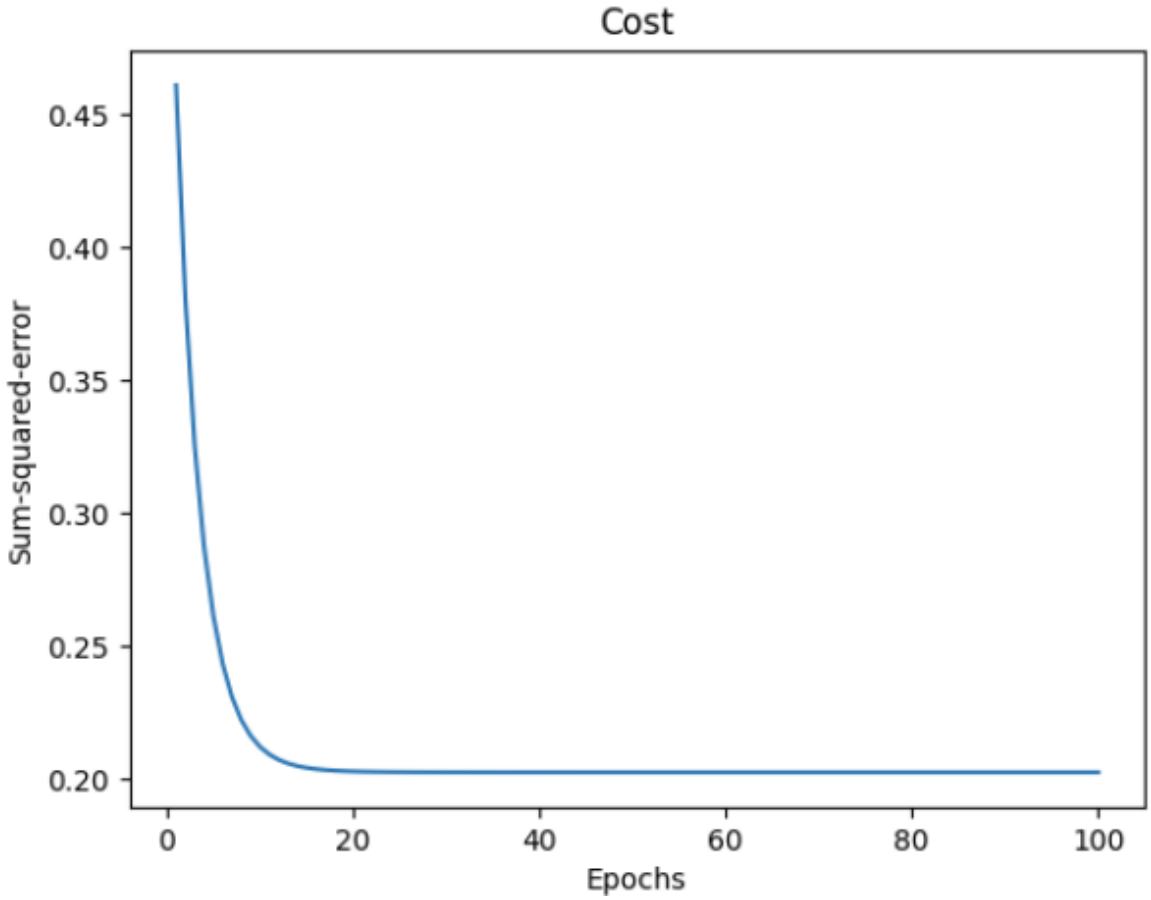
ب) این دفعه دو کلاس ما بر اساس این است که آیا شراب از نوع ۲ می‌باشد یا خیر و بر این اساس لیبل میکنیم و مانند قسمت الف داده‌ها را لیبل می‌زنیم



حال نمودار پراکندگی داده‌ها بر اساس دو ویژگی Alchol و Milad Acid رسم می‌کنیم کلاس قرمز نشان دهنده شراب‌هایی است از نوع دو هستند و آبی نشان دهنده شراب‌هایی که از نوع دو نیستند. مشاهده میکنیم که عموم قرمزها درسمت چپ و عموم آبی ها در سمت راست و نسبت به حالت قبل به صورت خطی جداپذیر تر هستند

```
model accuracy on given set: 88.76%
```

حال آدلاین را آموزش میدهیم و سپس ارزیابی می‌کنیم و میبینیم همانگونه که انتظار داشتیم(چرا که شبکه آدلاین خطی هست و یه خط جدا پذیر را پیدا میکند پس هرچقدر داده‌های ما به صورت خطی جدا پذیر تر باشند عملکرد ما بهتر می‌شود) و به 89 درصد رسیده است



مشاهده می‌شود که در نمودار تغییرات خطای ما این دفعه در نزدیکی ایپاک ۲۰ به ۰.۲۱ میرسد و تقریبا ثابت میماند که نسبت به حالت الف کمتر است و نشان میدهد که با این حالت مدل ما عملکرد بهتری را داشته است.

۲-۳. Madaline

الف) شبکه مادالاین شبکه‌ای دولایه است که در لایه پنهان نورونهای آدلاین دارد و به همین سبب قدرت جدا کنندگی بیشتر را نسبت به آدلاین دارا می‌باشد چرا که به واسطه چند خط می‌تواند فضای نمونه را جدا کند.

حال برای آموزش وزن‌های این شبکه دو الگوریتم MRI و MRII در کتاب تعریف شده است.

در الگوریتم MRI آموزش ما صرفا برروی وزن‌های لایه پنهان می‌باشد و وزن‌های لایه خروجی آموزشی ندارند و آن‌ها را طوری تعیین می‌کنیم که مانند یک OR منطقی باشند و در آن برای هر ورودی خروجی حاصل از آن برای شبکه را با لیبل آن مقایسه می‌کنیم و اگر یکی بودند کار نمیکنیم در غیر این صورت اگر لیبل واقعی منفی یک بود وزن نورونهای آدلاینی را آپدیت می‌کنیم که ورودی آن ها مثبت بوده است و اگر لیبل واقعی یک بود نورون‌های آدالاینی را ورودی آن‌ها بیشترین نزدیکی به صفر را دارا باشد.

در الگوریتم MRII مشابه الگوریتم MRI است با این تفاوت که تمام وزن‌ها را می‌خواهیم آموزش دهیم و همه وزن‌ها ممکن است آپدیت شوند به مانند الگوریتم قبل برای هر ورودی بررسی می‌کنیم که خروجی آن با لیبل واقعی

آن برابر است یا نه اگر بود که کاری نمی‌کنیم و اگر نبود بین آدلایین هایی که بیشترین نزدیکی به صفر را دارند انتخاب می‌کنیم و علامت خروجی آن را تغییر میدهیم و مجددا شبکه رو امتحان میکنیم اگر خطا کمتر شده بود خروجی جدید را به عنوان لیبل در نظر میگیریم و وزن‌ها را آپدیت میکنیم.

ب)داده‌های مصنوعی را همانطور که در صورت سوال گفته شده است ایجاد می‌کنیم و مشاهده می‌کنیم که لیبل‌های ما ۰ و ۱ هستند اما چون مادلاین داریم می‌خواهیم که ۱ و -۱ باشد برای همین لیبل های صفر را به منفی یک تبدیل می‌کنیم.

حال شبکه مادلاین با استفاده از الگوریتم آموزش MRI را پیاده سازی می‌کنیم.

یک کلاس Madaline تعریف میکنیم که شامل بخش‌های زیر می‌باشد.

تابع __init__:

که در آن تعداد نورون‌های لایه پنهان را ورودی می‌گیریم و وزن‌های اولیه لایه پنهان شبکه را به صورت تصادفی و با مانگین صفر و انحراف معیار 1 مقداردهی می‌کنیم.(از اونجایی که داده‌های ما فقط دارای دو ویژگی می باشند و همچنین دارای بایاس هم هستیم پس به تعداد نورونهای لایه در ۳ تا وزن داریم.)

و همانطور که در بخش الف گفتیم می‌خواهیم که لایه خورجی ما مانند OR کار کند پس وزن لایه خروجی را طوری می‌دهیم که اینکار را انجام دهد(وزن بایاس تعداد نورونها منهای یک و باقی وزن‌ها یک باشند که در اینطور اگر حتی خروجی یک از نورون‌های لایه پنهان یک باشند ورودی نورون خروجی نا منفی خواهد بود)

همچنین متغییر count را به وجود می‌اوریم که برای شرط توقف ما است و بررسی می‌کند که در چند دور متوالی وزن‌های ما تغییرات ناچیزی داشتند

تابع predict:

با گرفتن ورودی‌ها ،ورودی نورون های آدلاین را محاسبه می‌کند و سپس خروجی آن ها را حساب میکند و در انتها با استفاده از آنها ورودی نورون خروجی و همچنین خروجی آن را محسابه می‌کنیم

تابعupdate_weights_for_one_epoch:

که در آن ورودی‌های و خروجی‌های واقعی را می‌گیریم و براساس اینکه برای هر ورودی مدل چی پیش‌بینی کرده و آن را با لیبل واقعی مقایسه می‌کنیم اگر یکی بود که کار انجام نمی‌دهیم در غیر این صورت اگر لیبل واقعی ۱ بود مطابق الگوریتم گفته شده در الف ما به سراغ نزدیک ترین ورودی یک آدالاین به صفر باید برویم از آنجایی که لایه آخر ما مانندor است پس وقتی خروجی ما منفی یک شده یعنی ورودی تمان آدلاین‌ها منفی بوده پس بزرگترین ورودی‌های آدلاین را پیدا و نورون‌های آدلاینی که ورودی برابر بیشترین ورودی داشتند را آپدیت می‌کنیم در غیر این صورت اگر لیبل واقعی منفی یک بود نورون‌های آدالاینی که ورودیشون مثبت بود را آپدیت می‌کنیم .

در انتها بررسی می‌کنیم که وزن‌های ما نسبت به دفعه قبل چقدر تغییر کرده اند اگر خیلی ناچیز بود کانتر را یکی اضافه میکنیم و میبینیم که ایا به ترشولد ما رسیده است (اگر رسیده بود به الگوریتم پایان می‌دهیم.) و در غیر این صورت کانتر را صفر می‌کنیم

تابع fit:

به تعداد epoch های گفته شده وزن‌های شبکه را به روز رسانی می‌کنیم.

حال برای هرکدام از تعداد نورون‌های گفته شده برای لایه پنهان شبکه را با learning_rate = 0.001 و تعداد ۱۰۰۰ ایپاک آموزش می‌دهیم و سپس آن را ارزیابی می‌کنیم.

```
The model accuracy with 3 neurons: 86.67%
The model accuracy with 5 neurons: 90.0%
The model accuracy with 8 neurons: 95.0%
```

و همانطور که انتظار داشتیم با افزایش نورون‌های لایه پنهان پیچیدگی شبکه افزایش میباد و خط‌های جدا ساز بیشتری داریم(به تعداد نورون‌های لایه مخفی) پس دقت ما افزایش می‌یابد.

۴-۱. رگرشن

الف) بیش برازش (overfit) زمانی اتفاق می‌افتد که مدل ما روی داده‌های آموزشی بسیار خوب برازش شود و به نوعی آن‌ها را حفظ می‌کند و از تمام جزئیات آن‌ها برای تصمیم گیری استفاده می‌کند بنابراین اگر یک نمونه جدید(از داده‌های تست) که اندکی با نمونه‌های قبلی که مشاهده کرده است را به مدل بدهیم آنگاه از قابلیت تعمیم کمی برخوردار می‌باشد و پاسخ غیر دقیقی را تصمیم گیری می‌کند.

زمانی که بیش برازش داریم یعنی با تغییراتی اندک در ورودی مدل، خروجی ما دچار تغییرات زیادی شود و مدل ما حساسیت زیادی نسبت به تغییرات دارد و به این معنا می‌باشد که مدل ما دارای واریانس بالایی است و یکی از دلایل آن می‌تواند انعطاف پذیری بالای مدل باشد زیرا که هرچقدر که پارامترهای شبکه بیشتر باشند چیزهای بیشتری جهت تغییر در اختیار مدل قرار می‌گیرد

پس زمانی که مدل ما پیچیده است و داده‌های کمی داریم یا داده‌ها دارای نویز هستند و پیش پردازش نشدند یا ... دچار بیش برازش می‌شویم

ب) همانطور که در بخش قبل گفتیم یکی از دلایل بیش برازش می‌تواند انعطاف پذیری بالای مدل باشد زیرا که هرچقدر که پارامترهای شبکه بیشتر باشند چیزهای بیشتری جهت تغییر در اختیار مدل قرار می‌گیرد یکی از راه‌های کنترل کردن و محدود کردن این انعطاف پذیری به کمک کاهش وزن‌های شبکه هست چرا که می‌دانیم زمانی که یک مدل بیش برازش می‌کند یعنی به عده‌ای از ورودی‌ها بیش از حد اهمیت داده است و این به واسطه وزن‌های زیادی که مدل به آن ورودی‌ها داده است می‌باشد.

یکی از روش هایی که برای مقابله با بیش برازش وجود دارد dropout است که در آن در هر مرحله از آموزش شبکه به صورت تصادفی برخی از نورون‌های هر لایه را انتخاب می‌کنیم و آن‌ها را حذف می‌کنیم(به این معنا که در به روزرسانی وزن‌ها آنها را نادیده می‌گیریم) که در آن در هر مرحله آموزش هر نورون به احتمال p فعال نخواهد بود (p یک هایپرپارامتر است که توسط ما مشخص می‌شود.) پس در هرمرحله تعدادی از نورون‌ها غیرفعال خواهند بود و فقط عده‌ای از نورون‌ها در خروجی تاثیر خواهند داشت و این باعث می‌شود که هیچ نورونی تنبل نشود و با استفاده از این روش شبکه دیگر نمی‌تواند بر چند نورون خاص تکیه کند و با آن‌ها خروجی مشخص شود و وزن‌ها به طور عادلانه‌تری تقسیم می‌شوند که میتواند در جلوگیری از بیش برازش کمک کند.

در روش دیگر که 1L و 2L است در این روش‌ها به ترتیب مجموع قدر مطلق وزن‌ها و مجموع توان دوم وزن‌های شبکه را با یک وزنی(که هایپرپارامتر است) با تابع زیان جمع می‌کنیم (می‌خواهیم که به یک نسبتی با تابع زیان وزن‌ها هم کم شوند)و شبکه می‌خواهد آن را به حداقل مقدار خود برساند و مقدار مینیممش را بیابد.

روش دیگر Data augmentation است می‌دانیم که اگر مجموعه داده‌هامون رو بزرگ‌تر کنیم میتوانیم از بیش برازش جلوگیری کنیم پس در این روش به دنبال افزایش داده‌ها هستیم(مثلا در داده‌های عکسی با تغییر اندازه یا چرخش و... داده اضافه کنیم.)

روش دیگر Early stopping است که در آن در طول آموزش عملکرد مدل را روی داده‌های اعتبارسنجی می‌سنجیم و اگر در یک تعداد دور متوالی بهبود نیافت آنگاه آموزش را قطع و آخرین مدل قبل از عملکرد بد شود را به عنوان مدل نهایی خروجی می‌دهیم.

پ) پارامتر ویژگی‌های از شبکه است که برای آموزش آن الگوریتمی وجود دارد و با استفاده از آن‌ها یاد گرفته می‌شوند و مقادیرشان مشخص می‌شود اما ویژگی‌هایی از شبکه که باید خودمان آن‌ها را به طور دستی مشخص کنیم

را هایپرپارامتر می‌گوییم برای مثال تعداد لایه‌ها،تعدادی نورون‌های هر لایه،تابع فعال ساز هر لایه،بهینه‌ساز و نرخ یادگیری و ... همگی ویژگی‌هایی از ساختار شبکه هستند که توسط ما مشخص می‌شوند پس همانگونه که دیدیم تعداد هایپر پارامترها زیاد است و تنظیم و انتخاب مقدار مناسب برای آن‌ها کاری دشوار می‌باشد حال به سراغ های روش‌هایی میرویم که بتوانیم به کمک آن‌ها مقادیر بهینه‌تری برای هایپر پارامترها بیابیم

یکی از این روش‌ها Grid search است که در آن ترکیب‌های مختلف از هایپتر پارامترها را امتحان می‌کنیم (مثلا برای یکی از هایپرامترها ۷ حالت و دیگری ۶ حالت در نطر می‌گیریم و همه ۴۲ حالت را امتحان می‌کنیم) و با بررسی عملکرد آن‌ها روی داده‌های اعتبارسنجی بهترین آن‌ها را انتخاب می‌کنیم.

یکی دیگر از روش‌ها random search است که جای روش بالا که همه حالت ها را حساب کنیم به صورت تصادفی چند حالت را بررسی می‌کنیم که در این حالت وقت کمتری از ما می‌گیرد و احتمال دارد به بهترین جواب هم برسیم.

روش دیگر Zooming In است که مشابه حالت قبل است اما در جست‌وجوی تصادفی هرجا که دیدیم عملکرد بهتر شده جست‌وجوی بعدی را در اطراف آن حالت انجام می‌دهیم و به نوعی زوم می‌شویم

روش دیگر Evolutionary search که در آن سعی می‌شود به کمک الگوریتم‌های تکاملی و ژنتیک الگوریتم جواب‌های بهیته برای هایپتر پارامترها پیدا شوند

ت)

جدول . هایپرپارامترهای مربوط به مسئله افزایش اندازه داده های تست

1	تعداد لایه مخفی
100	تعداد نورون در هر لایه مخفی
selu	تابع فعالساز
100	تعداد epoch
10	اندازه هر batch

جدول . هایپرپارامترهای مربوط به مسئله افزایش تعداد لایه ها

بین 0 تا 20	تعداد لایه مخفی
5	تعداد نورون در هر لایه مخفی
tanh	تابع فعالساز لایه مخفی
70	تعداد epoch
16	اندازه هر batch

همانطور که مشاهده می شود با افزایش تعداد داده های test، دقت مدل در تعمیم داده ها کاهش می یابد و به اصطلاح مدل **overfit**می کند.

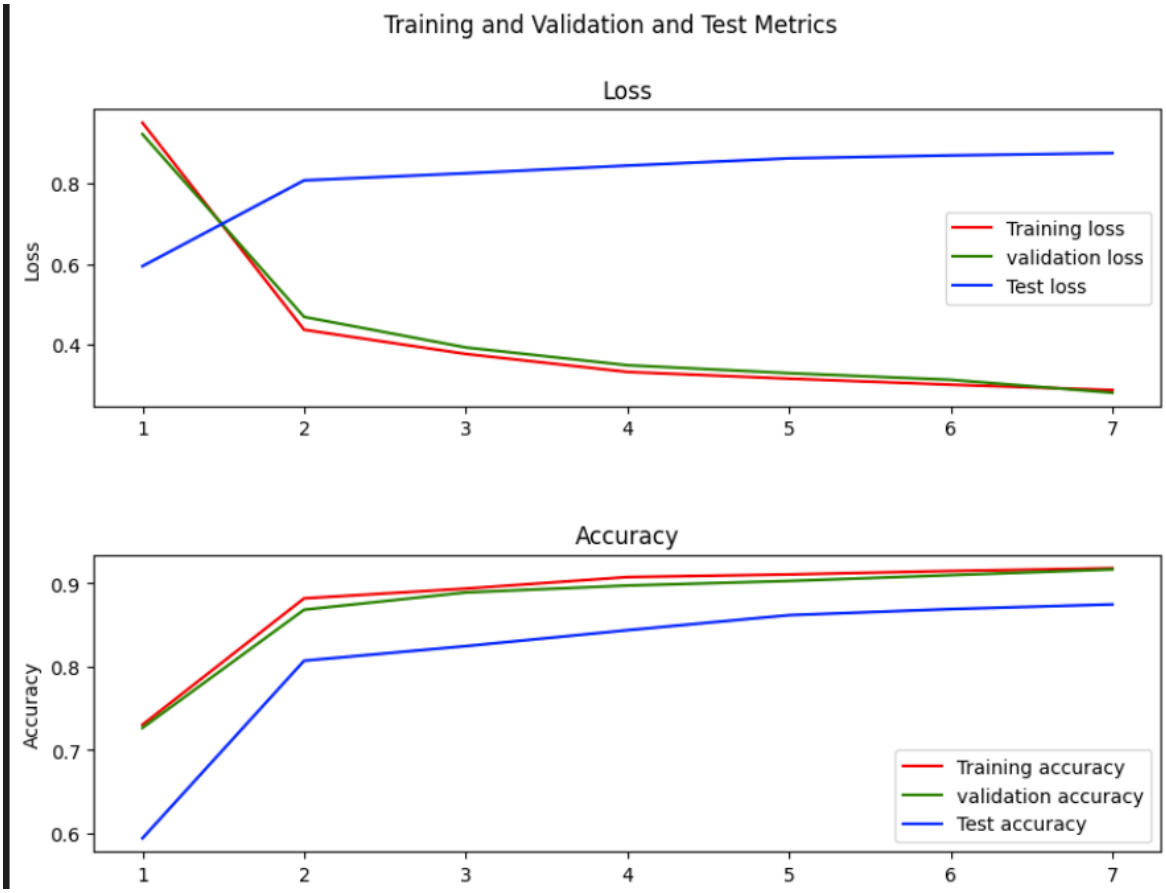
همچنین با افزایش تعداد لایه ها، تعداد پارامترهای مجهول افزایش می یابد و مدل **overfit**می کند. (در راه حل ارائه شده به علت ثابت ماندن تعداد epoch ها به ازای هر تعداد لایه، دقت داده های train شده هم پایین است).

ث) با استفاده از GridSearchValidation بهترین تعداد لایه های مخفی، 13 میباشد. استفاده از GridSearchValidation باعث یافتن بهترین تعداد لایه میشود اما اگر تعداد پارامتر ها و همچنین بازه منطقی این پارامترها زیاد باشد، استفاده از آن طولانی است و به صرفه نمی باشد.

2-4. طبقه بندی

جدول . هایپرپارامترهای مربوط به مسئله افزایش اندازه داده های تست

2	تعداد لایه مخفی
64 ,256	تعداد نورون در هر لایه مخفی
relu	تابع فعالساز لایه مخفی
25	تعداد epoch
10	اندازه هر batch
softmax	تابع فعالساز لایه خروجی



جدول . هایپرپارامترهای مربوط به مسئله افزایش تعداد لایه ها

بین 0 تا 20	تعداد لایه مخفی
20	تعداد نورون در هر لایه مخفی
selu	تابع فعالساز لایه مخفی
25	تعداد epoch
32	اندازه هر batch
softmax	تابع فعالساز لایه خروجی

همانند بخش رگرسیون، با افزایش تعداد لایه ها و افزایش نسبت داده های تست، مدل overfit میکند و قدرت تعمیم آن کم می شود.


Training and Validation and Test Metrics
Loss
Loss
1.0
0.8
0.6
0.4
0.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training loss
validation loss
Test loss
Accuracy
Accuracy
0.98
0.97
0.96
0.95
0.94
0.93
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Training accuracy
validation accuracy
Test accuracy

.