



## درس شبکه‌های عصبی و یادگیری عمیق تمرین اول

نام و نام خانوادگی	نام و نام خانوادگی	پرسش ۱ و ۴
احسان فروتن	شماره دانشجویی	
پویا حاجی محمدی گوهری	نام و نام خانوادگی	پرسش ۲ و ۳
۸۱۰۱۰۲۱۱۳	شماره دانشجویی	
۱۴۰۳۰۸۰۱۵	مهلت ارسال پاسخ	

## فهرست

۳	..... شکل ها
۵	..... جداول
۱	..... قوانین
۱	..... پرسش ۱. تحلیل و طراحی شبکه های عصبی چند لایه (MLP)
۵	..... اثر افزایش پیچیدگی
۶	..... بررسی تغییر تک تک Config ها
۸	..... متريک های انتخاب پيکربندی
۹	..... ۲-۱ آموزش ۲ مدل مختلف
۹	..... هيستوگرام وزن ها
۱۰	..... اثر بهينه ساز
۱۲	..... ۳-۱. الگوريتم بازگشت به عقب
۱۲	..... الگوريتم ها و تاثير آنها
۱۴	..... جستوجوی بيزي
۱۶	..... ۴. بررسی هايپر پارامتر های مختلف
۱۷	..... تغيير نرخ يادگيری
۱۸	..... تغيير تعداد نوروذ
۲۰	..... عوض کردن تعداد لایه
۲۱	..... استفاده از جستوجوی تصادفي
۲۳	..... پرسش ۲ - آموزش و ارزیابی یک شبکه عصبی ساده
۲۳	..... آموزش یک شبکه عصبی
۲۴	..... ۲-۱-۱.تابع forward
۲۴	..... ۲-۱-۲.تابع Backward
۲۵	..... ۲-۲-۱. آزمون شبکه عصبی بر روی یک مجموعه داده
۲۶	.....
۲۶	..... نمودار خطأ و تحليل آن
۲۶	..... root mean square error
۲۶	..... پيش‌بياني داده‌های تست و
۲۹	..... ۳-۱-۱. الگوريتم های MRI و MRII
۲۹	..... ۳-۱-۲. الگوريتم MRII
۳۱	..... ۳-۱-۳. الگوريتم MRII
۳۱	..... ۳-۲. نمودار پراکندگی داده‌ها
۳۱	..... ۳-۳. آموزش مدل
۳۲	..... ۳-۳-۱. مدل با سه نوروذ

۳۳	۳-۲. مدل با چهار نوروز.....
۳۳	۳-۳. مدل با هشت نوروز.....
۳۴	۳-۴. تحلیل نتایج.....
۳۵	پرسش ۴.....MLP
۳۵	۱-۴. نمایش تعداد ستون.....
۳۵	۴-۲. ماتریس همبستگی.....
۳۷	۴-۳. رسم نمودار.....
۳۸	۴-۴. پیش پردازش داده.....
۴۱	۴-۵. پیاده سازی مدل.....
۴۲	۴-۶. آموزش مدل.....
۴۳	مدل تک لایه.....
۵۰	مدل دو لایه.....
۵۹	۴-۷. تحلیل نتایج.....
۵۹	انتخاب ۱۰ داده تصادفی.....
۵۹	تحلیل نتایج.....

## شکل ها

۱	شکل ها ۱: خلاصه مدل مورد استفاده و تعداد پارامتر های آذبه ازای ورودی
۱	شکل ها ۲ شکل مدل و لایه مخفی
۲	شکل ها ۳: یک پیرهن در داده های آموزش FASHION MNIST
۲	شکل ها ۴: LOSS و ACCURACY برای مدل در حالت رگیولار کردن فقط لایه مخفی
۲	شکل ها ۵: LOSS و ACCURACY برای مدل در حالت رگیولار کردن تمام وزن ها
۳	شکل ها ۶: ماتریس آشفتگی داده های آموزش و تست در حال REGULAR کردن کل شبکه
۳	شکل ها ۷: ماتریس آشفتگی داده های تست در حالت رگولار کردن لایه مخفی
۴	شکل ها ۸: درصد بندی کلاس ها
۴	شکل ها ۹: Classification Report
۵	شکل ها ۱۰: آموزش و تست با setting های مختلف
۶	شکل ها ۱۱: اطلاعات خلاصه برای طبقه بندی در حالت ۲۰۰ نورون و یک لایه مخفی
۶	شکل ها ۱۲: شبکه با دو لایه
۶	شکل ها ۱۳: آموزش شبکه شکل ۱۲
۷	شکل ها ۱۴: بهترین شبکه با یک لایه
۷	شکل ها ۱۵: نتایج آموزش شبکه ۱۴
۷	شکل ها ۱۶: ماتریس آشفتگی شبکه شکل ۱۲ (دو لایه)
۸	شکل ها ۱۷: ماتریس آشفتگی شبکه شکل ۱۴
۹	شکل ها ۱۸: ۱۲۸ نورون در لایه مخفی، بدون regularizer و dropout
۹	شکل ها ۱۹: ۴۸ نورون در لایه مخفی، regularizer برابر با ۰.۲ و داشتن dropout rate
۹	شکل ها ۲۰: آموزش ۲ مدل بر روی یک دیتا و مقایسه آموزش و تست
۱۰	شکل ها ۲۱: هیستوگرام ورز مدل شکل ۱۹
۱۰	شکل ها ۲۲: هیستوگرام وزن شکل ۲۰: سمت راست بین ورودی و hidden, سمت چپ بین hidden و خروجی.
۱۱	شکل ها ۲۳: بهینه ساز SGD
۱۱	شکل ها ۲۴: بهینه ساز ADAM
۱۱	شکل ها ۲۵: بهینه ساز RMSprop
۱۲	شکل ها ۲۶: تاثیر منفی Overfitting بر RMSprop (خطوط آبی برای تست هستند)
۱۲	شکل ها ۲۷: شبکه استفاده شده در این بخش
۱۴	شکل ها ۲۸: مقایسه روش های مختلف
۱۶	شکل ها ۲۹: یادگیری بیزی بر روی مدل ۳ لایه
۱۶	شکل ها ۳۰: جواب Grid search
۱۶	شکل ها ۳۱: یک معماری مناسب
۱۷	شکل ها ۳۲: اثر ضربی یادگیری
۱۷	شکل ها ۳۳: اثر LR=10e-۲ بر روی Confusion matrix
۱۸	شکل ها ۳۴: اثر LR=10e-۳ بر روی Confusion matrix
۱۸	شکل ها ۳۵: اثر LR=10e-۴ بر روی Confusion matrix
۱۹	شکل ها ۳۶: اثر تغییر تعداد نورون ها
۱۹	شکل ها ۳۷: تعداد نورون ۱۲۸
۲۰	شکل ها ۳۸: تعداد نورون ۲۵۶
۲۰	شکل ها ۳۹: تعداد نورون ۴۰۰
۲۰	شکل ها ۴۰: تاثیر تعداد لایه
۲۲	شکل ها ۴۱: ماتریس آشفتگی حاصل از بهترین حالت Random Search
۲۳	شکل ها ۴۲: شکل ۱. مبنای یک شبکه عصبی با یک لایه
۲۴	شکل ها ۴۳: شکل ۲.تابع فوروارد

۲۴	..... شکل ۳. تابع کمکی که وزنهای جدید را بازمیگرداند.....
۲۵	..... شکل ۴. تابع backward
۲۶	..... شکل ۵. نمودار خطای بر حسب تکرار.....
۲۷	..... شکل ۶. نمودار خطای برای هر نرخ یادگیری.....
۲۹	..... شکل ۷. نمونه‌ای از کتاب مرجع برای شبکه Madline
۳۱	..... شکل ۸. نمودار پراکنده‌ی داده‌ها
۳۲	..... شکل ۹. نمودار پراکنده‌ی داده‌ها که توسط ۳ خط جدا شده‌اند.....
۳۳	..... شکل ۱۰. نمودار پراکنده‌ی داده‌ها به همراه چهار خط جداساز با مدل چهار نورونی.....
۳۳	..... شکل ۱۱. نمودار پراکنده‌ی داده‌ها به همراه هشت خط جداکننده با مدل هشت نورونی.....
۳۵	..... شکل ۱۲. ستون‌ها و تعداد nan ها در داده
۳۶	..... شکل ۱۳. ماتریس همبستگی داده ها
۳۷	..... شکل ۱۴. همبستگی ها با price
۳۷	..... شکل ۱۵. نمودار توزیع قیمت
۳۸	..... شکل ۱۶. ارتباط بالاترین correlation با prcie
۳۹	..... شکل ۱۷. تبدیل تاریخ به قیمت و سال
۴۰	..... شکل ۱۸. ماتریس جدید Correlation
۴۰	..... شکل ۱۹. داده های آموزش
۴۰	..... شکل ۲۰. داده های تست
۴۱	..... شکل ۲۱. میانگین آموزش بعد از scale شدن
۴۱	..... شکل ۲۲. شبکه تک لایه
۴۲	..... شکل ۲۳. شبکه دو لایه
۴۲	..... شکل ۲۴. کافیگ و تعداد پارامتر شبکه تک لایه
۴۳	..... شکل ۲۵. کافیگ شبکه ۲ لایه
۴۴	..... شکل ۲۶. آموزش و تست برای شبکه تک لایه با MAE
۴۴	..... شکل ۲۷. متریک R <sub>2</sub> برای تک لایه با MAE
۴۵	..... شکل ۲۸. آموزش شبکه تک لایه با MSE
۴۶	..... شکل ۲۹. آموزش شبکه با MSE، متریک R <sub>2</sub>
۴۷	..... شکل ۳۰. Standard Scaler با استفاده از MAE
۴۸	..... شکل ۳۱. Standard Scaler برای آموزش تک لایه MAE با استفاده از Standard Scaler
۴۹	..... شکل ۳۲. آموزش تک لایه MSE با استفاده از Standard Scaler
۵۰	..... شکل ۳۳. آموزش تک لایه MSE با استفاده از Standard Scaler
۵۱	..... شکل ۳۴. آموزش دو لایه MAE با استفاده از MinMax Scalar
۵۲	..... شکل ۳۵. آموزش دو لایه MAE با استفاده از MinMax Scalar
۵۳	..... شکل ۳۶. آموزش دو لایه MSE با استفاده از MinMax Scalar
۵۴	..... شکل ۳۷. آموزش دو لایه MSE با استفاده از MinMax Scalar
۵۵	..... شکل ۳۸. آموزش دو لایه MAE با استفاده از Standard Scalar
۵۶	..... شکل ۳۹. آموزش دو لایه MAE با استفاده از Standard Scalar
۵۷	..... شکل ۴۰. آموزش دو لایه MSE با استفاده از Standard Scalar
۵۸	..... شکل ۴۱. آموزش دو لایه MAE با استفاده از Standard Scalar
۵۹	..... شکل ۴۲. تفاوت سمپل های پیش بینی شده و واقعی تست در دیتای اسکیل شده
۶۰	..... شکل ۴۳. فرمول R <sub>2</sub> -Score

## جدوا

جدوا ۴ کلاس های بیشتر از همه اشتباه گرفته شده در تست(بر اساس شکل ۷).....	۴
جدوا ۵.هایپرپارامترهای مدل.....	۲۵
جدوا ۶. معیار RMSE برای نرخهای یادگیری مختلف.....	۲۷
جدوا ۷.۱. مقایسه شبکه تک لایه با MINMAX اسکیلر و دو Loss function مختلف.....	۴۶
جدوا ۸-۲. مقایسه شبکه تک لایه با Standard Scaler و دو Loss function مختلف.....	۵۰
جدوا ۹.۳. مقایسه شبکه دو لایه با MINMAX اسکیلر و دو Loss function مختلف.....	۵۴
جدوا ۱۰.۴. مقایسه شبکه دو لایه با Standard Scaler اسکیلر و دو Loss function مختلف.....	۵۸
جدوا ۱۱.۴.۵. بهترین حالت شبکه ۲ لایه و تک لایه با همدیگر.....	۵۹

قبل از پاسخ دادن به پرسش‌ها، موارد زیر را با دقت مطالعه نمایید:

- از پاسخ‌های خود یک گزارش در قالبی که در صفحه‌ی درس در سامانه‌ی Elearn با نام **REPORTS TEMPLATE.docx** قرار داده شده تهیه نمایید.

- پیشنهاد می‌شود تمرين‌ها را در قالب گروه‌های دو نفره انجام دهید. (بیش از دو نفر مجاز نیست و تحويل تک نفره نیز نمره‌ی اضافی ندارد) توجه نمایید الزامی در یکسان‌ماندن اعضای گروه تا انتهای ترم وجود ندارد. (یعنی، می‌توانید تمرين اول را با شخص A و تمرين دوم را با شخص B و ... انجام دهید)
- کیفیت گزارش شما در فرآیند تصحیح از اهمیت ویژه‌ای برخوردار است؛ بنابراین، لطفاً تمامی نکات و فرضهایی را که در پیاده‌سازیها و محاسبات خود در نظر می‌گیرید در گزارش ذکر کنید.
- در گزارش خود مطابق با آنچه در قالب نمونه قرار داده شده، برای شکل‌ها زیرنویس و برای جدول‌ها بالانویس در نظر بگیرید.
- الزامی به ارائه توضیح جزئیات کد در گزارش نیست، اما باید نتایج بدست آمده از آن را گزارش و تحلیل کنید.

- تحلیل نتایج الزامی می‌باشد، حتی اگر در صورت پرسش اشاره‌ای به آذنشده باشد.
- دستیاران آموزشی ملزم به اجرا کردن کدهای شما نیستند؛ بنابراین، هرگونه نتیجه و یا تحلیلی که در صورت پرسش از شما خواسته شده را به طور واضح و کامل در گزارش بیاورید. در صورت عدم رعایت این مورد، بدیهی است که از نمره تمرين کسر می‌شود.

- کدها حتماً باید در قالب نوت‌بوک با پسوند **ipynb**. تهیه شوند، در پایان کار، تمامی کد اجرا شود و خروجی هر سلول‌ها در این فایل ارسالی شما ذخیره شده باشد. بنابراین برای مثالاً اگر خروجی سلوی یک نمودار است که در گزارش آورده‌اید، این نمودار باید هم در نوت‌بوک کدها وجود داشته باشد.

- در صورت مشاهده‌ی تقلب امتیاز تمامی افراد شرکتکننده در آن ۱۰۰ - لحظه می‌شود.
- تنها زبان برنامه نویسی مجاز **Python** است.

- استفاده از کدهای آماده برای تمرينها به هیچ وجه مجاز نیست. در صورتی که دو گروه از یک منبع مشترک استفاده کنند و کدهای مشابه تحويل دهند، تقلب محسوب می‌شود.

- نحوه محاسبه تاخیر به این شکل است: پس از پایان رسیدن مهلت ارسال گزارش، حداکثر تا یک هفته امکان ارسال با تاخیر وجود دارد، پس از این یک هفته نمره آن تکلیف برای شما صفر خواهد شد.

- سه روز اول بدون جریمه
- روز چهارم: ۵ درصد
- روز پنجم: ۱۰ درصد
- روز ششم: ۱۵ درصد
- روز هفتم: ۲۰ درصد

- حداکثر نمره‌ای که برای هر سوال می‌توان اخذ کرد ۱۰۰ بوده و اگر مجموع بارم یک سوال بیشتر از ۱۰۰ باشد، در صورت اخذ نمره بیشتر از ۱۰۰، اعمال نخواهد شد.

- برای مثال اگر نمره اخذ شده از سوال ۱ برابر ۱۰۵ و نمره سوال ۲ برابر ۹۵ باشد، نمره نهایی تمرين ۹۷.۵ خواهد بود و نه ۱۰۰.

- لطفاً گزارش، کدها و سایر ضمایم را به در یک پوشه با نام زیر قرار داده و آن را فشرده سازید، سپس در سامانه‌ی Elearn بارگذاری نمایید:

HW[Number]\_[Lastname]\_[StudentNumber]\_[Lastname]\_[StudentNumber].zip

HW1\_Ahmadi\_۸۱۰۱۹۹۱۰۱\_Bagheri\_۸۱۰۱۹۹۱۰۲.zip (مثال)

- برای گروههای دو نفره، بارگذاری تمرین از جانب یکی از اعضا کافی است ولی پیشنهاد می‌شود هر دو نفر بارگذاری نمایند.

## پرسش ۱. تحلیل و طراحی شبکه های عصبی چند لایه (MLP)

(این سوال برای مقایسه بهتر نتایج توسط هر دو نفر کد زده شده و نتایج تجمعی حالت های مختلف برای کامل تر بودن تحلیل ها آورده شده است.)

### ۱-۱. طراحی MLP

یک شبکه عصبی به شکل زیر طراحی می شود و دارای یک لایه مخفی با ۱۰۰ نод به همراه فعال سازی RELU و Dropout با نرخ ۳۰ درصد و L2-Regularization برابر با ۰.۰۰۰۱ است.

Layer (type:depth-idx)	Output Shape	Param #
—Linear: 1-1	[ -1, 100]	78,500
—Dropout: 1-2	[ -1, 100]	--
—Linear: 1-3	[ -1, 10]	1,010
—Flatten: 1-4	[ -1, 10]	--
<hr/>		
Total params: 79,510		
Trainable params: 79,510		
Non-trainable params: 0		
Total mult-adds (M): 0.08		
<hr/>		
Input size (MB): 0.10		
Forward/backward pass size (MB): 0.00		
Params size (MB): 0.30		
Estimated Total Size (MB): 0.40		
<hr/>		

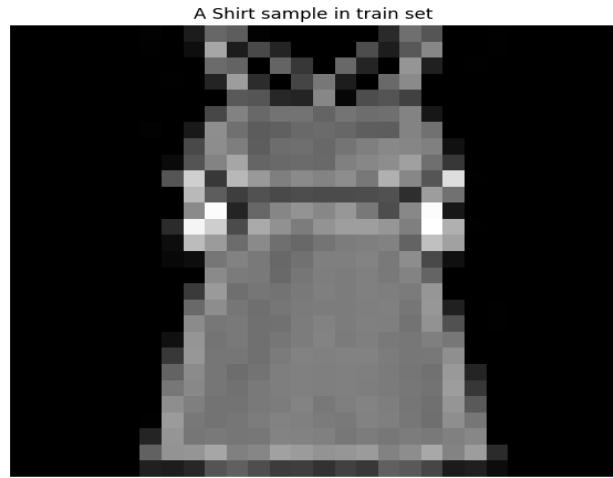
شکل ها ۱: خلاصه مدل مورد استفاده و تعداد پارامتر های آذ به ازای ورودی

برای آموزش این شبکه، داده های MNIST FASHION استفاده گردیده است. ورودی شبکه عکس های سیاه و سفید با بعد ۲۸ در ۲۸ بوده و در خروجی، باید بین ۱۰ کلاس طبقه بندی انجام گردد. به دلیل استفاده از MLP و اینکه تمامی نود های یک لایه به تمامی نود های لایه بعد، متصل می گردد، می تواند ورودی را از شکل یک ماتریس به یک بردار یک بعدی با ۷۸۴ عضو در آورد. همچنین، برای خروجی به جای طبقه بندی و خروجی داده بین ۱ و ۱۰، می تواند label های را One-Hot انکود کرده و خروجی شبکه را به صورت یک بردار ۱ در ۱۰ در نظر گرفت. این بردار در کنار Loss برابر با Cross Entropy به شکل پیدا کرد: محتمل ترین کلاس در خروجی در آمده (Maximum Log Likelihood) و برای رسیدن به جواب کلاس، صرفا ARGMAX می تواند گرفت. سایز ورودی و خروجی به صورت زیر آمده است:

```
... MLP_one_hidden_layer(  
    (hidden_layer): Linear(in_features=784, out_features=100, bias=True)  
    (dropout): Dropout(p=0.3, inplace=False)  
    (relu): ReLU()  
    (clf): Linear(in_features=100, out_features=10, bias=True)  
)
```

شکل ها ۲ شکل مدل و لایه مخفی

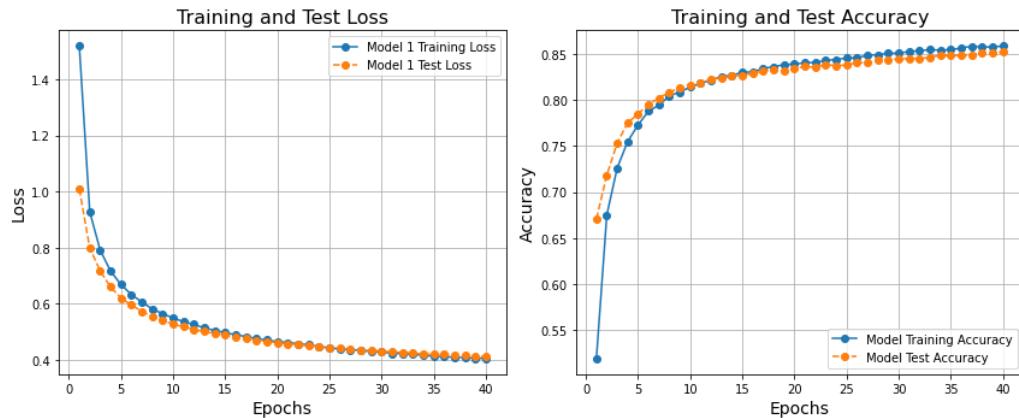
برای رفرانس، شکل یکی از داده های آموزش قبل از Flat شدن به صورت مثال آمده است:



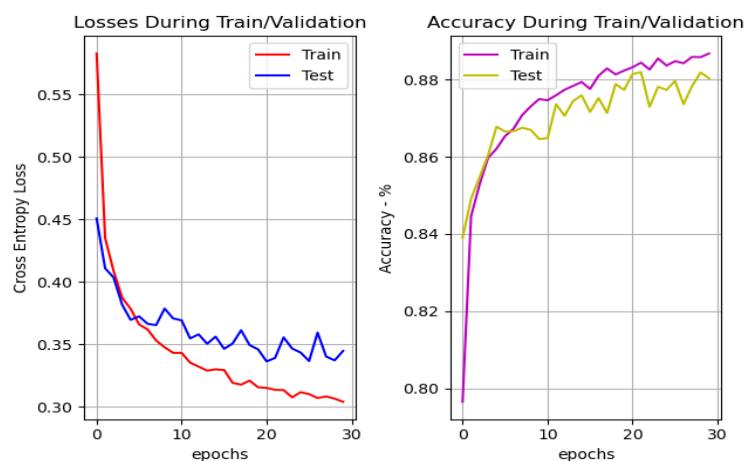
شکل ها ۳: یک پیرهنه در داده های آموزش FASHION MNIST

آموزش مدل به صورت ITERATIVE بوده و در هر گام پس از محاسبه Loss حاصل از گذاشتن اختلاف لیبل واقعی و پیش بینی شده بدست می آید.

در رابطه با Regularization، می توان آنرا تنها روی لایه مخفی اعمال کرده و یا به عنوان weight-decay برای کل وزنهای شبکه در نظر گرفت. در حالت اول آموزش و تست به صورت زیر است:



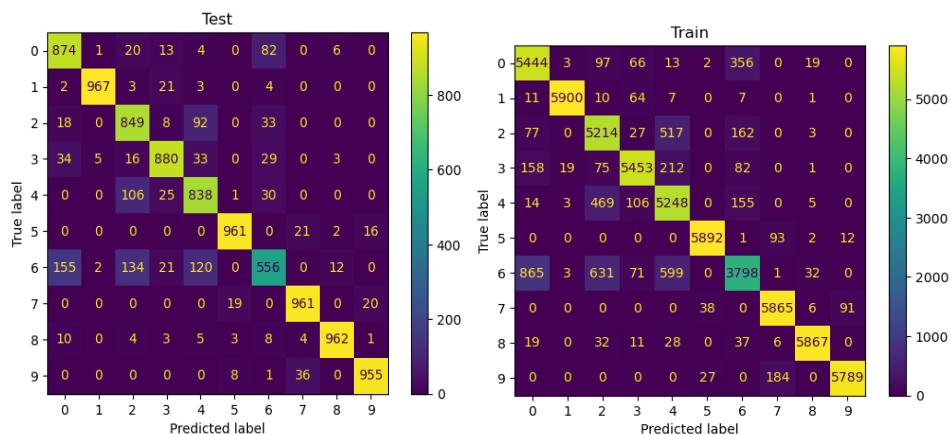
شکل ها ۴: ACCURACY و LOSS برای مدل در حالت رگیولار کردز فقط لایه مخفی



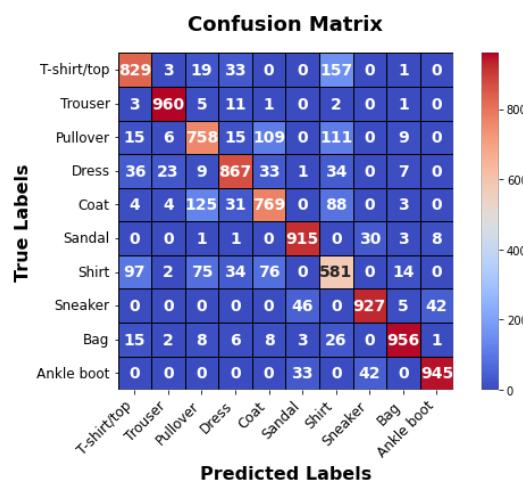
شکل ها ۵: ACCURACY و LOSS برای مدل در حالت رگیولار کردز تمام وزنهای

همانطور که مشاهده می شود، هر دو مدل همگرایی خوبی نشان داده و به دقت بالای ۸۵ درصد در حالت تست می رستند. البته مدل اول با استفاده از رگولارایز فقط در لایه مخفی بر روی تست و آموزش از پایداری بهتری برخوردار بوده، به دلیل محدود نکردن اجازه بزرگ شدن وزن ها در قسمت خروجی. همچنین مدل اول توسط روش **SGD** آموزش داده شده است که دیرتر به همگرایی رسیده، در حالی که روش دوم و با استفاده از **ADAM** همگرایی سریع تری داشته است.

به طور کلی، ماتریس های آشفتگی به صورت زیردر میابند:

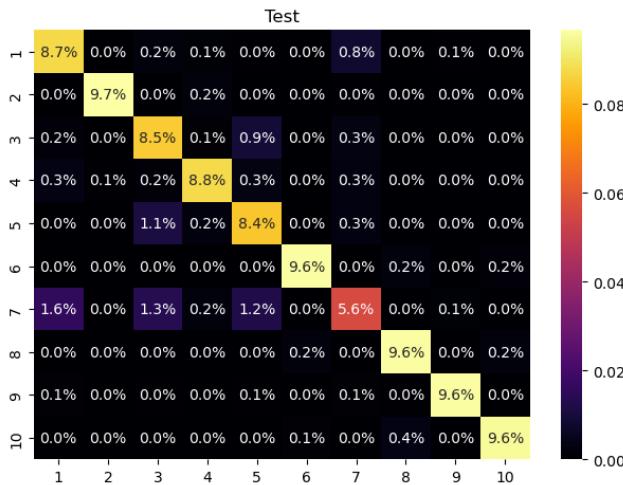


شکل ها ۶: ماتریس آشفتگی داده های آموزش و تست در حالت **REGULAR** کردز کل شبکه



شکل ها ۷: ماتریس آشفتگی داده های تست در حالت رگولار کردز لایه مخفی

ماتریس های آشفتگی تست در هر دو حالت نشان می دهند که اکثر کلاس ها با دقت بالاتر از ۹۰ درصد دسته بندی شده اند و تنها کلاس **Shirt** بیشتر اشتباه گرفته می شده است. شکل بعد این نحوه طبقه بندی را به صورت درصد از کل داده های تست (۱۰ درصد برای هر کلاس) آورده است:



شکل ها: درصد بندی کلاس ها

همانطور که از شکل ۸ دیده می شود، کلاس shirt در ۵۶ درصد موقع از بقیه درست تشخیص داده شده و در ۸ درصد موقع T-shirt تشخیص داده شده است. البته این موضوع به دلیل شباهت این دو کلاس قابل پیش بینی بوده است.

Test Report:		precision	recall	f1-score	support
0	0.80	0.87	0.84	1000	
1	0.99	0.97	0.98	1000	
2	0.75	0.85	0.80	1000	
3	0.91	0.88	0.89	1000	
4	0.77	0.84	0.80	1000	
5	0.97	0.96	0.96	1000	
6	0.75	0.56	0.64	1000	
7	0.94	0.96	0.95	1000	
8	0.98	0.96	0.97	1000	
9	0.96	0.95	0.96	1000	
accuracy				0.88	10000
macro avg		0.88	0.88	0.88	10000
weighted avg		0.88	0.88	0.88	10000

شکل ها: Classification Report

جزییات بیشتر کلاس بندی و معیار های مختلف برای قدرت کلاس بندی در شکل ۹ آورده شده است. به طور کلی نیز دقیق مدل بر روی داده تست به ۸۸ درصد می رسد.

همچنین طبق شکل ۶، کلاس هایی که بیشتر از همه با هم اشتباه گرفته می شوند، در جدول بعدی آمده است:

جدول ۱ کلاس های بیشتر از همه اشتباه گرفته شده در تست (بر اساس شکل ۷)

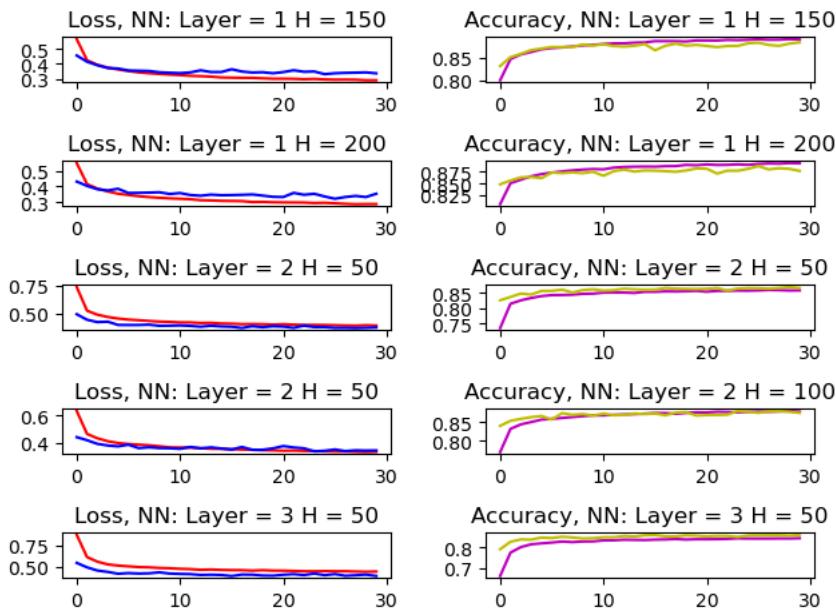
کلاس	کلاس اشتباه گرفته شده
T-shirt	Shirt
Trousers	Dress
Pullover	Shirt
Dress	T-Shirt

Coat	Pullover
Sandals	Sneakers
Shirt	T-shirt
Sneakers	Sandals
Bag	Shirt
Ankle Boot	Sandals

بیشترین کلاس های اشتباه گرفته نیز کلاس Tshirt و Shirt هستند، به این صورت که ۱۵۷ تا از T-shirt ها به عنوان Shirt بدست آمده اند.

اثر افزایش پیچیدگی به طور خلاصه، برای افزایش پیچیدگی می توان از ۲ تکنیک اضافه کردن لایه و اضافه کردن نوروز استفاده کرد. این دو افزایش می توانند در جهت بهتر شدن دقت و یا پایداری بر روی داده های آموزش منجرب شوند. در مقابل، برای رسیدن به دقت بالاتر بر روی تست، باید از تیوون کردن هایپر پارامتر ها نیز بهره برد. برای پیدا کردن یک بیس مدل می توان از MLP در حالتی که تعداد نوروز های لایه های مخفی با هم برابر اند استفاده نمود.

برای مثال شکل ۱۲ نشان می دهد نمودار های آموزش و تست با تغییر تعداد لایه و نوروز چگونه می توانند تغییر بکند:



شکل ها ۱۰: آموزش و تست با setting های مختلف

همانطور که مشاهده می شود، با یک لایه، افزایش نوروز ها منجر به دقت بهتر هم بر روی تست و هم آموزش می شود. به طور مشابه، این موضوع در تعداد لایه بالاتر نیز دیده می شود. در اینجا، افزایش تعداد لایه، لزوماً باعث نمی شود تا دقت بر روی داده تست بالاتر رفته و در واقع در تعداد یکسانی از epoch، به performance پایین تری بر روی این داده خاص دست می یابد. دلیل این اتفاق وجود gradient vanishing و این موضوع است که تعداد لایه بیشتر موجب تغییرات کمتر در لایه های مخفی نزدیک به ورودی شده و به تعداد بیشتری epoch نیاز است. همچنین ۳ لایه با تعداد نوروز کمتر ۲ لایه را outperform کرده است و نشان می دهد احتمالاً با زیاد کردن تعداد نوروز در آن حتاً از یک لایه نیز بهتر عمل کند که در قسمت های بعدی (قسمت ۴-۱) از این مدل استفاده شده است و به دقت بالاتری نیز می رسد. طبق قاعده کلی این داده خاص، زیاد کردن نوروز هر لایه و داشتن تعداد فردی لایه باعث بهتر شدن جواب می شود.

برای ۵ مدل بالا بهترین عملکرد در خصوص ۲۰۰ نوروز و داشتن یک لایه مخفی بوده است که اطلاعات کلاسیفیکیشن آذبه صورت زیر آمده است:

Test Report:				
	precision	recall	f1-score	support
0	0.81	0.85	0.83	1000
1	0.98	0.98	0.98	1000
2	0.79	0.80	0.79	1000
3	0.91	0.87	0.89	1000
4	0.82	0.79	0.81	1000
5	0.97	0.95	0.96	1000
6	0.72	0.65	0.68	1000
7	0.93	0.94	0.94	1000
8	0.87	0.98	0.93	1000
9	0.94	0.96	0.95	1000
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

شکل ها ۱۱: اطلاعات خلاصه برای طبقه بندی در حالت ۲۰۰ نوروز و یک لایه مخفی

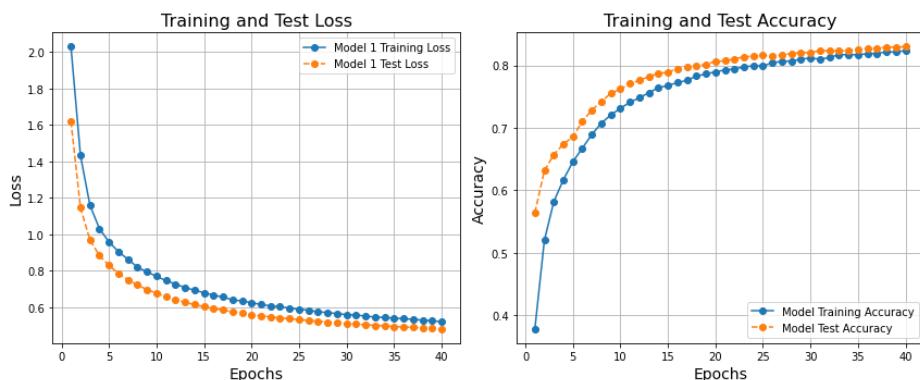
بررسی تغییر تک تک Config ها:

حال به عنوان حالت خاص و با توجه به بدست آورده بهترین حالت بالا می تواند دو شبکه مثال خاص زیر را پیشنهاد داد تا تاثیر تعداد لایه بیشتر مشخص گردد:

```
MLP_two_hidden_layers(
    (hidden_1): Linear(in_features=784, out_features=200, bias=True)
    (hidden_2): Linear(in_features=200, out_features=100, bias=True)
    (dropout): Dropout(p=0.3, inplace=False)
    (relu): ReLU()
    (clf): Linear(in_features=100, out_features=10, bias=True)
)
```

شکل ها ۱۲: شبکه با دو لایه

شبکه شکل ۱۲ شامل ۲۰۰ نوروز مخفی در لایه ۱ و ۱۰۰ نوروز مخفی در لایه ۲ است و نتیجه آموزش آن به شرح زیر است:



شکل ها ۱۳: آموزش شبکه شکل ۱۲

برای مقایسه، بهترین شبکه از مقایسه قبل قرار داده شده است:

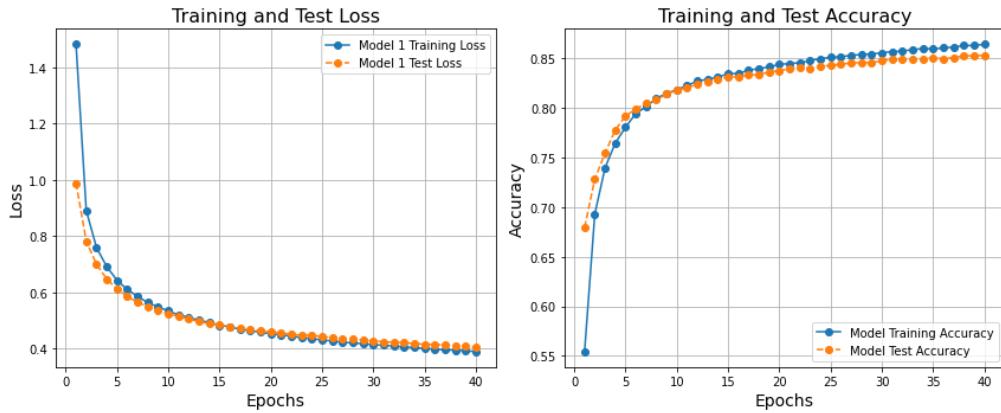
```

    MLP_one_hidden_layer(
        hidden_layer: Linear(in_features=784, out_features=200, bias=True)
        (dropout): Dropout(p=0.3, inplace=False)
        (relu): ReLU()
        (clf): Linear(in_features=200, out_features=10, bias=True)
    )

```

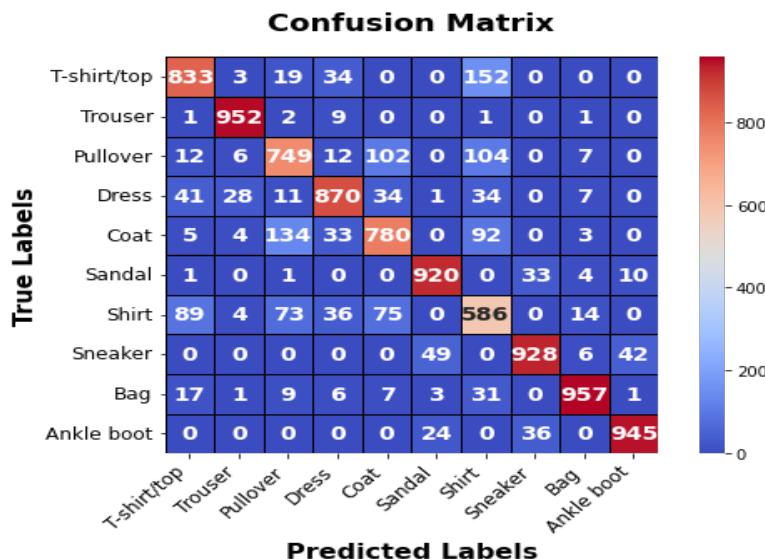
شکل ها ۱۴: بهترین شبکه با یک لایه

شکل ۱۴، یک شبکه با ۲۰۰ نورون مخفی در یک لایه را نشان می دهد. نتیجه آموزش این شبکه به صورت زیر است:



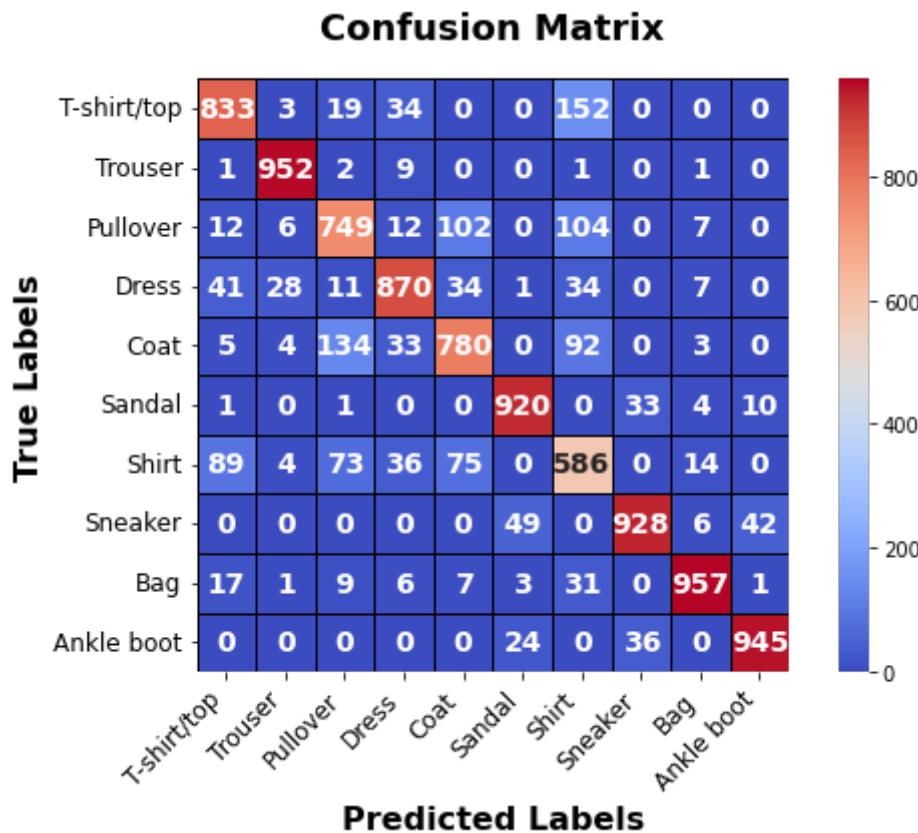
شکل ها ۱۵: نتایج آموزش شبکه ۱۴

به این ترتیب، شبکه شکل ۱۲ پر فورمنس پایین تری از شبکه شکل ۱۴ داشته و به دقت پایین تری می رسد. پس افزایش لایه بر روی Fashion MNIST تاثییر منفی و کم کننده از دقت داد. در انتها، ماتریس آشфтگی شبکه شکل ۱۴ آمده است که نشان می دهد در همه کلاس ها بهتر از بخش قبل و تعداد ۱۰۰ نورون عمل کرده ایم:



شکل ها ۱۶: ماتریس آشфтگی شبکه شکل ۱۲ (دو لایه)

همچنین ماتریس آشفتگی برای مدل شکل ۱۴ (مدل تک لایه ۲۰۰ نورون) در زیر آورده شده است:



شکل ها ۱۷: ماتریس آشفتگی شبکه شکل ۱۴

همانطور که دیده میشود، بیشترین تعداد اشتباه گرفته شدز در دو کلاس، در حالت تک لایه و ۲ لایه با ۲۰۰ نورون از ۱۵۲ تا در بدترین کلاس کاهش یافته است. البته تک لایه با ۲۰۰ نورون حتا بهتر از ۲ لایه عمل کرده است.

متريک های انتخاب پيکربندی

برای انتخاب بهترین پيکربندی، باید از METRIC ها استفاده بکنیم. اين INDEX ها تعیین می کنند که پيکربندی پیشنهاد داده شده چگونه عمل می کند.

این متريک ها در کيس خاص ما و برای طبقه بندی شامل

$$\text{ACCURACY} : \frac{TP}{\text{Total num of samples}} - \text{دقت}$$

۲ -percision & Recall :

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$3 - f1\text{-score} : \frac{2 \times precision \times recall}{precision + recall}$$

هستند. برای مثال شکل ۱۳ نشان می دهد همه اين متريک ها چگونه با افزایش تعداد نورون نسبت به شکل ۱۱ در بين تمام کلاس ها و به طور ميانگيري شده بهبود پيدا كرده و بنابراین شبکه بهتری برای اين داده خاص محسوب می شود.

## ۲-۱ آموزش ۲ مدل مختلف

۲ مدل با اطلاعات گفته شده سوال به صورت زیر هستند:

```

(MLP_one_hidden_layer(
    (hidden_layer): Linear(in_features=784, out_features=128, bias=True)
    (dropout): Dropout(p=0, inplace=False)
    (relu): ReLU()
    (clf): Linear(in_features=128, out_features=10, bias=True)
),
)

```

شکل ها ۱۸: ۱۲۸ نورون در لایه مخفی، بدون **regularizer** و **dropout**

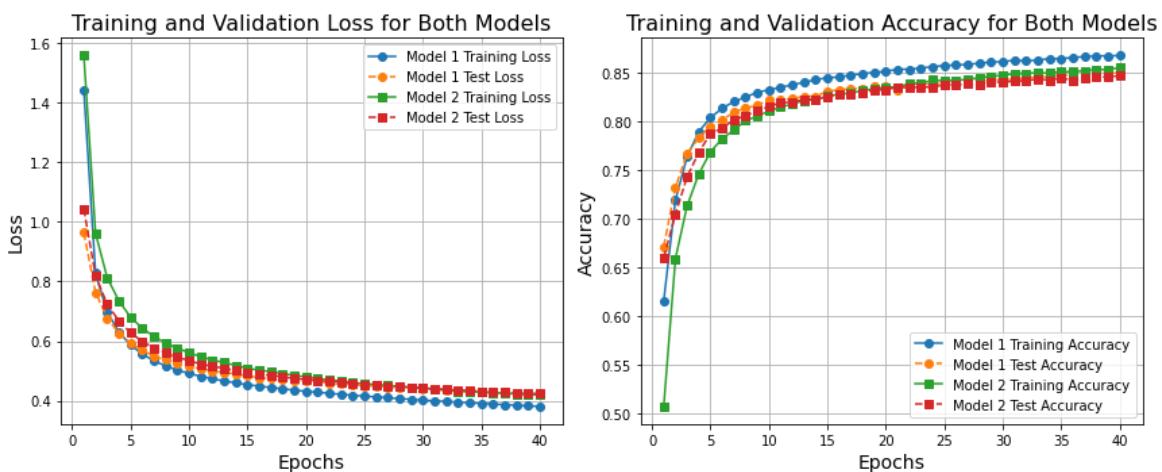
```

MLP_one_hidden_layer(
    (hidden_layer): Linear(in_features=784, out_features=48, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (relu): ReLU()
    (clf): Linear(in_features=48, out_features=10, bias=True)
)
)

```

شکل ها ۱۹: ۴۸ نورون در لایه مخفی، **regularizer** برابر با ۰.۲ و داشتن **dropout rate**

نتایج آموزش این دو مدل به شرح زیر است:

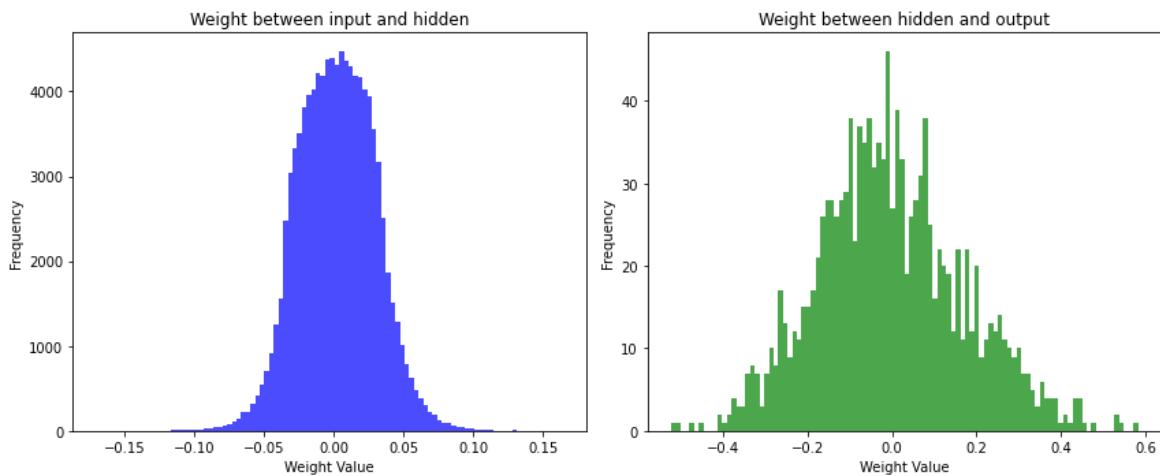


شکل ها ۲۰: آموزش ۲ مدل بر روی یک دیتا و مقایسه آموزش و تست

از آموزش این دو مدل دیده می شود که مدل شکل ۲۰ با داشتن ۴۸ نورون با اینکه به دقت پایین تری در تست دست یافته است، به دلیل وجود **regularizer** و **dropout** به دقت stable تر و بالاتری در تست دست یافته است، در حالی که در مدل با ۱۲۸ نورون شاهد overfit شدن هستیم.

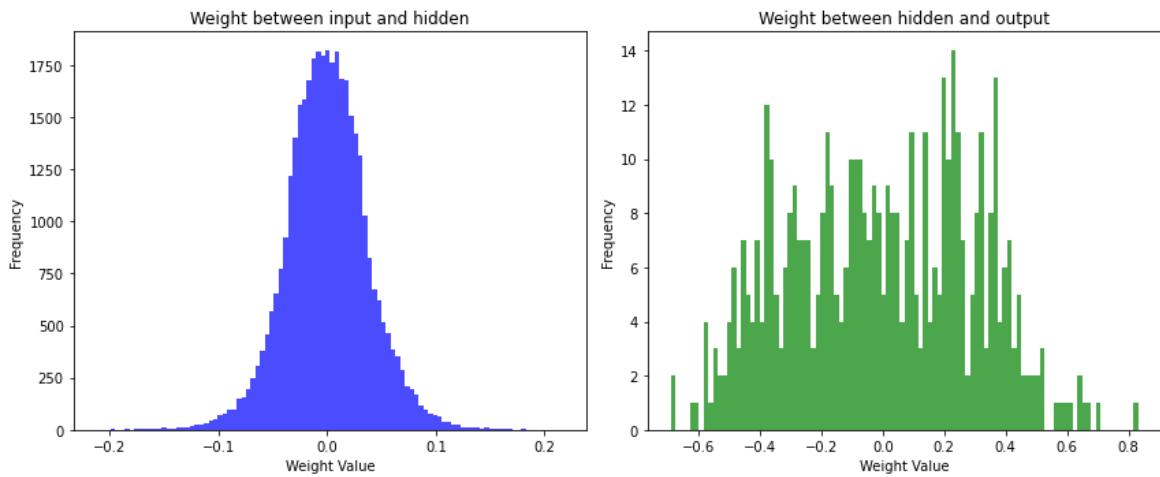
هیستوگرام وزن ها:

هیستوگرام وزن ها برای مدل ۱۲۸ نورون به صورت زیر است:



شکل ها ۲۱: هیستوگرام وزن مدل شکل ۱۹

هیستوگرام وزن ها برای مدل با ۴۸ نوروز به صورت زیر است:



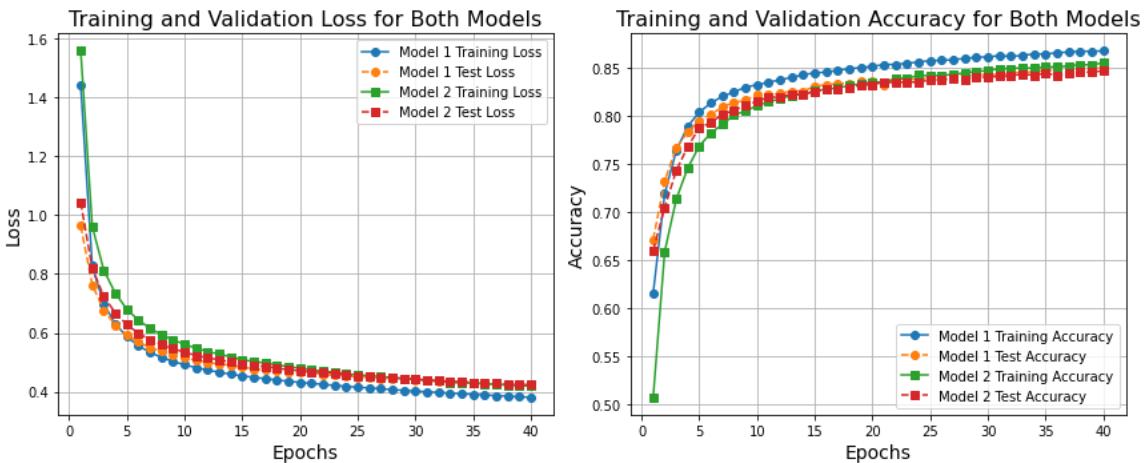
شکل ها ۲۰: هیستوگرام وزن شکل ۲۰: سمت راست بین ورودی و **hidden**, سمت چپ بین **hidden** و خروجی

طبق شکل ۲۲ و ۲۳، اثر regularizer در کوچک کردن وزن های شبکه است. این اثر در L<sub>2</sub>-regularizer تنبیه کردن وزن های بیشتر از اندازه است و خود باعث می شود تا overfitting اتفاق نیافتد و خیلی به داده آموزش نزدیک نشویم. شکل ۲۳ نیز توزیع روی اعداد کوچکتری نسبت به شکل ۲۰ دارد.

اثر dropout در اینجا خود را به صورت توزیع با نسبت انحراف معیار بیشتر در شکل ۲۳ خود را نشان داده است، به این صورت که در شکل ۲۳ وزن های بیشتری مقادیر دور از مرکز دارند، در حالی که این موضوع در شکل ۲۰ کمتر است. این موضوع نیز باعث می شود تا شبکه بتواند وزن های بزرگتری به feature های کمتر دیده شده به دلیل بزرگ بودن مقدار دیگر feature ها نسبت دهد که کمی از training accuracy می کاهد اما domain adaptation و به خاطر توزیع پراکنده تر test accuracy بالاتری دریافت می کند.

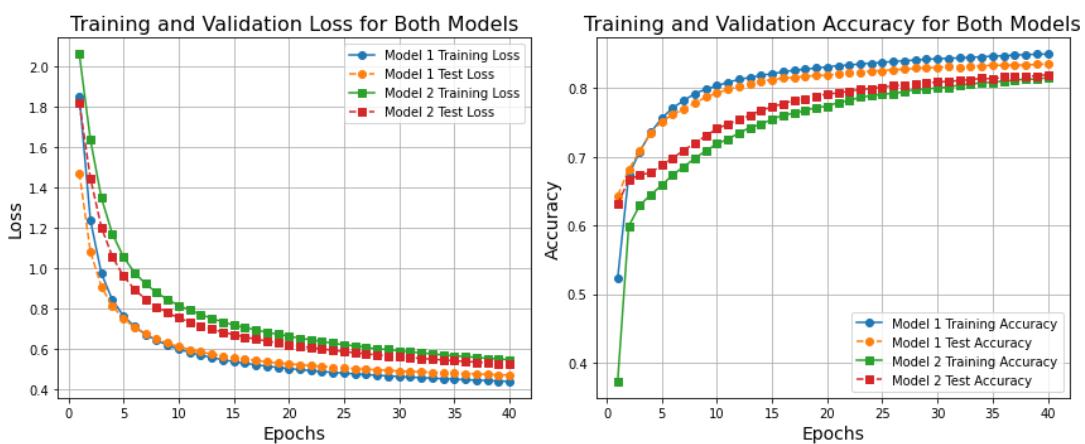
اثر بهینه ساز

مدل های قسمت بالا در استفاده از SGD به صورت زیر عمل می کنند:



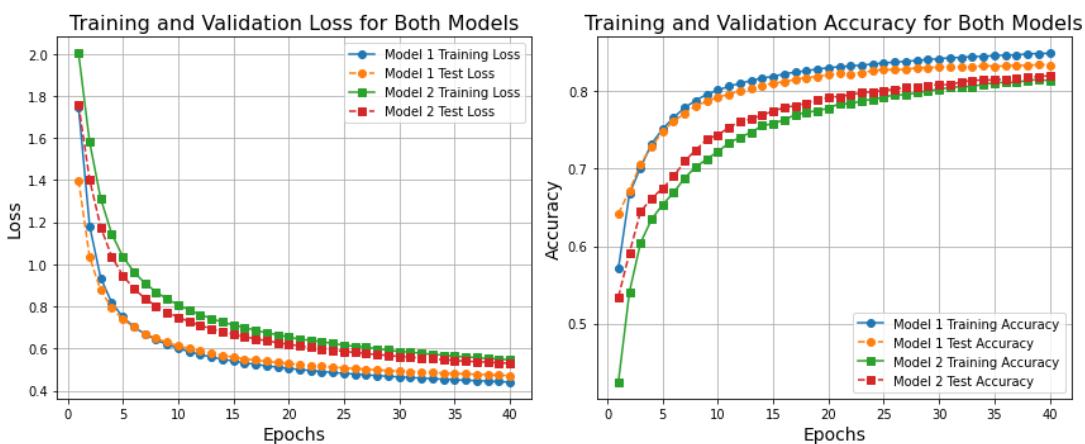
شکل ها ۲۳: بهینه ساز SGD

در حضور ADAM، بهینه سازی به شکل زیر انجام می پذیرد:



شکل ها ۲۴: بهینه ساز ADAM

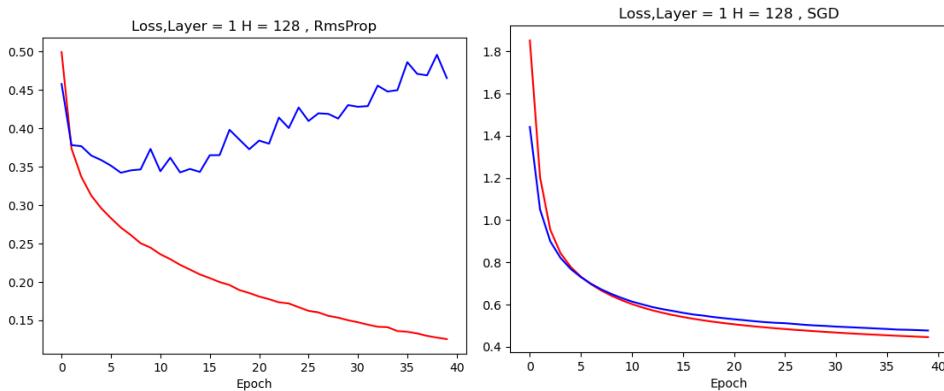
در بهینه ساز RMSprop، شکل زیر بدست می آید:



شکل ها ۲۵: بهینه ساز RMSprop

با استفاده از روش ADAM و یا RMSprop، به دلیل استفاده از moving average گرادیانها و مربع گرادیانها می توان انتظار داشت در برخی داده ها و به خصوص داده ها با توزیع Convex، عملکرد بهتری (حداقل نباید عملکرد بدتر

شود) بدست آید. در اینجا، روش SGD مسیر بهتری را پیدا کرده و loss تقریباً به ۰.۴ رسیده است، در حالی که RMSprop بدتر عملکرده و به بین ۰.۶ و ۰.۴ رسیده اند. این نشان می‌دهد گرادیان خالی بهتر عمل کرده است و توضیع داده‌ها بیشتر به حالت NON-Convex شباهت دارد. البته این موضوع وابسته به انتخاب حالت اولیه است. برای مثال دو عملکرد زیر از SGD و RMSprop با شروع از یک random state دیگر بدست می‌آید:



شکل ها ۲۶: تاثیر منفی Overfitting بر خطوط آبی برای تست هستند

این موضوع مورد انتظار است، چرا که در سیستم‌های غیر خطی، با شروع از یک نقطه اولیه و به دلیل توانایی وجود چندین نقطه تعادل و وجود focus point‌ها و limit cycle‌ها، می‌توان رفتار‌های چندگانه انتظار داشت.

۳.۱- الگوریتم بازگشت به عقب  
مدل استفاده شده در این قسمت به شرح زیر است:

```
MLP_mine(
    (layer_1): Linear(in_features=784, out_features=256, bias=True)
    (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (layer_2): Linear(in_features=256, out_features=128, bias=True)
    (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (layer_3): Linear(in_features=128, out_features=64, bias=True)
    (bn3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (clf): Linear(in_features=64, out_features=10, bias=True)
    (relu): LeakyReLU(negative_slope=0.01)
    (dropout): Dropout(p=0.2, inplace=False)
)
```

شکل ها ۲۷: شبکه استفاده شده در این بخش

شکل ۲۷ یک شبکه با ۳ لایه که هر لایه تعداد کمتری نوروز مخفی دارد و استفاده از Batch Normalization برای رسیدن به Validation Accuracy بهتر و همچنین برای رفتار یادگیری Smooth تر را نشان می‌دهد.

الگوریتم‌ها و تاثیر آنها:  
الگوریتم بهینه ساز Adam به صورت زیر کار می‌کند:

الگوریتم Adam (Adaptive Moment Estimation) از ترکیب دو ایده میانگین متحرک و momentum می‌کند. دو میانگین متحرک از گرادیانها نگه می‌دارد، یکی برای مقدار خود گرادیان (لحظه اول) و یکی برای مربع گرادیانها (لحظه دوم)، میانگین متحرک گرادیانها (لحظه اول) باعث می‌شود که شبکه شتاب بیشتری به سمت کمینه پیدا کند، و میانگین مربع گرادیانها (لحظه دوم) نرخ یادگیری را برای هر پارامتر تنظیم می‌کند تا از نوسانات جلوگیری کند. Adam با تنظیم نرخ یادگیری و اعمال میانگین متحرک در دو سطح، معمولاً به نرخ همگرایی سریع‌تری می‌رسد. ممکن است در برخی موارد به یک مینیمم محلی ضعیف همگرا شود.

الگوریتم به صورت زیر کار می کند:  
میانگین متحرک گرادیانها (لحظه اول):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

میانگین متحرک مربع گرادیانها (لحظه دوم):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

اصلاح سوگیری:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

به روزرسانی پارامترها:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

### الگوریتم NAdam

یک ورزش بهبود یافته Adam که از شتاب نستروف استفاده میکند. منجر به حرکت پیش‌بینی شده‌تری در جهت کمینه می‌شود.

تنها تفاوت با Adam در میانگین متحرک مربع گرادیانها است:

میانگین متحرک گرادیانها با مومنتوم نستروف:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

نسبت به Adam در بسیاری از موارد همگرایی بهتری دارد و به ویژه برای شبکه‌های عمیق و پیچیده مناسب‌تر است. کمی پیچیده‌تر است و ممکن است بهینه‌سازی بیشتری نسبت به Adam در برخی مسائل نداشته باشد.

### الگوریتم RMSprop

RMSprop (Root Mean Square Propagation) توسط جف هینتون معرفی شد و در اصل برای حل مشکلاتی SGD پیش می‌آیند، طراحی شده است.

میانگین مربع گرادیانها را برای هر پارامتر نگه می‌دارد و این میانگین به عنوان وزن در جهت‌گیری گرادیان استفاده می‌کند. ایده اصلی این است که در هر جهت، مقیاس تغییرات را با میانگین متحرک مربع گرادیانها تنظیم کند. این الگوریتم با توجه به میانگین متحرک مربع گرادیانها، مقیاس گام‌ها را در هر جهت کاهش می‌دهد تا مشکل نوسانات زیاد در جهت‌گیری کاهش یابد.

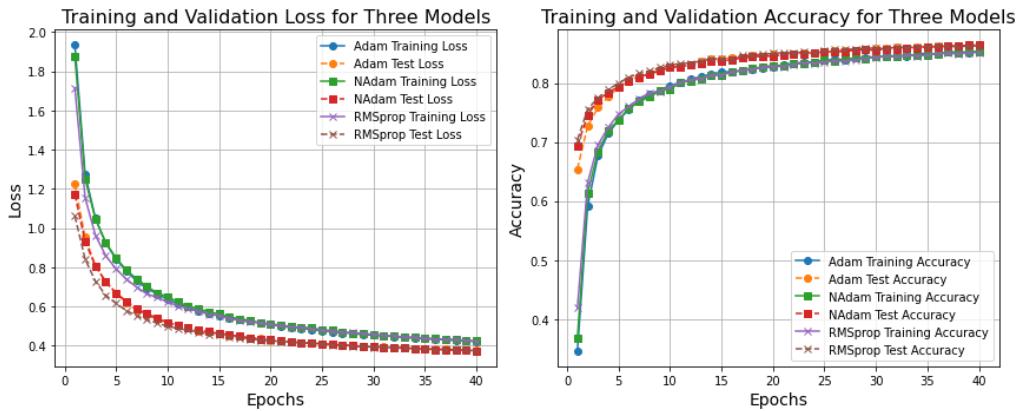
به روزرسانی مقدار میانگین مربع گرادیانها:

$$E[g^2]t = \beta E[g^2]t - 1 + (1 - \beta)g_t^2$$

به روزرسانی پارامترها:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

نتایج آموزش با روش های مختلف به شرح زیر است:



شکل ها ۲۸: مقایسه روش های مختلف

به طور کلی، در اینجا روش RMSprop توانسته است به بالاترین دقت دست پیدا بکند و Adam از همه ضعیف تر از نظر دقیقی بوده است. همچنین از نظر سرعت همگرایی RMSprop سریعتر بوده است، پس از آن NAdam قرار داشته و در انتهای Adam بوده است. همچنین مقدار نهایی Loss در RMSprop کمتر از Adam است. البته تفاوت Nadam و RMSprop بسیار زیاد نبوده و هر دو خوب عمل کرده اند.

### جستجوی بیزی

جستجوی بیزی (Bayesian Optimization) یک روش کارآمد برای تنظیم هایپرپارامترهای شبکه‌های عصبی است. این روش به جای جستجوی تصادفی یا جستجوی شبکه‌ای، بهینه‌سازی را به صورت هوشمندانه انجام می‌دهد و با استفاده از توزیع‌های احتمالاتی، سعی در پیش‌بینی بهترین ناحیه از فضای هایپرپارامترها برای آزمایش بعدی دارد.

### تأثیر جستجوی بیزی در تنظیم هایپرپارامترهای شبکه‌های عصبی:

بهبود دقت و عملکرد مدل: جستجوی بیزی با استفاده از اطلاعات قبلی هر آزمون تصمیم می‌گیرد که کدام مجموعه از هایپرپارامترها را برای مرحله‌ی بعدی آزمایش کند. این روش معمولاً منجر به یافتن مقادیر بهینه‌تر هایپرپارامترها (مانند نرخ یادگیری، تعداد نمونه، نرخ درآپاوت و ...) شده و باعث افزایش دقت مدل می‌شود.

کاهش تعداد آزمایش‌ها: در روش‌های ساده‌تر مانند جستجوی شبکه‌ای یا تصادفی، تمامی ترکیبات ممکن هایپرپارامترها به صورت کامل یا نیمه‌تصادفی بررسی می‌شوند. اما در جستجوی بیزی، تعداد آزمایش‌های موردنیاز به طرز چشمگیری کاهش می‌یابد، زیرا این روش فقط ترکیب‌هایی را که احتمالاً موثرتر هستند بررسی می‌کند.

استفاده‌ی بهینه از منابع محاسباتی: جستجوی بیزی به دلیل کاهش تعداد آزمایش‌ها و تمرکز بر مقادیر محتمل‌تر، باعث صرفه‌جویی در زمان و هزینه محاسباتی می‌شود. این امر به ویژه در شبکه‌های عصبی پیچیده و عمیق که هزینه محاسباتی بالایی دارند، بسیار مهم است.

همگرایی سریع‌تر به سمت مقادیر بهینه: با انتخاب هوشمندانه‌تر هایپرپارامترها، جستجوی بیزی معمولاً باعث می‌شود که شبکه با سرعت بیشتری به سمت تنظیمات بهینه همگرا شود. این امر منجر به دستیابی سریع‌تر به عملکرد مطلوب مدار می‌شود.

مدیریت تعادل بین جستجو (Exploration) و بهره‌برداری (Exploitation): جستجوی بیزی به طرز هوشمندانه‌ای بین امتحان کردن مقادیر جدید (جستجو) و تمرکز روی مقادیر احتمالی بهینه (بهره‌برداری) تعادل برقرار می‌کند. این ویژگی باعث می‌شود که به دام مقادیر زیر بهینه نیفتند و بهترین هایپرپارامترها را کشف کنند.

هزینه محاسباتی اولیه: در مراحل اولیه جستجوی بیزی، باید چندین ترکیب از هایپرپارامترها را آزمایش کرد تا اطلاعات اولیه کافی برای پیش‌بینی‌های بعدی جمع‌آوری شود. این امر ممکن است در ابتدا زمانبر باشد.

نیاز به تنظیم اولیه: انتخابتابع احتمالاً (مثل Gaussian Process) و دیگر تنظیمات اولیه جستجوی بیزی، می‌تواند بر عملکرد آن تأثیرگذار باشد و نیاز به تنظیم دقیق دارد.

مراحل انجام جستجوی بیزی:

تعریف تابع هدف: تابع هدف همان معیاری است که قصد بهینه‌سازی آنرا داریم (مثل دقت مدل یا کاهش خطای). هر بار که ترکیبی از هایپرپارامترها را انتخاب و آنرا در مدل‌اجرا می‌کنیم، تابع هدف مقدار مشخصی را برمی‌گرداند. هدف جستجوی بیزی این است که این تابع را با کمترین تعداد ارزیابی بهینه‌سازی کند.

انتخاب یک تابع جانبی (Surrogate Function):

تابع جانبی، تابعی است که فضای جستجو را تخمین می‌زند و به ما کمک می‌کند که بهترین ناحیه را برای بررسی‌های بعدی پیدا کنیم. پرکاربردترین تابع جانبی در جستجوی بیزی، فرآیند گاوی (Gaussian Process) است. این تابع، یک توزیع احتمالی بر روی تابع هدف ایجاد می‌کند و به طور پویا با اضافه شدن داده‌های جدید، به روز می‌شود. انتخاب معیار اکتشافی (Acquisition Function):

معیار اکتشافی:

تصمیم می‌گیرد که بهترین نقطه بعدی برای آزمایش کدام است. در اینجا دقت بر روی تست انتخاب می‌شود.

اجرای آزمایش با هایپرپارامتر پیشنهادی:

در هر گام، نقطه جدیدی که تابع اکتشافی پیشنهاد داده است، به عنوان ورودی به تابع هدف ارسال می‌شود. مثلاً ترکیب خاصی از هایپرپارامترها را در مدل شبکه عصبی آزمایش می‌کنیم و نتیجه را (مانند دقت مدل یا خطای) ثبت می‌کنیم.

بهروزرسانی تابع جانبی با داده‌های جدید:

پس از آزمایش نقطه جدید و به دست آمدن مقدار تابع هدف، مدل جانبی (مثلاً فرآیند گاوی) به روز می‌شود تا توزیع بهتری از فضای جستجو ایجاد کند. به این ترتیب، مدل جانبی هر بار با داده‌های جدید دقیق‌تر می‌شود.

برای مثال شکل بعد، انجام جست و جوی بیزی بر روی یک مدل ۳ لایه برای پیدا کردن dropout و learning rate را نشان می‌دهد:

iter	target	dropout	epochs	learni...	opt
1	0.6826	0.3124	14.51	0.0732	1.395
2	0.8335	0.2468	6.56	0.005818	2.465
3	0.8695	0.3803	12.08	0.002068	2.88
4	0.8093	0.4497	7.123	0.01819	-0.2664
5	0.734	0.2913	10.25	0.0432	0.1649
6	0.8202	0.3943	12.02	0.01762	2.889
7	0.8687	0.4197	12.11	0.002144	2.993
8	0.733	0.2013	5.404	0.04665	0.4084
9	0.7343	0.2951	12.24	0.05577	2.912
10	0.4976	0.4905	12.11	0.09871	2.961
11	0.7368	0.4815	12.06	0.04502	2.994
12	0.7978	0.4326	13.82	0.02272	0.4897
13	0.8794	0.3667	12.0	0.0004296	2.887
14	0.7423	0.2019	9.593	0.09767	1.107
15	0.8077	0.458	13.87	0.01162	0.4624
16	0.7395	0.3477	12.1	0.03041	2.931
17	0.8719	0.2619	14.7	0.001443	-0.7222
18	0.8484	0.3345	5.879	0.005181	2.348
19	0.6461	0.2249	12.98	0.0922	2.886
20	0.6786	0.4108	12.04	0.02754	2.888
21	0.3692	0.4961	11.05	0.09006	2.028
22	0.8522	0.4142	12.48	0.004341	0.6109
23	0.7756	0.4536	12.18	0.02955	1.163
...					
28	0.7866	0.4842	11.04	0.01688	0.7607
29	0.8066	0.3604	12.11	0.01521	2.854
30	0.6002	0.2184	10.16	0.04271	2.812

شکل ها ۲۹: یادگیری بیزی بر روی مدل ۳ لایه

همانطور که دیده می شود، می توان این پارامترها را با روشی سیستماتیک و بهتر از random search و یا grid search که روی بازه محدودیت دارند انجام داد و دقت مدل را بهتر کرد. در مثالی دیگر با برای مقایسه Grid search جستجوی بیزی، برای یک شبکه تک لایه کوچکتر، برای grid search یک بازه کوچکتر در نظر گرفته شد و زمان بسیار بیشتری برای شبکه کوچک تر برداشت شد:

```
best setting for the network is :
number of hidden neurons: 62
dropout rate: 0.1
learning rate: 0.001
```

شکل ها ۳۰: جواب Grid search

در شکل ۳۰ حداقل accuracy برابر با ۰.۸۶۹۹ بوده، در حالی که در شکل ۲۹ به دقت ۰.۸۷۹۴ نیز می رسیم. این موضوع در جاهایی که تاثیر پذیری از hyperparameter بالاتر و شبکه بزرگتر است، حتی خود را بیشتر نشان می دهد و جستجوی بیزی می تواند در زمان کمتر به دقت برابر و یا حتی بیشتر روی بازه بزرگتری از Hyperparameter ها برسد.

#### ۱.۴. بررسی هایپر پارامتر های مختلف معماری انتخاب شده به صورت زیر است:

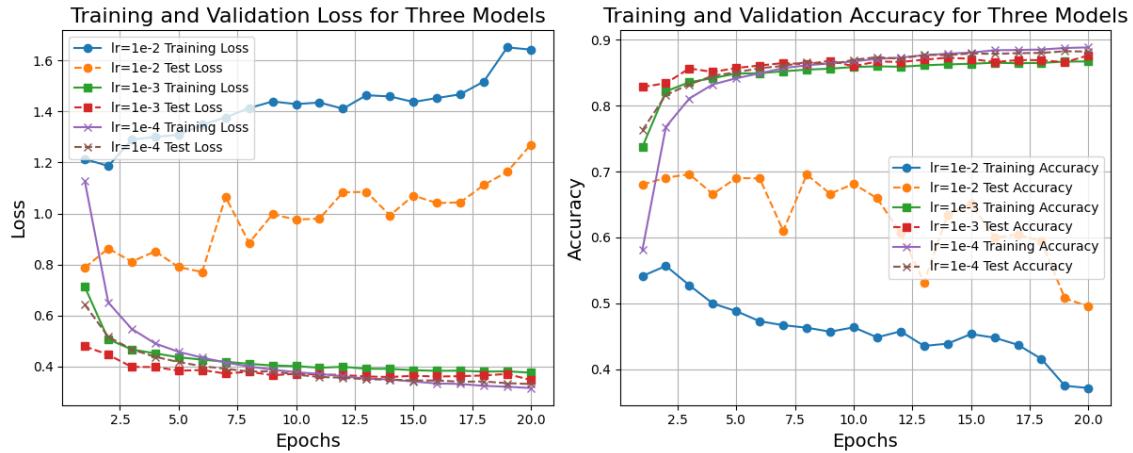
```
good_MLP(
    (first_layer): Linear(in_features=784, out_features=512, bias=True)
    (second_layer): Linear(in_features=512, out_features=256, bias=True)
    (third_layer): Linear(in_features=256, out_features=128, bias=True)
    (clf): Linear(in_features=128, out_features=10, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
    (relu): ReLU()
)
```

شکل ها ۳۱: یک معماری مناسب

دلیل انتخاب این معماری، استفاده از تعداد بالای نورون و لایه و همچنین بر اساس این موضوع است که افزایش تعداد نورون قطعاً پروفورمنس را روی این داده زیاد می‌کند و همچنین استفاده از تعداد فردی از لایه بر روی این داده، جواب بهتری را حاصل می‌کند.

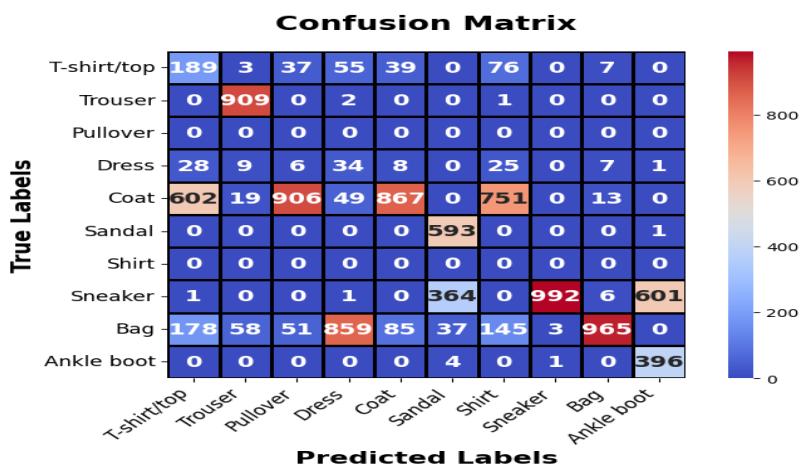
تغییر نرخ یادگیری:

نتیجه یادگیری این مدل با learning rate مختلف در شکل بعد آمده است:

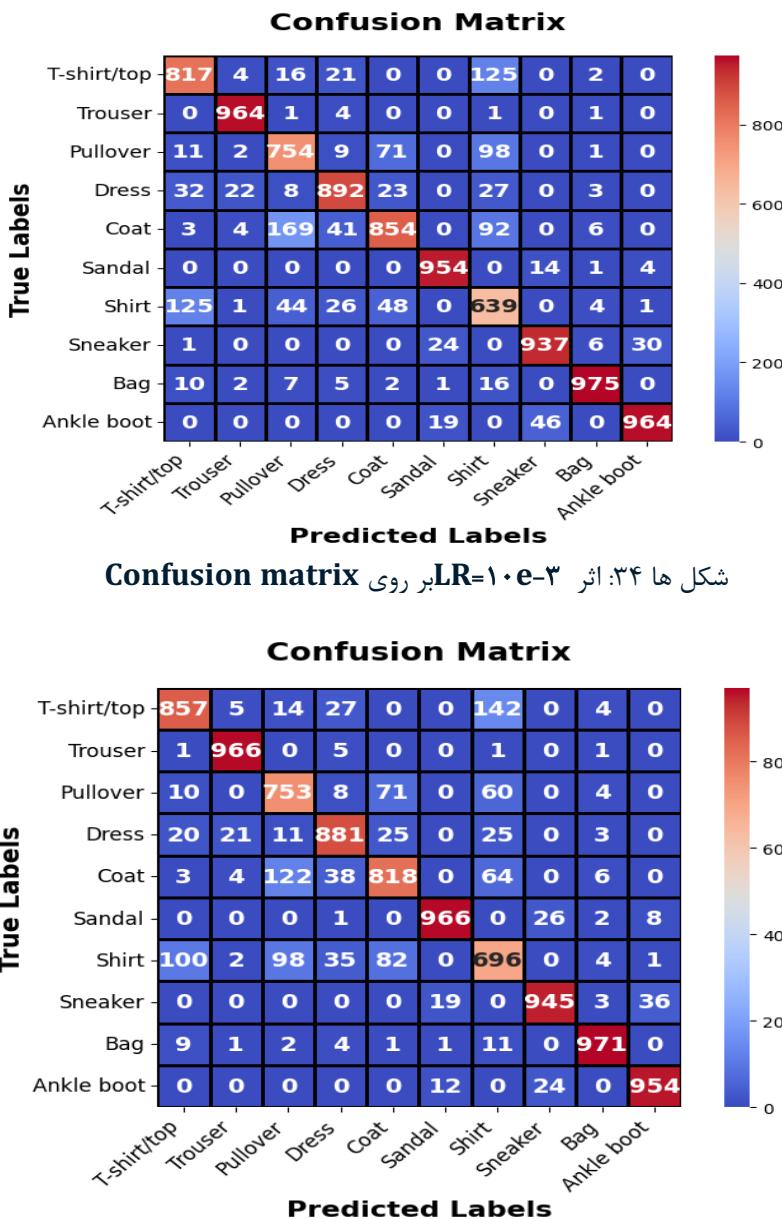


شکل ها ۳۲: اثر ضریب یادگیری

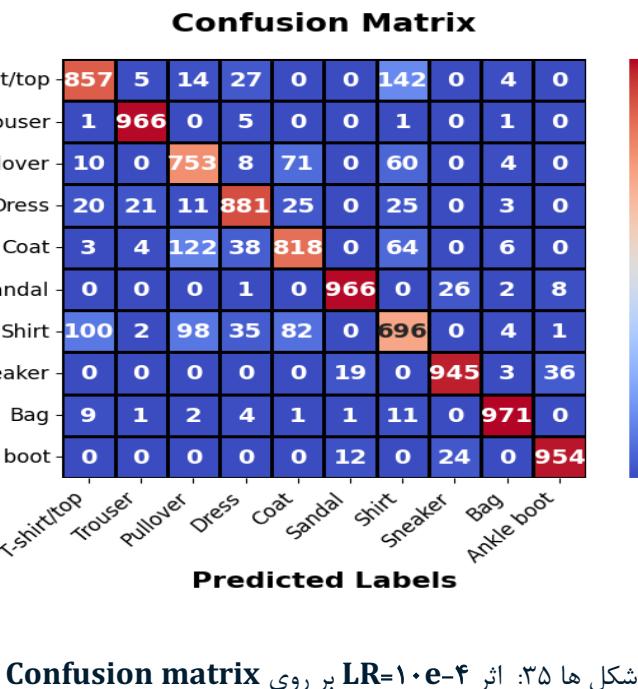
در اینجا با کم کردن نرخ یادگیری، به دقت بهتر و همگرایی و همچنین loss کمتر می‌رسیم. البته تغییر و حتا کوچکتر کردن از  $10^{-3}$  به  $10^{-4}$  اثر خیلی محسوسی ندارد و حتی با کم کردن بیشتر ممکن است تعداد بیشتری epoch برای رسیدن به مقدار یکسان از دقت باشیم.



شکل ها ۳۳: اثر  $LR=10^{-2}$  بر روی Confusion matrix



شکل ها ۳۴: اثر  $LR=1 \cdot 10^{-3}$  بر روی Confusion matrix

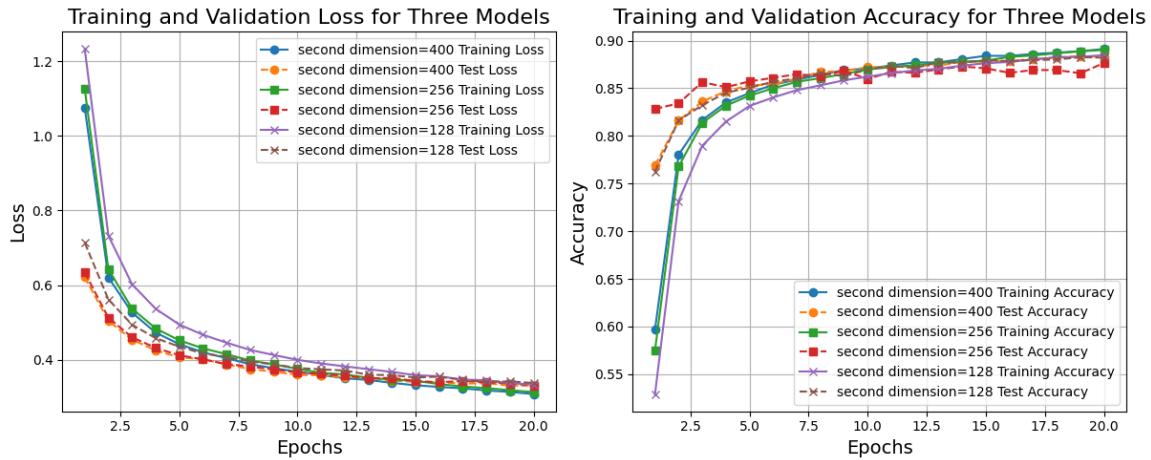


شکل ها ۳۵: اثر  $LR=1 \cdot 10^{-4}$  بر روی Confusion matrix

با توجه به شکل ۳۳ تا ۳۵ ، با کم کردن  $LR$ ، از تشخیص دادن بسیاری از کلاس ها به عنوان کلاس مشابه خود که یک local optimum محسوب می شود (برای مثال Shirt به عنوان Coat و یا بسیاری از کلاس ها به عنوان Bag) به سمت تشخیص دقیق تر پیش می رویم. دلیل آذین است که با تغییرات کم گرادیان می توان ۲ کلاس مشابه را تشخیص داد ولی وقتی نرخ یادگیری بالا است، روی تشخیص میانگین آنها بیشتر حرکت می کنیم و راحت تشخیص داده نمی شوند. (اثر پیش بینی میانگین و Moving Average فیت کردن به جای Fit شدن روی ویژگی ها)

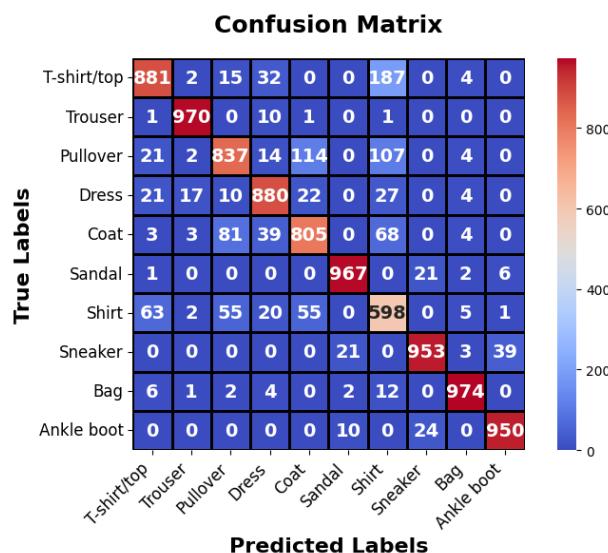
تغییر تعداد نوروز

در شبکه شکل ۳۱ با تغییر تعداد نوروز لایه میانی (به عنوان لایه فضای latent ورودی و تاثیر برابر از ورودی و گرادیان خروجی) داریم:

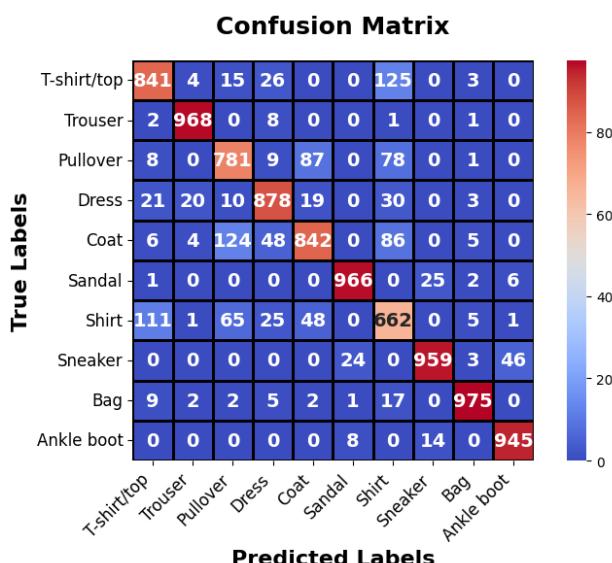


شکل ها ۳۶: اثر تغییر تعداد نوروز ها

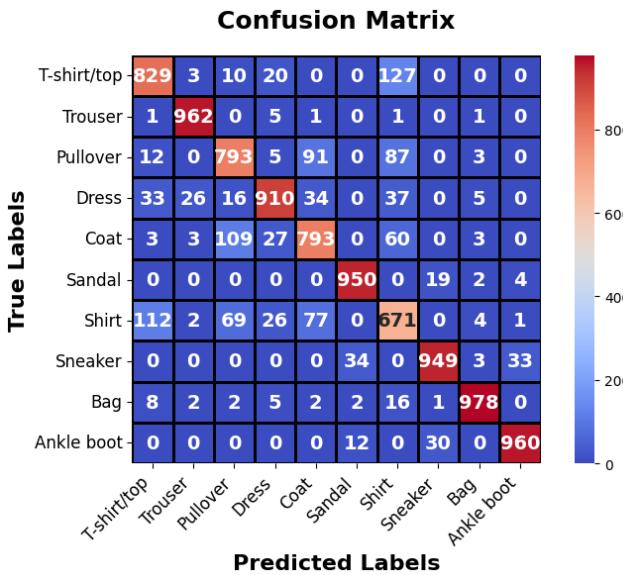
تغییر تعداد نوروز تاثیر خیلی محکمی نداشت و با زیاد شدن آن دقیقیت کمی بهتر می شود.



شکل ها ۳۷: تعداد نوروز ۱۲۸



شکل ها ۳۸: تعداد نوروز

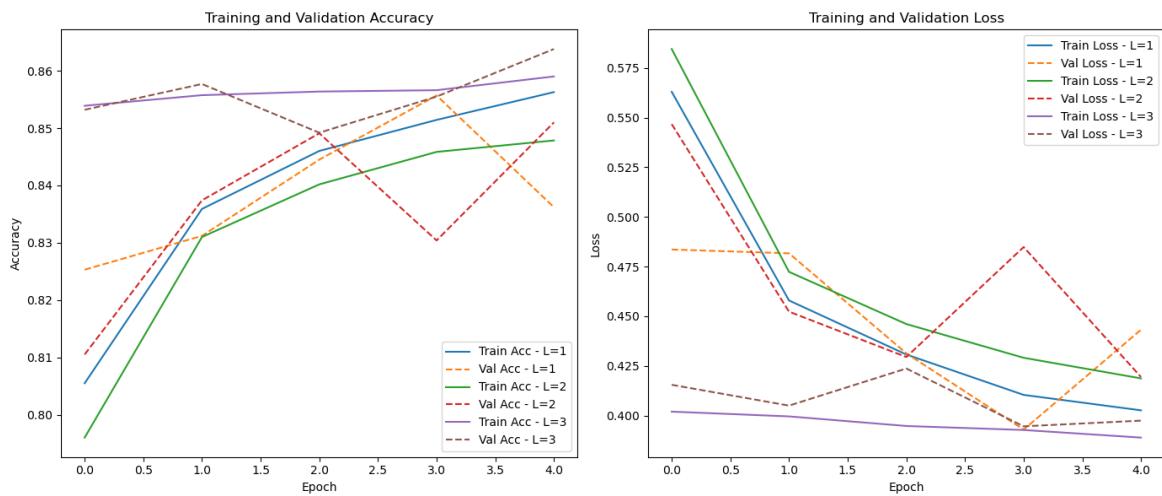


شکل ها ۳۹: تعداد نوروز

با توجه به شکل های ۳۷ تا ۳۹، با افزایش تعداد نورونه کلاس هایی مانند T-shirt و Coat و Pullover که به هم بسیار نزدیک (برعکس قسمت قبل که صرفا باید کمی مشابهت می داشتند) هستند، به دلیل زیاد کردن فیچر های latent داخلی شبکه و ریز تر کردن ناحیه های مقایسه، بهتر تشخیص داده می شوند.

عوض کردن تعداد لایه:

با عوض کردن تعداد لایه، داریم:



شکل ها ۴۰: تاثیر تعداد لایه

همانطور که دیده می شود، تعداد لایه فرد نتیجه بهتری را می دهد.

## استفاده از جستجوی تصادفی

```
1 param_grid = {
2     'optimizer_lr': [1e-3, 1e-4],
3     'module_n_layers': [1, 2],
4     'module_first_dimension': [128, 256, 512],
5     'module_second_dimension': [32, 64, 128],
6 }
```

با استفاده از Randomized search روی پارامتر های زیر: و بر روی شبکه :

ساختار زیر با بالاترین عملکرد بدست می آید:

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
▶      RandomizedSearchCV
▶      best_estimator_: NeuralNetClassifier
    ▶      NeuralNetClassifier
<class 'skorch.classifier.NeuralNetClassifier'>[initialized](
    module_=random_model(
        layers: ModuleList(
            (0): Linear(in_features=784, out_features=512, bias=True)
            (1): Linear(in_features=512, out_features=10, bias=True)
        )
        (act): ReLU()
        (dropout): Dropout(p=0.3, inplace=False)
    ),
    14 |     x = x.view(x.size(0), -1)
    15 |     for layer in self.layers[:-1]:
    16 |         x = self.dropout(self.act(layer(x)))
    17 |     x = self.layers[-1](x)
    18 |     return x
```

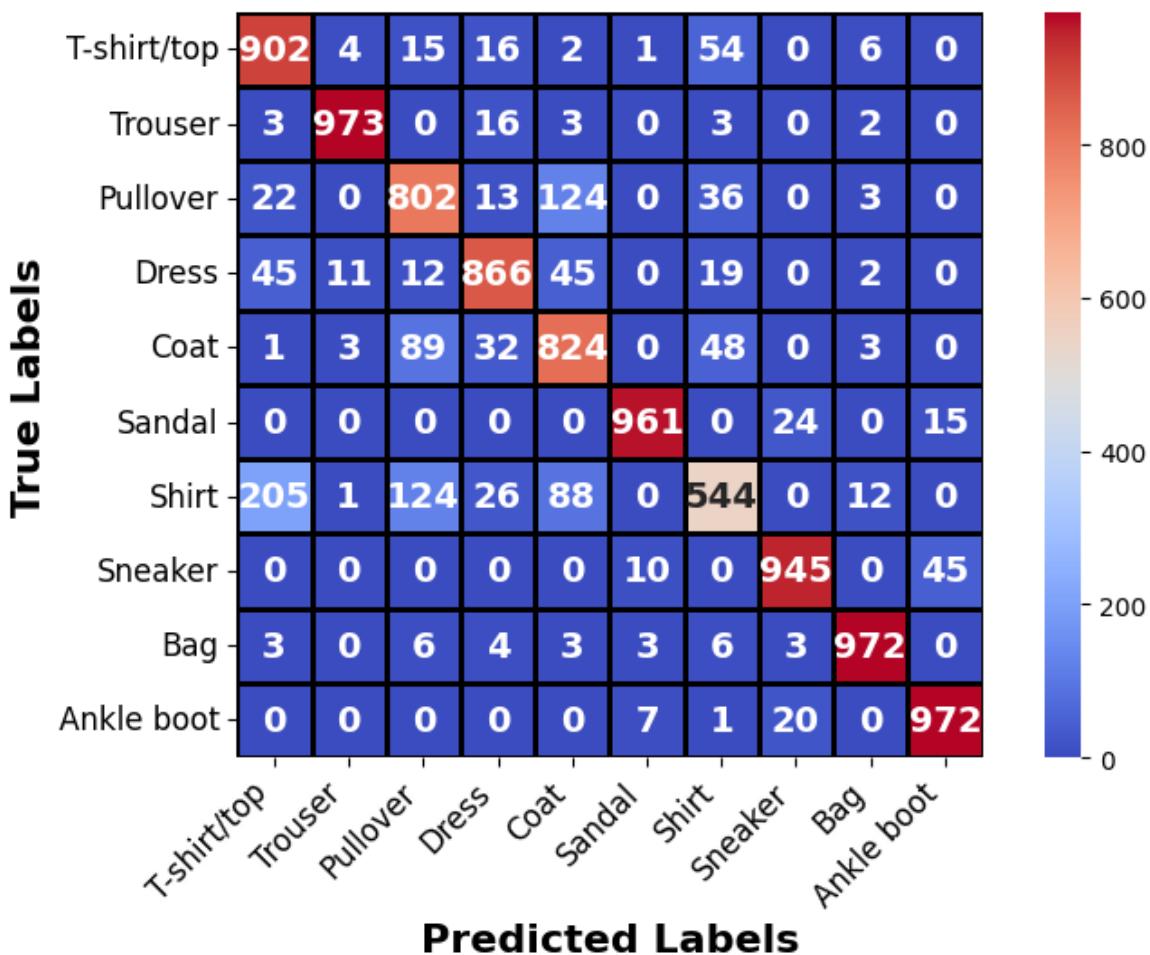
همچنین داریم:

The best parameters are: {'optimizer\_lr': 1e-3, 'module\_second\_dimension': 128, 'module\_n\_layers': 1, 'module\_first\_dimension': 512}

که برای این شبکه، امتیاز Accuracy score is ۰.۸۷۶۱ است.

همچنین ماتریس Confusion به صورت زیر در می آید:

## Confusion Matrix



شکل ها ۴۱: ماتریس آشفتگی حاصل از بهترین حالت Random Search

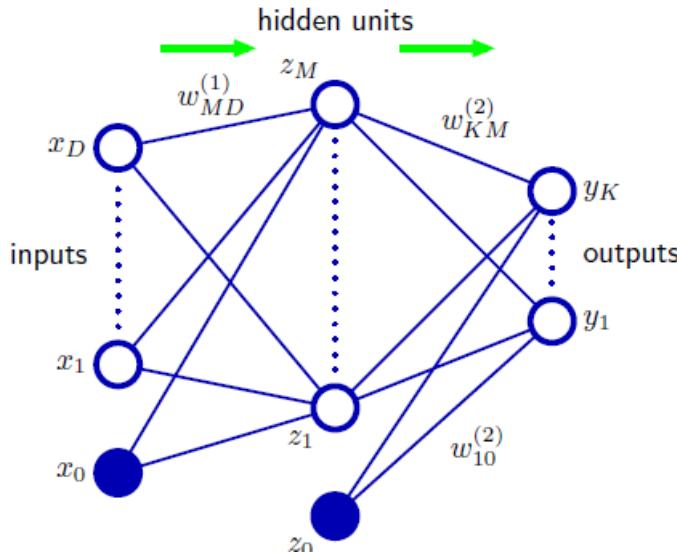
کاربرد این روش، این است که ابتدا یک ایده کلی درباره بازه مورد انتظار hyperparameter های شبکه با انتخاب از مجموعه ای شامل هایپرپارامتر های common Grid در یک بزرگتر داشته و یک بازه بهتر را به عنوان نقطه شروع تصادفی برای tune کردن هایپرپارامتر ها در روش های Grid Search و یا جستجوی بیزی داشته و باعث شویم همگرایی سریعتری رخدهد.

## پرسش ۲ - آموزش و ارزیابی یک شبکه عصبی ساده

آموزش یک شبکه عصبی.

در این تمرین قصد داریم تا با استفاده از مثال کتاب بیشتر یک شبکه عصبی را پیاده سازی کنیم. در ابتدا فرمولهای استفاده شده برای این مثال را خواهیم نوشت و سپس توابع را براساس آنها پیاده سازی خواهیم کرد.

شکل نشان دهنده مبنای کار ما خواهد بود.



شکل ها 42: شکل ۱. مبنای یک شبکه عصبی با یک لایه

براساس شکل بالا روابط را برای مثال خود شرح خواهیم داد. فرض کنیم که تابع فعالسازی لایه خروجی تابع خطی و لایه مخفی تابع تانژانت هایپربولیک باشد خواهیم داشت:

$$a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i$$

$$z_j = \tanh(a_j)$$

$$y_k = \sum_{j=0}^M w_{kj}^{(2)} z_j$$

اگر تابع هزینه را MSE درنظر بگیریم خواهیم داشت:

$$E_n = \frac{1}{2} \sum_{k=1}^K (y_k - t_k)^2$$

برای محاسبه  $\delta$  هر نод خروجی، خواهیم داشت:

$$\delta_k = y_k - t_k$$

برای محاسبه  $\delta$  هر نod میانی در لایه مخفی، خواهیم داشت:

$$\delta_j = (1 - z_j^2) \sum_{k=1}^K w_{kj} \delta_k$$

حال با استفاده از این فرمولها سراغ کد میرویم.

#### ۱-۲. تابع forward

در این تابع قصد داریم تا با استفاده از ورودی های  $X, W_1, W_2$  را بگیریم و خروجی های  $y_{pred}$ ,  $Z$  را برگردانیم. توجه داشته باشید که این کدهای خواسته شده را در قالب Object Oriented زده شده است. تابع فوروارد را در شکل زیر می بینید:

```
def _forward(self, W_1:np.array, W_2:np.array, X:np.array) -> dict:
    """
    Compute the activations backward!
    Parameters:
        - X: Input vectors contain N samples and D features. Shape is: [N, D]
        - W_1: Weights between input layer and hidden layer, shape of W_1 is [M, D]
        - W_2: Weights between hidden layer and output layer, shape of W_2 is [1, M]
    Return:
        A dictitony where
        1. Contain a N*1 vector for predicting given samples(y_pred)
        2. Contain a N*M matrix for activations of hidden neurons(z)
    """
    a = np.dot(X, W_1.T)
    z = self._activation_computation(a)
    y_in = np.dot(z, W_2.T)
    y_preds = y_in
    return z, y_preds
```

شکل ها 43: شکل ۲. تابع فوروارد

همانطور که در صورت مساله گفته شده است، برای هر ورودی براساس فرمول بیان شده، ورودی  $X$  را میگیریم و در وزن های بین لایه ورودی و مخفی ضرب می کنیم. در وهله بعدی خروجی لایه مخفی را میگیریم و در وزنهای بین لایه مخفی و لایه خروجی ضرب خواهیم کرد و درنهایت خروجی های خواسته شده را برگردانیم.

#### ۱-۲. تابع Backward

در این تابع ورودی های ( $X, y, M, iters, lr$ (learning rate)) را میگیریم و در ادامه شبکه خود را ساخته و مقدار دهی اولیه می کنیم. در هر تکرار یه نمونه را گرفته و از تابع فوروارد استفاده میکنیم و سپس خطأ رو پس انتشار میدهیم. برای آنکه کار آسانتر شود یک تابع کمکی دیگری برای این تابع درنظر میگیریم که صرفا وزن های جدید را بعد از پس انتشار داده بازمیگرداند.

شکل زیر حاکی از آن است:

```
def _backpropagation(self, W_1:np.array, W_2:np.array, target:np.array, net_output:np.array, z:np.array, lr:float, sample:np.array) -> tuple:
    """
    Back propagating the loss error and calculate the gradients
    Parameters:
        W_1: Current weights between input and hidden layers
        W_2: Current weights between hidden and output layers
        error: calculated error for network
    Return:
        W_1: Updated weights between input and hidden layers
        W_2: Updated weights between hidden and output layers
    """
    target = target.reshape(1)
    delta_last_layer = net_output - target
    delta_hidden_layer = (1-np.square(z)) * (W_2 * delta_last_layer )
    new_W_1, new_W_2 = np.copy(W_1), np.copy(W_2)
    new_W_2 -= lr * delta_last_layer * z
    new_W_1 -= lr * np.dot(delta_hidden_layer.T, sample.reshape(-1, 1).T)
    return new_W_1, new_W_2
```

شکل ها 44: شکل ۳. تابع کمکی که وزن های جدید را بازمیگرداند

حال می‌توانیم تابع backward را به کمک تابع بالا پیاده کنیم، تابع backward به صورت زیر پیاده‌سازی شده است.

```
def backward(self, X:np.array, y:np.array, M:int, iters:int, lr:float) -> dict:
    """
    Create your model here with given X,y for train set and train it.
    Parameters:
        X: Train samples. Shape is [N, D]
        y: Ground labels Shape is [N, 1]
        M: Number of neurons in hidden layer.
        iters: Number of iterations for learning
        lr: Learning rate
    Return:
        W_1, W_2: weights between layers
        error_over_time: error over iterations. Shape is [iter, 1]
    """
    W_1 = np.random.rand(M, X.shape[1])
    W_2 = np.random.rand(1, M)
    error_over_time = [0 for i in range(iters)]
    for i, _ in tqdm(enumerate(range(iters))):
        error = 0
        for idx, sample in enumerate(X):
            z, y_preds = self._forward(W_1=W_1, W_2=W_2, X=sample)
            error += (1/2)*np.square(np.subtract(y_preds, y[idx].reshape(1)))
            W_1, W_2 = self._backpropagation(W_1=W_1, W_2=W_2, target=y[idx], net_output=y_preds, z=z, lr=lr, sample=sample)
        error_over_time[i] = error
    return W_1, W_2, error_over_time
```

شکل ها 45: شکل ۴. تابع backward

که خروجی این تابع همانطور گفته شده است شامل وزنهای بین لایه و خطای برای هر تکرار می‌باشد.

## ۲-۲-۱. آزمود شبکه عصبی بر روی یک مجموعه داده

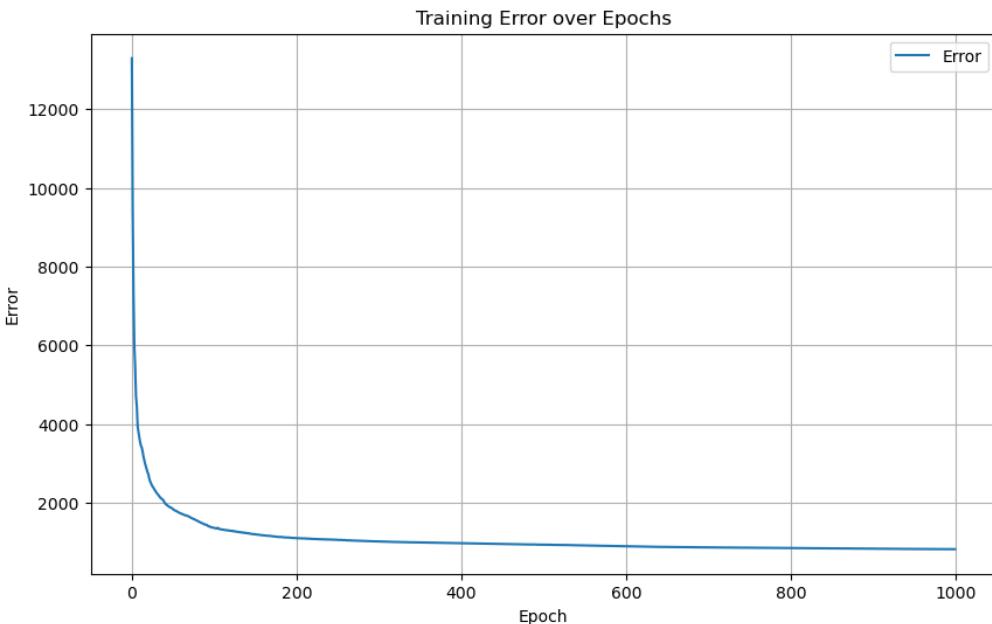
در این قسم بعد از دانلود و بارگذاری داده‌های wine quality از طریق کتابخونه sklearn داده‌ها را به دو مجموعه آموزشی و تست تقسیم کردیم. داده‌ها را نیز به این ترتیب استاندارد سازی کردیم که میانگین و انحراف هر ویژگی مربوط به داده‌های آموزشی را محاسبه کردیم و سپس هم داده‌های آموزشی و هم داده‌های تست را منهای میانگین هر ویژگی و تقسیم بر انحراف معیار کردیم.(به اینکار Z-SCORE تأثیر به سزایی روی عملکرد مدل خواهد گذاشت).

در نهایت یک ویژگی با مقدار ۱ به عنوان ستوز بایاس به دادگاه خود اضافه خواهیم کرد. لایه ها و تعداد تکرارها و ترخ یادگیری قرار گذاشته شده را در جدول زیر ثبت شده است.

جدول ۲. هایپرپارامترهای مدل

Parameters	Values
Iterations	۱۰۰۰
Number of neurons of hidden layer	۳۰
Learning Rate	۰.۰۱

حال که هایپرپارامترها را تنظیم کردیم به سراغ آموزش مدل خواهیم رفت:



شکل ها 46: نمودار خطأ بحسب تكرار

#### نمودار خطأ و تحلیل آن

همانطور که در شکل بالا دیده میشود، خطأ به ازای هر تکرار کمتر میشود که نشاندهندهی درست پیاده سازی شده مدل می باشد.(دقت کنید که معمولا خطأ بعد از هر تکرار معمولا تقسیم بر تعداد نمونه های آموزشی میشود اما به خاطر آنکه در کتاب و صورت سوال گفته نشده از تقسیم کردن پرهیز کردیم).

تحلیل بیشتر: خطأ بعد از تقریبا تکرار ۴۰۰ دیگر کم نمیشود که بدین معنا هست که دیگر وزنهای مدل در نقطه بهینه یا نقطه sub-optimal قرار دارد و دیگر به آن همگرا شده است. این در حالی است که خطأ به صورت خیلی تیز کم میشود که نشاندهندهی یادگیری الگوهای ورودی می باشد.

#### پیش‌بینی داده‌های تست و root mean square error

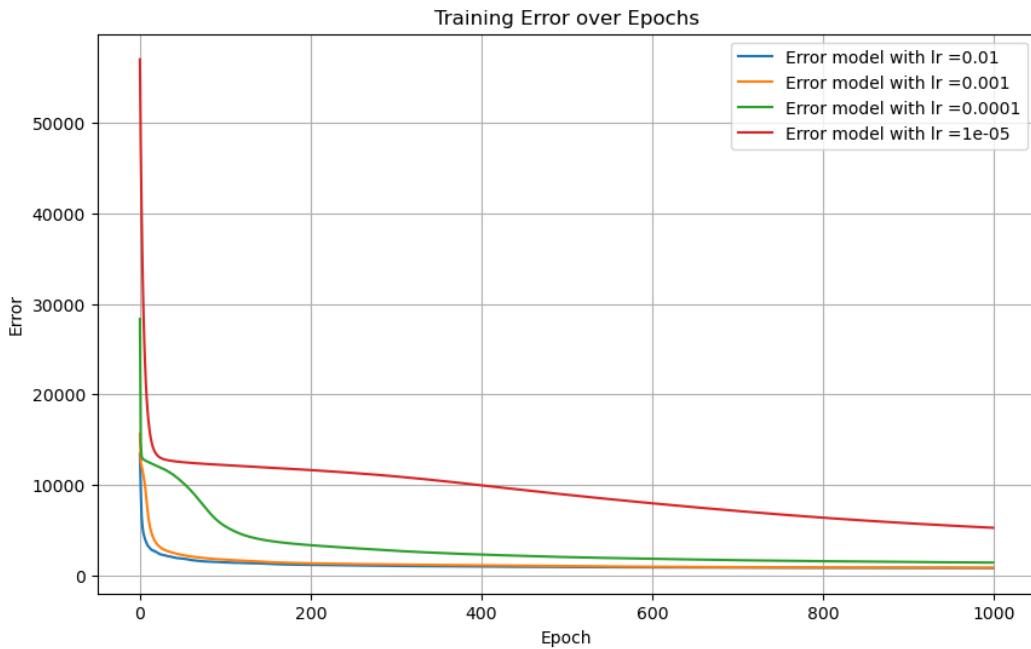
بعد از آموزش شبکه وزنهای شبکه را به دست آوردیم. حال به نوبت استفاده ازتابع فوروارد می‌باشد تا به استفاده از آن ریشه میانگین مربع خطأ را به دست آوریم. این خطأ در شکل زیر گزارش شده است.

root mean square error: 4.098029149338868

این معیار به ما میگوید که تخمین مدل به اندازه‌ی تقریب ۴۰۹ واحد با مقادیر اصلی داده‌های تست فاصله دارد. که به رنج کلاس‌ها نیز بستگی دارد فرض کنید که رنج مربوط به کلاس‌ها از ۰ تا ۱۰۰۰ باشد آنگاه این مقدار برای این معیار نشاندهندهی دقت بسیار عالی مدل می‌باشد اما در اینجا مدل‌ما دارای نهایت ۲۰ عدد می‌باشد فلذًا این حاکی از کارایی بد مدل می‌باشد.(برای بهبود می‌توانیم از regularization term استفاده کنیم یا لایه‌ها را بیشتر کنیم و یا مجموعه آموزشی به جای ۵۰ درصد دادگان را اختصاص دهیم و یا تعداد نورون‌های لایه مخفی را بیشتر کنیم. اما موثرترین کار از نظر ما به اندازه کلاس‌های موجود، نورون در آخرین لایه قرار دهیم و همینطور اگر یک نورون در آخرین لایه قرار می‌گیرد به جای تابع فعالسازی خطی از تابع فعالسازی softmax استفاده کنیم).

سه مقدار برای نرخ یادگیری

برای سه نرخ یادگیری ۱،۰۰۰۱، ۰،۰۰۰۱، ۱e-۵ مدل را آموزش می‌دهیم. در ابتدا خطای هر یک را در کنار یکدیگر در شکل زیر نمایش می‌دهیم.



شکل ها 47: نمودار خطای برای هر نرخ یادگیری

تحلیل: همانطور که می‌بینیم دو نرخ خطای  $1 \times 10^{-5}$  و  $1 \times 10^{-4}$  تقریباً با یک سرعت به خطای تقریباً یکسانی می‌رسند. این در حالی است که نرخ یادگیری  $1 \times 10^{-4}$  با سرعت کمتری به همان میزان خطای خواهد رسید. اما برای نرخ یادگیری  $1 \times 10^{-5}$  خطای با سرعت خیلی کمتری به خطای بیشتری با همان تعداد تکرار می‌رسد که نیاز به تعداد تکرار بیشتری دارد تا خطای خود را به خطای دیگر نرخ یادگیری‌ها برساند چرا که از طریق مدل پس انتشار داده می‌شود با ضریب کمتری در وزنها تاثیر می‌گذارد فلذا به تعداد تکرار بیشتری لازم دارد تا به همان خطای مینیمم برسد.

جدول زیر نیز حاکی از عملکرد مدل برای معیار خطای RMSE می‌باشد.

جدول ۳. معیار RMSE برای نرخهای یادگیری مختلف

Learning Rate	RMSE
$1 \times 10^{-1}$	4.06
$1 \times 10^{-2}$	3.9433.943
$1 \times 10^{-3}$	2.978
$1 \times 10^{-5}$	3.96

با اینکه نمودار خطای برای نرخ یادگیری  $1 \times 10^{-1}$  تقریباً یکی می‌باشد، اما معیار RMSE که برای خطای تست درنظر گرفتیم نشاندهنده‌ی آذ است که مدل با نرخ یادگیری  $1 \times 10^{-1}$  دقت بهتری دارد.

برای نرخ یادگیری  $1 \times 10^{-1}$  با اینکه خطای آخرین تکرار بالاتر هست، اما معیار RMSE این مدل بسیار بهتر از سایرین می‌باشد که می‌تواند حاکی از این باشد که مدل ما با نرخهای یادگیری  $1 \times 10^{-1}$  احتمال overfit شدن روی داده‌های آموزشی وجود دارد. در حالت  $1 \times 10^{-5}$  نیز بخاطر آنکه تکرار کمی برای این نرخ یادگیری می‌باشد، بنابراین به نقطه بهینه یا suboptimum نرسیده است بنابراین RMSE بالاتری دارد.

تحلیل کلی از این دو بدین صورت خواهد بود که برای نرخ یادگیری  $1 \times 10^{-1}$  تعداد خوبی بین سرعت و معیار RMSE دارد و در عین حال نیز از overfitting کمتری نسبت به نرخ یادگیری بزرگتر دارد و بنابراین این نرخ بسیار خوبی می‌باشد.

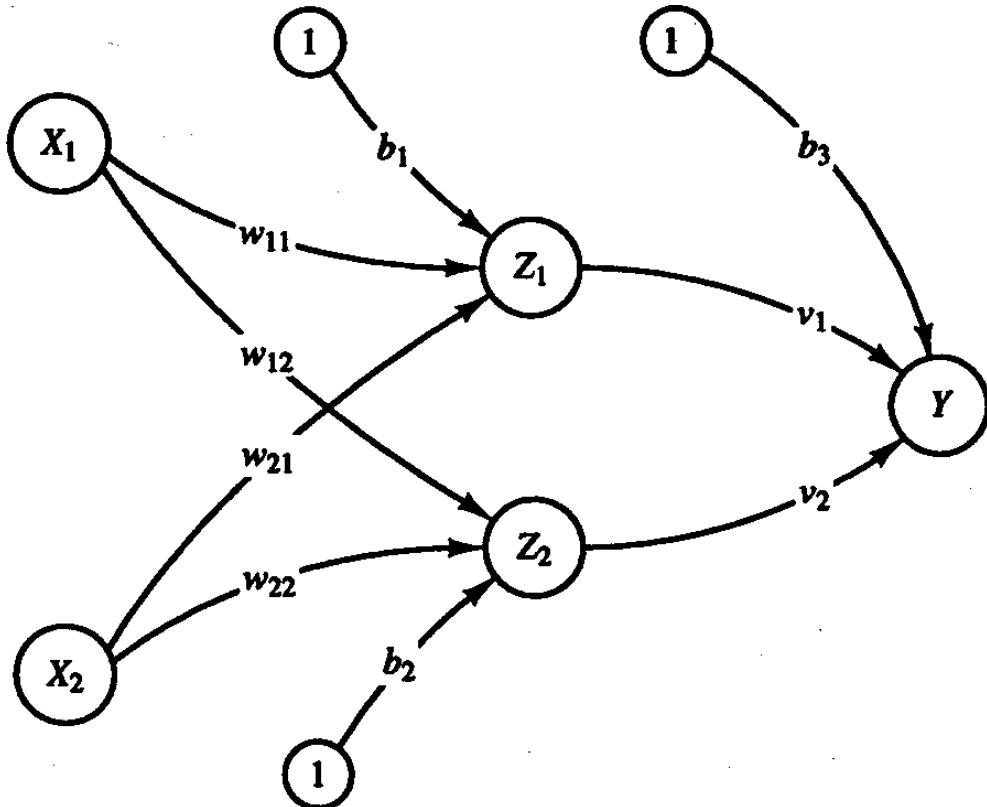
در شبکه‌های عصبی دیگر انتخاب بهینه نه تنها نرخ یادگیری بلکه دیگر هایپرپارامترها نیز حائز اهمیت می‌باشند. تحلیلی که روی مدل خود داشتیم مبنایی است برای انتخاب مدل بهینه چرا که هم براساس خطأ و هم براساس معیار دیگری که ببروی دادگاه تست می‌باشد بهترین مدل را انتخاب کردیم. درواقع از کشیدن نمودار خطأ و کشیدن نمودار دقیق روی تست توانستیم بهترین مدل را که احتمالاً کمتری برای overfit شدن را انتخاب کردیم. البته که چون مدل ثابت می‌باشد نمیتوان خطای بایاس را در نظر گرفت و به همین علت تنها از overfitting می‌توان صحبت نمود.

## پرسش - ۳ Madaline

. الگوریتم‌های MRI و MRII .

در این قسمت از تمرین هر دو الگوریتم MRI و MII را توضیح خواهیم داد و سپس یکی را انتخاب کرده و به پیاده‌سازی آن می‌پردازیم. توجه داشته باشید که پیاده‌سازی را در قسمت ۳ توضیح خواهیم داد.

قبل از آنکه وارد این دو الگوریتم شویم می‌دانیم، که شبکه Adaline حاصل چندین Madaline در یک لایه مخفی که درنهایت توسط یک AND یا OR گیت نوروز M&P ترکیب می‌شوند. حال به عنوان نمونه در شکل زیر که از کتاب مرجع آمده است نمونه‌ای از شبکه Madaline می‌باشد که دارای دو نوروز Madaline در لایه مخفی می‌باشد.



شکل ها 48: نمونه‌ای از کتاب مرجع برای شبکه Madaline

حال به سراغ الگوریتم MRI می‌رویم و آنرا توضیح خواهیم داد.

### MRI-۳. الگوریتم

در این الگوریتم که نسخه اصلی برای آموزش Madaline توسط Widrow و Hoff هست، وزنهای لایه مخفی را تغییر خواهیم داد و وزنهای خروجی ثابت هستند. حال این الگوریتم را اگر برای شبکه نمونه در شکل بالا اعمال کنیم. خروجی‌های  $v_1, v_2, b_3$  را که وزنهای و بایاس لایه خروجی هستند را همه برابر  $0.5$  درنظر می‌گیریم. می‌دانیم تابع فعالسازی برای هر نوروز برابر است با:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

حال که وزنهای لایه خروجی ثابت هستند، در این الگوریتم لازم است که وزنهای لایه مخفی را با مقادیر کوچک مقداردهی اولیه انجام دهیم حال فرض کنید که تعداد نورونهای لایه میانی  $n$  باشد آنگاه وزنهای لایه میانی را به صورت یک ماتریس با ابعاد  $[2, n]$  در نظر بگیریم و بایاس این لایه نیز به صورت یه بردار خواهد بود با اندازه  $n$  چرا که هر نورون دارای بایاس اولیه خود می‌باشد. که در معادله زیر بهتر دیده می‌شود:

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ w_{21} & \cdots & w_{2n} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

دقت کنید که وزنهای لایه خروجی را ثابت درنظر می‌گیریم و همه وزنهای  $1$  و بایاس برابر تعداد نورونهای لایه مخفی منتهای یک خواهد بود.(چرا که به صورت AND گیت عمل می‌کند)

حال الگوریتم MRI به صورت زیر عمل می‌کند:

1. وزنهای لایه مخفی را مقداردهی اولیه می‌کنیم.
2. برای هر جفت از داده‌های آموزشی کار زیر را تکرار خواهیم کرد:
  - a. ورودی را گرفته و به عنوان فعالساز به لایه ورودی می‌دهیم.
  - b. برای هر نورون لایه میانی ورودی آن را به صورت زیر حساب می‌کنیم:

$$Z_{in} = W^T \cdot X + b$$

- c. سپس از تابع فعالسازی، خروجی لایه مخفی را به دست می‌آوریم.

$$Z = f(Z_{in})$$

- d. اینکار را برای خروجی نیز انجام می‌دهیم تا خروجی مدل را به دست آوریم به این طریق:

$$y_{in} = [1 \quad \cdots \quad 1]^T \cdot Z + n - 1$$

$$y = f(y_{in})$$

- e. حال خطرا را محاسبه می‌کنیم بدین صورت که اگر خروجی مدل با کلاس آن نمونه مطابقت داشت هیچ وزنی آپدیت نمی‌شود اما اگر خروجی مدل با کلاس نمونه برابر نبود یکی از دو حالت زیر اتفاق می‌افتد:

- i. اگر نمونه به کلاس ۱ تعلق داشت، یا می‌توانیم وزن مربوط به بزرگترین ورودی نوروز (در نمونه آموزشی این تمرین در تلگرام بدین صورت آپدیت می‌شود) و یا نزدیک‌ترین ورودی نوروز به صفر را به صورت زیر آپدیت کنیم (مثل کتاب):

$$b_j(new) = b_j(old) + \alpha(1 - z_{in_j})$$

$$w_{ij}(new) = w_{ij}(old) + \alpha(1 - z_{in_j})x_i$$

- ii. ولی اگر نمونه به کلاس ۱ تعلق داشت، تمام وزنهای نورونهایی که ورودی مثبت دارند را به صورت زیر آپدیت می‌کنیم:

$$b_k(new) = b_k(new) + \alpha(-1 - z_{in_k})$$

$$w_{ik}(new) = w_{ik}(old) + \alpha(-1 - z_{in_k})x_i$$

- f. حالا اگر وزنها دیگر تغییر نکردند و یا به تغییر وزنها خیلی کم، کم می‌شوند و یا به تعداد تکرار مدنظر رسیدیم، آنگاه این فرآیند را دیگر تکرار خواهیم کرد.

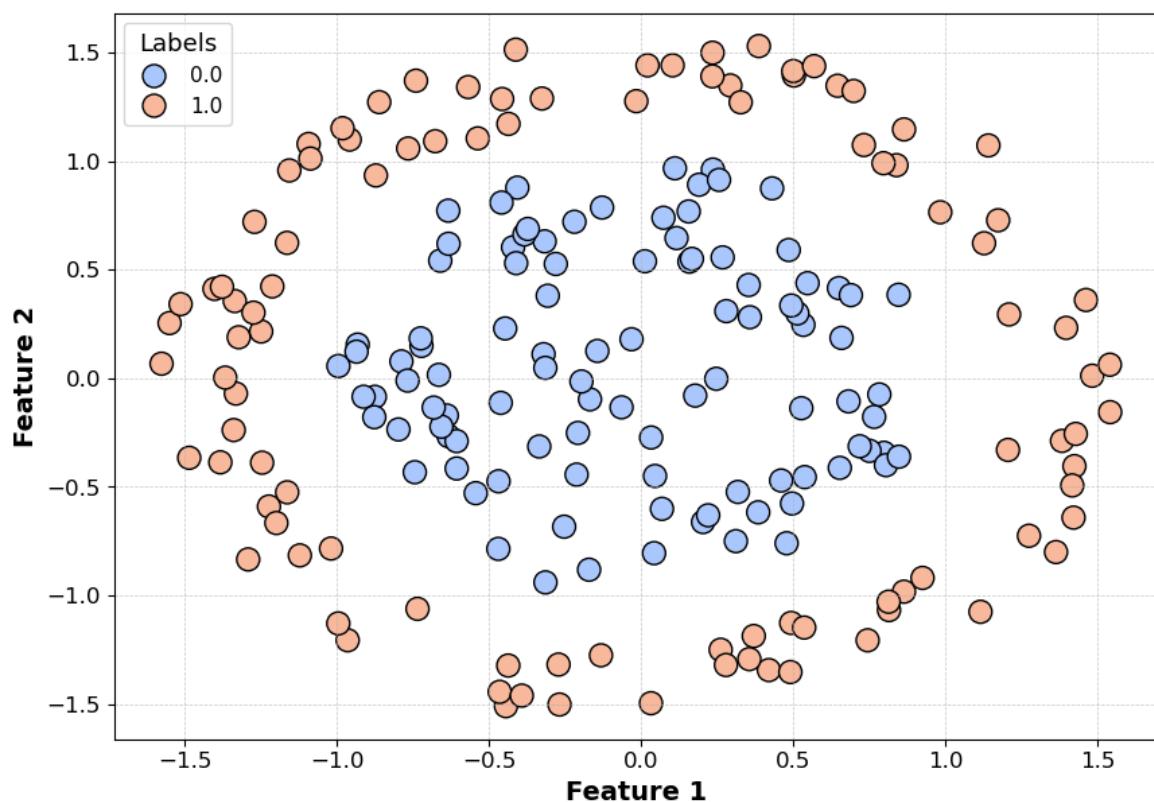
### ۳-۱-۲. الگوریتم MRII

در الگوریتم MRII یک تفاوتی که با الگوریتم MRI دارد آزاد است که نه تنها وزنهای لایه میانی آپدیت می‌شوند بلکه وزنهای لایه آخر نیز آپدیت می‌شوند. تفاوت دیگر آزاد است که زمان محاسبه خطأ، وزنها به صورت دیگر آپدیت خواهند شد. بدین صورت که اگر خروجی مدل را کلاس نمونه برابر نبود آنگاه برای هر نورونی که ورودی آذخیلی به صفر نزدیک بود مثلا در بازه منفی  $-0.25$  و مثبت  $0.25$  بود، خروجی آذ نورونها را اگر یک بودند منفی یک و بالعکس تغییر می‌داد و در مرحله بعد با این تغییرات خروجی مدل را محاسبه میکرد و اگر خطأ کم میشد آنگاه وزنها را به روزرسانی می‌کرد.

### ۳-۲. نمودار پراکندگی داده‌ها

در این قسمت با استفاده از کتابخوانه `pandas` مجموعه دادگان را لود کرده و سپس به کمک کتابخانه `matplotlib` و `seaborn` نمودار پراکندگی داده‌ها را به دست خواهیم آورد. نمودار پراکندگی داده‌ها را در شکل زیر خواهیم دید. همانطور که از شکل بالا می‌بینیم داده‌ها دارای دو کلاس می‌باشند که به صورت دایروی از یکدیگر قابل جداسازی می‌باشند. داده‌های با کلاس صفر دارای رنگ آبی و داده‌های با کلاس یک دارای رنگ نارنجی می‌باشند.

**Scatter Plot of Feature 1 vs Feature 2 by Label**



شکل ها 49: شکل ۸. نمودار پراکندگی داده‌ها

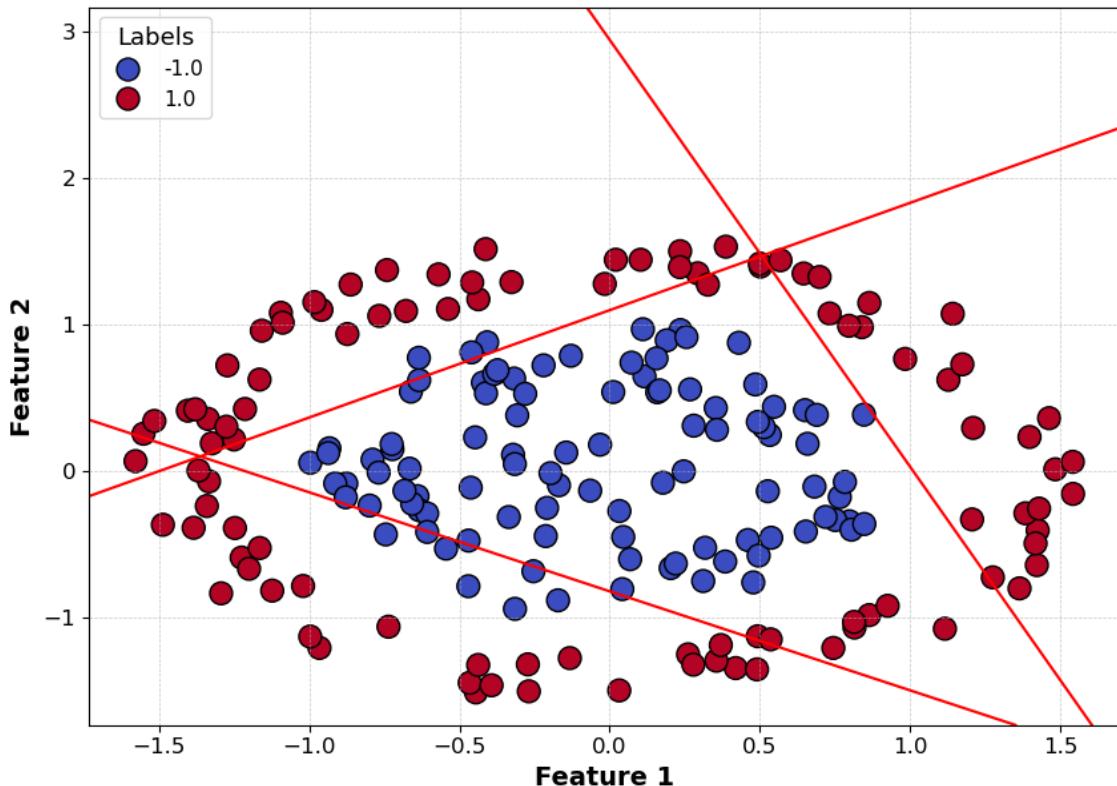
### ۳-۳. آموزش مدل

قبل از انکه مدل پیاده سازی شده را پیاده سازی کنیم، کدها به صورت object oriented زده شده اند اما چون در داخل هر متود توضیحات هر کدام را قرار داده ایم از توضیح کد صرف نظر خواهیم کرد. دقت کنید برای درست سنجی نیز مثلاً کتاب را پیاده کردیم تا از درستی مدل خود اطمینان حاصل کنیم. ما الگوریتم MRI را پیاده سازی کردیم. دقت فرمایید

که ما داده‌ها را به دودسته آموزشی و تست تقسیم کردیم که ۷۵ درصد برای دادگان آموزشی و ۲۵ درصد نیز برای دادگان تست. همین طور هم برای سادگی نمونه‌هایی که متعلق به کلاس صفر بودند را کلاس منفی یک درنظر گرفتیم.

۱-۳-۳. مدل با سه نوروز  
با قرار دادن نورونهای لایه مخفی به سه، تعداد تکرار برابر ۳۸۸۳ می‌باشد و نمودار خط‌های جداشونده در شکل زیر نشان داده شده است.

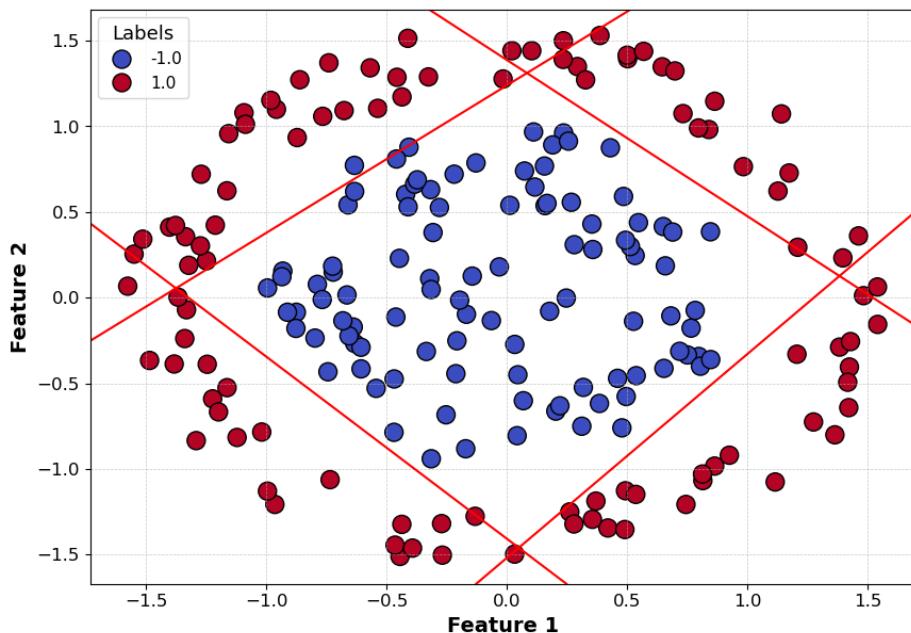
**Scatter Plot of Feature 1 vs Feature 2 by Label**



شکل ها ۵۰: شکل ۹. نمودار پراکندگی داده‌ها که توسط ۳ خط جداسهادند

به ازای هر نورونی که اضافه می‌شود یه مرز خطی جدا کننده اضافه می‌کند. دقت برای این مدل دارای ۳ نوروز برابر است با ۸۶ درصد.

**Scatter Plot of Feature 1 vs Feature 2 by Label**



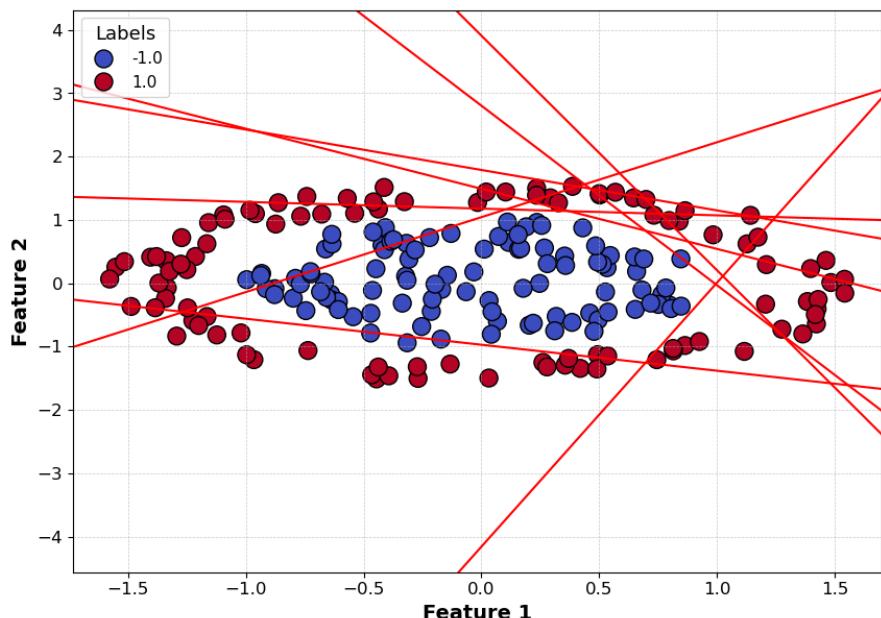
شکل ها ۵۱: ۱۰. نمودار پراکندگی دادهها به همراه چهار خط جداساز با مدل چهار نورونی

این مدل نیاز به ۱۷ تا تکرار داشت که به دقت ۹۸ درصد رسید. خطهای جداکننده در این حالت را در شکل زیر نمایش داده شده است. این مدل با چهار خط توانسته مرز بهتری نسبت به مدل قبلی داشته باشد.

### ۳-۳-۳. مدل با هشت نوروز

این مدل ۸ نورونی نیاز به ۲۲ تا تکرار داشته است و دقت این مدل ۹۲ درصد می‌باشد. نمودار پراکندگی دادهها به همراه این چهار خط در شکل زیر آورده شده است.

**Scatter Plot of Feature 1 vs Feature 2 by Label**



شکل ها ۵۲: ۱۱. نمودار پراکندگی دادهها به همراه هشت خط جداساز با مدل هشت نورونی

#### ۴-۳. تحلیل نتایج

در مدل سه نورونی دقت ۸۶ درصد بود و در ۳۸۸۸۳ تکرار مدل همگرا می شود. از روی نمودار پراکنده‌گی داده‌های به دست آمده از قسمت قبل می‌توان نتیجه گرفت که مدل نمی‌تواند با سه خط مجزا (سه نورون) به خوبی داده‌ها را از یکدیگر تمایز کند. یعنی با سه نورون توافقی ایجاد مربذنده مناسب را ندارد. چرا که نمیتوان انتظار ایجاد مربز پیچیده از این سه خط را داشته باشیم. بنابراین مدل دارای خطای بایاس می‌باشد چرا که مدل ساده‌تری نسبت به مدل‌های با نوروز بیشتر هست. (توجه داشته باشید که هر چقدر پیچیدگی مدل بیشتر شود خطای واریانس بیشتر شده و خطای بایاس کمتر خواهد بود و اگر پیچیدگی مدل کمتر شود خطای بایاس اما خطای واریانس کمتر خواهد بود و همینطور اگر مدل پیچیده تر شود به overfitting نیزندیکتر خواهیم شد و همینطور اگر مدل ساده‌تر باشد به Underfit خواهیم شد).

بخاطر آنکه مدل دارای خطای بایاس در این حالت می‌باشد و اینکه با مدل ساده سه نورونی نمیتوان انتظار داشت که این مدل بتواند مربز خوبی بین دو کلاس قرار دهد اما مدل سعی دارد بهترین سه خط را برای جداگردن این دو کلاس پیدا کند به همین علت تعداد تکرار بیشتری را خواهد داشت تا به بهترین سه خط جداساز همگرا شود.

در مدل چهار نورونی با ۱۷ تکرار به دقت ۹۸ درصد رسیده‌ایم که همانطور که در شکل پراکنده‌گی داده‌ها برای این مدل دیدیم این چهار خط توانسته است که به خوبی مربز خوبی را تصور شود. این حاکی از آن است که این مدل در مقایسه با مدل سه نورونی دارای پیچیدگی بیشتری می‌باشد و در مقایسه با مدل هشت نورونی مدلی ساده‌تری می‌باشد. این مدل مصالحه خوبی بین بایاس و واریانس می‌باشد.

با اضافه کردن تنها یک نورون می‌توان مربز بندی پیچیده تری را ترسیم کرد و بهتر می‌توان داده‌ها را جدا نمود فلذا تعداد تکرار کمتری نیاز خواهیم داشت تا همگرا شویم. این تکرار کم حاکی از آن است که مدل می‌تواند راحت‌تر دو کلاس را از هم جدا نماید.

در مدل هشت نورونی، دقت ۹۲ درصد شده و در ۲۲ تکرار همگرا شده‌است. همانطور که در نمودار پراکنده‌گی داده‌ها در قسمت قبل دیدیم، این مدل با ۸ خط جداساز سعی بر متمایز کردن داده‌های هر کلاس می‌باشد اما به خاطر پیچیدگی زیاد این مدل خطای حاصل از واریانس زیاد می‌باشد فلذا دقت آن به ۹۲ درصد رسیده است که نسبت به دقت مدل چهار نورونی کمتر می‌باشد. (به خاطر خطای واریانس تعداد اپیاک‌ها و تکرارها و اینکه وزنها بیشتر از حالت چهار نورونی می‌باشند کمی بیشتر از مدل قبلی می‌باشد. و همینطور هم چون پیچیدگی مدل زیاد شده است بنابراین دور از ذهن نیست که مدل نیز نیاز به تکرار بیشتری نسبت به مدل چهار نورونی داشته باشد تا مربذنده‌ی پیچیده تری را به دست آورد)

اما یک تحلیل جالت تر آن است که می‌دانیم طبق نمودار پراکنده‌گی داده‌ها این دو کلاس با یک دایره و یا یک بیضی می‌توانند از یکدیگر تمایز شوند. هر چقدر که تعداد نورونهای لایه مخفی را زیادتر می‌کنیم مدل سعی دارد که خطهای جداساز خود را منطبق بر این بیضی تشکیل دهد. در این حالت خاص داده‌ها که می‌توانند به صورت بیضی از یکدیگر تمایز شوند، اگر تعداد نورونهای لایه مخفی را به بینهایت میل بدھیم می‌توانیم جدا کننده‌ی بیضی را ببینیم.

## ۱-۴. نمایش تعداد ستون

با خواندن ماتریس با کمک pandas و نشان دادن تعداد nan ها داریم:

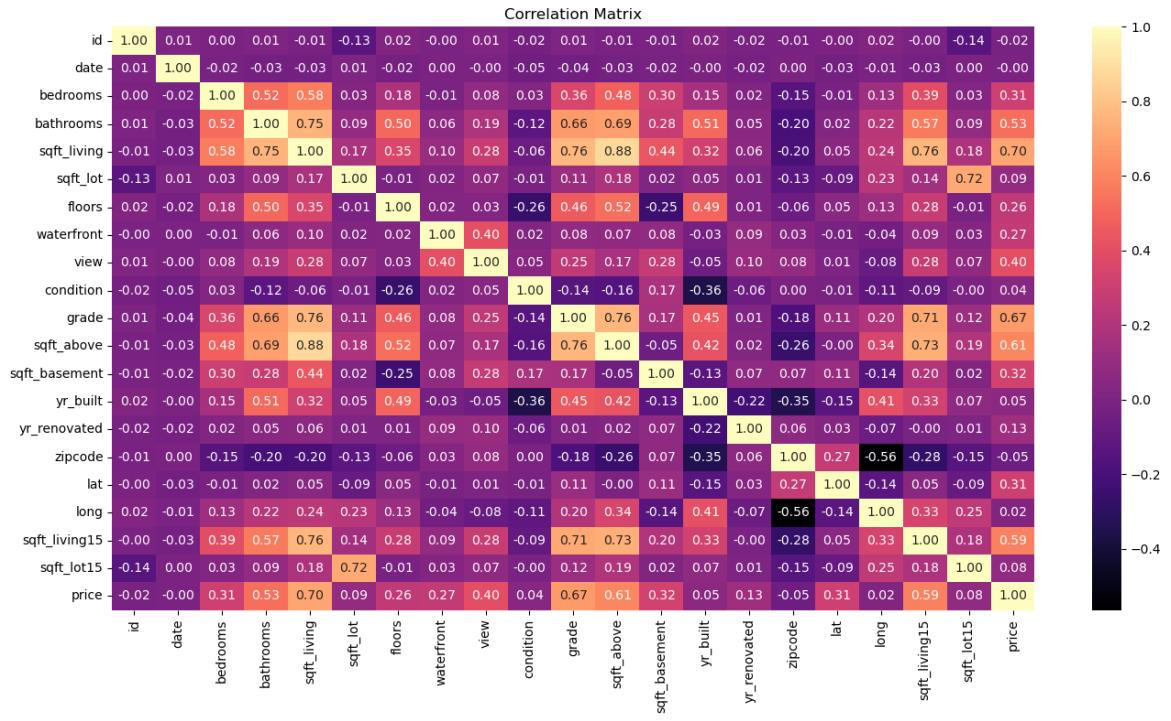
```
..   id          0
date        0
bedrooms    0
bathrooms   0
sqft_living 0
sqft_lot     0
floors      0
waterfront   0
view        0
condition    0
grade        0
sqft_above   0
sqft_basement 0
yr_built     0
yr_renovated 0
zipcode      0
lat          0
long         0
sqft_living15 0
sqft_lot15   0
price        0
dtype: int64
```

شکل ها ۵۳: شکل ۱.۴: ستون ها و تعداد nan ها در داده

همانطور که دیده می شود، nan ای در داده ها وجود ندارد. تعداد کل داده ها نیز ۲۱۶۱۳ است.

## ۴-۲. ماتریس همبستگی

با رسم ماتریس همبستگی برای داده ها، شکل ۲.۴ را داریم:



شکل ها ۵۴: شکل ۲.۴- ماتریس همبستگی داده ها

در این ماتریس، هرچقدر رنگ به سمت روشن تری داشته، میزان کورولیشن آنها بیشتر می شود. برای مثال `sqft_living` و `sqft_above` با هم همبستگی بالا و برابر با  $0.88$  دارند. هرچقدر قدر مطلق عدد کورولیشن به عدد نزدیک تر باشد، همبستگی بالاتر و هر چقدر به صفر نزدیکتر باشد، همبستگی کمتر است. واضح است که `yr_built` با خودش بالاترین همبستگی را دارد. در اینجا، ماتریس همبستگی، نسبت کوواریانس بین دو ویژگی  $i$  و  $j$  را با فرمول زیر مشخص می کند:

$$\begin{aligned} C_{ij} &= \text{Cov}(x_i, x_j) = E \left\{ (x_i - \bar{x}_i) \times (x_j - \bar{x}_j) \right\} \\ &= \frac{\sum_{i=1}^{N_i} (x_i - \bar{x}_i) \times (x_j - \bar{x}_j)}{N_i} \\ R_{ij} &= \frac{C_{ij}}{\sqrt{C_{ii} \times C_{jj}}} \end{aligned}$$

برای بدست آوردن بیشترین همبستگی ها با `price`، ستون آنرا با مقادیر و نام دیگر ستون ها می توان چاپ کرد:

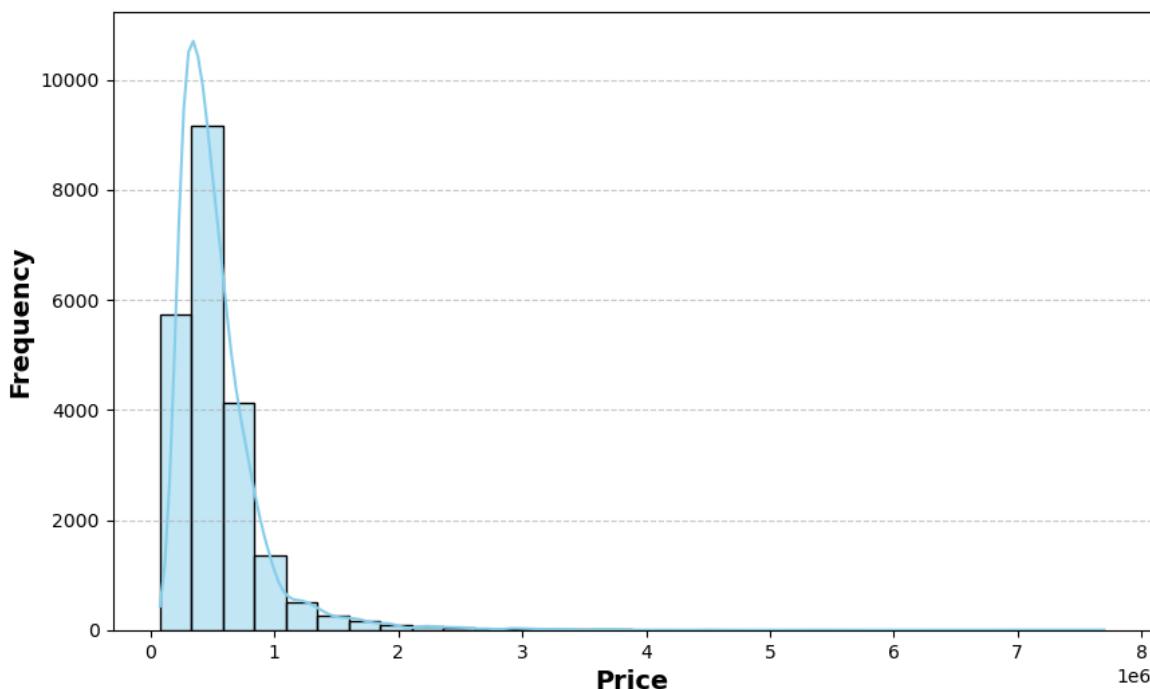
<code>id</code>	<code>-0.016762</code>
<code>date</code>	<code>-0.004357</code>
<code>bedrooms</code>	<code>0.308350</code>
<code>bathrooms</code>	<code>0.525138</code>
<code>sqft_living</code>	<code>0.702035</code>
<code>sqft_lot</code>	<code>0.089661</code>
<code>floors</code>	<code>0.256794</code>
<code>waterfront</code>	<code>0.266369</code>
<code>view</code>	<code>0.397293</code>
<code>condition</code>	<code>0.036362</code>
<code>grade</code>	<code>0.667434</code>
<code>sqft_above</code>	<code>0.605567</code>
<code>sqft_basement</code>	<code>0.323816</code>
<code>yr_built</code>	<code>0.054012</code>
<code>yr_renovated</code>	<code>0.126434</code>
<code>zipcode</code>	<code>-0.053203</code>
<code>lat</code>	<code>0.307003</code>
<code>long</code>	<code>0.021626</code>
<code>sqft_living15</code>	<code>0.585379</code>
<code>sqft_lot15</code>	<code>0.082447</code>
<code>price</code>	<code>1.000000</code>
<code>Name:</code>	<code>price, dtype: float64</code>

شکل ها ۵۵: شکل ۳.۴- همبستگی ها با `price`

بالاترین همبستگی ها با `price` به ترتیب با `sqft_above`, سپس با `grade`, سپس با `sqft_living` هستند با مقدار همبستگی به ترتیب  $0.667$  و  $0.605$  و  $0.585$ . به طور معمول همبستگی زیر  $0.6$  را می تواند به صورت همبستگی ضعیف در نظر گرفت. به این معنی که تغییرات یک ویژگی با تغییرات ویژگی دیگر به شکلی منظم تر و پیش بینی پذیرتر همراه نیست.

۴-۳. رسم نمودار  
نمودار توزیع قیمت به صورت زیر است:

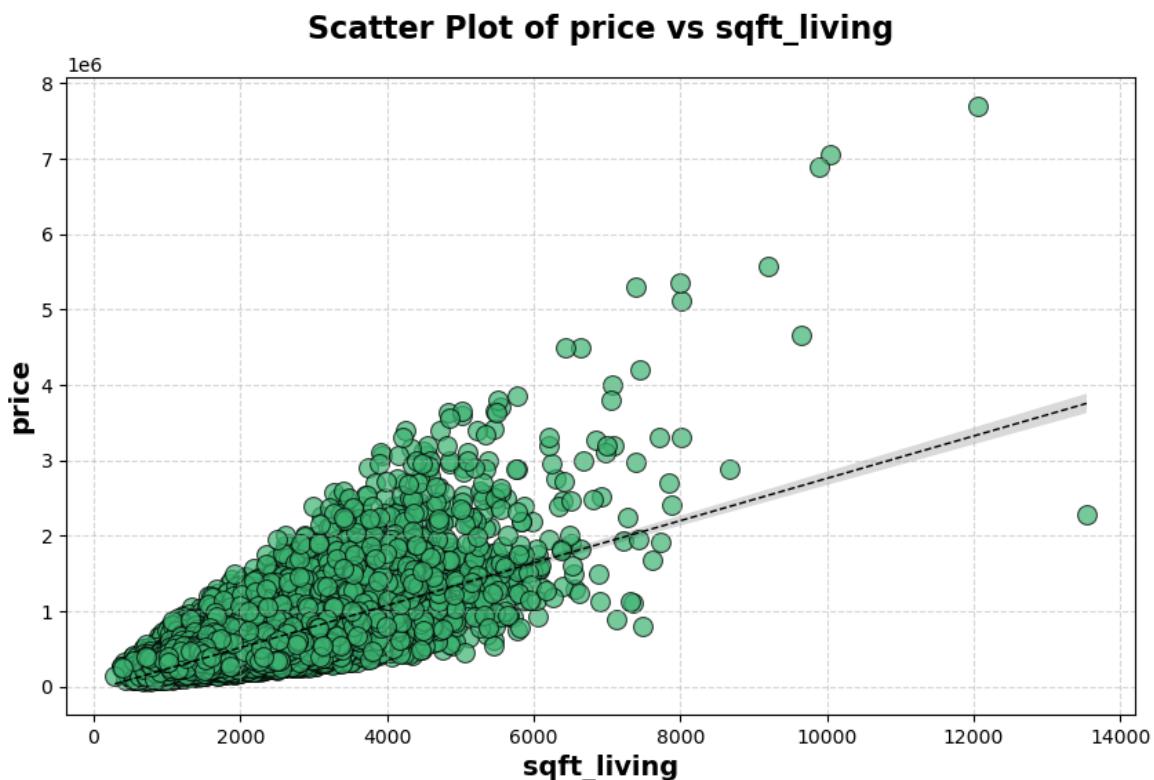
### Distribution of Price



شکل ها ۵۶: شکل ۴.۴- نمودار توزیع قیمت

همانطور که مشاهده می شود، توزیع قیمت میانگین برابر  $5400.88.1417665294$  بوده و انحراف معیار آن برابر با  $367127.1964826997$  است. البته این توزیع بیشتر به شکل پوآسون نزدیک تر بوده تا نرمال و دلیل آن می تواند انتخاب  $K$  نمونه از کل خانه ها در هنگام نمونه برداری باشد. (فرآیند اتفاقی نقطه ای)

نمودار ارتباط بالاترین correlation با price به صورت زیر است:



شكل ۵.۴: ارتباط بالاترین correlation با price

بر اساس این شکل، دیده می شود که  $price$  و  $sqft\_living$  تقریباً به صورت خطی با هم ارتباط دارند و اضافه شدن  $linear\ regression$  به طور مستقیم با زیاد شدن قیمت ارتباط دارد، البته اینجا دیده می شود که خط کشیده شده بر روی داده ها نشان می دهد لزوماً این ارتباط کاملاً خطی نبوده و فقط حدوداً ۷۰ درصد از اثر ( $r = 0.7$ ) از ارتباط خطی آنها می آید.

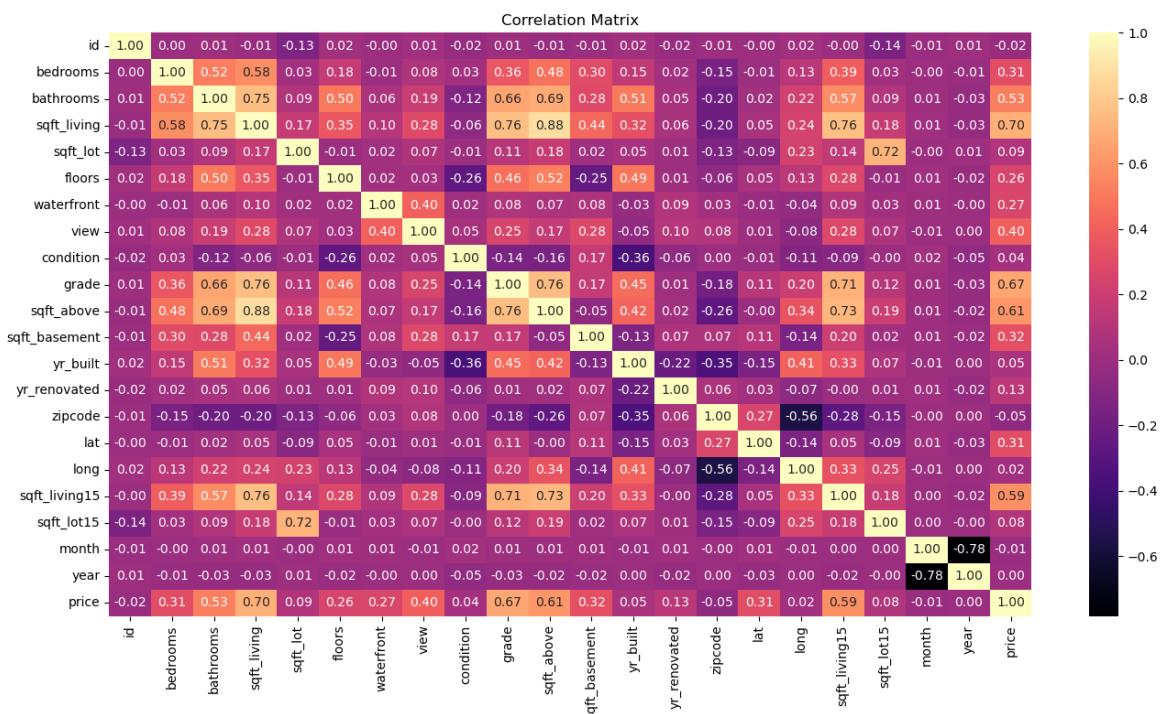
#### ۴-۴. پیش پردازش داده

با استفاده از خود pandas و تبدیل تاریخ و نشان داده ۳ ردیف آخر داریم:

	month	year	price
0	10	2014	221900.0
1	12	2014	538000.0
2	2	2015	180000.0
3	12	2014	604000.0
4	2	2015	510000.0

شکل ها ۵۸: شکل ۶.۴- تبدیل تاریخ به قیمت و سال

ماتریس Correlation جدید به شکل زیر است:



شکل ها ۵۹: شکل ۷.۴- ماتریس Correlation جدید

همانطور که دیده می شود، ارتباط بین year و month و دیگر ویژگی ها نیز دیده می شود که در ماتریس قبلی مشهود نبود.

تقسیم کردن و دادن ۲۵ درصد به تست و ۷۵ درصد به ترین، دو dataframe زیر را حاصل می کند:

	<b>id</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>condition</b>	<b>grade</b>	...	<b>yr_built</b>	<b>yr_renovated</b>
735	2591820310	4	2.25	2070	8893	2.0	0	0	4	8	...	1986	0
2830	7974200820	5	3.00	2900	6730	1.0	0	0	5	8	...	1977	0
4106	7701450110	4	2.50	3770	10893	2.0	0	2	3	11	...	1997	0
16218	9522300010	3	3.50	4560	14608	2.0	0	2	3	12	...	1990	0
19964	9510861140	3	2.50	2550	5376	2.0	0	0	3	9	...	2004	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4267	2621069066	3	2.00	3190	207346	2.0	0	0	3	9	...	1994	0
9158	6021501635	4	2.50	2560	4000	2.0	0	0	5	8	...	1929	0
15589	1117000050	3	2.25	1900	9990	1.0	0	0	3	7	...	1961	0
463	4166600473	4	2.25	2390	11250	2.0	0	0	3	9	...	1988	0
1754	5608000700	3	2.50	4570	10615	2.0	0	0	3	12	...	1991	0

16210 rows × 22 columns

شکل ها ۶۰: شکل -۸.۴ داده های آموزش

تعداد داده های آموزش برابر با ۱۶۲۱۰ است، که تقسیم بر تعداد کل داده ها که ۲۱۶۱۳ است برابر است با ۰.۷۵۰۰۱۱ که همان ۷۵ درصد کل داده ها است.

	<b>id</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>condition</b>	<b>grade</b>	...	<b>yr_built</b>	<b>yr_renovated</b>
9	3793500160	3	2.50	1890	6560	2.0	0	0	3	7	...	2003	0
11	9212900260	2	1.00	1160	6000	1.0	0	0	4	7	...	1942	0
13	6054650070	3	1.75	1370	9680	1.0	0	0	4	7	...	1977	0
16	1875500060	3	2.00	1890	14040	2.0	0	0	3	7	...	1994	0
24	3814700200	3	2.25	2450	6500	2.0	0	0	4	8	...	1985	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
21597	191100405	4	3.25	3410	10125	2.0	0	0	3	10	...	2007	0
21598	8956200760	4	2.50	3118	7866	2.0	0	2	3	9	...	2014	0
21604	9834201367	3	2.00	1490	1126	3.0	0	0	3	8	...	2014	0
21610	1523300141	2	0.75	1020	1350	2.0	0	0	3	7	...	2009	0
21612	1523300157	2	0.75	1020	1076	2.0	0	0	3	7	...	2008	0

5403 rows × 22 columns

شکل ها ۶۱: شکل -۹.۴ داده های تست

تعداد داده های تست برابر با ۵۴۰۳ است که تقسیم بر کل داده ها ۲۵ درصد می شود.

داده ها می شوند با کمک `scale` کتابخانه `MinMaxScalar` در `sikit-learn documentation` به صورت زیر است:

$$X_{std} = (X - X.\min(axis=0)) / (X.\max(axis=0) - X.\min(axis=0))$$

$$X_{scaled} = X_{std} * (max - min) + min$$

به این معنا که به صورت خطی مقادیر را بین ۱ و ۰ با شیب خط واصل `min` و `max` بیاوریم.

به این صورت، `mean` برای داده های آموزش برابر است با:

<b>id</b>	<b>0.461674</b>
<b>bedrooms</b>	<b>0.102193</b>
<b>bathrooms</b>	<b>0.264009</b>
<b>sqft_living</b>	<b>0.135181</b>
<b>sqft_lot</b>	<b>0.008847</b>
<b>floors</b>	<b>0.196249</b>
<b>waterfront</b>	<b>0.008143</b>
<b>view</b>	<b>0.059809</b>
<b>condition</b>	<b>0.601928</b>
<b>grade</b>	<b>0.554647</b>
<b>sqft_above</b>	<b>0.164364</b>
<b>sqft_basement</b>	<b>0.060611</b>
<b>yr_built</b>	<b>0.615328</b>
<b>yr_renovated</b>	<b>0.041244</b>
<b>zipcode</b>	<b>0.389751</b>
<b>lat</b>	<b>0.650136</b>
<b>long</b>	<b>0.253194</b>
<b>sqft_living15</b>	<b>0.273509</b>
<b>sqft_lot15</b>	<b>0.014243</b>
<b>month</b>	<b>0.506522</b>
<b>year</b>	<b>0.323257</b>
<b>price</b>	<b>0.061144</b>
<b>dtype:</b>	<b>float64</b>

شکل ها ۶۲: شکل ۱۰.۴ - میانگین آموزش بعد از **scale** شدن

۴-۵. پیاده سازی مدل  
مدل اولاً با یک لایه به شکل زیر طراحی می شود:

```

4 class MLP_One(tf.keras.Model):
5
6     def __init__(self):
7         super(MLP_One, self).__init__()
8         self.dense = keras.layers.Dense(2048 , activation='relu' , kernel_regularizer=keras.regularizers.L1L2(l1 =
9             self.flatten= keras.layers.Flatten()
10            self.drop = keras.layers.Dropout(0.2)
11            #self.bath_norm = keras.layers.BatchNormalization()
12            self.output_layer = keras.layers.Dense(1 , activation='linear' )
13
14
15    def call(self, inputs):
16        x = self.dense(inputs)
17        x = self.drop(x)
18        #x = self.bath_norm(x)
19        x = self.flatten(x)
20        x = self.output_layer(x)
21        return x

```

شکل ها ۶۳: شکل ۱۱.۴ - شبکه تک لایه

این شبکه دارای ۲۰۴۸ نوروز مخفی بوده و در لایه خروجی، یک عدد که قیمت اسکیل شده است را خروجی می دهد.  
همچنین لایه مخفی شامل regularizer L1L2 است تا وزن ها و توان ۲ وزن ها هردو کوچک شوند. در لایه مخفی نیز از Activation Function ReLU تیپیکالا در MLP ها، بهره می بریم.  
برای شبکه دو لایه داریم:

```

class MLP_two(tf.keras.Model):

    def __init__(self):
        super(MLP_two, self).__init__()
        self.dense_1 = keras.layers.Dense(256 , activation='relu' , kernel_regularizer=keras.regularizers.L1L2(l1 =
        self.dense_2 = keras.layers.Dense(512 , activation='relu' )
        self.flatten= keras.layers.Flatten()
        self.drop = keras.layers.Dropout(0.15)
#self.bath_norm = keras.layers.BatchNormalization()
        self.output_layer = keras.layers.Dense(1 , activation='linear')

    (variable) output_layer: Any
    def call(self, inputs):
        x = self.dense_1(inputs)
        x = self.drop(x)
        x = self.dense_2(x)
        x = self.drop(x)
        x = self.flatten(x)
        x = self.output_layer(x)
        return x

```

شکل ها ۱۲.۴- شبکه دو لایه

شبکه ۲ لایه مورد استفاده، دارای ۲ لایه، اولی با ۲۵۶ نورون و دومی با ۵۱۲ نورون است. این کار باعث می شود شبکه با اطلاعات افزونه کار بکند و شاید باعث دقت بالاتر بشود.

در هر دو شبکه، قبل از خروجی، Dropout قرار داده شده است تا اثر overfit کمتر شود.

#### ۶-۴. آموزش مدل

```

1 input_layer = keras.layers.Input(scaled_df_train.iloc[0,:-1].shape)
2 output = MLP_One()(input_layer)
3 model_1 = keras.Model(inputs=input_layer, outputs=output)
4 model_1.compile(optimizer=keras.optimizers.SGD(momentum=0.1), loss=keras.losses.mean_absolute_error
5 , metrics=[keras.metrics.R2Score , keras.metrics.mean_squared_error , keras.metrics.huber])
6 model_1.summary()

✓ 0.2s
Model: "functional"



| Layer (type)             | Output Shape | Param # |
|--------------------------|--------------|---------|
| input_layer (InputLayer) | (None, 21)   | 0       |
| mlp_one (MLP_One)        | (None, 1)    | 47,105  |


```

شکل ها ۱۲.۴- کانفیگ و تعداد پارامتر شبکه تک لایه

شبکه مورد استفاده با کمک Mean Absolute error آموزش داده می شود. دلیل استفاده از این متريک، اين است که چون مقادير خود داده ها نرمال شده و بين ۰ و ۱ قرار داده شده اند، اگر از متريک های تواد ۲ برای regression مانند Mean Squared Error استفاده شود، تفاوت کوچکتر از واقعیت دیده می شود. همچنین MAE نسبت به Outlier و استفاده از نرم منهتن مقاوم تر است. شبکه به تعداد epoch ۳۰ آموزش داده می شود. البته در ادامه نشان داده می شود که لزوماً این موضوع می تواند درست نباشد و بر اساس تعداد لایه و نوع preprocess، برای کیس های مختلف، کمی بهتر عمل می کند.

برای SGD به همراه Optimizer Momentum از بهره می بریم، به این دلیل که عدد گرادیانها کوچک بوده برای اسکیل کردن داده ها و می خواهیم اثر گرادیان های قبلی در آپدیت ها ماکسیم باشد و جلوی پدیده vanishing

حد ممکن بدوز skip connection گرفته شود. در متريک ها برای ارزیابی، نشان میدهيم که چرا انتخاب MAE بهتر از MSE بوده است.

برای شبکه ۲ لايه، به صورت مشابه داريم:

```
1 input_layer = keras.layers.Input(scaled_df_train.iloc[0,:-1].shape)
2 output = MLP_two()(input_layer)
3 model_2 = keras.Model(inputs=input_layer, outputs=output)
4 model_2.compile(optimizer=keras.optimizers.SGD(momentum=0.1), loss=keras.losses.mean_absolute_error
5 , metrics=[keras.metrics.R2Score , keras.metrics.mean_squared_error , keras.metrics.huber])
6 model_2.summary()

Model: "functional_1"



| Layer (type)               | Output Shape | Param # |
|----------------------------|--------------|---------|
| input_layer_1 (InputLayer) | (None, 21)   | 0       |
| mlp_two (MLP_two)          | (None, 1)    | 137,729 |



Total params: 137,729 (538.00 KB)

Trainable params: 137,729 (538.00 KB)

Non-trainable params: 0 (0.00 B)
```

شكل ها ۱۳.۴ - کانفیگ شبکه ۲ لايه

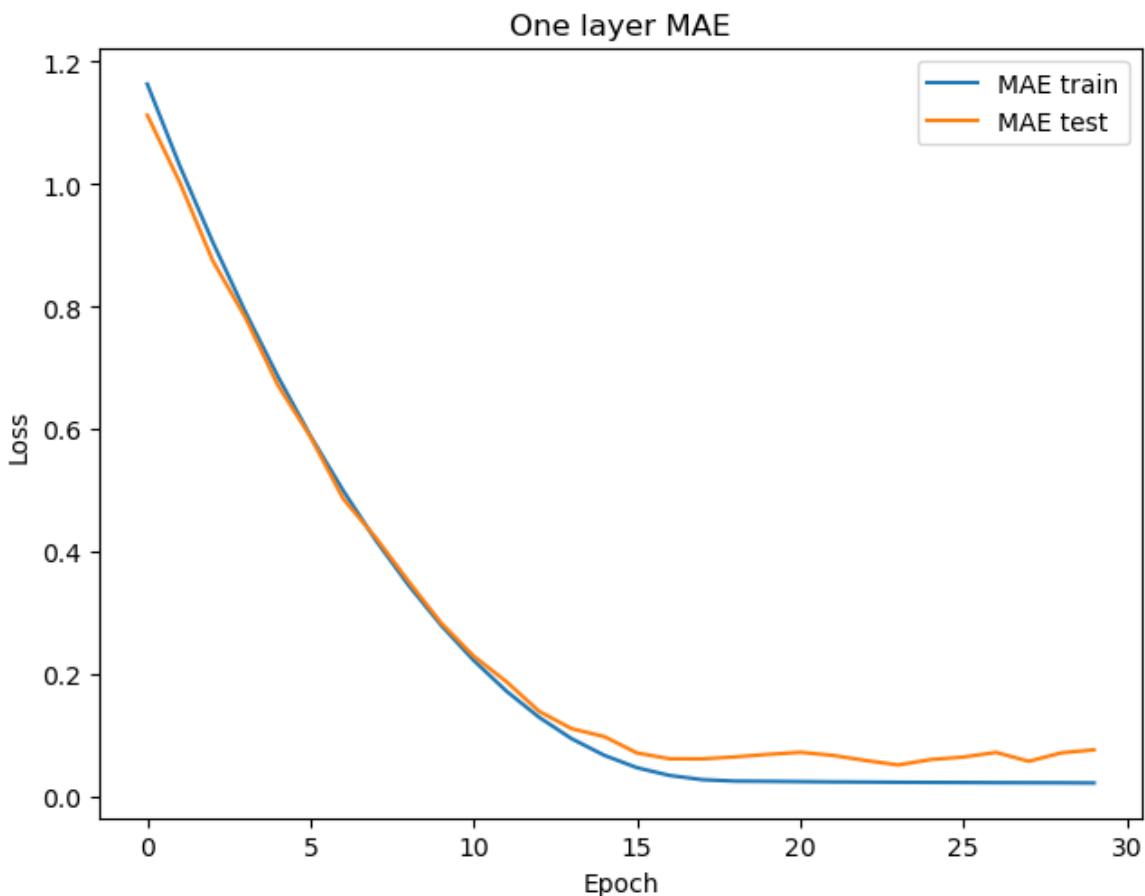
کانفیگ مورد استفاده مانند شبکه تک لايه است.

تعداد epoch مورد استفاده در شبکه ۲ لايه، به خاطر اثر تاخيری گراديان در لايه مخفى اول واضحاً باید بيشتر از شبکه تک لايه بوده و ۱۰۰ انتخاب شده است تا همگرائي انجام بپذيرد.

در تمام آموزش ها batch size برابر با ۶۴ است.

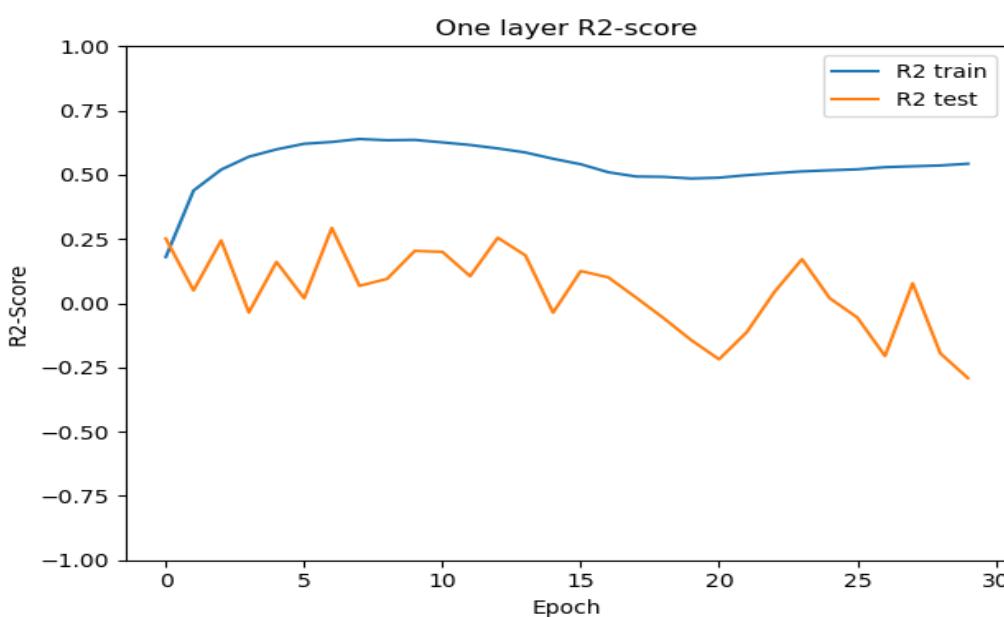
مدل تک لايه

در آموزش مدل تک لايه با MAE داريم:



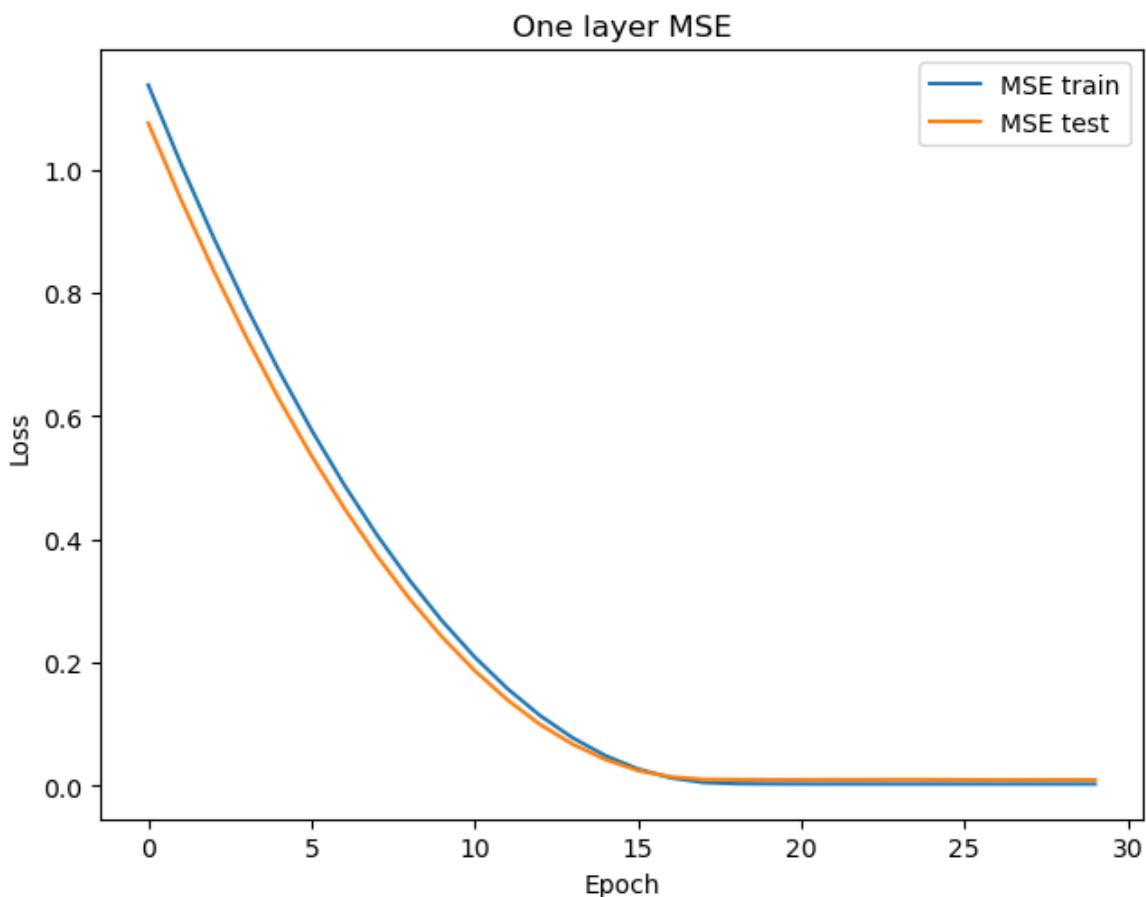
شکل ها ۶۷: شکل ۱۴.۴- آموزش و تست برای شبکه تک لایه با **MAE**

در این کیس، متریک  $R^2$  در طول آموزش به شرح زیر تغییر می کنند:



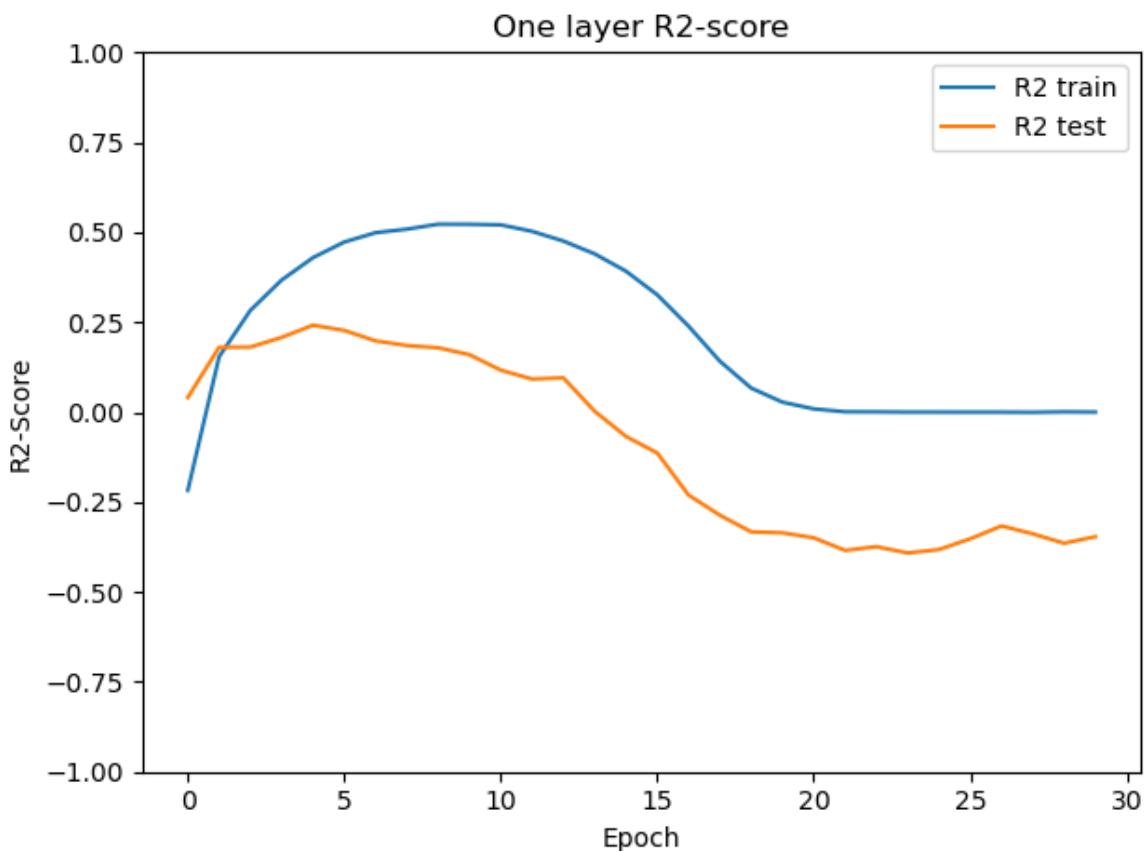
شکل ها ۶۸: شکل ۱۵.۴- متریک  $R^2$  برای تک لایه با **MAE**

در آموزش شبکه تک لایه با MSE داریم:



شکل ها ۱۶.۴-۶۹: آموزش شبکه تک لایه با MSE

همچنین متریک  $R^2$  به صورت زیر است:



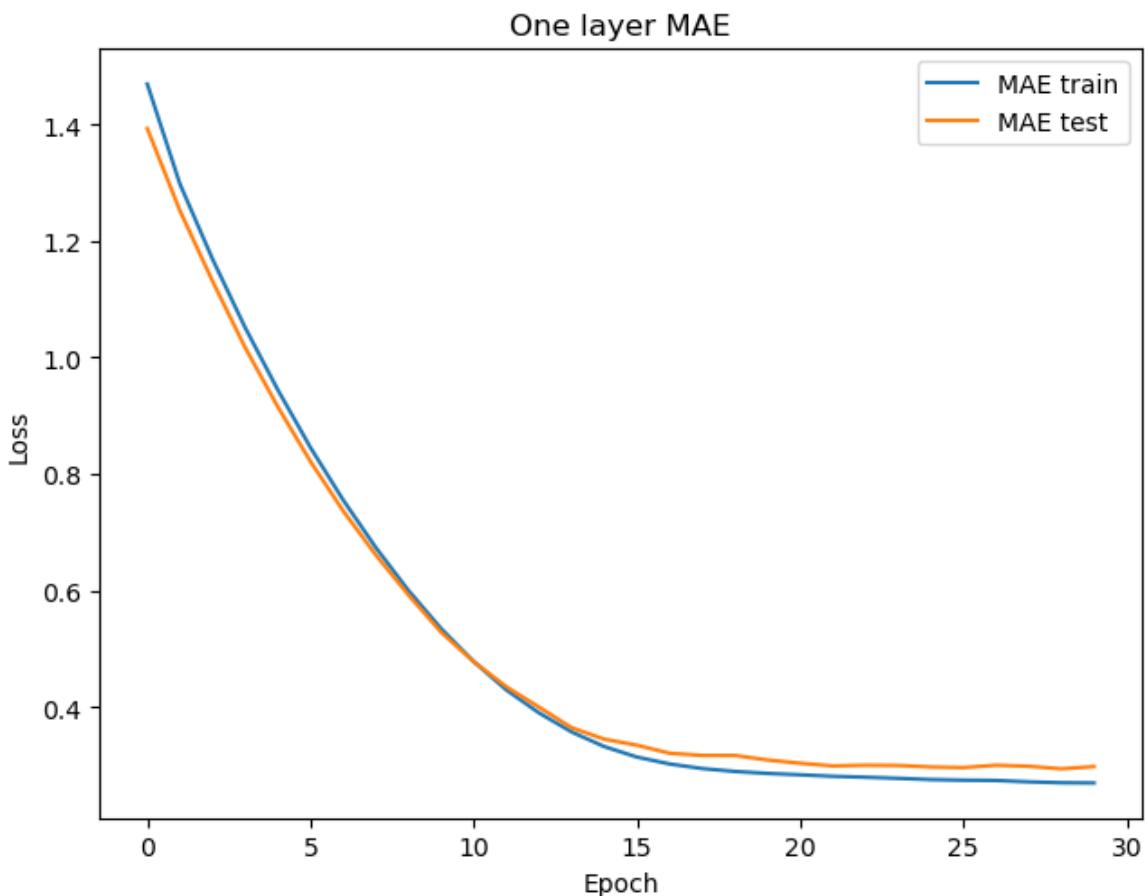
شکل ها ۷۰: شکل ۱۷.۴- آموزش شبکه با **MSE**، متریک **R<sup>2</sup>**

در حالت آموزش با **MSE**، و **MAE**، برای شبکه تک لایه و بر روی کل داده های تست داریم:

جدول ۱.۴. جواب ۱.۴. مقایسه شبکه تک لایه با **MINMAX** اسکیلر و دو مختلف

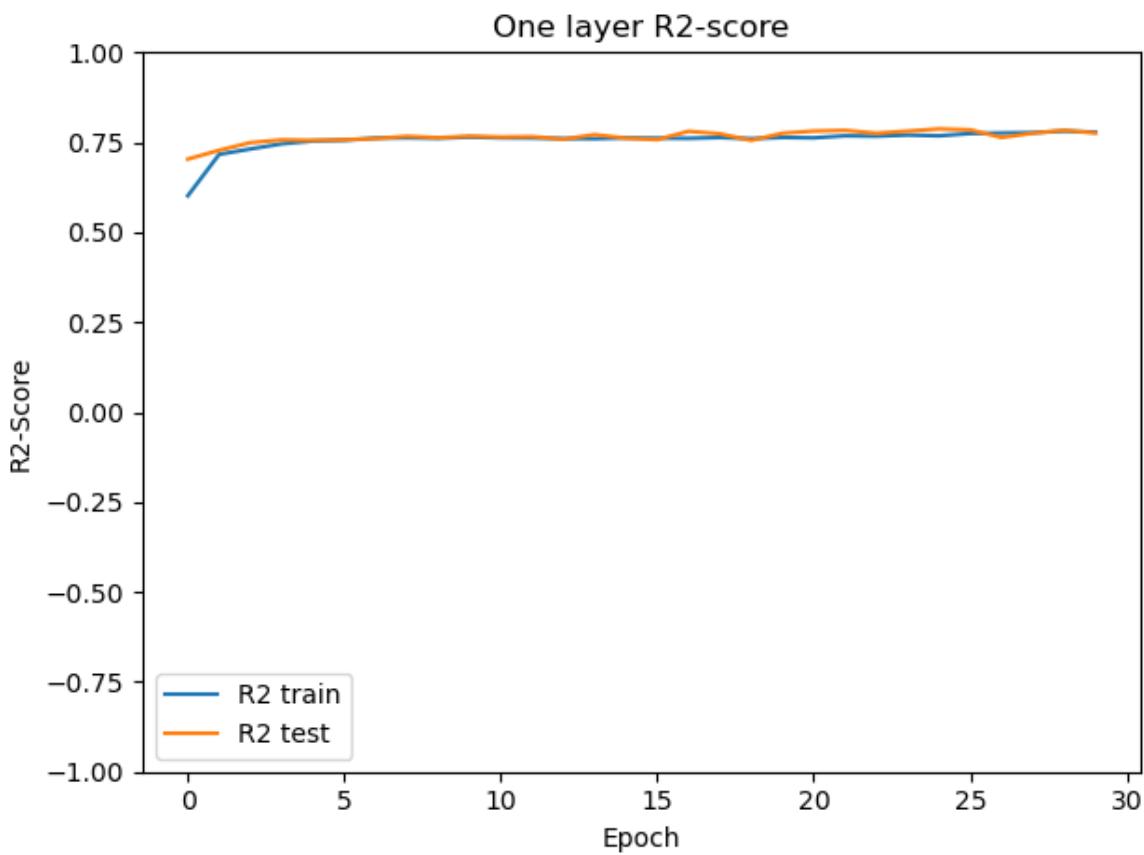
Method\Metric	MSE	MAE	Huber	R <sup>2</sup> -Score
MAE	۰.۰۰۸۳	۰.۰۷۴۰	۰.۰۰۴۲	-۰.۲۹۰۳
MSE	۰.۰۰۸۹	۰.۰۵۷۳	۰.۰۰۴۳	-۰.۳۳۷۰

به طور مشابه، اگر به جای **MINMAX** اسکیلر(موسوم در پردازش تصویر) از **STANDARD** اسکیلر(موسوم در TabularData) استفاده شود:



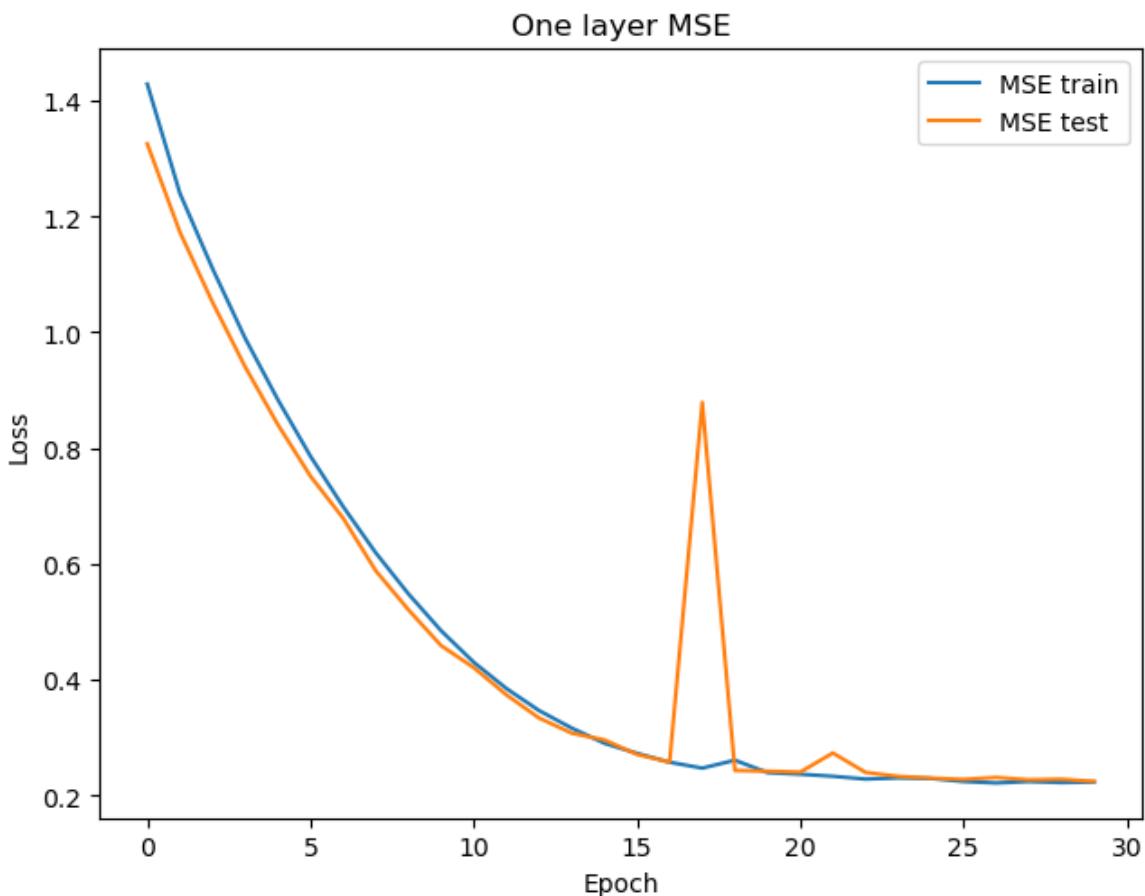
شکل ها ۷۱: شکل ۱۸.۴- آموزش تک لایه **MAE** با استفاده از **Standard Scaler**

مقدار متریک  $R^2$  نیز به شکل زیر تغییر می کند:



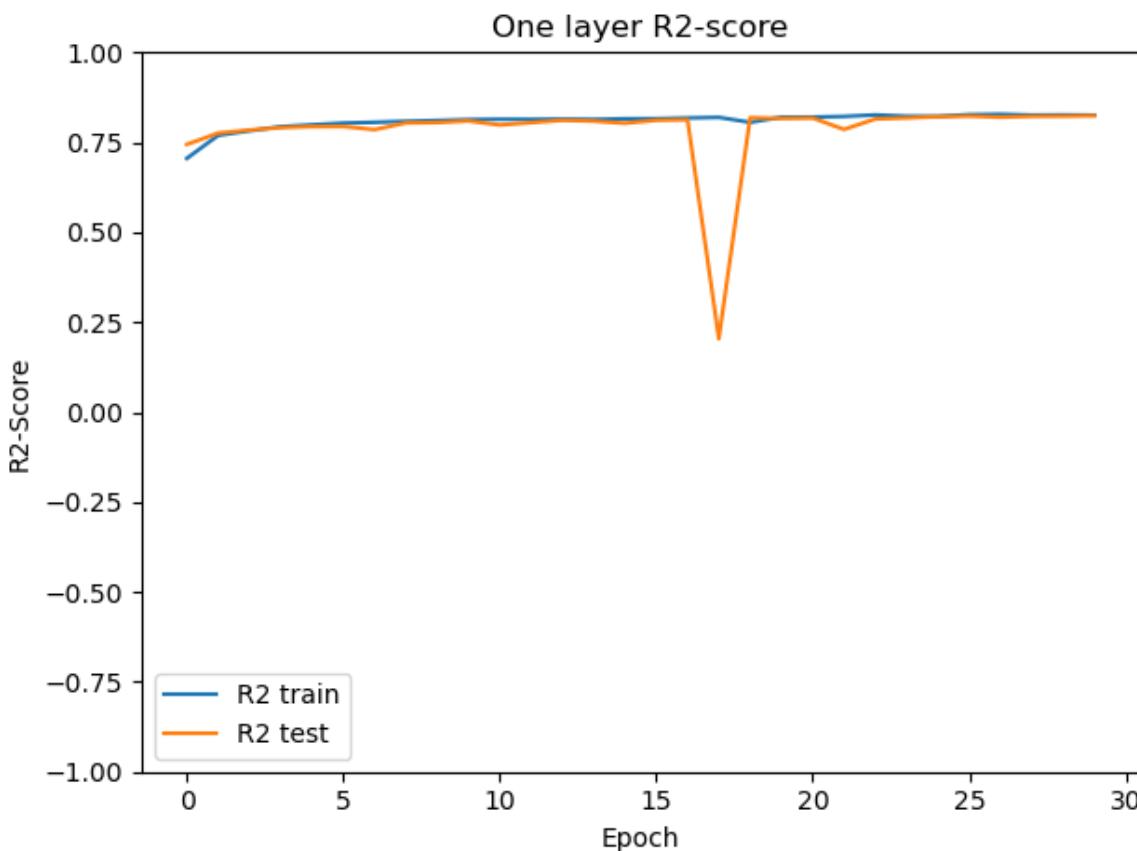
شکل ها ۷۲: شکل ۱۹.۴- متریک **R<sup>2</sup>** برای آموزش تک لایه **MAE** با استفاده از **Standard Scaler**

به طور مشابه، برای استفاده از **MSE** داریم:



شکل ها ۷۳: شکل ۲۰.۴- آموزش تک لایه **MSE** با استفاده از **Standard Scaler**

مقدار متریک  $R^2$  نیز به شکل زیر تغییر می کند:



شکل ها ۷۴: شکل ۲۱.۴ - متریک  $R^2$  برای آموزش تک لایه **MSE** با استفاده از **Standard Scaler**

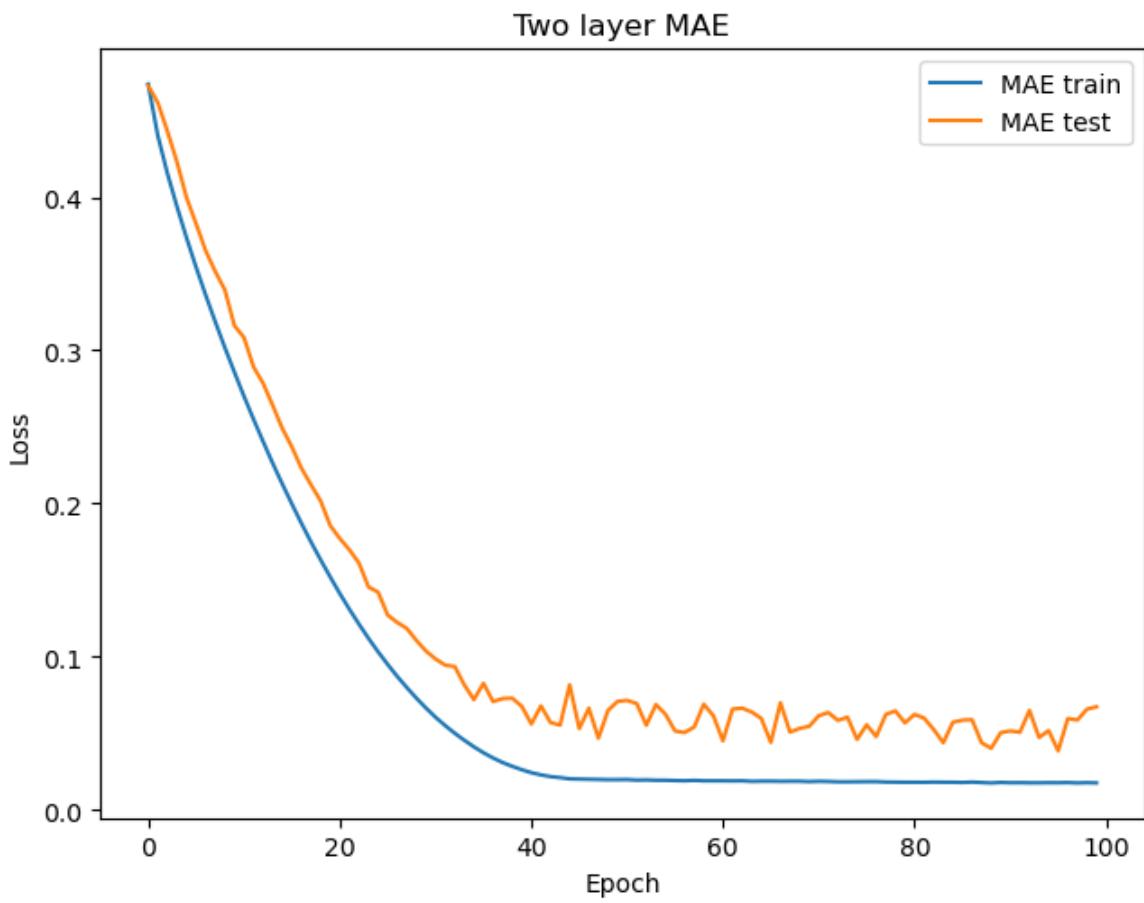
در حالت آموزش با **MSE**, **MAE**، برای شبکه تک لایه و بر روی کل داده های تست با **Standard Scaler** داریم:

جدول ۵.۲.۴ - مقایسه شبکه تک لایه با **Loss function** و **Standard Scaler** و دو مختلف

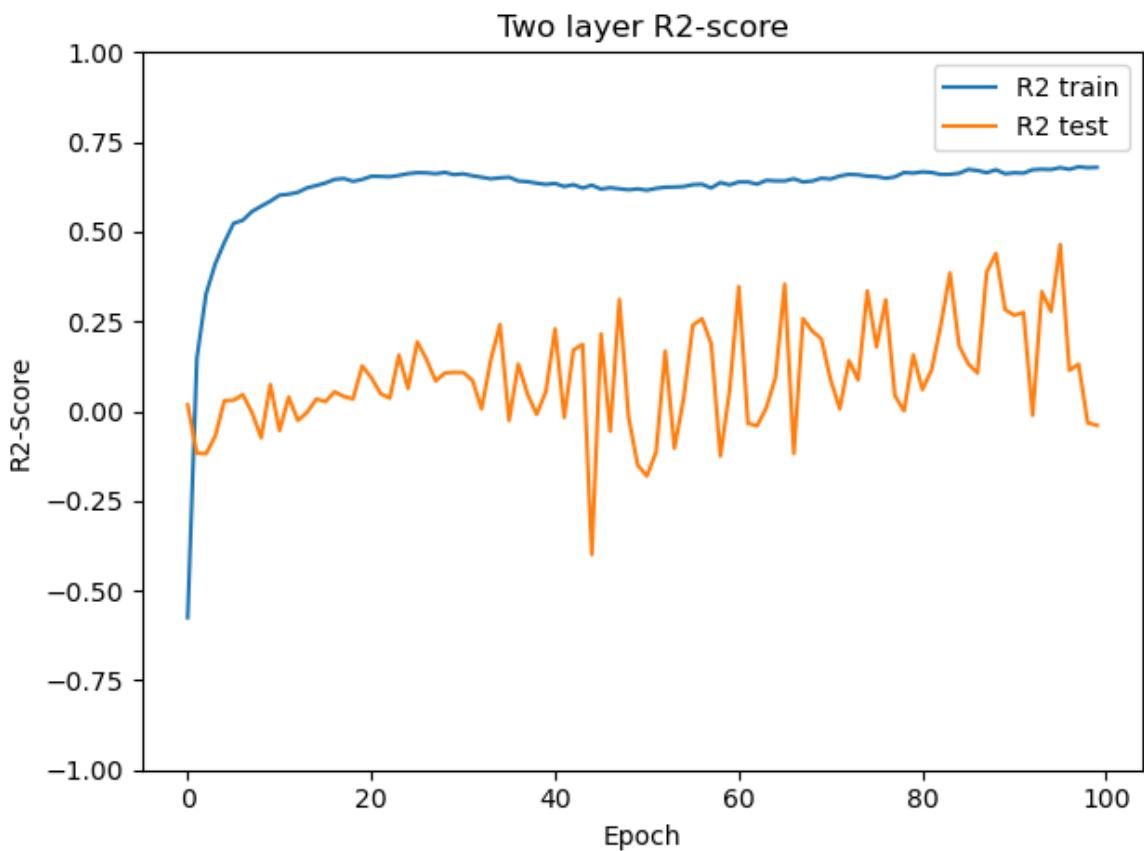
Method\Metric	MSE	MAE	Huber	$R^2$ -Score
MAE	۰.۲۰۶۸	۰.۲۹۰۱	۰.۰۸۱۱	۰.۷۸۱۴
MSE	۰.۲۱۲۳	۰.۲۵۱۶	۰.۰۷۲۵	۰.۸۲۵۵

مدل دو لایه

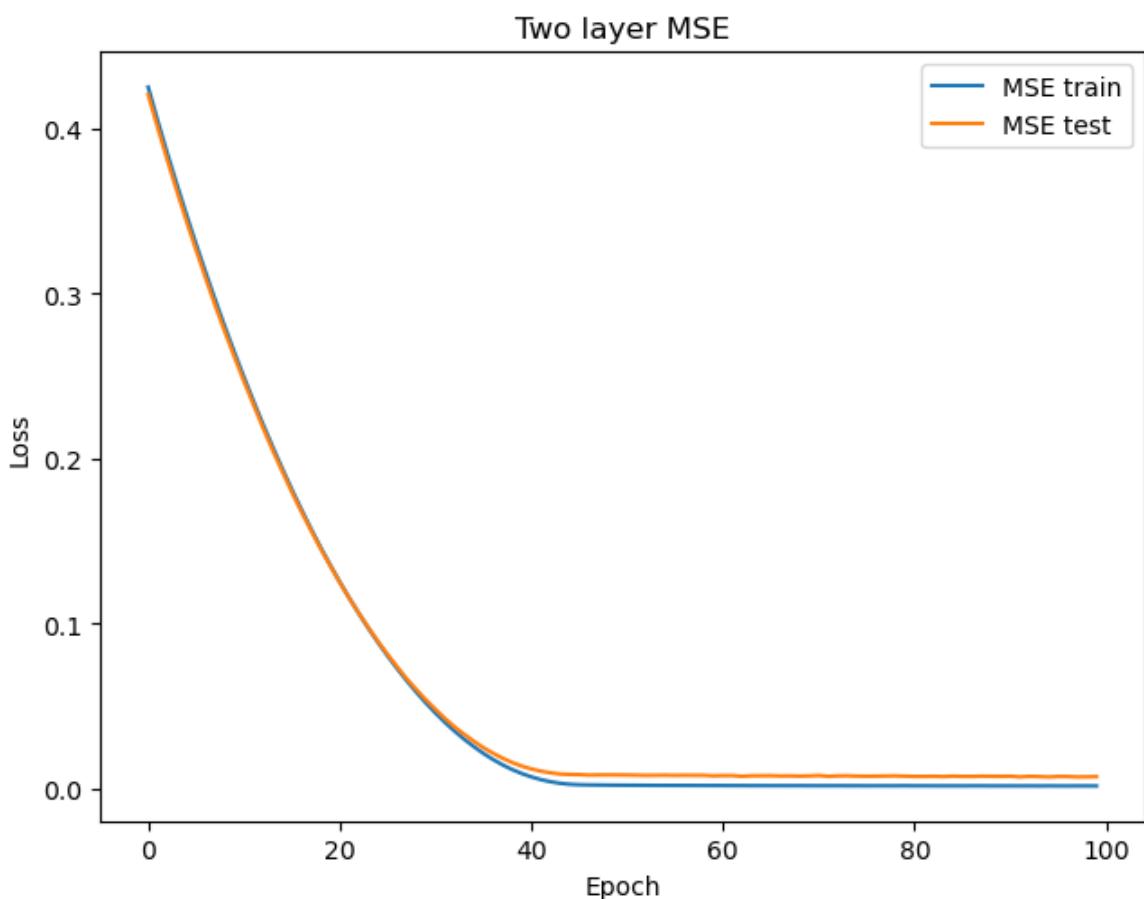
در حالت استفاده از **MINMAX Scalar**، مانند مدل تک لایه عمل کرده و نمودار های **Metric** و **Loss** زیر بدست می آیند:



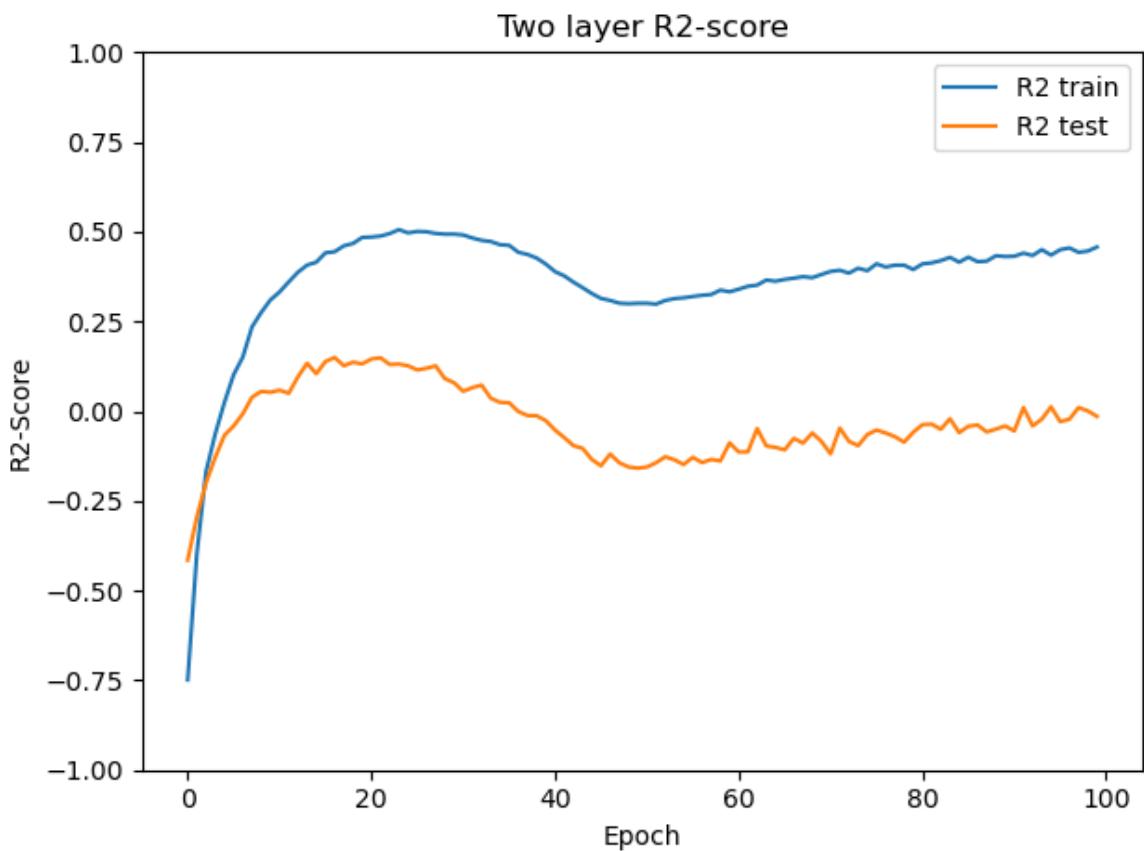
شکل ها ۷۵: شکل ۲۲.۴- آموزش دو لایه **MAE** با استفاده از **MinMax Scalar**



شکل ها: ۷۶- ۲۳.۴- آموزش دو لایه **MAE** با استفاده از **MinMax Scalar**، متریک **R2**



شکل ها ۷۷: شکل ۲۴.۴- آموزش دو لایه **MSE** با استفاده از **MinMax Scalar**



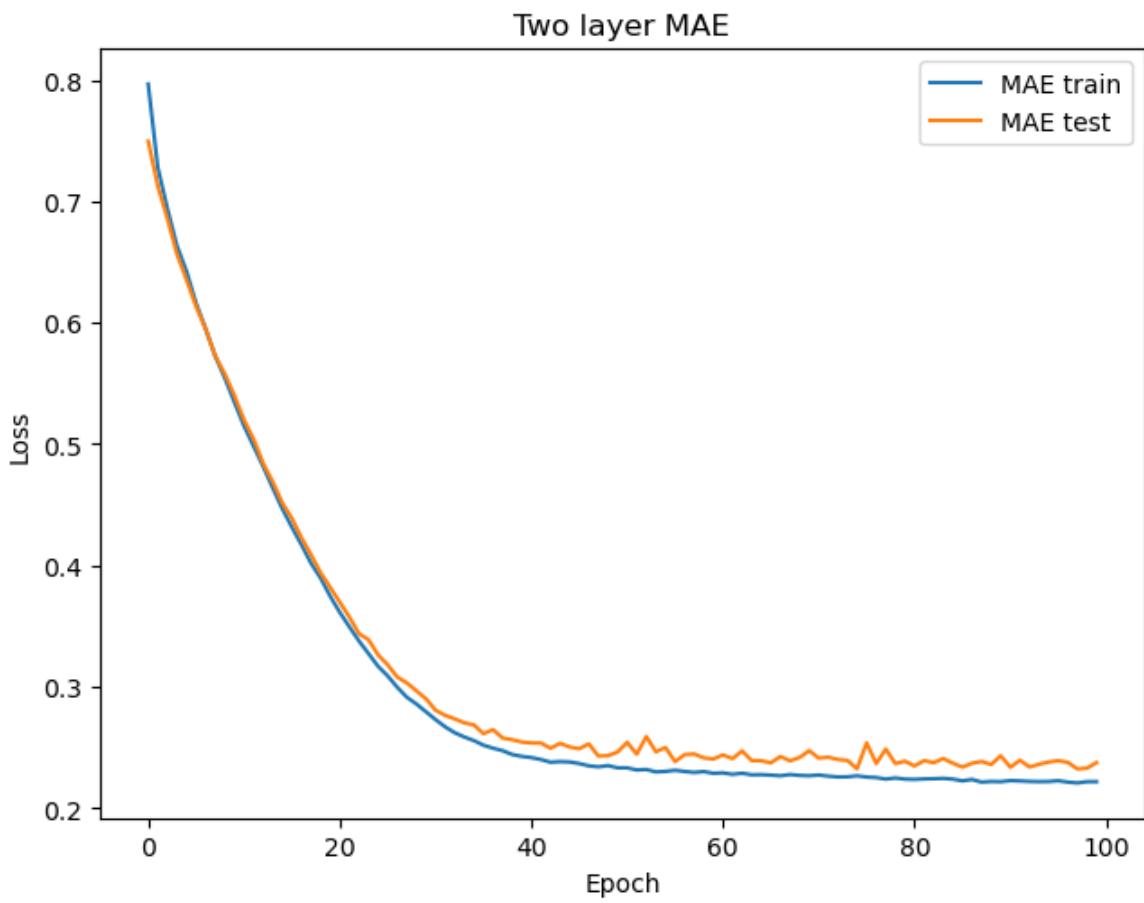
شکل ها: ۷۸- ۲۵.۴- آموزش دو لایه **MSE** با استفاده از **MinMax Scalar**، متریک **R<sub>2</sub>**

در نهایت، مقایسه بین شبکه های دو لایه با MINMAX Loss function مختلف در متریک ها به نحوه زیر بدست می آید:

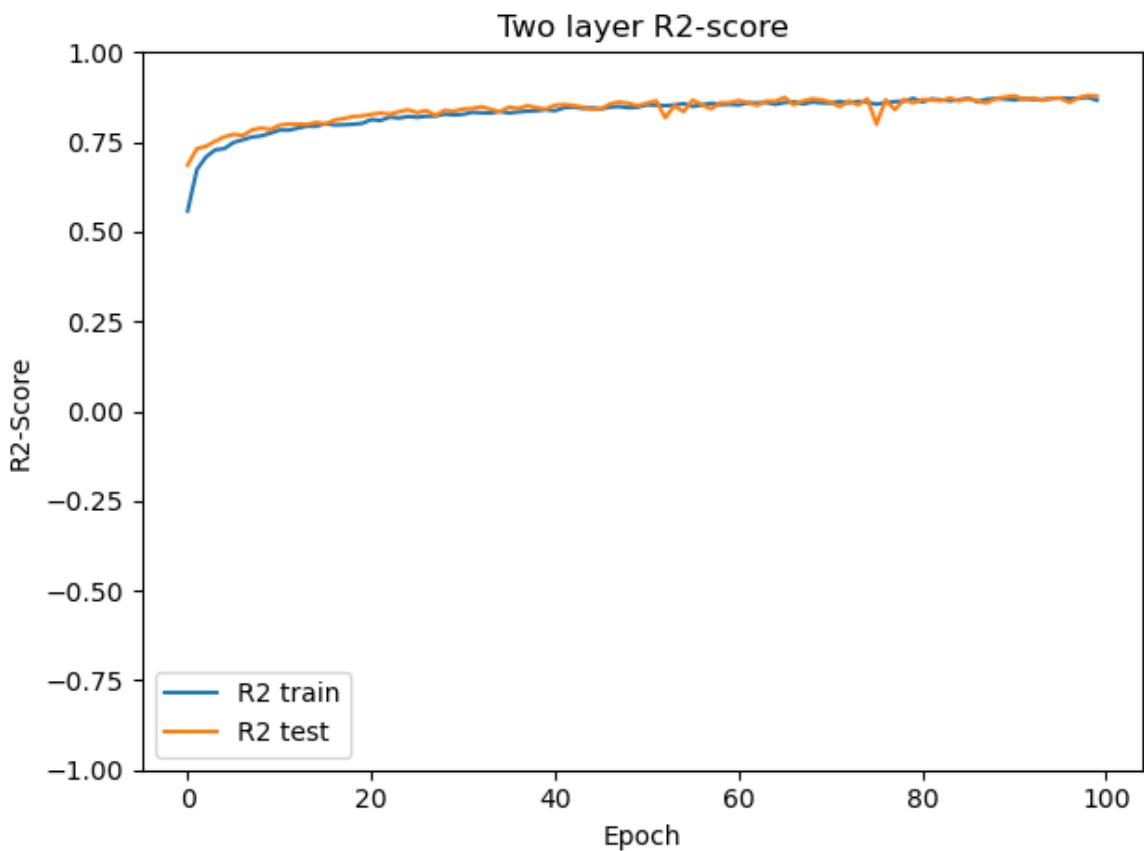
جدول ۳.۶.۴. مقایسه شبکه دو لایه با **MINMAX** اسکیلر و دو **Loss function** مختلف

Method\Metric	MSE	MAE	Huber	R <sub>2</sub> -Score
MAE	۰.۰۰۶۷	۰.۰۶۵۷	۰.۰۰۳۴	۰.۰۳۶۹-
MSE	۰.۰۰۷۰	۰.۰۵۱۷	۰.۰۰۳۳	۰.۰۰۶۳-

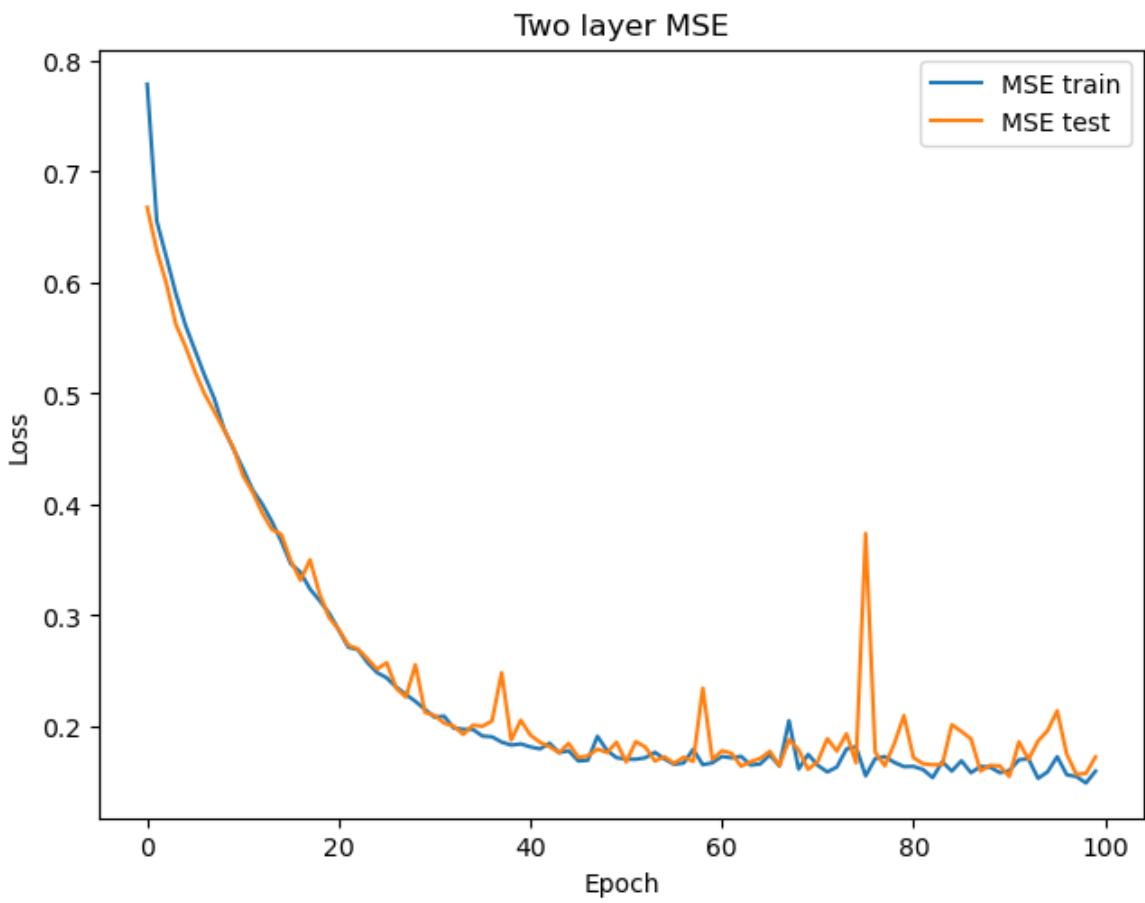
در حالت استفاده از Standard Scalar



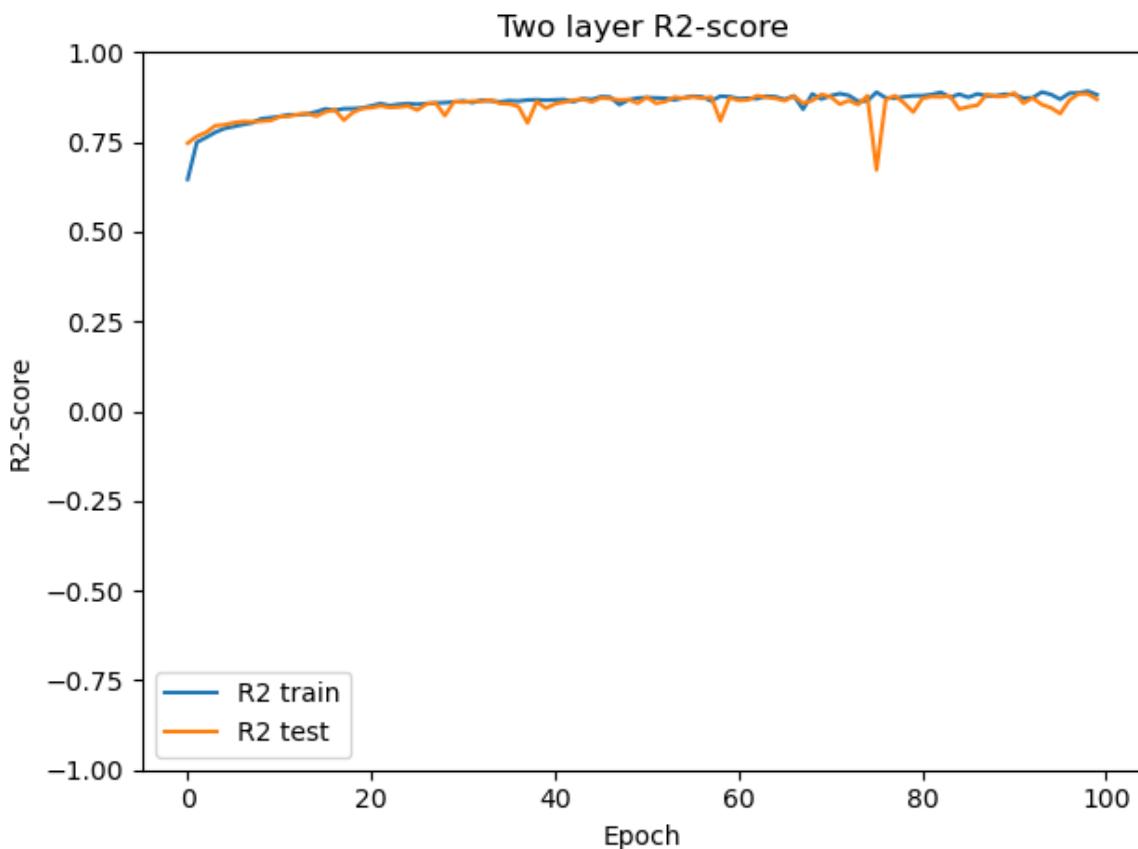
شکل ها ۷۹: شکل ۲۶.۴- آموزش دو لایه **MAE** با استفاده از **Standard Scalar**



شکل ها: ۸۰-۲۷.۴ آموزش دو لایه **MAE** با استفاده از **Standard Scalar** متریک **R<sub>2</sub>**



شکل ها ۸۱: شکل ۲۸.۴- آموزش دو لایه **MSE** با استفاده از **Standard Scalar**



شکل ها ۸۲: شکل ۲۹.۴- آموزش دو لایه MAE با استفاده از **Standard Scalar**. متریک **R<sub>2</sub>**

بنابراین، مقایسه بین MSE و MAE بر روی داده های آموزش به شکل زیر است:

جدول ۷.۴.۴. مقایسه شبکه دو لایه با **Standard Scaler** اسکیلر و دو Loss function مختلف

Method\Metric	MSE	MAE	Huber	R <sub>2</sub> -Score
MAE	۰.۱۱۰۸	۰.۲۳۲۰	۰.۰۵۰۶	۰.۸۸۲۰
MSE	۰.۱۶۷۴	۰.۲۱۲۱	۰.۰۵۳۹	۰.۸۶۵۸

در انتها، به دلیل بهتر بودن MAE در حالت تک لایه و استفاده از Standard Scalar (که این بهتر بودن بر اساس R<sub>2</sub> Score بوده و در قسمت تحلیل نتایج آمده است) و مقایسه این دو شبکه برای کل داده تست داریم:

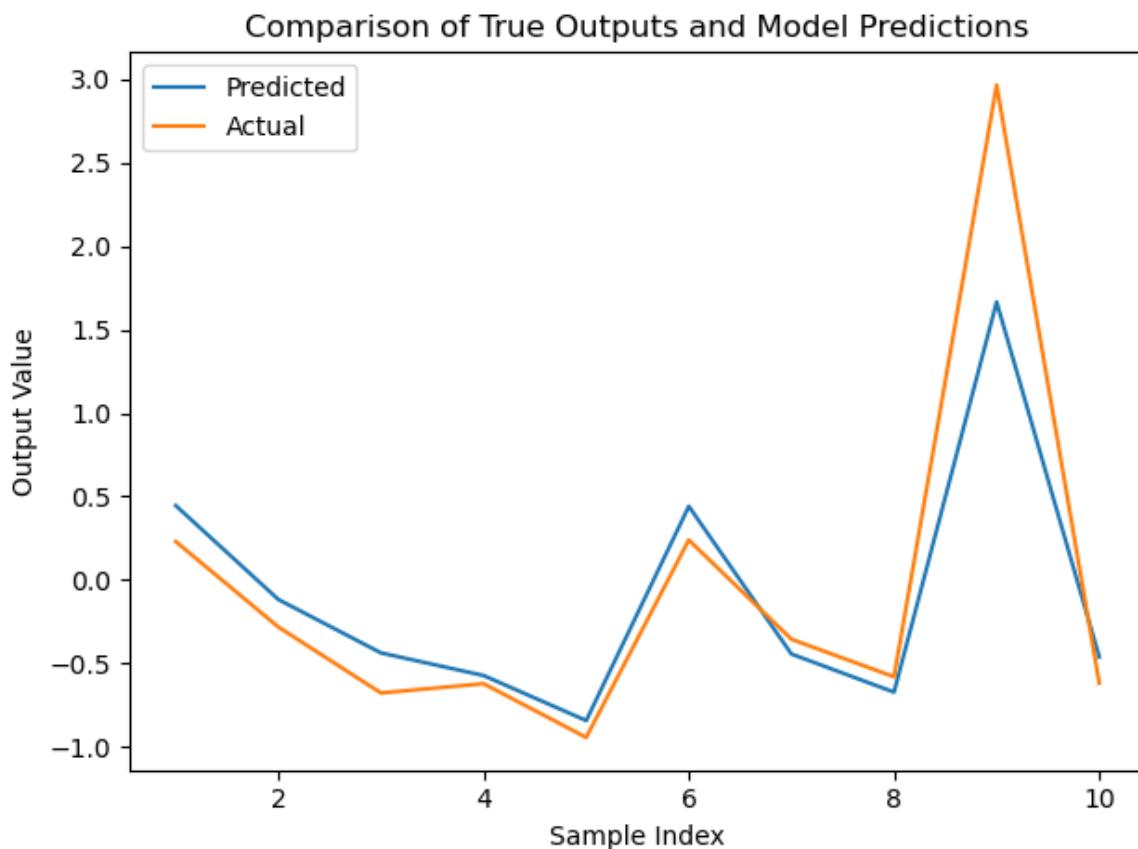
جدول ۴.۵. بهترین حالت شبکه ۲ لایه و تک لایه با همدیگر

Network \ Metric	MSE	MAE	Huber	R <sup>2</sup> -Score
Two Layer(MAE)	۰.۱۱۰۸	۰.۲۳۲۰	۰.۰۵۰۶	۰.۸۸۲۰
One Layer(MSE)	۰.۲۱۲۳	۰.۲۵۱۶	۰.۰۷۲۵	۰.۸۲۵۵

#### ۴-۷. تحلیل نتایج

انتخاب ۱۰ داده تصادفی

با انتخاب ۱۰ داده تصادفی از تست برای پیش بینی و استفاده از بهترین مدل ممکن (دو لایه، MAE, Standard)، شکل زیر حاصل می شود:



شکل ها ۸۳-۳۰.۴- تفاوت سمپل های پیش بینی شده و واقعی تست در دیتای اسکیل شده

از شکل ۴.۳۰، مشاهده می شود که مدل به خوبی می تواند بر اساس ویژگی های اسکیل شده، قیمت را بدست بیاورد و کار regression را به خوبی انجام می دهد.

تحلیل نتایج

اولین موضوع که به آذشاره می شود، دقیق بالاتر مدل هنگام استفاده از Standard Scaler در مقایسه با MINMAX است. از مقایسه جدول ۱.۴ با ۲.۴ و همچنین جدول ۳.۴ با ۴.۴ دیده می شود که عدد ۲R در مدل هایی که

آنها Preprocess آموزش منفی است، به این معنا که مدل‌ها بدنتر از میانگین به عنوان پیش‌بینی عمل کرده است. دلیل بهتر بودن نتایج حاصل از Standard Scaler (بر اساس بالاتر بودن R<sup>2</sup>-Score) این است که توزیع کردن داده‌ها با میانگین صفر و واریانس یک، باعث Smooth Hyperplane در فضای خروجی داده‌ها شده و تابع حاصل، پیوسته و کراندار و نزدیک‌تر به تابع محدب بوده و باعث همگراشی بهتر می‌شود. (از آنجا که SGD با Momentum اثبات می‌شود روی تابع محدب می‌تواند Min سراسری را پیدا کند)

برای مقایسه مدل‌ها، متریک R<sup>2</sup>، به صورت زیر معرفی شد:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

شکل ۳۱.۴- فرمول R<sup>2</sup>-Score

که این معیار، توضیح پذیری واریانس موجود در قیمت خانه را توسط ویژگی‌ها را نشان می‌دهد و هرچقدر این مقدار به ۱ نزدیک‌تر بشود، به این معنا است که مدل به fit بهتری دست یافته است.

با توجه به قسمت قبل، دیده می‌شود که به طور کلی، مدل ۲ لایه می‌تواند به R<sup>2</sup>-metric بالاتری از مدل‌تک لایه دست بیاید (جدول ۵.۴). این موضوع همچنین در مقایسه دو جدول ۱.۴ در دو حالت و جدول ۳.۴ در هر دو حالت دیده می‌شود برای مدل‌تک لایه و ۲ لایه در حالت استفاده از MinMax Scaler و همچنین دیده می‌شود که با اینکه R<sup>2</sup>-score هاستند، برای ۲ لایه عدد منفی کوچکتری بدست آمد. در مقایسه جدول ۲.۴ و ۴.۴ نیز دیده می‌شود که عدد مشت بزرگتری برای مدل‌های ۲ لایه در مقایسه با ۱ لایه داریم. این موضوع با اینکه مدل ۲ لایه General Function Approximator بهتری است، کاملاً منطبق بوده و افزونگی بیشتر، باعث شده تابعی پیچیده تر بر روی Data فیت کرده و به دقت بهتری بررسیم.

مقایسه بعدی، مقایسه MSE و MAE به عنوان Loss می‌باشد. در حالت تک لایه، در حالت MinMax Scaler، استفاده از MAE کمی بهتر بوده و R<sup>2</sup>-score کمی بزرگتر است، ولی در حالت Standard Scaler، استفاده از MSE باعث شده مدل در R<sup>2</sup>-Score با ۴ درصد عملکرد بیشتر مواجه شود. در مقایسه و در حالت ۲ لایه، برتری MAE در استفاده از Standard Scaler بوده که باعث ۲ درصد افزایش عملکرد ۲R شده و در حالت MinMax Scaler، تفاوت چندانی وجود ندارد. بنابراین، بهترین حالت مدل ۲ لایه با MAE بوده و بزرگتر بودن عدد Loss، باعث آپدیت بهتر لایه مخفی او که Vanishing Delayed Effect و Delayed Effect از گرادیان دارد شده و دقت بالاتری رسیده، در حالی که در ۱ لایه، چون Vanishing Effect و Vanishing Effect ای وجود نداشته، کوچکتر بودن اندازه گرادیان پایداری بیشتری به آپدیت‌ها داده و جمعاً بهتری در اثر بهینه‌سازی پیدا شده است. Equilibrium point

در مقایسه تعداد epoch مناسب، دیده می‌شود که بدون توجه به loss و روش Scale، مدل‌تک لایه تعداد کمتری از epoch‌ها را نیاز دارد تا به همگراشی برسد، در حالی که مدل ۲ لایه نیازمند تعداد بیشتری از epoch‌ها هست. این موضوع به دلیل این است که مدل ۲ لایه در لایه مخفی اول گرادیان حاصل از گرادیان داده دوم را دیده و چون داده‌ها بین ۰ و ۱ هستند،

ضرب دو عدد کوچکتر از ۱ را داریم پس از چند epoch اول و تغییرات کوچکتر است. بنابراین با در نظر گرفتن loss و Scaler، روش ۲ لایه نیازمند حدوداً ۴۰ دوره برای رسیدن به دقت یکسان با مدل تک لایه با ۱۵ بار آموزش است. این موضوع با استفاده از MSE شدت یافته و حتا تعداد دوره ها بیشتر می شود، چون عدد loss حتا کوچکتر از عدد loss برای MAE است. (تواند ۲ عدد بین ۰ و ۱ کوچک تر از خود عدد است).