به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

پرسش ۱ و ۴	نام و نام خانوادگی	سید مهدی حاجی سید حسین
	شماره دانشجویی	810100118
پرسش ۲ و ۳	نام و نام خانوادگی	علیرضا حسینی
	شماره دانشجویی	810100125
	مهلت ارسال پاسخ	۱۴۰۳.۱۲.۳۰

فهرست

شکل‌ها

جدول‌ها

پرسش ۱. تشخیص تقلب در کارت های اعتباری با استفاده از شبکه های عصبی چند لایه (MLP)

۱-۱. مقدمه

در این تمرین هدف تشخیص تقلب در کارت های اعتباری با استفاده از شبکه های عصبی چند لایه بوده است . که حالات مختلف شبکه ، به همراه هایپر پارامتر های متفاوت تست شده است .

۱-۲. پیش پردازش و بررسی دادگان

سوال دوم)

با استفاده از خود کتابخانه kagglehub دیتاست را لود کرده ایم . و خلاصه ای از آن به صورت زیر بوده است :

```
df.head(5)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.1
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.3
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.6
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.2

5 rows × 31 columns

شکل ۱ : پنج ردیف اول دیتاست

همچنین قسمتی از خلاصه آماری آن به صورت زیر است :

```
df.describe()
```

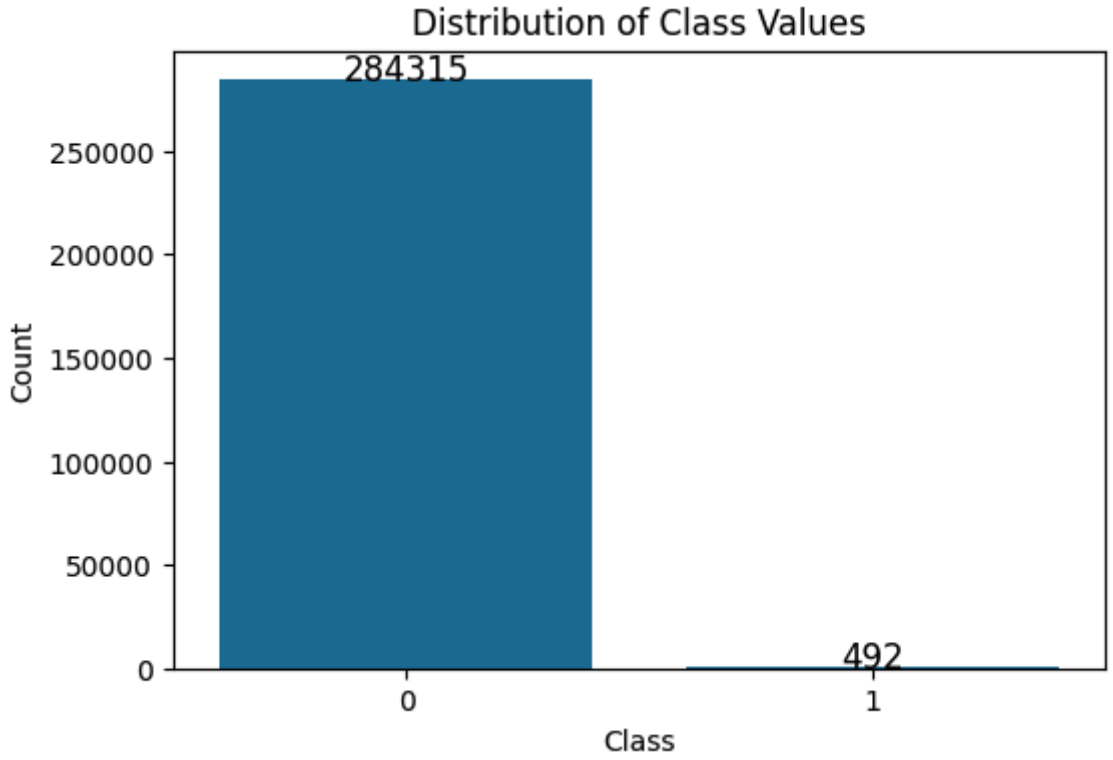
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	.
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	.
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	.
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	.
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	.
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	.
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	.
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	.
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	.

8 rows × 31 columns

شکل ۲ : قسمتی از خلاصه آماری دیتاست

سوال سوم)

توزیع ناهمگون داده ها با نمودار میله ای به صورت زیر نمایش داده شده است



شکل ۳ : توزیع ناهمگون دیتاهای کلاس های مختلف

سوال چهارم)

عدم تعادل داده‌ها باعث می‌شود مدل به سمت کلاس با داده‌های بیشتر متمایل شود، زیرا برای بهینه‌سازی تابع هزینه، پیش‌بینی کلاس غالب ساده‌تر است. این باعث کاهش توانایی مدل در شناسایی کلاس‌های کم‌تعداد و کاهش دقت (Accuracy) و F1-Score برای کلاس‌های نادر می‌شود.

سوال پنجم)

این کار را با استفاده از StandardScaler انجام دادیم که در واقع میانگین 0 و واریانس 1 برای همه ویژگی ها در نظر گرفته ایم . که با این کار باعث میشویم که همه مقدار ها در یک رنج بخصوصی باشند و با این کار باعث حفظ تعادل میان وزن های شبکه میشویم .

توجه شود که ستون Class چون مشخص کننده کلاس ما بوده است را دیگر اسکیل نکرده ایم و همان مقدار های binary را برایش در نظر گرفته ایم .

```
normalized_df.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24
0	-0.694242	-0.044075	1.672773	0.973366	-0.245117	0.347068	0.193679	0.082637	0.331128	0.083386	...	-0.024923	0.382854	-0.176911	0.110507
1	0.608496	0.161176	0.109797	0.316523	0.043483	-0.061820	-0.063700	0.071253	-0.232494	-0.153350	...	-0.307377	-0.880077	0.162201	-0.561131
2	-0.693500	-0.811578	1.169468	0.268231	-0.364572	1.351454	0.639776	0.207373	-1.378675	0.190700	...	0.337632	1.063358	1.456320	-1.138092
3	-0.493325	-0.112169	1.182516	-0.609727	-0.007469	0.936150	0.192071	0.316018	-1.262503	-0.050468	...	-0.147443	0.007267	-0.304777	-1.941027
4	-0.591330	0.531541	1.021412	0.284655	-0.295015	0.071999	0.479302	-0.226510	0.744326	0.691625	...	-0.012839	1.100011	-0.220123	0.233250

5 rows × 30 columns

شکل ۴ : ۵ ردیف اول دیتا ها پس از نرمال سازی

سوال ششم)

با استفاده از تابع train_test_split در کتابخانه sklearn این کار را انجام داده ایم و همچنین به آن یک seed رندون داده ایم تا دیتا به صورت همگون جدا سازی شوند . همانطور که مشاهده میکنید ۷۰ درصد داده هایی که کلاس Fraud را دارند ، درون دیتا آموزش قرار دارند .

```
train_data, test_data = train_test_split(normalized_df, test_size=0.3, stratify=normalized_df['Class'], random_state=39)
```

```
train_data['Class'].value_counts()
```

```
Class
0    199020
1       344
Name: count, dtype: int64
```

```
test_data['Class'].value_counts()
```

```
Class
0    85295
1      148
Name: count, dtype: int64
```

```
print(train_data['Class'].sum() / (test_data['Class'].sum() + train_data['Class'].sum()))
```

```
0.6991869918699187
```

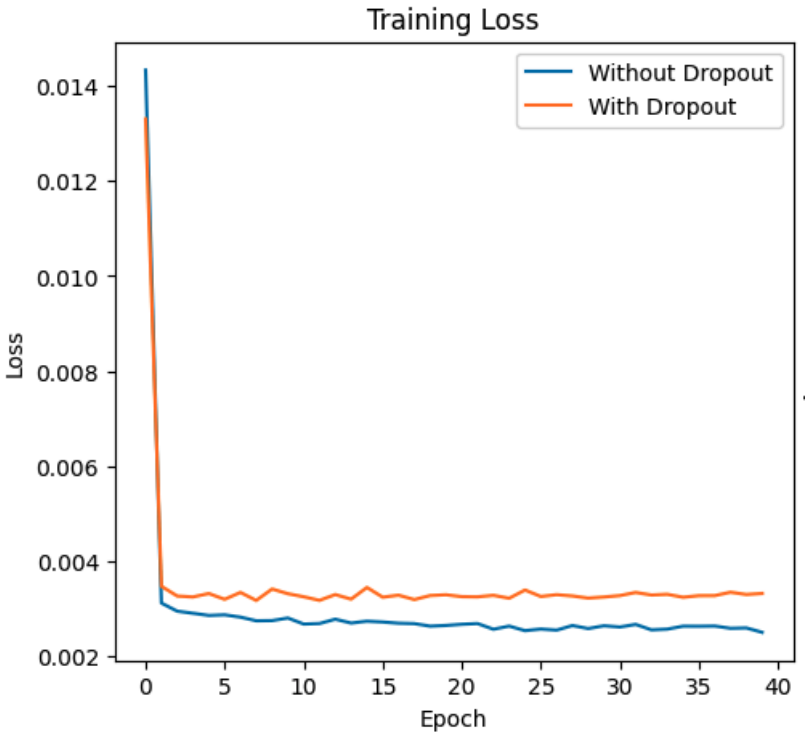
شکل ۵ : تقسیم داده های آموزش و ارزیابی

۱-۳. طراحی و پیاده سازی یک شبکه MLP ساده

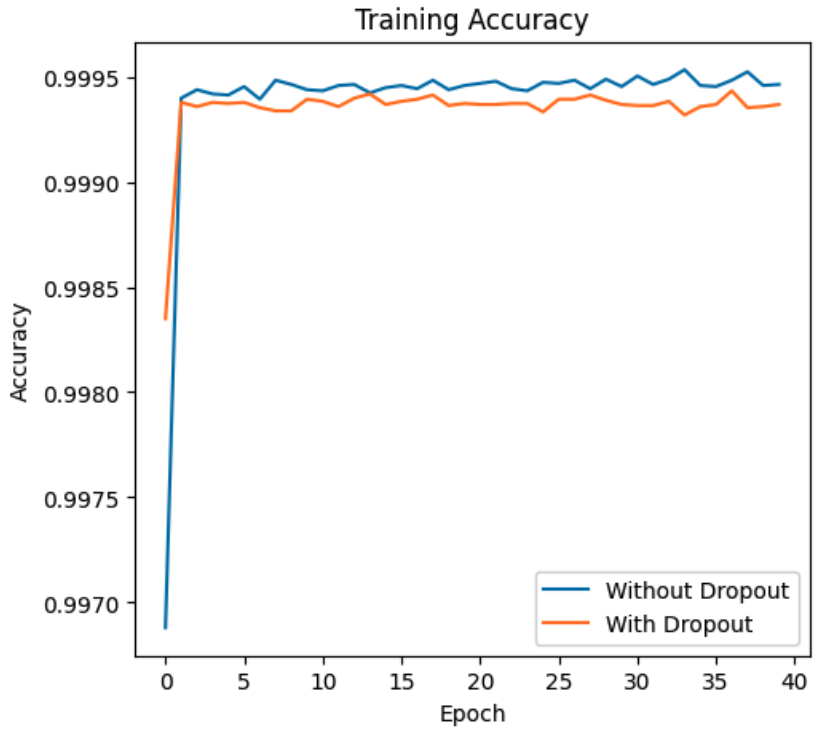
سوال دوم)

مقایسه شبکه عصبی بدون Dropout و به همراه Dropout 30% در زمان آموزش :

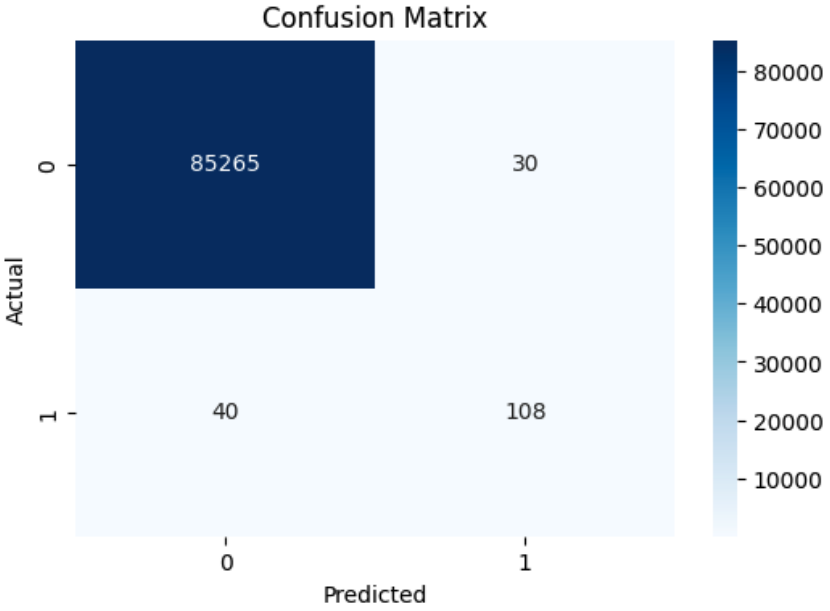
Dropout برای جلوگیری از overfitting استفاده می‌شود. با حذف تصادفی نورون‌ها در هر مرحله آموزش، شبکه یاد می‌گیرد تا به همه نورون‌ها وابسته نباشد و به‌طور کلی‌تر عمل کند. این باعث بهبود عمومیت (generalization) مدل می‌شود.



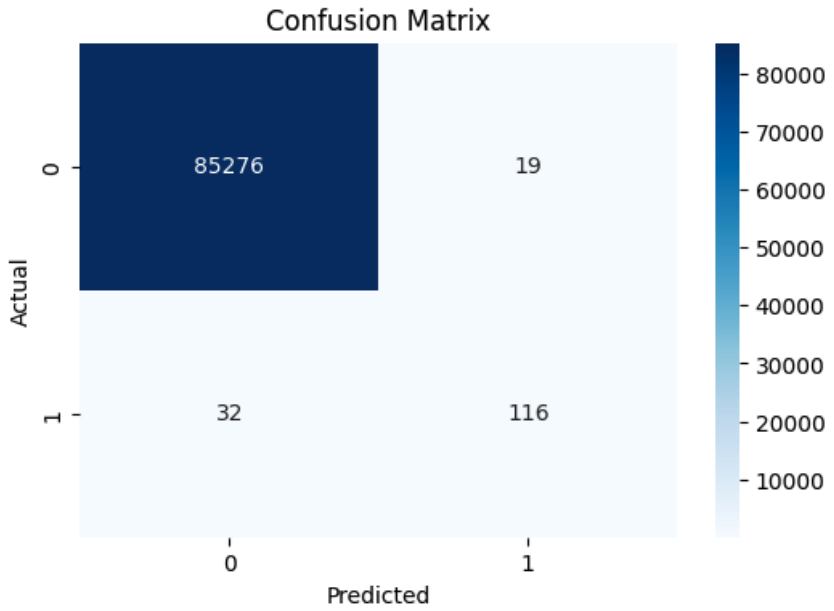
شکل ۶ : نمودار loss برای دو مدل به همراه dropout و بدون dropout



شکل ۷ : نمودار accuracy برای دو مدل به همراه dropout و بدون dropout



شکل ۸ : ماتریس آشفتگی برای مدل بدون dropout



شکل ۹ : ماتریس آشفتگی برای مدل به همراه dropout به میزان ۳۰ درصد

تحلیل ماتریس های آشفتگی : همانطور که مشاهده میکنید مدل به همراه dropout برای تشخیص کلاس ۱ (کلاس های تقلب دار) عملکرد بهتری داشته است و به میزان ۸ تقلب درست تر توانسته است از مدل بدون dropout تشخیص دهد ، که در مدل بدون dropout این نمونه ها به اشتباه متعلق به کلاس های بدون تقلب شناخته شده اند . پس مدل به همراه dropout در واقع یک generalization بهتری دارد .

جدول ۱ : ارزیابی مدل بدون dropout

متریک ارزیابی	میزان
دقت (Accuracy)	0.9992
دقت کلاس مثبت (Precision)	0.7826
بازخوانی (Recall)	0.7297
F1 - Score	0.9852

جدول ۲ : ارزیابی مدل به همراه dropout به میزان ۳۰ درصد

متریک ارزیابی	میزان
دقت (Accuracy)	0.9994
دقت کلاس مثبت (Precision)	0.8593
بازخوانی (Recall)	0.7838
F1 - Score	0.9803

$$Accuracy \Rightarrow \frac{True\ Pos + True\ Neg}{Total\ Samples}$$

$$Precision \Rightarrow \frac{True\ Pos}{True\ Pos + False\ Pos}$$

$$Recall \Rightarrow \frac{True\ Pos}{True\ Pos + False\ Neg}$$

$$F1 - Score \Rightarrow \frac{Precision \times Recall \times 2}{Precision + Recall}$$

همان طور که مشاهده میشود مدل به همراه dropout در همه متریک ها بجز (f1-score) به میزان قابل توجهی (با توجه به ناهمگونی کلاس ها) بهتر عمل کرده است ! علت این امر این بوده است که با استفاده از dropout توانسته ایم کاری کنیم که شبکه یاد بگیرد تا به همه نورون‌ها وابسته نباشد و به‌طور کلی‌تر عمل کند.

علت اینکه نمره متریک F1-Score بهتر نشده است ممکن است این باشد که در این دیتاست ما چون دیتا به صورت بسیار ناهمگون بوده است خیلی قابلیت تفکیک قابل توجهی ای نداریم و مدل ها به میزان کمی از یکدیگر تفاوت خواهد داشت .

منحنی ROC :

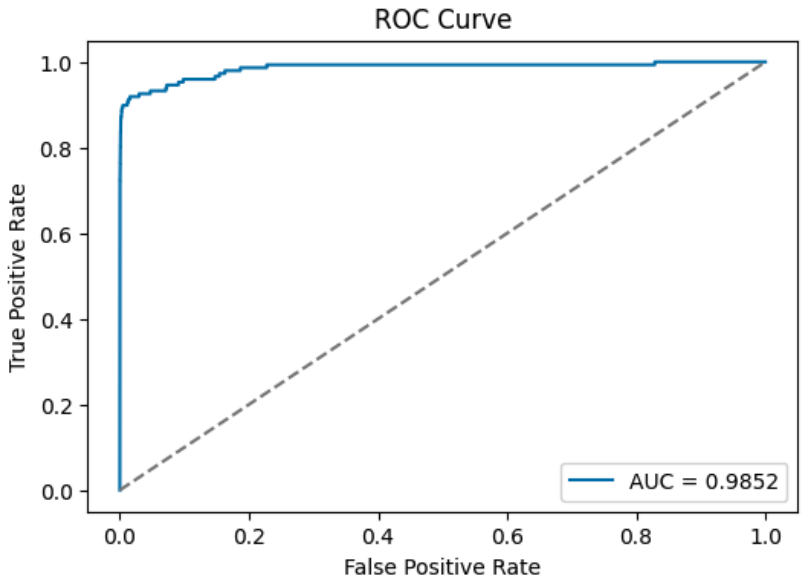
منحنی ROC یک نمودار است که عملکرد مدل در جداسازی کلاس‌های مثبت و منفی را نشان می‌دهد. این نمودار، نرخ مثبت واقعی (True Positive Rate یا TPR) را در برابر نرخ مثبت کاذب (False Positive Rate یا FPR) رسم می‌کند.

مقدار Area Under the Curve (AUC):

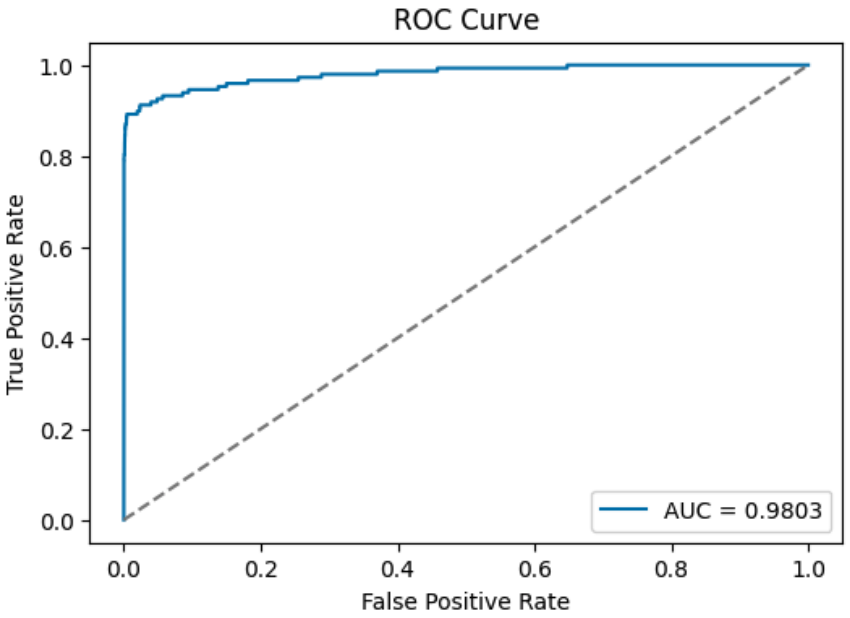
مقدار AUC مساحت زیر منحنی ROC است. AUC بین ۰ و ۱ قرار دارد:

- اگر AUC = 0.5 → مدل عملکرد تصادفی دارد.
- اگر AUC → 1 → مدل عملکرد عالی دارد.
- اگر AUC → 0 → مدل عملکرد ضعیف دارد (در حال معکوس عمل کردن است).

هرچه مقدار AUC به ۱ نزدیک‌تر باشد، مدل عملکرد بهتری در تفکیک کلاس‌ها دارد.



شکل ۱۰ : منحنی ROC و میزان مساحت زیر آن AUC برای مدل بدون dropout



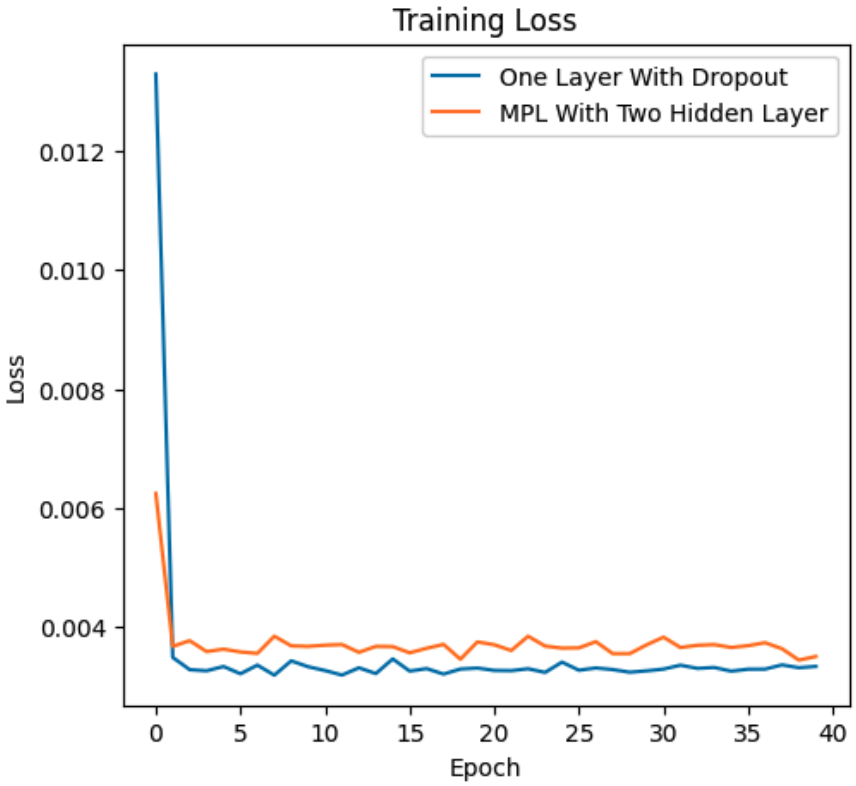
شکل ۱۱ : منحنی ROC و میزان مساحت زیر آن AUC برای مدل به همراه dropout

۱-۴. طراحی یک شبکه عصبی MLP عمیق تر

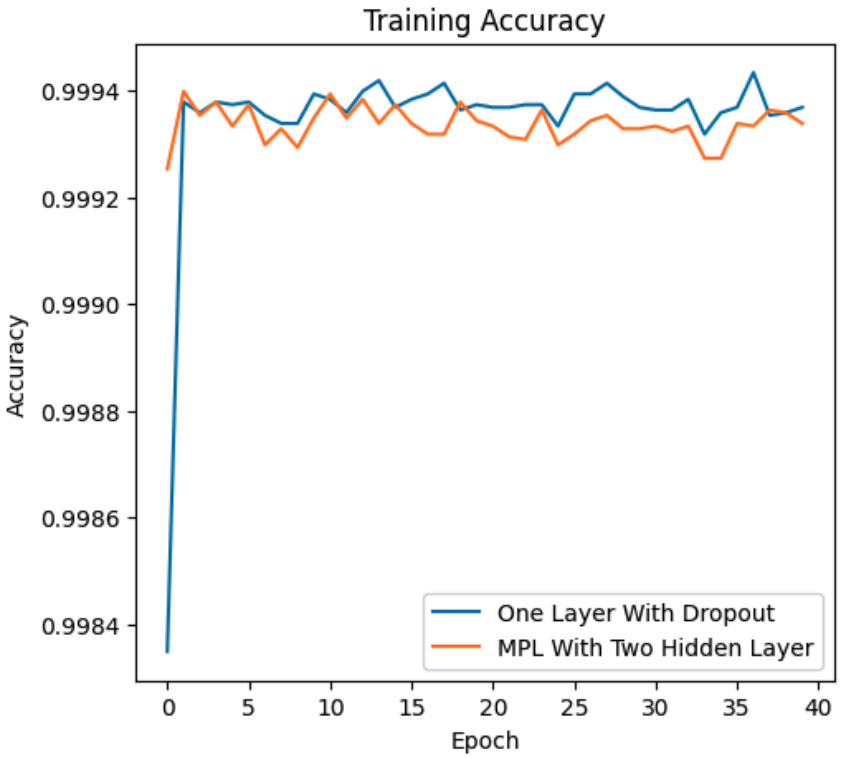
در این بخش یک شبکه ساختیم که حاوی دو لایه مخفی باشد :

- لایه مخفی اول : شامل ۱۲۸ نورون به همراه تابع ReLU
- لایه مخفی دوم : شامل ۶۴ نورون به همراه تابع ReLU
- به همراه dropout به میزان ۲۰ درصد برای هر یک از لایه ها

مقایسه مدل عمیق تر با مدل قسمت قبل (با یک لایه به همراه dropout به میزان ۳۰ درصد) :

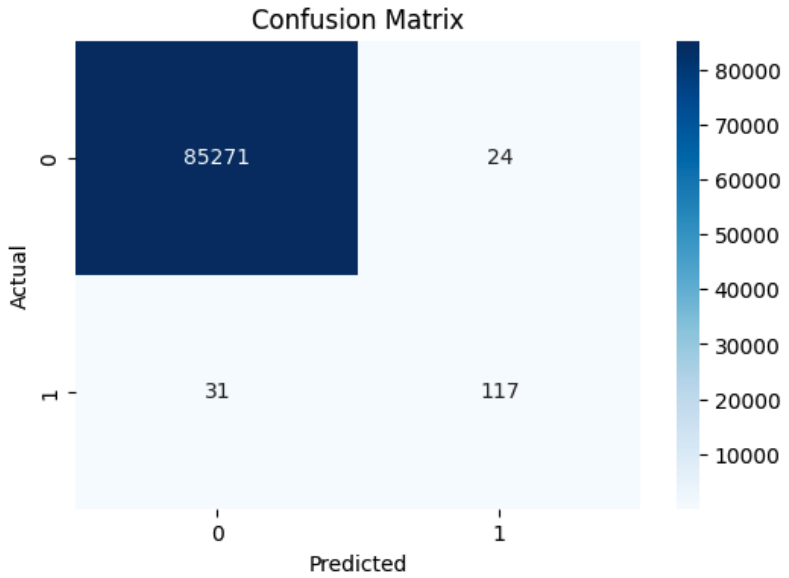


شکل ۱۲ : مقایسه میزان loss برای مدل عمیق و مدل ساده



شکل ۱۳ : مقایسه میزان accuracy برای مدل عمیق و مدل ساده

حال به سراغ ارزیابی مدل عمیق تر میرویم :



شکل ۱۴ :ماتریس آشفتگی برای مدل عمیقتر با دولایه

جدول ۳ : ارزیابی مدل عمیقتر

متریک ارزیابی	میزان
دقت (Accuracy)	0.9994
دقت کلاس مثبت (Precision)	0.8298
بازخوانی (Recall)	0.7905
F1 - Score	0.9512

دقت هردو مدل دقیقا به یک میزان است اما در بخص دقت کلاس های مثبت مدل اول عملکرد بهتری داشته است و در همچنین در متریک Recall هم به همین صورت است .

و در متریک F1-Score هم که ترکیبی از دقت کلاس مثبت و بازخوانی است هم به همین صورت بوده است .

علت این امر این بوده است که : اگر Precision یا Recall به شدت پایین باشد، F1-Score هم کاهش پیدا می‌کند.

۱-۵. تحلیل ماتریس آشفتگی و معیار های ارزیابی

- دقت (Accuracy) چه چیزی را اندازه میگیرد و چرا در داده های نامتوان معیار مناسبی نیست ؟
  - دقت نشان می‌دهد که چه نسبتی از پیش‌بینی‌های مدل به درستی انجام شده است.
  - $Accuracy \Rightarrow \frac{True\ Pos + True\ Neg}{Total\ Samples}$
  - *در داده‌های نامتوازن (زمانی که توزیع کلاس‌ها نامساوی است)، دقت معیار مناسبی نیست؛ زیرا مدل ممکن است با پیش‌بینی اکثریت نمونه‌ها به‌عنوان کلاس غالب، دقت بالایی نشان دهد اما عملکرد واقعی آن در تشخیص کلاس اقلیت ضعیف باشد.*

- معیار Precision چرا در تشخیص تقلب مهم است ؟
  - نشان می‌دهد که از بین تمام نمونه‌هایی که به‌عنوان مثبت پیش‌بینی شده‌اند، چند مورد واقعاً مثبت هستند.
  - $Precision \Rightarrow \frac{True\ Pos}{True\ Pos + False\ Pos}$
  - در تشخیص تقلب، Precision بالا به این معناست که تعداد پیش‌بینی‌های نادرست برای نمونه‌های غیرتقلبی کم است؛ یعنی هشدارهای کاذب کاهش می‌یابد.

- بازخوانی (Recall) مقدار بالای یا پایین آن چه معنایی دارد ؟
  - نشان می‌دهد که از بین تمام نمونه‌های مثبت واقعی، چه نسبتی به‌درستی شناسایی شده‌اند.
  - مقدار بالای Recall به این معناست که مدل توانایی بالایی در شناسایی نمونه‌های مثبت دارد (خطر از دست دادن نمونه‌های مثبت کم است).

- $Recall \Rightarrow \frac{True\ Pos}{True\ Pos + False\ Neg}$

- متریک F1-Score چرا بهتر از دقت معمولی است ؟
  - بهتر از دقت معمولی است، چون عدم تعادل بین Precision و Recall را متعادل می‌کند.
  - اگر Precision یا Recall به شدت پایین باشد، F1-Score هم کاهش پیدا می‌کند.

  - $F1 - Score \Rightarrow \frac{Precision \times Recall \times 2}{Precision + Recall}$

- منحنی ROC و مقدار AUC چه چیزی را نشان میدهد ؟
  - منحنی ROC توانایی مدل را در جداسازی کلاس‌ها نشان می‌دهد.
  - مقدار AUC هر چه به ۱ نزدیک‌تر باشد، مدل عملکرد بهتری دارد. که در واقع مساحت زیر نمودار ROC است .

- در ماتریس آشفتگی ، کدام کلاس بیشتر اشتباه طبقه بندی شده است :
  - برای مدل ساده یک لایه ای بدون Dropout :
    - ۴۰ عدد که واقعا تقلب بوده اند ، غیر تقلب شناخته شده اند .
    - ۳۰ عدد که تقلب نبوده اند ، به اشتباه تقلب شناسایی شده اند.
  - برای مدل ساده یک لایه ای به همراه Dropout :
    - ۳۲ عدد که واقعا تقلب بوده اند ، غیر تقلب شناخته شده اند .
    - ۱۹ عدد که تقلب نبوده اند ، به اشتباه تقلب شناسایی شده اند.
  - برای مدل عمیق تر با دولایه :
    - ۳۱ عدد که واقعا تقلب بوده اند ، غیر تقلب شناخته شده اند .
    - ۲۴ عدد که تقلب نبوده اند ، به اشتباه تقلب شناسایی شده اند.

بنابراین مدل ساده یک لایه ای به همراه Dropout بهترین عملکرد را داشته است و بعد از آن مدل عمیقتر ، و بدترین عملکرد در تشخیص اشتباه مربوط به مدل ساده یک لایه ای بدون Dropout است .

- بین دقت و بازخوانی چه تبادلی (Trade off) وجود دارد ؟
  - افزایش Precision معمولاً باعث کاهش Recall می‌شود (و بالعکس).
  - مثال :
    - اگر مدل حساسیت بالایی به موارد مثبت داشته باشد (Recall بالا) ، ممکن است که FP ها زیاد شوند که نشان دهنده این است که Precision کم میشود .
    - اگر مدل فقط نمونه‌های مطمئن را به‌عنوان مثبت شناسایی کند (Precision بالا)، ممکن است نمونه‌های مثبت واقعی دیده نشوند که یعنی Recall کم شده است .

۱-۶. جستجو ی بهترین هایپر پارامترها شبکه یک لایه مخفی با روش حریصانه (Grid Search)

هر دو روش Grid Search و Random Search درون فایل نوت بوک موجود است . روش Grid Search را به مدت ۸ ساعت اجرا کردیم و خروجی های بهینه شده به صورت جدول زیر بود :

جدول ۴ : یافتن بهترین هایپر پارامترها به روش grid search

Hidden size	dropout	l2_reg	batch size	AUC
64	0.2	0.001	16	0.97516853
64	0.2	0.001	32	0.97791021
64	0.2	0.001	64	Not Better
64	0.2	0.0001	16	Not Better
64	0.2	0.0001	32	0.97948265
64	0.2	0.0001	64	0.98646122
64	0.3	0.001	16	Not Better
64	0.3	0.001	32	Not Better
64	0.3	0.001	64	Not Better
64	0.3	0.0001	16	Not Better
64	0.3	0.0001	32	Not Better
64	0.3	0.0001	64	Not Better
64	0.4	0.001	16	Not Better
64	0.4	0.001	32	Not Better
64	0.4	0.001	64	Not Better
64	0.4	0.0001	16	Not Better
64	0.4	0.0001	32	Not Better
64	0.4	0.0001	64	Not Better
128	0.2	0.001	16	Not Better
128	0.2	0.001	32	Not Better
128	0.2	0.001	64	Not Better

128	0.2	0.0001	16	Not Better
128	0.2	0.0001	32	Not Better
128	0.2	0.0001	64	Not Better
128	0.3	0.001	16	Not Better
128	0.3	0.001	32	Not Better

بهترین مدلی که توانستیم تا به اینجای کار پیدا کنیم هایپر پارامتر های , hidden size = 64 batch_size =64 , l2_reg = 0.0001 , dropout = 0.2 را داشت که به ما خروجی AUC را در بهترین حالت 0.98646122 می داد.

۷-۱. مقایسه مدل MLP با مدل Logistic Regression

خروجی های مدل logistic regression به صورت زیر است :

مدل	Accuracy	F1	AUC
Logistic Regression	0.9993	0.7692	0.8378
One Layer With Dropout	0.9994	0.9803	0.9803
Deeper MLP with Two Hidden Layer	0.9994	0.9512	0.9852

همانطور که مشاهده میشود :

- مدل MLP به دلیل داشتن لایه‌های مخفی و توابع فعال‌سازی غیر خطی، عملکرد بهتری خواهد داشت.
- در چه شرایطی logistic regression بهتر عمل میکند :
  - وقتی داده‌ها رابطه‌ی **خطی** قوی با کلاس خروجی داشته باشند.
  - وقتی داده‌ها **ابعاد کمی** داشته باشند.

- وقتی **تعداد نمونه‌ها کم** باشد یا **داده‌ها نویزی** باشند، مدل ساده‌تر مانند رگرسیون لجستیک معمولا بهتر عمل می‌کند.

۸-۱. جمع بندی

- کدام مدل بهترین عملکرد را داشت و چرا؟
  - بهترین عملکرد برای مدلی با مشخصات زیر بود :
  - dropout = 0.2 , l2_reg = 0.0001 , batch_size =64 را داشت که به ما خروجی AUC را در بهترین حالت 0.98646122 می داد.
- افزودن لایه های بیشتر چه تأثیری روی عملکرد مدل داشت؟
  - لزوما باعث بیشتر شدن دقت نمیشد ، بهتر از برخی مدل ها بود ولی بهترین نبود .
- بهینه سازی پارامتر ها چگونه بر نتیجه مذل تاثیر گذاشت ؟
  - با استفاده از بهینه سازی هایپر پارامتر ها توانستیم به مدلی با AUC بیشتر دست پیدا کنیم .
- میزان خطای مدل چقدر بود و علت اصلی آن چه بود ؟
  - نمودار های loss در زمان آموزش و میزان خطا در زمان Prediction برای داده آزمایش در بخش های بالا به همراه نمودار قابل مشاهده هستند .
  - علت اصلی خطا نامتوازن بودن داده در دیتاست بود که باعث میشد مدل عملکرد بهتری در داده های بدون تقلب داشته باشد ، چون از جنس به همراه تقلب کمتر دیده بود .
- بر اساس جدول آشفتگی ، مدل بیشتر در کدام کلاس ها دچار اشتباه شده بود ؟
  - با توجه به شکل ۹ ، مدل در داده هایی که تقلب بوده اند اما تقلب نگرفته بود به میزان بیشتری از نظر تعدادی نسبت به بقیه دچار خطا شده بود .
  - ریفر به شکل ۹ و تحلیل آن .
- بین دقت و بازخوانی چه رابطه ای وجود دارد و چرا در تشخیص تقلب مهم است؟

- فرمول های آنها در بالا آورده شده است .
- در مسئله‌ی تشخیص تقلب، هدف اصلی شناسایی دقیق تراکنش‌های تقلبی است. پس بین دقت و بازخوانی، معیار مهم‌تر به نوع ریسک بستگی دارد:
  - اگر دقت (Precision) بالا باشد:
    - یعنی بیشتر تراکنش‌هایی که به عنوان تقلب شناسایی شده‌اند، واقعا تقلبی هستند.
    - اما ممکن است برخی تراکنش‌های تقلبی را از دست بدهیم (بازخوانی کم).
    - در مواردی که هزینه‌ی اشتباه مثبت (FP) بالاست (مثل بلوکه کردن حساب کاربر)، بالا بودن دقت اهمیت بیشتری دارد.
  - اگر بازخوانی (Recall) بالا باشد :
    - یعنی تقریبا همه‌ی تراکنش‌های تقلبی را شناسایی کرده‌ایم.
    - اما ممکن است برخی تراکنش‌های قانونی هم به اشتباه به عنوان تقلب شناسایی شوند (دقت پایین).
    - در مواردی که هزینه‌ی از دست دادن تقلب (FN) بالاست (مثل ضرر مالی مستقیم)، بالا بودن بازخوانی اهمیت بیشتری دارد.
- **در تشخیص تقلب، بازخوانی (Recall) مهم‌تر است :**
  - **چون ندیدن یک تراکنش تقلبی (FN) معمولا عواقب جدی‌تری دارد (ضرر مالی، آسیب به اعتبار شرکت) نسبت به بلوک کردن اشتباه یک تراکنش قانونی (FP).**
- مقایسه مدل ها : آیا مدل پیچیده تر عملکرد بهتری دارد یا فقط محاسبات را زیاد میکند ؟
  - طبق Grid Search ای که در جدول ۴ انجام دادیم ، لزوما مدل های پیچیده تر بهتر عمل نکرده اند .
- چه روش هایی برای بهبود عملکرد مدل پیشنهاد میشود ؟
  - بهبود دیتاست مهمترین کار و جدی ترین کاری است که میتوان برای بهتر شدن انجام داد :
    - پاکسازی داده‌ها (رفع داده‌های نادرست یا ناقص)
    - افزایش حجم داده (Data Augmentation)
    - تعادل‌بخشی کلاس‌ها (Class Balancing)
    - حذف داده‌های پرت (Outliers)
- مهم ترین چالش های تشخیص تقلب در داده های واقعی چیست ؟
  - در واقع به نظر من مهمترین چالش ها به صورت تیتروار به صورت زیر اند :

- عدم تعادل در کلاس‌ها (Class Imbalance)
- نرخ FALSE POS بالا در تشخیص و مسدود کردن الکی حساب کاربران عادی
- متقلب ها نباید از الگوریتم ها و Feature هایی که ما در مدل استفاده میکنیم آگاه شوند.
- دیتاست ها کم هستند . (در واقع متقلب ها کم اند)

- اگر مدل را دوباره طراحی کنید ، چه تغییری در آن ایجاد میکند ؟
  - ممکن است Optimizer ها متفاوتی رو تست کنم .
  - اجازه بدهم که کل Grid Search اجرا شود تا باقی هایپر پارامتر ها را هم ببینیم.
  - استفاده از Dropout ها با اندازه های بیشتر شاید .
  - دیتاست را با یک سری ترفند مانند augmentation سعی کنیم که balance تر کنیم .
  - روش های Early Stop را تست کنیم تا ببینیم بهتر عمل میکنند یا خیر .

پرسش ۲. طراحی شبکه عصبی چندلایه در مسئله رگرسیون مقاومت بتن

ابتدا دیتاست سوال را به کمک kagglehub بارگیری کردیم. این دیتاست طبق توضیحات صفحه خود آن مربوط به مقاومت بتن و پارامترهای تاثیرگذار در آن است.

۲-۱. بررسی آماری دادگان

خلاصه آماری دیتاست در زیر قابل مشاهده است.

	CementComponent	BlastFurnaceSlag	FlyAshComponent	WaterComponent	SuperplasticizerComponent	CoarseAggregateComponent	FineAggregateComponent	AgeInDays	Strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

جدول ۵. خلاصه آمای دیتاست مقاومت بتن

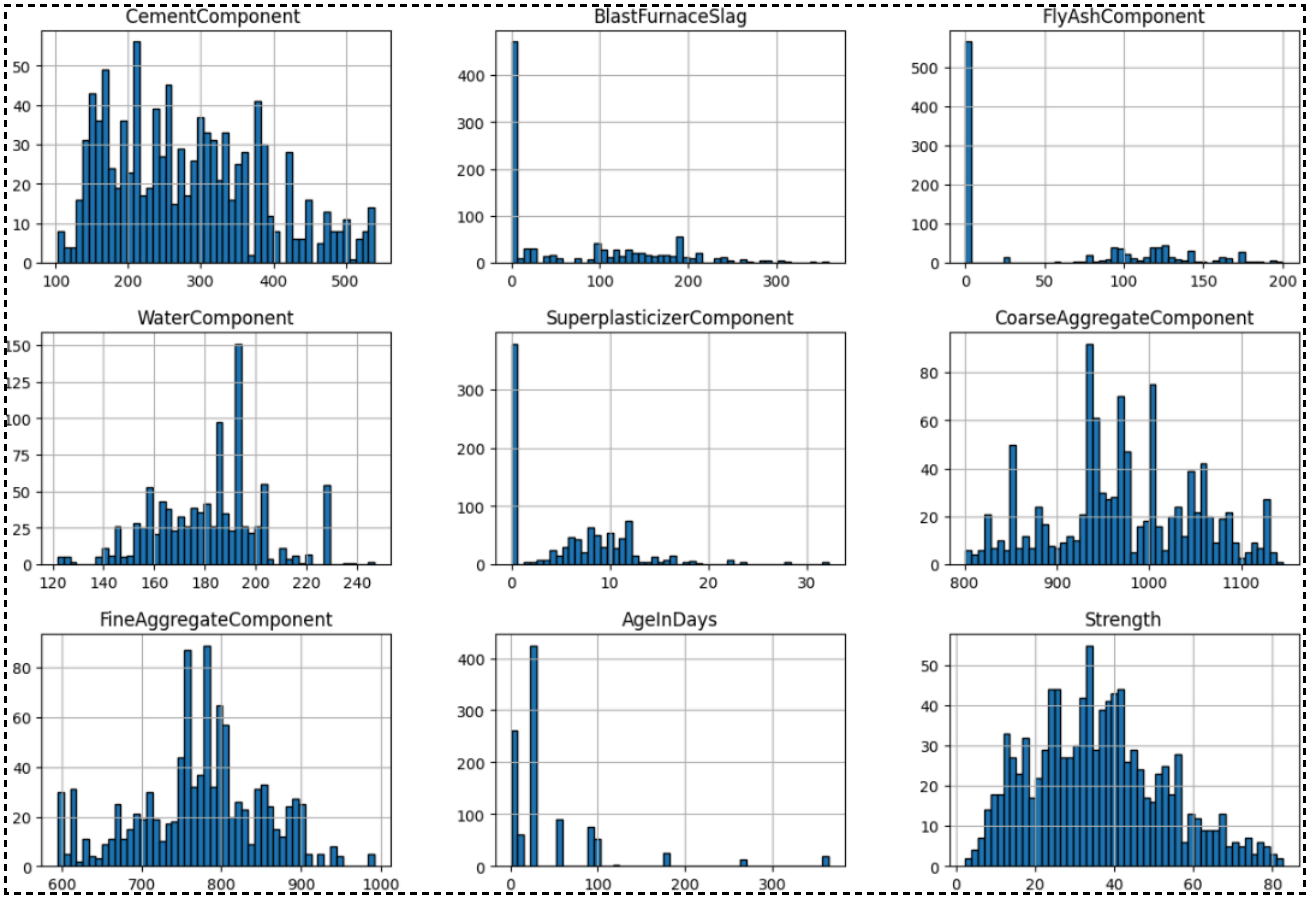
در جدول بالا میانگین، انحراف از معیار، کمترین و بیشترین مقدار برای هر ویژگی دیده می‌شود. همینطور مشخص است که دیتاست شامل ۱۰۳۰ رکورد است.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   CementComponent             1030 non-null   float64
 1   BlastFurnaceSlag            1030 non-null   float64
 2   FlyAshComponent             1030 non-null   float64
 3   WaterComponent              1030 non-null   float64
 4   SuperplasticizerComponent   1030 non-null   float64
 5   CoarseAggregateComponent    1030 non-null   float64
 6   FineAggregateComponent      1030 non-null   float64
 7   AgeInDays                   1030 non-null   int64
 8   Strength                    1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

جدول ۶. تایپ‌های دیتاست

همچنین مشاهده می‌شود که هیچ مقداری null در دیتاست وجود ندارد و به جز مقدار سن به روز (که صحیح است) بقیه ویژگی‌ها همگی از نوع اعشاری هستند. نبود هیچ مقدار null به ما کمک می‌کند که نیاز به پیش‌پردازش نداشته باشیم.

در ادامه هیستوگرام تمامی ویژگی‌ها آورده شده تا توزیع آن‌ها مشخص شود.



شکل ۱۵. هیستوگرام ویژگی‌ها

هیستوگرام بالا به ما نشان می‌دهد که ویژگی‌های BlastFurnaceSlag, FlyAshComponent, SuperplasticizerComponent, AgeInDays دارای توزیع مناسبی نبوده و بعضا Outlierهای زیادی نیز دارند.

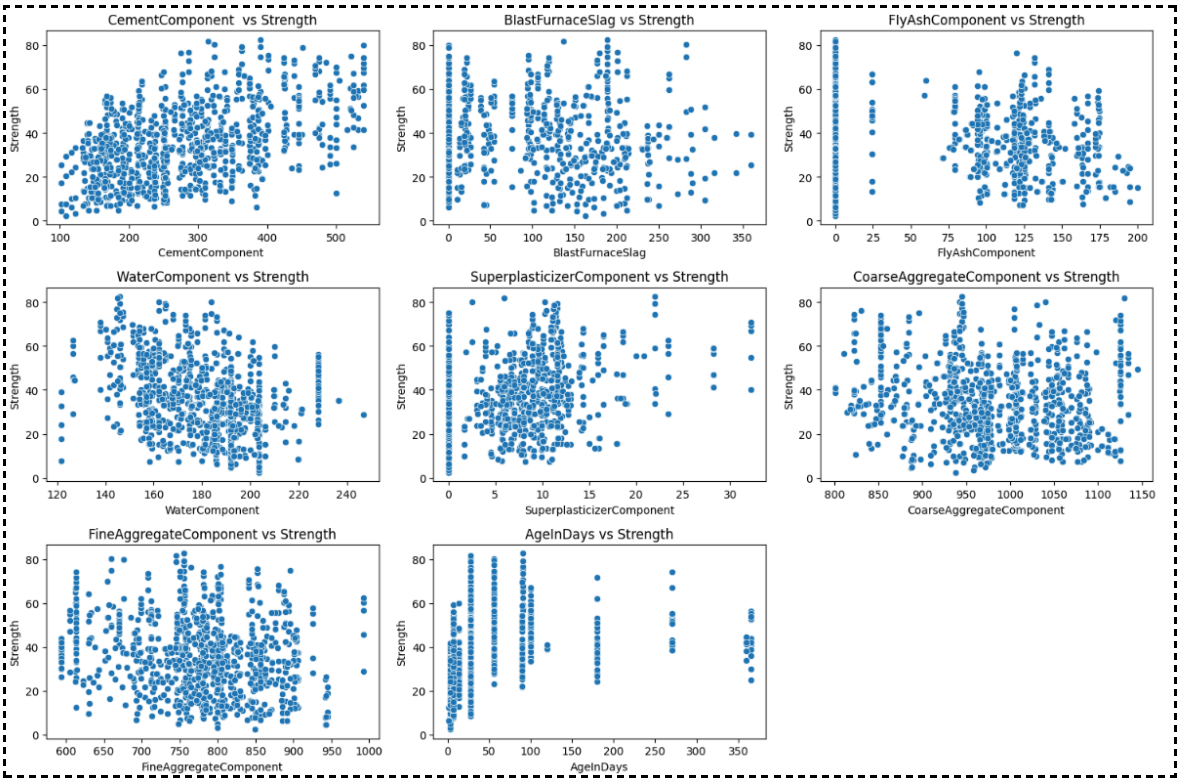
همچنین با توجه به scatter plot بالا دو ویژگی FlyAshComponent و SuperplasticizerComponent نسبت به مقدار هدف (Strength) توزیع چندان جالبی ندارند.

در صفحه بعد جدول همبستگی ویژگی‌ها نیز رسم شده است. بیشترین مقدار دیده شده نزدیک به منفی ۰.۶۶ است که به تنهایی مقدار همبستگی زیادی نیست.

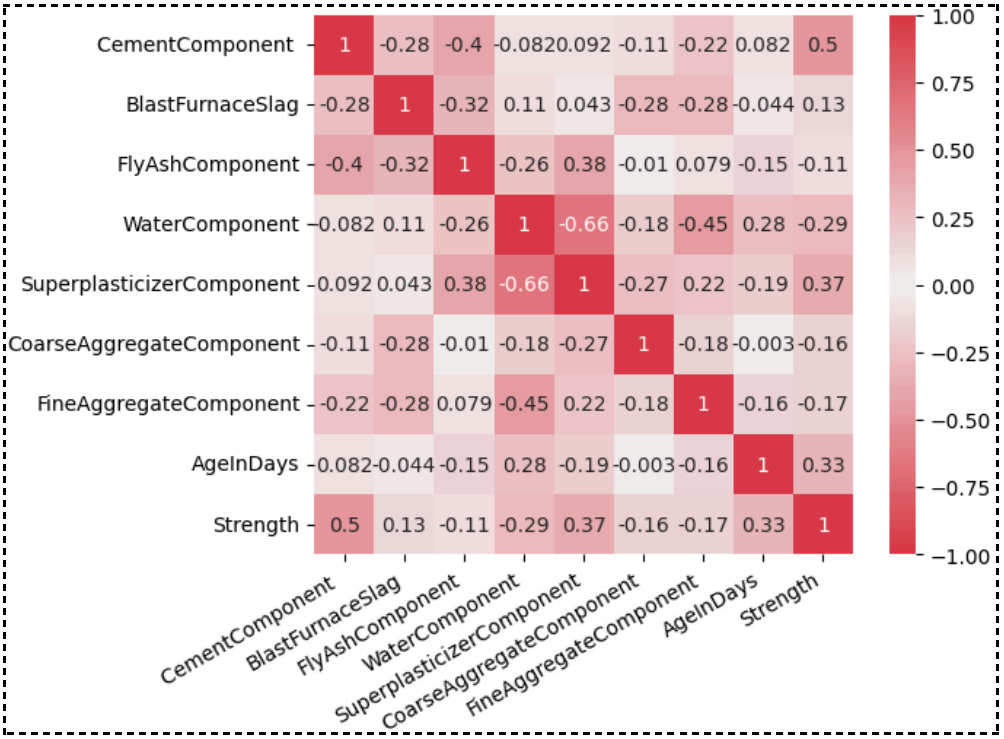
اما در نتیجه توزیع نامناسب و همبستگی همزمان، تصمیم بر این گرفتیم که از دو ویژگی FlyAshComponent و SuperplasticizerComponent صرف نظر کنیم.

این کار با کاهش پیچیدگی شبکه و کمتر شدن محاسبات شده و همچنین سریعتر همگرا شدن آن می‌شود.

قابل ذکر است که بدون حذف این دو نیز شبکه را ساختیم اما با حذف این دو ویژگی عملکرد بهتری مشاهده شد.



شکل ۱۶. وضعیت توزیع ویژگی‌ها نسبت به مقدار هدف



جدول ۷. ماتریس همبستگی ویژگی‌ها

ویژگی CementComponent بیشترین همبستگی را با مقاومت بتن دارد. (که قابل پیشبینی نیز بود چرا که میزان سیمان منطقا باید تاثیر زیادی داشته باشد)

به طور کلی همبستگی زیادی بین ویژگی‌ها وجود ندارد بجز مواردی که نزدیک به ۰.۶ می‌شوند.

صفحه قبل نیز توضیح داده شد که چرا می‌توان دو تا از ویژگی‌ها را با آمارهای بدست آمده از همبستگی و توزیع‌ها حذف نمود.

در صورت حذف ویژگی با همبستگی زیاد محاسبات را کاهش داده و شبکه را سریعتر می‌کنیم. در صورتی که ویژگی‌هایی با همبستگی کم را به جای یکدیگر استفاده کنیم اطلاعات زیادی را از دست می‌دهیم و دقت شبکه کاهش پیدا می‌کند.

۲-۲. پیاده‌سازی مدل MLP

در ابتدا لازم است که دیتا را به دو بخش آموزشی و تست تقسیم کنیم. در این مسئله ما از ۸۰ درصد دیتا برای آموزش و بقیه برای تست استفاده خواهیم کرد.

در ادامه به تعریف شبکه عصبی خواهیم پرداخت. دو شبکه عصبی با تعداد نورون متفاوت در لایه میانی پنهان (۱۶ و ۳۲) طراحی کردیم که لایه ورودی آن‌ها ۶ نورون (به تعداد ویژگی‌ها) و لایه خروجی ۱ نورون و بدون تابع فعال‌ساز است چرا که با مسئله رگرسیون طرف هستیم.

در شکل زیر تعریف یکی از شبکه‌ها قابل مشاهده است :

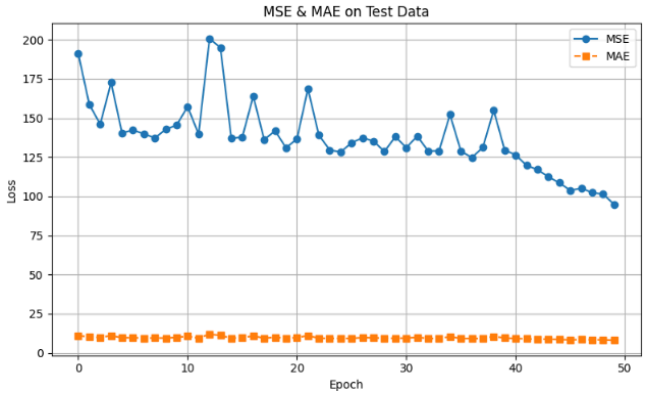
```
class MLP_16N(nn.Module):
    def __init__(self):
        super(MLP_16N, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(6, 16),
            nn.ReLU(),
            nn.Linear(16, 1)
        )

    def forward(self, x):
        return self.model(x)
```

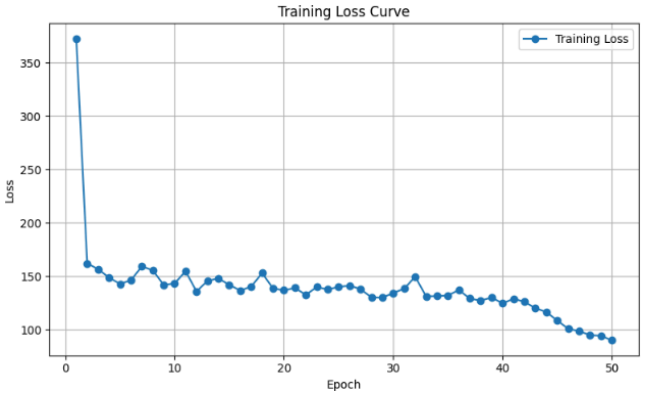
همچنین طبق خواسته سوال :

```
criterion = nn.MSELoss()  # MSE Loss for regression
optimizer = optim.Adam(mlp_16N.parameters(), lr=0.003)  # Adam optimizer

epochs = 50
```

وضعیت آموزش مدل با لایه پنهان **۱۶** نورونی :

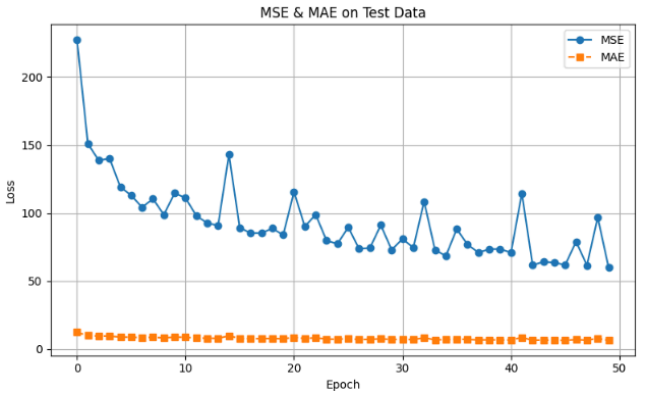
```
Epoch 5/50, Train Loss: 142.7984, Test MSE: 140.5018, Test MAE: 9.5861
Epoch 10/50, Train Loss: 143.2553, Test MSE: 145.5184, Test MAE: 9.8305
Epoch 15/50, Train Loss: 141.9433, Test MSE: 137.0133, Test MAE: 9.3379
Epoch 20/50, Train Loss: 136.7714, Test MSE: 131.0074, Test MAE: 9.3162
Epoch 25/50, Train Loss: 140.0031, Test MSE: 128.3009, Test MAE: 9.2232
Epoch 30/50, Train Loss: 134.3259, Test MSE: 138.0769, Test MAE: 9.2845
Epoch 35/50, Train Loss: 132.0440, Test MSE: 152.1862, Test MAE: 10.3740
Epoch 40/50, Train Loss: 124.5879, Test MSE: 129.5465, Test MAE: 9.4641
Epoch 45/50, Train Loss: 108.4976, Test MSE: 108.6030, Test MAE: 8.4636
Epoch 50/50, Train Loss: 89.9734, Test MSE: 94.9095, Test MAE: 7.9628
Training complete!
```

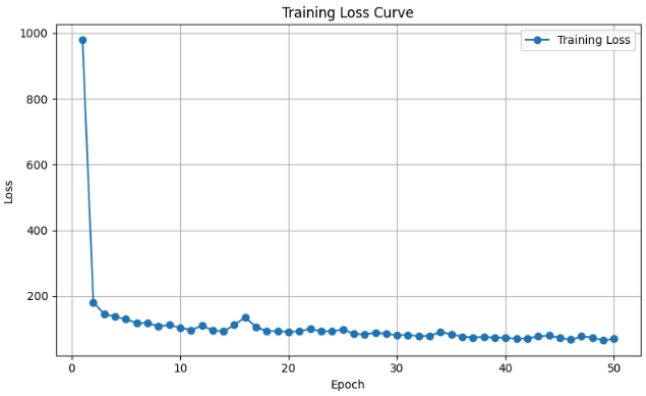




وضعیت آموزش مدل با لایه پنهان **۳۲** نورونی :

```
Epoch 5/50, Train Loss: 128.5297, Test MSE: 119.0732, Test MAE: 8.8472
Epoch 10/50, Train Loss: 103.4404, Test MSE: 114.6081, Test MAE: 8.7820
Epoch 15/50, Train Loss: 111.4962, Test MSE: 142.9538, Test MAE: 9.7464
Epoch 20/50, Train Loss: 90.0429, Test MSE: 83.8539, Test MAE: 7.5456
Epoch 25/50, Train Loss: 97.8331, Test MSE: 77.3217, Test MAE: 7.2176
Epoch 30/50, Train Loss: 80.5706, Test MSE: 72.9463, Test MAE: 7.0823
Epoch 35/50, Train Loss: 82.5370, Test MSE: 68.4982, Test MAE: 6.9156
Epoch 40/50, Train Loss: 72.4889, Test MSE: 73.5329, Test MAE: 6.7406
Epoch 45/50, Train Loss: 71.5920, Test MSE: 63.5032, Test MAE: 6.4001
Epoch 50/50, Train Loss: 69.2294, Test MSE: 60.0187, Test MAE: 6.3308
Training complete!
```





از این قسمت به بعد با توجه به عملکرد بهتر شبکه عصبی با **۳۲ نورون** در لایه پنهان از آن استفاده خواهیم کرد.

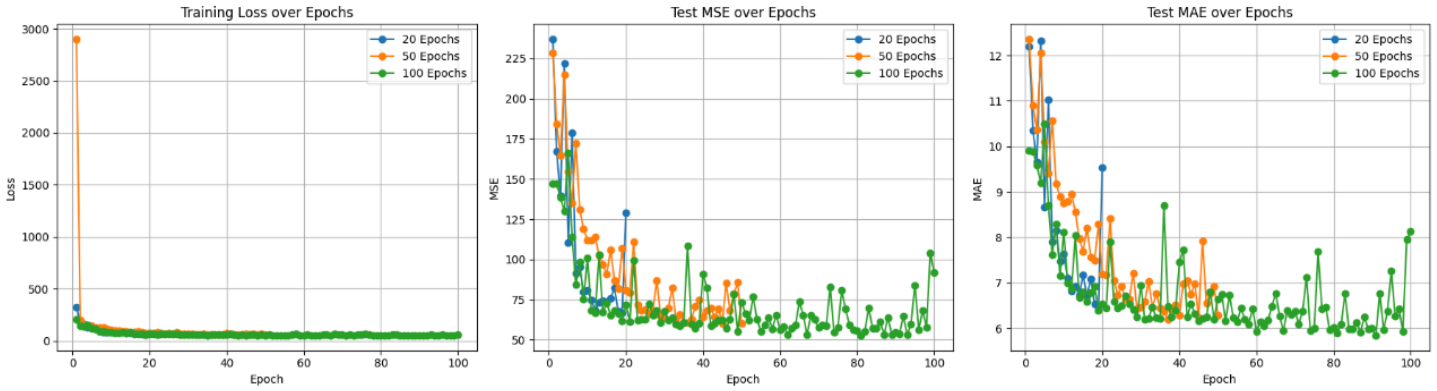
۳-۲. بررسی تغییرات تنظیمات مدل

تاثیر تعداد ایپاک‌ها

سه مدل جدید ساختیم و هرکدام را به ترتیب ۲۰، ۵۰ و ۱۰۰ ایپاک آموزش دادیم که به شکل زیر وضعیت آن‌ها در طول آموزش مشخص است :

```
20Ep - Epoch 5/20, Train Loss: 136.4293, Test MSE: 110.3942, Test MAE: 8.6674
20Ep - Epoch 10/20, Train Loss: 80.0174, Test MSE: 80.8047, Test MAE: 7.6377
20Ep - Epoch 15/20, Train Loss: 77.5924, Test MSE: 73.5344, Test MAE: 7.1741
20Ep - Epoch 20/20, Train Loss: 78.6342, Test MSE: 129.2684, Test MAE: 9.5329
Training complete for 20 epochs!

50Ep - Epoch 5/50, Train Loss: 142.3260, Test MSE: 154.6983, Test MAE: 10.0967
50Ep - Epoch 10/50, Train Loss: 109.1759, Test MSE: 111.9752, Test MAE: 8.7538
50Ep - Epoch 15/50, Train Loss: 83.2691, Test MSE: 90.7728, Test MAE: 7.6949
50Ep - Epoch 20/50, Train Loss: 77.3970, Test MSE: 80.6056, Test MAE: 7.1962
50Ep - Epoch 25/50, Train Loss: 73.2192, Test MSE: 68.8880, Test MAE: 6.9205
50Ep - Epoch 30/50, Train Loss: 70.1874, Test MSE: 63.7480, Test MAE: 6.4550
50Ep - Epoch 35/50, Train Loss: 60.9584, Test MSE: 60.9677, Test MAE: 6.4367
50Ep - Epoch 40/50, Train Loss: 76.1235, Test MSE: 64.2375, Test MAE: 6.2870
50Ep - Epoch 45/50, Train Loss: 64.9817, Test MSE: 60.1544, Test MAE: 6.2370
50Ep - Epoch 50/50, Train Loss: 62.6847, Test MSE: 60.0157, Test MAE: 6.2903
Training complete for 50 epochs!

100Ep - Epoch 5/100, Train Loss: 120.6263, Test MSE: 166.3810, Test MAE: 10.4953
100Ep - Epoch 10/100, Train Loss: 81.8022, Test MSE: 100.8233, Test MAE: 8.1173
100Ep - Epoch 15/100, Train Loss: 85.6358, Test MSE: 73.0062, Test MAE: 6.7972
100Ep - Epoch 20/100, Train Loss: 65.4225, Test MSE: 71.8446, Test MAE: 6.5291
100Ep - Epoch 25/100, Train Loss: 65.7416, Test MSE: 62.7859, Test MAE: 6.4964
100Ep - Epoch 30/100, Train Loss: 58.5374, Test MSE: 67.7122, Test MAE: 6.9468
100Ep - Epoch 35/100, Train Loss: 55.5736, Test MSE: 60.9479, Test MAE: 6.2151
100Ep - Epoch 40/100, Train Loss: 62.8772, Test MSE: 91.0956, Test MAE: 7.4629
100Ep - Epoch 45/100, Train Loss: 55.6829, Test MSE: 62.1935, Test MAE: 6.1663
100Ep - Epoch 50/100, Train Loss: 55.6800, Test MSE: 73.3771, Test MAE: 6.6451
100Ep - Epoch 55/100, Train Loss: 55.1025, Test MSE: 55.3194, Test MAE: 6.1480
100Ep - Epoch 60/100, Train Loss: 54.3239, Test MSE: 56.3266, Test MAE: 5.9366
100Ep - Epoch 65/100, Train Loss: 58.8416, Test MSE: 73.6319, Test MAE: 6.7698
100Ep - Epoch 70/100, Train Loss: 60.9078, Test MSE: 57.4727, Test MAE: 6.3696
100Ep - Epoch 75/100, Train Loss: 56.1305, Test MSE: 57.8574, Test MAE: 5.9989
100Ep - Epoch 80/100, Train Loss: 55.8853, Test MSE: 55.5261, Test MAE: 6.0113
100Ep - Epoch 85/100, Train Loss: 53.5587, Test MSE: 56.9905, Test MAE: 5.9778
100Ep - Epoch 90/100, Train Loss: 54.9352, Test MSE: 54.5082, Test MAE: 5.9980
100Ep - Epoch 95/100, Train Loss: 55.2232, Test MSE: 84.0310, Test MAE: 7.2580
100Ep - Epoch 100/100, Train Loss: 56.9957, Test MSE: 91.7138, Test MAE: 8.1246
Training complete for 100 epochs!
```

همچنین نمودار وضعیت هر سه پارامتر در برای هر سه مدل در صفحه بعد دیده می‌شود.



شکل ۱۷. روند MSE, MAE و Loss در حین تمرین سه مدل

همانطور که مشاهده می‌شود از یک میزانی بیشتر تعداد ایپاک تاثیر مثبتی نداره و حتی ممکن است در اعداد خیلی بالا تاثیر منفی گذاشته و باعث overfitting شود که دقت ما را روی داده تست کاهش خواهد داد! اما با کاهش نرخ یادگیری و افزایش ایپاک به طور همزمان ممکن است بتوانیم به دقت بالاتری برسیم.

مقایسه توابع هزینه

Mean Squared Error - میانگین مربعات خطا

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

- خطای هر نمونه را به توان ۲ می‌رساند، بنابراین خطاهای بزرگ‌تر تأثیر بیشتری دارند.
- نسبت به نویز و مقادیر پرت **حساس** است.

Mean Absolute Error - میانگین قدر مطلق خطا

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}.$$

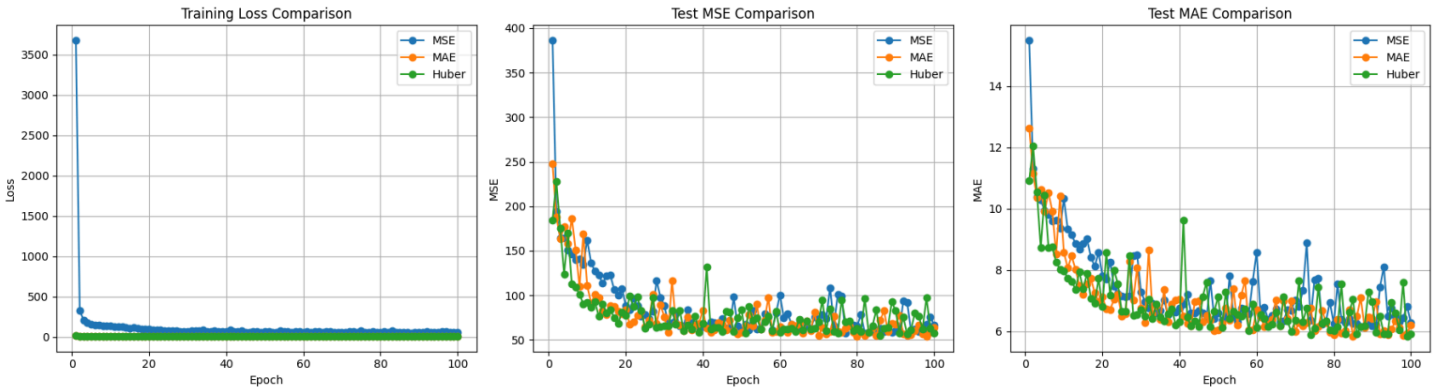
- فقط از قدر مطلق خطاها استفاده می‌کند، بنابراین تأثیر مقادیر پرت کمتر از MSE است.
- نسبت به MSE **مقاوم‌تر در برابر نویز** است اما مشتق‌پذیری کمتری دارد.

Huber Loss

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$

- ترکیبی از MSE و MAE است، برای کاهش تاثیر نقاط پرت.
- مقدار **δ** تعیین می‌کند که چه زمانی از MSE به MAE سوئیچ کند.
- در مناطق با خطای کوچک مانند MSE رفتار می‌کند و در مناطق با خطای بزرگ مانند MAE عمل می‌کند.

در مقایسه برای این مورد نرخ یادگیری در هر سه مدل ۰.۰۰۵ و تعداد ایپاک ۱۰۰ در نظر گرفته شده است.



شکل ۱۸. روند وضعیت مدل‌ها حین آموزش با سه تابع هزینه متفاوت

عملکرد دقیق‌تر مدل‌ها در خروجی زیر قابل مشاهده است (نیازمند زوم کردن)

MSE - Epoch 5/100, Train Loss: 160.0818, Test MSE: 150.6971, Test MAE: 9.9413
MSE - Epoch 10/100, Train Loss: 135.2760, Test MSE: 162.0534, Test MAE: 10.3253
MSE - Epoch 15/100, Train Loss: 112.7482, Test MSE: 121.6993, Test MAE: 8.8741
MSE - Epoch 20/100, Train Loss: 97.1858, Test MSE: 88.3403, Test MAE: 7.8235
MSE - Epoch 25/100, Train Loss: 77.0969, Test MSE: 78.2142, Test MAE: 7.1576
MSE - Epoch 30/100, Train Loss: 68.2126, Test MSE: 88.6133, Test MAE: 7.2960
MSE - Epoch 35/100, Train Loss: 67.0863, Test MSE: 65.2604, Test MAE: 6.6897
MSE - Epoch 40/100, Train Loss: 64.0134, Test MSE: 65.1706, Test MAE: 6.7643
MSE - Epoch 45/100, Train Loss: 62.7434, Test MSE: 74.1864, Test MAE: 6.6915
MSE - Epoch 50/100, Train Loss: 66.9801, Test MSE: 61.1125, Test MAE: 6.6107
MSE - Epoch 55/100, Train Loss: 68.1333, Test MSE: 62.3110, Test MAE: 6.3013
MSE - Epoch 60/100, Train Loss: 57.8843, Test MSE: 100.4521, Test MAE: 8.5821
MSE - Epoch 65/100, Train Loss: 61.8147, Test MSE: 62.1207, Test MAE: 6.7289
MSE - Epoch 70/100, Train Loss: 59.6008, Test MSE: 67.3135, Test MAE: 7.0185
MSE - Epoch 75/100, Train Loss: 80.6077, Test MSE: 100.9392, Test MAE: 7.6678
MSE - Epoch 80/100, Train Loss: 61.4664, Test MSE: 65.4079, Test MAE: 6.2496
MSE - Epoch 85/100, Train Loss: 56.5371, Test MSE: 61.4306, Test MAE: 6.2537
MSE - Epoch 90/100, Train Loss: 57.2810, Test MSE: 60.6010, Test MAE: 6.1686
MSE - Epoch 95/100, Train Loss: 62.4083, Test MSE: 62.0383, Test MAE: 6.7562
MSE - Epoch 100/100, Train Loss: 56.0576, Test MSE: 66.1553, Test MAE: 6.2928
Training complete for MSE model!

MAE - Epoch 5/100, Train Loss: 9.4848, Test MSE: 157.6740, Test MAE: 9.9159
MAE - Epoch 10/100, Train Loss: 8.9102, Test MSE: 110.7440, Test MAE: 8.5736
MAE - Epoch 15/100, Train Loss: 7.8055, Test MSE: 78.1697, Test MAE: 7.2138
MAE - Epoch 20/100, Train Loss: 6.9305, Test MSE: 83.4247, Test MAE: 7.1273
MAE - Epoch 25/100, Train Loss: 6.3437, Test MSE: 64.4661, Test MAE: 6.4849
MAE - Epoch 30/100, Train Loss: 7.0901, Test MSE: 75.7835, Test MAE: 6.7628
MAE - Epoch 35/100, Train Loss: 6.3300, Test MSE: 60.2418, Test MAE: 6.3815
MAE - Epoch 40/100, Train Loss: 6.0846, Test MSE: 82.9976, Test MAE: 7.0606
MAE - Epoch 45/100, Train Loss: 6.4926, Test MSE: 67.7172, Test MAE: 7.0029
MAE - Epoch 50/100, Train Loss: 6.0875, Test MSE: 57.9493, Test MAE: 6.0393
MAE - Epoch 55/100, Train Loss: 5.9446, Test MSE: 61.8271, Test MAE: 6.2066
MAE - Epoch 60/100, Train Loss: 5.9416, Test MSE: 64.8336, Test MAE: 6.7079
MAE - Epoch 65/100, Train Loss: 6.1321, Test MSE: 68.5129, Test MAE: 7.0356
MAE - Epoch 70/100, Train Loss: 6.0541, Test MSE: 54.4146, Test MAE: 5.9986
MAE - Epoch 75/100, Train Loss: 6.1983, Test MSE: 59.4025, Test MAE: 6.0295
MAE - Epoch 80/100, Train Loss: 5.8118, Test MSE: 53.5550, Test MAE: 5.8886
MAE - Epoch 85/100, Train Loss: 5.8914, Test MSE: 54.7910, Test MAE: 5.8298
MAE - Epoch 90/100, Train Loss: 5.7401, Test MSE: 67.4092, Test MAE: 6.4298
MAE - Epoch 95/100, Train Loss: 5.6400, Test MSE: 59.7930, Test MAE: 6.0724
MAE - Epoch 100/100, Train Loss: 5.6024, Test MSE: 64.1207, Test MAE: 6.2128
Training complete for MAE model!

Huber - Epoch 5/100, Train Loss: 8.2332, Test MSE: 169.6793, Test MAE: 10.4354
Huber - Epoch 10/100, Train Loss: 7.4588, Test MSE: 91.9407, Test MAE: 7.9735
Huber - Epoch 15/100, Train Loss: 6.4374, Test MSE: 80.0655, Test MAE: 7.3819
Huber - Epoch 20/100, Train Loss: 6.6240, Test MSE: 77.0319, Test MAE: 6.8217
Huber - Epoch 25/100, Train Loss: 6.0581, Test MSE: 63.1166, Test MAE: 6.6570
Huber - Epoch 30/100, Train Loss: 6.2785, Test MSE: 65.1933, Test MAE: 6.7040
Huber - Epoch 35/100, Train Loss: 5.8127, Test MSE: 60.0624, Test MAE: 6.4627
Huber - Epoch 40/100, Train Loss: 5.6112, Test MSE: 68.4889, Test MAE: 6.3206
Huber - Epoch 45/100, Train Loss: 5.5995, Test MSE: 58.8638, Test MAE: 6.1433
Huber - Epoch 50/100, Train Loss: 5.5378, Test MSE: 83.8177, Test MAE: 7.1255
Huber - Epoch 55/100, Train Loss: 5.5699, Test MSE: 62.3142, Test MAE: 6.5410
Huber - Epoch 60/100, Train Loss: 6.0967, Test MSE: 58.0347, Test MAE: 6.1261
Huber - Epoch 65/100, Train Loss: 5.5684, Test MSE: 72.6203, Test MAE: 6.6366
Huber - Epoch 70/100, Train Loss: 5.1718, Test MSE: 70.5643, Test MAE: 6.3586
Huber - Epoch 75/100, Train Loss: 5.4235, Test MSE: 57.8983, Test MAE: 6.1044
Huber - Epoch 80/100, Train Loss: 5.6112, Test MSE: 62.7782, Test MAE: 6.0179
Huber - Epoch 85/100, Train Loss: 5.3651, Test MSE: 83.4829, Test MAE: 7.0435
Huber - Epoch 90/100, Train Loss: 5.3700, Test MSE: 82.8662, Test MAE: 6.9642
Huber - Epoch 95/100, Train Loss: 5.6002, Test MSE: 80.2660, Test MAE: 6.9338
Huber - Epoch 100/100, Train Loss: 4.9949, Test MSE: 57.2385, Test MAE: 5.9136
Training complete for Huber model!

در مجموع مدل‌ها اختلاف زیادی نداشته‌اند اما دو چیز مشهود است :

1. در نهایت تابع هزینه Hubor به دقت بهتری دست یافته است.
2. سریعترین همگرایی با تابع هزینه Hubor اتفاق افتاده است.

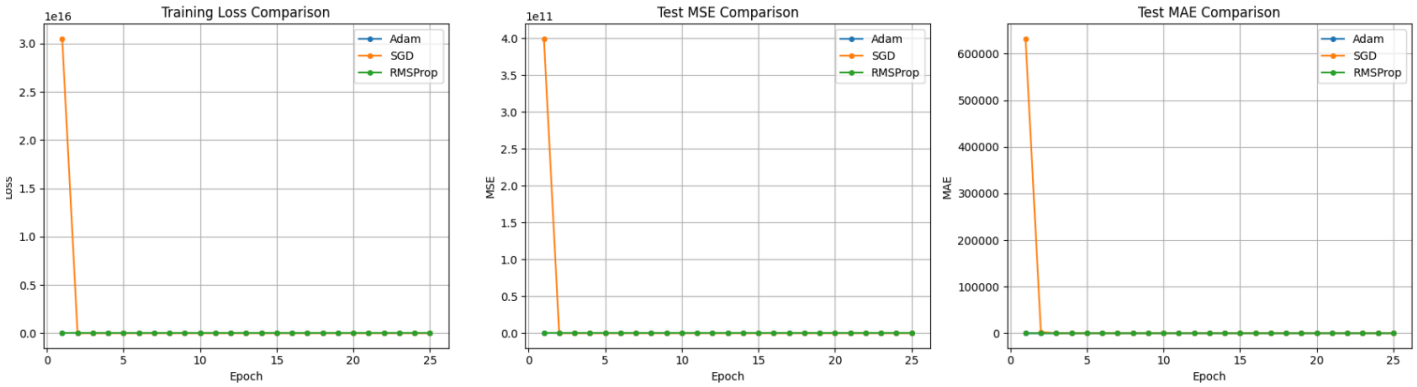
مقایسه توابع بهینه ساز

در این بخش برای تعداد ایپاک ۲۵ در نظر گرفتیم (تا تاثیر سرعت همگرایی مشخص‌تر باشد) همچنین برای RMSprop و Adam نرخ یادگیری ۰.۰۱ در نظر گرفته شده و برای SGD میزان ۰.۰۵ دلیل این کار اما سرعت پایین همگرایی SGD برمی‌گردد که در صورتی که SGD هم نرخ یادگیری پایین‌تری می‌داشت اصلا مقادیر معنی‌داری حتی در ایپاک‌های بالا هم دریافت نمی‌کردیم.

وضعیت مدل‌ها حین آموزش به شکل زیر بود :

```
Adam - Epoch 5/25, Train Loss: 137.1201, Test MSE: 145.8494, Test MAE: 10.0747
Adam - Epoch 10/25, Train Loss: 107.1240, Test MSE: 107.1974, Test MAE: 8.4258
Adam - Epoch 15/25, Train Loss: 75.2796, Test MSE: 128.6903, Test MAE: 9.1120
Adam - Epoch 20/25, Train Loss: 81.0709, Test MSE: 91.5034, Test MAE: 7.5270
Adam - Epoch 25/25, Train Loss: 79.8412, Test MSE: 62.8961, Test MAE: 6.4080
Training complete for Adam optimizer!

SGD - Epoch 5/25, Train Loss: 283.8362, Test MSE: 257.9806, Test MAE: 13.0590
SGD - Epoch 10/25, Train Loss: 286.6919, Test MSE: 258.1025, Test MAE: 13.0647
SGD - Epoch 15/25, Train Loss: 286.1188, Test MSE: 257.9671, Test MAE: 13.0292
SGD - Epoch 20/25, Train Loss: 285.1604, Test MSE: 259.6964, Test MAE: 13.0489
SGD - Epoch 25/25, Train Loss: 286.4728, Test MSE: 258.4207, Test MAE: 13.0795
Training complete for SGD optimizer!

RMSProp - Epoch 5/25, Train Loss: 557.8547, Test MSE: 699.7944, Test MAE: 23.3355
RMSProp - Epoch 10/25, Train Loss: 292.2838, Test MSE: 318.3895, Test MAE: 15.0716
RMSProp - Epoch 15/25, Train Loss: 206.3807, Test MSE: 118.6976, Test MAE: 9.1672
RMSProp - Epoch 20/25, Train Loss: 143.3705, Test MSE: 92.4553, Test MAE: 8.1048
RMSProp - Epoch 25/25, Train Loss: 136.0910, Test MSE: 103.6188, Test MAE: 8.4356
Training complete for RMSProp optimizer!
```

و همچنین روند مقادیر :



شکل ۱۹. روند وضعیت مدل‌ها با توابع بهینه‌ساز متفاوت در حین آموزش

همانطور که در نمودار بالا مشاهده می‌شود با وجود نرخ یادگیری بالاتر SGD همچنان مقادیر اولیه تابع هزینه آن اعدادی پرت به حساب می‌آید و شکل نمودار را خراب کرده است.

اما از جزییات بالاتر می‌توان به نتایج زیر رسید :

1. تابع بهینه‌سازی Adam سریعتر از بقیه است و در ایپاک کمتری به دقت مناسب می‌رسد.
2. نه تنها کندی SGD مشکل بزرگیست بلکه در همین ۲۵ ایپاک نیز مشاهده می‌شود که در local minima دچار مشکل می‌شود.

۲-۴. جمعبندی

همانطور که بالاتر هم اشاره شده بیشترین تاثیر روی مقاومت بتن مربوط به ویژگی CementComponent (مقدار سیمان) است که منطقی نیز به نظر می‌آید چرا که میزان سیمان می‌تواند روی استحکام بتن تاثیر زیادی بگذارد.

بهترین تنظیماتی که ما به آن دست یافتیم استفاده از Adam به عنوان تابع بهینه‌ساز، استفاده از نرخ یادگیری ۰.۰۱ با حدودا ۵۰ ایپاک بود.

مشخصا افزایش ایپاک به تنهایی لزوما باعث بهبود نمی‌شد و بعضا باعث overfitting می‌شود. در صورت کاهش نرخ یادگیری و افزایش ایپاک همزمان ممکن است بتوانیم به دقت بهتری برسیم.

بین توابع هزینه Huber Loss بهتر از بقیه عمل کرد چرا که قابل سوییچ از دو حالت برای خطاهای بزرگ و کوچک انعطاف پذیری بالایی را فراهم می‌کرد.

بین بهینه‌سازها Adam سریعتر از بقیه همگرا شد. توضیحات بیشتر در بخش مربوطه آورده شده است.

مهمترین **چالش‌های مدل‌سازی رگرسیون با شبکه عصبی** عبارتند از :

1. انتخاب هایپر پارامترهای مناسب برای مدل
2. انتخاب تابع هزینه مناسب
3. تصمیم گیری نسبت به outlier ها در داده (انتخاب تابع هزینه مناسب می‌تواند اثر آن‌ها را کم کند)
4. شبکه عصبی با داده‌های بیش از حد زیاد می‌توانند به Overfitting ختم شود.

پرسش ۳. پیاده‌سازی Adaline برای دیتاست IRIS

۳-۱. الگوریتم‌های Adaline و Madaline

Adaline : شبکه‌ای با تنها یک لایه (در واقع یک لایه ورودی و یک راس خروجی) که معماری آن شباهت زیادی به Perceptron داشته اما از قواعد متفاوتی برای یادگیری بهره می‌برد. تفاوت یادگیری را می‌شود به طور خلاصه در این توضیح داد که تغییرات وزن‌ها متناسب با اختلاف پیش‌بینی شبکه از مقدار واقعی داده آموزشی است. همچنین تابع فعال‌ساز Threshold نداشته و به صورت خطی کار می‌کند.

Madaline : شبکه‌ای که از ترکیب چندین Adaline به همراه یک نورون خروجی AND یا OR تشکل شده است. (Multi-Adaline) توانایی جداسازی الگوهایی با پیچیدگی‌هایی بیشتر نسبت به Adaline تنها را دارد.

Madaline *vs.* **MLP** : تفاوت‌های اصلی این دو در امکان وجود لایه‌های مخفی در MLP است. Madaline همچنان تعدادی Adaline را به صورت موازی در کنار هم قرار داده و خروجی را به یک OR یا AND می‌دهد. وجود لایه‌های مخفی به MLP اجازه می‌دهد پیچیدگی‌های بیشتری را در فرآیند یادگیری تشخیص داده و مدیریت کند.

در موارد جزئی دیگری مثل الگوریتم یادگیری و کاربردهای مهم نیست تفاوت دارند که در بالا توضیح داده شده است.

۳-۲. کار با دیتاست Iris

ابتدا دیتاسب را به شیوه گفته شده ذخیره می‌کنیم. سپس با حذف کلاس Virginica و ویژگی‌های sepal length و sepal width به دیتاستی میرسیم که ۱۰۰ رکورد که هر کدام دو ویژگی دارند می‌رسیم. سپس داده‌های را نرمالایز می‌کنیم (اسکیل بین ۱ و ۰) و ۳۰ درصد آن‌ها را برای تست و بقیه را برای آموزش نگه میداریم. در تصویر زیر توصیفی از دیتاست بعد از تمامی مراحل بالا قابل مشاهده است :

	petal length (cm)	petal width (cm)
count	100.000000	100.000000
mean	0.453902	0.403529
std	0.353548	0.332443
min	0.000000	0.000000
25%	0.121951	0.058824
50%	0.353659	0.411765
75%	0.810976	0.705882
max	1.000000	1.000000

```
Training features: (70, 2)
Test features: (30, 2)
Training target: (70,)
Test target: (30,)
```

شکل ۲۰. ابعاد و اطلاعات Iris بعد از تغییرات لازمه

۳-۳. پیاده‌سازی و آموزش Adaline

در این مرحله یک تابع برای آموزش یک مدل adaline با ورودی و خروجی‌های خواسته شده نوشتیم که تعریف آن به طور کامل در نوت‌بوک همراه گزارش موجود است.

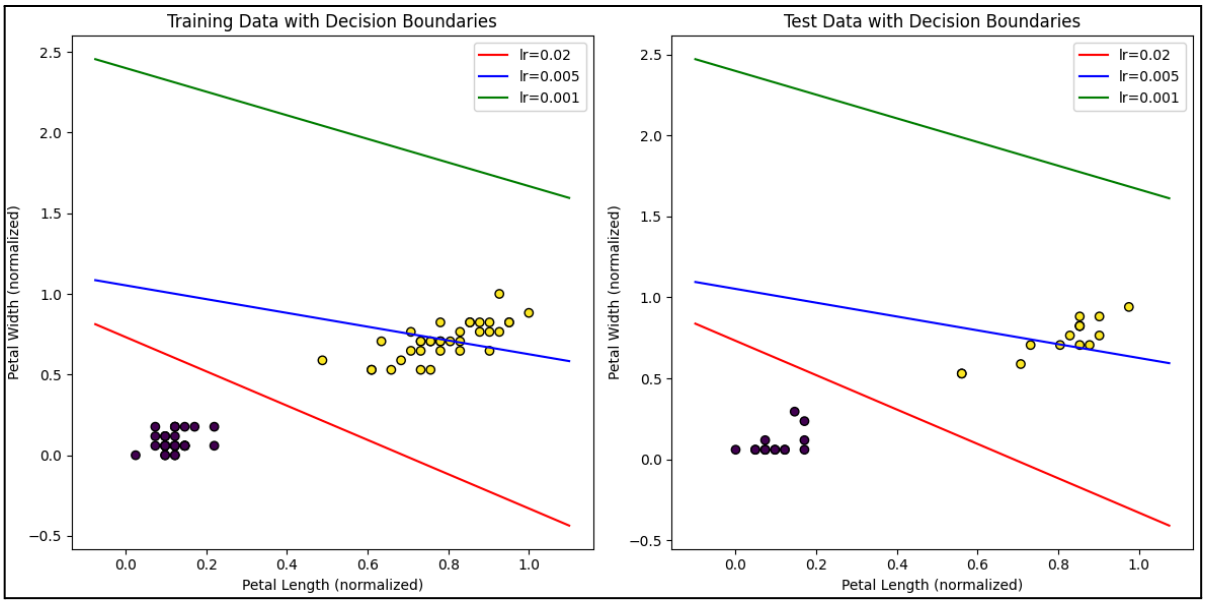
سپس ۳ مدل با نرخ‌های یادگیری متفاوت آموزش دادیم (همه با ۵۰ epoch) و دقت و خطای هر مدل در هر epoch را ذخیره کردیم تا در مراحل بعد به تحلیل‌ها بپردازیم. در هر دوره تمامی وزن‌ها و بایاس‌ها به همراه دقت و خطای مدل در آن دوره ذخیره می‌شوند.

نکته : موردی که در اینجا قابل ذکر است این است که با توجه به تعداد کم epoch های خواسته شده در صورت پروژه، در حالتی که وزن‌های اولیه (که رندم تعریف شده‌اند) خیلی دور باشند، حتی بالاترین نرخ یادگیری گفته شده (۰.۰۲) هم با ۱۰ دور آموزش به دقت کافی **نخواهد** رسید! پس به خاطر داشته باشید که می‌توان یا epoch های بیشتری آموزش داد و یا یک seed مناسب برای وزن‌های رندم اولیه تنظیم کرد. ما هر دوی این کارها را انجام دادیم که یعنی هم به جای ۱۰ epoch مدل‌ها را ۵۰ epoch آموزش دادیم و هم اینکه به صورت دستی دنبال seed مناسبی برای وزن‌های رندم اولیه گشتیم که نمودارهای خروجی معنی‌داری داشته باشیم. در صورتی که این موارد اتفاق نمی‌افتاد هم خروجی در epoch های بیشتر تفاوتی نمی‌کرد.
جزییات بیشتر در بخش تحلیل نمودارها گفته خواهد شد.

۴-۳. نمودارها و تحلیل نتایج

اولین نمودار موجود خطوط جدا کننده مربوط به هر مدل (با نرخ یادگیری متفاوت اما تعداد epoch یکسان) روی هر دو دیتای تست و آموزش است. در نمودار زیر نقاط نشان‌دهنده داده‌ها بوده و هر رنگ نشان دهنده یک کلاس متفاوت است.

همچنین هر خط (با توجه به راهنما) مربوط به یک مدل با نرخ یادگیری مشخص شده است.



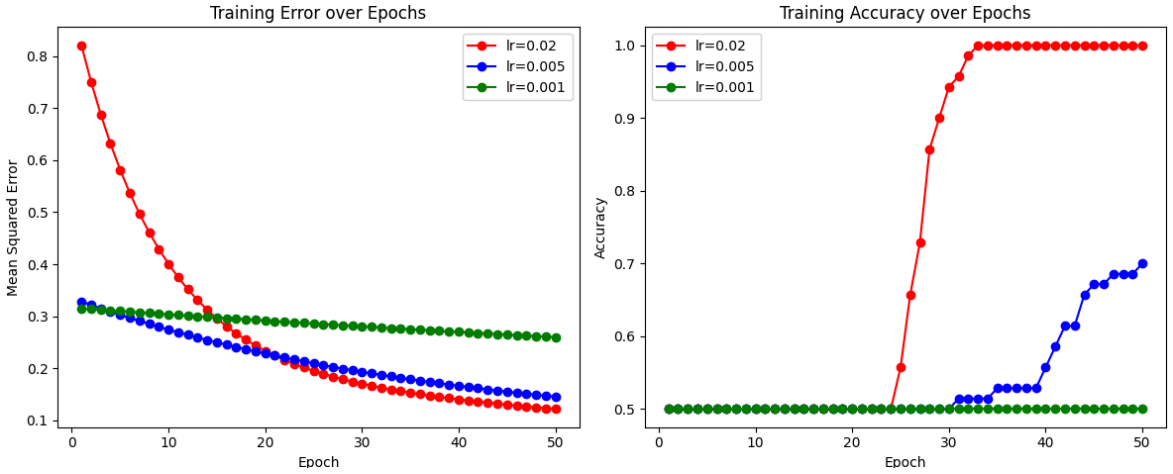
شکل ۲۱. خطوط جدا کننده مدل‌ها روی داده آموزشی و تست

تاثیری که در همین ابتدا قابل مشاهده است چرا که با توجه به یکسان بودن تعداد epochها، مدلی که نرخ یادگیری بالاتری داشته موفق شده به خط جداکننده مورد نظر برسد و دقت 100% را فراهم کند (در نمودارهای بعدی قابل مشاهده است) مدل با نرخ یادگیری متوسط تا حدی به هدف نزدیک شده اما هنوز مقداری از داده‌های تست و آموزش را اشتباه دسته‌بندی می‌کند. مدل با کمترین نرخ یادگیری نیز فاصله زیادی با جداکردن درست داده‌ها دارد و دقت ۵۰ درصدی که از آن گزارش می‌شود عملا تفاوتی با انتخاب **شانسی** ندارد! (چرا که در دیتاست ما توزیع داده‌ها یکسان است)

در مورد همگرایی نیز حداقل در حالت فعلی همگرایی رخ نداده است. در تعداد epochهای بسیار بالاتر احتمالا تاثیر نرخ یادگیری کمتر شده و به همگرایی نزدیک می‌شدند اما همچنان موردی که قابل توجه است این است که وزن‌های اولیه به صورت رندم داده شده بود؛ در نتیجه حتی در صورتی که نرخ یادگیری یکسانی هم می‌داشتیم لزوما خط جداکننده یکسانی دریافت نمی‌کردیم چرا که این خط **یکتا نیست**! اما به طور کلی می‌توان گفت مدل‌ها در نهایت به سمت همگرا شدن خواهند رفت.

در نمودار صفحه بعد می‌توان روند تغییرات خطا و دقت هر مدل را مشاهده کرد. همانطور که در نمودارهای قبلی هم دیده می‌شد مدل با کمترین نرخ یادگیری عملا هنوز به جداکننده لازم نرسید است (هرچند که خطای آن کم‌تر شده است)

دو مدل دیگر هر کدام در نقطه از آموزش به جداکننده مورد نظر رسیده و دقت ۱۰۰ درصدی را ثبت کرده اند.



شکل ۲۲. نمودار دقت و خطای مدل‌ها در هر epoch

آخرین موردی که به آن باید اشاره کرد عدم عملکرد مناسب Adaline روی دیتایی است که به صورت خطی قابل جداسازی نیست. در دیتاست فعلی به طور واضح این کار شدنی بود (به نمودار خطوط جداساز مراجعه شود) اما در صورتی که این اتفاق شدنی نبود (برای مثال با یک چند ضلعی جداسازی ممکن بود) باید از مدل‌های پیچیده‌تر (مثلا Madaline) استفاده نمود.

پرسش ۴. آموزش اتو انکودر و طبقه بندی با دیتاست MNIST

۴-۱. مقدمه

در این قسمت سعی داریم که با استفاده از اتو انکودر، دیتاست MNIST را طبقه بندی کنیم .

۴-۲. دانلود و پیش پردازش داده ها

با استفاده از ترکیب (Compose) کردن transform ها در هنگام لود کردن ، هم داده ها را Normalize می کنیم و هم آنها را به یک بردار 1×784 تایی Reshape میکنیم .

```
original_set = torchvision.datasets.MNIST(
    "/tmp",
    download=True,
    train=True,
    transform=transforms.Compose(
        [
            transforms.ToTensor(),
            transforms.Normalize(0, 1 ),
            transforms.Lambda(lambda x : x.view(-1))

        ]
    )
)
```

شکل ۲۳. ترکیب transform های مختلف جهت پیش پردازش داده ها

شکل ۲۴. نمونه ای از دیتاست آموزش MNIST

۳-۴. طراحی و پیاده سازی مدل

بخش اول : ساخت اتو انکودر ها

```python
class FatAutoEncoder(nn.Module):
    def __init__(self):
        super(FatAutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 8),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(8, 128),
            nn.ReLU(),
            nn.Linear(128, 784)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

شکل ۲۵.اتو انکودر چاق (با خروجی انکودر ۸تایی)

```python
class ThinAutoEncoder(nn.Module):
    def __init__(self):
        super(ThinAutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 4),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(4, 128),
            nn.ReLU(),
            nn.Linear(128, 784)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

شکل ۲۶.اتو انکودر لاغر (با خروجی انکودر ۴تایی و یک لایه کمتر)

بخش اول : ساخت طبقه بندی ها به کمک انکودر های آموزش دیده شده

```python
class FatClassifier(nn.Module):
    def __init__(self, encoder: nn.Sequential):
        super(FatClassifier, self).__init__()
        self.encoder = encoder
        for param in self.encoder.parameters():
            param.requires_grad = False

        self.classifier = nn.Sequential(
                nn.Linear(8, 4),
                nn.ReLU(),
                nn.Linear(4, 10),
                nn.Softmax(dim=1)
            )

    def forward(self, x):
        x = self.encoder(x)
        x = self.classifier(x)
        return x
```

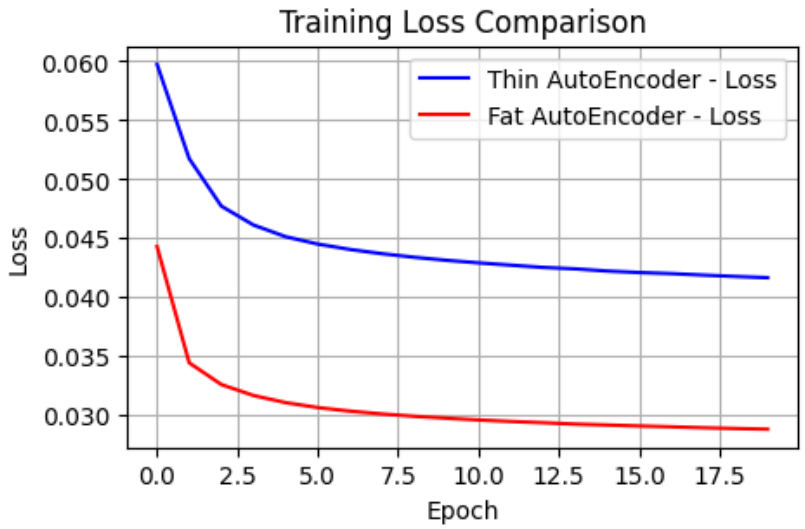
شکل ۲۷. مدل Fat Classifier به همراه انکودر چاق

```python
class ThinClassifier(nn.Module):
    def __init__(self,encoder: nn.Sequential):
        super(ThinClassifier, self).__init__()
        self.encoder = encoder
        for param in self.encoder.parameters():
            param.requires_grad = False

        self.classifier = nn.Sequential(
            nn.Linear(4, 10),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.classifier(x)
        return x
```

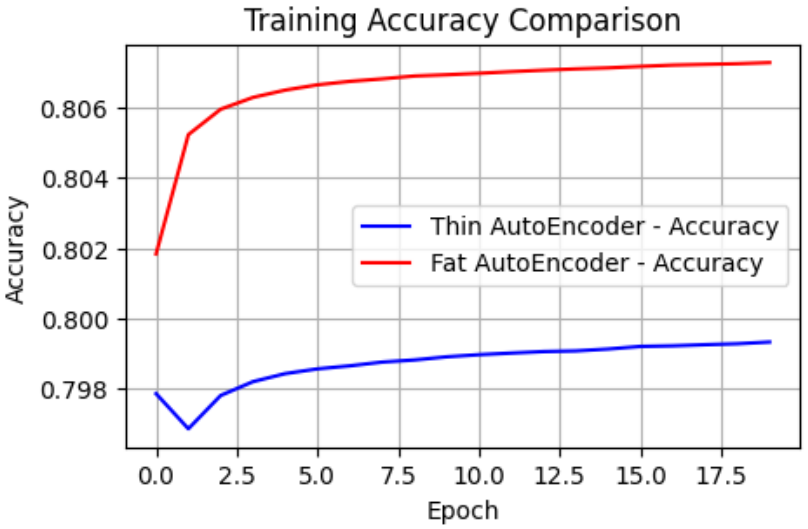
شکل ۲۸. مدل Thin Classifier به همراه انکودر لاغر

۴-۴. نتایج و تحلیل

- نمودار خطا بازسازی انکودرها را روی دادگان آموزش برحسب ایپاک رسم کنید.



شکل ۲۹. مقایسه خطا در آموزش مدلهای اتو انکودر

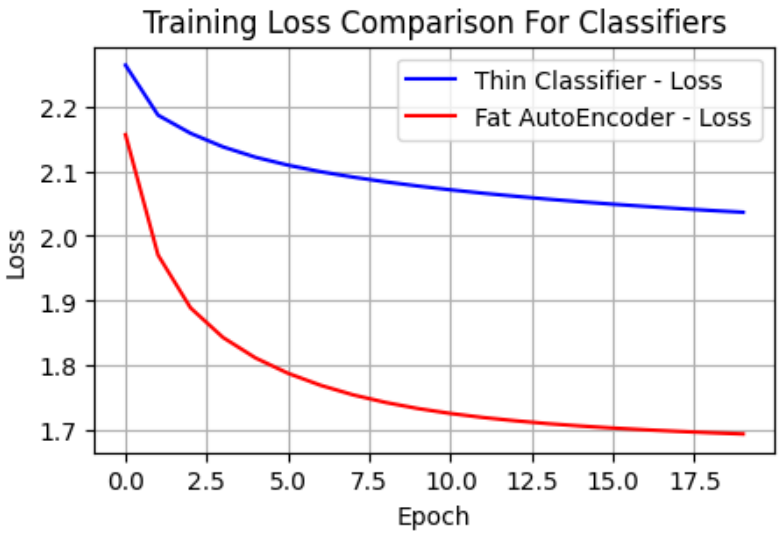


شکل ۳۰. مقایسه دقت در آموزش مدلهای اتو انکودر

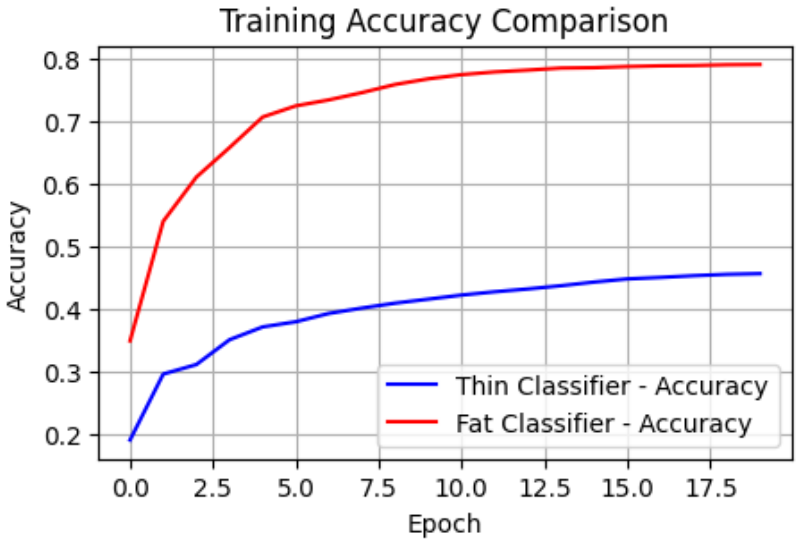
همانطور که مشاهده میشود در زمان آموزش مدل خطای مدل لاغر تر بیشتر است ، و همچنین در زمان آموزش مدل چاق تر Accuracy بیشتری دارد.

در ایپاک اول مدل لاغر، در یک لحظه به سمت دقت پایین تری می رود ولی در ایپاک های بعدی این موضوع اصلاح میشود و با grad decent به سمت اصلاح مدل پیش می رود.

- دقت هر مدل را روی داده های آموزش و دادگان تست محاسبه کنید و نمودار آن را برحسب ایپاک رسم کنید.

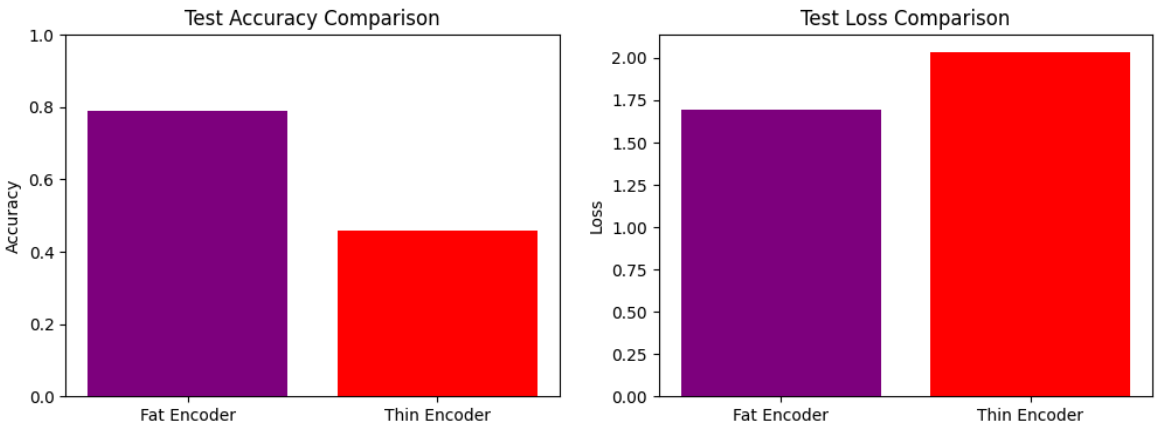


شکل ۳۱. مقایسه خطا در آموزش مدلهای classifier



شکل ۳۲. مقایسه Accuracy در آموزش مدلهای classifier

همانطور که مشاهده می شود در زمان آموزش مدل های classifier ها ، مدل چاق تر با نورون های بیشتر در انتهای انکودر عملکرد بهتری دارد، علت این امر این است که برای تبدیل وکتور ۷۴۸ تایی به یک بردار ساده شده ، کمتر از مقداری که information ارزشمند دارند ، از دست میروند.



شکل ۳۳. مقایسه مدلهای classifier در دیتاست تست

- تعداد پارامتر های هر مدل (انکودر + رمز گشا) و (انکودر + طبقه بندی) را گزارش کنید.

Fat Auto-encoder :

لایه اول : (128×784)+128=100,352

لایه دوم : (8×128)+8=1,032

لایه اول دیکودر : (128×8)+128=1,152

لایه دوم دیکودر : (784×128)+784=100,576

مجموع کل : 100,352+1,032+1,152+100,576=*203,112*

Thin Auto-encoder :

لایه اول : (128×784)+128=100,352

لایه دوم : (4×128)+4=516

لایه اول دیکودر : (128×4)+128=640

لایه دوم دیکودر : (784×128)+784=100,576

مجموع کل : 100352+516+640+100576=*202,084*

Fat Encoder + Classifier Parameters

لایه اول : $(128\times784)+128=100{,}352$

لایه دوم : $(128\times8)+8=1{,}032$

لایه اول طبقه بندی : $(8\times4)+4=36$

لایه دوم طبقه بندی : $(8\times10)+10=90$

مجموع کل : *101,510*

Thin Encoder + Classifier Parameters :

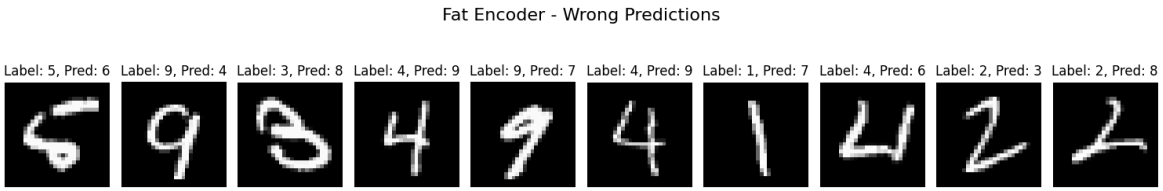
لایه اول : $(128\times784)+128=100{,}352$

لایه دوم : $(128\times4)+4=516$

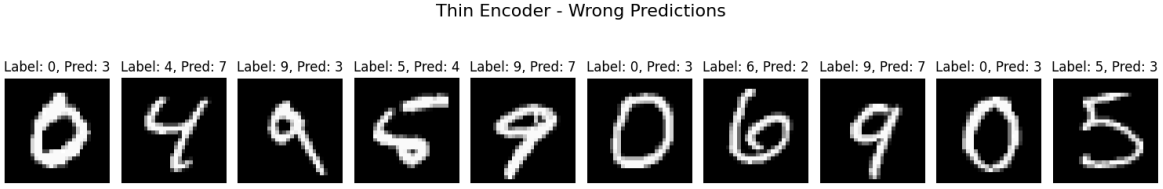
لایه اول طبقه بندی : $(4\times10)+10=50$

مجموع کل :*100,918*

- داده های غلطی که هر یک از مدل ها پیش بینی کرده بودند را به صورت نمونه در قسمت زیر میتوانید مشاهده نمایید :



شکل ۳۴. برخی از داده هایی که در مدل fat غلط پیش بینی شده بودند



شکل ۳۵. برخی از داده هایی که در مدل thin غلط پیش بینی شده بودند

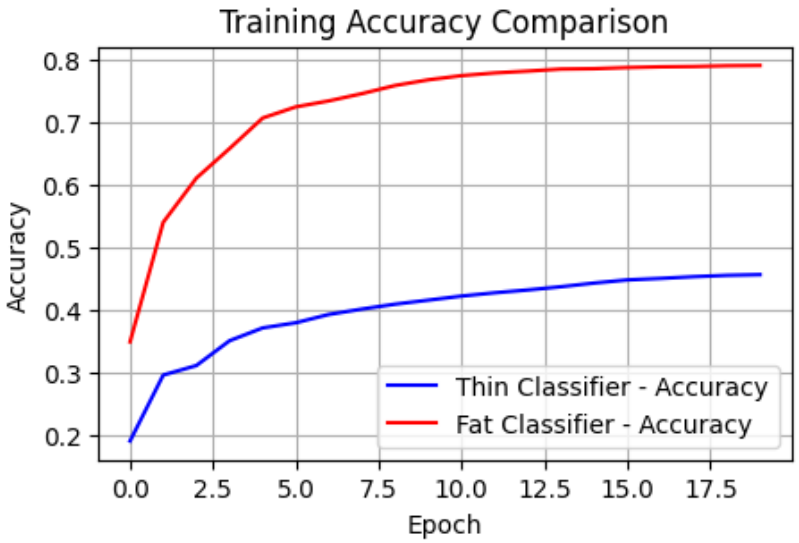
تحلیل و مقایسه :

- مقایسه خطای بازسازی : مدل با ۸ نورون بهتر عمل کرد یا مدل با ۴ نورون ؟ چرا ؟
  - مدل با ۸ نورون بهتر عمل کرده است ، چون برای reduction کمتر داده هایی که اطلاعات ارزشمند (valuable information)دارند را کاهش بعد میدهیم ، بنابراین کاهش بعد کمتری داشته ایم و توانسته ایم که disturbance ها را فقط حذف کنیم ، نه اطلاعات مهم و ارزشمند که قابلیت classification به ما میدهند.

- مقایسه دقت طبقه بندی : کدام اندازه نهان عملکرد بهتری داشت ؟ آیا کاهش ابعاد بیش از حد به دقت آسیب زد ؟
  - کاهش بعد های برای طبقه بندی باعث کاهش دقت شد ، چون که در مدل چاق تر ، بعد های وکتور ۷۸۴ تایی را کمتر کاهش داده ایم و اطلاعات بیشتری از دیتا ها داریم.

- مقایسه تعداد پارامترها: تأثیر تعداد پارامترها بر عملکرد چیست؟
  - در اینجا با مقایسه این دو مدل ، نتیجه میگیریم که هرچه تعداد پارامتر ها بیشتر باشد ، با توجه به این که کاهش بعد کمتری رخ داده است ، و اطلاعات کمتری را correlation هایشان را حذف کرده ایم ، بنابراین دقت بیشتری داریم . بنابراین توانسته ایم که disturbance ها را فقط حذف کنیم ، نه اطلاعات مهم و ارزشمند که قابلیت classification به ما میدهند.

- آیا نشانه هایی از بیش برازش مشاهده کردید ؟ چگونه میتوان آن را کاهش داد ؟
  - بله در مدل thin ، نشانه هایی از Overfitting کاملا مشهود است. همانطور که در شکل زیر نشان داده شده است ، مدل thin در هنگام آموزش به دقت ۰.۵ دست یافته است ، اما در هنگام مواجه با دیتاست تست ، این دقت به ۰.۴ کاهش میابد که این ۱۰ درصد کاهش دقت نشان دهنده ی این است که مدل ما رو داده های آموزشی Overfit شده است .



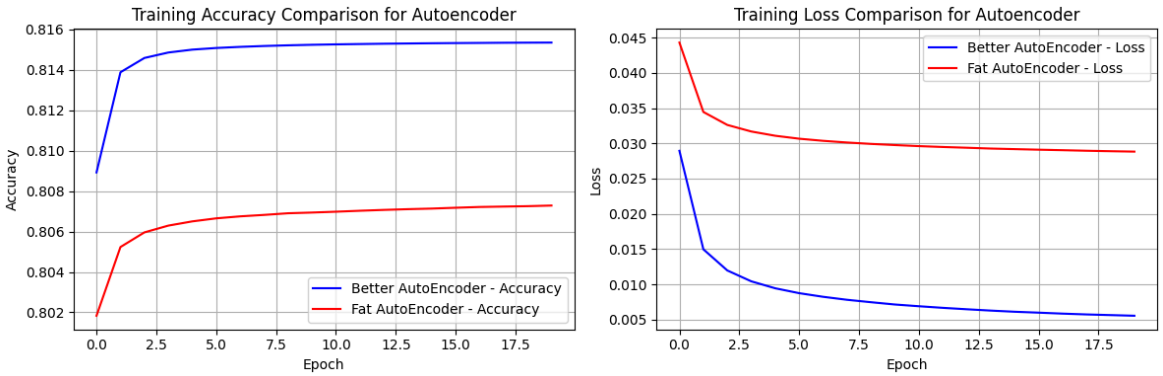
شکل ۳۶. مقایسه Accuracy در آموزش مدلهای classifier

با توجه به نتایج بدست آمده سعی میکنیم که یک مدل بهتری ، ارائه دهیم :

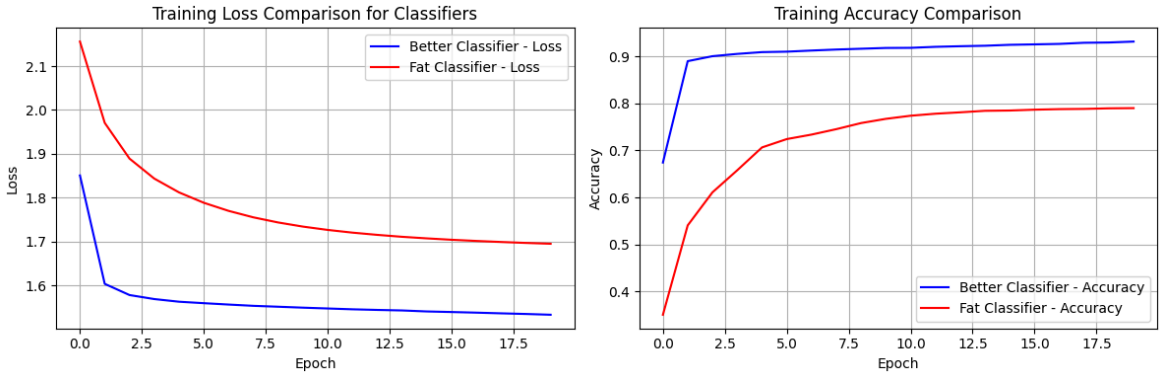
یک معماری کامل تری برای Autoencoder ارائه میدهیم که باعث کاهش کمتر بعد عکس شود :

```python
class BetterAutoEncoder(nn.Module):
    def __init__(self):
        super(BetterAutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(784, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 784)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

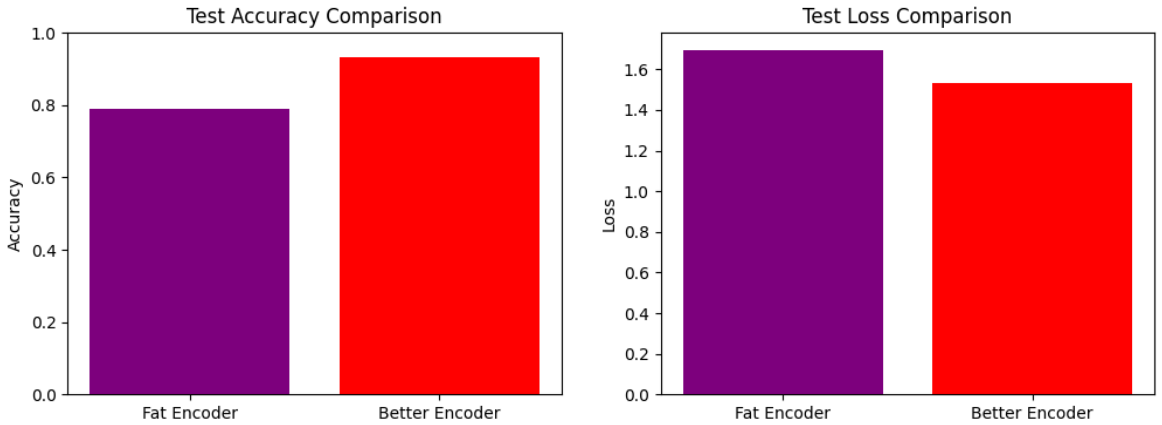
شکل ۳۷. مدل اتو انکودر بهتر و حجیم تر



شکل ۳۸. مقایسه مدل جدید با بهترین مدل قبلی (مدل چاق) در هنگام آموزش اتوانکودر



شکل ۳۹. مقایسه مدل جدید با بهترین مدل قبلی (مدل چاق) در هنگام آموزش برای classifier



شکل ۴۰. مقایسه مدل جدید با بهترین مدل قبلی (مدل چاق) بر روی دیتاست ارزیابی(تست)

مدل‌های اتو انکودر با پارامترهای بیشتر و کاهش بعد کمتر معمولاً دقت بیشتری دارند به دلایل زیر:

- ظرفیت مدل بیشتر : تعداد پارامترهای بیشتر به مدل این امکان را می‌دهد که روابط پیچیده‌تری را در داده‌ها یاد بگیرد. وقتی تعداد نودها و لایه‌ها بیشتر باشد، مدل توانایی بیشتری برای یادگیری ویژگی‌های پیچیده و الگوهای غیرخطی دارد.

- از دست نرفتن اطلاعات هنگام کاهش بعد : اگر کاهش بعد (Dimensionality Reduction) بیش از حد شدید باشد، مدل بخش زیادی از اطلاعات موجود در ورودی را از دست می‌دهد. وقتی کاهش بعد کمتر باشد (مثلاً کاهش به 8 بعد به جای 4 بعد)، مدل اطلاعات بیشتری از داده اصلی را حفظ کرده و بازسازی دقیق‌تری انجام می‌دهد.

- نمایش ویژگی‌های پیچیده‌تر : مدل‌های با کاهش بعد کمتر، امکان نگهداری ویژگی‌های با ابعاد بالاتر را دارند. این باعث می‌شود که ویژگی‌های پیچیده‌تری (مثل انحناها، بافت‌ها، یا الگوهای خاص) در داده‌ها به‌خوبی نمایش داده شوند.