به نام خدا
دانشگاه تهران
دانشکده مهندسی
برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین اول

	نام و نام خانوادگی	مبینا مهرآذر
	شماره دانشجویی	810100216
	نام و نام خانوادگی	محمدرضا محمدهاشمی
	شماره دانشجویی	810100206
	مهلت ارسال پاسخ	1403/08/15

فهرست

شکل‌ها

جدول‌ها

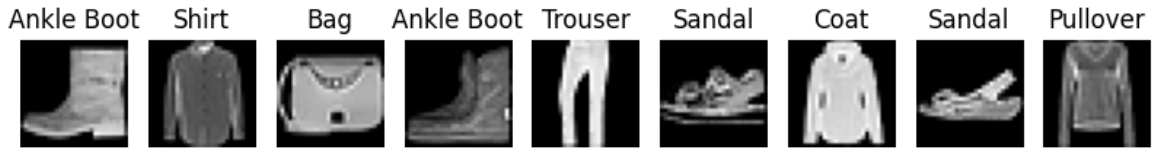
قوانین

HW1_MohammadHashemi_810100206_Mehrazar_810100216.zip

پرسش 1. تحلیل و طراحی شبکه های عصبی چند لایه

در این بخش، یک شبکه عصبی چندلایه یا همان MLP برای مجموعه دادگان Fashion-MNIST آموزش می‌دهیم.

این مجموعه داده را از کتابخانه torch vision دانلود کردیم و چند نمونه از تصاویر آن را در شکل زیر نمایش می‌دهیم.



شکل 1. **تصاویر نمونه از مجموعه داده**

همانطور که مشخص است، این مجموعه داده شامل تصاویر سیاه و سفید از چندین مدل پوشاک با ابعاد 28x28 می‌باشد.

توجه شود در تمامی بخش‌ها از batch size = 64 استفاده شده است. مقدار 64 در مقابل مقدار دیفایت 32 این پارامتر در این دیتاست عملکرد بهتری داشته است. دارد. اندازه batch size تعیین می‌کند که مدل چند نمونه را قبل از به‌روزرسانی وزن‌ها ببیند. دسته‌های بزرگ‌تر می‌توانند تخمین گرادیان پایدارتر و نماینده‌ای را برای هر به‌روزرسانی ارائه دهند و با توجه به ماهیت داده‌ی ما، این نتیجه حاصل شده است.

همینطور توجه کنید که در هر بخشی که از Dropout استفاده شده، نرخ مربوطه روی 0 ،evaluation بوده و تاثیری نخواهد داشت.

۱-۱. طراحی MLP

این مدل با یک لایه مخفی با 100 نود و تابع فعال‌سازی ReLU طراحی کرده‌ایم. نرخ Dropout به 30% و ضریب لامبدای L2 Regularizer به مقدار 0.0001 مقداردهی شده است.

```python
class Model(nn.Module):
    def __init__(self, h1, do_rate=0, in_features=28*28, out_features=10):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(in_features, h1)
        self.dropout = nn.Dropout(do_rate)
        self.out = nn.Linear(h1, out_features)

    def forward(self, x):
        # x = self.flatten(x)
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.out(x)
        return x
```

شکل 2. **ساختار مدل**

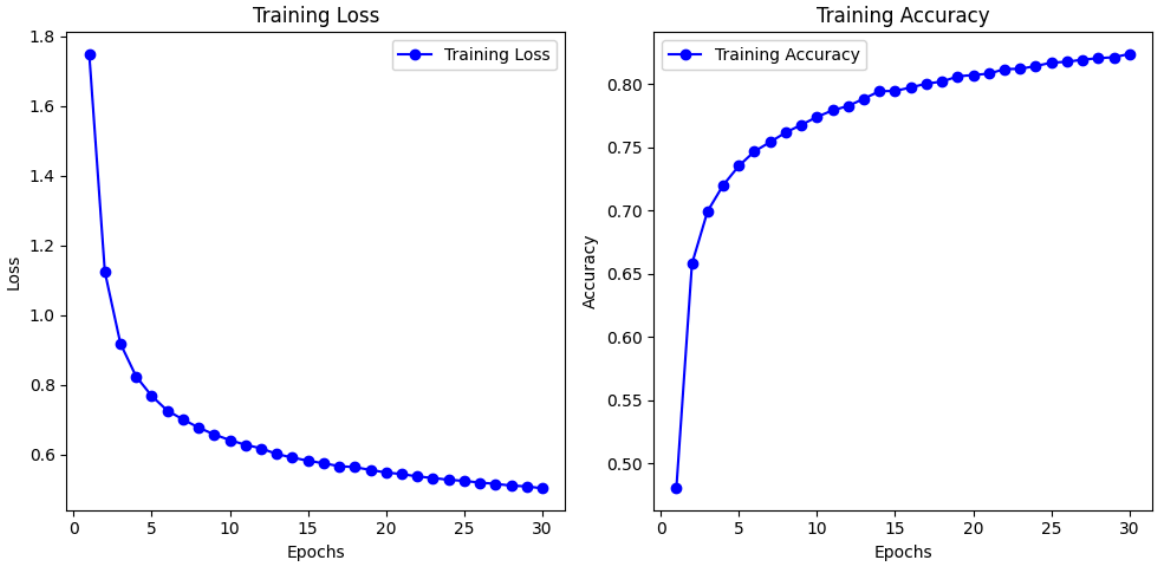
در تمامی بخش‌ها از تابع هزینه Cross Entropy استفاده شده است.

در این مدل از l2 regularizer استفاده شده که تکنیک مورد استفاده در یادگیری ماشین و یادگیری عمیق برای کاهش بیش از حد برازش با جریمه کردن وزنه های بزرگ در یک مدل است. این کار با افزودن یک عبارت منظم سازی به تابع هزینه کار می کند، که مدل را تشویق می کند تا وزن ها را در طول تمرین کوچکتر و پایدارتر نگه دارد.

معادله آن به صورت زیر است:

$$Loss \ += \ \lambda \sum_i w_i^2$$

نمودار دقت و خطای در حین آموزش مدل، به شرح زیر است:



شکل 2. **نمودار دقت و خطای مدل برای 30 ایپاک**

همانطور که مشاهده می‌شود، در ابتدا، میزان تغییر دقت و خطا بیشتر بوده و در ایپاک های آخر، همگرا تر می‌شوند.

نتایج نهایی به تفکیک کلاس ها به شرح زیر است:

دقت نهایی مدل ترین شده 0.84 است که دقت قابل قبولی برای این تنظیمات و عدم استفاده از Optimizer در مدل می‌باشد. دقت داده‌ی تست مقداری کمتر و برابر 0.82 است که بسیار نزدیک به دقت ترین بوده و این نشان می‌دهد مدل ما Overfit نکرده است. همینطور مقدار خطا در داده تست برابر 0.4913 شده است.

	precision	recall	f1-score	support
T-shirt/top	0.78	0.83	0.80	6000
Trouser	0.97	0.95	0.96	6000
Pullover	0.74	0.73	0.74	6000
Dress	0.84	0.87	0.85	6000
Coat	0.73	0.79	0.76	6000
Sandal	0.91	0.91	0.91	6000
Shirt	0.63	0.54	0.58	6000
Sneaker	0.89	0.88	0.89	6000
Bag	0.94	0.94	0.94	6000
Ankle boot	0.91	0.93	0.92	6000
accuracy			0.84	60000
macro avg	0.83	0.84	0.84	60000
weighted avg	0.83	0.84	0.84	60000

شکل 3. **نتایج معیار های مختلف بر روی هر کلاس داده، و نتایج کلی مدل روی داده ترین**

	precision	recall	f1-score	support
T-shirt/top	0.79	0.81	0.80	1000
Trouser	0.98	0.94	0.96	1000
Pullover	0.72	0.71	0.71	1000
Dress	0.81	0.85	0.83	1000
Coat	0.71	0.76	0.73	1000
Sandal	0.91	0.89	0.90	1000
Shirt	0.57	0.51	0.54	1000
Sneaker	0.88	0.88	0.88	1000
Bag	0.93	0.94	0.93	1000
Ankle boot	0.89	0.93	0.91	1000
accuracy			0.82	10000
macro avg	0.82	0.82	0.82	10000
weighted avg	0.82	0.82	0.82	10000

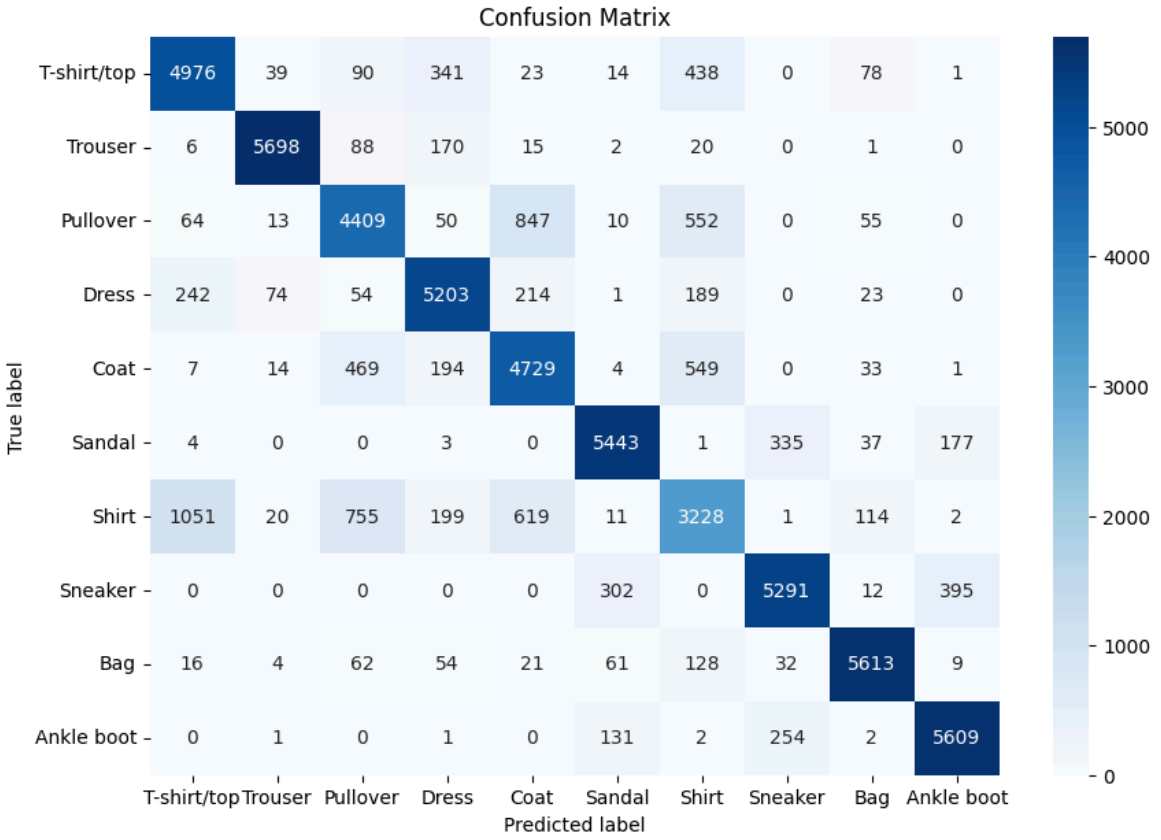
شکل 4. **نتایج معیار های مختلف بر روی هر کلاس داده، و نتایج کلی مدل روی داده تست**

با توجه به نتایج مدل روی داده‌ی تست، روی داده‌ی مربوط به Bag، Sandal، Ankle boot، Trouser، Sneaker نتایج و دقت خوبی دارد و مقادیر متریک های recall، precision و f 1 score بالا و خوبی دارند. یعنی در تشخیص عکس‌هایی با این کلاس‌ها مدل کمتر اشتباه کرده است.

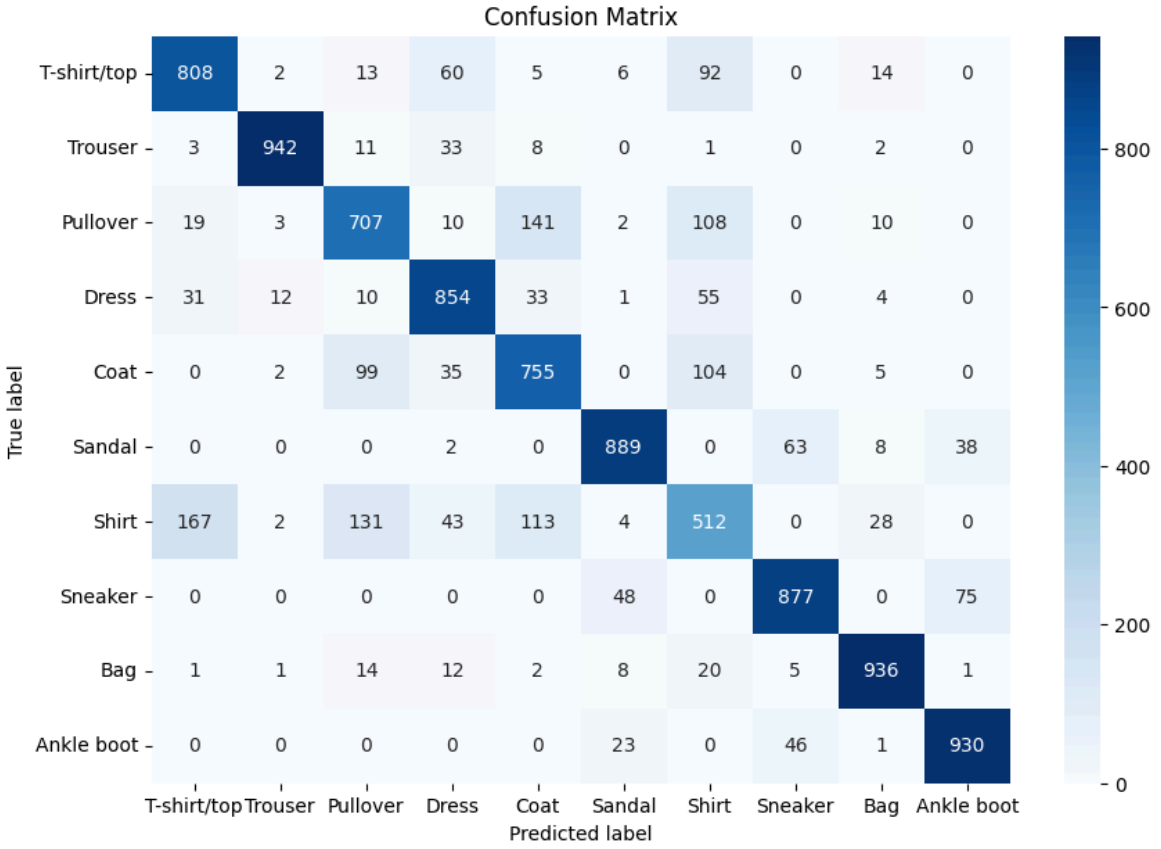
همینطور روی پوشاک Shirt بدترین عملکرد، روی بقیه کلاس ها هم عملکرد از بد به خوب به شرح زیر است:

Pullover، coat، T shirt/top، Dress

ماتریس آشفتگی برای ترین و تست، حاصل به شرح زیر است:



شکل 5. **ماتریس آشفتگی مربوط به داده ترین**



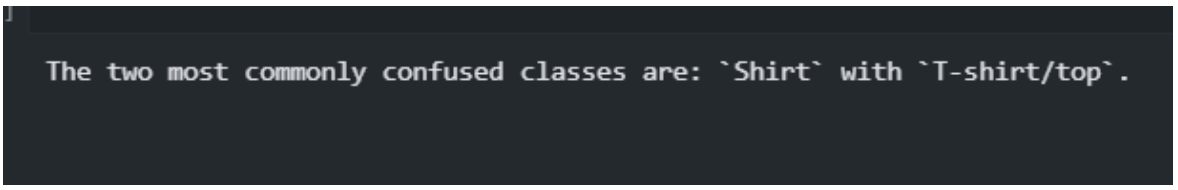
شکل 6. **ماتریس آشفتگی مربوط به داده تست**

برای هر کلاس، کلاسی که بیشتر با آن اشتباه گرفته می‌شود به شرح زیر است:

```
Class `T-shirt/top` is most often confused with class `Shirt`
Class `Trouser` is most often confused with class `Dress`
Class `Pullover` is most often confused with class `Coat`
Class `Dress` is most often confused with class `Shirt`
Class `Coat` is most often confused with class `Shirt`
Class `Sandal` is most often confused with class `Sneaker`
Class `Shirt` is most often confused with class `T-shirt/top`
Class `Sneaker` is most often confused with class `Ankle boot`
Class `Bag` is most often confused with class `Shirt`
Class `Ankle boot` is most often confused with class `Sneaker`
```

شکل 6. **نتایج شبیه ترین کلاس ها**

همینطور دو کلاسی که بیشتر با هم اشتباه گرفته می‌شوند، کلاس Shirt با T-shirt/top میباشد.



شکل 6. **شبیه ترین دو کلاس**

سوال) چگونه افزایش پیچیدگی مدل با استفاده از تعداد بیشتر لایه های مخفی میتواند بهبود عملکرد را در پی داشته باشد؟

این کار، ظرفیت مدل را برای یادگیری الگوهای پیچیده در در داده ورودی افزایش میدهد و مدل پیچیده تر، در تشخیص این الگوها در داده ترین و پیشبینی، بهتر عمل خواهد کرد، چون از قبل آنها را یاد گرفته است.

در واقع منظور از الگو و ویژگی های پیچیده تر، روابط غیر خطی و وابستگی های پیچیده از نظر ریاضی می‌باشد.

اما باید توجه داشت که افزایش پیچیدگی بی حد مدل همیشه خوب نیست و از جایی به بعد مدل شروع به یادگیری نویز موجود در داده ترین را میکند و اصطلاحا overfit میکند. در این حالت مدل ریزترین جزئیات داده ترین را یاد گرفته و عملکرد خیلی خوبی روی آن دارد اما نتیجه مشابهی روی داده تست نخواهد داشت. برای حل این مشکل، تکنیک هایی مثل dropout، l2 regularization در این سوال ارائه و بررسی شده است. همینطور، تکنیکی تحت عنوان Early Stopping مدل را وادار میکند قبل از overfit کردن، فرایند train را خاتمه دهد.

سوال) چه معیار هایی برای انتخاب بهترین پیکربندی وجود دارد؟

برای این بررسی سراغ متریک هایی مثل Accuracy - F1 score - Precision- Recall - Loss میرویم. نمودارها و متریک های دیگری نظیر AUC-ROC هم در برخی مسائل برای تحلیل مدل به کمک ما میایند. ما در ترین مدل سعی داریم پارامترهایی مثل دقت و اسکور را زیاد و خطا را به کمترین حالت خود برسانیم.

برای انتخاب مدل مناسب، باید به داده ورودی آن هم توجه کرد. میزان پیچیدگی داده، رابطه مستقیم با پیچیدگی مدل را خواهد داشت. شبکه های عصبی به علت ماهیت پیچیده ای که دارند، برای داده های پیچیده مثل عکس و داده های زبانی، استفاده میشوند.

در مدل هایی که از توابع activation بهره میبرند، میتوانیم ویژگی های غیرخطیی را متناسب با پیچیدگی داده تنظیم کنیم. بنابراین اجزای مدل هایی مثل شبکه های تعریف شده، با داده ورودی ارتباط دارند.

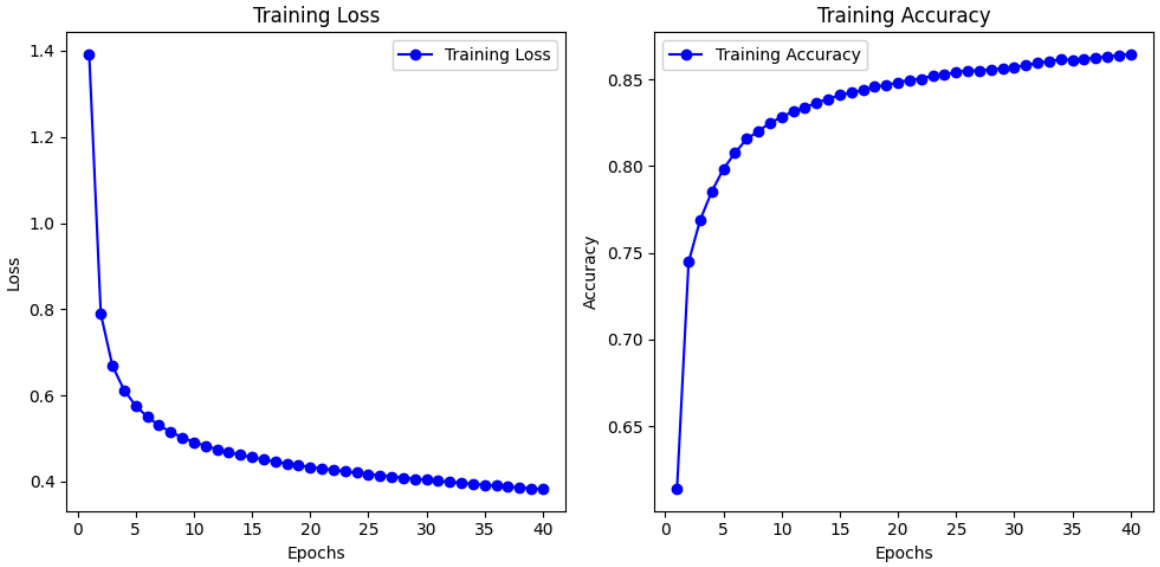
همینطور مدل خوب، باید دارای پیکربندی باشد که overfit یا under fit نشده باشد. روش های جلوگیری از overfit در بالا توضیح داده شده است. underfit زمانی رخ میدهد که مدل روی داده ترین هم نتیجه خوبی بدهد که با بررسی معیار ها و متریک های فوق در فرایند آموزش و انتخاب هایپر پارامتر های مختلف برای ارزیابی، جلوگیری میشود.

۱-۲. آموزش دو مدل متفاوت

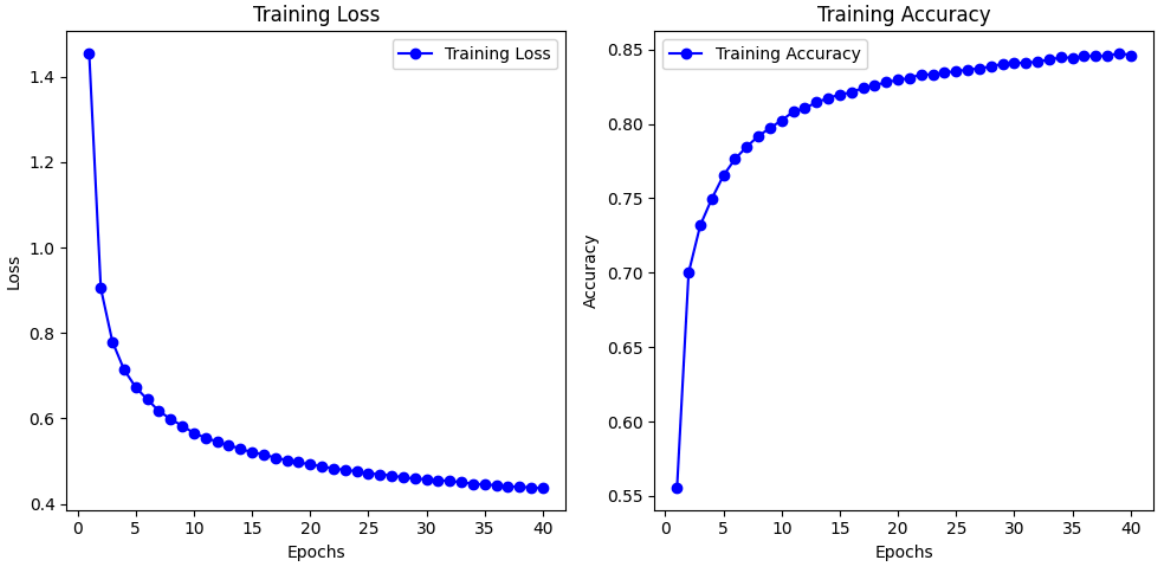
با اینکه دو مدل دقت‌های نزدیکی در اپیاک‌های یکسان نشان داده‌اند، اما مدل دوم، با اینکه در اپیاک‌های یکسان دقت کمتر و خطای بیشتری از مدل اول دارد، به علت پارامترهای بیشتر، robust تر است و روی داده تست، عملکرد بهتری دارد.

دقت در مدل اول 8581% و در مدل دوم 84.48% میباشد.

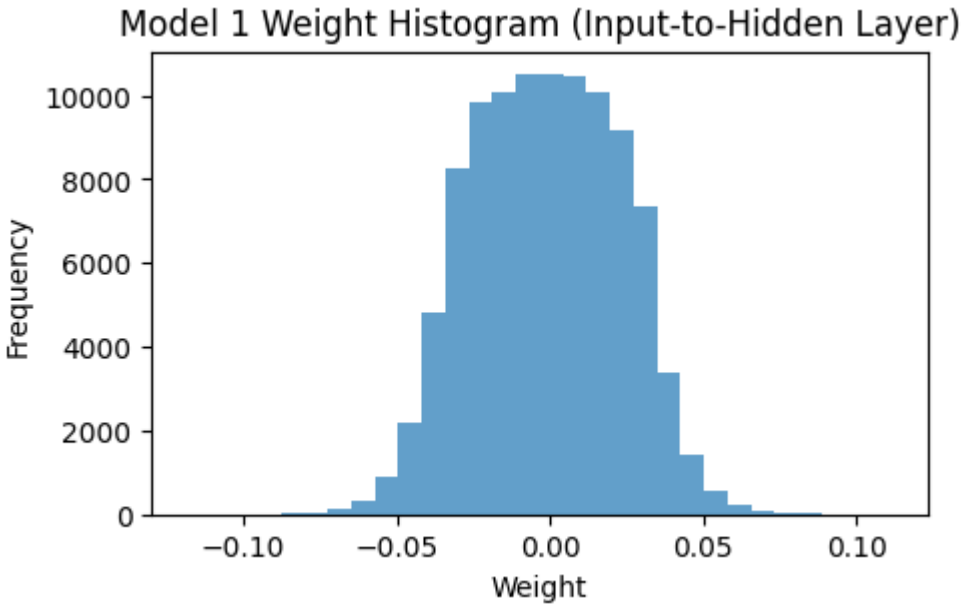
نمودار های دقت و خطای این دو مدل به شرح زیر است:



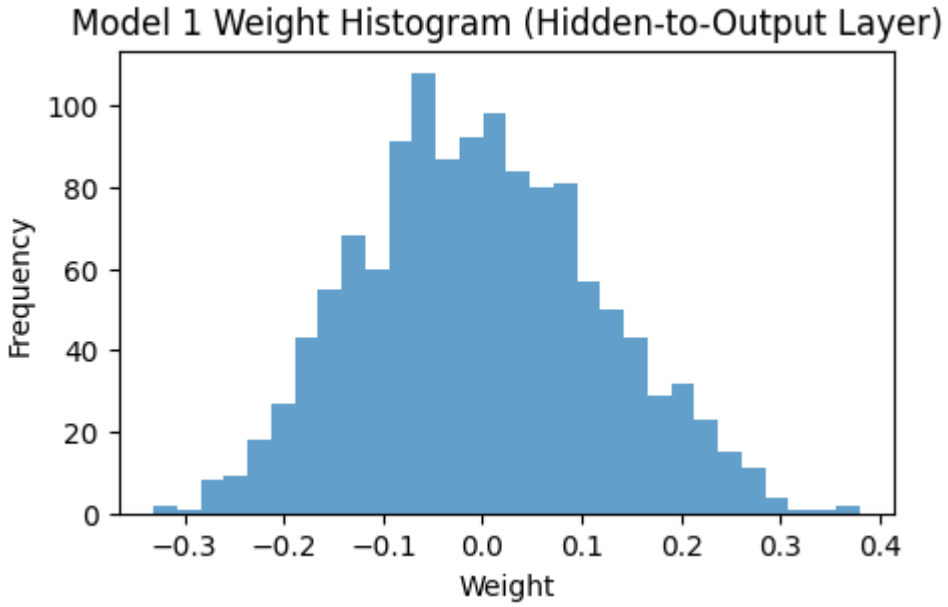
شکل 7. **نمودار دقت و خطای مدل 1**



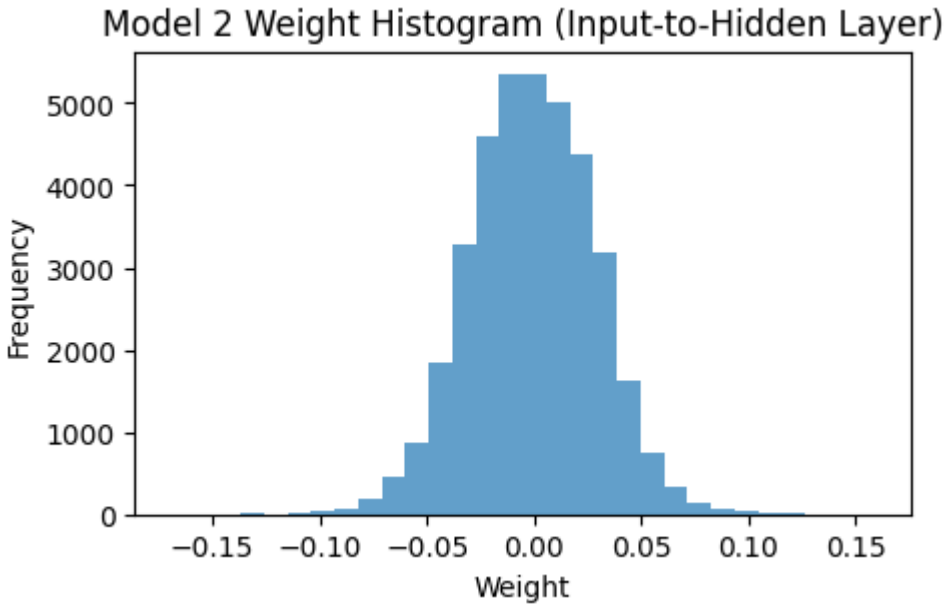
شکل 8. **نمودار دقت و خطای مدل 2**



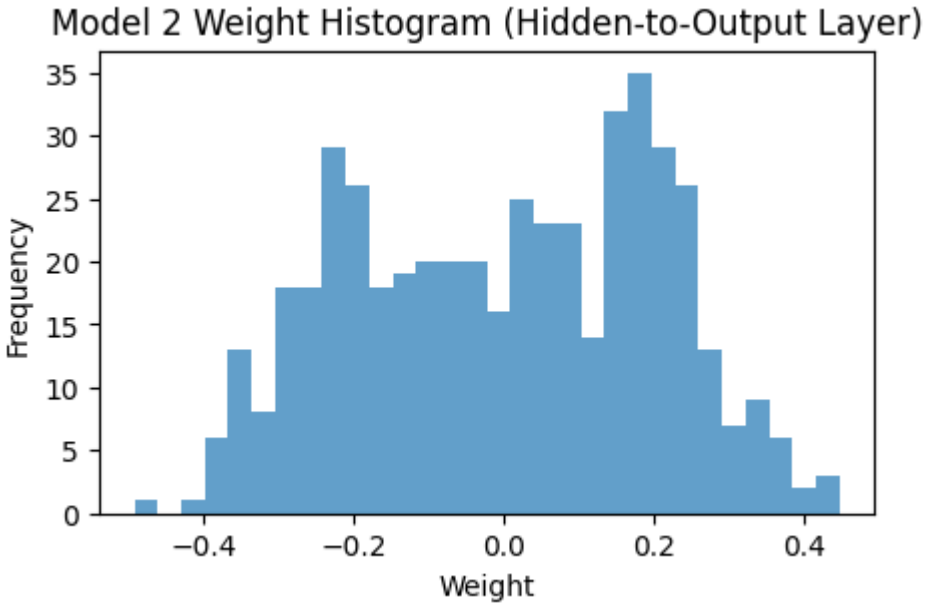
شکل 9. **نمودار وزن‌های لایه‌ی اول مدل 1**



شکل 10. **نمودار وزن‌های لایه‌ی دوم مدل 1**



شکل 11. **نمودار وزن‌های لایه‌ی اول مدل 2**



شکل 12. **نمودار وزن‌های لایه‌ی دوم مدل 2**

مدل دوم dropout دارد و از l2 regularizer استفاده کرده است.

همانطور که مشخص است، در مدلی که از l2 regularized استفاده شده، اندازه کلی وزن ها کمتر است و علت آن، محاسبه وزن های خیلی بزرگ به طور مستقیم در میزان خطا یا لاس مدل میباشد. در وابع تحت تاثیر آن، نورون های اول وزن‌هایی نزدیک تر به صفر دارند. در لایه اول مدل دوم، بیشتر وزن ها حدود 0 دارند. اما در شبکه اول، توزیع وزن ها حتی به حدود 0.75 و -0.75 هم رسیده است.

همینطور لایه dropout و ماهیت تصادفی آن در شبکه دوم باعث میشود شبکه برای پیشبینی وزن ها به نورون های به خصوصی وابسته نباشد. در این صورت همه نورون ها با توزیع یکنواخت آموزش می بینند و در فرایند تصمیم گیری تاثیر میگذارند.

مقدار وزن در لایه سافت مکس، به علت وجود dropout، بزرگتر میباشد.

بله، اضافه کردن اپتیمایزر ها به علت ریاضیات اعمال شده به بهبود انتخاب پارامترهای مدل می انجامد که در ادامه بررسی شده است.

۱-۳. الگوریتم بازگشت به عقب

- **الگوریتم Adam**

یک الگوریتم بهینه‌سازی است که از آن به جای روش گرادیان کاهشی تصادفی کلاسیک یا همان SGD برای به‌روزرسانی وزن‌های شبکه بر اساس تکرار در داده‌های آموزشی استفاده کرد. الگوریتم آدام را می‌توان به عنوان ترکیبی از RMSprop و گرادیان نزولی تصادفی با گشتاور (Momentum) در نظر گرفت.

با تلفیق ایده‌ی تکانه یا moment و تغییر نرخ یادگیری بر اساس گرادیان، الگوریتم Adam تعریف شد. در واقع اثر تکانه بر رابطه مربوط به الگوریتم RMSprop اضافه شده است.

$$w_i(t+1) = w_i(t) - lr(\frac{\partial L}{\partial w_i}(t));\ lr = -\frac{lr_0}{\sqrt{G_i(t)}+e}$$

$$G_i(t) = \beta G_i(t-1) + (1-\beta)((\frac{\partial L}{\partial w_i}t))^2;\ G_i(0) = 0$$

$$moment:\quad w_i(t+1) = w_i(t) - lr(\frac{\partial L}{\partial w_i}(t) + \alpha\frac{\partial L}{\partial w_i}(t-1))$$

و در نهایت رابطه آپدیت وزن‌ها در این الگوریتم به صورت زیر در می‌آید:

$$w_i(t+1) = w_i(t) - \frac{lr_0}{\sqrt{G_i(t)}+e}$$

$$M_i(t) = \alpha M_i(t-1) + (1-\alpha)\frac{\partial L}{\partial w_i}(t);\ M_i(0) = 0$$

از لحاظ محاسباتی بهینه بوده و به حافظه کمی نیاز دارد.

فرق بین الگوریتم Adam با گرادیان کاهشی این است که در روش گرادیان کاهشی تصادفی یک نرخ یادگیری واحد به نام آلفا را برای تمام به‌روزرسانی وزن‌ها حفظ می‌کند و این نرخ یادگیری در طول فرآیند آموزش تغییر نمی‌کند.

در مقابل نرخ یادگیری در الگوریتم بهینه‌سازی Adam برای هر یک از وزن‌های شبکه یا همان پارامترهایش حفظ می‌شود و این نرخ با شروع فرآیند یادگیری به صورت مجزا تطبیق داده می‌شود.

در روش بهینه سازی آدام ، هر یک از نرخ‌های یادگیری برای پارامترهای مختلف از گشتاورهای اول و دوم گرادیان‌ها محاسبه می‌شوند.

- **الگوریتم NAdam**

این بهینه‌ساز، ترکیب دو ایده یعنی ایده میانگین متحرک بهینه ساز Adam و Nesterov Momentum می‌باشد. در واقع این اپتیمایزر به کمک روش Nesterov Momentum در هر پیش از اپدیت کردن تقریبا یک نگاه به جلو

دارد و بر اساس جایی که میرود تصمیم به بهینه سازی میلگرد و این موضوع کمک میکند تا برخی از لوکال مینیمم ها را رد کرده و به سرعت بالاتری کانورج کنیم .

حال به روابط ریاضی می پردازیم :

ابتدا مووینگ اورج را حساب کرده :

$mt=\beta 1 \cdot mt-1+(1-\beta\ 1) \cdot gt$

$vt=\beta 2 \cdot vt-1+(1-\beta\ 2) \cdot gt2$

سپس بایوس را اصلاح میکند :

اصلاح شده مومنت اول:

$$m_t = \frac{m_t}{1-\beta_1^{\ t}}$$

اصلاح شده مومنت دوم:

$$v_t = \frac{v_t}{1-\beta_2^{\ t}}$$

در نهایت داریم:

$$\theta_t - \alpha . \frac{(\beta_1 . m_{t-1} + (1-\beta_1) g_t)}{\sqrt{v_t} + \epsilon}$$

- **الگوریتم RMSprop**

این الگوریتم با نام کاملتر Root Mean Squared Propagation

گرادیان در این الگوریتم به صورت زیر آپدیت می‌شود:

$$w_i(t+1) = w_i(t) - lr(\frac{\partial L}{\partial w_i}(t)); \; lr = -\frac{lr_0}{\sqrt{G_i(t)} + e}$$

$$G_i(t) = \beta G_i(t-1) + (1-\beta)((\frac{\partial L}{\partial w_i} t))^2; \; G_i(0) = 0$$

این الگوریتم با تغییر هوشمندانه نرخ یادگیری، به صورت per weight عمل می‌کند. یعنی هر کدام از وزن‌های موجود در شبکه، بر اساس تابع گرادیان خود قدم‌هایش را تعیین می‌کند. از آنجایی که وزن‌های مختلف برای یک مدل با بازه‌های متفاوت همزمان به نقطه بهینه نمی‌رسند، حائز اهمیت است.

برای استفاده از این بهینه ساز ها در مدل، آنها را به صورت زیر تعریف کرده و در هر مرحله، پارامترهای مدل را به آنهای برای بهینه سازی می‌دهیم.

```
optimizers = {
```

```
    'Adam': optim.Adam,
    'NAdam': optim.NAdam,
    'RMSprop': optim.RMSprop
}
```

بررسی نتایج

همانطور که در نمودارهای زیر مشخص است، عملکرد کلی بهینه‌ساز Nadam که حالت پیشرفته‌تر Adam می‌باشد، از Adam بهتر عمل کرده است. بهینه‌ساز Adam که خود حالت بهینه‌شده‌ی RMSprop است، از آن عملکرد بهتری داشته.

```
Test Result for Adam Optmizer
Test Loss: 0.3626
              precision    recall  f1-score   support

 T-shirt/top       0.83      0.81      0.82      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.78      0.81      0.79      1000
       Dress       0.87      0.89      0.88      1000
        Coat       0.80      0.82      0.81      1000
      Sandal       0.98      0.95      0.96      1000
       Shirt       0.69      0.65      0.67      1000
     Sneaker       0.93      0.96      0.94      1000
         Bag       0.98      0.97      0.97      1000
  Ankle boot       0.95      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 13. **نتایج بهینه ساز Adam**

```
Test Result for NAdam Optmizer
Test Loss: 0.3546
              precision    recall  f1-score   support

 T-shirt/top       0.79      0.88      0.83      1000
     Trouser       0.98      0.97      0.98      1000
    Pullover       0.81      0.77      0.79      1000
       Dress       0.89      0.88      0.89      1000
        Coat       0.80      0.82      0.81      1000
      Sandal       0.97      0.95      0.96      1000
       Shirt       0.69      0.64      0.67      1000
     Sneaker       0.93      0.96      0.95      1000
         Bag       0.97      0.97      0.97      1000
  Ankle boot       0.95      0.96      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
```

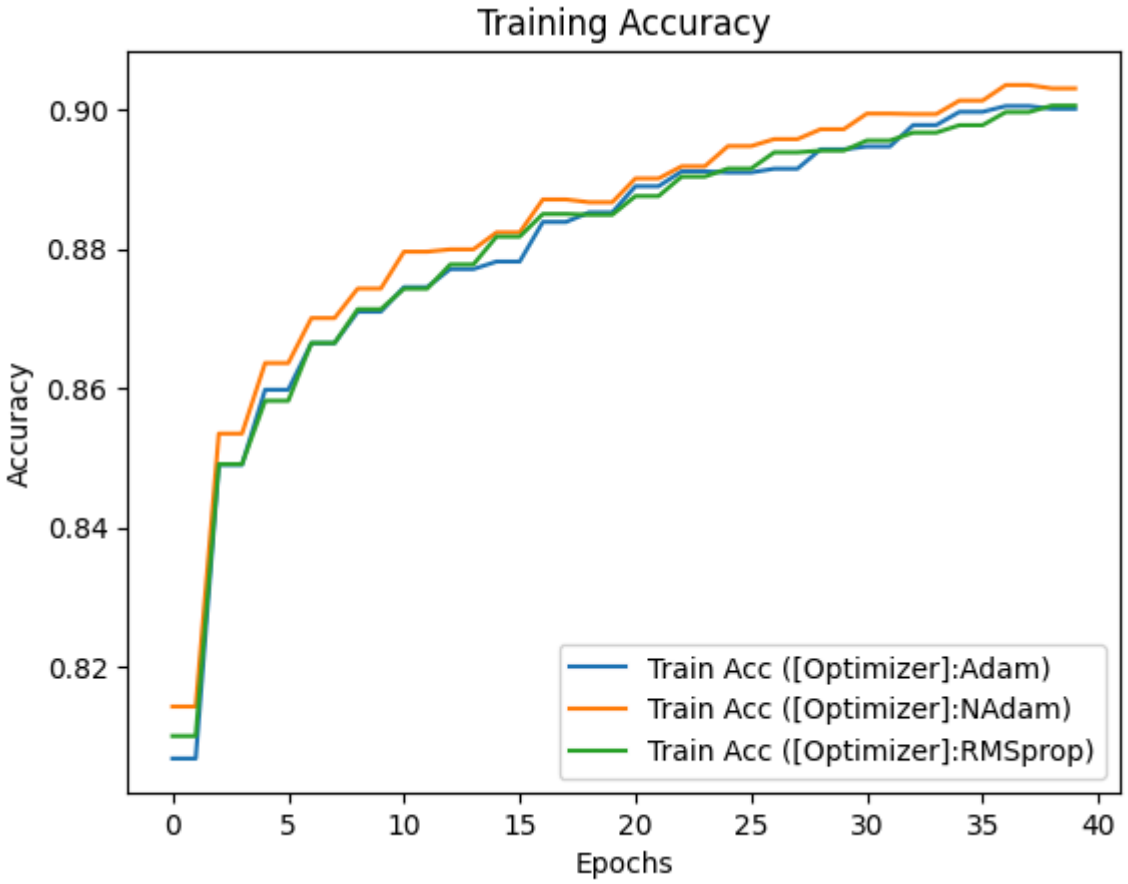
شکل 14. **نتایج بهینه ساز Nadam**

```
Test Result for RMSprop Optmizer
Test Loss: 0.4023
              precision    recall  f1-score   support

 T-shirt/top       0.86      0.79      0.83      1000
     Trouser       0.99      0.96      0.98      1000
    Pullover       0.82      0.75      0.78      1000
       Dress       0.88      0.88      0.88      1000
        Coat       0.77      0.83      0.80      1000
      Sandal       0.98      0.93      0.95      1000
       Shirt       0.65      0.74      0.69      1000
     Sneaker       0.91      0.97      0.94      1000
         Bag       0.98      0.95      0.96      1000
  Ankle boot       0.96      0.95      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
```

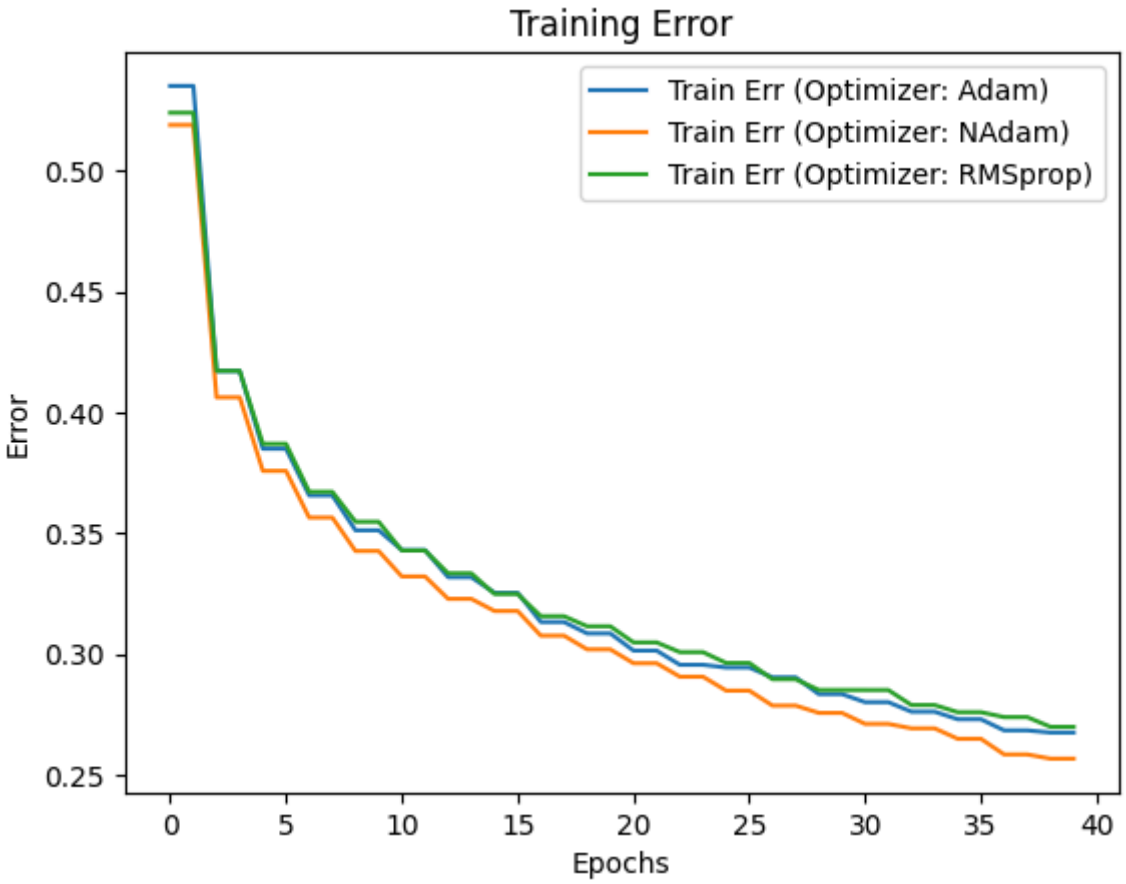
شکل 15. **نتایج بهینه ساز RMSprop**

نمودار های زیر، سرعت همگرایی و دقت کلی مدل با سه بهینه ساز که الگوریتم بازگشت به عقب را برای پارامترهای مدل پیاده میکنند، نشان میدهد. همانطور که در بخش تئوری ذکر شد، Nadam عملکرد بهتری نسبت به Adam دراد و علت آن این است که روش بهینه تر شده Adam میباشد. همینطور بهینه ساز Adam که بهینه یافته تر و پیشرفته تر از RMSprop است، نتیجه بهتری را هم در دقت و هم در خطا در کل ایپاک ها نشان داده است.

از آنجایی که دقت روی داده تست برای مدل‌ها روی حدود 88 می باشد، مدل ها بعد از اپیاک 15 به بعد، شروع به overfit روی داده ترین کرده اند.
نمودار های دقت و خطای این سه بهینه ساز به شرح زیر ترسیم شده است که توضیاحات فوق را تایید میکند.



شکل 16. **مقایسه نمودار دقت برای سه بهینه ساز**



شکل 17. **مقایسه نمودار خطا برای سه بهینه ساز**

سوال) تاثیر جست و جوی بیزی یا روش های دیگر روی بهبود فرایند

جستجوی بیزی و سایر روش‌های بهینه‌سازی هایپرپارامترها می‌توانند تاثیر قابل‌توجهی بر عملکرد الگوریتم (Backpropagation) در شبکه‌های عصبی داشته باشند. این روش‌ها با تنظیم مقادیر بهینه برای هایپر پارامترهای شبکه (مانند نرخ یادگیری، اندازه دسته، و تعداد لایه‌ها) به بهبود آموزش و همگرایی شبکه کمک می‌کنند. در بخش 4 به توضیح بیشتر راجب این مورد پرداخته شده است.

در روش جست و جوی بیزی، با مدل سازی احتمالاتی از نتایج هایپر پارامتر ها بهترین و بهینه ترین مقادیر مربوط به این پارامتر ها را پیدا میکند.

در واقع روش این روش بدین شکل است که با استفاده از یک تابع هدف و تابع کسب یا به عبارتی Acquisition Func، با عملکردی قابل قبول، ترکیبیات جدیدی از پارامترهای مربوطه را با مدل امتحان کرده و با هر بار امتحان، مدل احتمالاتی مذکور را آپدیت میکند. در نتیجه در تعداد ران کمتری به بهینه ترین مقادیر میرسد.

اگر مدل ما هزینه زمانی اجرای زیادی داشته باشد، از این روش استفاده میکنیم.

۴-۱. بررسی هایپر پارامترهای مختلف

در این بخش، اثر سه نوع هایپر پارامتر نرخ dropout، تعداد نورون و مقدار learning rate را بررسی می‌کنیم.

برای هر پارامتر، سه مقدار مختلف در نظر گرفته و نحوه اثر گذاری را مشخص می‌کنیم. توجه کنید همگی موارد در تعداد اپاک 20 بررسی شده اند.

- **نرخ Dropout**

در شبکه‌های عصبی، Dropout یک تکنیک تنظیم است که به جلوگیری از بیش‌برازش کمک می‌کند. نرخ Dropout یک هایپر پارامتر است که مشخص می‌کند چه درصدی از نورون‌ها در طول فرآیند آموزش به‌صورت تصادفی غیرفعال می‌شوند. این نرخ معمولاً بین ۰ تا ۰.۵ است. مثلاً اگر نرخ Dropout برابر با ۰.۳ باشد، به‌طور متوسط ۳۰ درصد از نورون‌ها در هر گام از آموزش غیرفعال می‌شوند.

برای سه مقدار 0.2، 0.4 و 0.8 روی مدل با 128 نود در لایه میانی با learning rate برابر 0.01، بررسی زیر صورت گرفته است.

جدول 1. **نتایج دقت و خطا برای سه مدل**

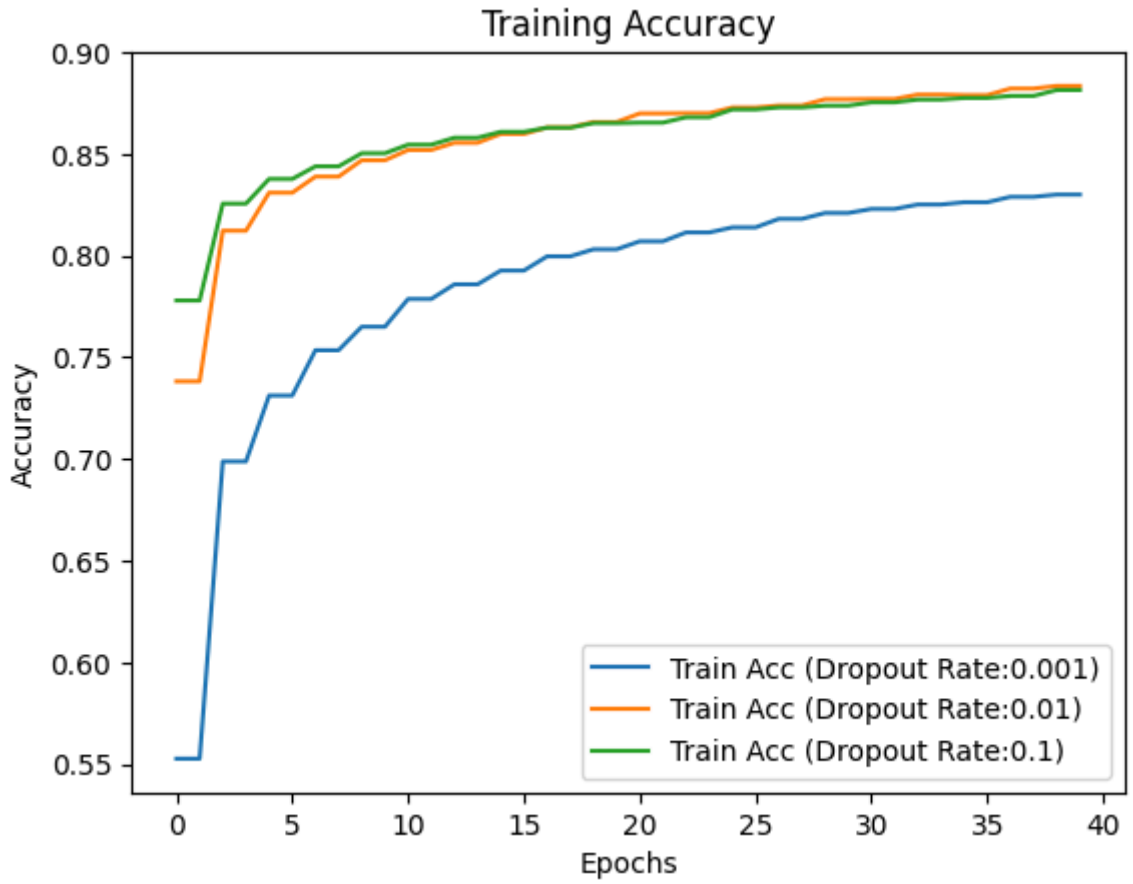
	دقت	خطا
Train 0.2	0.838	0.457
Test 0.2	0.834	0.824
Train 0.4	0.829	0.486
Test 0.4	0.830	0.470
Train 0.8	0.777	0.652
Test 0.8	0.817	0.506

در همگی مدل‌ها دقت تست مقداری کمتر از دقت ترین بوده که منطقی می‌باشد.

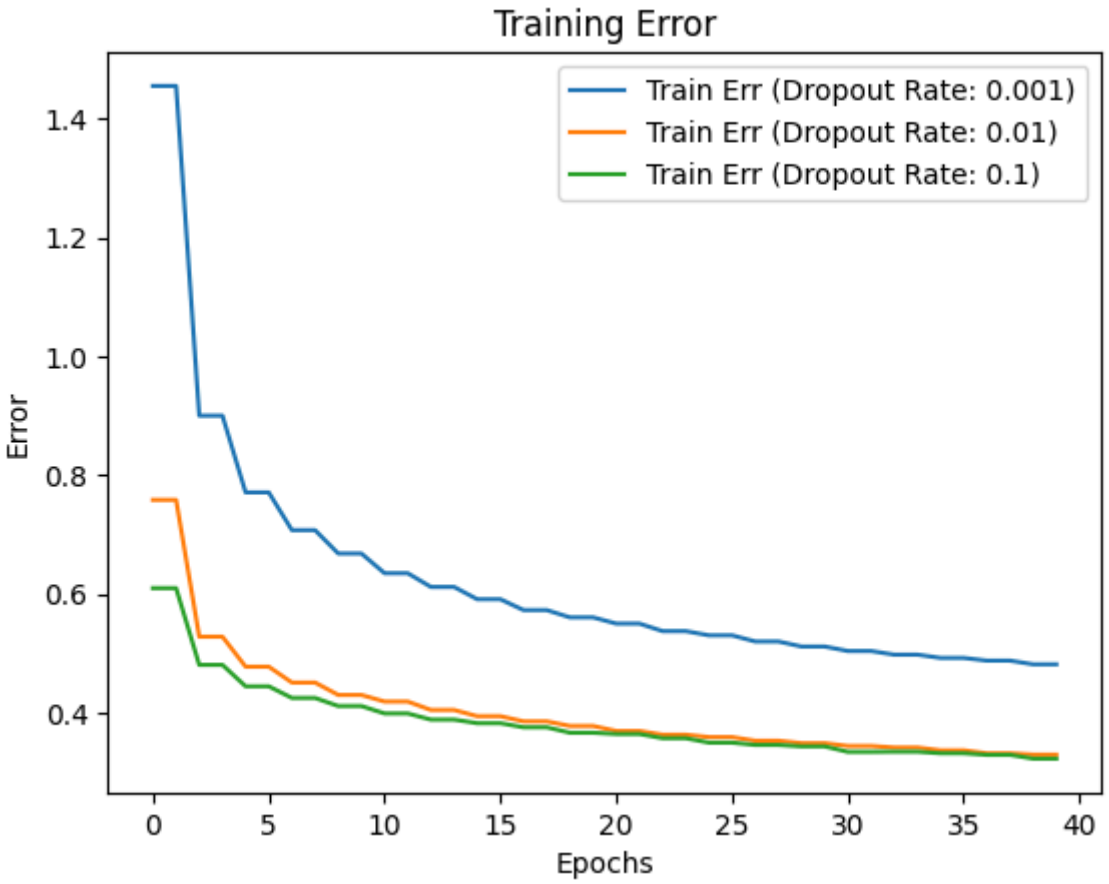
همانطور که مشخص است، در این مدل که نسبتا تعداد نورون مناسبی برای داده ما دارد، نرخ 0.8، عملکرد مدل را ضعیف تر کرده، و نتایج کمتری نشان داده است.

بین دو مدل دیگر، عملکرد خیلی نزدیکی داشته اند اما مدل با نرخ 0.2 نتایج بهتری داده است. بنابراین نتیجه می‌گیریم برای داده‌ی کنونی، مقدار نرخ خیلی بالای drop out کارا نبوده و تصمیم‌گیری های مدل را بی اثر می‌کند و به نفع ما نیست. در واقع در این مدل، overfit خیلی زیادی نداریم و مقادیر نرخ پایین برای این پارامتر پاسخگو میباشد.

نمودار های زیر، به خوبی عملکرد سه مدل را در تعداد 40 ایپاک نشان می‌دهد.



شکل 18. **نمودار دقت برای سه مقدار هایپر پارامتر Dropout**



شکل 19. **نمودار خطا برای سه مقدار هایپر پارامتر Dropout**

- **تعداد نورون ها**

در یک شبکه عصبی چندلایه‌ای (MLP)، این هایپر پارامتر مشخص می‌کند که هر لایه نهان (لایه‌ای بین لایه ورودی و لایه خروجی) چند نورون یا واحد محاسباتی دارد. تعداد نورون‌های بیشتر می‌تواند به مدل کمک کند که الگوهای پیچیده‌تر را یاد بگیرد، اما ممکن است منجر به پیچیدگی و هزینه محاسباتی بالاتر و احتمال بیش‌برازش شود. انتخاب بهینه تعداد نورون‌ها برای هر لایه نهان یکی از چالش‌های مهم در طراحی معماری MLP است.

برای سه مقدار 32، 64، و 128 روی مدل با 128 نود در لایه میانی با learning rate برابر 0.01، و نرخ dropout 0.4، بررسی زیر صورت گرفته است.

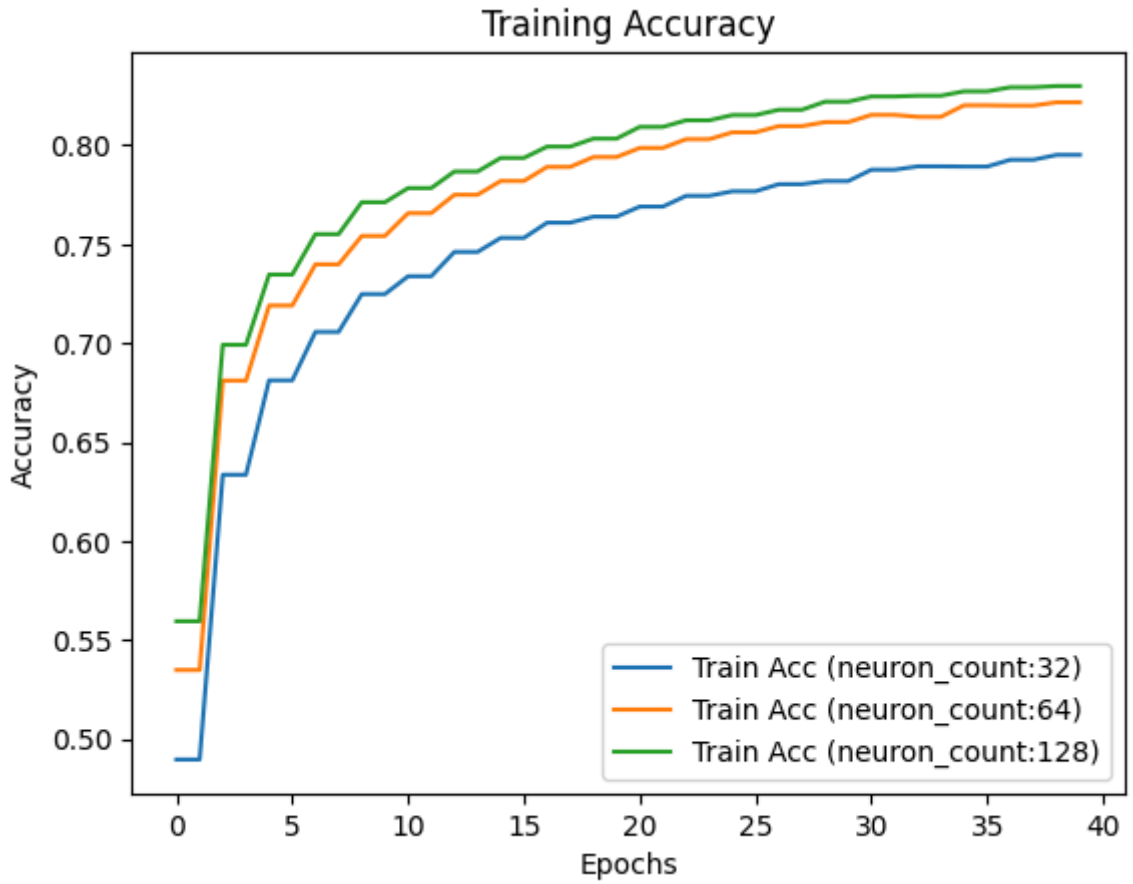
جدول 2. **نتایج دقت و خطا برای سه مدل**

	دقت	خطا
Train 32	0.795	0.588
Test 32	0.819	0.499
Train 64	0.822	0.513
Test 64	0.824	0.476
Train 128	0.830	0.483
Test 128	0.833	0.468

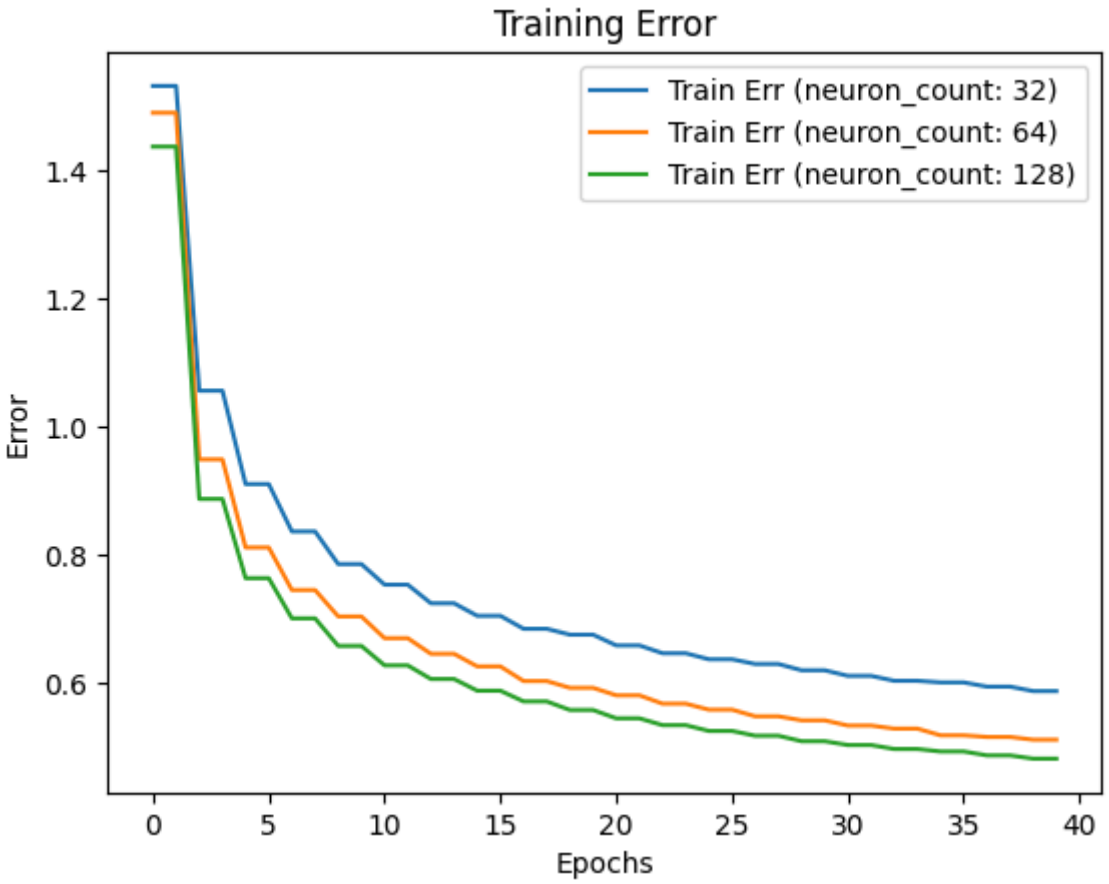
همانطور که از نتایج مشخص است، هرچقدر batch size بالاتری داشته باشیم، عملکرد مدل بهتر خواهد بود.

اندازه batch size تعیین می‌کند که مدل چند نمونه را قبل از به‌روزرسانی وزن‌ها ببیند. دسته‌های بزرگ‌تر می‌توانند تخمین گرادیان پایدارتر و نماینده‌ای را برای هر به‌روزرسانی ارائه دهند و با توجه به ماهیت داده‌ی ما، این نتیجه حاصل شده است.

نمودار های زیر، به خوبی عملکرد سه مدل را در تعداد 40 ایپاک نشان می‌دهد.



شکل 20. **نمودار دقت برای سه مقدار هایپر پارامتر تعداد نورون لایه میانی**



شکل 21. **نمودار خطا برای سه مقدار هایپر پارامتر تعداد نورون لایه میانی**

- **مقدار learning rate**

نرخ یادگیری، یک هایپر پارامتر کلیدی در الگوریتم‌های بهینه‌سازی است که مشخص می‌کند اندازه گام‌های به‌روزرسانی وزن‌ها در شبکه عصبی چقدر باشد. نرخ یادگیری کوچک‌تر منجر به بهبود آهسته اما دقیق‌تر شبکه می‌شود، در حالی که نرخ یادگیری بزرگ‌تر به مدل اجازه می‌دهد سریع‌تر آموزش ببیند، اما ممکن است باعث نوسانات زیاد یا عدم همگرایی شود.

برای سه مقدار 0.001، 0.01، و 0.1 روی مدل با 128 نود در لایه میانی با نرخ dropout برابر 0.4، بررسی زیر صورت گرفته است.

جدول 3. **نتایج دقت و خطا برای سه مدل**

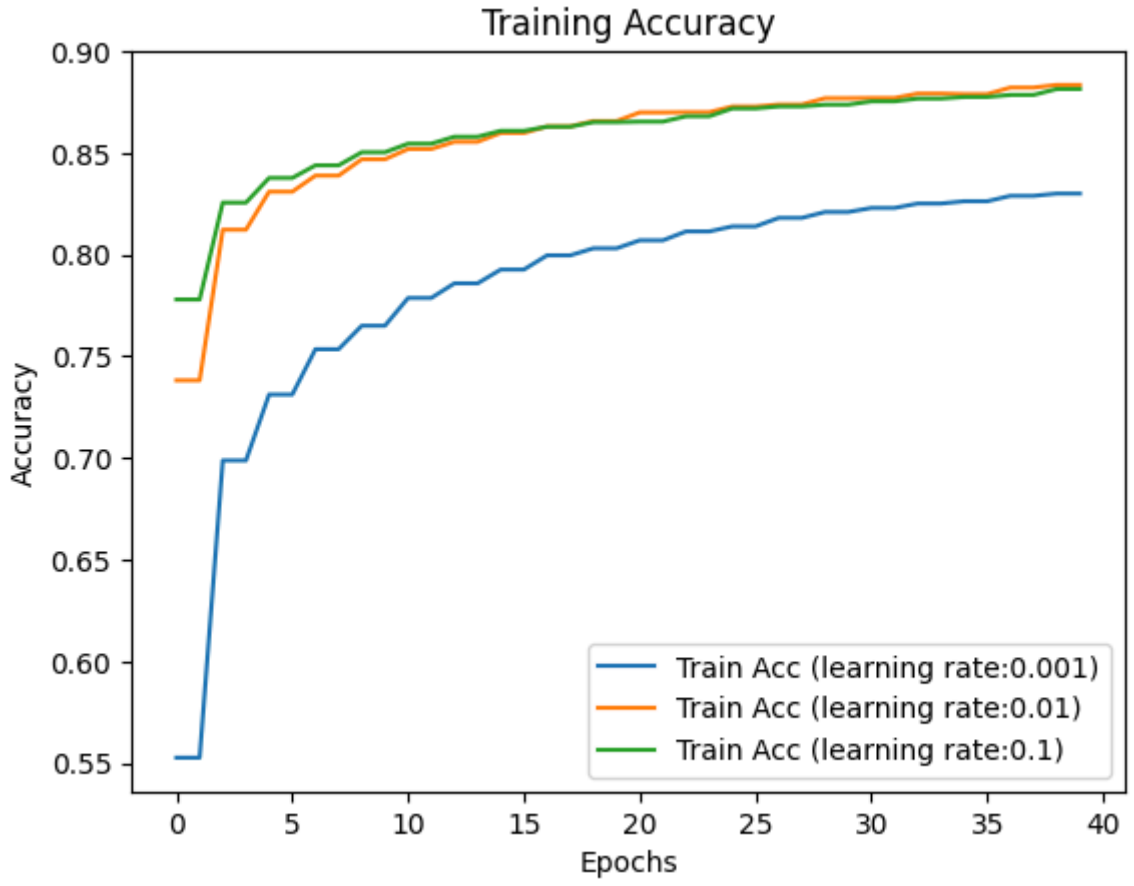
	دقت	خطا
Train 0.001	0.830	0.481
Test 0.001	0.831	0.466
Train 0.01	0.883	0.329
Test 0.01	0.878	0.346
Train 0.1	0.881	0.322
Test 0.1	0.869	0.365

همانطور که از نتایج مشخص است، لرنینگ ریت خیلی بزرگ یادگیری مدل را سریعتر میکند. در این سوال، بهترین عملکرد را مقدار میانی یعنی 0.01 داشته که در بقیه موارد از این مقدار استفاده شده است. با توجه به ساختار مدل شبکه و ماهیت دیتا در این سوال و تعداد اپیاک، ریت 0.1 از 0.001 عملکرد بهتری داشته یعنی در این تنظیمات، بهتر بوده مدل سریعتر پارامتر های خود را یاد بگیرد.

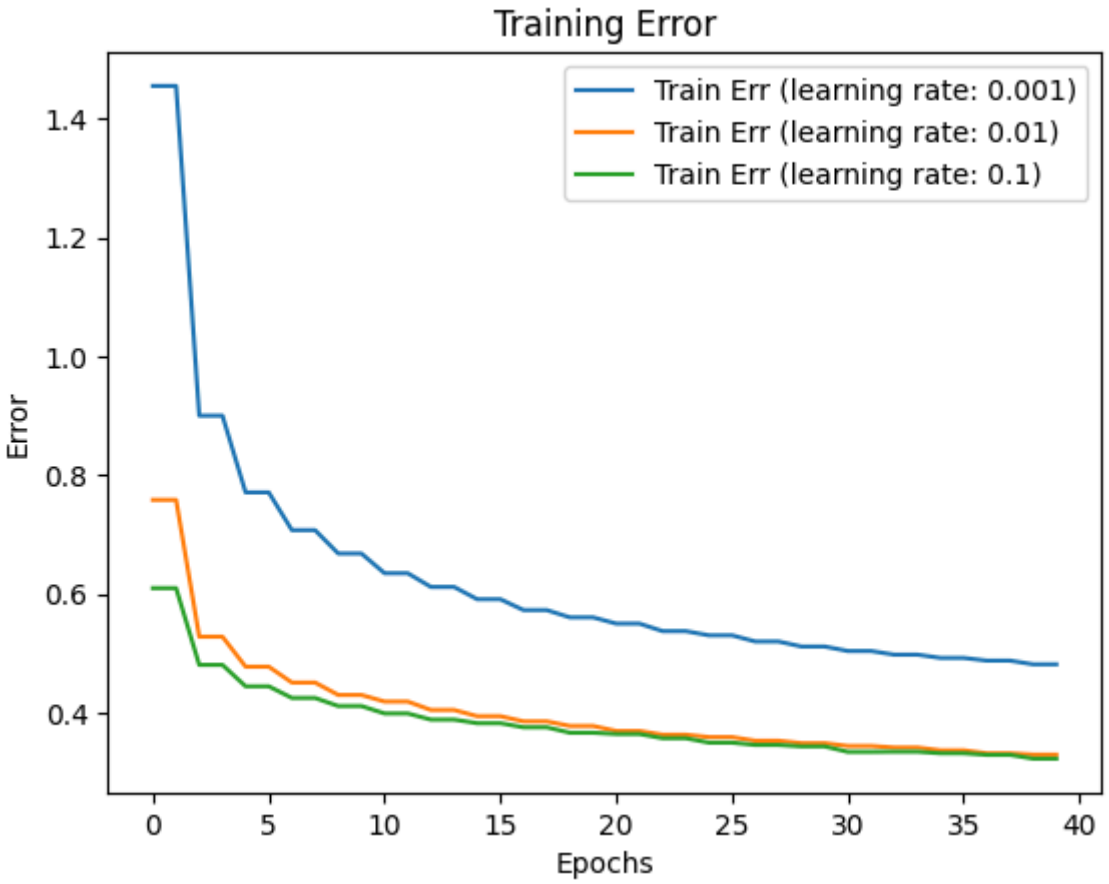
لرنینگ ریت خیلی کوچک هم به تعداد اپیاک بیشتری برای یادگیری مدل احتیاج خواهد داشت تا مدل بهتری ارائه دهد اما در 40 اپیاک، نتیجه بهتری نگرفته است.

نمودار های زیر، به خوبی عملکرد سه مدل را در تعداد 40 اپیاک نشان می‌دهد.

همانطور که مشخص است، سرعت همگرایی در بهینه ساز های قوی تر، سریع تر میباشد.



شکل 22. **نمودار دقت برای سه مقدار هایپر پارامتر مقدار learning rate**

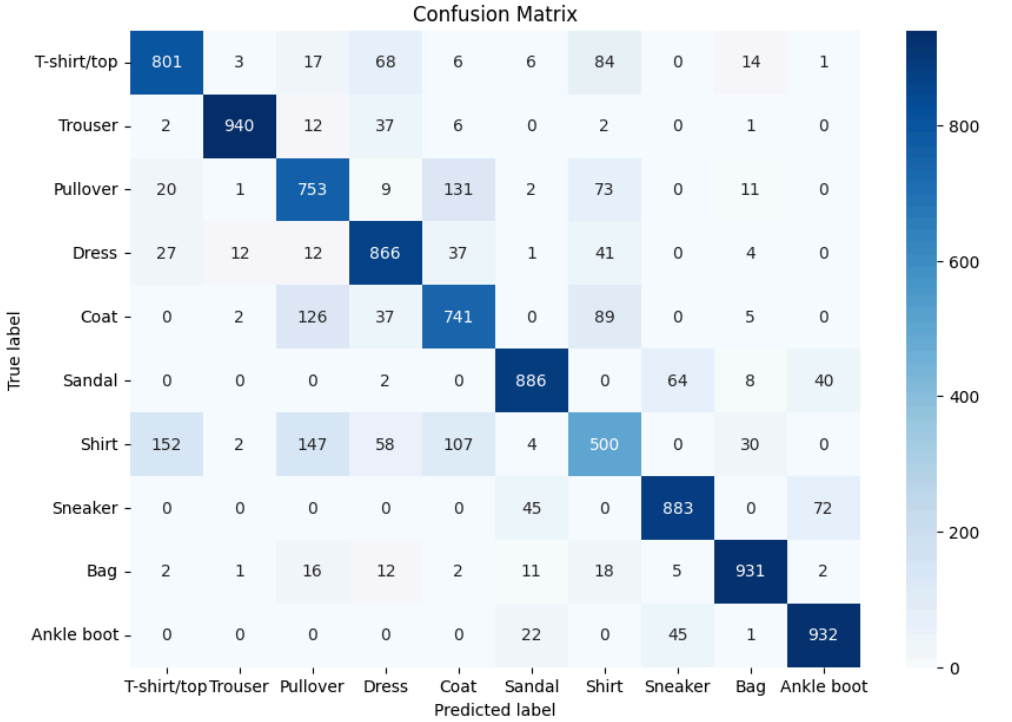


شکل 23. **نمودار خطا برای سه مقدار هایپر پارامتر مقدار learning rate**

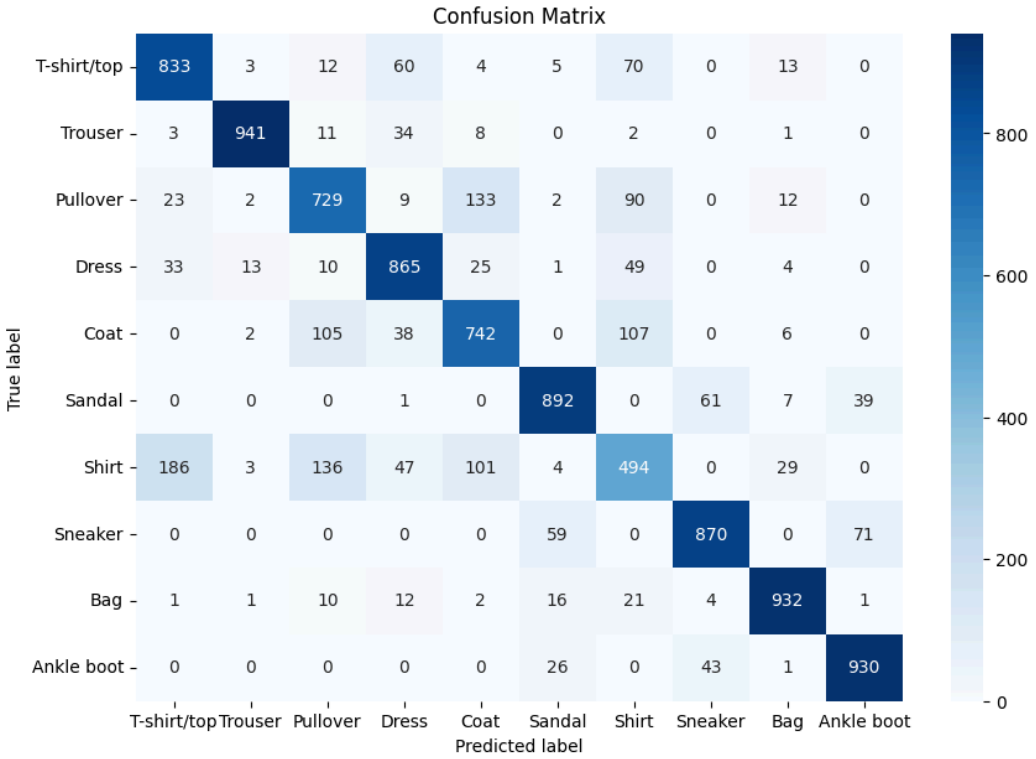
سوال) چگونه روش های بهینه سازی هایپر پارامتر مانند جستوجوی تصادفی می توانند به انتخاب بهترین ترکیب ها کمک کنند؟

روش‌های بهینه‌سازی هایپر پارامتر، مانند **جستجوی تصادفی** (Random Search)، به انتخاب بهترین ترکیب‌های هایپر پارامترها برای یک مدل کمک می‌کنند تا عملکرد آن در داده‌های جدید بهینه شود. در این روش که مثالی از آن در بخش 4 آمده، چندین پارامتر با مقادیر ست میشوند و مدل، با امتحان کردن پارامترها و ترکیب کردن آنها، بهترین ترکیب از فضای تحمیل شده را خروجی میدهد.

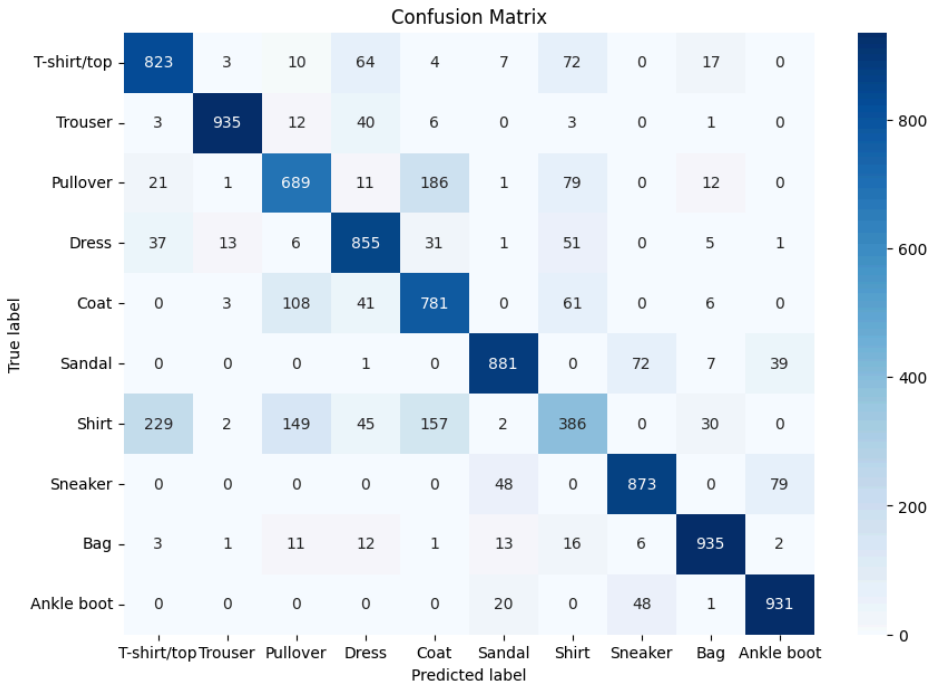
سوال) تحلیل کنید تغییر هر کدام از هایپر پارامترهای چه تغییری روی کلاس هایی که باهم اشتباه گرفته می‌شوند دارند؟ چرا؟



شکل 24. **ماتریس آشفتگی مدل با دراپ اوت 0.2**

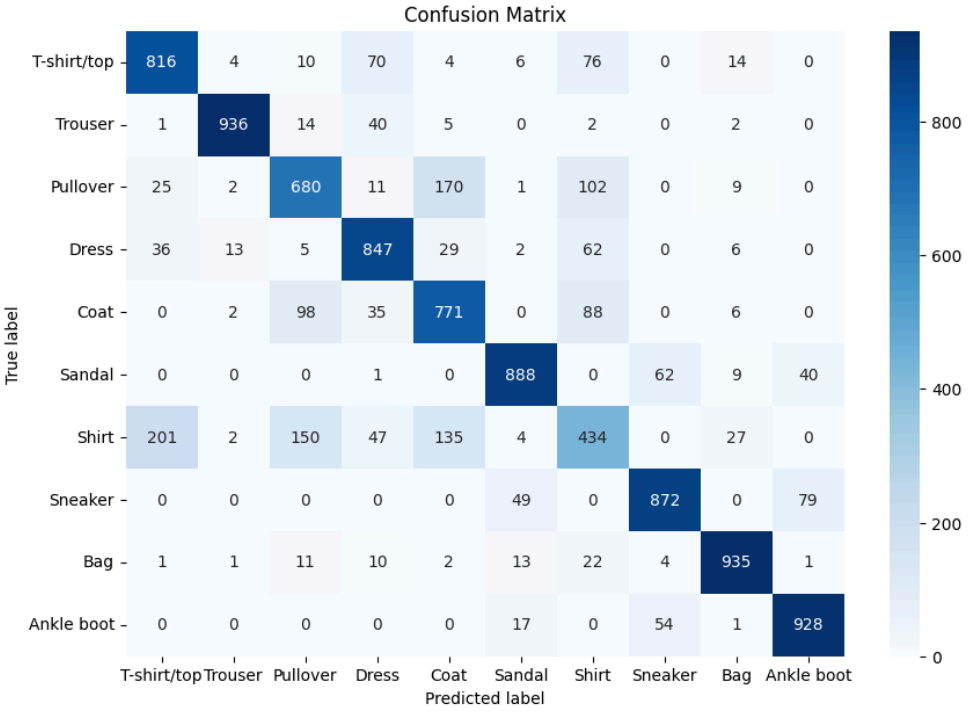


شکل 25. **ماتریس آشفتگی مدل با دراپ اوت 0.4**

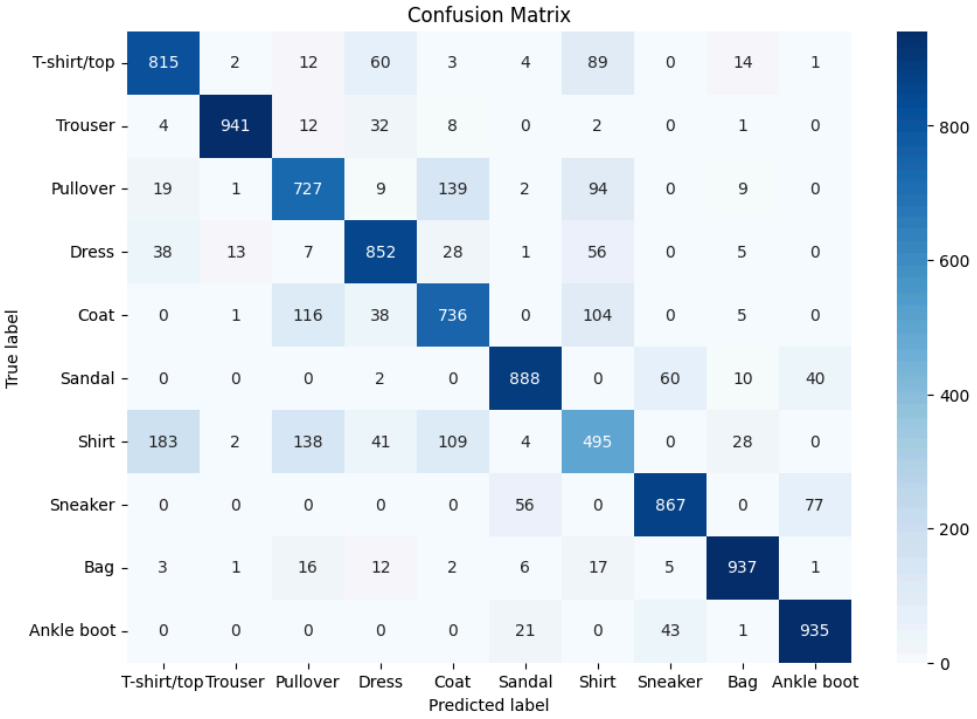


شکل 26. **ماتریس آشفتگی مدل با دراپ اوت 0.8**

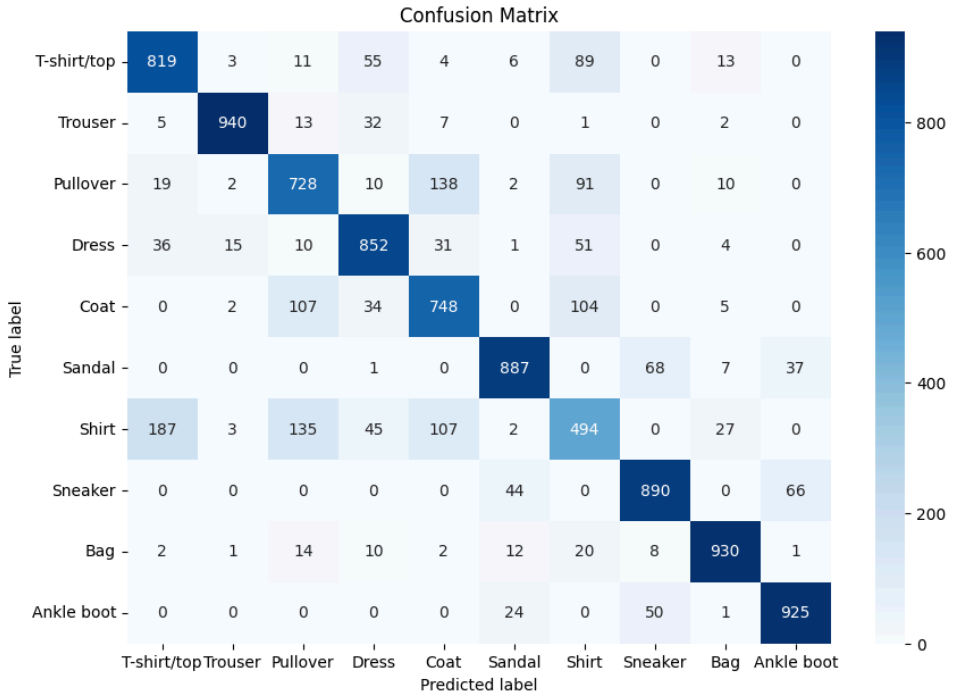
لباس از نوع Shirt را بررسی می‌کنیم که در مدل به do 0.2، به تعداد 152 عکس به اشتباه، T shirt/top پیشبینی شده اند. هرچقدر این ریت بالاتر رفته باشد، این اشتباه به تعداد بیشتری انجام شده است.



شکل 27. **ماتریس آشفتگی مدل با تعداد نورون 32**

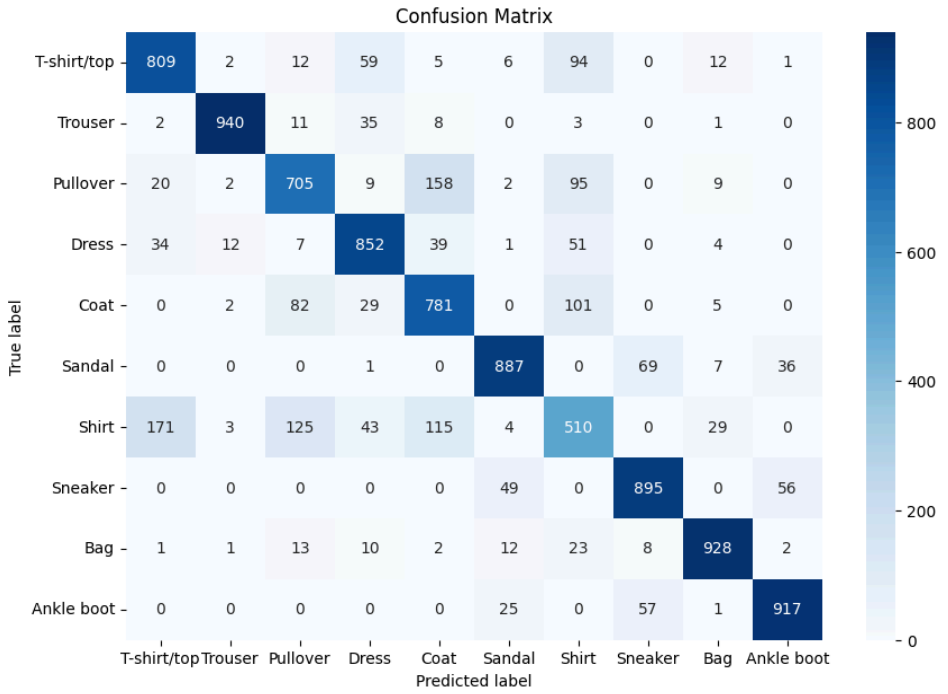


شکل 28. **ماتریس آشفتگی مدل با تعداد نورون 64**

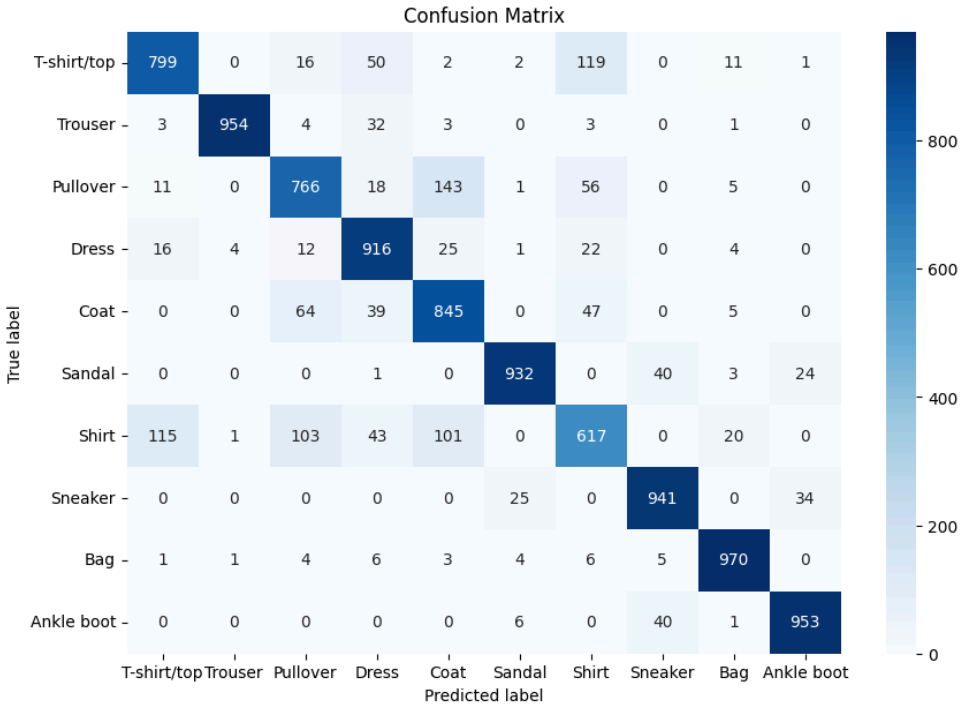


شکل 29. **ماتریس آشفتگی مدل با تعداد نورون 128**

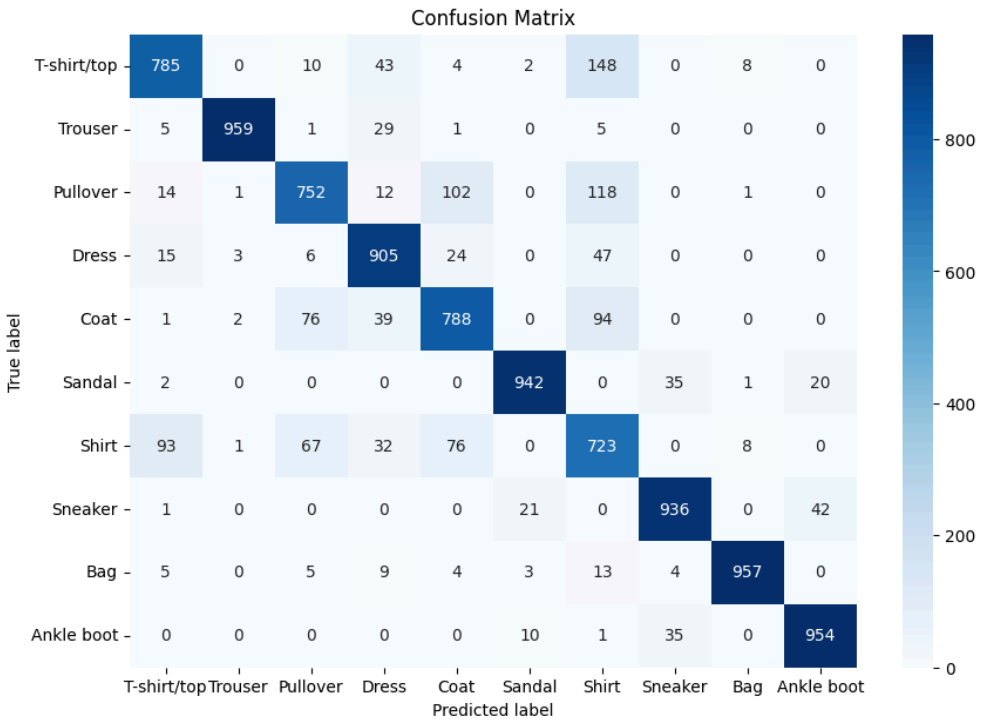
لباس از نوع shirt که به اشتباه t shirt حدس زده شده است، در مدل اول روی 201 عکس این خطا انجام شده که در دو مدل دیگر، میزان خطای مربوطه کاهش یافته است. در نمودار های دقت و خطا دو مدل با 64 و 128 نود نهان، عملکرد نزدیک تری نسبت به مدل سوم داشته اند.



شکل 30. **ماتریس آشفتگی مدل با لرنینگ ریت 0.001**



شکل 31. **ماتریس آشفتگی مدل با لرنینگ ریت 0.01**



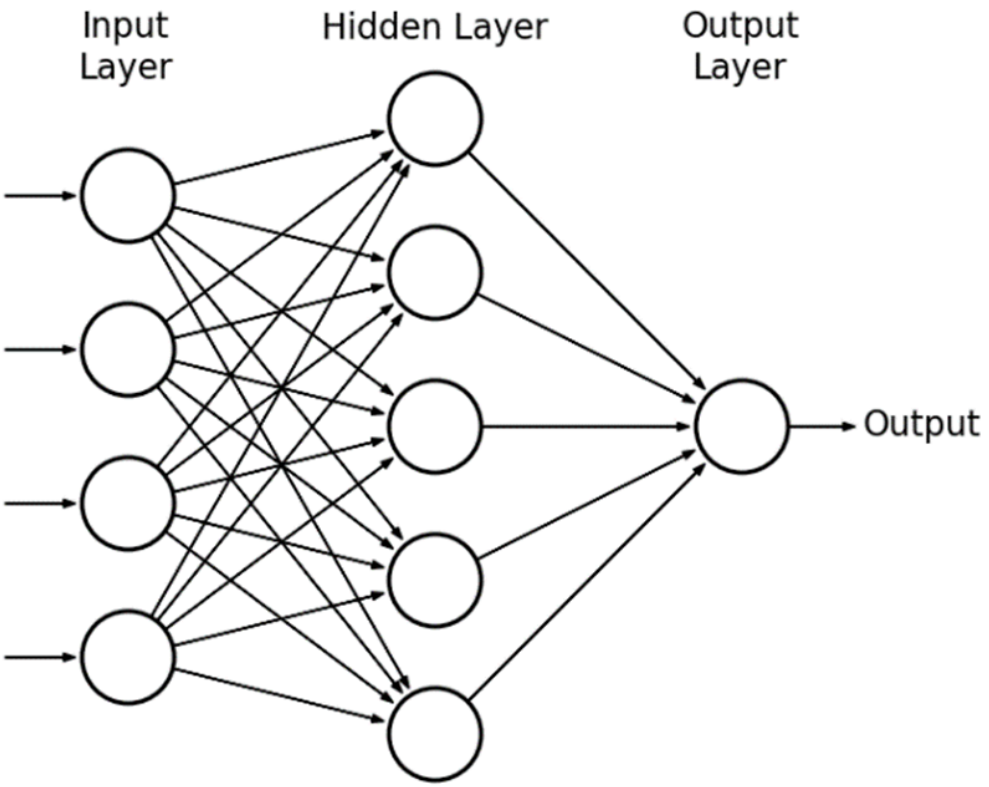
شکل 32. **ماتریس آشفتگی مدل با لرنینگ ریت 0.1**

در مدل اول لباس از نوع shirt که به اشتباه t shirt حدس زده شده است، تعداد 171 دارد. در مدل با نرخ 0.01، کمترین میزان یعنی 115 و در مدل با نرخ 0.01، به میزان 93 است که تحلیل های ما را تایید میکند.

پرسش ۲ - آموزش و ارزیابی یک شبکه عصبی ساده

۲-۱. آموزش یک شبکه عصبی

در این تمرین باید یک شبکه عصبی ساده را از پایه نوشته و آموزش داده و نهایتا ان را ارزیابی کنیم.



شکل 33. **تصویر یک mlp با یک لایه پنهان**

شبکه عصبیی که ما باید طراحی کنیم 30 نورون در لایه پنهان و 1 نورون در لایه خروجی دارد زیرا تسک هدف یک تسک رگرشنی است و اگر تسک کلسیفیکیشن بود تعداد نورون های لایه خروجی به تعداد کلاس ها تبدیل می‌شد. همچنین اکتیویشن فانکشن مورد استفاده در لایه پنهان تانژانت هیپربولیک است و در لایه خروجی ایدنتیتی فانکشن می‌باشد. همچنین در پیاده سازی این تکلیف تنها مجاز به استفاده از کتابخانه Numpy بودیم.

الف) در این قسمت باید تابع forward را می نوشتیم که جهت پیشبینی توسط مدل استفاده شود:

```
def forward(X, W1, W2):
    Z = tanh(np.dot(X, W1.T))
    y_pred = np.dot(Z, W2.T)
    return y_pred, Z
```

شکل 34. تصویر کد تابع forward

در پیاده سازی این تابع از محاسبات ماتریسی قابل انجام در Numpy استفاده کردیم ماتریس W1 یک ماتریس M*D است که D بعد ماتریس فیچر های ورودی است.در نتیجه با محاسبه دات پروداکت ترنسپوز W1 و X هر ورودی را در ضرایب متناظر ضرب کرده و به هر نورون در لایه پنهان Xi*W1i را داده

ایم . پس از آن اکتیویشن فانکشن تانژانت هیپربولیک را اعمال کرده و در نهایت به کمک روشی مشابه خروجی را با ضرب وزن ها در مقادیر لایه پنهان محاسبه می کنیم و از آنجا که اکتیویشن فانکشن لایه خروجی ایدنتیتی فانکشن است همین دات پروداکت را به عنوان خروجی باز میگردانیم .

ب) در این قسمت هدف پیاده سازی تابع بکوارد است که جهت ترین کردن مدل استفاده شود .

```python
def backward(X, y, M, iters, lr):
    N, D = X.shape
    W1 = np.random.randn(M, D) * 0.01
    W2 = np.random.randn(1, M) * 0.01
    error_over_time = []

    for _ in range(iters):
        i = np.random.randint(0, N)
        X_i = X[i:i+1]
        y_i = y[i:i+1]

        y_pred, Z = forward(X_i, W1, W2)

        error = y_i - y_pred
        loss = np.mean(error ** 2)
        error_over_time.append(loss)

        dW2 = -2 * np.dot(error.T, Z)
        dZ = np.dot(error, W2) * tanh_derivative(Z)
        dW1 = -2 * np.dot(dZ.T, X_i)

        W1 -= lr * dW1
        W2 -= lr * dW2

    return W1, W2, np.array(error_over_time)
```

شکل 35. تصویر تابع backward

ابتدا وزن ها را به صورت رندوم با مقادیری کوچک مقدار دهی میکنیم تا W1 و W2 اولیه را بسازیم.
سپس در هر ایتریشن به کمک فانکشن forward ورودی ها را پردیکت میکند و لاس را که از mse استفاده شده محاسبه میکنیم و لاس هر ایپاک را در ارور اور تایم ذخیره میکنیم .
سپس فرایند backpropagation را انجام میدهیم که مشتق هر کدام از وزن ها را حساب کرده و وزن ها را به کمک ان اپدیت میکنیم .

۲-۲. آزمون شبکه عصبی بر روی یکی مجموعه داده

در این بخش هدف پیشبینی کیفیت شراب از روی داده های دیتاست فراهم شده به کمک مدل پیاده سازی شده است. ابتدا داده ها را باید به دو مجموعه ترین و تست تقسیم کرده و سپس داده ها را استاندارد کنیم و یک بعد به عنوان بایاس هر دیتا پوینت اضافه کنیم.

```
X = df.drop(columns=['quality']).values
y = df['quality'].values.reshape(-1, 1)

np.random.seed(13)
shuffled_indices = np.random.permutation(len(X))
X, y = X[shuffled_indices], y[shuffled_indices]
```

```
split_index = len(X) // 2
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]
```

```
X_train_mean = X_train.mean(axis=0)
X_train_std = X_train.std(axis=0)
X_test_mean = X_test.mean(axis=0)
X_test_std = X_test.std(axis=0)
X_train = (X_train - X_train_mean) / X_train_std
X_test = (X_test - X_test_mean) / X_test_std
```

```
X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
```

شکل 36. تصویر یک mlp با یک لایه پنهان

در قطعه کد بالا همانطور که مشاهده می‌شود ابتدا X و y را تشکیل می دهیم که ورودی ها و فیلد هدف را جدا کنیم سپس به کمک تولید اینکس های شافل شده دیتا ست را شافل میکنیم تا به صورت رندوم داده ها در تست و ترین تقسیم شوند . و سپس تا میانه این داده های شافل شده به عنوان داده ترین و مابقی را به عنوان داده تست استفاده میکنیم. و در نهایت آنها را استاندارد کرده و ستون آخر را به داده ها اضافه کردیم .

سپس میبایست با 3 مقدار متفاوت برای لرنینگ ریت مدلی ترین کنیم و rmse و error_over_time را برای هر کدام رسم کنیم :

```python
def plot_losses(train_losses, title, epochs=50):
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, epochs + 1), train_losses, label='Training Loss', alpha=0.5)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'Training and Validation Loss Over Epochs for {title}')
    plt.legend()
    plt.show()
```

0.0s Python

```python
M = 30
iters = 1000
learning_rates = [0.01, 0.005, 0.001]
```

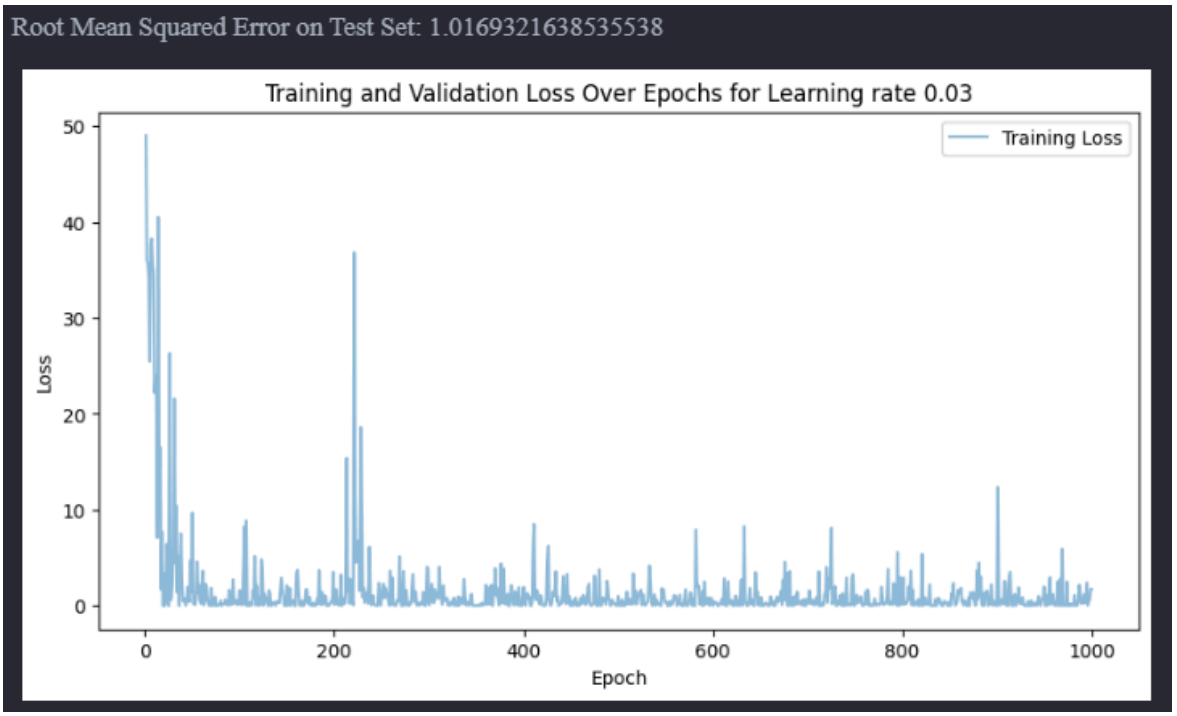
0.0s Python

```python
for learning_rate in learning_rates:
    W1, W2, error_over_time = backward(X_train, y_train, M, iters, learning_rate)
    y_pred_test, _ = forward(X_test, W1, W2)
    rmse = np.sqrt(np.mean((y_test - y_pred_test) ** 2))
    print(f"Root Mean Squared Error on Test Set: {rmse}")
    plot_losses(error_over_time, f'Learning rate {learning_rate}', iters)
```

شکل 37.تصویر پیاده سازی اجرا بر روی 3 لرنینگ ریت متفاوت

همانطور که در کد مشاهده میشود به ازا 3 لرنینگ ریت متفاوت 0.03 و 0.001 و 0.0005 مدل را ترین کردیم که به ترتیب نتایج را بررسی میکنیم :

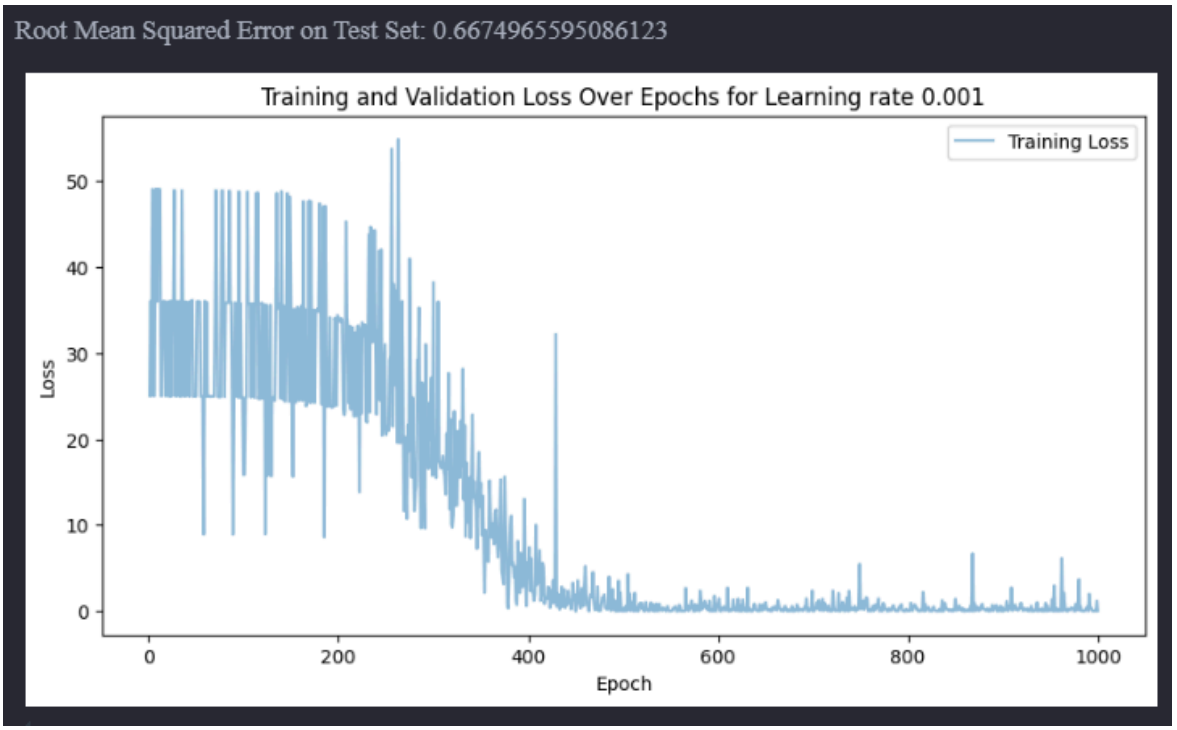
- learning_rate = 0.03 :



شکل 38. خروجی مدل mlp برای lr=0.03

همانطور که در پلات مشاهده میشود مدل کانورج کرده است و عملا بهترین وزن ها را حدودا به دست اورده است ولی به دلیل بالا بودن لرنینگ ریت و رویکرد بهینه سازی sgd نوسان بالایی داریم و بهترین انتخاب نیست .

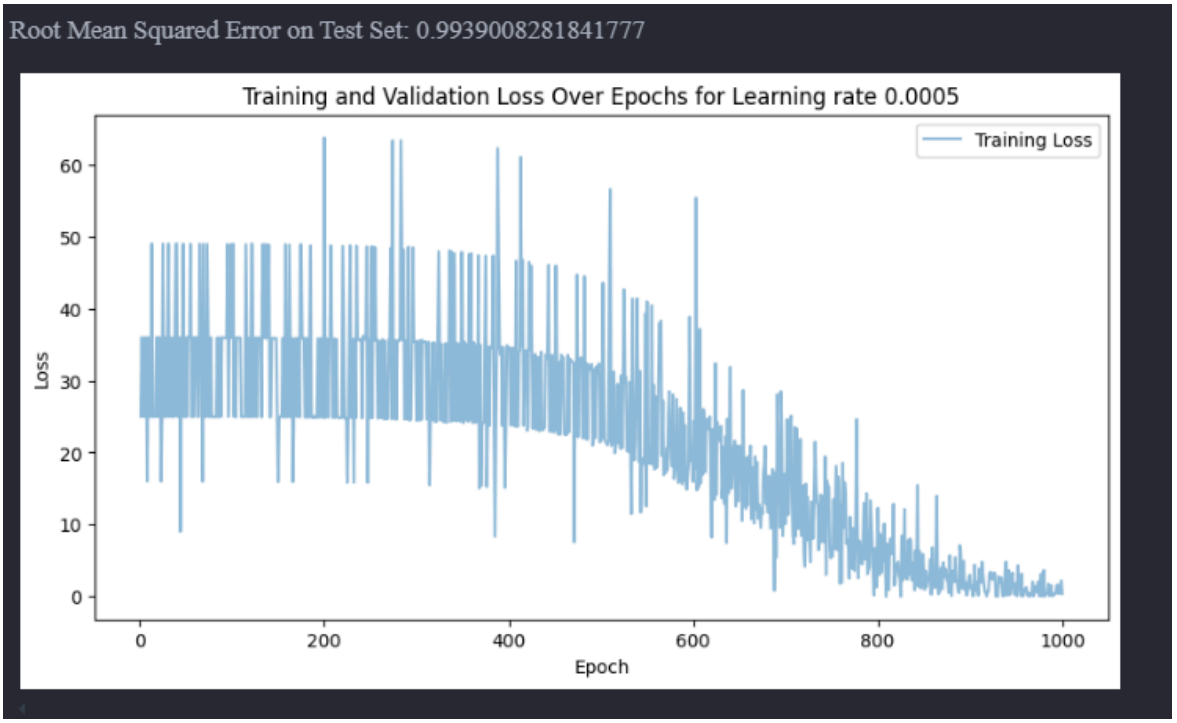
- learning_rate = 0.001 :



شکل 39. خروجی مدل mlp برای lr=0.001

همانطور که مشاهده میشود همچنان مدل کانورج کرده است و به دلیل پایین تر بود لرنینگ ریت دامنه نوسان نیز پایین تر است و بهترین rmse را از این مقدار گرفته ایم .

- learning_rate = 0.0005 :



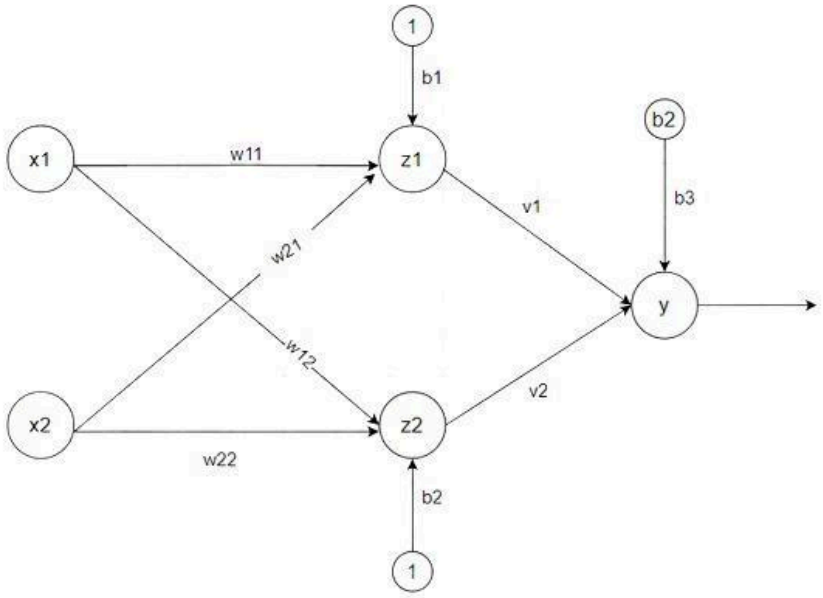
شکل 40. خروجی مدل mlp برای lr=0.0005

همانطور که مشاهده می‌شود این لرنینگ ریت مقداری زیادی کوچک است و مدل فرصت کاملا کانورج کردن را پیدا نکرده و علی رقم نوسان های کوچک تر با این تعداد ایتریشن کامل موفق به بهینه سازی نمیشود.

پرسش ۳ – Madaline

۳-۱. الگوریتم‌های MRI و MRII

الگوریتم Madaline که مخفف (Multiple Adaptive Linear Neuron) است. نوعی شبکه عصبی چند لایه که به صورت خاص برای مسائل دسته‌بندی توسعه یافته است. این شبکه از اولین شبکه‌هایی ست که به صورت سخت افزاری پیاده شده است. از loss back propagation برای آموزش استفاده می‌کند.
این الگوریتم از دو لایه اصلی تشکیل شده است:
لایه مخفی و لایه خروجی.
در لایه مخفی، چندین نورون وجود دارد که به صورت مستقل از بقیه نورون‌های این لایه یاد می‌گیرد. در نهایت خروجی نورون‌های لایه مخفی ترکیب می‌شوند تا خروجی نهایی مدل تولید شود.



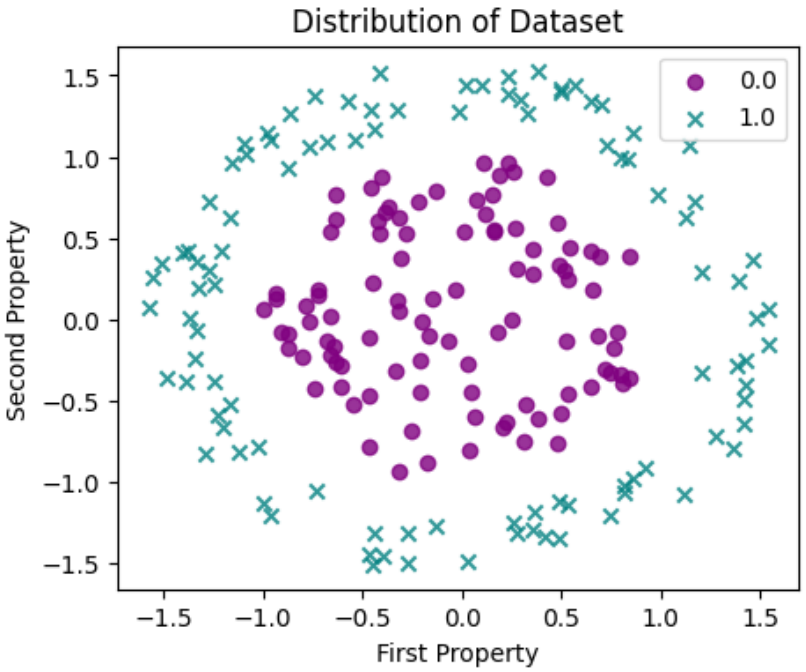
شکل 41. **مدل نمونه از Madaline**

در مدل فوق، یک Madaline ترسیم شده که لایه نهان آن از دو نورون z1 و z2 تشکیل شده است. نورون خروجی y در لایه خروجی قرار دارد.
در واقع به هر واحد از نورون های لایه میانی آن، یک Adaline می‌گویند.
در Madaline هر نورون از لایه مخفی دارای تابع گام یا Step Function به عنوان فعال‌سازی‌ست که فقط خروجی 1 یا -1 می‌دهد.
اصلاح وزن نورون‌هایی در این مدل به دو روش انجام می‌شود:
از مدل Madaline برای کلاس‌بندی داده‌هایی که به صورت خطی جداپذیر نیستند، استفاده می‌شود.
همانطور که در ادامه بررسی می‌کنیم، دیتاست این سوال هم این ویژگی را دارد.

Madaline I یا MRI نسخه اولیه است که از روش Rule I برای به روز رسانی وزن‌ها استفاده می‌کند. نورون‌ها تنها زمانی که خروجی شبکه با مقدار هدف (Label) مغایرت دارد، وزن‌های خود را به روز رسانی می‌کنند.
Madaline II یا MRII نسخه بهبود یافته از MRI است که در آن به روزرسانی وزن‌ها با استفاده از روش بهینه‌سازی Rule iI صورت می‌گیرد. توانایی تطبیق و همگرایی بهتری دارد و از نرخ یادگیری متفاوتی در طول فرایند یادگیری استفاده می‌کند.
این دو روش نیازی به مشتق پذیری تابع فعال سازی ندارند و به راحتی می‌توانیم از تابع پله یا آستانه برای مدل استفاده کنیم.
ما در این سوال، به طور عمیق روی MRI تمرکز می‌کنیم.
همانطور که ذکر شد در این الگوریتم از دو لایه مخفی و خروجی استفاده می‌شود. هر کدام از نورون‌های لایه مخفی، یک مرز تصمیم بر روی دیتاست پیدا می‌کنند و از آنجایی که در فرایند یادگیری مدام وزن نورون‌های این لایه عوض می‌شود، به مرور مرزهای تصمیم دقیق و دقیق‌تری پیدا خواهد شد.
همه‌ی نواحی تصمیم پیدا شده توسط لایه مخفی، به دست لایه خروجی می‌رسد و این لایه خطوط جدا کننده نهایی را انتخاب می‌کند. عموما این لایه عملکردی شبیه به or یا and برای تصمیم‌گیری روی مرزهایی گرفته‌شده از لایه قبل اعمال می‌کند. در MRI، وزن لایه خروجی ثابت است و عملکرد کلی آن تغییری نمی‌یابد. در واقع وزن لایه اول به نحوی تغییر می‌یابد که خروجی این لایه به درستی کلاس‌های داده را تفکیک کند.
الگوریتم پیاده شده به صورت زیر است:
ابتدا وزن‌های لایه مخفی شبکه را به مقادیر اولیه مقداردهی می‌کنیم.
برای پیاده‌سازی منطق or در لایه خروجی، وزن نورون‌ها را با 1 و بایاس مربوط به هر نورون را با $neuronCount - 1$ مقداردهی می‌کنیم.

۳-۲. نمودار پراکندگی داده‌ها

دیتاست ورودی، از سه ستون تشکیل شده که دو ستون اول ویژگی‌ها و ستون آخر کلاسی‌ست که داده مربوطه متعلق به آن است. نمودار پراکندگی این مجموعه داده به شکل زیر می‌باشد:



شکل 42. **توزیع داده‌های دو کلاس در دیتاست**

همانطور که مشخص است، داده‌های مربوط به کلاس 0 در مرکز تصویر فوق پخش شده اند. داده‌های مربوط به کلاس 1، به شکل یک ring در اطراف مرکز داده‌های مربوط به کلاس 0 قرار دارند. همانطور که مشخص است، این کلاس‌ها با هم همپوشانی ندارند. این داده‌ها به صورت خطی قابل تفکیک نیستند و بهتر است از مدل Madaline برای دستیابی به دقت بیشتر استفاده کرد.

۳-۳. آموزش مدل

برای شروع فرایند آموزش، وزن‌های اولیه پارامترهای قابل یادگیری در لایه نهان را به عدد بسیار ریز 0.0001 ست کرده‌ایم.

همینطور پارامترهای لایه خروجی را بدین صورت مقدار داده ایم:

وزن‌های لایه با مقدار 1 ست شده است. مقدار بایاس جمع شده، برابر یک واحد کمتر از تعداد نورون های لایه میانی است. علت این امر، پیاده سازی منطق or در لایه خروجی می‌باشد. در این حالت تنها در صورتی مقدار خروجی +1 است که حداقل یک نورون میانی مقدار +1 خروجی داده باشد. همینطور تنها زمانی کلاس -1 پیشبینی میشود که همگی نورون های میانی مقدار -1 خروجی داده باشند.

توجه شود که طبق عادت همیشگی برای پیاده‌سازی این مدل، کلاس ها را به دو کلاس +1 و -1 تبدیل می‌کنیم تا محاسبات آسانتر باشد. در واقع کلاس 0 را به کلاس -1 تبدیل کرده ایم.

تابع step_func نوعی فعال ساز می‌باشد که چک می‌کند اگر مقدار خروجی بزرگتر یا مساوی 0 بود، به آن کلاس +1 اختصاص دهد. در غیر این صورت، اگر مقدار آن منفی بود، کلاس -1 اختصاص داده می‌شود.

برای اندازه‌گیری دقت در مسائل regression، می‌تواند از MSE یا RMSE استفاده کرد که RMSE جذر MSE می‌باشد. ما در این سوال از روش زیر استفاده کرده‌ایم.

برای محاسبه خطا در این مدل، از فرمول خطای MSE به صورت زیر استفاده می‌شود.

$$loss = \frac{(y - label)^2}{2}$$

روش MSE به علت به توان دو رساندن اختلاف، اختلاف خطاهای بزرگ را در نهایت در خطای نهایی به میزان بزرگی‌شان دخالت می‌دهد و عملیات جذر گیری ندارد.

منطق پیاده شده در لایه های مدل، به صورت زیر می‌باشد.

$$y = f(\sum_{i=1} (w_i x_i) + b)$$

که در آن وزن‌ها به ترتیب در مقادیر ورودی ضرب شده است. در نهایت، مقدار بایاس به مجموع اضافه شده است. تابع f همان تابع فعالساز ذکر شده می‌باشد.

فرایند آپدیت وزن‌ها نیز به صورت زیر صورت می‌گیرد:

```
if label == 1 and y != label:
    to_update = raw_out.argmax()

self.hidden_weights[to_update,:]+=self.lr*(label-raw_out[to_update])*
x
            self.hidden_bias[to_update]   +=   self.lr   *   (label   -
raw_out[to_update])
elif label == -1 and y != label:
    to_update = (raw_out >= 0)
    n = to_update.sum()
    x1 = np.reshape(-1 - raw_out[to_update], (n, 1))
    x_ = np.tile(x, (n, 1))
    self.hidden_weights[to_update, :] += self.lr * x1 * x_
             self.hidden_bias[to_update]   +=   self.lr   *   (-1   -
raw_out[to_update])
```

همانطور که در کد مشخص است، این مدل، آپدیت وزن‌ها را تحت شرایطی انجام می‌دهد که، لیبل پیشبینی شده کلاس، با لیبل واقعی آن مغایرت داشته باشد. در این صورت، به ازای داده با کلاس +1 که به اشتباه به -1 پیشبینی شده، وزن‌های نورونی که بیشترین خروجی را برای داده‌ی مربوطه پیشبینی کرده، آپدیت می‌شود.

به ازاده داده‌ای که کلاس -1 داده و به اشتباه لیبل گذاری شده، وزن همه‌ی نورون‌هایی که خروجی نامنفی داده‌اند، آپدیت خواهد شد.

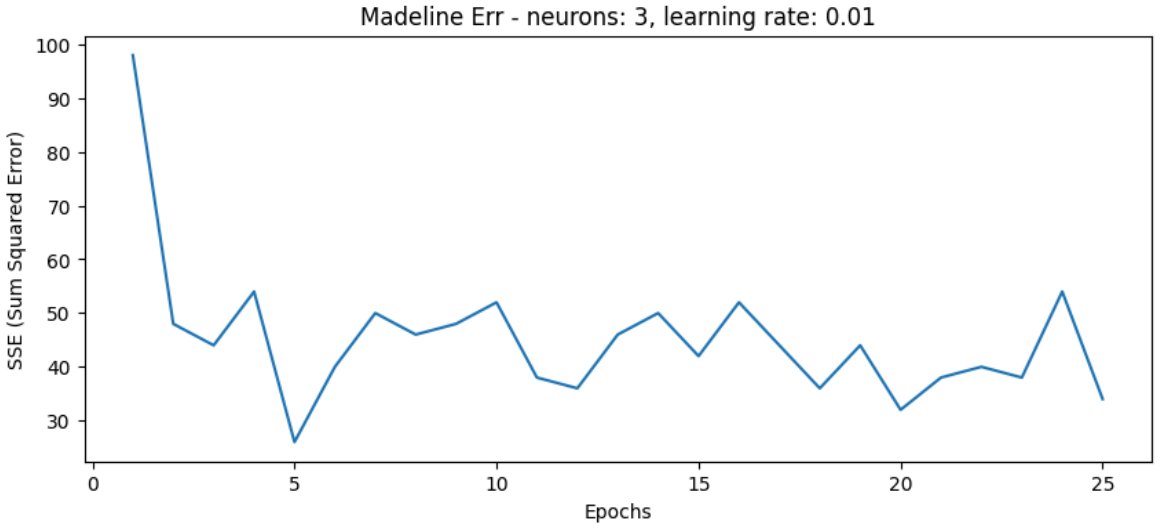
۳-۴. تحلیل نتایج

هر سه مدل، به تعداد ایپاک 20، مقدار learning rate برابر 0.01 آموزش داده شده است.

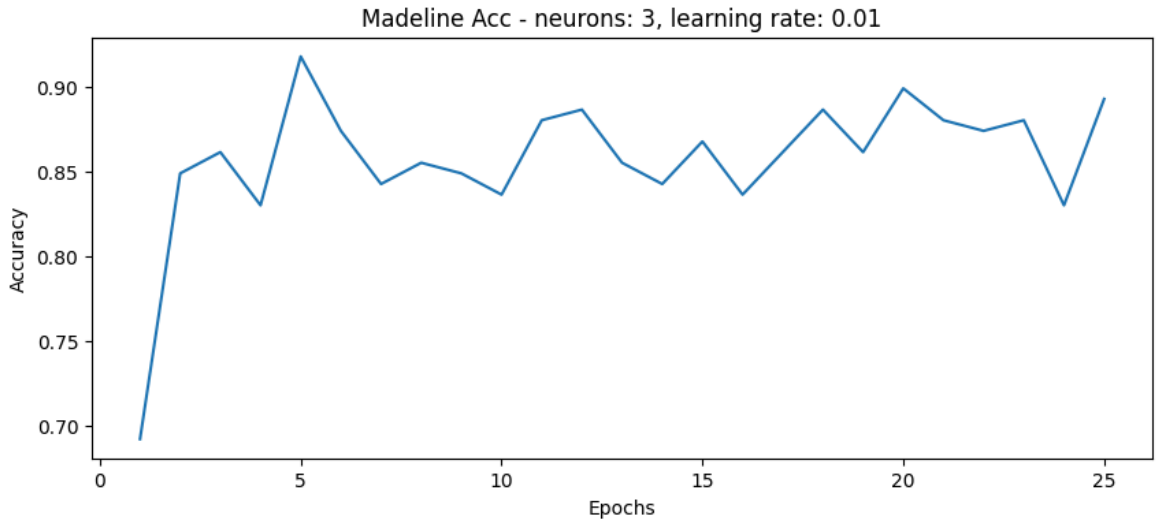
همانطور که مشخص است، مدل با 8 نورون در لایه مخفی در ایپاک نهم به دقت 100 و لاس 0 رسیده است. مدل با 4 نورون در لایه مخفی، در تعداد ایپاک 5 به دقت 100 و لاس صفر رسیده است. با این حال، مدل پیچیده‌تر یعنی مدل 8 نورون دار، روی داده‌ی تست، دقت بهتری نسبت به مدل 5 نورون دار نشان داده‌است. در واقع مدل پیچیده‌تر، دیر تر عملیات یادگیری را تمام کرده و علت آن، بالاتر بودن تعداد پارامترهای مدل می‌باشد ولی در عوض، عملکرد بهتری نسبت به داده‌ی از قبل دیده نشده نشان می‌دهد.

مدل با 3 نورون در لایه میانی نیز در تعداد ایپاک‌ها مشخص شده به دقت حدود 90 رسیده است و علت آن، ماهیت توزیع داده و شکل آن می‌باشد که با 3 خط جداکننده، به سادگی قابل تفکیک صددرصد نمی‌باشد. همانطور که قابل پیش‌بینی‌ست، در این مدل دقت در داده تست کمی کمتر از دقت در داده‌ی ترین می‌باشد اما با این حال دقت قابل قبولی گرفته‌ایم.

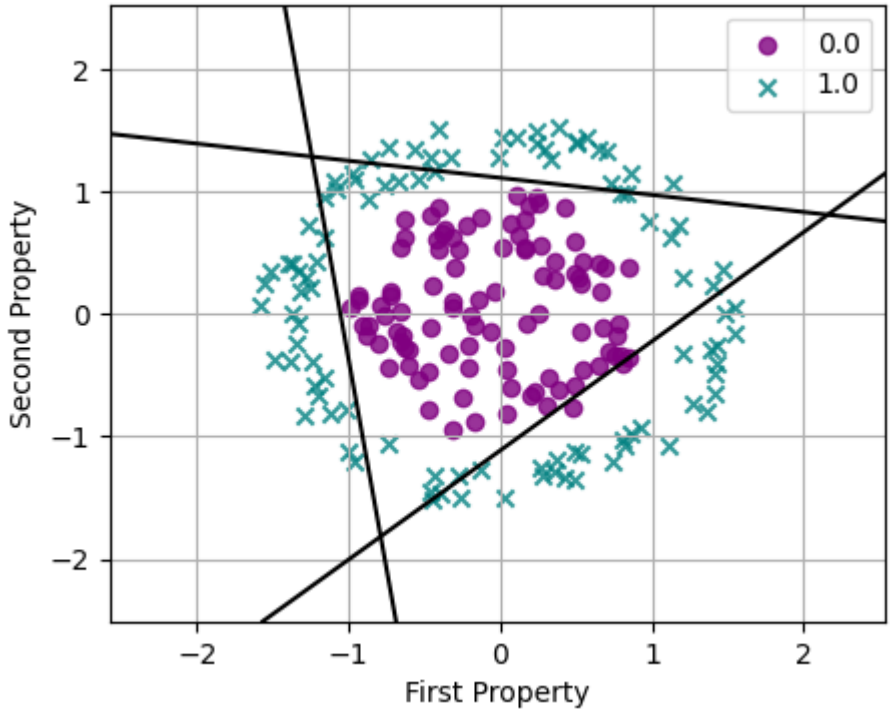
در شکل ؟ تفکیک داده‌ها با 3 خط جداکننده نمایش داده شده است. از آنجایی که مدل ما 3 نورون برای تعیین خطوط جداکننده دارد، نود لایه خروجی این سه خط را با هم or کرده و نتیجه، مرزبندی به 3 خط خواهد بود.



شکل 43. **نمودار خطای مربوط به مدل Madaline به 3 نورون در لایه مخفی**



شکل 44. **نمودار دقت مربوط به مدل Madaline به 3 نورون در لایه مخفی**



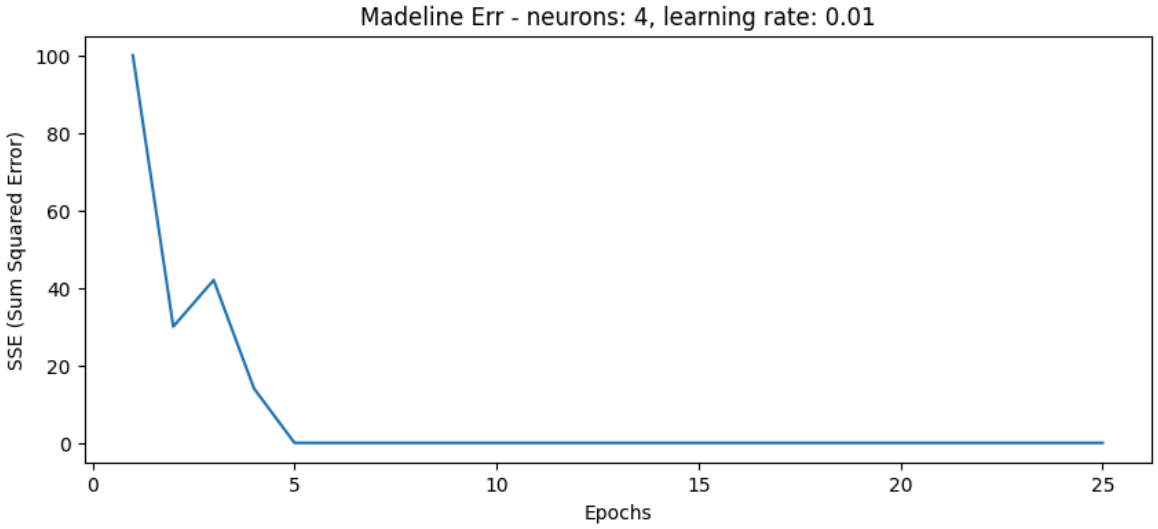
شکل 45. **خطور جداکننده مربوط به مدل Madaline به 3 نورون در لایه مخفی**

جدول 1. **نتایج مدل Madaline با 3 نورون در لایه میانی روی داده ترین و تست**

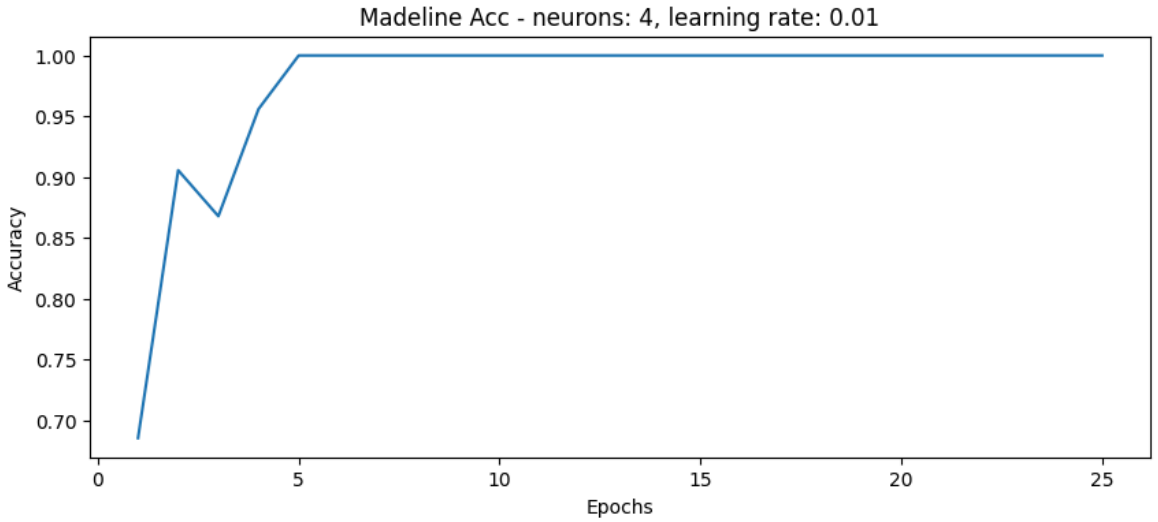
	دقت
Train	89.308
Test	85

جواب

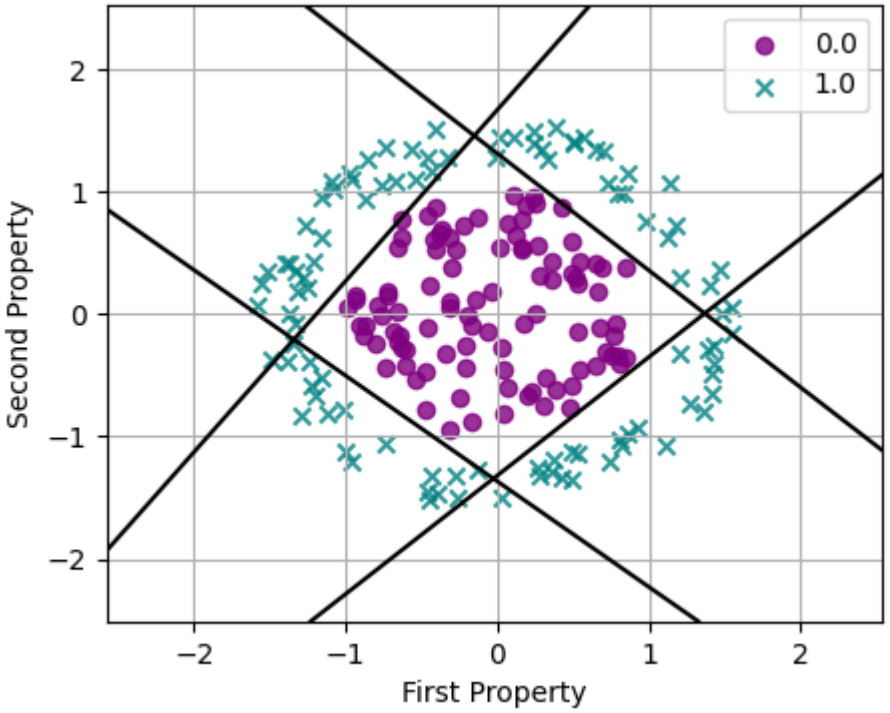
در شکل ؟ تفکیک داده‌ها با 4 خط جداکننده نمایش داده شده است. از آنجایی که مدل ما 4 نورون برای تعیین خطوط جداکننده دارد، نود لایه خروجی این سه خط را با هم or کرده و نتیجه، مرزبندی به 4 خط خواهد بود.



شکل 46. **نمودار خطای مربوط به مدل Madaline به 4 نورون در لایه مخفی**



شکل 47. **نمودار دقت مربوط به مدل Madaline به 4 نورون در لایه مخفی**

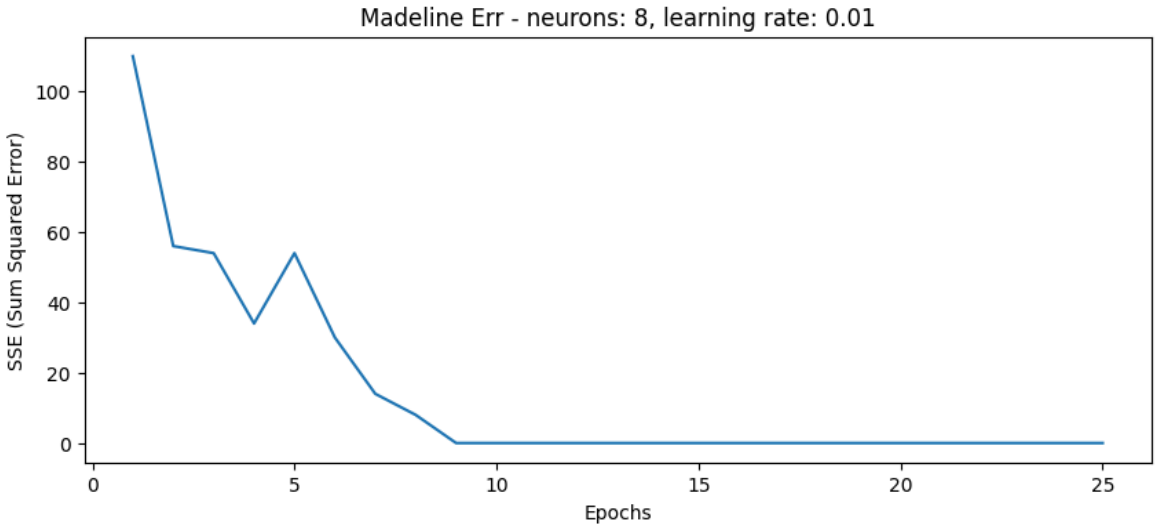


شکل 48. **خطور جداکننده مربوط به مدل Madaline به 4 نورون در لایه مخفی**

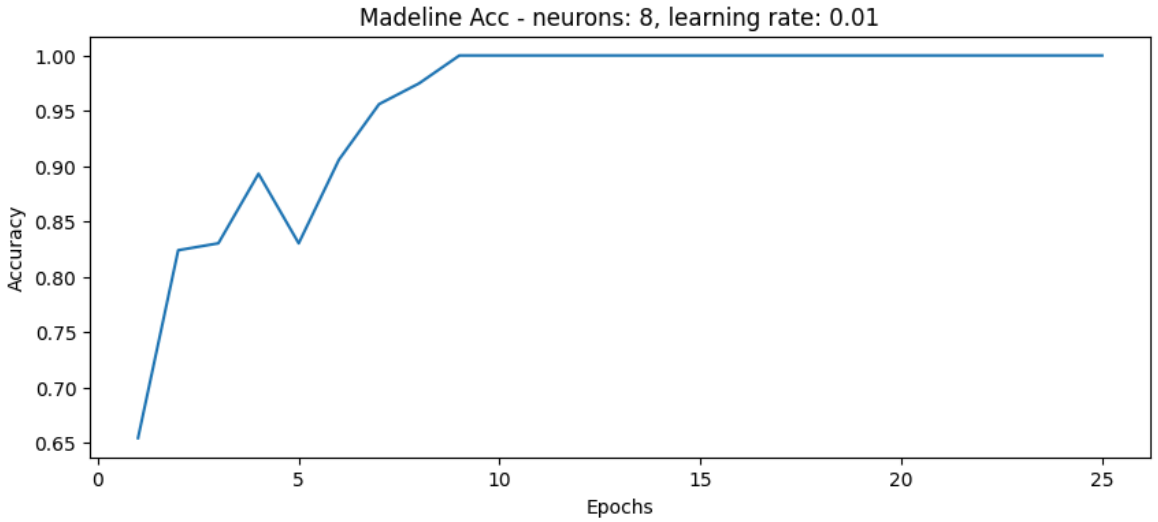
جدول 4. **نتایج مدل Madaline با 4 نورون در لایه مخفی روی داده ترین و تست**

	دقت
شبکه‌ی اول	100
شبکه‌ی دوم	97.5

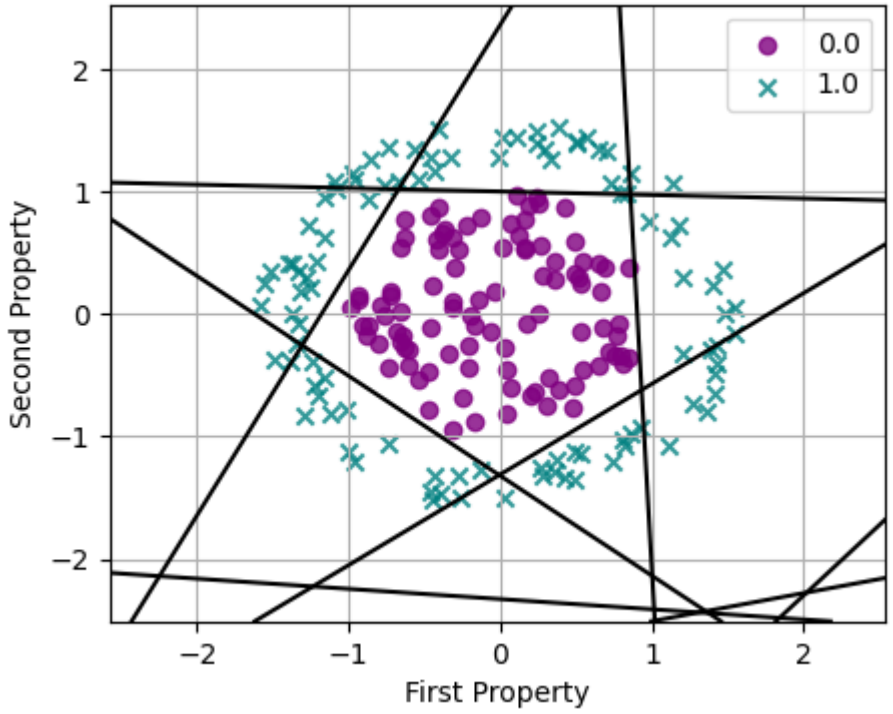
در شکل ؟ تفکیک داده‌ها با 8 خط جداکننده نمایش داده شده است. از آنجایی که مدل ما 8 نورون برای تعیین خطوط جداکننده دارد، نود لایه خروجی این سه خط را با هم or کرده و نتیجه، مرزبندی به 8 خط خواهد بود.



شکل 49. **نمودار خطای مربوط به مدل Madaline به 8 نورون در لایه مخفی**



شکل 50. **نمودار دقت مربوط به مدل Madaline به 8 نورون در لایه مخفی**



شکل 51. **خطور جداکننده مربوط به مدل Madaline به 8 نورون در لایه مخفی**

جدول 1. **نتایج مدل Madaline با 8 نورون در لایه مخفی روی داده ترین و تست**

	دقت
شبکه‌ی اول	100
شبکه‌ی دوم	100

همانطور که مشاهده شد، تعداد نورون بیشتر، مدل قوی‌تری برای کلاس بندی به ما می‌دهد. چرا که تعداد بی نهایت خط جداکننده، در نهایت به شکل تقریبا دایره در می‌آید و این با داده‌ی ما تطابق دارد.

توجه کنید که با کوچک‌تر کردن مقادیر اولیه ست شده به وزن‌ها و بایاس‌ها دقت روی هر سه مدل افزایش می‌یابد، اما برای مقایسه بهتر به مقدار 0.0001 بسنده کرده‌ایم.

همانطور که مشاهده شد، هرچقدر مدل پیچیده‌تر باشد، تعداد ایپاک بیشتری برای یادگیری وزن‌ها نیاز دارد. این پیچیدگی به همراه وزن‌های بهینه، عملکرد بهتری روی داده‌ی از قبل دیده نشده نشان می‌دهد که به علت ماهیت قوی‌تر بودن مدل مربوطه می‌باشد.

پرسش ۴ – MLP

۴-۱. نمایش تعداد ستون

در این بخش باید دیتا داده شده را خوانده و تعداد nan های هر ستون را نشان میدادیم :

```
df = pd.read_csv("../data/Question4.csv")
```

Python

Display the Number of Columns

```
nan_counts = df.isna().sum()
nan_counts_df = pd.DataFrame(nan_counts, columns=['NaN Counts'])
print(tabulate(nan_counts_df, headers='keys', tablefmt='pretty'))
```

Python

```
+---------------+------------+
|               | NaN Counts |
+---------------+------------+
|      id       |     0      |
|     date      |     0      |
|     price     |     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      |
+---------------+------------+
```

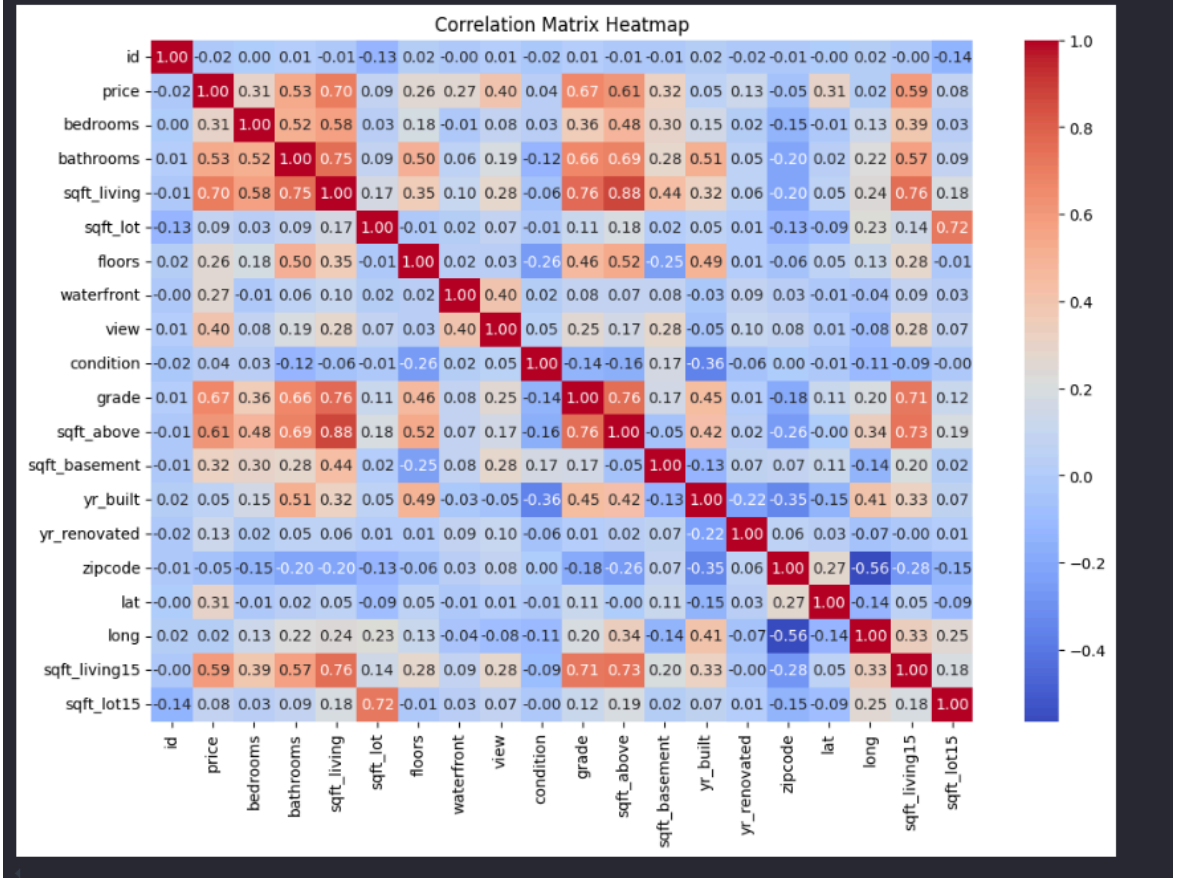
شکل 52. کد پیاده سازی شده جهت پرینت تعداد nan ها

همانطور که مشاهده می‌شود هیچ یک از ستون های داده فیلد nan ندارند .

۲-۴. ماتریس همبستگی

در این قسمت باید ماتریس همبستگی فیلد های داده را رسم کنیم :

```
plot_correlation_matrix(df)
```

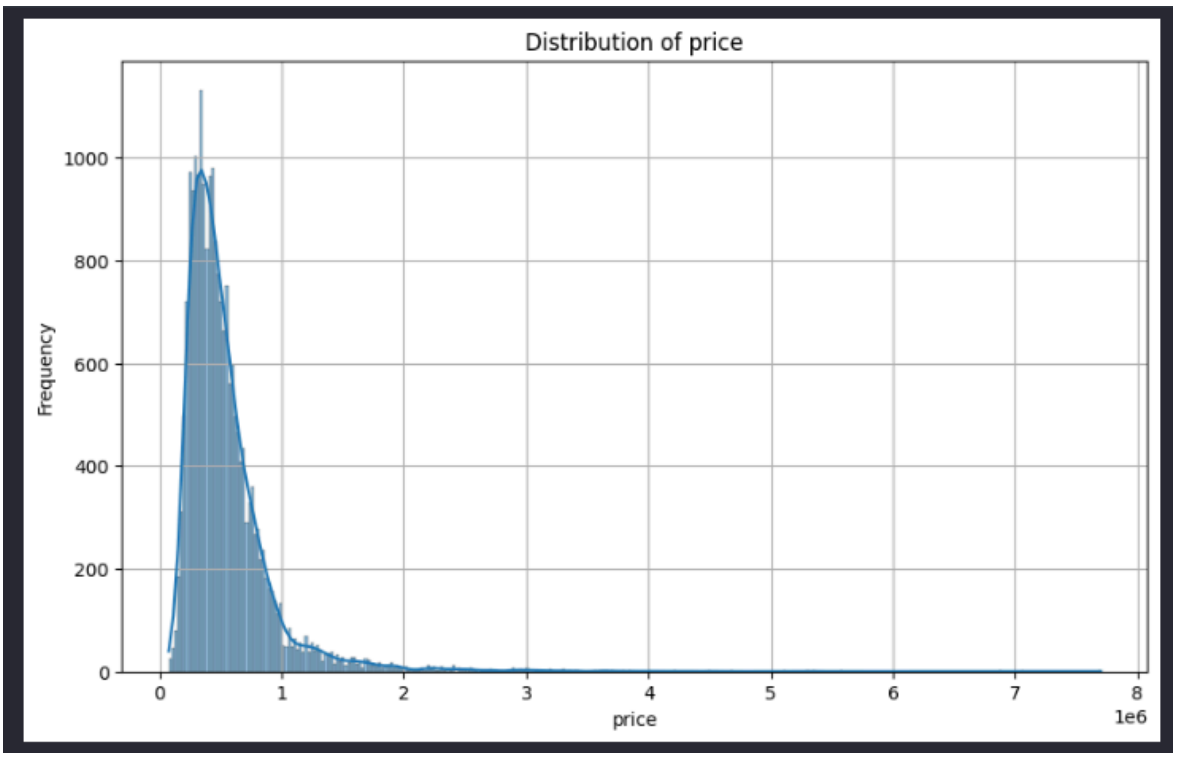


شکل 53. نمودار همبستگی دیتاست

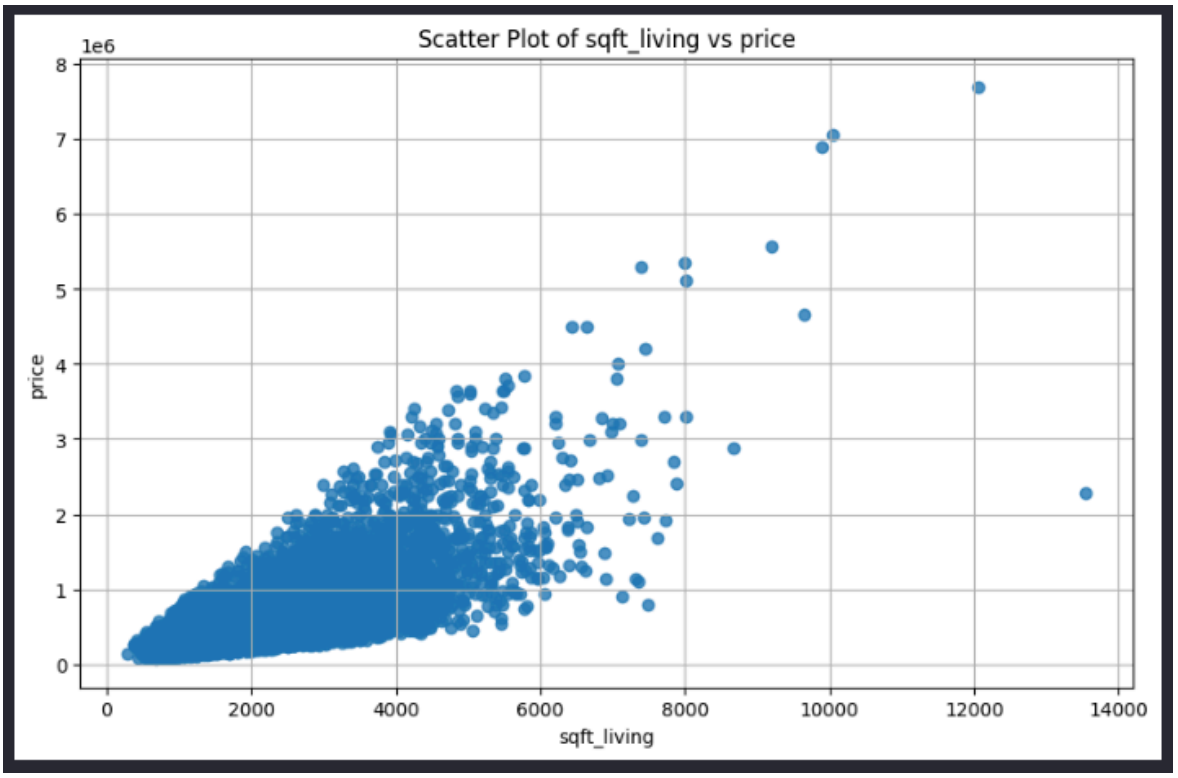
همانطور که مشاهده می شود فیلد sqft_living و grade و sqft_above بالاترین ضریب همبستگی را با فیلد هدف دارند.

۳-۴. رسم نمودار

در این قسمت ابتدا نمودار توزیع قیمت و سپس نمودار قیمت و فیلد sqft_living با قیمت را رسم کنیم :



شکل 54. شکل نمودار توزیع قیمت



شکل 55. نمودار sqft_living به price

۴-۴. پیش پردازش داده

ابتا باید ستون date را به دو ستون ماه و سال تبدیل می کردیم که در کد زیر انجام شده است :

```
def extract_year_month(df, date_field):
    df[date_field] = pd.to_datetime(df[date_field], format='%Y%m%dT%H%M%S')
    df['year'] = df[date_field].dt.year
    df['month'] = df[date_field].dt.month
    df = df.drop(columns=[date_field], axis=1, inplace=True)
    return
```

سپس باید داده ها را به دو بخش ترین و ولیدیشن تقسیم میکردیم :

```
X = df.drop('price', axis=1)
Y = df['price']
X_train, X_validation, y_train, y_validation = train_test_split(X, Y, test_size=0.25, random_state=13)
```

و پس از آن به کمک minmaxScaler داده ها را نرمالایز میکردیم :

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_validation_scaled = scaler.transform(X_validation)
y_train_df = pd.DataFrame(y_train, columns=['price'])
y_validation_df = pd.DataFrame(y_validation, columns=['price'])
```

۴-۴.پیاده سازی مدل

در این بخش باید کد دو مدل یکی با یک لایه پنهانی و دیگری با دو لایه پنهان را پیاده سازی می کردیم که به کمک کتابخانه پایتورچ و به صورت زیر انجام شد :

```python
class MLP_OneHidden(nn.Module):
    Tabnine | Edit | Test | Explain | Document | Ask
    def __init__(self):
        super(MLP_OneHidden, self).__init__()
        self.hidden = nn.Linear(X_train.shape[1], 128)
        self.dropout = nn.Dropout(p=0.15)
        self.output = nn.Linear(128, 1)
    Tabnine | Edit | Test | Explain | Document | Ask
    def forward(self, x):
        x = torch.relu(self.hidden(x))
        x = self.dropout(x)
        x = self.output(x)
        return x

class MLP_TwoHidden(nn.Module):
    Tabnine | Edit | Test | Explain | Document | Ask
    def __init__(self):
        super(MLP_TwoHidden, self).__init__()
        self.hidden1 = nn.Linear(X_train.shape[1], 64)
        self.dropout1 = nn.Dropout(p=0.1)
        self.hidden2 = nn.Linear(64, 128)
        self.dropout2 = nn.Dropout(p=0.1)
        self.output = nn.Linear(128, 1)
    Tabnine | Edit | Test | Explain | Document | Ask
    def forward(self, x):
        x = torch.relu(self.hidden1(x))
        x = self.dropout1(x)
        x = torch.relu(self.hidden2(x))
        x = self.dropout2(x)
        x = self.output(x)
        return x
```

شکل 56. پیاده سازی مدل های mlp

همانطور که مشاهده میشود در مدل های پیاده شده دراپ اوت نیز جهت جلوگیری از اورفیت قرار دارد .

۴-۴.آموزش مدل

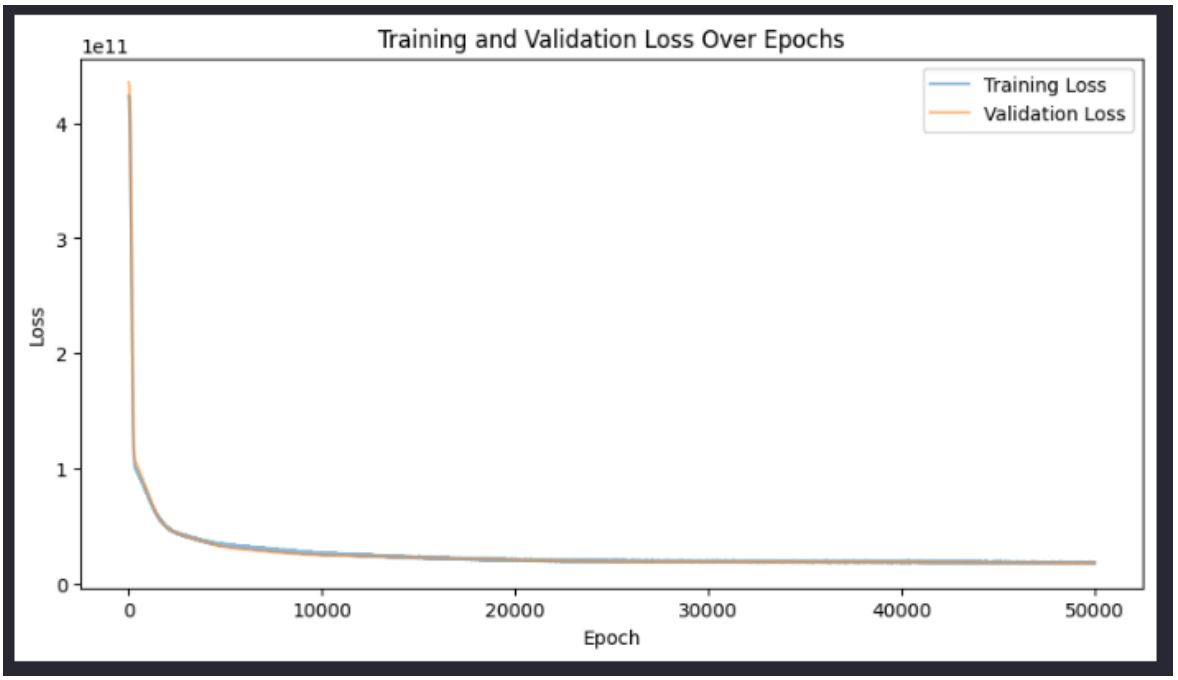
ابتدا باید اپتیمایزر و لاس فانکشن را انتخاب میکردیم از آنجا که یک تسک رگرشن داریم لاس فانکشن mse را انتخاب کردیم . و برای اپتیمایزر از adam به همراه weight_decay به عنوان regularization استفاده کردیم .

```python
def train_model(model, X_train, y_train, X_validation, y_validation, epochs=50, learning_rate=0.01, weight_decay=0.01):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=0.001)
    train_losses = []
    val_losses = []
    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
        loss.backward()
        optimizer.step()
        train_losses.append(loss.item())
        model.eval()
        val_outputs = model(X_validation)
        val_loss = criterion(val_outputs, y_validation)
        val_losses.append(val_loss.item())
        # if (epoch+1) % 100 == 0:
        #     if len(val_losses) > 510 and np.mean(val_losses[-1:-250]) > np.mean(val_losses[-250:-500]):
        #         for param_group in optimizer.param_groups:
        #             param_group['lr'] *= 0.5
        #         learning_rate *= 0.5
        #         print (f'reduced learning rate from {2*learning_rate} to {learning_rate}')
        if (epoch+1) % 500 == 0:
            print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.8f}, Validation Loss: {val_loss.item():.8f}')
    return train_losses, val_losses
```

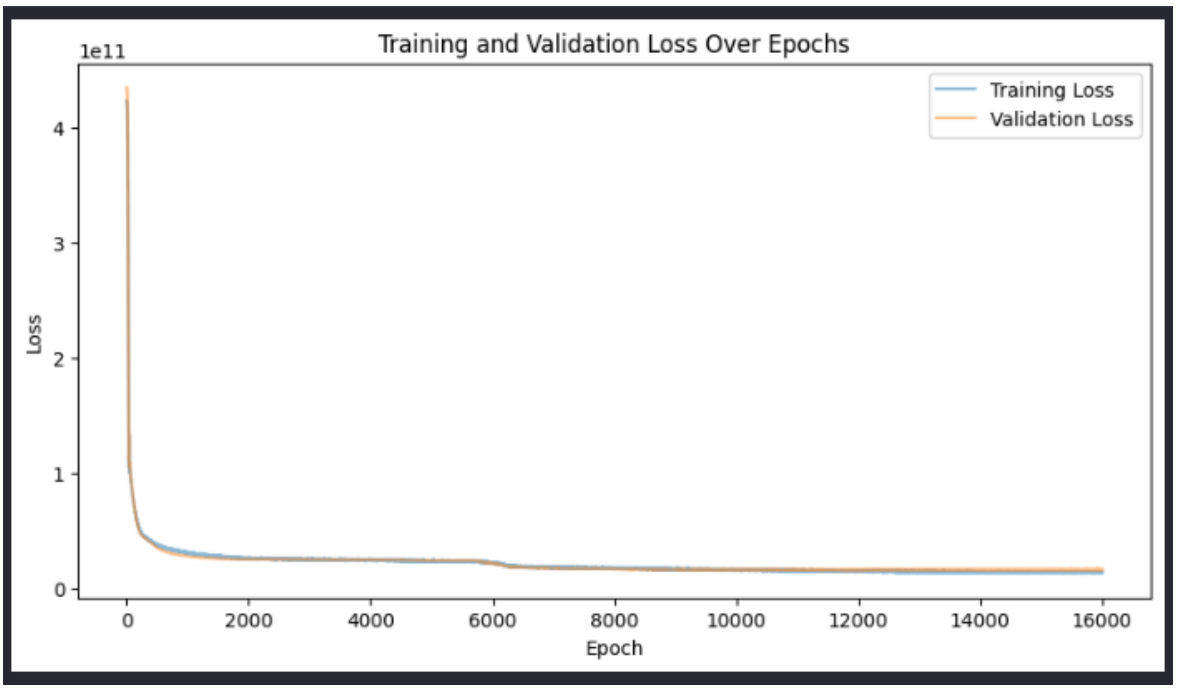
شکل 57. تابع ترین مدل ها

مدل ها را به کمک تابع بالاترین میکنیم.

نتایج به دست آمده :



شکل 58. نمودار ترین و ولیدیشن لاس در حین ترین مدل با یک لایه مخفی



شکل 59. نمودار ترین و ولیدیشن لاس در حین ترین مدل با دو لایه مخفی

۴-۴. تحلیل نتایج

مدل 2 لایه مخفی در آخرین ایپاک ها بر روی ولیدیشن ست به خطایی حدود 15654548480 رسید در حالی که مدل با 1 لایه مخفی به خطایی حدود 18005493760 رسید که علی رغم تعداد ایپاک بسیار بیشتر در مدل با یک لایه مخفی همچنان نتیجه ضعیف تری دریافت کردیم که موضوع طبیعیی نیز هست زیرا علی رغم اینکه میدانیم با مدل با یک لایه مخفی هر چیزی را می توان مدل کرد ولی به نود های بسیار بیشتر نیاز است و همچنین تعداد ایپاک بالا برای بهینه سازی و به صورت کلی به دلیل توانایی مدل با دو لایه مخفی برای تشخیص پترن های پیچیده تر موجود در دیتا میتواند نتیجه سریع تر و بهتری به ما بدهد.

برای مدل با دو لایه پنهان علی رغم حضور دراپ اوت همچنان بعد از ایپاک 16000 مدل شروع به اورفیت میکند پس نهایت ایپاک قابل انجام ما 16000 خواهد بود و همچنین از تعداد ایپاک های بسیار پایین نیز به دلیل استفاده از دراپ اوت و regularization نمیتوان استفاده کرد زیرا مدل فرصت بهینه شدن را پیدا نمی‌کند.در نتیجه میتوان بهترین ایپاک را بر اساس نمودار حدود 8000 در نظر گرفت گرچه ادامه ترین تا 16000 همچنان موجب بهبود ارام عملکرد مدل می‌شود.

برای مدل 1 لایه مخفی نیز در جایی حدود 20000 ایپاک میتوان فرایند آموزش را پایان داد در حالی که ادامه فرایند ترین تا 50000 ایپاک نیز صرفا موجب بهبود بسیار کمی در مدل خواهد شد و قابل انجام است .

سپس بر روی مدل 2 لایه پنهان 5 سمپل را پیش بینی کردیم که نتیجه به شکل زیر بود .

```
for sample 1: the prediction was 528936.5625 and the real value was 500000.0
for sample 2: the prediction was 461896.875 and the real value was 489900.0
for sample 3: the prediction was 545747.75 and the real value was 569000.0
for sample 4: the prediction was 637742.9375 and the real value was 650000.0
for sample 5: the prediction was 599771.125 and the real value was 647000.0
```