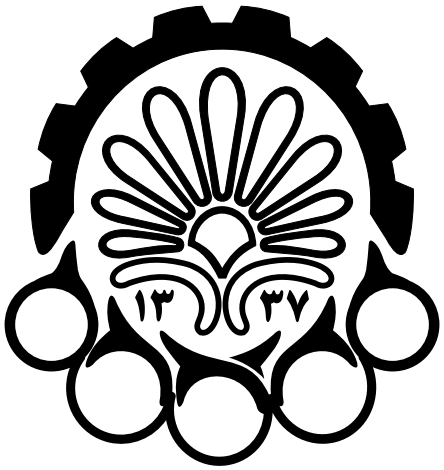


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمر ین دوم درس شبکه های عصبی

دکتر صفابخش

غلامرضا دار ۴۰۰۱۳۱۰۱۸

زمستان ۱۴۰۰

فهرست مطالب

سوال ۱) ۳
۱) ۴
۲) ۱۱
۳) ۱۳
۴) ۱۶
۵) ۱۸
۶) ۲۰

سوال ۱)

قبل از شروع، لازم است دیتاست را فراخوانی کنیم و کمی با آن آشنا شویم. همانطورکه در طرح سوال گفته شده، این دیتاست شامل ۶۹۰ داده با ۱۵ ویژگی مختلف است.

```
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      678 non-null    object
 1   A2      678 non-null    float64
 2   A3      690 non-null    float64
 3   A4      684 non-null    object
 4   A5      684 non-null    object
 5   A6      681 non-null    object
 6   A7      681 non-null    object
 7   A8      690 non-null    float64
 8   A9      690 non-null    object
 9   A10     690 non-null    object
 10  A11     690 non-null    int64
 11  A12     690 non-null    object
 12  A13     690 non-null    object
 13  A14     677 non-null    float64
 14  A15     690 non-null    int64
 15  A16     690 non-null    object
dtypes: float64(4), int64(2), object(10)
memory usage: 86.4+ KB
```

تعداد ۶ ویژگی از نوع عددی هستند و ۹ ویژگی دیگر از نوع Categorical هستند. همچنین ستون مربوط به Label داده ها نیز به صورت Categorical با دو مقدار '-' , '+' است. همچنین همانطورکه در تصویر زیر میبینید، مقادیر ستون های عددی در بازه های مختلف و با خواص آماری مختلفی هستند.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
count	678	678.000000	690.000000	684	684	681	681	690.000000	690	690	690.00000	690	690	677.000000	690.000000	690
unique	2	NaN	NaN	3	3	14	9	NaN	2	2	NaN	2	3	NaN	NaN	2
top	b	NaN	NaN	u	g	c	v	NaN	t	f	NaN	f	g	NaN	NaN	-
freq	468	NaN	NaN	519	519	137	399	NaN	361	395	NaN	374	625	NaN	NaN	383
mean	NaN	31.568171	4.758725	NaN	NaN	NaN	NaN	2.223406	NaN	NaN	2.40000	NaN	NaN	184.014771	1017.385507	NaN
std	NaN	11.957862	4.978163	NaN	NaN	NaN	NaN	3.346513	NaN	NaN	4.86294	NaN	NaN	173.806768	5210.102598	NaN
min	NaN	13.750000	0.000000	NaN	NaN	NaN	NaN	0.000000	NaN	NaN	0.00000	NaN	NaN	0.000000	0.000000	NaN
25%	NaN	22.602500	1.000000	NaN	NaN	NaN	NaN	0.165000	NaN	NaN	0.00000	NaN	NaN	75.000000	0.000000	NaN
50%	NaN	28.460000	2.750000	NaN	NaN	NaN	NaN	1.000000	NaN	NaN	0.00000	NaN	NaN	160.000000	5.000000	NaN
75%	NaN	38.230000	7.207500	NaN	NaN	NaN	NaN	2.625000	NaN	NaN	3.00000	NaN	NaN	276.000000	395.500000	NaN
max	NaN	80.250000	28.000000	NaN	NaN	NaN	NaN	28.500000	NaN	NaN	67.00000	NaN	NaN	2000.000000	100000.000000	NaN

(۱

در این بخش با کمک اطلاعاتی که نسبت به ویژگی های مختلف کسب کردیم میخواهیم به پیش پردازش داده ها بپردازیم. برای اینکار ۳ مرحله "**تصحیح مقادیر گم شده**"، "**نرمال سازی**" و "**کدگذاری (Encoding)**" را بر روی تک تک ویژگی ها اعمال میکنیم.

بنابراین در مرحله اول میخواهیم بدانیم چه ویژگی هایی مقادیر گم شده دارند.

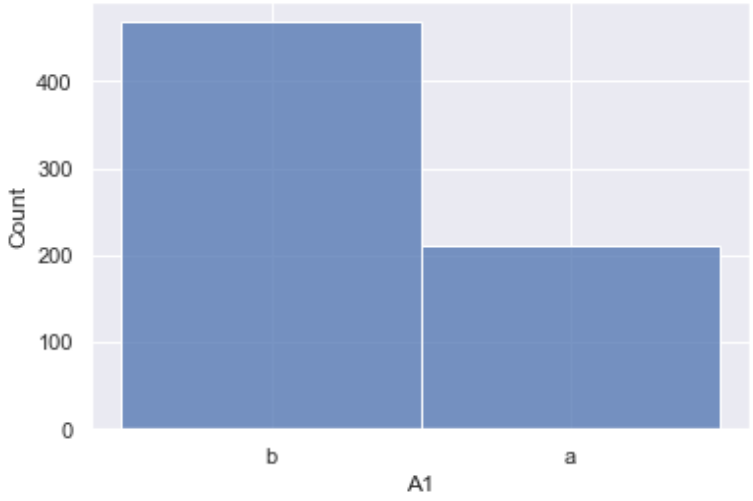
```
# Count of missing values per column
df.isna().sum()
```

[10] ✓ 0.1s

```
A1     12
A2     12
A3      0
A4      6
A5      6
A6      9
A7      9
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14    13
A15     0
A16     0
dtype: int64
```

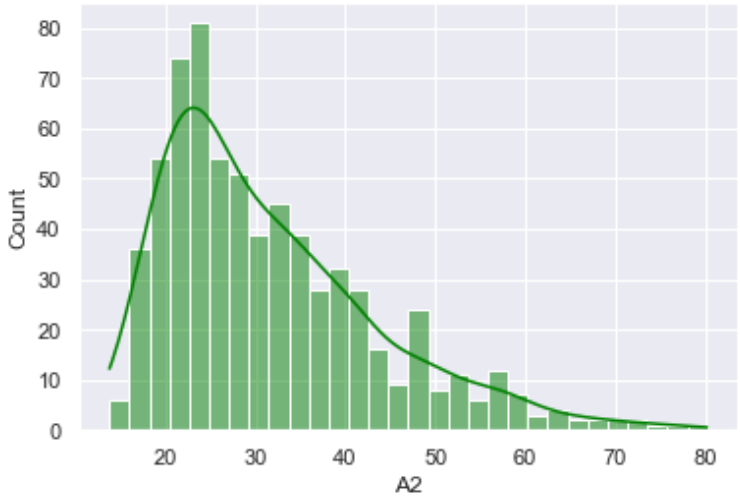
همانطورکه در تصویر بالا مشاهده میکنید، ویژگی های A1, A2, A4, A5, A6, A7, A14 دارای مقادیر گم شده هستند که در این بین ویژگی های A2 و A14 عددی هستند.

با ویژگی **A1** شروع میکنیم :



مشاهده میشود که ویژگی A1 دارای دو مقدار ’a’, ’b‘ است. تعداد ۴۶۸ داده مقدار b و تعداد ۲۱۰ داده مقدار a را برای این ویژگی دارند. از آنجایی که مقدار b فراوانی تقریبا دوبرابری نسبت به a دارد احتمالا اگر در آن ۱۲ داده ای که مقداری برای ستون A1 وجود ندارد b را قرار دهیم مشکلی پیش نیاید.

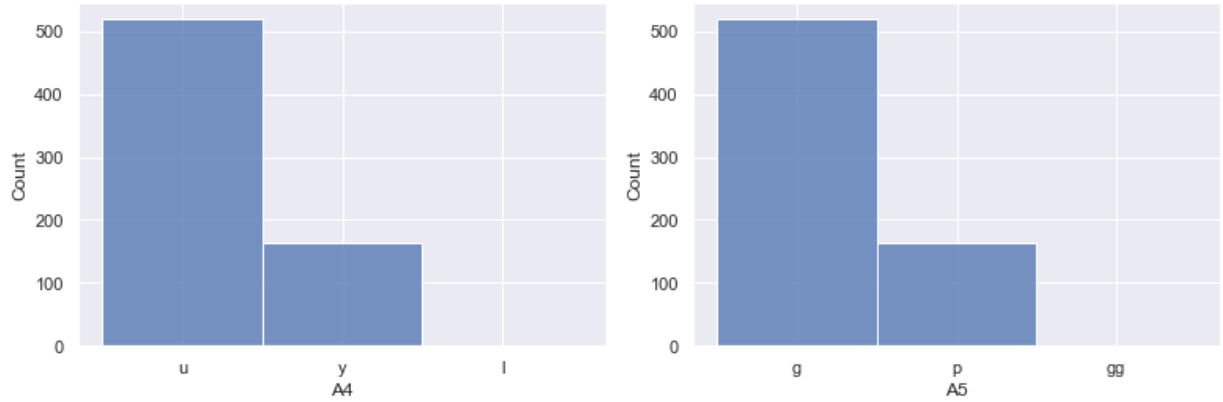
ویژگی **A2** :



ویژگی A2 یک ویژگی عددی است. بنابراین بهتر است برخلاف ویژگی A1 که از Mode برای جایگزینی مقادیر گم شده استفاده کردیم، اینجا از Mean استفاده کنیم. مقدار Mean برای این ویژگی عدد 31.56 است.

ویژگی های **A4, A5** :

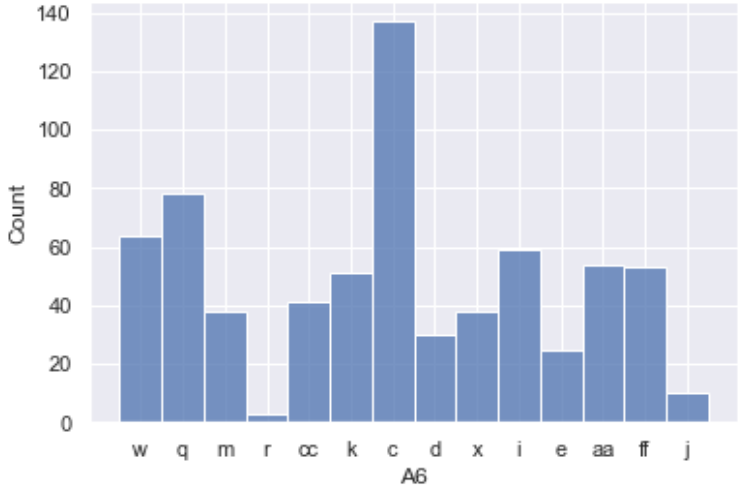
این دو ویژگی مانند ویژگی A1 کتگوریکال هستند و با توجه به توزیعی که مقادیر مختلف آنها دارد میتوانیم همانند A1 عمل کنیم و بجای مقادیر گم‌شده از Mode داده ها استفاده کنیم.

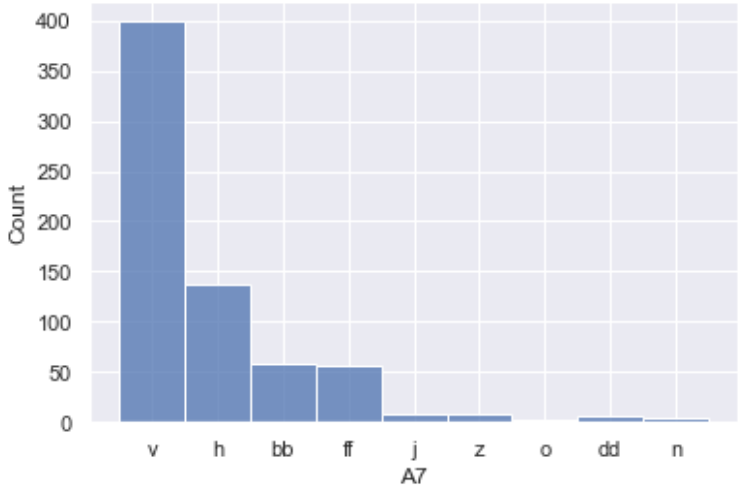


ویژگی های **A6, A7** :

تفاوتی که این دو ویژگی با ویژگی های گسسته قبلی دارند تعداد مقادیر یکتای آنهاست. همانطور که مشخص است این دو ویژگی تعداد بسیار بیشتری مقدار مختلف به عنوان ارزش خود قبول میکنند که این باعث میشود درستی انتخاب Mode برای تمامی خانه های خالی با شک همراه باشد. متاسفانه به دلیل حفظ امنیت کاربران، ستون ها و مقادیر این دیتاست با اسامی تصادفی و بی معنی جایگزین شدند. این باعث میشود نتوانیم برای این مرحله و مراحل بعد از معنای ویژگی ها استفاده کنیم و از تکنیک های Feature Engineering استفاده کنیم. به عنوان مثال ویژگی A7 **میتواند** میزان حقوق دریافتی ماهانه اشخاص(در بسته های گسسته) باشد با توجه به اینکه حقوق های بالاتر را افراد کمتری دارند که در این صورت استفاده از Mode برای این ویژگی گزینه مناسبی خواهد بود. اما همانطور که ذکر شد برای این دیتاست نمیتوانیم از معانی ویژگی‌ها استفاده کنیم.

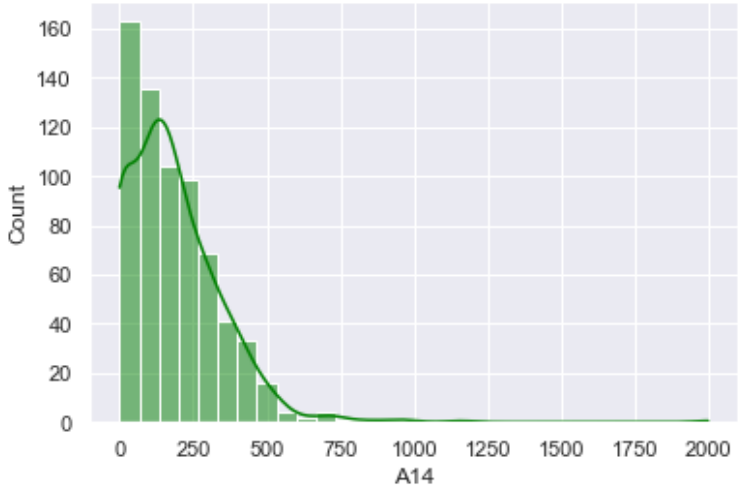
اما با توجه به اینکه تعداد داده های گم شده برای این دو ویژگی خیلی کم است (۶ داده از ۶۹۰ داده) احتمالا جایگزینی با Mode برای این دو ویژگی حتی اگر اشتباه هم باشد تاثیر زیادی نگذارد.





ویژگی **A14** :

در نهایت به ویژگی A14 میرسیم که همانند ویژگی A2 یک ویژگی عددی است. جایگزینی مقادیر گم شده این ویژگی با میانگین نیز انتخاب خوبی است.



همانطورکه در تصویر زیر میبینید پس از انجام این مرحله همه ویژگی ها تعداد ۶۹۰ داده دارند یعنی داده های گم شده به کلی از بین رفتند.

```
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      690 non-null    object
 1   A2      690 non-null    float64
 2   A3      690 non-null    float64
 3   A4      690 non-null    object
 4   A5      690 non-null    object
 5   A6      690 non-null    object
 6   A7      690 non-null    object
 7   A8      690 non-null    float64
 8   A9      690 non-null    object
 9   A10     690 non-null    object
 10  A11     690 non-null    int64
 11  A12     690 non-null    object
 12  A13     690 non-null    object
 13  A14     690 non-null    float64
 14  A15     690 non-null    int64
 15  A16     690 non-null    object
dtypes: float64(4), int64(2), object(10)
memory usage: 86.4+ KB
```

مرحله بعدی پیش‌پردازش میتواند **حذف ویژگی های اضافی** یا حتی **ترکیب ویژگی های مختلف** با هم باشد(مثلا روز و سال و ماه را به یک ویژگی عددی تبدیل کنیم). اما همانطورکه قبلا هم گفتیم معنی اصلی این ویژگی ها به دلیل مسائل امنیتی پوشانده شده اند بنابراین برای این دیتاست این مرحله را انجام نمیدهیم.

نرمال سازی مقادیر عددی:

همانطورکه میدانید، خیلی از الگوریتم های یادگیری ماشین با اعداد کوچک و حول صفر رفتار بهتری میکنند و این امر باعث همگرایی سریعتر بسیاری از الگوریتم ها میشود. بنابراین در این مرحله تمام ویژگی های عددی دیتاست را نرمال سازی میکنیم.

برای نرمال سازی ویژگی ها کافیست میانگین مقادیر یک ویژگی را از همه مقادیر آن ویژگی کم کنیم، سپس مقادیر حاصل را بر انحراف از معیار مقادیر تقسیم کنیم.

```python
for col in df.dtypes[df.dtypes != 'object'].index:
    # Normalization
    df[col] -= df[col].mean()
    df[col] /= df[col].std()
```

✓ 0.1s

خواص ویژگی های عددی پس از نرمال سازی را در تصویر زیر مشاهده میکنید.

	A2	A3	A8	A11	A14	A15
count	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02
mean	-5.663746e-17	1.029772e-17	1.029772e-16	1.029772e-17	4.633974e-17	1.029772e-17
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.503228e+00	-9.559198e-01	-6.643947e-01	-4.935286e-01	-1.068864e+00	-1.952717e-01
25%	-7.506932e-01	-7.550425e-01	-6.150897e-01	-4.935286e-01	-6.041776e-01	-1.952717e-01
50%	-2.483003e-01	-4.035072e-01	-3.655762e-01	-4.935286e-01	-1.394916e-01	-1.943120e-01
75%	5.179438e-01	4.919034e-01	1.200038e-01	1.233822e-01	5.110688e-01	-1.193615e-01
max	4.107037e+00	4.668645e+00	7.851932e+00	1.328414e+01	1.054829e+01	1.899821e+01

همانطورکه میبینید پس از نرمال سازی ویژگی ها، میانگین همه ویژگی ها تقریبا **صفر** و انحراف از معیار ویژگی ها حدودا **یک** میباشد.

Encoding

همانطور که میدانید اکثر الگوریتم های یادگیری ماشین نمیتوانند با داده های Categorical کار کنند. به همین دلیل نیاز است تمامی ویژگی هایی که مقادیر غیر عددی دارند را طوری تغییر دهیم که تمامی داده های موجود در دیتاست ما عدد باشند.

برای انکد کردن ویژگی های Categorical به دو روش عمل میکنیم. برای ویژگی هایی که ترتیب دارند(مانند مقطع تحصیلی : دیپلم/کارشناسی/کارشناسی ارشد/...) کافیست هر رشته را با یک عدد به ترتیب جایگزین کنیم. اما همانطور که بارها گفته شد ما نمیدانیم در این دیتاست چه ویژگی هایی مقادیر ترتیبی دارند و چه ویژگی هایی خیر. به عنوان مثال مقطع تحصیلی میتواند یکی از همین ۱۵ ویژگی دیتاست ما باشد. به همین علت از راه دیگری به اسم One-hot Encoding برای همه ستون ها استفاده میکنیم.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	-0.062276	-0.955920	u	g	w	v	-0.290872	t	t	-0.287892	f	g	0.104469	-0.195272	+
1	a	2.286443	-0.060007	u	g	q	h	0.244013	t	t	0.740293	f	g	-0.819095	-0.087788	+
2	a	-0.596305	-0.855481	u	g	q	h	-0.216167	t	f	-0.493529	f	g	0.557537	-0.037117	+
3	b	-0.315370	-0.646569	u	g	w	v	0.456175	t	t	0.534656	t	g	-0.488006	-0.194696	+
4	b	-0.961605	0.174015	u	g	w	v	-0.153415	t	f	-0.493529	f	s	-0.371835	-0.195272	+

و برای ستون آخر که Label ما است از یک Integer Encoding ساده استفاده میکنیم به این شکل که به جای علامت – عدد صفر و به جای علامت + عدد ۱ را جایگزین میکنیم.

```
1  # Integer encode the target column
2  df.A16.replace(['+', '-'], [1, 0], inplace=True)
✓ 0.1s
```

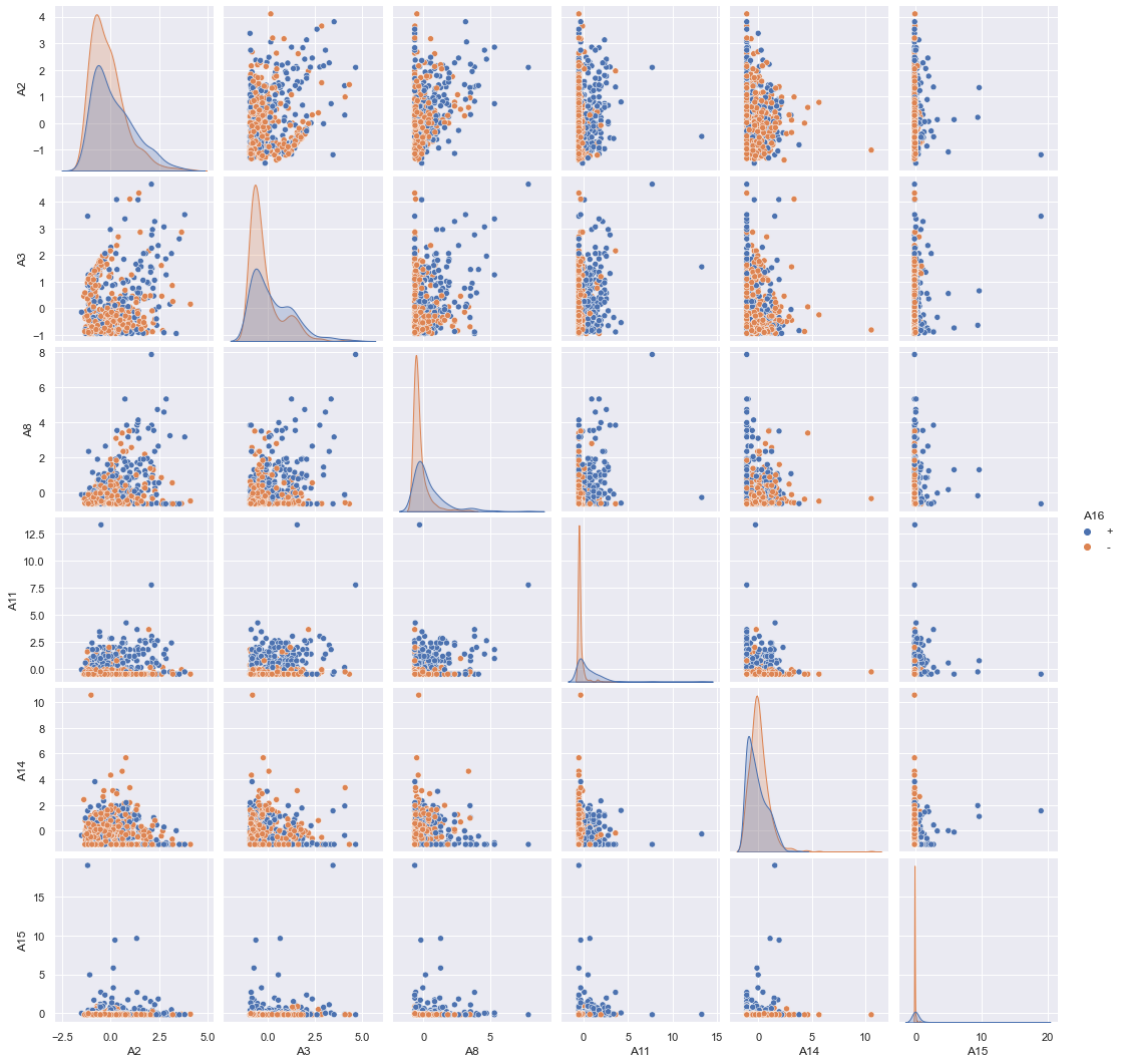
۲)

در ادامه دیتاست را به دو دسته آموزش و آزمون تقسیم میکنیم (۸۰٪ آموزش ـ ۲۰٪ آزمون). در تصویر زیر تعداد داده های موجود در هر دسته را مشاهده میکنید.

```
X size = 690
y size = 690
Train X size = 552
Train y size = 552
Test X size = 138
Test y size = 138
```

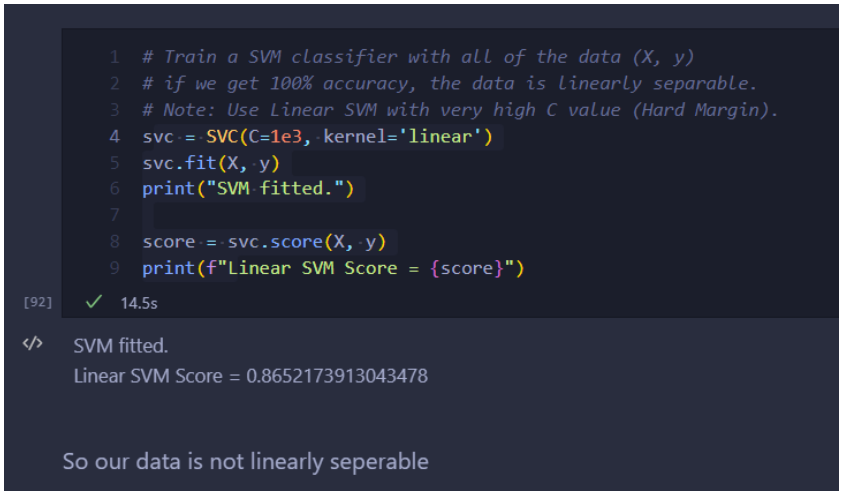
نکته : دلیل اینکه دسته Validation را در این مرحله تولید نکردیم این است که API تنسورفلو پارامتری به اسم Validation_split دارد که به طور خودکار بخشی از دسته آموزش را برای ارزیابی استفاده میکند.

پس از انجام جداسازی میتوانیم ویژگی های مختلف را دو به دو در ارتباط با هم رسم کنیم و مشاهده کنیم که آیا دو ویژگی وجود دارند که بتوانند به صورت خطی تمام داده ها را جدا کنند یا خیر؟



این نمودار اطلاعات زیادی به ما نمیدهد چون تا جایی که دیده میشود هیچ دو ویژگی ای وجود ندارند که به طور خطی داده هایی با برچسب های مختلف را از هم جدا کنند.

در ادامه برای بررسی جداپذیری خطی دیتاست از Hard Margin Linear SVM استفاده میکنیم. به این شکل که تمام داده را به SVM میدهیم و انتظار داریم اگر داده ها جداپذیر خطی باشند، SVM با دقت ۱۰۰ درصد آنها را دسته بندی کند. پس از انجام آزمایش متوجه میشویم که SVM به دقت حدود ۸۰ درصد میرسد که به این معنی است که داده ها جداپذیر خطی نیستند.

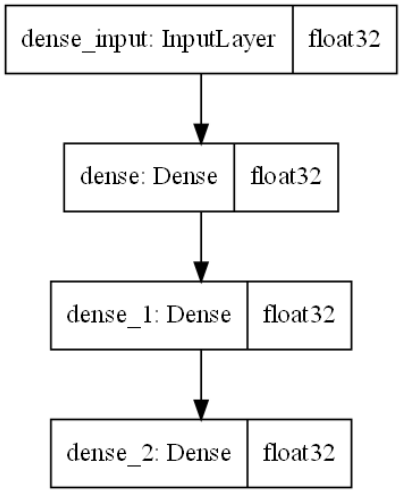


۳)

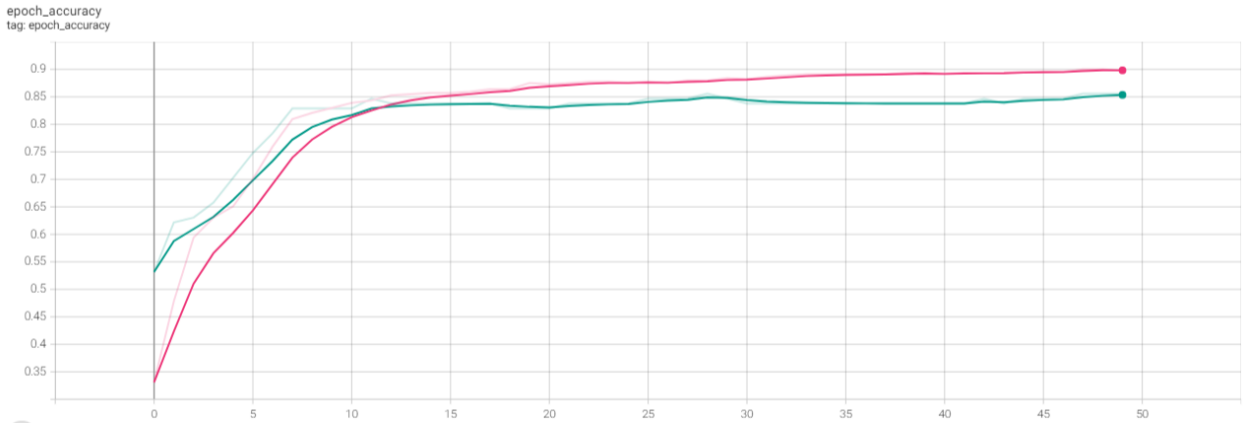
در این بخش یک شبکه پرسپترونی چند لایه برای دسته بندی داده ها میسازیم. از آنجایی که متوجه شدیم داده ها به صورت خطی جدایی پذیر نیستند لازم است تعداد لایه های مخفی این شبکه را حداقل ۲ در نظر بگیریم تا ارتباطات غیرخطی بین داده ها هم پیدا شوند.

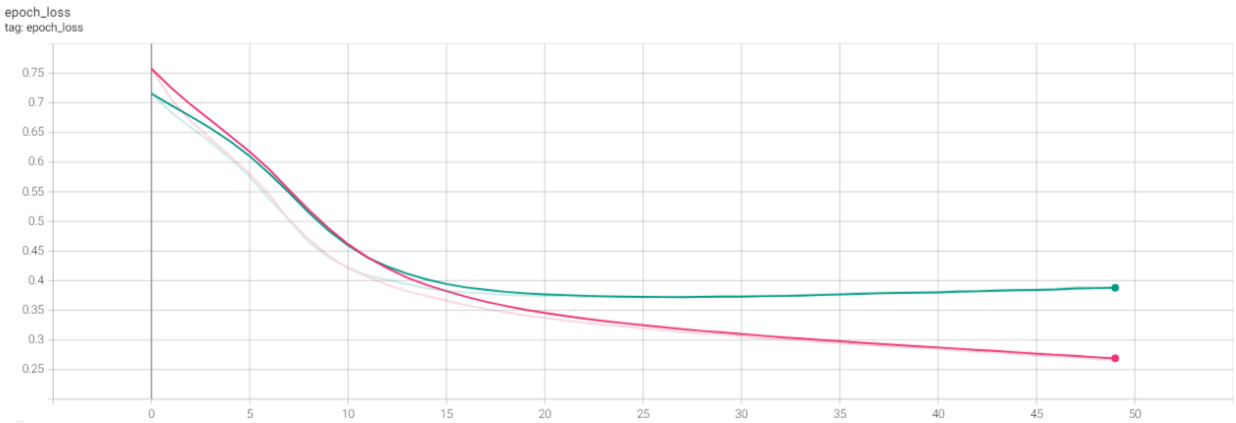
در تصویر زیر میتوانید ساختار شبکه ذکر شده را مشاهده کنید. همانطورکه مشخص است برای شروع، دو لایه مخفی از نوع Dense یا تماماً متصل داریم. همچنین به عنوان ورودی مدل به ازای هر کدام از ۴۷ ویژگی یک نورون ورودی داریم. تعداد نورون های لایه مخفی در این مدل به خصوص برابر با ۱۰ در نظر گرفته شده اند. تابع فعال سازی لایه های مخفی Relu و تابع فعال سازی لایه آخر Sigmoid است. برای آموزش از Adam به عنوان الگوریتم بهینه سازی و از تابع sparse categorical crossentropy به عنوان loss function استفاده شده است.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 10)                470
_________________________________________________________________
dense_4 (Dense)              (None, 10)                110
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 22
=================================================================
Total params: 602
Trainable params: 602
Non-trainable params: 0
```



پس از اتمام آموزش این مدل، نمودار های صحت بر اساس epoch و loss بر اساس epoch را رسم کردیم. (نمودار قرمز آموزش و سبز اعتبارسنجی است)

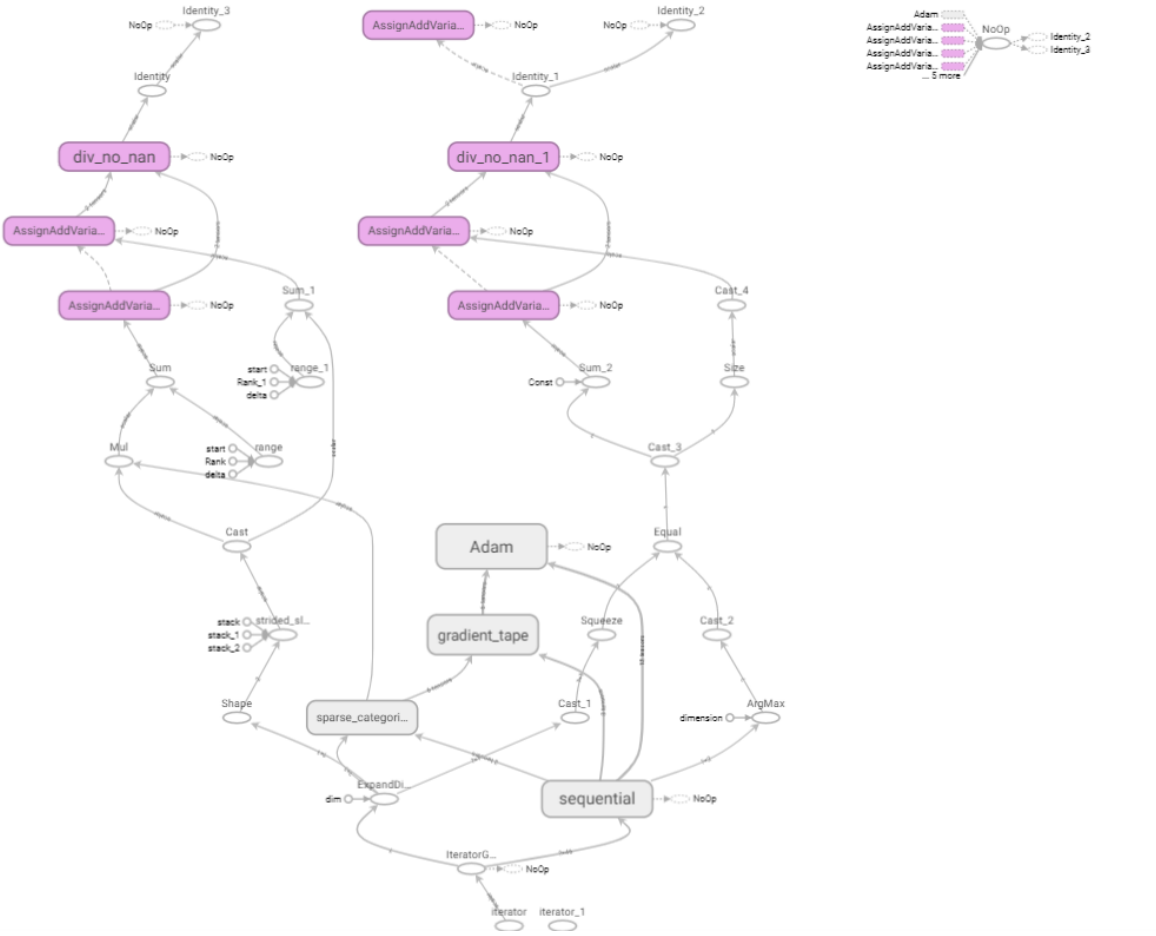




همانطورکه مشاهده میکنید نمودار epoch-accuracy صحت حدود ۸۵ درصد پس از ۵۰ epoch را نمایش میدهد اما اگر به نمودار epoch-loss توجه کنیم میبینیم که منحنی validation_loss (سبز رنگ) پس از حدود ۲۰ epoch دیگر کاهش نمی یابد در صورتی که training_loss همچنان درحال کاهش است. این به این معنی است که مدل از اینجا به بعد در حال overfit شدن است.

در بخش های بعدی با استفاده از Hyper parameter Tuning و Regularization سعی میکنیم جواب بهتری از یک شبکه پرسپترونی چند لایه بگیریم.

گراف حاصل از این مدل را در تصویر زیر مشاهده میکنید. (خروجی تب *graph* از *Tensorboard*)

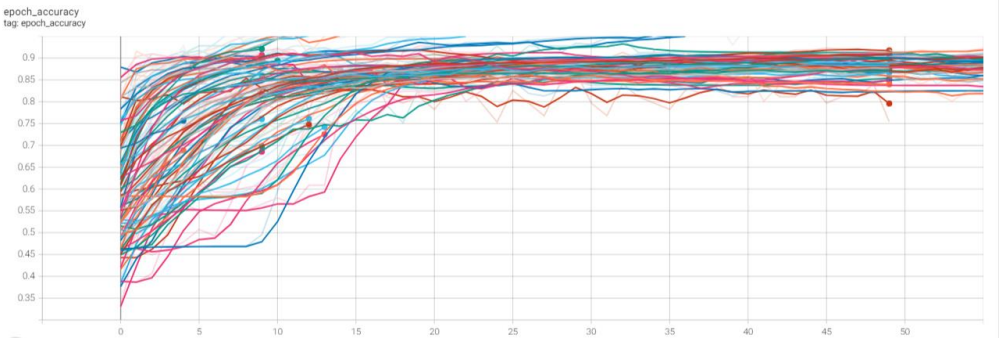


۴)

در این بخش از سوال میخواهیم سعی کنیم بهترین ابرپارامترها را برای مدل خود پیدا کنیم. برای این منظور تعدادی متغیر داریم:

- *تعداد لایه های مخفی*
- *تعداد نورون های هر لایه مخفی*
- *تابع فعال سازی هر لایه*
- *و ...*

پس از اجرای تعداد زیادی آزمایش با مدل های مختلف به یک سری نتایج رسیدیم.



اول اینکه با توجه به تعداد فیچری که داریم (۴۷) و تعداد داده آموزش موجود (۴۸۳)، مدل های بسیار عمیق و یا بسیار پهن خیلی زود overfit میشوند. به همین دلیل در انتخاب های نهایی بر روی مدل هایی با تعداد لایه کمتر و تعداد نورون های کمتر تمرکز کردیم. همچنین تغییر دادن تابع فعال سازی لایه ها تاثیر زیادی نداشت به همین دلیل برای سادگی مقایسه کم کردن تعداد حالت های ممکن برای همه لایه ها از Relu استفاده کردیم.

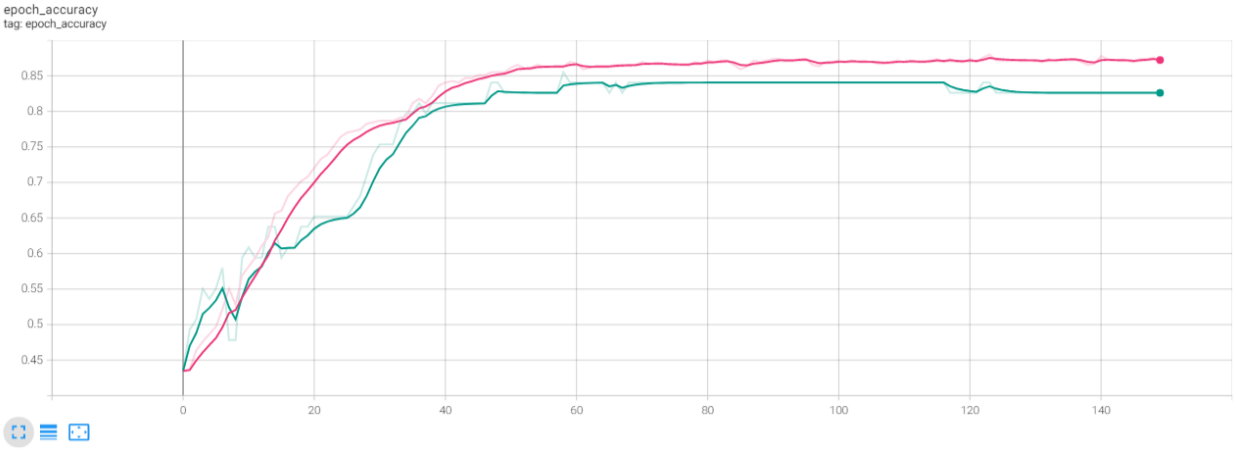
همچنین در زمینه پیدا کردن بهترین ابرپارامترها از Keras Tuner نیز استفاده شد. (برای دیدن تمام آزمایش ها و نتایج مدل های مختلف به نوت بوک مراجعه کنید.

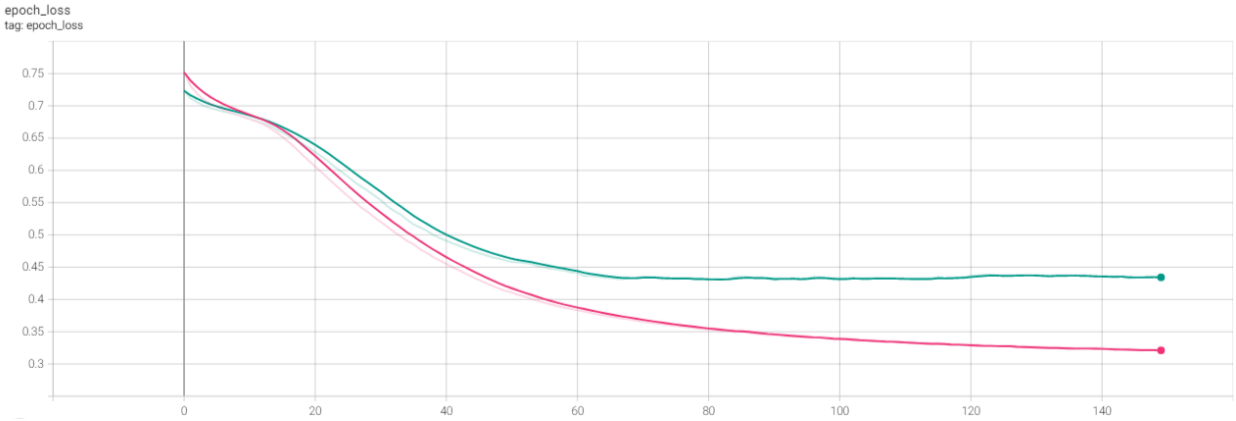
در نهایت مدلی که در تصویر زیر میبینید به عنوان بهترین مدل انتخاب شد.

```
model = Sequential([
    Dense(2, activation='relu', input_shape=(X.shape[1],)),
    Dense(4, activation='relu'),
    Dense(2, activation='softmax'),
])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 2)                 94
_________________________________________________________________
dense_28 (Dense)             (None, 4)                 12
_________________________________________________________________
dense_29 (Dense)             (None, 2)                 10
=================================================================
Total params: 116
Trainable params: 116
Non-trainable params: 0
```

نمودار های epoch-accuracy و epoch-loss را برای بهترین مدل میتوانید در تصاویر زیر مشاهده کنید. (نمودار قرمز آموزش و سبز اعتبارسنجی است)






```
1 model.evaluate(X_test, y_test)
[105] ✓ 0.3s
5/5 [==============================] - 0s 1ms/step - loss: 0.2542 - accuracy: 0.9058
[0.2541724145412445, 0.9057971239089966]
```

صحت تست بر روی این مدل برابر با **۹۰ درصد** بود.

برای مقایسه از XGBOOST برای دسته بندی داده ها استفاده کردیم و صحت ۸۴ درصد را بدست آوردیم.

از جمله مشکلاتی که در این مسئله وجود داشت تعداد کم داده اعتبارسنجی بود. با توجه به اینکه ۶۹۰ داده داشتیم، تعداد داده ها در دسته اعتبارسنجی برابر با ۶۹ عدد بود که این باعث می‌شد بین ران های مختلف مقادیر مختلفی بدست بیاید و همچنین نتیجه validation_acc, validation_loss خیلی معتبر نباشند. این که در این مسئله از one hot encoding هم استفاده کردیم و تعداد ویژگی ها به ۴۷ عدد رسید هم در این مشکل بی تاثیر نبود.

اما با این وجود با مقایسه نمودار ها و نتیجه صحت تست بین این مدل و مدل بخش قبل میتوانیم ببینیم که مدل های کوچک تر برای این مسئله مناسب تر هستند.

۵)

در این بخش سعی داریم مدل را به طور مصنوعی طوری تغییر دهیم که مدل Overfit شود. لازم است قبل از انجام این کار کمی به دلایل عمده Overfitting آشنایی داشته باشیم. یکی از دلایل عمده بیش برازش کمبود داده آموزش است. هر چه داده آموزش کمتر باشد مدل به راحتی بیشتر میتواند داده های آموزش را حفظ کند. در این سوال تصمیم گرفتیم به تعداد داده های آموزش دست نزنیم. یکی دیگر از دلایل Overfitting پیچیدگی زیاد مدل است. به گونه ای که مدل توانایی حفظ کردن تمام داده آموزش را داشته باشد. با افزایش تعداد لایه های مخفی و زیاد کردن تعداد نورون های هر لایه میتوانیم Capacity مدل را افزایش دهیم و عمل حفظ کردن داده آموزش را برای مدل راحت تر کنیم. انتظار داریم در این حالت به عدد های بسیار بالا در train_acc برسیم و valid_acc را تا حد زیادی کم کنیم. از آنجایی که تعداد داده اعتبارسنجی در این مسئله کم است ممکن است نتوانیم به خوبی این حفظ شدن داده آموزش را مشاهده کنیم و به طور شانسی دقت های بالایی بگیریم ولی با داشتن داده اعتبار و تست بیشتر باید کاهش چشم گیری در صحت اعتبارسنجی و تست مشاهده کنیم.

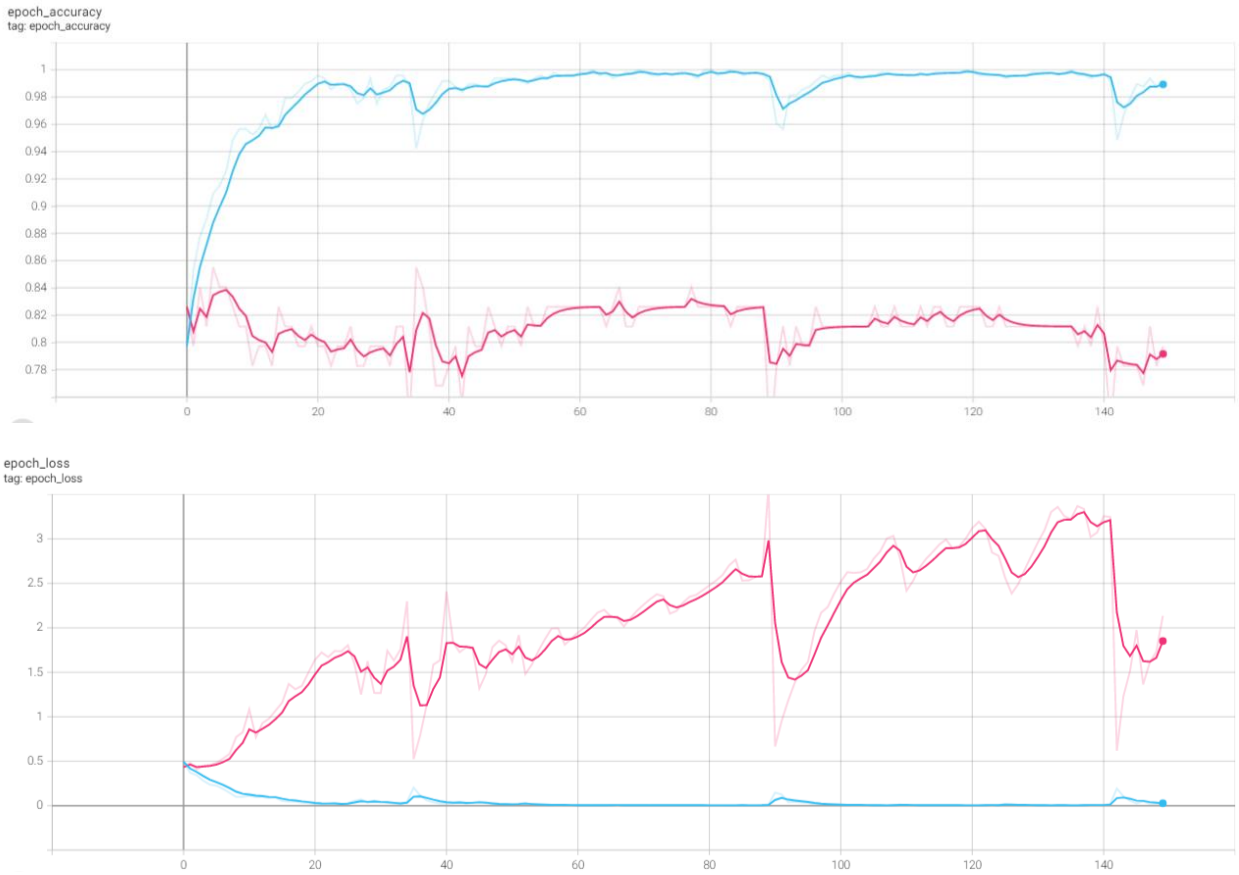
ساختار مدل را به شکل زیر تغییر میدهیم.

همانطور که میبینید هم تعداد لایه ها و هم تعداد نورون های هر لایه را به شکل چشم گیری افزایش دادیم.

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X.shape[1],)),
    Dense(128, activation='relu'),
    Dense(256, activation='relu'),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dense(2, activation='softmax'),
])
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_43 (Dense)             (None, 128)               6016
_________________________________________________________________
dense_44 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_45 (Dense)             (None, 256)               33024
_________________________________________________________________
dense_46 (Dense)             (None, 512)               131584
_________________________________________________________________
dense_47 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_48 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_49 (Dense)             (None, 2)                 258
=================================================================
Total params: 351,618
Trainable params: 351,618
Non-trainable params: 0
_________________________________________________________________
```

در نمودار های مربوطه هم به راحتی میتوان Overfitting را مشاهده کرد. صحت آموزش خیلی زود به اعدادی نزدیک به ۱۰۰ درصد رسیده و لاس مربوط به آموزش هم بسیار به صفر نزدیک شده است. (نمودار آبی آموزش و قرمز اعتبارسنجی است)



تعداد وزن های قابل تنظیم این مدل برابر حدود ۳۵۰ هزار عدد است.

۶)

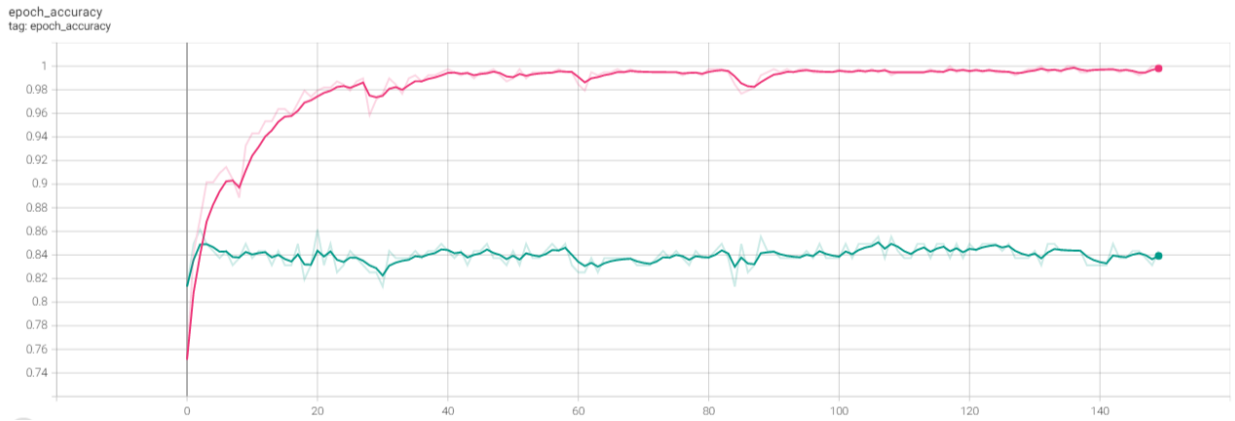
در این بخش سعی داریم با استفاده از روش های مختلفی مانند استفاده از L1|L2 Regularization و Dropout مدل بخش قبل را بهبود ببخشیم و از حالت Overfit در بیاوریم.

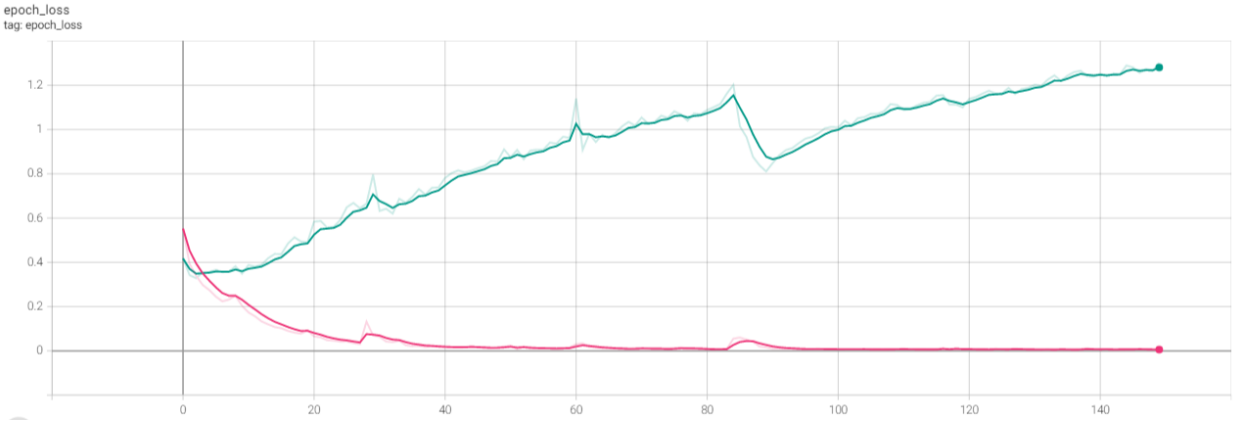
از بین این دو روش Regularization کمک میکند عدد متناظر با وزن ها خیلی زیاد نشوند. و در واقع بزرگی وزن ها را به عنوان یک جریمه به loss function اضافه میکند و باعث میشود مدل وزن های بسیار زیادی نداشته باشد. این کار باعث میشود با وجود اینکه تعداد نورون های زیادی داریم اما تاثیر هر نورون کم تر باشد. اگر از L1 استفاده کنیم حتی باعث میشود بعضی وزن ها صفر شوند که باعث میشود پیچیدگی مدل کاهش پیدا کند.

و Dropout باعث میشود در حین آموزش یک سری از نورون ها خاموش شوند و عملا پیچیدگی مدل را کاهش میدهد. دلیل این که این کار مفید است این است که وابستگی نورون ها به یکدیگر را از بین میبرد و باعث میشود هر نورون اطلاعات مفید و یکتایی را مستقل از اطلاعات سایر نورون ها (ی مجاورش) بیاموزد.

```
model = Sequential([
    Dense(256, activation='relu', input_shape=(X.shape[1],), ),
    Dense(128, activation='relu', ),
    Dense(64, activation='relu', ),
    Dense(2, activation='softmax'),
])
```

(***پیش از تعمیم پذیری**: نمودار قرمز آموزش و سبز اعتبارسنجی است*)



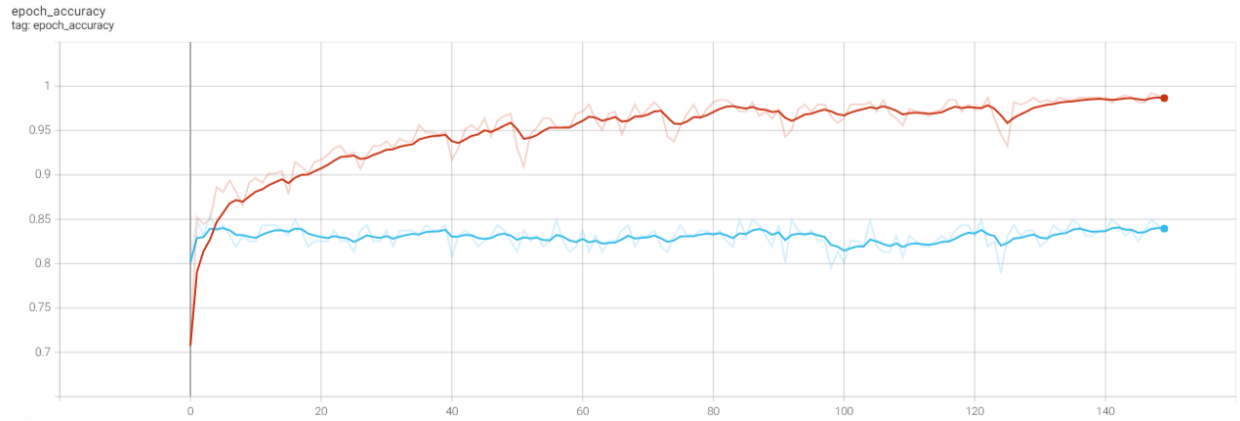


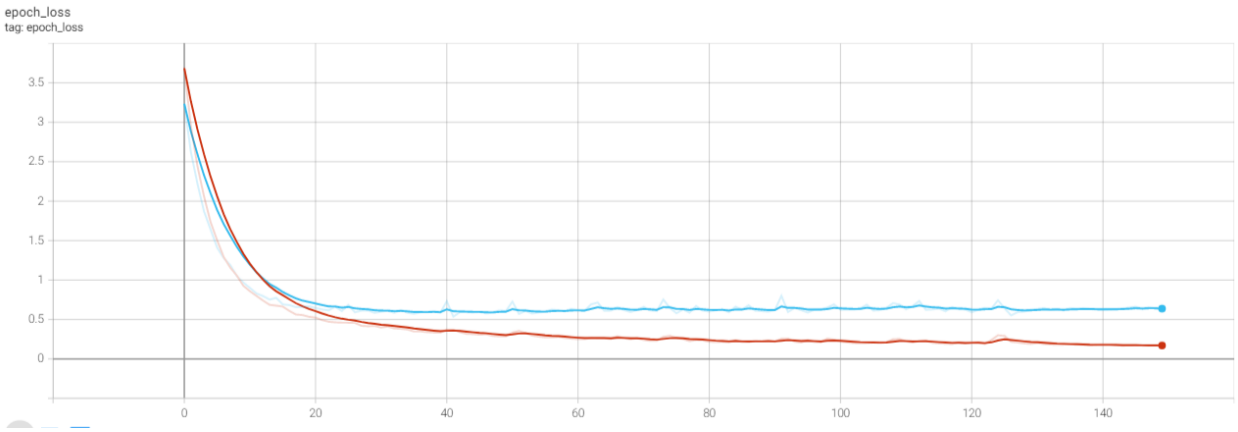
ساختار مدل پس از اعمال Regularization و Dropout را مشاهده میکنید.

```
model = Sequential([
    Dense(256, activation='relu', input_shape=(X.shape[1],), kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dropout(0.2),
    Dense(2, activation='softmax'),
])
```

در تصاویر زیر میتوانید نمودار های epoch-loss و epoch-accuracy را بعد از اعمال Regularization , Dropout مشاهده کنید.

(**پس از تعمیم پذیری**: نمودار نارنجی آموزش و آبی اعتبارسنجی است)





پایان