به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری 2

نام و نام خانوادگی	مهسا مسعود
شماره دانشجویی	810196635
تاریخ ارسال گزارش	00/02/03

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

سوال 1 – MLP(Regression)

الف) ابتدا داده را با کمک کتابخانه pandasمیخوانیم. هدف این است که با کمک یک شبکه MLP قیمت خانه را از روی ویژگی های آن پیش بینی کنیم.

دیتا ست داده شده از 81 ستون است که در اصل به جز Id و Sale_price ، 79 ویژگی مربوط به خانه مثل اندازه، خیابان، همسایگی و.. را بررسی میکند که مربوط به 1460 خانه است. ویژگی ها از نوع object , float64 , int64 هستند که نیاز به پیش پردازش دارند. ابتدا مقادیری که null value درصد زیادی از آن ها (بیشتر از 40) را تشکیل میدهد پیدا کرده و حذف میکنیم:

این ستون ها برابر با Alley , FireplaceQu, PoolQC , Fence, MiscFeature هستند.

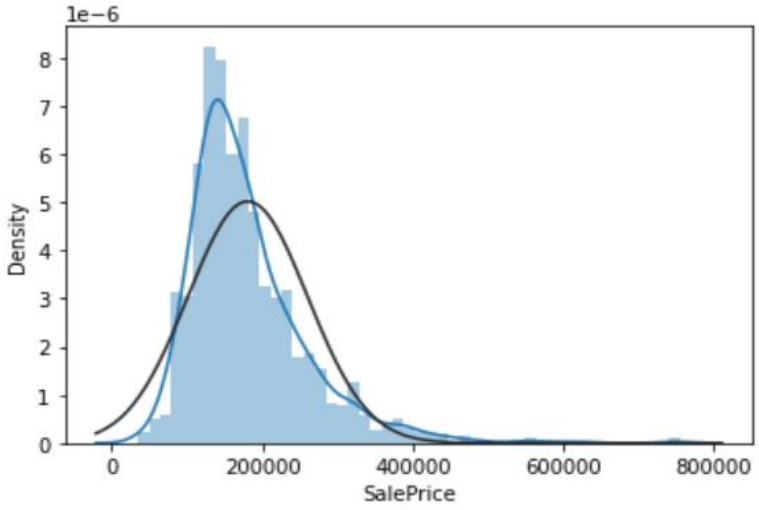
پس از حذف ستون های مذکور تعدادی ستون با مقادیر NAN محدود داریم که باید از جایگزین mean برای پر کردن ستون آن ها استفاده کنیم.

اما قبل از اینکار باید همه داده ها را numericکنیم تا همه دیتا ست قابل تبدیل به ماتریسی از اعداد شود و قابل محاسبه باشد. برای این کار از labelencoderکه یک کتابخانه دیفالت برای انکود کردن است استفاده میکنیم و سپس fit_transform را روی داده ها صدا میزنیم.

حال با دستور get_dummies متغیر های categoricalرا به indicator variables تبدیل میکنیم و در نهایت یک دیتا ست عددی جدید با 204 ستون داریم.

ب)

قبل از انجام training ، با استفاده از هیستوگرام یک شهودی از تجمع اکثریت قیمت ها پیدا میکنیم و به صورت زیر نمایش میدهیم:



شکل 1-1 - چگالی توزیع قیمت

در این قسمت برای دیتا های numeric باید نرمالیزیشن انجام دهیم تا شبکه مستقل از مقدار ویژگی ها تصمیم گیری کند. پس این داده ها را با استفاده از فرمول $x \leftarrow \frac{x-\mu}{\sigma}$ نرمالایز میکنیم.

در این مرحله از تابع train_test_split استفاده کرده و تقسیم بندی 80 به 20 را برای داده های train و test انجام میدهیم.

باید دقت کنیم که ستون های Idو SalePrice از داده های آموزشی جدا باشند و داده ی تست شامل ستون SalePrice باشد.

با استفاده از کتابخانه keras از مدل sequentialاستفاده میکنیم و شبکه مد نظر را ابتدا با یک لایه مخفی میسازیم . به این صورت که لایه اول دارای 10 نورون بوده و 202 ویژگی هرخانه را به عنوان ورودی دریافت میکند و لایه دوم همان لایه خروجی است که یک نورون را به عنوان خروجی شبکه می سازد. لازم به ذکر است که لایه ها dense و یا fully-connected هستند. در لایه پنهان از تابع های فعال ساز relu و softmax استفاده گردیده است. اپتیمایزر نیز Adam میباشد.

خلاصه ای از شبکه :

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                2030
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 2,041
Trainable params: 2,041
Non-trainable params: 0
_________________________________________________________________
```

همین کار را عینا برای تولید شبکه با دو لایه مخفی انجام میدهیم که در این حالت، یک لایه 8 نورنه در وسط دولایه مذکور در بالا قرار میگیرد. نتایج حاصله نشان میدهد که در شبکه با دولایه مخفی به مقدار loss کمتری میرسیم چون شبکه فیلتر های بیشتری رو پارامتر ها ایجاد میکند و دقیق تر میشود.

مقادیر loss در مرحله آموزش:

جدول 1 – مقادیر loss در مرحله training

تعداد لایه مخفی	تابع فعال ساز	Loss	Validation loss
1	Relu	0.0142	1.3981
1	Softmax	0.0436	0.5003
2	Relu	0.0087	0.5065
2	Softmax	0.1162	0.4906

برای اینکه بهترین معماری شبکه را بین حالات مذکور پیدا کنیم ، روی داده ی تست مدل را پیاده میکنیم و کمترین loss حاصله را به منظور بهترین مدل بین این 4 مدل انتخاب میکنیم:

مقادیر loss در مرحله تست:

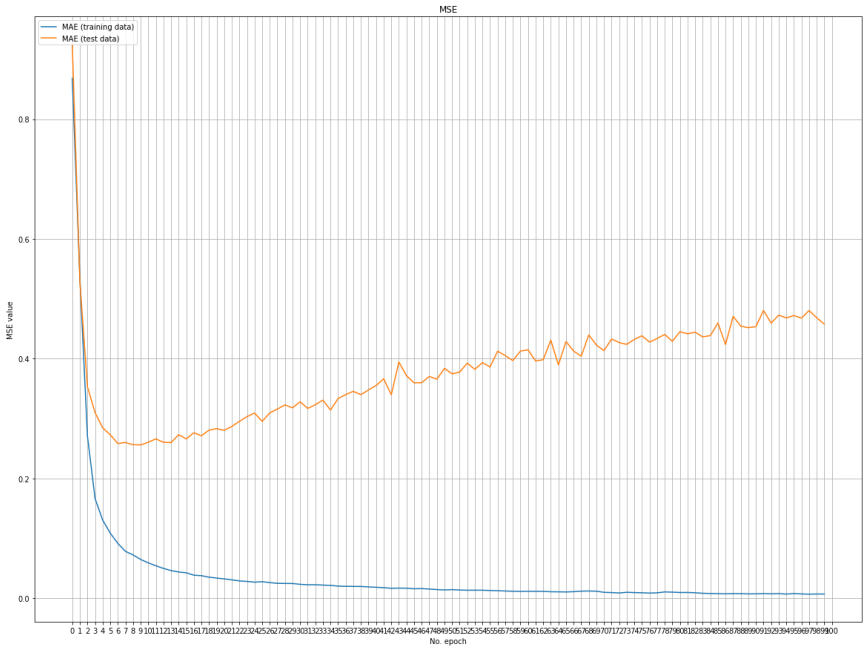
جدول 2 – مقادیر loss در مرحله test

تعداد لایه مخفی	تابع فعال ساز	Test Loss
1	Relu	0.2234
1	Softmax	0.2598
2	Relu	0.150
2	Softmax	0.330

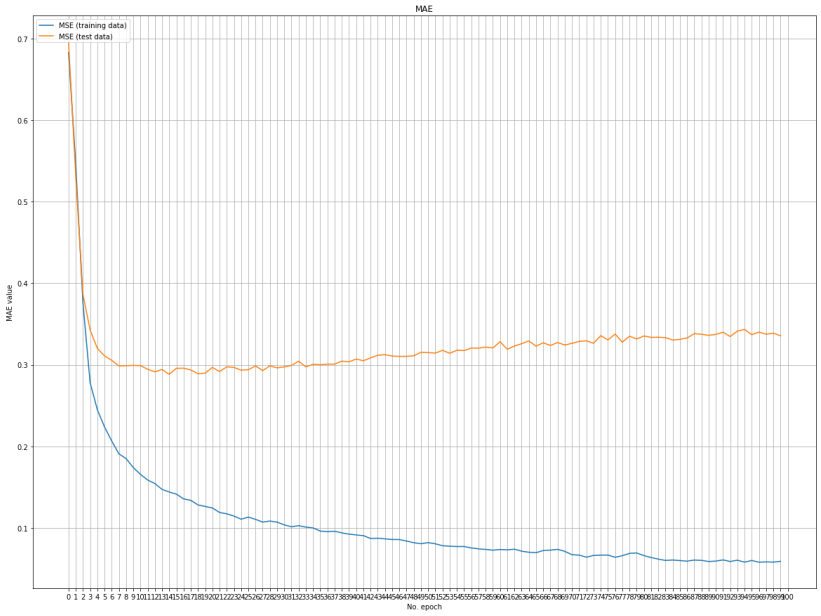
واضح است که بهترین مدل شبکه ی 2 لایه ای با تابع فعالساز relu می باشد.

ج) بهترین حالتی که از قسمت قبل به دست آمد(از نظر accuracy و مقدار loss) شبکه 2 لایه ای با اکتیویشن فانکشن relu میباشد و برای قسمت ج و د روی این شبکه نتایج حاصل شده است.

اگر تابع MSE را به عنوان تابع loss در نظر بگیریم:



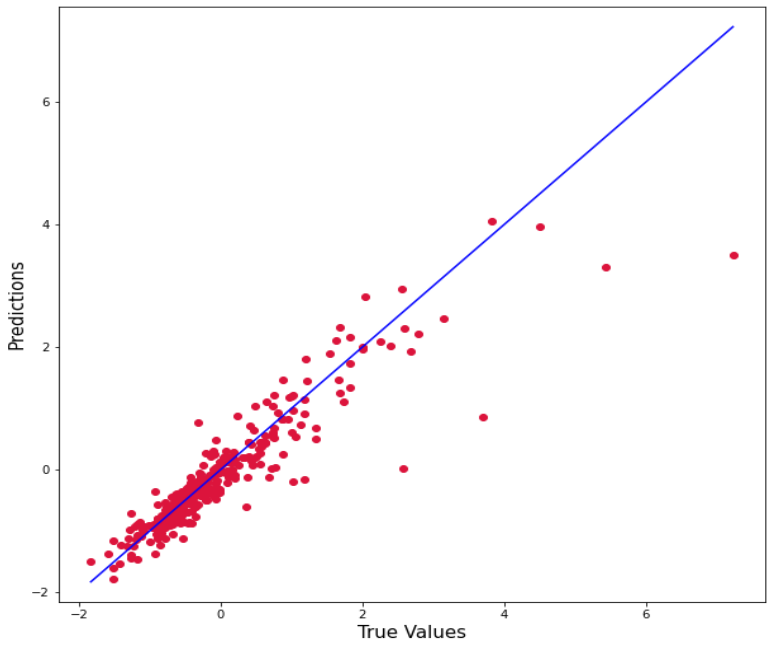
شکل 1-2- مقدار MSE برحسب epoch در حالت loss = MSE



شکل 1-3 - مقدار MAE برحسب epoch در حالت loss = MSE

تعداد ایپاک بهینه برای این حالت مینیمم بین ایپاک های بهینه در هر دو شکل میباشد . برای متریک MSE این عدد برابر با 10 و برای معیار MAE برابر با 34 می باشد(این مقادیر از روی نمودار با سایز برزگتر که خط کشی شده و اعداد روی محور ها با فاصله 1 به دست آمدند.)

نمودار مقادیر پیش بینی شده بر حسب مقادیر واقعی:

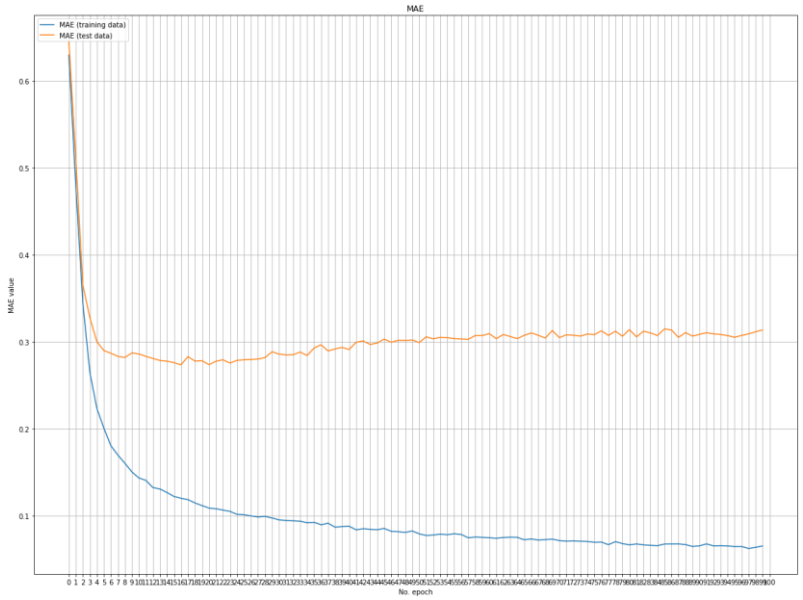


شکل 1-4 - مقادیر پیش بینی شده بر حسب مقادیر واقعی

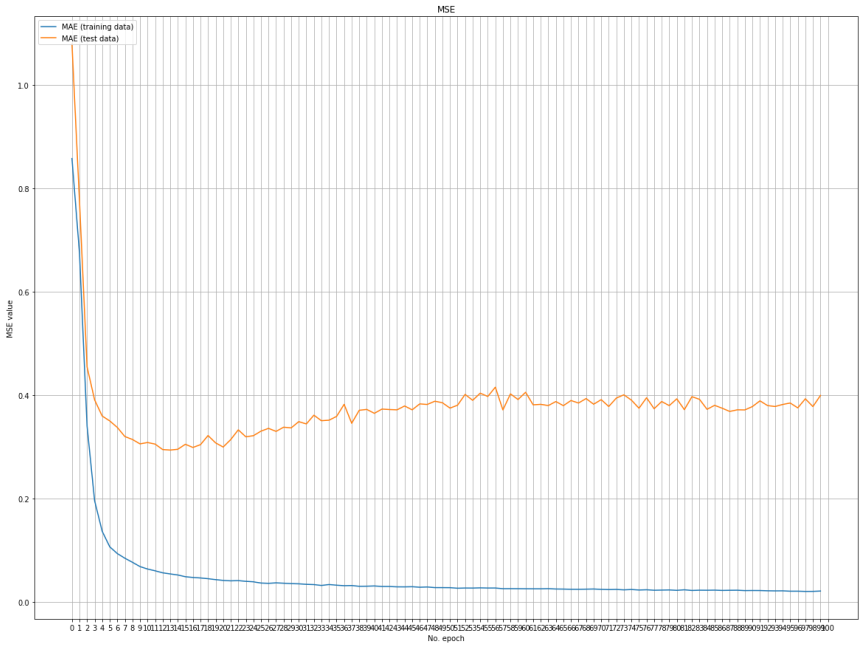
همانطور که از scatterplot ستون SalePrice و مقادیر predict شده مشاهده می شود مقادیر پیش بینی شده و مقادیر واقعی به خط x=y نزدیک هستند که نشان از پیش بینی درست مدل میباشد.

د)

اگر تابع MAEرا به عنوان تابع loss در نظر بگیریم:



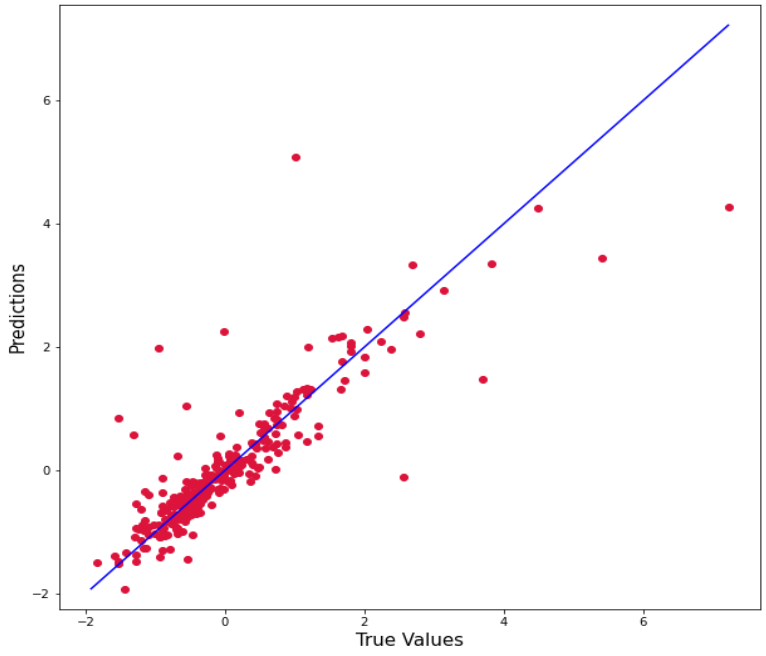
شکل 1-5 - مقدار MAE برحسب epoch در حالت loss = MAE



شکل 1-6 - مقدار MSE برحسب epoch در حالت loss = MAE

تعداد ایپاک بهینه برای این حالت مینیمم بین ایپاک های بهینه در هر دو شکل میباشد . برای متریک MSE این عدد برابر با 23 و برای معیار MAE برابر با 23 می باشد(این مقادیر از روی نمودار با سایز برزگتر که خط کشی شده و اعداد روی محور ها با فاصله 1 به دست آمدند)

نمودار مقادیر پیش بینی شده بر حسب مقادیر واقعی:



شکل 1-7 - مقادیر پیش بینی شده بر حسب مقادیر واقعی

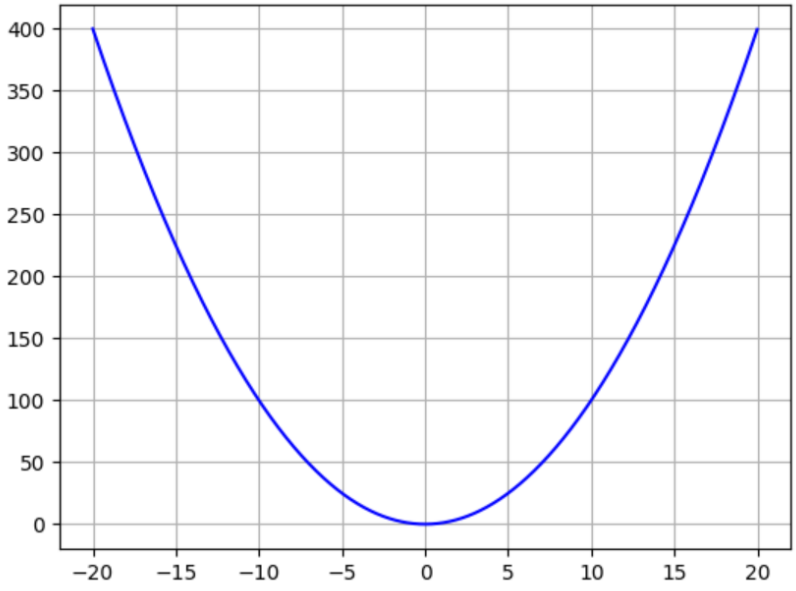
همانطور که از scatterplot ستون SalePrice و مقادیر predict شده مشاهده می شود مقادیر پیش بینی شده و مقادیر واقعی به خط x=y نزدیک هستند که نشان از پیش بینی درست مدل میباشد.

ه)

MSE: رایج ترین معیار loss function در مسایل رگرسیون است :

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

MSE هیچوقت منفی نخواهد بود و در فرمول بالا N نشانه ی تعداد سمپل ها و y لیبل واقعی و y_hat مقدار پیش بینی شده است.



شکل 1-8 - MSE loss function

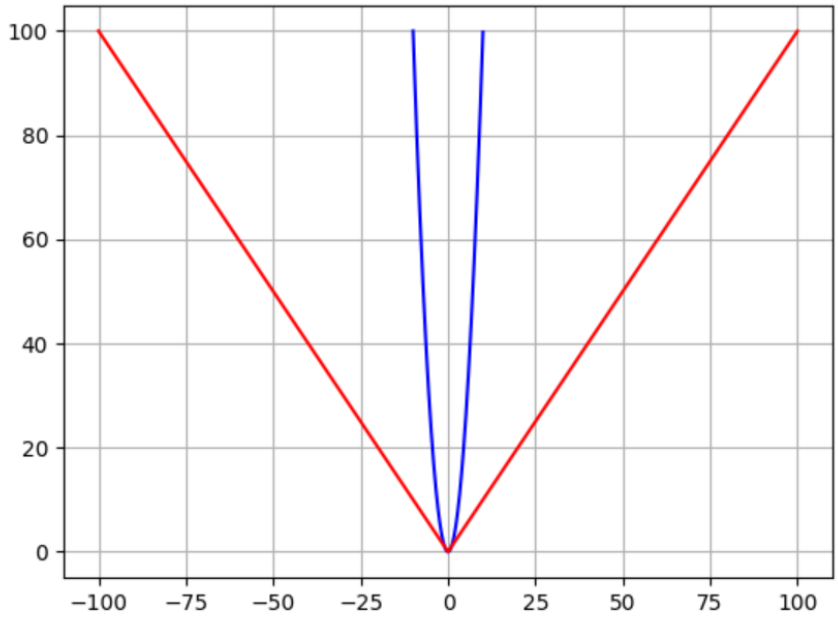
این معیار از نظر اینکه میتوانیم مطمین باشیم در outlierها ارور بزرگی نخواهیم داشت خوب است زیرا مقدار وزن بیشتر (به دلیل ماهیت توان 2 بودن آن) به آن ها نسبت میدهد. اما این ایراد را دارد که برای مقادیر بالا بایاس میشود.

MAE :

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

این معیار نیز همواره نا منفی است و ما مقدار قدر مطلق فاصله بین لیبل اصلی و لیبل پیش بینی شده را در نظر میگیریم.

فایده ی MAE این است که مشکل اصلی MSE را حل میکند و همه ارور ها (به دلیل در نظر گرفتن فاصله مطلقشان) در یک اسکیل خطی قرار دارند. میتوان گفت که در این حالت loss function یک مقدار general خواهد داشت.اما ایراد آن این است که خطاهای بزرگی که از outlier ها ناشی میشوند وزن یکسانی با بقیه خطاها خواهند داشت و این میتواند مدل را بزرگ کند درحالیکه پیش بینی صحیحی به ما ندهد.



شکل 1-9 - MSE loss function(آبی) و MAE loss function (قرمز)

از مقایسه نمودار های قسمت قبل نیز همین نکته مشخص می شود که در معیار MSE سرعت افزایش loss در مقادیر بالا زیاد میشود و این بخاطر ترم توان 2 آن است. این loss function روی معیار MSE نیز موثر بوده و شیب آرام اما صعودی ای به آن از مقدار 0.3 به بعد داده است.

درحالیکه در loss function MAE ما شاهد تغییرات آرامتر مقدار loss هستیم (حول 0.3) و این روند حتا در ایپاک های بالا نیز آرام و smooth است و خیلی در صعودی نیست.(مگر در حالتی که متریک ما MSE باشد که آن هم شیب افزایش از حالت قبل کمتر است.)

سوال ۲ – MLP (classification)

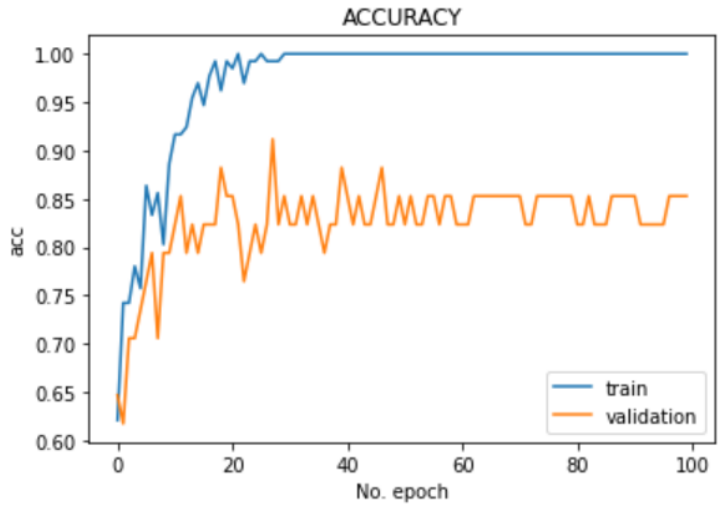
در این سوال قصد توسط MLP یا multi-layer perceptron داده هایی را classify کنیم. دیتاست ما sonar است که 60 ستون ویژگی دارد که هر یک مقادیر float دارند . هدف ما جداسازی و دادن لیبل R یا M به هر سطر است.

الف) ابتدا داده را با دستورات head و info و describe داده ها را بررسی می کنیم تا اگر مشکلی وجود دارد بر طرف کنیم. سپس توسط یک Min Max Scaler داده ها را به بازه ی 0 تا 1 میبریم و کلاس ها را ابتدا به عدد و سپس به one hot encoding تبدیل می کنیم. در نهایت داده های train و test را تقسیم می کنیم. در اینجا تقسیم بندی ساده کردیم و از روش های پیچیده تر مثل k-fold و ... استفاده نکردیم زیرا دستیار آموزشی گفتند لازم نیست و به دقت خوبی هم رسیده بودیم. در این روش به صورت رندوم 80 درصد از داده ها را train و باقی را test می گیریم سپس از بین دادگان یادگیری 20 درصد به عنوان validation مشخص میشوند که به کمک آنها hyperparameter های مدل را تنظیم می کنیم. میتوانستیم این کار را با داده های تست هم انجام بدهیم اما جواب ما biased میشد و قابل اطمینان نبود. مزیت این روش سادگی آن است و اینکه سریعتر به جواب میرسیم همچنین randomness خوبی داشتیم و داده های test در یادگیری نیامدند.

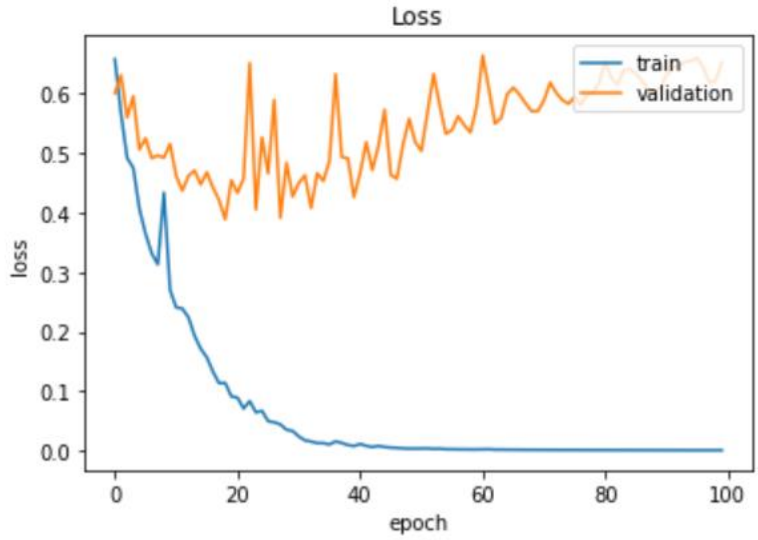
ابتدا خیلی مدلی با 2 لایه مخفی که هر یک 512 نورون باactivation function معمول یعنی relu و 2 لایه که یکی ورودی و یکی هم خروجی با activation function دیگری یعنی softmax شروع می کنیم. لایه ها fully connected می باشند پس کلا 294,914 پارامتر در مدل داریم.

ب)

ابتدا برای 100 ایپاک مقادری accuracy , loss را بررسی میکنیم:

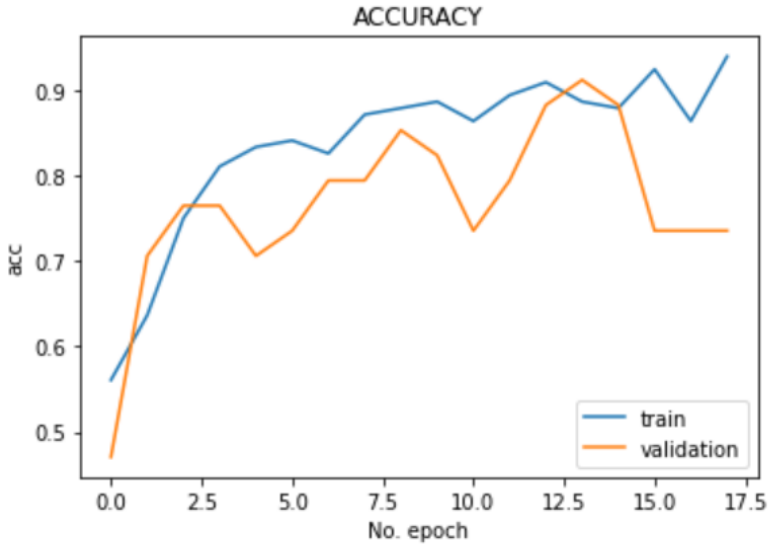


شکل 2-1 - دقت مدل در 100 ایپاک

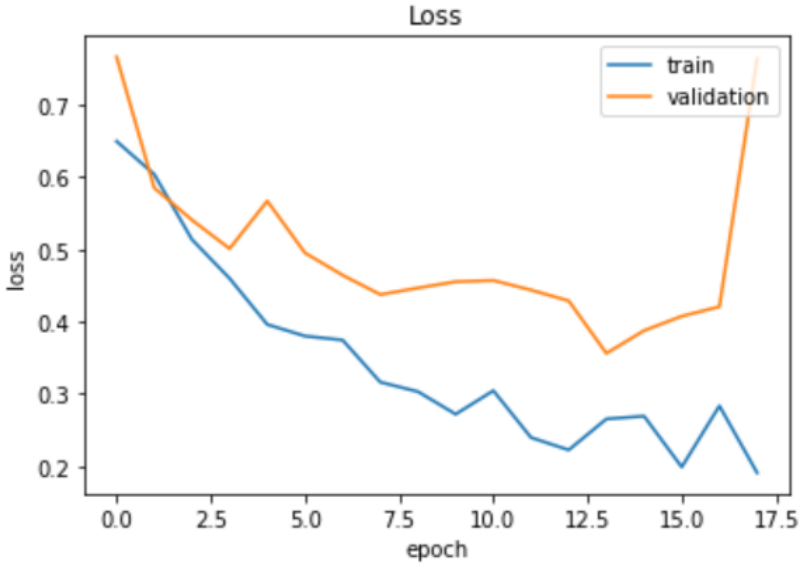


شکل 2-2 - loss مدل در 50 ایپاک

واضح است که مدل overfit شده زیرا loss برای داده validation ازحدود ایپاک هجدهم دیگر نزولی نبوده و حتی بعد از مدتی زیاد هم شده است. برای داده train هم طبق انتظار کلا نزولی است و بعد از 100 ایپاک به صفر رسیده است. پس با 18 ایپاک عمل یادگیری را به پایان میرسانیم.



شکل 2-3- دقت مدل در 18 ایپاک

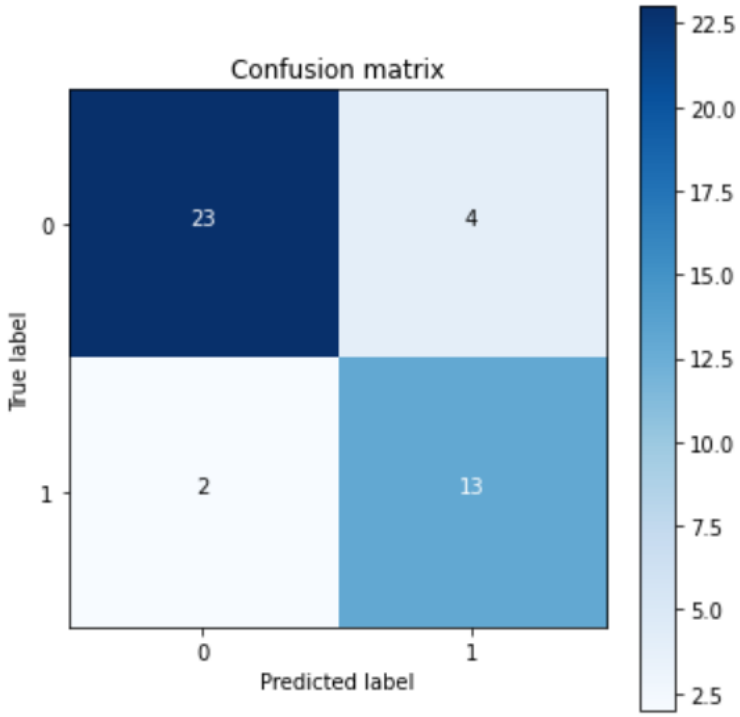


شکل 2-4 - loss مدل در 18 ایپاک

میتوان دید که به نتیجه دلخواه رسیدیم.

ج) دقت و خطا و confusion matrix مدل روی داده های تست به صورت زیر است:

```
Test Loss :  0.4204530715942383
Test Acc : 0.8571428656578064
```



شکل 2-5 - ماتریس confusion

میتوان دید که به دقت خوبی رسیده ایم و loss و confusion matrix هم این نکته را تایید می کنند.

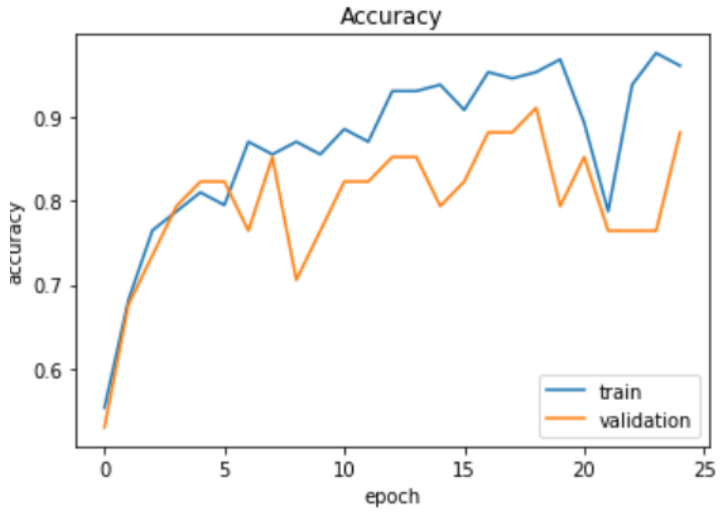
د) معیار استفاده شده categorical_crossentropy می باشد. یک معیار دیگر MSE بود که معمولا در مسایل regression توصیه می شود ولی برایbinary classification و multi-class classification معیار cross entropy مناسب تر است. آنتروپی متقاطع بین دو توزیع احتمال p و q گسسته روی یک مجموعه داده شده، به صورت زیر تعریف می‌شود:

$$H(p,q) = -\sum_{x} p(x)\ \log q(x).$$

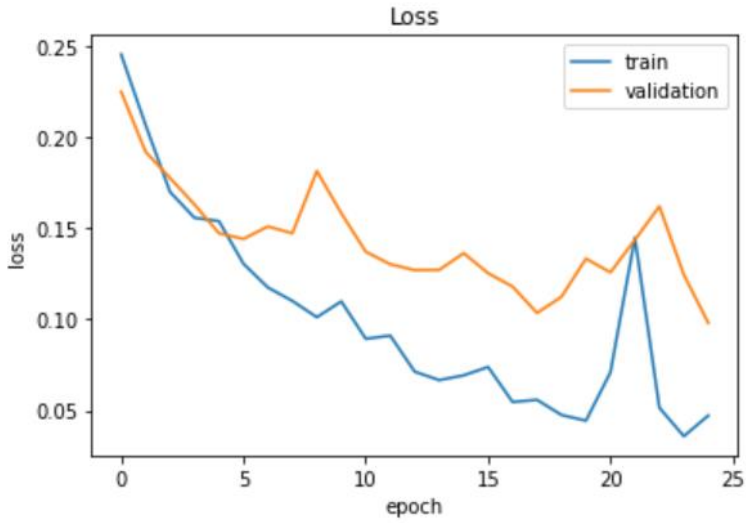
در مساله ما هم معادل $-(y\log(p)+(1-y)\log(1-p))$ می شود به عبارتی دیگر مدل ما احتمالی به عنوان خروجی می دهد بین 0 و 1 مثلا 0.1 و زمانی که لیبل درست 0 باشد loss کم خواهد بود ولی وقتی لیبل 1 باشد مقدار 0.1 حدس خوبی نبوده و نتیجه آن loss بزرگ خواهد بود. در این مساله MSE مناسب

نیست چون خطایی که محاسبه می شود نسبتا درست نمیباشد و باید معیاری انتخاب کنیم که مدل ما را به سمت انتخاب درست تر بین دو مقدار 0 یا 1 کند.

ه) این بار با MSE آموزش را انجام می دهیم و نتیجه بعد از 25 ایپاک که مقدار بهینه بود، به شرح زیر است:



شکل 2-6 - دقت بعد از 25 ایپاک با خطای MSE



شکل 2-7 - MSE loss بعد از 25 ایپاک

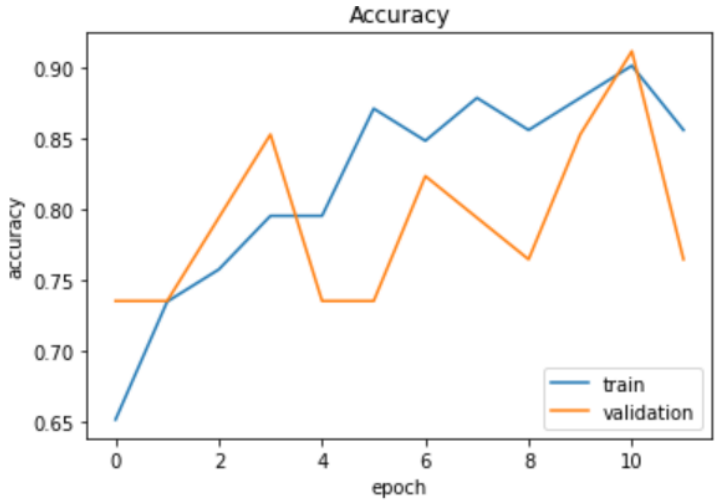
چون معیار خطاها یکسان است در دو مدل نمی توان نمودارشان را از لحاظ عددی مقایسه کرد ولی نمودار دقت روی داده های آموزش عملکرد بهتری نشان می دهد اما این نتیجه در داده های تست و

validation تکرار نشد که نشان از عملکرد ضعیف تر و generality کمتر مدل دوم است. نتایج روی داده تست به شرح زیر است:

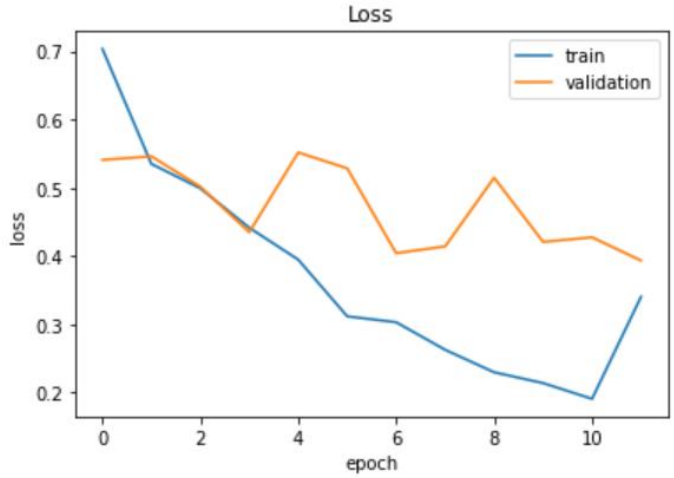
```
Test Loss 0.11421868950128555
Test Accuracy 0.8333333134651184

confusion matrix=
 [[25  2]
 [ 5 10]]
```

میتوان دید که عملکرد بدتر شده است.میتوان گفت که معیار انتخابی اولیه معیار مناسبی بوده است و به خوبی توانایی مدل را نشان می دهد.

و) حال نحوه ی ورود داده ها به مدل را تغییر می دهیم. ابتدا stochastic را بررسی می کنیم یعنی سایز batch را 1 می گذاریم:



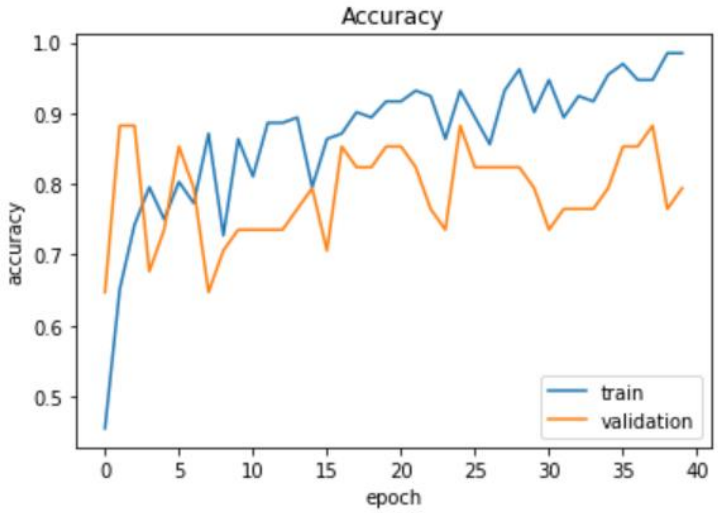
شکل 2-8 - دقت بعد از 11 ایپاک در stochastic



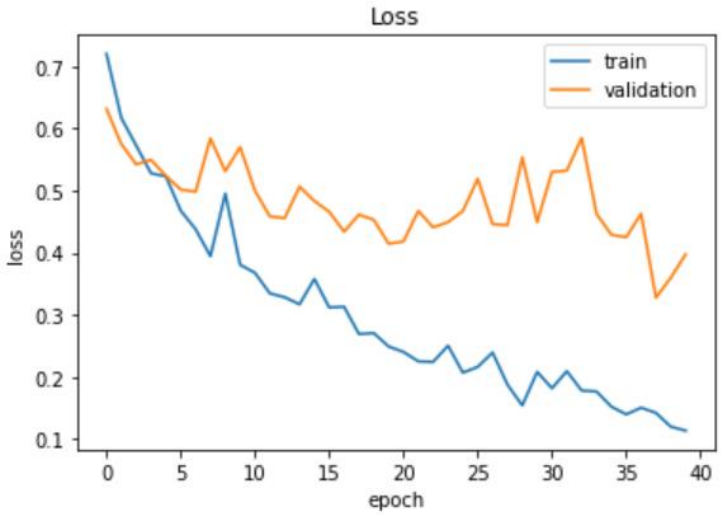
شکل 2-9 - loss بعد از 11 ایپاک در stochastic

```
Test Loss 0.5753465890884399
Test Accuracy 0.8095238208770752

confusion matrix=
 [[19  8]
 [ 0 15]]
```

برای mini batch با سایز 32 هم در مراحل قبل نتیجه را دیدیم پس اینبار برای batch size را 64 گذاشته و امتحان می کنیم:



شکل 2-10 - دقت بعد از 40 ایپاک در با batchهای 64 تایی

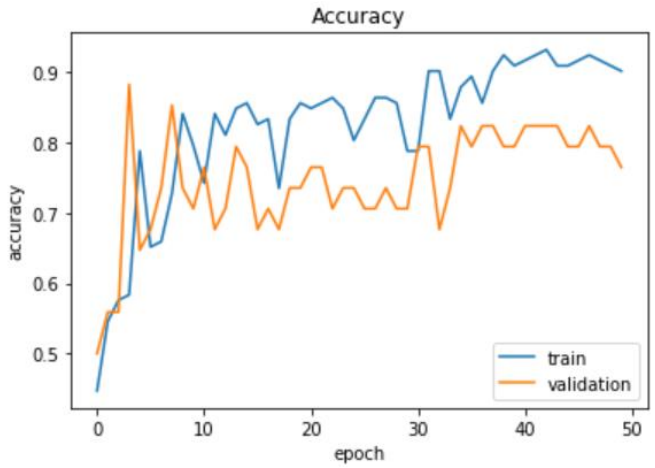


شکل 2-11 - loss بعد از 40 ایپاک در با batchهای 64 تایی

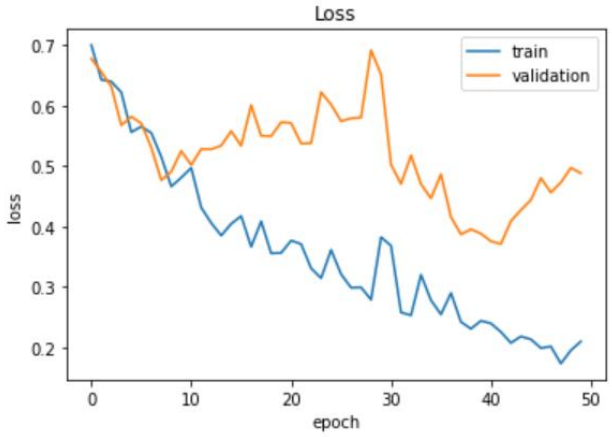
```
Test Loss 0.3551674783229828
Test Accuracy 0.8571428656578064
```

```
confusion matrix=
 [[24  3]
 [ 3 12]]
```

سپس برای سایز 128 برای batch ها بررسی می کنیم:



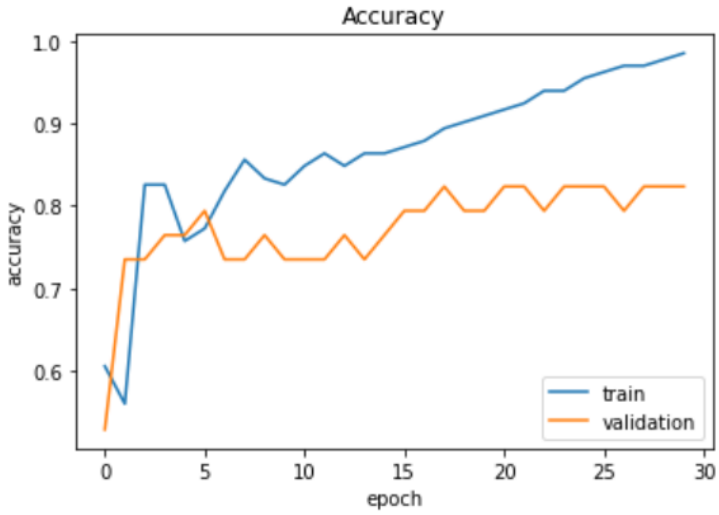
شکل2-12 - دقت بعد از 50 ایپاک با batchهای 128 تایی



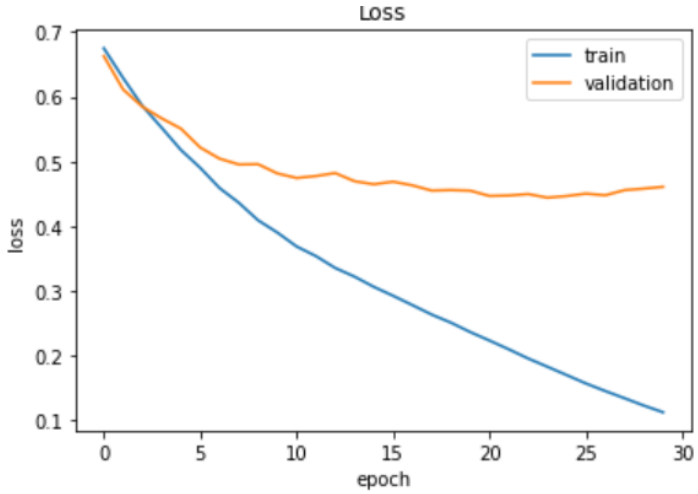
شکل 2-13 - loss بعد از 50 ایپاک با batchهای 128 تایی

```
Test Loss 0.40037035942077637
Test Accuracy 0.8095238208770752

confusion matrix=
 [[22  5]
 [ 3 12]]
```

batch را برابر 208 می گذاریم:(full batch)



شکل 2-14 - دقت بعد از 30 ایپاک با کل داده ها



شکل 2-15 - loss بعد از 30 ایپاک با کل داده ها

```
Test Loss 0.4132138788700104
Test Accuracy 0.7857142686843872

confusion matrix=
 [[21  6]
 [ 3 12]]
```

لازم به ذکر است که در تمامی حالات مقدار مناسب برای epoch را از روی نمودار خطای validation یافته ایم. اگر در نتایج دقت کنم می توان دید که هر چه اندازه batch کوچکتر بوده زمان بیشتری برای هر epoch طول کشیده و نوسان بیشتر است و در نمودارها صعود و نزول های اضافی داریم زیرا داریم با بخش کوچکتری از داده گرادیان را حساب می کنیم و ممکن است در حین یادگیری در جهت اشتباهی پیش رویم زیرا لزوما جهت گرادیان حاصل از 32 داده یکسان نیست.

هرچه batch بزرگتر شده نوسانات کمتر شده است. نتایج نهایی و دقت خیلی تغییری نکرده اند و می توان با تعیین epoch مناسب به جواب نسبتا قابل قبول رسید البته برای batch های کوچکتر مدل ناپایدارتر است و ممکن است با هر epoch ناگهان مدل از واقعیت دورتر شود. مشکل batch بزرگ هم زمان و حافظه ی زیادی است که اشغال می کند و برای داده های بزرگ مشکل زا است پس باید tradeoff بین سرعت بالا ولی نامعینی مدل و حافظه ی اضافه تر را رعایت کنیم.

علت تفاوت های نسبتا اندک نامعین بودن مدل های با batch کمتر است زیرا ممکن است پس از توقف به علت جهت های اشتباهی که در میانه مسیر رفته ایم به بهترین جواب نرسیم. معمولاbatch های کوچکتر در مجموع سریعتر به جواب می رسیم و epoch کمتری لازم است.

در ادامه ازmini batch های به سایز 32 و batch استفاده می کنیم زیرا در اولی مزایای هر دو حالت را داریم و در دومی نیز چون سایز داده کوچک است، مشکل حافظه نداریم و تشخیص epoch مناسب از روی نمودار loss آسانتر است زیرا نوسان نداریم.

ح) epoch:

برابر تعداد دفعاتی است که الگوریتم کل داده ها را دیده است .

iteration :

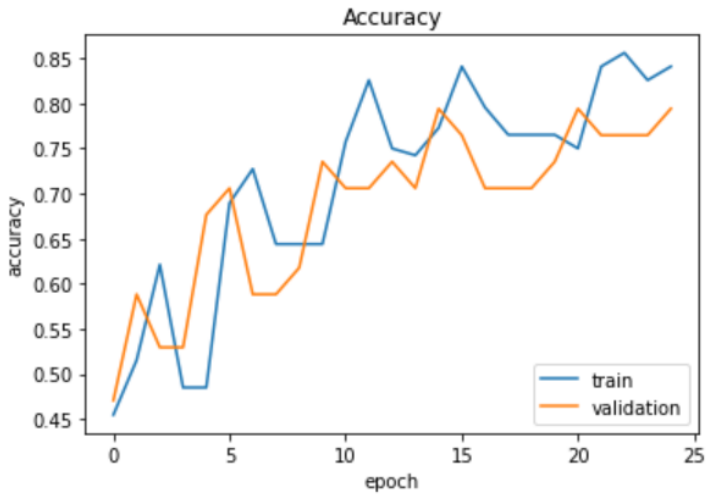
برابر است با تعداد دفعاتی که یک batch داده وارد مدل شده و عمل یادگیری انجام شده است.

مقدار بهینه epoch را می توان از نمودار های loss و accuracy متوجه شد یعنی تا جایی که با افزودن epoch مدل ما روی داده validation عملکرد بهتری داشته باشد(validation loss نباید افزایشی شود) نشانگر این است که باید تعداد epoch زیاد شود. وقتی دیگر loss کمتر نشد یا دقت بهتر نشد یعنی نباید دیگر جلو رویم زیرا overfitting پیش می آید و مدل روی داده های train زیادی fit می شود و generality و عملکرد آن روی داده هایی که قبلا ندیده است مثل تست و validation کاهش می یابد .

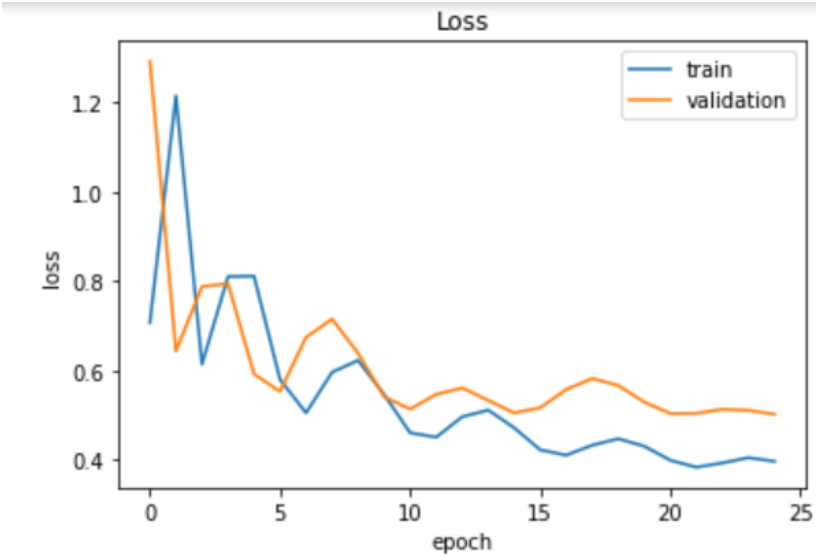
در batch تعداد iteration و epoch یکسان است. در stochastic تعداد iterationها 208 برابر epoch ها است، در حالی که در حالت mini batch با سایز 32 برای batch تقریبا تعداد iterationها 7 برابر epochها است.

ط) در این قسمت activation function های متفاوت را امتحان می کنیم. در قسمت های قبل relu گذاشته بودیم و در این قسمت sigmoid و tanh را هم امتحان می کنیم. relu معمولا برای MLP و CNN استفاده می شود و tanh , sigmoid برای recurrent networks استفاده می شوند. Tanh و sigmoid حساسیت به مساله ی معمول vanishing gradient را افزایش می دهند برای همین relu توصیه شده است.

حال نتایج حاصل از tanh را مشاهده می کنیم:



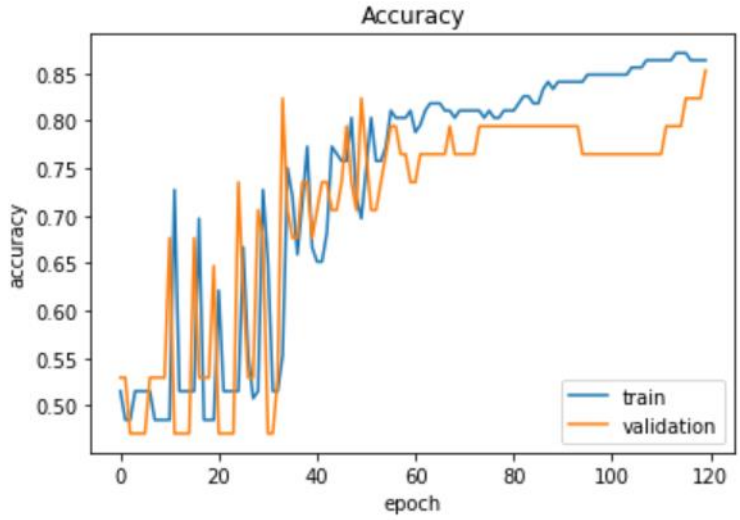
شکل 2-16 - دقت مدل بعد از 25 epoch با tanh به عنوان activation function



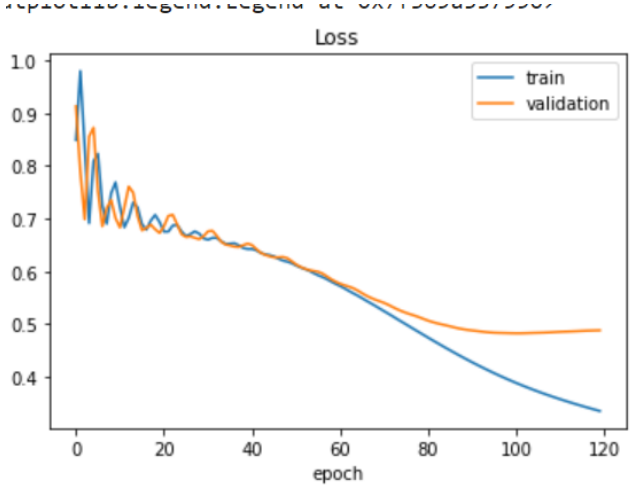
شکل 2-17 - loss مدل بعد از 25 epoch با tanh به عنوان activation function

```
Test Loss 0.5101635456085205
Test Accuracy 0.6904761791229248

confusion matrix=
 [[17 10]
 [ 3 12]]
```

سپس نتایج حاصل از sigmoid را می بینیم:



شکل 2-18 - دقت مدل بعد از 120 epoch با sigmoid به عنوان activation function



شکل 2-19 - loss مدل بعد از 120 epoch با sigmoid به عنوان activation function

```
confusion matrix=
 [[21  6]
 [ 4 11]]
Test Loss 0.5029429197311401
Test Accuracy 0.761904776096344
```

مطابق انتظار عملکرد مدل ضعیف تر شد و حتی در Sigmoid تعداد 120 ایپاک برای آموزش نیاز شد.

علت این اتفاق هم همانطور که بالاتر گفته شد vanishing gradient است یعنی در بعضی ایپاک ها آنقدر

گرادیان کوچک بوده است که تغییر محسوسی در وزن ها نبوده است. همچنین در ابتدای کار نوساناتی در فرایند یادگیری این مدل ها بوده که نشان می دهد تابع فعال ساز انتخابی مناسب نبوده است.

به طور خلاصه از فواید sigmoid میتوان به تغییر نرم آن و مشتق داشتن در همه نقاط به دلیل پیوستگی اشاره کرد. همچنین برای پیاده سازی و گرفتن مشتق ساده بوده و غیر خطی است پس میتواند خروجی غیر خطی بدهد.

- اما مشکلش این است که خیلی آهسته همگرا میشود و مرکز خروجی حول نقطه صفر نیست برای همین هر تغییری در گرادیان تاثیر زیادی داره و بهینه سازی دچار اختلال میشود.(Vanishing Gradient Problem)

از فواید tanh میتوان به وجود مشتق در همه نقاط به دلیل پیوستگی اشاره کرد و با مقادیر تمام مثبت منافاتی ندارد و غیر خطی است پس میتواند خروجی غیر خطی بدهد.اما مشکلش مقدار گرادیان کوچک و مساله (Vanishing Gradient Problem) است.

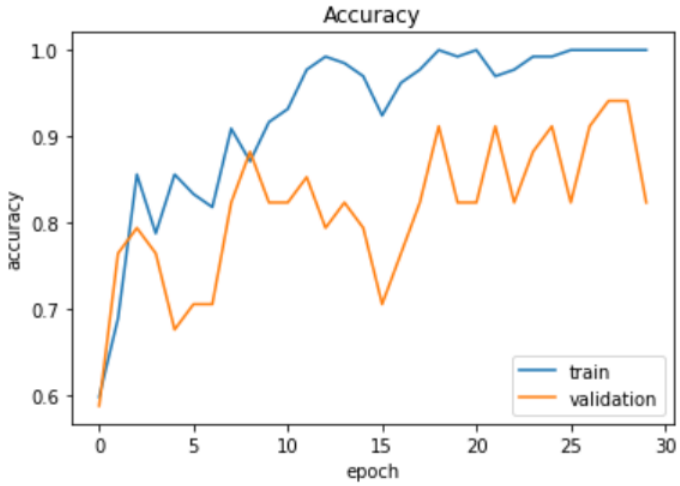
تابع reluغیرخطی است پس می تواند خروجی غیرخطی بدهد و در stochastic gradient خیلی سریعتر می تواند همگرا شود.همچنین همه ی نورون ها را همزمان فعال نمی کند که شبکه را پراکنده، به صرفه و آسان از نظر محاسبات می کند.

اما مشکل اصلی reluاین است که تا بی نهایت می رود و اینکه در صفر مشتق ندارد.همچنین مقدار گرادیان برای ورودی های منفی صفر است یعنی تغییری نمی کنند پس بعضی نورون های همواره غیر فعال می مانند که این مشکل را با تغییر learning rate و bias می توان حل کرد.بعلاوه مرکز خروجی حول صفر نیست و می تواند مشکل زا باشد. گرادیان وزن ها همه مثبت یا منفی اند که به عملکرد زیگزاگ منجر می شود که این مشکل را با batchnorm می توان حل کرد. مشکل دیگر این است که میانگین activation function صفر نیست پس بایاسی از طرف ReLU وارد شبکه می شود. اگرچه از دو تابه قبلی سریعتر همگرا می شود ولی این مقدار بایاس وارده در لایه های بعدی شبکه سرعت را کمتر می کنند.

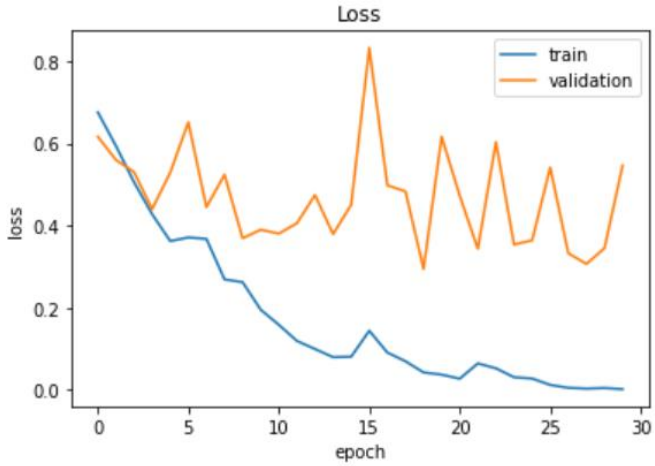
در نهایت برای مراحل بعد ReLU را انتخاب کردیم.

ی) در این مرحله یکبار به جای 2 لایه مخفی 4 لایه و بار دیگر 6 لایه می گذاریم و عملکرد مدل را می سنجیم. ابتدا شبکه با 4 لایه مخفی با مشخصات زیر را بررسی می کنیم:

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_66 (Dense)             (None, 200)               12200
_________________________________________________________________
dense_67 (Dense)             (None, 450)               90450
_________________________________________________________________
dense_68 (Dense)             (None, 400)               180400
_________________________________________________________________
dense_69 (Dense)             (None, 100)               40100
_________________________________________________________________
dense_70 (Dense)             (None, 2)                 202
=================================================================
Total params: 323,352
Trainable params: 323,352
Non-trainable params: 0
```



شکل 2-20 - دقت مدل در 30 ایپاک با 4 لایه مخفی



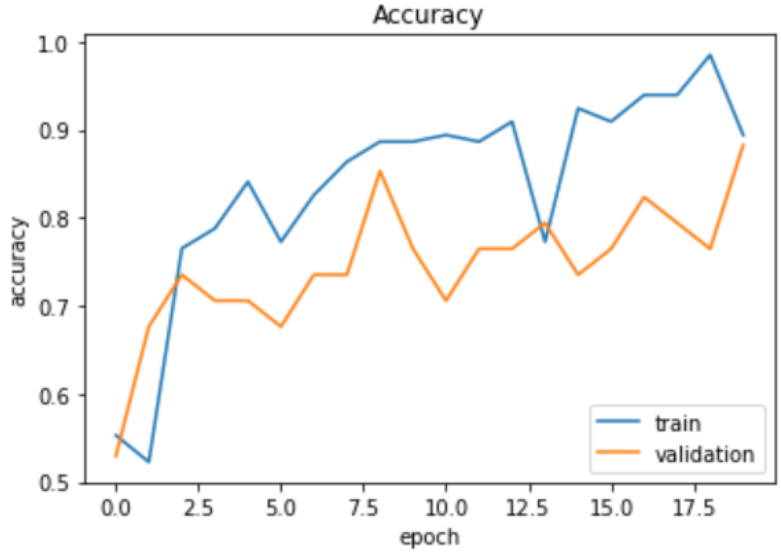
شکل 2-22 - loss مدل در 30 ایپاک با 4 لایه مخفی

```
confusion matrix=
 [[24  3]
 [ 0 15]]
```

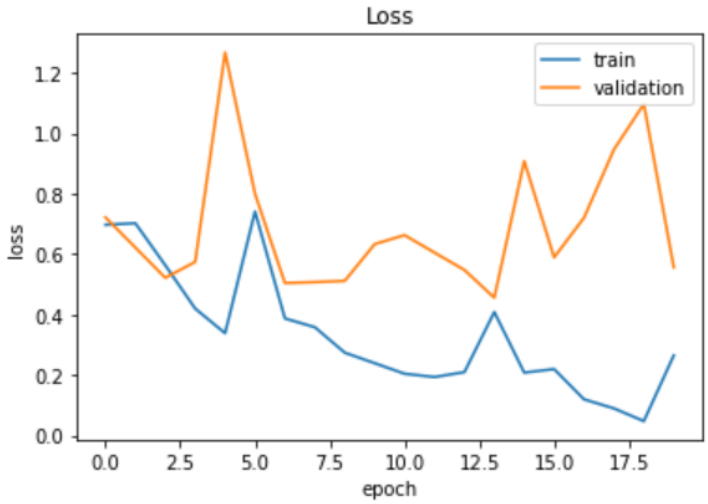
```
Test Loss 0.45417895913124084
Test Accuracy 0.9285714030265808
```

به وضوح می توان دید که دقت مدل تا 92 درصد افزایش یافته و عملکرد آن بهتر شده است البته این اصل همیشه برقرار نیست یعنی تا جایی باید لایه اضافه شود که پیچیدگی بیش از حد به مدل داده نشود و مساله Overfit نشود. حال 6 لایه مخفی را با تعداد نورون های زیر بررسی می کنیم:

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_85 (Dense)             (None, 200)               12200
_________________________________________________________________
dense_86 (Dense)             (None, 500)               100500
_________________________________________________________________
dense_87 (Dense)             (None, 800)               400800
_________________________________________________________________
dense_88 (Dense)             (None, 900)               720900
_________________________________________________________________
dense_89 (Dense)             (None, 300)               270300
_________________________________________________________________
dense_90 (Dense)             (None, 100)               30100
_________________________________________________________________
dense_91 (Dense)             (None, 2)                 202
=================================================================
Total params: 1,535,002
Trainable params: 1,535,002
Non-trainable params: 0
```



شکل 2-23 - دقت مدل در 18 ایپاک با 6 لایه مخفی



شکل 2-24 - loss مدل در 18 ایپاک با 6 لایه مخفی

```
confusion matrix=
 [[20  7]
 [ 0 15]]
```

```
Test Loss 0.7685147523880005
Test Accuracy 0.8333333134651184
```

می توان دید که علی رغم افزوده شدن 2 لایه نسبت به حالت قبل نتیجه روی داده تست بهتر نشده است. ممکن است برای train به دقت بهتری برسیم ولی لزوما روی داده های دیده نشده عملکرد بهبودی نداشته است. پس نتیجه می گیریم که با انتخاب معقول تعداد لایه ها که نه خیلی زیاد باشد و نه خیلی کم و متناسب با پیچیدگی فضای مساله باشد به نتیجه دلخواه میرسیم.

ک) شبکه 4 لایه به شرح زیر:

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_66 (Dense)             (None, 200)               12200
_________________________________________________________________
dense_67 (Dense)             (None, 450)               90450
_________________________________________________________________
dense_68 (Dense)             (None, 400)               180400
_________________________________________________________________
dense_69 (Dense)             (None, 100)               40100
_________________________________________________________________
dense_70 (Dense)             (None, 2)                 202
=================================================================
Total params: 323,352
Trainable params: 323,352
Non-trainable params: 0
```

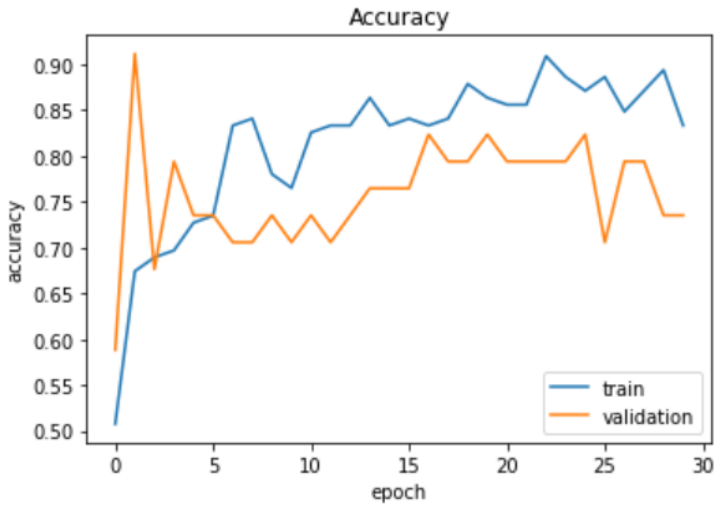
سایز batch برابر 32 است و lossهم cross entropy و تابع فعال ساز ReLU است. می توان با عوض کردن نحوه ی train test split و بهره بردن از روش k-fold، استفاده از CNN ها و تغییر تعداد نورون ها به نتیجه ی بهتر رسید.

ل)

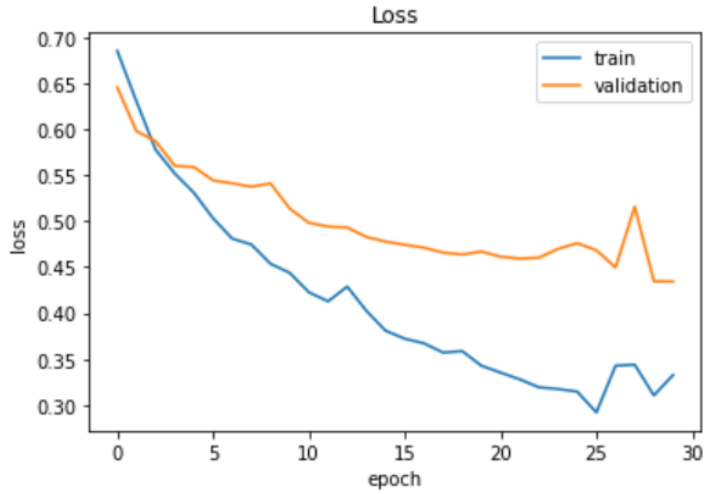
با کاهش تعداد نورون ها و لایه ها مدل زودتر overfit می شود زیرا پیچیدگی مدل کاهش یافته و می توان در epoch کمتر به نقطه بهینه رسید پس در epoch زودتر وارد ناحیه overfitting می شویم.

ابتدا شبکه زیر که لایه کمتری دارد را می سازیم:

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_94 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_95 (Dense)             (None, 2)                 1026
=================================================================
Total params: 32,258
Trainable params: 32,258
Non-trainable params: 0
```



شکل 2-24 - دقت بعد از 30 ایپاک در مدل با یک لایه مخفی

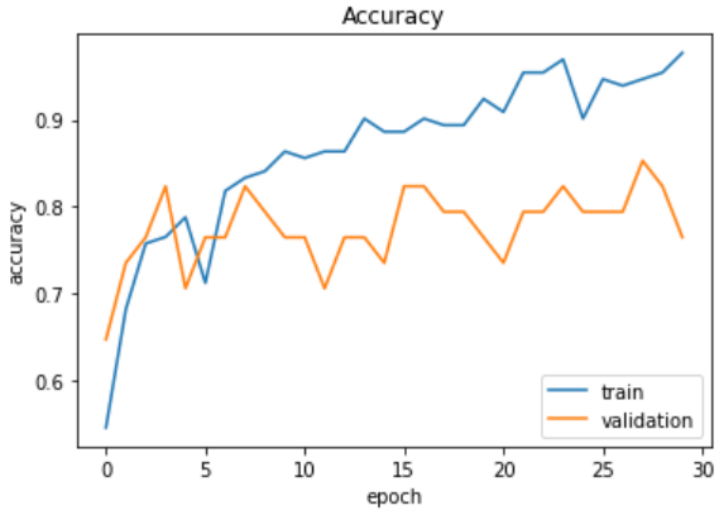


شکل 2-25 - loss بعد از 30 ایپاک در مدل با یک لایه مخفی

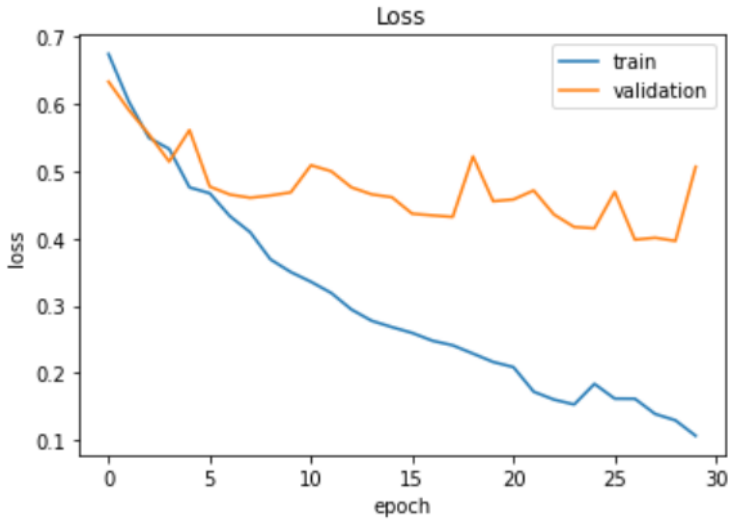
می توان دید که تفاوت زیادی در ایپاک بهینه و ایجاد overfitting نشده و کاهش لایه خیلی اثرگذار نبوده است.

حال تعداد نورون ها در دو لایه مخفی را کم می کنیم و با شبکه زیر به بررسی می پردازیم:

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_96 (Dense)             (None, 200)               12200
_________________________________________________________________
dense_97 (Dense)             (None, 300)               60300
_________________________________________________________________
dense_98 (Dense)             (None, 2)                 602
=================================================================
Total params: 73,102
Trainable params: 73,102
Non-trainable params: 0
```



شکل 2-26 - دقت بعد از 30 ایپاک در مدل با نورون های کمتر در لایه مخفی

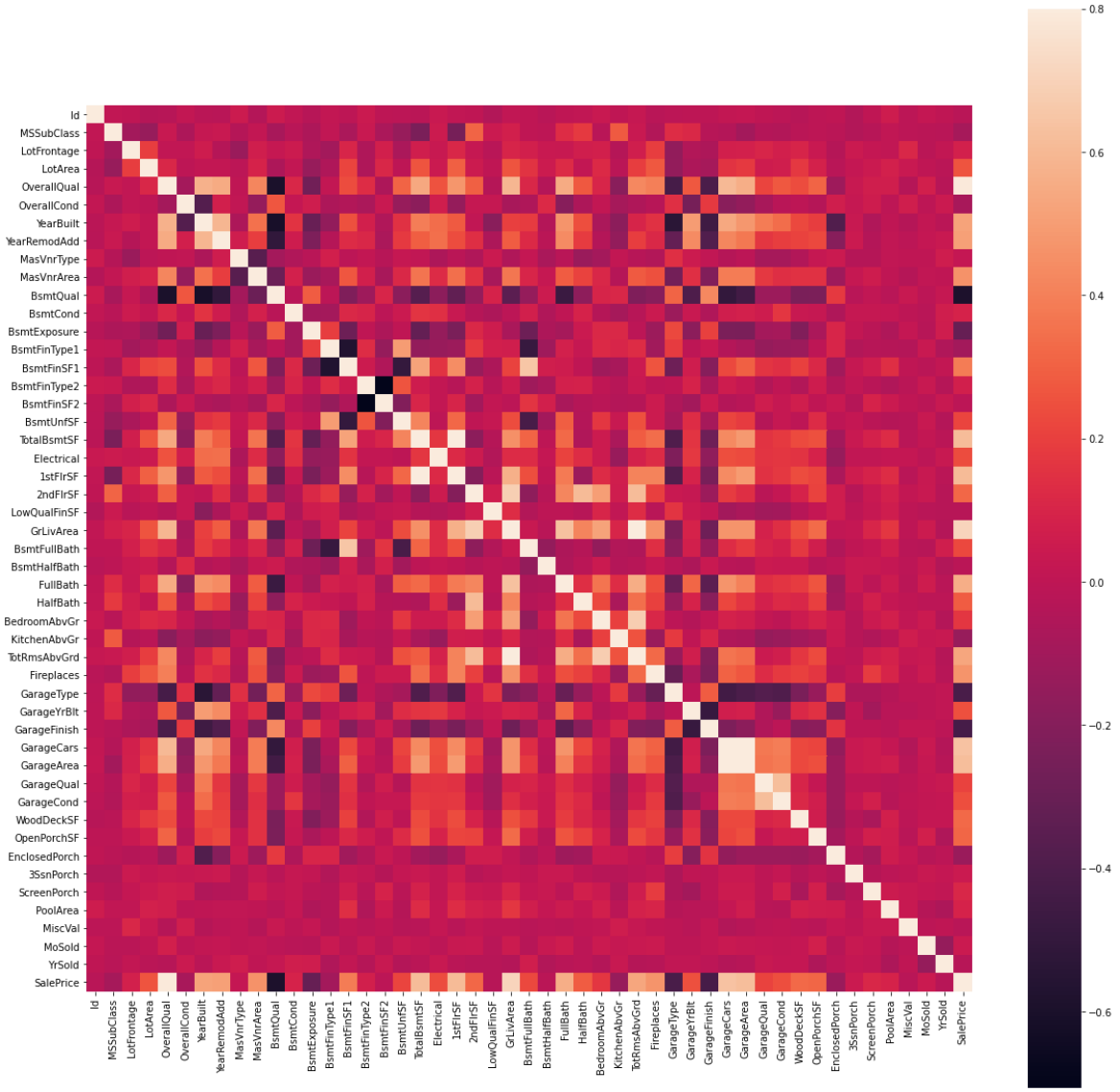


شکل 2-27 - loss بعد از 30 ایپاک در مدل با نورون های کمتر در لایه مخفی

می توان دید که به وضوح در ایپاک کمتری overfitting اتفاق افتاده است و فرضیه ما در این حالت به خوبی قابل مشاهده است.

سوال 3- Dimension Reduction

الف)



شکل 3-1- ماتریس همبستگی

این ماتریس نشان دهنده میزان همبستگی بین هر دو ویژگی متناظر می باشد که اگر دو ویژگی همبستگی بالایی داشته باشند نشان دهنده این است که اطلاعات جدیدی به ما نمیدهند و فقط ابعاد را

زیاد میکنند پس میتوان مقادیری که همبستگی زیادی دارند را با کد زیر پیدا کرد(از آنجایی که حجم ماتریس خیلی بالاست اینکار با چشم امکان پذیر نیست وگرنه برای مثال میتوان دید که همبستگی روی قطر اصلی ماتریس 1 است چون دو ویژگی یکسان هستند و کاملا همبسته. هرچه به سمت رنگ بنفش پررنگ میرویم این میزان کمتر میشود):

```
corr_columns=corrmat.columns
for i in corr_columns:
    for j in corr_columns:
        if i == j:
            continue
        if i == 'SalePrice' or j == 'SalePrice':
            continue
        if (corrmat.loc[i,j] > .5) or (corrmat.loc[i,j] < -.4):
            print(i," Vs ",j,"=",corrmat.loc[i,j])
            print(i," Vs ",'SalePrice',"=",corrmat.loc[i,'SalePrice'])
            print(j," Vs ",'SalePrice',"=",corrmat.loc[j,'SalePrice'])
            print('-'*50)
```

لازم به ذکر است در این محاسبه برای ستون آخر همبستگی را حساب نمیکنیم زیرا به همه ویژگی ها نیاز داریم و همچنین از قطر اصلی صرف نظر شده است.

مشاهده میشود که ویژگی های زیر همبستگی زیادی با همدیگر دارند و قابل حذف میباشند:

{'YearBuilt','YearRemodAdd','TotalBsmtSF','GrLivArea','GarageYrBlt','GarageCars','Garage Area','BsmtUnfSF','BsmtFinSF1','1stFlrSF','HalfBath','BedroomAbvGr','2ndFlrSF','FullBath'}

ب)

می دانیم که فرمول یک رگرسیون خطی به صورت $y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_P X_P$ می باشد.

از آنجایی که ما میخواهیم قیمت خانه را پیش بینی کنیم(Y) این واحد برابر با دلار میباشد و برای آنکه بتوانیم ضرایب بتا_x را با همدیگر مقایسه کنیم تا فیچر های مهمتر را بدست آوریم نیاز داریم که این ضرایب با واحد یکسان تعیین شوند . واضح است در حال حاضر یک واحد مثلا دلار/متراژ و یک واحد دیگر دلار/تعداداتاق ها میباشد. برای اینکه به یک متریک یکسان برای مقایسه برسیم از تکنیک انحراف از معیار استفاده میکنیم و انحراف از معیار هر فیچر را در ضریب متناظر آن ضرب میکنیم .

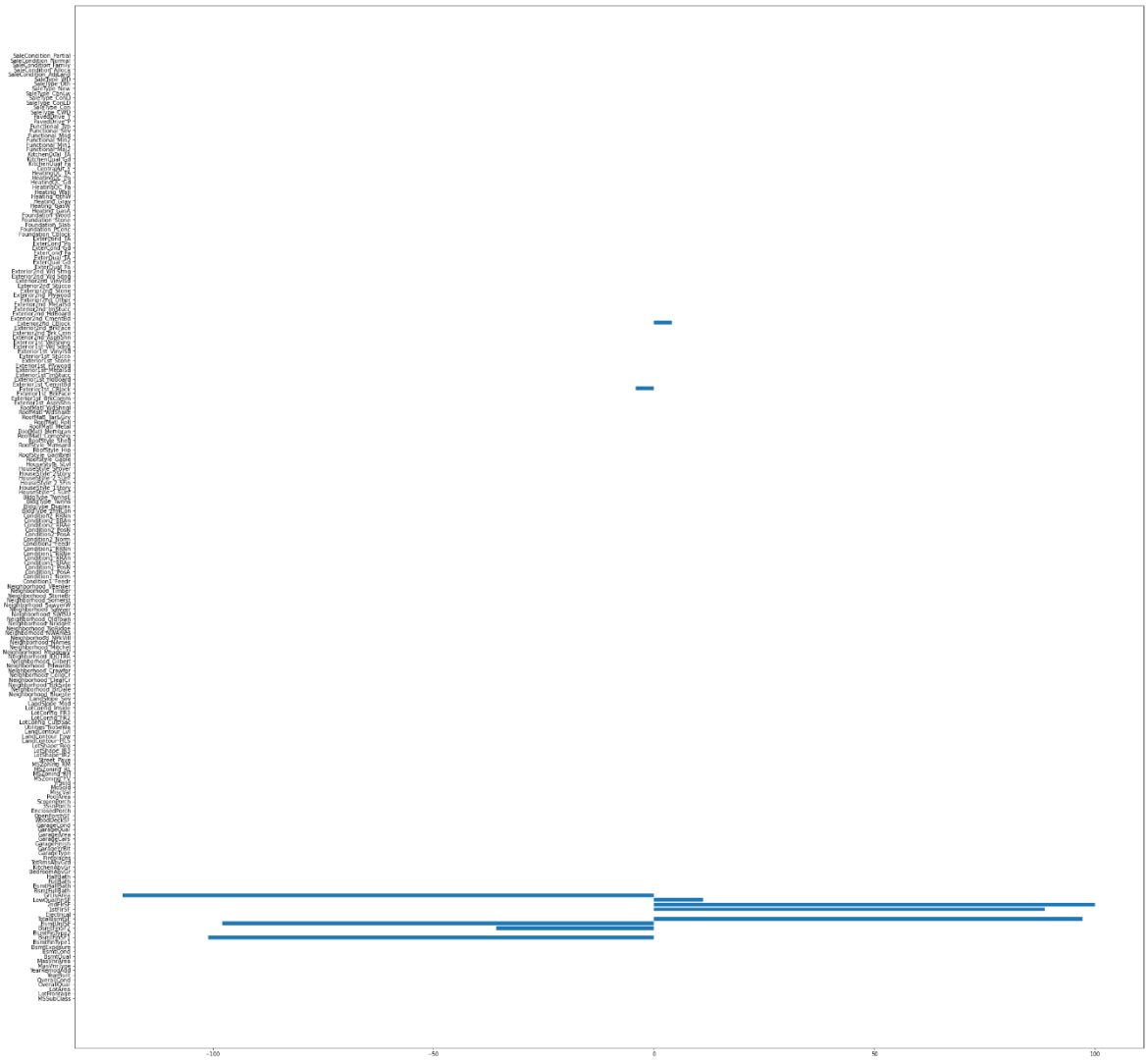
قطعه کد زیر این کار را برای ما انجام میدهد:

```
features = pd.DataFrame(regressor.coef_, xx.columns, columns=['coefficient'])
features.head()
stdevs = []
for i in xx.columns:
    stdev = df2[i].std()
    stdevs.append(stdev)

features["stdev"] = np.array(stdevs).reshape(-1,1)
features["importance"] = features["coefficient"] * features["stdev"]
```

سپس ستون importance را به بازه ی 0 تا 100 مپ میکنیم . قطعه کد زیر این نرمالیزشن را انجام می دهد:

```
features['importance_normalized'] = 100*features['importance'] / features['importance'].max()
```

حال با استفاده از متد LinearRegression و استفاده از Coefficient های آن داریم :

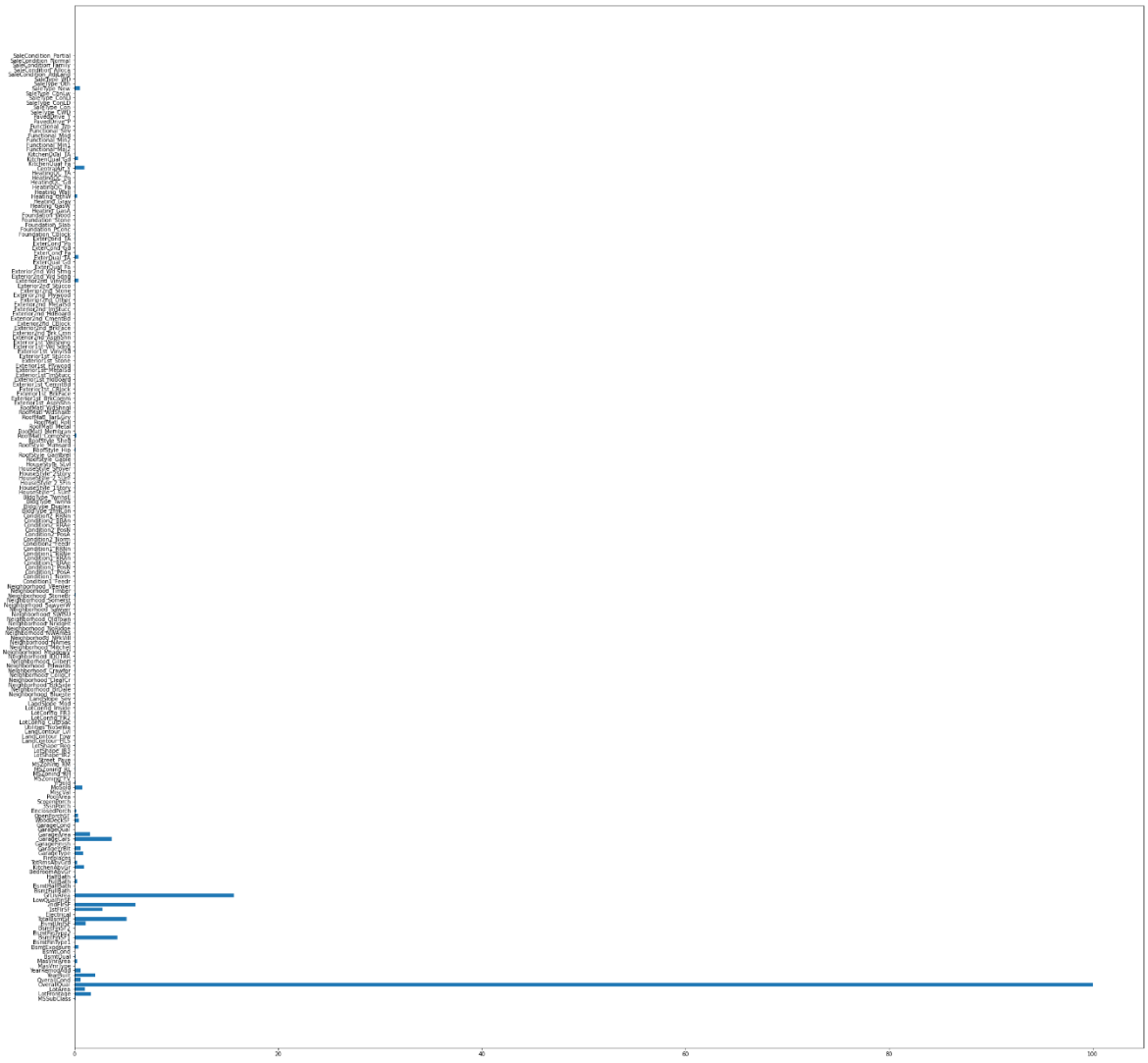


شکل 3-2- مقدار importance ویژگی ها با استفاده از linearRegression

برای استفاده از تابع DecisionTree نیز مراحل مثل حالت قبل است فقط از متد Feature_importance استفاده میکنیم :

```
features = pd.DataFrame(regressor_DT.feature_importances_, xx.columns, columns=['coefficient'])
features.head()
stdevs = []
for i in xx.columns:
    stdev = df2[i].std()
    stdevs.append(stdev)

features["stdev"] = np.array(stdevs).reshape(-1,1)
features["importance"] = features["coefficient"] * features["stdev"]
```

با استفاده از متد DecisionTreeRegressor و استفاده از feature_importanceهای آن داریم :



شکل 3-3- مقدار importance ویژگی ها با استفاده از DecisionTree

ج)

حال با استفاده از backward elimination بهترین مجموعه ویژگی ها را یافته و سپس مدل را روی آن ها اعمال میکنیم:

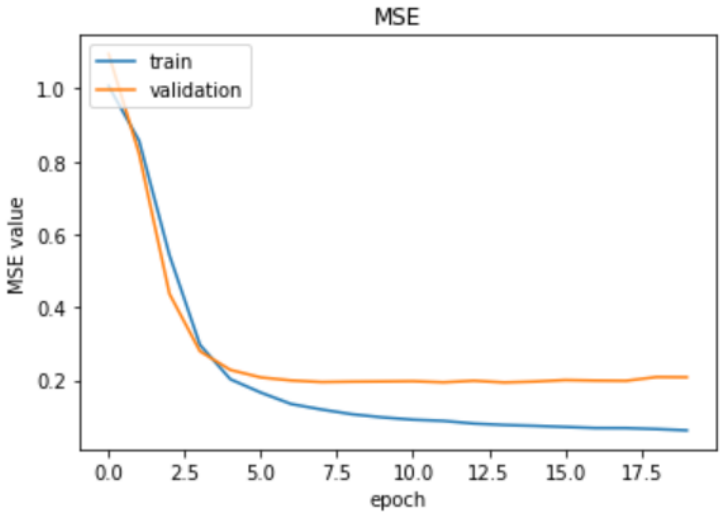
ابتدا یک لول 5 درصدی تعیین کرده و مدل را روی همه فیچر ها فیت میکنیم. فیچر با بالاترین p-value را یافته و اگر این مقدار از لول مرحله قبل بیشتر بود ، آن فیچر را حذف میکنیم وگرنه کار تمام است.

حال مدل را روی فیچر های بدست آمده جدید فیت کرده و این مراحل را تا رسیدن به دقت مطلوب ادامه میدهیم.

در 3 مرحله ابتدا لول را 0.1 گذاشته ، سپس 0.05 و در نهایت با 51 ویژگی بدست آمده ار backward elimination کار را پایان داده و مدل را فیت میکنیم.

مدل 2 لایه با اکتیویشن فانکشن relu میباشد.

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 30)                1560
_________________________________________________________________
dense_25 (Dense)             (None, 10)                310
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 11
=================================================================
Total params: 1,881
Trainable params: 1,881
Non-trainable params: 0
```



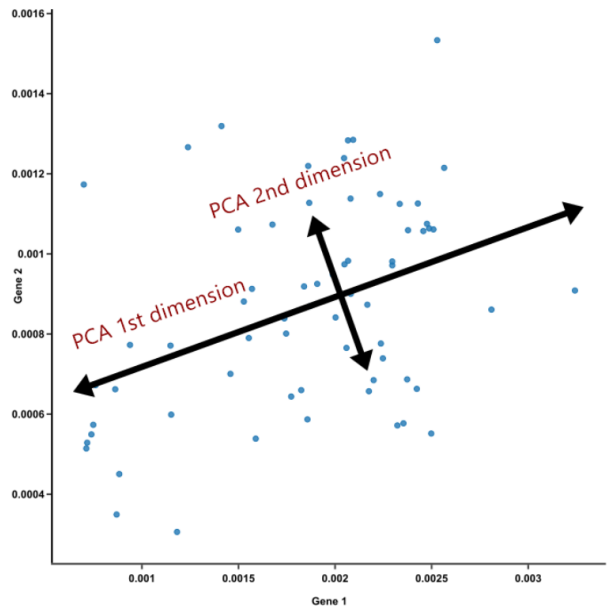
شکل 3-4- مقدار lossدر مدل bacwardElimination

Test Loss 0.4811735153198242

می بینیم که به مقدار lossبرابر با 0.48 رسیدیم که قابل قبول است و اگر بیشتر داده ها حذف کنیم و این مراحل را بیشتر ادامه دهیم به loss صفر هم می رسیم اما متاسفانه فرصت کافی نبود.

د) ابتدا به کمک کلاس PCA که در کتابخانه sklearn از پیش تعریف شده است تعداد component هایی که در آنها بیشترین واریانس و در مجموع 96 درصد واریانس داده ها وجود دارد را پیدا می کنیم. در این مساله 27 تا pc component اولی دارای 95 درصد اطلاعات داده ها هستند و با رفتن به فضای حاصل از آنها اطلاعات زیادی از دست نمی دهیم.

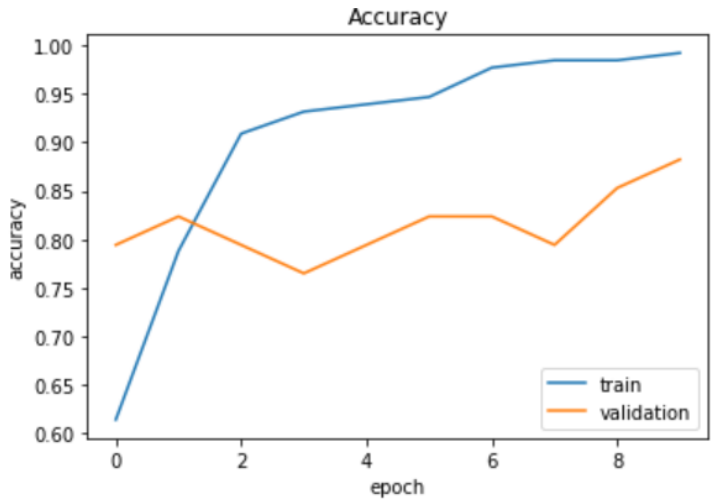
PCA روشی است که سعی می کند ابعاد جدید و ترجیحا کوچکتری را برای داده های ما پیدا کند که اطلاعات زیادی از بین نرود. این کار را با یافتن جهت هایی در فضای مساله می کند که داده بیشترین واریانس را دارا است برای مثال در شکل زیر داده ها در 2 بعد وجود دارند و به کمک PCA دو تا برداری که در جهت آنها داده های ما بیشترین واریانس ر دارا است پیدا می کنیم. اگر در این مثال اولین pc component را انتخاب کنیم و داده ها را به فضای یک بعدی آن ببریم، اطلاعات زیادی از دست نمی دهیم و در فضای ساده تری به محاسبات و یادگیری می پردازیم.



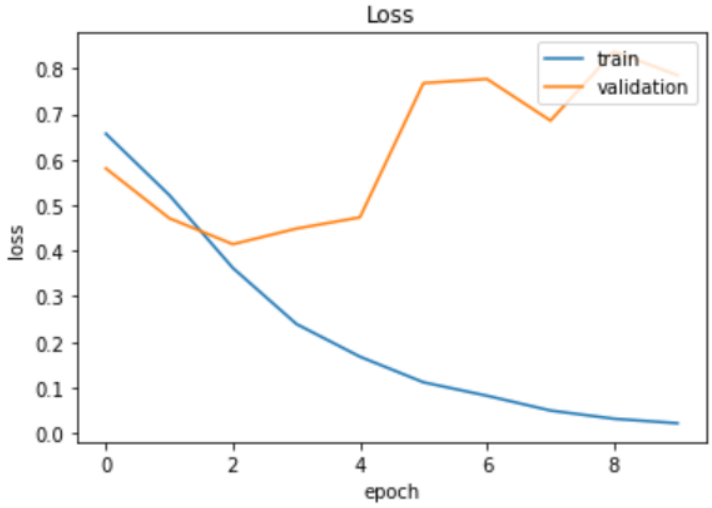
شکل 3-4- عملکرد PCA

پس از تبدیل فضا به فضای 27 بعدی که 95 درصد اطلاعات را در خود جای داده است و تغییر ورودی شبکه به 27 تا به کمک مدل سوال قبل یادگیری را انجام می دهیم.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 200)               5600
_________________________________________________________________
dense_26 (Dense)             (None, 450)               90450
_________________________________________________________________
dense_27 (Dense)             (None, 400)               180400
_________________________________________________________________
dense_28 (Dense)             (None, 100)               40100
_________________________________________________________________
dense_29 (Dense)             (None, 2)                 202
=================================================================
Total params: 316,752
Trainable params: 316,752
Non-trainable params: 0
```



شکل 3-5- خطا در مدل با PCA



شکل 3-6- loss در مدل با PCA

```
Test Loss 0.6178253293037415
Test Accuracy 0.8571428656578064
confusion matrix=
 [[25  2]
 [ 4 11]]
```

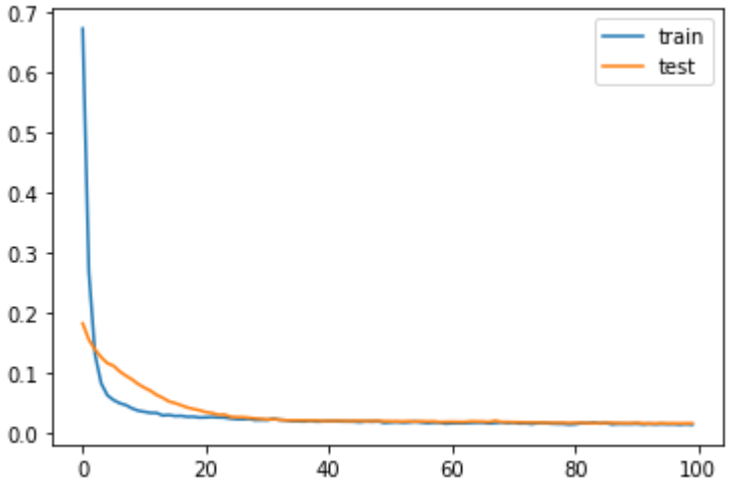
می توان دید که نتیجه ی حاصل تقریبا یکسان است و در زمانی حدودا یک سوم حالت قبل به جواب رسیدیم یعنی با وجود زمانی که به خاطر PCA اضافه شده است باز هم این کار زمان محاسبات و یادگیری را نصف کرده است.

همچنین رسیدن به جواب در epoch خیلی کمتر است یعنی زیر 10 اپیاک که نصف حالت قبل است و علت سریعتر شدن مدل را نشان می دهد.

ه) حال از autoencoder استفاده می کنیم و داده را قبل از ورودی به شبکه ی اصلی از encoder اتوانکودرمان عبور می دهیم.

Autoencoder: به صورت خلاصه اتوانکودر شبکه ای است که حالت ساعت شنی دارد یعنی لایه های وسط آن تعداد نورون کمتری دارد و اول و آخر آن به اندازه ی ابعاد ورودی نورون دارد. این شبکه سعی می کند با این ساختار ابتدا به کمک encoder داده را به فضایی کوچکتر ببرد و سپس با برگرداندن آن به فضای ورودی بررسی کند که آیا این کاهش بعد با از دست رفتن اطلاعات همراه بوده است یا نه و اگر loss آن کوچک باشد صحیح است.

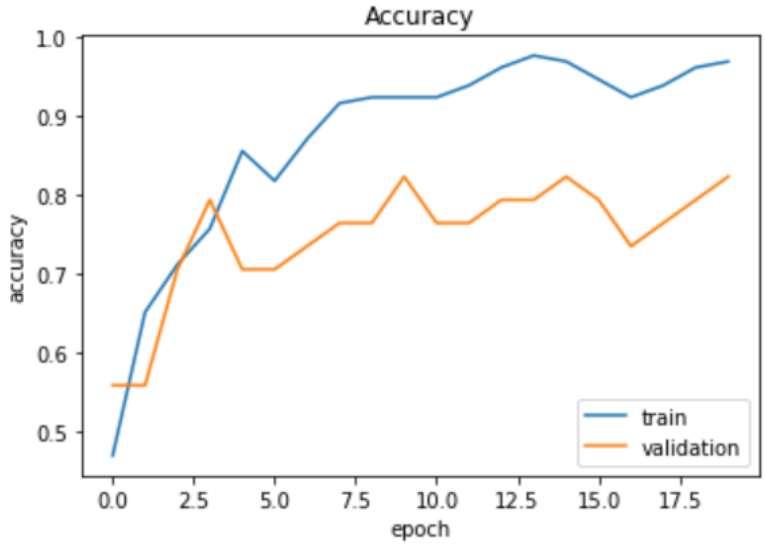
در این مساله یکبار نورون های لایه وسط که اسمش bottleneck است را اول برابر 60 یا همان ورودی می گذاریم و می بینیم که مشکلی ندارد سپس به ترتیب 30، 20، 10 می گذاریم و در هیچ یک اطلاعات از بین نرفته است ولی وقتی برای تعداد نورون 5 ، loss به وجود می آید و نتیجه ی یادگیری مناسب نمیشود پس 10 نورون در لایه وسط معقول است.



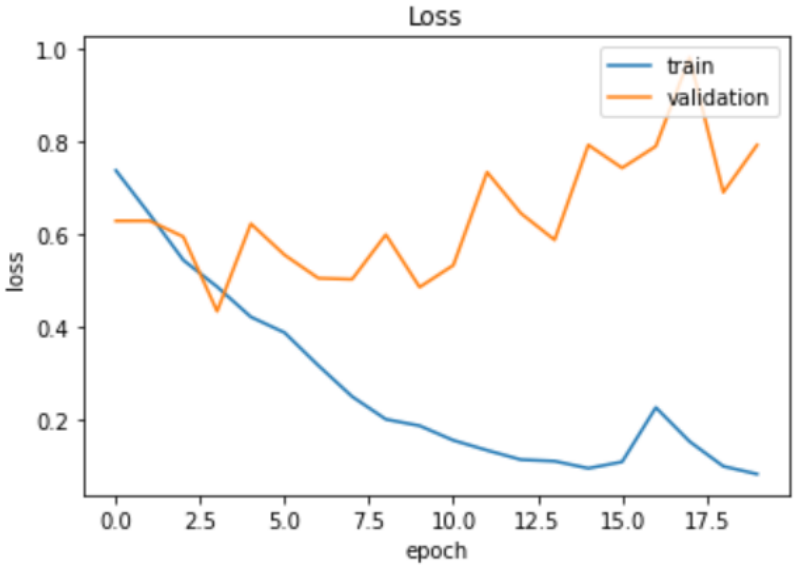
شکل 3-7- loss شبکه autoencoder با 10 نورون در لایه bottleneck

می توان دید که با کاهش بعد تا 10 بعد اطلاعات تقریبا دست نخورده باقی مانده اند و قابل قبول اند. حال نتیجه حاصل از مدل سوال قبل روی این داده ها را می بینیم:

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_215 (Dense)            (None, 200)               2200
_________________________________________________________________
dense_216 (Dense)            (None, 450)               90450
_________________________________________________________________
dense_217 (Dense)            (None, 400)               180400
_________________________________________________________________
dense_218 (Dense)            (None, 100)               40100
_________________________________________________________________
dense_219 (Dense)            (None, 2)                 202
=================================================================
Total params: 313,352
Trainable params: 313,352
Non-trainable params: 0
```



شکل 3-8- دقت مدل با ورودی گذرنده از اتوانکودر



شکل 3-9- loss مدل با ورودی گذرنده از اتوانکودر

```
Test Loss 0.7815537452697754
Test Accuracy 0.8333333134651184
confusion matrix=
 [[23  4]
 [ 3 12]]
```

نتیجه حاصل نشان می دهد که دقت مدل با کاهش بعد مثل قسمت قبل است و زمان یادگیری بخش دوم خیلی کمتر می شود اما زمان کلی افزایش چشمگیری داشته است زیرا قسمت اول نیاز به ایپاک های زیادی برای آموزش دارد.

و)

جدول 1 – مقایسه سه مدل مطرح شده

	دقت داده تست	خطای داده تست	زمان (ثانیه)
بهترین شبکه سوال 2	85.71٪	0.61	2.5
AutoEncoder	83.33٪	0.78	9
PCA	85.71٪	0.61	1.3

می توان دید که از نظر زمانی PCA خیلی بهتر عمل کرده است و AutoEncoder از همه بدتر عمل کرده است. هر سه مدل تقریبا به یک دقت و خطا رسیده اند که نشان می دهد تا جای ممکن به دقت ممکن در شبکه رسیده اند. همچنین اتوانکودر در رسیدن به کوچکترین فضای رودی قابل یادگیری بدون از دست رفتن اطلاعات، موفق تر عمل کرده است. استفاده از PCA در این مساله منطقی است زیرا آنقدر کاهش بعد آن با پیچیدگی همراه نبوده که از اتوانکودر استفاده کنیم و با PCA به زمانی کمتر و فضایی ساده تر برای محاسبات و یادگیری می رسیم. بهتر است از اتوانکودر در مسایلی پیچیده تر مثل image processing استفاده شود.