به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری 2

نام و نام خانوادگی	امید واهب
شماره دانشجویی	۸۱۰۱۹۶۵۸۲
تاریخ ارسال گزارش	۱۴۰۰/۰۲/۰۴

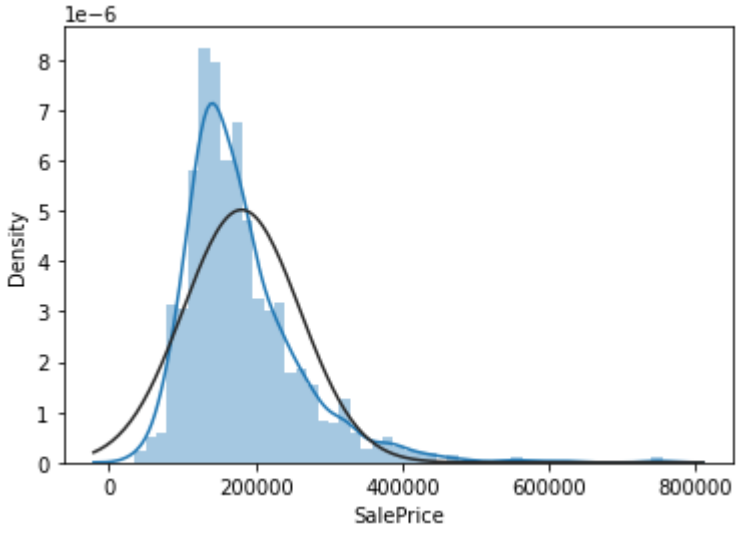
فهرست گزارش سوالات

سوال ۱ – (Regression) MLP

در این سوال قصد داریم توسط MLP یا multi-layer perceptron یک regressor بسازیم که قیمت خانه را تخمین بزند. دیتاست ما house prices است که ۸۰ ستون ویژگی دارد که هر ستون نوع متفاوتی از داده دارد پس نیاز به پیش پردازش زیادی هست. هدف ما تخمین قیمتی نزدیک به ستون آخر هر سطر با توجه به مقادیر دیگر ستون ها است. کد و توضیحات کد این سوال در فایل HW2-Q1.ipynb قرار دارد.

الف) ابتدا دیتاست را می خوانیم و پس از بررسی و observe کردن آن شروع به پیش پردازش می کنیم. تعداد زیادی از ستون ها داده های Nan دارند که اقدام به حذف آنها می کنیم به عبارتی ستون هایی که کمتر از ۶۰ درصد داده دارند را حذف می کنیم که ۵ ستون این ویژگی را دارند و پس از این مرحله ۷۶ ستون خواهیم داشت. سپس به کمک label encoder داده های categorical را numeric می کنیم و یا با One hot encoding ستون های آنها را اضافه می کنیم. پس از انجام همه ی اصلاحات و یکی کردن هر سه فرمت float، int و object به float به کمک اسکیل کردن داده ها را نرمال می کنیم. این کار را به کمک فرمول نرمال سازی انجام می دهیم یعنی میانگین را کم کرده و بر انحراف از معیار تقسیم می کنیم تا در نهایت داده هایی با میانگین صفر و واریانس ۱ داشته باشیم و اندازه ها بزرگتر در بعضی ستونها در یادگیری ما اثر نگذارند. همچنین ستون Id را هم حذف می کنیم زیرا اطلاعاتی ندارد.

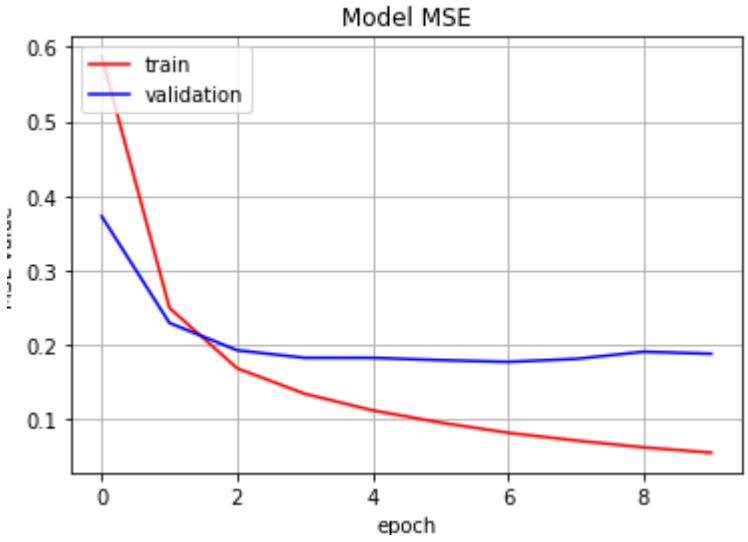
نمودار توزیع قیمت ها را هم visualize می کنیم که با داده بیشتر آشنا شویم:



شکل ۱ – نمودار توزیع قیمت های دیتاست

ب) حال داده را به train و test تقسیم می کنیم با نسبت ۸۰ و ۲۰ سپس مدل را تعریف کرده و به کمک آن یادگیری را انجام می دهیم. برای تعیین هایپرپارامترها دو activation function معمول یعنی ReLU و Softmax را امتحان کرده و مدل را یکبار با یک لایه و بار دیگر با دو لایه می سازیم. انتظار داریم ReLU با دو لایه بهترین عملکرد را داشته باشد. لازم به ذکر است که در تمامی حالت ها پس ابتدا تعداد epoch زیادی یادگیری را انجام می دهیم و سپس از روی نمودار loss ایپاکی که در آن دیگر loss داده ارزیابی یا validation بهبود نیافته است را به عنوان epoch بهینه در نظر می گیریم و بار دیگر مدل را با آن تعداد ایپاک آموزش می دهیم زیرا یادگیری بیشتر منجر به Overfitting می شود. همچنین سعی شده تعداد نورون های هر لایه منطقی و با شیب درست کم یا زیاد شوند. ابتدا تک لایه با ReLU را بررسی می کنیم:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 50)                10150
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 51
=================================================================
Total params: 10,201
Trainable params: 10,201
Non-trainable params: 0
```

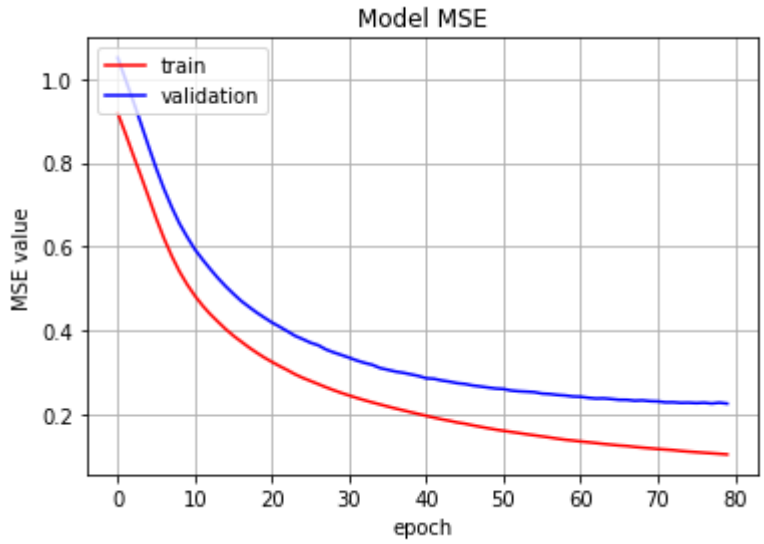


شکل ۲- loss مدل بر تک لایه باReLU بعد از ۱۰ ایپاک

```
Test Loss 0.1700265109539032
```

سپس تک لایه با Softmax را می بینیم:

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 50)                10150
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 51
=================================================================
Total params: 10,201
Trainable params: 10,201
Non-trainable params: 0
```

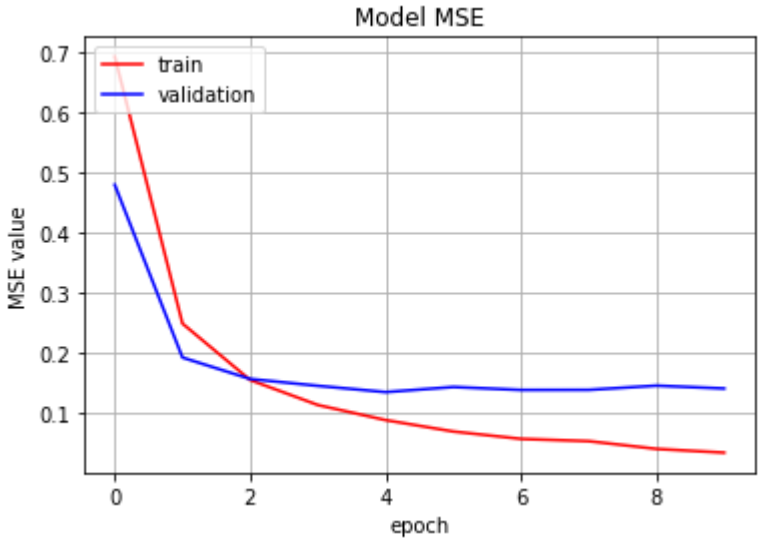


شکل ۳ – loss مدل تک لایه با Softmax بعد از ۸۰ ایپاک

```
Test Loss 0.272044837474823
```

حال مدل دو لایه با ReLU را که حدس می زنیم بهترین جواب را داشته باشد را بررسی می کنیم:

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_60 (Dense)             (None, 90)                18270
_________________________________________________________________
dense_61 (Dense)             (None, 15)                1365
_________________________________________________________________
dense_62 (Dense)             (None, 1)                 16
=================================================================
Total params: 19,651
Trainable params: 19,651
Non-trainable params: 0
```

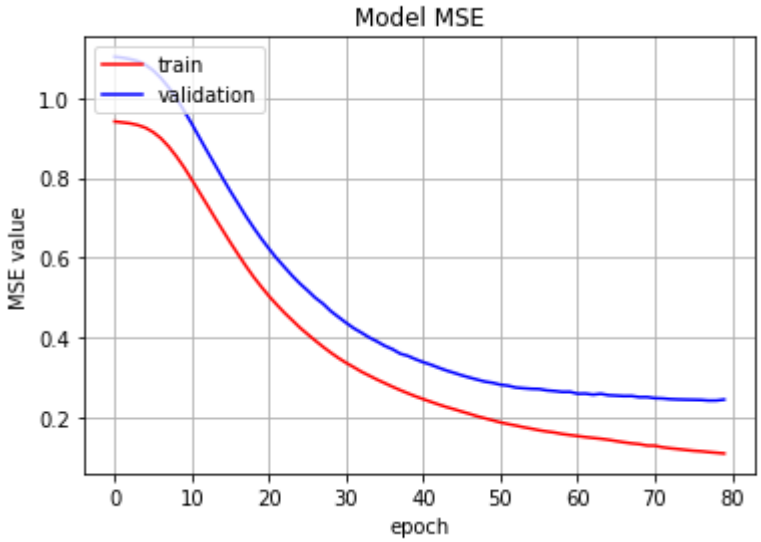


شکل ۴ – loss مدل دو لایه با ReLU بعد از ۱۰ ایپاک

```
Test Loss 0.16810338199138644
```

در نهایت مدل دو لایه با Softmax را بررسی می کنیم:

```
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_63 (Dense)             (None, 90)                18270
_________________________________________________________________
dense_64 (Dense)             (None, 15)                1365
_________________________________________________________________
dense_65 (Dense)             (None, 1)                 16
=================================================================
Total params: 19,651
Trainable params: 19,651
Non-trainable params: 0
```

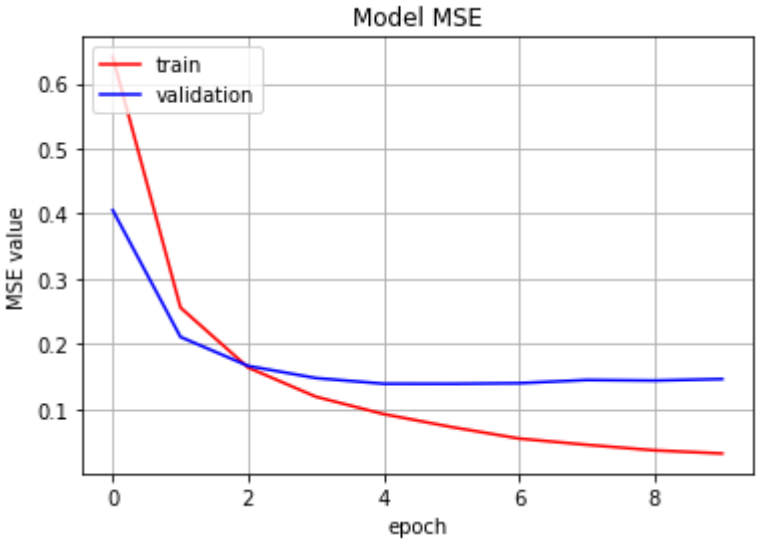


شکل ۵ – loss مدل دو لایه با Softmax بعد از ۸۰ ایپاک

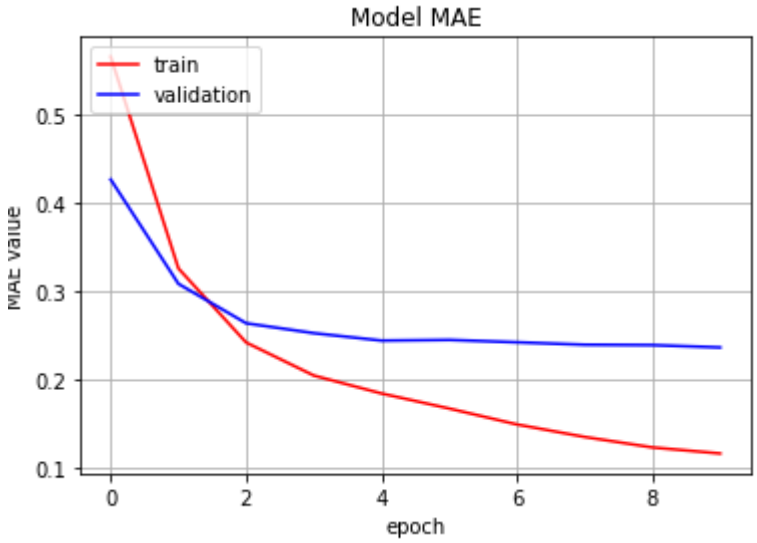
```
Test Loss 0.2658073902130127
```

واضح است که مدل هایی که از Softmax استفاده کرده اند ایپاک بیشتری طول کشیده اند و در نهایت عملکردی ضعیف تر داشته اند زیرا برای شبکه MLP تابع فعال ساز ReLU بهتر عمل می کند و دیگر توابع فعال از برای شبکه های recurrent مناسب تر می باشند و تحلیل کامل این موضوع در سوال ۲ انجام شده است. همچنین مدل های دو لایه طبق انتظار بهتر بوده اند زیرا پیچیدگی بیشتری دارند می توانند مساله فعلی که از پیچیدی نسبتا بالاتری به نسبت یک مساله خطی برخوردار است را بهتر مدل کنند پس مدل دو لایه با ReLU بهترین مدل است.

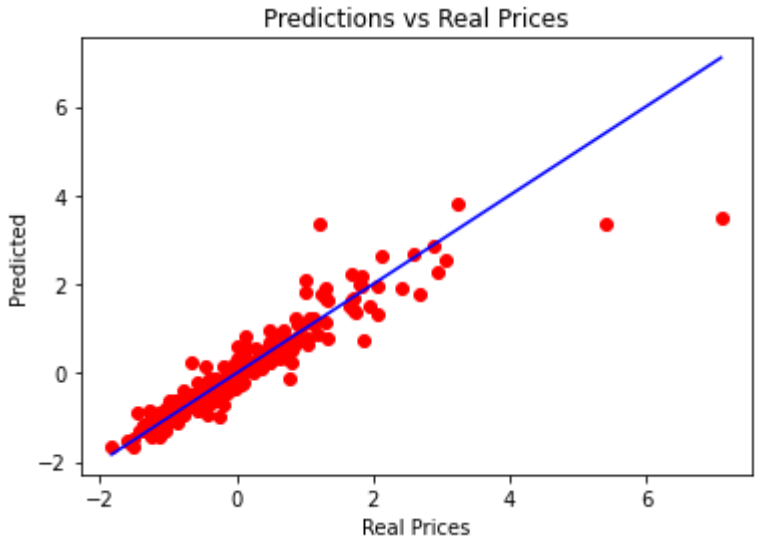
ج) ابتدا loss را MSE می گیریم:



شکل ۶ – نمودار MSE با مدل بر حسب MSE loss



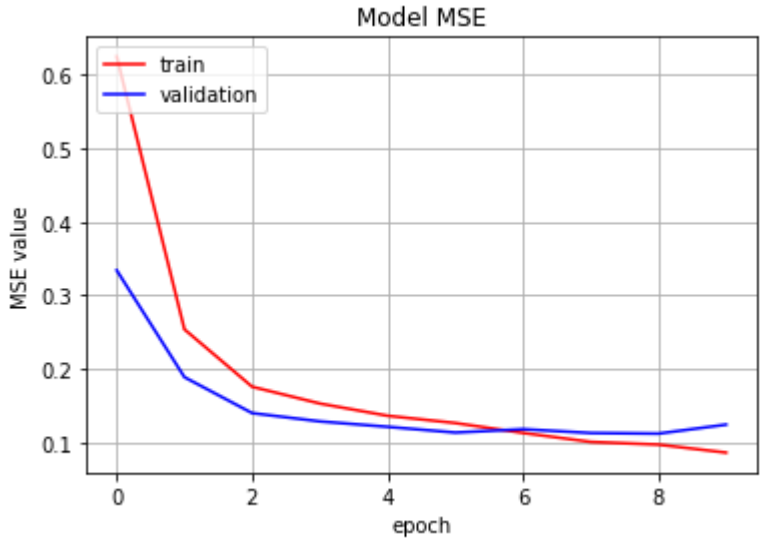
شکل ۷ – نمودار MAE با مدل بر حسب MSE loss



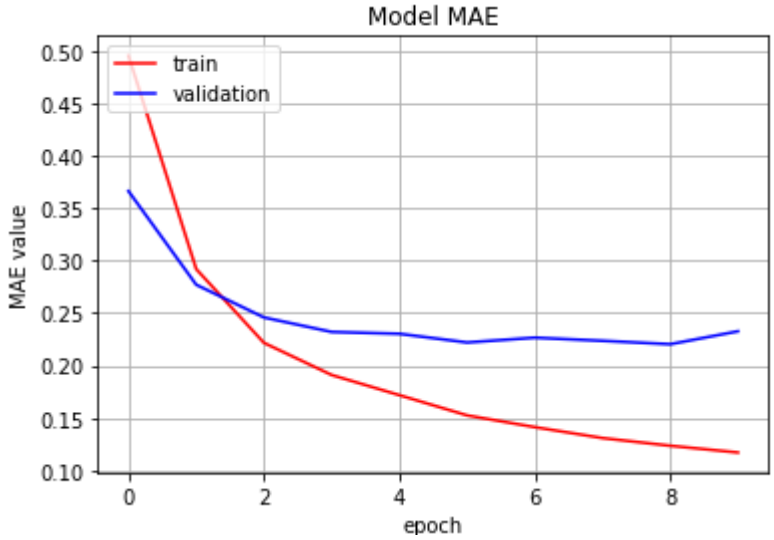
شکل ۸ - نمودار مقادیر پیشبینی شده بر حسب مقادیر واقعی در مدل MSE loss

واضح است که عملکرد خیلی خوبی داشته ایم زیرا هر چه خط آبی رنگ به خط با شیب ۱ و گذرنده از مبدا نزدیک تر باشد یا به عبارتی نقاط قرمز رنگ نزدیک قطر اصلی محور باشند، مدل بهتری داشته ایم که اینجا با تقریب خوبی هر دو برقرارند. همچنین ۱۰ ایپاک مقدار بهینه بود.

د) حال MAE را معیار loss می گیریم:



شکل ۹ – نمودار MSE با مدل بر حسب MAE loss



شکل ۱۰ – نمودار MAE با مدل بر حسب MAE loss

شکل ۱۱ - نمودار مقادیر پیشبینی شده بر حسب مقادیر واقعی در مدل MAE loss

واضح است که عملکرد خیلی خوبی داشته ایم زیرا هر چه خط آبی رنگ به خط با شیب ۱ و گذرنده از مبدا نزدیک تر باشد یا به عبارتی نقاط قرمز رنگ نزدیک قطر اصلی محور باشند، مدل بهتری داشته ایم که اینجا با تقریب خوبی هر دو برقرارند. همچنین ۱۰ ایپاک مقدار بهینه بود.

ه) خطای میانگین مربعات یا MSE روشی برای برآورد میزان خطاست که در واقع تفاوت بین مقادیر تخمینی و آنچه تخمین زده شده است و با فرمول زیر محاسبه می شود:

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

خطای میانگین مطلق یا MAE نیز روشی برای برآورد میزان خطاست که در واقع تفاوت بین مقادیر تخمینی و آنچه تخمین زده شده است و با فرمول زیر محاسبه می شود:

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

تفاوت این دو روش در این است که MSE توان دوم ترم های MAE است. نتایج هر دو مدل با معیار MAE و MSE برای خطا تقریبا یکسان است و حتی نمودار ها خواسته شده نیز برابرند اگر چه MAE و MSE در یک مدل اندکی متفاوت اند و مقدار MAE اندکی بیشتر است و منطقا نیز درست است زیرا MSE توان دوم را نی گیرد و چون مقادیر ما کوچکتر از یک می باشند توان دوم آنها از توان اول کوچکتر است و در نتیجه MSE مقداری کوچکتر از MAE خواهد داشت.

MSE معمول ترین و متداول ترین معیار مورد استفاده است. MAE در داده هایی که outlier های زیادی دارند خوب عمل می کند که این مشکل را با نرمال سازی می توان برطرف کرد ولی MSE راحت تر به جواب می رسد. مشکل دیگر MAE گرادیان بزرگ آن است

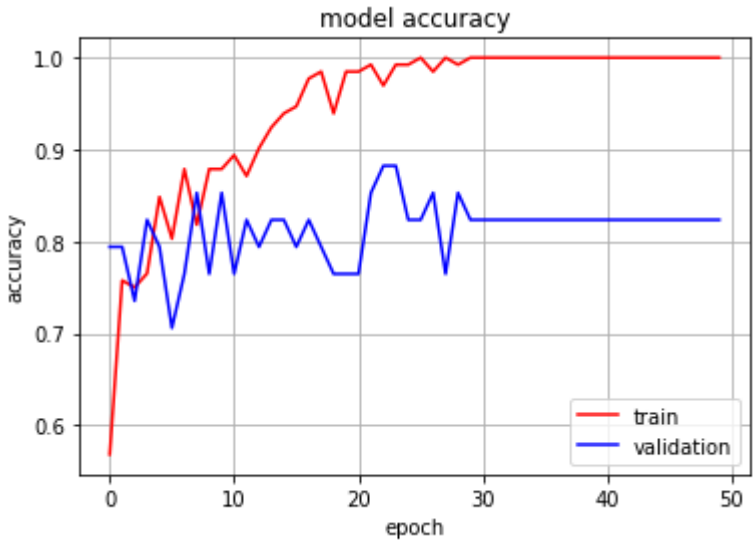
سوال ۲ – (Classification) MLP

در این سوال قصد توسط MLP یا multi-layer perceptron داده هایی را classify کنیم. دیتاست ما sonar است که ۶۰ ستون ویژگی دارد که هر یک مقادیر float دارند و از آنجایی که مقدار Nan ندارند نیازی به پر کردن دیتاست نیست. هدف ما جداسازی و دادن لیبل R یا M به هر سطر است. کد و توضیحات کد این سوال در فایل HW2-Q2.ipynb قرار دارد.

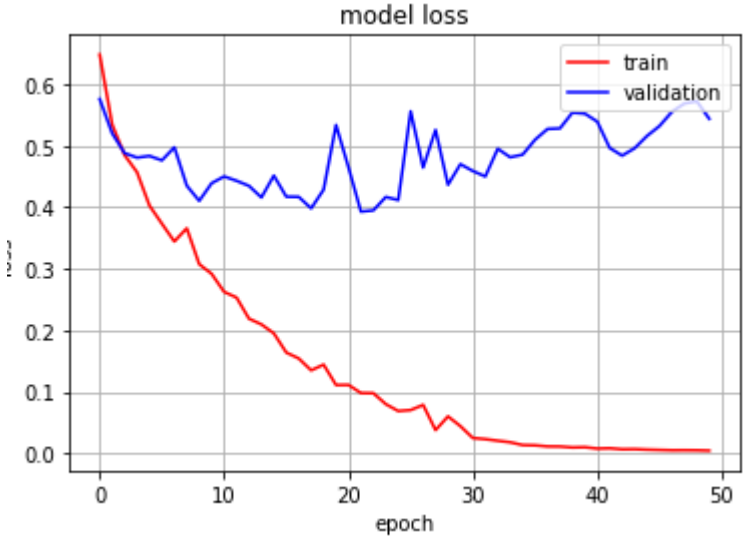
الف) ابتدا داده را در drive ذخیره می کنیم سپس با mount کردن drive آن را به عنوان فایل csv می خوانیم. سپس با دستورات head و info و describe داده ها را بررسی می کنیم تا اگر مشکلی وجود دارد بر طرف کنیم. خوشبختانه داده Nan نداریم پس نیازی به برطرف کردن این مشکل نیست. سپس توسط یک Min Max Scaler داده ها را به بازه ی ۰ تا ۱ 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 ۸۰ درصد از داده ها را train و باقی را test می گیریم سپس از بین دادگان یادگیری ۲۰ 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 مدل پیچیده و بزرگی نمی سازیم و با ۲ لایه مخفی که هر یک ۵۱۲ نورون با activation function معمول یعنی relu و ۲ 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) میتوان دقت و loss مدل را برای ۵۰ epoch را در شکل های ۱ و ۲ دید:

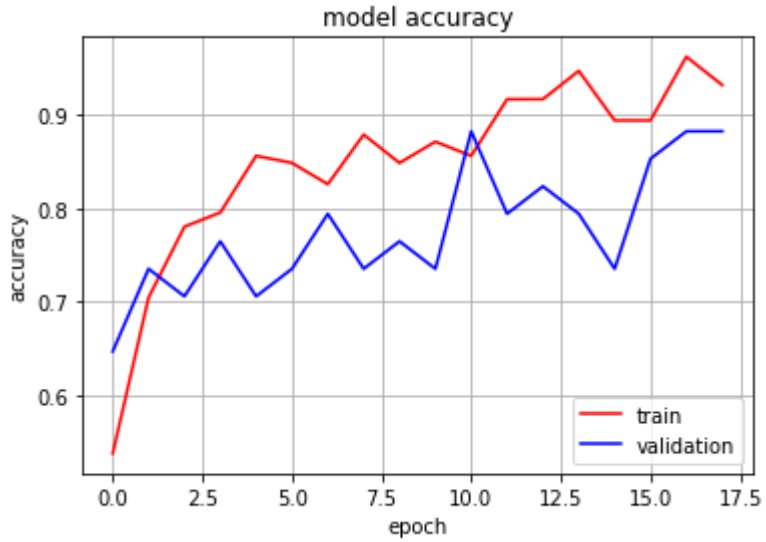


شکل ۱ – دقت مدل در ۵۰ ایپاک

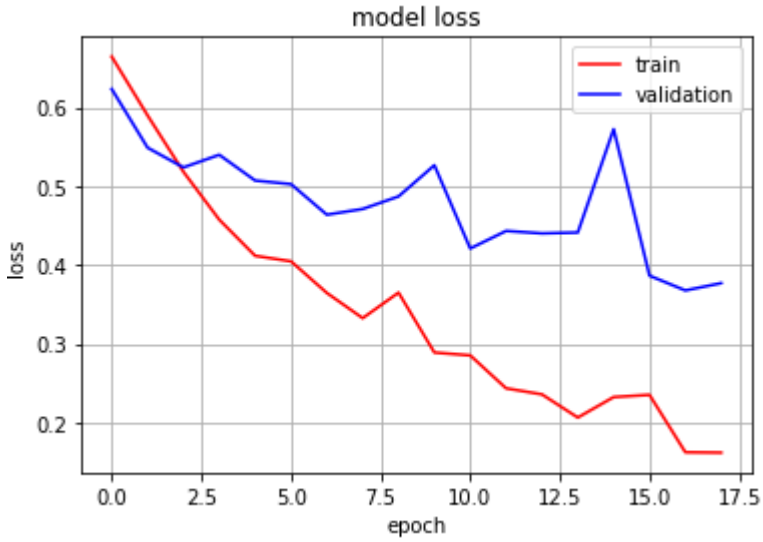


شکل ۲ – loss مدل در ۵۰ ایپاک

واضح است که مدل overfit شده زیرا loss برای داده validation از 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 ۵۰ ایپاک به صفر رسیده است. پس با ۱۸ ایپاک عمل یادگیری را به پایان میرسانیم.



شکل ۳ – دقت مدل در ۱۸ ایپاک

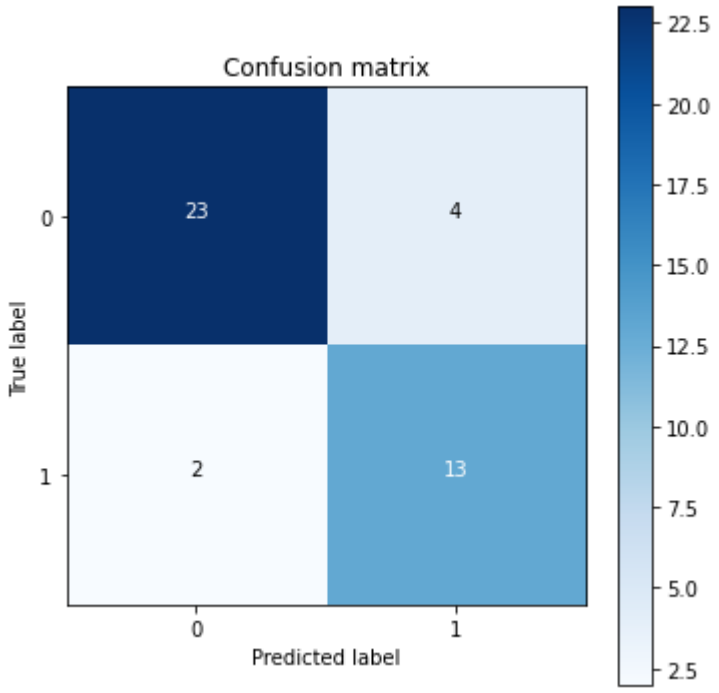


شکل ۴ – loss مدل در ۱۸ ایپاک

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 0.32490259408950806
Test Accuracy 0.8809523582458496

confusion matrix=
 [[24  3]
 [ 2 13]]
```



شکل ۵ – ماتریس confusion

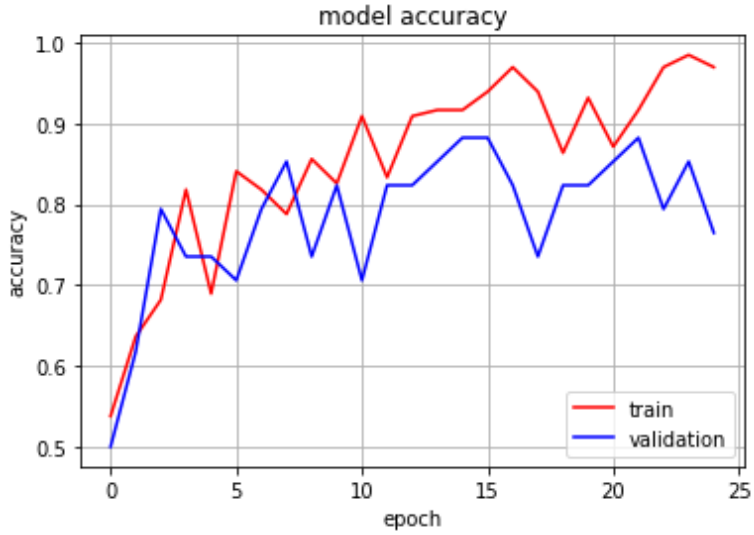
میتوان دید که به دقت خیلی خوبی رسیده ایم و loss و confusion matrix هم این نکته را تایید می کنند.

د) معیار استفاده شده categorical_crossentropy بود که طبق تدریس و جستجو در اینترنت پیشنهاد شده بود. معیار دیگر MSE بود که معمولا در مسایل regression توصیه می شود ولی برای binary 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:

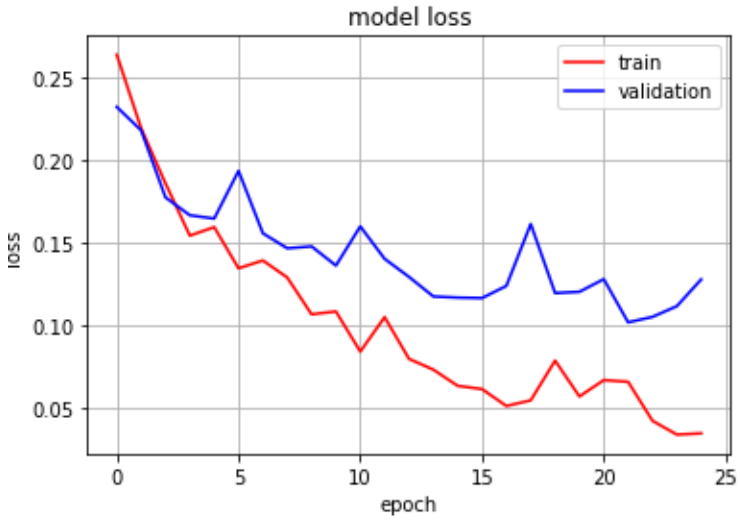
$$H(p,q) = -\sum_{x} p(x)\ \log q(x).$$

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 ۰ و ۱ مثلا ۰.۱ و زمانی که لیبل درست ۰ باشد loss کم خواهد بود ولی وقتی لیبل ۱ باشد مقدار ۰.۱ حدس خوبی نبوده و نتیجه آن loss بزرگ خواهد بود. در این مساله MSE مناسب نیست چون خطایی که محاسبه می شود خیلی درست نیست و باید معیاری انتخاب کنیم که مدل ما را به سمت انتخاب درست تر بین دو مقدار ۰ یا ۱ کند نه اینکه مثل مساله regression سعی در کم کردن فاصله کند.

ه) این بار با MSE آموزش را انجام می دهیم و نتیجه بعد از ۲۵ ایپاک که مقدار بهینه بود، به شرح زیر است:



شکل ۶ – دقت بعد از ۲۵ ایپاک با خطای MSE



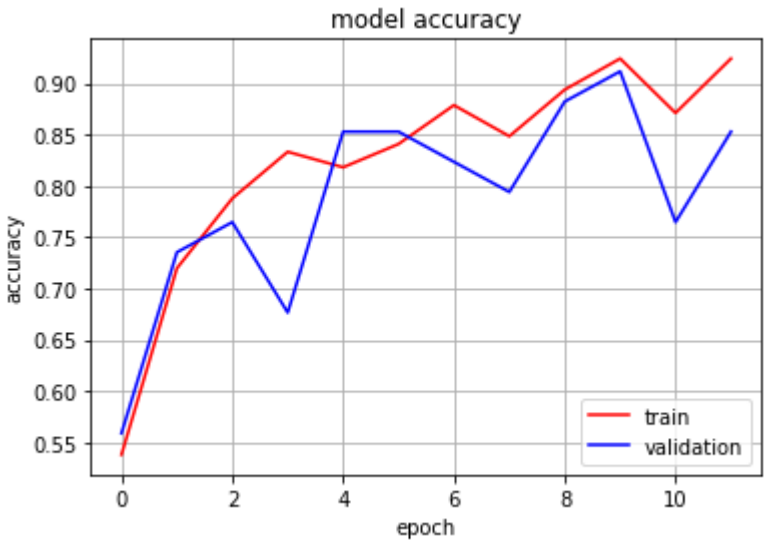
شکل ۷ – MSE loss بعد از ۲۵ ایپاک

چون معیار خطاها یکسان است در دو مدل نمی توان نمودارشان را از لحاظ عددی خیلی 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:

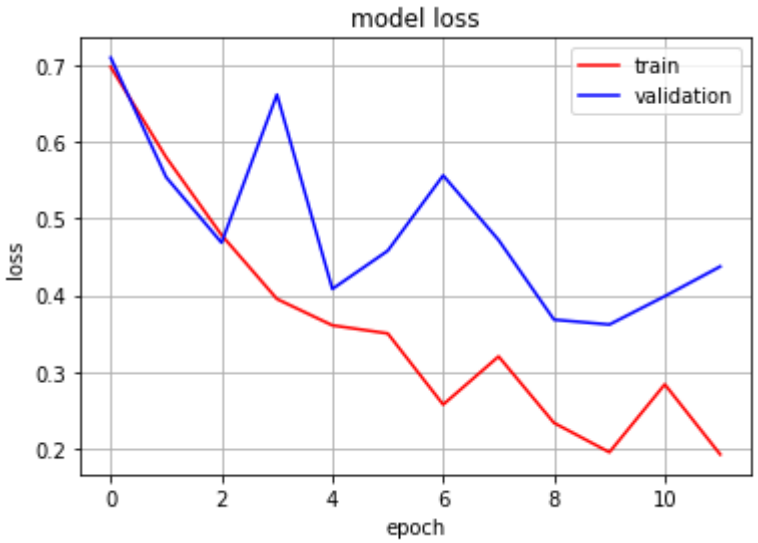
```
Test Loss 0.1225123405456543
Test Accuracy 0.8333333134651184

confusion matrix=
 [[22  5]
 [ 2 13]]
```

میتوان به وضوح دید که عملکرد بدتر شده است. نکته ی قابل توجه دیگر رندوم بودن بیشتر این مدل است جوری که اگر چند بار کد اجرا میشد نتایج نسبتا متفاوتی دیده میشد که اینجا بهترین آنها گذاشته 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 ۱ می گذاریم:



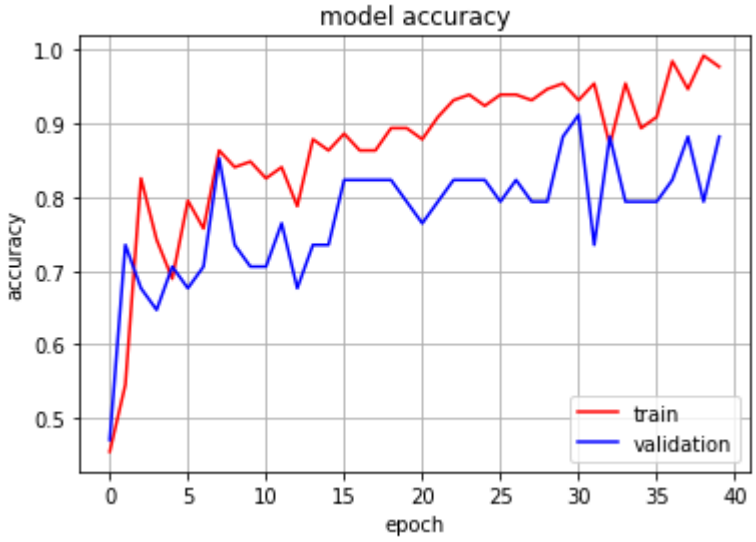
شکل ۸ – دقت بعد از ۱۱ ایپاک در stochastic



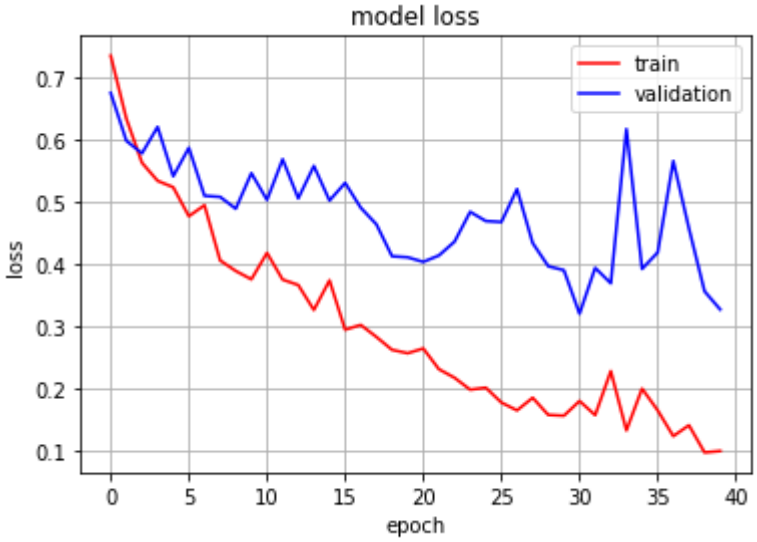
شکل ۹ – loss بعد از ۱۱ ایپاک در stochastic

```
Test Loss 0.5335174798965454
Test Accuracy 0.8333333134651184

confusion matrix=
 [[21  6]
 [ 1 14]]
```

برای mini batch با سایز ۳۲ هم در مراحل قبل نتیجه را دیدیم پس اینبار برای batch size را ۶۴ گذاشته و امتحان می کنیم:



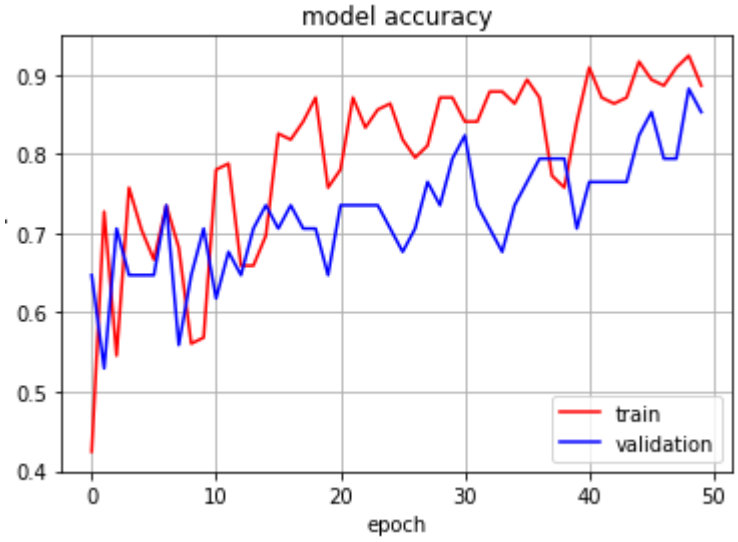
شکل ۱۰ – دقت بعد از ۴۰ ایپاک در با batchهای ۶۴ تایی



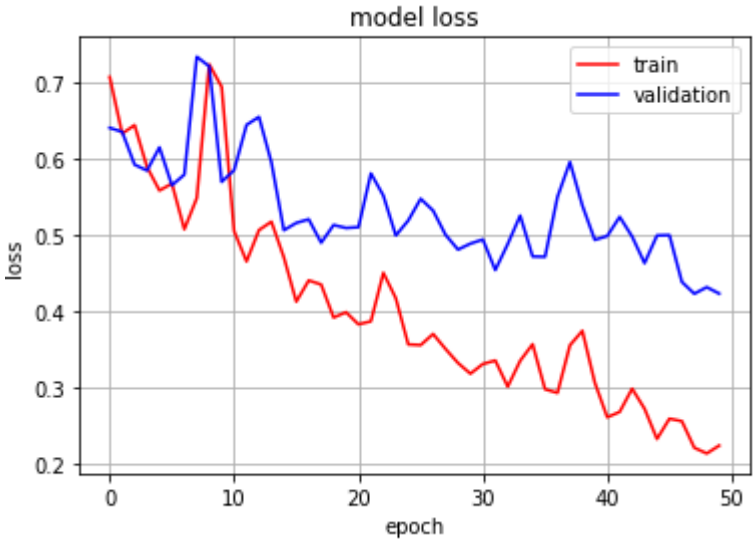
شکل ۱۱ – loss بعد از ۴۰ ایپاک در با batchهای ۶۴ تایی

```
Test Loss 0.3545846939086914
Test Accuracy 0.9047619104385376

confusion matrix=
 [[24  3]
 [ 1 14]]
```

سپس برای سایز ۱۲۸ برای batch ها بررسی می کنیم:



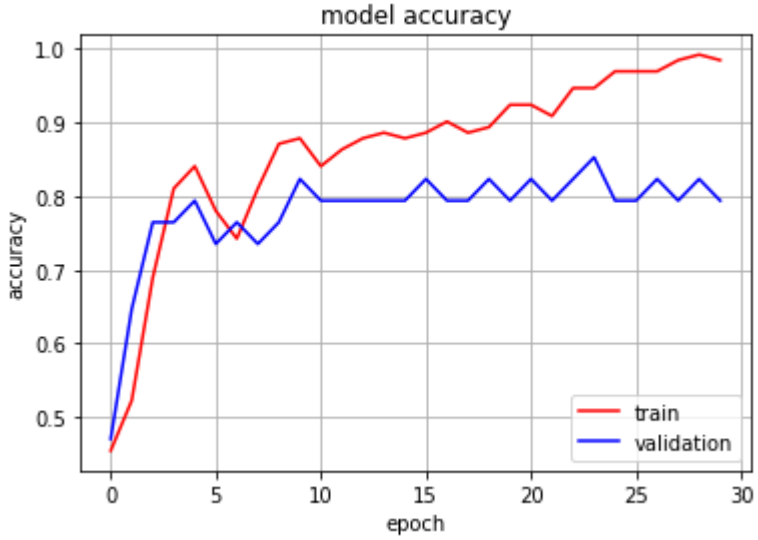
شکل ۱۲ – دقت بعد از ۵۰ ایپاک با batchهای ۱۲۸ تایی



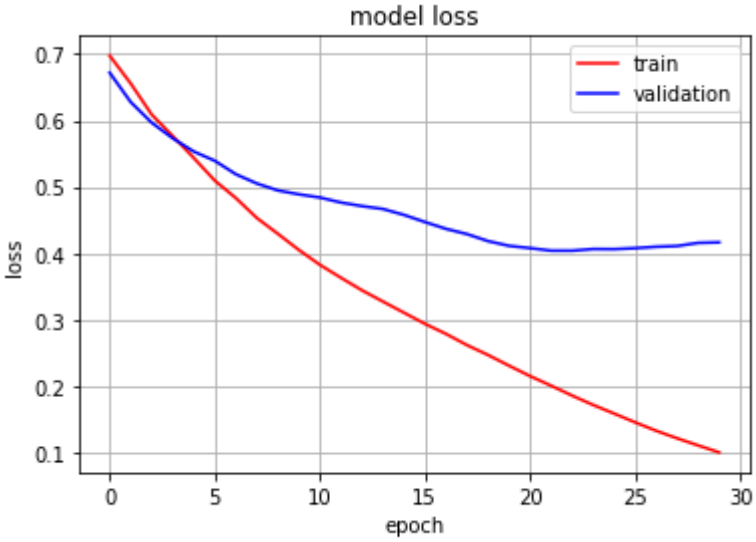
شکل ۱۳ – loss بعد از ۵۰ ایپاک با batchهای ۱۲۸ تایی

```
Test Loss 0.4068201184272766
Test Accuracy 0.8333333134651184

confusion matrix=
 [[23  4]
 [ 3 12]]
```

در نهایت حالتی که همه ی داده ها همزمان به مدل داده شوند را هم امتحان می کنیم یعنی batch را برابر ۲۰۸ می گذاریم:



شکل ۱۴ – دقت بعد از ۳۰ ایپاک با کل داده ها

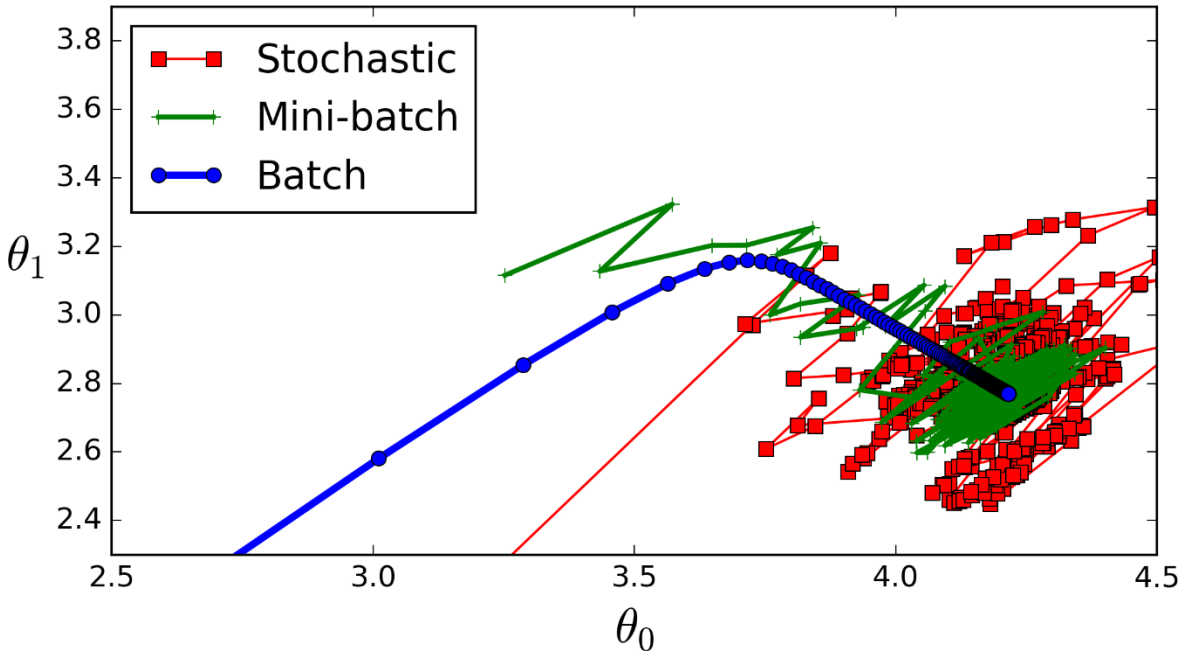


شکل ۱۵ – loss بعد از ۳۰ ایپاک با کل داده ها

```
Test Loss 0.4008658826351166
Test Accuracy 0.8809523582458496

confusion matrix=
 [[23  4]
 [ 1 14]]
```

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 کلی صعود و نزول اضافی داریم البته منطقا نیز درست است 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 ۳۲ داده با کا داده ها یکسان نیست. هرچه batch بزرگتر شده نوسانات کمتر شده همانطور که میتوان در شکل آخر دید که خیلی صاف و کم نوسان نمودارها تغییر کرده اند. 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 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. نکته ی دیگر این است که در batch های کوچکتر در مجموع سریعتر به جواب می رسیم و epoch کمتری لازم است. شکل زیر نمونه ای از تفاوت حالات مختلف در مساله ای ساده تر است که به

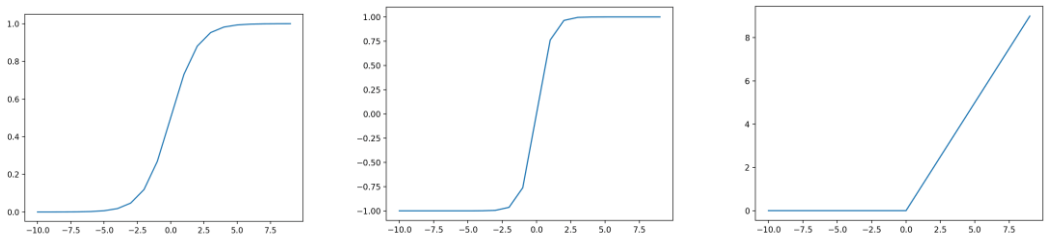
خوبی همه ی نکات گفته شده در آن مشخص است. در ادامه ازmini batch های به سایز ۳۲ و batch استفاده می کنیم زیرا در اولی مزایای هر دو حالت را داریم و در دومی نیز چون سایز داده کوچک است، مشکل حافظه نداریم و تشخیص epoch مناسب از روی نمودار loss آسانتر است زیرا نوسان نداریم.



شکل ۱۵ – مسیر طی شده توسط سه روش ورود داده ها به مدل

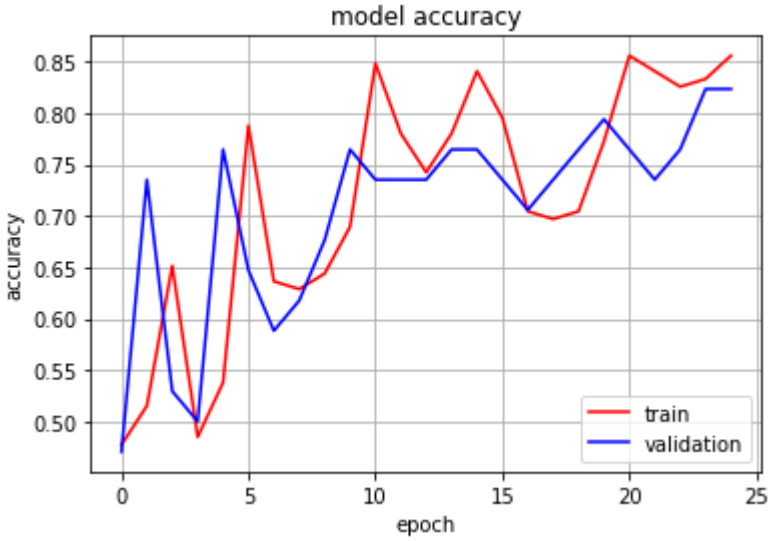
ح) epoch برابر تعداد دفعاتی است که الگوریتم کل داده ها را دیده است ولی iteration برابر است با تعداد دفعاتی که یک batch داده وارد مدل شده و عمل یادگیری انجام شده است. مقدار بهینه epoch را می توان از نمودار های loss و accuracy متوجه شد یعنی تا جایی که با افزودن epoch مدل ما روی داده validation عملکرد بهتری داشته باشد نشانگر این است که باید تعداد epoch زیاد شود. وقتی دیگر loss کمتر نشد یا دقت بهتر نشد یعنی نباید دیگر جلو رویم زیرا در صورتی که بیشتر از حد معقول مدل را آموزش دهیم 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 و مدل ما نامعتبر می شود. در batch تعداد iteration و epoch یکسان است. در stochastic تعداد iteration‌ها ۲۰۸ برابر epoch ها است. در mini batch با سایز ۳۲ برای batch تقریبا تعداد iteration‌ها ۷ برابرepoch ها 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. در سال های گذشته معمولا sigmoid و tanh بیشتر استفاده میشده ولی اخیرا relu نیز استفاده می شود. طبق جستجو در اینترنت relu برای MLP و CNN استفاده می شود و دوتای دیگر 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. در شکل زیر این توابع را مشاهده می کنید:

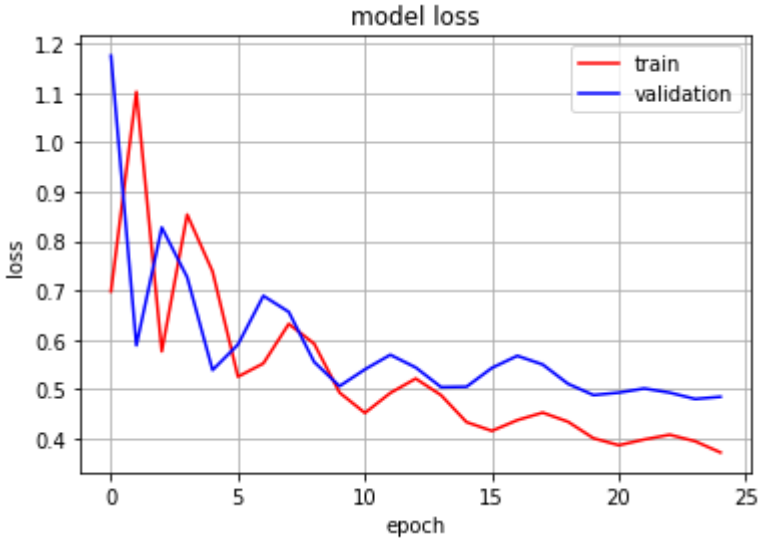


شکل ۱۶ – به ترتیب از راست relu، sigmoid و tanh

حال نتایج حاصل از tanh را مشاهده می کنیم:



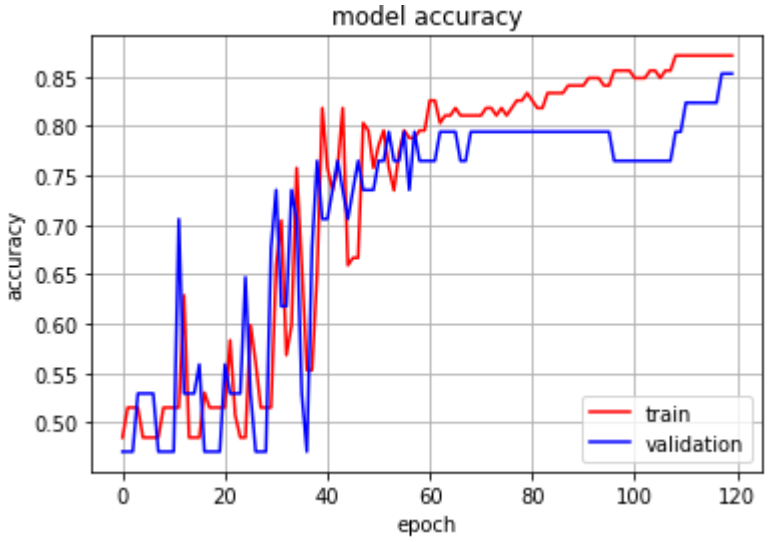
شکل ۱۷ – دقت مدل بعد از ۲۵ epoch با tanh به عنوان activation function



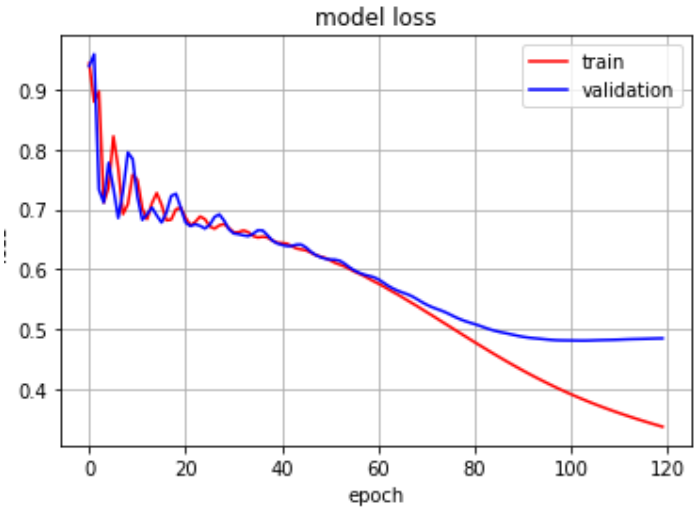
شکل ۱۸ – loss مدل بعد از ۲۵ epoch با tanh به عنوان activation function

```
Test Loss 0.4491579532623291
Test Accuracy 0.7857142686843872

confusion matrix=
 [[22  5]
 [ 4 11]]
```

سپس نتایج حاصل از sigmoid را می بینیم:



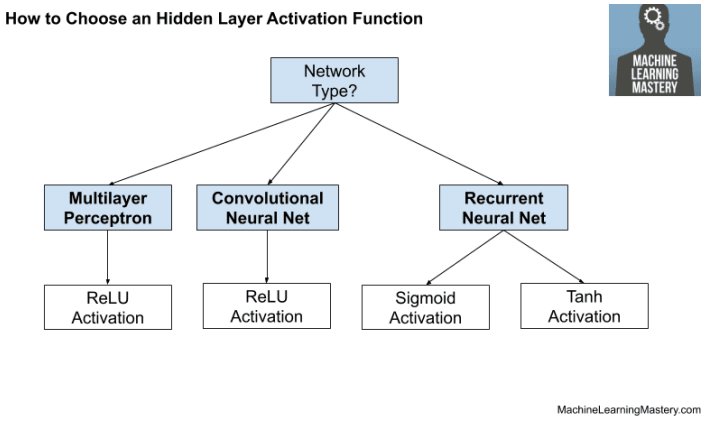
شکل ۱۹ – دقت مدل بعد از ۱۲۰ epoch با sigmoid به عنوان activation function



شکل ۲۰ – loss مدل بعد از 120 epoch با sigmoid به عنوان activation function

```
Test Loss 0.49619337916374207
Test Accuracy 0.761904776096344

confusion matrix=
 [[21  6]
 [ 4 11]]
```

مطابق انتظار عملکرد مدل ضعیف تر شد و حتی در Sigmoid تعداد ۱۲۰ 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.



شکل ۲۱ – نحوه انتخاب activation function برای شبکه

به طور خلاصه خوبی های sigmoid به شرح زیر است:

- نرم تغییر می کند و در تمامی نقاط دارای مشتق است.
- غیرخطی است پس می تواند خروجی غیرخطی بدهد.
- آسان است و به سادگی پیاده می شود.
- مشتق آن به سادگی محاسبه می شود.

بدی های آن هم در قسمت پایین آمده است:

- مرکز خروجی حول صفر نیست پس آپدیت های گرادیان خیلی تاثیر خواهند داشت و ممکن است بهینه سازی را دچار مشکل کند.
- آرام convergence می کند.
- Vanishing Gradient Problem

حال خوبی های tanh را بررسی می کنیم:

- پیوسته است و در تمامی نقاط مشتق دارد.
- اگر تمام مقادیر مثبت باشند به مشکل نخواهد خورد(بعضی توابع به مشکل خواهند خورد).
- غیرخطی است پس می تواند خروجی غیرخطی بدهد.

بدی ها هم به شرح زیر است:

- مقادیر گرادیان کوچک است.
- Vanishing Gradient Problem

در نهایت ReLU یا Rectified Linear Unit را مورد بررسی قرار می دهیم. نکات مثبت آن در قسمت زیر آمده اند:

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

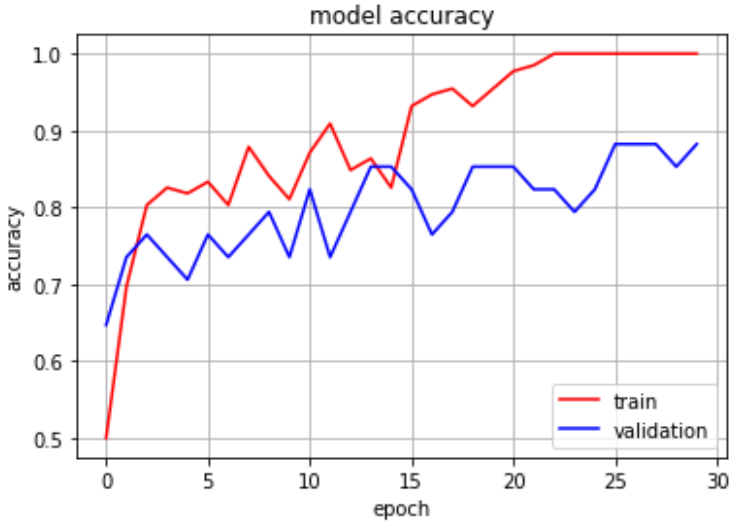
مشکلات ReLU هم به شرح زیر است:

- تا بی نهایت می رود و اینکه در صفر مشتق ندارد.
- مقدار گرادیان برای ورودی های منفی صفر است یعنی تغییری نمی کنند پس بعضی نورون های ما همیشه غیر فعال می مانند که این مشکل را با تغییر learning rate و bias می توان حل کرد.
- 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.
- 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.

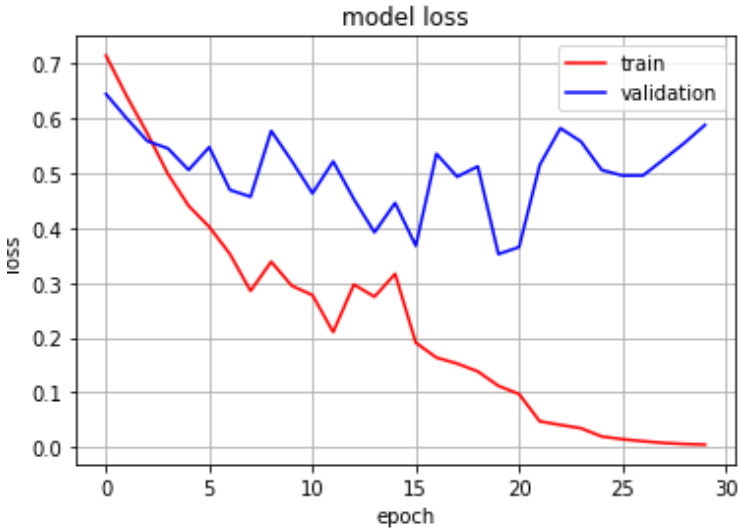
در نهایت برای مراحل بعد ReLU را انتخاب کردیم.

ی) در این مرحله یکبار به جای ۲ لایه مخفی ۴ لایه و بار دیگر ۶ لایه می گذاریم و عملکرد مدل را می سنجیم. ابتدا شبکه با ۴ 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 ۲۲ – دقت مدل در ۳۰ ایپاک با ۴ لایه مخفی

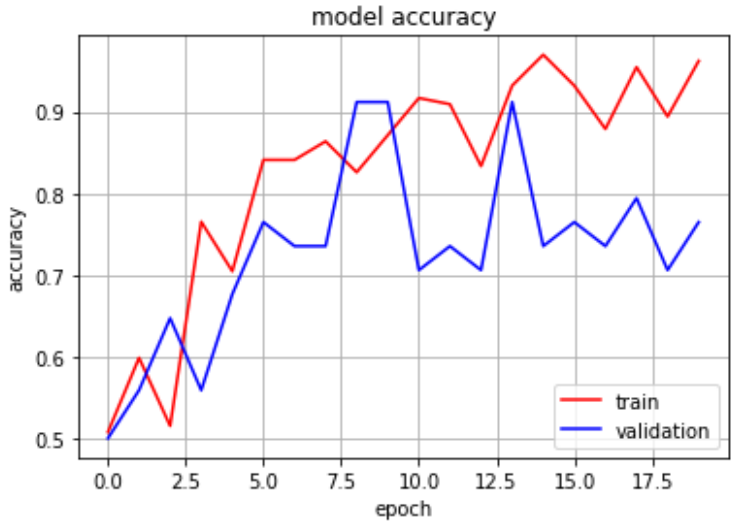


شکل ۲۳ – loss مدل در ۳۰ ایپاک با ۴ لایه مخفی

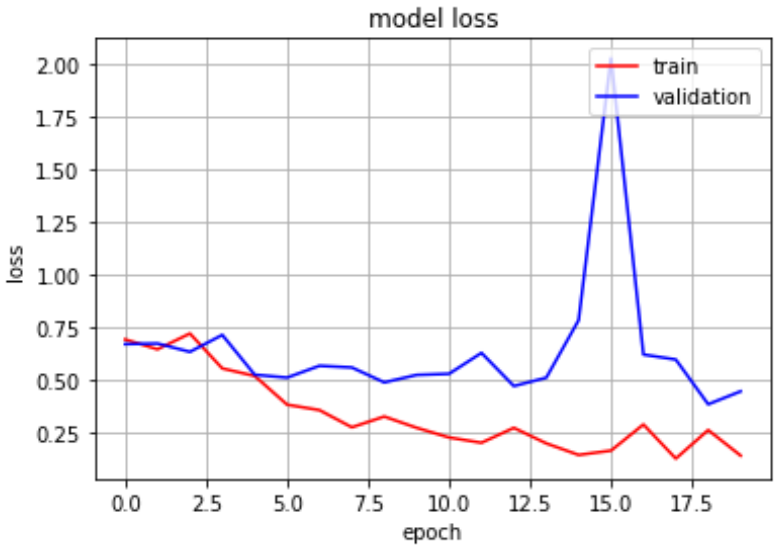
```
Test Loss 0.33054348826408386
Test Accuracy 0.8809523582458496

confusion matrix=
 [[22  5]
 [ 1 14]]
```

به وضوح می توان دید که توانایی مدل اندکی بیشتر است 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 ۶ 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 ۲۵ – دقت مدل در ۱۸ ایپاک با ۶ لایه مخفی



شکل ۲۵ – loss مدل در ۱۸ ایپاک با ۶ لایه مخفی

```
Test Loss 0.49493181705474854
Test Accuracy 0.8333333134651184

confusion matrix=
 [[21  6]
 [ 1 14]]
```

می توان دید که علی رغم افزوده شدن ۲ 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 ۴ 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
```

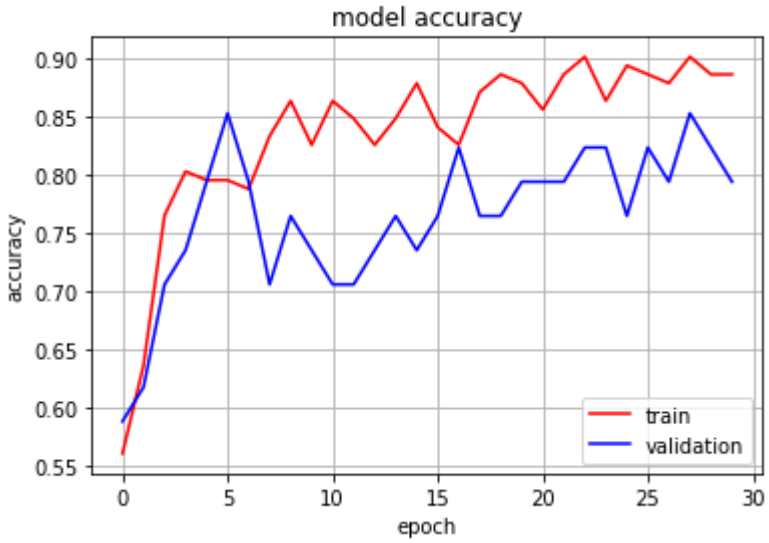
سایز batch برابر ۳۲ است و lossهم cross entropy و تابع فعال ساز ReLU است. می توان به نتیجه ی بهتر رسید با اقدامات زیر:

- عوض کردن نحوه ی train test split و بهره بردن از روش هایی مثل k-fold
- استفاده از مدل های پیچیده تر مثل CNN
- تغییر تعداد نورون ها که در مرحله بعد انجام می شود.

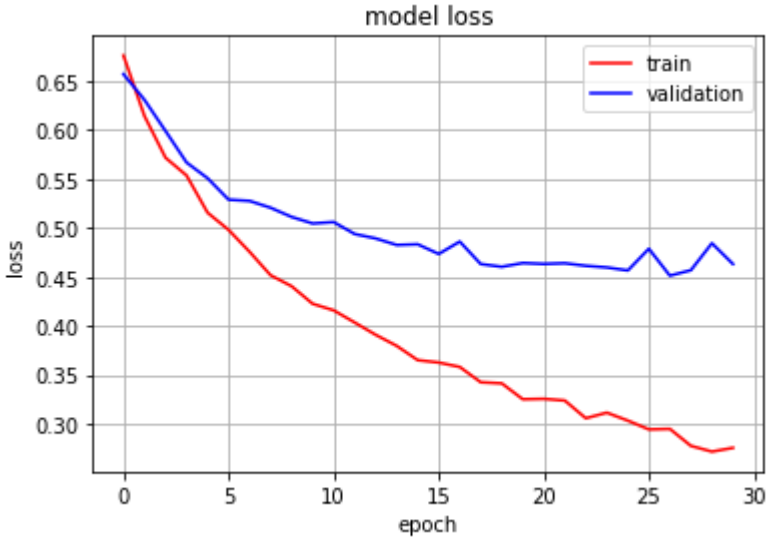
ل) به نظر می آید که کاهش تعداد نورون ها و لایه ها مدل زودتر overfit می شود زیرا complexity کاهش یافته و می توان در epoch کمتر به نقطه بهینه رسید پس در epoch زودتر وارد ناحیه overfitting می شویم.

ابتدا شبکه زیر که لایه کمتری دارد را می بینیم:

```
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 ۲۶ – دقت بعد از ۳۰ ایپاک در مدل با یک لایه مخفی

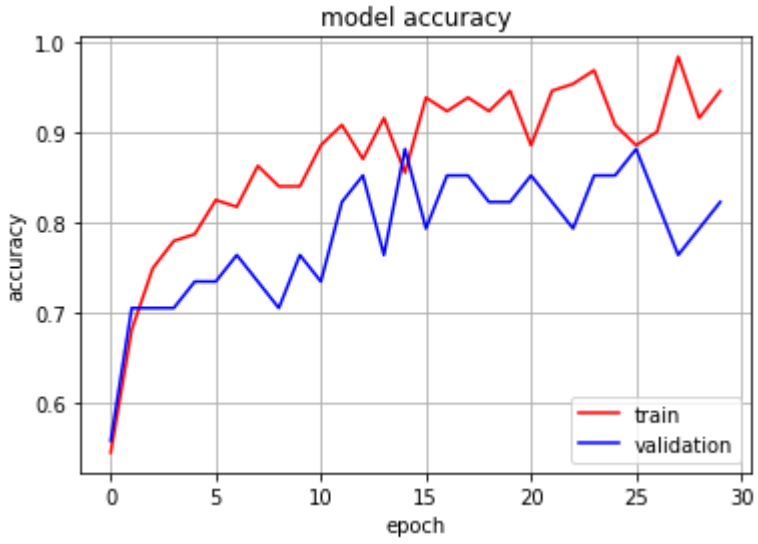


شکل ۲۷ – loss بعد از ۳۰ ایپاک در مدل با یک لایه مخفی

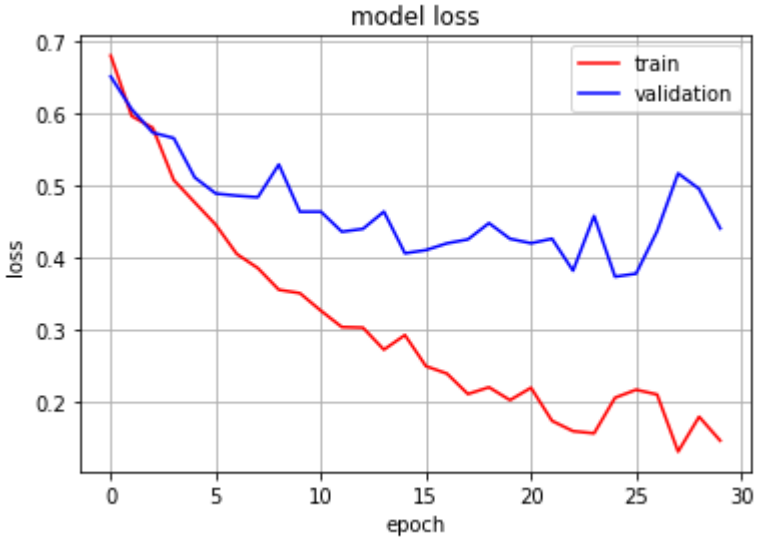
می توان دید که خیلی در ایپاک کمتری overfit نشده و کاه لایه آنقدر اثرگذار نبوده است.

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 ۲۸ – دقت بعد از ۳۰ ایپاک در مدل با نورون های کمتر در لایه مخفی



شکل ۲۹ – loss بعد از ۳۰ ایپاک در مدل با نورون های کمتر در لایه مخفی

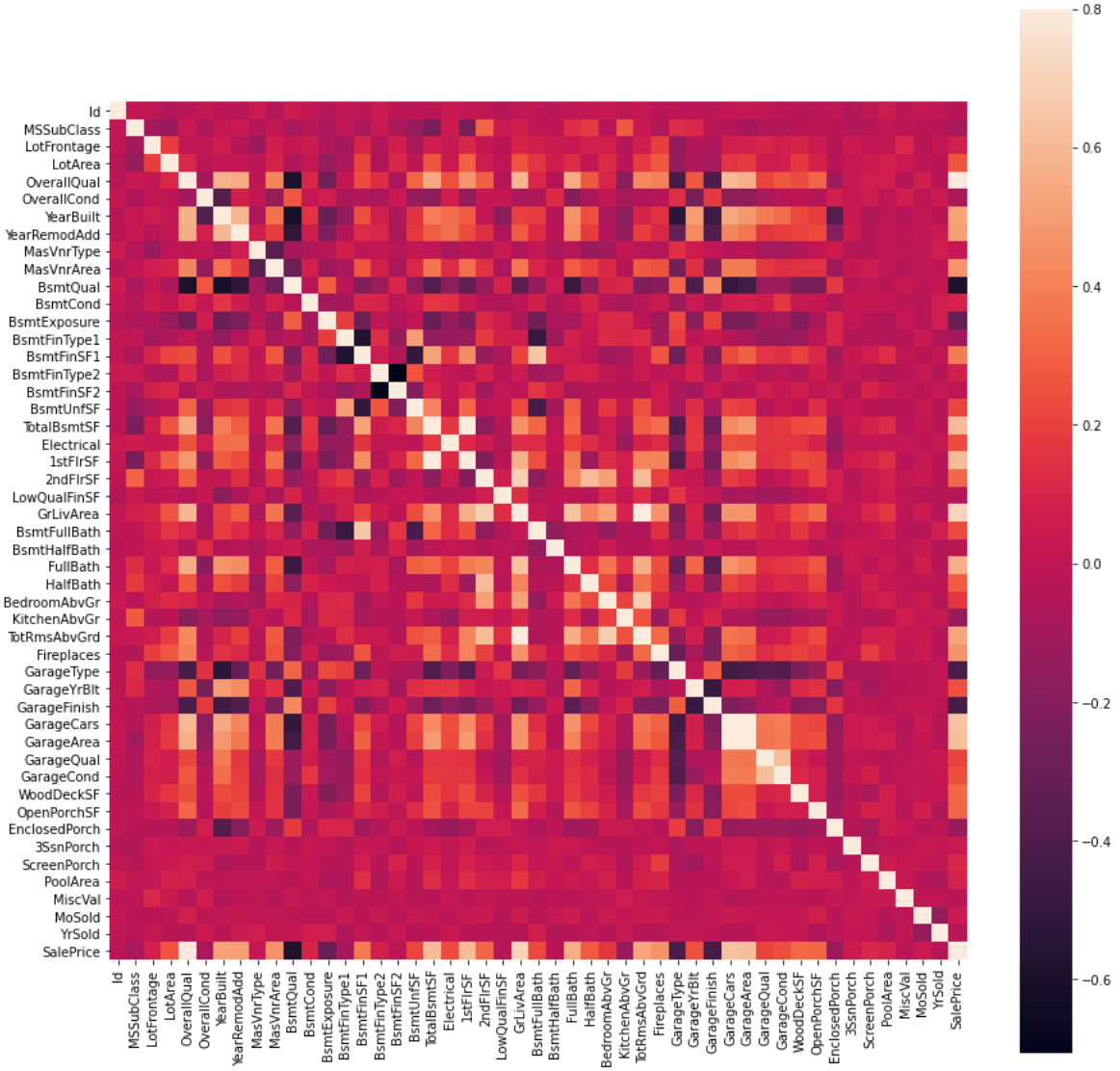
می توان دید که به وضوح در ایپاک کمتری overfitting اتفاق افتاده است و فرضیه ما در این حالت به خوبی قابل مشاهده است.

سوال ۳ – Dimension Reduction

در این سوال قصد داریم روی داده های سوال ۱ و ۲ پردازش های بیشتری انجام دهیم. کد و توضیحات کد این سوال در فایل HW2-Q3-Part1.ipynb و HW2-Q3-Part2.ipynb قرار دارد.

بخش های الف، ب و ج در Part1 و بخش های د، ه و و در Part2 قرار دارند

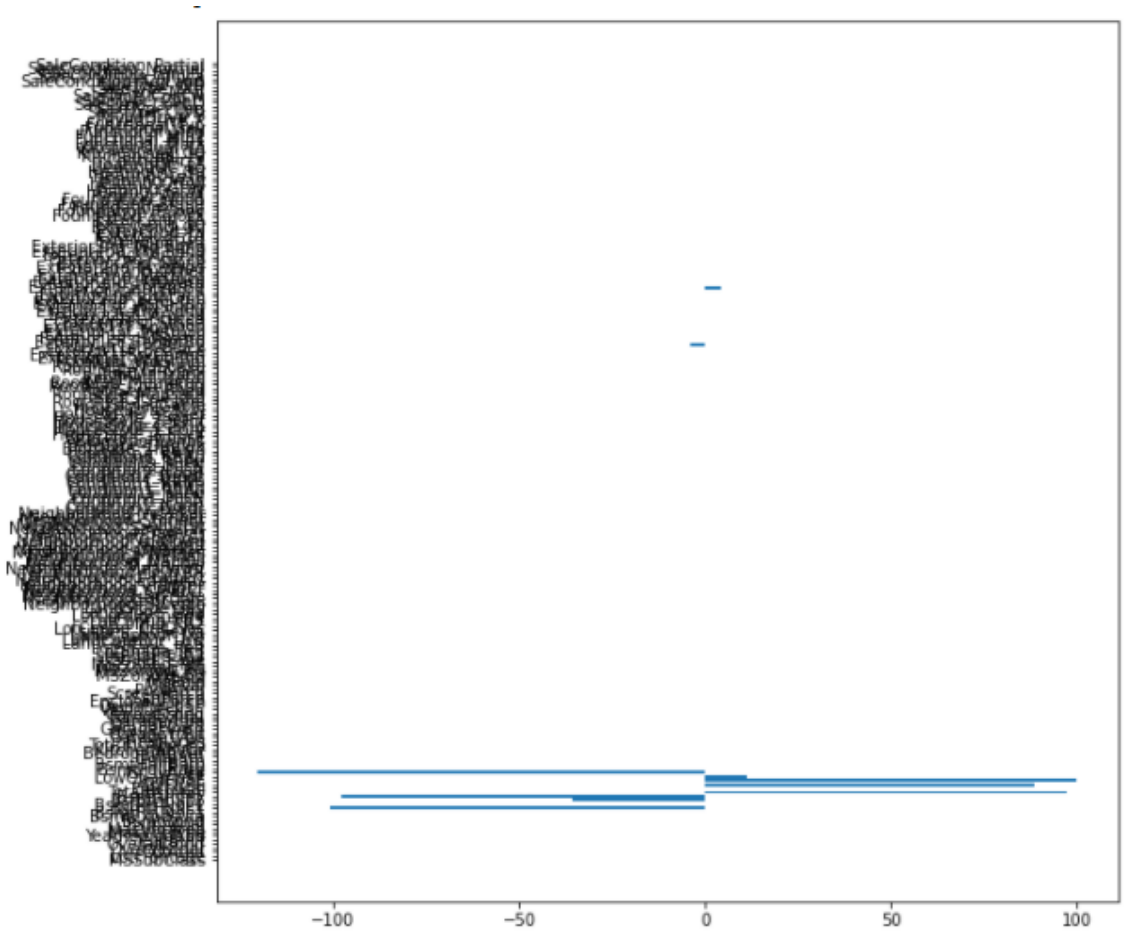
الف) ابتدا ماتریس همبستگی ویژگی ها را رسم می کنیم:



شکل ۱ – ماتریس همبستگی ویژگی ها

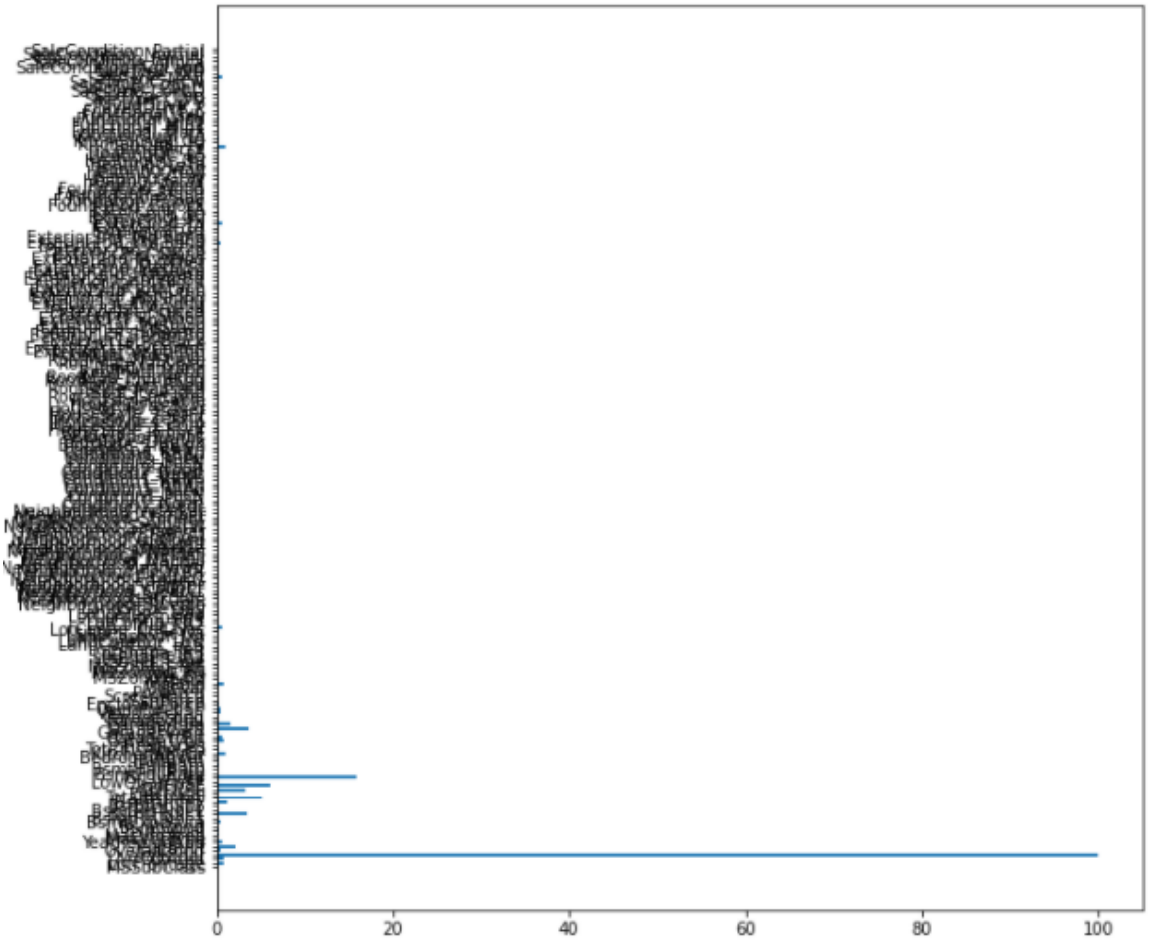
ماتریسcorrelation نشان دهنده میزان همبستگی بین هر دو ویژگی متناظر می باشد و اگر مقدار آن برای دو ویژگی بالا باشد یعنی نزدیک 1 یا 1- باشد، نشان می دهد که اطلاعات جدیدی به ما نمی دهند و می توان با یکی از آنها کار کرد و بدین گونه فضای مساله را کوچکتر کرد. همچنین می توان ویژگی هایی که کوریلیشن زیادی با SalesPrice دارند را پیدا کرد و اهمیت بیشتری به آنها داد زیرا رابطه ای خطی تر با هدف ما دارند. برای دو عمل گفته شده می توان به سادگی کد هایی نوشت و با تعیین threshold هایی ویژگی های خوب و مناسب را یافت.

ب) در این قسمت ابتدا یک regressor روی داده ها می زنیم و برای هر ویژگی یک ضریب پیدا می کنیم سپس انحراف از معیار هر ویژگی را در ضریب آن ضرب کرده و ویژگی به نام importance که اهمیت را نشان می دهد را مشخص می کنیم سپس نمودار میزان اهمیت ویژگی ها بر حسب این معیار که بین 100 و 100- است را رسم می کنیم:



شکل ۲ – میزان اهمیت ویژگی ها به کمک روش Regression

حال این کار را به کمک decision tree انجام می دهیم. این روش در هر مرحله داده ها را بر حسب هر ویژگی یکبار تقسیم بندی می کند و بر حسب یک معیار در information theory مثل آنتروپی بهترین ویژگی در آن مرحله را پیدا می کند و به همین ترتیب همه ی ویژگی ها را رده بندی می کند. نتایج به شرح زیر است:



شکل ۳ – میزان اهمیت ویژگی ها به کمک روش Decision Tree

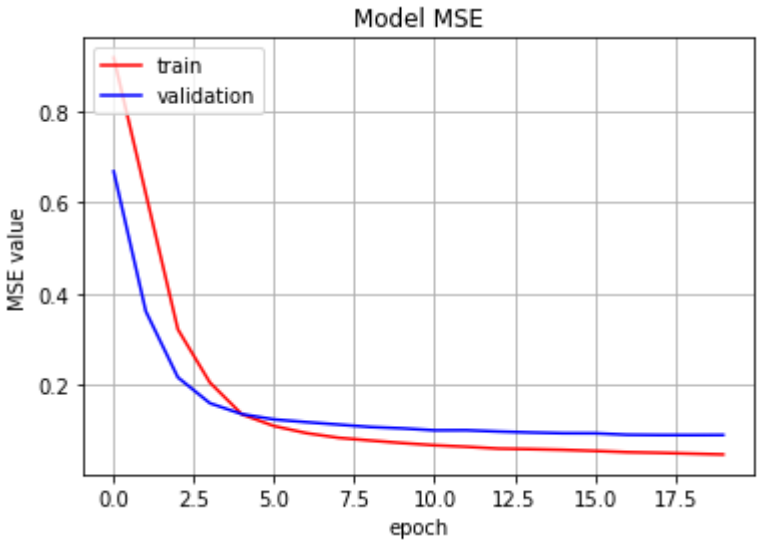
ج) حال سعی می کنیم از backward elimination استفاده کرده و بهترین feature set ممکن را پیدا کنیم. در این روش مراحل زیر را انجام داده و بعد از پیدا کردن بهترین مجموعه ویژگی ها مدل را پیاده می کنیم.

1. Select a significance level, say 5% (0.05)
2. Fit a model with all features (variables)
3. Consider the feature with the highest P-Value. If its P-value is greater than significance level (P > SL), go to step 4. Else, your model is ready.
4. Eliminate this feature (variable).
5. Fit a model with the new set of features, and go to step 3.

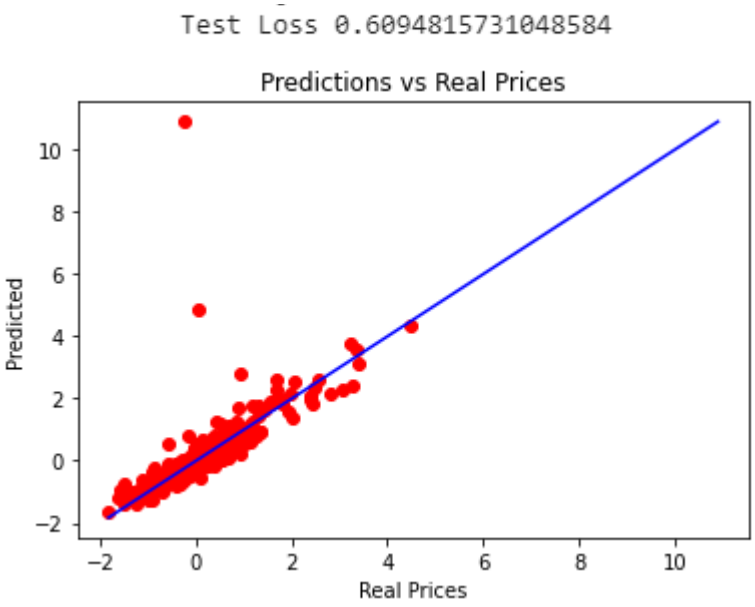
حال در 3 مرحله این کار را می کنیم. در مرحله اول threshold را ۰.۱ گذاشته و در مراحل بعدی ۰.۰۵ و در نهایت ۵۱ ویژگی که p-value کمتر از ۰.۰۵ دارند را انتخاب کرده و مدل را با آنها آموزش می دهیم:

```
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
```

نکته ی قابل توجه این است که در این روش باید در هر مرحله یکی از ویژگی ها را حذف می کردیم ولی از آنجایی که تعداد ویژگی ها خیلی بالاست و با حذف یک ویژگی p-value باقی ویژگی ها افزایش می یابد . وقتی یک ویژگی بالای threshold است در مراحل بعدی هم بالا از threshold خواهد بود، چند تا چند تا این کار را انجام می دهیم.



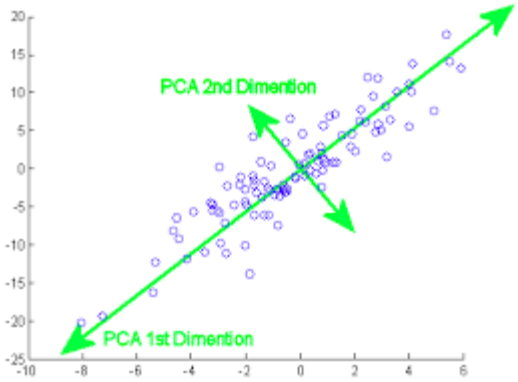
شکل ۴ – loss در مدل ۲ لایه با ReLU و استفاده از روش backward elimination روی داده ها



شکل ۵ - نمودار مقادیر پیشبینی شده بر حسب مقادیر واقعی

می توان دید که عملکرد مدل از لحاظ loss اندکی ضعیف تر شده و این روش در این مساله خیلی مناسب نبوده و شاید روش هایی مثل Bidirectional Sequential Elimination نتیجه ی بهتری داشتند.

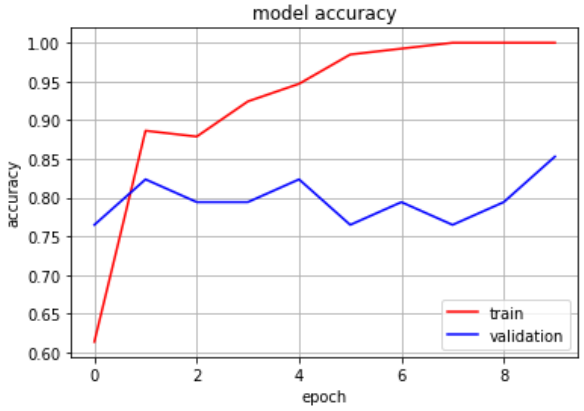
د) ابتدا به کمک کلاس PCA که در کتابخانه sklearn از پیش تعریف شده است تعداد component هایی که در آنها بیشترین واریانس و در مجموع ۹۶ درصد واریانس داده ها وجود دارد را پیدا می کنیم. در این مساله ۲۷ تا pc component اولی دارای ۹۵ درصد اطلاعات داده ها هستند و با رفتن به فضای حاصل از آنها اطلاعات زیادی از دست نمی دهیم. اگر به صورت خلاصه بخواهیم PCA را توضیح دهیم 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 ۲ 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.



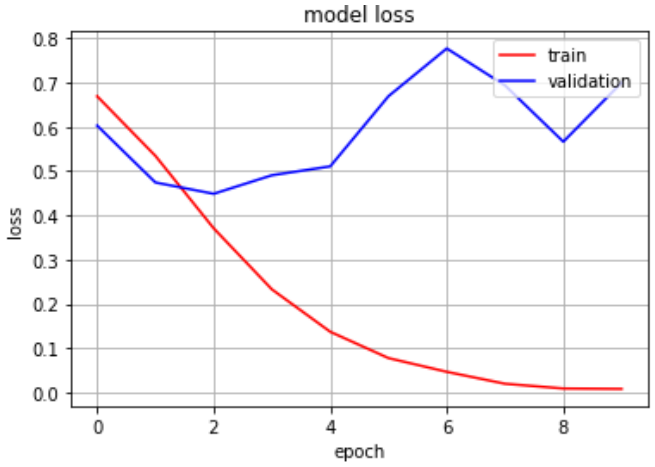
شکل ۶ – عملکرد PCA

پس از تبدیل فضا به فضای ۲۷ بعدی که ۹۵ درصد اطلاعات را در خود جای داده است و تغییر ورودی شبکه به ۲۷ 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 ۷ – خطا در مدل بهره مند از PCA



شکل ۸ – loss در مدل بهره مند از PCA

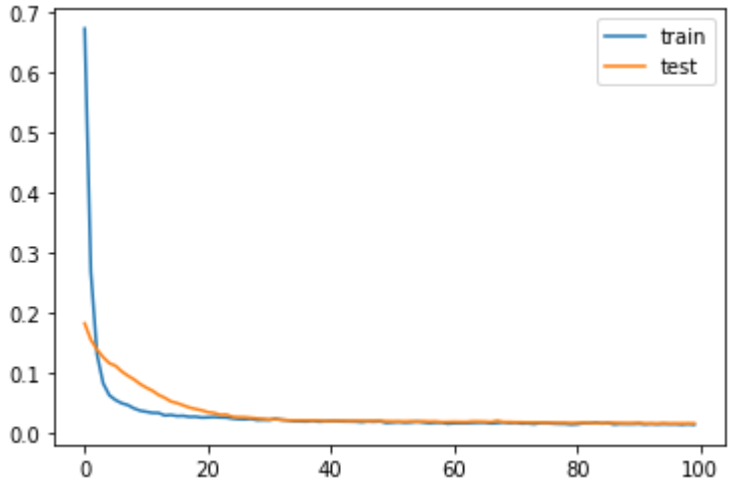
```
Test Loss 0.7445240020751953
Test Accuracy 0.8571428656578064
```

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

می توان دید که نتیجه ی حاصل تقریبا یکسان است و در زمانی نصف 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. نکته ی دیگر رسیدن به جواب در epoch خیلی کمتر است یعنی زیر ۱۰ 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. به صورت خلاصه اتوانکودر شبکه ای است که حالت ساعت شنی دارد یعنی

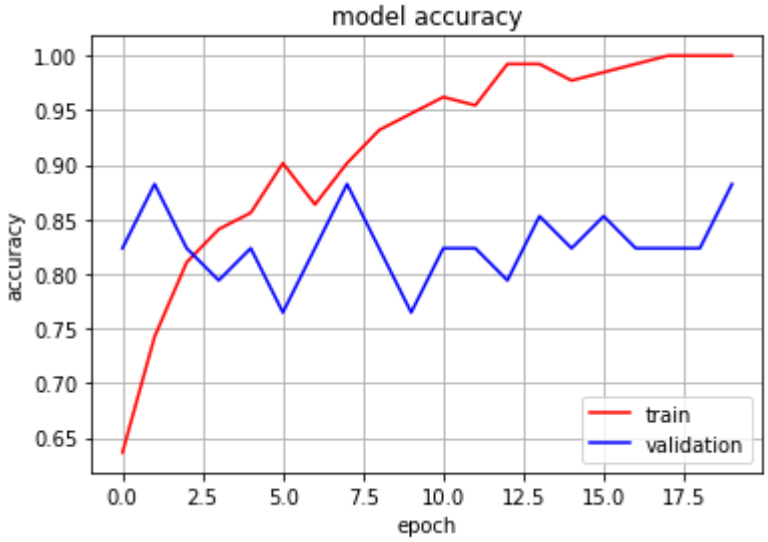
لایه های وسط آن تعداد نورون کمتری دارد و اول و آخر آن به اندازه ی ابعاد ورودی نورون دارد. این شبکه سعی می کند با این ساختار ابتدا به کمک encoder داده را به فضایی کوچکتر ببرد و سپس با برگرداندن آن به فضای ورودی بررسی کند که آیا این کاهش بعد با از دست رفتن اطلاعات همراه بوده است یا نه و اگر loss آن کوچک باشد نشان از موفقیتش است. در این مساله یکبار نورون های لایه وسط که اسمش bottleneck است را اول برابر ۶۰ یا همان ورودی می گذاریم و می بینیم که مشکلی ندارد سپس به ترتیب ۳۰، ۲۰، ۱۰ می گذاریم و در هیچ یک اطلاعات از بین نرفته است ولی وقتی ۵ گذاشتیم loss به وجود آمد و نتیجه ی یادگیری مناسب نبود پس ۱۰ نورون در لایه وسط معقول است.



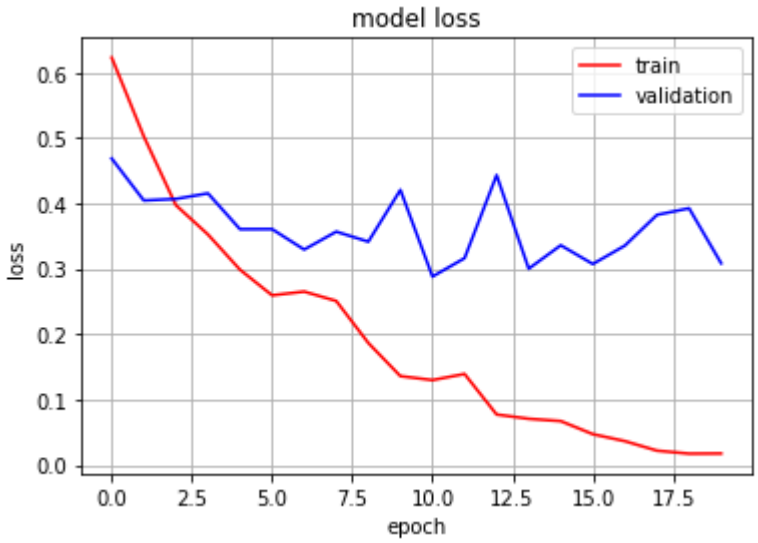
شکل ۹ – loss شبکه autoencoder با ۱۰ نورون در لایه bottleneck

می توان دید که با کاهش بعد تا ۱۰ 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 ۱۰ - دقت مدل با ورودی گذرنده از اتوانکودر



شکل ۱۱ - loss مدل با ورودی گذرنده از اتوانکودر

```
Test Loss 0.8088808655738831
Test Accuracy 0.8333333134651184

confusion matrix=
 [[24  3]
 [ 4 11]]
```

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 ۱ – مقایسه سه مدل مطرح شده

	دقت داده تست	خطای داده تست	زمان (ثانیه)
بهترین شبکه سوال ۲	٪۸۵.۷۱	۰.۶۱	۲.۵
AutoEncoder	٪۸۳.۳۳	۰.۸۰	۹
PCA	٪۸۵.۷۱	۰.۷۴	۱.۳

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.