به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری دوم

نام و نام خانوادگی	محمدعلی شاکردرگاه
شماره دانشجویی	81019487
تاریخ ارسال گزارش	1400/02/07

فهرست گزارش

سوال 1 – McCulloch-Pitts

الف: تبیین اولیه مساله و توضیحات پیش پردازش

در ابتدا داده های مسئله لود میشوند و در دیتا فریم مربوطه ذخیره میشوند. مشاهده میشود که داده ها 1460*81 هستند که به معنای 1460 داده و 81 ویژگی برای آن ها است.

در بخش الف مقصود پیش پردازش اطلاعات است که به معنای آماده سازی داده ها برای آموزش و پس از آن ارزیابی است. پیش پردازش را مرحله به مرحله انجام داده و روند کار و نتیجه آن در ادامه مکتوب میشود.

1- در ابتدا سعی میشود ویژگی هایی که بیشتر از 80% داده های آن گمشده اند و NAN در اختیار ما است حذف شوند و بدین گونه 6 ویژگی حذف خواهند شد و 76 ویژگی باقی میماند.
2- ستون هایی کهبرای پردازش مناسب نمیباشد که بسیاری از داده های آن گمشده اند و آن هایی هم که داده هایشان موجود است تنوع بالایی ندارند پس این ویژگی ها نیز در نظر گرفته نمیشود.
3- حال داده های Categorical میبایست به صورت Numerical در آورده شوند، برای اینکار ابتدا ستون هایی که Categorical هستند شناسایی شده و تک به تک cat.codes میشوند. در این داده ها 38 ویژگی Categorical بودند که بعد از تبدیل، Numerical کد شدند.
4- داده های تمامی ویژگی های عددی شده، نرمال شود
5- تمامی ستون هایی که همچنان NAN دارند (که با توجه به پیش پردازش اول، کمتر از 20% آن تعداد دارند) با مد داده ها جایگذاری میشود.

ب:

در ابتدا سعی میشود 80% داده ها(ردیف ها) به صورت کاملا تصادفی، به عنوان داده های آموزش و 20% بقیه داده ها به عنوان داده های آزمون جدا شوند.

حال برای مدل کردن شبکه دو حالت که خود شامل 2 حال میشوند در نظر گرفته خواهد شد:

1- شبکه با دو لایه مخفی و توابع فعال ساز relu :

First Model: 2 Hidden layers & both relu

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, input_dim = 75, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_45 (Dense)             (None, 512)               38912
_________________________________________________________________
dense_46 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 513
=================================================================
Total params: 302,081
Trainable params: 302,081
Non-trainable params: 0
_________________________________________________________________
```

شکل 1-1: مدل اول شبکه

همانطور که در شکل 1-1 قابل مشاهده است، در لایه مخفی اول 512 نورون با بعد ورودی 75 داده شده است و در لایه مخفی دوم نیز 512 نورون قرار گرفته است، هر دو لایه مخفی از تابع فعال ساز relu بهره میبرند و در لایه آخر نیز یک نورون برای تخمین قیمت قرار داده شده و چون Regression مورد بررسی قرار داده شده است، تابع فعال ساز آن Linear قرار داده شده است.

2- شبکه با دو لایه با تابع فعال ساز sigmoid

Second Model: 2 Hidden layers & one sigmoid

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, input_dim = 75, activation='sigmoid'))
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_54 (Dense)             (None, 512)               38912
_________________________________________________________________
dense_55 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_56 (Dense)             (None, 1)                 513
=================================================================
Total params: 302,081
Trainable params: 302,081
Non-trainable params: 0
_________________________________________________________________
```

شکل 1-2: دو لایه با یک تابع سیگموید

همانطور که در شکل 1-1 قابل مشاهده است، در لایه مخفی اول 512 نورون با بعد ورودی 75 داده شده است و در لایه مخفی دوم نیز 512 نورون قرار گرفته است، هر یک لایه مخفی از تابع فعال ساز relu بهره میبرد و دیگری از sigmoid در لایه آخر نیز یک نورون برای تخمین قیمت قرار داده شده و چون Regression مورد بررسی قرار داده شده است، تابع فعال ساز آن Linear قرار داده شده است.

3- شبکه با سه لایه مخفی و توابع فعال ساز relu

Third Model: 3 Hidden layers & relu

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, input_dim = 75, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_57 (Dense)             (None, 512)               38912
_________________________________________________________________
dense_58 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_59 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_60 (Dense)             (None, 1)                 513
=================================================================
Total params: 564,737
Trainable params: 564,737
Non-trainable params: 0
_________________________________________________________________
```

شکل 1-3: مدل سه لایه شبکه با رلو

همانطور که در شکل 1-3 قابل مشاهده است، در لایه مخفی اول 512 نورون با بعد ورودی 75 داده شده است و در لایه مخفی دوم و سوم نیز 512 نورون قرار گرفته است، هر سه لایه مخفی از تابع فعال ساز relu بهره میبرند و در لایه آخر نیز یک نورون برای تخمین قیمت قرار داده شده و چون Regression مورد بررسی قرار داده شده است، تابع فعال ساز آن Linear قرار داده شده است.

4- شبکه با سه لایه با تابع فعال ساز sigmoid

Forth Model: 3 Hidden layers & one sigmoid

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, input_dim = 75, activation='sigmoid'))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_65 (Dense)             (None, 512)               38912
_________________________________________________________________
dense_66 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_67 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_68 (Dense)             (None, 1)                 513
=================================================================
Total params: 564,737
Trainable params: 564,737
Non-trainable params: 0
_________________________________________________________________
```

شکل 1-4 سه لایه و یک سیگموید

همانطور که در شکل 1-4 قابل مشاهده است، در لایه مخفی اول 512 نورون با بعد ورودی 75 داده شده است و در لایه مخفی دوم و سوم نیز 512 نورون قرار گرفته است، لایه اول تابع فعال ساز sigmoid دارد و هر دو لایه مخفی بعدی از تابع فعال ساز relu بهره میبرند و در لایه آخر نیز یک نورون برای تخمین قیمت قرار داده شده و چون Regression مورد بررسی قرار داده شده است، تابع فعال ساز آن Linear قرار داده شده است.

ج: برای هر کدام از 4 حالت پیاده سازی مدل، تمام خواسته های مسئله براورده خواهد شد و نشان داده خواهد شد با فرض آنکه تابع loss برابر با MSE باشد.

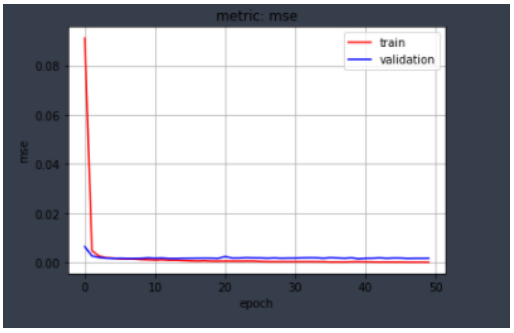
1- شبکه با دو لایه مخفی و توابع فعال ساز relu :

در این حالت، مدل ما بعد از 50 ایپاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

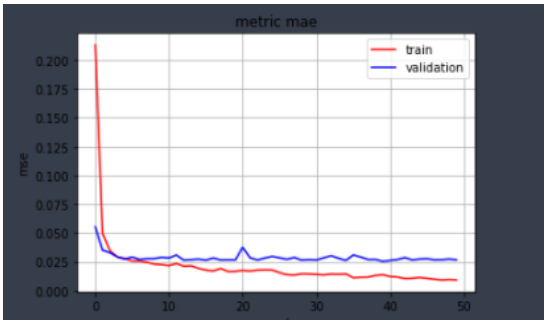
شکل 1-5- دولایه مخفی relu

برای متریک mse برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



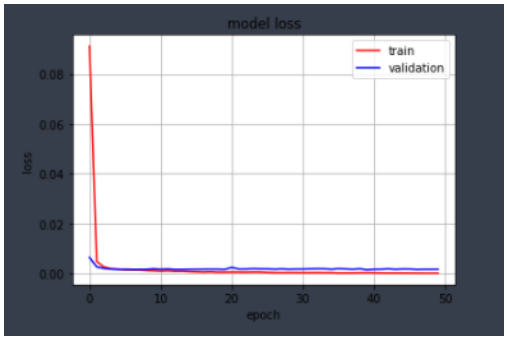
شکل 1-6 دو لایه مخفی relu: mse metric

برای متریک mae برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



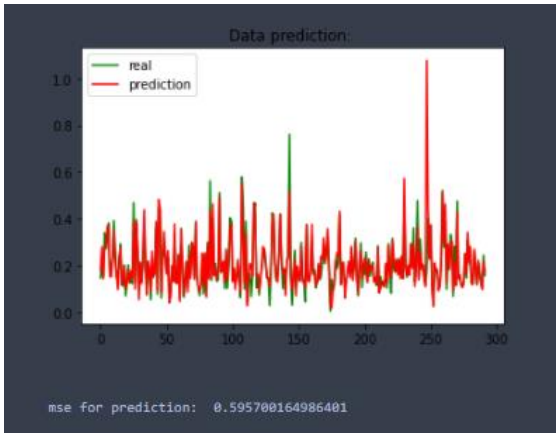
شکل 1-7 دو لایه مخفی relu: mae metric

برای تابع Loss برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



شکل 1-8دو لایه مخفی relu: mae metric

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید



شکل 1-9 دو لایه مخفی relu: prediction

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.595 شد
- ایپاک بهینه میتواند در این مورد 10 باشد

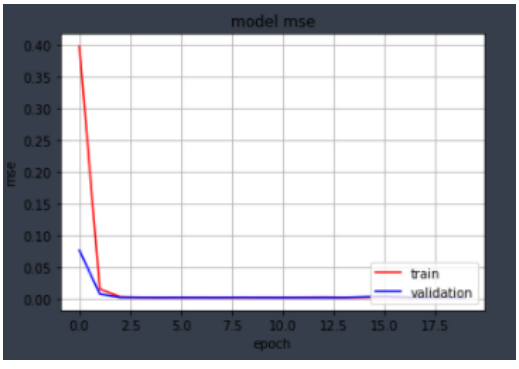
2- شبکه با دو لایه با تابع فعال ساز sigmoid :

در این حالت، مدل ما بعد از 20 ایپاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

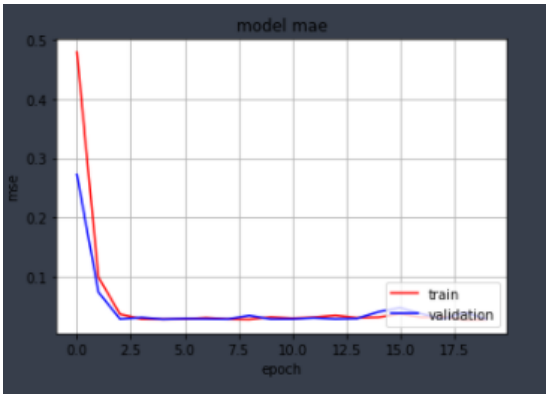
شکل 1-10- دولایه مخفی و یک سیگموید

برای متریک mse برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



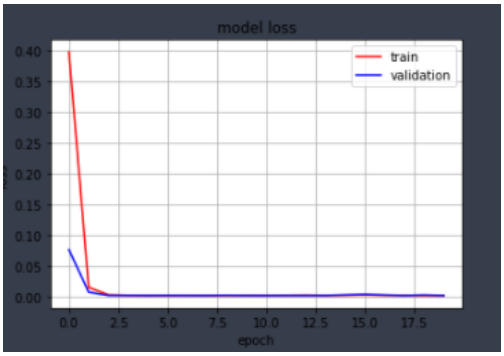
شکل 1-11 دو لایه مخفی سیگموید mse metric :

برای متریک mae برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



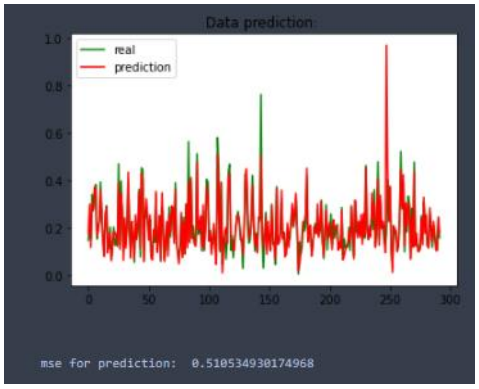
شکل 1-12 دو لایه مخفی سیگموید mae metric :

برای تابع Loss برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



شکل 1-13دو لایه مخفی سیگموید mae metric :

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید:



شکل 1-14 دو لایه مخفی سیگموید prediction :

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.510 شد
- ایپاک بهینه در این مورد میتواند 3 باشد

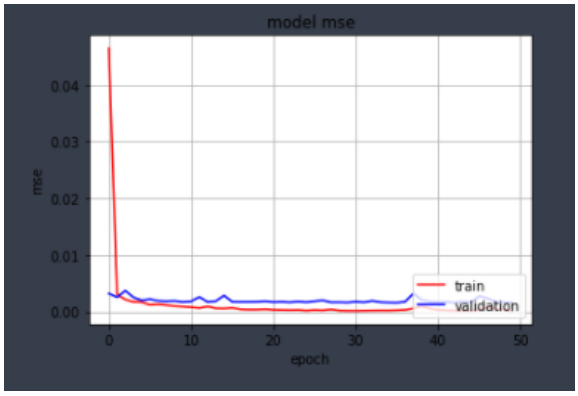
3- شبکه با سه لایه مخفی و توابع فعال ساز relu

در این حالت، مدل ما بعد از 50 ایپاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

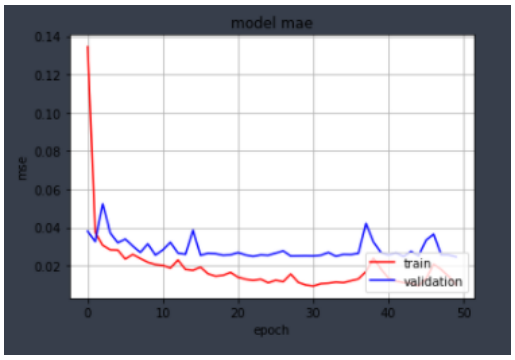
شکل 1-15- سه لایه مخفی relu

برای متریک mse برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



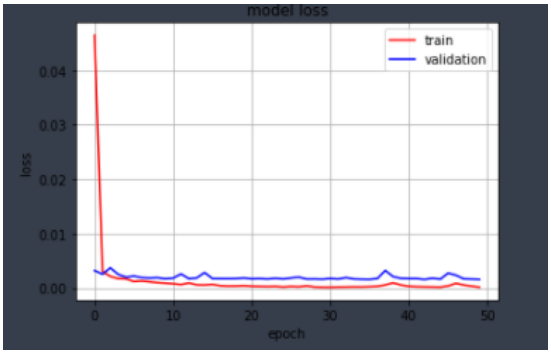
شکل 1-16 سه لایه مخفی relu: mse metric

برای متریک mae برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



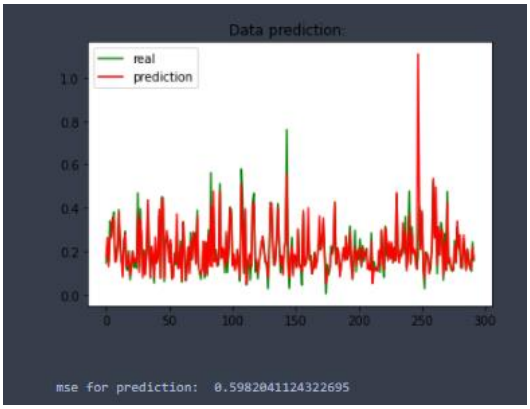
شکل 1-17 سه لایه مخفی relu: mae metric

برای تابع Loss برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



شکل 1-18 سه لایه مخفی relu: mae metric

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید



شکل 1-19 سه لایه مخفی relu: prediction

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.598 شد
- ایپاک بهینه میتواند در این مورد 10 باشد

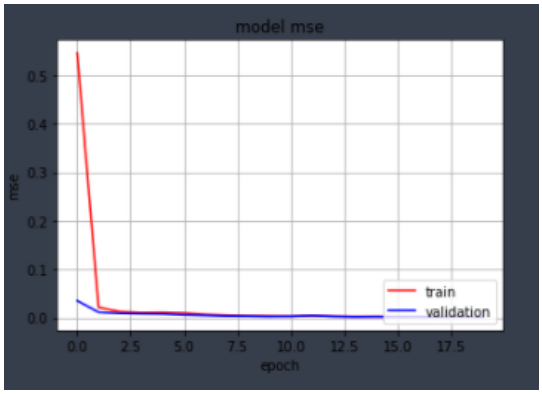
4- شبکه با سه لایه با تابع فعال ساز sigmoid

در این حالت، مدل ما بعد از 20 ایپاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
In [328]: model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
          history = model.fit(x_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

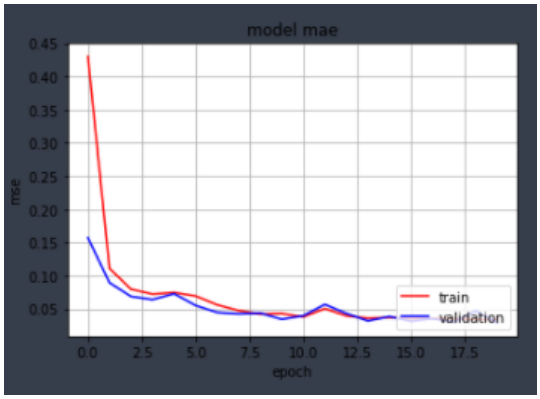
شکل 1-20- سه لایه مخفی و یک سیگموید

برای متریک mse برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



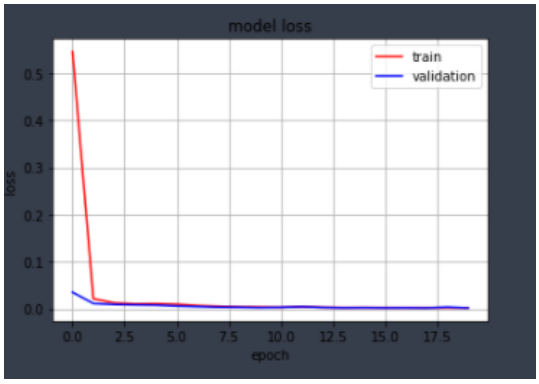
شکل 1-21 سه لایه مخفی سیگموید mse metric :

برای متریک mae برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



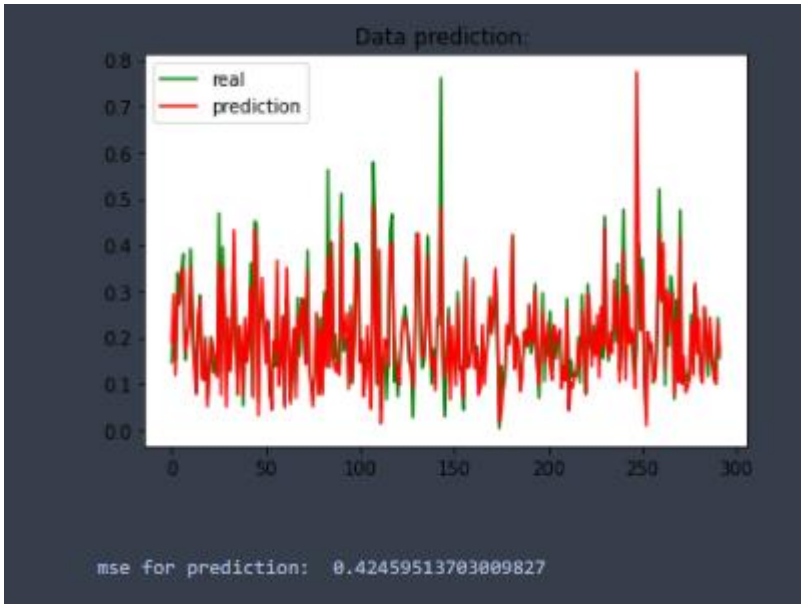
شکل 1-22 سه لایه مخفی سیگموید mae metric :

برای تابع Loss برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



شکل 1-23 سه لایه مخفی سیگموید mae metric :

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید:



شکل 1-24 سه لایه مخفی سیگموید prediction :

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.424 شد
- ایپاک بهینه در این مورد میتواند 15 باشد(به علت متریک mae)
- بهترین عملکرد بین مدل ها با Loss Function: mse را دارد

د: برای هر کدام از 4 حالت پیاده سازی مدل، تمام خواسته های مسئله براورده خواهد شد و نشان داده خواهد شد با فرض آنکه تابع loss برابر با MAE باشد

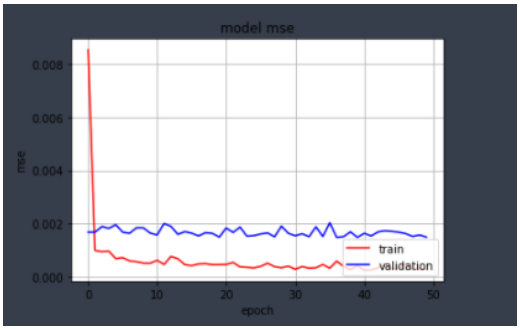
1- شبکه با دو لایه مخفی و توابع فعال ساز relu :

در این حالت، مدل ما بعد از 50 ایپاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mae', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

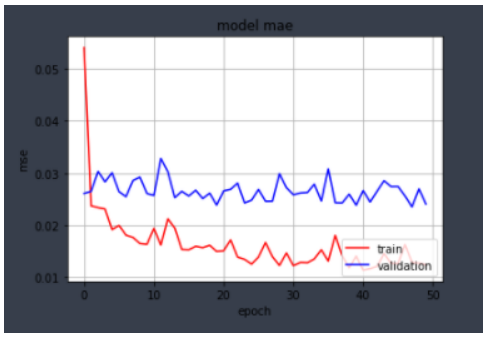
شکل 1-25- دولایه مخفی relu

برای متریک mse برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



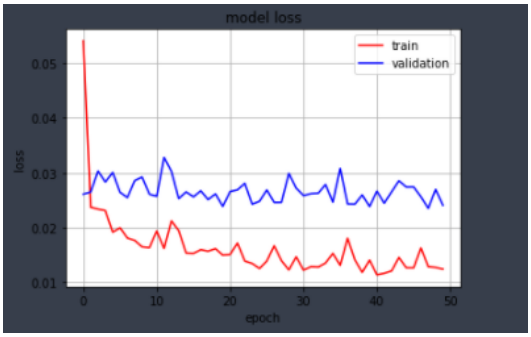
شکل 1-26 دو لایه مخفی relu: mse metric

برای متریک mae برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



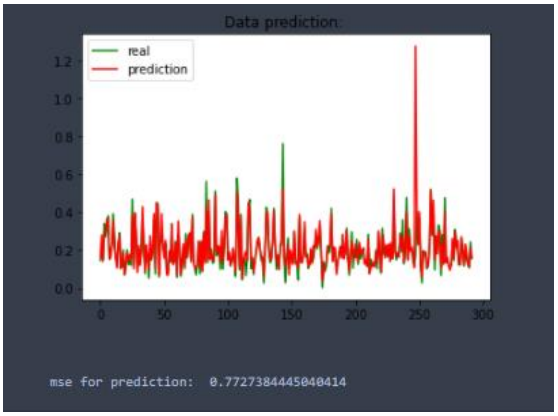
شکل 1-27 دو لایه مخفی relu: mae metric

برای تابع Loss برای داده های train و Evaluation پلات با 100 ایپاک در تصویر پایین آورده شده است:



شکل 1-28 دو لایه مخفی relu: mae metric

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید



شکل 1-29 دو لایه مخفی relu: prediction

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.772 شد
- تعداد اپیاک های بهینه میتواند 10 باشد

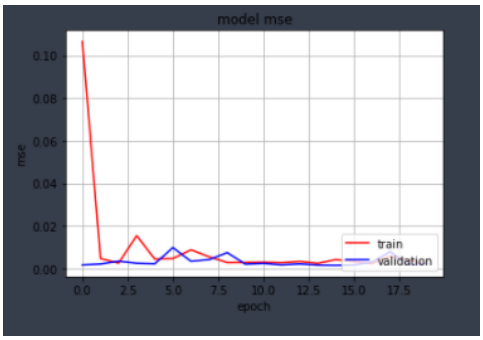
2- شبکه با دو لایه با تابع فعال ساز sigmoid

در این حالت ، مدل ما بعد از 20 اپیاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mae', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

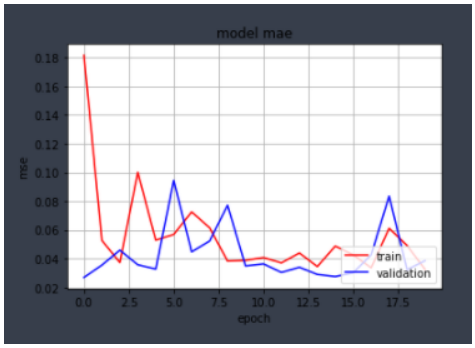
شکل 1-30- دولایه مخفی و یک سیگموید

برای متریک mse برای داده های train و Evaluation پلات با 20 اپیاک در تصویر پایین آورده شده است:



شکل 1-31 دو لایه مخفی سیگموید mse metric :

برای متریک mae برای داده های train و Evaluation پلات با 20 اپیاک در تصویر پایین آورده شده است:

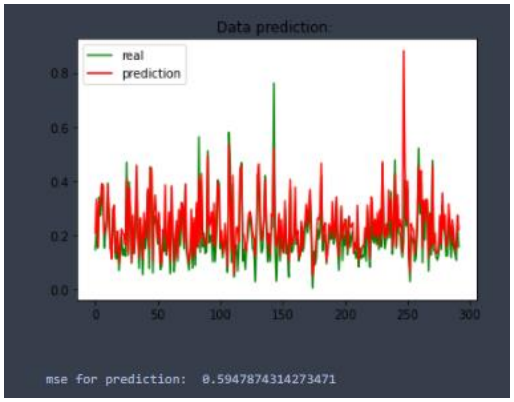


شکل 1-32 دو لایه مخفی سیگموید mae metric :

برای تابع Loss برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:

شکل 1-33دو لایه مخفی سیگموید mae metric :

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید:



شکل 1-34 دو لایه مخفی سیگموید prediction :

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.594 شد
- مشخص است که ایپاک بهینه در این مورد 10 میباشد

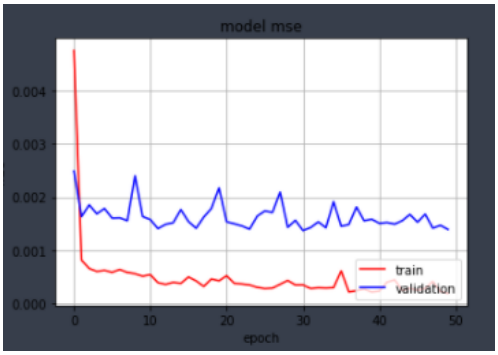
3- شبکه با سه لایه مخفی و توابع فعال ساز relu

در این مدل ما بعد از 50 ایپاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mae', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

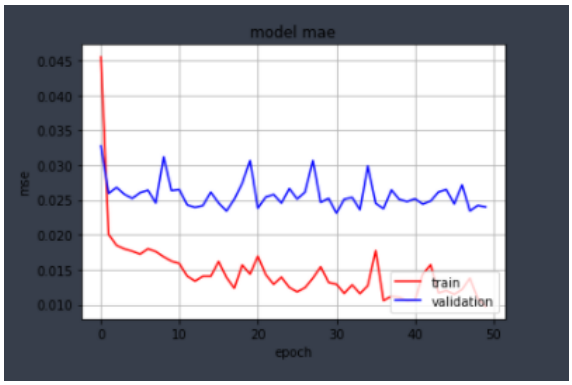
شکل 1-35- سه لایه مخفی relu

برای متریک mse برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



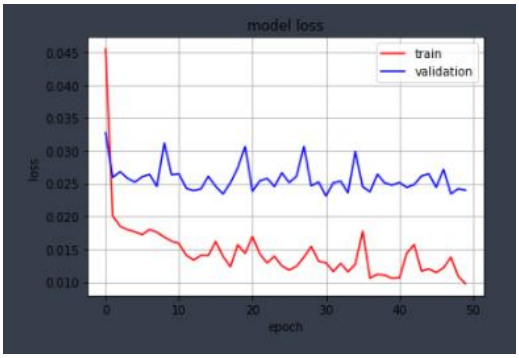
شکل 1-36 سه لایه مخفی relu: mse metric

برای متریک mae برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



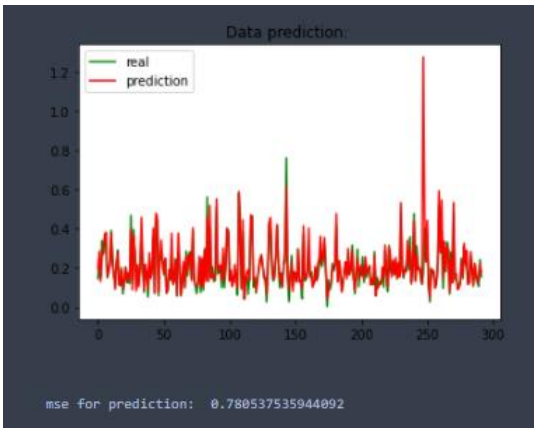
شکل 1-37 سه لایه مخفی relu: mae metric

برای تابع Loss برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



شکل 1-38 سه لایه مخفی relu: mae metric

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید



شکل 1-39 سه لایه مخفی relu: prediction

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.780شد
- اپیاک بهینه میتواند در این مورد 20 باشد

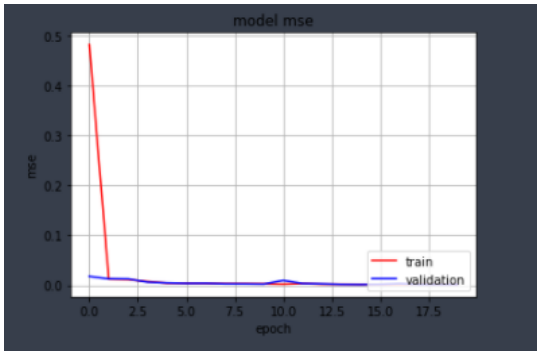
4- شبکه با سه لایه با تابع فعال ساز sigmoid

در این حالت، مدل ما بعد از 20 اپیاک و بچ سایز 32تایی در حالی که 0.2 از داده ها را برای Validation قرار داده است با مشخصات زیر آموزش میبیند:

```
model.compile(loss='mae', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

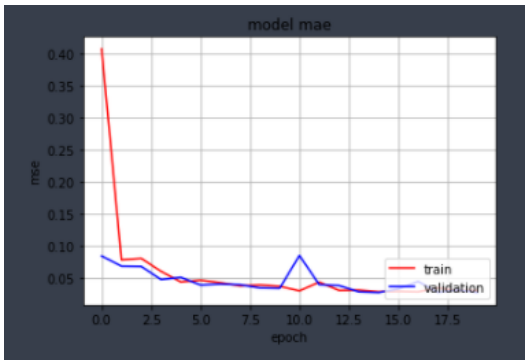
شکل 1-40- سه لایه مخفی و یک سیگموید

برای متریک mse برای داده های train و Evaluation پلات با 20 اپیاک در تصویر پایین آورده شده است:



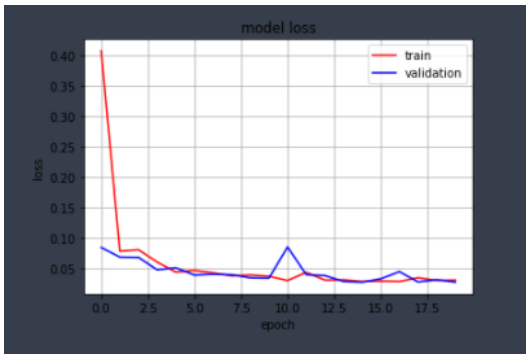
شکل 1-41 سه لایه مخفی سیگموید mse metric :

برای متریک mae برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



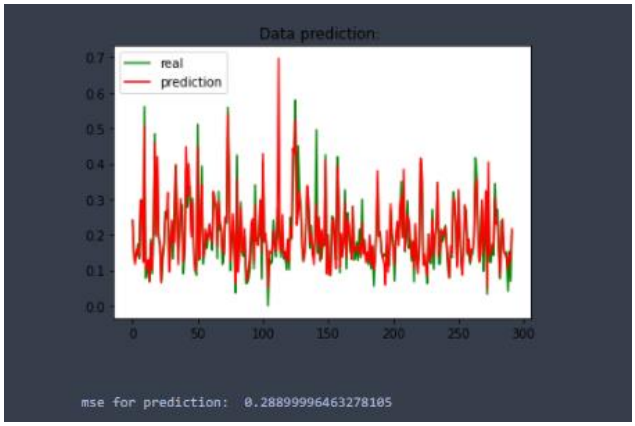
شکل 1-42 سه لایه مخفی سیگموید mae metric :

برای تابع Loss برای داده های train و Evaluation پلات با 20 ایپاک در تصویر پایین آورده شده است:



شکل 1-43 سه لایه مخفی سیگموید mae metric :

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید:



شکل 1-44 سه لایه مخفی سیگموید prediction :

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.28 شد
- اپاک بهینه در این مورد میتواند 15 باشد(به علت متریک mae)
- بهترین عملکرد بین مدل ها با Loss funcyion: mae را دارد

ه: روابط MSE و MAE در زیر تبیین میشود:

معیار خطای MSE :

معیار خطای MSE به صورت زیر بیان میشود:

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

MSE = mean squared error
n = number of data points
Y_i = observed values
$\hat{Y}_i$ = predicted values

همانطور که دیده میشود مقدار حقیقی از مقدار تخمینی کم میشود و **مربع مجموع** برای تمامی داده ها تقسیم بر تعداد داده ها رابطه MSE را میسازد.

معیار خطای MAE :

معیار خطای MAE به صورت زیر بیان میشود:

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

MAE = mean absolute error
y_i = prediction
x_i = true value
n = total number of data points

همانطور که دیده میشود مقدار حقیقی از مقدار تخمینی کم میشود و این کار برای تمامی داده ها انجام شده و تقسیم بر تعداد داده ها رابطه MAE را میسازد.

میتوان مشاهده نمون در که در MSE از اردر توان های دوم بهره بده شده، این در حالی است که در MAE از اردر توان اول بهره برده شده است.

هر دو مدل کاربرد های خود را دارند، به طور رایج از MSE بیشتر از MAE استفاده میشود و MAE بیشتر معیاری Tasked-Based میباشد مانند شرایطی که داده ها outlier زیادی دارند اما MSE متداول ترین و رایج ترین معیار برای ما میباشد.

با مشاهده به نمونه های مدل شده با دو لایه و سه لایه که در دو حالت تمام توابع Relu یا دارای لایه ای با تابع فعال ساز Sigmoid میتوان مشاهده نمود که مدل ها در حالت های مختلف پیاده سازی با توابع Loss مختلف، متفاوت عمل کرده اند و به صورت کلی میتوان گفت که توابع با MSE Loss بهتر عمل کرده اند چرا که به عنوان Fact هم پذیرفتنی است که به طور میانگین (مخصوصا برای داده های بزرگ) MSE بهتر از MAE عمل میکند.

(هرچند مشاهده میشود کمترین خطای بدست آمده از مدلسازی شبکه 3 لایه با یک تابع فعال ساز sigmoid و Loss function mae بوده است)

سوال ۲ – MLP (Classification)

در این پرسش میباست با استفاده از MLP در شاخه Classification داده های sonar طبقه بندی شوند و جنس سیلندر (آهن یا سنگ) به آن ها لیبل زده شود.

الف: پیش پردازش، تقسیم داده، مدل و معماری شبکه

پیش پردازش:

برای این سوال خواسته شده از فایل sonar استفاده شود، آن را دانلود کرده و لود کرده و شروع به پیش پردازش میکنیم، در ابتدا با این نکته مواجه میشویم که 60 ویژگی داریم و یک ستون کلاس بندی، پس در مجموع 61 ستون داریم اما ستون ها Label نخورده اند، پس ابتدا لیبل ها را برای 60 ستون ویژگی به صورت عددی اعمال میکنیم و ستون 61م را ابتدا عددی و سپس rename به 'class' میکنیم. داده های ستون 'class' یا 'R' یا 'M' هستند.

تمامی مقادیر ستون ها عددی هستند و نیازی به تبدیل Categorical به numerical نیست

هیچ ستونی NAN یا داده گمشده مشاهده نمیشود پس نیازی به پر کردن این مقادیر یا حذف آن ها نیست.

تمام داده های مربوط به ستون class را به 1 برای R و 0 برای M تقسیم میشود

تقسیم داده:

همانند روش ارائه شده در سوال قبل میتوان عمل کرد، به این صورت که 80% داده هارا **به صورت رندوم** برای داده های یادگیری و 20% بقیه را برای تست قرار میدهیم. دقت داریم که از آن 80% داده ای که برای یادگیری انتخاب میکنیم، 20% آن برای ارزیابی خواهد بود. مزیت این روش علاوه بر سادگی آن، این است که برای هنگامی که دیتا ست به صورت wide پخش نشده باشد، به اندازه کافی از هر دو کلاس انتخاب میشوند، به صور مثال در این دیتا ست تمام دیتا های کلاس R ابتدا آمده اند و بقیه اطلاعات کلاس M در زیر آن قرار دارد پس بهترین کار این است که سطر های رندوم استنخاب کنیم.

پیاده سازی شبکه:

Modeling

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(512, activation='relu'))    #Hidden Layer 2
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 512)               31232
_________________________________________________________________
dense_10 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 1026
=================================================================
Total params: 294,914
Trainable params: 294,914
Non-trainable params: 0
_________________________________________________________________
```

شکل 2-1 شبکه پیاده شده ابتدایی

همانطور که در شکل 2-1 قابل مشاهده است، در لایه مخفی اول 512 نورون با شکل ورودی 60 داده شده است و در لایه مخفی دوم نیز 512 نورون قرار گرفته است، هر دو لایه مخفی از تابع فعال ساز relu بهره میبرند و در لایه آخر نیز یک نورون برای تخمین قیمت قرار داده شده و چون Classification مورد بررسی قرار داده شده است، تابع فعال ساز آن softmax قرار داده شده است و شبکه از 294914 پارامتر بهره میبرد

ب: نمودار تغییرات دقت و خطای دقت در هر ایپاک

برای این قسمت ابتدا مدلمان به صورت مقابل compile و fit میشود:

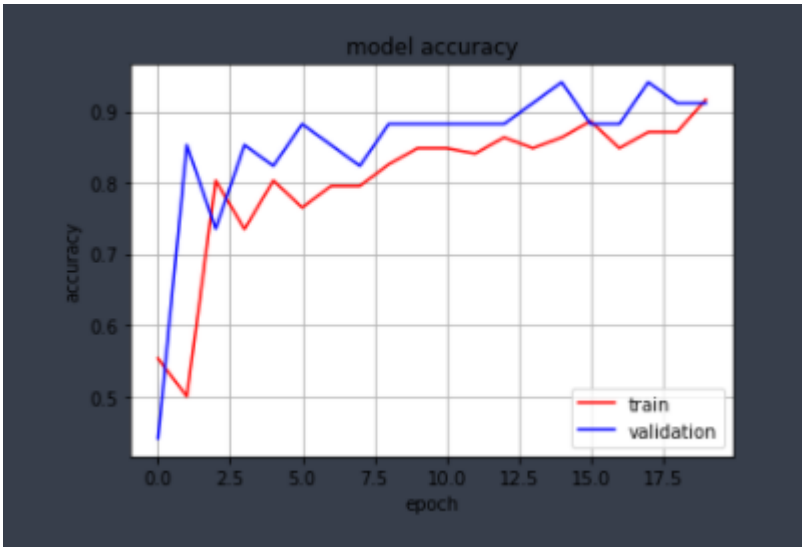
```
Compiling, Fitting, and Plots

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-2 کامپایل و فیت مدل شبکه classification

همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

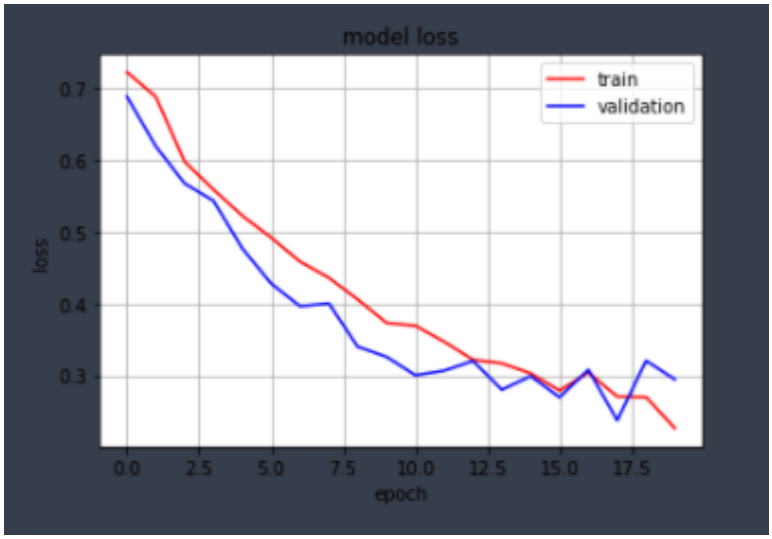
نمودار دفت در هر ایپاک در شکل 2-3 در زیر قابل دیدن است:



شکل 2-3 نمودار دقت قسمت ب

مشاهده میشود دقت رو به افزایش میباشد در هر ایپاک و به دقت خیلی خوبی رسیدیم بعد از 20 ایپاک.

نمودار Loss در هر ایپاک در شکل 2-4 در زیر قابل دیدن میباشد:



شکل 2-4 نمودار Loss قسمت ب

همانطور که مشاهده میشود، به نتیجه مطلوب رسیدیم، در 20 ایپاک، تابع Loss ما برای داده های Test & Validation بسیار کاهش پیدا کرده است.

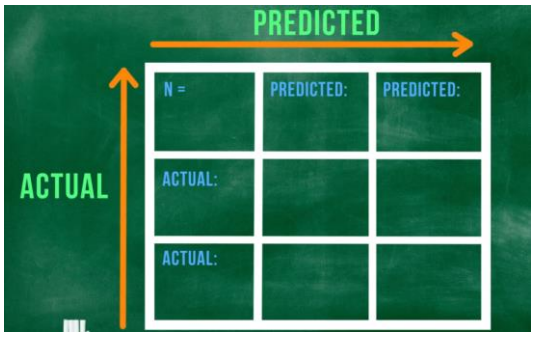
ج: مقادیر خطا، دقت و Confusion matrix

ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 999us/step - loss: 0.3064 - accuracy: 0.8571
Test Loss 0.30638477206230164
Test Accuracy 0.8571428656578064
confusion matrix=
 [[16  4]
 [ 2 20]]
```

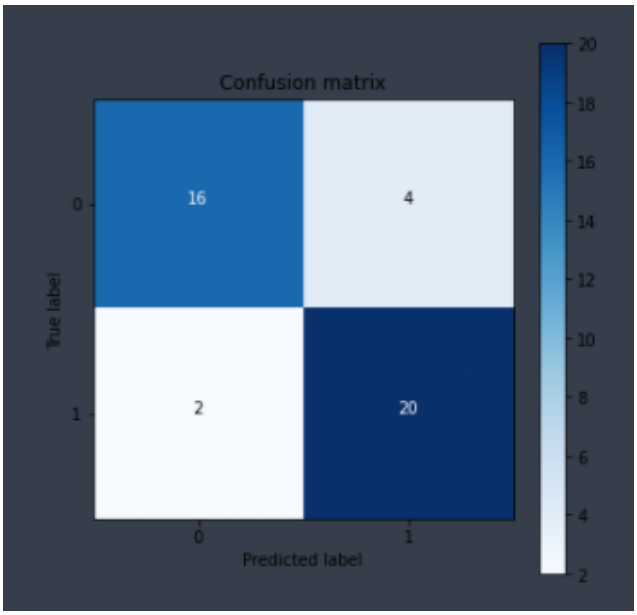
شکل 2-5 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن

رابطه مقدار دهی ماتریس Confusion به شکل مقابل است:



شکل 2-6 مقدار گیری ماتریس Confusion

و برای داده های ما پاسخ آن به شکل مقابل میشود:



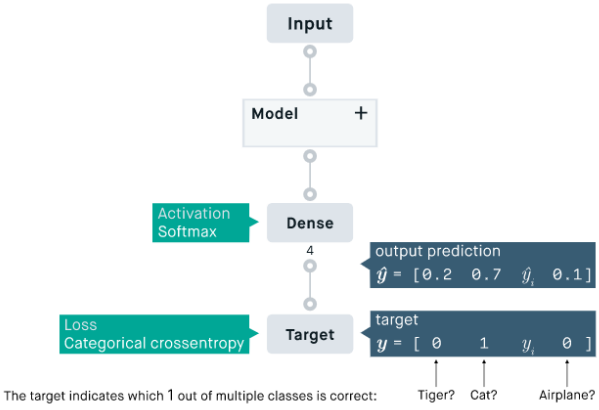
شکل 2-7 ماتریس Confusion برای مدل

مشاهده میشود که به بقت بالایی رسیده ایم و ماتریس Confusion هم این امر را تایید میکند

د:

معیار خطا در این بخش، categorical_crossentropy بود که در کلاس دستیار آموزشی درباره آن صحبت شد.

ابتدا قبل از تبیین این تابع به عنوان Loss function در شکل زیر فرایند جلو رفتن مسیر آموزش قابل مشاهده است:



شکل 2-8 تبیین مسیر آموزش

در این روند، در مرحله Targeting قابل مشاهده است که categorical_crossentropy به عنوان تابع Loss استفاده شده است، این تابع به صورت زیر مقدار دهی میشود:

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

شکل 2-9 تابع categorical_crossentropy

که در آن Y_i مقادیر حقیقی و Y_i^ مقادیر پیش بینی شده هستند.

مزیت:

در واقع این تابع برای تشخیص تفاوت دو توزیع احتمال گسسته از یکدیگر جایگاه دارد اهمیت ویژه این lost function در Single-Label Category میباشد. به این صورت که اگر کلاس ما 0 و 1 باشد اگر عددی به صفر نزدیک تر باشد، با لیبل اشتباه Loss آن برای یک بزرگ میشود و برعکس.

توابع دیگری به عنوان Loss function هستند که میتوانند مفید باشند مانند Hing Loss و Squared Hing Loss و MSE و ... که میتوانند مورد استفاده قرار بگیرند اما به واضحا همانطور که اشاره شد در این مساله مهم Classifyکردن Single-Label Category میباشد که این کار با تابع Loss استفاده شده بهتر انجام میشود

ه: معیار دقت و معیار های دیگر برای سنجش توانایی شبکه

ابتدا معیار دقت در مرحله قبل را دوباره به دست بررسی بگذاریم:

```
2/2 [==============================] - 0s 999us/step - loss: 0.3064 - accuracy: 0.8571
Test Loss 0.30638477206230164
Test Accuracy 0.8571428656578064
confusion matrix=
 [[16  4]
 [ 2 20]]
```

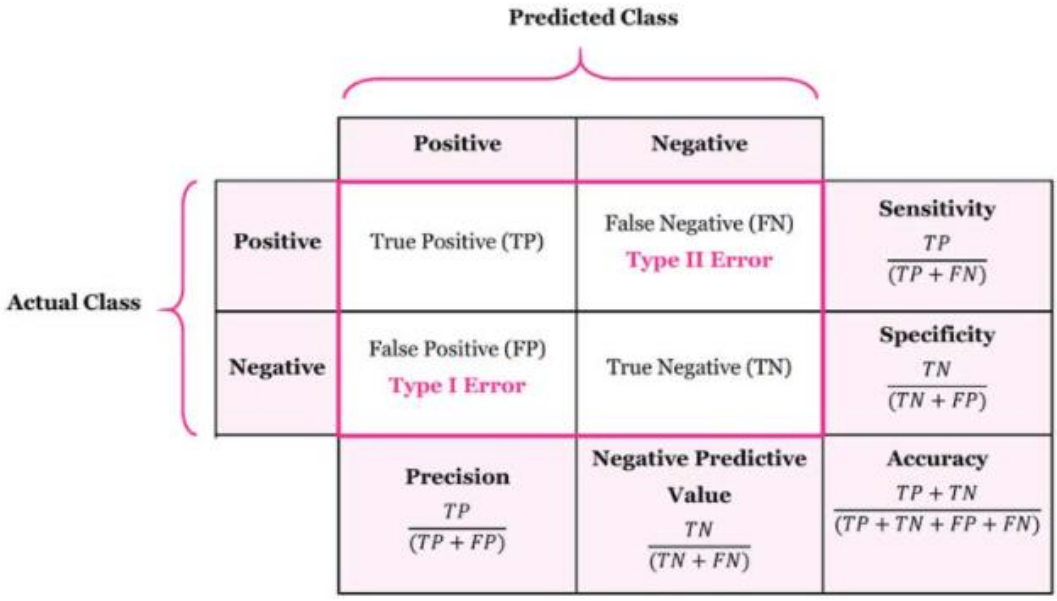
شکل 2-10 دقت در حالت اولیه

مشاهده میشود که دقت 0.857 و Loss برابر با 0.306 است.

ابتدا معیار دقت را تبیین میکنیم:

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

برای همین از ماتریس Confusion استفاده میشود:



برای آنکه معیار هایی برای سنجش توانایی شبکه ارائه کنیم میتوانیم به موارد زیر اشاره کنیم:

1- تشکیل ماتریس Confusion
2- ارزیابی دقت محض Accuracy
3- ارزیابی صحت Precision
4- Recall برای زمانی که ارزش False negative بالا باشد
5- F1 Score
6- MCC که بیانگر کیفیت کلاس بندی برای یک مجموعه باینری میباشد

و: Batch Sizes: 32, 64, and 128

همانطور که در سوال خواسته شده است، با روش stochastic mini batch based مسئله را دوباره مدل میکنیم و از batch size های 32 و 64 و 128 استفاده میکنیم(مرسوم است که از توان های 2 برای اینکار استفاده شود.) سپس هرکدام را شرح میدهیم و بهترین را انتخاب میکنیم:

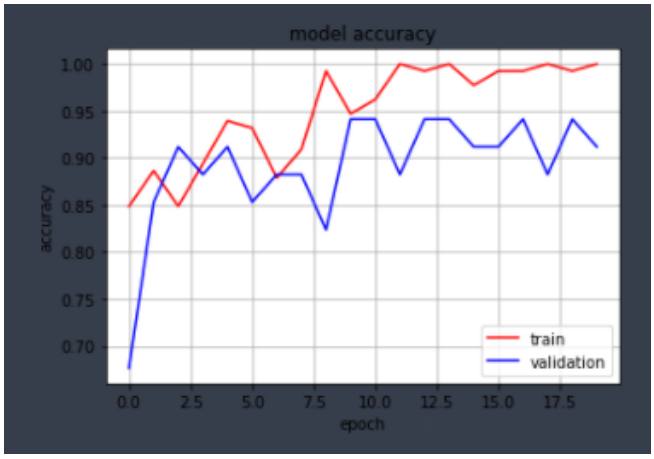
1- حالت اول stochastic mini batch based با batch_size=32

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-14 کامپایل مینی بچ 32تایی

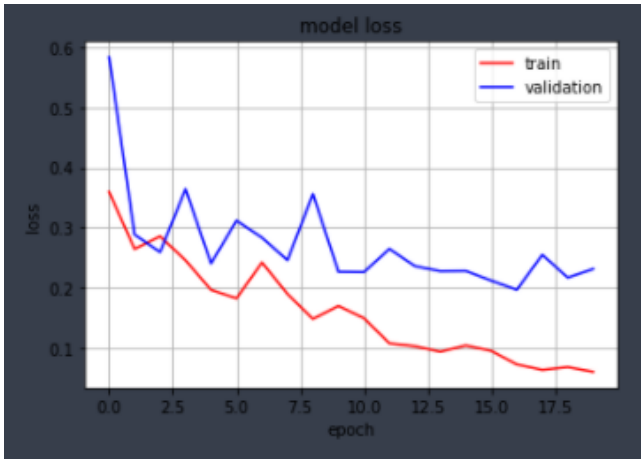
واضحا در طی 20 ایپاک شبکه ما با بچ سایز 32، train میشود و با 20% از داده های test برای Validation استفاده میشود.

نتیجه نمودار دقت :



شکل 2-15 دقت با بچ سایز 32

نتیجه نمودار Loss :



شکل 2-16 Loss با بچ سایز 32

همچنین اطلاعات دقیق به شرح زیر است:

```
2/2 [==============================] - 0s 1ms/step - loss: 0.2473 - accuracy: 0.9286
Test Loss 0.24725297093391418
Test Accuracy 0.9285714030265808
confusion matrix=
 [[20  0]
 [ 3 19]]
```

شکل 2-17 اطلاعات دقیق بچ سایز 32

دقت 0.982 و Loss برابر بت 0.247 شد و ماتریس Confusionقابل مشاهده است.

2- حالت دوم stochastic mini batch based با batch_size=64

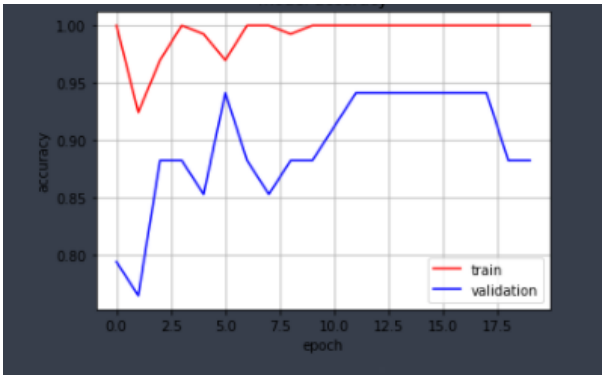
```
stochastic mini batch based Modeling: Size 64

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=64, validation_split=0.2)
```

شکل 2-18 کامپایل مینی بچ 64ایی

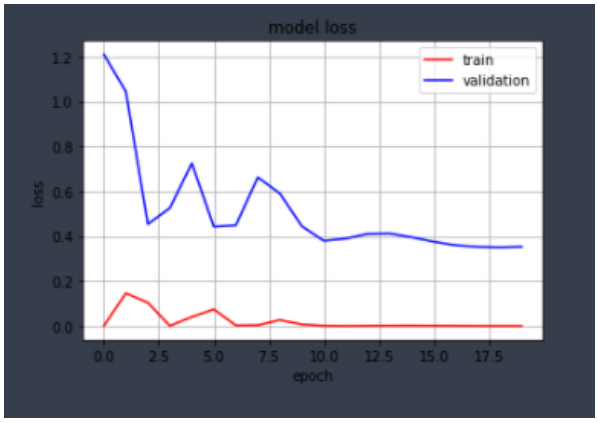
واضحا در طی 20 ایپاک شبکه ما با بچ سایز 64، train میشود و با 20% از داده های test برای Validation استفاده میشود.

نتیجه نمودار دقت :



شکل 2-19 دقت با بچ سایز 64

نتیجه نمودار Loss :



شکل 2-20 Loss با بچ سایز 64

همچنین اطلاعات دقیق به شرح زیر است:

```
2/2 [==============================] - 0s 1000us/step - loss: 0.2773 - accuracy: 0.9286
Test Loss 0.27726852893829346
Test Accuracy 0.9285714030265808
confusion matrix=
 [[20  0]
 [ 3 19]]
```

شکل 2-21 اطلاعات دقیق بچ سایز 64

دقت 0.928 و Loss برابر بت 0.277 شد و ماتریس Confusionقابل مشاهده است.

3- حالت سوم stochastic mini batch based با batch_size=128

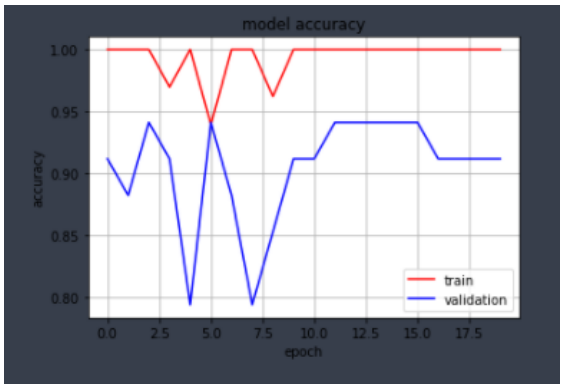
```
stochastic mini batch based Modeling: Size 64

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=64, validation_split=0.2)
```

شکل 2-22 کامپایل مینی بچ 128تایی

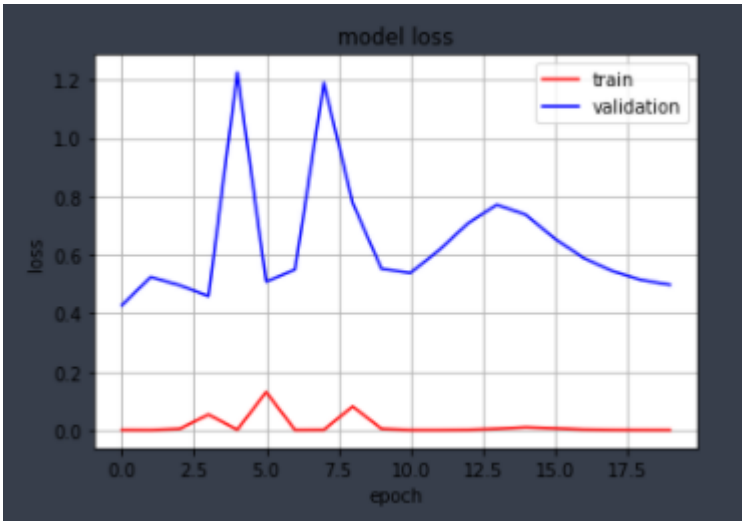
واضحا در طی 20 ایپاک شبکه ما با بچ سایز 128، train میشود و با 20% از داده های test برای Validation استفاده میشود.

نتیجه نمودار دقت :



شکل 2-23 دقت با بچ سایز 128

نتیجه نمودار Loss :



شکل 2-24 Loss با بچ سایز 128

همچنین اطلاعات دقیق به شرح زیر است:

```
2/2 [==============================] - 0s 2ms/step - loss: 0.4686 - accuracy: 0.8571
Test Loss 0.46860232949256897
Test Accuracy 0.8571428656578064
confusion matrix=
 [[20  0]
 [ 6 16]]
```

شکل 2-25 اطلاعات دقیق بچ سایز 128

دقت 0.857 و Loss برابر بت 0.468 شد و ماتریس Confusion قابل مشاهده است.

- نتیجه گیری:

مشاهده میشود که اطلاعات برای پردازش ها به این شکل شده اند:

Batch size 32: Accuracy of 0.982 & Loss 0.247
Batch size 64: Accuracy of 0.928 & Loss 0.277
Batch size 128: Accuracy of 0.857 & Loss 0.468

مشاهده میشود که با افزایش batch size دقت ما کاهش و Loss افزایش داشته است. برای توجیه این امر از سورس زیر استفاده میکنیم:

Resource: Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.

در این سورس تبیین میشود که " در عمل مشاهده شده است که هنگام استفاده از **Large batch** ، تخریب قابل توجهی در کیفیت مدل بوجود می آید ، که توسط توانایی آن برای تعمیم اندازه گیری می شود." همچنین بیان میشود که " فقدان توانایی تعمیم به این دلیل است که روشهای دسته بزرگ تمایل دارند تا به حداقل رساندن شدید عملکرد آموزش همگرا شوند." که دلیل آن ها **مقادیر ویژه مثبت بزرگ در تابع هسین** است.
اما از سوی دیگر، **Batch size های کوچکتر به سمت Flat minimum** ها همگرا میشوند و به **مقادیر ویژه کوچک ماتریس هسین** را در بر میگیرند

نتیجه گیری:
با توجه به صحبت های انجام شده و موارد ذکر شده، از Batch size 32 استفاده میشود.

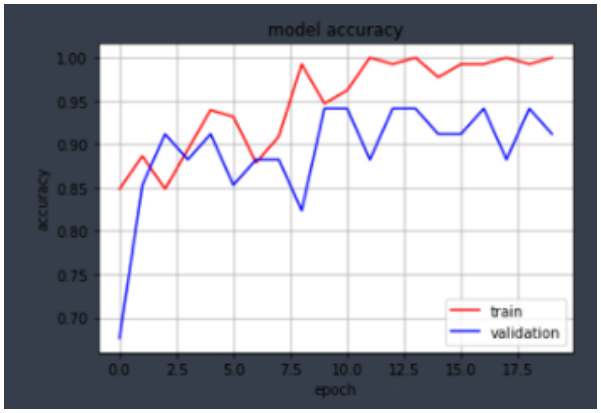
ح: Epoch VS Iterations

در شبکه های عصبی و یادگیری عمیق:

- ایپاک: یک Forward pass و Backward pass از تمام داده های یادگیری با ایپاک مینامیم
- تعداد Iteration : تعداد Passها که هر Pass به تعداد Batch size دارد

پس در واقع در هر ایپاک تمام داده های تمرین مورد پردازش قرار میگیرند و در هر Iteration یک Batch وارد پردازش میشود.

برای یافتن بهترین ایپاک به سادگی میتوان Metric مناسب را انتخاب کرد و Loss را نیز در نظر گرفت. به طور معمول در هر مسئله ایپاک مورد نظر ما میتواند متفاوت باشد چرا که هر مسئله دقت خود را میطلبد. به طور مثال در زیر میتوان اطلاعات حاصل از پردازش سری قبل را دید:



شکل 2-25 دقت مدل MLP

مسلم است که میتواند 10 ایپاک هم مناسب باشد و تا 20 ایپاک پیش نرویم.

افزایش تعداد ایپاک ها ممکن است به Overfitشدن مدل بیانجامد و دیگر برای داده های Test و Validation مطلوب عمل نکند.

به طور کلی ایتریشن ها و ایپاک ها میبایست ما را به نتایج مطلوب مسئله خودمان ببرد، همچنین میبایست دقت نمود که مدل Overfit نشود.

- در Batch تعداد ایپاک و Iteration یسکسان است
- در حالت با Batch size 32 تعداد Iteration ها تقریبا 7 برابر تعداد ایپاک هاست
- در حالت با Batch size 64 تعداد Iteration ها تقریبا 3.5 برابر تعداد ایپاک هاست
- در حالت با Batch size 128 تعداد Iteration ها تقریبا 2 برابر تعداد ایپاک هاست

ط: توابع فعال ساز tanh, sigmoid, reLU

در این بخش سعی میشود همه این سه توابع فعال ساز را برای مدل سازی استفاده کنیم، در مثال های قبل، دیده شد که با تابع فعال ساز reLU به نتایج بسیار خوبی رسیدیم، دلیل آن است که تابع reLU با تمام سادگی خود، بسیار ابزار قدرتمندی بوده و در اکثر Task های که با CNN / MLP میتوان شبکه را توسعه داد از این تابع فعال ساز استفاده میشود.

- تابع فعال ساز reLU:

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(512, activation='relu'))    #Hidden Layer 2
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_19 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_20 (Dense)             (None, 2)                 1026
=================================================================
Total params: 294,914
Trainable params: 294,914
Non-trainable params: 0
_________________________________________________________________
```

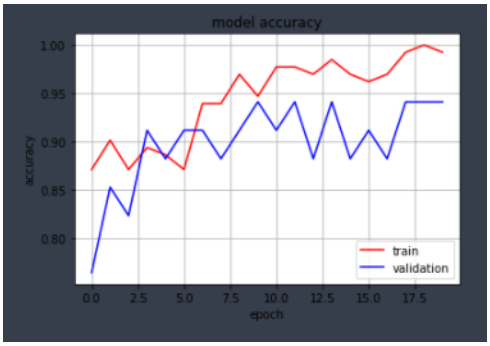
Compiling, Fitting, and Plots

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-27 کامپایل و فیت مدل شبکه classification ReLU

همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

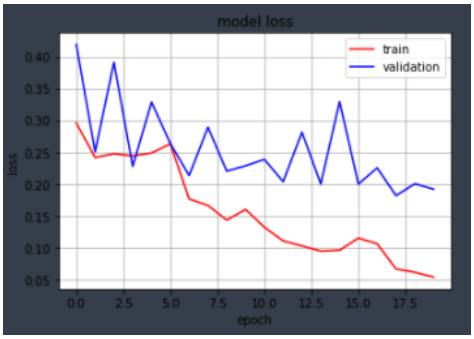
نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:



شکل 2-28 نمودار دقت reLU

مشاهده میشود دقت رو به افزایش میباشد در هر ایپاک و به دقت خیلی خوبی رسیدیم بعد از 17 ایپاک.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-29 نمودار ReLU Loss

همانطور که مشاهده میشود، به نتیجه مطلوب رسیدیم، در 17 ایپاک، تابع Loss ما برای داده های Test & Validation بسیار کاهش پیدا کرده است.

ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 2ms/step - loss: 0.2488 - accuracy: 0.8810
Test Loss 0.2488042563199997
Test Accuracy 0.8809523582458496
confusion matrix=
 [[18  2]
 [ 3 19]]
```

شکل 2-30 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن برای ReLU

- تابع فعال ساز Sigmoid:

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(512, activation='sigmoid')) #Hidden Layer 2
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

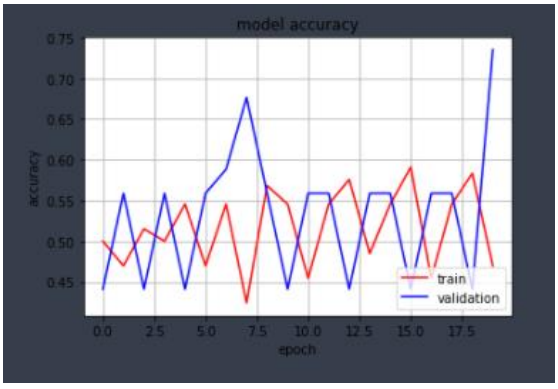
```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_22 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_23 (Dense)             (None, 2)                 1026
=================================================================
Total params: 294,914
Trainable params: 294,914
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-31 کامپایل و فیت مدل شبکه classification sigmoid

همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

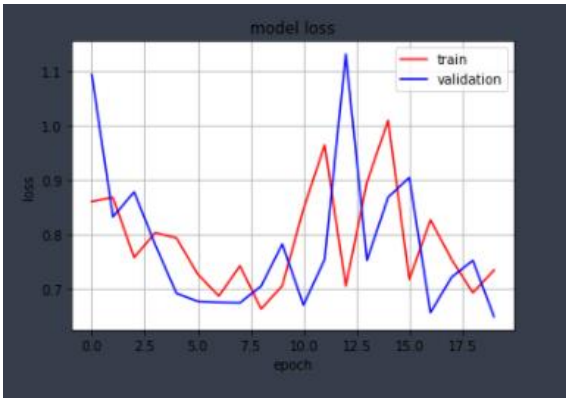
نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:



شکل 2-32 نمودار دقت Sigmoid

مشاهده میشود فرایندی بسیار نوسانی را طی کردیم و از ایپاک 17 مدل Overfit شده است.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-33 نمودار sigmoid Loss

همانطور که مشاهده میشود، در ایپاک 12م تابع Loss برای داده های Validation بسیار زیاد میشود و مراحلی نوسانی را طی میکند

ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 1ms/step - loss: 0.6514 - accuracy: 0.6429
Test Loss 0.651374101638794
Test Accuracy 0.6428571343421936
confusion matrix=
 [[13  7]
 [ 8 14]]
```

شکل 2-34 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن برای sigmoid

میبینیم که دقت بسیار کاهش داشت.

- تابع فعال ساز tanh:

Activation Functions: tanh

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(512, activation='tanh'))    #Hidden Layer 2
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_25 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_26 (Dense)             (None, 2)                 1026
=================================================================
Total params: 294,914
Trainable params: 294,914
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-35 کامپایل و فیت مدل شبکه classification tanh

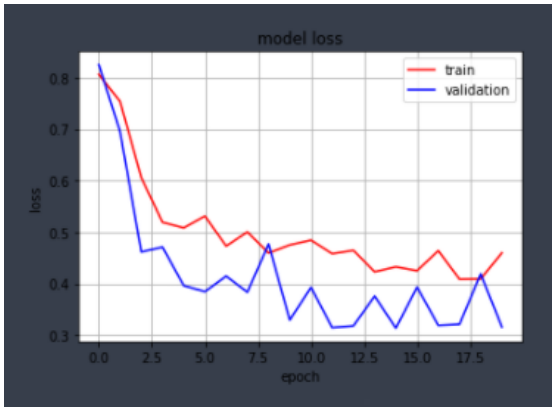
همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:

شکل 2-36 نمودار دقت Sigmoid

مشاهده میشود فرایندی بسیار نوسانی را طی کردیم و از ایپاک 17 مدل Overfit شده است.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-37 نمودار tanh Loss

همانطور که مشاهده میشود، به ازای ایپاک 10 به دقت خوبی رسیده ایم.

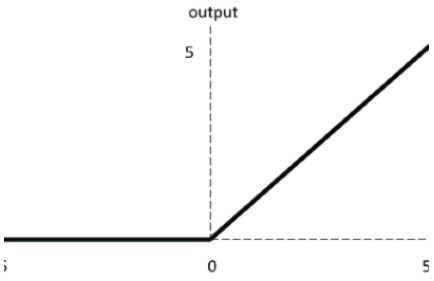
ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 999us/step - loss: 0.4449 - accuracy: 0.7857
Test Loss 0.4449012279510498
Test Accuracy 0.7857142686843872
confusion matrix=
 [[18  2]
 [ 7 15]]
```

شکل 2-38 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن برای tanh

میبینیم که دقت بهتری نسبت به sigmoid و بدتری نسبت به ReLU داشت.

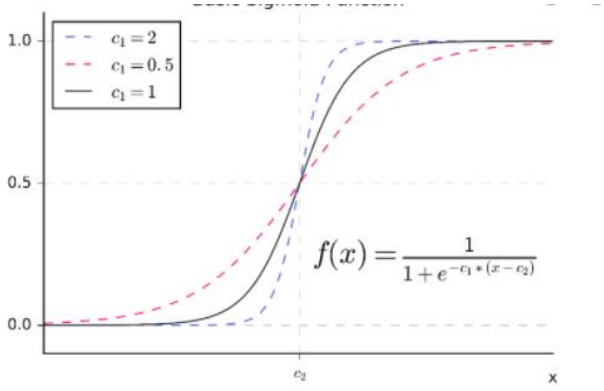
مزایا و معایب ReLU:



شکل 2-39 تابع reLU

- Vanish Gradient نیست +
- بسیار محاسبه راحت تری داردقبل از 0 را 0 و بعد از آن را x برمیگرداند. +
- متمایل به Blow up activation است –
- مشکل Dying ReLU : اگر بسیاری Activation کمتر از صفر باشد خروجی همه آن ها صفر میشود، که میتواند مشکل زا باشد. –

مزایا و معایب Sigmoid:



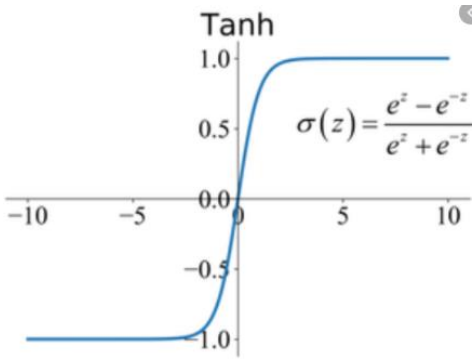
شکل 2-39 تابع Sigmoid

- متمایل به Blow up activation نیست +
- بر خلاف ReLU متاسفانه Vanish Gradient است –

$S'(a) = S(a)(1 - S(a))$. When "a" grows to infinite large
$S'(a) = S(a)(1 - S(a)) = 1 \times (1 - 1) = 0$).

شکل 2-40 عیب Sigmoid

مزایا و معایب tanh:



شکل 2-39 تابع tanh

- Tanh در واقع logistic sigmoid است که عملکرد بهتری دارد+
- قابل انتگرال گیری میباشد و غیر خطی است و میتواند خروجی غیر خطی تولید کند +

- در تمامی نقاط مشتق پذیر است +
- به نسبت reLU کاربرد کمتری دارد –
- تابع یکنواخت است اما مشتق آن یکنواخت نیست –
- معمولا فقط برای کلسیفای کردن 2 کلاس استفاده میشود -

ی: افزودن لایه به شبکه عصبی

در بخش های قبلی 2 لایه مخفی داشتیم، در این بخش یک بار4 لایه مخفی و بار بعدی 6 لایه مخفی گذاشته میشود تا تفاوت را مشاهده کنیم.

- شبکه 4 لایه مخفی:

Neural Network with 4 hidden layers:

```
#Creating a model with 2 hidden Layers
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(2, activation='softmax')) #Last Layer with one output per class
model.summary()
```

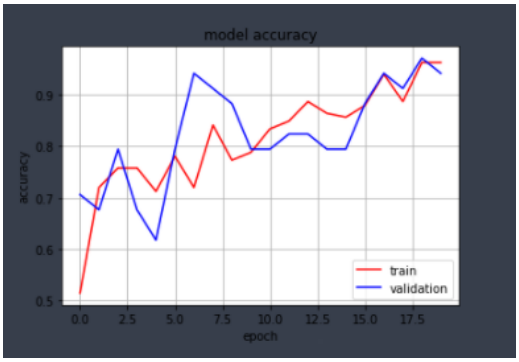
```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_28 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_29 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_30 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_31 (Dense)             (None, 2)                 1026
=================================================================
Total params: 820,226
Trainable params: 820,226
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-40 کامپایل و فیت مدل شبکه classification 4layers

همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

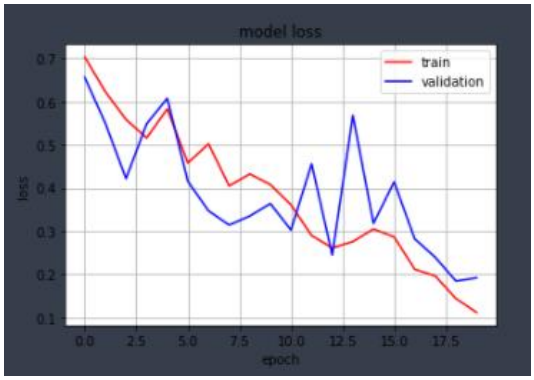
نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:



شکل 2-41 نمودار دقت 4 hidden layers

مشاهده میشود فرایندی بسیارخوبی را طی کرده است و در ایپاک 17م به دقت خوبی رسیده است.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-42 نمودار Loss 4 hidden layes

همانطور که مشاهده میشود، به ازای ایپاک 12 Loss زیاد میشود ولی دوباره به حالت مطلوب برمیگردد

ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 1ms/step - loss: 0.2516 - accuracy: 0.9048
Test Loss 0.25156891345977783
Test Accuracy 0.9047619104385376
confusion matrix=
 [[20  0]
 [ 4 18]]
```

شکل 2-43 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن برای 4 lidden layers

میبینیم که دقت بسیار بالایی داشت (با مقدار 0.904)

- شبکه 6 لایه مخفی:

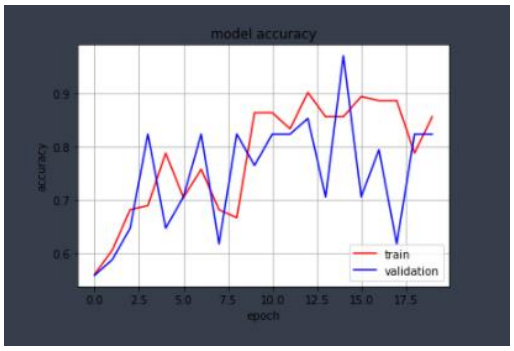
```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_32 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_33 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_34 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_35 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_36 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_37 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_38 (Dense)             (None, 2)                 1026
=================================================================
Total params: 1,345,538
Trainable params: 1,345,538
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-44 کامپایل و فیت مدل شبکه classification 6layers

همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

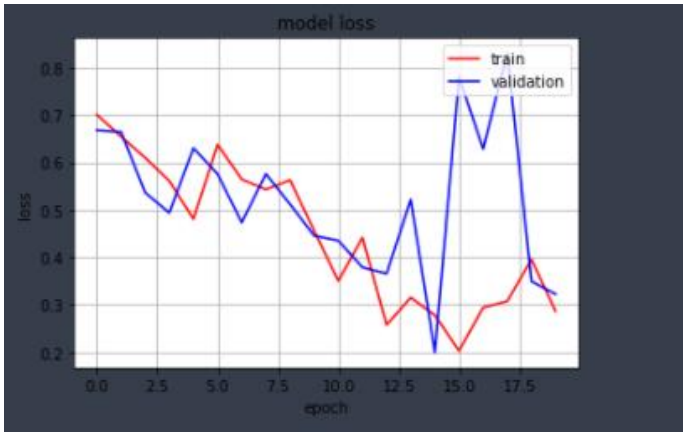
نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:



شکل 2-45 نمودار دقت 6 hidden layers

مشاهده میشود فرایندی خوبی را طی نکرده است دقت بسیار نوسانی است با جلو رفتن ایپاک ها.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-46 نمودار Loss 6 hidden layes

همانطور که مشاهده میشود، به ازای اپیاک 12 Loss زیاد میشود ولی دوباره tendency دارد که به حالت مطلوب برگردد، خطر پیش آمدن overfit در این پیاده سازی به چشم میخورد. در کل افزایش 2 لایه مخفی، به ضرر تمام شد

ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 1ms/step - loss: 0.3998 - accuracy: 0.7857
Test Loss 0.39984261989593506
Test Accuracy 0.7857142686843872
confusion matrix=
 [[20  0]
 [ 9 13]]
```

شکل 2-47 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن برای 6 lidden layers

میبینیم که دقت بسیار کمتری نسبت به حالت قبلی با 4 لایه دارد

نتیجه گیری:

همانطور که انتظار میرفت، تعداد 6 لایه مخفی به ضرر ما تمام شد چرا که با افزایش زیادی تعداد لایه های مخفی، دقت ما کاهش یافت، از جمله مشکلات لایه های زیادی، به خوبی انجام نشدن backpropagation است (یعنی Backpropagated errors خیلی کوچک میشوند و یادگیری صورت نمیگیرد) مشکل Overfit را نیز نباید فراموش کرد که نباید صورت گیرد.

ک: بهترین مدل در سری قبلی

همانطور که دیده شد، با 4 لایه مخفی به دقت خیلی خوبی رسیدیم که استفاده از آن را منطقی میسازد، مشخصات پارامتر آن در زیر مشخص است:

Neural Network with 4 hidden layers:

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 512)               31232
_________________________________________________________________
dense_28 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_29 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_30 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_31 (Dense)             (None, 2)                 1026
=================================================================
Total params: 820,226
Trainable params: 820,226
```

شکل 2-48 کامپایل و فیت مدل شبکه classification 4layers

اطلاعات مربوط به این پیاده سازی نیز در زیر آمده است

```
WARNING:tensorflow:6 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001E8D3F5E1
F0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.func
tion repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_te
nsor_args and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2 [==============================] - 0s 1ms/step - loss: 0.2516 - accuracy: 0.9048
Test Loss 0.25156891345977783
Test Accuracy 0.9047619104385376
confusion matrix=
 [[20  0]
 [ 4 18]]
```

شکل 2-49 اطلاعات فیت مدل شبکه classification 4layers

برای بهبود این شبکه میتوان راهکار های زیر را انجام داد:

- میتوان تعداد نورون های هر لایه را متفاوت کرد، این امر سبب میشود نتیجه آموزش متفاوت شود
- به جای MLP میتوان از CNN استفاده کرد که در مقابل اغتشاشات نیز مقاوم است
- میتوان سبک گزینش Data-set را متفاوت کرد و به نحوی دیگر داده های آزمون ، ارزیابی و یادگیری را انتخاب نمود
- میتوان اپیاک ها را تغییر داد تا به اپیاک بهینه برسیم

ل: نورون ها و لایه های کمتر و تاثیر Overfitting

در این قسمت، سعی میشود با کاهش تعداد نورون ها و تعداد نورون های آن ها، تاثیر Overfittingرا روی این امر بررسی کنیم:

- شبکه 2 لایه مخفی با تعداد نورون 20 در هر لایه:

Neural Network with 2 hidden layers and 20 neurons:

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(20, activation='relu', input_shape=(60,))) #Hidden Layer 1
model.add(Dense(20, activation='relu'))
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

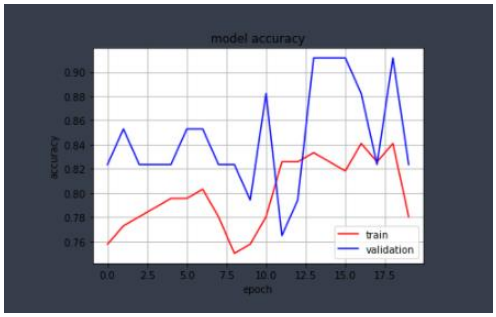
```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_48 (Dense)             (None, 20)                1220
_________________________________________________________________
dense_49 (Dense)             (None, 20)                420
_________________________________________________________________
dense_50 (Dense)             (None, 2)                 42
=================================================================
Total params: 1,682
Trainable params: 1,682
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_split=0.2)
```

شکل 2-40 کامپایل و فیت مدل شبکه classification 2layers

همانگونه که مشاهده میشود، مدل ما در 20 ایپاک فیت میشود و همانگونه که مشخص است، 20% داده های تست را برای validation انتخاب میکند. برای بدست آوردن و ترسیم نمودار accuracy را به عنوان متریک در نظر گرفتیم.

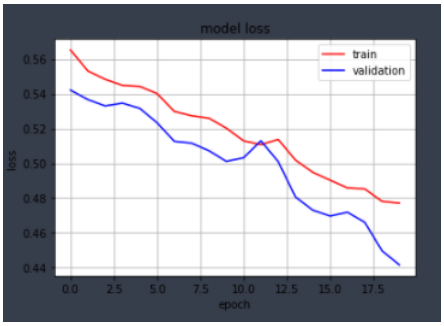
نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:



شکل 2-41 نمودار دقت 2 hidden layers

مشاهده میشود; که در مدل خیلی سریع Overfit میشود.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-42 نمودار Loss hidden layes2

همانطور که مشاهده میشودLoss به صورت کاهشی بوده و عملکرد خوبی داشتیم

ابتدا مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
Test Loss 0.5062450170516968
Test Accuracy 0.7857142686843872
confusion matrix=
 [[19  1]
 [ 8 14]]
```

شکل 2-43 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن برای lidden layers2

همانطور که انتظار میرفت چون تعداد پارامتر های مسئله بسیار کاهش داشته و پیچیدگی مسئله خیلی کمتر شده است، میبایست در ایپاک کمتری به جواب برسیم و بیشتر از آن ما را به سمت Overfitting میبرد.

سوال 3 – Dimension Reduction

کد مربوط به این سوال در آخر فایل های زیر است

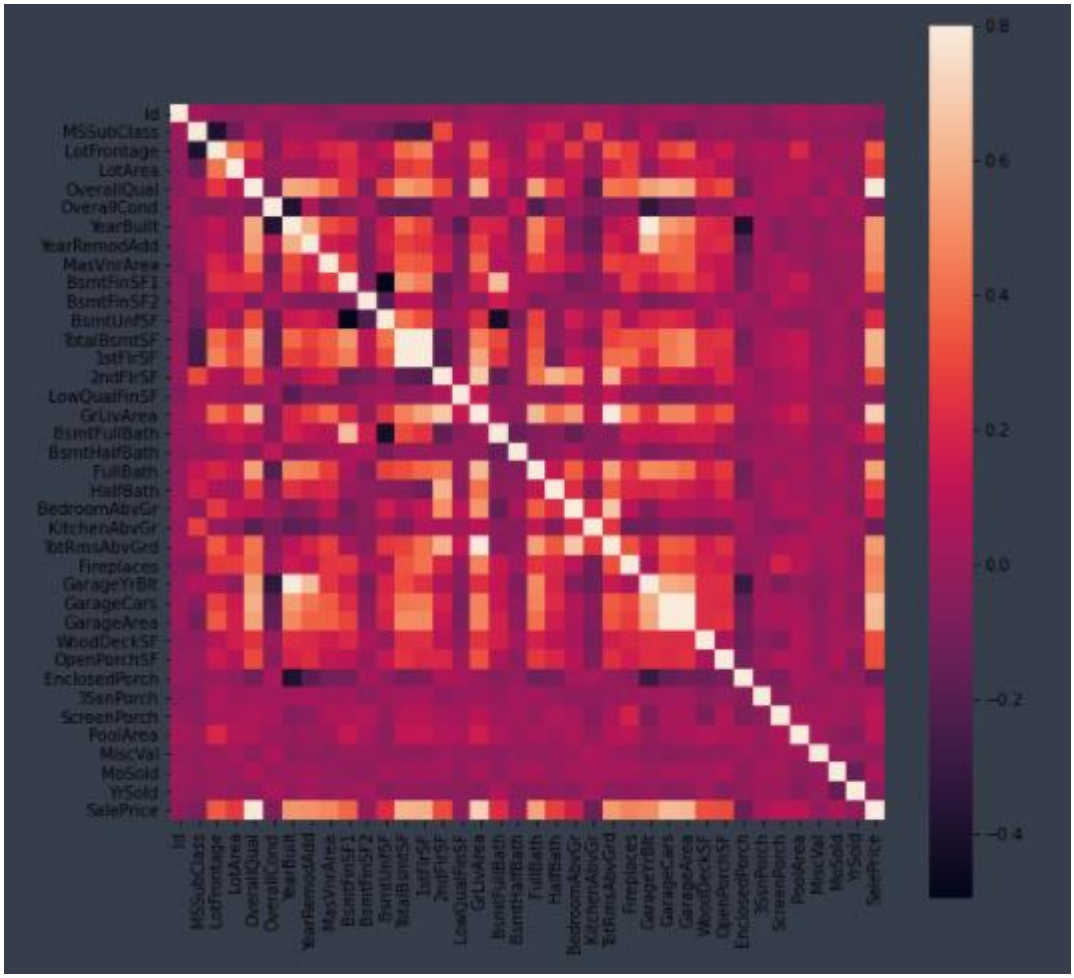
موارد الف-ب-ج در فایل کد NNDL_HW2_Q1_Q3

موارد د-ه-و در فایل کد NNDL_HW2_Q2_Q3

موجود میباشد

الف: ماتریس همبستگی:

در این قسمت برای آنکه بتوان با نام های درست(و نه عددی که به جای کتگوری ها استفاده شد) نمایش ماتریس همبستگی را داشته باشیم، دوباره دیتا ها Load شدند و فقط اینبار نام های ویژگی ها همان های اصلی هستند:



شکل 3-آ-1 ماتریس همبستگی داده ها

ماتریس همبستگی (Correlation Matrix)جدولی است که ضریب همبستگی بین متغییر ها را نشان می دهد. هر متغیر تصادفی مثل X_i در جدول با هریک از متغییرهای دیگر (X_j)همبستگی دارد(یعنی تغییر یکی از متغییرها بر مقدار پارامتر دیگری تاثیر می گذارد.) با این جدول می توانیم بدانیم کدام متغیرها بیشترین همبستگی را با هم دیگر دارند.

به صورت بدیهی میتوان در نظر گرفت که قظر اصلی تماما سفید است چرا که همبستگی هر ویژگی با خودش 1 است

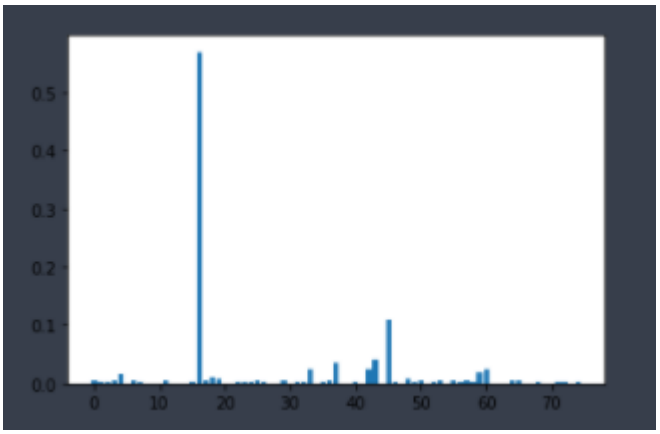
ب: Decision Tree , Linear Regression

در این بخش میخواهیم اهمیت هر ویژگی را به نمایش بگذاریم و دیده شود که کدام ویژگی از بقیه مهمتر بوده و حضورش ضروری تر است.

1- **Linear Regression** :

با استفاده از Regressor استفاده میکنیم به هر کدام از ویژگی ها امتیاز میدهد، نسبت میگیرد و به ما نمایش میدهد که کدام ویژگی ها از همه موثر تر بوده اند:

در تصویر زیر به علت آنکه به صورت عددی با Feature های Dataset برخورد کردیم، Label این Feature ها به ترتیب از 0 تا 70 نام گذاری شده است:

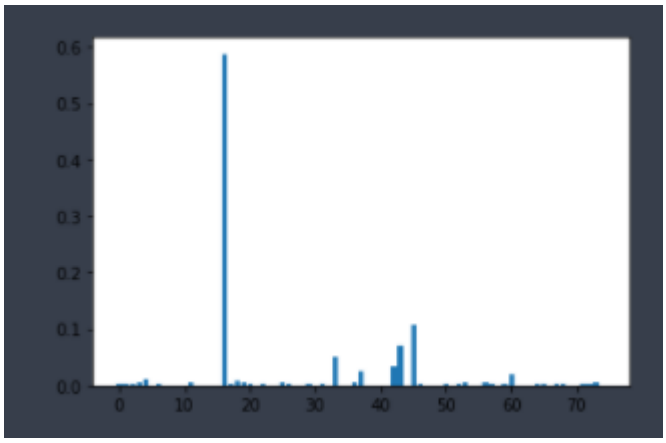


شکل 3ب- 2 Feature Importance Regressor

2- **Decision Tree** :

با استفاده از Decision Tree به هر کدام از ویژگی ها امتیاز میدهد، نسبت میگیرد و به ما نمایش میدهد که کدام ویژگی ها از همه موثر تر بوده اند:

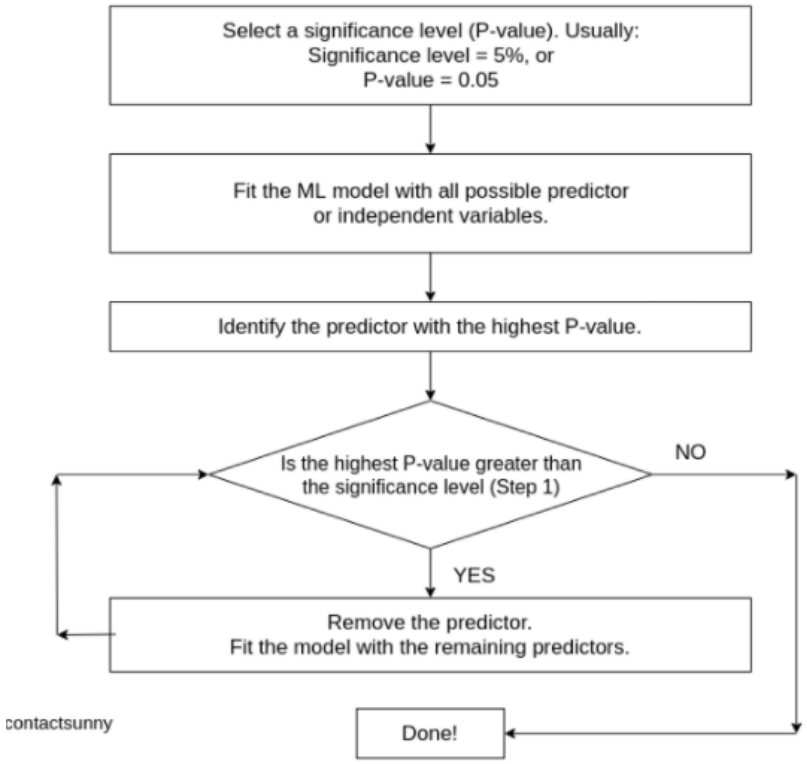
در تصویر زیر به علت آنکه به صورت عددی با Feature های Dataset برخورد کردیم، Label این Feature ها به ترتیب از 0 تا 70 نام گذاری شده است:



شکل 3ب- 2 Feature Importance Decision Tree

ج: Backward Elimination

فلوچارت این الگوریتم به مانند زیر است:



شکل 3ج-1 فلوچارت Backward Elimination

پس همانند الگوریتم عمل میکنیم و یک Level اولیه 0.5% ی در نظر میگیریم، با تمام ویژگی هایی که داریم، مدلی را طراحی میکنیم، اینجا از مدل Linear Regression استفاده شده است.

ویژگی با بالاترین لول را در نظر گیریم، اگر بیشتر از مقدار اولیه ست شده بود آن را حذف میکنیم و سپس دوباره امتحان میکنیم، اگر همه از آن کمتر بودند، الگوریتم به ایان میرسد

OLS Regression Results

Dep. Variable:	75	R-squared:	0.838
Model:	OLS	Adj. R-squared:	0.827
Method:	Least Squares	F-statistic:	77.43
Date:	Tue, 27 Apr 2021	Prob (F-statistic):	0.00
Time:	23:16:50	Log-Likelihood:	1971.2
No. Observations:	1168	AIC:	-3794.
Df Residuals:	1094	BIC:	-3420.
Df Model:	73		
Covariance Type:	nonrobust		

3ج-2 بعد از فیت کردن مدل

بعد از انجام عملیات فوق، به نتیجه رو به رو برای داده های Optimal رسیدیم

x_train_new

	1	3	4	5	7	11	13	16	17	18	21	25	26	29	30
171	0.000000	0.410959	0.142420	1.0	1.0	0.500000	0.285714	0.555556	0.500	0.637681	0.142857	0.070000	1.000000	1.00	0.5
881	0.176471	0.078767	0.058230	1.0	1.0	0.958333	0.285714	0.666667	0.500	0.855072	0.142857	0.073125	0.666667	0.75	1.0
1437	0.000000	0.256849	0.052088	1.0	1.0	0.666667	0.285714	0.777778	0.500	0.985507	0.142857	0.266250	0.000000	0.25	1.0
627	0.352941	0.202055	0.038795	1.0	1.0	0.500000	0.285714	0.555556	0.625	0.601449	0.142857	0.102500	1.000000	1.00	1.0
899	0.000000	0.150685	0.026610	1.0	1.0	0.791667	0.285714	0.444444	0.750	0.644928	0.142857	0.000000	1.000000	1.00	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1050	0.000000	0.178082	0.035958	1.0	1.0	0.333333	0.285714	0.666667	0.500	0.978261	0.142857	0.000000	0.666667	0.75	1.0
1171	0.000000	0.188356	0.036551	1.0	1.0	0.500000	0.285714	0.555556	0.625	0.623188	0.142857	0.000000	1.000000	1.00	1.0
353	0.058824	0.133562	0.033747	1.0	1.0	0.708333	0.285714	0.555556	0.875	0.405797	0.142857	0.000000	1.000000	1.00	1.0
78	0.411765	0.174658	0.044301	1.0	1.0	0.791667	0.285714	0.333333	0.500	0.695652	0.142857	0.000000	1.000000	1.00	1.0
1085	0.382353	0.178082	0.036313	1.0	1.0	0.833333	0.285714	0.555556	0.625	0.869565	0.142857	0.000000	1.000000	0.75	1.0

1168 rows × 34 columns

شکل 3ج 3 داده های اپتیمال

همانگونه که مشاهده میشود، فقط 34 ویژگی از 75 ویژگی باقی مانده است.

```
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, input_dim = 34, activation='sigmoid'))
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

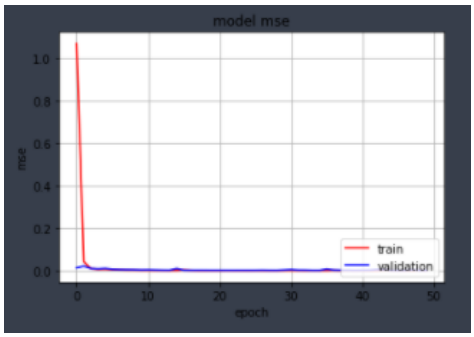
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               17920
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 513
=================================================================
Total params: 281,089
Trainable params: 281,089
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history = model.fit(x_train_new, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

شکل 3ج4 مدل کردن شبکه

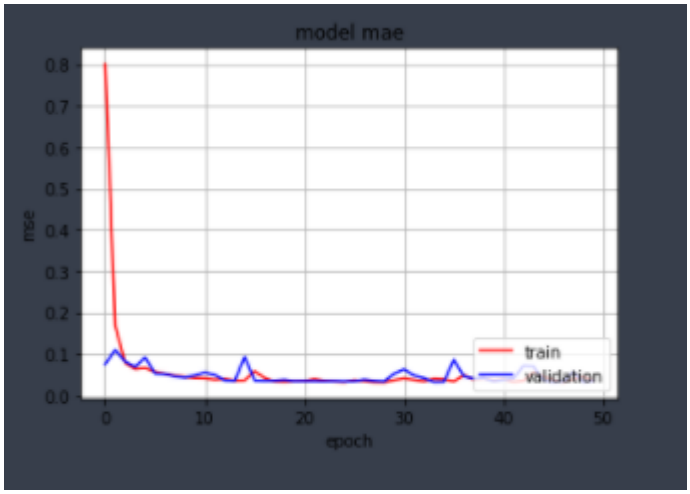
حال نوبت آن است که مدلمان را با این داده ها فیت کنیم.

برای متریک mse برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



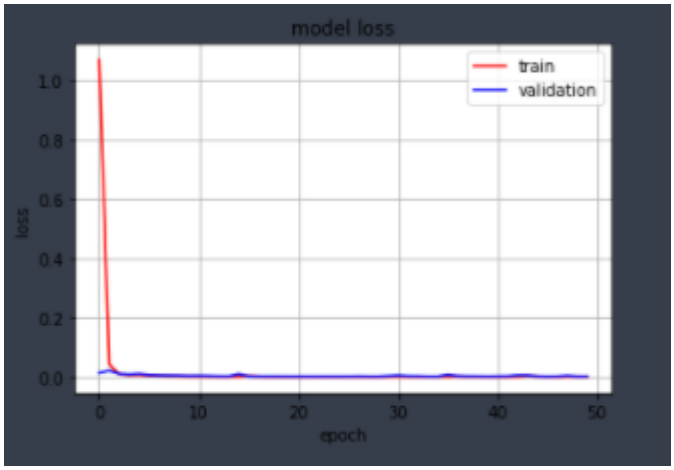
شکل 1-26 دو لایه مخفی relu: mse metric

برای متریک mae برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



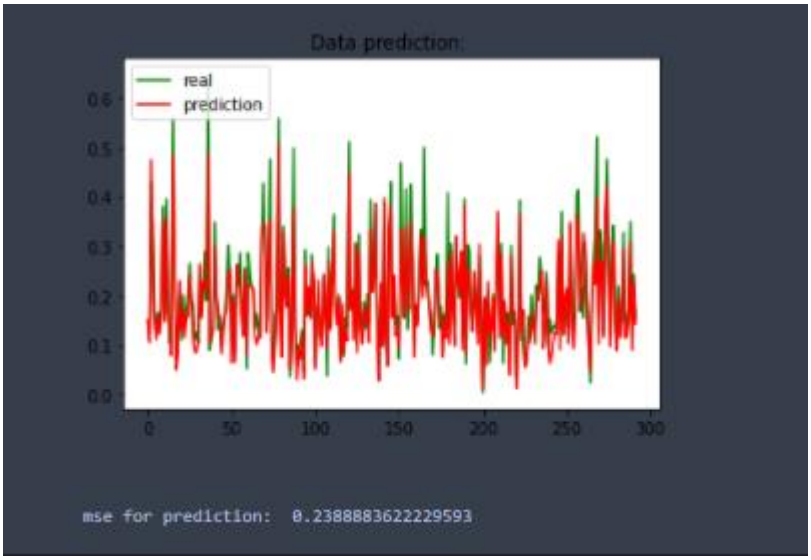
شکل 1-27 دو لایه مخفی relu: mae metric

برای تابع Loss برای داده های train و Evaluation پلات با 50 ایپاک در تصویر پایین آورده شده است:



شکل 1-28 دو لایه مخفی relu: mae metric

در نهایت بعد از تخمین، داریم که برای داده های تست میتوان به صورت دستی نیز با داده های پیشبینی شده mse آن ها را محاسبه کرد و یک شکل کلی از آن ها کشید



شکل 1-29 دو لایه مخفی relu: prediction

نتیجه گیری:

- همانطور که مشخص است MSE بین داده های تخمینی و حقیقی تست، برابر با 0.238 شد
- تعداد اپیاک های بهینه میتواند 2 باشد

د: Dimension reduction: PCA

در این سوال هدف آن است که با استفاده از Principle component algorithm ابعاد داده های خود را کم کنیم.

دلیل این امر: داده های زیاد یعنی پردازشش زیاد و این امر به افزایش زمان برنامه ما می انجامد، پس سعی میکنیم از این الگوریتم استفاده کرده و ابعاد مسئله خود را کاهش داده تا نه تنها دقت خوبی داشته باشیم بلکه در زمان و فضا نیز صرفه جویی کرده باشیم

ابتدا بهترین پیاده سازی خود را که 4 لایه ای بود را انتخاب میکنیم و روی آن **PCA** میزنیم:

```
import time
#Creating a model with 2 hidden layers
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(19,))) #Hidden Layer 1
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(2, activation='softmax')) #Last layer with one output per class
model.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_74 (Dense)             (None, 512)               10240
_________________________________________________________________
dense_75 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_76 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_77 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_78 (Dense)             (None, 2)                 1026
=================================================================
Total params: 799,234
Trainable params: 799,234
```

شکل 3د-1 مدل انجام شده برای pca

برای استفاده از این روش، از کتاب خانه SKLearn پایتون PCA را import میکنیم و ابعاد را کاهش میدهیم.
برای اینکار تا 0.96% اطلاعات را نگه میداریم و نتیجه را نگه میداریم:

```
Y_train = np_utils.to_categorical(y_train)
Y_test = np_utils.to_categorical(y_test)
Y_train_pca = Y_train
Y_test_pca = Y_test

time_begin = time.time()
pca = PCA(.96)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
time_end = time.time()

X_train_pca_df = pd.DataFrame(X_train_pca[0:,0:])
X_test_pca_df = pd.DataFrame(X_test_pca[0:,0:])
pca.n_components_
```

```
19
```

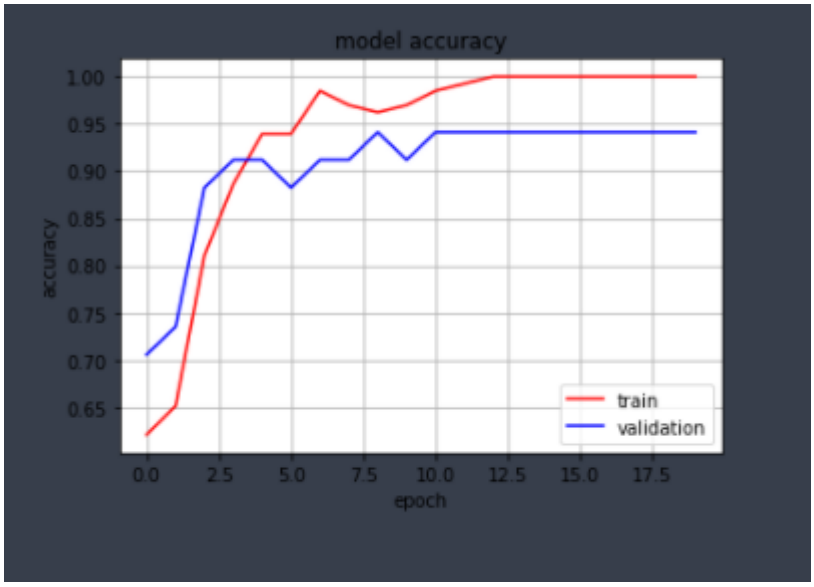
```
print(time_end - time_begin)
```

```
0.012992143630981445
```

شکل 3د-2 PCA Implementation

همانطور که در شکل بالا مشخص میباشد، ابعاد ما از 60 به 19 ماهش پیدا کرده و فقط 0.012 ثانیه اینکار به طول انجامید.

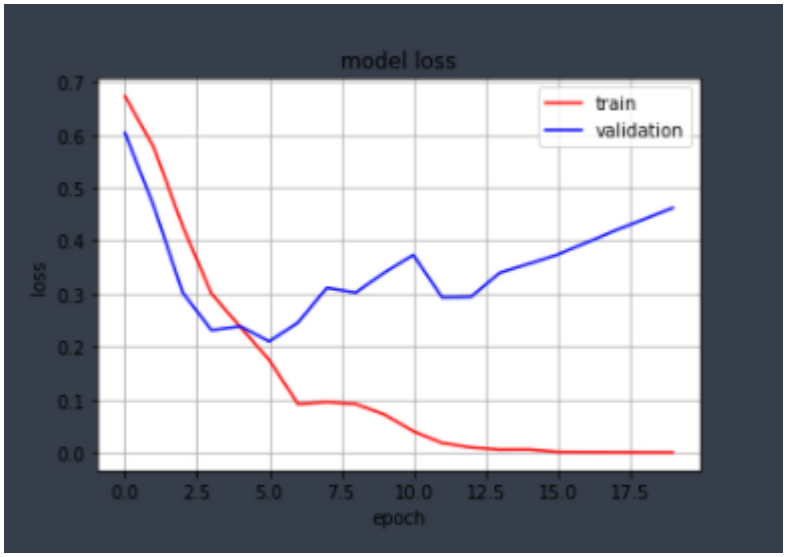
نمودار دفت در هر اپاک در شکل در زیر قابل دیدن است:



شکل 3-1 نمودار دقت pca

مشاهده میشود که در مدل خیلی سریع تر به دقت خوب میرسد , .

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 23-2 نمودار pca Loss

همانطور که مشاهده میشودLoss به صورت خیلی سریع کاهش می یابد و بعد Overfit میشود

مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
Test Loss 0.5081843733787537
Test Accuracy 0.8809523582458496
confusion matrix=
 [[20  0]
 [ 5 17]]
```

شکل 2-43 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن pca

همانطور که انتظار میرفت بسیار سریع تر به جواب رسیدیم و میتوان ایپاک 10 را خاتمه الگوریتم نام گذاری کنیم.

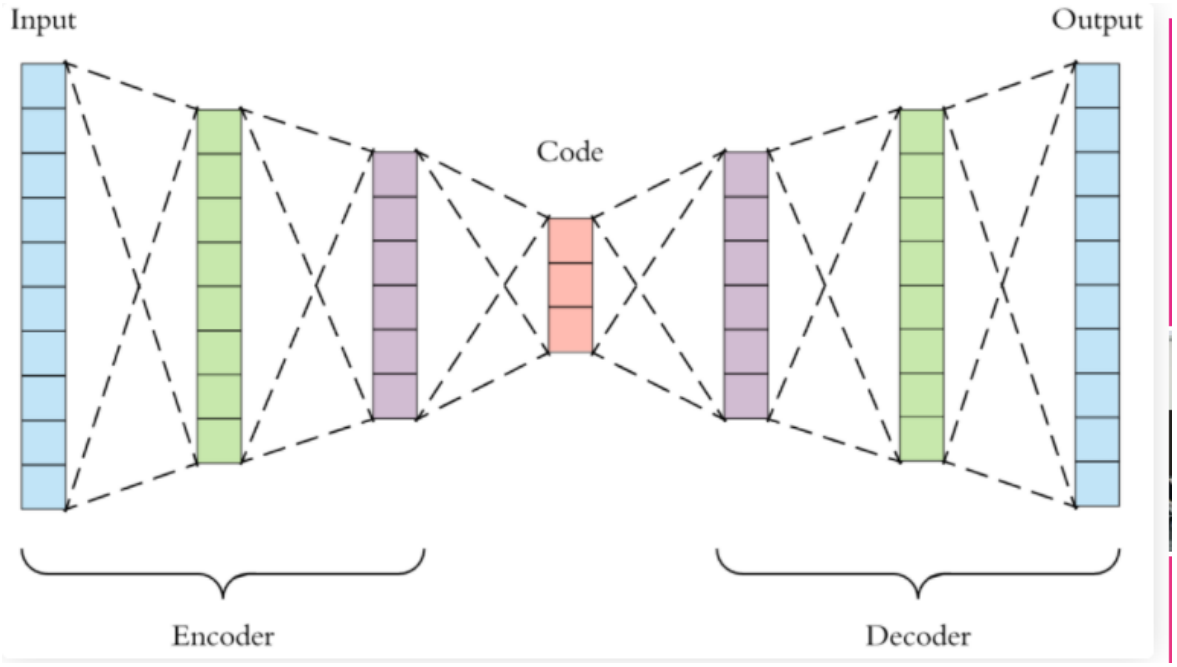
۵: Dimension reduction: Auto-Encoder

در این سوال هدف آن است که با استفاده از Auto Encoder ابعاد داده های خود را کم کنیم.

دلیل این امر: داده های زیاد یعنی پردازشش زیاد و این امر به افزایش زمان برنامه ما می انجامد، پس سعی میکنیم از این الگوریتم استفاده کرده و ابعاد مسئله خود را کاهش داده تا نه تنها دقت خوبی داشته باشیم بلکه در زمان و فضا نیز صرفه جویی کرده باشیم

بهترین پیاده سازی خود را که 4 لایه ای بود را انتخاب میکنیم و روی آن Auto Encoder میزنیم.

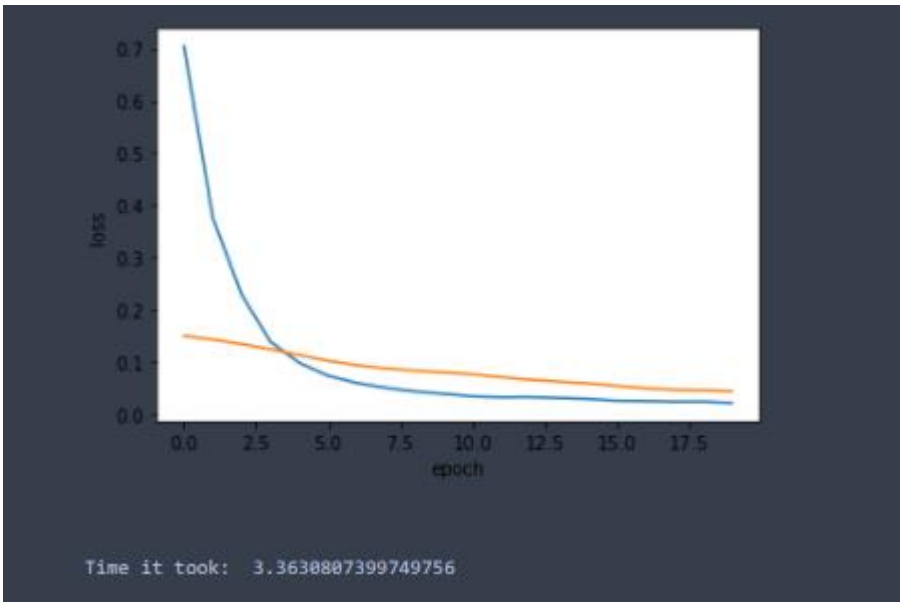
نکته مهم، تفهیم موضوع Auto Encoder است که مانند شکل زیر عمل میکنند:



شکل 3و 1 ساختار یک AutoEncoder

عملیات Auto Encoder به این صورت است که یک قسمت انکودر دارد و یک قسمت دیکودر دارد و سعی میکند لایه به لایه ابعاد را کوچکتر کند و ابعاد کوچکتر را دیکود کند، تا ببیند با داده های اصلی Correlation به چه صورت است.اگر Loss آن قابل قبول بود، از آن تعداد ابعاد استفاده میشود.

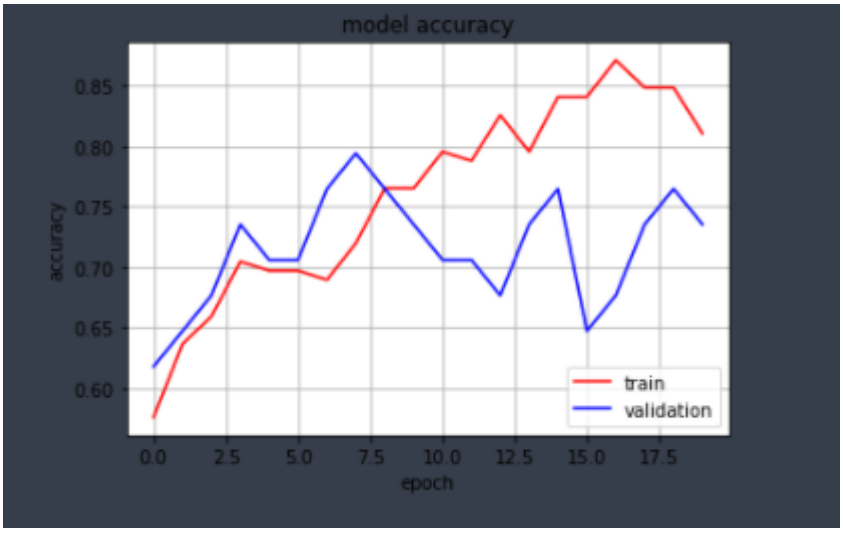
اینجا از 3 لایه انکودر و 3 لایه دیکودر استفاده شده است و کاهش ابعاد تا 15 بعد در نظر گرفته شده ، بعد از 20 ایپاک نتیجه قابل ترسیم است:



تصویر 3ه 2 خطای Loss

هم خطا مطلوب شده و هم زمان به دست آمده است که زمان آن اصلا مطلوب نبوده و 3.36 ثانیه شده است.

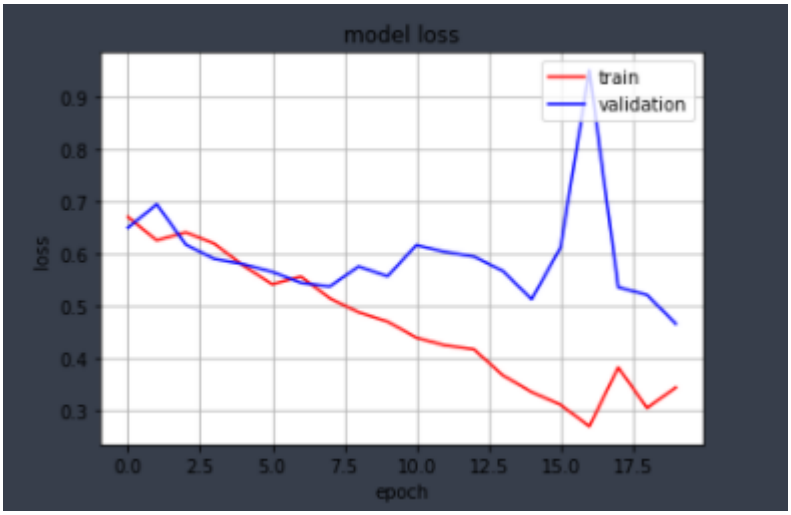
نمودار دقت در هر ایپاک در شکل در زیر قابل دیدن است:



شکل 3ه 1 نمودار دقت Auto-encoder

مشاهده میشود که در مدل به دقت خیلی خوبی نرسیدیم.

نمودار Loss در هر ایپاک در شکل زیر قابل دیدن میباشد:



شکل 2-42 نمودار pca Loss

همانطور که مشاهده میشودLoss به صورت خیلی سریع کاهش می یابد و بعد Overfit میشود

مقادیر خطا و دقتبرای داده ه نمایش داده میشود:

```
2/2 [==============================] - 0s 1ms/step - loss: 0.4280 - accuracy: 0.7857
Test Loss 0.42802247405052185
Test Accuracy 0.7857142686843872
confusion matrix=
 [[16  4]
 [ 5 17]]
```

شکل 2-43 مقادیر دقیق خطا و دقت و ماتریس کانفیوژن Auto-encoder

مقدار Loss برابر 0.428 و Accuracyبرابر با 0.785 میباشد.

ه: نتیجه گیری:

	دقت داده تست	خطای داده تست	زمان
بهترین شبکه سوال ۲	0.90	0.25	2.875
AutoEncoder	0.785	0.428	(2.875)/2 + 3.36 = 4.8
PCA	0.880	0.35	(2.875)/2 + 0.012 = 1.445

پر واضح است که استفاده از PCA برای ما بسیار مطلوب تر است چرا که PCA نه تنها از همه کمتر زمان برد بلکه دقت فوق العاده ای هم دارد، همچنین در PCA داده ها به 19 بعد کاهش یافته بودند که این خود مزیتی در حافظه هم محسوب میشود