|  |  |
| --- | --- |
| نام و نام خانوادگی | حمید رضا کاشانی |
| شماره دانشجویی | 810100441 |
| تاریخ ارسال گزارش | 19/1/1401 |

|  |  |  |
| --- | --- | --- |
|  | به نام خدا |  |
| **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر**  **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین سری 2** | | |

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

[سوال 1 –(Classification( MLP 1](#_Toc100374500)

[سوال ۲ – MLP (Regression) 18](#_Toc100374501)

[سوال 3 – کاهش ابعاد 31](#_Toc100374502)

# سوال 1 –(Classification( MLP

الف)روش های متفاوت تقسیم داده ها در ماشین لرنینگ:(منبع سایت howsam.org)

Resubstitution

ساده‌ترین و نه بهترین روش برای تعیین خطای مدل در الگوریتم‌های یادگیری ماشین، روش Resubstitution است. در این روش تمامی داده‌ها برای آموزش مدل استفاده می‌شوند. تفاوت خروجی مدل و خروجی واقعی به عنوان خطا در نظر گرفته شده و سپس میانگین خطای همه داده‌ها به عنوان خطای کل در نظر گرفته می‌شود. به این خطا، خطای Resubstitution گفته می‌شود.

Holdout

در این روش، داده‌ها به دو دسته train و test تقسیم می‌شوند. این تقسیم می‌تواند به صورت 40/60 ، 30/70 یا 20/80 باشد. بنابراین مدل مورد نظر روی داده‌های train آموزش دیده و روی داده‌های test مورد ارزیابی قرار می‌گیرد. به این روش، اعتبارسنجی Holdout گفته می‌شود. در روش Holdout، اگر کلاس‌های مختلف در هر گروه test یا train توزیع یکسانی نداشته باشند، مدل، درست آموزش نخواهد دید. از این جهت کلاس‌ها باید توزیع یکسانی در هر دو گروه train و test داشته باشند. به این پروسه، stratification گفته می‌شود.

K-Fold Cross-Validation

در این روش، داده‌ها به k بخش تبدیل می‌شوند. هرکدام از داده‌ها به صورت تصادفی در یکی از این k بخش قرار می‌گیرند. در این روش، آموزش و تست k بار تکرار می‌شوند. در هر تکرار k-1 بخش به عنوان داده train و یکی به عنوان test در نظر گرفته می‌شود. خطای مدل برابر با میانگین مدل در k تکرار است.

(Leave-One-Out Cross-Validation (LOOCV

در این تکنیک، از تمامی داده‌ها به جز یک داده برای آموزش و از داده باقی مانده برای تست مدل استفاده می‌شود. این فرآیند N بار تکرار می‌شود که N تعداد داده‌ها را نشان می‌دهد. مزیت این روش این است که از تمامی داده‌ها برای آموزش و تست مدل استفاده خواهد شد. نرخ خطای مدل در این روش برابر با میانگین نرخ خطا در هر تکرار است. شکل زیر تکنیک LOOCV را نشان می‌دهد.

Random Sub-sampling

در این تکنیک تعدادی از داده‌ها به صورت تصادفی انتخاب شده و داده‌های تست را تشکیل می‌دهند. باقیمانده داده‌ها نیز برای آموزش مورد استفاده قرار می‌گیرند. نرخ خطای مدل در این روش نیز برابر با میانگین نرخ خطا در هر تکرار است. شکل زیر تکنیک Random Subsampling را نشان می‌دهد.

Bootstrapping

در این تکنیک، داده‌های آموزش به صورت تصادفی و با استفاده از جایگذاری انتخاب می‌شوند. نمونه‌هایی که انتخاب نشده‌اند نیز برای تست مورد استفاده قرار می‌گیرد. در این روش، بر خلاف روش k-fold، تعداد نمونه‌های انتخاب شده در هر تکرار متفاوت است. نرخ خطای مدل در این روش نیز برابر با میانگین نرخ خطا در هر تکرار است. شکل زیر تکنیک Bootstrapping را نشان می‌دهد.

ما ابتدا داده ها را به دو دسته test و train تقسیم می کنیم و سپس داده های train را به به صورت رندوم تقسیم به دو قسمت train و validation تبدیل میکنیم و روی داده های train مدل را آموزش می دهیم

داده های ورودی ما خودش به صورت دو دسته train وtest به داده می شود و در هنگام تمرین دادن مدل ها را با 0.2 validation فیت می کنیم.

50000داده برای تمرین و 10000داده برای تست داریم

ب)

**ماتریس آشفتگی :**

یک ماتریس در هم ریختگی که به عنوان ماتریس خطا نیز شناخته می‌شود، یک جدول خلاصه‌شده برای ارزیابی عملکرد یک مدل طبقه‌بندی است. تعداد پیش‌بینی‌های صحیح و نادرست با مقادیر شمارش خلاصه می‌شوند و بر اساس هر کلاس شکسته می‌شوند.

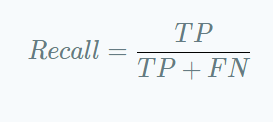
در زیر تصویری از ساختار یک ماتریس در هم ریختگی ۲×۲ آورده شده‌است. برای مثال، اجازه دهید فرض کنیم که ده نمونه وجود داشتند که در آن‌ها یک مدل طبقه‌بندی «بله» را پیش‌بینی می‌کرد که در آن ارزش واقعی نیز «بله» بود. سپس شماره ده در گوشه سمت چپ بالا در ربع مثبت حقیقی قرار می‌گرفت. این منجر به برخی اصطلاحات کلیدی می‌شود:

ساختار ماتریس در هم ریختگی ۲×۲
ساختار ماتریس در هم ریختگی ۲×۲

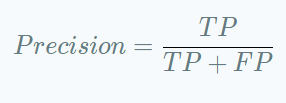
* مثبت (P): مشاهده مثبت است. (مثلا: سگ هست)
* منفی (N): مشاهده مثبت نیست. (مثلا: سگ نیست)
* مثبت واقعی (TP) : نتایج زمانی حاصل می‌شوند که مدل به درستی کلاس مثبت را پیش‌بینی می‌کند.
* منفی واقعی (TN) : در جایی که مدل به درستی کلاس منفی را پیش‌بینی می‌کند، نتایج به دست می‌آیند.
* مثبت کاذب (FP) : که همچنین خطای نوع ۱ نیز نامیده می‌شود، نتیجه‌ای که در آن مدل به اشتباه کلاس مثبت را هنگامی که در واقع منفی است، پیش‌بینی می‌کند.
* منفی کاذب (FN) : که همچنین یک خطای نوع ۲ نیز نامیده می‌شود، نتیجه‌ای که در آن مدل به طور نادرست کلاس منفی را وقتی که واقعا مثبت است پیش‌بینی می‌کند.

**معیار Recall یا یادآوری:**

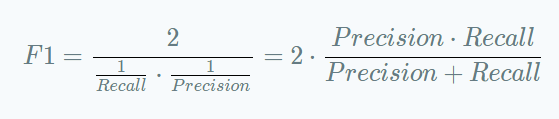
حداکثر مقدار این معیار یک ویا ۱۰۰ درصد و حداقل مقدار آن صفر است و هرچه مواردی که ما انتظار داشتیم پیش بینی شوند ولی برنامه پیش بینی نکرده‌است که به آن False Negative می‌گوییم نسبت به پیش بینی‌های درست یا True Positive بیشتر باشد مقدار Recall کمتر خواهد شد.

****

**معیار Precision یا دقت:**

حداکثر مقدار این معیار یک ویا ۱۰۰ درصد و حداقل مقدار آن صفر است و هرچه مواردی که برنامه‌ به غلط پیش بینی کرده است که به آن False Positive می‌گوییم نسبت به پیش بینی‌های درست یا True Positive بیشتر باشد مقدار Precision کمتر خواهد شد.

**معیار f1-score:**

زمانی که می‌خواهید معیار ارزیابی شما میانگینی از دو مورد قبلی باشد یعنی همان Recall یا Precision می‌توانید از میانگین هارمونیک این دو معیار استفاده کنید که به آن معیار f1-score می‌گویند.

تابع برای سیاه سفید کردن عکس:

def rgb\_to\_gray(img):

  R, G, B = img[:,:,:,0], img[:,:,:,1], img[:,:,:,2]

  imgGray = 0.2989 \* R + 0.5870 \* G + 0.1140 \* B

  return imgGray

پیش پردازش داده ها:

from keras.datasets import cifar10

(x\_train\_rgb,y\_train),(x\_test\_rgb,y\_test)=cifar10.load\_data()

x\_train\_gray =rgb\_to\_gray(x\_train\_rgb)

x\_test\_gray = rgb\_to\_gray(x\_test\_rgb)

x\_test\_gray = x\_test\_gray/255

x\_train\_gray = x\_train\_gray/255

x\_train = x\_train\_gray.reshape(x\_train\_gray.shape[0],32\*32)

x\_test = x\_test\_gray.reshape(x\_test\_gray.shape[0],32\*32)

y\_train = tf.keras.utils.to\_categorical(y\_train,10)

y\_test = tf.keras.utils.to\_categorical(y\_test,10)

ترسیم 10 عکس از داده ها:

figure = plt.figure(figsize=(20, 10))

for i in range(10):

  photo= np.random.choice(x\_train\_rgb.shape[0], size=15, replace=False)

  ax = figure.add\_subplot(2, 5, i+1)

  ax.imshow(np.squeeze(x\_train\_rgb[photo[i]]))

ت)3 شبکه با لایه های متفاوت

شبکه اول: دو لایه 32 نورون ، زمان ترین 2:33

model1 = Sequential([

    layers.Input(shape=(32\*32)),

    layers.Dense(32,activation='relu'),

    layers.Dense(32,activation='relu'),

    #layers.Dense(10,activation='relu'),

    #layers.Dense(3,activation='relu'),

    layers.Dense(10,activation='softmax')

])

model1.compile(

    optimizer='adam',

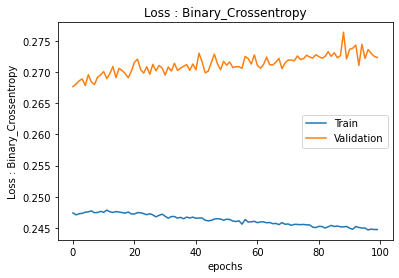
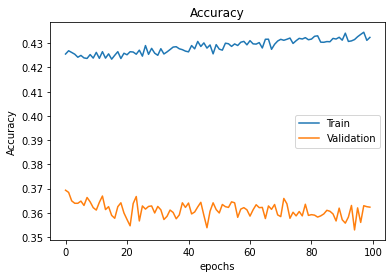
    loss='BinaryCrossentropy',

    metrics=['accuracy']

    #metrics=['mae','accuracy']

) train\_model1 = model1.fit(x\_train,y\_train,epochs=100, validation\_split=0.2,batch\_size=128)

نمودار ها:

و خطا و دقت به ترتیب:

0.2759501338005066

0.3573000133037567

و همینطور این 4 پارامتر precision\_recall\_fscore\_support یه ترتیب:

[0.42312579 0.39607201 0.29617486 0.27393617 0.27948718 0.37422553

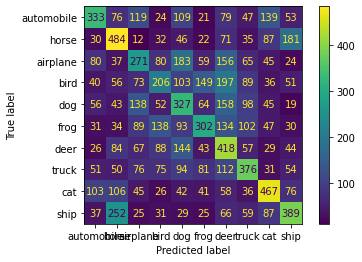
0.28847481 0.39004149 0.46100691 0.42236699]

[0.333 0.484 0.271 0.206 0.327 0.302 0.418 0.376 0.467 0.389]

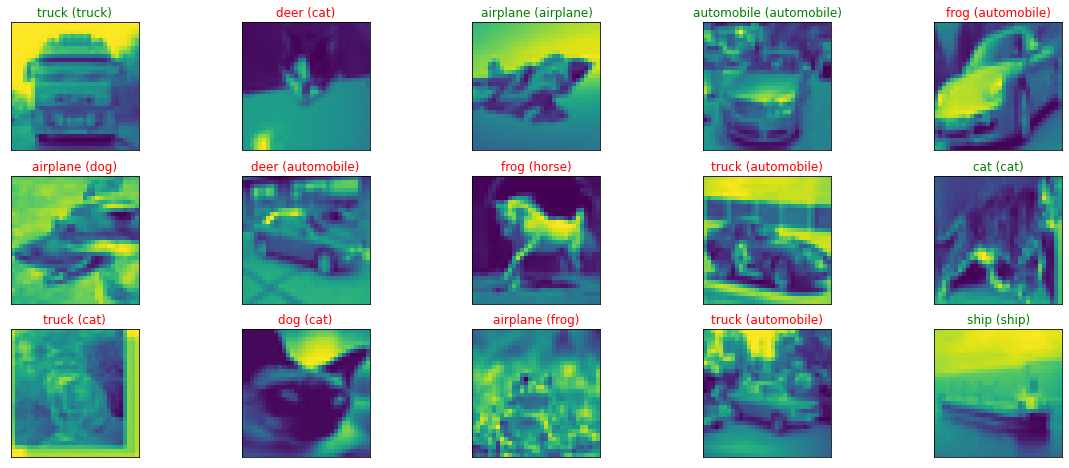
[0.37269166 0.43564356 0.28302872 0.23515982 0.30138249 0.33425567

0.34136382 0.38289206 0.4639841 0.4049974 ]

[1000 1000 1000 1000 1000 1000 1000 1000 1000 1000]

ماتریس آشفتگی:

و همینطور نتیجه رندوم از این شبکه:



شبکه دوم: یه لایه 32 و یه لایه 10 با زمان ترین 02:22

model2 = Sequential([

    layers.Input(shape=(32\*32)),

    layers.Dense(32,activation='relu'),

    layers.Dense(10,activation='relu'),

    #layers.Dense(10,activation='relu'),

    #layers.Dense(3,activation='relu'),

    layers.Dense(10,activation='softmax')

])

model2.compile(

    optimizer='adam',

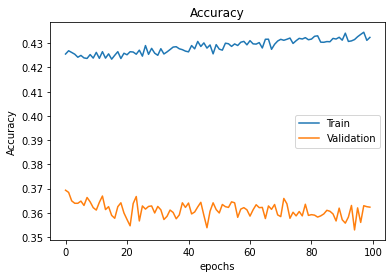
    loss='BinaryCrossentropy',

    metrics=['accuracy']

    #metrics=['mae','accuracy']

)

train\_model2 = model2.fit(x\_train,y\_train,epochs=100, validation\_split=0.2,batch\_size=128)



خطا و دقت به ترتیب:

0.2759501338005066

0.3573000133037567

همینطور این 4 پارامتر precision\_recall\_fscore\_support یه ترتیب:

[0.41275168 0.39226033 0.27754678 0.25 0.2512987 0.3580786

0.32324219 0.34016973 0.4073107 0.42448513]

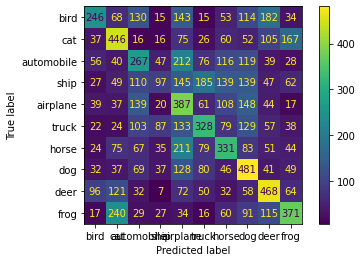
[0.246 0.446 0.267 0.097 0.387 0.328 0.331 0.481 0.468 0.371]

[0.30827068 0.41740758 0.27217125 0.13976945 0.30472441 0.34237996

0.3270751 0.3985087 0.43555142 0.3959445 ]

[1000 1000 1000 1000 1000 1000 1000 1000 1000 1000]

ماتریس آشفتگی:



شبکه سوم: لایه 32\*32 و لایه 32 و تایم ترین 11:39

model3 = Sequential([

    layers.Input(shape=(32\*32)),

    layers.Dense(32\*32,activation='relu'),

    layers.Dense(32,activation='relu'),

    #layers.Dense(10,activation='relu'),

    #layers.Dense(3,activation='relu'),

    layers.Dense(10,activation='softmax')

])

model3.compile(

    optimizer='adam',

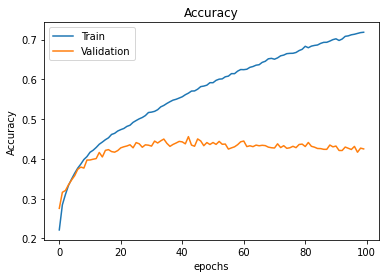
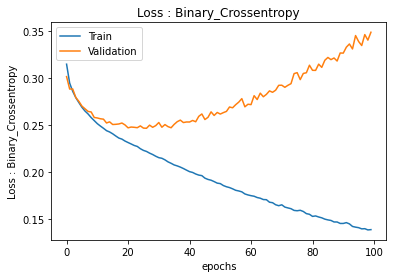
    loss='BinaryCrossentropy',

    metrics=['accuracy']

    #metrics=['mae','accuracy']

)

train\_model3 = model3.fit(x\_train,y\_train,epochs=100, validation\_split=0.2,batch\_size=128)



خطا و دقت به ترتیب:

0.35073500871658325

0.41819998621940613

همینطور این 4 پارامتر precision\_recall\_fscore\_support یه ترتیب:

[0.4275 0.55982437 0.34412266 0.27178153 0.36233611 0.32082695

0.47651007 0.44701195 0.50503018 0.53531599]

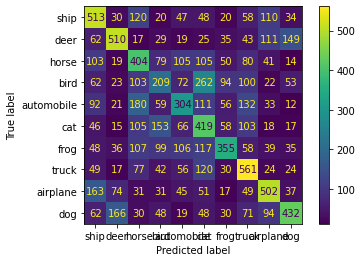
[0.513 0.51 0.404 0.209 0.304 0.419 0.355 0.561 0.502 0.432]

[0.46636364 0.53375196 0.37166513 0.23629169 0.33061446 0.36339983

0.40687679 0.49756098 0.50351053 0.47814056]

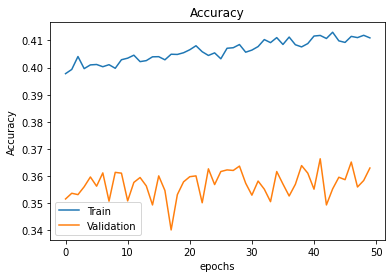
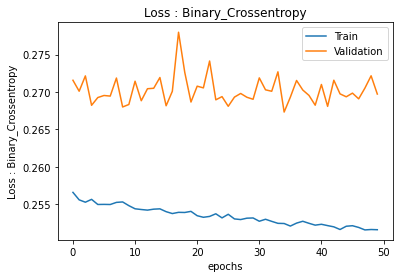
[1000 1000 1000 1000 1000 1000 1000 1000 1000 1000]

ماتریس آشفتگی:



د)مدل اول شبکه قبل مدل خوبی بود

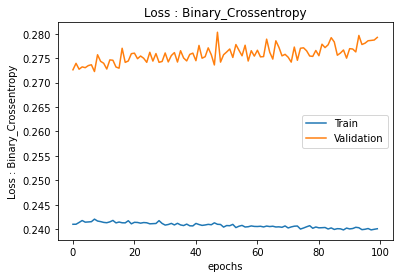
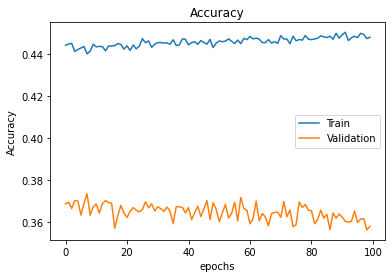
32: زمان آموزش 03:37



0.2702089548110962

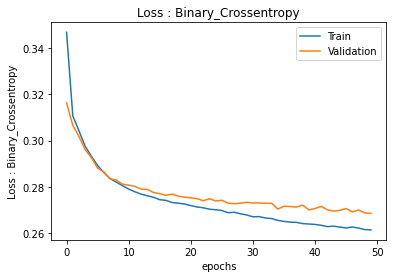
0.3594000041484833

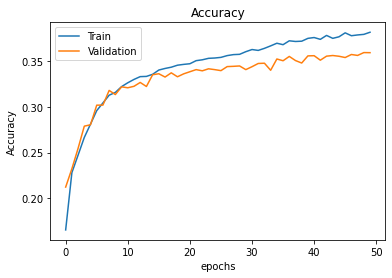
64: زمان آموزش 02:22



0.2691643238067627

0.35989999771118164

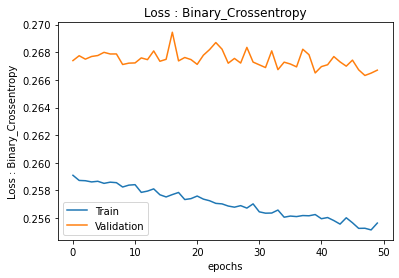
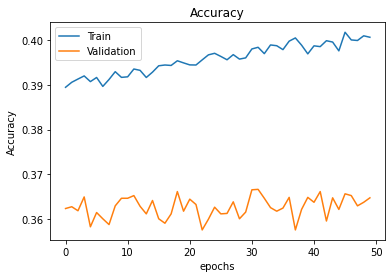
128: زمان آموزش 01:09



0.2681870460510254

0.36309999227523804

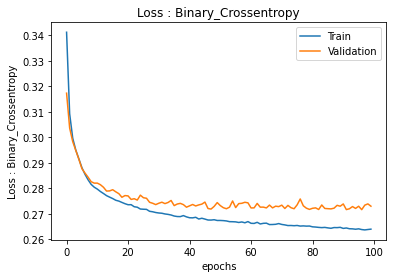
512: زمان آموزش 00:41

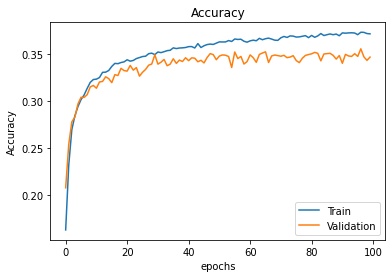


0.2666318714618683

0.37070000171661377

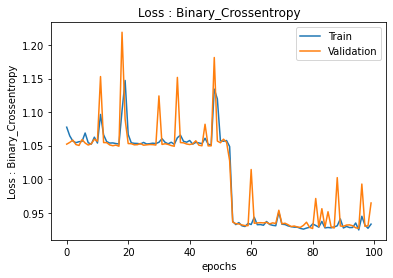
ه) کل توابع نورون ها رو عوض می کنیم و شبکه یک استفاده می شود

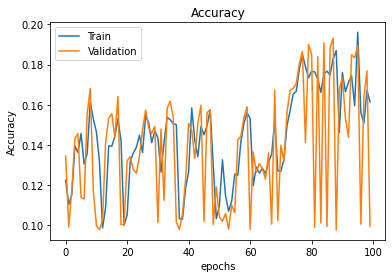
Relu: زمان اجزا 01:58



0.2720027267932892

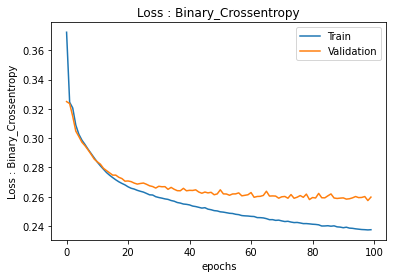
0.3517000079154968

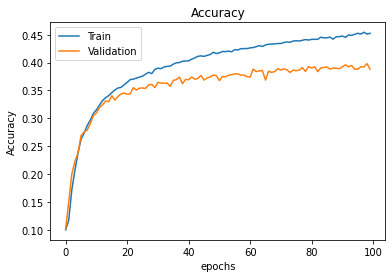
TanH: زمان اجرا 02:00



0.9621915817260742

0.09960000216960907

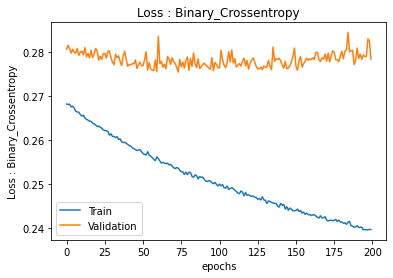
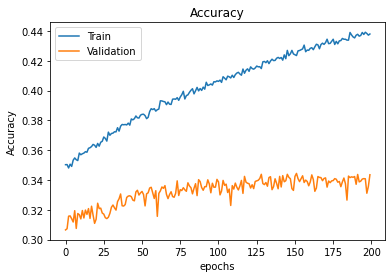
Sigmoid : زمان اجرا 02:22



0.2611045837402344

0.38449999690055847

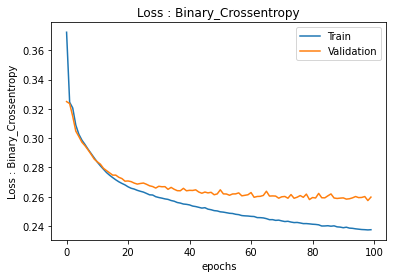
Softmax : رمان اجرا 03:22

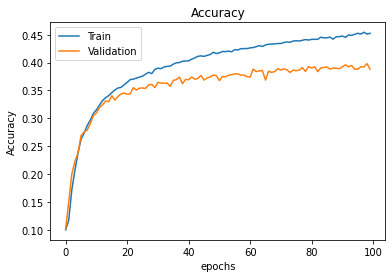


0.27904704213142395

0.3411000072956085

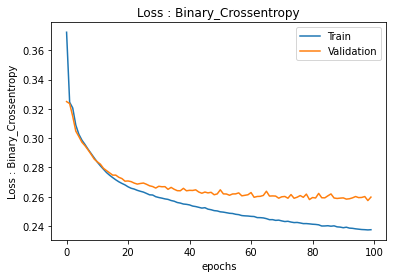
و) همان شبکه یک

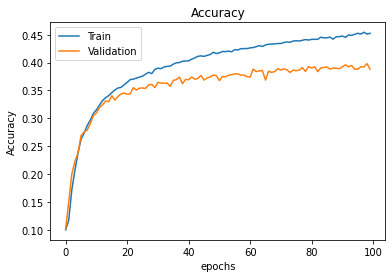
BinaryCrossentropy :



0.2611045837402344

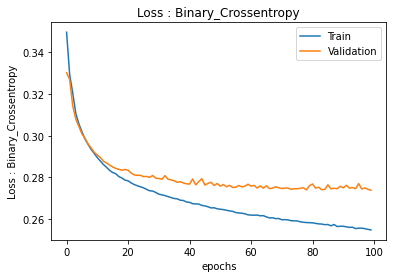
0.38449999690055847

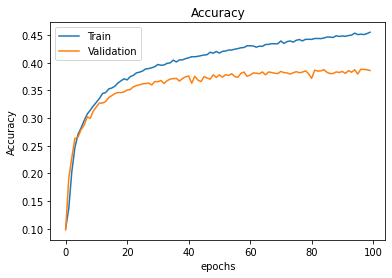
CategoricalCrossentropy :



0.2611045837402344

0.38449999690055847

Poisson :

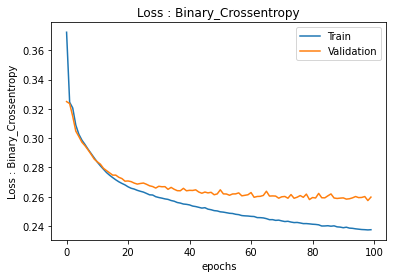
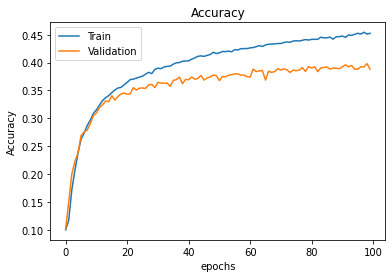


0.2740287184715271

0.3871000111103058

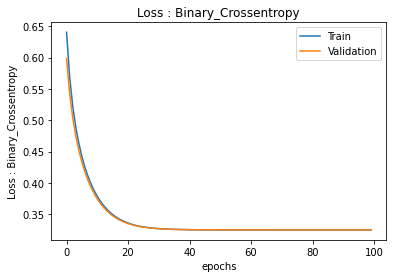
ج)همان شبکه شماره یک

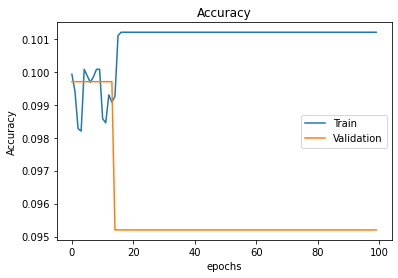
Adam :



0.2611045837402344

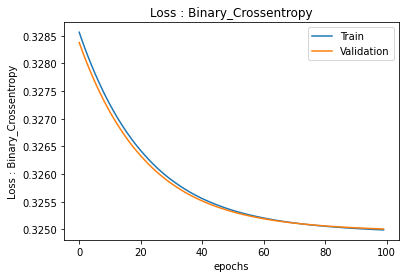
0.38449999690055847

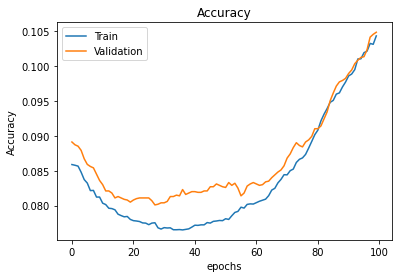
Ftrl :



0.3251250088214874

0.10000000149011612

Adadelta :



0.3249925971031189

0.10350000113248825

ح) افزودن لایه به شبکه

1]4 لایه 32\*10و32و32و32: زمان 06:22

model1\_sigmoid\_1 = Sequential([

    layers.Input(shape=(32\*32)),

    layers.Dense(32\*10,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(10,activation='sigmoid')

])

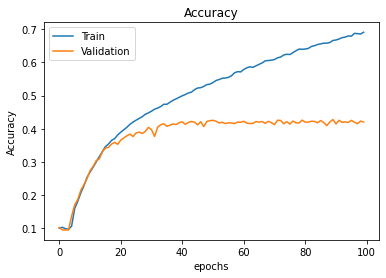
model1\_sigmoid\_1.compile(

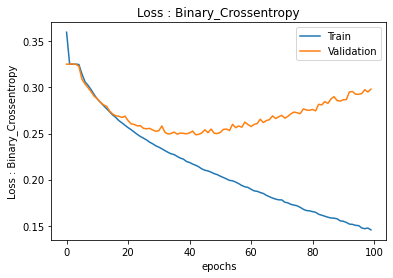
    optimizer='adam',

    loss='BinaryCrossentropy',

    metrics=['accuracy']

    #metrics=['mae','accuracy']

)



0.30024227499961853

0.41929998993873596

2)4لایه 32و32و32و32 : زمان 03:22

model1\_sigmoid\_2 = Sequential([

    layers.Input(shape=(32\*32)),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(10,activation='sigmoid')

])

model1\_sigmoid\_2.compile(

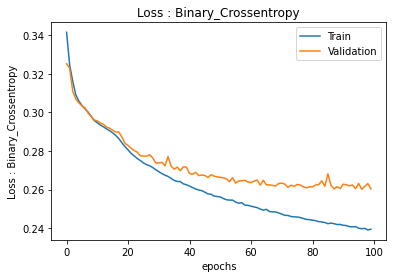
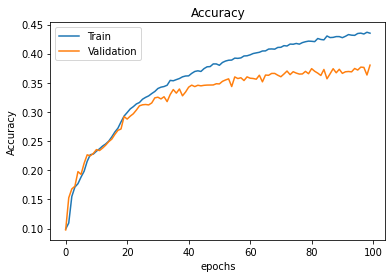
    optimizer='adam',

    loss='BinaryCrossentropy',

    metrics=['accuracy']

    #metrics=['mae','accuracy']

)



0.2622953951358795

0.37700000405311584

3)4 لایه 32و32و10و10 : زمان 03:22

model1\_sigmoid\_3 = Sequential([

    layers.Input(shape=(32\*32)),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(32,activation='sigmoid'),

    layers.Dense(10,activation='sigmoid'),

    layers.Dense(10,activation='sigmoid'),

    layers.Dense(10,activation='sigmoid')

])

model1\_sigmoid\_3.compile(

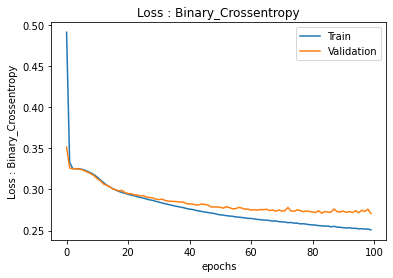
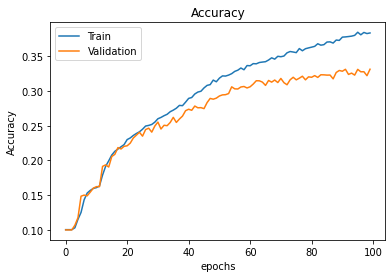
    optimizer='adam',

    loss='BinaryCrossentropy',

    metrics=['accuracy']

    #metrics=['mae','accuracy']

)



0.27165040373802185

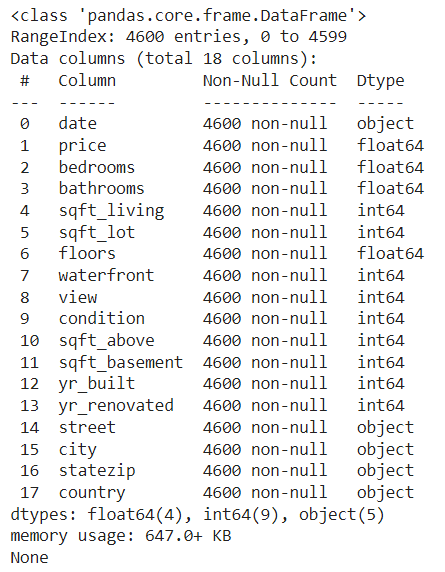
0.33079999685287476

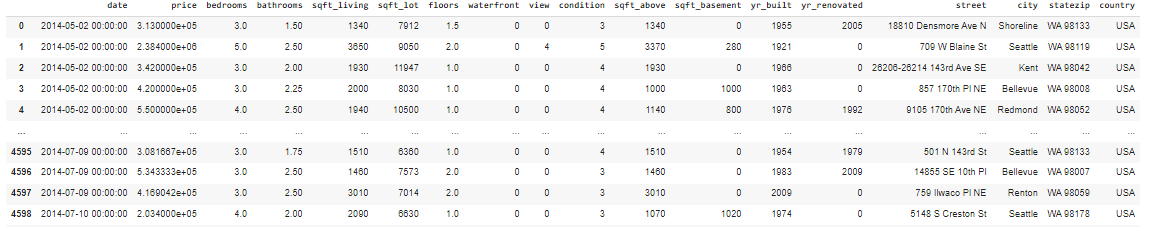
ط) انتخاب تعداد لایه بیشترین تاثیر را دارد سپس انتخاب تعداد نورون هر لایه سپس تعداد bach های هر دوره سپس تابع فعال ساز

ن) بهترین مدل برای شبکه 4 لایه قسمت ح شماره یک است که دقت 41 درصد داده است و زمان اجرای آن و بقیه اطلاعات در بخش خود آمده است

به علت نمایش تمام خروجی ها در قسمت قبل دیگر اینجا تکرار نمی شود

# سوال ۲ – MLP (Regression)

الف)ابتدا فایل csv را با استفاده از تابع پانداس read\_csv() وارد کرده و سپس نمایش می دهیم تا دیتا ها را مشاهده کنیم تا تعداد ویژگی ها و همینطوری کمی و کیفی بودن اطلاعات را بررسی می کنیم دید کلی از دیتا را مشاهده کنید:



سپس تابعی را تعریف می کنیم تا پیش پردازش های لازم را روی این دیتا انجام دهد و همینطور دیتا های کیفی را تبدیل به کمی کرده و همچینین مقادیر دیتا ها را نرمالیزه کند یعنی اعداد را به اعداد بین صفر و یک اسکیل می کنید.

اسم این تابع را ready\_to\_process(data) را گذاشته ایم که یه عنوان ورودی فقط دیتای خام پانداس را می گیرد و پیش پرداز های توضیح شده را روی آن انجام می دهد و دیتای آماده شده برای پردازش را به عنوان خروجی باز می گرداند.

کد تابع ساخته شده به صورت زیر است:

def ready\_to\_process(data) :

  data\_object = data.select\_dtypes(include='object')

  data\_number = data.drop(columns= data\_object.keys())

  data\_object = data\_object.fillna('NA')

  scaler = MinMaxScaler()

  for n in data\_object.keys():

    x=data\_object[n]

    x= x.astype("category").cat.codes

    x = scaler.fit\_transform(x.values.reshape((-1, 1)))

    data\_object[n]=x

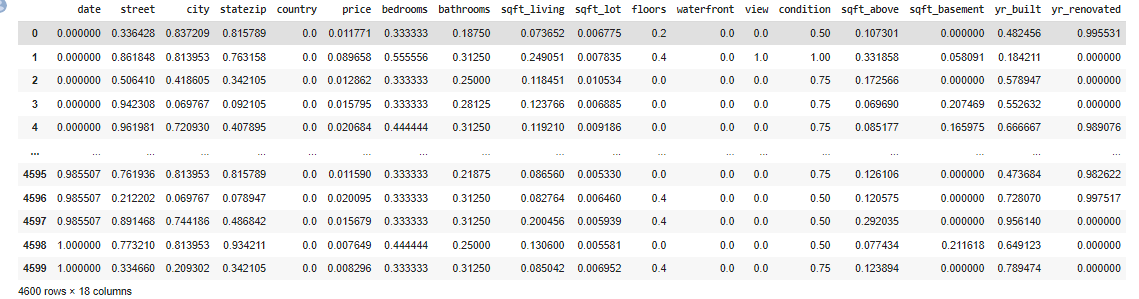
  for n in data\_number.keys():

    data\_number[n]= data\_number[n].fillna(data\_number.mean())

    data\_number[n] = scaler.fit\_transform(data\_number[n].values.reshape((-1, 1)))

  return pd.concat([data\_object , data\_number],axis=1)

این تابع ایتدا داده های کمی و کیفی را جدا می کند داده های کیفی را را کمی کرده سپس نرمال می کند و داده های ناموجودش را na می گذارد همچنین برای داده های کمی داده های ناموجود را میانگین داده ها جایگزین می کند و سپسنرمالیزه می کند سر آخر هر دو داده را کنار هم گذاشته و به عنوان خروجی بر می گرداند.

دیتا به صورت زیر در میاید:

این دیتا آماده پردازش است.

ب)حال ستون قیمت را از دیتا جدا کرده و آن را به عنوان برچسب در نظر می گیریم و سپس 20 درصد دیتا را برای تست جدا می کنیم :

Y = ready\_data['price'].to\_numpy().reshape((-1, 1))

X = ready\_data.drop(columns='price').to\_numpy()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

حال چهار شبکه عصبی برای آموزش این دیتای لیبل شده آماده می کنیم در دوتا تعداد لایه ها را تغییر می دهیم و در دو تای دیگر تایع فعال ساز نورون ها را:

شبکه اول:دو لایه 1024 تایی و تابع relu

activation1 = 'relu'

layer1 = 2

model1 = Sequential([

    layers.Input(shape=(X.shape[1],)),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

    layers.Dense(1, activation='linear')

])

شبکه دو : چهار لایه 1024 تایی و تابع relu

activation2 = 'relu'

layer2 = 4

model2 = Sequential([

    layers.Input(shape=(X.shape[1],)),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

    layers.Dense(1, activation='linear')

])

شبکه سه :د دو لایه 1024 تایی و تابع tanh

activation3 = 'tanh'

layer3 = 2

model3 = Sequential([

    layers.Input(shape=(X.shape[1],)),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

    layers.Dense(1, activation='linear')

])

شبکه چهار : چهار لایه 1024 تایی و تابع tanh

activation4 = 'tanh'

layer4 = 4

model4 = Sequential([

    layers.Input(shape=(X.shape[1],)),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

    layers.Dense(1024, activation=activation),

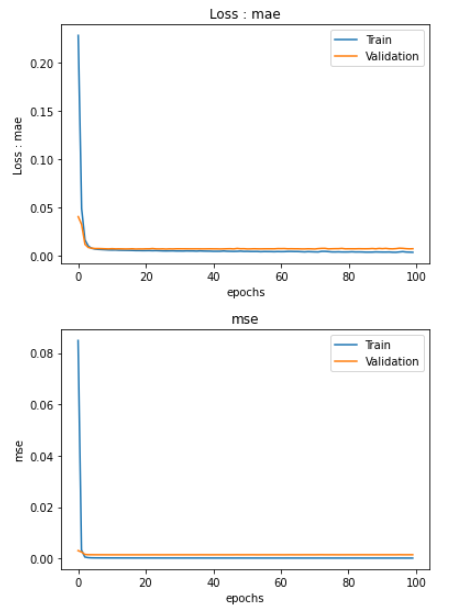
    layers.Dense(1024, activation=activation),

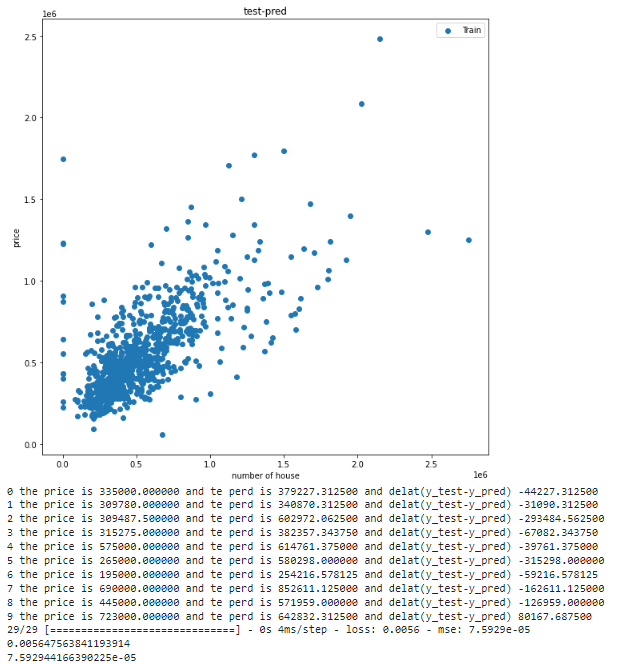
    layers.Dense(1, activation='linear')

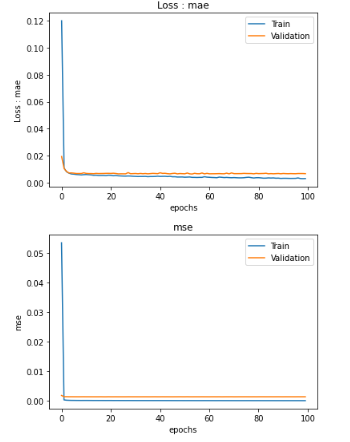
])

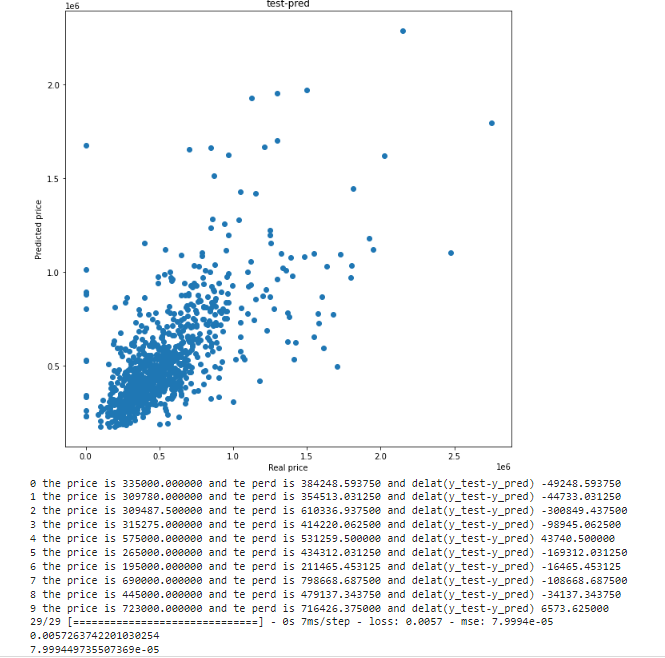
در شبکه اول یک رگرسیون دو لایه که ساده ترین نوع رگرسیون است با تعداد کافی نورن قرار داده شده است در شبکه دوم قبل از رگرسیون یه پیش پرداز و حذف اعوجاج و نویز دو لایه هم قرار دادیم ویک شبکه 4 لایه را تشکبل داده ایم و در دو شبکه دیگر همین دو شبکه را با تابع فعال ساز نرم تر جایگیزن کرده ایم تا ضرایب نرمتر به ضرایب بهینه سر بخورند.

ج) این شبکه را با تابع loss ، mse تمرین می دهیم و نمودار های خروجی را برای هر ایپاک نمایش می دهیم و برای داده های تست مقدار واقعی و مقدار پیشبینی شده را در یک نمودار رسم می کنیم که هر چه روی خط نیم ساز قرار بگیره پیشبینی دقیق تر است و سپس 10 خانه ی اول تست را قیمتشان به همراه پیشبینی آن پیرینت شده است و سر آخر هم مقدار خطای mse , mae آنها در پایین آن آمده است:

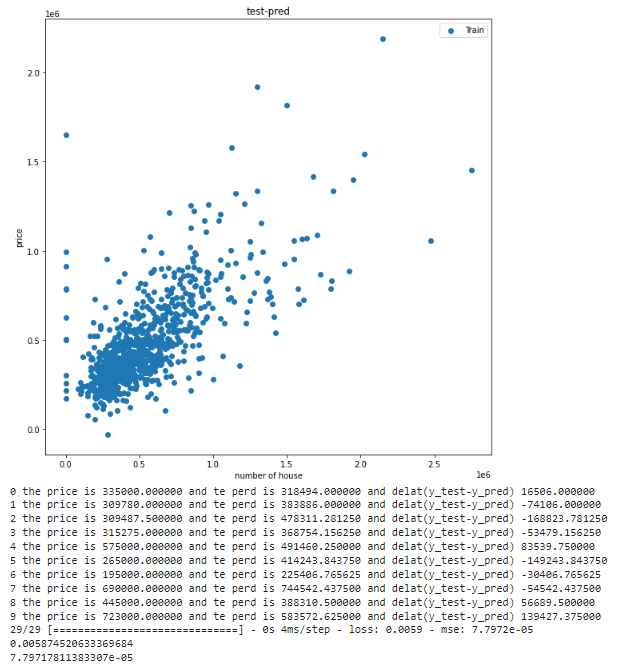
شبکه یک:



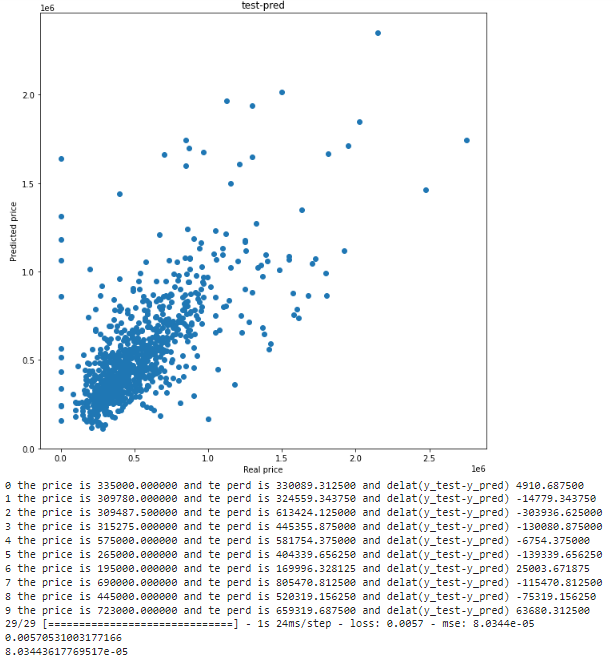
شبکه دو:



شبکه سه:

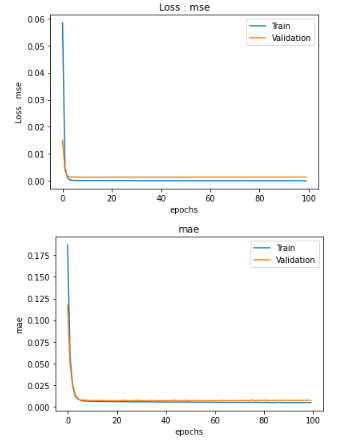


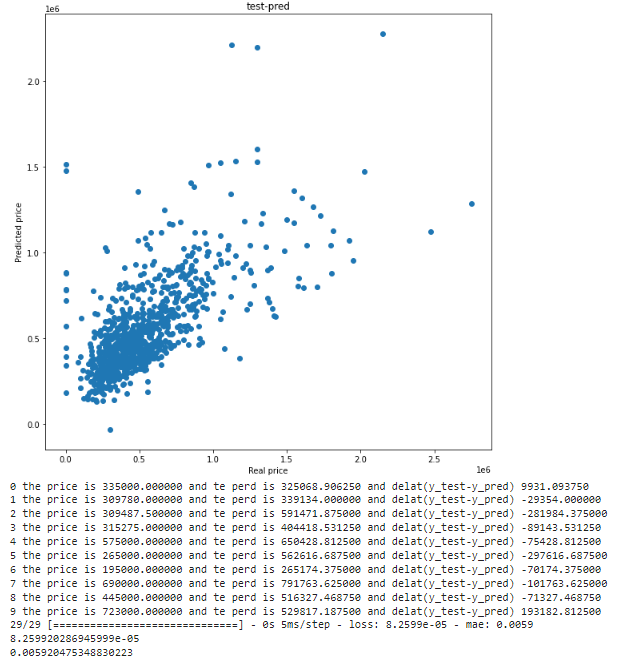
شبکه چهار:

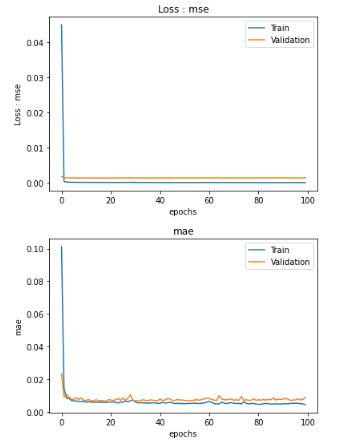


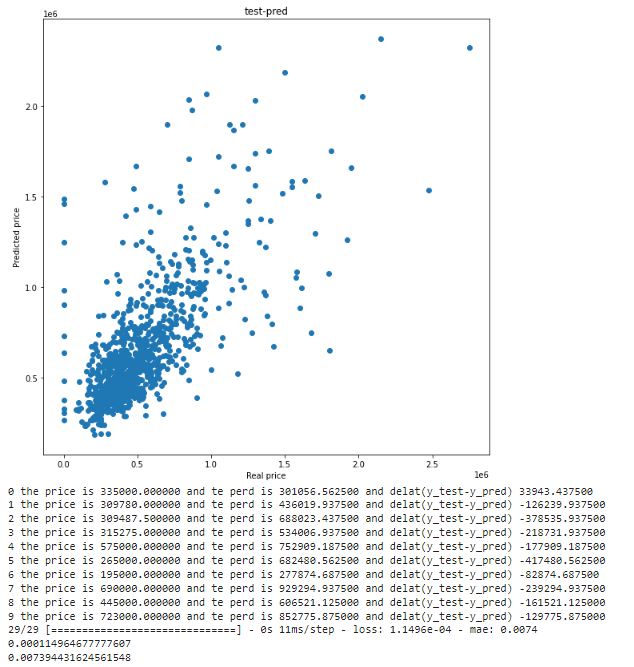
همان طور که ملاحضه می کنید شد شبکه تا 20 ایپاک حدودا ثایت می شود ولی می شود تعداد ایپاک ها رو بین 5 تا 10 هم در نظر گرفت اما با 20 ایپاک کاملا عملکرد شبکه مشخص می شود.

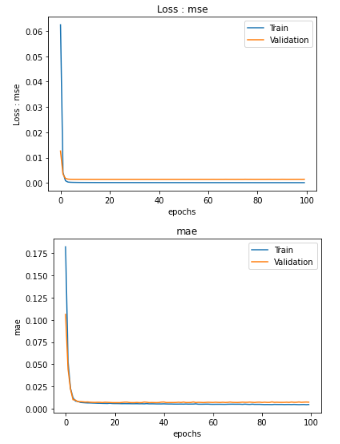
د)این قسمت تابع loss را mae می گیرم و دوباره شبکه ها را تمرین می دهیم:

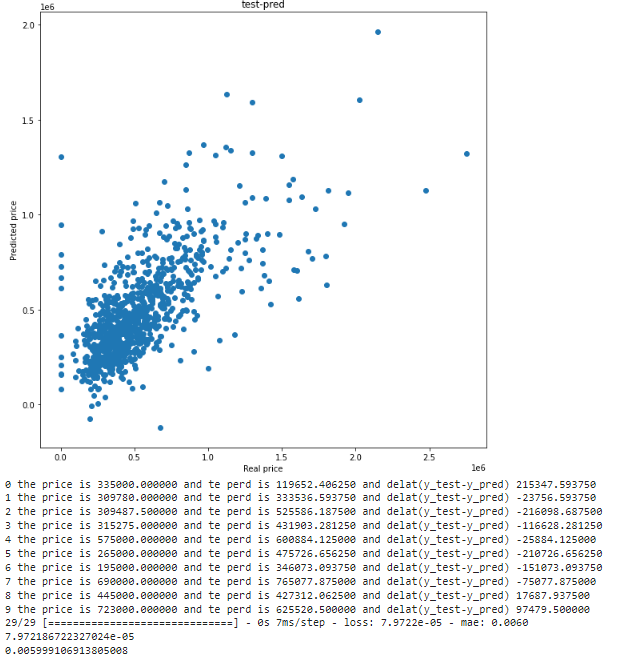
شبکه اول:



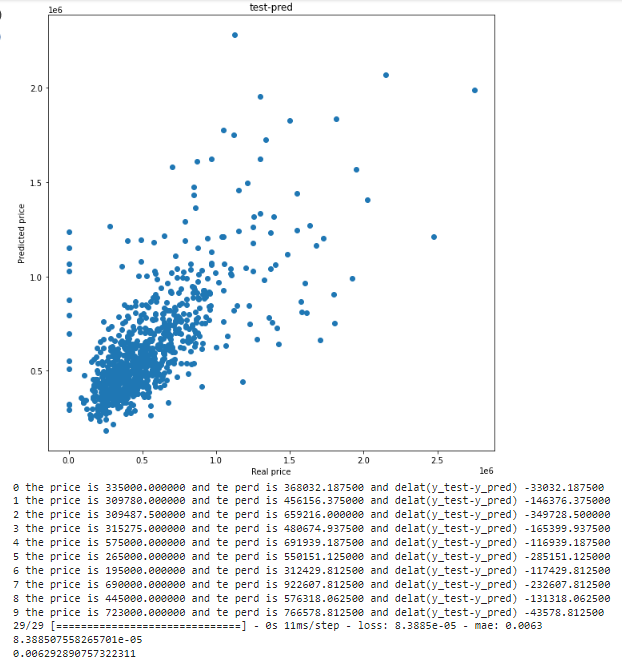
شبکه دوم:

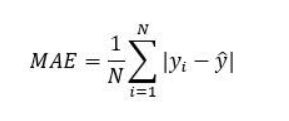


شبکه سوم:

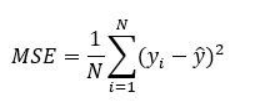


شبکه چهار:



ه) تابع MAE به صورت زیر است :

مجموع قدر مطلق اختلاف هر دیتا با میانگین خودش تقسیم بر تعداد کل داده ها میانگین تمام اختلاف های داده ها از میانگین داده ها

تابع MSE به صورت زیر است:

مجموع مجذور اختلاف هر دیتا با میانگین خودش تقسیم بر تعداد کل داده ها یا همان میانگین تمام مجذور اختلاف های داده ها از میانگین داده ها

MSE دیتا هایی که از میانگین دور ترهستن رو ضریب بیشتری می دهد و اهمیتش روی اختلاف های زیاد بیشتر است در واقع مینیم سازی این تابع داده هار رو بیشتر کنار هم نگه میداره نسبت به MAE

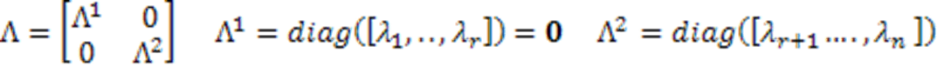
مقایسه ج و د نشان می دهند که شبکه های محاسبه شده با loss ، mae دارای مقدار خطای کمتری هستند اما نسب به شبکه های محاسبه شده با loss ، mse دارای پراکندگی بیشتری هستند در واقع اگر بخواهیم همه داده ها تا مقداری که می شود درست تشخیض داده بشوند mse را استفاده می کنیم ولی اگر بخواهیم اکثر داده ها با دقت بیشتری تشخیص داده شوند از mse استفاده می کنیم.

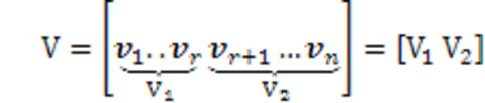
# سوال 3 – کاهش ابعاد

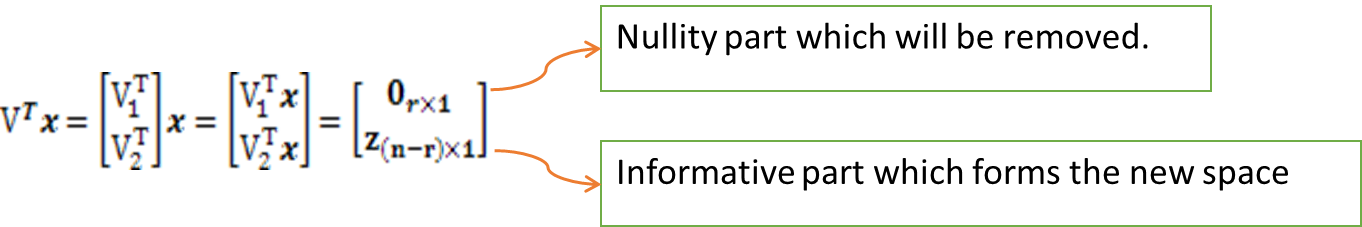
الف) ما در روش pca میخواهیم با استفاده از جبر خطی و مقادیر ویژه ابعاد اضافی و با اطلاعات کم دیتای خودمان را از بین ببیرم این روش به این صورت است که :

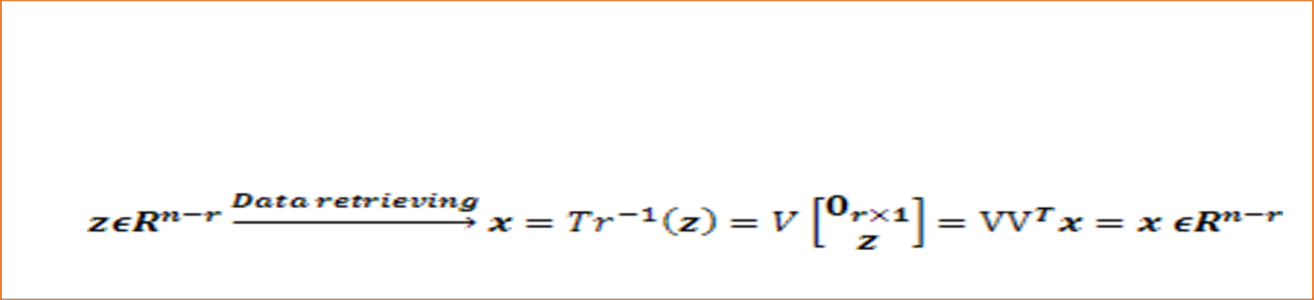
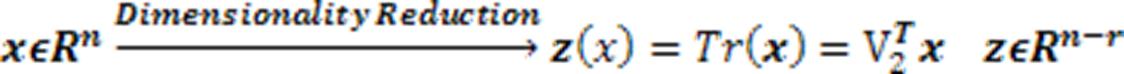
ابتدا ماتریس کواریانس داده خود را به دست می آوریم (**xT=[1,x1,x2,..,xn])**

این ماتریس مقدار وابستگی تمام ابعاد ورودی را به هم به ما می دهد حال ما با به دست آوردن مقادیر و بردار های ویژه این ماتریس می توانیم ابعاد با اطلاعات بیشتر را از ابعاد بدون اطلاعات جدا کنیم معیار وجود اطلاعات هم داشتن واریانس بیشتر است . پس بردار ویژه و مقدار ویژه ماتریس ها را به دست می آوریم و سپس مقادیر ویژه سفر را جدا کرده در ابتدا بردار قرار می دهیم و بردار های آنها را نیز در ماتریس جا به جا می کنیم تا بتوانیم ماتریس را به دو قسمت بردار ویژه صفر و گیر صفر تقسیم کنیم حال بردار های غیر صفر را جدا کرده و ماتریس جدید به وجود آمده در واقع تبدیلی برای کاهش بعد دیتا ی ورودی است و با ضرب کردن ترانهاده ی آن تابع در ابتدای دیتای تبدیل شده که به جای ورودی های حذف شده صفر گذاشتیم دیتا اولیه درست می شود که یعنی اطلاعاتی را از دست ندادیم.









حال چون گفته شده از کتابخانه pca استفاده نشود ما خودمان باید آن را پیاده کنیم تا بتوانیم ابتدا تشخیص دهیم چند بعد باید کاهش داد که یعنی بعد های با اندازه مقدار ویژه کم و نزدیک به صفر را حذف کنیم و بعد با کنار هم چیندن بردار های ویژه ای که حذف نکردیم در ماتریس ترانهاده شده و ضربش در داده ها ابعاد داده ها را کم کنبم.

ابتدا تابع منحنی تجمعی مقدار واریانس مقدار ویژه ها را تایپ میکنیم:

def plot\_comp\_var(X):

  cov\_mat = np.cov(X.T)

  eig\_vals = np.linalg.eig(cov\_mat)[0]

  eig\_vals\_sort = [(np.abs(eig\_vals[i])) for i in range(len(eig\_val)]

  cum\_exp\_val = np.cumsum(eig\_vals\_sort/sum(eig\_vals))

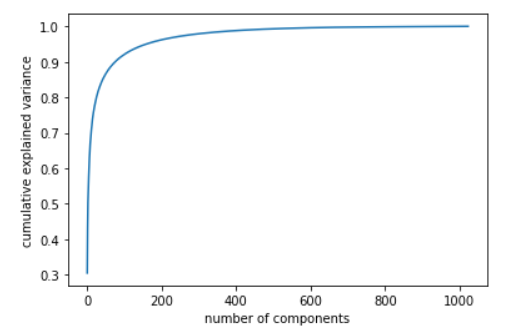
  return cum\_exp\_val

این تابع ابتدا ماتریس کواریانس ورودی را محاسبه می کند

سپس مقادیر ویژه آن را محاسبه کرده و در متغییری ذخیره میکند

بعد مقادیر ویژه را سورت کرده و نرمالیزه می کند تا جمع تمامش یک شود

سرآخر به صورت تجمعی در آورده و خروجی می دهد برای رسم کردن.

ما داده های سوال یک را همانند کاری که انجام دادیم پیش پردازش می کنیم و وارد این تابع کرده ایم تا خروجی را رسم کنیم .حاصل:

همان طور که ملاحضه می شود بیش از 90 درصد اطلاعات در 150 بعد ابتدایی موجود است پس می توان با دقت خوبی داده ها را از 32\*32 به 150 کاهش داد.

حال تابع کاهش بعد را می نویسیم:

def pca\_n\_dec(X,X1,n):

  cov\_mat = np.cov(X.T)

  eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)

  eig\_pairs = [(np.abs(eig\_vals[i]), eig\_vecs[:,i]) for i in range(len(eig\_vals))]

  v = eig\_pairs[:n]

  y=[]

  for i in v:

    y.append(i[1])

  v\_tr = np.array(y)

  x=[]

  for i in range(len(X1)):

    x += [np.dot(v\_tr,X1[i])]

  x\_out\_n=np.array(x)

  return x\_out\_n

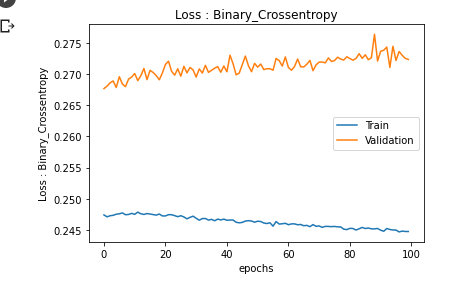
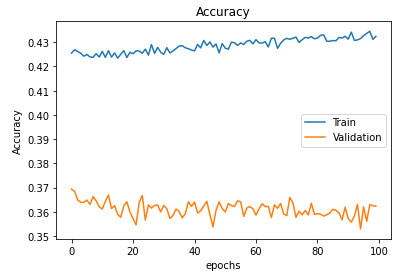
مقادیر ویژه و بردار های ویژه را سورت می کنیم

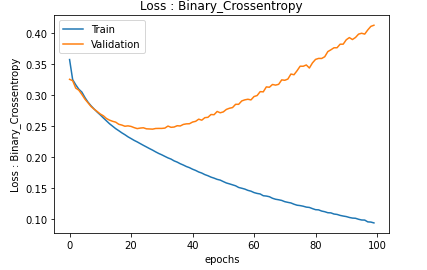
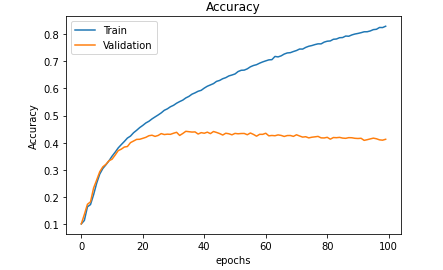
150 تای اول را نگه می داریم و تبدیل به آرایه میکنیم

سپس تمام داده ها رو درونش ضرب داخلی میکنیم تا داده های جدید به دست بیایند

سر آخر هم دادهها را به عنوان خروجی پاس میدهیم.

ب)حال داده های train و test را به تابع می دهیم تا کاهش بعد انجام شود و سپس به شبکه ی ایجاد شده در سوال یک میدهیم تا بهبود را مشاهده کنیم

که شرایط از دقت 35 درصد به دقت 40 درصد افزایش یافته است که نشان دهنده بهبود عملکرد شبکه است.

این دو نمودار عملکرد قدیم شبکه است.

و این دو نمودار برای بعد از انجام pca است که به وضوع بهبود یافته است.

ج) حال می خواهیم با روش شبکه auto-encoder ابعاد را کاهش دهیم و سپس دوباره با همان شبکه قبل مدل را آموزش دهیم تا نتیجه را مشاهده کنیم.

برای ساخت auto encoder از کتابخانه keras استفاده می شود و بر خلاف کد های قبلی این بار باید برای هر لایه اسم گذاری کرد تا بتوان لایه وسطی را که لایه encode شده است را از خروجیش استفاده کرد و ما شبکه auto encoder خود را اینگونه تعریف می کنیم:

import keras

input\_img = keras.Input(shape=(32\*32,))

encoded1 = layers.Dense(512, activation='relu')(input\_img)

encoded2 = layers.Dense(256, activation='relu')(encoded1)

encoded3 = layers.Dense(150, activation='relu')(encoded2)

decoded1 = layers.Dense(256, activation='relu')(encoded3)

decoded2 = layers.Dense(512, activation='relu')(decoded1)

decoded3 = layers.Dense(32\*32, activation='sigmoid')(decoded2)

autoencoder = keras.Model(input\_img, decoded3)

encoder = keras.Model(input\_img, encoded3)

encoded\_input = keras.Input(shape=(150,))

decoder\_layer = autoencoder.layers[-1]

#decoder = keras.Model(encoded\_input, decoded3)

autoencoder.compile(

    optimizer='adam',

    loss='mse',

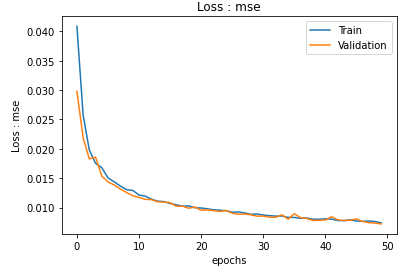
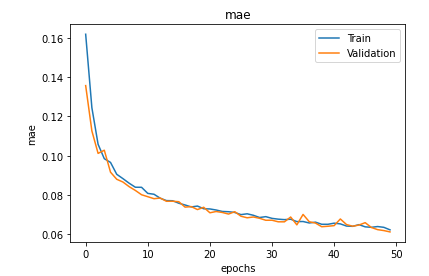
    metrics=['mae']

    #metrics=['mae','accuracy']

)

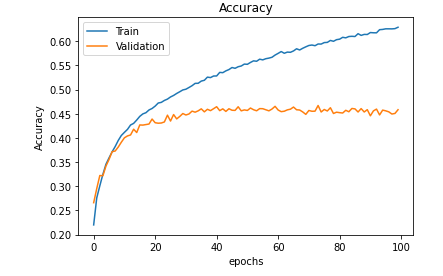
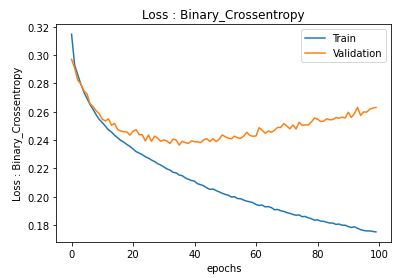
ما دو لایه دیکودر قرار دادیم تا ورودی را از 32\*32 به 150 برسانیم همان عددی که در قسمت قبل محاسبه کردیم این دولایه را 512 و256 نورونه قرار دادیم.

ما شبکه را به علت کمبود وقت به جای 100 ایپاک در 50 ایپاک ترین دادیم که همان هم نتیجه قایل قیولی را ارائه داد نمودار کاهش خطای آن را مشاهده بفرمایید:



حال خروجی شبکه را هم به نمایش گذاشته می شود و به علت کیفت پایین عکس ورودی و ورود عکس هی جدید test به شبکه عکس خارج شده از شبکه ما هم کیفت خوبی ندارد و زیاد قایل تشخیص نیست اما یه علت این که خطا کلی کمتر از 0.01 است قایل قبول است

حال داده های کاهش بعد یافته با استفاده از این شبکه را که از لایه ی میانی شبکه می گیرم را وارد شبکه قدیمی خودمان می کنیم تا بعد از تمرین دادن شبکه متوجه نتیجه بشویم که خروجی های آن به صورت زیر است:

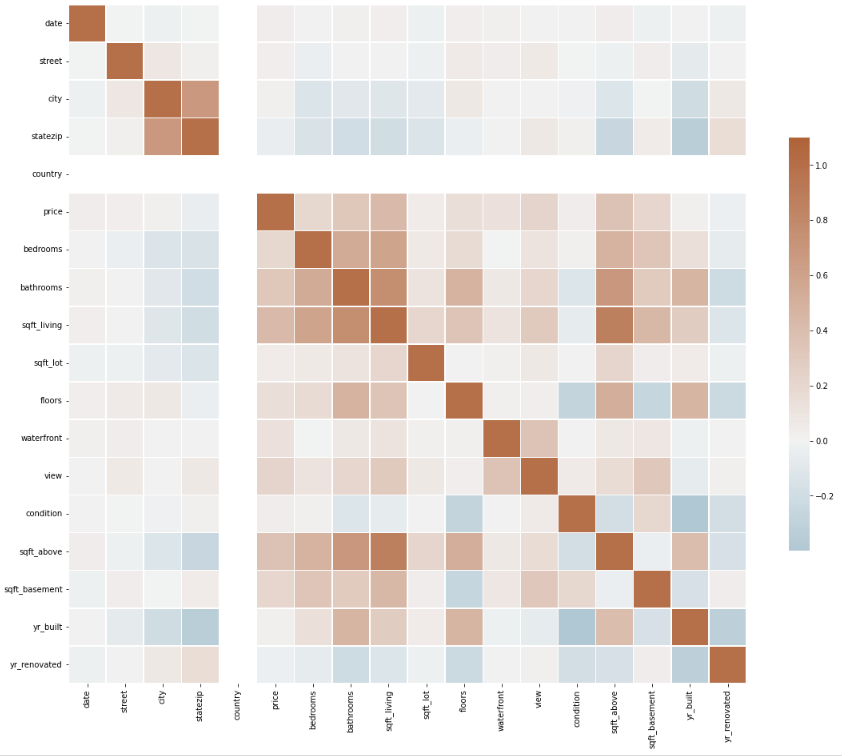


و همینطور برای داده های تست 45 درصد دقت و 0.26 خطا داریم که از دو شبکه قبل بهتر از و این شبکه از pca هم نتیجه بهتری گرفت.

|  |  |  |  |
| --- | --- | --- | --- |
|  | دقت (درصد) | خطا | زمان(دقیقه) |
| شبکه بدون کاهش بعد | 35.7 | 0.27 | 2:32 |
| pca | 40.2 | 0.41 | 4:22 |
| Auto encoder | 44.5 | 0.26 | 4:02 |

همانطور که مشاهده می کنید به وضوح شبکه با کاهش بعد auto encoder برای آن دو لایه شبکه که ما در سوال یک آموزش دادیم بهتر کار کرده است و دقت بالاتری برای داده های جدید دارد همینطور خطای کمتری هم در آن هست.

د) ابتدا ماتریس همبستگی را مطابق صورت سوال رسم میکنیم. فقط دقت داریم برای محاسبه ماتریس همبستگی باید همه ویژگیها به صورت عددی باشند، پس از پیشپردازشی که در سوال 1 استفاده کردیم برای تبدیل ویژگیهای categorical به numerical استفاده میکنیم.



همانگونه که مشاهده میشود، در این ماتریس میزان همبستگی خطی ویژگیها به یکدیگر مشخص میشود، یعنی هرچی درایه یک سطر و ستون از لحاظ اندازه بزرگتر باشند میزان همبستگی خطی میان ویژگی سطر و ستون متناظر بیشتر خواهد بود و با داشتن یکی میتوانیم دیگری را توصیف کنیم. با این تعریف انتظار داریم که درایههای قطر اصلی این ماتریس برابر 1 باشند که همینطور هم هست. برای انتخاب ویژگیهای مناسب جهت تخمین قیمت باید ویژگیهایی را انتخاب کنیم که بیشترین همبستگی را با قیمت خانه داشته باشند

و) با استفاده از مدلهای Linear Regression و Decision Tree ، اهمیت هر ویژگی را در سوال 2 بدست آورید و در یک بارپلات نمایش دهید. برای این کار میتوانید از کتابخانه scikit-learn و متدهای DecisionTreeRegressor و LinearRegression استفاده کنید .

ما داده ها را لیبل دار کرده و با استفاده از کتابخانه های معرفای شده بار پلات های هر یک را رسم کرده ایم

