به نام خدا

part1)

در سوال اول ابتدا ورودی و در دنبال آن خروجی مربوطه در ماتریس هایی ذخیره میشوند

```
n1 = 400
n2 = 3
X = np.zeros((n1,n2))
Y = np.zeros((n1,1))


for i in range(n1):
    X [i, :] = np.random.uniform(-1,1,n2)
    Y [i] = 0.1*np.sin(X[i ,0]) + 0.3*np.sin(X[i,1]) + 0.8*np.sin(X[i,2])
# by reduce the Coefficients by .01 the error reduced by 0.01
```

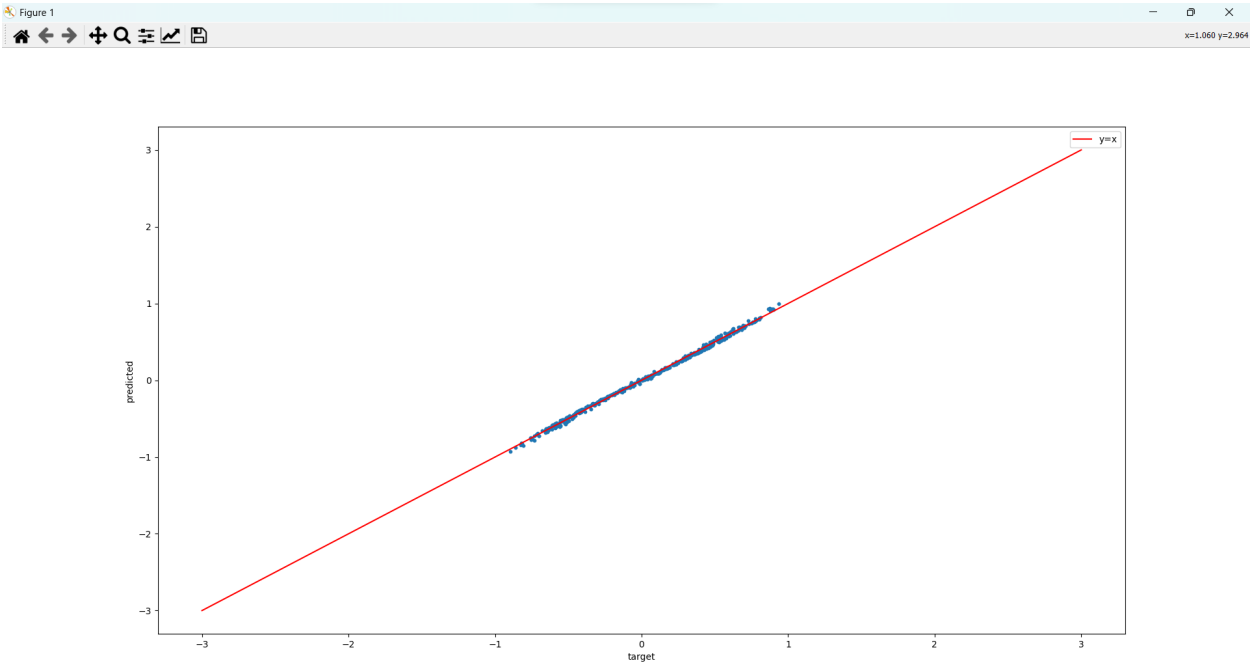
که با آزمایش و انجام با ضریبهای مختلف برای تابعمان به این نتیجه میرسیم که با یک دهم کردن ضرایب خطای نهاییمان یک صدم میشود.

حال یک مدل رگرسیونی میسازیم و روی داده ها ترین میکنیم

```
model = li.LinearRegression()

model.fit(X,Y)

o = model.predict(X)
```

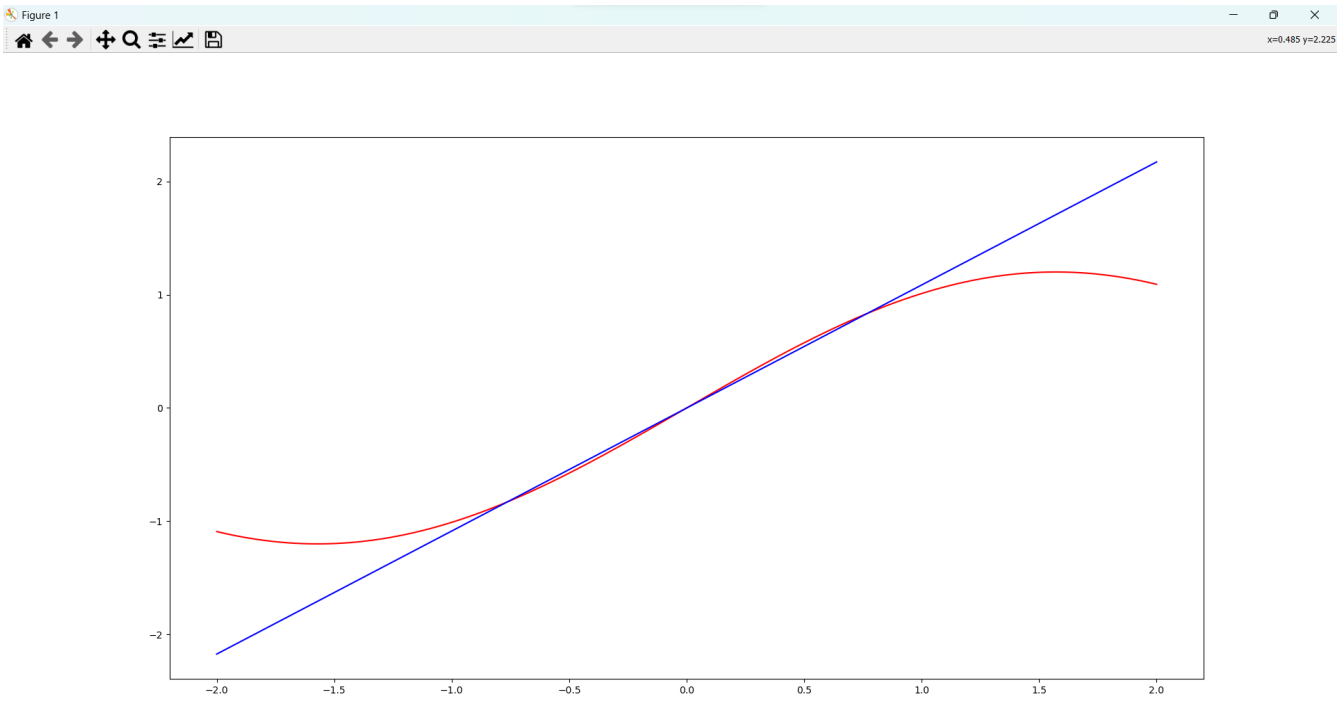
حال یک مدل شبکه عصبی که دارای وزن های خاص برای پیش بینی تابع مان می باشد در کنار خود تابع اصلی داریم.
حال خطای این دو را نسبت به یکدیگر به دست می آوریم و رسم می کنیم.

```
mse = met.mean_squared_error(Y,o)
print(f"the mean squared error is {mse}")
plt.scatter(Y[:] , o[:] , s = 10)
plt.xlabel('target')
plt.ylabel('predicted')
plt.plot([-3,3],[-3,3] , c = 'red',label = 'y=x')
plt.legend()
plt.show()
```



حال مدل شبکه عصبی را در کنار خود تابع بر روی 600 داده ی اعداد بین -2 تا 2 با فاصله های مساوی ، ترسیم می کنیم و با هم مقایسه می کنیم.(قرمز مربوط به تابع و آبی مربوط به تابع پیش بینی شده ی حاصل از شبکه ی عصبی می باشد.)

```
t = np.linspace(-2, 2, 600)

a = 0.1*np.sin(t) + 0.3*np.sin(t) + 0.8*np.sin(t)

t1 = np.zeros((len(t),n2))
for i in range(len(t)):
    t1[i,0] = t[i]
    t1[i,1] = t[i]
    t1[i,2] = t[i]

b = model.predict(t1)


plt.plot(t, a, 'r')
plt.plot(t, b, 'b')
plt.show()
```



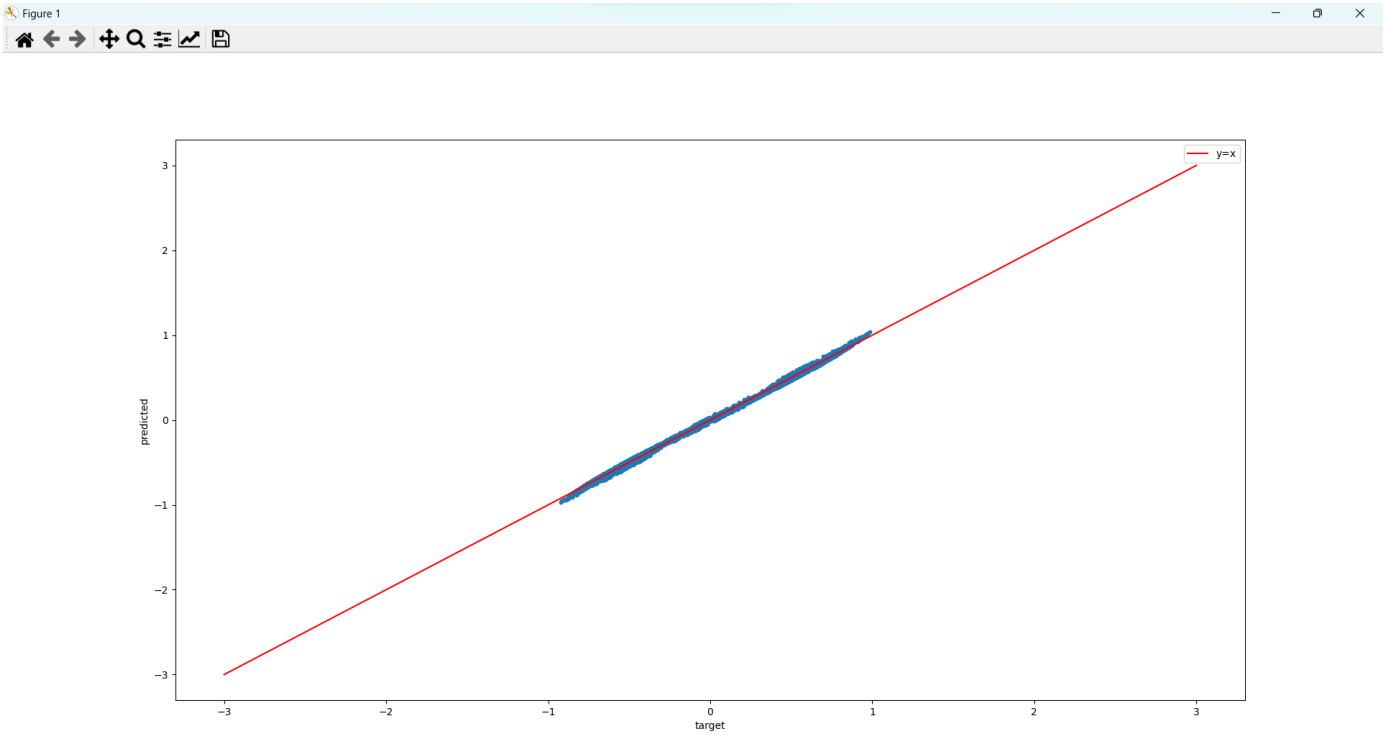
و مقدار خطای بدست آمده(mean squared error) به مقدار زیر حاصل میشود.

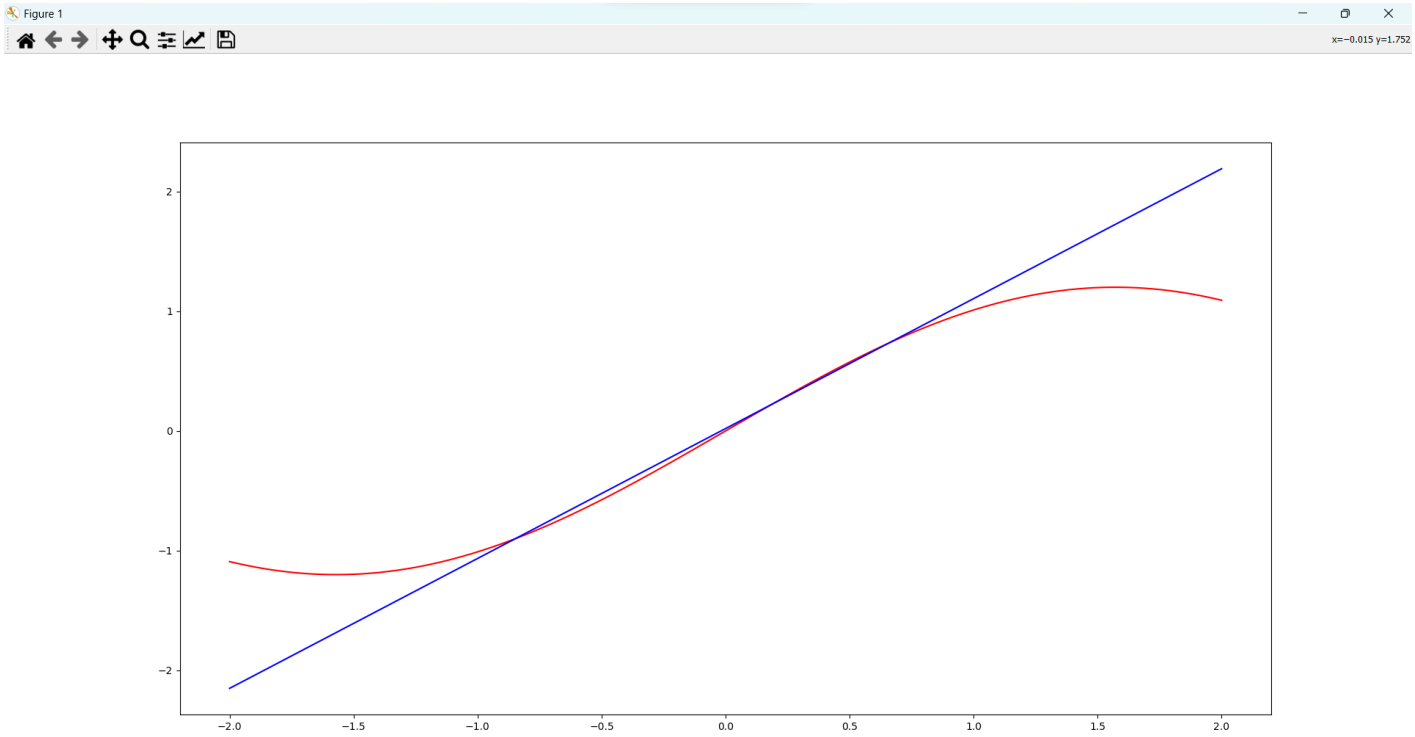
```
the mean squared error is 0.0004498201334568398 4
```

part2)

این قسمت همان قسمت قبلی می باشد با این تفاوت که به داده ها نویز اضافه میکنیم و قدرت یادگیری و رفع نویز شبکه ی عصبی را در این قسمت می سنجیم
که تنها خط متفاوت این قسمت با قسمت قبلی قسمت نویز و اضافه کردن آن به داده هایمان میباشد.

```
noise = np.random.uniform(-0.2,0.2)

for i in range(n1):
    X [i, :] = np.random.uniform(-1,1,n2)
    Y [i] = 0.1*np.sin(X[i ,0]) + 0.3*np.sin(X[i,1]) + 0.8*np.sin(X[i,2]) + noise
# by reduce the Coefficients by .01 the error reduced by 0.01
```





خطا مقدار زیر میشود که نسبت به حالت قبل کمتر شده اما باز هم با توجه به وجود نویز مقدار قابل قبولی می باشد.

```
the mean squared error is 0.00039979222078999916
```

part3)

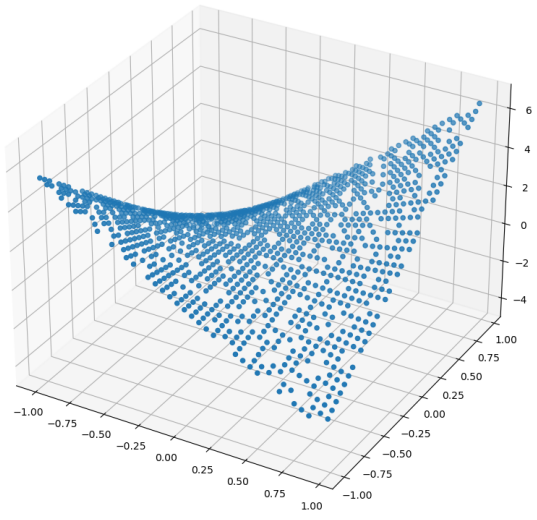
این قسمت نیز مشابه قسمت های قبل می باشد با این تفاوت که تابع ما دو بعدی می باشد.

تابع z ما دو ورودی x و y را میگیرد و خروجی را محاسبه می کند.

```
def z(x,y):
    return 5*x*y + 2**x
```

کدهای زیر نیز برای نمایش این داده ها و تابع روی آن ها به صورت سه بعدی با استفاده از کتابخانه های موجود، نوشته شده اند.

```
x_train, x_test, y_train, y_test = train_test_split(xy, out)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')


x1_vals = np.array([p[0] for p in x_train])
x2_vals = np.array([p[1] for p in x_train])


x1_valz = np.array([p[0] for p in x_test])
x2_valz = np.array([p[1] for p in x_test])


ax.scatter(x1_vals, x2_vals, y_train)
plt.show()
```



حال مدل شبکه ی عصبی را می نویسیم با آن را با داده های ورودی و خروجی حاصل از صدا زدن تابع روی آن ها، آموزش می دهیم.

```
mlp = MLPRegressor(
    hidden_layer_sizes=[20],
    max_iter=3000, #2000 best 3000 second
    tol=0,
)

# train network
mlp.fit(x_train,y_train)
```

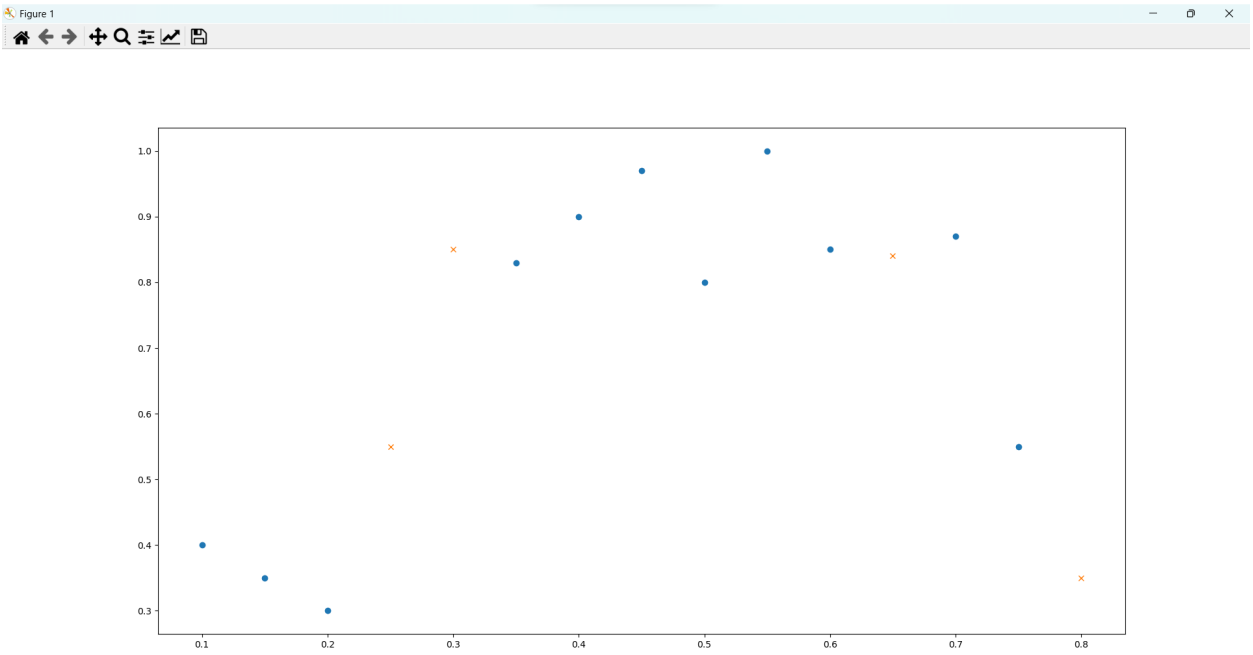
با دادن max_iter های مختلف به شبکه و آزمایش و تحلیل نتایج بدست آمد که بهترین مدل و در پی آن کمترین خطا زمانی رخ می دهد که max_iter برابر 2000 باشد. همچنین max_iter=3000 نیز نتیجه ی خوبی به ما می دهد.
و در آخر بعد از آموزش مدلمان نوبت تست آن و محاسبه ی دقت و خطای آن می باشد.که مدل را روی داده های تست پیش بینی می کنیم و با استفاده از جواب های مورد انتظار و فرمول MSE خطای آن و نمودار مربوطه را رسم می کنیم.

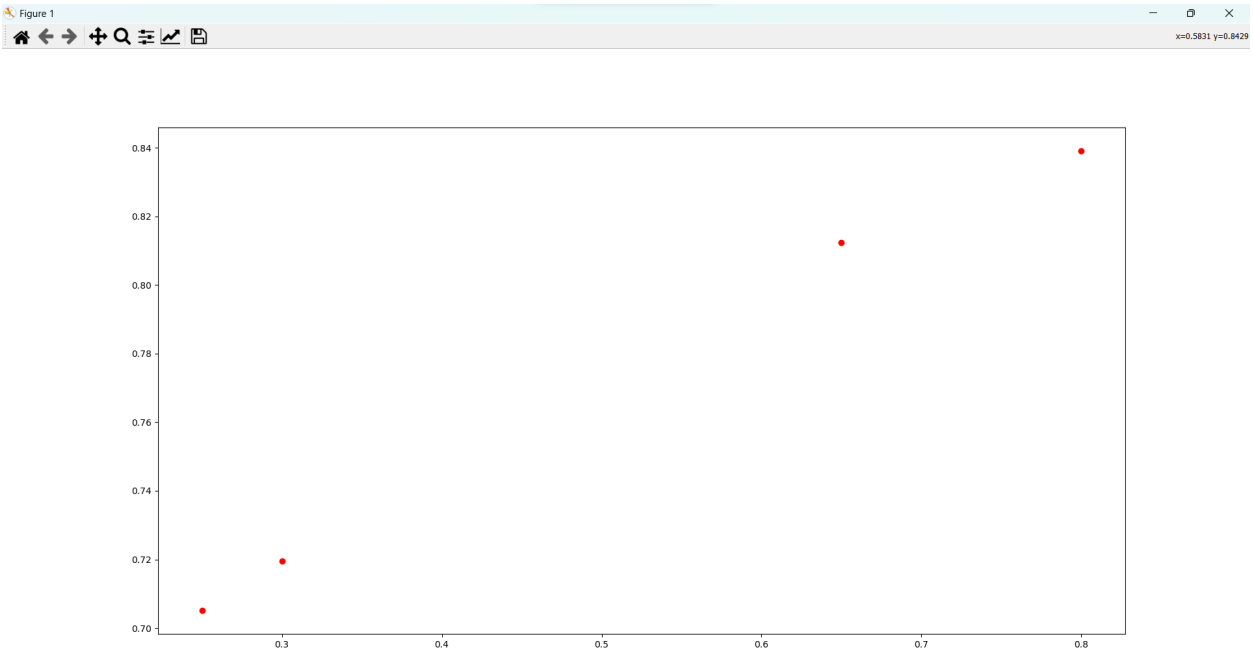
```
# test
predictions = mlp.predict(x_test)
mse = mean_squared_error(y_test, predictions)
print(mse)
ax.scatter(x1_valz, x2_valz, predictions, c='red')

plt.show()
```

part4)

در این قسمت یک مجموعه داده ی ورودی و خروجی به عنوان تابع دلخواه را بدست می آوریم و همانند قسمت های قبلی آن را با شبکه ی عصبی پیش بینی می کنیم و نتایج مدل بدست آمده را با مقادیر خروجی مقایسه می کنیم.

```
x = np.arange(0.1,0.85,0.05)
print(x)
out = [0.4, 0.35, 0.3, 0.55, 0.85, 0.83, 0.9, 0.97, 0.8, 1, 0.85, 0.84, 0.87, 0.55, 0.35]
```





part5)

در ابتدای این قسمت ما باید داده های مورد نظر را که انتخاب کردیم load کنیم که برای این کار از کد زیر استفاده میکنیم.

```
local_zip = '/content/USPS_images.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/content/trainntest')
train_dir = '/content/trainntest/USPS_images/train'
test_dir = '/content/trainntest/USPS_images/test'
```

فایل زیپ را باز کرده و داده های مربوط به train را در train_dir و داده های مربوط به tset را در test_dir میریزیم.

```
y_train = []
y_test = []
for path in os.listdir(train_dir):
    if os.path.isfile(os.path.join(train_dir, path)):
        y_train.append(int(path[0]))

for path in os.listdir(test_dir):
  if os.path.isfile(os.path.join(test_dir, path)):
      y_test.append(int(path[0]))

x_train = []
for path in os.listdir(train_dir):
  if os.path.isfile(os.path.join(train_dir,path)):
    x_train.append(cv2.cvtColor(cv2.imread(f"{train_dir}/{path}"), cv2.COLOR_RGB2GRAY))

x_test = []
for path in os.listdir(test_dir):
  if os.path.isfile(os.path.join(test_dir,path)):
    x_test.append(cv2.cvtColor(cv2.imread(f"{test_dir}/{path}"), cv2.COLOR_RGB2GRAY))
```

سپس باید ورودی و خروجی شبکه را برای آموزش بسازیم.ورودی ما یعنی x_train عکس های داخل trian را میگیرد و با استفاده از تابع cv2.imread همه ی عکس ها را به آرایه تبدیل میکند و در نهایت با تبدیل به GRAY آرایه ما دوبعدی می شود.
برای خروجی هر عکس که عددی بین 0 تا 9 است را باید از اسم فایل عکس بگیریم که حرف اول آن است و در y_train ذخیره میکنیم.

```
num_classes = 10

y_train_cat = to_categorical(y_train, num_classes)
y_test_cat = to_categorical(y_test, num_classes)
```

در این قسمت از کد ما اعداد خروجیمان را به one hot تبدیل می کنیم.یعنی آرایه ای به طول 10 برای هر عدد میسازیم که همه اعضای آن به جز ایندکس آن عدد را برابر 0 میگذاریم و ایندکس مورد نظر را 1 قرار می دهیم.

```
x_train_final = x_train.reshape(-1 ,16*16) / 255
x_test_final = x_test.reshape(-1 ,16*16) / 255
```

عکس های ورودی مان گفته شد که به صورت ماتریس است که با اعداد 0-256 پر شده است.با تقسیم بر 255 کردن تمامی اعضا ما اعدادی بین 0 و 1 داریم که برای کار با احتمال لازم است.

```
model = Sequential()
model.add(Input(shape = (16*16)))
model.add(Dense(20, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(num_classes , activation = 'softmax'))

model.compile(loss = 'categorical_crossentropy', optimizer = 'adam' , metrics = ['accuracy'])
```

پس از اینکه ورودی و خروجی شبکه را تعیین کردیم باید مدل را بسازیم و شبکه را آموزش دهیم.
تعدادی لایه با تعداد دلخواه نورون به شبکه اضافه می کنیم و لایه اخر به دلیل اینکه خروجی 10 حالت مختلف دارد 10 نورون قرار میدهیم.
برای تابع فعال سازی از softmax استفاده میکنیم زیرا خروجی بیش از 2 حالت است و در این تابع فعال سازی مجموع احتمال تمامی خروجی ها برابر یک است.

```
batch_size = 128
epochs = 30
model.fit(x_train_final, y_train_cat,
          batch_size= batch_size ,
          epochs=epochs, verbose= 1,
          validation_data=(x_test_final,y_test_cat))
```

ما نمی توانیم تمامی داده ها را بصورت یکسره به شبکه برای اموزش بدهیم،برای این کار از batch_size استفاده میکنیم یعنی هر سری 128 داده را به شبکه بدهد و این کار را 30 بار انجام دهد.
در این قسمت ورودی و خروجی شبکه را میدهیم و شبکه عصبی مان آموزش میبیند.

```
Epoch 1/30
57/57 [==============================] - 2s 8ms/step - loss: 1.9295 - accuracy: 0.3054 - val_loss: 1.4685 - val_accuracy: 0.5461
Epoch 2/30
57/57 [==============================] - 0s 4ms/step - loss: 0.9827 - accuracy: 0.7513 - val_loss: 0.7511 - val_accuracy: 0.8122
Epoch 3/30
57/57 [==============================] - 0s 3ms/step - loss: 0.4893 - accuracy: 0.8793 - val_loss: 0.5583 - val_accuracy: 0.8435
Epoch 4/30
57/57 [==============================] - 0s 3ms/step - loss: 0.3474 - accuracy: 0.9136 - val_loss: 0.4760 - val_accuracy: 0.8714
Epoch 5/30
57/57 [==============================] - 0s 3ms/step - loss: 0.2856 - accuracy: 0.9250 - val_loss: 0.4385 - val_accuracy: 0.8884
Epoch 6/30
57/57 [==============================] - 0s 4ms/step - loss: 0.2525 - accuracy: 0.9339 - val_loss: 0.4352 - val_accuracy: 0.8899
Epoch 7/30
57/57 [==============================] - 0s 4ms/step - loss: 0.2279 - accuracy: 0.9406 - val_loss: 0.4117 - val_accuracy: 0.8974
```

در این قسمت میبینم با بار اول آموزش دقت 0.3 و بار دوم 0.7 و به همین صورت دقت رو به افزایش است.

Part6)

در این پارت ما باید شبکه را جوری آموزش دهیم که عکس نویز دار بگیرد و خروجی نویز عکس را کاهش دهد.
ابتدا مشابه بخش قبل داده ها را load کرده و در لیست میریزیم.

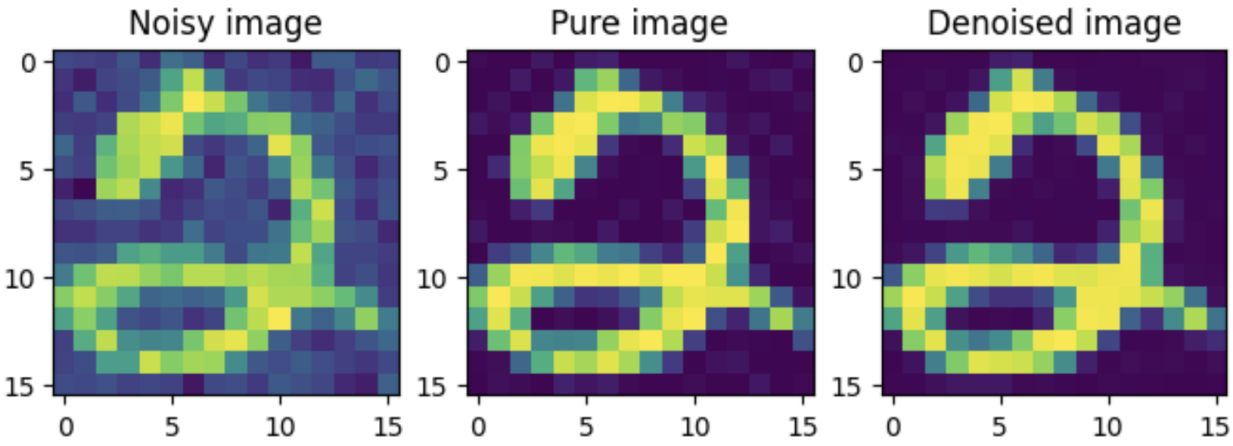
حال باید x_train هایمان که ورودی هستند را نویز دار کنیم همچنین داده های تست را هم باید نویز دار کنیم.
با تغییر دادن noise factor میزان نویز تغییر میکند.

```
pure = input_train
pure_test = input_test
noise = np.random.normal(0, 1, pure.shape)
noise_test = np.random.normal(0, 1, pure_test.shape)
noisy_input = pure + noise_factor * noise
noisy_input_test = pure_test + noise_factor * noise_test
```

```
model.compile(optimizer='adam', loss='binary_crossentropy')
model.fit(noisy_input, pure,
                epochs=no_epochs,
                batch_size=batch_size,
                validation_split=validation_split,
                verbose=0)
```

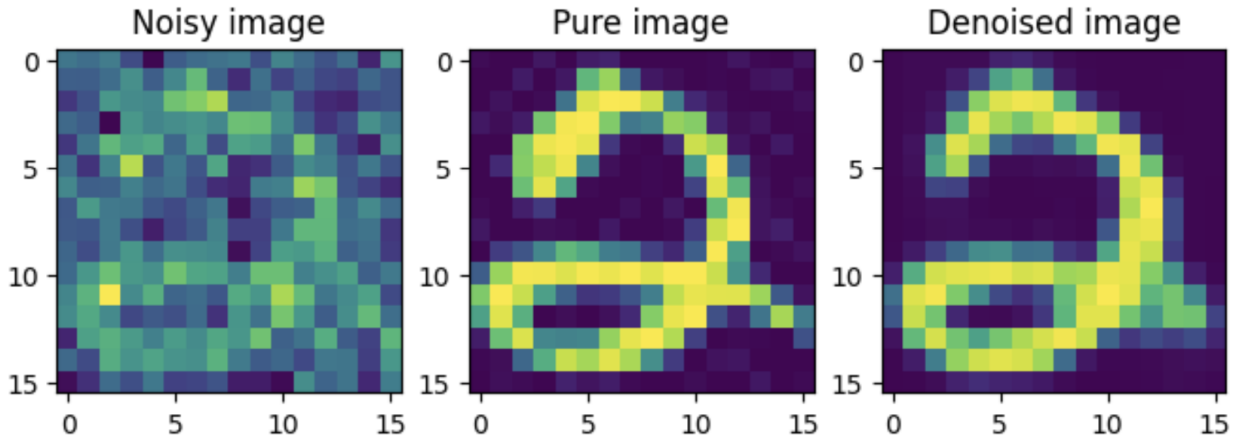
در این قسمت مدل را میسازیم با ورودی عکس های نویز دار و خروجی آن عکس بدون نویز.

target = [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]



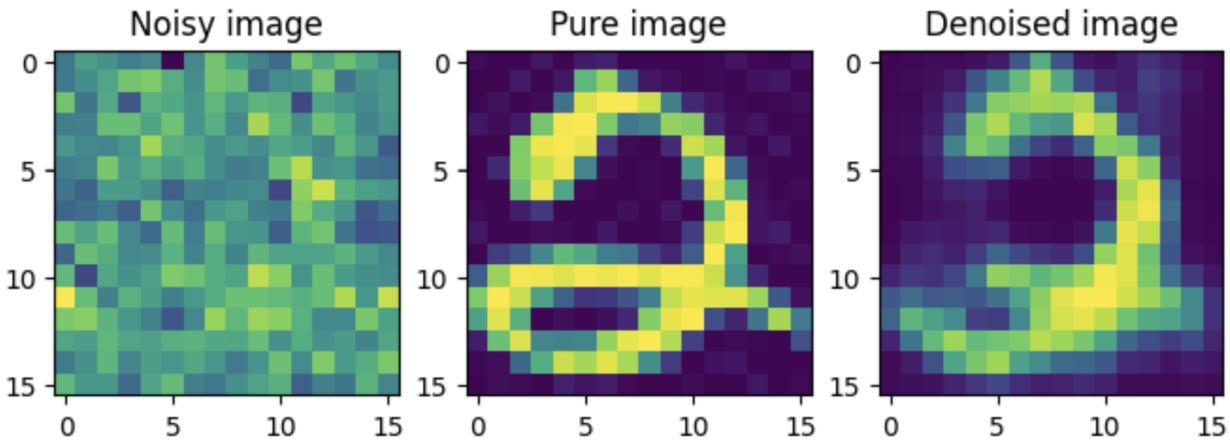
با میزان نویز 0.1 شبکه به خوبی توانسته نویز را برطرف کند.

target = [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]



با میزان نویز 0.4 باز هم شبکه توانسته به خوبی نویز را برطرف کند ولی با دقت کمتر.

target = [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]



با میزان نویز 0.8 میبینیم که عکس بسیار ناواضح است و به خوبی نتوانسته نویز ان را برطرف کند.

علی شیخ عطار
99542222
پروژه ی شبکه ی عصبی
استاد عبدی
1402
دانشکده ی کامپیوتر علم و صنعت