

هوش مصنوعی

پروژه پایانی
دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف
نیم سال اول ۱۰۰۰

اساتید:

جناب آقای دکتر آرش عبدی هجراندوسن

نام و نام خانوادگی:

ایمان علیپور

شماره دانشجویی:

۹۸۱۰۲۰۲۴

۱ مقدمه

در این پروژه قصد داریم در ۶ بخش مختلف، MLP^۱ طراحی کنیم و آنرا آموزش دهیم و با پارامتر های مختلف آن آزمایش هایی انجام دهیم.

۲ آماده سازی اولیه

در ابتدای کد تمامی کتابخانه های مورد نیاز را اضافه می کنیم:

```

1 import numpy as np
2 import random
3 import os
4 import matplotlib as mpl
5 from tensorflow import keras
6 import matplotlib.pyplot as plt
7 from keras.optimizers import *
8 from keras.losses import *
9 from sklearn.model_selection import KFold
10 from keras.layers import LeakyReLU, Dense
11 import tensorflow
12 import tensorflow as tf
13 from sklearn.model_selection import KFold
14 import matplotlib.pyplot as plt
15 import skimage
16 import requests
17 from io import BytesIO
18 from PIL import Image, ImageFilter
19 from matplotlib import cm
20
21 !pip install ipython-autotime
22 %load_ext autotime

```

۳ بخش اول

۱.۳ کدهای مربوط به MLP

ابتدا کد مربوط به این MLP را مینویسیم.

```

1 class Section_1_model:
2     def __init__(self, function=None, number_of_layers=5,
3                  batch_size=1, number_of_epochs=1,

```

¹Multi Layer Perceptron

```

3             loss_function=mean_squared_error ,
dataset_size=20000 ,
4                     train_min_value=1 , train_max_value=5 ,
test_min_value=0 ,
5                     test_max_value=10 ,
number_of_plotted_samples=2000 ,
neuran_scale = 128) :
6             self.number_of_layers = number_of_layers
7             self.batch_size = batch_size
8             self.number_of_epochs = number_of_epochs
9             self.loss_function = loss_function
10            self.dataset_size = dataset_size
11            self.train_min_value = train_min_value
12            self.train_max_value = train_max_value
13            self.test_min_value = test_min_value
14            self.test_max_value = test_max_value
15            self.function = function
16            self.number_of_plotted_samples =
17            number_of_plotted_samples
18            self.X_values = None
19            self.y_values = None
20            self.model = None
21            self.neuran_scale = neuran_scale

```

همانطور که مشاهده میشود، این مدل به عنوان ورودی از ما تعداد fold ها، اندازه epoch ها که همان تعداد گردش هایی است که مدل روی داده انجام میدهد، نوع تابع loss function ، اندازه dataset و بازه های تست و داده های آموزش مدل و همچنین تابعی که بر اساس آن دیتاست تولید میشود را میگیرد، همچنین آپشنی برای افزایش یا کاهش تعداد نورون ها نیز قرار دادم.

حال تابع مختلف این مدل را تشریح میکنم.

```

1     def make_dataset(self , include_noise = False):
2         X_values = np.linspace(self.train_min_value , self.
train_max_value , num=self.dataset_size)
3         y_values = self.function(X_values)
4         if include_noise:
5             y = np.array(list(map(lambda x: self.add_noise(x) ,
y_values)))
6             permuted_data = np.random.permutation(self .
dataset_size)
7             self.X_values = X_values[permuted_data]
8             self.y_values = y_values[permuted_data]

```

این تابع با توجه به تابع گفته شده و سایز دیتاست گفته شده، نقاطی را تولید میکند و اینگونه دیتاست مدل ما ساخته میشود.

```

1  def train(self):
2      k_fold = KFold(n_splits=self.number_of_layers,
3                      shuffle=True)
4
5      loss_in_each_fold = []
6      all_models = []
7      fold_number = 1
8      for train, test in k_fold.split(self.X_values, self.y_values):
9          model = keras.Sequential()
10         model.add(keras.layers.Dense(self.neuran_scale,
11             input_dim=1, kernel_initializer='normal', activation='relu'))
12         model.add(keras.layers.Dense(self.neuran_scale *2,
13             kernel_initializer='normal', activation='relu'))
14         model.add(keras.layers.Dense(self.neuran_scale *4,
15             kernel_initializer='normal', activation='exponential'))
16         model.add(keras.layers.Dense(self.neuran_scale *4,
17             kernel_initializer='normal', activation='relu'))
18         model.add(keras.layers.Dense(1,
19             kernel_initializer='normal', activation='linear'))
20
21         model.compile(loss=self.loss_function, optimizer='adam')
22
23         history = model.fit(self.X_values[train], self.y_values[train],
24                             batch_size=self.batch_size,
25                             epochs=self.number_of_epochs,
26                             verbose=False)
27         scores = model.evaluate(self.X_values[test], self.y_values[test],
28                             verbose=0)
29         loss_in_each_fold.append(scores)
30         all_models.append(model)
31
32         fold_number = fold_number + 1
33
34     print('#####')
35     print('Score per each fold is: ')

```

```

27     for i in range(0, len(loss_in_each_fold)):
28         print('#####')
29         print(f'Fold {i + 1} - Loss: {loss_in_each_fold[i]}')
30         print('#####')
31     print('Average scores for all folds:')
32     print(f'Loss: {np.mean(loss_in_each_fold)}')
33     print('#####')

34
35     best_model_index = np.argmin(loss_in_each_fold)
36     best_model = all_models[best_model_index]
37     self.model = best_model
38     self.best_loss = loss_in_each_fold[best_model_index]
39     self.avg_loss = np.mean(loss_in_each_fold)
40
        return best_model

```

این تابع، قلب اصلی مدل است، با استفاده از روش Cross Validation و با توجه به ورودی ما، دیتا را به بخش هایی تقسیم میکند و آموزش و آزمون را روی آنها انجام میدهد(روی همه بجز یکی آموزش را و validation را روی تکه باقی مانده). این کار را به تعداد epoch هایی که ورودی گرفته شده اند انجام میدهد و در انتهای میزان loss را برای هر fold گزارش میکند و میانگین آنرا نیز همینطور، من برای ساخت این MLP چون ایده ای نداشتیم که باید از کجا شروع کنم، ایده را از این لینک گرفتم و مانند آن عمل کردم و با پارامتر های مختلف آن آزمایش انجام دادم تا به مدل نهایی که در کد آمده رسیدم، در واقع من با activation function های مختلف تست انجام دادم و تعداد نورون هارا تغییر دادم تا به مدل نهایی رسیدم که معمولاً بسیار خوب بود.

```

41     def plot(self):
42         test_range = np.linspace(self.test_min_value, self.
43         test_max_value, self.number_of_plotted_samples)
44         myY = self.model.predict(test_range)
45         actualY = self.function(test_range)
46         plt.plot(test_range, myY, '-r', label = 'prediction
function')
47         plt.plot(test_range, actualY, '--b', alpha=0.5, label
= 'actual function')
48         plt.xlabel('x value')
49         plt.ylabel('y value')
50         plt.legend(loc="upper center", bbox_to_anchor=(0.5,
1.15), ncol=2)
51         plt.show()

```

این تابع عملیات رسم تصاویر و نمودار هارا برای تست انجام میدهد.

۲.۳ آزمایش ها

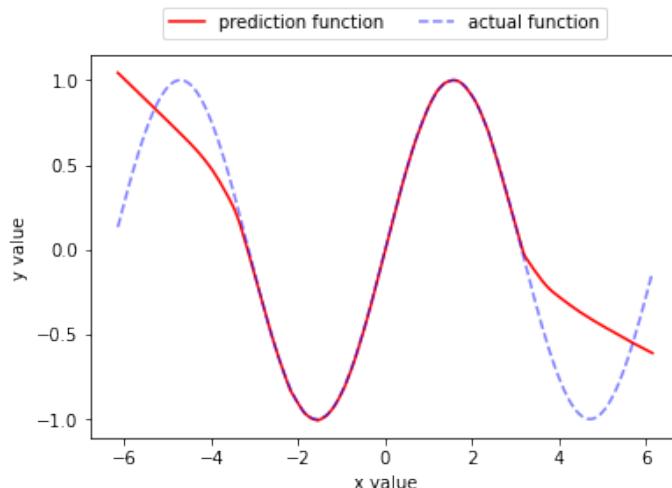
در ابتدا به مدل، تابع $\sin(x)$ را میدهیم و پارامتر های مختلف را هر بار تغییر میدهیم. مدل را اینگونه میسازیم

```

51 regressor = Section_1_model(number_of_layers = 5,
52     number_of_epochs = 10, batch_size = 32,
53         train_min_value = float(-3.14),
54         train_max_value = float(3.14),
55             test_min_value = float(-6.15),
56             test_max_value = float(6.15),
57                 dataset_size = 20000,
58                 number_of_plotted_samples = 2000,
59                     function = lambda x: eval('np.
60             sin(x)'),
61                         loss_function=mean_squared_error
62 )

```

در انتها میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۱ بوده اند و میزان loss برابر با ۰.۰۰۴۱ بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را به خوبی نشان دهد. مجدداً این کار را



شکل ۱ : نمودار یاد گرفته شده و نمودار اصلی

انجام میدهیم با این تفاوت که تعداد گردش ها و اندازه دیتاست را کم میکنیم.
مدل را اینگونه میسازیم

```

57 regressor = Section_1_model(number_of_layers = 5,
58     number_of_epochs = 5, batch_size = 32,
59         train_min_value = float(-3.14),
60         train_max_value = float(3.14),
61 )

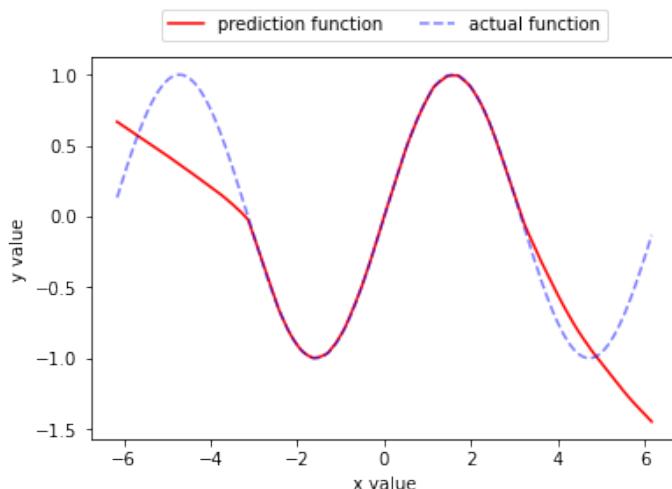
```

```

59             test_min_value = float(-6.15),
60             test_max_value = float(6.15),
61             dataset_size = 5000,
62             number_of_plotted_samples = 2000,
63             function = lambda x: eval('np.
sin(x)'),
64             loss_function=mean_squared_error
)

```

در انتها میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۲ بوده اند و میزان loss برابر با 0.0001 است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دینا فیت شده است ولی خارج آن نتوانسته رفتار را به خوبی نشان دهد.



شکل ۲: نمودار یاد گرفته شده و نمودار اصلی

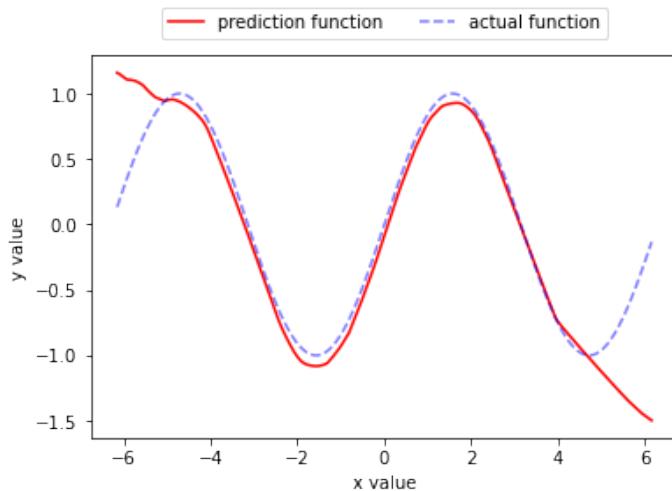
مجدها این کار را انجام میدهیم با این تفاوت که بازه دیتابست را افزایش میدهیم.
مدل را اینگونه میسازیم

```

63 regressor = Section_1_model(number_of_layers = 5,
64                             number_of_epochs = 5, batch_size = 32,
65                             train_min_value = float(-5.14),
66                             train_max_value = float(5.14),
67                             test_min_value = float(-6.15),
68                             test_max_value = float(6.15),
69                             dataset_size = 5000,
70                             number_of_plotted_samples = 2000,
71                             function = lambda x: eval('np.
sin(x)'),
72                             loss_function=mean_squared_error
)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۳ بوده اند و میزان loss برابر با 0.0085 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را به خوبی نشان دهد.



شکل ۳: نمودار یاد گرفته شده و نمودار اصلی

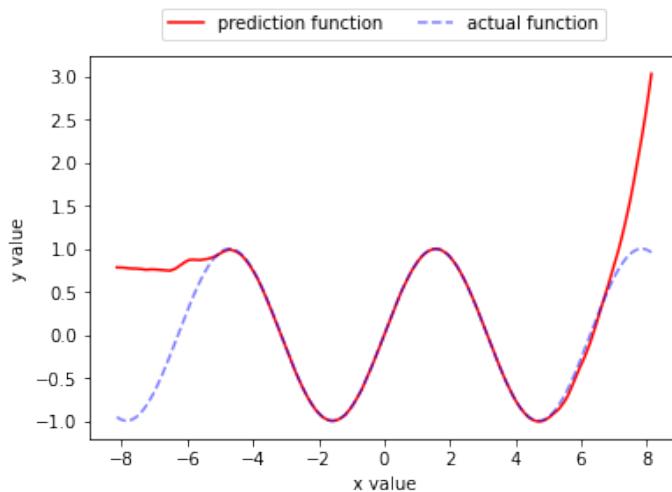
مجددا این کار را انجام میدهیم با این تفاوت که این بار تمام پارامتر های پیشین را افزایش میدهیم.
مدل را اینگونه میسازیم

```

69 regressor = Section_1_model(number_of_layers = 5,
    number_of_epochs = 5, batch_size = 32,
    train_min_value = float(-5.14),
70    train_max_value = float(5.14),
    test_min_value = float(-8.15),
71    test_max_value = float(8.15),
    dataset_size = 20000,
72    number_of_plotted_samples = 2000,
    function = lambda x: eval('np.
73        sin(x)'),
74        loss_function=mean_squared_error
)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۴ بوده اند و میزان loss برابر با 0.0066 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را به خوبی نشان دهد.
مجددا این کار را انجام میدهیم با این تفاوت که این بار تعداد نورون ها را افزایش میدهیم.
مدل را اینگونه میسازیم



شکل ۴: نمودار یاد گرفته شده و نمودار اصلی

```

75 regressor = Section_1_model(number_of_layers = 5,
76                             number_of_epochs = 5, batch_size = 32,
77                             train_min_value = float(-5.14),
78                             train_max_value = float(5.14),
79                             test_min_value = float(-8.15),
80                             test_max_value = float(8.15),
81                             dataset_size = 20000,
82                             number_of_plotted_samples = 2000,
83                             function = lambda x: eval('np.
sin(x)'),
84                             loss_function=mean_squared_error
, neuran_scale = 256)

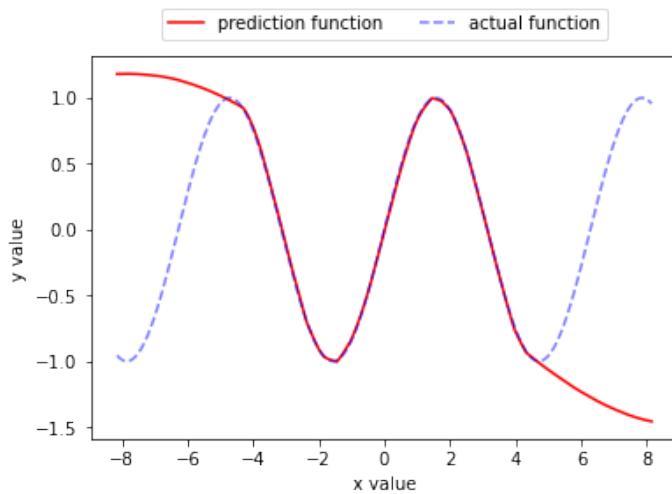
```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۵ بوده
اند و میزان loss برابر با 0.00069 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه
آموختی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را به خوبی نشان دهد.
مجددا این کار را انجام میدهیم با این تفاوت که این بار تعداد نورون ها را افزایش میدهیم.
مدل را اینگونه میسازیم

```

81 regressor = Section_1_model(number_of_layers = 5,
82                             number_of_epochs = 5, batch_size = 64,
83                             train_min_value = float(-5.14),
84                             train_max_value = float(5.14),
85                             test_min_value = float(-8.15),
86                             test_max_value = float(8.15),
87                             dataset_size = 20000,
88                             number_of_plotted_samples = 2000,

```



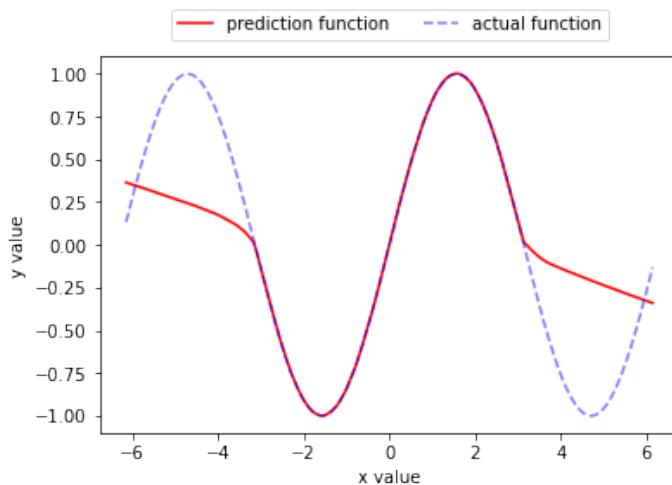
شکل ۵: نمودار یاد گرفته شده و نمودار اصلی

```

85     function = lambda x: eval('np.
  sin(x)'),
86
)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۶ بوده
اند و میزان loss برابر با 0.00069 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه
آموختی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را به خوبی نشان دهد. این بار تفاوت
دیگر در سرعت اجرا بوده است!



شکل ۶: نمودار یاد گرفته شده و نمودار اصلی

مجدها این کار را انجام میدهیم با این تفاوت که این بار تابع ورودی را عوض میکنیم و آنرا برابر با

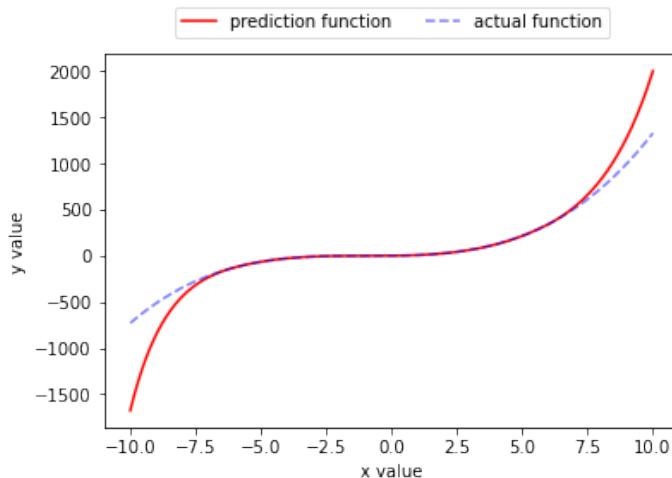
قرار میدهیم.
مدل را اینگونه میسازیم

```

97 regressor = Section_1_model(number_of_layers = 5,
98                             number_of_epochs = 5, batch_size = 32,
99                             train_min_value = float(-6),
100                            train_max_value = float(6),
101                            test_min_value = float(-10),
102                            test_max_value = float(10),
103                             dataset_size = 20000,
104                            number_of_plotted_samples = 2000,
105                            function = lambda x: eval('x
106                            **3+3*x**2+3*x+1'),
107                            loss_function=mean_squared_error
108                            , neuran_scale = 128)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۷ بوده اند و میزان loss برابر با ۱.۰۸ است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموختی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است.



شکل ۷: نمودار یاد گرفته شده و نمودار اصلی

مجدها این کار را انجام میدهیم با این تفاوت که این بار تابع ورودی را عوض میکنیم و آنرا برابر با $x + \sin(x)$ قرار میدهیم.
مدل را اینگونه میسازیم

```

93 regressor = Section_1_model(number_of_layers = 5,
94                             number_of_epochs = 5, batch_size = 32,
95                             train_min_value = float(-6),
96                             train_max_value = float(6),

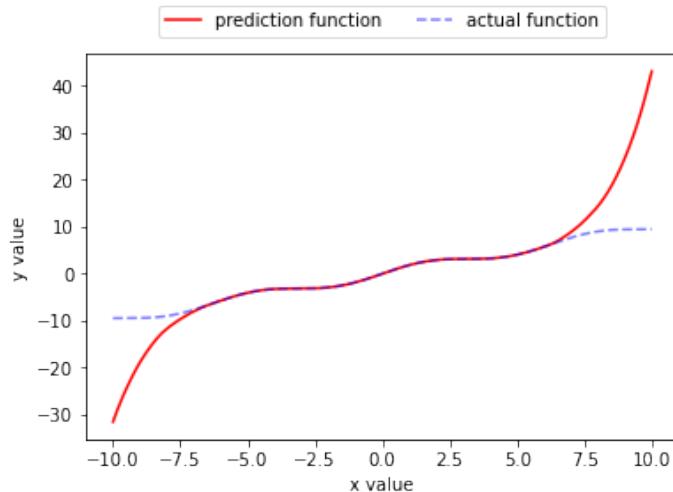
```

```

95             test_min_value = float(-10),
96             test_max_value = float(10),
97             dataset_size = 20000,
98             number_of_plotted_samples = 2000,
99             function = lambda x: eval('x+(np
100            .sin(x))'),
101            loss_function=mean_squared_error
102            , neuran_scale = 128)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل بوده اند و میزان loss برابر با 0.005 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است.



شکل ۸: نمودار یاد گرفته شده و نمودار اصلی

در انتهای به عنوان آخرین آزمایش این کار را انجام میدهیم با این تفاوت که این بار تابع ورودی را عوض میکنیم و آنرا برابر با $x^2 \times \ln(x)$ قرار میدهیم.
مدل را اینگونه میسازیم

```

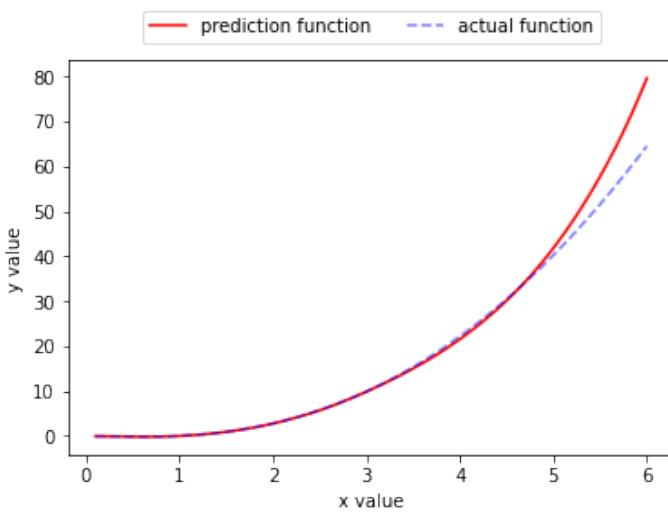
99 regressor = Section_1_model(number_of_layers = 5,
100   number_of_epochs = 5, batch_size = 32,
101   train_min_value = float(-6),
102   train_max_value = float(6),
103   test_min_value = float(-10),
104   test_max_value = float(10),
105   dataset_size = 20000,
106   number_of_plotted_samples = 2000,
107   function = lambda x: eval('x+(np
108     .sin(x))'),
109

```

104

```
loss_function=mean_squared_error
, neuran_scale = 128)
```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۹ بوده اند و میزان loss برابر با 0.001 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است.



شکل ۹: نمودار یاد گرفته شده و نمودار اصلی

۳.۳ نتایج

با آزمایش های من، در نهایت به این نتیجه رسیدم که با افزایش تعداد گذرها، دقت بیشتر میشود، همچنین با افزایش تعداد لایه ها و نورون ها هم همینطور، اما زمان یادگیری هم بسیار زیاد میشود، با افزایش دامنه به تنها یک دقت کم میشود که اگر پارامتر های دیگر را افزایش دهیم باعث میشود دقت در این محدوده ها هم قابل پیشビینی تر بشود، در کل با افزایش پیچیدگی تابع، مخصوصاً توابع متناوب یادگیری آنها سخت تر میشود و پیشビینی رفتار آنها به میزان داده بیشتری نیاز دارد و این نوع شبکه ها عصبی احتمالاً توانایی یادگیری آنها را ندارند، زمانی که رفتار تابع خطی است، یادگیری به خوبی انجام میشود و پیشビینی ها بسیار قابل قبولند، در کل همواره مدل من در بازه یادگیری به خوبی تابع را پیشビینی میکرد و فیت میشد اما خارج بازه اینطور نبود و پیشビینی ها دلچسب نبودند که البته طبیعی است.

۴ بخش دو

در این بخش بعضی تابع سازنده دیتاست را کمی تغییر میدهیم تا به دیتاست نویز اضافه کند.

105

```
class Section_2_model:
```

```

106     def __init__(self, function=None, number_of_layers=5,
107         batch_size=1, number_of_epochs=1,
108             loss_function=mean_squared_error,
109         dataset_size=20000,
110             train_min_value=1, train_max_value=5,
111         test_min_value=0,
112             test_max_value=10,
113         number_of_plotted_samples=2000,
114             neuran_scale = 128, noise_mult = 0):
115         self.number_of_layers = number_of_layers
116         self.batch_size = batch_size
117         self.number_of_epochs = number_of_epochs
118         self.loss_function = loss_function
119         self.dataset_size = dataset_size
120         self.train_min_value = train_min_value
121         self.train_max_value = train_max_value
122         self.test_min_value = test_min_value
123         self.test_max_value = test_max_value
124         self.function = function
125         self.number_of_plotted_samples =
126             number_of_plotted_samples
127         self.X_values = None
128         self.y_values = None
129         self.model = None
130         self.neuran_scale = neuran_scale
131         self.noise_mult = noise_mult

```

درواقع فیلد جدید noise mult برای همین است. تابع اضافه کننده نویز هم اینگونه است.

```

127 def add_noise(self, number):
128     noise_multiplier = (random.random() - 0.5) * 2 *
129     self.noise_mult
130     return number * (1 + noise_multiplier)

```

۱.۴ آزمایش ها

از همان تابع $\sin(x)$ در قسمت قبل استفاده میکنیم و به مرور نویز را زیادتر میکنیم.

```

130 noise_mult = 0.1
131 regressor = Section_1_model(number_of_layers = 5,
132     number_of_epochs = 3, batch_size = 32,
133             train_min_value = float(-3),
134             train_max_value = float(3),

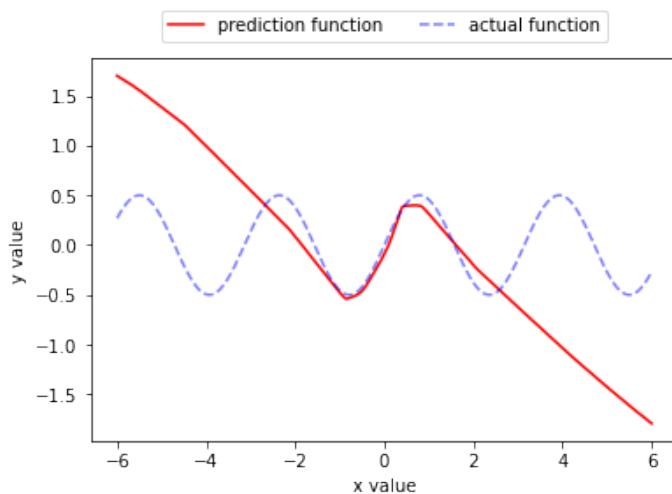
```

```

133                         test_min_value = float(-6),
134                         test_max_value = float(6),
135                         dataset_size = 2000,
136                         number_of_plotted_samples = 500,
137                         function = lambda x: eval('np.
sin(x)*np.cos(x)'), loss_function=mean_squared_error
, neuran_scale = 128, noise_mult=noise_mult)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۱۰ پوده اند و میزان loss برابر با ۰.۰۳ بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است. توجه کنید که میزان loss بسیار زیاد تر شده.



شکل ۱۰ : نمودار یاد گرفته شده با داده های دارای نویز و نمودار اصلی

آزمایش قبل را با نویز بیشتر تکرار میکنیم.

```

138 noise_mult = 1
139 regressor = Section_1_model(number_of_layers = 5,
140                               number_of_epochs = 3, batch_size = 32,
141                               train_min_value = float(-3),
142                               train_max_value = float(3),
143                               test_min_value = float(-6),
144                               test_max_value = float(6),
145                               dataset_size = 2000,
146                               number_of_plotted_samples = 500,
147                               function = lambda x: eval('np.
sin(x)*np.cos(x)'), loss_function=mean_squared_error
, neuran_scale = 128, noise_mult=noise_mult)

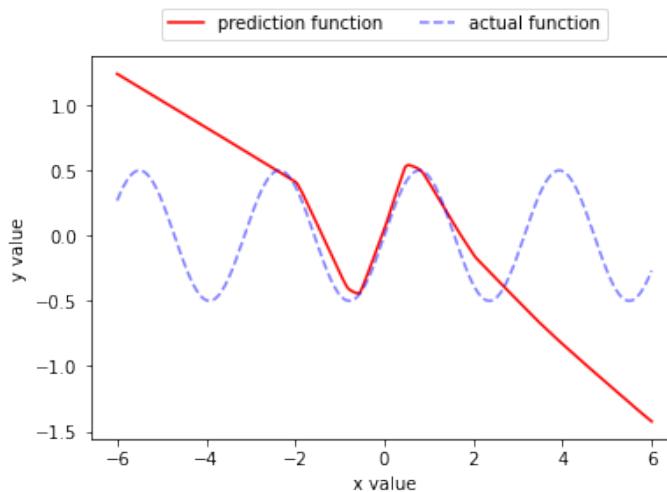
```

```

144           loss_function=mean_squared_error
145   , neuran_scale = 128,
           noise_mult=noise_mult)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۱۲ بوده اند و میزان loss برابر با ۰.۰۳ است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است. توجه کنید که میزان loss بسیار زیاد تر شده.



شکل ۱۱ : نمودار یاد گرفته شده با داده های دارای نویز و نمودار اصلی

آزمایش قبل را مجددا با نویز بیشتر تکرار میکنیم.

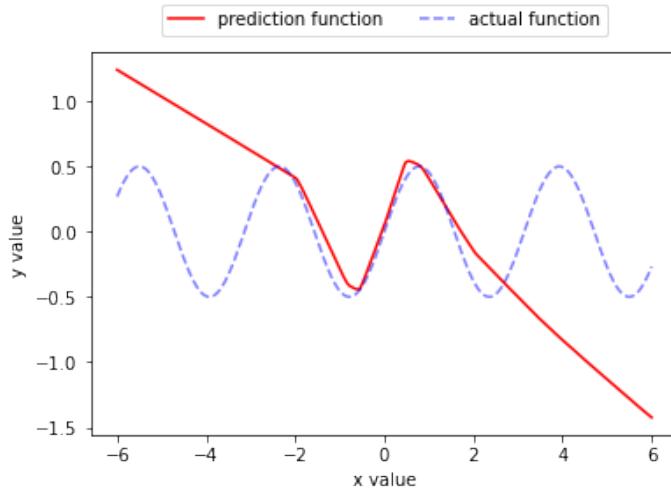
```

146 noise_mult = 0.5
147 regressor = Section_1_model(number_of_layers = 5,
148   number_of_epochs = 3, batch_size = 32,
149   train_min_value = float(-3),
150   train_max_value = float(3),
151   test_min_value = float(-6),
152   test_max_value = float(6),
153   dataset_size = 2000,
   number_of_plotted_samples = 500,
   function = lambda x: eval('np.
     sin(x)*np.cos(x)'),           loss_function=mean_squared_error
   , neuran_scale = 128,           noise_mult=noise_mult)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۱۲ بوده اند و میزان loss برابر با ۰.۰۳۲ است، همانطور که در شکل پیداست، مدل به خوبی در بازه

آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است. توجه کنید که میزان loss بسیار زیاد تر شده.



شکل ۱۲: نمودار یاد گرفته شده با داده های دارای نویز و نمودار اصلی

آزمایش قبل را مجددا با نویز بیشتر تکرار میکنیم.

```

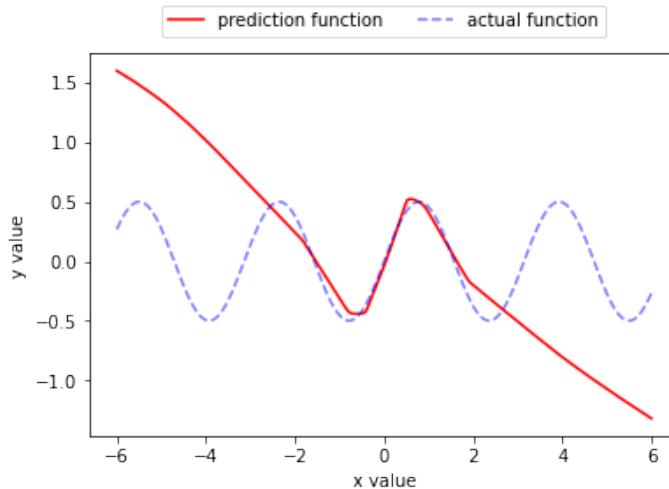
154 noise_mult = 0.5
155 regressor = Section_1_model(number_of_layers = 5,
156     number_of_epochs = 3, batch_size = 32,
157     train_min_value = float(-3),
158     train_max_value = float(3),
159     test_min_value = float(-6),
160     test_max_value = float(6),
161     dataset_size = 2000,
162     number_of_plotted_samples = 500,
163     function = lambda x: eval('np.
164     sin(x)*np.cos(x)'),
165     loss_function=mean_squared_error
166     , neuran_scale = 128,
167     noise_mult=noise_mult)

```

در انتها میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ۱۳ بوده اند و میزان loss برابر با 0.034 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموزشی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است. توجه کنید که میزان loss بسیار زیاد تر شده و از آزمایش قبل هم بیشتر شده است.

آزمایش قبل را مجددا با نویز بیشتر تکرار میکنیم.

```
162 noise_mult = 0.5
```



شکل ۱۳: نمودار یادگرفته شده با داده های دارای نویز و نمودار اصلی

```

163 regressor = Section_1_model(number_of_layers = 5,
164     number_of_epochs = 3, batch_size = 32,
165         train_min_value = float(-3),
166         train_max_value = float(3),
167         test_min_value = float(-6),
168         test_max_value = float(6),
169         dataset_size = 2000,
170         number_of_plotted_samples = 500,
171             function = lambda x: eval('np.
172 sin(x)*np.cos(x)'),
173             loss_function=mean_squared_error
174 , neuran_scale = 128,
175             noise_mult=noise_mult)

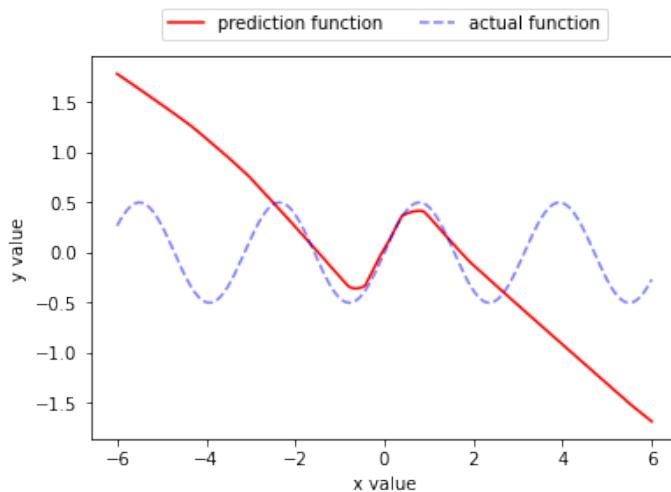
```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یادگرفته شده و تابع اصلی به شکل ۱۴ پوده اند و میزان loss برابر با ۰.۰۲۹ بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموختی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است. این بار به طرز عجیبی میزان loss کاهش یافته! آزمایش قبل را مجدداً با نویز بیشتر تکرار میکنیم.

```

170 noise_mult = 0.5
171 regressor = Section_1_model(number_of_layers = 5,
172     number_of_epochs = 3, batch_size = 32,
173         train_min_value = float(-3),
174         train_max_value = float(3),
175         test_min_value = float(-6),
176         test_max_value = float(6),
177             )

```



شکل ۱۴: نمودار یاد گرفته شده با داده های دارای نویز و نمودار اصلی

```

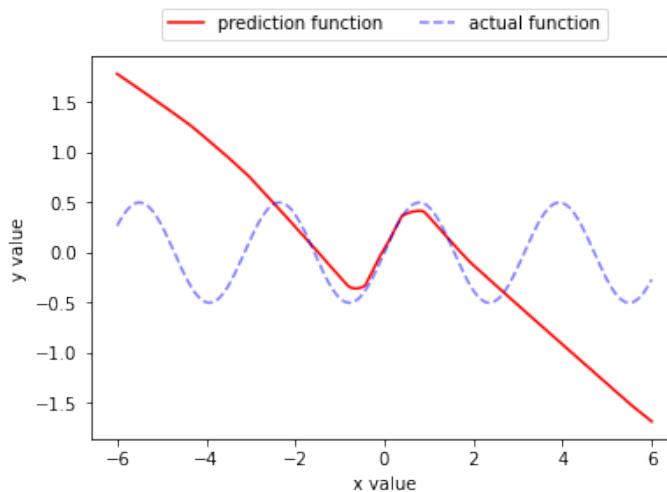
174     dataset_size = 2000,
175     number_of_plotted_samples = 500,
176     function = lambda x: eval('np.
sin(x)*np.cos(x)'),
177     loss_function=mean_squared_error
, neuran_scale = 128,
           noise_mult=noise_mult)

```

در انتهای میبینیم برای این حالت با این پارامتر ها تابع یاد گرفته شده و تابع اصلی به شکل ؟؟ بوده اند و میزان loss برابر با 0.040 بوده است، همانطور که در شکل پیداست، مدل به خوبی در بازه آموختی بر دیتا فیت شده است ولی خارج آن نتوانسته رفتار را بهتر نشان داده است و دلیل آن عدم رفتار نوسانی این تابع بوده است. این بار میزان loss افزایش یافته که طبق انتظار بوده است.

۲.۴ نتایج

انتظار من این بود که با افزایش نویز، فیت شدن در بازه آموختش به خوبی دفعات قبل اتفاق نیفتند که اینگونه هم بود، اما با توجه به شبکه و پارامتر ها حتی با نویز نسبتاً زیاد هم مدل میتوانست به خوبی فیت شود اما خطای داشت و میزان آن هم نسبتاً زیاد بود. در کل به همین علت است که یک مرحله بسیار مهم از جمع آوری دیتا و درواقع سخت ترین و پرهزینه ترین آن همین dataset cleaning است، چراکه داده ها نویز دارند و نویز عملکرد شبکه های عصبی را به شدت تحت تاثیر قرار میدهد. در کل نویز میتواند باعث شود عملیات فیت شدن سخت تر انجام شود که این برای زمان تست خوب است اما بهتر است در زمان آموختش نویز نداشته باشیم و این کار را به نحوی دیگر انجام بدھیم تا در زمان تست نتایج بهتری ببینیم.



شکل ۱۵: نمودار یادگرفته شده با داده های دارای نویز و نمودار اصلی

۵ بخش سوم

۱.۵ تغییرات کد

برای این بخش دوباره توابع را کمی تغییر میدهیم تا امکان استفاده از تابع چند بعدی را داشته باشند، بیشترین تغییر درمورد تابعی است که رسم را انجام میدهد چراکه فضا را به فضای ۳ بعدی میبرد و آنرا رسم میکند

```

178 def plot(self):
179
180     test_range = np.random.uniform(self.test_min_value,
181                                     self.test_max_value,
182                                     (self.
183                                     number_of_plotted_samples, self.dimentent))
184     z_value = self.function(*test_range)
185     predicted_z_value = self.model.predict(test_range)
186     z_value = z_value.reshape(self.
187                               number_of_plotted_samples)
188     predicted_z_value= predicted_z_value.reshape(self.
189                               number_of_plotted_samples)
190     fig = plt.figure()
191     ax = fig.gca(projection='3d')
192     ax.plot_trisurf(test_range[:, 0], test_range[:, 1],
193                     predicted_z_value, alpha=0.6, antialiased = True,
194                     cmap = plt.get_cmap('cool'))

```

```

190     ax.plot_trisurf(test_range[:, 0], test_range[:, 1],
191                     z_value, alpha=0.5, antialiased = True,
192                     cmap = plt.get_cmap('hot'))
193     ax.set_xlabel('X-axis', fontweight ='bold')
194     ax.set_ylabel('Y-axis', fontweight ='bold')
195     ax.set_zlabel('Z-axis', fontweight ='bold')
196     ax.set_title('my model Vs. actual function')
197     plt.show()

```

۲.۵ آزمایش ها

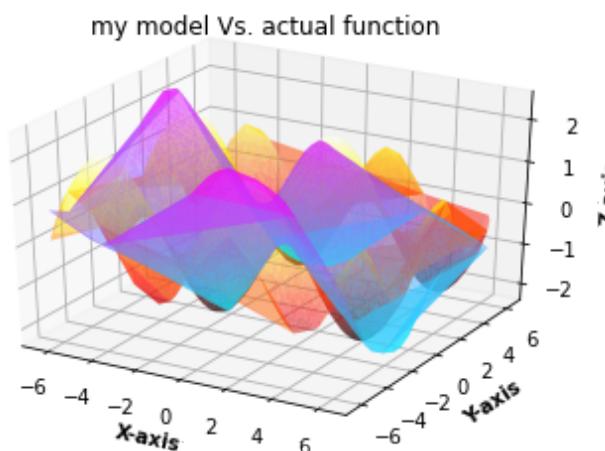
در همه آزمایش ها از تابع $\sin(x) + \sin(y)$ استفاده میکنیم و پارامتر هارا تغییر میدهیم.

```

197 regressor = Section_3_model(number_of_layers = 5,
198                               number_of_epochs = 5, batch_size = 32,
199                               train_min_value = [-np.pi, -np.
200                               pi], train_max_value = [np.pi, np.pi],
201                               test_min_value = [-2 * np.pi, -2
202                               * np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
203                               dataset_size = 2000,
204                               number_of_plotted_samples = 4000,
205                               function = data_generator,
206                               loss_function=mean_squared_error
207 )

```

شکل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss برابر با 0.019 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۱۶ هم این مشاهده میشود اما درک آن سخت است..

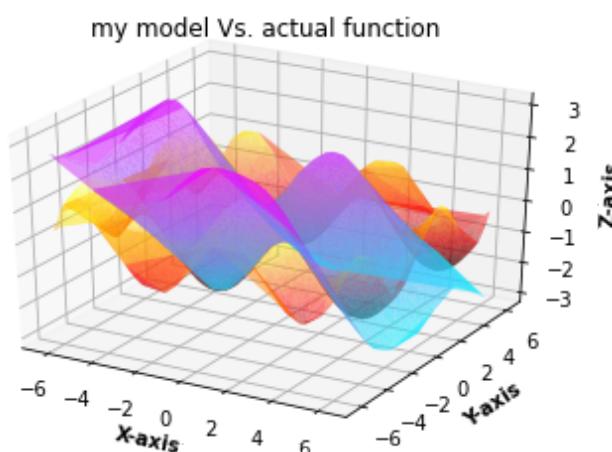


شکل ۱۶: نمودار یادگرفته شده و نمودار اصلی

حال این کار را با پارامتر های دیگری انجام میدهیم، در ابتدا تعداد گذر هارا افزایش میدهیم.

```
203 regressor = Section_3_model(number_of_layers = 5,
204     number_of_epochs = 10, batch_size = 32,
205             train_min_value = [-np.pi, -np.
206 pi], train_max_value = [np.pi, np.pi],
207             test_min_value = [-2 * np.pi, -2
208 * np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
209             dataset_size = 2000,
210 number_of_plotted_samples = 4000,
211             function = data_generator,
212 loss_function=mean_squared_error
213 )
```

شكل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss با 0.030 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۱۷ هم این مشاهده میشود اما درک آن سخت است.



شکل ۱۷: نمودار یادگرفته شده و نمودار اصلی

این یا این کار را افزایش سایز دیتابست انجام میدهیم.

```
209 regressor = Section_3_model(number_of_layers = 5,
210     number_of_epochs = 5, batch_size = 32,
211         train_min_value = [-np.pi, -np.
212     pi], train_max_value = [np.pi, np.pi],
213         test_min_value = [-2 * np.pi, -2
214     * np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
215         dataset_size = 4000,
216     number_of_plotted_samples = 4000,
217         function = data_generator,
```

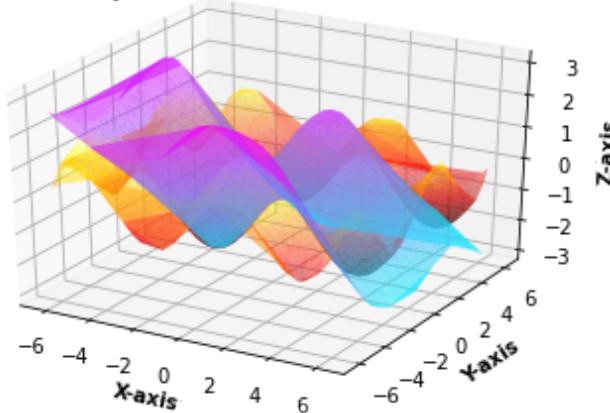
214

```
loss_function=mean_squared_error
```

)

شکل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss برابر با 0.019 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۱۸ هم این مشاهده میشود اما درک آن سخت است.

my model vs. actual function



شکل ۱۸ : نمودار یاد گرفته شده و نمودار اصلی

این بار این کار را با افزایش سایز دیتاست و افزایش تعداد گذرها انجام میدهیم.

215

```
regressor = Section_3_model(number_of_layers = 5,
    number_of_epochs = 10, batch_size = 32,
    train_min_value = [-np.pi, -np.
pi], train_max_value = [np.pi, np.pi],
    test_min_value = [-2 * np.pi, -2
* np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
    dataset_size = 4000,
    number_of_plotted_samples = 4000,
    function = data_generator,
    loss_function=mean_squared_error
)
```

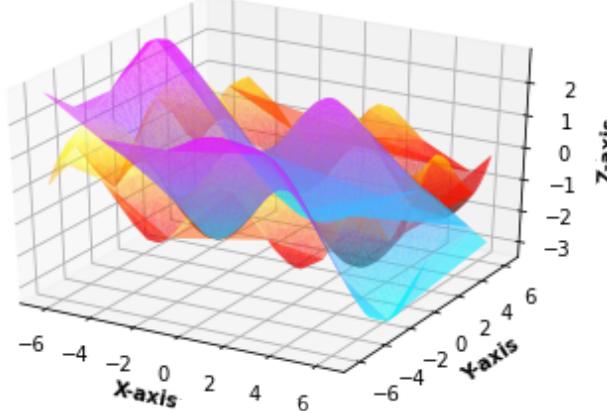
شکل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss برابر با 0.019 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۱۹ هم این مشاهده میشود اما درک آن سخت است.

این بار این کار را با افزایش batch size انجام میدهیم.

221

```
regressor = Section_3_model(number_of_layers = 5,
    number_of_epochs = 5, batch_size = 64,
    train_min_value = [-np.pi, -np.
pi], train_max_value = [np.pi, np.pi],
```

my model Vs. actual function



شکل ۱۹: نمودار یادگرفته شده و نمودار اصلی

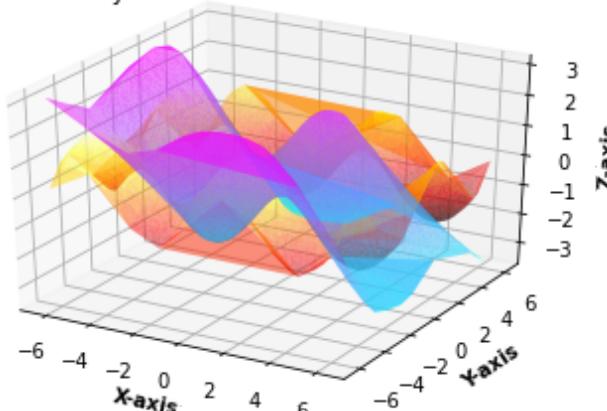
```

223             test_min_value = [-2 * np.pi, -2
224                 * np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
225                     dataset_size = 2000,
226             number_of_plotted_samples = 4000,
227                 function = data_generator,
228             loss_function=mean_squared_error
229         )

```

شکل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss برابر با 0.019 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۲۰ هم این مشاهده میشود اما درک آن سخت است، تفاوت دیگر در سرعت است که این بار سریع تر بوده..

my model Vs. actual function



شکل ۲۰: نمودار یادگرفته شده و نمودار اصلی

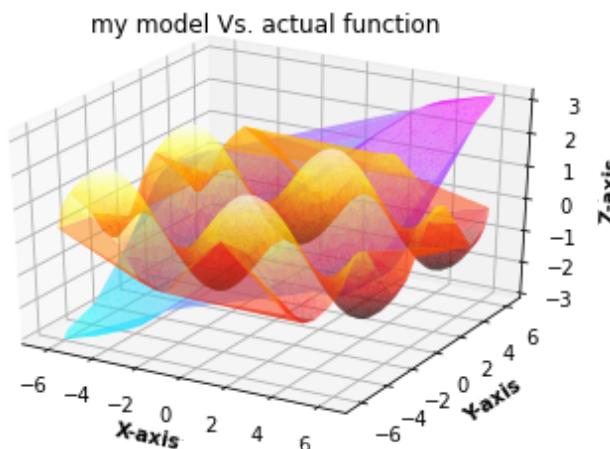
این بار این کار را با افزایش batch size و تعداد گذرها انجام میدهیم.

```

227 regressor = Section_3_model(number_of_layers = 5,
228     number_of_epochs = 5, batch_size = 64,
229             train_min_value = [-np.pi, -np.
230     pi], train_max_value = [np.pi, np.pi],
231             test_min_value = [-2 * np.pi, -2
232     * np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
233             dataset_size = 2000,
234     number_of_plotted_samples = 4000,
235             function = data_generator,
236             loss_function=mean_squared_error
237 )

```

شکل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss کمترین مقدار خود و برابر با 0.018 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۲۱ هم این مشاهده میشود اما درک آن همچنان سخت است.



شکل ۲۱: نمودار یادگرفته شده و نمودار اصلی

این بار این کار را با افزایش batch size و دو پارامتر دیگر انجام میدهیم.

```

233 regressor = Section_3_model(number_of_layers = 5,
234     number_of_epochs = 10, batch_size = 64,
235             train_min_value = [-np.pi, -np.
236     pi], train_max_value = [np.pi, np.pi],
237             test_min_value = [-2 * np.pi, -2
238     * np.pi], test_max_value = [2 * np.pi, 2 * np.pi],
239             dataset_size = 4000,
240     number_of_plotted_samples = 4000,
241             function = data_generator,
242             loss_function=mean_squared_error
243 )

```

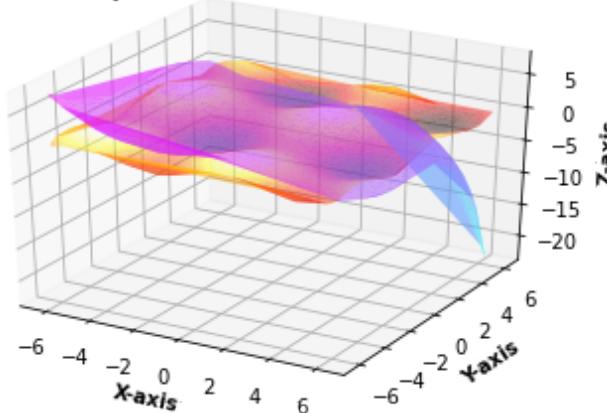
238

```
loss_function=mean_squared_error
```

)

شکل توابع را رسم میکنم، اما مقایسه بر مبنای میزان loss کار ساده تری است، در این حالت مقدار loss کمترین مقدار خود و برابر با 0.0009 است که نشان میدهد مدل به خوبی روی داده فیت شده، در شکل ۲۲ هم این مشاهده میشود اما درک آن همچنان سخت است.

my model vs. actual function



شکل ۲۲: نمودار یاد گرفته شده و نمودار اصلی

۳.۵ نتایج

در کل برای این بخش مدل نیاز داشت تا پیچیده تر شود تا بتواند روی دیتا به خوبی فیت شود، میزان کلی loss هم بیشتر بود و این طبیعی است چراکه خطای این حالت بیشتر است چراکه یک پارامتر اضافه تر داریم. من خیلی با ایجاد توابع سه بعدی مشکل داشتم برای همین فقط از یک تابع استفاده کردم و پیچیدگی تابع را زیاد نکردم.

۶ بخش چهار

۱.۶ تغییرات در کد

کد این بخش بسیار شبیه به کد بخش اول است، فقط من یک فایل تصویری را میخوانم و سعی میکنم بالای آن را به مدل به عنوان داده آموزشی بدهم که کمی کار سختی بود و تخمین زدن شاید ایده بهتری بود. کد تابع ایجاد کننده داده آموزشی به شکل زیر است.

239
240
241
242
243

```
def make_dataset(self, include_noise = False):
    input = Image.open(self.file_address, 'r')
    input = np.array(input.convert('L'))
    self.height, self.width = input.shape
    range_ = np.random.randint(0, self.width, self.width//4)
```

```

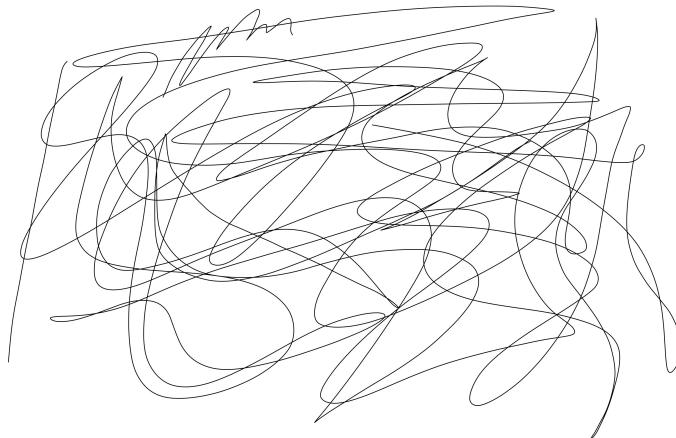
244     y_values = np.array(list(map(lambda x: self.scanner(
245         input, x, self.height), range_)))
246     first_point, last = self.all_valid_points(y_values,
247         self.height)
248     y_values = y_values[first_point:last]
249     range_ = range_[first_point:last]
250     y_values = (self.height - y_values)
251     new_len = len(y_values)
252     permutations = np.random.permutation(new_len)
253     self.X_values = range_[permutations]/self.width
254     self.y_values = y_values[permutations]/self.height

```

این تابع فایل را میخواند و بالاترین نقاط آنرا به عنوان داده آموزشی و دیتاست نگه میدارد.

۲.۶ آزمایش ها

در آزمایش اول شکل ۲۳ را به عنوان ورودی به تابع میدهیم:



شکل ۲۳: ورودی تابع

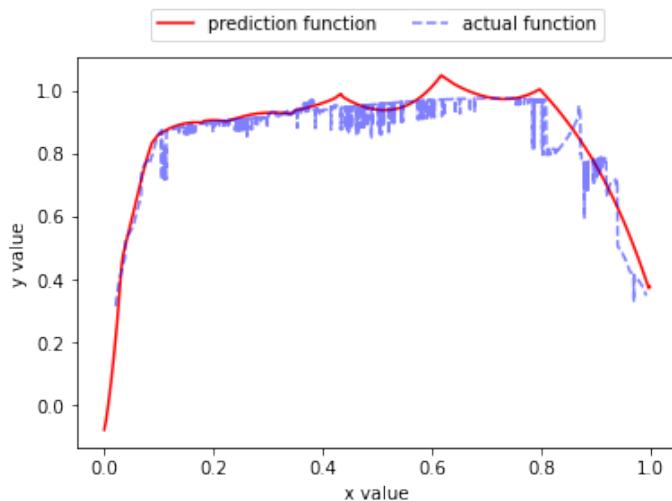
مدل را اینگونه میسازیم:

```

253 regressor = Section_4_model(number_of_layers=5, batch_size
254     =16,
255         number_of_epochs=50,
256         file_address = '/content/Draft-362.jpg',
257         neuran_scale = 40)

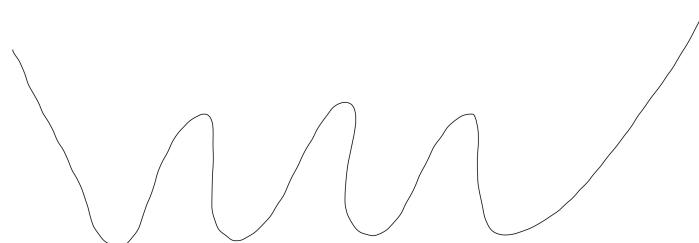
```

حاصل نهایی مشابه شکل ۲۴ است.



شکل ۲۴: خروجی مدل

همانطور که مشاهده میکنید در بازه‌ای که دیتاست بوده مدل تقریبا خوب به شکل اصلی فیت شده و نتایج دلچسب اند اما خارج آن مجددا نتایج معنا نخواهند داشت. قایل توجه است که میزان loss در این حالت برابر با 0.005 بوده است.
این آزمایش را چهار بار دیگر تکرار میکنیم. در آزمایش دوم شکل ۲۵ را به عنوان ورودی به تابع میدهیم:



شکل ۲۵: ورودی تابع

مدل را اینگونه میسازیم:

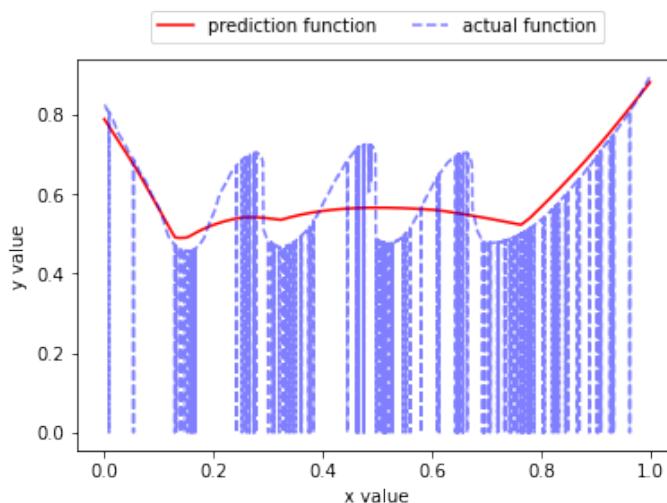
```
257 regressor = Section_4_model(number_of_layers=5, batch_size  
=16,
```

```

258     number_of_epochs=50,
259     file_address = '/content/Draft-363.jpg',
260     neuran_scale = 40)

```

حاصل نهایی مشابه شکل ۲۶ است.



شکل ۲۶: خروجی مدل

همانطور که مشاهده میکنید در بازه‌ای که دیتاست بوده مدل تقریبا خوب به شکل اصلی فیت شده و نتایج دلچسب اند اما خارج آن مجددا نتایج معنا نخواهند داشت. قایل توجه است که میزان loss در این حالت برابر با 0.023 بوده است.

در آزمایش سوم شکل ۲۷ را به عنوان ورودی به تابع میدهیم:
مدل را اینگونه میسازیم:

```

261 regressor = Section_4_model(number_of_layers=5, batch_size
262           =16,
263           number_of_epochs=50,
264           file_address = '/content/Draft-364.jpg',
           neuran_scale = 40)

```

حاصل نهایی مشابه شکل ۲۸ است.

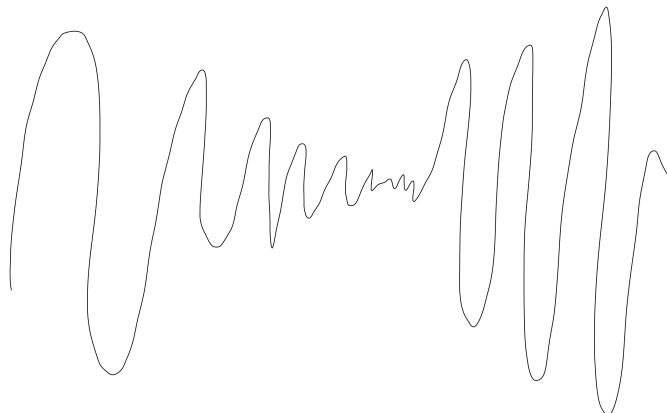
همانطور که مشاهده میکنید در بازه‌ای که دیتاست بوده مدل تقریبا خوب به شکل اصلی فیت شده و نتایج دلچسب اند اما خارج آن مجددا نتایج معنا نخواهند داشت. قایل توجه است که میزان loss در این حالت برابر با 0.041 بوده است.

در آزمایش آخر و چهارم شکل ۲۹ را به عنوان ورودی به تابع میدهیم:
مدل را اینگونه میسازیم:

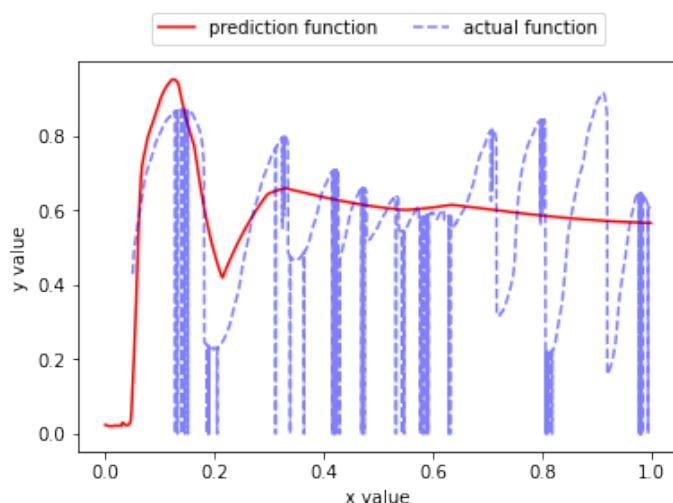
```

265 regressor = Section_4_model(number_of_layers=5, batch_size
266           =16,
           number_of_epochs=50,

```



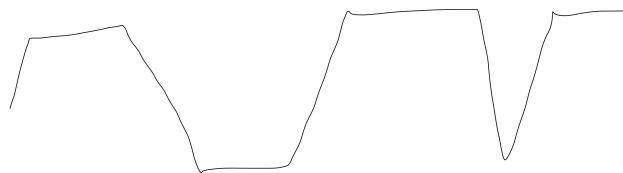
شکل ۲۷: ورودی تابع



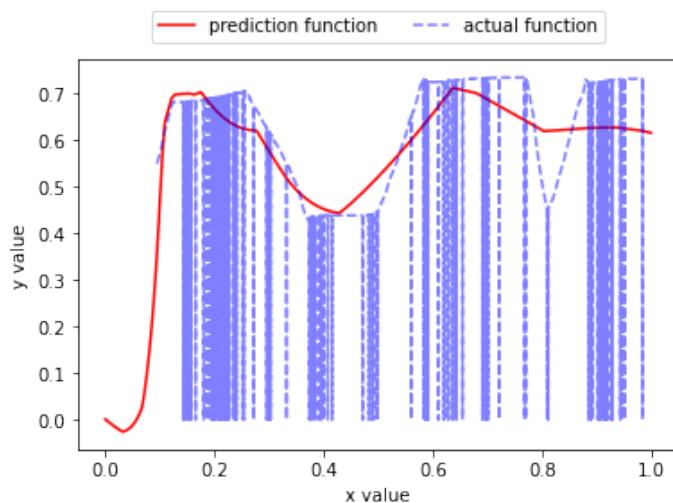
شکل ۲۸: خروجی مدل

```
267     file_address = '/content/Draft-365.jpg',
268     neuran_scale = 40)
```

حاصل نهایی مشابه شکل ۳۰ است.
همانطور که مشاهده میکنید در بازه‌ای که دیتاست بوده مدل تقریباً خوب به شکل اصلی فیت شده و نتایج دلچسب اند اما خارج آن مجدداً نتایج معنا نخواهند داشت. قایل توجه است که میزان loss در این حالت برابر با 0.0203 بوده است.



شکل ۲۹: ورودی تابع



شکل ۳۰: خروجی مدل

۳.۶ نتایج

از آنجا که توابع این بخش خیلی تصادفی بودند انتظار میرفت نتایج به خوبی قسمت های قبل نباشند که اینطور بود، وجود پرسش های ناگهانی را مدل به خوبی نمیتواند تشخیص دهد و خطی با شیب منطقی به آن فیت میکند که کمترین میزان loss را ایجاد کند و دقیقا همین اتفاق افتاده و تغییرات زیاد ناگهانی با خطوط با شیب های منطقی تقریب زده شده اند. چیزی که مهم است این است که تقریب ها خوب بوده اند و شکل کلی نمودار اولیه را میشود فهمید و مدل به آن نزدیک بوده است. توجه کنید تعداد نورون هارا برای این بخش بسیار کم کردم و فکر میکنم با تعداد نورون های بیشتر مدل بتواند خصوصیات بیشتری از دیتاست را یاد بگیرد و خط بهتر و دقیق تری بر آن فیت کند.

۷ بخش پنجم

برای این بخش از دیتاست MNIST استفاده میکنم که از اعداد عکس دارد، دلیل آن هم این است که این دیتاست در کراس موجود است و برای بدست آوردن آن دردسری نداریم و سایز آن هم بسیار بزرگ و خوب است. برای این بخش از GPU هایی که گوگل کولب است استفاده میکنم تا سرعت آموخته زیاد شود و تست های بیشتری انجام دهم.

۱.۷ توضیحات بخش کد

مدل را اینگونه تغییر میدهیم:

```

269         model = keras.Sequential()
270         model.add(keras.layers.Flatten(input_shape=(28,
28)))
271         model.add(keras.layers.Dense(28*28+2, activation
= 'sigmoid'))
272         model.add(keras.layers.Dense(10, activation='
softmax'))
273
274         model.compile(loss=self.loss_function, metrics=[

'accuracy'], optimizer='adam')
275
276         history = model.fit(self.images[train], self.
labels[train],
277                             batch_size=self.batch_size,
278                             epochs=self.number_of_epochs
,
279                             verbose=False)

```

این ساختار را در [لینک](#) دیدم و با کمی تست و ایجاد تغییرات به مدل نهایی ارسال شده رسیدم، مدل من فرم کلی مشابهی دارد اما با مدل نهایی لینک تفاوت های زیادی دارد.
همچنین برای خواندن دیتا داریم:

```

280
281     def load_dataset(self):
282         (train_images, train_labels), (test_images,
283         test_labels) = keras.datasets.mnist.load_data()
284
285         images = np.concatenate((train_images, test_images),
286         axis=0)
287         labels = np.concatenate((train_labels, test_labels),
288         axis=0)
289
290         self.images = images

```

288

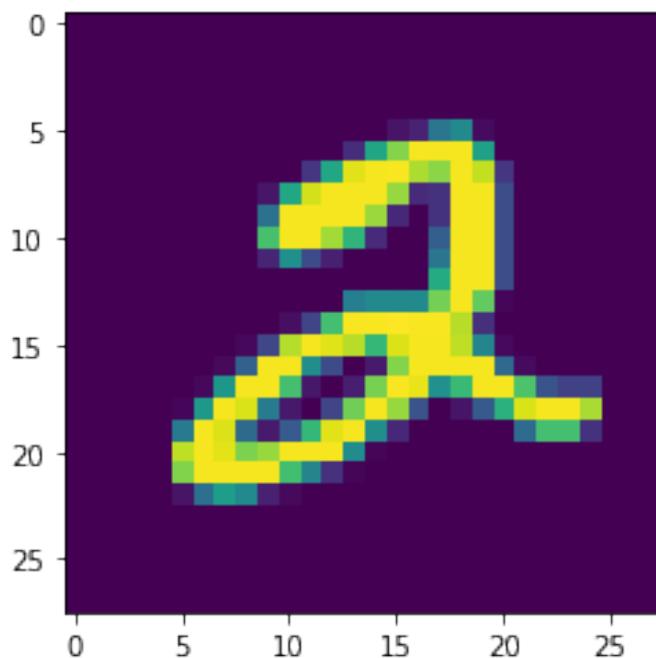
```
self.labels = labels
```

۲.۷ آزمایش ها

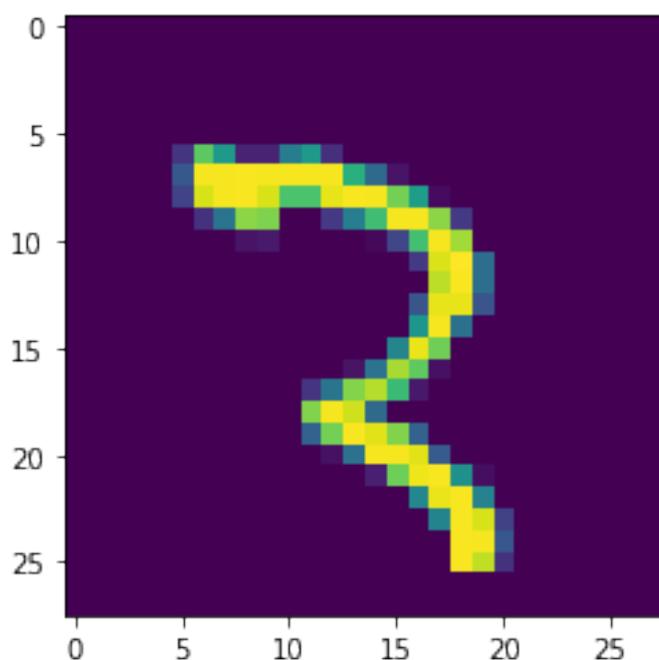
من برای این بخش آزمایش های بسیار زیادی کردم و پارامتر های زیر بهترین پارامتر ها و بهترین نتیجه من بودند، اما بقیه تست هارا نیاوردم، در بسیاری از مواقع دقต به بیش از ۹۰٪ میرسید و این بالاترین دقت من بود. واقعاً این بخش خیلی سخت و اذیت کننده بود و بیشترین زمان را از من گرفت. بعد از آموزش دادن مدل، عکس هایی را نگاه میکنیم و حدس مدل و واقعیت را چک میکنیم، در انتهای دقت مدل به ۹۵٪ رسید که دقت بسیار بالا و خوبی است.

```
289 classifier = Section_5_model(number_of_layers=10, batch_size  
=128,  
290 number_of_epochs=10,  
291 loss_function=  
sparse_categorical_crossentropy)
```

همان طور که در شکل های بعدی میبینید، مدل دقت خوبی دارد، حتی در موردی که حدس آن غلط بوده تشخیص عدد درست برای انسان ها هم سخت است! در شکل ۳۲ که حدس مدل غلط بوده عدد درست ۲ بوده.



شکل ۳۱: حدس مدل عدد ۲ بوده که درست است



شکل ۳۲: حدس مدل عدد ۷ بوده که غلط است

۳.۷ نتایج

من در مرحله تست کردن نمونه های بسیار زیادی را تست کردم تا یک جواب غلط یافتم که این نشان دهنده دقیق بسیار بالای مدل بود، در نهایت هم یک مورد اشتباه و یک مورد درست آوردم. اصلی ترین دستاوردهای این بخش این بود که این را فهمیدم چرا در بخش آخر درس از SVM تعریف شد، پیدا کردن پارامتر های خوب کار بسیار دشوار، خسته کننده و زمان بربی است و بسیار بعید است بهترین حالت با حدس و خطای ایجاد شود، باید الگوریتمی نوشت که مالاند جست و جوی تپه نورده یک اکسترمم محلی پیدا کند و حتی از ایده شبیه سازی ذوب فلزات استفاده کنیم تا شاید اکسترمم محلی بهتری را بیابیم. به طور کلی بنظرم کار کردن و بهینه کردن شبکه های عصبی بسیار دشوار است!

۸ بخش ششم

۱.۸ تغییرات کد

در این بخش مشابه با بخش دوم، بخ داده ها نویز اضافه میکنیم، کد نهایی به شکل زیر است

```

292 def add_noise(self, image, mode):
293     noisy_image = image
294     if mode is not None:
295         if (self.noise_degree == 1):

```

```

296             noisy_image = skimage.util.random_noise(
297     image, mode=mode, clip=True)
298         elif (self.noise_degree == 2):
299             stage_1 = skimage.util.random_noise(image,
300 mode=mode, clip=True)
301                 stage_2 = skimage.util.random_noise(stage_1,
302 mode=mode, clip=True)
303                     noisy_image = skimage.util.random_noise(
304 stage_2, mode=mode, clip=True)
305             elif (self.noise_degree == 3):
306                 stage_1 = skimage.util.random_noise(image,
307 mode=mode, clip=True)
308                     stage_2 = skimage.util.random_noise(stage_1,
309 mode=mode, clip=True)
310                         stage_3 = skimage.util.random_noise(stage_2,
311 mode=mode, clip=True)
312                             stage_4 = skimage.util.random_noise(stage_3,
313 mode=mode, clip=True)
314                                 noisy_image = skimage.util.random_noise(
315 stage_4, mode=mode, clip=False)
316             elif (self.noise_degree == 4):
317                 stage_1 = skimage.util.random_noise(image,
318 mode=mode, clip=True)
319                     stage_2 = skimage.util.random_noise(stage_1,
320 mode=mode, clip=True)
321                         stage_3 = skimage.util.random_noise(stage_2,
322 mode=mode, clip=True)
323                             stage_4 = skimage.util.random_noise(stage_3,
324 mode=mode, clip=True)
325                                 stage_5 = skimage.util.random_noise(stage_4,
326 mode=mode, clip=False)
327                                     noisy_image = skimage.util.random_noise(
328 stage_5, mode='s&p', clip=False)
329             elif (self.noise_degree == 5):
330                 stage_1 = skimage.util.random_noise(image,
331 mode=mode, clip=True)
332                     stage_2 = skimage.util.random_noise(stage_1,
333 mode=mode, clip=True)
334                         stage_3 = skimage.util.random_noise(stage_2,
335 mode=mode, clip=True)
336                             stage_4 = skimage.util.random_noise(stage_3,
337 mode=mode, clip=True)

```

```

319         stage_5 = skimage.util.random_noise(stage_4 ,
320                                         mode=mode , clip=False)
321         stage_6 = skimage.util.random_noise(stage_5 ,
322                                         mode='s&p' , clip=False)
323         stage_7 = Image.fromarray(np.uint8(cm.
324                                         gist_gray(stage_6) * 255))
325         stage_8 = stage_7.filter(ImageFilter.BLUR)
326         stage_9 = np.asarray(stage_8) / 255
327         gray_scale = lambda rgb: np.dot(rgb[...,
328                                         :3] , [0.299 , 0.587 , 0.114])
329         noisy_image = gray_scale(stage_9)
330     return noisy_image

```

این تابع با توجه به ورودی و میزان نویز گفته شده تغییراتی روی عکس میدهد و اینگونه یک دیتاست نویزی میسازیم. حال کافیست به مدل بجای دیتاست اصلی این دیتاست نویزی را بدهیم و بجای لیبل ها، دیتای بدون نویز یا دیتای تغییر نیافته.

۲.۸ آزمایش ها

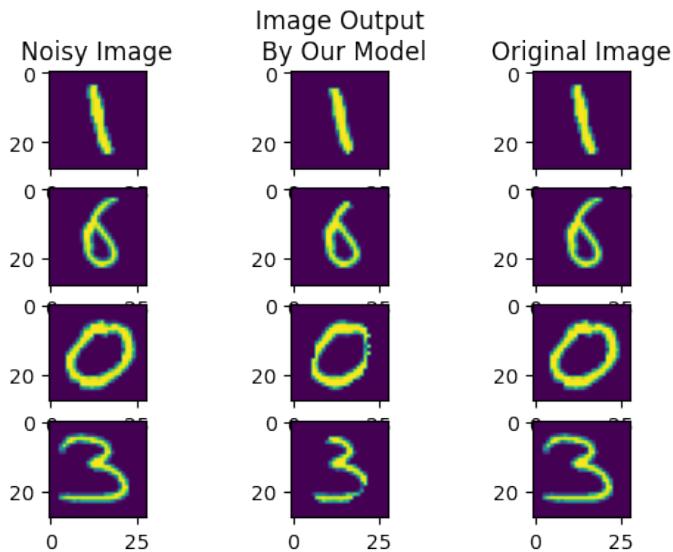
ابتدا مدل را با تست های بسیار زیاد آموزش میدهیم و پارامتر های خوب را پیدا میکنیم!))))

```

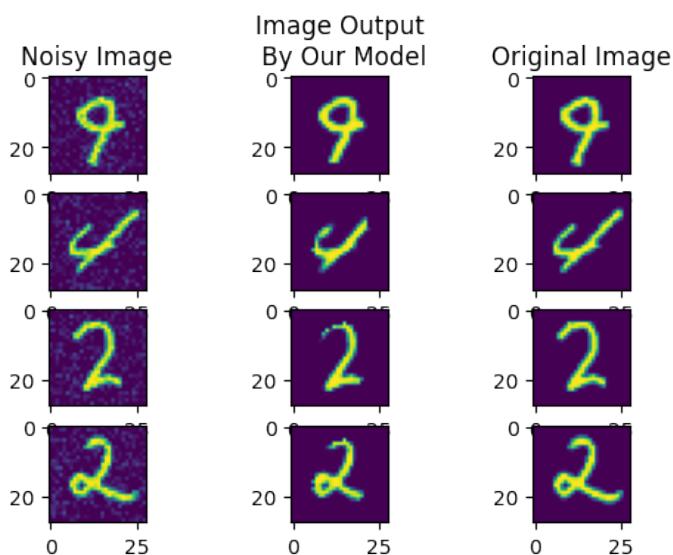
327 classifier_1 = Section_6_model(number_of_epochs=25 , folds=5 ,
328                                 noise_type='localvar' ,
329                                 noise_degree=0)

```

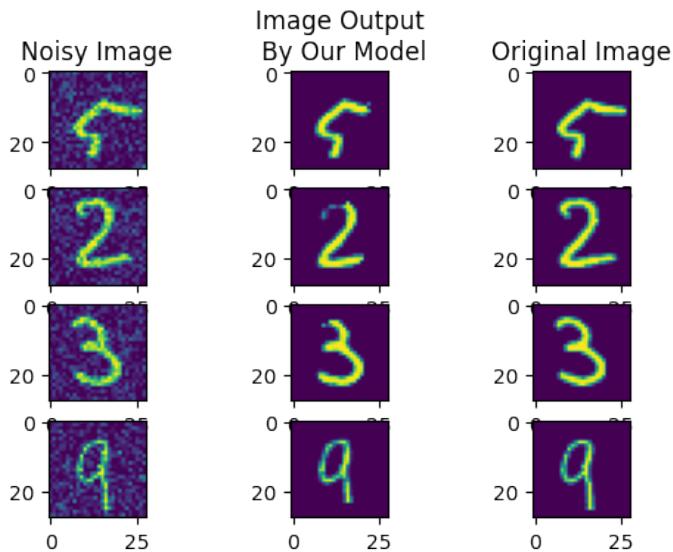
حال برای آزمایش چندین تست را انجام میدهیم و هربار میزان نویز دیتاست را زیادتر میکنیم. آزمایش اول را با دیتاست بدون نویز انجام میدهیم نتایج در شکل ۳۳ قابل مشاهده اند. آزمایش دوم را با دیتاست نویز درجه ۱ انجام میدهیم نتایج در شکل ۳۴ قابل مشاهده اند. آزمایش سوم را با دیتاست نویز درجه ۲ انجام میدهیم نتایج در شکل ۳۵ قابل مشاهده اند. آزمایش چهارم را با دیتاست نویز درجه ۳ انجام میدهیم نتایج در شکل ۳۶ قابل مشاهده اند. آزمایش پنجم را با دیتاست با نویز درجه ۴ انجام میدهیم نتایج در شکل ۳۷ قابل مشاهده اند. آزمایش ششم را با دیتاست با نویز درجه ۵ انجام میدهیم نتایج در شکل ۳۸ قابل مشاهده اند. همانطور که مشاهده میکنید نتایج بسیار خوب و دلچسب اند.



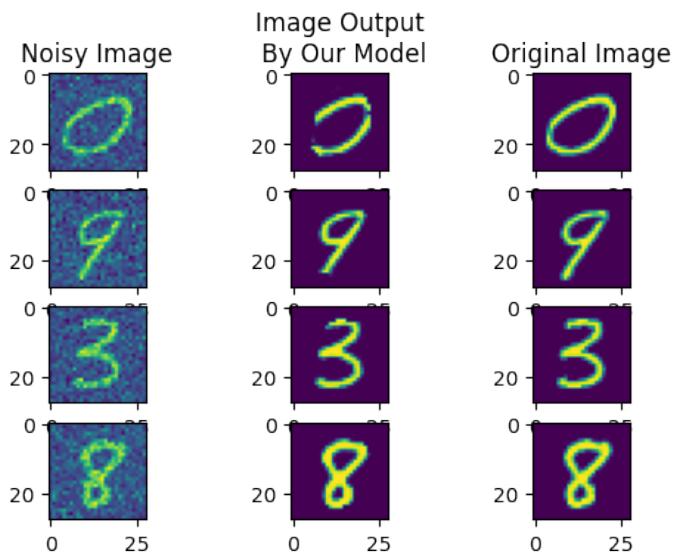
شکل ۳۳: نتایج مدل و عکس اصلی و عکس نویز دار با نویز درجه ۰



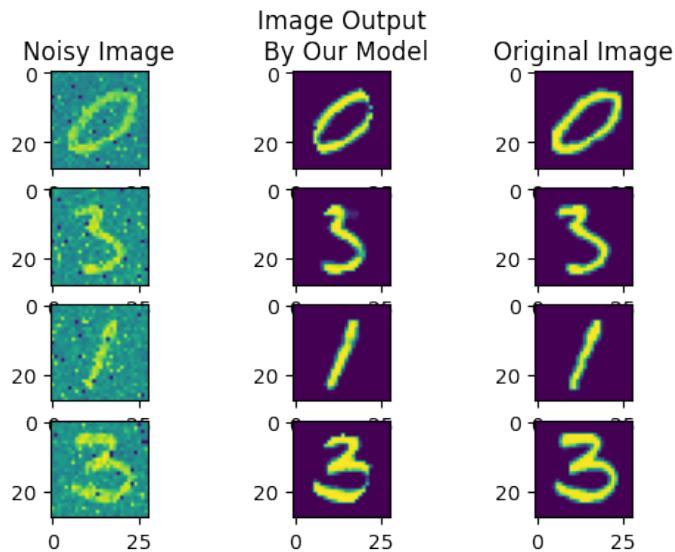
شکل ۳۴: نتایج مدل و عکس اصلی و عکس نویز دار با نویز درجه ۱



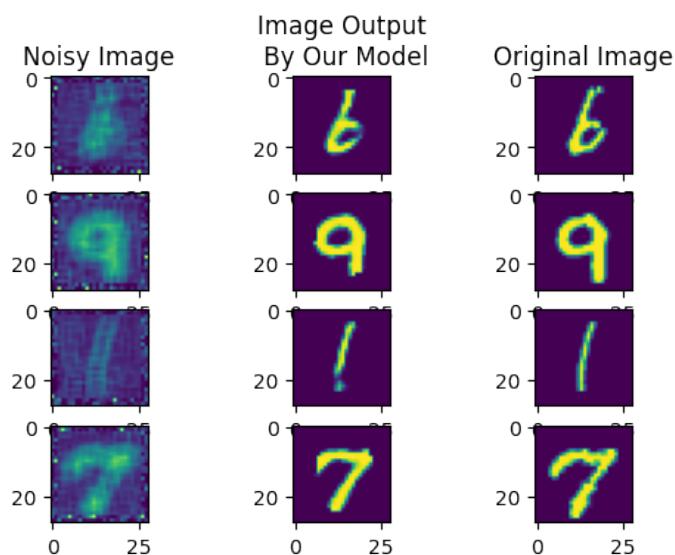
شکل ۳۵: نتایج مدل و عکس اصلی و عکس نویز دار با نویز درجه ۲



شکل ۳۶: نتایج مدل و عکس اصلی و عکس نویز دار با نویز درجه ۳

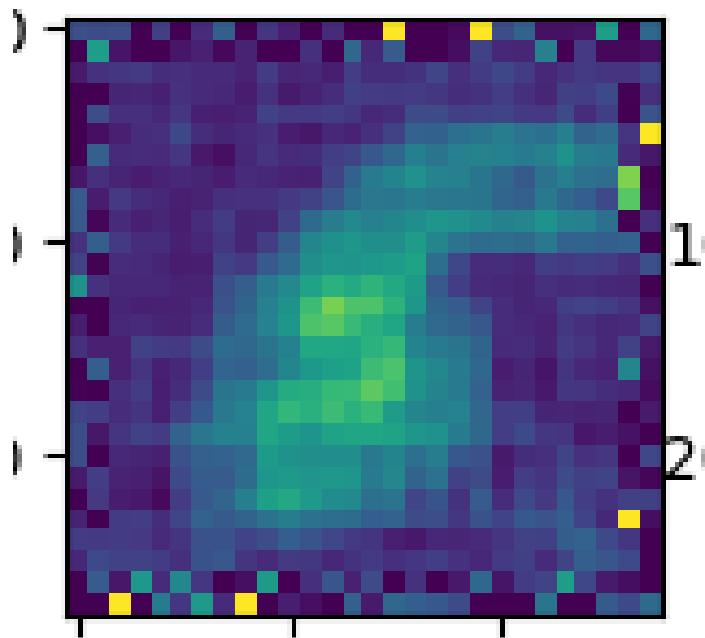


شکل ۳۷: نتایج مدل و عکس اصلی و عکس نویز دار با نویز درجه ۴



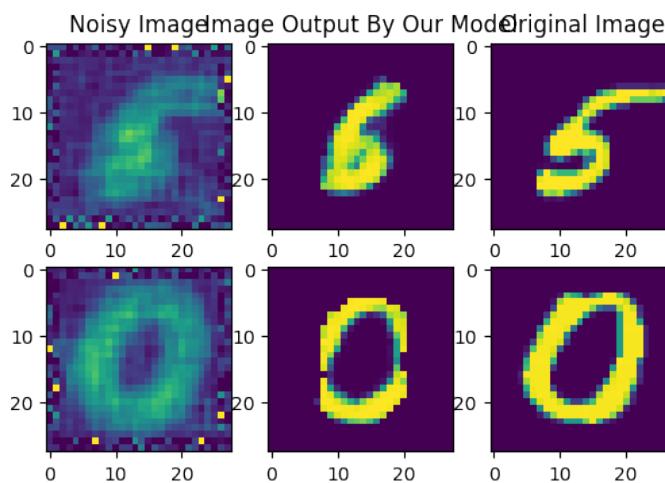
شکل ۳۸: نتایج مدل و عکس اصلی و عکس نویز دار با نویز درجه ۵

حال یک سوال، به نظر شما شکل ۳۹ تغییر یافته و نویزی شده چه عددی بوده است؟



شکل ۳۹: این چه عددی را نشان میدهد؟

جواب درست عدد ۵ است! حتی انسان‌ها هم نمیتوانند این را به درستی تشخیص دهند و. این بسیار برایم جالب بود، در واقع من زمانی که داشتم لبیل هارا درست میکردم شانسی به این برخوردم و برایم جالب بود و آنرا ذخیره کردم. شکل ۴۰ شکل واقعی و کامل است.



شکل ۴۰: جواب درست عدد ۵ است!

۳.۸ نتایج

در این بخش دیدیم چگونه میتوان از این مدل‌ها استفاده کرد و کارهای جالب و متنوعی که در ابتدا به ذهن آدم نمیرسد را با آن انجام داد. این یافته واضحاً به ما میگوید که دنیای شبکه‌های عصبی و توانایی‌های آنها بسیار بزرگ و زیادند. تنها مشکل اصلی یافتن پارامترهای خوب است که کار بسیار بسیار سخت و زمان بری است!

۹ نتیجه گیری نهایی

شبکه‌های عصبی، به ما امکانات بسیار زیادی را میدهد که با روش‌های دیگر مانند الگوریتم‌های ژنتیک و روش‌های دیگر توانایی‌های زیاد اینگونه را ندارند و توانایی‌های آنها محدود‌تر است، حال آنکه میتوان شبکه‌های عصبی را به اندازه کافی پیچیده کرد و با ایده زدن کارایی آنها را بسیار بسیار زیادتر کرد، مثلاً شبکه‌های Recurrent neural network که با ایجاد فلوی گرادیان به نورون‌های قبلی به صورت مستقیم باعث میشوند کارایی به گونه‌ای باشد که حداقل داده‌های تست را به خوبی فیت کند، دقیقاً همان مسئله‌ای که ممکن بود با برخی پارامترها و یا تعداد کمتر نورون به آن برخورد کنیم که مدل حتی دیتاست خودمان را هم نتواند فیت کند. مشکل اصلی این شبکه‌های عصبی این است پارامترهای بسیار زیادی دارند که تون کردن آنها بسیار زمان بر است و حتی توابع فعال‌ساز و متغیرهای دیگر هم بسیار تاثیر گذارند که من در این پروژه بسیاری از آنها با حدس زدن فهمیدم و یا با ایده گرفتن از اینترنت به آنها رسیدم. آموزش این شبکه‌ها هم زمان بسیار زیادی میبرد.

Optimizers and Activation Functions ۱۰

در این بخش درمورد Optimizer هایی که در طول پروژه استفاده کردیم صحبت میکنم.

۱.۱ Optimizer ها

در پلتفرمها و کتابخانه‌های یادگیری ماشین، Optimizer به الگوریتمی گفته می‌شود که بر مبنای آن تغییرات در وزن‌ها به شکل Backpropagation اعمال می‌شود. به عنوان مثال یکی از ابتدایی‌ترین روش‌ها شیوه نزول در راستای گرادیان (Gradient Descent) است که در آن، جهت حرکت وزن‌ها مناسب با گرادیان انتخاب می‌شود.

شیوه معمول Gradient Descent بر مبنای الگوریتم زیر است:

$$\theta = \theta - \alpha \nabla J(\theta)$$

که θ بردار وزن‌ها و α یک ضریب ثابت و J تابع محاسبه Loss و خطأ (MSE و ∇) هم نماد محاسبه گرادیان است. این روش پیاده‌سازی ساده‌ای دارد ولی در صورت زیاد بودن تعداد داده‌ها می‌تواند زمان زیادی طول بکشد و مشکل دیگر هم این است که ممکن است در یک مینیمم محلی گیر بیفتد و نتواند از آن خارج شود.

روش دیگر Stochastic Gradient Descent است که همان روند بالا را اجرا می‌کند، ولی به جای این که بعد از یک دور وارد شدن تمامی نمونه‌ها مقادیر آپدیت شوند، بعد از وارد شدن هر نمونه، کل وزن‌ها آپدیت شده و سراغ نمونه بعدی می‌روند. مشکل این روش این است که نوسانات زیاد می‌شوند و گاهی اوقات هم ممکن است حتی وقتی به بهترین مینیمم (مینیمم سراسری هم رسیده) از آن خارج بشود.

یکی دیگر از مواردی که از توسعه روش GD به دست می‌آید، اضافه کردن مومنتوم به کل سیستم است. در این روش، یک متغیر دیگر به نام $V(t)$ هم تعریف می‌شود (t نشان دهنده شماره iteration اجرا است). محاسبه آن به صورت زیر است:

$$V(t) = \gamma V(t-1) + \alpha \cdot \nabla J(\theta)$$

و آپدیت وزن‌ها با فرمول زیر اتفاق می‌افتد:

$$\theta = \theta - V(t)$$

که در آن γ یک پارامتر است که باید بر اساس مسئله تعیین شود و در شیوه‌های معمول، به طور پیش فرض مقدار 0.9 برای آن در نظر گرفته می‌شود. در این روش، نوسانات کم می‌شود ولی به دست اوردن مقدار درست γ برای الگوریتم کمی دشوار است.

در نهایت باید به روش Adam اشاره کنم که برای بیشتر بخش‌های این پروژه از آن استفاده کرده‌ام و معمولاً از نظر مدت زمان رسیدن به جواب (تعداد Epoch ها و تکرارهای لازم) و همچنین دقیق، عملکرد بهتری نسبت به سایر روش‌ها در بسیاری از مسائل دارد.

در این روش، دو متغیر با نماد V_t و M_t داریم که اولی میانگین وزن‌ها و دومی واریانس آن‌هاست. با کمک آن‌ها دو پارامتر زیر محاسبه می‌شوند:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

و سپس وزن‌ها طبق این فرمول به دست می‌آیند:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

که در شکل معمول آن $0.9 = \beta_1$ و $0.999 = \beta_2$ و همچنینی $10^{-7} = \epsilon$ است. ضمناً η پارامتری (Hyperparameter) است که بسته به مسئله باید تعیین شود و به آن طول گام می‌گویند. در کدهای پایتون Keras، می‌توان آن را با کلمه `learning_rate` تغییر داد. مقدار پیش فرض آن در Keras برابر 0.001 است.

روش دیگری به نام Adam نام Nestrov یا \hat{m} وجود دارد که کلیات آن شبیه روش بالاست ولی محاسبه \hat{v} در آن کمی متفاوت است و بعضی از اوقات بهتر از Adam می‌تواند مسئله را به جواب برساند.

۲.۱۰ Activation Function

در مورد توابع فعال سازی، به جز تابع رایج Sigmoid، توابع مختلف دیگری هم وجود دارند. در کدهای ارسالی، از پنج تابع زیر استفاده شده است:

- Linear این تابع، به شکل خیلی ساده همان x است. یعنی $linear(x) = x$. برای خروجی شبکه‌های مربوط به رگرسیون از این تابع استفاده می‌شود.
- Relu این تابع در اصل به این شکل تعریف می‌شود: $Relu(x) = max(0, x)$ یعنی به ازای مقادیر منفی خروجی آن صفر است و به ازای سایر مقادیر به صورت خطی عمل می‌کند.
- Sigmoid همان تابع سیگموید معروف که به این شکل تعریف می‌شود: $Sigmoid(x) = \frac{1}{1+e^{-x}}$
- Exponential این همان تابع e^x است و در شبکه‌های رگرسیون و تخمین تابع استفاده شده است و دلیل اصلی آن هم این بوده که اگر تابع نمایی داده می‌شد، لایه‌های Relu نمی‌توانستند به تنها ی خیلی خوب عمل کنند ولی لایه Exponential این مشکل را رفع می‌کند.
- Softmax عملکرد آن بدین صورت است که اگر یک بردار $(x_1, x_2, x_3, \dots, x_n)$ به آن بدھیم، خروجی خواهد داشت که خروجی i ام آن به صورت زیر محاسبه می‌شود:

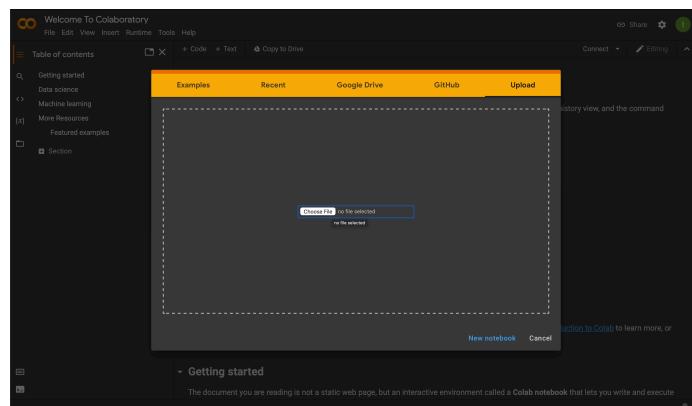
$$\frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

و به نوعی احتمال نمایی نرمالایز شده هر حالت را مشخص می‌کند. از این تابع در شبکه‌های Classifier و دسته بندی استفاده می‌شود.

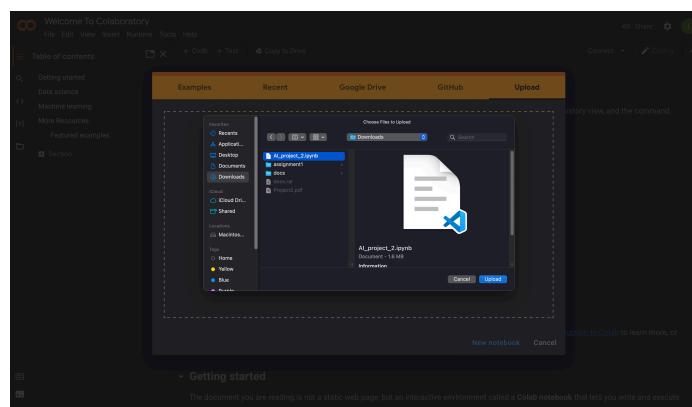
۱۱ راهنمای اجرای کد

من همه این کدهارا در محیط google colab زدم و هیچ کتابخانه ای را روی لپ تاپ خودم نصب نکردم، همچنین از قابلیت GPU بی که این سایت به ما میدهد استفاده کردم، برای اجرای کدها به روش زیر و طبق عکس ها عمل کنید.

- ابتدا به سایت colab.research.google.com یعنی این [لینک](#) بروید و گزینه upload را انتخاب کنید و فایل ipynb من را مانند شکل ۴۱ و شکل ۴۲ در آنجا آپلود کنید.



شکل ۴۱: رفتن به سایت



شکل ۴۲: آپلود فایل

- حال از سربرگ runtime گزینه change runtime type را انتخاب کنید و از آنجا برای گزینه acceleratoor GPU را انتخاب کنید، همانند شکل ۴۳ و شکل ۴۴.
- در انتهای فایل های عکس درون پوشه content را در سایت طبق شکل ۴۵ و شکل ۴۶ آپلود کنید.
- سلول بخش صفر را ران کنید تا کتابخانه ها ایمپورت شوند.

The screenshot shows the Jupyter Notebook interface with the 'Runtime' menu open. The 'GPU' option is highlighted, indicating it is selected as the runtime type.

شکل ۴۳: انتخاب شتاب دهنده

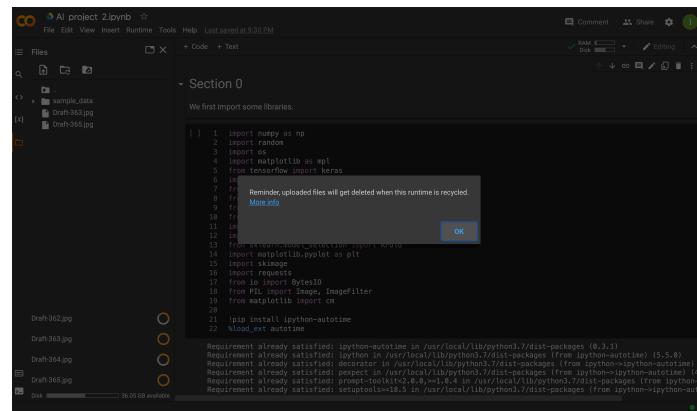
The screenshot shows the 'Notebook settings' dialog box open, with the 'GPU' option selected under the 'Runtime' tab. Other options like 'None' and 'TPU' are also visible.

شکل ۴۴: روشن کردن شتاب دهنده

The screenshot shows the Jupyter Notebook interface with the 'content' folder expanded. Inside the folder, there are four image files: 'Dish_001.jpg', 'Dish_002.jpg', 'Dish_003.jpg', and 'Dish_004.jpg'. These images are likely the training data used for the AI project.

شکل ۴۵: آپلود عکس های بخش چهار

با انجام اینها حال میتوانید کد های هر بخش را به دلخواه ران کنید و همه چیز باید بدون مشکل پیش برود.



شکل ۴۶: آپلود عکس های بخش چهار