گزارش پروژه شبکه عصبی

ریحانه باقری

۹۸۳۱۰۱۲

قسمت اول

ابتدا برای لود کردن داده ها در دو ماتریس train و test ، ۴ بار تابع لود را برای هر دسته عکس صدا کردیم و به عنوان ورودی نام فایل را همراه با شماره نوع آن ارسال کردیم .

اگر شماره برابر ۱ بود به ماتریس لیبل سطر ۱۰۰۰ را اضافه میکنیم که همان گروه airplane می باشد . اگر برابر با ۲ بود به ماتریس لیبل سطر۰۱۰۰را اضافه میکنیم که همان گروه automobileمی باشد . اگر برابر با ۳ بود به ماتریس لیبل سطر۰۰۱۰را اضافه میکنیم که همان گروهbirdمی باشد . اگر برابر با ۴ بود به ماتریس لیبل سطر۰۰۰۱را اضافه میکنیم که همان گروه catمی باشد .

در قسمت دوم نیز تصاویر را خاکستری میکنیم

در قسمت سوم و چهارم و پنجم نیز طبق تعریف پیش میرویم

برای شافل کردن از تابع زیر استفاده کردیم تا ترتیب لیبل ها با دیتاها یکی باشد

```
def shuffle_unison(array1, array2):
  shuffler = np.random.permutation(len(array1))
  array1_shuffled = array1[shuffler]
  array2_shuffled = array2[shuffler]
  return array1_shuffled, array2_shuffled
```

قسمت دوم feedforward

طبق تعریف پروژه پیش میرویم تابع سیگموید را برای محاسبه تابع فعالیت تعریف کرده و مقادیر رندوم را به بایاس ها و وزن ها اختصاص داده ایم :

```
def sigmoid(x):
    ans = 1 / (1 + np.exp(-x))
    return ans


# creating the matrix of weights randomly and the biases with all zeros.
W0 = np.random.randn(16, 1024)
W1 = np.random.randn(16, 16)
W2 = np.random.randn(4, 16)
b0 = np.zeros((16, 1))
b1 = np.zeros((16, 1))
b2 = np.zeros((4, 1))
```

```
#feedforward
minimize_train_set = train_data_shuffled[:200]
# feedforward for all datas
def feedforward(W0,W1,W2,b0,b1,b2):
  counter = 0
  for i in range(len(minimize_train_set)):
      # label and fetures of one input
      reshape_train = minimize_train_set[i].reshape(-1,1)
      # calculate precptron output with activiation sigmoid function
      S0 = reshape_train
      S1 = sigmoid(W0 @ S0 + b0)
      S2 = sigmoid(W1 @ S1 + b1)
      S3 = sigmoid(W2 @ S2 + b2)
      # find  index of maximum  of output
      # find index of maximum of input label
      if  np.argmax(S3) == np.argmax(train_label_shuffled[i],axis=0) :
          # if  correct counter++
          counter += 1
  # print accuracy and time of execution
  print("Accuracy is : " + str(counter*100 / 200))


feedforward(W0,W1,W2,b0,b1,b2)
```

```
Accuracy is : 20.5
```

توضیح: ۲۰۰ داده را به صورت رندوم برمیداریم (قبلا شافل شده)سپس روی آن پیمایش کرده و خروجی را برای همه آن ورودی ها محاسبه میکنیم.اگر با برچسب داده شده یکی بود ، به شمارنده خود اضافه میکنیم.

دقت نهایی برابر است با :

۲۰/۵

قسمت ۳ : back propagation

```
# Backpropagation
from datetime import datetime
start_time = datetime.now()
def merge(data,label):
  set=[]
  for i in range(label.T.shape[1]):
      set.append((data.T[:, i].reshape(1024, 1), label.T[:, i].reshape(4, 1)))
  return set

W0 = np.random.normal(size=(16,1024))
W1 = np.random.normal(size=(16, 16))
W2 = np.random.normal(size=(4, 16))
b0 = np.zeros((16, 1))
b1 = np.zeros((16, 1))
b2 = np.zeros((4, 1))
costs = []
num_of_train = 200
batch_size = 16
num_of_epochs = 10
batch_num = (20//16)
learning_rate = 0.3
train_set = []
test_set = []
#merge data and  label
train_set=merge(train_data_shuffled , train_label_shuffled )
test_set=merge(test_data_shuffled , test_label_shuffled)
```

با توجه به کد بالا داده های مورد نظر را تعریف کرده و مقدار وزن ها و بایاس هارا رندوم میدهیم

مانند قبل برای هر minibatch ، feedforward میزنیم با این تفاوت که در هر مرحله هزینه هارا محاسبه کرده و تابع گرادیان هزینه را نیز محاسبه میکنیم . در نهایت تغییرات را برای وزن ها اعمال میکنیم و سپس روی داده ها یک feedforwardبرای محاسبه نتیجه و دقت نهایی میزنیم . زمان و هزینه و دقت را گزارش میکنیم .

```
def merge(data,label):
  set=[]
  for i in range(label.T.shape[1]):
      set.append((data.T[:, i].reshape(1024, 1), label.T[:, i].reshape(4,
1)))
```

```
return set
```

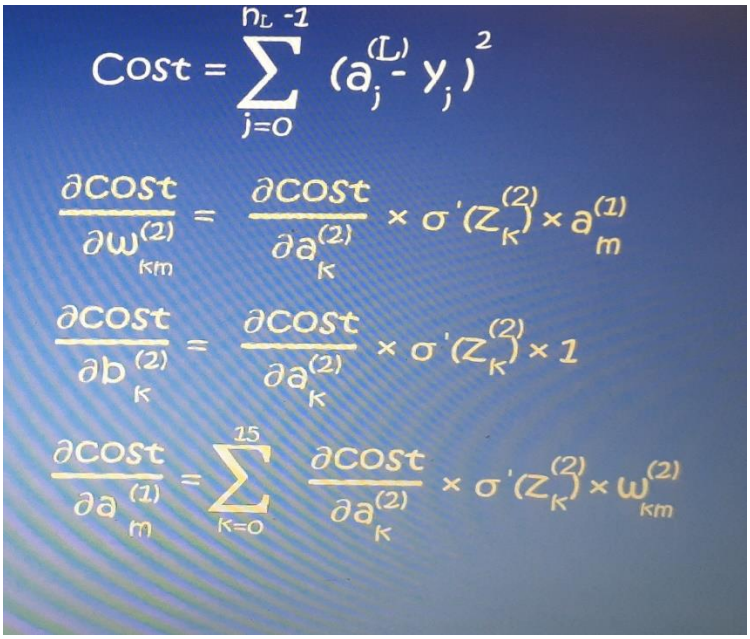
برای راحتی کار با لیبل ، آن را با ماتریس دیتایمان مرج میکنیم تا در دسترسی به دیتا ست راحت تر بتوانیم عکس را با لیبلش داشته باشیم.

قسمتی از کد :

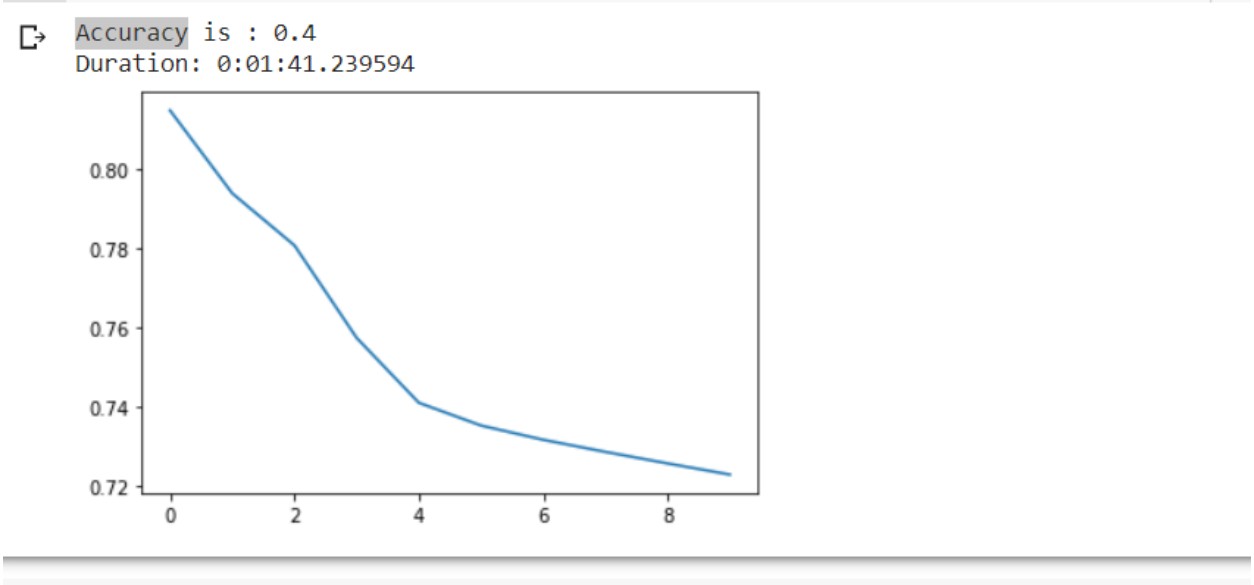
```
for _img, label in ba:

    a1 = sigmoid(W0 @ _img + b0)
    a2 = sigmoid(W1 @ a1 + b1)
    a3 = sigmoid(W2 @ a2 + b2)

    # The last layer Computing weight
    for j in range(g_W2.shape[0]):
        for k in range(g_W2.shape[1]):
            g_W2[j, k] += 2 * (a3[j, 0] - label[j, 0]) * a3[j, 0] * (1 - a3[j, 0]) * a2[k, 0]

    #The last layer Computing bias
    for j in range(g_b2.shape[0]):
            g_b2[j, 0] += 2 * (a3[j, 0] - label[j, 0]) * a3[j, 0] * (1 - a3[j, 0])

    #3rd layer Computing activation
    delta_2 = np.zeros((16, 1))
    for k in range(16):
        for j in range(4):
            delta_2[k, 0] += 2 * (a3[j, 0] - label[j, 0]) * a3[j, 0] * (1 - a3[j, 0]) * W2[j, k]
```

✓ 8m 43s completed at 12:02 AM

برای مشتق گیری از فرمول های زیر استفاده کردیم :



تصویر خروجی:دقت ۴۰ درصد



قسمت ۴ :vectorization

تنها تفاوت این قدم با قدم قبلی در زمان اجرای آن است که ما محاسبه گرادیان ها را به شکل زیر تغییر داده ایم.

```
# The last layer Computing weight
g_W2 += 2 * (a3 - label) * a3 * (1 - a3) @ np.transpose(a2)

#The last layer Computing bias
g_b2 += 2 * (a3 - label) * a3 * (1 - a3)


#3rd layer Computing activation
delta_2 = np.zeros((16, 1))
delta_2 += np.transpose(W2) @ (2 * (a3 - label) * (a3 * (1 - a3)))

# 3rd layer Computing weight
g_W1 += ( a2 * (1 - a2) *delta_2) @ np.transpose(a1)

#3rd layer Computing bias
g_b1 += delta_2 * a2* (1 - a2)
#  2nd layer computing activation
delta_1 = np.zeros((16, 1))
delta_1 +=   np.transpose(W1) @ (delta_2 * a2 * (1 - a2))

# 2nd layer computing weight
g_W0 += (delta_1 * a1* (1 - a1) ) @ np.transpose(_img)
# 2nd layer computing bias
g_b0 += delta_1 * a1* (1 - a1)
```

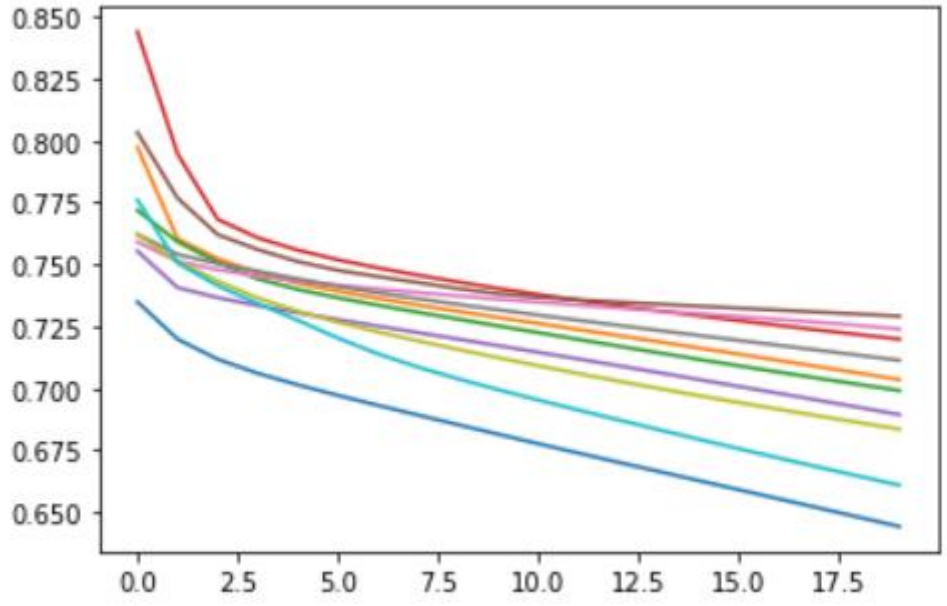
تصاویر ۱۰ خروجی زیر:

```
1
Avg_cost : 0.6813665991200091
Accuracy is : 0.445
Duration: 0:00:01.431198
2
Avg_cost : 0.7308082557412722
Accuracy is : 0.4
Duration: 0:00:01.405430
3
Avg_cost : 0.7261948465091648
Accuracy is : 0.43
Duration: 0:00:01.408855
4
Avg_cost : 0.7468378654495906
Accuracy is : 0.37
Duration: 0:00:01.417005
5
Avg_cost : 0.7164153946232503
Accuracy is : 0.42
Duration: 0:00:01.432656
6
Avg_cost : 0.7445115488530594
Accuracy is : 0.355
Duration: 0:00:01.464566
7
Avg_cost : 0.73671110178541
Accuracy is : 0.35
Duration: 0:00:01.470221
8
Avg_cost : 0.7320969783171118
Accuracy is : 0.43
Duration: 0:00:01.470152
9
Avg_cost : 0.7139208937067717
Accuracy is : 0.465
Duration: 0:00:01.525671
10
Avg_cost : 0.7025750370218704
Accuracy is : 0.45
Duration: 0:00:01.413890
Avg_ten_cost : 0.723143852112751
Avg_ten_Accuracy : 0.41150000000000003
```

میانگین دقت ۱۰ بار اجرا برابر است با ۴۱ درصد

تصویر نمودار هزینه این ۱۰ ورودی:

```
Avg_ten_cost : 0.7231438521127 51
Avg_ten_Accuracy : 0.41150000000000003
```



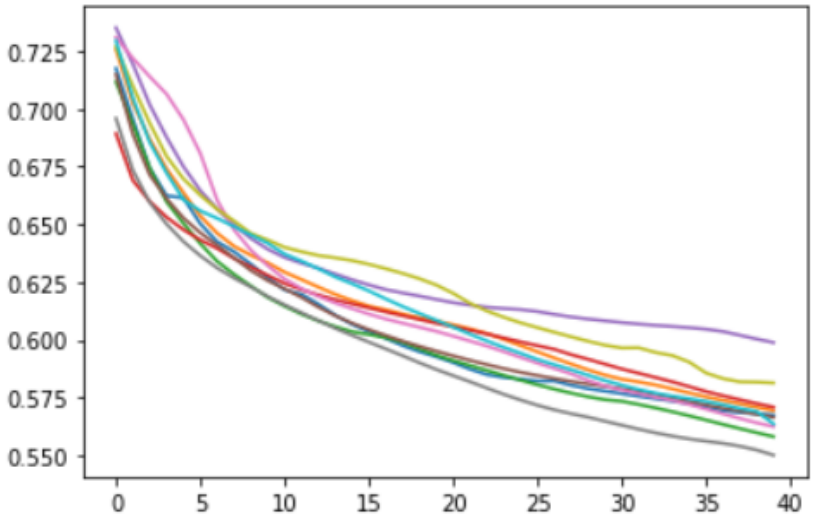
قسمت ۵ : testing model

در این مرحله تنها تعداد داده ها متفاوت است و آموزش را هم بر روی ۸۰۰train داده و هم بر روی ۴۰۰۰ داده testاعمال میکنیم

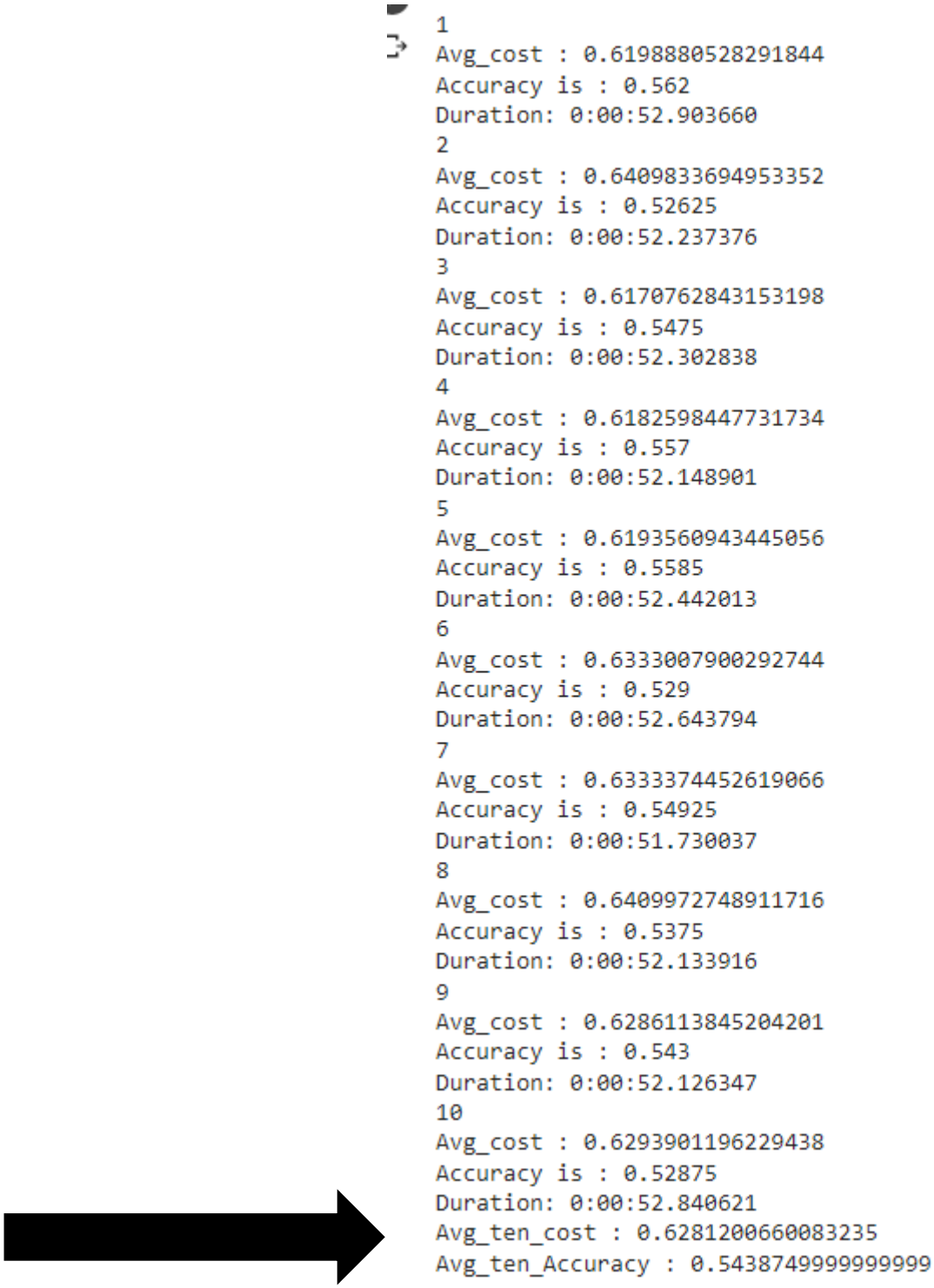
تصاویر خروجی برای train:

میانگین دقت برای این ۱۰ بار اجرا برابر با ۵۶ درصد است.

```
Avg_ten_cost : 0.6115206955104142
Avg_ten_Accuracy : 0.5628750000000001
```



تصاویر خروجی برای test :



```
1
Avg_cost : 0.6198880528291844
Accuracy is : 0.562
Duration: 0:00:52.903660
2
Avg_cost : 0.6409833694953352
Accuracy is : 0.52625
Duration: 0:00:52.237376
3
Avg_cost : 0.6170762843153198
Accuracy is : 0.5475
Duration: 0:00:52.302838
4
Avg_cost : 0.6182598447731734
Accuracy is : 0.557
Duration: 0:00:52.148901
5
Avg_cost : 0.6193560943445056
Accuracy is : 0.5585
Duration: 0:00:52.442013
6
Avg_cost : 0.6333007900292744
Accuracy is : 0.529
Duration: 0:00:52.643794
7
Avg_cost : 0.6333374452619066
Accuracy is : 0.54925
Duration: 0:00:51.730037
8
Avg_cost : 0.6409972748911716
Accuracy is : 0.5375
Duration: 0:00:52.133916
9
Avg_cost : 0.6286113845204201
Accuracy is : 0.543
Duration: 0:00:52.126347
10
Avg_cost : 0.6293901196229438
Accuracy is : 0.52875
Duration: 0:00:52.840621
Avg_ten_cost : 0.6281200660083235
Avg_ten_Accuracy : 0.5438749999999999
```

میانگین دقت برای ۱۰ بار اجرای تست برابر با ۵۴ درصد شده است.

```
Avg_ten_cost : 0.6281200660083235
Avg_ten_Accuracy : 0.5438749999999999
```