گزارش کد Hw1 درس شبکه عصبی

مهدی فقهی

401722136

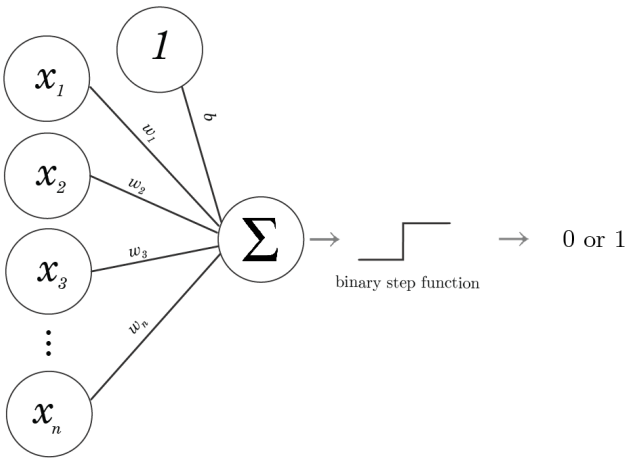
در ابتدا خواسته شد بود که یک متغیر به نام parameters ساخته شود که سه عضو دارد که به ترتیب معادل با w2 , w1 , b نرون ما خواهند بود .

```
# Create a zero numpy array with the shape of (3,1)

#####################################
## code here
parameters = np.zeros((3,1))
print(parameters)
#####################################
```

```
[[0.]
 [0.]
 [0.]]
```

در ادامه یک بردار ۵۰۰ تایی از یک می‌سازیم به نام one_add و به بردار X که ورودی نرون ما هست اضافه می کنیم که همان ورودی به b هست که در نظر گرفته نشده بود .



```
[258] ####################################
      ## code here
      one_add = np.ones((X.shape[0],1))
      X = np.concatenate((one_add,X), axis = 1)
      print(X)
      ####################################
```

```
[[ 1.          2.78091423  3.64950011]
 [ 1.          0.64947089  4.4284228 ]
 [ 1.         -3.39331054 -1.43941482]
 ...
 [ 1.          0.80867264  6.02633374]
 [ 1.         -6.6034882   0.16248827]
 [ 1.         -2.6522526  -0.27898589]]
```

این بردار را به کمک np.concatenate کردن بردار X و بردار one_add می‌سازیم .

در ادامه یک فانکشن برای محاسبه مقدار خروجی متناسب با ورودی به نرون و براساس وزن‌ها و ثابت b نرون به نام calculate output می‌سازیم .

حاصل را با ضرب کردن ماتریس ورودی در ماتریس parameters بدست می‌آوریم .

که برابر با حاصل ضرب یک ماتریس 500 در 3 در یک ماتریس 3 در 1 هست که منتهی به یک ماتریس 500 در 1 خواهد شد و در نهایت اگر بزرگتر از یک بود حاصل را برابر با یک و اگر کوچکتر بود را برابر با صفر در نظر می گیریم .

```
[260] def calculate_output(X, parameters):
          '''
          Calculate the output of perceptron
          for all samples
          intput(s):
          X (ndarray): a 2-D array with the s
          parameters (ndarray): a 2-D array w
          output(s):
          y_out (ndarray): a 2-D array with t
          '''
          ###################################
          ## code here
          ## y_out is the output of your neur
          ## First, you should implement matr
          ## Second, you should implement a U
          ## the y_out should be an array wit
          ## please note that you are not all
          y_out = X.dot(parameters) >= 1

          ###################################

          return y_out

      print(calculate_output(X, parameters))
```

```
[[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
```

در انتها یک پیاده سازی داریم که در آن برای بیست مرتبه عبارت ورودی X را به ورودی می‌دهیم و سپس حاصل را با تابع calculate_output بدست می‌آوریم که یک ماتریس به نام y_out به اندازه 500 در یک است .

سپس به تعداد ردیف‌های موجود از ورودی که برابر با ۵۰۰ عدد است نگاه می‌نماییم که آیا حاصل بدست آمده درست است یا خیر .

سپس برحسب ردیف موجود مقدار وزن نرون را برحسب مقدار خطا که برابر با ۱- و ۱ و صفر هست آپدیت می‌نماییم همانند فرمول Hebbian با این تفاوت که یک نرخ یادگیری نیز در حاصل بدست آمده ضرب می‌نماییم که از شدت تغییرات به صورت ناگهانی کم کنیم .

این روند را ۲۰ مرتبه انجام می‌دهیم تا درنهایت به یک وزن ایده‌آل برای شبکه عصبی خود برسیم که خروجی آن در فایل کد آمده است .

```
[262] for i in range(num_epochs):
          y_out = calculate_output(X, parameters)
          #print(y_out)
          errors = 0
          for j in range(X.shape[0]):
              if y[j,0] != y_out[j,0] :
                  errors += 1
              ####################################
              ## code here
              ## implement the code for updating each parameter
              ## note that you should calculate all parameters together.
              ## for example, you are not allowed to implement like the bellow:
              ## parameters[0] = ...
              ## paramters[1] = ...
              ## it should be like the bellow:
              ## paramters = ...
              parameters = parameters + X[j].reshape(3,1) * (y[j,0] - y_out[j,0]) * lr
              ####################################

          print("in epoch : ", i, " errors : ", errors)
      print(parameters)
```

همانطور که می‌دانید X[j] یک ماتریس ۳ در ۱ هست که باید تبدیل به ۱ در ۳ شود .

```
in epoch :  0  errors :  250
in epoch :  1  errors :  96
in epoch :  2  errors :  8
in epoch :  3  errors :  6
in epoch :  4  errors :  6
in epoch :  5  errors :  6
in epoch :  6  errors :  6
in epoch :  7  errors :  3
in epoch :  8  errors :  3
in epoch :  9  errors :  3
in epoch :  10  errors :  3
in epoch :  11  errors :  3
in epoch :  12  errors :  2
in epoch :  13  errors :  2
in epoch :  14  errors :  2
in epoch :  15  errors :  2
in epoch :  16  errors :  2
in epoch :  17  errors :  2
in epoch :  18  errors :  1
in epoch :  19  errors :  1
[[  9.3       ]
 [-98.00320591]
 [-51.11042401]]
```

با w1 , w2 ,b تابع خط خود را می‌سازیم و همانطور که در نمودار می‌بیند به کمک این تابع خط می‌توانیم ورودی‌ها را به خوبی از هم تفکیک نماییم .