الگوریتم های یادگیری ماشین
گزارش تمرین هاپفیلد

یاسمین رحیمی (۸۱۰۸۹۶۰۲۱)

صورت تمرین

در این تمرین ما همان داده های تمرین قبلی را به عنوان ورودی داریم، و با الگوریتم هاپفیلد میخواهیم داده های یادگیری و همچنین داده های نویزی را به درستی تشخیص دهد.

الگوریتم هاپفیلد

یادگیری

الگوریتم هاپفیلد به این صورت عمل می کند که برای هر داده یا کاراکتر یک ماتریس از ورودی می گیرد و از آنجایی که این الگوریتم اتواسیشییت است و خروجی مورد انتظار ما همون ورودی است، مقدار وزن این کاراکتر را برابر با ضرب ماتریس سطری ورودی در ماتریس ستونی ورودی می کند و برابر وزن قرار می دهد اما از ان جایی که در این ماتریس وزن داده های قطر اصلی همیشه برابر با یک میشود،قطر اصلی ماتریس وزن را برابر صفر قرار می دهیم.

تست

در الگوریتم هاپفیلد با داده های تست هم یادگیری انجام می شود، به این صورت که ماتریس کاراکتر مورد نظر را در یکی از ماتریس های وزن ضرب می کنیم و به جای درایه ی متناظر با این نود در تابع وزن مقدار خروجی تابع اکتیویشن را قرار می دهیم و دوباره تست می کنیم و این کار را انقدر تکرار میکنیم تا مقدار تابع وزن تغیری نکند.

این برنامه با زبان پایتون نوشته شده.

تابع Readfile()

در این تابع ما داده‌ها را که همان کارکترها به شکل یک ماتریس ۹*۷ است را می‌گیریم و به صورت یک آرایه نامپی از اعداد ۱ و 1- روی یک لیست ذخیره می‌کنیم و لیست را برمی‌گردانیم.

```
def Readfile():
    train_data = []
    with open("training-data.txt") as file:
        cnt,ind = 0,0
        train = num.zeros(63, dtype=int)
        for line in file:
            if cnt %%==%%:
                for element in line:
                    if element=='#':
                        train[ind] = 1
                        ind+=1
                    elif element=='*':
                        train[ind] = -1
                        ind+=1

            elif cnt==9:
                cnt,ind = 0,0
                train_data.append(train)
                train = num.zeros(64,dtype=int)
            else:
                for element in line:
                    if element=='#':
                        train[ind] = 1
                        ind+=1
                    elif element=='*':
                        train[ind] = -1
                        ind+=1
                cnt+=1
        return train_data
```

تابع weight()

این تابع ماتریس وزن را به ازای یک کاراکتر مشخص (مثلا اینجا کاراکتر A در نظر گرفه شده) ایجاد می‌کند تا برای داده‌ی نویزی این کاراکتر تست شود و کاراکتر بدون نویز A را برگرداند.

```
def weight(train_data):
    s = train_data[0]
    weightSum = num.outer( s, s)
    weightSum -= num.identity(63)
    return weightSum
```

تابع weightTotal()

در این تابع ما ماتریس وزن را به ازای ورودی بزرگتری یعنی تمام کاراکترها ایجاد می کنیم اما میخواهیم نشان دهیم که این روش یادگیری نتیجه ی درستی نمی دهد و تابع هاپفیلد حافظه دار نیست و بهتر است کاراکتر های کمتری برای مقداردهی هر ماتریس وزن در یادگیری شرکت کنند.

```
def weightTotal(train_data):
    weightSum = num.zeros((63,63) , dtype=float)
#     print (weightSum)
    weight_array = num.zeros((63,63), dtype=float)
    for i in range (0,7):
        s = num.zeros((1,63), dtype=float)
        s = train_data[i]

        weight_array = num.outer( s, s)
        weightSum += weight_array
    weightSum -= 7 * (num.identity(63))
    return weightSum
```

تابع test()

در تابع تست ما به ازای ورودی فایل تست الگوریتم هاپفیلد را به این صورت اجرا می کنیم :
در مرحله ی اول از داده های تست یک کپی میگیریم، سپس داده های تست را در ماتریس وزن ضرب میکنیم تا یک بردار ۱ در ۶۳ ایجاد شود، اگر این داده با کپی تست یکسان بود الگوریتم متوقف می شود و این آرایه را به عنوان خروجی(یعنی کاراکتر تشخیص داده شده بر می گرداند)
و اگر برابر نبود دوباره تست جدید را کپی می کند و انقدر مرحله ی قبلی را تکرار می کند تا در دو مرحله ی متوالی ضرب در ماتریس وزن خروجی یکسان بدهد و مانند بالا خروجی را بر می گردانیم.

```
def test(weightSum):
    test = num.zeros((1,63), dtype=float)
    ind = 0
    with open("test.txt") as file:
        for line in file:

            for element in line:
 #                test[ind] = int('element')
                if element=='1':
                   test[0][ind] = 1
                   ind += 1
                else:
                    if element=='0':
                        test[0][ind] = -1
                        ind += 1
                    elif element == '/n':
                        ind += 0
    print (test)
    testweight= weightSum
    epoch = 0
    while True:
        old_test = test.copy()
        y_in = num.zeros((1,63), dtype=float)
        for i in range (1):
 #            print (test)
            for l in range(63):
                for k in range(63):
                    x = test[i][k]
                    y = testweight[k][l]
                y_in[i][l] += x * y
                test[0][l] = Actfunc(y_in[0][l])
        if num.array_equal(old_test, test):
            break
        epoch += 1
    print (epoch)
    return test
```

فایل تست را با ورودی وزن تولید شده توسط مجموع وزن هر هفت کاراکتر اجرا می کنیم و بردار ورودی را می توانیم با بردار خروجی مقایسه کنیم:

خروجی آرایه دوم است و ورودی نویز دار آرایه اول است.

```
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct  3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /Users/yada/Documents/python/hopfield.py ==========
>>> main()
[[ 1. -1.  1.  1. -1. -1. -1. -1.  1. -1.  1. -1. -1. -1. -1. -1. -1.  1.
  -1.  1. -1. -1. -1.  1. -1.  1. -1. -1. -1. -1.  1.  1.  1. -1. -1. -1.
   1.  1.  1.  1.  1. -1. -1.  1. -1. -1. -1.  1. -1. -1.  1. -1. -1. -1.
   1. -1.  1.  1.  1. -1.  1.  1.  1.]]
2
[[-1. -1. -1.  1.  1.  1. -1. -1.  1. -1. -1. -1.  1.  1.  1. -1. -1. -1.
  -1.  1.  1.  1. -1. -1. -1. -1.  1.  1.  1. -1. -1. -1. -1.  1.  1.  1.
  -1. -1. -1. -1. -1.  1.  1. -1. -1. -1. -1. -1.  1. -1.  1. -1. -1. -1.
  -1.  1. -1. -1.  1.  1.  1. -1. -1.]]
None
None
>>>
```

و تابع ورودی آرایه ی زیر می باشد که ۰ ها باید ۱- باشند:

10110000101000000101000101000011100011111001000100100010111011 1