یادگیری عمیق

مینی پروژه اول

نام استاد: دکتر پیمان ستوده

هدف: یاد دادن عملیات های منطقی به یک نورون

نام دانشجو : حمزه قائدی

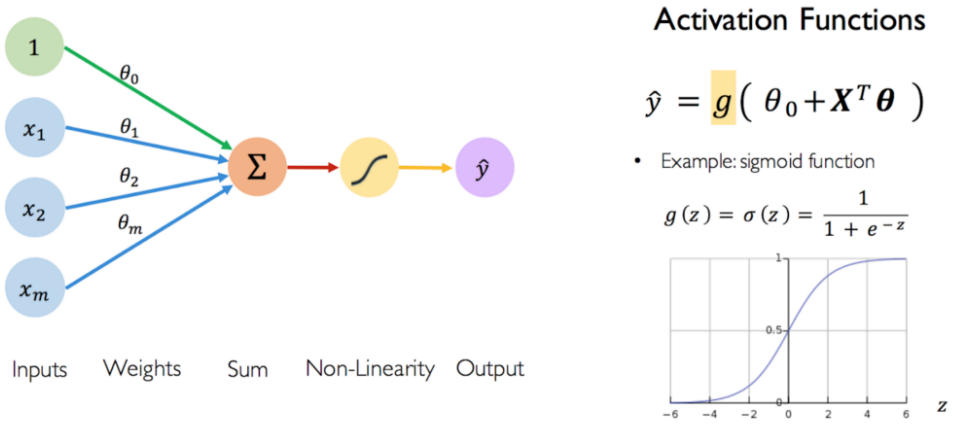
شماره دانشجویی: 9831419

تاریخ: 99/01/22

<<بخش صفرم>>

هدف: پیاده سازی یک مدل برای نورون و فرایند یادگیری آن

شکل زیر، ساختار یک نورون در شکبه عصبی را نشان میدهد. یک نورون، برداری از ورودی ها را دریافت کرده و با اعمال یک تابع غیر خطی (تحت عنوان تابع فعالساز) خروجی متناسب با ورودی تولید میکند



MIT: Alexander Amini, 2018 introtodeeplearning.com

با تغییر پارامترهای بایاس،وزنها و نیز تابع فعال ساز، میتوان نورون های متفاوتی ایجاد کرد .کد پایتون زیر یک نورون کلی(انتزاعی) را شبیه سازی میکند. توجه شود که تابع فعال ساز پیاده سازی نشده و انواع مختلف نورون ها را میتوان با ارث بری از این کلاس پایه و تغییر پارامترهای آن (مثلا بازنویسی تابع فعال ساز) مدل کرد.

```
#--------------IMPORTS-----------------------------------
import Sigmoid as sigmoid
import numpy as np
#--------------------------------------------------------

#abstract model for a neuron---------------------------------------
class Neuron():

    def __init__(self,weight,bias):
        self.weight = weight
        self.bias = bias


    #every concrete neuron implements this function in a specific manner!
    def activate(self,input): #some nonlinear operation
        pass
#-------------------------------------------------------------------
```

در این پروژه که هدف اموزش عملیات ها منطقی به یک نورون است، نورونی با دو ورودی و یک خروجی نیاز داریم. همچنین طبق دستور کار پروژه، باید از تابع سیگموید به عنوان فعال ساز این نورون استفاده شود. بدین منظور، درکد زیر، نورونی با نام LogicGateNeuron که از کلاس پایه Neuron مشتق شده است را ایجاد کرده ایم

ورودی و بردار وزن این نورون به دو بعد محدود شده است (اگر بردار ورودی یا بردار وزن این نورون دو بعدی نباشد،خطایی با عنوان (must be a 2d vector!....) ایجاد میشود) این نورون مقدار مشتق تابع فعالساز (در اینجا مشتق سیگموید) به ازای ورودی را هم در متغیر activate_prime نگه میدارد که از آن در مرحله بهینه سازی استفاده میشود:

```
class LogicGateNeuron(Neuron):


    def __init__(self,weight:list,bias:float):
        if(len(weight)!=2):
            raise Exception("weight muse be a 2d vector!")
        self.weight = weight
        self.bias = bias


    def linear_combinator(self,input):
        return np.dot(input,self.weight) + self.bias  # z = w1*x1 + w2*x2 + b


    #I injected inputs to the neuron through activate function
    def activate(self,input):
        if(len(input) != 2):
            raise Exception("input muse be a 2d vector!")

        z = self.linear_combinator(input)

        self.output = sigmoid.sigmoid_func(z) #applying sigmoid function as neuron activa
tion

        # derivative of the activation function at current input
        self.activate_prime = self.output * (1-self.output)

        return self.output
```

فرایند آموزش دادن به یک نورون، در کلاسی به نام Trainer کپسوله شده(!) و به صورت زیر است:

کلاس Trainer یک نورون را به همراه مجموعه دادهای لازم برای آموزش آن نورون دریافت میکند (از طریق تابع سازنده خود)، مجموعه داده های آموزش در قالب پارامتر training_set به trainer فرستاده میشود این پارامتر یک دیشکنری با قالب زیر است که زوجهای مرتب (کلید های دیکشنری) دو داده ورودی و عدد بعد از نقل قول خروجی متناظر (معادل مقدار نظیر هر کلید) را نمایش میدهد.

$$trainigset = \{(0,0): 0, (1,0): 1, \dots\}$$

```
class Trainer():

    def __init__(self,neuron:Neuron.LogicGateNeuron,training_set:dict):
        self.neuron = neuron
        self.training_set = training_set
        self.trainingerr = 0
        self.testerr = 0
        self.trainerrList = list()
```

Trainer با فراخوانی تابع train آموزش را شروع میکند (!). تابع train دو ورودی دارد، lr نرخ یادگیری و epchos هم تعداد دفعات تکرار فرایند یادگیری را مشخص میکند. (فرایند یادگیری هم یعنی تنظیم پارامترهای بایاس و وزون نورون !)

```
#lr : learning rate

    def train(self,epochs,lr):
        while(epochs > 0):
            self.GD(lr)
            self.trainerrList.append(self.trainingerr)
            epochs -=1
```

برای یافتن بایاس و وزن مناسب،ت ابع میانگین مربعات خطا (MSE) به عنوان تابع هزینه توسط الگوریتم GD بهیه سازی شده است . روابط مورد استفاده به صورت زیر است:

$$\text{هزینه: } J(\theta) = (\frac{1}{2})\sum(y - \hat{y}(X;\theta,b))^2$$

$$\hat{y} = \sigma(\vec{\theta}x + b)$$

$$\frac{\partial J}{\partial \theta_i} = (y - \hat{y}) * \acute{o}(\vec{\theta}x + b) * x_i$$

تابع GD با دریافت نرخ یادگیری، یک دور فرایند یادگیری را به ازای کلیه نقاط training_set انجام میدهد و بر اساس آن مقادیر وزن و بایاس نورون را به روز رسانی میکند .رسیدن به مقادیر بهینه وزن و بایاس این فرایند باید چندین بار تکرار شود.

تکرار فرایند یادگیری در تابع train اتفاق می افتد.این تابع به تعداد دفعات مشخص شده در پارامتر epochs ، فرایند یادگیری را تکرار میکند (تابع GD را فراخوانی میکند!)

به منظور بررسی خطای یادگیری، خطای هر دور اجرای فرایند یادگیری (هر epoch) در لیستی به نام trainingerrlist ثبت میشود

```
def GD(self,lr):

        self.trainingerr = 0
        num = len(self.training_set) #number of data poins in training_set
        for input,target in self.training_set.items():

            self.neuron.activate(input)
            self.trainingerr += (1/num) * (self.neuron.output - target)**2 #evaluating mean square error (MSE)

           #updates weights and bias of the neuron-----------------------------------------------------------
            self.neuron.weight[0] -= lr * input[0] * self.neuron.activate_prime * (self.neuron.output - target)
            self.neuron.weight[1] -= lr * input[1] * self.neuron.activate_prime * (self.neuron.output - target)
            self.neuron.bias -= lr * self.neuron.activate_prime * (self.neuron.output - target)
            #-------------------------------------------------------------
```

و در نهایت تابع test با دریافت یک مجموعه جدید (معمولا متفاوت از داده های مورد استفاده برای اموزش) از داده ها تحت عنوان test_set (که یک دیکشنری با قالبی شبیه به training_set است) میانگین مربعات خطای تعمیم را محاسبه میکند

```
#-------------testing a trained neuron over new set of data points------------------

    def test(self,test_set:dict):
        num = len(test_set)
        self.testerr = 0
        for input,target in test_set.items():
            self.neuron.activate(input)
            self.testerr += (1/num) * (self.neuron.output - target)**2
```

<<بخش اول>>

هدف: آموزش عملیات های منطقی به یک نورون

در قسمت قبل مدلی از نورون و فرایند آموزش به آن را پیاده سازی کردیم در این بخش، سعی میکنیم عملیات های منطقی AND,OR,XOR را به یک نورون یاد بدهیم! برای یاد دادن هریک از عملیات های فوق به نورون روند کلی زیر را دنبال میشود:

ابتدا یک نمونه LogicGateNeuron میسازیم و مقدار اولیه پارامترهای وزن آن را از توزیع گوسی انتخاب کرده و پارامتر بایاس ان را هم یک قرار میدهیم

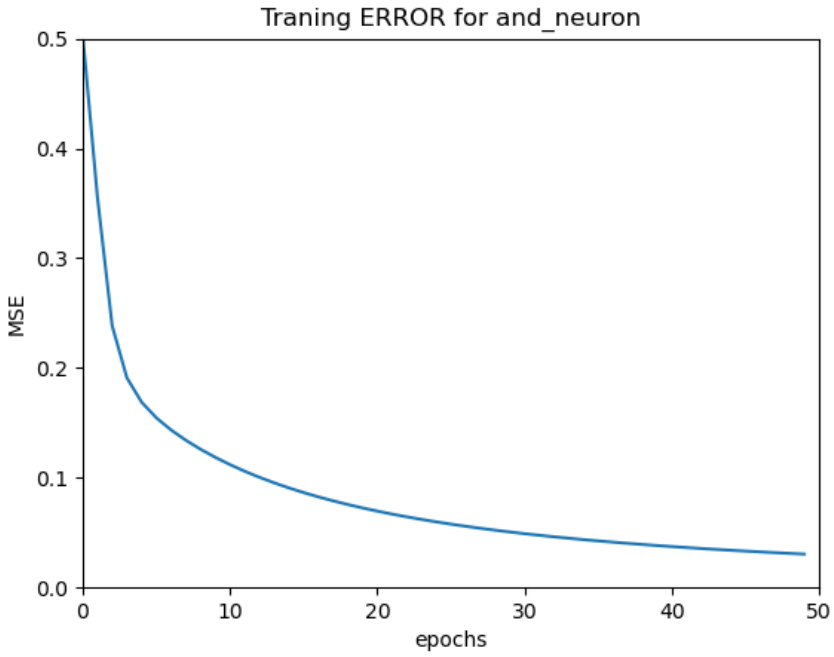
در مرحله بعد یک دیکشنری از داده های لازم برای یادگیری عملیات فوق ایجاد میکنیم. این دیکشنری شامل ورودی های گیت معادل نورن و خروجی مورد انتظار گیت میباشد

و در مرحله آخر با ساخت یک نمونه از کلاس trainer و ارسال نورون و مجموعه داده های لازم برای آموزش عملیات مورد نظر به آن، نورون را آموزش میدهیم

1) یاد دادن عملیات منطقی **AND** به نورون:

```
#miniproject PART1 - a neuron should learn logical and operation
#-----------------IMPORTS-------------------
import Neuron
import Trainer
import numpy as np
import matplotlib.pyplot as plt
#---------------------------------------

#first randomly initialized weights and set bias to 1
w1 = np.random.randn()
w2 = np.random.randn()
w = [w1,w2]
b = 1
and_training_set ={(0,0):0,(0,1):0,(1,0):0,(1,1):1}

and_neuron = Neuron.LogicGateNeuron(w,b) #create an instance of LogicGateNeuron

#using a trainer to train the and_neuron !
epochs = 50
and_trainer = Trainer.Trainer(and_neuron,and_training_set)
and_trainer.train(epochs,lr = 2)

#plotting training erro vs epochs:---------------------------
plt.plot(range(0,epochs),and_trainer.trainerrList)
plt.axis([0 , epochs , 0 , 0.5])
plt.title("Traning ERROR for and_neuron")
plt.xlabel("epochs")
plt.ylabel("MSE")
plt.show()
#-------------------------------------------------------------
acc = and_trainer.trainerrList[epochs-1] #accuracy
print(acc)
```

به کمک کتابخانه matplotlib ، نمودار خطای آموزش به ازای هر دور اجرای فرایند آموزش (هر epoch یا معادلا هر بار اجرای تابع GD) در شکل زیر قابل مشاهده است:



با تغییر پارامترهای epochs و lr میتوان فرایند آموزش را کنترل کرد. نکته قابل توجه (برای من البته!) این بود که االگوریتم خیلی دیر به مقدار بهینه همگرا شد که احتمالا به خاطر انتخاب MSE به عنوان تابع هزینه بوده چون بسیاری از ورودی ها و خروجی ها صفر است عملا برای تعداد تکرار های کم تغییر چندانی در مقدار پارامتر ها اتفاق نمی افتد.!..به هر حال پس از 50 بار تکرار و نرخ یادگیری اولیه 2 به خطای آموزش 0.0358237723059042 رسیدم.

2) یاد دادن عملیات منطقی OR به یک نورون:

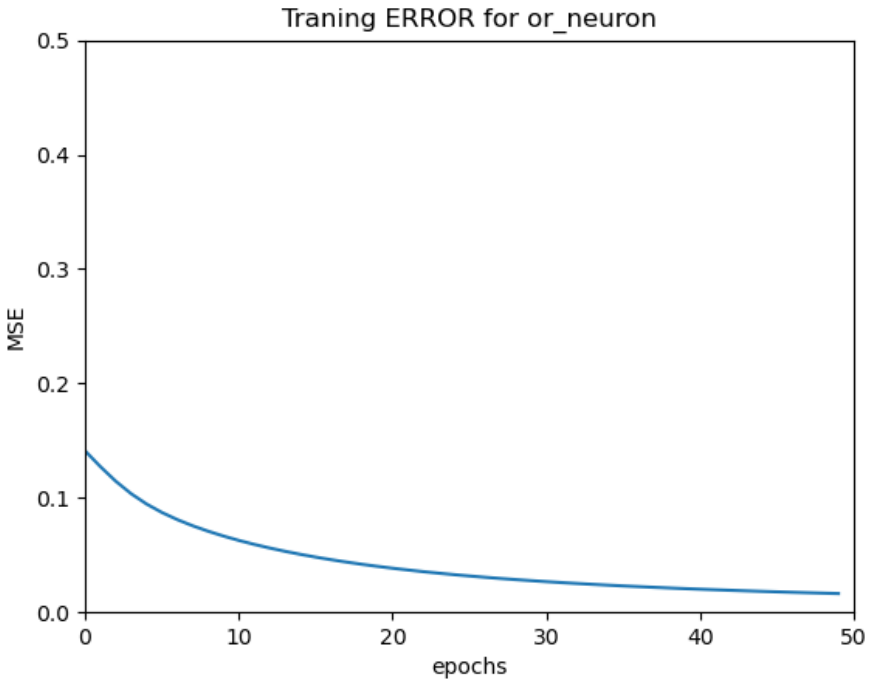
روند کلی مشابه قسمت اول بوده و صرفا مجموعه داده های آموزش (training_set) باید متناسب به عملیات OR به شکل زیر تغییر کند:

$$trainig\_set = \{(0,0): 0, (1,0): 1, (0,1): 1, (1,1): 1\}$$

کد هم عملا شبیه قسمت قبل بوده و دوباره تکرار نمیکنم!

نتایج حاصله نیز به صورت زیر است:

پس از 50 بار تکرار فرایند آموزش با نرخ آموزش 2، خطای نهایی آموزش برابر با 0.016352673208456944 شده که تقریبا نصف خطای نهایی برای یادگیری AND است. اینکه تعداد بیشتری از خروجی ها برابر یک است در نصف شدن خطا بی تاثیر نیست!...



3) یاد دادن عملیات XOR به نورون:

مشابه دو قسمت قبل است فقط مجموعه داده های اموزشی متناسب با عملیات XOR به صورت زیر تغییر میکند:

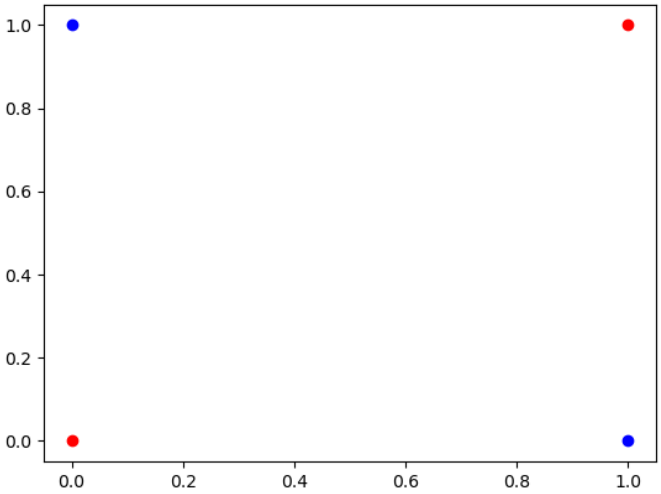
$$trainig\_set = \{(0,0): 0, (1,0): 1, (0,1): 1, (1,1): 0\}$$

با تنظیماتی مشابه دو حالت قبل (یعنی با epochs =50 وlr=2) به خطای 0.3240714402824504 میرسیم که خطای قابل قبولی نیست!

اگر آموزش را با epoch =100 تکرار کنیم به خطای 0.3241616609064404 میرسیم که فرق چندانی با حالتی که epoch=50/ بود ندارد. نمودار خطا بر حسب epoch هم به صورت زیر میشود که فرقی با نمودار قبل ندارد:

علت اشباع شدن خطا در فرایند یاد دادن XOR به تک نورون ایسنت که داده های عملیات XOR به صورت خطی قابل دسته بندی نیستند در حالیکه عملیات یاد دادن به تک نورون با دو مقدار ورودی (بردار ورودی دو بعدی یا دوتا ویژگی) و دو خروجی متفاوت(در اینجا 0 و 1)، عملا معادل یافتن بهترین خطرجداساز (مرز) در فضای ورودی ها (اینجا صفحه متشکل از دو ورودی!) میباشد و چون داده های XOR با یک خط قابل جدا سازی نیستند لذا الگوریتم یادگیری همگرا نشده و خطا به صفر میل نمیکند!

شکل زیر داده های ورودی را به همراه مقدار متناظر آنها در خروجی(آبی معادل خروجی یک و قرمز معادل خروجی صفر) برای عملیات XOR نمایش میدهد. با توجه به شکل زیر نیز مشخص است که نقاط قرمز و آبی با یک خط قابل جداساز نیستند



نمودار بالا به کمک کد زیر ایجاد شده است (کد در فایل PART1_XOR قرار دارد!)

```
plt.figure()

x1 = [0,1]
y1 = [0,1]
x2 = [0,1]
y2 = [1,0]

plt.scatter(x1,y1,color='red')
plt.scatter(x2,y2,color='blue')
plt.show()
```

<<بخش دوم>>

استفاده از داده های تولیدشده توسط make_blobs برای آموزش دادن یک نورون:

در اینجا نیز ساختن یک نورون و آموزش دادن به آن کاملا مشابه به قسمت های قبل است با این تفاوت که اینبار training_set به کمک make_blobs و به صورت زیر ایجاد میشود:

ابتدا 100 نمونه از توزیع میکسچر گوسی دو متغیره (معادل دو ورودی) با دو مرکز (معادل دو حالت در خروجی) ایجاد میشود سپس آنها را به یک دیشکنری تبدیل کرده (تا بتواند به عنوان ورودی به کلاس trainer ارسال شود) و به عنوان training_set استفاده میکنیم:

```
no_of_samples = 1000

x,y = make_blobs(no_of_samples,2,2)

#convert data set to a dictionary form so a trainer class can take it---------------
training_set = dict()
for i in range(0,no_of_samples):
    training_set[(x[i][0],x[i][1])] = y[i]
```

خطای آموزش برابر 2.47e-6 بوده و خطای تعمیم نیز 2.85e-6 میباشد که قابل قبول است