به نام خدا

دانشگاه صنعتی شریف - دانشکده مهندسی کامپیوتر

هوش مصنوعی

بهار ۱۴۰۰

تمرین عملی دوم - قسمت بهینه سازی

طراح: پویا معینی

موعد تحویل: ۲۴/۱/۱۴۰۰

همفکری در تمامی تمرین‌های درس توصیه می‌شود. در عین حال از شما خواسته می‌شود تا تمام پیاده‌سازی را به تنهایی و بدون مشاهده کد دیگران انجام دهید.

لطفا در فایل ارسالی تمام بلوک‌های کد اجرا شده و شامل نمودارها و خروجی‌های لازم باشند.

نام: مهدی سلمانی صالح آبادی

شماره دانشجویی: 98105824

توضیحات کلی

این دفترچه شامل ۳ بخش است. در کل تنها می‌توانید از کتابخانه‌هایی که در کد زیر import شده‌اند استفاده کنید. درنهایت نیز برای سابمیت، از دفترچه‌ی خود یک خروجی pdf بگیرید و آن را به همراه خود دفترچه در قالب یک فایل فشرده‌شده سابمیت کنید.

In [1]:

```
!pip install tqdm
```

```
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-pac
kages (4.50.2)
```

In [2]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import tqdm
```

سوال اول

در این سوال به شما تعدادی تابع و دامنه‌ی هرکدام داده‌شده است. نمودار هرکدام از توابع را در دامنه‌ی داده‌شده رسم کنید و سپس با استفاده از نمودار محدب بودن یا نبودن هر کدام را مشخص کنید (همراه با توضیحات).

یک منحنی محدب است اگر مجموعه نقاط بالای آن محدب باشد یا به نحوی دیگر هر دو نقطه روی خم را اگر به هم وصل کنیم خط حاصل بالای منحنی باشد.

الف) طبق توضیحات بالا از روی شکل مشخص است که هر دو نقطه روی محور را بگیریم، خط حاصل بالای محور می افتد پس تابع محدب است.

ب) برای مثال دو نقطه با طول 20 و 80 را در نظر بگیرید، خط واصل آنها نه تنها بالای نمودار نیست بلکه پایین آن قرار

دارد(تحدب منفی در آن بازه) پس تابع محدب نیست

ج) دو نقطه با طول 60 و 80 را برای مثال در نظر بگیرید. خط واصل بین آنها در بعضی بازه ها بالای نمودار و در بعضی بازه ها زیر نمودار است. پس می توان گفت که تابع مدنظر محدب نیست.

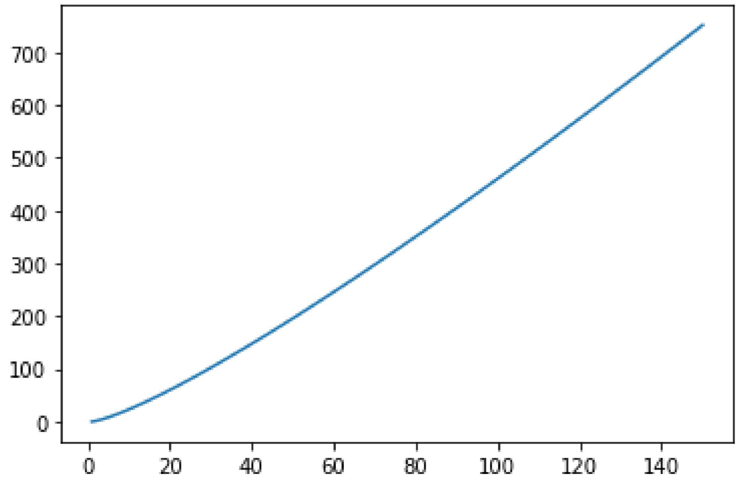
الف) $f(x) = xlog(x) \quad \text{where} \quad x \in [1, 150]$

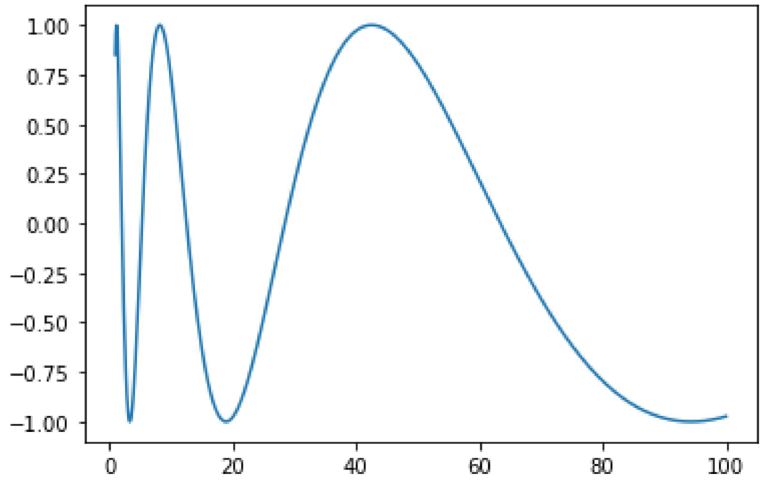
ب) $f(x) = \sin(-2\log(3x^2 + 5x)) \quad \text{where} \quad x \in [1, 100]$

پ) $f(x) = \log(0.02\cos(x) + \frac{5}{x}) \quad \text{where} \quad x \in [1, 100]$

In [3]:

```
x1 = np.linspace(1, 150, 1000)
y1 = np.log(x1) * x1
plt.plot(x1, y1)
plt.show()
x2 = np.linspace(1, 100, 1000)
y2 = np.sin(-2 * np.log(3 * x2 ** 2 + 5 * x2))
plt.plot(x2, y2)
plt.show()
x3 = np.linspace(1, 100, 1000)
y3 = np.log(0.02 * np.cos(x3) + 5 / x3)
plt.plot(x3, y3)
plt.show()
```





سوال دوم

در این سوال می‌خواهیم پیاده سازی تابع گرادیان و الگوریتم gradient descent و همچنین تاثیر learning rate را مشاهده کنیم.

تابع گرادیان، در ورودی خود تابع و نقاطی که باید گرادیان در آن‌ها محاسبه شود و همچنین learning rate را ورودی می‌گیرد. درخروجی نیز نقاط بروزرسانی شده را خروجی می‌دهد. در این سوال فرض می‌کنیم که همه‌ی تابع‌ها دو متغیری هستند.

در تابع دوم باید الگوریتم gradient descent را پیاده‌سازی کنید. در ورودی تابع، خود تابع f، مقادیر اولیه، learning rate و همچنین تعداد مراحل اجرای الگوریتم داده شده است. همچنین یک متغیر thereshold نیز ورودی داده‌شده است و هرگاه قدر مطلق مقدار x و یا y در آن مرحله از این thereshold بیشتر شد، اجرای الگوریتم را قطع کنید و خروجی‌ها را تا همان لحظه خروجی دهید. درنهایت تابع باید دو آرایه خروجی دهد که آرایه‌ی اول مقادیر x در طول زمان و آرایه‌ی دوم مقادیر y در طول زمان است. یعنی مثلا x_points[i] باید نشان دهنده‌ی مقدار x در مرحله‌ی iم باشد.

In [4]:

```
def gradient_update(f, x0, y0, alpha):
    #     todo
    e = np.finfo(np.float32).eps
    gradient_x, gradient_y = (f(x0 + e, y0) - f(x0, y0)) / e,  (f(x0, y0 + e) - f(x0, y0))
    new_x, new_y = x0 - alpha * gradient_x, y0 - alpha * gradient_y
    return new_x, new_y

def gd(f, initial, alpha, thereshold, steps):
    #     todo
    current_x, current_y = initial[0], initial[1]
    x_points, y_points = [current_x], [current_y]
    for i in range(steps):
        current_x, current_y = gradient_update(f, current_x, current_y, alpha)
        x_points.append(current_x)
        y_points.append(current_y)
        if (abs(current_x) > thereshold or abs(current_y) > thereshold):
            break;
    return x_points, y_points
```

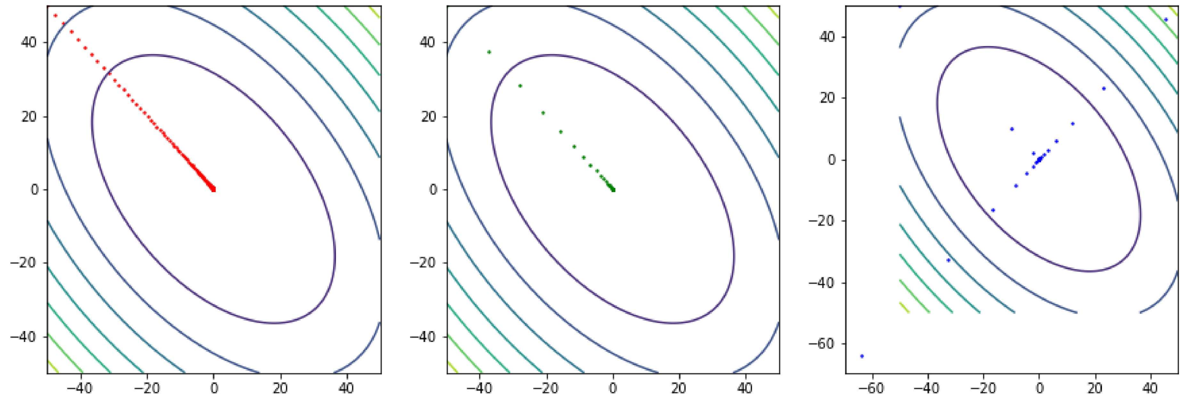
سپس پس از تکمیل دو تابع بالا، تابع زیر به عنوان مثال آورده شده است. با اجرای دو بلاک زیر سه نمودار مشاهده خواهید کرد که در هرکدام تاثیر learning rate مشاهده می‌شود. درنهایت این ۳ تصویر را تفسیر و توجیه کنید. دقت کنید برای ارزیابی نهایی، تابع‌های دیگری نیز درمرحله تصحیح ورودی داده خواهند شد.

In [5]:

```
def f(x, y):
    return x**2 + y**2 + x*y
```

In [6]:

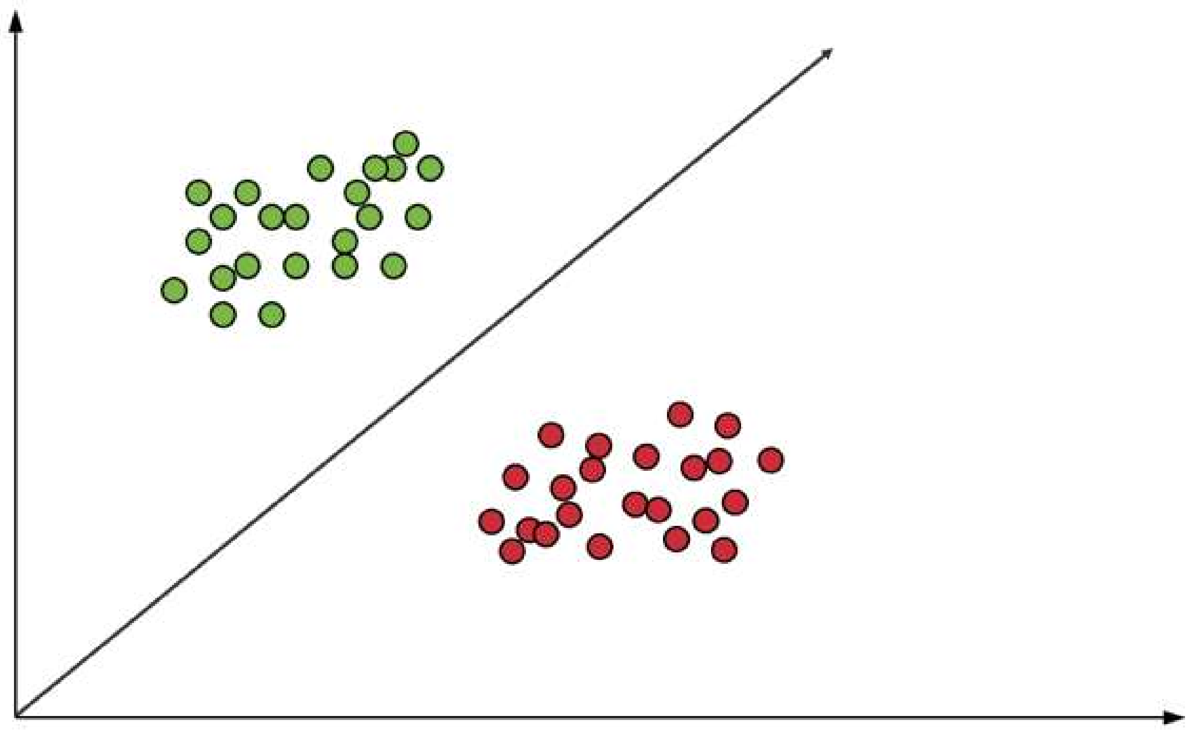
```
v_func = np.vectorize(f)

x, y = np.meshgrid(np.linspace(-50, 50, 100),
                   np.linspace(-50, 50, 100))

alpha_1, alpha_2,alpha_3 = 0.05, 0.25, 0.8
xs_1, ys_1 = gd(f, [-50, 50], alpha_1, 50, 1000)
xs_2, ys_2 = gd(f, [-50, 50], alpha_2, 50, 1000)
xs_3, ys_3 = gd(f, [-50, 50], alpha_3, 50, 1000)
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,5))
ax1.contour(x, y, v_func(x, y))
ax2.contour(x, y, v_func(x, y))
ax3.contour(x, y, v_func(x, y))
ax1.scatter(xs_1, ys_1, s=2, c="r")
ax2.scatter(xs_2, ys_2, s=2, c="g")
ax3.scatter(xs_3, ys_3, s=2, c="b")
plt.show()
```



در ابتدا دقت کنید که مینیمم تابع مشخص شده در نقطه صفر و صفر رخ می دهد. (دقت کنید که از مربع سازی تابع مذکور همواره بزرگتر مساوی صفر است) حال در شکل سمت چپ مقدار learning_rate کم است. برای همین به آرامی به جواب درست رسیدیم. در شکل وسط learing_rate اندازه مناسبی دارد و نتیجه آن هم درست است و سرعتش از حالت سمت چپ بیشتر است. در شکل سمت راست مقدار learning_rate بیشتر از حد مجاز خود شده است و الگوریتم حول نقطه بهینه حرکت زیگزاگی انجام می دهد چون با توجه به مقدار زیاد learning_rate نمی تواند به آن میل کند .

سوال سوم

در این سوال قصد داریم یکی از کاربردهای الگوریتم کاهش گرادیان را در یادگیری ماشین بررسی کنیم. در ادامه‌ی درس به بررسی دقیق‌تر و عمیق‌تر الگوریتم‌های یادگیری خواهیم پرداخت بنابراین در این سوال یک حالت ساده را در این چارجوب بررسی می کنیم. دیتاستی در اختیار شما قرار داده‌ایم که شامل سه‌تایی‌های (x_i, y_i, z_i) است که قاعدتا x و y مختصات دکارتی داده است و z یک متغیر باینری است. قصد داریم برحسب مختصات هر داده در صفحه‌ی مختصات پارامتر z آن را حدس بزنیم. برای این کار به عنوان یک فرض ساده کننده فرض می‌کنیم که یک خط وجود دارد که می‌تواند نقاط با لیبل مثبت و منفی را جدا کند. به عنوان مثال به شکل زیر توجه کنید. (البته قاعدتا لزومی ندارد خط جداکننده، مبدا گذار باشد)



بنابراین مساله‌ی یادگیری مان پیدا کردن این خط با استفاده از داده‌هایی که در اختیار داریم می‌باشد. لیبل هر داده‌را به‌صورت زیر می‌سنجیم::

$$z = \text{sign}(ax + by + c)$$

پس باید سه پارامتر a, b, c را محاسبه کنیم. برای اینکار فرض کنید تعریف کنیم $w = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ و به همین منوال هر داده را نیز به‌صورت $u_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$ تعریف می‌کنیم. با این نمادگذاری داریم:

$$z(u) = \text{sign}(< w, u >)$$

تابع هزینه‌ای که برای این مساله درنظر می‌گیریم به‌صورت زیر می‌باشد:

$$J(w) = \frac{1}{2}||w||^2 + G[\frac{1}{N}\sum_i \max(0, 1 - z_i * (< w, u_i >)]$$

که G یک ثابت مثبت مشخص است. گرادیان این تابع نیز به‌صورت زیر محاسب می‌شود.

$$\nabla J(w) = \frac{1}{n}\sum_i \begin{cases} w, & \text{if } \max\,(0, 1 - z_i(< w, u_i >)) = 0 \\ w - G z_i u_i, & \text{otherwise} \end{cases}$$

که n تعداد کل داده‌ها است.

دو دیتاست در اختیارتان قرار داده شده که اولین دیتاست شامل ۳۰۰ داده است و برای مرحله‌ی یادگیری استفاده می‌شود.

دیتاست دوم شامل ۱۰۰ داده‌است و برای مرحله‌ی ارزیابی استفاده می‌شود. با اجرای cell زیر این دو دیتاست را در قالب دیتافریم pandas لود کنید. فعلا با دیتاست train کار داریم.

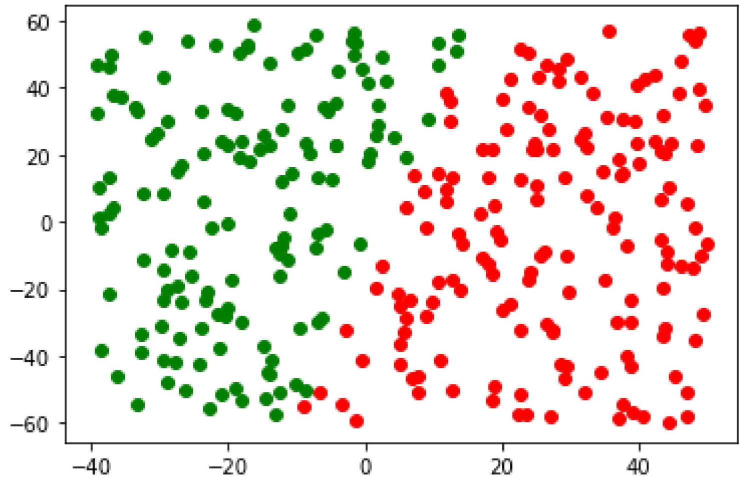
In [8]:

```
train_dataset = pd.read_csv('./train.csv')
test_dataset = pd.read_csv('./test.csv')
```

الف) نموداری رسم کنید که شامل نقطه‌های دیتاست train باشد و رنگ هرنقطه نشان‌دهنده‌ی لیبل آن باشد.

In [9]:

```
train_df = pd.DataFrame(train_dataset)
reds = train_df[train_df.z == -1]
greens = train_df[train_df.z == 1]
plt.scatter(reds.x, reds.y, color='Red')
plt.scatter(greens.x, greens.y, color='Green')
plt.show()
```



ب) تابع زیر را کامل کنید. این تابع مقدار لحظه‌ای بردار w مجموعه‌ی داده‌ی U و همچنین Z را ورودی می‌گیرید. Z

شامل آرایه‌ای از برچسب‌های داده‌ها است. U نیز به صورت آرایه‌ای به فرم $U[i] = (x_i, y_i, 1)$ ورودی داده می‌شود.

همچنین G نیز در ورودی تابع داده می‌شود.

In [10]:

```
def compute_loss(w, G, U, Z):
#     todo
    temp = 0
    for i in range(len(U)):
        inner = np.inner(w, U[i])
        temp += max(0, 1 - Z[i] * inner)
    loss = np.linalg.norm(w) ** 2 / 2 + G * temp / len(U)
    return loss
```

پ) در تابع زیر باید مرحله‌ی محاسبه‌ی گرادیان را به ازای تک داده‌ی u و z محاسبه کنید.

In [11]:

```
def compute_gradient(w, G, u, z):
#     todo
    inner = np.inner(w, u)
    condition = max(0, 1 - z * inner)
    if condition == 0:
        gradient = w
    else:
        gradient = w - G * z * u;
    return gradient
```

تابع زیر الگوریتم کاهش گرادیان را انجام می‌دهد. **(دقت کنید که می‌توانید این ۳ تابع گفته‌شده را خودتان نیز پیاده‌سازی کنید. قالب گفته‌شده صرفا پیشنهادی و برای راحتی شما است.)**

In [12]:

```
def GD(U, Z):
    STEPS = 10000
    G = 10000
    LEARNING_STEP = 0.000001
    losses = []
    w = [0.0, 0.0, 0.0]
    for i in tqdm.tqdm(range(STEPS)):
        gradient = np.array([0.0, 0.0, 0.0])
        for j in range(len(x)):
            gradient += compute_gradient(w, G, U[j], Z[j])
        losses.append(compute_loss(w, G, U, Z))
        w -= (LEARNING_STEP / len(U)) * gradient

    print("CALCULATED WEIGHTS: ", w)
    return w, losses

vectors = train_df[['x', 'y']].values.tolist()
U = [[vector[0], vector[1], 1] for vector in vectors]
Z = [vector[0] for vector in train_df[['z']].values.tolist()]
Z, U = np.array(Z), np.array(U)
W, L = GD(U, Z)
```

```
100%|██████████| 10000/10000 [00:26<00:00, 381.69it/s]

CALCULATED WEIGHTS:  [-0.42771385  0.1038443   1.11549339]
```

ت) حال خطی که به‌دست آورده‌اید را در نموداری که در قسمت الف رسم کرده‌اید نشان دهید و شکل به‌دست آمده را تحلیل و بررسی کنید. همچنین یک آرایه losses نیز توسط تابع خروجی داده‌شده است که عضو iام آن نشان‌دهنده‌ی مقدار تابع

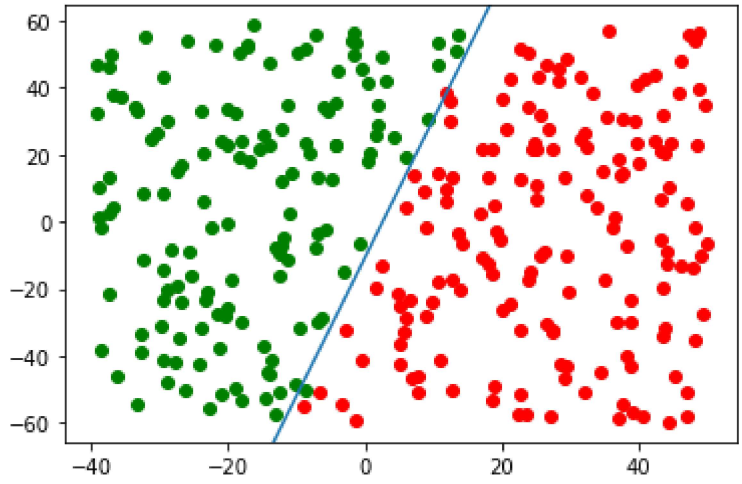
هزینه در مرحله‌ی iم است. نموداری برحسب مقدار تابع هزینه برحسب مرحله‌ی متناظر رسم کنید و بررسی کنید که آیا نمودار مطابق انتظارتان است یا خیر؟

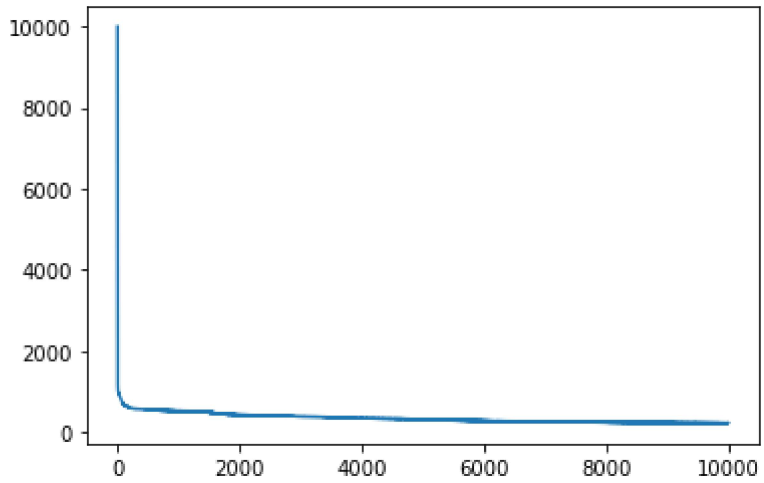
1. بررسی خط به دست آمده : همانطور که از شکل مشخص است خط به دست آمده با دقت خوبی خط جداکننده است. البته دقت کنید که کمی خطا وجود دارد. (یک نقطه سبز و یک نقطه قرمز به صورت نادرست توسط خط جدا شده اند.) ولی این خطا نیز کاملا طبیعی است چون از ابتدا فرض کردیم داده ها توسط خط جدا می شوند در حالیکه احتمالا واقعا اینگونه نبوده است.

2. بررسی نمودار هزینه بر حسب زمان : طبق نمودار در ابتدا هزینه در طی تعداد کمی مرحله کاهش شدید پیدا می کند که به دلیل اندازه زیاد گرادیان در نقاط با هزینه بیشتر(طبق تعریف) طبیعیست و هرچه جلوتر می رویم اندازه گرادیان کوچک تر می شود و در نتیجه شیب کاهش هزینه کمتر میشود ولی همچنان به دلیل صفر نشدن گرادیان کاهش رخ می دهد تا زمانی که هزینه به مینیمم خود یعنی صفر می رسد. پس نمودار کاملا با منطق پشت الگوریتم سازگار است.

In [13]:

```
plt.scatter(reds.x, reds.y, color='Red')
plt.scatter(greens.x, greens.y, color='Green')
plt.axline((1, (W[0] + W[2]) / (-1 * W[1])), (2, (2 * W[0] + W[2]) / (-1 * W[1])))
plt.show()
plt.plot(range(len(L)), L)
plt.show()
```





ث) حال برچسب داده‌های دیتاست تست را با استفاده از خطی که به‌دست آوردید محاسبه کنید و با برچسب واقعی آن‌ها مقایسه کنید. مقایسه را نیز به‌صورت عددی که از رابطه‌ی زیر به‌دست می‌آید گزارش کنید.

$$acc = \frac{1}{m}\sum_{i=1}^{m}(\hat{z}_i == z_i)$$

In [14]:

```
test_df = pd.DataFrame(test_dataset)
reds = test_df[test_df.z == -1]
greens = test_df[test_df.z == 1]
acc = 0
for point in test_df[['x', 'y', 'z']].values.tolist():
    sign = point[0] * W[0] + point[1] * W[1] + W[2]
    if sign >= 0 and point[2] == 1:
        acc += 1
    if sign <= 0 and point[2] == -1:
        acc += 1
print("ACC: ", acc / len(test_df))
```

```
ACC:  1.0
```