计算机基础与手撕代码篇！准算法工程师总结出的超强面经（含答案）

Original CV开发者都爱看的 极市平台 Yesterday

收录于话题
#CV面经

7个

作者丨灯会

来源丨极市平台

编辑丨极市平台

极市导读

作者灯会为21届中部985研究生，七月份将入职某互联网大厂cv算法工程师。在去年灰飞烟灭的算法求职季中，经过几十场不同公司以及不同部门的面试中积累出了CV总复习系列，此为计算机基础与手撕代码篇。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

系列文章：

- 深度学习三十问！一位算法工程师经历30+场CV面试后总结的常见问题合集（含答案）

- 深度学习六十问！一位算法工程师经历30+场CV面试后总结的常见问题合集下篇（含答案）
- 一位算法工程师从30+场秋招面试中总结出的超强面经—语义分割篇（含答案）
- 一位算法工程师从30+场秋招面试中总结出的超强面经——目标检测篇（含答案）
- 图像处理知多少？准大厂算法工程师30+场秋招后总结的面经问题详解
- 一位算法工程师从30+场秋招面试中总结出的超强面经—文本检测与GAN篇（含答案）
- 准算法工程师从30+场秋招中总结出的超强面经—C、Python与算法篇篇（含答案）

操作系统

1.线程和进程的区别？

进程： 是执行中一段程序，即一旦程序被载入到内存中并准备执行，它就是一个进程。进程是表示资源分配的基本概念，又是调度运行的基本单位，是系统中的并发执行的单位。

进程是程序的一次执行过程，是一个动态概念，是程序在执行过程中分配和管理资源的基本单位，每一个进程都有一个自己的地址空间，至少有 5 种基本状态，它们是：初始态，执行态，等待状态，就绪状态，终止状态。

线程：单个进程中执行中每个任务就是一个线程。线程是进程中执行运算的最小单位。

线程是CPU调度和分派的基本单位，它可与同属一个进程的其他的线程共享进程所拥有的全部资源。

一个线程只能属于一个进程，但是一个进程可以拥有多个线程。多线程处理就是允许一个进程中在同一时刻执行多个任务。

相同点：进程和线程都有ID/寄存器组、状态和优先权、信息块，创建后都可更改自己的属性，都可与父进程共享资源、都不能直接访问其他无关进程或线程的资源。

联系：线程是进程的一部分，一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。

区别： 1.一个程序至少有一个进程,一个进程至少有一个线程2. 线程的划分尺度小于进程，使得多线程程序的并发性高3. 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率4. 线程在执行过程中与进程还是有区别

的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制 5. 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

什么时候用线程什么时候进程

进程与线程的选择取决以下几点：①需要频繁创建销毁的优先使用线程；因为对进程来说创建和销毁一个进程代价是很大的。②线程的切换速度快，所以在需要大量计算，切换频繁时用线程，还有耗时的操作使用线程可提高应用程序的响应。③因为对CPU系统的效率使用上线程更占优，所以可能要发展到多机分布的用进程，多核分布用线程。④并行操作时使用线程，如C/S架构的服务器端并发线程响应用户的请求。⑤需要更稳定安全时，适合选择进程；需要速度时，选择线程更好。

2.线程有哪些状态

5种基本状态，它们是：初始态，执行态，等待状态，就绪状态，终止状态。

3.那进程间通信的方式？线程可以通信吗？

进程间通信（IPC，InterProcess Communication）是指在不同进程之间传播或交换信息。IPC的方式通常有管道（包括无名管道和命名管道）、消息队列、信号量、共享存储、Socket、Streams等。其中 Socket和Streams支持不同主机上的两个进程IPC。

线程通信常用的方式有:wait/notify 等待、Volatile 内存共享、CountDownLatch 并发工具、CyclicBarrier 并发工具

4.多线程多进程

对比维度	多进程	多线程	总结
数据共享、同步	数据共享复杂，需要用IPC；数据是分开的，同步简单	因为共享进程数据，数据共享简单， 但也是因为这个原因导致同步复杂	各有优势
内存、CPU	占用内存多，切换复杂，CPU利用率低	占用内存少，切换简单，CPU利用率低	线程占优
创建销毁、切换	创建销毁、切换复杂，速度慢	创建销毁、切换简单，速度很快	线程占优
编程、调试	编程简单，调试简单	编程复杂，调试复杂	进程占优
可靠性	进程间不会互相影响	一个线程挂掉将导致整个进程挂掉	进程占优
分布式	适应于多核、多机分布式；如果一台机器不够， 扩展到多台机器比较简单	适应于多核分布式	进程占优

5.阻塞与非阻塞

同步：所谓同步，就是在发出一个功能调用时，在没有得到结果之前，该调用就不返回。也就是必须一件一件事做,等前一件做完了才能做下一件事。

例如普通B/S模式（同步）：提交请求->等待服务器处理->处理完毕返回 这个期间客户端浏览器不能干任何事

异步：异步的概念和同步相对。当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者。

例如 ajax请求（异步）：请求通过事件触发->服务器处理（这是浏览器仍然可以作其他事情）->处理完毕

阻塞：阻塞调用是指调用结果返回之前，当前线程会被挂起（线程进入非可执行状态，在这个状态下，cpu不会给线程分配时间片，即线程暂停运行）。函数只有在得到结果之后才会返回。

有人也许会把阻塞调用和同步调用等同起来，实际上他是不同的。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回,它还会抢占cpu去执行其他逻辑，也会主动检测io是否准备好。

非阻塞：非阻塞和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。

再简单点理解就是：

1. 同步，就是我调用一个功能，该功能没有结束前，我死等结果。

2. 异步，就是我调用一个功能，不需要知道该功能结果，该功能有结果后通知我（回调通知）

3. 阻塞，就是调用我（函数），我（函数）没有接收完数据或者没有得到结果之前，我不会返回。

4. 非阻塞，就是调用我（函数），我（函数）立即返回，通过select通知调用者

6. 五种IO模型

1)阻塞I/O（blocking I/O）

2)非阻塞I/O （nonblocking I/O）

3. I/O复用(select 和poll) （I/O multiplexing）

4)信号驱动I/O （signal driven I/O (SIGIO)）

5)异步I/O （asynchronous I/O (the POSIX aio_functions)）

LINUX

1.Linux的一些常用命令

①重启reboot

②关机poweroff、shutdown -h now

③查看本机ip信息的名称ifconfig

④vi和vim编辑器

一般模式，插入模式，底行模式

一般模式(通过按iaoIAO键)-->插入模式　插入模式(按Esc键)--> 一般模式

一般模式（通过按:键）-->底行模式　底行模式(按Esc键)--> 一般模式

底行模式中，wq = write quit 写入并退出

wq! 如果有不能保存退出的情况可以使用wq!！强制退出

q! = quit !强制 不写入强制退出

⑤ 查看目录下的内容

ls = list

语法:

ls [目录名称]

实例:

ls 查看当前目录下的所有内容

ls /etc 查看etc目录下的所有内容（绝对路径）

目录下的所有文件

ls spring/ 当前目录下存在spring可以使用相对路径查看

ls spring/springmvc

-a 查看目录下所有的文件，包括隐藏文件

-l 以长格式显示目录下的所有文件（显示文件或者目录的详细信息）

ls -l 可以简化为 ll

-t 按更新时间倒叙排序显示目录下的内容

ls -a /etc

ls -l /etc

ls -l -t /etc 等同于 ls -lt /etc

⑥切换目录 cd = change directory

⑦删除文件 rm =remove

2. Linux中grep命令详解

Linux系统中grep命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。grep全称是Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

grep [options]

主要参数：grep --help可查看

-c：只输出匹配行的计数。

-i：不区分大小写。

-h：查询多文件时不显示文件名。

-l：查询多文件时只输出包含匹配字符的文件名。

-n：显示匹配行及 行号。

-s：不显示不存在或无匹配文本的错误信息。

-v：显示不包含匹配文本的所有行。

--color=auto ：可以将找到的关键词部分加上颜色的显示。

3.按时间顺序打印出文件列表，按文件大小打印文件列表

ls

按大小排序：[root@localhost ~]# ls -Sh

#按时间排序:[root@localhost ~]# ls -rt

4.Linux如何查看某进程关联的相关文件有哪些?

lsof命令是什么？可以列出被进程所打开的文件的信息。

5.Linux启动的过程

Linux系统的启动过程并不是大家想象中的那么复杂，其过程可以分为5个阶段：

内核的引导。运行 init。系统初始化。建立终端 。用户登录系统。linux如何查看进程

6.linux查看线程用哪个命令

1.使用top命令，具体用法是 top –H 加上这个选项，top的每一行就不是显示一个进程，而是一个线程。

2.使用ps命令，具体用法是 ps –xH

这样可以查看所有存在的线程，也可以使用grep作进一步的过滤。

3.使用ps命令，具体用法是 ps -mq PID 这样可以看到指定的进程产生的线程数目。

手撕代码篇

1.计算卷积网络输出尺寸

卷积神经网络的计算公式为：N=(W-F+2P)/S+1 其中N：输出大小

W：输入大小 F：卷积核大小 P：填充值的大小 S：步长大小

2. NMS

```
import numpy as np
def py_cpu_nms(dets, thresh):
    """Pure Python NMS baseline."""
    x1 = dets[:, 0]
    y1 = dets[:, 1]
    x2 = dets[:, 2]
    y2 = dets[:, 3]
    scores = dets[:, 4]
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    order = scores.argsort()[::-1]  #[::-1]表示降序排序，输出为其对应序号
    keep = []                       #需要保留的bounding box
    while order.size > 0:
        i = order[0]                #取置信度最大的（即第一个）框
        keep.append(i)              #将其作为保留的框
```

```
        #以下计算置信度最大的框（order[0]）与其它所有的框（order[1:]，即第二到最后一个）框的IOU，以下都是以向量形式表示和计算
        xx1 = np.maximum(x1[i], x1[order[1:]]) #计算xmin的max,即overlap的xmin
        yy1 = np.maximum(y1[i], y1[order[1:]]) #计算ymin的max,即overlap的ymin
        xx2 = np.minimum(x2[i], x2[order[1:]]) #计算xmax的min,即overlap的xmax
        yy2 = np.minimum(y2[i], y2[order[1:]]) #计算ymax的min,即overlap的ymax
        w = np.maximum(0.0, xx2 - xx1 + 1)      #计算overlap的width
        h = np.maximum(0.0, yy2 - yy1 + 1)      #计算overlap的hight
        inter = w * h                           #计算overlap的面积
        ovr = inter / (areas[i] + areas[order[1:]] - inter) #计算并，-inter是因为交集部分加了两次。
        inds = np.where(ovr <= thresh)[0]#本轮，order仅保留IOU不大于阈值的下标
        order = order[inds + 1]                    #删除IOU大于阈值的框
return keep
```

3.手写计算IOU代码

```
def IOU(x1,y1,X1,Y1, x2,y2,X2,Y2):
    xx = max(x1,x2)
    XX = min(X1,X2)
    yy = max(y1,y2)
    YY = min(Y1,Y2)
    m = max(0., XX-xx)
    n = max(0., YY-yy)
    Jiao = m*n
    Bing = (X1-x1)*(Y1-y1)+(X2-x2)*(Y2-y2)-Jiao
return Jiao/Bing
```

```
11
12 def bb_intersection_over_union(boxA, boxB):
13     boxA = [int(x) for x in boxA]
14     boxB = [int(x) for x in boxB]
15     xA = max(boxA[0], boxB[0])
16     yA = max(boxA[1], boxB[1])
17     xB = min(boxA[2], boxB[2])
18     yB = min(boxA[3], boxB[3])
19     interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
20     boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
21     boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
22     iou = interArea / float(boxAArea + boxBArea - interArea)
23 return iou
```

4.手撕SoftNMS

```
1  import numpy as np
2  def soft_nms(dets, sigma=0.5, Nt=0.5, method=2, threshold=0.1):
3      box_len = len(dets)   # box的个数
4      for i in range(box_len):
5          tmpx1, tmpy1, tmpx2, tmpy2, ts = dets[i, 0], dets[i, 1], dets[i, 2], dets[i, 3], dets[i, 4]
6          max_pos = i
7          max_scores = ts
8          # get max box
9          pos = i+1
10         while pos < box_len:
```

```
            if max_scores < dets[pos, 4]:
                max_scores = dets[pos, 4]
                max_pos = pos
            pos += 1
        # add max box as a detection
        dets[i, :] = dets[max_pos, :]
        # swap ith box with position of max box
        dets[max_pos, 0] = tmpx1
        dets[max_pos, 1] = tmpy1
        dets[max_pos, 2] = tmpx2
        dets[max_pos, 3] = tmpy2
        dets[max_pos, 4] = ts
        # 将置信度最高的 box 赋给临时变量
        tmpx1, tmpy1, tmpx2, tmpy2, ts = dets[i, 0], dets[i, 1], dets[i, 2], dets[i, 3], dets[i, 4]
        pos = i+1
        # NMS iterations, note that box_len changes if detection boxes fall below threshold
        while pos < box_len:
            x1, y1, x2, y2 = dets[pos, 0], dets[pos, 1], dets[pos, 2], dets[pos, 3]
            area = (x2 - x1 + 1)*(y2 - y1 + 1)
            iw = (min(tmpx2, x2) - max(tmpx1, x1) + 1)
            ih = (min(tmpy2, y2) - max(tmpy1, y1) + 1)
            if iw > 0 and ih > 0:
                overlaps = iw * ih
                ious = overlaps / ((tmpx2 - tmpx1 + 1) * (tmpy2 - tmpy1 + 1) + area - overlaps)
                if method == 1:    # 线性
                    if ious > Nt:
                        weight = 1 - ious
                    else:
```

```
                    weight = 1
                elif method == 2:  # gaussian
                    weight = np.exp(-(ious**2) / sigma)
                else:              # original NMS
                    if ious > Nt:
                        weight = 0
                    else:
                        weight = 1
                # 赋予该box新的置信度
                dets[pos, 4] = weight * dets[pos, 4]
                # 如果box得分低于阈值thresh，则通过与最后一个框交换来丢弃该框
                if dets[pos, 4] < threshold:
                    dets[pos, 0] = dets[box_len-1, 0]
                    dets[pos, 1] = dets[box_len-1, 1]
                    dets[pos, 2] = dets[box_len-1, 2]
                    dets[pos, 3] = dets[box_len-1, 3]
                    dets[pos, 4] = dets[box_len-1, 4]
                    box_len = box_len-1
                    pos = pos-1
            pos += 1
    keep = [i for i in range(box_len)]
    return keep
if __name__ == '__main__':
    dets = [[0, 0, 100, 101, 0.9], [5, 6, 90, 110, 0.7], [17, 19, 80, 120, 0.8], [10, 8, 115, 105, 0.5]]
    dets = np.array(dets)
    result = soft_nms(dets, 0.5)
    print(result)
```

5.手写k-means

```
import pandas as pd
import numpy as np
import random as ran
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d #
from sklearn.cluster import KMeans

def model_test():
    data = open_file("C:\\Users\\happy\\Desktop\\Iris1.csv")
    dataset = np.delete(data,-1,axis=1) #去掉最后一列
    k_means = KMeans(n_clusters=3) #构建模型
    k_means.fit(dataset)
    km4_labels = k_means.labels_
    ax = plt.subplot(projection='3d')
    ax.scatter(dataset[:,0],dataset[:,1],dataset[:,2],\
               c=km4_labels.astype(np.float))
    ax.set_zlabel('Z')  # 坐标轴
    ax.set_ylabel('Y')
    ax.set_xlabel('X')
    plt.show()
```

6.写python set的基本操作

集合常用的两个场景是：1.去重（如：列表去重）； 2.关系测试（如：取交集、取并集、取差集等）

7.写一个交叉熵损失函数

交叉熵损失函数：实际输出（概率）与期望输出（概率）的距离，也就是交叉熵的值越小，两个概率分布就越接近。

$$H(p,q) = -\sum_x (p(x)\log q(x) + (1-p(x))\log(1-q(x)))$$

```
def cross_entropy(a,y):
    return np.sum(np.nan_to_num(-y*np.log(a)-(1-y)*np.log(1-a)))
#tensorflow版
loss = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),reduction_indices=[1]))

#numpy版
loss = np.mean(-np.sum(y_*np.log(y),axis=1))
```

8. Softmax函数

Softmax 函数：将激活值与所有神经元的输出值联系在一起，所有神经元的激活值加起来为1。第L层（最后一层）的第j个神经元的激活输出为：

$$a_j^L = \frac{e^{Z_j^L}}{\sum_k e^{Z_t^L}}$$

```
def softmax(x):
    shift_x = x - np.max(x)#防止输入增大时输出为nan
    exp_x = np.exp(shift_x)
```

```
    return exp_x / np.sum(exp_x)
```

9.手推BN公式

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

上面的公式中m指的是mini-batch size。

源码实现

```
m = K.mean(X, axis=-1, keepdims=True)#计算均值
std = K.std(X, axis=-1, keepdims=True)#计算标准差
X_normed = (X - m)/(std + self.epsilon)#归一化
out = self.gamma * X_normed + self.beta#重构变换
```

10.Python打开一个文件，找出某个字符串最快的方法

python 字符串查找有4个方法，1 find, 2 index方法，3 rfind方法,4 rindex方法。

11.写一下混淆矩阵、精确率和召回率的公式

	预测值=1	预测值=0
真实值=1	TP	FN
真实值=0	FP	TN

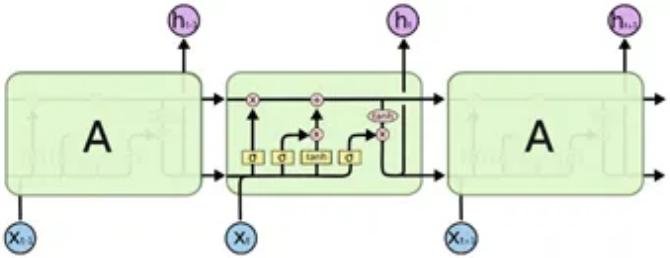
准确度(Accuracy) = (TP+TN) / (TP+TN+FN+TN)

精度(precision, 或者PPV, positive predictive value) = TP / (TP + FP)

召回(recall, 或者敏感度，sensitivity，真阳性率，TPR，True Positive Rate) = TP / (TP + FN)

F1-值(F1-score) = 2TP / (2TP+FP+FN)

12.要求画出LSTM的结构图（写了cell state，hidden state，以及门结构和信号传递过程）



参考链接

https://blog.csdn.net/u013348164/article/details/53730056

https://blog.csdn.net/A_snail/article/details/1491790

https://blog.csdn.net/weixin_42205987/article/details/82972767

https://blog.csdn.net/weixin_46539107/article/details/108847694

https://blog.csdn.net/jacke121/article/details/82756797

https://blog.csdn.net/zhangliaobet/article/details/99699675

https://blog.csdn.net/weixin_42077852/article/details/89061825

如果觉得有用，就请分享到朋友圈吧！

极市干货

YOLO教程：一文读懂YOLO V5 与 YOLO V4｜大盘点｜YOLO 系目标检测算法总览｜全面解析YOLO V4网络结构

实操教程：PyTorch vs LibTorch：网络推理速度谁更快？｜只用两行代码，我让Transformer推理加速了50倍｜PyTorch AutoGrad C++层实现

算法技巧（trick）：深度学习训练tricks总结（有实验支撑）｜深度强化学习调参Tricks合集｜长尾识别中的Tricks汇总（AAAI2021）

觉得有用麻烦给个在看啦~

收录于话题 #CV面经·7个

下一篇 · 准算法工程师从30+场秋招中总结出的超强面经—C、Python与算法篇篇（含答案）

Read more

People who liked this content also liked

只需一行代码，就能导入所有的Python库?

Python大数据分析

精心整理了10个Python自动化办公"案例"，一口吃一个，效率提高100倍！

简说Python

TIOBE 6 月编程语言排行榜：Python 有望超越 C 语言成为第一名

Python那些事